Massachusetts Institute of Technology

6.111 Final Report

Delta Sigma Heart Rate Monitor

Joe Griffin

Hugo Malpica

December 9, 2015

Contents

List of figures

Abstract

In the realm of medicine, people are trying to use non-invasive methods to measure vital signs. One method to do this is to use a pulse-oximetry probe to read a patient's heart-rate. For our final project we will develop a Delta Sigma Heart Rate Monitor to demonstrate a waveform on a monitor display, a heart rate detection system as well as warnings for heart rates that require attention.

1 Design Overview

Our goal for this final project is to implement a system that will take in a signal from a pulse oximetry probe and be able to display that signal on a monitor. The signal will be used to determine heart rate and let the patient see how fast their heart might be beating. If a patient's heart rate went above or below a set threshold a warning sign would advise them that something is wrong.

The scheme that we used is depicted in Figure 1. Our implementations has two major components. The first portion of our system generates bits of information which are then used by the second portion of our system to generate waveforms as well as to calculate critical information about the patient. Both of these portions are modular which means that they can be worked on separately, but in the final implementation they depend on one another.

As this project has two major sections, Joe and Hugo split the work in two. Joe worked on generating bits from the pulse oximetry probe using a Delta Sigma ADC. Hugo was responsible for the processing of those bits as well as displaying important information. The project was mostly completed on the Labkits provided by 6.111 Lab but a significant portion was also built on the Nexys4.

2 Block Diagram

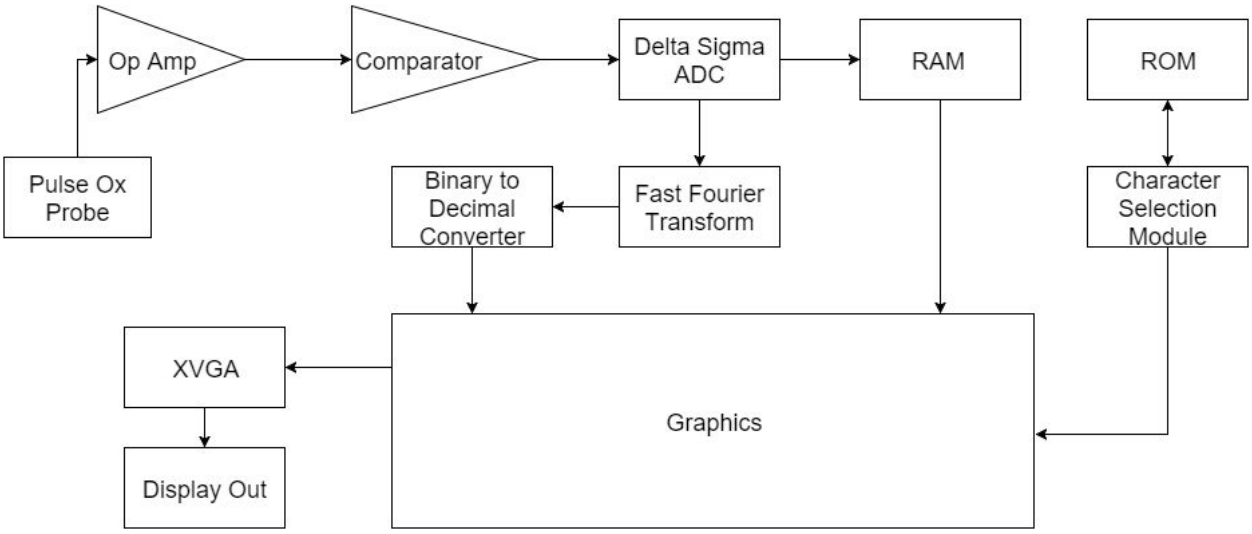

Figure 1: High level system diagram detailing major sub-blocks and interconnects between them.

3 Bit Generator by Joe Griffin

3.1 Analog Circuitry

As the pulse oximetry probe is an analog circuit, the first step in the signal path proposed above is to manage the analog signal in a way that allows the FPGA to meaningfully interact with the data. First, the signal was AC coupled to an Operational Amplifier with a feedback gain of 6. This amplified the signal enough to usefully allow a comparator to interpret the output waveform in a way that could be used by the FPGA, and it also removed the low-frequency drift in the output voltage of the circuit.

3.2 Analog to Digital Converter

The analog to digital converter (ADC) for this system was designed originally to use the Delta-Sigma algorithm. The basis for this algorithm is that, knowing where the zero-crossings of an input waveform are and what the bandwidth is that an input signal is limited to, one can reconstruct a signal by generating a band-limited interpolation of the signal with the zero-crossings and the correct sign. To do this, all that is required in the analog domain is a comparator and an anti-aliasing filter. In the analog circuitry above, an LM311 comparator and the RC circuit used to couple the input signal to the gain stage served these two functions.

To improve the precision of the digitized data, one would need to improve the precision of the zero-crossing timing. Naturally, this means increasing the sampling frequency of the data. Since the LM311 is a very fast comparator relative to the input signal, the initial intention of the project was to sample at a very high rate. This would provide a high-precision waveform that could be displayed on the screen.

3.3 Filtering

The Delta-Sigma algorithm relies on one key tradeoff: the quantization noise introduced by quantizing the input signal to a single bit with a comparator can be decreased by feeding the input signal through a low-pass filter with a cutoff frequency to match the bandwidth of the input signal. Since the intention was to obtain a high-precision signal with a high sample rate, the cutoff frequency of the filter relative to the sample frequency was very low. Specifically, the original filter was designed to attenuate frequencies above 50Hz in a signal sampled at 3.24MHz.

The Delta-Sigma ADC was never completed as it was originally designed. Any digital filter includes with it an inherent tradeoff between memory and frequency resolution. To successfully attenuate frequencies so close to DC without attenuating the entire signal, a finite impulse response (FIR) filter would require thousands of coefficients. Such a filter was not deemed feasible early on in the project. An infinite impulse response (IIR) filter, however, has a much longer impulse response and requires far fewer coefficients for the same impulse response effective length. A realization that escaped both the designers and course staff, however, was that the filter would need coefficient precision that far exceeded the original design architecture. The original specifications for the filter required coefficients with a minimum of 28 bits. With fixed-point arithmetic, this precision was of course infeasible, so the project design began to focus on the implementation of floating-point arithmetic filter architecture.

3.4 Floating Point Arithmetic

The amount of time required to create the supporting framework code for the rest of the Delta-Sigma ADC had pushed the coefficient precision realization back to the night before the first round of checkoffs. With so little time left, although the team was able to get a floating-point filter functional in the allotted time, they were unable to integrate the fully-featured ADC into the rest of the project.

Although the floating-point filter was never integrated into the main project, it still represents a valuable piece of code that is potentially very useful in later applications. The floating-point filter code implements a second-order IIR filter with 32-bit floating-point precision that can be repurposed for various applications. It requires two clocks: one to keep it synchronized with the time index of the incoming signal, and a second to control the internal state machine. The internal state machine must be operated at no less than 40 times the sample rate. Such a filter can be cascaded with other instances and parameterized to implement an

extremely high-resolution filter. The code takes advantage of two instantiations of an IP core supplied by Xilinx to perform multiplications and additions with 32-bit floating point numbers.

3.5 Noise Shaping

Although the use of noise shaping was a stretch goal, it would have been relatively easy to implement once the filter was integrated. The introduction of noise shaping requires a system to take advantage of a feedback loop that acts as an identity system relative to the input signal. Since the quantization error is introduced elsewhere in the loop, the power spectral density of the quantization error is modified heavily. Namely, the error is shifted one clock cycle and added to itself. Under the assumption that the error is white noise, which admittedly can be potentially false, the low-frequency content of the signal is significantly attenuated while the high-frequency content is amplified. Since white noise is uncorrelated at all time shifts except zero, the shift and add tactic amplifies the total noise power, but moves it away from low frequencies. Once the noise has been shifted away from baseband, the lowpass filter that improves the sample precision of the input signal attenuates most of the noise power.

In terms of implementation, this is as simple as providing a feedback pin on the FPGA and assembling integrator and subtractor circuits with suitable time constants. Noise shaping is an elegant and pleasing technique, considering the benefit it has relative to the difficulty of implementation.

3.6 Discussion

The original intent behind this project was to explore the implementation of a Delta-Sigma ADC. Although the ADC itself was never integrated into the system, a version that functioned correctly in simulation was very satisfying to develop. Much of the time I spent on the project was actually dedicated to developing a base-level system with Hugo. While a smooth waveform would have been pleasing to display on the screen, a basic display of the heart rate is the natural goal of any heart rate monitor, and we completed such a system successfully. Since the analog circuitry worked correctly, the final system displayed during the checkoff used an edge counter that totaled the number of heartbeats over a 10-second period and displayed an average heart rate.

We had some interesting challenges that required I spend some of my time assisting Hugo in developing the graphics and fft modules for the system. Displaying the waveform on the screen and keeping it up to date was a challenge that Hugo and I worked on together, for which I recommended he use a two-port memory that would allow him to reload and access the data simultaneously. The instantiation of the fft module was a difficult process to understand and move confidently through, so I assisted Hugo in understanding how to generate and manipulate the fft module, and how to interpret its output.

4 Digital Signal Processing by Hugo Malpica

After generating the high precision bits in the previous stage, those bits need to be processed so important information can be extracted from them. In a similar fashion the bits need to be stored so they can be displayed so the waveform can be displayed on the monitor. To do this, implementation of signal processing is required. The system diagram of the digital signal processing block can be seen in Figure 2.

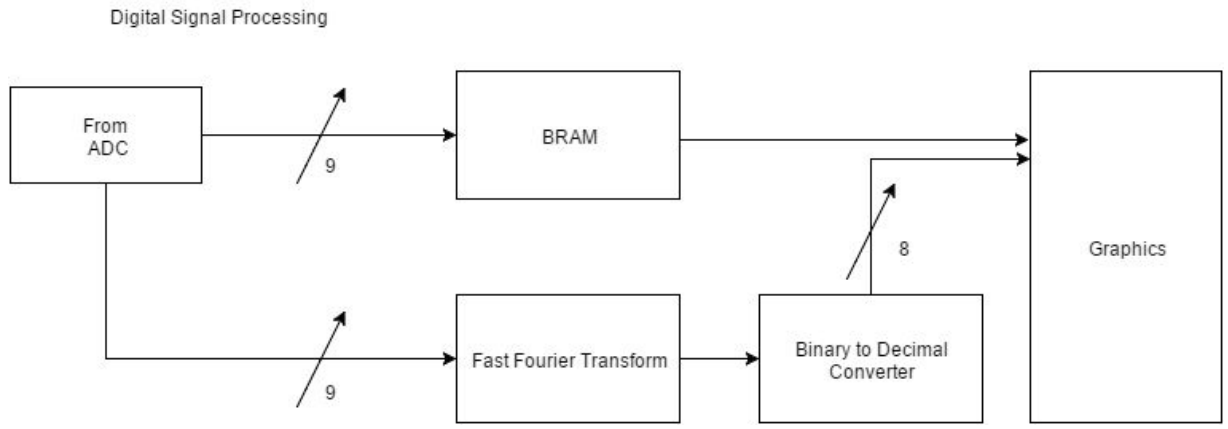

Figure 2: Digital Signal Processing Block

4.1 Storing Bits

The first task in the digital processing block was to instantiate a memory block to store all the incoming bits the Joe was giving after going through the analog circuitry, shown in Figure 3, and then passed through the ADC. This was an important step since the bits were being generated at a 256Hz clock while all of the other modules run off a 65MHz clock on the Labkit (25MHz clock if on the Nexys4).

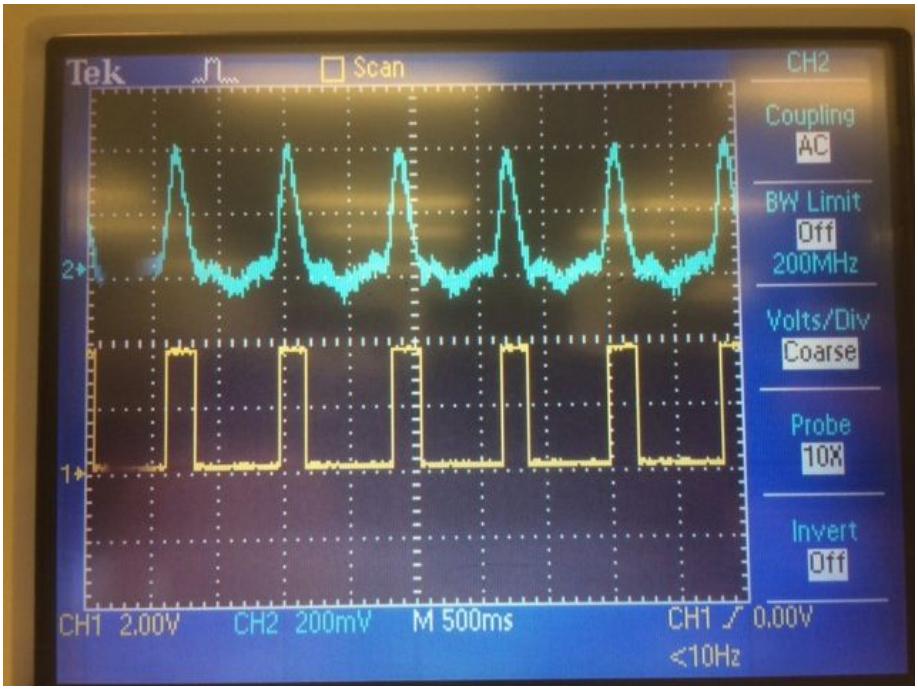

Figure 3: The signal waveform of the pulse ox probe (top) and analog circuitry (bottom).

To carry out this task a dual port RAM was made from the Intellectual Property (IP) cores that come with the FPGA. A dual port RAM has two main functions: a read function and a write function. To store bits you have to enable the write function on the RAM at the same clock at which they were generated and assign them an address at the same clock speed.

When ready to read the values on the RAM, the clock speed used to display images is then used to go through the addresses and read the values associated with the addresses. This enables these values to be then used by the Graphics Block to then display the original waveform fed in which is detailed in section 5.4.

4.2 Fast Fourier Transform

One of the most important factors tasks in the digital signal processing block is to extract the heart rate from the incoming data and convert it to beats per minute so it can be displayed on the screen. In order to do this, an Fast Fourier Transform was performed on the ADC data bits.

The FFT module takes in the data bits at the clock speed at which the bits are generated and produces a set of real and imaginary output values. These values are put into a process heartrate module where the magnitude is calculated along with the addresses they correspond to. It is then easy to then go through the first twenty addresses generated and pick the highest magnitude from those addresses. The address is then stored into a register.

After taking the address of the highest magnitude, the address is multiplied by 60 since 1Hz is equal to 60 beats per minute (bpm). By then right shifting by an appropriate value

determined by taking the depth of the FFT, it is possible to determine an approximate binary value for heart rate.

The reason this approach is taken is that the addresses generated by the process heartrate module are integers representing a value of zero to our generation frequency minus one, in this case 255 since our generation frequency is 256Hz. This set of values is then divided by the depth of the FFT module to determine the true representations of these addresses, namely decimal representations. By shifting right by the value it would take to get to a value of 1 from 0 using these decimal representations, it is possible to obtain a 8-bit binary representation of a the heart rate in bpm, without producing a value our binary to decimal converter cannot handle, as stated in the next section.

4.3 Binary to Decimal Converter

After obtaining the binary representation for the heart rate in bpm, it is necessary to take that binary representation and turn it into three individual decimal representations; one for the hundreds place, one for the tens place and one for the ones place. To carry out this task, a Binary to Decimal Converter was created. With this module, the binary number created three separate wires that contained numbers that were used to select a specific case in the graphics module.

4.4 Discussion

Overall I say that this portion was interesting to implement. During the course of this block many problems were encountered. Initially for the generation of individual numbers used to select for a specific case in the graphics module, a mathematical operation module was created. This turned out bad for several reasons. When this module was first integrated for testing, the input to this module was controlled by switches. The computation time that it took to calculate these values was too much and any changes caused by the switches would greatly distort the images displayed on the monitor.

In order to correct this issue, the module was pipelined to try to get a more stable image by breaking down the computation into simpler stages. Although this managed to reduced noise, the images were still distorted. The computation time seemed to be affecting the images so a almost zero computation binary to decimal converter was instantiated. This solved two problems: it took in the switches as inputs for ease of testing and stopped the distortion of the images.

Another problem encountered was the instantiation of the FFT module. The FFT module can be generated from the current IP cores but to facilitate our implementation, I used the module given to us by 6.111. Understanding the FFT module was difficult and when implemented it did not give me the output I was looking for. After several dozen iterations of trying to make it work, a simpler solution of using an edge counter was used to do the job of the FFT. Both pieces of code can be found in the submitted verilog files in the 6.111 website.

Although we were able to implement FFT on the Labkit, the same could not be said for the Nexys4 due to its higher order complexity and different methods of instantiation. If it could be implemented on there, then the whole project could have been completed on both FPGAs.

5 Graphics by Hugo Malpica

After processing the information, the most important part was to display the data on the monitor in a pleasing manner. This way it would be possible to interpret it easily. Figure 4. demonstrates the block diagram of how this portion of the system was implemented.

The foundation of the graphics module was first implemented on the Nexys4 for it high availability of travel but later switched to the Labkit where the rest of the project was complemented on.

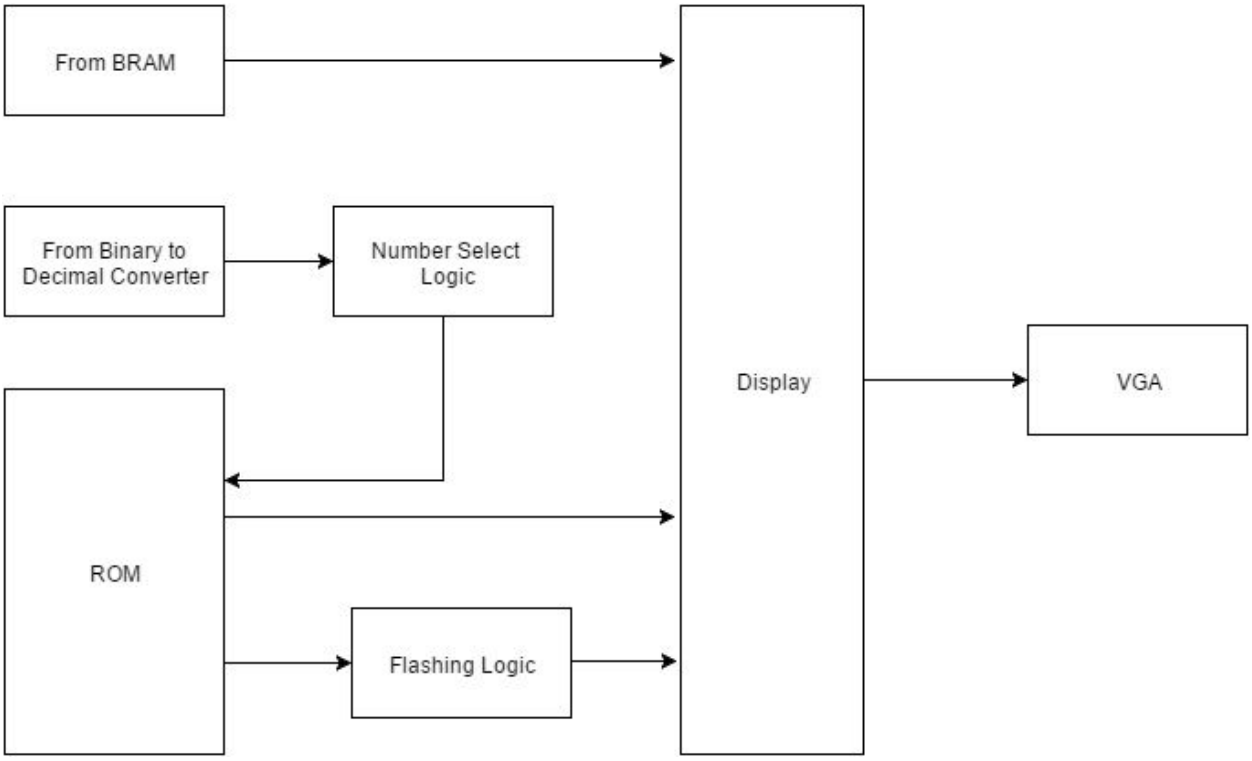

Figure 4: System Implementation of the Graphics block.

5.1 ROM and Images

The first thing that has to be done is implement various ROMs to store images on the FPGA so that they can be displayed. This is done by taking an image, opening the image on photo editing software and transforming it into a .bmp format. Then by running a matlab script, a

.coe file can be generated for the image, as well as .coe files for the three color maps needed to display titles and images.

After the .coe files are generated, a block memory module has to be instantiated for each picture. In this case, a simple ROM was enough for each image. These ROMs were then used inside of a picture blob module to generate the bits necessary to display color picture. All of our images used 16 bit color with the exception of the beating heart which used 256 colors. Figure 5 shows a sprite of all the images used.

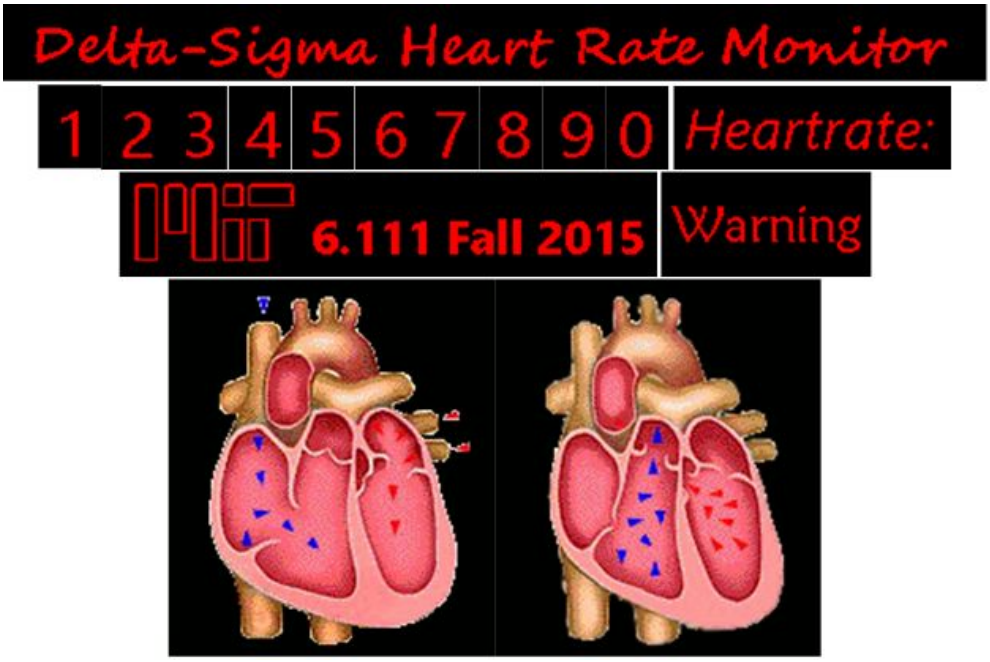

Figure 5: All the pictures displayed on the heart rate monitor.

The same process is used on both the Labkit and the Nexys4, but the only difference is that the Nexys4 uses a 25mhz clock and 12 bit RGB while the Labkit uses a 65mhz clock to beat clocking issues and outputs 24 bit RGB.

5.2 Beating Heart and Warning Images

One of the best parts of having the Delta Sigma Heart Rate Monitor is having a visual representation of a heart beating as well as a warning system when heart rate becomes too high or too low.

To implement the beating heart, two pictures were stored into two ROMs and then flashed, alternating between the first heart image and the second. To flash the images, a 8 bit counter was incremented on the negative edge of the signal vsync. Using a conditional operator, an 24 bit register was used to store either the first or second heart image to display at the switching rate of the nth bit of the 8 bit counter. By using the heart rate value and setting

thresholds, the heart was able to flash at appropriate speeds. If the heart rate turns out to be 0, logic was made so only the first heart was displayed representative of a heart that stops beating.

When testing this module, it was recorded that the 4th bit would switch at a rate approximately of 80 beats per minute and the 5th bit would switch would at a rate of 60 beats per minutes. Given this data thresholds were made for ranges of heart rate.

A similar implementation was used to generate the warning signals. A threshold of above 180 beats per minute and below 41 beats per minute are considered dangerous. Since this is the case, a warning image was set to flash only when the current beats per minute passed these thresholds in a similar fashion to the beating heart. A picture of how this looks like can be seen in Figure 6.

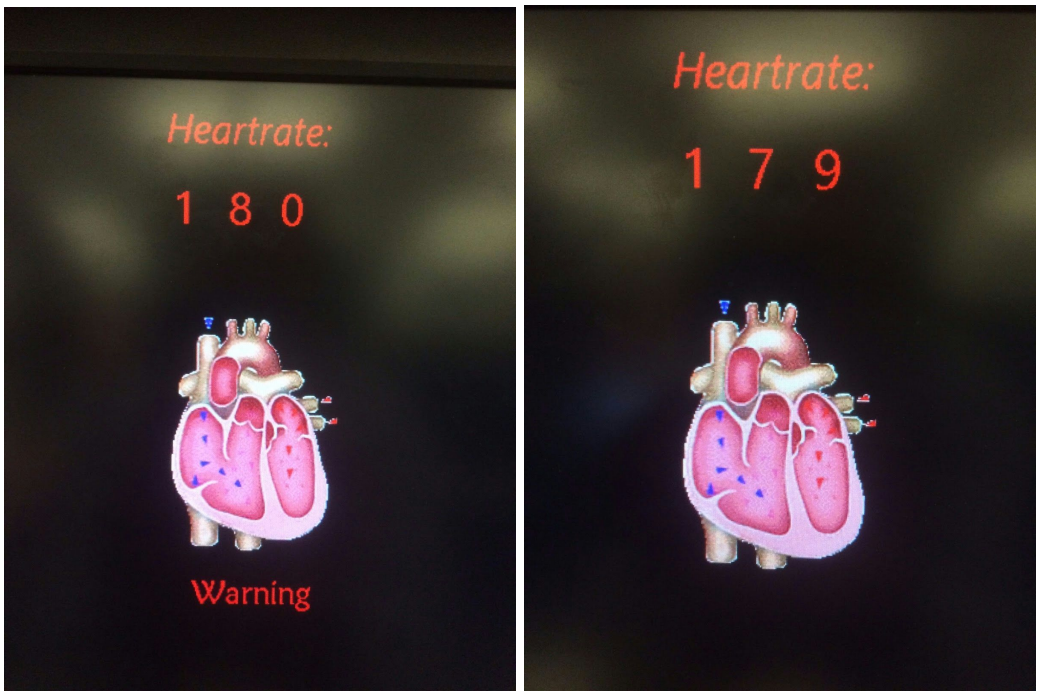

Figure 6: Demonstration that at or after 180 beats per minute heart rate, the warning signal starts.

5.3 Heartrate display

The most important aspect if not the most important aspect of a heart rate monitor after the heartwave display is the heart rate display in beats per minute. In order display the images without distortion, it was better graphics wise to use ROMs to store the images. Due to this some assumptions were made about the maximum heart rate our system should be able to handle. It was determined that 299, would be more than enough and it was this value that determined the number of instantiations needed. In the end 23 instantiations of the numbers 0 to 9 were needed.

These numbers were instantiated in the same region, meaning 10 instantiations would be in the ones place, 10 in the tens and 3 in the hundreds with a register handling the memory of

which number image was stored there. A case statement was used to determine which number in the hundreds, tens and ones place would be displayed by using the case selectors to be the signals generated by the binary to decimal converter described in section 4.3. An image of how this can be found in Figure 6.

5.4 Heartwave display

To have a heart rate monitor it is important to be able to demonstrate the waveform of the patient's heart to see if any irregularities are present in the signal. To do this, a blob module was instantiated to follow the x and y coordinate values of the pixel during a given cycle.

When first testing this module, a heartwave signal was not available. A sine wave look up table (LUT) was made and running through the LUT at the 65mhz clock, the sine wave was able to be displayed as points on the monitor as shown in Figure 7.

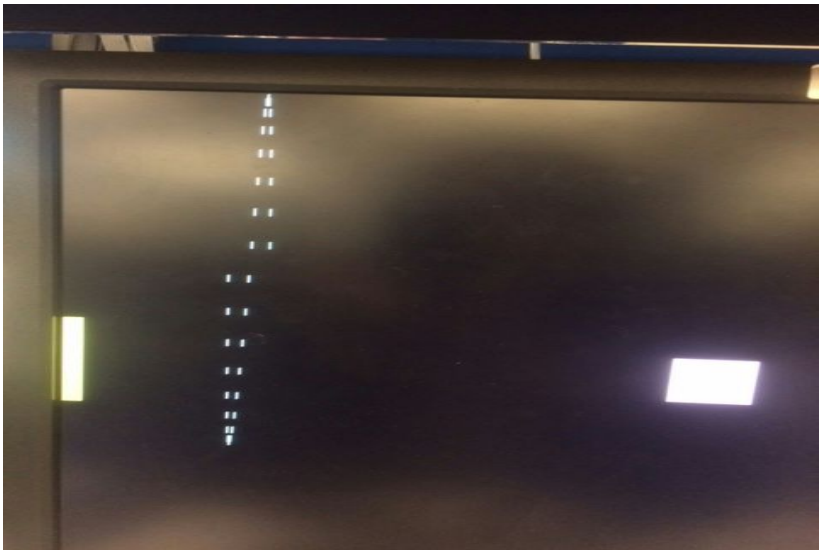

Figure 7: Sine wave display from LUT.

The second phase of testing was to make sure that this module worked with more complicated images. A .wav file was converted into a .coe file and then made into a LUT. In the same manner as testing the sine wave LUT, a nice alternating oscillatory signal was produced.

After testing and having a waveform that could be displayed, addresses from the BRAM were called and stored in the x-coordinate input of the blob display module and the values stored in those addresses were given to the blob display module as y-coordinate inputs. Figure 8 shows the end result of waveform given as a square wave. In the future, the waveform signal could be slowed down to be able to see the waveform more clearly while allowing for interpolation to connect the bits generated on the monitor.

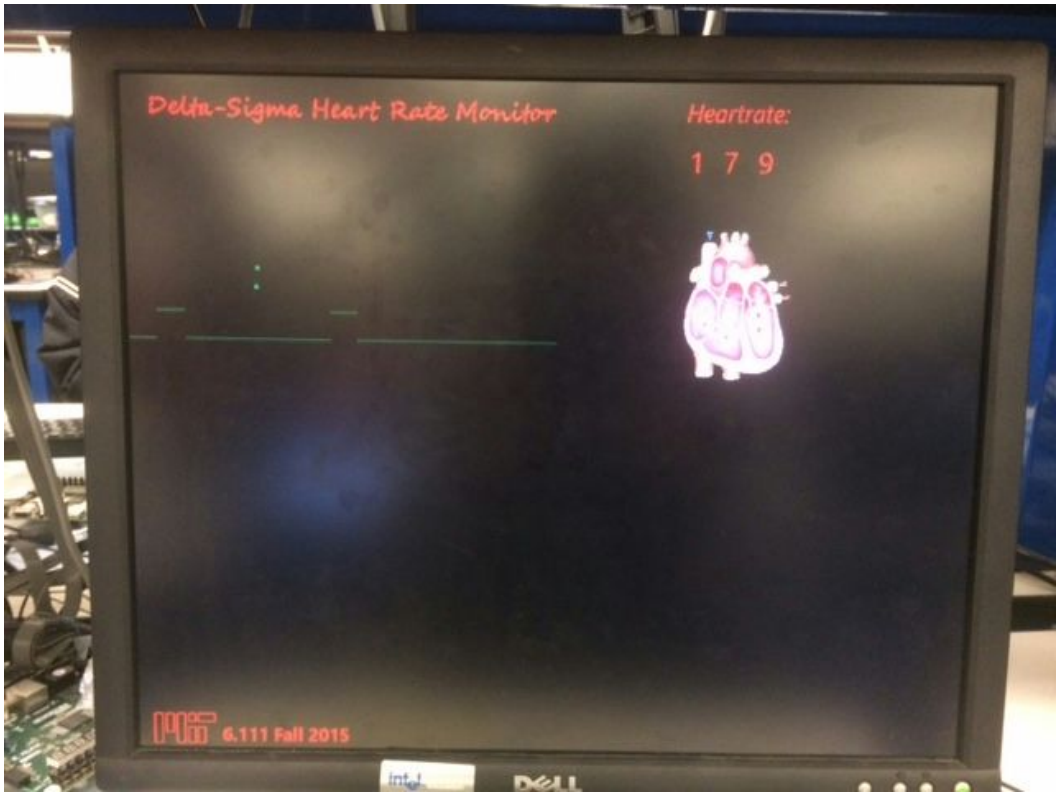

Figure 8: The green heart rate waveform.

5.5 XGA vs. VGA

As mentioned previously, this project was made both on the Nexys4 as well as on the Labkit. For the demonstration, the Labkit was used. Although the process used to display the images are both the same there is a difference to look and that is XGA vs VGA. The Labkit uses XGA to display 1024 by 768 pixel resolution while the Nexys4 uses VGA to display 640 by 480 pixel resolution. The differences in resolution affects how many bits you can introduce into the system. The Labkit uses 24 bit RGB and the Nexys4 uses 12 bit RGB but the Nexys4 does have a slightly better graphics card. An image of the video output of the Nexys4 can be found in Figure 9.

Figure 9: The video output generated by the Nexys4.

5.6 Discussion

In the end, the graphics module was fun to work on. I learned much about the positioning and the implementation of pixels which was one of my personal goals as well timing issues. Something to keep in mind about using ROMs and the Labkit is that the labkit needs precise bit registers while on the Nexys4, sometimes you can get away with certain errors when making your ROMs..

Now for the most part, trying to detect an error is graphics is hard and it can be affected by the computation time of other modules trying to select an image to display. To remedy this situation computational logic has to be used as well as pipelining.

Another issue that comes with graphics is using the BRAM to display the waveform. In the future to slow down the output of the BRAM to the monitor, what can be tried is implementing another BRAM at an intermediate clock speed to slow down output. The timing would be more difficult but it could generate a cleaner waveform in the end.

And to anyone working with programming graphics, something that needs to be remembered is that every time the FPGA is programmed, it needs to be power cycled first or else the images will come out distorted in some sense.

6 Results

In the end, our Delta Sigma Heart Rate Monitor did not meet all the standards the team set for it. This can be attributed to a multitude of reasons. One of the main reasons was that when this project was started, the team wanted to implement this on the Nexys4. After working for a while on the Nexys 4, the team came to the conclusion that some of the modules that needed to be implemented were hardware intensive and the Nexys4 was not enough. This made us switch over to the Labkit.

Originally we wished to display the digitalized version of the analog heart waveform produced by the pulse oximetry probe but this was only possible by implementing the Delta-Sigma ADC in the first portion of this project. In trying to implement the Delta Sigma ADC, it was discovered that a high precision filter was needed and by the time this filter was stated to be implemented, it was too late and a pulse wave modulated waveform of the heart was all we could display.

Without the first part of the project being complete when originally stated on the timeline it was difficult to fine tune some aspects of the graphics. Using test signals such as sine waves and beating heart sound files demonstrated that if the heartwave signal had been properly implemented and passed to the graphics and signal processing modules, then a more stable and robust system could have been built.

On the other hand, the team believes that with the exception of the pulse width modulated waveform signal, the other aspects of the graphics were detailed and smooth, without many glitches and they displayed what they needed to display.

7 Further Implementation

These are some of the things that were thought of as how this project could be taken further as well as a list of improvements that could be made to this design:

1. Slow down the signal waveform so that it can be seen more clearly.
2. Make the beating of the heart proportional to the heart rate calculated.
3. Make a memory bank to store previous signals to a certain point to redisplay those images using a FSM.
4. Show the FFT histogram of the heart rate.
5. Interpolate separate bits to make them look connected.
6. Integrate the noise shaping feedback and circuitry to improve precision.

8 Acknowledgments

We would like to acknowledge and give our thanks to Gim Hom, Mirem Bamforth, and Ariana Eisenstein. Gim was instrumental in giving us ideas that eventually led to our final project and offered his expertise in design and debugging when we needed it the most as well as being our project advisor, and guiding us through the design process. We thank Miren and Ariana for being incredible TA's and helping us out, during the hardest of times. Last but not least, we thank all the 6.111 students, the LA's and the rest of the staff for an awesome semester.

Disclaimer All pictures shown in this report belong to the students who's section they appear in. This means that Joes's pictures appear in his section, Hugo's pictures appear in his section. Unless given credit otherwise, these figures and pictures belong to these students.