Massachusetts Institute of Technology

Department of Electrical Engineering and Computer Science

6.111 - Introductory Digital Systems Laboratory

Digital AM Receiver

Hassen Abdu, Ebad Ahmed, and Wajahat Khan

May 10, 2005

TA: Hyunjoo Jenny Lee

Abstract

AM radio reception is conventionally done by using analog circuitry. However, this paper describes the design and implementation of a digital AM receiver. The receiver digitizes the entire AM band, executes digital signal processing, and extracts the radio channel that the user desires. The receiver is a combination of digital signal processing and a user interface designed to achieve a powerful, user-friendly AM receiver.

Chapter 1

Introduction

Implementing an AM radio using analog electronics has always been the norm. However, with digital systems improving and becoming more powerful, it is becoming easier to do some things that are usually done in analog circuitry in digital circuitry. Therefore, the Digital AM Receiver is a digital system that attempts to achieve the same analog AM radio functionality by just using an FPGA and a small of amount of analog electronics to interface with the real world.

The motivation for this project came from the work done on software radio by companies like Vanu. Software radio allows a single device to receive many different wireless transmissions. By using digital signal processing techniques in the FPGA, the software radio could possibly be achieved in digital systems. However, since building an AM radio is quite easy to learn, it was sensible to focus on AM radio transmission instead of FM and other more intricate wireless transmission. This project is worthwhile also because it develops digital design techniques that can be applicable to more advanced communication systems. For example, this project could be expanded to receive FM and other wireless transmissions if the necessary modifications are made on the fundamental blocks.

1.1 Overview

The Digital AM Receiver will be implemented in four main parts: the Analog Front End, the Digital Signal Processing unit, the User Interface, and the Display. The Analog Front End will receive the entire AM frequency band, which ranges from 530kHz to 1700kHz. The Front End passes the desired frequency band and digitizes it with an ADC. Next, the digital signal processing module extracts a particular radio channel and demodulates it to retrieve the audio signal. The User Interface module enables the user to customize the system to his or her preferences. For example, the user can program presets for favorite radio stations, pause and replay live broadcast, or simply scan through the AM radio transmissions. Finally, the display provides the user with real time information

about the system (i.e. Paused, Replaying, Scanning, etc.), instructions, options that have been selected by the user, and signal waveforms. In Figure 1.1, the entire Digital AM Receiver is shown with its major components: Digital Signal Processing module, Video module, and User Interface module. Each of the modules will be discussed in greater detail below.

Figure 1.1: Detailed System Block Diagram

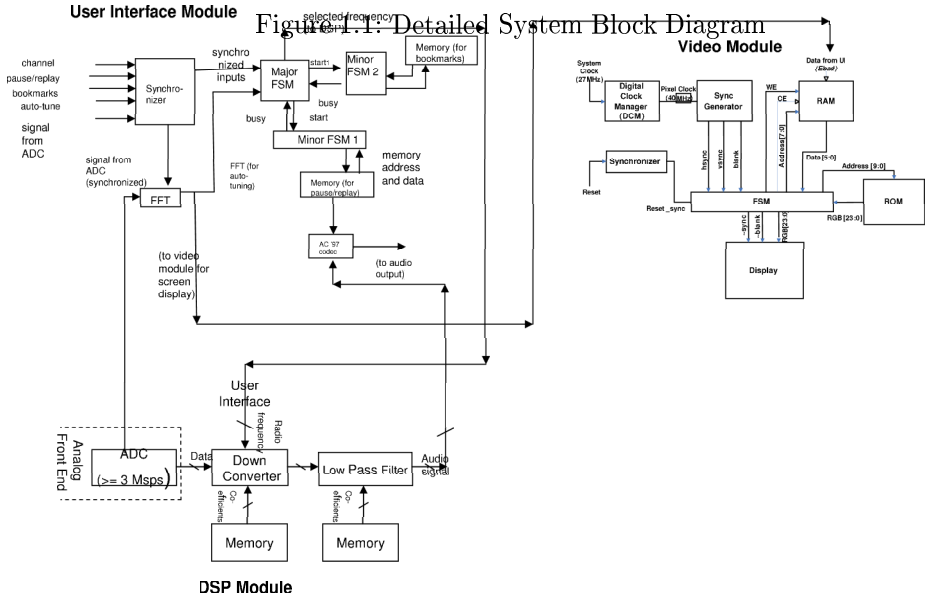

Figure 1. Project Block Diagram

Chapter 2

Architecture

2.1 Analog Front End

The analog front end, the unit that receives the entire AM band and digitizes it, was designed and built by Joe Sousa at Linear Technology. By using the ADC chip he designed, the LTC1403A, the system was able to achieve 14-bit digital resolution at about 3 megasamples per second (MSPS). However, the ADC produces a serial output, so a serial-to-parallel converter was used. The converter was comprised of 19 registers. The reason is that the complete conversion to 14-bits requires 18 cycles. And after every cycle, the 14-bits propagate through the 18 registers. After conversion is done, the 19th register takes in 14 of the 18 register outputs as inputs. The output of the 19th register is then the parallel data from one conversion cycle. After the 19th register has latched its input, a ready signal is sent to the DSP module to indicate that a new 14-bit sample is ready.

2.2 Digital Signal Processing

Many approaches can be adopted to demodulate AM frequency. The one that was chosen in this project consists of two important stages:

1. Down Conversion: Multiplication of signal in time domain by sinusoidal wave of the appropriate frequency of the radio channel to bring the signal to the base band.

2. Low-Pass Filtering: Convolution in time domain of the signal with an impulse response that attenuates frequencies higher than 5 kHz. The low-pass filter removes all the signals at higher frequencies, selecting only the demodulated signal at the selected frequency pass through.

This modulating scheme was first tested in Matlab. The Matlab code for the simulations is included in the Appendix. Recorded Audio signal was first modulated at different frequencies and then all the signals were added to model a radio spectrum in air. The original audio signals were successfully recovered

when the demodulation technique mentioned above was used to demodulate the signal at the appropriate frequencies.

Matlab was used to generate the coefficients for the sinusoidal wave and an Finite Impulse Response (FIR) filter of 30-taps. For a start, 1030 kHz was chosen as the radio frequency to demodulate which corresponds to WBZ news radio, a Boston based radio station. Another frequency that can be demodulated is 850 WEEI, a Boston based sports radio channel. Since the coefficients of the sinusoidal wave are stored in a ROM, more radio frequencies can be added just by storing corresponding sinusoid coefficients in the ROM. The sinusoids were generated using Matlab.

The sampling frequency of the system had to meet Nyquist criterion. Since the AM radio band ranges upto 1700 kHz, we need to sample around 3.5 Msps or higher. During the initial stages of the project, an ADC of a speed of 3 Msps was chosen because of its special noise filtering feature. This ADC, thus, limits our resolvable frequency band to 1500 kHz at the upper limit.

The ADC clock runs at 55 MHz. In order to be in sync with the conversion of ADC, the system clock also runs at 55 MHz. There is one output sample computed for every input sample. So there are only 18 clock cycles for a complete cycle of the system. Keeping in mind the propagation delays associated with arithmetic operations such as multiply and addition, and given the high clock rate that the system is running , one of the biggest challenges in making the system was to drive the Digital Signal Processing at this speed.

Parallel processing is the solution when it comes to driving systems at higher speeds. The DSP module carries the same theme in its design. Apart from the register file, ROM and the output latch, every other digital logic in the system is combinational. Down coversion only takes about a cycle in terms of propagation delay, since it is just a single multiplication. The hard part is to do the convolution with the 30-tap FIR. Rather than adopting a serial approach of using an accumulator which sums up results of convolution multiplication operations, the system is implemented by computing every multiplication required in parallel, using 30 2-input, 18-bit multipliers, and then adding all the results in parallel as well through a 30-input 32-bit adder. Fig. 2.1 on next page shows the system block diagram of the DSP Module.

Figure 2.1: DSP Module

The DSP module takes in data_inready signal from the Analog Front End in addition to the 14-bit data stream. The DSP FSM triggers on the a 4 cycle registered-delay signal of data_inready. The signal is delayed just to make sure that the propagation dealy of the down coversion multiplier has been met and the data is ready to be stored in the 30 tap delay register line. Next, the down-converted sample is stored in the register line. The most recent downcoverted sample gets stored at the beginning of the delay line while every other data stored in the register delay line gets shifted down the line by one. The oldest data is discarded while the 2nd to oldest data takes its spot. Every logic used after this point in system is combinational. The thirty data points are hardwired to a 30 input constant multiplier. The coefficients of the FIR are hard coded into the multiplier and they multiply the data through the delay line from the register file according to the position of the coreresponding downconverted data in the delay line. The multiplier sends all the computed 30 muliplication results to the 30 input adder, which sends out the computed result after some propagation delay. The propagation delay for the whole convolution process is several cycles which we conveniently meet in context of the upper limit of 18 clock cycles per sample on one cycle of computation. The data is latched towards the end of the available number of cycles for interfacing with the audio controller part of the project. The system then goes into the a state where it looks at the data_inready signal to start another cycle of computation going through the same steps outlined in this paragraph again and again.

2.3 User Interface

The user interface allows users to interact with the core of the system. Users can customize the system in any way they like. Features available to the user

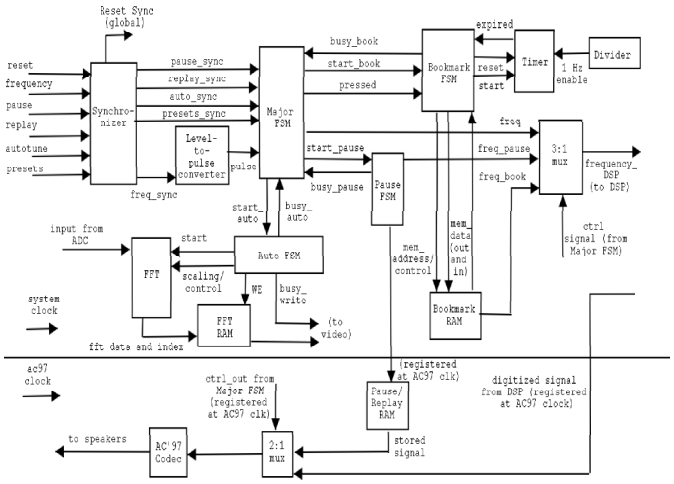

Figure 2.2: User Interface Block Diagram

include selecting and playing different radio stations, pausing and replaying a live transmission, book-marking various stations (similar to the preset system in an automobile), and auto-tuning to different stations. Users interact with the system through switches. A detailed block diagram for the user interface is shown in Fig 2.2.

The user interface is essentially divided into two parts that run on different clocks (the 27 MHz system clock and the AC97 codec clock we will talk about the AC 97 codec in more detail). Any control signals that go from the main system to the AC97 system are registered at the AC97 clock (for instance, the digitized signal from the DSP module that needs to be output to the speakers), while control signals going in the other direction are registered at the system clock.

In the main system, all switch inputs pass through the Synchronizer, which consists of a pair of registers. All synchronized inputs go directly to the Major FSM. Additionally, the channel/frequency select input passes through a level-to-pulse converter that sends out a pulse to the Major FSM whenever the user selects a different station.

2.3.1 The Major FSM

The Major FSM receives all inputs and coordinates the minor FSMs based on those inputs. The implementation of the Major FSM is summarized by the state transition diagram of Figure 4.4. The FSM has five states; namely IDLE, CHANNEL, PB, START_AUTO and WAIT_AUTO. The FSM has as inputs and outputs handshaking signals to and from the minor FSMs. In addition, it also controls the multiplexors at the output to the DSP module and to the

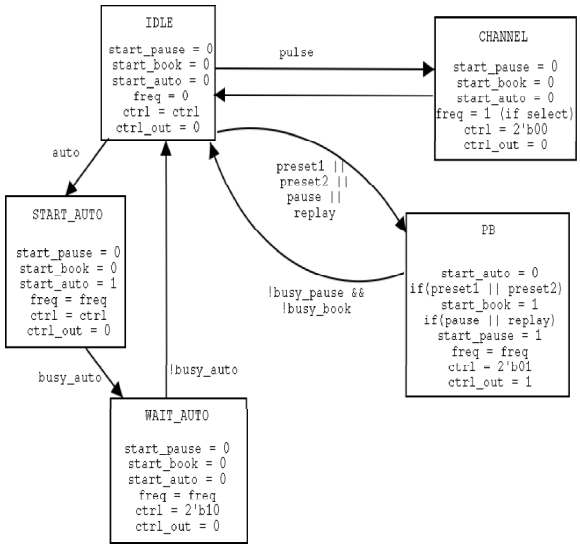

Figure 2.3: Major FSM

speakers by sending out control signals for those muxes.

The Major FSM is in the IDLE state by default. It transitions to different states depending on user inputs. When the user changes channels, the level-to-pulse detector outputs a pulse that makes the pulse input to the FSM go high. The FSM then transitions to the CHANNEL state. It sets the control of the DSP output multiplexor so that it can output its own frequency value to the DSP module, and sets the output frequency bit based on the position of the channel selection switch. The FSM then transitions back to IDLE.

If the user wants to pause/replay a live transmission, or to bookmark a particular channel, the FSM transitions to the PB state. It sends a start signal to either the Pause Minor FSM or the Bookmark Minor FSM depending on whether the user requested pause/replay or bookmark. It sets the control signal (ctrl) to the DSP output mux so that the frequency can be directly output by the Pause Minor FSM or the BookMark minor FSM, and also sets the control signal to the speaker output mux (ctrl_out) so that the signal can be output directly from memory rather than the DSP module in case the user requested a replay.

In case the user requests auto-tuning, the FSM transitions to START_AUTO, and sends a start signal to the Auto-tuning Minor FSM. It waits in this state until the Minor FSM asserts the busy signal, and then transitions to WAIT_AUTO, where it sets the control bit for the DSP mux so that the DSP frequency can be output directly from the Auto-tuning FSM, and waits for the minor FSM to finish operation.

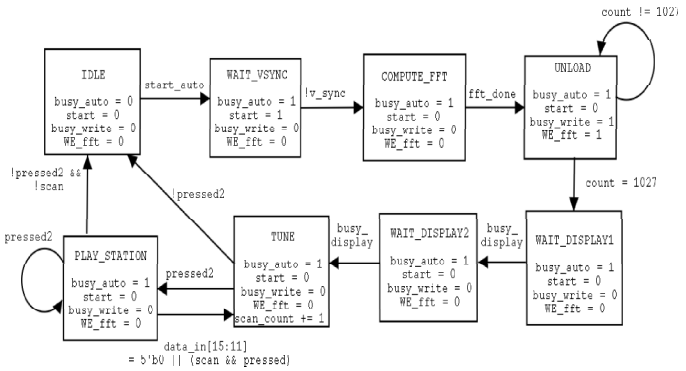

Figure 2.4: Auto FSM

2.3.2 The Auto FSM

The Auto FSM is responsible for auto-tuning to different radio stations. It does so by computing the FFT of the input signal from the ADC and identifying peaks and the corresponding frequencies. The FSM therefore also controls the FFT module. The implementation of the Auto FSM is summarized by the state transition diagram of Figure 2.4. The FSM has eight states; namely IDLE, WAIT_VSYNC, COMPUTE_FFT, UNLOAD, WAIT_DISPLAY1, WAIT_DISPLAY2, TUNE and PLAY_STATION. The FSM has as inputs and outputs handshaking signals to and from the Major FSM and control signals to the FFT and the corresponding RAM.

The FSM is initially in the IDLE state. When the user requests auto-tuning, the Major FSM sends the start_auto signal to the Auto FSM, which then transitions to WAIT_VSYNC. The FSM waits in this state until the v_sync signal goes low, and then transitions to COMPUTE_FFT. The FSM sets the control signals for the FFT module and sends it a start signal to indicate that FFT computation is ready to begin. It waits in this state until the fft_done signal goes high, indicating that the FFT computation has ended and data is ready to be unloaded. When it receives the fft_done signal, it moves to the UNLOAD state, where it routes the data from the FFT to the FFT RAM and sets busy_write high to indicate to the Video module that data is being written to memory. The data is in real and imaginary format, and must be squared and added to compute the magnitude before it is stored. This is accomplished through multipliers and adders. Since the resolution of the FFT is 1024, the FSM waits in this state for approximately 1024 clock cycles in order to ensure that all data has been unloaded from the FFT module and stored in the RAM.

The FSM then moves to WAIT_DISPLAY1 and WAIT_DISPLAY2, where it waits for the Video module to read the data from the RAM and display it on the screen. When the display is done, the FSM moves to the TUNE state in order to begin the auto-tuning. The FSM selects a RAM address and moves to

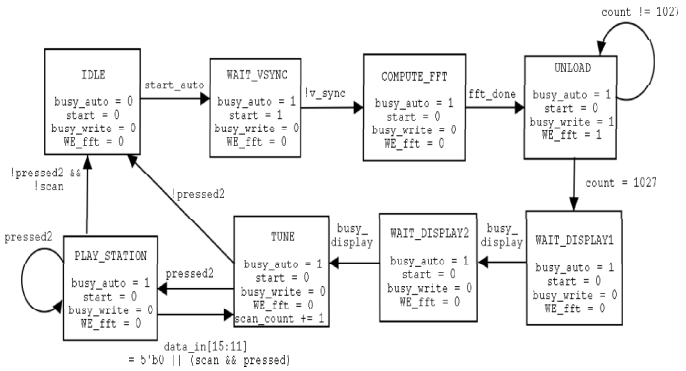

Figure 2.5: Bookmark FSM

PLAY_STATION, where it compares the data value from that RAM address to a threshold of 210, by checking to see if the first five bits of the data value are zero. If it is below the threshold, the FSM transitions back to TUNE and looks at the next RAM address. If the data value is actually greater than the threshold, the FSM outputs that frequency to the DSP module so that the station can be played. The FSM will transition back and forth between TUNE and PLAY_STATION until the user de-asserts the auto-tuning input.

2.3.3 The Bookmark FSM

The Bookmark FSM is responsible for presetting different switches for playing certain stations. The implementation of the Book FSM is shown by the state transition diagram of Figure 2.5. The FSM has four states; namely IDLE, READ, WAIT and WRITE. The FSM takes as inputs the signals Reset_Sync, start_book (from the Major FSM), expired (from the timer), data_in (from the Bookmark RAM, which implements a table lookup memory for storing data about preset stations), and other signals from the Major FSM that carry information about precisely what button has been pressed, and outputs the signals busy_book(to the Major FSM), control signals to the timer, and control signals to the Book RAM.

The FSM is initially in the IDLE state. When the user requests bookmarking or playing a previously bookmarked channel, the Major FSM sends a start_book signal to the Bookmark FSM, and also tells it whether or not the user has been holding the reset button down. The FSM then transitions to the READ state, and does a read operation on the Book RAM in order to find out if that button has been preset before. If the button has been previously preset, the BookRAM outputs the corresponding station frequency to the DSP module and moves back to IDLE. If the button has not been preset, the FSM moves to WAIT and waits for the user to hold the button for a period of 5 seconds. It measures 5 seconds by making use of the expired signal from the timer.

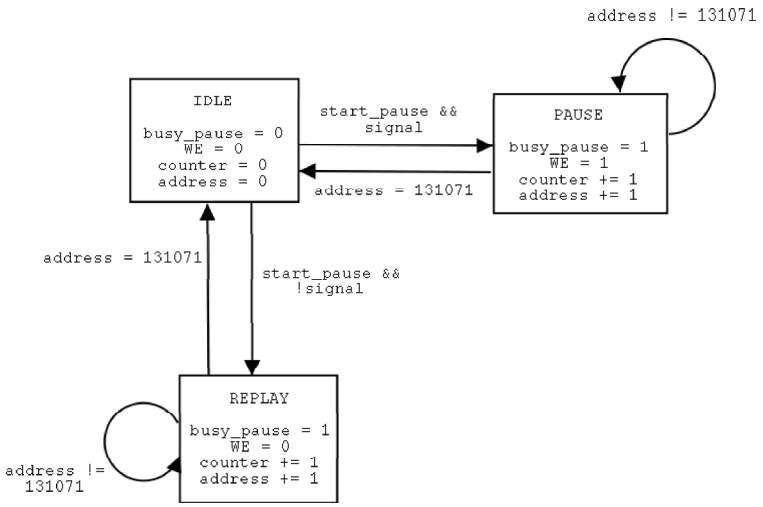

Figure 2.6: Pause FSM

If the user releases the button before 5 seconds have elapsed, the FSM simply transitions back to IDLE. If the user does hold the button for 5 seconds or more, the FSM transitions to the WRITE state, and stores the button (represented as a binary string) and the corresponding frequency value in the Book RAM by doing a write operation. The FSM eventually transitions back to IDLE.

2.3.4 The Pause FSM

The Pause FSM is responsible for pausing and replaying a live transmission. It does so by storing the digitized signal from the DSP module into memory, and replaying the stored signal from memory. The implementation of the Book FSM is shown by the state transition diagram of Figure 2.6. The FSM has three states; namely IDLE, PAUSE, and REPLAY. The FSM takes as inputs the signals Reset_Sync, and start_pause and signal (from the Major FSM), and outputs the signals busy_pause(to the Major FSM), and control signals to the Book RAM.

If the user requests a pause in the current transmission, the Major FSM sends it the start_pause signal and tells it whether the request is for pausing or replaying. The FSM then transitions to PAUSE, and writes the signal coming from the DSP module to the Pause RAM. It then transitions back to IDLE. If the user requests a replay, the FSM moves to the REPLAY state, and routes the data stored in the Pause RAM to the output speakers through the speaker output mux.

2.3.5 The AC97 Codec

The other part of the system consists of the AC97 codec, which runs on the AC97 clock. The AC97 codec downsamples the digitized signal from the DSP module to 48 kHz so that it is in the audible range for humans, and can be heard on the speakers.

2.4 VGA Display

The VGA display, as shown in Figure 2.7, provides a means for the user to interface with the Digital AM receiver by providing the user with a user menu, Fourier transform of the AM frequency band, real-time information about the status of the system, and the name and station that the user is listening to. With this graphical interface, the user can take full advantage of the features of the Digital AM receiver.

The VGA output is timed by a 27 mHz clock which is converted to a 50 MHz pixel clock for 800x600 VGA at 72 Hz. The VGA control block, by using the pixel clock and the parameters for screen resolution and refresh rate, outputs control signals are sent to the DAC in the labkit.

The VGA control block also communicates with the RAM control and ROM control in order to access the RAM and ROM, respectively, to output the display correctly. The RAM will store the dynamic information, the Fourier Transform magnitudes, that will change during the usage of the system. However, the RAM will only be written by the FFT module and will be read only for displaying the FFT. The ROM will store the static information, the character 8x12 bitmap representation, that will be initialized but will be read throughout system use.

The VGA Control is designed to make decisions on what to output to the display based on a grid with the number of character horizontally and vertically. If the VGA Control was at pixel point (16,24), this point represents character (2,2). The VGA control runs through all of the character coordinates and decides what character to output by either looking up a RAM, for dynamic text, or immediately looks up the ROM for a hard-coded character. Actually, implementing the control signals for accessing the RAM and ROM was only based on counters, and no FSMs were used. This is a design choice discussed below.

The RAM interpreter module is used to read the magnitude values of the Fourier Transform that are stored in the RAM and decide on the height of the character points to display the waveform. For example, if the FFT magnitue at frequency 700kHz had a magnitude greater that 2, the RAM interpreter would plot in bar graph form that FFT magnitude. Also, the data is written to the RAM in such a way that the FFT magnitude corresponding to a certain frequency is stored in the same location in RAM for every FFT process. The reason is that as the VGA control module sweeps through the horizontal dimension on the display, which has been plotted to represent frequencies in the AM band, the RAM Control module should only have to look at the same address location

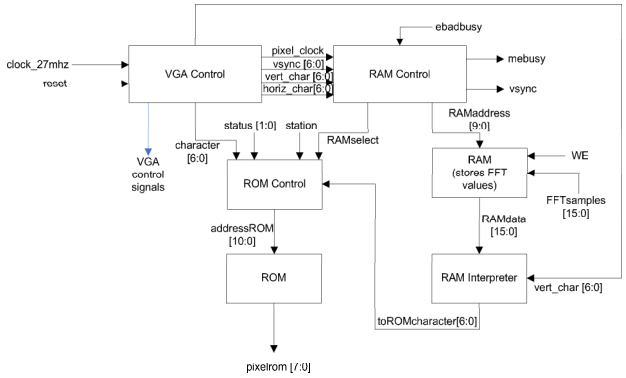

Figure 2.7: VGA Output

that corresponds to a certain frequency.

Chapter 3

Design Methodology

3.1 Analog Front End

Although it would have been desirable to maintain the original idea of our project of receiving true AM radio waves, after doing several tests in the lab, we realized that AM reception would almost be impossible in the lab. The lab behaves like a Faraday cage and receiving electromagnetic waves in the AM band proved to be very difficult. Therefore, we sacrificed the antenna and AM radio reception, and resorted to directly feeding in an AM modulate signal to the ADC module. This tradeoff was made because this modified setup would still be able to prove the functionality of the digital AM receiver. Also, it avoids the possible noise that is introduced from the recption itself by using an antenna and other analog parts.

Also, since the ADC produces the digital signal in a serial format, with the MSB produced first, a module was made to convert this serial signal into a 14-bit parallel signal to be used by the digital signal processing unit. This tradeoff was made because using the data serially would have been inefficient when it was time to carry out the digital signal processing.

3.2 Digital Signal Processing

Many critical design choices were made throughout the design and implementation part of the system.

Going back to the basics of the digital signal processing required to demodulate a signal, the particular adopted technique in this module was chosen because of its edge over all other techniques in terms of the amount of computation involved. This technique reduces the demodulation into its most basic and only necessary steps: downcoversion and low-pass filtering. This is in contrast to conventional AM radio designs where another stage is added at the beginning to band-pass filter the radio spectrum, only selecting the radio band one wants to listen to. The next steps to follow are the usual down conversion and low-pass

filtering. The additional stage is just added to compensate for the imperfection in the real world filters in terms of not completely nullifying the signals in the stop band and the transition region. Additional stages can also be added to the current DSP methodology to improve the quality of demodulation.

Another important design choice was the number of the taps for the FIR. The Matlab simulation mentioned in the implementation part was modified to be discretized to 16-bit numbers and ran with filters of different tap sizes. A copy of the Malab code is included in Appendix. Distortion in demodulated signal appeared around a size of 30 taps, and so a 30 tap filter was chosen for the FIR for the DSP module.

At the hardware implementation level, there were more decisions to be made about the digital architecture. Many of the initial decisions turned out to be unsuccessful and they had to be changed later. One example is how the design for implementing convolution developed. Initially, the digital architecture was designed to preserve all the new bits that were created as a result of a multiplication or an addition. As a result, 61 bits, an unusually large number of bits, came out of the DSP module. The convolution was designed to be carried out in 6 cycles, using a 10-input (5 pairs) of 28-bit multipliers and a 61-bit accumulator. When the system was simulated after incorporating all delays post map and route, the timing constraints turned out to be not met. The clock speed was too fast for the design. Different techniques were employed to work around this problem. The number of bits were truncated after every stage to decrease the size of the logic; the unused clock cycles of 18 clock cycle time were used to give twice as much time for the multiplier and accumulator to finish processing and lastly, the convolution multiplier and accumulator were pipelined. All these steps made the propagation delays smaller but one could see large propagation and contamination delays in the timing diagrams. The fsm could have been designed to work around it. However, after consultation with the Professor and the Teaching Assistant, it was decided not to take that route because the Modelsim delays did not really defined contract on the system timing; they are just approximations. So it was decided to expand the digital architecture to buy more time for processing. The convolution system was changed from a sequential logic to a combinational logic architecture where 30 18-bit multipliers were put in to get all multiplication results in serial simultaneously and add them serially as well using 30 adders. The simulations for the revised system pleasingly showed that the output was ready well in time before the end of the upper limit of 18 clock cycles.

Although it was difficult to make all these decisions but it increased understanding of how digital systems work and how different approaches affect the speed and size of digital logic blocks. Implementing the DSP module thus revealed the inside limitations of digital logic, particularly the duality between size and speed.

3.3 User Interface

There were quite a few design decisions that were made regarding the design of the user interface. Since the user interface was also responsible for interacting with the AC97 codec (that operates on a different clock frequency) in order to output data to the speakers, the most important design decision was to tackle two different clock frequencies at the same time. This issue was resolved by dividing the system into two parts as naturally as possible, with both parts running on different clocks. The aim was to minimize the number of signals that crossed the interface between the two parts. The signals that did have to cross the interface were registered at the clock of the sub-module to which they were input. The main issue was deciding about the location of the pause RAM, since it had to interact directly with both the Pause FSM, which ran on the system clock, and the AC97 codec, which ran on the AC97 clock. It was decided that it would be best to clock the memory on the AC97 clock, since it was supposed to output data to the AC97 codec. If the memory were run on the system clock, the data would first have to be registered on the Ac97 clock before actually being routed to the speakers. Since the aim was to minimize the number of signals crossing the interface, it was achieved by running the memory on the AC97 clock.

Other design decisions that were made were regarding the freedom to be provided to the user in customizing the system. It was decided that the user should not be able to change channels while pausing or bookmarking a channel. This meant that the Major FSM had separate states for channel selection and for pausing and bookmarking. However, the user was given the freedom to pause/replay a channel and to bookmark it at the same time. Consequently, the Major FSM had a single state for both pausing/replaying and bookmarking, and both the Minor FSMs were triggered from that state, depending on the inputs.

3.4 VGA Display

A VGA with a resolution of 800x600 is almost impossible to manually encode 1's and 0's to represent the displayed picture. However, since the display displays mostly ASCII characters, it is better to design a system that is concerned with characters, and has a separate code that paints the pixels that represent the characters. Therefore, when the VGA control module lands on a point on the screen that represents the character point (2,2) and that character is an ASCII character #65 (represents A), the code then addresses the character ROM to obtain the bitmap of that character. With the bitmap, the system sweeps through the vertical and horizontal dimensions, by incrementing variables vert_char and horiz_char, of the character bitmap and look for any 1's. If a 1, a foreground color is displayed and if a 0, a background color is displayed. We felt that this design choice saved us a lot of time, but the tradeoff was that we could not display more than one symbol per character location. Actually, it basically meant

that we working with a screen with a resolution of 100x50.

Another decision we made was that we did not need any FSMs to achieve a VGA display. At first, I approached the task with FSMs in mind. However, while designing the display, I realized that a counter would suffice. Actually, I am quite convinced that the counter is the best approach for a video display because it is a repetitive process. With a counter sensitive to the pixel clock, one can confirm that timing issues are hardly a problem.

Chapter 4

Testing

4.1 Analog Front End

There were several problems with the anlog front end. The first, as mentioned above, is that AM reception in the lab was almost impossible. The antenna could have been improved to increase its reception quality, but that would have been extremely difficult and time consuming.

Another problem was the signal integrity of the digital outputs that were connected to the FPGA. The signals were so noisy that it worsened the performance of our system. The cause of this problem was that the analog front end output signals, ClkOut, Fsout, and Dout, were transmitted over approximately 50 Ω transmission lines. The lines were terminated correctly, but the wires introduced so much parasitics, that the signal had degraded. From looking at the logic analyzer, the ADC was generating a good 55mHz clock and a conversion signal every 18 cycles, but was generating random high bits even with 0 V input. This noise was actually quite audible during testing. Figure 4.1 shows a sample waveform produced by the ADC with a 0v input.

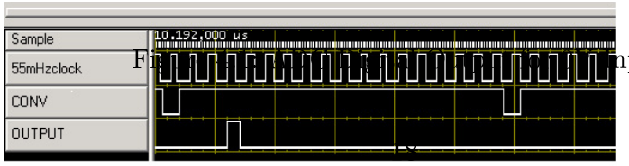

Figure 4.1: 0v input

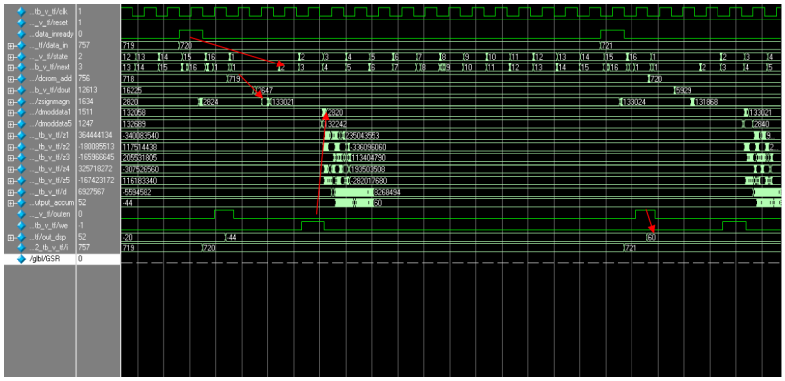

Figure 4.2: Post Map and Route Simulation

4.2 Digital Signal Processing

Figure 4.2 shows the post map and route simulation waveform for critical test signals across the different modules in the DSP module.

The data_in_ready signal is delayed through four register so that by the time the FSM detects that there is an input sample, the input signal is already passed through the multiplier and reached the register file, ready to be stored. The write enable signal goes high for a cycle so that the register file can store the new demodulated data and shift the rest by one. The critical path delay then passes through the multiplier into the adder and after a few cycles the output signal is ready at the output of the adder. After waiting a few cycles to allow for the propagation delay through the combinational logic, the output is latched into out_dsp by asserting the output enable signal high for one clock cycle. The timing of write enable and output enable works as a first in first out (FIFO) buffer. So the register file is not written until the previous data has been latched away. And similarly, the output enable is not asserted high until the signals has gone stable after all the propagation delays.

4.3 User Interface

Testing the user interface was somewhat difficult, because the outputs of the user interface were internal to the system, and were actually inputs to other modules, rather than outputs of the system as a whole. Consequently, the major technique employed for testing and debugging the user interface was through simulations on ModelSim and Max+II. In order to further test the system, the user interface outputs could be connected to video and their correctness could be verified. Simulations on ModelSim helped a lot in debugging, and the post-map and post-place and route simulation tool in Model Sim was very accurate and helpful in demonstrating the actual gate delays on hardware. Shown below

are simulations for the main modules of user interface.

4.3.1 Book FSM

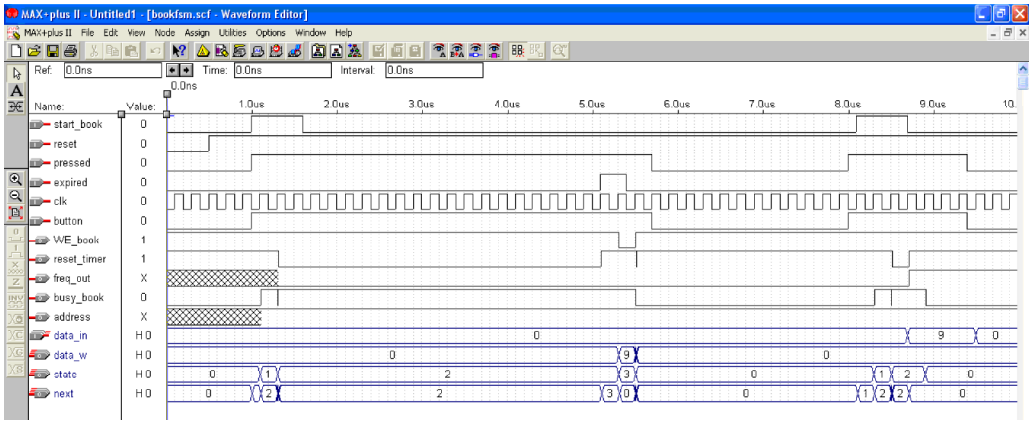

Figure 4.3: Book FSM Simulation

4.3.2 Major FSM

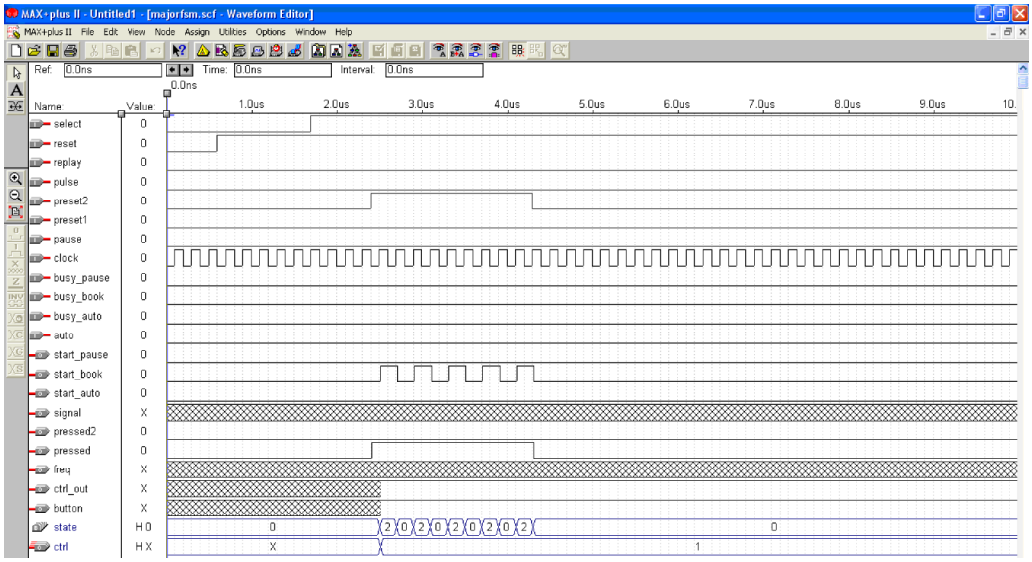

Figure 4.4: Major FSM Simulation

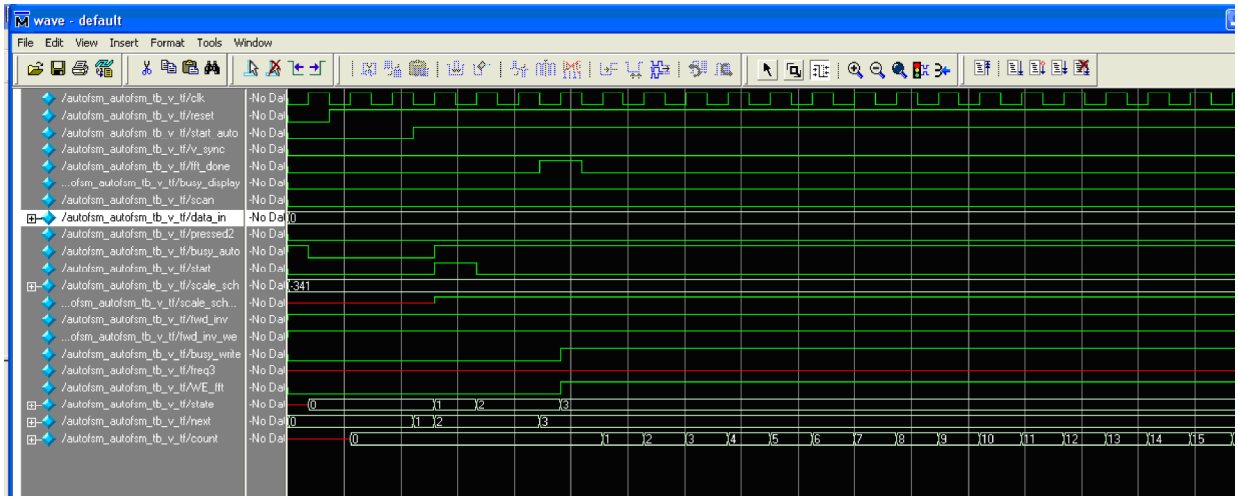

Figure 4.5: Simulation

4.4 VGA Display

An important detail to look at when doing VGA testing is timing. As stated before, the counter was chosen to help alleviate any timing issues. If timing was incorrect, one would see that the screen had lines and characters misalligned. It was crucial to ensure that the timing between the FPGA choosing a character to display and actually displaying it is constant. If it was not constant, parts of characters would overlap and very little would be discernable. Therefore in my testing, I made sure that the time for a pixel to output was the same for both when content information came from the RAM or from hard code. Also, I made sure that as the bitmap of the characters returned from the ROM I was picking the correct pixel to display. There were several cases in my simulations that I saw that the character bit representation was off by a couple of pixels. However, by fixing how the module reads the horiz_char variable to select that bit of the 8-bit row of the pixel, the problem was resolved.

Chapter 5

Conclusion

The Digital AM receiver receives the entire AM frequency band, digitizes it, and processes the digitized AM band. When the user selects a station, the dsp module selects the appropriate channel and plays it out to the user. Our goal was to implement a Digital AM receiver that achieves the same functionality as a conventional analog AM receiver. With additional features such as a video display and a user interface, the Digital AM receiver was intended to be a rich, full-featured AM receiver implemented in the digital domain.

Though we were unsuccessful in completely achieving our project, there are a number of valuable lessons that were learnt in the process of implementing the DSP module. There are real constraints on the speediness and accuracy of the execution of the signal processing done in digital domain. However, these constraints can be offset by expansions in digital architecture. Another lesson learnt was that the new labkit is huge in terms of gates but slow as far as the propagation delays associated with wires is concerned. The state signal which is just a register, changed after half a clock cycle. As far as digital AM radio is concerned, there are ways to improve it. More radio frequencies can be downconverted by storing more samples in the down conversion ROM. The filter size can be increased to implement a better low-pass filter using the remaining built-in multipliers in the labkit. Also more work can be done on the analog front end so that a better signal comes to the DSP module.

This multidisciplinary topic project helped us apply the powerful digital concepts we have learned this year and yet learn other concepts beyond the scope of this class. The final project was a great way to conclude this challenging term of fundamental concepts in Electrical Engineering.

Chapter 6

Acknowledgements

We would like to thank the 6.111 staff: Professor Anantha Chandrakasan and TA's: Jenny Lee, Chris Forker, and Charlie Kehoe. Their commitment and dedication to the class helped motivate students to stay late to complete their labs. This teaching staff has been the best we have seen here at MIT.
We would also like to thank Joe Sousa for his assistance in building the analog front end. Without his help, the project would have been a lot more difficult to achieve.
Lastly, we would like to thank Luca Daniel and Faisal Kashif for their assistance in designing the digital signal processing module. There were many concepts that were beyond the scope of this class and their guidance along the way helped us tremendously.