Lyne Tchapmi
Szu-Po Wang
Wenting Zheng

D2H
Dr. House at Home
A Portable EKG System

6.111 Final Project

December 13, 2011

Abstract

Dr. House at Home (D2H) is a portable EKG (electrocardiogram) system that seeks to solve the problem of limited portability for medical devices. D2H uses infrared transmission to transmit patient's heartbeat data and remotely analyze the signals from a base station. The data is then displayed in a video format that includes the waveform, patient heartbeat rate, and animation of a beating heart.

Overview

Technology in the biomedical field has been advancing rapidly in the recent years, giving rise to a wonderful diverse set of machines that provide fast, efficient, personalized patient care. However, problems came along with improvements. One of the main problems is limited portability of medical electronics. Currently, many medical machines that are used to monitor patients' vital signs are heavy machines that confine patients within a certain space with limited mobility. In order to solve this problem, one needs a scaled down version of the EKG detection circuitry.

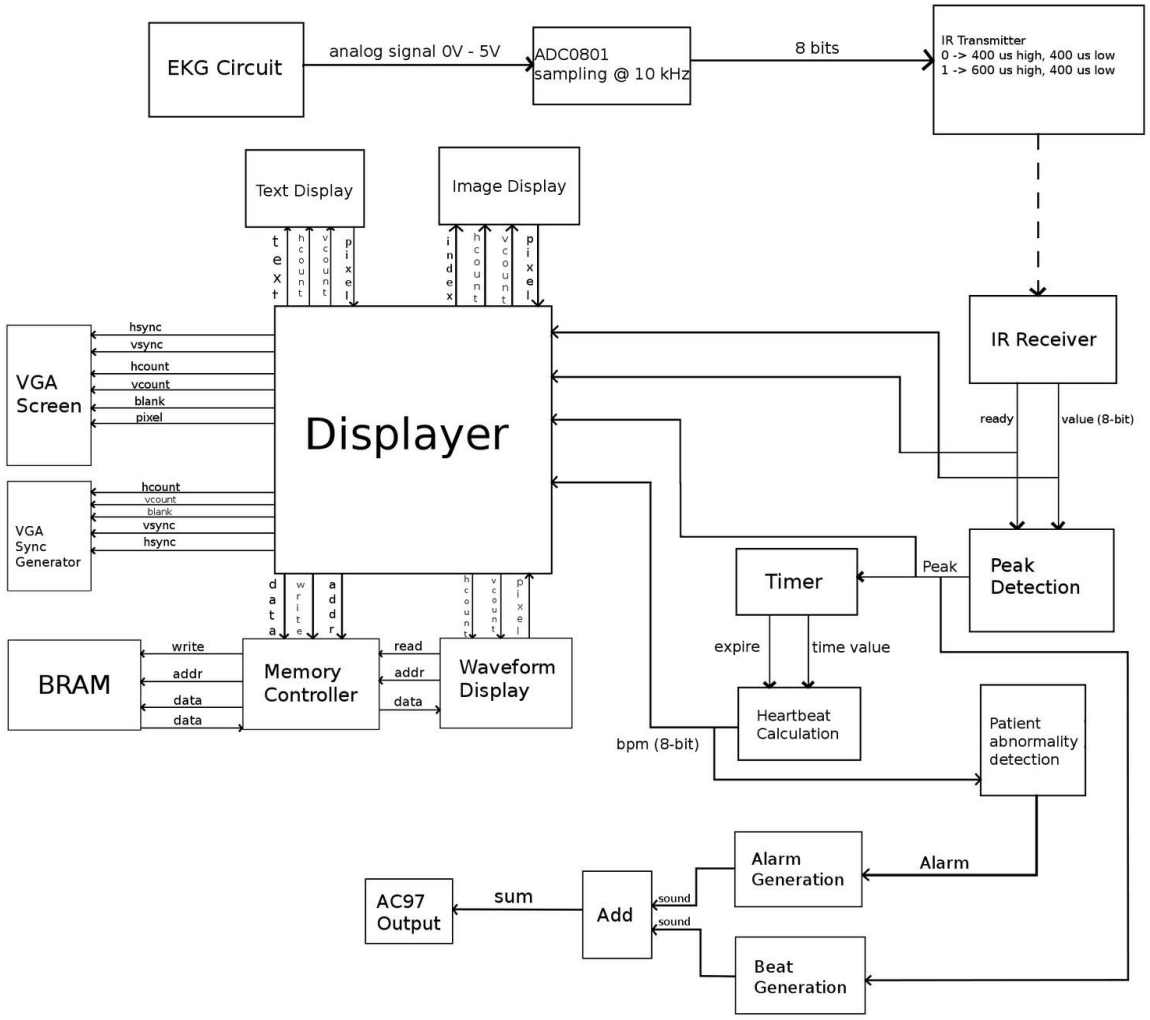
For this project, we prototyped a portable EKG device that measures patients' heartbeat, and wirelessly transmits the information to a remote location using an infrared transmitter. That information is then processed at the remote station and displayed on a screen.

Our system is named D2H, which stands for "Dr House at Home". This system has a base station and a portable EKG meter, and we use infrared (IR) to transmit data between them. The EKG meter is designed to be easily-carried by patients. Therefore, it only consists of a small FPGA board and the necessary circuits for EKG sensing and IR transmission. Moreover, the EKG meter can be powered by a 5V battery in consideration of mobility and patient safety.

The base station gathers data from the IR receiver circuit and stores them into its own FPGA board. It then displays the EKG waveform on the monitor, and processes the data to retrieve patient information, like the heartbeat rate. Sound effects, such as the sound of a beating heart and a patient abnormality alarm, are also included to enhance the monitoring efficiency. As a result, we have built a system that monitors patient's EKG data remotely.

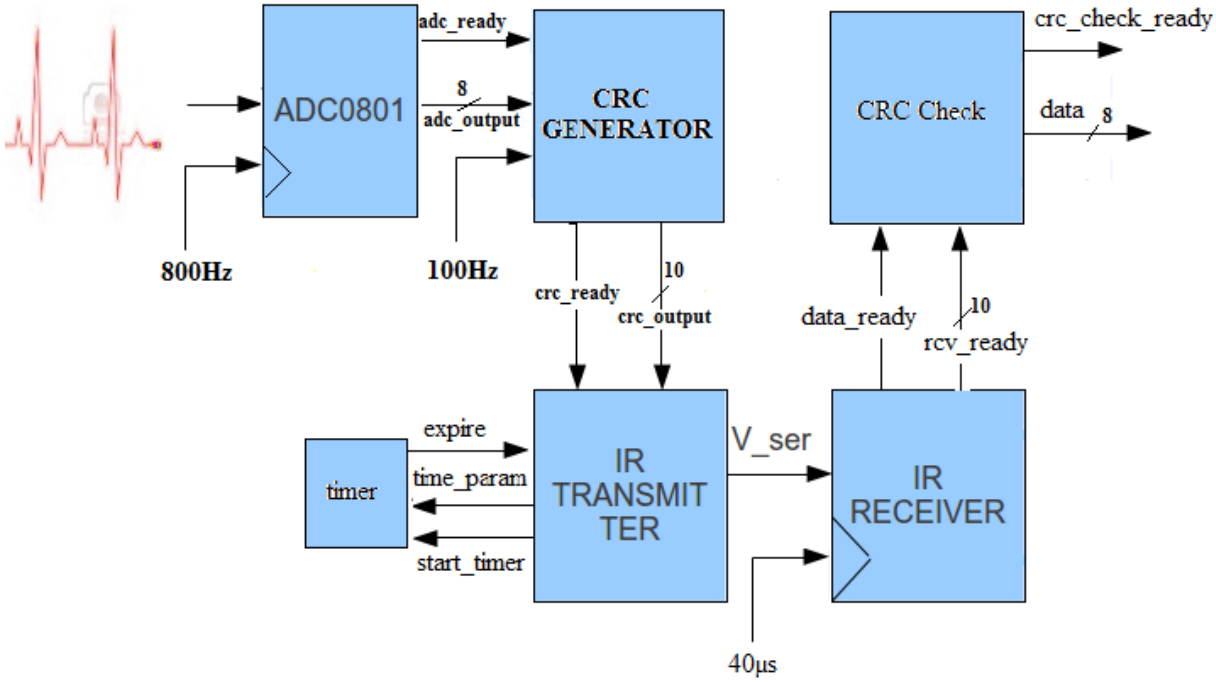
Modules Description

The project was divided into 3 main blocks: data acquisition and transmission, signal processing, display and sound. Below is a block diagram of our system. The description of individual blocks follows.



1- DATA ACQUISITION AND TRANSMISSION (Lyne Tchapmi)

The first part of the project consists in acquiring heartbeat data and sending it to a remote station using infrared transmission. For the purpose of our project we use a waveform generator to generate a heartbeat waveform. The input data is sampled at 800Hz using an 8-bit analog to digital converter, the ADC0801. Cyclic Redundancy Check (CRC) is used to modify the data for error detection. The modified data is then sent through the IR channel as a serial output, received and reconstructed on the receiver FPGA where it is used for further heartbeat analysis. A block diagram of the data acquisition system follows:

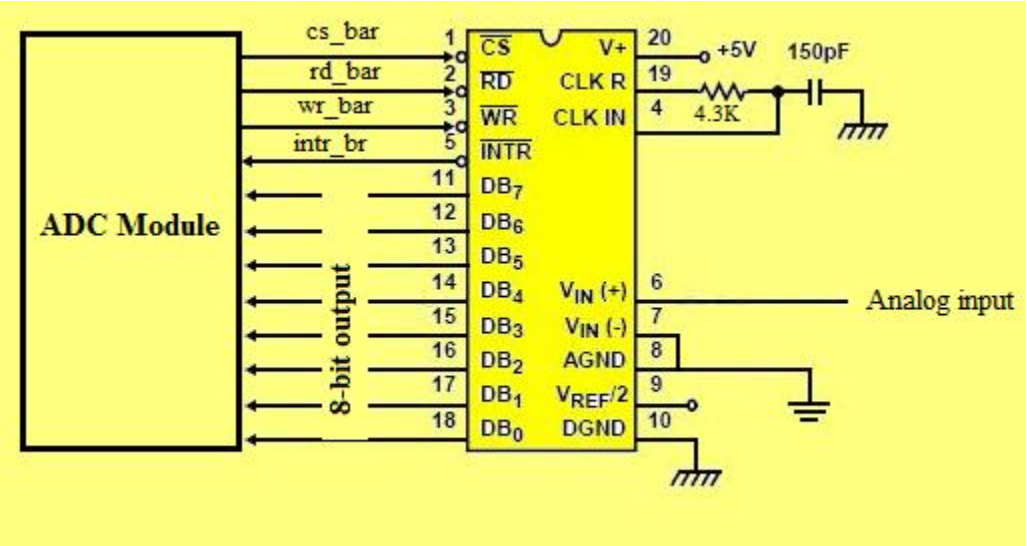


1.1- ADC MODULE

- **Hardware setup**

The analog heartbeat data from the waveform generator is sampled at 800Hz with the ADC0801. We built the following circuit to communicate with the ADC0801.

The choice of a 150pF capacitor and a 4.3K resistor for our circuit gives us an internal clock -- CLK IN-- of about 640 KHz for the ADC.



- **Communication with ADC0801**

The ADC module produces "*read*" and "*write*" low-active pulses for the ADC (rd_bar and wr_bar on the diagram). Each "*write-read-wait*" cycle is triggered by the rising edge of the 800Hz clock. A "*write*" pulse is sent at the beginning of the cycle. After each "*write*" pulse, we wait for "*interrupt*" to be asserted. We then wait an additional 8 ADC internal clock periods after "*interrupt*", before sending a "*read*" pulse. The length of the read pulse was chosen so that the ADC output data would be ready and stable when "*read*" is deasserted. As a consequence, the falling edge of "*read*" is used as an output to signal when data is ready to be sampled. This "*ready*" pulse is crucial in sampling given that the output of the ADC is not always stable and thus needs to be sampled at the right times. The length of the "*read*" / "*write*" pulses and the duration of the wait after the "*interrupt*" signal are monitored using a counter.

- **Testing and issues**

Testing of the ADC module was mostly done using the logic analyzer. The first check point was to verify that the "*read*", "*write*" and "*interrupt*" pulses had the expected duration and/or happened when expected. The second checkpoint was to verify that the 8-bit output of the ADC was correct. This 2nd test was a bit more complex since we needed to sample the output of the ADC only when data was stable. We used a nice feature of the logic analyzer—the external clock-- to sample all waveforms using the "*ready*" signal of our ADC as the logic analyzer sampling clock. We displayed the magnitude of the ADC output which enabled us to visualize the actual waveform that was being sampled.

The ADC chip had to be replaced a couple times because it sometimes stopped functioning properly after a prolonged use.

1.2- CRC GENERATOR

- **CRC length and sampling rate**

The CRC generator samples the filtered ADC data at 100Hz and computes a 2-bit CRC for each 8-bit input data. We used a 2-bit CRC since the IR transmission channel has a limit transmission rate of 1Kbps. Sending a 2-bit CRC with an 8-bit data give us 10bits to send after each clock pulse. In the worst case scenario, our transmitter takes 1ms to send a data bit, that is, 10ms to send 10bits –or 1output of the CRC generator. This corresponds to the 100Hz sampling clock chosen for the CRC generator.

- **CRC computation**

We use 101 (3'b101) as our CRC generator. The CRC computation starts by appending 2 zeros to the 8-bit input data from the ADC. The generator and the modified input data are then aligned with their most significant bits (MSB) overlapping. A sum of this alignment is then made by adding up overlapping bits (ignoring carry) and reporting bits with no overlap correspondents to the resulting 10-bit sum at their corresponding index —effectively treating them as if they

overlap with a “0”. We then slide the generator along the resulting sum until the generator’s most significant bit is aligned with a “1” of the 10-bit data. At this point, we do another sum as above. The slide and sum cycle is repeated until the least significant bit (LSB) of the generator and the 10-bit data are aligned. Then, the last slide and sum is performed. The last 2 bits of the final result constitute the CRC of the 8-bit input data and the first 8-bit of the data should all be 0s. A “*ready*” signal is asserted at the end of each CRC computation and the output is a 10-bit data bus where the 8 most significant bits are the input data and the remaining 2 bits represent the CRC.

- **Extension: Patient ID**

As an extension we planned to simulate displaying data from multiple patients. To do so we added some inputs in the CRC generator module to allow us to change the patient’s ID. The patient ID is set using an 8-bit value switch, and the ID is changed by pressing a button. Whenever the “change_id” input is asserted, the current “patient_id” is stored and sent at the next pulse of the 100Hz clock instead of sending heartbeat data. A “*crc_gen_is_id*” pulse is sent with the ready pulse to tell the transmitter that the current data is an ID.

- **Testing and issues**

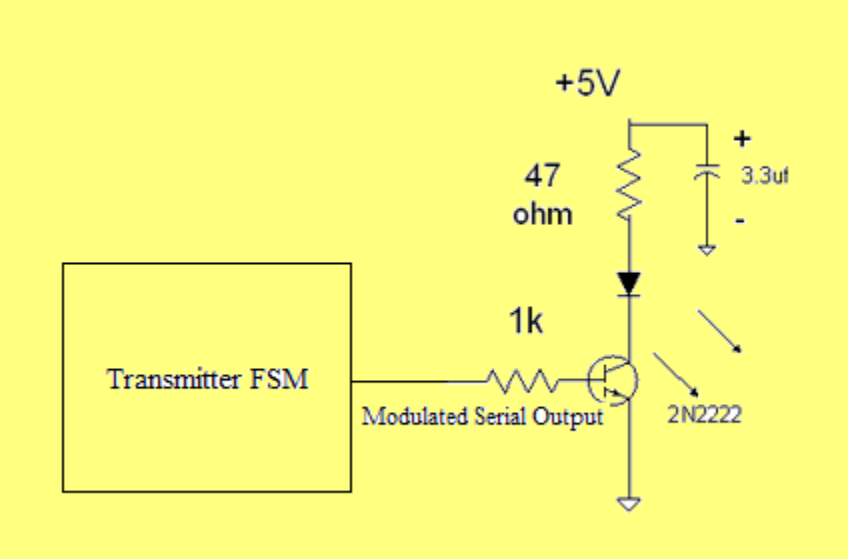
Testing the output of the CRC generator module involved using a test-bench on the ModelSim simulator. We also displayed that output on the logic analyzer as a waveform, and we were able to detect a bug in the pre-written fir filter module. It created glitches when given a sine waveform as input. The heartbeat waveform and other classic waveforms did not seem negatively affected by the filter though, so we kept it for this stage of the project

1.3- IR TRANSMITTER

- **Hardware setup**

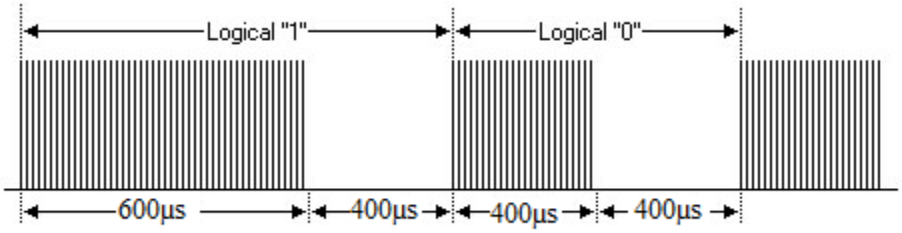
The IR transmitter takes in the 10-bit parallel input of the CRC generator and sends a serial representation of that data to an IR transmission circuit. The serial output of the transmitter

is then modulated by a 40 KHz carrier before being sent through the IR channel. We used the following IR transmitter circuit:



- **Serial encoding of data**

The transmitter state machine waits for the "*ready*" signal of the CRC generator to start transmission of each 10-bit data. Transmission starts with a ready pulse, which lasts 1.2ms for transmission of regular heartbeat data and 2.4ms for transmission of a patient's id. Data is then transmitted from least significant bit to most significant bit. A "1" being represented as a high of 600µs followed by a low of 400µs (a transition), and a "0" being represented by a high of 400µs followed by a low of 400µs as shown below:



The serial output is determined by a combinational block and depends on the current state. States "*START*", "*ONE*", and "*ZERO*" all have a serial output of 1, whereas states "*IDLE*", and

"*TRANSITION*" have a serial output of 0. A timer determines how long we stay in each state – exception made for the "*IDLE*" state- effectively determining the length of a "*start*", "*zero*", or "*one*" pulse.

- **Transmission rate**

Ignoring the "*start*" pulse, sending a "1" takes more time -about 1ms. Given that the CRC generator module samples at 100 Hz, the transmitter receives "*ready*" pulses every 10ms giving it enough time to send 10bits of 1ms each. Effectively this corresponds to a transmission rate of 1Kbps, the maximum transmission rate of the IR channel. In practice, our transmission rate is slightly less than 1Kbps since we don't send "1s" continuously.

- **Timer**

The count for each transmission is performed by a timer that uses a clock of 40µs to count. When entering a "*ZERO*", "*ONE*", "*START*" or "*TRANSITION*" state, the transmitter state machine sends out a "*start_timer*" pulse to start the timer. It also sends out a time parameter that helps the timer determine the length of the count that is being started. The timer is a basic counter that takes in the start signal and the time parameter value. Depending on the input time parameter it chooses the length of the count from a pre-determined parameters list. An "*expire*" pulse is sent once each count is over.

- **Testing and issues**

Throughout our transmitter state machine, we used several pulses-- such as the "*expire*" and "*ready*" pulses-- to trigger different events. When trying to change the length of a start pulse for a patient's ID, we encountered timing issues due to that practice; the time parameter for the patient ID was not assigned at the right start_timer pulse. After switching to using our 27 MHz clock as the master clock, we encountered other timing issues, mainly a delay of one state transition in setting the time parameter variable for the timer module. We were not able to solve

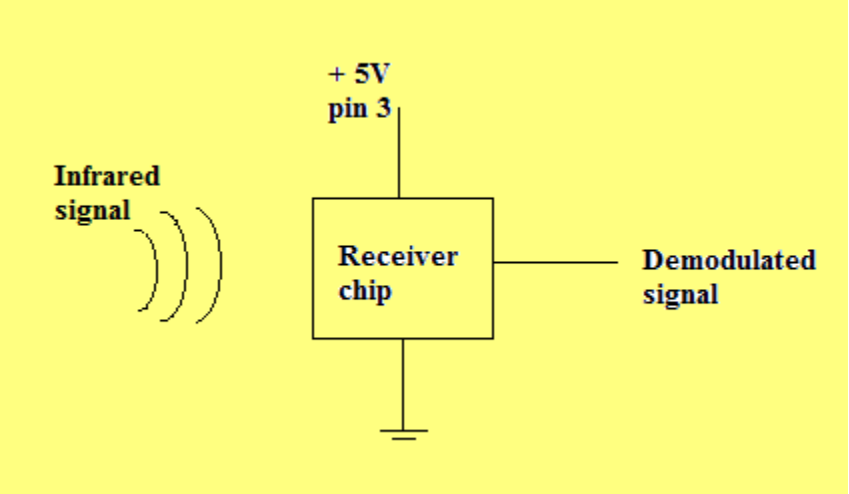
the issue due to time constraints and therefore did not include the patient ID feature in our project. (We also went back to using different pulses for triggering events).

Testing for the IR transmitter was first performed with the simulator then the logic analyzer to check for the length and time of each pulse as well as the appropriate sequential transmission of bits. We later connected the transmitter to the receiver for further testing.

1.4- IR RECEIVER

- **Hardware setup**

The IR receiver chip demodulates the modulated serial input from the transmitter and sends it over to the 2nd FPGA for processing by the IR receiver module. We built the following receiver circuit on the 2nd FPGA:



The pin out for the 2N2222 BJT is shown below. Note that the collector (pin 3) is *connected electrically to the case!*

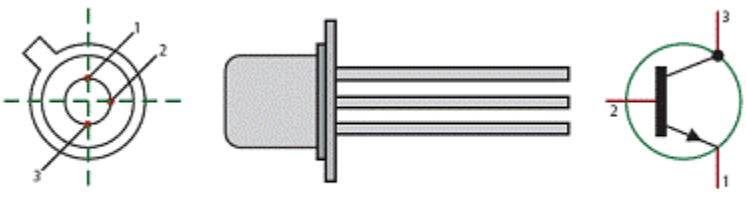


Figure by MIT OpenCourseWare.

- **Signal reconstruction**

The receiver module waits for a "*start*" signal to determine the start of each 10-bit data sent over the IR channel. After a start pulse, the following 10 pulses are decoded as either 1s or 0s. The "*start*", "*one*" and "*zero*" pulses are detected by counting the length of the high pulses and comparing it to the expected length of a "*start*", "*one*" and "*zero*" pulse. As bits are processed, they are shoved into a 10-bit shift register so that the final 10-bit result has the "8-bit data, 2-bit CRC" original format. Once the 10-bit data is obtained, a ready pulse is sent for the CRC check module as well as the recovered 10-bit data.

- **Testing and issues**

With IR transmission, it is possible that several bits get lost during transmission. One way to attenuate the effect of lost data was to throw out any non valid start pulse. On the other hand lost "1s" and "0s" can also affect the transmitted data. So we found a transmission range for which data was minimally lost. For the lab environment, this range was about 1m between the transmitter and the receiver.

After testing the receiver using the simulator we combined it with the IR transmitter on the same FPGA for further testing on the logic analyzer. We displayed several waveforms on the logic analyzer representing data sent and data received. Later on, the receiver was ported to a different FPGA for further testing.

1.5- CRC CHECK

- **CRC Computation**

The CRC check module is used for error detection. It takes in the "ready" signal from the IR receiver and computes the CRC for the 10-bit input data using the same generator as the CRC generator module 3'b101 but without appending additional 0s to the input, as is the case for the first CRC module. If the computed CRC is not 0, there is an error in the transmitted data and the data is discarded. This is done through a ready pulse, which goes high if the computed is 0, meaning no error was detected in the received data. Simultaneously, the output of the CRC

module is changed to the currently received data. If an error is detected, the ready signal is not asserted and the output data keeps its previous valid output.

- **Testing and issues**

We first tested the CRC check module with the simulator. Since CRC can only perform error detection and no correction, we had to set up our environment adequately to get a decent output signal. After setting the transmitter and receiver's distance and angles, we tested the CRC check module with the logic analyzer to check that the "ready" pulse was sent when CRC computation was done and the CRC was 0. We also drew out the waveform output coming from the receiver.

The waveform obtained was a good representation of our input signal but wasn't very smooth. We put it through our FIR filter but the filter's performance was very poor for this stage and we had to use the raw output from the CRC check module.

The data acquisition system has a very good overall performance and its main liability is the environment in which it is used. Given the proper environment settings- receiver transmission alignment and distance- the system is able to sample and transmit a signal reliable enough to be processed and used by the following stages of the D2H.

2- SIGNAL PROCESSING BLOCK (Wenting)

The signal processing block's functions are quite clear: to use the samples from the IR receiver and perform the necessary calculations for display. The most important analysis the signal processing block performs is peak detection. This in turn will provide the necessary information to other modules in this block to perform calculations, such as the heartbeat rate.

There are several challenges faced by the signal processing block, most of which need to be solved by peak detection techniques. Real EKG signals often have an offset that is constantly oscillating. Added to the baseline wander problems is noisy data from infrared transmission. Thus, a robust peak detection module would need to take care of these problems. After a peak is

appropriately detected, another challenge is to display the patient's heartbeat in real time. Calculation needs to be done in real time and be displayed on the screen.

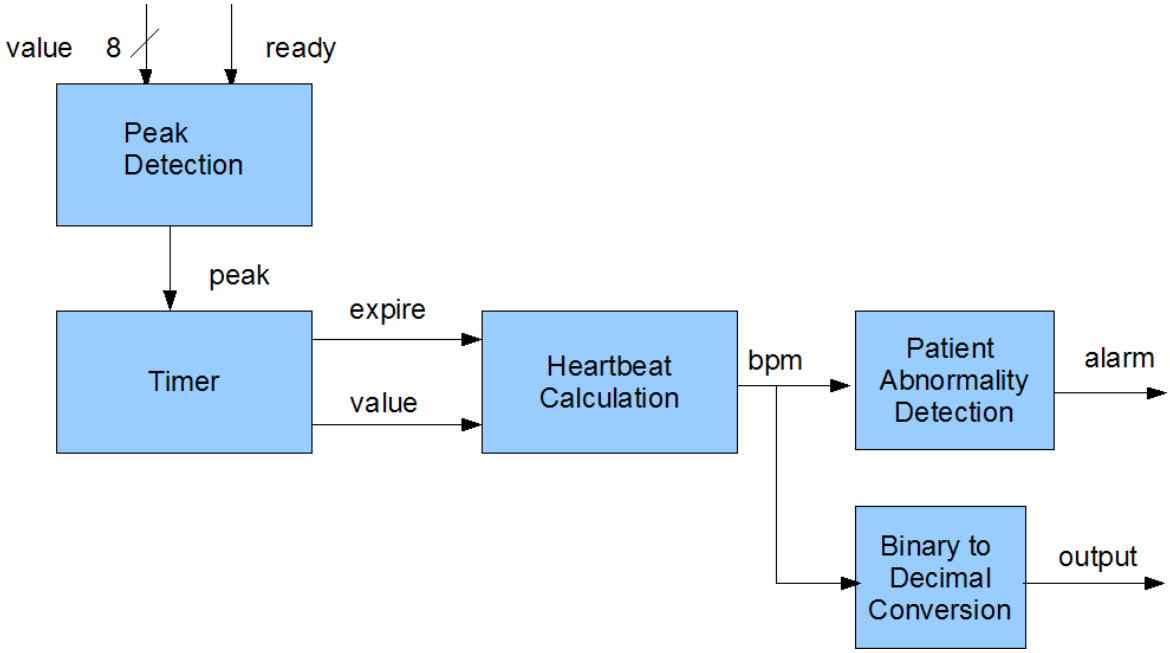


Figure 3.1 Block diagram for signal processing

2.1- PEAK DETECTION

This module's inputs are a one-bit ready signal and an 8-bit value. The module is implemented as a finite state machine and is responsible for detecting the main QRS peak of the heartbeat waveform.

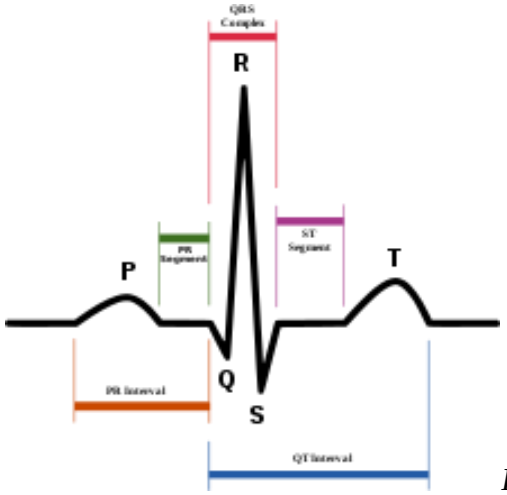


Figure 3.2 EKG waveform

The graph above shows the waveform classification of an EKG wave. A normal heartbeat waveform consists of a continuous sequence of PQRST waves. The P wave represents a normal heart's atrial depolarization process. The QRS wave represents the rapid depolarization of the right and the left ventricles. The T wave is generated during the heart ventricles' repolarization or recovery.

I wrote several versions of the peak detection module that used various detection methods, including both adaptive slope threshold and amplitude threshold algorithms. However, as I later found out when testing with real data, the IR receiver's samples are relatively scarce compared to the number of samples the peaks are represented as. Since the analog-to-digital-converter (ADC) samples the EKG waveform at 800 hz, but the IR channel is limited to 1000 bits/sec. Since each sample taken from the ADC contains eight bits of information, this means that the real time sampling rate is limited to about 100 hz. Thus, the quick QRS peak waveform often merely contains 3 to 5 samples. With such few samples, slope detection methods that are more complex and require more computations would actually have an adverse effect on peak detection. Real data testing showed a very high false negative rate. Thus, I decided to implement an amplitude detection algorithm.

The peak detection module is implemented as a four-state finite state machine (FSM), although the number of equivalent states is actually five. The following diagram illustrates the FSM's operation:

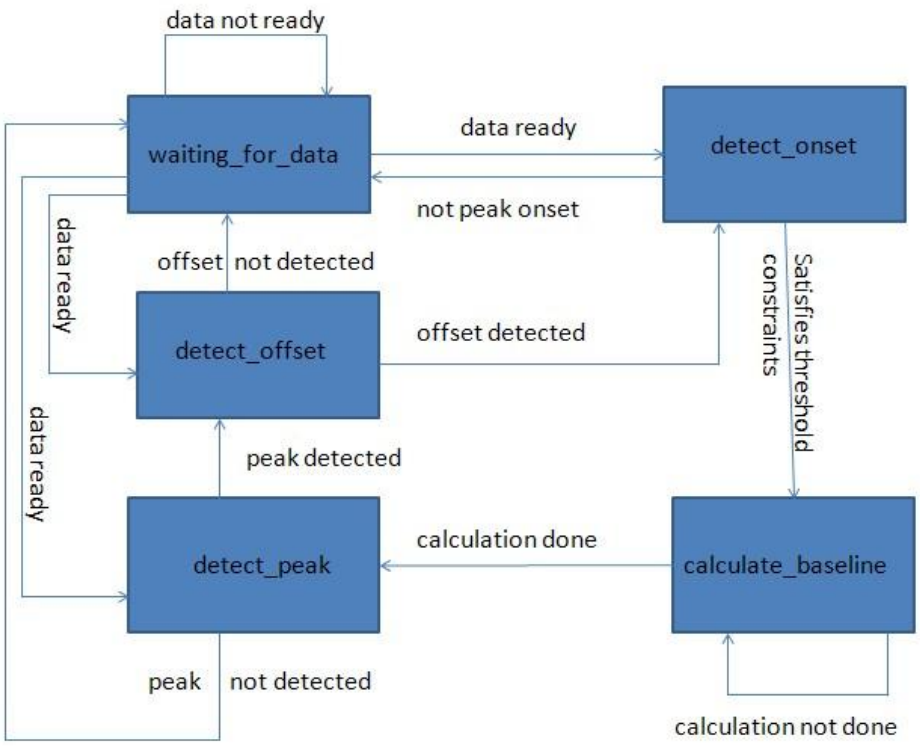


Figure 3.3 State machine diagram for peak detection

The starting state is *waiting_for_data*, which basically waits for a ready pulse and an 8-bit data from the appropriate module. When the data is ready, the peak detection module will store the data both in an internal 512 sample long buffer and send a write enable signal to the waveform memory module (to be described later in the display block). Thus, the peak detection module keeps track of both the waveform memory module's address (15-bit) and the internal circular buffer's address (9-bit). The two different address trackers are always increased at the same time to be synchronized with each other.

There are two more equivalent states inside the *waiting_for_data* state. When the portable EKG system is first starts, the peak detection module actually stays in *waiting_for_data* state for the first 511 samples. The samples are used to calculate an initial estimate for the amplitude threshold. This extra step, compared with the method of an estimation of what the threshold could be based on past experiments, will make signal processing a little more accurate. We cannot assume what the starting data will look like. The other equivalent state will be explained later in this description.

Once data is collected, the state machine will first enter *detect_onset* state. This state checks to see if two consecutive points are above the amplitude threshold. This prevents one glitchy sample from false triggering the state machine. If two consecutive points satisfy this constraint, the state is set to *calculate_baseline*.

As its name suggests, *calculate_baseline* performs the simple function of estimating the baseline of the EKG waveform. One of the big problems in analyzing EKG waveform is its baseline wander. There is often a changing DC offset to the signal that makes it difficult to perform amplitude detection. Thus, each time the offset is detected, the state machine will recalculate the baseline value. It sums up the collected data for the past 512 samples and averages them. However, this value is not quite the baseline because of the high voltage amplitude of the QRS peak. Thus, the average value is halved to get the baseline estimate.

Once the baseline calculation is finished, the FSM will transition to *detect_peak*. This state will detect the peak by comparing a particular sample with two samples before its index and two samples after its index. If x[n] is the sample we are analyzing, the state machine will compare x[n] with x[n+2] and x[n-2]. If x[n] is larger than both of these values, a peak is assumed to be detected. The FSM asserts the peak output signal to be high, and produces the appropriate peak address for the waveform module. This address will help the display block to highlight the peak on its waveform. The FSM will actually stay in the state machine for another clock cycle in order to calculate the new amplitude threshold. The calculation uses the peak point's amplitude and the baseline to estimate a threshold. The FSM then transitions to *detect_offset*. If a peak is not detected, the FSM will go back to *waiting_for_data* to wait for more samples.

The *detect_offset* state is quite similar to *detect_onset* state. It compares the two most current samples and triggers when the both samples are below the amplitude threshold. If it does not trigger, the state will simply go back to *waiting_for_data*. If the comparison does trigger, the FSM will go back to *detect_onset* state and also set a variable called *false_detection* to be zero. This variable is the second hidden state for the *waiting_for_data* state. When *detect_onset* fails, the state machine will look for more samples. However, it will stay in *waiting_for_data* for the next sixteen samples and not look for another offset. I added this state after some experimentation with IR transmitted data. As I mentioned before, the transmitted data often has a

high level of noise, and as such could trigger multiple times after the peak is detected. False triggering makes peak detection more robust.

2.2- TIMER BLOCK

The timer block consists of two modules, timer_counter and ten_hz. This block is responsible for calculating the period between the heartbeat peaks.

The ten_hz module uses the internal 65 mhz clock on the FPGA to output a high signal every 0.1 second. The module also takes in a reset signal that, when asserted high, will reset the module's internal counter to zero and synchronizes it with the module that is using the ten_hz signal.

The timer_counter module takes in a peak signal from the peak detection module and outputs a time value and an expire signal. This module will start counting time when peak signal from peak detection module is asserted. On the next asserted peak signal, timer_counter will output the time elapsed since the last peak signal, along with an expire signal to tell the next module the time is ready. It will also reset its internal time to zero.

2.3- HEARTBEAT CALCULATION

This module, as its name suggests, is responsible for calculating the patient's heart beat in beats per minute. The heartbeat calculation module takes in an expire signal and a time value from timer_counter. It is implemented as a simple finite state machine. The states consist of:

waiting: The waiting state waits for the new time_value to come from the timer_counter module. Once there is a new time_value, it updates the internal circular buffer with the new value and enters *summing* state.

summing: This state uses a counter to cycle through the internal circular buffer and calculates the sum of the past eight periods between the peaks. Once the sum is ready, state is changed to *dividing*.

dividing: In this state, the FSM only needs to wait for a few clock cycles and then output the result of the division, taking the least significant nine bits.

Since heartbeat calculation module needs to convert period to frequency, it needs hardware that will be able to perform a calculation that is a little bit more intensive than usual. The formula used here is 4800 / (sum of all eight time values). This calculation will give the patient's heartbeat in beats per minute. The formula is derived as follows:

$$\text{average time value} = \text{sum of eight time values} / 8$$

$$\text{frequency of heartbeat} = 1 / (\text{average time value} / 10) = 10 / \text{average time value}$$

$$\text{beats per minute} = 60 * \text{frequency of heartbeat} = 4800 / \text{sum of eight time values}$$

2.4- PATIENT ABNORMALITY DETECTION

This module takes in the output of the heartbeat calculation module. Since the heartbeat calculation module already performs the necessary calculation to output the beats per minute information, the patient abnormality module can simply use a threshold to determine the state of the patient. The thresholds are set at 45 and 160. If the calculated beats per minute exceeds 160 or goes below 45, then the alarm output will be asserted high.

2.5- BINARY TO BINARY-CODED-DECIMAL CONVERSION MODULE

Since the beats-per-minute output from the heartbeat calculation module is in binary, it is necessary to convert the binary number into a decimal number to be displayed by the video displayer module on the screen. This module uses a fast algorithm that can convert a 9-bit binary numbers into a 12-bit binary-coded-decimal numbers in 8 clock cycles. The binary-coded-decimal consists of three groups of 4-bit numbers that will represent the three digits of a decimal. The algorithm used is named the "double dabble" algorithm, which uses the following steps:

1. Initialize a 12-bit number with all zeros

2. Divide the 12-bit number into three 4-bit groups. Each group represents one digit of the resulting decimal number
3. Append the 9-bit number to the 12 bit number
4. Shift the binary number to the left by one bit
5. If any of the three groups exceeds 5, add 3 to the group
6. Go to Step 4 and repeat until all of the 9 bits have been shifted left

2.6- TESTING AND DEBUGGING

My testing strategy consisted of both simulation and real data testing. For some modules it was sufficient to test using mainly simulation, such as the binary to binary-coded-decimal conversion module. Other modules required a combination of both.

The heartbeat calculation could not be completely simulated because of the divider module. Therefore I proceeded to test the module by using writing a simple pulse generator to simulate the peak detection. The result of the heartbeat calculation is then displayed on the FPGA hex display. I also displayed several other useful variables, such as the period calculated and the calculated sum.

The peak detection was the most difficult module to test. Since the ADC was not set up until later, I proceeded to use ModelSim to test the peak_detection module. I used a helpful Matlab command to generate EKG waveform samples. Then I used these samples to test the peak detection module in ModelSim. The most important part was looking for state transitions. However, some of my more complex peak detection modules could not be completely tested because of the attempt P wave backtracking, which would take up several hundred clock cycles. Later, after the ADC was set up, I sampled directly from the ADC and tested peak detection by displaying the heartbeat calculation (after verifying that the heartbeat calculation block was correctly implemented). After the video part was integrated, I could then display the waveform form the ADC and see if the right peaks were highlighted.

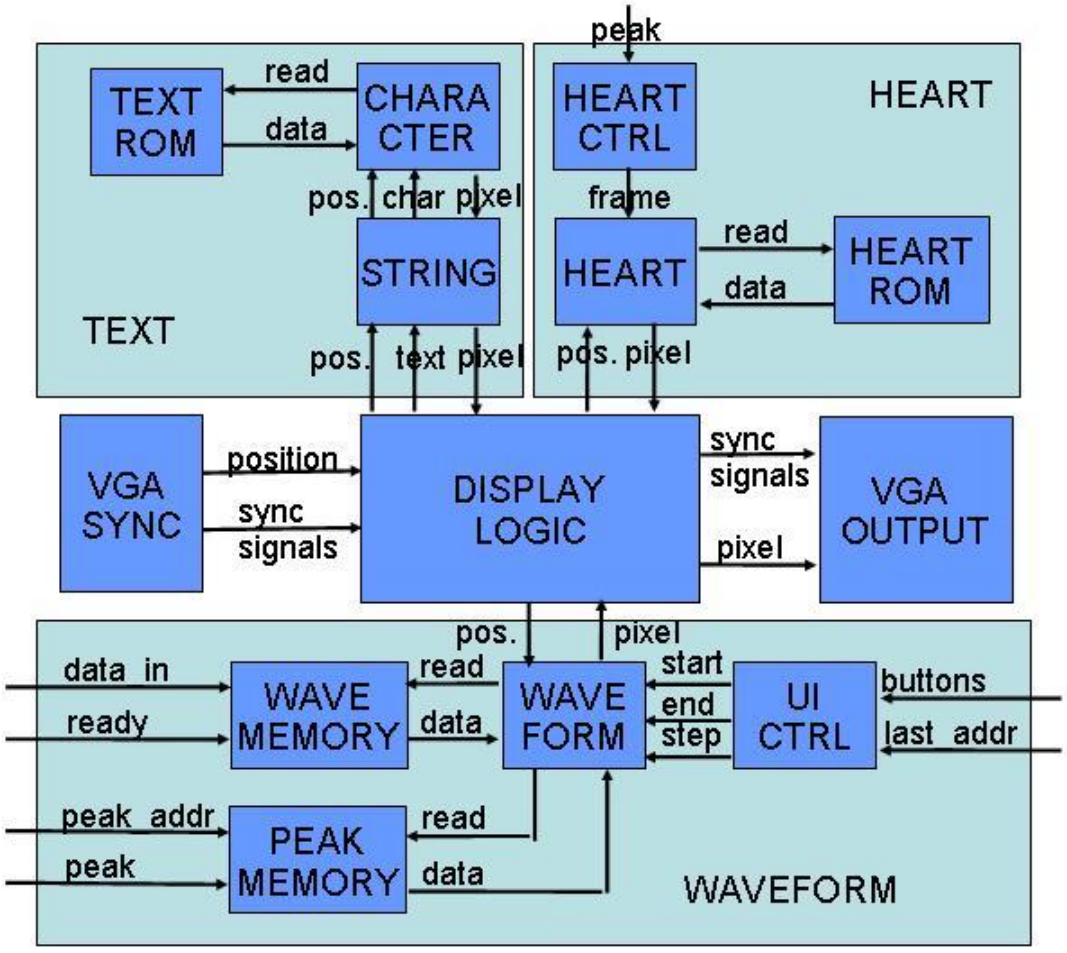
3- DISPLAY BLOCK (Szu-Po)

3.1. OVERVIEW

3.1.1. Functionalities:

The display block takes information of the patient from the signal processing block and displays it on VGA (Video Graphic Array) screen. Three types of information are shown: EKG waveform with detected peaks, heartbeat rate, and heartbeat animation synchronized to the patient's heart beat. VGA signals for each of them are generated by separated modules, and display logic selects the signal to send to the monitor.

The users can use a button-based user interface to select which part of the stored waveform they want to see, and also change the horizontal time scaling of the waveform. This is accomplished by using user interface module to generate the control signals for the waveform display.



3.1.2. Connection to Other Blocks:

The data displayed all come from the signal processing block. For the waveform, the signal processing block writes the samples to a shared BRAM (Block Random Access Memory), and notifies the display block about the valid addresses of the samples. The display block then reads the data from the BRAM. For the peaks detected by the signal processing block, the address corresponding to the peaks is sent. The display block stores the information of the peaks and shows them on the waveform. The heartbeat animation is synchronized to these peak signals. The heart beat rate is passed in the BCD (Binary-Coded Decimal) form and can be directly shown on the screen.

3.1.3. Hardware Constraints

Timing:

For the VGA display of resolution 1024*768, a clock period of 15 ns must be used. In order to meet this timing constraint, we use pipelines in the logic to split the calculation time into different clock periods. In the final design, there is a 5-stage pipeline used for the display block, i.e. the output VGA signals come out at five clock periods after the pixel count is generated. In the future, we may consider incorporating ZBT memory as a buffer of the display instead of calculating it in the real time. This may separate the calculation from the display, and relieve the timing constraints of producing the pixels.

Memory:

We use the built-in SRAM (Static Random Access Memory) blocks of the labkit for implementing all memory used by the display modules. The amount of memory is limited by the size of SRAM, which is 2304 kilobit if the data width is a power of two. Therefore, we need to tightly control the memory usage in order to fit in the available resources.

The memory budget is allocated as follows: 256 kb for the waveform BRAM, 32 kb for the peak BRAM, 576 kb for the text ROM, 1280 kb for the heart image ROM, and 32 kb for the sound ROM. It is 2176 kb of memory used in total.

3.1.4. Testing:

Since the display block corresponds to a large number of signals and extensive usage of automatically generated memory blocks, it is hard to test it through simulation. Instead, we test it by manually providing data and check if the result is appropriately shown on screen.

For the waveform, we generate artificial waveform data by sampling the switch input at 100 Hz on the labkit. On the other hand, the text display is tested by showing the character having its ASCII value corresponding to the switch. Moreover, the heartbeat animation is tested by manually press a button to provide the peaks pulses, and see if the beating heart can be synchronized to the rhythm of the peaks. As long as all these data can be displayed properly on the screen, we can make sure that the modules are working correctly.

3.2. WAVEFORM DISPLAY

3.2.1. Waveform Memory Module:

The EKG waveform is stored in a BRAM of depth 32768 and width 8 bits. This corresponds to about 3.5 minutes of waveform data at the sampling rate of 100Hz. The samples are stored in the sequential order of address. When the memory is full, the address goes back to the beginning and overwrites the out-dated samples. Therefore, waveform data in the last 3.5 minutes are always available.

In order to ensure that the waveform data are consistent on the screen, we need to protect the memory from being written when the waveform display module is reading it. To accomplish the protection, we use a single-port BRAM which prevents simultaneous access to the memory. When writing and reading operation occurs at the same time, the reading operation is given the priority to access the data. On the other hand, the writing operation is stored in a queue, and executed only on later spare clock cycles. As a result, the data will be constant as long as the reading operations happen on consecutive clock cycles.

Since the sample rate (100Hz) is slow comparing to the 65MHz clock, no more than one writing operation will happen during a consecutive reading of 512 data points. Hence, one is sufficient for the size of the writing operation queue. Size 4 is used in the actual implementation of this module.

3.2.2. Peak Memory Module:

To display the detected peaks on the screen, we need to store the positions of the peaks in the memory. We store the peaks in the same type of memory module as the waveform memory, where we store the 1-bit peak information in a BRAM to indicate whether each sample corresponding to the address is a peak or not.

At first glance, it seems reasonable to store it as a logic-high in the address of each peak. However, when we try to change the time scale of the waveform, we will downsample the waveform and no longer read consecutive data points. As a result, we may miss the peak if we do not read on the exact position of it. In order to solve this issue, we store the peak information as a stream of 0s and 1s, and switch every time there is a peak on the address. In this way, if consecutive samples have the different value stored, we could know that there is a peak between them and display it on the screen.

In order to generate the peak-indication bit stream and store it in the memory, we keep track of the address of the last data point, last address written of peak information, and the current peak-indication bit. Each time a new peak is found, we change the peak-indication bit, and write it to the memory starting from the peak position, and continue writing until we reaches the last data point. Since the clock rate is much faster than the sampling rate, we can write to the last data point before the next peak arrives.

Nevertheless, another issue appears since now we are writing in consecutive clock cycles, which will fill up the writing operation queue easily when overlapped with reading operations. The solution of this issue is easy: since the pace of writing the peak-indication stream can be controlled by this module, we can let this module send out operation only when the peak memory is not being read.

3.2.3. Waveform Display Module:

The waveform display module displays the EKG samples within a specified address range. It takes in the input address range and the address spacing between samples (both generated by the user interface control). The EKG samples are read by sending operations on the waveform memory and peak memory. After the EKG samples are read and stored in the

registers, the waveform is displayed on the screen by comparing the samples with the pixel position.

The reading operations begin after the previous frame is shown on the screen. The address starts at the first address in the specified range. In each subsequent clock cycle, the address is incremented by the input step size until the end of the address range is reached. The result of the reading operation is stored after waiting for two clock cycles (one for buffering the reading operation, the other for reading the BRAM). For the peak information, the address for reading is the same as the sample, and by using XOR operation on consecutive samples we can regenerate the peaks and store it in the registers.

In the region of the waveform, the module checks if the vertical position of the pixel matches the sample value stored in the corresponding register. Since the sample value is discrete, we need not only showing the pixel corresponding to the sample, but also connect consecutive samples, in order to provide a continuous waveform. The segment between adjacent vertical positions is split into two pieces with equal length and displayed in the adjacent columns. Therefore, the combinational logic for displaying the waveform also checks if the pixel belongs to the segment in both sides. If either of the conditions holds, the pixel is displayed as part of the waveform.

3.2.4. User Interface Control Module:

This module allows user to control the region of the EKG waveform being displayed. It takes inputs from the buttons on FPGA board, and generates the control signals (starting and ending address, the spacing between samples) for the waveform display module.

Two display modes are presented: the slide mode, where the new data are shown instantaneously, making the waveform slide leftward; and the refresh mode, which start to display the waveform from left side at first, and clean up the waveform each time the right side is reached. The user can press a button to switch between these two modes. The user can also freeze the current waveform, and retrieve the past waveform data by the left and right buttons. Moreover, user can use the up and down buttons to control the horizontal time scale of the waveform. In this way, user can choose the desired precision of the waveform.

For the slide mode, the module checks if the current ending address is the same as the address of the last sample. Otherwise, the module changes the starting and ending address by the

same amount (“sliding” the window for display). Every time the user presses the time scale, the spacing between addresses of adjacent samples is changed. The ending position is also automatically adjusted to preserve the same number of samples. If the spacing is not one, the increment of the addresses must be in multiples of the spacing to prevent inconsistent aliasing in the downsampling process.

On the other hand, in the refresh mode, the module changes only the ending position when new sample arrived. Once the number of samples in the current interval becomes equal to the maximum capacity of the waveform window, both the starting and ending address are reset into the address of last sample (“refreshing” the window).

In the case that the waveform is freeze, the starting and ending position do not change automatically. When user presses the left or right button, the interval is shifted accordingly as long as it still stays in the range of valid memory addresses. Once the user unfreezes the waveform, it goes back to its original mode and starts to display the new data again.

If the waveform is freeze after a while, the waveform memory may run out and override the interval of the freeze waveform we are displaying. This issue is handled by not updating the values in registers of the waveform display module when the data is overwritten. When user tries to move the waveform in this situation, it jumps to the first samples stored in the memory, since they is the nearest available data points.

3.3. TEXT

3.3.1. Text ROM:

The text ROM stores images of all 95 visible characters, each represented by a bit matrix of size 64*96. This memory is shared among multiple modules displaying texts. The coordination of the memory operations is done by allowing a module to access this ROM when the current pixel is in the region of the corresponding character. As long as the characters shown are not overlapping, it is ensured that only one read to the memory take place in each clock cycle.

Multiplying the number and dimensions of the characters, we can find that the size required for this memory is 570 kilobit. However, the size of memory generated is always a power of two. As a result, it uses 1024 kb if we directly implement it as a block ROM, wasting a

lot of memory. In the actual implementation, this ROM is split into two pieces: one of size 512kb and the other of size 64kb, saving the memory usage by 43 percent.

3.3.2. Character Display Module

In the character display module, the pixel output is generated by reading the shared text ROM in the correct position, and checking if the current pixel is part of the character. The user can specify the position, size, and color of the character as parameters of the module. The text being displayed is selected by an input bus of ASCII value. Therefore, the module offers the flexibility to design the layout of the screen, and also the possibility to change the text in runtime.

The module first generates the address for reading the ROM. The data are ready at the output of ROM after two clock cycles. Since the output of ROM is not capable for driving all the character modules directly, an additional stage of register is used for buffering the data. As a result, this module reads in the buffered data at three cycles after the reading operation, and uses that information to decide whether the text color or the background color should be outputted for that pixel.

3.3.3. String Display Module

A string is a list of characters; therefore, we can display a string by creating a list of character modules. The string display module creates these modules automatically, and selects the appropriate output signals from the list of modules. In this way, a string can be displayed by specifying its length and the character array, instead of generating a list of character modules manually.

3.4. IMAGE

3.4.1. Image ROM

The beating heart animation consists of 16 frames of 16-color images of size 128*160. This module, with the similar structure with the text ROM, is used for storing the pixels of the images. Under the same consideration of memory constraint, the ROM is also split to two pieces, with sizes 1024 kb and 256 kb.

The images come from *Animated Heart*, Knowledge Weavers Project, Spencer S. Eccles Health Sciences Library, University of Utah (http://library.med.utah.edu/kw/pharm/).

3.4.2. Image Display Module

This module reads from the image ROM and displays the specified frame of image on the screen. The position of the current pixel and the image frame number are used to calculate the address for reading the ROM. In two clock periods, the ROM responds with a 4-bit number representing one of the 16 colors. The number is then passed to a color-mapping table to reconstruct the color of the pixel.

3.4.3. Heartbeat Synchronization Logic Module

This module is used for generating the frame number of the images to synchronize the animation with the heartbeats. The peak signals detected in the signal processing block is used for this synchronization. For each heartbeat cycle, the output frame number is incremented from 0 to 15 and looped back to 0, corresponding to a cycle of the images.

The synchronization logic maintains several variables: PERIOD, the expected period of the heartbeat cycle (in terms of the number of the VGA refresh); FRAME, the frame number, stored as a binary real number with a fixed number of digits after the decimal mark; STEP, the amount of frame needs to be incremented in each VGA refresh; and GOAL, the frame of finishing the next heartbeat cycle. The control logic is as following.

Each time VGA refreshes:

if FRAME<GOAL

FRAME = FRAME + STEP

When VGA refreshes, the frame number is incremented by the amount of step. The integer component of the frame output can then be sent to output for displaying the heart. The frame number is not incremented when it already exceeded the frame number of the next peak. This ensures that only one loop of animation is displayed in each heartbeat cycle.

Each time a peak is detected:

PERIOD = (PERIOD + current_period)/2

GOAL = GOAL + 16

next_peak = FRAME + PERIOD * STEP

STEP = STEP – (next_peak - GOAL)/256

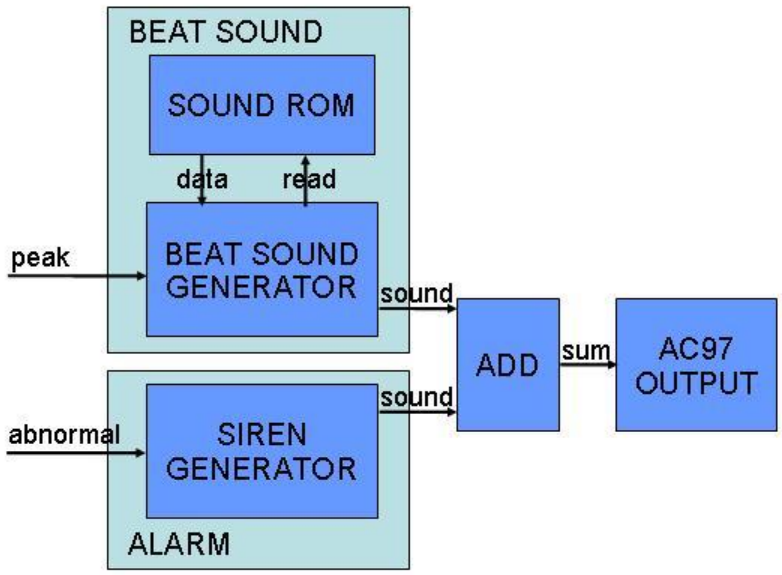
The expected period is calculated by averaging the new heartbeat period with the previous value of PERIOD. For each detected peak, the GOAL is incremented by 16, corresponding to one cycle of heartbeat images. The frame number of the next peak can then be predicted by current frame number, expected period, and the step of frame number. The difference between GOAL (the desired frame number of the next peak) and next_peak (the expected frame number) is then used to adjust the step of frame number. By this negative feedback control, the value of STEP will be stable and synchronized to the detected peaks.

4- SOUND GENERATION BLOCK (Szu-Po)

4.1. OVERVIEW:

4.1.1. Functionalities:

The sound generation block provides audio output for the ease of the monitoring of EKG data. Two types of sounds are generated: heart beating sound, produced at each peak of EKG waveform; and alarm, sent when the signal processing block detected abnormality of the patient. These sounds are generated by two separate modules, then added together and sent out to the loudspeaker through the AC97 chip.



4.1.2. Connection to Other Blocks:

This block depends only on the outputs of the signal processing block. Each time the signal processing block detects a peak, a pulse is sent to this block, and then the heartbeat sound generation module starts to play the heart beating sound one time. The patient abnormity value will be set high throughout the time when the abnormity is detected. The alarm generation module takes in the abnormity value and generates the siren whenever it is high.

4.2. HEARTBEAT SOUND GENERATION MODULE:

This module has a ROM storing the sound of one heartbeat cycle. When the peak pulse is asserted, this block reads through the ROM from the beginning and sends out the received value. The sound is stored as 4096 samples at a sampling rate of 8 kHz. Since the sampling rate of AC97 chip is 48 kHz, each data point is duplicated 6 times and passed through a lowpass filter.

4.3. ALARM GENERATION MODULE:

The alarm generation module produces siren sound with its frequency looping between 400 Hz and 700 Hz. This module keeps track of the current half period, gradually changing it between the 46320 and 80880 periods of 65MHz clock. A counter is used for calculating the elapsed periods, and the output value is inversed each time the current half period is reached. Consequently, the resulting waveform is a rectangular wave with altering period.

CONCLUSION (Lyne Tchapmi)

The D2H is a prototype of a portable EKG system. It operates by acquiring heartbeat data and transmitting it wirelessly to a remote station using infrared. At the remote station, the data is processed to detect peaks in the heartbeat, abnormalities in the heart rate, and to display the heartbeat waveform. Visual effects and sound help in monitoring the heartbeat.

The main issues encountered were hardware issues. Mainly, the limitations of infrared transmission—range and scope limitations— limit the environment in which our system is reliable. The limited data rate of the IR channel also put a limit on the amount of data that could be sent per time period. This issue, along with the lack of an appropriate filter to smooth the data, gave us a waveform that wasn't very smooth and made it hard for the peak detection module to detect all peaks. Despite this limitation, we had a sufficiently good performance to be able to display a good outline of an input waveform, detect the main peaks, generate heart beat sounds, and make the heart animation beat at the rate of the heart beat waveform.

Areas of improvement include finding a more reliable transmission medium/technology—RF seems like a very good candidate. We could also improve our peak detection to be able to detect the P, Q, R peaks in the heartbeat. Finally being able to simultaneously send and display data from different patients would be a great feature.