FPGA Telephony

Nandi Bugg
Kiarash Adl
Sachin Shinde

I. Abstract

With a wide array of applications ranging from telecommunications to distributed computing, networking has greatly enhanced our ability to share information and resources. Audio telephony stands out amongst all the applications as a ubiquitous innovation that requires many complex tasks to be performed synchronously and quickly for successful operation. The ability to communicate over long distances at the press of a button is often taken for granted in the digital age, and being able to provide these services for those who need them is certainly a worthwhile endeavour. For these reasons, we would like to use networking to bring audio telephony to FPGA.

II. Overview

We utilize standard networking design principles to create a system for sharing data between many Field-Programmable Gate Arrays (FPGAs) running in parallel. Specifically, we implement a secure, fast, and reliable method of packet transfer between FPGAs through a shared channel. This method is then used to transmit audio data for the application of a telephony system equipped with standard telephony features.

To facilitate independence between modules and manage complexity efficiently, we have based the network component of our system on the Open Systems Interconnection (OSI) model. This model traditionally separates the functionality of a communications system into seven layers of abstraction: in order from highest to lowest, Application, Presentation, Session, Transport, Network, Data Link Control (DLC), and Physical Interface (PHY). During the design process, we omitted three layers: Application, Presentation, and Network. For the remaining layers, we adopt the convention of assigning each of these layers to single Verilog modules, except for the bottom two layers. For each of these layers, a copy is instantiated for each transceiver on the FPGA. On top of the highest layer, the Session layer, is the User Interface (UI) layer, which handles communications with the user through the Xilinx Virtex II XC2V6000's dot display, buttons, and switches.

The Application layer became unnecessary when we reduced the number of telephony features we planned to implement, and the Presentation layer became unnecessary once the Internet Low Bitrate Codec (iLBC) was abandoned in lieu of a simpler compression scheme, which could be implemented in the Transport layer. The Network layer was dismissed in the interests of time and would have allowed us to route packets over multiple links; however, we can still transfer data between up to 32 FPGAs over a single twisted-pair wire link without it using the DLC layer.

III. Implementation

The system as a whole works to provide standard telephony features to FPGA users through a wired telecommunications network. Shown below is a sample network topology showing transceiver connections on one of the FPGAs, as well as a high-level block diagram illustrating

data flow between modules within each FPGA. Each of the OSI layer modules can be thought of as providing services to the OSI layer modules above it. For the sake of simplicity, we'll start our description of functionalities with the lowest layer module and work upwards.

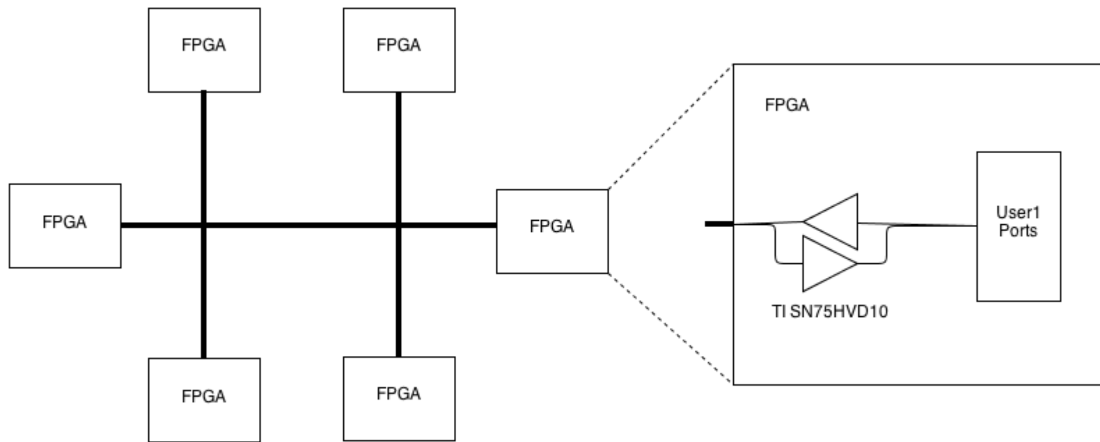

Diagram 1: Sample Network Topology

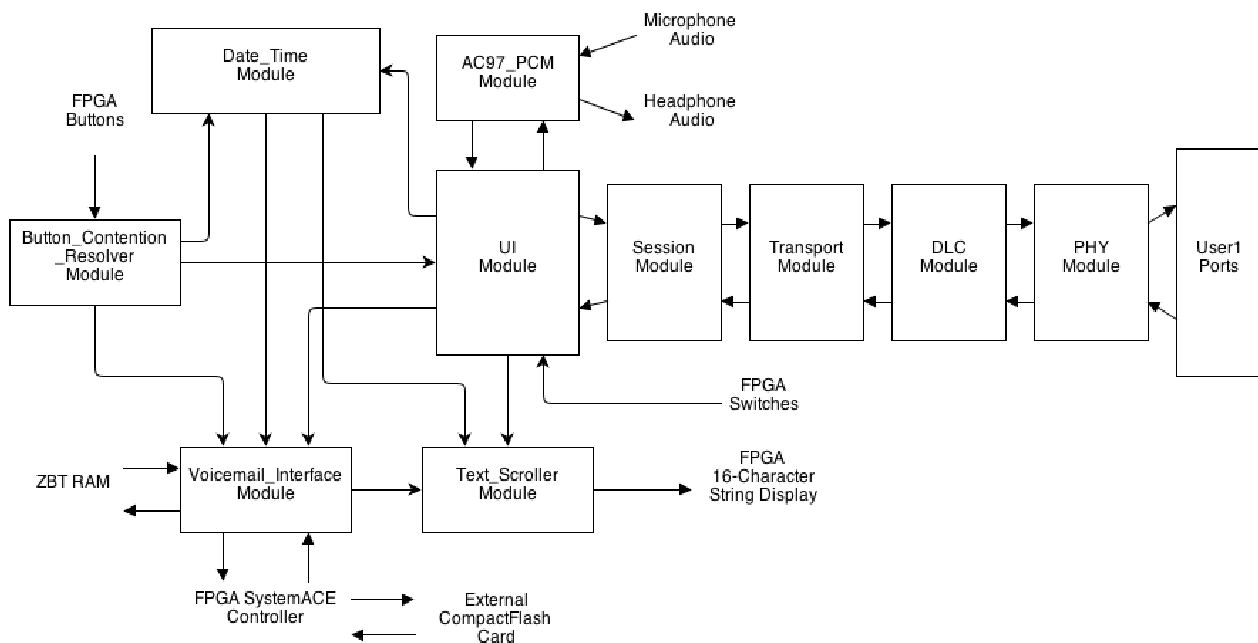

Diagram 2: High-Level Block Diagram

Physical Interface (PHY) [Shinde]

This module provides a virtual bit pipe to the DLC module above it, not immune to errors or collisions, capable of transmitting packets up to 64 bytes in length.

At the front end, the module has a full-duplex data link with the DLC module, with which it receives outgoing packets and sends incoming packets. Data inputs and outputs are one byte wide with the DLC interface. The back end of this module has a serial full-duplex link with a transceiver, which operates at a raw bit rate of 20.25 Mbps. It also controls the driver enable

(DE) bit of the transceiver, but not the receiver enable (RE) bit, as the latter is always active low so that the module may always read the physical data on the link.

A diagram of the module is shown below. It should be noted that the 8b/10b encoder and decoder are instantiated outside of the PHY module in reality since each encoder and decoder can process two streams of data, so instead the PHY_Pair module packages a single encoder, a single decoder, and two instantiations of the PHY module, in order to optimize resource allocation. They are shown in the diagram as inside the module for the sake of simplicity.

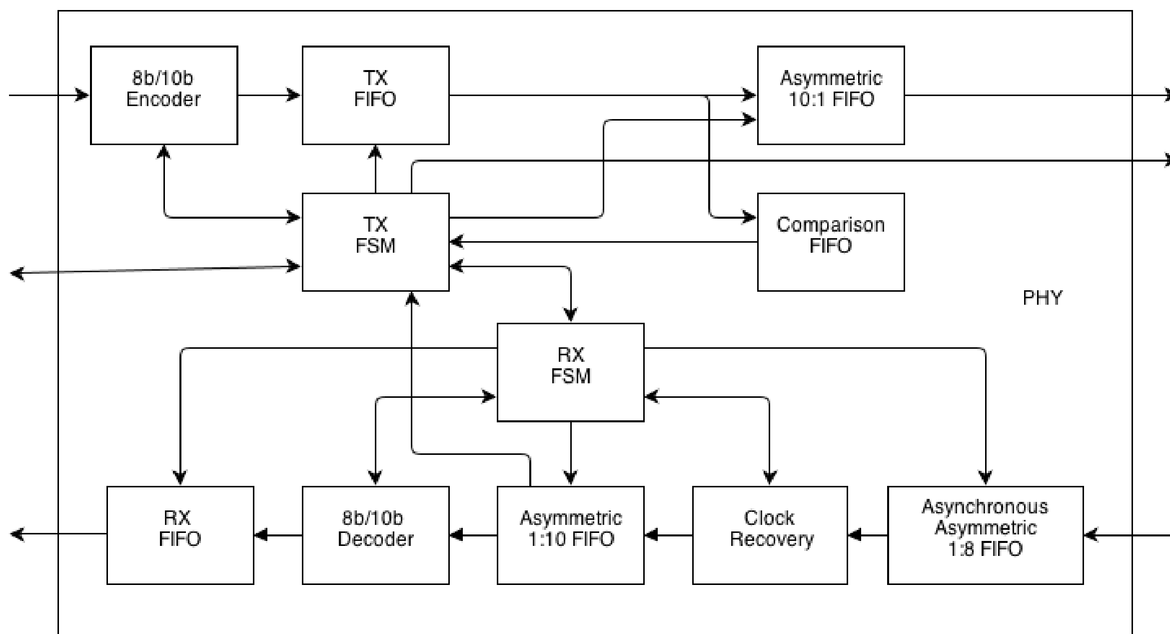

Diagram 3: PHY

When the PHY module receives data from the transceiver, it samples the data at 162 MHz and passes the line data through a series of three registers to reduce metastability substantially, and then feeds the stable data into an asynchronous asymmetric 1:8 FIFO, with an output clock of 40.5 MHz but with a read enable signal that goes high every other clock. This 40.5 MHz clock is the clock for the rest of the system up to the Session layer, after which it changes to 27 Mhz for the UI and external component interfaces (with the interfacing between the 27 MHz and 40 MHz clock domains performed with asynchronous FIFOs and/or handshaking). With the link operating at 20.25 Mbps, this effectively allows us to sample each incoming bit of the signal at a high frequency and perform operations on it at a lower frequency, so that we have 8 samples per bit. These 8 samples are then fed every other clock into a 3-byte wide shift register.

The module next performs clock recovery using the shift register. A circuit external to the FPGA drives the line to a logical one if no transceivers are driving the bus, so by detecting a run of seven consecutive ones (56 samples of ones), the module infers that the bus is idle. On the first falling edge of the bus, the module sets the location of the transition edge (modulo eight) at the location of the earliest transition from one to zero. When a new byte of data is loaded into the

shift register, it then checks if the next transition location is shifted earlier or later by one 160MHz clock by looking at the appropriate samples in the shift register, and finds the earliest location among those where a transition occurs to avoid transition glitches. If no transition occurs, then the module assumes the consecutive data bits are the same, and sets the transition modulo eight to its previous value. It also handles the edge cases where the transition location changes from 7 to 0 or 0 to 7 modulo 8, which correspond to an extra bit of data per two 40.5MHz clocks or no bit per two 40.5 MHz clocks, respectively, which allows for the data bus speeds to vary in the range of approximately 18 to 23Mbps to differences in clock frequency between FPGAs. After the clock where it sets the transition edge, it then determines whether the data was consistent by ensuring that the four samples before the location of the transition edge are the same value as that of the sample, and if it isn't a signal collision detect flag is raised. If it is the same value, then that value is fed through a 1:10 asymmetric FIFO which acts as a demultiplexer for turning a stream of serial data into 10-bit parallel data.

An FSM at the receiver determines the operations to be performed on the parallel stream. When the bus is idle, the receiver enters an idle bus state, and tells the clock recovery logic to detect the first transition from one to zero. After that first transition, the FSM then tells the clock recovery logic to resume normal operation for a signal as described previously, entering a failure state if the signal collision detection flag is raised. It checks the first 10 bits from the 1:10 asymmetric FIFO to ensure that they are 0101010101, which is provided by the transmitting FPGA for the purposes of setting the initial transition edge, and after that it receives actual data, which it sends to the IPCoreGen 8b/10b Decoder module for performing 8b/10b decoding. As the data is decoded, it is sent to an 8-bit wide, 64-byte deep FIFO. If the decoder detects an error in the running disparity of the 8b/10b decoding or it detects an error due to an used code a word, a decoder collision detection signal is raised high, and the receiver FSM enters a failure state. The decoding does not finish until a special 10-bit signal is received (one of the special K-characters of the 8b/10b encoding scheme proposed by Widmer & Franaszek, which is special in that it is guaranteed to not occur in any 10-bit subsequence of an encoded stream of bits). Once this bit is received, the packet is presumed to be completely sent, and the receiver FSM enters a success state, where it waits on a higher-order state machine to continue. This other state machine will decide whether to send the data from the FIFO (as the data received may just be from a packet that was sent by the same transmitter), and once it does it enters the failure state. Once in the failure state, the FSM waits for a run of seven consecutive ones, after which it enters the idle bus state and alerts the clock recovery logic to detect an initial transition.

The main state of the PHY module is controlled by another FSM which handles both transmitting and receiving. If the module is not transmitting and the receiver FSM detects an initial falling edge in the data stream, then the main FSM enters a receiving state, where it waits on the receiver FSM to either enter the fail state or the succeed state. If there is a success, this FSM reads out data from the 8-bit, 64-byte wide RX FIFO and sends it to the layer above it on a byte parallel bus with a ready signal, and tells the receiver FSM to proceed to the failure state. After which it enter the idle state. If the receiving FSM does not succeed, the main FSM does nothing with the data in the RX FIFO, and it is instead reset. Receiver failures are transparent to

the layers above the PHY, and once the receiver FSM returns to the idle state, the main state FSM sets the idle bus flag high for the DLC layer module above it.

If the bus is idle and a send command is received (by the main FSM by the use of a ready signal along with transmission data), then the main FSM enters a transmit state, and passes the incoming data through the IPCoreGen 8b/10b encoder module before placing the data into a TX FIFO. The data from this TX FIFO is read into an asymmetric 10:1 FIFO (implemented with a shift register with the initial value 0101010101) whenever the TX FIFO is not empty and the 10:1 FIFO is about to be empty, and this 10:1 FIFO outputs data to the transceiver for transmission while also driving the DE bit of the transceiver active high as needed.

The main FSM monitors the signal collision detection from the clock recovery. It also feeds on each read of the TX FIFO the data from that FIFO into a small 3-byte comparison FIFO. This FIFO is read whenever the Asymmetric 1:10 FIFO has data to send to the decoder, and the data is compared to ensure that the data received is the same as the data sent to detect a comparison collision detection. On the detection of a collision from one of the signals, the main state FSM enters a failure state and raises a collision detection flag for the DLC module above it. If the transmission is a success, which is evident when both the comparison and TX FIFOs are empty, then the main FSM notifies the higher layer of the success, and proceeds back to its idle state, setting the idle bus signal high when the receiver FSM indicates the bus is idle.

The troublesome part of this module was that the SN75HVD10 transceivers had low output resistance, so when both drivers tried to drive the line to different values, the voltage gap appeared across the line and it was difficult for the transceivers to detect the collision using the voltages at their locations. What was done to alleviate this problem was to force the receiving FSM to drive the bus low if it was about to enter the failure state without a success, and keep forcing it low until a run of seven zeros appeared on the line (56 samples of zeros). This results in other receivers on the line also driving the signal low in an avalanche effect, until the receiver at the transmitter also detects the collision and drives the signal low. After the bus is driven low for 56 samples, then the receivers enter the failure state, and wait for the external circuit to drive the bus high again, after which the receiver FSM indicates that the bus is idle. This lowers throughput in general (since seven or more bits are wasted driving zeros across the line), but offers better collision detection to increase throughput when multiple transceivers are trying to transmit simultaneously.

These modules were tested by sending 64-byte packets between FPGAs and verifying that data was received correctly. Proper collision detection was also verified by attempting to send continuously on both ends of the PHY, and ensuring that the collision detection flag was raised when needed through the TLA5202 Logic Analyzer.

Data Link Control (DLC) [Shinde]

This module provides a point-to-point virtual link for reliable transmission of error-free packets (in order) for the Session module above it. In particular, it allows for the transmission of up to 57

byte packets, with an 8-bit phone number as input for packet transmission and and output for reception. Data inputs and outputs are one byte wide.

At the front end, the DLC module has a full-duplex data link with the Session module, with which it receives outgoing packets and sends error-free incoming packets. The back end of the DLC module is as described in the previous section. A diagram for the DLC module is shown below.

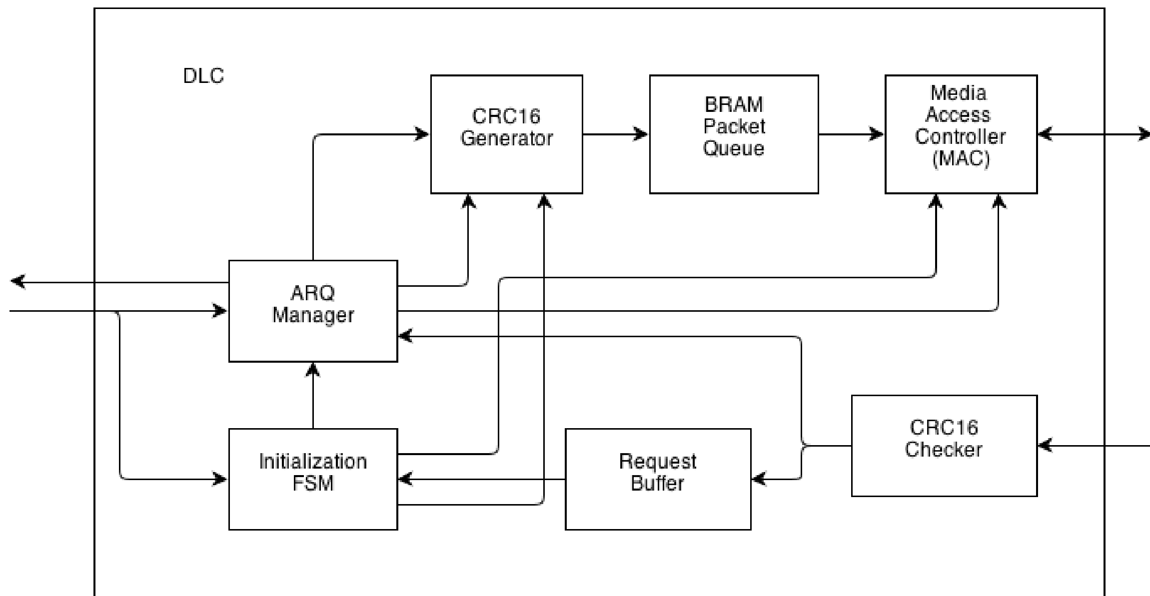

Diagram 4: DLC

The Media Access Control (MAC) layer of the DLC is managed by the MAC module. It sets a status signal to tell the module that wishes to send when it can send. It starts out in the idle state. When it detects that the bus idle and it sets its status to ready and waits for a send command, accompanied by an address in BRAM which gives the location of the packet in the packet queue. Once it receives this command and address, it waits a random number of clock cycles (with the largest possible number of bits in this wait time being the size of the contention window), and sends a command to the PHY layer to send the packet while reading out the packet data from the packet queue. It then waits for either a success, a collision detection, or an idle bus to occur from the PHY. If there is a success, it returns to the idle state while indicating it is ready to send another packet, and decreases the contention window by one unless it hits its parameterized minimum value. If there is a failure or an idle bus signal (which the MAC assumes to be a failure if it occurs before the transmission success signal goes high), then the MAC increases the contention window by one unless it surpasses its parameterized maximum value, and again waits a random number of bits limited by the size of the contention window until attempting to send again. This method is Carrier Sense Multiple Access with Collision Detection (CSMA/CD) with truncated binary exponential backoff, and works fairly well for managing collisions.

The Cyclic Redundancy Check (CRC) generator and checker work to generate the CRC16-ANSI for an outgoing packet and check the CRC16-ANSI for an incoming packet, respectively.

They do so byte-by-byte, using a parallel-update method that updates as each byte is read out from BRAM [1]. The main differences between the two modules is that the checker has a BRAM packet queue for incoming packets inside it while the generator does not, and that the checker updates its sixteen checksum registers with the checksum it receives after updating with the packet payload, and checks whether those sixteen registers are zero afterwards to ensure the packet has no errors that can be detected with a CRC16 (the CRC16 generator updates only with the packet payload, and takes the results in the checksum registers as the CRC16). The generator takes as input the packet payload and a location in the BRAM packet queue to store the packet. After the generator generates a CRC16, it appends it to the packet and stores the packet in the BRAM queue. The checker takes as input data received from the PHY layer, and outputs packet payload only if there is no CRC16 error with a ready signal. Together, the CRC checkers and the MAC provide a way to send and receive error-free packets without collision, but do not offer reliability. That is where the rest of the DLC module comes into play.

After power-on reset, the first thing the DLC module does is start the initialization FSM, which has the job of initializing the link. It first waits on a command from the higher layer to set the phone number, after which it waits for a command to initialize or an initialization packet to arrive from the CRC checker through the request buffer. The difference between the two determines which one of the DLCs amongst the nodes on the link is the master (the rest of which are then slaves). Sending an initialization command to the DLC causes the DLC to attempt to be the master and initialize the link by sending an INIT request continually to all of the other nodes on the link, which is a three-way handshake: once a DLC receives an INIT request, it automatically enters the slave state, and sends back an INIT_ACK, after which the master sends back an INIT_ACK_ACK upon reception.

Each INIT request contains the sender's phone number, so if two nodes attempt to initialize the same time, they will each see each others INIT requests and phone numbers. They then compare phone numbers, and the DLC with the higher phone number becomes the slave, while the DLC with the lower phone number stays the master. If a master sends out an INIT but receives no ACK, it assumes after a timeout period that the link is down. If it receives at least one ACK, after the reception of each ACK it will start counting down. If the countdown timer reaches zero, it assumes all links have ACKed, and sends a final FINISH_INIT to all the nodes on the links, which upon reception by the slaves, will result in them sending back FINISH_INIT_ACK to the master. Once the slaves receives the FINISH_INIT, they declare their initialization sequences finished, while for the master this is after it receives a FINISH_INIT_ACK. It should be noted that if a slave that receives an INIT_ACK_ACK receives an INIT afterwards, it goes back to waiting for an INIT_ACK_ACK while sending back ACK continually using an ACK timeout period. This is to account for the case when a master with a lower address attempts to initialize later than a master with a higher address.

This initialization procedure has been tested in real-time using the TLA5202 Logic Analyzer, and succeeds in initializing the nodes on a link with high probability. It has also been tested in the case that two nodes on the link attempt to be master at the same time, and has been shown to work successfully with high probability.

After an initialization sequence has been finished, the initialization FSM sends a message to the ARQ manager to start the parallelized Stop-and-Wait ARQ (SWARQ). We have opted to use SWARQ instead of Selective Repeat Automatic Repeat Request (SRARQ) because the round-trip time of a packet is relatively small on a single wire and the memory required to do so is large, so the benefit of having multiple outstanding packets between two specific addresses is small. However, to implement SWARQ with multipoint links, we either have to update every single nodes parity bit on each packet transmission, or keep track of the parity bit for each sender-receiver pair. We have chosen the latter, because the former is very hard to coordinate and doesn't scale too well (it is wasteful to require the retransmission of data for a node that doesn't need it). Hence, we instead implement a parallelized SWARQ in the ARQ_Manager module. This consists of four state machines running in parallel, two FIFOs, a BRAM, and a memory array.

The first FIFO (FIFO 1) keeps track of deleted spaces in the outgoing packet queue, which needs to be read from to write to the packet queue and written to in order to delete from the packet queue. The second FIFO (FIFO 2) keeps a list of outgoing packets to send, by taking as input the position in the outgoing packet queue. The BRAM keeps track of timeouts for the ARQ protocol to attempt resending the data, as well as positions in the packet queue and whether there is an outstanding packet for a particular receiver. The memory array keeps track of the receiver and transmitter parity bits for the SWARQ, with the address as the receiver. All of these memory structures (with the exception of the parity memory array) have locks on them to ensure that only one state machine may access them at any given time. A lock is also placed on the CRC generator input, as multiple state machines may wish to send packets.

The first state machine is the front end FSM, which is responsible for handling data transmission from higher layers. Upon receiving a send command in the idle state, it checks to see if there isn't already an outgoing packet by reading BRAM, and makes sure that FIFO 1 isn't empty. If either of these conditions aren't satisfied, then the FSM sends back a status of STS_TX_REJECT. If both conditions are satisfied, after acquiring a lock on the CRC generator and reading an empty packet queue number from FIFO 1, it sends back a status of STS_TX_ACCEPT, and expects data to be transmitted on the next clock cycle. It sends this data to the CRC generator, along with the appropriate header, sender, receiver, transmitter parity bit, and packet queue index. This lock is the same lock for FIFO 2, which the FSM then writes the packet queue index to in order to indicate it wishes to send the packet. It also appends a 0 to the packet queue index before writing to FIFO 2 to indicate it does not wish to delete the packet after sending. After these operations, it then returns to the idle state.

The second state machine is the back end FSM, which is responsible for handling received packets from the CRC checker. If the packet consists of data, it checks the packet to make sure it was meant for itself, and returns to idle if it isn't. If it is, it records the sender of the data packet, and then checks the parity receiver bit to ensure that the data is valid and not data that was retransmitted from earlier. If it is, then it ignores the data, and transmits an ACK. If it isn't, then it updates the parity bit for that receiver, sends the data through to the higher layer, and

transmits an ACK. The transmission of the ACK entails writing the ACK to the CRC generator, which consists of the appropriate header, the sender, the receiver, the new receiver parity bit, and the packet queue index that was determined from the received data packet. It then writes the packet queue index of that ACK to FIFO 2, appending a 1 to the index before writing to FIFO 2 to indicate that it does wish to delete the packet after sending, and updates the BRAM to indicate that an outstanding packet exists for that receiver as well as the packet queue index for that receiver and a delete timeout parameter. After these operations, it returns to the idle state.

If the packet that the back end FSM receives consists of an ACK, then the FSM verifies that the ACK is meant for itself, and if it isn't, it returns to the idle state. If it is, then it records the sender of the ACK, and then looks at the parity bit received and compares it to the transmitter parity bit. If they are not different, it ignores the ACK and returns to idle. If they are different, then this means the packet transmitted successfully, and the DLC may proceed to delete the packet from the queue, while also updating its transmitter parity bit. After doing the latter, it takes the packet queue index it gets back in the ACK packet and writes it to FIFO 1, indicating that the packet queue index should be reused. It also updates the BRAM to indicate that there is no outstanding packet.

The third FSM is the sender FSM, which is responsible for reading from FIFO 2 and sending commands to the MAC when it indicates it is ready. In the idle state, it waits for the MAC to be ready and for FIFO 2 to not be empty. Once those conditions are satisfied, it reads from FIFO 2 and sends the appropriate write command to the MAC. Once the command completes and the MAC returns to idle, it then proceeds to check if it must delete the packet. If it does, it writes the packet queue index to FIFO 1 (it doesn't need to make updates to BRAM since only ACKs are deleted through this method). It then returns to the idle state.

The fourth and final FSM is the counter FSM, which handles ARQ timeout retransmissions and deletions. It goes through each of the 256 address in BRAM one by one, and checks if an outstanding packet exists. If it does, it decreases its deletion counter by one. If a lower-order subset of those bits is zero, it retransmits the packet. If the deletion counter reaches zero, then it deletes the packet from the packet queue, and sets the BRAM so that the outstanding packet bit is null. For packet retransmission, it writes the packet queue index stored in the BRAM to FIFO 2, with a 0 appended to it to indicate that the packet should not be deleted after transmission. For packet deletion, it writes the packet queue index stored in the BRAM to FIFO 1.

These four FSMs working in parallel implement a parallelized SWARQ protocol which works fairly well. Unfortunately, not much testing has been performed in the case of more than two transceivers on the same line. However, for the case of two transceivers, voice data could be sent over the link with virtually no packet loss and virtually no lag, as was demonstrated in the project video. We take this to be a sufficient demonstration that it functions correctly, but if time had allowed, we would have tested multiple transceivers tapped to the same line to verify further functionality. The functionality of the CRC generator and checker, as well as the MAC, were tested with behavioral simulations as well.

Transport module (TransportSend and TransportReceive) [Kiarash]
This module can be thought to provide a virtual link for end-to-end transmission of error-free messages for the Session module above it. It is implemented in two pieces: TransportSend and TransportReceive.

At the front end of the TransportSend module it has a data link with the Session module with which it receives messages to be assembled into packets. It also has a simplex data link to receive “phone numbers” of the outgoing messages. Operations on outgoing messages will involve regrouping of data into packets as well as the addition of headers and/or footers, in accordance with the layering abstraction. This module first buffers the data from the Session module to build a packet and then it collects the ready packets into another buffer to be sent to the network when possible.

TransportRecieve does the exact opposite of the TransportSend module. It receives all the packets from the network as they are ready and buffers them. Then it disassembles them and provides their data to the Session module.

Both of the Transport modules are implemented using Finite State Machines. States of the TransportSend module includes the cases that the incoming data is audio or it is service request signals. In the first case, the outgoing packets is collection of the audio bytes and a header stating that the packet includes audio. When the incoming data is service request signals, the outgoing packet will be padded with zeros and on the other hand, TransportReceive removes the zeroes and provides the Session module the actual request. In both cases, TransportSent also put the outgoing phone number in the packet.

In the original idea the phone number of an outgoing packet was embedded as a header in the packet and that is how TransportSent works. However, in the last minutes, a helper module (TranToNet) was implemented to buffer between TransportSend module and the Network module. The job of the helper module was to buffer one packet and extract the phone number and output it so the network decides whether it can send the packet or not before receiving the packet.

A CombinedTransport module has an instance of TransportSend and an instance of TranToNet with the inputs of the original TransportSend and the outputs of the TranToNet compatible with the Network Module.

Session (combined with **Application** and **Presentation** modules) [Kiarash]

In the implementation process we decided to combine the three layers of Application, Presentation and Session into one module which is called Session.

This module provides a virtual communication session between two nodes of the network. Furthermore, It provides common network services for the User Interface (UI) module above it.

On one device, it receives the UI commands and audio data and makes decision on the status of the call. Then, it prepares request control signal or the audio data for the ransportSend module. On the other device, it provides the data received from TransportReceive to the UI, also deciding on the status of the session.

In general, the front end of this module has a full-duplex data link with the UI that takes as input service requests from the UI module and output service responses to the UI module. The format of a service request consists of a request control signal, with possible 16-bit 8 kHz audio input and a phone number. A service response consists of a response control signal, with possible 16-bit 8 kHz audio output and a phone number.

The basic service requests that the Session module can receive from the UI are such as dial a call, end a call, accept an incoming call, direct an incoming call to the voicemail, reject an incoming call.

The Session module is also tasked with the purpose of starting, maintaining, and ending a call. For instance, the Session module is responsible for building-up a session at the start of a telephone call, maintaining that session for the duration of the call, and breaking it down at the end of a telephone call.

This module, after receiving service requests and audio data from the UI, decides on the status of the connection and converts corresponding commands and/or audio data with a protocol compatible with the TransportSend module and passes them to that module.

To keep track of the status of the session, this module utilized a Finite State Machine. When it receives a “dial a number” command from the user, it goes to a “connecting” state, and send a request control signal to the TransportSend module. It waits for the response which could be “call rejected”, “call answered” or “call went to voicemail”. As a result to goes to the corresponding state. When there was a connection in progress, Session module prepares the audio data for the TrasnportSend module and also receives audio data from TransportRcv module and provides them to the UI. At any time if the user “cancels a call”, this module goes to an idle state and send a “call drop” signal to the other nodes in the session.

Testing Transport and Session Modules [Kiarash]

To test the two modules before integrating the whole system a separate module called CompleteTest was implemented. This module makes two instances of each module corresponding to two virtual nodes of the network in order to test the modules on one FPGA board. As input this module requests the User Input of both of the users and it output their status on the network two the Display on the FPGA board.

This module is implemented such that the first instance of Session (SessionOne) outputs to an instance of TransportSend, (senderOne), the ready packets from senderOne are wired to an

instance of TransportReceive for the virtual second node called ReceiveTwo, and the disassembled data go to an instance of session (SessionTwo) and so the loop completes by connecting the output of SessionTwo to another instance if TransportSend (senderTwo) and two connect these output to ReceiveOne and two wire ReceiveOne to SessionOne.

This CompleteTest module was compiled on an FPGA and it perfectly verifies the working status of these modules.

User Interface (UI) [Bugg]
The User Interface acts as a mediator between the lower layers of the system and the user. As the system's highest layer, it controls the audio and visual outputs of the system. During calls, audio from the Application layer is routed to the user. The routing is changed to the Voicemail module when a user is leaving a message or playing recorded messages. The UI then passes this audio, along with the current headphone volume set by the user to the AC97 module.

Three different modules can have control over the FPGA's ASCII 16-character string display to display text: Voicemail, Date and Time, and the UI module itself. The Voicemail module has access during navigation, playback, and recording of messages. The Date and Time module outputs to the display when the date and time are being set and when those items are displayed in the main menu. Otherwise, the UI module has control over the display and shows the current menu item or activity.

In addition to muxing the display and audio, the UI must also hand over input control to the Date and Time and Voicemail modules when appropriate. Generally the up and down buttons are used to navigate menus. The left button escapes to a higher level menu and the right button selects sub menus. The right and enter buttons are also used to select menu items. These buttons are needed by the Date and Time and Voicemail Modules. A signal called override hands over control to certain button inputs when necessary. When the user escapes these modules, control goes back to the UI.

The context-dependent menus are based in the five states of the system: Initialization, Idle, Incoming, Outgoing, and Busy. The Initialization state for the UI simply consists of a message indicating the user press the enter button to initialize the system. Once initialized, the system enters Idle mode. No call is being made, received, or ongoing in the Idle state. The first message prompts the user to initialize the system. Once initialization occurs, a welcome message is displayed. The main menu has the following options: Call Number, Set Headphone Volume, Voicemail, Display System Number, and Set Date & Time. The Voicemail menu item is hidden if a compact flash card is not detected in the FPGA, as it is the medium messages are recorded onto.

If someone wants to initiate a call with the user, the UI enters the Incoming state. This is triggered by the Application layer sending a message that there is an incoming call. The first

message indicates that there is an incoming call. The user can then navigate using the up and down buttons to either accept or reject the call. There is also the option of sending the call to voicemail immediately if the user has that function enabled.

If the user wants to initiate a call, he dials the number using the FPGA's switches in the Call Number sub-menu and pressing enter. This causes the UI to send a command to the Application layer to initiate a session between the user and the node being called. The Outgoing menu shows that the call is outgoing. The user can use the menu to end the call before the other person picks up if necessary.

When a phone call is ongoing, the UI is in the Busy state. There, the default message displayed is that there is a "Current Call". The user can navigate the menu to either set the headphone volume or end the call.

External Interfaces [Shinde]
There are several parts of the FPGA that interface with external components, such as CompactFlash, ZBT, or the dot display. These modules had to be given extra care, because they cannot be easily simulated in testbenches. They include the Text_Scroller module, the AC97_PCM module, Button_Contention_Resolver module, and the Voicemail_Interface module. I was also responsible for the Date_Time module, which was important for the correct operation of the Voicemail_Interface.

The Button_Contention_Resolver module made sure that only one button was high each cycle by checking that when button signals went high that only one went high. It took as inputs the debounced button signals. If only one did go high (indicated by bitwise anding the concatenation of button signals with one minus that concatenation and detecting whether the result was zero), it would set that button high, and wait for it to go low before attempting to check if another button went high.

The Text_Scroller module allowed for the display of scrolling text through the display_string module. On a ready signal, it stored ascii characters to be displayed in BRAM, and used a counter address to generate the 128-bit long string value to send to the display_string module that incremented according to a parameter. It also had initial and end parameters to ensure that the counter address stayed low and high for the right period of time, and kept track of the last address written to BRAM to know when to loop.

The AC97_PCM module is the same as that given to us in previous labs, with one small change. The sampling rate of the chip was manually altered to sample at 8 kHz instead of 48 kHz by setting the VRA bit with commands to the LM4550 and monitoring the slot request bit. The details for these signals and the method of changing the sampling frequency are in the data sheet for the LM4550.

The Date_Time module simply keeps track of the date and time (YY/MM/DD HH:MM:SS) through various counters, as well as displays the date and time in a format that the

Text_Scroller can take as input whenever the data and time change or whenever the display enable bit goes high (i.e. an 8-bit wide ASCII bus with a ready signal). It also allows for the setting of date and time, and takes into account leap years in the calculation of the length of days in months. This is implemented through cascaded if/else statements. The Date_Time module converted from binary to decimal using a lookup table in BRAM.

All of these modules were tested in real-time tests to ensure correct functionality, and the Date_Time and Button_Contention_Resolver modules were tested in testbenches as well.

Voicemail Interface [Shinde]

The Voicemail Interface is tasked with creating a method to store voicemail data to an external CompactFlash card as well as creating a visual interface to access it. A high-level diagram is shown below.

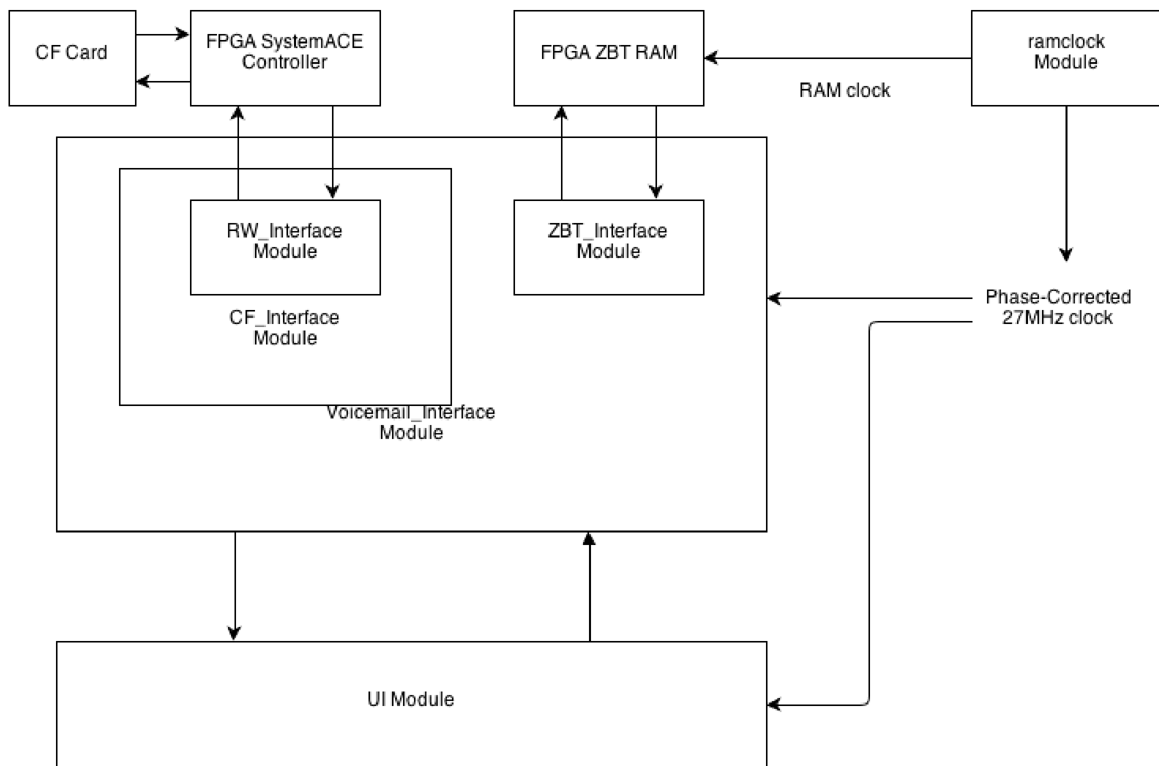

Diagram 5: Voicemail

The Voicemail_Interface module was one of the trickier interfaces to implement because it involved communications with both Compact Flash, as well as ZBT RAM. The latter is essentially performed using read and write operations to Xilinx SystemACE controller registers. The timing constraints are met using the RW_Interface module, which ensure that the signals sent to the SystemACE controller adhere to the specifications set in the controller datasheet (DS080). Through this, we can develop a system to write to and read from external CompactFlash cards. The module that does this is CF_Interface, which provides to other modules the ability to write and read up to 256 sectors (each 512 bytes) to a set of consecutive

addresses starting at some initial Logical Block Address (LBA). It also identifies the size of the CompactFlash card, and whether a CompactFlash card is inserted.

This is essentially implemented with a fairly long state machine, the details of which are in the datasheet for the Xilinx SystemACE controller CompactFlash solution, and will be glossed over in this report for the most part due to the fact that it requires reading the datasheet in full. However, for the purposes of illustrating the process, the operations for reads and writes are shown below in the following block diagrams. The system is set to operate in 16-bit word mode shortly after power-on reset. Diagram 6 shows the general process for reads. There are three subprocesses that occur within this process that are described in further detail with more diagrams. These subprocesses are indicated by the set of two overlapping rectangles, and their three diagrams follow Diagram 6. The third diagram, Diagram 9, also has a subprocess, given in Diagram 10. The write process is shown in Diagram 11, and it contains the same subprocesses as Diagram 6, except in this case we write to the buffer instead of reading from it. This subprocess is shown in Diagram 12. This has the same subprocess as reading from the buffer in Diagram 9, which is waiting for the buffer to be ready, shown in Diagram 10.

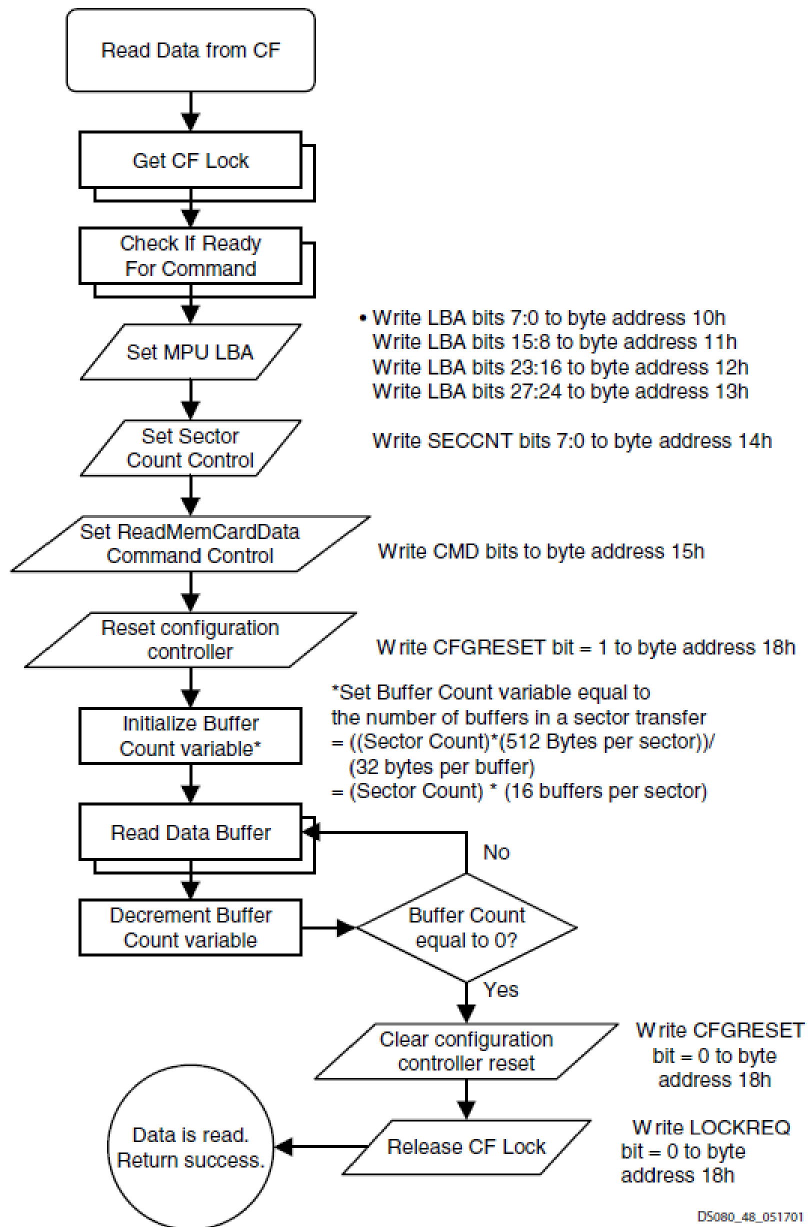

Diagram 6: Reading from CompactFlash

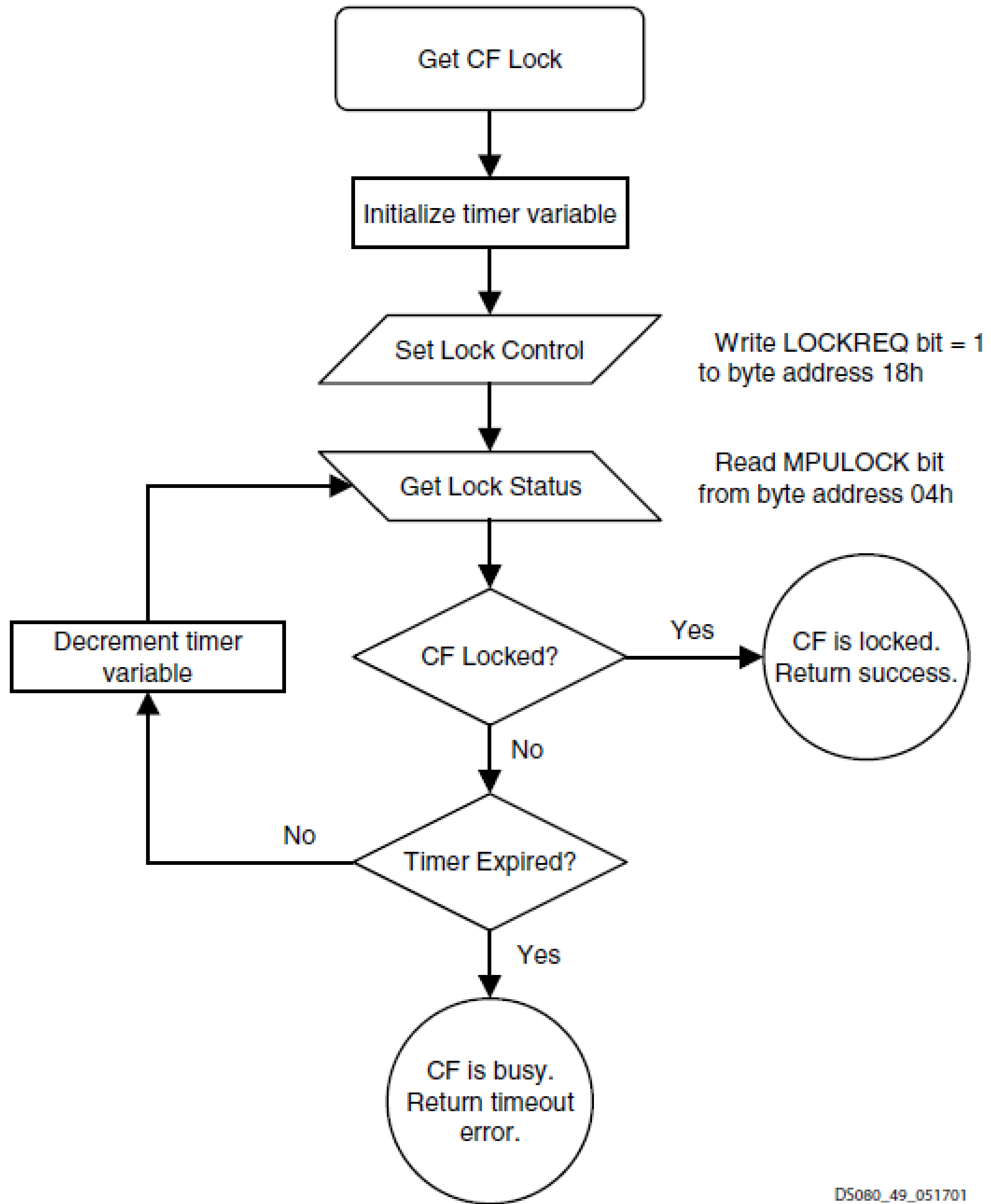

Diagram 7: Acquiring the CF Lock

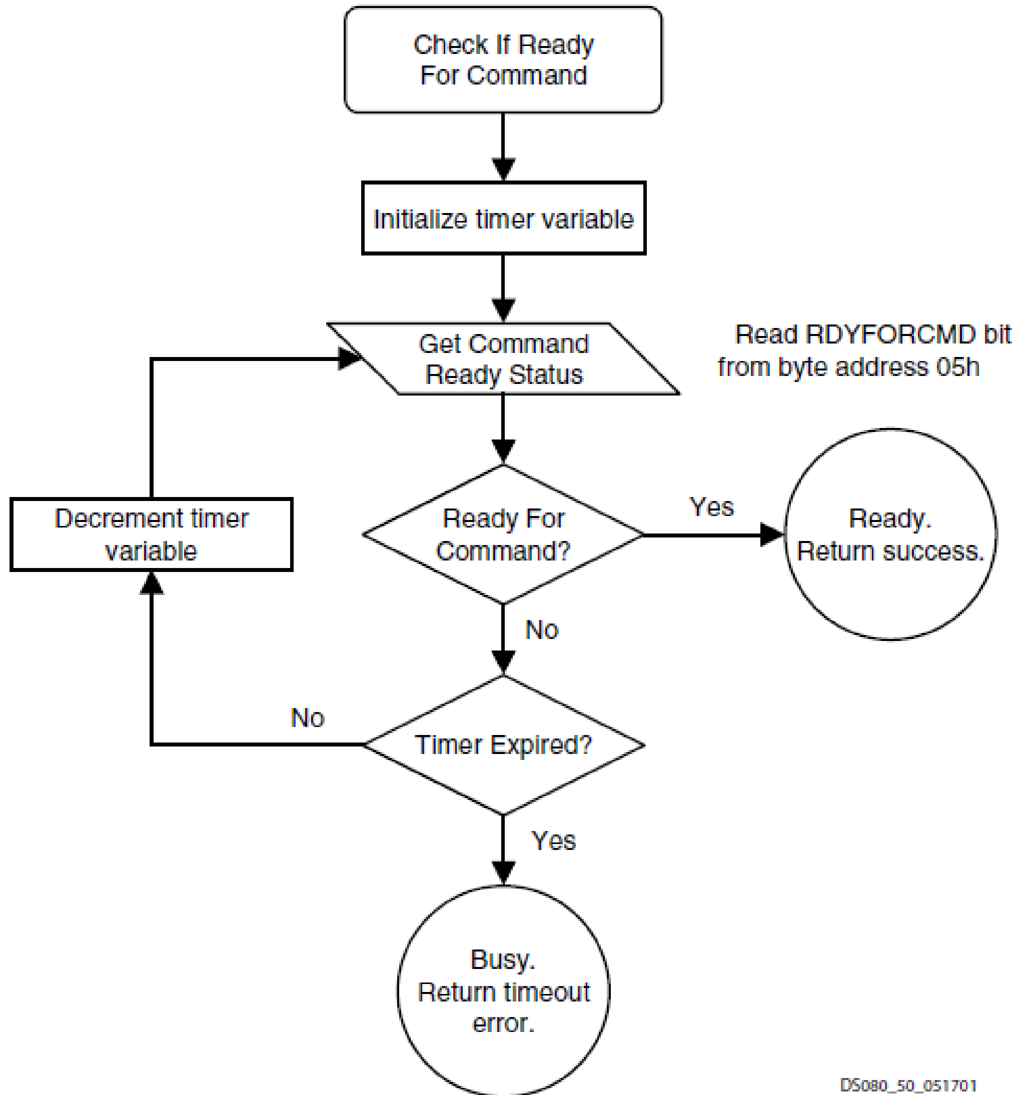

Diagram 8: Check if Ready for Command

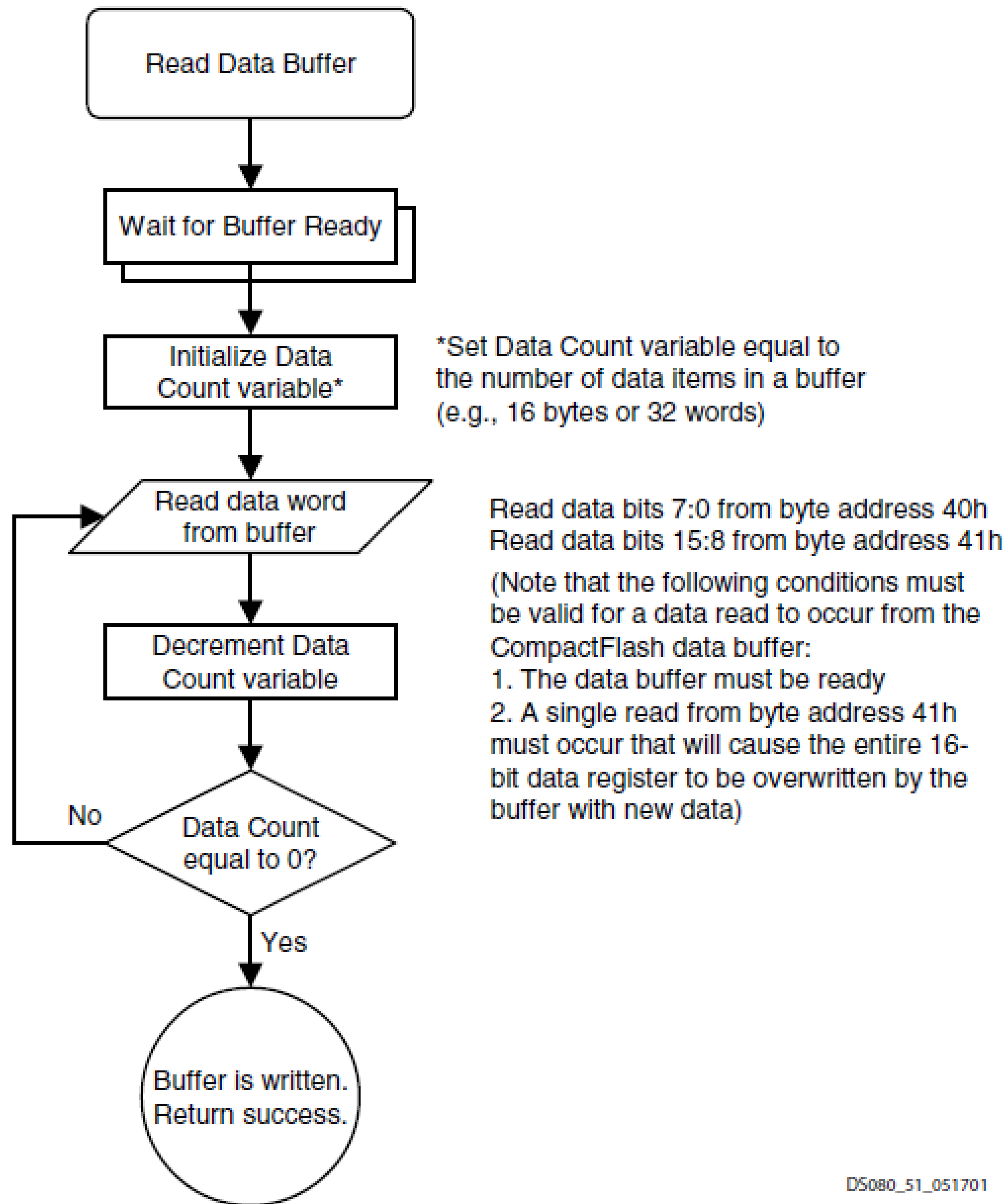

Diagram 9: Read Data Buffer

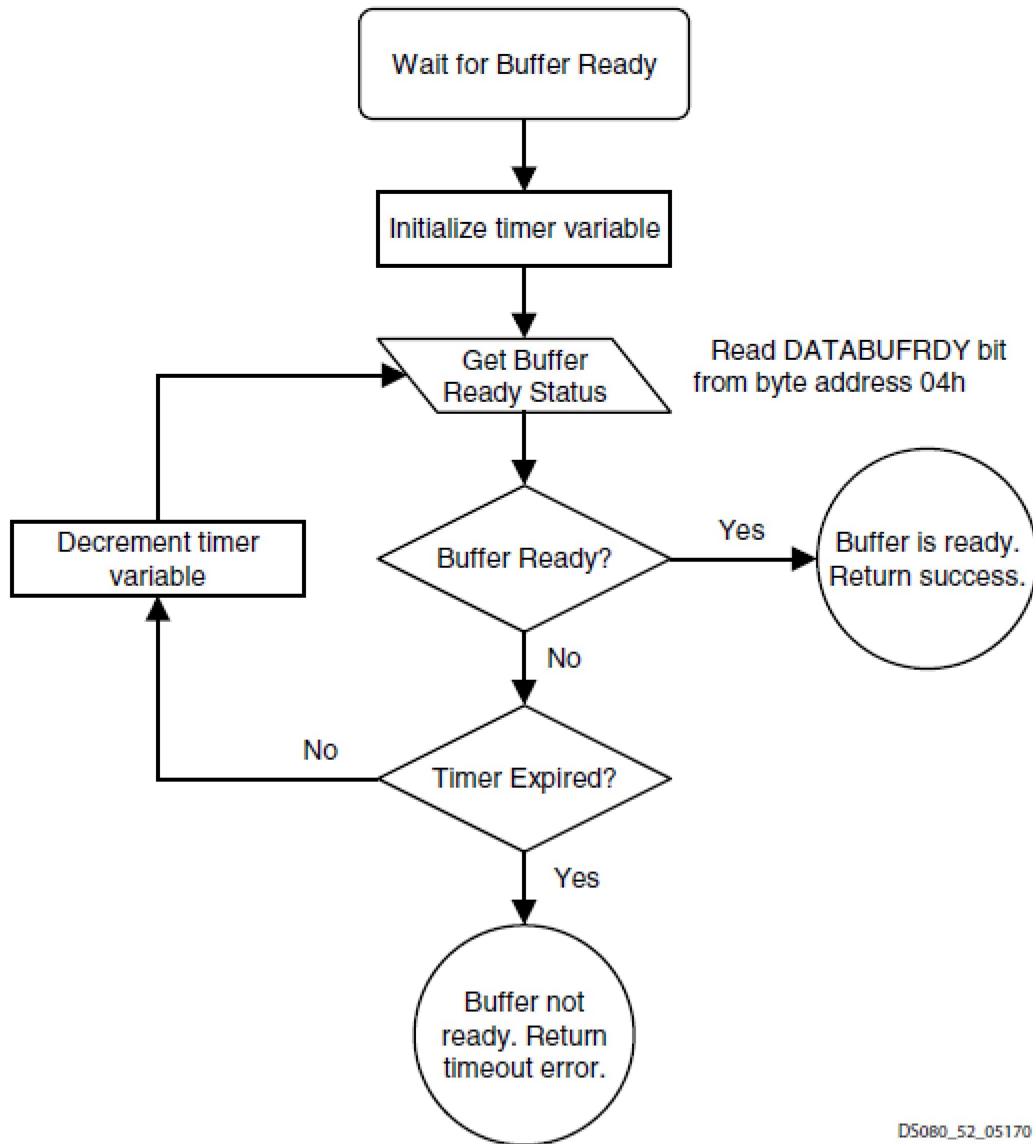

Diagram 10: Wait for Buffer Ready

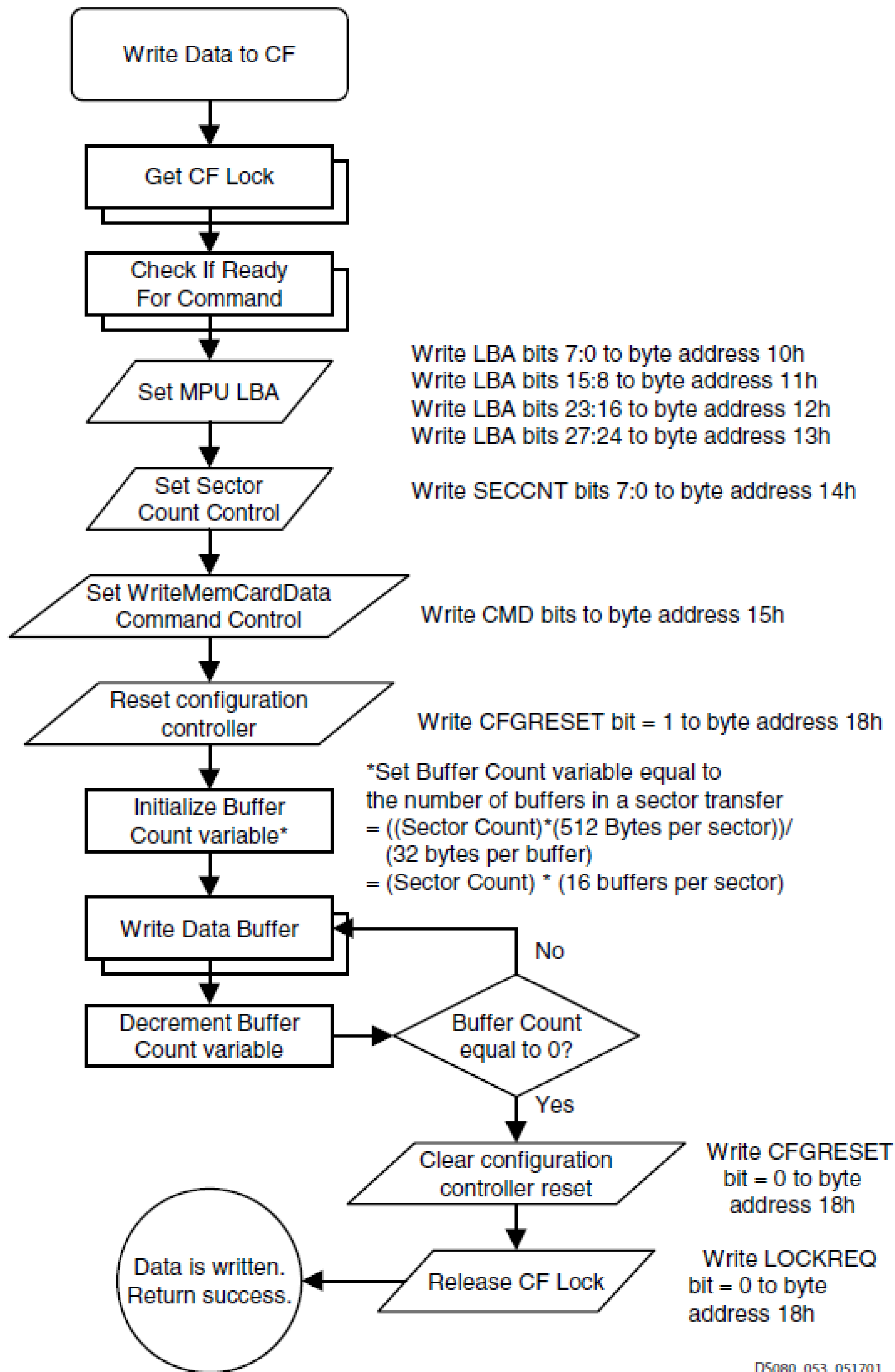

Diagram 11: Writing to CompactFlash

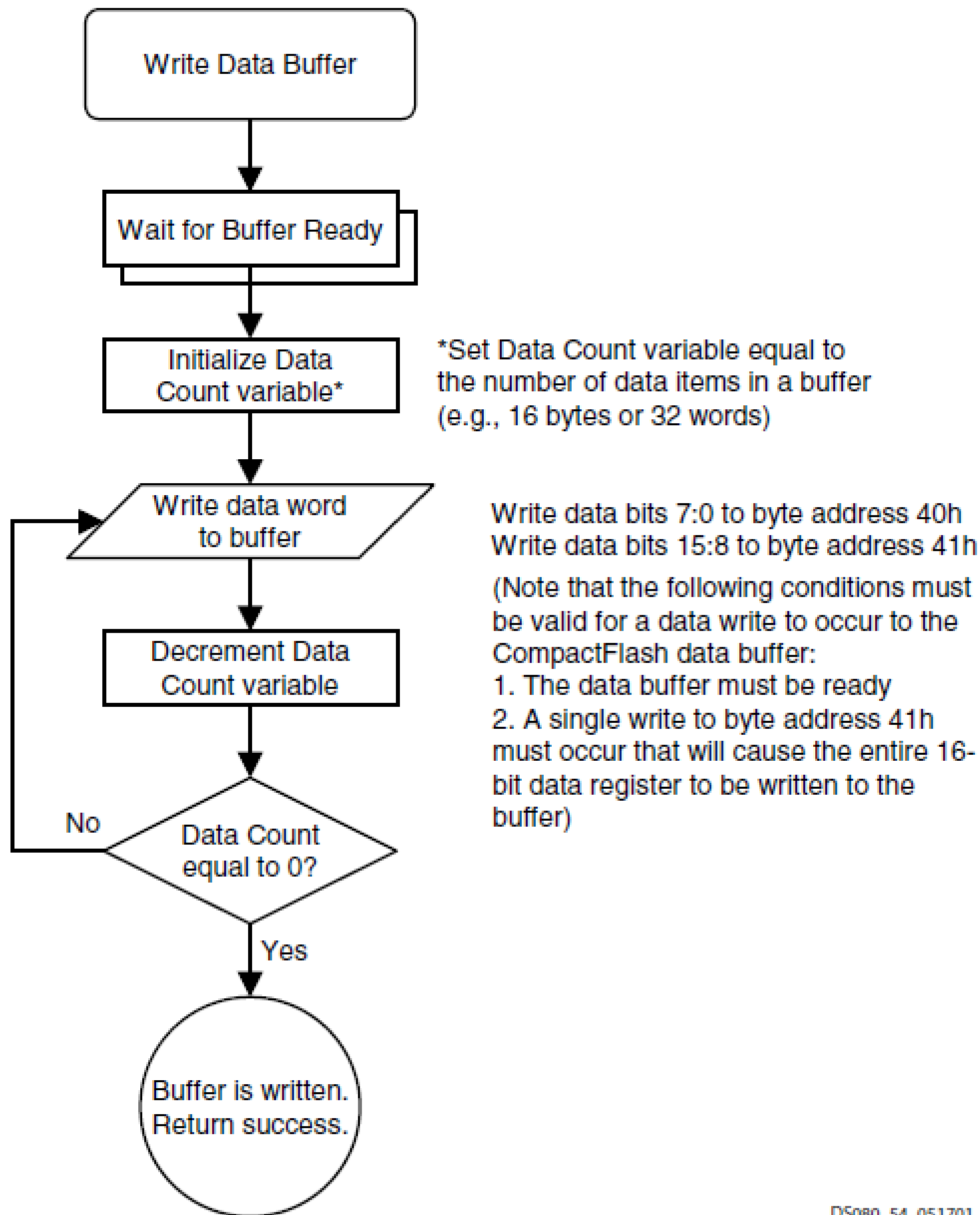

Diagram 12: Write Data Buffer

These operations describe the process of reading and writing to CompactFlash. These diagram are roughly the ones that were followed for the actual design, with the exception that we continually attempt to force and obtain the lock at power-on reset, and we don't release the lock once we obtain it.

The Voicemail_Interface module itself uses the CF_Interface module to write and read from CompactFlash, as well as ZBT_Interface to communicate with ZBT. An unusual problem we found with the ZBT was that despite using ramclock to phase-correct the 27MHz clock, the data sent from the ZBT was sent too early. To fix this process, ZBT_Interface was created to ensure reads and writes met timing constraints that would guarantee successful operation.

In particular, writes took three cycles and reads took two. During a write, the data was written for two cycles and read from the same address for the next cycle, and for a read, the data was read for two cycles with the data output latched in between the cycles the read data appeared at output. The Voicemail_Interface is, again, a fairly involved state machine, the exact details of which would be too tedious to detail in this report since there is a lot of state interleaving and optimizations to reduce the number of states. We will instead work off of somewhat higher concepts to explain the implementation.

The Voicemail_Interface module designates 8192 sectors per voicemail, which is approximately 4 minutes and 24 seconds with 8 kHZ 16-bit audio. It partitions the ZBT RAM into two equal spaces: one for two interleaved, doubly-linked lists, and the other for a large FIFO to indicate which voicemail slots (given by initial LBA) in CompactFlash have been deleted and are ready to be recycled. The two doubly-linked lists keep track of the LBAs for a list of unread and saved voicemails, and a pointer into ZBT RAM keeps track of which list element the Voicemail_Interface is currently displaying. We keep track of the beginning of the doubly-linked lists so that if another module (the User Interface module) were to send commands to the Voicemail_Interface telling it to display the start of one of the lists, it can move its pointer to the beginning of the appropriate list. We keep track of the ends of the doubly-linked lists so that when we need to add a message to either of lists we know where to insert it. When we delete a message we remove it from the appropriate list and place its ZBT address into the FIFO (which has a one-to-one correspondence with an LBA address in CompactFlash). When we save a message we delete it from the unread list and add the message to the end of the saved list. Reading and writing voicemail is done through BRAM input and output FIFOs, with the appropriate number of sectors being written depending on how many sectors are in the FIFOs. ASCII output to the Text_Scroller is generated whenever the display enable signal goes high or whenever the display (given by the ZBT pointer) changes and the display enable signal is high. The voicemail automatically cuts of writes when they reach 8192 sectors worth of CompactFlash space, and cut off reads at their end by remembering their ending sample count in ZBT.

The Voicemail_Interface was tested in conjunction with the Date_Time, Text_Scroller, Button_Contention_Resolver, and AC97_PCM in a labkit file labkit_test2.v. These were put through strenuous real-time tests, and they were demonstrated to work successfully in conjunction.

IV. External Components [Shinde]

Each physical link in the network consists of a serial twisted-pair cable, with transceivers tapping into the cable in a daisy-chain configuration to minimize driver signal reflections at the stubs. Differential signaling in the twisted-pair cable greatly attenuates the effects of electromagnetic interference, and both ends of the cable require impedance matching.

The transceiver chips are Texas Instruments SN75HVD10P DIP8 half-duplex transceivers that conform to or exceed TIA/EIA 485-A standards [2], with maximum signaling rates of 32 Mbps and open- and short-bus internal failsafe. However, the Xilinx Virtex-II XC2V6000 FPGA can

only sustain signaling rates at its physical user ports of approximately 20 Mbps without significant signal degradation from ringing. Furthermore, operating the SN75HVD10P at maximum signaling rate places a limit on cable length before timing jitter renders the signal unusable (about 40 feet). To alleviate both issues, we will operate the physical link at a signaling rate of 20 Mbps.

Because of the lack of idle-bus internal failsafe, an idle-bus external failsafe bias circuit must be used to keep the cable at a differential voltage of 250 mV when the bus is idle, detailed in [3]. This circuit requires two 549 Ω bias resistors and a 110 Ω termination resistor per line, along with a 6 V source. Since a 6 V source is not available on the Xilinx Virtex-II XC2V6000, we will create one from the 12 V source using a Texas Instruments LM317 as a voltage regulator. This idle-bus failsafe permits the detection of an idle bus in the MAC layer of the design, which allows for more sophisticated MAC protocols to increase throughput.

This idle-bus external failsafe circuit imposes a common-mode load on a driver, theoretically limiting the common-mode load from receivers to 1.183 kΩ (10 RS-485 Unit Loads, equivalent to 20 SN75HVD10P transceivers). However, this estimate is conservative, applying to a wide range of operating conditions. Tests we have performed attempting to drive a common-mode load of 375 Ω (32 RS-485 Unit Loads, equivalent to 64 SN75HVD10P transceivers) along with the common-mode load of the idle-bus failsafe circuit at 20 Mbps signaling rate at distances around 60 feet for ground potential differences of at most 7 V in magnitude show that output differential voltage still lies well outside the indeterminate range of the input differential voltage to the receiver.

V. Conclusion

We were not able to integrate all components of the project together into a single project due to time constraints. The Voicemail_Interface, Date_Time, Text_Scroller, Button_Contention_Resolver, and AC97_PCM modules were tested comprehensively in real-time, and verified to work correctly to provide a functioning system for recording voice. The PHY and DLC modules were tested in real-time, and were verified to function correctly by sending voice data between transceivers and checking if the data was received correctly through the TLA5202 Logic Analyzer as well as checking whether the quality of the voice data was acceptable. The Transport and Session modules were verified to work in testbenches and some real-time tests, but without integration a comprehensive test could not be performed. The UI module was able to reach the main menu properly in real-time tests, and could integrate with Date_Time, but other menus could not be tested since integration with other the other layer modules would be necessary to test them. Had we more time, we could have been able to consolidate our code and create a working system.

Citations

[1] Jon Buller (1996-03-15). "<31498ED0.7C0A@nortel.com> Re: 8051 and CRC-CCITT". comp.arch.embedded.

https://groups.google.com/forum/?fromgroups=#!msg/comp.arch.embedded/fvQ7yM5F6ys/3xcgqF3Kqc4J
[2] *Electrical Characteristics of Generators and Receivers for Use in Balanced Multipoint Systems*, TIA/EIA 485-A, 2003.
[3] T. Kugelstadt, “RS-485: Passive failsafe for an idle bus,” Texas Instruments Inc., Dallas, TX, Rep. SLYT324, 2009.

```
1         `timescale 1ns / 1ps
2         ////////////////////////////////////////////////////////////////////////////////
3         // Company:        6.111
4         // Engineer:       Sachin Shinde
5         //
6         // Create Date:    19:21:10 11/23/2012
7         // Design Name:
8         // Module Name:    Voicemail_Interface
9         // Project Name:
10        // Target Devices:
11        // Tool versions:
12        // Description:   Allows for reading and writing of voicemails to CompactFlash.
13        //                Voicemail max time is around 4 minutes (8192 sectors)
14        //                Max number of voicemails is 1024.
15        //
16        // Dependencies:
17        //
18        // Revision:
19        // Revision 0.01 - File Created
20        // Additional Comments:
21        //
22        ////////////////////////////////////////////////////////////////////////////////
23
24        ////////////////////////////////////////////////////////////////////////////////
25        //// High-Level Interface Module
26        ////////////////////////////////////////////////////////////////////////////////
27
28        module Voicemail_Interface(
29           input clk_27mhz,       // 27MHz clock
30           input reset,           // Synchronous reset
31           // Main Interface ports
32           output [3:0] sts,          // Status port
33           input [3:0] cmd,           // Port for issuing commands
34           input [7:0] phn_num,       // Port for phone number (on writes)
35           input [15:0] din,          // Sample Data in
36           output [15:0] dout,        // Sample Data out
37           input d_ready,             // Sample Data Ready Signal
38           input disp_en,             // Display Enable
39           // Button inputs
40           input button_up,
41           input button_down,
42           // ASCII output
43           output [7:0] ascii_out,    // Port for ASCII data
44           output ascii_out_ready,    // Ready signal for ASCII data
45           // ZBT RAM I/Os
46           inout  [35:0] ram_data,
47           output [18:0] ram_address,
48           output ram_we_b,
49           output [3:0] ram_bwe_b,
50           // Date & Time inputs
51           input [6:0] year,
52           input [3:0] month,
53           input [4:0] day,
54           input [4:0] hour,
55           input [5:0] minute,
56           input [5:0] second,
57           // Binary-to-Decimal Lookup-Table I/O
58           output [6:0] addr,
59           input [7:0] data,
60           // SystemACE ports
61           inout [15:0] systemace_data,     // SystemACE R/W data
```

```
 62          output [6:0] systemace_address,  // SystemACE R/W address
 63          output systemace_ce_b,           // SystemACE chip enable (Active Low)
 64          output systemace_we_b,           // SystemACE write enable (Active Low)
 65          output systemace_oe_b,           // SystemACE output enable (Active Low)
 66          input systemace_mpbrdy           // SystemACE buffer ready
 67          );
 68
 69          // Define command parameters
 70          parameter CMD_IDLE       = 4'd0;
 71          parameter CMD_START_RD   = 4'd1;
 72          parameter CMD_END_RD     = 4'd2;
 73          parameter CMD_START_WR   = 4'd3;
 74          parameter CMD_END_WR     = 4'd4;
 75          parameter CMD_VIEW_UNRD  = 4'd5;
 76          parameter CMD_VIEW_SAVED = 4'd6;
 77          parameter CMD_DEL        = 4'd7;
 78          parameter CMD_SAVE       = 4'd8;
 79
 80          // Instantiate status register
 81          reg [3:0] sts_reg;
 82
 83          // Assign status signal
 84          assign sts = sts_reg;
 85
 86          // Define Status parameters
 87          parameter STS_NO_CF_DEVICE = 4'd0;
 88          parameter STS_CMD_RDY      = 4'd1;
 89          parameter STS_BUSY         = 4'd2;
 90          parameter STS_RDING        = 4'd3;
 91          parameter STS_RD_FIN       = 4'd4;
 92          parameter STS_WRING        = 4'd5;
 93          parameter STS_WR_FIN       = 4'd6;
 94          parameter STS_ERR_VM_FULL  = 4'd7;
 95          parameter STS_ERR_RD_FAIL  = 4'd8;
 96          parameter STS_ERR_WR_FAIL  = 4'd9;
 97
 98          // Declare CF Interface command parameters
 99          parameter CF_CMD_IDLE   = 2'b00;
100          parameter CF_CMD_DETECT = 2'b01;
101          parameter CF_CMD_READ   = 2'b10;
102          parameter CF_CMD_WRITE  = 2'b11;
103
104          // Define ZBT Operation parameters (OPCODE)
105          parameter OP_IDLE  = 2'b00;
106          parameter OP_READ  = 2'b01;
107          parameter OP_WRITE = 2'b10;
108
109          // Instantiate state
110          reg [5:0] state;
111
112          // Define state parameters
113          parameter S_INIT                 = 6'h00;
114          parameter S_IDLE                 = 6'h01;
115          parameter S_START_RW_INIT        = 6'h02;
116          parameter S_START_RW_INIT_1      = 6'h03;
117          parameter S_START_WR             = 6'h04;
118          parameter S_START_WR_1           = 6'h05;
119          parameter S_START_WR_2           = 6'h06;
120          parameter S_START_WR_3           = 6'h07;
121          parameter S_START_WR_4           = 6'h08;
122          parameter S_WRING                = 6'h09;
```

```
123          parameter S_END_WR                = 6'h0A;
124          parameter S_END_WR_2              = 6'h0B;
125          parameter S_END_WR_3              = 6'h0C;
126          parameter S_START_RD              = 6'h0D;
127          parameter S_START_RD_1            = 6'h0E;
128          parameter S_START_RD_2            = 6'h0F;
129          parameter S_RDING                 = 6'h10;
130          parameter S_END_RD                = 6'h11;
131          parameter S_DEL_MSG               = 6'h12;
132          parameter S_DEL_MSG_1             = 6'h13;
133          parameter S_DEL_MSG_2             = 6'h14;
134          parameter S_DEL_MSG_3             = 6'h15;
135          parameter S_DEL_MSG_4             = 6'h16;
136          parameter S_DEL_MSG_5             = 6'h17;
137          parameter S_ADD_MSG               = 6'h18;
138          parameter S_ADD_MSG_1             = 6'h19;
139          parameter S_ADD_MSG_2             = 6'h1A;
140          parameter S_ADD_MSG_3             = 6'h1B;
141          parameter S_ADD_MSG_4             = 6'h1C;
142          parameter S_ADD_MSG_5             = 6'h1D;
143          parameter S_UPDT_MSG_POS_DATA     = 6'h1E;
144          parameter S_UPDT_MSG_POS_DATA_1   = 6'h1F;
145          parameter S_UPDT_MSG_POS_DATA_2   = 6'h20;
146          parameter S_UPDT_MSG_POS_DATA_3   = 6'h21;
147          parameter S_UPDT_MSG_POS_DATA_4   = 6'h22;
148          parameter S_UPDT_MSG_POS_DATA_5   = 6'h23;
149          parameter S_UPDT_MSG_POS_DATA_6   = 6'h24;
150          parameter S_UPDT_MSG_POS_DATA_7   = 6'h25;
151          parameter S_UPDT_MSG_POS_DATA_8   = 6'h26;
152          parameter S_UPDT_MSG_POS_DATA_9   = 6'h27;
153          parameter S_UPDT_MSG_POS_DATA_10  = 6'h28;
154
155          // Generate button rise signals
156          reg bl_up, bl_down;
157          always @(posedge clk_27mhz) begin
158             if (reset) begin
159                bl_up <= 0;
160                bl_down <= 0;
161             end
162             else begin
163                bl_up <= button_up;
164                bl_down <= button_down;
165             end
166          end
167          assign b_up_rise = button_up & ~bl_up;
168          assign b_down_rise = button_down & ~bl_down;
169
170          // Instantiate CF_Interface
171          wire [1:0] CF_cmd;
172          wire [27:0] CF_LBA;
173          wire [7:0] CF_SC;
174          wire [15:0] CF_din;
175          wire CF_we_req;
176          wire [15:0] CF_dout;
177          wire CF_nd;
178          wire CF_CF_detect;
179          wire [27:0] CF_LBA_max;
180          wire CF_ready;
181          CF_Interface CF_INTERFACE_1(
182             .clk_27mhz(clk_27mhz),                  // 27 MHz clock
183             .reset(reset),                          // Synchronous reset
```

```
184             // Mid-level I/O
185             .cmd(CF_cmd),                           // Command to be performed
186             .LBA(CF_LBA),                           // Logical Block Address for r/w
187             .SC(CF_SC),                             // Sector Count
188             .din(CF_din),                           // Data input for writes
189             .we_req(CF_we_req),                     // Request for write data
190             .dout(CF_dout),                         // Data output for reads
191             .nd(CF_nd),                             // New Data available at output
192             .CF_detect(CF_CF_detect),               // Detect if CF d evice is connected
193             .LBA_max(CF_LBA_max),                   // Maximum number of LBAs if CF detected
194             .ready(CF_ready),                       // Command ready signal
195             // SystemACE ports
196             .systemace_data(systemace_data),        // SystemACE R/W data
197             .systemace_address(systemace_address),  // SystemACE R/W address
198             .systemace_ce_b(systemace_ce_b),        // SystemACE chip enable (Active Low)
199             .systemace_we_b(systemace_we_b),        // SystemACE write enable (Active Low)
200             .systemace_oe_b(systemace_oe_b),        // SystemACE output enable (Active Low)
201             .systemace_mpbrdy(systemace_mpbrdy)     // SystemACE MPU buffer ready
202             );
203
204          // Instantiate CF Interface input registers
205          reg [1:0] CF_cmd_reg;
206          reg [27:0] CF_LBA_reg;
207          reg [7:0] CF_SC_reg;
208
209          // Assign CF Interface inputs
210          assign CF_cmd = CF_cmd_reg;
211          assign CF_LBA = CF_LBA_reg;
212          assign CF_SC = CF_SC_reg;
213
214          // Instantiate 4-Sector FIFO for Incoming Voicemail
215          reg wr_en_override, rst_in_FIFO;
216          wire [10:0] in_data_cnt;
217          wire in_wr_en;
218          assign in_wr_en = wr_en_override & d_ready;
219          wire in_full, in_empty;
220          Voicemail_FIFO VM_FIFO_IN(
221             .clk(clk_27mhz),
222             .rst(rst_in_FIFO),
223             .din(din),
224             .dout(CF_din),
225             .wr_en(in_wr_en),
226             .rd_en(CF_we_req),
227             .data_count(in_data_cnt),
228             .empty(in_empty),
229             .full(in_full)
230             );
231
232          // Instantiate 4-Sector FIFOs for Outgoing Voicemail
233          reg rd_en_override, rst_out_FIFO;
234          wire [10:0] out_data_cnt;
235          wire out_rd_en;
236          assign out_rd_en = rd_en_override & d_ready;
237          wire out_empty, out_full;
238          Voicemail_FIFO VM_FIFO_OUT(
239             .clk(clk_27mhz),
240             .rst(rst_out_FIFO),
241             .din(CF_dout),
242             .dout(dout),
243             .wr_en(CF_nd),
244             .rd_en(out_rd_en),
```

```
245             .data_count(out_data_cnt),
246             .empty(out_empty),
247             .full(out_full)
248             );
249
250          // Instantiate ZBT Interface
251          wire [35:0] ZBT_din, ZBT_dout;
252          wire [18:0] ZBT_addr;
253          wire [1:0] ZBT_op;
254          wire [3:0] ZBT_bwe;
255          wire ZBT_nd, ZBT_ready;
256          ZBT_Interface ZBT_INTERFACE_1(
257             .clk_27mhz(clk_27mhz),     // 27MHz clock
258             .reset(reset),             // Synchronous reset
259             // Low-Level I/O
260             .addr(ZBT_addr),           // Address signal for r/w
261             .din(ZBT_din),             // Data input for writes
262             .op(ZBT_op),               // Operation Code
263             .bwe(ZBT_bwe),             // Partial (byte) writes
264             .dout(ZBT_dout),           // Data output for reads
265             .nd(ZBT_nd),               // New data available at output
266             .ready(ZBT_ready),         // Command Ready signal
267             // ZBT ports
268             .ram_data(ram_data),
269             .ram_address(ram_address),
270             .ram_we_b(ram_we_b),
271             .ram_bwe_b(ram_bwe_b)
272             );
273
274          // Instantiate ZBT Interface registers
275          reg [18:0] ZBT_addr_reg;
276          reg [35:0] ZBT_din_reg;
277          reg [1:0] ZBT_op_reg;
278          reg [3:0] ZBT_bwe_reg;
279
280          // Assign ZBT Interface registers
281          assign ZBT_addr = ZBT_addr_reg;
282          assign ZBT_din = ZBT_din_reg;
283          assign ZBT_op = ZBT_op_reg;
284          assign ZBT_bwe = ZBT_bwe_reg;
285
286          // Instantiate ZBT doubly-linked list (DLL) pointers
287          reg [14:0] unread_start;
288          reg [14:0] unread_end;
289          reg [14:0] saved_start;
290          reg [14:0] saved_end;
291
292          // Instantiate ZBT DLL message empty regs
293          reg unread_exist;  // High if at least one unread message
294          reg saved_exist;   // High if at least one saved message
295
296          // Instantiate ZBT FIFO pointers
297          reg [14:0] FIFO_start;
298          reg [14:0] FIFO_end;
299
300          // Instantiate CF Interface operation (internal) registers
301          reg [2:0] CF_Interface_op;
302
303          // Declare CF Interface operation parameters
304          parameter CF_OP_IDLE     = 3'h0;
305          parameter CF_OP_WR_EN    = 3'h1;
```

```
306          parameter CF_OP_RD_EN    = 3'h2;
307          parameter CF_OP_WR_FORCE = 3'h3;
308          parameter CF_OP_DETECT   = 3'h4;
309
310          // Latch input parameters
311          reg [7:0] latch_phn_num;
312          reg [32:0] latch_DT;
313
314          // Instantiate other internal registers
315          reg [14:0] msg_max;                                    // Maximum number of messages
316          reg [14:0] msg_chkout_cnt;                             // Allocates message IDs initially
317          reg [21:0] sample_cnt;                                 // Counts samples on R/W (0 to
       8192*256 inclusive)
318          reg [14:0] msg_pos, msg_pos_prev, msg_pos_next;   // Current message position and data
319          reg [7:0] msg_pos_phn_num;                             // Message position data
320          reg [32:0] msg_pos_DT;                                 // Message position data
321          reg [21:0] msg_pos_sample_cnt;                         // Message position data
322          reg disp_req;                                            // Set high for display request
323          reg S_DEL_save;                                          // Parameter for delete procedure
324          reg S_ADD_MSG_unread;                                    // Parameter for add message proc
325          reg wr_op;                                               // Used to control whether RD or W
       op after Initial CF Detect
326          reg cur_view;                                            // Currently viewed list (either 0
       for unread, or 1 for saved)
327
328          // Manage main state transitions
329          always @(posedge clk_27mhz) begin
330             if (reset | ~CF_CF_detect) begin
331                // Set High-Level output registers
332                sts_reg <= STS_NO_CF_DEVICE;
333                // Set ZBT Interface output registers
334                ZBT_addr_reg <= 0;
335                ZBT_din_reg <= 0;
336                ZBT_op_reg <= OP_IDLE;
337                ZBT_bwe_reg <= 0;
338                // Set ZBT DLL pointers
339                unread_start <= 0;
340                unread_end <= 0;
341                saved_start <= 0;
342                saved_end <= 0;
343                unread_exist <= 0;
344                saved_exist <= 0;
345                // Set ZBT FIFO pointers
346                FIFO_start <= 0;
347                FIFO_end <= 0;
348                // Set override registers
349                rd_en_override <= 0;
350                wr_en_override <= 0;
351                // Set FIFO registers
352                rst_in_FIFO <= 0;
353                rst_out_FIFO <= 0;
354                // Set CF Interface op register
355                CF_Interface_op <= CF_OP_IDLE;
356                // Set Internal registers
357                msg_max <= 0;
358                msg_chkout_cnt <= 0;
359                sample_cnt <= 0;
360                msg_pos <= 0;
361                msg_pos_prev <= 0;
362                msg_pos_next <= 0;
363                msg_pos_phn_num <= 0;
```

```
364                    msg_pos_DT <= 0;
365                    msg_pos_sample_cnt <= 0;
366                    disp_req <= 0;
367                    wr_op <= 0;
368                    cur_view <= 0;
369                    // Set State
370                    state <= S_INIT;
371                 end
372                 else begin
373                    case (state)
374                       S_INIT: begin
375                          if (CF_ready) begin
376                             // Set parent and child to itself
377                             if (ZBT_ready) begin
378                                msg_max <= CF_LBA_max[27:13];
379                                sts_reg <= STS_CMD_RDY;
380                                ZBT_addr_reg <= {2'b00, 15'd0, 2'b00};
381                                ZBT_din_reg <= 36'd0;
382                                ZBT_op_reg <= OP_WRITE;
383                                ZBT_bwe_reg <= 4'b1111;
384                                state <= state + 1;  // S_IDLE
385                                disp_req <= 1;
386                             end
387                          end
388                       end
389                       S_IDLE: begin
390                          disp_req <= 0;
391                          ZBT_op_reg <= OP_IDLE;
392                          CF_Interface_op <= CF_OP_IDLE;
393                          if ((b_up_rise | b_down_rise)&(unread_exist | cur_view)&(saved_exist | ~
           cur_view)) begin
394                             sts_reg <= STS_BUSY;
395                             case (b_up_rise)
396                                1'b0: begin  // Down
397                                   state <= S_UPDT_MSG_POS_DATA;
398                                   msg_pos <= msg_pos_next;
399                                end
400                                1'b1: begin  // Up
401                                   state <= S_UPDT_MSG_POS_DATA;
402                                   msg_pos <= msg_pos_prev;
403                                end
404                             endcase
405                          end
406                          else begin
407                             case (cmd)
408                                CMD_START_RD: begin
409                                   state <= S_START_RW_INIT;
410                                   sts_reg <= STS_BUSY;
411                                   wr_op <= 0;
412                                end
413                                CMD_START_WR: begin
414                                   state <= S_START_RW_INIT;
415                                   latch_phn_num <= phn_num;
416                                   latch_DT <= {year, month, day, hour, minute, second};
417                                   sts_reg <= STS_BUSY;
418                                   wr_op <= 1;
419                                end
420                                CMD_VIEW_UNRD: begin
421                                   state <= S_UPDT_MSG_POS_DATA;
422                                   msg_pos <= unread_start;
423                                   sts_reg <= STS_BUSY;
```

```
424                                 cur_view <= 0;
425                              end
426                              CMD_VIEW_SAVED: begin
427                                 state <= S_UPDT_MSG_POS_DATA;
428                                 msg_pos <= saved_start;
429                                 sts_reg <= STS_BUSY;
430                                 cur_view <= 1;
431                              end
432                              CMD_DEL: begin
433                                 state <= S_DEL_MSG;
434                                 S_DEL_save <= 0;
435                                 sts_reg <= STS_BUSY;
436                              end
437                              CMD_SAVE: begin
438                                 if (~cur_view) begin  // Make sure current view is unread
439                                    state <= S_DEL_MSG;
440                                    S_DEL_save <= 1;
441                                    sts_reg <= STS_BUSY;
442                                 end
443                                 else
444                                    sts_reg <= STS_CMD_RDY;
445                              end
446                              default: sts_reg <= STS_CMD_RDY;
447                           endcase
448                        end
449                     end
450 
451                     ///////////////////////////////////////////////////////////
452                     ////  Detect CF Device before RW operations
453                     ///////////////////////////////////////////////////////////
454 
455                     S_START_RW_INIT: begin  // Detect CF device first
456                        if (CF_cmd_reg == CF_CMD_DETECT) begin
457                           CF_Interface_op <= CF_OP_IDLE;  // S_START_WR_INIT_2
458                           state <= state + 1;  // S_START_RW_INIT_1
459                        end
460                        else
461                           CF_Interface_op <= CF_OP_DETECT;
462                     end
463                     S_START_RW_INIT_1: begin  // Wait for CF Interface to be ready before procee
464                        if (CF_ready) begin
465                           if (wr_op)
466                              state <= state + 1;  // S_START_WR
467                           else
468                              state <= S_START_RD;
469                        end
470                     end
471 
472                     ///////////////////////////////////////////////////////////
473                     ////  Write Voicemail
474                     ///////////////////////////////////////////////////////////
475 
476                     S_START_WR: begin  // Checkout message ID
477                        sample_cnt <= 0;
478                        rst_in_FIFO <= 1;
479                        if (ZBT_ready) begin
480                           if (msg_chkout_cnt != msg_max) begin
481                              msg_chkout_cnt <= msg_chkout_cnt + 1;
482                              msg_pos <= msg_chkout_cnt;
483                              ZBT_addr_reg <= {2'b00, msg_chkout_cnt, 2'b01};
484                              ZBT_din_reg <= {27'h0000000, latch_phn_num, 1'b0};
```

```
485                             ZBT_op_reg <= OP_WRITE;
486                             ZBT_bwe_reg <= 4'b0001;
487                             state <= state + 2;  // S_START_WR_2
488                          end
489                          else if (FIFO_start != FIFO_end) begin
490                             ZBT_addr_reg <= {4'b0010, FIFO_start};
491                             ZBT_op_reg <= OP_READ;
492                             FIFO_start <= FIFO_start + 1;
493                             state <= state + 1;  // S_START_WR_1
494                          end
495                          else begin
496                             state <= S_IDLE;
497                             sts_reg <= STS_ERR_VM_FULL;
498                          end
499                       end
500                    end
501                    S_START_WR_1: begin
502                       if (ZBT_nd) begin
503                          ZBT_addr_reg <= {2'b00, ZBT_dout[35:21], 2'b01};
504                          ZBT_din_reg <= {27'h0000000, latch_phn_num, 1'b0};
505                          ZBT_op_reg <= OP_WRITE;
506                          ZBT_bwe_reg <= 4'b0001;
507                          msg_pos <= ZBT_dout[35:21];
508                          state <= state + 2;  // S_START_WR_3
509                       end
510                       else
511                          ZBT_op_reg <= OP_IDLE;
512                    end
513                    S_START_WR_2: begin  // Wait constraint on consective ZBT ops
514                       ZBT_op_reg <= OP_IDLE;
515                       state <= state + 1;
516                    end
517                    S_START_WR_3: begin  // Store Date & Time
518                       rst_in_FIFO <= 0;
519                       if (ZBT_ready) begin
520                          ZBT_addr_reg <= {2'b00, msg_pos, 2'b10};
521                          ZBT_din_reg <= {latch_DT, 3'b000};
522                          ZBT_op_reg <= OP_WRITE;
523                          ZBT_bwe_reg <= 4'b1111;
524                          state <= state + 1;
525                       end
526                    end
527                    S_START_WR_4: begin // Wait constraint on consective ZBT ops
528                       state <= S_ADD_MSG;
529                       S_ADD_MSG_unread <= 1;
530                       ZBT_op_reg <= OP_IDLE;
531                    end
532                    S_WRING: begin
533                       ZBT_op_reg <= OP_IDLE;
534                       if (cmd == CMD_END_WR) begin
535                          sts_reg <= STS_WR_FIN;
536                          wr_en_override <= 0;
537                          state <= state + 1;  // S_END_WR
538                       end
539                       else if (d_ready) begin
540                          if (in_full) begin  // if write buffer full
541                             sts_reg <= STS_ERR_WR_FAIL;
542                             wr_en_override <= 0;
543                             state <= state + 1; // S_END_WR
544                          end
545                          else if (&sample_cnt[20:0]) begin  // if next sample will be last
```

```
546                     sts_reg <= STS_WR_FIN;
547                     wr_en_override <= 0;
548                     state <= state + 1;  // S_END_WR
549                     sample_cnt <= sample_cnt + 1;
550                  end
551                  else
552                     sample_cnt <= sample_cnt + 1;
553               end
554            end
555            S_END_WR: begin
556               if (!in_data_cnt[10:8] & CF_ready) begin
557                  state <= state + 1;  // S_END_WR_2
558                  if (in_data_cnt[7:0]) begin //  if there are samples left in the incomi
         FIFO
559                     CF_Interface_op <= CF_OP_WR_FORCE; // force remainder of FIFO to be
         written
560                  end
561                  else begin
562                     CF_Interface_op <= CF_OP_IDLE;
563                  end
564               end
565            end
566            S_END_WR_2: begin
567               if (!in_data_cnt && CF_ready) begin  // wait until incoming FIFO is empty
         and CF Interface is ready before moving forward
568                  state <= state + 1;
569                  CF_Interface_op <= CF_OP_IDLE;
570               end
571            end
572            S_END_WR_3: begin  // Store sample count (ZBT should be ready, no need to che
573               ZBT_addr_reg <= {2'b00, msg_pos, 2'b01};
574               ZBT_din_reg <= {sample_cnt, 14'd0};
575               ZBT_op_reg <= OP_WRITE;
576               ZBT_bwe_reg <= 4'b1110;
577               state <= S_IDLE;
578               sts_reg <= STS_CMD_RDY;
579            end
580
581            /////////////////////////////////////////////////////////
582            ////  Read Voicemail
583            /////////////////////////////////////////////////////////
584
585            S_START_RD: begin  // check if message exists before reading
586               if ((~unread_exist & ~cur_view)|(~saved_exist & cur_view)) begin
587                  state <= S_IDLE;
588                  sts_reg <= STS_CMD_RDY;
589               end
590               else
591                  state <= state + 1;  // S_START_RD_1
592            end
593            S_START_RD_1: begin
594               sample_cnt <= msg_pos_sample_cnt;
595               rst_out_FIFO <= 1;
596               state <= state + 1;  // S_START_RD_2
597            end
598            S_START_RD_2: begin
599               rst_out_FIFO <= 0;
600               CF_Interface_op <= CF_OP_RD_EN;
601               if (out_data_cnt[10:8]) begin  // Preload 256 samples into output FIFO
         before proceeding
602                  state <= state + 1;  // S_RDING
```

```
603                             sts_reg <= STS_RDING;
604                             rd_en_override <= 1;
605                          end
606                       end
607                       S_RDING: begin
608                          if (cmd == CMD_END_RD) begin
609                             sts_reg <= STS_RD_FIN;
610                             rd_en_override <= 0;
611                             CF_Interface_op <= CF_OP_IDLE;
612                             state <= state + 1;  // S_END_RD
613                          end
614                          else if (d_ready) begin
615                             if (out_empty) begin  // if read buffer empty
616                                sts_reg <= STS_ERR_RD_FAIL;
617                                rd_en_override <= 0;
618                                CF_Interface_op <= CF_OP_IDLE;
619                                state <= state + 1; // S_END_RD
620                             end
621                             else if (sample_cnt == 1) begin  // if next sample will be last
622                                sts_reg <= STS_RD_FIN;
623                                rd_en_override <= 0;
624                                CF_Interface_op <= CF_OP_IDLE;
625                                state <= state + 1;  // S_END_RD
626                             end
627                             else
628                                sample_cnt <= sample_cnt - 1;
629                          end
630                       end
631                       S_END_RD: begin  // Wait until CF Interface is ready before moving forward
632                          if (CF_ready) begin
633                             state <= S_IDLE;
634                             sts_reg <= STS_CMD_RDY;
635                          end
636                       end
637
638                       ////////////////////////////////////////////////////////
639                       ////  Delete Message from List
640                       ////////////////////////////////////////////////////////
641
642                       S_DEL_MSG: begin
643                          if ((~unread_exist & ~cur_view)|(~saved_exist & cur_view)) begin  // Check
            if del/sve cmd issued when lists are empty
644                             state <= S_IDLE;
645                             sts_reg <= STS_CMD_RDY;
646                          end
647                          else
648                             state <= state + 1;  // S_DEL_MSG_1
649                       end
650                       S_DEL_MSG_1: begin
651                          if (ZBT_ready) begin
652                             ZBT_addr_reg <= {2'b00, msg_pos_prev, 2'b00};  // Set previous msg's
            child to next msg
653                             ZBT_din_reg <= {18'h00000, msg_pos_next, 3'b000};
654                             ZBT_op_reg <= OP_WRITE;
655                             ZBT_bwe_reg <= 4'b0011;
656                             state <= state + 1;  // S_DEL_MSG_2
657                             // Update if msg to delete is start or end of a list;
658                             if (cur_view) begin
659                                if (msg_pos == saved_end) saved_end <= msg_pos_prev;
660                                if (msg_pos == saved_start) saved_start <= msg_pos_next;
661                             end
```

```
662                         else begin
663                            if (msg_pos == unread_end) unread_end <= msg_pos_prev;
664                            if (msg_pos == unread_start) unread_start <= msg_pos_next;
665                         end
666                      end
667                   end
668                   S_DEL_MSG_2: begin  // Wait constraint on consective ZBT ops
669                      state <= state + 1;  // S_DEL_MSG_3
670                      ZBT_op_reg <= OP_IDLE;
671                   end
672                   S_DEL_MSG_3: begin
673                      if (ZBT_ready) begin
674                         ZBT_addr_reg <= {2'b00, msg_pos_next, 2'b00};  // Set next msg's parent
           to previous msg
675                         ZBT_din_reg <= {msg_pos_prev, 21'h000000};
676                         ZBT_op_reg <= OP_WRITE;
677                         ZBT_bwe_reg <= 4'b1100;
678                         state <= state + 1;  // S_DEL_MSG_4
679                      end
680                   end
681                   S_DEL_MSG_4: begin  // Wait constraint on consective ZBT ops
682                      state <= state + 1;  // S_DEL_MSG_5
683                      ZBT_op_reg <= OP_IDLE;
684                   end
685                   S_DEL_MSG_5: begin
686                      if (ZBT_ready) begin
687                         if (msg_pos_prev == msg_pos) begin  // Check if only one message left
688                            if (cur_view)
689                               saved_exist <= 0;
690                            else
691                               unread_exist <= 0;
692                         end
693                         if (S_DEL_save) begin
694                            ZBT_op_reg <= OP_IDLE;
695                            state <= S_ADD_MSG;
696                            S_ADD_MSG_unread <= 0;
697                         end
698                         else begin  // Add deleted msg ID to end of FIFO
699                            ZBT_addr_reg <= {4'b0100, FIFO_end};
700                            ZBT_din_reg <= {msg_pos, 21'h000000};
701                            ZBT_op_reg <= OP_WRITE;
702                            ZBT_bwe_reg <= 4'b1111;
703                            FIFO_end <= FIFO_end + 1;
704                            msg_pos <= msg_pos_prev;
705                            state <= S_UPDT_MSG_POS_DATA;
706                         end
707                      end
708                   end
709
710                   ///////////////////////////////////////////////////////////
711                   ////  Add Message to End of List
712                   ///////////////////////////////////////////////////////////
713
714                   S_ADD_MSG: begin  // Wait constraint on consective ZBT ops
715                      state <= state + 1;  // S_ADD_MSG_1
716                      ZBT_op_reg <= OP_IDLE;
717                   end
718                   S_ADD_MSG_1: begin
719                      if (ZBT_ready) begin
720                         ZBT_op_reg <= OP_WRITE;
721                         ZBT_addr_reg <= {2'b00, msg_pos, 2'b00};
```

```
722                             ZBT_bwe_reg <= 4'b1111;
723                             if (S_ADD_MSG_unread) begin  // writing
724                                if (unread_exist) begin
725                                   ZBT_din_reg <= {unread_end, 3'b000, unread_start, 3'b000};
726                                   state <= state + 1;  // S_ADD_MSG_2
727                                end
728                                else begin
729                                   ZBT_din_reg <= {msg_pos, 3'b000, msg_pos, 3'b000};
730                                   unread_start <= msg_pos;
731                                   unread_end <= msg_pos;
732                                   unread_exist <= 1;
733                                   state <= S_WRING;
734                                   sts_reg <= STS_WRING;
735                                   wr_en_override <= 1;
736                                   CF_Interface_op <= CF_OP_WR_EN;
737                                end
738                             end
739                             else begin  // saving
740                                if (saved_exist) begin
741                                   ZBT_din_reg <= {saved_end, 3'b000, saved_start, 3'b000};
742                                   state <= state + 1;  // S_ADD_MSG_2
743                                end
744                                else begin
745                                   ZBT_din_reg <= {msg_pos, 3'b000, msg_pos, 3'b000};
746                                   saved_start <= msg_pos;
747                                   saved_end <= msg_pos;
748                                   saved_exist <= 1;
749                                   state <= S_UPDT_MSG_POS_DATA;
750                                   msg_pos <= msg_pos_prev;
751                                end
752                             end
753                          end
754                       end
755                       S_ADD_MSG_2: begin  // Wait constraint on consective ZBT ops
756                          state <= state + 1;  // S_ADD_MSG_3
757                          ZBT_op_reg <= OP_IDLE;
758                       end
759                       S_ADD_MSG_3: begin  // Set start's parent to message
760                          if (ZBT_ready) begin
761                             ZBT_op_reg <= OP_WRITE;
762                             ZBT_bwe_reg <= 4'b1100;
763                             ZBT_din_reg <= {msg_pos, 21'h000000};
764                             state <= state + 1;  // S_ADD_MSG_4
765                             if (S_ADD_MSG_unread)
766                                ZBT_addr_reg <= {2'b00, unread_start, 2'b00};
767                             else
768                                ZBT_addr_reg <= {2'b00, saved_start, 2'b00};
769                          end
770                       end
771                       S_ADD_MSG_4: begin  // Wait constraint on consective ZBT ops
772                          state <= state + 1;  // S_ADD_MSG_5
773                          ZBT_op_reg <= OP_IDLE;
774                       end
775                       S_ADD_MSG_5: begin  // Set end's child to message
776                          if (ZBT_ready) begin
777                             ZBT_op_reg <= OP_WRITE;
778                             ZBT_bwe_reg <= 4'b0011;
779                             ZBT_din_reg <= {18'h00000, msg_pos, 3'b000};
780                             if (S_ADD_MSG_unread) begin
781                                ZBT_addr_reg <= {2'b00, unread_end, 2'b00};
782                                unread_end <= msg_pos;
```

```
783                             state <= S_WRING;
784                             sts_reg <= STS_WRING;
785                             wr_en_override <= 1;
786                             CF_Interface_op <= CF_OP_WR_EN;
787                          end
788                          else begin
789                             ZBT_addr_reg <= {2'b00, saved_end, 2'b00};
790                             saved_end <= msg_pos;
791                             state <= S_UPDT_MSG_POS_DATA;
792                             msg_pos <= msg_pos_prev;
793                          end
794                       end
795                    end
796
797                    ////////////////////////////////////////////////////////
798                    ////  Update Message Position Data
799                    ////////////////////////////////////////////////////////
800
801                    S_UPDT_MSG_POS_DATA: begin // Wait constraint on consective ZBT ops
802                       state <= state + 1; // S_UPDT_MSG_POS_DATA_1
803                       ZBT_op_reg <= OP_IDLE;
804                    end
805                    S_UPDT_MSG_POS_DATA_1: begin
806                       if (ZBT_ready) begin
807                          ZBT_addr_reg <= {2'b00, msg_pos, 2'b00};  // Read Parent (Prev) & Chil
          (Next)
808                          ZBT_op_reg <= OP_READ;
809                          state <= state + 1;  // S_UPDT_MSG_POS_DATA_2
810                       end
811                    end
812                    S_UPDT_MSG_POS_DATA_2: begin // Wait constraint on consective ZBT ops
813                       state <= state + 1; // S_UPDT_MSG_POS_DATA_3
814                       ZBT_op_reg <= OP_IDLE;
815                    end
816                    S_UPDT_MSG_POS_DATA_3: begin
817                       ZBT_addr_reg <= {2'b00, msg_pos, 2'b01};  // Read Sample Count & Phone nu
818                       ZBT_op_reg <= OP_READ;
819                       state <= state + 1;  // S_UPDT_MSG_POS_DATA_4
820                    end
821                    S_UPDT_MSG_POS_DATA_4: begin // Wait constraint on consective ZBT ops
822                       state <= state + 1; // S_UPDT_MSG_POS_DATA_5
823                       ZBT_op_reg <= OP_IDLE;
824                    end
825                    S_UPDT_MSG_POS_DATA_5: begin
826                       ZBT_addr_reg <= {2'b00, msg_pos, 2'b10};  // Read Date & Time
827                       ZBT_op_reg <= OP_READ;
828                       state <= state + 1;  // S_UPDT_MSG_POS_DATA_6
829                    end
830                    S_UPDT_MSG_POS_DATA_6: begin
831                       ZBT_op_reg <= OP_IDLE;
832                       msg_pos_prev <= ZBT_dout[35:21];
833                       msg_pos_next <= ZBT_dout[17:3];
834                       state <= state + 1;  // S_UPDT_MSG_POS_DATA_7
835                    end
836                    S_UPDT_MSG_POS_DATA_7: begin
837                       state <= state + 1;  // S_UPDT_MSG_POS_DATA_8
838                    end
839                    S_UPDT_MSG_POS_DATA_8: begin
840                       msg_pos_sample_cnt <= ZBT_dout[35:14];
841                       msg_pos_phn_num <= ZBT_dout[8:1];
842                       state <= state + 1;  // S_UPDT_MSG_POS_DATA_9
```

```
843                     end
844                     S_UPDT_MSG_POS_DATA_9: begin
845                        state <= state + 1;  // S_UPDT_MSG_POS_DATA_10
846                     end
847                     S_UPDT_MSG_POS_DATA_10: begin
848                        msg_pos_DT <= ZBT_dout[35:3];
849                        sts_reg <= STS_CMD_RDY;
850                        state <= S_IDLE;
851                        disp_req <= 1;
852                     end
853                  endcase
854               end
855            end
856
857            // Manage commands to CF Interface as needed
858            reg [12:0] cur_LBA;
859            wire [7:0] in_SC, out_SC;
860            wire [1:0] out_SC_low;
861            assign in_SC = (in_data_cnt[10]) ? 8'd4: {6'd0, in_data_cnt[9:8]};
862            assign out_SC_low = 2'd3 - out_data_cnt[9:8];
863            assign out_SC = (out_data_cnt[10]) ? 8'd0: {6'd0, out_SC_low};
864            always @(posedge clk_27mhz) begin
865               if (reset) begin
866                  CF_cmd_reg <= CF_CMD_IDLE;
867                  CF_LBA_reg <= 0;
868                  CF_SC_reg <= 0;
869                  cur_LBA <= 0;
870               end
871               else begin
872                  case (CF_Interface_op)
873                     CF_OP_WR_EN: begin
874                        if (CF_ready && in_SC) begin
875                           CF_cmd_reg <= CF_CMD_WRITE;
876                           CF_LBA_reg <= {msg_pos, cur_LBA};
877                           CF_SC_reg <= in_SC;
878                           cur_LBA <= (cur_LBA + CF_SC_reg);
879                        end
880                        else
881                           CF_cmd_reg <= CF_CMD_IDLE;
882                     end
883                     CF_OP_RD_EN: begin
884                        if (CF_ready && out_SC) begin
885                           CF_cmd_reg <= CF_CMD_READ;
886                           CF_LBA_reg <= {msg_pos, cur_LBA};
887                           CF_SC_reg <= out_SC;
888                           cur_LBA <= (cur_LBA + CF_SC_reg);
889                        end
890                        else
891                           CF_cmd_reg <= CF_CMD_IDLE;
892                     end
893                     CF_OP_WR_FORCE: begin
894                        if (CF_ready  && in_data_cnt) begin
895                           CF_cmd_reg <= CF_CMD_WRITE;
896                           CF_LBA_reg <= {msg_pos, cur_LBA};
897                           CF_SC_reg <= 1;
898                        end
899                        else
900                           CF_cmd_reg <= CF_CMD_IDLE;
901                     end
902                     CF_OP_DETECT: begin
903                        if (CF_ready)
```

```
904                         CF_cmd_reg <= CF_CMD_DETECT;
905                   end
906                   default: begin // CD_OP_IDLE
907                      CF_cmd_reg <= CF_CMD_IDLE;
908                      cur_LBA <= 0;
909                      CF_SC_reg <= 0;
910                   end
911 
912                endcase
913             end
914          end
915 
916          // Instantiate ASCII state
917          reg [4:0] ascii_state;
918 
919          // Declare ASCII state parameters
920          parameter S_ASCII_IDLE          = 5'h00;
921          parameter S_ASCII_DISP          = 5'h01;
922          parameter S_ASCII_DISP_1        = 5'h02;
923          parameter S_ASCII_DISP_2        = 5'h03;
924          parameter S_ASCII_DISP_3        = 5'h04;
925          parameter S_ASCII_DISP_4        = 5'h05;
926          parameter S_ASCII_DISP_5        = 5'h06;
927          parameter S_ASCII_DISP_6        = 5'h07;
928          parameter S_ASCII_DISP_7        = 5'h08;
929          parameter S_ASCII_DISP_8        = 5'h09;
930          parameter S_ASCII_DISP_9        = 5'h0A;
931          parameter S_ASCII_DISP_10       = 5'h0B;
932          parameter S_ASCII_DISP_11       = 5'h0C;
933          parameter S_ASCII_DISP_12       = 5'h0D;
934          parameter S_ASCII_DISP_13       = 5'h0E;
935          parameter S_ASCII_DISP_14       = 5'h0F;
936          parameter S_ASCII_DISP_15       = 5'h10;
937          parameter S_ASCII_DISP_16       = 5'h11;
938          parameter S_ASCII_DISP_17       = 5'h12;
939          parameter S_ASCII_DISP_18       = 5'h13;
940          parameter S_ASCII_DISP_19       = 5'h14;
941          parameter S_ASCII_DISP_20       = 5'h15;
942          parameter S_ASCII_DISP_EMPTY    = 5'h16;
943          parameter S_ASCII_DISP_EMPTY_1  = 5'h17;
944          parameter S_ASCII_DISP_EMPTY_2  = 5'h18;
945          parameter S_ASCII_DISP_EMPTY_3  = 5'h19;
946 
947 
948          // Instantiate ASCII registers
949          reg [7:0] ascii_out_reg;
950          reg ascii_out_ready_reg;
951          reg [6:0] addr_reg;
952 
953          // Assign ASCII registers
954          assign ascii_out = ascii_out_reg;
955          assign ascii_out_ready = ascii_out_ready_reg;
956          assign addr = addr_reg;
957 
958          // Instantiate capture registers
959          reg [25:0] DT_capture;
960          reg [7:0] phn_num_capture;
961          reg [3:0] phn_num_minus_nine;
962          reg empty_capture;
963 
964          // Queue display requests as needed
```

```
965         reg q_disp_req, l_disp_req;
966         always @(posedge clk_27mhz) begin
967            if (reset) begin
968               l_disp_req <= 0;
969            end
970            else if (disp_req) begin  // latch if high
971               l_disp_req <= 1;
972            end
973            else if (ascii_state == S_ASCII_IDLE) begin
974               l_disp_req <= 0;
975            end
976         end
977         always @(posedge clk_27mhz) begin
978            if (reset) begin
979               q_disp_req <= 0;
980            end
981            else if (l_disp_req & (ascii_state == S_ASCII_IDLE)) begin
982               q_disp_req <= 1;
983            end
984            else begin
985               q_disp_req <= 0;
986            end
987         end
988
989         // Manage ASCII output when display requested or enabled
990         always @(posedge clk_27mhz) begin
991            if (reset) begin
992               ascii_state <= 0;  // S_ASCII_IDLE
993               ascii_out_reg <= 0;
994               ascii_out_ready_reg <= 0;
995               addr_reg <= 0;
996               DT_capture <= 0;
997               phn_num_capture <= 0;
998               empty_capture <= 0;
999            end
1000           else begin
1001              case (ascii_state)
1002                 S_ASCII_IDLE: begin
1003                    ascii_out_reg <= 0;
1004                    ascii_out_ready_reg <= 0;
1005                    addr_reg <= msg_pos_DT[32:26];
1006                    if (disp_en & q_disp_req) begin
1007                       ascii_state <= ascii_state + 1;  // S_ASCII_DISP
1008                       DT_capture <= msg_pos_DT[25:0];
1009                       phn_num_capture <= msg_pos_phn_num;
1010                       empty_capture <= ((~unread_exist & ~cur_view)|(~saved_exist & cur_view
1011                    end
1012                 end
1013                 S_ASCII_DISP: begin
1014                    if (empty_capture)  // Test if empty
1015                       ascii_state <= S_ASCII_DISP_EMPTY;
1016                    else
1017                       ascii_state <= ascii_state + 1;  // S_ASCII_DISP_1
1018                 end
1019                 S_ASCII_DISP_1: begin
1020                    addr_reg <= {3'b000, DT_capture[25:22]};
1021                    ascii_out_reg <= {4'b0011, data[7:4]};  // year, first digit
1022                    ascii_out_ready_reg <= 1;
1023                    ascii_state <= ascii_state + 1;  // S_ASCII_DISP_2
1024                 end
1025                 S_ASCII_DISP_2: begin
```

```
1026                            ascii_out_reg <= {4'b0011, data[3:0]};  // year, second digit
1027                            ascii_state <= ascii_state + 1;  // S_ASCII_DISP_3
1028                         end
1029                         S_ASCII_DISP_3: begin
1030                            ascii_out_reg <= 8'h2F;  // forward slash
1031                            ascii_state <= ascii_state + 1;  // S_ASCII_DISP_4
1032                         end
1033                         S_ASCII_DISP_4: begin
1034                            addr_reg <= {2'b00, DT_capture[21:17]};
1035                            ascii_out_reg <= {4'b0011, data[7:4]};  // month, first digit
1036                            ascii_state <= ascii_state + 1;  // S_ASCII_DISP_5
1037                         end
1038                         S_ASCII_DISP_5: begin
1039                            ascii_out_reg <= {4'b0011, data[3:0]};  // month, second digit
1040                            ascii_state <= ascii_state + 1;  // S_ASCII_DISP_6
1041                         end
1042                         S_ASCII_DISP_6: begin
1043                            ascii_out_reg <= 8'h2F;  // forward slash
1044                            ascii_state <= ascii_state + 1;  // S_ASCII_DISP_7
1045                         end
1046                         S_ASCII_DISP_7: begin
1047                            addr_reg <= {2'b00, DT_capture[16:12]};
1048                            ascii_out_reg <= {4'b0011, data[7:4]};  // day, first digit
1049                            ascii_state <= ascii_state + 1;  // S_ASCII_DISP_8
1050                         end
1051                         S_ASCII_DISP_8: begin
1052                            ascii_out_reg <= {4'b0011, data[3:0]};  // day, second digit
1053                            ascii_state <= ascii_state + 1;  // S_ASCII_DISP_9
1054                         end
1055                         S_ASCII_DISP_9: begin
1056                            ascii_out_reg <= 8'h20;  // space
1057                            ascii_state <= ascii_state + 1;  // S_ASCII_DISP_10
1058                         end
1059                         S_ASCII_DISP_10: begin
1060                            addr_reg <= {1'b0, DT_capture[11:6]};
1061                            ascii_out_reg <= {4'b0011, data[7:4]};  // hour, first digit
1062                            ascii_state <= ascii_state + 1;  // S_ASCII_DISP_11
1063                         end
1064                         S_ASCII_DISP_11: begin
1065                            ascii_out_reg <= {4'b0011, data[3:0]};  // hour, second digit
1066                            ascii_state <= ascii_state + 1;  // S_ASCII_DISP_12
1067                         end
1068                         S_ASCII_DISP_12: begin
1069                            ascii_out_reg <= 8'h3A;  // colon
1070                            ascii_state <= ascii_state + 1;  // S_ASCII_DISP_13
1071                         end
1072                         S_ASCII_DISP_13: begin
1073                            addr_reg <= {1'b0, DT_capture[5:0]};
1074                            ascii_out_reg <= {4'b0011, data[7:4]};  // minute, first digit
1075                            ascii_state <= ascii_state + 1;  // S_ASCII_DISP_14
1076                         end
1077                         S_ASCII_DISP_14: begin
1078                            ascii_out_reg <= {4'b0011, data[3:0]};  // minute, second digit
1079                            ascii_state <= ascii_state + 1;  // S_ASCII_DISP_15
1080                         end
1081                         S_ASCII_DISP_15: begin
1082                            ascii_out_reg <= 8'h3A;  // colon
1083                            ascii_state <= ascii_state + 1;  // S_ASCII_DISP_16
1084                         end
1085                         S_ASCII_DISP_16: begin
1086                            ascii_out_reg <= {4'b0011, data[7:4]};  // second, first digit
```

```
1087                    ascii_state <= ascii_state + 1;  // S_ASCII_DISP_17
1088                 end
1089                 S_ASCII_DISP_17: begin
1090                    ascii_out_reg <= {4'b0011, data[3:0]};  // second, second digit
1091                    ascii_state <= ascii_state + 1;  // S_ASCII_DISP_18
1092                 end
1093                 S_ASCII_DISP_18: begin
1094                    ascii_out_reg <= 8'h20;  // space
1095                    ascii_state <= ascii_state + 1;  // S_ASCII_DISP_19
1096                    phn_num_minus_nine <= phn_num_capture[7:4] - 4'd9;
1097                 end
1098                 S_ASCII_DISP_19: begin
1099                    ascii_out_reg <= (phn_num_capture[7:4] < 4'd10) ? {4'b0011, phn_num_captu
        :4]}: {4'b0100, phn_num_minus_nine};
1100                    ascii_state <= ascii_state + 1;  // S_ASCII_DISP_20
1101                    phn_num_minus_nine <= phn_num_capture[3:0] - 4'd9;
1102                 end
1103                 S_ASCII_DISP_20: begin
1104                    ascii_out_reg <= (phn_num_capture[3:0] < 4'd10) ? {4'b0011, phn_num_captu
        :0]}: {4'b0100, phn_num_minus_nine};
1105                    ascii_state <= S_ASCII_IDLE;
1106                 end
1107                 S_ASCII_DISP_EMPTY: begin  // Display "None" since empty
1108                    ascii_out_reg <= 8'h4E; // N
1109                    ascii_state <= ascii_state + 1;  // S_ASCII_DISP_EMPTY_1
1110                    ascii_out_ready_reg <= 1;
1111                 end
1112                 S_ASCII_DISP_EMPTY_1: begin
1113                    ascii_out_reg <= 8'h6F; // o
1114                    ascii_state <= ascii_state + 1;  // S_ASCII_DISP_EMPTY_2
1115                 end
1116                 S_ASCII_DISP_EMPTY_2: begin
1117                    ascii_out_reg <= 8'h6E; // n
1118                    ascii_state <= ascii_state + 1;  // S_ASCII_DISP_EMPTY_3
1119                 end
1120                 S_ASCII_DISP_EMPTY_3: begin
1121                    ascii_out_reg <= 8'h65; // e
1122                    ascii_state <= S_ASCII_IDLE;
1123                 end
1124              endcase
1125           end
1126        end
1127     endmodule
1128
1129     ////////////////////////////////////////////////////////////////////////////////////
1130     //// Mid-Level Interface Module
1131     ////////////////////////////////////////////////////////////////////////////////////
1132
1133     ////////////////////////////////////////////////////////////////////////////////////
1134     // Provide a mid-level interface for making reads and writes to Compact Flash.
1135     // READING:
1136     //     o On read request, provide read command, LBA, and sector count (SC).
1137     //     o Sends back data in 16-bit words as it arrives with nd (new data) signal.
1138     // WRITING:
1139     //     o On write request, provide write command, LBA, and sector count (SC).
1140     //     o Sends back we_req for requesting write data to be sent.
1141     //     o Write data must be sent during the SAME clock cycle we_req is asserted.
1142     //     o Use FWFT on any external FIFOs for 0 latency reads from the FIFOs.
1143     // STATUS:
1144     //     o Tells if Compact Flash device is present. Manual CF Detect commands
1145     //        needed in idle state. Done automatically when needed for R/W.
```

```
1146        //     o If CF device is present, gives number of LBAs on device.
1147        //     o Provides signal indicating whether system is ready for command.
1148        // COMMANDS:
1149        //     o CMD_IDLE:   2'b00
1150        //     o CMD_DETECT: 2'b01
1151        //     o CMD_READ:   2'b10
1152        //     o CMD_WRITE:  2'b11
1153        // SPECIAL NOTES:
1154        //     o SC value of 0 corresponds to 256.
1155        //     o Ready signal is asserted cycle BEFORE data can be issued.
1156        //     o Ensure successive commands have at least one cycle between them.
1157        ///////////////////////////////////////////////////////////////////////////////////
1158
1159        module CF_Interface #(
1160           parameter CMD_RDY_TO_PD = 1620000,  // SystemACE Command Ready Timeout Period
1161           parameter BFR_RDY_TO_PD = 1620000   // SystemACE Buffer Ready Timeout Period
1162           )(
1163           input clk_27mhz,          // 27 MHz clock
1164           input reset,              // Synchronous reset
1165           // Mid-level I/O
1166           input [1:0] cmd,          // Command to be performed
1167           input [27:0] LBA,         // Logical Block Address for r/w
1168           input [7:0] SC,           // Sector Count
1169           input [15:0] din,         // Data input for writes
1170           output we_req,            // Request for write data
1171           output [15:0] dout,       // Data output for reads
1172           output nd,                // New Data available at output
1173           output CF_detect,         // Detect if CF device is connected
1174           output [27:0] LBA_max,    // Maximum number of LBAs if CF detected
1175           output ready,             // Command ready signal
1176           // SystemACE ports
1177           inout [15:0] systemace_data,     // SystemACE R/W data
1178           output [6:0] systemace_address,  // SystemACE R/W address
1179           output systemace_ce_b,           // SystemACE chip enable (Active Low)
1180           output systemace_we_b,           // SystemACE write enable (Active Low)
1181           output systemace_oe_b,           // SystemACE output enable (Active Low)
1182           input systemace_mpbrdy           // SystemACE MPU buffer ready
1183           );
1184
1185           // Instantiate Low-Level I/O for basic R/W to SystemACE MPU
1186           wire [15:0] rw_din, rw_dout;
1187           wire [5:0] addr;
1188           wire [6:0] addr_large;
1189           wire re, we, rw_nd, rw_ready;
1190           assign addr_large = {addr, 1'b0};
1191           RW_Interface RW_INTERFACE_1(
1192              .clk_27mhz(clk_27mhz),
1193              .reset(reset),
1194              // RW ports
1195              .addr(addr_large),
1196              .din(rw_din),
1197              .we(we),
1198              .re(re),
1199              .dout(rw_dout),
1200              .nd(rw_nd),
1201              .ready(rw_ready),
1202              // SystemACE ports
1203              .systemace_data(systemace_data),
1204              .systemace_address(systemace_address),
1205              .systemace_ce_b(systemace_ce_b),
1206              .systemace_we_b(systemace_we_b),
```

```
1207                .systemace_oe_b(systemace_oe_b)
1208            );
1209
1210            // Declare command parameters
1211            parameter CMD_IDLE   = 2'b00;
1212            parameter CMD_DETECT = 2'b01;
1213            parameter CMD_READ   = 2'b10;
1214            parameter CMD_WRITE  = 2'b11;
1215
1216            // Instantiate state
1217            reg [5:0] state;
1218
1219            // Declare state parameters
1220            parameter S_SET_WORD_MODE    = 6'h00;
1221            parameter S_SET_WORD_MODE_1  = 6'h01;
1222            parameter S_FORCE_LOCK       = 6'h02;
1223            parameter S_FORCE_LOCK_1     = 6'h03;
1224            parameter S_CHK_LOCK         = 6'h04;
1225            parameter S_CHK_LOCK_1       = 6'h05;
1226            parameter S_IDLE             = 6'h06;
1227            parameter S_CHK_RDY          = 6'h07;
1228            parameter S_CHK_RDY_1        = 6'h08;
1229            parameter S_CHK_RDY_2        = 6'h09;
1230            parameter S_SET_LBA          = 6'h0A;
1231            parameter S_SET_LBA_1        = 6'h0B;
1232            parameter S_SET_LBA_2        = 6'h0C;
1233            parameter S_SET_LBA_3        = 6'h0D;
1234            parameter S_SET_SC_CMD       = 6'h0E;
1235            parameter S_SET_SC_CMD_1     = 6'h0F;
1236            parameter S_RESET_CFG        = 6'h10;
1237            parameter S_RESET_CFG_1      = 6'h11;
1238            parameter S_INIT_BFR_CNT     = 6'h12;
1239            parameter S_CHK_BFR_RDY      = 6'h13;
1240            parameter S_CHK_BFR_RDY_1    = 6'h14;
1241            parameter S_CHK_BFR_RDY_2    = 6'h15;
1242            parameter S_CHK_BFR_RDY_3    = 6'h16;
1243            parameter S_INIT_DATA_CNT    = 6'h17;
1244            parameter S_WR_BFR           = 6'h18;
1245            parameter S_WR_BFR_1         = 6'h19;
1246            parameter S_RD_BFR           = 6'h1A;
1247            parameter S_RD_BFR_1         = 6'h1B;
1248            parameter S_RD_BFR_2         = 6'h1C;
1249            parameter S_CLR_RESET_CFG    = 6'h1D;
1250            parameter S_CLR_RESET_CFG_1  = 6'h1E;
1251            parameter S_ABORT_CMD        = 6'h1F;
1252            parameter S_ABORT_CMD_1      = 6'h20;
1253            parameter S_ABORT_CMD_2      = 6'h21;
1254            parameter S_ABORT_CMD_3      = 6'h22;
1255            parameter S_CF_DETECT        = 6'h23;
1256            parameter S_CF_DETECT_1      = 6'h24;
1257            parameter S_CF_DETECT_FAIL   = 6'h25;
1258            parameter S_CF_DETECT_FAIL_1 = 6'h26;
1259
1260            // Instantiate RW Interface input registers
1261            reg [5:0] addr_reg;
1262            reg [15:0] rw_din_reg;
1263            reg re_reg, we_reg;
1264
1265            // Assign RW Interface inputs
1266            assign addr = addr_reg;
1267            assign rw_din = rw_din_reg;
```

```
1268            assign we = we_reg;
1269            assign re = re_reg;
1270
1271            // Instantiate Mid-Level output registers
1272            reg we_req_reg;
1273            reg nd_reg;
1274            reg CF_detect_reg;
1275            reg [27:0] LBA_max_reg;
1276            reg ready_reg;
1277
1278            // Assign Mid-Level outputs
1279            assign we_req = we_req_reg;
1280            assign dout = rw_dout;
1281            assign nd = nd_reg;
1282            assign CF_detect = CF_detect_reg;
1283            assign LBA_max = LBA_max_reg;
1284            assign ready = ready_reg;
1285
1286            // Instantiate return states for function calls
1287            reg [5:0] rtn_state_CF_detect;
1288
1289            // Instantiate op register for passing function parameters
1290            reg [1:0] op_reg;
1291
1292            // Declare op parameters
1293            parameter OP_IDENTIFY = 2'h0;
1294            parameter OP_READ     = 2'h1;
1295            parameter OP_WRITE    = 2'h2;
1296
1297            // Instantiate registers for storing Mid-Level input
1298            reg [27:0] LBA_reg;
1299            reg [7:0] SC_reg;
1300
1301            // Instantiate internal registers
1302            reg nd_raw_reg;
1303            reg [3:0] data_cnt;
1304            reg [11:0] bfr_cnt;
1305            reg [20:0] timeout_cntr;  // Ensure enough bits for TO_PD parameters
1306
1307            // Manage state transitions
1308            always @(posedge clk_27mhz) begin
1309               if (reset) begin
1310                  // RW Interface Inputs
1311                  addr_reg <= 0;
1312                  rw_din_reg <= 0;
1313                  we_reg <= 0;
1314                  re_reg <= 0;
1315                  // Mid-Level outputs
1316                  we_req_reg <= 0;
1317                  nd_reg <= 0;
1318                  CF_detect_reg <= 0;
1319                  LBA_max_reg <= 0;
1320                  ready_reg <= 0;
1321                  // Mid-Level inputs
1322                  LBA_reg <= 0;
1323                  SC_reg <= 0;
1324                  // Internal regsiters
1325                  nd_raw_reg <= 0;
1326                  data_cnt <= 0;
1327                  bfr_cnt <= 0;
1328                  timeout_cntr <= 0;
```

```
1329                 // Set state to enter after reset
1330                 state <= 0; // S_SET_WORD_MODE
1331                 rtn_state_CF_detect <= S_CF_DETECT_FAIL;
1332              end
1333              else
1334                 case (state)
1335                    S_SET_WORD_MODE: begin  // Set Bus Mode to WORD MODE
1336                       if (rw_ready) begin
1337                          addr_reg <= 6'h00;  // Write BUSMODEREG[7:0]
1338                          rw_din_reg <= 16'h0001;
1339                          re_reg <= 0;
1340                          we_reg <= 1;
1341                          state <= state + 1;  // S_SET_WORD_MODE_1
1342                       end
1343                    end
1344                    S_SET_WORD_MODE_1: begin
1345                       re_reg <= 0;
1346                       we_reg <= 0;
1347                       state <= state + 1;  // S_FORCE_LOCK
1348                    end
1349                    S_FORCE_LOCK: begin  // Force lock on CompactFlash resource
1350                       if (rw_ready) begin
1351                          addr_reg <= 6'h0C;  // Write CONTROLREG[15:0]
1352                          rw_din_reg <= 16'h0003;
1353                          re_reg <= 0;
1354                          we_reg <= 1;
1355                          state <= state + 1;  // S_FORCE_LOCK_1
1356                       end
1357                    end
1358                    S_FORCE_LOCK_1: begin
1359                       re_reg <= 0;
1360                       we_reg <= 0;
1361                       state <= state + 1;  // S_CHK_LOCK
1362                    end
1363                    S_CHK_LOCK: begin  // Check if lock has been acquired
1364                       if (rw_ready) begin
1365                          addr_reg <= 6'h02;  // Read STATUSREG[15:0]
1366                          re_reg <= 1;
1367                          we_reg <= 0;
1368                          state <= state + 1;  // S_CHK_LOCK_1
1369                       end
1370                    end
1371                    S_CHK_LOCK_1: begin
1372                       re_reg <= 0;
1373                       we_reg <= 0;
1374                       if (rw_nd) begin
1375                          if (rw_dout[1]) begin
1376                             state <= S_CF_DETECT_FAIL;
1377                          end
1378                          else
1379                             state <= state - 1; // S_CHK_LOCK
1380                       end
1381                    end
1382                    S_IDLE: begin
1383                       case (cmd)
1384                          CMD_IDLE: ready_reg <= 1;
1385                          CMD_DETECT: begin
1386                             ready_reg <= 0;
1387                             rtn_state_CF_detect <= state;  // S_IDLE
1388                             state <= S_CF_DETECT;
1389                          end
```

```
1390                        CMD_READ: begin
1391                           ready_reg <= 0;
1392                           LBA_reg <= LBA;
1393                           SC_reg <= SC;
1394                           op_reg <= OP_READ;
1395                           state <= state + 1;  // S_CHK_RDY
1396                        end
1397                        CMD_WRITE: begin
1398                           ready_reg <= 0;
1399                           LBA_reg <= LBA;
1400                           SC_reg <= SC;
1401                           op_reg <= OP_WRITE;
1402                           state <= state + 1;  // S_CHK_RDY
1403                        end
1404                        // No default needed
1405                     endcase
1406                  end
1407
1408                  ////////////////////////////////////////////////////////
1409                  ////  R/W & Identify
1410                  ////////////////////////////////////////////////////////
1411
1412                  S_CHK_RDY: begin  // Check if the SystemACE controller is ready
1413                     timeout_cntr <= CMD_RDY_TO_PD;
1414                     state <= state + 1;  // S_CHK_RDY_1
1415                  end
1416                  S_CHK_RDY_1: begin
1417                     if (rw_ready) begin
1418                        addr_reg <= 6'h02;  // Read STATUSREG[15:0]
1419                        re_reg <= 1;
1420                        we_reg <= 0;
1421                        state <= state + 1;  // S_CHK_RDY_2
1422                     end
1423                  end
1424                  S_CHK_RDY_2: begin
1425                     re_reg <= 0;
1426                     we_reg <= 0;
1427                     timeout_cntr <= timeout_cntr - 1;
1428                     if (!timeout_cntr) begin
1429                        rtn_state_CF_detect <= state - 2;  // S_CHK_RDY
1430                        state <= S_ABORT_CMD;
1431                     end
1432                     else if (rw_nd) begin
1433                        if (rw_dout[8]) begin
1434                           state <= state + 1;  // S_SET_LBA
1435                        end
1436                        else begin
1437                           state <= state - 1;  // S_CHK_RDY_1
1438                        end
1439                     end
1440                  end
1441                  S_SET_LBA: begin  // Set Logical Block Address
1442                     if (rw_ready) begin
1443                        addr_reg <= 6'h08;  // Write MPULBAREG[15:0]
1444                        rw_din_reg <= (op_reg == OP_IDENTIFY) ? 16'h0000: LBA_reg[15:0];
1445                        re_reg <= 0;
1446                        we_reg <= 1;
1447                        state <= state + 1;  // S_SET_LBA_1
1448                     end
1449                  end
1450                  S_SET_LBA_1: begin
```

```
1451                            re_reg <= 0;
1452                            we_reg <= 0;
1453                            state <= state + 1;  // S_SET_LBA_2
1454                         end
1455                         S_SET_LBA_2: begin
1456                            if (rw_ready) begin
1457                               addr_reg <= 6'h09;  // Write MPULBAREG[31:16]
1458                               rw_din_reg <= (op_reg == OP_IDENTIFY) ? 16'h0000: {4'h0, LBA_reg[27:16
1459                               re_reg <= 0;
1460                               we_reg <= 1;
1461                               state <= state + 1;  // S_SET_LBA_3
1462                            end
1463                         end
1464                         S_SET_LBA_3: begin
1465                            re_reg <= 0;
1466                            we_reg <= 0;
1467                            state <= state + 1;  // S_SET_SC_CMD
1468                         end
1469                         S_SET_SC_CMD: begin  // Set sector count & command
1470                            if (rw_ready) begin
1471                               addr_reg <= 6'h0A;  // Write SECCNTCMDREG[15:0]
1472                               case (op_reg)
1473                                  OP_IDENTIFY: rw_din_reg <= 16'h0201;
1474                                  OP_READ: rw_din_reg <= {8'h03, SC_reg};
1475                                  OP_WRITE: rw_din_reg <= {8'h04, SC_reg};
1476                                  default: rw_din_reg <= 16'h0001;  // if unused, do nothing
1477                               endcase
1478                               re_reg <= 0;
1479                               we_reg <= 1;
1480                               state <= state + 1;  // S_SET_SC_CMD_1
1481                            end
1482                         end
1483                         S_SET_SC_CMD_1: begin
1484                            re_reg <= 0;
1485                            we_reg <= 0;
1486                            state <= state + 1;  // S_RESET_CFG
1487                         end
1488                         S_RESET_CFG: begin  // Reset the Configuration/CompactFlash Controllers
1489                            if (rw_ready) begin
1490                               addr_reg <= 6'h0C;  // Write CONTROLREG[31:16]
1491                               rw_din_reg <= 16'h0083;
1492                               re_reg <= 0;
1493                               we_reg <= 1;
1494                               state <= state + 1;  // S_RESET_CFG_1
1495                            end
1496                         end
1497                         S_RESET_CFG_1: begin
1498                            re_reg <= 0;
1499                            we_reg <= 0;
1500                            state <= state + 1;  // S_INIT_BFR_CNT
1501                         end
1502                         S_INIT_BFR_CNT: begin  // Initialize the buffer count
1503                            bfr_cnt <= (op_reg == OP_IDENTIFY) ? 12'h010: {SC_reg, 4'h0};
1504                            state <= state + 1;  // S_CHK_BFR_RDY
1505                         end
1506                         S_CHK_BFR_RDY: begin
1507                            timeout_cntr <= BFR_RDY_TO_PD;
1508                            state <= state + 1;  // S_CHK_BFR_RDY_1
1509                         end
1510                         S_CHK_BFR_RDY_1: begin  // takes times for buffer ready signal to set
1511                            state <= state + 1;  // S_CHK_BFR_RDY_2
```

```
1512                    end
1513                    S_CHK_BFR_RDY_2: begin
1514                       state <= state + 1;  // S_CHK_BFR_RDY_3
1515                    end
1516                    S_CHK_BFR_RDY_3: begin
1517                       timeout_cntr <= timeout_cntr - 21'd1;
1518                       if (systemace_mpbrdy)
1519                          state <= state + 1;  // S_INIT_DATA_CNT;
1520                       else if (!timeout_cntr) begin
1521                          rtn_state_CF_detect <= state - 3;  // S_CHK_BFR_RDY
1522                          state <= S_ABORT_CMD;
1523                       end
1524                    end
1525                    S_INIT_DATA_CNT: begin  // Initialize data count
1526                       data_cnt <= 4'hF;  // one less than real data count
1527                       if (op_reg == OP_WRITE) begin
1528                          state <= S_WR_BFR;
1529                          we_req_reg <= 1;
1530                       end
1531                       else begin
1532                          addr_reg <= 6'h20;  // Read DATABUFREG[15:0]
1533                          re_reg <= 1;
1534                          we_reg <= 0;
1535                          state <= S_RD_BFR;
1536                          nd_raw_reg <= 0;
1537                          nd_reg <= 0;
1538                       end
1539                    end
1540                    S_WR_BFR: begin  // Write data word to buffer
1541                       // Assume rw_ready
1542                       addr_reg <= 6'h20;  // Write DATABUFREG[15:0]
1543                       rw_din_reg <= din;
1544                       re_reg <= 0;
1545                       we_reg <= 1;
1546                       state <= state + 1;  // S_WR_BFR_1
1547                       we_req_reg <= 0;
1548                    end
1549                    S_WR_BFR_1: begin
1550                       data_cnt <= data_cnt - 1;
1551                       re_reg <= 0;
1552                       we_reg <= 0;
1553                       if (!data_cnt) begin
1554                          bfr_cnt <= bfr_cnt - 1;
1555                          if (bfr_cnt == 1)
1556                             state <= S_CLR_RESET_CFG;
1557                          else
1558                             state <= S_CHK_BFR_RDY;
1559                       end
1560                       else begin
1561                          state <= state - 1;  // S_WR_BFR
1562                          we_req_reg <= 1;
1563                       end
1564                    end
1565                    S_RD_BFR: begin  // Read data word from buffer, set LBA_max if identifying
1566                       re_reg <= 0;
1567                       we_reg <= 0;
1568                       nd_raw_reg <= 0;
1569                       nd_reg <= 0;
1570                       if (nd_raw_reg) data_cnt <= data_cnt - 1;
1571                       if ((op_reg == OP_IDENTIFY) & nd_raw_reg & (bfr_cnt == 13)) begin
1572                          if (data_cnt == 3) // Word 60
```

```
1573                    LBA_max_reg <= {12'h000, rw_dout};
1574                 if (data_cnt == 2) // Word 61
1575                    LBA_max_reg <= {rw_dout[11:0], LBA_max_reg[15:0]};
1576              end
1577              if (nd_raw_reg & !data_cnt) begin
1578                 bfr_cnt <= bfr_cnt - 1;
1579                 if (bfr_cnt == 1)
1580                    state <= S_CLR_RESET_CFG;
1581                 else
1582                    state <= S_CHK_BFR_RDY;
1583              end
1584              else
1585                 state <= state + 1;  // S_RD_BFR_1
1586           end
1587           S_RD_BFR_1: begin
1588              state <= state + 1;  // S_RD_BFR_2
1589           end
1590           S_RD_BFR_2: begin
1591              // Assume rw_ready
1592              addr_reg <= 6'h20;  // Read DATABUFREG[15:0]
1593              re_reg <= 1;
1594              we_reg <= 0;
1595              state <= state - 2; // S_RD_BFR_1
1596              nd_raw_reg <= 1;
1597              nd_reg <= (op_reg == OP_READ);
1598           end
1599           S_CLR_RESET_CFG: begin  // Clear Configuration/CompactFlash controller reset
1600              if (rw_ready) begin
1601                 addr_reg <= 6'h0C;  // Write CONTROLREG[31:16]
1602                 rw_din_reg <= 16'h0003;
1603                 re_reg <= 0;
1604                 we_reg <= 1;
1605                 state <= state + 1;  // S_RESET_CFG_1
1606              end
1607           end
1608           S_CLR_RESET_CFG_1: begin  // Exit subroutine, return to S_IDLE
1609              re_reg <= 0;
1610              we_reg <= 0;
1611              state <= S_IDLE;
1612              ready_reg <= 1;
1613              CF_detect_reg <= 1;
1614           end
1615
1616           //////////////////////////////////////////////////////////
1617           ////  Error Handling
1618           //////////////////////////////////////////////////////////
1619
1620           S_ABORT_CMD: begin  // Abort current command
1621              if (rw_ready) begin
1622                 addr_reg <= 6'h0A;  // Write SECCNTCMDREG[15:0]
1623                 rw_din_reg <= 16'h0601;
1624                 re_reg <= 0;
1625                 we_reg <= 1;
1626                 state <= state + 1;  // S_ABORT_CMD_1
1627              end
1628           end
1629           S_ABORT_CMD_1: begin
1630              re_reg <= 0;
1631              we_reg <= 0;
1632              state <= state + 1;  // S_ABORT_CMD_2
1633           end
```

```
1634                S_ABORT_CMD_2: begin  // Clear command bits
1635                   if (rw_ready) begin
1636                      addr_reg <= 6'h0A;  // Write SECCNTCMDREG[15:0]
1637                      rw_din_reg <= 16'h0001;
1638                      re_reg <= 0;
1639                      we_reg <= 1;
1640                      state <= state + 1;  // S_ABORT_CMD_3
1641                   end
1642                end
1643                S_ABORT_CMD_3: begin
1644                   re_reg <= 0;
1645                   we_reg <= 0;
1646                   state <= state + 1;  // S_CF_DETECT
1647                end
1648                S_CF_DETECT: begin  // Detect the presence of CF Device
1649                   if (rw_ready) begin
1650                      addr_reg <= 6'h02;  // Read STATUSREG[15:0]
1651                      re_reg <= 1;
1652                      we_reg <= 0;
1653                      state <= state + 1;  // S_CF_DETECT_1
1654                   end
1655                end
1656                S_CF_DETECT_1: begin
1657                   re_reg <= 0;
1658                   we_reg <= 0;
1659                   if (rw_nd) begin
1660                      if (rw_dout[4])
1661                         state <= rtn_state_CF_detect;
1662                      else begin
1663                         state <= state + 1;  // S_CF_DETECT_FAIL
1664                         CF_detect_reg <= 0;
1665                         LBA_max_reg <= 0;
1666                         ready_reg <= 0;
1667                      end
1668                   end
1669                end
1670                S_CF_DETECT_FAIL: begin  // CF Device not detected, wait until then
1671                   if (rw_ready) begin
1672                      addr_reg <= 6'h02;  // Read STATUSREG[15:0]
1673                      re_reg <= 1;
1674                      we_reg <= 0;
1675                      state <= state + 1;  // S_CF_DETECT_FAIL_1
1676                   end
1677                end
1678                S_CF_DETECT_FAIL_1: begin
1679                   re_reg <= 0;
1680                   we_reg <= 0;
1681                   if (rw_nd) begin
1682                      if (rw_dout[4]) begin
1683                         op_reg <= OP_IDENTIFY;
1684                         state <= S_CHK_RDY;
1685                      end
1686                      else begin
1687                         state <= S_FORCE_LOCK;  // S_FORCE_LOCK;
1688                         CF_detect_reg <= 0;
1689                         LBA_max_reg <= 0;
1690                         ready_reg <= 0;
1691                      end
1692                   end
1693                end
1694             endcase
```

```
1695            end
1696         endmodule
1697
1698         ////////////////////////////////////////////////////////////////////////////////
1699         ////  Low-Level Interface Modules
1700         ////////////////////////////////////////////////////////////////////////////////
1701
1702         // Provides low-level interface for making reads and writes to MPU Interface.
1703         // Ensures signal timing constraints are met.
1704         // Ready signal is asserted one cycle BEFORE r/w request can be sent.
1705         // MUST have at least one clock of no r/w requests between successive r/w requests.
1706         module RW_Interface(
1707            input clk_27mhz,      // 27MHz clock
1708            input reset,          // Synchronous reset
1709            // Low-Level I/O
1710            input [6:0] addr,     // Address signal for r/w
1711            input [15:0] din,     // Data input for writes
1712            input we,             // Write enable
1713            input re,             // Read enable
1714            output [15:0] dout,   // Data output for reads
1715            output nd,            // New data available at output
1716            output ready,         // Command ready signal
1717            // SystemACE ports
1718            inout [15:0] systemace_data,     // SystemACE R/W data
1719            output [6:0] systemace_address,  // SystemACE R/W address
1720            output systemace_ce_b,           // SystemACE chip enable (Active Low)
1721            output systemace_we_b,           // SystemACE write enable (Active Low)
1722            output systemace_oe_b            // SystemACE output enable (Active Low)
1723            );
1724
1725            // Instantiate state
1726            reg [1:0] state;
1727
1728            // Define state parameters
1729            parameter S_IDLE    = 2'd0;
1730            parameter S_READ_1  = 2'd1;
1731            parameter S_READ_2  = 2'd2;
1732            parameter S_WRITE_1 = 2'd3;
1733
1734
1735            // Instantiate SystemACE output registers
1736            reg [15:0] systemace_din_reg;
1737            reg [6:0] systemace_address_reg;
1738            reg systemace_ce_b_reg;
1739            reg systemace_we_b_reg;
1740            reg systemace_oe_b_reg;
1741
1742            // Instantiate RW Interface input register
1743            reg [15:0] din_reg;
1744
1745            // Instantiate RW Interface output registers
1746            reg ready_reg;
1747
1748            // Assign SystemACE outputs
1749            assign systemace_data = systemace_din_reg;
1750            assign systemace_address = systemace_address_reg;
1751            assign systemace_ce_b = systemace_ce_b_reg;
1752            assign systemace_we_b = systemace_we_b_reg;
1753            assign systemace_oe_b = systemace_oe_b_reg;
1754
1755            // Assign RW Interface outputs
```

```
1756          assign dout = systemace_data;
1757          assign nd = ~systemace_oe_b_reg;
1758          assign ready = ready_reg;
1759
1760          // Manage state transitions
1761          always @(posedge clk_27mhz) begin
1762             if (reset) begin
1763                state <= S_IDLE;
1764                // Reset RW Interface signals
1765                din_reg <= 0;
1766                ready_reg <= 0;
1767                // Reset SystemACE signals
1768                systemace_din_reg <= 16'hZZZZ;
1769                systemace_address_reg <= 0;
1770                systemace_ce_b_reg <= 1;
1771                systemace_we_b_reg <= 1;
1772                systemace_oe_b_reg <= 1;
1773             end
1774             else begin
1775                case (state)
1776                   S_IDLE: begin
1777                      // Set definite RW Interface signals
1778                      din_reg <= din;
1779                      // Set definite SystemACE signals
1780                      systemace_address_reg <= addr;
1781                      systemace_din_reg <= 16'hZZZZ;
1782                      systemace_we_b_reg <= 1;
1783                      systemace_oe_b_reg <= 1;
1784                      // Manage R/W
1785                      if (re) begin
1786                         state <= S_READ_1;
1787                         ready_reg <= 0;
1788                         systemace_ce_b_reg <= 0;
1789                      end
1790                      else if (we) begin
1791                         state <= S_WRITE_1;
1792                         ready_reg <= 1;
1793                         systemace_ce_b_reg <= 0;
1794                      end
1795                      else begin
1796                         state <= S_IDLE;
1797                         ready_reg <= 1;
1798                         systemace_ce_b_reg <= 1;
1799                      end
1800                   end
1801                   S_READ_1: begin
1802                      state <= S_READ_2;
1803                      // Set changed RW Interface signals
1804                      ready_reg <= 1;
1805                   end
1806                   S_READ_2: begin
1807                      state <= S_IDLE;
1808                      // Set changed SystemACE signals
1809                      systemace_oe_b_reg <= 0;
1810                   end
1811                   S_WRITE_1: begin
1812                      state <= S_IDLE;
1813                      // Set changed SystemAce signals
1814                      systemace_din_reg <= din_reg;
1815                      systemace_we_b_reg <= 0;
1816                   end
```

```
1817                    // Since all state encodings used, no need for default.
1818                endcase
1819            end
1820        end
1821    endmodule
1822
1823    // Provides low-level interface for reading to and writing from ZBT.
1824    // Ensures timing constraints are met.
1825    // Command ready set BEFORE a command can be sent.
1826    // At least one clock cycle between consecutive commands.
1827    // OPCODES:
1828    //     o OP_IDLE:   2'b00
1829    //     o OP_READ:   2'b01
1830    //     o OP_WRITE:  2'b10
1831    // SPECIAL NOTES:
1832    //     o Partial (byte) write enable (bwe) is Active High
1833
1834    module ZBT_Interface(
1835        input clk_27mhz,        // 27MHz clock
1836        input reset,            // Synchronous reset
1837        // Low-Level I/O
1838        input [18:0] addr,       // Address signal for r/w
1839        input [35:0] din,       // Data input for writes
1840        input [1:0] op,         // Operation Code
1841        input [3:0] bwe,        // Partial (byte) writes
1842        output [35:0] dout,     // Data output for reads
1843        output nd,              // New data available at output
1844        output ready,             // Command ready signal
1845        // ZBT ports
1846        inout  [35:0] ram_data,
1847        output [18:0] ram_address,
1848        output ram_we_b,
1849        output [3:0] ram_bwe_b
1850        );
1851
1852        // Define Operation parameters (OPCODE)
1853        parameter OP_IDLE = 2'b00;
1854        parameter OP_READ = 2'b01;
1855        parameter OP_WRITE = 2'b10;
1856
1857        // Instantiate ZBT output registers
1858        reg [35:0] ram_data_reg;  // Tri-state Buffer
1859        reg [18:0] ram_address_reg;
1860        reg ram_we_b_reg;
1861        reg [3:0] ram_bwe_b_reg;
1862
1863        // Assign ZBT outputs
1864        assign ram_data = ram_data_reg;
1865        assign ram_address = ram_address_reg;
1866        assign ram_we_b = ram_we_b_reg;
1867        assign ram_bwe_b = ram_bwe_b_reg;
1868
1869        // Instantiate Low-Level Interface output registers
1870        reg [35:0] dout_reg;
1871        reg nd_reg, ready_reg;
1872
1873        // Assign Low-Level Interface output registers
1874        assign nd = nd_reg;
1875        assign dout = dout_reg;
1876        assign ready = ready_reg;
1877
```

```
1878         // Instantiate internal registers
1879         reg nd_reg_delay, nd_reg_double_delay, nd_reg_triple_delay;
1880         reg [35:0] ram_data_reg_delay;
1881         reg ram_we_b_reg_delay;
1882
1883         // Instantiate state
1884         reg [1:0] state;
1885
1886         // Define state paramters
1887         parameter S_IDLE = 2'b00;
1888         parameter S_WR   = 2'b01;
1889         parameter S_WR_1 = 2'b10;
1890         parameter S_RD   = 2'b11;
1891
1892         // Manage state transitions
1893         always @(posedge clk_27mhz) begin
1894            if (reset) begin
1895               // Set ZBT signals
1896               ram_data_reg <= 36'hZZZZZZZZZ;
1897               ram_address_reg <= 0;
1898               ram_we_b_reg <= 1'b1;
1899               ram_bwe_b_reg <= 4'hF;
1900               // Set Low-Level signals
1901               nd_reg <= 0;
1902               ready_reg <= 0;
1903               dout_reg <= 0;
1904               // Set internal signals
1905               nd_reg_delay <= 0;
1906               nd_reg_double_delay <= 0;
1907               nd_reg_triple_delay <= 0;
1908               ram_data_reg_delay <= 0;
1909               ram_we_b_reg_delay <= 1;
1910               // Set initial state
1911               state <= S_IDLE;
1912            end
1913            else begin
1914               ram_we_b_reg_delay <= ram_we_b_reg;
1915               ram_data_reg <= (ram_we_b_reg_delay) ? 36'hZZZZZZZZZ: ram_data_reg_delay;
1916               nd_reg_double_delay <= nd_reg_delay;
1917               nd_reg_triple_delay <= nd_reg_double_delay;
1918               nd_reg <= nd_reg_triple_delay;
1919               if (nd_reg_triple_delay) dout_reg <= ram_data;
1920               case (state)
1921                  S_IDLE: begin
1922                     ram_address_reg <= addr;
1923                     ram_data_reg_delay <= din;
1924                     case (op)
1925                        OP_READ: begin
1926                           ready_reg <= 1;
1927                           state <= S_RD;
1928                           nd_reg_delay <= 1;
1929                        end
1930                        OP_WRITE: begin
1931                           ready_reg <= 0;
1932                           state <= S_WR;
1933                           ram_we_b_reg <= 0;
1934                           ram_bwe_b_reg <= ~bwe;
1935                           nd_reg_delay <= 0;
1936                        end
1937                        default: begin
1938                           ready_reg <= 1;  // OP_IDLE
```

```
1939                            nd_reg_delay <= 0;
1940                         end
1941                      endcase
1942                   end
1943                   S_WR: begin
1944                      state <= state + 1;  // S_WR_1
1945                      ready_reg <= 1;
1946                   end
1947                   S_WR_1: begin
1948                      state <= S_IDLE;
1949                      ram_we_b_reg <= 1;
1950                      ram_bwe_b_reg <= 4'hF;
1951                   end
1952                   S_RD: begin
1953                      state <= S_IDLE;
1954                      nd_reg_delay <= 0;
1955                   end
1956                endcase
1957             end
1958          end
1959       endmodule
1960
```

```
1
2        ////////////////////////////////////////////////////////////////////////////////
3        //
4        // 6.111 FPGA Labkit -- 16 characer ASCII string display
5        //
6        //
7        // File:   display_string.v
8        // Date:   24-Sep-05
9        // Author: I. Chuang <ichuang@mit.edu> & Sachin Shinde (edits)
10       //
11       // Based on Nathan Ickes' hex display code
12       //
13       // 28-Nov-2006 CJT: fixed race condition between CE and RS
14       //
15       // This module drives the labkit hex displays and shows the value of
16       // 8 ascii bytes as characters on the displays.
17       //
18       // Uses the Jae's ascii2dots module
19       //
20       // Inputs:
21       //
22       //   reset        - active high
23       //   clock_27mhz - the synchronous clock
24       //   string_data - 128 bits; each 8 bits gives an ASCII coded character
25       //
26       // Outputs:
27       //
28       //    disp_*      - display lines used in the 6.111 labkit (rev 003 & 004)
29       //
30       ////////////////////////////////////////////////////////////////////////////////
31
32       module display_string (reset, clock_27mhz, string_data,
33             disp_blank, disp_clock, disp_rs, disp_ce_b,
34             disp_reset_b, disp_data_out);
35
36          input reset, clock_27mhz;      // clock and reset (active high reset)
37          input [16*8-1:0] string_data; // 8 ascii bytes to display
38
39          output disp_blank, disp_clock, disp_data_out, disp_rs, disp_ce_b,
40            disp_reset_b;
41
42          reg disp_data_out, disp_rs, disp_ce_b, disp_reset_b;
43
44          ////////////////////////////////////////////////////////////////////////////
45          //
46          // Display Clock
47          //
48          // Generate a 500kHz clock for driving the displays.
49          //
50          ////////////////////////////////////////////////////////////////////////////
51
52          reg [4:0] count;
53          reg [7:0] reset_count;
54          reg clock;
55          wire dreset;
56
57          always @(posedge clock_27mhz)
58            begin
59          if (reset)
60            begin
61               count = 0;
```

```
 62                 clock = 0;
 63              end
 64            else if (count == 26)
 65              begin
 66                 clock = ~clock;
 67                 count = 5'h00;
 68              end
 69            else
 70              count = count+1;
 71              end
 72
 73            always @(posedge clock_27mhz)
 74              if (reset)
 75                reset_count <= 100;
 76              else
 77                reset_count <= (reset_count==0) ? 0 : reset_count-1;
 78
 79            assign dreset = (reset_count != 0);
 80
 81            assign disp_clock = ~clock;
 82
 83            ////////////////////////////////////////////////////////////////////////////
 84            //
 85            // Display State Machine
 86            //
 87            ////////////////////////////////////////////////////////////////////////////
 88
 89            reg [7:0] state;     // FSM state
 90            reg [9:0] dot_index;    // index to current dot being clocked out
 91            reg [31:0] control;     // control register
 92            reg [3:0] char_index;   // index of current character
 93            wire [39:0] dots;    // dots for a single digit
 94            reg [39:0] rdots;    // pipelined dots
 95            reg [7:0] ascii;     // ascii value of current character
 96
 97            assign disp_blank = 1'b0; // low <= not blanked
 98
 99            always @(posedge clock)
100              if (dreset)
101                begin
102              state <= 0;
103              dot_index <= 0;
104              control <= 32'h7F7F7F7F;
105                end
106              else
107                casex (state)
108             8'h00:
109               begin
110                  // Reset displays
111                  disp_data_out <= 1'b0;
112                  disp_rs <= 1'b0; // dot register
113                  disp_ce_b <= 1'b1;
114                  disp_reset_b <= 1'b0;
115                  dot_index <= 0;
116                  state <= state+1;
117               end
118
119             8'h01:
120               begin
121                  // End reset
122                  disp_reset_b <= 1'b1;
```

```
123                 state <= state+1;
124               end
125
126             8'h02:
127               begin
128                 // Initialize dot register (set all dots to zero)
129                 disp_ce_b <= 1'b0;
130                 disp_data_out <= 1'b0; // dot_index[0];
131                 if (dot_index == 639)
132               state <= state+1;
133                 else
134               dot_index <= dot_index+1;
135               end
136
137             8'h03:
138               begin
139                 // Latch dot data
140                 disp_ce_b <= 1'b1;
141                 dot_index <= 31;      // re-purpose to init ctrl reg
142                 state <= state+1;
143                 disp_rs <= 1'b1; // Select the control register
144               end
145
146             8'h04:
147               begin
148                 // Setup the control register
149                 disp_ce_b <= 1'b0;
150                 disp_data_out <= control[31];
151                 control <= {control[30:0], 1'b0};    // shift left
152                 if (dot_index == 0)
153               state <= state+1;
154                 else
155               dot_index <= dot_index-1;
156                 char_index <= 15;     // set this up early for pipeline
157               end
158
159             8'h05:
160               begin
161                 // Latch the control register data / dot data
162                 disp_ce_b <= 1'b1;
163                 dot_index <= 39;      // init for single char
164                 rdots <= dots;     // store dots of char 15
165                 char_index <= 14;     // ready for next char
166                 state <= state+1;
167                 disp_rs <= 1'b0;         // Select the dot register
168               end
169
170             8'h06:
171               begin
172                 // Load the user's dot data into the dot reg, char by char
173                 disp_ce_b <= 1'b0;
174                 disp_data_out <= rdots[dot_index]; // dot data from msb
175                 if (dot_index == 0)
176                   if (char_index == 15)
177                     state <= 5;          // all done, latch data
178               else
179               begin
180                 char_index <= char_index - 1;   // goto next char
181                 dot_index <= 39;
182                 rdots <= dots;      // latch in next char dots
183               end
```

```
184                     else
185                  dot_index <= dot_index-1;  // else loop thru all dots
186                  end
187
188                   endcase
189
190               // combinatorial logic to generate dots for current character
191               // this mux, and the ascii table lookup, are slow, so note that
192               // this is pipelined by one display clock stage in the always
193               // loop above.
194
195               always @(string_data or char_index)
196                 case (char_index)
197                   4'h0: ascii = string_data[7:0];
198                   4'h1: ascii = string_data[7+1*8:1*8];
199                   4'h2: ascii = string_data[7+2*8:2*8];
200                   4'h3: ascii = string_data[7+3*8:3*8];
201                   4'h4: ascii = string_data[7+4*8:4*8];
202                   4'h5: ascii = string_data[7+5*8:5*8];
203                   4'h6: ascii = string_data[7+6*8:6*8];
204                   4'h7: ascii = string_data[7+7*8:7*8];
205                   4'h8: ascii = string_data[7+8*8:8*8];
206                   4'h9: ascii = string_data[7+9*8:9*8];
207                   4'hA: ascii = string_data[7+10*8:10*8];
208                   4'hB: ascii = string_data[7+11*8:11*8];
209                   4'hC: ascii = string_data[7+12*8:12*8];
210                   4'hD: ascii = string_data[7+13*8:13*8];
211                   4'hE: ascii = string_data[7+14*8:14*8];
212                   4'hF: ascii = string_data[7+15*8:15*8];
213                 endcase
214
215               ascii2dots a2d(ascii,dots);
216
217            endmodule
218
219            ///////////////////////////////////////////////////////////////////////////////
220            // Display font dots generation from ASCII code
221
222            module ascii2dots(ascii_in,char_dots);
223
224            input [7:0] ascii_in;
225            output [39:0] char_dots;
226
227              ///////////////////////////////////////////////////////////////////////////
228              // ROM: ASCII-->DOTS conversion
229              ///////////////////////////////////////////////////////////////////////////
230               reg [39:0] char_dots;
231
232               always @(ascii_in)
233                case(ascii_in)
234                  8'h10:   char_dots = 40'b11111111_11100001_11011110_11100001_11111111; //  16
           0_INVERTED
235                  8'h11:   char_dots = 40'b11111111_11011101_11000000_11011111_11111111; //  17
           1_INVERTED
236                  8'h12:   char_dots = 40'b11011101_11001110_11010110_11011001_11111111; //  18
           2_INVERTED
237                  8'h13:   char_dots = 40'b11101110_11011010_11011010_11100100_11111111; //  19
           3_INVERTED
238                  8'h14:   char_dots = 40'b11110011_11110101_11000000_11110111_11111111; //  20
           4_INVERTED
239                  8'h15:   char_dots = 40'b11101000_11011010_11011010_11100110_11111111; //  21
```

```
           5_INVERTED
240              8'h16:    char_dots = 40'b11100001_11011010_11011010_11100111_11111111; //  22
           6_INVERTED
241              8'h17:    char_dots = 40'b11111110_11001110_11110010_11111100_11111111; //  23
           7_INVERTED
242              8'h18:    char_dots = 40'b11100101_11011010_11011010_11100101_11111111; //  24
           8_INVERTED
243              8'h19:    char_dots = 40'b11111001_11010110_11010110_11100001_11111111; //  25
           9_INVERTED
244              8'h20:    char_dots = 40'b00000000_00000000_00000000_00000000_00000000; //  32 ' '
245              8'h21:    char_dots = 40'b00000000_00000000_00101111_00000000_00000000; //  33  !
246              8'h22:    char_dots = 40'b00000000_00000111_00000000_00000111_00000000; //  34 "
247              8'h23:    char_dots = 40'b00010100_00111110_00010100_00111110_00010100; //  35  #
248              8'h24:    char_dots = 40'b00000100_00101010_00111110_00101010_00010000; //  36  $
249              8'h25:    char_dots = 40'b00010011_00001000_00000100_00110010_00000000; //  37  %
250              8'h26:    char_dots = 40'b00010100_00101010_00010100_00100000_00000000; //  38  &
251              8'h27:    char_dots = 40'b00000000_00000000_00000111_00000000_00000000; //  39 '
252              8'h28:    char_dots = 40'b00000000_00011110_00100001_00000000_00000000; //  40  (
253              8'h29:    char_dots = 40'b00000000_00100001_00011110_00000000_00000000; //  41  )
254              8'h2A:    char_dots = 40'b00000000_00101010_00011100_00101010_00000000; //  42  *
255              8'h2B:    char_dots = 40'b00001000_00001000_00111110_00001000_00001000; //  43   +
256              8'h2C:    char_dots = 40'b00000000_01000000_00110000_00010000_00000000; //  44 ,
257              8'h2D:    char_dots = 40'b00001000_00001000_00001000_00001000_00000000; //  45  -
258              8'h2E:    char_dots = 40'b00000000_00110000_00110000_00000000_00000000; //  46  .
259              8'h2F:    char_dots = 40'b00010000_00001000_00000100_00000010_00000000; //  47  /
260              8'h30:    char_dots = 40'b00000000_00011110_00100001_00011110_00000000; //  48  0
           --> 17
261              8'h31:    char_dots = 40'b00000000_00100010_00111111_00100000_00000000; //  49  1
262              8'h32:    char_dots = 40'b00100010_00110001_00101001_00100110_00000000; //  50  2
263              8'h33:    char_dots = 40'b00010001_00100101_00100101_00011011_00000000; //  51  3
264              8'h34:    char_dots = 40'b00001100_00001010_00111111_00001000_00000000; //  52  4
265              8'h35:    char_dots = 40'b00010111_00100101_00100101_00011001_00000000; //  53  5
266              8'h36:    char_dots = 40'b00011110_00100101_00100101_00011000_00000000; //  54  6
267              8'h37:    char_dots = 40'b00000001_00110001_00001101_00000011_00000000; //  55  7
268              8'h38:    char_dots = 40'b00011010_00100101_00100101_00011010_00000000; //  56  8
269              8'h39:    char_dots = 40'b00000110_00101001_00101001_00011110_00000000; //  57  9
270              8'h3A:    char_dots = 40'b00000000_00110110_00110110_00000000_00000000; //  58  :
           --> 27
271              8'h3B:    char_dots = 40'b01000000_00110110_00010110_00000000_00000000; //  59  ;
272              8'h3C:    char_dots = 40'b00000000_00001000_00010100_00100010_00000000; //  60  <
273              8'h3D:    char_dots = 40'b00010100_00010100_00010100_00010100_00000000; //  61  =
274              8'h3E:    char_dots = 40'b00000000_00100010_00010100_00001000_00000000; //  62  >
275              8'h3F:    char_dots = 40'b00000000_00000010_00101001_00000110_00000000; //  63  ?
276              8'h40:    char_dots = 40'b00011110_00100001_00101101_00001110_00000000; //  64  @
277              8'h41:    char_dots = 40'b00111110_00001001_00001001_00111110_00000000; //  65  A
           --> 34
278              8'h42:    char_dots = 40'b00111111_00100101_00100101_00011010_00000000; //  66  B
279              8'h43:    char_dots = 40'b00011110_00100001_00100001_00010010_00000000; //  67  C
280              8'h44:    char_dots = 40'b00111111_00100001_00100001_00011110_00000000; //  68  D
281              8'h45:    char_dots = 40'b00111111_00100101_00100101_00100001_00000000; //  69  E
282              8'h46:    char_dots = 40'b00111111_00000101_00000101_00000001_00000000; //  70  F
283              8'h47:    char_dots = 40'b00011110_00100001_00101001_00111010_00000000; //  71  G
284              8'h48:    char_dots = 40'b00111111_00000100_00000100_00111111_00000000; //  72  H
285              8'h49:    char_dots = 40'b00000000_00100001_00111111_00100001_00000000; //  73  I
286              8'h4A:    char_dots = 40'b00010000_00100000_00100000_00011111_00000000; //  74  J
287              8'h4B:    char_dots = 40'b00111111_00001100_00010010_00100001_00000000; //  75  K
288              8'h4C:    char_dots = 40'b00111111_00100000_00100000_00100000_00000000; //  76  L
289              8'h4D:    char_dots = 40'b00111111_00000110_00000110_00111111_00000000; //  77  M
290              8'h4E:    char_dots = 40'b00111111_00000110_00011000_00111111_00000000; //  78  N
291              8'h4F:    char_dots = 40'b00011110_00100001_00100001_00011110_00000000; //  79  O
292              8'h50:    char_dots = 40'b00111111_00001001_00001001_00000110_00000000; //  80  P
```

```
293             8'h51:   char_dots = 40'b00011110_00110001_00100001_01011110_00000000; //  81  Q
294             8'h52:   char_dots = 40'b00111111_00001001_00011001_00100110_00000000; //  82  R
295             8'h53:   char_dots = 40'b00010010_00100101_00101001_00010010_00000000; //  83  S
296             8'h54:   char_dots = 40'b00000000_00000001_00111111_00000001_00000000; //  84  T
297             8'h55:   char_dots = 40'b00011111_00100000_00100000_00011111_00000000; //  85  U
298             8'h56:   char_dots = 40'b00001111_00110000_00110000_00001111_00000000; //  86  V
299             8'h57:   char_dots = 40'b00111111_00011000_00011000_00111111_00000000; //  87  W
300             8'h58:   char_dots = 40'b00110011_00001100_00001100_00110011_00000000; //  88  X
301             8'h59:   char_dots = 40'b00000000_00000111_00111000_00000111_00000000; //  89  Y
302             8'h5A:   char_dots = 40'b00110001_00101001_00100101_00100011_00000000; //  90  Z
        --> 59
303             8'h5B:   char_dots = 40'b00000000_00111111_00100001_00100001_00000000; //  91  [
304             8'h5C:   char_dots = 40'b00000010_00000100_00001000_00010000_00000000; //  92  \
305             8'h5D:   char_dots = 40'b00000000_00100001_00100001_00111111_00000000; //  93  ]
306             8'h5E:   char_dots = 40'b00000000_00000010_00000001_00000010_00000000; //  94  ^
307             8'h5F:   char_dots = 40'b00100000_00100000_00100000_00100000_00000000; //  95  _
308             8'h60:   char_dots = 40'b00000000_00000001_00000010_00000000_00000000; //  96  '
309             8'h61:   char_dots = 40'b00011000_00100100_00010100_00111100_00000000; //  97  a
        --> 66
310             8'h62:   char_dots = 40'b00111111_00100100_00100100_00011000_00000000; //  98  b
311             8'h63:   char_dots = 40'b00011000_00100100_00100100_00000000_00000000; //  99  c
312             8'h64:   char_dots = 40'b00011000_00100100_00100100_00111111_00000000; // 100  d
313             8'h65:   char_dots = 40'b00011000_00110100_00101100_00001000_00000000; // 101  e
314             8'h66:   char_dots = 40'b00001000_00111110_00001001_00000010_00000000; // 102  f
315             8'h67:   char_dots = 40'b00101000_01010100_01010100_01001100_00000000; // 103  g
316             8'h68:   char_dots = 40'b00111111_00000100_00000100_00111000_00000000; // 104  h
317             8'h69:   char_dots = 40'b00000000_00100100_00111101_00100000_00000000; // 105  i
318             8'h6A:   char_dots = 40'b00000000_00100000_01000000_00111101_00000000; // 106  j
319             8'h6B:   char_dots = 40'b00111111_00001000_00010100_00100000_00000000; // 107  k
320             8'h6C:   char_dots = 40'b00000000_00100001_00111111_00100000_00000000; // 108  l
321             8'h6D:   char_dots = 40'b00111100_00001000_00001100_00111000_00000000; // 109  m
322             8'h6E:   char_dots = 40'b00111100_00000100_00000100_00111000_00000000; // 110  n
323             8'h6F:   char_dots = 40'b00011000_00100100_00100100_00011000_00000000; // 111  o
324             8'h70:   char_dots = 40'b01111100_00100100_00100100_00011000_00000000; // 112  p
325             8'h71:   char_dots = 40'b00011000_00100100_00100100_01111100_00000000; // 113  q
326             8'h72:   char_dots = 40'b00111100_00000100_00000100_00001000_00000000; // 114  r
327             8'h73:   char_dots = 40'b00101000_00101100_00110100_00010100_00000000; // 115  s
328             8'h74:   char_dots = 40'b00000100_00011111_00100100_00100000_00000000; // 116  t
329             8'h75:   char_dots = 40'b00011100_00100000_00100000_00111100_00000000; // 117  u
330             8'h76:   char_dots = 40'b00000000_00011100_00100000_00011100_00000000; // 118  v
331             8'h77:   char_dots = 40'b00111100_00110000_00110000_00111100_00000000; // 119  w
332             8'h78:   char_dots = 40'b00100100_00011000_00011000_00100100_00000000; // 120  x
333             8'h79:   char_dots = 40'b00001100_01010000_00100000_00011100_00000000; // 121  y
334             8'h7A:   char_dots = 40'b00100100_00110100_00101100_00100100_00000000; // 122  z
        --> 91
335             8'h7B:   char_dots = 40'b00000000_00000100_00011110_00100001_00000000; // 123  {
336             8'h7C:   char_dots = 40'b00000000_00000000_00111111_00000000_00000000; // 124  |
337             8'h7D:   char_dots = 40'b00000000_00100001_00011110_00000100_00000000; // 125  }
338             8'h7E:   char_dots = 40'b00000010_00000001_00000010_00000001_00000000; // 126  ~
        --> 95
339             default: char_dots = 40'b01000001_01000001_01000001_01000001_01000001;
340           endcase
341
342       endmodule
343
344
```

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
3   // Company:
4   // Engineer:       Sachin Shinde
5   //
6   // Create Date:    08:07:32 11/22/2012
7   // Design Name:
8   // Module Name:    Text_Scroller
9   // Project Name:
10  // Target Devices:
11  // Tool versions:
12  // Description:    Takes ASCII data and displays it as string data on the screen.
13  //                 Use the ready signal to indicate when data is being sent, with
14  //                 sent data starting at the time when the ready signal goes
15  //                 active high and ending at the time when ready goes inactive low.
16  //
17  //                 Synchronous reset blanks the screen. In this implementation,
18  //                 a single block of BRAM is used for the character memory,
19  //                 putting a maximum on message length of 2048 characters. Note
20  //                 that all parameter counts must be greater than 20.
21  //
22  //
23  // Dependencies:
24  //
25  // Revision:
26  // Revision 0.01 - File Created
27  // Additional Comments:
28  //
29  //////////////////////////////////////////////////////////////////////////////////
30  module Text_Scroller #(
31     parameter SCROLL_SPEED_CNT = 9000000,   // Counter limit before scrolling by one
32     parameter SCROLL_BEGIN_CNT = 27000000,  // Counter limit before starting to scroll
33     parameter SCROLL_END_CNT = 27000000     // Counter limit before moving back to start
34     )(
35     input clk,
36     input reset,
37     // Data In
38     input [7:0] ascii_data,
39     input ascii_data_ready,
40     // Data Out
41     output [127:0] string_data
42  // ,// DEBUG
43  // output wr_en_DEBUG,
44  // output [7:0] wr_data_DEBUG,
45  // output [10:0] wr_addr_DEBUG,
46  // output cntr_DEBUG,
47  // output set_disp_DEBUG,
48  // output [3:0] rel_pos_DEBUG,
49  // output [10:0] rd_addr_DEBUG,
50  // output [7:0] rd_data_DEBUG
51  // // END DEBUG
52     );
53
54     // Determine parameters
55     parameter SCROLL_SPEED_LMT = SCROLL_SPEED_CNT - 1;
56     parameter SCROLL_BEGIN_LMT = SCROLL_BEGIN_CNT - 1;
57     parameter SCROLL_END_LMT = SCROLL_END_CNT - 1;
58
59     // Instantiate state vars
60     reg [10:0] msg_length;          // Really message length minus one
61     reg [7:0] string_data_reg [0:15];   // Storage for string data output
```

```
 62
 63          // Assign output
 64          assign string_data = {string_data_reg[0],
 65                                string_data_reg[1],
 66                                string_data_reg[2],
 67                                string_data_reg[3],
 68                                string_data_reg[4],
 69                                string_data_reg[5],
 70                                string_data_reg[6],
 71                                string_data_reg[7],
 72                                string_data_reg[8],
 73                                string_data_reg[9],
 74                                string_data_reg[10],
 75                                string_data_reg[11],
 76                                string_data_reg[12],
 77                                string_data_reg[13],
 78                                string_data_reg[14],
 79                                string_data_reg[15]
 80                                };
 81
 82          // Instantiate Dual-Port BRAM for storing ASCII chars
 83          reg [10:0] wr_addr, rd_addr;
 84          reg wr_en;
 85          reg [7:0] wr_data;
 86          wire [7:0] rd_data;
 87          UI_Text_Scroller_BRAM UI_TEXT_SCROLLER_BRAM_1(
 88             .clka(clk),
 89             .clkb(clk),
 90             .addra(wr_addr),
 91             .addrb(rd_addr),
 92             .dina(wr_data),
 93             .doutb(rd_data),
 94             .wea(wr_en)
 95          );
 96
 97          // Assign write data
 98          always @(posedge clk)
 99             wr_data <= ascii_data;
100
101          // Manage writes
102          always @(posedge clk) begin
103             if (reset) begin
104                wr_en <= 1'b0;
105                msg_length <= 11'd0;
106                wr_addr <= 11'd0;
107             end
108             else begin
109                wr_en <= ascii_data_ready;
110                if (wr_en) begin
111                   wr_addr <= wr_addr + 1;
112                   msg_length <= wr_addr;
113                end
114                else
115                   wr_addr <= 11'd0;
116             end
117          end
118
119          // Create variable limit counter
120          reg [24:0] cntr_lim, cntr_val;  // limit to 33554431
121          reg cntr;
122          wire cntr_reset, manual_cntr_reset;
```

```
123         assign cntr_reset = reset | manual_cntr_reset;
124         assign manual_cntr_reset = ascii_data_ready & !wr_addr[10:4];
125         always @(posedge clk) begin
126            if (cntr_reset) begin
127               cntr_val <= 0;
128               cntr <= 1'b0;
129            end
130            else begin
131               cntr_val <= (cntr_val == cntr_lim) ? 0: cntr_val + 1;
132               cntr <= !cntr_val;
133            end
134         end
135
136         // Manage display chars position and counter timing limit
137         reg [10:0] disp_pos;
138         wire [10:0] disp_pos_last;
139         assign disp_pos_last = (msg_length[10:4]) ? (msg_length - 15): 0;
140         always @(posedge clk) begin
141            if (cntr_reset) begin
142               disp_pos <= 0;
143               cntr_lim <= SCROLL_BEGIN_LMT;
144            end
145            else if (cntr) begin
146               disp_pos <= (disp_pos == disp_pos_last) ? 0: (disp_pos + 1);
147               cntr_lim <= (!disp_pos) ? SCROLL_BEGIN_LMT:
148                           (disp_pos == disp_pos_last) ? SCROLL_END_LMT:
149                           SCROLL_SPEED_LMT;
150            end
151         end
152
153         // Update display chars when counter goes active high
154         reg [3:0] rel_pos;
155         reg set_disp, set_disp_delay;
156         always @(posedge clk) begin
157            if (cntr_reset) begin
158               set_disp <= 1'b0;
159               set_disp_delay <= 1'b0;
160               // Blank on reset
161               string_data_reg[0] <= 8'h20;
162               string_data_reg[1] <= 8'h20;
163               string_data_reg[2] <= 8'h20;
164               string_data_reg[3] <= 8'h20;
165               string_data_reg[4] <= 8'h20;
166               string_data_reg[5] <= 8'h20;
167               string_data_reg[6] <= 8'h20;
168               string_data_reg[7] <= 8'h20;
169               string_data_reg[8] <= 8'h20;
170               string_data_reg[9] <= 8'h20;
171               string_data_reg[10] <= 8'h20;
172               string_data_reg[11] <= 8'h20;
173               string_data_reg[12] <= 8'h20;
174               string_data_reg[13] <= 8'h20;
175               string_data_reg[14] <= 8'h20;
176               string_data_reg[15] <= 8'h20;
177            end
178            else if (cntr) begin
179               rd_addr <= disp_pos;
180               rel_pos <= 4'd0;
181               set_disp <= 1'b1;
182            end
183            else if (set_disp) begin  // need delay for BRAM one-cycle read latency
```

```
184                 set_disp <= 1'b0;
185                 set_disp_delay <= 1'b1;
186                 rd_addr <= rd_addr + 1;
187              end
188              else if (set_disp_delay) begin
189                 rd_addr <= rd_addr + 1;
190                 rel_pos <= rel_pos + 4'd1;
191                 set_disp_delay <= ~(&rel_pos);
192                 string_data_reg[rel_pos] <= (rel_pos > msg_length) ? 8'h20: rd_data;
193              end
194           end
195
196        // // DEBUG
197        // assign wr_en_DEBUG = wr_en;
198        // assign wr_data_DEBUG = wr_data;
199        // assign wr_addr_DEBUG = wr_addr;
200        // assign cntr_DEBUG = cntr;
201        // assign set_disp_DEBUG = set_disp;
202        // assign rel_pos_DEBUG = rel_pos;
203        // assign rd_addr_DEBUG = rd_addr;
204        // assign rd_data_DEBUG = rd_data;
205        // // END DEBUG
206
207        endmodule
208
```

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
3   // Company:
4   // Engineer:      Sachin Shinde
5   //
6   // Create Date:    21:07:49 11/24/2012
7   // Design Name:
8   // Module Name:    Date_Time
9   // Project Name:
10  // Target Devices:
11  // Tool versions:
12  // Description:    Keeps track of date and time, allows for setting it, and
13  //                 generates ASCII bit-stream of data when D&T needed by asserting
14  //                 a display enable bit.
15  //
16  // Dependencies:
17  //
18  // Revision:
19  // Revision 0.01 - File Created
20  // Additional Comments:
21  //
22  //////////////////////////////////////////////////////////////////////////////////
23  module Date_Time #(
24     parameter SEC_COUNT = 27_000_000  // number of clock cycles in a second
25     )(
26     input clk_27mhz,           // 27 MHz clock
27     input reset,               // Asynchronous reset
28     // Control signals
29     input set,                 // High for setting clock, low otherwise
30     input disp_en,             // Assert to generate ASCII stream when D&T changes
31     input button_up,
32     input button_down,
33     input button_left,
34     input button_right,
35     // Date & Time binary outputs
36     output [6:0] year,
37     output [3:0] month,
38     output [4:0] day,
39     output [4:0] hour,
40     output [5:0] minute,
41     output [5:0] second,
42     // ASCII output
43     output [7:0] ascii_out,
44     output ascii_out_ready,
45     // BRAM Binary-to-Decimal Lookup-Table I/O
46     output [6:0] addr,
47     input [7:0] data
48     );
49
50     // Define counter parameters
51     parameter SEC_CNTR_LMT = SEC_COUNT - 1;
52
53     // Generate one-cycle-high signals for button rises
54     reg button_up_reg, button_down_reg, button_left_reg, button_right_reg;
55     reg bu, bd, bl, br;
56     always @(posedge clk_27mhz) begin
57        if (reset) begin
58           button_up_reg <= 0;
59           button_down_reg <= 0;
60           button_left_reg <= 0;
61           button_right_reg <= 0;
```

```
 62                 bu <= 0;
 63                 bd <= 0;
 64                 bl <= 0;
 65                 br <= 0;
 66              end
 67              else begin
 68                 button_up_reg <= button_up;
 69                 button_down_reg <= button_down;
 70                 button_left_reg <= button_left;
 71                 button_right_reg <= button_right;
 72                 bu <= button_up & ~button_up_reg;
 73                 bd <= button_down & ~button_down_reg;
 74                 bl <= button_left & ~button_left_reg;
 75                 br <= button_right & ~button_right_reg;
 76              end
 77           end
 78
 79           // Keep track of previous disp_en to detect rise
 80           reg disp_en_reg;
 81           always @(posedge clk_27mhz) disp_en_reg <= (reset) ? 0: disp_en;
 82
 83           // Instantiate DT output registers
 84           reg [6:0] year_reg;
 85           reg [3:0] month_reg;
 86           reg [4:0] day_reg;
 87           reg [4:0] hour_reg;
 88           reg [5:0] minute_reg;
 89           reg [5:0] second_reg;
 90
 91           // Assign DT output
 92           assign year = year_reg;
 93           assign month = month_reg;
 94           assign day = day_reg;
 95           assign hour = hour_reg;
 96           assign minute = minute_reg;
 97           assign second = second_reg;
 98
 99           // Instantiate ASCII output registers
100           reg [7:0] ascii_out_reg;
101           reg ascii_out_ready_reg;
102
103           // Assign ASCII output
104           assign ascii_out = ascii_out_reg;
105           assign ascii_out_ready = ascii_out_ready_reg;
106
107           // Instantiate BRAM I/O regisers
108           reg [6:0] addr_reg;
109
110           // Assign output
111           assign addr = addr_reg;
112
113           // Instantiate display request register
114           reg disp_req;
115
116           // Create second counter
117           reg [24:0] sec_cnt;
118           reg sec_rdy;
119           always @(posedge clk_27mhz) begin
120              if (reset) begin
121                 sec_cnt <= 0;
122                 sec_rdy <= 0;
```

```
123             end
124             else if (sec_cnt == SEC_CNTR_LMT) begin
125                sec_cnt <= 0;
126                sec_rdy <= 1;
127             end
128             else begin
129                sec_cnt <= sec_cnt + 1;
130                sec_rdy <= 0;
131             end
132          end
133
134          // Instantiate cursor registers
135          reg [2:0] cursor_pos;
136          reg cursor_blink;
137
138          // Declare position parameters
139          parameter POS_SECOND = 0;
140          parameter POS_MINUTE = 1;
141          parameter POS_HOUR   = 2;
142          parameter POS_DAY    = 3;
143          parameter POS_MONTH  = 4;
144          parameter POS_YEAR   = 5;
145
146          // Assign number of days in month signal
147          wire [4:0] num_days_cur_month;
148          assign num_days_cur_month = (month_reg == 2) ? ((year_reg[1:0]) ? 28: 29): ((month_reg
          ]^month_reg[0]) ? 31: 30);
149
150          // Manage Date & Time
151          reg start;
152          always @(posedge clk_27mhz) begin
153             if (reset) begin
154                year_reg <= 0;
155                month_reg <= 1;
156                day_reg <= 1;
157                hour_reg <= 0;
158                minute_reg <= 0;
159                second_reg <= 0;
160                disp_req <= 0;
161                start <= 0;
162             end
163             else if (set & (bu|bd)) begin  // Manage button presses when setting
164                disp_req <= 0;
165                start <= 1;
166                if (bu) begin
167                   case (cursor_pos)
168                      POS_SECOND: second_reg <= 0;
169                      POS_MINUTE: minute_reg <= (minute_reg == 59) ? 0: minute_reg + 1;
170                      POS_HOUR: hour_reg <= (hour_reg == 23) ? 0: hour_reg + 1;
171                      POS_DAY: day_reg <= (day_reg == num_days_cur_month) ? 1: day_reg + 1;
172                      POS_MONTH: month_reg <= (month_reg == 12) ? 1: month_reg + 1;
173                      POS_YEAR: year_reg <= (year_reg == 99) ? 0: year_reg + 1;
174                   endcase
175                end
176                else begin
177                   case (cursor_pos)
178                      POS_SECOND: second_reg <= 0;
179                      POS_MINUTE: minute_reg <= (minute_reg == 0) ? 59: minute_reg - 1;
180                      POS_HOUR: hour_reg <= (hour_reg == 0) ? 23: hour_reg - 1;
181                      POS_DAY: day_reg <= (day_reg == 1) ? num_days_cur_month: day_reg - 1;
182                      POS_MONTH: month_reg <= (month_reg == 1) ? 12: month_reg - 1;
```

```
183                     POS_YEAR: year_reg <= (year_reg == 0) ? 99: year_reg - 1;
184                  endcase
185               end
186            end
187            else if (start) begin  // Correct days in month if it exceeds max
188               start <= 0;
189               disp_req <= 1;
190               if (day_reg > num_days_cur_month) day_reg <= num_days_cur_month;
191            end
192            else if (sec_rdy) begin
193               disp_req <= 1;
194               if (second_reg == 59) begin
195                  second_reg <= 0;
196                  if (minute_reg == 59) begin
197                     minute_reg <= 0;
198                     if (hour_reg == 23) begin
199                        hour_reg <= 0;
200                        if (day_reg == num_days_cur_month) begin
201                           day_reg <= 1;
202                           if (month_reg == 12) begin
203                              month_reg <= 1;
204                              if (year_reg == 99) begin
205                                 year_reg <= 0;
206                              end
207                              else
208                                 year_reg <= year_reg + 1;
209                           end
210                           else
211                              month_reg <= month_reg + 1;
212                        end
213                        else
214                           day_reg <= day_reg + 1;
215                     end
216                     else
217                        hour_reg <= hour_reg + 1;
218                  end
219                  else
220                     minute_reg <= minute_reg + 1;
221               end
222               else
223                  second_reg <= second_reg + 1;
224            end
225            else begin
226               disp_req <= 0;
227            end
228         end
229
230         // Manage cursor blinking
231         always @(posedge clk_27mhz) begin
232            if (reset | ~set)
233               cursor_blink <= 0;
234            else if (sec_rdy)
235               cursor_blink <= ~cursor_blink;
236         end
237
238         // Manage cursor position
239         always @(posedge clk_27mhz) begin
240            if (reset | ~set)
241               cursor_pos <= 0;
242            else if (bl)
243               cursor_pos <= (cursor_pos == POS_YEAR) ? POS_YEAR: cursor_pos + 1;
```

```
244             else if (br)
245                cursor_pos <= (cursor_pos == POS_SECOND) ? POS_SECOND: cursor_pos - 1;
246          end
247
248          // Instantiate state for ASCII display requests
249          reg [4:0] state;
250
251          // Declare state parameters
252          parameter S_IDLE    = 5'h00;
253          parameter S_DISP    = 5'h01;
254          parameter S_DISP_1  = 5'h02;
255          parameter S_DISP_2  = 5'h03;
256          parameter S_DISP_3  = 5'h04;
257          parameter S_DISP_4  = 5'h05;
258          parameter S_DISP_5  = 5'h06;
259          parameter S_DISP_6  = 5'h07;
260          parameter S_DISP_7  = 5'h08;
261          parameter S_DISP_8  = 5'h09;
262          parameter S_DISP_9  = 5'h0A;
263          parameter S_DISP_10 = 5'h0B;
264          parameter S_DISP_11 = 5'h0C;
265          parameter S_DISP_12 = 5'h0D;
266          parameter S_DISP_13 = 5'h0E;
267          parameter S_DISP_14 = 5'h0F;
268          parameter S_DISP_15 = 5'h10;
269          parameter S_DISP_16 = 5'h11;
270
271          // Instantiate capture registers
272          reg [25:0] DT_capture;
273          reg [2:0] cursor_pos_capture;
274          reg cursor_blink_capture;
275          reg temp;
276
277          // Manage display requests
278          always @(posedge clk_27mhz) begin
279             if (reset) begin
280                state <= 0;  // S_IDLE
281                ascii_out_reg <= 0;
282                ascii_out_ready_reg <= 0;
283                addr_reg <= 0;
284             end
285             else begin
286                case (state)
287                   S_IDLE: begin
288                      ascii_out_reg <= 0;
289                      ascii_out_ready_reg <= 0;
290                      addr_reg <= year_reg;
291                      if (disp_en & (disp_req | ~disp_en_reg)) begin
292                         state <= state + 1;  // S_DISP
293                         DT_capture <= {month_reg, day_reg, hour_reg, minute_reg, second_reg};
294                         cursor_pos_capture <= cursor_pos;
295                         cursor_blink_capture <= cursor_blink;
296                      end
297                   end
298                   S_DISP: begin
299                      temp <= ~(cursor_blink_capture & (cursor_pos_capture == 5));
300                      state <= state + 1;  // S_DISP_1
301                   end
302                   S_DISP_1: begin
303                      addr_reg <= {3'b000, DT_capture[25:22]};
304                      ascii_out_reg <= {2'b00, temp, 1'b1, data[7:4]};  // year, first digit
```

```
305                 ascii_out_ready_reg <= 1;
306                 state <= state + 1;  // S_DISP_2
307              end
308              S_DISP_2: begin
309                 ascii_out_reg <= {2'b00, temp, 1'b1, data[3:0]};  // year, second digit
310                 state <= state + 1;  // S_DISP_3
311              end
312              S_DISP_3: begin
313                 ascii_out_reg <= 8'h2F;  // forward slash
314                 temp <= ~(cursor_blink_capture & (cursor_pos_capture == 4));
315                 state <= state + 1;  // S_DISP_4
316              end
317              S_DISP_4: begin
318                 addr_reg <= {2'b00, DT_capture[21:17]};
319                 ascii_out_reg <= {2'b00, temp, 1'b1, data[7:4]};  // month, first digit
320                 state <= state + 1;  // S_DISP_5
321              end
322              S_DISP_5: begin
323                 ascii_out_reg <= {2'b00, temp, 1'b1, data[3:0]};  // month, second digit
324                 state <= state + 1;  // S_DISP_6
325              end
326              S_DISP_6: begin
327                 ascii_out_reg <= 8'h2F;  // forward slash
328                 temp <= ~(cursor_blink_capture & (cursor_pos_capture == 3));
329                 state <= state + 1;  // S_DISP_7
330              end
331              S_DISP_7: begin
332                 addr_reg <= {2'b00, DT_capture[16:12]};
333                 ascii_out_reg <= {2'b00, temp, 1'b1, data[7:4]};  // day, first digit
334                 state <= state + 1;  // S_DISP_8
335              end
336              S_DISP_8: begin
337                 ascii_out_reg <= {2'b00, temp, 1'b1, data[3:0]};  // day, second digit
338                 state <= state + 1;  // S_DISP_9
339                 temp <= ~(cursor_blink_capture & (cursor_pos_capture == 2));
340              end
341              S_DISP_9: begin
342                 addr_reg <= {1'b0, DT_capture[11:6]};
343                 ascii_out_reg <= {2'b00, temp, 1'b1, data[7:4]};  // hour, first digit
344                 state <= state + 1;  // S_DISP_10
345              end
346              S_DISP_10: begin
347                 ascii_out_reg <= {2'b00, temp, 1'b1, data[3:0]};  // hour, second digit
348                 state <= state + 1;  // S_DISP_11
349              end
350              S_DISP_11: begin
351                 ascii_out_reg <= 8'h3A;  // colon
352                 temp <= ~(cursor_blink_capture & (cursor_pos_capture == 1));
353                 state <= state + 1;  // S_DISP_12
354              end
355              S_DISP_12: begin
356                 addr_reg <= {1'b0, DT_capture[5:0]};
357                 ascii_out_reg <= {2'b00, temp, 1'b1, data[7:4]};  // minute, first digit
358                 state <= state + 1;  // S_DISP_13
359              end
360              S_DISP_13: begin
361                 ascii_out_reg <= {2'b00, temp, 1'b1, data[3:0]};  // minute, second digit
362                 state <= state + 1;  // S_DISP_14
363              end
364              S_DISP_14: begin
365                 ascii_out_reg <= 8'h3A;  // colon
```

```
366                     temp <= ~(cursor_blink_capture & (cursor_pos_capture == 0));
367                     state <= state + 1;  // S_DISP_15
368                  end
369                  S_DISP_15: begin
370                     ascii_out_reg <= {2'b00, temp, 1'b1, data[7:4]};  // second, first digit
371                     state <= state + 1;  // S_DISP_16
372                  end
373                  S_DISP_16: begin
374                     ascii_out_reg <= {2'b00, temp, 1'b1, data[3:0]};  // second, second digit
375                     state <= S_IDLE;
376                  end
377               endcase
378            end
379         end
380      endmodule
381
```

```
1         `timescale 1ns / 1ps
2         ////////////////////////////////////////////////////////////////////////////////
3         // Company:          6.111
4         // Engineer:         Sachin Shinde
5         //
6         // Create Date:      03:07:52 10/31/2012
7         // Design Name:      AC97-PCM Interface
8         // Module Name:      AC97_PCM
9         // Project Name:     6.111 Final Project
10        // Target Devices: Xilinx XC2V6000
11        // Tool versions:
12        // Description:      An interface between the LM4550 AC '97 Audio Codec and the
13        //                   48kHz PCM data stream that:
14        //                   1) Assembles PCM audio output into LM4550 sdata_out frames
15        //                   2) Disassembles LM4550 sdata_in frames into PCM audio input
16        //                   3) Manages all LM4550 control signals (including headphone
17        //                      volume)
18        //
19        //                   Ready (a one-clock pulse) times I/O of the PCM stream. On the
20        //                   rising edge of ready, audio input is avaiable. On the falling
21        //                   edge of ready, audio output is latched.
22        //
23        //                   The PCM stream bit depth is 16 and sampling rate is 8kHz.
24        //
25        // Dependencies:
26        //
27        // Revision:
28        // Revision 0.01 - File Created
29        // Additional Comments:
30        //                   Audio input will be held for at least 128 AC97 bit clocks
31        //                   (around 280 27MHz clocks w/ AC97 bit clock @ 12.288MHz).
32        //
33        ////////////////////////////////////////////////////////////////////////////////
34        module AC97_PCM(
35           input wire clock_27mhz,
36           input wire reset,
37           input wire [4:0] volume,
38           // PCM interface signals
39           output wire [15:0] audio_in_data,
40           input wire [15:0] audio_out_data,
41           output wire ready,
42           // LM4550 interface signals
43           output reg audio_reset_b,
44           output wire ac97_sdata_out,
45           input wire ac97_sdata_in,
46           output wire ac97_synch,
47           input wire ac97_bit_clock
48           );
49
50           wire [7:0] command_address;
51           wire [15:0] command_data;
52           wire command_valid;
53           wire [19:0] left_in_data, right_in_data;
54           wire [19:0] left_out_data, right_out_data;
55
56           // Wait 1024 clock cycles before enabling the LM4550
57           reg [9:0] reset_count;
58           always @(posedge clock_27mhz) begin
59              if (reset) begin
60                 audio_reset_b = 1'b0;
61                 reset_count = 0;
```

```
 62             end
 63             else if (reset_count == 1023)
 64                audio_reset_b = 1'b1;
 65             else
 66                reset_count = reset_count+1;
 67          end
 68
 69          // Assemble/Disassemble serial frames for LM4550
 70          wire ac97_ready;
 71          wire slot_req;    // slot request for Variable Rate Audio (VRA)
 72          ac97 ac97(.ready(ac97_ready),
 73                    .command_address(command_address),
 74                    .command_data(command_data),
 75                    .command_valid(command_valid),
 76                    .left_data(left_out_data), .left_valid(1'b1),
 77                    .right_data(right_out_data), .right_valid(1'b1),
 78                    .left_in_data(left_in_data), .right_in_data(right_in_data),
 79                    .slot_req(slot_req),
 80                    .ac97_sdata_out(ac97_sdata_out),
 81                    .ac97_sdata_in(ac97_sdata_in),
 82                    .ac97_synch(ac97_synch),
 83                    .ac97_bit_clock(ac97_bit_clock));
 84
 85          // Generate int_ready (one-clock pulse, synchronous with clock_27mhz)
 86          // from ac97_ready (long pulse, synchronouse with ac97_bit_clock)
 87          reg [2:0] ready_sync;
 88          wire int_ready;
 89          always @ (posedge clock_27mhz)
 90             ready_sync <= {ready_sync[1:0], ac97_ready};
 91          assign int_ready = ready_sync[1] & ~ready_sync[2];
 92
 93          // Generate ready signal taking into account LM4550 slot requests
 94          assign ready = int_ready & slot_req;
 95
 96          // Latch audio output on falling edge of ready
 97          reg [15:0] out_data;
 98          always @ (posedge clock_27mhz)
 99             if (ready) out_data <= audio_out_data;
100
101          // Assign audio input (16 bits)
102          assign audio_in_data = left_in_data[19:4];
103
104          // Assign LM4550 L/R output data signals
105          assign left_out_data = {out_data, 4'h0};
106          assign right_out_data = left_out_data;
107
108          // Generate R/W control signals for LM4550
109          ac97commands cmds(.clock(clock_27mhz), .ready(int_ready),
110                            .command_address(command_address),
111                            .command_data(command_data),
112                            .command_valid(command_valid),
113                            .volume(volume),
114                            .source(3'b000));     // mic
115       endmodule
116
117       // Assemble/Disassemble serial frames for LM4550
118       module ac97 (
119          output reg ready,
120          // Commmand signals
121          input wire [7:0] command_address,
122          input wire [15:0] command_data,
```

```
123          input wire command_valid,
124          // L/R data output signals
125          input wire [19:0] left_data, right_data,
126          input wire left_valid, right_valid,
127          // L/R data input signals
128          output reg [19:0] left_in_data, right_in_data,
129          output reg slot_req,
130          // LM4550 Serial Interface signals
131          output reg ac97_sdata_out,
132          input wire ac97_sdata_in,
133          output reg ac97_synch,
134          input wire ac97_bit_clock
135          );
136
137          // Counter for bit position in frame
138          reg [7:0] bit_count;
139
140          // Latched output signals
141          reg [19:0] l_cmd_addr;
142          reg [19:0] l_cmd_data;
143          reg [19:0] l_left_data, l_right_data;
144          reg l_cmd_v, l_left_v, l_right_v;
145
146          initial begin
147             ready <= 1'b0;
148             // synthesis attribute init of ready is "0";
149             ac97_sdata_out <= 1'b0;
150             // synthesis attribute init of ac97_sdata_out is "0";
151             ac97_synch <= 1'b0;
152             // synthesis attribute init of ac97_synch is "0";
153
154             bit_count <= 8'h00;
155             // synthesis attribute init of bit_count is "0000";
156             l_cmd_v <= 1'b0;
157             // synthesis attribute init of l_cmd_v is "0";
158             l_left_v <= 1'b0;
159             // synthesis attribute init of l_left_v is "0";
160             l_right_v <= 1'b0;
161             // synthesis attribute init of l_right_v is "0";
162
163             left_in_data <= 20'h00000;
164             // synthesis attribute init of left_in_data is "00000";
165             right_in_data <= 20'h00000;
166             // synthesis attribute init of right_in_data is "00000";
167
168             slot_req <= 1'b0;
169             // synthesis attribute init of right_in_data is "0";
170          end
171
172          // Generate basic signals
173          always @(posedge ac97_bit_clock) begin
174             // Generate the sync signal
175             if (bit_count == 255)
176                ac97_synch <= 1'b1;
177             if (bit_count == 15)
178                ac97_synch <= 1'b0;
179
180             // Generate the ready signal (synchronous with ac97_bit_clock)
181             if (bit_count == 128)
182                ready <= 1'b1;
183             if (bit_count == 2)
```

```
184                  ready <= 1'b0;
185
186               // Update bit_count
187               bit_count <= bit_count+1;
188            end
189
190            // Generate ac97_sdata_out from latched audio output
191            always @(posedge ac97_bit_clock) begin
192               if ((bit_count >= 0) && (bit_count <= 15)) begin
193                  // Slot 0: Tags
194                  case (bit_count[3:0])
195                     4'h0: ac97_sdata_out <= 1'b1;      // Frame valid
196                     4'h1: ac97_sdata_out <= l_cmd_v;   // Command address valid
197                     4'h2: ac97_sdata_out <= l_cmd_v;   // Command data valid
198                     4'h3: ac97_sdata_out <= l_left_v;  // Left data valid
199                     4'h4: ac97_sdata_out <= l_right_v; // Right data valid
200                     default: ac97_sdata_out <= 1'b0;
201                  endcase
202               end
203               else if ((bit_count >= 16) && (bit_count <= 35))
204                  // Slot 1: Command address (8-bits, left justified)
205                  ac97_sdata_out <= l_cmd_v ? l_cmd_addr[35-bit_count] : 1'b0;
206               else if ((bit_count >= 36) && (bit_count <= 55))
207                  // Slot 2: Command data (16-bits, left justified)
208                  ac97_sdata_out <= l_cmd_v ? l_cmd_data[55-bit_count] : 1'b0;
209               else if ((bit_count >= 56) && (bit_count <= 75)) begin
210                  // Slot 3: Left channel
211                  ac97_sdata_out <= l_left_v ? l_left_data[19] : 1'b0;
212                  l_left_data <= { l_left_data[18:0], l_left_data[19] };
213               end
214               else if ((bit_count >= 76) && (bit_count <= 95))
215                  // Slot 4: Right channel
216                  ac97_sdata_out <= l_right_v ? l_right_data[95-bit_count] : 1'b0;
217               // Latch output signals at the end of each frame. This ensures that
218               // the first frame after reset will be empty.
219               else if (bit_count == 255) begin
220                  l_cmd_addr <= {command_address, 12'h000};
221                  l_cmd_data <= {command_data, 4'h0};
222                  l_cmd_v <= command_valid;
223                  l_left_data <= left_data;
224                  l_left_v <= left_valid;
225                  l_right_data <= right_data;
226                  l_right_v <= right_valid;
227                  ac97_sdata_out <= 1'b0;
228               end
229               else
230                  ac97_sdata_out <= 1'b0;
231            end
232
233            // Extract audio input from ac97_sdata_in
234            always @(negedge ac97_bit_clock) begin
235               if ((bit_count >= 57) && (bit_count <= 76))
236                  // Slot 3: Left channel
237                  left_in_data <= { left_in_data[18:0], ac97_sdata_in };
238               else if ((bit_count >= 77) && (bit_count <= 96))
239                  // Slot 4: Right channel
240                  right_in_data <= { right_in_data[18:0], ac97_sdata_in };
241            end
242
243            // Extract slot request bit for Variable Rate Audio (VRA)
244            always @(negedge ac97_bit_clock) begin
```

```
245             if (bit_count == 25)
246                slot_req <= ~ac97_sdata_in;
247          end
248       endmodule
249
250       // Generate R/W control signals for LM4550
251       module ac97commands (
252          input wire clock,
253          input wire ready,
254          // Command signals
255          output wire [7:0] command_address,
256          output wire [15:0] command_data,
257          output reg command_valid,
258          // Other signals
259          input wire [4:0] volume,
260          input wire [2:0] source
261          );
262
263          // Create command register & state
264          reg [23:0] command;
265          reg [3:0] state;
266
267          // Assign command signal outputs
268          assign command_address = command[23:16];
269          assign command_data = command[15:0];
270
271          initial begin
272             command <= 24'h00_0000;
273             // synthesis attribute init of command is "00_0000";
274             command_valid <= 1'b0;
275             // synthesis attribute init of command_valid is "0";
276             state <= 4'h0;
277             // synthesis attribute init of state is "0";
278          end
279
280          // Convert volume to attentuation
281          wire [4:0] vol;
282          assign vol = 5'd31 - volume;
283
284          // Update state and command vars
285          always @(posedge clock) begin
286             // Increment state
287             if (ready) state <= state+1;
288
289             // Set command
290             case (state)
291                4'h0: begin // Read reset register (7'h00)
292                   command <= 24'h80_0000;
293                   command_valid <= 1'b1;
294                end
295                4'h1: // Read reset register (7'h00)
296                   command <= 24'h80_0000;
297                4'h2: // Set Variable Rate Audio (7'h2A) LSB to 1
298                   command <= 24'h2A_0001;
299                4'h3: // Set PCM DAC Sampling rate (7'h2C) to 8kHz
300                   command <= 24'h2C_1F40;
301                4'h4: // Set PCM ADC Sampling rate (7'h32) to 8kHz
302                   command <= 24'h32_1F40;
303                4'h5: // Set headphone volume (7'h04) on both L/R to vol
304                   command <= { 8'h04, 3'b000, vol, 3'b000, vol };
305                4'h6: // Set PCM output volume (7'h18) on both L/R to 6dB gain
```

```
306                 command <= 24'h18_0808;
307             4'h7: // Set record select (7'h1A) to source (mic)
308                 command <= { 8'h1A, 5'b00000, source, 5'b00000, source};
309             4'h8: // Set record gain (7'h1C) to max (22.5dB gain)
310                 command <= 24'h1C_0F0F;
311             4'h9: // Set mic volume (7'h0E) bit 6 for 20dB boost amplifier
312                 command <= 24'h0E_8048;
313             4'hA: // Set PC beep volume (7'h0A) to 0dB attentuation
314                 command <= 24'h0A_0000;
315             4'hB: // Set general purpose register (7'h20) so that:
316                   //   * PCM output bypasses 3D circuitry
317                   //   * National 3D Sound is off
318                   //   * mic1 is selected
319                   //   * No ADC/DAC loopback
320                 command <= 24'h20_8000;
321             default:
322                 command <= 24'h80_0000;
323         endcase
324     end
325 endmodule
326
```

```
1    `timescale 1ns / 1ps
2    //////////////////////////////////////////////////////////////////////////////////
3    // Company:
4    // Engineer:       Sachin Shinde
5    //
6    // Create Date:    07:03:26 11/21/2012
7    // Design Name:
8    // Module Name:    PHY_Pair
9    // Project Name:
10   // Target Devices:
11   // Tool versions:
12   // Description:    Instantiate a pair of PHY modules, which share an encoder and
13   //                 decoder.
14   //
15   // Dependencies:
16   //
17   // Revision:
18   // Revision 0.01 - File Created
19   // Additional Comments:
20   //
21   //////////////////////////////////////////////////////////////////////////////////
22   module PHY_Pair(
23      // PHY I/O
24      input clk_40mhz,      // 40Mhz Clock
25      input clk_160mhz,     // Sampling Clock
26      input reset,          // Active-High Reset
27      // PHY_1 I/O
28      input TCV_RX_1,                // Transceiver RX
29      output TCV_TX_1,               // Transceiver TX
30      output TCV_TX_en_1,            // Transceiver TX enable
31      input [7:0] D_TX_1,            // DLC Data TX
32      output [7:0] D_RX_1,         // DLC Data RX
33      input D_TX_ready_1,            // DLC Data TX Ready
34      output D_RX_ready_1,           // DLC Data RX Ready
35      output CD_1,                   // Collision Detect (while Sending)
36      output TX_success_1,           // Successful transmission
37      output IB_1,                   // Idle Bus
38      // PHY_2 I/O
39      input TCV_RX_2,                // Transceiver RX
40      output TCV_TX_2,               // Transceiver TX
41      output TCV_TX_en_2,            // Transceiver TX enable
42      input [7:0] D_TX_2,            // DLC Data TX
43      output [7:0] D_RX_2,         // DLC Data RX
44      input D_TX_ready_2,            // DLC Data TX Ready
45      output D_RX_ready_2,           // DLC Data RX Ready
46      output CD_2,                   // Collision Detect (while Sending)
47      output TX_success_2,           // Successful transmission
48      output IB_2                   // Idle Bus
49      ,// DEBUG
50      output [2:0] rcv_state_DEBUG,
51      output [2:0] state_DEBUG,
52      output [7:0] encoder_din_1_DEBUG,
53      output [9:0] encoder_dout_1_DEBUG,
54      output encoder_nd_1_DEBUG,
55      output in_rd_en_DEBUG,
56      output [9:0] in_dout_DEBUG,
57      output in_rst_DEBUG,
58      output in_empty_DEBUG,
59      output [3:0] data_to_send_DEBUG,
60      output [9:0] TX_shift_reg_DEBUG,
61      output clk_20mhz_DEBUG,
```

```
 62          output TX_started_DEBUG,
 63          output [23:0] shift_reg_DEBUG,
 64          output RX_sampled_ready_DEBUG,
 65          output cnsy_zero_DEBUG,
 66          output cnsy_one_DEBUG,
 67          output check_cnsy_DEBUG,
 68          output [2:0] trans_reg_DEBUG,
 69          output init_trans_detected_DEBUG,
 70          output trans_zero_DEBUG,
 71          output trans_late_DEBUG,
 72          output trans_not_exist_DEBUG,
 73          output init_trans_detect_DEBUG,
 74          output demux_nd_DEBUG,
 75          output [9:0] demux_dout_DEBUG,
 76          output CD_comp_1_DEBUG,
 77          output CD_signal_1_DEBUG,
 78          output CD_decoder_1_DEBUG,
 79          output CD_comp_2_DEBUG,
 80          output CD_signal_2_DEBUG,
 81          output CD_decoder_2_DEBUG,
 82          output [9:0] comp_dout_DEBUG,
 83          output encoder_ce_1_DEBUG,
 84          output [9:0] demux_out_2_DEBUG,     // Demultiplexer Output
 85          output demux_out_ready_2_DEBUG,      // Demultiplexer Output Ready
 86          output decoder_init_2_DEBUG,         // Decoder Initialization
 87          output [7:0] decoder_dout_2_DEBUG,  // Decoder Data Output
 88          output decoder_kout_2_DEBUG,         // Decoder Special Char Output
 89          output decoder_dout_ready_2_DEBUG    // Decoder Output Ready
 90          // END DEBUG
 91          );
 92
 93          // Instantiate first PHY
 94          // Decoder I/O
 95          wire [9:0] demux_out_1;    // Demultiplexer Output
 96          wire demux_out_ready_1;    // Demultiplexer Output Ready
 97          wire decoder_init_1;       // Decoder Initialization
 98          wire [7:0] decoder_dout_1; // Decoder Data Output
 99          wire decoder_kout_1;       // Decoder Special Char Output
100          wire decoder_dout_ready_1; // Decoder Output Ready
101          wire CD_decoder_1;         // Collision Detect
102          // Encoder I/O
103          wire encoder_force_code_1;  // Encoder reset
104          wire encoder_ce_1;          // Encoder chip reset
105          wire [7:0] encoder_din_1;   // Encoder data wire
106          wire encoder_kin_1;         // Encoder special character wire
107          wire [9:0] encoder_dout_1;  // Encoder character dout
108          wire encoder_nd_1;          // Encoder new data signal
109          PHY PHY_1(
110             // PHY_1 I/O
111             .clk_40mhz(clk_40mhz),            // System Clock
112             .clk_160mhz(clk_160mhz),          // Sampling Clock
113             .reset(reset),                    // Active-High Reset
114             .TCV_RX(TCV_RX_1),                  // Transceiver RX
115             .TCV_TX(TCV_TX_1),                  // Transceiver TX
116             .TCV_TX_en(TCV_TX_en_1),            // Transceiver TX enable
117             .D_TX(D_TX_1),                      // DLC Data TX
118             .D_RX(D_RX_1),                      // DLC Data RX
119             .D_TX_ready(D_TX_ready_1),          // DLC Data TX Ready
120             .D_RX_ready(D_RX_ready_1),          // DLC Data RX Ready
121             .CD(CD_1),                          // Collision Detect (while Sending_1)
122             .TX_success(TX_success_1),          // Successful transmission
```

```
123             .IB(IB_1),                              // Idle Bus
124             // Decoder I/O
125             .demux_out(demux_out_1),                     // Demultiplexer Output
126             .demux_out_ready(demux_out_ready_1),         // Demultiplexer Output Ready
127             .decoder_init(decoder_init_1),               // Decoder Initialization
128             .decoder_dout(decoder_dout_1),               // Decoder Data Output
129             .decoder_kout(decoder_kout_1),               // Decoder Special Char Output
130             .decoder_dout_ready(decoder_dout_ready_1),   // Decoder Output Ready
131             .CD_decoder(CD_decoder_1),                   // Collision Detect
132             // Encoder I/O
133             .encoder_force_code(encoder_force_code_1),
134             .encoder_ce(encoder_ce_1),
135             .encoder_din(encoder_din_1),
136             .encoder_kin(encoder_kin_1),
137             .encoder_dout(encoder_dout_1),
138             .encoder_nd(encoder_nd_1)
139             ,// DEBUG
140             .rcv_state_DEBUG(rcv_state_DEBUG),
141             .state_DEBUG(state_DEBUG),
142             .in_rd_en_DEBUG(in_rd_en_DEBUG),
143             .in_dout_DEBUG(in_dout_DEBUG),
144             .in_rst_DEBUG(in_rst_DEBUG),
145             .in_empty_DEBUG(in_empty_DEBUG),
146             .data_to_send_DEBUG(data_to_send_DEBUG),
147             .TX_shift_reg_DEBUG(TX_shift_reg_DEBUG),
148             .clk_20mhz_DEBUG(clk_20mhz_DEBUG),
149             .TX_started_DEBUG(TX_started_DEBUG),
150             .shift_reg_DEBUG(shift_reg_DEBUG),
151             .RX_sampled_ready_DEBUG(RX_sampled_ready_DEBUG),
152             .cnsy_zero_DEBUG(cnsy_zero_DEBUG),
153             .cnsy_one_DEBUG(cnsy_one_DEBUG),
154             .check_cnsy_DEBUG(check_cnsy_DEBUG),
155             .trans_reg_DEBUG(trans_reg_DEBUG),
156             .init_trans_detected_DEBUG(init_trans_detected_DEBUG),
157             .trans_zero_DEBUG(trans_zero_DEBUG),
158             .trans_late_DEBUG(trans_late_DEBUG),
159             .trans_not_exist_DEBUG(trans_not_exist_DEBUG),
160             .init_trans_detect_DEBUG(init_trans_detect_DEBUG),
161             .demux_nd_DEBUG(demux_nd_DEBUG),
162             .demux_dout_DEBUG(demux_dout_DEBUG),
163             .CD_comp_DEBUG(CD_comp_1_DEBUG),
164             .CD_signal_DEBUG(CD_signal_1_DEBUG),
165             .CD_decoder_DEBUG(CD_decoder_1_DEBUG),
166             .comp_dout_DEBUG(comp_dout_DEBUG)
167             // END DEBUG
168          );
169
170          // Instantiate second PHY
171          // Decoder I/O
172          wire [9:0] demux_out_2;    // Demultiplexer Output
173          wire demux_out_ready_2;    // Demultiplexer Output Ready
174          wire decoder_init_2;       // Decoder Initialization
175          wire [7:0] decoder_dout_2; // Decoder Data Output
176          wire decoder_kout_2;       // Decoder Special Char Output
177          wire decoder_dout_ready_2; // Decoder Output Ready
178          wire CD_decoder_2;         // Collision Detect
179          // Encoder I/O
180          wire encoder_force_code_2;  // Encoder reset
181          wire encoder_ce_2;          // Encoder chip reset
182          wire [7:0] encoder_din_2;   // Encoder data wire
183          wire encoder_kin_2;         // Encoder special character wire
```

```
184         wire [9:0] encoder_dout_2;  // Encoder character dout
185         wire encoder_nd_2;          // Encoder new data signal
186         PHY PHY_2(
187            // PHY_2 I/O
188            .clk_40mhz(clk_40mhz),      // System Clock
189            .clk_160mhz(clk_160mhz),    // Sampling Clock
190            .reset(reset),              // Active-High Reset
191            .TCV_RX(TCV_RX_2),                   // Transceiver RX
192            .TCV_TX(TCV_TX_2),                   // Transceiver TX
193            .TCV_TX_en(TCV_TX_en_2),             // Transceiver TX enable
194            .D_TX(D_TX_2),                       // DLC Data TX
195            .D_RX(D_RX_2),                       // DLC Data RX
196            .D_TX_ready(D_TX_ready_2),           // DLC Data TX Ready
197            .D_RX_ready(D_RX_ready_2),           // DLC Data RX Ready
198            .CD(CD_2),                           // Collision Detect (while Sending_2)
199            .TX_success(TX_success_2),           // Successful transmission
200            .IB(IB_2),                           // Idle Bus
201            // Decoder I/O
202            .demux_out(demux_out_2),                  // Demultiplexer Output
203            .demux_out_ready(demux_out_ready_2),      // Demultiplexer Output Ready
204            .decoder_init(decoder_init_2),            // Decoder Initialization
205            .decoder_dout(decoder_dout_2),            // Decoder Data Output
206            .decoder_kout(decoder_kout_2),            // Decoder Special Char Output
207            .decoder_dout_ready(decoder_dout_ready_2),  // Decoder Output Ready
208            .CD_decoder(CD_decoder_2),                // Collision Detect
209            // Encoder I/O
210            .encoder_force_code(encoder_force_code_2),
211            .encoder_ce(encoder_ce_2),
212            .encoder_din(encoder_din_2),
213            .encoder_kin(encoder_kin_2),
214            .encoder_dout(encoder_dout_2),
215            .encoder_nd(encoder_nd_2)
216            ,// DEBUG
217            .CD_comp_DEBUG(CD_comp_2_DEBUG),
218            .CD_signal_DEBUG(CD_signal_2_DEBUG),
219            .CD_decoder_DEBUG(CD_decoder_2_DEBUG)
220            // END DEBUG
221         );
222
223         // Instantiate Decoder
224         wire decoder_code_err_1;
225         wire decoder_disp_err_1;
226         wire decoder_ce_1;
227         assign decoder_ce_1 = demux_out_ready_1 | decoder_init_1;
228         wire decoder_code_err_2;
229         wire decoder_disp_err_2;
230         wire decoder_ce_2;
231         assign decoder_ce_2 = demux_out_ready_2 | decoder_init_2;
232         PHY_Decoder PHY_DEC_1(
233            // First Decoder I/O
234            .clk(clk_40mhz),
235            .ce(decoder_ce_1),
236            .sinit(decoder_init_1),
237            .din(demux_out_1),
238            .dout(decoder_dout_1),
239            .kout(decoder_kout_1),
240            .code_err(decoder_code_err_1),
241            .disp_err(decoder_disp_err_1),
242            .nd(decoder_dout_ready_1),
243            // Second Decoder I/O
244            .clk_b(clk_40mhz),
```

```
245             .ce_b(decoder_ce_2),
246             .sinit_b(decoder_init_2),
247             .din_b(demux_out_2),
248             .dout_b(decoder_dout_2),
249             .kout_b(decoder_kout_2),
250             .code_err_b(decoder_code_err_2),
251             .disp_err_b(decoder_disp_err_2),
252             .nd_b(decoder_dout_ready_2)
253          );
254 
255          // Assign outputs
256          assign CD_decoder_1 = decoder_code_err_1 | decoder_disp_err_1;
257          assign CD_decoder_2 = decoder_code_err_2 | decoder_disp_err_2;
258 
259          // Instantiate Encoder
260          wire encoder_real_ce_1;
261          wire encoder_real_ce_2;
262          assign encoder_real_ce_1 = encoder_ce_1 | encoder_force_code_1;
263          assign encoder_real_ce_2 = encoder_ce_2 | encoder_force_code_2;
264          PHY_Encoder PHY_ENC_1(
265             // First Encoder I/O
266             .clk(clk_40mhz),
267             .force_code(encoder_force_code_1),
268             .din(encoder_din_1),
269             .kin(encoder_kin_1),
270             .ce(encoder_real_ce_1),
271             .dout(encoder_dout_1),
272             .nd(encoder_nd_1),
273             // Second Encoder I/O
274             .clk_b(clk_40mhz),
275             .force_code_b(encoder_force_code_2),
276             .din_b(encoder_din_2),
277             .kin_b(encoder_kin_2),
278             .ce_b(encoder_real_ce_2),
279             .dout_b(encoder_dout_2),
280             .nd_b(encoder_nd_2)
281             );
282 
283          // DEBUG
284          assign encoder_din_2_DEBUG = encoder_din_2;
285          assign encoder_dout_2_DEBUG= encoder_dout_2;
286          assign encoder_nd_2_DEBUG = encoder_nd_2;
287          assign encoder_ce_2_DEBUG = encoder_ce_2;
288          assign demux_out_2_DEBUG = demux_out_2;
289          assign demux_out_ready_2_DEBUG = demux_out_ready_2;
290          assign decoder_init_2_DEBUG = decoder_init_2;
291          assign decoder_dout_2_DEBUG = decoder_dout_2;
292          assign decoder_kout_2_DEBUG = decoder_kout_2;
293          assign decoder_dout_ready_2_DEBUG = decoder_dout_ready_2;
294       endmodule
295 
```

```
1   `timescale 1ns / 1ps
2   ////////////////////////////////////////////////////////////////////////////////
3   // Company:
4   // Engineer:
5   //
6   // Create Date:    13:03:25 11/12/2012
7   // Design Name:
8   // Module Name:    PHY
9   // Project Name:
10  // Target Devices:
11  // Tool versions:
12  // Description:    Physical Layer in OSI Model.
13  //
14  // Dependencies:
15  //
16  // Revision:
17  // Revision 0.01 - File Created
18  // Additional Comments:
19  //
20  ////////////////////////////////////////////////////////////////////////////////
21  module PHY(
22     // PHY I/O
23     input clk_40mhz,              // 40MHz Clock
24     input clk_160mhz,             // Sampling 160 MHz Clock
25     input reset,                  // Active-High Reset
26     input TCV_RX,                 // Transceiver RX
27     output TCV_TX,                // Transceiver TX
28     output TCV_TX_en,             // Transceiver TX enable
29     input [7:0] D_TX,             // DLC Data TX
30     output [7:0] D_RX,          // DLC Data RX
31     input D_TX_ready,             // DLC Data TX Ready
32     output D_RX_ready,            // DLC Data RX Ready
33     output CD,                    // Collision Detect (while Sending)
34     output TX_success,            // Successful transmission
35     output IB,                    // Idle Bus
36     // Decoder I/O
37     output [9:0] demux_out,     // Demultiplexer Output
38     output demux_out_ready,     // Demultiplexer Output Ready
39     output decoder_init,        // Decoder Initialization
40     input [7:0] decoder_dout,   // Decoder Data Output
41     input decoder_kout,         // Decoder Special Char Output
42     input decoder_dout_ready,   // Decoder Output Ready
43     input CD_decoder,           // Decoder Collision Detect
44     // Encoder I/O
45     output encoder_force_code, // Encoder reset
46     output encoder_ce,          // Encoder chip reset
47     output [7:0] encoder_din,   // Encoder data input
48     output encoder_kin,         // Encoder special character input
49     input [9:0] encoder_dout,   // Encoder character dout
50     input encoder_nd            // Encoder new data signal
51     ,// DEBUG
52     output [2:0] rcv_state_DEBUG,
53     output [2:0] state_DEBUG,
54     output in_rd_en_DEBUG,
55     output [9:0] in_dout_DEBUG,
56     output in_rst_DEBUG,
57     output in_empty_DEBUG,
58     output [3:0] data_to_send_DEBUG,
59     output [9:0] TX_shift_reg_DEBUG,
60     output clk_20mhz_DEBUG,
61     output TX_started_DEBUG,
```

```
62          output [23:0] shift_reg_DEBUG,
63          output RX_sampled_ready_DEBUG,
64          output cnsy_zero_DEBUG,
65          output cnsy_one_DEBUG,
66          output check_cnsy_DEBUG,
67          output [2:0] trans_reg_DEBUG,
68          output init_trans_detected_DEBUG,
69          output trans_zero_DEBUG,
70          output trans_late_DEBUG,
71          output trans_not_exist_DEBUG,
72          output init_trans_detect_DEBUG,
73          output demux_nd_DEBUG,
74          output [9:0] demux_dout_DEBUG,
75          output CD_comp_DEBUG,
76          output CD_signal_DEBUG,
77          output CD_decoder_DEBUG,
78          output [9:0] comp_dout_DEBUG
79          // END DEBUG
80          );
81
82          // Instantiate main ready & enable output registers
83          reg TCV_TX_reg;
84          reg TCV_TX_en_reg;
85          reg D_RX_ready_reg;
86          reg CD_reg;
87          reg IB_reg;
88          reg TX_success_reg;
89
90          // Assign outputs
91          assign TCV_TX = TCV_TX_reg;
92          assign TCV_TX_en = TCV_TX_en_reg;
93          assign D_RX_ready = D_RX_ready_reg;
94          assign CD = CD_reg;
95          assign IB = IB_reg;
96          assign TX_success = TX_success_reg;
97
98          ///////////////////////////////////////////////////////////////////////////////
99          // RX LOGIC
100         ///////////////////////////////////////////////////////////////////////////////
101
102         // Instantiate RX Sampler for demultiplexing
103         wire [7:0] RX_sampled;
104         wire RX_sampled_ready;
105         PHY_RX_Sample PHY_RX_SAMPLE_1(
106            .clk_40mhz(clk_40mhz),
107            .clk_160mhz(clk_160mhz),
108            .reset(reset),
109            .RX(TCV_RX),
110            .out_ready(RX_sampled_ready),
111            .RX_sampled(RX_sampled)
112         );
113
114         // Instantiate shift registers for clock recovery
115         reg [23:0] shift_reg;
116         always @(posedge clk_40mhz) begin
117            if (RX_sampled_ready) begin
118               shift_reg <= {shift_reg[15:0], RX_sampled};
119            end
120         end
121
122         // Instantiate transition address registers
```

```
123         reg [2:0] trans_reg, init_trans_reg;
124         wire [3:0] trans_reg_early, trans_reg_zero, trans_reg_late;
125         wire [4:0] trans_reg_cnsy1 [0:4];
126
127         // Instantiate transition check registers
128         reg trans_zero, trans_late, trans_not_exist;
129
130         // Instantiate signal consistency check registers
131         reg check_cnsy, cnsy_one, cnsy_zero;
132
133         // Assign shift register addresses
134         assign trans_reg_early = trans_reg + 5;
135         assign trans_reg_zero = trans_reg + 4;
136         assign trans_reg_late = trans_reg + 3;
137         assign trans_reg_cnsy1[0] = trans_reg + 12;
138         assign trans_reg_cnsy1[1] = trans_reg + 13;
139         assign trans_reg_cnsy1[2] = trans_reg + 14;
140         assign trans_reg_cnsy1[3] = trans_reg + 15;
141         assign trans_reg_cnsy1[4] = trans_reg + 16;
142
143         // Recover clock from transitions, sample as needed
144         reg skip_frame;
145         reg init_trans_detect;
146         reg init_trans_detected;
147         always @(posedge clk_40mhz) begin
148            if (reset) begin
149               trans_reg <= 0;
150               init_trans_detected <= 0;
151               check_cnsy <= 0;
152               skip_frame <= 0;
153            end
154            // Determine transition edge, store next transition measure points
155            // Handle edge cases where frame pointer moves out of frame (both too early and too
        late)
156            else if (RX_sampled_ready) begin
157               cnsy_one <= shift_reg[12] & shift_reg[13] & shift_reg[14] & shift_reg[15] &
        shift_reg[16];
158               cnsy_zero <= ~shift_reg[12] & ~shift_reg[13] & ~shift_reg[14] & ~shift_reg[15]
        shift_reg[16];
159               if (init_trans_detect) begin  // if detecting initial edge
160                  trans_reg <= (|init_trans_reg) ? init_trans_reg: (~shift_reg[5]) ? 3'd2: (~
        shift_reg[4]) ? 3'd1: (~shift_reg[3]) ? 3'd0: trans_reg;
161                  init_trans_detected <= (init_trans_reg) || (|(~shift_reg[5:3]));
162                  check_cnsy <= 1'b0;
163                  skip_frame <= 1'b0;
164               end
165               else if (trans_not_exist) begin  // if transition doesn't exist
166                  init_trans_detected <= 0;
167                  trans_reg <= trans_reg;
168                  check_cnsy <= 1'b0;
169                  skip_frame <= 1'b0;
170               end
171               else if (trans_late) begin
172                  init_trans_detected <= 0;
173                  trans_reg <= trans_reg - 1;
174                  check_cnsy <= 1'b0;
175                  skip_frame <= (trans_reg == 0); // EDGE CASE: Frame pointer moves too late (
        transition in frame), so skip frame.
176               end
177               else if (trans_zero) begin
178                  init_trans_detected <= 0;
```

```
179                 trans_reg <= trans_reg;
180                 check_cnsy <= 1'b0;
181                 skip_frame <= 1'b0;
182              end
183              else begin // if (trans_early)
184                 init_trans_detected <= 0;
185                 trans_reg <= trans_reg + 1;
186                 check_cnsy <= (trans_reg == 7); // EDGE CASE: Frame pointer moves too early
        (two transitions in frame), so assume next transition is at 0.
187                 skip_frame <= 1'b0;
188              end
189           end
190           // Assign transition checks. Determine from previously found transitions whether th
        signal is consistent.
191           else begin // if (~RX_sampled_ready)
192              trans_zero <= (shift_reg[trans_reg_early] == shift_reg[trans_reg_zero]);
193              trans_late <= (shift_reg[trans_reg_early] == shift_reg[trans_reg_zero]) & (
        shift_reg[trans_reg_early] == shift_reg[trans_reg_late]);
194              trans_not_exist <= (shift_reg[trans_reg_early] == shift_reg[trans_reg]);
195              cnsy_one <= shift_reg[trans_reg_cnsy1[0]] & shift_reg[trans_reg_cnsy1[1]] &
        shift_reg[trans_reg_cnsy1[2]] & shift_reg[trans_reg_cnsy1[3]] & shift_reg[trans_reg_cnsy1
        ]];
196              cnsy_zero <= (~shift_reg[trans_reg_cnsy1[0]] & ~shift_reg[trans_reg_cnsy1[1]] &
        shift_reg[trans_reg_cnsy1[2]] & ~shift_reg[trans_reg_cnsy1[3]] & ~shift_reg[trans_reg_cns
        4]]);
197              check_cnsy <= ~skip_frame;
198              skip_frame <= 1'b0;
199              init_trans_reg <= (~shift_reg[10]) ? 3'd7: (~shift_reg[9]) ? 3'd6: (~shift_reg[
        ? 3'd5: (~shift_reg[7]) ? 3'd4:(~shift_reg[6]) ? 3'd3: 3'd0;
200           end
201        end
202
203        // Instantiate sample value registers
204        reg recov_wr_en;
205        reg recovered_signal;
206
207        // Recover bit stream, check for signal consistency
208        reg CD_signal;
209        reg recov_wr_en_override;
210        always @(posedge clk_40mhz) begin
211           recovered_signal <= cnsy_one;
212           recov_wr_en <= (cnsy_one | cnsy_zero) & check_cnsy & recov_wr_en_override;
213           CD_signal <= (~(cnsy_one | cnsy_zero)) & check_cnsy;
214        end
215
216        // Demultiplex recovered bit stream into 10 bit words for 8b/10b Decoder
217        wire demux_reset, demux_nd;
218        reg manual_demux_reset;
219        assign demux_reset = reset | manual_demux_reset;
220        wire [9:0] demux_dout;
221        assign demux_out = demux_dout;
222        PHY_Demux PHY_DEMUX_1(
223           .clk_40mhz(clk_40mhz),
224           .demux_reset(demux_reset),
225           .demux_din(recovered_signal),
226           .demux_wr_en(recov_wr_en),
227           .demux_dout(demux_dout),
228           .demux_nd(demux_nd)
229           );
230
231        // Count runs of ones
```

```
232         reg [2:0] runs_of_ones;
233         wire ro_reset;
234         reg manual_ro_reset;
235         always @(posedge clk_40mhz) begin
236            if (ro_reset)
237               runs_of_ones <= 3'd0;
238            else if (~RX_sampled_ready)
239               runs_of_ones <= (~(&shift_reg[7:0])) ? 3'd0: (runs_of_ones == 7) ? 3'd7: (
         runs_of_ones + 3'd1);
240         end
241
242         // Assign runs of ones reset
243         assign ro_reset = reset | manual_ro_reset;
244
245         // Count runs of zeros
246         reg [2:0] runs_of_zeros;
247         wire rz_reset;
248         reg manual_rz_reset;
249         always @(posedge clk_40mhz) begin
250            if (rz_reset)
251               runs_of_zeros <= 3'd0;
252            else if (~RX_sampled_ready)
253               runs_of_zeros <= (|shift_reg[7:0]) ? 3'd0: (runs_of_zeros == 7) ? 3'd7: (
         runs_of_zeros + 3'd1);
254         end
255
256         // Assign runs of ones reset
257         assign rz_reset = reset | manual_rz_reset;
258
259         ////////////////////////////////////////////////////////////////////////////////
260         // I/O BUFFERS & ENC/DEC
261         ////////////////////////////////////////////////////////////////////////////////
262
263         // Instantiate asymmetric aspect ratio output FIFO
264         wire out_rst;
265         wire out_wr_en;
266         wire out_almost_empty;
267         wire out_rd_en;
268         PHY_OUT_FIFO PHY_OUT_FIFO_1(
269            .clk(clk_40mhz),
270            .rst(out_rst),
271            .din(decoder_dout),
272            .wr_en(out_wr_en),
273            .dout(D_RX),
274            .rd_en(out_rd_en),
275            .almost_empty(out_almost_empty)
276            );
277
278         // Assign output FIFO inputs
279         reg out_rst_reg;
280         reg out_rd_en_reg;
281         assign out_rst = out_rst_reg;
282         assign out_rd_en = out_rd_en_reg;
283
284         // Instantiate FWFT input FIFO
285         wire in_rd_en;
286         wire in_rst;
287         wire [9:0] in_dout;
288         wire in_empty;
289         PHY_IN_FIFO PHY_IN_FIFO_1(
290            .clk(clk_40mhz),
```

```
291            .rst(in_rst),
292            .din(encoder_dout),
293            .dout(in_dout),
294            .rd_en(in_rd_en),
295            .wr_en(encoder_nd),
296            .empty(in_empty)
297            );
298
299         // Assign input FIFO inputs
300         reg in_rd_en_reg;
301         reg in_rst_reg;
302         assign in_rd_en = in_rd_en_reg;
303         assign in_rst = reset | in_rst_reg;
304
305         // Instantiate secondary input FIFO for comparision
306         wire [9:0] comp_dout;
307         wire comp_overflow, comp_empty;
308         PHY_small_FIFO PHY_SMALL_FIFO_1(
309            .clk_40mhz(clk_40mhz),
310            .reset(in_rst),
311            .din(in_dout),
312            .dout(comp_dout),
313            .wr_en(in_rd_en_reg),
314            .rd_en(demux_out_ready),
315            .overflow_err(comp_overflow),
316            .empty(comp_empty)
317            );
318
319         // Instantiate 8b/10b Decoder I/O registers
320         reg demux_out_ready_override;
321         reg decoder_init_reg;
322         reg out_wr_en_override;
323         assign demux_out_ready = demux_nd & demux_out_ready_override;
324         assign out_wr_en = (decoder_dout_ready & (~decoder_kout) & out_wr_en_override);
325         assign decoder_init = decoder_init_reg;
326
327         // Instantiate 8b/10b Encoder I/O registers
328         reg encoder_ce_reg;
329         reg encoder_force_code_reg;
330         reg encoder_kin_reg;
331         reg [7:0] encoder_din_reg;
332         assign encoder_ce = encoder_ce_reg;
333         assign encoder_force_code = encoder_force_code_reg;
334         assign encoder_kin = encoder_kin_reg;
335         always @(posedge clk_40mhz) encoder_din_reg <= (D_TX_ready) ? D_TX: 8'b00011100;
336         assign encoder_din = encoder_din_reg;
337
338         ////////////////////////////////////////////////////////////////////////////////
339         // TX LOGIC
340         ////////////////////////////////////////////////////////////////////////////////
341
342         // Generate 20Mhz clock
343         reg clk_20mhz;
344         always @(posedge clk_40mhz) clk_20mhz <= (reset) ? 1'b0: ~clk_20mhz;
345
346         // Drive TCV TX as needed
347         reg [3:0] data_to_send;
348         reg [9:0] TX_shift_reg;
349         reg TX_started;
350         reg TX_drive_low;
351         always @(posedge clk_40mhz) begin
```

```
352             if (reset | in_rst) begin
353                TCV_TX_en_reg <= (TX_drive_low) ? 1'b1: 1'b0;
354                TCV_TX_reg <= (TX_drive_low) ? 1'b0: 1'b1;
355                data_to_send <= 4'd9;
356                TX_started <= 1'b0;
357                TX_shift_reg <= 10'b0101010101;
358             end
359             else if (clk_20mhz) begin
360                if (~TX_started & ~in_empty) begin
361                   TCV_TX_en_reg <= 1'b1;
362                   TX_started <= 1'b1;
363                   data_to_send <= 4'd9;
364                   TCV_TX_reg <= 1'b1;
365                   TX_shift_reg <= 10'b0010101010;
366                end
367                else if (!data_to_send & in_rd_en_reg & TX_started) begin
368                   data_to_send <= 4'd9;
369                   TCV_TX_reg <= in_dout[0];
370                   TX_shift_reg <= {1'b0, in_dout[9:1]};
371                end
372                else if (data_to_send && TX_started) begin
373                   TX_shift_reg <= {1'b0, TX_shift_reg[9:1]};
374                   data_to_send <= data_to_send - 1;
375                   TCV_TX_reg <= TX_shift_reg[0];
376                end
377                else begin
378                   TCV_TX_en_reg <= 1'b0;
379                   TCV_TX_reg <= 1'b1;
380                   data_to_send <= 4'd9;
381                   TX_started <= 1'b0;
382                   TX_shift_reg <= 10'b0101010101;
383                end
384             end
385          end
386
387          // Set read enable from FWFT input FIFO
388          always @(posedge clk_40mhz) begin
389             if (reset | in_rst) begin
390                in_rd_en_reg <= 1'b0;
391             end
392             else if (~in_rd_en_reg & (~in_empty) & (data_to_send == 0) & TX_started) begin
393                in_rd_en_reg <= 1'b1;
394             end
395             else begin
396                in_rd_en_reg <= 1'b0;
397             end
398          end
399
400          // Compare when needed to detect transmission errors
401          reg CD_comp;
402          always @(posedge clk_40mhz) begin
403             if (reset | in_rst) begin
404                CD_comp <= 1'b0;
405             end
406             else if ((demux_out_ready & (comp_dout != demux_out)) | comp_overflow)  begin
407                CD_comp <= 1'b1;
408             end
409          end
410
411          ////////////////////////////////////////////////////////////////////////////////
412          // STATE MACHINES
```

```
413     ////////////////////////////////////////////////////////////////////////////////
414
415     // Instantiate receiver state
416     reg [2:0] rcv_state;
417
418     // Declare receiver state parameters
419     parameter SR_RST_WAIT    = 3'h0;
420     parameter SR_IB           = 3'h1;
421     parameter SR_RX           = 3'h2;
422     parameter SR_RX_1         = 3'h3;
423     parameter SR_RX_SUCCESS   = 3'h4;
424     parameter SR_RX_DRIVE_LOW = 3'h5;
425     parameter SR_RX_FAIL      = 3'h6;
426
427     // Instantiate main state
428     reg [3:0] state;
429
430     // Declare main state parameters
431     parameter S_IDLE       = 4'h0;
432     parameter S_TX         = 4'h1;
433     parameter S_TX_1       = 4'h2;
434     parameter S_TX_2       = 4'h3;
435     parameter S_TX_FAIL    = 4'h4;
436     parameter S_TX_SUCCESS = 4'h5;
437     parameter S_RX         = 4'h6;
438     parameter S_RX_1       = 4'h7;
439     parameter S_WAIT       = 4'h8;
440
441     // Manage receiver rcv_state transitions
442     reg [5:0] out_byte_cnt;
443     reg D_RX_ready_reg_l;
444     wire D_RX_ready_reg_fall;
445     always @(posedge clk_40mhz) D_RX_ready_reg_l <= (reset) ? 0: D_RX_ready_reg;
446     assign D_RX_ready_reg_fall = ~D_RX_ready_reg & D_RX_ready_reg_l;
447
448     always @(posedge clk_40mhz) begin
449        if (reset) begin
450           rcv_state <= SR_RST_WAIT;
451           manual_demux_reset <= 1'b1;
452           recov_wr_en_override <= 1'b0;
453           init_trans_detect <= 1'b1;
454           decoder_init_reg <= 1'b1;
455           demux_out_ready_override <= 1'b0;
456           out_rst_reg <= 1'b1;
457           out_byte_cnt <= 6'd0;
458           TX_drive_low <= 0;
459        end
460        else begin
461           case (rcv_state)
462              SR_RST_WAIT:   begin  // wait until sampler is ready
463                 if (RX_sampled_ready) begin
464                    rcv_state <= SR_RX_FAIL;
465                    manual_ro_reset <= 1'b0;
466                    manual_rz_reset <= 1'b0;
467                 end
468                 else begin
469                    rcv_state <= SR_RST_WAIT;
470                    manual_ro_reset <= 1'b1;
471                    manual_rz_reset <= 1'b1;
472                 end
473              end
```

```
474                 SR_IB: begin  // bus idle, wait for first 1->0 transition
475                    if (init_trans_detected) begin
476                       manual_demux_reset <= 1'b0;
477                       recov_wr_en_override <= 1'b1;
478                       init_trans_detect <= 1'b0;
479                       decoder_init_reg <= 1'b0;
480                       rcv_state <= SR_RX;
481                    end
482                 end
483                 SR_RX: begin  // test if first 10 bits are 10'b1010101010
484                    if (CD_signal) begin
485                       rcv_state <= SR_RX_DRIVE_LOW;
486                    end
487                    else if (demux_nd) begin  // on first sample
488                       if (demux_dout == 10'b0101010101) begin
489                          demux_out_ready_override <= 1'b1;
490                          out_wr_en_override <= 1'b1;
491                          out_rst_reg <= 1'b0;
492                          rcv_state <= SR_RX_1;
493                          out_byte_cnt <= 6'd0;
494                       end
495                       else
496                          rcv_state <= SR_RX_DRIVE_LOW;
497                    end
498                 end
499                 SR_RX_1: begin   // wait until either end character or decoder/signal error
500                    if (CD_signal | CD_comp) begin
501                       rcv_state <= SR_RX_DRIVE_LOW;
502                       out_wr_en_override <= 1'b0;
503                       out_rst_reg <= 1'b1;
504                    end
505                    else if (decoder_dout_ready) begin
506                       out_byte_cnt <= (decoder_kout) ? out_byte_cnt: (out_byte_cnt + 1);
507                       if (CD_decoder) begin
508                          rcv_state <= SR_RX_DRIVE_LOW;
509                          out_wr_en_override <= 1'b0;
510                          out_rst_reg <= 1'b1;
511                       end
512                       else if (decoder_kout & (decoder_dout == 8'b00011100)) begin
513                          rcv_state <= SR_RX_SUCCESS;
514                          out_wr_en_override <= 1'b0;
515                       end
516                    end
517                 end
518                 SR_RX_SUCCESS: begin  // go to fail rcv_state when finished, wait until idle
        bus. Also, manage deadlocks with other FSM.
519                    if (D_RX_ready_reg_fall | TX_success_reg | (state == S_TX_FAIL) | (state
        S_IDLE)) begin
520                       rcv_state <= SR_RX_FAIL;
521                       out_byte_cnt <= 6'd0;
522                       out_rst_reg <= 1'b1;
523                    end
524                 end
525                 SR_RX_DRIVE_LOW: begin // drive the signal low until seven cycles of zeros
526                    if (runs_of_zeros == 3'd7) begin
527                       rcv_state <= SR_RX_FAIL;
528                       TX_drive_low <= 0;
529                    end
530                    else begin
531                       TX_drive_low <= 1;
532                    end
```

```
533                    end
534                    SR_RX_FAIL: begin  // wait until seven cycles of ones before declaring link 
535                       manual_demux_reset <= 1'b1;
536                       recov_wr_en_override <= 1'b0;
537                       init_trans_detect <= 1'b1;
538                       decoder_init_reg <= 1'b1;
539                       demux_out_ready_override <= 1'b0;
540                       out_byte_cnt <= 6'd0;
541                       rcv_state <= ((runs_of_ones == 3'd7) & ~D_TX_ready) ? SR_IB: SR_RX_FAIL;
542                    end
543                 endcase
544              end
545           end
546
547           // Manage main state transitions
548           always @(posedge clk_40mhz) begin
549              if (reset) begin
550                 D_RX_ready_reg <= 1'b0;
551                 TX_success_reg <= 1'b0;
552                 IB_reg <= 1'b0;
553                 CD_reg <= 1'b0;
554                 encoder_ce_reg <= 1'b0;
555                 encoder_force_code_reg <= 1'b1;
556                 encoder_kin_reg <= 1'b0;
557                 in_rst_reg <= 1'b1;
558                 out_rd_en_reg <= 1'b0;
559                 state <= S_WAIT;
560              end
561              else begin
562                 case (state)
563                    S_WAIT: begin
564                       if (rcv_state != SR_RST_WAIT) begin
565                          state <= S_IDLE;
566                       end
567                    end
568                    S_IDLE: begin
569                       D_RX_ready_reg <= 1'b0;
570                       encoder_kin_reg <= 1'b0;
571                       in_rst_reg <= 1'b1;
572                       out_rd_en_reg <= 1'b0;
573                       CD_reg <= 1'b0;
574                       TX_success_reg <= 1'b0;
575                       if ((init_trans_detected && (rcv_state == SR_IB)) || (rcv_state == SR_RX))
          begin
576                          state <= S_RX;
577                          encoder_force_code_reg <= 1'b1;
578                          encoder_ce_reg <= 1'b0;
579                          IB_reg <= 1'b0;
580                       end
581                       else if (IB_reg & D_TX_ready) begin
582                          state <= S_TX;
583                          encoder_force_code_reg <= 1'b0;
584                          encoder_ce_reg <= 1'b1;
585                          IB_reg <= 1'b0;
586                       end
587                       else begin
588                          encoder_force_code_reg <= 1'b1;
589                          encoder_ce_reg <= 1'b0;
590                          IB_reg <= (rcv_state == SR_IB);
591                       end
592                    end
```

```
593                 S_TX: begin  // Take care of last character being special character (to show
594                    in_rst_reg <= 1'b0;
595                    if (CD_signal | CD_comp | (rcv_state == SR_RX_FAIL) | (rcv_state ==
            SR_RX_DRIVE_LOW)) begin
596                       state <= S_TX_FAIL;
597                       in_rst_reg <= 1'b1;
598                       encoder_force_code_reg <= 1'b1;
599                       encoder_ce_reg <= 1'b0;
600                    end
601                    else if (~D_TX_ready) begin
602                       encoder_kin_reg <= 1'b1;
603                       state <= S_TX_1;
604                    end
605                 end
606                 S_TX_1: begin  // Waiting on last special character to be encoded
607                    encoder_force_code_reg <= 1'b1;
608                    encoder_ce_reg <= 1'b0;
609                    if (CD_signal | CD_comp | (rcv_state == SR_RX_FAIL) | (rcv_state ==
            SR_RX_DRIVE_LOW)) begin
610                       state <= S_TX_FAIL;
611                       in_rst_reg <= 1'b1;
612                    end
613                    else begin
614                       state <= S_TX_2;
615                    end
616                 end
617                 S_TX_2: begin  // Waiting on input FIFOs to become empty
618                    if (CD_signal | CD_comp| (rcv_state == SR_RX_FAIL) | (rcv_state ==
            SR_RX_DRIVE_LOW)) begin
619                       state <= S_TX_FAIL;
620                       in_rst_reg <= 1'b1;
621                    end
622                    else if (in_empty & comp_empty) begin
623                       state <= S_TX_SUCCESS;
624                       in_rst_reg <= 1'b1;
625                    end
626                 end
627                 S_TX_FAIL: begin
628                    CD_reg <= 1'b1;
629                    if (~D_TX_ready & ((rcv_state == SR_IB) | (rcv_state == SR_RX_FAIL))) beg
630                       state <= S_IDLE;
631                    end
632                 end
633                 S_TX_SUCCESS: begin
634                    if (rcv_state == SR_RX_SUCCESS) begin
635                       TX_success_reg <= 1'b1;
636                       state <= S_IDLE;
637                    end
638                 end
639                 S_RX: begin  // Look at receiver state machine to determine behavior
640                    if (rcv_state == SR_RX_SUCCESS) begin
641                       out_rd_en_reg <= 1'b1;
642                       state <= state + 1;
643                    end
644                    else if ((rcv_state == SR_RX_FAIL) | (rcv_state == SR_RX_DRIVE_LOW)) begi
645                       state <= S_IDLE;
646                    end
647                 end
648                 S_RX_1: begin
649                    D_RX_ready_reg <= 1'b1;
650                    if (out_almost_empty) begin
```

```
651                         state <= S_IDLE;
652                         out_rd_en_reg <= 1'b0;
653                      end
654                   end
655                endcase
656             end
657          end
658
659          // DEBUG
660          assign rcv_state_DEBUG = rcv_state;
661          assign state_DEBUG = state;
662          assign in_rd_en_DEBUG = in_rd_en;
663          assign in_dout_DEBUG = in_dout;
664          assign in_rst_DEBUG = in_rst;
665          assign in_empty_DEBUG = in_empty;
666          assign data_to_send_DEBUG = data_to_send;
667          assign TX_shift_reg_DEBUG = TX_shift_reg;
668          assign clk_20mhz_DEBUG = clk_20mhz;
669          assign TX_started_DEBUG = TX_started;
670          assign shift_reg_DEBUG = shift_reg;
671          assign RX_sampled_ready_DEBUG = RX_sampled_ready;
672          assign cnsy_zero_DEBUG = cnsy_zero;
673          assign cnsy_one_DEBUG = cnsy_one;
674          assign check_cnsy_DEBUG = check_cnsy;
675          assign trans_reg_DEBUG = trans_reg;
676          assign init_trans_detected_DEBUG = init_trans_detected;
677          assign trans_zero_DEBUG = trans_zero;
678          assign trans_late_DEBUG = trans_late;
679          assign trans_not_exist_DEBUG = trans_not_exist;
680          assign init_trans_detect_DEBUG = init_trans_detect;
681          assign demux_nd_DEBUG = demux_nd;
682          assign demux_dout_DEBUG = demux_dout;
683          assign CD_comp_DEBUG = CD_comp;
684          assign CD_signal_DEBUG = CD_signal;
685          assign CD_decoder_DEBUG = CD_decoder;
686          assign comp_dout_DEBUG = comp_dout;
687          // END DEBUG
688       endmodule
689
690       ////////////////////////////////////////////////////////////////////////////////////
691       // DEMUX MODULES
692       ////////////////////////////////////////////////////////////////////////////////////
693
694       // General purpose demultiplexer with:
695       // o 1-bit wide data input
696       // o write enable
697       // o synchronous reset
698       // o new data signal
699       module PHY_Demux #(
700          parameter P_DEMUX_SIZE = 10,     // Size of Demux
701          parameter P_LOG_DEMUX_SIZE = 4   // Log Size of Demux
702          )(
703          input clk_40mhz,
704          input demux_reset,                        // Demux Synchronous Reset
705          input demux_din,                          // Demux Data Input
706          input demux_wr_en,                        // Demux Write Enable
707          output [P_DEMUX_SIZE-1:0] demux_dout,     // Demux Data Output
708          output demux_nd                           // Demux New Data (at Output)
709          );
710
711          // Instantiate state vars
```

```
712          reg [P_LOG_DEMUX_SIZE-1:0] demux_ind_reg;
713          reg [P_DEMUX_SIZE-1:0] demux_data_reg;
714          reg demux_nd_reg;
715          reg demux_start_reg;
716
717          // Assign outputs
718          assign demux_dout = demux_data_reg;
719          assign demux_nd = demux_nd_reg;
720
721          // Manage state transitions
722          always @(posedge clk_40mhz) begin
723             if (demux_reset) begin
724                demux_ind_reg <= 0;
725                demux_data_reg <= 0;
726                demux_nd_reg <= 1'b0;
727                demux_start_reg <= 1'b1;
728             end
729             else begin
730                if (demux_wr_en) begin
731                   demux_ind_reg <= (demux_ind_reg == (P_DEMUX_SIZE-1)) ? 0: (demux_ind_reg + 1
732                   demux_data_reg[demux_ind_reg] <= demux_din;
733                end
734                if (demux_ind_reg == 1) demux_start_reg <= 1'b0;
735                demux_nd_reg <= (demux_start_reg) ? 1'b0: (demux_wr_en & (demux_ind_reg == (
       P_DEMUX_SIZE-1)));
736             end
737          end
738       endmodule
739
740       // FWFT FIFO with depth 3, variable width
741       module PHY_small_FIFO #(
742          parameter P_FIFO_WIDTH = 10,
743          parameter P_LOG_FIFO_DEPTH = 1
744          )(
745          input clk_40mhz,
746          input reset,
747          input [(P_FIFO_WIDTH - 1):0] din,
748          output [(P_FIFO_WIDTH - 1):0] dout,
749          input wr_en,
750          input rd_en,
751          output overflow_err,
752          output empty
753          );
754
755          // Instantiate state vars
756          reg [1:0] data_cnt;
757          reg [(P_FIFO_WIDTH-1): 0] mem [0:2];
758
759          // Assign output
760          assign dout = mem[0];
761          assign empty = (data_cnt == 2'd0);
762          assign overflow_err = (wr_en & (~rd_en) & (data_cnt == 2'd3));
763
764          // Manage memory changes
765          always @(posedge clk_40mhz) begin
766             if (reset) begin
767                data_cnt <= 2'd0;
768                mem[0] <= 0;
769                mem[1] <= 0;
770                mem[2] <= 0;
771             end
```

```
772             else begin
773                case ({wr_en, rd_en})
774                   2'b10: begin
775                      if (data_cnt != 2'd3) begin
776                         mem[data_cnt] <= din;
777                         data_cnt <= data_cnt + 1;
778                      end
779                   end
780                   2'b01: begin
781                      mem[0] <= mem[1];
782                      mem[1] <= mem[2];
783                      data_cnt <= (!data_cnt) ? 2'd0: (data_cnt - 1);
784                   end
785                   2'b11: begin
786                      if (data_cnt == 2'd1) begin
787                         mem[0] <= din;
788                      end
789                      else if (data_cnt == 2'd2) begin
790                         mem[0] <= mem[1];
791                         mem[1] <= din;
792                      end
793                      else if (data_cnt == 2'd3) begin
794                         mem[0] <= mem[1];
795                         mem[1] <= mem[2];
796                         mem[2] <= din;
797                      end
798                   end
799                   default:;  // Do nothing otherwise
800                endcase
801             end
802          end
803       endmodule
804
```

```
1     `timescale 1ns / 1ps
2     ////////////////////////////////////////////////////////////////////////////////
3     // Company:
4     // Engineer:       Sachin Shinde
5     //
6     // Create Date:    05:53:42 11/12/2012
7     // Design Name:
8     // Module Name:    PHY_RX_Sample
9     // Project Name:
10    // Target Devices:
11    // Tool versions:
12    // Description: Take samples at 160Mhz, output as 8-bit bus at 40Mhz w/ ready
13    //              signal (ready signal initially low after reset, then high every
14    //              other cycle). New data available on rise of the ready.
15    //
16    // Dependencies:
17    //
18    // Revision:
19    // Revision 0.01 - File Created
20    // Additional Comments:
21    //
22    ////////////////////////////////////////////////////////////////////////////////
23    module PHY_RX_Sample(
24       input clk_40mhz,
25       input clk_160mhz,
26       input reset,
27       input RX,
28       output reg out_ready,
29       output [7:0] RX_sampled
30    // ,// DEBUG OUTPUTS
31    // output RX_stable_DEBUG,
32    // output empty_DEBUG,
33    // output [1:0] rd_d_cnt_DEBUG,
34    // output start_DEBUG
35    // // END DEBUG OUTPUTS
36       );
37
38       // Reduce odds of metastability in raw input signal
39       reg RX_1, RX_2, RX_stable;
40       always @(posedge clk_160mhz) begin
41          RX_1 <= RX;
42          RX_2 <= RX_1;
43          RX_stable <= RX_2;
44       end
45
46       // Generate 1:8 Asymmetric Asynchronous FWFT FIFO for demultiplexing
47       reg rd_en, wr_en;
48       wire [7:0] RX_DMX;
49       wire empty;
50       wire [4:0] rd_d_cnt;
51       PHY_RX_FIFO_1 RX_FIFO_1(
52          .din(RX_stable),
53          .rd_clk(clk_40mhz),
54          .rd_en(rd_en),
55          .rst(reset),
56          .wr_clk(clk_160mhz),
57          .wr_en(wr_en),
58          .dout(RX_DMX),
59          .empty(empty),
60          .rd_data_count(rd_d_cnt)
61       );
```

```
62
63          // Read from FIFO
64          reg start;  // Needed to buffer some data so 40Mhz output is continuous
65          always @(posedge clk_40mhz) begin
66             if (reset) begin
67                out_ready <= 1'b0;
68                wr_en <= 1'b0;
69                rd_en <= 1'b0;
70                start <= 1'b1;
71             end
72             else if (start) begin
73                out_ready <= 1'b0;
74                rd_en <= 1'b0;
75                wr_en <= 1'b1;
76                start <= (rd_d_cnt < 5'd2);
77             end
78             else begin
79                out_ready <= rd_en;
80                rd_en <= ~rd_en;
81             end
82          end
83
84          // Assign output
85          assign RX_sampled = RX_DMX;
86
87          // DEBUG ASSIGN
88       // assign RX_stable_DEBUG = RX_stable;
89       // assign empty_DEBUG = empty;
90       // assign rd_d_cnt_DEBUG = rd_d_cnt;
91       // assign start_DEBUG = start;
92          // END DEBUG ASSIGN
93
94       endmodule
95
```

```
1        `timescale 1ns / 1ps
2        ////////////////////////////////////////////////////////////////////////////////
3        // Company:
4        // Engineer:       Sachin Shinde
5        //
6        // Create Date:    12:07:55 12/07/2012
7        // Design Name:
8        // Module Name:    DLC
9        // Project Name:
10       // Target Devices:
11       // Tool versions:
12       // Description:
13       //
14       // Dependencies:
15       //
16       // Revision:
17       // Revision 0.01 - File Created
18       // Additional Comments:
19       //
20       ////////////////////////////////////////////////////////////////////////////////
21       module DLC #(
22          parameter P_INIT_SEND_MAX = 15,
23          parameter P_INIT_NO_ACK_TIMEOUT = 100000,
24          parameter P_INIT_TIMEOUT = 1000000,
25          parameter P_ACK_TIMEOUT = 100000
26          )(
27          input clk_40mhz,              // 40MHz Clock
28          input reset,                  // Active-High Reset
29          // PHY I/O
30          output [7:0] PHY_TX,           // PHY Data TX
31          output PHY_TX_ready,             // PHY Data TX Ready
32          input [7:0] PHY_RX,            // PHY Data RX
33          input PHY_RX_ready,              // PHY Data RX Ready
34          input PHY_CD,                    // Collision Detect (while Sending)
35          input PHY_TX_success,            // Successful transmission
36          input PHY_IB,                    // Idle Bus
37          // Network I/O
38          input [1:0] cmd,               // DLC command from status
39          output link_sts,                 // Active High for Up
40          output [1:0] sts,               // DLC status
41          input [7:0] self_phn_num,       // Self Phone Number
42          input [7:0] D_TX,               // outgoing data port
43          input D_TX_ready,                 // outgoing ready
44          input [7:0] D_TX_addr,        // phone number for transmission
45          output [7:0] D_RX,              // incoming data port
46          output D_RX_ready,                // incoming ready
47          output [7:0] D_RX_addr,          // phone number for reception
48          // PRNG I/O
49          input [63:0] rand_num            // 64-bit random number
50          ,// DEBUG
51          output [5:0] init_state_DEBUG,
52          output [5:0] pckt_size_DEBUG,
53          output CRC_E_we_DEBUG,
54          output [10:0] CRC_E_addr_DEBUG,
55          output [7:0] CRC_E_data_DEBUG,
56          output [7:0] MAC_data_DEBUG,
57          output [10:0] MAC_addr_DEBUG,
58          output [2:0] MAC_state_DEBUG,
59          output [4:0] b_state_DEBUG,
60          output [4:0] f_state_DEBUG,
61          output [2:0] s_state_DEBUG,
```

```
 62          output [3:0] c_state_DEBUG,
 63          output b_f_lock_DEBUG,
 64          output b_b_lock_DEBUG,
 65          output b_c_lock_DEBUG,
 66          output f1w_s_lock_DEBUG,
 67          output f1w_b_lock_DEBUG,
 68          output f1w_c_lock_DEBUG,
 69          output f1r_f_lock_DEBUG,
 70          output f1r_b_lock_DEBUG,
 71          output f2w_f_lock_DEBUG,
 72          output f2w_b_lock_DEBUG,
 73          output f2w_c_lock_DEBUG ,
 74          output [4:0] MAC_pckt_num_DEBUG,
 75          output MAC_sts_DEBUG,
 76          output MAC_cmd_DEBUG,
 77          output CRC_sts_DEBUG,
 78          output [7:0] CRC_D_TX_DEBUG,
 79          output CRC_D_TX_ready_DEBUG,
 80          output [4:0] CRC_pckt_num_DEBUG,
 81          output CRC_D_RX_ready_DEBUG,
 82          output [7:0] CRC_D_RX_DEBUG,
 83          output p_DEBUG,
 84          output [15:0] dina_DEBUG,
 85          output wea_DEBUG,
 86          output [7:0] addra_DEBUG,
 87          output [15:0] douta_DEBUG,
 88          output [4:0] rd_pointer_DEBUG,
 89          output [4:0] wr_pointer_DEBUG,
 90          output [2:0] CRC_state_DEBUG
 91          );
 92
 93
 94          // Define link status
 95          reg link_up;
 96          assign link_sts = link_up;
 97
 98          // Generate serial random number signal
 99          wire rand_num_serial;
100          assign rand_num_serial = rand_num[0];
101
102          // Instantiate Media Access Controller Module
103          wire MAC_sts;
104          wire MAC_cmd, MAC_cmd_ARQ;
105          reg MAC_cmd_INIT_reg;
106          assign MAC_cmd = (link_up) ? MAC_cmd_ARQ: MAC_cmd_INIT_reg;
107          wire [4:0] MAC_pckt_num, MAC_pckt_num_ARQ;
108          reg [4:0] MAC_pckt_num_INIT_reg;
109          assign MAC_pckt_num = (link_up) ? MAC_pckt_num_ARQ: MAC_pckt_num_INIT_reg;
110          wire [7:0] MAC_data;
111          wire [10:0] MAC_addr;
112          MAC MAC_1(
113             .clk_40mhz(clk_40mhz),
114             .reset(reset),
115             // Main I/O
116             .cmd(MAC_cmd),
117             .sts(MAC_sts),
118             .pckt_num(MAC_pckt_num),
119             // PHY I/O
120             .PHY_TX(PHY_TX),
121             .PHY_TX_ready(PHY_TX_ready),
122             .PHY_CD(PHY_CD),
```

```
123                 .PHY_TX_success(PHY_TX_success),
124                 .PHY_IB(PHY_IB),
125                 // Packet Storage I/O
126                 .data(MAC_data),
127                 .addr(MAC_addr),
128                 // PRNG I/O
129                 .rand_num_serial(rand_num_serial)
130                 ,// DEBUG
131                 .pckt_size_DEBUG(pckt_size_DEBUG),
132                 .MAC_state_DEBUG(MAC_state_DEBUG)
133                 );
134
135              // Instantiate Block RAM for outgoing packets
136              wire CRC_E_we;
137              wire [10:0] CRC_E_addr;
138              wire [7:0] CRC_E_data;
139              DLC_BRAM_TX DLC_BRAM_TX_1(
140                 .clka(clk_40mhz),
141                 .clkb(clk_40mhz),
142                 .addra(CRC_E_addr),
143                 .dina(CRC_E_data),
144                 .wea(CRC_E_we),
145                 .addrb(MAC_addr),
146                 .doutb(MAC_data)
147                 );
148
149              // Instantiate CRC16-ANSI to write to Outgoing BRAM
150              wire CRC_sts;
151              wire [7:0] CRC_D_TX, CRC_D_TX_ARQ;
152              wire CRC_D_TX_ready, CRC_D_TX_ready_ARQ;
153              wire [4:0] CRC_pckt_num, CRC_pckt_num_ARQ;
154              CRC_Enc CRC_ENC_1(
155                 .clk_40mhz(clk_40mhz),
156                 .reset(reset),
157                 // Data I/O
158                 .sts(CRC_sts),
159                 .D_TX(CRC_D_TX),
160                 .D_TX_ready(CRC_D_TX_ready),
161                 .pckt_num(CRC_pckt_num),
162                 // BRAM I/O
163                 .we(CRC_E_we),
164                 .addr(CRC_E_addr),
165                 .data(CRC_E_data)
166                 );
167
168              reg [7:0] CRC_D_TX_INIT_reg;
169              reg CRC_D_TX_ready_INIT_reg;
170              reg [4:0] CRC_pckt_num_INIT_reg;
171              assign CRC_D_TX = (link_up) ? CRC_D_TX_ARQ: CRC_D_TX_INIT_reg;
172              assign CRC_D_TX_ready = (link_up) ? CRC_D_TX_ready_ARQ: CRC_D_TX_ready_INIT_reg;
173              assign CRC_pckt_num = (link_up) ? CRC_pckt_num_ARQ: CRC_pckt_num_INIT_reg;
174
175              // Instantiate CRC16-ANSI for incoming packets
176              wire [7:0] CRC_D_RX;
177              wire CRC_D_RX_ready;
178              CRC_Dec CRC_DEC_1(
179                 .clk_40mhz(clk_40mhz),
180                 .reset(reset),
181                 // PHY I/O
182                 .PHY_RX(PHY_RX),
183                 .PHY_RX_ready(PHY_RX_ready),
```

```
184             // Data I/O
185             .D_RX(CRC_D_RX),
186             .D_RX_ready(CRC_D_RX_ready)
187             ,// DEBUG
188             .rd_pointer_DEBUG(rd_pointer_DEBUG),
189             .wr_pointer_DEBUG(wr_pointer_DEBUG),
190             .CRC_state_DEBUG(CRC_state_DEBUG)
191             );
192
193         // Instantiate request buffer
194         wire [7:0] req;
195         wire [7:0] req_param;
196         reg req_rd;
197         wire [3:0] ignore;
198         reg [3:0] ignore_reg;
199         assign ignore = ignore_reg;
200         Request_Buffer REQUEST_BUFFER_1(
201         .clk_40mhz(clk_40mhz),
202         .reset(reset),
203         // DLC Control I/O
204         .req(req),                      // request type
205         .req_param(req_param),          // request paramter
206         .req_rd(req_rd),                // clear current request, load next
207         .self_phn_num(self_phn_num),
208         .ignore(ignore),
209         // CRC Checker I/O
210         .D_RX(CRC_D_RX),                   // incoming data port
211         .D_RX_ready(CRC_D_RX_ready)        // incoming ready
212         );
213
214         // Instantiate init state
215         reg [5:0] init_state;
216
217         // Declare init state parameters
218         parameter SI_WAIT_PHN_NUM          = 6'h00;
219         parameter SI_WAIT_INIT             = 6'h01;
220         parameter SI_MASTER_INIT           = 6'h02;
221         parameter SI_MASTER_INIT_1         = 6'h03;
222         parameter SI_MASTER_INIT_2         = 6'h04;
223         parameter SI_MASTER_INIT_3         = 6'h05;
224         parameter SI_MASTER_INIT_4         = 6'h06;
225         parameter SI_MASTER_INIT_5         = 6'h07;
226         parameter SI_MASTER_INIT_6         = 6'h08;
227         parameter SI_MASTER_INIT_7         = 6'h09;
228         parameter SI_MASTER_INIT_8         = 6'h0A;
229         parameter SI_MASTER_INIT_9         = 6'h0B;
230         parameter SI_MASTER_INIT_10        = 6'h0C;
231         parameter SI_MASTER_INIT_11        = 6'h0D;
232         parameter SI_MASTER_INIT_12        = 6'h0E;
233         parameter SI_MASTER_FINISH_INIT    = 6'h0F;
234         parameter SI_MASTER_FINISH_INIT_1 = 6'h10;
235         parameter SI_MASTER_FINISH_INIT_2 = 6'h11;
236         parameter SI_MASTER_FINISH_INIT_3 = 6'h12;
237         parameter SI_MASTER_FINISH_INIT_4 = 6'h13;
238         parameter SI_MASTER_FINISH_INIT_5 = 6'h14;
239         parameter SI_MASTER_FINISH_INIT_6 = 6'h15;
240         parameter SI_SLAVE_INIT            = 6'h16;
241         parameter SI_SLAVE_INIT_1          = 6'h17;
242         parameter SI_SLAVE_INIT_2          = 6'h18;
243         parameter SI_SLAVE_INIT_3          = 6'h19;
244         parameter SI_SLAVE_INIT_4          = 6'h1A;
```

```
245         parameter SI_SLAVE_INIT_5         = 6'h1B;
246         parameter SI_SLAVE_FINISH_INIT    = 6'h1C;
247         parameter SI_SLAVE_FINISH_INIT_1  = 6'h1D;
248         parameter SI_SLAVE_FINISH_INIT_2  = 6'h1E;
249         parameter SI_SLAVE_FINISH_INIT_3  = 6'h1F;
250         parameter SI_SLAVE_FINISH_INIT_4  = 6'h20;
251         parameter SI_SLAVE_FINISH_INIT_5  = 6'h21;
252         parameter SI_SLAVE_FINISH_INIT_6  = 6'h22;
253         parameter SI_INIT_SUCCESS         = 6'h23;
254         parameter SI_INIT_FAIL            = 6'h24;
255
256         // Define command parameters
257         parameter CMD_IDLE        = 2'd0;
258         parameter CMD_SET_PHN_NUM = 2'd1;
259         parameter CMD_INIT        = 2'd2;
260         parameter CMD_TX          = 2'd3;
261
262         // Define request parameters
263         parameter REQ_NONE            = 8'h00;
264         parameter REQ_INIT            = 8'h01;
265         parameter REQ_INIT_ACK        = 8'h02;
266         parameter REQ_INIT_ACK_ACK    = 8'h03;
267         parameter REQ_INIT_FINISH     = 8'h04;
268         parameter REQ_INIT_FINISH_ACK = 8'h05;
269         parameter REQ_DATA            = 8'h06;
270         parameter REQ_DATA_ACK        = 8'h07;
271
272         // Define status parameters
273         parameter STS_IDLE      = 2'd0;
274         parameter STS_BUSY      = 2'd1;
275         parameter STS_TX_ACCEPT = 2'd2;
276         parameter STS_TX_REJECT = 2'd3;
277
278         // Manage initialization state transitions
279         reg [19:0] cntr;
280         reg [3:0] init_cnt;
281         reg ACK_received;
282         reg [7:0] temp_phn_num;
283         always @(posedge clk_40mhz) begin
284            if (reset) begin
285               init_state <= SI_WAIT_PHN_NUM;
286               ignore_reg <= 5'b00000;
287               link_up <= 0;
288               cntr <= 0;
289               init_cnt <= 0;
290               ACK_received <= 0;
291            end
292            else begin
293               case (init_state)
294                  SI_WAIT_PHN_NUM: begin
295                     if (cmd == CMD_SET_PHN_NUM) begin
296                        init_state <= init_state + 1;
297                     end
298                  end
299                  SI_WAIT_INIT: begin
300                     if (req != REQ_NONE) begin // if remote init started
301                        req_rd <= 1;
302                        init_state <= SI_SLAVE_INIT;
303                        ignore_reg <= 5'b11010;  // Ignore ACK requests
304                     end
305                     else if (cmd == CMD_INIT) begin
```

```
306                         init_state <= SI_MASTER_INIT;
307                         init_cnt <= P_INIT_SEND_MAX;
308                      end
309                   end
310
311                   ///////////////////////////////////////////////////////////////////////////
312                   // MASTER INITIALIZATION PROTOCOL
313                   ///////////////////////////////////////////////////////////////////////////
314
315                   SI_MASTER_INIT: begin
316                      if ((req == REQ_INIT) && (req_param[7:0] < self_phn_num)) begin
317                         req_rd <= 1;
318                         init_state <= SI_SLAVE_INIT;
319                         ignore_reg <= 5'b11010;  // Ignore ACK requests
320                         cntr <= 0;
321                      end
322                      else begin
323                         req_rd <= 0;
324                         if (init_cnt) begin
325                            if (CRC_sts == 0) begin  // If encoder idle, write to buffer
326                               CRC_D_TX_ready_INIT_reg <= 1'b1;
327                               CRC_D_TX_INIT_reg <= REQ_INIT;
328                               CRC_pckt_num_INIT_reg <= 5'd0;
329                               init_state <= init_state + 1;
330                            end
331                         end
332                         else begin
333                            init_state <= SI_INIT_FAIL;
334                         end
335                      end
336                   end
337                   SI_MASTER_INIT_1: begin
338                      CRC_D_TX_INIT_reg <= self_phn_num;
339                      init_state <= init_state + 1;
340                   end
341                   SI_MASTER_INIT_2: begin
342                      CRC_D_TX_ready_INIT_reg <= 1'b0;
343                      if (CRC_sts == 0) init_state <= init_state + 1;
344                   end
345                   SI_MASTER_INIT_3: begin
346                      if (MAC_sts == 0) begin  // If MAC idle, send packet to it
347                         MAC_cmd_INIT_reg <= 1'b1;
348                         MAC_pckt_num_INIT_reg <= 5'd0;
349                         init_state <= init_state + 1;
350                      end
351                   end
352                   SI_MASTER_INIT_4: begin
353                      MAC_cmd_INIT_reg <= 1'b0;
354                      init_state <= init_state + 1;
355                   end
356                   SI_MASTER_INIT_5: begin
357                      if (MAC_sts == 0) begin // If MAC done sending
358                         init_state <= init_state + 1;
359                         cntr <= P_INIT_NO_ACK_TIMEOUT;
360                         ACK_received <= 1'b0;
361                      end
362                   end
363                   SI_MASTER_INIT_6: begin
364                      if ((req == REQ_INIT) && (req_param[7:0] < self_phn_num)) begin
365                         req_rd <= 1;
366                         init_state <= SI_SLAVE_INIT;
```

```
367                         ignore_reg <= 5'b11010;  // Ignore ACK requests
368                         cntr <= 0;
369                      end
370                      else if (req == REQ_INIT_ACK) begin // If INIT_ACK comes, send back
         INIT_ACK_ACK
371                         req_rd <= 1;
372                         temp_phn_num <= req_param[7:0];
373                         ACK_received <= 1'b1;
374                         init_state <= init_state + 1;
375                      end
376                      else if (req != REQ_NONE) begin
377                         req_rd <= 1;
378                      end
379                      else if (cntr) begin
380                         req_rd <= 0;
381                         cntr <= cntr - 1;
382                      end
383                      else begin // (!cntr)
384                         req_rd <= 0;
385                         if (ACK_received) begin // assume all FPGAs have ACKed, complete init
386                            init_state <= SI_MASTER_FINISH_INIT;
387                            ignore_reg <= 5'b01111;
388                            cntr <= 0;
389                         end
390                         else begin  // no FPGAs have ACKed, try sending INIT again until init
         count exhausted
391                            init_state <= SI_MASTER_INIT;
392                            init_cnt <= init_cnt - 1;
393                         end
394                      end
395                   end
396                   SI_MASTER_INIT_7: begin
397                      req_rd <= 0;
398                      if (CRC_sts == 0) begin // If CRC idle, send packet
399                         CRC_D_TX_ready_INIT_reg <= 1'b1;
400                         CRC_D_TX_INIT_reg <= REQ_INIT_ACK_ACK;
401                         CRC_pckt_num_INIT_reg <= 5'd0;
402                         init_state <= init_state + 1;
403                      end
404                   end
405                   SI_MASTER_INIT_8: begin
406                      CRC_D_TX_INIT_reg <= temp_phn_num;
407                      init_state <= init_state + 1;
408                   end
409                   SI_MASTER_INIT_9: begin
410                      CRC_D_TX_ready_INIT_reg <= 0;
411                      if (CRC_sts == 0) init_state <= init_state + 1;
412                   end
413                   SI_MASTER_INIT_10: begin
414                      if (MAC_sts == 0) begin  // If MAC idle, send packet to it
415                         MAC_cmd_INIT_reg <= 1'b1;
416                         MAC_pckt_num_INIT_reg <= 5'd0;
417                         init_state <= init_state + 1;
418                      end
419                   end
420                   SI_MASTER_INIT_11: begin
421                      MAC_cmd_INIT_reg <= 1'b0;
422                      init_state <= init_state + 1;
423                   end
424                   SI_MASTER_INIT_12: begin
425                      if (MAC_sts == 0) begin // If MAC done sending
```

```
426                         init_state <= SI_MASTER_INIT_6;
427                         cntr <= P_INIT_TIMEOUT;
428                      end
429                   end
430
431                   ////////////////////////////////////////////////////////////////////////////
432                   // MASTER INITIALIZATION FINISH PROTOCOL
433                   ////////////////////////////////////////////////////////////////////////////
434
435                   SI_MASTER_FINISH_INIT: begin
436                      if (req == REQ_INIT_FINISH_ACK) begin  // Received Finish ACK, success
437                         req_rd <= 1;
438                         init_state <= SI_INIT_SUCCESS;
439                         ignore_reg <= 5'b10111;
440                      end
441                      else if (req != REQ_NONE) begin
442                         req_rd <= 1;
443                      end
444                      else if (cntr) begin
445                         req_rd <= 0;
446                         cntr <= cntr - 1;
447                      end
448                      else begin  // Send init finish ACK
449                         req_rd <= 0;
450                         init_state <= init_state + 1;
451                      end
452                   end
453                   SI_MASTER_FINISH_INIT_1: begin
454                      req_rd <= 0;
455                      if (CRC_sts == 0) begin
456                         CRC_D_TX_ready_INIT_reg <= 1'b1;
457                         CRC_D_TX_INIT_reg <= REQ_INIT_FINISH;
458                         CRC_pckt_num_INIT_reg <= 5'd0;
459                         init_state <= init_state + 1;
460                      end
461                   end
462                   SI_MASTER_FINISH_INIT_2: begin
463                      CRC_D_TX_ready_INIT_reg <= 0;
464                      init_state <= init_state + 1;
465                   end
466                   SI_MASTER_FINISH_INIT_3: begin
467                      if (CRC_sts == 0) init_state <= init_state + 1;
468                   end
469                   SI_MASTER_FINISH_INIT_4: begin
470                      if (MAC_sts == 0) begin  // If MAC idle, send packet to it
471                         MAC_cmd_INIT_reg <= 1'b1;
472                         MAC_pckt_num_INIT_reg <= 5'd0;
473                         init_state <= init_state + 1;
474                      end
475                   end
476                   SI_MASTER_FINISH_INIT_5: begin
477                      MAC_cmd_INIT_reg <= 1'b0;
478                      init_state <= init_state + 1;
479                   end
480                   SI_MASTER_FINISH_INIT_6: begin
481                      if (MAC_sts == 0) begin // If MAC done sending
482                         init_state <= SI_MASTER_FINISH_INIT;
483                         cntr <= P_ACK_TIMEOUT;
484                      end
485                   end
486
```

```
487                     ////////////////////////////////////////////////////////////////////////////
488                     // SLAVE INITIALIZATION PROTOCOL
489                     ////////////////////////////////////////////////////////////////////////////
490
491                     SI_SLAVE_INIT: begin
492                        if (req == REQ_INIT_ACK_ACK) begin
493                           req_rd <= 1;
494                           init_state <= SI_SLAVE_FINISH_INIT;
495                           ignore_reg <= 5'b10110;
496                        end
497                        else if (req != REQ_NONE) begin  // ignore other request types
498                           req_rd <= 1;
499                        end
500                        else if (cntr) begin
501                           req_rd <= 0;
502                           cntr <= cntr - 1;
503                        end
504                        else begin // (!cntr), start sending INIT_ACK
505                           req_rd <= 0;
506                           if (CRC_sts == 0) begin
507                              CRC_D_TX_ready_INIT_reg <= 1'b1;
508                              CRC_D_TX_INIT_reg <= REQ_INIT_ACK;
509                              CRC_pckt_num_INIT_reg <= 5'd0;
510                              init_state <= init_state + 1;
511                           end
512                        end
513                     end
514                     SI_SLAVE_INIT_1: begin
515                        CRC_D_TX_INIT_reg <= self_phn_num;
516                        init_state <= init_state + 1;
517                     end
518                     SI_SLAVE_INIT_2: begin
519                        CRC_D_TX_ready_INIT_reg <= 0;
520                        if (CRC_sts == 0) init_state <= init_state + 1;
521                     end
522                     SI_SLAVE_INIT_3: begin
523                        if (MAC_sts == 0) begin  // If MAC idle, send packet to it
524                           MAC_cmd_INIT_reg <= 1'b1;
525                           MAC_pckt_num_INIT_reg <= 5'd0;
526                           init_state <= init_state + 1;
527                        end
528                     end
529                     SI_SLAVE_INIT_4: begin
530                        MAC_cmd_INIT_reg <= 1'b0;
531                        init_state <= init_state + 1;
532                     end
533                     SI_SLAVE_INIT_5: begin
534                        if (MAC_sts == 0) begin // If MAC done sending
535                           init_state <= SI_SLAVE_INIT;
536                           cntr <= P_ACK_TIMEOUT;
537                        end
538                     end
539
540                     ////////////////////////////////////////////////////////////////////////////
541                     // SLAVE INITIALIZATION FINISH PROTOCOL
542                     ////////////////////////////////////////////////////////////////////////////
543
544                     SI_SLAVE_FINISH_INIT: begin
545                        if (req == REQ_INIT) begin
546                           req_rd <= 1;
547                           init_state <= SI_SLAVE_INIT;
```

```
548                         ignore_reg <= 5'b11010;  // Ignore ACK requests
549                         cntr <= 0;
550                      end
551                      else if (req == REQ_INIT_FINISH) begin // Send FINISH_ACK
552                         req_rd <= 1;
553                         init_state <= init_state + 1;
554                      end
555                      else if (req != REQ_NONE) begin
556                         req_rd <= 1;
557                      end
558                      else begin
559                         req_rd <= 0;
560                      end
561                   end
562                   SI_SLAVE_FINISH_INIT_1: begin
563                      req_rd <= 0;
564                      if (CRC_sts == 0) begin
565                         CRC_D_TX_ready_INIT_reg <= 1'b1;
566                         CRC_D_TX_INIT_reg <= REQ_INIT_FINISH_ACK;
567                         CRC_pckt_num_INIT_reg <= 5'd0;
568                         init_state <= init_state + 1;
569                      end
570                   end
571                   SI_SLAVE_FINISH_INIT_2: begin
572                      CRC_D_TX_ready_INIT_reg <= 1'b0;
573                      init_state <= init_state + 1;
574                   end
575                   SI_SLAVE_FINISH_INIT_3: begin
576                      if (CRC_sts == 0) init_state <= init_state + 1;
577                   end
578                   SI_SLAVE_FINISH_INIT_4: begin
579                      if (MAC_sts == 0) begin  // If MAC idle, send packet to it
580                         MAC_cmd_INIT_reg <= 1'b1;
581                         MAC_pckt_num_INIT_reg <= 5'd0;
582                         init_state <= init_state + 1;
583                      end
584                   end
585                   SI_SLAVE_FINISH_INIT_5: begin
586                      MAC_cmd_INIT_reg <= 1'b0;
587                      init_state <= init_state + 1;
588                   end
589                   SI_SLAVE_FINISH_INIT_6: begin
590                      if (MAC_sts == 0) begin // If MAC done sending
591                         init_state <= SI_INIT_SUCCESS;
592                         ignore_reg <= 5'b10111;
593                      end
594                   end
595
596                   ///////////////////////////////////////////////////////////////////////////
597                   // INITIALIZATION END STATES
598
          ///////////////////////////////////////////////////////////////////////////
599
600                   SI_INIT_SUCCESS: begin
601                      if (req == REQ_INIT_FINISH) begin  // resend INIT_FINISH_ACK
602                         req_rd <= 1;
603                         init_state <= SI_SLAVE_FINISH_INIT_1;
604                      end
605                      else if (req != REQ_NONE) begin
606                         req_rd <= 1;
607                      end
```

```
608                   else begin
609                      req_rd <= 0;
610                   end
611                   link_up <= 1;
612                end
613                SI_INIT_FAIL: begin
614                   link_up <= 0;
615                end
616             endcase
617          end
618       end
619
620       // Instantiate ARQ manager for post-initialization
621       ARQ_Manager ARQ_MANAGER_1(
622          .clk_40mhz(clk_40mhz),
623          .reset(reset),
624          // General signals
625          .start(link_up),
626          .self_phn_num(self_phn_num),
627          // Network I/O
628          .NW_cmd(cmd),
629          .NW_sts(sts),
630          .NW_D_TX_addr(D_TX_addr),
631          .NW_D_TX(D_TX),
632          .NW_D_TX_ready(D_TX_ready),
633          .NW_D_RX_addr(D_RX_addr),
634          .NW_D_RX(D_RX),
635          .NW_D_RX_ready(D_RX_ready),
636          // CRC Encoder I/O
637          .CRC_sts(CRC_sts),
638          .D_TX(CRC_D_TX_ARQ),
639          .D_TX_ready(CRC_D_TX_ready_ARQ),
640          .pckt_num(CRC_pckt_num_ARQ),
641          // MAC I/O
642          .MAC_cmd(MAC_cmd_ARQ),
643          .MAC_sts(MAC_sts),
644          .MAC_pckt_num(MAC_pckt_num_ARQ),
645          // CRC Decoder I/O
646          .D_RX(CRC_D_RX),
647          .D_RX_ready(CRC_D_RX_ready)
648          ,// DEBUG
649          .b_state_DEBUG(b_state_DEBUG),
650          .f_state_DEBUG(f_state_DEBUG),
651          .s_state_DEBUG(s_state_DEBUG),
652          .c_state_DEBUG(c_state_DEBUG),
653          .b_f_lock_DEBUG(b_f_lock_DEBUG),
654          .b_b_lock_DEBUG(b_b_lock_DEBUG),
655          .b_c_lock_DEBUG(b_c_lock_DEBUG),
656          .f1w_s_lock_DEBUG(f1w_s_lock_DEBUG),
657          .f1w_b_lock_DEBUG(f1w_b_lock_DEBUG),
658          .f1w_c_lock_DEBUG(f1w_c_lock_DEBUG),
659          .f1r_f_lock_DEBUG(f1r_f_lock_DEBUG),
660          .f1r_b_lock_DEBUG(f1r_b_lock_DEBUG),
661          .f2w_f_lock_DEBUG(f2w_f_lock_DEBUG),
662          .f2w_b_lock_DEBUG(f2w_b_lock_DEBUG),
663          .f2w_c_lock_DEBUG (f2w_c_lock_DEBUG),
664          .p_DEBUG(p_DEBUG),
665          .dina_DEBUG(dina_DEBUG),
666          .wea_DEBUG(wea_DEBUG),
667          .addra_DEBUG(addra_DEBUG),
668          .douta_DEBUG(douta_DEBUG)
```

```
669             );
670
671          // DEBUG
672          assign init_state_DEBUG = init_state;
673          assign CRC_E_we_DEBUG = CRC_E_we;
674          assign CRC_E_addr_DEBUG = CRC_E_addr;
675          assign CRC_E_data_DEBUG = CRC_E_data;
676          assign MAC_data_DEBUG = MAC_data;
677          assign MAC_addr_DEBUG = MAC_addr;
678          assign MAC_pckt_num_DEBUG = MAC_pckt_num;
679          assign MAC_sts_DEBUG = MAC_sts;
680          assign MAC_cmd_DEBUG = MAC_cmd;
681          assign CRC_sts_DEBUG = CRC_sts;
682          assign CRC_D_TX_DEBUG = CRC_D_TX;
683          assign CRC_D_TX_ready_DEBUG = CRC_D_TX_ready;
684          assign CRC_pckt_num_DEBUG = CRC_pckt_num;
685          assign CRC_D_RX_ready_DEBUG = CRC_D_RX_ready;
686          assign CRC_D_RX_DEBUG = CRC_D_RX;
687
688       endmodule
689
690       // Media Access Control
691       //  o Truncated binary exponential backoff sending
692       //  o Reads from BRAM using packet number
693       module MAC #(
694          parameter P_CW_MIN = 4,
695          parameter P_CW_MAX = 19,
696          parameter P_LOG_CW_MAX = 5,
697          parameter P_TIMEOUT_PERIOD = 6'd63
698          )(
699          input clk_40mhz,
700          input reset,
701          // Main I/O
702          input cmd,
703          output sts,
704          input [4:0] pckt_num,
705          // PHY I/O
706          output [7:0] PHY_TX,
707          output PHY_TX_ready,
708          input PHY_CD,
709          input PHY_TX_success,
710          input PHY_IB,
711          // Packet Storage I/O
712          input [7:0] data,
713          output [10:0] addr,
714          // PRNG I/O
715          input rand_num_serial
716          ,// DEBUG
717          output [5:0] pckt_size_DEBUG,
718          output [2:0] MAC_state_DEBUG
719          );
720
721          // Assign TCV TX data
722          assign PHY_TX = data;
723
724          // Instantiate TCV TX ready signal register
725          reg PHY_TX_ready_reg;
726          assign PHY_TX_ready = PHY_TX_ready_reg;
727
728          // Instantiate status register
729          reg sts_reg;
```

```
730            assign sts = sts_reg;
731
732            // Declare command parameters
733            parameter CMD_IDLE = 0;
734            parameter CMD_SEND = 1;
735
736            // Declare status parameters
737            parameter STS_CMD_RDY = 0;
738            parameter STS_BUSY    = 1;
739
740            // Instantiate state
741            reg [2:0] state;
742
743            // Declare state parameters
744            parameter S_IDLE      = 3'h0;
745            parameter S_TX_WAIT   = 3'h1;
746            parameter S_TX_WAIT_1 = 3'h2;
747            parameter S_TX_WAIT_2 = 3'h3;
748            parameter S_TX        = 3'h4;
749            parameter S_TX_1      = 3'h5;
750            parameter S_TO        = 3'h6;
751
752            // Instantiate counter state
753            reg [1:0] cnt_state;
754
755            // Declare counter state parameters
756            parameter SC_IDLE     = 2'h0;
757            parameter SC_INIT_CNT = 2'h1;
758            parameter SC_CNT      = 2'h2;
759
760            // Set & Run CW Counter as needed
761            reg start_cnt;
762            reg [(P_LOG_CW_MAX-1):0] CW, CW_l;
763            reg [(P_CW_MAX-1):0] cnt;
764            always @(posedge clk_40mhz) begin
765               if (reset) begin
766                  cnt <= 0;
767                  CW_l <= 0;
768                  cnt_state <= SC_IDLE;
769               end
770               else begin
771                  case (cnt_state)
772                     SC_IDLE: begin
773                        if (start_cnt) begin
774                           cnt_state <= cnt_state + 1;
775                           CW_l <= CW;
776                           cnt <= 0;
777                        end
778                     end
779                     SC_INIT_CNT: begin
780                        if (CW_l) begin
781                           cnt <= {cnt[(P_CW_MAX-2):0], rand_num_serial};
782                           CW_l <= CW_l - 1;
783                        end
784                        else begin
785                           cnt_state <= cnt_state + 1;
786                        end
787                     end
788                     SC_CNT: begin
789                        if (cnt) begin
790                           cnt <= cnt - 1;
```

```
791                      end
792                      else begin
793                         cnt_state <= SC_IDLE;
794                      end
795                   end
796                endcase
797             end
798          end
799
800          // Instantiate packet storage output registers
801          reg [10:0] addr_reg;
802          assign addr = addr_reg;
803
804          // Instantiate next byte wire
805          wire [5:0] next_byte;
806          assign next_byte = addr_reg[5:0] + 1;
807
808          // Manage main state transitions
809          reg [5:0] cntr;
810          reg [5:0] pckt_size;
811          always @(posedge clk_40mhz) begin
812             if (reset) begin
813                state <= S_IDLE;
814                CW <= P_CW_MIN;
815                sts_reg <= STS_BUSY;
816                addr_reg <= 0;
817                PHY_TX_ready_reg <= 1'b0;
818             end
819             else begin
820                case (state)
821                   S_IDLE: begin
822                      PHY_TX_ready_reg <= 1'b0;
823                      if (cmd == CMD_SEND) begin
824                         sts_reg <= STS_BUSY;
825                         addr_reg <= {pckt_num, 6'd63};
826                         state <= S_TX_WAIT;
827                         start_cnt <= 1'b1;
828                      end
829                      else begin
830                         sts_reg <= STS_CMD_RDY;
831                      end
832                   end
833                   S_TX_WAIT: begin  // Wait two cycles for counter to start
834                      addr_reg <= {addr_reg[10:6], next_byte};
835                      start_cnt <= 1'b0;
836                      state <= state + 1;
837                   end
838                   S_TX_WAIT_1: begin // Store packet size
839                      pckt_size <= data[5:0];
840                      state <= state + 1;
841                   end
842                   S_TX_WAIT_2: begin  // Wait until counter finished to start transmission
843                      if ((cnt_state == SC_IDLE) && (PHY_IB)) begin
844                         PHY_TX_ready_reg <= 1'b1;
845                         addr_reg <= {addr_reg[10:6], next_byte};
846                         pckt_size <= pckt_size - 1;
847                         state <= state + 1;
848                      end
849                   end
850                   S_TX: begin
851                      if (PHY_CD) begin  // if collision detected, go back to wait with higher
```

```
852                             PHY_TX_ready_reg <= 0;
853                             CW <= (CW == P_CW_MAX) ? P_CW_MAX: (CW+1);
854                             addr_reg <= {addr_reg[10:6], 6'd63};
855                             state <= S_TX_WAIT;
856                             start_cnt <= 1'b1;
857                          end
858                          else if (pckt_size) begin
859                             pckt_size <= pckt_size - 1;
860                             addr_reg <= {addr_reg[10:6], next_byte};
861                          end
862                          else begin // if nothing left to transfer
863                             PHY_TX_ready_reg <= 1'b0;
864                             state <= state + 1;
865                          end
866                       end
867                       S_TX_1: begin
868                          if (PHY_CD) begin  // If collision detected, go back to wait with higher C
869                             CW <= (CW == P_CW_MAX) ? P_CW_MAX: (CW+1);
870                             addr_reg <= {addr_reg[10:6], 6'd63};
871                             state <= S_TX_WAIT;
872                             start_cnt <= 1'b1;
873                          end
874                          else if (PHY_TX_success) begin // If transmission successful, decrease CW
875                             CW <= (CW == P_CW_MIN) ? P_CW_MIN: (CW-1);
876                             cntr <= P_TIMEOUT_PERIOD;
877                             state <= state + 1;
878                          end
879                          else if (PHY_IB) begin
880                             CW <= (CW == P_CW_MAX) ? P_CW_MAX: (CW+1);
881                             addr_reg <= {addr_reg[10:6], 6'd63};
882                             state <= S_TX_WAIT;
883                             start_cnt <= 1'b1;
884                          end
885                       end
886                       S_TO: begin
887                          if (cntr) begin
888                             cntr <= cntr - 1;
889                          end
890                          else begin
891                             state <= S_IDLE;
892                             sts_reg <= STS_CMD_RDY;
893                          end
894                       end
895                    endcase
896                 end
897              end
898
899              // DEBUG
900              assign pckt_size_DEBUG = pckt_size;
901              assign MAC_state_DEBUG = state;
902           endmodule
903
904           // CRC16-ANSI generator
905           // o 8-bit data in w/ ready
906           // o writes payload and packet length to BRAM
907
908           module CRC_Enc(
909              input clk_40mhz,
910              input reset,
911              // Data I/O
912              output sts,
```

```
913         input [7:0] D_TX,
914         input D_TX_ready,
915         input [4:0] pckt_num,
916         // BRAM I/O
917         output we,
918         output [10:0] addr,
919         output [7:0] data
920         );
921
922         // Instantiate status register
923         reg sts_reg;
924         assign sts = sts_reg;
925
926         // Instantiate output registers
927         reg we_reg;
928         assign we = we_reg;
929         reg [7:0] data_reg;
930         assign data = data_reg;
931         reg [10:0] addr_reg;
932         assign addr = addr_reg;
933
934         // Declare command parameters
935         parameter CMD_IDLE = 0;
936         parameter CMD_WRITE = 1;
937
938         // Declare status parameters
939         parameter STS_CMD_RDY = 0;
940         parameter STS_BUSY    = 1;
941
942         // Instantiate state
943         reg [1:0] state;
944
945         // Declare state parameters
946         parameter S_IDLE = 2'h0;
947         parameter S_WR   = 2'h1;
948         parameter S_WR_1 = 2'h2;
949         parameter S_WR_2 = 2'h3;
950
951         // Instantiate CRC register
952         reg [15:0] CRC_reg;
953
954         // Assign wire for next CRC
955         wire [15:0] CRC_new;
956         assign CRC_new = {(^data_reg) ^ (^CRC_reg[15:7]),
957                          CRC_reg[6],
958                          CRC_reg[5],
959                          CRC_reg[4],
960                          CRC_reg[3],
961                          CRC_reg[2],
962                          CRC_reg[1] ^ CRC_reg[15] ^ data_reg[7],
963                          CRC_reg[0] ^ CRC_reg[15] ^ data_reg[7] ^ CRC_reg[14] ^ data_reg[6],
964                          CRC_reg[14] ^ data_reg[6] ^ CRC_reg[13] ^ data_reg[5],
965                          CRC_reg[13] ^ data_reg[5] ^ CRC_reg[12] ^ data_reg[4],
966                          CRC_reg[12] ^ data_reg[4] ^ CRC_reg[11] ^ data_reg[3],
967                          CRC_reg[11] ^ data_reg[3] ^ CRC_reg[10] ^ data_reg[2],
968                          CRC_reg[10] ^ data_reg[2] ^ CRC_reg[9] ^ data_reg[1],
969                          CRC_reg[9] ^ data_reg[1] ^ CRC_reg[8] ^ data_reg[0],
970                          (^data_reg[7:1]) ^ (^CRC_reg[15:9]),
971                          (^data_reg) ^ (^CRC_reg[15:8])
972                          };
973
```

```
974           // Manage state transitions
975           wire [5:0] next_byte;
976           assign next_byte = addr_reg[5:0] + 1;
977           always @(posedge clk_40mhz) begin
978              if (reset) begin
979                 state <= S_IDLE;
980                 CRC_reg <= 0;
981                 we_reg <= 1'b0;
982                 addr_reg <= 11'd0;
983                 data_reg <= 8'd0;
984                 sts_reg <= STS_BUSY;
985              end
986              else begin
987                 case (state)
988                    S_IDLE: begin
989                       if (D_TX_ready) begin
990                          state <= state + 1;
991                          addr_reg <= {pckt_num, 6'd0};
992                          data_reg <= D_TX;
993                          we_reg <= 1'b1;
994                          sts_reg <= STS_BUSY;
995                       end
996                       else begin
997                          CRC_reg <= 0;
998                          we_reg <= 0;
999                          sts_reg <= STS_CMD_RDY;
1000                      end
1001                   end
1002                   S_WR: begin  // write payload
1003                      addr_reg <= {addr_reg[10:6], next_byte};
1004                      if (D_TX_ready) begin
1005                         data_reg <= D_TX;
1006                         CRC_reg <= CRC_new;
1007                      end
1008                      else begin // write MSByte of CRC
1009                         data_reg <= CRC_new[15:8];
1010                         CRC_reg <= CRC_new;
1011                         state <= state + 1;
1012                      end
1013                   end
1014                   S_WR_1: begin // write LSByte of CRC
1015                      addr_reg <= {addr_reg[10:6], next_byte};
1016                      data_reg <= CRC_reg[7:0];
1017                      CRC_reg <= 0;
1018                      state <= state + 1;
1019                   end
1020                   S_WR_2: begin  // write packet count
1021                      addr_reg <= {addr_reg[10:6], 6'd63};
1022                      data_reg <= {2'b00, next_byte};
1023                      state <= S_IDLE;
1024                      sts_reg <= STS_CMD_RDY;
1025                   end
1026                endcase
1027             end
1028          end
1029       endmodule
1030
1031       // CRC16-ANSI checker
1032       // o Interfaces with PHY RCV signals
1033       // Two sets of output ports, one for I/O, one for
1034       module CRC_Dec(
```

```
1035            input clk_40mhz,
1036            input reset,
1037            // PHY I/O
1038            input [7:0] PHY_RX,            // PHY Data RX
1039            input PHY_RX_ready,               // PHY Data RX Ready
1040            // Data I/O
1041            output [7:0] D_RX,              // incoming data port
1042            output D_RX_ready                 // incoming ready
1043            ,// DEBUG
1044            output [4:0] rd_pointer_DEBUG,
1045            output [4:0] wr_pointer_DEBUG,
1046            output [2:0] CRC_state_DEBUG
1047            );
1048
1049            // Instantiate Incoming BRAM pointers
1050            reg [4:0] rd_pointer, wr_pointer;
1051
1052            // Instantiate Block RAM for incoming packets
1053            wire [10:0] rd_addr, wr_addr;
1054            wire [7:0] wr_data, rd_data;
1055            wire we;
1056            DLC_BRAM_RX DLC_BRAM_RX_1(
1057               .clka(clk_40mhz),
1058               .clkb(clk_40mhz),
1059               .addra(wr_addr),
1060               .dina(wr_data),
1061               .wea(we),
1062               .addrb(rd_addr),
1063               .doutb(rd_data)
1064               );
1065
1066            // Assign data outputs
1067            reg D_RX_ready_reg;
1068            assign D_RX_ready = D_RX_ready_reg;
1069            assign D_RX = rd_data;
1070
1071            ///////////////////////////////////////////////////////////////////////////////
1072            // Incoming BRAM Write Logic
1073            ///////////////////////////////////////////////////////////////////////////////
1074
1075            // Latch RX_ready
1076            reg PHY_RX_ready_l;
1077            always @(posedge clk_40mhz) PHY_RX_ready_l <= (reset) ? 1'b0: PHY_RX_ready;
1078
1079            // Track fall of latched RX_ready
1080            reg PHY_RX_ready_ll;
1081            wire PHY_RX_ready_l_fall;
1082            always @(posedge clk_40mhz) PHY_RX_ready_ll <= (reset) ? 1'b0: PHY_RX_ready_l;
1083            assign PHY_RX_ready_l_fall = ~PHY_RX_ready_l & PHY_RX_ready_ll;
1084
1085            // Instantiate Block RAM input registers
1086            reg we_override;
1087            reg [5:0] wr_addr_reg;
1088            wire [5:0] wr_addr_low;
1089            assign we = (PHY_RX_ready | PHY_RX_ready_l) & we_override;
1090            assign wr_addr_low = (PHY_RX_ready) ? wr_addr_reg: 6'd63;
1091            assign wr_addr = {wr_pointer, wr_addr_low};
1092            assign wr_data = (PHY_RX_ready) ? PHY_RX: {2'b00, wr_addr_reg};
1093
1094            // Set addr_reg
1095            always @(posedge clk_40mhz) begin
```

```
1096            if (reset)
1097               wr_addr_reg <= 6'd0;
1098            else if (PHY_RX_ready)
1099               wr_addr_reg <= wr_addr_reg + 1;
1100            else
1101               wr_addr_reg <= 6'd0;
1102         end
1103
1104         // Set we override, write pointer
1105         always @(posedge clk_40mhz) begin
1106            if (reset) begin
1107               we_override <= 1'b0;
1108               wr_pointer <= 5'd1;
1109            end
1110            else if (wr_pointer != rd_pointer) begin
1111               we_override <= 1'b1;
1112               if (PHY_RX_ready_l_fall) wr_pointer <= wr_pointer + 1;
1113            end
1114            else begin
1115               we_override <= 1'b0;
1116            end
1117         end
1118
1119         ////////////////////////////////////////////////////////////////////////////////
1120         // Incoming BRAM Read Logic
1121         ////////////////////////////////////////////////////////////////////////////////
1122
1123         // Instantiate BRAM input registers
1124         reg [5:0] rd_addr_reg;
1125         assign rd_addr = {rd_pointer, rd_addr_reg};
1126
1127         // Instantiate state
1128         reg [2:0] state;
1129
1130         // Declare state parameters
1131         parameter S_IDLE          = 3'h0;
1132         parameter S_RD_BYTE_CNT   = 3'h1;
1133         parameter S_RD_BYTE_CNT_1 = 3'h2;
1134         parameter S_CRC           = 3'h3;
1135         parameter S_CRC_1         = 3'h4;
1136         parameter S_RD            = 3'h5;
1137         parameter S_RD_1          = 3'h6;
1138
1139         // Instantiate/Assign CRC regs/wires
1140         reg [7:0] data_reg;
1141         reg [15:0] CRC_reg;
1142         wire [15:0] CRC_new;
1143         assign CRC_new = {(^data_reg) ^ (^CRC_reg[15:7]),
1144                           CRC_reg[6],
1145                           CRC_reg[5],
1146                           CRC_reg[4],
1147                           CRC_reg[3],
1148                           CRC_reg[2],
1149                           CRC_reg[1] ^ CRC_reg[15] ^ data_reg[7],
1150                           CRC_reg[0] ^ CRC_reg[15] ^ data_reg[7] ^ CRC_reg[14] ^ data_reg[6],
1151                           CRC_reg[14] ^ data_reg[6] ^ CRC_reg[13] ^ data_reg[5],
1152                           CRC_reg[13] ^ data_reg[5] ^ CRC_reg[12] ^ data_reg[4],
1153                           CRC_reg[12] ^ data_reg[4] ^ CRC_reg[11] ^ data_reg[3],
1154                           CRC_reg[11] ^ data_reg[3] ^ CRC_reg[10] ^ data_reg[2],
1155                           CRC_reg[10] ^ data_reg[2] ^ CRC_reg[9] ^ data_reg[1],
1156                           CRC_reg[9] ^ data_reg[1] ^ CRC_reg[8] ^ data_reg[0],
```

```
1157                            (^data_reg[7:1]) ^ (^CRC_reg[15:9]),
1158                            (^data_reg) ^ (^CRC_reg[15:8])
1159                            };
1160
1161            // Manage state transitions
1162            reg [5:0] byte_cnt, out_byte_cnt;
1163            wire [4:0] wr_pointer_minus_one;
1164            assign wr_pointer_minus_one = wr_pointer - 1;
1165            always @(posedge clk_40mhz) begin
1166               if (reset) begin
1167                  rd_pointer <= 5'd0;
1168                  rd_addr_reg <= 6'd0;
1169                  state <= S_IDLE;
1170                  CRC_reg <= 0;
1171                  data_reg <= 0;
1172                  state <= S_IDLE;
1173                  D_RX_ready_reg <= 1'b0;
1174               end
1175               else begin
1176                  case (state)
1177                     S_IDLE: begin
1178                        D_RX_ready_reg <= 1'b0;
1179                        if (rd_pointer != wr_pointer_minus_one) begin
1180                           rd_pointer <= rd_pointer + 1;
1181                           state <= state + 1;
1182                           rd_addr_reg <= 6'd63;
1183                           state <= state + 1;
1184                        end
1185                     end
1186                     S_RD_BYTE_CNT: begin
1187                        rd_addr_reg <= 6'd0;
1188                        state <= state + 1;
1189                     end
1190                     S_RD_BYTE_CNT_1: begin
1191                        byte_cnt <= rd_data;
1192                        out_byte_cnt <= rd_data - 2;
1193                        rd_addr_reg <= 6'd1;
1194                        state <= state + 1;
1195                     end
1196                     S_CRC: begin
1197                        byte_cnt <= byte_cnt - 1;
1198                        data_reg <= rd_data;
1199                        CRC_reg <= 0;
1200                        rd_addr_reg <= 6'd2;
1201                        state <= state + 1;
1202                     end
1203                     S_CRC_1: begin
1204                        if (byte_cnt) begin
1205                           rd_addr_reg <= rd_addr_reg + 1;
1206                           byte_cnt <= byte_cnt - 1;
1207                           CRC_reg <= CRC_new;
1208                           data_reg <= rd_data;
1209                        end
1210                        else begin
1211                           rd_addr_reg <= 0;
1212                           if (CRC_new) begin // CRC failed
1213                              state <= S_IDLE;
1214                           end
1215                           else begin // CRC success, readout again for sending
1216                              state <= state + 1;
1217                           end
```

```
1218                    end
1219                 end
1220                 S_RD: begin
1221                    out_byte_cnt <= out_byte_cnt - 1;
1222                    rd_addr_reg <= 1;
1223                    D_RX_ready_reg <= 1;
1224                    state <= state + 1;
1225                 end
1226                 S_RD_1: begin
1227                    if (out_byte_cnt) begin
1228                       out_byte_cnt <= out_byte_cnt - 1;
1229                       rd_addr_reg <= rd_addr_reg + 1;
1230                    end
1231                    else begin
1232                       D_RX_ready_reg <= 0;
1233                       state <= S_IDLE;
1234                       rd_addr_reg <= 0;
1235                    end
1236                 end
1237              endcase
1238           end
1239        end
1240
1241        assign rd_pointer_DEBUG = rd_pointer;
1242        assign wr_pointer_DEBUG = wr_pointer;
1243        assign CRC_state_DEBUG = state;
1244     endmodule
1245
1246     // Module for pushing requests to a queue
1247     module Request_Buffer(
1248        input clk_40mhz,
1249        input reset,
1250        // DLC Control I/O
1251        output [7:0] req,         // request type
1252        output [7:0] req_param,   // request paramter
1253        input req_rd,              // clear current request, load next
1254        input [7:0] self_phn_num,
1255        input [4:0] ignore,
1256        // CRC Decoder I/O
1257        input [7:0] D_RX,         // incoming data port
1258        input D_RX_ready            // incoming ready
1259        );
1260
1261        // Declare request parameters
1262        // Define request parameters
1263        parameter REQ_NONE            = 8'h00;
1264        parameter REQ_INIT            = 8'h01;
1265        parameter REQ_INIT_ACK        = 8'h02;
1266        parameter REQ_INIT_ACK_ACK    = 8'h03;
1267        parameter REQ_INIT_FINISH     = 8'h04;
1268        parameter REQ_INIT_FINISH_ACK = 8'h05;
1269
1270        // Instantiate FIFO
1271        wire wr_en, rd_en, empty, full;
1272        wire [15:0] din, dout;
1273        DLC_Request_FIFO DLC_REQUEST_FIFO_1(
1274           .clk(clk_40mhz),
1275           .rst(reset),
1276           .din(din),
1277           .wr_en(wr_en),
1278           .dout(dout),
```

```
1279            .rd_en(rd_en),
1280            .empty(empty),
1281            .full(full)
1282            );
1283
1284         // Instantiate FIFO inputs
1285         reg [15:0] din_reg;
1286         reg wr_en_reg;
1287         assign din = din_reg;
1288         assign rd_en = req_rd;
1289         assign wr_en = wr_en_reg;
1290
1291         // Assign outputs
1292         assign req = (empty) ? (REQ_NONE): dout[7:0];
1293         assign req_param = dout[15:8];
1294
1295         // Instantiate state
1296         reg [2:0] state;
1297
1298         // Define state parameters
1299         parameter S_IDLE             = 3'h0;
1300         parameter S_INIT             = 3'h1;
1301         parameter S_INIT_ACK         = 3'h2;
1302         parameter S_INIT_ACK_ACK     = 3'h3;
1303         parameter S_INIT_FINISH      = 3'h4;
1304         parameter S_INIT_FINISH_ACK = 3'h5;
1305
1306         // Detect ready rise
1307         reg D_RX_ready_l;
1308         wire D_RX_ready_rise;
1309         always @(posedge clk_40mhz) D_RX_ready_l = (reset) ? 1'b0: D_RX_ready;
1310         assign D_RX_ready_rise = D_RX_ready & ~D_RX_ready_l;
1311
1312         // Manage state transitions
1313         always @(posedge clk_40mhz) begin
1314            if (reset) begin
1315               wr_en_reg <= 0;
1316               din_reg <= 0;
1317               state <= S_IDLE;
1318            end
1319            else begin
1320               case (state)
1321                  S_IDLE: begin
1322                     wr_en_reg <= 0;
1323                     din_reg <= {8'h00, D_RX};
1324                     if (D_RX_ready_rise) begin
1325                        case (D_RX)
1326                           REQ_INIT: begin
1327                              if (~ignore[0]) state <= S_INIT;
1328                           end
1329                           REQ_INIT_ACK: begin
1330                              if (~ignore[1]) state <= S_INIT_ACK;
1331                           end
1332                           REQ_INIT_ACK_ACK: begin
1333                              if (~ignore[2]) state <= S_INIT_ACK_ACK;
1334                           end
1335                           REQ_INIT_FINISH: begin
1336                              if (~ignore[3]) state <= S_INIT_FINISH;
1337                           end
1338                           REQ_INIT_FINISH_ACK: begin
1339                              if (~ignore[4]) state <= S_INIT_FINISH_ACK;
```

```
1340                    end
1341                 endcase
1342              end
1343           end
1344           S_INIT: begin
1345              wr_en_reg <= 1;
1346              din_reg <= {D_RX, din_reg[7:0]};
1347              state <= S_IDLE;
1348           end
1349           S_INIT_ACK: begin
1350              wr_en_reg <= 1;
1351              din_reg <= {D_RX, din_reg[7:0]};
1352              state <= S_IDLE;
1353           end
1354           S_INIT_ACK_ACK: begin
1355              if (D_RX == self_phn_num) begin
1356                 wr_en_reg <= 1;
1357                 din_reg <= {D_RX, din_reg[7:0]};
1358                 state <= S_IDLE;
1359              end
1360           end
1361           S_INIT_FINISH: begin
1362              wr_en_reg <= 1;
1363              din_reg <= {8'h00, REQ_INIT_FINISH};
1364              state <= S_IDLE;
1365           end
1366           S_INIT_FINISH_ACK: begin
1367              wr_en_reg <= 1;
1368              din_reg <= {8'h00, REQ_INIT_FINISH_ACK};
1369              state <= S_IDLE;
1370           end
1371        endcase
1372     end
1373  end
1374 endmodule
1375
1376 // Module for managing Parallel Stop-and-Wait ARQ
1377
1378 module ARQ_Manager #(
1379    parameter P_DELETE_TIMEOUT = 10'd1023,       // Approximately 2048 cycles per count
1380    parameter P_LOG_SIZE_ARQ_TIMEOUT = 6 // Approximately 2048 cycles per count
1381    )(
1382    input clk_40mhz,
1383    input reset,
1384    // General signals
1385    input start,
1386    input [7:0] self_phn_num,
1387    // Network I/O
1388    input [1:0] NW_cmd,
1389    output [1:0] NW_sts,
1390    input [7:0] NW_D_TX_addr,
1391    input [7:0] NW_D_TX,
1392    input NW_D_TX_ready,
1393    output [7:0] NW_D_RX_addr,
1394    output [7:0] NW_D_RX,
1395    output NW_D_RX_ready,
1396    // CRC Encoder I/O
1397    input CRC_sts,
1398    output [7:0] D_TX,
1399    output D_TX_ready,
1400    output [4:0] pckt_num,
```

```
1401            // MAC I/O
1402            output MAC_cmd,
1403            input MAC_sts,
1404            output [4:0] MAC_pckt_num,
1405            // CRC Decoder I/O
1406            input [7:0] D_RX,          // incoming data port
1407            input D_RX_ready             // incoming ready
1408            ,// DEBUG
1409            output [4:0] b_state_DEBUG,
1410            output [4:0] f_state_DEBUG,
1411            output [2:0] s_state_DEBUG,
1412            output [3:0] c_state_DEBUG,
1413            output b_f_lock_DEBUG,
1414            output b_b_lock_DEBUG,
1415            output b_c_lock_DEBUG,
1416            output f1w_s_lock_DEBUG,
1417            output f1w_b_lock_DEBUG,
1418            output f1w_c_lock_DEBUG,
1419            output f1r_f_lock_DEBUG,
1420            output f1r_b_lock_DEBUG,
1421            output f2w_f_lock_DEBUG,
1422            output f2w_b_lock_DEBUG,
1423            output f2w_c_lock_DEBUG ,
1424            output p_DEBUG,
1425            output [15:0] dina_DEBUG,
1426            output wea_DEBUG,
1427            output [7:0] addra_DEBUG,
1428            output [15:0] douta_DEBUG
1429            );
1430
1431            // Define command parameters
1432            parameter CMD_IDLE        = 2'd0;
1433            parameter CMD_SET_PHN_NUM = 2'd1;
1434            parameter CMD_INIT        = 2'd2;
1435            parameter CMD_TX          = 2'd3;
1436
1437            // Define status parameters
1438            parameter STS_IDLE      = 2'd0;
1439            parameter STS_BUSY      = 2'd1;
1440            parameter STS_TX_ACCEPT = 2'd2;
1441            parameter STS_TX_REJECT = 2'd3;
1442
1443            // Define request parameters
1444            parameter REQ_NONE            = 8'h00;
1445            parameter REQ_INIT            = 8'h01;
1446            parameter REQ_INIT_ACK        = 8'h02;
1447            parameter REQ_INIT_ACK_ACK    = 8'h03;
1448            parameter REQ_INIT_FINISH     = 8'h04;
1449            parameter REQ_INIT_FINISH_ACK = 8'h05;
1450            parameter REQ_DATA            = 8'h06;
1451            parameter REQ_DATA_ACK        = 8'h07;
1452
1453            // Instantiate output registers
1454            reg [1:0] NW_sts_reg;
1455            reg [7:0] NW_D_RX_reg;
1456            reg NW_D_RX_ready_reg;
1457            reg MAC_cmd_reg;
1458            reg [4:0] MAC_pckt_num_reg;
1459            reg [7:0] NW_D_RX_addr_reg;
1460
1461            // Assign outputs
```

```
1462        assign NW_sts = NW_sts_reg;
1463        assign NW_D_RX = NW_D_RX_reg;
1464        assign NW_D_RX_ready = NW_D_RX_ready_reg;
1465        assign MAC_cmd = MAC_cmd_reg;
1466        assign MAC_pckt_num = MAC_pckt_num_reg;
1467        assign NW_D_RX_addr = NW_D_RX_addr_reg;
1468
1469        // Instantiate FIFO for open spots in outgoing packet queue (latency is one)
1470        wire [4:0] f1_din, f1_dout;
1471        wire f1_re, f1_we;
1472        wire [4:0] f1_dc_low;
1473        wire [5:0] f1_dc;
1474        wire f1_empty, f1_full;
1475        assign f1_dc = {f1_full, f1_dc_low};
1476        DLC_ARQ_FIFO1 DLC_ARQ_FIFO1_1(
1477           .clk(clk_40mhz),
1478           .rst(reset),
1479           .din(f1_din),
1480           .rd_en(f1_re),
1481           .dout(f1_dout),
1482           .wr_en(f1_we),
1483           .empty(f1_empty),
1484           .full(f1_full),
1485           .data_count(f1_dc_low)
1486           );
1487
1488        // Instantiate r/w locks
1489        reg f1w_s_lock;
1490        reg f1w_b_lock;
1491        reg f1w_c_lock;
1492        reg f1r_f_lock;
1493        reg f1r_b_lock;
1494
1495        // Instantiate input regs
1496        reg [4:0] f1_c_din_reg;
1497        reg f1_c_we_reg;
1498
1499        reg [4:0] f1_s_din_reg;
1500        reg f1_s_we_reg;
1501
1502        reg f1_f_re_reg;
1503        reg [7:0] D_TX_f_reg;
1504        reg D_TX_ready_f_reg;
1505        reg [4:0] pckt_num_f_reg;
1506
1507        reg [4:0] f1_b_din_reg;
1508        reg f1_b_we_reg, f1_b_re_reg;
1509        reg [7:0] D_TX_b_reg;
1510        reg D_TX_ready_b_reg;
1511        reg [4:0] pckt_num_b_reg;
1512
1513        // Mux inputs according to locks
1514        assign f1_din = (f1w_b_lock) ? f1_b_din_reg: (f1w_s_lock) ? f1_s_din_reg: f1_c_din_reg
1515        assign f1_we = (f1w_b_lock) ? f1_b_we_reg: (f1w_s_lock) ? f1_s_we_reg: (f1w_c_lock) ?
         f1_c_we_reg: 0;
1516
1517        assign f1_re = (f1r_b_lock) ? f1_b_re_reg: (f1r_f_lock) ? f1_f_re_reg: 0;
1518        assign D_TX = (f1r_b_lock) ? D_TX_b_reg: (f1r_f_lock) ? D_TX_f_reg: 0;
1519        assign D_TX_ready = (f1r_b_lock) ? D_TX_ready_b_reg: (f1r_f_lock) ? D_TX_ready_f_reg:
1520        assign pckt_num = (f1r_b_lock) ? pckt_num_b_reg: (f1r_f_lock) ? pckt_num_f_reg: 0;
1521
```

```
1522        // Instantiate FIFO for outgoing send commands to MAC (latency is one)
1523        wire [5:0] f2_din, f2_dout;
1524        wire f2_re, f2_we;
1525        wire f2_empty, f2_full;
1526        DLC_ARQ_FIFO2 DLC_ARQ_FIFO2_1(
1527           .clk(clk_40mhz),
1528           .rst(reset),
1529           .din(f2_din),
1530           .rd_en(f2_re),
1531           .dout(f2_dout),
1532           .wr_en(f2_we),
1533           .empty(f2_empty),
1534           .full(f2_full)
1535           );
1536
1537        // Instantiate write locks
1538        reg f2w_f_lock;
1539        reg f2w_b_lock;
1540        reg f2w_c_lock;
1541
1542        // Instantiate input regs
1543        reg [5:0] f2_f_din_reg;
1544        reg f2_f_we_reg;
1545        reg [5:0] f2_b_din_reg;
1546        reg f2_b_we_reg;
1547        reg [5:0] f2_c_din_reg;
1548        reg f2_c_we_reg;
1549        reg f2_s_re_reg;
1550
1551        // Mux inputs according to locks
1552        assign f2_din = (f2w_b_lock) ? f2_b_din_reg: (f2w_f_lock) ? f2_f_din_reg: f2_c_din_reg
1553        assign f2_we = (f2w_b_lock) ? f2_b_we_reg: (f2w_f_lock) ? f2_f_we_reg: (f2w_c_lock) ?
         f2_c_we_reg: 0;
1554        assign f2_re = f2_s_re_reg;
1555
1556        // Instantiate BRAM for counter timeouts, and outstanding message bits (simple dual po
1557        wire [7:0] addra;
1558        wire [15:0] dina, douta;
1559        wire [15:0] doutb;
1560        wire wea;
1561        DLC_ARQ_BRAM1 DLC_ARQ_BRAM1_1(
1562           .clka(clk_40mhz),
1563           .clkb(clk_40mhz),
1564           .addra(addra),
1565           .dina(dina),
1566           .douta(douta),
1567           .wea(wea),
1568           .addrb(NW_D_TX_addr),
1569           .dinb(8'h00),
1570           .doutb(doutb),
1571           .web(1'b0)
1572           );
1573
1574        // Instantiate r/w locks
1575        reg b_f_lock;
1576        reg b_b_lock;
1577        reg b_c_lock;
1578
1579        // Instantiate input regs
1580        reg [7:0] b_f_addr_reg;
1581        reg [15:0] b_f_din_reg;
```

```
1582        reg b_f_we_reg;
1583        reg [7:0] b_b_addr_reg;
1584        reg [15:0] b_b_din_reg;
1585        reg b_b_we_reg;
1586        reg [7:0] b_c_addr_reg;
1587        reg [15:0] b_c_din_reg;
1588        reg b_c_we_reg;
1589
1590        // Mux inputs according to locks
1591        assign addra = (b_c_lock) ? b_c_addr_reg: (b_b_lock) ? b_b_addr_reg: b_f_addr_reg;
1592        assign dina = (b_c_lock) ? b_c_din_reg: (b_b_lock) ? b_b_din_reg: b_f_din_reg;
1593        assign wea = (b_c_lock) ? b_c_we_reg: (b_b_lock) ? b_b_we_reg: (b_f_lock) ? b_f_we_reg
1594
1595        // Instantiate Memory Array for TX/RX parity bits
1596        reg parity_tx [0:255];
1597        reg parity_rx [0:255];
1598
1599        // Track rise of CRC_D_RX_ready
1600        reg D_RX_ready_l;
1601        wire D_RX_ready_rise;
1602        always @(posedge clk_40mhz) D_RX_ready_l <= (reset) ? 1'b0 :D_RX_ready;
1603        assign D_RX_ready_rise = D_RX_ready & ~D_RX_ready_l;
1604
1605        ////////////////////////////////////////////////////////////////////////////////
1606        /// STATE MACHINES
1607        ////////////////////////////////////////////////////////////////////////////////
1608
1609        // Handle back end state machine
1610        reg [4:0] b_state;
1611
1612        // Declare back end state parameters
1613        parameter SB_START       = 5'h00;
1614        parameter SB_START_1     = 5'h01;
1615        parameter SB_IDLE        = 5'h02;
1616        parameter SB_DATA        = 5'h03;
1617        parameter SB_DATA_1      = 5'h04;
1618        parameter SB_DATA_2      = 5'h05;
1619        parameter SB_DATA_3      = 5'h06;
1620        parameter SB_DATA_4      = 5'h07;
1621        parameter SB_DATA_5      = 5'h08;
1622        parameter SB_DATA_6      = 5'h09;
1623        parameter SB_DATA_7      = 5'h0A;
1624        parameter SB_DATA_8      = 5'h0B;
1625        parameter SB_DATA_9      = 5'h0C;
1626        parameter SB_DATA_10     = 5'h0D;
1627        parameter SB_DATA_11     = 5'h0E;
1628        parameter SB_DATA_12     = 5'h0F;
1629        parameter SB_DATA_ACK    = 5'h10;
1630        parameter SB_DATA_ACK_1  = 5'h11;
1631        parameter SB_DATA_ACK_2  = 5'h12;
1632        parameter SB_DATA_ACK_3  = 5'h13;
1633        parameter SB_DATA_ACK_4  = 5'h14;
1634        parameter SB_DATA_ACK_5  = 5'h15;
1635        parameter SB_DATA_ACK_6  = 5'h16;
1636        parameter SB_DATA_ACK_7  = 5'h17;
1637
1638        genvar i;
1639        reg global_start;
1640        reg RX_start;
1641        reg [4:0] ACK_pckt_num;
1642        reg [7:0] sender;
```

```
1643            always @(posedge clk_40mhz) begin
1644               if (reset) begin
1645                  f1w_b_lock <= 0;
1646                  f1r_b_lock <= 0;
1647                  f2w_b_lock <= 0;
1648                  b_b_lock <= 0;
1649                  global_start <= 0;
1650                  b_state <= SB_START;
1651         //         generate
1652         //            for (i=0; i<256; i=i+1)
1653         //               begin: INIT_LOOP
1654         //                  parity_rx[i] <= 1'd0;
1655         //                  parity_tx[i] <= 1'd0;
1656         //               end
1657         //         endgenerate
1658               end
1659               else begin
1660                  case (b_state)
1661                     SB_START: begin // Fill outgoing FIFO with 1-32
1662                        if (start) begin
1663                           f1w_b_lock <= 1;
1664                           f1_b_we_reg <= 1;
1665                           f1_b_din_reg <= 5'd0;
1666                           b_state <= b_state + 1;
1667                        end
1668                     end
1669                     SB_START_1: begin
1670                        if (f1_b_din_reg == 5'd31) begin
1671                           f1w_b_lock <= 0;
1672                           f1_b_we_reg <= 0;
1673                           b_state <= b_state + 1;
1674                           global_start <= 1;
1675                        end
1676                        else begin
1677                           f1_b_din_reg <= f1_b_din_reg + 1;
1678                        end
1679                     end
1680                     SB_IDLE: begin
1681                        f1w_b_lock <= 0;
1682                        f1r_b_lock <= 0;
1683                        f2w_b_lock <= 0;
1684                        b_b_lock <= 0;
1685                        if (D_RX_ready_rise) begin
1686                           case (D_RX)
1687                              REQ_DATA: begin
1688                                 b_state <= SB_DATA;
1689                              end
1690                              REQ_DATA_ACK: begin
1691                                 b_state <= SB_DATA_ACK;
1692                              end
1693                              default:;  // do nothing otherwise
1694                           endcase
1695                        end
1696                     end
1697
1698                     ///////////////////////////////////////////////////////////////////////////////
1699                     // DATA PROTOCOL
1700
         ////////////////////////////////////////////////////////////////////////////
1701
1702                     SB_DATA: begin // next byte is Receiver ID
```

```
1703                        if (D_RX == self_phn_num) begin  // data for us
1704                           b_state <= b_state + 1;
1705                        end
1706                        else begin  // data for different node
1707                           b_state <= SB_IDLE;
1708                        end
1709                     end
1710                     SB_DATA_1: begin  // next byte is Sender ID
1711                        sender <= D_RX;
1712                        b_state <= b_state + 1;
1713                     end
1714                     SB_DATA_2: begin  // next byte is parity + packet number
1715                        if (parity_rx[sender] == D_RX[7]) begin  // right data parity, output data
         and update parity
1716                           ACK_pckt_num <= D_RX[4:0];
1717                           parity_rx[sender] <= ~D_RX[7];
1718                           RX_start <= 1;
1719                        end
1720                        else begin  // wrong data parity
1721                           ACK_pckt_num <= D_RX[4:0];
1722                        end
1723                        b_state <= b_state + 1;
1724                     end
1725                     SB_DATA_3: begin  // Send back an ACK
1726                        RX_start <= 0;
1727                        if (~f1r_f_lock) begin
1728                           if (f1_empty) begin  // no spaces available to send, go back to idle
1729                              b_state <= SB_IDLE;
1730                           end
1731                           else begin
1732                              f1r_b_lock <= 1;
1733                              f1_b_re_reg <= 1;
1734                              b_state <= b_state + 1;
1735                           end
1736                        end
1737                     end
1738                     SB_DATA_4: begin  // spaces available to send, read from queue
1739                        f1_b_re_reg <= 0;
1740                        b_state <= b_state + 1;
1741                     end
1742                     SB_DATA_5: begin  // set request
1743                        pckt_num_b_reg <= f1_dout;
1744                        b_state <= b_state + 1;
1745                     end
1746                     SB_DATA_6: begin
1747                        if (CRC_sts == 0) begin
1748                           D_TX_b_reg <= REQ_DATA_ACK;
1749                           D_TX_ready_b_reg <= 1;
1750                           b_state <= b_state + 1;
1751                        end
1752                     end
1753                     SB_DATA_7: begin  // set Receiver ID
1754                        D_TX_b_reg <= sender;
1755                        b_state <= b_state + 1;
1756                     end
1757                     SB_DATA_8: begin  // set Sender ID
1758                        D_TX_b_reg <= self_phn_num;
1759                        b_state <= b_state + 1;
1760                     end
1761                     SB_DATA_9: begin  // set parity + packet number
1762                        D_TX_b_reg <= {parity_rx[sender], 2'b00, ACK_pckt_num};
```

```
1763                      b_state <= b_state + 1;
1764                   end
1765                   SB_DATA_10: begin  // release current lock, acquire lock on sending FIFO
1766                      f1r_b_lock <= 0;
1767                      D_TX_ready_b_reg <= 0;
1768                      if (~f2w_f_lock) begin
1769                         f2w_b_lock <= 1;
1770                         f2_b_we_reg <= 0;
1771                         b_state <= b_state + 1;
1772                      end
1773                   end
1774                   SB_DATA_11: begin  // write to sending FIFO when not full
1775                      if (~f2_full) begin
1776                         f2_b_we_reg <= 1;
1777                         f2_b_din_reg <= {1'b1, pckt_num_b_reg};
1778                         b_state <= b_state + 1;
1779                      end
1780                   end
1781                   SB_DATA_12: begin
1782                      f2w_b_lock <= 0;
1783                      f2_b_we_reg <= 0;
1784                      b_state <= SB_IDLE;
1785                   end
1786
1787                   ///////////////////////////////////////////////////////////////////////////
1788                   // DATA ACK PROTOCOL
1789
         ///////////////////////////////////////////////////////////////////////////
1790
1791                   SB_DATA_ACK: begin  // next byte is Receiver ID
1792                      if (D_RX == self_phn_num) begin  // data for us
1793                         b_state <= b_state + 1;
1794                      end
1795                      else begin  // data for different node
1796                         b_state <= SB_IDLE;
1797                      end
1798                   end
1799                   SB_DATA_ACK_1: begin  // next byte is Sender ID
1800                      sender <= D_RX;
1801                      b_state <= b_state + 1;
1802                   end
1803                   SB_DATA_ACK_2: begin  // next byte is parity + packet number
1804                      if (parity_tx[sender] != D_RX[7]) begin  // right data parity, output data
1805                         ACK_pckt_num <= D_RX[4:0];
1806                         parity_tx[sender] <= D_RX[7];
1807                      end
1808                      else begin  // wrong data parity
1809                         ACK_pckt_num <= D_RX[4:0];
1810                      end
1811                      b_state <= b_state + 1;
1812                   end
1813                   SB_DATA_ACK_3: begin  // acquire write lock on empty spots in queue
1814                      if (~f1w_s_lock) begin
1815                         f1w_b_lock <= 1;
1816                         f1_b_we_reg <= 0;
1817                         b_state <= b_state + 1;
1818                      end
1819                   end
1820                   SB_DATA_ACK_4: begin  // This lock has low priority, manage appropriately
1821                      if (f1w_s_lock) begin
1822                         f1w_b_lock <= 0;
```

```
1823                          b_state <= b_state - 1;
1824                       end
1825                       else begin
1826                          f1_b_we_reg <= 1;
1827                          f1_b_din_reg <= ACK_pckt_num;
1828                          b_state <= b_state + 1;
1829                       end
1830                    end
1831                    SB_DATA_ACK_5: begin // Write finished, release lock. Acquire lock on BRAM
1832                       f1w_b_lock <= 0;
1833                       f1_b_we_reg <= 0;
1834                       if (~b_c_lock & ~b_f_lock) begin
1835                          b_b_lock <= 1;
1836                          b_b_we_reg <= 0;
1837                          b_state <= b_state + 1;
1838                       end
1839                    end
1840                    SB_DATA_ACK_6: begin  // This lock has medium priority, manage appropriately
1841                       if (b_c_lock) begin
1842                          b_b_lock <= 0;
1843                          b_state <= b_state - 1;
1844                       end
1845                       else begin
1846                          b_b_we_reg <= 1;
1847                          b_b_addr_reg <= sender;
1848                          b_b_din_reg <= 16'h00;
1849                          b_state <= b_state + 1;
1850                       end
1851                    end
1852                    SB_DATA_ACK_7: begin  // Write finished, release lock.
1853                       b_b_lock <= 0;
1854                       b_b_we_reg <= 0;
1855                       b_state <= SB_IDLE;
1856                    end
1857                 endcase
1858              end
1859           end
1860
1861           // Manage network data output
1862           reg started;
1863           always @(posedge clk_40mhz) begin
1864              if (reset | ~global_start) begin
1865                 NW_D_RX_reg <= 8'd0;
1866                 NW_D_RX_ready_reg <= 0;
1867                 NW_D_RX_addr_reg <= 0;
1868                 started <= 0;
1869              end
1870              else if (RX_start) begin
1871                 if (D_RX_ready) begin
1872                    started <= 1;
1873                    NW_D_RX_reg <= D_RX;
1874                    NW_D_RX_ready_reg <= 1;
1875                    NW_D_RX_addr_reg <= sender;
1876                 end
1877                 else begin
1878                    started <= 0;
1879                    NW_D_RX_ready_reg <= 0;
1880                 end
1881              end
1882              else if (started) begin
1883                 if (D_RX_ready) begin
```

```
1884                NW_D_RX_reg <= D_RX;
1885                NW_D_RX_ready_reg <= 1;
1886             end
1887             else begin
1888                NW_D_RX_ready_reg <= 0;
1889                started <= 0;
1890             end
1891          end
1892          else begin
1893             NW_D_RX_ready_reg <= 0;
1894             started <= 0;
1895          end
1896       end
1897
1898       // Handle front end state machine
1899       reg [4:0] f_state;
1900
1901       // Declare front end state parameters
1902       parameter SF_IDLE    = 5'h00;
1903       parameter SF_CHECK   = 5'h01;
1904       parameter SF_CHECK_1 = 5'h02;
1905       parameter SF_CHECK_2 = 5'h03;
1906       parameter SF_SEND    = 5'h04;
1907       parameter SF_SEND_1  = 5'h05;
1908       parameter SF_SEND_2  = 5'h06;
1909       parameter SF_SEND_3  = 5'h07;
1910       parameter SF_SEND_4  = 5'h08;
1911       parameter SF_SEND_5  = 5'h09;
1912       parameter SF_SEND_6  = 5'h0A;
1913       parameter SF_SEND_7  = 5'h0B;
1914       parameter SF_SEND_8  = 5'h0C;
1915       parameter SF_SEND_9  = 5'h0D;
1916       parameter SF_SEND_10 = 5'h0E;
1917       parameter SF_SEND_11 = 5'h0F;
1918       parameter SF_SEND_12 = 5'h10;
1919
1920       reg [7:0] NW_D_TX_addr_l;
1921       always @(posedge clk_40mhz) begin
1922          if (reset | ~global_start) begin
1923             f1r_f_lock <= 0;
1924             f2w_f_lock <= 0;
1925             b_f_lock <= 0;
1926             f_state <= SF_IDLE;
1927             NW_sts_reg <= STS_BUSY;
1928          end
1929          else begin
1930             case (f_state)
1931                SF_IDLE: begin
1932                   f1r_f_lock <= 0;
1933                   f2w_f_lock <= 0;
1934                   b_f_lock <= 0;
1935                   if (NW_cmd == CMD_TX) begin
1936                      NW_sts_reg <= STS_BUSY;
1937                      NW_D_TX_addr_l <= NW_D_TX_addr;
1938                      f_state <= f_state + 1;
1939                   end
1940                   else begin
1941                      NW_sts_reg <= STS_IDLE;
1942                   end
1943                end
1944                SF_CHECK: begin
```

```
1945                         if (doutb[15]) begin  // outgoing packet already exists
1946                            NW_sts_reg <= STS_TX_REJECT;
1947                            f_state <= SF_IDLE;
1948                         end
1949                         else begin
1950                            f_state <= f_state + 1;
1951                         end
1952                      end
1953                      SF_CHECK_1: begin // acquire lock on open buffer spots FIFO
1954                         if (~f1r_b_lock) begin
1955                            f1r_f_lock <= 1;
1956                            f1_f_re_reg <= 0;
1957                            f_state <= f_state + 1;
1958                         end
1959                      end
1960                      SF_CHECK_2: begin // This lock has low priority, manage appropriately
1961                         if (f1r_b_lock) begin
1962                            f1r_f_lock <= 0;
1963                            f_state <= f_state - 1;
1964                         end
1965                         else if (f1_empty) begin  // no more available spots in buffer
1966                            f1r_f_lock <= 0;
1967                            f_state <= SF_IDLE;
1968                            NW_sts_reg <= STS_TX_REJECT;
1969                         end
1970                         else begin
1971                            f1_f_re_reg <= 1;
1972                            f_state <= f_state + 1;
1973                         end
1974                      end
1975                      SF_SEND: begin
1976                         f1_f_re_reg <= 0;
1977                         f_state <= f_state + 1;
1978                      end
1979                      SF_SEND_1: begin
1980                         pckt_num_f_reg <= f1_dout;
1981                         f_state <= f_state + 1;
1982                      end
1983                      SF_SEND_2: begin  // First byte is request type
1984                         if (CRC_sts == 0) begin
1985                            D_TX_ready_f_reg <= 1;
1986                            D_TX_f_reg <= REQ_DATA;
1987                            f_state <= f_state + 1;
1988                         end
1989                      end
1990                      SF_SEND_3: begin  // Next byte is Receiver ID
1991                         D_TX_f_reg <= NW_D_TX_addr_l;
1992                         f_state <= f_state + 1;
1993                      end
1994                      SF_SEND_4: begin  // Next byte is Sender ID
1995                         D_TX_f_reg <= self_phn_num;
1996                         f_state <= f_state + 1;
1997                         NW_sts_reg <= STS_TX_ACCEPT;
1998                      end
1999                      SF_SEND_5: begin  // Next byte is parity + packet number
2000                         D_TX_f_reg <=  {parity_tx[NW_D_TX_addr_l], 2'b00, pckt_num_f_reg};
2001                         f_state <= f_state + 1;
2002                      end
2003                      SF_SEND_6: begin  // Rest is data
2004                         if (NW_D_TX_ready) begin
2005                            D_TX_f_reg <= NW_D_TX;
```

```
2006                        end
2007                        else begin
2008                           f1r_f_lock <= 0;
2009                           D_TX_ready_f_reg <= 0;
2010                           f_state <= f_state + 1;
2011                        end
2012                     end
2013                     SF_SEND_7: begin  // Acquire write lock on send FIFO
2014                        if (~f2w_b_lock) begin
2015                           f2w_f_lock <= 1;
2016                           f2_f_we_reg <= 0;
2017                           f_state <= f_state + 1;
2018                        end
2019                     end
2020                     SF_SEND_8: begin  // This lock has low priority, manage appropriately
2021                        if (f2w_b_lock) begin
2022                           f2w_f_lock <= 0;
2023                           f_state <= f_state - 1;
2024                        end
2025                        else begin
2026                           f_state <= f_state + 1;
2027                        end
2028                     end
2029                     SF_SEND_9: begin
2030                        if (~f2_full) begin
2031                           f2_f_we_reg <= 1;
2032                           f2_f_din_reg <= {1'b0, pckt_num_f_reg};
2033                           f_state <= f_state + 1;
2034                        end
2035                     end
2036                     SF_SEND_10: begin // Release lock on sending FIFO, acquire lock on BRAM
2037                        f2w_f_lock <= 0;
2038                        f2_f_we_reg <= 0;
2039                        if (~b_b_lock & ~b_c_lock) begin
2040                           b_f_lock <= 1;
2041                           b_f_we_reg <= 0;
2042                           f_state <= f_state + 1;
2043                        end
2044                     end
2045                     SF_SEND_11: begin // This lock has low priority, manage appropriately
2046                        if (b_b_lock | b_c_lock) begin
2047                           b_f_lock <= 0;
2048                           f_state <= f_state - 1;
2049                        end
2050                        else begin
2051                           b_f_we_reg <= 1;
2052                           b_f_addr_reg <= NW_D_TX_addr_l;
2053                           b_f_din_reg <= {1'b1, pckt_num_f_reg, P_DELETE_TIMEOUT[9:0]};
2054                           f_state <= f_state + 1;
2055                        end
2056                     end
2057                     SF_SEND_12: begin
2058                        b_f_lock <= 0;
2059                        b_f_we_reg <= 0;
2060                        f_state <= SF_IDLE;
2061                     end
2062                  endcase
2063               end
2064            end
2065
2066            // Handle MAC send commands state machine
```

```
2067          reg [3:0] s_state;
2068
2069          // Declare send state machine parameters
2070          parameter SS_IDLE   = 4'h0;
2071          parameter SS_SEND   = 4'h1;
2072          parameter SS_SEND_1 = 4'h2;
2073          parameter SS_SEND_2 = 4'h3;
2074          parameter SS_SEND_3 = 4'h4;
2075          parameter SS_SEND_4 = 4'h5;
2076          parameter SS_DEL    = 4'h6;
2077          parameter SS_DEL_1  = 4'h7;
2078          parameter SS_DEL_2  = 4'h8;
2079
2080          reg del_packet;
2081          always @(posedge clk_40mhz) begin
2082             if (reset | ~global_start) begin
2083                s_state <= SS_IDLE;
2084                f2_s_re_reg <= 0;
2085                MAC_cmd_reg <= 0;
2086                MAC_pckt_num_reg <= 5'd0;
2087             end
2088             else begin
2089                case (s_state)
2090                   SS_IDLE: begin
2091                      if (~f2_empty) begin
2092                         f2_s_re_reg <= 1;
2093                         s_state <= s_state + 1;
2094                      end
2095                      else begin
2096                         f2_s_re_reg <= 0;
2097                      end
2098                   end
2099                   SS_SEND: begin  // Send write command to MAC
2100                      f2_s_re_reg <= 0;
2101                      s_state <= s_state + 1;
2102                   end
2103                   SS_SEND_1: begin
2104                      del_packet <= f2_dout[5];
2105                      MAC_pckt_num_reg <= f2_dout[4:0];
2106                      s_state <= s_state + 1;
2107                   end
2108                   SS_SEND_2: begin  // Wait until MAC is idle
2109                      if (MAC_sts == 0) begin
2110                         MAC_cmd_reg <= 1;
2111                         s_state <= s_state + 1;
2112                      end
2113                   end
2114                   SS_SEND_3: begin
2115                      MAC_cmd_reg <= 0;
2116                      if (del_packet) begin
2117                         s_state <= s_state + 1;
2118                      end
2119                      else begin
2120                         s_state <= SS_IDLE;
2121                      end
2122                   end
2123                   SS_SEND_4: begin  // wait for MAC to finish sending before deletion
2124                      if (MAC_sts == 0) begin
2125                         s_state <= s_state + 1;
2126                      end
2127                   end
```

```
2128                  SS_DEL: begin  // Delete packet if it was an ACK. Acquire write lock.
2129                     if (~f1w_b_lock) begin
2130                        f1w_s_lock <= 1;
2131                        f1_s_we_reg <= 0;
2132                        s_state <= s_state + 1;
2133                     end
2134                  end
2135                  SS_DEL_1: begin  // This lock has low priority, manage appropriately
2136                     if (f1w_b_lock) begin
2137                        f1w_s_lock <= 0;
2138                        s_state <= s_state - 1;
2139                     end
2140                     else begin
2141                        f1_s_we_reg <= 1;
2142                        f1_s_din_reg <= MAC_pckt_num_reg;
2143                        s_state <= s_state + 1;
2144                     end
2145                  end
2146                  SS_DEL_2: begin  // Release lock
2147                     f1w_s_lock <= 0;
2148                     f1_s_we_reg <= 0;
2149                     s_state <= SS_IDLE;
2150                  end
2151               endcase
2152            end
2153         end
2154
2155         // Handle ARQ timeout state machine
2156         reg [3:0] c_state;
2157
2158         // Define ARQ timeout state parameters
2159         parameter SC_IDLE   = 4'h0;
2160         parameter SC_DEC    = 4'h1;
2161         parameter SC_DEC_1  = 4'h2;
2162         parameter SC_DEC_2  = 4'h3;
2163         parameter SC_DEC_3  = 4'h4;
2164         parameter SC_DEL    = 4'h5;
2165         parameter SC_DEL_1  = 4'h6;
2166         parameter SC_DEL_2  = 4'h7;
2167         parameter SC_SEND   = 4'h8;
2168         parameter SC_SEND_1 = 4'h9;
2169         parameter SC_SEND_2 = 4'hA;
2170         parameter SC_SEND_3 = 4'hB;
2171
2172         reg [2:0] cntr;
2173         reg [4:0] c_pckt_num;
2174         wire [9:0] timeout_minus_one;
2175         assign timeout_minus_one = douta[9:0] - 1;
2176         always @(posedge clk_40mhz) begin
2177            if (reset | ~global_start) begin
2178               f2w_c_lock <= 0;
2179               f1w_c_lock <= 0;
2180               b_c_lock <= 0;
2181               c_state <= SC_IDLE;
2182               cntr <= 3'd0;
2183               b_c_addr_reg <= 0;
2184            end
2185            else begin
2186               case (c_state)
2187                  SC_IDLE: begin  // acquire lock on BRAM
2188                     if (~b_f_lock & ~b_b_lock & !cntr) begin
```

```
2189                        b_c_lock <= 1;
2190                        c_state <= c_state + 1;
2191                     end
2192                     else if (cntr) begin
2193                        cntr <= cntr - 1;
2194                        b_c_lock <= 0;
2195                     end
2196                     else begin
2197                        b_c_lock <= 0;
2198                     end
2199                     b_c_we_reg <= 0;
2200                     f2w_c_lock <= 0;
2201                     f1w_c_lock <= 0;
2202                  end
2203                  SC_DEC: begin  // guaranteed to get lock
2204                     c_state <= c_state + 1;
2205                  end
2206                  SC_DEC_1: begin  // wait a state for the address to be set for the read
2207                     c_state <= c_state + 1;
2208                  end
2209                  SC_DEC_2: begin
2210                     if (douta[15]) begin
2211                        b_c_we_reg <= 1;
2212                        b_c_din_reg <= {(|douta[9:0]), douta[14:10], timeout_minus_one};
2213                        c_pckt_num <= douta[14:10];
2214                        if (!douta[9:0]) begin // delete the packet
2215                           c_state <= SC_DEL;
2216                        end
2217                        else if (!douta[P_LOG_SIZE_ARQ_TIMEOUT-1:0]) begin // resend the packe
2218                           c_state <= SC_SEND;
2219                        end
2220                        else begin // decrement ARQ timeout
2221                           c_state <= c_state + 1;
2222                        end
2223                     end
2224                     else begin  // no outstanding message, release the lock
2225                        b_c_lock <= 0;
2226                        b_c_we_reg <= 0;
2227                        b_c_addr_reg <= b_c_addr_reg + 1;
2228                        cntr <= 3'd7;
2229                        c_state <= SC_IDLE;
2230                     end
2231                  end
2232                  SC_DEC_3: begin
2233                        b_c_lock <= 0;
2234                        b_c_we_reg <= 0;
2235                        b_c_addr_reg <= b_c_addr_reg + 1;
2236                        cntr <= 3'd7;
2237                        c_state <= SC_IDLE;
2238                  end
2239                  SC_DEL: begin // get lock on open spots FIFO
2240                     b_c_lock <= 0;
2241                     b_c_we_reg <= 0;
2242                     if (~f1w_b_lock & ~f1w_s_lock) begin
2243                        f1w_c_lock <= 1;
2244                        f1_c_we_reg <= 0;
2245                        c_state <= c_state + 1;
2246                     end
2247                  end
2248                  SC_DEL_1: begin
2249                     if (f1w_b_lock | f1w_s_lock) begin
```

```
2250                    f1w_c_lock <= 0;
2251                    c_state <= c_state - 1;
2252                 end
2253                 else begin
2254                    f1_c_we_reg <= 1;
2255                    f1_c_din_reg <= c_pckt_num;
2256                    c_state <= c_state + 1;
2257                 end
2258              end
2259              SC_DEL_2: begin  // release lock
2260                 f1_c_we_reg <= 0;
2261                 f1w_c_lock <= 0;
2262                 b_c_addr_reg <= b_c_addr_reg + 1;
2263                 cntr <= 3'd7;
2264                 c_state <= SC_IDLE;
2265              end
2266              SC_SEND: begin  // get lock on sending FIFO
2267                 b_c_lock <= 0;
2268                 b_c_we_reg <= 0;
2269                 if (~f2w_b_lock & ~f2w_f_lock) begin
2270                    f2w_c_lock <= 1;
2271                    f2_c_we_reg <= 0;
2272                    c_state <= c_state + 1;
2273                 end
2274              end
2275              SC_SEND_1: begin
2276                 if (f2w_b_lock | f2w_f_lock) begin
2277                    f2w_c_lock <= 0;
2278                    c_state <= c_state - 1;
2279                 end
2280                 else begin
2281                    c_state <= c_state + 1;
2282                 end
2283              end
2284              SC_SEND_2: begin
2285                 if (~f2_full) begin
2286                    f2_c_we_reg <= 1;
2287                    f2_c_din_reg <= c_pckt_num;
2288                    c_state <= c_state + 1;
2289                 end
2290              end
2291              SC_SEND_3: begin
2292                 f2_c_we_reg <= 0;
2293                 f2w_c_lock <= 0;
2294                 b_c_addr_reg <= b_c_addr_reg + 1;
2295                 cntr <= 3'd7;
2296                 c_state <= SC_IDLE;
2297              end
2298           endcase
2299        end
2300     end
2301
2302     // DEBUG
2303     assign b_state_DEBUG = b_state;
2304     assign f_state_DEBUG = f_state;
2305     assign s_state_DEBUG = s_state;
2306     assign c_state_DEBUG = c_state;
2307     assign b_f_lock_DEBUG = b_f_lock;
2308     assign b_b_lock_DEBUG = b_b_lock;
2309     assign b_c_lock_DEBUG = b_c_lock;
2310     assign f1w_s_lock_DEBUG = f1w_s_lock;
```

```
2311            assign f1w_b_lock_DEBUG = f1w_b_lock;
2312            assign f1w_c_lock_DEBUG = f1w_c_lock;
2313            assign f1r_f_lock_DEBUG = f1r_f_lock;
2314            assign f1r_b_lock_DEBUG = f1r_b_lock;
2315            assign f2w_f_lock_DEBUG = f2w_f_lock;
2316            assign f2w_b_lock_DEBUG = f2w_b_lock;
2317            assign f2w_c_lock_DEBUG  = f2w_c_lock;
2318            assign p_DEBUG = parity_tx[8'hAD];
2319            assign dina_DEBUG = dina;
2320            assign wea_DEBUG = wea;
2321            assign addra_DEBUG = addra;
2322            assign douta_DEBUG = douta;
2323         endmodule
```

```
1	`timescale 1ns / 1ps
2	////////////////////////////////////////////////////////////////////////////////
3	// Company:
4	// Engineer:       Sachin Shinde
5	//
6	// Create Date:    04:08:44 11/22/2012
7	// Design Name:
8	// Module Name:    Button_Contention_Resolver
9	// Project Name:
10	// Target Devices:
11	// Tool versions:
12	// Description:    Takes debounced button inputs, and manages button contention
13	//                 such that only one button signal is high per clock. There is
14	//                 also guaranteed to be at least one cycle of no button presses
15	//                 between consecutive button presses. This is useful for
16	//                 generically determining when a button is released, as it
17	//                 becomes the bitwise-or of the button signals.
18	//
19	// Dependencies:
20	//
21	// Revision:
22	// Revision 0.01 - File Created
23	// Additional Comments:
24	//
25	////////////////////////////////////////////////////////////////////////////////
26	module Button_Contention_Resolver(
27	   input clk,
28	   input reset,
29	   // Button Inputs
30	   input button0_in,
31	   input button1_in,
32	   input button2_in,
33	   input button3_in,
34	   input button_enter_in,
35	   input button_left_in,
36	   input button_right_in,
37	   input button_up_in,
38	   input button_down_in,
39	   // Button Outputs
40	   output button0_out,
41	   output button1_out,
42	   output button2_out,
43	   output button3_out,
44	   output button_enter_out,
45	   output button_left_out,
46	   output button_right_out,
47	   output button_up_out,
48	   output button_down_out
49	   );
50
51	   // Instantiate state var
52	   reg state;
53
54	   // Define state parameters
55	   parameter S_RESET = 0;
56	   parameter S_SET =   1;
57
58	   // Assign input
59	   wire [8:0] button_in;
60	   assign button_in =
61	          {button0_in, button1_in, button2_in, button3_in, button_enter_in,
```

```
62                 button_left_in, button_right_in, button_up_in, button_down_in};
63
64         // Assign output
65         reg [8:0] button_out;
66         assign {button0_out, button1_out, button2_out, button3_out, button_enter_out,
67                 button_left_out, button_right_out, button_up_out, button_down_out}
68                = button_out;
69
70         // Manage state transitions and output
71         wire [8:0] button_in_minus_one;
72         assign button_in_minus_one = button_in - 1;
73         always @(posedge clk) begin
74            if (reset) begin
75               state <= S_RESET;
76               button_out <= 9'd0;
77            end
78            else begin
79               case (state)
80                  S_RESET: begin
81                     if ((|button_in) & !(button_in_minus_one & button_in)) begin
82                        state <= S_SET;
83                        button_out <= button_in;
84                     end
85                  end
86                  S_SET: begin
87                     if (!(button_out & button_in)) begin
88                        state <= S_RESET;
89                        button_out <= 9'd0;
90                     end
91                  end
92               endcase
93            end
94         end
95      endmodule
96
```

```
1     ///////////////////////////////////////////////////////////////////////////////
2     //
3     // 6.111 FPGA Labkit -- Template Toplevel Module
4     //
5     // For Labkit Revision 004
6     //
7     //
8     // Created: October 31, 2004, from revision 003 file
9     // Author: Sachin Shinde
10    //
11    ///////////////////////////////////////////////////////////////////////////////
12    //
13    // CHANGES FOR BOARD REVISION 004
14    //
15    // 1) Added signals for logic analyzer pods 2-4.
16    // 2) Expanded "tv_in_ycrcb" to 20 bits.
17    // 3) Renamed "tv_out_data" to "tv_out_i2c_data" and "tv_out_sclk" to
18    //    "tv_out_i2c_clock".
19    // 4) Reversed disp_data_in and disp_data_out signals, so that "out" is an
20    //    output of the FPGA, and "in" is an input.
21    //
22    // CHANGES FOR BOARD REVISION 003
23    //
24    // 1) Combined flash chip enables into a single signal, flash_ce_b.
25    //
26    // CHANGES FOR BOARD REVISION 002
27    //
28    // 1) Added SRAM clock feedback path input and output
29    // 2) Renamed "mousedata" to "mouse_data"
30    // 3) Renamed some ZBT memory signals. Parity bits are now incorporated into
31    //    the data bus, and the byte write enables have been combined into the
32    //    4-bit ram#_bwe_b bus.
33    // 4) Removed the "systemace_clock" net, since the SystemACE clock is now
34    //    hardwired on the PCB to the oscillator.
35    //
36    ///////////////////////////////////////////////////////////////////////////////
37    //
38    // Complete change history (including bug fixes)
39    //
40    // 2006-Mar-08: Corrected default assignments to "vga_out_red", "vga_out_green"
41    //              and "vga_out_blue". (Was 10'h0, now 8'h0.)
42    //
43    // 2005-Sep-09: Added missing default assignments to "ac97_sdata_out",
44    //              "disp_data_out", "analyzer[2-3]_clock" and
45    //              "analyzer[2-3]_data".
46    //
47    // 2005-Jan-23: Reduced flash address bus to 24 bits, to match 128Mb devices
48    //              actually populated on the boards. (The boards support up to
49    //              256Mb devices, with 25 address lines.)
50    //
51    // 2004-Oct-31: Adapted to new revision 004 board.
52    //
53    // 2004-May-01: Changed "disp_data_in" to be an output, and gave it a default
54    //              value. (Previous versions of this file declared this port to
55    //              be an input.)
56    //
57    // 2004-Apr-29: Reduced SRAM address busses to 19 bits, to match 18Mb devices
58    //              actually populated on the boards. (The boards support up to
59    //              72Mb devices, with 21 address lines.)
60    //
61    // 2004-Apr-29: Change history started
```

```
 62       //
 63       ///////////////////////////////////////////////////////////////////////////////
 64
 65       module labkit (beep, audio_reset_b, ac97_sdata_out, ac97_sdata_in, ac97_synch,
 66                ac97_bit_clock,
 67
 68                vga_out_red, vga_out_green, vga_out_blue, vga_out_sync_b,
 69                vga_out_blank_b, vga_out_pixel_clock, vga_out_hsync,
 70                vga_out_vsync,
 71
 72                tv_out_ycrcb, tv_out_reset_b, tv_out_clock, tv_out_i2c_clock,
 73                tv_out_i2c_data, tv_out_pal_ntsc, tv_out_hsync_b,
 74                tv_out_vsync_b, tv_out_blank_b, tv_out_subcar_reset,
 75
 76                tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1,
 77                tv_in_line_clock2, tv_in_aef, tv_in_hff, tv_in_aff,
 78                tv_in_i2c_clock, tv_in_i2c_data, tv_in_fifo_read,
 79                tv_in_fifo_clock, tv_in_iso, tv_in_reset_b, tv_in_clock,
 80
 81                ram0_data, ram0_address, ram0_adv_ld, ram0_clk, ram0_cen_b,
 82                ram0_ce_b, ram0_oe_b, ram0_we_b, ram0_bwe_b,
 83
 84                ram1_data, ram1_address, ram1_adv_ld, ram1_clk, ram1_cen_b,
 85                ram1_ce_b, ram1_oe_b, ram1_we_b, ram1_bwe_b,
 86
 87                clock_feedback_out, clock_feedback_in,
 88
 89                flash_data, flash_address, flash_ce_b, flash_oe_b, flash_we_b,
 90                flash_reset_b, flash_sts, flash_byte_b,
 91
 92                rs232_txd, rs232_rxd, rs232_rts, rs232_cts,
 93
 94                mouse_clock, mouse_data, keyboard_clock, keyboard_data,
 95
 96                clock_27mhz, clock1, clock2,
 97
 98                disp_blank, disp_data_out, disp_clock, disp_rs, disp_ce_b,
 99                disp_reset_b, disp_data_in,
100
101                button0, button1, button2, button3, button_enter, button_right,
102                button_left, button_down, button_up,
103
104                switch,
105
106                led,
107
108                user1, user2, user3, user4,
109
110                daughtercard,
111
112                systemace_data, systemace_address, systemace_ce_b,
113                systemace_we_b, systemace_oe_b, systemace_irq, systemace_mpbrdy,
114
115                analyzer1_data, analyzer1_clock,
116                analyzer2_data, analyzer2_clock,
117                analyzer3_data, analyzer3_clock,
118                analyzer4_data, analyzer4_clock);
119
120          output beep, audio_reset_b, ac97_synch, ac97_sdata_out;
121          input  ac97_bit_clock, ac97_sdata_in;
122
```

```
123            output [7:0] vga_out_red, vga_out_green, vga_out_blue;
124            output vga_out_sync_b, vga_out_blank_b, vga_out_pixel_clock,
125              vga_out_hsync, vga_out_vsync;
126
127            output [9:0] tv_out_ycrcb;
128            output tv_out_reset_b, tv_out_clock, tv_out_i2c_clock, tv_out_i2c_data,
129              tv_out_pal_ntsc, tv_out_hsync_b, tv_out_vsync_b, tv_out_blank_b,
130              tv_out_subcar_reset;
131
132            input  [19:0] tv_in_ycrcb;
133            input  tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2, tv_in_aef,
134              tv_in_hff, tv_in_aff;
135            output tv_in_i2c_clock, tv_in_fifo_read, tv_in_fifo_clock, tv_in_iso,
136              tv_in_reset_b, tv_in_clock;
137            inout  tv_in_i2c_data;
138
139            inout  [35:0] ram0_data;
140            output [18:0] ram0_address;
141            output ram0_adv_ld, ram0_clk, ram0_cen_b, ram0_ce_b, ram0_oe_b, ram0_we_b;
142            output [3:0] ram0_bwe_b;
143
144            inout  [35:0] ram1_data;
145            output [18:0] ram1_address;
146            output ram1_adv_ld, ram1_clk, ram1_cen_b, ram1_ce_b, ram1_oe_b, ram1_we_b;
147            output [3:0] ram1_bwe_b;
148
149            input  clock_feedback_in;
150            output clock_feedback_out;
151
152            inout  [15:0] flash_data;
153            output [23:0] flash_address;
154            output flash_ce_b, flash_oe_b, flash_we_b, flash_reset_b, flash_byte_b;
155            input  flash_sts;
156
157            output rs232_txd, rs232_rts;
158            input  rs232_rxd, rs232_cts;
159
160            input  mouse_clock, mouse_data, keyboard_clock, keyboard_data;
161
162            input  clock_27mhz, clock1, clock2;
163
164            output disp_blank, disp_clock, disp_rs, disp_ce_b, disp_reset_b;
165            input  disp_data_in;
166            output  disp_data_out;
167
168            input  button0, button1, button2, button3, button_enter, button_right,
169              button_left, button_down, button_up;
170            input  [7:0] switch;
171            output [7:0] led;
172
173            inout [31:0] user1, user2, user3, user4;
174
175            inout [43:0] daughtercard;
176
177            inout  [15:0] systemace_data;
178            output [6:0]  systemace_address;
179            output systemace_ce_b, systemace_we_b, systemace_oe_b;
180            input  systemace_irq, systemace_mpbrdy;
181
182            output [15:0] analyzer1_data, analyzer2_data, analyzer3_data,
183                analyzer4_data;
```

```
184          output analyzer1_clock, analyzer2_clock, analyzer3_clock, analyzer4_clock;
185
186          ////////////////////////////////////////////////////////////////////////////
187          //
188          // I/O Assignments
189          //
190          ////////////////////////////////////////////////////////////////////////////
191
192          // Audio Input and Output
193          assign beep= 1'b0;
194       //   assign audio_reset_b = 1'b0;
195       //   assign ac97_synch = 1'b0;
196       //   assign ac97_sdata_out = 1'b0;
197          // ac97_sdata_in is an input
198
199          // VGA Output
200          assign vga_out_red = 8'h0;
201          assign vga_out_green = 8'h0;
202          assign vga_out_blue = 8'h0;
203          assign vga_out_sync_b = 1'b1;
204          assign vga_out_blank_b = 1'b1;
205          assign vga_out_pixel_clock = 1'b0;
206          assign vga_out_hsync = 1'b0;
207          assign vga_out_vsync = 1'b0;
208
209          // Video Output
210          assign tv_out_ycrcb = 10'h0;
211          assign tv_out_reset_b = 1'b0;
212          assign tv_out_clock = 1'b0;
213          assign tv_out_i2c_clock = 1'b0;
214          assign tv_out_i2c_data = 1'b0;
215          assign tv_out_pal_ntsc = 1'b0;
216          assign tv_out_hsync_b = 1'b1;
217          assign tv_out_vsync_b = 1'b1;
218          assign tv_out_blank_b = 1'b1;
219          assign tv_out_subcar_reset = 1'b0;
220
221          // Video Input
222          assign tv_in_i2c_clock = 1'b0;
223          assign tv_in_fifo_read = 1'b0;
224          assign tv_in_fifo_clock = 1'b0;
225          assign tv_in_iso = 1'b0;
226          assign tv_in_reset_b = 1'b0;
227          assign tv_in_clock = 1'b0;
228          assign tv_in_i2c_data = 1'bZ;
229          // tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2,
230          // tv_in_aef, tv_in_hff, and tv_in_aff are inputs
231
232          // SRAMs
233       //   assign ram0_data = 36'hZ;
234       //   assign ram0_address = 19'h0;
235          assign ram0_adv_ld = 1'b0;
236       //   assign ram0_clk = 1'b0;
237          assign ram0_cen_b = 1'b0;
238          assign ram0_ce_b = 1'b0;
239          assign ram0_oe_b = 1'b0;
240       //   assign ram0_we_b = 1'b1;
241       //   assign ram0_bwe_b = 4'hF;
242          assign ram1_data = 36'hZ;
243          assign ram1_address = 19'h0;
244          assign ram1_adv_ld = 1'b0;
```

```
245        //   assign ram1_clk = 1'b0;
246           assign ram1_cen_b = 1'b1;
247           assign ram1_ce_b = 1'b1;
248           assign ram1_oe_b = 1'b1;
249           assign ram1_we_b = 1'b1;
250           assign ram1_bwe_b = 4'hF;
251        //   assign clock_feedback_out = 1'b0;
252           // clock_feedback_in is an input
253
254           // Flash ROM
255           assign flash_data = 16'hZ;
256           assign flash_address = 24'h0;
257           assign flash_ce_b = 1'b1;
258           assign flash_oe_b = 1'b1;
259           assign flash_we_b = 1'b1;
260           assign flash_reset_b = 1'b0;
261           assign flash_byte_b = 1'b1;
262           // flash_sts is an input
263
264           // RS-232 Interface
265           assign rs232_txd = 1'b1;
266           assign rs232_rts = 1'b1;
267           // rs232_rxd and rs232_cts are inputs
268
269           // PS/2 Ports
270           // mouse_clock, mouse_data, keyboard_clock, and keyboard_data are inputs
271
272           // LED Displays
273           //   assign disp_blank = 1'b1;
274           //   assign disp_clock = 1'b0;
275           //   assign disp_rs = 1'b0;
276           //   assign disp_ce_b = 1'b1;
277           //   assign disp_reset_b = 1'b0;
278           //   assign disp_data_out = 1'b0;
279           // disp_data_in is an input
280
281           // Buttons, Switches, and Individual LEDs
282        //   assign led = 8'hFF;
283           // button0, button1, button2, button3, button_enter, button_right,
284           // button_left, button_down, button_up, and switches are inputs
285
286           // User I/Os
287           assign user1 = 32'hZ;
288           assign user2 = 32'hZ;
289           assign user3 = 32'hZ;
290           assign user4 = 32'hZ;
291
292           // Daughtercard Connectors
293           assign daughtercard = 44'hZ;
294
295           // SystemACE Microprocessor Port
296        //   assign systemace_data = 16'hZ;
297        //   assign systemace_address = 7'h0;
298        //   assign systemace_ce_b = 1'b1;
299        //   assign systemace_we_b = 1'b1;
300        //   assign systemace_oe_b = 1'b1;
301           // systemace_irq and systemace_mpbrdy are inputs
302
303           // Logic Analyzer
304           assign analyzer1_data = 16'h0;
305           assign analyzer1_clock = 1'b1;
```

```
306        assign analyzer2_data = 16'h0;
307        assign analyzer2_clock = 1'b1;
308        assign analyzer3_data = 16'h0;
309        assign analyzer3_clock = 1'b1;
310        assign analyzer4_data = 16'h0;
311        assign analyzer4_clock = 1'b1;
312
313     ////////////////////////////////////////////////////////////////////////////
314        //
315        // Reset Generation
316        //
317        // A shift register primitive is used to generate an active-high reset
318        // signal that remains high for 16 clock cycles after configuration finishes
319        // and the FPGA's internal clocks begin toggling.
320        //
321        ////////////////////////////////////////////////////////////////////////////
322        wire clk_27mhz, locked;
323        ramclock RAMCLOCK_1(
324           .ref_clock(clock_27mhz),
325           .fpga_clock(clk_27mhz),
326           .ram0_clock(ram0_clk),
327           .ram1_clock(ram1_clk),
328           .clock_feedback_in(clock_feedback_in),
329           .clock_feedback_out(clock_feedback_out),
330           .locked(locked)
331           );
332
333        wire pre_reset, reset;
334        SRL16 reset_sr(.D(1'b0), .CLK(clk_27mhz), .Q(pre_reset),
335                 .A0(1'b1), .A1(1'b1), .A2(1'b1), .A3(1'b1));
336        defparam reset_sr.INIT = 16'hFFFF;
337
338        assign reset = pre_reset | ~locked;
339
340        /////////////////////////////////////////////////////////////////////////////
341        //
342        // Main Modules
343        //
344        /////////////////////////////////////////////////////////////////////////////
345
346
347        wire b_up, b_down, b_left, b_right, b_enter, b0, b1, b2, b3;
348        wire br_up, br_down, br_left, br_right, br_enter, br0, br1, br2, br3;
349        wire [7:0] sw;
350        // Instantiate Button and Switch Debouncers
351        debounce DB1(.clock(clk_27mhz),.reset(reset),.noisy(~button_up),.clean(br_up));
352        debounce DB2(.clock(clk_27mhz),.reset(reset),.noisy(~button_down),.clean(br_down));
353        debounce DB3(.clock(clk_27mhz),.reset(reset),.noisy(~button_left),.clean(br_left));
354        debounce DB4(.clock(clk_27mhz),.reset(reset),.noisy(~button_right),.clean(br_right));
355        debounce DB5(.clock(clk_27mhz),.reset(reset),.noisy(~button_enter),.clean(br_enter));
356        debounce DB6(.clock(clk_27mhz),.reset(reset),.noisy(~button0),.clean(br0));
357        debounce DB7(.clock(clk_27mhz),.reset(reset),.noisy(~button1),.clean(br1));
358        debounce DB8(.clock(clk_27mhz),.reset(reset),.noisy(~button2),.clean(br2));
359        debounce DB9(.clock(clk_27mhz),.reset(reset),.noisy(~button3),.clean(br3));
360        debounce DB10(.clock(clk_27mhz),.reset(reset),.noisy(switch[0]),.clean(sw[0]));
361        debounce DB11(.clock(clk_27mhz),.reset(reset),.noisy(switch[1]),.clean(sw[1]));
362        debounce DB12(.clock(clk_27mhz),.reset(reset),.noisy(switch[2]),.clean(sw[2]));
363        debounce DB13(.clock(clk_27mhz),.reset(reset),.noisy(switch[3]),.clean(sw[3]));
364        debounce DB14(.clock(clk_27mhz),.reset(reset),.noisy(switch[4]),.clean(sw[4]));
365        debounce DB15(.clock(clk_27mhz),.reset(reset),.noisy(switch[5]),.clean(sw[5]));
366        debounce DB16(.clock(clk_27mhz),.reset(reset),.noisy(switch[6]),.clean(sw[6]));
```

```
367            debounce DB17(.clock(clk_27mhz),.reset(reset),.noisy(switch[7]),.clean(sw[7]));
368
369            // Instante Button Contention Resolver
370            Button_Contention_Resolver BCR1(
371               .clk(clk_27mhz),
372               .reset(reset),
373               // Button Inputs
374               .button0_in(br0),
375               .button1_in(br1),
376               .button2_in(br2),
377               .button3_in(br3),
378               .button_enter_in(br_enter),
379               .button_left_in(br_left),
380               .button_right_in(br_right),
381               .button_up_in(br_up),
382               .button_down_in(br_down),
383               // Button Outputs
384               .button0_out(b0),
385               .button1_out(b1),
386               .button2_out(b2),
387               .button3_out(b3),
388               .button_enter_out(b_enter),
389               .button_left_out(b_left),
390               .button_right_out(b_right),
391               .button_up_out(b_up),
392               .button_down_out(b_down)
393            );
394
395            // Latch switches to detect transitions
396            reg [7:0] sw_l;
397            always @(posedge clk_27mhz) sw_l <= (reset) ? 0: sw;
398            wire [7:0] sw_rise, sw_fall;
399            assign sw_rise = sw & ~sw_l;
400            assign sw_fall = ~sw & sw_l;
401
402
403            // Latch buttons to detect transitions
404            reg bl_up, bl_down, bl_left, bl_right, bl_enter, bl0, bl1, bl2, bl3;
405            always @(posedge clk_27mhz) begin
406               if (reset) begin
407                  bl_up <= 0;
408                  bl_down <= 0;
409                  bl_left <= 0;
410                  bl_right <= 0;
411                  bl_enter <= 0;
412                  bl0 <= 0;
413                  bl1 <= 0;
414                  bl2 <= 0;
415                  bl3 <= 0;
416               end
417               else begin
418                  bl_up <= b_up;
419                  bl_down <= b_down;
420                  bl_left <= b_left;
421                  bl_right <= b_right;
422                  bl_enter <= b_enter;
423                  bl0 <= b0;
424                  bl1 <= b1;
425                  bl2 <= b2;
426                  bl3 <= b3;
427               end
```

```
428             end
429             assign b_up_rise = b_up & ~bl_up;
430             assign b_up_fall = ~b_up & bl_up;
431             assign b_down_rise = b_down & ~bl_down;
432             assign b_down_fall = ~b_down & bl_down;
433             assign b_left_rise = b_left & ~bl_left;
434             assign b_left_fall = ~b_left & bl_left;
435             assign b_right_rise = b_right & ~bl_right;
436             assign b_right_fall = ~b_right & bl_right;
437             assign b_enter_rise = b_enter & ~bl_enter;
438             assign b_enter_fall = ~b_enter & bl_enter;
439             assign b0_rise = b0 & ~bl0;
440             assign b0_fall = ~b0 & bl0;
441             assign b1_rise = b1 & ~bl1;
442             assign b1_fall = ~b1 & bl1;
443             assign b2_rise = b2 & ~bl2;
444             assign b2_fall = ~b2 & bl2;
445             assign b3_rise = b3 & ~bl3;
446             assign b3_fall = ~b3 & bl3;
447
448
449             // Initiate DT module
450             wire [6:0] year;
451             wire [3:0] month;
452             wire [4:0] day;
453             wire [4:0] hour;
454             wire [5:0] minute;
455             wire [5:0] second;
456             wire [7:0] DT_ascii_out;
457             wire DT_ascii_out_ready;
458             wire [6:0] DT_addr;
459             wire [7:0] DT_data;
460             wire DT_disp_en;
461             reg DT_set;
462             Date_Time DT1(
463                .clk_27mhz(clk_27mhz),
464                .reset(reset),
465                // Control signals
466                .set(DT_set),
467                .disp_en(DT_disp_en),
468                .button_up(b_up),
469                .button_down(b_down),
470                .button_left(b_left),
471                .button_right(b_right),
472                // Date & Time binary outputs
473                .year(year),
474                .month(month),
475                .day(day),
476                .hour(hour),
477                .minute(minute),
478                .second(second),
479                // ASCII output
480                .ascii_out(DT_ascii_out),
481                .ascii_out_ready(DT_ascii_out_ready),
482                // BRAM Binary-to-Decimal Lookup-Table I/O
483                .addr(DT_addr),
484                .data(DT_data)
485                );
486
487             // Instantiate text scroller
488             wire [127:0] string_data;
```

```
489         wire [7:0] ascii_in;
490         wire ascii_in_ready;
491         Text_Scroller TS1(
492            .clk(clk_27mhz),
493            .reset(reset),
494            // Data In
495            .ascii_data(ascii_in),
496            .ascii_data_ready(ascii_in_ready),
497            // Data Out
498            .string_data(string_data)
499            );
500
501         // Instantiate string and hex displays
502         display_string DS1(
503            .clock_27mhz(clk_27mhz),
504            .reset(reset),
505            .string_data(string_data),
506            .disp_blank(disp_blank_0),
507            .disp_clock(disp_clock_0),
508            .disp_rs(disp_rs_0),
509            .disp_ce_b(disp_ce_b_0),
510            .disp_reset_b(disp_reset_b_0),
511            .disp_data_out(disp_data_out_0)
512            );
513
514         wire [63:0] data;
515         display_16hex DISP_HEX_1(
516            .clock_27mhz(clk_27mhz),
517            .reset(reset),
518            .data(data),
519            .disp_blank(disp_blank_1),
520            .disp_clock(disp_clock_1),
521            .disp_rs(disp_rs_1),
522            .disp_ce_b(disp_ce_b_1),
523            .disp_reset_b(disp_reset_b_1),
524            .disp_data_out(disp_data_out_1)
525            );
526
527         // Mux displays as needed
528         reg disp_blank_reg, disp_clock_reg, disp_rs_reg, disp_ce_b_reg, disp_reset_b_reg,
         disp_data_out_reg;
529         assign disp_blank = disp_blank_reg;
530         assign disp_clock = disp_clock_reg;
531         assign disp_rs = disp_rs_reg;
532         assign disp_ce_b = disp_ce_b_reg;
533         assign disp_reset_b = disp_reset_b_reg;
534         assign disp_data_out = disp_data_out_reg;
535         always @(posedge clk_27mhz) begin
536            if (reset) begin
537               disp_blank_reg <= 1'b1;
538               disp_clock_reg <= 1'b0;
539               disp_rs_reg <= 1'b0;
540               disp_ce_b_reg <= 1'b1;
541               disp_reset_b_reg <= 1'b0;
542               disp_data_out_reg <= 1'b0;
543            end
544            else if (sw[0]) begin
545               disp_blank_reg <= disp_blank_0;
546               disp_clock_reg <= disp_clock_0;
547               disp_rs_reg <= disp_rs_0;
548               disp_ce_b_reg <= disp_ce_b_0;
```

```
549                 disp_reset_b_reg <= disp_reset_b_0;
550                 disp_data_out_reg <= disp_data_out_0;
551              end
552              else begin
553                 disp_blank_reg <= disp_blank_1;
554                 disp_clock_reg <= disp_clock_1;
555                 disp_rs_reg <= disp_rs_1;
556                 disp_ce_b_reg <= disp_ce_b_1;
557                 disp_reset_b_reg <= disp_reset_b_1;
558                 disp_data_out_reg <= disp_data_out_1;
559              end
560           end
561
562
563           // Instantiate AC '97 PCM
564           wire [15:0] audio_in_data,audio_out_data;
565           wire audio_ready;
566           AC97_PCM AC97_PCM_1(
567              .clock_27mhz(clk_27mhz),
568              .reset(reset),
569              .volume(4'b1111),
570              // PCM interface signals
571              .audio_in_data(audio_in_data),
572              .audio_out_data(audio_out_data),
573              .ready(audio_ready),
574              // LM4550 interface signals
575              .audio_reset_b(audio_reset_b),
576              .ac97_sdata_out(ac97_sdata_out),
577              .ac97_sdata_in(ac97_sdata_in),
578              .ac97_synch(ac97_synch),
579              .ac97_bit_clock(ac97_bit_clock)
580              );
581
582           // Instantiate Voicemail Interface
583           wire [3:0] sts;
584           wire [3:0] cmd;
585           wire [7:0] phn_num;
586           wire [7:0] vm_ascii_out;
587           wire vm_ascii_out_ready;
588           wire [6:0] vm_addr;
589           wire [7:0] vm_data;
590           wire vm_disp_en;
591           Voicemail_Interface VM_INTERFACE_1(
592              .clk_27mhz(clk_27mhz),      // 27MHz clock
593              .reset(reset),              // Synchronous reset
594              // Main Interface ports
595              .sts(sts),                  // Status port
596              .cmd(cmd),                  // Port for issuing commands
597              .phn_num(phn_num),          // Port for phone number (on writes)
598              .din(audio_in_data),        // Sample Data in
599              .dout(audio_out_data),      // Sample Data out
600              .d_ready(audio_ready),      // Sample Data Ready Signal
601              .disp_en(vm_disp_en),       // Display Enable
602              // Button inputs
603              .button_up(b_up),
604              .button_down(b_down),
605              // ASCII output
606              .ascii_out(vm_ascii_out),                // Port for ASCII data
607              .ascii_out_ready(vm_ascii_out_ready),    // Ready signal for ASCII data
608              // ZBT RAM I/Os
609              .ram_data(ram0_data),
```

```
610             .ram_address(ram0_address),
611             .ram_we_b(ram0_we_b),
612             .ram_bwe_b(ram0_bwe_b),
613             // Date & Time inputs
614             .year(year),
615             .month(month),
616             .day(day),
617             .hour(hour),
618             .minute(minute),
619             .second(second),
620             // Binary-to-Decimal Lookup-Table I/O
621             .addr(vm_addr),
622             .data(vm_data),
623             // SystemACE ports
624             .systemace_data(systemace_data),         // SystemACE R/W data
625             .systemace_address(systemace_address),   // SystemACE R/W address
626             .systemace_ce_b(systemace_ce_b),      // SystemACE chip enable (Active Low)
627             .systemace_we_b(systemace_we_b),         // SystemACE write enable (Active Low)
628             .systemace_oe_b(systemace_oe_b),         // SystemACE output enable (Active Low)
629             .systemace_mpbrdy(systemace_mpbrdy)        // SystemACE buffer ready
630             );
631
632          reg [3:0] cmd_reg;
633          reg [7:0] phn_num_reg;
634
635          assign cmd = cmd_reg;
636          assign phn_num = phn_num_reg;
637
638          // Define command parameters
639          parameter CMD_IDLE       = 4'd0;
640          parameter CMD_START_RD   = 4'd1;
641          parameter CMD_END_RD     = 4'd2;
642          parameter CMD_START_WR   = 4'd3;
643          parameter CMD_END_WR     = 4'd4;
644          parameter CMD_VIEW_UNRD  = 4'd5;
645          parameter CMD_VIEW_SAVED = 4'd6;
646          parameter CMD_DEL        = 4'd7;
647          parameter CMD_SAVE       = 4'd8;
648
649          // Define Status parameters
650          parameter STS_NO_CF_DEVICE = 4'd0;
651          parameter STS_CMD_RDY      = 4'd1;
652          parameter STS_BUSY         = 4'd2;
653          parameter STS_RDING        = 4'd3;
654          parameter STS_RD_FIN       = 4'd4;
655          parameter STS_WRING        = 4'd5;
656          parameter STS_WR_FIN       = 4'd6;
657          parameter STS_ERR_VM_FULL  = 4'd7;
658          parameter STS_ERR_RD_FAIL  = 4'd8;
659          parameter STS_ERR_WR_FAIL  = 4'd9;
660
661          // Binary_to_Decimal Instantiation
662          Binary_to_Decimal BtD1(
663             .clka(clk_27mhz),
664             .clkb(clk_27mhz),
665             .addra(DT_addr),
666             .addrb(vm_addr),
667             .douta(DT_data),
668             .doutb(vm_data)
669             );
670
```

```
671            // Set output to Text Scroller
672            assign ascii_in = vm_ascii_out;
673            assign ascii_in_ready = vm_ascii_out_ready;
674            assign vm_disp_en = 1'b1;
675            assign DT_disp_en = 1'b0;
676
677            // Instantiate state
678            reg [3:0] state;
679
680            // Define state paramters
681            parameter S_IDLE       = 4'h0;
682            parameter S_VIEW_UNRD  = 4'h1;
683            parameter S_VIEW_SAVED = 4'h2;
684            parameter S_START_RD   = 4'h3;
685            parameter S_START_RD_1 = 4'h4;
686            parameter S_START_RD_2 = 4'h5;
687            parameter S_START_WR   = 4'h6;
688            parameter S_START_WR_1 = 4'h7;
689            parameter S_DEL        = 4'h8;
690            parameter S_SAVE       = 4'h9;
691
692            // Set commands with buttons
693            always @(posedge clk_27mhz) begin
694               if (reset) begin
695                  cmd_reg <= CMD_IDLE;
696                  phn_num_reg <= 8'hFF;
697                  state <= S_IDLE;
698               end
699               else begin
700                  case (state)
701                     S_IDLE: begin
702                        cmd_reg <= CMD_IDLE;
703                        if (sts == STS_CMD_RDY) begin
704                           if (b_left_rise)
705                              state <= S_VIEW_UNRD;
706                           else if (b_right_rise)
707                              state <= S_VIEW_SAVED;
708                           else if (b_enter_rise)
709                              state <= S_START_RD;
710                           else if (b3_rise)
711                              state <= S_START_WR;
712                           else if (b0_rise)
713                              state <= S_DEL;
714                           else if (b1_rise)
715                              state <= S_SAVE;
716                        end
717                     end
718                     S_VIEW_UNRD: begin
719                        if (sts == STS_CMD_RDY) begin
720                           cmd_reg <= CMD_VIEW_UNRD;
721                           state <= S_IDLE;
722                        end
723                     end
724                     S_VIEW_SAVED: begin
725                        if (sts == STS_CMD_RDY) begin
726                           cmd_reg <= CMD_VIEW_SAVED;
727                           state <= S_IDLE;
728                        end
729                     end
730                     S_START_RD: begin
731                        if (sts == STS_CMD_RDY) begin
```

```
732                         cmd_reg <= CMD_START_RD;
733                         state <= state + 1;
734                      end
735                   end
736                   S_START_RD_1: begin
737                      cmd_reg <= CMD_IDLE;
738                      state <= state + 1;
739                   end
740                   S_START_RD_2: begin
741                      if (sts == STS_RDING) begin
742                         if (~b_enter) begin
743                            cmd_reg <= CMD_END_RD;
744                            state <= S_IDLE;
745                         end
746                      end
747                      else if ((sts == STS_RD_FIN)|(sts == STS_CMD_RDY))
748                         state <= S_IDLE;
749                   end
750                   S_START_WR: begin
751                      if (sts == STS_CMD_RDY) begin
752                         cmd_reg <= CMD_START_WR;
753                         phn_num_reg <= sw[7:1];
754                         state <= state + 1;
755                      end
756                   end
757                   S_START_WR_1: begin
758                      if (sts == STS_WRING) begin
759                         if (~b3) begin
760                            cmd_reg <= CMD_END_WR;
761                            state <= S_IDLE;
762                         end
763                      end
764                      else if (sts == STS_WR_FIN) begin
765                         state <= S_IDLE;
766                      end
767                   end
768                   S_DEL: begin
769                      if (sts == STS_CMD_RDY) begin
770                         cmd_reg <= CMD_DEL;
771                         state <= S_IDLE;
772                      end
773                   end
774                   S_SAVE: begin
775                      if (sts == STS_CMD_RDY) begin
776                         cmd_reg <= CMD_SAVE;
777                         state <= S_IDLE;
778                      end
779                   end
780                endcase
781             end
782          end
783
784          // Latch LEDs to temporary statuses, with reset button b2
785          reg led3, led4, led5, led6, led7;
786          assign led[7] = ~led7;
787          assign led[6] = ~led6;
788          assign led[5] = ~led5;
789          assign led[4] = ~led4;
790          assign led[3] = ~led3;
791          assign led[2] = ~(sts == STS_RDING);
792          assign led[1] = ~(sts == STS_WRING);
```

```
793         assign led[0] = ~(sts == STS_NO_CF_DEVICE);
794
795         always @(posedge clk_27mhz) begin
796            if (reset | b2) begin
797               led3 <= 0;
798               led4 <= 0;
799               led5 <= 0;
800               led6 <= 0;
801               led7 <= 0;
802            end
803            else begin
804               if (sts == STS_RD_FIN)      led3 <= 1;
805               if (sts == STS_WR_FIN)      led4 <= 1;
806               if (sts == STS_ERR_RD_FAIL) led5 <= 1;
807               if (sts == STS_ERR_WR_FAIL) led6 <= 1;
808               if (sts == STS_ERR_VM_FULL) led7 <= 1;
809            end
810         end
811
812      endmodule
813
```

```
1   ////////////////////////////////////////////////////////////////////////////////
2   // Engineer: Kiarash Adl
3   // Module Name:  Session Module
4   ////////////////////////////////////////////////////////////////////////////////
5
6   /*
7    possible user input:
8    call phone number  5'h1
9    answer call  5'h2
10   disconnect phone number 5'h5
11   voicmail  5'h3
12  */
13
14  module session (input clk, input reset, input [7:0] phoneNum, input [4:0] userInp,
15        input [1:0] cmdIn, input [15:0] packetIn, input transportBusy,
16        output reg [1:0] cmd, output reg [15:0] dataOut, output reg sessionBusy,
17        output reg [7:0] phoneOut, output [3:0] current_state,
18
19        input ac97_clk, input [15:0] micBufferIn,
20        output [15:0] spkBufferOut, output micBufferFull, output micBufferEmpty,
21        output spkBufferFull, output spkBufferEmpty
22        );
23
24
25      reg micBuffer_wr_en, spkBuffer_rd_en;
26
27      reg [3:0] state=0;
28
29      assign current_state = state;
30
31      parameter s_idle=4'd0;
32      parameter s_calling=4'd1;
33      parameter s_connected=4'd2;
34      parameter s_noAnswer=4'd3;
35      parameter s_voicemail=4'd4;
36      parameter s_connectedToVoice=4'd5;
37      parameter s_ringing=4'd6;
38
39      reg [7:0] phone;
40
41      reg spkBuffer_wr_en=0;
42      wire [15:0] spkBufferIn;
43      assign spkBufferIn=packetIn;
44
45      wire [15:0] micBufferOut;
46      reg micBuffer_rd_en;
47
48
49      wire [15:0] spkOut;
50
51      audioBuffer micBuffer ( .din(micBufferIn), .rd_clk(clk),.rd_en(micBuffer_rd_en), .
        rst(reset),
52      .wr_clk(ac97_clk),  .wr_en(micBuffer_wr_en), .dout(micBufferOut), .empty(
```

```
        micBufferEmpty), .full(micBufferFull));
 53
 54
 55     audioBuffer spkBuffer ( .din(spkBufferIn), .rd_clk(ac97_clk),.rd_en(spkBuffer_rd_en
        ), .rst(reset),
 56     .wr_clk(clk),   .wr_en(spkBuffer_wr_en), .dout(spkOut), .empty(spkBufferEmpty), .
        full(spkBufferFull));
 57
 58     assign spkBufferOut= spkBufferEmpty? 16'b0 : spkOut;
 59
 60     always @(posedge clk) begin
 61
 62         if (reset) begin
 63             state<=s_idle;
 64                 spkBuffer_wr_en<=0;
 65                 micBuffer_rd_en<=0;
 66
 67         end else case (state)
 68
 69             s_idle:  begin
 70
 71                 sessionBusy<=0;
 72
 73                 if (userInp==5'h1) begin  //call a phone number
 74                     cmd<=2'b01;       //sending a control packet
 75                     dataOut[15:8]<=phoneNum;
 76                     dataOut[7:0]<=8'h1;
 77                     state<=s_calling;
 78                     phone<=phoneNum;
 79                 end else if (cmdIn==2'b01 && packetIn[7:0]==8'h1) begin //recieving a
                call
 80                     phone<=packetIn[15:8];
 81                     state<=s_ringing;
 82                     phoneOut<=packetIn[15:8];
 83                     cmd<=0;
 84                 end else begin
 85                     state<=s_idle;
 86                     cmd<=0;
 87                 end
 88
 89             end
 90
 91
 92             s_calling: begin
 93                 sessionBusy<=0;
 94                 cmd<=0;
 95
 96                 if ((cmdIn==2'b01) && (packetIn[7:0]==8'h2)) begin  //Answered
 97                     phone<=packetIn[15:8];
 98                     state<=s_connected;
 99                 end else if ((cmdIn==2'b01) && (packetIn[7:0]==8'h3)) begin //Voice-mail
100                     phone<=packetIn[15:8];
101                     state<=s_connectedToVoice;
```

```
102                     end else if ((cmdIn==2'b01) && (packetIn[7:0]==8'h5)) begin //rejected
103                         state<=s_noAnswer;
104                     end else if (userInp==5'h5) begin //user disconnect
105                         cmd<=2'b01;      //sending a control packet
106                         dataOut[15:8]<=phone;
107                         dataOut[7:0]<=8'h5;
108                         state<=s_idle;
109                     end else state<=s_calling;
110 
111                 end
112 
113                 s_ringing: begin
114                     sessionBusy<=0;
115                     phoneOut<=phone; //outputing the caller's number
116                     if ((cmdIn==2'b01) && (packetIn[7:0]==8'h5)) begin //rejected
117                         state<=s_idle;
118                     end else if (userInp==5'h5) begin //reject call
119                         state<=s_idle;
120                         cmd<=2'b01; //send a control packet
121                         dataOut[15:8]<=phone;
122                         dataOut[7:0]<=8'h5;
123                     end else if (userInp==5'h3) begin //voice-mail
124                         cmd<=2'b01;
125                         dataOut[15:8]<=phone;
126                         dataOut[7:0]<=8'h3;
127                         state<=s_voicemail;
128                     end else if (userInp==5'h2) begin   //user answered
129                         cmd<=2'b01;
130                         dataOut[15:8]<=phone;
131                         dataOut[7:0]<=8'h2;
132                         state<=s_connected;
133                     end else begin
134                         state<=s_ringing;
135                     end
136 
137                 end
138 
139                 s_connected: begin
140                     sessionBusy<=0;
141 
142                     if (userInp==5'h5) begin  //user disconnects the call
143                         state<=s_idle;
144                         micBuffer_wr_en<=0;
145                         cmd<=2'b01;
146                         dataOut[7:0]<=8'h5;
147                         dataOut[15:8]<=phone;
148                     end else if ((cmdIn==2'b01) && (packetIn[7:0]==8'h5)) begin //they hung
                        up
149                         micBuffer_wr_en<=0;
150                         state<=s_idle;
151                         cmd<=0;
152                     end else begin
153 
```

```
154                         micBuffer_wr_en<=1;spkBuffer_rd_en<=1;
155
156                         if ((!transportBusy) && (!micBufferEmpty)) begin //sending audio
157                             micBuffer_rd_en<=1;
158                             cmd<=2'b10;
159                             dataOut<=micBufferOut;
160                         end else begin
161                             micBuffer_rd_en<=0;
162                             cmd<=0;
163                         end
164
165                         if (cmdIn==2'b10) begin  //incoming audio
166                             spkBuffer_wr_en<=1;
167                         end else begin
168                             spkBuffer_wr_en<=0;
169                         end
170
171                         state<=s_connected;
172                     end
173
174                 end
175
176                 s_noAnswer: begin    //UI needs this state change
177                     state<=s_idle;
178                 end
179
180                 s_voicemail: begin
181                     sessionBusy<=0;
182
183                     if (userInp==5'h5) begin  //user disconnects the call
184
185                         cmd<=2'b01;
186                         dataOut[7:0]<=8'h5;
187                         dataOut[15:8]<=phone;
188                         state<=s_idle;
189
190                     end else if ((cmdIn==2'b01) && (packetIn[7:0]==8'h5)) begin //they hung
                        up
191                             state<=s_idle;
192                             cmd<=0;
193                     end else if (cmdIn==2'b10) begin  //incoming audio
194                             spkBuffer_wr_en<=1;
195                             state<=s_voicemail;
196                     end else begin
197                             spkBuffer_wr_en<=0;
198                             state<=s_voicemail;
199                     end
200
201                 end
202
203                 s_connectedToVoice: begin
204                     sessionBusy<=0;
205                     if (userInp==5'h5) begin  //user disconnects the call
```

```
206                             micBuffer_wr_en<=0;
207                             cmd<=2'b01;
208                             dataOut[7:0]<=8'h5;
209                             dataOut[15:8]<=phone;
210                             state<=s_idle;
211                         end else if ((cmdIn==2'b01) && (packetIn[7:0]==8'h5)) begin //they hung
                        up
212                             micBuffer_wr_en<=0;
213                             state<=s_idle;
214                             cmd<=0;
215                         end else begin
216
217                             micBuffer_wr_en<=1;spkBuffer_rd_en<=1;
218
219
220                             if ((!transportBusy) && (!micBufferEmpty)) begin //sending audio
221                                 micBuffer_rd_en<=1;
222                                 cmd<=2'b10;
223                                 dataOut<=micBufferOut;
224                             end else begin
225                                 micBuffer_rd_en<=0;
226                                 cmd<=0;
227                             end
228
229                             state<=s_connectedToVoice;
230
231                         end
232
233                     end
234
235                 endcase
236
237             end
238
239         endmodule
240
241
242
243
244
245
246
```

```
1   ////////////////////////////////////////////////////////////////////////////////
2   // Engineer: Kiarash Adl
3   // Module Name:  TransportSend Module
4   ////////////////////////////////////////////////////////////////////////////////
5
6   module transportSend #(parameter packetSize=16) //in bytes
7       (input clk, input reset, input [1:0] cmd, input [15:0] data,
8        input sendData, output reg sending, output [7:0] packetOut,
9        output reg busy, output [10:0] ready_data_count);
10
11      //cmd == 2'b00 idle ; 2'b01  command control data; 2'b10  audio
12      reg goingToSend=0;
13
14      initial begin
15          sending=0;
16          busy=0;
17      end
18
19      //initializing buffer packets' fifo
20      reg [7:0] bufferIn=0;
21      reg buffer_rd_en=0;
22      reg buffer_wr_en=0;
23      wire [10:0] buffer_data_count;
24      wire [7:0] bufferOut;
25      wire bufferEmpty;
26      wire bufferFull;
27      packetBuffer packetBuffer (.clk(clk), .din(bufferIn), .rd_en(buffer_rd_en), .srst(
        reset), .wr_en(buffer_wr_en),
28          .data_count(buffer_data_count), .dout(bufferOut), .empty(bufferEmpty), .full(
            bufferFull));
29
30
31      //initializing ready packets' fifo
32      reg [7:0] readyIn=0;
33      reg ready_rd_en=0;
34      reg ready_wr_en=0;
35      wire [7:0] readyOut;
36      wire readyEmpty;
37      wire readyFull;
38      readyPackets readyPackets (.clk(clk), .din(readyIn), .rd_en(ready_rd_en), .srst(
        reset), .wr_en(ready_wr_en),
39          .data_count(ready_data_count), .dout(readyOut), .empty(readyEmpty), .full(
            readyFull));
40
41
42      assign packetOut=readyOut;
43
44      //reg [7:0] addrBook;
45      //reg [1:0] addrBookTop; //number of phone numbers in the addressbook
46
47      reg [15:0] buffer;
48      reg [packetSize:0] packetSizeCounter=0;
49      reg [packetSize:0] packetSizeCounter2=0;
```

```
50
51      reg [1:0] twoCounter=0;
52
53      reg cdFlag=0;
54      reg auFlag=0;
55
56      always @(posedge clk) begin
57          if (reset) begin
58              cdFlag<=0;
59              auFlag<=0;
60              twoCounter<=0;
61              sending<=0;
62              goingToSend<=0;
63              busy<=0;
64              buffer_wr_en<=0;
65              buffer_rd_en<=0;
66              ready_wr_en<=0;
67              ready_wr_en<=0;
68          end else if(cmd==2'b01 && cdFlag==0) begin  //recieving contol data
69              busy<=1;
70              cdFlag<=1;
71              buffer<=data[15:0];
72              readyIn<=8'b0100_0000;
73              ready_wr_en<=1;
74              twoCounter<=0;
75          end else if (cdFlag==1) begin
76
77              if (twoCounter==2'b00) begin
78                  readyIn<=buffer[15:8];
79                  twoCounter<=2'b01;
80              end else if (twoCounter==2'b01) begin
81                  readyIn<=buffer[7:0];
82                  twoCounter<=2'b10;
83                  packetSizeCounter<=packetSize-3;
84              end else if (twoCounter==2'b10) begin
85                  if (packetSizeCounter==0) begin
86                      busy<=0;
87                      cdFlag<=0;
88                      ready_wr_en<=0;
89                  end else begin
90                      packetSizeCounter<=packetSizeCounter-1;
91                      readyIn<=0;
92                  end
93              end
94
95          end else if (cmd==2'b10 && auFlag==0) begin  // recieving audio
96                  busy<=1;
97                  auFlag<=1;
98                  buffer<=data[15:0];
99                  twoCounter<=0;
100
101         end else if (auFlag==1) begin
102             if (bufferEmpty && twoCounter==0) begin
```

```
103                     if (buffer_wr_en==0) begin
104                         buffer_wr_en<=1;
105                         bufferIn<=8'b1000_0000;
106                     end else buffer_wr_en<=0;
107
108                 end else if (twoCounter==0) begin
109                     buffer_wr_en<=1;
110                     bufferIn<=buffer[15:8];
111                     twoCounter<=2'b01;
112
113                 end else if (twoCounter==2'b01) begin
114                     bufferIn<=buffer[7:0];
115                     twoCounter<=2'b10;
116
117
118                 end else if (!bufferEmpty && twoCounter==2'b11) begin
119                     if (buffer_wr_en==1)
120                         buffer_wr_en<=0;
121                     else begin
122                         readyIn<=bufferOut;
123                         ready_wr_en<=1;
124                     end
125                 end else if (bufferEmpty && twoCounter==2'b11) begin
126                     twoCounter<=2'b10;
127                     readyIn<=bufferOut;
128
129                 end else if (buffer_data_count==packetSize-2) begin
130                     bufferIn<=8'b1111_1111;
131                     twoCounter<=2'b11;
132                     buffer_rd_en<=1;
133                     readyIn<=bufferOut;
134
135                 end else begin
136                     twoCounter<=0;
137                     busy<=0;
138                     auFlag<=0;
139                     buffer_wr_en<=0;
140                     buffer_rd_en<=0;
141                     ready_wr_en<=0;
142                 end
143             end
144
145
146             if (goingToSend==1) begin
147                 if (packetSizeCounter2==1) begin
148                     ready_rd_en<=0;
149                     packetSizeCounter2<=packetSizeCounter2-1;
150                 end else if (packetSizeCounter2==0) begin
151                     sending<=0;
152                     goingToSend<=0;
153                 end else begin
154                     sending<=1;
155                     packetSizeCounter2<=packetSizeCounter2-1;
```

```
156                 end
157             end else if (sendData && (ready_data_count>=packetSize) ) begin
158                 ready_rd_en<=1;
159                 packetSizeCounter2<=packetSize;
160                 goingToSend<=1;
161             end
162 
163         end
164 
165     endmodule
```

```
1    ////////////////////////////////////////////////////////////////////////////////
2    // Engineer: Kiarash Adl
3    // Module Name:  TransportReceive Module
4    ////////////////////////////////////////////////////////////////////////////////
5
6    // set rcvSignal to high and send data from network to here.
7
8    module transportRcv #(parameter packetSize=16)  //in bytes
9        (input clk, input reset, input rcvSignal, input [7:0] packetIn, input sessionBusy,
10       output reg [1:0]  sendingToSession,  output reg [15:0] data,
11       output [10:0] rcv_data_count, output [7:0] dafuq);
12
13
14
15       //initializing recieved packets' FIFO
16       wire [7:0] rcvIn;
17       reg rcv_rd_en=0;
18       wire rcv_wr_en  ;
19       wire [7:0] rcvOut;
20       wire rcvEmpty;
21       wire rcvFull;
22       readyPackets rcvPackets (.clk(clk), .din(rcvIn), .rd_en(rcv_rd_en), .srst(reset), .
         wr_en(rcv_wr_en),
23           .data_count(rcv_data_count), .dout(rcvOut), .empty(rcvEmpty), .full(rcvFull));
24
25       //This module buffers data as they are available.
26       assign rcv_wr_en=rcvSignal;
27       assign rcvIn=packetIn;
28
29       reg [1:0] rcvState=0;
30       reg [packetSize:0] packetSizeCounter;
31       reg [packetSize:0] counter;
32
33       reg [7:0] buffer;
34
35
36       parameter s_idle=0;
37       parameter s_before_sending=1;
38
39       parameter s_sending=2;
40       parameter s_control=3;
41       parameter s_controlTwo=4;
42       parameter s_zeros=5;
43       parameter s_countDown=6;
44       parameter s_audio=7;
45       parameter s_audioTwo=8;
46       parameter s_audioThree=9;
47
48
49       reg [4:0] state=s_idle;
50
51
52       //for debugging purposes
```

```
53        assign dafuq=rcvOut;
54
55
56        initial begin
57            sendingToSession=0;
58            rcvState=0;
59        end
60
61
62        always @(posedge clk) begin
63                if (reset) begin
64                    state<=s_idle;
65
66                end else case (state)
67
68                    s_idle:  begin   //Idle mode
69                        if ((rcv_data_count>=packetSize) && (sessionBusy==0) ) begin
70                            rcv_rd_en<=1;
71                            state<=s_sending;
72                        end else begin
73                            rcv_rd_en<=0;
74                            state<=s_idle;
75                        end
76                    end
77
78                    s_sending: begin   //checking whether the recieved packet has a
                      "control message" or audio data.
79                        if (rcvOut==8'b0100_0000) begin
80
81                            state<=s_control;
82                        end else if (rcvOut==8'b1000_0000) begin
83
84                            state<=s_audio;
85                            packetSizeCounter<=packetSize-2;
86                        end
87
88                    end
89
90                    //we need two clock cycles two receive 16 bits of data
91
92                    s_control: begin
93                        data[15:8]<=rcvOut;
94                        state<=s_controlTwo;
95                    end
96
97                    s_controlTwo: begin
98                        data[7:0]<=rcvOut;
99                        sendingToSession<=2'b01;
100                       state<=s_zeros;
101
102                   end
103
104                   //After the 3rd byte of "control" packets are just zeros; just
```

```
                    discarding.
105                 s_zeros: begin
106                     sendingToSession<=0;
107                     counter<=packetSize-4;
108                     state<=s_countDown;
109                 end
110
111                 s_countDown: begin
112
113                     if (counter==1) begin
114                         state<=s_idle;
115                     end else begin
116                         state<=s_countDown;
117                         counter<=counter-1;
118                     end
119
120                 end
121
122
123                 // every two continous bytes go together.
124                 s_audio: begin
125                     sendingToSession<=0;
126                     if (packetSizeCounter==2) begin
127                         state<=s_audioThree;
128                     end else begin
129                         data[15:8]<=rcvOut;
130                         state<=s_audioTwo;
131                         packetSizeCounter<=packetSizeCounter-1;
132                     end
133
134                 end
135
136                 s_audioTwo: begin
137                     data[7:0]<=rcvOut;
138                     sendingToSession<=2'b10;
139                     state<=s_audio;
140                     packetSizeCounter<=packetSizeCounter-1;
141                 end
142
143                 s_audioThree: begin
144                     rcv_rd_en<=0;
145                     state<=s_idle;
146                 end
147             endcase
148
149
150     end
151
152 endmodule
153
```

```
1   ////////////////////////////////////////////////////////////////////////////////////
2   // Engineer: Kiarash Adl
3   // Module: combinedTransport
4   ////////////////////////////////////////////////////////////////////////////////////
5
6   module combinedTransport #(parameter packetSize=16) //in bytes
7       (input clk, input reset, input [1:0] cmd, input [15:0] data,
8         output [7:0] packetOut, input dummyBufferRd,output busy,
9        output [7:0] phoneNum, output [9:0] dummyBufferCount, output dummyBufferEmpty);
10
11      wire sending;
12      wire [7:0] sendPacketOut;
13      wire [10:0] ready_data_count;
14      wire sendData;
15
16      transportSend sender (
17          .clk(clk),
18          .reset(reset),
19          .cmd(cmd),
20          .data(data),
21          .sendData(sendData),
22          .sending(sending),
23          .packetOut(sendPacketOut),
24          .busy(busy),
25          .ready_data_count(ready_data_count)
26      );
27
28      wire [2:0] debug;
29
30      tranToNet oneTran
31      (.clk(clk), .reset(reset), .data(sendPacketOut), .sending(sending), .dummyBufferRd(
        dummyBufferRd),
32        .sendData(sendData), .phoneNum(phoneNum), .packetOut(packetOut),
33        .dummyBufferCount(dummyBufferCount), .dummyBufferEmpty(dummyBufferEmpty),
34        .debug(debug));
35
36
37  endmodule
38
```

```
1   ////////////////////////////////////////////////////////////////////////////////
2   // Engineer: Kiarash Adl
3   // Module Name:  TranToNet Module -- a helper module to connect TransportSend to Network.
4   ////////////////////////////////////////////////////////////////////////////////
5
6   module tranToNet #(parameter packetSize=16)
7       (input clk, input reset, input [7:0] data, input sending, input dummyBufferRd,
8        output reg sendData, output [7:0] packetOut,
9        output [7:0] phoneNum, output [9:0] dummyBufferCount, output dummyBufferEmpty,
         output [2:0] debug);
10
11      reg [2:0] state=0;
12      assign debug=state;
13
14      reg [7:0] phone=0;
15
16      assign phoneNum=phone;
17
18      parameter s_idle=0;
19      parameter s_receive=1;
20      parameter s_phone=2;
21      parameter s_continue=3;
22
23       wire dummyBufferFull;
24
25       transferFIFO dummyBuffer( .clk(clk),   .din(data), .rd_en(dummyBufferRd),
26         .srst(reset), .wr_en(sending), .data_count(dummyBufferCount),
27          .dout(packetOut),.empty(dummyBufferEmpty),.full(dummyBufferFull));
28
29      always @(posedge clk) begin
30          if (reset) begin
31              state<=s_idle;
32          end
33
34          case(state)
35
36              s_idle: begin
37                  if (dummyBufferEmpty) begin
38                      state<=s_receive;
39                  end else
40                      state<=s_idle;
41              end
42
43              s_receive: begin
44                      sendData<=1;
45                      if (sending) begin
46                            state<=s_phone;
47                      end else state<=s_receive;
48              end
49
50              s_phone: begin
51                  phone<=data;
52                  state<=s_continue;
```

```
53              end
54
55                  s_continue: begin
56                  if (dummyBufferCount==packetSize) begin
57                      sendData<=0;
58                      state<=s_idle;
59                  end else begin
60                          if (!sending)
61                                  state<=s_idle;
62                  end
63                  end
64
65          endcase
66
67      end
68  endmodule
69
```

```
1   ////////////////////////////////////////////////////////////////////////////////
2   // Engineer: Kiarash Adl
3   // Module Name:  CompleteTest Module
4   ////////////////////////////////////////////////////////////////////////////////
5
6   module complete(
7       input clk,  input reset,
8       input [3:0] oneInp,
9       input [3:0] twoInp,
10      output [3:0] onecurrent_state,
11      output [3:0] twocurrent_state
12      );
13
14  //session "one" is connected to transport "sender" and the result is connected to
    transportRcv "recieve"
15  // then there sesult is connected to session "two" ... session "two" is then outputs
    data to transport "s2" and the packets will
16  // be recieved by transportRcv "r2" which outputs the result to session "one".
17
18      wire [4:0] oneuserInp;
19      assign oneuserInp={1'b0,oneInp};
20
21      wire [4:0] twouserInp;
22      assign twouserInp={1'b0, twoInp};
23
24
25      wire [1:0] onecmdIn;
26      wire twotransportBusy;
27      wire [7:0] onephoneNum=8'b1111_1111;
28      wire [7:0] twophoneOut;
29
30      wire onetransportBusy;
31
32      wire onemicFlag;
33      wire [1:0] onecmd;
34      wire [15:0] onedataOut;
35      wire onesessionBusy;
36      wire [7:0] onephoneOut;
37
38
39      wire [15:0] onespkBufferOut;
40      wire onemicBufferFull;
41      wire onemicBufferEmpty;
42      wire onespkBufferFull;
43      wire onespkBufferEmpty;
44      wire [15:0] onemicBufferOut;
45
46      wire [15:0] onepacketInp;
47
48
49      session one (
50          .clk(clk),
51          .reset(reset),
```

```
52            .phoneNum(onephoneNum),
53            .userInp(oneuserInp),
54            .cmdIn(onecmdIn),
55            .packetIn(onepacketInp),
56            .transportBusy(onetransportBusy),
57            .cmd(onecmd),
58            .dataOut(onedataOut),
59            .sessionBusy(onesessionBusy),
60            .phoneOut(onephoneOut),
61            .current_state(onecurrent_state),
62            .ac97_clk(clk),
63            .micBufferIn(16'b0),
64            .spkBufferOut(onemicBufferOut),
65            .micBufferFull(onemicBufferFull),
66            .micBufferEmpty(onemicBufferEmpty),
67            .spkBufferFull(onespkBufferFull),
68            .spkBufferEmpty(onespkBufferEmpty)
69        );
70
71        wire sending;
72        wire [7:0] sendPacketOut;
73        wire [10:0] senderCounter;
74
75        transportSend sender (
76            .clk(clk),
77            .reset(reset),
78            .cmd(onecmd),
79            .data(onedataOut),
80            .sendData(1'b1),
81            .sending(sending),
82            .packetOut(sendPacketOut),
83            .busy(onetransportBusy),
84            .ready_data_count(senderCounter)
85        );
86
87        wire sessionBusy;
88
89        wire [1:0] sendingToSession;
90        wire [15:0] sessionData;
91        wire [7:0] dafuq;
92        wire [10:0] rcvCounter;
93
94        transportRcv receive (
95            .clk(clk),
96            .reset(reset),
97            .rcvSignal(sending),
98            .packetIn(sendPacketOut),
99            .sessionBusy(sessionBusy),
100           .sendingToSession(sendingToSession),
101           .data(sessionData),
102           .rcv_data_count(rcvCounter),
103           .dafuq(dafuq)
104       );
```

```
105
106
107         wire twomicFlag;
108         wire [15:0] twoaudioOut;
109         wire [1:0] twocmd;
110         wire [15:0] twodataOut;
111         wire [7:0] twophoneNum;
112
113
114         wire [15:0] twospkBufferOut;
115         wire twomicBufferFull;
116         wire twomicBufferEmpty;
117         wire twospkBufferFull;
118         wire twospkBufferEmpty;
119         wire [15:0] twomicBufferOut;
120
121         session two (
122             .clk(clk),
123             .reset(reset),
124             .phoneNum(twophoneNum),
125             .userInp(twouserInp),
126             .cmdIn(sendingToSession),
127             .packetIn(sessionData),
128             .transportBusy(twotransportBusy),
129             .cmd(twocmd),
130             .dataOut(twodataOut),
131             .sessionBusy(sessionBusy),
132             .phoneOut(twophoneOut),
133             .current_state(twocurrent_state),
134             .ac97_clk(clk),
135             .micBufferIn(16'b0),
136             .spkBufferOut(twomicBufferOut),
137             .micBufferFull(twomicBufferFull),
138             .micBufferEmpty(twomicBufferEmpty),
139             .spkBufferFull(twospkBufferFull),
140             .spkBufferEmpty(twospkBufferEmpty)
141         );
142
143         wire s2sending;
144         wire [7:0] s2sendPacketOut;
145         wire [10:0] s2senderCounter;
146
147         transportSend s2 (
148             .clk(clk),
149             .reset(reset),
150             .cmd(twocmd),
151             .data(twodataOut),
152             .sendData(1'b1),
153             .sending(s2sending),
154             .packetOut(s2sendPacketOut),
155             .busy(twotransportBusy),
156             .ready_data_count(s2senderCounter)
157         );
```

```
158
159
160     wire [7:0] r2dafuq;
161     wire [10:0] r2rcvCounter;
162
163     transportRcv r2 (
164         .clk(clk),
165         .reset(reset),
166         .rcvSignal(s2sending),
167         .packetIn(s2sendPacketOut),
168         .sessionBusy(onesessionBusy),
169         .sendingToSession(onecmdIn),
170         .data(onepacketInp),
171         .rcv_data_count(r2rcvCounter),
172         .dafuq(r2dafuq)
173     );
174
175
176 endmodule
177
```

```
1   ////////////////////////////////////////////////////////////////////////////////
2   // Engineer: Kiarash Adl
3   // Module Name:  Session Module Test Bench
4   ////////////////////////////////////////////////////////////////////////////////
5
6   module session_tb;
7
8       // Inputs
9       reg clk;
10      reg reset;
11      reg [7:0] phoneNum;
12      reg [4:0] userInp;
13      reg [1:0] cmdIn;
14      reg [15:0] packetIn;
15      reg transportBusy;
16      reg ac97_clk;
17      reg micBuffer_wr_en;
18      reg [15:0] micBufferIn;
19      reg spkBuffer_rd_en;
20
21      // Outputs
22      wire micFlag;
23      wire [1:0] cmd;
24      wire [15:0] dataOut;
25      wire sessionBusy;
26      wire [7:0] phoneOut;
27      wire [3:0] current_state;
28      wire [15:0] spkBufferOut;
29      wire micBufferFull;
30      wire micBufferEmpty;
31      wire spkBufferFull;
32      wire spkBufferEmpty;
33
34      // Instantiate the Unit Under Test (UUT)
35      session uut (
36          .clk(clk),
37          .reset(reset),
38          .phoneNum(phoneNum),
39          .userInp(userInp),
40          .cmdIn(cmdIn),
41          .packetIn(packetIn),
42          .transportBusy(transportBusy),
43          .micFlag(micFlag),
44          .cmd(cmd),
45          .dataOut(dataOut),
46          .sessionBusy(sessionBusy),
47          .phoneOut(phoneOut),
48          .current_state(current_state),
49          .ac97_clk(ac97_clk),
50          .micBuffer_wr_en(micBuffer_wr_en),
51          .micBufferIn(micBufferIn),
52          .spkBuffer_rd_en(spkBuffer_rd_en),
53          .spkBufferOut(spkBufferOut),
```

```
54            .micBufferFull(micBufferFull),
55            .micBufferEmpty(micBufferEmpty),
56            .spkBufferFull(spkBufferFull),
57            .spkBufferEmpty(spkBufferEmpty)
58        );
59
60        always #5 clk=!clk;
61
62        initial begin
63            // Initialize Inputs
64            clk = 0;
65            reset = 0;
66            phoneNum = 0;
67            userInp = 0;
68            cmdIn = 0;
69            packetIn = 0;
70            transportBusy = 0;
71            ac97_clk = 0;
72            micBuffer_wr_en = 0;
73            micBufferIn = 0;
74            spkBuffer_rd_en = 0;
75
76            // Wait 100 ns for global reset to finish
77            #100;
78
79            // Add stimulus here
80            phoneNum=8'h20;
81            userInp=5'h01;
82            #100;
83            phoneNum=0;
84            userInp=0;
85            #100;
86            cmdIn=2'b01;
87            packetIn[7:0]=8'h5;
88            packetIn[15:8]=8'h30;
89            #100;
90            cmdIn=2'b01;
91            packetIn[7:0]=8'h1;
92            packetIn[15:8]=8'h30;
93            #100;
94
95        end
96
97    endmodule
98
99
```

```
1   ////////////////////////////////////////////////////////////////////////////////
2   // Engineer: Kiarash Adl
3   // Module Name:  TransportSend Test Bench
4   ////////////////////////////////////////////////////////////////////////////////
5   
6   module tSend_tb;
7   
8       // Inputs
9       reg clk;
10      reg reset;
11      reg [1:0] cmd;
12      reg [15:0] data;
13      reg sendData;
14  
15      // Outputs
16      wire sending;
17      wire [7:0] packetOut;
18      wire busy;
19      wire ready_data_count;
20  
21      // Instantiate the Unit Under Test (UUT)
22      transportSend uut (
23          .clk(clk),
24          .reset(reset),
25          .cmd(cmd),
26          .data(data),
27          .sendData(sendData),
28          .sending(sending),
29          .packetOut(packetOut),
30          .busy(busy),
31        .ready_data_count(ready_data_count)
32  
33      );
34  
35      always #5 clk= !clk;
36  
37      initial begin
38          // Initialize Inputs
39          clk = 0;
40          reset = 0;
41          cmd = 0;
42          data = 0;
43          sendData = 0;
44  
45          // Wait 100 ns for global reset to finish
46          #100;
47  
48          // Add stimulus here
49      /*  cmd=2'b01;
50          #20;
51          cmd=0;
52          #200;
53          cmd=2'b10;
```

```
54              #400;
55              cmd=0;
56              #400;
57              sendData=1;
58              #20;
59              sendData=0;
60              #200;
61              cmd=2'b10;
62              #200;
63              sendData=1;
64              #20;
65              sendData=0;
66              #500;
67          */
68              cmd=2'b01;
69              data=16'h44;
70              #100;
71              sendData=1;
72
73
74          end
75
76      endmodule
77
78
```

```
1   ////////////////////////////////////////////////////////////////////////////////
2   // Engineer: Kiarash Adl
3   // Module Name:  TransportReceive Test Bench
4   ////////////////////////////////////////////////////////////////////////////////
5
6
7   module tRcv_tb;
8
9       // Inputs
10      reg clk;
11      reg reset;
12      reg rcvSignal;
13      reg [7:0] packetIn;
14      reg sessionBusy;
15
16      // Outputs
17      wire [1:0] sendingToSession;
18      wire [15:0] data;
19      wire dafuq;
20
21      // Instantiate the Unit Under Test (UUT)
22      transportRcv uut (
23          .clk(clk),
24          .reset(reset),
25          .rcvSignal(rcvSignal),
26          .packetIn(packetIn),
27          .sessionBusy(sessionBusy),
28          .sendingToSession(sendingToSession),
29          .data(data),
30          .dafuq(dafuq)
31      );
32
33
34      always #5 clk= !clk;
35  //  always #20 packetIn=packetIn+2;
36
37      initial begin
38          // Initialize Inputs
39          clk = 0;
40          reset = 0;
41          rcvSignal = 0;
42          packetIn = 0;
43          sessionBusy = 1;
44
45          // Wait 100 ns for global reset to finish
46          #100;
47
48          // Add stimulus here
49          rcvSignal=1;
50          packetIn=8'b1000_0000;
51          #10;
52          packetIn=8'h4;
53          #10;
```

```
54          packetIn=8'h5;
55          #10;
56          packetIn=8'h6;
57          #10;
58          packetIn=8'h7;
59          #10;
60          packetIn=8'h8;
61          #10;
62          packetIn=8'h9;
63          #10;
64          packetIn=8'h10;
65          #10;
66          packetIn=8'h11;
67          #10;
68          packetIn=8'h12;
69          #10;
70          packetIn=8'h13;
71          #10;
72          packetIn=8'h14;
73          #10;
74          packetIn=8'h15;
75          #10;
76          packetIn=8'h16;
77          #10;
78          packetIn=8'h17;
79          #10;
80          packetIn=8'b1111_1111;
81          #10;
82          rcvSignal=0;
83          #200;
84          rcvSignal=1;
85
86
87          packetIn=8'b0100_0000;
88          #10;
89          packetIn=8'h2;
90          #20;
91          packetIn=2;
92          #130;
93          rcvSignal=0;
94          #100;
95          sessionBusy=0;
96          #500;
97          reset=1;
98          #50;
99          reset=0;
100
101     end
102
103 endmodule
104
105
```

```
1   ////////////////////////////////////////////////////////////////////////////////
2   // Engineer: Kiarash Adl
3   // Module: combinedTransport test bench
4   ////////////////////////////////////////////////////////////////////////////////
5
6   module combinedTransport_tb;
7
8       // Inputs
9       reg clk;
10      reg reset;
11      reg [1:0] cmd;
12      reg [15:0] data;
13      reg dummyBufferRd;
14
15      // Outputs
16      wire [7:0] packetOut;
17      wire [7:0] phoneNum;
18      wire [9:0] dummyBufferCount;
19      wire dummyBufferEmpty;
20      wire busy;
21
22      // Instantiate the Unit Under Test (UUT)
23      combinedTransport uut (
24          .clk(clk),
25          .reset(reset),
26          .cmd(cmd),
27          .data(data),
28          .packetOut(packetOut),
29          .dummyBufferRd(dummyBufferRd),
30          .busy(busy),
31          .phoneNum(phoneNum),
32          .dummyBufferCount(dummyBufferCount),
33          .dummyBufferEmpty(dummyBufferEmpty)
34      );
35
36      always #5 clk= !clk;
37
38      initial begin
39          // Initialize Inputs
40          clk = 0;
41          reset = 0;
42          cmd = 0;
43          data = 0;
44          dummyBufferRd=0;
45
46          // Wait 100 ns for global reset to finish
47          #100;
48
49          // Add stimulus here
50          cmd=2'b01;
51          data=16'b1010_0011_1111_0001;
52          #50;
53          cmd=0;
```

```
54          #400;
55          dummyBufferRd=1;
56          cmd=2'b10;
57          data=16'b1010_0011_1111_0001;
58
59
60      end
61
62  endmodule
63
64
```

```
1   ////////////////////////////////////////////////////////////////////////////////
2   // Engineer: Kiarash Adl
3   // Module Name:  CompleteTest  test bench
4   ////////////////////////////////////////////////////////////////////////////////
5
6   module complete_tb;
7
8       // Inputs
9       reg clk;
10      reg reset;
11      reg [3:0] oneInp;
12      reg [3:0] twoInp;
13
14      // Outputs
15      wire [3:0] onecurrent_state;
16      wire [3:0] twocurrent_state;
17
18      // Instantiate the Unit Under Test (UUT)
19      complete uut (
20          .clk(clk),
21          .reset(reset),
22          .oneInp(oneInp),
23          .twoInp(twoInp),
24          .onecurrent_state(onecurrent_state),
25          .twocurrent_state(twocurrent_state)
26      );
27
28      always #5 clk=!clk;
29
30      initial begin
31          // Initialize Inputs
32          clk = 0;
33          reset = 0;
34          oneInp = 0;
35          twoInp = 0;
36
37          // Wait 100 ns for global reset to finish
38          #100;
39
40          // Add stimulus here
41          oneInp=4'h1;
42          #500;
43          twoInp=4'h5;
44
45
46
47      end
48
49  endmodule
50
51
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Nandi Bugg
//
// Create Date:    12:49:56 11/15/2012
// Design Name:
// Module Name:    user_interface
// Project Name:
// Target Devices:
// Tool versions:
// Description:
// 1. Responsible for displaying menu items on screen. Muxes between the time module,
// voicemail module, and itself to do so.
// 2. Chooses proper audio from session layer and voicemail module
// 3. Sends and receives commands from application layer to achieve goals
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module user_interface(
    input clk,
    input s7,
    input s6,
    input s5,
    input s4,
    input s3,
    input s2,
    input s1,
	 input s0,
    input b3,
    input b2,
    input b1,
    input b0,
    input reset,
    input enter,
    input up,
    input down,
    input left,
    input right,
	 input [3:0] inc_command,
```

```
        input init,
        input [7:0] inc_address,
        output [15:0] audio_out_data,
        input ready,
        input [3:0] voicemail_status,
        input [15:0] dout,
        output [3:0] voicemail_command,
        output [7:0] phn_num,
        output [15:0] din,
        output [1:0] disp_control, //who has control over display
    output [7:0] address,
    output [4:0] command,
    output [2:0] current_state,
        output [5:0] current_menu_item,
        output [4:0] headphone_volume,
        input [15:0] audio_in_data,
        output [11:0] txt_addr,
        output [11:0] txt_length,
        output txt_start,
        input done,
        output set_date
    );


///////////////////////////////////////////////////////
//States
//////////////////////////////////////////////////////
        parameter [2:0] idle=0; //no current calls
        parameter [2:0] incoming=3'd1; //user is being called
        parameter [2:0] outgoing=3'd2; //user calls another party
        parameter [2:0] busy=3'd3; //call in progress
        parameter [2:0] call_while_busy=3'd4; //user in call, receiving incoming call
        parameter [2:0] initialize=3'd5; //initialize node
        parameter [2:0] in_voicemail=3'd6;
/////////////////////////////////////////////////////////




/////////////////////////////////////////////////////////
//Overall parameters
/////////////////////////////////////////////////////////
```

```
	parameter conference=0; //conference call on/off
	parameter selective=0; //selective mode switch for call forwarding
	parameter block_state=0; //0=off,1=on call blocking
	reg voicemail_state=0; //0=off,1=on
	reg override=1; //used for transferring control over up/down buttons from UI to voicemail
///////////////////////////////////////////////////////



//////////////////////////////////////////////////////////
//Menu Items
//////////////////////////////////////////////////////////
//	//call forward mode states
//	parameter [1:0] all=0;
//	parameter [1:0] busy_f=3'd1;
//	parameter [1:0] no_answer=3'd2;
//	parameter [1:0] selective=3'd3;
//
//
//	parameter [1:0] fwd_mode_state = 3'd3; //call forward mode, default is selective
//	parameter [1:0] temp_fwd_mode;
//	parameter fwd_state = 0; //toggle forwarding on/off
//
	////////////////////////////////////
	//Main menu parameters
	/////////////////////////////////////
	parameter call_number=0;//start a phone call
	parameter volume=6'd1; //set headphone volume
	parameter voicemail=6'd2;//voicemail options
	// parameter call_block=6'd5;//call block options
	// parameter call_fwd=6'd6;//call forwarding options
	parameter get_num=6'd3;//display FPGA's number
	parameter set_time=6'd4;//set system date and time

	////////////////////////////////////////
	//Call number
	///////////////////////////////////////
	parameter dialing=6'd7;

	/////////////////////////////////////
	//Volume
	////////////////////////////////////
	parameter change_vol=6'd8;
```

```
	////////////////////////////////////////
	//Voicemail menu
	////////////////////////////////////////
	parameter toggle_v=6'd9; //toggle voicemail on/off
	parameter unread=6'd10; //unread voicemail
	parameter saved=6'd11; //saved voicemail

	////////////////////////////////////////
	//Unread voicemail menu
	///////////////////////////////////////
	parameter play_unread=6'd12; //play unread voicemail
	parameter del_unread=6'd13; //delete unread voicemail
	parameter del_all_unread=6'd14; //delete all unread voicemail

	///////////////////////////////////////
	//Saved voicemail menu
	/////////////////////////////////////////
	parameter play_saved=6'd15; //play saved voicemail
	parameter del_saved=6'd16; //delete saved voicemail
	parameter del_all_saved=6'd17; //delete all saved voicemail

//	//call blocking menu
//	parameter  toggle_block=6'd18; //toggle call blocking on/off
//	parameter  view_blocked=6'd19; //view blocked numbers
//	parameter  add_blocked=6'd20; //add blocked number
//	parameter  del_blocked=6'd21; //delete a blocked number
//	parameter  del_all_blocked=6'd22; //delete all blocked numbers
//
//	//call forward menu
//	parameter  toggle_fwd=6'd23; //toggle call forwarding on/off
//	parameter  fwd_mode=6'd24; //set forwarding mode
//	parameter  set_fwd_number=6'd25; //set forwarding number
//	parameter  view_sel_num=6'd26;	//view selective numbers
//	parameter  add_sel_num=6'd27;	//add selective numbers
//	parameter  del_sel_num=6'd28;	//delete selective numbers
//	parameter  del_all_sel_num=6'd29; //delete all selective numbers

//	parameter [7:0] fwd_address; //forward address added

	/////////////////////////////////////////////
	//Display user's number
	////////////////////////////////////////////
	parameter  disp_num = 6'd30;
```

```
/////////////////////////////////////////////
//Set system date and time
////////////////////////////////////////////
parameter  set_dt=6'd31;

////////////////////////////////////////////
//Incoming call menu
/////////////////////////////////////////////
parameter  def_incoming=6'd32;
parameter  accept=6'd33;
parameter  reject=6'd34;
parameter  send_to_v=6'd35; //send to voicemail

////////////////////////////////////////////
//Outgoing call menu
////////////////////////////////////////////
parameter  def_outgoing=6'd36;
parameter  end_call=6'd37;

//////////////////////////////////////////////
//Busy (call-in-progress) menu
//////////////////////////////////////////////
parameter  def_busy=6'd38;
parameter  end_call_b=6'd39;
parameter  set_volume=6'd40;
//          parameter  called_IDs=6'd41;
//          parameter  xfer_call=6'd42;
//          parameter  hold_call=6'd43;
//          parameter  resume_call=6'd44;
//          parameter  conf_call=6'd45;

//////////////////////////////////////////////
//Incoming while busy
/////////////////////////////////////////////
parameter  reject_call=6'd46;
parameter  send_2_v=6'd47; //send to voicemail
parameter  end_call_inc=6'd48;
parameter  hold_curr=6'd49; //hold current call

//////////////////////////////////////////////
//Default displays
/////////////////////////////////////////////
parameter  def_welcome=6'd50; //welcome for idle state
```

```
	parameter  def_init=6'd51;
	parameter  def_inc_busy=6'd52; //default display for incoming-while-busy state
	parameter  def_sys=6'd53; //sys date and time for idle state
////////////////////////////////////////////////////////////////////////




///////////////////////////////////////////////////////
//UI=>application layer commands
/////////////////////////////////////////////////////
	parameter init_signal=0; //user wants to initialize system
	parameter make_call=5'h1;	//user trying to make phone call
	parameter answer_call=5'h2; //user accepts call
	parameter go_to_voicemail=3'h3; //user wants to send call to voicemail
	parameter disconnect=5'h5; //user wants to end call
//////////////////////////////////////////////////////


///////////////////////////////////////////////////////
//Application layer=>UI commands
///////////////////////////////////////////////////////
   parameter disconnected=4'd0; //no current call
	parameter outgoing_call=4'd1; //outgoing call
	parameter connected=4'd2; //call went through successfully
	parameter no_answer=4'd3; //pickup didn't occur during 30s and voicemail is off
	parameter sent_to_v=4'd4; //incoming call sent to voicemail
	parameter connected_to_v=4'd5; //outgoing call in voicemail
	parameter incoming_call=4'd6; //incoming call
//////////////////////////////////////////////////////////


/////////////////////////////////////////////////////////
//Temp variables for outputs + initialization
///////////////////////////////////////////////////////
	reg [2:0] state=initialize;
	reg [2:0] c_state=initialize;

	reg [7:0] temp_addr;
	reg [7:0] temp_command;


	reg [5:0] menu_item=def_init;
	reg [5:0] menu_item_latch;
```

```
	reg [3:0] temp_voicemail_command;

	reg temp_set_date;

	//parameters for display control
	parameter UI=0;
	parameter date_time=2'd1;
	parameter voicemail_disp=2'd2;
	reg [1:0] temp_display_control=UI;

////////////////////////////////////////////////////



///////////////////////////////////////////////////////////
// I/O for text_scroller_interface
///////////////////////////////////////////////////////////
	reg start;
	reg [10:0] addr;
	reg [10:0] length;
//	wire [7:0] ascii_out;
//	wire ascii_out_ready;
//	wire done;
//////////////////////////////////////////////////////////////
//Voicemail commands/statuses
/////////////////////////////////////

//commands
parameter CMD_IDLE       = 4'd0;
parameter CMD_START_RD   = 4'd1;
parameter CMD_END_RD     = 4'd2;
parameter CMD_START_WR   = 4'd3;
parameter CMD_END_WR     = 4'd4;
parameter CMD_VIEW_UNRD  = 4'd5;
parameter CMD_VIEW_SAVED = 4'd6;
parameter CMD_DEL        = 4'd7;
parameter CMD_SAVE       = 4'd8;

//statuses
parameter STS_NO_CF_DEVICE = 4'd0;
parameter STS_CMD_RDY      = 4'd1;
parameter STS_BUSY         = 4'd2;
parameter STS_RDING        = 4'd3;
```

```
parameter STS_RD_FIN       = 4'd4;
parameter STS_WRING        = 4'd5;
parameter STS_WR_FIN       = 4'd6;
parameter STS_ERR_VM_FULL  = 4'd7;
parameter STS_ERR_RD_FAIL  = 4'd8;
parameter STS_ERR_WR_FAIL  = 4'd9;

////////////////////////////////////////////////////////////
//Audio
////////////////////////////////////////////////////////////
reg [4:0] temp_headphone_volume=5'd16; //default volume
reg [4:0] headphone_change; //amount user has changed headphone volume by


//Audio Mux (switches between voicemail and call audio)
//AC97_PCM ac(.clock_27mhz(clk),.reset(reset),.volume(headphone_volume),
//.audio_in_data(audio_in_data),.audio_out_data(audio_out_data),
//.ready(ready),
//.audio_reset_b(audio_reset_b),.ac97_sdata_out(ac97_sdata_out),
//.ac97_sdata_in(ac97_sdata_in),
//.ac97_synch(ac97_synch),.ac97_bit_clock(ac97_bit_clock));

assign audio_out_data=(menu_item==play_unread||menu_item==play_saved
||inc_command==sent_to_v)?(dout):(audio_in_data);




///////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////
//Timer for Incoming Calls
///////////////////////////////////////////////////////////
//I/Os
wire enable;
wire start_timer;
reg start_t;
wire expired;
wire countdown;
wire [4:0] inc_timer=5'd30;


 //Divider
Divider#(.N(27000000),.W(25)) div(.clk(clk),.reset(reset),.sys_reset(reset),
```

```
                                            .new_clk(enable));

 //Timer
timer tim(.start_timer(start_timer),.sys_reset(sys_reset),.clk(clk),
              .enable(enable),.clk_value(inc_timer),.expired(expired),.countdown(countdown));



//////////////////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////////////////
//Latches for Buttons
////////////////////////////////////////////////////////////////////////
reg up_latch;
reg down_latch;
reg right_latch;
reg left_latch;
reg b0_latch;
reg b1_latch;
reg b2_latch;
reg b3_latch;
reg enter_latch;
reg reset_latch;
///////////////////////////////////////////////////////////////////////


//set display text here
          always @(posedge clk) begin
                    //deal with latches here
                    up_latch<=up;
     down_latch<=down;
     right_latch<=right;
                    left_latch<=left;
                    b0_latch<=b0;
                    b1_latch<=b1;
                    b2_latch<=b2;
                    b3_latch<=b3;
                    enter_latch<=enter;
     reset_latch<=reset;

                    menu_item_latch <= menu_item;

                    start<=menu_item_latch!=menu_item;
```

```
		if (menu_item_latch!=menu_item) begin
			case (menu_item) //select address for start of text, set length as
well
		//initialization state
				def_init: begin
					addr<=0; //text=Press "Enter" to start
network initialization

					length<=11'd45;
				end



		//idle state
				//welcome
				def_welcome:begin
					addr<=11'd45; //text=Welcome
					length<=11'd7;
				end

				call_number:begin
					addr<=11'd52;//text=Call Number
					length<=11'd11;
				end
				volume: begin
					addr<=11'd63;//91 //text=Set Headphone
Volume

					length<=11'd20;
				end
				voicemail: begin
					addr<=11'd83; //text=Voicemail
					length<=11'd9;
				end
				get_num: begin
					addr<=11'd92; //text=Display System
Number

					length<=11'd21;
				end
				set_time:begin
					addr<=11'd113; //text=Set Date & Time
					length<=11'd15;
				end

				//call number
```

```
dialing: begin
        //convert switches somehow, display in hex
or binary?
end

//volume
change_vol: begin
        //current volume
end

//voicemail
toggle_v:begin
        if (voicemail_state) begin //voicemail
currently on
                addr<=11'd142; //text=Turn
Voicemail Off

                length<=11'd18;
        end

        else begin //voicemail currently off
                addr<=11'd160;//text=Turn
Voicemail On

                length<=11'd17;
        end
end

unread: begin  //text=Unread Voicemail
        addr<=11'd177;
        length<=11'd16;
end

saved: begin   //text=Saved Voicemail
        addr<=11'd268;
        length<=11'd15;
end

//get number
disp_num:begin
        //disp. in hex or binary?
end

//incoming state
accept:begin //text=Accept Incoming Call
        addr<=11'd374;
```

```
                        length<=11'd20;
                    end
                    reject:begin     //text=Reject Incoming Call
                        addr<=11'd395;
                        length<=11'd20;
                    end
                    send_to_v:begin         //text=Send to Voicemail
                        addr<=11'd416;
                        length<=11'd17;
                    end
                    def_incoming:begin    //text=Incoming Call
                        addr<=11'd360;
                        length<=11'd13;
                    end

                //outgoing state
                def_outgoing:begin //text=Calling
                    addr<=11'd434;
                    length<=11'd7;
                end
                end_call:begin//text=End Call
                    addr<=11'd442;
                    length<=11'd8;
                end


                //busy state
                def_busy:begin //text=Current Call
                    addr<=11'd489;
                    length<=11'd12;
                end
                end_call_b:begin //text=End Call
                    addr<=11'd442;
                    length<=11'd8;
                end
            endcase
        end

end


always @(posedge clk) begin
        c_state<=state;
```

```
                if (reset&&!reset_latch) begin
                        menu_item<=def_init;
                        state<=initialize;
                        voicemail_state<=0;
                end

                else if (voicemail_command!=0)
                        temp_voicemail_command<=CMD_IDLE;

                else if (done) begin
                        case (state)
                                //initialize state logic
                                initialize: begin               //text=Press "Enter" to start network
initialization
                                                if (enter&&!enter_latch) begin
                                                        menu_item<=def_welcome;
                                                        state<=idle;
                                                end
                                        end





                                // idle state logic
                                idle: begin
                                        if (inc_command==incoming_call) begin
                                                menu_item<=def_incoming;
                                                start_t<=1;
                                                state<=incoming;
                                        end

                                        else if (up&&override&&!up_latch) begin //up button
pressed
                                                case (menu_item)
                                                        call_number: menu_item<=set_time;//
main_menu
                                                        toggle_v: menu_item<=saved; //voicemail
                                                        play_unread: menu_item<=play_unread; //
unread voicemail
                                                        play_saved: menu_item<=play_saved; //
saved voicemail
```

```
                                        accept: menu_item<=send_to_v; //incoming
call menu
                                        end_call: menu_item<=set_volume; //busy
                                        change_vol: begin
                                                if (headphone_volume<31)
        headphone_change<=headphone_change+1;
                                        end
                                        get_num:begin
                                                if
(voicemail_status==STS_NO_CF_DEVICE) //no CF device found so hide voicemail menu
                                                        menu_item<=volume;
                                                else
                                                        menu_item<=voicemail;
                                        end
                                        def_sys: menu_item<=def_sys;
                                        def_init:menu_item<=def_init;
                                        def_welcome:menu_item<=def_welcome;
                                        set_dt:menu_item<=set_dt;
                                        default: menu_item<=menu_item-1;
                                endcase

                        end

                        else if (down&&override&&!down_latch) begin //down
button pressed
                                case (menu_item)
                                        set_time: menu_item<=call_number;//
main_menu
                                        saved: menu_item<=toggle_v; //voicemail
                                        play_unread: menu_item<=play_unread;//
unread voicemail
                                        play_saved: menu_item<=play_saved; //
saved voicemail
                                        accept: menu_item<=send_to_v; //incoming
call menu
                                        set_volume: menu_item<=end_call; //busy
                                        change_vol: begin
                                                if (headphone_volume>0)
        headphone_change<=headphone_change-1;
                                        end
                                        volume:begin
                                                if
(voicemail_status==STS_NO_CF_DEVICE) //no CF device found so hide voicemail menu
                                                        menu_item<=get_num;
```

```
								else
									menu_item<=voicemail;
							end
							def_sys: menu_item<=def_sys;
							def_init:menu_item<=def_init;
							def_welcome:menu_item<=def_welcome;
							set_dt:menu_item<=set_dt;
							default: menu_item<=menu_item+1;
						endcase
					end

					else if (((right&&!right_latch)||(enter&&!enter_latch))
&&override) begin //menu item selected
						case (menu_item)
							def_welcome: menu_item<=call_number;
							def_sys: begin
								temp_display_control<=UI;
								menu_item<=call_number;
							end
							call_number: menu_item<=dialing;
							volume: menu_item<=change_vol;
							voicemail: menu_item<=toggle_v;
//							get_num: menu_item<=set_time;
//							set_time: menu_item<=call_number;
							def_init:menu_item<=def_welcome;

							//call number
							dialing: begin
	temp_addr<={s7,s6,s5,s4,s3,s2,s1,s0};
								temp_command<=make_call;
								menu_item<=def_outgoing;
								state<=outgoing;
							end

							//voicemail menu
							toggle_v: begin
								case (voicemail_state)
									1:voicemail_state<=0;
									0:voicemail_state<=1;
								endcase
							end

							//Change Volume Menu (save changes)
							change_vol:begin
```

```
        temp_headphone_volume<=temp_headphone_volume + headphone_change;
                                        menu_item<=volume;
                                    end

                                    //Unread Voicemail
                                    unread:begin
                                        if
(voicemail_status==STS_CMD_RDY) begin
                                             override<=0; //disable up/
down buttons for UI
        temp_voicemail_command<=CMD_VIEW_UNRD;
        temp_display_control<=voicemail_disp; //voicemail now controls display
        menu_item<=play_unread;
                                        end

                                        else
                                            menu_item<=unread;
                                    end

                                    //Saved Voicemail
                                    saved:begin
                                        if
(voicemail_status==STS_CMD_RDY) begin
                                            override<=0;
        temp_voicemail_command<=CMD_VIEW_SAVED;
        temp_display_control<=voicemail_disp;//voicemail now controls display
                                            menu_item<=play_saved;
                                        end

                                        else
                                            menu_item<=saved;
                                    end

                                    set_time:begin
                                        temp_set_date<=1;
                                        temp_display_control<=date_time;
                                        override<=0;
                                        menu_item<=set_dt;
                                    end
                                endcase
                            end
```

```
else if (left&&!left_latch) begin //move to higher level menu

        //go back to system date and time from
main menu

        if (menu_item>=0 && menu_item<=4) begin
            temp_display_control<=date_time;
                menu_item<=def_sys;
        end

        //back to call number
        else if (menu_item==7)
                menu_item<=call_number;

        //back to set headphone volume
        else if (menu_item==8)
                menu_item<=volume;

        //back to voicemail
        else if (menu_item>=9 && menu_item<=11)
                menu_item<=voicemail;

        //back to unread voicemail
        else if (menu_item>=12 &&
menu_item<=14) begin

                menu_item<=unread;
                override<=1; //up/down buttons
enabled for UI

                temp_display_control<=UI; //UI now
controls display

         end

        //back to saved voicemail
        else if (menu_item>=15 &&
menu_item<=17) begin

                menu_item<=saved;
                override<=1; //up/down buttons
enabled for UI

                temp_display_control<=UI;//UI now
controls display

        end

        //escape to display number
        else if (menu_item==30)
                menu_item<=get_num;
```

```
//                                                              //escape to set time/date
//                                                              else if (menu_item==31) begin
//                                                                      menu_item<=set_time;
//                                                                      temp_display_control<=UI;
//                                                              end

                                        end

                                        else if (b0&&!b0_latch&&(menu_item==15||
menu_item==12)) begin        //voicemail delete button
                                                case (menu_item)
                                                        play_unread:
temp_voicemail_command<=CMD_DEL;
                                                        play_saved:
temp_voicemail_command<=CMD_DEL;
                                                endcase
                                        end

                                        else if (b1&&!b1_latch&&(menu_item==15||
menu_item==12)) begin //voicemail save button
                                                        temp_voicemail_command<=CMD_SAVE;
                                        end

                                        else if (b2&&b2_latch&&(menu_item==15||
menu_item==12)) begin //voicemail stop button
                                                case (menu_item)
                                                        play_unread:
temp_voicemail_command<=CMD_END_RD;
                                                        play_saved:
temp_voicemail_command<=CMD_END_RD;
                                                endcase
                                        end

                                        else if (b3&&b3_latch&&(menu_item==play_saved||
menu_item==play_unread)) begin //voicemail play button
                                                case (menu_item)
                                                        play_unread:
temp_voicemail_command<=CMD_START_RD;
                                                        play_saved:
temp_voicemail_command<=CMD_START_RD;
                                                endcase
                                        end
```

```
					else if (b0&&!b1_latch&&menu_item==set_dt) begin
						override<=1;
						temp_set_date<=0;
						temp_display_control<=UI;
						menu_item<=set_time; //escape from setting time
					end

					//no button presses
					case (menu_item)
							//display Number
							get_num: begin
							end
					endcase


				end




				//incoming state logic
				incoming: begin
					start_t<=0;

					if (expired==0) begin
						if (up&&!up_latch) begin
							case (menu_item)
								def_incoming: begin
									if
(voicemail_status==STS_NO_CF_DEVICE||voicemail_state==0)
										menu_item<=reject;
									else
	menu_item<=send_to_v;
								end
								default: menu_item<=menu_item-1;
							endcase
						end

						else if (down&&!down_latch) begin
							case (menu_item)
								send_to_v:
menu_item<=def_incoming;

								reject:begin
```

```
                                                                    if
(voicemail_status==STS_NO_CF_DEVICE||voicemail_state==0)
        menu_item<=def_incoming;
                                                                    else
        menu_item<=send_to_v;
                                                            end
                                                            default: menu_item<=menu_item+1;
                                                    endcase
                                            end

                                            else if ((right&&!right_latch)||(enter&&!enter_latch))
begin
                                                    case (menu_item)
        def_incoming:temp_command<=disconnect;
                                                            accept:
temp_command<=answer_call;
                                                            reject: begin
                                                                    if (voicemail_status==1)
        temp_command<=go_to_voicemail;
                                                                    else
        temp_command<=disconnect;
                                                            end
        send_to_v:temp_command<=go_to_voicemail;
                                                    endcase
                                            end
                            end

                            //timer expired w/o call being answered
                            else begin
                                            if (voicemail_state==1)
                                                    temp_command<=go_to_voicemail;
                                            else
                                                    temp_command<=disconnect;
                            end



                                    case (menu_item)       //waiting on application layer
                                            def_incoming: begin
                                                            if (inc_command==connected) begin
                                                                    menu_item<=def_busy;
                                                                    state<=busy;
                                                            end
                                            end
```

```
                    accept: begin
                        if (inc_command==connected) begin
                                menu_item<=def_busy;
                                state<=busy;
                        end
                    end
                    reject:begin
                        if (inc_command==disconnected) begin
                            menu_item<=def_sys;
                            state<=idle;
                        end
                    end
                    send_to_v:begin
                        if (inc_command==disconnected)begin
                            menu_item<=def_sys;
                            state<=idle;
                        end
                    end
                endcase

            end



            //outgoing state logic
            outgoing: begin
                if (up&&override&&!up_latch) begin
                    case (menu_item)
                        def_outgoing: menu_item<=end_call;
                        default: menu_item<=menu_item-1;
                    endcase
                end

                else if (down&&override&&!down_latch) begin
                    case (menu_item)
                        end_call: menu_item<=def_outgoing;
                        default: menu_item<=menu_item+1;
                    endcase
                end

                else if (((enter&&!enter_latch)||(right&&!right_latch)) &&
override) begin
                    case(menu_item)
                        end_call: begin
```

```
						temp_command<=disconnect;
						if (inc_command==disconnected)
begin
							menu_item<=def_sys;
							state<=idle;
						end
					end
				endcase
			end




			if (inc_command==connected) begin
				menu_item<=def_busy;
				state<=busy;
			end

			else if (inc_command==no_answer||
inc_command==disconnected) begin //call disconnected or voicemail
				menu_item<=def_sys;
				state<=idle;
			end

			else if (inc_command==connected_to_v) begin //call sent
to voicemail
				override<=0;
				temp_display_control<=voicemail_disp;
				if (voicemail_status==STS_ERR_VM_FULL) begin
					override<=1;
					temp_display_control<=UI;
					temp_command<=disconnect;
				end

				else if (voicemail_status==STS_CMD_RDY)
					if (b0&&!b0_latch) //begin recording
	temp_voicemail_command<=CMD_START_WR;
					else if (b1&&b1_latch) begin //end recording
	temp_voicemail_command<=CMD_END_WR;
						override<=1;
						temp_command<=disconnect;
					end
```

```
                                                else if
(voicemail_status==STS_ERR_VM_FULL) begin //voicemail becomes full while recording
                                                        override<=1;
                                                        temp_display_control<=UI;
                                                        temp_command<=disconnect;
                                                end
                                        end
                        end




                        //busy state
                        busy: begin
                                if (up&&!up_latch) begin
                                        case (menu_item)
                                                def_busy: menu_item<=set_volume;
                                                change_vol: begin
                                                        if (headphone_volume<31)
        headphone_change<=headphone_change+1;
                                                end
                                                default: menu_item<=menu_item-1;
                                        endcase
                                end
                                else if (down&&!down_latch) begin
                                        case (menu_item)
                                                set_volume: menu_item<=def_busy;
                                                change_vol: begin
                                                        if (headphone_volume>0)
        headphone_change<=headphone_change-1;
                                                end
                                                default: menu_item<=menu_item+1;
                                        endcase
                                end
                                else if ((right&&!right_latch)||(enter&&!enter_latch)) begin
                                        case (menu_item)
                                                end_call_b:
        temp_command<=disconnect;
                                                set_volume: menu_item<=change_vol;
                                                change_vol:begin
                                                        if (headphone_volume<31) begin
        temp_headphone_volume<=temp_headphone_volume + headphone_change;
        menu_item<=change_vol;
                                                                end
```

```
                                        end
                                endcase
                        end
                        else if (left&&!left_latch) begin
                                case (menu_item)
                                        change_vol: menu_item<=set_volume;
                                endcase
                        end

                        case (menu_item) //no button presses currently
                                end_call_b: begin
                                        if (inc_command==disconnected) begin
                                                menu_item<=def_sys;
                                                state<=idle;
                                        end
                                end
                        endcase

                end
            endcase
        end

    end


    //assign outputs
    assign current_state=c_state;
    assign address=temp_addr;
    assign command=temp_command;
    assign current_menu_item=menu_item;
    assign headphone_volume=temp_headphone_volume;
    assign voicemail_command=temp_voicemail_command;
    assign disp_control=temp_display_control;
    assign start_timer=start_t;
    assign txt_addr=addr;
    assign txt_length=length;
    assign txt_start=start;
    assign set_date=temp_set_date;



endmodule
```

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Nandi Bugg
//
// Create Date:   16:22:15 12/10/2012
// Design Name:   user_interface
// Module Name:   /afs/athena.mit.edu/user/n/b/nbugg/FPGA_Telephony/UI/UI_Part2/
new_ui_test.v
// Project Name:  UI_Part2
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: user_interface
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module new_ui_test;

	// Inputs
	reg clk;
	reg s7;
	reg s6;
	reg s5;
	reg s4;
	reg s3;
	reg s2;
	reg s1;
	reg s0;
	reg b3;
	reg b2;
	reg b1;
	reg b0;
	reg reset;
	reg enter;
```

```
reg up;
reg down;
reg left;
reg right;
reg [3:0] inc_command;
reg init;
reg [7:0] inc_address;
reg ready;
reg [3:0] voicemail_status;
reg [15:0] dout;
reg [15:0] audio_in_data;
reg done;

// Outputs
wire [15:0] audio_out_data;
wire [3:0] voicemail_command;
wire [7:0] phn_num;
wire [15:0] din;
wire [1:0] disp_control;
wire [7:0] address;
wire [4:0] command;
wire [2:0] current_state;
wire [5:0] current_menu_item;
wire [4:0] headphone_volume;
wire [11:0] txt_addr;
wire [11:0] txt_length;
wire txt_start;
wire set_date;

// Instantiate the Unit Under Test (UUT)
user_interface uut (
        .clk(clk),
        .s7(s7),
        .s6(s6),
        .s5(s5),
        .s4(s4),
        .s3(s3),
        .s2(s2),
        .s1(s1),
        .s0(s0),
        .b3(b3),
        .b2(b2),
        .b1(b1),
        .b0(b0),
```

```
		.reset(reset),
		.enter(enter),
		.up(up),
		.down(down),
		.left(left),
		.right(right),
		.inc_command(inc_command),
		.init(init),
		.inc_address(inc_address),
		.audio_out_data(audio_out_data),
		.ready(ready),
		.voicemail_status(voicemail_status),
		.dout(dout),
		.voicemail_command(voicemail_command),
		.phn_num(phn_num),
		.din(din),
		.disp_control(disp_control),
		.address(address),
		.command(command),
		.current_state(current_state),
		.current_menu_item(current_menu_item),
		.headphone_volume(headphone_volume),
		.audio_in_data(audio_in_data),
		.txt_addr(txt_addr),
		.txt_length(txt_length),
		.txt_start(txt_start),
		.done(done),
		.set_date(set_date)
	);

	always #5 clk=!clk;

	initial begin
		// Initialize Inputs
		clk = 0;
		s7 = 0;
		s6 = 0;
		s5 = 0;
		s4 = 0;
		s3 = 0;
		s2 = 0;
		s1 = 0;
		s0 = 0;
		b3 = 0;
```

```
b2 = 0;
b1 = 0;
b0 = 0;
reset = 0;
enter = 0;
up = 0;
down = 0;
left = 0;
right = 0;
inc_command = 0;
init = 0;
inc_address = 0;
ready = 0;
voicemail_status = 0;
dout = 0;
audio_in_data = 0;
done = 1;

// Wait 100 ns for global reset to finish
#100;

// Add stimulus here
                    // Add stimulus here
enter=1;
#10;
enter=0; //should be in idle mode - checked
#10;
right=1;
#10;
right=0; //move from welcome message to main menu - checked
#10;
inc_command=4'd6; //incoming call, move to incoming mode - checked
#10;
down=1; //move to accept menu item - checked
#10;
down=0;
#10;
enter=1; //accept call - wait to be connected -
#10;
enter=0;
#10
inc_command=4'd2; //connected - checked
#10; //transition to busy states
down=1;
```

```
#10;
down=0; //move to end call menu item - checked
#10;
enter=1;          //end call, wait for signal from application layer -
#10
enter=0;
#10;
inc_command=4'd0; //call ended, back to idle state - checked
#10; //system date and time should be displayed -checked
right=1;
#10;
right=0; //enter main menu, call number menu item - checked
#10;
left=1;
#10;
left=0; //display sys date & time -checked
#10;
right=1; //back to main menu - checked
#10;
right=0;
#10;
enter=1; //call number, dialing menu item - checked
#10;
enter=0;
#10;
s2=1; //dial number 4, transition to outgoing state - checked
#10;
enter=1;          //transition to outgoing state - checked
#10;
enter=0;
#10;
inc_command=4'd2;//connected, go to busy state - checked
#30;
down=1;
#10;
down=0; //move to end call menu item - checked
#10;
enter=1;          //end call, wait for signal from application layer - checked
#10
enter=0;
#10;
inc_command=4'd0; //call ended, back to idle state - checked
#10; //system date and time should be displayed -checked
reset=1;
```

```
        #10;
        reset=0;//back to initialization state - checked




    end

endmodule
```

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Nandi Bugg
//
// Create Date:   16:37:42 12/10/2012
// Design Name:   user_interface
// Module Name:   /afs/athena.mit.edu/user/n/b/nbugg/FPGA_Telephony/UI/UI_Part2/
new_voicemail_interface.v
// Project Name:  UI_Part2
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: user_interface
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module new_voicemail_interface;

	// Inputs
	reg clk;
	reg s7;
	reg s6;
	reg s5;
	reg s4;
	reg s3;
	reg s2;
	reg s1;
	reg s0;
	reg b3;
	reg b2;
	reg b1;
	reg b0;
	reg reset;
	reg enter;
	reg up;
```

```
reg down;
reg left;
reg right;
reg [3:0] inc_command;
reg init;
reg [7:0] inc_address;
reg ready;
reg [3:0] voicemail_status;
reg [15:0] dout;
reg [15:0] audio_in_data;
reg done;

// Outputs
wire [15:0] audio_out_data;
wire [3:0] voicemail_command;
wire [7:0] phn_num;
wire [15:0] din;
wire [1:0] disp_control;
wire [7:0] address;
wire [4:0] command;
wire [2:0] current_state;
wire [5:0] current_menu_item;
wire [4:0] headphone_volume;
wire [11:0] txt_addr;
wire [11:0] txt_length;
wire txt_start;
wire set_date;

// Instantiate the Unit Under Test (UUT)
user_interface uut (
        .clk(clk),
        .s7(s7),
        .s6(s6),
        .s5(s5),
        .s4(s4),
        .s3(s3),
        .s2(s2),
        .s1(s1),
        .s0(s0),
        .b3(b3),
        .b2(b2),
        .b1(b1),
        .b0(b0),
        .reset(reset),
```

```
		.enter(enter),
		.up(up),
		.down(down),
		.left(left),
		.right(right),
		.inc_command(inc_command),
		.init(init),
		.inc_address(inc_address),
		.audio_out_data(audio_out_data),
		.ready(ready),
		.voicemail_status(voicemail_status),
		.dout(dout),
		.voicemail_command(voicemail_command),
		.phn_num(phn_num),
		.din(din),
		.disp_control(disp_control),
		.address(address),
		.command(command),
		.current_state(current_state),
		.current_menu_item(current_menu_item),
		.headphone_volume(headphone_volume),
		.audio_in_data(audio_in_data),
		.txt_addr(txt_addr),
		.txt_length(txt_length),
		.txt_start(txt_start),
		.done(done),
		.set_date(set_date)
	);

	always #5 clk=!clk;
	initial begin
		// Initialize Inputs
		clk = 0;
		s7 = 0;
		s6 = 0;
		s5 = 0;
		s4 = 0;
		s3 = 0;
		s2 = 0;
		s1 = 0;
		s0 = 0;
		b3 = 0;
		b2 = 0;
		b1 = 0;
```

```
b0 = 0;
reset = 0;
enter = 0;
up = 0;
down = 0;
left = 0;
right = 0;
inc_command = 0;
init = 0;
inc_address = 0;
ready = 0;
voicemail_status = 0;
dout = 0;
audio_in_data = 0;
done = 1;

// Wait 100 ns for global reset to finish
#100;

// Add stimulus here
enter=1;
#10;
enter=0; //should be in idle mode - checked
#10;
right=1;
#10;
right=0; //move from welcome message to main menu - checked
#10;
down=1;
#10;
down=0;
#10;
down=1;
#10;
down=0; //should be at get_num menu_item - checked
#10;
voicemail_status=4'd1; //CF card detected
#10;
up=1;
#10;
up=0; //should be at voicemail menu_item - checked
#10;
enter=1;
#10;
```

```
        enter=0; //toggle voicemail menu - checked
        #10;
        enter=1;
        #10;
        enter=0; //turn voicemail on -checked



    end

endmodule
```

```
///////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Template Toplevel Module
//
// For Labkit Revision 004
//
//
// Created: October 31, 2004, from revision 003 file
// Author: Nathan Ickes
//
///////////////////////////////////////////////////////////////////////////////
//
// CHANGES FOR BOARD REVISION 004
//
// 1) Added signals for logic analyzer pods 2-4.
// 2) Expanded "tv_in_ycrcb" to 20 bits.
// 3) Renamed "tv_out_data" to "tv_out_i2c_data" and "tv_out_sclk" to
//    "tv_out_i2c_clock".
// 4) Reversed disp_data_in and disp_data_out signals, so that "out" is an
//    output of the FPGA, and "in" is an input.
//
// CHANGES FOR BOARD REVISION 003
//
// 1) Combined flash chip enables into a single signal, flash_ce_b.
//
// CHANGES FOR BOARD REVISION 002
//
// 1) Added SRAM clock feedback path input and output
// 2) Renamed "mousedata" to "mouse_data"
// 3) Renamed some ZBT memory signals. Parity bits are now incorporated into
//    the data bus, and the byte write enables have been combined into the
//    4-bit ram#_bwe_b bus.
// 4) Removed the "systemace_clock" net, since the SystemACE clock is now
//    hardwired on the PCB to the oscillator.
//
///////////////////////////////////////////////////////////////////////////////
//
// Complete change history (including bug fixes)
//
// 2006-Mar-08: Corrected default assignments to "vga_out_red", "vga_out_green"
//              and "vga_out_blue". (Was 10'h0, now 8'h0.)
//
// 2005-Sep-09: Added missing default assignments to "ac97_sdata_out",
//              "disp_data_out", "analyzer[2-3]_clock" and
```

```
//            "analyzer[2-3]_data".
//
// 2005-Jan-23: Reduced flash address bus to 24 bits, to match 128Mb devices
//            actually populated on the boards. (The boards support up to
//            256Mb devices, with 25 address lines.)
//
// 2004-Oct-31: Adapted to new revision 004 board.
//
// 2004-May-01: Changed "disp_data_in" to be an output, and gave it a default
//            value. (Previous versions of this file declared this port to
//            be an input.)
//
// 2004-Apr-29: Reduced SRAM address busses to 19 bits, to match 18Mb devices
//            actually populated on the boards. (The boards support up to
//            72Mb devices, with 21 address lines.)
//
// 2004-Apr-29: Change history started
//
///////////////////////////////////////////////////////////////////////////////

//File used to mux audio and display for UI

module labkit (beep, audio_reset_b, ac97_sdata_out, ac97_sdata_in, ac97_synch,
              ac97_bit_clock,

              vga_out_red, vga_out_green, vga_out_blue, vga_out_sync_b,
              vga_out_blank_b, vga_out_pixel_clock, vga_out_hsync,
              vga_out_vsync,

              tv_out_ycrcb, tv_out_reset_b, tv_out_clock, tv_out_i2c_clock,
              tv_out_i2c_data, tv_out_pal_ntsc, tv_out_hsync_b,
              tv_out_vsync_b, tv_out_blank_b, tv_out_subcar_reset,

              tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1,
              tv_in_line_clock2, tv_in_aef, tv_in_hff, tv_in_aff,
              tv_in_i2c_clock, tv_in_i2c_data, tv_in_fifo_read,
              tv_in_fifo_clock, tv_in_iso, tv_in_reset_b, tv_in_clock,

              ram0_data, ram0_address, ram0_adv_ld, ram0_clk, ram0_cen_b,
              ram0_ce_b, ram0_oe_b, ram0_we_b, ram0_bwe_b,

              ram1_data, ram1_address, ram1_adv_ld, ram1_clk, ram1_cen_b,
              ram1_ce_b, ram1_oe_b, ram1_we_b, ram1_bwe_b,
```

```
		clock_feedback_out, clock_feedback_in,

		flash_data, flash_address, flash_ce_b, flash_oe_b, flash_we_b,
		flash_reset_b, flash_sts, flash_byte_b,

		rs232_txd, rs232_rxd, rs232_rts, rs232_cts,

		mouse_clock, mouse_data, keyboard_clock, keyboard_data,

		clock_27mhz, clock1, clock2,

		disp_blank, disp_data_out, disp_clock, disp_rs, disp_ce_b,
		disp_reset_b, disp_data_in,

		button0, button1, button2, button3, button_enter, button_right,
		button_left, button_down, button_up,

		switch,

		led,

		user1, user2, user3, user4,

		daughtercard,

		systemace_data, systemace_address, systemace_ce_b,
		systemace_we_b, systemace_oe_b, systemace_irq, systemace_mpbrdy,

		analyzer1_data, analyzer1_clock,
		analyzer2_data, analyzer2_clock,
		analyzer3_data, analyzer3_clock,
		analyzer4_data, analyzer4_clock);

output beep, ac97_synch, ac97_sdata_out,audio_reset_b;
input  ac97_bit_clock, ac97_sdata_in;

output [7:0] vga_out_red, vga_out_green, vga_out_blue;
output vga_out_sync_b, vga_out_blank_b, vga_out_pixel_clock,
	vga_out_hsync, vga_out_vsync;

output [9:0] tv_out_ycrcb;
output tv_out_reset_b, tv_out_clock, tv_out_i2c_clock, tv_out_i2c_data,
	tv_out_pal_ntsc, tv_out_hsync_b, tv_out_vsync_b, tv_out_blank_b,
	tv_out_subcar_reset;
```

```
input  [19:0] tv_in_ycrcb;
input  tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2, tv_in_aef,
         tv_in_hff, tv_in_aff;
output tv_in_i2c_clock, tv_in_fifo_read, tv_in_fifo_clock, tv_in_iso,
         tv_in_reset_b, tv_in_clock;
inout  tv_in_i2c_data;

inout  [35:0] ram0_data;
output [18:0] ram0_address;
output ram0_adv_ld, ram0_clk, ram0_cen_b, ram0_ce_b, ram0_oe_b, ram0_we_b;
output [3:0] ram0_bwe_b;

inout  [35:0] ram1_data;
output [18:0] ram1_address;
output ram1_adv_ld, ram1_clk, ram1_cen_b, ram1_ce_b, ram1_oe_b, ram1_we_b;
output [3:0] ram1_bwe_b;

input  clock_feedback_in;
output clock_feedback_out;

inout  [15:0] flash_data;
output [23:0] flash_address;
output flash_ce_b, flash_oe_b, flash_we_b, flash_reset_b, flash_byte_b;
input  flash_sts;

output rs232_txd, rs232_rts;
input  rs232_rxd, rs232_cts;

input  mouse_clock, mouse_data, keyboard_clock, keyboard_data;

input  clock_27mhz, clock1, clock2;

output disp_blank, disp_clock, disp_rs, disp_ce_b, disp_reset_b;
input  disp_data_in;
output  disp_data_out;

input  button0, button1, button2, button3, button_enter, button_right,
         button_left, button_down, button_up;
input  [7:0] switch;
output [7:0] led;

inout [31:0] user1, user2, user3, user4;
```

```
   inout [43:0] daughtercard;

   inout  [15:0] systemace_data;
   output [6:0]  systemace_address;
   output systemace_ce_b, systemace_we_b, systemace_oe_b;
   input  systemace_irq, systemace_mpbrdy;

   output [15:0] analyzer1_data, analyzer2_data, analyzer3_data,
                 analyzer4_data;
   output analyzer1_clock, analyzer2_clock, analyzer3_clock, analyzer4_clock;

   ////////////////////////////////////////////////////////////////////////////
   //
   // I/O Assignments
   //
   ////////////////////////////////////////////////////////////////////////////

   // Audio Input and Output
   assign beep= 1'b0;
//   assign audio_reset_b = 1'b0;
//   assign ac97_synch = 1'b0;
//   assign ac97_sdata_out = 1'b0;
   // ac97_sdata_in is an input

   // VGA Output
   assign vga_out_red = 8'h0;
   assign vga_out_green = 8'h0;
   assign vga_out_blue = 8'h0;
   assign vga_out_sync_b = 1'b1;
   assign vga_out_blank_b = 1'b1;
   assign vga_out_pixel_clock = 1'b0;
   assign vga_out_hsync = 1'b0;
   assign vga_out_vsync = 1'b0;

   // Video Output
   assign tv_out_ycrcb = 10'h0;
   assign tv_out_reset_b = 1'b0;
   assign tv_out_clock = 1'b0;
   assign tv_out_i2c_clock = 1'b0;
   assign tv_out_i2c_data = 1'b0;
   assign tv_out_pal_ntsc = 1'b0;
   assign tv_out_hsync_b = 1'b1;
   assign tv_out_vsync_b = 1'b1;
   assign tv_out_blank_b = 1'b1;
```

```
   assign tv_out_subcar_reset = 1'b0;

   // Video Input
   assign tv_in_i2c_clock = 1'b0;
   assign tv_in_fifo_read = 1'b0;
   assign tv_in_fifo_clock = 1'b0;
   assign tv_in_iso = 1'b0;
   assign tv_in_reset_b = 1'b0;
   assign tv_in_clock = 1'b0;
   assign tv_in_i2c_data = 1'bZ;
   // tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2,
   // tv_in_aef, tv_in_hff, and tv_in_aff are inputs

   // SRAMs
//   assign ram0_data = 36'hZ;
//   assign ram0_address = 19'h0;
   assign ram0_adv_ld = 1'b0;
//   assign ram0_clk = 1'b0;
   assign ram0_cen_b = 1'b1;
   assign ram0_ce_b = 1'b1;
   assign ram0_oe_b = 1'b1;
//   assign ram0_we_b = 1'b1;
//   assign ram0_bwe_b = 4'hF;
   assign ram1_data = 36'hZ;
   assign ram1_address = 19'h0;
   assign ram1_adv_ld = 1'b0;
//   assign ram1_clk = 1'b0;
   assign ram1_cen_b = 1'b1;
   assign ram1_ce_b = 1'b1;
   assign ram1_oe_b = 1'b1;
   assign ram1_we_b = 1'b1;
   assign ram1_bwe_b = 4'hF;
//   assign clock_feedback_out = 1'b0;
   // clock_feedback_in is an input

   // Flash ROM
   assign flash_data = 16'hZ;
   assign flash_address = 24'h0;
   assign flash_ce_b = 1'b1;
   assign flash_oe_b = 1'b1;
   assign flash_we_b = 1'b1;
   assign flash_reset_b = 1'b0;
   assign flash_byte_b = 1'b1;
   // flash_sts is an input
```

```
   // RS-232 Interface
   assign rs232_txd = 1'b1;
   assign rs232_rts = 1'b1;
   // rs232_rxd and rs232_cts are inputs

   // PS/2 Ports
   // mouse_clock, mouse_data, keyboard_clock, and keyboard_data are inputs

   // LED Displays
//   assign disp_blank = 1'b1;
//   assign disp_clock = 1'b0;
//   assign disp_rs = 1'b0;
//   assign disp_ce_b = 1'b1;
//   assign disp_reset_b = 1'b0;
//   assign disp_data_out = 1'b0;
   // disp_data_in is an input

   // Buttons, Switches, and Individual LEDs
   assign led = 8'hFF;
   // button0, button1, button2, button3, button_enter, button_right,
   // button_left, button_down, button_up, and switches are inputs

   // User I/Os
   assign user1 = 32'hZ;
   assign user2 = 32'hZ;
   assign user3 = 32'hZ;
   assign user4 = 32'hZ;

   // Daughtercard Connectors
   assign daughtercard = 44'hZ;

   // SystemACE Microprocessor Port
//   assign systemace_data = 16'hZ;
//   assign systemace_address = 7'h0;
//   assign systemace_ce_b = 1'b1;
//   assign systemace_we_b = 1'b1;
//   assign systemace_oe_b = 1'b1;
   // systemace_irq and systemace_mpbrdy are inputs

   // Logic Analyzer
//   assign analyzer1_data[15:9] = 0;
//   assign analyzer1_clock = 1'b1;
   assign analyzer2_data[15:7]= 9'h0;
```

```
//   assign analyzer2_clock = clock_27mhz;
//   assign analyzer3_data = 16'h0;
//   assign analyzer3_clock = clock_27mhz;
   assign analyzer4_data = 16'h0;
//   assign analyzer4_clock = clock_27mhz;


        ////////////////////////////////////////////////////
        //Project Code Starts Here
        ///////////////////////////////////////////////////////
        //Declarations for Buttons
        wire reset,debb0,debb1,debb2,debb3,
        debup,debdown,debleft,debright,debenter,
        b0,b1,b2,b3,up,down,left,right,enter;

        //Declarations for Switches
        wire s0,s1,s2,s3,s4,s5,s6,s7;

        //UI inputs
        wire [3:0] inc_command;
        wire init, ready;
        wire [7:0] inc_address;
        wire [15:0] audio_in_data, dout,session_audio_data;
        wire [3:0] voicemail_status;

        //UI 2 Inputs
        wire [3:0] inc_command_2;
        wire init_2, ready_2;
        wire [7:0] inc_address_2;
        wire [15:0] audio_in_data_2, dout_2,session_audio_data_2;
        wire [3:0] voicemail_status;




        //UI outputs
        wire [3:0] voicemail_command;
        wire [7:0] phn_num,address,ui_ascii_out,ascii_out;
        wire [15:0] din,audio_out_data;
        wire [1:0] disp_control;
        wire [2:0] current_state;
        wire [5:0] current_menu_item;
        wire [4:0] headphone_volume, command;
        wire ui_ascii_out_ready, start,ascii_out_ready;
```

```
wire [10:0] addr,length;
wire done, over;

reg [7:0] a_out;
reg a_out_ready;

//RAM clock I/O
wire clk_27mhz,locked;


//RAM clock
ramclock RAMCLOCK_1(
        .ref_clock(clock_27mhz),
        .fpga_clock(clk_27mhz),
        .ram0_clock(ram0_clk),
        .ram1_clock(ram1_clk),
  .clock_feedback_in(clock_feedback_in),
        .clock_feedback_out(clock_feedback_out),
        .locked(locked)
);



//Debounce buttons
debounce db0(.reset(reset),.clock(clk_27mhz),.noisy(~button0),
.clean(debb0));
debounce db1(.reset(reset),.clock(clk_27mhz),.noisy(~button1),
.clean(debb1));
debounce db2(.reset(reset),.clock(clk_27mhz),.noisy(~button2),
.clean(debb2));
debounce db3(.reset(reset),.clock(clk_27mhz),.noisy(~button3),
.clean(debb3));

debounce dup(.reset(reset),.clock(clk_27mhz),.noisy(~button_up),
.clean(debup));
debounce ddown(.reset(reset),.clock(clk_27mhz),.noisy(~button_down),
.clean(debdown));
debounce dleft(.reset(reset),.clock(clk_27mhz),.noisy(~button_left),
.clean(debleft));
debounce dright(.reset(reset),.clock(clk_27mhz),.noisy(~button_right),
.clean(debright));
debounce denter(.reset(reset),.clock(clk_27mhz),.noisy(~button_enter),
.clean(debenter));
```

```
//Contention Resolver
Button_Contention_Resolver bcr(.clk(clk_27mhz),.reset(reset),
.button0_in(debb0),.button1_in(debb1),.button2_in(debb2),
.button3_in(debb3),.button_enter_in(debenter),
.button_left_in(debleft),.button_right_in(debright),
.button_up_in(debup),.button_down_in(debdown),
.button0_out(b0),.button1_out(b1),.button2_out(b2),
.button3_out(b3),.button_enter_out(enter),
.button_left_out(left),.button_right_out(right),
.button_up_out(up),.button_down_out(down));


//Synchronize Switches
synchronize #(.NSYNC(2)) synch6(.clk(clk_27mhz),.in(switch[7]),.out(s7));
synchronize #(.NSYNC(2)) synch7(.clk(clk_27mhz),.in(switch[5]),.out(s5));
synchronize #(.NSYNC(2)) synch8(.clk(clk_27mhz),.in(switch[4]),.out(s4));
synchronize #(.NSYNC(2)) synch9(.clk(clk_27mhz),.in(switch[3]),.out(s3));
synchronize #(.NSYNC(2)) synch10(.clk(clk_27mhz),.in(switch[2]),.out(s2));
synchronize #(.NSYNC(2)) synch11(.clk(clk_27mhz),.in(switch[1]),.out(s1));
synchronize #(.NSYNC(2)) synch12(.clk(clk_27mhz),.in(switch[0]),.out(s0));


//AC97
AC97_PCM ac(.clock_27mhz(clk_27mhz),.reset(reset),.volume(headphone_volume),
.audio_in_data(audio_in_data),.audio_out_data(audio_out_data),
.ready(ready),
.audio_reset_b(audio_reset_b),.ac97_sdata_out(ac97_sdata_out),
.ac97_sdata_in(ac97_sdata_in),
.ac97_synch(ac97_synch),.ac97_bit_clock(ac97_bit_clock));

wire ui_ready;

//Instantiate User Interface module
user_interface ui(.clk(clk_27mhz),.s7(s7),.s6(s6),.s5(s5),.s4(s4),
.s3(s3),.s2(s2),.s1(s1),.s0(s0),.b3(b3),.b2(b2),
.b1(b1),.b0(b0),.reset(reset),.enter(enter),
.up(up),.down(down),.left(left),.right(right),
.inc_command(inc_command),.init(init),
.audio_in_data(audio_in_data),.ready(ui_ready),
.voicemail_status(voicemail_status),.voicemail_command(voicemail_command),
.phn_num(phn_num),.dout(dout),
.din(din),.disp_control(disp_control),
.inc_address(inc_address),.address(address),
.command(command),.current_state(current_state),
```

```
.current_menu_item(current_menu_item),
.headphone_volume(headphone_volume),
.audio_out_data(audio_out_data),.txt_addr(addr),
.txt_length(length),.txt_start(start),.done(done),.set_date(set));

/////////////////////////////////////////
//Text Interfaces
///////////////////////////////////////////
wire [127:0] string_data;
wire wr_en_DEBUG;
wire [7:0] wr_data_DEBUG;
wire [10:0] wr_addr_DEBUG;
wire cntr_DEBUG;
wire set_disp_DEBUG;
wire [3:0] rel_pos_DEBUG;
wire [10:0] rd_addr_DEBUG;
wire [7:0] rd_data_DEBUG;

Text_Scroller_Interface tsi(.clk(clk_27mhz),.reset(reset),
.addr(addr),.length(length),.start(start),
.ascii_out(ui_ascii_out),.ascii_out_ready(ui_ascii_out_ready),.done(done));

Text_Scroller ts (.clk(clk_27mhz),.reset(reset),.ascii_data(ascii_out),
.ascii_data_ready(ascii_out_ready),.string_data(string_data),.wr_en_DEBUG(wr_en_DE
BUG)
,.wr_data_DEBUG(wr_data_DEBUG),.wr_addr_DEBUG(wr_addr_DEBUG)
,.cntr_DEBUG(cntr_DEBUG),.set_disp_DEBUG(set_disp_DEBUG),
.rel_pos_DEBUG(rel_pos_DEBUG),.rd_addr_DEBUG(rd_addr_DEBUG),
.rd_data_DEBUG(rd_data_DEBUG));

//display here
display_string ds(.reset(reset),.clock_27mhz(clk_27mhz),
.string_data(string_data),.disp_blank(disp_blank),
.disp_clock(disp_clock),.disp_rs(disp_rs),
.disp_ce_b(disp_ce_b),.disp_reset_b(disp_reset_b)
,.disp_data_out(disp_data_out));

//Voicemail I/Os
reg vmail_disp_en;
wire [6:0] vm_addr;
wire [7:0] vm_data;
wire v_disp_en;
wire [7:0] v_ascii_out;
wire v_ascii_out_ready;
```

```
//Date and Time I/Os
wire date_disp_en;
reg d_disp_en;
wire [6:0] year;
wire [3:0] month;
wire [4:0] day, hour;
wire [5:0] minute, second;
wire [7:0] date_ascii_out;
wire date_ascii_out_ready;
wire [6:0] DT_addr;
wire [7:0] DT_data;

//Binary to Decimal
Binary_to_Decimal BtD1(
        .clka(clk_27mhz),
        .clkb(clk_27mhz),
        .addra(DT_addr),
        .addrb(vm_addr),
        .douta(DT_data),
        .doutb(vm_data)
);


//System Date and Time
Date_Time dt(.clk_27mhz(clk_27mhz),.reset(reset),
.set(set),.disp_en(date_disp_en),.button_up(up),
.button_down(down),.button_left(left),
.button_right(right),.year(year),.month(month),
.day(day),.hour(hour),.minute(minute),.second(second),
.ascii_out(date_ascii_out),.ascii_out_ready(date_ascii_out_ready)
,.addr(DT_addr),.data(DT_data));


//Voicemail
 Voicemail_Interface vmail(.clk_27mhz(clk_27mhz),
.reset(reset),.sts(voicemail_status),.cmd(voicemail_command),
.phn_num(phn_num),.din(audio_out_data),.dout(dout),.d_ready(ready),
.disp_en(v_disp_en),.button_up(up),.button_down(down),
.ascii_out(v_ascii_out),.ascii_out_ready(v_ascii_out_ready),
.ram_data(ram0_data),.ram_address(ram0_address),.ram_we_b(ram0_we_b),.ram_bwe
_b(ram0_bwe_b),
.year(year),.month(month),.day(day),.hour(hour),.minute(minute),.addr(vm_addr),
```

```
	.data(vm_data),.systemace_data(systemace_data),.systemace_address(systemace_add
ress),
	.systemace_ce_b(systemace_ce_b),.systemace_we_b(systemace_we_b),
	.systemace_oe_b(systemace_oe_b),.systemace_mpbrdy(systemace_mpbrdy));

	//Display Control Mux
	always @(posedge clk_27mhz) begin
		case(disp_control)
			0: begin //UI case
				vmail_disp_en<=0;
				d_disp_en<=0;
				a_out<=ui_ascii_out;
				a_out_ready<=ui_ascii_out_ready;
			end
			1: begin //Date&Time
				vmail_disp_en<=0;
				a_out<=date_ascii_out;
				a_out_ready<=date_ascii_out_ready;
				d_disp_en<=1;
			end
			2:	begin //Voicemail
				d_disp_en<=0;
				a_out<=v_ascii_out;
				a_out_ready<=v_ascii_out_ready;
				vmail_disp_en<=1;
			end
		endcase
	end

	assign v_disp_en=vmail_disp_en;
	assign date_disp_en=d_disp_en;
	assign ascii_out_ready=a_out_ready;
	assign ascii_out=a_out;

	//Logic Analyzer
	assign analyzer1_data[7:0] = ascii_out;
	assign analyzer1_data[8]=ascii_out_ready;
	assign analyzer1_clock=clk_27mhz;
	assign analyzer1_data[9]=start;
	assign analyzer1_data[15:10]=addr[10:5];
	assign analyzer3_data[3:0]=voicemail_status;
	assign analyzer3_data[5:4]=disp_control;
	assign analyzer3_data[15:6]=0;
	assign analyzer2_data[0]=a_out;
```

```
	assign analyzer2_data[6:1]=current_menu_item;
	assign analyzer2_clock=clk_27mhz;
	assign analyzer3_clock=clk_27mhz;
	assign analyzer4_clock=clk_27mhz;



	//////////////////////////////////////////////////

endmodule
```