Frequency Domain Analysis and Processing of Audio Signals

Dan R. K. Ports
Albert C. Chiou
Andrew L. Clough

December 12, 2003

Abstract

The frequency domain is a natural representation for audio signals. Our project explores the frequency domain representation of an audio signal by allowing the user to view the frequency spectrum of a supplied analog input signal. It also performs processing of the audio signal in the frequency domain, making possible a variety of audio effects: frequency shifting, harmonic generation, peak sharpening, etc. Hence, the project implements — in the FFT / Conversion module — analog/digital and digital/analog sampling and transforms via the Fast Fourier Transform and Inverse Fast Fourier Transform. The Processing module implements the frequency domain processing. Finally, the Video module displays frequency spectra of the input and output signals on a video monitor using the MC6847 video processor chip.

Issues of testing and debugging are discussed. Though the project was not fully implemented due to some numeric error and integration issues, all components were implemented and tested in simulation, and most were also realized in hardware.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

1.1 Overview

Audio signals are typically described in the time domain: by measuring the amplitude of the signal at regular time intervals. This is the representation seen when viewing a signal on the oscilloscope, for example. But the frequency domain is another natural representation for an audio signal. Instead, the signal is described in terms of its frequency components, built up from exponential functions. This representation proves useful because we frequently think of audio signals in terms that relate to their frequency components: describing the pitch of a musical note, or the quality of harmonics, for example.

This project is intended to explore audio signals in the frequency domain. An incoming audio signal is first sampled and converted to the frequency domain, using the Fast Fourier Transform. Its spectrum is displayed on a video monitor. The signal is processed in the frequency domain, using a user-selected filter. A variety of interesting effects become possible by manipulating the frequency domain representation. The spectrum of the output signal is also displayed, and the output signal is converted back to the time domain by the Inverse Fast Fourier Transform and output as an analog signal.

1.2 Operation

Operation of this device proceeds nearly automatically. The reset button on the FFT module must be held down first while the reset button on the Processing module is pressed. This synchronizes the starting states to begin parallel bus communication.

The FFT module will immediately begin sampling the input signal and computing the Fast Fourier Transform, as well as transmitting data over the parallel bus, and receiving processed results from the Processing module. The output signal will be automatically transformed via the inverse FFT and output via the digital-to-analog converter.

This device accepts a differential voltage input (V_{in}) with amplitude at most ± 1.28V and processes it. The output is generated with a 1.28V offset, i.e. a range of 0V – 2.56V referenced to ground. Sampling is performed at approximately 22.143 kHz, so input signals can contain frequency components up to approximately 11 kHz without causing aliasing. Additional analog stages were originally planned to allow for different input and output voltage ranges, but this was not implemented due to time constraints.

The Processing module provides a variety of audio effects that can be applied to the input in the frequency domain. It can perform frequency shifting, generate harmonics, generate half-harmonics, and sharpen frequency peaks. The active effect can be selected by means of a group of switches on

the Processing kit; initially, the parameters to each effect were to be configurable via a knob, but this functionality became impossible to implement due to a failure of the lab kit's NuBus interface.

The Video module automatically displays the frequency spectra of the input and output signals on a video monitor.

1.3 Design Overview

The design of this system was divided into three components, each designed and implemented by one person, using one lab kit.

The Fast Fourier Transform / Conversion module, implemented by Dan Ports, performs the necessary conversions between the analog and digital and time and frequency domains. It uses an analog-to-digital converter and digital-to-analog converter to sample the input signal and generate the output signal; samples are stored in RAM buffers. It can perform the Fast Fourier Transform and inverse Fast Fourier Transform to convert the input and output signals from the time and frequency domains, respectively. Finally, it also exchanges data with the other two kits, transmitting frequency coefficients for the input signal to the Processing and Video modules, and receiving frequency coefficients for the output signal from the Processing module.

The alterations to the time domain data are accomplished in the Processor module. Here the incoming data is saved into a RAM, then read out in a slightly different order, while sometimes being multiplied by a constant factor. In this way the data can be frequency shifted up or down, sharpened, or have harmonics added to it.

The Video module displays the frequency spectra of the input and output signals on a video monitor. This is accomplished by reading data from the three-way parallel bus (described below) into a RAM, then using a finite state machine to generate a video representation in RAM. This is translated to a video signal by a MC6847 chip and an output stage.

1.4 Interface Protocol

The three modules of this kit must communicate frequency data between each other. The FFT module must send frequency coefficients for the input signal to the Processing and Video modules, and the Processing module must send the frequency coefficients for the output signal to the FFT and Video modules. This communication is implemented by using a three-way 8-bit parallel bus.

The parallel bus requires eight data bits, plus five control signals. These signals are connected between all three lab kits. The signals are listed in Table 1.1. All bus lines are interfaced to the FPGAs using Schmitt-triggered line drivers and receivers (e.g. 74LS244 or 74LS245).

Signal	Source	Description
DATA	FFT or Processing	8-bit wide data bus
FFTSENDING	FFT	High when FFT is transmitting
PROCSENDING	Processing	High when Processing module is transmitting
FFTRDYACK	FFT	Data ready (if FFT sending) or acknowledge (if not)
PROCRDYACK	Processing	Data ready (if Processing sending) or acknowledge (if not)
ACK2	Video	Acknowledge from video module

Table 1.1: Parallel Bus Signals

Each transmission consists of 2048 bytes: 1024 frequency coefficients, with the real byte transmitted first, followed by the imaginary byte. A transmission cycle begins when the FFT module sets FFTSENDING high. It then places the first byte on the data bus, and sets FFTRDYACK

high. The two other modules read the data bus when they sense FFTRDYACK is high. Once they are finished, they set their respective acknowledge signals (PROCRDYACK and ACK2) high. The FFT module will set FFTRDYACK low again when both acknowledge lines are high, and wait for both acknowledge lines to go low again. It then places the next byte on the data bus, and repeats the cycle. After the final byte has been transmitted and acknowledged, it sets FFTSENDING low again.

This signals the Processing module to set PROCSENDING high, taking control of the data bus. It then repeats essentially the same cycle, except that PROCRDYACK becomes its ready line and FFTRDYACK becomes the FFT's acknowledge line: placing the first byte on the data bus, setting PROCRDYACK high, waiting for FFTRDYACK and ACK2 to go high, setting PROCRDYACK low, waiting for the two acknowledge signals to go low, and repeating. When it completes, it sets PROCSENDING again and waits for the FFT to start another cycle.

Chapter 2

Conversion / FFT Module (D. Ports)

2.1 Overview

The Conversion / FFT Module is used to convert between the analog, time domain input and output signals that interface to the external world, and the digital, frequency domain representations used by the Processing and Video Display modules. Hence, it must perform several functions. It must convert the input signal from analog to digital, and the output signal from digital to analog. It must transform the input and output signals between the time and frequency domains; this is implemented using the Fast Fourier Transform (FFT). Finally, it must exchange frequency coefficent data with the other components of this project via a parallel interface. These are implemented using three subsystems which operate concurrently: an Analog/Digital (AD) subsystem, a FFT subsystem (which also implements the inverse FFT), and a parallel interface subsystem. In addition, a control unit and a memory manager are used to allow the three subsystems to operate at the same time without any concurrent execution issues.

The design of the module is represented by the FPGA Block Diagram and Hardware Wiring Diagram in Figures 2.1 and 2.2.

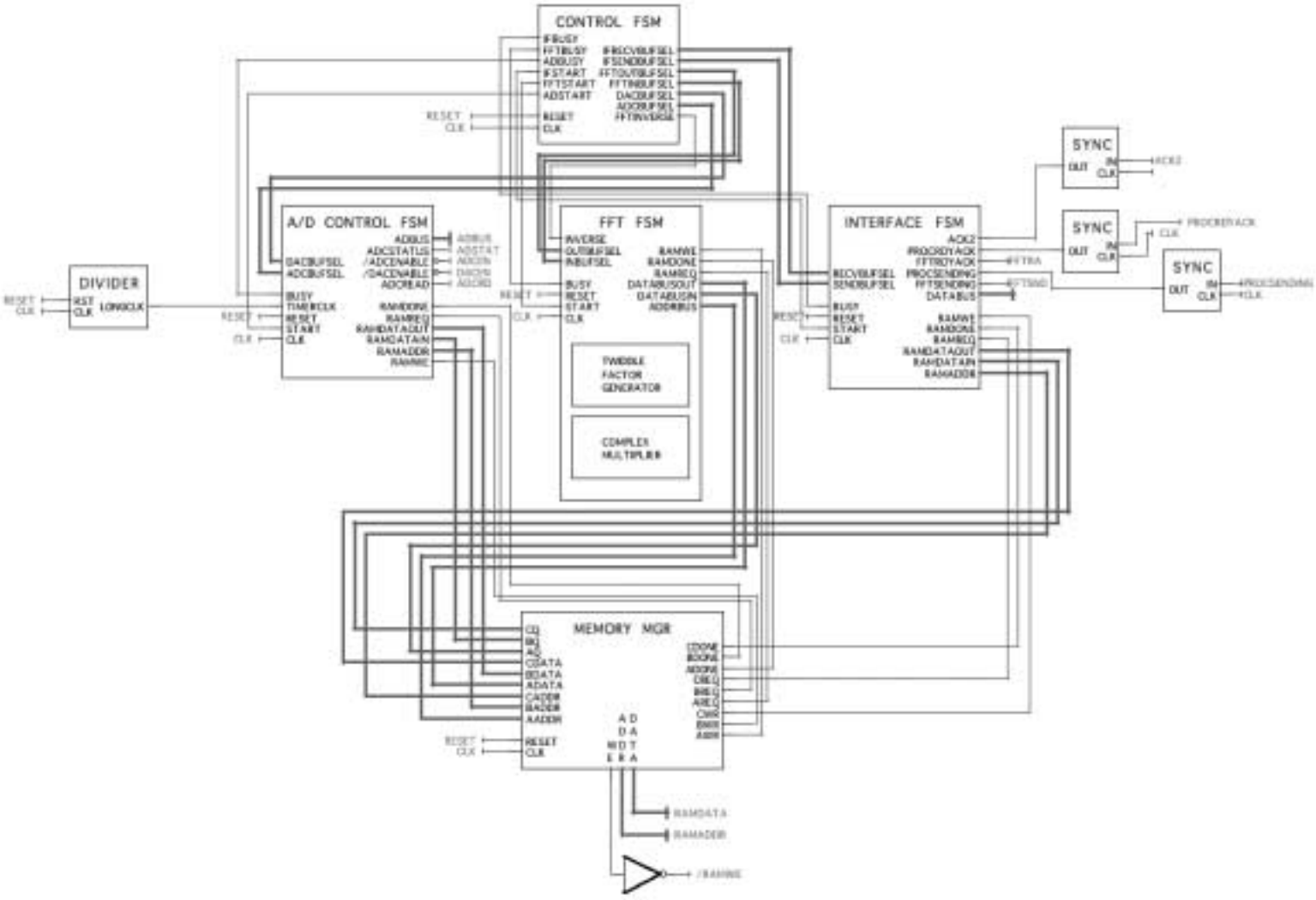

Figure 2.1: FPGA Block Diagram

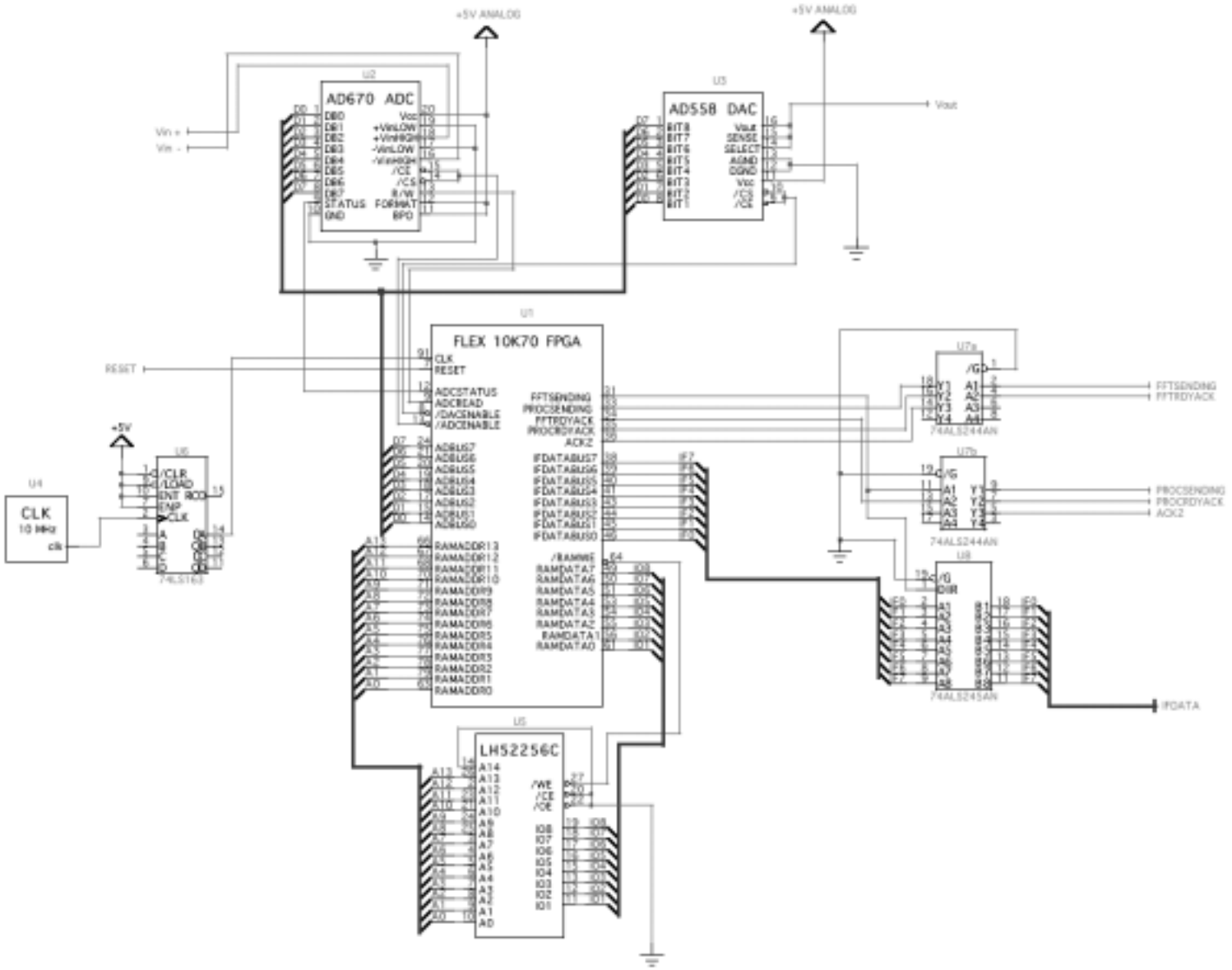

Figure 2.2: Hardware Wiring Diagram

2.2 Control / Memory

The three subsystems of this module operate concurrently with a reasonably complex memory usage scheme. They therefore require complex control logic and memory management to ensure that they are operating at the correct times and using the correct buffers.

2.2.1 Clock

All synchronous components in this system operate on the rising edge of a single, global clock signal. Because a 10 MHz clock signal was too fast to satisfy the access time requirements of the SRAM chip, a 5 MHz clock was chosen instead. This clock signal is generated by dividing the standard 10 MHz 25% duty cycle NuBus clock integral to the lab kit by half using a 74LS163 counter.

2.2.2 RAM

Eight logical buffers, each containing 2048 locations of eight bits each are used. These eight buffers are arranged as four "rotating buffers"; i.e. pairs of buffers such that the previous computation's results are read from one buffer while the current computation's results are computed and stored into the other buffer. After each cycle of operation (one FFT/inverse FFT computation), the input and output buffers are swapped.

One rotating buffer is used for storing the samples read from the analog-to-digital converter; it also serves as the input to the FFT. A second rotating buffer is used to hold the source data for the digital-to-analog converter and results of the inverse FFT. Another buffer is used to hold the results of the FFT of the input signal; it serves as the source for the parallel bus's output to the other kits. A final rotating buffer is used to hold the data received via the parallel bus and acts as the source for the inverse FFT.

The FPGAs used in this implementation were not capable of synthesizing a RAM unit of appropriate size (8 buffers $\times$ 2048 locations $\times$ 8 bits=16384 bytes) to hold the eight buffers. Hence, an external RAM chip was used. A Sharp LH52256C-70LL 32K $\times$ 8 SRAM chip was selected. With twice as much capacity available as necessary, one address line was tied to ground. The remaining address lines were controlled by the memory manager implemented in the FPGA. The output enable signal was tied low, and pulses on the write enable signal were used to indicate when data was available for writing. The write enable signal was gated with the clock to avoid spurious writes due to glitches; however, this required the clock period to be increased so that one half period would be greater than the 70ns access time of the RAM.

2.2.3 Memory Manager

A memory manager is used to control access to memory. The memory manager has one external port that connects to the RAM module, and provides three logical ports that can be accessed by the A/D, FFT, and interface controllers. These three ports are multiplexed in such a way that each controller can send a request simultaneously, and they will be serviced sequentially as the RAM becomes available.

Each logical port consists of a 14-bit address bus, 8-bit input and output data buses, a positive-true write enable signal, a request signal, and a done signal. The request signal is used to indicate to the controller that a request is available and should be executed; the done signal is high for one clock cycle after the request has been serviced. The request signal must return low on the next clock cycle after the done signal is asserted, or the request will be repeated.

Internally, the memory manager maintains two state variables. One maintains the current port information, corresponding to the number of the port whose request is currently being serviced, or zero if no requests are in progress. This is used to multiplex the RAM address, data, and write enable lines. The write enable line is also gated with the clock, so that glitches can be avoided.

The port to be accessed on the next clock cycle is determined combinationally by inspecting the request lines from each port. A port's request line will be ignored if its done line is currently high. The other state variable used by the memory manager is used to help determine which port should be accessed first. Each cycle, a different port is given priority; this ensures that processing does not stall while one FSM continually requests memory access. An internal state variable is incremented each clock cycle, and this is used to determine the precedence of port request lines.

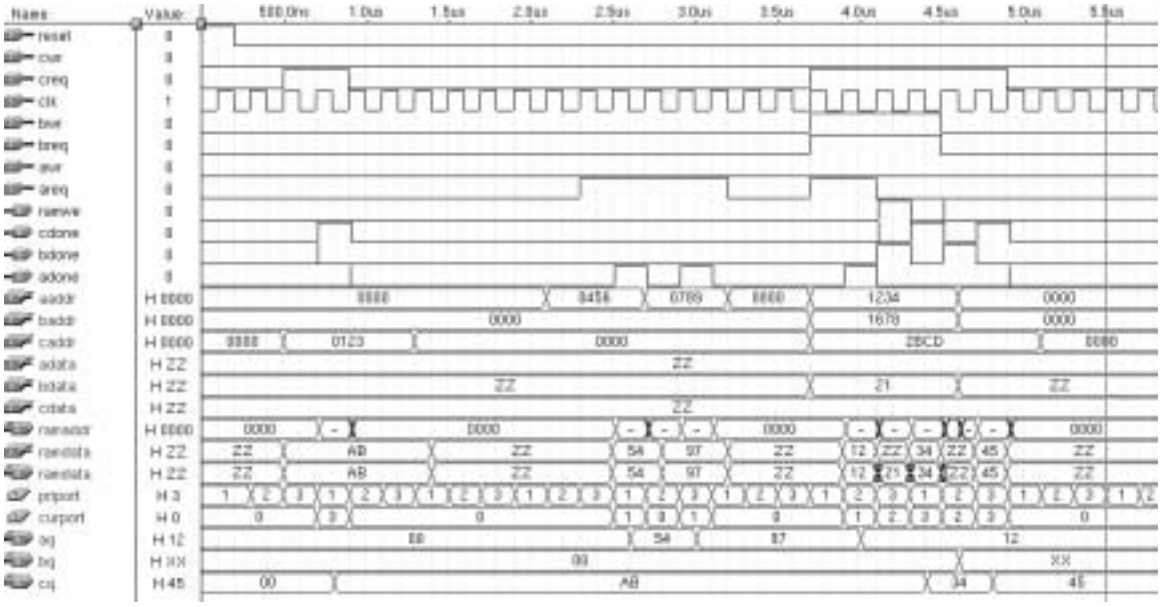

Figure 2.3: Memory Manager Timing Diagram

2.2.4 Control Unit

The control unit handles the task of ensuring that all three subcomponents are operating simultaneously. It also controls their buffer selection signals. It alternates between the input and output buffers in the rotating buffers described above, and also selects the appropriate signals for the FFT and inverse FFT modes.

The control unit is implemented as a synchronous finite state machine. Its state transitions are shown in the state diagram in Figure 2.4.

The control unit begins in the s_Start state; it also returns to this state on a reset signal. In this state, it asserts the start signal to each of the three FSMs. It waits until each of the three busy signals from the FSMs become high, then moves to the s_Wait1 state, in which it waits for the FFT to terminate. When the FFTBusy signal goes low again, it moves to the s_StartIFFT state. In this state, it selects the appropriate input and output buffers to perform the inverse FFT, sets the FFTInverse signal high, and asserts the StartFFT signal again. As soon as the FFTBusy signal goes high, it then moves to the s_Wait2 stage, in which it waits for all three busy signals to go low. Finally, it moves to the s_Rotate state, in which it alternates between buffer states as described in Table 2.1. It then returns to the s_Start state and starts another cycle.

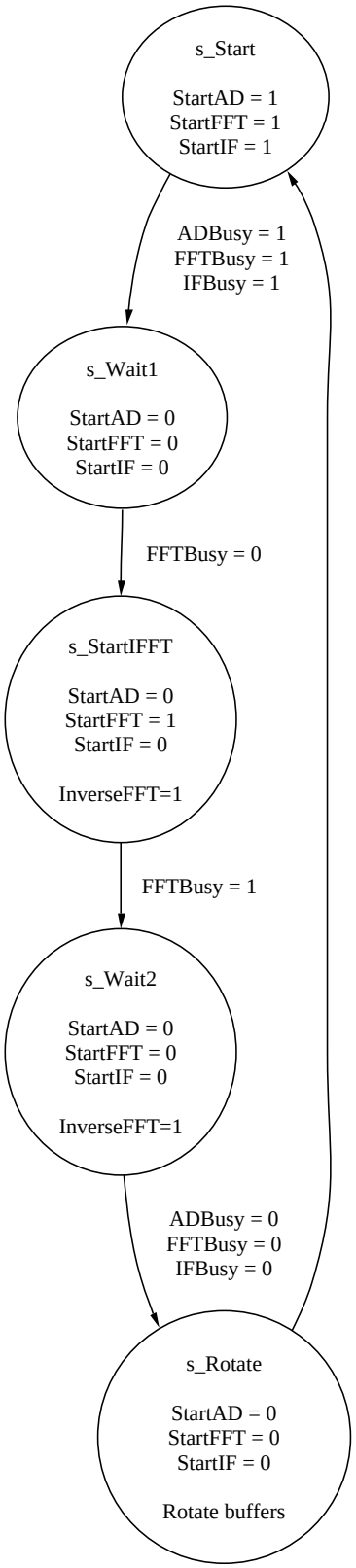

Figure 2.4: Control FSM State Diagram

Buffer Variable	State 0	State 1
ADC Output	000	111
DAC Output	001	010
FFT Input	111	000
FFT Output	110	101
IFFT Input	100	011
IFFT Output	010	001
Interface Send	101	110
Interface Receive	011	100

Table 2.1: Buffer Variable Rotation States

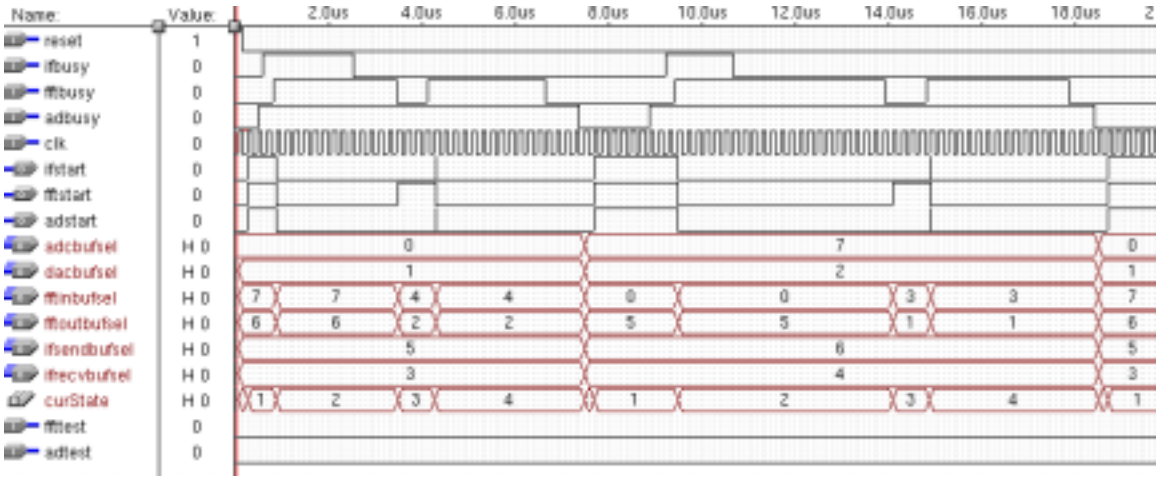

Figure 2.5: Control FSM Timing Diagram

2.3 Analog / Digital Subsystem

The Analog/Digital (A/D) subsystem converts analog input values to digital representations for processing, and converts the output values back to an analog signal. All A/D processing is done using blocks of 1024 samples. A finite state machine controller operates the external analog to digital and digital to analog converters.

2.3.1 Analog / Digital Control FSM

The A/D Control FSM controls the analog-to-digital (ADC) and digital-to-analog (DAC) converters. It is controlled by the major control FSM, which provides a Start signal to indicate when a conversion cycle should be performed, and selects the ADC Output and DAC Input buffers. The A/D FSM then performs a series of 1024 conversions, during which the FSM outputs a busy signal. When the conversion cycle is completed, the FSM sets its busy signal low and returns to an idle state to await the next start signal.

The state transitions for the FSM are depicted in Figure 2.6. The FSM begins in the s_Idle state, in which it waits for a start signal from the Control FSM. When the start signal is received, it moves to the s_WaitForTimer state, and waits for a pulse from the clock divider; this indicates that it is time to perform one sample. It then loads the DAC output value from the DAC input buffer (in the s_RAMRead state), and outputs it to the DAC (in the s_DACWrite state). The next two states, s_ADCEnable and s_ADCWait together activate the ADC and wait for it to perform a conversion. Once conversion is completed, the s_ADCRead state reads the output of the ADC into an internal buffer; it is then written to the ADC output RAM buffer in the s_RAMWrite1 state.

The address counter is then incremented in the s_AddrInc1 state. As the analog input and output values can take only real values (not complex), the imaginary component of the buffer clearly must be ignored. Thus, the next memory location in the ADC buffer must be filled with a zero, and the next memory location in the DAC buffer must be ignored. This is performed by writing a zero in the s_RAMWrite2 state, and incrementing the address counter again in the s_AddrInc2 state. Finally, the FSM returns to the s_WaitForTimer state to await the next sample timer pulse, or instead to the s_Idle state after 1024 input samples have been recorded and the address counter returns to zero.

2.3.2 Analog-to-Digital Converter

An analog-to-digital converter (ADC) is required to convert the incoming analog signal to a digital representation. The Analog Devices AD670 chip is used to fulfill this requirement. The chip accepts a differential voltage input, and is configured for a bipolar ±1.28V range. The two's complement output format is selected, using the wiring configuration shown in Figure 2.2.

The A/D control FSM handles the task of starting ADC conversions when appropriate and monitors the status line.

2.3.3 Digital-to-Analog Converter

The digital-to-analog converter (DAC) converts the computed output value to an analog voltage. This is implemented with an Analog Devices AD558 chip. The V_{out}, V_{out} Sense, and V_{out} Select outputs are tied together to select the +2.56V range. The DAC is powered by the lab kit's +5 Analog supply, and referenced to ground. The output range is 0 – +2.56V, so the zero value has a 1.28V offset relative to ground.

The DAC and ADC share a data bus. Because both chips are controlled by the Controller FSM, only one is active at any given time, and bus contention can never occur.

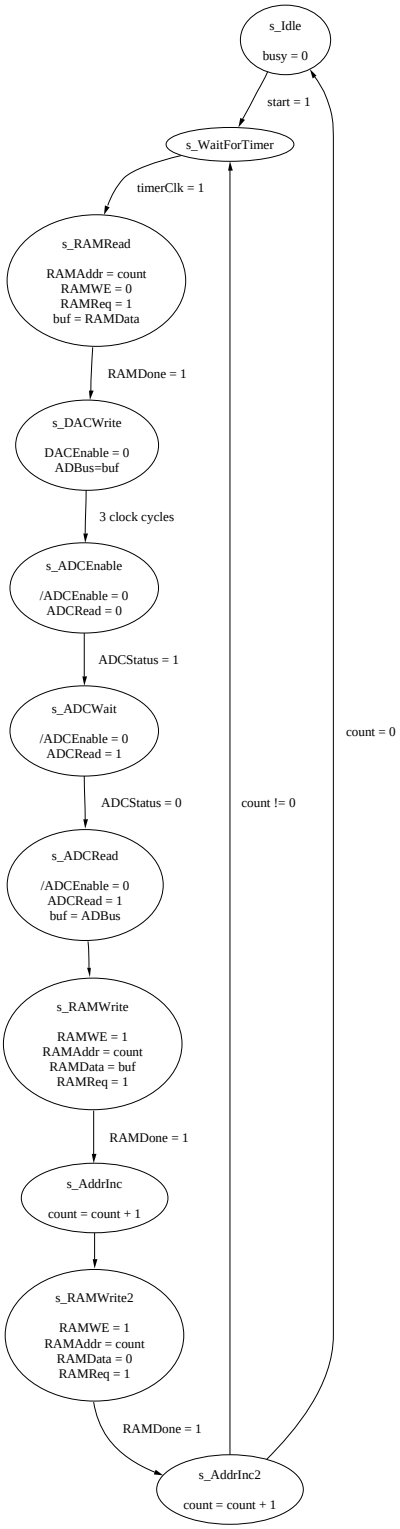

Figure 2.6: A/D Control FSM State Diagram

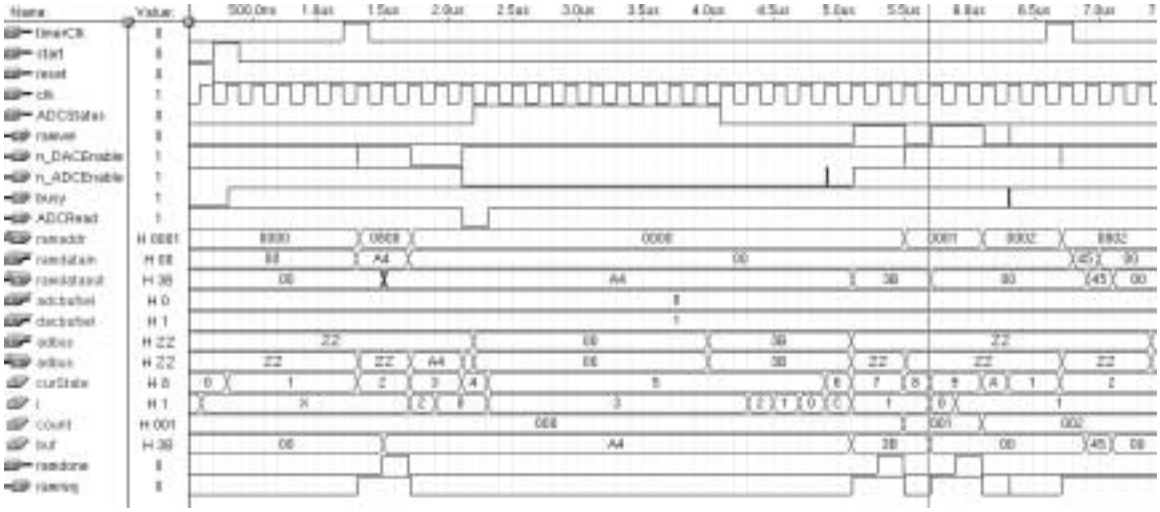

Figure 2.7: A/D Control FSM Timing Diagram

2.3.4 Divider

The purpose of the divider is to generate a 22.143 kHz sample timer signal from the 5 MHz clock signal. The output of the divider is a signal that is high for one pulse of the system clock, each time a sample needs to be performed. This is done by maintaining an 8-bit counter that is incremented each clock pulse. When the counter reaches 225, the output is set high. On the next clock pulse (when the counter is 226), the counter is reset to zero and the output is set low again.

The divider also has a reset signal that allows the counter to be reset to zero. This is connected to the RESET input.

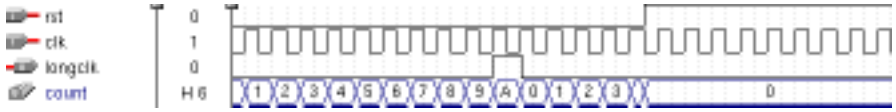

Figure 2.8: Divider timing diagram (ratio changed from 226:1 to 10:1)

2.4 Fast Fourier Transform Subsystem

2.4.1 Theory

Discrete Fourier Transform

The discrete Fourier transform (DFT) converts a discrete-time representation of a signal into a discrete frequency representation: the sampled spectrum of the input signal. The DFT of a signal of length N is defined by the analysis equation

$$\mathrm{DFT}\left\{x[n]\right\}[k] = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x[n] W_N^{kn} \tag{2.1}$$

and also satisfies the synthesis equation

$$x[n] = \frac{1}{\sqrt{n}} \sum_{k=0}^{N-1} \text{DFT}\left\{x[n]\right\}[k] W_N^{-kn} \tag{2.2}$$

in which W_N^k is known as the kth twiddle factor and is defined by

$$W_N^k = e^{-\frac{2\pi jk}{N}} \tag{2.3}$$

The similarity between the analysis and synthesis equations is noteworthy. The inverse DFT is simply the same as the forward DFT, except that the twiddle factor W_N^{kn} is replaced by W_N^{-kn}. Applying the definition of the twiddle factor, it is not difficult to see that the only change necessary to perform an inverse DFT is to invert the sign of the imaginary component of the twiddle factors.

Fast Fourier Transform

The Fast Fourier Transform (FFT) is an efficient method for computing the DFT. The specific form of the FFT used in this implementation is a radix-2 in-place decimation-in-time FFT, a variant of the algorithm originally presented by Cooley and Tukey in 1965.

This algorithm relies on the fact that the DFT of $x[n]$ can be written as

$$X[k] = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x[n] W_N^{kn} = \frac{1}{\sqrt{N}} \sum_{i=0}^{\frac{N}{2}-1} x[2i] W_N^{2ik} + \frac{1}{\sqrt{N}} W_N^k \sum_{i=0}^{\frac{N}{2}-1} x[2i+1] W_N^{2ik} \tag{2.4}$$

i.e. that the DFT can be written as the sum of the DFTs of the even and odd halves of the input sequence, with one multiplied by a twiddle factor. This allows a DFT of size N to be decomposed into two DFTs of size $\frac{N}{2}$. The recursive application of this procedure is the radix-2 decimation-in-time FFT.

Of course, since recursion is difficult to implement in hardware, an iterative approach is used instead. This performs the DFT from the bottom up, beginning by computing a number of two-point DFTs, and then using these to compute four-point DFTs, and so on until the full DFT is computed.

The elementary computation used is a butterfly, so named because of its graphical representation. It takes two points from the input or an intermediate stage, and determines their new values in the next stage:

$$X[p]' = X[p] + W_N^r X[q] \tag{2.5}$$

$$X[q]' = X[p] - W_N^r X[q] \tag{2.6}$$

The values of p, q, and r depend on the arrangement of the butterflies in each stage. However, it is noteworthy that the next-stage values of $X[p]$ and $X[q]$ (denoted $X[p]'$ and $X[q]'$ respectively) depend only on the previous stage's values of $X[p]$ and $X[q]$. Therefore, the FFT computation can be performed in-place, with only one buffer required.

The arrangement of butterflies in each pass requires some reasonably complex logic. There are $\log_2 N$ passes (in this case, $N = 1024$ and $\log_2 N = 10$). In each pass, $\frac{N}{2}$ butterflies are performed. However, they are divided into a different number of blocks: in the pth pass, $\frac{N}{2^{p+1}}$ blocks are performed, each consisting of 2^p butterflies.

The iterative implementation is best understood by examining the pseudocode in Listing 2.1, written in a C-like language that supports complex multiplications[1].

[1] Based on an procedure given by Engineering Productivity Tools at http://www.eptools.com/tn/T0001/PT04.HTM

Listing 2.1: Pseudocode for FFT Algorithm

```
fft()
{
        for (i = 0; i < 1024; i++)
        {
                // copy from input to output buffer in
                // bit-reversed order
                outbuf[bitreverse(i) & 0] = inbuf[i & 0];
                outbuf[bitreverse(i) & 1] = inbuf[i & 1];
        }

        Bp = 512;
        Np = 2;

        for (p = 0; p < 10; p++)
        {
                // perform a pass
                BaseT = 0;

                for (b = 0; b < Bp; b++)
                {
                        // perform a block
                        TwiddleSel = 0;
                        BaseB = BaseT + Np * 2;
                        for (k = 0; k < Np/2; k++)
                        {
                                // perform a butterfly
                                t = outbuf[BaseT + k];
                                b = outbuf[BaseB + k];
                                outbuf[BaseT + k] = t + twiddle(k * Np/2) * b;
                                outbuf[BaseB + k] = b + twiddle(k * Np/2) * b;
                        }
                        BaseT = BaseT + Np;
                }
                Np = Np * 2;
                Bp = Bp / 2;
        }
}
```

2.4.2 Sources of Error

While the DFT and FFT are mathematically ideal algorithms that can be proven to give the correct result for any input sequence, this implementation is only a numeric approximation to the ideal algorithm, subject to several sources of imprecision.

Our implementation of the FFT was verified to be working in simulation and gave somewhat acceptable results for some input sequences, but was subject to considerable error that made the results unacceptable. For more information about the results, see Section 2.6.

Finite-Precision Arithmetic

As a mathematical ideal, the FFT should be performed using complex numbers with infinite precision. Of course, this is not possible to implement in hardware. Instead, we use 8-bit representations of the real and imaginary components of a complex number, providing one bit of sign information and 7 bits of magnitude.

This necessarily introduces some error into the calculations. In particular, the error was observed to be substantially greater when moving from the 16-point FFT in testing to the 1024-point FFT. This is not surprising, because more stages of computation are required, in which the errors are compounded. Indeed, the Cooley-Tukey FFT algorithm has an error bound of $O(\epsilon \log N)$, where ϵ is the numeric precision in use.[2]

The error is especially sensitive in this application, because an inverse FFT is performed to the values generated by the FFT. This allows the errors to be compounded even more than they would be by the 1024-point FFT alone. In addition, any error introduced by the processing stage will also have an effect.

Scaling, Overflow, and Underflow

A second problem is the limited range of the fixed-point values used in the computation. Each variable can hold only values of magnitude less than or equal to 128 (0x7F or 0x80). The input values produced by the ADC can reach these extremes, and there are very few assumptions that can be made in this application about the properties of the input signal.

Suppose the input signal is a large constant value distributed evenly over the time domain. Then the FFT of this signal will be a large impulse concentrated at one point. This is particularly true if the FFT analysis and synthesis equations are expressed with the scaling factors typically used in engineering applications:

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn} \tag{2.7}$$

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] W_N^{-kn} \tag{2.8}$$

Here, the FFT can have a greater magnitude than the input signal. Since the input signal can already reach the extremes of the memory limits, overflow is clearly a problem. To avoid this, we express the analysis and synthesis equations using a scaling factor of $\frac{1}{\sqrt{N}}$, as is commonly done in

[2] Wikipedia — Fast Fourier Transform, http://en.wikipedia.org/wiki/Fast_Fourier_transform

theory:

$$X[k] = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x[n] W_N^{kn} \tag{2.9}$$

$$x[n] = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} X[k] W_N^{-kn} \tag{2.10}$$

This is implemented by shifting each intermediate result right one bit during every alternate pass. But this is discarding data, so it results in an increased imprecision, and underflow problems become apparent for input signals of small magnitude. We thus find ourselves in the paradoxical situation of having problems with both overflow and underflow simultaneously. This problem cannot easily be solved short of increasing the width of the data values, which was not practical due to concerns with available FPGA resources.

Twiddle Factor Error

The twiddle factor values are constants, limited in precision by the width of the value. An 8-bit two's complement fixed point representation was chosen, which provided the twiddle factors with the same numeric precision as the sample values. However, as the twiddle factors are used in every butterfly calculation, the numeric instability caused by error in the twiddle factors is substantial. The Cooley-Tukey algorithm is very sensitive to errors in the twiddle factors.[3] As there were 512 twiddle factors used in the computation of the 1024-point FFT, and an 8-bit representation was used for the real and imaginary components of each value, providing only 256 possible values for each component, it is clear that some error is inevitable. This had a negative effect on the results of the FFT.

In addition, one of the most commonly used twiddle factors, $W_N^0 = 1$, could only be represented as 0x7F, a value of $\frac{127}{128}$. This particular special case could have been specifically accounted for if time had been available, but it demonstrates the problem with error in the twiddle factors. It did cause an observable scaling down of the results.

2.4.3 FFT FSM

The Fast Fourier Transform is implemented using a finite state machine that handles all the logic of loading and storing values from RAM, performing butterfly arithmetic, and ensuring that the butterflies are performed in the correct order. The FFT FSM is controlled by the Control FSM. When it receives a start signal, the FFT FSM first copies the input values from the input buffer to the output buffer (buffers are selected by the Control FSM). It then perform a series of ten passes of blocks of butterflies. When these complete, the DFT of the input signal will be stored in the output buffer.

The state transitions for the FFT FSM are shown in the state diagram in Figure 2.9.

The FSM begins in the s_Idle state, in which it waits for the start signal. When this is received, it begins to copy from the input buffer to the output buffer in bit-reversed order. All address bits are reversed, except for the first three, which specify the buffer, and the last one, which specifies the real or imaginary component. This process is achieved by the s_CopySel and s_CopyWr states. After all 2048 values are copied, the computation of the FFT begins.

The execution of the FFT is performed using ten passes, each made up of 1 to 512 blocks of 512 to 1 butterflies, as in the pseudocode in Listing 2.1. Hence, the s_PassStart, s_PassEnd, s_BlockStart, s_BlockEnd, and s_BflyEnd states determine when to start and end passes and blocks, and update the necessary state variables upon each transition — see the state diagram in Figure 2.9 for details.

[3] Wikipedia — Fast Fourier Transform, http://en.wikipedia.org/wiki/Fast_Fourier_transform

The s_BflyLoad and s_BflyWrite states perform the actual butterfly computation. In the s_BflyLoad state, the real component of outbuf[BaseT+k], the imaginary component of outbuf[BaseT+k], the real component of outbuf[BaseB+k], and the imaginary component of outbuf[BaseB+k] are loaded into buffers tre, tim, bre, and bim respectively. These are then combined combinationally with the twiddle factors via the complex multiplier, adders, and subtractors, to generate the output values. In the s_BflyWrite state, the output values are written to the same locations. The FSM then moves to the s_BflyEnd state, in which it increments the k counter, and determines whether to perform another butterfly in the same block, or move to the next block.

Timing diagrams for the FFT FSM are provided in Figures 2.16, 2.17, 2.18, 2.19, 2.20, and 2.21.

2.4.4 Complex Multiplier

A complex combinational multiplier is used to multiply the twiddle factors and the sample values. The multiplication is performed according to the identity

$$(a + bj)(c + dj) = (ac - bd) + j(bc + ad) \tag{2.11}$$

Hence, four combinational multipliers (described below) are instantiated in order to multiply each of the product terms, and an adder and subtractor unit combine the results to form the real and complex parts of the product.

While this approach is expensive in terms of FPGA resource allocation, it allows the computation to be performed rapidly, in less than one clock cycle. This allowed for greater optimization of the butterfly execution loop.

2.4.5 Multiplier

Four combinational multipliers were instantiated as part of the complex multiplier unit. Each of these multipliers accepted two 8-bit two's complement input values, and generated as output their product in a 16-bit two's complement representation.

The combinational multiplier was implemented using Altera's parameterized module library: specifically, the LPM_MULT component.

2.4.6 Twiddle Factor Generator

A twiddle factor generator was used to generate the values of the twiddle factors. The twiddle factor generator takes as input the index of the twiddle factor requested, and outputs the real and imaginary components of the appropriate twiddle factor. The imaginary component can then be negated by the FFT FSM if necessary in order to perform the inverse FFT.

Twiddle factor real and imaginary coefficients are represented as 8-bit two's complement values, in a fixed-point representation where they are scaled by a factor of 256. The values are stored in two ROMs; the twiddle factor generator selects the appropriate address line for each ROM and passes the output through.

A twiddle factor generator capable of generating 512 twiddle factors was used in the implementation of the 1024-point FFT. Two additional generators, with 8 and 16 points respectively, were also created for test purposes.

2.4.7 Twiddle Factor ROMs

Two ROMs are used to store the values of the twiddle factors. These ROMs are implemented using the FPGA's onboard memory, using the Altera LPM_ROM parameterized module. Each ROM (a sine rom and a cosine rom, used for the imaginary and real components of the twiddle factors) is a 512-location by 8-bit unclocked ROM.

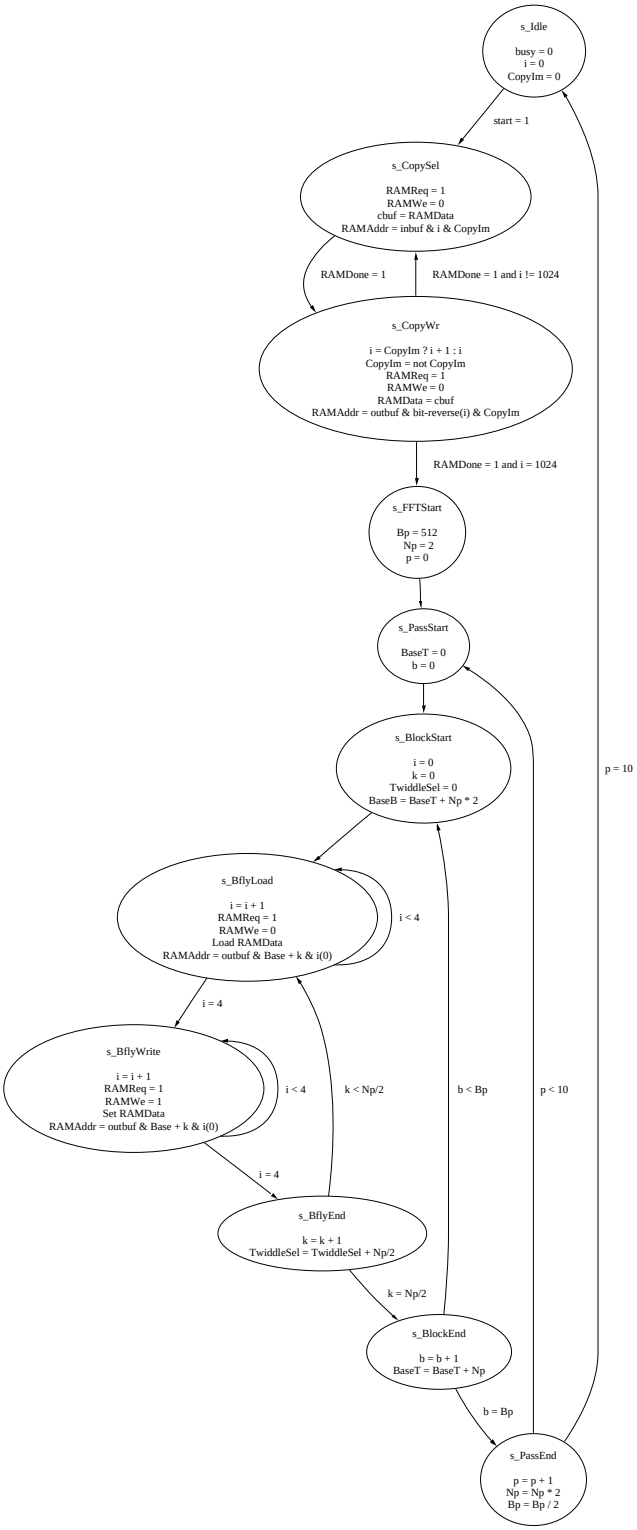

Figure 2.9: FFT FSM State Diagram

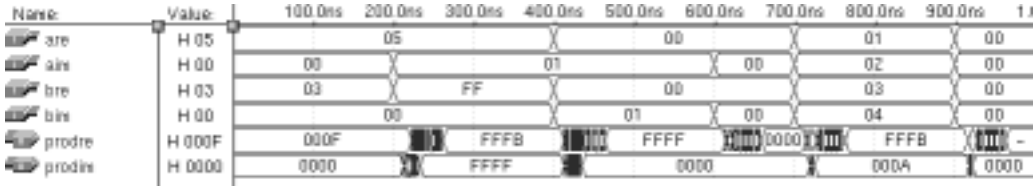

Figure 2.10: Complex Multiplier Timing Diagram

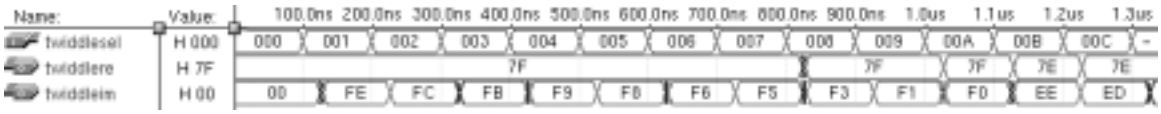

Figure 2.11: Twiddle Factor Generator Timing Diagram

The values for each ROM were generated by a C program whose source code is available in Appendix A.1.24.

2.5 Parallel Interface Subsystem

The parallel interface subsystem handles the task of exchanging data between this module and the Processing and Video modules. When directed to do so by the Control FSM, it transmits one set of samples from an input buffer to the other two modules, then receives another set of samples from the Processing module into an output buffer.

2.5.1 Interface FSM

The Interface FSM implements the parallel interface, handling the tasks of handshaking with the other two modules via the control lines, as well as transmitting and receiving the appropriate data. It is controlled by the Control FSM, and starts transmitting and receiving one block of information when it receives the Start signal.

The state transitions are shown in the state diagram in Figure 2.12. It implements the protocol described in Section 1.4.

The FSM begins in the s_Idle state, and waits for a Start pulse from the Control FSM. In the s_SendStart state, it asserts the FFTSending signal. Through the s_SendAddrSelect and s_SendRAMWait states, it loads a value from the RAM. In the s_SendReady state, the FSM then outputs the value on the data bus and asserts the FFTRdyAck line in order to indicate that the data is ready. It waits until both the ProcRdyAck and Ack2 lines have been set high, indicating that the other modules have received and acknowledged the data, then moves to the s_SendAck stage, setting the FFTRdyAck line low again. When the other two modules set their acknowledge lines low again, indicating that they are ready for the next data, the FSM returns to the s_SendAddrSelect state. If the counter overflows and reaches 0 again, then sending is complete, and the FSM moves to the s_SendComplete state.

In the s_SendComplete state, the FFTSending line is set low, indicating that the FFT has finished its transmission, and the FSM waits for the Processing module to begin its transmission cycle by setting ProcSending high. When this signal is received, it moves to the s_RecvWait state. In this state, the FSM reads the data bus when the ProcRdyAck signal goes high, then moves to the s_WriteAddrSelect and s_Write states, writing the appropriate value to RAM. It then moves to the

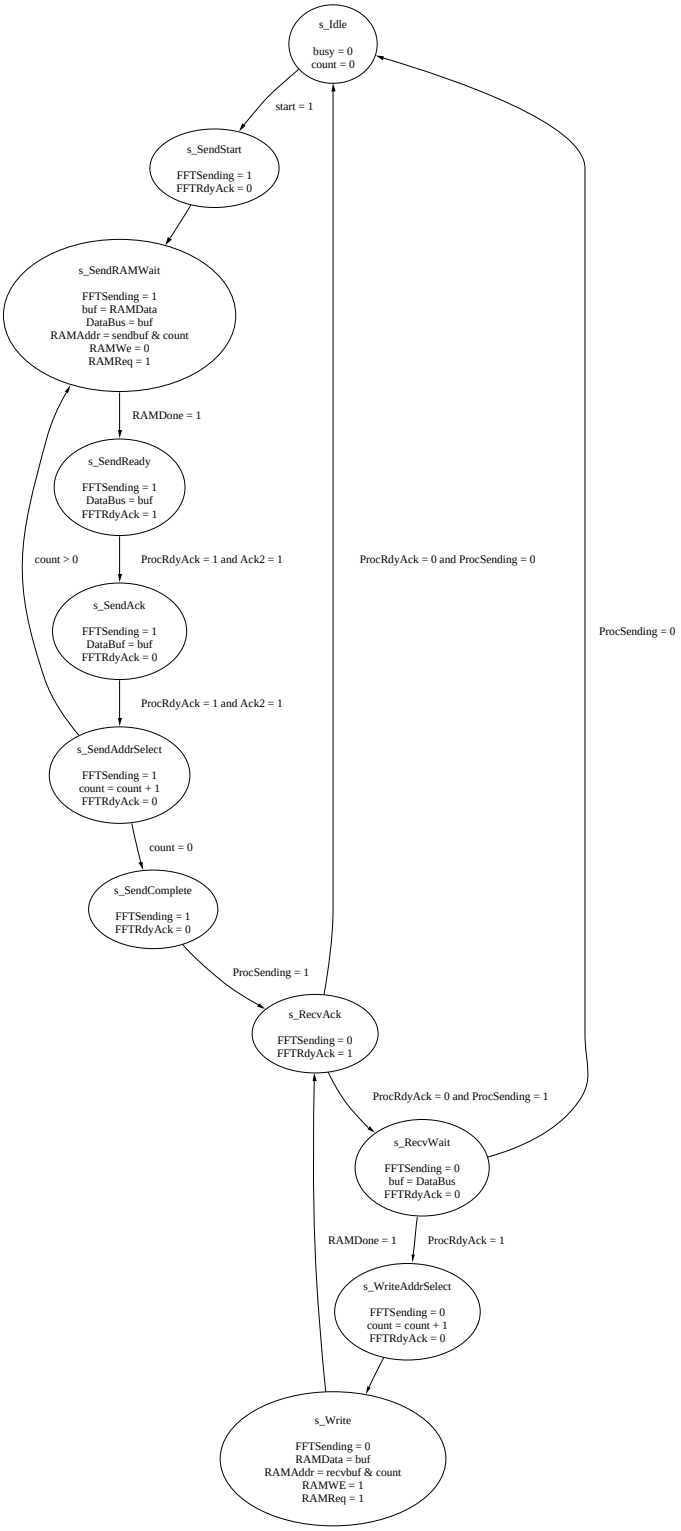

Figure 2.12: Interface FSM State Diagram

s_RecvAck state, setting the FFTRdyAck signal high to acknowledge reception of the data, and waits for ProcRdyAck to be set low before moving to s_RecvWait and waiting for the next data to become available. If instead ProcSending is set low again, the FSM detects that the Processing module has finished transmitting, and it returns to the s_Idle state.

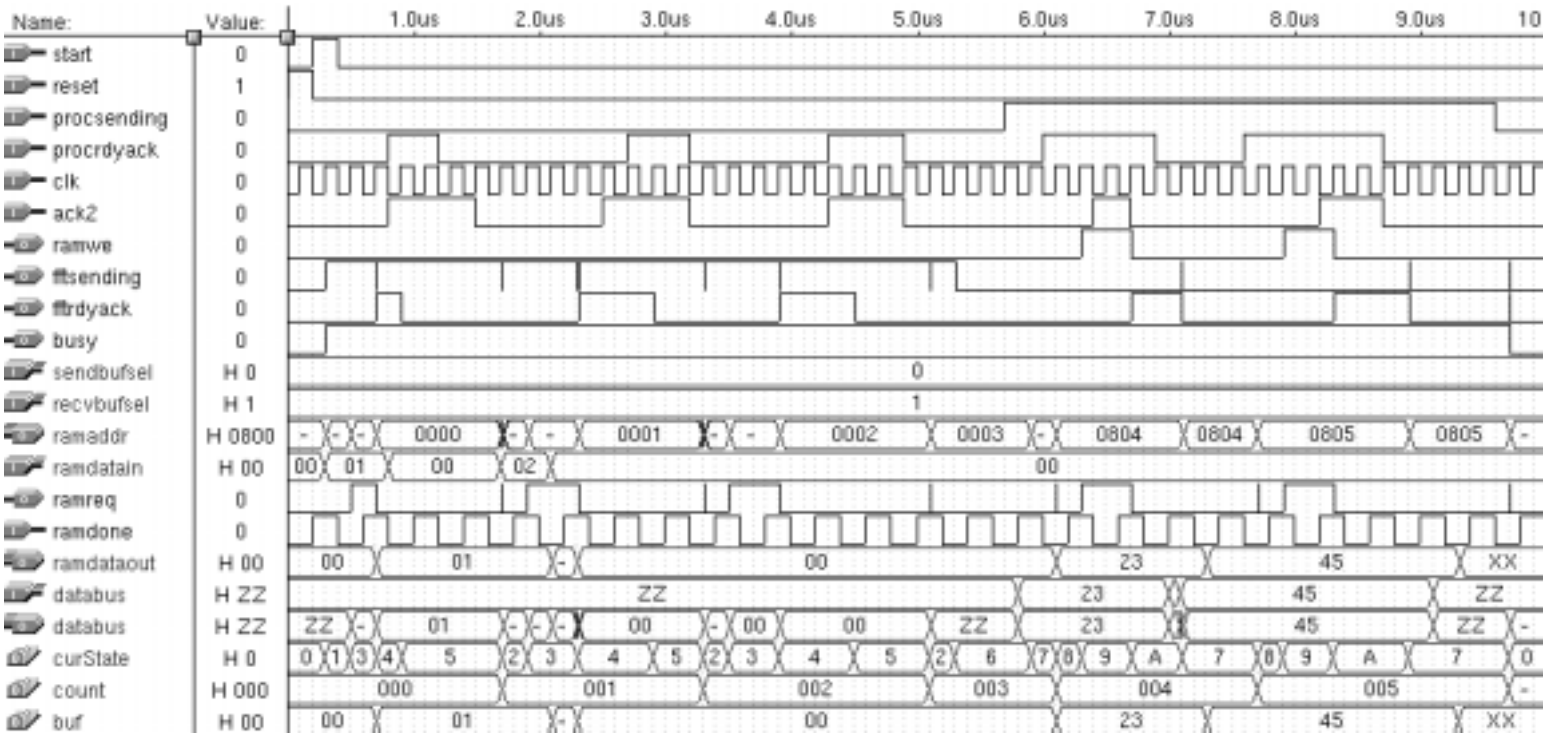

Figure 2.13: Interface FSM Timing Diagram

2.5.2 Synchronizer

The input signals to the module from the other modules are asynchronous with respect to the local system clock. A synchronizer is used to generate synchronized versions of the The synchonized output is guaranteed not to change except on rising edges of the clock.

The implementation of the synchronizer is simply a D flip-flop, as in Figure 2.14.

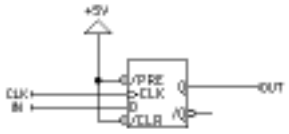

Figure 2.14: Synchronizer logic diagram

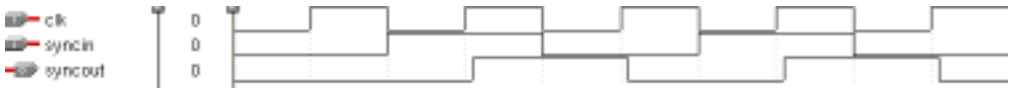

Figure 2.15: Synchronizer timing diagram

2.5.3 Line Drivers / Receivers

To isolate the FPGA from the transmission lines, line drivers and line receivers were used. These were realized using 74LS244 and 74LS245 chips. Schmitt triggering of the line receivers provided hysteresis, which was useful in minimizing transmission line effects. These are depicted in the wiring diagram in Figure 2.2.

2.6 Testing and Debugging

All components were tested first in simulation using MAX+PLUS II software, then in hardware when the FFT was programmed.

2.6.1 Simulation

Testing of most components was relatively straightforward: for numeric components, a representative sample of test inputs was provided, and the correct results were verified. To test FSMs, input signals were provided that placed the FSMs into each of their states. The correct outputs and internal state variables were observed. All possible states and state transitions were validated. Correct output was observed.

Testing of the FFT was difficult, as a 1024-point FFT was too complex to simulate. Thus, the FFT FSM was modified to compute a 16-point FFT instead, which was considerably simpler to simulate. In addition, it was modified to access a smaller RAM. A test jig was constructed, which consisted of the FFT FSM (containing its subcomponents required for complex multiplication and twiddle factor generation), integrated with a 128 location by 8 bit RAM. This was used to test the FFT FSM, initially without the memory manager. After these tests were completed, a separate test jig was constructed consisting of the memory manager and RAM to test the memory manager,

and finally a test jig containing the FFT FSM, memory manager, and RAM. Extensive testing was performed with this configuration.

Several FFTs and inverse FFTs were performed in simulation, with mixed results. The order of butterflies was observed to be correct, and processing was otherwise satisfactory; however, the results were not always acceptable. Some amount of numeric error was inevitable, but some results were unacceptable. A demonstration of a FFT being performed, then an inverse FFT being performed on the results is presented in Figures 2.16, 2.17, 2.18, 2.19, 2.20, and 2.21. The input sequence provided consists of alternating values of 0x20 (real) and zero. Note that the results of the FFT are correct, and the inverse FFT returns the original sequence. There is, however, some observable error that causes the magnitude to be slightly reduced; this is due to the causes described in Section 2.4.2.

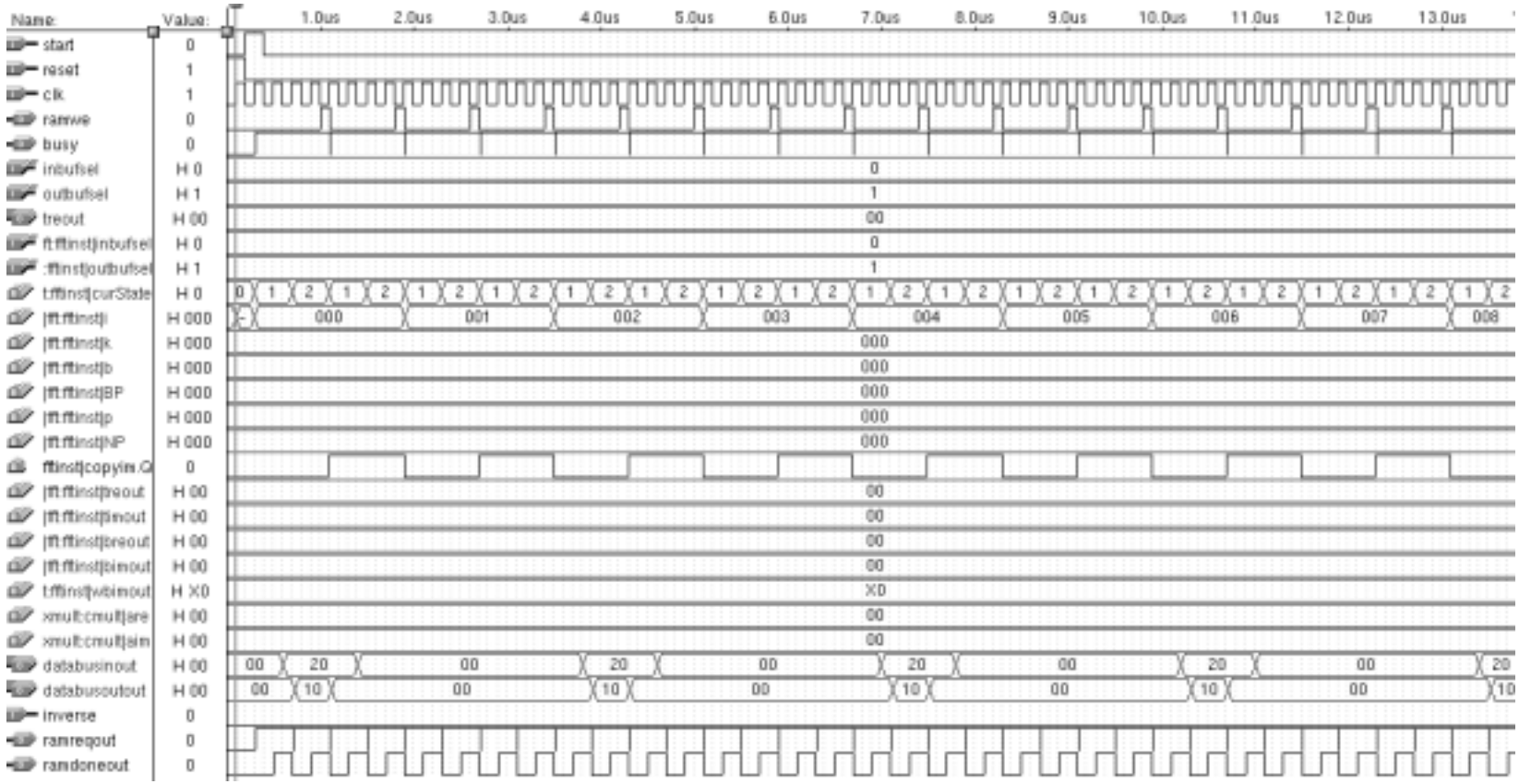

Figure 2.16: FFT Timing Diagram — start of copy phase

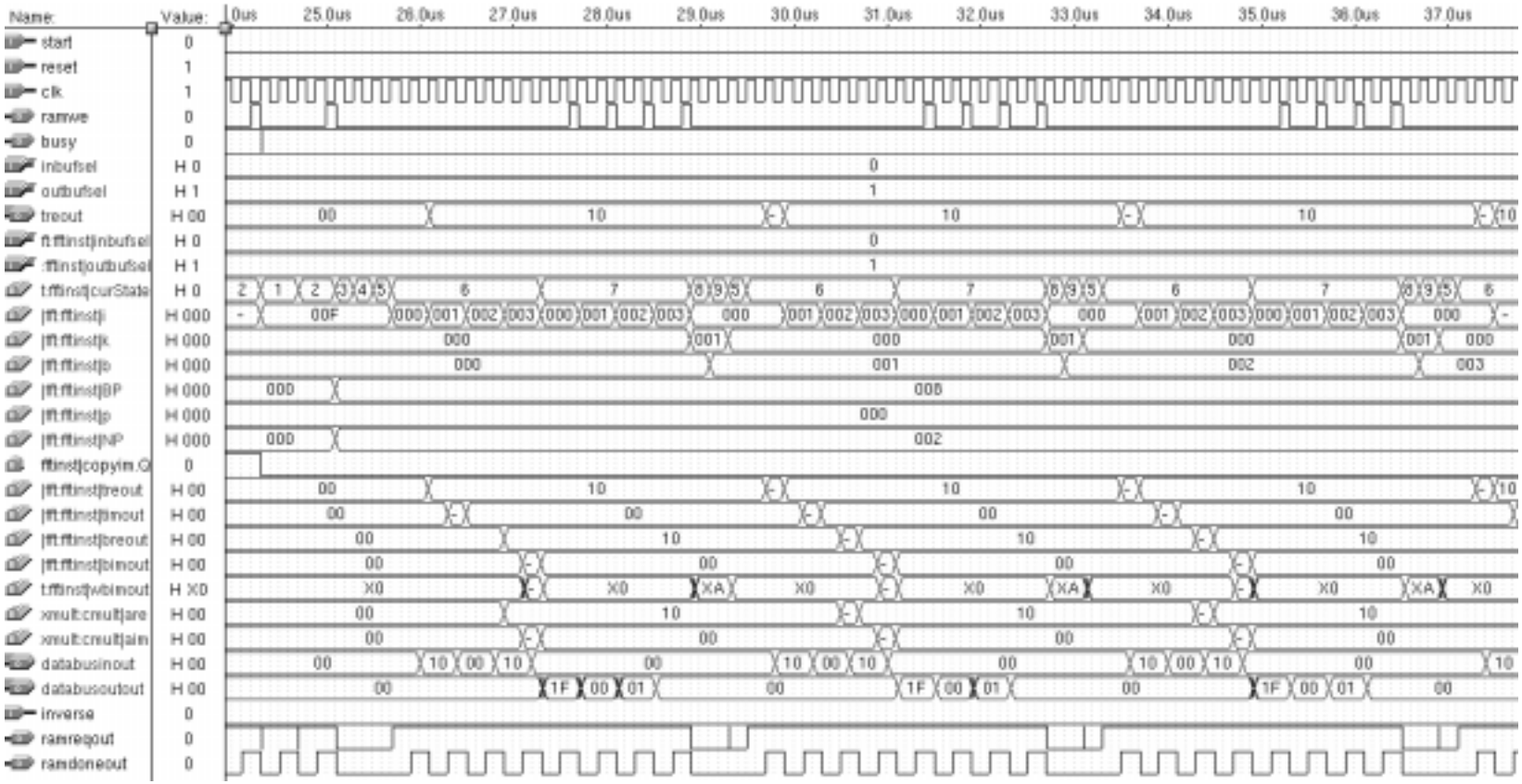

Figure 2.17: FFT Timing Diagram — start of FFT computation

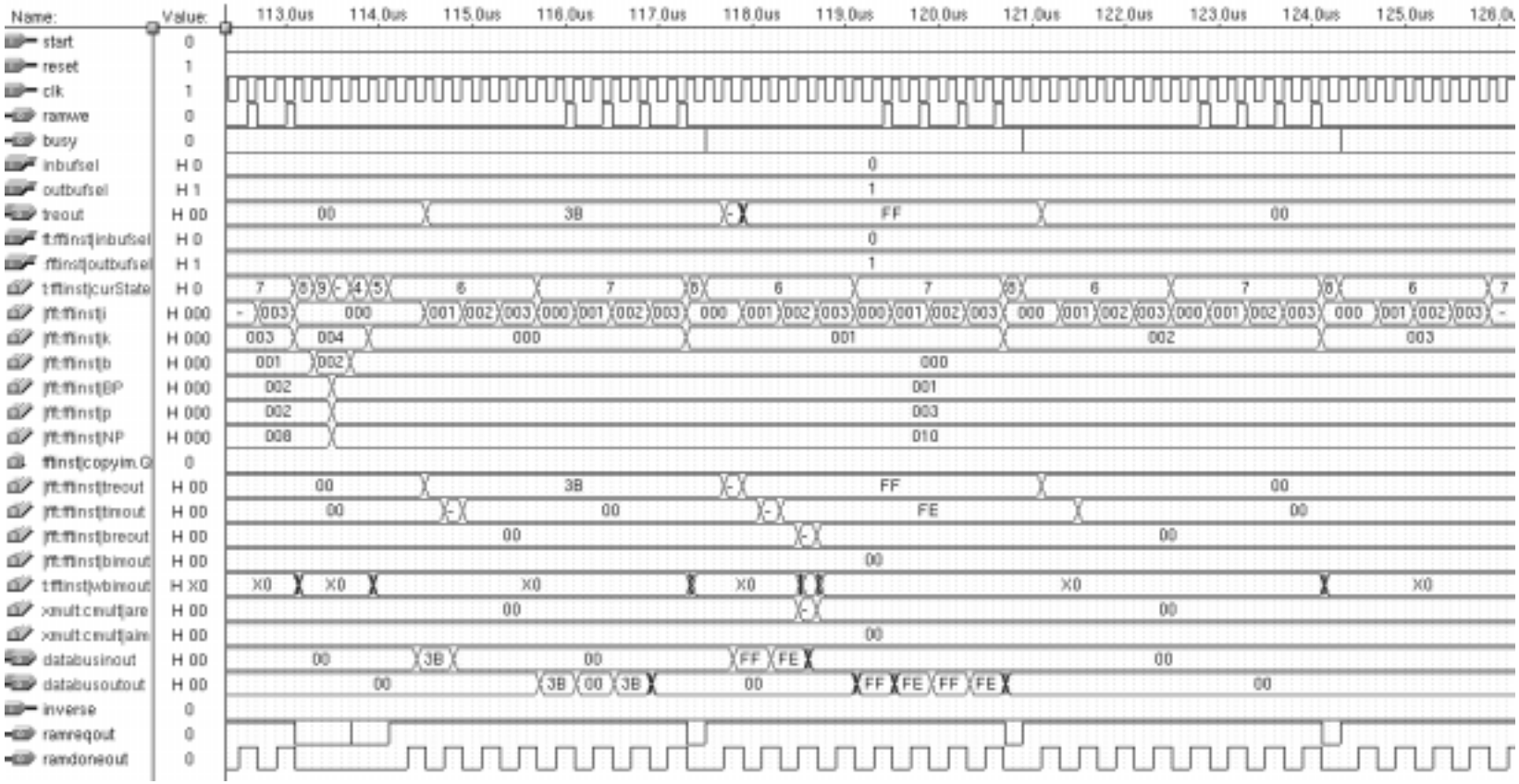

Figure 2.18: FFT Timing Diagram — first part of results

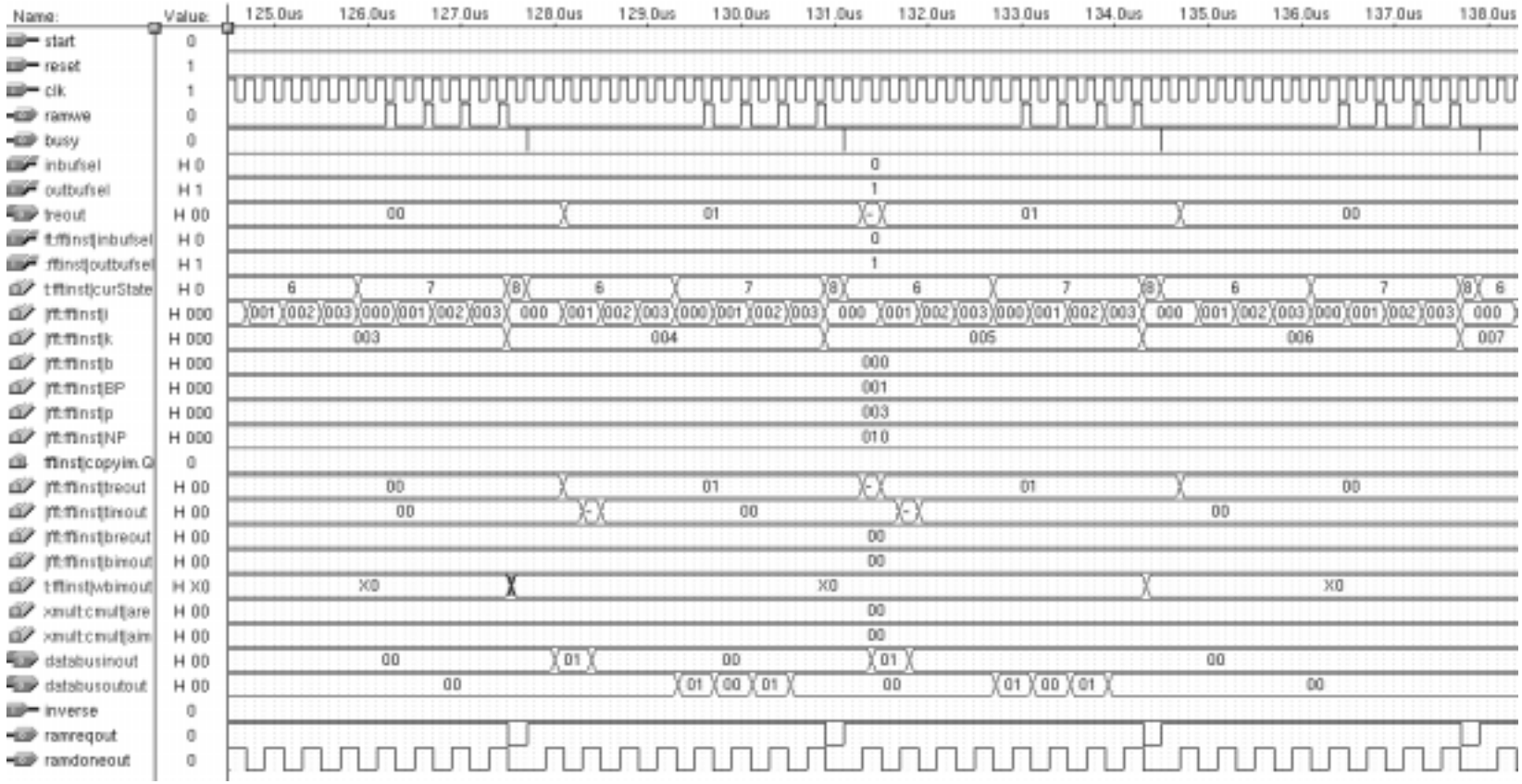

Figure 2.19: FFT Timing Diagram — second part of results

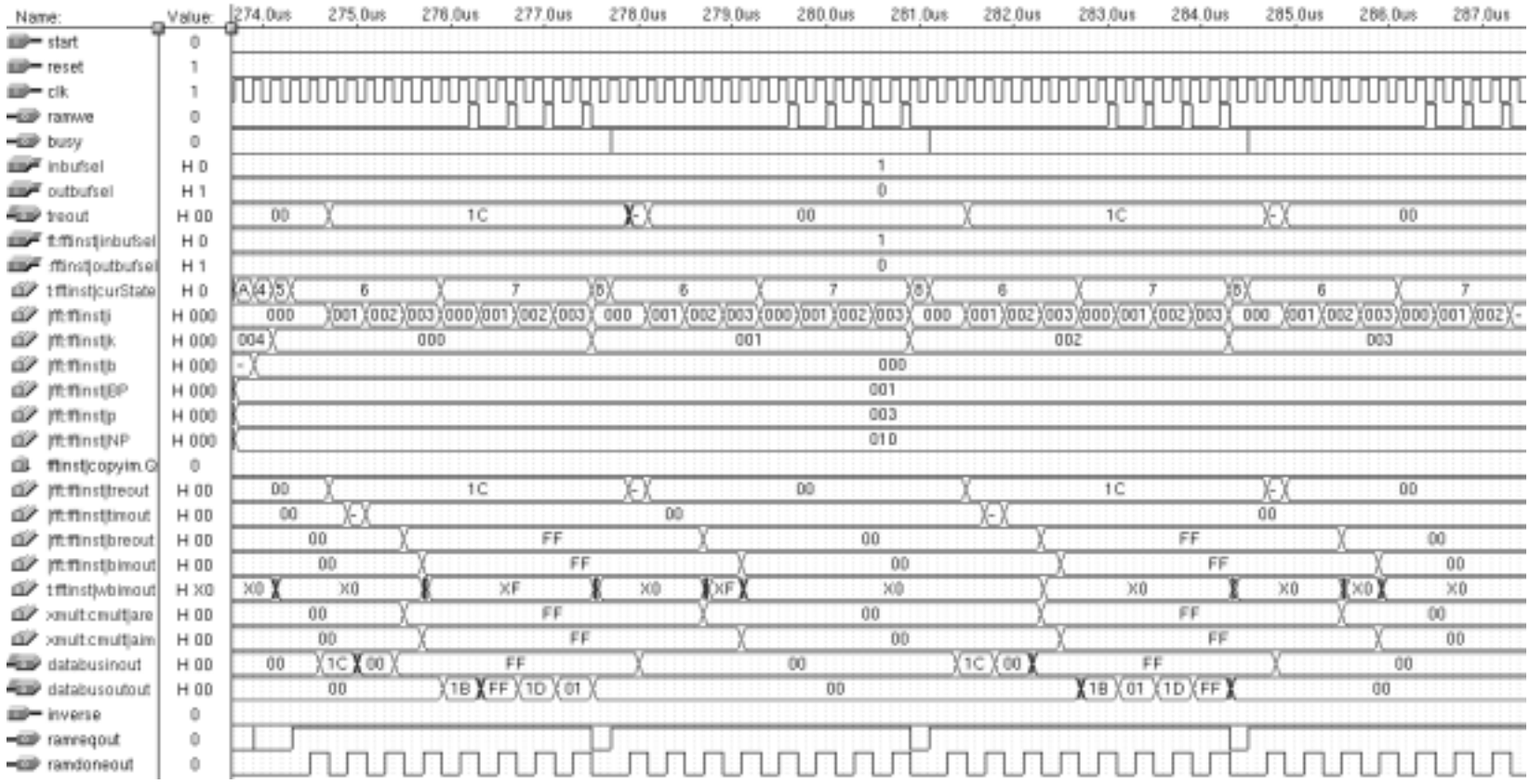

Figure 2.20: FFT Timing Diagram — first part of inverse FFT results

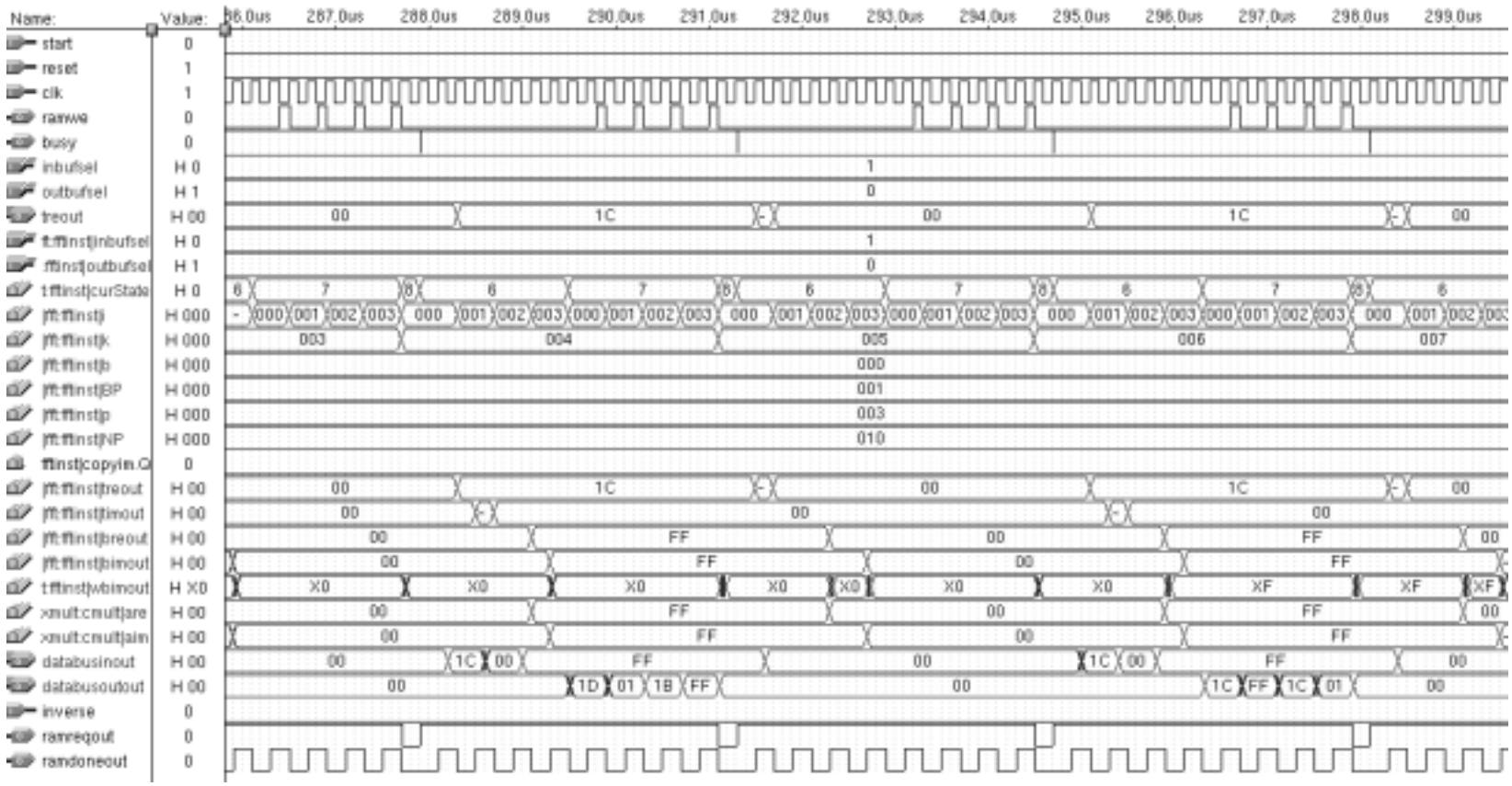

Figure 2.21: FFT Timing Diagram — second part of inverse FFT results

2.6.2 Hardware Testing

After testing in simulation was complete, the hardware was tested. First, a test program was programmed onto the 10K70 FPGA to ensure that the FPGA was operating correctly and all output pins were available. This test was completed successfully, though it was discovered that the L1-0 pin on the lab kit was not operating normally.

Next, the RAM module was tested. As some problems were observed with the original RAM chips selected (two Cypress CY7C185 8K $\times$ 8 chips), a test program for verifying RAM operation was programmed onto the FPGA. This test program wrote values to several RAM addresses, then read them back. A timing diagram is provided in Figure 2.22. The RAM address and data buses were monitored using the logic analyzer. Eventually the two RAM chips were replaced with a single Sharp LH52256C-70LL 32K $\times$ 8 chip. Correct RAM operation was observed.

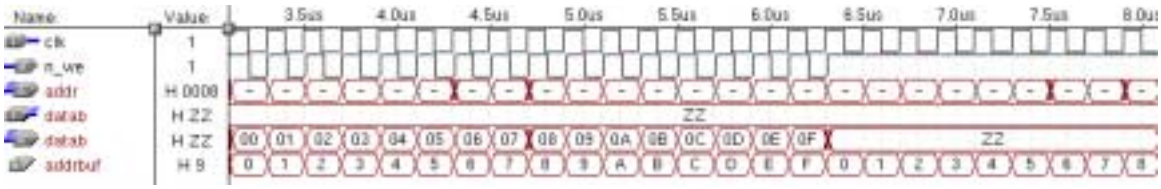

Figure 2.22: RAM Test Program Timing Diagram

The next stage of testing called for the FPGA to be programmed with the project code. Two test modes were provided in the Control FSM, and wired to the lab kit's switches. The first test was a test of the Analog / Digital subsystem. With this mode enabled, the FFT, inverse FFT, and interface were disabled. The Control FSM directed the A/D FSM to continuously sample input into a buffer, then output the input samples. The input was connected to a function generator and the output connected to an oscilloscope; correct operation was verified. Oscilloscope traces are provided in Figures 2.23 and 2.24. While this is not a particularly exciting output, it demonstrates the correct operation of a number of components: not merely the ADC, DAC, and the A/D Control FSM, but also the Control FSM, memory manager, and RAM, since the sample points must be buffered. The logic analyzer was used to verify that RAM accesses were occurring correctly.

Once analog/digital testing was completed, the Control FSM's FFT test mode was selected. In this mode, the Interface FSM was bypassed. The A/D FSM reads input samples into a buffer, which are then transformed by the FFT FSM. The results of the FFT are then passed through the inverse FFT, and output by the DAC. Thus, the output signal should approximate the input signal. Again, the function generator was used as an input, and the oscilloscope used to monitor the output. The logic analyzer was used to verify that RAM accesses were working correctly, and correctly being multiplexed between the FFT and A/D FSMs.

The results of this test were not satisfactory. An output was generated, but it did not reliably reproduce the original signal. However, it did bear some resemblance to the original signal, as in Figure 2.25, where the sinusoidal input generates an output that has a somewhat wavelike shape. This problem was likely due to the numeric error described in Section 2.4.2.

Next both test modes were disabled, and the output was observed on the logic analyzer. As expected, the module stalled because the Interface FSM was waiting for handshaking on the control lines of the parallel bus, and the other two modules were not connected. Debounced switches were wired to the control line inputs, and the correct outputs were observed on the logic analyzer.

The two other modules were then connected to the parallel bus. After resetting this module and the Processing module, data transmission was observed for a few moments. However, the two modules could not communicate reliably for more than a few hundred milliseconds. Afterwards, both the FFTSending and ProcSending lines of the bus were being held high, which is a forbidden

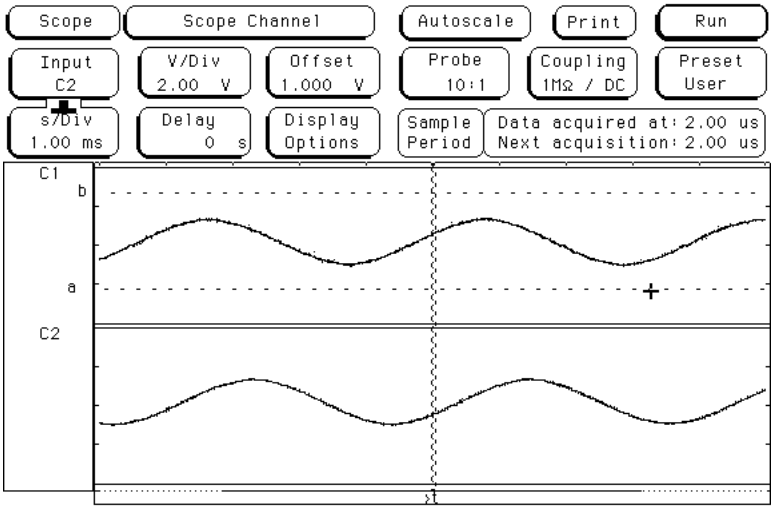

Figure 2.23: A/D Test Mode — sine wave input

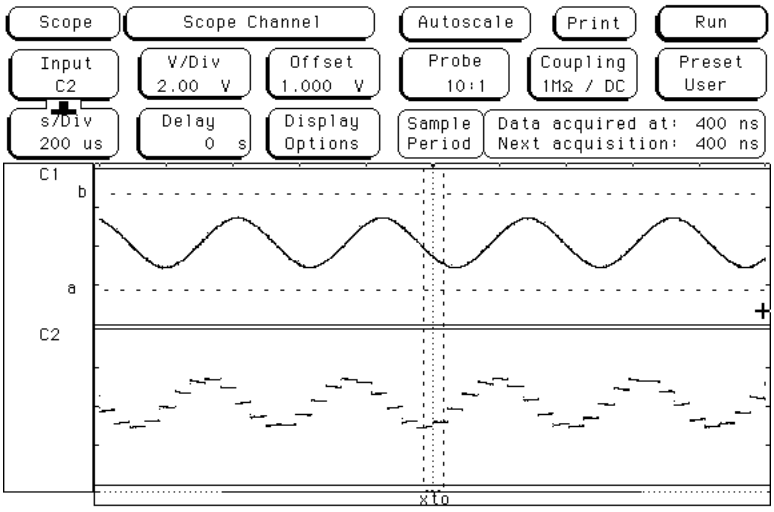

Figure 2.24: A/D Test Mode — sine wave input, demonstrating sampling effects

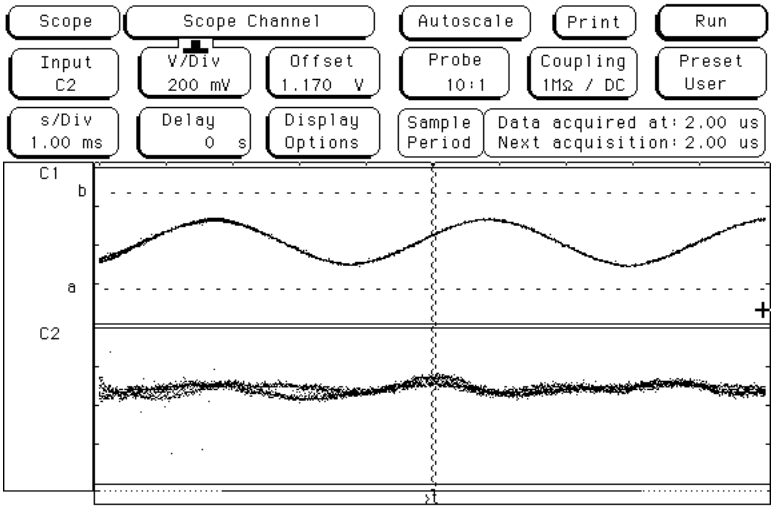

Figure 2.25: FFT Test Mode Input and Output

state. It was not clear what the cause of this deadlock was, and not enough time was available to debug it entirely. A probable cause is that transmission line effects led to glitches that caused the two FSMs to become out of synchronization and caused each to believe that it was time for it to transmit.

Chapter 3

Processing Module (A. Clough)

3.1 Description

The job of the processor kit is to alter and rearrange the frequency domain signals passed to it by the conversion kit, before passing on the result to the video kit and back to the conversion kit. When the kits are started up the RESET button is pressed to tell the kit to wait for the conversion kit to begin passing the first set of time domain data. At the same time RESET is pressed, the mode of operation specified by the three Settings switches is entered, though this will have no effect until this kit begins to output its processed data. When FFTSENDING goes high, the processor kit begins to accept data from the conversion kit and to store it into RAM locations starting at 0 and going to 2047. The kit will only exit recieving mode if FFTSENDING goes low, and if more than 2048 pieces of data are sent the last RAM location will be overwritten as many times as it takes. Once FFTSENDING goes low, the kit goes to its data sending phase.

During the sending phase the processor kit sends out each of the 2048 new pieces of data, altered according to the mode of the kit. The values of each data piece are calculated immediatly before they are sent, but because each piece is independant of the others and because the time constraints are relativly low compared to the clock speed, this doesn't represent a problem. Before each piece of data is sent out, the RAM is accessed either once or twice and the data loaded into the Arithmetic component. There it is manipulated, before being output on the DataBus shared with the the RAM as the ACKPROC is set high.

A diagram of the processor kit is presented in Figure 3.1 The five main components of the kit, as well as the various datalines, are shown. The dotted line represents the boundary of the FPGA, with everything inside being implemented in VHDL, and everything outside being in hardware.

KIT WIRING DIAGRAM GOES HERE

Figure 3.1: Processing Module Wiring Diagram

MODES DIAGRAM GOES HERE

Figure 3.2: Modes Diagram

3.1.1 Arithmetic

The Arithmetic section is where the inputs from the RAM are manipulated and output. Its operation is controlled by the inputs Add, Go, and Stop, as well as its connection with the multiplier. Each time Add goes high, Arithmetic takes a value from the DataBus, on which the RAM ought to be outputting. When Go goes high, the value stored in RAM is output onto the DataBus for the other kits. Finally, when Stop goes high the current value is erased.

There are three modes of operation. In the first, the input Sharpen is low, and Add only pulses once before Go and then Stop are activated. In this case Arithmetic simply graps the value being output by the RAM and returns it to the DataBus without altering it. If Add is pulsed twice, Arithmetic will first grab the value on the DataBus, then use Mult to multiply it by the Scale before adding it to the second value it gets from the DataBus, before outputing this sum. If the result is larger than the 8-bit output, it is simply rounded down to 11111111. In the first of the two following timing diagram Arithemtic activates first in the first mode, inputting 40 and outputting 40. Second it activates in the other mode, inputting 20, multiplying that by the Scale (01000000 or $\frac{1}{2}$), inputting and adding 40, and returning 50.

If Sharpen is high, then Arithmetic simply takes the most recent value, checks to see if its magnitude in two's complement is greater than the scale, and if it is returns the value. If it is less than then scale, a magnitude of zero is returned. In the second of the two following timing diagrams, you can see two positive and two negative numbers checked against the scale.

TIMING DIAGRAM DIAGRAM GOES HERE

Figure 3.3: Arithmetic Timing Diagram

KIT WIRING DIAGRAM GOES HERE

Figure 3.4: Arithmetic Timing Diagram

3.1.2 Control

The job of the control seciton of the kit is to translate the input Settings into numeric values that can be used to actually control the rest of the kit in such a way that the right result occurs. The AFactor and the BFactor are what the addresses of a given output are multiplied by in order to find the RAM addresses that the inputs will be taken from. For instance, if the kit is in the harmonic mode, the address of the first value to be given to Arithmetic will be multiplied by the BFactor, 01010110, and the address of the second value fed to Arithmetic will be multiplied by the AFactor, 10000000. The scale factor is also determined here, and in this case it would be 10000000. If only one number is to be used by Arithmetic, then BFactor is left as 00000000.

3.1.3 MassiveFSM

This is the most complicated part of the processor kit. This FSM has three functions: it synchronizes with the other kits on the parallel interface, it controls the activation and addressing of the RAM, and it controls the actions of the Arithmetic module. In naming the modes, those that begin with an R are entered when the FSM is recieving data, and those that begin with an S are entered when the FSM is sending data.

When the FSM starts up or RESET is high, the FSM goes into the RWaiting state. Here the count is initialized and it waits until FFTSENDING goes high so it can enter the RListen stage. In RListen the address is made the same as the count, and the FSM waits for ACKCON to go high so it can go to RPull and load the value on the DataBus. Alternativly, if FFTSENDING goes to 0 when the FSM is in this state, it assumes that the transmission of data has stopped, and goes to SWaiting.

In RPull Pull dips low, causing the RAM to load the value on the DataBus into its current memory address. After RPull the FSM goes to RAck where it sets ACKPROC high to signal that it has recieved the data. Now hopefully ACKCON will go down, in which case the FSM will enter the state RIncrementing where it will incriment the count by 1 before returning to RListen.

When the conversion kit finishes and FFTSENDING goes low, the FSM switches to SWaiting. Here, as in all states starting with S, PROCSENDING is set high to let the other kits know that the process kit is sending processed data. After doublechecking that FFTSENDING is 0 and reinitializing the count the FSM shifts to SStarting. From here the kit will go on to SBAddressing if there is a BFactor, and otherwise skip to SAAddressing if BFactor is 0.

In SBAddressing the count is multiplied by the BFactor, then in SBSettling this value is loaded into the address of the RAM. In order to keep parity between the imaginary and real components of the signal, the lowest order bit of the address is just taken straight from the count. In SBSettling the FSM pauses for a short while, and lets RAMOut low to tell the RAM to output the value in the address onto the DataBus. When the time is up, the FSM Switches to SBAdding, where Add is set high for a cycle, before the FSM goes on to SAAddressing.

The SA cycle is just like the SB cycle, but after it ends the FSM goes to SSetuping, where Go is set high to tell Arithmetic to start outputting onto the DataBus. Next, SBroadcasting sets ACKPROC to 1 to tell the other kits that there is valid data on the DataBus, and waits for them to acknowledge. When they do, it goes to SIncrementing where the count in incremented and the FSM decides whether it has sent enough data to say that it has finished sending and ought to go to RWaiting, or whether it has more to go and should go back to SStarting.

FSM DIAGRAM GOES HERE

Figure 3.5: FSM Diagram

The following are three timing diagrams of the FSM in action. The first one takes place while the processor kit is receiving data, and it responds to ACKCON, telling the RAM to load data and incrementing the address. In the second diagram the FSM is in sending mode. In this case BFactor is 00000000, so it only pulses Add once for each cycle, and increasing the address one by one. In the third diagram the BFactor is 11111111, twice as high as the AFactor. In this diagram, the way the FSM uses a different address for each Add pulse is clearly visible.

FSM TIMING DIAGRAM GOES HERE

Figure 3.6: FSM Timing Diagram

FSM TIMING DIAGRAM GOES HERE

Figure 3.7: FSM Diagram

FSM TIMING DIAGRAM GOES HERE

Figure 3.8: FSM Timing Diagram

3.2 Testing and Debugging

Certainly the largest problem with debugging the process kit was discovering that only a small fraction of the kits NuBus worked. Because some of the NuBus pins were stuck high, some stuck low, and some indeterminate, it was difficult to isolate which were working and which were not. Originally this kit was to have a knob which could be used to specify the exact degree of frequency shift or strength of the harmonic or such. Due to a tightly limited number of pins, though, this and a few other features had to be cut out of the design.

The fact that the state transitions were time-insensitive meant that I could hook up ACKCON and ACKVID to a switch and manually simulate the other kits. This greatly aided debugging, and allowed me to verify the correct opperation of my kit without needing to interface with the other kits. Since it could transition from sending to receiving by setting FFTSENDING to 0 and from recieving to sending by hitting RESET, any state can be easily reached.

The other problem during debugging was a glitch that caused the FSM to switch states inappropriatly during a small percentage of a certain type of state transition. Eventually it turned out that by going into a state where the next state was governed by an if statement, and then suffering from a transient which brought it through the wrong state on its way to the right state, it could sometimes get stuck in the wrong state. Fixing this problem turned out to be a case of rearranging the state listing so that the states were associated with different numbers and the problematic transitions wouldn't occur. Unfortunatly, the most recent rearrangement has caused another problem, making the count reset to 0 in roughly one out of every 200 transitions. Since this problem does not seem to occur when the kits are connected as opposed to the states being transitioned manually, it might be that this is an interaction between the switches and the nubus, which has RESET on it. A transitory triggering of RESET by shaking in the NuBus would be enough to account for the problem.

Though kit communication remains elusive, I have verified that values can be stored into sequential addresses in the RAM and retrieved; and that the correspondance between input and output can be altered by using the Settings switches.

Chapter 4

Video Module (A. Chiou)

4.1 Overview

This section of the report describes the operation and implementation of the video portion of the project. To generate the video signals, a Motorola MC6847 video display generator was used in External Alphanumeric graphics mode. The chip was controlled by a FLEX 10K70 FPGA, which also handled the formatting of data received on a 3-way 8-bit parallel bus. The video was displayed to an FTC-1201-R 12" color CRT display at a resolution of 192 pixels high by 256 pixels wide.

4.2 Operation

The job of the video module of this project is to display the input and output data, as well as useful operational data. The screen is divided into three sections. Along the right side is a sidebar which displays information on the current filter mode and variable settings. The rest of the screen is split into the top and bottom halves. The top half displays the frequency response of the input signal and the bottom half displays the frequency response of the output signal, which will have been processed by the selected filter.

The video information is stored in two 8Kx8 SRAMs which are muxed such that one is being written to while the other is being read from. The purpose of such a setup is so that one frame of information can be held for as long as necessary before switching to the next frame. This allows data processing and transfer to take more than the standard time allotted for a single frame in the NTSC standard, which the video display complies with. To prevent visible flickering, the frame is only changed during the vertical blanking period.

There are no direct user controls for the video module other than a reset button, which clears the state of the video FSMs. All control is dependent upon the data transmission cycles across the parallel BUS. Information on the screen is only updated when enough new information has been acquired, in particular, after the end of a full data transmission cycle. Because the job of the video module is to simply display the information it is given, there is no need for any elaborate control interface.

4.3 Design

4.3.1 Overview

A detailed circuit diagram of the video controller module can be seen in Figure 4.1. One thing of note is that while the MC6847 is clocked at 3.579545 MHz, all other synchronous components are

clocked to an external 10 MHz crystal oscillator. There is no need to worry about timing conflicts since the SRAM is asynchronous and each memory module is either being read from by the MC6847 (3.579545 MHz) or written to by the video FSMs (10 MHz). The analog portions of the circuit are clearly visible on the bottom half. Within the FPGA component are the three different FSM: the read FSM, the ram FSM, and the write FSM, as well as a dual-port memory buffer. These will be described later in this section.

The video display cycle is dependent on the flow of information across the parallel bus. The steps of the cycle are as follows. First, the unprocessed frequency samples are received from the parallel bus. After that, the processed frequency samples from the previous cycle are received across the parallel bus. Because the data is transmitted in the form of real and complex components, the magnitude must be calculated. For ease, we chose to display the square of the magnitude. The main reason for this choice is that it allows us to omit a complex square-root circuit, which would only take up precious space on the FPGA without adding any meaningful functionality to the video module. After the magnitudes are calculated, the samples must still be time averaged, since there are 1024 frequency samples and only 256 pixels across the width of the screen (only 192 pixels for the actually viewing area of the frequency response, since the right side of the screen is occupied by the "taskbar"). When proper values are finally calculated, the samples are written to SRAM in such a way as to properly display on the monitor when the SRAM is being read from.

The main components of the video module are the MC6847 video display generator, the two 8Kx8 SRAM modules, the monitor, the analog components, the dual-port memory buffer, and the three FSMs — the read FSM, the ram FSM, and the write FSM. These can all be seen in the simplified block diagram in Figure 4.2.

4.3.2 Clock

Aforementioned, there are two clocks in the video module. Because of the availability of the 10 MHz clock built into the project kits, we decided to use the 10 MHz signal as the standard clock. Each project module had its own clock, which affected our data communication protocol, as will be explained later. Specific to the video module is a 3.579545 MHz clock, which was created from a chain of inverters and a crystal. This clock is needed for the MC6847, which must run at that speed in order to produce an output compliant with NTSC standards.

4.3.3 Monitor

The monitor is a 12" CRT display capable of 8 or 16 color modes. For this project, however, we only use 2 colors (monochrome) displayed at a resolution of 256 dots by 192 dots. The monitor accepts as input intensity, red, green, blue, horizontal sync, and vertical sync. Since we are using only two colors, the red, green, and blue lines are all tied to the same line, which comes from the Y output after it is passed through a LM357 op-amp. The intensity line is left unconnected.

4.3.4 MC6847 Video Display Generator

The MC6847 Video Display Generator is used to generate the appropriate Hsync, Vsync, and RGB outputs to the monitor. Of the different modes available, we chose to use the external alphanumerics mode. With the external mode, we could create our own graphics by writing directly to the external character ROM. We did not chose a full graphics mode since we still wanted to retain the option of switching to the internal character mode to utilize the built in character ROM for displaying text. Because of switching time issues, we deemed that we would not even use any of the semigraphics modes that are somewhat compatible with the full text modes. The semigraphics mode divides up the normal 8 x 12 character blocks into smaller blocks which are colored depending on the values in external RAM. While the semigraphics modes offer the option of 4 or even 8 colors, we decided

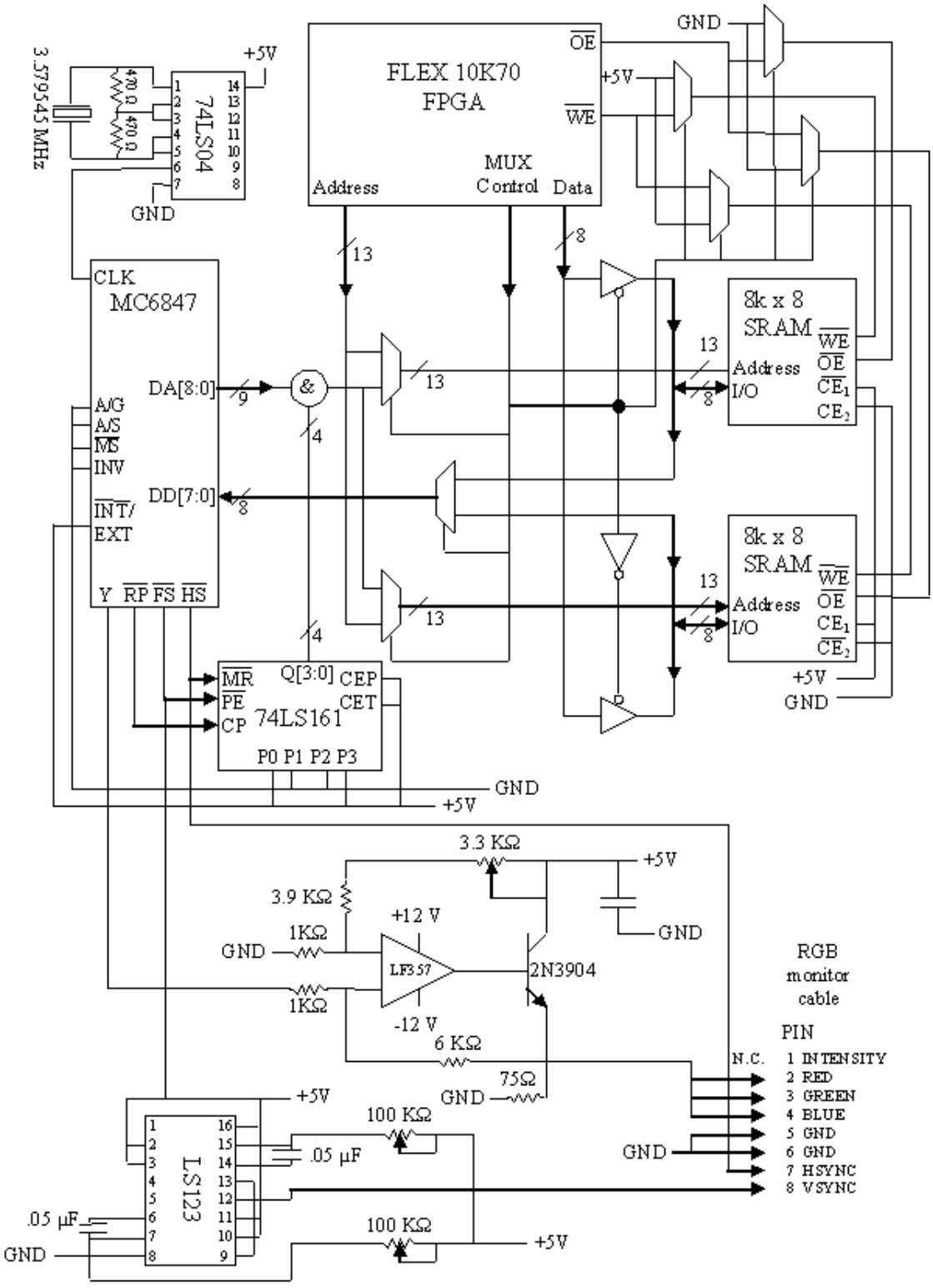

Figure 4.1: Detailed block diagram of the video module

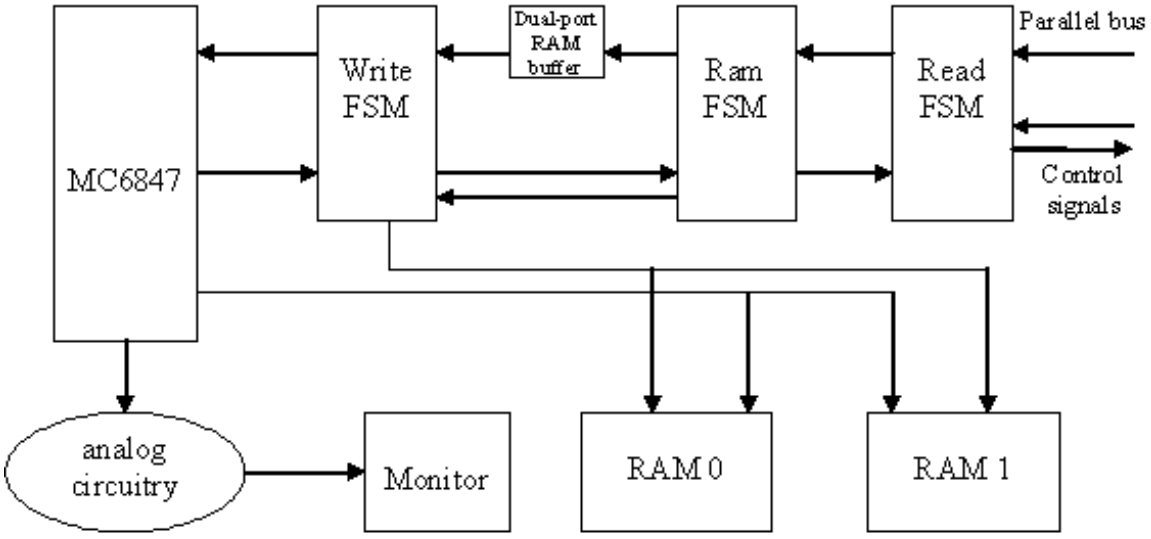

Figure 4.2: Highly simplified block diagram showing the relationship between the major components

that color was not important to our project, as the color would add nothing significant to the overall project.

The use of the external alphanumerics mode required that an external row counter be created using a 74LS161 chip. This chip would count from 0 to 11, and would consist of the bottom four address bits when accessing memory. Since the full space taken up by a character is 8 dots wide by 12 dots high, the counter is needed to count through the 12 rows of bits each. The upper addresses then correspond to locations of 8x12 blocks, while the bottom four bits correspond to the row within the 8x12 block. The upper address lines are supplied by the pins DA[8:0]. The MC6847 counts through these sequentially, allowing it to read the data in an orderly fashion.

The display itself will have two screen sections. The right screen, which will be 64 dots by 192 dots, will contain information, such as the current filter mode, and other important data, such as the value and name the knobs that set the filter parameters. The left screen will by 192 dots by 192 dots. This screen will be split into a top and bottom section of equal size. The top section, a 96 x 192 rectangle, will display the frequency information in the form of a solid line graph. The bottom section will also show frequency information, but of the processed data.

4.3.5 Analog circuitry

The analog portion of the video module can be divided into three parts. The first part is the simple circuit used to make the 3.579545 MHz clock. A crystal and two inverters were placed in a loop, and then inverted once more (also parallel to each inverter is a resistor). The other two parts are the circuitry generating the RGB output and the circuitry generating the VSync. The HSync is generated directly from the MC6847. The Y pin of the MC6847 contained the output that was required; we did not need the other output pins since we were working in monochrome, where a pixel is either on or off. The output on the Y pin is wired to an op-amp to adjust the voltage to a

range appropriate for the monitor. A 3.3k potentiometer provides a means of adjusting the threshold voltage, directly affecting the intensity of the signal on the monitor. The MC6847 generates a vertical clearing pulse instead of a VSync signal, so a resetable one-shot is used to change the pulse into a VSync signal. Two 50k potentiometers allow control of the VSync to fine tune the picture.

4.3.6 Read FSM

The read FSM handles communication over the parallel bus as well as the conversion of the frequency samples from real and complex components into magnitude squared. The state diagram below in Figure 4.3 shows the operation of the read FSM.

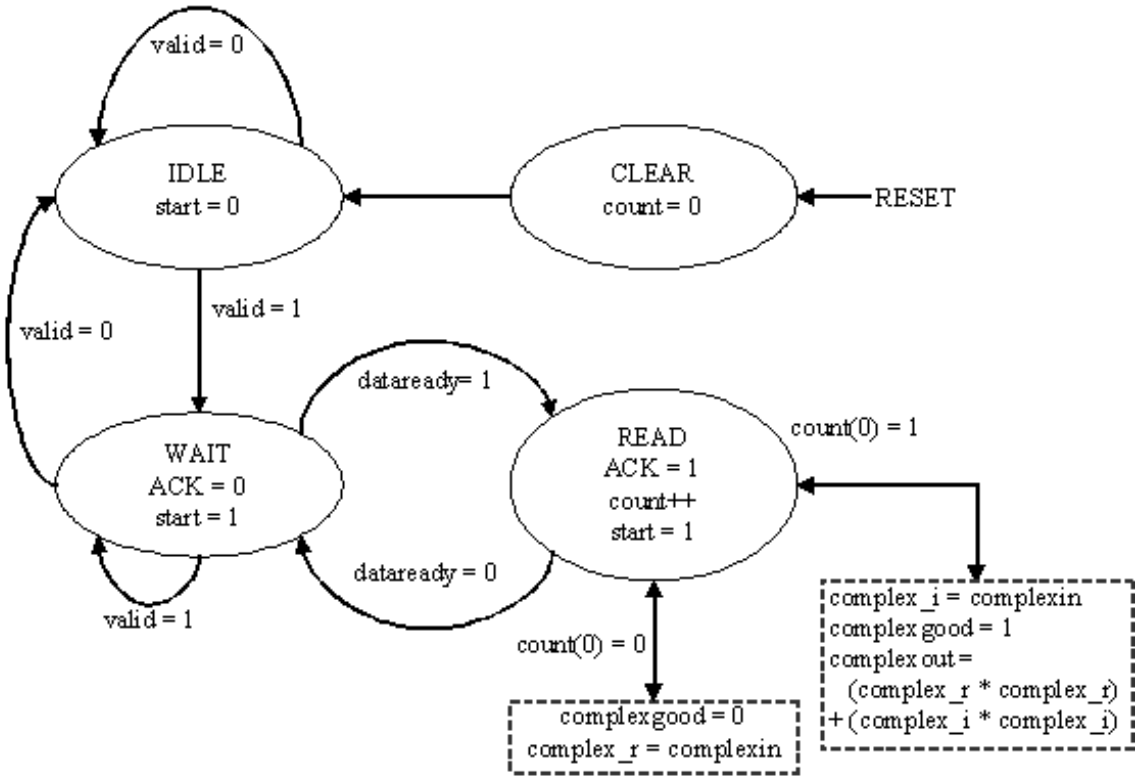

Figure 4.3: Read FSM state diagram

The FSM stays in an idle state until a valid signal goes high. The valid signal is an XOR of the FFTsend and PROCsend lines. When the FFTsend line is high, unprocessed frequency data is incoming from the FFT unit; when the PROCsend line, processed and filtered frequency data is incoming from the processing unit. An XOR was used to avoid the invalid state when both the FFTsend and PROCsend lines are high, which should never happen. The valid signal is high, then, whenever any data is incoming on the parallel bus.

After receiving a valid signal, the FSM will go into a wait state where it waits for a dataready signal to go high. When FFTsend is high, the FSM will consider FFTready to be the dataready signal; when the PROCsend line is high, the FSM will consider the PROCready line to be the dataready signal. Upon receiving a dataready signal, the FSM transitions to the read state, where

it stores the data, increments a counter, and acknowledges the a successful reception of data by sending ACK high. The FSM then waits for dataready to go low once more before setting the ACK line low and returning to the wait state.

In the read state, if the FSM sees the lowest bit of the counter to be 0, it will store the data as complex_r, since it will be a real component of a frequency sample. If the FSM sees that count(0) = 1, though, the FSM stores the incoming data at complex_i, and outputs the result of ((complex_r * complex_r) + (complex_i + complex_i)) on the complexout output line, which is connected to the ram FSM. This cycle repeats until all data is sent, when the PROCsend and FFT send lines both go low. The FSM then waits for about 16 cycles before going back to the idle state.

A start signal is also asserted during the wait and read state — basically whenever the FSM is active. This signal allows the ram FSM to know when the read FSM is active. The ram FSM will only operate if the start signal is high. The counter incremented in the read state is used to count the data samples. Because of this, one can tell when the information on frequency samples have all been sent, and when transmissions of other data have begun.

4.3.7 RAM FSM

The ram FSM controls the preparation of the data to be written into RAM. Namely, it controls the organization of the data into groups, and the control of the external RAM modules. Because there are only 192 dots wide in the display, and we want to leave room for things such as axes and labels, we need to somehow convert 1024 samples into a set of smaller samples to be displayed. For this, I decided that a simple time average would be appropriate, as it maintains the general shape of the signal better than a straight downsample (i.e. taking every nth sample).

Accounting for a one column (one column is 8 pixels) margin on the right and a two column margin on the left, this leaves 192 - (3 * 8) = 168 pixels for the display of 1024 samples. 1024 divides into 168 six times, with a remainder of 16, so that means that 6 * 168 = 1008 samples will be used, but 16 samples will be ignored. We decided that the last 16 samples should be ignored for a number of reasons. The first is that in the higher frequencies, octaves are spaced farther apart, so 16 samples at the upper end have a shorter range than 16 samples at the lower end of the frequency spectrum. Another choice would have been to toss out 16 equally spaced samples, but again, tossing out the higher frequency samples results in the loss of slightly less audible information. Performing the division in hardware is a potential problem which we will soon solve.

In addition to the width of the display, the height of the display is also a constraint. The full height of the visible screen is 192 pixels, divided up into two halves of 96. Accounting for a one row margin above and below each half (one row is 12 pixels), this leaves 96 (2 * 12) = 72 pixels for the display of frequency information. Since we are transmitting 8-bit real and imaginary components of frequency data, the largest numerical value of the magnitude of the frequency data is on the order of 17 bits (we square two 8-bit numbers and add them), or 2 * (256 * 256) = 131072. But we needed to keep track of the sum of six of these numbers together, since we were taking a time average, so we end up storing a 20-bit number. We still need to divide by six, but this can be solved concurrently with the problem of scaling. Noting that 72 = 9 * 8, we see that we must scale a power of two down to a number that is the product of a power of 2 and a power of 3. We can divide out the factors of two by bit masking, but the scaling of the factors of three could pose a problem. In order to be able to divide a 20 bit number down to 72 discrete values using bit masking, we need to multiply the factor of 9 to the 20 bit number. However, we still need to divide by six, so we can multiply by another factor of two, thus canceling out all divisions. The multiply by 9 becomes a multiply by 18, but when combined with the divide by six, reduces to a multiply by three. We can optimize further by simply adding a bit shifted version of the now 22-bit number, so a complete 22-bit multiplier is not synthesized. The appropriate number of lower bits can now be masked, scaling the 22 bit number to 72 and averaging the six samples at the same time. Shown below in Figure 4.4 is a flow diagram showing the operations performed on the frequency samples.

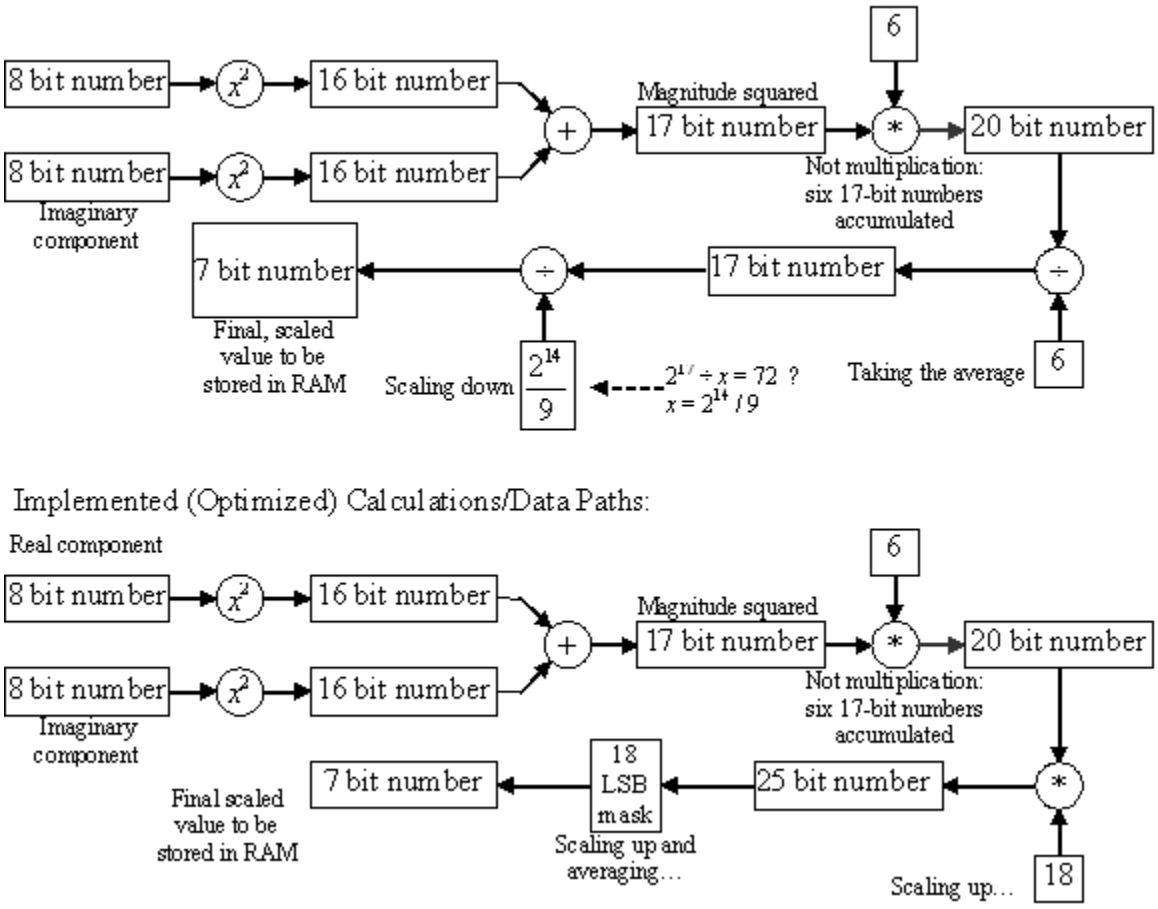

Figure 4.4: Operations converting 2 7-bit numbers into one 7-bit scaled value

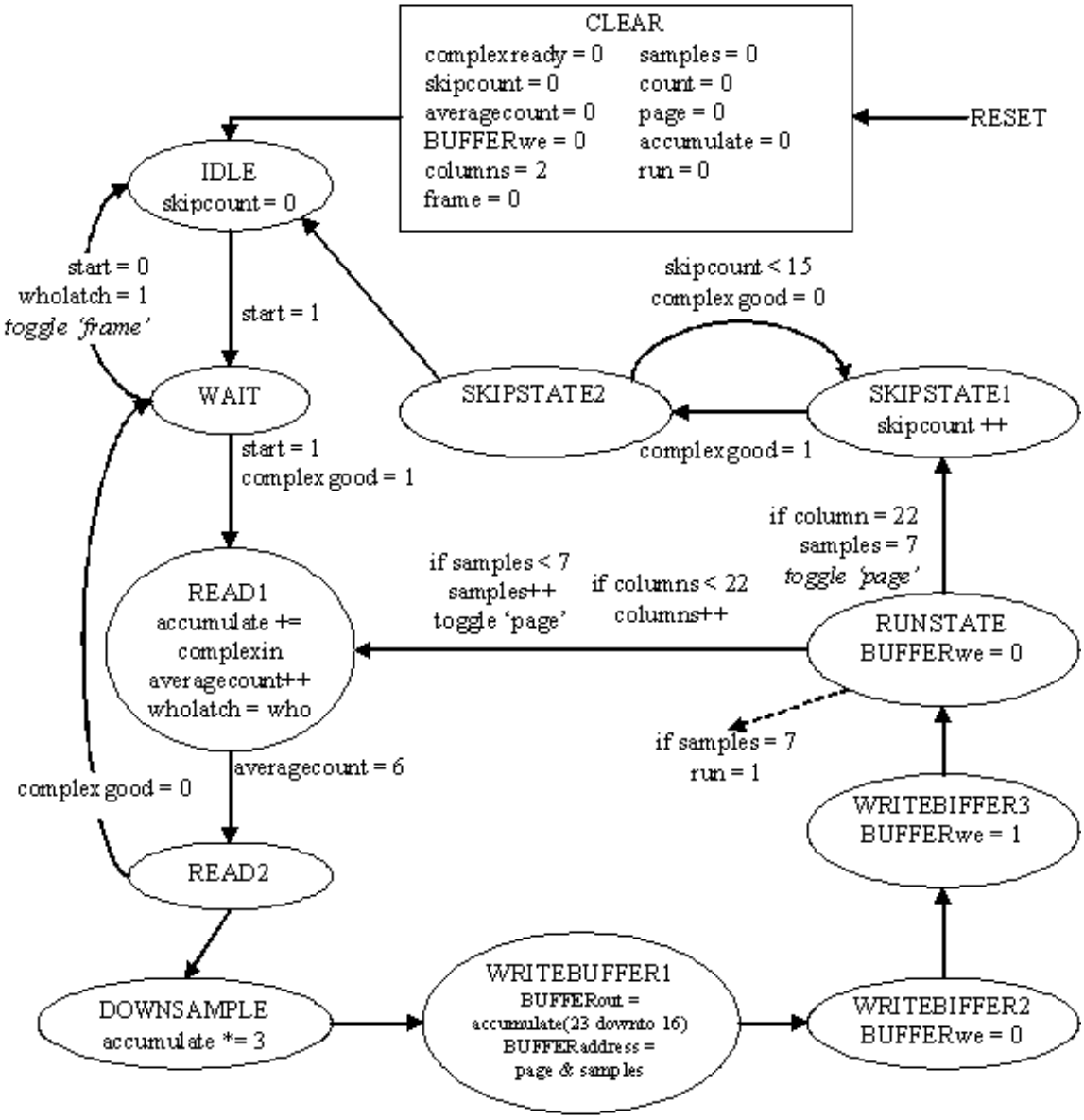

Figure 4.5: The state diagram for the RAM FSM

Because of the need to keep track of multiple counters, the ram FSM has a complicated state diagram, shown in Figure 4.5. The ram FSM starts in an idle state, and is brought into a wait state when the start signal from the read FSM goes high. In the wait state, the ram FSM waits for the complexgood signal to go high, which indicates that the read FSM has completed calculating a frequency sample. The ram FSM then enters the read1 state where it adds the sample to an accumulator, records the current id of the sending party in the wholatch, and increments the averagecount counter. The FSM then goes into the read2 state where it checks if averagecount equals six. If it does, the FSM continues on to the downsample state. Otherwise, it will wait for copmlexgood to go low before returning to the wait state, where it will wait for complexgood to go high once more. In the downsample state, the value in the accumulator, which is now holding the sum of six frequency samples, is multiplied by three. The FSM then goes into the Writebuffer1 state, where it sets the data output value and address of the information to be written to the dual-port RAM. Every new sample — the average of six frequency samples — is written to the dual-port RAM, and when eight samples have been written, the write FSM translates this into data to be written to the video memory. The dual-port memory is organized into two pages of 8 8-bit locations each. The MSB of the address signifies which page is to be used, while the three LSBs correspond to the order of the stored samples (000 being the oldest and 111 being the most recent). The page signal stores the page number while the sample signal stores the id number of the current sample.

After writing to the buffer, the ram FSM enters the runstate, where it sets run high if samples = 7 (note that when samples equals 7, eight samples have been written to the buffer). The run signal tells the write FSM that an entire page of the buffer has been written to and is ready for final processing. The FSM will also toggle the value of the page signal so the other page in the buffer is written to when the FSM collects the next samples. In addition, if the columns signal also equals 22, then the FSM is done receiving visible samples so it proceeds to the skipstates. In the skipstates, the next 16 new samples are ignored. By the time skipstate2 transitions to the idle state, 2048 pieces of data should have been sent across the parallel bus and received. If samples = 7 but columns < 22, then the columns signal is incremented, the samples signal is reset to 0, the page signal is toggled, and the FSM transitions to the read1 state, where it is ready to start writing data for the next set of 8 samples. Finally, if samples < 7, then samples is incremented and the FSM returns to the read1 state, ready to read the next sample from the read FSM.

On the screen, we are writing from columns 2 to 22, for a total of 21 columns. Every time 8 samples are written, that is enough data for an entire column to be displayed. So, when the column counter hits 22 and the sample counter hits 7, we know we have read in all the useful information for all the columns and will ignore the next 16 samples.

4.3.8 Write FSM

The write FSM is responsible for writing the correct values to the proper memory locations so the video displays correctly. Its main job consists of translating the eight samples stored in the buffer into data representing an entire column. Data for the video is written eight bits at a time, which corresponds to one of the rows within an 8x12 block. Each column of pixels is represented by a value in the buffer (the value in the buffer is the scaled magnitude of the frequency samples, which translates into the number of pixels that should be lit).

The data is converted by starting a counter at 72 and decrementing it, checking each column to see if it should be lit. The values stored in the buffer will be a number between 0 and 72, which designates how many pixels starting from 0 should be lit. The counter represents the highest column and also the lowest memory location. If the value in memory is greater than or equal to the counter value, then that means the pixel in that column should be lit; if it is less than the counter value, the pixel in that column should not be lit. The lower three address bits of the memory buffer correspond to the eight columns in the 8x12 character cell. Figure 4.6 below shows how the read data corresponds to video data.

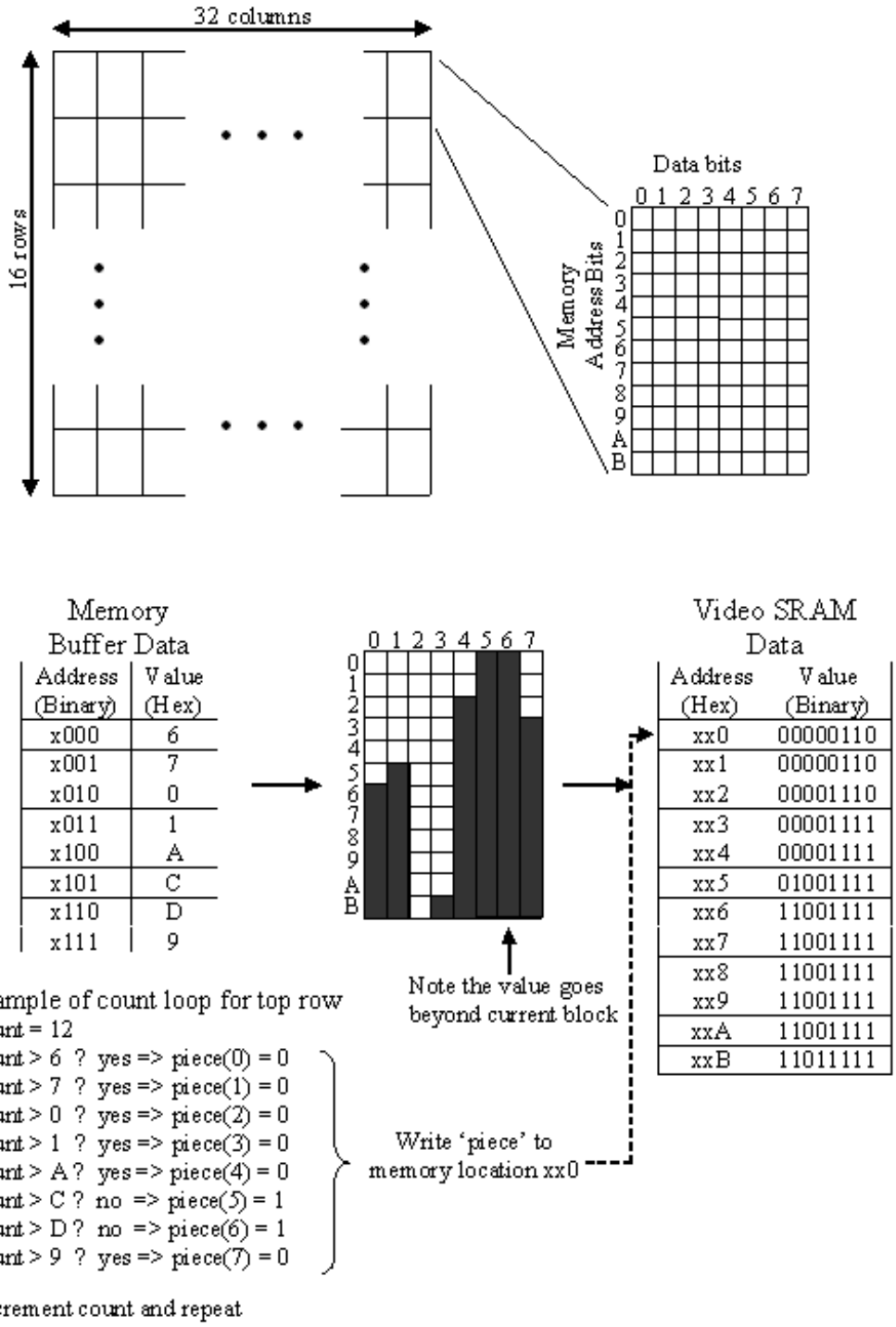

Figure 4.6: The conversion of "vertical data" (frequency sample magnitude) to "horizontal data" (video RAM memory values)

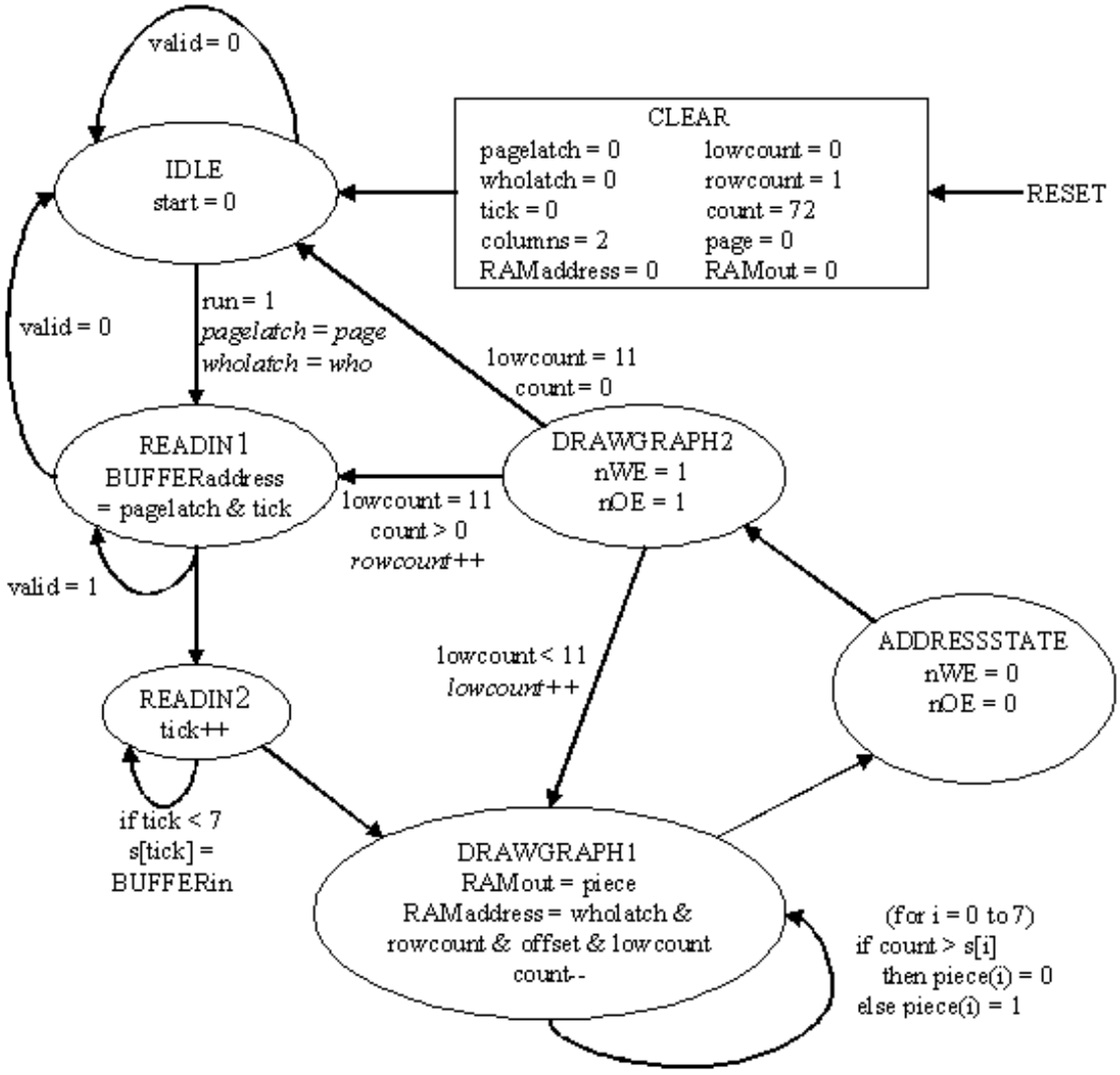

Figure 4.7: Write FSM state diagram

Figure 4.7 above shows the state diagram of the write FSM. It starts in an idle mode and is activated when the run signal from the ram FSM goes high. At this time, there are three pieces of information sent from the ram FSM to the write FSM: the who signal, which contains the id of the data sender, the page signal, which designates which memory page is to be read from, and the offset signal, which tells which column should be written to. The write FSM then goes to the readin state where it reads through the memory buffer page specified by the page signal, and stored the values locally. The FSM them transitions to the drawgraph1 state where it enters a loop that runs until count = 0 (counts is initialized to 72). In the loop, the count value is compared to the values read from the buffer. At each comparison, a one or a zero is generated, which is then stored in the signal piece, which is then written to video memory. Count is then decremented and the loop repeats. When count finally reaches 0, the FSM returns to the idle state and waits for the next run signal.

4.3.9 8Kx8 Video SRAM

The two 8kx8 video SRAMs used has access times on the order of tens of nanoseconds, which we thought was more appropriate than the available 6264, which had access time on the order of hundreds of nanoseconds. Our clock period being 100 ns, this allowed us to avoid having any wait states when accessing memory. The original design of the video module called for an 8Kx8 dual-port memory module, but unavailability of the chip prompted us to a rotating-buffer design, where the MC6847 alternates between reading the two memory modules. One of the benefits of using two separate memory chips is that the information on one frame can be held for as long as it takes to write new information to the other RAM. One of the drawbacks, however, is that information that does not change must still be written to the other memory.

The control signals to the memory had to be multiplexed to ensure that the FPGA had access to control one memory module while the other memory module was placed in read mode for the MC6847. The complex wiring did prove to be an obstacle, but painstaking tedium and proper clipping and routing of wires overcame it. To minimize the use of inverters on the MUX control signal, alternating MUXes were often cross-wired. The connection of the memory I/O ports to the FSM also had to be tristate buffered, since the memory is driving the data lines in read mode and the FSM is driving the data lines in write mode.

4.3.10 Dual-port memory buffer

The dual-port memory buffer has 16 8-bit locations, and is organized into 2 pages of 8-locations each. Each page corresponds to a column (8 pixel-widths) of data to be displayed in the frequency response graph. The dual-port memory was implemented on the FPGA as a latch array, as the FLEX family of FPGAs does not support true dual-port memory. The dual-port module is necessary because the ram FSM needs full write access to the buffer at all times, while the write FSM needs full read access to the buffer at all times. Because the memory was only 16 locations, the amount of space taken up on the FPGA was tolerable, when compared to the ease of operation that it brings.

4.4 Possible expansion

Among the many ideas for expansion were a ROM containing the video information for static portions of the screen, such as x- and y- axis labels, titles, and other asthetic designs. The possibility of switching from a full text to a full graphics mode was also considered, as this would allow a more detailed view of the spectrum, and with eye-pleasing graphical effects. At the height of our ambition, we considered a visualization system similar to those seen on the multimedia players on computers. We did not expect to be able to implement some of those ideas, but the brainstorming process produced many creative results that would be interesting to see, to say the least.

4.5 Testing and Debugging

The testing of the video came in two general phases: the testing of the analog portion and the testing of the digital portion. The analog portion was first tested using a simple test circuit wired to the MC6847, allowing the switches to manipulate the data shown on the monitor. This was largely successful, with only a few minor issues. Then next step as to test a digitally controlled version of the video display as well as the ability to switch from reading and writing to each of the memory modules. A simple FSM was created that would write to the screen based on the inputs from switches to the FSM. The MUX control line was wired to a switch, which allowed me to manually initiate a writing cycle, and then flip the switch to view the contents of the RAM. This ran into some problems that were largely the result of wiring. To resolve any doubts, I completely rewired the system, being sure not to strip too much insulation, which would risk shorts, and also being sure to make them the appropriate length.

A final video test was done to ensure that I understood the way the MC6847 was reading from memory, i.e. which memory locations corresponded to which 8x12 blocks on the screen. This test was done by simply writing in consecutive numbers 1, 2, 3, etc. to consecutive memory locations and observing the results on the screen. After this test, I found out that I had used horizontal addressing in designing my FSMs instead of what appeared to be vertical addressing. After making the change, the display still did not function properly. I had to perform the address test once more, and on the second time, I realized that horizontal addressing was used, but the MSB specified which side (left or right) that the information would be displayed on.

To test the FSMs, Max+Plus II software simulations were used. Below in Figure 4.8, the results of a top level 1.23 ms simulation can be seen.

The most frustrating portion of the debugging came from the need to explicitly state the values of every variable in every state. Oftentimes, when left unspecified, the compiler will decide to give the variable any value it pleases. Because of this, I had to create many latched signals, highly noticeable by the "x0 <= x" format that they take in my code. One of the more challenging portions of debugging was the attempt to circumvent the use of the division operator. Oftentimes it took some time to verify that the simulation was outputting the correct value, because the correct values were often confusing.

Of course by far the hardest part to debug was the communications subsection. Because of its nature, it was impossible to test without wiring all three kits together. And even so, it was hard to tell which kits were causing any problems. Extensive use of the logic analyzer allowed us to make some modifications to improve the communications interface, but it remains in a barely functional state before decaying into an invalid state, where two kits try to transmit data to each other at the same time. Of course spent much time pondering over these problems, as our simulations often looked fine, as in Figure 4.9 below. Also below are other simulation results of the other FSMs.

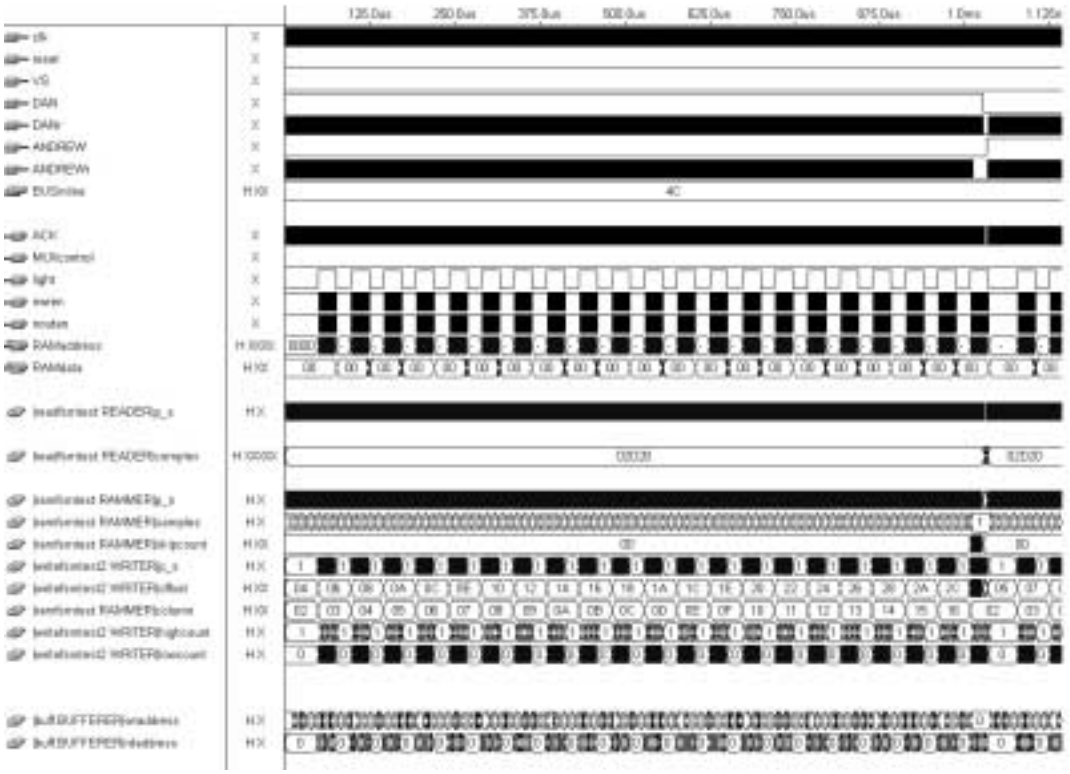

Figure 4.8: Top level simulation results. Notice how the values on "column" increase smoothly up until the skipping period.

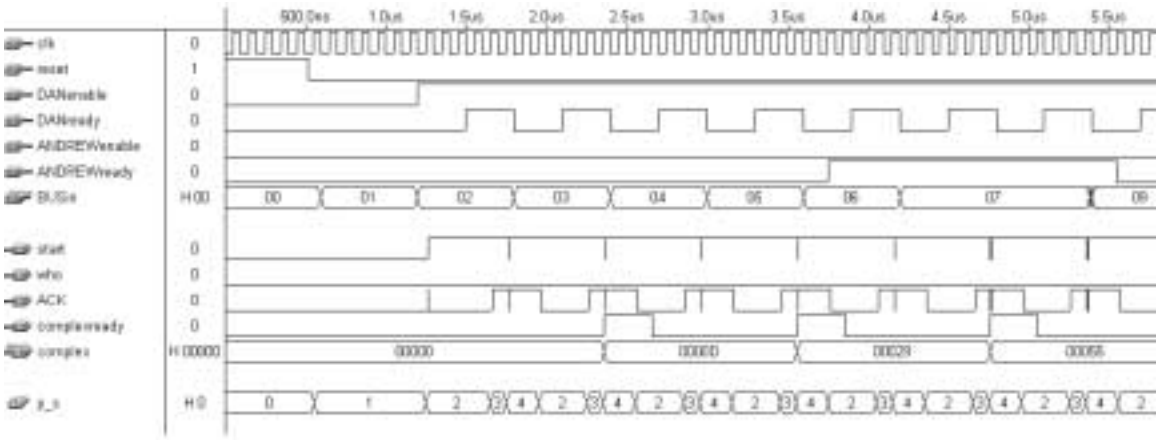

Figure 4.9: Simulation results of the read FSM. It acknowledges ready signals and reads the data properly.

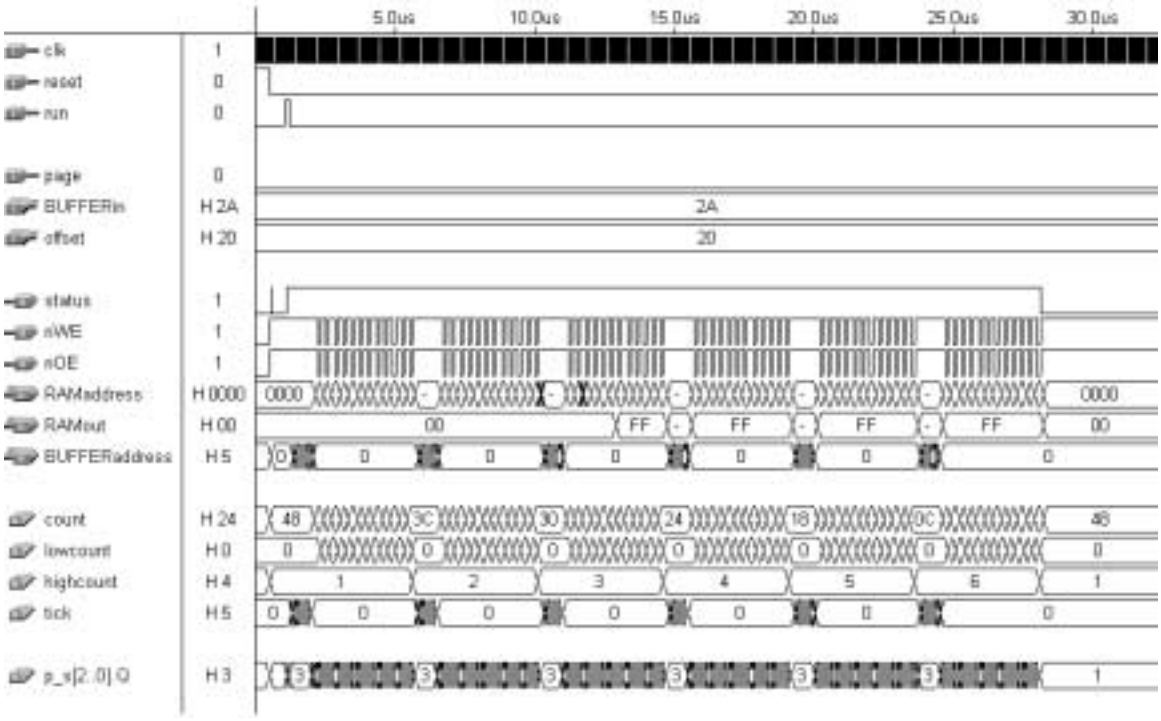

Figure 4.10: Simulation results of a full cycle of the write FSM.

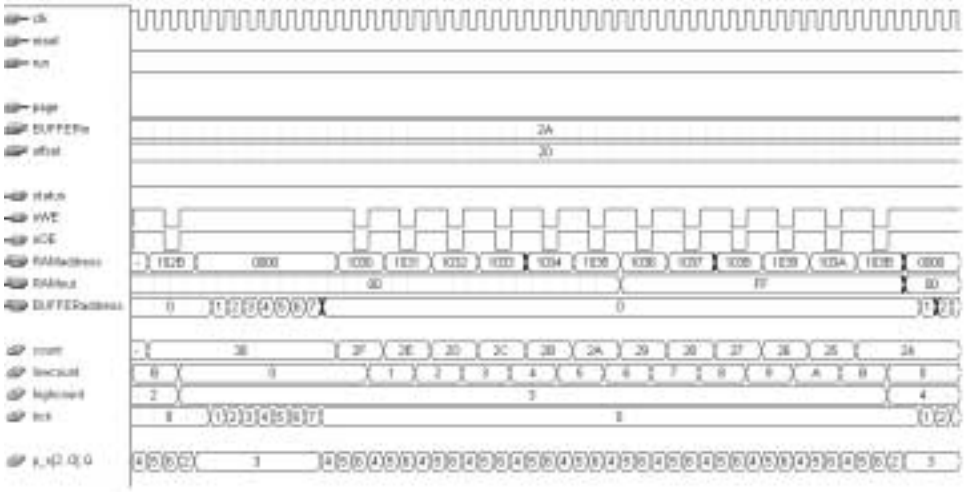

Figure 4.11: Simulation results of the write FSM during a single read/write cycle. See how the FSM reads the 8 samples in first and then goes through a series of writes.

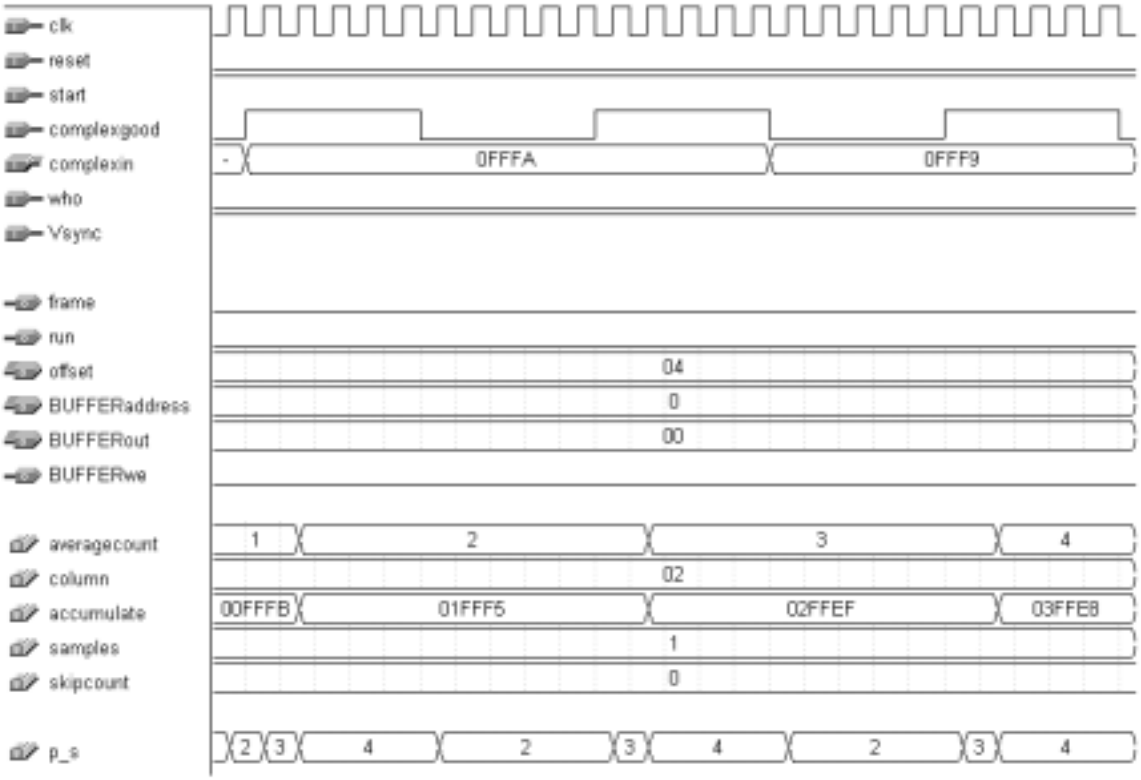

Figure 4.12: Simulation results of the ram FSM during a single loop. The accumulation and counting of the complexin input is shown in glitch-free operation.

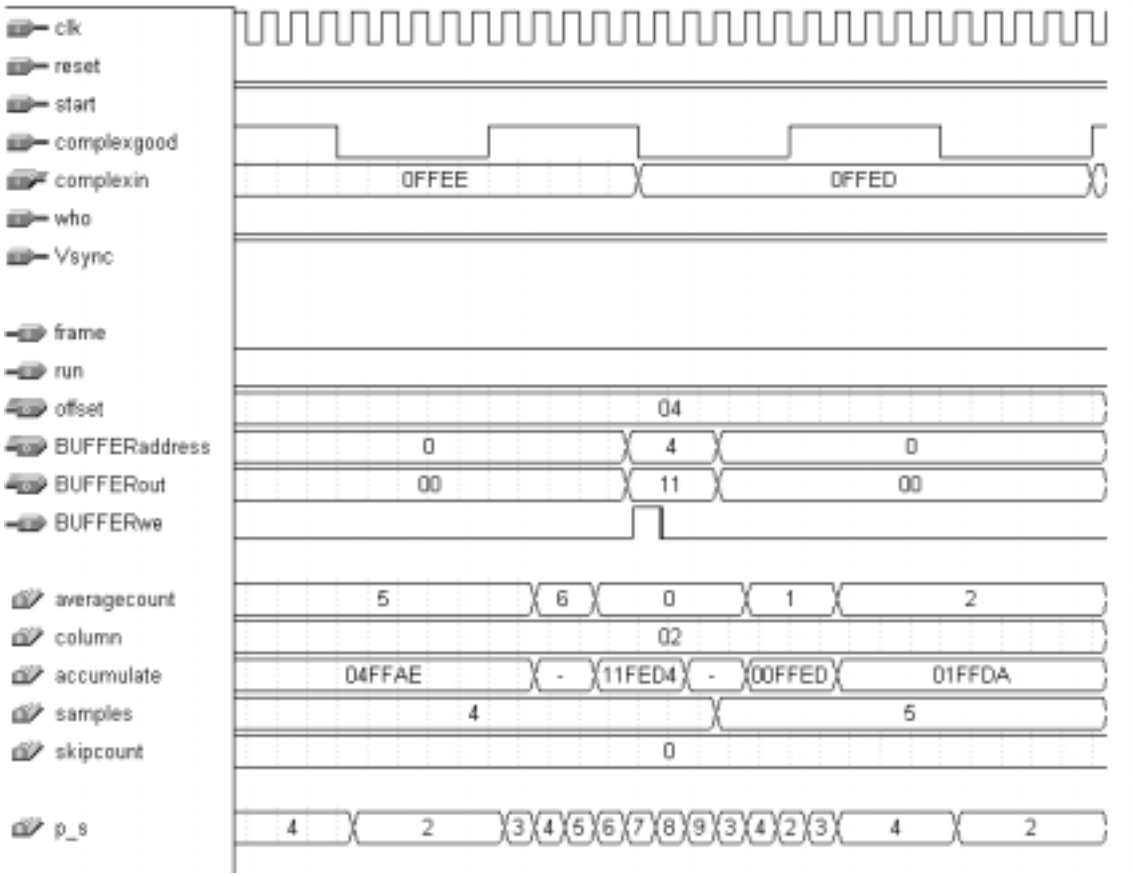

Figure 4.13: Simulation results of the ram FSM performing a write to the dual-port buffer. The aeragecount counter is reset after the signal is written. Note the address lines are stable during the write operation.

4.6 Conclusion

Despite the trouble we had in implementing our design, we are certain that we could have been more successful had we more time to hammer out the communication problems, more I/O pins, and a more powerful (larger) FPGA. The mere undertaking of a project of this scale and difficult has taught us many things about digital design, so I think we can measure a good deal of success from our work. Probably the most important thing about this last project was the exposure to more analog components, since in the end, the real world is analog. In particular, the topics of motors, transmission and communications, and video and audio signals are sure to be encountered in the real world, and will help us greatly in the future.

The many hours of coding, simulation, wiring, and debugging software and hardware have certainly left their mark on me and my groupmates, and I suspect that it will last for some time as a testament to our endurance. I am willing to admit that I had a good deal of fun in this great learning process of digital design, and in the end, the rewards for all the hard work invested could not be sweeter. Thus ends a long and hard year in the shadowy and frighteningly real world of hardware.

Chapter 5

Conclusion

All three modules of this project were successfully designed, implemented, and tested in simulation. Some problems occurred when these implementations were translated from the simulator to the hardware, and when the three modules were integrated together.

The FFT module was successfully implemented in simulation. The analog/digital tests passed in hardware, indicating that the analog/digital hardware and controller was working correctly, as well as the control unit, memory manager, and RAM. A 16-point FFT was successfully performed in simulation, but numeric instability made the 1024-point FFT unreliable in the actual implementation.

The Processing module was also successfully implemented in simulation, and verified to be working in hardware using the logic analyzer, though problems with the lab kit forced some features to be removed, due to a malfunctioning NuBus interface.

The Video module was also successfully implemented in simulation. It was tested in hardware using a video monitor. A test circuit verified the correct operation of the analog component, and a simple memory test verified the correct operation of the RAM and memory multiplexors. Address testing determined the correct format for addressing RAM locations.

Integrating the three modules proved to be a problem. It was not possible to make the parallel interface bus function reliably. The bus would transmit some data, but then stall in a forbidden state in which both the FFTSENDING and PROCSENDING lines were high. This suggests that both the FFT module and Processing module interface FSMs believed that they were transmitting data. This result was unexpected, as the interface FSMs were tested in simulation and no such error was observed. The problem is still unclear because not enough time to perform extensive testing was available. Ultimately, we theorized that transmission line effects were causing glitches on the bus control lines that caused the two FSMs to become out of sync. Line drivers and receivers were added to minimize these effects, which caused the bus to become more functional for brief periods. However, these problems were not solvable in the time available. If more time were available, we would modify the interface FSMs of each module to make them more robust to this sort of error.

As a result, our project was not entirely functional, since all three components needed to be working and integrated in hardware to have an operational result. Nonetheless, a valid design was completed, and working code was implemented and validated, and testing was performed. The correct operation of nearly every sub-module was verified, which represented a substantial accomplishment. Indeed, the only problems that remained were some numeric error in the FFT, which was unavoidable, and some integration issues that were not able to be debugged thoroughly. We are confident that, given enough time and resources, these obstacles could be overcome while maintaining our original, valid design.

On a more personal level, though we are obviously disappointed that the full extent of our project could not be fully realized, we now recognize the full complexity of the problem we were attempting

to solve. Even so, we were able to create a workable design for approaching the problem, and an implementation of this design, including some rather complex components. We were each able to implement our respective designs, verifying their correct operation in simulation, and translating most of them to hardware. Though some problems were encountered that prevented the project from working successfully, the majority of the project was nearly operational. We consider this a substantial accomplishment. Moreover, the process of designing and implementing this project proved quite educational, demonstrating the process of taking a problem, generating a design to approach it, and translating that design to a hardware implementation. We found this to be quite rewarding.

Appendix A

Source Code

A.1 Conversion / FFT Module

A.1.1 top.vhd

```
1  ----------------------------------------------------------------------------
2  --
3  -- top.vhd: top-level structural definition
4  -- Dan R. K. Ports  <drkp@mit.edu>
5  -- 6.111 final project, 2003/12/08
6  --
7  ----------------------------------------------------------------------------
8
9  library ieee;
10 use ieee.std_logic_1164.all;
11 use ieee.std_logic_arith.all;
12 use ieee.std_logic_signed.all;
13
14 entity top is
15
16   port (
17     clk, reset                        : in    std_logic;
18     n_DACEnable, n_ADCEnable, ADCRead : out   std_logic;
19     ADCStatus                         : in    std_logic;
20     adbus                             : inout std_logic_vector(7 downto 0);
21     fftsending, fftrdyack             : out   std_logic;
22     procsending, procrdyack, ack2     : in    std_logic;
23     ramaddr                           : out   std_logic_vector(13 downto 0)
           ;
24 --    ramdatain                         : in    std_logic_vector(7 downto 0);
25 --    ramdataout                        : out   std_logic_vector(7 downto 0);
26     ramdata                           : inout std_logic_vector(7 downto 0);
27     n_ramwe                           : out   std_logic;
28     adtest, ffttest                   : in    std_logic;
29     curportout                        : out   std_logic_vector(1 downto 0);
```

```
30      ifdatabus                       : inout std_logic_vector(7 downto 0))
            ;
31
32  end top;
33
34
35  architecture structural of top is
36
37    component adfsm
38    port (
39      clk, start            : in  std_logic;
40      reset                 : in  std_logic;
41      timerClk              : in  std_logic;
42      busy                  : out std_logic;
43      ADCRead, n_DACEnable  : out std_logic;
44      n_ADCEnable           : out std_logic;
45      ADCStatus             : in  std_logic;
46      ramwe                 : out std_logic;
47      ramaddr               : out std_logic_vector(13 downto 0);
48      ramdatain             : in  std_logic_vector(7 downto 0);
49      ramdataout            : out std_logic_vector(7 downto 0);
50      adcbufsel             : in  std_logic_vector(2 downto 0);
51      dacbufsel             : in  std_logic_vector(2 downto 0);
52      ramreq                : out std_logic;
53      ramdone               : in  std_logic;
54      adbus                 : inout std_logic_vector(7 downto 0));
55    end component;
56
57    component control
58    port (
59      reset, clk                            : in  std_logic;
60      adstart, fftstart, ifstart            : out std_logic;
61      adbusy, fftbusy, ifbusy               : in  std_logic;
62      adtest, ffttest                       : in  std_logic;
63      fftinverse                            : out std_logic;
64      adcbufsel, dacbufsel, fftinbufsel,
65      fftoutbufsel, ifsendbufsel, ifrecvbufsel
66                                : out std_logic_vector(2 downto 0));
67    end component;
68
69    component divider
70    port (
71      clk, rst : in  std_logic;
72      longclk  : out std_logic);
73    end component;
74
75    component fft
76    port (
77      start, reset, clk         : in    std_logic;
78      addrbus                   : out   std_logic_vector(13 downto 0);
79      testaddrbus                  : out std_logic_vector(6 downto 0);
```

```
80 --    databus                    : inout std_logic_vector(7 downto 0);
81       databusout                 : out   std_logic_vector(7 downto 0);
82       databusin                  : in    std_logic_vector(7 downto 0);
83       busy                       : out   std_logic;
84       inbufsel, outbufsel        : in    std_logic_vector(2 downto 0);
85       inverse                    : in    std_logic;
86       treout,timout,breout,bimout,
87       wbreout, wbimout           : out   std_logic_vector(7 downto 0);
88       databusinout               : out   std_logic_vector(7 downto 0);
89       nextiout                   : out   std_logic_vector(9 downto 0);
90       copyimout                  : out   std_logic;
91       ramreq                     : out   std_logic;
92       ramdone                    : in    std_logic;
93       ramwe                      : out   std_logic);
94     end component;
95
96     component interfacefsm
97     port (
98       start                    : in    std_logic;
99       reset                    : in    std_logic;
100      busy                     : out   std_logic;
101      sendbufsel, recvbufsel   : in    std_logic_vector(2 downto 0);
102      clk                      : in    std_logic;
103      fftsending               : out   std_logic;
104      procsending              : in    std_logic;
105      fftrdyack                : out   std_logic;
106      procrdyack               : in    std_logic;
107      ack2                     : in    std_logic;
108      ramaddr                  : out   std_logic_vector(13 downto 0);
109      ramdatain                : in    std_logic_vector(7 downto 0);
110      ramdataout               : out   std_logic_vector(7 downto 0);
111      databus                  : inout std_logic_vector(7 downto 0);
112      ramreq                   : out   std_logic;
113      ramdone                  : in    std_logic;
114      ramwe                    : out   std_logic);
115    end component;
116
117    component memmgr
118    port (
119      aaddr, baddr, caddr : in    std_logic_vector(14-1 downto 0);
120      adata, bdata, cdata : in
121             std_logic_vector(8-1 downto 0) := (others => 'Z');
122      aq, bq, cq          : out
123             std_logic_vector(8-1 downto 0) := (others => 'Z');
124      awr, bwr, cwr       : in    std_logic;
125      clk, reset          : in    std_logic;
126      areq, breq, creq    : in    std_logic;
127      adone, bdone, cdone : out   std_logic;
128      ramaddr             : out   std_logic_vector(14-1 downto 0);
129      ramdata             : inout
130             std_logic_vector(8-1 downto 0) := (others => 'Z');
```

```
131 --     ramdatain            : in     std_logic_vector(7 downto 0);
132 --     ramdataout           : out    std_logic_vector(7 downto 0);
133       curportout           : out    std_logic_vector(1 downto 0);
134       ramwe                : out    std_logic);
135     end component;
136
137     component synchronizer1
138     port (
139       clk, syncin : in  std_logic;
140       syncout     : out std_logic);
141     end component;
142
143     signal adstart, adbusy, fftstart, fftbusy, ifstart,
144            ifbusy, timerClk, awr, bwr, cwr, areq, breq,
145            creq, adone, bdone, cdone : std_logic;
146     signal aaddr, baddr, caddr : std_logic_vector(13 downto 0);
147     signal ad, bd, cd, aq, bq, cq : std_logic_vector(7 downto 0);
148     signal adcbufsel, dacbufsel, fftinbufsel, fftoutbufsel,
149            ifsendbufsel, ifrecvbufsel : std_logic_vector(2 downto 0);
150     signal fftinverse, procsendingsync,
151          procrdyacksync, ack2sync : std_logic;
152     signal ramwe : std_logic;
153
154
155   begin  -- structural
156
157     adfsminst : adfsm port map (
158       clk          => clk,
159       start        => adstart,
160       reset        => reset,
161       timerClk     => timerClk,
162       busy         => adbusy,
163       ADCRead      => ADCRead,
164       n_DACEnable => n_DACEnable,
165       n_ADCEnable => n_ADCEnable,
166       ADCStatus    => ADCStatus,
167       ramwe        => bwr,
168       ramaddr      => baddr,
169       ramdatain    => bq,
170       ramdataout   => bd,
171       adcbufsel    => adcbufsel,
172       dacbufsel    => dacbufsel,
173       ramreq       => breq,
174       ramdone      => bdone,
175       adbus        => adbus);
176
177     controlinst : control port map (
178       reset         => reset,
179       clk           => clk,
180       adtest        => adtest,
181       ffttest       => ffttest,
```

```
182       adstart       => adstart,
183       fftstart      => fftstart,
184       ifstart       => ifstart,
185       adbusy        => adbusy,
186       fftbusy       => fftbusy,
187       ifbusy        => ifbusy,
188       fftinverse    => fftinverse,
189       adcbufsel     => adcbufsel,
190       dacbufsel     => dacbufsel,
191       fftinbufsel   => fftinbufsel,
192       fftoutbufsel => fftoutbufsel,
193       ifsendbufsel => ifsendbufsel,
194       ifrecvbufsel => ifrecvbufsel);
195
196     dividerinst : divider port map (
197       clk     => clk,
198       rst     => reset,
199       longclk => timerClk);
200
201     fftinst : fft port map (
202       start       => fftstart,
203       reset       => reset,
204       clk         => clk,
205       addrbus     => aaddr,
206       databusout => ad,
207       databusin   => aq,
208       busy        => fftbusy,
209       inbufsel    => fftinbufsel,
210       outbufsel   => fftoutbufsel,
211       inverse     => fftinverse,
212       ramreq      => areq,
213       ramdone     => adone,
214       ramwe       => awr);
215
216     ifinst : interfacefsm port map (
217       start        => ifstart,
218       reset        => reset,
219       busy         => ifbusy,
220       sendbufsel   => ifsendbufsel,
221       recvbufsel   => ifrecvbufsel,
222       clk          => clk,
223       fftsending   => fftsending,
224       procsending => procsendingsync,
225       fftrdyack    => fftrdyack,
226       procrdyack   => procrdyacksync,
227       ack2         => ack2sync,
228       ramaddr      => caddr,
229       ramdatain    => cq,
230       ramdataout   => cd,
231       databus      => ifdatabus,
232       ramreq       => creq,
```

```
233       ramdone      => cdone,
234       ramwe        => cwr);
235
236     memmgrinst : memmgr port map (
237       aaddr    => aaddr,
238       baddr    => baddr,
239       caddr    => caddr,
240       adata    => ad,
241       bdata    => bd,
242       cdata    => cd,
243       aq       => aq,
244       bq       => bq,
245       cq       => cq,
246       awr      => awr,
247       bwr      => bwr,
248       cwr      => cwr,
249       clk      => clk,
250       reset    => reset,
251       areq     => areq,
252       breq     => breq,
253       creq     => creq,
254       adone    => adone,
255       bdone    => bdone,
256       cdone    => cdone,
257       ramaddr => ramaddr,
258 --      ramdatain => ramdatain,
259 --      ramdataout => ramdataout,
260       ramdata => ramdata,
261       curportout => curportout,
262       ramwe    => ramwe);
263
264     syncprocsending : synchronizer1 port map (
265       clk      => clk,
266       syncin   => procsending,
267       syncout => procsendingsync);
268
269     syncprocrdyack : synchronizer1 port map (
270       clk      => clk,
271       syncin   => procrdyack,
272       syncout => procrdyacksync);
273
274     syncack2 : synchronizer1 port map (
275       clk      => clk,
276       syncin   => ack2,
277       syncout => ack2sync);
278
279
280     n_ramwe <=  not ramwe;
281
282 end structural;
```

A.1.2 adfsm.vhd

```
1  ----------------------------------------------------------------------------
2  --
3  -- adfsm.vhd: analog/digital control FSM
4  -- Dan R. K. Ports <drkp@mit.edu>
5  -- 6.111 final project, 2003/12/01
6  --
7  ----------------------------------------------------------------------------
8
9  library ieee;
10 use ieee.std_logic_1164.all;
11 use ieee.std_logic_arith.all;
12 use ieee.std_logic_unsigned.all;
13
14 entity adfsm is
15
16   port (
17     clk, start           : in  std_logic;
18     reset                : in  std_logic;
19     timerClk             : in  std_logic;
20     busy                 : out std_logic;
21     ADCRead, n_DACEnable : out std_logic;
22     n_ADCEnable          : out std_logic;
23     ADCStatus            : in  std_logic;
24     ramwe                : out std_logic;
25     ramaddr              : out std_logic_vector(13 downto 0);
26     ramdatain            : in  std_logic_vector(7 downto 0);
27     ramdataout           : out std_logic_vector(7 downto 0);
28     adcbufsel            : in  std_logic_vector(2 downto 0);
29     dacbufsel            : in  std_logic_vector(2 downto 0);
30     ramreq               : out std_logic;
31     ramdone              : in  std_logic;
32     adbus                : inout std_logic_vector(7 downto 0));
33
34 end adfsm;
35
36 architecture behavioral of adfsm is
37
38   type StateType is (s_Idle, s_WaitForTimer, s_RAMRead, s_DACWrite,
39                      s_ADCEnable, s_ADCWait, s_ADCRead, s_RAMWrite,
40                      s_AddrInc, s_RAMWrite2, s_AddrInc2);
41
42   signal curState, nextState : StateType := s_Idle;
43   signal i, nexti : std_logic_vector(3 downto 0);
44                                        -- iterator for multi-cycle
                                            waits
45   signal count : std_logic_vector(10 downto 0);
46   signal buf : std_logic_vector(7 downto 0);
```

```
47
48
49  begin  -- behavioral
50
51    -- purpose: select next state
52    -- type    : combinational
53    -- inputs : curState, ADCStatus, start
54    -- outputs: nextState, nexti
55    next_state: process (curState, ADCStatus, start, i)
56    begin  -- process next_state
57      if reset = '1' then
58        nextState <= s_Idle;
59      else
60        case curState is
61          when s_Idle =>
62            if start = '1' then
63              nextState <= s_WaitForTimer;
64            else
65              nextState <= s_Idle;
66            end if;
67          when s_WaitForTimer =>
68            if timerClk = '1' then
69              nextState <= s_RAMRead;
70            else
71              nextState <= s_WaitForTimer;
72            end if;
73          when s_RAMRead =>
74            if ramdone = '1' then
75              nextState <= s_DACWrite;
76              nexti <= "0010";
77            else
78              nextState <= s_RAMRead;
79              nexti <= i;
80            end if;
81          when s_DACWrite =>
82            if i = 0 then
83              nextState <=  s_ADCEnable;
84            else
85              nextState <=  s_DACWrite;
86              nexti <= i-i;
87            end if;
88          when s_ADCEnable =>
89            if ADCStatus = '1' then
90              nextState <= s_ADCWait;
91              nexti <= "0011";
92            else
93              nextState <= s_ADCEnable;
94            end if;
95          when s_ADCWait =>
96            if ADCStatus = '0' then
97              if i = 0 then
```

```
98                nextState <= s_ADCRead;
99              else
100               nextState <= s_ADCWait;
101               nexti <= i - 1;
102             end if;
103           else
104               nextState <= s_ADCWait;
105               nexti <= "0011";
106           end if;
107         when s_ADCRead =>
108           nextState <= s_RAMWrite;
109           nexti <= "0001";
110         when s_AddrInc =>
111           nextState <= s_RAMWrite2;
112         when s_RAMWrite =>
113           if ramdone = '1' then
114             nextState <= s_AddrInc;
115           else
116             nextState <= s_RAMWrite;
117           end if;
118         when s_RAMWrite2 =>
119           if ramdone = '1' then
120             nextState <= s_AddrInc2;
121           else
122             nextState <= s_RAMWrite2;
123           end if;
124         when s_AddrInc2 =>
125           if count = 0 then
126             nextState <= s_Idle;
127           else
128             nextState <= s_WaitForTimer;
129           end if;
130         when others =>
131           nextState <= s_Idle;
132       end case;
133     end if;
134   end process next_state;
135
136   -- purpose: update state on clock
137   -- type    : combinational
138   -- inputs : clk, nextState, nexti
139   -- outputs: curState, i
140   state_clocked: process (clk)
141   begin  -- process state_clocked
142     if rising_edge(clk) then
143       curState <= nextState;
144       i <= nexti;
145       if nextState = s_AddrInc or nextState = s_AddrInc2 then
146         count <= count + 1;
147       elsif nextState = s_Idle then
148         count <= (others => '0');
```

```
149         end if;
150         if curState = s_RAMRead then
151           buf <= ramdatain;
152         elsif curState = s_ADCRead then
153           buf <= adbus;
154         elsif curState = s_AddrInc then
155           buf <= (others => '0');
156         end if;
157       end if;
158     end process state_clocked;
159
160     n_DACEnable <= '0' when curState = s_DACWrite else '1';
161     ramwe <= '1' when curState = s_RAMWrite or curState = s_RAMWrite2 else
            '0';
162     ADCRead <= '0' when curState = s_ADCEnable else '1';
163     busy <= '0' when curState = s_Idle else '1';
164     n_ADCEnable <= '0' when curState = s_ADCEnable or curState = s_ADCWait
165                    or curState = s_ADCRead else '1';
166
167     ramdataout <= buf;
168
169     -- purpose: generate adbus output
170     -- type    : combinational
171     -- inputs : curState
172     -- outputs: adbus
173     adbusout: process (curState)
174     begin  -- process adbusout
175       if curState = s_DACWrite then
176         adbus <= (not buf(7)) & buf(6 downto 0);
177       else
178         adbus <= (others => 'Z');
179       end if;
180     end process adbusout;
181
182     ramaddr <= dacbufsel & count when curState = s_RAMRead
183                else adcbufsel & count;
184
185     ramreq <= '1' when curState = s_RAMRead or curState = s_RAMWrite
186               or curState = s_RAMWrite2 else '0';
187
188   end behavioral;
```

A.1.3 complexmult.vhd

```
1  -- complexmult.vhd: combinational complex multiplier
2  -- Dan R. K. Ports  <drkp@mit.edu>
3  -- 6.111 final project, 2003/12/03
4
5  library ieee;
6  use ieee.std_logic_1164.all;
7  use ieee.std_logic_arith.all;
8  use ieee.std_logic_signed.all;
9
10 entity complexmult is
11
12   port (
13     are, aim, bre, bim : in  std_logic_vector(7 downto 0);
14     prodre, prodim     : out std_logic_vector(15 downto 0));
15
16 end complexmult;
17
18 architecture structural of complexmult is
19
20   component mult
21         PORT
22         (
23                 dataa           : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
24                 datab           : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
25                 result          : OUT STD_LOGIC_VECTOR (15 DOWNTO 0)
26         );
27   end component;
28
29   signal re1, re2, im1, im2 : std_logic_vector(15 downto 0);
30
31 begin  -- structural
32
33   mult1 : mult port map (
34     dataa  => are,
35     datab  => bre,
36     result => re1);
37
38   mult2 : mult port map (
39     dataa  => aim,
40     datab  => bim,
41     result => re2);
42
43   mult3 : mult port map (
44     dataa  => are,
45     datab  => bim,
46     result => im1);
47
48   mult4 : mult port map (
49     dataa  => aim,
```

```
50        datab  => bre,
51        result => im2);
52
53     prodre <= re1 - re2;
54     prodim <= im1 + im2;
55
56
57  end structural;
```

A.1.4 control.vhd

```
1  ----------------------------------------------------------------------------
2  --
3  -- control.vhd: control finite state machine
4  -- Dan R. K. Ports  <drkp@mit.edu>
5  -- 6.111 final project, 2003/12/08
6  --
7  ----------------------------------------------------------------------------
8
9  library ieee;
10 use ieee.std_logic_1164.all;
11 use ieee.std_logic_arith.all;
12 use ieee.std_logic_signed.all;
13
14 entity control is
15
16   port (
17     reset, clk                          : in  std_logic;
18     adtest, ffttest                     : in  std_logic;
19     adstart, fftstart, ifstart          : out std_logic;
20     adbusy, fftbusy, ifbusy             : in  std_logic;
21     fftinverse                          : out std_logic;
22     adcbufsel, dacbufsel, fftinbufsel,
23     fftoutbufsel, ifsendbufsel, ifrecvbufsel
24                                         : out std_logic_vector(2 downto
                                                0));
25
26 end control;
27
28 architecture behavioral of control is
29
30   type StateType is (s_Idle, s_Start, s_Wait1, s_StartIFFT, s_Wait2,
31                      s_Rotate);
32   signal curState, nextState : StateType := s_Idle;
33
34   signal adcbuf, dacbuf, fftinbuf, fftoutbuf, ifftinbuf, ifftoutbuf,
35          ifsendbuf, ifrecvbuf : std_logic_vector(2 downto 0);
36
37   signal rotatestate : std_logic := '0';
38
39 begin  -- behavioral
40
41   -- purpose: determine next state
42   -- type    : combinational
43   -- inputs : curState, adbusy, fftbusy, ifbusy
44   -- outputs: nextState
45   next_state: process (curState, adbusy, fftbusy, ifbusy)
46   begin  -- process next_state
```

```
47      if reset = '1' then
48        nextState <= s_Rotate;
49      else
50        case curState is
51          when s_Idle =>
52            nextState <= s_Start;
53          when s_Start =>
54            if adbusy = '1' and (fftbusy = '1' or adtest = '1') and
55              (ifbusy = '1' or adtest = '1' or ffttest = '1') then
56              nextState <= s_Wait1;
57            else
58              nextState <= s_Start;
59            end if;
60          when s_Wait1 =>
61            if fftbusy = '0' or adtest = '1' then
62              nextState <= s_StartIFFT;
63            else
64              nextState <= s_Wait1;
65            end if;
66          when s_StartIFFT =>
67            if fftbusy = '1' or adtest = '1' then
68              nextState <= s_Wait2;
69            else
70              nextState <= s_StartIFFT;
71            end if;
72          when s_Wait2 =>
73            if adbusy = '0' and (fftbusy = '0' or adtest = '1')
74              and (ifbusy = '0' or adtest = '1' or ffttest = '1') then
75              nextState <= s_Rotate;
76            else
77              nextState <= s_Wait2;
78            end if;
79          when s_Rotate =>
80            nextState <= s_Start;
81          when others =>
82            nextState <= s_Idle;
83        end case;
84      end if;
85    end process next_state;
86
87    -- purpose: perform state transitions and buffer rotations
88    -- type    : combinational
89    -- inputs : clk
90    -- outputs: curState, bufsels
91    state_clocked: process (clk)
92    begin  -- process state_clocked
93      if rising_edge(clk) then
94        curState <= nextState;
95
96        if nextState = s_Rotate then
97
```

```
98          if adtest = '1' then
99            if rotatestate = '0' or reset = '1' then
100             adcbuf <= "000";
101             dacbuf <= "001";
102             rotatestate <= '1';
103           else
104             adcbuf <= "001";
105             dacbuf <= "000";
106             rotatestate <= '0';
107           end if;
108         elsif ffttest = '1' then
109           if rotatestate = '0' or reset = '1' then
110             adcbuf <= "000";
111             dacbuf <= "001";
112             fftinbuf <= "111";
113             fftoutbuf <= "011";
114             ifftinbuf <= "100";
115             ifftoutbuf <= "010";
116             rotatestate <= '1';
117           else
118             adcbuf <= "111";
119             dacbuf <= "010";
120             fftinbuf <= "000";
121             fftoutbuf <= "100";
122             ifftinbuf <= "011";
123             ifftoutbuf <= "001";
124             rotatestate <= '0';
125           end if;
126         else
127
128           if rotatestate = '0' or reset = '1' then
129             adcbuf <= "000";
130             dacbuf <= "001";
131             fftinbuf <= "111";
132             fftoutbuf <= "110";
133             ifftinbuf <= "100";
134             ifftoutbuf <= "010";
135             ifsendbuf <= "101";
136             ifrecvbuf <= "011";
137             rotatestate <= '1';
138           else
139             adcbuf <= "111";
140             dacbuf <= "010";
141             fftinbuf <= "000";
142             fftoutbuf <= "101";
143             ifftinbuf <= "011";
144             ifftoutbuf <= "001";
145             ifsendbuf <= "110";
146             ifrecvbuf <= "100";
147             rotatestate <= '0';
148           end if;
```

```
149
150         end if;
151       end if;
152     end if;
153   end process state_clocked;
154
155   adcbufsel <= adcbuf;
156   dacbufsel <= dacbuf;
157   fftinbufsel <= ifftinbuf when curState = s_StartIFFT or curState =
        s_Wait2
158                  else fftinbuf;
159   fftoutbufsel <= ifftoutbuf when curState = s_StartIFFT or curState =
        s_Wait2
160                   else fftoutbuf;
161   fftinverse <= '1' when curState = s_StartIFFT or curState = s_Wait2
162                   else '0';
163   ifsendbufsel <= ifsendbuf;
164   ifrecvbufsel <= ifrecvbuf;
165
166   adstart <= '1' when curState = s_Start else '0';
167   ifstart <= '1' when curState = s_Start and adtest = '0' and ffttest
        = '0'
168              else '0';
169   fftstart <= '1' when (curState = s_Start or curState = s_StartIFFT)
170               and adtest = '0' else '0';
171
172
173 end behavioral;
```

A.1.5 divider.vhd

```
1  ----------------------------------------------------------------------------------

2  --
3  -- divider.vhd: 10 MHz to 44.248 kHz clock divider
4  -- Dan R. K. Ports  <drkp@mit.edu>
5  -- 6.111 Lab 2, 2003/10/01
6  -- Modified for 6.111 Lab 3, 2003/10/15
7  -- Modified for 6.111 final project, 2003/12/03
8  --
9  ----------------------------------------------------------------------------------

10
11 library ieee;
12 use ieee.std_logic_1164.all;
13 use ieee.std_logic_unsigned.all;
14
15 entity divider is
16
17   port (
18     clk, rst : in  std_logic;
19     longclk  : out std_logic);
20
21 end divider;
22
23 architecture behavioral of divider is
24
25   signal longclkstate : std_logic := '0';
26   signal count : std_logic_vector(24 downto 0);
27
28 begin  -- behavioral
29
30   -- purpose: generate output pulse when counter reaches 1000000
31   -- type   : sequential
32   -- inputs : clk, rst
33   -- outputs: longclk
34   process (clk, rst)
35   begin  -- process
36     if rst = '1' then            -- asynchronous reset (active high)
37       count <= (others => '0');
38     elsif clk'event and clk = '1' then  -- rising clock edge
39       count <= count + 1;
40     if count = 225 then
41       longclkstate <= '1';
42     elsif count = 226 then
43       count <= (others => '0');
44       longclkstate <= '0';
45     end if;
46     end if;
47
```

```
48
49    end process;
50
51    longclk <= longclkstate;
52
53  end behavioral;
```

A.1.6 fft.vhd

```
1  ----------------------------------------------------------------------------
2  --
3  -- fft.vhd: Fast Fourier Transform finite state machine
4  -- Dan R. K. Ports  <drkp@mit.edu>
5  -- 6.111 final project, 2003/12/04
6  --
7  ----------------------------------------------------------------------------
8
9  library ieee;
10 use ieee.std_logic_1164.all;
11 use ieee.std_logic_arith.all;
12 use ieee.std_logic_signed.all;
13
14 entity fft is
15
16   port (
17     start, reset, clk          : in    std_logic;
18     addrbus                    : out   std_logic_vector(13 downto 0);
19     testaddrbus                   : out std_logic_vector(6 downto 0);
20 --    databus                   : inout std_logic_vector(7 downto 0);
21     databusout                 : out   std_logic_vector(7 downto 0);
22     databusin                  : in    std_logic_vector(7 downto 0);
23     busy                       : out   std_logic;
24     inbufsel, outbufsel        : in    std_logic_vector(2 downto 0);
25     inverse                    : in    std_logic;
26     treout,timout,breout,bimout,
27     wbreout, wbimout                      : out   std_logic_vector(7
         downto 0);
28     databusinout               : out   std_logic_vector(7 downto 0);
29     nextiout                   : out   std_logic_vector(9 downto 0);
30     copyimout                  : out   std_logic;
31     ramreq                     : out   std_logic;
32     ramdone                    : in    std_logic;
33     ramwe                      : out   std_logic);
34
35 end fft;
36
37 architecture behavioral of fft is
38
39   component twiddler512
40     port (
41     twiddlesel : in  std_logic_vector(8 downto 0);
42     twiddlere, twiddleim  : out std_logic_vector(7 downto 0));
43   end component;
44
45   component twiddler8
46     port (
```

```
47      twiddlesel : in  std_logic_vector(8 downto 0);
48      twiddlere, twiddleim  : out std_logic_vector(7 downto 0));
49    end component;
50
51    component complexmult
52      port (
53      are, aim, bre, bim : in  std_logic_vector(7 downto 0);
54      prodre, prodim     : out std_logic_vector(15 downto 0));
55
56    end component;
57
58    type StateType is (s_Idle, s_CopySel, s_CopyWr, s_FFTStart,
59                       s_PassStart, s_BlockStart, s_BflyLoad,
60                       s_BflyWrite, s_BflyEnd, s_BlockEnd, s_PassEnd);
61
62    signal curState, nextState : StateType;
63    signal i, nexti : std_logic_vector(9 downto 0);
64    signal copyim : std_logic := '0';
65    signal p : std_logic_vector(4 downto 0);
66    signal BP, b, NP, NPp, BaseB, BaseT, k : std_logic_vector(9 downto 0);
67    signal tre, tim, bre, bim, cbuf : std_logic_vector(7 downto 0);
68    signal wbre, wbim : std_logic_vector(15 downto 0);
69    signal twiddlesel : std_logic_vector(8 downto 0);
70    signal twiddlere, twiddleim, twiddleimi : std_logic_vector(7 downto 0)
        ;
71    signal addrbusint : std_logic_vector(13 downto 0);
72    signal dataoutbuf : std_logic_vector(7 downto 0);
73
74  begin  -- behavioral
75
76    treout <= tre;
77    timout <= tim;
78    bimout <= bim;
79    breout <= bre;
80    wbreout <= wbre(15 downto 8);
81    wbimout <= wbim(15 downto 8);
82    copyimout <= copyim;
83    nextiout <= nexti;
84    databusinout <= databusin;
85
86    twiddler : twiddler512 port map (
87      twiddlesel => twiddlesel,
88      twiddlere  => twiddlere,
89      twiddleim  => twiddleim);
90
91    cmult : complexmult port map (
92      are    => bre,
93      aim    => bim,
94      bre    => twiddlere,
95      bim    => twiddleimi,
96      prodre => wbre,
```

```
97      prodim => wbim);
98
99    twiddleimi <= twiddleim when inverse = '0' else ((not twiddleim) + 1);
100
101   -- purpose: select next state
102   -- type    : combinational
103   -- inputs : curState
104   -- outputs: nextState, nexti
105   newstate: process (curState, i, start, reset)
106   begin  -- process newstate
107
108     if reset = '1' then
109       nextState <= s_Idle;
110     else
111       case curState is
112         when s_Idle =>
113           if start = '1' then
114             nextState <= s_CopySel;
115             nexti <= (others => '0');
116           else
117             nextState <= s_Idle;
118             nexti <= i;
119           end if;
120         when s_CopySel =>
121           if ramdone = '1' then
122             nextState <= s_CopyWr;
123             nexti <= i;
124           else
125             nextState <= s_CopySel;
126             nexti <= i;
127           end if;
128         when s_CopyWr =>
129           if ramdone = '1' then
130           if i = 1023 then --*
131
132 --            if i = 15 then
133               nextState <= s_FFTStart;
134             else
135               nextState <= s_CopySel;
136             end if;
137             if copyim = '1' then
138               nexti <= i + 1;
139             end if;
140           else
141             nextState <= s_CopyWr;
142             nexti <= i;
143           end if;
144         when s_FFTStart =>
145           nextState <= s_PassStart;
146         when s_PassStart =>
147           nextState <= s_BlockStart;
```

```
148           when s_BlockStart =>
149             nextState <= s_BflyLoad;
150             nexti <= (others => '0');
151           when s_BflyLoad =>
152             if ramdone = '1' then
153               if i = 3 then
154                 nextState <= s_BflyWrite;
155                 nexti <= (others => '0');
156               else
157                 nextState <= s_BflyLoad;
158                 nexti <= i + 1;
159               end if;
160             else
161               nextState <= s_BflyLoad;
162               nexti <= i;
163             end if;
164           when s_BflyWrite =>
165             if ramdone = '1' then
166               if i = 3 then
167                 nextState <= s_BflyEnd;
168                 nexti <= (others => '0');
169               else
170                 nextState <= s_BflyWrite;
171                 nexti <= i + 1;
172               end if;
173             else
174               nextState <= s_BflyWrite;
175               nexti <= i;
176             end if;
177           when s_BflyEnd =>
178             nexti <= (others => '0');
179             if k = npp then
180               nextState <= s_BlockEnd;
181             else
182               nextState <= s_BflyLoad;
183             end if;
184           when s_BlockEnd =>
185             if b = bp then
186               nextState <= s_PassEnd;
187             else
188               nextState <= s_BlockStart;
189             end if;
190           when s_PassEnd =>
191             if p = 10 then                    --*
192 --            if p = 4 then
193               nextState <= s_Idle;
194             else
195               nextState <= s_PassStart;
196             end if;
197           when others =>
198             nextState <= s_Idle;
```

```
199         end case;
200       end if;
201
202     end process newstate;
203
204     -- purpose: update state and i on clock
205     -- type    : combinational
206     -- inputs : clk
207     -- outputs: curState, i
208     state_clocked: process (clk)
209     begin  -- process state_clocked
210       if rising_edge(clk) then
211
212         curState <= nextState;
213         i <= nexti;
214
215         if nextState = s_CopySel then
216           if curState = s_CopyWr then
217             copyim <= not copyim;
218           elsif curState = s_CopySel then
219             copyim <= copyim;
220           else
221             copyim <= '0';
222           end if;
223         elsif nextState = s_FFTStart then
224           Bp <= "1000000000";                --*
225 --          BP <= "0000001000";
226           Np <= "0000000010";
227           p <= (others => '0');
228         elsif nextState = s_PassStart then
229           BaseT <= (others => '0');
230           b <= (others => '0');
231         elsif nextState = s_BlockStart then
232           k <= (others => '0');
233           twiddlesel <= (others => '0');
234           BaseB <= BaseT + npp;
235         elsif nextState = s_BflyEnd then
236           k <= k + 1;
237           twiddlesel <= twiddlesel + ('0' & Bp(9 downto 1));
238         elsif nextState = s_BlockEnd then
239           b <= b + 1;
240           BaseT <= BaseT + np;
241         elsif nextState = s_PassEnd then
242           p <= p + 1;
243           Np <= Np(8 downto 0) & '0';
244           Bp <= '0' & Bp(9 downto 1);
245         end if;
246
247         if curState = s_BflyLoad and i = 0 then
248           tre <= databusin;
249         elsif curState = s_BflyLoad and i = 1 then
```

```
250           tim <= databusin;
251         elsif curState = s_BflyLoad and i = 2 then
252           bre <= databusin;
253         elsif curState = s_BflyLoad and i = 3 then
254           bim <= databusin;
255         end if;
256       end if;
257
258     end process state_clocked;
259
260
261     -- purpose: read input values from RAM at appropriate times
262     -- type    : combinational
263     -- inputs : curState, databus
264     -- outputs: tre, tim, bre, bim
265 --    grab_inputs: process (curState, databusin)
266 --    begin  -- process grab_inputs
267 --      if curState = s_BflyLoad then
268 --        if i = 0 then
269 --          tre <= databusin;
270 --          tim <= tim;
271 --          bre <= bre;
272 --          bim <= bim;
273 --        elsif i = 1 then
274 --          tre <= tre;
275 --          tim <= databusin;
276 --          bre <= bre;
277 --          bim <= bim;
278 --        elsif i = 2 then
279 --          tre <= tre;
280 --          tim <= tim;
281 --          bre <= databusin;
282 --          bim <= bim;
283 --        elsif i = 3 then
284 --          tre <= tre;
285 --          tim <= tim;
286 --          bre <= bre;
287 --          bim <= databusin;
288 --        end if;
289 --      elsif curState = s_BflyWrite then
290 --        tre <= tre;
291 --        tim <= tim;
292 --        bre <= bre;
293 --        bim <= bim;
294 --      elsif curState = s_CopySel then
295 --        tre <= databusin;
296 --      elsif curState = s_CopyWr then
297 --        tre <= tre;
298 --      else
299 --        tre <= tre;
300 --        tim <= tim;
```

```
301 --        bre <= bre;
302 --        bim <= bim;
303 --      end if;
304 --    end process grab_inputs;
305 --
306 --    temp_grab_inputs: process (curState, databus)
307 --    begin  -- process temp_grab_inputs
308 --      if curState = s_CopySel then
309 --        tre <= databus;
310 --        tim <= databus;
311 --        bre <= databus;
312 --        bim <= databus;
313 --      end if;
314 --    end process temp_grab_inputs;
315 --
316 --    process (curState, i, databusin)
317 --    begin
318 --      if curState = s_BflyLoad and i=0 then
319 --        tre <= databusin;
320 --      else
321 --        tre <= tre;
322 --      end if;
323 --    end process;
324 --
325 --
326 --    process (curState, i, databusin)
327 --    begin
328 --      if curState = s_BflyLoad and i=1 then
329 --        tim <= databusin;
330 --      else
331 --        tim <= tim;
332 --      end if;
333 --    end process;
334 --
335 --    process (curState, i, databusin)
336 --    begin
337 --      if curState = s_BflyLoad and i=2 then
338 --        bre <= databusin;
339 --      else
340 --        bre <= bre;
341 --      end if;
342 --    end process;
343 --
344 --    process (curState, i, databusin)
345 --    begin
346 --      if curState = s_BflyLoad and i=3 then
347 --        bim <= databusin;
348 --      else
349 --        bim <= bim;
350 --      end if;
351 --    end process;
```

```
352
353   process (curState, i, databusin)
354   begin
355     if curState = s_CopySel then
356       cbuf <= databusin;
357     else
358       cbuf <= cbuf;
359     end if;
360   end process;
361
362
363
364   busy <= '0' when curState = s_Idle else '1';
365   ramwe <= '1' when curState = s_CopyWr or curState = s_BflyWrite else
          '0';
366   ramreq <= '1' when curState = s_CopySel or curState = s_CopyWr
367             or curState = s_BflyLoad or curState = s_BflyWrite else '0';
368
369
370
371   -- purpose: generate data bus outputs
372   -- type    : combinational
373   -- inputs : curState, tre, tim, bre, bim
374   -- outputs: databus
375   gen_bus_outputs: process (curState, tre, tim, wbre, wbim)
376   begin  -- process gen_bus_outputs
377
378     if curState = s_BflyWrite then
379       if i = 0 then
380           dataoutbuf <= tre + (wbre(15) & wbre(13 downto 7));
381       elsif i = 1 then
382           dataoutbuf <= tim + (wbim(15) & wbim(13 downto 7));
383       elsif i = 2 then
384           dataoutbuf <= tre - (wbre(15) & wbre(13 downto 7));
385       else
386           dataoutbuf <= tim - (wbim(15) & wbim(13 downto 7));
387       end if;
388     elsif curState = s_CopyWr then
389       dataoutbuf <= cbuf;
390     else
391 --      databusout <= (others => 'Z');
392       dataoutbuf <= (others => '0');
393     end if;
394   end process gen_bus_outputs;
395
396   -- purpose: chop off low-order bit on alternate pass
397   -- type    : combinational
398   -- inputs : curState, p, dataoutbuf
399   -- outputs: databusout
400   gen_real_bus_out: process (curState, p, dataoutbuf)
401   begin  -- process gen_real_bus_out
```

```
402       if (curState = s_BflyWrite and p(0) = '1' and not (p = 9)) or
              curState = s_CopyWr then
403         databusout <= dataoutbuf(7) & dataoutbuf(7 downto 1);
404       else
405         databusout <= dataoutbuf;
406       end if;
407     end process gen_real_bus_out;
408
409     gen_addr_out: process (clk)
410     begin  -- process gen_addr_out
411       if curState = s_BflyLoad or curState = s_BflyWrite then
412         if i = 0 or i = 1 then
413           addrbusint(10 downto 1) <= BaseT + k;
414         else
415           addrbusint(10 downto 1) <= BaseB + k;
416         end if;
417
418         addrbusint(0) <= i(0);
419         addrbusint(13 downto 11) <= outbufsel;
420       elsif curState = s_CopySel then
421         addrbusint <= inbufsel & i & copyim;
422       elsif curState = s_CopyWr then
423         --addrbusint <= outbufsel & i(0) & i(1) & i(2) & i(3) & i(4) & i
                (5) & i(6)
424         --           & i(7) & i(8) & i(9) & i(10);
425         addrbusint <= outbufsel & "000000" & i(0) & i(1) & i(2) & i(3) &
                copyim;
426       else
427         addrbusint <= (others => '0');
428       end if;
429     end process gen_addr_out;
430
431     npp <= '0' & np(9 downto 1);
432     testaddrbus <= addrbusint(12 downto 11) & addrbusint(4 downto 0);
433
434     addrbus <= addrbusint;
435   end behavioral;
```

A.1.7 interfacefsm.vhd

```
1  ----------------------------------------------------------------------------------

2  --
3  -- interfacefsm.vhd: FSM for controlling parallel bus interface
4  -- Dan R. K. Ports <drkp@mit.edu>
5  -- 6.111 final project, 2003/12/01
6  --
7  ----------------------------------------------------------------------------------

8
9  library ieee;
10 use ieee.std_logic_1164.all;
11 use ieee.std_logic_arith.all;
12 use ieee.std_logic_unsigned.all;
13
14 entity interfacefsm is
15
16   port (
17     start                  : in    std_logic;
18     reset                  : in    std_logic;
19     busy                   : out   std_logic;
20     sendbufsel, recvbufsel : in    std_logic_vector(2 downto 0);
21     clk                    : in    std_logic;
22     fftsending             : out   std_logic;
23     procsending            : in    std_logic;
24     fftrdyack              : out   std_logic;
25     procrdyack             : in    std_logic;
26     ack2                   : in    std_logic;
27     ramaddr                : out   std_logic_vector(13 downto 0);
28     ramdatain              : in    std_logic_vector(7 downto 0);
29     ramdataout             : out   std_logic_vector(7 downto 0);
30     databus                : inout std_logic_vector(7 downto 0);
31     ramreq                 : out   std_logic;
32     ramdone                : in    std_logic;
33     ramwe                  : out   std_logic);
34
35
36
37 end interfacefsm;
38
39 architecture behavioral of interfacefsm is
40
41   type StateType is (s_Idle, s_SendStart, s_SendAddrSelect,
       s_SendRAMWait,
42                      s_SendReady, s_SendAck, s_SendComplete, s_RecvWait,
43                      s_WriteAddrSelect, s_Write, s_RecvAck);
44
45   signal curState, nextState : StateType := s_Idle;
46
```

```
47    signal count : std_logic_vector(10 downto 0);
48    signal buf : std_logic_vector(7 downto 0);
49
50  begin  -- behavioral
51
52      -- purpose: determine next state
53      -- type    : combinational
54      -- inputs : curState, count, procsending, procrdyack, ack2, reset
55      -- outputs: nextState
56    newState: process (curState, count, procsending, procrdyack, ack2,
        reset)
57      begin  -- process newState
58        if reset = '1' then
59          nextState <= s_Idle;
60        else
61          case curState is
62            when s_Idle =>
63              if start = '1' then
64                nextState <= s_SendStart;
65              else
66                nextState <= s_Idle;
67              end if;
68            when s_SendStart => nextState <= s_SendRAMWait;
69            when s_SendRAMWait =>
70              if ramdone = '1' then
71                nextState <= s_SendReady;
72              else
73                nextState <= s_SendRAMWait;
74              end if;
75            when s_SendReady =>
76              if procrdyack = '1' and ack2 = '1' then
77                nextState <= s_SendAck;
78              else
79                nextState <= s_SendReady;
80              end if;
81            when s_SendAck =>
82              if procrdyack = '0' and ack2 = '0' then
83                nextState <= s_SendAddrSelect;
84              else
85                nextState <= s_SendAck;
86              end if;
87            when s_SendAddrSelect =>
88  --              if count = "00000000000" then
89              if count = 0 then
90                  nextState <= s_SendComplete;
91              else
92                  nextState <= s_SendRAMWait;
93              end if;
94            when s_SendComplete =>
95              if procsending = '1' then
96                nextState <= s_RecvWait;
```

```
97              else
98                nextState <= s_SendComplete;
99              end if;
100           when s_RecvWait =>
101             if procsending = '0' then
102               nextState <= s_Idle;
103             else
104               if procrdyack = '1' then
105                 nextState <= s_WriteAddrSelect;
106               else
107                 nextState <= s_RecvWait;
108               end if;
109             end if;
110           when s_WriteAddrSelect =>
111             nextState <= s_Write;
112           when s_Write =>
113             if ramdone = '1' then
114               nextState <= s_RecvAck;
115             else
116               nextState <= s_Write;
117             end if;
118           when s_RecvAck =>
119             if procrdyack = '0' then
120               if procsending = '0' then
121                 nextState <= s_Idle;
122               else
123                 nextState <= s_RecvWait;
124               end if;
125             else
126               nextState <= s_RecvAck;
127             end if;
128           when others => nextState <= s_Idle;
129       end case;
130     end if;
131   end process newState;
132
133   -- purpose: update current state on clock rising edge
134   -- type    : combinational
135   -- inputs : clk, nextState
136   -- outputs: curState
137   state_clocked: process (clk)
138   begin  -- process state_clocked
139     if rising_edge(clk) then
140       curState <= nextState;
141       if nextState = s_SendAddrSelect or nextState = s_WriteAddrSelect
              then
142         count <= count + 1;
143       elsif nextState = s_Idle then
144         count <= (others => '0');
145       end if;
146
```

```
147           if curState = s_SendRAMWait then
148             buf <= ramdatain;
149           elsif curState = s_RecvWait then
150             buf <= databus;
151           end if;
152         end if;
153       end process state_clocked;
154
155       databus <= buf when curState = s_SendRAMWait or curState =
              s_SendReady or curState = s_SendAck else (others => 'Z');
156
157       ramaddr <= sendbufsel & count when curState = s_SendAddrSelect
158                  or curState = s_SendAck or curState = s_SendReady
159                  or curState = s_SendStart or curState = s_SendComplete
160                  else recvbufsel & count;
161
162       ramwe <= '1' when curState = s_Write else '0';
163       ramdataout <= buf;
164
165       -- purpose: generate output values
166       -- type    : combinational
167       -- inputs : curState
168       -- outputs: busy, fftsending, ctrenable, readenable, writeenable
169     --  gen_outputs: process (curState)
170     --  begin  -- process gen_outputs
171         busy <= '0' when curState = s_Idle else '1';
172         fftsending <= '1' when curState = s_SendStart
173                                or curState = s_SendAddrSelect
174                                or curState = s_SendReady
175                                or curState = s_SendAck
176                                or curState = s_SendRAMWait
177                               else '0';
178 --        readenable <= '1' when curState = s_SendStart
179 --                                or curState = s_SendAddrSelect
180 --                                or curState = s_SendReady
181 --                                or curState = s_SendAck
182 --                                or curState = s_SendRAMWait
183 --                               else '0';
184
185       with curState select
186         fftrdyack <=
187         '0' when s_SendStart,
188         '0' when s_SendRAMWait,
189         '0' when s_SendAck,
190         '0' when s_SendAddrSelect,
191         '1' when s_SendReady,
192         '0' when s_WriteAddrSelect,
193         '0' when s_Write,
194         '0' when s_RecvWait,
195         '1' when s_RecvAck,
196         '0' when others;
```

```
197
198 --        ctrenable <= '1' when curState = s_SendAddrSelect
199 --                              or curState = s_WriteAddrSelect
200 --                          else '0';
201 --        writeenable <= '1' when curState = s_Write else '0';
202 --
203
204   --  end process gen_outputs;
205
206     ramreq <= '1' when curState = s_SendRAMWait or curState = s_Write
207               else '0';
208
209 end behavioral;
```

A.1.8 mult.vhd

```
1  -- megafunction wizard: %LPM_MULT%
2  -- GENERATION: STANDARD
3  -- VERSION: WM1.0
4  -- MODULE: lpm_mult
5
6  --============================================================
7  -- File Name: mult.vhd
8  -- Megafunction Name(s):
9  --                         lpm_mult
10 --============================================================
11 -- ************************************************************
12 -- THIS IS A WIZARD GENERATED FILE. DO NOT EDIT THIS FILE!
13 -- ************************************************************
14
15
16 --       Copyright (C) 1988-2002 Altera Corporation
17
18 --       Any megafunction design, and related net list (encrypted or
       decrypted),
19 --       support information, device programming or simulation file, and
       any other
20 --       associated documentation or information provided by Altera or a
       partner
21 --       under Altera's Megafunction Partnership Program may be used only
        to
22 --       program PLD devices (but not masked PLD devices) from Altera.
        Any other
23 --       use of such megafunction design, net list, support information,
       device
24 --       programming or simulation file, or any other related
       documentation or
25 --       information is prohibited for any other purpose, including, but
       not
26 --       limited to modification, reverse engineering, de-compiling, or
       use with
27 --       any other silicon devices, unless such use is explicitly
       licensed under
28 --       a separate agreement with Altera or a megafunction partner.
        Title to
29 --       the intellectual property, including patents, copyrights,
       trademarks,
30 --       trade secrets, or maskworks, embodied in any such megafunction
       design,
31 --       net list, support information, device programming or simulation
       file, or
32 --       any other related documentation or information provided by
       Altera or a
33 --       megafunction partner, remains with Altera, the megafunction
       partner, or
```

```
34 --       their respective licensors. No other licenses, including any
       licenses
35 --       needed under any third party's intellectual property, are
       provided herein.
36
37 LIBRARY ieee;
38 USE ieee.std_logic_1164.all;
39
40 ENTITY mult IS
41         PORT
42         (
43                 dataa           : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
44                 datab           : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
45                 result          : OUT STD_LOGIC_VECTOR (15 DOWNTO 0)
46         );
47 END mult;
48
49
50 ARCHITECTURE SYN OF mult IS
51
52         SIGNAL sub_wire0        : STD_LOGIC_VECTOR (15 DOWNTO 0);
53
54
55
56         COMPONENT lpm_mult
57         GENERIC (
58                 lpm_widtha              : NATURAL;
59                 lpm_widthb              : NATURAL;
60                 lpm_widthp              : NATURAL;
61                 lpm_widths              : NATURAL;
62                 input_b_is_constant             : STRING;
63                 lpm_representation              : STRING;
64                 use_eab         : STRING;
65                 maximize_speed          : NATURAL
66         );
67         PORT (
68                         dataa   : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
69                         datab   : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
70                         result  : OUT STD_LOGIC_VECTOR (15 DOWNTO 0)
71         );
72         END COMPONENT;
73
74 BEGIN
75         result    <= sub_wire0(15 DOWNTO 0);
76
77         lpm_mult_component : lpm_mult
78         GENERIC MAP (
79                 LPM_WIDTHA => 8,
80                 LPM_WIDTHB => 8,
81                 LPM_WIDTHP => 16,
82                 LPM_WIDTHS => 16,
```

```
83                      INPUT_B_IS_CONSTANT => "NO",
84                      LPM_REPRESENTATION => "SIGNED",
85                      USE_EAB => "OFF",
86                      MAXIMIZE_SPEED => 1
87              )
88              PORT MAP (
89                      dataa => dataa,
90                      datab => datab,
91                      result => sub_wire0
92              );
93
94
95
96  END SYN;
97
98  -- ============================================================
99  -- CNX file retrieval info
100 -- ============================================================
101 -- Retrieval info: PRIVATE: WidthA NUMERIC "8"
102 -- Retrieval info: PRIVATE: WidthB NUMERIC "8"
103 -- Retrieval info: PRIVATE: WidthS NUMERIC "16"
104 -- Retrieval info: PRIVATE: WidthP NUMERIC "16"
105 -- Retrieval info: PRIVATE: OptionalSum NUMERIC "0"
106 -- Retrieval info: PRIVATE: AutoSizeResult NUMERIC "1"
107 -- Retrieval info: PRIVATE: B_isConstant NUMERIC "0"
108 -- Retrieval info: PRIVATE: SignedMult NUMERIC "1"
109 -- Retrieval info: PRIVATE: USE_EAB NUMERIC "0"
110 -- Retrieval info: PRIVATE: ConstantB NUMERIC "0"
111 -- Retrieval info: PRIVATE: ValidConstant NUMERIC "1"
112 -- Retrieval info: PRIVATE: Latency NUMERIC "0"
113 -- Retrieval info: PRIVATE: aclr NUMERIC "0"
114 -- Retrieval info: PRIVATE: LPM_PIPELINE NUMERIC "1"
115 -- Retrieval info: PRIVATE: optimize NUMERIC "2"
116 -- Retrieval info: CONSTANT: LPM_WIDTHA NUMERIC "8"
117 -- Retrieval info: CONSTANT: LPM_WIDTHB NUMERIC "8"
118 -- Retrieval info: CONSTANT: LPM_WIDTHP NUMERIC "16"
119 -- Retrieval info: CONSTANT: LPM_WIDTHS NUMERIC "16"
120 -- Retrieval info: CONSTANT: INPUT_B_IS_CONSTANT STRING "NO"
121 -- Retrieval info: CONSTANT: LPM_REPRESENTATION STRING "SIGNED"
122 -- Retrieval info: CONSTANT: USE_EAB STRING "OFF"
123 -- Retrieval info: CONSTANT: MAXIMIZE_SPEED NUMERIC "1"
124 -- Retrieval info: USED_PORT: dataa 0 0 8 0 INPUT NODEFVAL dataa[7..0]
125 -- Retrieval info: USED_PORT: result 0 0 16 0 OUTPUT NODEFVAL result
        [15..0]
126 -- Retrieval info: USED_PORT: datab 0 0 8 0 INPUT NODEFVAL datab[7..0]
127 -- Retrieval info: CONNECT: @dataa 0 0 8 0 dataa 0 0 8 0
128 -- Retrieval info: CONNECT: result 0 0 16 0 @result 0 0 16 0
129 -- Retrieval info: CONNECT: @datab 0 0 8 0 datab 0 0 8 0
```

A.1.9 synchronizer.vhd

```
1  ------------------------------------------------------------------------------
2  --
3  -- synchronizer.vhd: multi-input synchronizer
4  -- Dan R. K. Ports   <drkp@mit.edu>
5  -- 6.111 final project, 2003/12/07
6  --
7  ------------------------------------------------------------------------------
8
9  library ieee;
10 use ieee.std_logic_1164.all;
11
12 entity synchronizer is
13   generic (
14     width : integer := 8);
15   port (
16     clk         : in  std_logic;
17     syncin      : in  std_logic_vector(width-1 downto 0),
18     syncout     : out std_logic_vector(width-1 downto 0));
19 end synchronizer;
20
21 architecture synchronizer of synchronizer is
22
23 begin  -- synchronizer
24
25   -- purpose: synchronize output on rising edge of clock
26   -- type    : combinational
27   -- inputs : clk
28   -- outputs: syncout
29   sync: process (clk)
30   begin  -- process sync
31     if rising_edge(clk) then
32       syncout <= syncin;
33     end if;
34   end process sync;
35
36 end synchronizer;
```

A.1.10 synchronizer1.vhd

```
1  ------------------------------------------------------------------------------
2  --
3  -- synchronizer1.vhd: 1-input synchronizer
4  -- Dan R. K. Ports   <drkp@mit.edu>
5  -- 6.111 Lab 2, 2003/10/01
6  -- Modified for 6.111 final project, 2003/12/08
7  --
8  ------------------------------------------------------------------------------
9
10 library ieee;
11 use ieee.std_logic_1164.all;
12
13 entity synchronizer1 is
14   port (
15     clk, syncin : in  std_logic;
16     syncout     : out std_logic);
17 end synchronizer1;
18
19 architecture synchronizer1 of synchronizer1 is
20
21 begin  -- synchronizer1
22
23   -- purpose: synchronize output on rising edge of clock
24   -- type   : combinational
25   -- inputs : clk
26   -- outputs: syncout
27   sync: process (clk)
28   begin  -- process sync
29     if rising_edge(clk) then
30       syncout <= syncin;
31     end if;
32   end process sync;
33
34 end synchronizer1;
```

A.1.11 twiddler8.vhd

```
1  -- twiddler8.vhd: 8-bit twiddle factor generator
2  -- Dan R. K. Ports <drkp@mit.edu>
3  -- 6.111 final project, 2003/12/05
4
5  library ieee;
6  use ieee.std_logic_1164.all;
7  use ieee.std_logic_arith.all;
8  use ieee.std_logic_signed.all;
9
10 entity twiddler8 is
11
12   port (
13     twiddlesel : in  std_logic_vector(8 downto 0);
14     twiddlere, twiddleim  : out std_logic_vector(7 downto 0));
15
16 end twiddler8;
17
18 architecture structural of twiddler8 is
19
20   component twiddlerom8c
21         PORT
22         (
23                 address         : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
24                 q               : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
25         );
26   end component;
27
28
29   component twiddlerom8s
30         PORT
31         (
32                 address         : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
33                 q               : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
34         );
35   end component;
36
37
38 begin  -- structural
39
40   sinrom : twiddlerom8s port map (
41     address => twiddlesel(2 downto 0),
42     q       => twiddleim);
43
44   cosrom : twiddlerom8c port map (
45     address => twiddlesel(2 downto 0),
46     q       => twiddlere);
47
48 end structural;
```

A.1.12 twiddler16.vhd

```
1  -- twiddler16.vhd: 16-bit twiddle factor generator
2  -- Dan R. K. Ports <drkp@mit.edu>
3  -- 6.111 final project, 2003/12/05
4
5  library ieee;
6  use ieee.std_logic_1164.all;
7  use ieee.std_logic_arith.all;
8  use ieee.std_logic_signed.all;
9
10 entity twiddler16 is
11
12   port (
13     twiddlesel : in  std_logic_vector(8 downto 0);
14     twiddlere, twiddleim  : out std_logic_vector(7 downto 0));
15
16 end twiddler16;
17
18 architecture structural of twiddler16 is
19
20   component twiddlerom16c
21         PORT
22         (
23                 address         : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
24                 q               : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
25         );
26   end component;
27
28
29   component twiddlerom16s
30         PORT
31         (
32                 address         : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
33                 q               : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
34         );
35   end component;
36
37
38 begin  -- structural
39
40   sinrom : twiddlerom16s port map (
41     address => twiddlesel(3 downto 0),
42     q       => twiddleim);
43
44   cosrom : twiddlerom16c port map (
45     address => twiddlesel(3 downto 0),
46     q       => twiddlere);
47
48 end structural;
```

A.1.13 twiddler512.vhd

```
1  -- twiddler512.vhd: 512 twiddle factor generator
2  -- Dan R. K. Ports <drkp@mit.edu>
3  -- 6.111 final project, 2003/12/05
4
5  library ieee;
6  use ieee.std_logic_1164.all;
7  use ieee.std_logic_arith.all;
8  use ieee.std_logic_signed.all;
9
10 entity twiddler512 is
11
12   port (
13     twiddlesel : in  std_logic_vector(8 downto 0);
14     twiddlere, twiddleim  : out std_logic_vector(7 downto 0));
15
16 end twiddler512;
17
18 architecture structural of twiddler512 is
19
20   component twiddlerom512c
21         PORT
22         (
23                 address         : IN STD_LOGIC_VECTOR (8 DOWNTO 0);
24                 q               : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
25         );
26   end component;
27
28
29   component twiddlerom512s
30         PORT
31         (
32                 address         : IN STD_LOGIC_VECTOR (8 DOWNTO 0);
33                 q               : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
34         );
35   end component;
36
37
38 begin  -- structural
39
40   sinrom : twiddlerom512s port map (
41     address => twiddlesel,
42     q       => twiddleim);
43
44   cosrom : twiddlerom512c port map (
45     address => twiddlesel,
46     q       => twiddlere);
47
48 end structural;
```

A.1.14 twiddlerom8c.vhd

```
1  -- megafunction wizard: %LPM_ROM%
2  -- GENERATION: STANDARD
3  -- VERSION: WM1.0
4  -- MODULE: lpm_rom
5
6  --============================================================
7  -- File Name: twiddlerom8c.vhd
8  -- Megafunction Name(s):
9  --                         lpm_rom
10 --============================================================
11 -- ************************************************************
12 -- THIS IS A WIZARD GENERATED FILE. DO NOT EDIT THIS FILE!
13 -- ************************************************************
14
15
16 --       Copyright (C) 1988-2002 Altera Corporation
17
18 --       Any megafunction design, and related net list (encrypted or
       decrypted),
19 --       support information, device programming or simulation file, and
       any other
20 --       associated documentation or information provided by Altera or a
       partner
21 --       under Altera's Megafunction Partnership Program may be used only
        to
22 --       program PLD devices (but not masked PLD devices) from Altera.
        Any other
23 --       use of such megafunction design, net list, support information,
       device
24 --       programming or simulation file, or any other related
       documentation or
25 --       information is prohibited for any other purpose, including, but
       not
26 --       limited to modification, reverse engineering, de-compiling, or
       use with
27 --       any other silicon devices, unless such use is explicitly
       licensed under
28 --       a separate agreement with Altera or a megafunction partner.
        Title to
29 --       the intellectual property, including patents, copyrights,
       trademarks,
30 --       trade secrets, or maskworks, embodied in any such megafunction
       design,
31 --       net list, support information, device programming or simulation
       file, or
32 --       any other related documentation or information provided by
       Altera or a
33 --       megafunction partner, remains with Altera, the megafunction
       partner, or
```

```
34 --       their respective licensors.  No other licenses, including any
       licenses
35 --       needed under any third party's intellectual property, are
       provided herein.
36
37 LIBRARY ieee;
38 USE ieee.std_logic_1164.all;
39
40 ENTITY twiddlerom8c IS
41         PORT
42         (
43                 address         : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
44                 q               : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
45         );
46 END twiddlerom8c;
47
48
49 ARCHITECTURE SYN OF twiddlerom8c IS
50
51         SIGNAL sub_wire0        : STD_LOGIC_VECTOR (7 DOWNTO 0);
52
53
54
55         COMPONENT lpm_rom
56         GENERIC (
57                 lpm_width               : NATURAL;
58                 lpm_widthad             : NATURAL;
59                 lpm_address_control             : STRING;
60                 lpm_outdata             : STRING;
61                 lpm_file                : STRING
62         );
63         PORT (
64                         address : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
65                         q       : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
66         );
67         END COMPONENT;
68
69 BEGIN
70         q     <= sub_wire0(7 DOWNTO 0);
71
72         lpm_rom_component : lpm_rom
73         GENERIC MAP (
74                 LPM_WIDTH => 8,
75                 LPM_WIDTHAD => 3,
76                 LPM_ADDRESS_CONTROL => "UNREGISTERED",
77                 LPM_OUTDATA => "UNREGISTERED",
78                 LPM_FILE => "/afs/athena.mit.edu/user/d/r/drkp/Private
                       /6.111/proj/twiddlegenerator/twiddle8c.ntl"
79         )
80         PORT MAP (
81                 address => address,
```

```
82                      q => sub_wire0
83              );
84
85
86
87  END SYN;
88
89  -- ============================================================
90  -- CNX file retrieval info
91  -- ============================================================
92  -- Retrieval info: PRIVATE: WidthData NUMERIC "8"
93  -- Retrieval info: PRIVATE: WidthAddr NUMERIC "3"
94  -- Retrieval info: PRIVATE: RegAdd NUMERIC "0"
95  -- Retrieval info: PRIVATE: OutputRegistered NUMERIC "0"
96  -- Retrieval info: PRIVATE: MIFfilename STRING "/afs/athena.mit.edu/user
        /d/r/drkp/Private/6.111/proj/twiddlegenerator/twiddle8c.ntl"
97  -- Retrieval info: CONSTANT: LPM_WIDTH NUMERIC "8"
98  -- Retrieval info: CONSTANT: LPM_WIDTHAD NUMERIC "3"
99  -- Retrieval info: CONSTANT: LPM_ADDRESS_CONTROL STRING "UNREGISTERED"
100 -- Retrieval info: CONSTANT: LPM_OUTDATA STRING "UNREGISTERED"
101 -- Retrieval info: CONSTANT: LPM_FILE STRING "/afs/athena.mit.edu/user/d
        /r/drkp/Private/6.111/proj/twiddlegenerator/twiddle8c.ntl"
102 -- Retrieval info: USED_PORT: address 0 0 3 0 INPUT NODEFVAL address
        [2..0]
103 -- Retrieval info: USED_PORT: q 0 0 8 0 OUTPUT NODEFVAL q[7..0]
104 -- Retrieval info: CONNECT: @address 0 0 3 0 address 0 0 3 0
105 -- Retrieval info: CONNECT: q 0 0 8 0 @q 0 0 8 0
```

A.1.15 twiddlerom8s.vhd

```
1  -- megafunction wizard: %LPM_ROM%
2  -- GENERATION: STANDARD
3  -- VERSION: WM1.0
4  -- MODULE: lpm_rom
5
6  --============================================================
7  -- File Name: twiddlerom8c.vhd
8  -- Megafunction Name(s):
9  --                         lpm_rom
10 --============================================================
11 -- ************************************************************
12 -- THIS IS A WIZARD GENERATED FILE. DO NOT EDIT THIS FILE!
13 -- ************************************************************
14
15
16 --       Copyright (C) 1988-2002 Altera Corporation
17
18 --       Any megafunction design, and related net list (encrypted or
       decrypted),
19 --       support information, device programming or simulation file, and
       any other
20 --       associated documentation or information provided by Altera or a
       partner
21 --       under Altera's Megafunction Partnership Program may be used only
        to
22 --       program PLD devices (but not masked PLD devices) from Altera.
        Any other
23 --       use of such megafunction design, net list, support information,
       device
24 --       programming or simulation file, or any other related
       documentation or
25 --       information is prohibited for any other purpose, including, but
       not
26 --       limited to modification, reverse engineering, de-compiling, or
       use with
27 --       any other silicon devices, unless such use is explicitly
       licensed under
28 --       a separate agreement with Altera or a megafunction partner.
        Title to
29 --       the intellectual property, including patents, copyrights,
       trademarks,
30 --       trade secrets, or maskworks, embodied in any such megafunction
       design,
31 --       net list, support information, device programming or simulation
       file, or
32 --       any other related documentation or information provided by
       Altera or a
33 --       megafunction partner, remains with Altera, the megafunction
       partner, or
```

```
34 --       their respective licensors.  No other licenses, including any
       licenses
35 --       needed under any third party's intellectual property, are
       provided herein.
36
37 LIBRARY ieee;
38 USE ieee.std_logic_1164.all;
39
40 ENTITY twiddlerom8c IS
41         PORT
42         (
43                 address         : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
44                 q               : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
45         );
46 END twiddlerom8c;
47
48
49 ARCHITECTURE SYN OF twiddlerom8c IS
50
51         SIGNAL sub_wire0        : STD_LOGIC_VECTOR (7 DOWNTO 0);
52
53
54
55         COMPONENT lpm_rom
56         GENERIC (
57                 lpm_width               : NATURAL;
58                 lpm_widthad             : NATURAL;
59                 lpm_address_control             : STRING;
60                 lpm_outdata             : STRING;
61                 lpm_file                : STRING
62         );
63         PORT (
64                         address : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
65                         q       : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
66         );
67         END COMPONENT;
68
69 BEGIN
70         q     <= sub_wire0(7 DOWNTO 0);
71
72         lpm_rom_component : lpm_rom
73         GENERIC MAP (
74                 LPM_WIDTH => 8,
75                 LPM_WIDTHAD => 3,
76                 LPM_ADDRESS_CONTROL => "UNREGISTERED",
77                 LPM_OUTDATA => "UNREGISTERED",
78                 LPM_FILE => "/afs/athena.mit.edu/user/d/r/drkp/Private
                       /6.111/proj/twiddlegenerator/twiddle8c.ntl"
79         )
80         PORT MAP (
81                 address => address,
```

```
82                      q => sub_wire0
83              );
84
85
86
87  END SYN;
88
89  -- ============================================================
90  -- CNX file retrieval info
91  -- ============================================================
92  -- Retrieval info: PRIVATE: WidthData NUMERIC "8"
93  -- Retrieval info: PRIVATE: WidthAddr NUMERIC "3"
94  -- Retrieval info: PRIVATE: RegAdd NUMERIC "0"
95  -- Retrieval info: PRIVATE: OutputRegistered NUMERIC "0"
96  -- Retrieval info: PRIVATE: MIFfilename STRING "/afs/athena.mit.edu/user
        /d/r/drkp/Private/6.111/proj/twiddlegenerator/twiddle8c.ntl"
97  -- Retrieval info: CONSTANT: LPM_WIDTH NUMERIC "8"
98  -- Retrieval info: CONSTANT: LPM_WIDTHAD NUMERIC "3"
99  -- Retrieval info: CONSTANT: LPM_ADDRESS_CONTROL STRING "UNREGISTERED"
100 -- Retrieval info: CONSTANT: LPM_OUTDATA STRING "UNREGISTERED"
101 -- Retrieval info: CONSTANT: LPM_FILE STRING "/afs/athena.mit.edu/user/d
        /r/drkp/Private/6.111/proj/twiddlegenerator/twiddle8c.ntl"
102 -- Retrieval info: USED_PORT: address 0 0 3 0 INPUT NODEFVAL address
        [2..0]
103 -- Retrieval info: USED_PORT: q 0 0 8 0 OUTPUT NODEFVAL q[7..0]
104 -- Retrieval info: CONNECT: @address 0 0 3 0 address 0 0 3 0
105 -- Retrieval info: CONNECT: q 0 0 8 0 @q 0 0 8 0
```

A.1.16 twiddlerom16c.vhd

```
1  -- megafunction wizard: %LPM_ROM%
2  -- GENERATION: STANDARD
3  -- VERSION: WM1.0
4  -- MODULE: lpm_rom
5
6  --============================================================
7  -- File Name: twiddlerom16c.vhd
8  -- Megafunction Name(s):
9  --                         lpm_rom
10 --============================================================
11 -- ************************************************************
12 -- THIS IS A WIZARD GENERATED FILE. DO NOT EDIT THIS FILE!
13 -- ************************************************************
14
15
16 --       Copyright (C) 1988-2002 Altera Corporation
17
18 --       Any megafunction design, and related net list (encrypted or
       decrypted),
19 --       support information, device programming or simulation file, and
       any other
20 --       associated documentation or information provided by Altera or a
       partner
21 --       under Altera's Megafunction Partnership Program may be used only
        to
22 --       program PLD devices (but not masked PLD devices) from Altera.
        Any other
23 --       use of such megafunction design, net list, support information,
       device
24 --       programming or simulation file, or any other related
       documentation or
25 --       information is prohibited for any other purpose, including, but
       not
26 --       limited to modification, reverse engineering, de-compiling, or
       use with
27 --       any other silicon devices, unless such use is explicitly
       licensed under
28 --       a separate agreement with Altera or a megafunction partner.
        Title to
29 --       the intellectual property, including patents, copyrights,
       trademarks,
30 --       trade secrets, or maskworks, embodied in any such megafunction
       design,
31 --       net list, support information, device programming or simulation
       file, or
32 --       any other related documentation or information provided by
       Altera or a
33 --       megafunction partner, remains with Altera, the megafunction
       partner, or
```

```
34 --       their respective licensors.  No other licenses, including any
       licenses
35 --       needed under any third party's intellectual property, are
       provided herein.
36
37 LIBRARY ieee;
38 USE ieee.std_logic_1164.all;
39
40 ENTITY twiddlerom16c IS
41         PORT
42         (
43                 address         : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
44                 q               : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
45         );
46 END twiddlerom16c;
47
48
49 ARCHITECTURE SYN OF twiddlerom16c IS
50
51         SIGNAL sub_wire0        : STD_LOGIC_VECTOR (7 DOWNTO 0);
52
53
54
55         COMPONENT lpm_rom
56         GENERIC (
57                 lpm_width               : NATURAL;
58                 lpm_widthad             : NATURAL;
59                 lpm_address_control             : STRING;
60                 lpm_outdata             : STRING;
61                 lpm_file                : STRING
62         );
63         PORT (
64                         address : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
65                         q       : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
66         );
67         END COMPONENT;
68
69 BEGIN
70         q    <= sub_wire0(7 DOWNTO 0);
71
72         lpm_rom_component : lpm_rom
73         GENERIC MAP (
74                 LPM_WIDTH => 8,
75                 LPM_WIDTHAD => 4,
76                 LPM_ADDRESS_CONTROL => "UNREGISTERED",
77                 LPM_OUTDATA => "UNREGISTERED",
78                 LPM_FILE => "/afs/athena.mit.edu/user/d/r/drkp/Private
                       /6.111/proj/twiddlegenerator/twiddle16c.ntl"
79         )
80         PORT MAP (
81                 address => address,
```

```
82                      q => sub_wire0
83              );
84
85
86
87  END SYN;
88
89  -- ============================================================
90  -- CNX file retrieval info
91  -- ============================================================
92  -- Retrieval info: PRIVATE: WidthData NUMERIC "8"
93  -- Retrieval info: PRIVATE: WidthAddr NUMERIC "4"
94  -- Retrieval info: PRIVATE: RegAdd NUMERIC "0"
95  -- Retrieval info: PRIVATE: OutputRegistered NUMERIC "0"
96  -- Retrieval info: PRIVATE: MIFfilename STRING "/afs/athena.mit.edu/user
        /d/r/drkp/Private/6.111/proj/twiddlegenerator/twiddle16c.ntl"
97  -- Retrieval info: CONSTANT: LPM_WIDTH NUMERIC "8"
98  -- Retrieval info: CONSTANT: LPM_WIDTHAD NUMERIC "4"
99  -- Retrieval info: CONSTANT: LPM_ADDRESS_CONTROL STRING "UNREGISTERED"
100 -- Retrieval info: CONSTANT: LPM_OUTDATA STRING "UNREGISTERED"
101 -- Retrieval info: CONSTANT: LPM_FILE STRING "/afs/athena.mit.edu/user/d
        /r/drkp/Private/6.111/proj/twiddlegenerator/twiddle16c.ntl"
102 -- Retrieval info: USED_PORT: address 0 0 4 0 INPUT NODEFVAL address
        [3..0]
103 -- Retrieval info: USED_PORT: q 0 0 8 0 OUTPUT NODEFVAL q[7..0]
104 -- Retrieval info: CONNECT: @address 0 0 4 0 address 0 0 4 0
105 -- Retrieval info: CONNECT: q 0 0 8 0 @q 0 0 8 0
```

A.1.17 twiddlerom16s.vhd

```
1  -- megafunction wizard: %LPM_ROM%
2  -- GENERATION: STANDARD
3  -- VERSION: WM1.0
4  -- MODULE: lpm_rom
5
6  --============================================================
7  -- File Name: twiddlerom16c.vhd
8  -- Megafunction Name(s):
9  --                         lpm_rom
10 --============================================================
11 -- ************************************************************
12 -- THIS IS A WIZARD GENERATED FILE. DO NOT EDIT THIS FILE!
13 -- ************************************************************
14
15
16 --      Copyright (C) 1988-2002 Altera Corporation
17
18 --      Any megafunction design, and related net list (encrypted or
       decrypted),
19 --      support information, device programming or simulation file, and
       any other
20 --      associated documentation or information provided by Altera or a
       partner
21 --      under Altera's Megafunction Partnership Program may be used only
        to
22 --      program PLD devices (but not masked PLD devices) from Altera.
        Any other
23 --      use of such megafunction design, net list, support information,
       device
24 --      programming or simulation file, or any other related
       documentation or
25 --      information is prohibited for any other purpose, including, but
       not
26 --      limited to modification, reverse engineering, de-compiling, or
       use with
27 --      any other silicon devices, unless such use is explicitly
       licensed under
28 --      a separate agreement with Altera or a megafunction partner.
        Title to
29 --      the intellectual property, including patents, copyrights,
       trademarks,
30 --      trade secrets, or maskworks, embodied in any such megafunction
       design,
31 --      net list, support information, device programming or simulation
       file, or
32 --      any other related documentation or information provided by
       Altera or a
33 --      megafunction partner, remains with Altera, the megafunction
       partner, or
```

```
34 --       their respective licensors.  No other licenses, including any
       licenses
35 --       needed under any third party's intellectual property, are
       provided herein.
36
37 LIBRARY ieee;
38 USE ieee.std_logic_1164.all;
39
40 ENTITY twiddlerom16c IS
41         PORT
42         (
43                 address         : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
44                 q               : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
45         );
46 END twiddlerom16c;
47
48
49 ARCHITECTURE SYN OF twiddlerom16c IS
50
51         SIGNAL sub_wire0        : STD_LOGIC_VECTOR (7 DOWNTO 0);
52
53
54
55         COMPONENT lpm_rom
56         GENERIC (
57                 lpm_width               : NATURAL;
58                 lpm_widthad             : NATURAL;
59                 lpm_address_control             : STRING;
60                 lpm_outdata             : STRING;
61                 lpm_file                : STRING
62         );
63         PORT (
64                         address : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
65                         q       : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
66         );
67         END COMPONENT;
68
69 BEGIN
70         q    <= sub_wire0(7 DOWNTO 0);
71
72         lpm_rom_component : lpm_rom
73         GENERIC MAP (
74                 LPM_WIDTH => 8,
75                 LPM_WIDTHAD => 4,
76                 LPM_ADDRESS_CONTROL => "UNREGISTERED",
77                 LPM_OUTDATA => "UNREGISTERED",
78                 LPM_FILE => "/afs/athena.mit.edu/user/d/r/drkp/Private
                       /6.111/proj/twiddlegenerator/twiddle16c.ntl"
79         )
80         PORT MAP (
81                 address => address,
```

```
82                      q => sub_wire0
83              );
84
85
86
87  END SYN;
88
89  -- ============================================================
90  -- CNX file retrieval info
91  -- ============================================================
92  -- Retrieval info: PRIVATE: WidthData NUMERIC "8"
93  -- Retrieval info: PRIVATE: WidthAddr NUMERIC "4"
94  -- Retrieval info: PRIVATE: RegAdd NUMERIC "0"
95  -- Retrieval info: PRIVATE: OutputRegistered NUMERIC "0"
96  -- Retrieval info: PRIVATE: MIFfilename STRING "/afs/athena.mit.edu/user
        /d/r/drkp/Private/6.111/proj/twiddlegenerator/twiddle16c.ntl"
97  -- Retrieval info: CONSTANT: LPM_WIDTH NUMERIC "8"
98  -- Retrieval info: CONSTANT: LPM_WIDTHAD NUMERIC "4"
99  -- Retrieval info: CONSTANT: LPM_ADDRESS_CONTROL STRING "UNREGISTERED"
100 -- Retrieval info: CONSTANT: LPM_OUTDATA STRING "UNREGISTERED"
101 -- Retrieval info: CONSTANT: LPM_FILE STRING "/afs/athena.mit.edu/user/d
        /r/drkp/Private/6.111/proj/twiddlegenerator/twiddle16c.ntl"
102 -- Retrieval info: USED_PORT: address 0 0 4 0 INPUT NODEFVAL address
        [3..0]
103 -- Retrieval info: USED_PORT: q 0 0 8 0 OUTPUT NODEFVAL q[7..0]
104 -- Retrieval info: CONNECT: @address 0 0 4 0 address 0 0 4 0
105 -- Retrieval info: CONNECT: q 0 0 8 0 @q 0 0 8 0
```

A.1.18 twiddlerom512c.vhd

```
1  -- megafunction wizard: %LPM_ROM%
2  -- GENERATION: STANDARD
3  -- VERSION: WM1.0
4  -- MODULE: lpm_rom
5
6  --============================================================
7  -- File Name: twiddlerom16c.vhd
8  -- Megafunction Name(s):
9  --                         lpm_rom
10 --============================================================
11 -- ************************************************************
12 -- THIS IS A WIZARD GENERATED FILE. DO NOT EDIT THIS FILE!
13 -- ************************************************************
14
15
16 --       Copyright (C) 1988-2002 Altera Corporation
17
18 --       Any megafunction design, and related net list (encrypted or
       decrypted),
19 --       support information, device programming or simulation file, and
       any other
20 --       associated documentation or information provided by Altera or a
       partner
21 --       under Altera's Megafunction Partnership Program may be used only
        to
22 --       program PLD devices (but not masked PLD devices) from Altera.
        Any other
23 --       use of such megafunction design, net list, support information,
       device
24 --       programming or simulation file, or any other related
       documentation or
25 --       information is prohibited for any other purpose, including, but
       not
26 --       limited to modification, reverse engineering, de-compiling, or
       use with
27 --       any other silicon devices, unless such use is explicitly
       licensed under
28 --       a separate agreement with Altera or a megafunction partner.
        Title to
29 --       the intellectual property, including patents, copyrights,
       trademarks,
30 --       trade secrets, or maskworks, embodied in any such megafunction
       design,
31 --       net list, support information, device programming or simulation
       file, or
32 --       any other related documentation or information provided by
       Altera or a
33 --       megafunction partner, remains with Altera, the megafunction
       partner, or
```

```
34 --       their respective licensors.  No other licenses, including any
       licenses
35 --       needed under any third party's intellectual property, are
       provided herein.
36
37 LIBRARY ieee;
38 USE ieee.std_logic_1164.all;
39
40 ENTITY twiddlerom16c IS
41         PORT
42         (
43                 address         : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
44                 q               : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
45         );
46 END twiddlerom16c;
47
48
49 ARCHITECTURE SYN OF twiddlerom16c IS
50
51         SIGNAL sub_wire0        : STD_LOGIC_VECTOR (7 DOWNTO 0);
52
53
54
55         COMPONENT lpm_rom
56         GENERIC (
57                 lpm_width               : NATURAL;
58                 lpm_widthad             : NATURAL;
59                 lpm_address_control             : STRING;
60                 lpm_outdata             : STRING;
61                 lpm_file                : STRING
62         );
63         PORT (
64                         address : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
65                         q       : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
66         );
67         END COMPONENT;
68
69 BEGIN
70         q    <= sub_wire0(7 DOWNTO 0);
71
72         lpm_rom_component : lpm_rom
73         GENERIC MAP (
74                 LPM_WIDTH => 8,
75                 LPM_WIDTHAD => 4,
76                 LPM_ADDRESS_CONTROL => "UNREGISTERED",
77                 LPM_OUTDATA => "UNREGISTERED",
78                 LPM_FILE => "/afs/athena.mit.edu/user/d/r/drkp/Private
                       /6.111/proj/twiddlegenerator/twiddle16c.ntl"
79         )
80         PORT MAP (
81                 address => address,
```

```
82                      q => sub_wire0
83              );
84
85
86
87  END SYN;
88
89  -- ============================================================
90  -- CNX file retrieval info
91  -- ============================================================
92  -- Retrieval info: PRIVATE: WidthData NUMERIC "8"
93  -- Retrieval info: PRIVATE: WidthAddr NUMERIC "4"
94  -- Retrieval info: PRIVATE: RegAdd NUMERIC "0"
95  -- Retrieval info: PRIVATE: OutputRegistered NUMERIC "0"
96  -- Retrieval info: PRIVATE: MIFfilename STRING "/afs/athena.mit.edu/user
        /d/r/drkp/Private/6.111/proj/twiddlegenerator/twiddle16c.ntl"
97  -- Retrieval info: CONSTANT: LPM_WIDTH NUMERIC "8"
98  -- Retrieval info: CONSTANT: LPM_WIDTHAD NUMERIC "4"
99  -- Retrieval info: CONSTANT: LPM_ADDRESS_CONTROL STRING "UNREGISTERED"
100 -- Retrieval info: CONSTANT: LPM_OUTDATA STRING "UNREGISTERED"
101 -- Retrieval info: CONSTANT: LPM_FILE STRING "/afs/athena.mit.edu/user/d
        /r/drkp/Private/6.111/proj/twiddlegenerator/twiddle16c.ntl"
102 -- Retrieval info: USED_PORT: address 0 0 4 0 INPUT NODEFVAL address
        [3..0]
103 -- Retrieval info: USED_PORT: q 0 0 8 0 OUTPUT NODEFVAL q[7..0]
104 -- Retrieval info: CONNECT: @address 0 0 4 0 address 0 0 4 0
105 -- Retrieval info: CONNECT: q 0 0 8 0 @q 0 0 8 0
```

A.1.19 twiddlerom512s.vhd

```
1  -- megafunction wizard: %LPM_ROM%
2  -- GENERATION: STANDARD
3  -- VERSION: WM1.0
4  -- MODULE: lpm_rom
5
6  --============================================================
7  -- File Name: twiddlerom16c.vhd
8  -- Megafunction Name(s):
9  --                         lpm_rom
10 --============================================================
11 -- ************************************************************
12 -- THIS IS A WIZARD GENERATED FILE. DO NOT EDIT THIS FILE!
13 -- ************************************************************
14
15
16 --       Copyright (C) 1988-2002 Altera Corporation
17
18 --       Any megafunction design, and related net list (encrypted or
       decrypted),
19 --       support information, device programming or simulation file, and
       any other
20 --       associated documentation or information provided by Altera or a
       partner
21 --       under Altera's Megafunction Partnership Program may be used only
        to
22 --       program PLD devices (but not masked PLD devices) from Altera.
        Any other
23 --       use of such megafunction design, net list, support information,
       device
24 --       programming or simulation file, or any other related
       documentation or
25 --       information is prohibited for any other purpose, including, but
       not
26 --       limited to modification, reverse engineering, de-compiling, or
       use with
27 --       any other silicon devices, unless such use is explicitly
       licensed under
28 --       a separate agreement with Altera or a megafunction partner.
        Title to
29 --       the intellectual property, including patents, copyrights,
       trademarks,
30 --       trade secrets, or maskworks, embodied in any such megafunction
       design,
31 --       net list, support information, device programming or simulation
       file, or
32 --       any other related documentation or information provided by
       Altera or a
33 --       megafunction partner, remains with Altera, the megafunction
       partner, or
```

```
34 --       their respective licensors.  No other licenses, including any
       licenses
35 --       needed under any third party's intellectual property, are
       provided herein.
36
37 LIBRARY ieee;
38 USE ieee.std_logic_1164.all;
39
40 ENTITY twiddlerom16c IS
41         PORT
42         (
43                 address         : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
44                 q               : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
45         );
46 END twiddlerom16c;
47
48
49 ARCHITECTURE SYN OF twiddlerom16c IS
50
51         SIGNAL sub_wire0        : STD_LOGIC_VECTOR (7 DOWNTO 0);
52
53
54
55         COMPONENT lpm_rom
56         GENERIC (
57                 lpm_width               : NATURAL;
58                 lpm_widthad             : NATURAL;
59                 lpm_address_control             : STRING;
60                 lpm_outdata             : STRING;
61                 lpm_file                : STRING
62         );
63         PORT (
64                         address : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
65                         q       : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
66         );
67         END COMPONENT;
68
69 BEGIN
70         q     <= sub_wire0(7 DOWNTO 0);
71
72         lpm_rom_component : lpm_rom
73         GENERIC MAP (
74                 LPM_WIDTH => 8,
75                 LPM_WIDTHAD => 4,
76                 LPM_ADDRESS_CONTROL => "UNREGISTERED",
77                 LPM_OUTDATA => "UNREGISTERED",
78                 LPM_FILE => "/afs/athena.mit.edu/user/d/r/drkp/Private
                       /6.111/proj/twiddlegenerator/twiddle16c.ntl"
79         )
80         PORT MAP (
81                 address => address,
```

```
82                      q => sub_wire0
83              );
84
85
86
87  END SYN;
88
89  -- ============================================================
90  -- CNX file retrieval info
91  -- ============================================================
92  -- Retrieval info: PRIVATE: WidthData NUMERIC "8"
93  -- Retrieval info: PRIVATE: WidthAddr NUMERIC "4"
94  -- Retrieval info: PRIVATE: RegAdd NUMERIC "0"
95  -- Retrieval info: PRIVATE: OutputRegistered NUMERIC "0"
96  -- Retrieval info: PRIVATE: MIFfilename STRING "/afs/athena.mit.edu/user
        /d/r/drkp/Private/6.111/proj/twiddlegenerator/twiddle16c.ntl"
97  -- Retrieval info: CONSTANT: LPM_WIDTH NUMERIC "8"
98  -- Retrieval info: CONSTANT: LPM_WIDTHAD NUMERIC "4"
99  -- Retrieval info: CONSTANT: LPM_ADDRESS_CONTROL STRING "UNREGISTERED"
100 -- Retrieval info: CONSTANT: LPM_OUTDATA STRING "UNREGISTERED"
101 -- Retrieval info: CONSTANT: LPM_FILE STRING "/afs/athena.mit.edu/user/d
        /r/drkp/Private/6.111/proj/twiddlegenerator/twiddle16c.ntl"
102 -- Retrieval info: USED_PORT: address 0 0 4 0 INPUT NODEFVAL address
        [3..0]
103 -- Retrieval info: USED_PORT: q 0 0 8 0 OUTPUT NODEFVAL q[7..0]
104 -- Retrieval info: CONNECT: @address 0 0 4 0 address 0 0 4 0
105 -- Retrieval info: CONNECT: q 0 0 8 0 @q 0 0 8 0
```

A.1.20 ffttestjig.vhd

```
1  ----------------------------------------------------------------------------
2  --
3  -- ffttestjig.vhd: test jig for validating FFT processor
4  -- Dan R. K. Ports  <drkp@mit.edu>
5  -- 6.111 final project, 2003/12/05
6  --
7  ----------------------------------------------------------------------------
8
9  library ieee;
10 use ieee.std_logic_1164.all;
11 use ieee.std_logic_arith.all;
12 use ieee.std_logic_signed.all;
13
14 entity ffttestjig is
15
16     port (
17     start, reset, clk          : in    std_logic;
18     busy                       : out   std_logic;
19     inbufsel, outbufsel        : in    std_logic_vector(1 downto 0);
20     treout                     : out   std_logic_vector(7 downto 0);
21     databusinout               : out   std_logic_vector(7 downto 0);
22     databusoutout              : out   std_logic_vector(7 downto 0);
23     inverse                    : in    std_logic;
24     ramwe                      : out   std_logic);
25
26 end ffttestjig;
27
28 architecture structural of ffttestjig is
29
30   component fft
31     port (
32     start, reset, clk          : in    std_logic;
33     testaddrbus                  : out   std_logic_vector(6 downto 0);
34     databusout                  : out std_logic_vector(7 downto 0);
35     databusin                  : in std_logic_vector(7 downto 0);
36     busy                       : out   std_logic;
37     inbufsel, outbufsel        : in    std_logic_vector(1 downto 0);
38     treout                     : out   std_logic_vector(7 downto 0);
39     databusinout               : out   std_logic_vector(7 downto 0);
40     inverse                    : in    std_logic;
41     ramwe                      : out   std_logic);
42
43   end component;
44
45 --   component testram
46 --         PORT
47 --         (
```

```
48 --                   address          : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
49 --                   we               : IN STD_LOGIC  := '1';
50 --                   outenab          : IN STD_LOGIC  := '1';
51 --                   dio              : INOUT STD_LOGIC_VECTOR (13 DOWNTO 0)
52 --         );
53 --   end component;
54 --
55  component testram
56        PORT
57        (
58                address          : IN STD_LOGIC_VECTOR (6 DOWNTO 0);
59                we               : IN STD_LOGIC  := '1';
60                data             : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
61                q                : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
62        );
63 end component;
64
65   signal addrbus : std_logic_vector(6 downto 0);
66   signal databus, d, q : std_logic_vector(7 downto 0);
67   signal fftwe, oe, ramweint : std_logic;
68
69 begin  -- structural
70
71
72   fftinst : fft port map (
73     start      => start,
74     reset      => reset,
75     clk        => clk,
76     testaddrbus    => addrbus,
77     databusout    => d,
78     databusin => q,
79     busy       => busy,
80     inbufsel   => inbufsel,
81     outbufsel => outbufsel,
82     treout     => treout,
83     inverse    => inverse,
84     databusinout => databusinout,
85     ramwe      => fftwe);
86
87   raminst : testram port map (
88     address    => addrbus,
89     data         => d,
90     q    => q,
91     we         => ramweint);
92
93
94   oe <= '1';
95   ramweint <= fftwe and not clk;
96
97   ramwe <= ramweint;
98
```

```
 99     databusoutout <= d;
100
101  end structural;
```

A.1.21 memmgrtestjig.vhd

```
1  ----------------------------------------------------------------------------
2  --
3  -- memmgrtestjig.vhd: test jig for validating memory manager
4  -- Dan R. K. Ports  <drkp@mit.edu>
5  -- 6.111 final project, 2003/12/07
6  --
7  ----------------------------------------------------------------------------
8
9  library ieee;
10 use ieee.std_logic_1164.all;
11 use ieee.std_logic_arith.all;
12 use ieee.std_logic_signed.all;
13
14 entity memmgrtestjig is
15
16 port (
17     aaddr, baddr, caddr : in    std_logic_vector(6 downto 0);
18     adata, bdata, cdata : in    std_logic_vector(7 downto 0) := (others
           => 'Z');
19     aq, bq, cq          : out    std_logic_vector(7 downto 0) := (others
           => 'Z');
20     awr, bwr, cwr       : in    std_logic;
21     clk, reset          : in    std_logic;
22     areq, breq, creq    : in    std_logic;
23     adone, bdone, cdone : out   std_logic);
24 --    ramaddr             : out   std_logic_vector(14-1 downto 0);
25 --    ramdata             : inout std_logic_vector(8-1 downto 0) := (
       others => 'Z');
26 --    ramdatain           : in    std_logic_vector(7 downto 0);
27 --    ramdataout          : out   std_logic_vector(7 downto 0);
28 --    ramwe               : out   std_logic);
29
30 end memmgrtestjig;
31
32 architecture structural of memmgrtestjig is
33
34   component memmgr
35
36   port (
37     aaddr, baddr, caddr : in    std_logic_vector(14-1 downto 0);
38     adata, bdata, cdata : in    std_logic_vector(8-1 downto 0) := (
         others => 'Z');
39     aq, bq, cq          : out    std_logic_vector(8-1 downto 0) := (
         others => 'Z');
40     awr, bwr, cwr       : in    std_logic;
41     clk, reset          : in    std_logic;
42     areq, breq, creq    : in    std_logic;
```

```
43      adone, bdone, cdone : out    std_logic;
44      ramaddr             : out    std_logic_vector(14-1 downto 0);
45  --    ramdata             : inout std_logic_vector(8-1 downto 0) := (
        others => 'Z');
46      ramdatain           : in     std_logic_vector(7 downto 0);
47      ramdataout          : out    std_logic_vector(7 downto 0);
48      ramwe               : out    std_logic);
49
50
51    end component;
52
53
54     component testram
55         PORT
56         (
57                 address          : IN STD_LOGIC_VECTOR (6 DOWNTO 0);
58                 we               : IN STD_LOGIC  := '1';
59                 data             : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
60                 q                : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
61         );
62     end component;
63
64
65    signal address : std_logic_vector(6 downto 0);
66    signal data, q : std_logic_vector(7 downto 0);
67    signal we : std_logic;
68    signal mgraddress : std_logic_vector(13 downto 0);
69    signal aaddre, baddre, caddre : std_logic_vector(13 downto 0);
70
71  begin  -- structural
72
73    aaddre <= "0000000" & aaddr;
74    baddre <= "0000000" & baddr;
75    caddre <= "0000000" & caddr;
76
77    mgr : memmgr port map (
78      aaddr        => aaddre,
79      baddr       =>  baddre,
80      caddr       =>  caddre,
81      adata       => adata,
82      bdata       => bdata,
83      cdata       => cdata,
84      aq          => aq,
85      bq          => bq,
86      cq          => cq,
87      awr         => awr,
88      bwr         => bwr,
89      cwr         => cwr,
90      clk         => clk,
91      reset       => reset,
92      areq        => areq,
```

```
93        breq         => breq,
94        creq         => creq,
95        adone        => adone,
96        bdone        => bdone,
97        cdone        => cdone,
98        ramaddr      => mgraddress,
99        ramdatain    => q,
100       ramdataout => data,
101       ramwe        => we);
102
103     raminst : testram port map (
104       address => address,
105       data     => data,
106       q        => q,
107       we       => we);
108
109     address <= mgraddress(6 downto 0);
110
111   end structural;
```

A.1.22 fftmgrtestjig.vhd

```
1  ----------------------------------------------------------------------------
2  --
3  -- fftmgrtestjig.vhd: test jig for validating FFT processor with memory
       mgr
4  -- Dan R. K. Ports  <drkp@mit.edu>
5  -- 6.111 final project, 2003/12/05
6  --
7  ----------------------------------------------------------------------------
8
9  library ieee;
10 use ieee.std_logic_1164.all;
11 use ieee.std_logic_arith.all;
12 use ieee.std_logic_signed.all;
13
14 entity fftmgrtestjig is
15
16     port (
17     start, reset, clk          : in    std_logic;
18     busy                       : out   std_logic;
19     inbufsel, outbufsel        : in    std_logic_vector(1 downto 0);
20     treout                     : out   std_logic_vector(7 downto 0);
21     databusinout               : out   std_logic_vector(7 downto 0);
22     databusoutout              : out   std_logic_vector(7 downto 0);
23     inverse                    : in    std_logic;
24     ramreqout, ramdoneout      : out   std_logic;
25     ramwe                      : out   std_logic);
26
27 end fftmgrtestjig;
28
29 architecture structural of fftmgrtestjig is
30
31   component fft
32     port (
33     start, reset, clk          : in    std_logic;
34     testaddrbus                  : out   std_logic_vector(6 downto 0);
35     databusout                  : out std_logic_vector(7 downto 0);
36     databusin                  : in std_logic_vector(7 downto 0);
37     busy                       : out   std_logic;
38     inbufsel, outbufsel        : in    std_logic_vector(1 downto 0);
39     treout                     : out   std_logic_vector(7 downto 0);
40     databusinout               : out   std_logic_vector(7 downto 0);
41     inverse                    : in    std_logic;
42     ramreq                     : out   std_logic;
43     ramdone                    : in    std_logic;
44     ramwe                      : out   std_logic);
45
46   end component;
```

```
47
48    component memmgr
49
50
51      port (
52      aaddr, baddr, caddr : in     std_logic_vector(14-1 downto 0);
53      adata, bdata, cdata : in     std_logic_vector(8-1 downto 0) := (
            others => 'Z');
54      aq, bq, cq          : out     std_logic_vector(8-1 downto 0) := (
            others => 'Z');
55      awr, bwr, cwr       : in     std_logic;
56      clk, reset          : in     std_logic;
57      areq, breq, creq    : in     std_logic;
58      adone, bdone, cdone : out    std_logic;
59      ramaddr             : out    std_logic_vector(14-1 downto 0);
60  --    ramdata             : inout std_logic_vector(8-1 downto 0) := (
        others => 'Z');
61      ramdatain           : in     std_logic_vector(7 downto 0);
62      ramdataout          : out    std_logic_vector(7 downto 0);
63      ramwe               : out    std_logic);
64
65
66
67    end component;
68
69  --  component testram
70  --        PORT
71  --        (
72  --                address          : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
73  --                we               : IN STD_LOGIC  := '1';
74  --                outenab          : IN STD_LOGIC  := '1';
75  --                dio              : INOUT STD_LOGIC_VECTOR (13 DOWNTO 0)
76  --        );
77  --  end component;
78  --
79   component testram
80         PORT
81         (
82                 address          : IN STD_LOGIC_VECTOR (6 DOWNTO 0);
83                 we               : IN STD_LOGIC  := '1';
84                 data             : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
85                 q                : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
86         );
87  end component;
88
89    signal fftaddrbus, addrbus : std_logic_vector(6 downto 0);
90    signal aaddr, nulladdr, mgrramaddrbus : std_logic_vector(13 downto 0);
91    signal ramdatain, ramdataout, nulldata, bq, cq : std_logic_vector(7
          downto 0);
92    signal databus, d, q : std_logic_vector(7 downto 0);
93    signal fftwe, oe, ramweint, ramreq, ramdone, nullflag, bdone, cdone :
```

```
        std_logic;
94
95  begin  -- structural
96
97
98    fftinst : fft port map (
99      start      => start,
100     reset      => reset,
101     clk        => clk,
102     testaddrbus    => fftaddrbus,
103     databusout    => d,
104     databusin => q,
105     busy       => busy,
106     inbufsel   => inbufsel,
107     outbufsel => outbufsel,
108     treout     => treout,
109     inverse    => inverse,
110     databusinout => databusinout,
111     ramreq => ramreq,
112     ramdone => ramdone,
113     ramwe      => fftwe);
114
115   raminst : testram port map (
116     address    => addrbus,
117     data          => ramdataout,
118     q    => ramdatain,
119     we         => ramweint);
120
121   mmgr : memmgr port map (
122     aaddr       => aaddr,
123     adata       => d,
124     aq          => q,
125     awr         => fftwe,
126     clk         => clk,
127     reset       => reset,
128     areq        => ramreq,
129     adone       => ramdone,
130     ramaddr     => mgrramaddrbus,
131     ramdatain   => ramdatain,
132     ramdataout => ramdataout,
133     ramwe       => ramweint,
134     baddr       => nulladdr,
135     caddr       => nulladdr,
136     bdata       => nulldata,
137     cdata       => nulldata,
138     bq          => bq,
139     cq          => cq,
140     bwr         => nullflag,
141     cwr         => nullflag,
142     breq        => nullflag,
143     creq        => nullflag,
```

```
144       bdone        => bdone,
145       cdone        => cdone);
146
147     oe <= '1';
148
149     ramwe <= ramweint;
150
151     databusoutout <= d;
152
153     nulladdr <= (others => '0');
154     nullflag <= '0';
155     nulldata <= (others => '0');
156
157     addrbus <= mgrramaddrbus(6 downto 0);
158     aaddr <= "0000000" & fftaddrbus;
159
160     ramreqout <= ramreq;
161     ramdoneout <= ramdone;
162
163 end structural;
```

A.1.23 testram.vhd

```
1  -- megafunction wizard: %LPM_RAM_DQ%
2  -- GENERATION: STANDARD
3  -- VERSION: WM1.0
4  -- MODULE: lpm_ram_dq
5
6  --============================================================
7  -- File Name: testram.vhd
8  -- Megafunction Name(s):
9  --                         lpm_ram_dq
10 --============================================================
11 -- ************************************************************
12 -- THIS IS A WIZARD GENERATED FILE. DO NOT EDIT THIS FILE!
13 -- ************************************************************
14
15
16 --       Copyright (C) 1988-2002 Altera Corporation
17
18 --       Any megafunction design, and related net list (encrypted or
       decrypted),
19 --       support information, device programming or simulation file, and
       any other
20 --       associated documentation or information provided by Altera or a
       partner
21 --       under Altera's Megafunction Partnership Program may be used only
        to
22 --       program PLD devices (but not masked PLD devices) from Altera.
        Any other
23 --       use of such megafunction design, net list, support information,
       device
24 --       programming or simulation file, or any other related
       documentation or
25 --       information is prohibited for any other purpose, including, but
       not
26 --       limited to modification, reverse engineering, de-compiling, or
       use with
27 --       any other silicon devices, unless such use is explicitly
       licensed under
28 --       a separate agreement with Altera or a megafunction partner.
        Title to
29 --       the intellectual property, including patents, copyrights,
       trademarks,
30 --       trade secrets, or maskworks, embodied in any such megafunction
       design,
31 --       net list, support information, device programming or simulation
       file, or
32 --       any other related documentation or information provided by
       Altera or a
33 --       megafunction partner, remains with Altera, the megafunction
       partner, or
```

```
34 --       their respective licensors. No other licenses, including any
       licenses
35 --       needed under any third party's intellectual property, are
       provided herein.
36
37 LIBRARY ieee;
38 USE ieee.std_logic_1164.all;
39
40 ENTITY testram IS
41         PORT
42         (
43                 address          : IN STD_LOGIC_VECTOR (6 DOWNTO 0);
44                 we               : IN STD_LOGIC  := '1';
45                 data             : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
46                 q                : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
47         );
48 END testram;
49
50
51 ARCHITECTURE SYN OF testram IS
52
53         SIGNAL sub_wire0         : STD_LOGIC_VECTOR (7 DOWNTO 0);
54
55
56
57         COMPONENT lpm_ram_dq
58         GENERIC (
59                 lpm_width                : NATURAL;
60                 lpm_widthad              : NATURAL;
61                 lpm_indata               : STRING;
62                 lpm_address_control              : STRING;
63                 lpm_outdata              : STRING;
64                 lpm_file                 : STRING;
65                 lpm_hint                 : STRING
66         );
67         PORT (
68                         address : IN STD_LOGIC_VECTOR (6 DOWNTO 0);
69                         q       : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
70                         data    : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
71                         we      : IN STD_LOGIC
72         );
73         END COMPONENT;
74
75 BEGIN
76         q    <= sub_wire0(7 DOWNTO 0);
77
78         lpm_ram_dq_component : lpm_ram_dq
79         GENERIC MAP (
80                 LPM_WIDTH => 8,
81                 LPM_WIDTHAD => 7,
82                 LPM_INDATA => "UNREGISTERED",
```

```
83                      LPM_ADDRESS_CONTROL => "UNREGISTERED",
84                      LPM_OUTDATA => "UNREGISTERED",
85                      LPM_FILE => "/afs/athena.mit.edu/user/d/r/drkp/Private
                            /6.111/proj/twiddlegenerator/testram.ntl",
86                      LPM_HINT => "USE_EAB=ON"
87              )
88              PORT MAP (
89                      address => address,
90                      data => data,
91                      we => we,
92                      q => sub_wire0
93              );
94
95
96
97  END SYN;
98
99  -- ============================================================
100 -- CNX file retrieval info
101 -- ============================================================
102 -- Retrieval info: PRIVATE: WidthData NUMERIC "8"
103 -- Retrieval info: PRIVATE: WidthAddr NUMERIC "7"
104 -- Retrieval info: PRIVATE: RegData NUMERIC "0"
105 -- Retrieval info: PRIVATE: RegAdd NUMERIC "0"
106 -- Retrieval info: PRIVATE: OutputRegistered NUMERIC "0"
107 -- Retrieval info: PRIVATE: BlankMemory NUMERIC "0"
108 -- Retrieval info: PRIVATE: MIFfilename STRING "/afs/athena.mit.edu/user
        /d/r/drkp/Private/6.111/proj/twiddlegenerator/testram.ntl"
109 -- Retrieval info: PRIVATE: UseLCs NUMERIC "0"
110 -- Retrieval info: PRIVATE: DataBusSeparated NUMERIC "1"
111 -- Retrieval info: CONSTANT: LPM_WIDTH NUMERIC "8"
112 -- Retrieval info: CONSTANT: LPM_WIDTHAD NUMERIC "7"
113 -- Retrieval info: CONSTANT: LPM_INDATA STRING "UNREGISTERED"
114 -- Retrieval info: CONSTANT: LPM_ADDRESS_CONTROL STRING "UNREGISTERED"
115 -- Retrieval info: CONSTANT: LPM_OUTDATA STRING "UNREGISTERED"
116 -- Retrieval info: CONSTANT: LPM_FILE STRING "/afs/athena.mit.edu/user/d
        /r/drkp/Private/6.111/proj/twiddlegenerator/testram.ntl"
117 -- Retrieval info: CONSTANT: LPM_HINT STRING "USE_EAB=ON"
118 -- Retrieval info: USED_PORT: address 0 0 7 0 INPUT NODEFVAL address
        [6..0]
119 -- Retrieval info: USED_PORT: we 0 0 0 0 INPUT VCC we
120 -- Retrieval info: USED_PORT: q 0 0 8 0 OUTPUT NODEFVAL q[7..0]
121 -- Retrieval info: USED_PORT: data 0 0 8 0 INPUT NODEFVAL data[7..0]
122 -- Retrieval info: CONNECT: @address 0 0 7 0 address 0 0 7 0
123 -- Retrieval info: CONNECT: @we 0 0 0 0 we 0 0 0 0
124 -- Retrieval info: CONNECT: q 0 0 8 0 @q 0 0 8 0
125 -- Retrieval info: CONNECT: @data 0 0 8 0 data 0 0 8 0
```

A.1.24 twiddlify.c

```
/*************************************************************************
 *
 * twiddlify.c : generate twiddle factor table in .dat format
 * Dan R. K. Ports  <drkp@mit.edu>
 * 6.111 final project, 2003/12/04
 *
 *************************************************************************/

#define pi 3.141592653589793

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main(int argc, char **argv)
{
        int i;
        double theta;
        double s,c;
        int N;
        int gensin = 0;

        N = strtol(argv[1], NULL, 0);
        if (argv[2][0] == 's') gensin = 1;

        for (i = 0; i < N; i++)
        {
                theta = ((double)-2) * pi * ((double) i) / ((double)N);
                s = sin(theta) * 128;
                c = cos(theta) * 128;
                if (s >= 127) s = 127;
                if (c >= 127) c = 127;
                if ((s < 0) && (s > -1)) s = 0;
                if ((c < 0) && (c > -1)) c = 0;
                if (s < 0) s = 0xFF + s + 1;
                if (c < 0) c = 0xFF + c + 1;
                if (gensin)
                 printf("%02X\n", (int)s);
                else
                 printf("%02X\n", (int)c);
        }
}
```

A.1.25 twiddle8c.dat

```
1  #SET_ADDRESS=0000;
2  #MASK_COUNT=8;
3  #BASE=HEX;
4
5
6   7F
7   5A
8   00
9   A5
10  80
11  A5
12  00
13  5A
```

A.1.26 twiddle8c.ntl

```
1 :080000007F5A00A580A5005AFB
2 :00000001FF
```

A.1.27 twiddle8s.dat

```
1  #SET_ADDRESS=0000;
2  #MASK_COUNT=8;
3  #BASE=HEX;
4
5
6  00
7  A5
8  80
9  A5
10 00
11 5A
12 7F
13 5A
```

A.1.28 twiddle8s.ntl

```
1 :0800000000A580A5005A7F5AFB
2 :00000001FF
```

A.1.29 twiddle16c.dat

```
1  #SET_ADDRESS=0000;
2  #MASK_COUNT=8;
3  #BASE=HEX;
4
5
6   7F
7   76
8   5A
9   30
10  00
11 CF
12  A5
13  89
14  80
15  89
16  A5
17 CF
18  00
19  30
20  5A
21  76
```

A.1.30 twiddle16c.ntl

```
1 :080000007F765A3000CFA5897C
2 :080008008089A5CF00305A7673
3 :00000001FF
```

A.1.31 twiddle16s.dat

```
#SET_ADDRESS=0000;
#MASK_COUNT=8;
#BASE=HEX;


 00
 CF
 A5
 89
 80
 89
 A5
 CF
 00
 30
 5A
 76
 7F
 76
 5A
 30
```

A.1.32 twiddle16s.ntl

```
1 :0800000000CFA5898089A5CF7E
2 :0800080000305A767F765A3071
3 :00000001FF
```

A.1.33 twiddle512c.dat

```
#SET_ADDRESS=0000;
#MASK_COUNT=8;
#BASE=HEX;


7F
7F
7F
7F
7F
7F
7F
7F
7F
7F
7F
7E
7E
7E
7E
7D
7D
7D
7C
7C
7C
7B
7B
7A
7A
7A
79
79
78
77
77
76
76
75
75
74
73
73
72
71
70
70
6F
6E
```

50	6D
51	6C
52	6C
53	6B
54	6A
55	69
56	68
57	67
58	66
59	65
60	64
61	63
62	62
63	61
64	60
65	5F
66	5E
67	5D
68	5C
69	5B
70	5A
71	59
72	58
73	57
74	55
75	54
76	53
77	52
78	51
79	4F
80	4E
81	4D
82	4C
83	4A
84	49
85	48
86	47
87	45
88	44
89	43
90	41
91	40
92	3F
93	3D
94	3C
95	3A
96	39
97	38
98	36
99	35
100	33

101	32
102	30
103	2F
104	2E
105	2C
106	2B
107	29
108	28
109	26
110	25
111	23
112	22
113	20
114	1F
115	1D
116	1C
117	1A
118	18
119	17
120	15
121	14
122	12
123	11
124	0F
125	0E
126	0C
127	0A
128	09
129	07
130	06
131	04
132	03
133	01
134	00
135	FE
136	FC
137	FB
138	F9
139	F8
140	F6
141	F5
142	F3
143	F1
144	F0
145	EE
146	ED
147	EB
148	EA
149	E8
150	E7
151	E5

152 E3
153 E2
154 E0
155 DF
156 DD
157 DC
158 DA
159 D9
160 D7
161 D6
162 D4
163 D3
164 D1
165 D0
166 CF
167 CD
168 CC
169 CA
170 C9
171 C7
172 C6
173 C5
174 C3
175 C2
176 C0
177 BF
178 BE
179 BC
180 BB
181 BA
182 B8
183 B7
184 B6
185 B5
186 B3
187 B2
188 B1
189 B0
190 AE
191 AD
192 AC
193 AB
194 AA
195 A8
196 A7
197 A6
198 A5
199 A4
200 A3
201 A2
202 A1

203 A0
204 9F
205 9E
206 9D
207 9C
208 9B
209 9A
210 99
211 98
212 97
213 96
214 95
215 94
216 93
217 93
218 92
219 91
220 90
221 8F
222 8F
223 8E
224 8D
225 8C
226 8C
227 8B
228 8A
229 8A
230 89
231 89
232 88
233 88
234 87
235 86
236 86
237 85
238 85
239 85
240 84
241 84
242 83
243 83
244 83
245 82
246 82
247 82
248 81
249 81
250 81
251 81
252 80
253 80

254 80
255 80
256 80
257 80
258 80
259 80
260 80
261 80
262 80
263 80
264 80
265 80
266 80
267 80
268 80
269 80
270 80
271 80
272 80
273 81
274 81
275 81
276 81
277 82
278 82
279 82
280 83
281 83
282 83
283 84
284 84
285 85
286 85
287 85
288 86
289 86
290 87
291 88
292 88
293 89
294 89
295 8A
296 8A
297 8B
298 8C
299 8C
300 8D
301 8E
302 8F
303 8F
304 90

305 91
306 92
307 93
308 93
309 94
310 95
311 96
312 97
313 98
314 99
315 9A
316 9B
317 9C
318 9D
319 9E
320 9F
321 A0
322 A1
323 A2
324 A3
325 A4
326 A5
327 A6
328 A7
329 A8
330 AA
331 AB
332 AC
333 AD
334 AE
335 B0
336 B1
337 B2
338 B3
339 B5
340 B6
341 B7
342 B8
343 BA
344 BB
345 BC
346 BE
347 BF
348 C0
349 C2
350 C3
351 C5
352 C6
353 C7
354 C9
355 CA

356 CC
357 CD
358 CF
359 D0
360 D1
361 D3
362 D4
363 D6
364 D7
365 D9
366 DA
367 DC
368 DD
369 DF
370 E0
371 E2
372 E3
373 E5
374 E7
375 E8
376 EA
377 EB
378 ED
379 EE
380 F0
381 F1
382 F3
383 F5
384 F6
385 F8
386 F9
387 FB
388 FC
389 FE
390 00
391 01
392 03
393 04
394 06
395 07
396 09
397 0A
398 0C
399 0E
400 0F
401 11
402 12
403 14
404 15
405 17
406 18

407 1A
408 1C
409 1D
410 1F
411 20
412 22
413 23
414 25
415 26
416 28
417 29
418 2B
419 2C
420 2E
421 2F
422 30
423 32
424 33
425 35
426 36
427 38
428 39
429 3A
430 3C
431 3D
432 3F
433 40
434 41
435 43
436 44
437 45
438 47
439 48
440 49
441 4A
442 4C
443 4D
444 4E
445 4F
446 51
447 52
448 53
449 54
450 55
451 57
452 58
453 59
454 5A
455 5B
456 5C
457 5D

458 5E
459 5F
460 60
461 61
462 62
463 63
464 64
465 65
466 66
467 67
468 68
469 69
470 6A
471 6B
472 6C
473 6C
474 6D
475 6E
476 6F
477 70
478 70
479 71
480 72
481 73
482 73
483 74
484 75
485 75
486 76
487 76
488 77
489 77
490 78
491 79
492 79
493 7A
494 7A
495 7A
496 7B
497 7B
498 7C
499 7C
500 7C
501 7D
502 7D
503 7D
504 7E
505 7E
506 7E
507 7E
508 7F

509 7F
510 7F
511 7F
512 7F
513 7F
514 7F
515 7F
516 7F
517 7F

A.1.34 twiddle512c.ntl

```
:080000007F7F7F7F7F7F7F7F00
:080008007F7F7F7E7E7E7E7DFE
:080010007D7D7C7C7C7B7B7A0A
:080018007A7A7979787777761E
:0800200076757574737372713B
:0800280070706F6E6D6C6C6B63
:080030006A6968676665646394
:080038006261605F5E5D5C5BCC
:080040005A5958575554535208
:08004800514F4E4D4C4A49484E
:080050004745444341403F3D98
:080058003C3A393836353332E9
:08006000302F2E2C2B2928263D
:0800680025232220 1F1D1C1A94
:0800700018171514121 10F0EF0
:080078000C0A09070604030 14C
:0800800000FEFCFBF9F8F6F5A7
:08008800F3F1F0EEEDEBEAE804
:08009000E7E5E3E2E0DFDDDC5F
:08009800DAD9D7D6D4D3D1D0B8
:0800A000CFCDCCCAC9C7C6C50B
:0800A800C3C2C0BFBEBCBBBA5D
:0800B000B8B7B6B5B3B2B1B0A8
:0800B800AEADACABAAA8A7A6EF
:0800C000A5A4A3A2A1A09F9E2C
:0800C8009D9C9B9A9998979664
:0800D000959493939291908F97
:0800D8008F8E8D8C8C8B8A8ABF
:0800E000898988887868685DE
:0800E8008585848383838 2F3
:0800F000828218181808000
:0800F8008080808080808000
:08010000808080808080080F7
:080108008080808181818182E9
:0801100082828383848485CD
:08011800858586868788888 9A9
:08012000898A8A8B8C8C8D8E7C
:080128008F8F90919293939444
:0801300095969798999A9B9C03
:080138009D9E9FA0A1A2A3A4BB
:08014000A5A6A7A8AAABACAD6F
:08014800AEB0B1B2B3B5B6B719
:08015000B8BABBBCBEBFC0C2BF
:08015800C3C5C6C7C9CACCCD5E
:08016000CFD0D1D3D4D6D7D9FA
:08016800DADCDDDFE0E2E3E593
:08017000E7E8EAEBEDEEF0F127
:08017800F3F5F6F8F9FBFCFEBB
:0801800000010304060709 0A4F
```

```
:080188000C0E0F1112141517E3
:08019000181A1C1D1F20222378
:0801980025262829 2B2C2E2F0F
:0801A000303233353638393AAC
:0801A8003C3D3F40414344454A
:0801B0004748494A4C4D4E4FEF
:0801B8005152535455575859 98
:0801C0005A5B5C5D5E5F60614B
:0801C80062636465666768690 3
:0801D0006A6B6C6C6D6E6F70C0
:0801D80070717273737475758 8
:0801E000767677777879797A59
:0801E8007A7A7B7B7C7C7C7D34
:0801F0007D7D7E7E7E7E7F7F17
:0801F8007F7F7F7F7F7F7F7F07
:00000001FF
```

A.1.35 twiddle512s.dat

```
#SET_ADDRESS=0000;
#MASK_COUNT=8;
#BASE=HEX;


 00
 FE
 FC
 FB
 F9
 F8
 F6
 F5
 F3
 F1
 F0
 EE
 ED
 EB
 EA
 E8
 E7
 E5
 E3
 E2
 E0
 DF
 DD
 DC
 DA
 D9
 D7
 D6
 D4
 D3
 D1
 D0
 CF
 CD
 CC
 CA
 C9
 C7
 C6
 C5
 C3
 C2
 C0
 BF
```

50 BE
51 BC
52 BB
53 BA
54 B8
55 B7
56 B6
57 B5
58 B3
59 B2
60 B1
61 B0
62 AE
63 AD
64 AC
65 AB
66 AA
67 A8
68 A7
69 A6
70 A5
71 A4
72 A3
73 A2
74 A1
75 A0
76 9F
77 9E
78 9D
79 9C
80 9B
81 9A
82 99
83 98
84 97
85 96
86 95
87 94
88 93
89 93
90 92
91 91
92 90
93 8F
94 8F
95 8E
96 8D
97 8C
98 8C
99 8B
100 8A

101 8A
102 89
103 89
104 88
105 88
106 87
107 86
108 86
109 85
110 85
111 85
112 84
113 84
114 83
115 83
116 83
117 82
118 82
119 82
120 81
121 81
122 81
123 81
124 80
125 80
126 80
127 80
128 80
129 80
130 80
131 80
132 80
133 80
134 80
135 80
136 80
137 80
138 80
139 80
140 80
141 80
142 80
143 80
144 80
145 81
146 81
147 81
148 81
149 82
150 82
151 82

152 83
153 83
154 83
155 84
156 84
157 85
158 85
159 85
160 86
161 86
162 87
163 88
164 88
165 89
166 89
167 8A
168 8A
169 8B
170 8C
171 8C
172 8D
173 8E
174 8F
175 8F
176 90
177 91
178 92
179 93
180 93
181 94
182 95
183 96
184 97
185 98
186 99
187 9A
188 9B
189 9C
190 9D
191 9E
192 9F
193 A0
194 A1
195 A2
196 A3
197 A4
198 A5
199 A6
200 A7
201 A8
202 AA

203	AB
204	AC
205	AD
206	AE
207	B0
208	B1
209	B2
210	B3
211	B5
212	B6
213	B7
214	B8
215	BA
216	BB
217	BC
218	BE
219	BF
220	C0
221	C2
222	C3
223	C5
224	C6
225	C7
226	C9
227	CA
228	CC
229	CD
230	CF
231	D0
232	D1
233	D3
234	D4
235	D6
236	D7
237	D9
238	DA
239	DC
240	DD
241	DF
242	E0
243	E2
244	E3
245	E5
246	E7
247	E8
248	EA
249	EB
250	ED
251	EE
252	F0
253	F1

254	F3
255	F5
256	F6
257	F8
258	F9
259	FB
260	FC
261	FE
262	00
263	01
264	03
265	04
266	06
267	07
268	09
269	0A
270	0C
271	0E
272	0F
273	11
274	12
275	14
276	15
277	17
278	18
279	1A
280	1C
281	1D
282	1F
283	20
284	22
285	23
286	25
287	26
288	28
289	29
290	2B
291	2C
292	2E
293	2F
294	30
295	32
296	33
297	35
298	36
299	38
300	39
301	3A
302	3C
303	3D
304	3F

305 40
306 41
307 43
308 44
309 45
310 47
311 48
312 49
313 4A
314 4C
315 4D
316 4E
317 4F
318 51
319 52
320 53
321 54
322 55
323 57
324 58
325 59
326 5A
327 5B
328 5C
329 5D
330 5E
331 5F
332 60
333 61
334 62
335 63
336 64
337 65
338 66
339 67
340 68
341 69
342 6A
343 6B
344 6C
345 6C
346 6D
347 6E
348 6F
349 70
350 70
351 71
352 72
353 73
354 73
355 74

356 75
357 75
358 76
359 76
360 77
361 77
362 78
363 79
364 79
365 7A
366 7A
367 7A
368 7B
369 7B
370 7C
371 7C
372 7C
373 7D
374 7D
375 7D
376 7E
377 7E
378 7E
379 7E
380 7F
381 7F
382 7F
383 7F
384 7F
385 7F
386 7F
387 7F
388 7F
389 7F
390 7F
391 7F
392 7F
393 7F
394 7F
395 7F
396 7F
397 7F
398 7F
399 7F
400 7F
401 7E
402 7E
403 7E
404 7E
405 7D
406 7D

407 7D
408 7C
409 7C
410 7C
411 7B
412 7B
413 7A
414 7A
415 7A
416 79
417 79
418 78
419 77
420 77
421 76
422 76
423 75
424 75
425 74
426 73
427 73
428 72
429 71
430 70
431 70
432 6F
433 6E
434 6D
435 6C
436 6C
437 6B
438 6A
439 69
440 68
441 67
442 66
443 65
444 64
445 63
446 62
447 61
448 60
449 5F
450 5E
451 5D
452 5C
453 5B
454 5A
455 59
456 58
457 57

458 55
459 54
460 53
461 52
462 51
463 4F
464 4E
465 4D
466 4C
467 4A
468 49
469 48
470 47
471 45
472 44
473 43
474 41
475 40
476 3F
477 3D
478 3C
479 3A
480 39
481 38
482 36
483 35
484 33
485 32
486 30
487 2F
488 2E
489 2C
490 2B
491 29
492 28
493 26
494 25
495 23
496 22
497 20
498 1F
499 1D
500 1C
501 1A
502 18
503 17
504 15
505 14
506 12
507 11
508 0F

509 0E
510 0C
511 0A
512 09
513 07
514 06
515 04
516 03
517 01

A.1.36 twiddle512s.ntl

```
:0800000000FEFCFBF9F8F6F527
:08000800F3F1F0EEEDEBEAE884
:08001000E7E5E3E2E0DFDDDCDF
:08001800DAD9D7D6D4D3D1D038
:08002000CFCDCCCAC9C7C6C58B
:08002800C3C2C0BFBEBCBBBADD
:08003000B8B7B6B5B3B2B1B028
:08003800AEADACABAAA8A7A66F
:08004000A5A4A3A2A1A09F9EAC
:080048009D9C9B9A99989796E4
:0800500095949393929190 8F17
:080058008F8E8D8C8C8B8A8A3F
:0800600089898888878686855E
:0800680085858484838383 8273
:0800700082828181818180 8080
:0800780080808080808080 8080
:0800800080808080808080 8078
:0800880080808081818181826A
:0800900082828383838484854E
:0800980085858686878888892A
:0800A000898A8A8B8C8C8D8EFD
:0800A8008F8F909192939394C5
:0800B00095969798999A9B9C84
:0800B8009D9E9FA0A1A2A3A43C
:0800C000A5A6A7A8AAABACADF0
:0800C800AEB0B1B2B3B5B6B79A
:0800D000B8BABBBCBEBFC0C240
:0800D800C3C5C6C7C9CACCCDDF
:0800E000CFD0D1D3D4D6D7D97B
:0800E800DADCDDDFE0E2E3E514
:0800F000E7E8EAEBEDEEF0F1A8
:0800F800F3F5F6F8F9FBFCFE3C
:0801000000010304060709 0ACF
:080108000C0E0F111214151763
:08011000181A1C1D1F202223F8
:0801180025262829 2B2C2E2F8F
:0801200030323335363839 3A2C
:080128003C3D3F4041434445CA
:08013000474849 4A4C4D4E4F6F
:0801380051525354555758 5918
:080140005A5B5C5D5E5F6061CB
:0801480062636465666768 6983
:080150006A6B6C6C6D6E6F7040
:0801580070717273737475 7508
:0801600076767777787979 7AD9
:080168007A7A7B7B7C7C7C7DB4
:080170007D7D7E7E7E7E7F7F97
:080178007F7F7F7F7F7F7F7F87
:080180007F7F7F7F7F7F7F7F7F
```

```
:080188007F7F7F7E7E7E7E7D7D
:080190007D7D7C7C7C7B7B7A89
:080198007A7A7979787777769D
:0801A000767575747373727 1BA
:0801A80070706F6E6D6C6C6BE2
:0801B0006A6968676665646313
:0801B8006261605F5E5D5C5B4B
:0801C0005A5958575554535287
:0801C800514F4E4D4C4A4948CD
:0801D0004745444341403F3D17
:0801D8003C3A39383635333268
:0801E000302F2E2C2B292826BC
:0801E800252322201F1D1C1A13
:0801F0001817151412110F0E6F
:0801F8000C0A090706040301CB
:00000001FF
```

A.1.37 top.acf

```
1  --
2  --  Copyright (C) 1988-2002 Altera Corporation
3  --  Any megafunction design, and related net list (encrypted or
       decrypted),
4  --  support information, device programming or simulation file, and any
       other
5  --  associated documentation or information provided by Altera or a
       partner
6  --  under Altera's Megafunction Partnership Program may be used only to
7  --  program PLD devices (but not masked PLD devices) from Altera.  Any
       other
8  --  use of such megafunction design, net list, support information,
       device
9  --  programming or simulation file, or any other related documentation
       or
10 --  information is prohibited for any other purpose, including, but not
11 --  limited to modification, reverse engineering, de-compiling, or use
       with
12 --  any other silicon devices, unless such use is explicitly licensed
       under
13 --  a separate agreement with Altera or a megafunction partner.  Title
       to
14 --  the intellectual property, including patents, copyrights, trademarks
       ,
15 --  trade secrets, or maskworks, embodied in any such megafunction
       design,
16 --  net list, support information, device programming or simulation file
       , or
17 --  any other related documentation or information provided by Altera or
        a
18 --  megafunction partner, remains with Altera, the megafunction partner
       , or
19 --  their respective licensors.  No other licenses, including any
       licenses
20 --  needed under any third party's intellectual property, are provided
       herein.
21 --
22 CHIP top
23 BEGIN
24         |curportout1 :  PIN = 30;
25         |curportout0 :  OUTPUT_PIN = 29;
26         |ramaddr0 :     OUTPUT_PIN = 63;
27         |ffttest :      INPUT_PIN = 28;
28         |adtest :       INPUT_PIN = 25;
29         |ramdata4 :     BIDIR_PIN = 53;
30         |ramdata3 :     BIDIR_PIN = 54;
31         |ramdata2 :     BIDIR_PIN = 55;
32         |ramdata1 :     BIDIR_PIN = 56;
33         |ramdata0 :     BIDIR_PIN = 61;
```

```
34          |adbus7 :         BIDIR_PIN = 24;
35          |adbus6 :         BIDIR_PIN = 21;
36          |adbus5 :         BIDIR_PIN = 20;
37          |adbus4 :         BIDIR_PIN = 19;
38          |adbus2 :         BIDIR_PIN = 17;
39          |adbus1 :         BIDIR_PIN = 15;
40          |adbus0 :         BIDIR_PIN = 14;
41          |adbus3 :         BIDIR_PIN = 18;
42          |ramdata7 :       BIDIR_PIN = 49;
43          |ramdata6 :       BIDIR_PIN = 50;
44          |ramdata5 :       BIDIR_PIN = 51;
45          |n_ramwe :        OUTPUT_PIN = 64;
46          |ramaddr13 :      OUTPUT_PIN = 65;
47          |ramaddr12 :      OUTPUT_PIN = 66;
48          |ramaddr11 :      OUTPUT_PIN = 67;
49          |ramaddr10 :      OUTPUT_PIN = 68;
50          |ramaddr9 :       OUTPUT_PIN = 70;
51          |ramaddr8 :       OUTPUT_PIN = 71;
52          |ramaddr7 :       OUTPUT_PIN = 72;
53          |ramaddr6 :       OUTPUT_PIN = 73;
54          |ramaddr5 :       OUTPUT_PIN = 74;
55          |ramaddr4 :       OUTPUT_PIN = 75;
56          |ramaddr3 :       OUTPUT_PIN = 76;
57          |ramaddr2 :       OUTPUT_PIN = 78;
58          |ramaddr1 :       OUTPUT_PIN = 79;
59          |ifdatabus0 :     BIDIR_PIN = 38;
60          |ifdatabus1 :     BIDIR_PIN = 39;
61          |ifdatabus2 :     BIDIR_PIN = 40;
62          |ifdatabus3 :     BIDIR_PIN = 41;
63          |ifdatabus4 :     BIDIR_PIN = 43;
64          |ifdatabus5 :     BIDIR_PIN = 44;
65          |ifdatabus6 :     BIDIR_PIN = 45;
66          |ifdatabus7 :     BIDIR_PIN = 46;
67          |ack2 : INPUT_PIN = 36;
68          |procrdyack :     INPUT_PIN = 35;
69          |fftrdyack :      OUTPUT_PIN = 34;
70          |procsending :    INPUT_PIN = 33;
71          |fftsending :     OUTPUT_PIN = 31;
72          |n_ADCEnable :    OUTPUT_PIN = 13;
73          |ADCStatus :      INPUT_PIN = 12;
74          |ADCRead :        OUTPUT_PIN = 9;
75          |n_DACEnable :    OUTPUT_PIN = 8;
76          |reset :          INPUT_PIN = 7;
77          |clk :  INPUT_PIN = 91;
78          DEVICE = EPF10K70RC240-2;
79          FLEX_CONFIGURATION_EPROM = EPC2LC20;
80  END;
81
82  DEFAULT_DEVICES
83  BEGIN
84          AUTO_DEVICE = EPF10K70RC240-2;
```

```
85              AUTO_DEVICE = EPF10K50BC356-3;
86              AUTO_DEVICE = EPF10K50RC240-3;
87              AUTO_DEVICE = EPF10K40RC240-3;
88              AUTO_DEVICE = EPF10K40RC208-3;
89              AUTO_DEVICE = EPF10K30BC356-3;
90              AUTO_DEVICE = EPF10K30RC240-3;
91              AUTO_DEVICE = EPF10K30RC208-3;
92              AUTO_DEVICE = EPF10K20RC240-3;
93              AUTO_DEVICE = EPF10K20RC208-3;
94              AUTO_DEVICE = EPF10K20TC144-3;
95              AUTO_DEVICE = EPF10K10QC208-3;
96              AUTO_DEVICE = EPF10K10TC144-3;
97              AUTO_DEVICE = EPF10K10LC84-3;
98              ASK_BEFORE_ADDING_EXTRA_DEVICES = ON;
99  END;
100
101 TIMING_POINT
102 BEGIN
103             MAINTAIN_STABLE_SYNTHESIS = ON;
104             DEVICE_FOR_TIMING_SYNTHESIS = FLEX10K;
105             CUT_ALL_CLEAR_PRESET = ON;
106             CUT_ALL_BIDIR = ON;
107 END;
108
109 IGNORED_ASSIGNMENTS
110 BEGIN
111             FIT_IGNORE_TIMING = ON;
112             DEMOTE_SPECIFIC_LCELL_ASSIGNMENTS_TO_LAB_ASSIGNMENTS = OFF;
113             IGNORE_LOCAL_ROUTING_ASSIGNMENTS = OFF;
114             IGNORE_DEVICE_ASSIGNMENTS = OFF;
115             IGNORE_LC_ASSIGNMENTS = OFF;
116             IGNORE_PIN_ASSIGNMENTS = OFF;
117             IGNORE_CHIP_ASSIGNMENTS = OFF;
118             IGNORE_TIMING_ASSIGNMENTS = OFF;
119             IGNORE_LOGIC_OPTION_ASSIGNMENTS = OFF;
120             IGNORE_CLIQUE_ASSIGNMENTS = OFF;
121 END;
122
123 GLOBAL_PROJECT_DEVICE_OPTIONS
124 BEGIN
125             MAX7000B_ENABLE_VREFB = OFF;
126             MAX7000B_ENABLE_VREFA = OFF;
127             MAX7000B_VCCIO_IOBANK2 = 3.3V;
128             MAX7000B_VCCIO_IOBANK1 = 3.3V;
129             CONFIG_EPROM_PULLUP_RESISTOR = ON;
130             CONFIG_EPROM_USER_CODE = FFFFFFFF;
131             FLEX_CONFIGURATION_EPROM = AUTO;
132             MAX7000AE_ENABLE_JTAG = ON;
133             MAX7000AE_USER_CODE = FFFFFFFF;
134             FLEX6000_USE_LOW_VOLTAGE_CONFIGURATION_EPROM = OFF;
135             FLEX10KA_USE_LOW_VOLTAGE_CONFIGURATION_EPROM = ON;
```

```
136             FLEX10K_USE_LOW_VOLTAGE_CONFIGURATION_EPROM = OFF;
137             FLEX6000_ENABLE_JTAG = OFF;
138             CONFIG_SCHEME_FLEX_6000 = PASSIVE_SERIAL;
139             MULTIVOLT_IO = OFF;
140             MAX7000S_ENABLE_JTAG = ON;
141             FLEX10K_ENABLE_LOCK_OUTPUT = OFF;
142             MAX7000S_USER_CODE = FFFF;
143             CONFIG_SCHEME_10K = PASSIVE_SERIAL;
144             FLEX10K_JTAG_USER_CODE = 7F;
145             ENABLE_INIT_DONE_OUTPUT = OFF;
146             ENABLE_CHIP_WIDE_OE = OFF;
147             ENABLE_CHIP_WIDE_RESET = OFF;
148             nCEO = UNRESERVED;
149             CLKUSR = UNRESERVED;
150             ADD17 = UNRESERVED;
151             ADD16 = UNRESERVED;
152             ADD15 = UNRESERVED;
153             ADD14 = UNRESERVED;
154             ADD13 = UNRESERVED;
155             ADD0_TO_ADD12 = UNRESERVED;
156             SDOUT = RESERVED_DRIVES_OUT;
157             RDCLK = UNRESERVED;
158             RDYnBUSY = UNRESERVED;
159             nWS_nRS_nCS_CS = UNRESERVED;
160             DATA1_TO_DATA7 = UNRESERVED;
161             DATA0 = RESERVED_TRI_STATED;
162             FLEX8000_ENABLE_JTAG = OFF;
163             CONFIG_SCHEME = ACTIVE_SERIAL;
164             DISABLE_TIME_OUT = OFF;
165             ENABLE_DCLK_OUTPUT = OFF;
166             RELEASE_CLEARS = OFF;
167             AUTO_RESTART = OFF;
168             USER_CLOCK = OFF;
169             SECURITY_BIT = OFF;
170             RESERVED_PINS_PERCENT = 0;
171             RESERVED_LCELLS_PERCENT = 0;
172 END;
173
174 GLOBAL_PROJECT_SYNTHESIS_ASSIGNMENT_OPTIONS
175 BEGIN
176             DEVICE_FAMILY = FLEX10K;
177             MULTI_LEVEL_SYNTHESIS_MAX9000 = ON;
178             AUTO_IMPLEMENT_IN_EAB = OFF;
179             AUTO_OPEN_DRAIN_PINS = ON;
180             ONE_HOT_STATE_MACHINE_ENCODING = OFF;
181             AUTO_REGISTER_PACKING = OFF;
182             STYLE = NORMAL;
183             AUTO_FAST_IO = OFF;
184             AUTO_GLOBAL_OE = ON;
185             AUTO_GLOBAL_PRESET = ON;
186             AUTO_GLOBAL_CLEAR = ON;
```

```
187             AUTO_GLOBAL_CLOCK = ON;
188             MULTI_LEVEL_SYNTHESIS_MAX5000_7000 = OFF;
189             OPTIMIZE_FOR_SPEED = 5;
190 END;
191
192 COMPILER_PROCESSING_CONFIGURATION
193 BEGIN
194             PRESERVE_ALL_NODE_NAME_SYNONYMS = OFF;
195             FITTER_SETTINGS = NORMAL;
196             SMART_RECOMPILE = OFF;
197             GENERATE_AHDL_TDO_FILE = OFF;
198             RPT_FILE_USER_ASSIGNMENTS = ON;
199             RPT_FILE_LCELL_INTERCONNECT = ON;
200             RPT_FILE_HIERARCHY = ON;
201             RPT_FILE_EQUATIONS = ON;
202             LINKED_SNF_EXTRACTOR = OFF;
203             OPTIMIZE_TIMING_SNF = OFF;
204             TIMING_SNF_EXTRACTOR = ON;
205             FUNCTIONAL_SNF_EXTRACTOR = OFF;
206             DESIGN_DOCTOR_RULES = EPLD;
207             DESIGN_DOCTOR = OFF;
208 END;
209
210 COMPILER_INTERFACES_CONFIGURATION
211 BEGIN
212             NETLIST_OUTPUT_TIME_SCALE = 0.1 ns;
213             EDIF_INPUT_SHOW_LMF_MAPPING_MESSAGES = OFF;
214             EDIF_BUS_DELIMITERS = [];
215             EDIF_FLATTEN_BUS = OFF;
216             EDIF_OUTPUT_FORCE_0NS_DELAYS = OFF;
217             EDIF_OUTPUT_INCLUDE_SPECIAL_PRIM = OFF;
218             EDIF_OUTPUT_MAP_ILLEGAL_CHAR = OFF;
219             EDIF_OUTPUT_DELAY_CONSTRUCTS = EDO_FILE;
220             EDIF_OUTPUT_USE_EDC = OFF;
221             EDIF_INPUT_USE_LMF2 = OFF;
222             EDIF_INPUT_USE_LMF1 = OFF;
223             EDIF_OUTPUT_GND = GND;
224             EDIF_OUTPUT_VCC = VCC;
225             EDIF_INPUT_GND = GND;
226             EDIF_INPUT_VCC = VCC;
227             EDIF_OUTPUT_EDC_FILE = *.edc;
228             EDIF_INPUT_LMF2 = *.lmf;
229             EDIF_INPUT_LMF1 = *.lmf;
230             VHDL_GENERATE_CONFIGURATION_DECLARATION = OFF;
231             VHDL_OUTPUT_DELAY_CONSTRUCTS = VHO_FILE;
232             VERILOG_OUTPUT_DELAY_CONSTRUCTS = VO_FILE;
233             VHDL_FLATTEN_BUS = OFF;
234             VERILOG_FLATTEN_BUS = OFF;
235             EDIF_TRUNCATE_HIERARCHY_PATH = OFF;
236             VHDL_TRUNCATE_HIERARCHY_PATH = OFF;
237             VERILOG_TRUNCATE_HIERARCHY_PATH = OFF;
```

```
238             VERILOG_OUTPUT_MAP_ILLEGAL_CHAR = OFF;
239             VHDL_WRITER_VERSION = VHDL93;
240             VHDL_READER_VERSION = VHDL93;
241             USE_ADT_PALACE_FOR_MAX = OFF;
242             SYNOPSYS_MAPPING_EFFORT = MEDIUM;
243             SYNOPSYS_BOUNDARY_OPTIMIZATION = OFF;
244             SYNOPSYS_HIERARCHICAL_COMPILATION = ON;
245             SYNOPSYS_DESIGNWARE = OFF;
246             SYNOPSYS_COMPILER = DESIGN;
247             USE_SYNOPSYS_SYNTHESIS = OFF;
248             VHDL_NETLIST_WRITER = OFF;
249             VERILOG_NETLIST_WRITER = OFF;
250             XNF_GENERATE_AHDL_TDX_FILE = ON;
251             XNF_TRANSLATE_INTERNAL_NODE_NAMES = ON;
252             XNF_EMULATE_TRI_STATE_BUSES = INTERNAL_LOGIC;
253             EDIF_OUTPUT_VERSION = 200;
254             EDIF_NETLIST_WRITER = OFF;
255 END;
256
257 CUSTOM_DESIGN_DOCTOR_RULES
258 BEGIN
259             MASTER_RESET = OFF;
260             EXPANDER_NETWORKS = ON;
261             RACE_CONDITIONS = ON;
262             DELAY_CHAINS = ON;
263             ASYNCHRONOUS_INPUTS = ON;
264             PRESET_CLEAR_NETWORKS = ON;
265             STATIC_HAZARDS_AFTER_SYNTHESIS = OFF;
266             STATIC_HAZARDS_BEFORE_SYNTHESIS = ON;
267             MULTI_CLOCK_NETWORKS = ON;
268             MULTI_LEVEL_CLOCKS = ON;
269             GATED_CLOCKS = ON;
270             RIPPLE_CLOCKS = ON;
271 END;
272
273 SIMULATOR_CONFIGURATION
274 BEGIN
275             END_TIME = 500.0us;
276             BIDIR_PIN = STRONG;
277             START_TIME = 0.0ns;
278             GLITCH_TIME = 0.0ns;
279             GLITCH = OFF;
280             OSCILLATION_TIME = 0.0ns;
281             OSCILLATION = OFF;
282             CHECK_OUTPUTS = OFF;
283             SETUP_HOLD = OFF;
284             USE_DEVICE = OFF;
285 END;
286
287 TIMING_ANALYZER_CONFIGURATION
288 BEGIN
```

```
289             CUT_OFF_RAM_REGISTERED_WE_PATHS = OFF;
290             LIST_PATH_FREQUENCY = 10MHz;
291             LIST_PATH_COUNT = 10;
292             REGISTERED_PERFORMANCE_OPTIONS = NUMBER_OF_PATHS;
293             INCLUDE_PATHS_LESS_THAN_VALUE = 214.7483647ms;
294             INCLUDE_PATHS_LESS_THAN = OFF;
295             INCLUDE_PATHS_GREATER_THAN_VALUE = 0.0ns;
296             INCLUDE_PATHS_GREATER_THAN = OFF;
297             DELAY_MATRIX_OPTIONS = SHOW_ALL_PATHS;
298             CELL_WIDTH = 18;
299             LIST_ONLY_LONGEST_PATH = ON;
300             CUT_OFF_CLEAR_AND_PRESET_PATHS = ON;
301             CUT_OFF_IO_PIN_FEEDBACK = ON;
302             AUTO_RECALCULATE = OFF;
303             ANALYSIS_MODE = DELAY_MATRIX;
304 END;
305
306 OTHER_CONFIGURATION
307 BEGIN
308             EXPLICIT_FAMILY = 1;
309             LAST_MAXPLUS2_VERSION = 10.2;
310             FLEX_10K_52_COLUMNS = 40;
311             DEFAULT_9K_EXP_PER_LCELL = 1/2;
312             LOCAL_INTERCONNECT_PER_LAB_PERCENT = 100;
313             LCELLS_PER_ROW_PERCENT = 100;
314             FAN_IN_PER_LCELL_PERCENT = 100;
315             EXP_PER_LCELL_PERCENT = 100;
316             ROW_PINS_PERCENT = 50;
317             ORIGINAL_MAXPLUS2_VERSION = 10.2;
318             COMPILER_DATA = "1,1,0,1,0,0,0,1,1,1,1,0,1,1,1";
319 END;
320
321 DEFINE_LOGIC_SYNTHESIS_STYLE NORMAL.MAX5000
322 BEGIN
323             REGISTER_OPTIMIZATION = ON;
324             USE_LPM_FOR_AHDL_OPERATORS = OFF;
325             RESYNTHESIZE_NETWORK = ON;
326             MULTI_LEVEL_FACTORING = ON;
327             SUBFACTOR_EXTRACTION = ON;
328             REFACTORIZATION = ON;
329             NOT_GATE_PUSH_BACK = ON;
330             DUPLICATE_LOGIC_EXTRACTION = ON;
331             REDUCE_LOGIC = ON;
332             DECOMPOSE_GATES = ON;
333             SOFT_BUFFER_INSERTION = ON;
334             FAST_IO = OFF;
335             IGNORE_SOFT_BUFFERS = OFF;
336             PARALLEL_EXPANDERS = OFF;
337             TURBO_BIT = OFF;
338             XOR_SYNTHESIS = ON;
339             SLOW_SLEW_RATE = OFF;
```

```
340             MINIMIZATION = FULL;
341             CARRY_CHAIN_LENGTH = -1;
342             CARRY_CHAIN = IGNORE;
343             CASCADE_CHAIN_LENGTH = -1;
344             CASCADE_CHAIN = IGNORE;
345 END;
346
347 DEFINE_LOGIC_SYNTHESIS_STYLE NORMAL.MAX7000
348 BEGIN
349             REGISTER_OPTIMIZATION = ON;
350             USE_LPM_FOR_AHDL_OPERATORS = OFF;
351             RESYNTHESIZE_NETWORK = ON;
352             MULTI_LEVEL_FACTORING = ON;
353             SUBFACTOR_EXTRACTION = ON;
354             REFACTORIZATION = ON;
355             NOT_GATE_PUSH_BACK = ON;
356             DUPLICATE_LOGIC_EXTRACTION = ON;
357             REDUCE_LOGIC = ON;
358             DECOMPOSE_GATES = ON;
359             SOFT_BUFFER_INSERTION = ON;
360             FAST_IO = OFF;
361             IGNORE_SOFT_BUFFERS = OFF;
362             PARALLEL_EXPANDERS = OFF;
363             TURBO_BIT = ON;
364             XOR_SYNTHESIS = ON;
365             SLOW_SLEW_RATE = OFF;
366             MINIMIZATION = FULL;
367             CARRY_CHAIN_LENGTH = -1;
368             CARRY_CHAIN = IGNORE;
369             CASCADE_CHAIN_LENGTH = -1;
370             CASCADE_CHAIN = IGNORE;
371 END;
372
373 DEFINE_LOGIC_SYNTHESIS_STYLE NORMAL.CLASSIC
374 BEGIN
375             REGISTER_OPTIMIZATION = OFF;
376             USE_LPM_FOR_AHDL_OPERATORS = OFF;
377             RESYNTHESIZE_NETWORK = ON;
378             MULTI_LEVEL_FACTORING = OFF;
379             SUBFACTOR_EXTRACTION = OFF;
380             REFACTORIZATION = OFF;
381             NOT_GATE_PUSH_BACK = ON;
382             DUPLICATE_LOGIC_EXTRACTION = OFF;
383             REDUCE_LOGIC = OFF;
384             DECOMPOSE_GATES = ON;
385             SOFT_BUFFER_INSERTION = ON;
386             FAST_IO = OFF;
387             IGNORE_SOFT_BUFFERS = OFF;
388             PARALLEL_EXPANDERS = OFF;
389             TURBO_BIT = ON;
390             XOR_SYNTHESIS = OFF;
```

```
391             SLOW_SLEW_RATE = OFF;
392             MINIMIZATION = FULL;
393             CARRY_CHAIN_LENGTH = -1;
394             CARRY_CHAIN = IGNORE;
395             CASCADE_CHAIN_LENGTH = -1;
396             CASCADE_CHAIN = IGNORE;
397 END;
398
399 DEFINE_LOGIC_SYNTHESIS_STYLE NORMAL.FLEX8000
400 BEGIN
401             REGISTER_OPTIMIZATION = ON;
402             USE_LPM_FOR_AHDL_OPERATORS = OFF;
403             RESYNTHESIZE_NETWORK = ON;
404             MULTI_LEVEL_FACTORING = ON;
405             SUBFACTOR_EXTRACTION = ON;
406             REFACTORIZATION = ON;
407             NOT_GATE_PUSH_BACK = ON;
408             DUPLICATE_LOGIC_EXTRACTION = ON;
409             REDUCE_LOGIC = ON;
410             DECOMPOSE_GATES = ON;
411             SOFT_BUFFER_INSERTION = ON;
412             IGNORE_SOFT_BUFFERS = ON;
413             PARALLEL_EXPANDERS = OFF;
414             TURBO_BIT = OFF;
415             XOR_SYNTHESIS = OFF;
416             SLOW_SLEW_RATE = OFF;
417             MINIMIZATION = FULL;
418             CARRY_CHAIN_LENGTH = 32;
419             CARRY_CHAIN = IGNORE;
420             CASCADE_CHAIN_LENGTH = 2;
421             CASCADE_CHAIN = IGNORE;
422 END;
423
424 DEFINE_LOGIC_SYNTHESIS_STYLE FAST.MAX5000
425 BEGIN
426             REGISTER_OPTIMIZATION = ON;
427             USE_LPM_FOR_AHDL_OPERATORS = OFF;
428             RESYNTHESIZE_NETWORK = ON;
429             MULTI_LEVEL_FACTORING = ON;
430             SUBFACTOR_EXTRACTION = OFF;
431             REFACTORIZATION = OFF;
432             NOT_GATE_PUSH_BACK = ON;
433             DUPLICATE_LOGIC_EXTRACTION = ON;
434             REDUCE_LOGIC = ON;
435             DECOMPOSE_GATES = ON;
436             SOFT_BUFFER_INSERTION = ON;
437             FAST_IO = OFF;
438             IGNORE_SOFT_BUFFERS = OFF;
439             PARALLEL_EXPANDERS = OFF;
440             TURBO_BIT = OFF;
441             XOR_SYNTHESIS = ON;
```

```
442             SLOW_SLEW_RATE = OFF;
443             MINIMIZATION = FULL;
444             CARRY_CHAIN_LENGTH = -1;
445             CARRY_CHAIN = IGNORE;
446             CASCADE_CHAIN_LENGTH = -1;
447             CASCADE_CHAIN = IGNORE;
448 END;
449
450 DEFINE_LOGIC_SYNTHESIS_STYLE FAST.MAX7000
451 BEGIN
452             REGISTER_OPTIMIZATION = ON;
453             USE_LPM_FOR_AHDL_OPERATORS = OFF;
454             RESYNTHESIZE_NETWORK = ON;
455             MULTI_LEVEL_FACTORING = ON;
456             SUBFACTOR_EXTRACTION = OFF;
457             REFACTORIZATION = OFF;
458             NOT_GATE_PUSH_BACK = ON;
459             DUPLICATE_LOGIC_EXTRACTION = ON;
460             REDUCE_LOGIC = ON;
461             DECOMPOSE_GATES = ON;
462             SOFT_BUFFER_INSERTION = ON;
463             FAST_IO = OFF;
464             IGNORE_SOFT_BUFFERS = OFF;
465             PARALLEL_EXPANDERS = ON;
466             TURBO_BIT = ON;
467             XOR_SYNTHESIS = ON;
468             SLOW_SLEW_RATE = OFF;
469             MINIMIZATION = FULL;
470             CARRY_CHAIN_LENGTH = -1;
471             CARRY_CHAIN = IGNORE;
472             CASCADE_CHAIN_LENGTH = -1;
473             CASCADE_CHAIN = IGNORE;
474 END;
475
476 DEFINE_LOGIC_SYNTHESIS_STYLE FAST.CLASSIC
477 BEGIN
478             REGISTER_OPTIMIZATION = OFF;
479             USE_LPM_FOR_AHDL_OPERATORS = OFF;
480             RESYNTHESIZE_NETWORK = ON;
481             MULTI_LEVEL_FACTORING = OFF;
482             SUBFACTOR_EXTRACTION = OFF;
483             REFACTORIZATION = OFF;
484             NOT_GATE_PUSH_BACK = ON;
485             DUPLICATE_LOGIC_EXTRACTION = OFF;
486             REDUCE_LOGIC = OFF;
487             DECOMPOSE_GATES = ON;
488             SOFT_BUFFER_INSERTION = ON;
489             FAST_IO = OFF;
490             IGNORE_SOFT_BUFFERS = OFF;
491             PARALLEL_EXPANDERS = OFF;
492             TURBO_BIT = ON;
```

```
493             XOR_SYNTHESIS = OFF;
494             SLOW_SLEW_RATE = OFF;
495             MINIMIZATION = FULL;
496             CARRY_CHAIN_LENGTH = -1;
497             CARRY_CHAIN = IGNORE;
498             CASCADE_CHAIN_LENGTH = -1;
499             CASCADE_CHAIN = IGNORE;
500 END;
501
502 DEFINE_LOGIC_SYNTHESIS_STYLE FAST.FLEX8000
503 BEGIN
504             REGISTER_OPTIMIZATION = ON;
505             USE_LPM_FOR_AHDL_OPERATORS = OFF;
506             RESYNTHESIZE_NETWORK = ON;
507             MULTI_LEVEL_FACTORING = ON;
508             SUBFACTOR_EXTRACTION = OFF;
509             REFACTORIZATION = OFF;
510             NOT_GATE_PUSH_BACK = ON;
511             DUPLICATE_LOGIC_EXTRACTION = ON;
512             REDUCE_LOGIC = ON;
513             DECOMPOSE_GATES = ON;
514             SOFT_BUFFER_INSERTION = ON;
515             IGNORE_SOFT_BUFFERS = ON;
516             PARALLEL_EXPANDERS = OFF;
517             TURBO_BIT = OFF;
518             XOR_SYNTHESIS = OFF;
519             SLOW_SLEW_RATE = OFF;
520             MINIMIZATION = FULL;
521             CARRY_CHAIN_LENGTH = 32;
522             CARRY_CHAIN = AUTO;
523             CASCADE_CHAIN_LENGTH = 2;
524             CASCADE_CHAIN = AUTO;
525 END;
526
527 DEFINE_LOGIC_SYNTHESIS_STYLE WYSIWYG.MAX5000
528 BEGIN
529             REGISTER_OPTIMIZATION = OFF;
530             USE_LPM_FOR_AHDL_OPERATORS = OFF;
531             RESYNTHESIZE_NETWORK = OFF;
532             MULTI_LEVEL_FACTORING = OFF;
533             SUBFACTOR_EXTRACTION = OFF;
534             REFACTORIZATION = OFF;
535             NOT_GATE_PUSH_BACK = ON;
536             DUPLICATE_LOGIC_EXTRACTION = OFF;
537             REDUCE_LOGIC = OFF;
538             DECOMPOSE_GATES = OFF;
539             SOFT_BUFFER_INSERTION = OFF;
540             FAST_IO = OFF;
541             IGNORE_SOFT_BUFFERS = OFF;
542             PARALLEL_EXPANDERS = OFF;
543             TURBO_BIT = OFF;
```

```
544             XOR_SYNTHESIS = OFF;
545             SLOW_SLEW_RATE = OFF;
546             MINIMIZATION = PARTIAL;
547             CARRY_CHAIN_LENGTH = -1;
548             CARRY_CHAIN = IGNORE;
549             CASCADE_CHAIN_LENGTH = -1;
550             CASCADE_CHAIN = IGNORE;
551 END;
552
553 DEFINE_LOGIC_SYNTHESIS_STYLE WYSIWYG.MAX7000
554 BEGIN
555             REGISTER_OPTIMIZATION = OFF;
556             USE_LPM_FOR_AHDL_OPERATORS = OFF;
557             RESYNTHESIZE_NETWORK = OFF;
558             MULTI_LEVEL_FACTORING = OFF;
559             SUBFACTOR_EXTRACTION = OFF;
560             REFACTORIZATION = OFF;
561             NOT_GATE_PUSH_BACK = ON;
562             DUPLICATE_LOGIC_EXTRACTION = OFF;
563             REDUCE_LOGIC = OFF;
564             DECOMPOSE_GATES = OFF;
565             SOFT_BUFFER_INSERTION = OFF;
566             FAST_IO = OFF;
567             IGNORE_SOFT_BUFFERS = OFF;
568             PARALLEL_EXPANDERS = OFF;
569             TURBO_BIT = ON;
570             XOR_SYNTHESIS = OFF;
571             SLOW_SLEW_RATE = OFF;
572             MINIMIZATION = PARTIAL;
573             CARRY_CHAIN_LENGTH = -1;
574             CARRY_CHAIN = IGNORE;
575             CASCADE_CHAIN_LENGTH = -1;
576             CASCADE_CHAIN = IGNORE;
577 END;
578
579 DEFINE_LOGIC_SYNTHESIS_STYLE WYSIWYG.CLASSIC
580 BEGIN
581             REGISTER_OPTIMIZATION = OFF;
582             USE_LPM_FOR_AHDL_OPERATORS = OFF;
583             RESYNTHESIZE_NETWORK = ON;
584             MULTI_LEVEL_FACTORING = OFF;
585             SUBFACTOR_EXTRACTION = OFF;
586             REFACTORIZATION = OFF;
587             NOT_GATE_PUSH_BACK = ON;
588             DUPLICATE_LOGIC_EXTRACTION = OFF;
589             REDUCE_LOGIC = OFF;
590             DECOMPOSE_GATES = ON;
591             SOFT_BUFFER_INSERTION = OFF;
592             FAST_IO = OFF;
593             IGNORE_SOFT_BUFFERS = OFF;
594             PARALLEL_EXPANDERS = OFF;
```

```
595             TURBO_BIT = ON;
596             XOR_SYNTHESIS = OFF;
597             SLOW_SLEW_RATE = OFF;
598             MINIMIZATION = PARTIAL;
599             CARRY_CHAIN_LENGTH = -1;
600             CARRY_CHAIN = IGNORE;
601             CASCADE_CHAIN_LENGTH = -1;
602             CASCADE_CHAIN = IGNORE;
603 END;
604
605 DEFINE_LOGIC_SYNTHESIS_STYLE WYSIWYG.FLEX8000
606 BEGIN
607             REGISTER_OPTIMIZATION = OFF;
608             USE_LPM_FOR_AHDL_OPERATORS = OFF;
609             RESYNTHESIZE_NETWORK = OFF;
610             MULTI_LEVEL_FACTORING = OFF;
611             SUBFACTOR_EXTRACTION = OFF;
612             REFACTORIZATION = OFF;
613             NOT_GATE_PUSH_BACK = ON;
614             DUPLICATE_LOGIC_EXTRACTION = OFF;
615             REDUCE_LOGIC = OFF;
616             DECOMPOSE_GATES = OFF;
617             SOFT_BUFFER_INSERTION = ON;
618             IGNORE_SOFT_BUFFERS = ON;
619             PARALLEL_EXPANDERS = OFF;
620             TURBO_BIT = OFF;
621             XOR_SYNTHESIS = OFF;
622             SLOW_SLEW_RATE = OFF;
623             MINIMIZATION = PARTIAL;
624             CARRY_CHAIN_LENGTH = 32;
625             CARRY_CHAIN = MANUAL;
626             CASCADE_CHAIN_LENGTH = 2;
627             CASCADE_CHAIN = MANUAL;
628 END;
```

A.1.38 top.pin

```
1  -- Copyright (C) 1988-2002 Altera Corporation
2  -- Any megafunction design, and related net list (encrypted or decrypted
       ),
3  -- support information, device programming or simulation file, and any
       other
4  -- associated documentation or information provided by Altera or a
       partner
5  -- under Altera's Megafunction Partnership Program may be used only to
6  -- program PLD devices (but not masked PLD devices) from Altera.  Any
       other
7  -- use of such megafunction design, net list, support information,
       device
8  -- programming or simulation file, or any other related documentation or
9  -- information is prohibited for any other purpose, including, but not
10 -- limited to modification, reverse engineering, de-compiling, or use
       with
11 -- any other silicon devices, unless such use is explicitly licensed
       under
12 -- a separate agreement with Altera or a megafunction partner.  Title to
13 -- the intellectual property, including patents, copyrights, trademarks,
14 -- trade secrets, or maskworks, embodied in any such megafunction design
       ,
15 -- net list, support information, device programming or simulation file
       , or
16 -- any other related documentation or information provided by Altera or
       a
17 -- megafunction partner, remains with Altera, the megafunction partner,
       or
18 -- their respective licensors.  No other licenses, including any
       licenses
19 -- needed under any third party's intellectual property, are provided
       herein.
20
21
22
23 N.C. = No Connect. This pin has no internal connection to the device.
24 VCCINT = Dedicated power pin, which MUST be connected to VCC (5.0 volts)
       .
25 VCCIO = Dedicated power pin, which MUST be connected to VCC (5.0 volts).
26 GNDINT = Dedicated ground pin or unused dedicated input, which MUST be
       connected to GND.
27 GNDIO = Dedicated ground pin, which MUST be connected to GND.
28 RESERVED = Unused I/O pin, which MUST be left unconnected.
29
30 ------------------------------------------------------------------------
31
32 CHIP "top" ASSIGNED TO AN EPF10K70RC240-2
33 TCK                                    : 1
```

```
34 CONF_DONE                    : 2
35 nCEO                         : 3
36 TDO                          : 4
37 VCCINT                       : 5
38 RESERVED                     : 6
39 reset                        : 7
40 n_DACEnable                  : 8
41 ADCRead                      : 9
42 GNDINT                       : 10
43 RESERVED                     : 11
44 ADCStatus                    : 12
45 n_ADCEnable                  : 13
46 adbus0                       : 14
47 adbus1                       : 15
48 VCCINT                       : 16
49 adbus2                       : 17
50 adbus3                       : 18
51 adbus4                       : 19
52 adbus5                       : 20
53 adbus6                       : 21
54 GNDINT                       : 22
55 RESERVED                     : 23
56 adbus7                       : 24
57 adtest                       : 25
58 RESERVED                     : 26
59 VCCINT                       : 27
60 ffttest                      : 28
61 curportout0                  : 29
62 curportout1                  : 30
63 fftsending                   : 31
64 GNDINT                       : 32
65 procsending                  : 33
66 fftrdyack                    : 34
67 procrdyack                   : 35
68 ack2                         : 36
69 VCCINT                       : 37
70 ifdatabus0                   : 38
71 ifdatabus1                   : 39
72 ifdatabus2                   : 40
73 ifdatabus3                   : 41
74 GNDINT                       : 42
75 ifdatabus4                   : 43
76 ifdatabus5                   : 44
77 ifdatabus6                   : 45
78 ifdatabus7                   : 46
79 VCCINT                       : 47
80 RESERVED                     : 48
81 ramdata7                     : 49
82 ramdata6                     : 50
83 ramdata5                     : 51
84 GNDINT                       : 52
```

85	ramdata4	: 53
86	ramdata3	: 54
87	ramdata2	: 55
88	ramdata1	: 56
89	VCCINT	: 57
90	TMS	: 58
91	TRST	: 59
92	nSTATUS	: 60
93	ramdata0	: 61
94	RESERVED	: 62
95	ramaddr0	: 63
96	n_ramwe	: 64
97	ramaddr13	: 65
98	ramaddr12	: 66
99	ramaddr11	: 67
100	ramaddr10	: 68
101	GNDINT	: 69
102	ramaddr9	: 70
103	ramaddr8	: 71
104	ramaddr7	: 72
105	ramaddr6	: 73
106	ramaddr5	: 74
107	ramaddr4	: 75
108	ramaddr3	: 76
109	VCCINT	: 77
110	ramaddr2	: 78
111	ramaddr1	: 79
112	RESERVED	: 80
113	RESERVED	: 81
114	RESERVED	: 82
115	RESERVED	: 83
116	RESERVED	: 84
117	GNDINT	: 85
118	RESERVED	: 86
119	RESERVED	: 87
120	RESERVED	: 88
121	VCCINT	: 89
122	GNDINT	: 90
123	clk	: 91
124	GNDINT	: 92
125	GNDINT	: 93
126	RESERVED	: 94
127	RESERVED	: 95
128	VCCINT	: 96
129	RESERVED	: 97
130	RESERVED	: 98
131	RESERVED	: 99
132	RESERVED	: 100
133	RESERVED	: 101
134	RESERVED	: 102
135	RESERVED	: 103

136	GNDINT	: 104
137	RESERVED	: 105
138	RESERVED	: 106
139	RESERVED	: 107
140	RESERVED	: 108
141	RESERVED	: 109
142	RESERVED	: 110
143	RESERVED	: 111
144	VCCINT	: 112
145	RESERVED	: 113
146	RESERVED	: 114
147	RESERVED	: 115
148	RESERVED	: 116
149	RESERVED	: 117
150	RESERVED	: 118
151	RESERVED	: 119
152	RESERVED	: 120
153	nCONFIG	: 121
154	VCCINT	: 122
155	MSEL1	: 123
156	MSEL0	: 124
157	GNDINT	: 125
158	RESERVED	: 126
159	RESERVED	: 127
160	RESERVED	: 128
161	RESERVED	: 129
162	VCCINT	: 130
163	RESERVED	: 131
164	RESERVED	: 132
165	RESERVED	: 133
166	RESERVED	: 134
167	GNDINT	: 135
168	RESERVED	: 136
169	RESERVED	: 137
170	RESERVED	: 138
171	RESERVED	: 139
172	VCCINT	: 140
173	RESERVED	: 141
174	RESERVED	: 142
175	RESERVED	: 143
176	RESERVED	: 144
177	GNDINT	: 145
178	RESERVED	: 146
179	RESERVED	: 147
180	RESERVED	: 148
181	RESERVED	: 149
182	VCCINT	: 150
183	RESERVED	: 151
184	RESERVED	: 152
185	RESERVED	: 153
186	RESERVED	: 154

187	GNDINT	: 155
188	RESERVED	: 156
189	RESERVED	: 157
190	RESERVED	: 158
191	RESERVED	: 159
192	VCCINT	: 160
193	RESERVED	: 161
194	RESERVED	: 162
195	RESERVED	: 163
196	RESERVED	: 164
197	GNDINT	: 165
198	RESERVED	: 166
199	RESERVED	: 167
200	RESERVED	: 168
201	RESERVED	: 169
202	VCCINT	: 170
203	RESERVED	: 171
204	RESERVED	: 172
205	RESERVED	: 173
206	RESERVED	: 174
207	RESERVED	: 175
208	GNDINT	: 176
209	TDI	: 177
210	nCE	: 178
211	DCLK	: 179
212	DATA0	: 180
213	RESERVED	: 181
214	RESERVED	: 182
215	RESERVED	: 183
216	RESERVED	: 184
217	RESERVED	: 185
218	RESERVED	: 186
219	RESERVED	: 187
220	RESERVED	: 188
221	VCCINT	: 189
222	RESERVED	: 190
223	RESERVED	: 191
224	RESERVED	: 192
225	RESERVED	: 193
226	RESERVED	: 194
227	RESERVED	: 195
228	RESERVED	: 196
229	GNDINT	: 197
230	RESERVED	: 198
231	RESERVED	: 199
232	RESERVED	: 200
233	RESERVED	: 201
234	RESERVED	: 202
235	RESERVED	: 203
236	RESERVED	: 204
237	VCCINT	: 205

238	RESERVED	: 206
239	RESERVED	: 207
240	RESERVED	: 208
241	RESERVED	: 209
242	GNDINT	: 210
243	GNDINT	: 211
244	GNDINT	: 212
245	RESERVED	: 213
246	RESERVED	: 214
247	RESERVED	: 215
248	GNDINT	: 216
249	RESERVED	: 217
250	RESERVED	: 218
251	RESERVED	: 219
252	RESERVED	: 220
253	RESERVED	: 221
254	RESERVED	: 222
255	RESERVED	: 223
256	VCCINT	: 224
257	RESERVED	: 225
258	RESERVED	: 226
259	RESERVED	: 227
260	RESERVED	: 228
261	RESERVED	: 229
262	RESERVED	: 230
263	RESERVED	: 231
264	GNDINT	: 232
265	RESERVED	: 233
266	RESERVED	: 234
267	RESERVED	: 235
268	RESERVED	: 236
269	RESERVED	: 237
270	RESERVED	: 238
271	RESERVED	: 239
272	RESERVED	: 240

A.2 Processing Module

A.2.1 P4Overshell.vhd

A.2.2 MassiveFSM.vhd

A.2.3 P4Control.vhd

A.2.4 P4MathShell.vhd

A.2.5 P4Arithmetic.vhd

A.3 Video Module

A.3.1 top.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4  use ieee.std_logic_arith.all;
5
6  entity top is
7    port(clk, reset                                    : in std_logic;
8         DAN, DANr, ANDREW, ANDREWr                    : in std_logic;
9         ACK, MUXcontrol                               : out std_logic;
10        BUSinline                                     : in std_logic_vector(7 downto
              0);
11        nwren, nouten                                 : out std_logic;
12        RAMaddress                                    : out std_logic_vector(12
              downto 0);
13        tstate                                        : out std_logic_vector(12
              downto 0);
14        RAMdata                                       : out std_logic_vector(7
              downto 0)
15 -- DEBUG
16 --;
17 --goodsignal : out std_logic;
18 --runrunrun: out std_logic;
19 --pager : out std_logic
20 -- DEBUG
21        );
22 end top;
23
24 architecture testing of top is
25
26 component readfsmtest
27   port(clk, reset                                     : in std_logic;
28        DANenable, DANready                           : in std_logic;
29        ANDREWenable, ANDREWready                     : in std_logic;
30        ACK                                           : out std_logic;
31        who, complexready, start                      : out std_logic; -- out
              to RAMfsm
32        complex                                              : out
              std_logic_vector(16 downto 0); -- out to RAMfsm
33        BUSin                                         : in std_logic_vector(7
              downto 0)); -- in from Parallel BUS
34 end component;
35
36 component ramfsmtest
37   port(clk, reset                          : in std_logic;
38        complexgood, who, start             : in std_logic; -- in from readFSM
39        run, page                           : out std_logic; -- out to writeFSM
40        frame                               : out std_logic; -- out to MUX
```

```
            Control
41          BUFFERwe                        : out std_logic; -- out to buffer
42          offset                          : out std_logic_vector(5 downto 0);
43          BUFFERaddress                   : out std_logic_vector(3 downto 0);
44          complexin                       : in std_logic_vector(16 downto 0)
                ; -- out to D/A
45          BUFFERout                       : out std_logic_vector(7 downto 0))
                ; -- out to Buffer RAM
46  end component;
47
48  component writefsmtest2
49    port(clk, reset                       : in std_logic;
50          run, page                       : in std_logic; -- in from ramFSM
51          BUFFERaddress                   : out std_logic_vector(3 downto 0);
                -- out to buffer RAM
52          BUFFERin                        : in std_logic_vector(7 downto 0); --
                 in from Buffer
53          offset                          : in std_logic_vector(5 downto 0); --
                 in from ramFSM
54          nWE, nOE                        : out std_logic; -- out to video RAMs
55          RAMaddress                      : out std_logic_vector(12 downto 0);
                -- out to video RAMs
56          RAMout                          : out std_logic_vector(7 downto 0));
                -- out to video RAMs
57  end component;
58
59  component buff IS
60           PORT
61           (
62                    data              : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
63                    wraddress                 : IN STD_LOGIC_VECTOR (3 DOWNTO
                          0);
64                    rdaddress                 : IN STD_LOGIC_VECTOR (3 DOWNTO
                          0);
65                    wren              : IN STD_LOGIC  := '1';
66                    q                 : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
67           );
68  END component;
69
70
71  --  COMPONENT ROM
72  --       PORT(address                          : IN STD_LOGIC_VECTOR
          (7 DOWNTO 0);
73  --            inclock                        : IN STD_LOGIC;
74  --            outclock                       : IN STD_LOGIC ;
75  --            q                              : OUT STD_LOGIC_VECTOR (24
        DOWNTO 0));
76  --  END COMPONENT;
77
78    signal whosignal, startsignal                                 : std_logic;
79    signal wrenable, complexsignal, runsignal, pagesignal  : std_logic;
```

```
80    signal offsetsignal, who                                    :
          std_logic_vector(5 downto 0);
81    signal BUFFaddress, writeaddress                            :
          std_logic_vector(3 downto 0);
82    signal XMsecond, BUFFin, dataline                           :
          std_logic_vector(7 downto 0);
83    signal complexdataline                                      :
          std_logic_vector(16 downto 0);
84
85  begin
86
87
88  tstate <= "ZZZZZZZZZZZZZ";
89
90  --clk <= clksignal;
91  --resetsignal <= reset;
92
93  -- DEBUG
94  --goodsignal <= complexsignal;
95  --runrunrun <= runsignal;
96  --pager <= pagesignal;
97  -- DEBUG
98
99  READER: readfsmtest
100   port map(clk => clk,
101            reset => reset,
102            DANenable => DAN,
103            DANready => DANr,
104            ANDREWenable => ANDREW,
105            ANDREWready => ANDREWr,
106            ACK => ACK,
107            who => whosignal,
108            complexready => complexsignal,
109            start => startsignal,
110            complex => complexdataline,
111            BUSin => BUSinline);
112
113 RAMMER: ramfsmtest
114   port map(clk => clk,
115            reset => reset,
116            complexgood => complexsignal,
117            who => whosignal,
118            start => startsignal,
119            run => runsignal,
120            page => pagesignal,
121            frame => MUXcontrol,
122            BUFFERwe => wrenable,
123            offset => offsetsignal,
124            BUFFERaddress => writeaddress,
125            complexin => complexdataline,
126            BUFFERout => dataline);
```

```
127
128 WRITER: writefsmtest2
129    port map(clk => clk,
130             reset => reset,
131             run => runsignal,
132             page => pagesignal,
133             BUFFERaddress => BUFFaddress,
134             BUFFERin => BUFFin,
135             offset => offsetsignal,
136            nWE => nwren,
137             nOE => nouten,
138             RAMaddress => RAMaddress,
139             RAMout => RAMdata);
140
141 BUFFERER: buff
142    PORT map(data => dataline,
143                     wraddress => writeaddress,
144             rdaddress => BUFFaddress,
145             wren => wrenable,
146             q => BUFFin);
147
148 end architecture testing;
```

A.3.2 readfsm.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4  use ieee.std_logic_arith.all;
5
6  entity readfsmtest is
7    port(clk, reset                                  : in std_logic;
8         DANenable, DANready                         : in std_logic;
9         ANDREWenable, ANDREWready                   : in std_logic;
10        ACK, start                                  : out std_logic;
11        who, complexready                           : out std_logic; -- out
              to RAMfsm
12        complex                                              : out
              std_logic_vector(16 downto 0); -- out to RAMfsm
13        BUSin                                       : in std_logic_vector(7
              downto 0)); -- in from Parallel BUS
14 end readfsmtest;
15
16 architecture behavioral of readfsmtest is
17
18   type StateType is (clear, idle, waitdata, readdata1, readdata2);
19   attribute enum_encoding : string;
20   attribute enum_encoding of StateType:
21     type is "000 001 010 011 100";
22   -- 000 clear
23   -- 001 idle
24   -- 010 waitdata
25   -- 011 readdata1
26   -- 100 readdata2
27
28   signal p_s, n_s : StateType;
29
30   signal complex_r, complex_i   : std_logic_vector(7 downto 0);
31   signal delay, delay0          : std_logic_vector(3 downto 0);
32   signal count, count0          : std_logic_vector(11 downto 0);
33   signal complexlatch, complexlatch0          : std_logic_vector(16
       downto 0);
34   signal valid, dataready, cready, cready0       : std_logic;
35
36 begin
37
38
39   valid <= DANenable xor ANDREWenable;
40   complex <= complexlatch;
41   complexready <= cready;
42
43   clocked:process(clk)
44   begin
45     if clk'event and clk = '1' then
```

```
46        if reset = '1' then
47          p_s <= clear;
48        else p_s <= n_s;
49        delay <= delay0;
50        count <= count0;
51        complexlatch <= complexlatch0;
52        cready <= cready0;
53        end if;
54
55      if DANenable = '1' then
56        who <= '0';
57        dataready <= DANready;
58      elsif ANDREWenable = '1' then
59        who <= '1';
60        dataready <= ANDREWready;
61      else who <= '0';
62        dataready <= '0';
63      end if;
64
65
66
67      end if;
68    end process;
69
70    states:process(p_s, n_s, count, delay, complexlatch, complexlatch0,
        valid, dataready, count0, BUSin, DAnenable, ANDREWenable)
71    begin
72      case p_s is
73        when clear =>                    -- 000
74          count0 <= "000000000000";
75          delay0 <= "0000";
76          complexlatch0 <= "0000000000000000";
77          start <= '0';
78          ACK <= '0';
79          cready0 <= '0';
80          n_s <= idle;
81
82        when idle =>                     -- 001
83          ACK <= '0';
84          start <= '0';
85          count0 <= "000000000000";
86          complexlatch0 <= "0000000000000000";
87          cready0 <= '0';
88          delay0 <= "0000";
89          if valid = '1' then
90            n_s <= waitdata;
91          else n_s <= p_s;
92          end if;
93
94        when waitdata =>                 -- 010
95          complexlatch0 <= complexlatch;
```

```
96          count0 <= count;
97          cready0 <= '0';
98          ACK <= '0';
99          delay0 <= delay + 1;
100         if valid = '1' then
101           start <= '1';
102           if dataready = '1' then
103             n_s <= readdata1;
104           else n_s <= p_s;
105           end if;
106         else
107           start <= '0';
108           if delay = "1111" then
109             n_s <= idle;
110           else n_s <= p_s;
111           end if;
112         end if;
113
114       when readdata1 =>                -- 011
115         delay0 <= "0000";
116         complexlatch0 <= complexlatch;
117         if count(0) = '0' then
118           cready0 <= '0';
119           if BUSin(7) = '0' then
120             complex_r <= BUSin;
121           else complex_r <= not(BUSin) + 1;
122           end if;
123         else
124           cready0 <= '1';
125           if BUSin(7) = '0' then
126             complex_i <= BUSin;
127           else complex_i <= not(BUSin) + 1;
128           end if;
129           complexlatch0 <= (("000000000" & complex_r) * ("000000000" &
                complex_r)) + (("000000000" & complex_i) * ("000000000" &
                complex_i));
130         end if;
131         ACK <= '1';
132         count0 <= count + 1;
133         n_s <= readdata2;
134
135       when readdata2 =>              -- 100
136         ACK <= '1';
137         start <= '1';
138         count0 <= count;
139         complexlatch0 <= complexlatch;
140         cready0 <= cready;
141         if dataready = '0' then
142           n_s <= waitdata;
143         else n_s <= p_s;
144         end if;
```

```
145
146       end case;
147     end process states;
148
149  end architecture behavioral;
```

A.3.3 ramfsm.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4  use ieee.std_logic_arith.all;
5
6  entity ramfsmtest is
7    port(clk, reset                    : in std_logic;
8         complexgood, who, start       : in std_logic; -- in from readFSM
9         run, page                     : out std_logic; -- out to writeFSM
10        frame                         : out std_logic; -- out to MUX
              Control
11        BUFFERwe                      : out std_logic; -- out to buffer
12        whopass                       : out std_logic;
13        offset                        : out std_logic_vector(4 downto 0);
14        BUFFERaddress                 : out std_logic_vector(3 downto 0);
15        complexin                     : in std_logic_vector(16 downto 0)
              ; -- out to D/A
16        BUFFERout                     : out std_logic_vector(7 downto 0))
              ; -- out to Buffer RAM
17
18 end ramfsmtest;
19
20 architecture behavioral of ramfsmtest is
21
22   type StateType is (clear, idle, waitstate, read1, read2, downsample,
         writebuffer1, writebuffer2, writebuffer3,
23                      runstate, skipstate1, skipstate2);
24   attribute enum_encoding : string;
25   attribute enum_encoding of StateType:
26     type is "0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011
           ";
27   -- 0000 = clear
28   -- 0010 = idle
29   -- 0011 = waitstate
30   -- 0100 = read
31   -- 0101 = downsample
32   -- 0110 = writebuffer
33
34   signal p_s, n_s : StateType;
35
36   signal accumulate, accumulate0                                    :
         std_logic_vector(23 downto 0);
37   signal BUFFout, BUFFout0                                          :
         std_logic_vector(7 downto 0);
38   signal column, column0                                            :
         std_logic_vector(4 downto 0);
39   signal BUFFadd, BUFFadd0, skipcount, skipcount0                   :
         std_logic_vector(3 downto 0);
40   signal averagecount, averagecount0, samples, samples0             :
```

```
        std_logic_vector(2 downto 0);
41    signal wholatch, wholatch0, BUFFpage, BUFFpage0                    :
        std_logic;
42    signal frameout, frameout0, run0                                   :
        std_logic;
43
44
45
46  begin
47
48    frame <= frameout;
49    BUFFERaddress <= BUFFadd;
50    BUFFERout <= BUFFout;
51    page <= BUFFpage;
52    offset <= column;
53    whopass <= wholatch;
54
55    clocked:process(clk)
56    begin
57      if clk'event and clk = '1' then
58        if reset = '1' then
59          p_s <= clear;
60        else p_s <= n_s;
61        end if;
62        accumulate <= accumulate0;
63        averagecount <= averagecount0;
64        samples <= samples0;
65        skipcount <= skipcount0;
66        wholatch <= wholatch0;
67        column <= column0;
68        frameout <= frameout0;
69        BUFFpage <= BUFFpage0;
70        run <= run0;
71        BUFFadd <= BUFFadd0;
72        BUFFout <= BUFFout0;
73      end if;
74    end process;
75
76    states:process(p_s, n_s)
77    begin
78      case p_s is
79        when clear =>                          -- 0000
80          run0 <= '0';
81          accumulate0 <= "000000000000000000000000";
82          samples0 <= "000";
83          averagecount0 <= "000";
84          column0 <= "00010";
85          frameout0 <= '0';
86          BUFFpage0 <= '0';
87          BUFFadd0 <= "0000";
88          BUFFout0 <= "00000000";
```

```
89          BUFFERwe <= '0';
90          skipcount0 <= "0000";
91          wholatch0 <= '0';
92
93          n_s <= idle;
94
95        when idle =>                              -- 0001
96          run0 <= '0';
97          accumulate0 <= "000000000000000000000000";
98          samples0 <= "000";
99          averagecount0 <= "000";
100         BUFFpage0 <= '0';
101         column0 <= "00010";
102         frameout0 <= frameout;
103         BUFFadd0 <= "0000";
104         BUFFout0 <= "00000000";
105         BUFFERwe <= '0';
106         skipcount0 <= "0000";
107         wholatch0 <= '0';
108
109         if start = '1' then
110           n_s <= waitstate;
111         else n_s <= p_s;
112         end if;
113
114       when waitstate=>                          -- 0010
115         run0 <= '0';
116         accumulate0 <= accumulate;
117         samples0 <= samples;
118         averagecount0 <= averagecount;
119         BUFFpage0 <= BUFFpage;
120         column0 <= column;
121         BUFFadd0 <= "0000";
122         BUFFout0 <= "00000000";
123         BUFFERwe <= '0';
124         wholatch0 <= wholatch;
125
126         if start = '0' then
127           if wholatch = '1' then
128             frameout0 <= not frameout;
129             n_s <= idle;
130           else frameout0 <= frameout;
131             n_s <= p_s;
132           end if;
133         elsif complexgood = '1' then
134           frameout0 <= frameout;
135           n_s <= read1;
136         else frameout0 <= frameout;
137           n_s <= p_s;
138         end if;
139
```

```
140         when read1 =>                          -- 0011
141           run0 <= '0';
142           column0 <= column;
143           BUFFpage0 <= BUFFpage;
144           BUFFadd0 <= "0000";
145           BUFFout0 <= "00000000";
146           samples0 <= samples;
147           accumulate0 <= ("0000000" & complexin) + accumulate;
148           averagecount0 <= averagecount + 1;
149           wholatch0 <= who;
150
151           n_s <= read2;
152
153         when read2 =>                          -- 0100
154           run0 <= '0';
155           column0 <= column;
156           samples0 <= samples;
157           BUFFERwe <= '0';
158           BUFFadd0 <= "0000";
159           BUFFout0 <= "00000000";
160           accumulate0 <= accumulate;
161           averagecount0 <= averagecount;
162           wholatch0 <= wholatch;
163
164           if averagecount = "110" then
165             n_s <= downsample;
166           elsif complexgood = '0' then
167             n_s <= waitstate;
168           else n_s <= p_s;
169           end if;
170
171         when downsample =>                     -- 0101
172           run0 <= '0';
173           column0 <= column;
174           averagecount0 <= "000";
175           samples0 <= samples;
176           accumulate0 <= accumulate + ( accumulate(22 downto 0) & '0');
177             -- divide by 6 to average + multiply by 18 to scale = multiply
                  by 3
178           BUFFadd0 <= "0000";
179           BUFFout0 <= "00000000";
180           wholatch0 <= wholatch;
181
182           n_s <= writebuffer1;
183
184         when writebuffer1 =>                   -- 0110
185           run0 <= '0';
186           column0 <= column;
187           accumulate0 <= accumulate;
188           BUFFadd0 <= BUFFpage & samples;
189           BUFFout0 <= accumulate(23 downto 16);
```

```
190           BUFFERwe <= '0';
191           samples0 <= samples;
192           wholatch0 <= wholatch;
193
194           n_s <= writebuffer2;
195
196         when writebuffer2 =>                -- 0111
197           run0 <= '0';
198           column0 <= column;
199           accumulate0 <= accumulate;
200           BUFFadd0 <= BUFFadd;
201           BUFFout0 <= BUFFout;
202           BUFFERwe <= '1';
203           samples0 <= samples;
204           wholatch0 <= wholatch;
205
206           n_s <= writebuffer3;
207
208         when writebuffer3 =>                -- 1000
209           column0 <= column;
210                   accumulate0 <= "00000000000000000000000";
211           BUFFadd0 <= BUFFadd;
212           BUFFout0 <= BUFFout;
213           BUFFERwe <= '0';
214           samples0 <= samples;
215           wholatch0 <= wholatch;
216
217           if samples = "111" then
218             run0 <= '1';
219           else run0 <= '0';
220           end if;
221
222           n_s <= runstate;
223
224         when runstate =>                          -- 1001
225           run0 <= '0';
226           accumulate0 <= accumulate;
227           BUFFadd0 <= "0000";
228           BUFFout0 <= "00000000";
229           BUFFERwe <= '0';
230           wholatch0 <= wholatch;
231
232           if column = "10110" then
233             if samples = "111" then
234               column0 <= "00010";
235               samples0 <= "000";
236               BUFFpage0 <= not BUFFpage;
237               n_s <= skipstate1;
238             else column0 <= column;
239               samples0 <= samples + 1;
240               BUFFpage0 <= BUFFpage;
```

```
241             n_s <= read1;
242           end if;
243         elsif samples = "111" then
244           column0 <= column + 1;
245           samples0 <= "000";
246           BUFFpage0 <= not BUFFpage;
247           n_s <= read1;
248         else column0 <= column;
249           samples0 <= samples + 1;
250           BUFFpage0 <= BUFFpage;
251           n_s <= read1;
252         end if;
253
254       when skipstate1 =>                -- 1010
255         BUFFERwe <= '0';
256         wholatch0 <= wholatch;
257         column0 <= "00010";
258         samples0 <= "000";
259         skipcount0 <= skipcount;
260         if complexgood = '1' then
261           n_s <= skipstate2;
262         else n_s <= p_s;
263         end if;
264
265       when skipstate2 =>                -- 1011
266         BUFFERwe <= '0';
267         wholatch0 <= wholatch;
268         column0 <= "00010";
269         samples0 <= "000";
270         if complexgood = '0' then
271           if skipcount = "1111" then
272             skipcount0 <= "0000";
273             n_s <= waitstate;
274           else skipcount0 <= skipcount + 1;
275             n_s <= skipstate1;
276           end if;
277         else skipcount0 <= skipcount;
278           n_s <= p_s;
279         end if;
280
281     end case;
282
283   end process states;
284 end architecture behavioral;
```

A.3.4 writefsmtest2.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4  use ieee.std_logic_arith.all;
5
6  entity writefsmtest2 is
7    port(clk, reset                    : in std_logic;
8
9  --direct inputs
10 --      switch1, switch2, switch3    : in std_logic_vector(3 downto 0);
11 --      mode                         : in std_logic_vector(1 downto 0);
12
13         run, page                    : in std_logic; -- in from ramFSM
14         BUFFERaddress                : out std_logic_vector(3 downto 0);
              -- out to buffer RAM
15         BUFFERin                     : in std_logic_vector(7 downto 0); --
              in from Buffer
16         offset                       : in std_logic_vector(4 downto 0); --
              in from ramFSM
17         whopass                      : in std_logic; -- in from ramFSM
18         nWE, nOE                     : out std_logic; -- out to video RAMs
19         RAMaddress                   : out std_logic_vector(12 downto 0);
              -- out to video RAMs
20         RAMout                       : out std_logic_vector(7 downto 0));
              -- out to video RAMs
21 end writefsmtest2;
22
23 architecture behavioral of writefsmtest2 is
24
25   type StateType is (clear, idle, readin1, readin2, drawgraph1,
       addressstate, drawgraph2);
26   attribute enum_encoding : string;
27   attribute enum_encoding of StateType:
28     type is "000_001_010_011_100_101_110";
29   -- 000 = clear
30   -- 001 = idle
31   -- 010 = readin1
32   -- 011 = readin2
33   -- 100 = drawgraph1
34   -- 101 = drawgraph2
35
36
37   signal p_s, n_s : StateType;
38
39   signal RAMaddress0                                              :
       std_logic_vector(12 downto 0);
40   signal piece, RAMout0, RAMout1, s0, s1, s2, s3, s4, s5, s6, s7
           : std_logic_vector(7 downto 0);
41   signal count, count0                                            :
```

```
        std_logic_vector(6 downto 0);
42    signal lowcount, lowcount0                                    :
        std_logic_vector(3 downto 0);
43    signal rowcount, rowcount0                                    :
        std_logic_vector(2 downto 0);
44    signal tick, tick0                                            :
        std_logic_vector(2 downto 0);
45    signal pagelatch, pagelatch0, wholatch, wholatch0             :
        std_logic;
46    signal nWElatch, nWElatch0, nOElatch, nOElatch0               :
        std_logic;
47
48  begin
49
50    nWE <= nWElatch;
51    nOE <= nOElatch;
52    RAMout <= RAMout1;
53
54    clocked:process(clk)
55    begin
56      if clk'event and clk = '1' then
57        if reset = '1' then
58          p_s <= clear;
59        else count <= count0;
60          tick <= tick0;
61          lowcount <= lowcount0;
62          rowcount <= rowcount0;
63  --          RAMout <= RAMout0;
64  RAMout1 <= RAMout0;
65          RAMaddress <= RAMaddress0;
66          pagelatch <= pagelatch0;
67          wholatch <= wholatch0;
68          nWElatch <= nWElatch0;
69          nOElatch <= nOElatch0;
70          p_s <= n_s;
71        end if;
72      end if;
73    end process;
74
75    states:process(p_s, n_s, lowcount0, lowcount, rowcount0, rowcount,
        tick, tick0, wholatch, wholatch0,
76                   whopass, count, count0, s0, s1, s2, s3, s4,  s5, s6, s7
                       , piece,  BUFFERin, offset,
77                   run, page, RAMaddress0, RAMout0, nWElatch, nWElatch0,
                       nOElatch, nOElatch0)
78    begin
79      case p_s is
80
81        when clear =>                        -- 0000
82          tick0 <= "000";
83          count0 <= "1001000";
```

```
84            lowcount0 <= "0000";
85            rowcount0 <= "001";
86            RAMaddress0 <= "0000000000000";
87            RAMout0 <= "00000000";
88            pagelatch0 <= '0';
89            wholatch0 <= '0';
90            nWElatch0 <= '1';
91            nOElatch0 <= '1';
92
93            n_s <= idle;
94
95          when idle =>                          -- 0001
96            tick0 <= "000";
97            count0 <= "1001000";
98            lowcount0 <= lowcount;
99            rowcount0 <= rowcount;
100           RAMaddress0 <= "0000000000000";
101           RAMout0 <= "00000000";
102           BUFFERaddress <= "0000";
103
104           if run = '1' then
105             pagelatch0 <= page;
106             wholatch0 <= whopass;
107             n_s <= readin1;
108           else pagelatch0 <= '0';
109             wholatch0 <= '0';
110             n_s <= p_s;
111           end if;
112
113         when readin1 =>                         -- 0010
114           tick0 <= tick;
115           count0 <= count;
116           lowcount0 <= lowcount;
117           rowcount0 <= rowcount;
118           RAMaddress0 <= "0000000000000";
119           RAMout0 <= "00000000";
120           BUFFERaddress <= pagelatch & tick;
121           pagelatch0 <= pagelatch;
122           nWElatch0 <= '1';
123           nOElatch0 <= '1';
124           piece <= "00000000";
125
126           n_s <= readin2;
127
128         when readin2 =>                         -- 0011
129           count0 <= count;
130           lowcount0 <= lowcount;
131           rowcount0 <= rowcount;
132           RAMaddress0 <= "0000000000000";
133           RAMout0 <= "00000000";
134           BUFFERaddress <= pagelatch & tick;
```

```
135         pagelatch0 <= pagelatch;
136         nWElatch0 <= '1';
137         nOElatch0 <= '1';
138         piece <= "00000000";
139
140         tick0 <= tick + 1;
141
142         case tick is
143           when "000" =>
144             s0 <= BUFFERin;
145             n_s <= readin2;
146           when "001" =>
147             s1 <= BUFFERin;
148             n_s <= readin2;
149           when "010" =>
150             s2 <= BUFFERin;
151             n_s <= readin2;
152           when "011" =>
153             s3 <= BUFFERin;
154             n_s <= readin2;
155           when "100" =>
156             s4 <= BUFFERin;
157             n_s <= readin2;
158           when "101" =>
159             s5 <= BUFFERin;
160             n_s <= readin2;
161           when "110" =>
162             s6 <= BUFFERin;
163             n_s <= readin2;
164           when "111" =>
165             s7 <= BUFFERin;
166             n_s <= drawgraph1;
167           when others =>
168             null;
169         end case;
170
171       when drawgraph1 =>                -- 0100
172
173         if count > s0 then
174           piece(0) <= '0';
175         else piece(0) <= '1';
176         end if;
177         if count > s1 then
178           piece(1) <= '0';
179         else piece(1) <= '1';
180         end if;
181         if count > s2 then
182           piece(2) <= '0';
183         else piece(2) <= '1';
184         end if;
185         if count > s3 then
```

```
186             piece(3) <= '0';
187           else piece(3) <= '1';
188           end if;
189           if count > s4 then
190             piece(4) <= '0';
191           else piece(4) <= '1';
192           end if;
193           if count > s5 then
194             piece(5) <= '0';
195           else piece(5) <= '1';
196           end if;
197           if count > s6 then
198             piece(6) <= '0';
199           else piece(6) <= '1';
200           end if;
201           if count > s7 then
202             piece(7) <= '0';
203           else piece(7) <= '1';
204           end if;
205
206           tick0 <= "000";
207           count0 <= count - 1;
208           lowcount0 <= lowcount;
209           rowcount0 <= rowcount;
210           RAMaddress0 <= (((wholatch & rowcount) & offset) & lowcount);
211           RAMout0 <= piece;
212           BUFFERaddress <= "0000";
213           pagelatch0 <= pagelatch;
214           nWElatch0 <= '1';
215           nOElatch0 <= '1';
216
217           n_s <= addressstate;
218
219         when addressstate =>                    -- 0101
220           tick0 <= tick;
221           count0 <= count;
222           lowcount0 <= lowcount;
223           rowcount0 <= rowcount;
224           RAMaddress0 <= (((wholatch & rowcount) & offset) & lowcount);
225 --          RAMout0 <= piece;
226           BUFFERaddress <= "0000";
227           pagelatch0 <= pagelatch;
228           nWElatch0 <= '0';
229           nOElatch0 <= '0';
230
231           n_s <= drawgraph2;
232
233         when drawgraph2 =>          -- 0110
234           tick0 <= tick;
235           count0 <= count;
236           RAMaddress0 <= (((wholatch & rowcount) & offset) & lowcount);
```

```
237 --          RAMout0 <= piece;
238           BUFFERaddress <= "0000";
239           pagelatch0 <= pagelatch;
240           nWElatch0 <= '1';
241           nOElatch0 <= '1';
242
243           if lowcount = "1011" then
244             if count = "0000000" then
245               lowcount0 <= "0000";
246               rowcount0 <= "001";
247               n_s <= idle;
248             else lowcount0 <= "0000";
249               rowcount0 <= rowcount + 1;
250               n_s <= readin1;
251             end if;
252           else lowcount0 <= lowcount + 1;
253             rowcount0 <= rowcount;
254             n_s <= drawgraph1;
255           end if;
256       end case;
257     end process states;
258 end architecture behavioral;
```

A.3.5 buff.vhd

```
1  -- megafunction wizard: %Dual-port RAM%
2  -- GENERATION: STANDARD
3  -- VERSION: WM1.0
4  -- MODULE: altdpram
5
6  -- ============================================================
7  -- File Name: buff.vhd
8  -- Megafunction Name(s):
9  --                         altdpram
10 -- ============================================================
11 -- ************************************************************
12 -- THIS IS A WIZARD GENERATED FILE. DO NOT EDIT THIS FILE!
13 -- ************************************************************
14
15
16 --       Copyright (C) 1988-2002 Altera Corporation
17
18 --       Any megafunction design, and related net list (encrypted or
       decrypted),
19 --       support information, device programming or simulation file, and
       any other
20 --       associated documentation or information provided by Altera or a
       partner
21 --       under Altera's Megafunction Partnership Program may be used only
        to
22 --       program PLD devices (but not masked PLD devices) from Altera.
        Any other
23 --       use of such megafunction design, net list, support information,
       device
24 --       programming or simulation file, or any other related
       documentation or
25 --       information is prohibited for any other purpose, including, but
       not
26 --       limited to modification, reverse engineering, de-compiling, or
       use with
27 --       any other silicon devices, unless such use is explicitly
       licensed under
28 --       a separate agreement with Altera or a megafunction partner.
        Title to
29 --       the intellectual property, including patents, copyrights,
       trademarks,
30 --       trade secrets, or maskworks, embodied in any such megafunction
       design,
31 --       net list, support information, device programming or simulation
       file, or
32 --       any other related documentation or information provided by
       Altera or a
33 --       megafunction partner, remains with Altera, the megafunction
       partner, or
```

```
34 --        their respective licensors.  No other licenses, including any
       licenses
35 --        needed under any third party's intellectual property, are
       provided herein.
36
37 LIBRARY ieee;
38 USE ieee.std_logic_1164.all;
39
40 ENTITY buff IS
41         PORT
42         (
43                 data            : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
44                 wraddress               : IN STD_LOGIC_VECTOR (3 DOWNTO
                       0);
45                 rdaddress               : IN STD_LOGIC_VECTOR (3 DOWNTO
                       0);
46                 wren            : IN STD_LOGIC  := '1';
47                 q               : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
48         );
49 END buff;
50
51
52 ARCHITECTURE SYN OF buff IS
53
54         SIGNAL sub_wire0        : STD_LOGIC_VECTOR (7 DOWNTO 0);
55
56
57
58         COMPONENT altdpram
59         GENERIC (
60                 width           : NATURAL;
61                 widthad         : NATURAL;
62                 indata_reg              : STRING;
63                 wraddress_reg           : STRING;
64                 wrcontrol_reg           : STRING;
65                 rdaddress_reg           : STRING;
66                 rdcontrol_reg           : STRING;
67                 outdata_reg             : STRING;
68                 indata_aclr             : STRING;
69                 wraddress_aclr          : STRING;
70                 wrcontrol_aclr          : STRING;
71                 rdaddress_aclr          : STRING;
72                 rdcontrol_aclr          : STRING;
73                 outdata_aclr            : STRING;
74                 lpm_hint                : STRING
75         );
76         PORT (
77                         wren    : IN STD_LOGIC ;
78                         q       : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
79                         data    : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
80                         rdaddress       : IN STD_LOGIC_VECTOR (3 DOWNTO
```

```
                                0);
81                              wraddress       : IN STD_LOGIC_VECTOR (3 DOWNTO
                                0)
82          );
83          END COMPONENT;
84
85  BEGIN
86          q    <= sub_wire0(7 DOWNTO 0);
87
88          altdpram_component : altdpram
89          GENERIC MAP (
90                  WIDTH => 8,
91                  WIDTHAD => 4,
92                  INDATA_REG => "UNREGISTERED",
93                  WRADDRESS_REG => "UNREGISTERED",
94                  WRCONTROL_REG => "UNREGISTERED",
95                  RDADDRESS_REG => "UNREGISTERED",
96                  RDCONTROL_REG => "UNREGISTERED",
97                  OUTDATA_REG => "UNREGISTERED",
98                  INDATA_ACLR => "OFF",
99                  WRADDRESS_ACLR => "OFF",
100                 WRCONTROL_ACLR => "OFF",
101                 RDADDRESS_ACLR => "OFF",
102                 RDCONTROL_ACLR => "OFF",
103                 OUTDATA_ACLR => "OFF",
104                 LPM_HINT => "USE_EAB=ON"
105         )
106         PORT MAP (
107                 wren => wren,
108                 data => data,
109                 rdaddress => rdaddress,
110                 wraddress => wraddress,
111                 q => sub_wire0
112         );
113
114
115
116 END SYN;
117
118 -- ============================================================
119 -- CNX file retrieval info
120 -- ============================================================
121 -- Retrieval info: PRIVATE: WidthData NUMERIC "8"
122 -- Retrieval info: PRIVATE: WidthAddr NUMERIC "4"
123 -- Retrieval info: PRIVATE: Clock NUMERIC "3"
124 -- Retrieval info: PRIVATE: rden NUMERIC "0"
125 -- Retrieval info: PRIVATE: REGdata NUMERIC "1"
126 -- Retrieval info: PRIVATE: REGwraddress NUMERIC "1"
127 -- Retrieval info: PRIVATE: REGwren NUMERIC "1"
128 -- Retrieval info: PRIVATE: REGrdaddress NUMERIC "1"
129 -- Retrieval info: PRIVATE: REGrren NUMERIC "1"
```

```
130 -- Retrieval info: PRIVATE: REGq NUMERIC "0"
131 -- Retrieval info: PRIVATE: enable NUMERIC "0"
132 -- Retrieval info: PRIVATE: CLRdata NUMERIC "0"
133 -- Retrieval info: PRIVATE: CLRwraddress NUMERIC "0"
134 -- Retrieval info: PRIVATE: CLRwren NUMERIC "0"
135 -- Retrieval info: PRIVATE: CLRrdaddress NUMERIC "0"
136 -- Retrieval info: PRIVATE: CLRrren NUMERIC "0"
137 -- Retrieval info: PRIVATE: CLRq NUMERIC "0"
138 -- Retrieval info: PRIVATE: BlankMemory NUMERIC "1"
139 -- Retrieval info: PRIVATE: MIFfilename STRING ""
140 -- Retrieval info: PRIVATE: UseLCs NUMERIC "0"
141 -- Retrieval info: CONSTANT: WIDTH NUMERIC "8"
142 -- Retrieval info: CONSTANT: WIDTHAD NUMERIC "4"
143 -- Retrieval info: CONSTANT: INDATA_REG STRING "UNREGISTERED"
144 -- Retrieval info: CONSTANT: WRADDRESS_REG STRING "UNREGISTERED"
145 -- Retrieval info: CONSTANT: WRCONTROL_REG STRING "UNREGISTERED"
146 -- Retrieval info: CONSTANT: RDADDRESS_REG STRING "UNREGISTERED"
147 -- Retrieval info: CONSTANT: RDCONTROL_REG STRING "UNREGISTERED"
148 -- Retrieval info: CONSTANT: OUTDATA_REG STRING "UNREGISTERED"
149 -- Retrieval info: CONSTANT: INDATA_ACLR STRING "OFF"
150 -- Retrieval info: CONSTANT: WRADDRESS_ACLR STRING "OFF"
151 -- Retrieval info: CONSTANT: WRCONTROL_ACLR STRING "OFF"
152 -- Retrieval info: CONSTANT: RDADDRESS_ACLR STRING "OFF"
153 -- Retrieval info: CONSTANT: RDCONTROL_ACLR STRING "OFF"
154 -- Retrieval info: CONSTANT: OUTDATA_ACLR STRING "OFF"
155 -- Retrieval info: CONSTANT: LPM_HINT STRING "USE_EAB=ON"
156 -- Retrieval info: USED_PORT: data 0 0 8 0 INPUT NODEFVAL data[7..0]
157 -- Retrieval info: USED_PORT: q 0 0 8 0 OUTPUT NODEFVAL q[7..0]
158 -- Retrieval info: USED_PORT: wraddress 0 0 4 0 INPUT NODEFVAL wraddress
        [3..0]
159 -- Retrieval info: USED_PORT: rdaddress 0 0 4 0 INPUT NODEFVAL rdaddress
        [3..0]
160 -- Retrieval info: USED_PORT: wren 0 0 0 0 INPUT VCC wren
161 -- Retrieval info: CONNECT: @data 0 0 8 0 data 0 0 8 0
162 -- Retrieval info: CONNECT: q 0 0 8 0 @q 0 0 8 0
163 -- Retrieval info: CONNECT: @wraddress 0 0 4 0 wraddress 0 0 4 0
164 -- Retrieval info: CONNECT: @rdaddress 0 0 4 0 rdaddress 0 0 4 0
165 -- Retrieval info: CONNECT: @wren 0 0 0 0 wren 0 0 0 0
```