A Two-Input Polygraph

6.111 – Introductory Digital Systems Laboratory
Final Project

Archana Venkataraman, Christopher Buenrostro, Isaac Rosmarin

TA: Theodoros Konstantakopoulos

May 18, 2006

Abstract

A two-input polygraph was implemented on the Xilinx Virtex2 FPGA. The system relied on physiological data in the form of skin conductivity and pulse rate, and made a binary decision as to whether or not the subject is lying. The project was divided into three portions. The first part consisted of designing and implementing the physiological sensors. The second part functioned as the main digital control unit for the system. The final portion was the output display which displayed both dynamic data as well as the T/F decision. Due to the modularity of the design, each of the portions were designed and implemented individually. Integrating the system, therefore, posed few problems. Although the digital part of the system functioned correctly, the lie detector could not be tested very effectively due to the poor quality of the sensor data.

Acknowledgements:

Before launching into our project details, there are several people that we need to thank for a wonderful semester and without whose help we all would have finished the course just a little more sane a sleep-deprived.

First and foremost, we would like to thank Professor Chandrakasan and his unbounded enthusiasm for all things digital. His lectures were always informative, and he was always available for help, especially during hectic times before assignments were due.

We would also like to thank our wonderful TA's Theodoros, Jae, and Javier for all their help late at night in lab. We would also like to thank our lab assistant Gim Hom for yet another semester devoted to course 6 students. He was always knowledgeable and helpful when it came to debugging code that really should work, but for some reason kept crashing.

Last, but not least, we would like to thank Nigel for giving us (well, just Archana) unwavering support through all those hectic hours in lab. His compassion was much appreciated, especially since ModelSim does not share his attitude of thoughtfulness and helpfulness.

Table of Contents

List of Figures

1.0 – Introduction and System Overview

In this project a two-input polygraph was implemented using the Xilinx Virtex2 FPGA. The physiological inputs used are pulse rate and skin conductivity. These signals were chosen because they are fairly easy to measure and interpret. During times of emotional stress, such as when the subject is made uncomfortable and forced to lie, the pulse rate increases. Likewise, the subject is more likely to sweat, which increases his or her skin conductivity.

The project is divided into three portions: the analog sensors to measure skin conductivity and pulse, the digital control unit, and the output display.

The two sensors were implemented on a breadboard using analog circuitry. The schematics for these circuits are shown in section 2. The ADC0804 was used to convert the analog signal into a digital one, which was then stored in the main memory.

The Digital Control Unit forms the backbone of the lie detector by integrating the digital data acquisition, data processing, and system control signals. It has three major roles in the overall system. First, it captures and registers user input signals. This is implemented in a manner similar to the Walk Register in the Traffic Light Controller lab. Second, it controls the sequence of reads and writes of the sensor data to the main memory. Since it is the only module which controls reading and writing, data acquisition becomes easier and more straightforward. Lastly, the Digital Control Unit implements the decision algorithms which output a binary T/F decision as to whether or not the subject is lying.

The video display portion of the project serves two purposes. First, it communicates the result of the algorithm's decision to the user. Secondly, it displays the data that is actually being collected by the sensors. This allows an operator to make their own decisions on whether or not the person is lying, and it facilitates tuning of the overall system. The screen display consists of three main sections. The first section displays the current data from the sensors, so that the operator can see how the subject is currently responding. The second section displays screen captures of the fist, that can be stored and later compared against new data. Finally, there is a section that displays the data from the output of the decision making unit, either "true" or "false".

A simplified block diagram of the overall system is shown in figure 1.1 on the following page. (Note that the individual sections will be addressed in greater detail in the following sections). As seen, the system portioning was chosen to exploit modularity. The there is minimal interaction between the different parts of the system, meaning that all modules can be implemented and tested separately. Furthermore, once all modules have been tested, integrating all parts of the entire system becomes fairly straightforward

The polygraph presented an interesting challenge for us to tackle as a final project. First of all, it is an intriguing real-world application. Second, it combines many of the elements which we learned about in class such as analog interfacing with the sensors, signal processing, and output display. This allowed us to carry the knowledge that we had acquired thus far to a higher level. Additionally, a "successful" lie detector test is as much an art as a science, so it was interesting for us to explore more biological and psychological issues of the project.

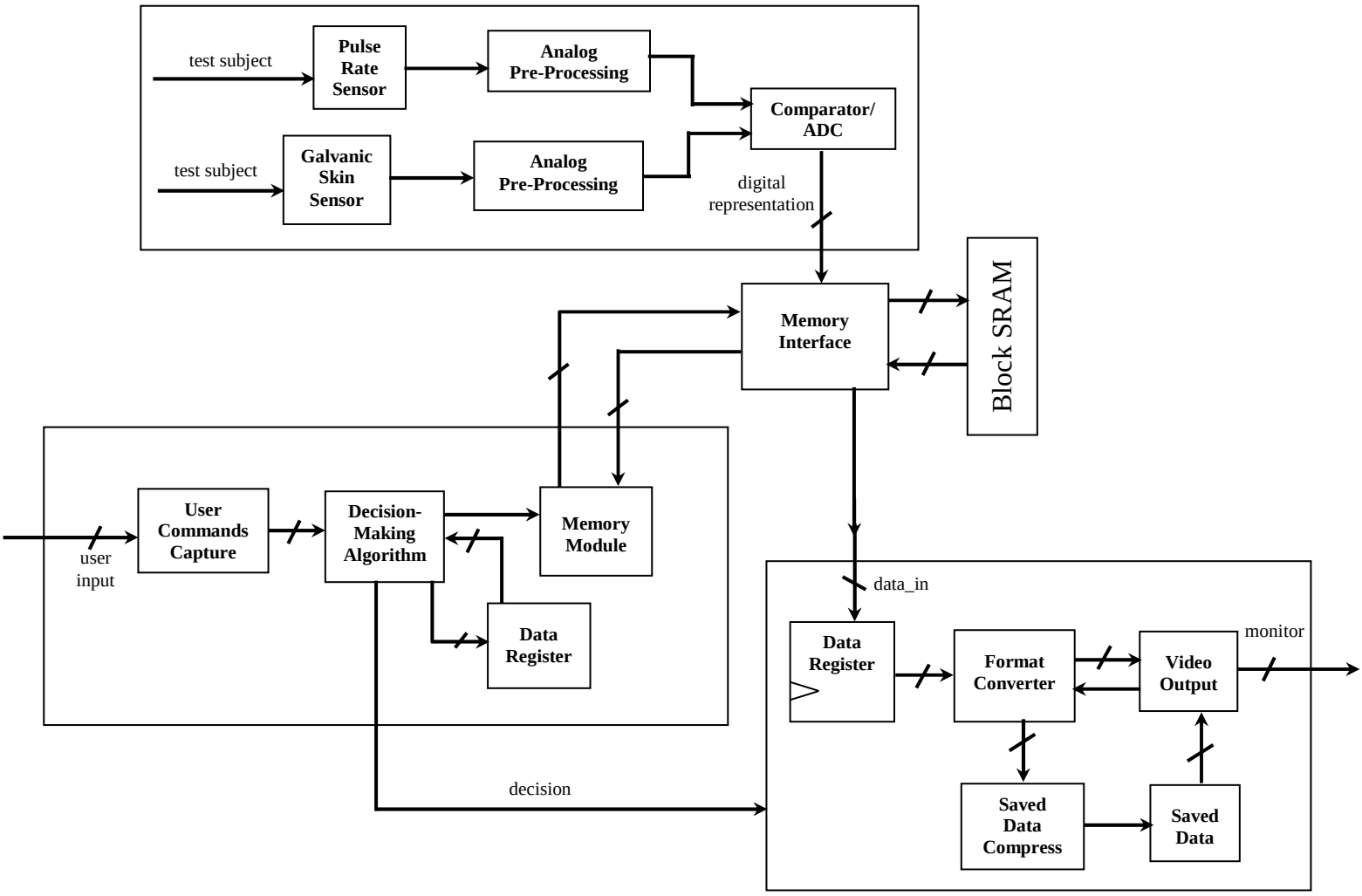


Figure 1.1 – Block Diagram of the Overall Lie Detector System

2.0 – The Physiological Sensors

The device interface portion of this project can be separated into three major stages. The first stage is the biological sensors. This stage includes the skin conductivity sensor and the heart rate monitor. The second portion is the analog circuitry. This stage includes the circuitry used to clean up the biological signals and put them into the proper form for the analog to digital converters. The final stage is the actual interface with the lab kit, which includes the modules written in Verilog HDL to gather the real world data. This section of the report first presents the hardware description and the software implementation.

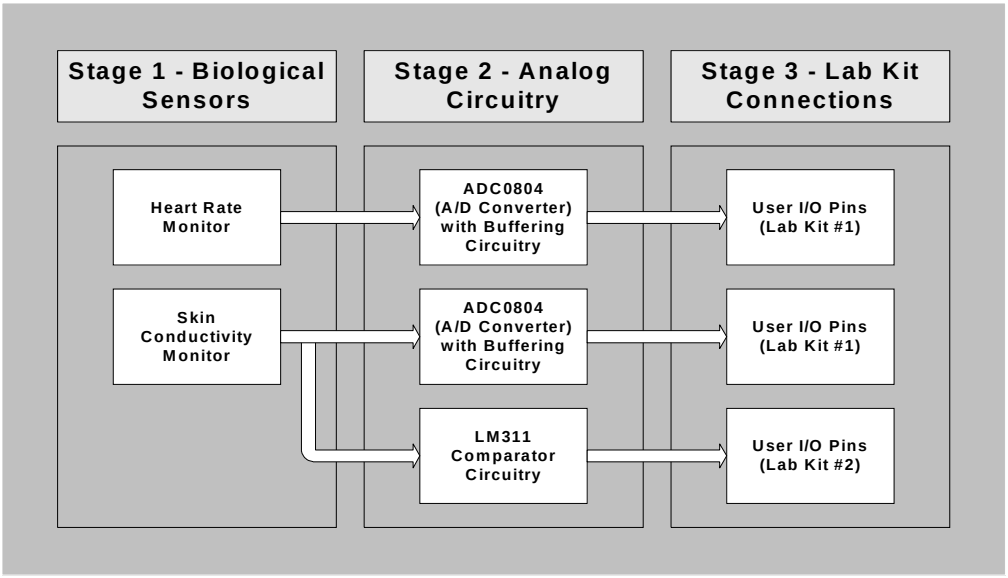


Figure 2.1 – Hardware Block Diagram.

To begin the hardware description, the first stage proved to be the most difficult to implement. The body, it appears, has a lot of noise (in terms of electric voltage). Having never interfaced with biological sensors before, the amount of filtering required for clean signals proved to be a project unto itself (and more in the analog realm). As a result, the biological signals were not very clean.

Monitoring the heart rate was done using Nicolet patches attached to the inner elbows along with a single patch attached to a point far away from the test subject as a ground reference. These signals were then fed to a differential amplifier with a large common mode rejection ratio. A circuit was constructed using the circuit diagram as given in the figure on the following page. This circuit was constructed on a single breadboard for ease of switching components in and out. A distinct heart rate spike was detectable, but only sporadically.

The skin conductivity monitor proved to be very simple to implement. This monitor gave out a perfectly clear signal every time. The one difficulty with this sensor was the time it took to notice a significant change. However, this complication was more of a theoretical one than an actual circuit problem primarily due to the fact that the circuit did exactly what was expected. The schematic is shown on the following page.

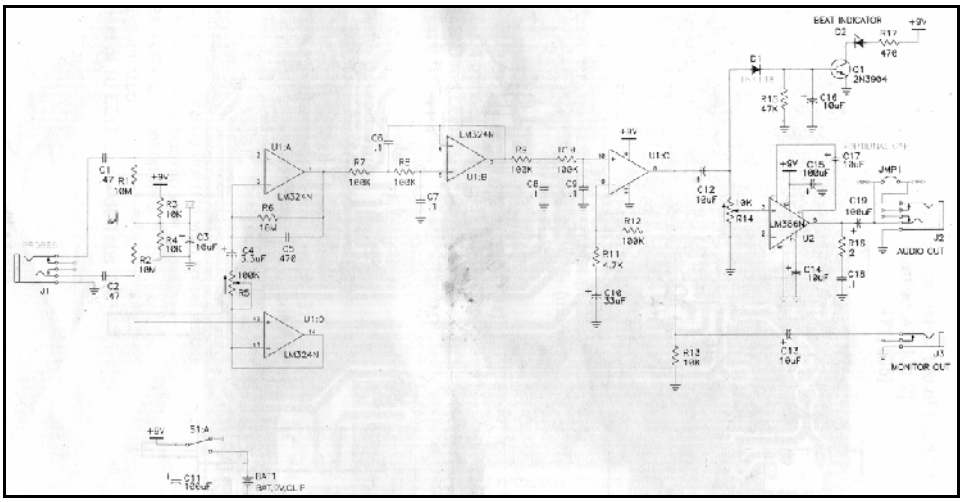


Figure 2.2 – Electrocardiogram Generator

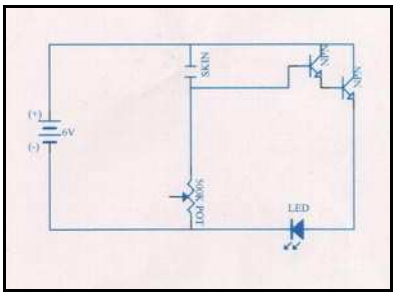


Figure 2.3 – Skin Conductivity Detector

The second stage was to build the analog circuitry to convert the signal to something usable by the analog to digital converters. This required the simple use a single gain stage for each input device. The major difficulty while attempting to do this is keeping the voltage in the range such that you do not harm the ADC chips while putting data into them. Aside from protecting the chips, the gain stage must provide a way to take a small variation and turn it into meaningful data without amplifying noise.

The ADCs themselves were National Semiconductor ADC0804s. These chips provided more than enough time processing required for the 5-Hertz signal we expected from the heart rate signal. The skin conductivity did not present a timing issue primarily due to the fact that the output would not have much variation. For our statistical analysis, the resolution of the ADC proved to be more than adequate.

To begin the software description, the ADC_module state design has three divisions: the initial firing sequence consists of INITIAL_ASSERT, INITIAL_WAIT, and INITIAL_DE_ASSERT, the IDLE state which manages the converters when they are not required to read, and the remaining states which service data requests.

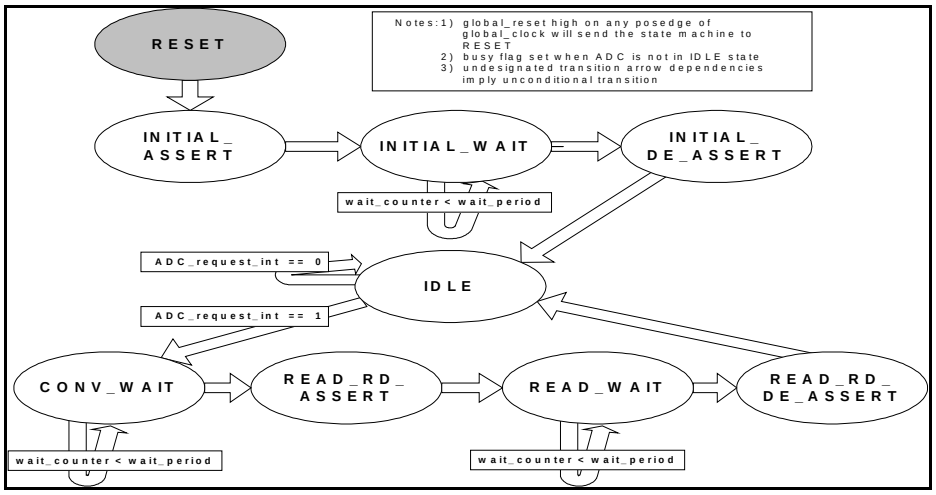


Figure 2.4 – Analog-to-Digital State Transition Diagram

The initial firing sequence is required for proper operation of the ADC0804. The specification sheet states that for the chip to work properly there must be an initial pulse on the write enable line (hereafter referred to as "WE_bar"). This initial pulse must be long enough to register as a valid pulse thus requiring the INITIAL_WAIT state.

The remaining states tailor to the timing requirements on the specification sheet depicted below. The most important time constraint is that the read strobe must be at least eight clock cycles after the interrupt signal even if the interrupt pin does not signal state transition. This state transition design operates the ADC0804 in what the specification sheet refers to as "free-running mode". In this mode, the user ties the chip select pin (active low) to ground then WE_bar and the read enable line (hereafter referred to as "RE_bar") together. Doing this allows a single line strobe to both collect data and send a new data request at the same time.

The RAM_interface module state transition diagram has two major components: the read and write sequences. Both sequences are very straightforward. The READ_ADDRESS state enters the address requested to the block RAM's external input register as specified in the Coregen specification sheet. This READ_DATA state triggers a wire inside the RAM_interface module to latch the data on the output data bus of the Coregen RAM. The READ_VALID state indicates to the algorithmic modules in the project that the data on the RAM_interface bus is valid data. Handshaking proved to be the most challenging aspect of this module. In order to ensure proper handshaking between the RAM_interface module and those requesting data, the READ_VALID state was added. This state gives clear signaling that the RAM is done reading.

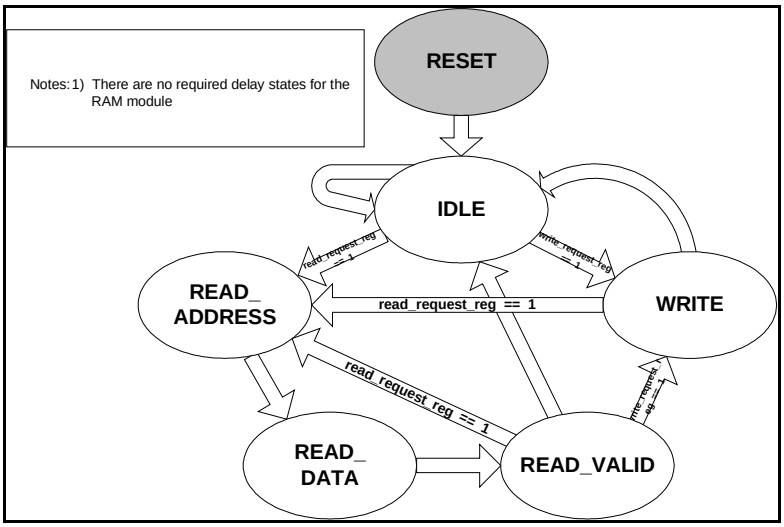


Figure 2.5 – RAM State Transition Diagram

Finally, the Recorder module simply combines the device interface and memory storage into a single package that can be called on to record data. This modules exists so that modules doing data analysis can simply press a record button and be sure that the analog to digital converters will continuously run and gather data as the tester desires. In this way, the tester can simply press a record button and the memory and ADCs communicate with each other automatically. If any of the other modules are busy (indicated by a single wire contained in each respective module), then the Recorder will signal to the requester that it is currently waiting and stay in its current state. The state transition diagram for the Recorder module appears below.

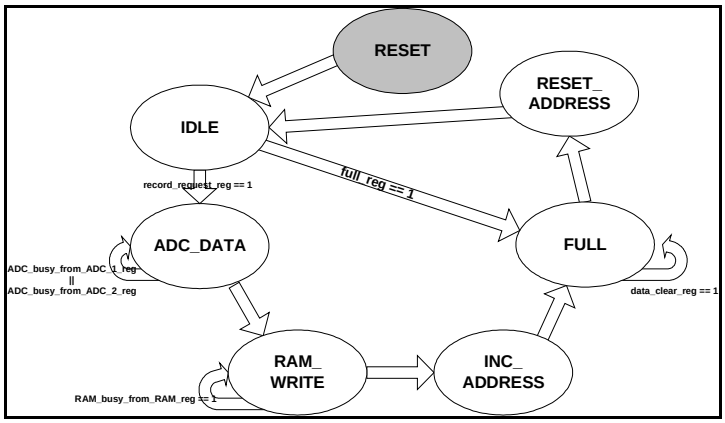


Figure 2.6 – Recorder State Transition Diagram

The last piece of the device interface is connecting to the lab kit through the user input/output ports on the lab kit. To do this, eleven wires are connected directly to the kit and respective pins on the breadboard. In order to receive inputs the data bus as input, eight pins for each port reside in high impedance so that data can be written to the bus without contention.

Debugging was done with the aide of the revised alphanumeric display code. This proved to be the quickest tool for easy verification at low speeds (i.e. switch testing through the lab kit at very slow rates). To do further testing at actually processing speeds, the logic analyzer proved invaluable. This made quick debugging on what was actually going on in the circuit. Of course, the analog circuitry was debugged using scope probes and waveform generators. If the hardware implementation of this project were to be redesigned, much more attention would have been given to making back up input devices in the event that (which was the case) the desired devices did not provide adequate, or accurate, data.

3.0 – The Digital Control Unit

3.1 – Design Considerations

The Digital Control Unit (DCU) serves as the main digital processing and control block for the overall system. It can be divided into three major blocks. The first block, referred to as the User Capture block, is used to capture the three main control signals input by the user: global reset, question type, and store data command. The second block, referred to as the Memory Module, controls the sequence of reading and writing the sensor data to and from the main memory (implemented using a Coregen BRAM). The final block, referred to as the Digital Decision-Making Unit (DDMU), implements the decision-making algorithm which analyzed the sensor data and made a binary T/F decision.

The design and implementation of the DCU were shaped by two key considerations. The first, and perhaps most important, was modularity. Rather than using one large module for all the necessary tasks, the functionality was partitioned into sub-modules, each of which implements a basic task. Not only did this facilitate debugging, but it helped to prevent oversights, and made the design easier to integrate with the reset of the system.

The theme of modularity runs throughout the DCU. For example, the Memory Module and Digital Decision-Making Unit are implemented as separate blocks which pass information to each other using handshake signals. This allowed each block to be implemented and debugged independently of each other, and allows for greater flexibility if one part of system needs to be altered. A second example is that both the Memory Module and the DDMU consist of a Major FSM and several Minor FSMs. The Major FSM is used to control the data acquisition and/or processing sequence. The Minor FSMs, on the other hand, are used to perform the actual computation or desired functionality. The advantage of this architecture is that each part can be implemented, tested, and modified more easily than if all functions were combined into one large module. For example, since the actual T/F decision-making occurs in one Minor FSM, if it is determined after calibrating the system that another decision criterion should be used, then it is relatively simple to change.

The second consideration when designing the DCU was the collection and storage of data. The first issue to consider was the speed and efficiency of the system. Since data acquisition would last on the order of minutes for each question analyzed, a ping-pong memory structure was used to speed up the data processing. Under this ping-pong architecture, the total memory needed is divided into two block RAMs. While one RAM is being filled with sensor data, the other is being read and processed. This allowed continuous data acquisition, meaning that a question could be analyzed immediately following the previous question.

Another challenge was devising a method to handle the fairly large volume of sensor data that would be needed for an entire lie detector exam. Assuming the exam lasts approximately 20 minutes and that 16 bits total are needed to represent both the skin conductivity and pulse, at the specified 50 Hz sampling rate, a potential 1 million bits needed to be stored for each exam. Although this may not seem very much, only a portion of the data is actually useful (most of the data occurs during the dead time between questions), and only the computed statistics are really necessary to make the T/F decision. Therefore, on-the-fly processing was implemented. As data is read to the DDMU, it is processed and analyzed as a stream. The resulting statistics (rather than the sensor data) are stored in a separate data register. The raw data in main memory is then overwritten for the next question.

A simplified block diagram of the Digital Control Unit is shown in Figure 3.1 below. A more detailed figure which includes wire names can be found in the Appendix

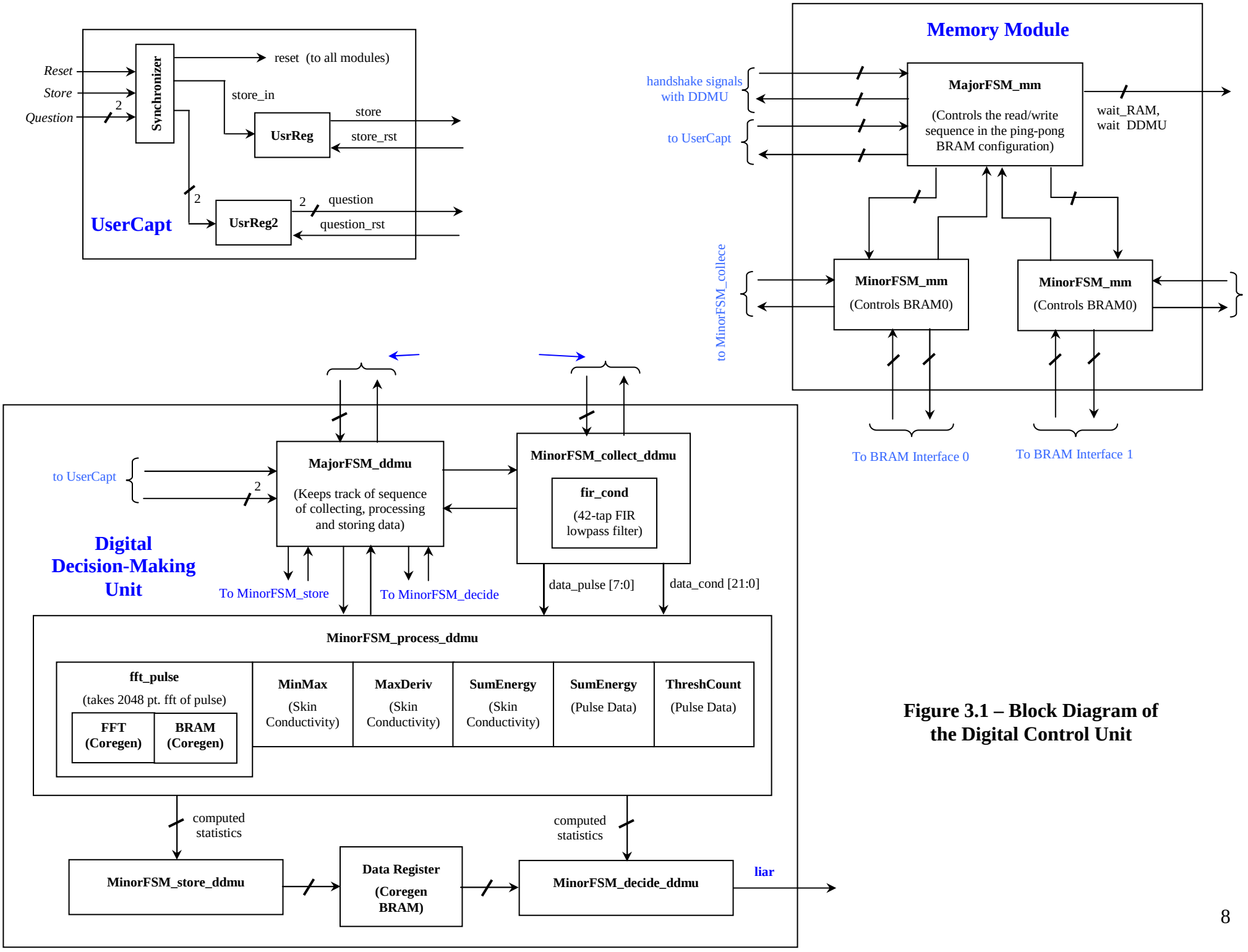


Figure 3.1 – Block Diagram of the Digital Control Unit

3.2 – Module Descriptions

As stated above, the Digital Control Unit is divided into three separate blocks: the User Capture block, the Memory Module, and the Digital Decision-Making Unit. The following sections describe each of these three blocks as well as their internal modules.

Note: Please refer to the Appendix for the Verilog code

3.2.1 – The User Capture Block

The User Capture block takes care of the three user inputs for the DCU: *reset*, *question*, and *store*. The *reset* command initializes the system. The *store* command causes the ADC data to be stored in the main memory and the data processing to start in the DDMU. The *question* input is a two-bit value which indicates to the system how to treat the data being analyzed. A value of 0 indicates an irrelevant question and is not analyzed at all. A value of 1 indicates a control question meaning the statistics are computed and stored in the separate data register. A value of 2 indicates a relevant question meaning that the statistics are computed and compared against previously-stored statistics for the control questions.

The User Capture will first debounce and synchronize the inputs. Once *store* is pressed, it will hold the value of *store* and *question* until the registers are reset. This module is a variation on the Walk Register which was implemented in the Traffic Light Controller lab.

3.2.2 – The Memory Module

The Memory Module controls the sequence of writes (from the ADC to the main memory) and reads (from the main memory to the DDMU). It uses a Major FSM and two identical Minor FSMs to implement the ping-pong memory architecture.

The Major FSM – The Major FSM supervises the memory access. It interacts with the DDMU using handshake signals (two signals are used to indicate which block the Memory Module is currently writing, and two signals are used to indicate which block the DDMU is currently reading). In addition, it is responsible for resetting the *store* and *question* registers after data has been written to both memories, and it sends start signals to the Minor FSMs to indicate when to perform a read and when to perform a write.

The Minor FSM – The Memory Module uses two identical Minor FSMs, one to control each block of memory. The Minor FSM has three states: an idle state, a read state, and a write state. During a read, it obtains data from the BRAM Interface sequentially from each address in the corresponding memory. During a write, it tells the BRAM Interface to write sequentially to all addresses in the corresponding memory block.

The state transition diagram for the Major FSM is shown in Figure 3.2 below. It shows the sequence of reads and writes for the ping-pong memory implementation. The outputs Read0, Write0, Read1, and Write1 are control signals for the two Minor FSMs. Due to space constraints, the state transition diagram for the Minor FSM is omitted.

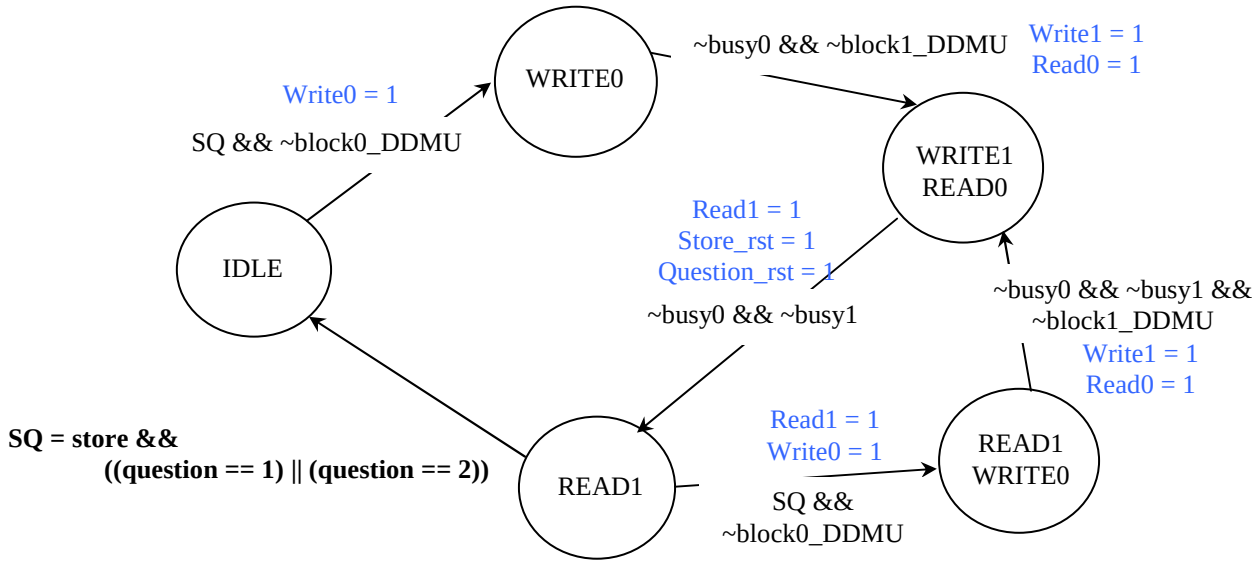


Figure 3.2 – State Transition Diagram for Major FSM in Memory Module

3.2.3 – The Digital Decision-Making Unit

One of the key elements of a lie detector test lies in the types of questions asked. Questions are grouped into 3 categories. The first type is called a *control* question. They are simple, easily-verifiable questions that are interspersed within the test. The second type is called a *relevant* question. These are personal, often embarrassing questions that are designed to provoke a physiological response. The last type is called a *relevant* question. These are questions which are pertinent to the actual examination. The T/F decision is based on the assumption that a subject with nothing to hide will react more strongly to a control question, whereas a subject who is lying will react more strongly to a relevant question. This is the theory used when implementing the Digital Decision-Making Unit.

The Digital Decision-Making Unit is the largest and most complicated part of the DCU. It consists of a Major FSM, a Data Register, and four Minor FSMs (used to collect, process and store data as well as make a binary T/F decision).

The Major FSM – The Major FSM again acts as the controlling module for data acquisition and processing. It interacts with the Memory Module using handshake signals (to determine which unit is reading or writing from each block). It also sends out start signal to each of the four Minor FSM's to indicate that they should begin their computations. The state transition diagram is shown below. It shows the sequence of data acquisition, processing, and storage/decision. Again, due to space constraints, the state transition diagrams for the Minor FSMs are omitted.

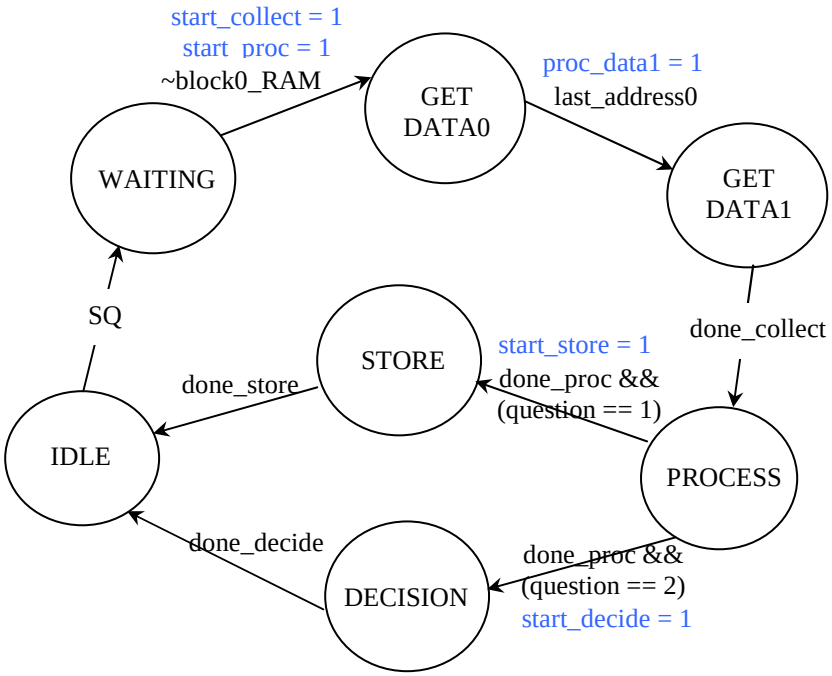


Figure 3.3 – State Transition Diagram for Major FSM in DDMU

The Minor FSM Collect – The MinorFSM_collect is responsible for obtaining the raw sensor data from the Memory Module. Since the data is a concatenation of the skin conductivity and pulse data, this module separates the data and lowpass filters the skin conductivity data to remove high-frequency noise. Note: the pulse data was not filtered since it might destroy the heartbeat information (which typically occurs as short, high-frequency pulses)

The FIR Lowpass Filter – A 42-tap lowpass filter was created to process the skin conductivity data. It consists of 42 registers (to hold each of 42 delayed samples needed to compute the output). When activated, the filter shifts all the register values by one and stores the incoming value in the first register. It computes the output by multiplying all the data registers by the appropriate coefficient and summing the products. To avoid overflow problems, the number of bits on the output was adjusted to account for the maximum and minimum output values. The primary advantage for building this module rather than using the Coregen FIR module is that all the registers are cleared immediately upon reset, whereas the Coregen module requires a long latency to clear the data registers.

The Minor FSM Process – This module computes the various statistics on the input data sequences. It uses 5 sub-modules, one to calculate each set of statistics, registers the computed output, and resets the sub-modules when the computation is complete. The sub-modules are described in more detail below.

The FFT Module – This module computes a 2048-point FFT of the pulse data and determines both the index of the maximum frequency content (in terms of the transform magnitude) as well as the maximum value at that index. It uses the Coregen FFT module to compute the transform. Since the Coregen module requires that the data appear at the input exactly 3 clock cycles after the index,in order to meet the timing requirements, the pulse data is first collected and stored in an internal BRAM. The BRAM is then accessed at the appropriate times to satisfy the FFT timing. The Coregen module outputs the transform

values and corresponding index in a stream. Therefore, the FFT Module searches for the maximum value (other from DC) and returns the index at this maximum as well as the maximum output.

Other Processing Modules – The other processing modules follow the same pattern. When activated, they search through the incoming data stream and keep updating an internal register which holds the desired properties. This continues until the MinorFSM_collect signals the end of the data stream. The modules are as follows:

- **MinMax** – Searches for the Minimum and Maximum skin conductivity value
- **MaxDeriv** – Searches for the Maximum First Difference in the skin conductivity data (in terms of magnitude)
- **SumEnergy_cond** – Returns the Sum and Energy (sum of absolute values) of the skin conductivity data
- **SumEnergy_pulse** – Returns the Sum (also the Energy) of the pulse data
- **ThreshCount** – Returns the number of heart beats in the sampling intervals

The Minor FSM Store – The MinorFSM_store is used to store the computed statistics for control questions in the Data Register (Coregen BRAM). It concatenates all the statistics into a large 254-bit vector and stores it in one row of memory. In addition, it will increment the memory address after every store operation so that the data is not overwritten.

The Minor FSM Decide – The MinorFSM_decide is called when analyzing a relevant question. The current algorithm simply accesses the last row of computed statistics in the Data Register and compares them to the computed statistics for the current question. It makes a binary T/F decision based on how the two sets of values compare.

3.3 – Testing and Debugging

Since the group was given only 2 lab kits, most of the debugging was done using ModelSim post-place and route simulations. During the first testing phase, each module was simulated individually to ensure basic functionality. During the second testing phase, the User Capture block, the Memory Module and the DDMU were each put together and simulated. No major problems were encountered during this phase. Most of the modifications were very minor, and were usually the result of some small oversight.

The third testing phase did not begin until the lab kit access was available. The modules were hooked together and the code downloaded onto the FPGA. The Logic Analyzer was used to view internal signals. Despite previous simulations, there were several problems, the biggest being timing issues with the hand-shake signals. It turned out that since some of the handshake signals were one-cycle pulses, the receiving module was not able to recognize and react to them. This debugging phase took nearly two days of full-time lab work. The problem was finally solved by using ModelSim to simulate the entire DCU and by viewing the necessary signals.

In hindsight, more time should have spent simulating the entire DCU, because there were a number of unforeseen problems when the User Capture, Memory Module and DDMU were connected together. Many of them were hard to debug on the FPGA and could have been avoided by proper simulation. In addition, having access to the FPGA sooner would have greatly eased the stress during the last couple of days of the project because the testing could have been completed earlier rather than pushing it back to the final two days.

4.0 – The Video Display

The video display consisted of one main module which contained several smaller modules, rather than just having several peer level modules. The system takes inputs from the main labkit (such as the clock and debounced signals from user inputs) and outputs the necessary video control signals (such as h_sync or v_sync).

A block diagram of the system is shown in figure 4.1 below. A larger version is found in the appendix.

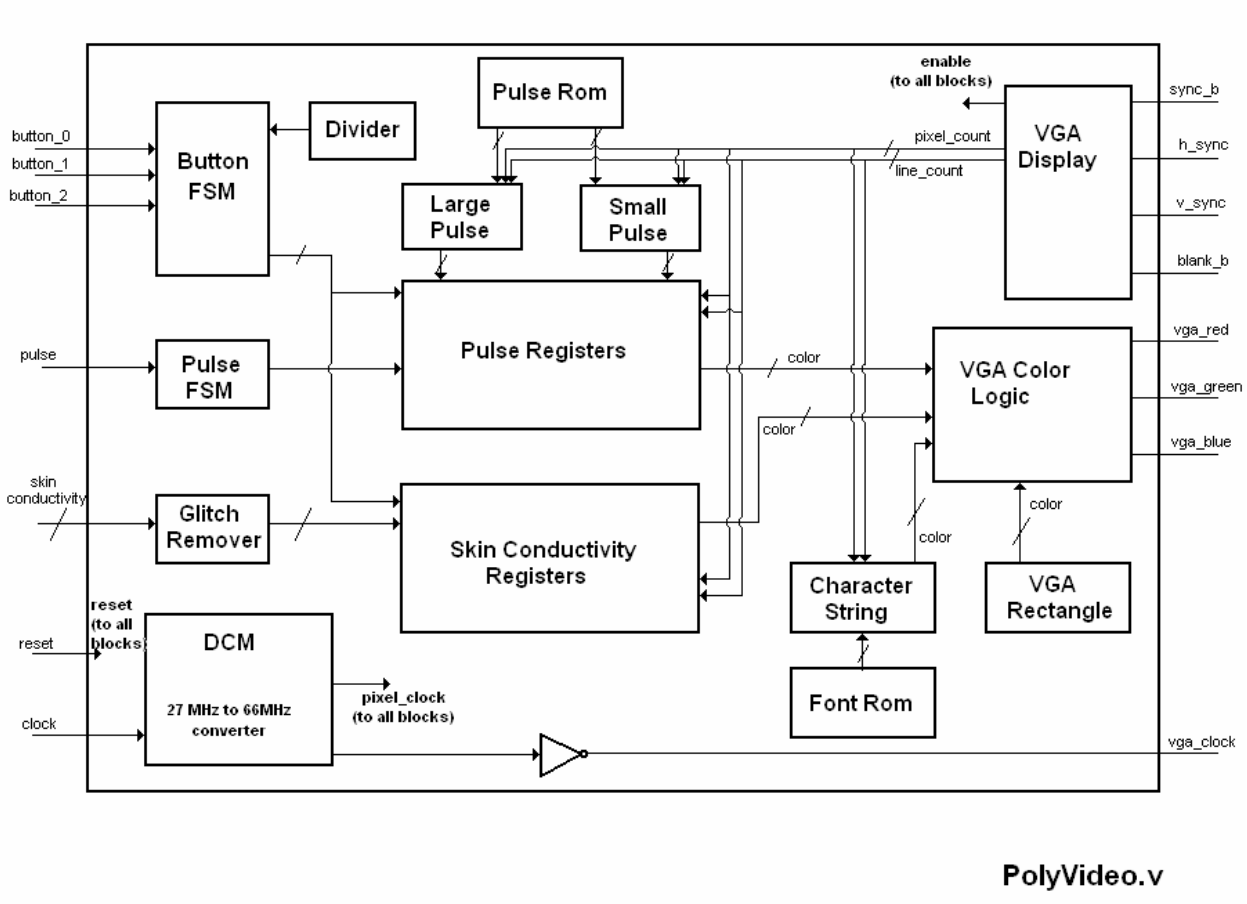


Figure 4.1 – Block Diagram of the Video Display

The inputs were synchronized via either an FSM or (in the case of the skin conductivity) a glitch remover. The outputs were sent directly to the labkit module, to the appropriate VGA ports. The individual blocks will be discussed in further detail below. As most of the modules dealt with data being displayed on the screen, all testing (except for that of the FSMs) was done by synthesizing the code and providing simulated inputs to the FPGA.

DCM

The most important element in a synchronized digital system is the clock. The DCM module takes in the 27 Mhz clock signal and outputs the 66Mhz clock used by the monitor. This faster clock is also used by all the modules in the video display. There are two other signals which are used by (or at least sent to) every module in the system except for the ROMs: those are the **enable** signal, which goes

high when v_sync transitions (that is, when v_sync goes from high to low, thus ensuring that changes to the data happen offscreen) and **reset**, which is an input to the system and globally resets all the modules.

The output from the DCM is also inverted and sent to the VGA system as the pixel clock. The 180° lag between the clocks ensres that the outputs from the sync and color signals will be glitch free by the time the monitor registers them.

Divider

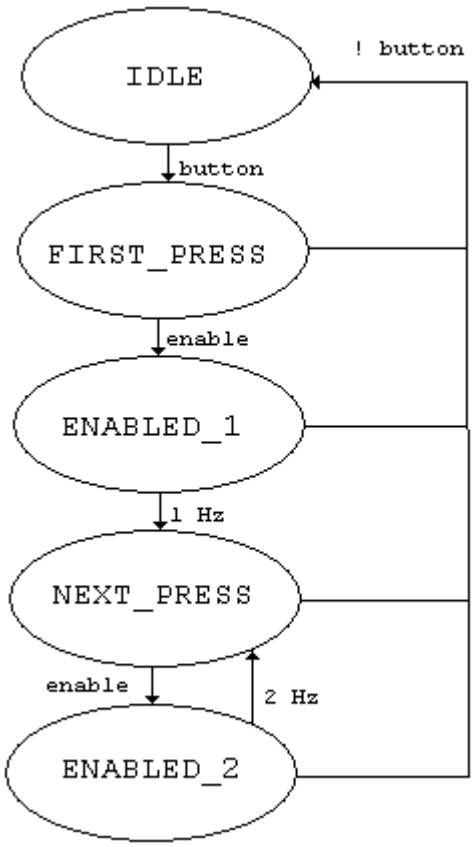
The divider was a simple module which created a one Hertz enable signal and a two hertz enable signal. It was used by the button FSMs for timing purposes.

Button FSM

The button FSMs were simple FSMs that served two purposes: first, they synchronized the user inputs to the system, and second, they provided a mechanism to slow down user inputs to a speed that was useful.

The inputs to the button FSMs came from debounced pushbuttons on the FPGA labkits. When the input was low, the system was in the IDLE state. Also, any time the inputs when low, the system would, regardless of its state, return to the IDLE state.

When in the IDLE state, if the input went high, the FSM went to the FIRST_PRESS state, and the output was high. It would remain in this state until the **enable** signal went high, at which point it went into the ENABLED_1 state. during this state, the output was low. Once the 1 Hz timer went high, the state changed to NEXT_PRESS. In NEXT_PRESS, the output was again high until the **enable** signal was sent again. After this, the state went to the ENABLED_2 state, and would cycle back to the NEXT_PRESS state after every 2 Hz signal. This meant that when the button was held down, the button would seem to be pressed every half-second for the purposes of the rest of the modules, ensuring that holding the button down would not send a repeat faster than a human could react to.

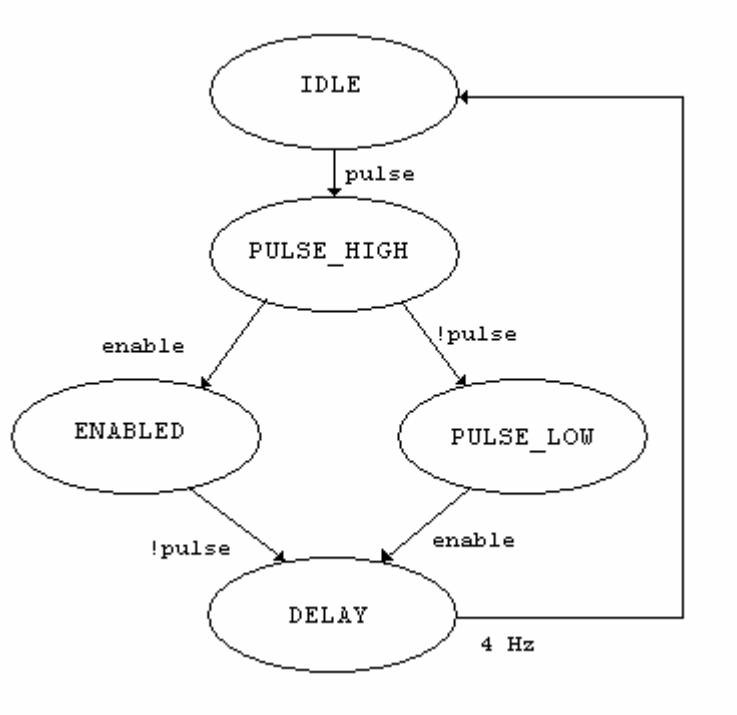


Glitch Remover

This module existed to try to clean up noise when registering the input to the skin conductivity module. As the change in skin conductivity could not be guaranteed to remain the same on the rising edge of the clock, this module ensured that the signal would be clean. It sampled the value on the rising edge of the clock, and compared that to the previous two values. If any two of those matched, then it would output that value; if all three were different then it would simply hold the last value it had.

This took advantage of the fact that first, actual skin conductivity changes have a frequency of less than one-half of a Hertz, and also the fact that the sampling rate of 66 MHz was more than twice that of the input clock (27 MHz).

Pulse FSM



The pulse FSM acted slightly differently than the button FSM. In this case, I didn't know how many cycles the pulse would go high for – whether it would be for one or many. In addition, I needed to sync it to the **enable** signal, so I decided to do that through an FSM.

When the pulse when high, the FSM moved from its IDLE state to the PULSE_HIGH state. It would hold that state (and have a high output) until one of two things happened: either the **enable** signal would go high (in which case it would go to the ENABLED_1 state, and the output would go low), or the input would go low (in which case it went into the PULSE_LOW_1 state, and the output remained high). After this, when whichever the other one occurred (either enable when high or the input went low) the system waited for a 4 Hz enable and then reset to the IDLE state (this was to block out noise; if a person's pulse was over 4 Hz, they would not be in a situation to answer questions anyway).

Large Pulse and **Small Pulse**

These modules create the pulse waveforms. The large pulse consists of a series of counters, which are set when the pulse is high and decrement every **enable**. The smaller one set the values of the pulse edges from serial data sent in when the save button is pressed. They get their color data by accessing the Pulse ROM, which stores the waveform map for both pulses.

When the pixel_count or line_count changes, the current pixel is compared with the location of the nearest edge. If it's within the range of where the waveform should be, the difference between the current pixel and the nearest edge is sent to the ROM, and the value of that pixel is returned.

Pulse Registers

This module takes in the pulse from the pulse FSM every **enable**. It stores the data in a 340 bit vector, which it exports serially to a small pulse module when a save button is pressed. It also sends a start signal to the large pulse module when both **enable** and pulse are high.

Skin Conductivity Registers

This was the largest module, both area-wise and code-wise, and as such was the hardest to test and debug, and also was the most important to prevent glitches for the system. It consisted primarily of four banks of 340 registers, one of which was seven bits wide, and the other three were six bits wide.

When the new conductivity data was read in, the data in the largest register bank was shifted one place (that is, data in register [i] went into register [i + 1]) and the seven most significant bits of new data were read into the first register. When a save button was pressed, the six most significant bits of the data were saved into the corresponding register in the appropriate small bank (that is, small [i][5:0] <= large [i][6:1]).

To present this information on the screen, the data had to be pipelined several steps. First, the index of the appropriate register to check had to be selected. To do this, for the small register, the difference between the edge of the area it occupied on screen and the current pixel was subtracted from 340; for the large area, the 1020 pixels across were divided into 340 three-pixel blocks, and the block number was subtracted from 340. The inversion (340 to 0) occurred because while the screen's x increases from left to right, the data was meant to scroll from right to left.

After the register was determined, the appropriate register was selected to produce its data. This value (0-127 for the large register, 0-63 for the small one) was added to the height of the current line. If this value was higher than the bottom of the skin conductivity screen area, then the pixel was considered "on"; if not, then it was "off".

Character String

For the character string, I just downloaded the verilog files available on the web. Given a vector of bits that is n*8 bits long (where n is the number of characters) and a location, it creates a string of ASCII characters at that location on the screen.

VGA Rectangle

The VGA rectangle module was just the rectangle module we used in lab 4. Originally, the screen consisted of several background rectangles with the data overlaid on top of them, but as I implemented more and more of the modules, I used the modules themselves to create the "empty" background display. In the end, the only rectangle on the screen was the one for the "true or false" data from the DDMU. The rectangle changed color (green for true, red for false) to indicate whether or not the person was lying.

VGA Display

The VGA Display module was basically a glorified counter. It took in the clock signal and produced a two-dimensional output – pixel_count and line_count. These values reflected the current location being drawn on the screen.

There were also several bits that were output from this module for synchronization purposes; they let the monitor know where each pixel was (based on the values of h_sync, v_sync, and blank_bar). This properly oriented each pixel on screen. It also produced an **enable** signal on the falling edge of v_sync – this let all the registers know when it was time to update.

The data from this module was directly output to the ports which connected to the video display in the labkit.

VGA Color Logic

This was a very simple module, which started out as a very complicated one. Initially, there were two modules; one for the foreground and one for the background. However, as the background rectangles disappeared, eventually there was only need for one module. It simply did a bitwise OR on the different color outputs, and sent the result to the color output ports.

Pulse Rom and **Font Rom**

These ROMs stored the pixel data for the pulse waveforms and the VGA ASCII characters, repectively.

Implementing, Testing, and Debugging

This display, all told, took up about 30% of the labkit. Most of which, as I mentioned before, was in storing the skin conductivity data. In order to optimize this design, I did several things. First, all data that could be compressed into a single module was. I created a ROM for the pulse information, and sent data between modules serially to minimize the number of wires needed. Also, I used only 6 bits to convey color information rather than 24. This vastly cut down on the amount of wiring I needed.

Since most of the data was changing at slower than 2 Hertz, and since the VGA clock was running at over 60 MHz, checking the performance of the large pieces of this module in simulation was not feasible. I tested all of the FSMs in simulation, and made sure that the values in the counters (both in the pulse module and the divider) worked properly, but most of the modules had to be optimized by synthesizing the data to the FPGA. I used the logic analyzer to ensure that the proper signals were being output, and the LED display on the labkit as well.

All in all, the video module was a fun one to implement, and also rather instructive. Nothing teaches you to remember that while verilog looks like a programming language, it actually corresponds to physical components like spending hours trying to fix your program only to find that you didn't have enough wires to route your signal.

In the end, hardware optimization is not just about speed, power, or area, but also about remembering that there is a finite number of interconnects available, and wiring up a VGA display conveys that information well.

5.0 – Connecting and Debugging the Entire System

Once the individual modules had been implemented and tested, the system components were connected. Due to the modularity in partitioning the design, this phase went quite smoothly. The interfacing took place in two stages. In the first stage, the ADC and memory interface were connected to the Digital Control Unit in one of the lab kits. In the second stage, the output display (on the second lab kit) was added into the system using jumper wires from the user I/O ports.

The most complex interfacing in this project was the one between the ADC and memory interface portion and the DCU. Although most of the read and write operations to and from memory functioned properly, there were a few problems to debug such as the memory address not being reset properly, and control signals not being asserted. The debugging for this stage of the project consisted primarily of looking at signals on the logic analyzer and using the alpha-numeric display (explained in section 2).

The second stage of interfacing simply involved running wires to the output display. The output display was able to receive dynamic data and the binary T/F decision from the beginning, so there was hardly any debugging.

Although the digital portion of the overall system functioned correctly, the sensor data was too poor to calibrate and test the system. The pulse waveform in particular was too noisy to give a solid enough heartbeat that could be analyzed, and the skin conductivity waveform hardly changed from one operating condition to another.

Overall, the project was a mixed success. Although the system functioned as intended, it was not possible to implement the complete lie detector because the sensors could not provide reliable data. In hindsight, more care should have been given to the physiological sensor implementation. For example, it may have been better to use a piezoelectric pressure sensor rather than an electrical signal for the pulse. Another course of action may have been to purchase medical sensors from an appropriate company rather than try to construct them from scratch.

Regardless of the overall result, the project was educational in its own right. It provided a lot of insight into different design architectures and different implementations. It also provided the opportunity to expand one's skills and to think on one's feet, since the system had never before been implemented.