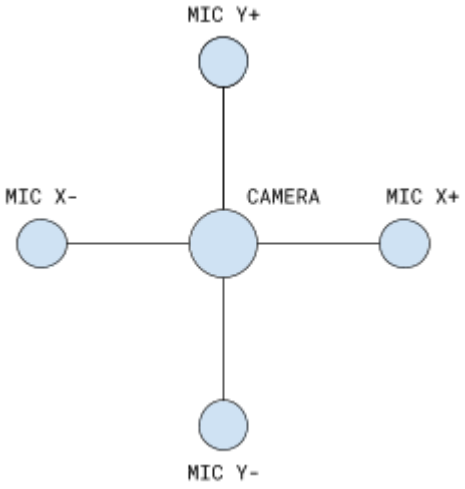
6.111 Project Report

Auditory Localization

Francis Wang and Keshav Gupta

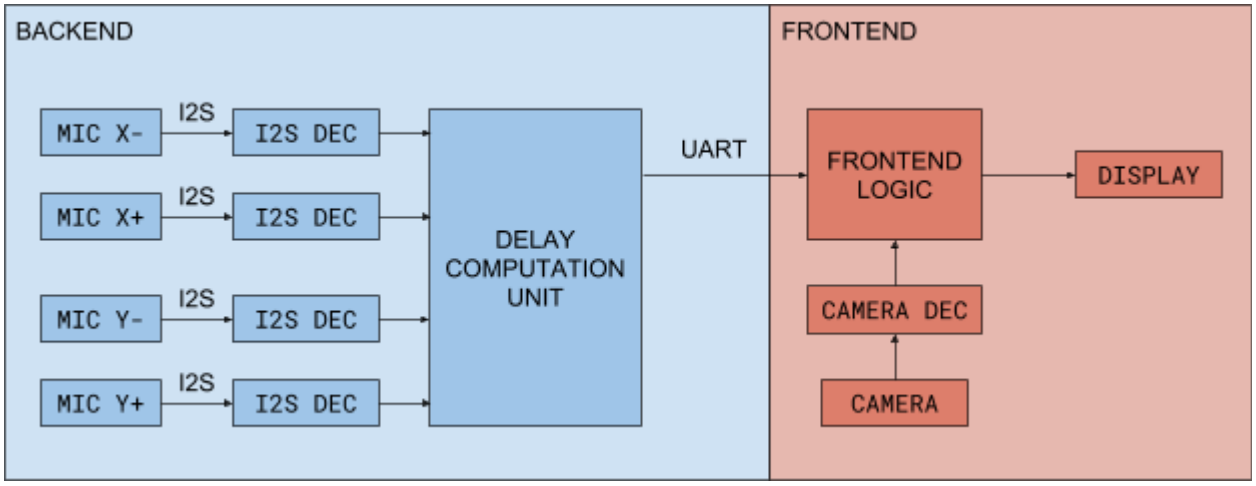
Overview

In this project we demonstrate a system that is able to discern the direction of incoming sound through an array of microphones. The physical configuration of the system is as depicted in the diagram to the right. Four microphones will be arranged in a cross and the direction of the sound source is determined by analyzing the relative delay of the waveforms measured by each pair of microphones in the X and Y axis. At the center of the cross is a camera, and the position of the inferred sound source is overlaid on the scene and displayed on a computer monitor.



For the microphones we have used the Adafruit I2S MEMS Microphone Breakout Board based on the SPH0645LM4H IC. It is an 18-bit digital microphone with a sampling frequency of up to 64 kHz. We have chosen a digital microphone as it can provide better audio fidelity and we would not have to worry about signal distortion during data transmission. For the camera we have used a standard NTSC camera provided by the lab.

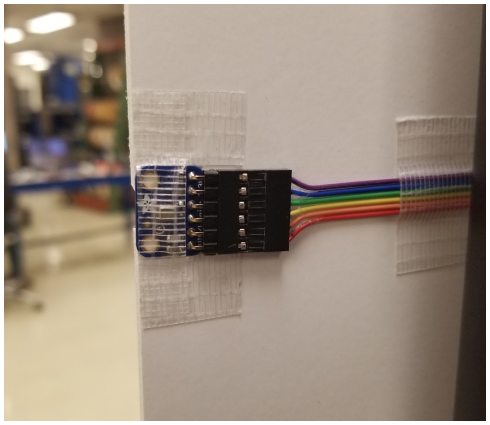
We have split this project into two sections, which we will refer to as the “backend” and the “frontend.” The backend is responsible for decoding the audio signals from the microphones and performing the digital signal processing to extract the time delay information. The frontend takes this delay information and combines it with the video feed from the camera to present a visual user interface on a VGA monitor. In order to separate the computationally complex signal processing from the more fluid user interface layer, we have developed our system on two FPGAs. The backend is implemented on the Nexys 4 while the frontend is implemented on the Virtex 2 labkit. The two systems communicate over the UART protocol based on an agreed upon data exchange format. A high level block diagram of the system is presented below.



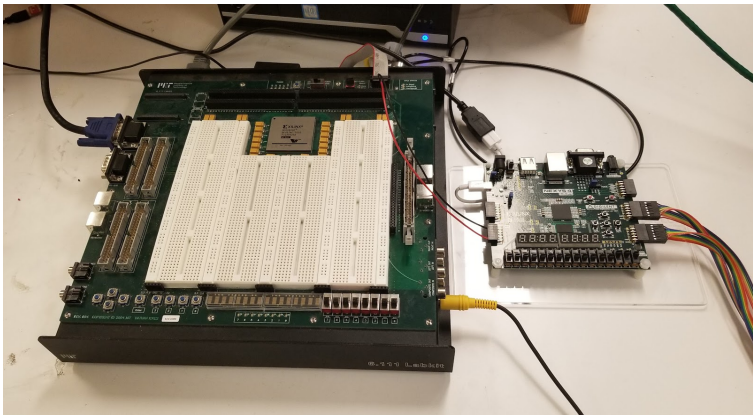
The signal processing required for auditory localization is highly parallelizable and well suited for implementation in hardware. It has applications in robotics and other interactive systems such as smart home devices.

Setup

Our microphones and camera are attached to a 40” x 32” foam board, and the entire setup is mounted against a portable free standing whiteboard. The camera is taped to the center of the board and the microphones are fitted to notches cut in the middle of each edge of the foam board. The microphones are connected to the Nexys 4 board through 6-wire ribbon cables. By a stroke of luck the pinout for the microphone exactly matches the general purpose IO ports on the Nexys 4. The 3V3 and GND pins are located at the same positions so the microphones can be plugged directly into the board without any additional components. The pair of microphones for the X and Y axis plug into ports A and B respectively. Note that the red wire on each of the ribbon cables corresponds to the 3V3 pin on the microphones and the Nexys 4.



The Nexys 4 communicates with the Virtex 2 labkit through UART and they are physically connected by the red wire in the picture below. Additionally, the labkit is also connected to the camera through an RCA connector and the computer monitor through a VGA connector. Because of the space requirements of our project the camera and microphones are set facing an empty section of the lab.



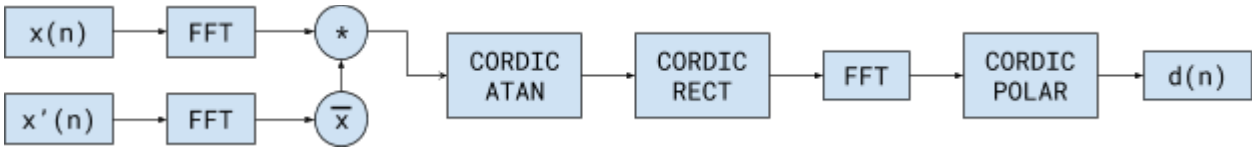
Algorithm

For delay profile extraction we make use of the generalized cross-correlation phase transform (GCC-PHAT) method. GCC-PHAT is a robust and computationally efficient method of calculating the correlation of two signals. It is a well known fact that convolution in the time domain can be represented as multiplication in the frequency domain. The “phase transform” portion of the algorithm normalizes the contributions of each frequency component and ensures that if two signals are identical except for a relative delay, the output is an impulse function centered around that delay. As each frequency component contributes equally to the final delay profile, sounds with a broad frequency spectrum (such as white noise) will be more reliable at triggering a detection. Conversely, sounds with a narrow frequency spectrum such as a whistle will produce a weak correlation signal. This can also be understood in terms of the degeneracy in the possible locations of a sinusoidal sound source.

Mathematically, given two input samples $x(n)$ and $x'(n)$, the time delay profile is given by $d(n) = |g(n)|$ where

$$G(f) = \frac{X(f)X'(f)^*}{\left|X(f)X'(f)^*\right|} .$$

In the above expression, $X(f)$, $X'(f)$, and $G(f)$ denote the discrete fourier transform (DFT) of $x(n)$, $x'(n)$, and $g(n)$ respectively. A block diagram of the GCC-PHAT data processing pathway is presented below.

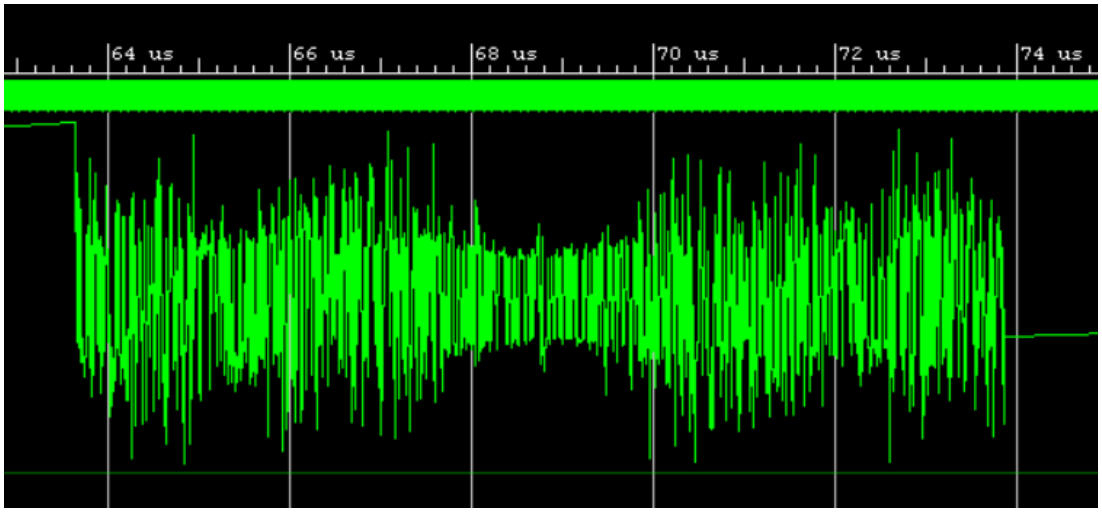


Normalization is done by first extracting the phase of the product $X(f)X'(f)^*$ and calculating the coordinates of a unit vector with that phase. This is computationally simpler than dividing by the modulus of the product, as it avoids the need to take a reciprocal square root.

The delay profile can be translated into an angular spectrum through a small angle approximation. The relation is given by $\theta = \delta c/d$ where d is the baseline between the pair of microphones.

Dithering

An unexpected phenomenon that showed up over the course of our testing is a persistent common mode noise in the form of a peak at $t = 0$. It appears to be that rounding errors in the digital signal processing chain cause the phase of the product $X(f)X'(f)^*$ to be correlated when the incoming signal is weak. It tends to zero rather than being uniformly distributed over $(-\pi, \pi]$. The following waveform of the phase for a set of randomly generated inputs demonstrates the phenomenon.



It is unclear if this rounding error is the only source of the common mode noise. It appears that the peak at $t = 0$ is more prominent in actual measurements than in simulations of randomly generated inputs. One possible explanation is that fluctuations in the power rail voltage could generate such correlated noise, although we have not had the opportunity to test this hypothesis.

Whatever the cause of the phenomenon, we have found that by injecting uncorrelated noise into the system during preprocessing we can wash out the common mode noise and generate a more robust localization signal. This technique of intentionally introducing noise to a signal to improve performance is known as dithering. The switches on the Nexys 4 select the level of dithering to apply to the signal. Empirically, 7 bits of dithering give the optimal balance between signal fidelity and noise rejection.

The source of entropy for our dithering is a 16-bit linear congruential generator. The multiplier and increment of the generator are selected such that all 2^{16} states are visited every period. Linear congruential generators of modulus 2^n are well suited to digital synthesis they utilize only multiplication and addition operations and can run in a single clock cycle.

Resolution and throughput

We sample our microphones at a frequency of 65.104 kHz. This is above the manufacturer's stated maximum of 64 kHz but it does not appear to impair the functionality of the microphone. The data is processed in batches of 1024 samples giving us a refresh rate of about 63 FPS. Data processing is buffered and pipelined such that few samples are dropped in between frames. The time resolution of the delay profile is the reciprocal of the sampling frequency. This works out to be about 15 us for our setup, in which a sound wave travels about 5.1 mm. As such, 5.1 mm is the theoretical limit on the resolution of our path difference. The X and Y axis have a baseline of 40" and 32" respectively. With this we estimate the angular resolution limit of our system to be about 0.29° for the X axis and 0.36° for the Y axis.

Interface

The backend system transmits the delay profile for the X and Y axis through UART at a rate of 1 Mbaud. The messages are encoded in ASCII to allow for ease of debugging and to make it straightforward to identify the start and end of each data frame. 256 samples of one byte each is transmitted for each axis, starting with the X axis and followed with the Y axis. Each sample represents the correlation of the signal

corresponding to a particular delay, where the index of the sample should be interpreted as an 8-bit signed value with a time step that is the reciprocal of the sampling frequency. Each sample is written as two hexadecimal characters. Presented below is an example of a data frame.

```
"FF0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000FF0000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000\n"
```

The above data frame represents a peak of magnitude 0xFF at the origin (X = 0 and Y = 0). Note that the peak occurs at the first index of the message even though it is in the middle of our angular range. This is due to the signed encoding of the delay coordinates.

Backend Implementation (Francis)

The key challenge to the digital signal processing is in configuring Vivado's IP cores and interfacing with them over the AXI4-Stream protocol. The AXI4-Stream protocol enforces flow control through a set of control signals, and by obeying certain rules of composition AXI4-Stream protocol compliant modules can be chained arbitrarily to carry out complex computations. Vivado provides a block diagram based interface (IP Integrator) to aid in the generation of such systems. However, it quickly became apparent that with the complexity of the GCC-PHAT algorithm it was much easier and clearer to simply write everything in verilog. The widths of the data busses and the internal precision of the IP cores demanded careful thought. Data busses had to be sufficiently wide that the computations do not overflow even in the worst case, and computations had to be sufficiently precise that rounding errors do not wash out the signal. However, the busses cannot be made arbitrarily wide as the computational resources of the FPGA are limited so a balance had to be struck between resource

utilization and precision. I will comment on this when pertinent to the discussion of the module at hand. The main system clock is driven at 100 MHz.

i2s_receiver	Generates the clocking signals (BCLK and LRCLK) required to drive the I2S microphones. BCLK is driven at 1/24 the frequency of the main system clock to give a sample rate of 65.104 kHz. Converts the incoming I2S bitstream from serial to parallel and passes it on to the rest of the system. Based on our testing the dynamic range of the microphone is only about 14 bits so we truncate the samples from 18 bits to 16 bits.
led_indicator	Controls the LEDs to give visual confirmation when the microphones are plugged in. LEDs will turn on when the samples are anything besides 0xFFFF.
upstream_hub	Alternates between buffering the incoming audio samples and feeding the samples into the AXI4-Stream compliant *gcc_phat_core*. It works in frames of 1024 samples, which is the length of the DFT used in the GCC-PHAT algorithm.
upstream_bram	Block ram for upstream buffer. It has a bus width of 64 bits to accommodate four channels of samples each 16 bits wide.
gcc_phat_core	Main digital signal processing core. Takes the audio samples and does the necessary computations to extract the delay profiles. At the end of the computation, the 16 bit delay profile is downsampled to 8 bits for transmission. The output is clipped to 0xFF in the case of saturation.
dither_0	Dithering preprocessor for the audio samples. Makes use of a linear congruence generator to inject white noise into the system. The level of dithering is set by switches SW[3:0] on the Nexys 4.
xfft_0	Vivado Fast Fourier Transform core. Four channel DFT utilizing the Radix-2 Burst I/O architecture. Makes use of unscaled arithmetic to pick up on weak frequency domain signals. Bus width is increased from 16 bits to 27 bits through this module.

multiply_0	Computes the product $X(f)X'(f)^*$. Care must be taken to use signed multiplication on the components. Because of the considerable logic depth of the multiplication operation, register slice *slice_0* is used to provide timing isolation. Bus width is increased from 27 bits to 56 bits to prevent arithmetic overflow. Although only 55 bits are required, the last bit is added for byte alignment.
slice_0	Vivado AXI4-Stream Register Slice core. Provides timing isolation for *multiply_0* and *shift_0*. It has a bus width of 28 bytes to accommodate four channels of 7 bytes each.
scale_0	Scales product $X(f)X'(f)^*$ up without changing its phase. 56 bits is much too wide for the CORDIC core so a coarse scaling followed by a downsampling must be done to ensure the precision of the argument extraction in *normalize_0*. Six instances of *shift_0* are connected in series to perform a leftward bitshift of up to 6 bytes. The leading 22 bits are sign extended to 24 bits and passed forwards. The sign extension is necessary to comply with the fix24_22 input format of *cordic_0*.
shift_0	Performs a leftward bitshift of up to 8 bits while maintaining the phase of the input. Makes use of register slice *slice_0* for timing isolation.
normalize_0	Computes normalized product $G(f)$ by first extracting the phase of $X(f)X'(f)^*$ then generating a unit vector with the given phase. Makes use of a pair of Vivado CORDIC cores for the manipulations.
broadcaster_0	Vivado AXI4-Stream Broadcaster core. Splits the X and Y channels so that they can be normalized independently. This is necessary as the CORDIC cores may have variable latency.
cordic_0	Vivado CORDIC core. Makes use of Volder's algorithm to extract the phase of $X(f)X'(f)^*$. Bus width is decreased from 24 bits to 16 bits. According to the datasheet, a precision of (INPUT_W + OUTPUT_W + LOG2 (OUTPUT_W)) is required for the output phase to be accurate to the output width under all conditions, and as such, the internal precision of the core is set

	to 44 bits. The core is fully pipelined for high throughput operation.
cordic_1	Vivado CORIC core. Makes use of Volder's algorithm to generate a unit vector with the given phase. Bus width remains at 16 bits. The core is fully pipelined for high throughput operation.
combiner_0	Vivado AXI4-Stream Combiner core. Combines X and Y channels so that they can be fed simultaneously into *xfft_1*.
xfft_1	Vivado Fast Fourier Transform core. Dual channel inverse DFT utilizing the Radix-2 Burst I/O architecture. Makes use of scaled arithmetic to keep the bus width at 16 bits. Unscaled arithmetic is not necessary at this point as the delicate signal extraction had already been done.
magnitude_0	Extracts the magnitude of $g(n)$ to give the final delay profile. Makes use of the Vivado CORDIC core for the computation. Sign extends the bus from 16 bits to 18 bits to comply with the fix18_16 input format of *cordic_2*.
broadcaster_1	Vivado AXI4-Stream Broadcaster core. Splits the X and Y channels so that the magnitude of each can be computed independently. This is necessary as the CORDIC cores may have variable latency
cordic_2	Vivado CORIC core. Makes use of Volder's algorithm to extract the magnitude of $g(n)$. Bus width is decreased from 18 bits back to 16 bits. The core is fully pipelined for high throughput operation.
combiner_1	Vivado AXI4-Stream Combiner core. Combines X and Y channels so that they can be output simultaneously from *magnitude_0*.
downstream_hub	Buffers delay profiles from *gcc_phat_core* and feeds them into *byte_transmitter* for transmission. Out of the 1024 samples, it keeps only the first and last 128 as these are the samples corresponding to the delay window that we are interested in.

downstream_bram	Block ram for downstream buffer. It has a bus width of 16 bits to accommodate the 8 bit delay profile samples for the X and Y axis.
byte_transmitter	Generates the ASCII characters that make up each data frame and feeds them to *uart_transmitter*.
hex_ascii	Combinational logic module for converting four data bits to its ASCII hexadecimal equivalent.
uart_transmitter	Transmits bytes over the UART protocol at a data rate of 1 Mbaud.

The resource utilization of the design is given in the table below. Surprisingly, despite the 2 FFT and 6 CORDIC cores, only about a quarter of the available LUTs and DSPs are utilized. This indicates that a higher internal precision or a greater number of microphone pairs can be accommodated comfortably by the hardware.

Resource	Utilization	Available	Utilization %
LUT	17719	63400	27.95
LUTRAM	927	19000	4.88
FF	17326	126800	13.66
BRAM	13.50	135	10.00
DSP	56	240	23.33
IO	38	210	18.10
BUFG	1	32	3.13

At a sampling frequency of 65.104 kHz, the duration between each sample is about 15.4 us and the duration between each 1024 sample data frame is about 15.7 ms. Based on simulation results, the upstream FFT module take about 73.4 us to process each data frame and is the rate limiting step of *gcc_phat_core*. This is much faster than the frame rate of the system, so much so that the FFT module spends about 99.5% of its time idle. Furthermore, it takes about 10.2 us to load the samples from the upstream buffer into *gcc_phat_core*. This is faster than the sampling frequency, and as such, in principle no samples are dropped in between frames.

The implemented design has a worst negative slack (WNS) of 0.274 ns. The computation that contributes to this figure is the product $X(f)X'(f)^*$. This is justification for the timing isolation registers *slice_0* and *slice_1*. Without them the design will not be able to achieve timing closure.

Frontend Implementation (Keshav)

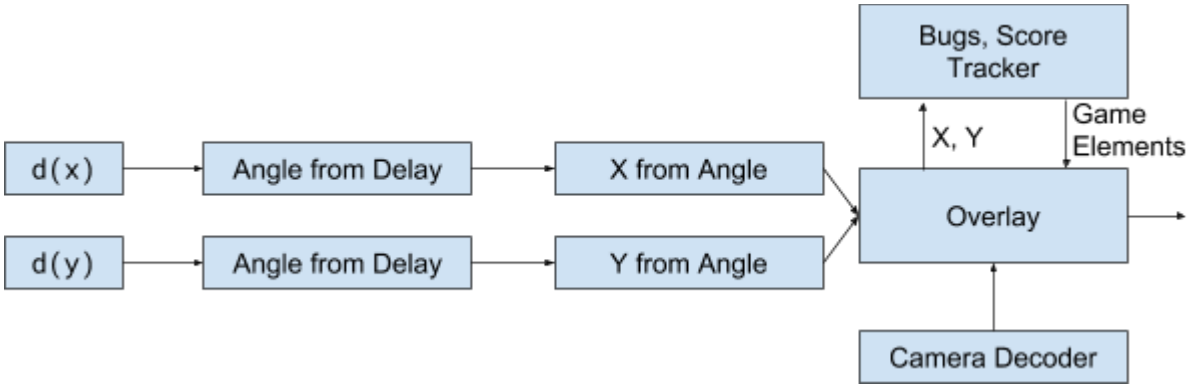
The frontend can be broken down functionally into three parts, Camera Decoder, Overlay, and Controller (which also implements the bug squashing game).

Camera Decoder:
This module uses example code provided generously by 6.111 staff, modified to implement color display, to center the frame and introduce a black letterbox frame to hide the white noise. To implement color functionality, the ZBT now stores Y, Cr, Cb sampled down to 6 bits each for 2 pixels at a time, instead of 8 bits of Y for 4 pixels.

Overlay:
As the data frame is being received as collection of 1024 bytes (256 x 2 bytes for each channel) over UART, we find the maximum, i.e. location of the peak and only this information is stored. The remaining information is discarded. Once the x and y indices of the peak are found, they are offset and scaled by set constants that can be changed through the buttons and switches on the front panel on the labkit. Where the peak of the received data represented the delay δ, after scaling it represents $\frac{\delta c}{d}$, the predicted angle of the origin of sound w.r.t optical axis of the camera (c being the speed of sound in air, d being the baseline i.e. separation between the microphones). This can be further scaled by the angular resolution of the camera to get $\frac{\delta c}{fd}$, the approximate x and the y coordinates on screen. All of $\frac{c}{fd}$ can be treated as a single constant that can be measured experimentally (in our case, 0.78125 and 1.5625 for the x and the y axis respectively). A box is then drawn to indicate the calculated coordinates. These constants are set by setting the switches to (desired value * 64) and then asserting buttons 0, 1, 2 for peak detection threshold, xoffset and yoffset respectively.

Controller:
Functions of this block include: a) Instantiate some “bug” objects and get display data from them b) Get VGA timing signals from the xvga module and distribute them to other modules c) Keep track of score and display it.

The functional block diagram is as follows:



The exact modules names and their descriptions are as listed below:

zbt_6111_sample	The central labkit module. Has the usual labkit IO configuration.
ramclock	Deskews clocks for the ZBT SRAMs
debounce	Debounces switch inputs
display_16hex	Displays information on the 8x5x16 LED Displays on the Labkit. used for debugging (in this case, displays the xoffset, yoffset, xscale, yscale and threshold)
xvga	Generates hcount, vcount and sync signals for VGA display
zbt_6111	Wrapper/Driver for the onboard ZBT memory
vram_display	Reads the onboard ZBT and generates RGB pixel data (modified from the example code to center the image, display black border around non-image area, and display color instead of black and white)
translate_hvcount	Translates the given hcount and vcount values by a constant offset. Used to center the vram_display output
YCrCb2RGB	Converts Y, Cb, Cr signals from vram_display to RGB
adv7185_init	Initializes the onboard ADV7185 chip
ntsc_decode	Decodes input from ADV7185 into Y, Cr, Cb data for storage onto the ZBT memory
ntsc_to_zbt	Stores video data on the ZBT
overlay	The overall module that generates overlays (a box around the estimated x and y coordinates) after scaling and offset

uart_receive	Receive data from the Nexys 4 over UART at 1 Mbaud
clock_divider	Waits for the START BIT and generates clock pulses at 1 MHz
ascii2hex	Converts two ASCII values to one 8 bit value
*bug**	Generate the overlays for different bugs. Assign random x and y position on reset, and then simulates movements with a random acceleration. Randomness generated with a Linear congruential generator
score_tracker	Generates the overlay for “Score: xxx” display
hexfont	Block ROM that stores font information

The initial plan was to use a Block ROM for the font information for the entire ASCII table, however since the text associated with the bugs is fixed, we decided to used fixed signals as constant busses. We still use a block ROM for the hex digits in the score display. This results in the following resource usage:

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Flip Flops	1,168	67,584	1%
Number of 4 input LUTs	1,906	67,584	2%
Number of occupied Slices	1,355	33,792	4%
Number of Slices containing only related logic	1,355	1,355	100%
Number of Slices containing unrelated logic	0	1,355	0%
Total Number of 4 input LUTs	2,261	67,584	3%
Number used as logic	1,829		
Number used as a route-thru	355		
Number used as Shift registers	77		
Number of bonded IOBs	577	684	84%
IOB Flip Flops	6		
Number of RAMB16s	2	144	1%
Number of MULT18X18s	10	144	6%
Number of BUFGMUXs	5	16	31%
Number of DCMs	3	12	25%

All source code for the implementation (backend and frontend) can be found in the appendix.

Results and discussion

In this project we have demonstrated the feasibility of accurate audio localization through the path delay it introduces to spatially separated microphones. The GCC-PHAT algorithm has provided a robust and effective means of extracting this delay by operating on the audio samples in the frequency domain. The Artix-7 FPGA on the Nexys-4 board was well suited to this task. Only about a quarter of the available LUTs ad DSPs were utilized and data frames were processed much faster than they were generated. The system has low latency, with the delay dominated by the 60 Hz refresh rate of the monitor.

Audio clips played from a smartphone were observed to be the most reliable way of triggering a detection. This is likely due to their sustained amplitude and broad frequency spectrum. With such a sound source we can achieve a localization precision close to the resolution limit of our system and movement on the centimeter scale at a distance of a few meters can clearly be discerned. Human voice is also fairly reliable at triggering detections while claps and snaps are localized with a greater than 75% success rate. Occasionally, for short noises such as claps and snaps the localization is far off the mark. We suspect that this could be due to interference from echos bouncing off the floor or the ceiling. Such indirect path signal interference would be difficult to mitigate without more sophisticated signal processing. The system has better resolution in the horizontal direction than the vertical direction as the baseline in the X axis is 25% longer than that in the Y axis.

Appendix
Backend Source Code

byte_transmitter.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/17/2018 10:15:10 AM
// Design Name:
// Module Name: byte_transmitter
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module byte_transmitter(
    input clock,
    input [7:0] byte_data,
    input byte_start,
    output byte_ready,
    output reg [7:0] uart_data,
    output reg uart_start,
    input uart_ready
    );

    parameter STATE_LOAD = 0;
    parameter STATE_UNLOAD_0 = 1;
    parameter STATE_UNLOAD_1 = 2;
    parameter STATE_NEW_LINE = 3;

    reg [1:0] state;
    reg [7:0] data;
    reg [8:0] index;

    wire [7:0] ascii_0;
    wire [7:0] ascii_1;
```

```
hex_ascii hex_ascii_0(
    .hex(data[7:4]),
    .ascii(ascii_0)
);
hex_ascii hex_ascii_1(
    .hex(data[3:0]),
    .ascii(ascii_1)
);

assign byte_ready = (state == STATE_LOAD);

initial begin
    state = STATE_LOAD;
    index = 0;
end

always @(posedge clock) begin
    case (state)
        STATE_LOAD: begin
            uart_start <= 0;
            if (byte_start) begin
                data <= byte_data;
                state <= STATE_UNLOAD_0;
            end
        end
        STATE_UNLOAD_0: begin
            if (uart_ready) begin
                uart_data <= ascii_0;
                uart_start <= 1;
                state <= STATE_UNLOAD_1;
            end
        end
        STATE_UNLOAD_1: begin
            if (uart_ready) begin
                uart_data <= ascii_1;
                uart_start <= 1;
                index <= index+1;
                if (index != {9{1'b1}}) begin
                    state <= STATE_LOAD;
                end
                else begin
                    state <= STATE_NEW_LINE;
                end
            end
            else begin
                uart_start <= 0;
            end
```

```
            end
            STATE_NEW_LINE: begin
                if (uart_ready) begin
                    uart_data <= "\n";
                    uart_start <= 1;
                    state <= STATE_LOAD;
                end
                else begin
                    uart_start <= 0;
                end
            end
        endcase
    end

endmodule
```

dither_0.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/28/2018 11:16:13 PM
// Design Name:
// Module Name: dither_0
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module dither_0(
    input aclk,
    input [3:0] dither,
    input [63:0] s_axis_tdata,
    input s_axis_tvalid,
    output s_axis_tready,
    output [63:0] m_axis_tdata,
    output m_axis_tvalid,
    input m_axis_tready
```

```
    );

    parameter INC_0 = 5319;
    parameter MUL_0 = 9721;
    parameter INC_1 = 11017;
    parameter MUL_1 = 6997;

    reg [15:0] rand [1:0];
    wire [15:0] noise [1:0][15:0];
    wire [15:0] signal [3:0];

    genvar i;

    generate
        for (i = 0; i < 16; i = i+1) begin
            assign noise[0][i] = rand[0][15:15-i];
            assign noise[1][i] = rand[1][15:15-i];
        end
    endgenerate

    assign signal[3] = s_axis_tdata[63:48] + noise[1][dither];
    assign signal[2] = s_axis_tdata[47:32] + noise[0][dither];
    assign signal[1] = s_axis_tdata[31:16] + noise[1][dither];
    assign signal[0] = s_axis_tdata[15:0] + noise[0][dither];

    initial begin
        rand[0] = 0;
        rand[1] = 0;
    end

    always @(posedge aclk) begin
        if (s_axis_tvalid && s_axis_tready) begin
            rand[0] <= MUL_0 * rand[0] + INC_0;
            rand[1] <= MUL_1 * rand[1] + INC_1;
        end
    end

    assign m_axis_tdata = {
        signal[3],
        signal[2],
        signal[1],
        signal[0]
    };
    assign m_axis_tvalid = s_axis_tvalid;
    assign s_axis_tready = m_axis_tready;

endmodule
```

downstream_bram.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/17/2018 10:02:55 AM
// Design Name:
// Module Name: downstream_bram
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module downstream_bram(
    input clock,
    input [7:0] addr,
    input [15:0] din,
    output reg [15:0] dout,
    input we,
    output stable
    );

    (* ram_style = "block" *)
    reg [15:0] bram [(1<<8)-1:0];

    reg [7:0] last;

    assign stable = (last == addr);

    always @(posedge clock) begin
        if (we) bram[addr] <= din;
        dout <= bram[addr];
        last <= addr;
    end

endmodule
```

downstream_hub.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/17/2018 12:48:01 AM
// Design Name:
// Module Name: downstream_hub
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module downstream_hub(
    input clock,
    input [15:0] s_axis_out_tdata,
    input s_axis_out_tvalid,
    output s_axis_out_tready,
    output reg [7:0] byte_data,
    output reg byte_start,
    input byte_ready
    );

    parameter STATE_LOAD = 0;
    parameter STATE_UNLOAD = 1;

    reg state;
    reg [9:0] index;
    wire [15:0] dout;
    wire we;
    wire stable;

    downstream_bram downstream_bram(
        .clock(clock),
        .addr(index[7:0]),
        .din(s_axis_out_tdata),
        .dout(dout),
        .we(we),
        .stable(stable)
```

```
);

assign s_axis_out_tready = (state == STATE_LOAD);
assign we = (state == STATE_LOAD) && s_axis_out_tvalid
    && ((index[9:7] == 0) || (index[9:7] == {3{1'b1}}));

initial begin
    state = STATE_LOAD;
    index = 0;
    byte_start = 0;
end

always @(posedge clock) begin
    case (state)
        STATE_LOAD: begin
            byte_start <= 0;
            if (s_axis_out_tvalid) begin
                index <= index+1;
                if (index == {10{1'b1}}) begin
                    state <= STATE_UNLOAD;
                end
            end
        end
        STATE_UNLOAD: begin
            if (stable && byte_ready) begin
                byte_start <= 1;
                if (index[8]) begin
                    byte_data <= dout[15:8];
                end
                else begin
                    byte_data <= dout[7:0];
                end

                if (index != {9{1'b1}}) begin
                    index <= index+1;
                end
                else begin
                    index <= 0;
                    state <= STATE_LOAD;
                end
            end
            else begin
                byte_start <= 0;
            end
        end
    endcase
end
```

```
endmodule
```

gcc_phat_core.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/17/2018 12:44:50 AM
// Design Name:
// Module Name: gcc_phat_core
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module gcc_phat_core(
        input clock,
        input [3:0] dither,
        input [63:0] s_axis_in_tdata,
        input s_axis_in_tvalid,
        output s_axis_in_tready,
        output [15:0] m_axis_out_tdata,
        output m_axis_out_tvalid,
        input m_axis_out_tready
    );

    wire [63:0] axis_dither_0_tdata;
    wire axis_dither_0_tvalid;
    wire axis_dither_0_tready;

    dither_0 dither_0(
        .aclk(clock),
        .dither(dither),
        .s_axis_tdata(s_axis_in_tdata),
        .s_axis_tvalid(s_axis_in_tvalid),
        .s_axis_tready(s_axis_in_tready),
        .m_axis_tdata(axis_dither_0_tdata),
        .m_axis_tvalid(axis_dither_0_tvalid),
```

```
    .m_axis_tready(axis_dither_0_tready)
);

wire [255:0] axis_xfft_0_tdata;
wire axis_xfft_0_tvalid;
wire axis_xfft_0_tready;

xfft_0 xfft_0(
    .aclk(clock),
    .s_axis_data_tdata({
        16'b0, axis_dither_0_tdata[63:48],
        16'b0, axis_dither_0_tdata[47:32],
        16'b0, axis_dither_0_tdata[31:16],
        16'b0, axis_dither_0_tdata[15:0]
    }),
    .s_axis_data_tvalid(axis_dither_0_tvalid),
    .s_axis_data_tready(axis_dither_0_tready),
    .s_axis_data_tlast(0),
    .s_axis_config_tdata(16'h000F),
    .s_axis_config_tvalid(1),
    .m_axis_data_tdata(axis_xfft_0_tdata),
    .m_axis_data_tvalid(axis_xfft_0_tvalid),
    .m_axis_data_tready(axis_xfft_0_tready)
);

wire [223:0] axis_multiply_0_tdata;
wire axis_multiply_0_tvalid;
wire axis_multiply_0_tready;

multiply_0 multiply_0(
    .aclk(clock),
    .s_axis_tdata(axis_xfft_0_tdata),
    .s_axis_tvalid(axis_xfft_0_tvalid),
    .s_axis_tready(axis_xfft_0_tready),
    .m_axis_tdata(axis_multiply_0_tdata),
    .m_axis_tvalid(axis_multiply_0_tvalid),
    .m_axis_tready(axis_multiply_0_tready)
);

wire [95:0] axis_scale_0_tdata;
wire axis_scale_0_tvalid;
wire axis_scale_0_tready;

scale_0 scale_0(
    .aclk(clock),
    .s_axis_tdata(axis_multiply_0_tdata),
    .s_axis_tvalid(axis_multiply_0_tvalid),
    .s_axis_tready(axis_multiply_0_tready),
```

```
    .m_axis_tdata(axis_scale_0_tdata),
    .m_axis_tvalid(axis_scale_0_tvalid),
    .m_axis_tready(axis_scale_0_tready)
);

wire [63:0] axis_normalize_0_tdata;
wire axis_normalize_0_tvalid;
wire axis_normalize_0_tready;

normalize_0 normalize_0(
    .aclk(clock),
    .s_axis_tdata(axis_scale_0_tdata),
    .s_axis_tvalid(axis_scale_0_tvalid),
    .s_axis_tready(axis_scale_0_tready),
    .m_axis_tdata(axis_normalize_0_tdata),
    .m_axis_tvalid(axis_normalize_0_tvalid),
    .m_axis_tready(axis_normalize_0_tready)
);

wire [63:0] axis_xfft_1_tdata;
wire axis_xfft_1_tvalid;
wire axis_xfft_1_tready;

xfft_1 xfft_1(
    .aclk(clock),
    .s_axis_data_tdata(axis_normalize_0_tdata),
    .s_axis_data_tvalid(axis_normalize_0_tvalid),
    .s_axis_data_tready(axis_normalize_0_tready),
    .s_axis_data_tlast(0),
    .s_axis_config_tdata({6'b0, {20{2'b01}}, 2'b0}),
    .s_axis_config_tvalid(1),
    .m_axis_data_tdata(axis_xfft_1_tdata),
    .m_axis_data_tvalid(axis_xfft_1_tvalid),
    .m_axis_data_tready(axis_xfft_1_tready)
);

wire [31:0] axis_magnitude_0_tdata;
wire axis_magnitude_0_tvalid;
wire axis_magnitude_0_tready;

magnitude_0 magnitude_0(
    .aclk(clock),
    .s_axis_tdata(axis_xfft_1_tdata),
    .s_axis_tvalid(axis_xfft_1_tvalid),
    .s_axis_tready(axis_xfft_1_tready),
    .m_axis_tdata(axis_magnitude_0_tdata),
    .m_axis_tvalid(axis_magnitude_0_tvalid),
    .m_axis_tready(axis_magnitude_0_tready)
```

```
    );

    wire [15:0] delay_raw [1:0];
    wire [7:0] delay_profile[1:0];

    assign delay_raw[1] = axis_magnitude_0_tdata[31:16];
    assign delay_raw[0] = axis_magnitude_0_tdata[15:0];
    assign delay_profile[1] = (delay_raw[1] > 16'hFFF) ? 8'hFF : delay_raw[1][11:4];
    assign delay_profile[0] = (delay_raw[0] > 16'hFFF) ? 8'hFF : delay_raw[0][11:4];

    assign m_axis_out_tdata = {delay_profile[1], delay_profile[0]};
    assign m_axis_out_tvalid = axis_magnitude_0_tvalid;
    assign axis_magnitude_0_tready = m_axis_out_tready;

endmodule
```

hex_ascii.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/15/2018 03:25:58 PM
// Design Name:
// Module Name: hex_ascii
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module hex_ascii(
    input [3:0] hex,
    output reg [7:0] ascii
    );

    always @(*) begin
        case (hex)
            4'h0: ascii = "0";
            4'h1: ascii = "1";
```

```
            4'h2: ascii = "2";
            4'h3: ascii = "3";
            4'h4: ascii = "4";
            4'h5: ascii = "5";
            4'h6: ascii = "6";
            4'h7: ascii = "7";
            4'h8: ascii = "8";
            4'h9: ascii = "9";
            4'hA: ascii = "A";
            4'hB: ascii = "B";
            4'hC: ascii = "C";
            4'hD: ascii = "D";
            4'hE: ascii = "E";
            4'hF: ascii = "F";
        endcase
    end

endmodule
```

i2s_receiver.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/15/2018 03:20:16 PM
// Design Name:
// Module Name: i2s_receiver
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module i2s_receiver(
    input clock,
    output i2s_bclk,
    output i2s_lrclk,
    input [3:0] i2s_din,
    output reg i2s_ready,
```

```
    output reg [63:0] i2s_data
    );

    parameter DIVISOR = 12;

    reg [4:0] clock_count;
    reg [6:0] bit_count;

    initial begin
        i2s_ready = 0;
        clock_count = 0;
        bit_count = 0;
    end

    assign i2s_bclk = bit_count[0];
    assign i2s_lrclk = bit_count[6];

    always @(posedge clock) begin
        if (clock_count == DIVISOR-1) begin
            clock_count <= 0;
            bit_count <= bit_count+1;
            if (!i2s_bclk) begin
                i2s_data <= {
                    i2s_data[62:48], i2s_din[3],
                    i2s_data[46:32], i2s_din[2],
                    i2s_data[30:16], i2s_din[1],
                    i2s_data[14:0], i2s_din[0]
                };
            end
            if (bit_count == 36) begin
                i2s_ready <= 1;
            end
        end
        else begin
            clock_count <= clock_count+1;
            i2s_ready <= 0;
        end
    end

endmodule
```

labkit.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/15/2018 03:17:35 PM
```

```
// Design Name:
// Module Name: labkit
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module labkit(
    input CLOCK,
    output [3:0] I2S_BCLK,
    output [3:0] I2S_LRCLK,
    output [3:0] I2S_SEL,
    input [3:0] I2S_DIN,
    output UART_DOUT,
    output [15:0] LED,
    input [3:0] SW
    );

    wire i2s_bclk;
    wire i2s_lrclk;
    wire i2s_ready;
    wire [63:0] i2s_data;

    assign I2S_BCLK = {4{i2s_bclk}};
    assign I2S_LRCLK = {4{i2s_lrclk}};
    assign I2S_SEL = {4{1'b0}};

    i2s_receiver i2s_receiver(
        .clock(CLOCK),
        .i2s_bclk(i2s_bclk),
        .i2s_lrclk(i2s_lrclk),
        .i2s_din(I2S_DIN),
        .i2s_ready(i2s_ready),
        .i2s_data(i2s_data)
    );

    led_indicator led_indicator(
        .clock(CLOCK),
        .i2s_ready(i2s_ready),
```

```
    .i2s_data(i2s_data),
    .led(LED)
);

wire [63:0] axis_in_tdata;
wire axis_in_tvalid;
wire axis_in_tready;

upstream_hub upstream_hub(
    .clock(CLOCK),
    .i2s_ready(i2s_ready),
    .i2s_data(i2s_data),
    .m_axis_in_tdata(axis_in_tdata),
    .m_axis_in_tvalid(axis_in_tvalid),
    .m_axis_in_tready(axis_in_tready)
);

wire [15:0] axis_out_tdata;
wire axis_out_tvalid;
wire axis_out_tready;

gcc_phat_core gcc_phat_core(
    .clock(CLOCK),
    .dither(SW),
    .s_axis_in_tdata(axis_in_tdata),
    .s_axis_in_tvalid(axis_in_tvalid),
    .s_axis_in_tready(axis_in_tready),
    .m_axis_out_tdata(axis_out_tdata),
    .m_axis_out_tvalid(axis_out_tvalid),
    .m_axis_out_tready(axis_out_tready)
);

wire [7:0] byte_data;
wire byte_start;
wire byte_ready;

downstream_hub downstream_hub(
    .clock(CLOCK),
    .s_axis_out_tdata(axis_out_tdata),
    .s_axis_out_tvalid(axis_out_tvalid),
    .s_axis_out_tready(axis_out_tready),
    .byte_data(byte_data),
    .byte_start(byte_start),
    .byte_ready(byte_ready)
);

wire [7:0] uart_data;
wire uart_start;
```

```
    wire uart_ready;

    byte_transmitter byte_transmitter(
        .clock(CLOCK),
        .byte_data(byte_data),
        .byte_start(byte_start),
        .byte_ready(byte_ready),
        .uart_data(uart_data),
        .uart_start(uart_start),
        .uart_ready(uart_ready)
    );

    uart_transmitter uart_transmitter(
        .clock(CLOCK),
        .uart_data(uart_data),
        .uart_start(uart_start),
        .uart_ready(uart_ready),
        .uart_dout(UART_DOUT)
    );

endmodule
```

led_indicator.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/16/2018 12:44:33 PM
// Design Name:
// Module Name: led_indicator
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module led_indicator(
    input clock,
    input i2s_ready,
```

```
    input [63:0] i2s_data,
    output [15:0] led
    );

    reg [3:0] active;

    assign led = {
        {4{active[3]}},
        {4{active[2]}},
        {4{active[1]}},
        {4{active[0]}}
    };

    always @(posedge clock) begin
        if (i2s_ready) begin
            active[3] <= (i2s_data[63:48] != {16{1'b1}});
            active[2] <= (i2s_data[47:32] != {16{1'b1}});
            active[1] <= (i2s_data[31:16] != {16{1'b1}});
            active[0] <= (i2s_data[15:0] != {16{1'b1}});
        end
    end

endmodule
```

magnitude_0.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/21/2018 06:05:17 PM
// Design Name:
// Module Name: magnitude_0
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module magnitude_0(
```

```
input aclk,
input [63:0] s_axis_tdata,
input s_axis_tvalid,
output s_axis_tready,
output [31:0] m_axis_tdata,
output m_axis_tvalid,
input m_axis_tready
);

wire [31:0] axis_broadcaster_1_tdata [1:0];
wire [1:0] axis_broadcaster_1_tvalid;
wire [1:0] axis_broadcaster_1_tready;

broadcaster_1 broadcaster_1(
    .aclk(aclk),
    .aresetn(1),
    .s_axis_tdata(s_axis_tdata),
    .s_axis_tvalid(s_axis_tvalid),
    .s_axis_tready(s_axis_tready),
    .m_axis_tdata({
        axis_broadcaster_1_tdata[1],
        axis_broadcaster_1_tdata[0]
    }),
    .m_axis_tvalid(axis_broadcaster_1_tvalid),
    .m_axis_tready(axis_broadcaster_1_tready)
);

wire [31:0] axis_cordic_2_tdata [1:0];
wire [1:0] axis_cordic_2_tvalid;
wire [1:0] axis_cordic_2_tready;

cordic_2 cordic_2_0(
    .aclk(aclk),
    .s_axis_cartesian_tdata({
        6'b0,
        {2{axis_broadcaster_1_tdata[0][31]}},
        axis_broadcaster_1_tdata[0][31:16],
        6'b0,
        {2{axis_broadcaster_1_tdata[0][15]}},
        axis_broadcaster_1_tdata[0][15:0]
    }),
    .s_axis_cartesian_tvalid(axis_broadcaster_1_tvalid[0]),
    .s_axis_cartesian_tready(axis_broadcaster_1_tready[0]),
    .m_axis_dout_tdata(axis_cordic_2_tdata[0]),
    .m_axis_dout_tvalid(axis_cordic_2_tvalid[0]),
    .m_axis_dout_tready(axis_cordic_2_tready[0])
);
```

```
    cordic_2 cordic_2_1(
        .aclk(aclk),
        .s_axis_cartesian_tdata({
            6'b0,
            {2{axis_broadcaster_1_tdata[1][31]}},
            axis_broadcaster_1_tdata[1][31:16],
            6'b0,
            {2{axis_broadcaster_1_tdata[1][15]}},
            axis_broadcaster_1_tdata[1][15:0]
        }),
        .s_axis_cartesian_tvalid(axis_broadcaster_1_tvalid[1]),
        .s_axis_cartesian_tready(axis_broadcaster_1_tready[1]),
        .m_axis_dout_tdata(axis_cordic_2_tdata[1]),
        .m_axis_dout_tvalid(axis_cordic_2_tvalid[1]),
        .m_axis_dout_tready(axis_cordic_2_tready[1])
    );

    combiner_1 combiner_1(
        .aclk(aclk),
        .aresetn(1),
        .s_axis_tdata({
            axis_cordic_2_tdata[1][15:0],
            axis_cordic_2_tdata[0][15:0]
        }),
        .s_axis_tvalid(axis_cordic_2_tvalid),
        .s_axis_tready(axis_cordic_2_tready),
        .m_axis_tdata(m_axis_tdata),
        .m_axis_tvalid(m_axis_tvalid),
        .m_axis_tready(m_axis_tready)
    );

endmodule
```

multiply_0.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/21/2018 06:28:40 PM
// Design Name:
// Module Name: multiply_0
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
```

```
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module multiply_0(
    input aclk,
    input [255:0] s_axis_tdata,
    input s_axis_tvalid,
    output s_axis_tready,
    output [223:0] m_axis_tdata,
    output m_axis_tvalid,
    input m_axis_tready
    );

    wire signed [26:0] freq_re [3:0];
    wire signed [26:0] freq_im [3:0];

    assign freq_re[3] = s_axis_tdata[218:192];
    assign freq_re[2] = s_axis_tdata[154:128];
    assign freq_re[1] = s_axis_tdata[90:64];
    assign freq_re[0] = s_axis_tdata[26:0];

    assign freq_im[3] = s_axis_tdata[250:224];
    assign freq_im[2] = s_axis_tdata[186:160];
    assign freq_im[1] = s_axis_tdata[122:96];
    assign freq_im[0] = s_axis_tdata[58:32];

    wire signed [55:0] conj_terms [7:0];
    wire signed [55:0] conj_product [3:0];

    assign conj_terms[7] = freq_im[2] * freq_re[3];
    assign conj_terms[6] = freq_re[2] * freq_im[3];
    assign conj_terms[5] = freq_re[2] * freq_re[3];
    assign conj_terms[4] = freq_im[2] * freq_im[3];
    assign conj_terms[3] = freq_im[0] * freq_re[1];
    assign conj_terms[2] = freq_re[0] * freq_im[1];
    assign conj_terms[1] = freq_re[0] * freq_re[1];
    assign conj_terms[0] = freq_im[0] * freq_im[1];

    assign conj_product[3] = conj_terms[7] - conj_terms[6];
    assign conj_product[2] = conj_terms[5] + conj_terms[4];
    assign conj_product[1] = conj_terms[3] - conj_terms[2];
    assign conj_product[0] = conj_terms[1] + conj_terms[0];
```

```
    slice_0 slice_0(
        .aclk(aclk),
        .aresetn(1),
        .s_axis_tdata({
            conj_product[3],
            conj_product[2],
            conj_product[1],
            conj_product[0]
        }),
        .s_axis_tvalid(s_axis_tvalid),
        .s_axis_tready(s_axis_tready),
        .m_axis_tdata(m_axis_tdata),
        .m_axis_tvalid(m_axis_tvalid),
        .m_axis_tready(m_axis_tready)
    );
endmodule

normalize_0.v
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/21/2018 06:18:10 PM
// Design Name:
// Module Name: normalize_0
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module normalize_0(
    input aclk,
    input [95:0] s_axis_tdata,
    input s_axis_tvalid,
    output s_axis_tready,
    output [63:0] m_axis_tdata,
    output m_axis_tvalid,
    input m_axis_tready
```

```
);

wire [47:0] axis_broadcaster_0_tdata [1:0];
wire [1:0] axis_broadcaster_0_tvalid;
wire [1:0] axis_broadcaster_0_tready;

broadcaster_0 broadcaster_0(
    .aclk(aclk),
    .aresetn(1),
    .s_axis_tdata(s_axis_tdata),
    .s_axis_tvalid(s_axis_tvalid),
    .s_axis_tready(s_axis_tready),
    .m_axis_tdata({
        axis_broadcaster_0_tdata[1],
        axis_broadcaster_0_tdata[0]
    }),
    .m_axis_tvalid(axis_broadcaster_0_tvalid),
    .m_axis_tready(axis_broadcaster_0_tready)
);

wire [15:0] axis_cordic_0_tdata [1:0];
wire [1:0] axis_cordic_0_tvalid;
wire [1:0] axis_cordic_0_tready;

cordic_0 cordic_0_0(
    .aclk(aclk),
    .s_axis_cartesian_tdata(axis_broadcaster_0_tdata[0]),
    .s_axis_cartesian_tvalid(axis_broadcaster_0_tvalid[0]),
    .s_axis_cartesian_tready(axis_broadcaster_0_tready[0]),
    .m_axis_dout_tdata(axis_cordic_0_tdata[0]),
    .m_axis_dout_tvalid(axis_cordic_0_tvalid[0]),
    .m_axis_dout_tready(axis_cordic_0_tready[0])
);

cordic_0 cordic_0_1(
    .aclk(aclk),
    .s_axis_cartesian_tdata(axis_broadcaster_0_tdata[1]),
    .s_axis_cartesian_tvalid(axis_broadcaster_0_tvalid[1]),
    .s_axis_cartesian_tready(axis_broadcaster_0_tready[1]),
    .m_axis_dout_tdata(axis_cordic_0_tdata[1]),
    .m_axis_dout_tvalid(axis_cordic_0_tvalid[1]),
    .m_axis_dout_tready(axis_cordic_0_tready[1])
);

wire [31:0] axis_cordic_1_tdata [1:0];
wire [1:0] axis_cordic_1_tvalid;
wire [1:0] axis_cordic_1_tready;
```

```
    cordic_1 cordic_1_0(
        .aclk(aclk),
        .s_axis_phase_tdata(axis_cordic_0_tdata[0]),
        .s_axis_phase_tvalid(axis_cordic_0_tvalid[0]),
        .s_axis_phase_tready(axis_cordic_0_tready[0]),
        .m_axis_dout_tdata(axis_cordic_1_tdata[0]),
        .m_axis_dout_tvalid(axis_cordic_1_tvalid[0]),
        .m_axis_dout_tready(axis_cordic_1_tready[0])
    );

    cordic_1 cordic_1_1(
        .aclk(aclk),
        .s_axis_phase_tdata(axis_cordic_0_tdata[1]),
        .s_axis_phase_tvalid(axis_cordic_0_tvalid[1]),
        .s_axis_phase_tready(axis_cordic_0_tready[1]),
        .m_axis_dout_tdata(axis_cordic_1_tdata[1]),
        .m_axis_dout_tvalid(axis_cordic_1_tvalid[1]),
        .m_axis_dout_tready(axis_cordic_1_tready[1])
    );

    combiner_0 combiner_0(
        .aclk(aclk),
        .aresetn(1),
        .s_axis_tdata({
            axis_cordic_1_tdata[1],
            axis_cordic_1_tdata[0]
        }),
        .s_axis_tvalid(axis_cordic_1_tvalid),
        .s_axis_tready(axis_cordic_1_tready),
        .m_axis_tdata(m_axis_tdata),
        .m_axis_tvalid(m_axis_tvalid),
        .m_axis_tready(m_axis_tready)
    );

endmodule
```

scale_0.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/21/2018 07:33:06 PM
// Design Name:
// Module Name: scale_0
// Project Name:
// Target Devices:
// Tool Versions:
```

```
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module scale_0(
    input aclk,
    input [223:0] s_axis_tdata,
    input s_axis_tvalid,
    output s_axis_tready,
    output [95:0] m_axis_tdata,
    output m_axis_tvalid,
    input m_axis_tready
    );

    parameter N = 6;

    wire [223:0] axis_shift_0_tdata [N:0];
    wire axis_shift_0_tvalid [N:0];
    wire axis_shift_0_tready [N:0];

    assign axis_shift_0_tdata[0] = s_axis_tdata;
    assign axis_shift_0_tvalid[0] = s_axis_tvalid;
    assign s_axis_tready = axis_shift_0_tready[0];

    genvar i;

    generate
        for (i = 0; i < N; i = i+1) begin
            shift_0 shift_0(
                .aclk(aclk),
                .s_axis_tdata(axis_shift_0_tdata[i]),
                .s_axis_tvalid(axis_shift_0_tvalid[i]),
                .s_axis_tready(axis_shift_0_tready[i]),
                .m_axis_tdata(axis_shift_0_tdata[i+1]),
                .m_axis_tvalid(axis_shift_0_tvalid[i+1]),
                .m_axis_tready(axis_shift_0_tready[i+1])
            );
        end
    endgenerate

    assign m_axis_tdata = {
```

```
        {2{axis_shift_0_tdata[N][223]}},
        axis_shift_0_tdata[N][223:202],
        {2{axis_shift_0_tdata[N][167]}},
        axis_shift_0_tdata[N][167:146],
        {2{axis_shift_0_tdata[N][111]}},
        axis_shift_0_tdata[N][111:90],
        {2{axis_shift_0_tdata[N][55]}},
        axis_shift_0_tdata[N][55:34]
    };
    assign m_axis_tvalid = axis_shift_0_tvalid[N];
    assign axis_shift_0_tready[N] = m_axis_tready;

endmodule
```

shift_0.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/21/2018 07:48:13 PM
// Design Name:
// Module Name: shift_0
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module shift_0(
    input aclk,
    input [223:0] s_axis_tdata,
    input s_axis_tvalid,
    output s_axis_tready,
    output [223:0] m_axis_tdata,
    output m_axis_tvalid,
    input m_axis_tready
    );

    parameter N = 8;
```

```
    wire [55:0] data [N:0][3:0];

    assign data[0][3] = s_axis_tdata[223:168];
    assign data[0][2] = s_axis_tdata[167:112];
    assign data[0][1] = s_axis_tdata[111:56];
    assign data[0][0] = s_axis_tdata[55:0];

    wire shift [N-1:0][1:0];

    genvar i, j;

    generate
        for (i = 0; i < N; i = i+1) begin
            for (j = 0; j < 2; j = j+1) begin
                assign shift[i][j] = (
                    (data[i][j*2][55:54] == 2'b00 || data[i][j*2][55:54] == 2'b11) &&
                    (data[i][j*2+1][55:54] == 2'b00 || data[i][j*2+1][55:54] ==
2'b11)
                );
            end
            for (j = 0; j < 4; j = j+1) begin
                assign data[i+1][j] = shift[i][j/2] ? {data[i][j][54:0], 1'b0} :
data[i][j];
            end
        end
    endgenerate

    slice_0 slice_0(
        .aclk(aclk),
        .aresetn(1),
        .s_axis_tdata({
            data[N][3],
            data[N][2],
            data[N][1],
            data[N][0]
        }),
        .s_axis_tvalid(s_axis_tvalid),
        .s_axis_tready(s_axis_tready),
        .m_axis_tdata(m_axis_tdata),
        .m_axis_tvalid(m_axis_tvalid),
        .m_axis_tready(m_axis_tready)
    );

endmodule
uart_transmitter.v
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
```

```
// Company:
// Engineer:
//
// Create Date: 11/15/2018 03:25:14 PM
// Design Name:
// Module Name: uart_transmitter
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module uart_transmitter(
    input clock,
    input [7:0] uart_data,
    input uart_start,
    output uart_ready,
    output reg uart_dout
    );

    parameter DIVISOR = 100;
    parameter STOP_BIT = 1'b1;
    parameter START_BIT = 1'b0;

    reg [7:0] shift_reg;
    reg [6:0] clock_count;
    reg [3:0] bit_count;

    initial begin
        bit_count = 3'b0;
        uart_dout = STOP_BIT;
    end

    assign uart_ready = (bit_count == 0) && !uart_start;

    always @(posedge clock) begin
        if (uart_start) begin
            shift_reg <= {uart_data};
            clock_count <= 0;
            bit_count <= 10;
```

```
            uart_dout <= START_BIT;
        end
        if (bit_count != 0) begin
            if (clock_count == DIVISOR-1) begin
                clock_count <= 0;
                shift_reg <= {STOP_BIT, shift_reg[7:1]};
                bit_count <= bit_count-1;
                uart_dout <= shift_reg[0];
            end
            else begin
                clock_count <= clock_count+1;
            end
        end
    end

endmodule
```

upstream_bram.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/15/2018 07:18:44 PM
// Design Name:
// Module Name: upstream_bram
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module upstream_bram(
    input clock,
    input [9:0] addr,
    input [63:0] din,
    output reg [63:0] dout,
    input we,
    output stable
    );
```

```
    (* ram_style = "block" *)
    reg [63:0] bram [(1<<10)-1:0];

    reg [9:0] last;

    assign stable = (last == addr);

    always @(posedge clock) begin
        if (we) bram[addr] <= din;
        dout <= bram[addr];
        last <= addr;
    end

endmodule
```

upstream_hub.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/15/2018 07:17:58 PM
// Design Name:
// Module Name: upstream_hub
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module upstream_hub(
    input clock,
    input i2s_ready,
    input [63:0] i2s_data,
    output [63:0] m_axis_in_tdata,
    output m_axis_in_tvalid,
    input m_axis_in_tready
    );
```

```
    parameter STATE_LOAD = 0;
    parameter STATE_UNLOAD = 1;

    reg state;
    reg [9:0] index;
    wire we;
    wire stable;

    upstream_bram upstream_bram(
        .clock(clock),
        .addr(index),
        .din(i2s_data),
        .dout(m_axis_in_tdata),
        .we(we),
        .stable(stable)
    );

    assign we = (state == STATE_LOAD) && i2s_ready;
    assign m_axis_in_tvalid = (state == STATE_UNLOAD) && stable;

    initial begin
        state = STATE_LOAD;
        index = 0;
    end

    always @(posedge clock) begin
        case (state)
            STATE_LOAD: begin
                if (i2s_ready) begin
                    index <= index+1;
                    if (index == {10{1'b1}}) begin
                        state <= STATE_UNLOAD;
                    end
                end
            end
            STATE_UNLOAD: begin
                if (stable && m_axis_in_tready) begin
                    index <= index+1;
                    if (index == {10{1'b1}}) begin
                        state <= STATE_LOAD;
                    end
                end
            end
        endcase
    end
endmodule
```

Frontend Source Code

buggame.v

```
`timescale 1ns / 1ps
module bugJS(
   input clk,
   input reset,
   input die,
   input [10:0] hcount,
   input [9:0] vcount,
   output draw,
   output reg dead);

   parameter MUL = 26361;
   parameter ADD = 21321;

   reg [31:0] rand;

   wire signed [11:0] hcount_t = {1'b0, hcount};
   wire signed [10:0] vcount_t = {1'b0, vcount};

   reg signed [11:0] xpos;
   reg signed [10:0] ypos;
   reg signed [4:0] xvel;
   reg signed [4:0] yvel;
   reg signed [1:0] xacc;
   reg signed [1:0] yacc;

   wire signed [11:0] dx = (hcount_t - xpos) >>> 2;
   wire signed [10:0] dy = (vcount_t - ypos) >>> 2;
   wire drawable = (dx < 5) && (dx >= -5) && (dy >= -4) && (dy < 4);
   wire [3:0] col = 4 - dx;
   wire [2:0] row = {~dy[2], dy[1:0]};
   wire [79:0] font = 80'h2040413F014854545420;
   assign draw = drawable && font[{col, row}];

   wire outofpos = (xpos < 148) || (xpos > 850) || (ypos < 142) || (ypos > 631);
   reg [31:0] counter;
   always @(posedge clk) begin
      counter <= counter + 1;
      rand <= rand * MUL + ADD;
      if (reset) begin
         counter <= 0;
         dead <= 0;
         xpos <= 243 + rand[8:0];
         ypos <= 258 + rand[15:8];
         xvel <= rand[13:9];
         yvel <= rand[8:4];
         xacc <= rand[15:14];
         yacc <= rand[13:12];
      end else if (die) begin
```

```
            dead <= 1;
            counter <= 0;
        end else if (&(counter[21:0]) && !dead) begin
            if (outofpos) begin
                xpos <= 243 + rand[8:0];
                ypos <= 258 + rand[15:8];
                xvel <= rand[13:9];
                yvel <= rand[8:4];
                xacc <= rand[15:14];
                yacc <= rand[13:12];
            end else begin
                xpos <= xpos + xvel;
                ypos <= ypos + yvel;
                xvel <= xvel + xacc;
                yvel <= yvel + yacc;
                xacc <= rand[1:0];
                yacc <= rand[3:2];
            end
        end else if (&(counter[26:0]) && dead) begin
            dead <= 0;
        end
    end
endmodule

module bugEquals(
    input clk,
    input reset,
    input die,
    input [10:0] hcount,
    input [9:0] vcount,
    output draw,
    output reg dead);

    parameter MUL = 47363;
    parameter ADD = 63823;

    reg [31:0] rand;

    wire signed [11:0] hcount_t = {1'b0, hcount};
    wire signed [10:0] vcount_t = {1'b0, vcount};

    reg signed [11:0] xpos;
    reg signed [10:0] ypos;
    reg signed [4:0] xvel;
    reg signed [4:0] yvel;
    reg signed [1:0] xacc;
    reg signed [1:0] yacc;
```

```
   wire signed [11:0] dx = (hcount_t - xpos) >>> 1;
   wire signed [10:0] dy = (vcount_t - ypos) >>> 1;
   wire drawable = (dx < 20) && (dx >= -20) && (dy >= -4) && (dy < 4);
   wire [5:0] col = 19 - dx;
   wire [2:0] row = {~dy[2], dy[1:0]};
   wire [319:0] font =
320'h00447D4000087E090102001C22410044281028441414141414143E5149453E0041221C000008364100
;

   assign draw = drawable && font[{col, row}];

   wire outofpos = (xpos < 148) || (xpos > 850) || (ypos < 142) || (ypos > 631);
   reg [31:0] counter;
   always @(posedge clk) begin
      counter <= counter + 1;
      rand <= rand * MUL + ADD;
      if (reset) begin
         counter <= 0;
         dead <= 0;
         xpos <= 243 + rand[8:0];
         ypos <= 258 + rand[15:8];
         xvel <= rand[13:9];
         yvel <= rand[8:4];
         xacc <= rand[15:14];
         yacc <= rand[13:12];
      end else if (die) begin
         dead <= 1;
      end else if (&(counter[21:0]) && !dead) begin
         if (outofpos) begin
            xpos <= 243 + rand[8:0];
            ypos <= 258 + rand[15:8];
            xvel <= rand[13:9];
            yvel <= rand[8:4];
            xacc <= rand[15:14];
            yacc <= rand[13:12];
         end else begin
            xpos <= xpos + xvel;
            ypos <= ypos + yvel;
            xvel <= xvel + xacc;
            yvel <= yvel + yacc;
            xacc <= rand[1:0];
            yacc <= rand[3:2];
         end
      end else if (&(counter[26:0]) && dead) begin
         dead <= 0;
      end
   end
endmodule
```

```
module bugString(
   input clk,
   input reset,
   input die,
   input [10:0] hcount,
   input [9:0] vcount,
   output draw,
   output reg dead);

   parameter MUL = 8131;
   parameter ADD = 18741;

   reg [31:0] rand;

   wire signed [11:0] hcount_t = {1'b0, hcount};
   wire signed [10:0] vcount_t = {1'b0, vcount};

   reg signed [11:0] xpos;
   reg signed [10:0] ypos;
   reg signed [4:0] xvel;
   reg signed [4:0] yvel;
   reg signed [1:0] xacc;
   reg signed [1:0] yacc;

   wire signed [11:0] dx = (hcount_t - xpos) >>> 1;
   wire signed [10:0] dy = (vcount_t - ypos) >>> 1;
   wire drawable = (dx < 30) && (dx >= -30) && (dy >= -4) && (dy < 4);
   wire [5:0] col = 29 - dx;
   wire [2:0] row = {~dy[2], dy[1:0]};
   wire [519:0] font =
520'h00447D4000087E090102001C22410048545454200043F4440207C080404081414141414141414141414
00070007007E1111117E000700070000041221C00;

   assign draw = drawable && font[{col, row}];


   wire outofpos = (xpos < 148) || (xpos > 850) || (ypos < 142) || (ypos > 631);
   reg [31:0] counter;
   always @(posedge clk) begin
      counter <= counter + 1;
      rand <= rand * MUL + ADD;
      if (reset) begin
         counter <= 0;
         dead <= 0;
         xpos <= 243 + rand[8:0];
         ypos <= 258 + rand[15:8];
         xvel <= rand[13:9];
```

```
            yvel <= rand[8:4];
            xacc <= rand[15:14];
            yacc <= rand[13:12];
         end else if (die) begin
            dead <= 1;
         end else if (&(counter[21:0]) && !dead) begin
            if (outofpos) begin
               xpos <= 243 + rand[8:0];
               ypos <= 258 + rand[15:8];
               xvel <= rand[13:9];
               yvel <= rand[8:4];
               xacc <= rand[15:14];
               yacc <= rand[13:12];
            end else begin
               xpos <= xpos + xvel;
               ypos <= ypos + yvel;
               xvel <= xvel + xacc;
               yvel <= yvel + yacc;
               xacc <= rand[1:0];
               yacc <= rand[3:2];
            end
         end else if (&(counter[26:0]) && dead) begin
            dead <= 0;
         end
      end
   endmodule
```

debounce.v

```
////////////////////////////////////////////////////////////////////////////////
//
// Pushbutton Debounce Module
//
////////////////////////////////////////////////////////////////////////////////

module debounce (reset, clk, noisy, clean);
   input reset, clk, noisy;
   output clean;

   parameter NDELAY = 650000;
   parameter NBITS = 20;

   reg [NBITS-1:0] count;
   reg xnew, clean;

   always @(posedge clk)
     if (reset) begin xnew <= noisy; clean <= noisy; count <= 0; end
     else if (noisy != xnew) begin xnew <= noisy; count <= 0; end
```

```
     else if (count == NDELAY) clean <= xnew;
     else count <= count+1;

endmodule
```

display_16hex.v

```
//////////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Hex display driver
//
//
// File:   display_16hex.v
// Date:   24-Sep-05
//
// Created: April 27, 2004
// Author: Nathan Ickes
//
// This module drives the labkit hex displays and shows the value of
// 8 bytes (16 hex digits) on the displays.
//
// 24-Sep-05 Ike: updated to use new reset-once state machine, remove clear
// 02-Nov-05 Ike: updated to make it completely synchronous
//
// Inputs:
//
//   reset       - active high
//   clock_27mhz - the synchronous clock
//   data        - 64 bits; each 4 bits gives a hex digit
//
// Outputs:
//
//    disp_*     - display lines used in the 6.111 labkit (rev 003 & 004)
//
//////////////////////////////////////////////////////////////////////////////////

module display_16hex (reset, clock_27mhz, data_in,
             disp_blank, disp_clock, disp_rs, disp_ce_b,
             disp_reset_b, disp_data_out);

   input reset, clock_27mhz;    // clock and reset (active high reset)
   input [63:0] data_in;         // 16 hex nibbles to display

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

   reg disp_data_out, disp_rs, disp_ce_b, disp_reset_b;
```

```
   ////////////////////////////////////////////////////////////////////////
   //
   // Display Clock
   //
   // Generate a 500kHz clock for driving the displays.
   //
   ////////////////////////////////////////////////////////////////////////

   reg [5:0] count;
   reg [7:0] reset_count;
//   reg          old_clock;
   wire      dreset;
   wire      clock = (count<27) ? 0 : 1;

   always @(posedge clock_27mhz)
     begin
       count <= reset ? 0 : (count==53 ? 0 : count+1);
       reset_count <= reset ? 100 : ((reset_count==0) ? 0 : reset_count-1);
//     old_clock <= clock;
     end

   assign dreset = (reset_count != 0);
   assign disp_clock = ~clock;
   wire   clock_tick = ((count==27) ? 1 : 0);
//   wire   clock_tick = clock & ~old_clock;

   ////////////////////////////////////////////////////////////////////////
   //
   // Display State Machine
   //
   ////////////////////////////////////////////////////////////////////////

   reg [7:0] state;       // FSM state
   reg [9:0] dot_index;          // index to current dot being clocked out
   reg [31:0] control;           // control register
   reg [3:0] char_index;  // index of current character
   reg [39:0] dots;       // dots for a single digit
   reg [3:0] nibble;             // hex nibble of current character
   reg [63:0] data;

   assign disp_blank = 1'b0; // low <= not blanked

   always @(posedge clock_27mhz)
     if (clock_tick)
       begin
        if (dreset)
          begin
             state <= 0;
```

```
      dot_index <= 0;
      control <= 32'h7F7F7F7F;
    end
  else
    casex (state)
      8'h00:
      begin
         // Reset displays
         disp_data_out <= 1'b0;
         disp_rs <= 1'b0; // dot register
         disp_ce_b <= 1'b1;
         disp_reset_b <= 1'b0;
         dot_index <= 0;
         state <= state+1;
      end

      8'h01:
      begin
         // End reset
         disp_reset_b <= 1'b1;
         state <= state+1;
      end

      8'h02:
      begin
         // Initialize dot register (set all dots to zero)
         disp_ce_b <= 1'b0;
         disp_data_out <= 1'b0; // dot_index[0];
         if (dot_index == 639)
           state <= state+1;
         else
           dot_index <= dot_index+1;
      end

      8'h03:
      begin
         // Latch dot data
         disp_ce_b <= 1'b1;
         dot_index <= 31;       // re-purpose to init ctrl reg
         state <= state+1;
      end

      8'h04:
      begin
         // Setup the control register
         disp_rs <= 1'b1; // Select the control register
         disp_ce_b <= 1'b0;
         disp_data_out <= control[31];
```

```
            control <= {control[30:0], 1'b0};    // shift left
            if (dot_index == 0)
              state <= state+1;
            else
              dot_index <= dot_index-1;
          end

         8'h05:
          begin
            // Latch the control register data / dot data
            disp_ce_b <= 1'b1;
            dot_index <= 39;        // init for single char
            char_index <= 15;              // start with MS char
            data <= data_in;
            state <= state+1;
          end

         8'h06:
          begin
            // Load the user's dot data into the dot reg, char by char
            disp_rs <= 1'b0;               // Select the dot register
            disp_ce_b <= 1'b0;
            disp_data_out <= dots[dot_index]; // dot data from msb
            if (dot_index == 0)
                if (char_index == 0)
                  state <= 5;              // all done, latch data
              else
                begin
                  char_index <= char_index - 1;  // goto next char
                  data <= data_in;
                  dot_index <= 39;
                end
            else
              dot_index <= dot_index-1;    // else loop thru all dots
          end

        endcase // casex(state)
    end

always @ (data or char_index)
  case (char_index)
    4'h0:        nibble <= data[3:0];
    4'h1:        nibble <= data[7:4];
    4'h2:        nibble <= data[11:8];
    4'h3:        nibble <= data[15:12];
    4'h4:        nibble <= data[19:16];
    4'h5:        nibble <= data[23:20];
    4'h6:        nibble <= data[27:24];
```

```
      4'h7:        nibble <= data[31:28];
      4'h8:        nibble <= data[35:32];
      4'h9:        nibble <= data[39:36];
      4'hA:        nibble <= data[43:40];
      4'hB:        nibble <= data[47:44];
      4'hC:        nibble <= data[51:48];
      4'hD:        nibble <= data[55:52];
      4'hE:        nibble <= data[59:56];
      4'hF:        nibble <= data[63:60];
    endcase

  always @(nibble)
    case (nibble)
      4'h0: dots <= 40'b00111110_01010001_01001001_01000101_00111110;
      4'h1: dots <= 40'b00000000_01000010_01111111_01000000_00000000;
      4'h2: dots <= 40'b01100010_01010001_01001001_01001001_01000110;
      4'h3: dots <= 40'b00100010_01000001_01001001_01001001_00110110;
      4'h4: dots <= 40'b00011000_00010100_00010010_01111111_00010000;
      4'h5: dots <= 40'b00100111_01000101_01000101_01000101_00111001;
      4'h6: dots <= 40'b00111100_01001010_01001001_01001001_00110000;
      4'h7: dots <= 40'b00000001_01110001_00001001_00000101_00000011;
      4'h8: dots <= 40'b00110110_01001001_01001001_01001001_00110110;
      4'h9: dots <= 40'b00000110_01001001_01001001_00101001_00011110;
      4'hA: dots <= 40'b01111110_00001001_00001001_00001001_01111110;
      4'hB: dots <= 40'b01111111_01001001_01001001_01001001_00110110;
      4'hC: dots <= 40'b00111110_01000001_01000001_01000001_00100010;
      4'hD: dots <= 40'b01111111_01000001_01000001_01000001_00111110;
      4'hE: dots <= 40'b01111111_01001001_01001001_01001001_01000001;
      4'hF: dots <= 40'b01111111_00001001_00001001_00001001_00000001;
    endcase

endmodule
```

ntsc2zbt.v

```
//
// File:   ntsc2zbt.v
// Date:   27-Nov-05
// Author: I. Chuang <ichuang@mit.edu>
//
// Example for MIT 6.111 labkit showing how to prepare NTSC data
// (from Javier's decoder) to be loaded into the ZBT RAM for video
// display.
//
// The ZBT memory is 36 bits wide; we only use 32 bits of this, to
// store 4 bytes of black-and-white intensity data from the NTSC
// video input.
//
// Bug fix: Jonathan P. Mailoa <jpmailoa@mit.edu>
```

```
// Date   : 11-May-09  // gph mod 11/3/2011
//
//
// Bug due to memory management will be fixed. It happens because
// the memory addressing protocol is off between ntsc2zbt.v and
// vram_display.v. There are 2 solutions:
// -. Fix the memory addressing in this module (neat addressing protocol)
//    and do memory forecast in vram_display module.
// -. Do nothing in this module and do memory forecast in vram_display
//    module (different forecast count) while cutting off reading from
//    address(0,0,0).
//
// Bug in this module causes 4 pixel on the rightmost side of the camera
// to be stored in the address that belongs to the leftmost side of the
// screen.
//
// In this example, the second method is used. NOTICE will be provided
// on the crucial source of the bug.
//
//////////////////////////////////////////////////////////////////////////
// Prepare data and address values to fill ZBT memory with NTSC data

module ntsc_to_zbt(clk, vclk, fvh, dv, din, ntsc_addr, ntsc_data, ntsc_we, sw);

   input      clk;  // system clock
   input      vclk; // video clock from camera
   input [2:0]       fvh;
   input      dv;
   input [17:0]      din;
   output [18:0] ntsc_addr;
   output [35:0] ntsc_data;
   output     ntsc_we;     // write enable for NTSC data
   input      sw;          // switch which determines mode (for debugging)

   parameter  COL_START = 10'd30;
   parameter  ROW_START = 10'd30;

   // here put the luminance data from the ntsc decoder into the ram
   // this is for 1024 * 788 XGA display

   reg [9:0]  col = 0;
   reg [9:0]  row = 0;
   reg [17:0]        vdata = 0;
   reg               vwe;
   reg               old_dv;
   reg               old_frame;  // frames are even / odd interlaced
   reg               even_odd;   // decode interlaced frame to this wire
```

```
wire        frame = fvh[2];
wire        frame_edge = frame & ~old_frame;

always @ (posedge vclk) //LLC1 is reference
  begin
    old_dv <= dv;
    vwe <= dv && !fvh[2] & ~old_dv; // if data valid, write it
    old_frame <= frame;
    even_odd = frame_edge ? ~even_odd : even_odd;

    if (!fvh[2])
      begin
         col <= fvh[0] ? COL_START :
               (!fvh[2] && !fvh[1] && dv && (col < 1024)) ? col + 1 : col;
         row <= fvh[1] ? ROW_START :
               (!fvh[2] && fvh[0] && (row < 768)) ? row + 1 : row;
         vdata <= (dv && !fvh[2]) ? din : vdata;
      end
  end

// synchronize with system clock

reg [9:0] x[1:0],y[1:0];
reg [17:0] data[1:0];
reg       we[1:0];
reg            eo[1:0];

always @(posedge clk)
  begin
    {x[1],x[0]} <= {x[0],col};
    {y[1],y[0]} <= {y[0],row};
    {data[1],data[0]} <= {data[0],vdata};
    {we[1],we[0]} <= {we[0],vwe};
    {eo[1],eo[0]} <= {eo[0],even_odd};
  end

// edge detection on write enable signal

reg old_we;
wire we_edge = we[1] & ~old_we;
always @(posedge clk) old_we <= we[1];

// shift each set of four bytes into a large register for the ZBT

reg [35:0] mydata;
always @(posedge clk)
  if (we_edge)
    mydata <= { mydata[17:0], data[1] };
```

```
// NOTICE : Here we have put 4 pixel delay on mydata. For example, when:
// (x[1], y[1]) = (60, 80) and eo[1] = 0, then:
// mydata[31:0] = ( pixel(56,160), pixel(57,160), pixel(58,160), pixel(59,160) )
// This is the root of the original addressing bug.


// NOTICE : Notice that we have decided to store mydata, which
//          contains pixel(56,160) to pixel(59,160) in address
//          (0, 160 (10 bits), 60 >> 2 = 15 (8 bits)).
//
//          This protocol is dangerous, because it means
//          pixel(0,0) to pixel(3,0) is NOT stored in address
//          (0, 0 (10 bits), 0 (8 bits)) but is rather stored
//          in address (0, 0 (10 bits), 4 >> 2 = 1 (8 bits)). This
//          calculation ignores COL_START & ROW_START.
//
//          4 pixels from the right side of the camera input will
//          be stored in address corresponding to x = 0.
//
//          To fix, delay col & row by 4 clock cycles.
//          Delay other signals as well.

reg [19:0] x_delay;
reg [19:0] y_delay;
reg [1:0] we_delay;
reg [1:0] eo_delay;

always @ (posedge clk)
begin
  x_delay <= {x_delay[19:0], x[1]};
  y_delay <= {y_delay[19:0], y[1]};
  we_delay <= {we_delay[1:0], we[1]};
  eo_delay <= {eo_delay[1:0], eo[1]};
end

// compute address to store data in
wire [8:0] y_addr = y_delay[18:10];
    wire [9:0] x_addr = x_delay[19:10];

wire [18:0] myaddr = {y_addr[8:0], eo_delay[1], x_addr[9:1]};

// Now address (0,0,0) contains pixel data(0,0) etc.


// alternate (256x192) image data and address
wire [31:0] mydata2 = {data[1],data[1],data[1],data[1]};
wire [18:0] myaddr2 = {1'b0, y_addr[8:0], eo_delay[1], x_addr[7:0]};
```

```
   // update the output address and data only when four bytes ready

   reg [18:0] ntsc_addr;
   reg [35:0] ntsc_data;
   wire       ntsc_we = sw ? we_edge : (we_edge & (x_delay[10]==1'b0));

   always @(posedge clk)
     if ( ntsc_we )
       begin
         ntsc_addr <= sw ? myaddr2 : myaddr;   // normal and expanded modes
         ntsc_data <= sw ? {4'b0,mydata2} : {mydata};
       end

endmodule // ntsc_to_zbt


score_tracker.v
`timescale 1ns / 1ps

module score_tracker(
   input clk,
   input [7:0] score,
   input [10:0] hcount,
   input [9:0] vcount,
   input reset,
   output draw);

//   wire [39:0] font;
//   reg [7:0] addr;
//   fullfont ff(.ADDRA(addr), .CLK(clk), .DOUTA(font));

//   parameter S_LOAD_FONT_START = 0;
//   parameter S_LOAD_FONT = 1;
//   parameter S_DISPLAY = 2;
//
//   reg [1:0] state;
//   reg [3:0] indx;
//   wire [47:0] scorestring = 48'h53636f72653a;
   wire [287:0] font =
287'h464949493100384444442000384444443800 7C080404080038545454180036360000000;
   wire [8:0] index_text = {36 - hcount[7:2], vcount[4:2]};
   wire [8:0] index_score = {18 - hcount[7:2], vcount[4:2]};
   wire drawable_text = (hcount < 144) && (vcount <= 31);
   wire drawable_score = (hcount < 216) && (hcount >= 144) && (vcount <= 31);
   wire [143:0] score_pixels;
   assign score_pixels[143:96] = 0;
   assign draw = (drawable_text && font[index_text]) || (drawable_score &&
```

```
score_pixels[index_score]);

   hexfont hf(.clka(clk), .clkb(clk), .addra(score[7:4]),
.douta(score_pixels[95:48]), .addrb(score[3:0]), .doutb(score_pixels[47:0]));

   always @(posedge clk) begin
   end
endmodule
```

translate_hvcount.v

```
`timescale 1ns / 1ps
`define left_x 0
`define right_x 1023
`define top_y 0
`define bottom_y 767

module translate_hvcount(
   input [10:0] hcount,
   input [9:0] vcount,
   output [10:0] hcount_t,
   output [9:0] vcount_t);

//   wire signed [11:0] left_x;
//   wire signed [11:0] right_x;
//   wire signed [10:0] top_y;
//   wire signed [10:0] bottom_y;
//   assign left_x = 12'd0;
//   assign right_x = 12'd1023;
//   assign top_y = 11'd0;
//   assign bottom_y = 11'd767;

//   // (hcount-left_x)/(right_x-left_x) * 1024;
//   wire signed [11:0] hcount_s = {1'b0, hcount};
//   wire signed [10:0] vcount_s = {1'b0, vcount};
//
//   wire signed [11:0] dx = hcount_s - left_x;
//   wire signed [10:0] dy = vcount_s - top_y;
//
   assign hcount_t = 1023 - (hcount + 140);
   assign vcount_t = vcount - 62;
endmodule
```

video_decoder.v

```
//
// File:   video_decoder.v
// Date:   31-Oct-05
// Author: J. Castro (MIT 6.111, fall 2005)
//
```

```
// This file contains the ntsc_decode and adv7185init modules
//
// These modules are used to grab input NTSC video data from the RCA
// phono jack on the right hand side of the 6.111 labkit (connect
// the camera to the LOWER jack).
//

//////////////////////////////////////////////////////////////////////////////
//
// NTSC decode - 16-bit CCIR656 decoder
// By Javier Castro
// This module takes a stream of LLC data from the adv7185
// NTSC/PAL video decoder and generates the corresponding pixels,
// that are encoded within the stream, in YCrCb format.

// Make sure that the adv7185 is set to run in 16-bit LLC2 mode.

module ntsc_decode(clk, reset, tv_in_ycrcb, ycrcb, f, v, h, data_valid);

   // clk - line-locked clock (in this case, LLC1 which runs at 27Mhz)
   // reset - system reset
   // tv_in_ycrcb - 10-bit input from chip. should map to pins [19:10]
   // ycrcb - 24 bit luminance and chrominance (8 bits each)
   // f - field: 1 indicates an even field, 0 an odd field
   // v - vertical sync: 1 means vertical sync
   // h - horizontal sync: 1 means horizontal sync

   input clk;
   input reset;
   input [9:0] tv_in_ycrcb; // modified for 10 bit input - should be P[19:10]
   output [29:0] ycrcb;
   output    f;
   output    v;
   output    h;
   output    data_valid;
   // output [4:0] state;

   parameter SYNC_1 = 0;
   parameter SYNC_2 = 1;
   parameter SYNC_3 = 2;
   parameter SAV_f1_cb0 = 3;
   parameter SAV_f1_y0 = 4;
   parameter SAV_f1_cr1 = 5;
   parameter SAV_f1_y1 = 6;
   parameter EAV_f1 = 7;
   parameter SAV_VBI_f1 = 8;
   parameter EAV_VBI_f1 = 9;
   parameter SAV_f2_cb0 = 10;
```

```
   parameter SAV_f2_y0 = 11;
   parameter SAV_f2_cr1 = 12;
   parameter SAV_f2_y1 = 13;
   parameter EAV_f2 = 14;
   parameter SAV_VBI_f2 = 15;
   parameter EAV_VBI_f2 = 16;




   // In the start state, the module doesn't know where
   // in the sequence of pixels, it is looking.

   // Once we determine where to start, the FSM goes through a normal
   // sequence of SAV process_YCrCb EAV... repeat

   // The data stream looks as follows
   // SAV_FF | SAV_00 | SAV_00 | SAV_XY | Cb0 | Y0 | Cr1 | Y1 | Cb2 | Y2 | ... | EAV
sequence
   // There are two things we need to do:
   //   1. Find the two SAV blocks (stands for Start Active Video perhaps?)
   //   2. Decode the subsequent data

   reg [4:0] current_state = 5'h00;
   reg [9:0] y = 10'h000;  // luminance
   reg [9:0] cr = 10'h000; // chrominance
   reg [9:0] cb = 10'h000; // more chrominance

   assign    state = current_state;

   always @ (posedge clk)
     begin
       if (reset)
         begin

         end
       else
         begin
            // these states don't do much except allow us to know where we are in the
stream.
            // whenever the synchronization code is seen, go back to the sync_state
before
            // transitioning to the new state
            case (current_state)
              SYNC_1: current_state <= (tv_in_ycrcb == 10'h000) ? SYNC_2 : SYNC_1;
              SYNC_2: current_state <= (tv_in_ycrcb == 10'h000) ? SYNC_3 : SYNC_1;
              SYNC_3: current_state <= (tv_in_ycrcb == 10'h200) ? SAV_f1_cb0 :
                                       (tv_in_ycrcb == 10'h274) ? EAV_f1 :
```

```
                                (tv_in_ycrcb == 10'h2ac) ? SAV_VBI_f1 :
                                (tv_in_ycrcb == 10'h2d8) ? EAV_VBI_f1 :
                                (tv_in_ycrcb == 10'h31c) ? SAV_f2_cb0 :
                                (tv_in_ycrcb == 10'h368) ? EAV_f2 :
                                (tv_in_ycrcb == 10'h3b0) ? SAV_VBI_f2 :
                                (tv_in_ycrcb == 10'h3c4) ? EAV_VBI_f2 : SYNC_1;

              SAV_f1_cb0: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 :
SAV_f1_y0;
              SAV_f1_y0: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 :
SAV_f1_cr1;
              SAV_f1_cr1: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 :
SAV_f1_y1;
              SAV_f1_y1: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 :
SAV_f1_cb0;

              SAV_f2_cb0: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 :
SAV_f2_y0;
              SAV_f2_y0: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 :
SAV_f2_cr1;
              SAV_f2_cr1: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 :
SAV_f2_y1;
              SAV_f2_y1: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 :
SAV_f2_cb0;

              // These states are here in the event that we want to cover these
signals
              // in the future. For now, they just send the state machine back to
SYNC_1
              EAV_f1: current_state <= SYNC_1;
              SAV_VBI_f1: current_state <= SYNC_1;
              EAV_VBI_f1: current_state <= SYNC_1;
              EAV_f2: current_state <= SYNC_1;
              SAV_VBI_f2: current_state <= SYNC_1;
              EAV_VBI_f2: current_state <= SYNC_1;

           endcase
        end
     end // always @ (posedge clk)

   // implement our decoding mechanism

   wire y_enable;
   wire cr_enable;
   wire cb_enable;

   // if y is coming in, enable the register
   // likewise for cr and cb
```

```
   assign y_enable = (current_state == SAV_f1_y0) ||
                     (current_state == SAV_f1_y1) ||
                     (current_state == SAV_f2_y0) ||
                     (current_state == SAV_f2_y1);
   assign cr_enable = (current_state == SAV_f1_cr1) ||
                      (current_state == SAV_f2_cr1);
   assign cb_enable = (current_state == SAV_f1_cb0) ||
                      (current_state == SAV_f2_cb0);

   // f, v, and h only go high when active
   assign {v,h} = (current_state == SYNC_3) ? tv_in_ycrcb[7:6] : 2'b00;

   // data is valid when we have all three values: y, cr, cb
   assign data_valid = y_enable;
   assign ycrcb = {y,cr,cb};

   reg         f = 0;

   always @ (posedge clk)
     begin
       y <= y_enable ? tv_in_ycrcb : y;
       cr <= cr_enable ? tv_in_ycrcb : cr;
       cb <= cb_enable ? tv_in_ycrcb : cb;
       f <= (current_state == SYNC_3) ? tv_in_ycrcb[8] : f;
     end

endmodule



//////////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- ADV7185 Video Decoder Configuration Init
//
// Created:
// Author: Nathan Ickes
//
//////////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////////
// Register 0
//////////////////////////////////////////////////////////////////////////////////

`define INPUT_SELECT                            4'h0
  // 0: CVBS on AIN1 (composite video in)
  // 7: Y on AIN2, C on AIN5 (s-video in)
  // (These are the only configurations supported by the 6.111 labkit hardware)
`define INPUT_MODE                              4'h0
```

```
  // 0: Autodetect: NTSC or PAL (BGHID), w/o pedestal
  // 1: Autodetect: NTSC or PAL (BGHID), w/pedestal
  // 2: Autodetect: NTSC or PAL (N), w/o pedestal
  // 3: Autodetect: NTSC or PAL (N), w/pedestal
  // 4: NTSC w/o pedestal
  // 5: NTSC w/pedestal
  // 6: NTSC 4.43 w/o pedestal
  // 7: NTSC 4.43 w/pedestal
  // 8: PAL BGHID w/o pedestal
  // 9: PAL N w/pedestal
  // A: PAL M w/o pedestal
  // B: PAL M w/pedestal
  // C: PAL combination N
  // D: PAL combination N w/pedestal
  // E-F: [Not valid]

`define ADV7185_REGISTER_0 {`INPUT_MODE, `INPUT_SELECT}

////////////////////////////////////////////////////////////////////////////////
// Register 1
////////////////////////////////////////////////////////////////////////////////

`define VIDEO_QUALITY                           2'h0
  // 0: Broadcast quality
  // 1: TV quality
  // 2: VCR quality
  // 3: Surveillance quality
`define SQUARE_PIXEL_IN_MODE                    1'b0
  // 0: Normal mode
  // 1: Square pixel mode
`define DIFFERENTIAL_INPUT                      1'b0
  // 0: Single-ended inputs
  // 1: Differential inputs
`define FOUR_TIMES_SAMPLING                     1'b0
  // 0: Standard sampling rate
  // 1: 4x sampling rate (NTSC only)
`define BETACAM                                 1'b0
  // 0: Standard video input
  // 1: Betacam video input
`define AUTOMATIC_STARTUP_ENABLE                1'b1
  // 0: Change of input triggers reacquire
  // 1: Change of input does not trigger reacquire

`define ADV7185_REGISTER_1 {`AUTOMATIC_STARTUP_ENABLE, 1'b0, `BETACAM,
`FOUR_TIMES_SAMPLING, `DIFFERENTIAL_INPUT, `SQUARE_PIXEL_IN_MODE, `VIDEO_QUALITY}

////////////////////////////////////////////////////////////////////////////////
// Register 2
```

```
////////////////////////////////////////////////////////////////////////////////

`define Y_PEAKING_FILTER                        3'h4
  // 0: Composite =  4.5dB,  s-video =  9.25dB
  // 1: Composite =  4.5dB,  s-video =  9.25dB
  // 2: Composite =  4.5dB,  s-video =  5.75dB
  // 3: Composite =  1.25dB, s-video =  3.3dB
  // 4: Composite =  0.0dB,  s-video =  0.0dB
  // 5: Composite = -1.25dB, s-video = -3.0dB
  // 6: Composite = -1.75dB, s-video = -8.0dB
  // 7: Composite = -3.0dB,  s-video = -8.0dB
`define CORING                                  2'h0
  // 0: No coring
  // 1: Truncate if Y < black+8
  // 2: Truncate if Y < black+16
  // 3: Truncate if Y < black+32

`define ADV7185_REGISTER_2 {3'b000, `CORING, `Y_PEAKING_FILTER}

////////////////////////////////////////////////////////////////////////////////
// Register 3
////////////////////////////////////////////////////////////////////////////////

`define INTERFACE_SELECT                        2'h0
  // 0: Philips-compatible
  // 1: Broktree API A-compatible
  // 2: Broktree API B-compatible
  // 3: [Not valid]
`define OUTPUT_FORMAT                           4'h0
  // 0: 10-bit @ LLC, 4:2:2 CCIR656
  // 1: 20-bit @ LLC, 4:2:2 CCIR656
  // 2: 16-bit @ LLC, 4:2:2 CCIR656
  // 3: 8-bit @ LLC, 4:2:2 CCIR656
  // 4: 12-bit @ LLC, 4:1:1
  // 5-F: [Not valid]
  // (Note that the 6.111 labkit hardware provides only a 10-bit interface to
  // the ADV7185.)
`define TRISTATE_OUTPUT_DRIVERS                 1'b0
  // 0: Drivers tristated when ~OE is high
  // 1: Drivers always tristated
`define VBI_ENABLE                              1'b0
  // 0: Decode lines during vertical blanking interval
  // 1: Decode only active video regions

`define ADV7185_REGISTER_3 {`VBI_ENABLE, `TRISTATE_OUTPUT_DRIVERS, `OUTPUT_FORMAT,
`INTERFACE_SELECT}

////////////////////////////////////////////////////////////////////////////////
```

```
// Register 4
//////////////////////////////////////////////////////////////////////////////

`define OUTPUT_DATA_RANGE                       1'b0
  // 0: Output values restricted to CCIR-compliant range
  // 1: Use full output range
`define BT656_TYPE                              1'b0
  // 0: BT656-3-compatible
  // 1: BT656-4-compatible

`define ADV7185_REGISTER_4 {`BT656_TYPE, 3'b000, 3'b110, `OUTPUT_DATA_RANGE}

//////////////////////////////////////////////////////////////////////////////
// Register 5
//////////////////////////////////////////////////////////////////////////////


`define GENERAL_PURPOSE_OUTPUTS                 4'b0000
`define GPO_0_1_ENABLE                          1'b0
  // 0: General purpose outputs 0 and 1 tristated
  // 1: General purpose outputs 0 and 1 enabled
`define GPO_2_3_ENABLE                          1'b0
  // 0: General purpose outputs 2 and 3 tristated
  // 1: General purpose outputs 2 and 3 enabled
`define BLANK_CHROMA_IN_VBI                     1'b1
  // 0: Chroma decoded and output during vertical blanking
  // 1: Chroma blanked during vertical blanking
`define HLOCK_ENABLE                            1'b0
  // 0: GPO 0 is a general purpose output
  // 1: GPO 0 shows HLOCK status

`define ADV7185_REGISTER_5 {`HLOCK_ENABLE, `BLANK_CHROMA_IN_VBI, `GPO_2_3_ENABLE,
`GPO_0_1_ENABLE, `GENERAL_PURPOSE_OUTPUTS}

//////////////////////////////////////////////////////////////////////////////
// Register 7
//////////////////////////////////////////////////////////////////////////////

`define FIFO_FLAG_MARGIN                        5'h10
  // Sets the locations where FIFO almost-full and almost-empty flags are set
`define FIFO_RESET                              1'b0
  // 0: Normal operation
  // 1: Reset FIFO. This bit is automatically cleared
`define AUTOMATIC_FIFO_RESET                    1'b0
  // 0: No automatic reset
  // 1: FIFO is autmatically reset at the end of each video field
`define FIFO_FLAG_SELF_TIME                     1'b1
  // 0: FIFO flags are synchronized to CLKIN
```

```
  // 1: FIFO flags are synchronized to internal 27MHz clock

`define ADV7185_REGISTER_7 {`FIFO_FLAG_SELF_TIME, `AUTOMATIC_FIFO_RESET, `FIFO_RESET,
`FIFO_FLAG_MARGIN}

////////////////////////////////////////////////////////////////////////////
// Register 8
////////////////////////////////////////////////////////////////////////////

`define INPUT_CONTRAST_ADJUST                       8'h80

`define ADV7185_REGISTER_8 {`INPUT_CONTRAST_ADJUST}

////////////////////////////////////////////////////////////////////////////
// Register 9
////////////////////////////////////////////////////////////////////////////

`define INPUT_SATURATION_ADJUST                     8'h8C

`define ADV7185_REGISTER_9 {`INPUT_SATURATION_ADJUST}

////////////////////////////////////////////////////////////////////////////
// Register A
////////////////////////////////////////////////////////////////////////////

`define INPUT_BRIGHTNESS_ADJUST                     8'h00

`define ADV7185_REGISTER_A {`INPUT_BRIGHTNESS_ADJUST}

////////////////////////////////////////////////////////////////////////////
// Register B
////////////////////////////////////////////////////////////////////////////

`define INPUT_HUE_ADJUST                            8'h00

`define ADV7185_REGISTER_B {`INPUT_HUE_ADJUST}

////////////////////////////////////////////////////////////////////////////
// Register C
////////////////////////////////////////////////////////////////////////////

`define DEFAULT_VALUE_ENABLE                    1'b0
  // 0: Use programmed Y, Cr, and Cb values
  // 1: Use default values
`define DEFAULT_VALUE_AUTOMATIC_ENABLE          1'b0
  // 0: Use programmed Y, Cr, and Cb values
  // 1: Use default values if lock is lost
`define DEFAULT_Y_VALUE                         6'h0C
```

```
  // Default Y value

`define ADV7185_REGISTER_C {`DEFAULT_Y_VALUE, `DEFAULT_VALUE_AUTOMATIC_ENABLE,
`DEFAULT_VALUE_ENABLE}

////////////////////////////////////////////////////////////////////////////////
// Register D
////////////////////////////////////////////////////////////////////////////////

`define DEFAULT_CR_VALUE                        4'h8
  // Most-significant four bits of default Cr value
`define DEFAULT_CB_VALUE                        4'h8
  // Most-significant four bits of default Cb value

`define ADV7185_REGISTER_D {`DEFAULT_CB_VALUE, `DEFAULT_CR_VALUE}

////////////////////////////////////////////////////////////////////////////////
// Register E
////////////////////////////////////////////////////////////////////////////////

`define TEMPORAL_DECIMATION_ENABLE              1'b0
  // 0: Disable
  // 1: Enable
`define TEMPORAL_DECIMATION_CONTROL             2'h0
  // 0: Supress frames, start with even field
  // 1: Supress frames, start with odd field
  // 2: Supress even fields only
  // 3: Supress odd fields only
`define TEMPORAL_DECIMATION_RATE                4'h0
  // 0-F: Number of fields/frames to skip

`define ADV7185_REGISTER_E {1'b0, `TEMPORAL_DECIMATION_RATE,
`TEMPORAL_DECIMATION_CONTROL, `TEMPORAL_DECIMATION_ENABLE}

////////////////////////////////////////////////////////////////////////////////
// Register F
////////////////////////////////////////////////////////////////////////////////

`define POWER_SAVE_CONTROL                      2'h0
  // 0: Full operation
  // 1: CVBS only
  // 2: Digital only
  // 3: Power save mode
`define POWER_DOWN_SOURCE_PRIORITY              1'b0
  // 0: Power-down pin has priority
  // 1: Power-down control bit has priority
`define POWER_DOWN_REFERENCE                    1'b0
  // 0: Reference is functional
```

```
  // 1: Reference is powered down
`define POWER_DOWN_LLC_GENERATOR                1'b0
  // 0: LLC generator is functional
  // 1: LLC generator is powered down
`define POWER_DOWN_CHIP                         1'b0
  // 0: Chip is functional
  // 1: Input pads disabled and clocks stopped
`define TIMING_REACQUIRE                        1'b0
  // 0: Normal operation
  // 1: Reacquire video signal (bit will automatically reset)
`define RESET_CHIP                              1'b0
  // 0: Normal operation
  // 1: Reset digital core and I2C interface (bit will automatically reset)

`define ADV7185_REGISTER_F {`RESET_CHIP, `TIMING_REACQUIRE, `POWER_DOWN_CHIP,
`POWER_DOWN_LLC_GENERATOR, `POWER_DOWN_REFERENCE, `POWER_DOWN_SOURCE_PRIORITY,
`POWER_SAVE_CONTROL}

////////////////////////////////////////////////////////////////////////////////
// Register 33
////////////////////////////////////////////////////////////////////////////////

`define PEAK_WHITE_UPDATE                       1'b1
  // 0: Update gain once per line
  // 1: Update gain once per field
`define AVERAGE_BIRIGHTNESS_LINES               1'b1
  // 0: Use lines 33 to 310
  // 1: Use lines 33 to 270
`define MAXIMUM_IRE                             3'h0
  // 0: PAL: 133, NTSC: 122
  // 1: PAL: 125, NTSC: 115
  // 2: PAL: 120, NTSC: 110
  // 3: PAL: 115, NTSC: 105
  // 4: PAL: 110, NTSC: 100
  // 5: PAL: 105, NTSC: 100
  // 6-7: PAL: 100, NTSC: 100
`define COLOR_KILL                              1'b1
  // 0: Disable color kill
  // 1: Enable color kill

`define ADV7185_REGISTER_33 {1'b1, `COLOR_KILL, 1'b1, `MAXIMUM_IRE,
`AVERAGE_BIRIGHTNESS_LINES, `PEAK_WHITE_UPDATE}

`define ADV7185_REGISTER_10 8'h00
`define ADV7185_REGISTER_11 8'h00
`define ADV7185_REGISTER_12 8'h00
`define ADV7185_REGISTER_13 8'h45
`define ADV7185_REGISTER_14 8'h18
```

```
`define ADV7185_REGISTER_15 8'h60
`define ADV7185_REGISTER_16 8'h00
`define ADV7185_REGISTER_17 8'h01
`define ADV7185_REGISTER_18 8'h00
`define ADV7185_REGISTER_19 8'h10
`define ADV7185_REGISTER_1A 8'h10
`define ADV7185_REGISTER_1B 8'hF0
`define ADV7185_REGISTER_1C 8'h16
`define ADV7185_REGISTER_1D 8'h01
`define ADV7185_REGISTER_1E 8'h00
`define ADV7185_REGISTER_1F 8'h3D
`define ADV7185_REGISTER_20 8'hD0
`define ADV7185_REGISTER_21 8'h09
`define ADV7185_REGISTER_22 8'h8C
`define ADV7185_REGISTER_23 8'hE2
`define ADV7185_REGISTER_24 8'h1F
`define ADV7185_REGISTER_25 8'h07
`define ADV7185_REGISTER_26 8'hC2
`define ADV7185_REGISTER_27 8'h58
`define ADV7185_REGISTER_28 8'h3C
`define ADV7185_REGISTER_29 8'h00
`define ADV7185_REGISTER_2A 8'h00
`define ADV7185_REGISTER_2B 8'hA0
`define ADV7185_REGISTER_2C 8'hCE
`define ADV7185_REGISTER_2D 8'hF0
`define ADV7185_REGISTER_2E 8'h00
`define ADV7185_REGISTER_2F 8'hF0
`define ADV7185_REGISTER_30 8'h00
`define ADV7185_REGISTER_31 8'h70
`define ADV7185_REGISTER_32 8'h00
`define ADV7185_REGISTER_34 8'h0F
`define ADV7185_REGISTER_35 8'h01
`define ADV7185_REGISTER_36 8'h00
`define ADV7185_REGISTER_37 8'h00
`define ADV7185_REGISTER_38 8'h00
`define ADV7185_REGISTER_39 8'h00
`define ADV7185_REGISTER_3A 8'h00
`define ADV7185_REGISTER_3B 8'h00

`define ADV7185_REGISTER_44 8'h41
`define ADV7185_REGISTER_45 8'hBB

`define ADV7185_REGISTER_F1 8'hEF
`define ADV7185_REGISTER_F2 8'h80


module adv7185init (reset, clock_27mhz, source, tv_in_reset_b,
                    tv_in_i2c_clock, tv_in_i2c_data);
```

```
input reset;
input clock_27mhz;
output tv_in_reset_b; // Reset signal to ADV7185
output tv_in_i2c_clock; // I2C clock output to ADV7185
output tv_in_i2c_data; // I2C data line to ADV7185
input source; // 0: composite, 1: s-video

initial begin
   $display("ADV7185 Initialization values:");
   $display("  Register 0:  0x%X", `ADV7185_REGISTER_0);
   $display("  Register 1:  0x%X", `ADV7185_REGISTER_1);
   $display("  Register 2:  0x%X", `ADV7185_REGISTER_2);
   $display("  Register 3:  0x%X", `ADV7185_REGISTER_3);
   $display("  Register 4:  0x%X", `ADV7185_REGISTER_4);
   $display("  Register 5:  0x%X", `ADV7185_REGISTER_5);
   $display("  Register 7:  0x%X", `ADV7185_REGISTER_7);
   $display("  Register 8:  0x%X", `ADV7185_REGISTER_8);
   $display("  Register 9:  0x%X", `ADV7185_REGISTER_9);
   $display("  Register A:  0x%X", `ADV7185_REGISTER_A);
   $display("  Register B:  0x%X", `ADV7185_REGISTER_B);
   $display("  Register C:  0x%X", `ADV7185_REGISTER_C);
   $display("  Register D:  0x%X", `ADV7185_REGISTER_D);
   $display("  Register E:  0x%X", `ADV7185_REGISTER_E);
   $display("  Register F:  0x%X", `ADV7185_REGISTER_F);
   $display("  Register 33: 0x%X", `ADV7185_REGISTER_33);
end

//
// Generate a 1MHz for the I2C driver (resulting I2C clock rate is 250kHz)
//

reg [7:0] clk_div_count, reset_count;
reg clock_slow;
wire reset_slow;

initial
  begin
    clk_div_count <= 8'h00;
    // synthesis attribute init of clk_div_count is "00";
    clock_slow <= 1'b0;
    // synthesis attribute init of clock_slow is "0";
  end

always @(posedge clock_27mhz)
  if (clk_div_count == 26)
    begin
      clock_slow <= ~clock_slow;
```

```
      clk_div_count <= 0;
    end
  else
    clk_div_count <= clk_div_count+1;

always @(posedge clock_27mhz)
  if (reset)
    reset_count <= 100;
  else
    reset_count <= (reset_count==0) ? 0 : reset_count-1;

assign reset_slow = reset_count != 0;

//
// I2C driver
//

reg load;
reg [7:0] data;
wire ack, idle;

i2c i2c(.reset(reset_slow), .clock4x(clock_slow), .data(data), .load(load),
       .ack(ack), .idle(idle), .scl(tv_in_i2c_clock),
       .sda(tv_in_i2c_data));

//
// State machine
//

reg [7:0] state;
reg tv_in_reset_b;
reg old_source;

always @(posedge clock_slow)
   if (reset_slow)
    begin
       state <= 0;
       load <= 0;
       tv_in_reset_b <= 0;
       old_source <= 0;
    end
   else
    case (state)
      8'h00:
        begin
           // Assert reset
           load <= 1'b0;
           tv_in_reset_b <= 1'b0;
```

```
      if (!ack)
       state <= state+1;
  end
8'h01:
  state <= state+1;
8'h02:
  begin
     // Release reset
     tv_in_reset_b <= 1'b1;
     state <= state+1;
        end
8'h03:
  begin
     // Send ADV7185 address
     data <= 8'h8A;
     load <= 1'b1;
     if (ack)
      state <= state+1;
  end
8'h04:
  begin
     // Send subaddress of first register
     data <= 8'h00;
     if (ack)
      state <= state+1;
  end
8'h05:
  begin
     // Write to register 0
     data <= `ADV7185_REGISTER_0 | {5'h00, {3{source}}};
     if (ack)
      state <= state+1;
  end
8'h06:
  begin
     // Write to register 1
     data <= `ADV7185_REGISTER_1;
     if (ack)
      state <= state+1;
  end
8'h07:
  begin
     // Write to register 2
     data <= `ADV7185_REGISTER_2;
     if (ack)
      state <= state+1;
  end
8'h08:
```

```
  begin
     // Write to register 3
     data <= `ADV7185_REGISTER_3;
     if (ack)
      state <= state+1;
  end
8'h09:
  begin
     // Write to register 4
     data <= `ADV7185_REGISTER_4;
     if (ack)
      state <= state+1;
  end
8'h0A:
  begin
     // Write to register 5
     data <= `ADV7185_REGISTER_5;
     if (ack)
      state <= state+1;
  end
8'h0B:
  begin
     // Write to register 6
     data <= 8'h00; // Reserved register, write all zeros
     if (ack)
      state <= state+1;
  end
8'h0C:
  begin
     // Write to register 7
     data <= `ADV7185_REGISTER_7;
     if (ack)
      state <= state+1;
  end
8'h0D:
  begin
     // Write to register 8
     data <= `ADV7185_REGISTER_8;
     if (ack)
      state <= state+1;
  end
8'h0E:
  begin
     // Write to register 9
     data <= `ADV7185_REGISTER_9;
     if (ack)
      state <= state+1;
  end
```

```
8'h0F: begin
   // Write to register A
   data <= `ADV7185_REGISTER_A;
 if (ack)
   state <= state+1;
end
8'h10:
  begin
     // Write to register B
     data <= `ADV7185_REGISTER_B;
     if (ack)
      state <= state+1;
  end
8'h11:
  begin
     // Write to register C
     data <= `ADV7185_REGISTER_C;
     if (ack)
      state <= state+1;
  end
8'h12:
  begin
     // Write to register D
     data <= `ADV7185_REGISTER_D;
     if (ack)
      state <= state+1;
  end
8'h13:
  begin
     // Write to register E
     data <= `ADV7185_REGISTER_E;
     if (ack)
      state <= state+1;
  end
8'h14:
  begin
     // Write to register F
     data <= `ADV7185_REGISTER_F;
     if (ack)
      state <= state+1;
  end
8'h15:
  begin
     // Wait for I2C transmitter to finish
     load <= 1'b0;
     if (idle)
      state <= state+1;
  end
```

```
8'h16:
  begin
     // Write address
     data <= 8'h8A;
     load <= 1'b1;
     if (ack)
      state <= state+1;
  end
8'h17:
  begin
     data <= 8'h33;
     if (ack)
      state <= state+1;
  end
8'h18:
  begin
     data <= `ADV7185_REGISTER_33;
     if (ack)
      state <= state+1;
  end
8'h19:
  begin
     load <= 1'b0;
     if (idle)
      state <= state+1;
  end

8'h1A: begin
   data <= 8'h8A;
   load <= 1'b1;
   if (ack)
     state <= state+1;
end
8'h1B:
  begin
     data <= 8'h33;
     if (ack)
      state <= state+1;
  end
8'h1C:
  begin
     load <= 1'b0;
     if (idle)
      state <= state+1;
  end
8'h1D:
  begin
     load <= 1'b1;
```

```
            data <= 8'h8B;
            if (ack)
             state <= state+1;
          end
        8'h1E:
          begin
            data <= 8'hFF;
            if (ack)
             state <= state+1;
          end
        8'h1F:
          begin
            load <= 1'b0;
            if (idle)
             state <= state+1;
          end
        8'h20:
          begin
            // Idle
            if (old_source != source) state <= state+1;
            old_source <= source;
          end
        8'h21: begin
           // Send ADV7185 address
           data <= 8'h8A;
           load <= 1'b1;
           if (ack) state <= state+1;
         end
        8'h22: begin
           // Send subaddress of register 0
           data <= 8'h00;
           if (ack) state <= state+1;
         end
        8'h23: begin
           // Write to register 0
           data <= `ADV7185_REGISTER_0 | {5'h00, {3{source}}};
           if (ack) state <= state+1;
         end
        8'h24: begin
           // Wait for I2C transmitter to finish
           load <= 1'b0;
           if (idle) state <= 8'h20;
         end
       endcase

endmodule

// i2c module for use with the ADV7185
```

```
module i2c (reset, clock4x, data, load, idle, ack, scl, sda);

   input reset;
   input clock4x;
   input [7:0] data;
   input load;
   output ack;
   output idle;
   output scl;
   output sda;

   reg [7:0] ldata;
   reg ack, idle;
   reg scl;
   reg sdai;

   reg [7:0] state;

   assign sda = sdai ? 1'bZ : 1'b0;

   always @(posedge clock4x)
     if (reset)
       begin
         state <= 0;
         ack <= 0;
       end
     else
       case (state)
        8'h00: // idle
          begin
             scl <= 1'b1;
             sdai <= 1'b1;
             ack <= 1'b0;
             idle <= 1'b1;
             if (load)
              begin
                 ldata <= data;
                 ack <= 1'b1;
                 state <= state+1;
              end
          end
        8'h01: // Start
          begin
             ack <= 1'b0;
             idle <= 1'b0;
             sdai <= 1'b0;
             state <= state+1;
```

```
  end
8'h02:
  begin
     scl <= 1'b0;
     state <= state+1;
  end
8'h03: // Send bit 7
  begin
     ack <= 1'b0;
     sdai <= ldata[7];
     state <= state+1;
  end
8'h04:
  begin
     scl <= 1'b1;
     state <= state+1;
  end
8'h05:
  begin
     state <= state+1;
  end
8'h06:
  begin
     scl <= 1'b0;
     state <= state+1;
  end
8'h07:
  begin
     sdai <= ldata[6];
     state <= state+1;
  end
8'h08:
  begin
     scl <= 1'b1;
     state <= state+1;
  end
8'h09:
  begin
     state <= state+1;
  end
8'h0A:
  begin
     scl <= 1'b0;
     state <= state+1;
  end
8'h0B:
  begin
     sdai <= ldata[5];
```

```
      state <= state+1;
    end
  8'h0C:
    begin
      scl <= 1'b1;
      state <= state+1;
    end
  8'h0D:
    begin
      state <= state+1;
    end
  8'h0E:
    begin
      scl <= 1'b0;
      state <= state+1;
    end
  8'h0F:
    begin
      sdai <= ldata[4];
      state <= state+1;
    end
  8'h10:
    begin
      scl <= 1'b1;
      state <= state+1;
    end
  8'h11:
    begin
      state <= state+1;
    end
  8'h12:
    begin
      scl <= 1'b0;
      state <= state+1;
    end
  8'h13:
    begin
      sdai <= ldata[3];
      state <= state+1;
    end
  8'h14:
    begin
      scl <= 1'b1;
      state <= state+1;
    end
  8'h15:
    begin
      state <= state+1;
```

```
  end
8'h16:
  begin
     scl <= 1'b0;
     state <= state+1;
  end
8'h17:
  begin
     sdai <= ldata[2];
     state <= state+1;
  end
8'h18:
  begin
     scl <= 1'b1;
     state <= state+1;
  end
8'h19:
  begin
     state <= state+1;
  end
8'h1A:
  begin
     scl <= 1'b0;
     state <= state+1;
  end
8'h1B:
  begin
     sdai <= ldata[1];
     state <= state+1;
  end
8'h1C:
  begin
     scl <= 1'b1;
     state <= state+1;
  end
8'h1D:
  begin
     state <= state+1;
  end
8'h1E:
  begin
     scl <= 1'b0;
     state <= state+1;
  end
8'h1F:
  begin
     sdai <= ldata[0];
     state <= state+1;
```

```
  end
8'h20:
  begin
     scl <= 1'b1;
     state <= state+1;
  end
8'h21:
  begin
     state <= state+1;
  end
8'h22:
  begin
     scl <= 1'b0;
     state <= state+1;
  end
8'h23: // Acknowledge bit
  begin
     state <= state+1;
  end
8'h24:
  begin
     scl <= 1'b1;
     state <= state+1;
  end
8'h25:
  begin
     state <= state+1;
  end
8'h26:
  begin
     scl <= 1'b0;
     if (load)
      begin
         ldata <= data;
         ack <= 1'b1;
         state <= 3;
      end
     else
      state <= state+1;
  end
8'h27:
  begin
     sdai <= 1'b0;
     state <= state+1;
  end
8'h28:
  begin
     scl <= 1'b1;
```

```
            state <= state+1;
          end
        8'h29:
          begin
            sdai <= 1'b1;
            state <= 0;
          end
       endcase

endmodule
```

ycrcb2rgb.v

```
/*************************************************************************
 **
 ** Module: ycrcb2rgb
 **
 ** Generic Equations:
 *************************************************************************/

module YCrCb2RGB ( R, G, B, clk, rst, Y, Cr, Cb );

output [7:0]  R, G, B;

input clk,rst;
input[9:0] Y, Cr, Cb;

wire [7:0] R,G,B;
reg [20:0] R_int,G_int,B_int,X_int,A_int,B1_int,B2_int,C_int;
reg [9:0] const1,const2,const3,const4,const5;
reg[9:0] Y_reg, Cr_reg, Cb_reg;

//registering constants
always @ (posedge clk)
begin
 const1 = 10'b 0100101010; //1.164 = 01.00101010
 const2 = 10'b 0110011000; //1.596 = 01.10011000
 const3 = 10'b 0011010000; //0.813 = 00.11010000
 const4 = 10'b 0001100100; //0.392 = 00.01100100
 const5 = 10'b 1000000100; //2.017 = 10.00000100
end

always @ (posedge clk or posedge rst)
   if (rst)
      begin
      Y_reg <= 0; Cr_reg <= 0; Cb_reg <= 0;
      end
   else
```

```
      begin
        Y_reg <= Y; Cr_reg <= Cr; Cb_reg <= Cb;
      end

always @ (posedge clk or posedge rst)
   if (rst)
      begin
       A_int <= 0; B1_int <= 0; B2_int <= 0; C_int <= 0; X_int <= 0;
      end
   else
     begin
     X_int <= (const1 * (Y_reg - 'd64)) ;
     A_int <= (const2 * (Cr_reg - 'd512));
     B1_int <= (const3 * (Cr_reg - 'd512));
     B2_int <= (const4 * (Cb_reg - 'd512));
     C_int <= (const5 * (Cb_reg - 'd512));
     end

always @ (posedge clk or posedge rst)
   if (rst)
      begin
       R_int <= 0; G_int <= 0; B_int <= 0;
      end
   else
     begin
     R_int <= X_int + A_int;
     G_int <= X_int - B1_int - B2_int;
     B_int <= X_int + C_int;
     end



/*always @ (posedge clk or posedge rst)
   if (rst)
      begin
       R_int <= 0; G_int <= 0; B_int <= 0;
      end
   else
     begin
     X_int <= (const1 * (Y_reg - 'd64)) ;
     R_int <= X_int + (const2 * (Cr_reg - 'd512));
     G_int <= X_int - (const3 * (Cr_reg - 'd512)) - (const4 * (Cb_reg - 'd512));
     B_int <= X_int + (const5 * (Cb_reg - 'd512));
     end

*/
/* limit output to 0 - 4095, <0 equals o and >4095 equals 4095 */
assign R = (R_int[20]) ? 0 : (R_int[19:18] == 2'b0) ? R_int[17:10] : 8'b11111111;
```

```
assign G = (G_int[20]) ? 0 : (G_int[19:18] == 2'b0) ? G_int[17:10] : 8'b11111111;
assign B = (B_int[20]) ? 0 : (B_int[19:18] == 2'b0) ? B_int[17:10] : 8'b11111111;

endmodule
```

zbt_6111.v

```
//
// File:   zbt_6111.v
// Date:   27-Nov-05
// Author: I. Chuang <ichuang@mit.edu>
//
// Simple ZBT driver for the MIT 6.111 labkit, which does not hide the
// pipeline delays of the ZBT from the user.  The ZBT memories have
// two cycle latencies on read and write, and also need extra-long data hold
// times around the clock positive edge to work reliably.
//

////////////////////////////////////////////////////////////////////////////////
// Ike's simple ZBT RAM driver for the MIT 6.111 labkit
//
// Data for writes can be presented and clocked in immediately; the actual
// writing to RAM will happen two cycles later.
//
// Read requests are processed immediately, but the read data is not available
// until two cycles after the intial request.
//
// A clock enable signal is provided; it enables the RAM clock when high.

module zbt_6111(clk, cen, we, addr, write_data, read_data,
                ram_clk, ram_we_b, ram_address, ram_data, ram_cen_b);

   input clk;             // system clock
   input cen;             // clock enable for gating ZBT cycles
   input we;              // write enable (active HIGH)
   input [18:0] addr;           // memory address
   input [35:0] write_data;     // data to write
   output [35:0] read_data;     // data read from memory
   output     ram_clk;    // physical line to ram clock
   output     ram_we_b;   // physical line to ram we_b
   output [18:0] ram_address;    // physical line to ram address
   inout [35:0]  ram_data;// physical line to ram data
   output     ram_cen_b;  // physical line to ram clock enable

   // clock enable (should be synchronous and one cycle high at a time)
```

```
   wire       ram_cen_b = ~cen;

   // create delayed ram_we signal: note the delay is by two cycles!
   // ie we present the data to be written two cycles after we is raised
   // this means the bus is tri-stated two cycles after we is raised.

   reg [1:0]   we_delay;

   always @(posedge clk)
     we_delay <= cen ? {we_delay[0],we} : we_delay;

   // create two-stage pipeline for write data

   reg [35:0]  write_data_old1;
   reg [35:0]  write_data_old2;
   always @(posedge clk)
     if (cen)
       {write_data_old2, write_data_old1} <= {write_data_old1, write_data};

   // wire to ZBT RAM signals

   assign      ram_we_b = ~we;
   assign      ram_clk = 1'b0;  // gph 2011-Nov-10
                                // set to zero as place holder

//   assign      ram_clk = ~clk;     // RAM is not happy with our data hold
                                   // times if its clk edges equal FPGA's
                                   // so we clock it on the falling edges
                                   // and thus let data stabilize longer
   assign      ram_address = addr;

   assign      ram_data = we_delay[1] ? write_data_old2 : {36{1'bZ}};
   assign      read_data = ram_data;

endmodule // zbt_6111
```

zbt_6111_sample.v

```
//
// File:   zbt_6111_sample.v
// Date:   26-Nov-05
// Author: I. Chuang <ichuang@mit.edu>
//
// Sample code for the MIT 6.111 labkit demonstrating use of the ZBT
// memories for video display.  Video input from the NTSC digitizer is
// displayed within an XGA 1024x768 window.  One ZBT memory (ram0) is used
// as the video frame buffer, with 8 bits used per pixel (black & white).
//
```

```
// Since the ZBT is read once for every four pixels, this frees up time for
// data to be stored to the ZBT during other pixel times.  The NTSC decoder
// runs at 27 MHz, whereas the XGA runs at 65 MHz, so we synchronize
// signals between the two (see ntsc2zbt.v) and let the NTSC data be
// stored to ZBT memory whenever it is available, during cycles when
// pixel reads are not being performed.
//
// We use a very simple ZBT interface, which does not involve any clock
// generation or hiding of the pipelining.  See zbt_6111.v for more info.
//
// switch[7] selects between display of NTSC video and test bars
// switch[6] is used for testing the NTSC decoder
// switch[1] selects between test bar periods; these are stored to ZBT
//           during blanking periods
// switch[0] selects vertical test bars (hardwired; not stored in ZBT)
//
//
// Bug fix: Jonathan P. Mailoa <jpmailoa@mit.edu>
// Date   : 11-May-09
//
// Use ramclock module to deskew clocks;  GPH
// To change display from 1024*787 to 800*600, use clock_40mhz and change
// accordingly. Verilog ntsc2zbt.v will also need changes to change resolution.
//
// Date   : 10-Nov-11

//////////////////////////////////////////////////////////////////////////////
//
// 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)
//
// 2011-Nov-10: Changed resolution to 1024 * 768.
//                                Added back ramclok to deskew RAM clock
//
// 2009-May-11: Fixed memory management bug by 8 clock cycle forecast.
//              Changed resolution to  800 * 600.
//              Reduced clock speed to 40MHz.
//              Disconnected zbt_6111's ram_clk signal.
//              Added ramclock to control RAM.
//              Added notes about ram1 default values.
//              Commented out clock_feedback_out assignment.
//              Removed delayN modules because ZBT's latency has no more effect.
//
// 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
```

```
//
//////////////////////////////////////////////////////////////////////////////

module zbt_6111_sample(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, audio_reset_b, ac97_synch, ac97_sdata_out;
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

   // 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'b1;
   assign tv_in_fifo_clock = 1'b0;
   assign tv_in_iso = 1'b1;
   //assign tv_in_reset_b = 1'b0;
   assign tv_in_clock = clock_27mhz;//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

/* change lines below to enable ZBT RAM bank0 */

/*
   assign ram0_data = 36'hZ;
   assign ram0_address = 19'h0;
   assign ram0_clk = 1'b0;
   assign ram0_we_b = 1'b1;
   assign ram0_cen_b = 1'b0;     // clock enable
*/

/* enable RAM pins */

   assign ram0_ce_b = 1'b0;
   assign ram0_oe_b = 1'b0;
   assign ram0_adv_ld = 1'b0;
   assign ram0_bwe_b = 4'h0;

/**********/

   assign ram1_data = 36'hZ;
   assign ram1_address = 19'h0;
   assign ram1_adv_ld = 1'b0;
   assign ram1_clk = 1'b0;

   //These values has to be set to 0 like ram0 if ram1 is used.
   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;

   // clock_feedback_out will be assigned by ramclock
   // assign clock_feedback_out = 1'b0;  //2011-Nov-10
   // 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 = 16'h0;
   assign analyzer1_clock = 1'b1;
   assign analyzer2_data = 16'h0;
   assign analyzer2_clock = 1'b1;
   assign analyzer3_data = 16'h0;
   assign analyzer3_clock = 1'b1;
   assign analyzer4_data = 16'h0;
   assign analyzer4_clock = 1'b1;

   ////////////////////////////////////////////////////////////////////////
   // Demonstration of ZBT RAM as video memory

   // use FPGA's digital clock manager to produce a
   // 65MHz clock (actually 64.8MHz)
   wire clock_65mhz_unbuf,clock_65mhz;
   DCM vclk1(.CLKIN(clock_27mhz),.CLKFX(clock_65mhz_unbuf));
   // synthesis attribute CLKFX_DIVIDE of vclk1 is 10
   // synthesis attribute CLKFX_MULTIPLY of vclk1 is 24
   // synthesis attribute CLK_FEEDBACK of vclk1 is NONE
   // synthesis attribute CLKIN_PERIOD of vclk1 is 37
   BUFG vclk2(.O(clock_65mhz),.I(clock_65mhz_unbuf));

//   wire clk = clock_65mhz;  // gph 2011-Nov-10

/*   ////////////////////////////////////////////////////////////////////////
   // Demonstration of ZBT RAM as video memory

   // use FPGA's digital clock manager to produce a
   // 40MHz clock (actually 40.5MHz)
   wire clock_40mhz_unbuf,clock_40mhz;
   DCM vclk1(.CLKIN(clock_27mhz),.CLKFX(clock_40mhz_unbuf));
   // synthesis attribute CLKFX_DIVIDE of vclk1 is 2
   // synthesis attribute CLKFX_MULTIPLY of vclk1 is 3
   // synthesis attribute CLK_FEEDBACK of vclk1 is NONE
   // synthesis attribute CLKIN_PERIOD of vclk1 is 37
   BUFG vclk2(.O(clock_40mhz),.I(clock_40mhz_unbuf));
```

```
   wire clk = clock_40mhz;
*/
       wire locked;
       //assign clock_feedback_out = 0; // gph 2011-Nov-10

   ramclock rc(.ref_clock(clock_65mhz), .fpga_clock(clk),
                                .ram0_clock(ram0_clk),
                                //.ram1_clock(ram1_clk),   //uncomment if ram1 is
used
                                .clock_feedback_in(clock_feedback_in),
                                .clock_feedback_out(clock_feedback_out),
.locked(locked));


   // power-on reset generation
   wire power_on_reset;    // remain high for first 16 clocks
   SRL16 reset_sr (.D(1'b0), .CLK(clk), .Q(power_on_reset),
                .A0(1'b1), .A1(1'b1), .A2(1'b1), .A3(1'b1));
   defparam reset_sr.INIT = 16'hFFFF;

   // ENTER button is user reset
   wire reset,user_reset;
   debounce db1(power_on_reset, clk, ~button_enter, user_reset);
   assign reset = user_reset | power_on_reset;

   // display module for debugging

   wire [63:0] dispdata;
   display_16hex hexdisp1(reset, clk, dispdata,
                     disp_blank, disp_clock, disp_rs, disp_ce_b,
                     disp_reset_b, disp_data_out);

   // generate basic XVGA video signals
   wire [10:0] hcount;
   wire [9:0]  vcount;
   wire hsync,vsync,blank;
   xvga xvga1(clk,hcount,vcount,hsync,vsync,blank);

   // wire up to ZBT ram

   wire [35:0] vram_write_data;
   wire [35:0] vram_read_data;
   wire [18:0] vram_addr;
   wire        vram_we;

   wire ram0_clk_not_used;
   zbt_6111 zbt1(clk, 1'b1, vram_we, vram_addr,
```

```
                vram_write_data, vram_read_data,
                ram0_clk_not_used,   //to get good timing, don't connect ram_clk to
zbt_6111
                ram0_we_b, ram0_address, ram0_data, ram0_cen_b);

   // generate pixel value from reading ZBT memory
   wire [23:0]      vr_pixel;
   wire [18:0]      vram_addr1;

   vram_display vd1(reset,clk,hcount,vcount,vr_pixel,
                vram_addr1,vram_read_data);

   // ADV7185 NTSC decoder interface code
   // adv7185 initialization module
   adv7185init adv7185(.reset(reset), .clock_27mhz(clock_27mhz),
                   .source(1'b0), .tv_in_reset_b(tv_in_reset_b),
                   .tv_in_i2c_clock(tv_in_i2c_clock),
                   .tv_in_i2c_data(tv_in_i2c_data));

   wire [29:0] ycrcb;     // video data (luminance, chrominance)
   wire [2:0] fvh;  // sync for field, vertical, horizontal
   wire       dv;   // data valid

   ntsc_decode decode (.clk(tv_in_line_clock1), .reset(reset),
                   .tv_in_ycrcb(tv_in_ycrcb[19:10]),
                   .ycrcb(ycrcb), .f(fvh[2]),
                   .v(fvh[1]), .h(fvh[0]), .data_valid(dv));

   // code to write NTSC data to video memory

   wire [17:0] downsampled_ycrcb;
   wire [18:0] ntsc_addr;
   wire [35:0] ntsc_data;
   wire        ntsc_we;

   assign downsampled_ycrcb = {ycrcb[29:24], ycrcb[19:14], ycrcb[9:4]};
   ntsc_to_zbt n2z (clk, tv_in_line_clock1, fvh, dv, downsampled_ycrcb,
                ntsc_addr, ntsc_data, ntsc_we, 1'b0);

   // code to write pattern to ZBT memory
   reg [31:0]       count;
   always @(posedge clk) count <= reset ? 0 : count + 1;

   wire [18:0]      vram_addr2 = count[0+18:0];
   wire [35:0]      vpat = ( 1'b0 ? {4{count[3+3:3],4'b0}}
                    : {4{count[3+4:4],4'b0}} );

   // mux selecting read/write to memory based on which write-enable is chosen
```

```
   wire      sw_ntsc = 1'b1;
   wire      my_we = sw_ntsc ? (hcount[1:0]==2'd2) : blank;
   wire [18:0]      write_addr = sw_ntsc ? ntsc_addr : vram_addr2;
   wire [35:0]      write_data = sw_ntsc ? ntsc_data : vpat;

//   wire    write_enable = sw_ntsc ? (my_we & ntsc_we) : my_we;
//   assign  vram_addr = write_enable ? write_addr : vram_addr1;
//   assign  vram_we = write_enable;

   assign    vram_addr = my_we ? write_addr : vram_addr1;
   assign    vram_we = my_we;
   assign    vram_write_data = write_data;

   // calibration
   reg signed [7:0] xoffset;
   reg signed [7:0] yoffset;
   reg [7:0] xscale;
   reg [7:0] yscale;
   reg [7:0] threshold;
   wire button_left_clean, button_right_clean, button_up_clean, button_down_clean;
   wire button_threshold_clean, button_xscale_clean, button_yscale_clean;
   reg move_up, move_down, move_left, move_right;
   debounce dbl(.reset(reset), .clk(clk), .noisy(button_left),
.clean(button_left_clean));
   debounce dbr(.reset(reset), .clk(clk), .noisy(button_right),
.clean(button_right_clean));
   debounce dbu(.reset(reset), .clk(clk), .noisy(button_up),
.clean(button_up_clean));
   debounce dbd(.reset(reset), .clk(clk), .noisy(button_down),
.clean(button_down_clean));
   debounce dbt(.reset(reset), .clk(clk), .noisy(!button0),
.clean(button_threshold_clean));
   debounce dbxs(.reset(reset), .clk(clk), .noisy(!button1),
.clean(button_xscale_clean));
   debounce dbys(.reset(reset), .clk(clk), .noisy(!button2),
.clean(button_yscale_clean));
   assign dispdata = {8'd0, 8'd0, 8'd0, yscale, 8'd0, xscale, 8'd0, threshold};

   always @(posedge clk) begin
      if (reset) begin
         xoffset <= 0;
         yoffset <= 0;
         move_up <= 0;
         move_down <= 0;
         move_left <= 0;
         move_right <= 0;
         xscale <= 72;
```

```
      yscale <= 100;
      threshold <= 32;
   end else begin
      if (button_up_clean && !move_up) begin
         move_up <= 1;
      end else if (!button_up_clean && move_up) begin
         move_up <= 0;
         yoffset <= yoffset - 1;
      end else if (button_down_clean && !move_down) begin
         move_down <= 1;
      end else if (!button_down_clean && move_down) begin
         move_down <= 0;
         yoffset <= yoffset + 1;
      end

      if (button_left_clean && !move_left) begin
         move_left <= 1;
      end else if (!button_left_clean && move_left) begin
         move_left <= 0;
         xoffset <= xoffset - 1;
      end else if (button_right_clean && !move_right) begin
         move_right <= 1;
      end else if (!button_right_clean && move_right) begin
         move_right <= 0;
         xoffset <= xoffset + 1;
      end

      if (button_threshold_clean) begin
         threshold <= switch;
      end else if (button_xscale_clean) begin
         xscale <= switch;
      end else if (button_yscale_clean) begin
         yscale <= switch;
      end
   end
end

// select output pixel data
reg [23:0] pixel;
reg       b,hs,vs;
wire ol_draw;
wire ol_box;
overlay ol(.vclock(clock_65mhz),.reset(reset),
           .uart_rcv(user4[0]),
           .hcount(hcount),.vcount(vcount),
           .hsync(hsync),.vsync(vsync),.blank(blank),
           .border(ol_draw),
           .solid(ol_box),
```

```
                .threshold(threshold),
                .xscale({1'b0, xscale}),
                .yscale({1'b0, yscale}),
                .xoffset(xoffset),
                .yoffset(yoffset));

   wire bug1_draw;
   wire bug2_draw;
   wire bug3_draw;
   wire bug1_die = ol_box && bug1_draw;
   wire bug2_die = ol_box && bug2_draw;
   wire bug3_die = ol_box && bug3_draw;
   wire bug1_dead;
   wire bug2_dead;
   wire bug3_dead;
   reg [7:0] score = 0;
   bugEquals bugequals(.clk(clock_65mhz), .reset(reset), .die(bug1_die),
.hcount(hcount), .vcount(vcount), .draw(bug1_draw), .dead(bug1_dead));
   bugJS bugjs(.clk(clock_65mhz), .reset(reset), .die(bug2_die), .hcount(hcount),
.vcount(vcount), .draw(bug2_draw), .dead(bug2_dead));
   bugString bugstr(.clk(clock_65mhz), .reset(reset), .die(bug3_die),
.hcount(hcount), .vcount(vcount), .draw(bug3_draw), .dead(bug3_dead));

   wire score_draw;
   score_tracker st(.clk(clk), .reset(reset), .score(score), .hcount(hcount),
.vcount(vcount), .draw(score_draw));

   reg [2:0] bug_dead_last;
   wire [2:0] dead_posedge;

   assign dead_posedge[0] = bug1_dead && !bug_dead_last[0];
   assign dead_posedge[1] = bug2_dead && !bug_dead_last[1];
   assign dead_posedge[2] = bug3_dead && !bug_dead_last[2];

   always @(posedge clk) begin
      bug_dead_last[0] <= bug1_dead;
      bug_dead_last[1] <= bug2_dead;
      bug_dead_last[2] <= bug3_dead;
      if (reset) begin
         score <= 0;
      end else begin
         score <= score + dead_posedge[0] + dead_posedge[1] + dead_posedge[2];
      end
   end

   always @(posedge clk) begin
      if (ol_draw) begin
         pixel <= 24'h0000FF;
```

```
      end else if (bug1_draw) begin
         if (bug1_dead) begin
            pixel <= 24'hFF0000;
         end else begin
            pixel <= 24'h00FF00;
         end
      end else if (bug2_draw) begin
         if (bug2_dead) begin
            pixel <= 24'hFF0000;
         end else begin
            pixel <= 24'h00FF00;
         end
      end else if (bug3_draw) begin
         if (bug3_dead) begin
            pixel <= 24'hFF0000;
         end else begin
            pixel <= 24'h00FF00;
         end
      end else if (score_draw) begin
         pixel <= 24'hFFFFFF;
      end else begin
         pixel <= vr_pixel;
      end
      b <= blank;
      hs <= hsync;
      vs <= vsync;
   end

   // VGA Output.  In order to meet the setup and hold times of the
   // AD7125, we send it ~clk.
   assign vga_out_red = pixel[23:16];
   assign vga_out_green = pixel[15:8];
   assign vga_out_blue = pixel[7:0];
   assign vga_out_sync_b = 1'b1;    // not used
   assign vga_out_pixel_clock = ~clk;
   assign vga_out_blank_b = ~b;
   assign vga_out_hsync = hs;
   assign vga_out_vsync = vs;

endmodule

//////////////////////////////////////////////////////////////////////////////////
// xvga: Generate XVGA display signals (1024 x 768 @ 60Hz)

module xvga(vclock,hcount,vcount,hsync,vsync,blank);
   input vclock;
   output [10:0] hcount;
   output [9:0] vcount;
```

```
   output    vsync;
   output    hsync;
   output    blank;

   reg         hsync,vsync,hblank,vblank,blank;
   reg [10:0]       hcount;    // pixel number on current line
   reg [9:0] vcount;       // line number

   // horizontal: 1344 pixels total
   // display 1024 pixels per line
   wire      hsyncon,hsyncoff,hreset,hblankon;
   assign    hblankon = (hcount == 1023);
   assign    hsyncon = (hcount == 1047);
   assign    hsyncoff = (hcount == 1183);
   assign    hreset = (hcount == 1343);

   // vertical: 806 lines total
   // display 768 lines
   wire      vsyncon,vsyncoff,vreset,vblankon;
   assign    vblankon = hreset & (vcount == 767);
   assign    vsyncon = hreset & (vcount == 776);
   assign    vsyncoff = hreset & (vcount == 782);
   assign    vreset = hreset & (vcount == 805);

   // sync and blanking
   wire      next_hblank,next_vblank;
   assign next_hblank = hreset ? 0 : hblankon ? 1 : hblank;
   assign next_vblank = vreset ? 0 : vblankon ? 1 : vblank;
   always @(posedge vclock) begin
      hcount <= hreset ? 0 : hcount + 1;
      hblank <= next_hblank;
      hsync <= hsyncon ? 0 : hsyncoff ? 1 : hsync;  // active low

      vcount <= hreset ? (vreset ? 0 : vcount + 1) : vcount;
      vblank <= next_vblank;
      vsync <= vsyncon ? 0 : vsyncoff ? 1 : vsync;  // active low

      blank <= next_vblank | (next_hblank & ~hreset);
   end
endmodule

/*
/////////////////////////////////////////////////////////////////////////////////
// xvga: Generate XVGA display signals (800 x 600 @ 60Hz)

module xvga(vclock,hcount,vcount,hsyn106 + 525c,vsync,blank);
   input vclock;
   output [10:0] hcount;
```

```
   output [9:0] vcount;
   output    vsync;
   output    hsync;
   output    blank;

   reg         hsync,vsync,hblank,vblank,blank;
   reg [10:0]       hcount;    // pixel number on current line
   reg [9:0] vcount;       // line number

   // horizontal: 1056 pixels total
   // display 800 pixels per line
   wire      hsyncon,hsyncoff,hreset,hblankon;
   assign    hblankon = (hcount == 799);
   assign    hsyncon = (hcount == 839);
   assign    hsyncoff = (hcount == 967);
   assign    hreset = (hcount == 1055);

   // vertical: 628 lines total
   // display 600 lines
   wire      vsyncon,vsyncoff,vreset,vblankon;
   assign    vblankon = hreset & (vcount == 599);
   assign    vsyncon = hreset & (vcount == 600);
   assign    vsyncoff = hreset & (vcount == 604);
   assign    vreset = hreset & (vcount == 627);

   // sync and blanking
   wire      next_hblank,next_vblank;
   assign next_hblank = hreset ? 0 : hblankon ? 1 : hblank;
   assign next_vblank = vreset ? 0 : vblankon ? 1 : vblank;
   always @(posedge vclock) begin
      hcount <= hreset ? 0 : hcount + 1;
      hblank <= next_hblank;
      hsync <= hsyncon ? 0 : hsyncoff ? 1 : hsync;  // active low

      vcount <= hreset ? (vreset ? 0 : vcount + 1) : vcount;
      vblank <= next_vblank;
      vsync <= vsyncon ? 0 : vsyncoff ? 1 : vsync;  // active low

      blank <= next_vblank | (next_hblank & ~hreset);
   end
endmodule */

//////////////////////////////////////////////////////////////////////////
// generate display pixels from reading the ZBT ram
// note that the ZBT ram has 2 cycles of read (and write) latency
//
// We take care of that by latching the data at an appropriate time.
//
```

```
// Note that the ZBT stores 36 bits per word; we use only 32 bits here,
// decoded into four bytes of pixel data.
//
// Bug due to memory management will be fixed. The bug happens because
// memory is called based on current 106 + 525hcount & vcount, which will actually
// shows up 2 cycle in the future. Not to mention that these incoming data
// are latched for 2 cycles before they are used. Also remember that the
// ntsc2zbt's addressing protocol has been fixed.

// The original bug:
// -. At (hcount, vcount) = (100, 201) data at memory address(0,100,49)
//    arrives at vram_read_data, latch it to vr_data_latched.
// -. At (hcount, vcount) = (100, 203) data at memory address(0,100,49)
//    is latched to last_vr_data to be used for display.
// -. Remember that memory address(0,100,49) contains camera data
//    pixel(100,192) - pixel(100,195).
// -. At (hcount, vcount) = (100, 204) camera pixel data(100,192) is shown.
// -. At (hcount, vcount) = (100, 205) camera pixel data(100,193) is shown.
// -. At (hcount, vcount) = (100, 206) camera pixel data(100,194) is shown.
// -. At (hcount, vcount) = (100, 207) camera pixel data(100,195) is shown.
//
// Unfortunately this means that at (hcount == 0) to (hcount == 11) data from
// the right side of the camera is shown instead (including possible sync signals).

// To fix this, two corrections has been made:
// -. Fix addressing protocol in ntsc_to_zbt module.
// -. Forecast hcount & vcount 8 clock cycles ahead and use that
//    instead to call data from ZBT.


module vram_display(reset,clk,hcount,vcount,vr_pixel,
                    vram_addr,vram_read_data);

   input reset, clk;
   input [10:0] hcount;
   input [9:0]      vcount;
   output [23:0] vr_pixel;
   output [18:0] vram_addr;
   input [35:0]  vram_read_data;

   //forecast hcount & vcount 8 clock cycles ahead to get data from ZBT
   wire [10:0] hcount_f = (hcount >= 1048) ? (hcount - 1048) : (hcount + 8);
   wire [9:0] vcount_f = (hcount >= 1048) ? ((vcount == 805) ? 0 : vcount + 1) :
vcount;

   wire [10:0] hcount_t;
   wire [9:0] vcount_t;
   translate_hvcount thvc(.hcount(hcount_f), .vcount(vcount_f), .hcount_t(hcount_t),
```

```
.vcount_t(vcount_t));
   wire [18:0]       vram_addr = {vcount_t, hcount_t[9:1]};

   wire hc2 = hcount[0];
   reg [17:0]  vr_ycrcb;
   reg [35:0]        last_vr_data;

   wire border;

   assign border = (hcount < 148) || (hcount > 850) || (vcount < 142) || (vcount >
631);

   always @(posedge clk)
     last_vr_data <= (hc2 == 1'b1) ? vram_read_data : last_vr_data;

   always @(*)            // each 36-bit word from RAM is decoded to 4 bytes
     begin
        case (hc2)
          2'd0: vr_ycrcb = last_vr_data[17:0];
          2'd1: vr_ycrcb = last_vr_data[35:18];
        endcase
     end

   wire [7:0] R, G, B;
   wire [5:0] Y, Cr, Cb;

   assign Y = vr_ycrcb[17:12];
   assign Cr = vr_ycrcb[11:6];
   assign Cb = vr_ycrcb[5:0];

   YCrCb2RGB ycrcb2rgb(.R(R), .G(G), .B(B), .clk(clk), .rst(1'b0),
      .Y({Y, 4'b0}),
      .Cr({Cr, 4'b0}),
      .Cb({Cb, 4'b0})
      );

   assign vr_pixel = border ? 24'b0 : {R, G, B};
endmodule // vram_display

//////////////////////////////////////////////////////////////////////////////////
// parameterized delay line

module delayN(clk,in,out);
   input clk;
   input in;
   output out;

   parameter NDELAY = 3;
```

```
   reg [NDELAY-1:0] shiftreg;
   wire          out = shiftreg[NDELAY-1];

   always @(posedge clk)
     shiftreg <= {shiftreg[NDELAY-2:0],in};

endmodule // delayN

//////////////////////////////////////////////////////////////////////////////
// ramclock module

//////////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- ZBT RAM clock generation
//
//
// Created: April 27, 2004
// Author: Nathan Ickes
//
//////////////////////////////////////////////////////////////////////////////////
//
// This module generates deskewed clocks for driving the ZBT SRAMs and FPGA
// registers. A special feedback trace on the labkit PCB (which is length
// matched to the RAM traces) is used to adjust the RAM clock phase so that
// rising clock edges reach the RAMs at exactly the same time as rising clock
// edges reach the registers in the FPGA.
//
// The RAM clock signals are driven by DDR output buffers, which further
// ensures that the clock-to-pad delay is the same for the RAM clocks as it is
// for any other registered RAM signal.
//
// When the FPGA is configured, the DCMs are enabled before the chip-level I/O
// drivers are released from tristate. It is therefore necessary to
// artificially hold the DCMs in reset for a few cycles after configuration.
// This is done using a 16-bit shift register. When the DCMs have locked, the
// <lock> output of this mnodule will go high. Until the DCMs are locked, the
// ouput clock timings are not guaranteed, so any logic driven by the
// <fpga_clock> should probably be held inreset until <locked> is high.
//
//////////////////////////////////////////////////////////////////////////////////

module ramclock(ref_clock, fpga_clock, ram0_clock, ram1_clock,
                clock_feedback_in, clock_feedback_out, locked);

   input ref_clock;                 // Reference clock input
   output fpga_clock;               // Output clock to drive FPGA logic
   output ram0_clock, ram1_clock;   // Output clocks for each RAM chip
```

```
input  clock_feedback_in;        // Output to feedback trace
output clock_feedback_out;       // Input from feedback trace
output locked;                   // Indicates that clock outputs are stable

wire  ref_clk, fpga_clk, ram_clk, fb_clk, lock1, lock2, dcm_reset;

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

//To force ISE to compile the ramclock, this line has to be removed.
//IBUFG ref_buf (.O(ref_clk), .I(ref_clock));

    assign ref_clk = ref_clock;

BUFG int_buf (.O(fpga_clock), .I(fpga_clk));

DCM int_dcm (.CLKFB(fpga_clock),
          .CLKIN(ref_clk),
          .RST(dcm_reset),
          .CLK0(fpga_clk),
          .LOCKED(lock1));
// synthesis attribute DLL_FREQUENCY_MODE of int_dcm is "LOW"
// synthesis attribute DUTY_CYCLE_CORRECTION of int_dcm is "TRUE"
// synthesis attribute STARTUP_WAIT of int_dcm is "FALSE"
// synthesis attribute DFS_FREQUENCY_MODE of int_dcm is "LOW"
// synthesis attribute CLK_FEEDBACK of int_dcm  is "1X"
// synthesis attribute CLKOUT_PHASE_SHIFT of int_dcm is "NONE"
// synthesis attribute PHASE_SHIFT of int_dcm is 0

BUFG ext_buf (.O(ram_clock), .I(ram_clk));

IBUFG fb_buf (.O(fb_clk), .I(clock_feedback_in));

DCM ext_dcm (.CLKFB(fb_clk),
             .CLKIN(ref_clk),
             .RST(dcm_reset),
             .CLK0(ram_clk),
             .LOCKED(lock2));
// synthesis attribute DLL_FREQUENCY_MODE of ext_dcm is "LOW"
// synthesis attribute DUTY_CYCLE_CORRECTION of ext_dcm is "TRUE"
// synthesis attribute STARTUP_WAIT of ext_dcm is "FALSE"
// synthesis attribute DFS_FREQUENCY_MODE of ext_dcm is "LOW"
// synthesis attribute CLK_FEEDBACK of ext_dcm  is "1X"
// synthesis attribute CLKOUT_PHASE_SHIFT of ext_dcm is "NONE"
// synthesis attribute PHASE_SHIFT of ext_dcm is 0

SRL16 dcm_rst_sr (.D(1'b0), .CLK(ref_clk), .Q(dcm_reset),
              .A0(1'b1), .A1(1'b1), .A2(1'b1), .A3(1'b1));
// synthesis attribute init of dcm_rst_sr is "000F";
```

```
   OFDDRRSE ddr_reg0 (.Q(ram0_clock), .C0(ram_clock), .C1(~ram_clock),
                  .CE (1'b1), .D0(1'b1), .D1(1'b0), .R(1'b0), .S(1'b0));
   OFDDRRSE ddr_reg1 (.Q(ram1_clock), .C0(ram_clock), .C1(~ram_clock),
                  .CE (1'b1), .D0(1'b1), .D1(1'b0), .R(1'b0), .S(1'b0));
   OFDDRRSE ddr_reg2 (.Q(clock_feedback_out), .C0(ram_clock), .C1(~ram_clock),
                  .CE (1'b1), .D0(1'b1), .D1(1'b0), .R(1'b0), .S(1'b0));

   assign locked = lock1 && lock2;
endmodule
```