Design and Development of a One Gigasample per Second Radar Data Acquisition System

Ryan Kelly Eakin

B.S.E.E., University of Kansas, 1999

Submitted to the
Department of Electrical Engineering and Computer Science
and the Faculty of the Graduate School of
the University of Kansas
in partial fulfillment of the requirements for the degree of
Master of Science.

_____

Professor in Charge

_____

_____

Committee Members

_____

Date Thesis Presented

ACKNOWLEDGEMENTS

I would like to thank God for blessing me with a wonderful family who sacrificed so much to make this degree possible. Thanks to Dr. Allen and Dr. Gogineni for allowing me to work on such a challenging project. The experience has been irreplaceable and played a key role in landing my current job.

ABSTRACT

To more accurately determine the role of glacier ice systems in sea level rise, glacier accumulation rate measurements are needed. This information has traditionally been gathered using core samples resulting in a measurement error exceeding 20 percent when sparse measurements are extrapolated over the entire ice sheet. To improve temporal and spatial accuracy of these measurements, the University of Kansas Radar and Remote Sensing Laboratory is developing a wideband, 600- to 900-MHz pulse compression airborne accumulation radar system. This thesis presents a 1 Gsample/sec radar data acquisition system developed for this accumulation radar. The system has the ability to digitize signals with bandwidth as high as 500-MHz and frequencies as high as 2.2 GHz. A special digital timing system is also presented which provides precision timing signals to the acquisition system as well as general timing for the entire radar. The analog to digital conversion section of the system was shown to operate at 1 Gsample/sec with an input up to 2.2 GHz as stated in the manufacturers datasheet. The timing system was shown to operate up to 2.4 GHz. The entire acquisition system could not be completed because one card in the system could not be made to work properly due to manufacturability issue. The logical design of this card has been shown to be correct and the system could be completed with a revision of this one board.

Table of Contents

List of Figures

List of Tables

CHAPTER 1 INTRODUCTION

To more accurately determine the role of glacier ice systems in sea level rise, glacier accumulation rate measurements are needed. Historically, this information has been gathered using core samples collected at various points on a glacier resulting in an accumulation rate measurement error of 20 to 24 percent [1]. For improved accuracy, remote sensing techniques are required to improve temporal and spatial coverage of accumulation measurements.

During the summers of 1998-99, the University of Kansas Radar and Remote Sensing Laboratory (RSL) operated a prototype ground based frequency modulated continuous wave (FMCW) radar on the Greenland ice sheet. Results from the prototype demonstrated a long-term accumulation measurement error of less than 5 percent. The results also showed the optimum frequency of operation for an accumulation radar was 500 to 1000 MHz [1]. Based on these results, it was desired to build an airborne pulse radar operating from 600 to 900 MHz. This project was the first attempt at building a wide bandwidth radar data acquisition system for this high-resolution radar.

The enabling technology for this system is the introduction of gigahertz speed analog to digital converters (ADCs) onto the consumer market. The converters selected for this project can directly digitize signals with frequencies as high as 2.2 GHz with bandwidths up to 500 MHz. The immediate goal of the acquisition system was to directly digitize the accumulation radar output using a single acquisition channel. A single acquisition channel is crucial for a high resolution radar because it eliminates the

need for quadrature demodulation which creates spurious components in the radar analog output resulting in ghost images in the processed data.

The final system consists of a computer system and five unique circuit boards: ADC Card, Averaging Card, Multiplexer Card, Timing Main Card, and Timing Mezzanine Card. The ADC Card samples the radar analog signal at up to 1 Gsample/sec and outputs digital signals in sixteen 8-bit streams. The Averaging Card buffers those digital streams and performs a coherent averaging algorithm using four digital signal processors. The Multiplexer Card is used to switch multiple acquisition channels into a single channel for transfer to the acquisition computer. The final two boards (Main Card and Mezzanine Card) create a precision timing system that is required to maintain coherency with system clock speeds of 1 GHz.

In this project, each of the five circuits was designed and constructed. The timing system was completed and shown to work with clock frequencies as high as 2.4 GHz. The ADC Card was also validated and shown to have a frequency response similar to the manufacturers ratings up to 2.2 GHz. The Averaging Card concept was validated but the circuit could not be made to work properly due to manufacturability issues. The board's functionality was correct but the DSP's were found to operate intermittently either due to a solderability problem or inability to implement "jumper wire" changes. It is believed that this board would operate if a PCB revision was implemented. The Multiplexer Card was shown to work although the full capabilities could not be tested without fully functional Averaging Cards.

Chapter 2 is a review of the theory necessary to understand the full potential of this wide bandwidth acquisition system. Chapter 3 is a detailed description of the ADC

Card, Averaging Card, Multiplexer Card, and computer interface. Chapter 4 is a detailed description of the precision timing system including the Main Card, Mezzanine Card, and timing system microcontroller. Chapter 5 is a description of special test and measurement circuits that were built for system validation and debugging. Chapter 6 presents laboratory measurement results. Chapter 7 presents conclusions and recommendations for further development.

CHAPTER 2 THEORY AND CONCEPTS

In order to understand the full potential of the ADCs used in this project, a complete understanding of the Nyquist sampling theorem and ADC characteristics is required. This chapter is an overview of these fundamentals.

2-1 Nyquist Sampling Theorem Review

The Nyquist sampling theorem defines the frequency domain limits for a sampled signal versus the sampling frequency. The most common interpretation of the theorem is that the sample frequency must be greater than twice the highest analog frequency to be digitized. Once the sample rate has been defined, the frequency spectrum can then be broken into "Nyquist zones" where each zone has a bandwidth equal to half the sample frequency. This is shown graphically in Figure 2-1 with an arbitrary signal present in the first Nyquist zone. By the previous definition, any signal falling into the first Nyquist zone can be precisely reconstructed.

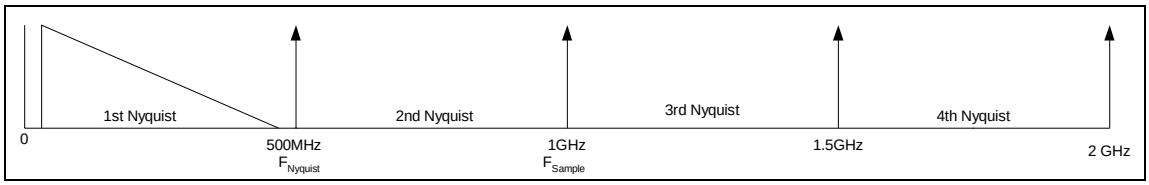


Figure 2-1 Frequency Domain of Traditional Sampling

If the bandwidth of the analog signal of Figure 2-1 is allowed to increase into the second Nyquist zone, the signal cannot be reconstructed. This effect is called aliasing and is shown graphically in Figure 2-2. As seen in the figure, any signals present in the second Nyquist zone will "fold backwards" into the first Nyquist zone and add to the signals already present. A time domain example of aliasing is shown in Figure 2-3. In

that figure, a 490-MHz sine wave and a 510-MHz sine wave produce identical samples when the Nyquist frequency is 500 MHz. The figure clearly shows that once an analog signal is digitized, it is impossible to determine which Nyquist zone it came from. In practice, aliasing can be eliminated by attenuating all frequencies outside the first Nyquist zone. The filter must attenuate unwanted signals until their amplitude is below the noise floor of the ADC [2]. This is called an antialiasing filter and is present in most real-world acquisition systems.

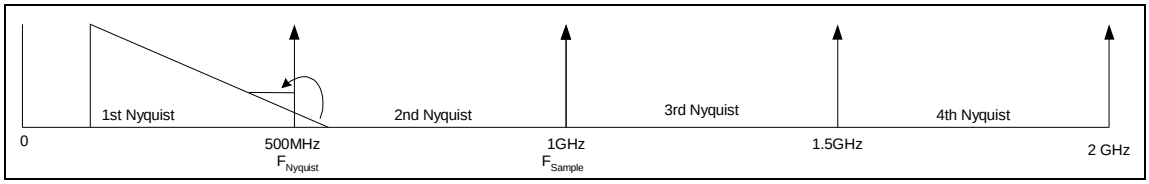


Figure 2-2 Aliasing Example (Frequency Domain)

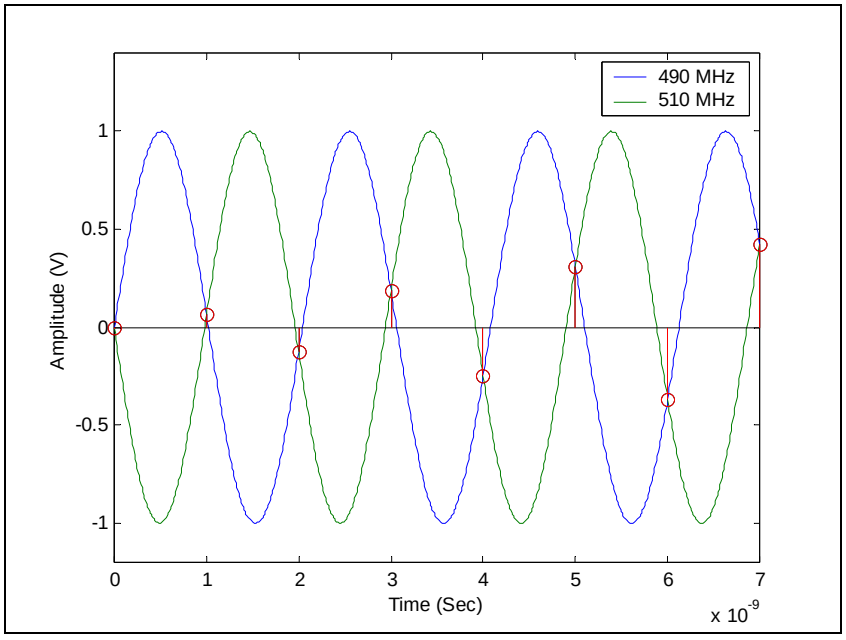


Figure 2-3 Aliasing Example (Time Domain)

The previous definition of the Nyquist theorem is only a partial application of the theorem. A broader definition would be that in order to perfectly reconstruct an analog signal, the following three conditions must be met:

1. The sampling rate must be at least twice the bandwidth of the analog signal.
2. The analog signal must lie entirely within a single Nyquist zone.
3. The original Nyquist zone must be known during reconstruction.

By this definition, a signal does not have to be in the first Nyquist zone. For example, if a signal lies entirely within the second Nyquist zone, the signal's spectrum will "fold back" into the first Nyquist zone after sampling but all necessary information is retained. This is shown graphically in Figure 2-4. Note from the figure, that any Nyquist zone may be used although even zones produce a "frequency folding" effect where odd Nyquist zones have only a "frequency shifting" effect. This "frequency shifting" effect is equivalent to downconversion and is highly desirable in many radar and radio applications. This type of sampling outside the first Nyquist zone is called "undersampling" or "bandpass sampling." When it is used for a downconversion application it is called "direct downconversion" [2].

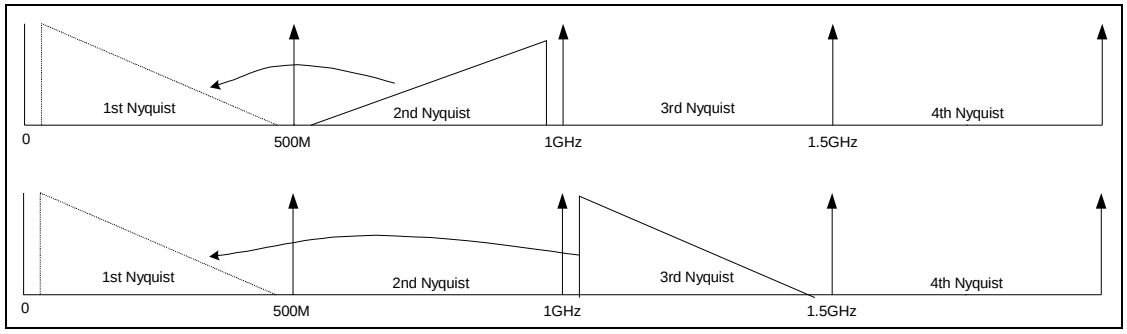


Figure 2-4 Undersampling from Even and Odd Nyquist Zones

Special ADCs must be designed for undersampling. Ordinary ADCs are designed strictly to operate in the first Nyquist zone. The frequency response of the analog input section of the ADC normally rolls off rapidly above the Nyquist frequency to reduce the

requirements of the antialiasing filter. Because of this, undersampling even in the second Nyquist zone is not possible. Special ADCs with an analog bandwidth that extend into multiple Nyquist zones are necessary for undersampling applications. The MAX104 ADCs chosen for this project have a Nyquist frequency of 500 MHz with an analog bandwidth of 2.2 GHz thus enabling undersampling into the 5^{th} Nyquist zone.

Assuming direct downconversion is not desired, additional post-processing is necessary to recreate a signal after undersampling. Following the previous example (from Figure 2-4), with the knowledge that this signal came from the second Nyquist zone, reconstruction is a two-step process. First, an interpolation filter must be applied to double the sample frequency. Next, the signal is moved back to its original frequency band by either folding across 500 MHz in the frequency domain or by multiplying with a 500-MHz sine wave in the time domain.

2-2 Limitations and Characteristics of Real Analog-to-Digital Converters

The first and most obvious limitation of a real ADC is quantization. Real ADCs can be purchased with output resolutions ranging typically from 6 to 16 bits. The effect of quantization is to raise the noise floor in the digital domain thus increasing the amplitude of the smallest signal that can be discerned. The performance measure used to characterize this is the signal-to-noise ratio (SNR). The SNR of an ideal ADC is calculated using the following formula [2]:

$$SNR = 6.02B + 1.76 + 10\log_{10}\left(\frac{f_s}{2f_{\max}}\right) \quad (1)$$

where,

B = number of converter output bits, no units
f_s = sample frequency, Hz
f_{max} = maximum frequency of a lowpass input, Hz

Ignoring the frequency dependent term for now (discussed below), the SNR is directly proportional to the number of output bits. Using the equation, an ideal 8-bit ADC has a maximum SNR of 49.9 dB. This SNR is only achieved with a full scale input. For amplitudes less than full scale, the quantization step size becomes a larger percent of the signal and the SNR is degraded. In practice, real ADCs will approach this value but fall short by 2 to 5 dB due to analog limitations, thermal noise, and non-linearity in the conversion process.

Real ADCs are sometimes rated using the effective number of bits (ENOB). This is the number of bits of an ideal ADC (found using the equation 1) necessary to produce the SNR of the real ADC. For example, the MAX104 8-bit ADC achieves an SNR of 47.4 dB which is an ENOB of 7.74 bits.

The frequency dependent term of (1) applies to one special case and actually allows the ENOB to increase beyond the normal rating of the converter. This term applies for the case of oversampling (when the sample frequency (bandwidth) is much greater than the minimum frequency specified by the Nyquist theorem). Oversampling improves performance because the quantization noise has a fixed amount of power. As the sample rate is increased, the quantization noise is spread over a larger bandwidth. By applying digital filtering that passes only the desired signal, some of the quantization noise power is removed. Without this special processing, the SNR does not improve because the full quantization noise power is still present in the sampled waveform. With this special technique, an 8-bit ADC, sampling a 100 kHz signal at 20 MHz can provide an SNR of 68 dB resulting in an ENOB of 11 bits [2].

One additional rating of a real ADC is the spurious free dynamic range (SFDR). The SFDR is measured by inputting a very pure sinusoid into the ADC an observing the fast Fourier transform (FFT) of the output data. The FFT will contain a single large impulse representing the sine wave, but will also contain smaller impulses that extend above the noise floor. These smaller impulses are generally due to non-linearities in the conversion process [2]. The SFDR is then the difference (in dB) between the sine wave and the largest spurious component. An example of SFDR can be seen graphically at the bottom of Figure 2-6. For wideband applications, the SFDR will limit the difference between the largest and smallest signals that can be received because it is difficult to discern between a tiny signal and the spurious component of a large signal.

2-3 Interleaving of Analog-to-Digital Converters

For ADCs with analog bandwidth extending into multiple Nyquist zones, interleaving of multiple converters can be used as an alternative to undersampling of high frequency signals. With interleaving, the analog waveform is sampled by multiple converters, each with a unique sampling phase, and the converter digital output streams are then combined together to effectively increasing the sample rate. Examples of this are two converters sampling at 0° and 180° to double the sample rate or four converters sampling at 0°, 90°, 180° and 270° to quadruple the sample rate. A time domain representation of two converter interleaving is shown in Figure 2-5.

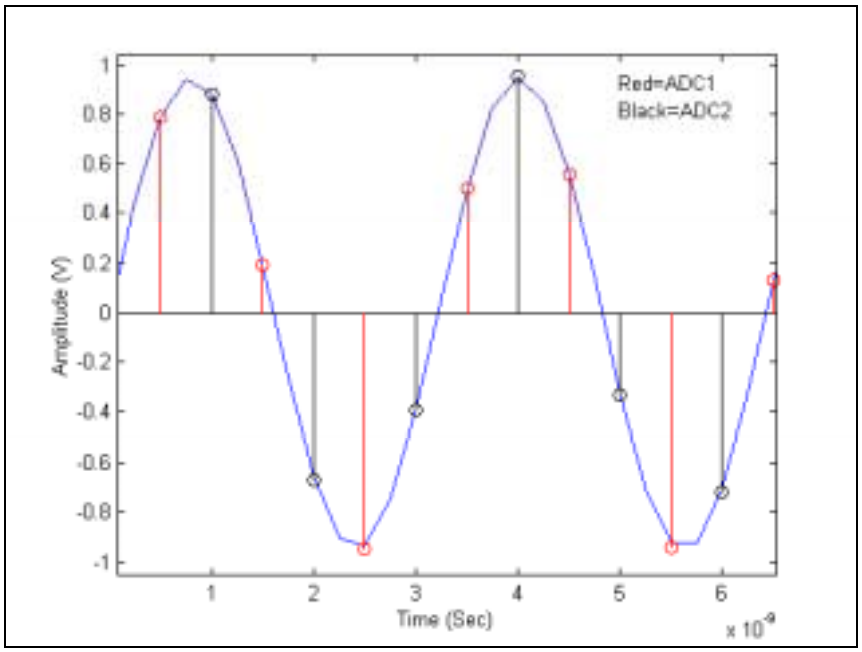


Figure 2-5 Time Domain Example of Two Converter Interleaving

Theoretically, interleaving should produce results identical to a single high speed converter. Based on the high-speed characteristics of real ADCs it would seem interleaving would produce better results than a single converter since the interleaved converters operate at lower individual speeds. In practice, interleaving introduces spurious components because the two channels cannot be precisely matched in amplitude or phase. As the frequency of the input waveform and sample clock increases, the timing requirements to prevent these spurious components become evermore stringent. Figure 2-6 shows the output of a Matlab simulation illustrating the effects of even a slight phase offset when interleaving with converters operating at 1 GSPS with an input frequency of 310 MHz.

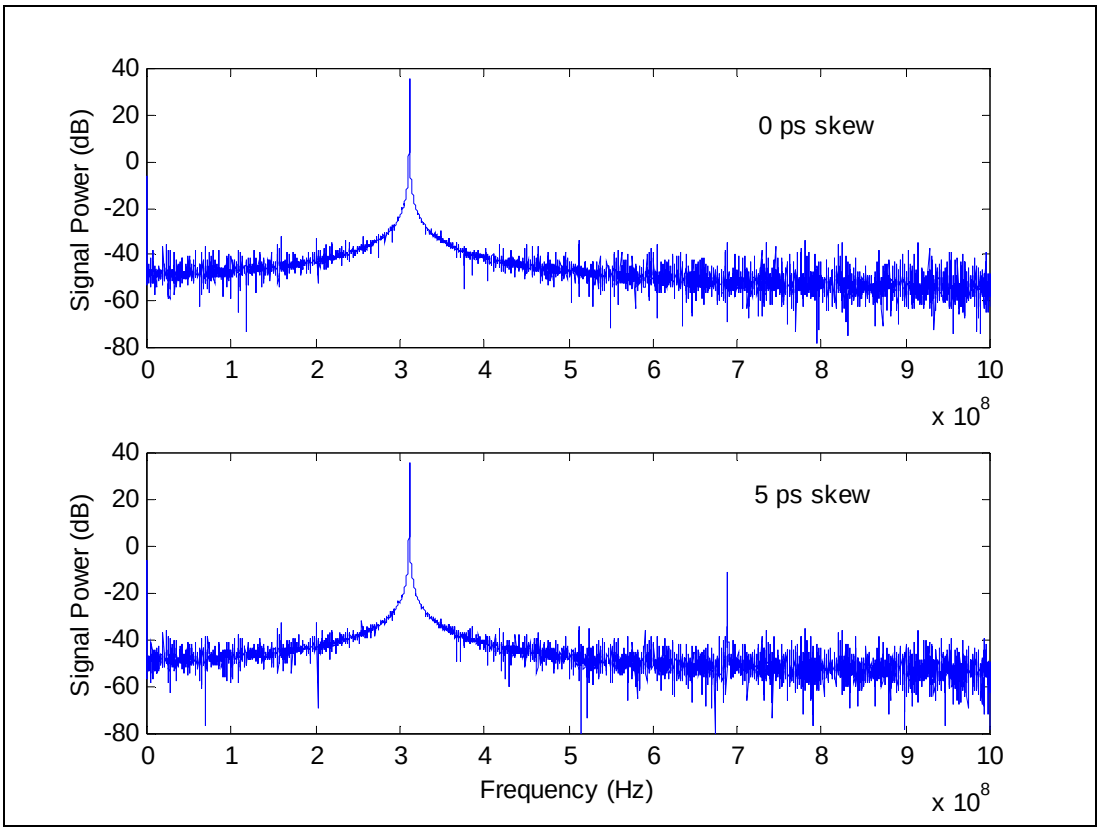


Figure 2-6 Spurious Components due to Mismatched Interleaving

As seen in the figure, an SFDR of 70 dB is achieved with perfect 180° timing. With the interleaved clock skewed by only 5 ps (1.8°) from the ideal value, the SFDR is reduced to 46 dB as a spurious component is mirrored across the Nyquist frequency (500 MHz). Using the same simulation it can be shown that a 1% difference in amplitude gain of the ADC's has a nearly identical effect. Surprisingly, the simulation shows that having a DC offset of a few LSBs on one channel does not create spurious components and only adds to the DC value of the interleaved signal (which can be easily removed during post processing).

In practice, it may be possible to achieve less than 5 ps clock skew through calibration but it would be extremely difficult to hold that calibration for any meaningful amount of time. Therefore, interleaving at very high speeds with the intent of increasing

the digitized bandwidth is not practical without sacrificing SFDR or applying special post processing to compensate for non-uniform sampling (algorithms for this are discussed in [3] and [4]). If the input signal is confined within a Nyquist zone, all spurious components can be filtered out during post processing and thus the side effects of interleaving are eliminated.

If the signal lies inside of a Nyquist zone, it is not necessary to interleave but interleaving does have three advantages in that situation. First, post-processing is simplified versus undersampling, requiring only a bandpass filter instead of interpolation and shifting in the frequency domain. Second, the SNR is improved by around 3 dB (for two converter interleaving) since half of the quantization noise will be filtered out by the bandpass filter operation [2]. Third, the system is more reliable since redundant data are being recorded. If one of the converters fails during acquisition, the data can still be recovered by treating the remaining converter stream as an undersampled signal.

2-4 Coherent Averaging of Radar Signals

The dynamic range of an ADC is significantly improved if a measurement can be taken multiple times in the presence of noise. The basic theory behind coherent averaging is that if a signal is repetitive and the ADC is coherent to that signal, it can be sampled at exactly the same locations for each repeated measurement. The repeated measurements will then be identical except for random noise. This is shown graphically in Figure 2-7. From the figure, it is seen that the ensemble average (of a given sample number) of a coherent (noisy) waveform is equivalent to sampling a DC voltage in the presence of noise.

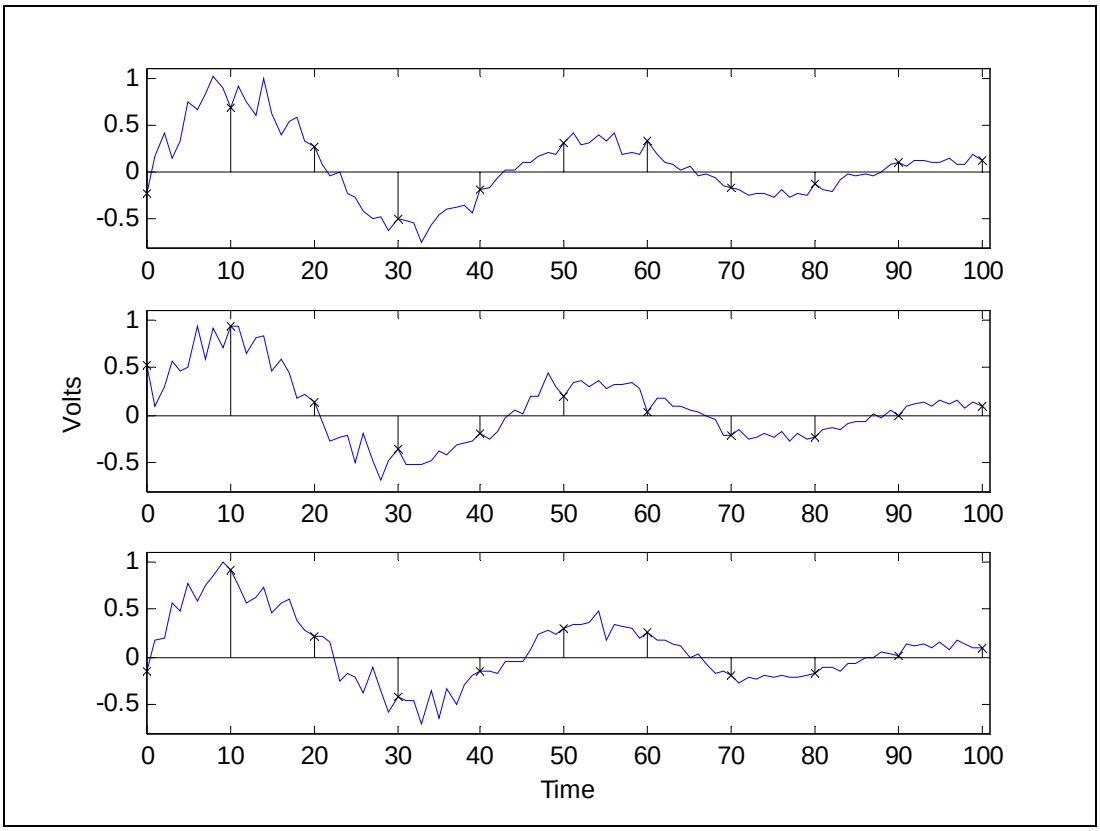


Figure 2-7 Coherent Sampling of Repetitive (Radar) Signals

The precision gain due to averaging is illustrated in the following example. Assume a DC voltage is input into an ideal 8-bit ADC with a voltage such that the quantization level would be 131.576839. In the absence of noise, this would quantize to level 132 for every measurement taken, thus there is nothing to be gained by averaging without noise. However, if ½ LSB of uniformly distributed noise is present in addition to the DC voltage, the converter will statistically output both 131 and 132. The histogram of these two values would favor 132 slightly since the DC voltage is slightly higher than 131.5 (the decision threshold). The expected histogram values for a varying number of samples is shown in Table 2-1.

	Histogram					Histogram		
Averages	131	132	Average Value		Averages	131	132	Average Value
10	4	6	131.6		2	1	1	131.5
100	42	58	131.58		4	2	2	131.5
1000	423	577	131.577		8	3	5	131.625
10000	4232	5768	131.5768		16	7	9	131.5625

Table 2-1 Histogram and Averaging for a Input Level of 131.576839

From the right hand side of the table, the following formula can be derived for estimating the effective number of bits with averaging ($ENOB_A$):

$$ENOB_A = B + \log_2(N) \tag{2}$$

where,

B = number of converter output bits
N = number of averages

It is important to note that this relationship is only valid for a large number of averages as the noise will not be sufficiently uniform over a small number of samples. Additionally the equation is setting an upper bound on ENOB gain as a real system will not have exactly ½ LSB of uniform noise. A real system will typically have a few LSBs of Gaussian noise resulting in an ENOB gain less than that of the ideal case. A simple way to find $ENOB_A$ experimentally is to measure the SNR of a real signal with a known noise distribution and convert SNR into ENOB using equation 1.

CHAPTER 3 DATA ACQUISITION SYSTEM DESCRIPTION

3-1 System Overview

The data acquisition system consists of four modules: A/D Converter Card, Averaging Card, Multiplexer (MUX) Card, and Computer Interface Card. The configuration of these modules is shown graphically in Figure 3-1. Each A/D converter card contains a gigasample A/D integrated circuit (IC) that is the enabling technology of the entire system. Since the output signals of the converter change too rapidly for current memory/processing technologies, the A/D converter card also performs a 1:16 de-multiplexing to widen the bus and slow the clock rates. The Averaging Card performs averaging on the coherent data using four digital signal processors (DSPs) and also multiplexes the data into the correct order after processing. Finally, data from the Averaging Card pass through a multiplexer card which switches access from the four acquisition channels to a single digital input/output (DIO) card on the host computer.

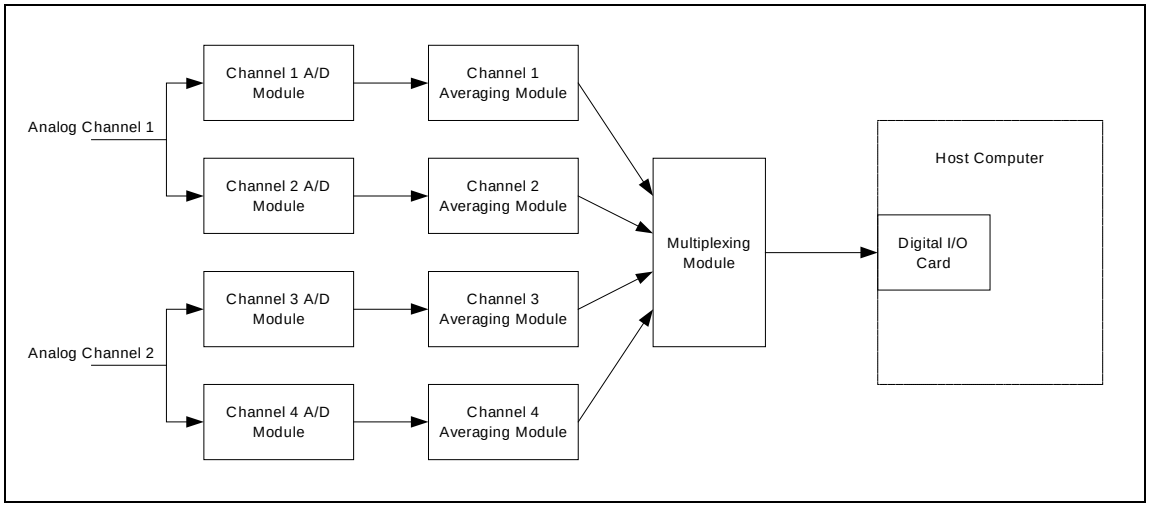


Figure 3-1 Overview of the Data Acquisition System

In Figure 3-1 it can be seen that the system actually consists of four separate A/D channels. Using independent analog inputs the system can be operated with 4 independent channels at 1-GSPS each. By placing a power divider at the inputs of channels 1-2 and 3-4 as shown in the figure and providing 180° clock timing to the converters a 2 channel, 2-GSPS (interleaved) system is realized. This demonstrates the easy reconfigurability which is a major advantage of a modular system.

3-2 Analog to Digital Converter Card

Overview

An overview of the analog to digital converter (ADC) card is shown in Figure 3-2. The 8-bit ADC is clocked using a 1-GHz sine wave from the timing system. The analog waveform is sampled upon every rising edge of this clock and the ADC divides the clock and bus frequency by two using a built in 1:2 demultiplexer. The 16-bit output from the ADC changes at a 500 MHz rate and has PECL voltage levels. These high speed signals are input directly into four 4:32 demultiplexers (demuxes) synchronized as a 16:128 demux. This serves to further reduce the bus rate to 62.5 MHz where it is output through a 272 pin connector to the Averaging Card. A synchronization signal is needed from the timing system to synchronize the demux structures so that the first sample after the PRF pulse will always be present in the same demux slot. Timing on this signal is critical as it must be consistently timed with the 1-GHz sine wave. Without this signal, coherency of the system would be lost as the first sample after the PRF pulse would be in a random demux channel.

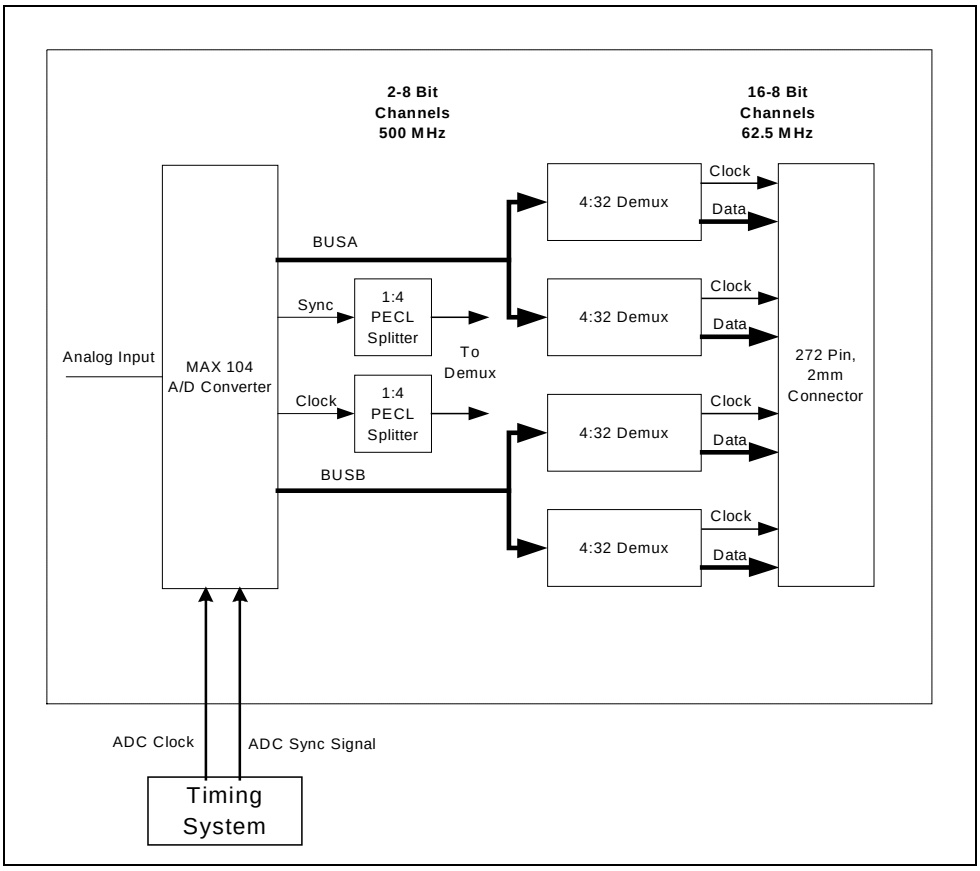


Figure 3-2 A/D Converter Card Overview

Detailed Description

Board Physical Properties

Special low loss board material is necessary for the ADC card due to the high frequencies that the analog signals may operate at (up to 2-GHz). Although the trace length is only one inch, the attenuation due to dielectric losses increases with frequency. For precise measurement of wideband signals this loss must either be minimized or compensated for digitally (in post processing). A comparison of board loss vs. frequency for a 50 Ω microstrip line (8 mil dielectric thickness) for two board materials is shown in Figure 3-3. As seen in the figure, the loss from one inch of standard FR-4 would vary from 0.05 dB to over 0.2 dB in the frequencies of interest where 0.2 dB represents about 2 least significant bits (LSBs) of the ADC with a full scale input voltage. For the same

configuration, the Rogers 4403 varies from 0.02 to 0.065 over the same range. This represents less than one LSB of error and can therefore be neglected. Additionally, if the ADCs were ever used up to the full 2.2-GHz input bandwidth, the loss difference between the two materials would become even more prominent.

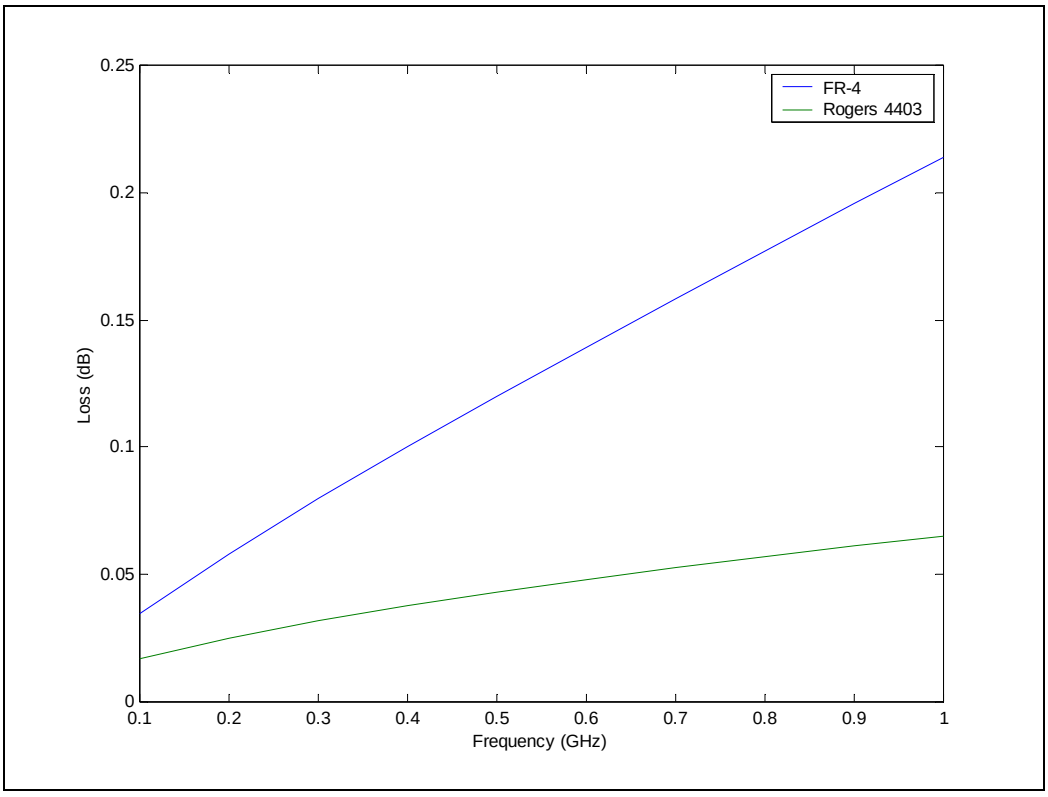


Figure 3-3 Tangential Loss vs. Frequency for a One Inch Trace

An additional advantage of using low loss (RF) material is the ability to control line impedances more closely. When line impedances are closer to the ideal value, standing waves are reduced resulting in more accurate analog measurements over a wide frequency band. For digital signals, the tighter tolerance line impedance results in smaller reflections, lowering the probability of a bit error or false trigger.

A stackup of the circuit board is shown in Figure 3-4. The internal planes are not dedicated to any particular power supply and are highly segmented due to the multiple

power sources of the mixed signal IC as seen in the board layout of APPENDIX F. The general layout of the board is modeled after the manufacturers recommended layout used in the evaluation board of the MAX104 ADC [6].

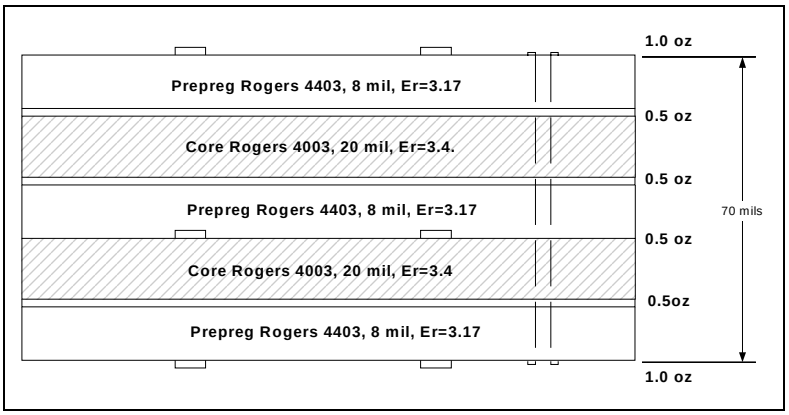


Figure 3-4 ADC Circuit Board Stackup

Since the logic levels of the ADC outputs are all differential PECL, a special coupled microstrip transmission line structure was used to reduce board space. The rule of thumb for digital transmission lines is to allow a spacing greater than three times the height of the dielectric [7]. For the board stackup of Figure 3-4, the line spacing would therefore be 24 mils. The 50 Ω line width is 18 mils resulting in a minimum width of 60 mils for two microstrip traces side by side. By taking advantage of the differential signaling, a coupled transmission line structure was used which acts as an RF coupler in the odd mode. The 50 Ω dimensions for this were found using the Linecalc [8] software tool and were determined to be 11 mil line widths with an 8 mil gap thereby using half the board area of regular microstrip. Since this board operates at such high frequencies, the PCB manufacturers undercut factor of 1.25 mils was added onto all high speed lines to make them as close the ideal value as possible after processing. A drawing of the two

high-speed microstrip configurations (target values after processing) is shown in Figure 3-5.

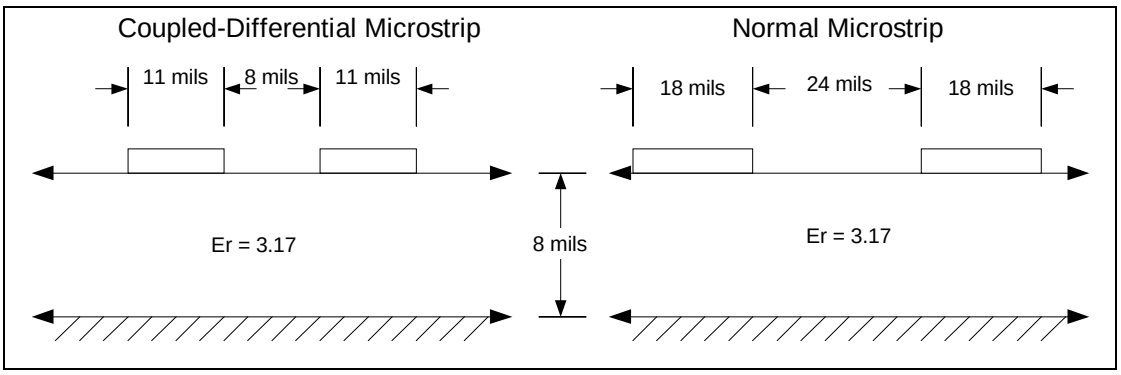


Figure 3-5 Microstrip Structures Used on High Speed Lines

For low speed (62-MHz) TTL signals the main goal was to minimize the board area while maintaining some impedance control. To minimize board area the manufacturer's minimum standard line width of 8 mils was selected (6.75 mils finished width). This results in an impedance of about 80 Ω for the microstrip and about 60 Ω for the offset stripline on layer 4. Although there is a slight impedance mismatch between these lines, Advanced Design System (ADS) simulations (and later prototype verification) showed that this was insignificant when the lines were properly terminated.

Circuit Design

Analysis of the circuit design should begin with the MAX104 ADC shown in Figure 3-6. A 1-GHz sine wave clock signal is input into transformer T1 which has a center tapped output to produce 0° and 180° (-3 dB) replicas of the GHz clock. These differential sine waves then drive the differential clock inputs on the MAX104. The MAX104 has internal 50 Ω laser trimmed termination resistors on the analog and clock inputs, eliminating the need for external terminations. On the rising edge of the true

clock input, the ADC's on board 2.2-GHz track and hold (T/H) amplifier latches the voltage at the analog input and the internal flash converter digitizes that value.

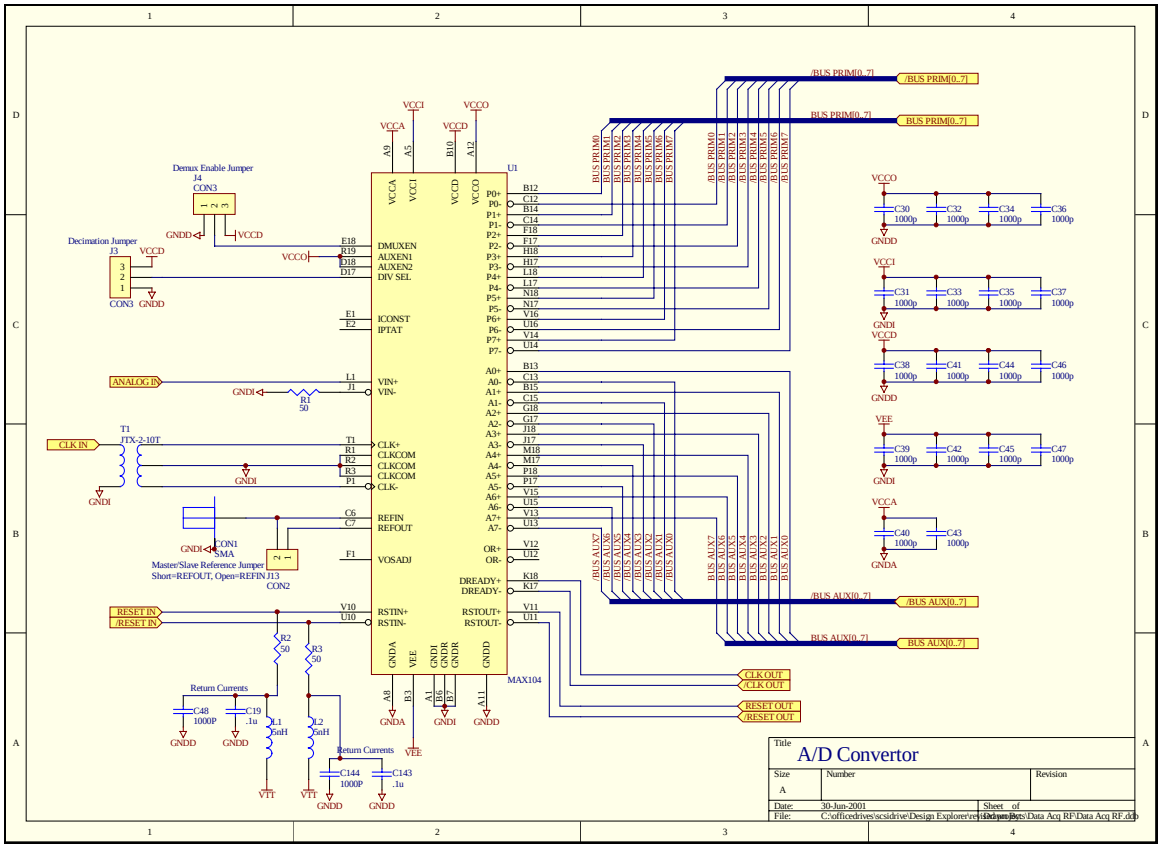


Figure 3-6 Schematic Layout of the MAX104 ADC

The acquired sample is output from the MAX104 via complementary PECL signaling that can operate at speeds equal to the input clock frequency. However, to simplify integration into surrounding circuitry the chip includes an on-board 1:2 demultiplexer to widen the output bus and slow the output bus speed from 1 GHz to 500 MHz. These outputs are shown in Figure 3-6 as buses P[0:7] (primary) and A[0:7] (auxiliary). Jumper J4 was added so the output multiplexer could be enabled or disabled. This was intended for debugging purposes as the downstream hardware is not capable of operating above 625 MHz.

When the demultiplexer is enabled there are two output paths that a particular sample may take. This would result in a potential loss of coherency in the radar because the same sample must be output in exactly the same path after each PRF pulse. This is complicated further because the ADC uses a pipelined architecture where the data are delayed by 6-8 clock cycles from acquisition to output. To resolve this issue, the MAX104 has a synchronization input (reset) that is used to eliminate this ambiguity. By asserting a copy of the PRF pulse onto this synchronization input, the same samples are guaranteed to be on the same ADC output ports 12 clock cycles after the sync pulse is sampled high. Because there are two possibilities for the arrival of the sync pulse (demux in phase and demux out of phase) there are two possible timing diagrams shown in Figure 3-7 and Figure 3-8.

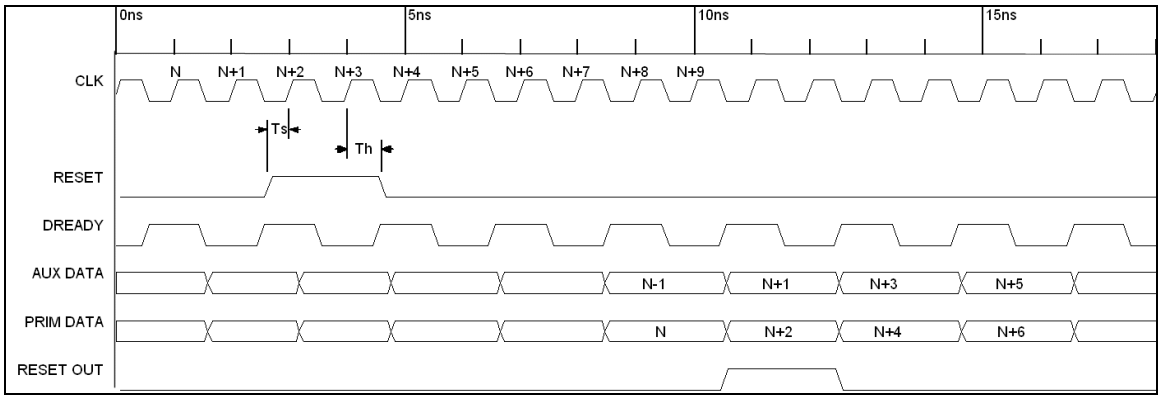


Figure 3-7 MAX104 Timing Diagram (Demux in Phase with Sync) [5]

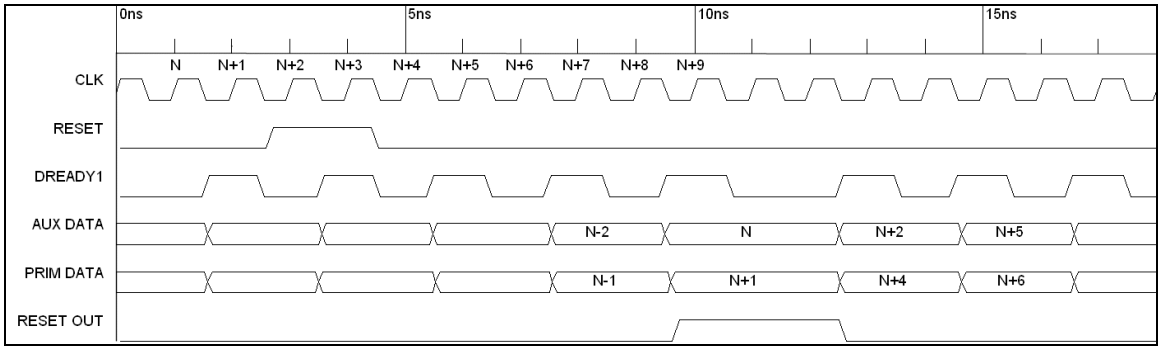


Figure 3-8 MAX104 Timing Diagram (Demux out of Phase with Sync) [5]

For the case where *RESET* and *DREADY* are both sampled high, the output is considered in phase and no change in the flow of data is observed. For the case where *DREADY* is sampled low and *RESET* is sampled high, the immediate samples from the demux are considered out of phase. This is corrected with the converter skipping one clock cycle and losing one sample. As seen in the figures, the ADC will guarantee that the same samples (N+5 and N+6) are in the primary and secondary ports 12 clock cycles after *RESET* is sampled high. It is this synchronization feature of the demultiplexer that enables the system to maintain coherency. Note that using this type of synchronization places requirements on the timing system to consistently meet the setup and hold times of the *RESET* signal with respect to the 1-GHz sample clock.

The 2:1 demultiplexed output from the MAX104 is then output into four GD16333 4:32 demultiplexers as shown in Figure 3-2. This results in an overall demultiplexing ratio of 1:16, reducing the clock rate from 1 GHz to 62.5 MHz while increasing the bus width from 8 bits to 128 bits. Since additional demultiplexers are used, the synchronization output from the MAX104 is repeated to each of the demultiplexers to maintain coherency as described above. A 1:4 PECL fanout buffer is used to fan the clock and sync signals from the MAX104 to the GD16333s. Figure 3-9 shows the schematic layout of one GD16333. As seen in the layout, all input signals are differential PECL pairs terminated through a high quality 50 Ω resistors to +3.0 V. The GD16333 inputs the 500-MHz clock and outputs a 62.5-MHz clock that is routed directly to the output connector for use by the Averaging Card. Note that each GD16333 demultiplexes signals from the ADC primary and auxiliary ports. This is because routing of the 500-MHz signals was given priority over the natural logic flow. A timing diagram

of the entire demultiplexing structure is shown in Figure 3-10 (note the GD16333 is a negative edge triggered device).

As seen in the figure, the setup and hold times of the GD16333 inputs are easily met due to the additional 500 ps propagation delay of the fanout buffers. The parameter, $T_{pd,data}$, in the figure is rated in the manufacturer's datasheet to be 3.2 ns. After board construction, this was measured on four chips to be about 1 ns (as shown in the figure) resulting in hold time violations on the output bus. To solve this, inverters were added on the clock output of each GD16333 resulting in the parameter $T_{pd,meas}$ which places the rising clock edge nearly in the center of the eye. Also from the figure, it can be seen that the overall delay from the sync pulse to valid (coherent) output is 41 ns. This places a constraint on the timing system that acquisition cannot begin for at least 41 ns after the ADC sync pulse occurs. For added margin, it is recommended to delay acquisition for at least 60 ns.

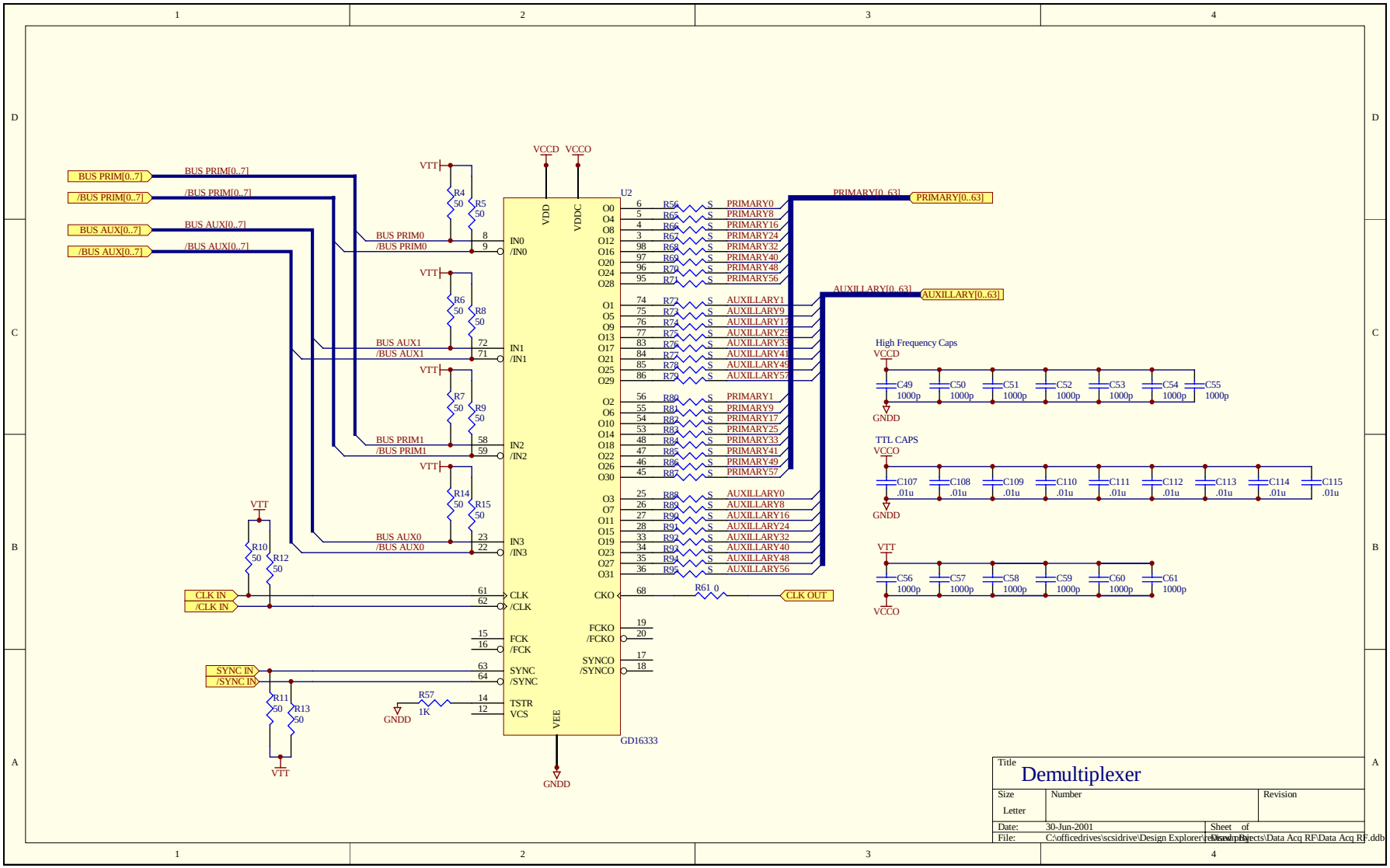


Figure 3-9 GD16333 Demultiplexer Schematics

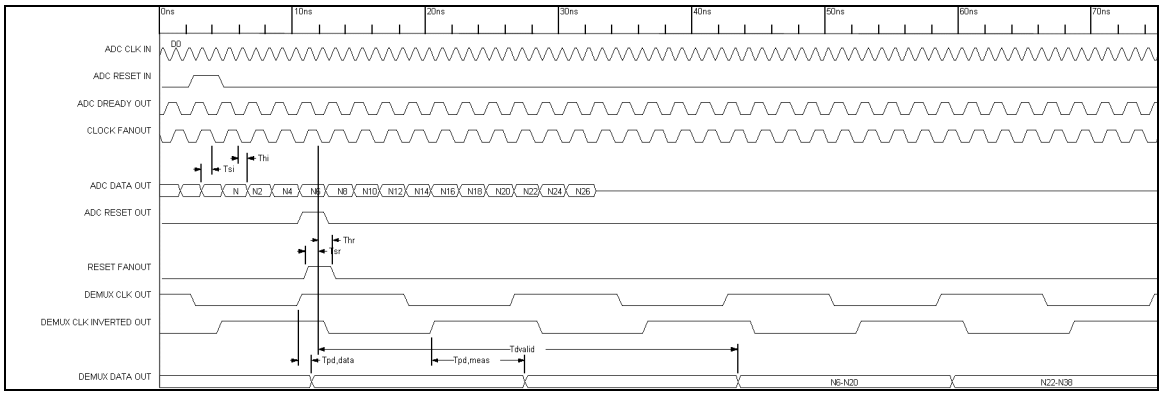


Figure 3-10 Timing Diagram of the ADC Board

A photograph of a completed board is shown in Figure 3-11 with the complete board layout shown given in APPENDIX F. As seen in the figure, a semicircular layout was used for optimum signal integrity and equal propagation delay to each demultiplexer. Because the board layout and schematics are both designed around this one goal, the low speed (62.5-MHz) data are fragmented with two bits from each of the output bytes going to each demultiplexer. To defragment the low speed data before exiting through the card connector approximately 25% of the board area was needed to route the lines from a seemingly “random” to an orderly pattern.

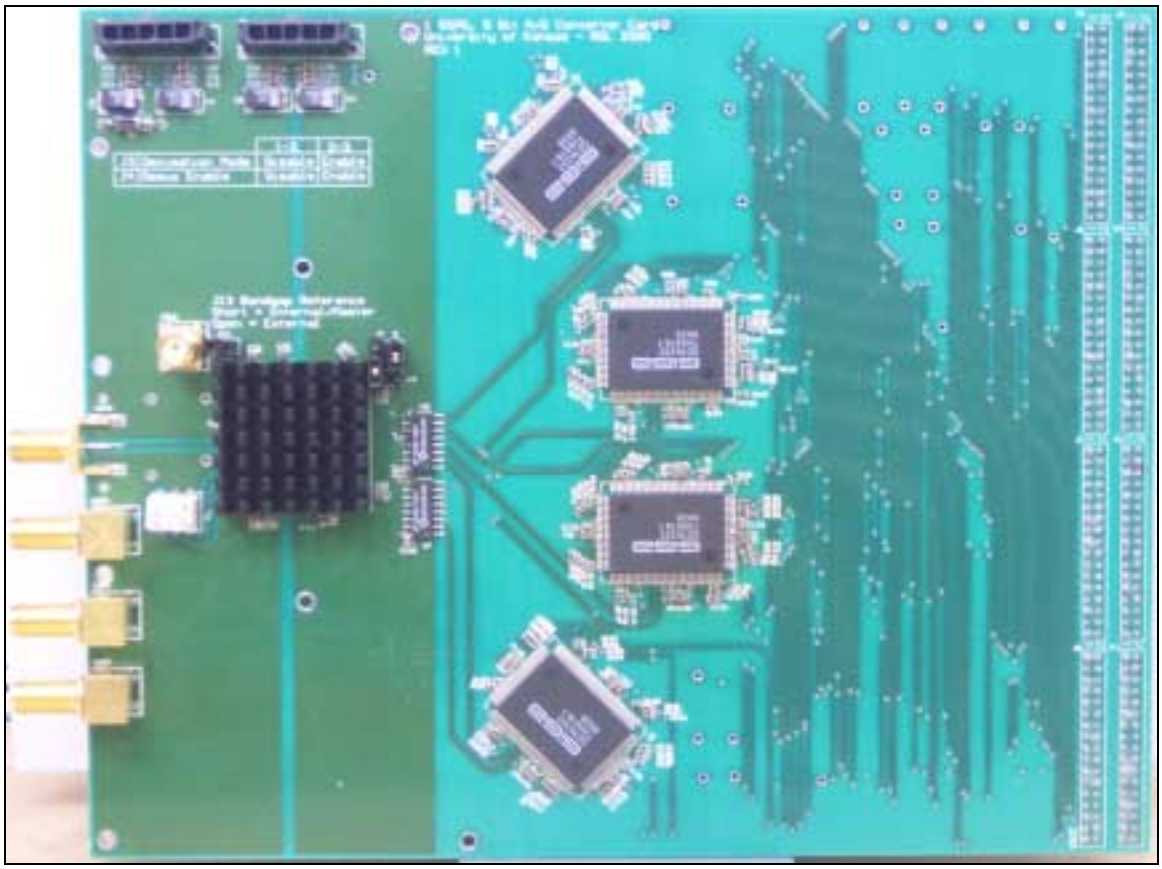


Figure 3-11 Photograph of a Completed ADC Card

Revision History/Future

This board has been revised once for a board material change. The original board material selected was the Roger 3000 series. The timing Mezzanine Card was manufactured using this material and the physical properties were found to be highly undesirable for multilayered boards. The 3000 series material is soft and pliable resulting in excessive warpage around drill holes. The 4000 series material is very similar to standard FR-4 except that it has low loss and a controlled dielectric constant making it ideal for multilayered high frequency boards.

The goal of the original design was to have the data output through the connector in chronological order with the 1st sample at the top and the 16th sample at the bottom of

the card. However, when testing the Averaging Card it was discovered that this ideal pattern was not realized due to an oversight in the design. This was easily corrected with a firmware change by re-ordering the data in the Averaging Card multiplexer. A comparison of the two patterns is shown in Table 3-1. In the future, if a completely new system is designed using similar technology, this should be corrected to simplify signal flow in the downstream hardware.

	Desired Order								
	Sample 3	Sample 4	Sample 7	Sample 8	Sample 11	Sample 12	Sample 15	Sample 16	272
1	Sample 1	Sample 2	Sample 5	Sample 6	Sample 9	Sample 10	Sample 13	Sample 14	
	Actual Order								
	Sample 6	Sample 8	Sample 14	Sample 16	Sample 5	Sample 7	Sample 13	Sample 15	272
1	Sample 2	Sample 4	Sample 10	Sample 12	Sample 1	Sample 3	Sample 9	Sample 11	

Table 3-1 Comparison of Intended and Actual Output Connector Pinout

3-3 Averaging Card

Overview

The averaging card provides a link between the ADC and computer. In addition to a physical link, it must perform the following functions:

- Perform coherent averaging on the acquired data.
- Gate the output from the ADC card so only the desired number of samples are acquired.
- Provide buffering between the computer DIO card and the ADC.
- Multiplex the 16-ADC output busses into a single bus for the computer DIO card link.

An overview of the averaging card is shown in Figure 3-12.

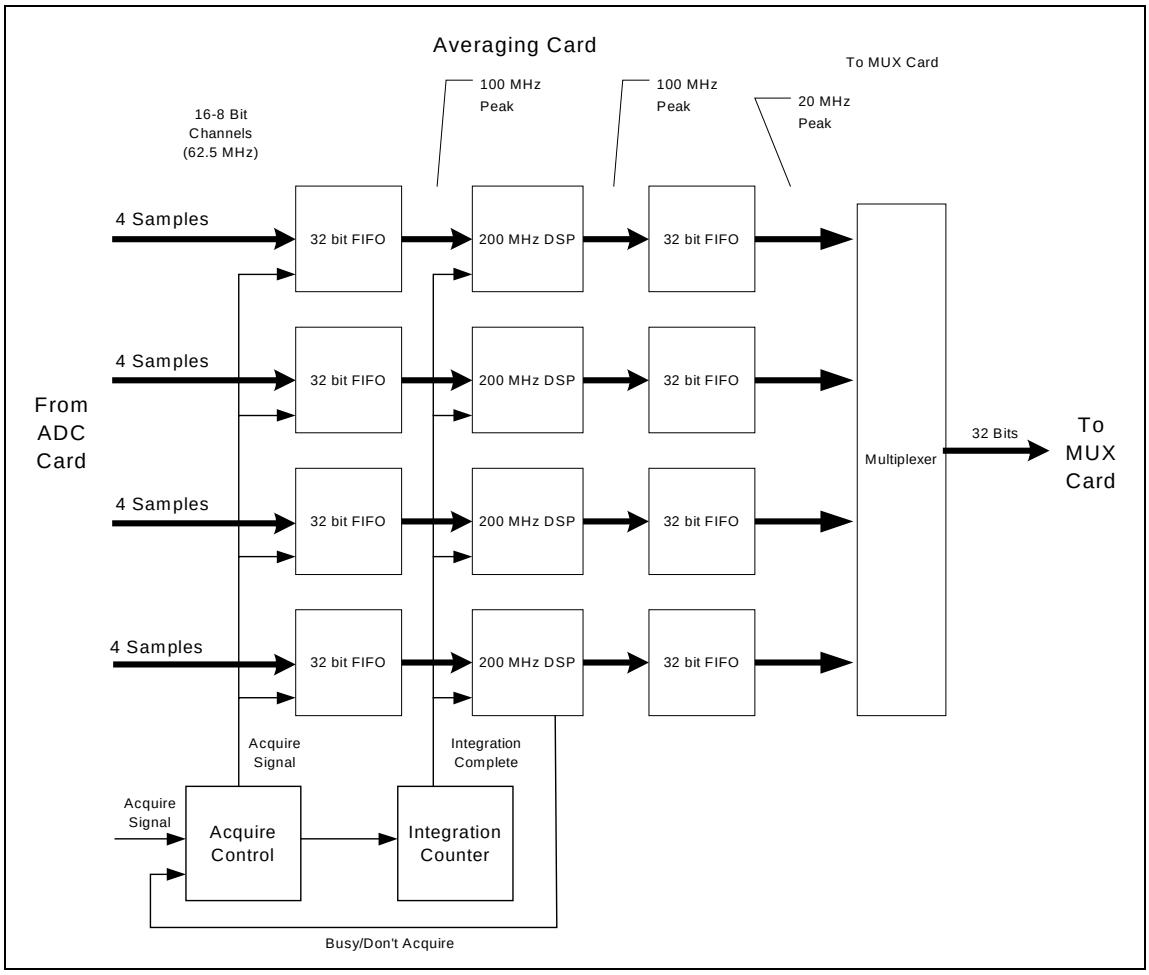


Figure 3-12 Averaging Card Overview

As seen in the figure, the 200-MHz DSPs are the focal point of the entire card. The DSPs perform coherent averaging by adding incoming samples to the previous samples stored in internal memory. Using the DSP's internal memory results in a cleaner PCB layout as no external memory chips are required. It also simplifies the design since a signal level timing analysis is not required for the internal memory interface. After the DSP has completed the required number of averages, the processed data are transferred to the output FIFOs where they are stored until the computer DIO card is ready for transfer.

A high level timing analysis for the entire board is shown in Figure 3-13. As seen in the figure, the maximum PRF of the radar can be limited by the DSPs processing rate.

The averaging card does not input a raw PRF signal, the input must be a PRF pulse with a delay and width equal to the desired sampling window. This is represented in the figure as the *Acquire* signal. For example, with a sampling interval of 1 ns, if the user wishes to acquire 8,000 samples the *Acquire* signal must have a pulse width of 8,000 ns. Figure 3-13 shows the processing time goal of the actual system for an 8,000-sample acquisition window. As seen in the figure, the processing time was estimated to be approximately 45 μs, which limits the PRI to around 50 μs (PRF = 20 kHz). This processing time could be reduced by upgrading the DSPs to operate at a higher frequency. If 300-MHz DSPs were used, the estimated maximum PRF would increase to approximately 30 kHz for an 8,000-sample acquisition window.

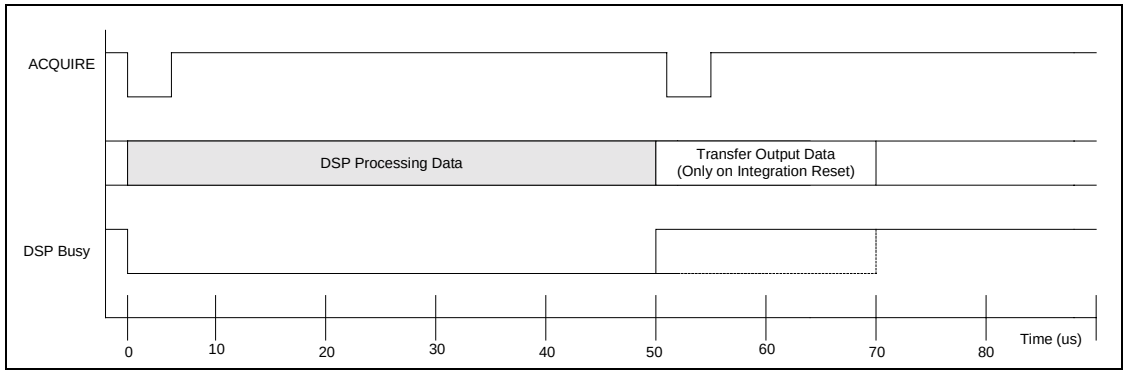


Figure 3-13 Averaging Card High Level Timing

When averaging is complete, the DSPs take approximately 20 μs to transfer the data to the output buffers and clear internal memory to start a new averaging cycle. During this time, if the *ACQUIRE* signal becomes low, it is ignored until the output transfer is complete. By implementing this "acquire blocking" feature, the PRF can be increased until it is limited only by processing time and not by the output FIFO transfer time that only occurs every 100-1000 acquisition cycles (the number of coherent

averages). If the transfer completes while the *ACQUIRE* signal is still low, the system will not acquire a partial interval as the acquire circuitry is edge triggered.

Detailed Description

A detailed discussion of the Averaging Card should begin with the top level of the schematics shown in Figure 3-14. The complete schematics are shown in 0. Initially, all FIFOs are empty and the signal processors are idle, waiting to process data. Data are input from the ADC card through a 272-pin, 2-mm connector and are presented to the input FIFOs of the four averaging card channels. Each channel receives it's own clock from the ADC card and the data and clocks are always present (they are not gated in any way). To capture data from the stream, an *ACQUIRE* signal is asserted at connector *CON1* which is input into the acquire control circuitry. The acquire control circuitry then activates the input FIFOs which capture the data.

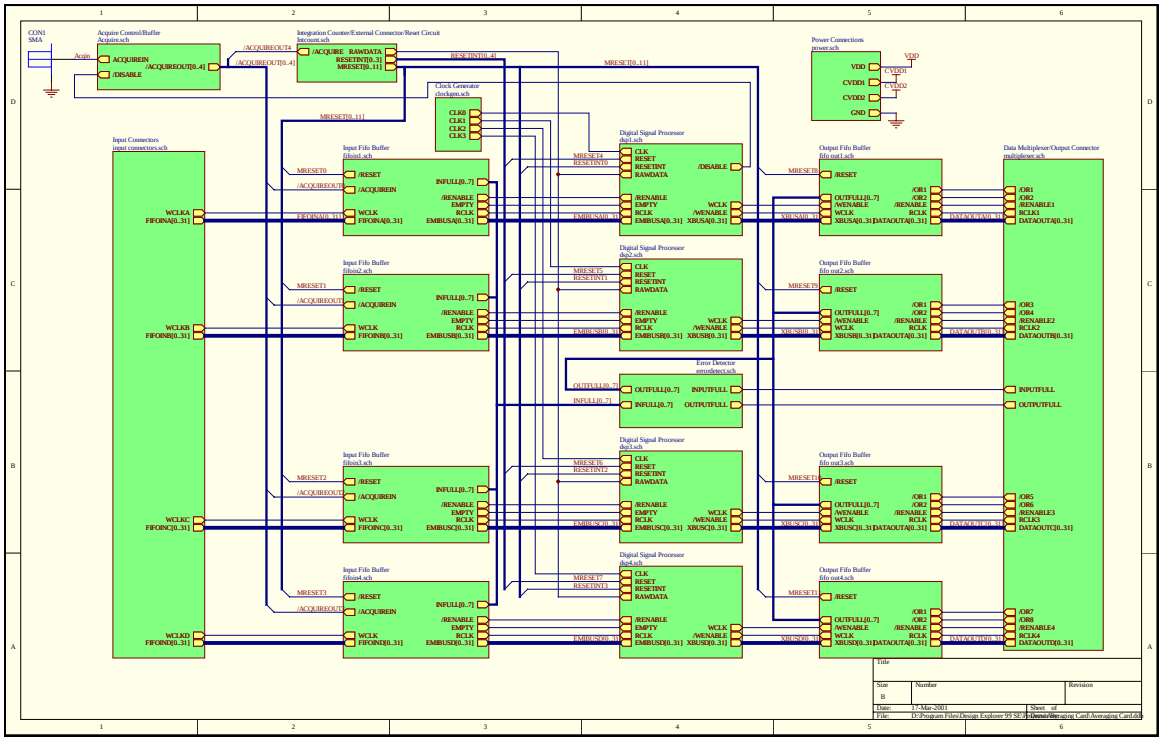


Figure 3-14 Top level Schematic of the Averaging Card

As data are input into the FIFOs, the FIFO empty flags will toggle from high to low. This triggers an interrupt in the associated DSP which then executes code to read and processed data from the FIFO. After each new sample is read, an internal count is incremented so that samples will be stored in consecutive locations in the processors internal memory. After the desired number of samples have been acquired, the *ACQUIRE* signal is de-asserted which inhibits the input FIFOs from storing new data. With no input data, the FIFO will eventually become empty as the DSP continues to read. The FIFO empty flag then triggers an interrupt in the DSP causing the DSP to cease reading new data and go into an idle state until another acquisition cycle begins.

An average counter operates in parallel to the FIFOs and DSPs. This counter is incremented for each acquisition period and counts the number of coherent integrations that have occurred. When the desired number of integrations is complete, the counter toggles the *resetint* lines to tell each DSP that, at the end of the current processing period, the data must be flushed to begin a new averaging sequence. When the input FIFOs become empty, instead of immediately going to an idle state, the processors will output the averaged data from internal memory to the output FIFOs where the data are stored until the downstream hardware is ready for transfer to the computer. As data are transferred to the output FIFOs, the FIFO empty flags are de-asserted signaling the downstream hardware that the Averaging Card has data to transfer. After all data are transferred to the output FIFOs, the DSPs clear their internal memory and go to an idle state to wait for a new acquisition period.

When the computer DIO card is ready to input data, a control line on the Averaging Card output multiplexer is toggled by the downstream hardware. The output

multiplexer will then begin transferring data from the output FIFOs to the 60 pin output connector. An additional (non obvious) function of the multiplexer is to resequence the four processing channels to the correct temporal order to ease post-processing requirements and reduce processing time for real time display on the computer monitor.

The remainder of this section will discuss the circuitry of each of the hierarchical blocks in detail.

Input Connector

The purpose of the input connector is to provide a mechanical and electrical connection between the Averaging Card and ADC card with minimal disturbance to the integrity of the high-speed digital signals. The input bus is 128 bits wide with four clock signals for a total of 132 signals that must be transmitted between the two boards. The design constraint was to allow these signals to operate at 125 MHz to support a doubling of the current 62.5-MHz rate. At these speeds signal integrity is a prime concern so a 1:1 ratio of grounds and signals was desired requiring approximately 264 total connections. Based on connector availability, cost, and physical constraints, a double row of standard 2 mm connectors was selected. A photograph of the connectors with the signal-ground pattern is shown in Figure 3-15.

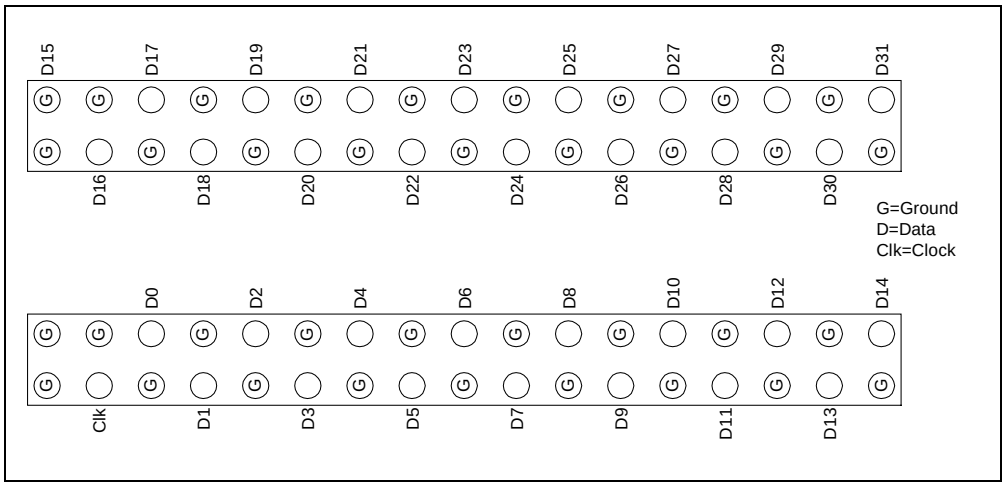


Figure 3-15 Input Connector and Pinout Pattern

The pinout of Figure 3-15 was selected for it's effective use of ground signals to reduce the possibility of crosstalk and provide a low inductance path for return currents. As seen in the figure, three ground pins surround every signal and no two signal pins are adjacent to each other. The impedance of this configuration was approximated by treating the pins as parallel wires [10]. Since the exact electrical properties of the connector's dielectric were unknown the impedance was calculated with a dielectric constant varying from 2 to 5. Using this approximation, the impedance was estimated to be around 40-60 Ω. The entire transmission line structure was simulated in EESOF as shown in the Input FIFO section below. The simulation shows that since the length of the connector is only 200 mils, the impedance can vary drastically with little effect on signal integrity and therefore a more precise impedance analysis is not necessary.

Acquire Controller

The purpose of the acquire controller is to fan out four copies of the acquire signal and to allow disabling of the acquire signal when the DSPs are transferring data to the output FIFOs. This function was implemented in a single Altera PLD. The PLD

program schematic is shown in APPENDIX K. A timing analysis of the program is shown in Figure 3-16.

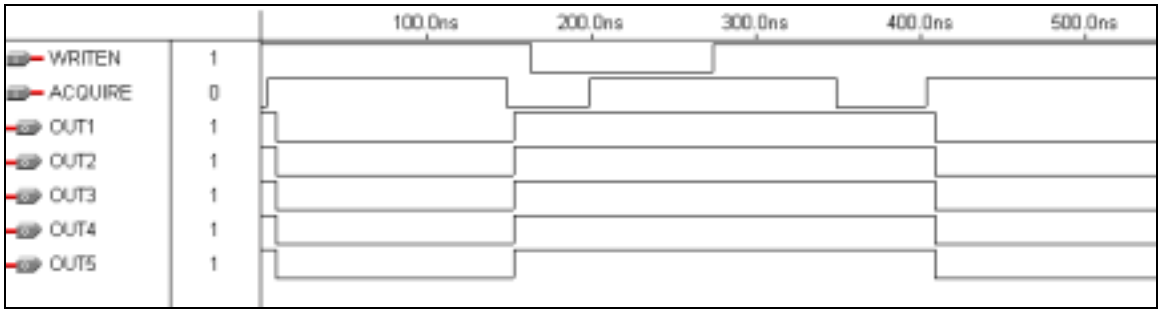


Figure 3-16 Acquire Control Timing Diagram

As seen in the timing diagram, when the *WRITEN* signal is high, the acquire signal simply passes through the device to the five outputs. When the DSPs are busy transferring data to the output FIFOs, they will pull the *WRITEN* signal low and any *ACQUIRE* signals will be ignored. After the DSPs are ready for more data, the *WRITEN* signal goes high and acquisition continues. Note again that the blocking is edge sensitive so that if the *WRITEN* signal is released while *ACQUIRE* is still high (275 ns in the figure), the output does not change until the next *ACQUIRE* rising edge (400 ns in the figure).

Input FIFOs

The input FIFO structure is shown in Figure 3-17. A 32 bit FIFO was needed for each of the four DSP channels, however, due to availability constraints two 18 bit FIFOs were paired to construct a 36 bit FIFO with four unused bits. The circuit operates as described previously with the data and clock always present on the input. When the acquire signal is asserted, the write enable is asserted on the FIFO input causing it to capture the input data. After three output clock cycles [9], the output ready (*OR*) flag toggles low signifying that the FIFO has data on the output ready to be transferred. This

operating mode is called first word fall through (FWFT) [9] and it allows instant access to the first word when the DSP is ready to read. The DSP will then assert the *RENABLE* signal causing the first word to output immediately and the FIFO will increment it's memory pointer to the next location in sequence. When the FIFO becomes empty, the *OR* flag toggles high and further output accesses will return the same (final) value.

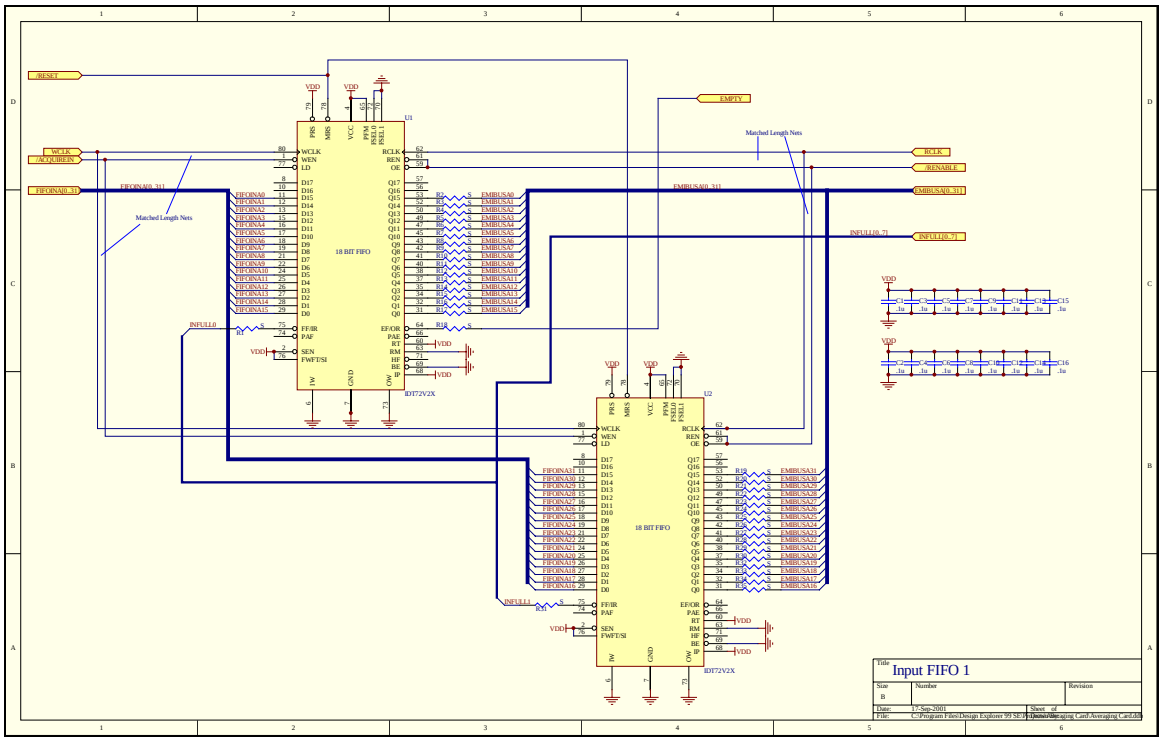


Figure 3-17 Input FIFO Schematic

The disadvantages of this "dual FIFO" method are increased PCB area and additional loading on the clock and enable signals which must drive two devices in parallel instead of a single 36-bit device. The increased board space was not a significant problem in this design as the only size constraint was the manufacturability of the PCB. However, signal integrity of the clock and enable lines was a concern since the FIFOs were designed to operate at 125 MHz. A split (matched length) transmission line

structure was incorporated to minimize reflections and avoid daisy chaining on the series terminated lines. This special structure is shown graphically in Figure 3-18.

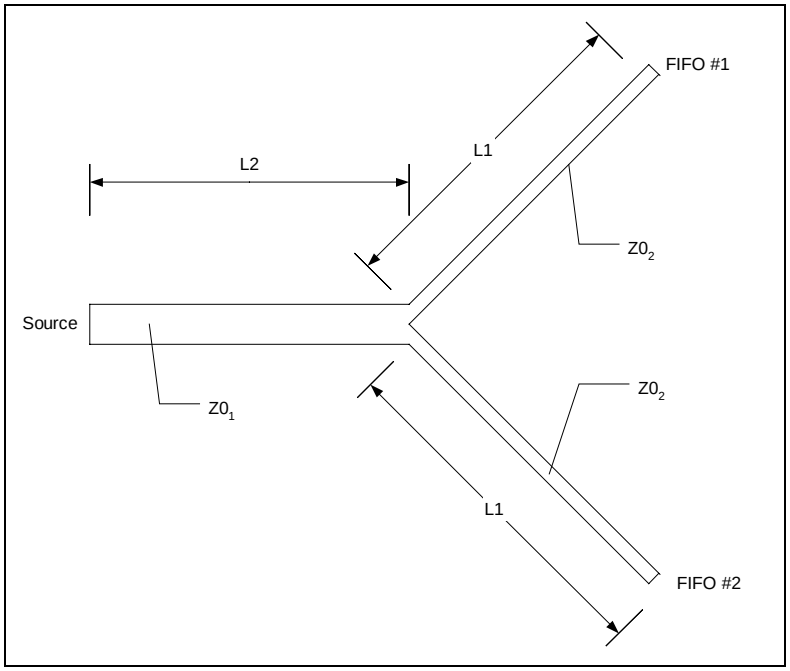


Figure 3-18 Split Transmission Line Structure for Dual FIFOs

For the structure shown in the figure, no reflection will occur at the junction if $Z0_2$ is twice $Z0_1$. Additionally, for a source terminated line, the length, *L1*, of the two splits must be exactly equal to create the ideal response. In practice, line impedances on a dense board will typically range from 60 to 80 Ω. To minimize reflections the source line impedance would therefore need to be 30 to 40 Ω for a matched condition. Since the source line impedance on the ADC card is 75 Ω, a significant mismatch is generated. An EESOF simulation model of the entire structure is shown in Figure 3-19 and Figure 3-20 and the actual eye measurement at the FIFO inputs is shown in Figure 3-21. Note that the simulation includes non identical loads (representing the two FIFO inputs) and non-identical line lengths as is the case in a real PCB.

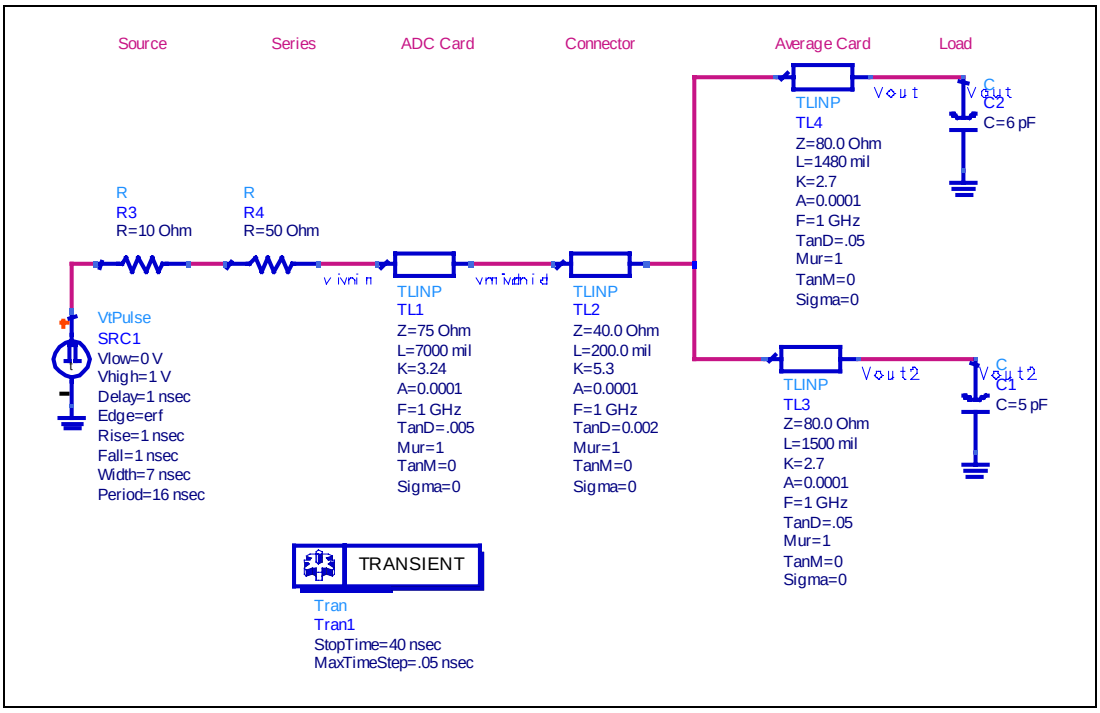


Figure 3-19 Simulation Model of Input Transmission Line Structure

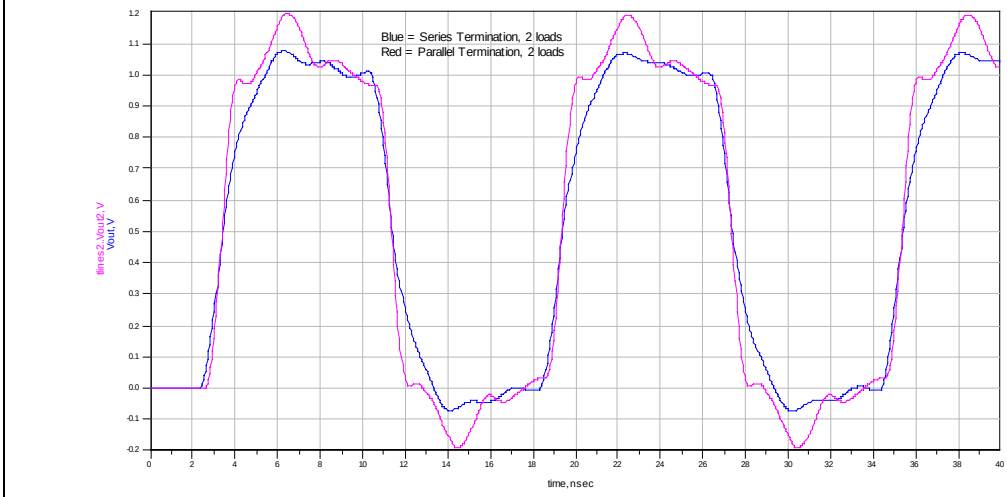


Figure 3-20 Input Transmission Line Simulation Results

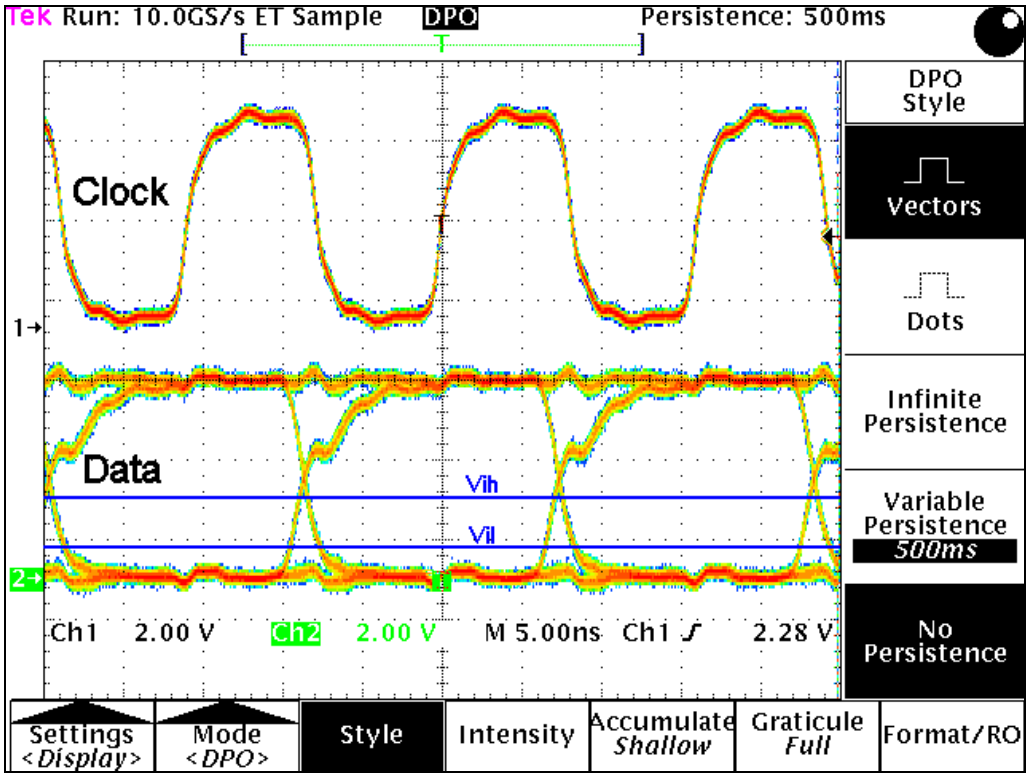


Figure 3-21 Eye Measurement at the FIFO Inputs

The simulation shows that the impedance, length, and load mismatches are acceptable since the length, *L1*, is relatively short (1.5 inches). The simulation also shows that the series termination method produces a similar but slightly smoother step response than a parallel termination. The series termination was selected for this reason and because a capacitively coupled or Thevenin termination is required for TTL parallel terminations. This would require two termination components for each line where the series termination method requires only one. Looking at the actual measurement results, the exact shape of the clock waveform is not identical to the simulated waveform but the smoothness and amplitude of the reflections are similar.

For the FIFO input data lines, the transmission line structure is more traditional with a source termination and a single load. However, viewing the waveform of Figure 3-21, the rising edge of the data signals has a significant distortion. This is due to the

differing source impedance of the data line driver for a high or low state. Again, using EESOF simulation, the line response was recreated with a source impedance of 60 Ω for a logic high and 25 Ω for a logic low. The simulation and results are shown in Figure 3-22 and Figure 3-23. The current level of distortion on the rising edge is still acceptable given the logic levels shown in the figure. If it is desired to optimize waveform distortion, a smaller value of series resistor (25-35 Ω) could be selected to compromise between rising edge and falling edge distortion. The clock signals shown above do not exhibit this non-symmetric behavior because a separate buffer with different output characteristics was used as a source on those lines.

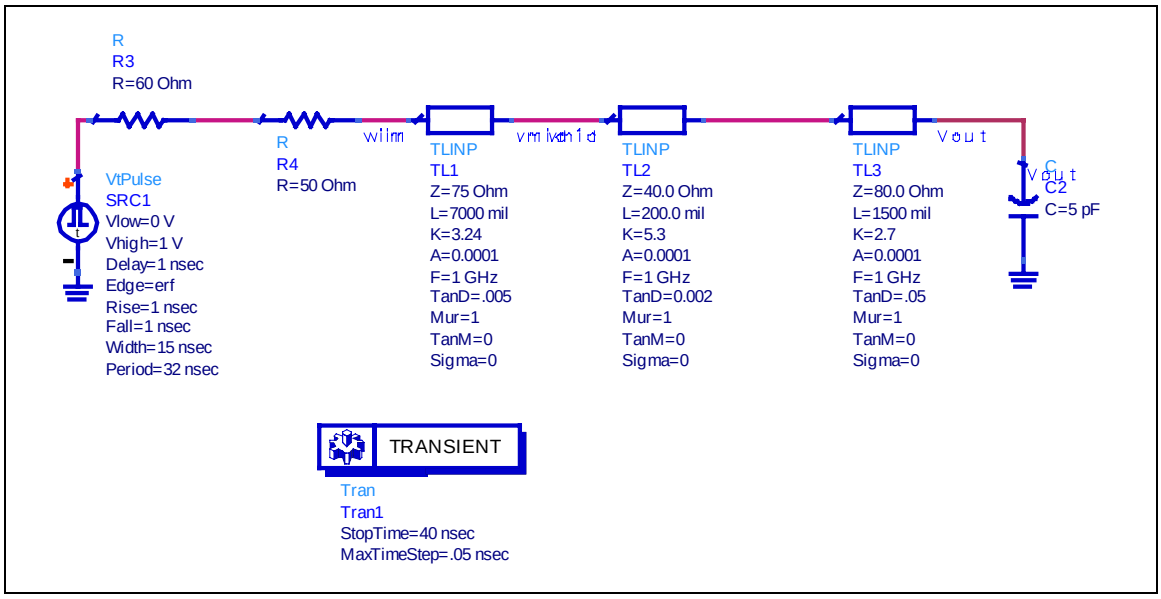


Figure 3-22 EESOF Simulation of the FIFO Input Data Lines

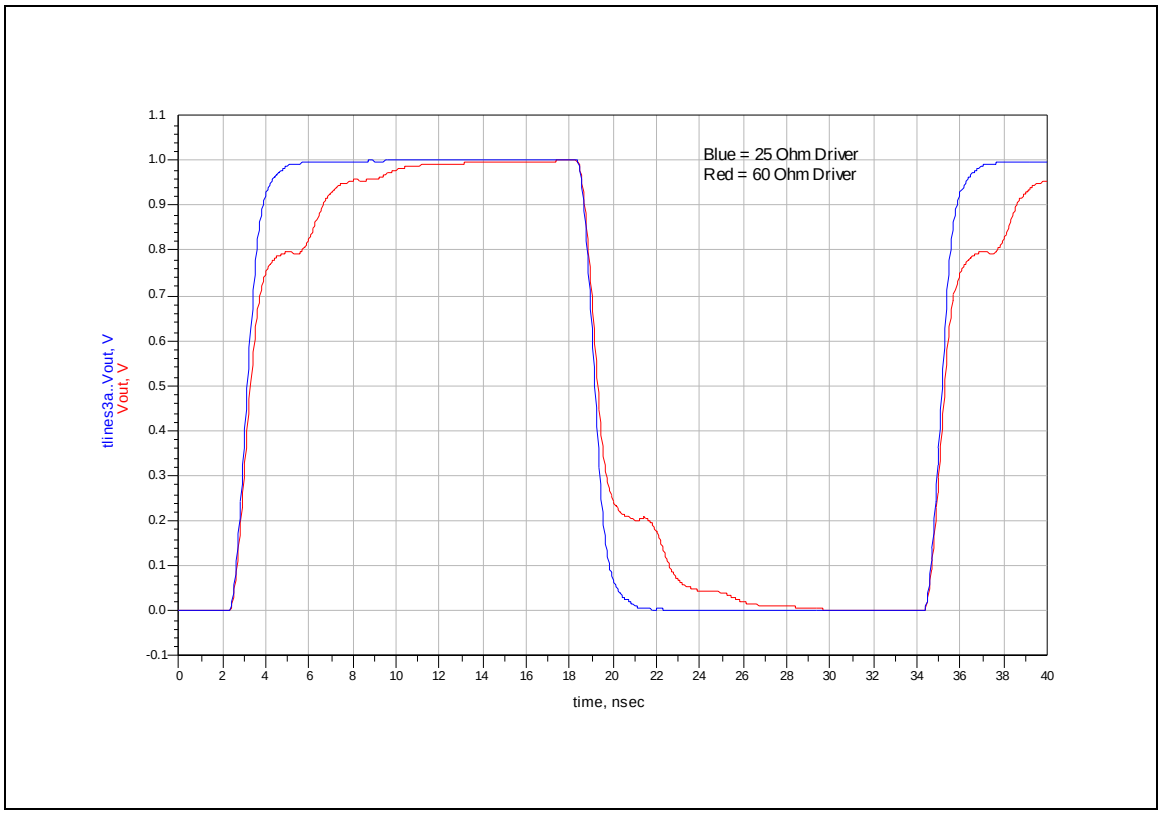


Figure 3-23 EESOF FIFO Input Data Simulation Results

Integration Counter and Reset Controller

The integration counter and reset controller were both implemented using Altera PLDs. The schematic for both circuits is shown in Figure 3-24. Because the data acquisition system will have multiple averaging cards and the integration count on all cards must be synchronized, the reset and integration circuit operates in either a master or slave mode. The mode is manually selected using jumper J9.

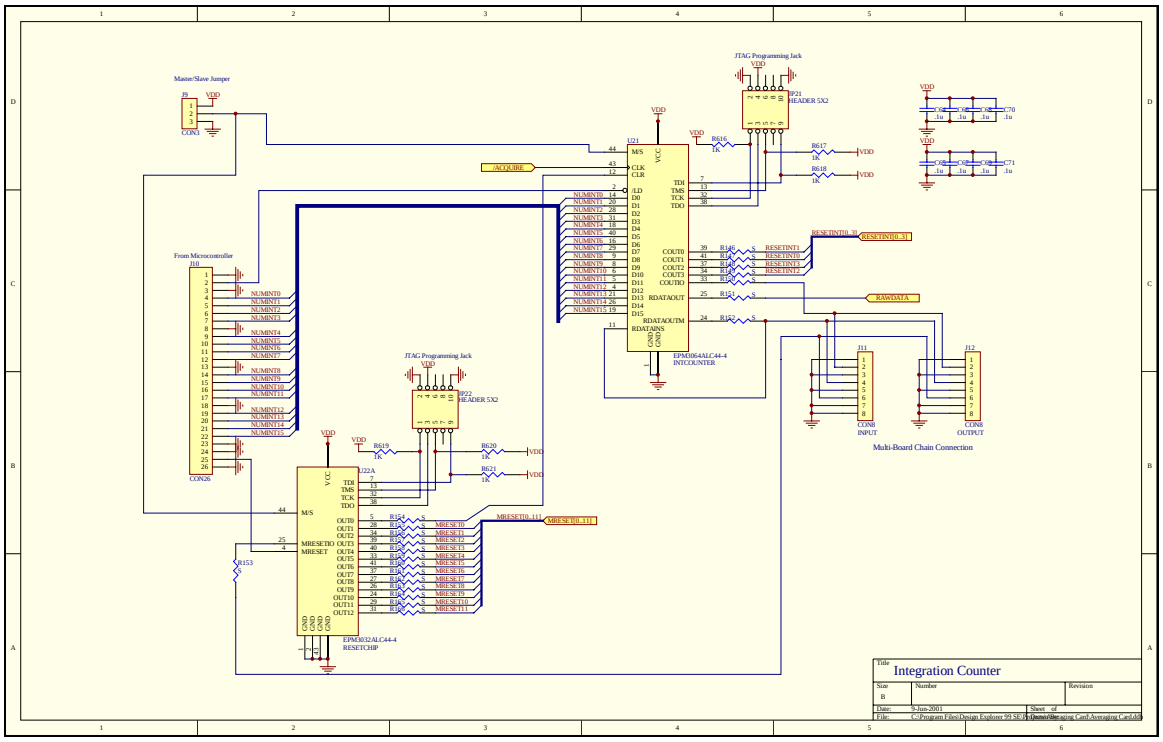


Figure 3-24 Integration Counter and Reset Schematics

Master Averaging Card

The master Averaging Card is the only card in the system that actually counts the number of integrations. This is performed by PLD U21 by incrementing an internal counter at each rising edge of the *ACQUIRE* signal. After reset, the computer system programs the desired number of integrations into the integration counter. When the internal count is equal to the programmed value, the *RESETINT* signals are asserted to signal the DSPs that an averaging cycle is complete.

The master Averaging Card must pass the *RESETINT* signal to the slave cards in the system. This is output via pin 33 on the integration counter and transferred to the other boards via connectors J11 and J12. Two connectors are used so that multiple boards may be chained together vertically. Connector J11 is located on the top of the PCB and J12 on the bottom. When the cards are mounted horizontally and stacked

vertically in a rack, the user connects J11 of the bottom card to J12 of the top card to distribute the *RESETINT* signal through the entire stack of cards. Note, with this configuration, the master card can be placed anywhere in the stack. The master PLD output is set to a slow slew rate mode, reducing the edge rate of the PLD to over 10 ns thus making transmission line effects negligible for the chain.

The master Averaging Card is connected directly to the timing system microcontroller through connector J10. This connector contains the necessary signals to transfer the number of integrations and the reset signal from the system microcontroller to the integration counter and reset controller. The number of integrations is asserted on the 16 data lines and the load line (J10 pin 2) is asserted to latch the value into the integration counter chip. The reset signal is asserted onto the reset controller chip. In master mode, the reset controller distributes the signal with the correct polarity (active high or active low) to 13 lines on the master Averaging Card. Additionally, line 25 is used as an output and distributes the reset signal to the other Averaging Cards via connectors J11 and J12 as described previously.

Originally, the system was designed for a 16-bit latched parallel connection to transfer the integration count. In order to reduce the pin count on the microcontroller, the integration counter was later modified for a serial load connection. The serial connection remaps the data pins as shown in Table 3-2.

Pin	Parallel Load Function	Serial Load Function		Pin	Parallel Load Function	Serial Load Function
1	GND	GND		14	Data8	
2	Latch	Latch		15	Data9	
3	GND	GND		16	Data10	
4	Data0	Ser Data		17	Data11	
5	Data1	Ser Clock		18	GND	GND
6	Data2			19	Data12	
7	Data3			20	Data13	
8	GND	GND		21	Data14	
9	Data4			22	Data15	
10	Data5			23	GND	GND
11	Data6			24	GND	GND
12	Data7			25	Reset	Reset
13	GND	GND		26	GND	GND

Table 3-2 Serial and Parallel Load Pinouts of Connector J10

From the schematics, two additional signals (*RDATAINS* and *RDATAOUTM*) are shown from the integration counter to connectors J11 and J12. The *RDATAOUTM* signal is asserted by the master Averaging Card when the integration count is set to one (raw data). The RDATAINS is used by a slave card as an input for the same signal. These lines were originally intended so the DSP could execute different code to improve the throughput of raw data. This feature was never implemented in the DSPs and these lines are currently unused.

Slave Averaging Card

In slave mode, the reset and integration complete signals are input from connectors J11 and J12. These signals are asserted on pin 25 of the reset controller and pin 33 of the integration counter. The reset controller operates identically in master or slave mode except that in slave mode the reset input comes from the master Averaging Card instead of the microcontroller. In slave mode, the integration counter simply fans the *INTCOUNT* signal from connectors J11 and J12 to the four DSPs and does not perform any other function.

Digital Signal Processors

The Texas Instruments TMS320C6202 200-MHz digital signal processors were selected for this design based on their high speed and dual 32-bit memory interface busses. The digital signal processor schematic is shown in Figure 3-25. From the figure, the two main DSP interface busses can be seen. The 32-bit external memory interface (EMIF) bus shown on the left side of the schematic is used for the input FIFOs and the flash memory used for processor boot loading. The 32-bit expansion memory interface (XMIF) bus shown on the right side of the schematic is used solely for the output FIFOs. In the lower left of the schematic are the reset, input FIFO empty, and reset integration interrupts all of which were previously described. The JTAG programming interface seen at the top of the schematic is used to interface the DSP directly to a computer for programming and debugging. A disable signal is output from the general purpose output *TOUT0*. This signal is used to disable the acquire signal when the DSP is sending data to the output FIFO.

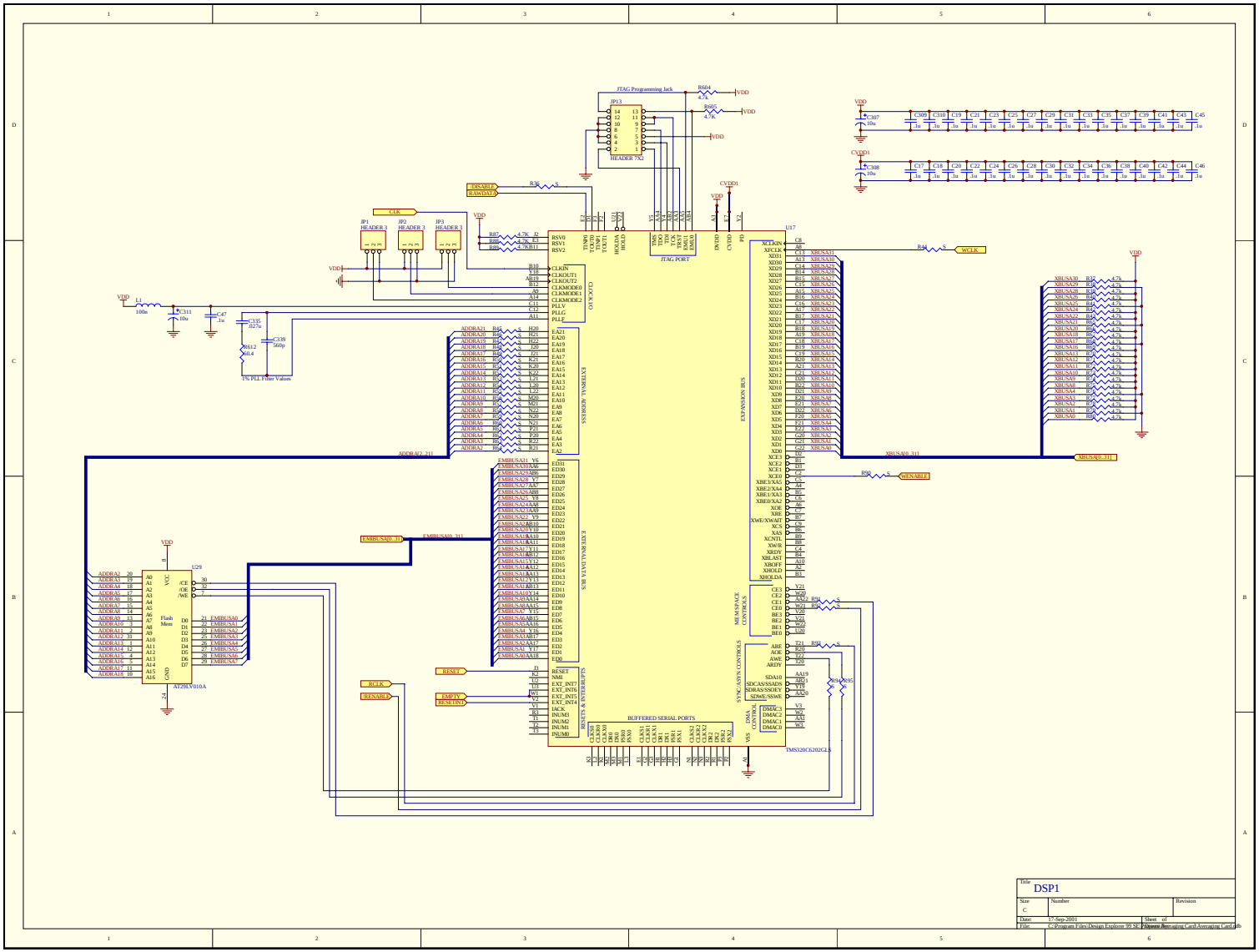


Figure 3-25 DSP Schematic

EMIF Interface

The EMIF interface is a 32-bit synchronous or asynchronous bus that provides a glueless interface to a variety of standard memory types. Four separate address spaces can be accessed without glue logic and are activated using signals *CE0-CE3*. In asynchronous mode, the bus can be used as a generic interface to almost any I/O device. In this design, a standard 8-bit asynchronous mode is used for boot loading and programming the flash memory while a custom interface was designed for interfacing with the input FIFOs. As seen in the schematic, the flash interface uses address space *CE1* and it directly connects to the asynchronous write enable (AWE) and asynchronous output enable (AOE) lines of the DSP. To read and write from the flash memory, the programmer simply sets the CE1 address space for an 8-bit asynchronous interface, then accesses the associated address.

A non-standard asynchronous interface was designed for the input FIFO because the synchronous interface was designed for random access memory (RAM) and the standard asynchronous mode is not appropriate for a synchronous (clocked) FIFO. As seen in the schematic, the asynchronous read enable (ARE) line is used as the FIFO clock signal and the CE0 signal is used as a FIFO read enable. The disadvantage to this connection is that a bogus asynchronous read must be repeatedly asserted by the DSP in order to clock the FIFO to update the FIFO flags. This manual clocking must only be performed when the DSP is in the idle state waiting for data to arrive so performance is not compromised in any significant way. During a FIFO read operation, the DSP simply reads a single address in CE0 causing the CE0 and ARE lines to toggle and the FIFO to output a 32-bit word.

XMIF Interface

The XMIF interface is an auxiliary 32-bit bus for interface to standard memory devices including synchronous FIFOs. Up to four devices can be connected to the XMIF without glue logic by using the address spaces mapped to XCE0-XCE3. As seen in the schematic, XCE0 was used to connect to the output FIFOs. The FIFOs connect directly and seamlessly to the XMIF using the XFCLK, XD0-31, and XCE0 signals. The disadvantage to using the XMIF is that the DSP core does not have direct access to it. The only way to access devices on the XMIF is through the use of direct memory access (DMA) transfers. Therefore, the bus is used for large block transfers where the size of the transfer is known (such as the output stage of the data acquisition algorithm) or when an interrupt can signal the DMA system to start/stop transfers.

DSP Boot Configuration

Since the DSP can interface to a variety of memory types, the memory type for the boot ROM must be programmed prior to power up. This is selected via 23-pullup/pulldown resistors on the XMIF bus as shown in Figure 3-25. The boot configuration selected for the averaging card is shown in Table 3-3.

XMIF pin	Function	Set to	Value
0-4	BOOTMODE	Internal Memory at Address 0, 8 bit ROM bootload	10110
5-7	RESERVED		
8	LEND	Little Endian	1
9	FMOD	XOE Reserved for a single FIFO in XCE3	1
10	XARB	Internal Expansion Bus Arbiter Enabled	1
11	HMOD	External Host Interface in Synchronous Mode	1
12	RWPOL	XR Signal is Active High	1
13	BLPOL	XBLAST is Active High	1
14-15	RESERVED		
16-18	MTYPE XCE0	32 Bit Synchronous FIFO Interface for XCE0	101
19	RESERVED		
20-22	MTYPE XCE1	32 Bit Synchronous FIFO Interface for XCE1	101
23	RESERVED		
24-26	MTYPE XCE2	32 Bit Synchronous FIFO Interface for XCE2	101
27	RESERVED		
28-30	MTYPE XCE3	32 Bit Synchronous FIFO Interface for XCE3	101
31	RESERVED		

Table 3-3 Processor Boot Configuration

Flash Memory

The flash memory selected for the design is the Atmel AT29LV010A. This flash memory has 128 kB of storage which was deemed appropriate since the DSP code for this application is around 30 kB. The advantage to flash memory over traditional EEPROMs is that flash memory does not require special programming voltages and can be reprogrammed directly by the DSP by following a special programming algorithm prior to performing a write operation. As a result, flash memory can be packaged in small surface mount devices and soldered directly to the PCB. The algorithm for the AT29LV010A flash is shown in Figure 3-26.

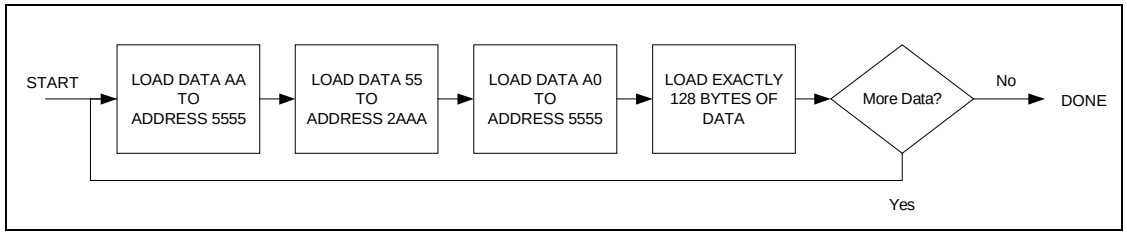


Figure 3-26 Flash Programming Algorithm [11]

As seen in Figure 3-26, an exact sequence of events must occur for the flash memory to be overwritten. Special keywords must be written to three registers inside the flash followed by exactly 128 bytes of data. If any key is wrong or the wrong amount of data are written, the memory location will not be overwritten. For data larger than 128 bytes, the algorithm may be repeated after a short delay. The exact delay does not need to be measured as the processor can loop, reading the last memory location written to the flash, until data are read back as valid.

DSP Clock

The selected DSPs operate at speeds up to 200 MHz. The processors can be directly clocked at that frequency, or, to simplify clock system design, the processors have a built in phase locked loop (PLL) which is used to multiply a lower frequency clock. For the averaging card, a 50-MHz clock signal is input to the DSP and the PLL multiplies by four to obtain 200 MHz. The PLL loop components are shown in Figure 3-25 as C335, C339, and R612. The values were recommended by the manufacturer and were specified as 1% tolerance. However, due to problems obtaining the exact values and tolerances, close nominal values of 5% tolerance were selected. These values were empirically found to be acceptable as the 200-MHz clock output was stable and excessive jitter was not observed. In the future, the exact 1% values shown in the schematic should be used if possible.

Clock Generator

The DSP clock generation circuit is shown in Figure 3-27. Oscillator U27 outputs a 50 MHz LVTTL clock signal to the Reset PLD. There it is fanned out through a single transmission line to each processor on the board. This fanout was necessary to preserve

signal integrity of the oscillator signal since it must traverse the entire PCB, connecting to four loads.

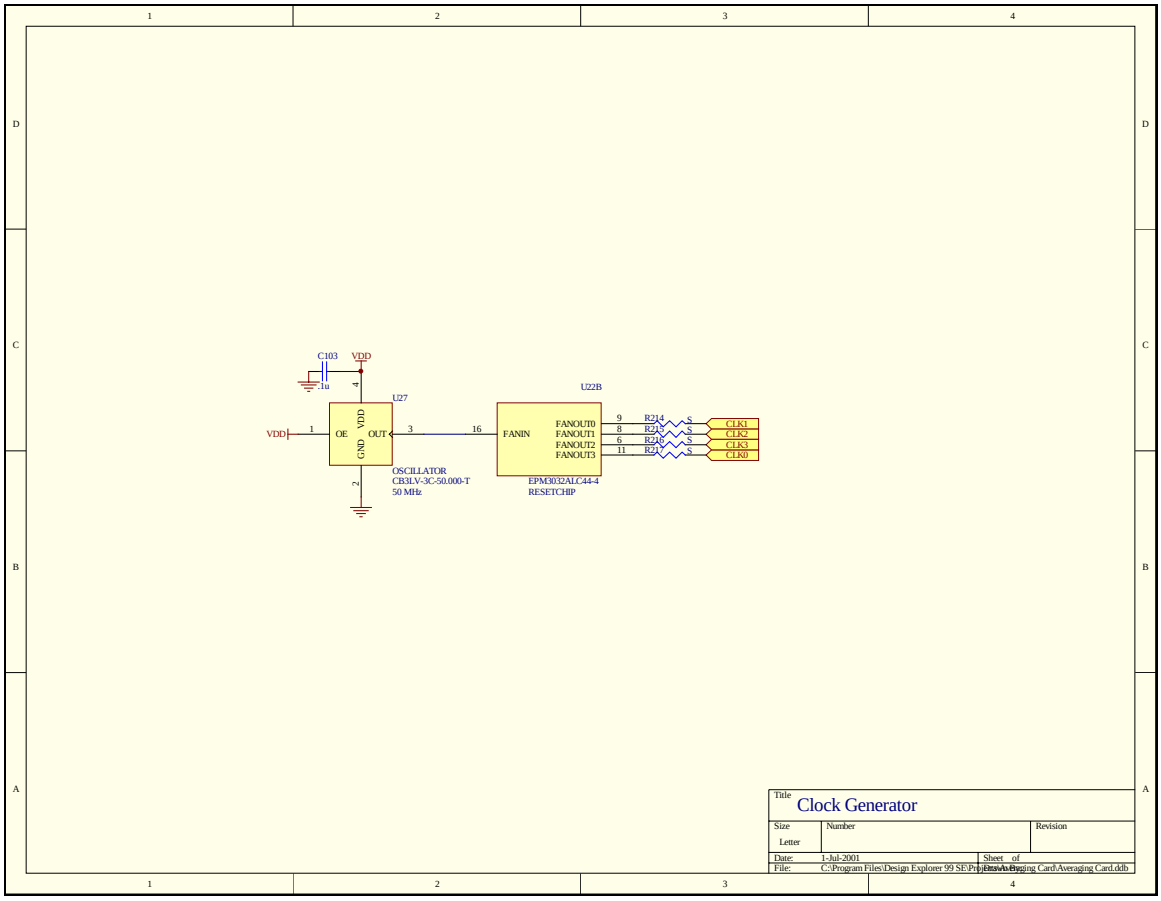


Figure 3-27 DSP Clock Generation Schematic

Output FIFOs

The output FIFOs are used to store the averaged data until the computer is ready to transfer to the hard drive. The same IDT FIFOs are used as for the input FIFOs and are used in an identical dual-18-bit configuration. A schematic of the output FIFOs is shown in Figure 3-28.

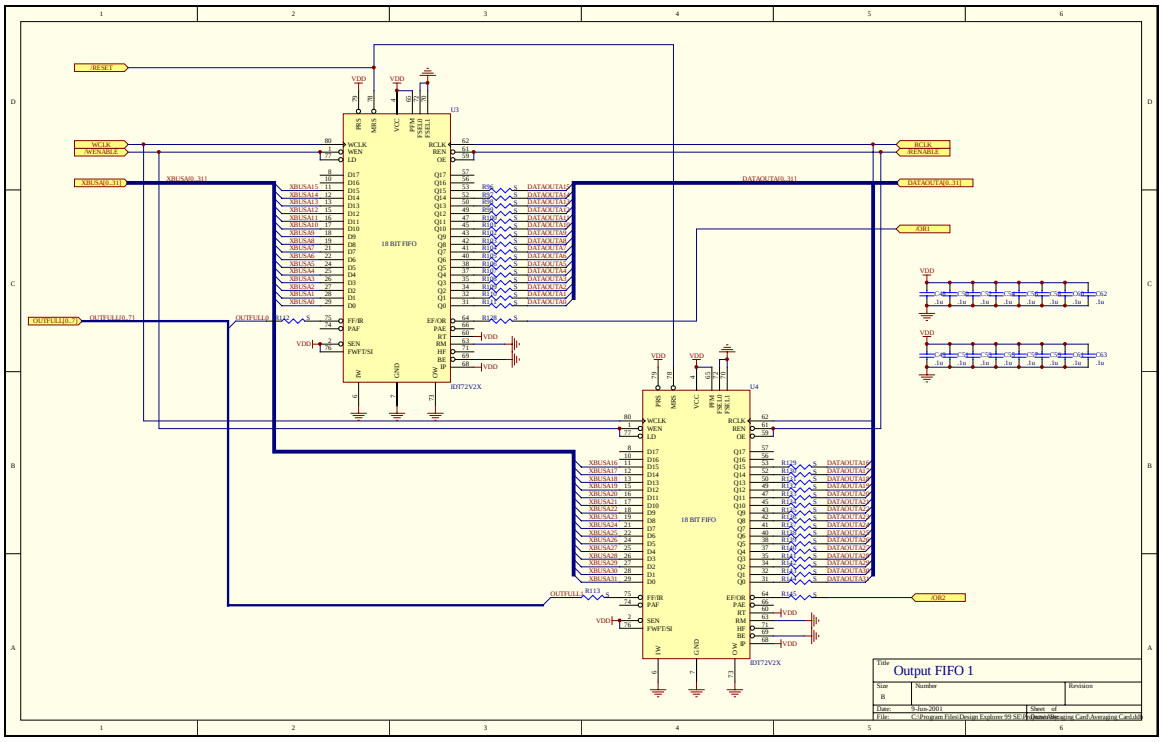


Figure 3-28 Output FIFO Schematic

The input to the FIFOs is the XMIF bus of the associated DSP. Since the DSP provides a glueless interface, the data, write enable, and clock lines are connected directly to the DSP. Again, a series terminated, split transmission line structure was used with results similar to the input FIFOs. The output of the FIFOs is connected to the output multiplexer circuit. The output clock operates at 20 MHz and is constantly asserted in order to update the FIFOs output ready flag. When the output multiplexer is ready to transfer data and the FIFO has data (*OR* flag is low) the multiplexer asserts the read enable (*RE*) line and data are immediately present on the FIFO outputs since FWFT mode is used (just as the input FIFO). When all data have transferred, the *OR* flag goes high and the multiplexer will de-assert the *RE* line and stop the transfer.

Output Multiplexer and Connector

The output multiplexer transfers data from the output FIFOs to the downstream hardware when the acquisition computer is ready. The multiplexer circuit was designed using three Altera PLDs. As seen in Figure 3-29, the multiplexer circuit switches four 32-bit busses into a single 32-bit bus. This requires 160 pins for data plus additional pins for controlling and synchronizing the structure. The large pin count was satisfied by using two 144-pin PLDs used solely for switching the data lines while a separate 44-pin PLD is used to control and synchronize the switching.

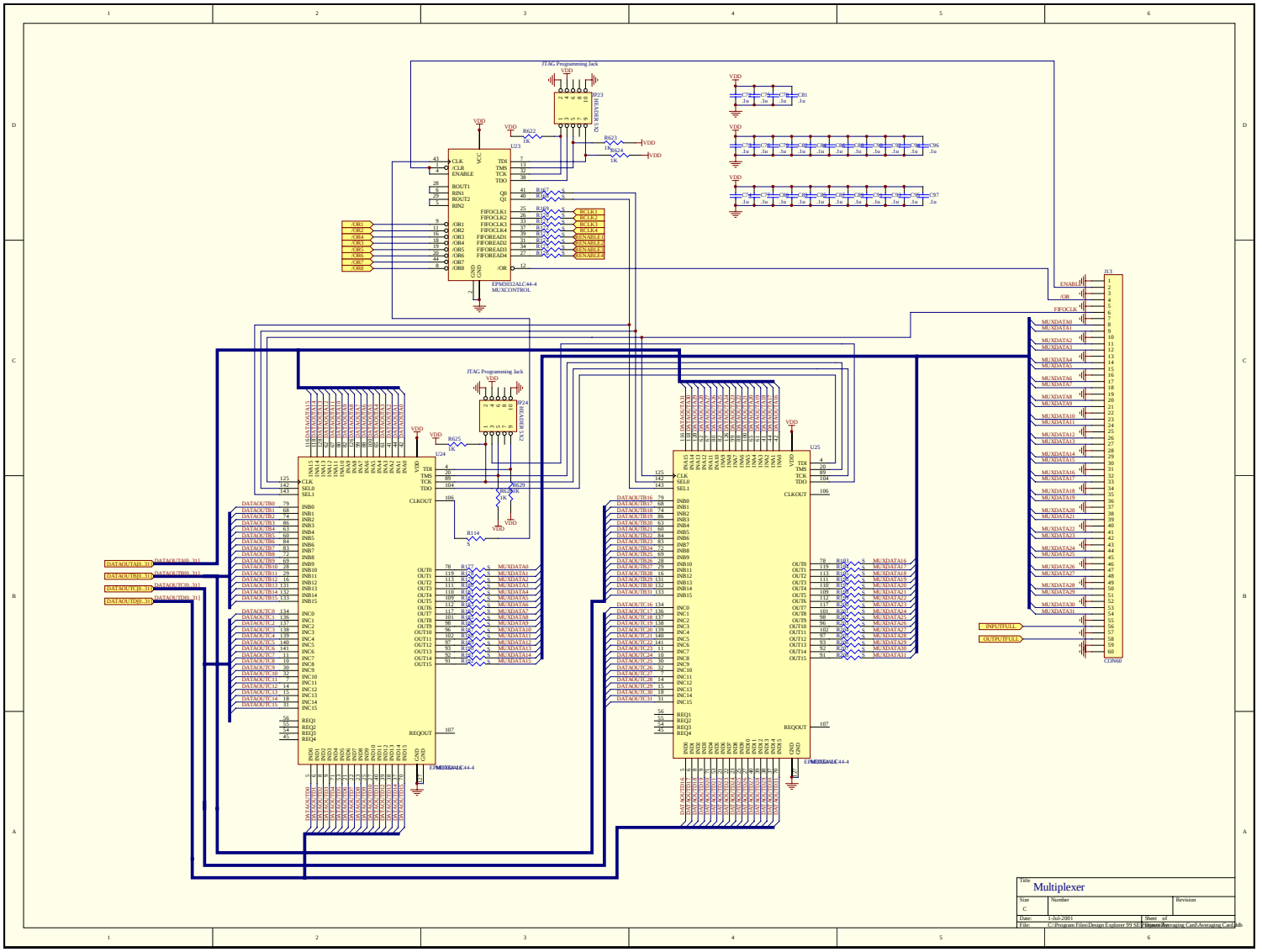


Figure 3-29 Output Multiplexer Schematic

The multiplexer operation begins when the *OR* signals from the FIFOs are asserted. The internal logic of U23 is such that a FIFO ready condition is only detected when all 8 output FIFOs are ready to transfer data. When this condition is met, the */OR* signal is output from U23 signaling the downstream hardware that the Averaging Card has data ready to transfer. When the downstream hardware is ready to accept data, it responds with the *ENABLE* signal and U23 initiates the transfer. U23 will assert the *RENABLE* line for a single FIFO while asserting the SEL[0:1] signals to U24-25 so that the associated data lines are switched to the Averaging Card outputs. After transferring data from the first FIFO, the *RENABLE* line of the second FIFO is selected along with the associated SEL[0:1] signals and data are transferred from the second FIFO to the output connector. U23 continues selecting between the FIFOs until all data are transferred and the FIFOs become empty.

When the FIFOs become empty, the */OR* flags will go high. The internal logic of U23 is such that all */OR* flags must be high before a FIFO empty condition is detected. At that time, the */OR* output of U23 will be de-asserted signaling to the downstream hardware that the FIFOs are empty. The downstream hardware will de-assert the ENABLE signal to halt the transfer of data.

Due to the ordering of data on the input connector of the Averaging Card, the multiplexer must switch the output data in a non-obvious way. The input connector data sequence was shown in Table 3-1. The data sequence from that table can be mapped to the DSP channels as shown in Table 3-4 resulting in the multiplexer having the switching pattern shown in Table 3-5.

DSP Number	Sample Numbers
1	2,4,6,8
2	10,12,14,16
3	1,3,5,7
4	9,11,13,15

Table 3-4 Correlation Between DSP and Temporal Sample Numbers

Sample Number	MUX FIFO number	SEL0	SEL1
1	3	1	0
2	1	0	1
3	3	1	0
4	1	0	1
5	3	1	0
6	1	0	1
7	3	1	0
8	1	0	1
9	4	1	1
10	2	0	1
11	4	1	1
12	2	0	1
13	4	1	1
14	2	0	1
15	4	1	1
16	2	0	1

Table 3-5 Output Multiplexer Circuit Switching Pattern

The output connector, shown in Figure 3-29, is a 60-pin standard 100-mil header and is used to connect the Averaging Card to the multiplexer card via a standard 60-pin ribbon cable. The connector carries a 32-bit output word, the /OR flag, and the ENABLE flag discussed above. The connector also inputs the 20-MHz output clock used to clock the output FIFOs as well as two error detection signals, *INPUTFULL* and *OUTPUTFULL*, described in the next section. The connector has a ground wire between each control/clock signal and one ground wire between each pair of data signals in order to maintain signal integrity. To reduce the possibility of transmission line effects, the slow slew rate outputs were used on all PLDs and all lines were series terminated. Since the signals operate at less than 20 MHz, the design results in excellent signal integrity.

Error Detection Circuitry

Error detection circuitry is used to detect an overflow of the input or output FIFOs. This circuit was implemented as a single Altera PLD, U28, which asserts a separate output if any of the input or output FIFOs become full. These two signals are passed to the output connector for use by the acquisition computer and/or LED indicators on the front of the system.

Averaging Card PCB

The averaging card PCB is constructed using industry standard FR-4 board material. Due to the ultra fine pitch BGAs used on the DSPs, unusually small feature sizes were required to produce the board including 4 mil/4 mil line width/spacing and 10 mil via holes with 15 mil via pads. A stackup of the PCB is shown in Figure 3-30.

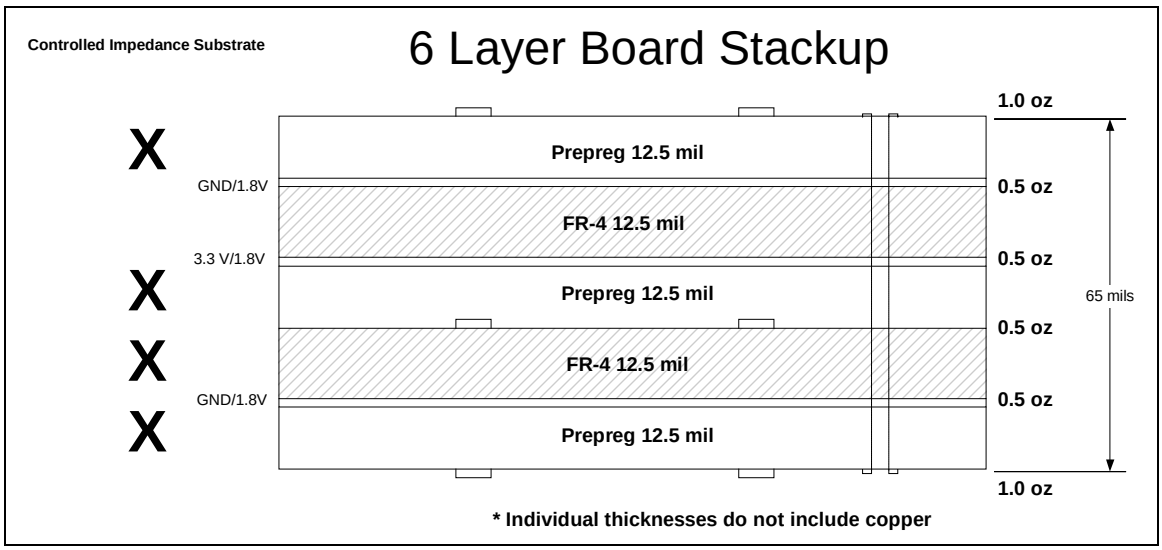


Figure 3-30 Averaging Card PCB Stackup

As seen in the figure, the stackup is highly standard. The only unusual feature about the board stackup is that a split 1.8 V plane is routed on the plane layers to supply the core voltage to the DSPs. The planes had to be segmented beneath the DSPs so that the DSPs would have access to 1.8V, 3.3V, and ground. At the same time, care was taken so that the 1.8V plane did not disturb the return path for any critical transmission lines. The split plane

structure can be seen in the PCB layout of APPENDIX G. A photograph of the prototype board shown in Figure 3-31.

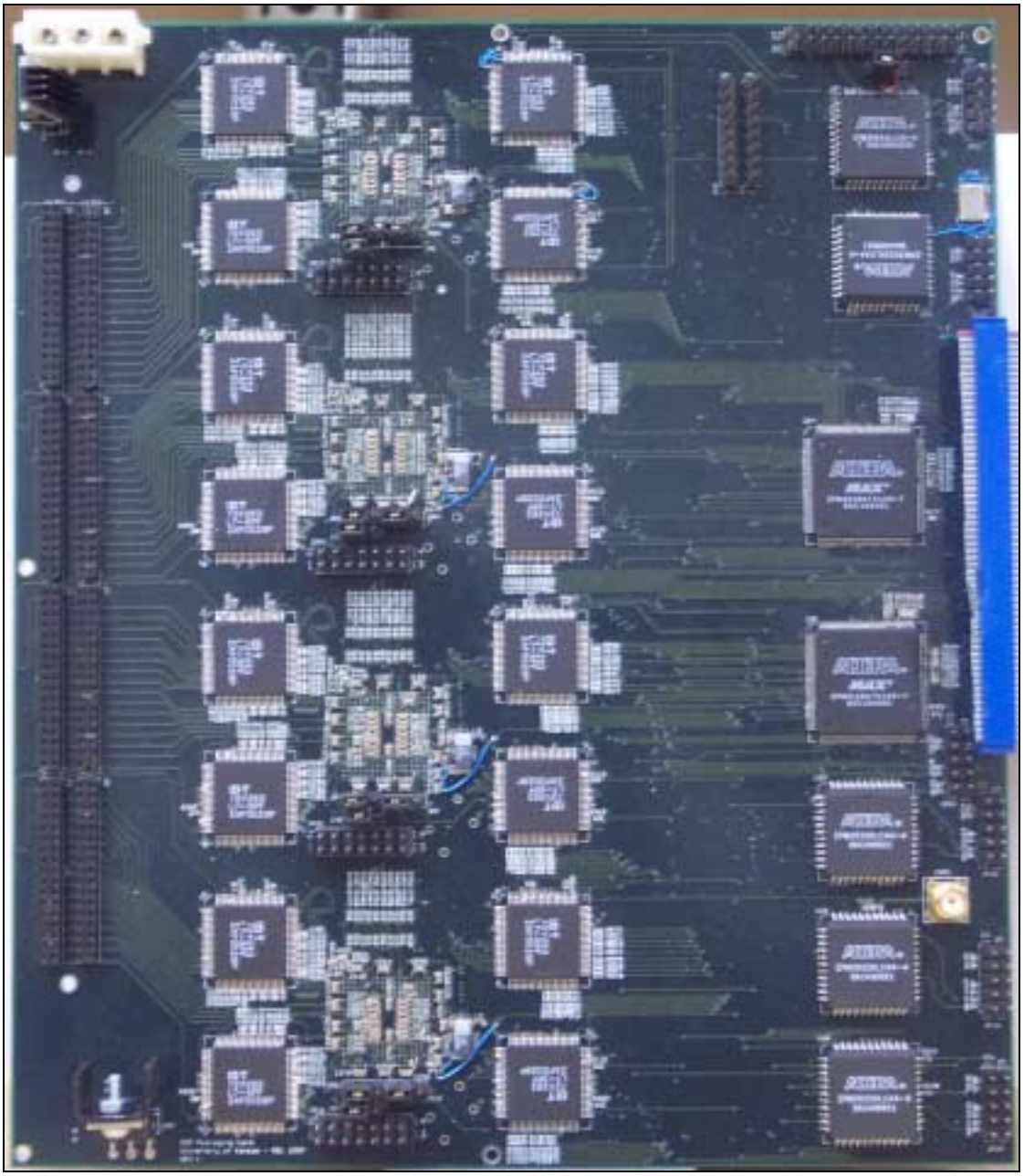


Figure 3-31 Averaging Card Photograph

DSP Programming

Processing Program

The processing program is a DSP based implementation of the coherent integration algorithm. This program is implemented using raw assembly language, analyzed to single

clock cycles at times, to yield the maximum possible performance from the system. The actual code is shown in APPENDIX L. In order to write raw assembly, the pipelined structure of the DSP must be known and compensated for to avoid data access hazards and optimize branch statements. A flow diagram of the program is shown in Figure 3-32.

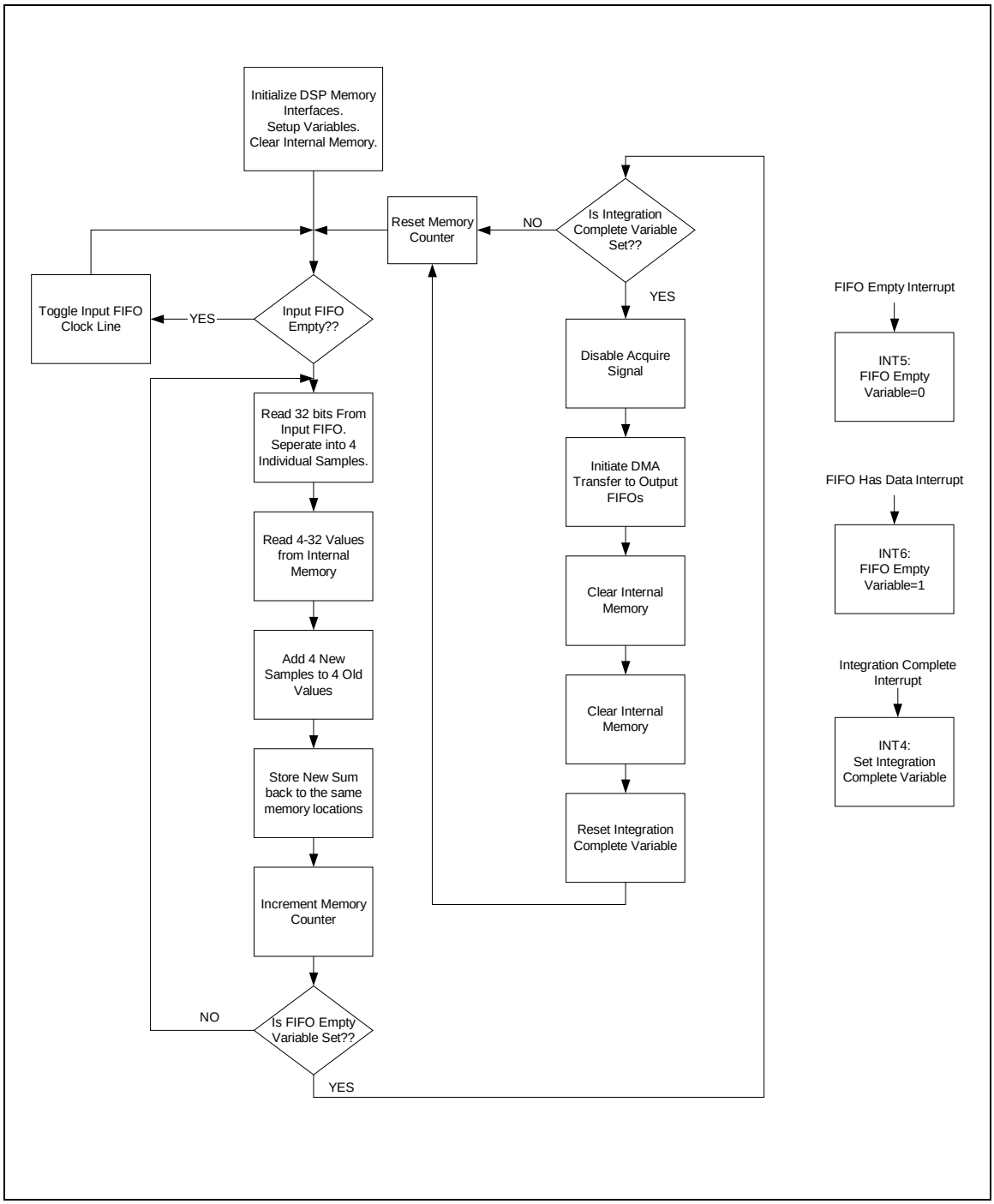


Figure 3-32 Flow Diagram of the DSP Assembly Program

As seen in the figure, the program consists of two main loops. In the first loop, the FIFO is empty and the DSP loops while toggling the FIFO clock line (in order to update the FIFO empty flag). When the FIFO receives data, the empty flag will toggle triggering interrupt #6 in the processor which toggles the FIFO empty variable. Returning from the interrupt the processor will still be in the first loop which will then exit because the empty variable has changed.

The second loop is entered in which the DSP reads and processes data from the input FIFO. Data are input as four packed 8-bit words. These words must be unpacked into individual 8-bit words and added to the associated values in memory. After the four words are written to memory, the memory counter is incremented by four and new data are read from the input FIFO. Eventually, the input FIFO will empty, triggering interrupt #5 which clears the *FIFO EMPTY* variable. The processor returns from this interrupt into the averaging loop which will then exit since the *FIFO EMPTY* variable is changed.

Upon exit from the second loop, the processor will either return to the main loop (with the memory count reset) or it will transfer the averaged data to the output FIFO depending on whether the *INTEGRATION COMPLETE* variable has been set. If the integration is complete, the length of memory is calculated from the current memory pointer and this value is used to initiate a DMA transfer to the output FIFOs via the XMIF bus. Once the DMA transfer is underway, the program performs a loop, clearing all data memory up to the depth of the input FIFOs (anything beyond this is not necessary). Finally, after clearing memory, the *INTEGRATION COMPLETE* variable and memory counter are reset and the processor enters back into the first loop waiting for new data at the input FIFO.

The time needed to process one acquisition cycle of data can be estimated based on the assembly language loops and can be used as a measure of the performance of the system. The exact number of CPU cycles for register based instructions and internal memory access is given in [12]. For external memory accesses the delay consists of a pipeline delay, a setup delay, and an external device delay. The pipeline delay is simply the number of clock cycles that must pass before the data are available, assuming the setup and device delay is zero. The setup delay represents the processors overhead in the memory controller. Finally, the device delay in this case is zero since the FIFOs are used in first word fall through (FWFT) mode (the first word input into the FIFO is available immediately at the output). The critical loop of the code is shown in Figure 3-33 with the number of clock cycles for each operation shown in braces.

```
acquire_data_loop:
            ldw   *A3, A2            ;grab  4 bytes from FIFO                    {10}*
            ldw   *+A8[0],A10        ;get old data byte 1 from memory            {2}
            ldw   *+A8[1],A11        ;get old data byte 2 from memory            {2}
            ldw   *+A8[2],B10        ;get old data byte 3 from memory            {2}
            ldw   *+A8[3],B11        ;get old data byte 4 from memory            {2}
            nop                       ;5 cycles needed to access FIFO            {1}

            mv    A2,B2;             ;copy data into B register                  {1}

            and   A2,A12,A14         ;unpack data by masking                     {1}
       ||   and   A2,A13,A15         ;masked data is R's 14, 15
       ||   and   B2,B12,B14
       ||   and   B2,B13,B15

            shru  A15,8,A15          ;unpack data by shifting                    {1}
       ||   shru  B14,16,B14

            shru  B15,24,B15                                                     {1}

            add   A10,A14,A10        ;add new and old data byte 1                {1}
       ||   add   A11,A15,A11        ;add new and old data byte 2
       ||   add   B10,B14,B10        ;add new and old data byte 3
       ||   add   B11,B15,B11        ;add new and old data byte 4

            cmpeq   0,B3,B0          ;check loop condition (updated by interrupt) {1}
    [!B0]   B     acquire_data_loop  ;Use 5 delay slots after a branch           {1}
            stw   A10,*A8++;         ;store to memory, increment data address    {1}
            stw   A11,*A8++;         ;"                                          {1}
            stw   B10,*A8++;         ;"                                          {1}
            stw   B11,*A8++;         ;"                                          {1}
            nop                                                                  {1}
            nop
```

Figure 3-33 Performance Analysis of the Integration Code

As seen in the figure, the total number of CPU clock cycles to process four 8-bit samples of data is 35. This can be reduced to 29 cycles by parallelizing internal memory

accesses, cutting the memory access penalty in half. This code was implemented but makes the algorithm more difficult to understand and the 5 clock cycle benefit is not significant enough to justify it. The 2 cycle penalty on memory accesses was discovered only after the system was designed and represents one cycle of execution and one cycle of pipeline stall as the processors internal memory bus is 16-bits and all memory accesses in this code are 32 bits. The FIFO access latency is an empirical value found after the system was designed. It appears to be EMIF latency as it exists on the evaluation board for other memory types as well.

The minimum PRI can then be estimated as follows:

$$PRI_{\min} = \frac{N_s}{N_d} \, x \, CPU \, x \, T_{cpu} \tag{3}$$

where,

N_s	=	Number of samples acquired during an acquisition interval, samples
N_d	=	Number of DSPs on the Averaging Card, no units
CPU	=	Number of CPU cycles per input sample, cycles/sample
T_{cpu}	=	Period of the CPU clock, ns/cycle

For a typical acquisition session, the number of samples per channel would be 8,000 divided among the 4 DSPs. For a clock frequency of 200 MHz, the clock period is 5 ns. Based on the calculations above, the CPU cycles per sample for the non-optimized code is calculated to be 8.75 cycles/sample. The minimum PRI is therefore calculated to be:

$$PRI_{\min} = \left(\frac{8{,}000 \; samples}{4} \right) \left(\frac{8.75 \; cycles}{sample} \right) \left(\frac{5 \; ns}{cycle} \right) = 87.5 \; \mu s$$

The maximum PRF is therefore 11 kHz, far below the design goal of 30 kHz. Even with the optimized code, the maximum PRF is calculated to be 13 kHz. The original design estimates were calculated based on a memory and FIFO latency of 1 clock cycle each resulting in an initial performance estimate of 18 cycles per loop and a predicted PRF of 22

kHz which would approach the goal of 30 kHz with 300 MHz DSPs. Because most of the performance penalty is due to the slow EMIF access it would be worthwhile to focus on this aspect of the design for a speed improvement. The specification for this could not be found in any of the documentation so it would be worthwhile to contact Texas Instruments for more information if, in the future, it is desired to optimize the current design.

Flash Loading Program

In order to boot load the processor from the flash memory, the processor must write it's own program into the flash memory. This is accomplished via an in-circuit debugging (JTAG) cable and special flash programming code. An overview of the processors memory map for boot loading and flash programming is shown in Figure 3-34.

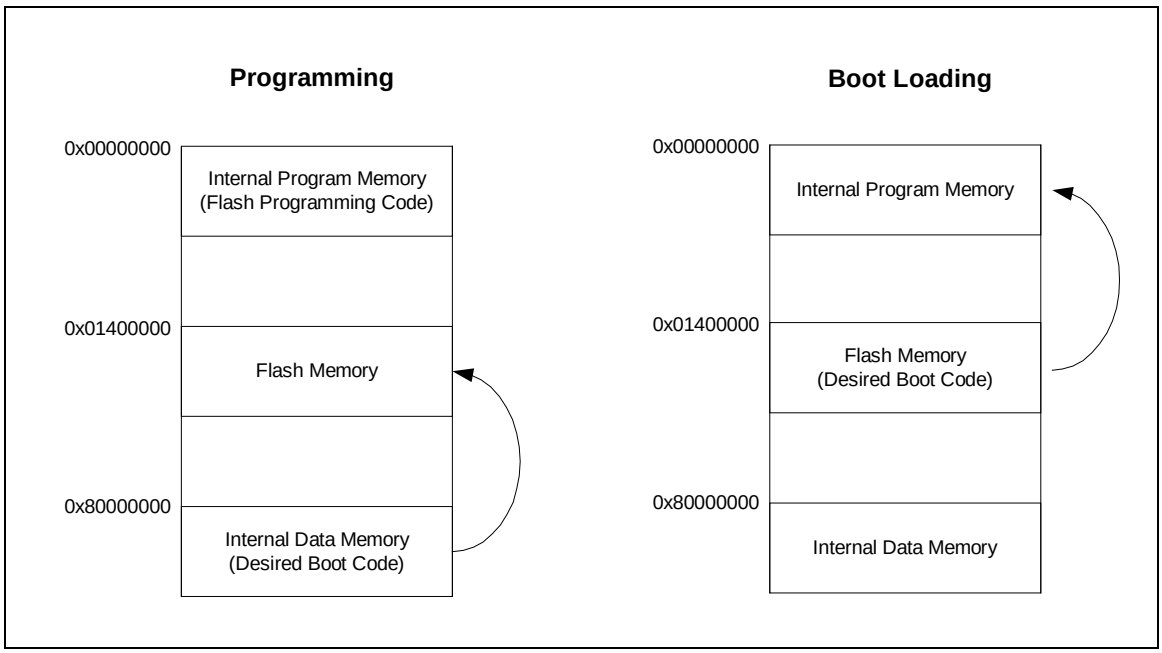


Figure 3-34 Overview of Flash Programming and Boot Loading

The flash programming operation is shown on the left side of the figure. The user compiles the desired boot code (i.e., averaging program) with start address 0x80000000. This code is then uploaded to the processor via the JTAG cable and is placed in the internal data memory. The flash loader code given in APPENDIX L is then compiled for start address 0x00000000 and uploaded in the same fashion. When the DSP is run, it will

execute the flash loader program which copies the boot code from data memory to the flash memory (using the flash programming algorithm discussed previously). Once finished, the JTAG cable is removed and, upon reset, the processor will perform a boot load as shown on the right side of the figure and immediately begin executing the boot code.

Revision History/Future

The circuit schematics and PCB have each been revised once. The BGA for the DSP is shown as a bottom view and was originally entered into the PCB layout as a top view. The simplest solution for this problem was to move the BGAs to the bottom side of the board thus preserving the internal routing pattern but swapping the top and bottom routing and components in that area. The BGAs should be moved back to the top layer to improve manufacturability if the board is ever re-routed in the future.

The original schematics and PCB had a number of errors that were corrected with the use of jumper wires and trace cutting on the prototype PCBs. These errors include the following:

- The PCB land pattern for oscillator U27 was drawn as a top view when the manufacturer has supplied a bottom view. The pads must therefore be swapped left to right.
- The pull up/down resistors for the DSP boot load were not included in the original design. These were hand soldered onto unused series resistor pads.
- The *WCLK* and *WENABLE* signals from each DSP were swapped in the schematics. This was corrected using jumper wires on the PCB.
- The */LD* pin on all FIFOs was incorrectly tied to the */WEN* pin in the schematics. This was corrected by lifting the pin and soldering it to VCC.
- The *RSV0, RSV1,* and *RSV2* lines on the DSPs were left floating in the schematics. These lines must be pulled up for correct operation of the DSP. This was corrected through a manual artwork change on the PCB combined with jumper wires.

All schematic errors have been corrected in APPENDIX B so that the schematics represent an operational board. However, since these changes were implemented by hand,

the PCB layout of APPENDIX G does not reflect these changes and must be revised for any future board production.

The design of this card utilizes cutting edge technology for PCB manufacturing through the use of the 0.8-mm pitch BGAs used on the DSPs. This ultra-dense packaging results in via sizes and copper widths/spacings that push the limits of manufacturability. As a result of this, the blank PCBs had a low yield (less than 25%) from the manufacturer. Additionally, the DSPs on the finished cards operated sporadically. This is believed to be due to either inadequate soldering of the BGAs or poor connectivity of required jumper wires to the BGA pads (also a soldering issue). Either way, a new board revision will be required to have a usable design. The design of the logic is known to be good as the board will perform as designed for a short period of time before a connection failure occurs causing intermittent operation of one or more DSPs.

The manufacturability issues, combined with the excessive number of jumper wires and hand soldering needed for this board, result in the need for an immediate PCB revision. The simplest solution is to re-route the board with larger (1.27 mm) BGAs for the DSPs. Additionally, the board should be made thicker (80-90 mils) to reduce flexibility and increase BGA adhesion. This, combined with implementing the jumper wire changes, should result in a solid and usable version of the board.

3-4 Multiplexer Card

Overview

The final data acquisition system will consist of two to four sets of ADC cards and Averaging Cards (each set making up one channel). It would be impractical to supply a separate parallel link to the acquisition computer for each channel, therefore additional hardware is needed to switch the four channels into a single channel for transfer to the

computer. The Multiplexer (Mux) Card performs this function and supports the physical layer handshaking required for interfacing with the computer DIO card. An overview of the Mux Card is shown in Figure 3-35.

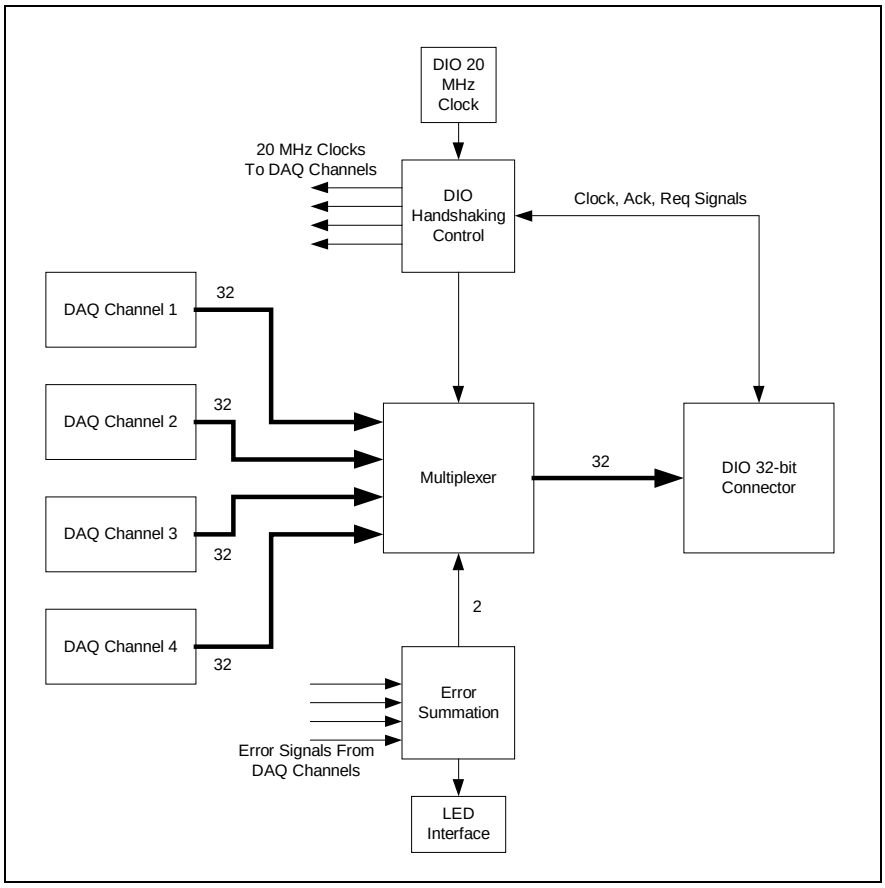


Figure 3-35 Multiplexer Card Overview

Detailed Description

The top level schematic of the Mux Card is shown in Figure 3-36. Circuit operation begins with the ACK signal being asserted by the DIO card signifying that it is ready to transfer data. When an averaging card has data, it asserts it's /*OR* flag to tell the multiplexer card that it has data to transfer. The multiplexer control circuitry senses this flag and selects the correct Averaging Card channel. When the channel is found, one word of the data is transferred when the Mux Card sends the *ENABLE* signal to the averaging card's output FIFO. When the word arrives at the Mux Card, it is latched to the *DATAOUT*

bus and asserted on the DIO cable. As the output word is transferred to the *DATAOUT* bus, the multiplexer control circuitry asserts the *REQ* signal to tell the DIO card that it has data ready to transfer. As long as the REQ and ACK signals are both asserted, the DIO card will latch data and the multiplexer circuit will continue to get new data upon every rising edge of the 20-MHz clock. The peak data transfer rate of the DIO card link is therefore 80 MB/s since the *DATAOUT* bus is 4 bytes wide.

If during transfer the DIO card's input buffer becomes full, it will de-assert the *ACK* signal and all data in the pipeline will freeze until the signal is reasserted. When the output FIFO of the averaging card becomes empty, the */OR* signal is de-asserted and the *REQ* signal is de-asserted after the last word is latched by the DIO card. The multiplexer circuitry will either select the next Averaging Card that has data or become idle if there are no more data to transfer.

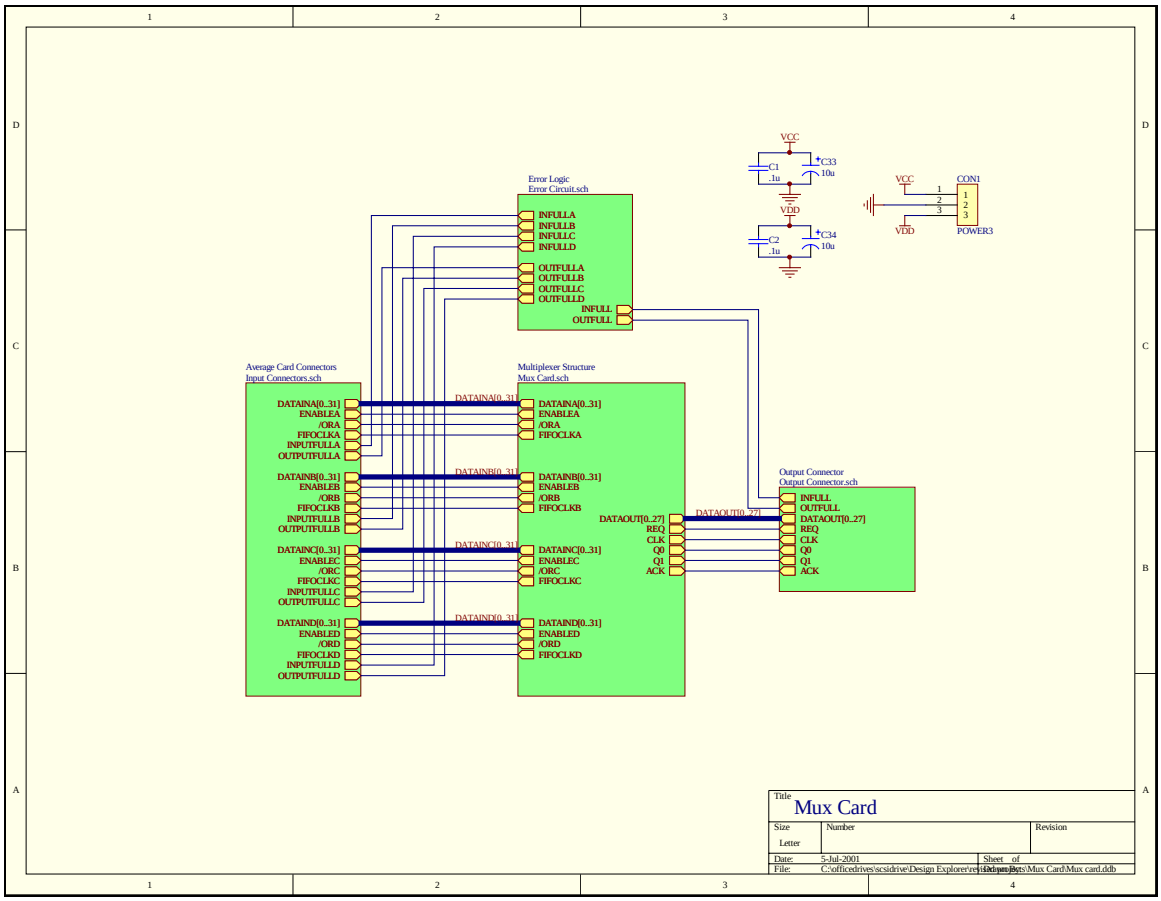


Figure 3-36 Top Level of the Multiplexer Card Schematics

Two FIFO full flags are output from each Averaging Card (one for the input buffer and one for the output buffer). These signals are then used to drive LED error indicators located on the front panel of the card enclosure. The eight error signals are also combined into two signals (*INFULL* and *OUTFULL*) that are output to the DIO card and transferred as the top two bits of data. These two bits allow the computer software to detect that an error condition has occurred and halt processing. The remainder of this section will be a detailed discussion of each of the hierarchical blocks shown in Figure 3-36.

Input Connectors

The input connectors are standard 60-pin, 50-mil latched header connectors. The latched connector was used because a non-latched 60-pin connector requires excessive force to disconnect. Standard 60 pin ribbon cables mate with each of the four headers to

connect the Mux Card to the four averaging cards. Since the signals are terminated and only operate at 20 MHz, the signal integrity through 2 feet of ribbon cable was found to be excellent. The pinout for a single connector is shown in Figure 3-37.

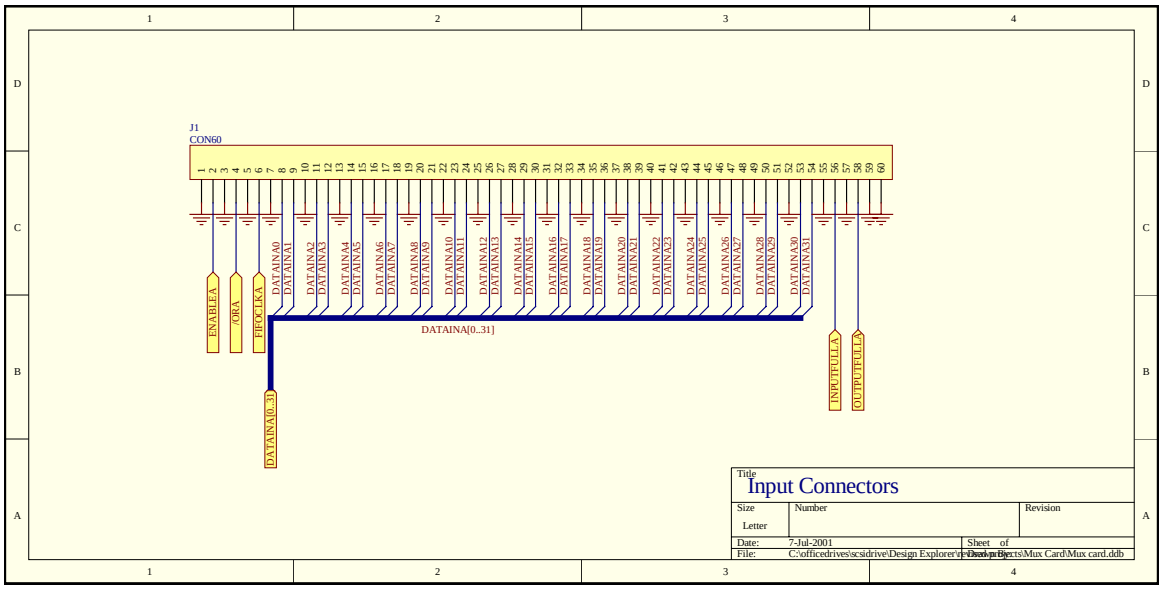


Figure 3-37 Mux Card Input Connector Pinout

Multiplexer Circuit

The multiplexer circuit is the heart of the Mux Card as it provides the actual switching between the four averaging cards. A schematic diagram is shown in Figure 3-38. As seen in the figure, the multiplexer switches four 32-bit busses into a single 32-bit bus. The multiplexer circuit is nearly identical to the multiplexer on the averaging card. In fact, the same PLD program is used for the EPM3128s on the Mux Card and the averaging card. The difference in these two multiplexers lies in the control circuitry, implemented as a single Altera PLD, *U3*.

A timing diagram of a data transfer operation is shown in Figure 3-39. The multiplexer operation begins with all averaging card FIFOs empty, oscillator U5 free running at 20 MHz, and the DIO card asserting the *ACK* signal (ready to transfer data). The clock signal is fanned out by U3 to the switching chips (U1 and U2) and the DIO card. The

clock is fanned out again by U1 and U2 to become the FIFO clocks that will eventually clock data from the averaging cards. When an averaging card has data to transfer it will lower its */OR* flag. The multiplexer controller will select that card and raise the *ENABLE* line to clock data out of the output FIFO. The *ENABLE* signals are repeated as signals *REQ[1:4]*. When a particular channel is selected, it's *REQ* signal will also be asserted on U1 and U2 along with the first word of data. The *REQ* signals are then switched through a separate 4:1 multiplexer located in U2 resulting in the *REQOUT* signal being output to the DIO card synchronous to the first word of data. The DIO card then latches 32-bits of data on each rising edge of the 20-MHz clock in which the *REQOUT* and *ACK* signals are asserted.

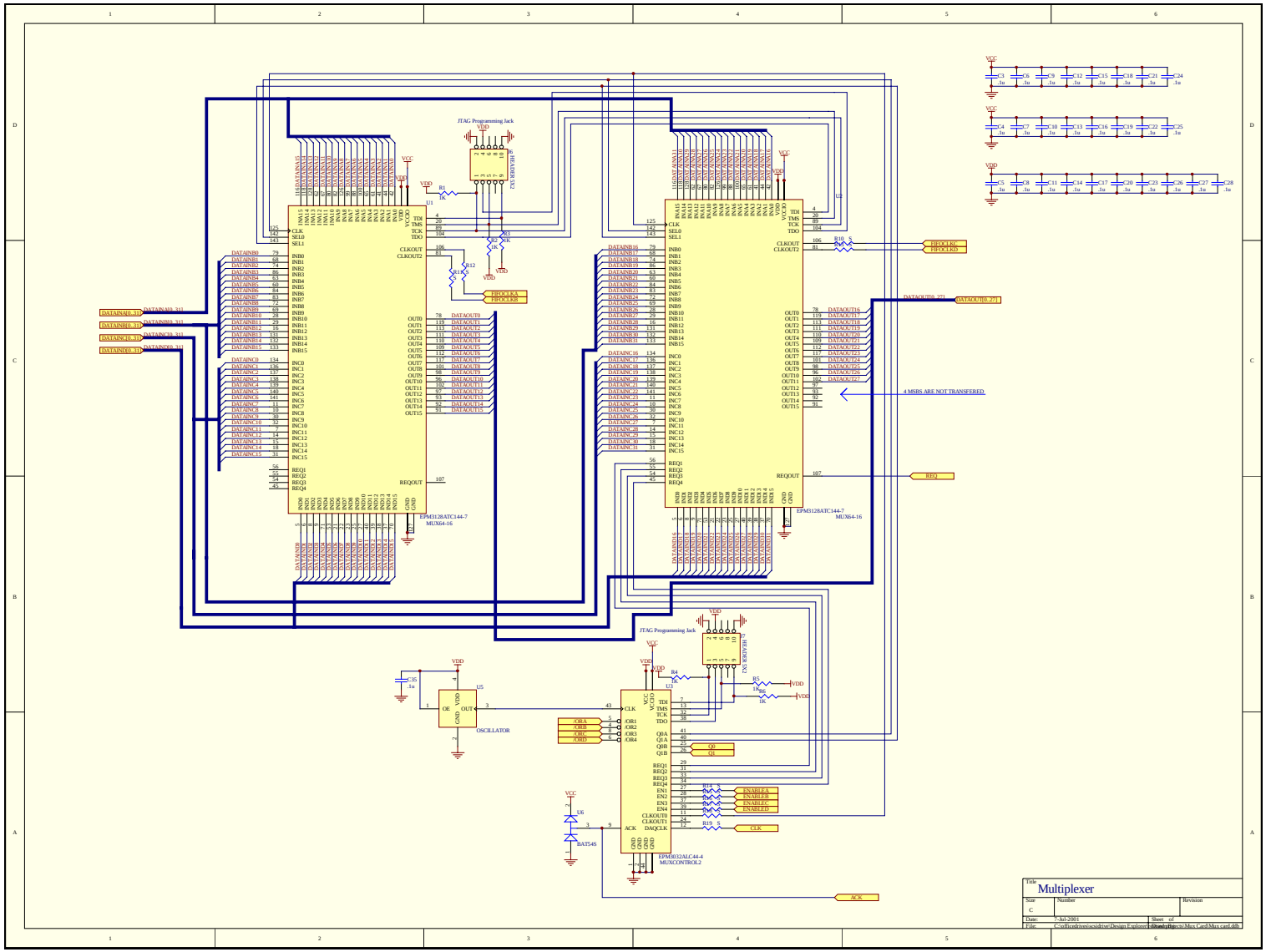


Figure 3-38 Multiplexer Schematic

If the DIO card input buffers become full before all data are transferred, it will deassert the *ACK* signal. When this signal is deasserted, the FIFO and multiplexer clocks are disabled thus freezing all data in the transfer pipeline. When the DIO card is ready for more data, the ACK signal is re-asserted and transfer continues exactly where it ceased. When the last word of data is transferred from the averaging card, the */OR* signal from the output FIFO is pulled high causing the REQOUT signal to de-assert after the last word of data is transferred to the DIO card. The multiplexer controller will then seek out the next Averaging Card that has data to transfer or it will freeze in it's current state if there are no more data to transfer.

From the schematic of Figure 3-38, it is evident that only 28-bits of data are transferred from the averaging card and the 4 MSBs are thrown away after the multiplexing stage. This is because 4 of the 32 DIO channel bits are used for control and error information (two bits indicate the current acquisition channel number and two bits indicate input and output FIFO overflow flags). This does not result in a loss of data because a maximum of 24 bits is expected from the Averaging Card for the case when a sample is always at a maximum (255) and the number of averages is set to 10,000 (an unreasonably high value). If the Averaging Card is ever used in a 12-bit system, only 28-bits are needed to perform 10,000 averages. Therefore the 32-bit output from the multiplexer card only contains 28 bits of waveform data and the upper four bits (of the 32) must be stripped in order to view the actual waveform in post processing.

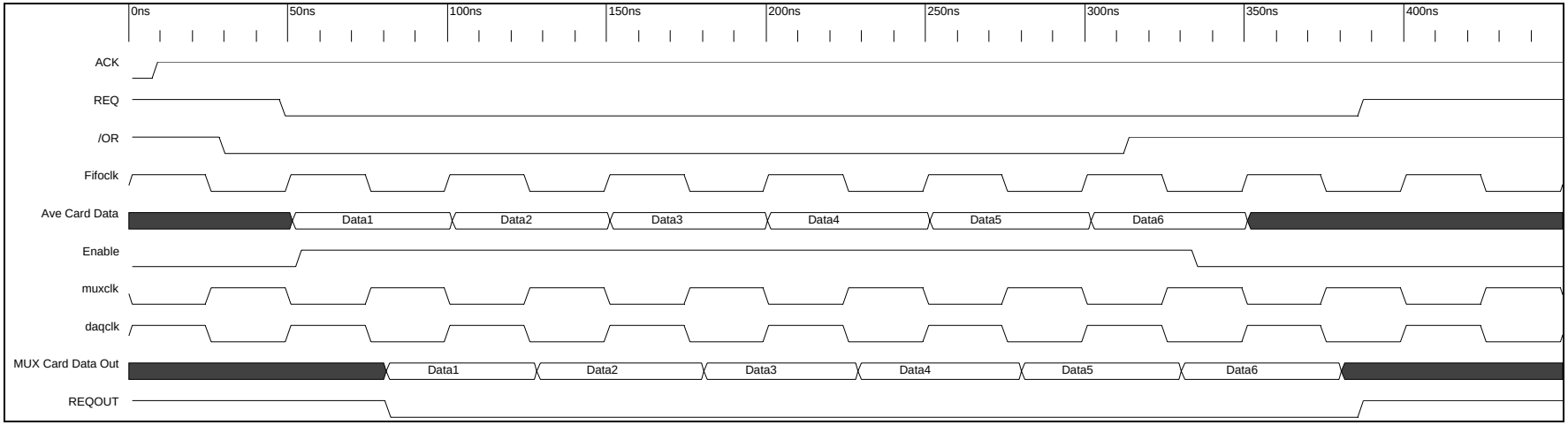


Figure 3-39 Mux Card Timing Diagram

Error Logic

The error logic serves the following two functions:

1. Drive the 8 Averaging Card error signals to LED indicators.
2. Combine those 8 signals into two signals for communication with the computer.

These functions were implemented as a single Altera PLD as shown in the schematics of Figure 3-40. As seen in the figure, four input FIFO full signals and four output FIFO signals are input into the circuit, one pair of signals from each Averaging Card. These signals are buffered and output as LED drivers through series resistors R20-R27. Connector CON16 is then used to attach a single row ribbon cable between the board and the LED panel. Each set of four inputs is also ORd together to create the INFULL and OUTFULL signals. These two signals are output to the DIO card and allow the computer to detect if an input or output buffer overrun has occurred.

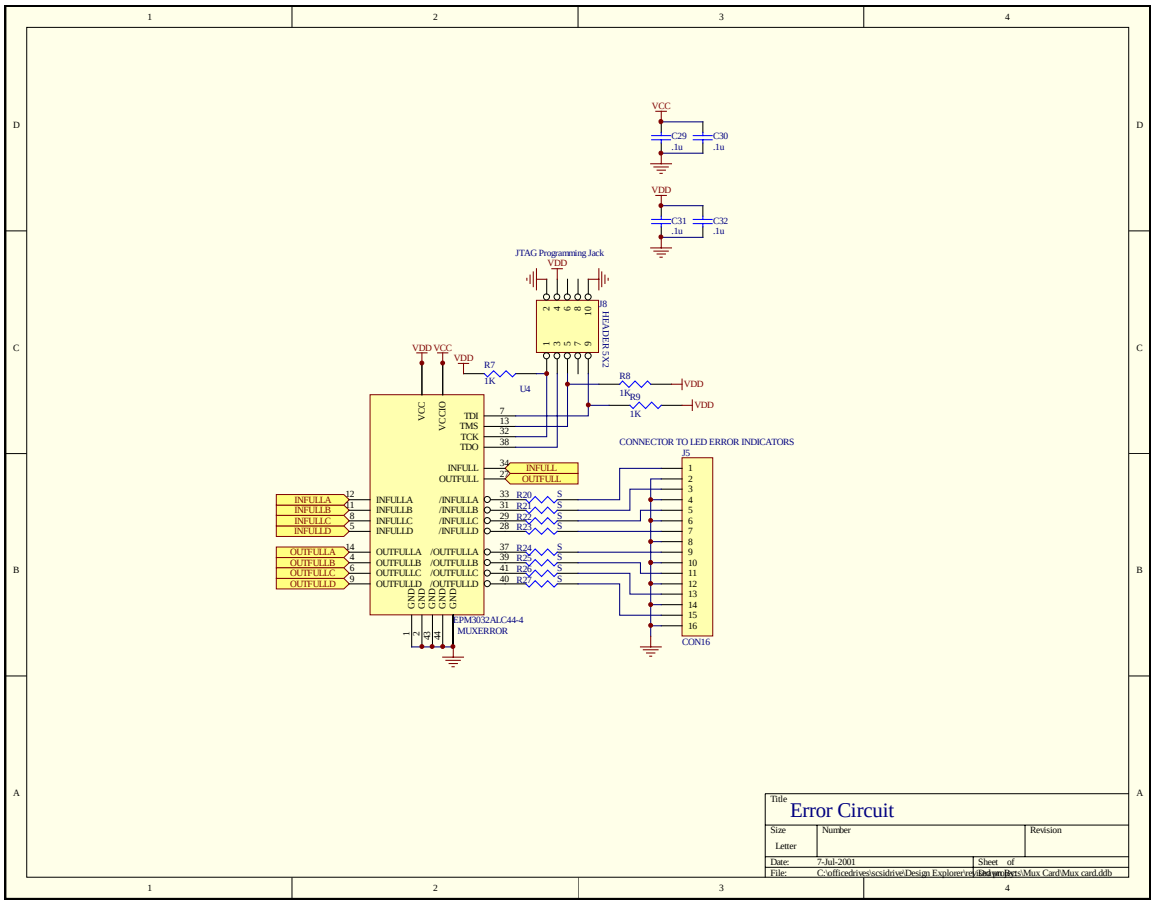


Figure 3-40 Error Logic Schematic

Output Connector

The purpose of the output connector is to connect the Mux Card directly to the DIO card. A reverse gender 68-pin SCSI cable and pinout was specified by the DIO manufacturer. The pinout for this is shown in the output connector schematics of Figure 3-41.

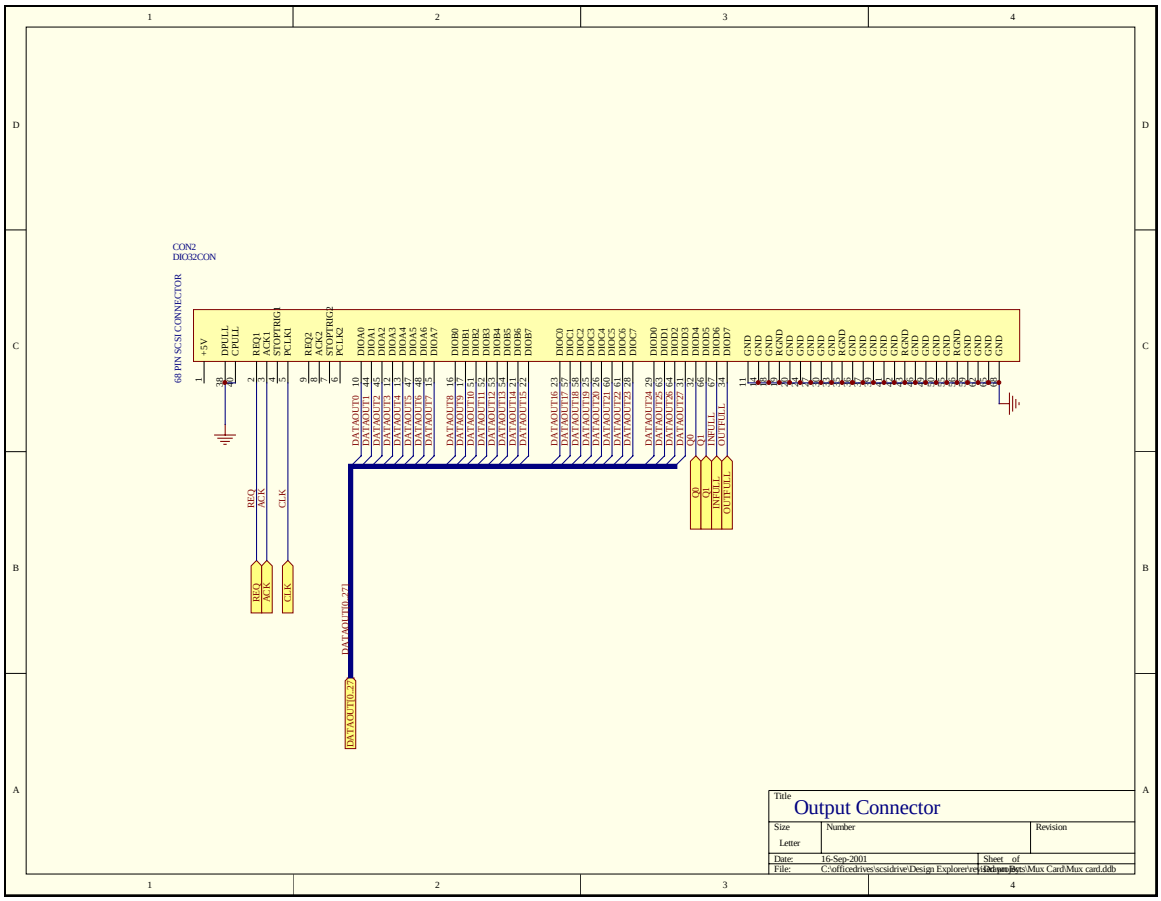


Figure 3-41 Output Connector Schematic

Physical Layout

The Mux Card was layed out using standard FR-4 with a standard 4-layer stackup. The board stackup is shown in Figure 3-42. It is the only board in this project that was simple enough to allow the PCB autorouter to route the entire board. As a result of using the autorouter and restricting it to 2 signal layers, the signal and via density is very high as seen in photograph of and the PCB layout of APPENDIX H.

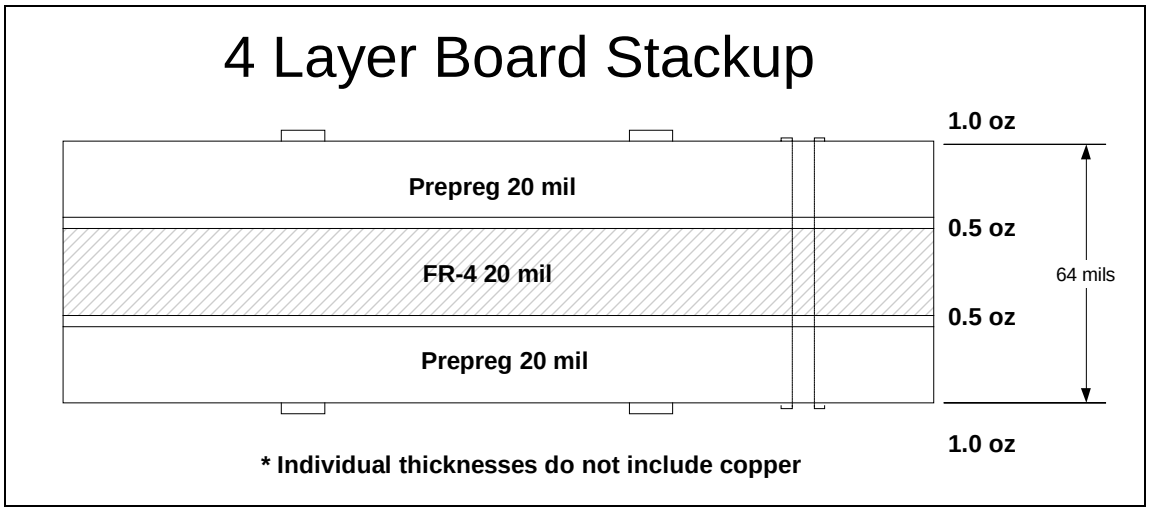


Figure 3-42 Multiplexer PCB Stackup

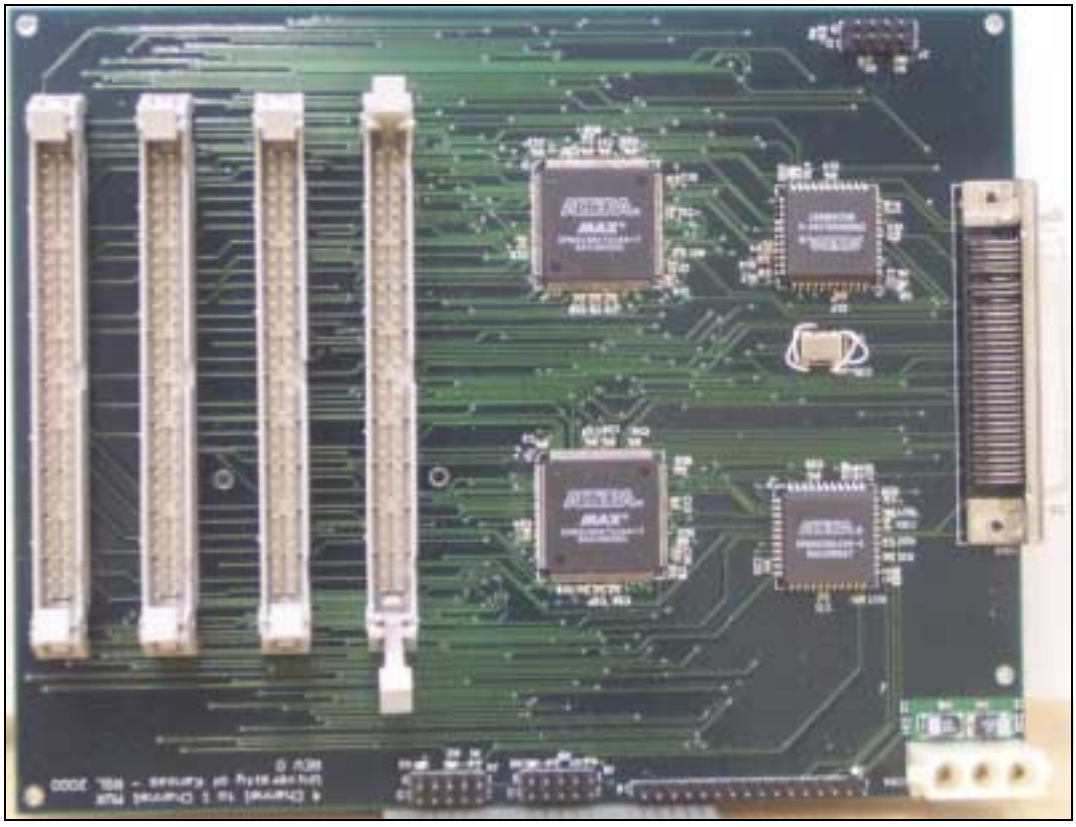


Figure 3-43 Multiplexer Card Photograph

Revision History/Future

This board has not been revised. Two minor problems were found during testing.

First, pins 6 and 9 on the output connector CON2 were shorted to pins 5 and 2 and these

must be left floating for the DIO card to operate properly. This is seen on the right side of the board photograph as the two pins that have been removed from the SCSI connector. The second error is that the oscillator package was drawn as a top view when the manufacturer supplied a bottom view. This was corrected with four jumper wires on the prototype and should be updated for future revisions.

3-5 Computer Interface

The purpose of the computer interface is to store the processed data and display a sample of the data every few seconds so that the operator can see what is being recorded. The interface consists of a personal computer (PC) running the Windows operating system, one DIO card, and custom data acquisition software.

The DIO card selected for this system is the National Instruments PCI-DIO-32HS. This board was selected for it's high speed operation (80 MB/s peak), 32-bit bus width, and PCI interface. Since the board is made by National Instruments, drivers and programming support are readily available to simplify programming. This pre-existing board greatly simplified the system design as the only hardware design requirement was to apply the signals with the proper pinout (shown in Figure 3-41).

The custom data acquisition software was written using National Instruments Labview software. This software provides a graphical programming environment designed specifically for instrument control and data acquisition. Since Windows and Labview drivers were supplied by the manufacturer the only programming required was to write the actual acquisition program. The graphical code for this is shown in 0 with a screenshot of the running program shown in Figure 3-44.

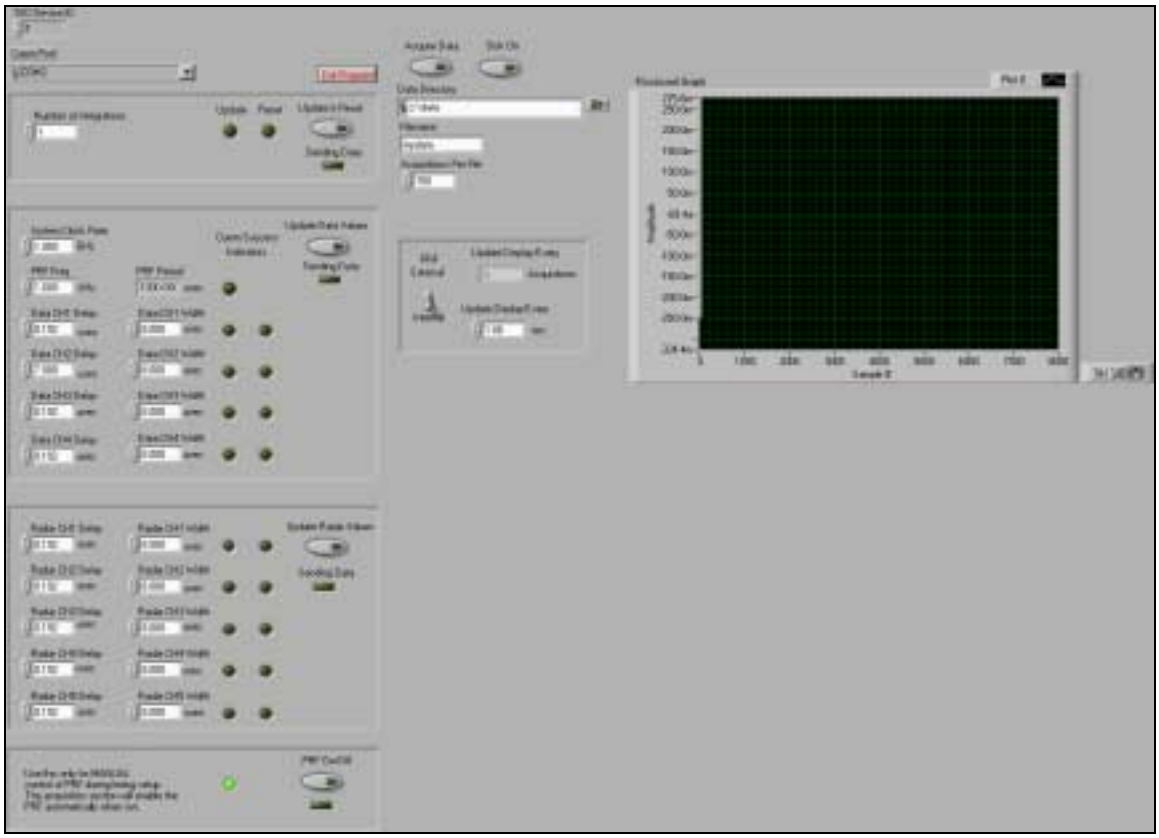


Figure 3-44 Acquisition Software Screenshot

The program is operated as follows. Upon starting Labview, the screen of Figure 3-44 appears and a signal is sent to the timing system to disable the PRF. The user then selects the PRF and timing values (width and delay) on the left side of the screen. This is divided into two sections with four channels and the PRF for the data acquisition timing and five channels for the radar analog timing. Once the values are entered, the user presses the "Update Values" button. The "Sending Data" indicator illuminates and the indicators to the right of each of the values will illuminate as each of the values is programmed in the timing system (via the RS-232 interface). The update operation takes approximately 2 seconds for each of the two sections.

The number of integrations is selected in the upper left pane. After the value is entered, the user must click the "Update and Reset" button for the new value to become

active. When this button is clicked, a special command is sent to the timing microcontroller which will pass the new integration number to the master Averaging Card. The reset signal is then sent to reset the integration count and clears all FIFOs in the Averaging Cards.

Other acquisition items include the data storage and screen update options seen in the center of the display. Data storage options include the data storage directory, the filename, and the number of acquisitions per file. The data directory may be typed in manually or selected via a standard windows "browse" window available by clicking the button at the right of the "Data Directory" box. Finally, the user chooses a screen update option. Normally, if the internal PRF generator is used, the user specifies the number of seconds between screen updates. If the external PRF signal is used, the user enters in the number of acquisitions to skip between screen updates. The updating is designed this way because the external PRF is normally used for single shot acquisitions and the repetition frequency is unknown therefore it does not make sense to update versus time.

When the user has all of the settings entered and the timing system has been updated, the system is ready to acquire data. The user clicks the "Acquire Data" button to initiate the process. Once acquisition is initiated, all settings are "grayed out" so they cannot be changed during acquisition. The software sends the reset command to the Averaging Card, then sends a command to the timing system to turn the PRF signal on. After a number of acquisition cycles (based on the screen update rate), one cycle of data is processed and displayed on the right of the screen. When the user is satisfied that all settings are correct and the output is acceptable to record, the "Disk On" button is pressed to activate disk storage. With disk storage activated, every byte of data entering the DIO

card is stored directly to disk with no processing. The screen graph will be updated at the rate specified for a live display of the acquisition data. When the user is finished acquiring data, they must de-activate the "Acquire Data" button and the screen returns to the original "start up" state.

During acquisition, a maximum number of acquisitions per file is specified so that long acquisition periods do not result in a single, excessively large data file. To accommodate this, a sequence number is always appended to the filename entered by the user. For example, if the user enters the filename "mydata" the first file generated in a new directory would be "mydata0000.dat". After the maximum number of acquisitions has been reached, a new data file is opened with the next sequence number. Following the example, the next file would be "mydata0001.dat". If acquisition is started in a non-empty directory and the filename is reused, the software will continue the pre-existing number sequence. The file size is calculated using the following relation:

$$File\ Size = N_s \ x\ N_f \ x\ B \tag{4}$$

where,

N_s = Number of samples acquired during an acquisition interval, samples
N_f = Number of acquisitions per file, no units
B = Storage bytes per sample, 4 bytes/sample

For an acquisition interval of 8,000 samples and 150 acquisitions per file the size of each file would be 4.8 MB.

Future Work

The acquisition software could not be fully completed due to the malfunctioning Averaging Card prototypes. Two key features have not yet been implemented. First, a data file header is needed to describe the acquisition settings. Without this, post

processing is nearly impossible because the number of samples in an acquisition cycle must be known to extract each line from the data. Second, the current graphing function assumes only a single data channel is used. An additional graphing window should be added for each additional acquisition channel along with code to send data to the correct graph.

CHAPTER 4 TIMING SYSTEM DESCRIPTION

4-1 Universal Timing System Overview

A special timing system was needed for the acquisition system due to the speed and precision required by the gigasample converters. The timing system has been specifically designed as a generic standalone module. By building the system in this way, the range of use is greatly extended as this can be used to provide timing on all radar systems in the foreseeable future. The requirements of this system are as follows:

- The system must accept an input clock frequency of at least 1 GHz.
- The system must accept the following input clock formats: AC coupled sine wave, single ended PECL, and differential PECL.
- The ADC card sync signal must be consistent and adjustable to within a 400 ps setup and hold "window" with respect to the 1 GHz clock.
- The system should have a "single shot" PRF input for taking raw ADC samples and to accept an external PRF signal.
- The system must be controllable from the data acquisition computer.
- The system should generate multiple divided versions of the input clock signal and multiple copies of the PRF signal to provide maximum flexibility in timing radar systems that have not yet been defined.

TTL technologies would be too slow with propagation delays too inconsistent to meet the strict timing requirement (400 ps setup and hold window) of the gigasample converter. Therefore, the system was built primarily using PECL technology components. These components operate at speeds in excess of 2 GHz with nominal propagation delays of 250-600 ps and a typical propagation delay variance of 100-200 ps making them ideal for this application. A block diagram overview of the entire timing system is shown in Figure 4-1.

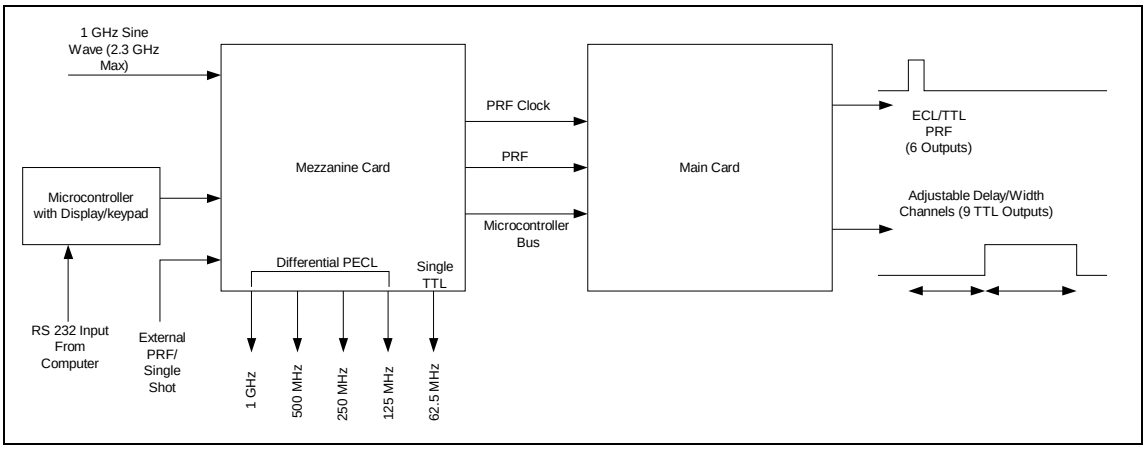


Figure 4-1 Timing System Overview

As seen in the figure, the system inputs a 1-GHz clock and outputs five divided versions of that clock as well as a PRF pulse and 9 copies of that pulse with adjustable delay and width. After some preliminary work it was obvious that the amount of circuitry required to support all of this functionality using PECL technology would not fit on a single PCB. Therefore, a natural break point was identified and the design was broken into two mating circuit boards, dubbed the Mezzanine Card and Main Card. In addition to these two custom circuit boards, a microcontroller is used to electronically control the PRF and the delay and width of the 9 PRF pulse copies. The microcontroller also provides an RS-232 link to the computer to give the data acquisition software control over the timing system. In addition to computer control, the system can be manually controlled via an LCD pushbutton panel that interfaces directly into the microcontroller.

About ECL Technology

The timing system is comprised almost entirely of emitter coupled logic (ECL) components. This is a special digital technology used in instances where speed and precision take priority over cost and power consumption. The key to ECL technology is that it is a "current steering" technology as compared to TTL and CMOS which are

"current switching" technologies. A simplified schematic of an ECL gate is shown in Figure 4-2.

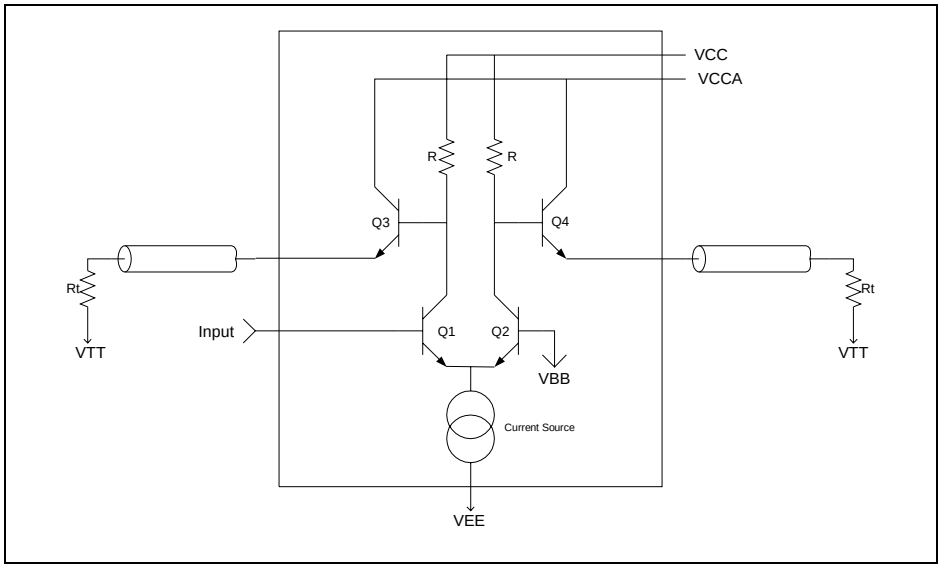


Figure 4-2 Simplified Schematic of an ECL Gate [13]

The "current steering" concept is seen in the figure as the current flow through the VCCA node is constant and a different path is taken for a logic high or logic low. This results in a significant reduction in power supply noise compared to current switching technologies. The output transistors are designed to always operate in the active region resulting in fast switching speeds at the expense of power consumption [14]. The power consumption is nearly constant with frequency (due to the current steering) with each differential output pair drawing approximately 24 mA of current (from a –2.0 V supply). Since the output drivers are single transistors with no internal DC biasing an external biasing/termination network is required before an ECL gate will produce any output. The two most typical biasing/termination methods are shown in Figure 4-3.

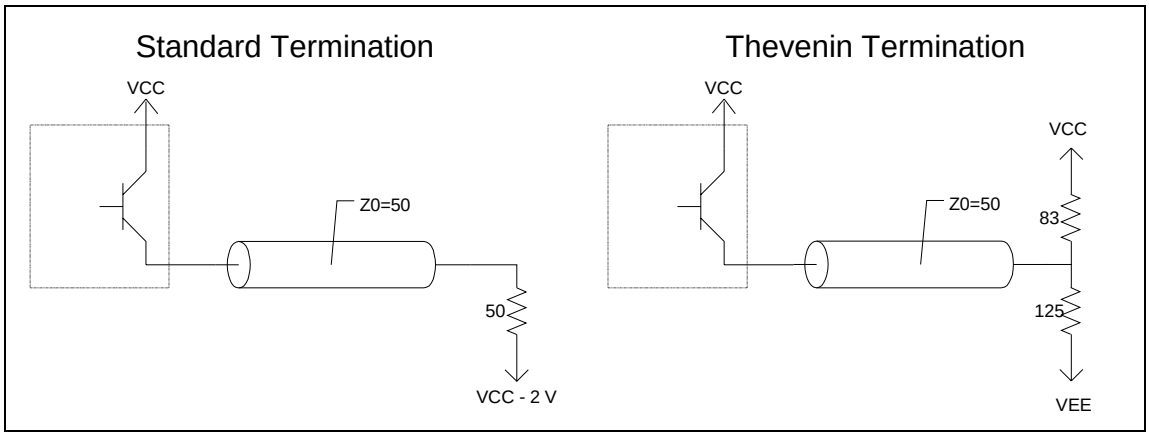


Figure 4-3 Typical Biasing and Termination Methods for ECL Gates

The standard termination scheme shown in the figure is preferred when an additional power supply is available to generate the special termination voltage. The Thevenin termination is used in applications where small amounts of ECL are used and a single supply is preferred. For the timing system, since many ECL devices were used, it was only natural to select the standard termination scheme and provide a dual power supply for the system.

Because the output transistor is connected directly to a supply and the quietist supply on a PCB is typically ground, traditional ECL circuits are designed by tying the VCC terminal to ground and using a VEE supply of –5.2 V as shown in Figure 4-4. Using this scheme the termination voltage is –2.0 V. In most systems, the ECL signals must be converted to TTL in order to interface with traditional circuitry (i.e., display, memory, microcontroller, etc.). Therefore, a minimum of three supply voltages is required, two negative supplies for the ECL components and one positive supply for the TTL components. In order to reduce these supply requirements, the ECL supplies can be shifted up by 5 volts to create positive ECL (PECL) as shown on the right side of Figure 4-4. The advantages of this are a reduction in the number of power supplies and the

ability to directly interface a TTL output into a PECL input. The disadvantage is that the VCC supply used by TTL will typically be noisier than ground and care must be taken to ensure the VCC plane does not become excessively noisy. Because few TTL signals were used in the timing system, excessive noise was not a concern and PECL was selected for this design.

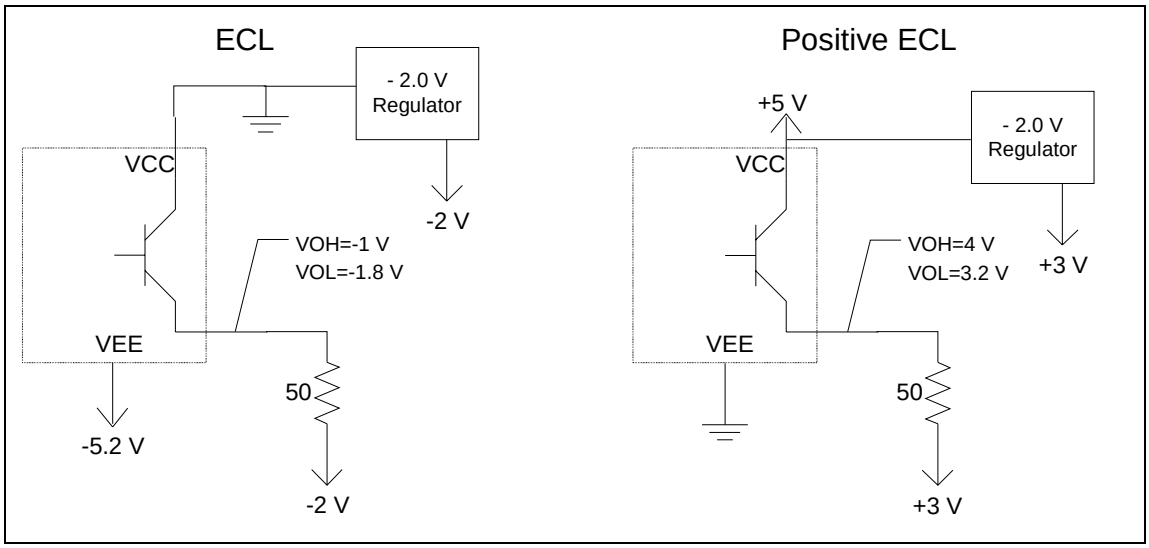


Figure 4-4 ECL and PECL Supply Requirements

The special termination voltage required by the standard termination scheme must be generated using a special type of supply. As seen on the right side of Figure 4-4, when the output transistor is conducting, the ECL output pin pulls the node toward +5.0 V. An ordinary +3.0 V supply is designed only for the load to pull towards ground and will not remain regulated if the load pulls toward a more positive voltage. A negative regulator is designed for the load to pull toward a more positive supply. Therefore, a –2.0 V regulator is connected to the +5.0 V supply to create the +3.0 V termination voltage for the entire system.

4-2 Timing System Mezzanine Card

Overview

The purpose of the Mezzanine Card is to input a clock signal at speeds exceeding 1 GHz, output 5 divided copies of that signal, and generate a PRF pulse based on one of those signals. In the case of the current radar, the input is a 1-GHz sine wave and the PRF pulse is based on the 5th division of that signal (62.5 MHz). A block diagram of the Mezzanine Card is shown in Figure 4-5.

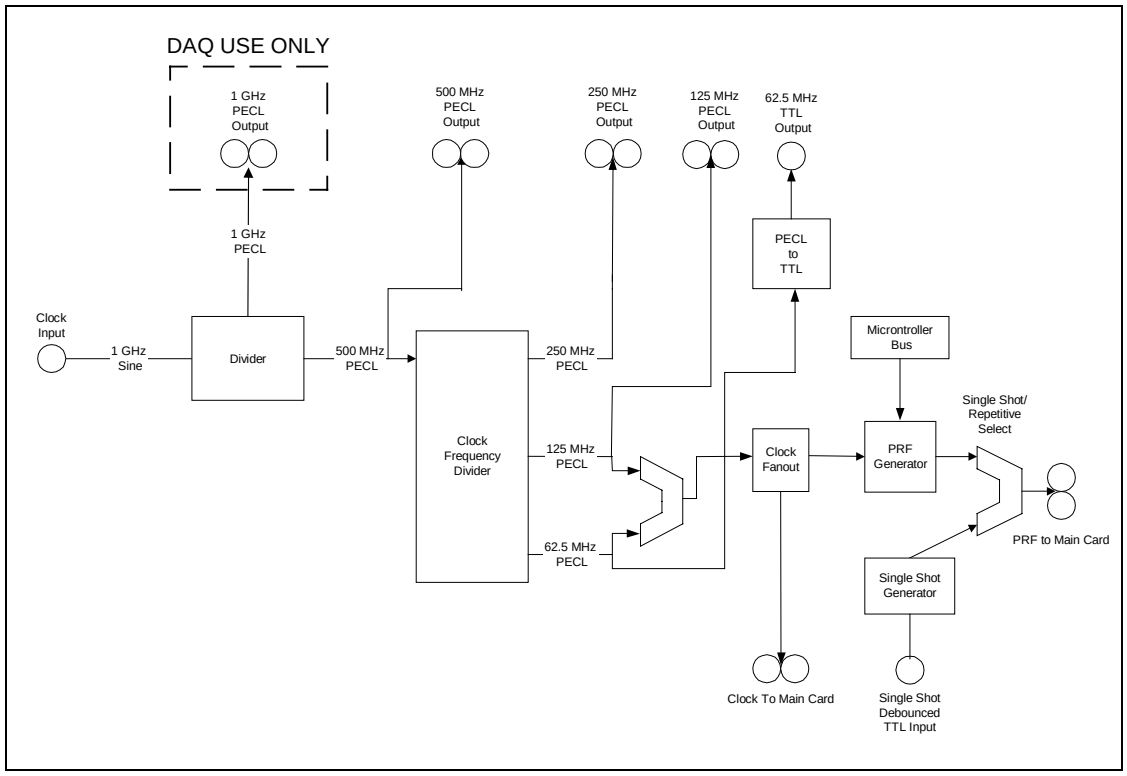


Figure 4-5 Mezzanine Card Overview

As seen in the figure, the 1-GHz input is divided into 500, 250, 125, and 62.5 MHz. The 4 higher frequencies are all output from the board as differential PECL signaling while the 62.5 MHz is output at TTL levels. A multiplexer allows selection of the 62.5 or 125 MHz signals for use as the PRF generator clock. This clock is input to the PRF generator circuit which outputs a PRF pulse at the frequency programmed by the

microcontroller. A final multiplexer selects between the internally generated PRF pulse or an externally generated signal that can be converted into a PRF pulse. This signal can be used for single shot acquisitions or for synchronizing to an external PRF signal.

Detailed Description

A top level schematic of the Mezzanine Card is shown in Figure 4-6 with a full version of the schematics shown in APPENDIX D. Circuit operation begins with the 1-GHz clock signal input into connector CON5 (note that input connector CON2 is for differential inputs and is currently unused). The 1-GHz clock is input into the GHz Divider section which then outputs a PECL level copy of the clock and two 500-MHz divided versions. The 1-GHz copy is output immediately as differential PECL signals through connectors CON5 and CON16. One 500-MHz clock is sent to downstream hardware while the other is sent through a trim delay section which allows adjustment of the clock edges down to 20 ps of accuracy. After the trim delay section, the 500-MHz signal is output as differential PECL signals through connectors CON6 and CON17. Note that differential signaling is used wherever possible on the Mezzanine Card to maximize speed and precision.

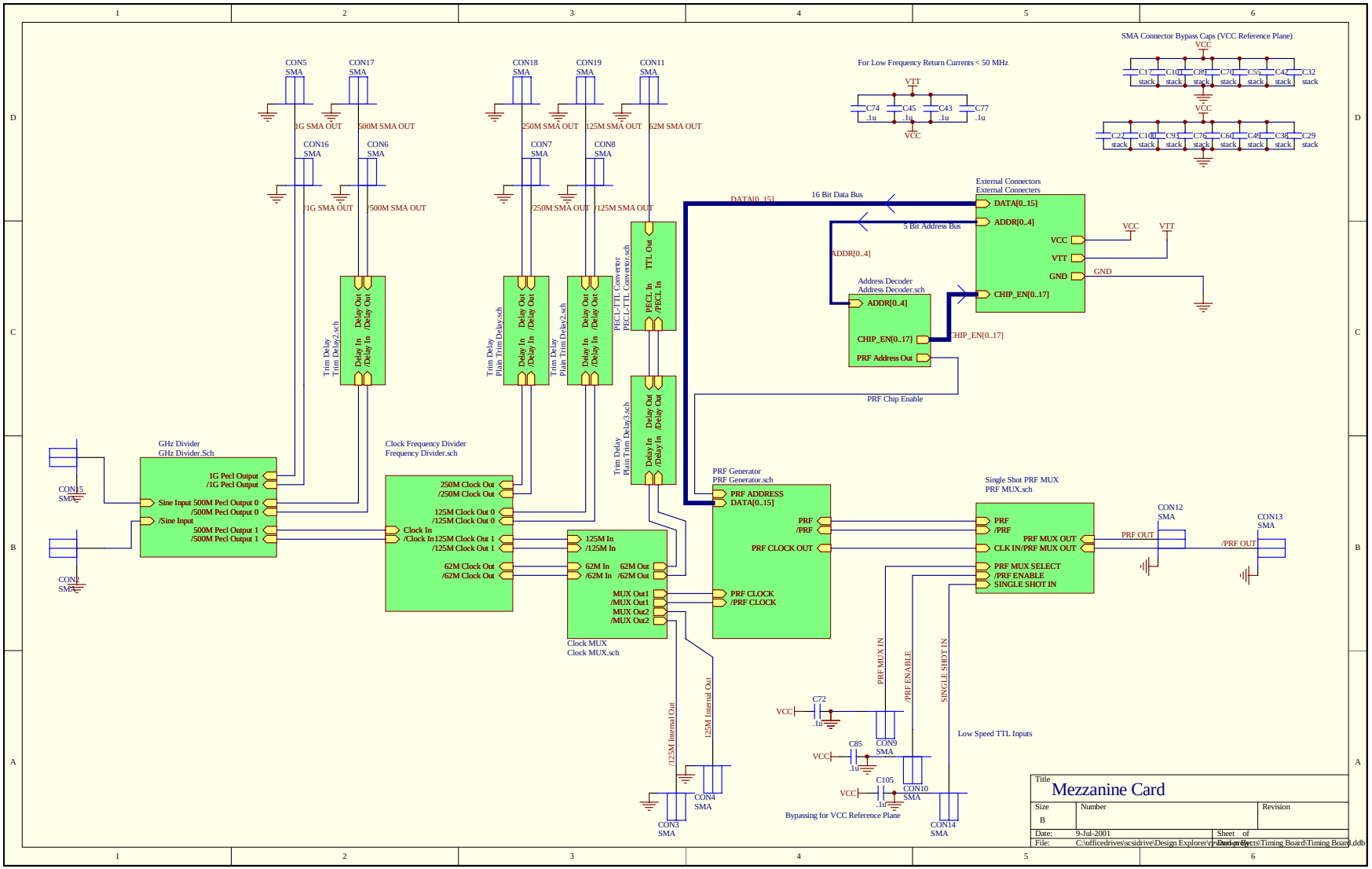


Figure 4-6 Mezzanine Card Top Level Schematic

The 500-MHz signal that remains on the board is then input into the Frequency Divider section. Here the frequency is divided three more times resulting in one 250-MHz clock, two 125-MHz clocks, and one 62.5-MHz clock. From that section, the 250-MHz and one 125-MHz signal are passed through individual trim delays and output as differential PECL signals through connectors CON7, CON8, CON18, and CON19. The remaining 125-MHz signal and the 62.5-MHz signal are input into the Clock Mux section.

The Clock Mux section is used to select the clock that will drive the PRF Generator section. Two copies of the selected clock are output. One copy goes to the PRF Generator and one copy is sent to connectors CON3 and CON4 to drive the Main Card. The Clock Mux section also outputs the 62.5-MHz signal where it passes through a trim delay and a PECL to TTL converter. The TTL version is then output from the Mezzanine Card through connector CON11.

An Address Decoder block and microcontroller connection operate in parallel to the clock sections discussed thus far. These two sections connect directly to the timing system microcontroller through a 16-bit data bus and a 5-bit address bus. The 5-bit address is decoded for the entire timing system in the Address Decoder section of the Mezzanine Card. The decoder inputs the 5-bit address and outputs 19 chip enable signals, 18 of which are passed to the Main Card and one that is used in the Mezzanine Card PRF Generator.

The PRF Generator section inputs the clock from the Clock Mux section along with a 16-bit word from the microcontroller and produces a PRF pulse with a frequency proportional to the 16-bit word. Since the PRF pulse is based on the clock input, the

period of that clock determines the resolution of the PRF as well as the frequency range that can be attained. A single chip enable signal is input from the Address Decoder section to latch the 16-bit word when it appears on the data bus.

Finally, the generated PRF signal is input into the PRF Mux section. This section is used to switch between the constantly generated PRF signal and an external PRF input. The ADC card timing constraint requires the PRF signal to always be synchronized to the system clock. Therefore, the external PRF input is synchronized in the PRF Mux section prior to switching. This section also adds the ability to enable and disable the PRF signal electronically. This PRF disable feature is used by the acquisition computer to disable acquisition until the Averaging Card has been reset and is ready for data. It could also be used to silence the radar while still maintaining system power. The final PRF signal is output from the PRF Mux section as differential PECL signals and is output from the board via connectors CON12 and CON13.

The remainder of this section will be a detailed description of each of the hierarchical blocks discussed thus far. For this discussion we will assume the input clock frequency is 1 GHz.

GHz Divider

The GHz Divider schematic is shown in Figure 4-7. The main clock input for the entire timing system is input as signals *Sine Input* and */Sine Input*. These signals enter a selection network that can be reconfigured (by soldering components) for three input signal types: single ended sine wave (800 mV p-p), differential PECL, and single ended PECL. The exact components required for each mode is shown in the schematic text, the result of which is shown in Figure 4-8. The input clock signal is then fanned out using a 1:4 signal driver (U8) to produce two differential PECL versions of the clock. The 1:4

signal driver was selected because the available 1:2 drivers did not have the *VBB* pin which is required for biasing single ended inputs. One 1-GHz copy is output from the block immediately and is sent directly to output connectors. The remaining copy is input into U5 which is a special divide-by-two chip. This special chip was selected over a flip-flop because it operates at the highest frequency in the system and a flip-flop would cause additional output loading as external feedback would be required to perform a divide by two. The output of U5 is then a 500-MHz signal which is input into a 1:2 fanout driver so that the module will output one copy for driving off of the PCB and one copy of further use on board.

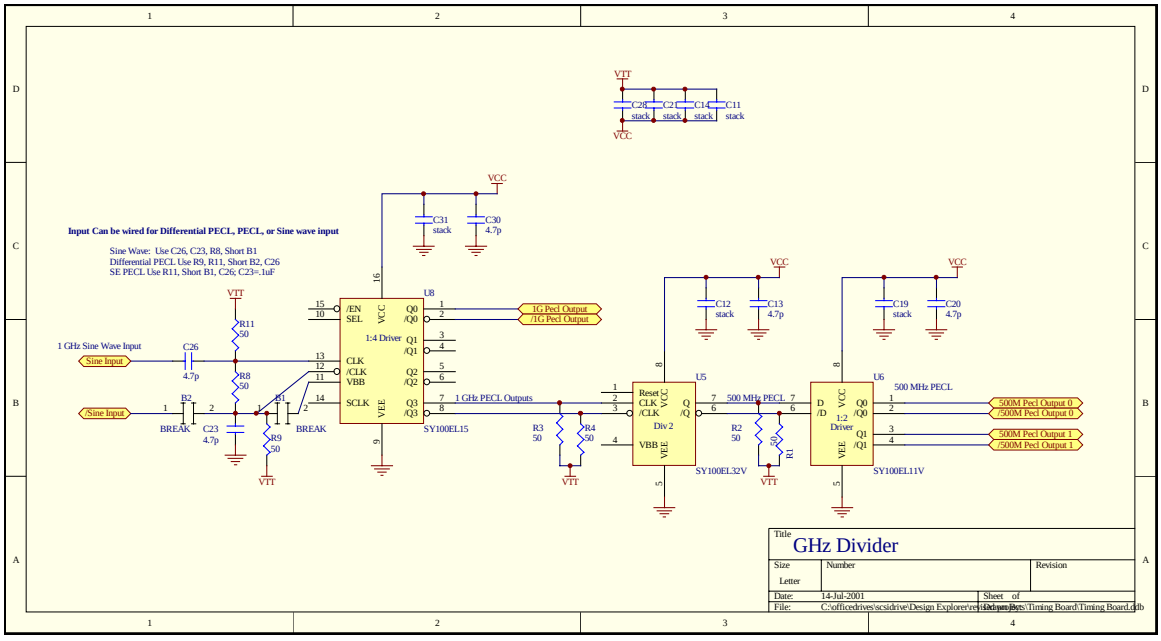


Figure 4-7 GHz Divider Schematic

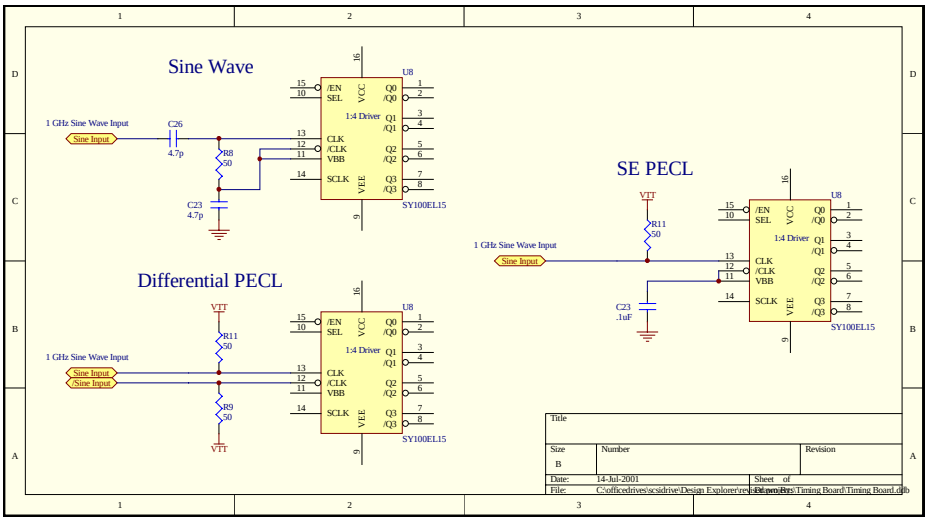


Figure 4-8 Mezzanine Card Clock Input Configurations

Note that in all Mezzanine card schematics, the bypass capacitor values are specified as “stack”. This is used to denote that a stack of two capacitors (1000 pF and 22 pF) was used to increase the bypass bandwidth to help achieve maximum operating frequency for this board. This stacking is necessary because all capacitors have multiple self-resonant frequencies where the impedance is substantially higher than that of an ideal component. A comparison of the impedance characteristics of two 1000 pF capacitors is shown in Figure 4-9. The American Technical Ceramic (ATC) 700a series capacitor was selected for the Mezzanine Card due to the excellent frequency response shown in the figure.

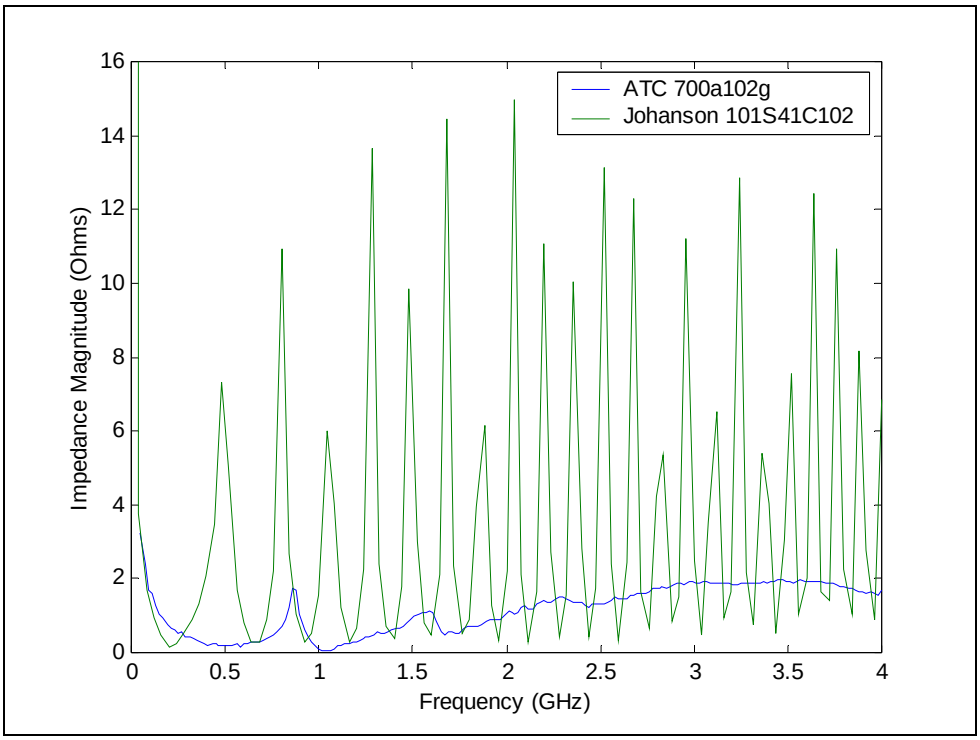


Figure 4-9 Comparison of Standard and RF Capacitor Frequency Response

Although the bypass capacitor stacks are only necessary on the first few stages of the frequency divider structure, they were used throughout the design to maximize precision and reduce noise.

Frequency Divider

The Frequency Divider schematic is shown in Figure 4-10. From the figure, the 500-MHz clock signal is input as signals *Clock In* and */Clock In*. This then drives Flip-Flop U4 which is configured as a divide-by-two circuit. The 250-MHz output of U4 is then fed back to its input and daisy chained to U7 and on to one additional device outside the module. Flip-Flop U7 is configured in the same fashion and outputs a 125-MHz clock. The 125-MHz clock is input into flip-flop U10 which divides the signal one last time to achieve a frequency of 62.5-MHz. The 62.5-MHz and 125-MHz signals must both drive the Clock MUX and Trim Delay sections downstream. If the 125-MHz clock

were daisy chained, it would therefore be driving four loads which was thought to be excessive. Therefore a 1:2 fanout driver (U12) was added so that the worst case fanout on any of these critical clocks is limited to three devices.

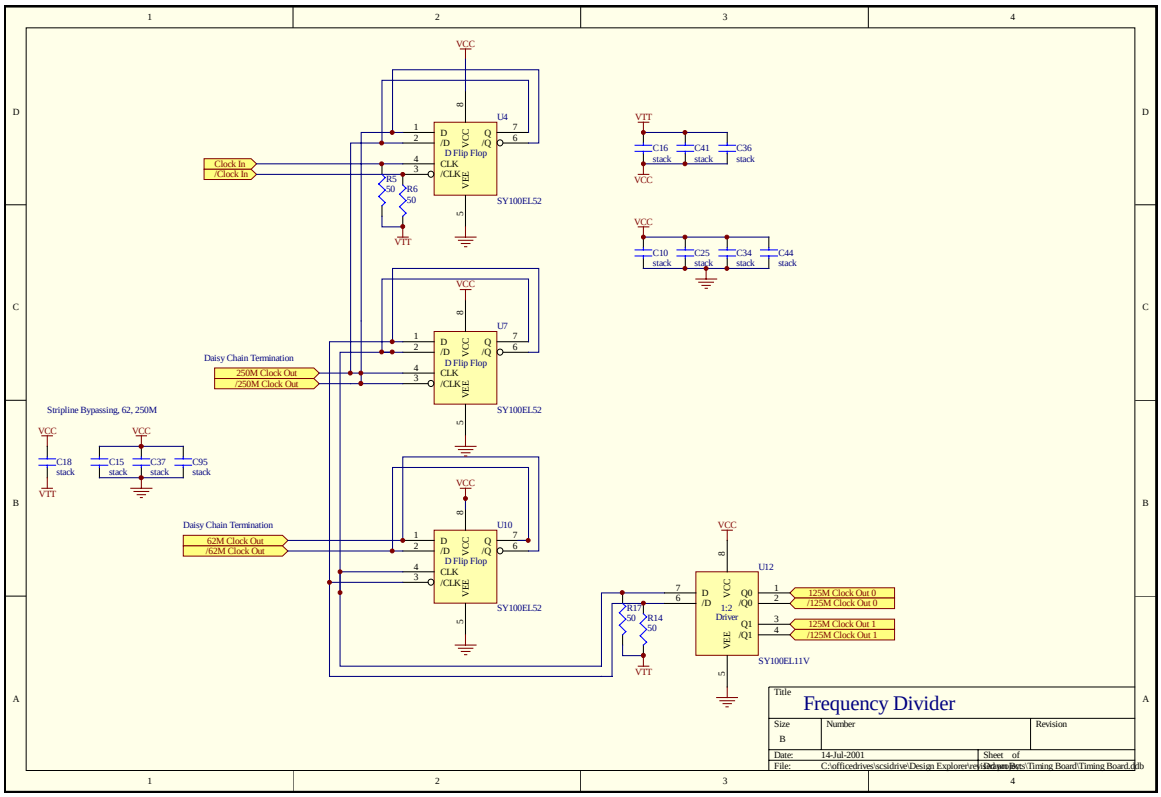


Figure 4-10 Frequency Divider Schematic

Clock Mux

The Clock Mux schematic is shown in Figure 4-11. From the figure it is seen that all functionality in this block comes from the 4:1 multiplexer chip, U11. Only two of the four inputs of the multiplexer are used to switch between the 125-MHz and 62.5-MHz clocks. Note that unused inputs in any of the PECL devices of this design are internally pulled up/down and may be left floating without the possibility of self oscillation. Jumper J2 connects to the select input of the multiplexer and is used by the operator to select between the two clocks. The selected clock is output to a 1:2 fanout chip so that two copies of the clock are available on the board.

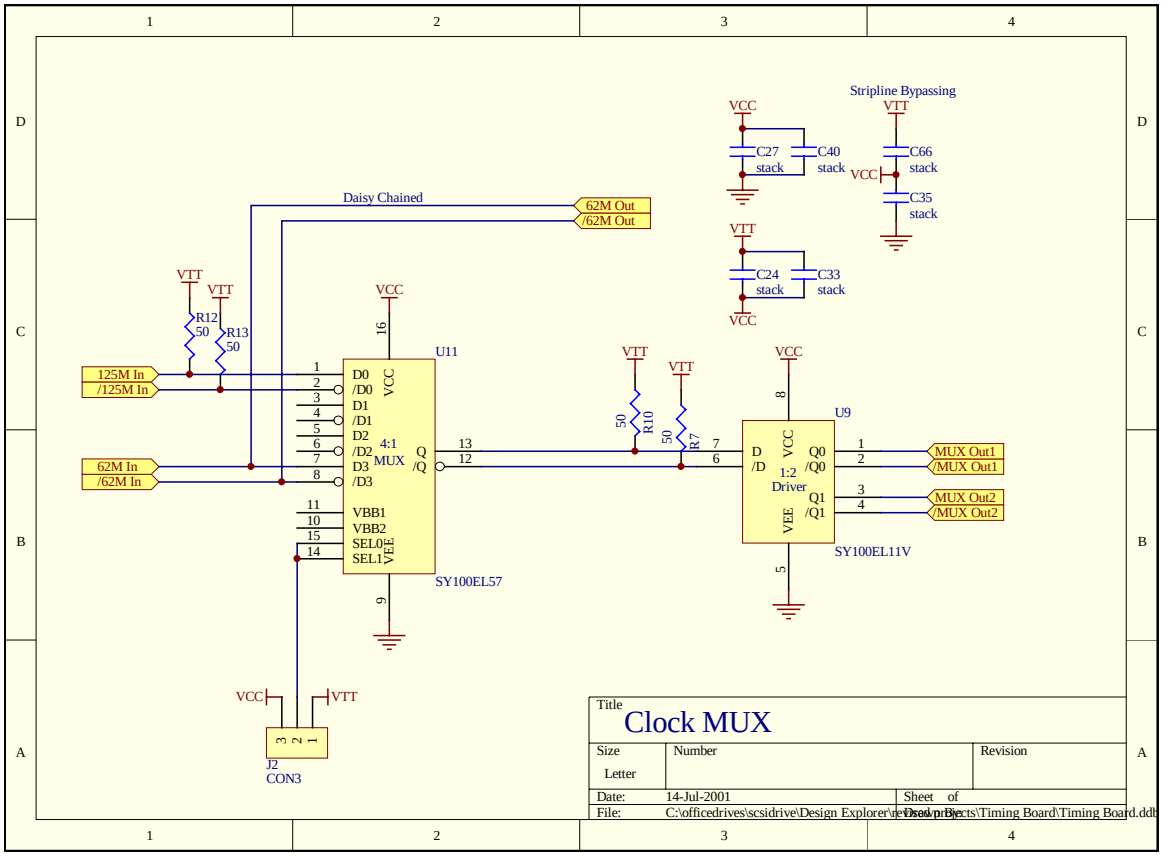


Figure 4-11 Clock Mux Schematic

PRF Generator

The PRF Generator is the heart of the entire timing system. This circuit generates a narrow pulse with a repetition frequency of a few kilohertz that is synchronized to the GHz clock. A schematic of the PRF generator circuit is shown in Figure 4-12 with a timing diagram shown in Figure 4-13. Analysis of the circuit begins with the circuit in a steady state condition (the circuit will reach this state within a few milliseconds regardless of the startup state of the counters and flip-flops). In the steady state, the PRF data from the microcontroller are already loaded into latches U14 and U18 and is constantly asserted onto the inputs of counters U15 and U17. Also, the /PRF signal will be high thus disabling the parallel loading function of counters U15 and U17. The two 8-

bit counters are chained together to form a 16-bit counting structure with a 16-bit parallel load. With the parallel loading function disabled and counting enabled on U15 the counters will increment upon each rising edge of the input clock (assume to be 62.5 MHz).

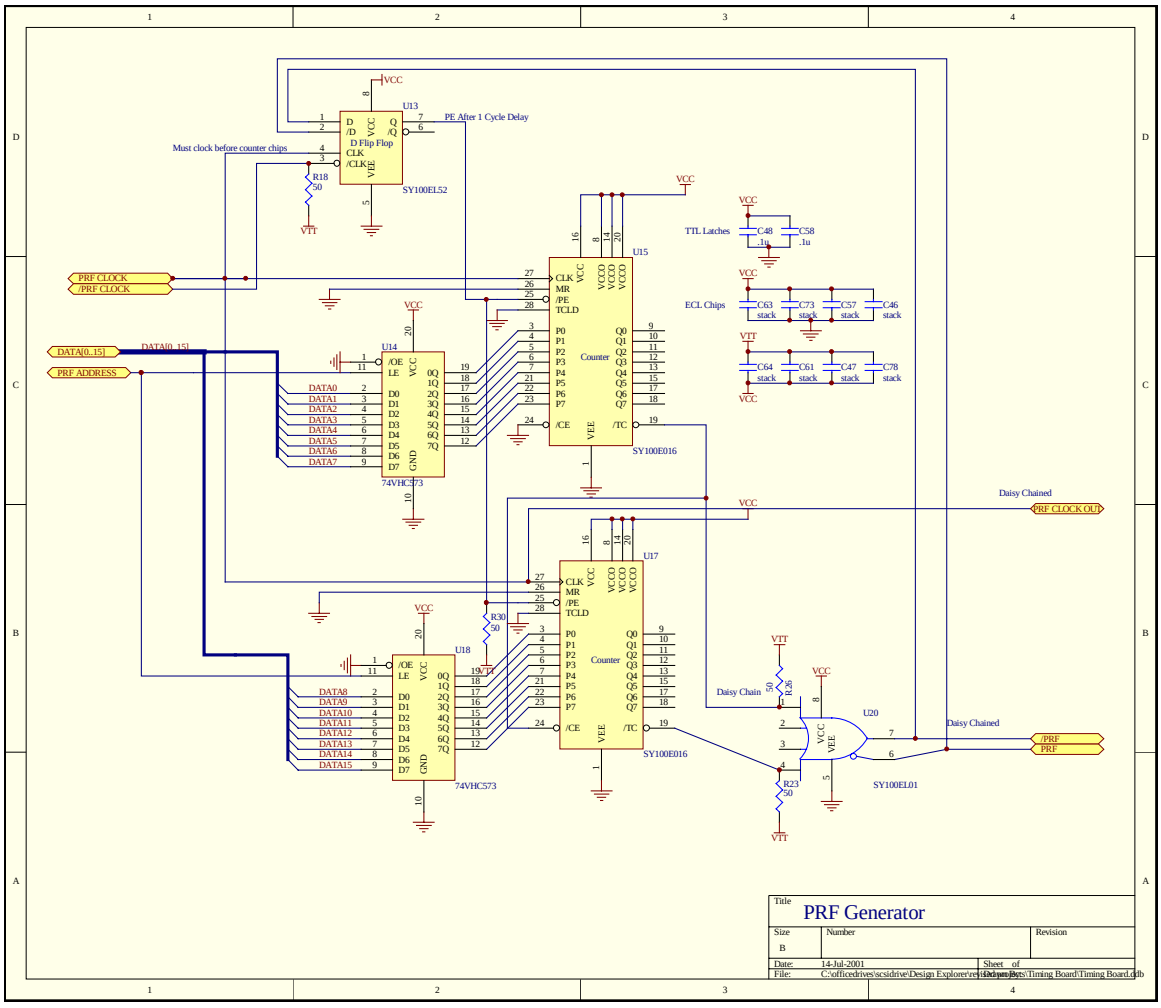


Figure 4-12 PRF Generator Schematic

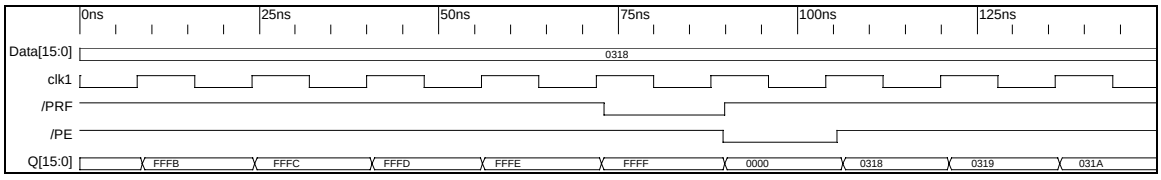


Figure 4-13 PRF Generator Timing Diagram

As the LSB counter increments, it will eventually reach the all ones state. When this happens it's /TC output will become low for a single clock cycle enabling the MSB counter to increment by one and asserting the /TC signal onto the OR gate, U20. Finally, when the MSB counter reaches the all ones state, its /TC signal will be asserted onto the OR gate for 255 clock cycles at which time the LSB counter's /TC will be asserted for a single clock cycle. When both /TC signals are asserted, the output of the OR gate (PRF and /PRF) will toggle for one clock cycle. At the start of the next clock cycle, the counters roll over to the all zeros state. During the rising edge of that clock, the /PRF signal will be asserted onto the counter parallel enable inputs since it is delayed by one clock cycle through the flip-flop U13. On the next rising edge of the clock the counters will load the parallel data onto their outputs. A few hundred picoseconds later, the flip-flop output will toggle releasing the parallel load signal. With the count values loaded into the counters and the /PE signal de-asserted, the cycle starts all over and the counters run from the loaded value to 0xFFFF. Timing analysis of this circuit (including trace delays) results in a maximum operating frequency of around 400 MHz. Since the current maximum operating frequency is 125 MHz, the circuit is operated well below it's maximum speed.

The pulse repetition interval for the PRF generator is calculated using the following relation:

$$PRI = (65{,}537 - N)\ x\ T \tag{5}$$

where,

N = base 10 equivalent of 16-bit counter load value, no units
T = period of the generator clock, seconds

With a 62.5-MHz source clock the minimum PRF of 953.6 Hz is achieved by asserting all zeros into the counter parallel load. The maximum PRF is 32.25 MHz, although in real world radar systems the maximum PRF would be limited to tens of kilohertz. Note that increasing the generator clock frequency results in finer frequency resolution but raises the lowest PRF attainable. For example, with the 125-MHz clock the minimum PRF increases to 1.907 kHz but the PRI resolution is 8 ns instead of 16 ns.

PRF Mux

The *PRF Mux* section selects between internal and external PRF signals. A schematic diagram of the section is shown in Figure 4-14. The schematic is comprised of two major sections. The first section is U23 and U24 which are used to synchronize the external PRF signal, *SINGLE SHOT IN*, to the PRF clock. The second section consists of U22 which is the actual multiplexer chip that selects between internal and external PRF signals.

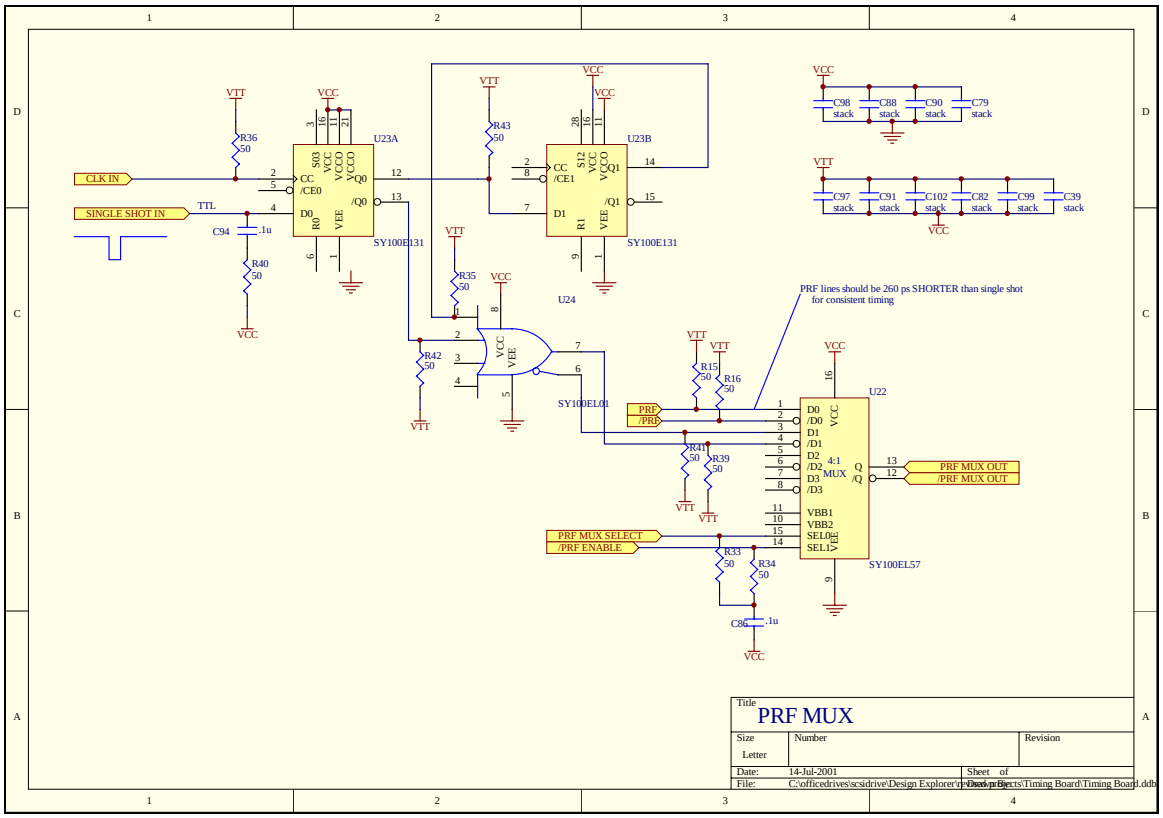


Figure 4-14 PRF Mux Schematic

The circuit operates as follows. The PRF clock is driving the two flip-flops of U23. When the PRF input changes from low to high, the synchronizing circuit will output a single pulse with width equal to one cycle of the PRF clock. This synchronized external pulse and the internally generated PRF signal both appear at the inputs of the multiplexer, U22. When */PRF ENABLE* is low one of the two PRF signals will pass through the mux and become the PRF signal that drives the entire system. The *PRF MUX SELECT* signal is connected to a switch on the front panel and selects which of the two PRF signals is used. The */PRF ENABLE* signal is connected to the microcontroller and is used to mute the PRF signal output for the entire system.

Trim Delay

The trim delay is used to trim the edges of various output signals so they can be accurately aligned with respect to the 1-GHz clock. A schematic of a typical trim delay is shown in Figure 4-15. As seen in the figure, the circuit consists of a 7-bit programmable delay IC and a set of dip switches. The switches select the delay in powers of two with the LSB adding 20 ps and the MSB adding 1.1 ns. The overall delay range is 2 ns with 20 ps steps. Note the terminator and series capacitor option on the trim delay outputs. These appear in the trim delay schematic because it is used as the coaxial cable driver that outputs the signals off board. On the current board, the capacitors have been replaced with 0 Ω resistors and the resistors left open but the pads are in place so that a capacitively coupled PECL output could be easily configured in the future.

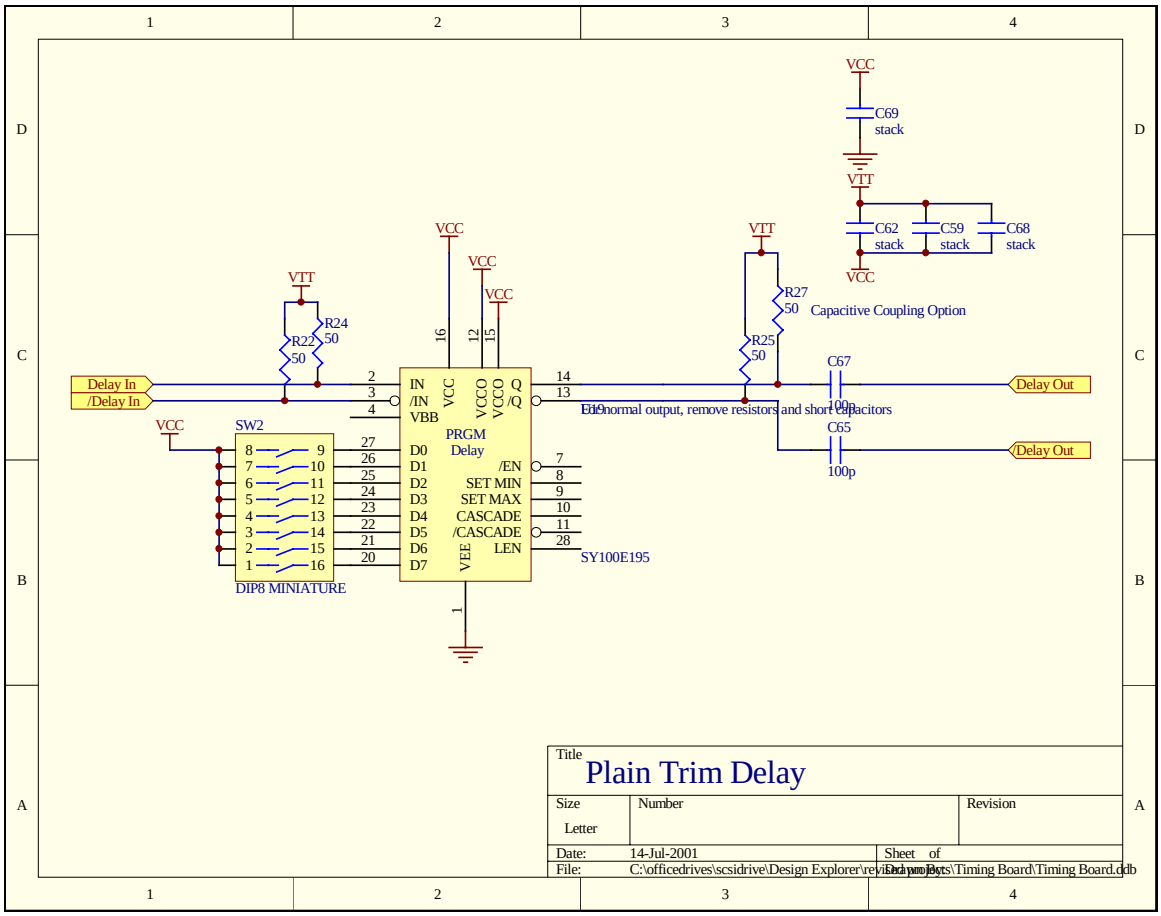


Figure 4-15 Trim Delay Schematic

Address Decoder and Microcontroller Connection

The timing system microcontroller transmits the timing information to the two PCBs via a 16-bit data bus and a 5-bit address bus. The Mezzanine Card contains a decoder network that converts the 5-bit addresses into 24 chip enable signals for the entire system. Of the 24 chip enable signals, one is used on the Mezzanine Card to latch the PRF control data, 5 are unused, and the remaining 18 are passed on to the Main Card. A memory map of the timing system address bus is shown in Table 4-1 with a schematic of the address decoder circuit shown in Figure 4-16.

Address	Device
0x00	PRF Generator
0x01	DAQ Channel 1 Delay
0x02	Radar Channel 1 Delay
0x03	DAQ Channel 2 Delay
0x04	Radar Channel 2 Delay
0x05	DAQ Channel 3 Delay
0x06	Radar Channel 3 Delay
0x07	DAQ Channel 4 Delay
0x08	Radar Channel 4 Delay
0x09	Radar Channel 5 Delay
0x0A	DAQ Channel 1 Width
0x0B	Radar Channel 1 Width
0x0C	DAQ Channel 2 Width
0x0D	Radar Channel 2 Width
0x0E	DAQ Channel 3 Width
0x0F	Radar Channel 3 Width
0x10	DAQ Channel 4 Width
0x11	Radar Channel 4 Width
0x12	Radar Channel 5 Width
0x13-0x1F	Not Used

Table 4-1 Timing System Address Map

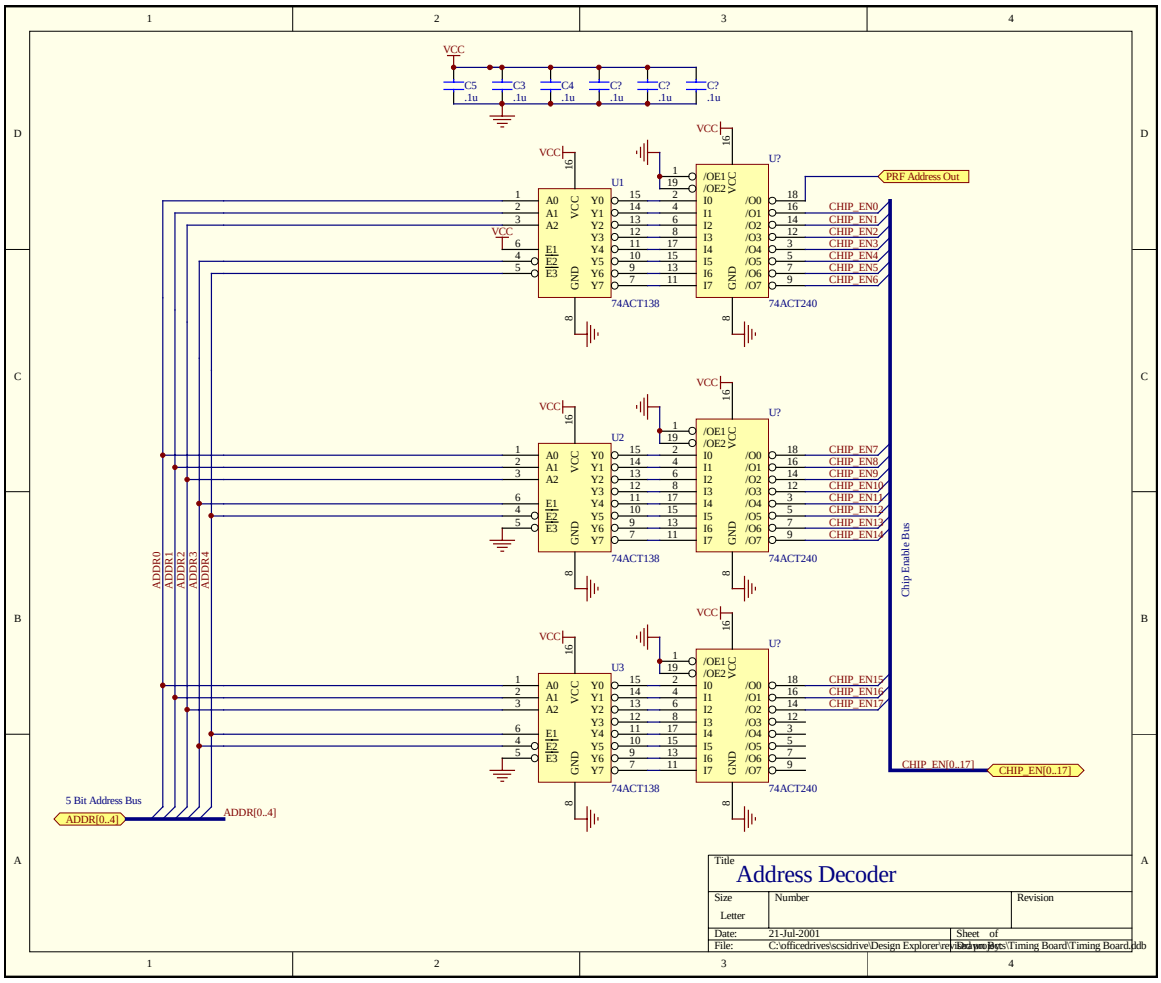


Figure 4-16 Address Decoder Schematic

The microcontroller data and address busses interface with the Mezzanine Card through a standard 26-pin 50-mil ribbon cable. The pinout for this is seen on connector J1 of Figure 4-17. From the figure, it is seen that the microcontroller data bus is also passed directly to the Main Card through J4. The decoded chip enable signals are passed to the Main Card through J3. In addition to the microcontroller bus electrical connection, connectors J3 and J4 are aligned on the Main and Mezzanine cards to provide a mechanical connection to tie the two cards together.

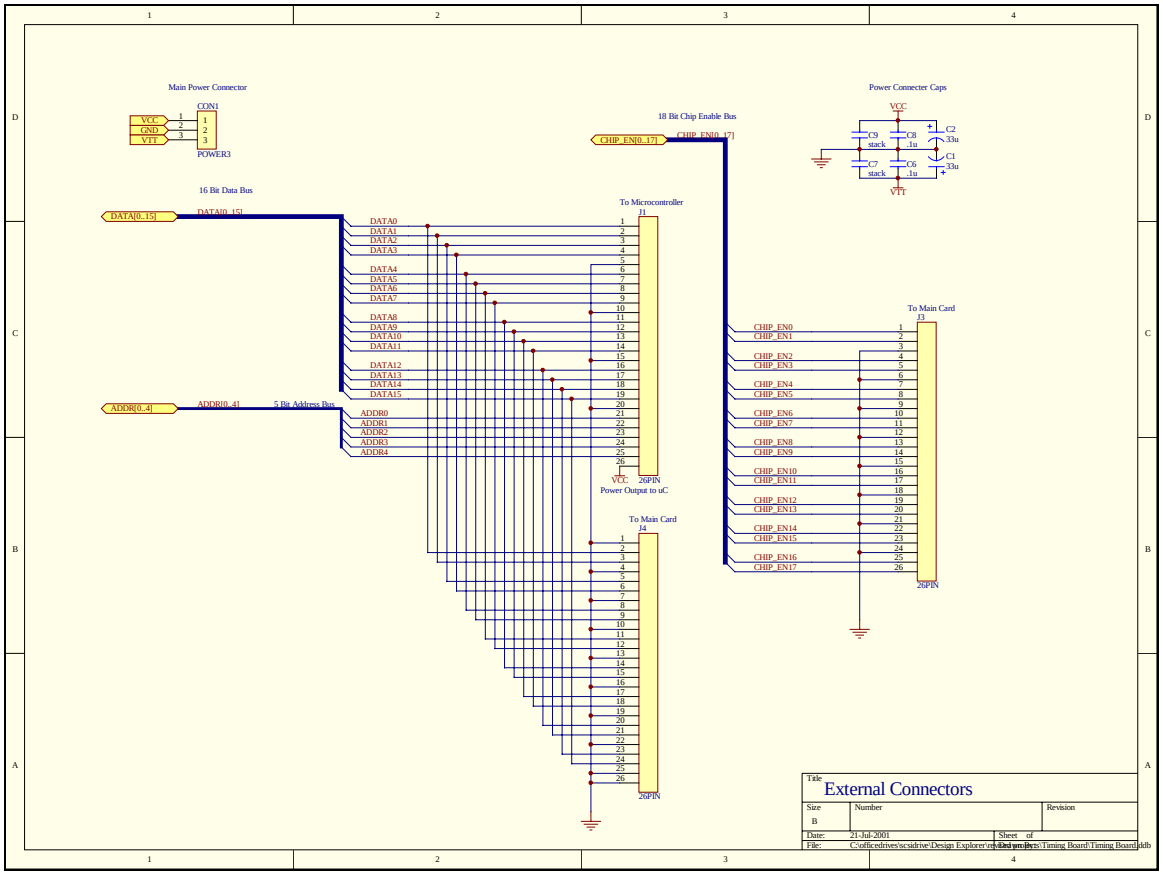


Figure 4-17 Mezzanine Microcontroller Connection Schematic

Mezzanine Card Physical Description

The Mezzanine Card was constructed as a 6-layer PCB. Rogers 3000 series high frequency, low loss board material was selected due to the high frequency of the input

clock circuitry. A board stackup is shown in Figure 4-18 with a board photograph in Figure 4-19.

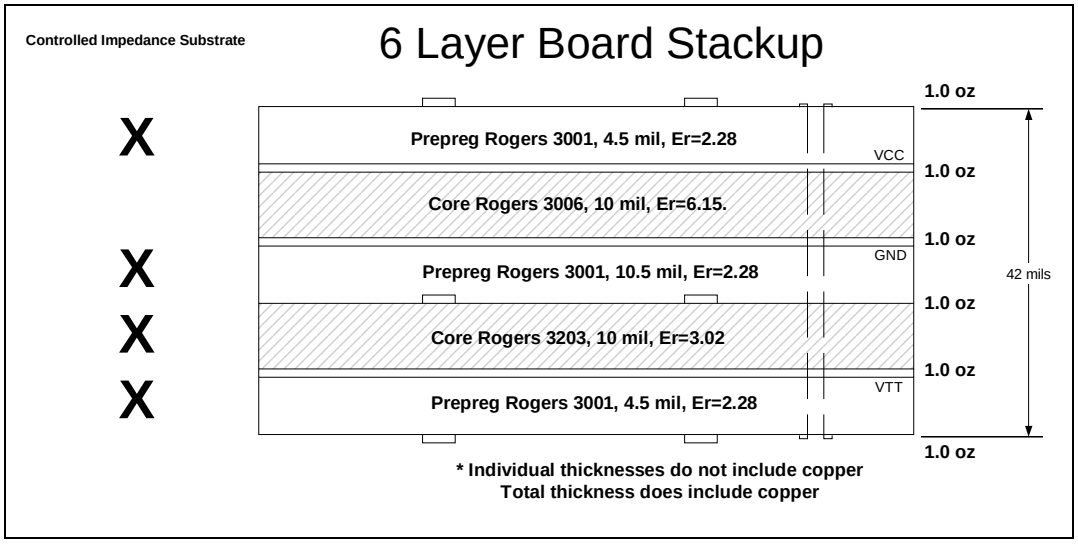


Figure 4-18 Mezzanine Card Board Stackup

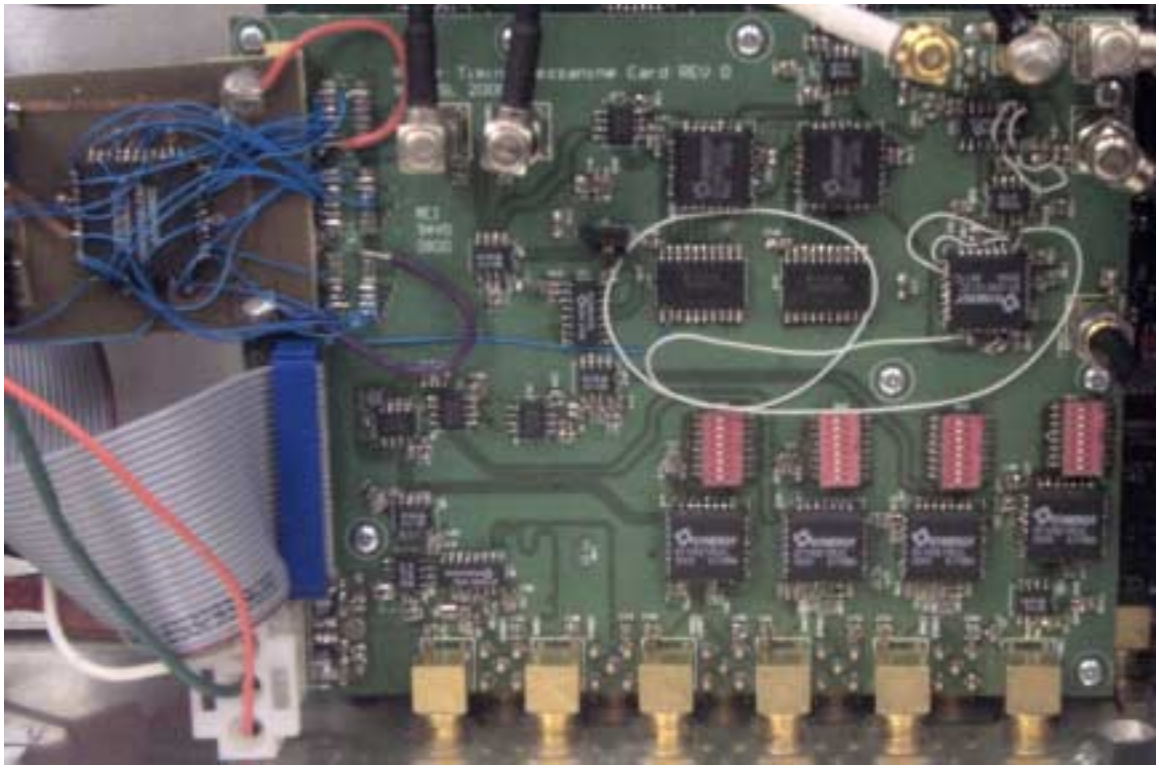


Figure 4-19 Mezzanine Card Photograph

The top layer routing of the input clock section is shown in Figure 4-20 with a complete board layout shown in APPENDIX I. As seen in the figure, the individual lines of the differential pairs are kept equal in length by "meandering" the shorter lines. Special pads are used to match the transmission line width to the pad width in order to reduce signal reflections when a signal passes beneath an input pin. These special pads are seen on pins 6 and 7 of *U6* in the figure. Additionally, each bypass capacitor connects to the internal layers with three adjacent vias placed directly on the capacitor pads to reduce via and trace inductance. These details, combined with the low loss board material, enable the input section to operate with an input clock frequency in excess of 2 GHz.

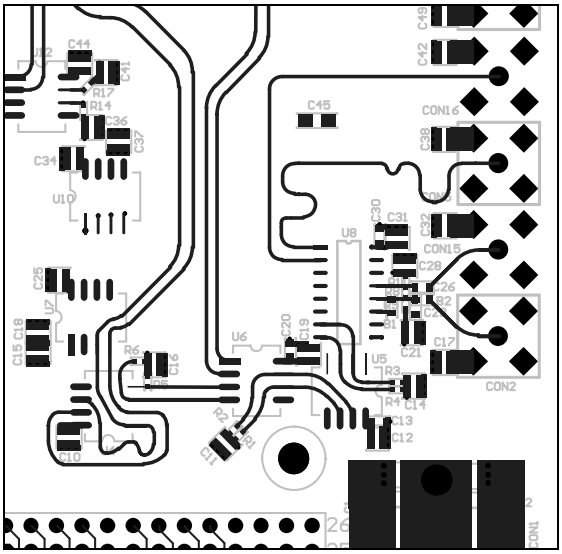


Figure 4-20 Mezzanine Card PCB Routing Style

Revision History/Future

This board has not been revised although extensive hand wiring of the Address Decoder section was needed for it to operate. In the initial design, the input polarity of the data bus latches was overlooked. As a result, the *CHIP ENABLE* outputs of the address decoders all had the wrong polarity. Since it was not feasible to hand wire 19

invertors on the board, a small PCB was constructed with an Altera PLD that performs the address decoder function (with the outputs inverted). For future revisions, the invertors shown in Figure 4-16 should be added onto the PCB to correct this problem.

4-3 Timing System Main Card

Overview

The purpose of the timing system Main Card is to generate numerous copies of the radar PRF pulse to synchronize the analog and digital sections of the radar. In addition to outputting direct copies of the PRF pulse, nine copies are output as special timing channels with user adjustable delay and pulse width. These adjustable channels are used to precisely define the radar transmit, receive, and data acquisition windows. A block diagram of the Main Card is shown in Figure 4-21.

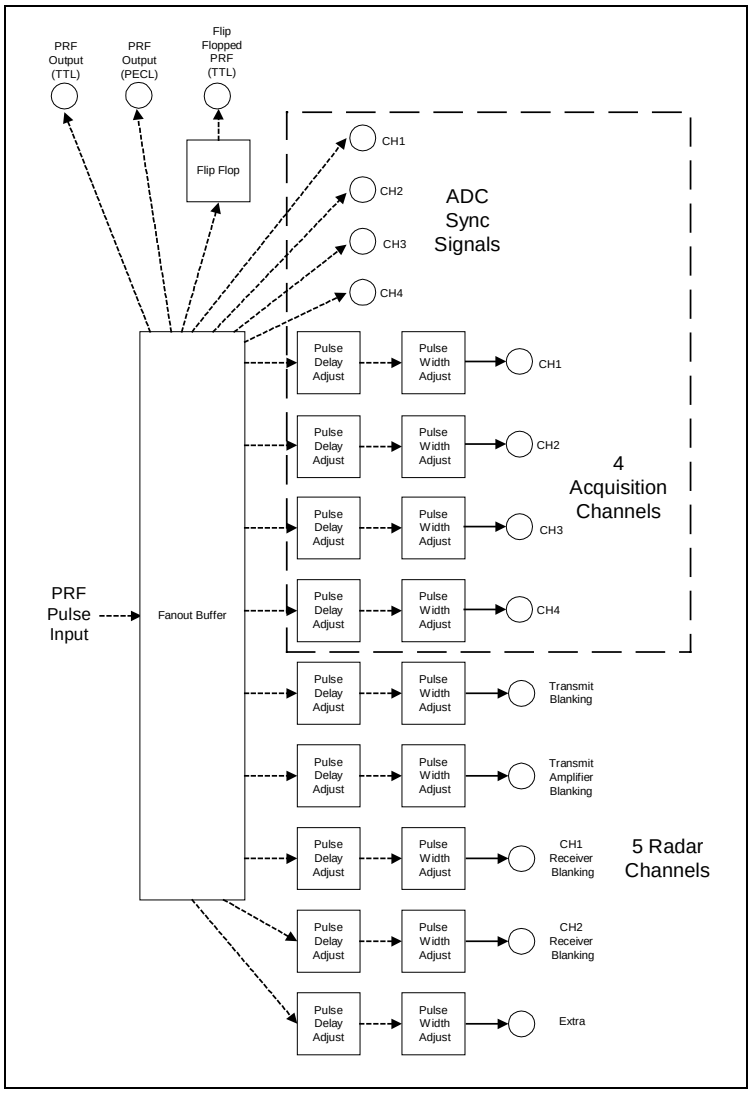


Figure 4-21 Block Diagram of the Timing Main Card

From the figure it is seen that most functionality on the card lies within the delay and width adjust circuits which are repeated nine times each. For the data acquisition section, the card outputs four PRF copies for the ADC synchronization and four adjustable channels to define the data acquisition window. For the radar analog section, five adjustable channels are output that can be used for general-purpose timing throughout the system. The most likely usage of these signals is transmitter and receiver blanking. A PRF flip flop signal is provided that toggles state with each PRF pulse. This

signal will be used for the special case when the radar toggles between two states on alternating PRF pulses (i.e., two transmit waveforms). Additionally, two PRF copies are output directly for future expandability.

Detailed Description

The top level schematic diagram of the Main Card is shown in Figure 4-22 with a full copy of the schematics given in APPENDIX E. Operation begins with the input clock and input PRF signals fanning out to the various sections of the circuit. Some copies of the PRF pulse enter the Delay Adjust circuits which output a PRF pulse after the time delay specified by the user (via the timing microcontroller). These delayed pulses then enter the Width Adjust circuits which output pulses with a width defined by the user. The width and delay adjust circuits operate by counting the specified number of input clock pulses. Since the delay and width are determined by digital counters, they are precisely repeatable with a resolution limited by the input clock period. The width adjusted pulses then pass through the Trim Delay circuits (similar to the Mezzanine Card) that allow the pulse edges to be adjusted to 20 ps of accuracy. Finally, the delay and width adjusted pulses are converted to TTL level signals and output to the radar and data acquisition systems.

In addition to the adjustable circuits, direct copies of the PRF signal are also output. The ADC synchronization signals are shown on the left of the schematic. These signals are generated by passing the PRF pulse through a trim delay circuit and outputting it directly as differential PECL level signals. Other PRF copies include a PECL level differential pair (similar to the ADC sync signals) and a single ended TTL version, both with trim delay circuits. The final circuit is a special flip-flopped PRF signal that toggles

state for each PRF pulse. This is passed through a trim delay and output as single ended TTL signal.

The Mezzanine Card interface connectors are shown on the top level schematic. Connector J2 transfers the 16-bit data bus directly from the microcontroller (via the Mezzanine Card connection). Connector J1 transfers the 18 chip enable lines from the Mezzanine Card address decoder. These lines are used to latch the delay and width counter values for the 9 delay/width adjust channels.

The remainder of this section is a detailed discussion of each of the hierarchical blocks in the schematic.

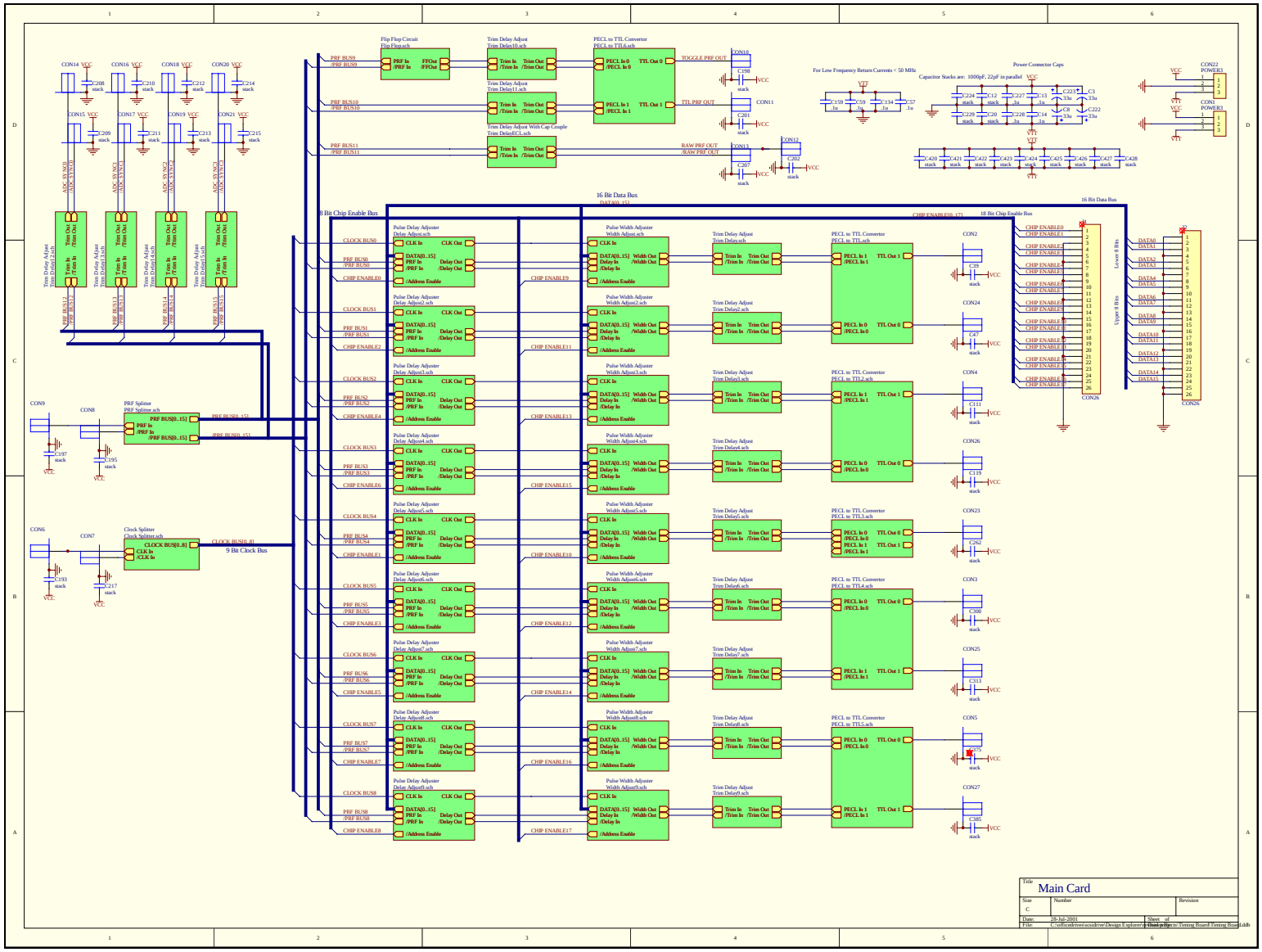


Figure 4-22 Main Card Top Level Schematic

Clock and PRF Splitters

The clock and PRF splitters are fanout circuits used to create multiple copies of the input signal. The schematics for both circuits are shown in Figure 4-23 and Figure 4-24. From the figures, it is seen that the same 1:4 and 1:9 fanout buffers are used in both circuits. The PRF signal is fanned out to a maximum width of 16 differential signals and the clock signal is fanned out to 9 signals. Differential signaling is preferred over single ended, especially on clock signals, however the downstream components using the clock signals have single ended clock inputs so the complementary clock signals are unused. From the schematic level it would appear that both fanout circuits could be reduced by 1-2 chips. The additional fanout chips are placed near their loads in order to streamline the PCB layout as opposed to having a single chip centrally located with numerous long PCB traces.

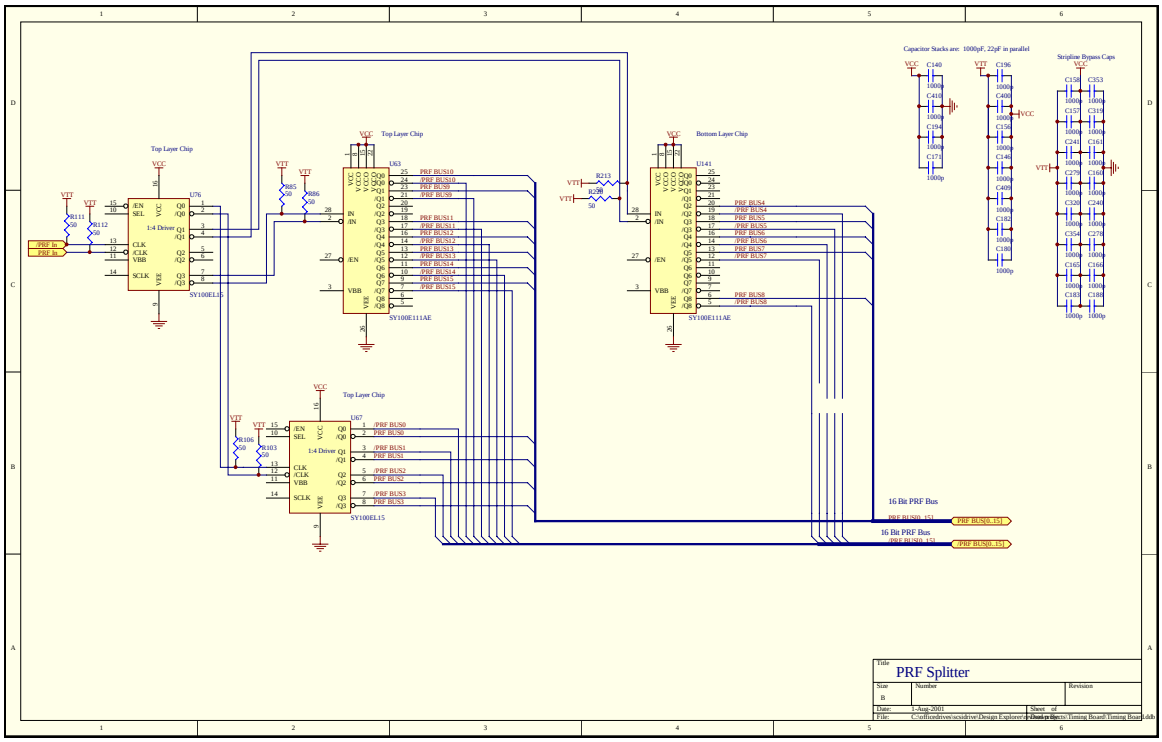


Figure 4-23 PRF Splitter Schematic

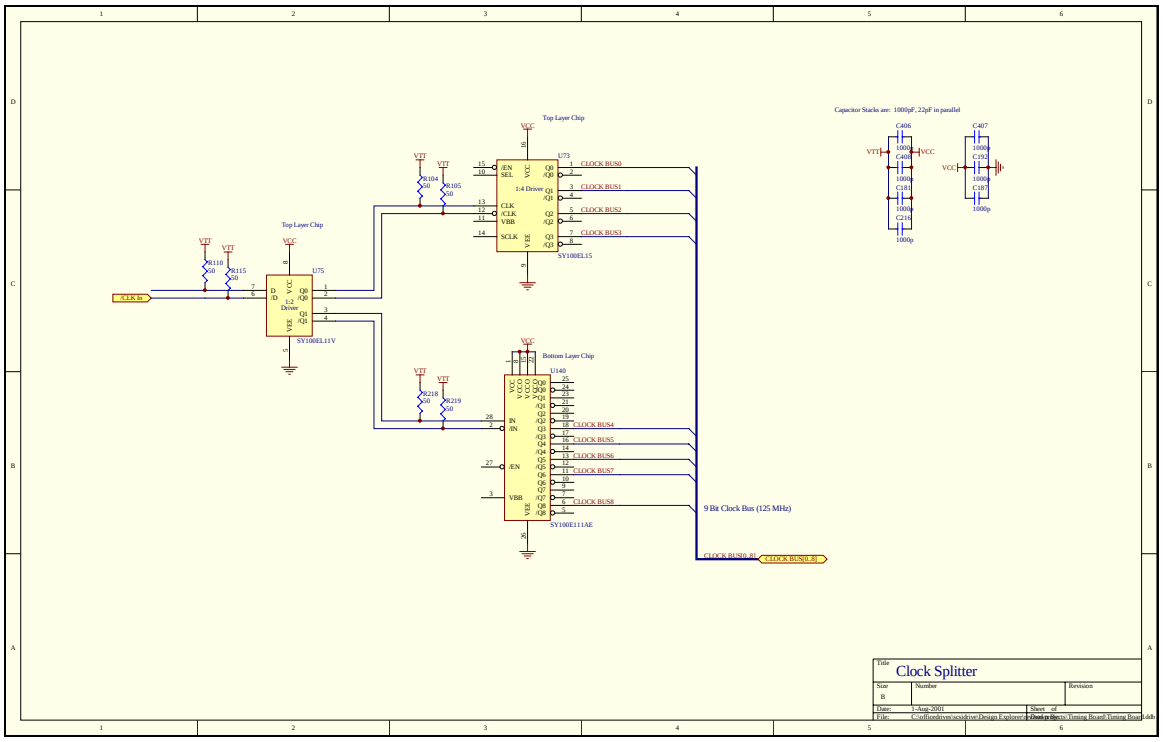


Figure 4-24 Clock Splitter Schematic

Pulse Delay Adjust

The pulse delay adjust is used to delay the PRF pulse by an amount specified by the user. This is accomplished using the 16 bit counter circuit shown in Figure 4-25. Circuit operation begins with *PRF* low, */PRF* high, and the clock operating at 62.5 MHz. In the steady state, U14 will have triggered on the previous cycle and will be asserting a high signal on the */CE* pin of *U4* and *U5*, disabling the counters. The desired count values will be previously programmed into *U1* and *U3* by the timing microcontroller via the 16-bit data bus and the */Address Enable* signal. When the PRF pulse arrives, *PRF* will go high and */PRF* will go low for one clock cycle. This will force the counters to load their parallel values from *U1* and *U3*. At the same time, *U14* will reset enabling the counters to count after the parallel load signal is released. On the next clock cycle, *PRF* returns low (/PRF high) and counting begins. Upon each rising edge of the input clock,

U4 will increment it's count value by one. When it reaches 255, its /TC signal will go low for one clock cycle, enabling the second counter, U5, to increment its count by one. The low counter then rolls back to zero and starts counting to 255 again. When *U5* reaches 255 it's /TC signal will go low for 255 clock cycles (the time it takes for the low order counter to roll over again). Finally, when *U4* reaches 255, its /TC signal goes low for a single clock cycle. When *U4* and *U5* both assert their */TC* signals, the output of the OR gate, U10, will go low. On the next clock cycle, both /TC signals will go high causing U10 to toggle from low to high. The rising edge output of U10 causes the flip flop (U14) output to go high thus disabling the counting process. The output of U10 is therefore a single clock cycle pulse that is delayed by the count value loaded by the timing microcontroller. This output is converted from single to differential signaling by U11 and output to the next block.

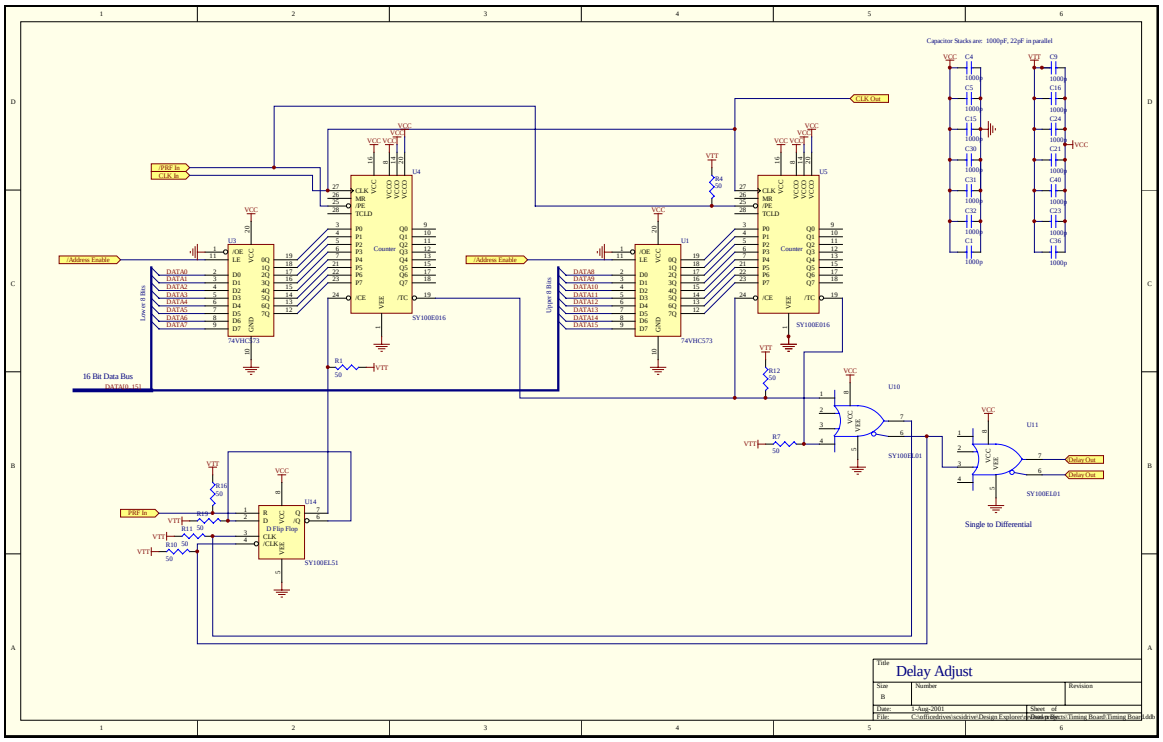


Figure 4-25 Delay Adjust Schematic

Detailed analysis of the circuit timing yields the following equation for calculating the circuit delay:

$$T_d = T_{clock}\left((255 - X) + 256(255 - Y)\right) \tag{6}$$

where

T_d	=	time delay through the circuit, seconds
T_{clock}	=	period of the input clock signal, seconds
X	=	decimal value of the lower byte latched by the user, no units
Y	=	decimal value of the upper byte latched by the user, no units

From the equation, the maximum time delay is 65,535 clock cycles which is 1.049 ms with a 62.5 MHz clock.

The circuit is speed limited by the propagation delay from the first counters */TC* output to the second counters */CE* input. This restricts the maximum clock frequency to about 400 MHz. Under normal operating conditions, the maximum speed anticipated is 125 MHz.

Width Adjust

The width adjust circuit inputs the delayed PRF pulse (single clock period width) and outputs a replica pulse with a width programmable by the timing microcontroller. The Width Adjust schematic is shown in Figure 4-26. Comparing Figure 4-26 with Figure 4-25, it is seen that the Width Adjuster uses the same circuit configuration as the Delay Adjuster except that the output is taken from the flip flop instead of the terminal count outputs. Circuit operation is also identical and the pulse width may be calculated using the same formula as that given in the Delay Adjust section.

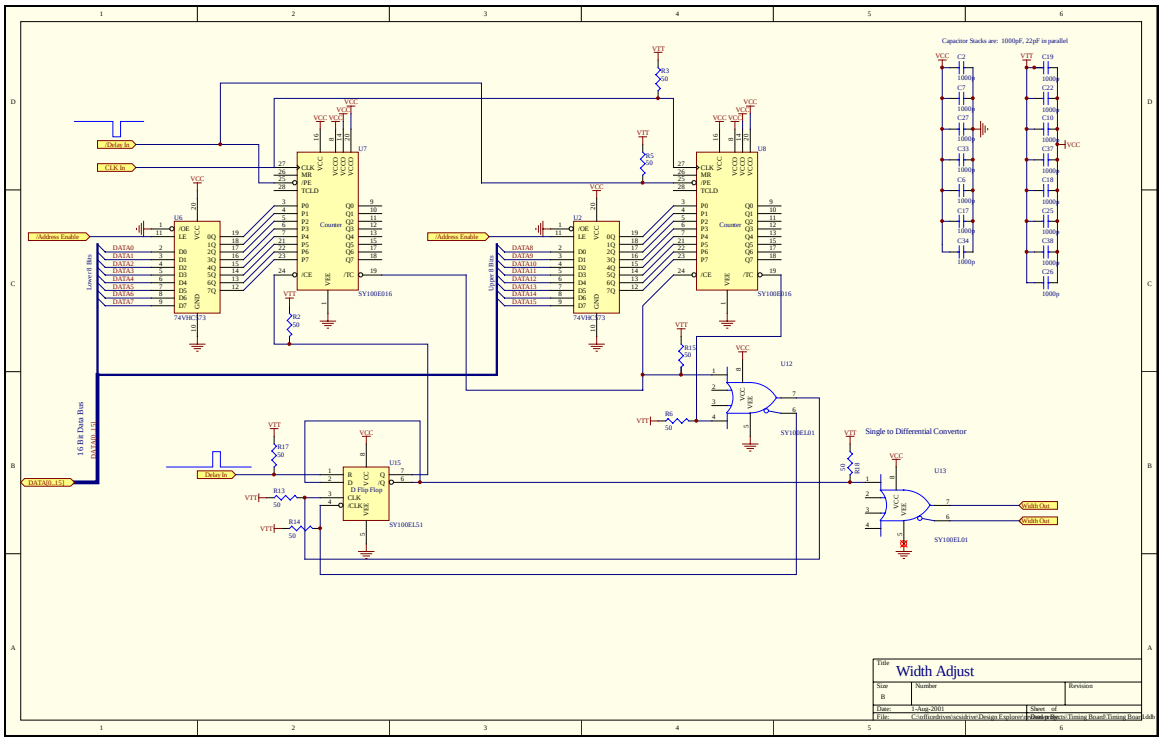


Figure 4-26 Width Adjust Schematic

Trim Delay

The Trim Delay circuit provides a precision (20 ps step) edge delay for synchronizing the timing system outputs to other synchronous circuits. A schematic of the Trim Delay circuit is shown in Figure 4-27. The circuit is identical to the Trim Delay discussed in the Mezzanine Card section. The user selects the amount of delay in a binary fashion using the dip switch, *SW11*. This binary word loaded onto the delay chip provides approximately 2 ns of delay with 20 ps increments.

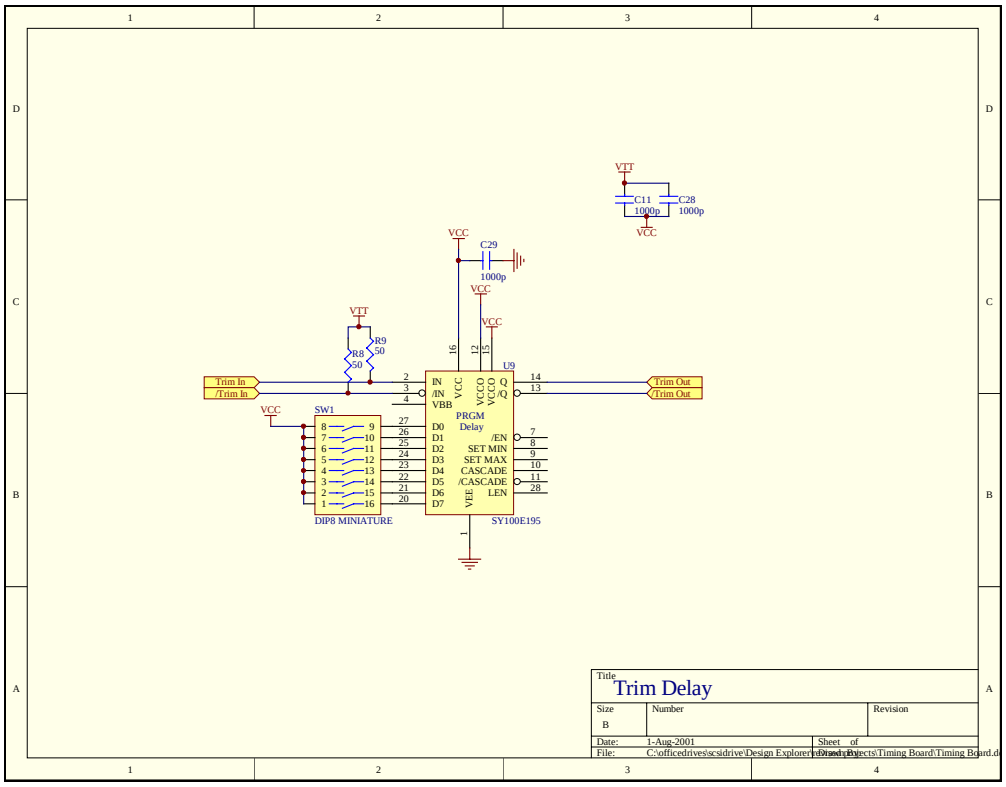


Figure 4-27 Trim Delay Schematic

Flip Flop

The purpose of the Flip Flop circuit is to toggle its output for each PRF pulse that is input. The Flip Flop schematic is shown in Figure 4-28. From the figure it is seen that the circuit consists of a single frequency divider chip. For each rising edge on the input, the output toggles state.

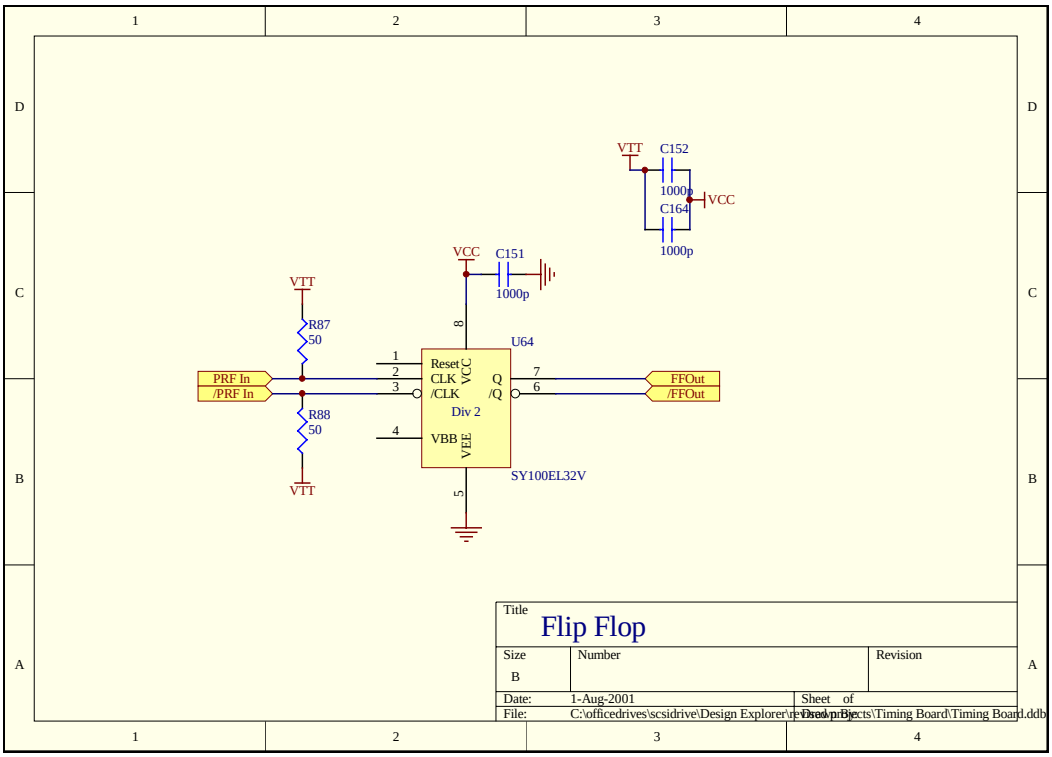


Figure 4-28 Flip Flop Schematic

PECL to TTL

The PECL to TTL block converts the timing system onboard signals from PECL levels to TTL levels before exiting the system to be used by other hardware. The schematic for this is shown in Figure 4-29. The circuit inputs two differential PECL signals and outputs two TTL signals. The TTL outputs may be capacitively coupled if desired by the user but the capacitors are initially populated with 0 Ω resistors during board production.

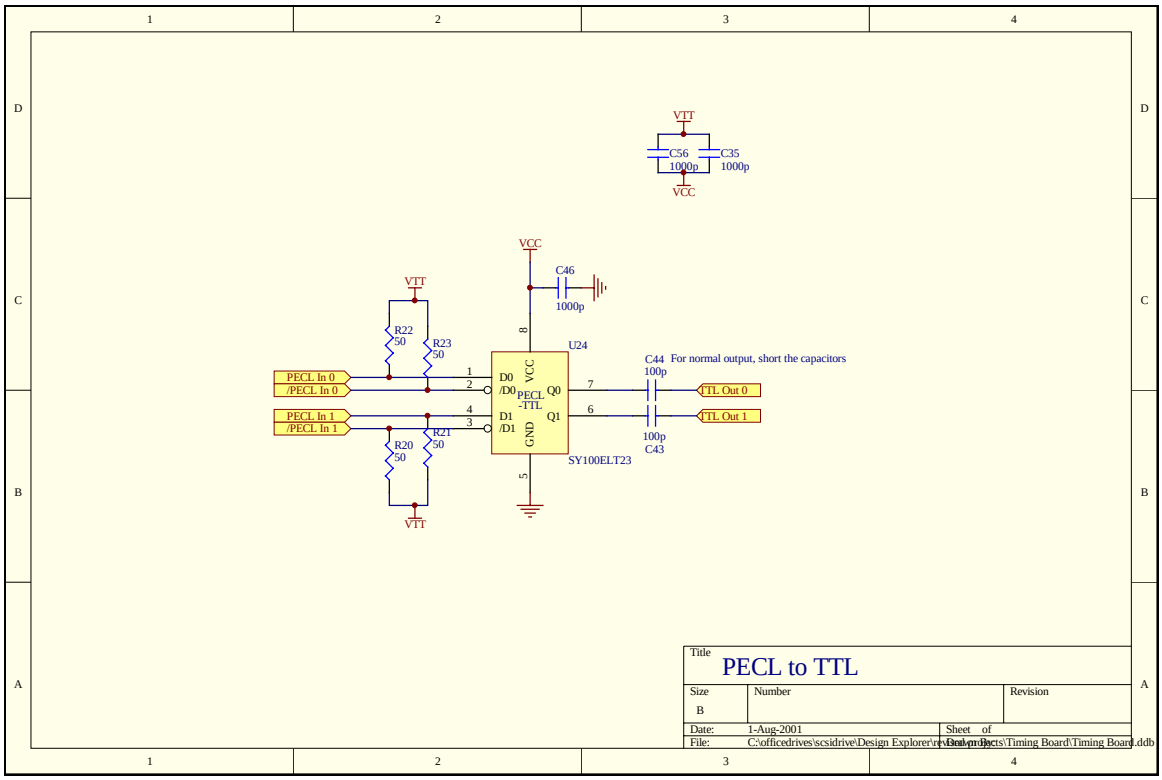


Figure 4-29 PECL to TTL Schematic

Main Card Physical Description

The Main Card is built of 8-layer FR-4 construction with standard 8 mil line width/spacing technology. A board stackup is shown in Figure 4-30. As seen in the figure, the board layers and layer thicknesses were carefully chosen to allow a 50 ohm line impedance on the microstrip layers (with a 14 mil line width) and 56 Ohms on the offset stripline (lower limited by the 8 mil line width). The offset stripline results from the desire to have the board as thick as possible (to reduce warpage) and the restriction of core material to a few discrete thicknesses. Taking these variables into account, the three prepreg layers are about 17.5 mils thick if they are evenly distributed. In hind site, the center prepreg layer could have been made 12.5 mils thick resulting in ordinary stripline with an impedance of around 52 ohms.

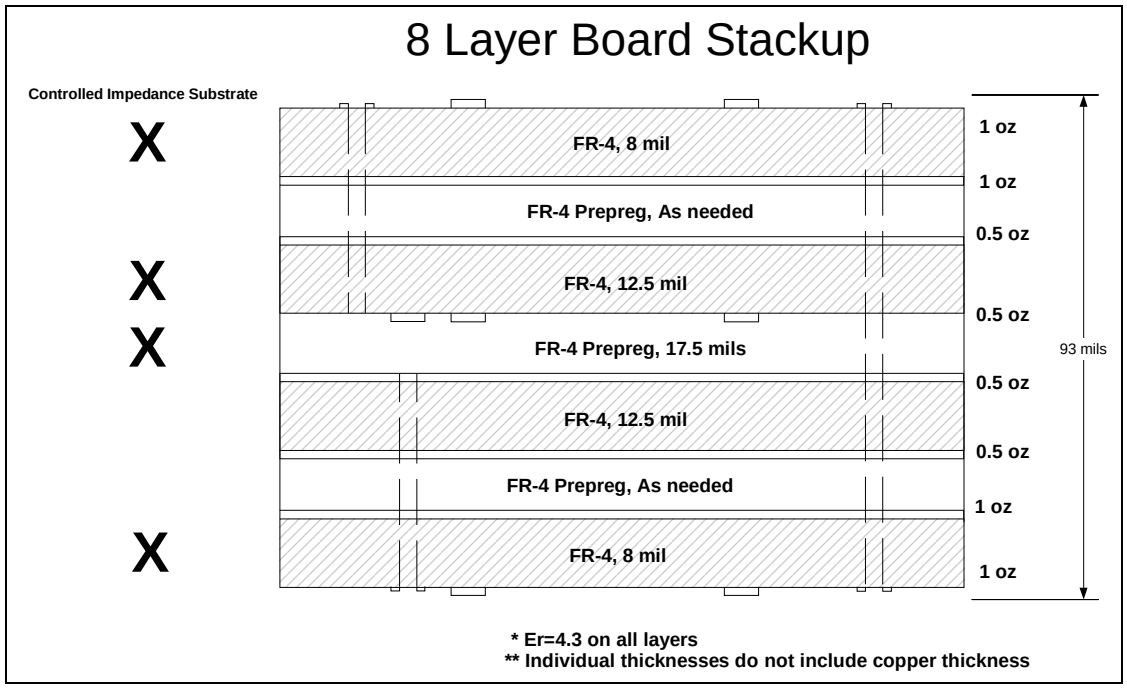


Figure 4-30 Main Card Stackup

Because of the large number of ECL components, both sides of the board are populated with integrated circuits as seen in the board layout of APPENDIX J. Because of the difficulty in routing such a board to strict controlled impedance specifications, the circuits are laid out with most high speed signals routed on a single layer with the components they connect to. Since the circuit contains nine identical channels, the component layout is repeated nine times with four channels on the top layer and five channels on the bottom. Because vias from each side of the board would interfere with the components on the opposite side, blind vias were used wherever possible. The blind vias go from layer 1 to layer 4 and from layer 5 to layer 8. Using this via layering, the card is constructed as two four layer boards which are then pressed together for the final 8-layer assembly. A photograph the finished board is shown in Figure 4-31.

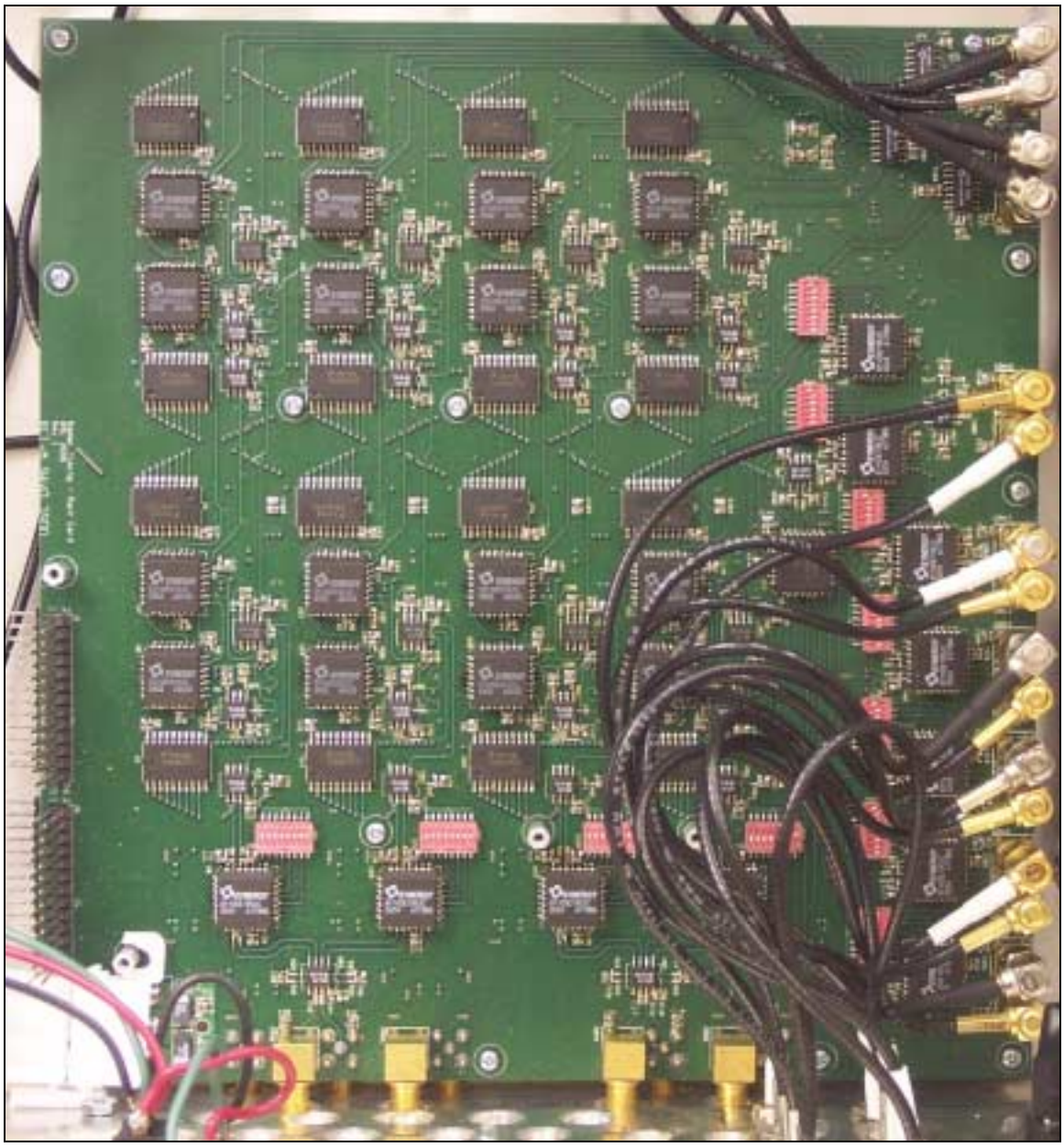


Figure 4-31 Photograph of the Main Card

With the fast edge rates (as low as 200 ps) of the ECL logic, low dielectric loss board material was considered during the design phase. This was deemed unnecessary after simulation of tangential loss on short transmission lines using EESOF. Because the signals operate at relatively low frequencies, the dielectric loss only affects the signal edges. The effect of dielectric loss on the signal edges was determined to be negligible

for the relatively short lines used on a PCB (less than 12 inches). This is in contrast to the case when the ECL signals toggle at frequencies greater than 1 GHz where the fundamental component of the signal becomes attenuated resulting in a reduction of the maximum operating frequency.

Revision History/Future

This board has been revised once in order to change the board material and blind via style. The board was originally designed to use the Rogers 3000 series material. After seeing the results of this material on the Mezzanine Card and performing analysis on the effects of tangential loss, it was determined that FR-4 would be adequate for this lower frequency design. Additionally, the original blind vias overlapped the middle layer resulting in the need for controlled depth drilling in the manufacturing process. At the advise of the PCB manufacturer, the blind vias were altered to go from layers 1 and 8 to layers 4 and 5 as described previously. The PCB is then constructed as two four layer PCBs which are then stacked together to form the 8-layer board.

Some hand wiring was necessary to make the board operate due to three errors in the schematics. Pins 3 and 4 on all SY100EL51s were swapped in the original schematics. This was corrected on the prototype by cutting PCB traces and swapping the differential pins of the driving devices. In the Delay Adjust schematics, the input of the second OR gate (single to differential) was attached to the wrong differential output. This was corrected automatically when the EL51 problem was corrected. Finally, pins 3 and 4 on all SY100ELT23s were swapped in the schematics. This was corrected by lifting and crossing pins on the PCB. All of these errors have been corrected in the

schematics that appear in this document so that the documentation reflects a working circuit.

4-4 Timing System Microcontroller

Description

The Timing System Microcontroller is used to provide an interface for programming the timing system hardware. The F16-Mite evaluation kit was selected for this based on cost and speed of development. This evaluation kit includes the following key components:

- Motorola HC16 microcontroller (including evaluation PCB with RAM and flash memory)
- In circuit debugging cable with PC debugging software
- PC based "C" compiler with an extensive set of libraries written specifically for the evaluation board.

A photograph of the evaluation board is shown in Figure 4-32.

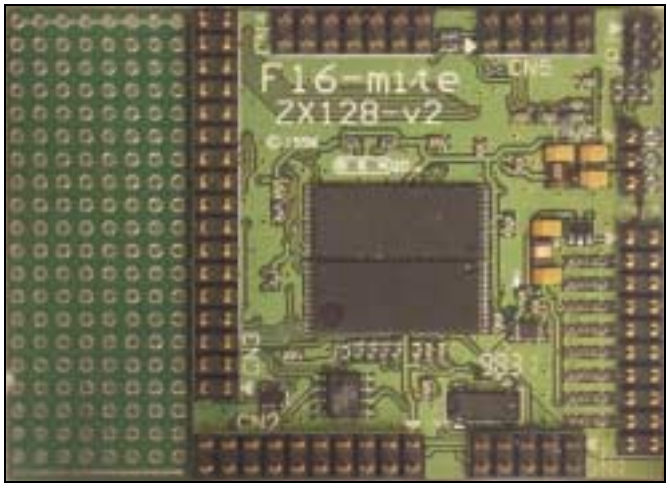


Figure 4-32 Timing System Microcontroller Photograph

A pinout of the microcontroller connectors and their application is shown in Table 4-2. From the table, the custom pinouts used for the timing system are an 8-bit data bus, 5-bit address bus, one PRF enable line, and one single shot line. Note that only an 8-bit

data interface is provided for the 16-bit timing data bus. This bus was multiplexed using the circuit of Figure 4-33 in order to conserve pins on the microcontroller interface. This circuit was implemented on a small PCB which was mounted adjacent to the microcontroller in the timing system chassis. To multiplex the data, the microcontroller writes in the following sequence:

1. Write the lower data byte to the 8-bit data bus.
2. Toggle the latch address to latch 8-bits to the 16-bit data bus LSB.
3. Write the upper data byte to the 8-bit data bus.
4. Toggle the timing system address to latch all 16 bits.

Connector/ Pin	Microcontroller Port/Bit	Function		Connector/ Pin	Microcontroller Port/Bit	Function
CN2-10	PF0	DATA0		CN4-11	OC1	RESET INT
CN2-9	PF1	DATA1		CN4-10	OC2	INT SER DATA
CN2-8	PF2	DATA2		CN4-9	OC3	INT LATCH
CN2-7	PF3	DATA3		CN4-8	OC4	INT SER CLK
CN2-6	PF4	DATA4				
CN2-5	PF5	DATA5		CN5-ALL		RS232 LINK
CN2-4	PF6	DATA6				
CN2-3	PF7	DATA7		CN3-ALL		LCD-TOUCHPAD
CN2-18	PE0	ADDR0				
CN2-17	PE1	ADDR1				
CN2-16	PE2	ADDR2				
CN2-15	PE4	ADDR3				
CN2-14	PE5	ADDR4				
CN2-13	PE6	SINGLE SHOT				
CN2-12	PE7	PRF ENABLE				

Table 4-2 Microcontroller Pinout

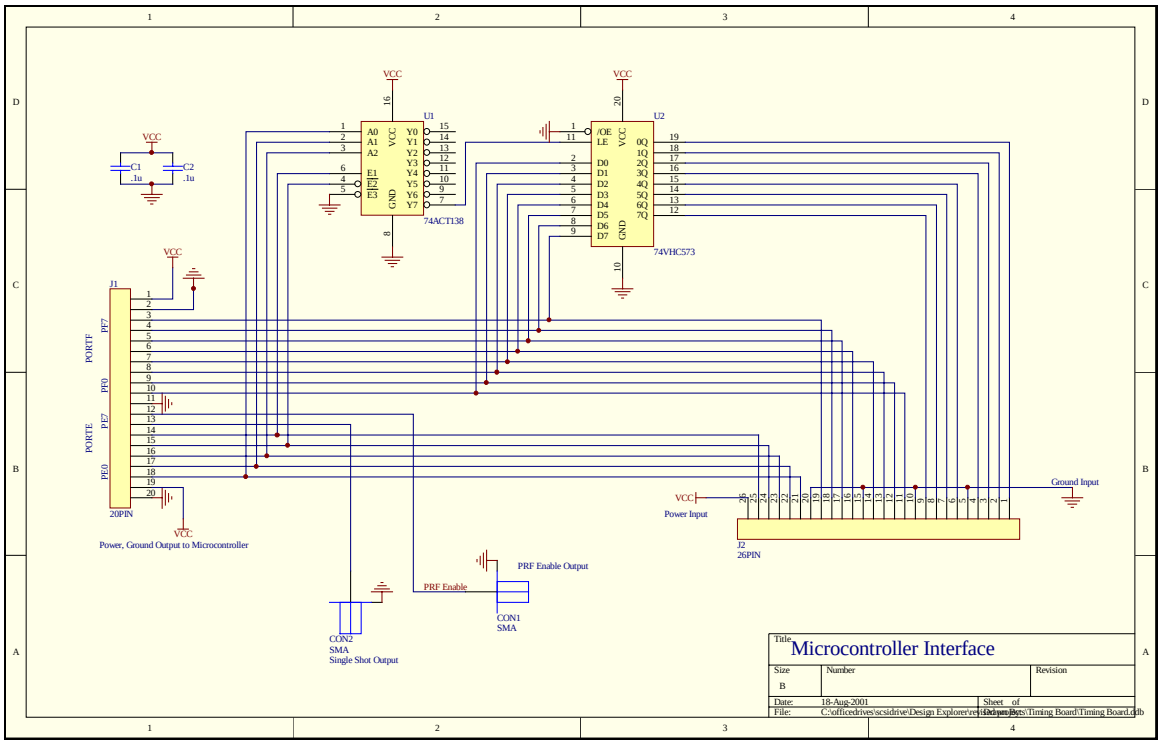


Figure 4-33 Microcontroller Interface Schematic

Other interfaces shown in Table 4-2 include an RS-232 connection, the LCD/Touchpad connection, and a custom interface for the Averaging Card. The Averaging Card interface is used to transfer the integration count from the acquisition computer and also supplies the reset signal for all Averaging Cards. The integration count interface is a three wire connection with serial clock, serial data, and data latch signals. The 16-bit data are written by shifting a single bit at a time (LSB first) onto the data wire while toggling the serial clock signal for each bit. After all 16 bits are written, the data latch signal is toggled to latch the bits into the integration counter circuit.

The microcontroller can be controlled directly by the user via an LCD-Touchpad panel or via an RS-232 link to the acquisition computer. The LCD interface panel is shown in Figure 4-34. As seen in the figure, the panel has seven buttons and a two line backlit LCD display. To operate, the user presses the left/right and center buttons to

navigate through the menu system. When a value is selected that the user wishes to modify, the up/down buttons are pressed to increase or decrease that timing value. The *Cycle Digits* button is used to move the cursor to each of 6 digits to speed programming (i.e., user can increment in 1, 10, 100 ns, etc.) A *Single Shot* button is provided that allows the microcontroller to output a single (debounced) pulse that can be input into the Mezzanine Card to generate a single PRF pulse. The LCD menus are listed in Table 4-3.

Figure 4-34 Microcontroller LCD Interface

Menu	1	2	3-4	5	6	7-9	10
Line 1	PRF Period	DCH1 Delay	. . .	DCH4 Delay	RCH1 Delay	. . .	RCH5 Delay
Line 2	PRF Frequency	DCH1 Width	. . .	DCH4 Wdith	RCH1 Width	. . .	RCH5 Width

Table 4-3 Microcontroller Interface Menus

For interfacing directly to a computer, the F16-Mite system provides an RS-232 serial connection. This link allows the acquisition computer to control all timing parameters by sending special commands through the serial port. Library routines were provided with the microcontroller kit to initialize the port and send and receive data, eliminating the need for knowledge of the low level circuitry and signaling. Valid commands that can be sent through the serial port are shown in Table 4-4. All commands

must be sent as a three byte packet to be received properly. After each byte is sent, the microcontroller will respond by sending the value 0xF back to the computer. This allows handshaking between the two units so that a sequencing error can be detected at either end of the link.

Function	First Byte (Address)	Second Byte (Data)	Third Byte (Data)
Send Timing Data	0x00-0x12	Data MSB	Data LSB
Single Shot Pulse	0x20	Any	Any
Reset Averaging Cards	0x21	Any	Any
Send Integration Count	0x22	Data MSB	Data LSB
Enable/Disable PRF	0x23	1=enable 0=disable	Any

Table 4-4 Valid Microcontroller Serial Commands

The microcontroller code was written in "C" and is presented in APPENDIX M. The code was compiled using the Imagecraft ICC16 compiler which was included in the F-16 Mite kit. Extensive use of the special libraries included in the kit for the LCD interface, general I/O, and RS-232 I/O resulted in a large reduction of design time versus assembly language programming.

CHAPTER 5 SPECIAL TEST AND MEASUREMENT DEVICES

5-1 ECL-to-Sine Wave Converter

An ECL-to-sine wave converter can be used in the system to convert the timing system (high frequency) output clock signals to sine waves. The converter can be used to produce a coherent sine wave for input into the A/D converters (for testing purposes) or for generating special clocking configurations of multiple channels. One possible clock configuration requiring the converter is the interleaved configuration shown in Figure 5-1.

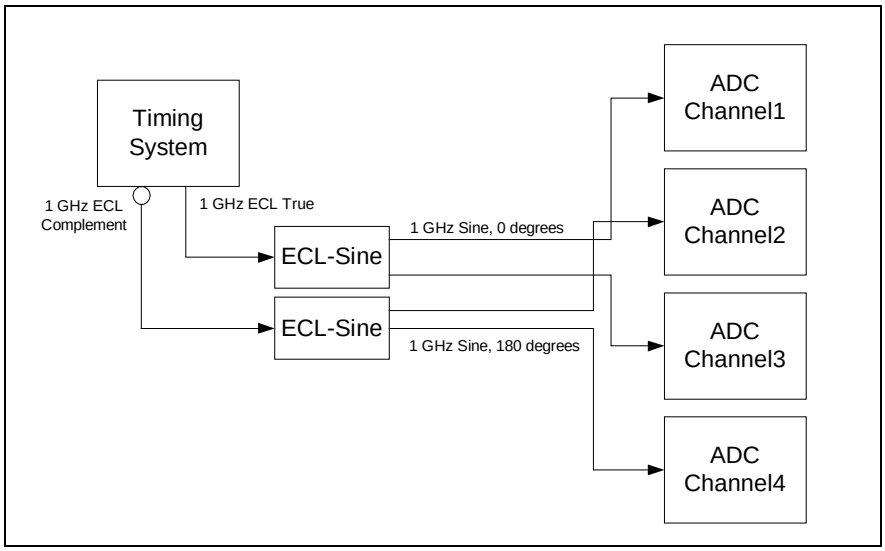


Figure 5-1 Interleaved ADC Clock Configuration

A schematic of the ECL to Sine Wave Converter is shown in Figure 5-2. The circuit operates as follows. A PECL signal is input into connector CON1. This signal is DC biased with 50 Ω to +3 V through inductor L1. The signal is AC-coupled to the amplifier Q2 through capacitor C6. The amplifier has an input impedance of 40 to 50 Ω at frequencies up to 3 GHz. A small value resistor can be inserted into R6 to be used as a voltage divider for the amplifier input although in most cases it will be populated with a

zero Ω resistor. The amplifier is biased as recommended by the manufacturer using the bias and AC-coupled network of L6, R3, C9, and C1. This amplified "square wave" then enters the filter section. The filter was implemented as a removable PCB section so the filter could be characterized prior to insertion into the circuit. The generic layout of the filter PCB allows the designer to easily change filter configurations. Finally, the filter outputs a sine wave which is input into the power divider, U2, which produces two copies of the sine wave. A photograph of the prototype converter is shown in Figure 5-3.

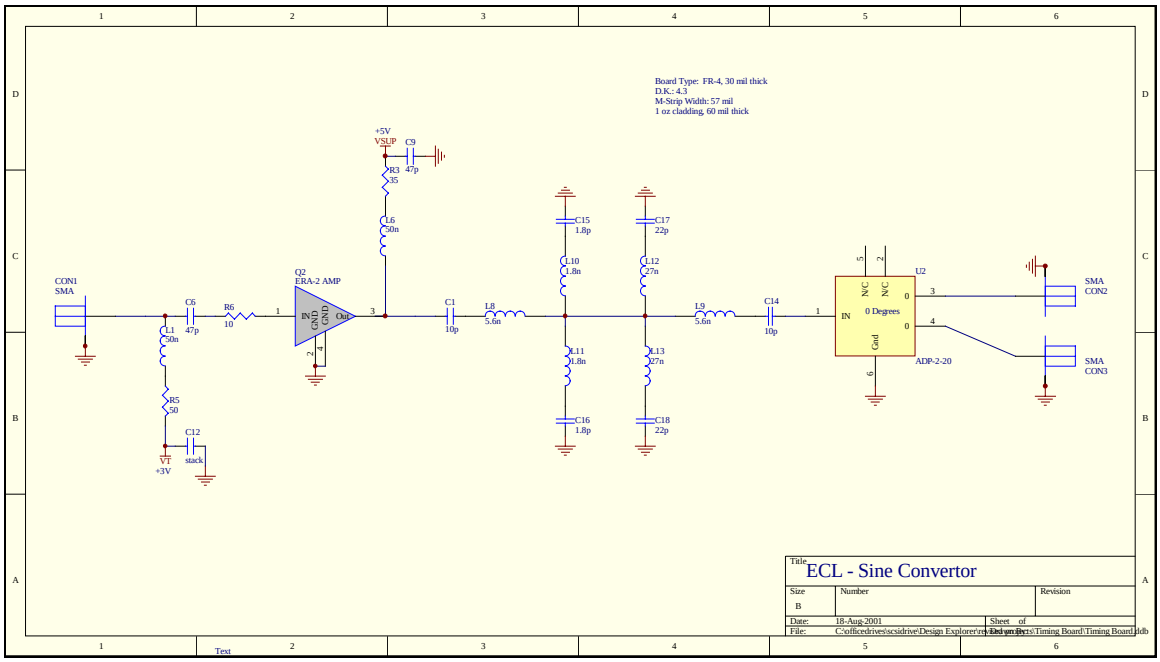


Figure 5-2 ECL-Sine Wave Converter Schematic

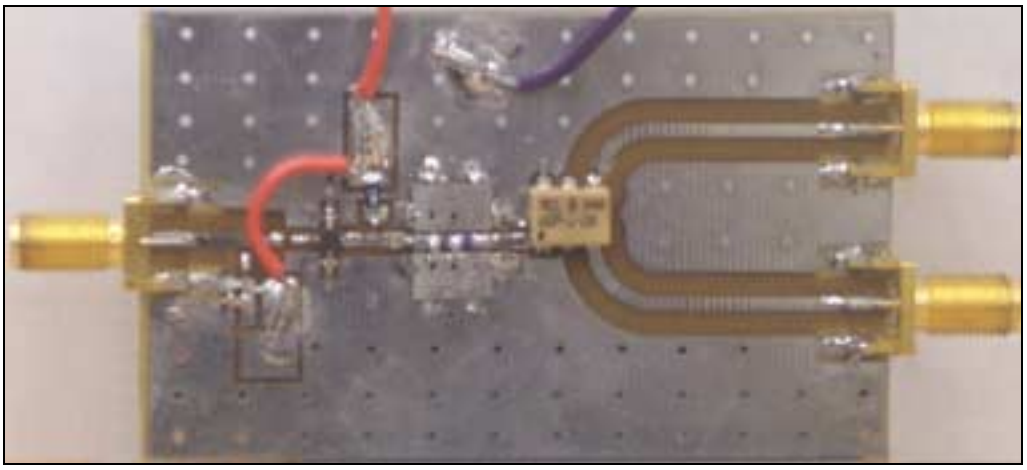


Figure 5-3 ECL-Sine Wave Converter Photograph

5-2 ADC Card Test Circuit

A simple method was needed to validate the basic operation of the ADC cards. The two basic test methods are to capture and analyze the digital signals or to convert the digital signals back to analog for viewing on an oscilloscope. While the first method is most accurate, it is also the most complex requiring a logic analyzer with a special test fixture to connect to the ADC Card. The second method was chosen for this project for simplicity and because it can be used to validate a card in the field with only an oscilloscope. A block diagram of the testing process is shown in Figure 5-4.

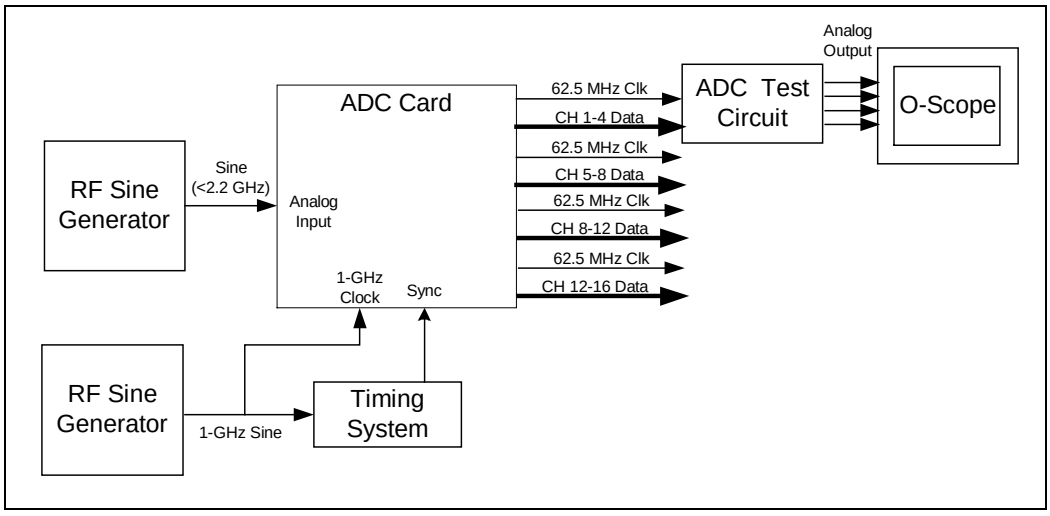


Figure 5-4 ADC Card Testing Diagram

The ADC test circuit contains four 100-MHz DACs that each convert a single channel of 8-bit data to an analog signal. A schematic of the circuit is shown in Figure 5-5 with a board photograph shown in Figure 5-6. From the figures it is seen that the test circuit connects to two of the eight ADC card connectors at a time. Each pair of connectors inputs four of the demultiplexed 8-bit paths and a single clock operating at 62.5 MHz. Each "channel" of data and the clock signal are input into the four 8-bit DACs which regenerate the analog waveform that is input into the ADC card.

Although the A/D converter is sampling at 1 GSample/s the output waveform is recreated at a rate of 62.5 Msamples/s (since only 1 out of 16 samples is recreated per channel) and therefore has a Nyquist frequency of 31.25 MHz. Because of this, only signals up to 31.25 MHz are recreated with the correct frequency. This does not limit evaluation of the ADC Card as the input can still be operated to the full bandwidth of 2.2 GHz, the frequency will simply be folded to the first Nyquist zone as if it were aliased. A sample output waveform of the test circuit is shown in Figure 5-7. For the figure, the input frequency was 66.4 MHz resulting in a DAC output of 3.9 MHz. Note, the DAC steps are clearly visible as only 16 samples are output per period of the input waveform. It is important to realize that the A/D converter is actually converting 256 samples per input period for this measurement but only 1 of every 16 is being displayed.

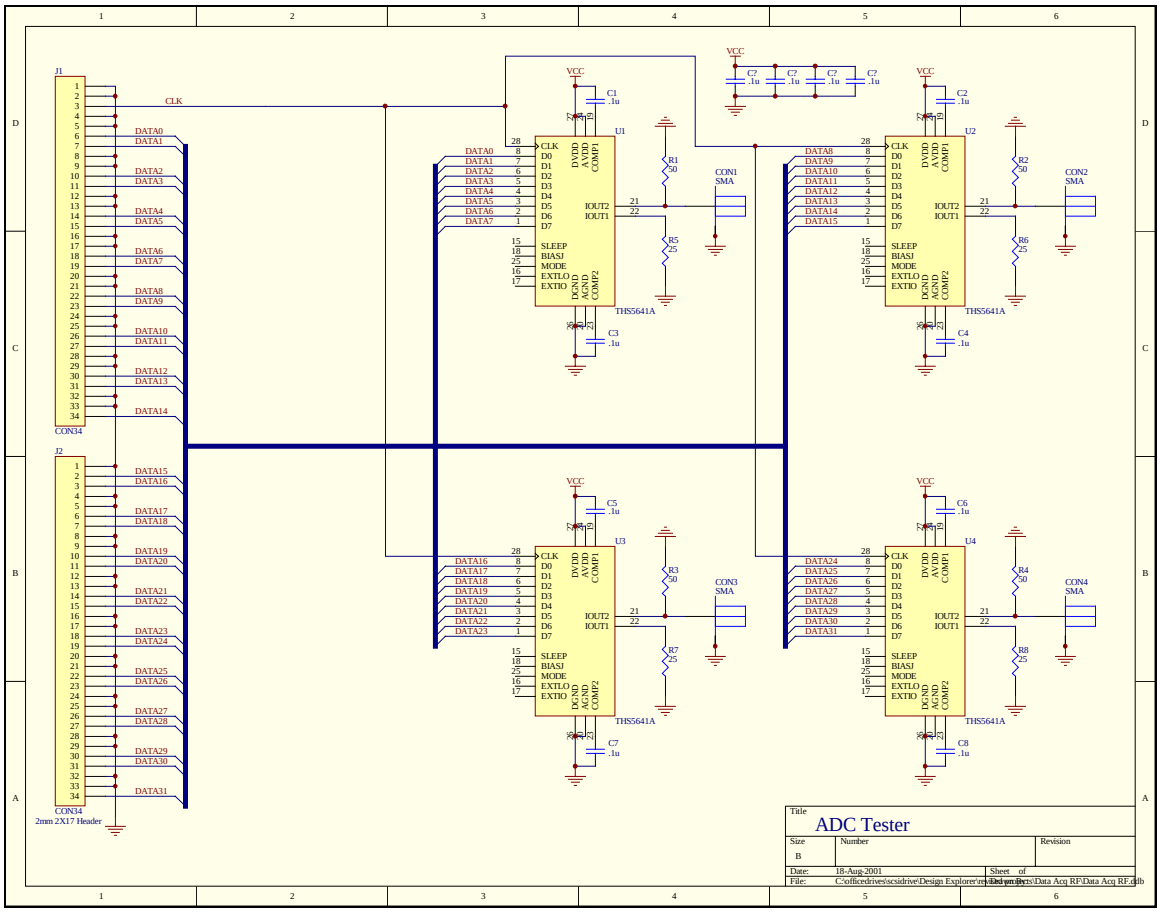


Figure 5-5 ADC Card Test Circuit Schematic

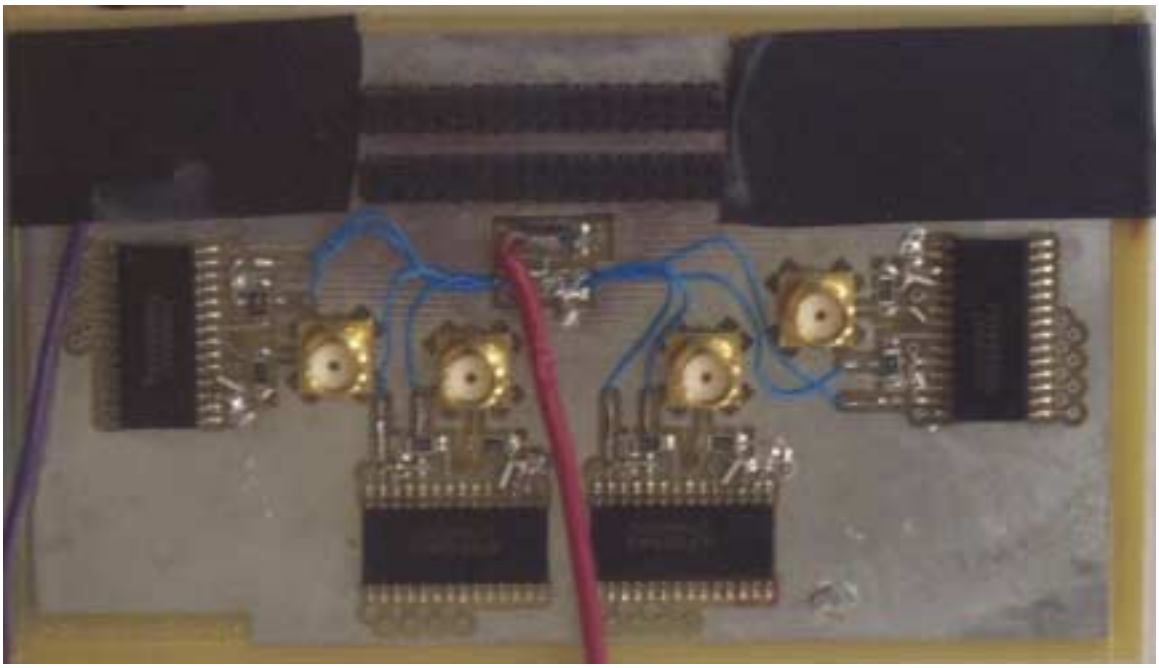


Figure 5-6 ADC Card Tester Photograph

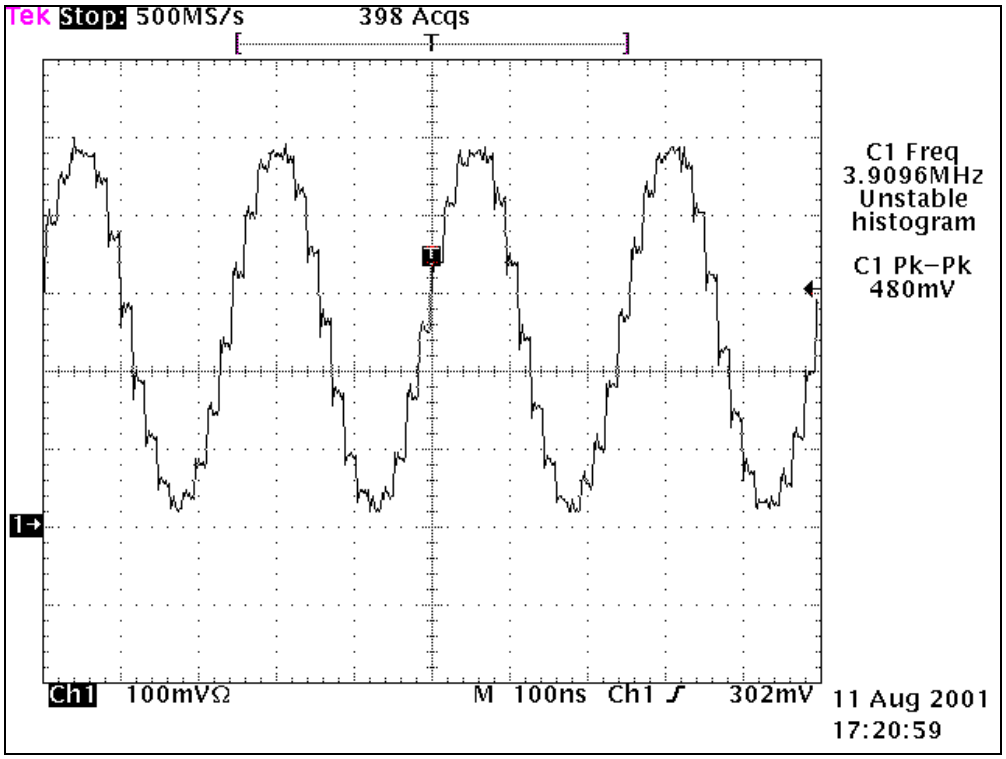


Figure 5-7 ADC Card Tester Output Waveform

5-3 Averaging Card Test Circuit

A special test circuit was needed to generate a known stimulus for validating the Averaging Cards. The circuit of Figure 5-8 was designed to output an 8-bit counting sequence, four bytes at a time, into one Averaging Card channel at a time. The output is asserted onto the input FIFOs and eventually flows into the DSPs. Once read into the DSPs internal memory, it can be viewed using the DSP in-circuit debugger. The data can therefore be used to validate all input and output functionality of the Averaging Cards including all input and output PCB connections, acquisition windowing circuitry, DSP interrupts, and the DSP averaging algorithm.

The circuit operates as follows. A 62.5-MHz clock and a TTL level PRF signal are input from the timing system through the SMA connectors CON1 and CON2. These signals drive the Altera PLD, U1, which generates the continuous counting sequence

shown in Table 5-1. The rising edge of the PRF signal resets the counting sequence thus making the sequence coherent to the timing system. The signals are output through two 34 pin connectors which directly connect to a single Averaging Card channel. A photograph of the completed circuit is shown in Figure 5-9.

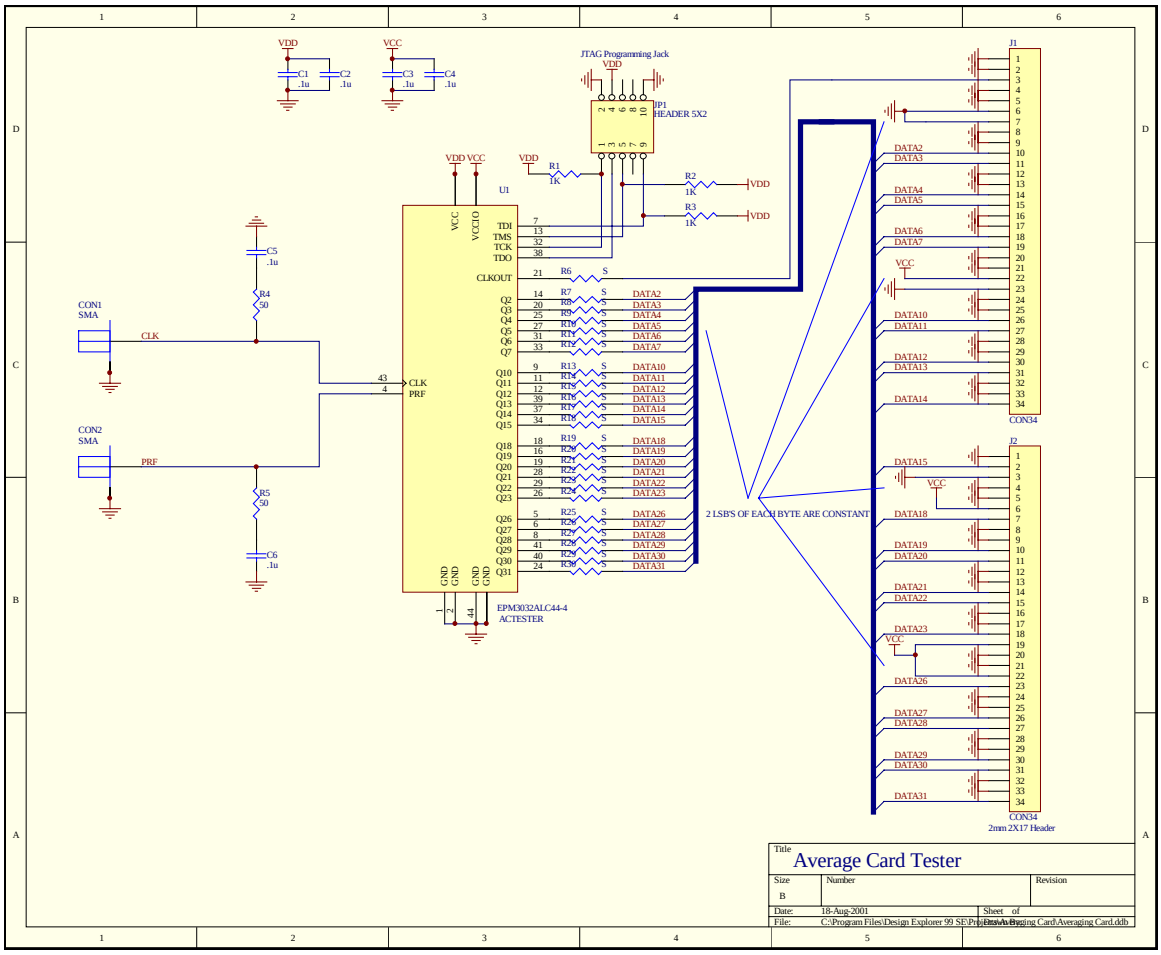


Figure 5-8 Average Card Tester Schematic

Output / Clock Cycle	Channel 1	Channel 2	Channel 3	Channel 4
1	0	1	2	3
2	4	5	6	7
3	8	9	10	11
…	…	…	…	…
63	248	249	250	251
64	252	253	254	255
65	0	1	2	3

Table 5-1 Average Card Tester Output

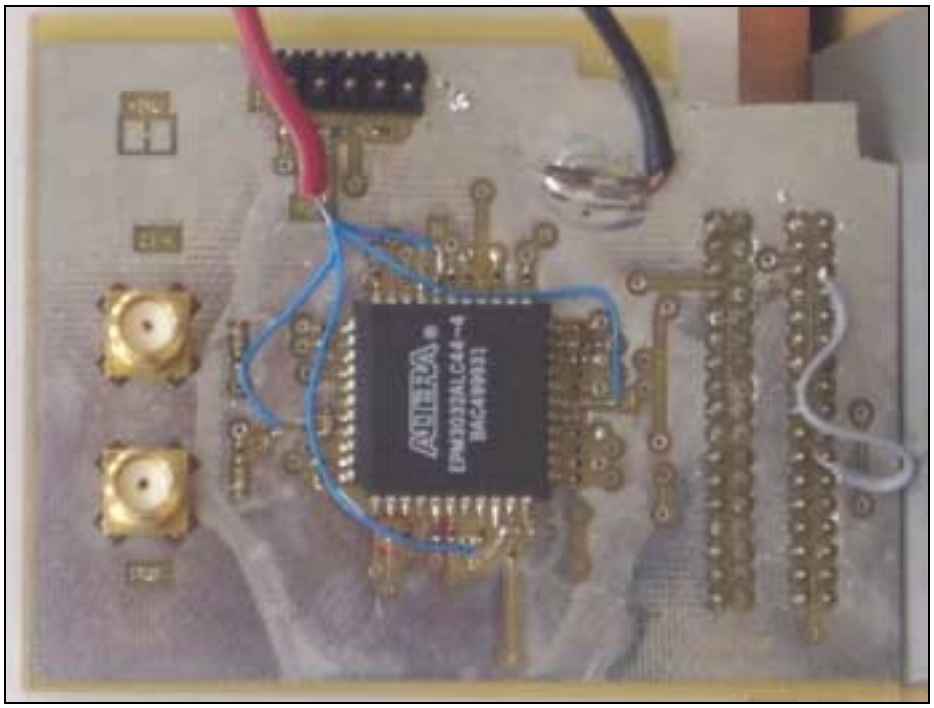


Figure 5-9 Average Card Tester Photograph

5-4 Other Special Equipment

ECL/PECL Coaxial Terminators

Measurement of ECL/PECL signals creates difficulties for traditional test equipment (i.e., oscilloscopes, spectrum analyzers) because of the special termination requirements of ECL. These measurement devices typically terminate high speed cabling with 50 Ω to ground where ECL devices require 50 Ω to the termination voltage. In order to view the signals for circuit validation and signal integrity checking, special terminators are required that satisfy the ECL termination requirements, operate with an output load of 50 Ω to ground, and have a bandwidth greater than 2 GHz. These special terminators are commercially available and a single terminator can be used for both ECL and PECL signals. A photograph of the terminator is shown in Figure 5-10.

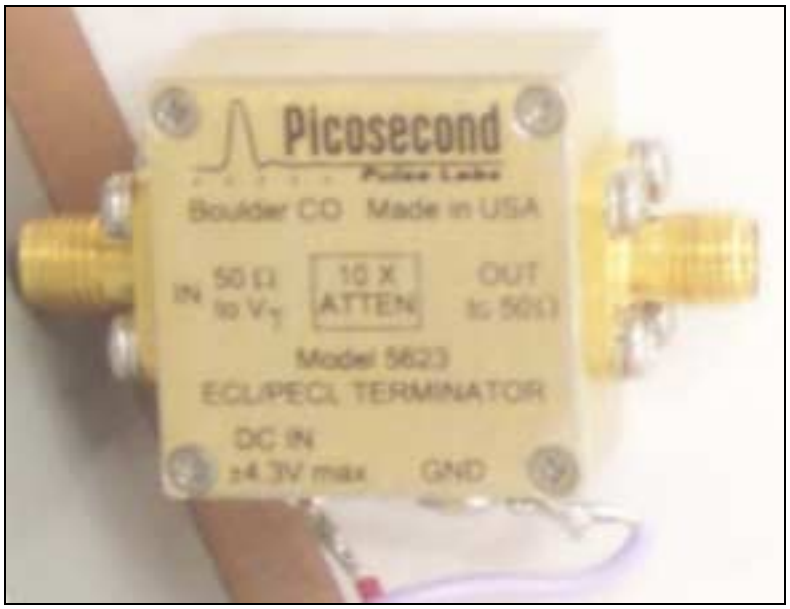


Figure 5-10 ECL Terminator Photograph

High Speed Oscilloscope

With edge rates of 200 ps and frequencies in excess of 2 GHz, the timing system signals exceed the limits of most digital oscilloscopes available today. To have confidence in circuit performance and signal integrity, 3 GHz is the minimum analog bandwidth needed to view a 1-GHz ECL square wave since this will include the signal's third harmonic. With less bandwidth the signal will appear as a sine wave. Even greater bandwidth is required to accurately see all signal features. For this project a 20-GHz oscilloscope was available which provided ample bandwidth for accurate signal measurement. The output of this scope is shown in Figure 5-11 with the 1-GHz PECL output of the timing system captured on the screen. As seen in the figure, the output eye at this frequency is still nearly full at 750 mVp-p and the 10-90% edge rates are approximately 200 ps.

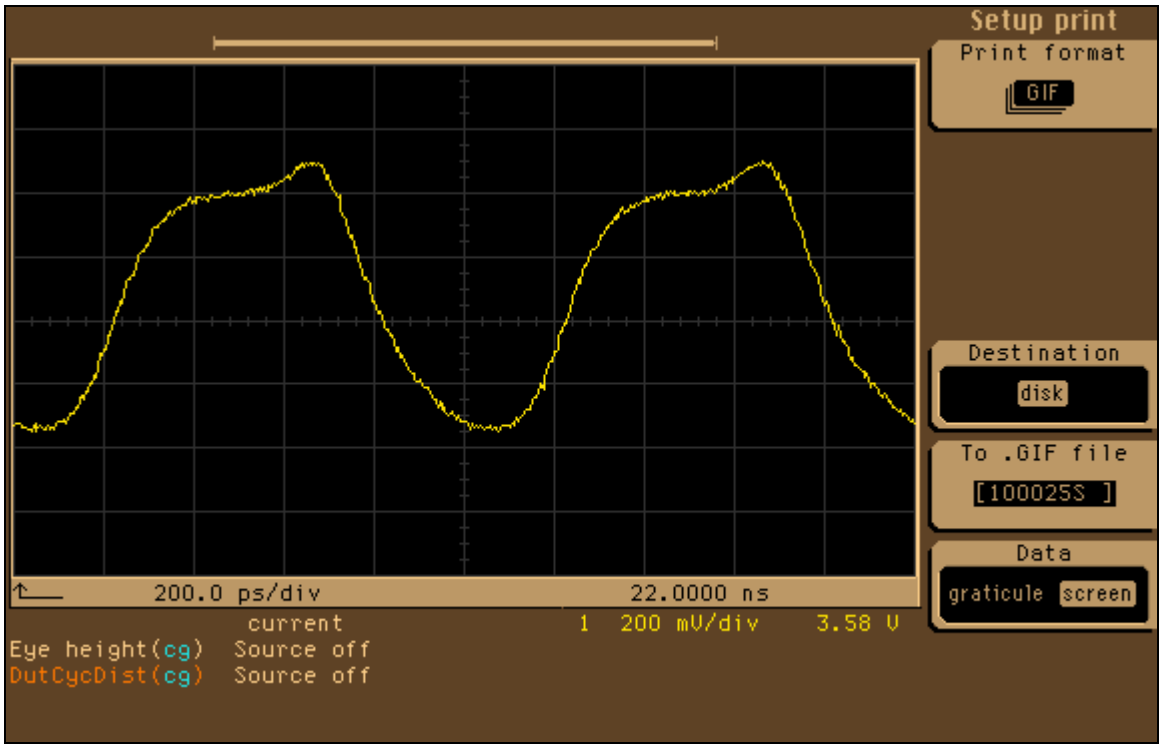


Figure 5-11 Oscilloscope Output of a 1-GHz PECL Signal

CHAPTER 6 MEASUREMENTS

6-1 Timing System Measurements

For functional validation, the timing system was set up as shown in Figure 6-1. The system was operated with an input clock frequency of 1 GHz. With the oscilloscope triggering on the PRF signal, the output of each channel was viewed on the oscilloscope. The pulse width and delay was shown to be independently adjustable in 16 ns increments up to a maximum of 1.048 ms when operated from the LCD front panel. Additionally, the system was placed in serial control mode and each channel was operated with computer control.

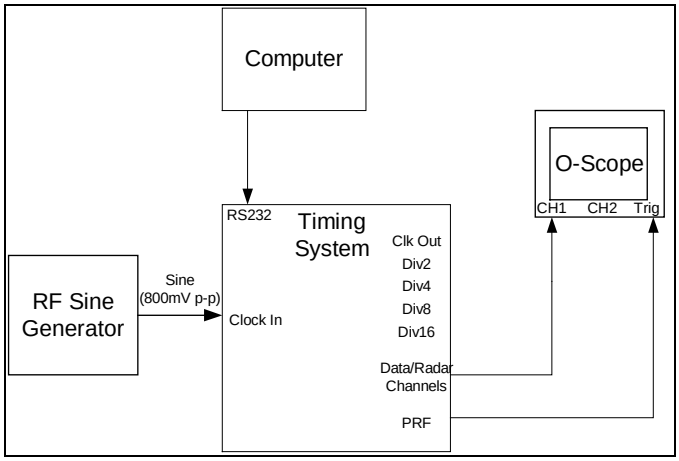


Figure 6-1 Timing System Validation Setup

To verify the input operating range, the system was again setup as shown in Figure 6-1. First the input clock signal amplitude was varied with an input frequency of 1 GHz. The clock divider outputs were viewed on the oscilloscope and found to be stable with input voltages between 400 mV and 1 Vp-p (voltages greater than 1 Vp-p were not tested as it exceeds the normal input range of the ECL circuitry).

Finally, the maximum operating frequency of the system was tested by leaving the input voltage at 800 mVp-p and increasing the input frequency. By viewing the divided clock outputs on the oscilloscope it is possible to see the frequency where input failure occurs. This frequency was found to be 2.45 GHz. An oscilloscope screenshot of the stable clock divider outputs with an input frequency of 2.4 GHz is shown in Figure 6-2. In the figure, channel 1 is the divide by 8 PECL output operating at 300 MHz and channel 2 is the divide by 16 TTL output operating at 150 MHz. Note from the figure that signal integrity is excellent and system noise is low.

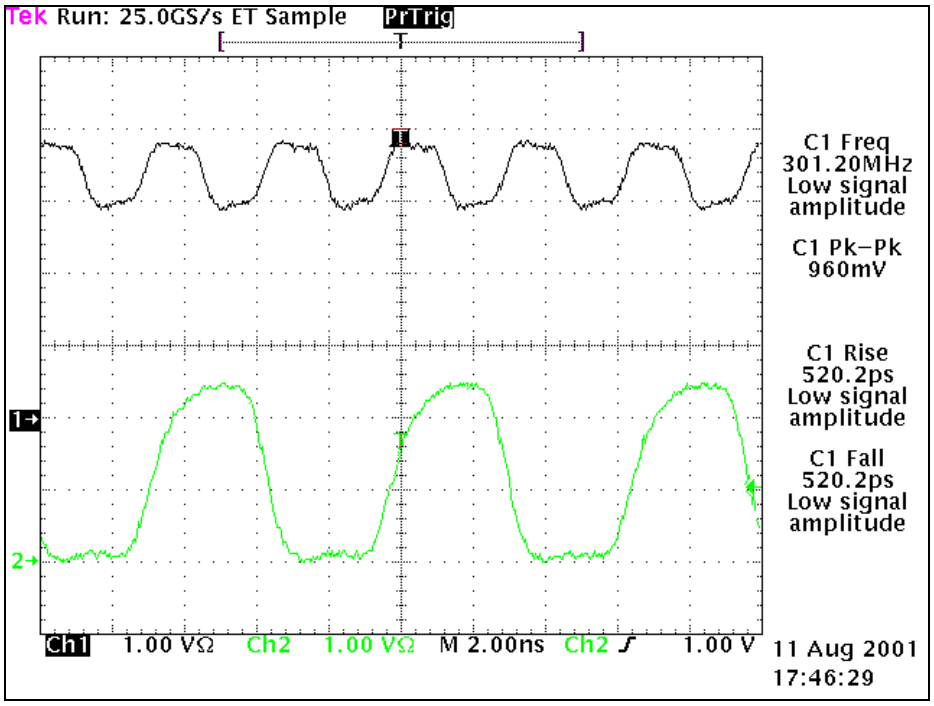


Figure 6-2 Timing System Divided Outputs With 2.4 GHz Input Frequency

6-2 ADC Card Measurements

The ADC Card was validated using the equipment setup shown in Figure 6-3. For all ADC Card tests, the input clock was a 1-GHz, 800 mVp-p sine wave. For the first test, the sine wave generator was connected to the ADC analog input and the output of the ADC card tester was viewed on the oscilloscope. For input frequencies below 32.25

MHz, the output is a sine wave with the same frequency as the input waveform, sampled at 64.5 MHz (since only 1 out of 16 samples is being displayed). This test was repeated for all 16 channels of the ADC card and all were shown to be functional. The oscilloscope output of this test was shown in Figure 5-7.

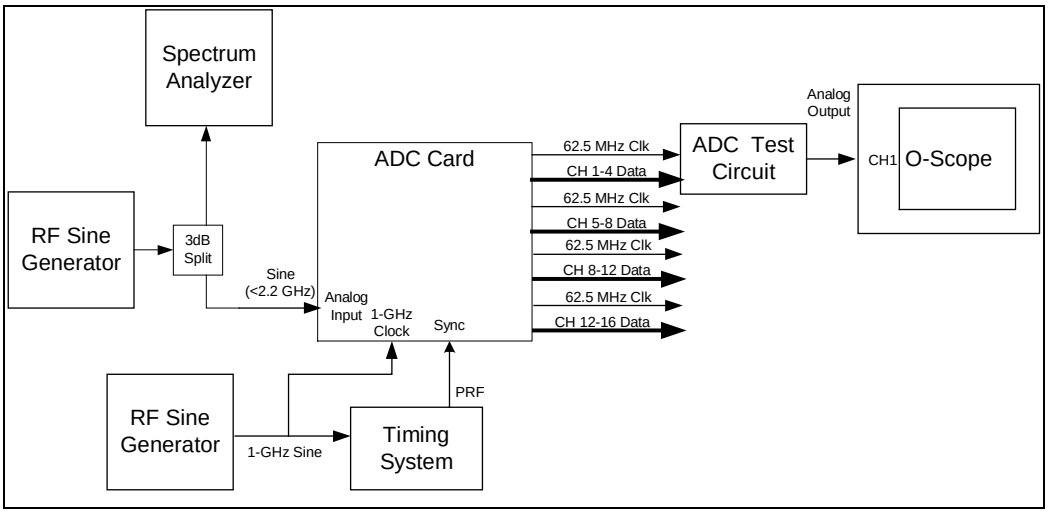


Figure 6-3 ADC Card Validation Setup

To verify the 2.2 GHz input bandwidth of the ADC card, the input frequency of the sine wave generator was manually swept from 64 MHz to 2.5 GHz. In this test the ADC test card output is an analog representation of the ADC digital output. The output frequency of the ADC tester "aliases" every 32.25 MHz since only 1 out of 16 samples is being displayed. To obtain accurate measurements, the input frequency was set approximately 1 MHz above the "alias" frequencies to obtain a stable 1-MHz output signal with small step sizes. At each frequency, the input voltage was measured using the spectrum analyzer and the output voltage of the ADC tester was measured using the oscilloscope. Since the output DAC of the ADC tester does not output the exact voltage levels of the input analog waveform, the output voltage must be scaled for comparison. To accurately compare the input and output voltages, the output voltage was scaled by a

factor of 1.127 which corresponds to 0 dB loss at a frequency of 65.5 MHz. The raw and processed data are shown in Table 6-1 with a plot of ADC frequency response shown in Figure 6-4. Although the measurement method is not particularly precise, the figure demonstrates that the A/D converter does operate with input frequencies over 2.2 GHz and shows the same general trend as the curve specified by the manufacturer.

Freq (MHz)	Input (mVrms)	DAC Ouput (mVrms)	Compensated Output (mVrms)	Calculated Loss (dB)	Rated Loss (dB)
62.5	263	233	263	0.0	0
125	258	225	254	-0.1	
187.5	248	225	254	0.2	
250	248	215	242	-0.2	-0.1
312.5	242	216	244	0.1	
375	239	214	241	0.1	
437.5	235	209	236	0.0	
500	223	215	242	0.7	-0.25
562.5	235	208	234	0.0	
625	232	203	229	-0.1	
687.5	220	210	237	0.6	
750	222	200	225	0.1	-0.6
812.5	222	210	237	0.6	
875	206	184	207	0.1	
937.5	228	199	224	-0.2	
1000	187	185	208	1.0	-0.75
1062.5	221	182	205	-0.6	
1125	209	181	204	-0.2	
1187.5	195	189	213	0.7	
1250	209	174	196	-0.6	-1.15
1312.5	194	182	205	0.5	
1375	212	168	189	-1.0	
1437.5	208	171	192	-0.7	
1500	184	175	197	0.6	-1.4
1562.5	210	154	174	-1.7	
1625	183	165	185	0.1	
1687.5	172	147	166	-0.3	
1750	174	124	140	-1.9	-2.1
1812.5	116	110	124	0.5	
1875	109	76	85	-2.1	
1937.5	107	80	90	-1.5	
2000	102	84	95	-0.7	-2.5
2062.5	152	87	98	-3.9	
2125	149	116	130	-1.2	
2187.5	147	92	103	-3.1	-2.9
2250	181	90	101	-5.1	
2312.5	149	112	126	-1.4	
2375	160	83	94	-4.6	
2437.5	200	84	95	-6.5	-4

Table 6-1 ADC Frequency Response Data

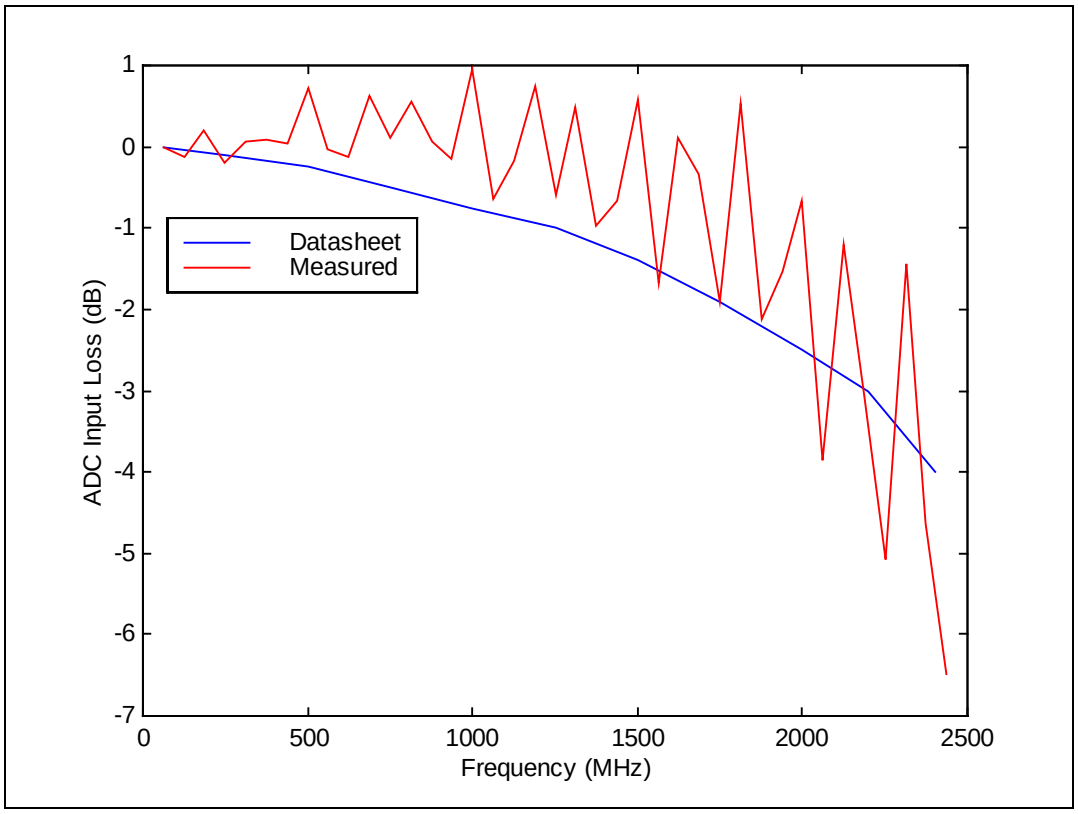


Figure 6-4 Measured vs. Datasheet ADC Analog Frequency Response

6-3 Averaging Card Measurements

The Averaging Card input section was tested using the Averaging Card tester with the configuration shown in Figure 6-5. The operation of each Average Card DSP and the input FIFOs were verified by connecting the debugging cable to the DSP and verifying the card tester output values were input properly into each DSP. This procedure validates all data lines and interrupts associated with the input section of the Averaging Card.

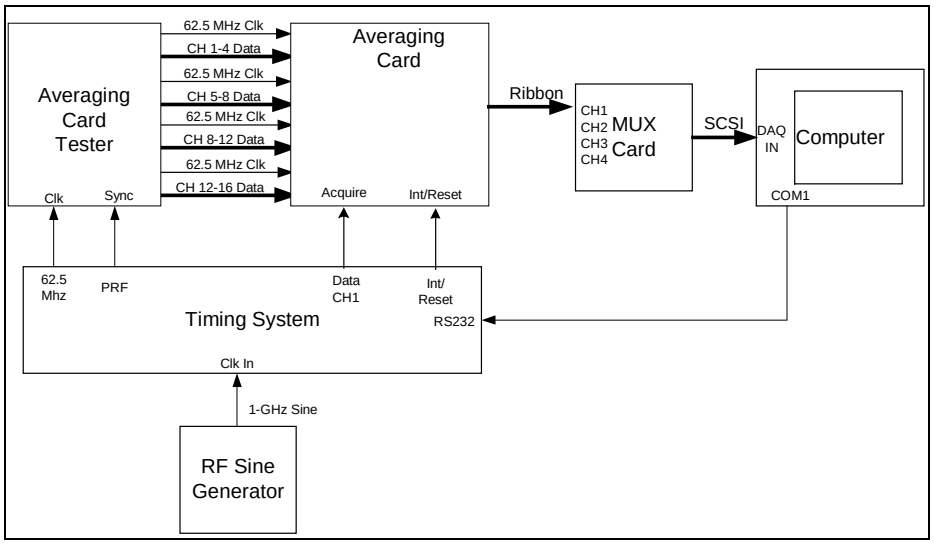


Figure 6-5 Averaging Card Test Setup

Next, the computer system was connected and the output section of the Averaging Card was validated. A special test program was written into the flash memory of each DSP so that, upon reset, a 32-bit counting sequence was output from the four DSPs. The output of this test should be a triangle wave on the acquisition display as the counting sequence runs from 0x0000000 to 0xFFFFFFF (since only 28 bits are output to the computer system). The triangle wave was displayed but would sometimes have corrupt or missing data from one or two DSPs. This problem appears to be due to inadequate soldering of the fine pitch BGAs as described in section 3-3. When all four DSPs operate properly, this test produced the correct results thus validating the logic and circuit layout of the Averaging Cards and downstream hardware.

6-4 Acquisition System Measurements

Finally, the entire system was tested as shown in Figure 6-6. The system was operated in both raw data (single shot) and averaging modes. For the raw data mode, a non-coherent function generator was used as the input source. Because of the malfunctioning Averaging Card, continuous valid results were difficult to obtain. An

entire acquisition window with 7,250 samples of a 300-kHz sine wave was obtained. A screenshot of the time and frequency domain output of this is shown in Figure 6-7 and Figure 6-8.

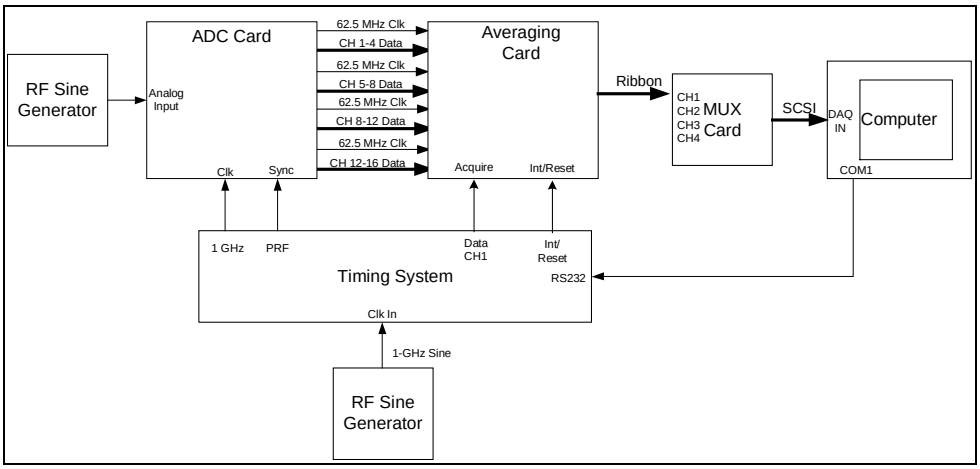


Figure 6-6 Complete System Test Setup

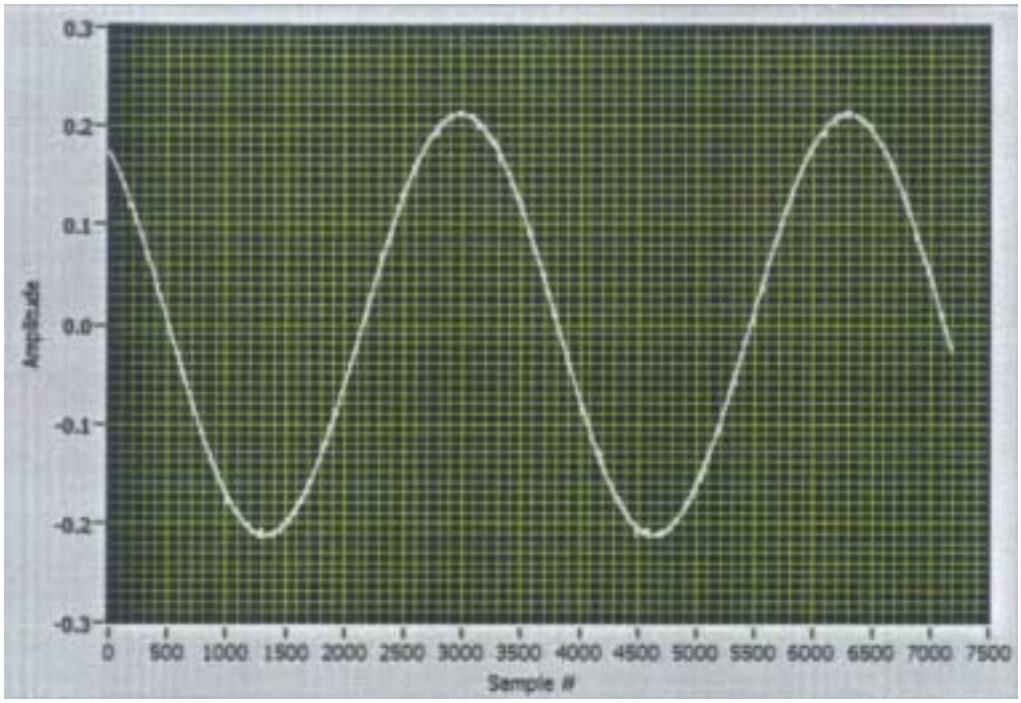


Figure 6-7 System Time Domain Output with a 300-kHz Input

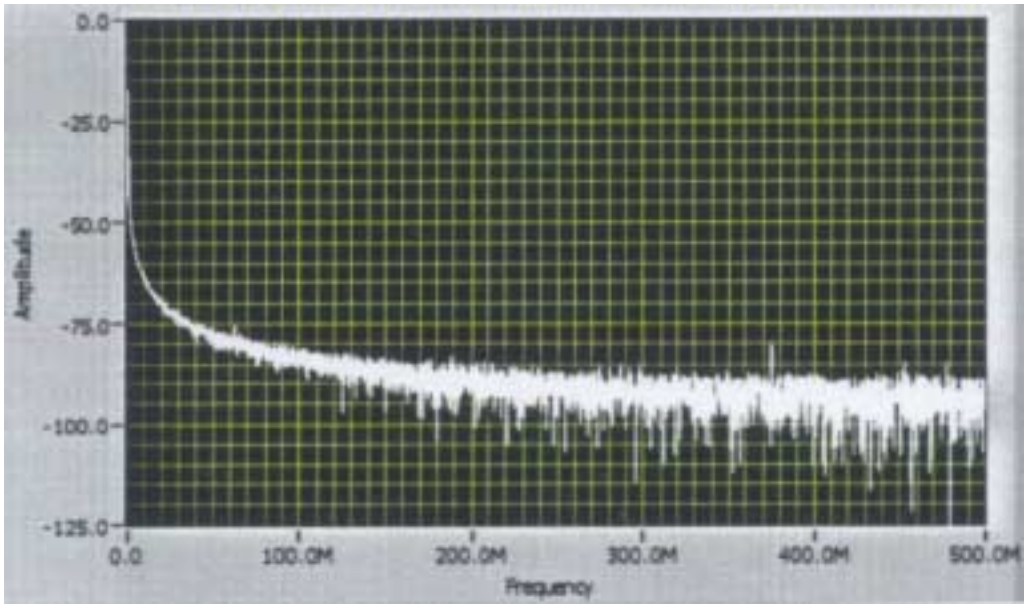


Figure 6-8 System Frequency Domain Output with a 300-kHz Input

Finally, the maximum transfer rate to the acquisition computer was measured. The computer was tested with a 300-MHz, Intel Pentium II processor running the Windows 2000 operating system with 128 MB of RAM and an IDE (33 MB/s peak) hard drive. The acquisition system was operated with the number of integrations set to 10 and the PRF was increased until the Averaging Card output FIFOs overflowed. Hard drive recording was then activated for 30 seconds. The average hard drive transfer rate was measured to be 1.3 MB/s. Based on experience using the software, it is believed that Labview is very processor intensive and is most likely the limiting factor in this transfer rate. The transfer rate should be measured in the future on a faster computer to see if this maximum rate could be increased.

CHAPTER 7 CONCLUSIONS AND RECOMMENDATIONS

Conclusion

Overall, the project was highly successful and is an excellent first attempt at a new concept. Of the five circuit boards designed, four are stable and meet or exceed specifications. The remaining board (Averaging Card) has been shown to operate intermittently due to manufacturability issues which a minor PCB revision should correct. With this PCB revision and some additional fine-tuning, the entire system will be operational and is expected to significantly impact future radar development at the University.

Lessons Learned

The following lessons were learned during this project that I would like to pass on to aid future designers of systems such as this:

1. The role of FPGAs in large circuit board design. The averaging circuitry of the previous acquisition system was designed with EEPROM based programmable logic. That design was rejected for the new Averaging Card because it does not scale reasonably to 16 data paths. Field programmable gate arrays (FPGAs) are designed for this exact situation, when it is necessary to pack a large amount of programmable logic into a single device. With today's average sized FPGA it would be possible to implement 4 to 8 Averaging Card data paths in a single programmable chip. This would greatly simplify the board design and would likely process data faster than the rate of the current ADC Card output.

2. Recognizing the scale of a project. In hindsight, I feel that this project was too large for a single person. At the point in the project where I recognized that this would

require five complex circuit boards, I should have broken the project into two parts and left the second half for another person. The timing of this project was such that if a single card could not be made to work, there would be no time for another revision. A more proper way to scale this project would have been to allow enough time for a complete redesign of one PCB.

3. The importance of designing for manufacturability. As a new designer, it is exciting to work with the latest trends and technologies. This was taken too far with the Averaging Card and resulted in a PCB that few manufacturers could produce, even in prototype quantities. The lesson learned here is, all things being equal, the technology that is the most manufacturable should be selected. If space is not a limitation, choose the largest packaging that will perform at the desired speed. When given a choice between a fine pitch flat pack, fine pitch BGA, and standard BGA, the best overall choice is the standard BGA package because it is the easiest to manufacture with.

Suggestions for the Future

In light of the degraded performance of the DSP based Averaging Card, I would propose an FPGA based Averaging Card for future system revisions. A block diagram of a possible circuit is shown in Figure 7-1. This is basically an extension of the current DSP-RAM based Averaging Card with the DSPs replaced by FPGAs. With current FPGA speeds, this circuit should be limited entirely by the transfer rate of the RAM. If double data rate RAM is used, the circuit could actually process data at rates higher than the ADC Card output. This circuit has an additional advantage in that it can average multiple waveforms by storing each waveform in a different section of RAM. This would enabling a radar to change it's transmit waveform on consecutive PRF pulses for

maximum diversity. The number of distinct waveforms would be limited only by the size of the RAM.

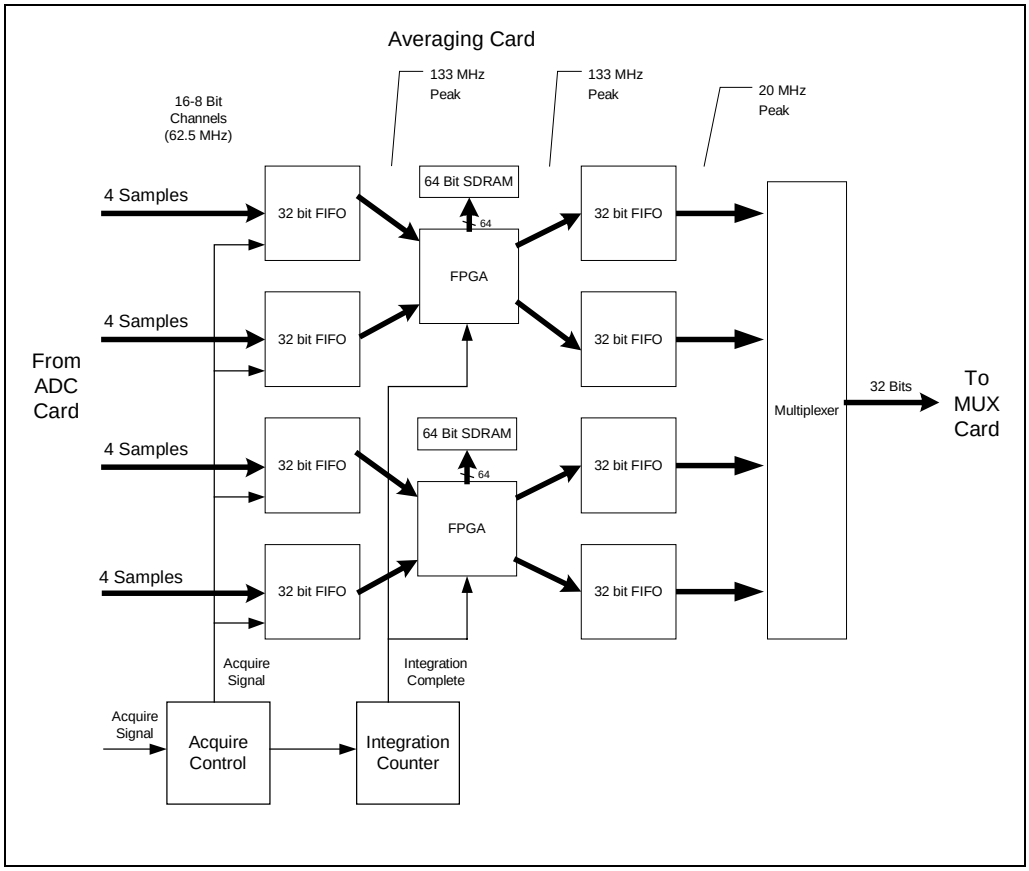


Figure 7-1 FPGA Based Averaging Card Design

REFERENCES

1. Kanagaratnam, P., S. P. Gogineni, N. Gundestrup, and L. Larsen, "High-Resolution Radar Mapping of Internal Layers at NGRIP," submitted July 2000 to J. Geophysical Research (Climate & Physics of the Atmosphere) special issue on Program for Arctic Regional Climate Assessment (PARCA).

2. Wepman, J.A., Hoffman, J.R., “RF and IF Digitaization in Radio Receivers: Theory, Concepts, and Examples,” NTIA Report 96-328, March, 1996.

3. Gröchenig, Karlheinz, “Efficient Algorithms in Irregular Sampling of Band-Limited Functions,” 10. International Phoenix Conference on Computers and Communication 1991, pp. 490-495.

4. Feichtinger, H. G. and Strohmer, T., “Fast Iterative Reconstruction of Band-limited Images from Non-Uniform Sampling Values,” Proceedings of the Computer Analysis of Images and Patterns (CAIP) Conference (Budapest, 1993), pp. 82-91.

5. Maxim Integrated Products, “MAX104 Datasheet,” Sunnyvale, CA 94086, 1999.

6. Maxim Integrated Products, “MAX104 Evaluation Board Datasheet,” Sunnyvale, CA 94086, 1999.

7. Johson, H., Graham, M., “High Speed Digital Design,” Olympic Technology Group, Redmond, WA, 1993.

8. Agilent Technologies, Advanced Design System V 1.5 (software), Palo Alto, CA 94304, 2000.

9. Integrated Device Technology, Inc, “IDT72V263 Datasheet,” www.idt.com, 1999.

10. Paul, C. R., “Introduction to Electromagnetic Compatibility,” Wiley Publications, 1992.

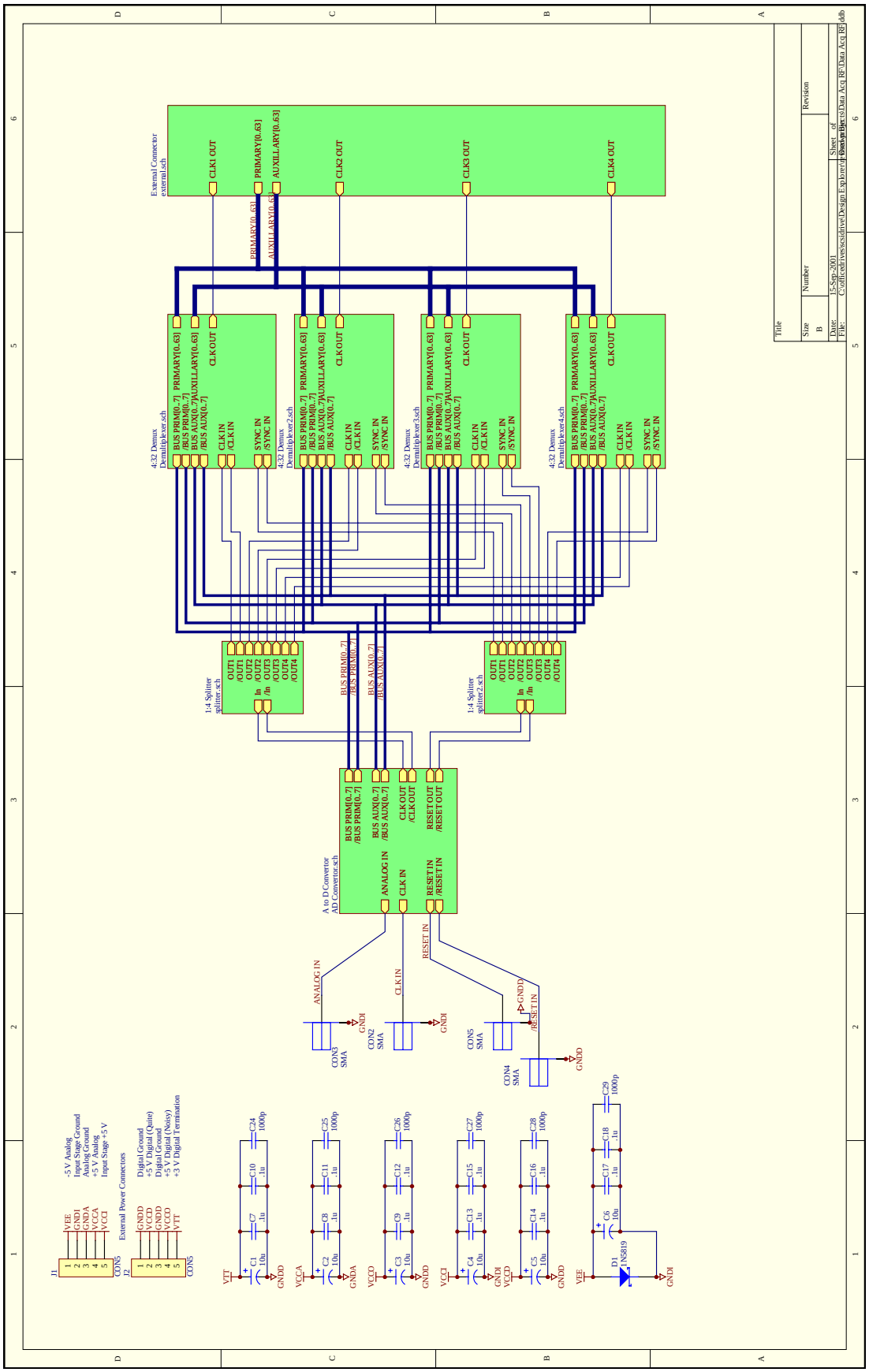
11. Atmel Corporation, “AT29LV010A Datasheet,” www.atmel.com, 1998.

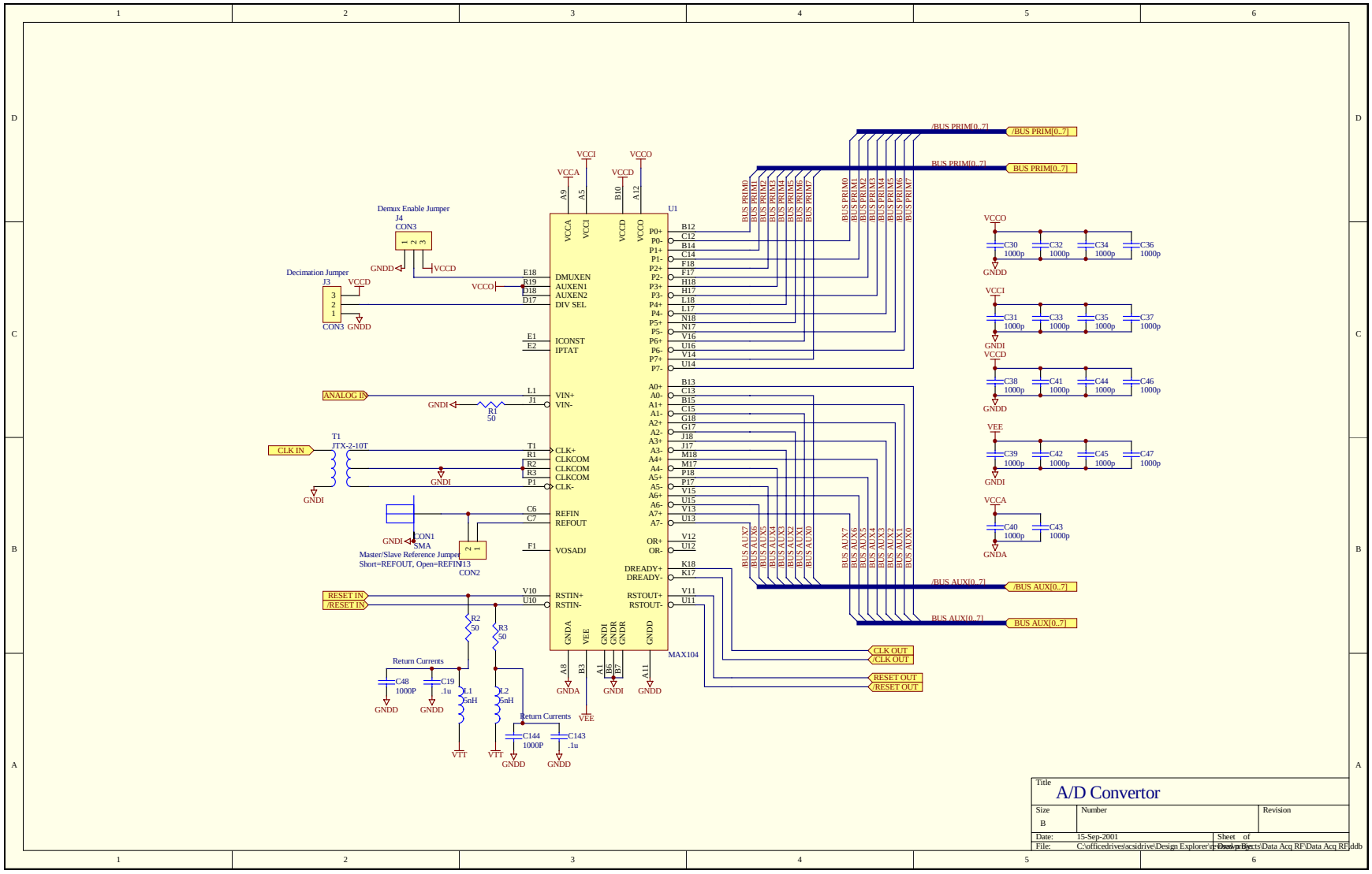
12. Texas Instruments, Inc., “TMS320C6000 Peripherals Reference Guide,” www.ti.com, 1999.

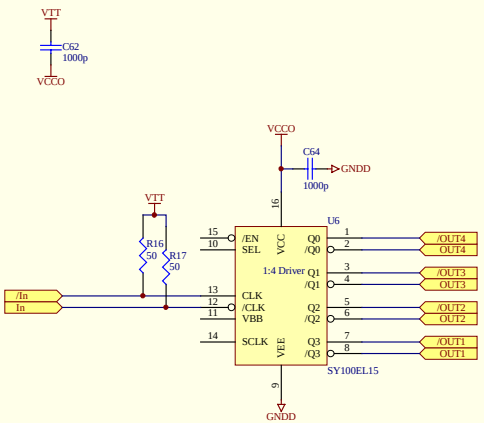
13. Fairchild Semiconductor, “AN-768 ECL Backplane Design,” www.fairchildsemi.com, 1991.

14. Maxim Integrated Products, “HFAN-1.0 Introduction to LVDS, PECL, and CML,” Sunnyvale, CA 94086, 2000.

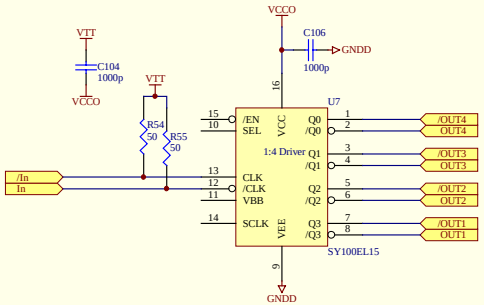
APPENDIX A ADC CARD SCHEMATICS


External Connector
external.sch
CLK1 OUT
PRIMARY[0..63]
AUXILLARY[0..63]
CLK2 OUT
CLK3 OUT
CLK4 OUT
4:32 Demux
Demultiplexer.sch
4:32 Demux
Demultiplexer2.sch
4:32 Demux
Demultiplexer3.sch
4:32 Demux
Demultiplexer4.sch
BUS PRIM[0..7] PRIMARY[0..63]
/BUS PRIM[0..7]
BUS AUX[0..7]AUXILLARY[0..63]
/BUS AUX[0..7]
CLK OUT
CLK IN
/CLK IN
SYNC IN
/SYNC IN
1:4 Splitter
splitter.sch
1:4 Splitter
splitter2.sch
OUT1
/OUT1
OUT2
/OUT2
OUT3
/OUT3
OUT4
/OUT4
In
/In
BUS PRIM[0..7]
/BUS PRIM[0..7]
BUS AUX[0..7]
/BUS AUX[0..7]
A to D Convertor
AD Convertor.sch
ANALOG IN
CLK IN
RESET IN
/RESET IN
CLK OUT
/CLK OUT
RESET OUT
/RESET OUT
CON3
SMA
CON2
SMA
CON5
SMA
CON4
SMA
GNDI
GNDD
J1
J2
CON5
-5 V Analog
Inverter Signal Ground
Analog Ground
+5 V Analog
Input Stage +5 V
External Power Connectors
Digital Ground
+5 V Digital (Quite)
Digital Ground
+5 V Digital (Noisy)
+3 V Digital Termination
VEE
GNDI
GNDA
VCCA
VCCI
GNDD
VCCD
VCCO
VTT
C1
C2
C3
C4
C5
C6
C7
C8
C9
C10
C11
C12
C13
C14
C15
C16
C17
C18
C24
C25
C26
C27
C28
C29
10u
.1u
1000p
D1
IN5819
Title
Size
B
Number
Revision
Date:
15-Sep-2001
Sheet
of
File:
C:\officelt\revisesitive\Design Explorer 99SE\Design Projects\Data Acq RF\Data Acq RF.ddb

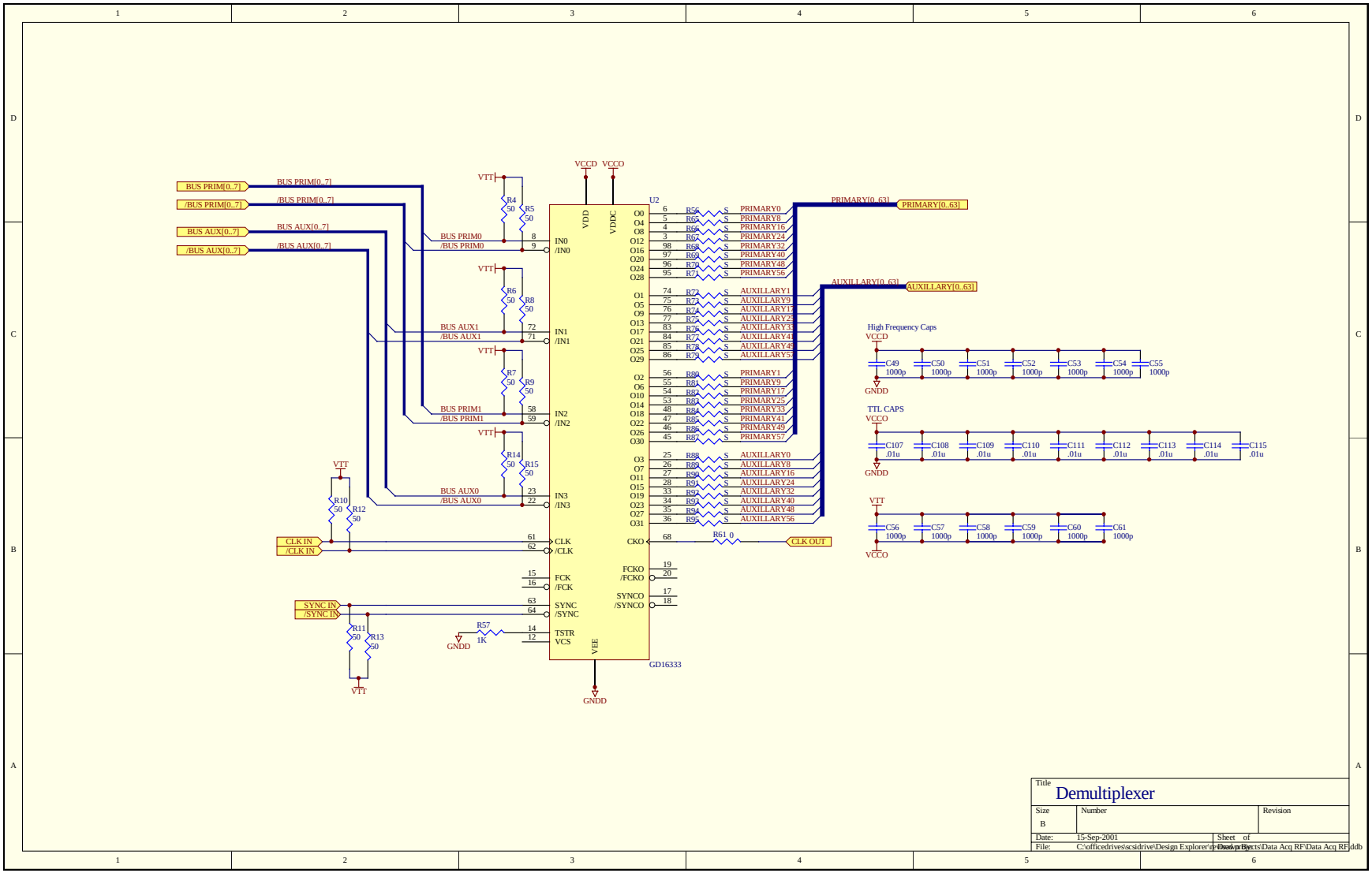

Demux Enable Jumper
J4
CON3
GNDD
VCCD
Decimation Jumper
J3
CON3
VCCD
GNDD
VCCO
DMUXEN
AUXEN1
AUXEN2
DIV SEL
ICONST
IPTAT
ANALOG IN
GNDI
R1
50
VIN+
VIN-
T1
JTX-2-10T
CLK IN
CLK+
CLKCOM
CLKCOM
CLKCOM
CLK-
C6
C7
REFIN
REFOUT
CON1
SMA
Master/Slave Reference Jumper
Short=REFOUT, Open=REFIN
CON2
VOSADJ
RESET IN
/RESET IN
RSTIN+
RSTIN-
R2
50
R3
50
Return Currents
C48
1000P
C19
.1u
L1
5nH
L2
5nH
VTT
C144
1000P
C143
.1u
VCCI
VCCA
VCCD
VCCO
VEE
GNDA
GNDI
GNDD
U1
MAX104
P0+
P0-
P1+
P1-
P2+
P2-
P3+
P3-
P4+
P4-
P5+
P5-
P6+
P6-
P7+
P7-
A0+
A0-
A1+
A1-
A2+
A2-
A3+
A3-
A4+
A4-
A5+
A5-
A6+
A6-
A7+
A7-
OR+
OR-
DREADY+
DREADY-
RSTOUT+
RSTOUT-
CLK OUT
/CLK OUT
RESET OUT
/RESET OUT
/BUS PRIM[0..7]
BUS PRIM[0..7]
/BUS AUX[0..7]
BUS AUX[0..7]
C30
C32
C34
C36
C31
C33
C35
C37
C38
C41
C44
C46
C39
C42
C45
C47
C40
C43
1000p
Title
A/D Convertor
Size
B
Number
Revision
Date:
15-Sep-2001
Sheet of
File:
C:\officedrives\scsidrive\Design Explorer\Projects\Data Acq RF\Data Acq RF.ddb



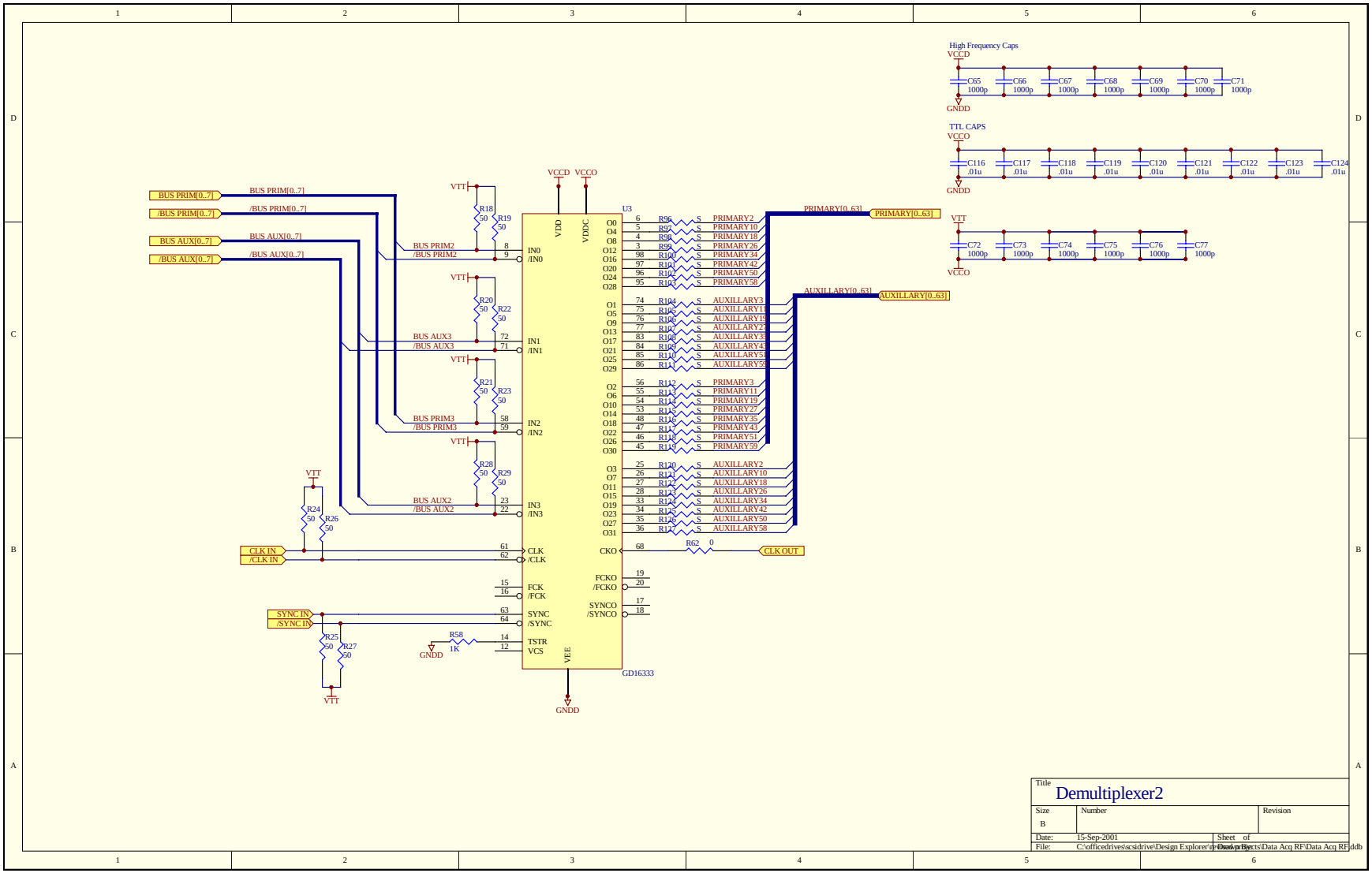
Title: Splitter			
Size: B	Number:	Revision:	
Date: 15-Sep-2001	Sheet of		
File: C:\officedrives\scsidrive\Design Explorer\projects\Data Acq RF\Data Acq RF.ddb	Drawn By:		

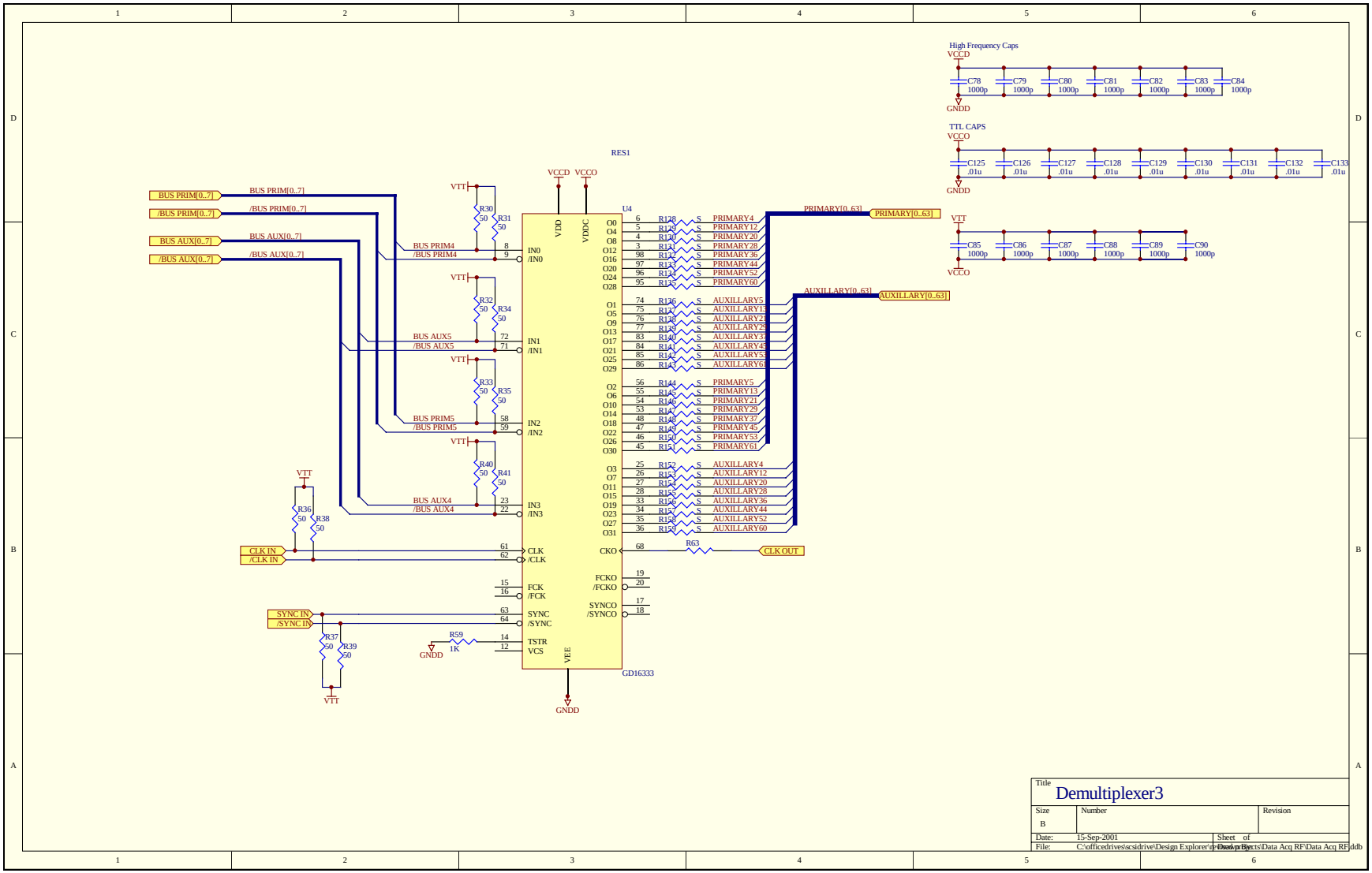


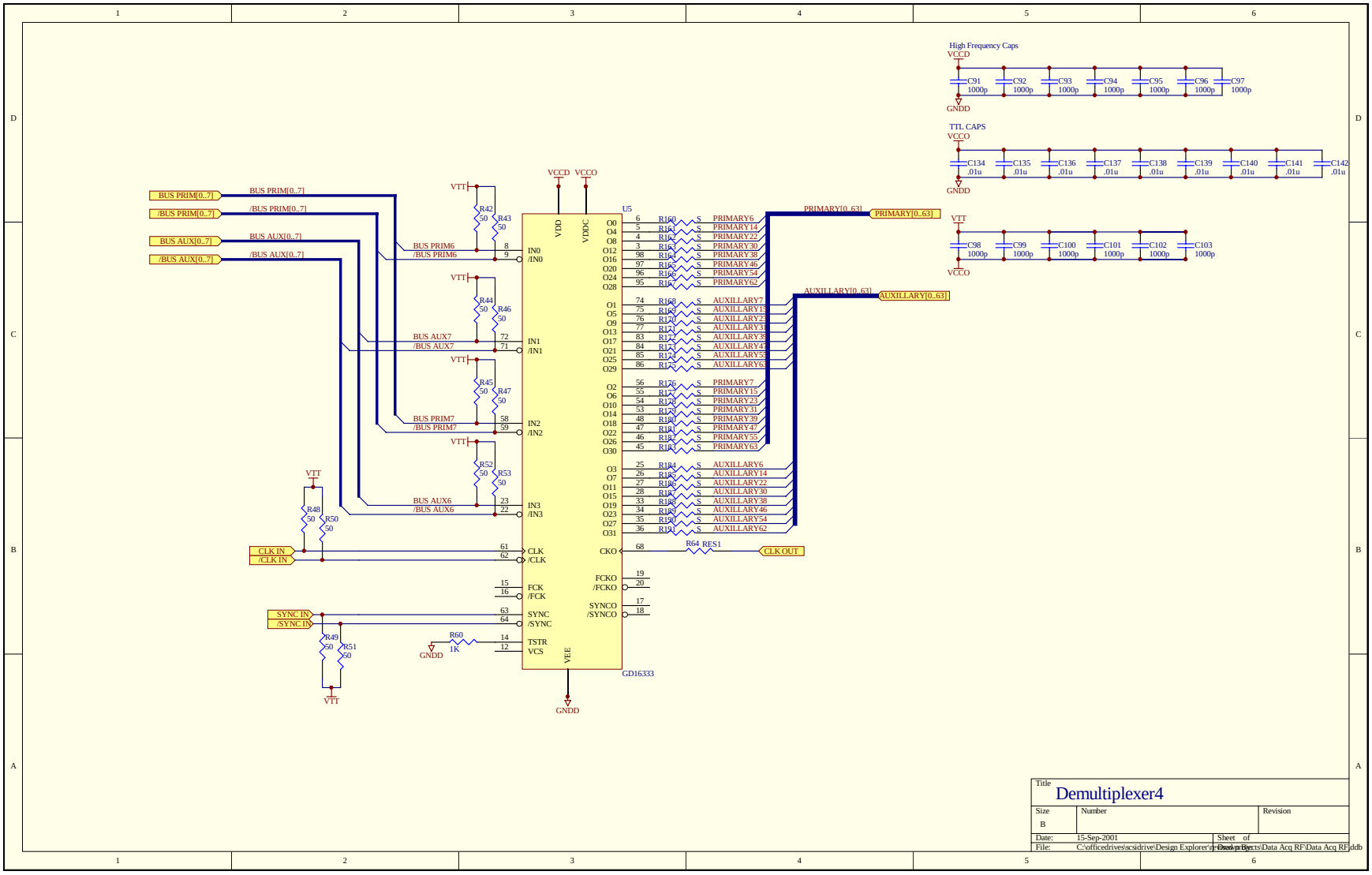
Title: Splitter2		
Size: B	Number:	Revision:
Date: 15-Sep-2001	Sheet of	
File: C:\officedrives\scsidrive\Design Explorer\Projects\Data Acq RF\Data Acq RF.ddb	Drawn By:	

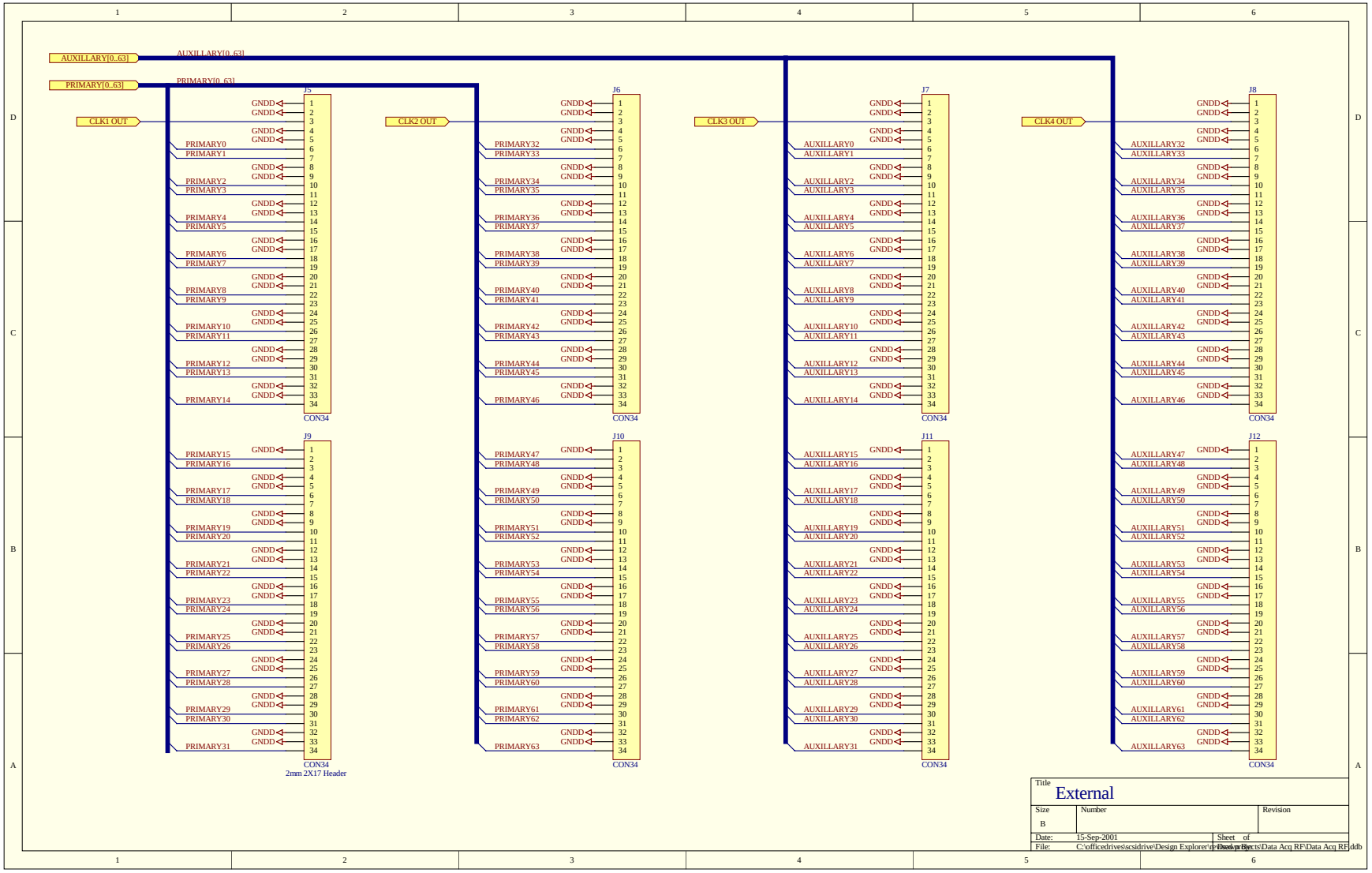


Title	Demultiplexer
Size	B
Number	
Revision	
Date:	15-Sep-2001
Sheet	of
File:	C:\officedrives\scsidrive\Design Explorer\projects\Data Acq RF\Data Acq RF.ddb

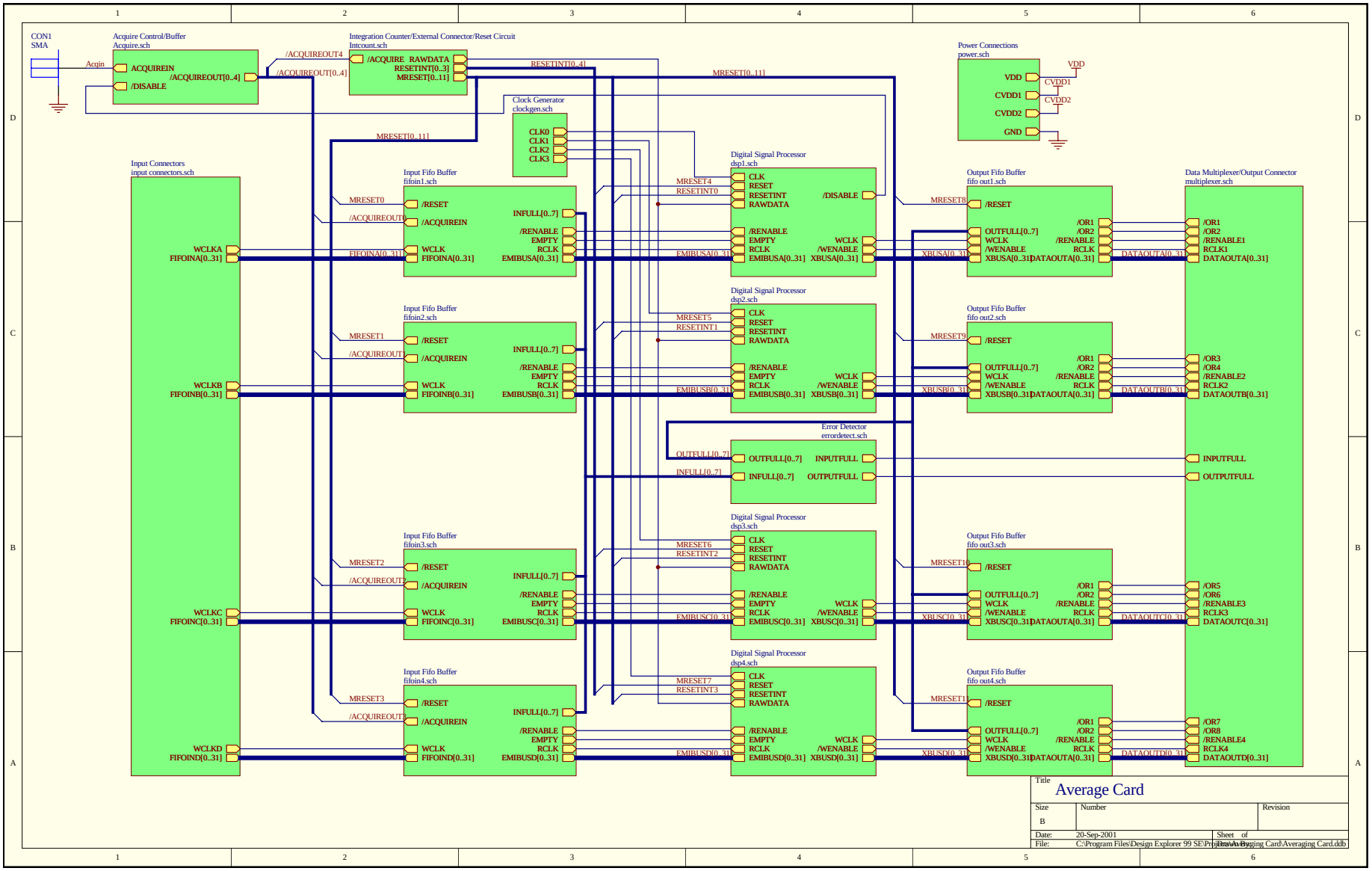

High Frequency Caps
VCCD
C65 1000p
C66 1000p
C67 1000p
C68 1000p
C69 1000p
C70 1000p
C71 1000p
GNDD
TTL CAPS
VCCO
C116 .01u
C117 .01u
C118 .01u
C119 .01u
C120 .01u
C121 .01u
C122 .01u
C123 .01u
C124 .01u
GNDD
VTT
C72 1000p
C73 1000p
C74 1000p
C75 1000p
C76 1000p
C77 1000p
VCCO
BUS PRIM[0..7]
/BUS PRIM[0..7]
BUS AUX[0..7]
/BUS AUX[0..7]
BUS PRIM2
/BUS PRIM2
BUS AUX3
/BUS AUX3
BUS PRIM3
/BUS PRIM3
BUS AUX2
/BUS AUX2
R18 50
R19 50
R20 50
R22 50
R21 50
R23 50
R28 50
R29 50
R24 50
R26 50
IN0 /IN0
IN1 /IN1
IN2 /IN2
IN3 /IN3
CLK IN
/CLK IN
CLK /CLK
FCK /FCK
SYNC IN
/SYNC IN
SYNC /SYNC
R25 50
R27 50
R58 1K
TSTR VCS
VCCD VCCO
VDD VDDC VEE
GNDD
U3
GD16333
O0 O4 O8 O12 O16 O20 O24 O28
O1 O5 O9 O13 O17 O21 O25 O29
O2 O6 O10 O14 O18 O22 O26 O30
O3 O7 O11 O15 O19 O23 O27 O31
CKO
FCKO /FCKO
SYNCO /SYNCO
R96 R97 R98 R99 R100 R101 R102 R103
R104 R105 R106 R107 R108 R109 R110 R111
R112 R113 R114 R115 R116 R117 R118 R119
R120 R121 R122 R123 R124 R125 R126 R127
R62 0
PRIMARY2 PRIMARY10 PRIMARY18 PRIMARY26 PRIMARY34 PRIMARY42 PRIMARY50 PRIMARY58
AUXILLARY3 AUXILLARY11 AUXILLARY19 AUXILLARY27 AUXILLARY35 AUXILLARY43 AUXILLARY51 AUXILLARY59
PRIMARY3 PRIMARY11 PRIMARY19 PRIMARY27 PRIMARY35 PRIMARY43 PRIMARY51 PRIMARY59
AUXILLARY2 AUXILLARY10 AUXILLARY18 AUXILLARY26 AUXILLARY34 AUXILLARY42 AUXILLARY50 AUXILLARY58
PRIMARY[0..63]
AUXILLARY[0..63]
CLK OUT
Title
Demultiplexer2
Size B
Number
Revision
Date: 15-Sep-2001
Sheet of
File: C:\officedrives\scsidrive\Design Explorer\projects\Data Acq RF\Data Acq RF.ddb

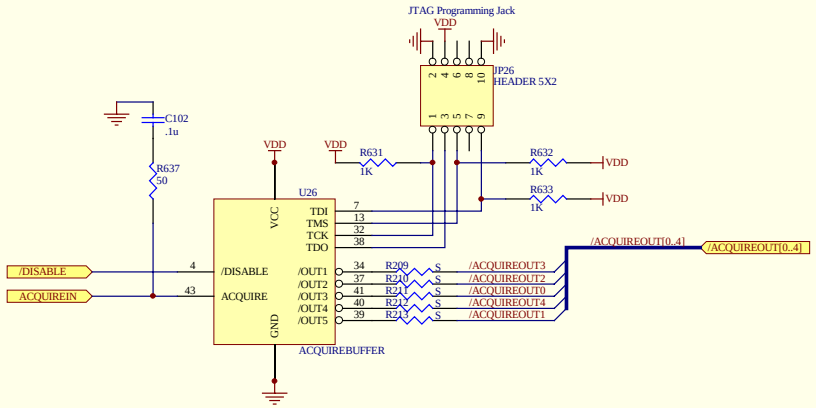

High Frequency Caps
VCCD
C78 1000p
C79 1000p
C80 1000p
C81 1000p
C82 1000p
C83 1000p
C84 1000p
GNDD
TTL CAPS
VCCO
C125 .01u
C126 .01u
C127 .01u
C128 .01u
C129 .01u
C130 .01u
C131 .01u
C132 .01u
C133 .01u
GNDD
VTT
C85 1000p
C86 1000p
C87 1000p
C88 1000p
C89 1000p
C90 1000p
VCCO
RES1
U4
GD16333
BUS PRIM[0..7]
/BUS PRIM[0..7]
BUS AUX[0..7]
/BUS AUX[0..7]
BUS PRIM4
/BUS PRIM4
BUS AUX5
/BUS AUX5
BUS PRIM5
/BUS PRIM5
BUS AUX4
/BUS AUX4
IN0
/IN0
IN1
/IN1
IN2
/IN2
IN3
/IN3
CLK IN
/CLK IN
CLK
/CLK
FCK
/FCK
SYNC IN
/SYNC IN
SYNC
/SYNC
TSTR
VCS
VDD
VDDC
VEE
FCKO
/FCKO
SYNCO
/SYNCO
CKO
CLK OUT
PRIMARY[0..63]
AUXILLARY[0..63]
PRIMARY4
PRIMARY12
PRIMARY20
PRIMARY28
PRIMARY36
PRIMARY44
PRIMARY52
PRIMARY60
AUXILLARY5
AUXILLARY13
AUXILLARY21
AUXILLARY29
AUXILLARY37
AUXILLARY45
AUXILLARY53
AUXILLARY61
PRIMARY5
PRIMARY13
PRIMARY21
PRIMARY29
PRIMARY37
PRIMARY45
PRIMARY53
PRIMARY61
AUXILLARY4
AUXILLARY12
AUXILLARY20
AUXILLARY28
AUXILLARY36
AUXILLARY44
AUXILLARY52
AUXILLARY60
GNDD
VTT
VCCD
VCCO
Title
Demultiplexer3
Size
B
Number
Revision
Date: 15-Sep-2001
Sheet of
File: C:\officedrives\scsidrive\Design Explorer\projects\Data Acq RF\Data Acq RF.ddb


High Frequency Caps
VCCD
C91 1000p
C92 1000p
C93 1000p
C94 1000p
C95 1000p
C96 1000p
C97 1000p
GNDD
TTL CAPS
VCCO
C134 .01u
C135 .01u
C136 .01u
C137 .01u
C138 .01u
C139 .01u
C140 .01u
C141 .01u
C142 .01u
GNDD
VTT
C98 1000p
C99 1000p
C100 1000p
C101 1000p
C102 1000p
C103 1000p
VCCO
BUS PRIM[0..7]
/BUS PRIM[0..7]
BUS AUX[0..7]
/BUS AUX[0..7]
VTT
R42 50
R43 50
R44 50
R46 50
R45 50
R47 50
R52 50
R53 50
R48 50
R50 50
R49 50
R51 50
BUS PRIM6
/BUS PRIM6
BUS AUX7
/BUS AUX7
BUS PRIM7
/BUS PRIM7
BUS AUX6
/BUS AUX6
CLK IN
/CLK IN
SYNC IN
/SYNC IN
R60 1K
GNDD
VCCD
VCCO
VDD
VDDC
VEE
U5
IN0
/IN0
IN1
/IN1
IN2
/IN2
IN3
/IN3
CLK
/CLK
FCK
/FCK
SYNC
/SYNC
TSTR
VCS
GD16333
O0 O4 O8 O12 O16 O20 O24 O28
O1 O5 O9 O13 O17 O21 O25 O29
O2 O6 O10 O14 O18 O22 O26 O30
O3 O7 O11 O15 O19 O23 O27 O31
CKO
FCKO
/FCKO
SYNCO
/SYNCO
R160 PRIMARY6
R161 PRIMARY14
R162 PRIMARY22
R163 PRIMARY30
R164 PRIMARY38
R165 PRIMARY46
R166 PRIMARY54
R167 PRIMARY62
R168 AUXILLARY7
R169 AUXILLARY15
R170 AUXILLARY23
R171 AUXILLARY31
R172 AUXILLARY39
R173 AUXILLARY47
R174 AUXILLARY55
R175 AUXILLARY63
R176 PRIMARY7
R177 PRIMARY15
R178 PRIMARY23
R179 PRIMARY31
R180 PRIMARY39
R181 PRIMARY47
R182 PRIMARY55
R183 PRIMARY63
R184 AUXILLARY6
R185 AUXILLARY14
R186 AUXILLARY22
R187 AUXILLARY30
R188 AUXILLARY38
R189 AUXILLARY46
R190 AUXILLARY54
R191 AUXILLARY62
R64 RES1
PRIMARY[0..63]
AUXILLARY[0..63]
CLK OUT
Title
Demultiplexer4
Size
B
Number
Revision
Date: 15-Sep-2001
Sheet of
File: C:\officedrives\scsidrive\Design Explorer\Projects\Data Acq RF\Data Acq RF.ddb

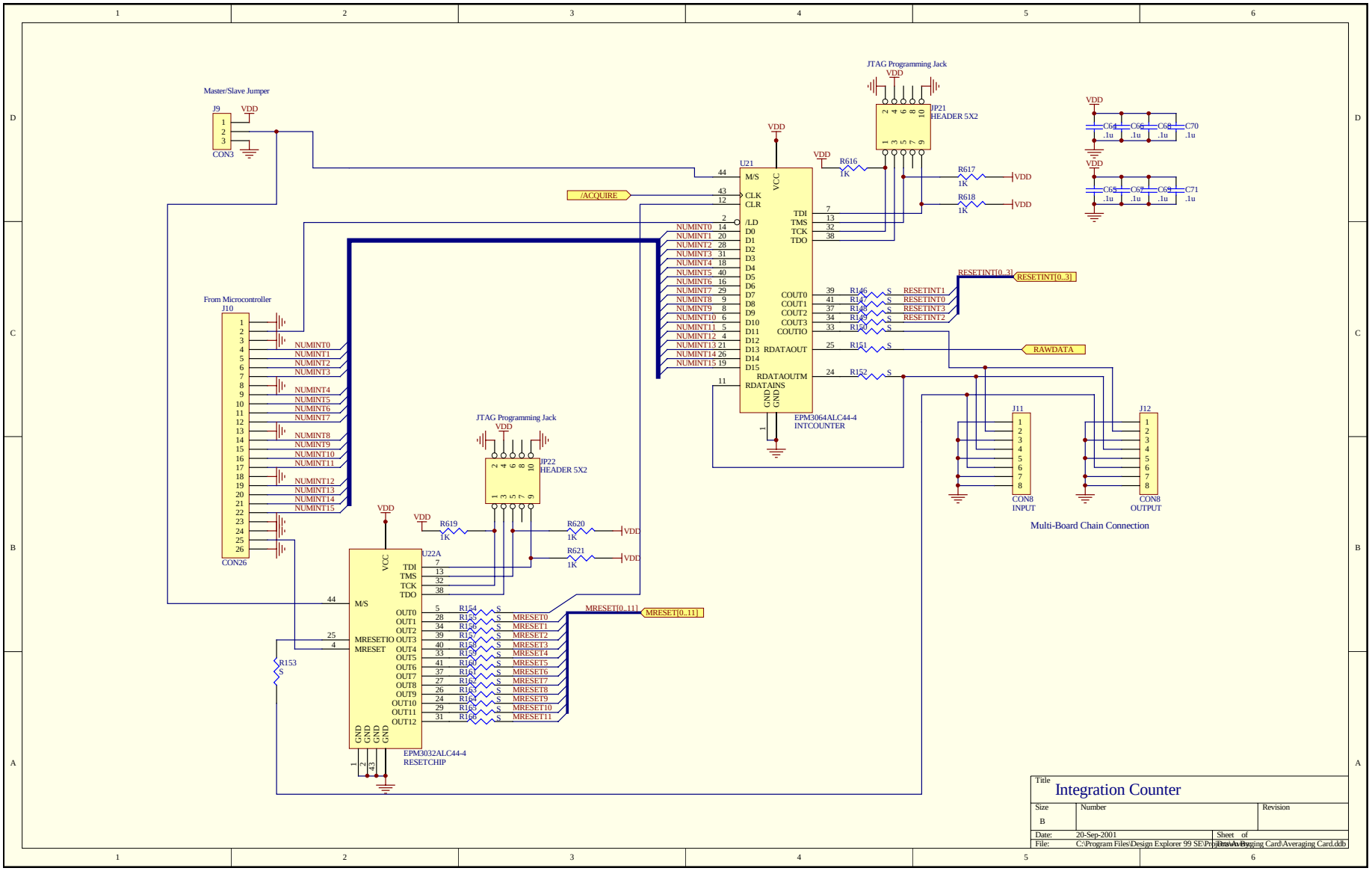

AUXILLARY[0..63]
PRIMARY[0..63]
CLK1 OUT
CLK2 OUT
CLK3 OUT
CLK4 OUT
J5
J6
J7
J8
J9
J10
J11
J12
CON34
2mm 2X17 Header
Title
External
Size
B
Number
Revision
Date: 15-Sep-2001
File: C:\officedrives\scsidrive\Design Explorer\projects\Data Acq RF\Data Acq RF.ddb

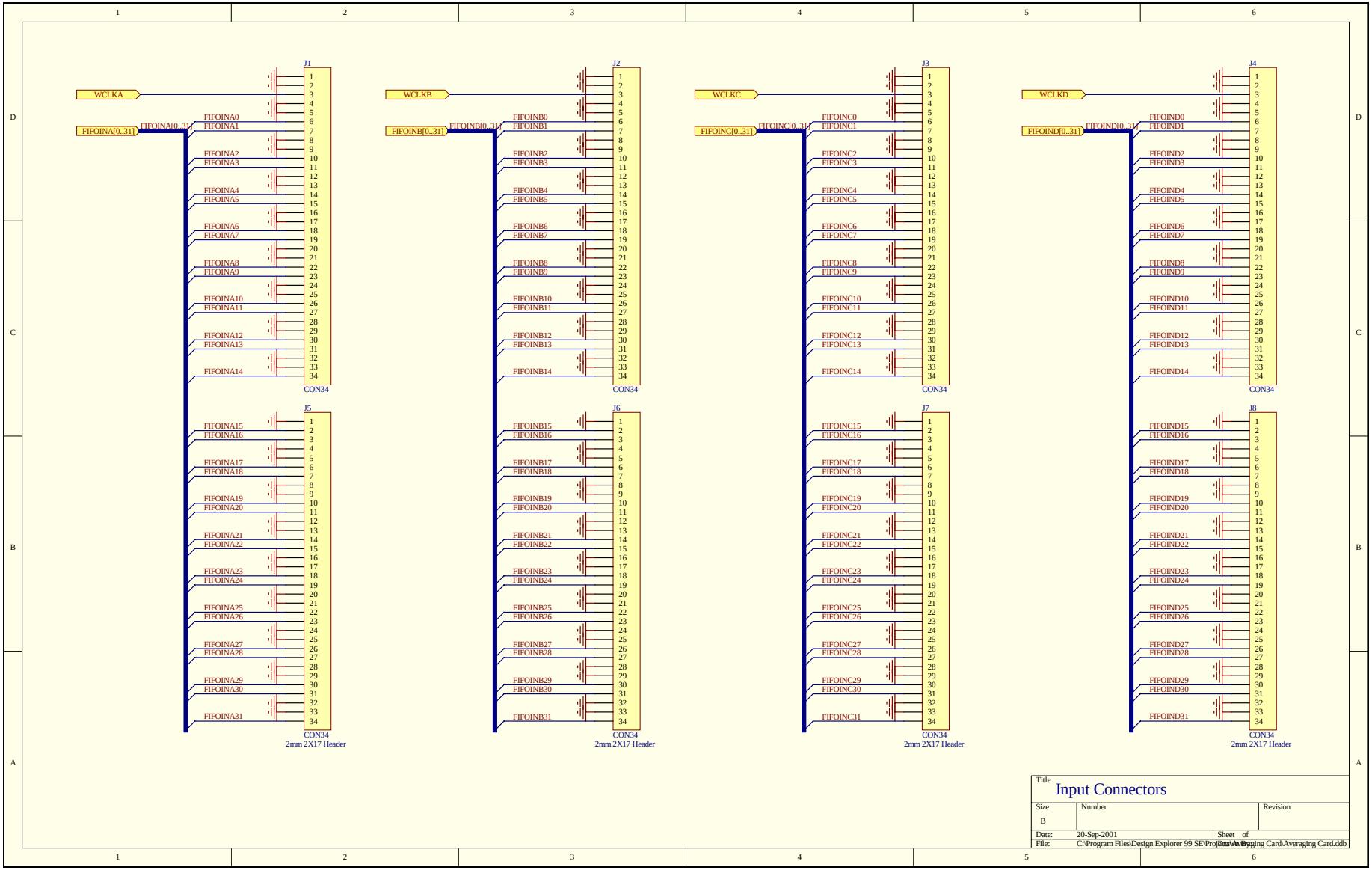
APPENDIX B AVERAGING CARD SCHEMATICS

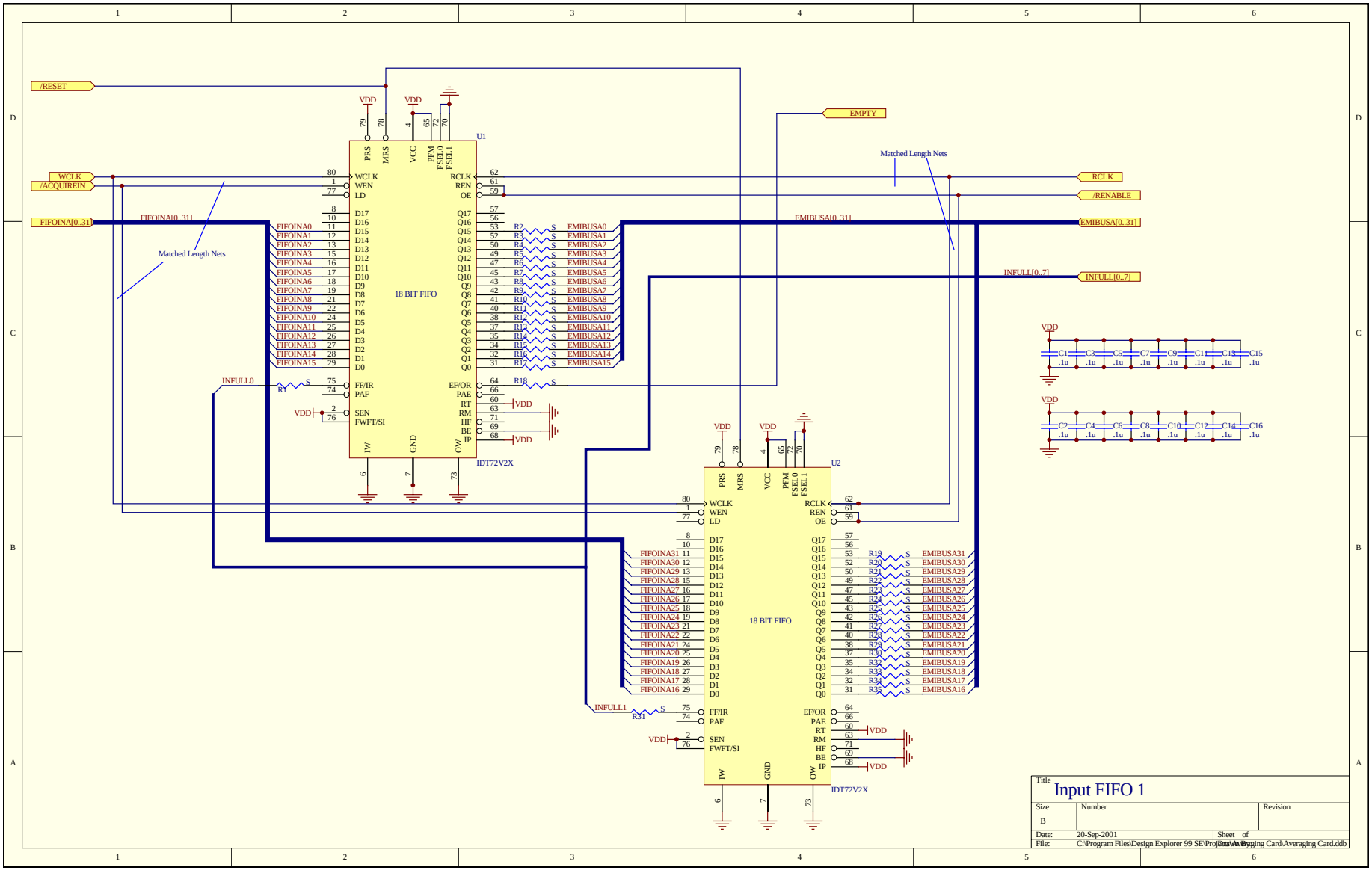

CON1
SMA
Acquin
Acquire Control/Buffer
Acquire.sch
ACQUIREIN
/ACQUIREOUT[0..4]
/DISABLE
/ACQUIREOUT4
/ACQUIREOUT[0..4]
Integration Counter/External Connector/Reset Circuit
Intcount.sch
/ACQUIRE_RAWDATA
RESETINT[0..3]
MRESET[0..11]
RESETINT[0..4]
MRESET[0..11]
Power Connections
power.sch
VDD
CVDD1
CVDD2
GND
VDD
CVDD1
CVDD2
Clock Generator
clockgen.sch
CLK0
CLK1
CLK2
CLK3
MRESET[0..11]
Input Connectors
input connectors.sch
WCLKA
FIFOINA[0..31]
WCLKB
FIFOINB[0..31]
WCLKC
FIFOINC[0..31]
WCLKD
FIFOIND[0..31]
Input Fifo Buffer
fifoin1.sch
MRESET0
/ACQUIREOUT0
/RESET
/ACQUIREIN
WCLK
FIFOINA[0..31]
INFULL[0..7]
/RENABLE
EMPTY
RCLK
EMIBUSA[0..31]
FIFOINA[0..31]
Input Fifo Buffer
fifoin2.sch
MRESET1
/ACQUIREOUT1
/RESET
/ACQUIREIN
WCLK
FIFOINB[0..31]
INFULL[0..7]
/RENABLE
EMPTY
RCLK
EMIBUSB[0..31]
Input Fifo Buffer
fifoin3.sch
MRESET2
/ACQUIREOUT2
/RESET
/ACQUIREIN
WCLK
FIFOINC[0..31]
INFULL[0..7]
/RENABLE
EMPTY
RCLK
EMIBUSC[0..31]
Input Fifo Buffer
fifoin4.sch
MRESET3
/ACQUIREOUT3
/RESET
/ACQUIREIN
WCLK
FIFOIND[0..31]
INFULL[0..7]
/RENABLE
EMPTY
RCLK
EMIBUSD[0..31]
Digital Signal Processor
dsp1.sch
MRESET4
RESETINT0
CLK
RESET
RESETINT
RAWDATA
/DISABLE
/RENABLE
EMPTY
RCLK
WCLK
/WENABLE
EMIBUSA[0..31]
XBUSA[0..31]
EMIBUSA[0..31]
Digital Signal Processor
dsp2.sch
MRESET5
RESETINT1
CLK
RESET
RESETINT
RAWDATA
/RENABLE
EMPTY
RCLK
WCLK
/WENABLE
EMIBUSB[0..31]
XBUSB[0..31]
EMIBUSB[0..31]
Error Detector
errordetect.sch
OUTFULL[0..7]
INFULL[0..7]
OUTFULL[0..7]
INFULL[0..7]
INPUTFULL
OUTPUTFULL
Digital Signal Processor
dsp3.sch
MRESET6
RESETINT2
CLK
RESET
RESETINT
RAWDATA
/RENABLE
EMPTY
RCLK
WCLK
/WENABLE
EMIBUSC[0..31]
XBUSC[0..31]
EMIBUSC[0..31]
Digital Signal Processor
dsp4.sch
MRESET7
RESETINT3
CLK
RESET
RESETINT
RAWDATA
/RENABLE
EMPTY
RCLK
WCLK
/WENABLE
EMIBUSD[0..31]
XBUSD[0..31]
EMIBUSD[0..31]
Output Fifo Buffer
fifo out1.sch
MRESET8
/RESET
OUTFULL[0..7]
WCLK
/WENABLE
XBUSA[0..31]
/OR1
/OR2
/RENABLE
RCLK
DATAOUTA[0..31]
XBUSA[0..31]
DATAOUTA[0..31]
Output Fifo Buffer
fifo out2.sch
MRESET9
/RESET
OUTFULL[0..7]
WCLK
/WENABLE
XBUSB[0..31]
/OR1
/OR2
/RENABLE
RCLK
DATAOUTA[0..31]
XBUSB[0..31]
DATAOUTB[0..31]
Output Fifo Buffer
fifo out3.sch
MRESET10
/RESET
OUTFULL[0..7]
WCLK
/WENABLE
XBUSC[0..31]
/OR1
/OR2
/RENABLE
RCLK
DATAOUTA[0..31]
XBUSC[0..31]
DATAOUTC[0..31]
Output Fifo Buffer
fifo out4.sch
MRESET11
/RESET
OUTFULL[0..7]
WCLK
/WENABLE
XBUSD[0..31]
/OR1
/OR2
/RENABLE
RCLK
DATAOUTA[0..31]
XBUSD[0..31]
DATAOUTD[0..31]
Data Multiplexer/Output Connector
multiplexer.sch
/OR1
/OR2
/RENABLE1
RCLK1
DATAOUTA[0..31]
/OR3
/OR4
/RENABLE2
RCLK2
DATAOUTB[0..31]
INPUTFULL
OUTPUTFULL
/OR5
/OR6
/RENABLE3
RCLK3
DATAOUTC[0..31]
/OR7
/OR8
/RENABLE4
RCLK4
DATAOUTD[0..31]
Title
Average Card
Size
B
Number
Revision
Date:
20-Sep-2001
Sheet of
File:
C:\Program Files\Design Explorer 99 SE\Projects\Averaging Card\Averaging Card.ddb
Drawn By:

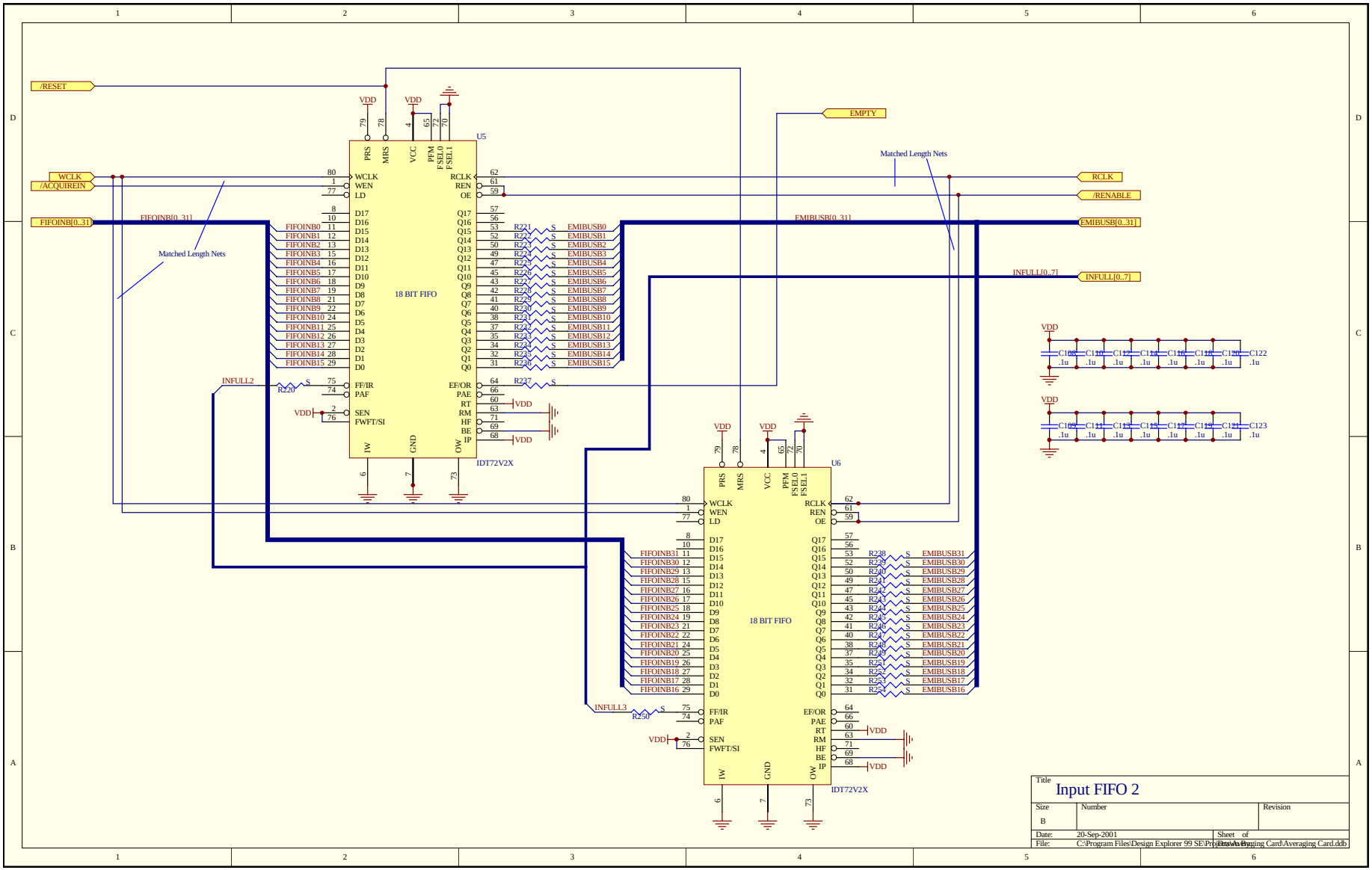


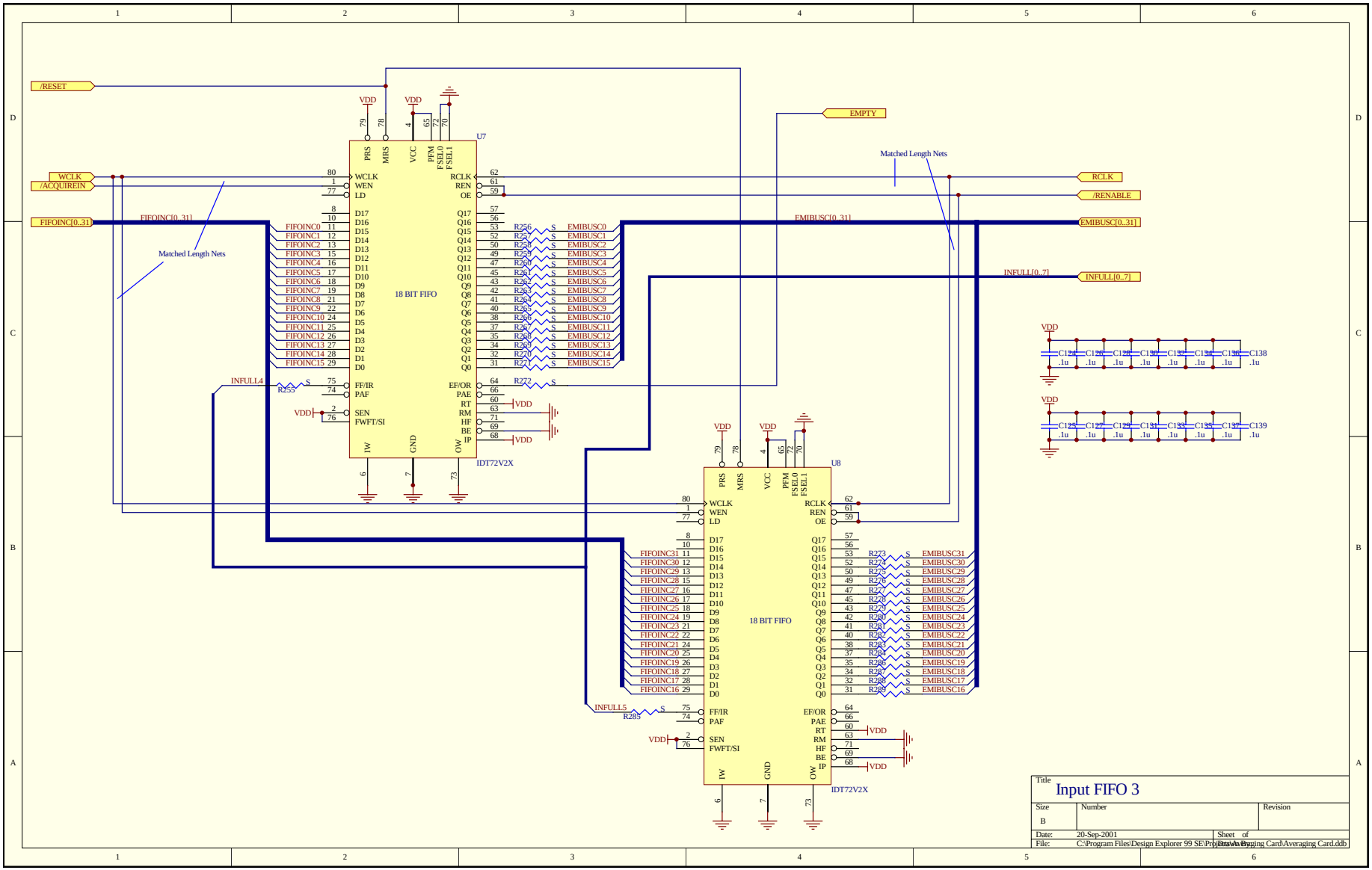
Title: Acquire Circuit		
Size: B	Number:	Revision:
Date: 20-Sep-2001	Sheet of	
File: C:\Program Files\Design Explorer 99 SE\Projects\Averaging Card\Averaging Card.ddb	Drawn By:	

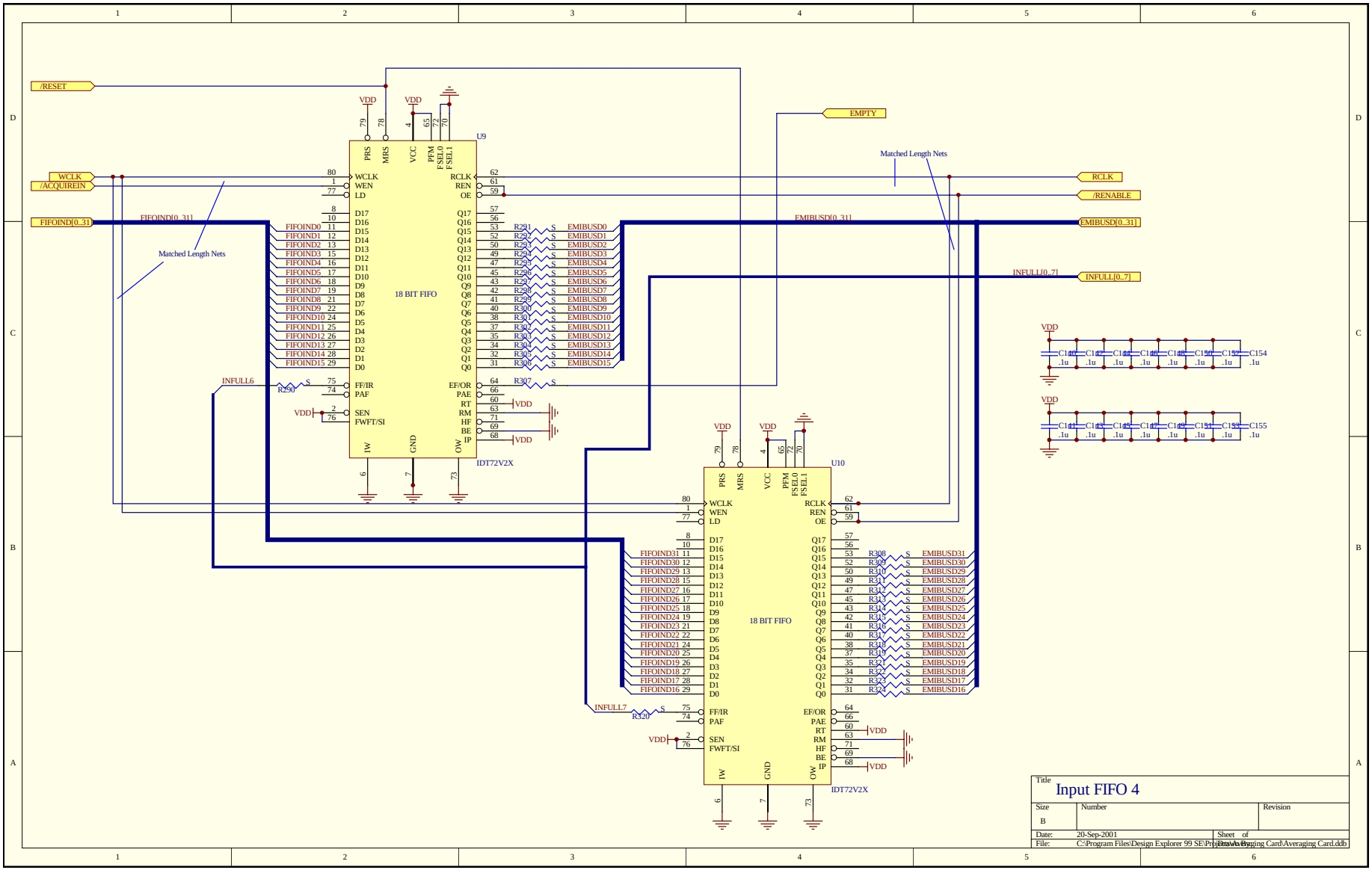

Master/Slave Jumper
J9
CON3
From Microcontroller
J10
CON26
JTAG Programming Jack
JP21
HEADER 5X2
JTAG Programming Jack
JP22
HEADER 5X2
U21
EPM3064ALC44-4
INTCOUNTER
U22A
EPM3032ALC44-4
RESETCHIP
/ACQUIRE
RESETINT[0..3]
RAWDATA
MRESET[0..11]
J11
CON8
INPUT
J12
CON8
OUTPUT
Multi-Board Chain Connection
Title
Integration Counter
Size
B
Number
Revision
Date: 20-Sep-2001
Sheet of
File: C:\Program Files\Design Explorer 99 SE\Projects\Imaging Card\Averaging Card.ddb
Drawn By:

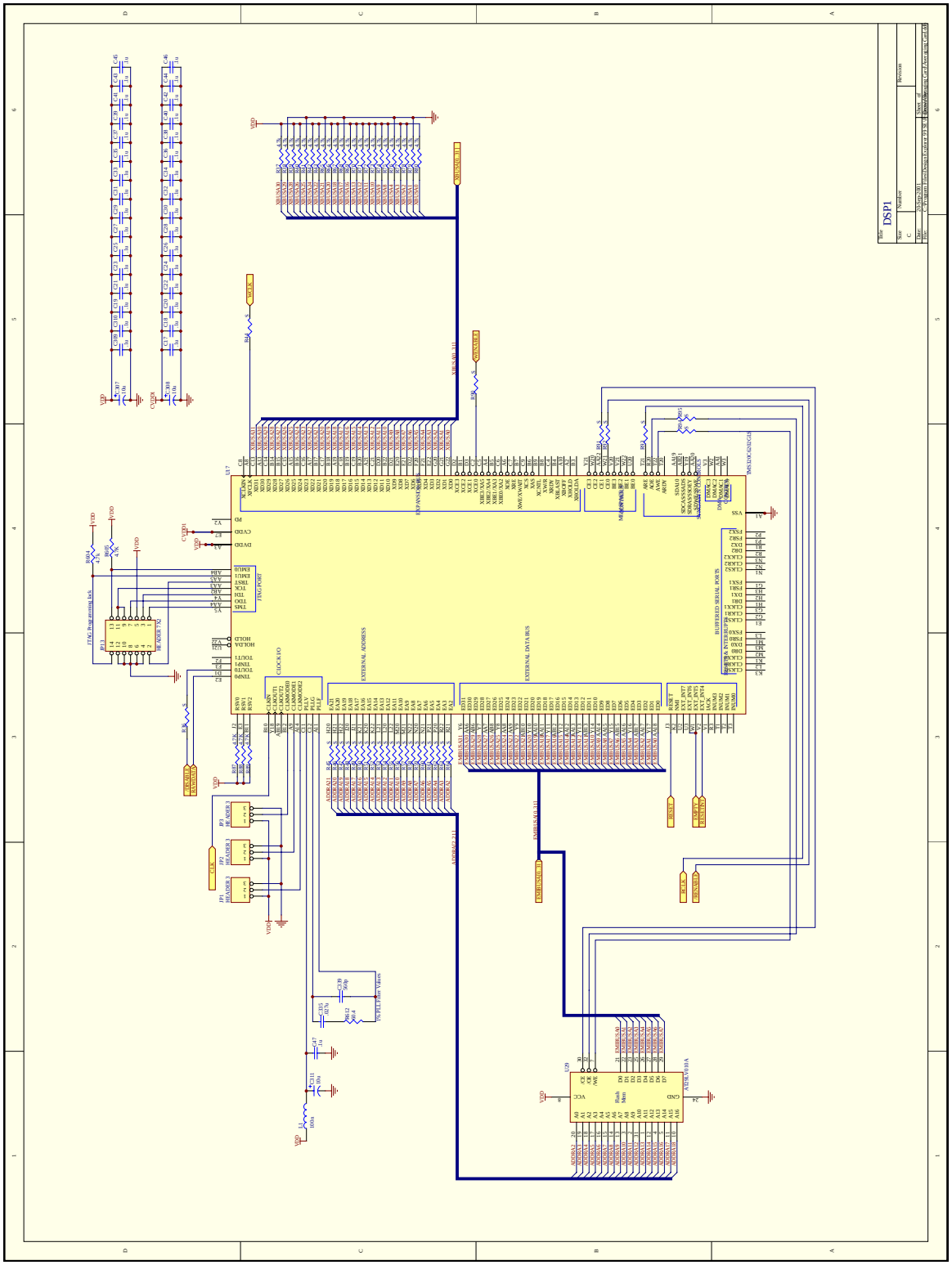

J1
WCLKA
FIFOINA[0..31]
FIFOINA0
FIFOINA1
FIFOINA2
FIFOINA3
FIFOINA4
FIFOINA5
FIFOINA6
FIFOINA7
FIFOINA8
FIFOINA9
FIFOINA10
FIFOINA11
FIFOINA12
FIFOINA13
FIFOINA14
CON34
J5
FIFOINA15
FIFOINA16
FIFOINA17
FIFOINA18
FIFOINA19
FIFOINA20
FIFOINA21
FIFOINA22
FIFOINA23
FIFOINA24
FIFOINA25
FIFOINA26
FIFOINA27
FIFOINA28
FIFOINA29
FIFOINA30
FIFOINA31
CON34
2mm 2X17 Header
J2
WCLKB
FIFOINB[0..31]
FIFOINB0
FIFOINB1
FIFOINB2
FIFOINB3
FIFOINB4
FIFOINB5
FIFOINB6
FIFOINB7
FIFOINB8
FIFOINB9
FIFOINB10
FIFOINB11
FIFOINB12
FIFOINB13
FIFOINB14
CON34
J6
FIFOINB15
FIFOINB16
FIFOINB17
FIFOINB18
FIFOINB19
FIFOINB20
FIFOINB21
FIFOINB22
FIFOINB23
FIFOINB24
FIFOINB25
FIFOINB26
FIFOINB27
FIFOINB28
FIFOINB29
FIFOINB30
FIFOINB31
CON34
2mm 2X17 Header
J3
WCLKC
FIFOINC[0..31]
FIFOINC0
FIFOINC1
FIFOINC2
FIFOINC3
FIFOINC4
FIFOINC5
FIFOINC6
FIFOINC7
FIFOINC8
FIFOINC9
FIFOINC10
FIFOINC11
FIFOINC12
FIFOINC13
FIFOINC14
CON34
J7
FIFOINC15
FIFOINC16
FIFOINC17
FIFOINC18
FIFOINC19
FIFOINC20
FIFOINC21
FIFOINC22
FIFOINC23
FIFOINC24
FIFOINC25
FIFOINC26
FIFOINC27
FIFOINC28
FIFOINC29
FIFOINC30
FIFOINC31
CON34
2mm 2X17 Header
J4
WCLKD
FIFOIND[0..31]
FIFOIND0
FIFOIND1
FIFOIND2
FIFOIND3
FIFOIND4
FIFOIND5
FIFOIND6
FIFOIND7
FIFOIND8
FIFOIND9
FIFOIND10
FIFOIND11
FIFOIND12
FIFOIND13
FIFOIND14
CON34
J8
FIFOIND15
FIFOIND16
FIFOIND17
FIFOIND18
FIFOIND19
FIFOIND20
FIFOIND21
FIFOIND22
FIFOIND23
FIFOIND24
FIFOIND25
FIFOIND26
FIFOIND27
FIFOIND28
FIFOIND29
FIFOIND30
FIFOIND31
CON34
2mm 2X17 Header
Title
Input Connectors
Size
B
Number
Revision
Date: 20-Sep-2001
Sheet of
File: C:\Program Files\Design Explorer 99 SE\Projects\Imaging Card\Averaging Card.ddb
Drawn By:

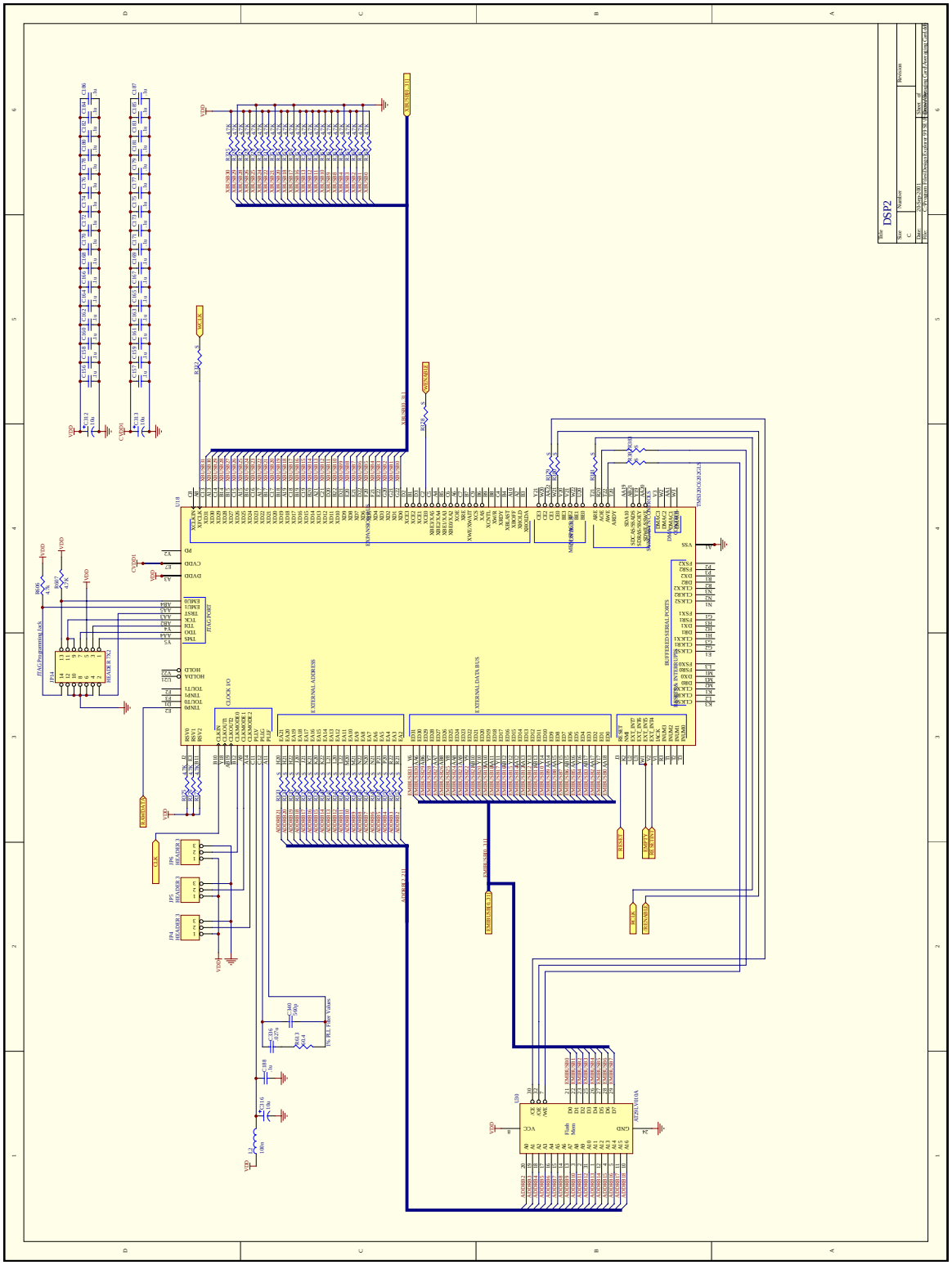

/RESET
WCLK
/ACQUIREIN
FIFOINA[0..31]
Matched Length Nets
INFULL0
EMPTY
Matched Length Nets
RCLK
/RENABLE
EMIBUSA[0..31]
EMIBUSA[0..31]
INFULL[0..7]
INFULL[0..7]
U1
18 BIT FIFO
IDT72V2X
U2
18 BIT FIFO
IDT72V2X
INFULL1
VDD
C1 C3 C5 C7 C9 C11 C13 C15 .1u
C2 C4 C6 C8 C10 C12 C14 C16 .1u
Title
Input FIFO 1
Size B
Number
Revision
Date: 20-Sep-2001
Sheet of
File: C:\Program Files\Design Explorer 99 SE\Projects\Imaging Card\Averaging Card.ddb
Drawn By:

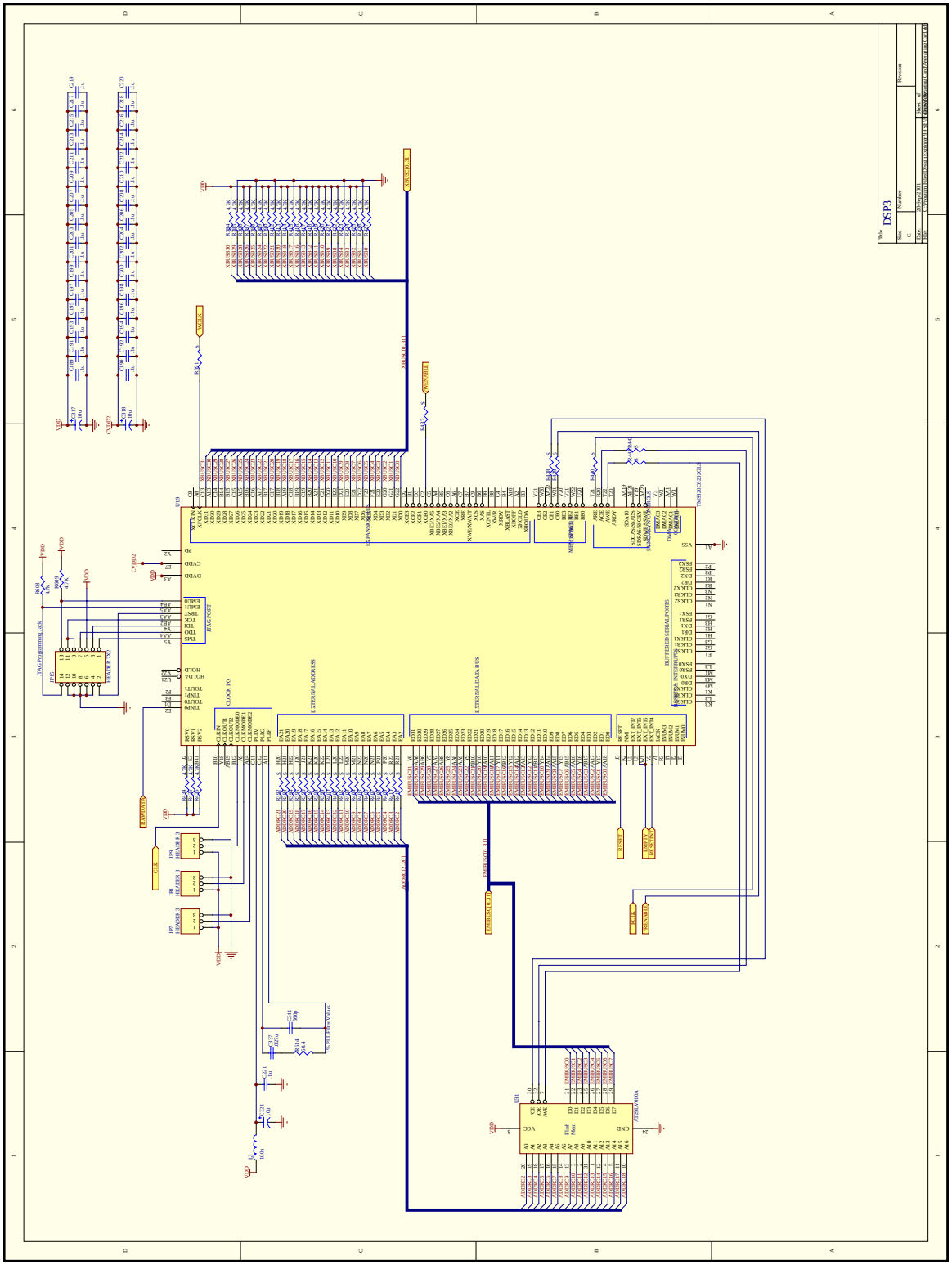

/RESET
WCLK
/ACQUIREIN
FIFOINB[0..31]
FIFOINB[0..31]
Matched Length Nets
Matched Length Nets
EMPTY
INFULL2
INFULL3
U5
U6
18 BIT FIFO
IDT72V2X
VDD
R220
R250
R221
R222
R223
R224
R225
R226
R227
R228
R229
R230
R231
R232
R233
R234
R235
R236
R237
R238
R239
R240
R241
R242
R243
R244
R245
R246
R247
R248
R249
R250
R251
R252
R253
R254
EMIBUSB0
EMIBUSB1
EMIBUSB2
EMIBUSB3
EMIBUSB4
EMIBUSB5
EMIBUSB6
EMIBUSB7
EMIBUSB8
EMIBUSB9
EMIBUSB10
EMIBUSB11
EMIBUSB12
EMIBUSB13
EMIBUSB14
EMIBUSB15
EMIBUSB16
EMIBUSB17
EMIBUSB18
EMIBUSB19
EMIBUSB20
EMIBUSB21
EMIBUSB22
EMIBUSB23
EMIBUSB24
EMIBUSB25
EMIBUSB26
EMIBUSB27
EMIBUSB28
EMIBUSB29
EMIBUSB30
EMIBUSB31
EMIBUSB[0..31]
INFULL[0..7]
RCLK
/RENABLE
C109
C110
C112
C114
C116
C118
C120
C122
C109
C111
C113
C115
C117
C119
C121
C123
.1u
Title
Input FIFO 2
Size
B
Number
Revision
Date:
20-Sep-2001
Sheet of
File:
C:\Program Files\Design Explorer 99 SE\Projects\Imaging Card\Averaging Card.ddb
Drawn By:

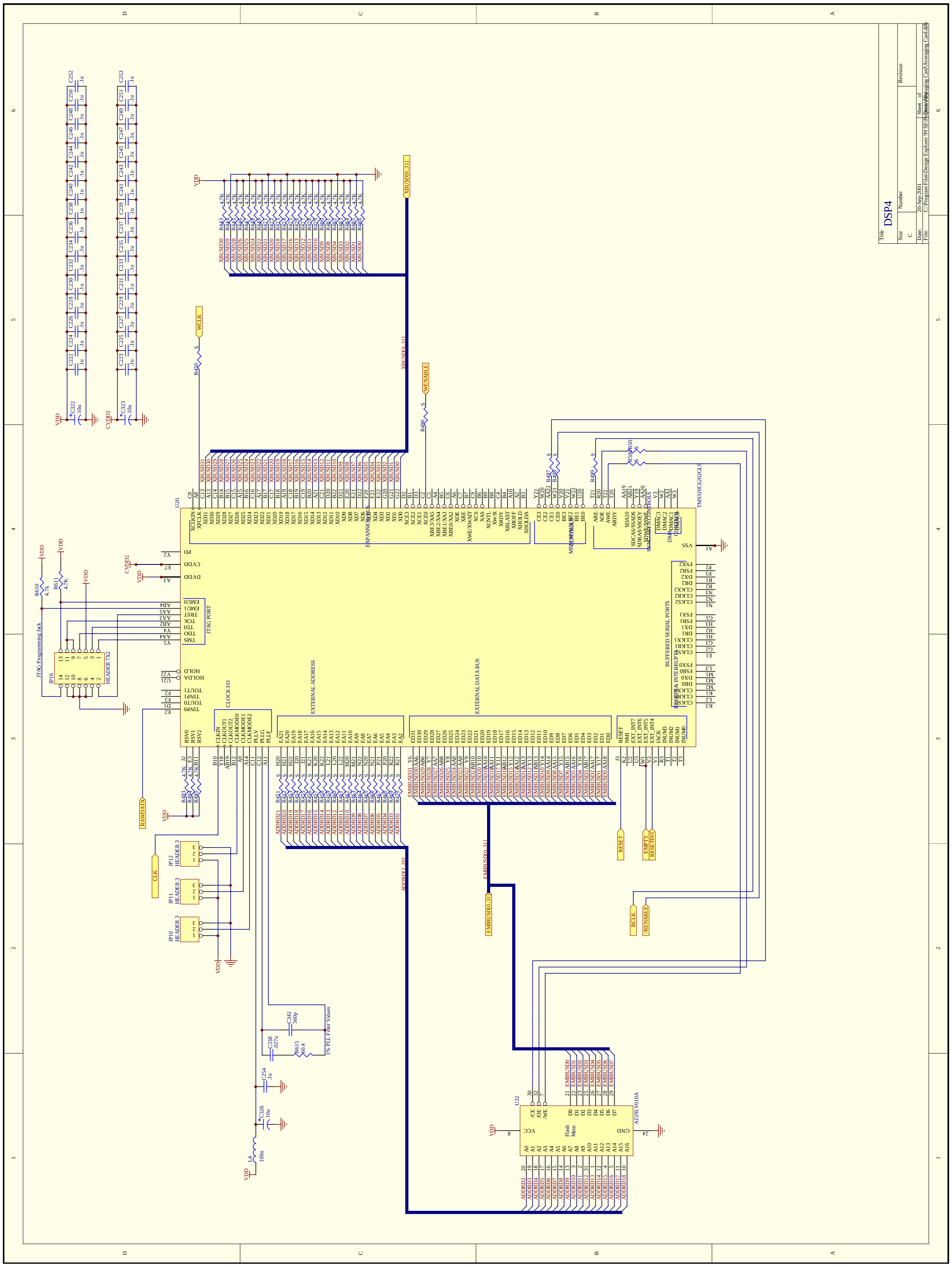

/RESET
WCLK
/ACQUIREIN
FIFOINC[0..31]
Matched Length Nets
EMPTY
Matched Length Nets
RCLK
/RENABLE
EMIBUSC[0..31]
INFULL[0..7]
INFULL[0..7]
U7
18 BIT FIFO
IDT72V2X
U8
18 BIT FIFO
IDT72V2X
INFULL4
INFULL5
VDD
Title
Input FIFO 3
Size
B
Number
Revision
Date: 20-Sep-2001
Sheet of
File: C:\Program Files\Design Explorer 99 SE\Projects\Imaging Card\Averaging Card.ddb
Drawn By:

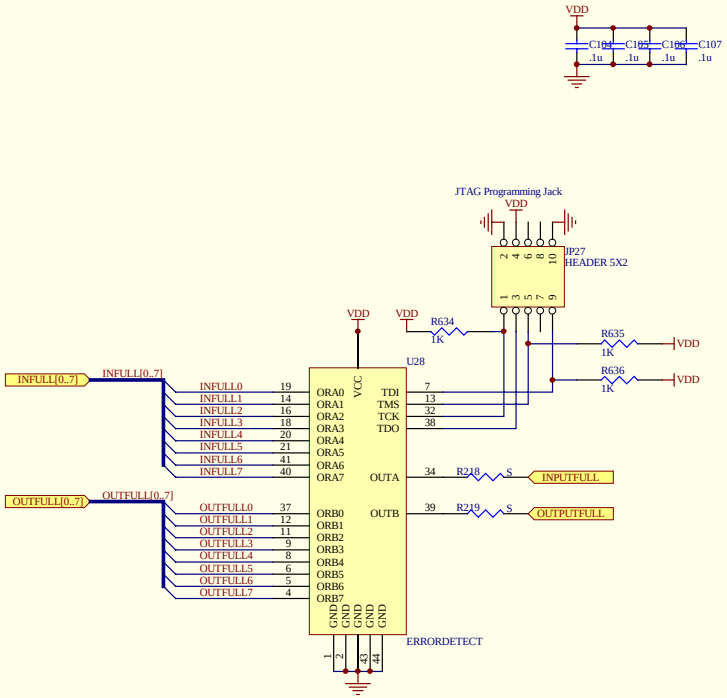

/RESET
WCLK
/ACQUIREIN
FIFOIND[0..31]
Matched Length Nets
U9
18 BIT FIFO
IDT72V2X
U10
EMPTY
Matched Length Nets
RCLK
/RENABLE
EMIBUSD[0..31]
INFULL[0..7]
INFULL6
INFULL7
Title
Input FIFO 4
Size
B
Number
Revision
Date: 20-Sep-2001
Sheet of
File: C:\Program Files\Design Explorer 99 SE\Projects\Imaging Card\Averaging Card.ddb
Drawn By:



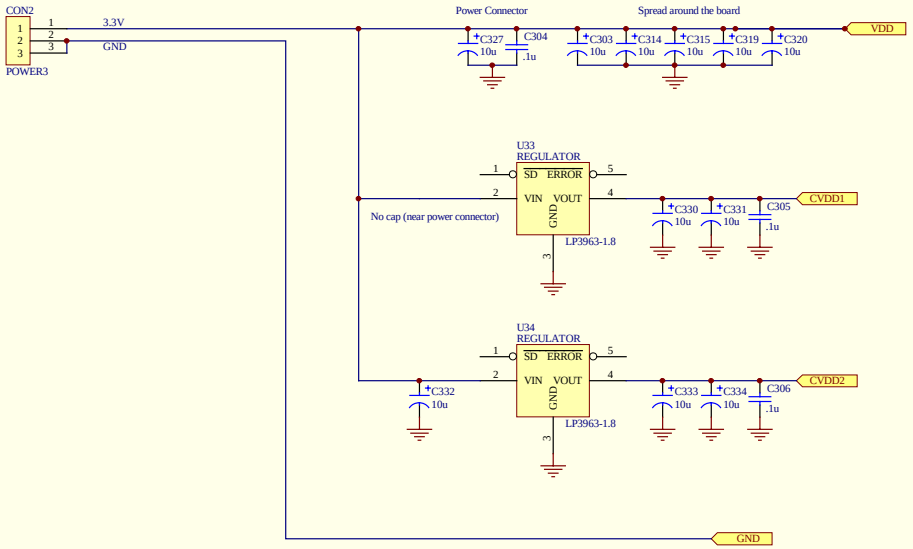




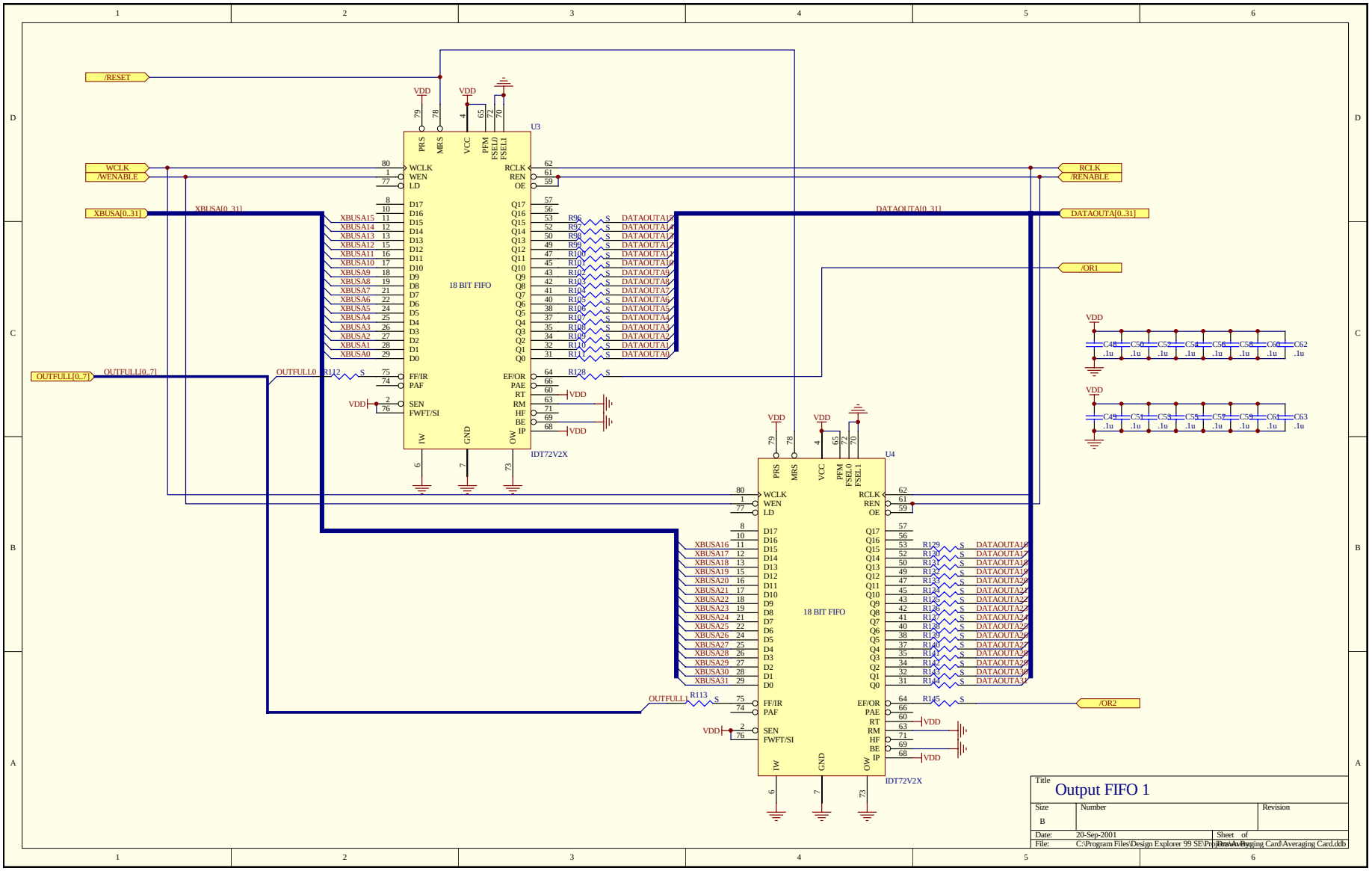


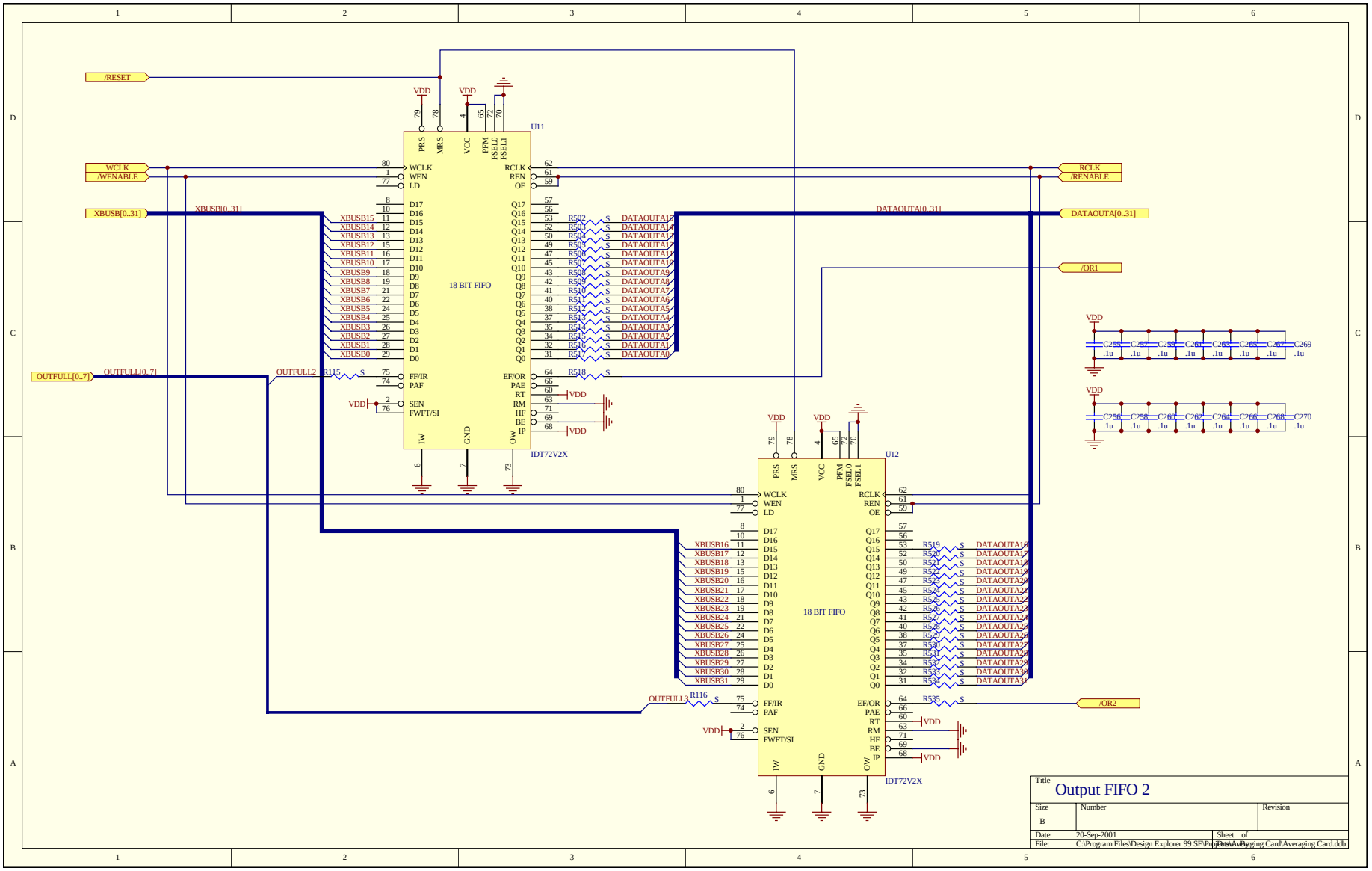


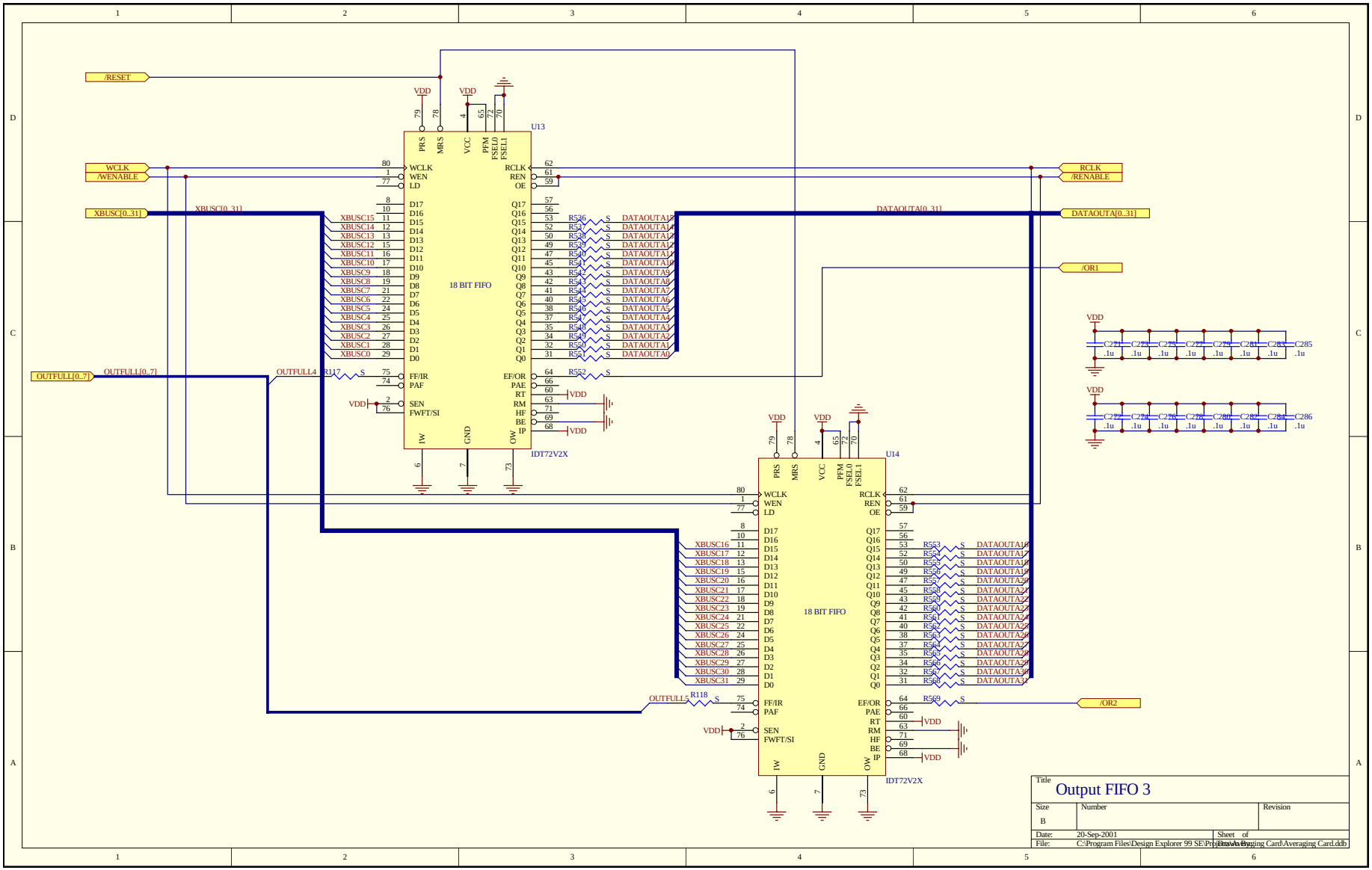
Title: Error Detector		
Size: B	Number:	Revision:
Date: 20-Sep-2001	Sheet of	
File: C:\Program Files\Design Explorer 99 SE\Projects\Imaging Card\Averaging Card.ddb	Drawn By:	

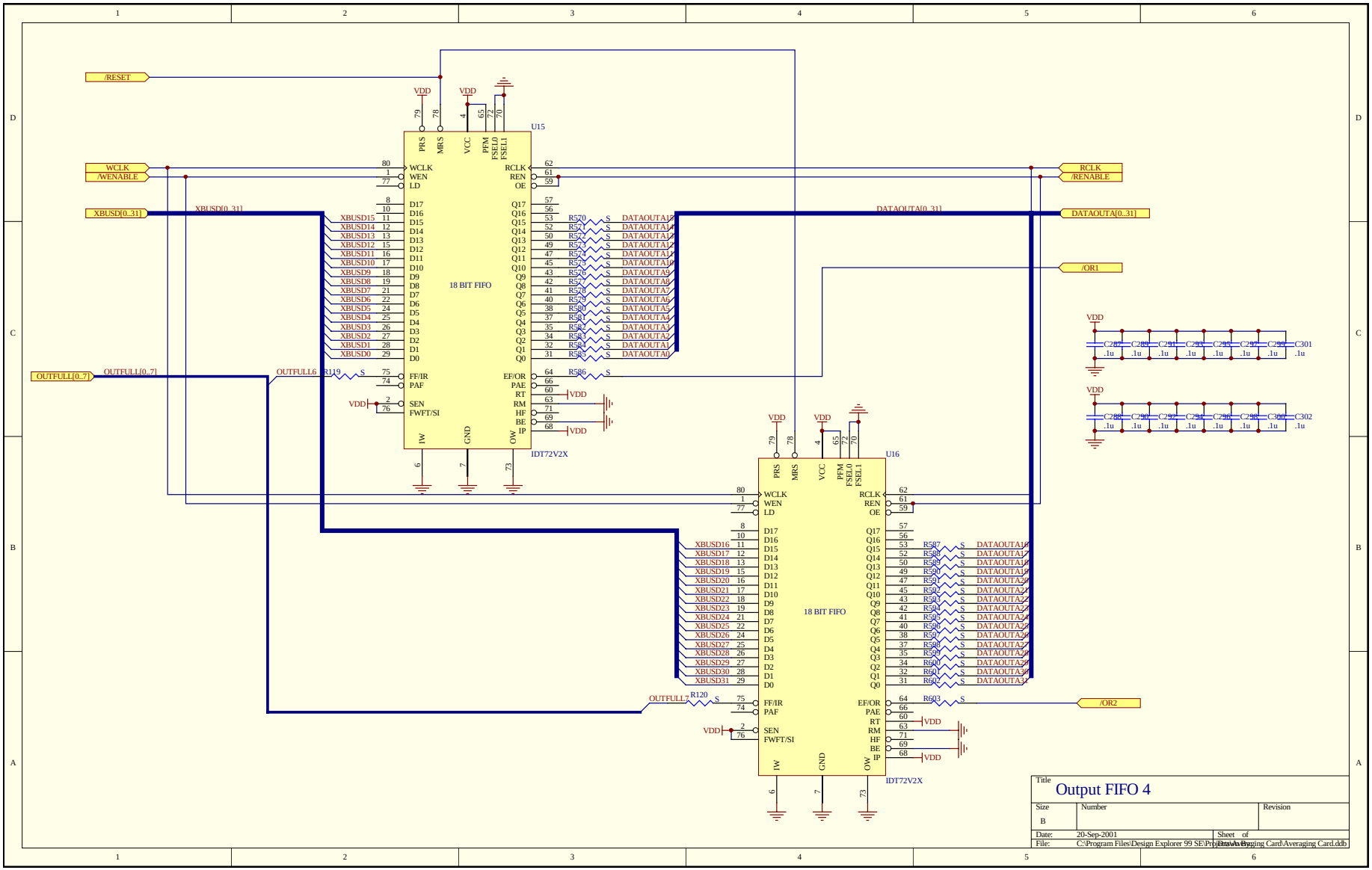


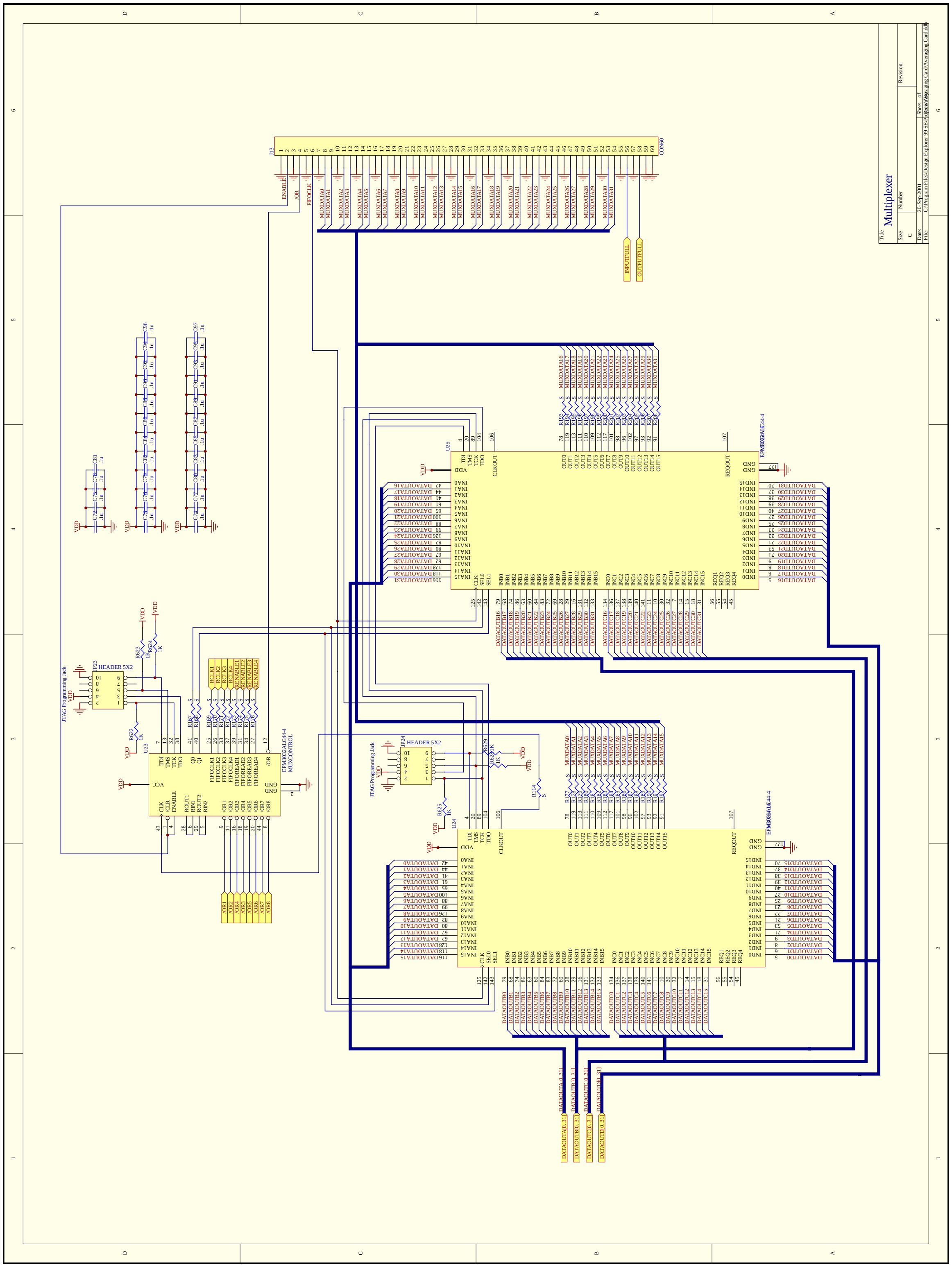
Title: Power		
Size: B	Number:	Revision:
Date: 20-Sep-2001	Sheet of	
File: C:\Program Files\Design Explorer 99 SE\Projects\Averaging Card\Averaging Card.ddb	Drawn By:	


/RESET
WCLK
/WENABLE
XBUSA[0..31]
OUTFULL[0..7]
RCLK
/RENABLE
DATAOUTA[0..31]
/OR1
/OR2
U3
U4
18 BIT FIFO
IDT72V2X
VDD
C48 C50 C52 C54 C56 C58 C60 C62 .1u
C49 C51 C53 C55 C57 C59 C61 C63 .1u
Title
Output FIFO 1
Size
B
Number
Revision
Date: 20-Sep-2001
Sheet of
File: C:\Program Files\Design Explorer 99 SE\Projects\Imaging Card\Averaging Card.ddb
Drawn By:

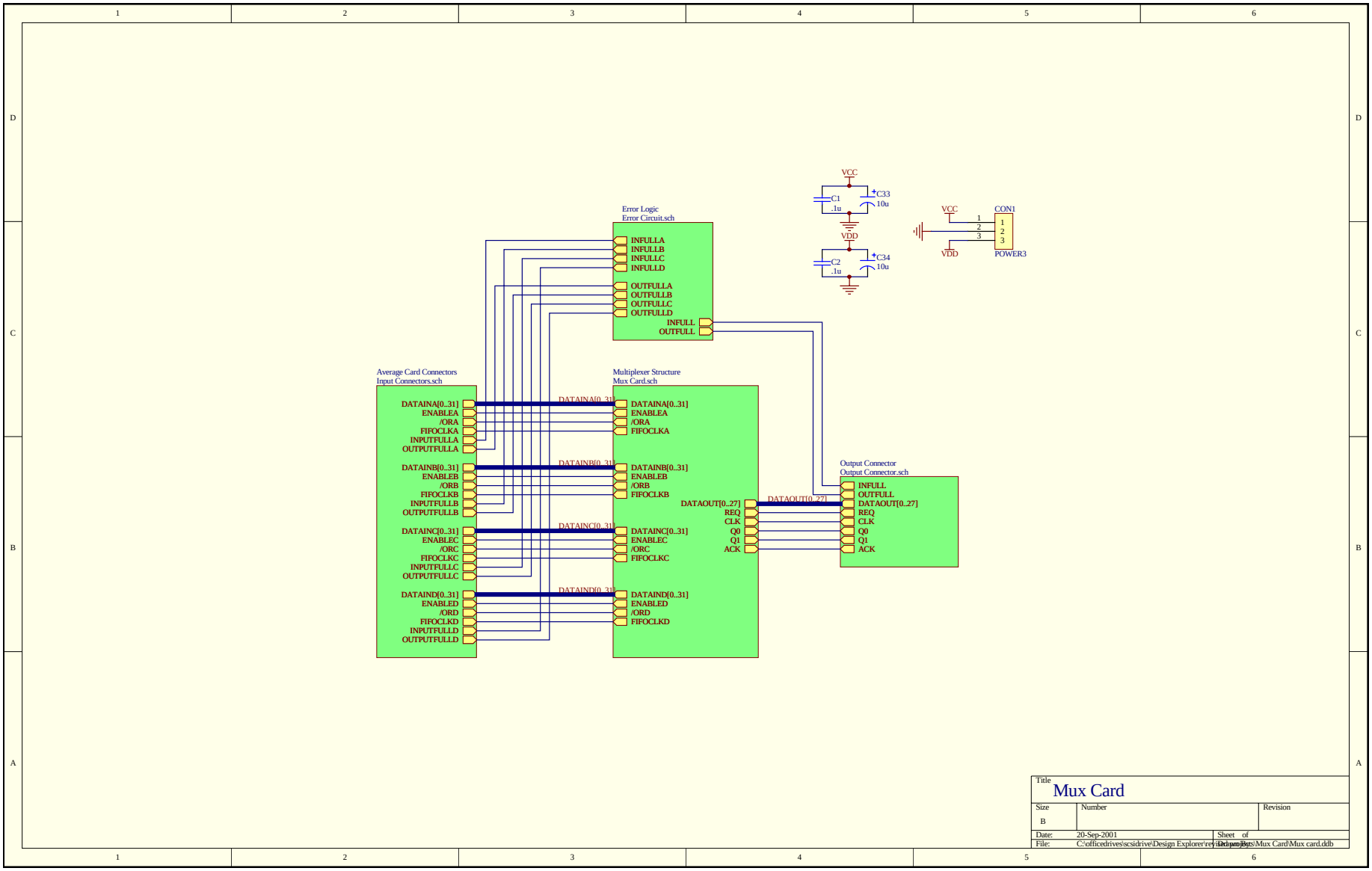

/RESET
WCLK
/WENABLE
XBUSB[0..31]
OUTFULL[0..7]
RCLK
/RENABLE
DATAOUTA[0..31]
/OR1
/OR2
U11
U12
18 BIT FIFO
IDT72V2X
VDD
GND
Title
Output FIFO 2
Size
B
Number
Revision
Date: 20-Sep-2001
Sheet of
File: C:\Program Files\Design Explorer 99 SE\Projects\Imaging Card\Averaging Card.ddb
Drawn By:


/RESET
WCLK
/WENABLE
XBUSC[0..31]
OUTFULL[0..7]
RCLK
/RENABLE
DATAOUTA[0..31]
/OR1
/OR2
U13
U14
18 BIT FIFO
IDT72V2X
WCLK
WEN
LD
SEN
FWFT/SI
FF/IR
PAF
EF/OR
PAE
RT
RM
HF
BE
IP
PRS
MRS
VCC
PFM
FSEL0
FSEL1
IW
GND
OW
VDD
Title
Output FIFO 3
Size
B
Number
Revision
Date: 20-Sep-2001
Sheet of
File: C:\Program Files\Design Explorer 99 SE\Projects\Imaging Card\Averaging Card.ddb
Drawn By:

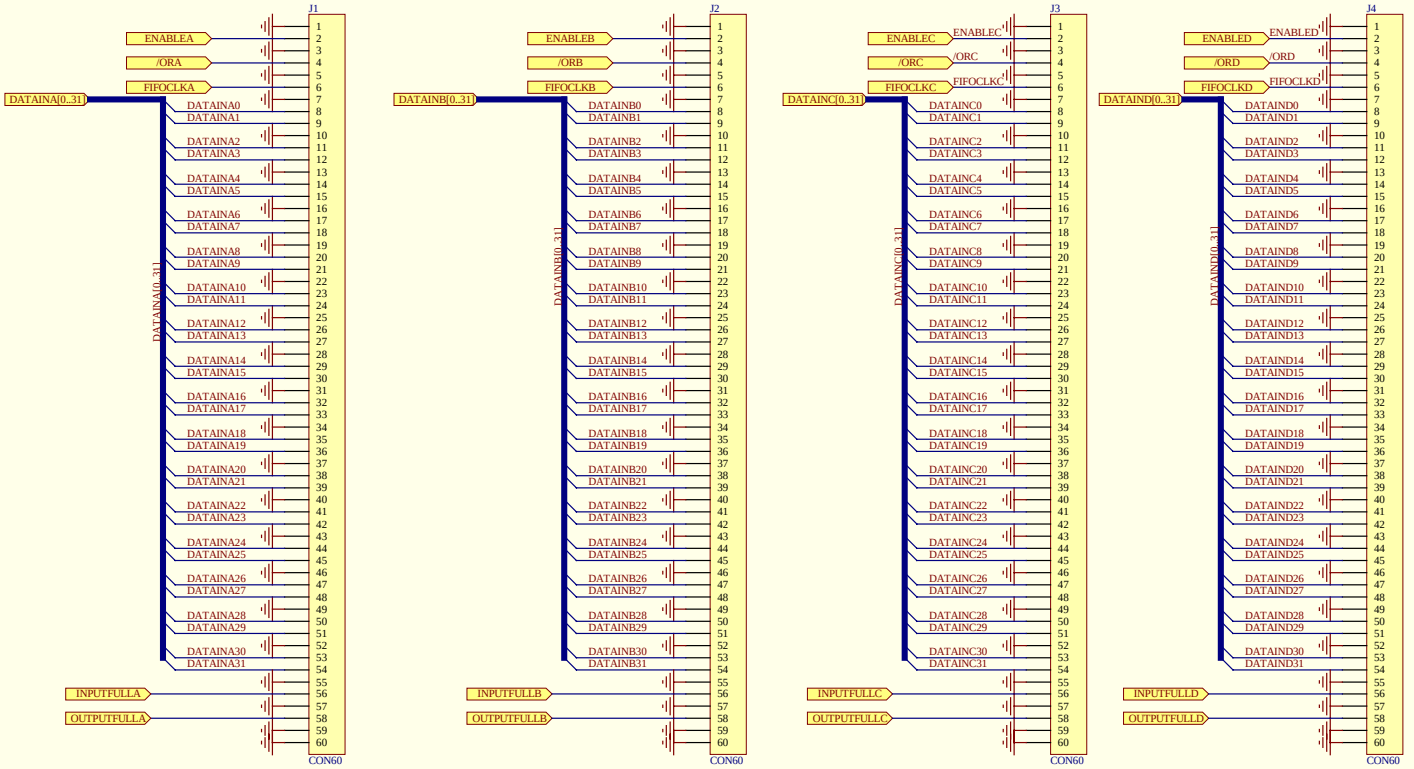

/RESET
WCLK
/WENABLE
XBUSD[0..31]
OUTFULL[0..7]
RCLK
/RENABLE
DATAOUTA[0..31]
/OR1
/OR2
U15
U16
18 BIT FIFO
IDT72V2X
VDD
GND
Title
Output FIFO 4
Size
B
Number
Revision
Date: 20-Sep-2001
Sheet of
File: C:\Program Files\Design Explorer 99 SE\Projects\Imaging Card\Averaging Card.ddb
Drawn By:

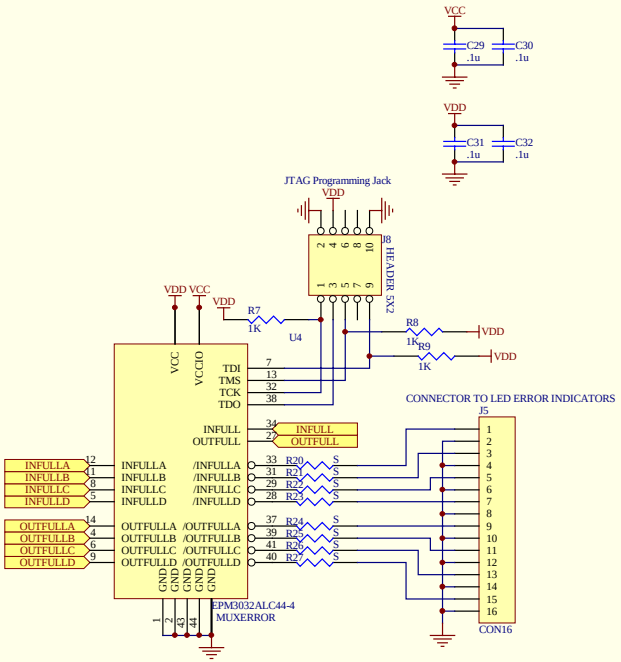

Multiplexer

APPENDIX C MULTIPLEXER CARD SCHEMATICS

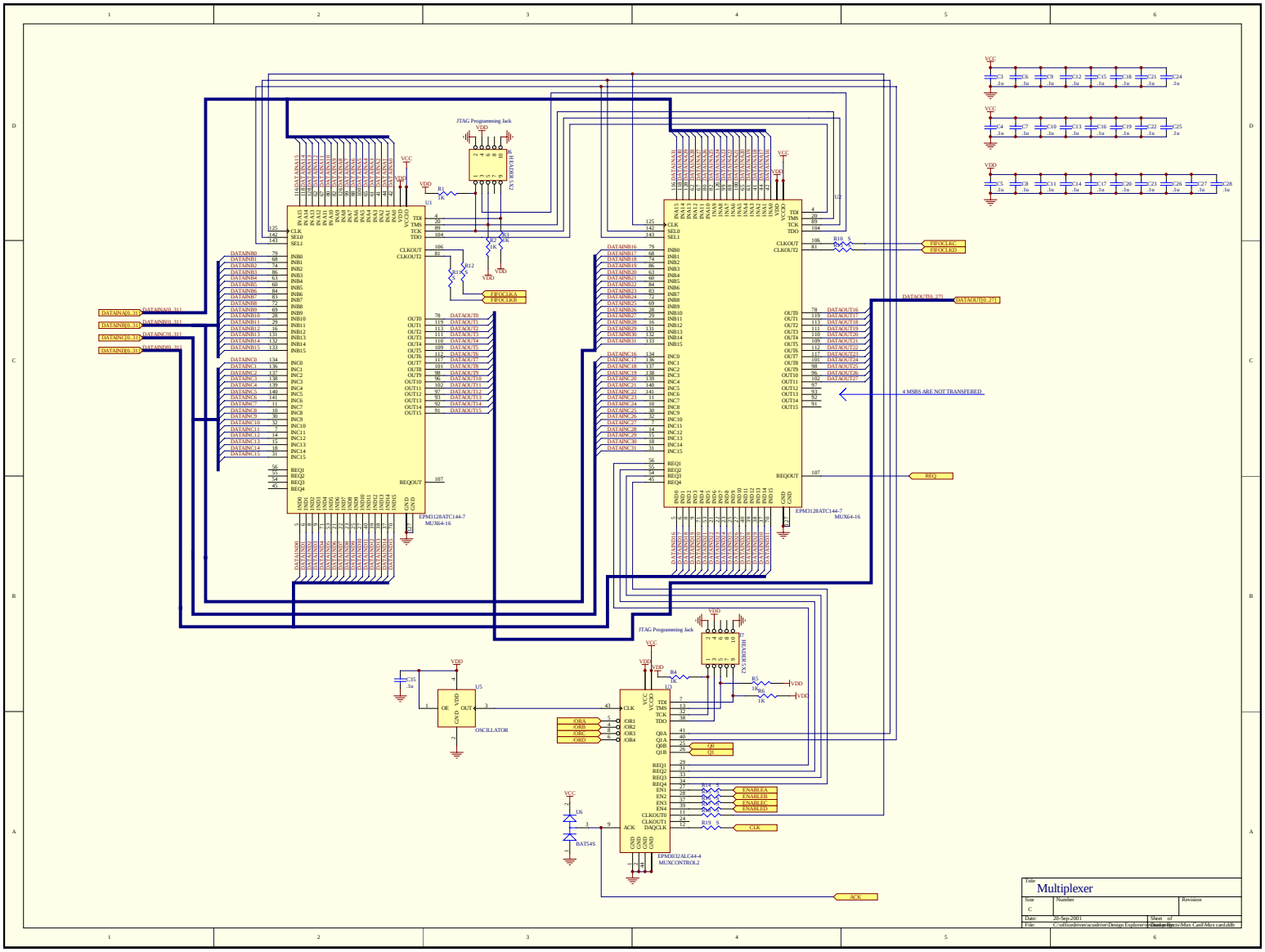


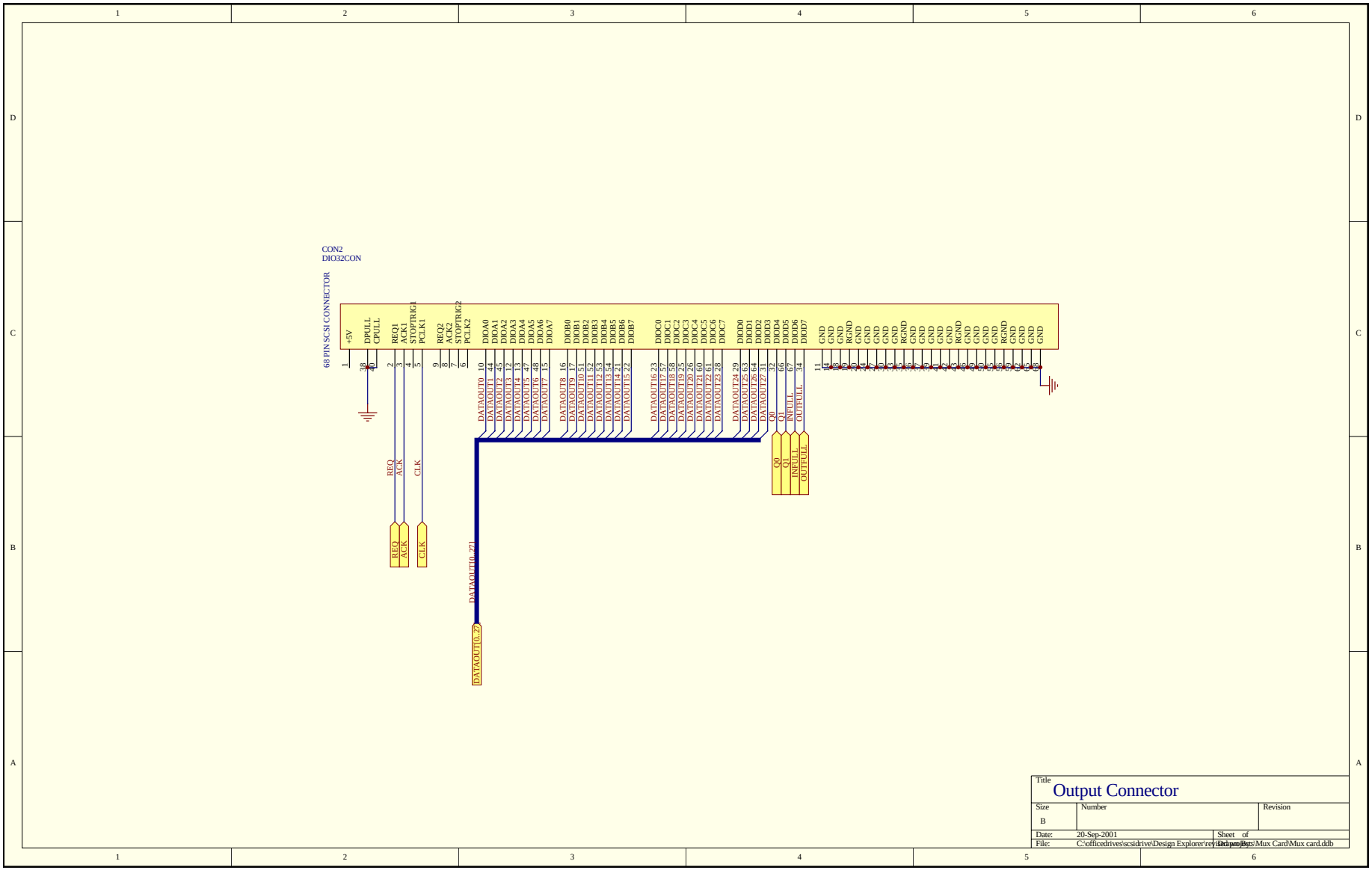
Title: Mux Card		
Size: B	Number:	Revision:
Date: 20-Sep-2001	Sheet of	
File: C:\officedrives\scsidrive\Design Explorer\revisions\Documents\Mux Card\Mux card.ddb		


J1
ENABLEA
/ORA
FIFOCLKA
DATAINA[0..31]
DATAINA0
DATAINA1
DATAINA2
DATAINA3
DATAINA4
DATAINA5
DATAINA6
DATAINA7
DATAINA8
DATAINA9
DATAINA10
DATAINA11
DATAINA12
DATAINA13
DATAINA14
DATAINA15
DATAINA16
DATAINA17
DATAINA18
DATAINA19
DATAINA20
DATAINA21
DATAINA22
DATAINA23
DATAINA24
DATAINA25
DATAINA26
DATAINA27
DATAINA28
DATAINA29
DATAINA30
DATAINA31
INPUTFULLA
OUTPUTFULLA
CON60
J2
ENABLEB
/ORB
FIFOCLKB
DATAINB[0..31]
DATAINB0
DATAINB1
DATAINB2
DATAINB3
DATAINB4
DATAINB5
DATAINB6
DATAINB7
DATAINB8
DATAINB9
DATAINB10
DATAINB11
DATAINB12
DATAINB13
DATAINB14
DATAINB15
DATAINB16
DATAINB17
DATAINB18
DATAINB19
DATAINB20
DATAINB21
DATAINB22
DATAINB23
DATAINB24
DATAINB25
DATAINB26
DATAINB27
DATAINB28
DATAINB29
DATAINB30
DATAINB31
INPUTFULLB
OUTPUTFULLB
CON60
J3
ENABLEC
/ORC
FIFOCLKC
DATAINC[0..31]
DATAINC0
DATAINC1
DATAINC2
DATAINC3
DATAINC4
DATAINC5
DATAINC6
DATAINC7
DATAINC8
DATAINC9
DATAINC10
DATAINC11
DATAINC12
DATAINC13
DATAINC14
DATAINC15
DATAINC16
DATAINC17
DATAINC18
DATAINC19
DATAINC20
DATAINC21
DATAINC22
DATAINC23
DATAINC24
DATAINC25
DATAINC26
DATAINC27
DATAINC28
DATAINC29
DATAINC30
DATAINC31
INPUTFULLC
OUTPUTFULLC
CON60
J4
ENABLED
/ORD
FIFOCLKD
DATAIND[0..31]
DATAIND0
DATAIND1
DATAIND2
DATAIND3
DATAIND4
DATAIND5
DATAIND6
DATAIND7
DATAIND8
DATAIND9
DATAIND10
DATAIND11
DATAIND12
DATAIND13
DATAIND14
DATAIND15
DATAIND16
DATAIND17
DATAIND18
DATAIND19
DATAIND20
DATAIND21
DATAIND22
DATAIND23
DATAIND24
DATAIND25
DATAIND26
DATAIND27
DATAIND28
DATAIND29
DATAIND30
DATAIND31
INPUTFULLD
OUTPUTFULLD
CON60
Title
Input Connectors
Size
B
Number
Revision
Date: 20-Sep-2001
Sheet of
File: C:\officedrives\scsidrive\Design Explorer\revisions\Schematics\Mux Card\Mux card.ddb

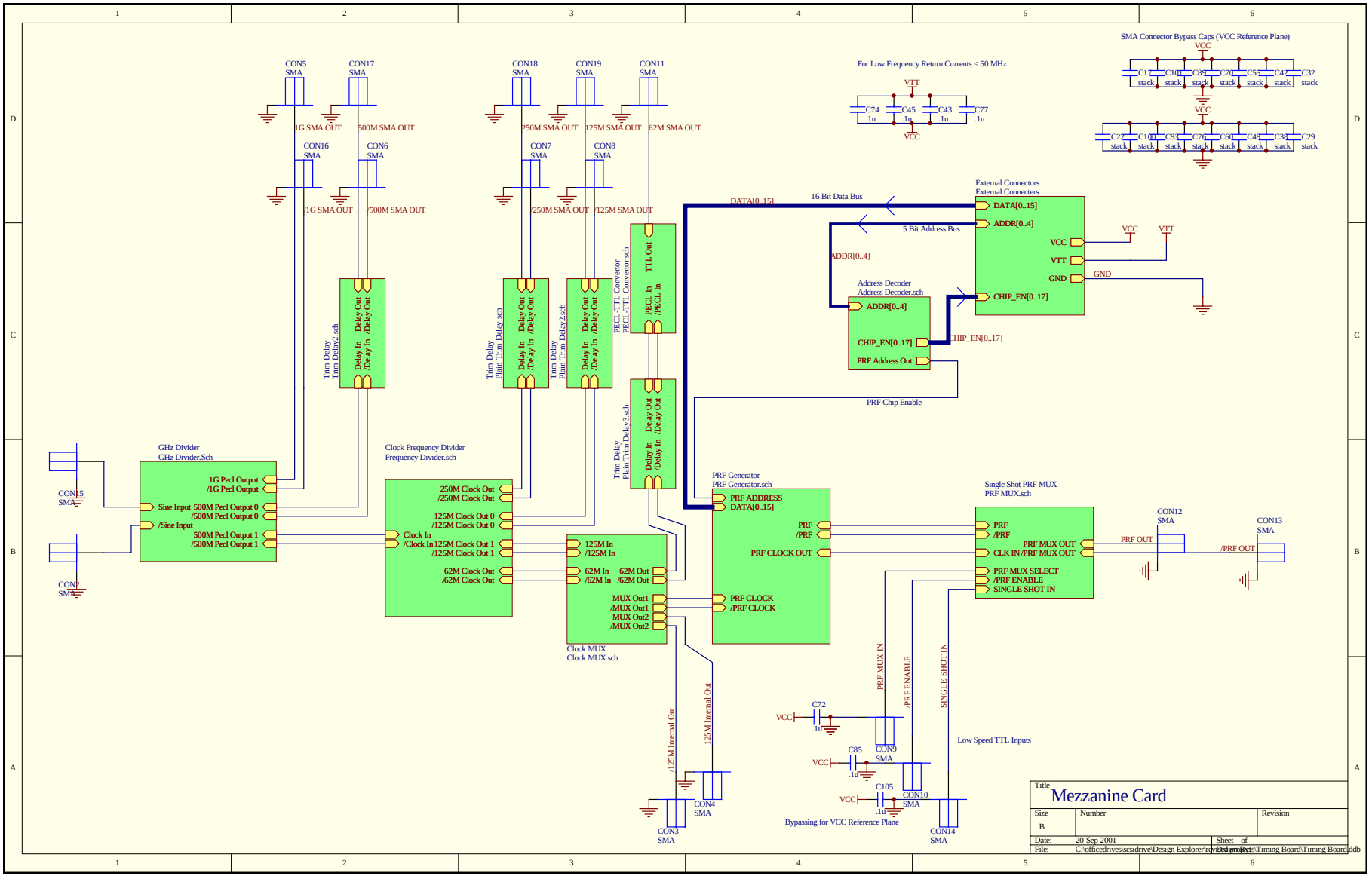


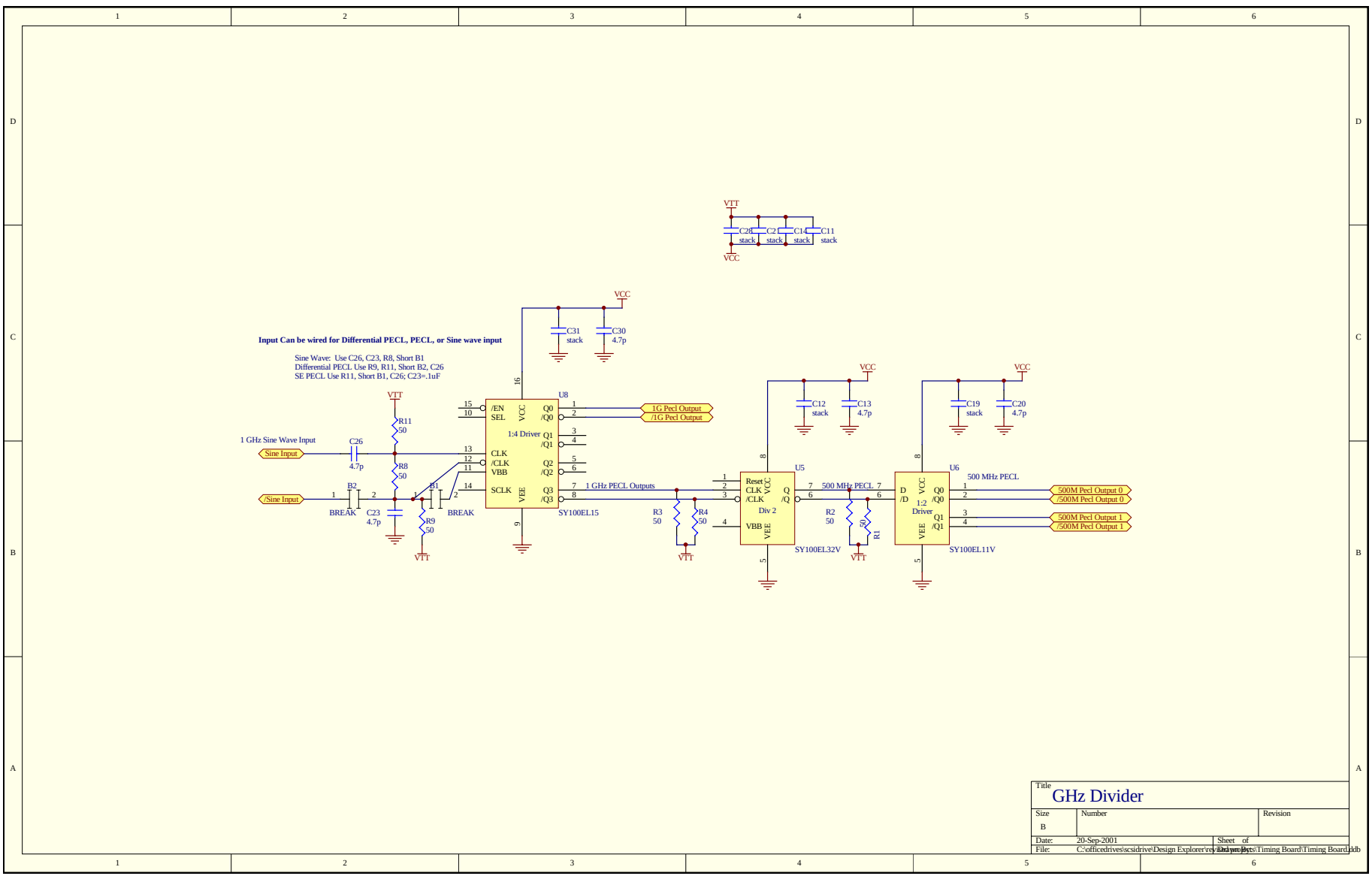
Title: Error Circuit		
Size: B	Number:	Revision:
Date: 20-Sep-2001	Sheet of	
File: C:\officedrives\scsidrive\Design Explorer\revisions\Mux Card\Mux card.ddb		


Multiplexer
EPM3128ATC144-7
MUX64-16
EPM3032ALC44-4
MUXCONTROL2
OSCILLATOR
JTAG Programming Jack
4 MSBS ARE NOT TRANSFERED
FIFOCLKA
FIFOCLKB
FIFOCLKC
FIFOCLKD
REQ
ACK
CLK
ENABLEA
ENABLEB
ENABLEC
ENABLED
Date: 20-Sep-2001
Sheet 6 of 6
File: C:\altiledriver\acaidriver\Design Explorer\Schematics\Max Card\Max card.ddb

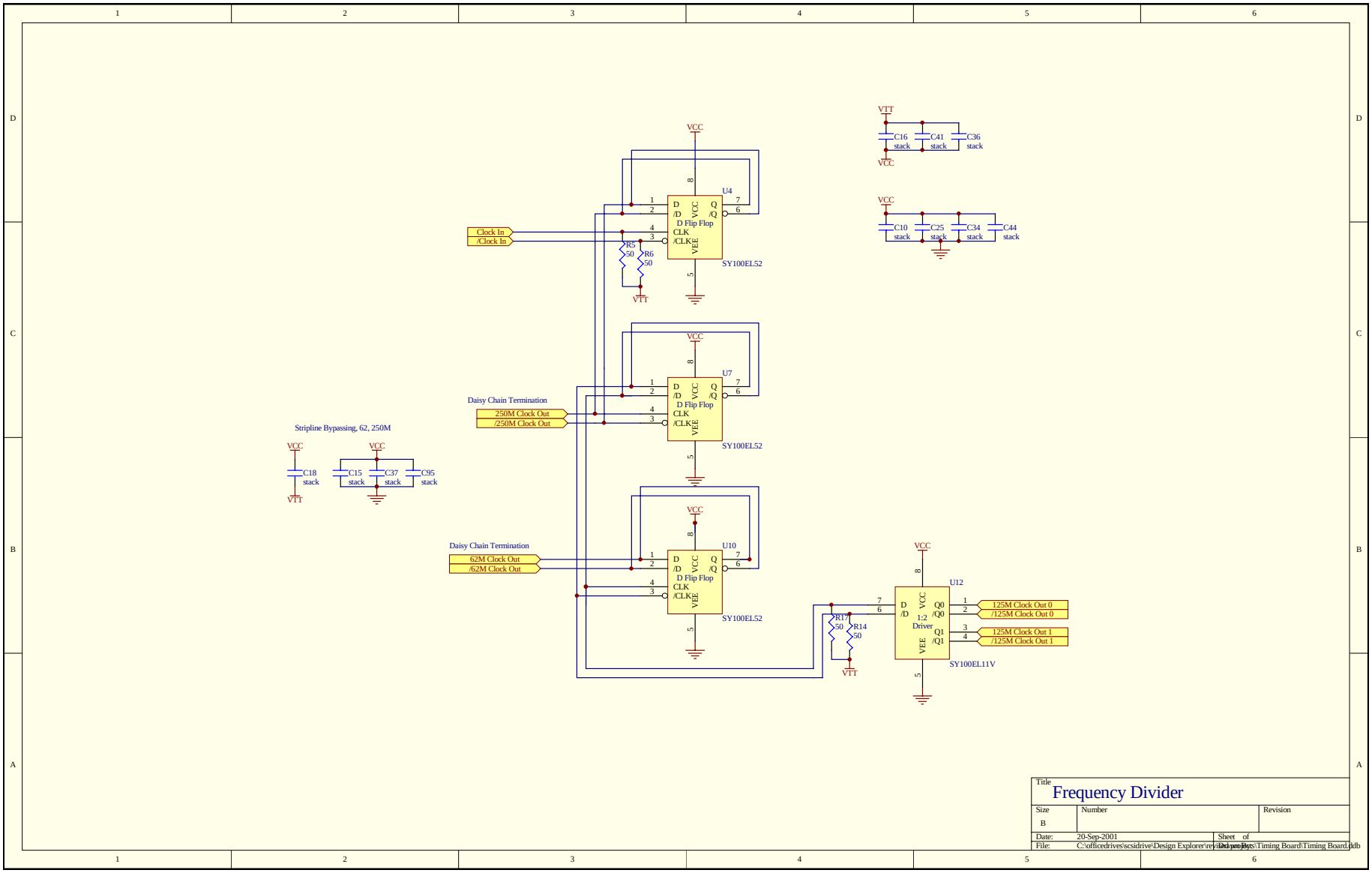

CON2
DIO32CON
68 PIN SCSI CONNECTOR
+5V
DPULL
CPULL
REQ1
ACK1
STOPTRIG1
PCLK1
REQ2
ACK2
STOPTRIG2
PCLK2
DIOA0
DIOA1
DIOA2
DIOA3
DIOA4
DIOA5
DIOA6
DIOA7
DIOB0
DIOB1
DIOB2
DIOB3
DIOB4
DIOB5
DIOB6
DIOB7
DIOC0
DIOC1
DIOC2
DIOC3
DIOC4
DIOC5
DIOC6
DIOC7
DIOD0
DIOD1
DIOD2
DIOD3
DIOD4
DIOD5
DIOD6
DIOD7
GND
RGND
DATAOUT0
DATAOUT1
DATAOUT2
DATAOUT3
DATAOUT4
DATAOUT5
DATAOUT6
DATAOUT7
DATAOUT8
DATAOUT9
DATAOUT10
DATAOUT11
DATAOUT12
DATAOUT13
DATAOUT14
DATAOUT15
DATAOUT16
DATAOUT17
DATAOUT18
DATAOUT19
DATAOUT20
DATAOUT21
DATAOUT22
DATAOUT23
DATAOUT24
DATAOUT25
DATAOUT26
DATAOUT27
Q0
Q1
INFULL
OUTFULL
REQ
ACK
CLK
DATAOUT[0..27]
Title
Output Connector
Size
B
Number
Revision
Date: 20-Sep-2001
Sheet of
File: C:\officedrives\scsidrive\Design Explorer\revisions\Mux Card\Mux card.ddb

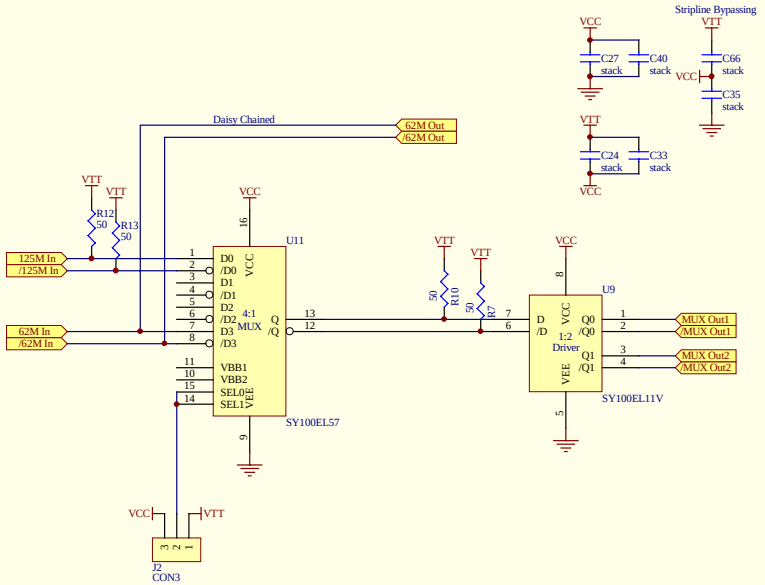
APPENDIX D TIMING MEZZANINE CARD SCHEMATICS


SMA Connector Bypass Caps (VCC Reference Plane)
For Low Frequency Return Currents < 50 MHz
CON5 SMA
CON17 SMA
CON18 SMA
CON19 SMA
CON11 SMA
CON16 SMA
CON6 SMA
CON7 SMA
CON8 SMA
1G SMA OUT
500M SMA OUT
250M SMA OUT
125M SMA OUT
62M SMA OUT
Trim Delay Trim Delay2.sch
Trim Delay Plain Trim Delay.sch
Trim Delay Plain Trim Delay2.sch
PECL-TTL Convertor PECL-TTL Convertor.sch
Trim Delay Plain Trim Delay3.sch
TTL Out
PECL In /PECL In
Delay Out /Delay Out
Delay In /Delay In
DATA[0..15]
16 Bit Data Bus
5 Bit Address Bus
ADDR[0..4]
External Connectors
External Connecters
DATA[0..15]
ADDR[0..4]
VCC
VTT
GND
CHIP_EN[0..17]
Address Decoder
Address Decoder.sch
PRF Address Out
PRF Chip Enable
GHz Divider
GHz Divider.Sch
1G Pecl Output
/1G Pecl Output
Sine Input
/Sine Input
500M Pecl Output 0
/500M Pecl Output 0
500M Pecl Output 1
/500M Pecl Output 1
CON15 SMA
CON2 SMA
Clock Frequency Divider
Frequency Divider.sch
250M Clock Out
/250M Clock Out
125M Clock Out 0
/125M Clock Out 0
Clock In
/Clock In
125M Clock Out 1
/125M Clock Out 1
62M Clock Out
/62M Clock Out
125M In
/125M In
62M In
62M Out
/62M In
/62M Out
MUX Out1
/MUX Out1
MUX Out2
/MUX Out2
Clock MUX
Clock MUX.sch
PRF Generator
PRF Generator.sch
PRF ADDRESS
DATA[0..15]
PRF
/PRF
PRF CLOCK OUT
PRF CLOCK
/PRF CLOCK
Single Shot PRF MUX
PRF MUX.sch
PRF MUX OUT
CLK IN /PRF MUX OUT
PRF MUX SELECT
/PRF ENABLE
SINGLE SHOT IN
CON12 SMA
CON13 SMA
PRF OUT
/PRF OUT
/125M Internal Out
125M Internal Out
CON3 SMA
CON4 SMA
PRF MUX IN
/PRF ENABLE
SINGLE SHOT IN
Low Speed TTL Inputs
CON9 SMA
CON10 SMA
CON14 SMA
Bypassing for VCC Reference Plane
Title
Mezzanine Card
Size
B
Number
Revision
Date: 20-Sep-2001
Sheet of
File: C:\officedrives\scsidrive\Design Explorer\revb\projects\Timing Board\Timing Board.ddb

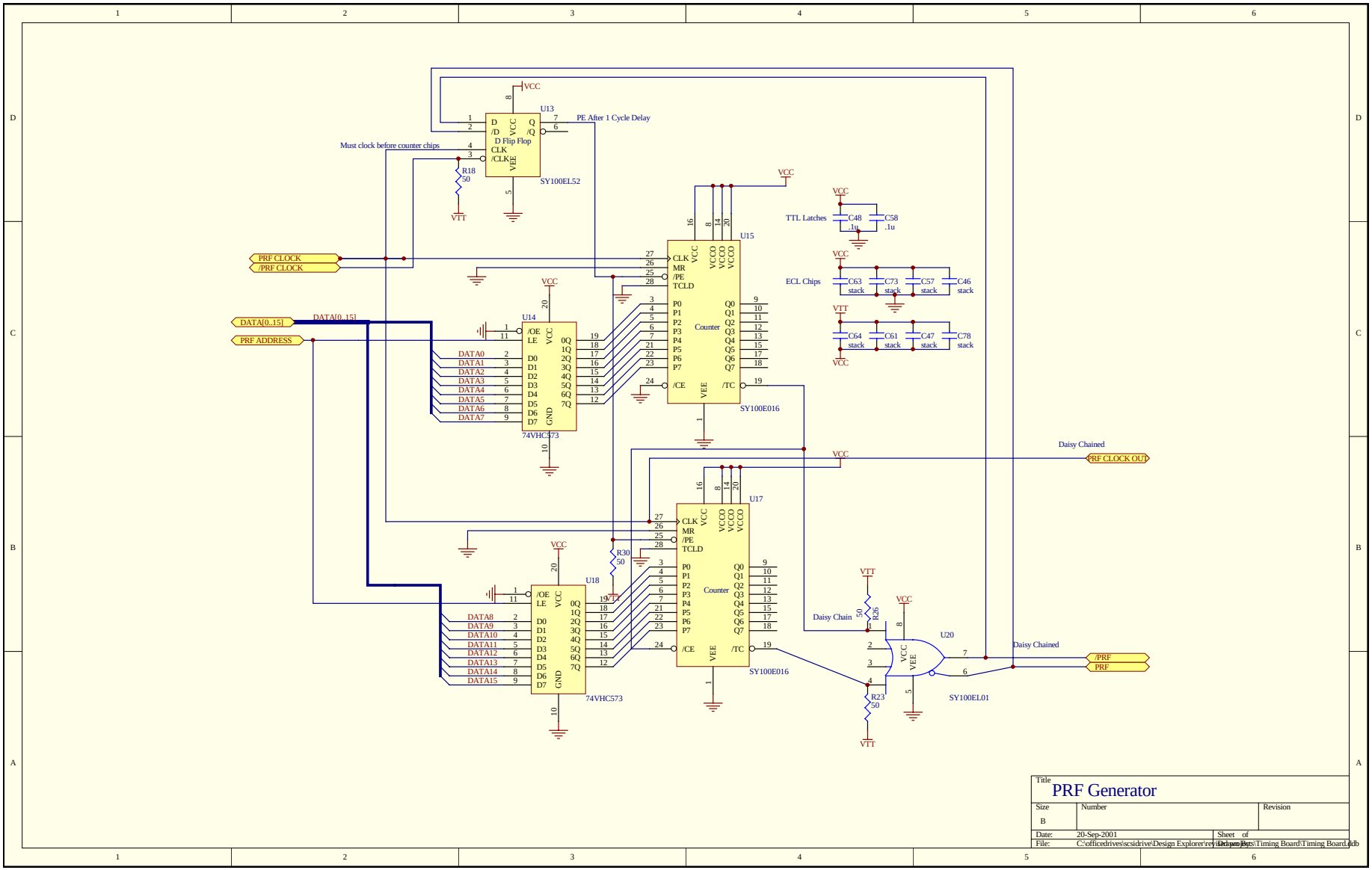


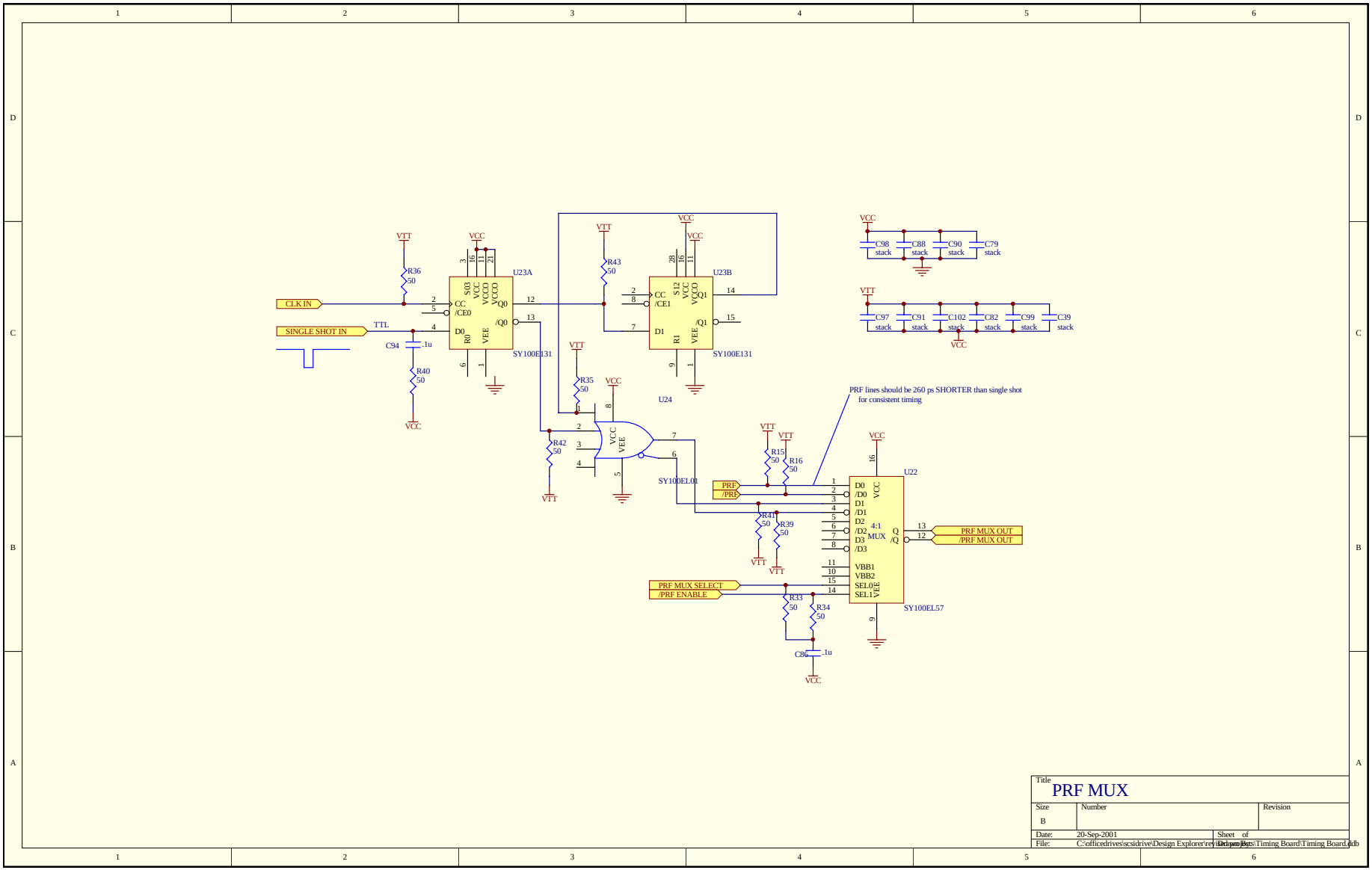
Title	GHz Divider	
Size: B	Number:	Revision:
Date: 20-Sep-2001	Sheet of	
File: C:\officedrives\scsidrive\Design Explorer\revisions\Timing Board\Timing Board.ddb		

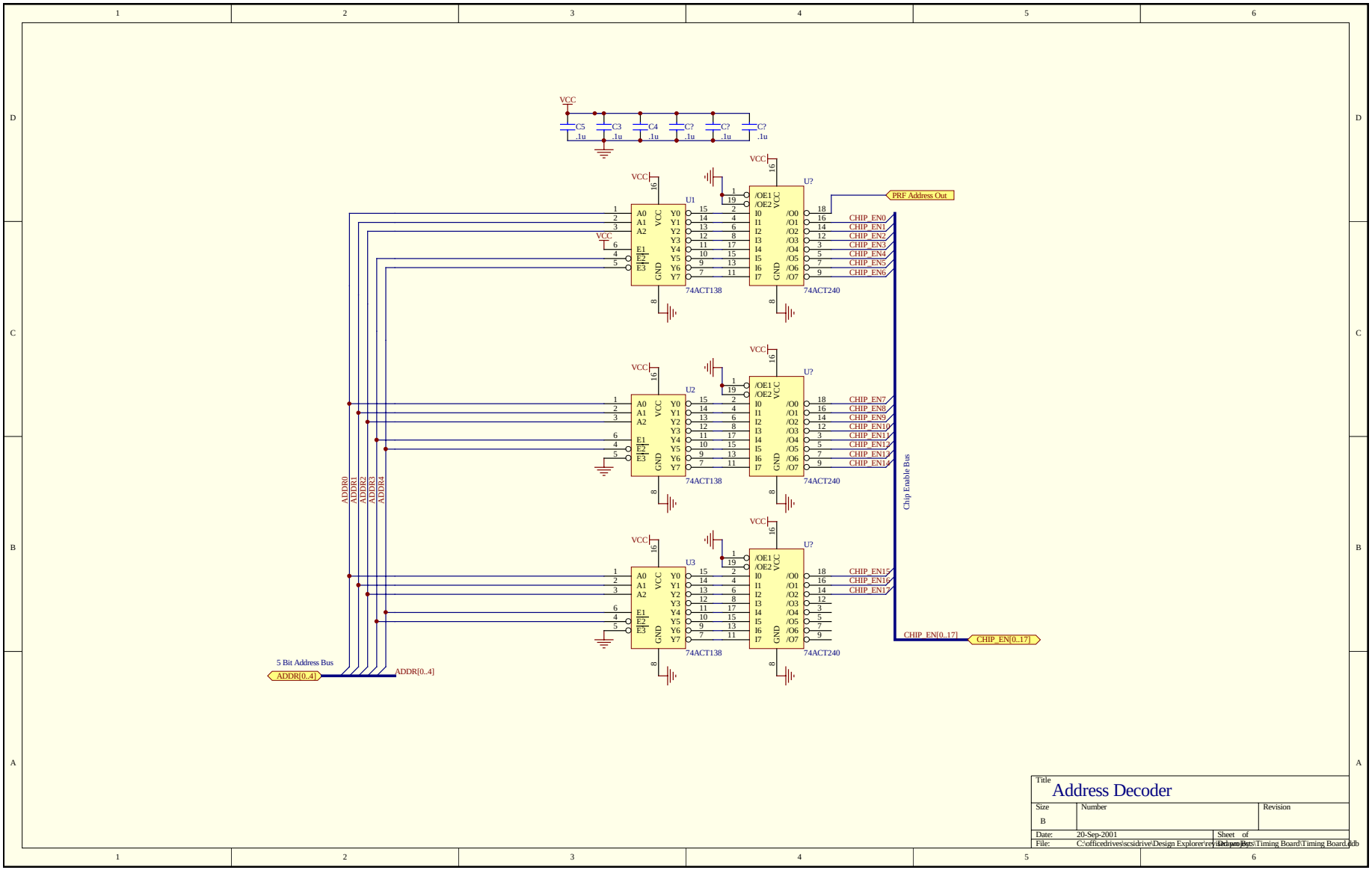

VCC
VTT
U4
U7
U10
U12
D
/D
Q
/Q
VCC
VEE
CLK
/CLK
D Flip Flop
SY100EL52
SY100EL11V
1:2
Driver
Clock In
/Clock In
R5
50
R6
50
R14
50
R17
50
Daisy Chain Termination
250M Clock Out
/250M Clock Out
62M Clock Out
/62M Clock Out
125M Clock Out 0
/125M Clock Out 0
125M Clock Out 1
/125M Clock Out 1
Stripline Bypassing, 62, 250M
C16
C41
C36
C10
C25
C34
C44
C18
C15
C37
C95
stack
Title
Frequency Divider
Size
B
Number
Revision
Date:
20-Sep-2001
Sheet of
File:
C:\officedrives\scsidrive\Design Explorer\reference\Timing Board\Timing Board.ddb

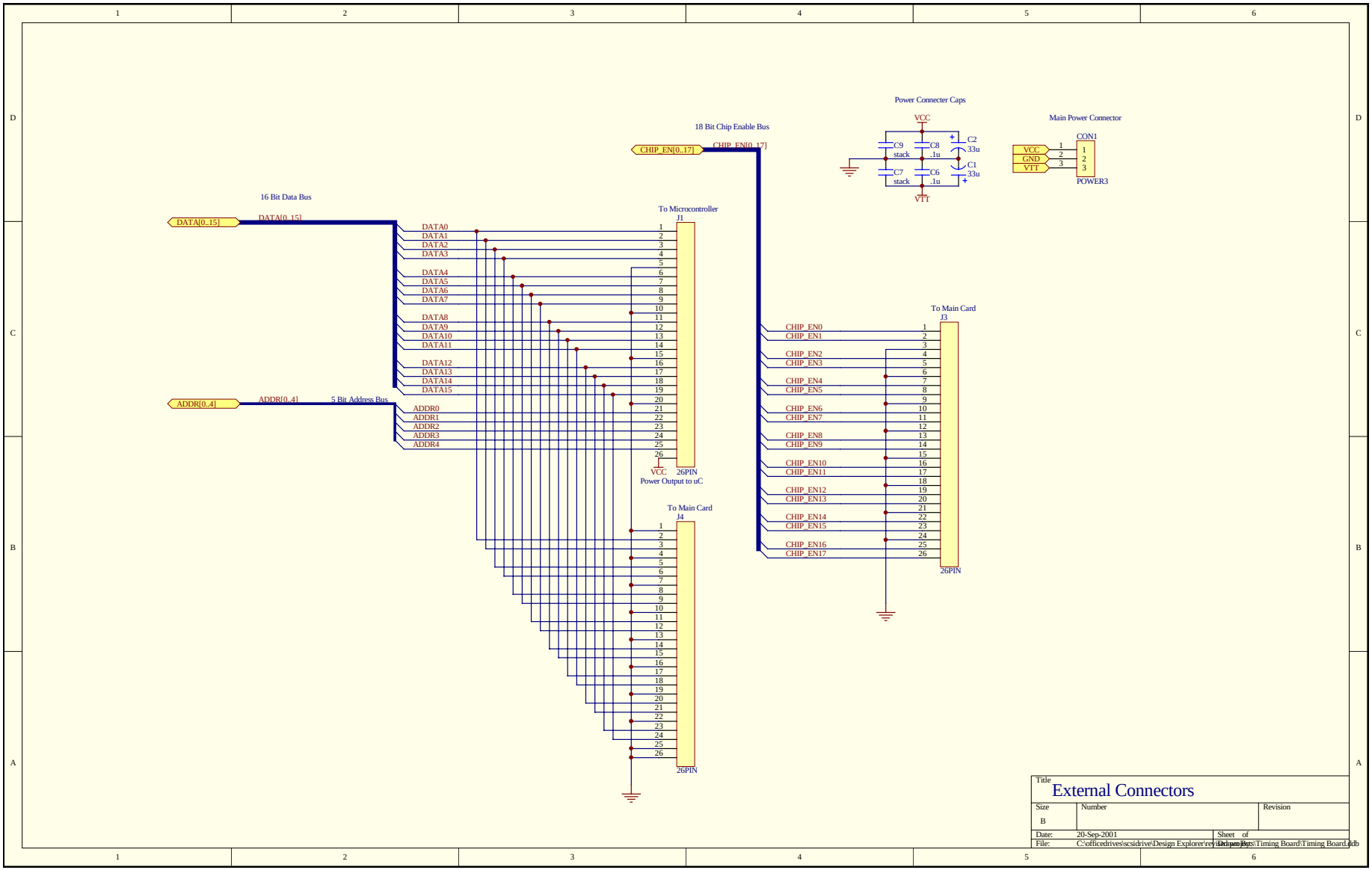


Title	Clock MUX		
Size	Number		Revision
B			
Date:	20-Sep-2001	Sheet	of
File:	C:\officedrives\scsidrive\Design Explorer\revisions\Timing Board\Timing Board.ddb		

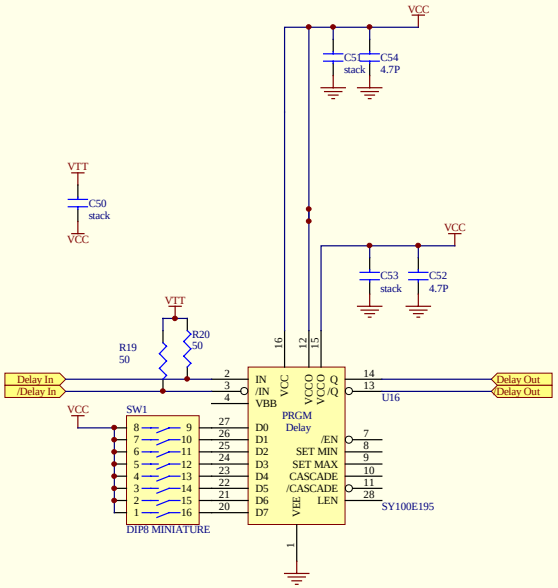

U13
D Flip Flop
SY100EL52
PE After 1 Cycle Delay
Must clock before counter chips
R18
50
VTT
VCC
PRF CLOCK
/PRF CLOCK
DATA[0..15]
PRF ADDRESS
U14
74VHC573
U15
SY100E016
Counter
TTL Latches
C48
.1u
C58
.1u
ECL Chips
C63
stack
C73
stack
C57
stack
C46
stack
C64
stack
C61
stack
C47
stack
C78
stack
Daisy Chained
PRF CLOCK OUT
U17
U18
R30
50
Daisy Chain
R26
50
U20
SY100EL01
R23
50
/PRF
PRF
Title
PRF Generator
Size
B
Number
Revision
Date:
20-Sep-2001
Sheet of
File:
C:\officedrives\scsidrive\Design Explorer\revisions\Timing Board\Timing Board.ddb


CLK IN
SINGLE SHOT IN
TTL
U23A
U23B
SY100E131
U24
SY100EL01
U22
SY100EL57
PRF
/PRF
PRF MUX SELECT
/PRF ENABLE
PRF MUX OUT
/PRF MUX OUT
PRF lines should be 260 ps SHORTER than single shot for consistent timing
VTT
VCC
R36 50
R43 50
R35 50
R42 50
R15 50
R16 50
R41 50
R39 50
R33 50
R34 50
R40 50
C94 .1u
C86 .1u
C98 stack
C88 stack
C90 stack
C79 stack
C97 stack
C91 stack
C102 stack
C82 stack
C99 stack
C39 stack
Title
PRF MUX
Size
B
Number
Revision
Date: 20-Sep-2001
Sheet of
File: C:\officedrives\scsidrive\Design Explorer\revisions\Timing Board\Timing Board.ddb

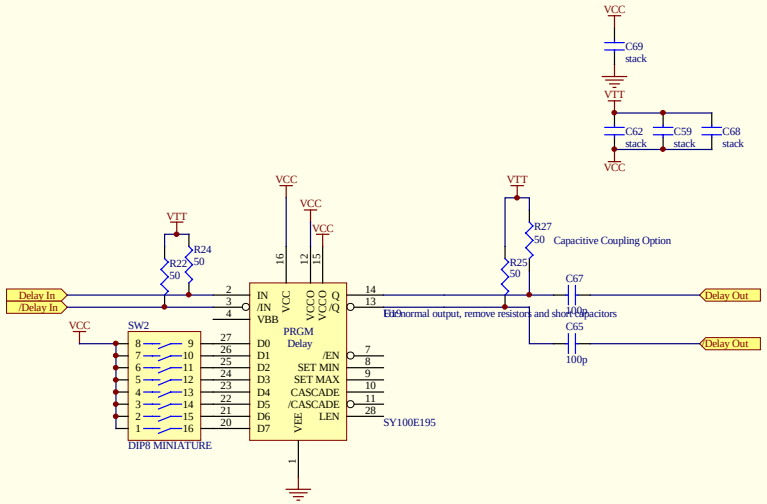

VCC
C5 .1u
C3 .1u
C4 .1u
C? .1u
C? .1u
C? .1u
U1
U2
U3
U?
74ACT138
74ACT240
PRF Address Out
CHIP_EN0
CHIP_EN1
CHIP_EN2
CHIP_EN3
CHIP_EN4
CHIP_EN5
CHIP_EN6
CHIP_EN7
CHIP_EN8
CHIP_EN9
CHIP_EN10
CHIP_EN11
CHIP_EN12
CHIP_EN13
CHIP_EN14
CHIP_EN15
CHIP_EN16
CHIP_EN17
Chip Enable Bus
CHIP_EN[0..17]
ADDR0
ADDR1
ADDR2
ADDR3
ADDR4
5 Bit Address Bus
ADDR[0..4]
Title
Address Decoder
Size
B
Number
Revision
Date: 20-Sep-2001
Sheet of
File: C:\officedrives\scsidrive\Design Explorer\revisions\Timing Board\Timing Board.ddb


Power Connecter Caps
VCC
C9
stack
C8
.1u
C2
33u
C1
33u
C7
stack
C6
.1u
VTT
Main Power Connector
CON1
VCC
GND
VTT
POWER3
18 Bit Chip Enable Bus
CHIP_EN[0..17]
16 Bit Data Bus
DATA[0..15]
ADDR[0..4]
5 Bit Address Bus
To Microcontroller
J1
DATA0
DATA1
DATA2
DATA3
DATA4
DATA5
DATA6
DATA7
DATA8
DATA9
DATA10
DATA11
DATA12
DATA13
DATA14
DATA15
ADDR0
ADDR1
ADDR2
ADDR3
ADDR4
VCC
26PIN
Power Output to uC
To Main Card
J3
CHIP_EN0
CHIP_EN1
CHIP_EN2
CHIP_EN3
CHIP_EN4
CHIP_EN5
CHIP_EN6
CHIP_EN7
CHIP_EN8
CHIP_EN9
CHIP_EN10
CHIP_EN11
CHIP_EN12
CHIP_EN13
CHIP_EN14
CHIP_EN15
CHIP_EN16
CHIP_EN17
26PIN
To Main Card
J4
26PIN
Title
External Connectors
Size
B
Number
Revision
Date:
20-Sep-2001
Sheet of
File:
C:\officedrives\scsidrive\Design Explorer\revisions\Timing Board\Timing Board.ddb

High Quality Bypassing For Low Jitter ADC Clock Output

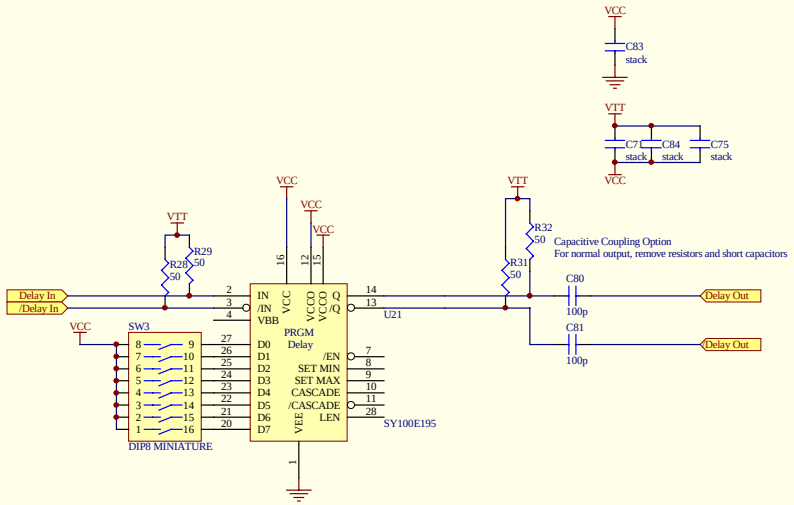


Title: Trim Delay 2		
Size: B	Number:	Revision:
Date: 20-Sep-2001	Sheet of	
File: C:\officedrives\scsidrive\Design Explorer\revisions\Timing Board\Timing Board.ddb		

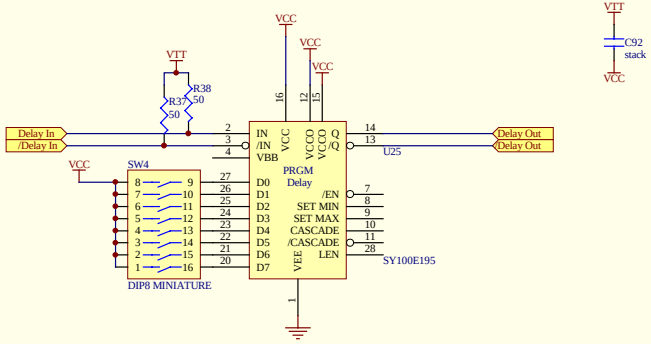


Title
Plain Trim Delay

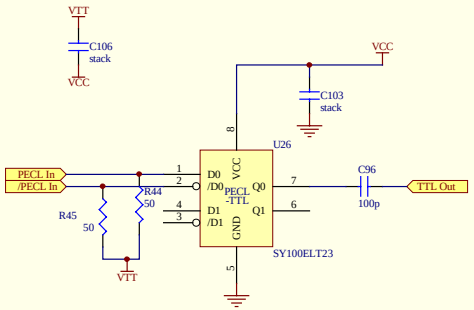
Size	Number	Revision
B		
Date: 20-Sep-2001	Sheet of	
File: C:\officedrives\scsidrive\Design Explorer\revisions\Schematics\Timing Board\Timing Board.ddb		



Title	Plain Trim Delay 2		
Size: B	Number:		Revision:
Date: 20-Sep-2001		Sheet of	
File: C:\officedrives\scsidrive\Design Explorer\revisions\Schematics\Timing Board\Timing Board.ddb			

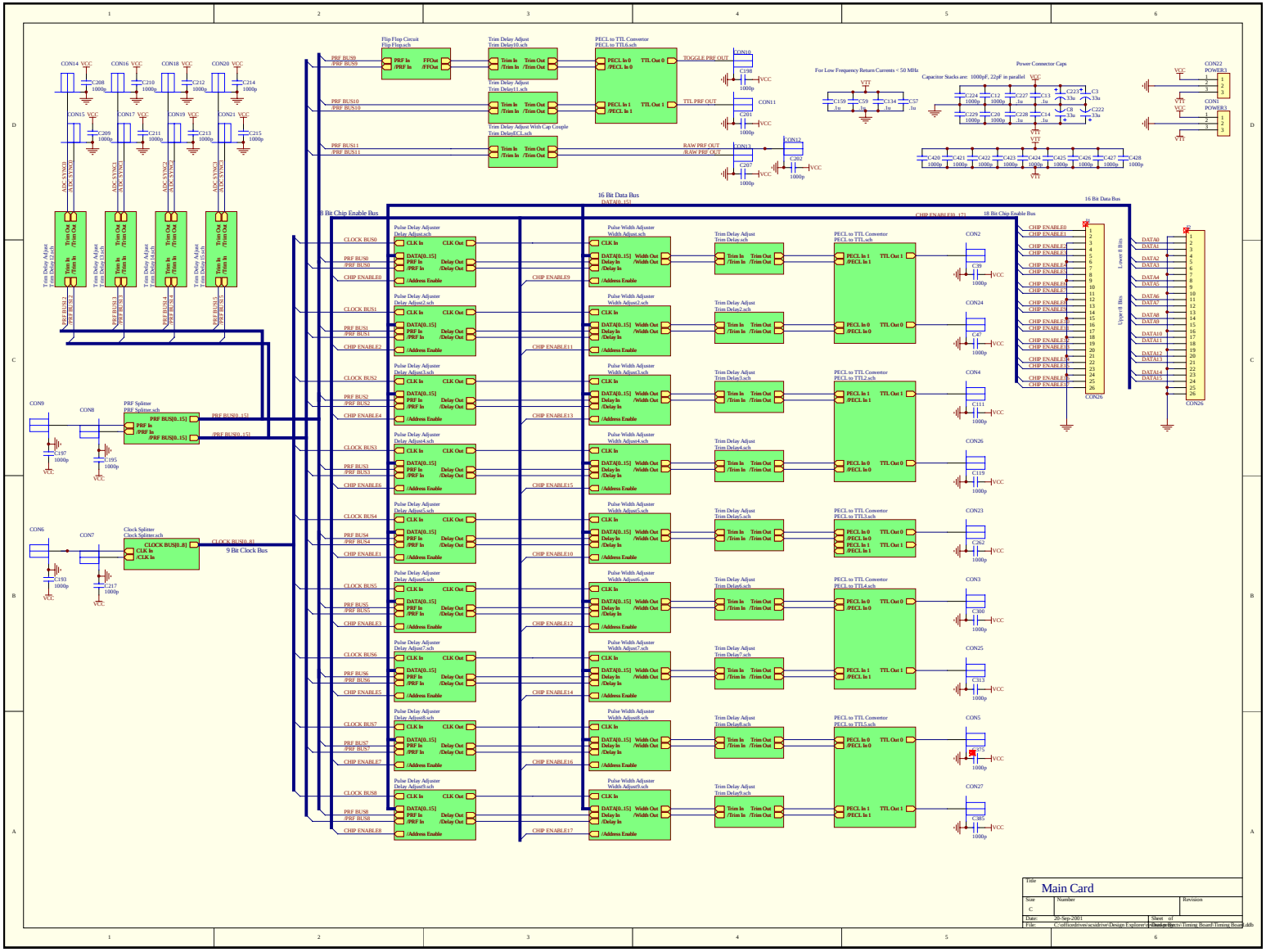


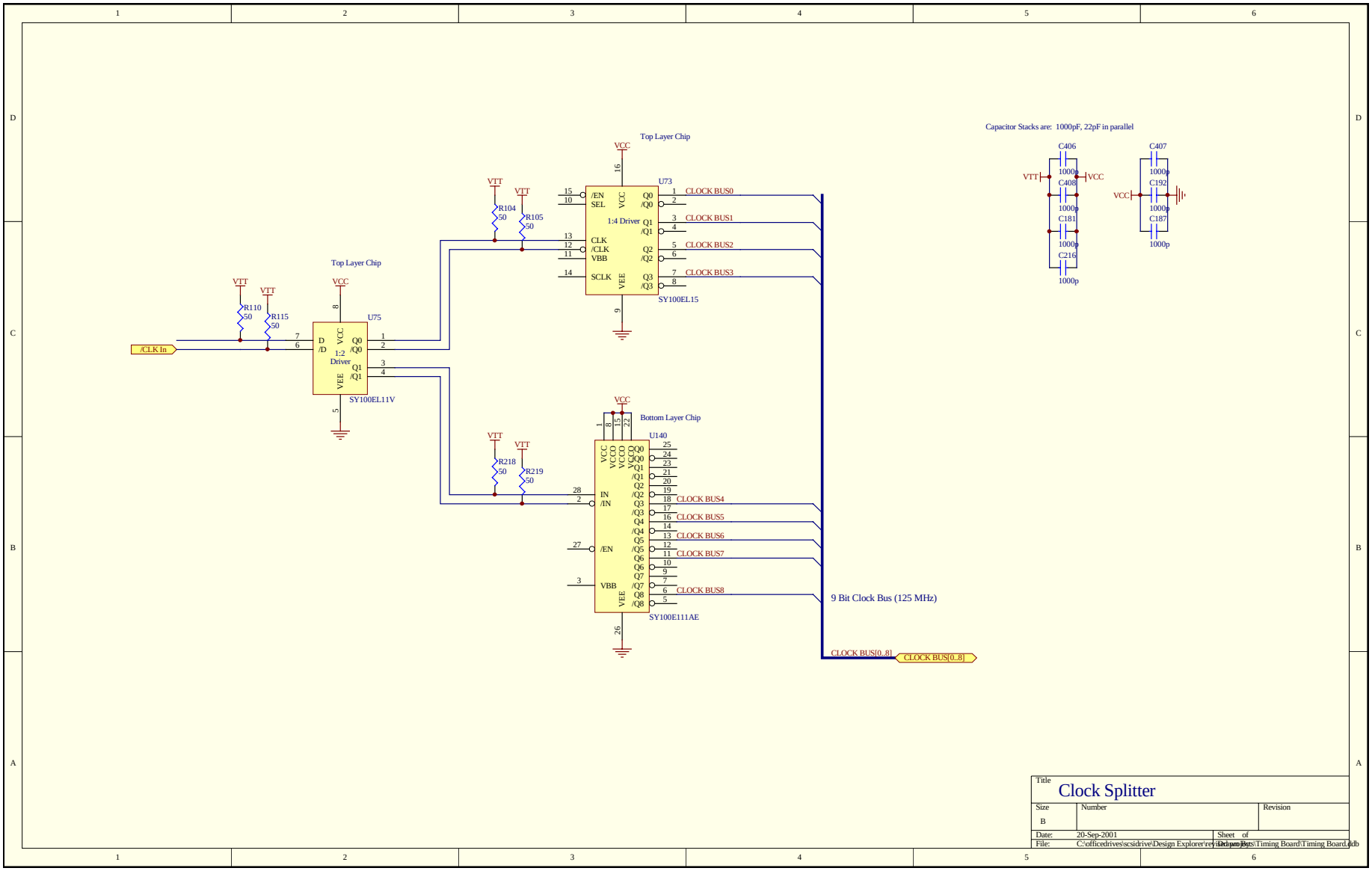
Title	Plain Trim Delay 3		
Size: B	Number:		Revision:
Date: 20-Sep-2001		Sheet of	
File: C:\officedrives\scsidrive\Design Explorer\revisions\Schematics\Timing Board\Timing Board.ddb			

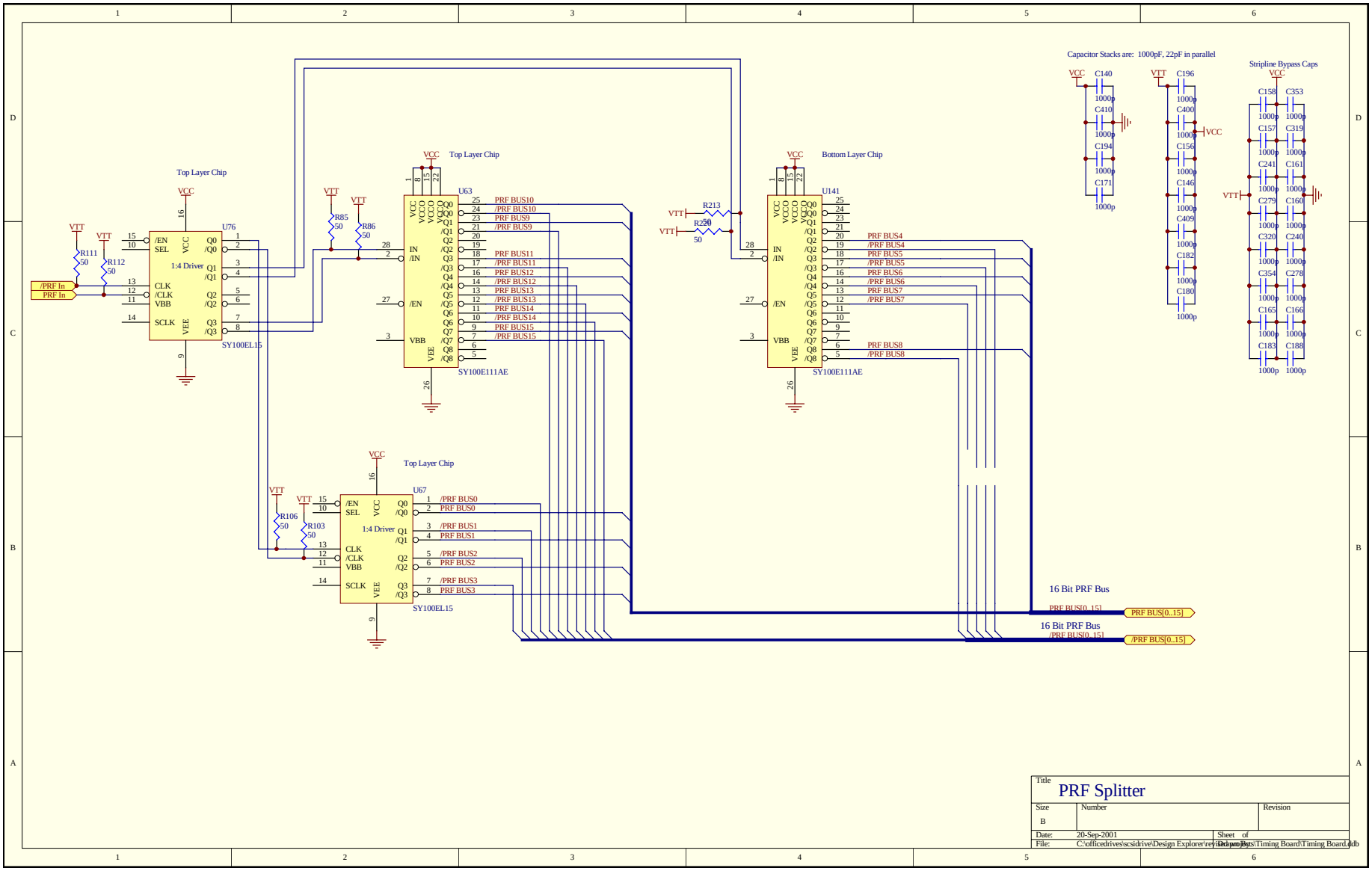


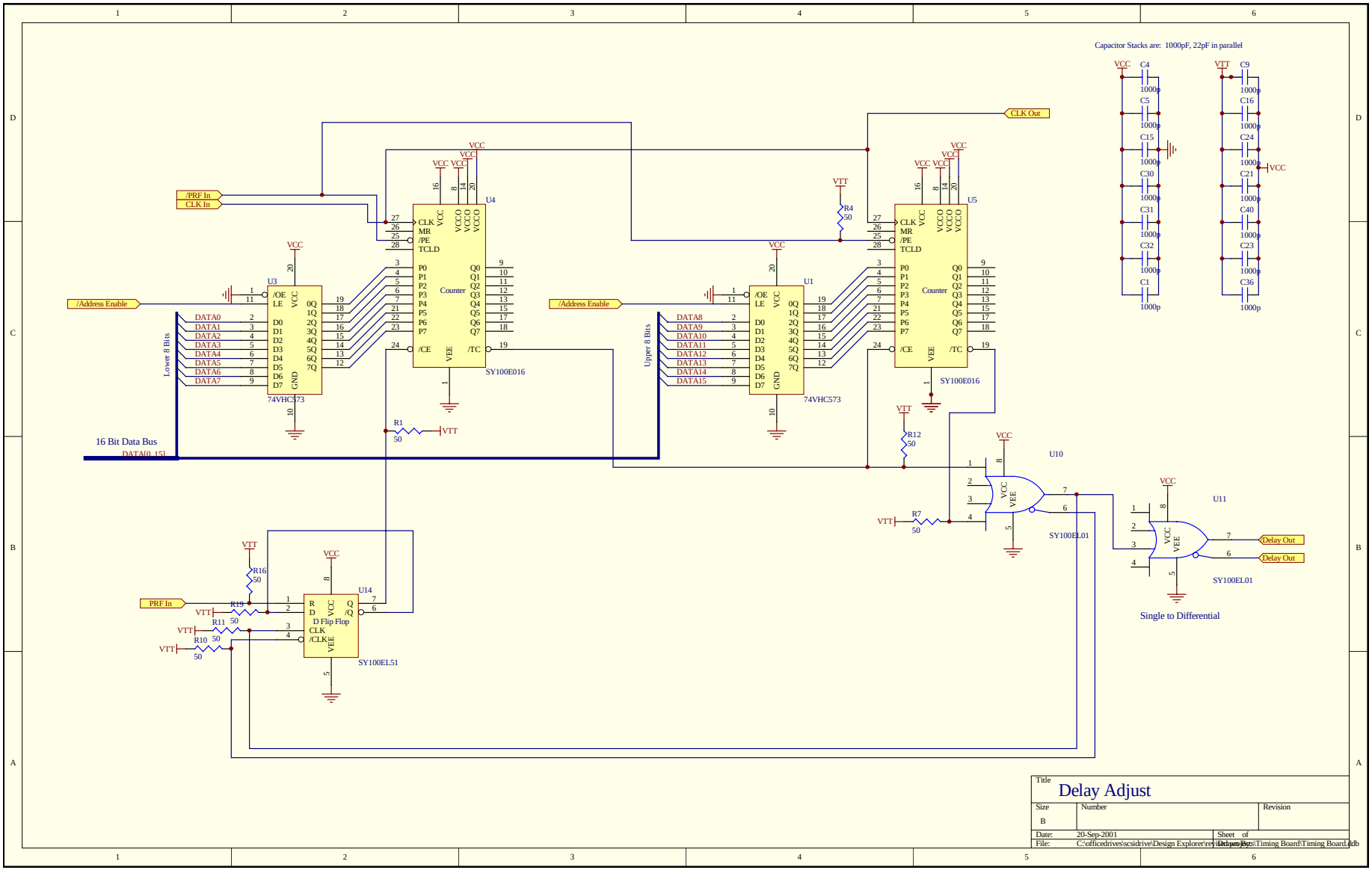
Title	PECL-TTL Convertor		
Size: B	Number:	Revision:	
Date: 20-Sep-2001	Sheet of		
File: C:\officedrives\scsidrive\Design Explorer\revisions\Schematics\Timing Board\Timing Board.ddb			

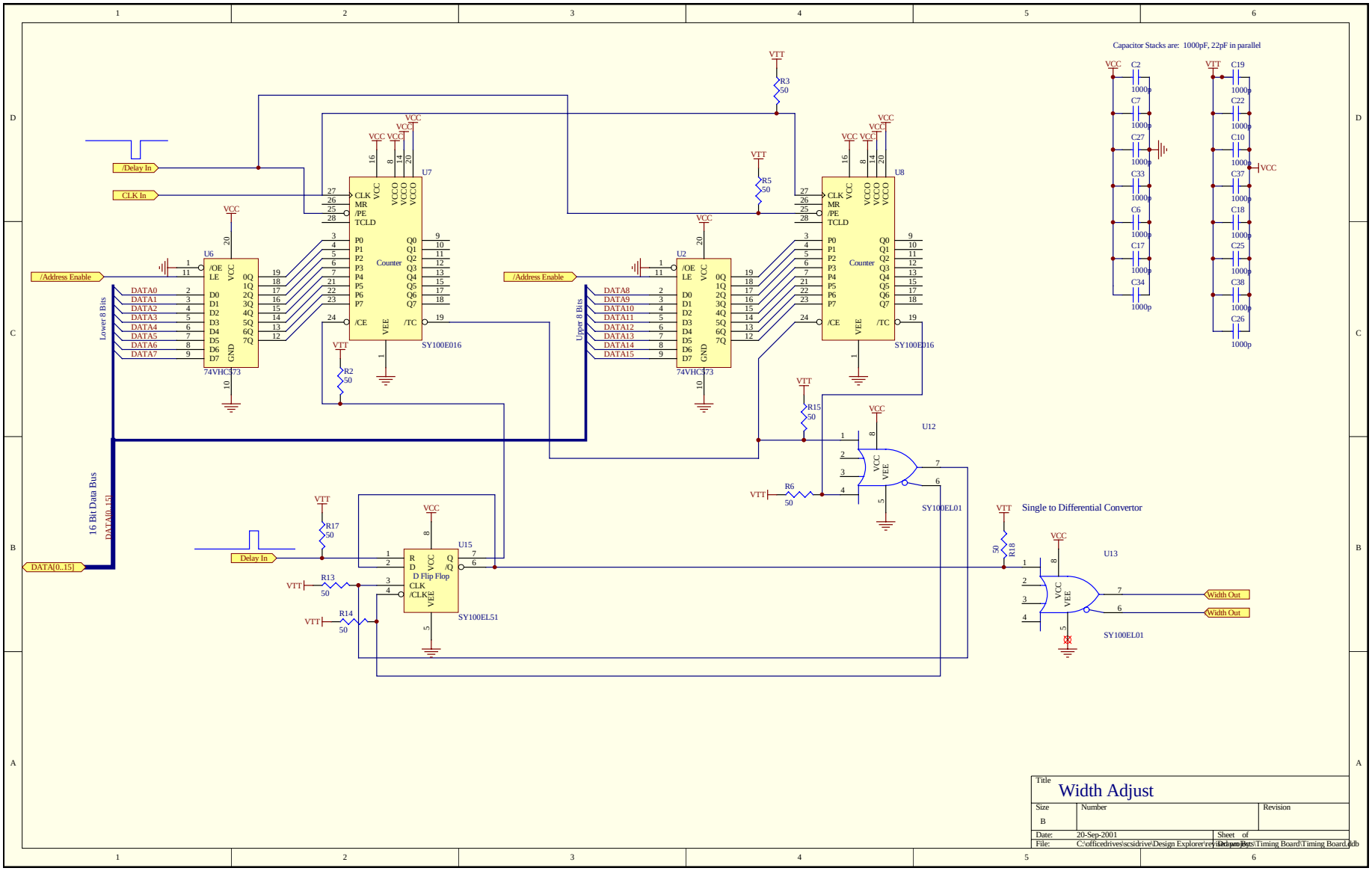
APPENDIX E TIMING MAIN CARD SCHEMATICS

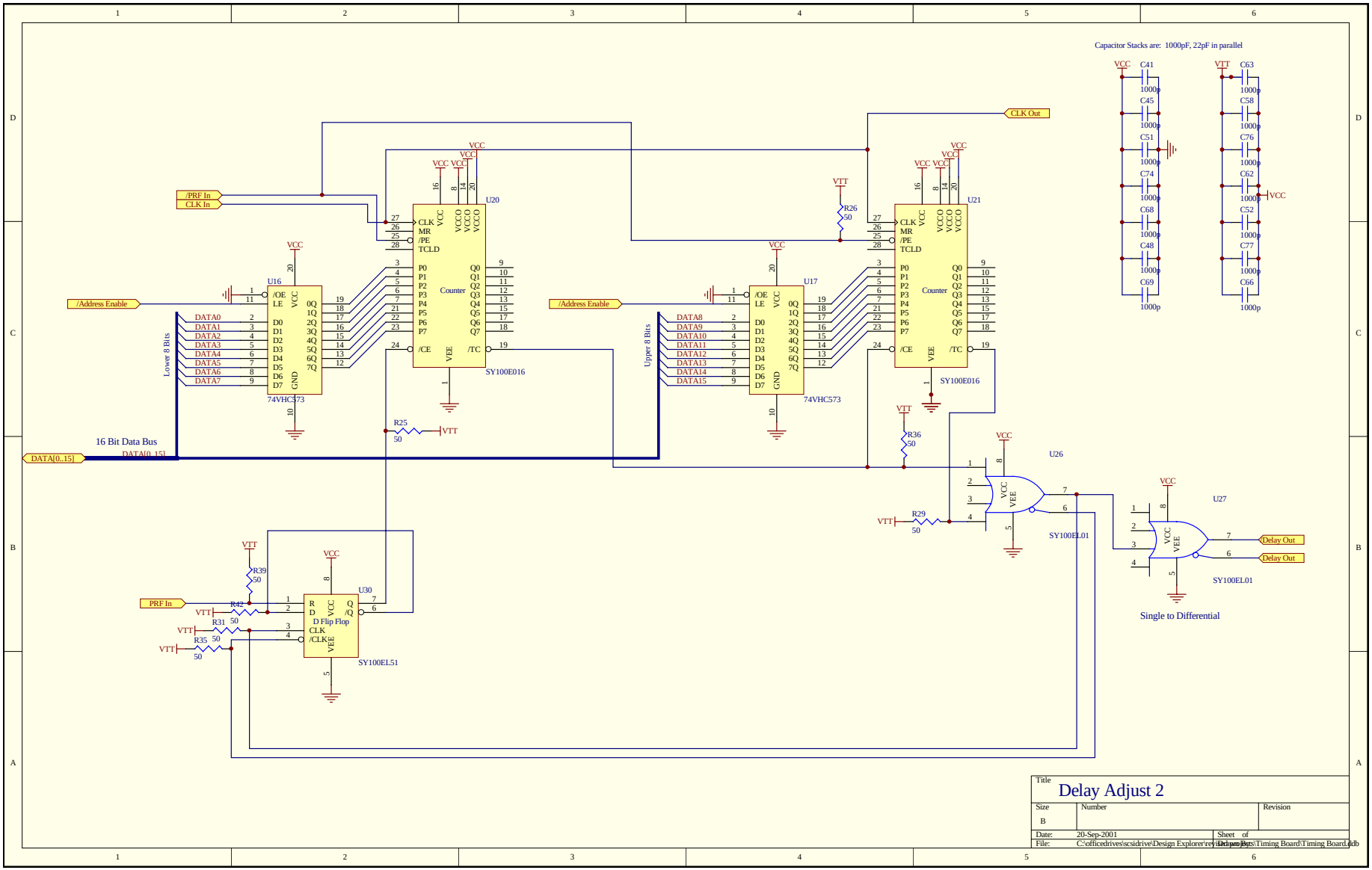

Main Card
Flip Flop Circuit
Trim Delay Adjust
PECL to TTL Convertor
Power Connector Caps
16 Bit Data Bus
9 Bit Chip Enable Bus
18 Bit Chip Enable Bus
Pulse Delay Adjuster
Pulse Width Adjuster
PECL to TTL Convertor
PRF Splitter
Clock Splitter
9 Bit Clock Bus
Upper8 Bits
Lower 8 Bits

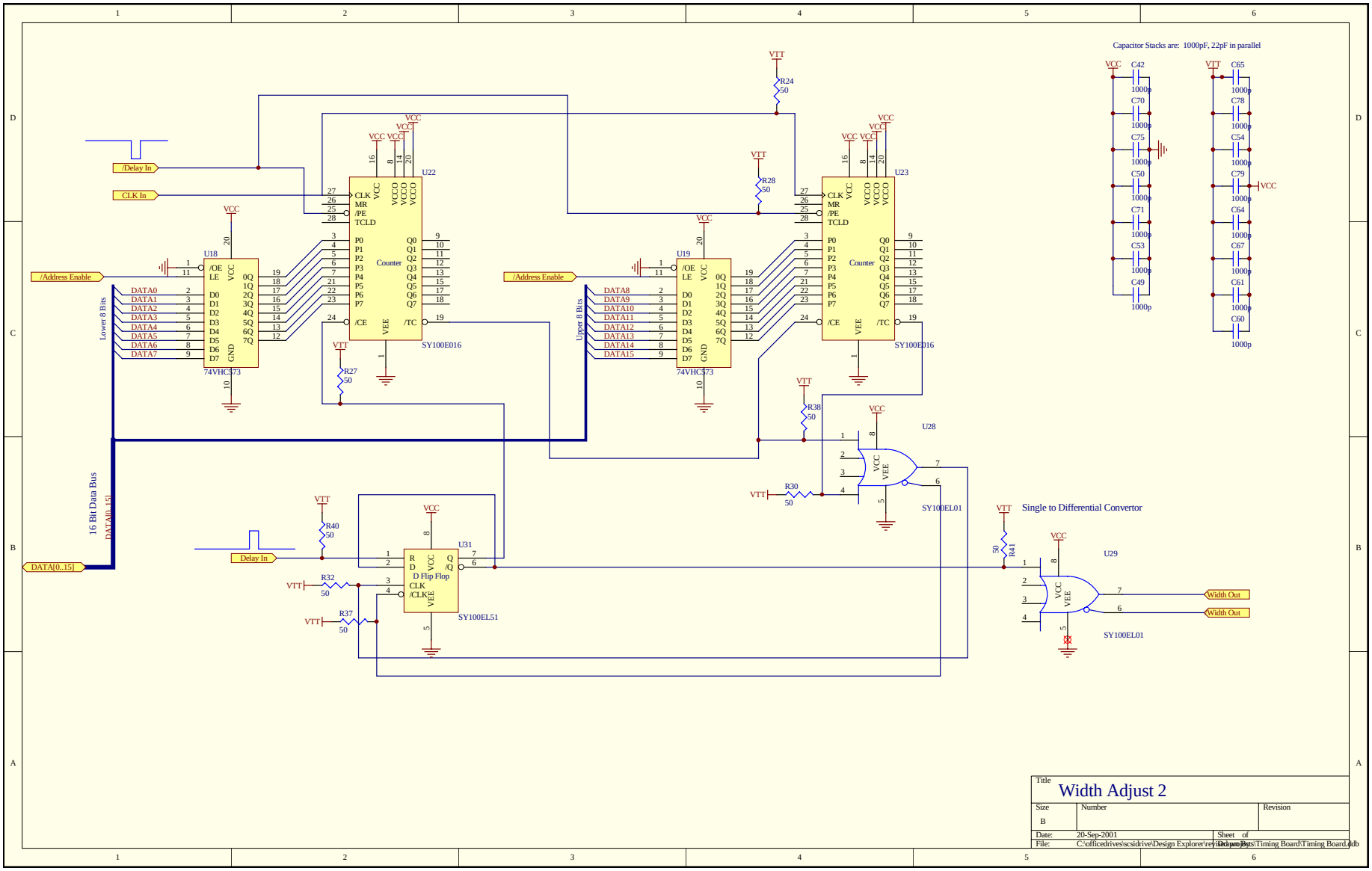

Capacitor Stacks are: 1000pF, 22pF in parallel
C406
C408
C181
C216
C407
C192
C187
1000p
VTT
VCC
Top Layer Chip
U73
/EN
SEL
1:4 Driver
CLK
/CLK
VBB
SCLK
VCC
VEE
Q0
/Q0
Q1
/Q1
Q2
/Q2
Q3
/Q3
CLOCK BUS0
CLOCK BUS1
CLOCK BUS2
CLOCK BUS3
SY100EL15
R104
R105
50
Top Layer Chip
U75
/CLK In
D
/D
1:2
Driver
SY100EL11V
R110
R115
Bottom Layer Chip
U140
VCCO
IN
/IN
/EN
SY100E111AE
R218
R219
CLOCK BUS4
CLOCK BUS5
CLOCK BUS6
CLOCK BUS7
CLOCK BUS8
9 Bit Clock Bus (125 MHz)
CLOCK BUS[0..8]
Title
Clock Splitter
Size
B
Number
Revision
Date: 20-Sep-2001
Sheet of
File: C:\officedrives\scsidrive\Design Explorer\revisions\Schematics\Timing Board\Timing Board.ddb

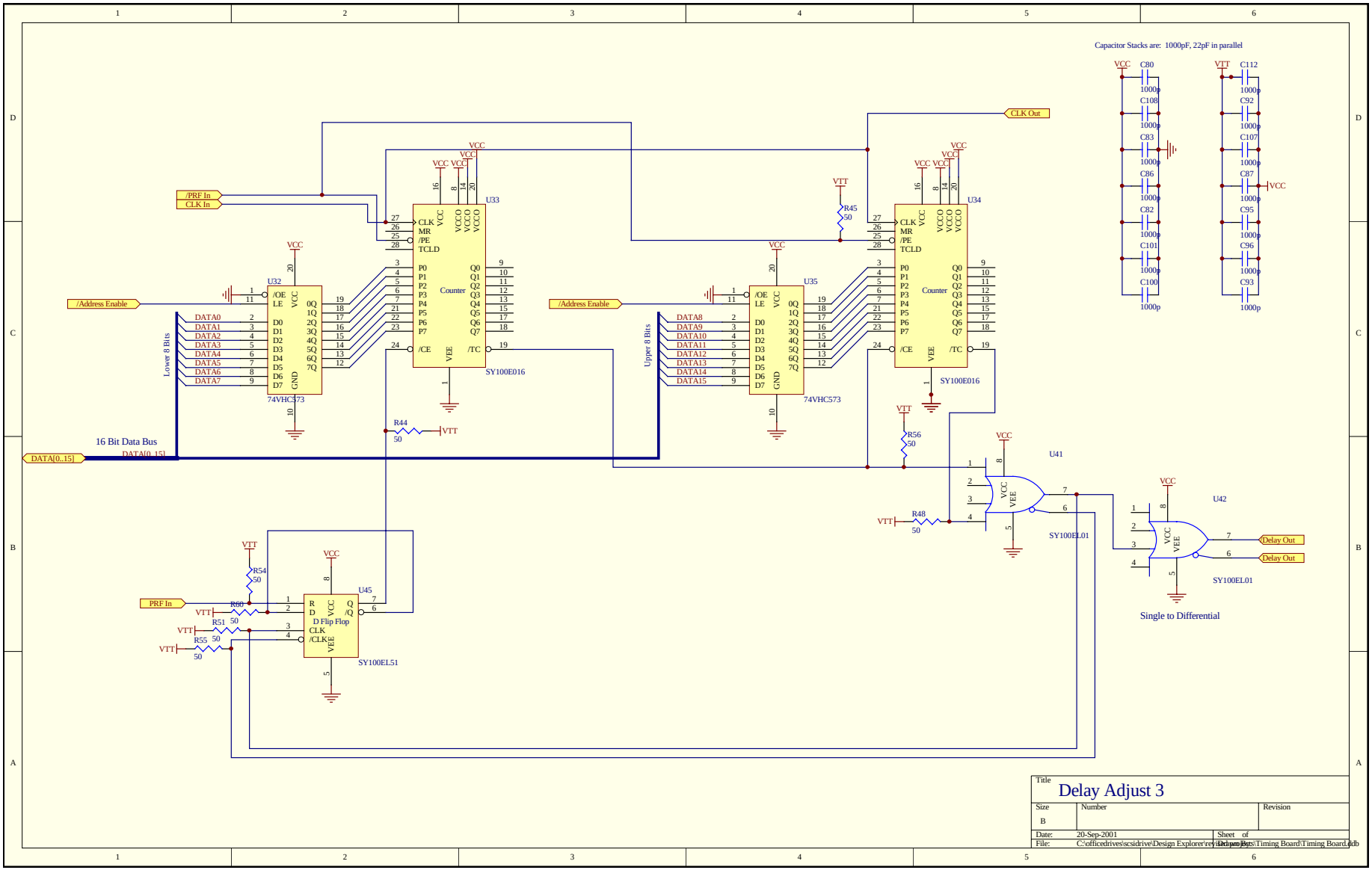

Capacitor Stacks are: 1000pF, 22pF in parallel
Stripline Bypass Caps
Top Layer Chip
Top Layer Chip
Bottom Layer Chip
Top Layer Chip
U76
U63
U141
U67
SY100EL15
SY100E111AE
SY100E111AE
SY100EL15
1:4 Driver
/PRF In
PRF In
16 Bit PRF Bus
PRF BUS[0..15]
16 Bit PRF Bus
/PRF BUS[0..15]
PRF Splitter
Size
B
Number
Revision
Date: 20-Sep-2001
Sheet of
File: C:\officedrives\scsidrive\Design Explorer\revisions\Timing Board\Timing Board.ddb

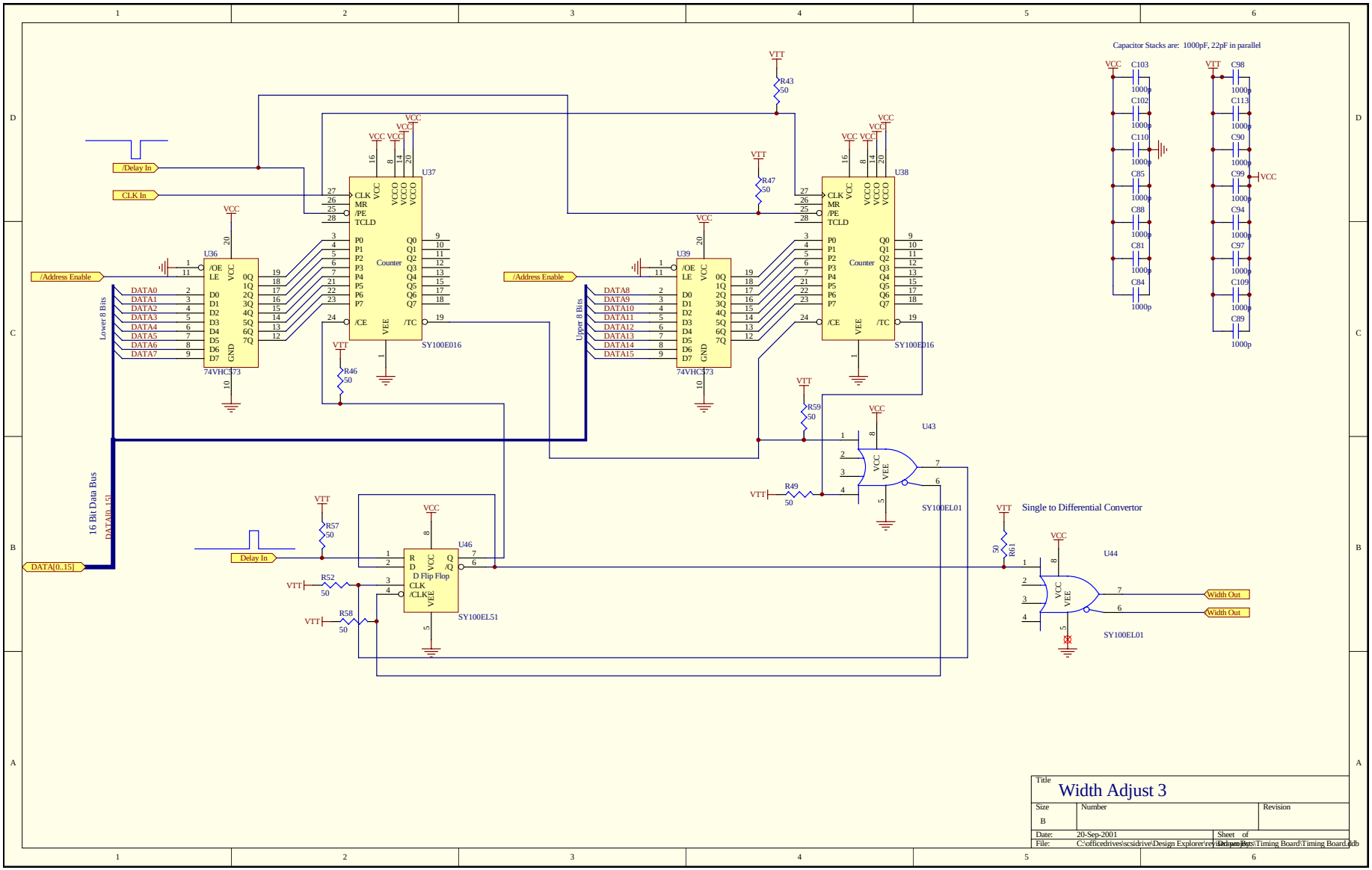

Capacitor Stacks are: 1000pF, 22pF in parallel
/PRF In
CLK In
CLK Out
/Address Enable
Lower 8 Bits
Upper 8 Bits
16 Bit Data Bus
DATA0_15
74VHC573
SY100E016
SY100EL51
SY100EL01
Counter
D Flip Flop
Single to Differential
Delay Out
Delay Out
Title
Delay Adjust
Size
B
Number
Revision
Date: 20-Sep-2001
Sheet of
File: C:\officedrives\scsidrive\Design Explorer\revisions\Timing Board\Timing Board.ddb

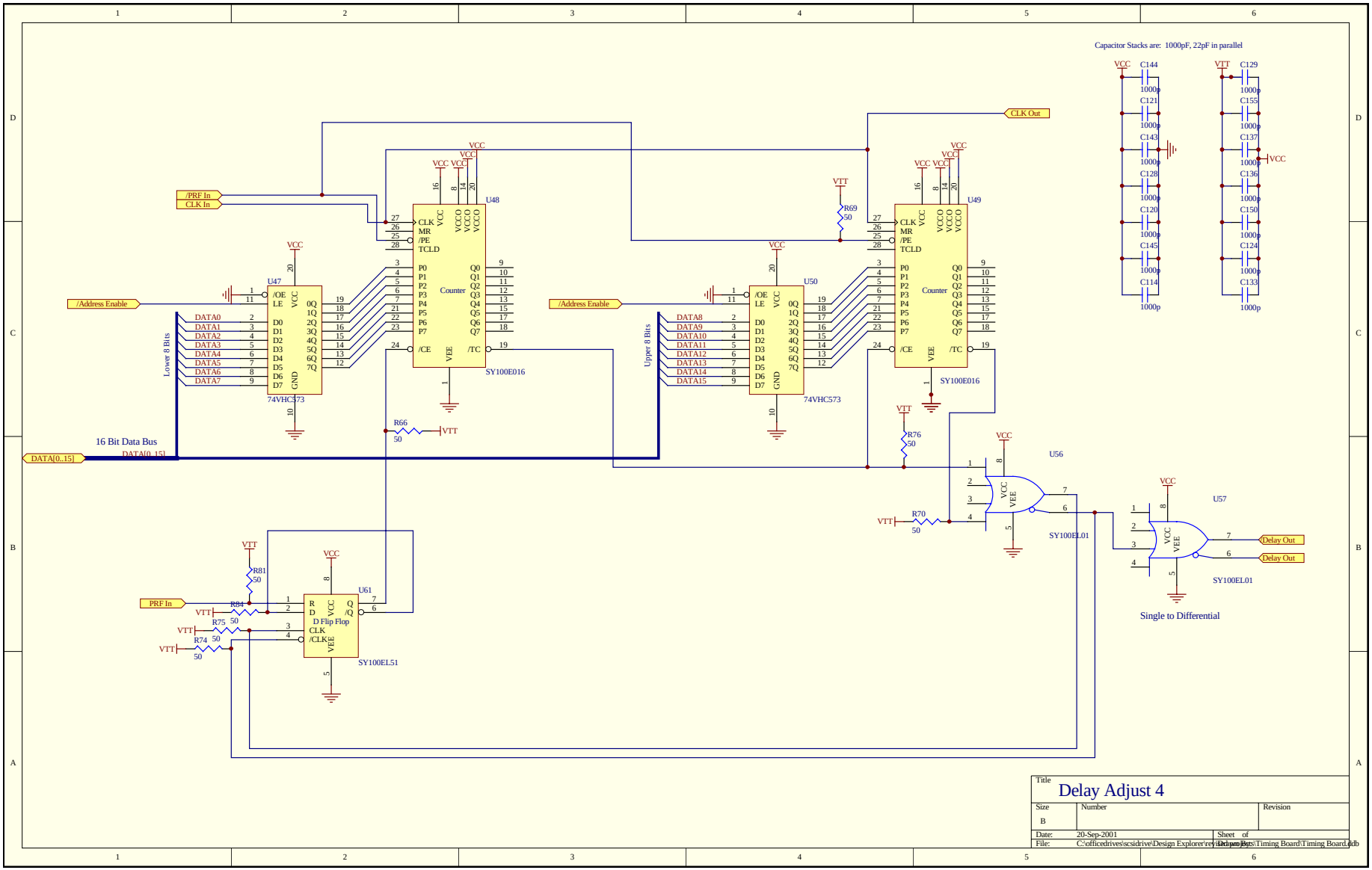

Capacitor Stacks are: 1000pF, 22pF in parallel
/Delay In
CLK In
/Address Enable
Lower 8 Bits
Upper 8 Bits
16 Bit Data Bus
DATA[0..15]
U6
U7
U8
U2
74VHC373
SY100E016
U12
SY100EL01
U15
D Flip Flop
SY100EL51
U13
Single to Differential Convertor
Width Out
Counter
Title
Width Adjust
Size
B
Number
Revision
Date: 20-Sep-2001
Sheet of
File: C:\officedrives\scsidrive\Design Explorer\revisions\Schematics\Timing Board\Timing Board.ddb

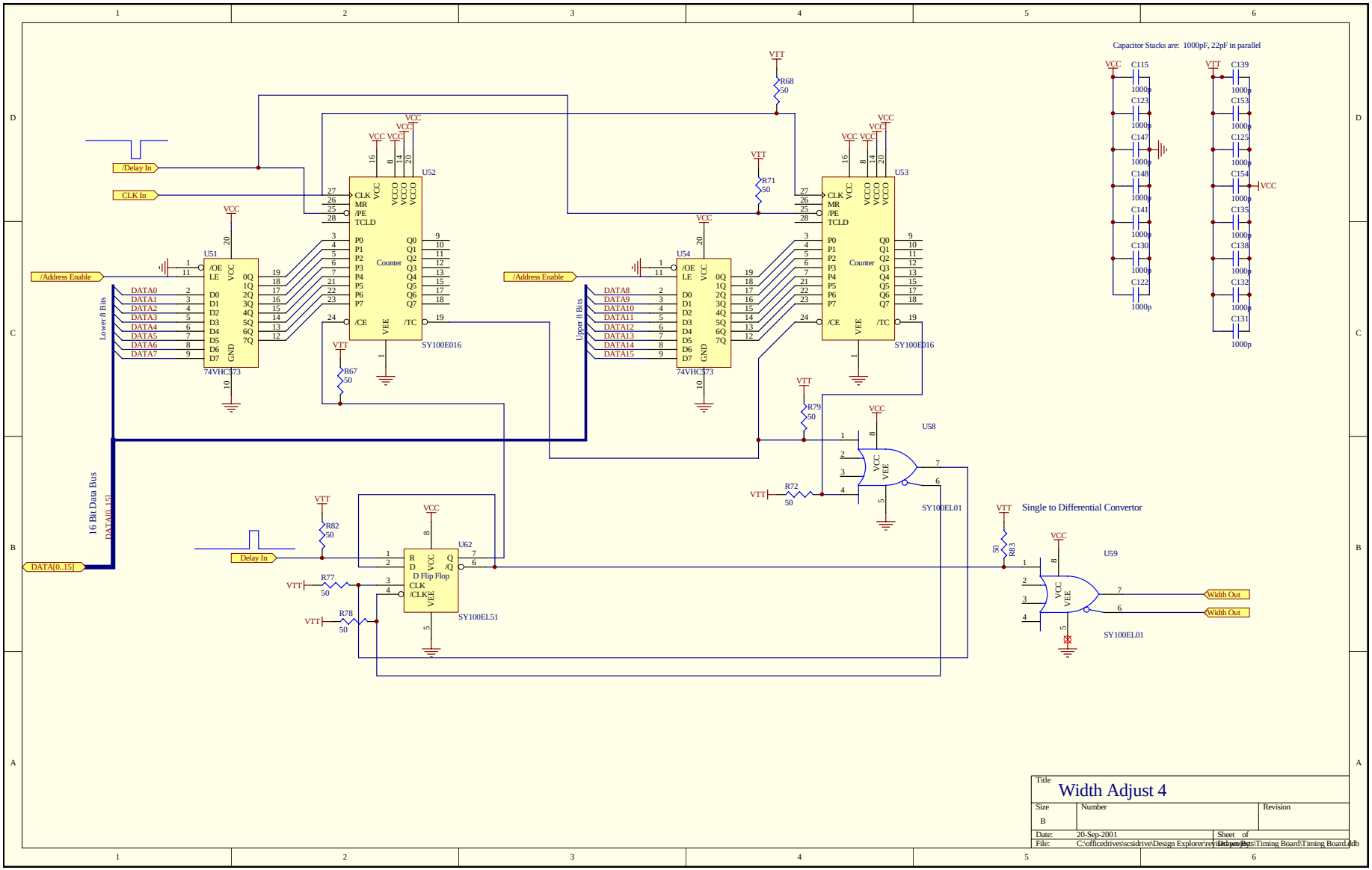

Capacitor Stacks are: 1000pF, 22pF in parallel
C41 1000p
C45 1000p
C51 1000p
C74 1000p
C68 1000p
C48 1000p
C69 1000p
C63 1000p
C58 1000p
C76 1000p
C62 1000p
C52 1000p
C77 1000p
C66 1000p
/PRF In
CLK In
CLK Out
U20
U21
U16
U17
U26
U27
U30
Counter
SY100E016
74VHC573
/Address Enable
Lower 8 Bits
Upper 8 Bits
DATA0
DATA1
DATA2
DATA3
DATA4
DATA5
DATA6
DATA7
DATA8
DATA9
DATA10
DATA11
DATA12
DATA13
DATA14
DATA15
16 Bit Data Bus
DATA[0..15]
R25 50
R26 50
R36 50
R29 50
R39 50
R42 50
R31 50
R35 50
VTT
VCC
D Flip Flop
SY100EL51
SY100EL01
Delay Out
Single to Differential
Title
Delay Adjust 2
Size
B
Number
Revision
Date: 20-Sep-2001
Sheet of
File: C:\officedrives\scsidrive\Design Explorer\revisions\Delay Bits\Timing Board\Timing Board.ddb

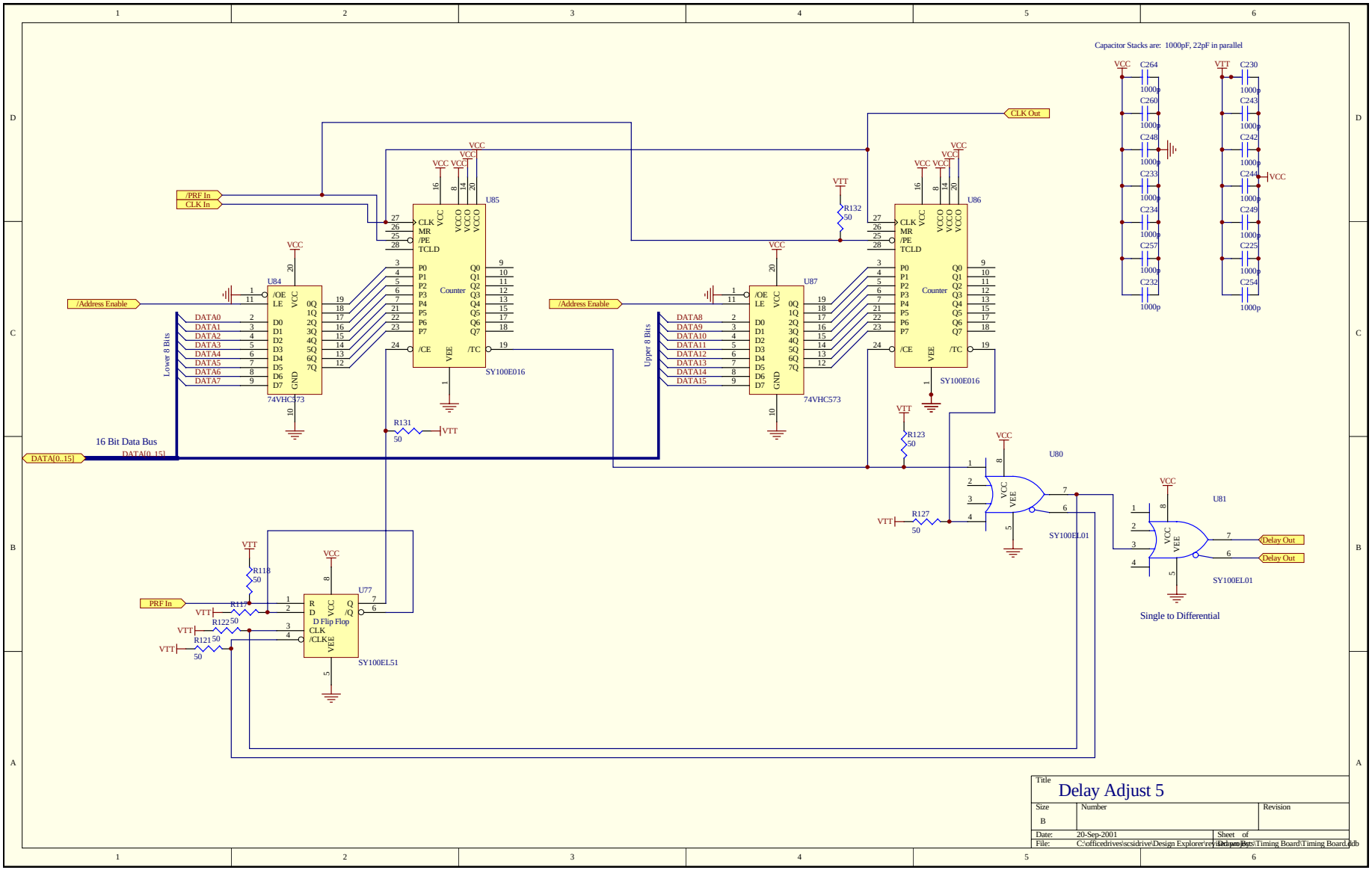

Capacitor Stacks are: 1000pF, 22pF in parallel
/Delay In
CLK In
/Address Enable
Lower 8 Bits
Upper 8 Bits
16 Bit Data Bus
DATA[0..15]
U18
U19
U22
U23
U28
U29
U31
74VHC373
SY100E016
SY100EL01
SY100EL51
D Flip Flop
Counter
Single to Differential Convertor
Width Out
Width Out
Title
Width Adjust 2
Size
B
Number
Revision
Date: 20-Sep-2001
Sheet of
File: C:\officedrives\scsidrive\Design Explorer\revisions\Timing Boards\Timing Board\Timing Board.ddb

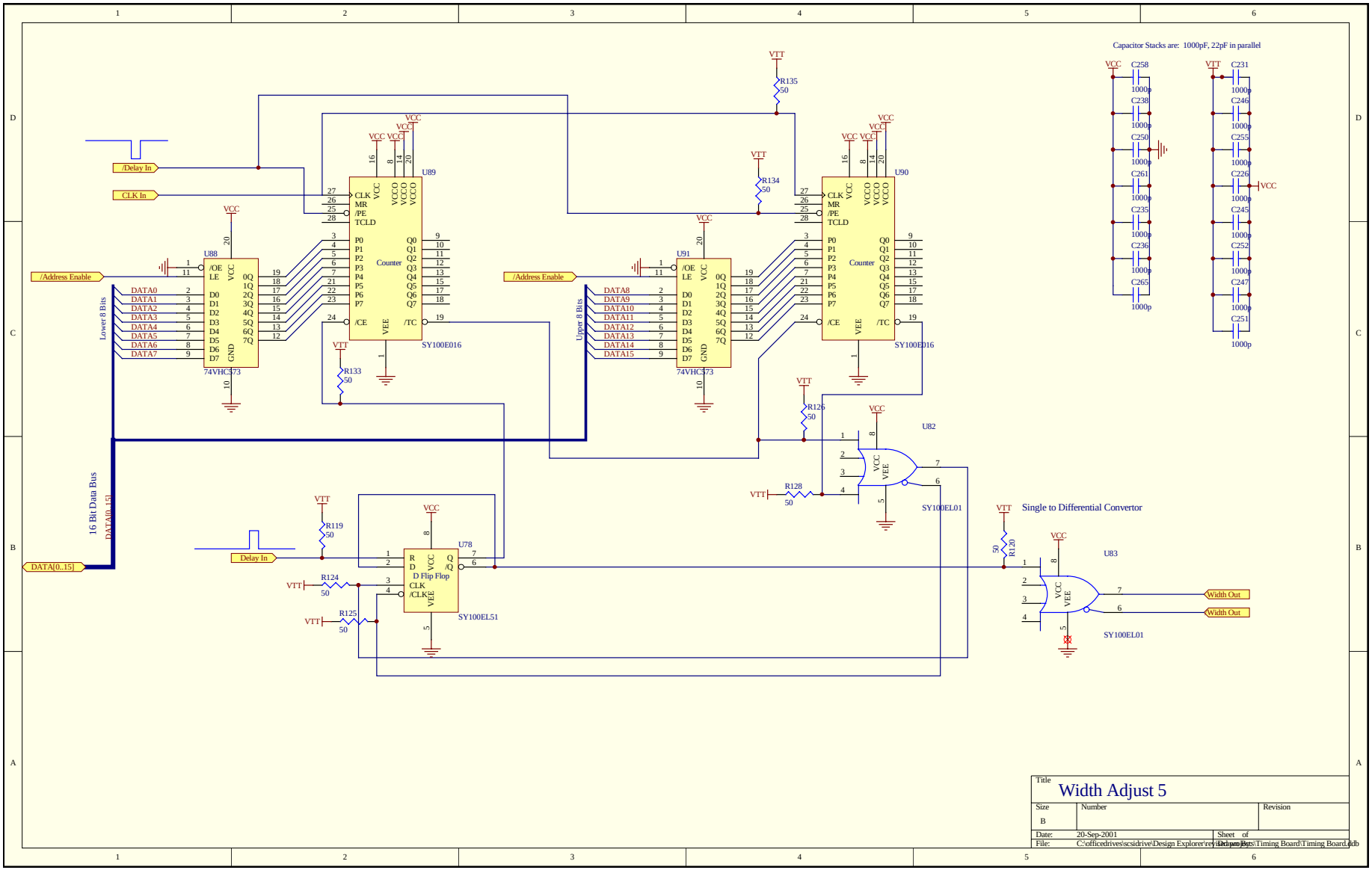

Capacitor Stacks are: 1000pF, 22pF in parallel
C80
C108
C83
C86
C82
C101
C100
C112
C92
C107
C87
C95
C96
C93
1000p
VCC
VTT
/PRF In
CLK In
CLK Out
/Address Enable
U32
U33
U34
U35
U41
U42
U45
74VHC573
SY100E016
SY100EL01
SY100EL51
Counter
Lower 8 Bits
Upper 8 Bits
DATA0
DATA1
DATA2
DATA3
DATA4
DATA5
DATA6
DATA7
DATA8
DATA9
DATA10
DATA11
DATA12
DATA13
DATA14
DATA15
16 Bit Data Bus
DATA[0..15]
R44
R45
R48
R51
R54
R55
R56
R60
50
PRF In
D Flip Flop
Delay Out
Single to Differential
Title
Delay Adjust 3
Size
B
Number
Revision
Date: 20-Sep-2001
Sheet of
File: C:\officedrives\scsidrive\Design Explorer\revisions\Delay Bits\Timing Board\Timing Board.ddb

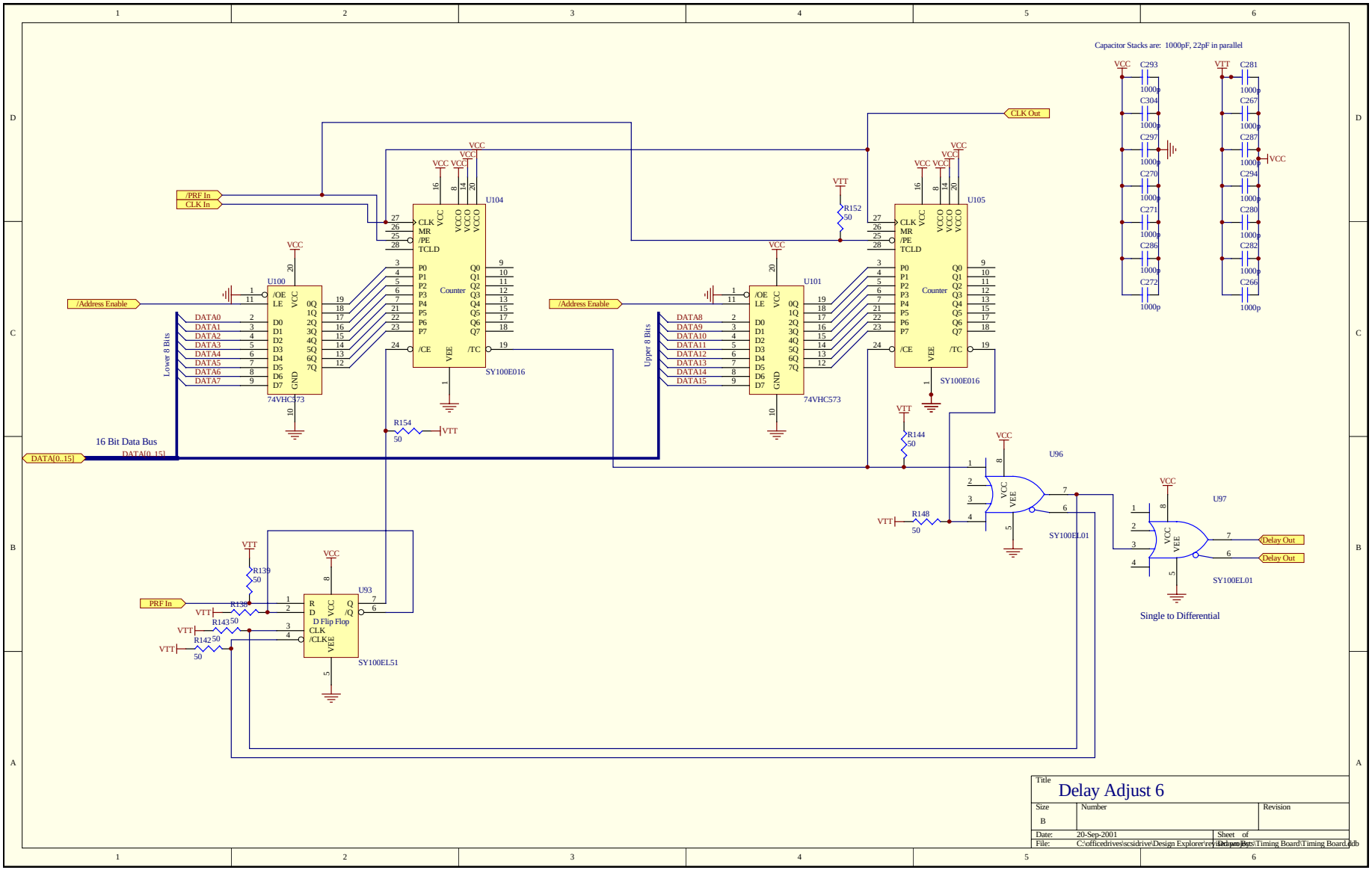

Capacitor Stacks are: 1000pF, 22pF in parallel
/Delay In
CLK In
/Address Enable
/Address Enable
Lower 8 Bits
Upper 8 Bits
16 Bit Data Bus
DATA[0..15]
DATA0
DATA1
DATA2
DATA3
DATA4
DATA5
DATA6
DATA7
DATA8
DATA9
DATA10
DATA11
DATA12
DATA13
DATA14
DATA15
U36
U37
U38
U39
74VHC373
SY100E016
Counter
R43
R47
R46
R59
R49
R57
R52
R58
U43
U44
U46
SY100EL01
SY100EL51
D Flip Flop
Single to Differential Convertor
Delay In
Width Out
Width Out
C103
C102
C110
C85
C88
C81
C84
C98
C113
C90
C99
C94
C97
C109
C89
1000p
VCC
VTT
Title
Width Adjust 3
Size
B
Number
Revision
Date: 20-Sep-2001
Sheet of
File: C:\officedrives\scsidrive\Design Explorer\revisions\Schematics\Timing Board\Timing Board.ddb

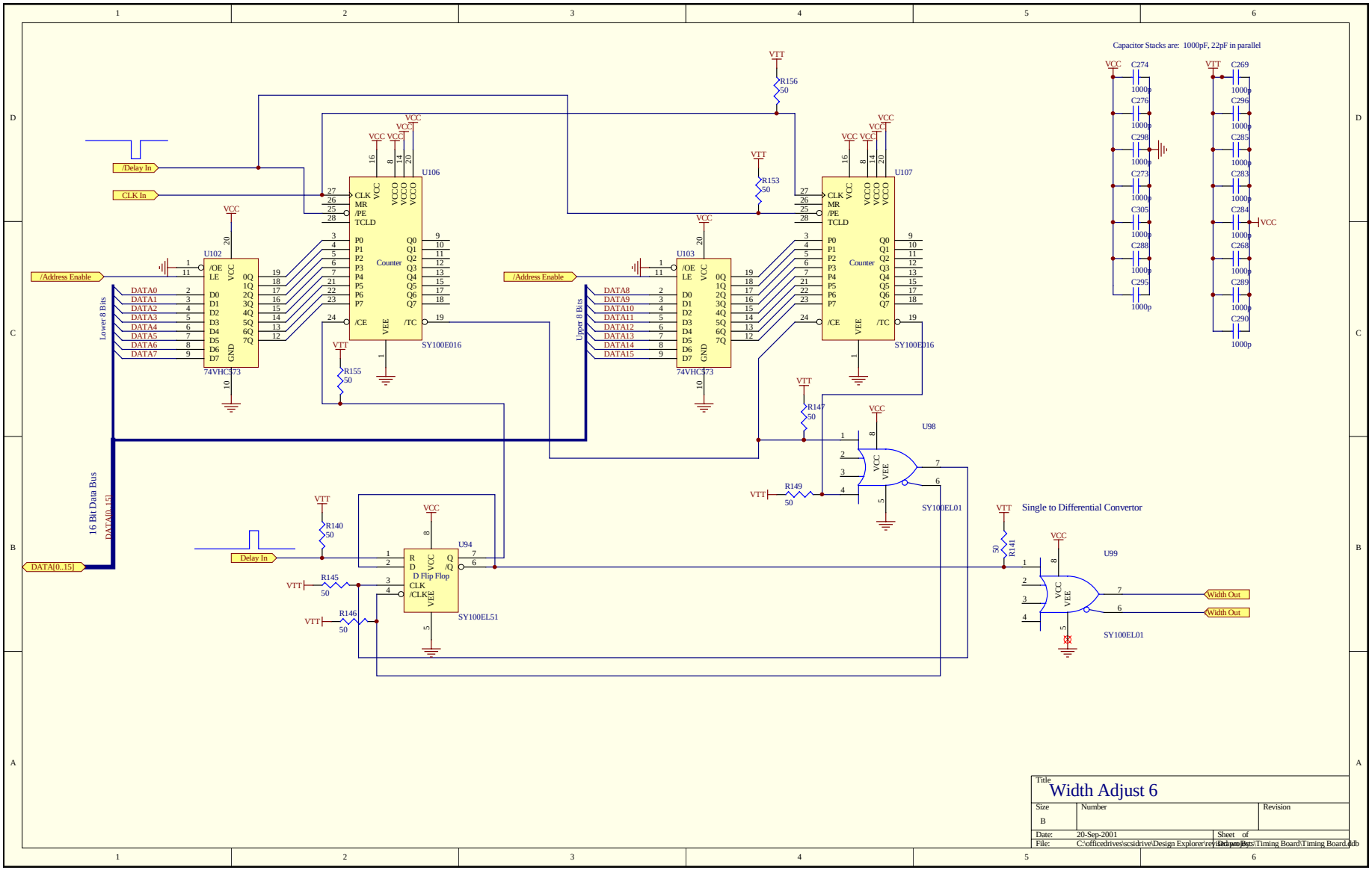

Capacitor Stacks are: 1000pF, 22pF in parallel
/PRF In
CLK In
/Address Enable
Lower 8 Bits
Upper 8 Bits
16 Bit Data Bus
DATA[0..15]
CLK Out
U47
U48
U49
U50
U56
U57
U61
74VHC573
SY100E016
SY100EL01
SY100EL51
Counter
D Flip Flop
Single to Differential
Delay Out
Delay Out
Title
Delay Adjust 4
Size
B
Number
Revision
Date: 20-Sep-2001
Sheet of
File: C:\officedrives\scsidrive\Design Explorer\revisions\Delay Bits\Timing Board\Timing Board.ddb

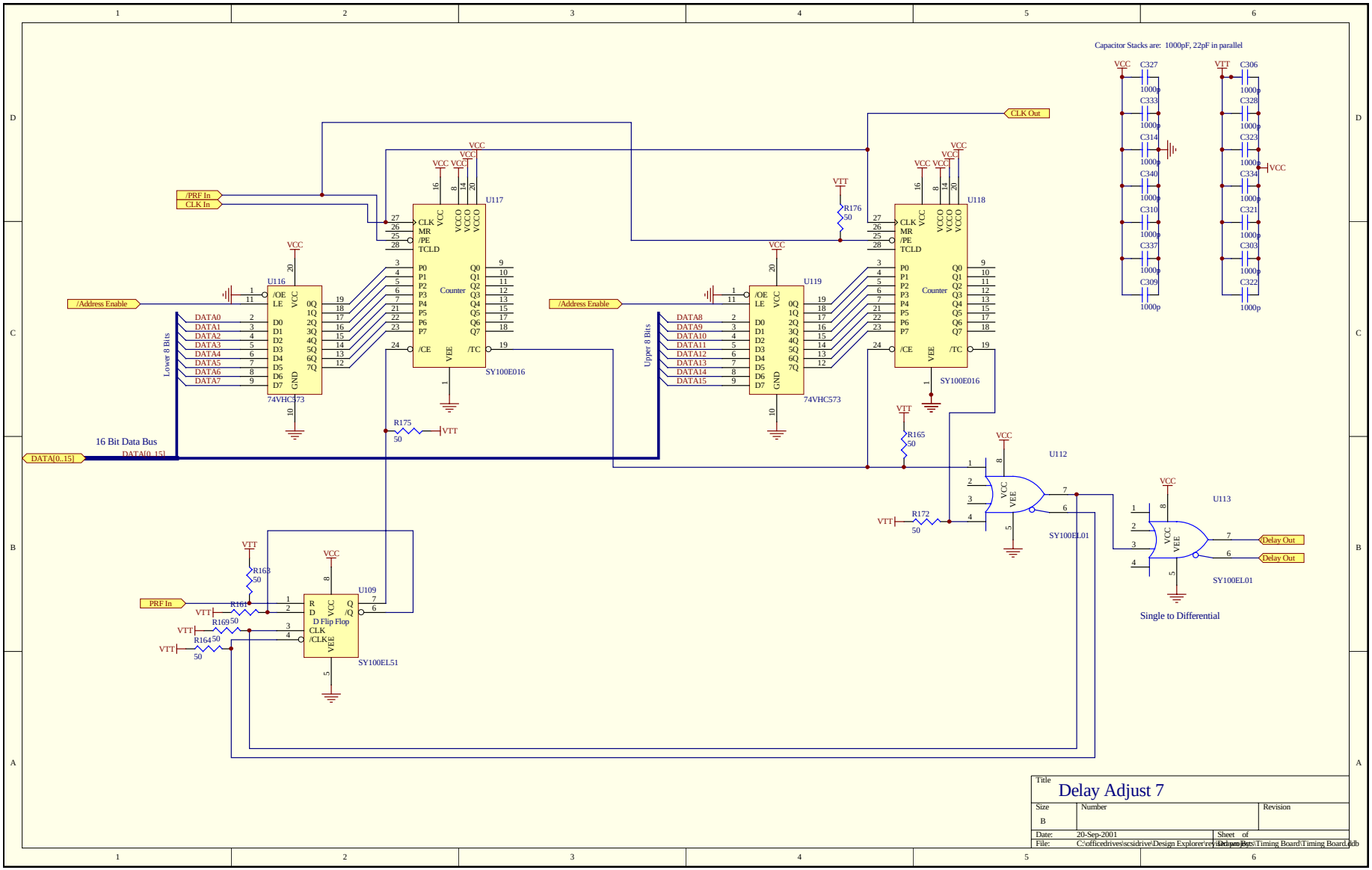

Capacitor Stacks are: 1000pF, 22pF in parallel
/Delay In
CLK In
/Address Enable
Lower 8 Bits
Upper 8 Bits
16 Bit Data Bus
DATA[0..15]
U51
U52
U53
U54
74VHC373
SY100E016
Counter
U58
SY100EL01
Single to Differential Convertor
U59
U62
D Flip Flop
SY100EL51
Width Out
Title
Width Adjust 4
Size
B
Number
Revision
Date: 20-Sep-2001
Sheet of
File: C:\officedrives\scsidrive\Design Explorer\revisions\Timing Board\Timing Board.ddb

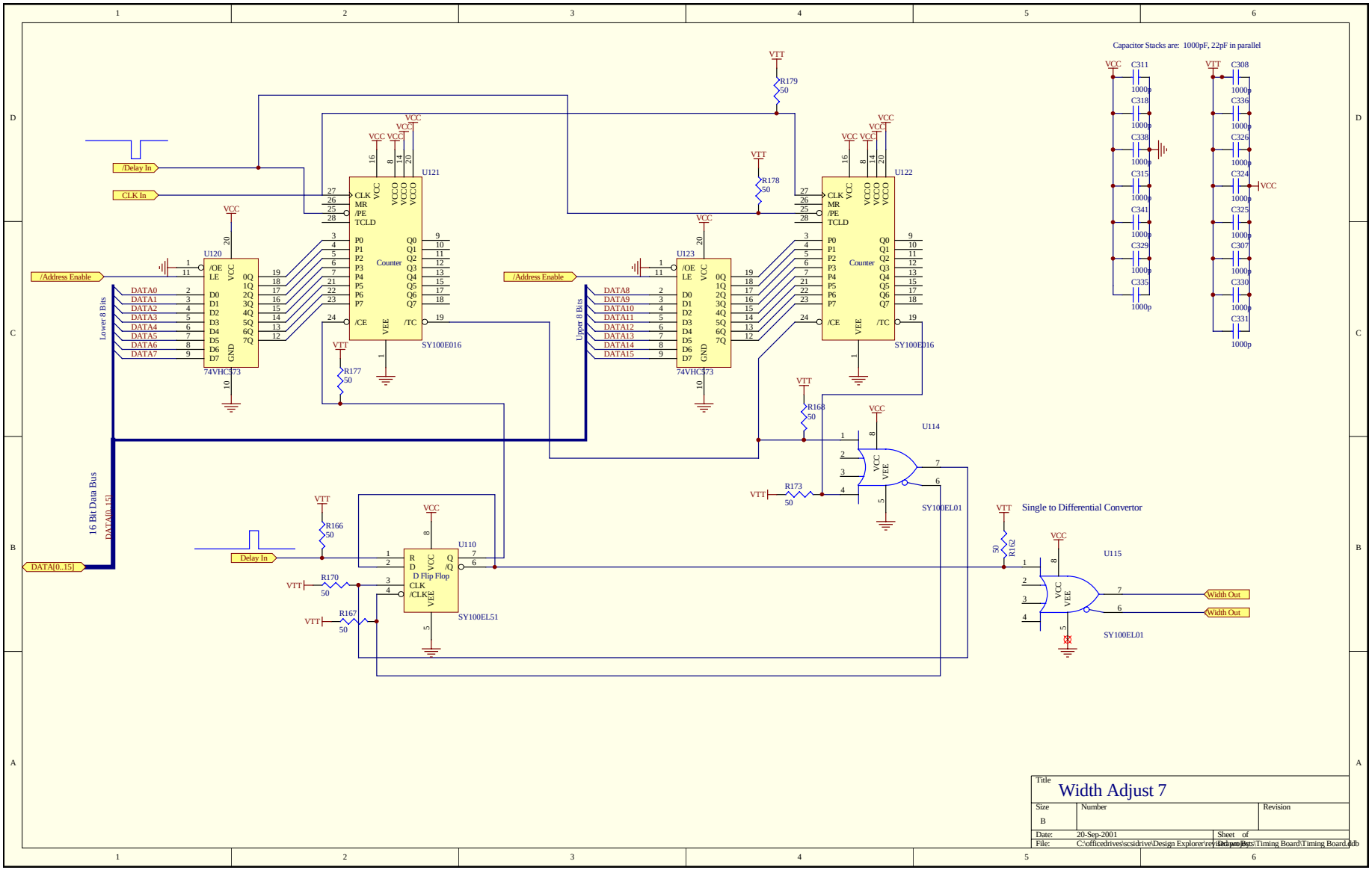

Capacitor Stacks are: 1000pF, 22pF in parallel
/PRF In
CLK In
CLK Out
/Address Enable
/Address Enable
Lower 8 Bits
Upper 8 Bits
16 Bit Data Bus
DATA[0..15]
DATA[0..15]
U84
U85
U86
U87
U80
U81
U77
74VHC573
74VHC573
SY100E016
SY100E016
SY100EL01
SY100EL01
SY100EL51
Counter
D Flip Flop
Single to Differential
Delay Out
Delay Out
R131
R132
R123
R127
R118
R117
R122
R121
Title
Delay Adjust 5
Size
B
Number
Revision
Date: 20-Sep-2001
Sheet of
File: C:\officedrives\scsidrive\Design Explorer\revisions\Delay Bits\Timing Board\Timing Board.ddb

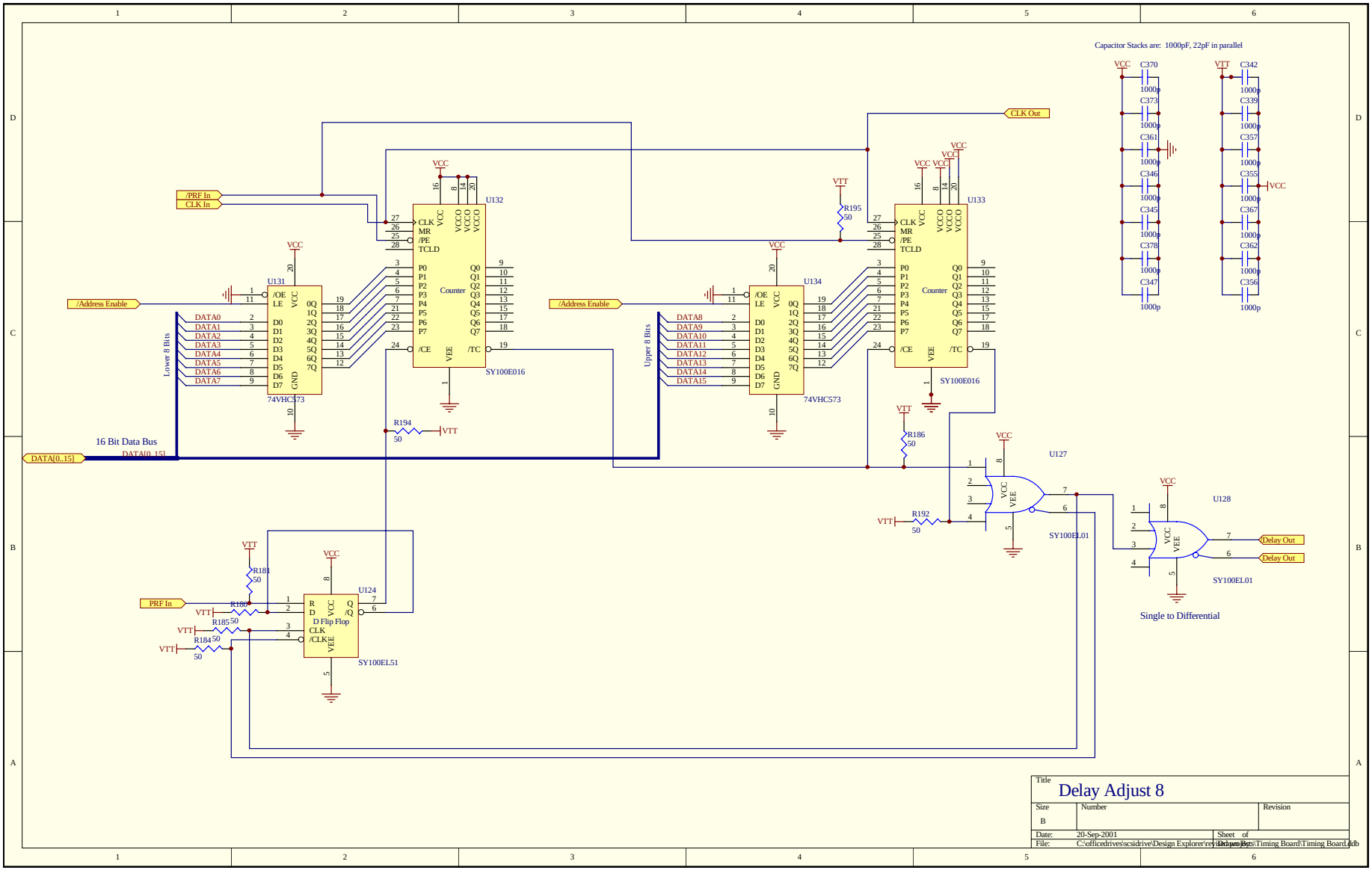

Capacitor Stacks are: 1000pF, 22pF in parallel
/Delay In
CLK In
/Address Enable
Lower 8 Bits
Upper 8 Bits
16 Bit Data Bus
DATA[0..15]
U88
U89
U90
U91
74VHC373
SY100E016
Counter
U82
SY100EL01
U78
D Flip Flop
SY100EL51
Delay In
Single to Differential Convertor
U83
Width Out
Width Out
VCC
VTT
R135 50
R134 50
R133 50
R126 50
R128 50
R119 50
R124 50
R125 50
R120 50
C258 1000p
C238 1000p
C250 1000p
C261 1000p
C235 1000p
C236 1000p
C265 1000p
C231 1000p
C246 1000p
C255 1000p
C226 1000p
C245 1000p
C252 1000p
C247 1000p
C251 1000p
Title
Width Adjust 5
Size
B
Number
Revision
Date: 20-Sep-2001
Sheet of
File: C:\officedrives\scsidrive\Design Explorer\revisions\Drawn Bits\Timing Board\Timing Board.ddb

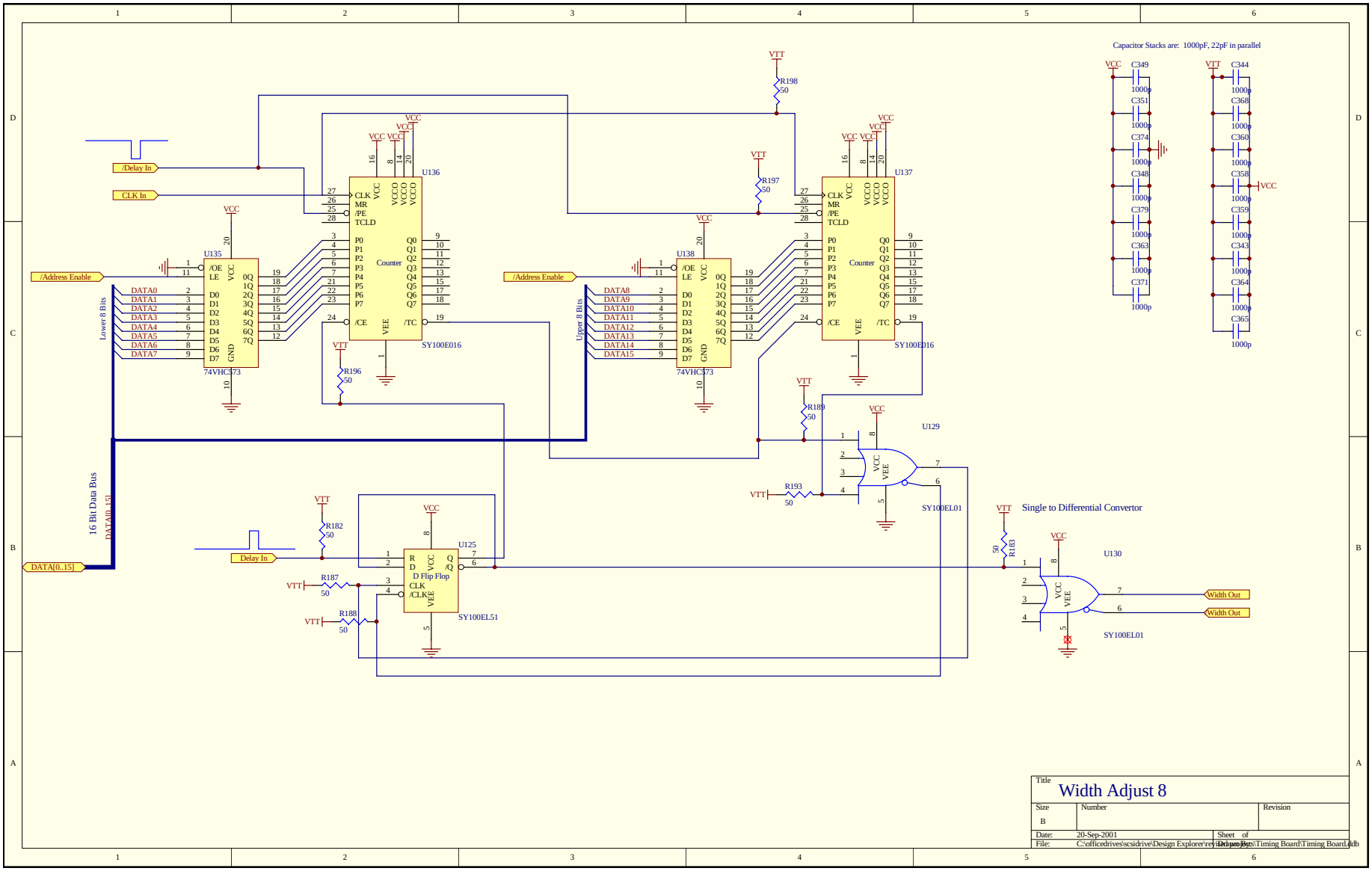

Capacitor Stacks are: 1000pF, 22pF in parallel
C293 1000p
C304 1000p
C297 1000p
C270 1000p
C271 1000p
C286 1000p
C272 1000p
C281 1000p
C267 1000p
C287 1000p
C294 1000p
C280 1000p
C282 1000p
C266 1000p
VCC
VTT
CLK Out
/PRF In
CLK In
U104
U105
U100
U101
SY100E016
74VHC573
/Address Enable
Lower 8 Bits
Upper 8 Bits
DATA0
DATA1
DATA2
DATA3
DATA4
DATA5
DATA6
DATA7
DATA8
DATA9
DATA10
DATA11
DATA12
DATA13
DATA14
DATA15
R152 50
R154 50
R144 50
R148 50
R139 50
R138 50
R143 50
R142 50
16 Bit Data Bus
DATA[0..15]
U96
SY100EL01
U97
Delay Out
Single to Differential
U93
D Flip Flop
SY100EL51
Title
Delay Adjust 6
Size
B
Number
Revision
Date: 20-Sep-2001
Sheet of
File: C:\officedrives\scsidrive\Design Explorer\revisions\Delay Bits\Timing Board\Timing Board.ddb

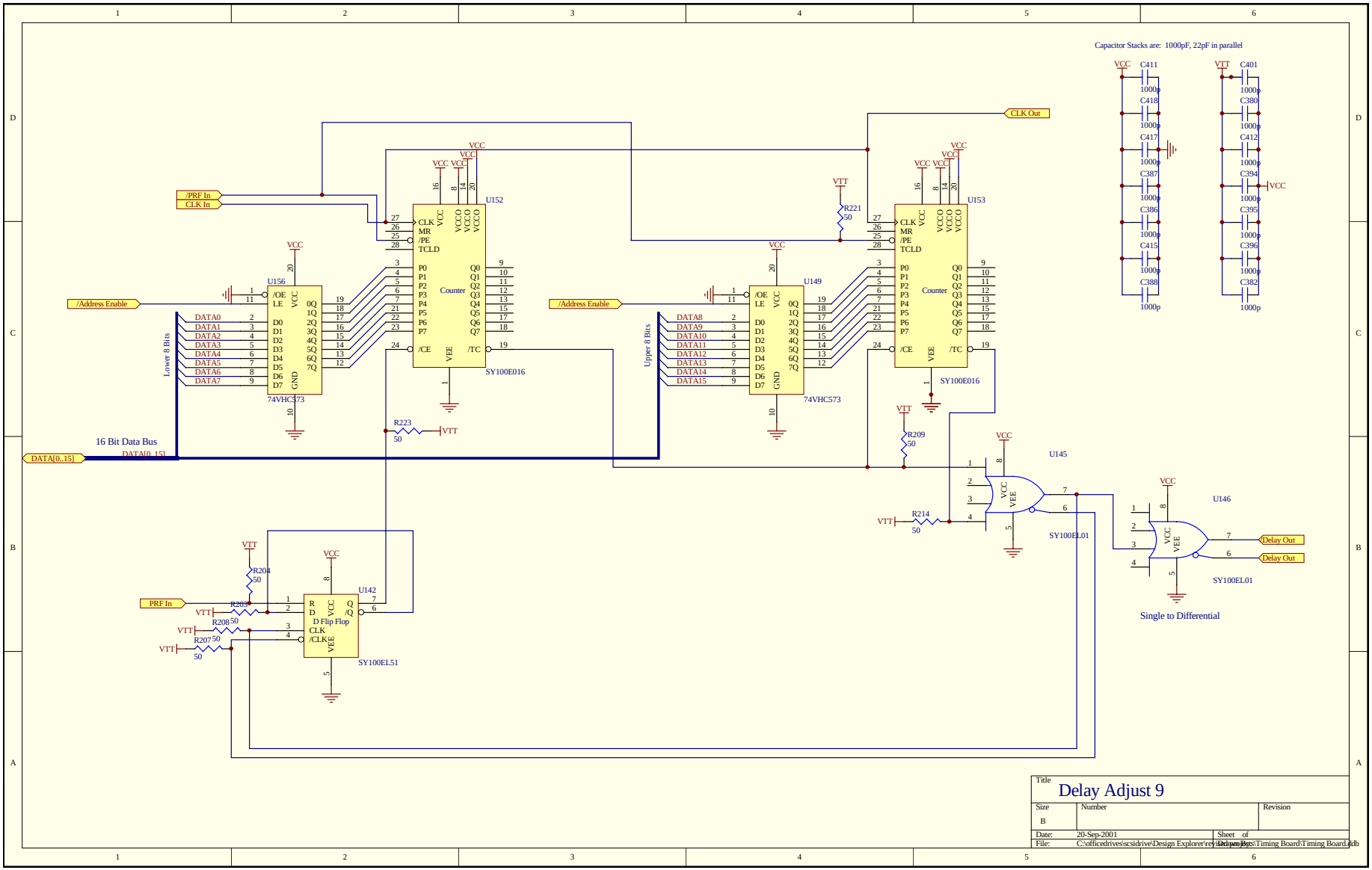

Capacitor Stacks are: 1000pF, 22pF in parallel
/Delay In
CLK In
/Address Enable
Lower 8 Bits
Upper 8 Bits
16 Bit Data Bus
DATA[0..15]
U102
U103
U106
U107
74VHC573
SY100E016
Counter
U94
D Flip Flop
SY100EL51
U98
U99
SY100EL01
Single to Differential Convertor
Width Out
Width Out
Title
Width Adjust 6
Size
B
Number
Revision
Date: 20-Sep-2001
Sheet of
File: C:\officedrives\scsidrive\Design Explorer\revisions\Timing Boards\Timing Board\Timing Board.ddb

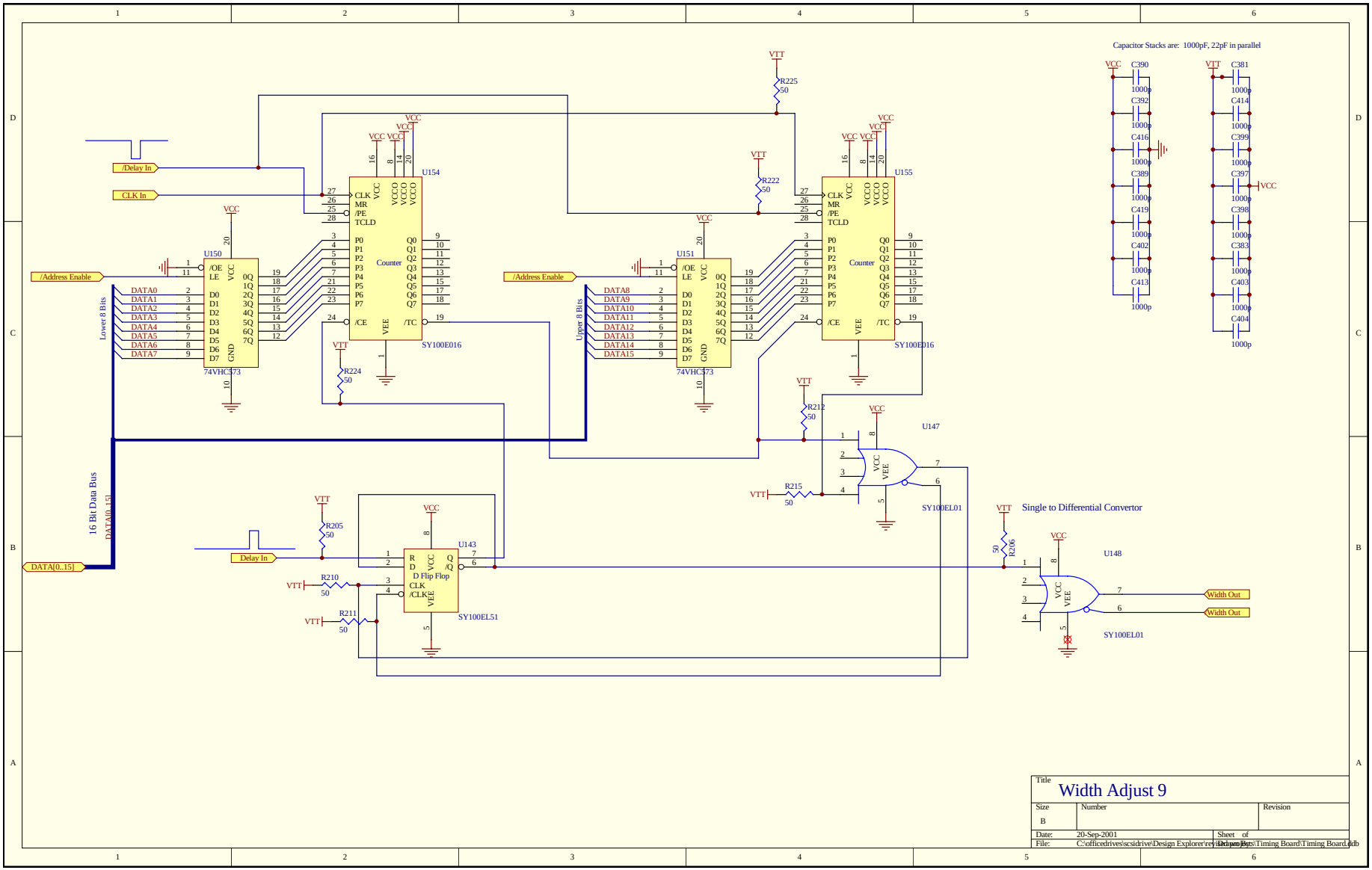

Capacitor Stacks are: 1000pF, 22pF in parallel
/PRF In
CLK In
CLK Out
/Address Enable
Lower 8 Bits
Upper 8 Bits
16 Bit Data Bus
DATA[0..15]
DATA[0..15]
U116
U117
U118
U119
74VHC573
SY100E016
SY100EL51
SY100EL01
U109
U112
U113
R161
R163
R164
R165
R169
R172
R175
R176
Delay Out
Single to Differential
Title
Delay Adjust 7
Size
B
Number
Revision
Date: 20-Sep-2001
Sheet of
File: C:\officedrives\scsidrive\Design Explorer\revisions\Delay Adjust\Timing Board\Timing Board.ddb

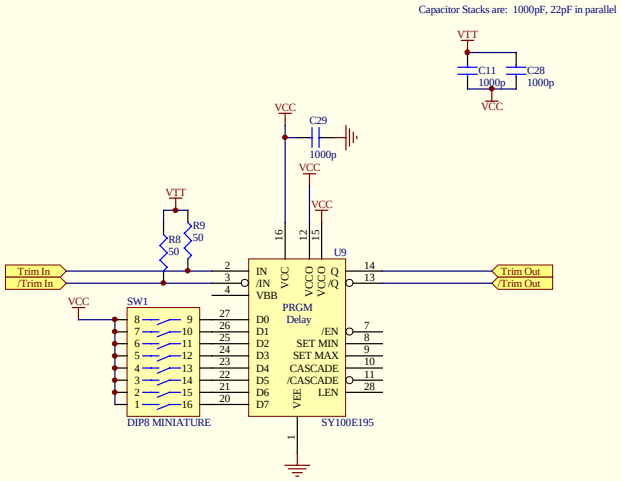

Capacitor Stacks are: 1000pF, 22pF in parallel
/Delay In
CLK In
/Address Enable
Lower 8 Bits
Upper 8 Bits
16 Bit Data Bus
DATA[0..15]
U120
U121
U122
U123
74VHC373
SY100E016
Counter
U110
D Flip Flop
SY100EL51
U114
U115
SY100EL01
Single to Differential Convertor
Delay In
Width Out
Width Out
VTT
VCC
R179 50
R178 50
R177 50
R168 50
R173 50
R166 50
R170 50
R167 50
R162 50
C311 1000p
C318 1000p
C338 1000p
C315 1000p
C341 1000p
C329 1000p
C335 1000p
C308 1000p
C336 1000p
C326 1000p
C324 1000p
C325 1000p
C307 1000p
C330 1000p
C331 1000p
Title
Width Adjust 7
Size
B
Number
Revision
Date: 20-Sep-2001
Sheet of
File: C:\officedrives\scsidrive\Design Explorer\revisions\Drawings\Timing Board\Timing Board.ddb


Capacitor Stacks are: 1000pF, 22pF in parallel
/PRF In
CLK In
/Address Enable
DATA[0..15]
16 Bit Data Bus
Lower 8 Bits
Upper 8 Bits
CLK Out
Delay Out
Delay Out
Single to Differential
U131
U132
U133
U134
U124
U127
U128
74VHC573
SY100E016
SY100EL51
SY100EL01
Title
Delay Adjust 8
Size
B
Number
Revision
Date: 20-Sep-2001
Sheet of
File: C:\officedrives\scsidrive\Design Explorer\revisions\Delays\Timing Board\Timing Board.ddb

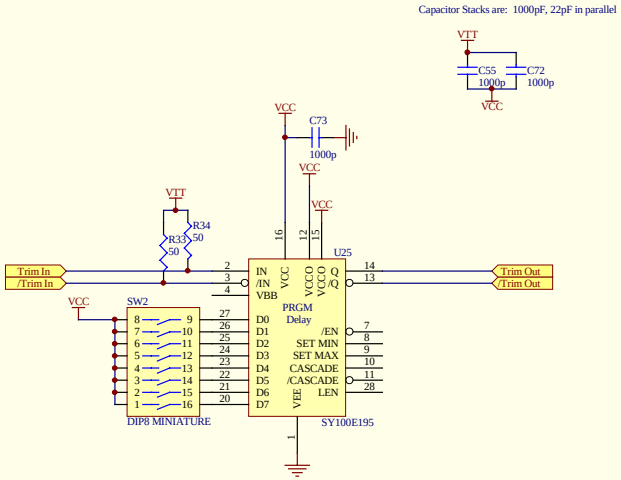

Capacitor Stacks are: 1000pF, 22pF in parallel
/Delay In
CLK In
/Address Enable
Lower 8 Bits
Upper 8 Bits
16 Bit Data Bus
DATA[0..15]
U135
U136
U137
U138
74VHC373
SY100E016
Counter
U125
D Flip Flop
SY100EL51
U129
SY100EL01
Single to Differential Convertor
U130
Width Out
Width Out
Title
Width Adjust 8
Size
B
Number
Revision
Date: 20-Sep-2001
Sheet of
File: C:\officedrives\scsidrive\Design Explorer\revisions\Schematics\Timing Board\Timing Board.ddb


Capacitor Stacks are: 1000pF, 22pF in parallel
C411 1000p
C418 1000p
C417 1000p
C387 1000p
C386 1000p
C415 1000p
C388 1000p
C401 1000p
C380 1000p
C412 1000p
C394 1000p
C395 1000p
C396 1000p
C382 1000p
/PRF In
CLK In
CLK Out
U152
U153
SY100E016
Counter
R221 50
R223 50
R209 50
R214 50
R204 50
R203 50
R208 50
R207 50
/Address Enable
U156
U149
74VHC573
Lower 8 Bits
Upper 8 Bits
DATA0
DATA1
DATA2
DATA3
DATA4
DATA5
DATA6
DATA7
DATA8
DATA9
DATA10
DATA11
DATA12
DATA13
DATA14
DATA15
16 Bit Data Bus
DATA[0..15]
DATA[0..15]
U145
SY100EL01
U146
Delay Out
Delay Out
Single to Differential
U142
D Flip Flop
SY100EL51
PRF In
VCC
VTT
VEE
Title
Delay Adjust 9
Size
B
Number
Revision
Date: 20-Sep-2001
Sheet of
File: C:\officedrives\scsidrive\Design Explorer\revisions\Delay Adjust\Timing Board\Timing Board.ddb

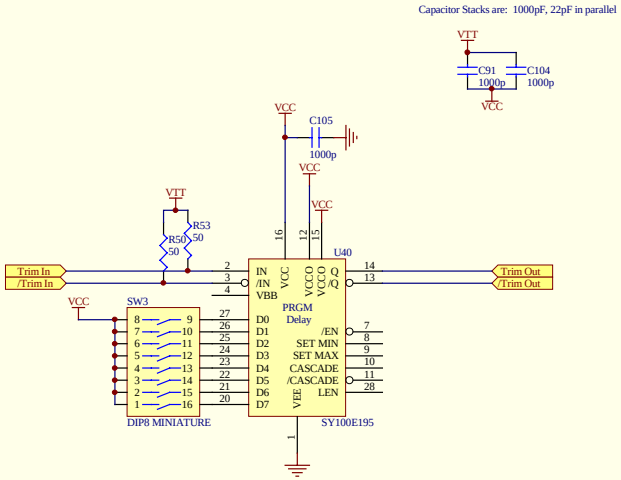

Capacitor Stacks are: 1000pF, 22pF in parallel
/Delay In
CLK In
/Address Enable
/Address Enable
Lower 8 Bits
Upper 8 Bits
16 Bit Data Bus
DATA[0..15]
DATA0
DATA1
DATA2
DATA3
DATA4
DATA5
DATA6
DATA7
DATA8
DATA9
DATA10
DATA11
DATA12
DATA13
DATA14
DATA15
U150
U151
U154
U155
74VHC373
SY100E016
Counter
R225
R222
R224
R212
R215
R205
R210
R211
U147
SY100EL01
U143
D Flip Flop
SY100EL51
Delay In
Single to Differential Convertor
R206
U148
Width Out
Width Out
Title
Width Adjust 9
Size
B
Number
Revision
Date: 20-Sep-2001
Sheet of
File: C:\officedrives\scsidrive\Design Explorer\revisions\Timing Boards\Timing Board\Timing Board.ddb



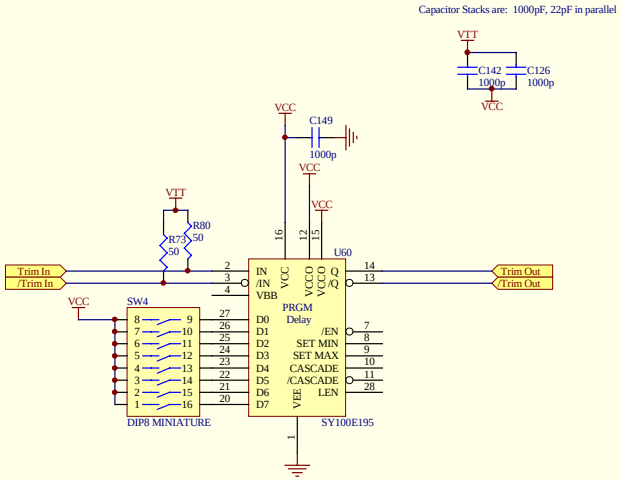
Title	Trim Delay		
Size: B	Number:		Revision:
Date: 20-Sep-2001		Sheet of	
File: C:\officedrives\scsidrive\Design Explorer\revisions\Schematics\Timing Board\Timing Board.ddb			



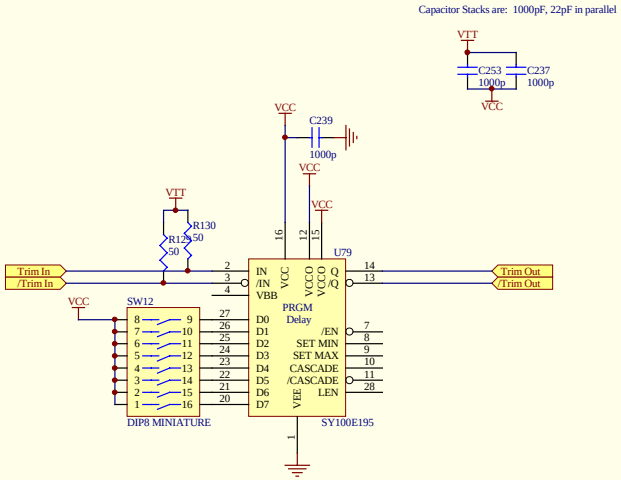
Title	Trim Delay 2		
Size: B	Number:		Revision:
Date:	20-Sep-2001	Sheet	of
File:	C:\officedrives\scsidrive\Design Explorer\revisions\Schematics\Timing Board\Timing Board.ddb		



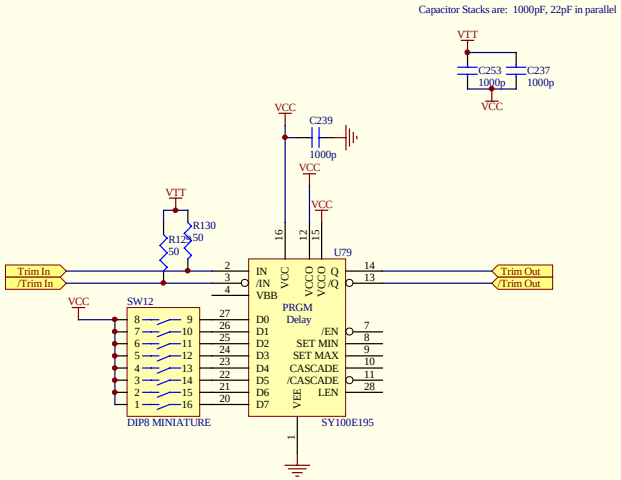
Title	Trim Delay 3		
Size: B	Number:		Revision:
Date: 20-Sep-2001		Sheet of	
File: C:\officedrives\scsidrive\Design Explorer\revisions\Schematics\Timing Board\Timing Board.ddb			



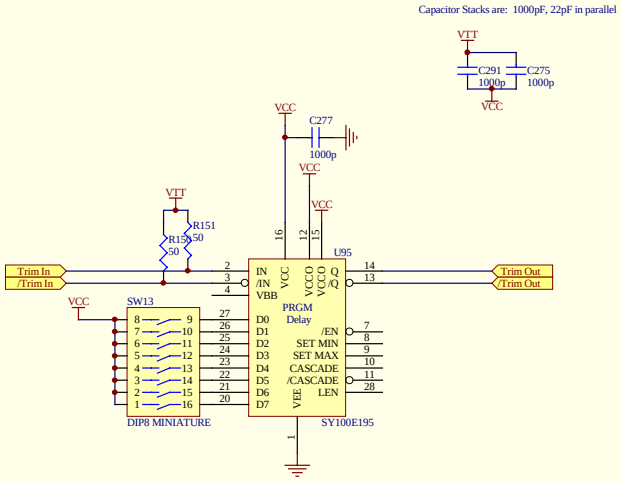
Title	Trim Delay 4		
Size: B	Number:		Revision:
Date: 20-Sep-2001		Sheet of	
File: C:\officedrives\scsidrive\Design Explorer\revisions\Timing Board\Timing Board.ddb			



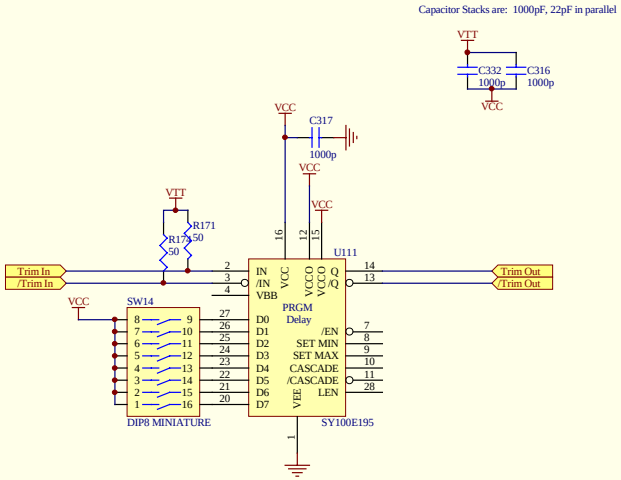
Title	Trim Delay 5		
Size: B	Number:		Revision:
Date: 20-Sep-2001		Sheet of	
File: C:\officedrives\scsidrive\Design Explorer\revisions\Documents\Timing Board\Timing Board.ddb			



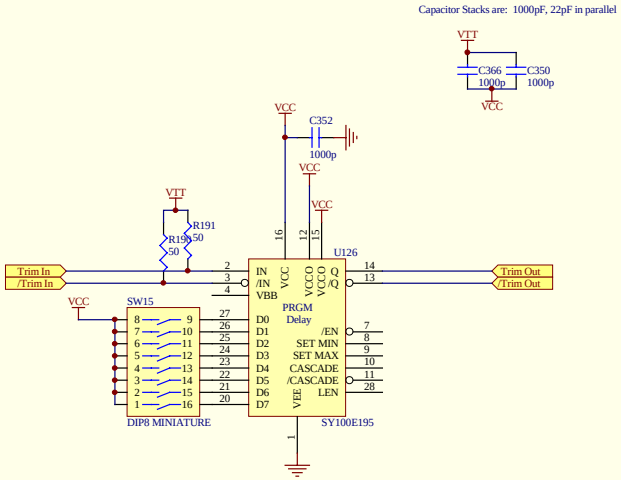
Title	Trim Delay 5		
Size: B	Number:		Revision:
Date: 20-Sep-2001		Sheet of	
File: C:\officedrives\scsidrive\Design Explorer\revisions\Documents\Timing Board\Timing Board.ddb			



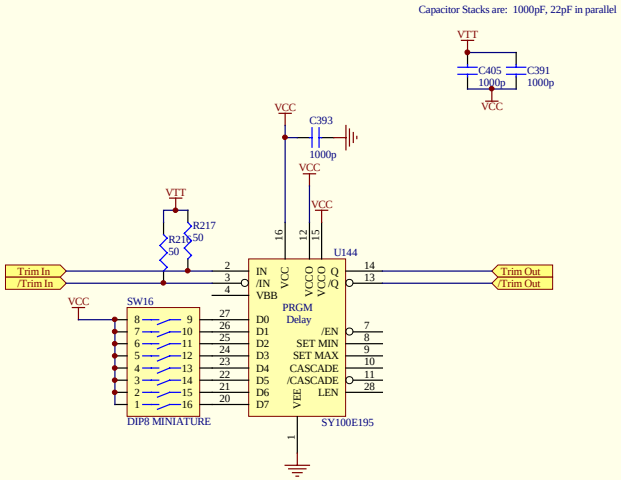
Title	Trim Delay 6		
Size: B	Number:		Revision:
Date:	20-Sep-2001	Sheet	of
File:	C:\officedrives\scsidrive\Design Explorer\revisions\Schematics\Timing Board\Timing Board.ddb		



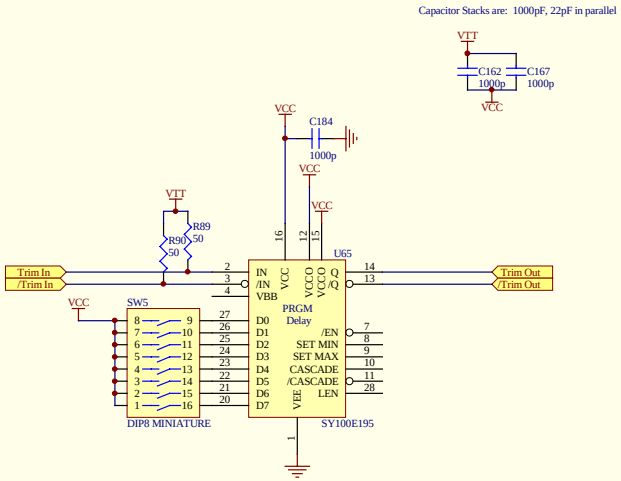
Title: Trim Delay 7		
Size: B	Number:	Revision:
Date: 20-Sep-2001	Sheet of	
File: C:\officedrives\scsidrive\Design Explorer\revisions\Timing Board\Timing Board.ddb		



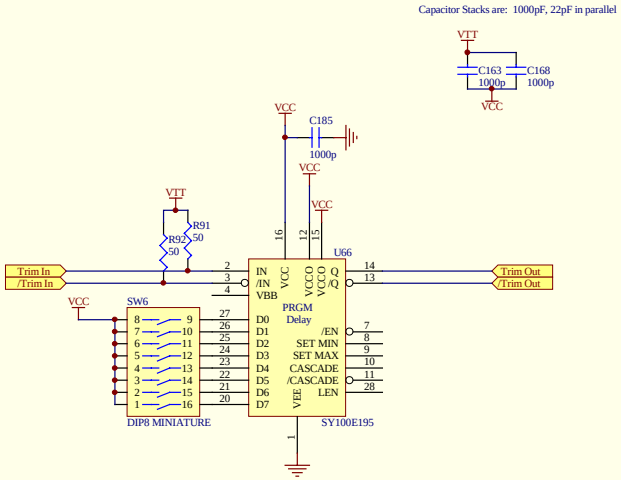
Title	Trim Delay 8		
Size: B	Number:		Revision:
Date:	20-Sep-2001	Sheet	of
File:	C:\officedrives\scsidrive\Design Explorer\revisions\Timing Board\Timing Board.ddb		



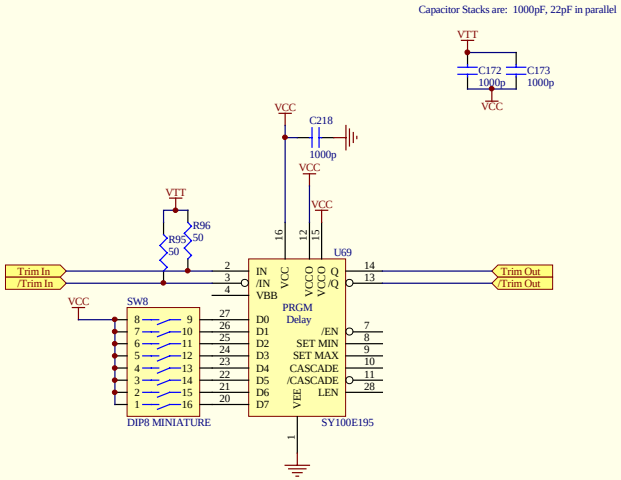
Title	Trim Delay 9	
Size: B	Number:	Revision:
Date: 20-Sep-2001	Sheet of	
File: C:\officedrives\scsidrive\Design Explorer\revisions\Schematics\Timing Board\Timing Board.ddb		



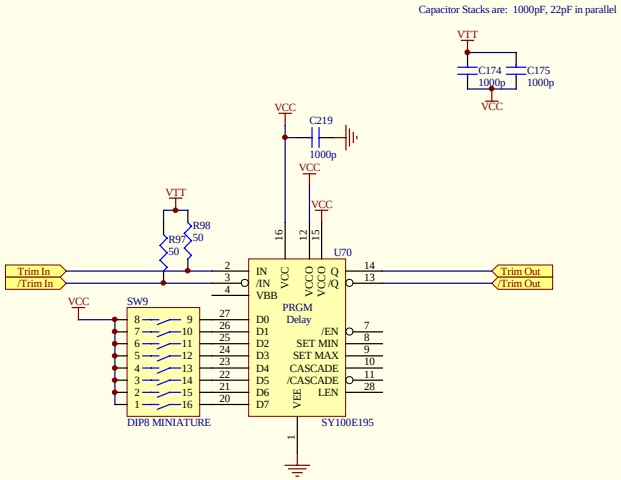
Title	Trim Delay 10		
Size: B	Number:		Revision:
Date: 20-Sep-2001		Sheet of	
File: C:\officedrives\scsidrive\Design Explorer\revisions\Timing Boards\Timing Board.ddb			



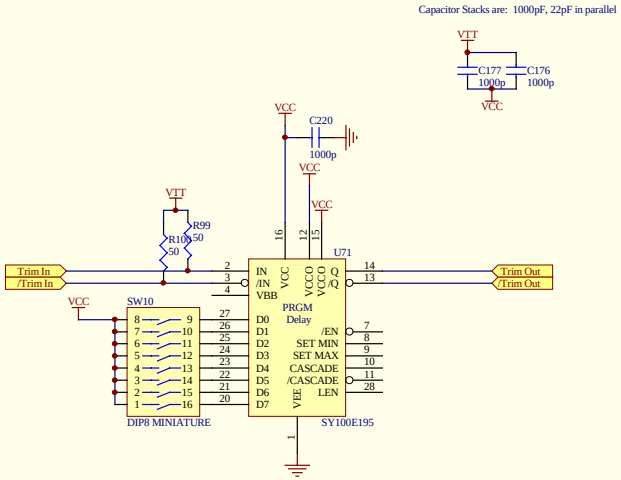
Title	Trim Delay 11		
Size: B	Number:		Revision:
Date: 20-Sep-2001		Sheet of	
File: C:\officedrives\scsidrive\Design Explorer\revisions\Schematics\Timing Board\Timing Board.ddb			



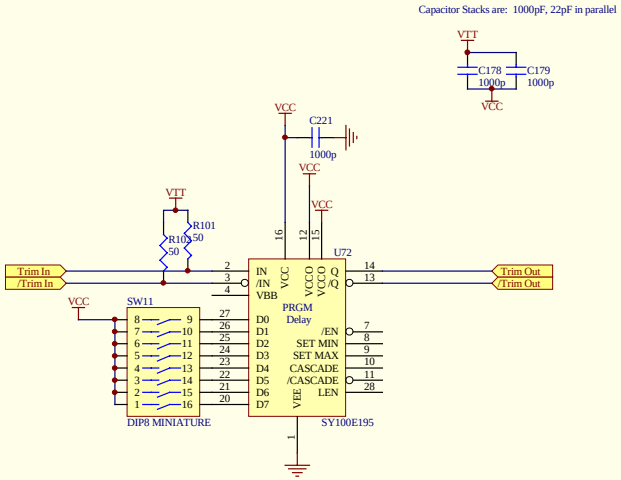
Title	Trim Delay 12		
Size: B	Number:		Revision:
Date: 20-Sep-2001		Sheet of	
File: C:\officedrives\scsidrive\Design Explorer\revisions\Documents\Timing Board\Timing Board.ddb			



Title	Trim Delay 13		
Size: B	Number:		Revision:
Date: 20-Sep-2001		Sheet of	
File: C:\officedrives\scsidrive\Design Explorer\revisions\Schematics\Timing Board\Timing Board.ddb			

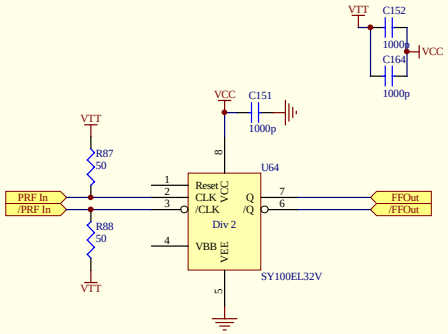


Title	Trim Delay 14	
Size: B	Number:	Revision:
Date: 20-Sep-2001	Sheet of	
File: C:\officedrives\scsidrive\Design Explorer\revisions\Schematics\Timing Board\Timing Board.ddb		

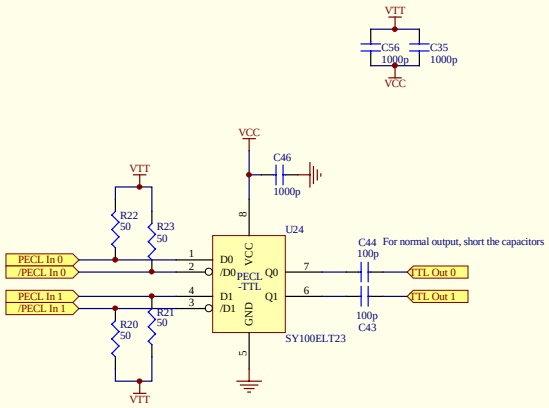


Title	Trim Delay 15		
Size: B	Number:		Revision:
Date: 20-Sep-2001		Sheet of	
File: C:\officedrives\scsidrive\Design Explorer\revisions\Schematics\Timing Board\Timing Board.ddb			

Capacitor Stacks are: 1000pF, 22pF in parallel

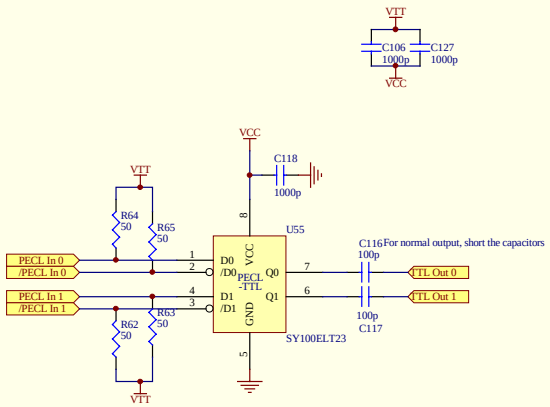


Title: Flip Flop		
Size: B	Number:	Revision:
Date: 20-Sep-2001	Sheet of	
File: C:\officedrives\scsidrive\Design Explorer\revisions\Schematics\Timing Board\Timing Board.ddb		



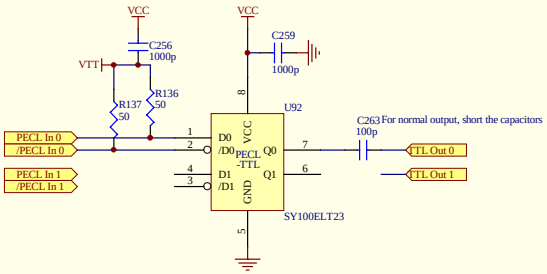
Title: PECL to TTL		
Size: B	Number:	Revision:
Date: 20-Sep-2001	Sheet of	
File: C:\officedrives\scsidrive\Design Explorer\revisions\Timing Board\Timing Board.ddb		

Capacitor Stacks are: 1000pF, 22pF in parallel



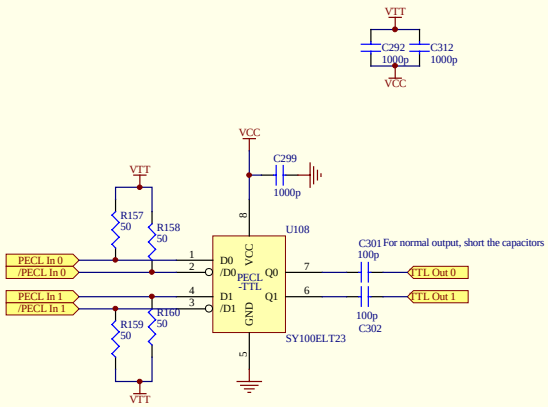
Title	PECL to TTL 2		
Size: B	Number:		Revision:
Date: 20-Sep-2001		Sheet of	
File: C:\officedrives\scsidrive\Design Explorer\revisions\Schematics\Timing Board\Timing Board.ddb			

Capacitor Stacks are: 1000pF, 22pF in parallel



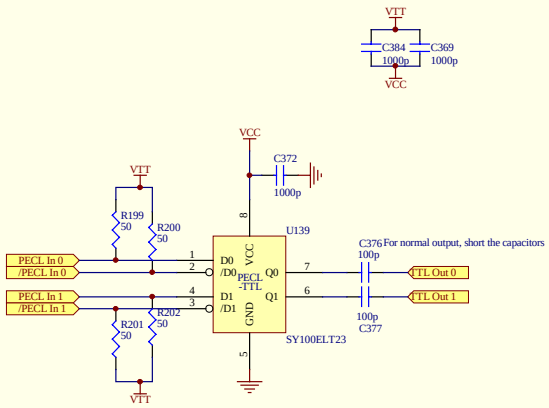
Title: PECL to TTL 3		
Size: B	Number:	Revision:
Date: 20-Sep-2001	Sheet of	
File: C:\officedrives\scsidrive\Design Explorer\revisions\Schematics\Timing Board\Timing Board.ddb		

Capacitor Stacks are: 1000pF, 22pF in parallel



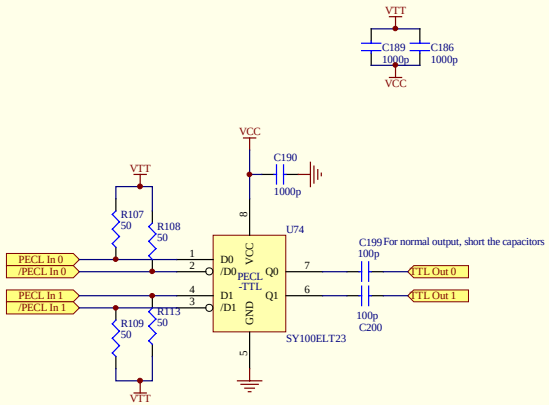
Title	PECL to TTL 4		
Size: B	Number:		Revision:
Date: 20-Sep-2001		Sheet of	
File: C:\officedrives\scsidrive\Design Explorer\revisions\Documents\Timing Board\Timing Board.ddb			

Capacitor Stacks are: 1000pF, 22pF in parallel



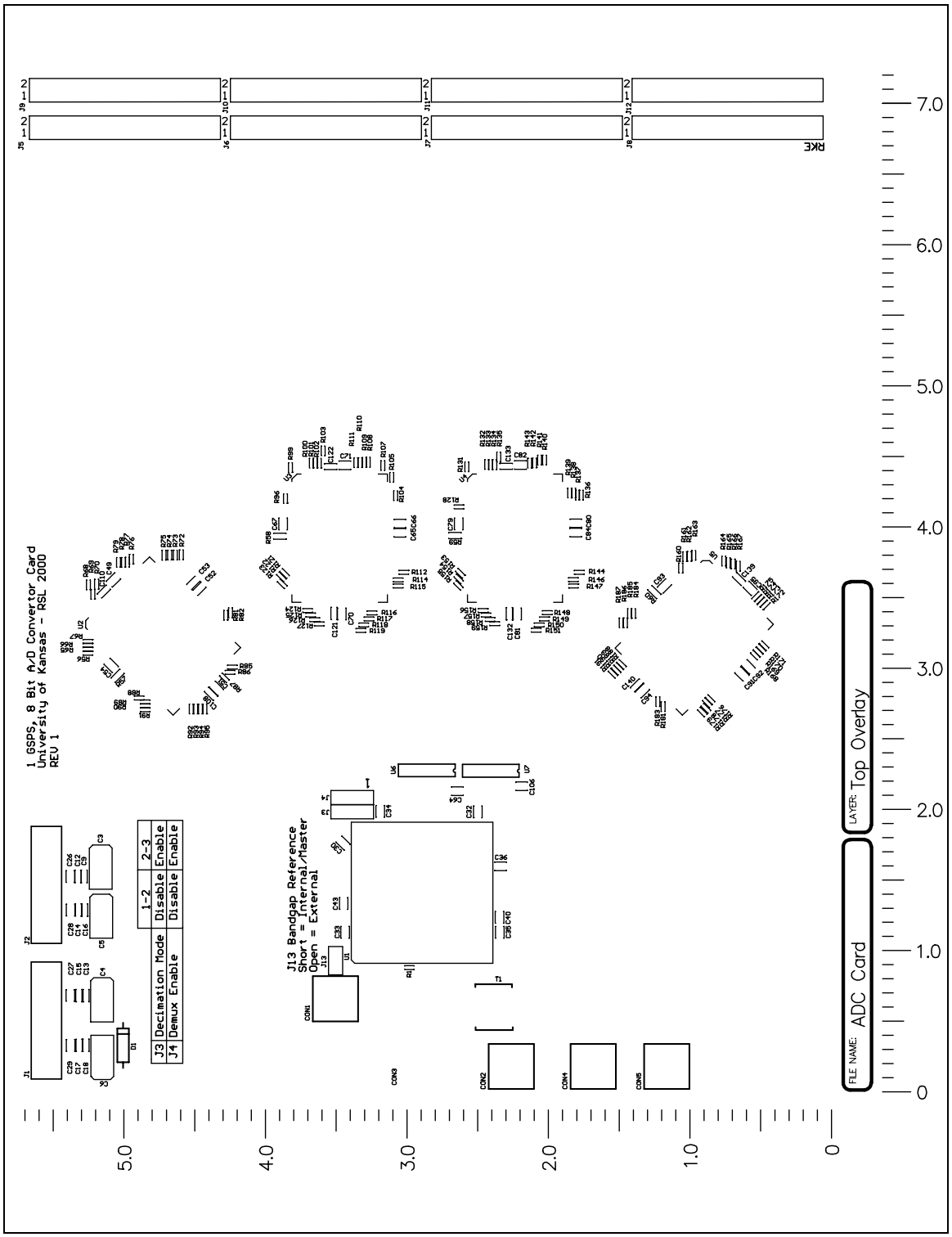
Title	PECL to TTL 5		
Size: B	Number:		Revision:
Date: 20-Sep-2001		Sheet of	
File: C:\officedrives\scsidrive\Design Explorer\revisions\Schematics\Timing Board\Timing Board.ddb			

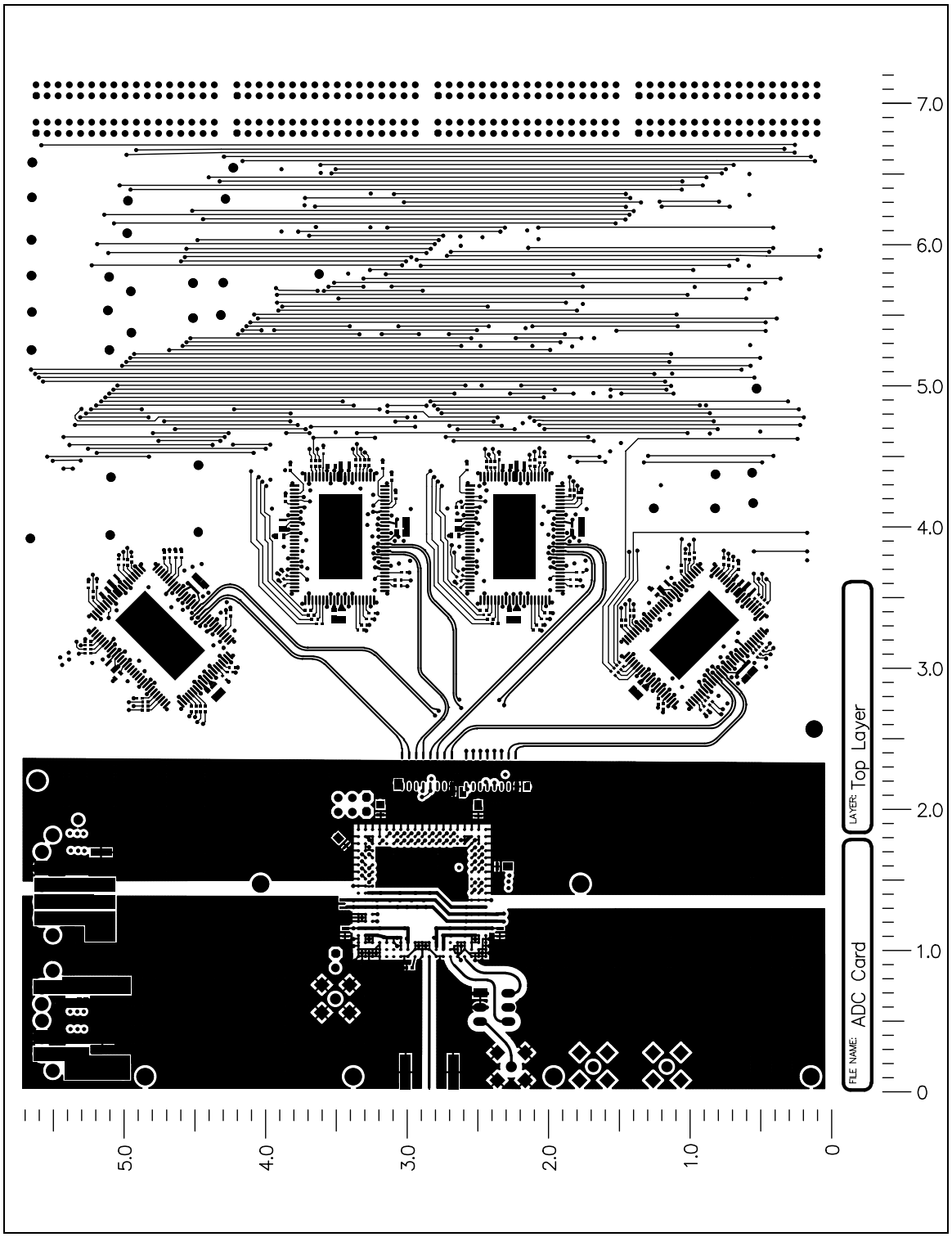
Capacitor Stacks are: 1000pF, 22pF in parallel

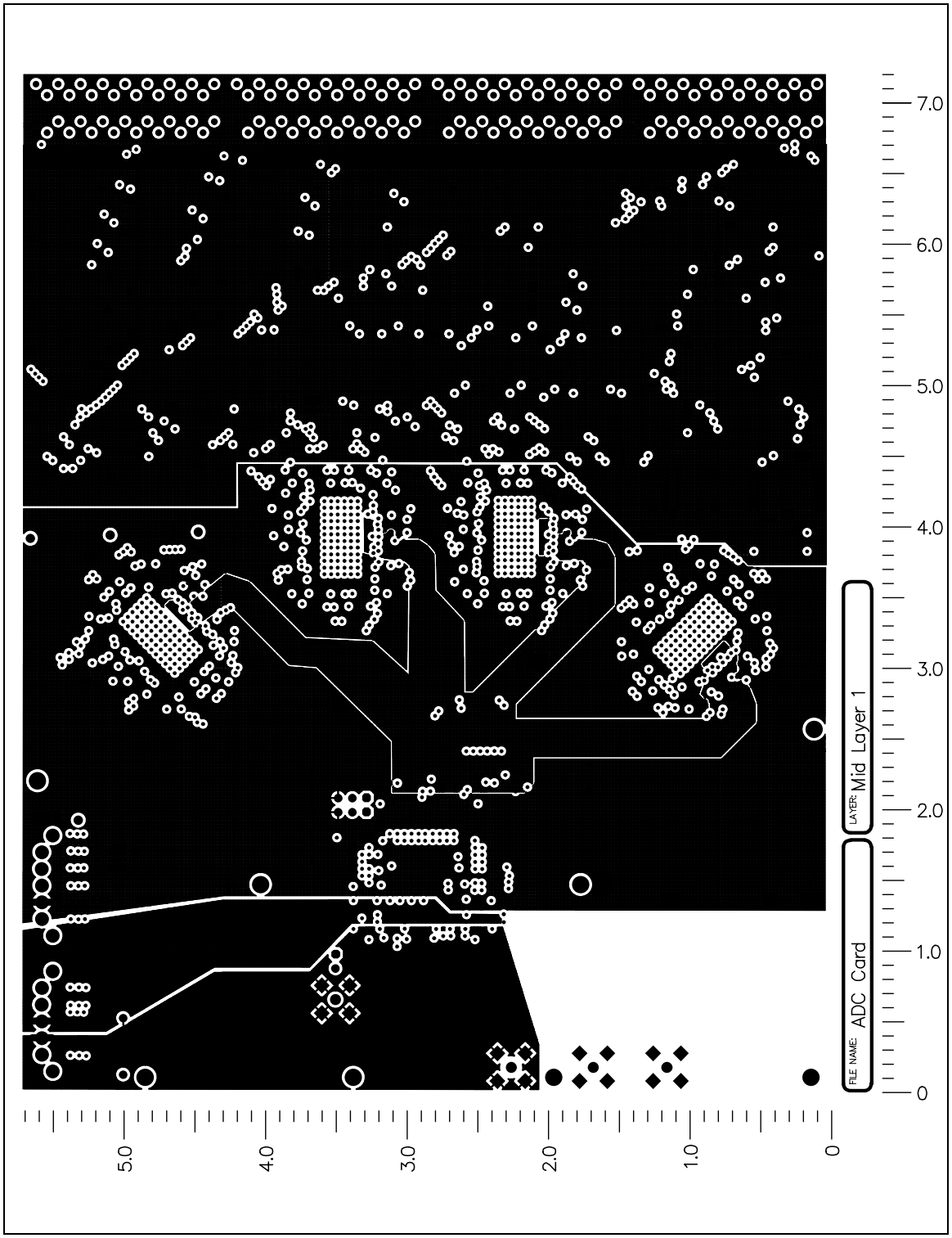


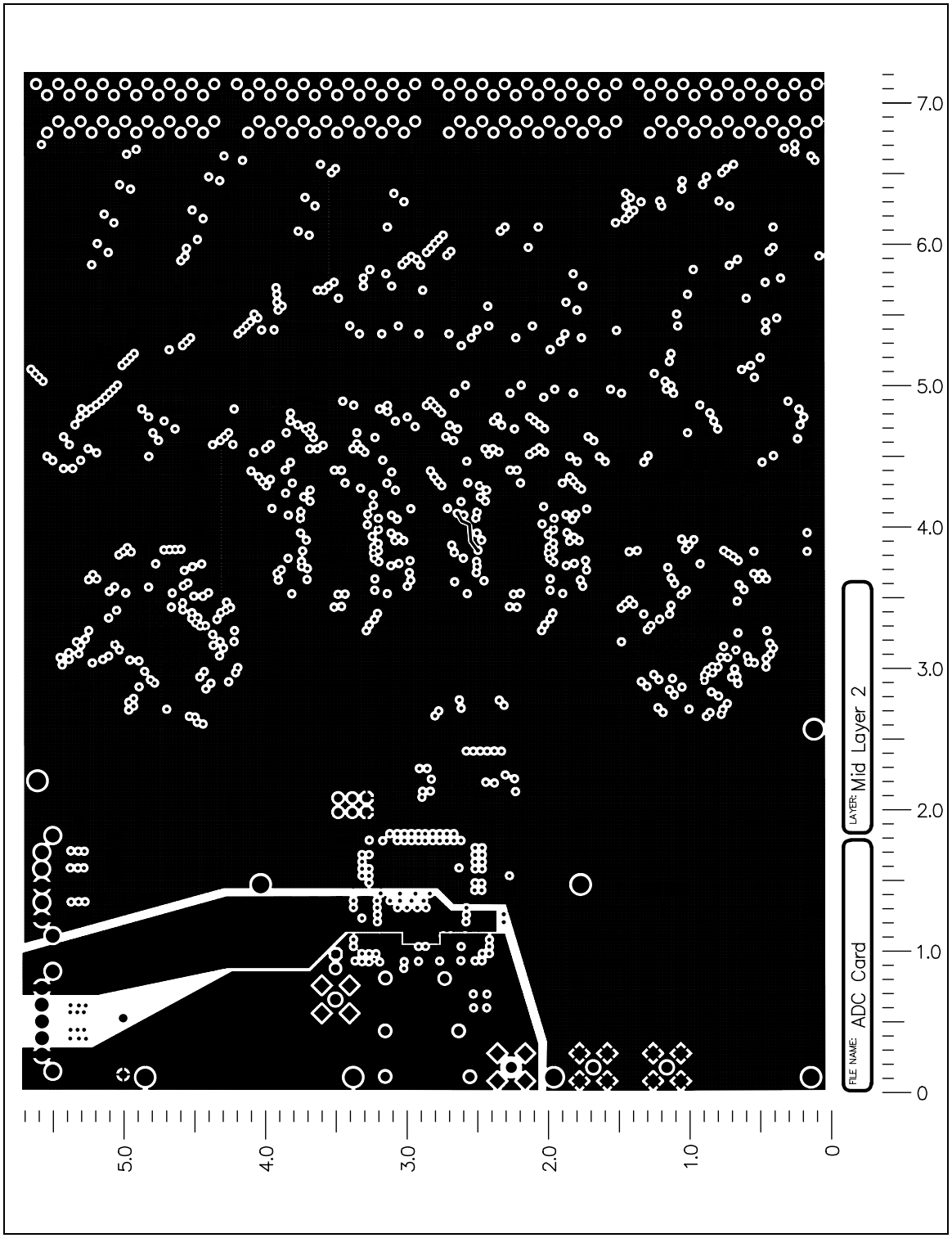
Title	PECL to TTL 6		
Size: B	Number:		Revision:
Date: 20-Sep-2001		Sheet of	
File: C:\officedrives\scsidrive\Design Explorer\revisions\Documents\Timing Board\Timing Board.ddb			

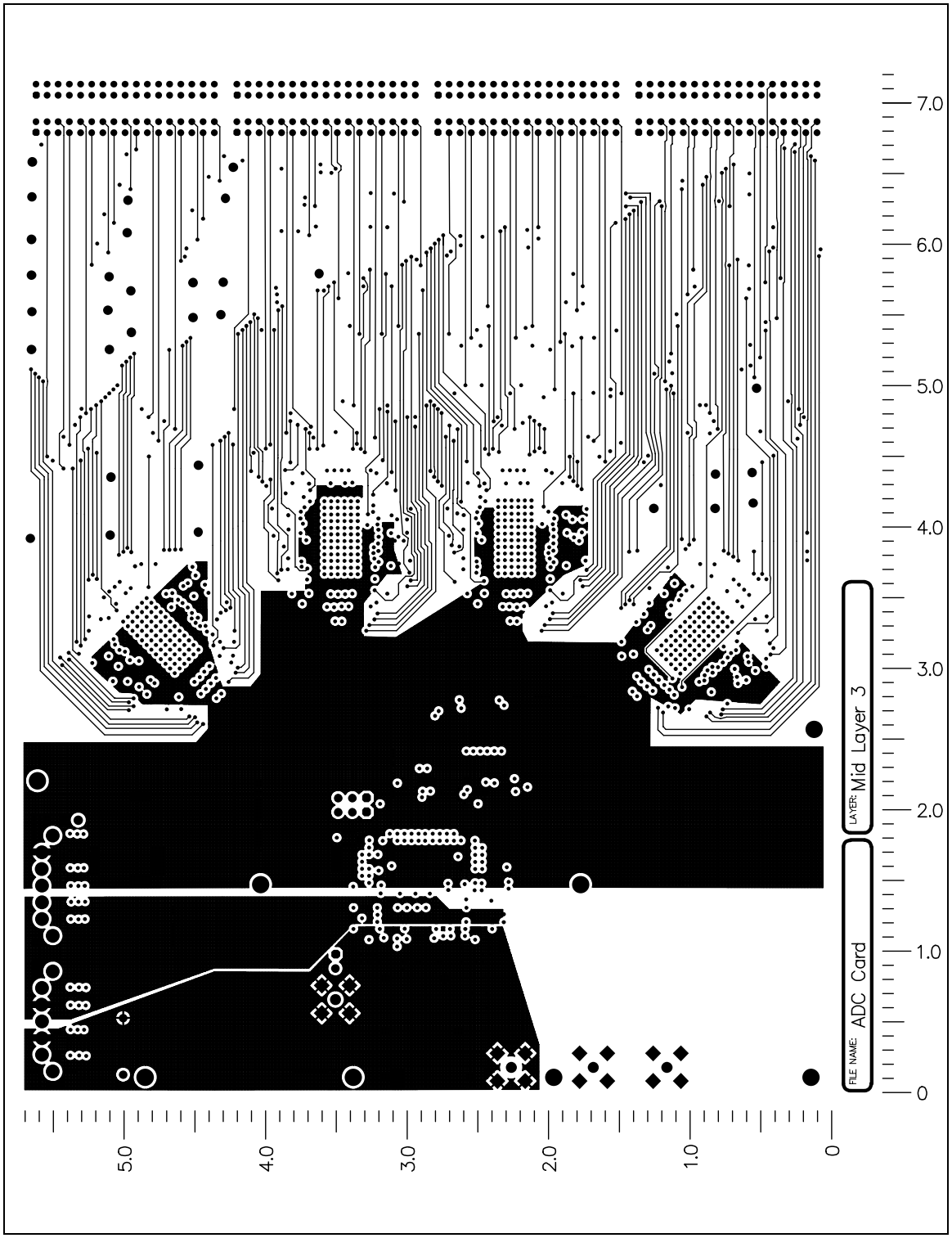
APPENDIX F ADC CARD PCB LAYOUT

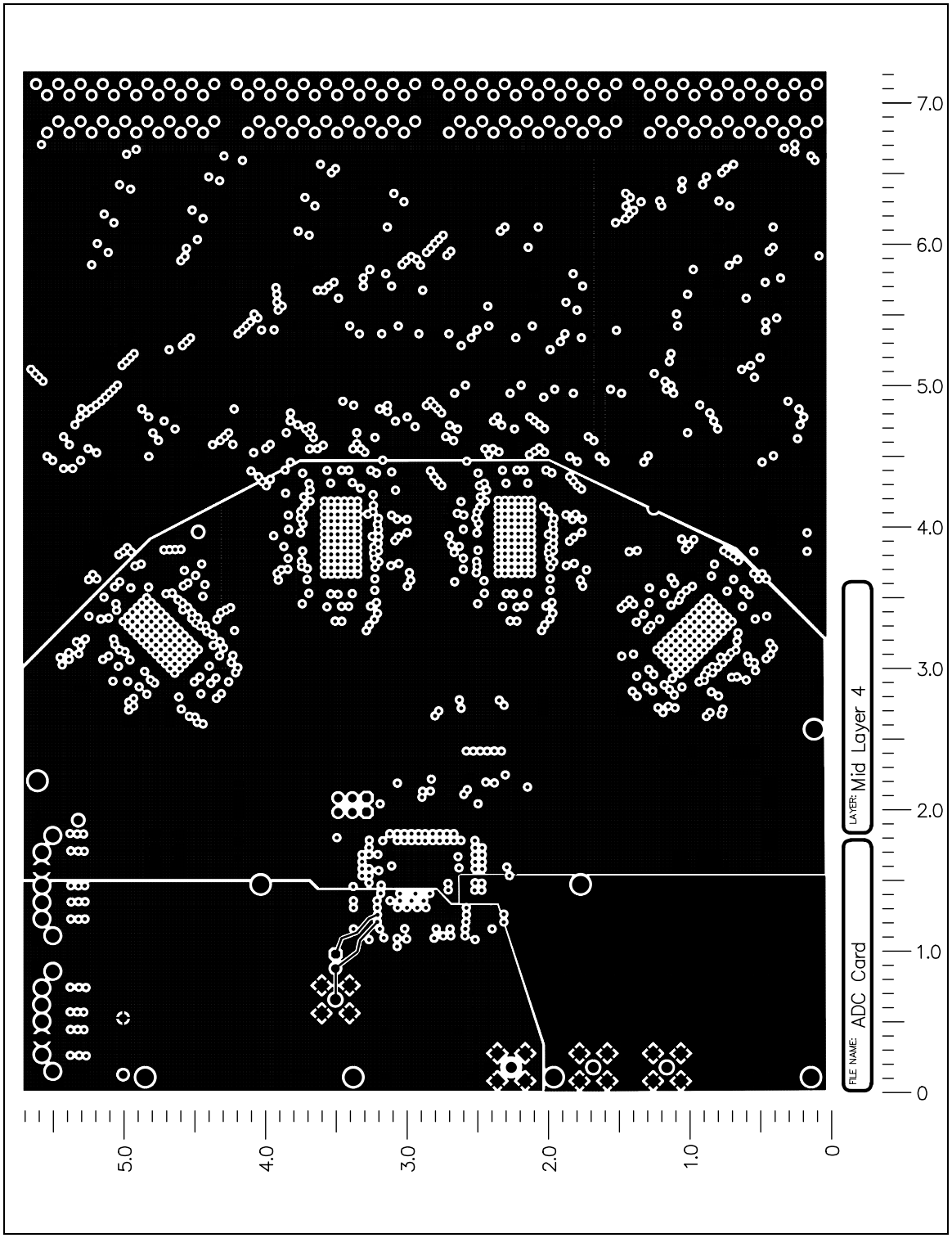

1 GSPS, 8 Bit A/D Convertor Card
University of Kansas - RSL 2000
REV 1
J13 Bandgap Reference
Short = Internal/Master
Open = External
J3 Decimation Mode 1-2 Disable 2-3 Enable
J4 Demux Enable Disable Enable
FILE NAME: ADC Card
LAYER: Top Overlay

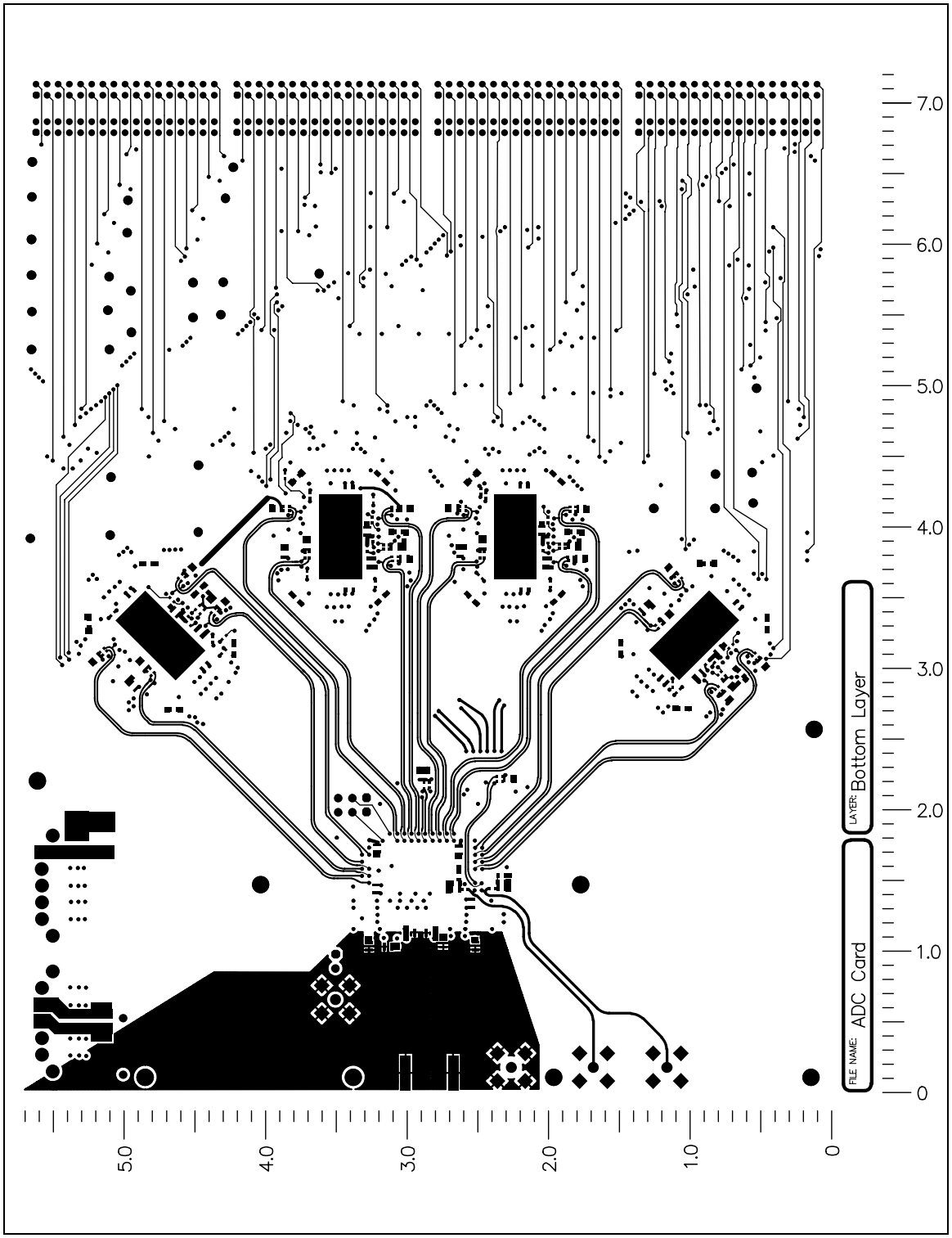

FILE NAME: ADC Card
LAYER: Top Layer
0
1.0
2.0
3.0
4.0
5.0
6.0
7.0

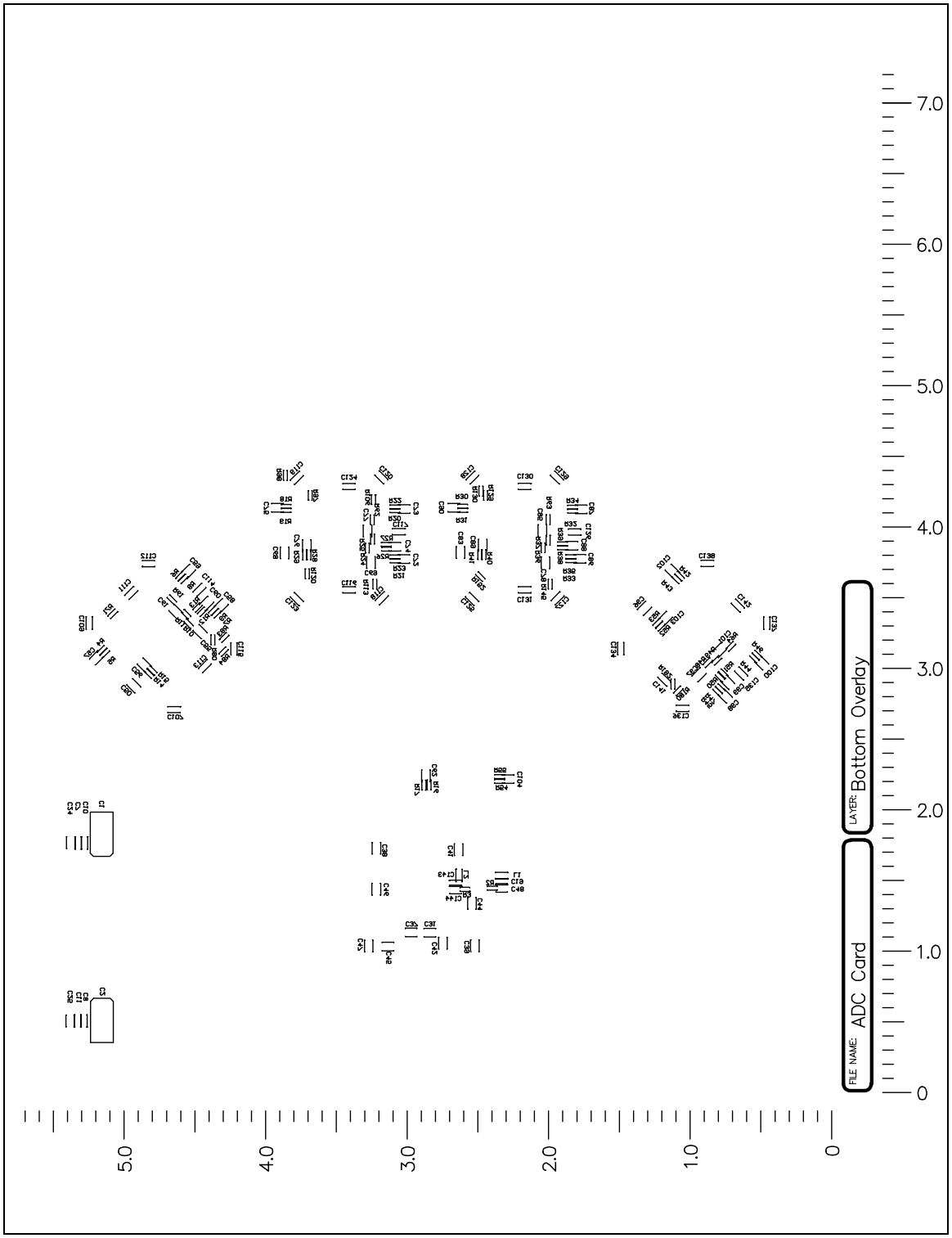

FILE NAME: ADC Card
LAYER: Mid Layer 1
0
1.0
2.0
3.0
4.0
5.0
6.0
7.0


FILE NAME: ADC Card
LAYER: Mid Layer 2
0
1.0
2.0
3.0
4.0
5.0
6.0
7.0

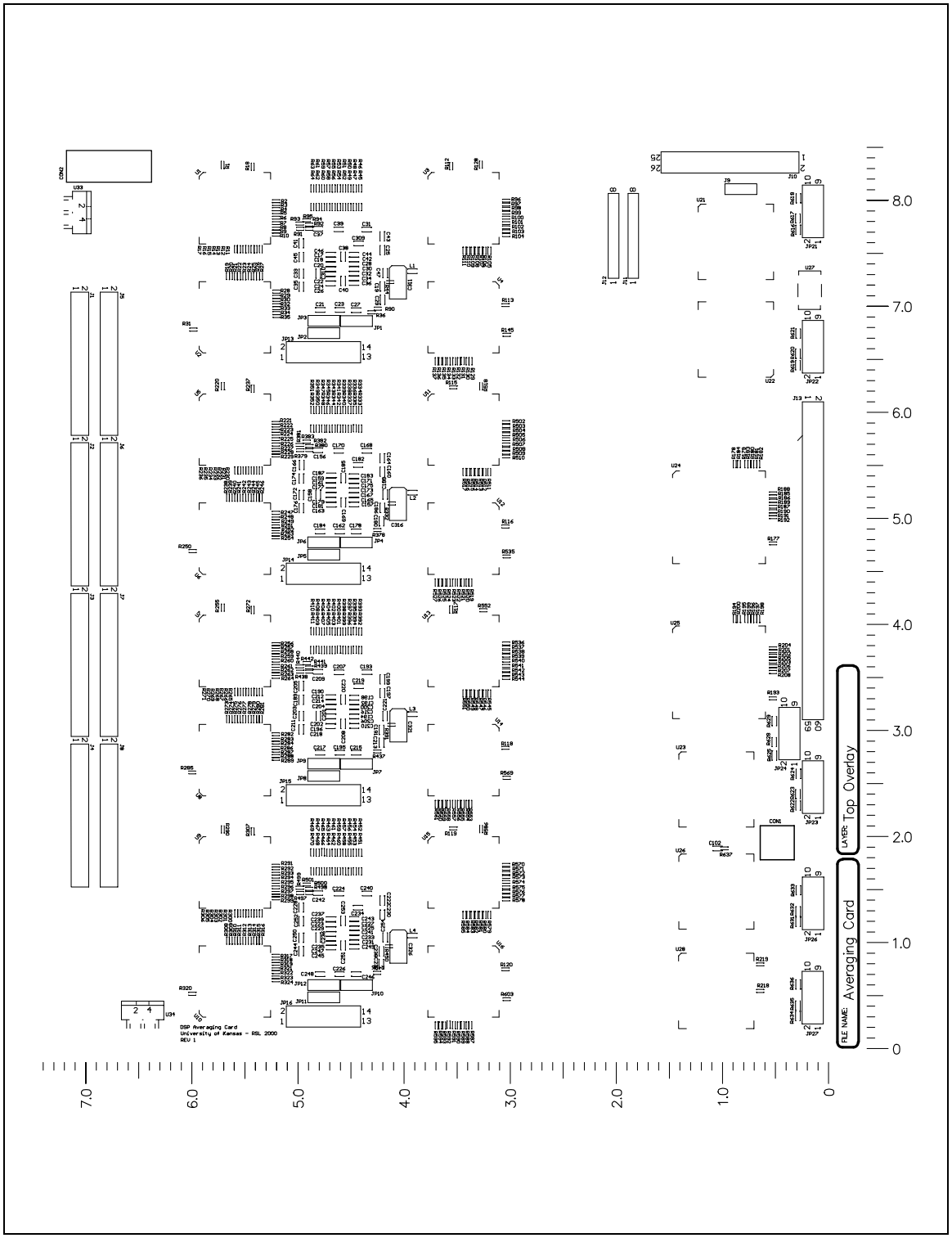

FILE NAME: ADC Card
LAYER: Mid Layer 3
0
1.0
2.0
3.0
4.0
5.0
6.0
7.0

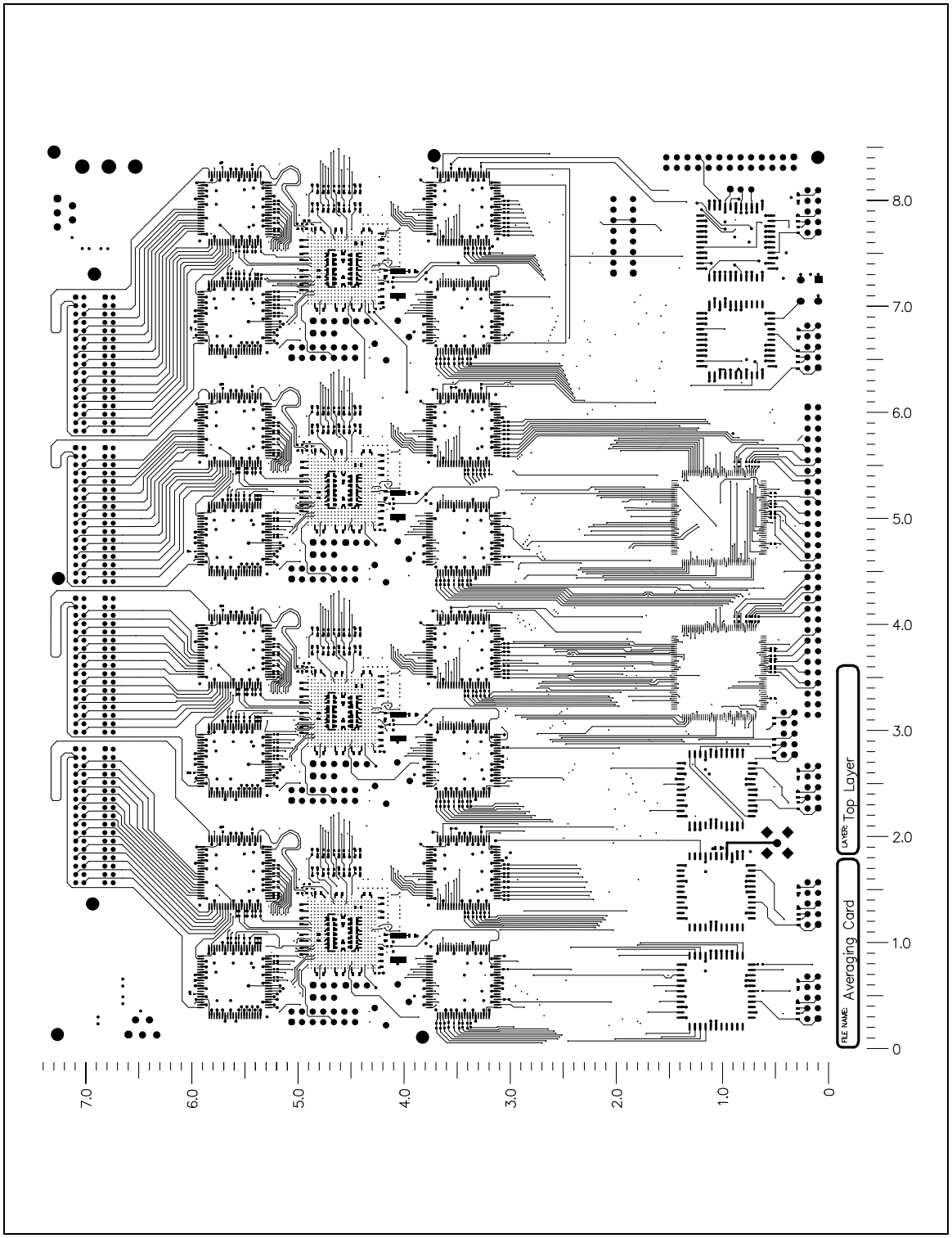

FILE NAME: ADC Card
LAYER: Mid Layer 4
0
1.0
2.0
3.0
4.0
5.0
6.0
7.0

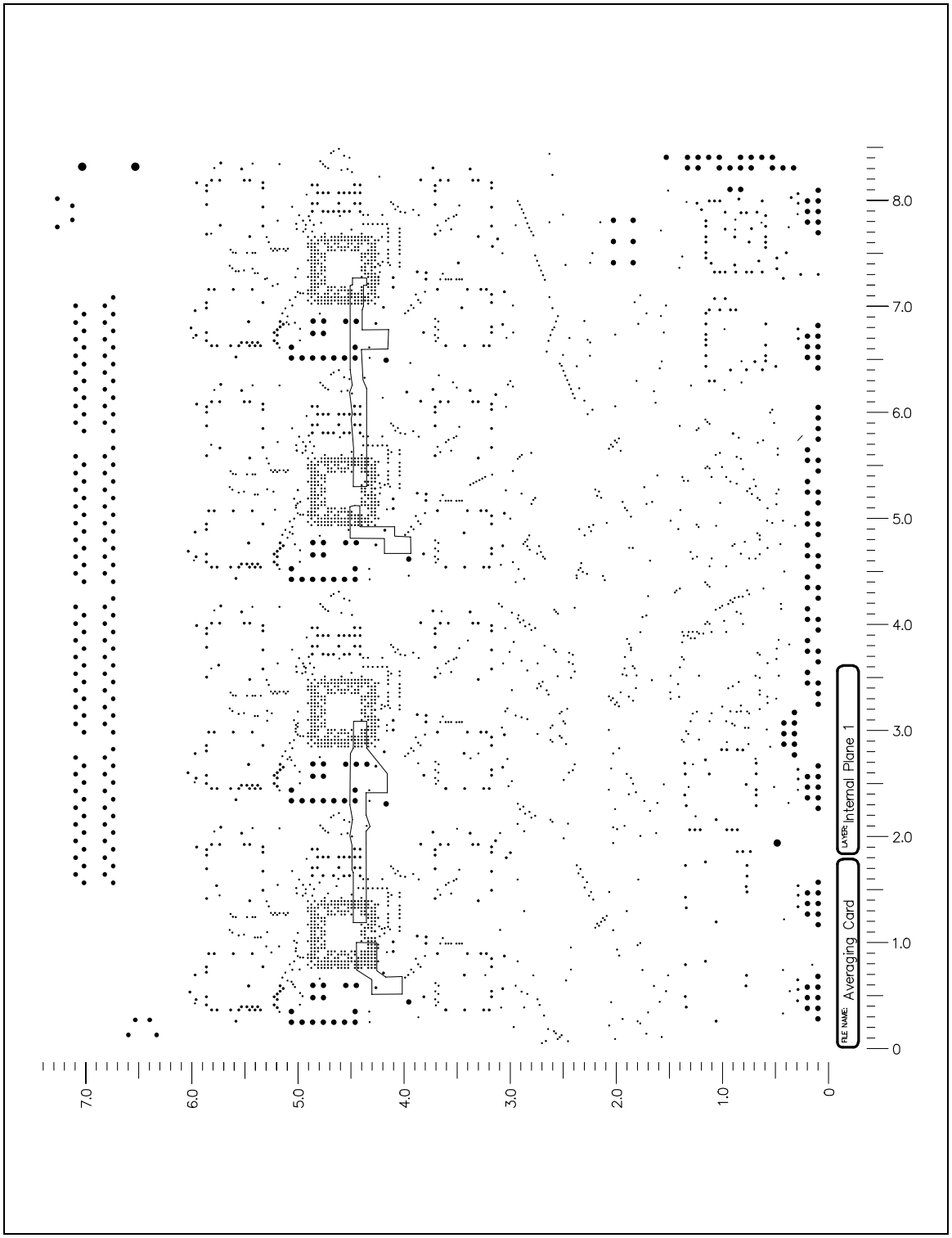

FILE NAME: ADC Card
LAYER: Bottom Layer
0
1.0
2.0
3.0
4.0
5.0
6.0
7.0

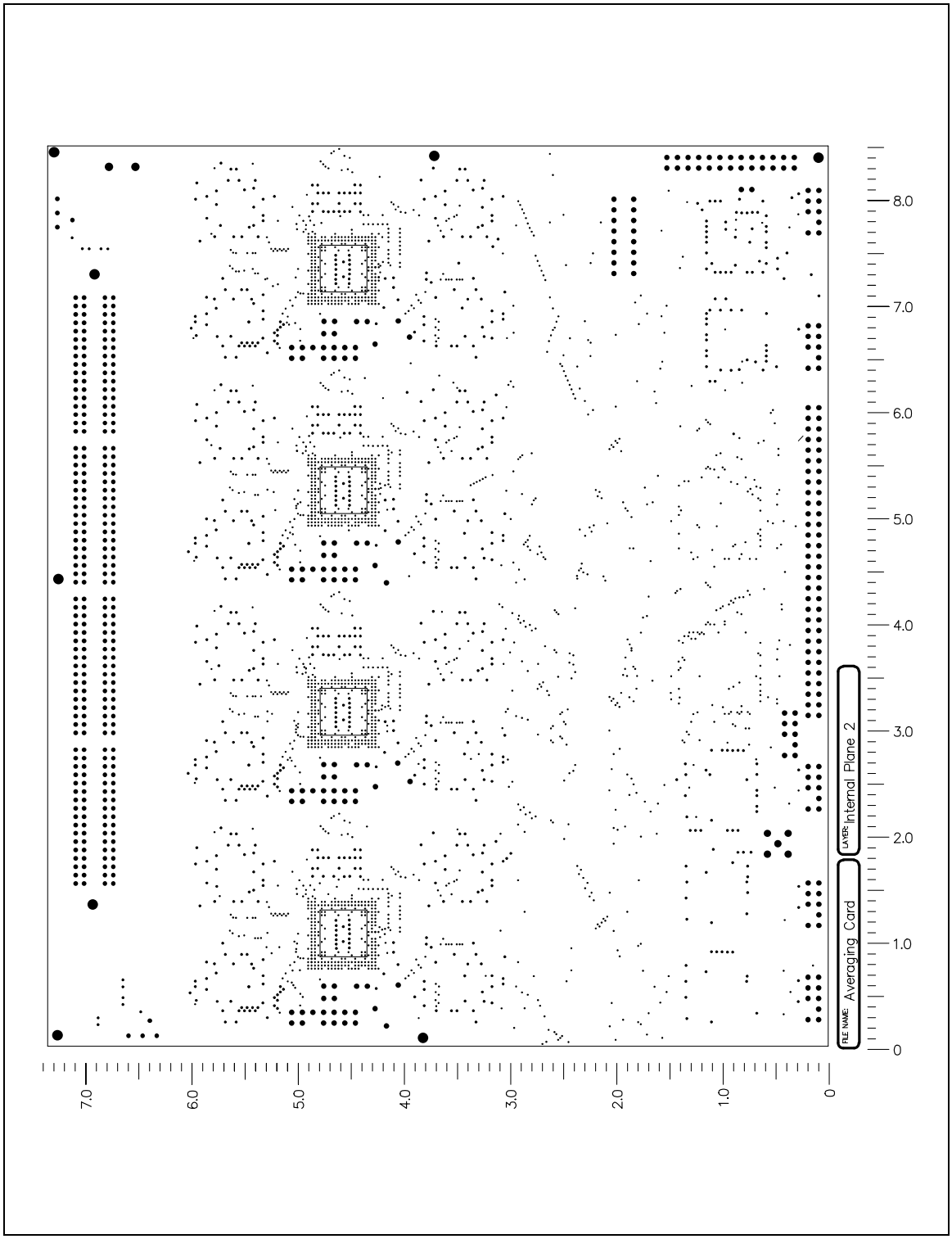

FILE NAME: ADC Card
LAYER: Bottom Overlay
0
1.0
2.0
3.0
4.0
5.0
6.0
7.0

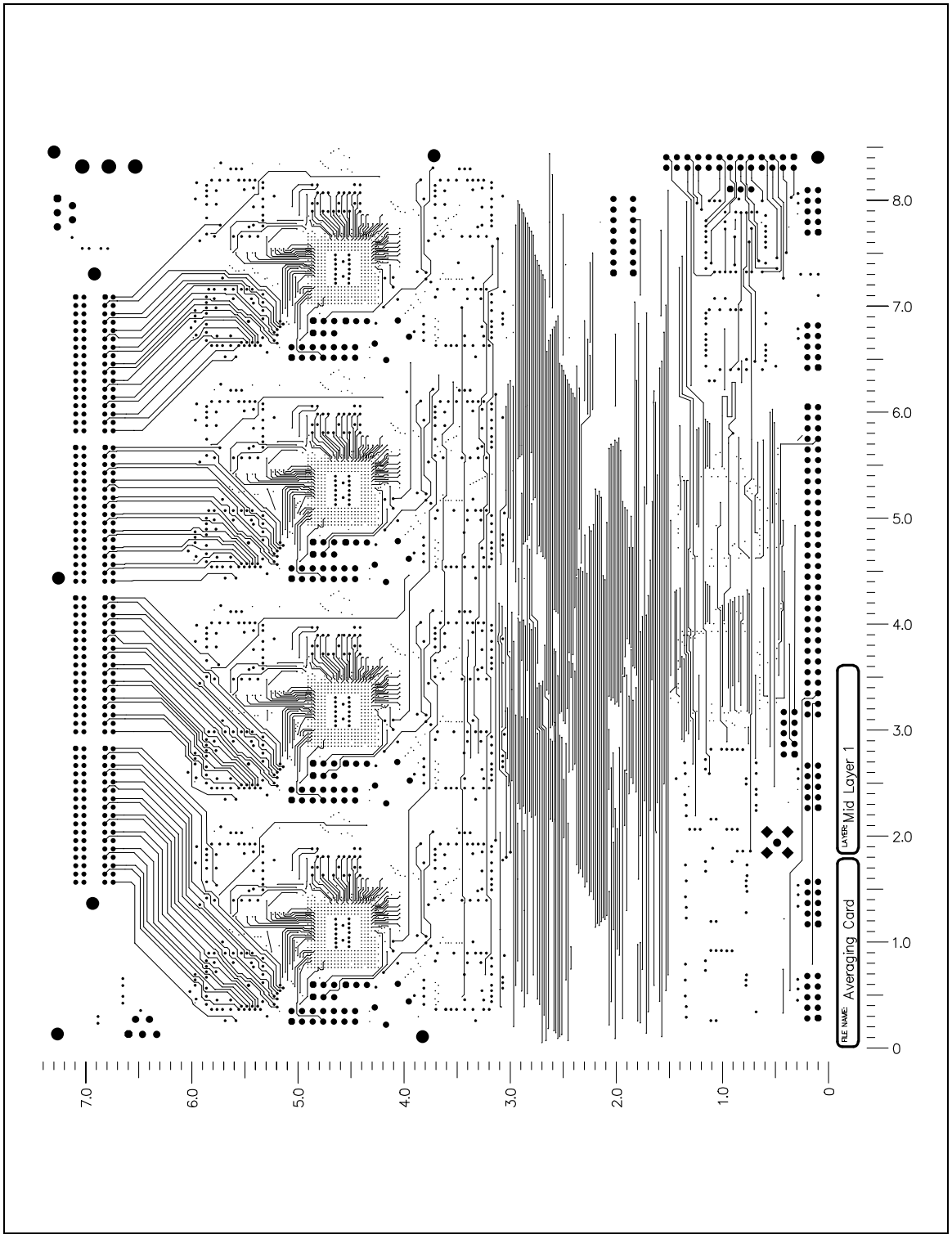
APPENDIX G AVERAGING CARD PCB LAYOUT

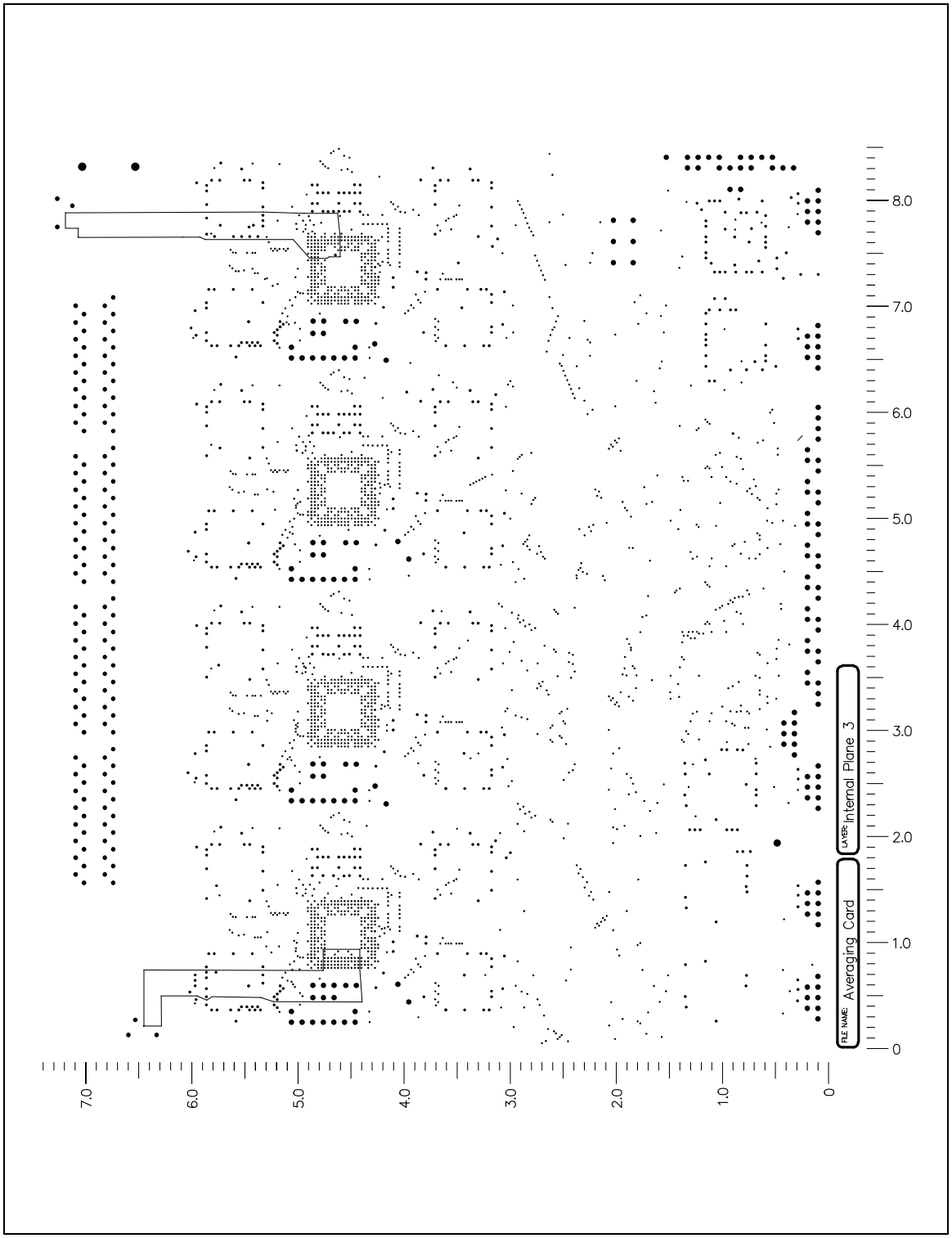

FILE NAME: Averaging Card
LAYER: Top Overlay
DSP Averaging Card
University of Kansas - RSL 2000
REV 1

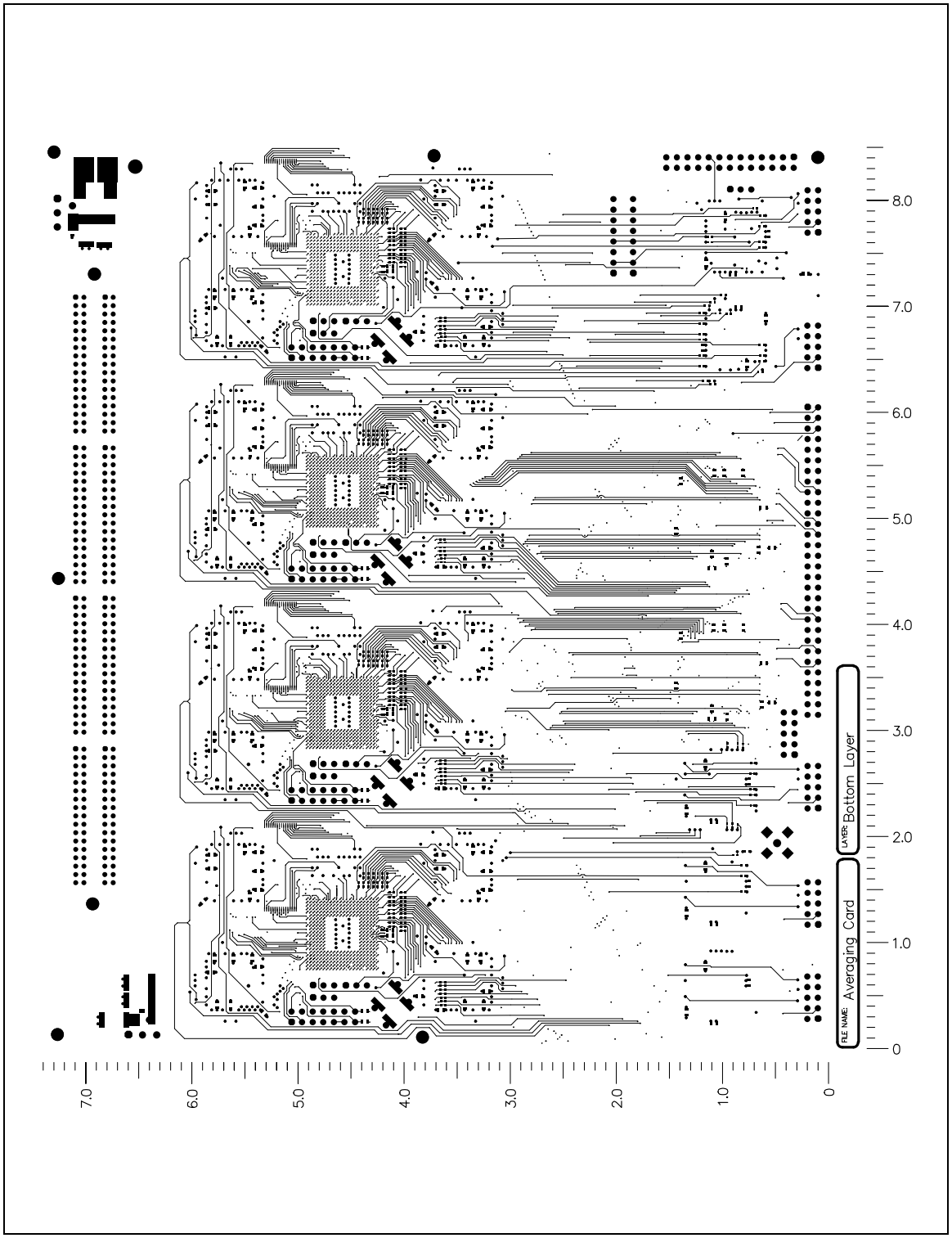

FILE NAME: Averaging Card
LAYER: Top Layer
0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0

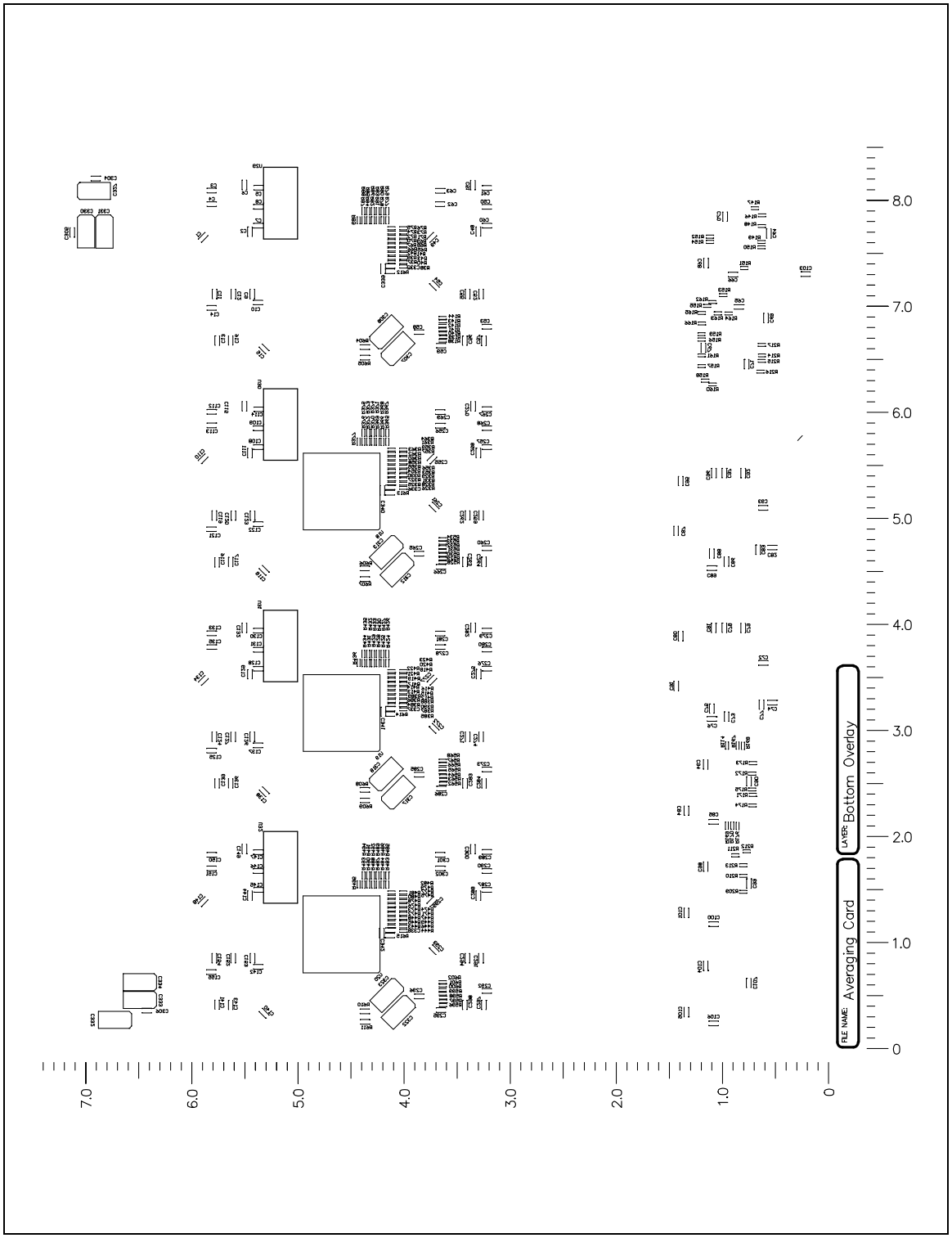

FILE NAME: Averaging Card
LAYER: Internal Plane 1
0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0


FILE NAME: Averaging Card
LAYER: Internal Plane 2
0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0

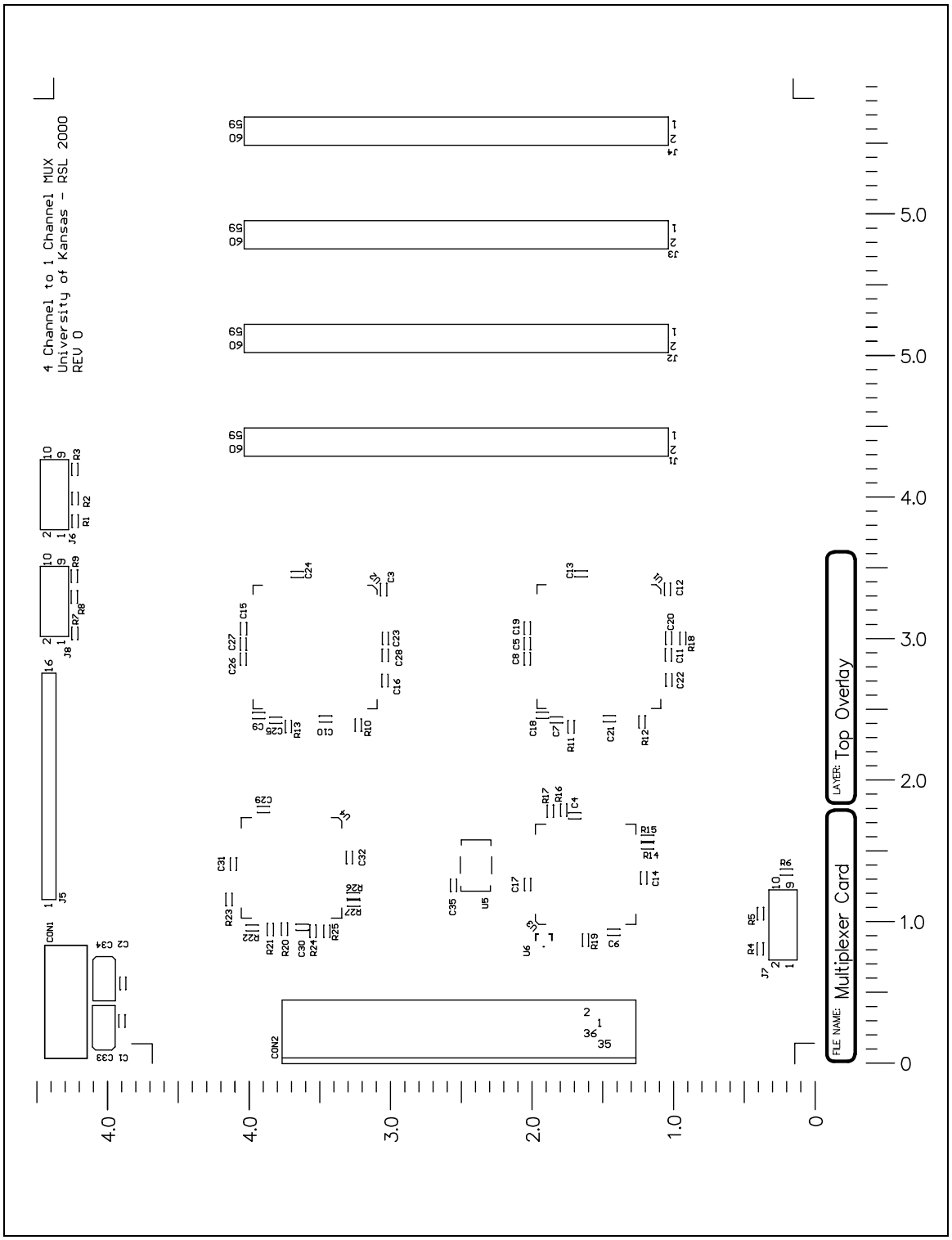

FILE NAME: Averaging Card
LAYER: Mid Layer 1
0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0

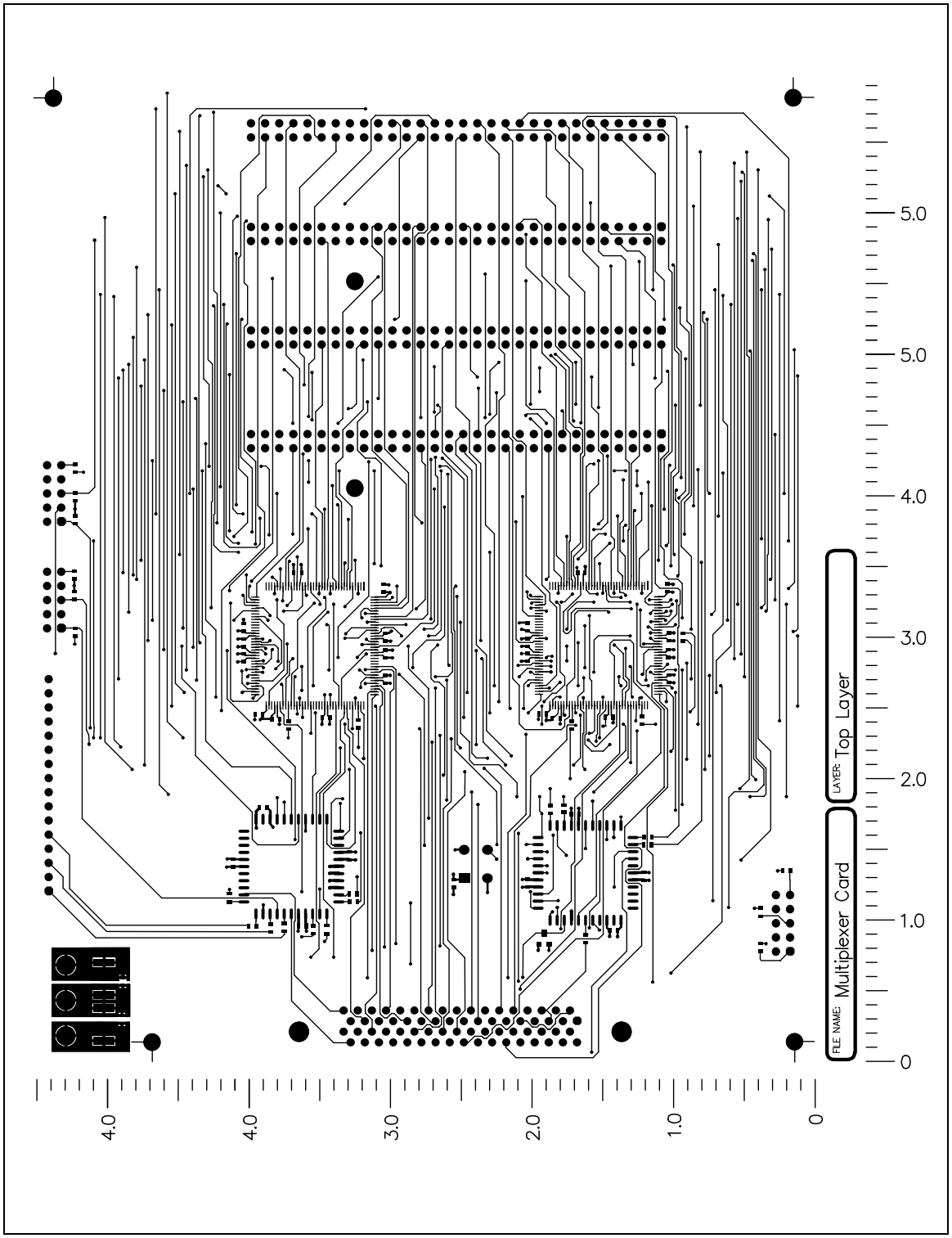

FILE NAME: Averaging Card
LAYER: Internal Plane 3
0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0

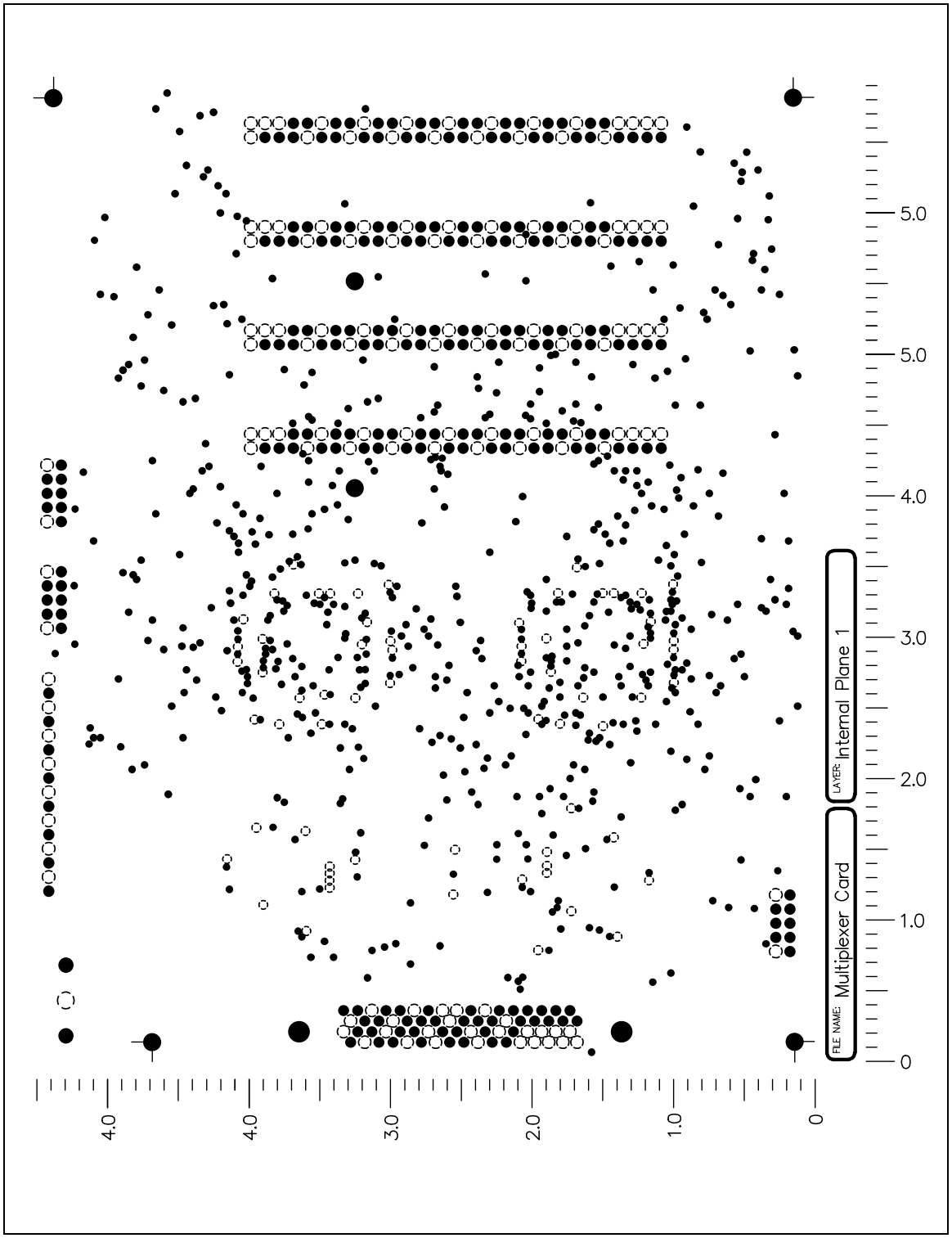

FILE NAME: Averaging Card
LAYER: Bottom Layer
0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0

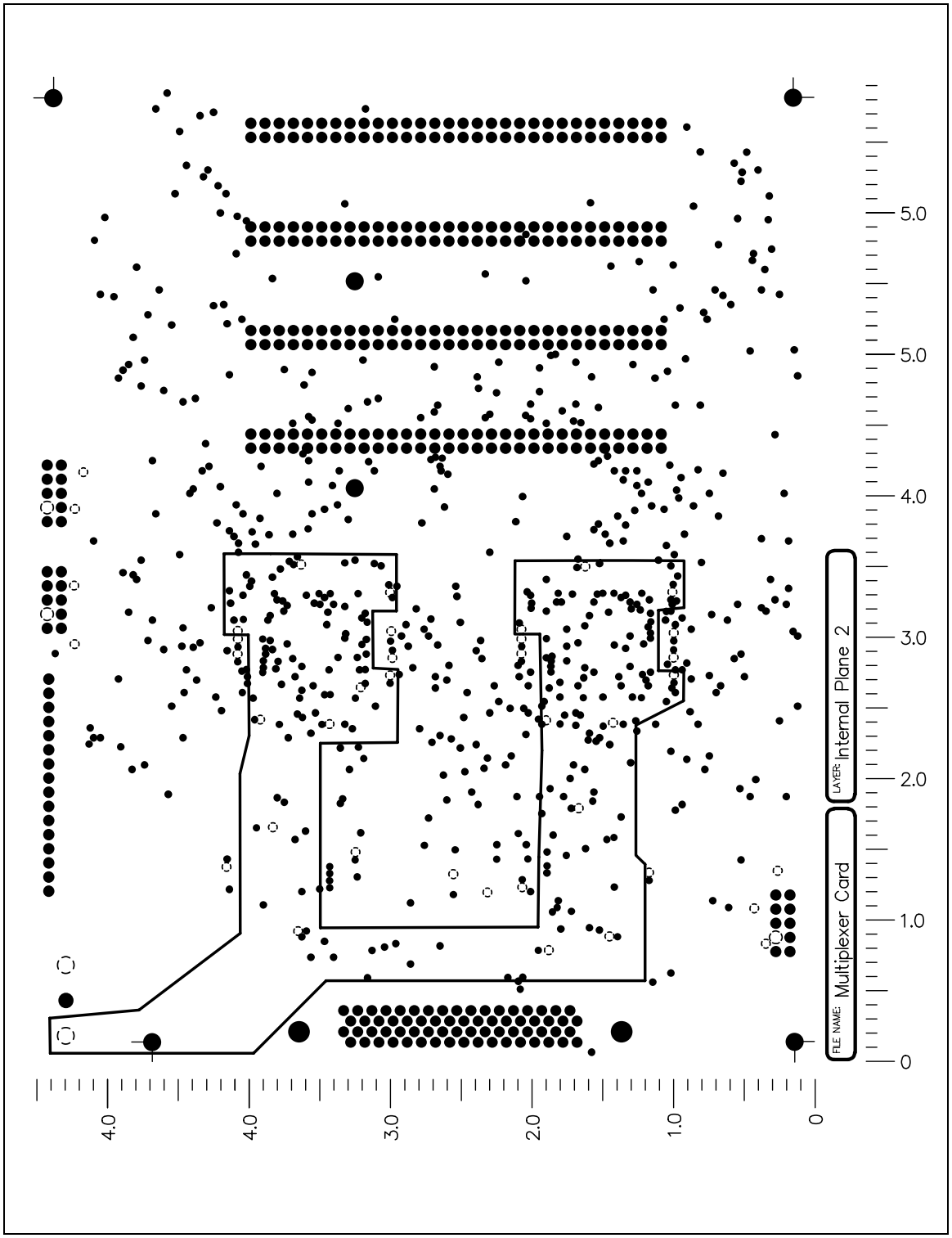

FILE NAME: Averaging Card
LAYER: Bottom Overlay
0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0

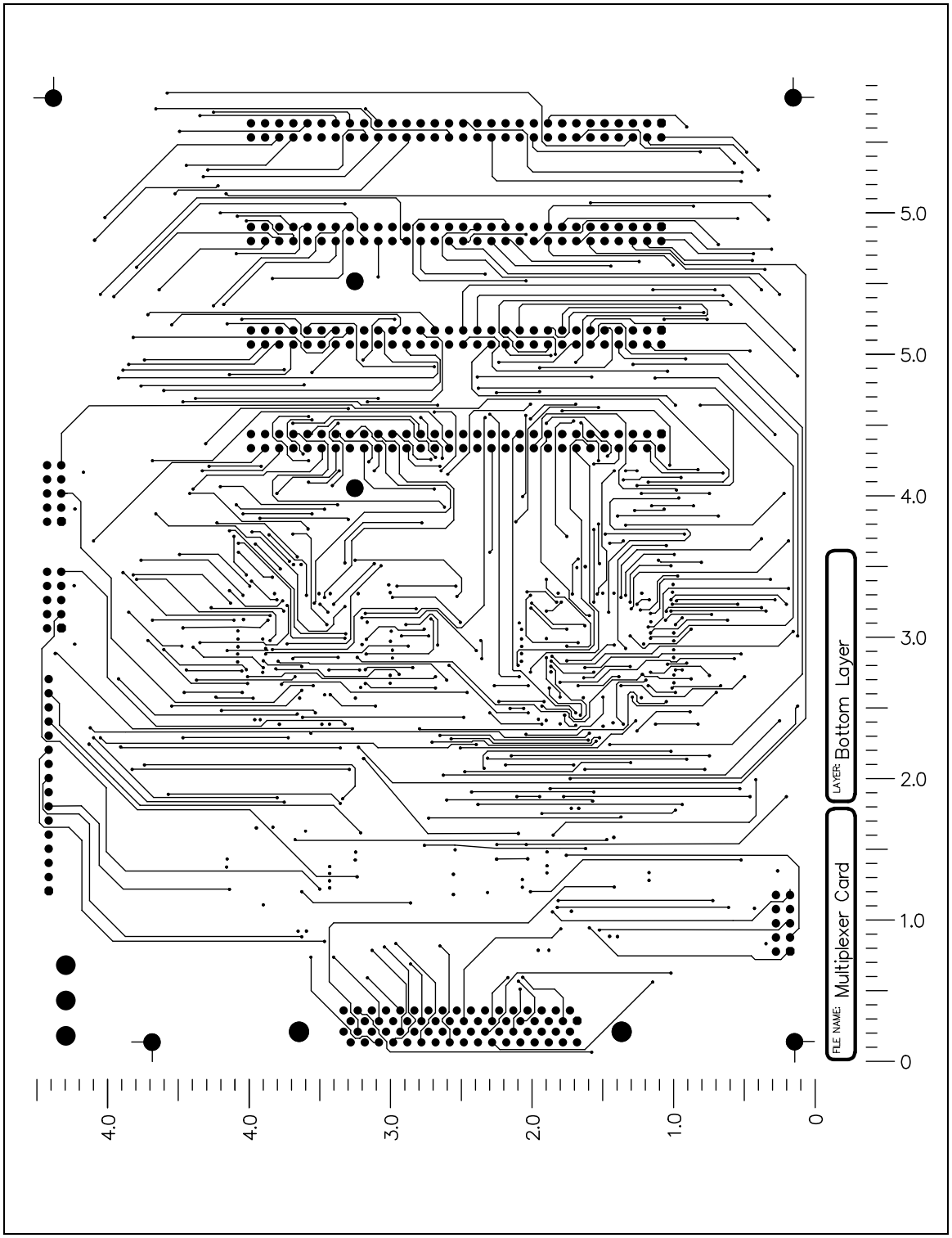
APPENDIX H MULTIPLEXER CARD PCB LAYOUT


4 Channel to 1 Channel MUX
University of Kansas - RSL 2000
REV 0
FILE NAME: Multiplexer Card
LAYER: Top Overlay

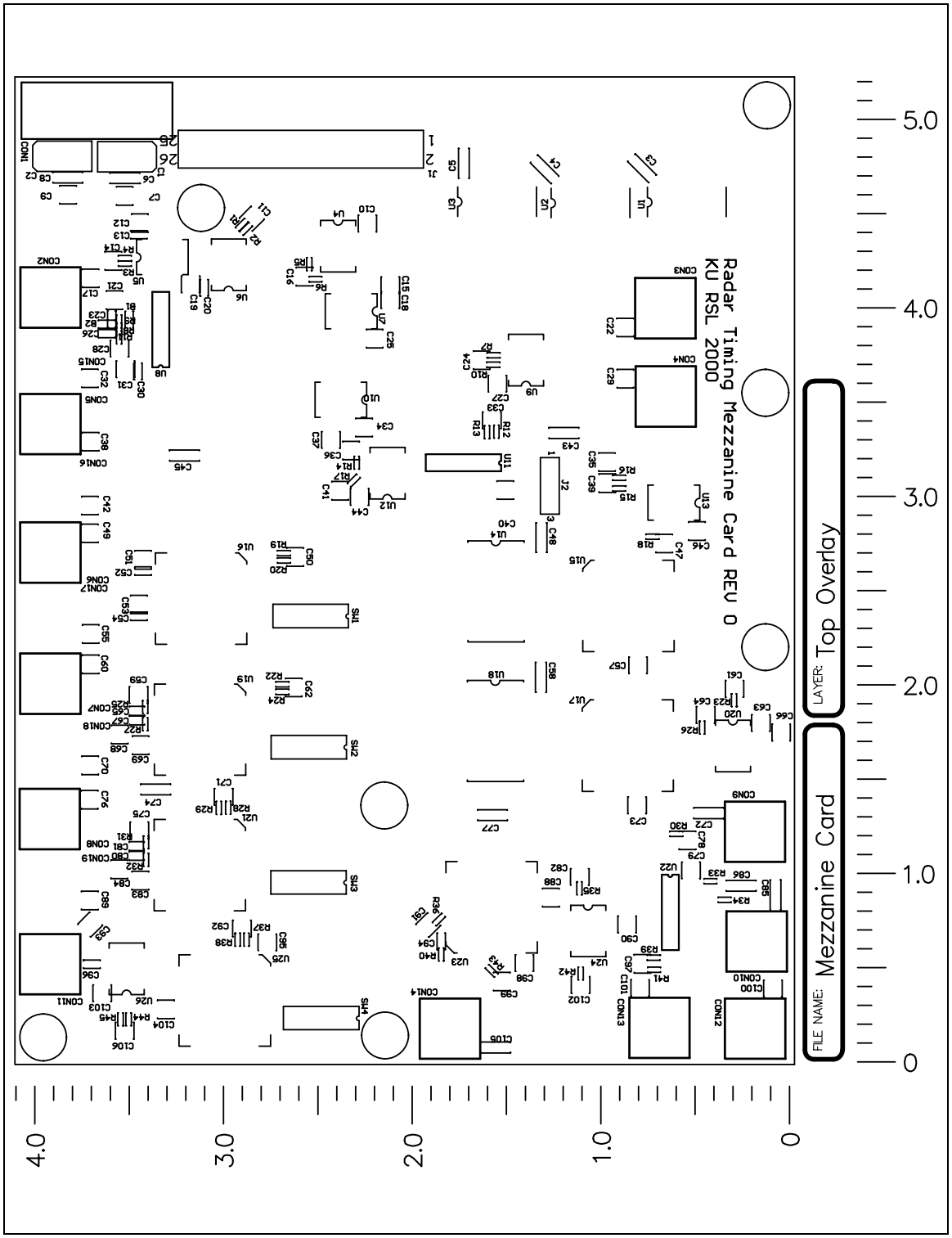

FILE NAME: Multiplexer Card
LAYER: Top Layer
0
1.0
2.0
3.0
4.0
5.0
5.0
4.0
3.0
2.0
1.0
0
4.0

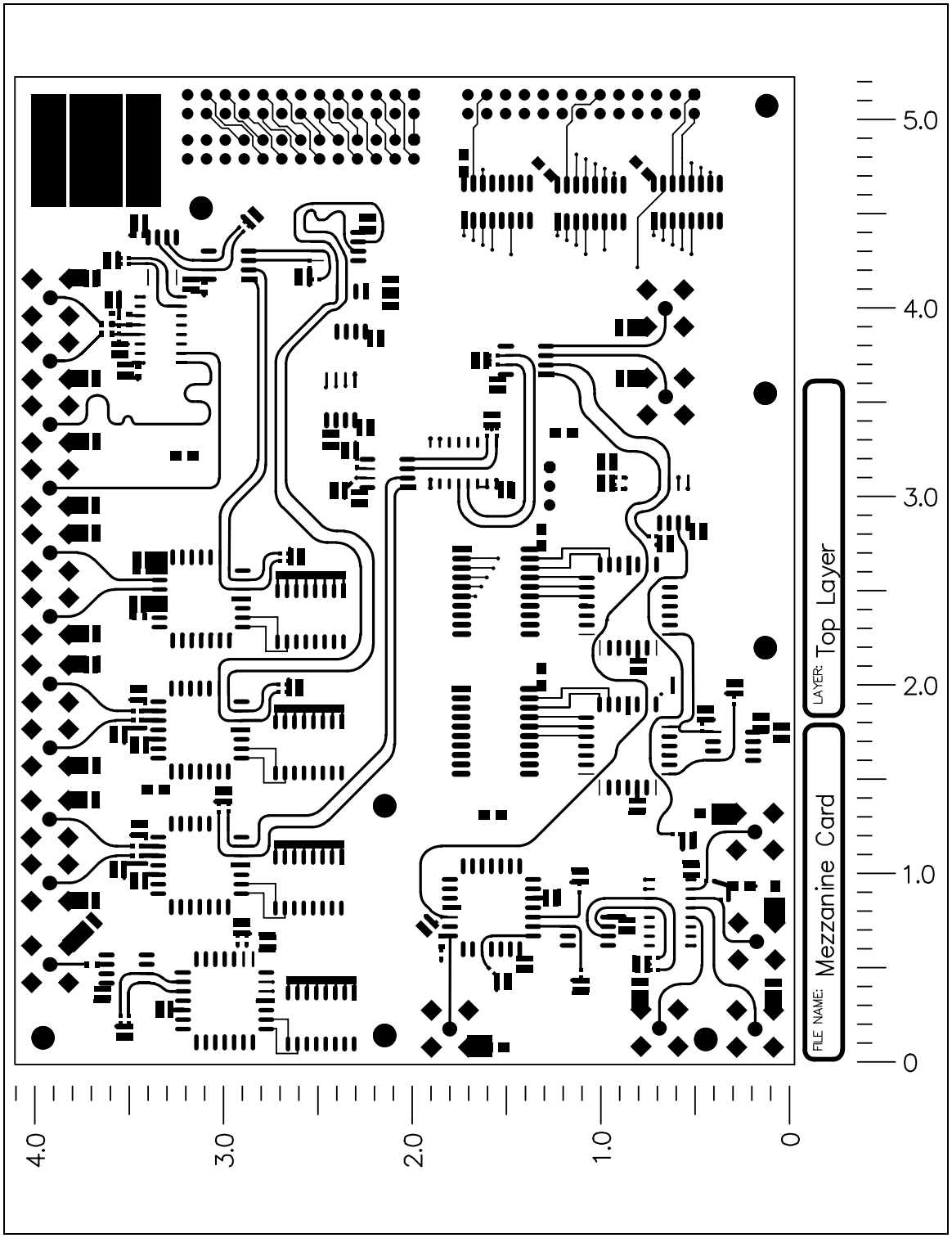

FILE NAME: Multiplexer Card
LAYER: Internal Plane 1
0
1.0
2.0
3.0
4.0
5.0
5.0
4.0
3.0
2.0
1.0
0

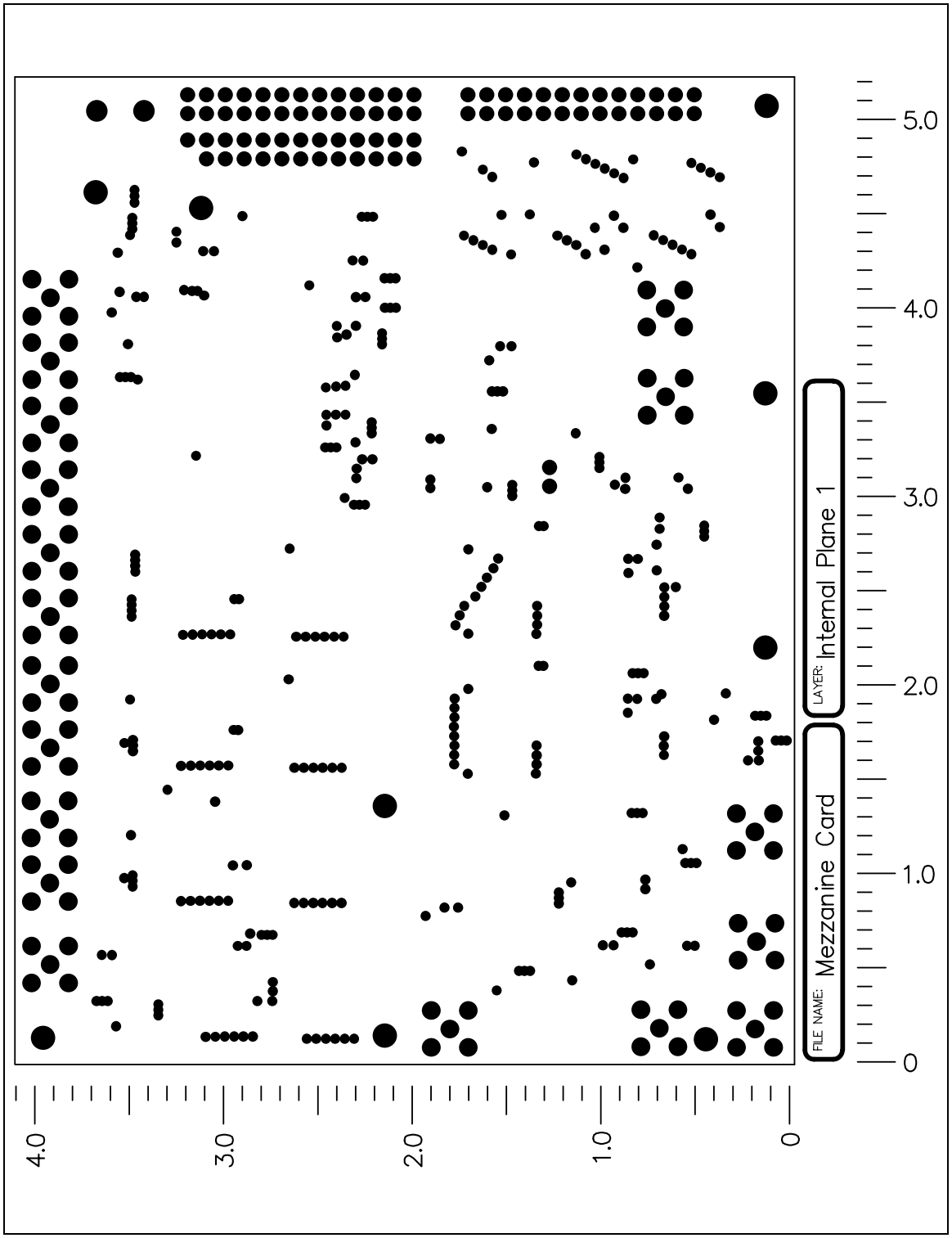

FILE NAME: Multiplexer Card
LAYER: Internal Plane 2
0
1.0
2.0
3.0
4.0
5.0
5.0
4.0
3.0
2.0
1.0
0

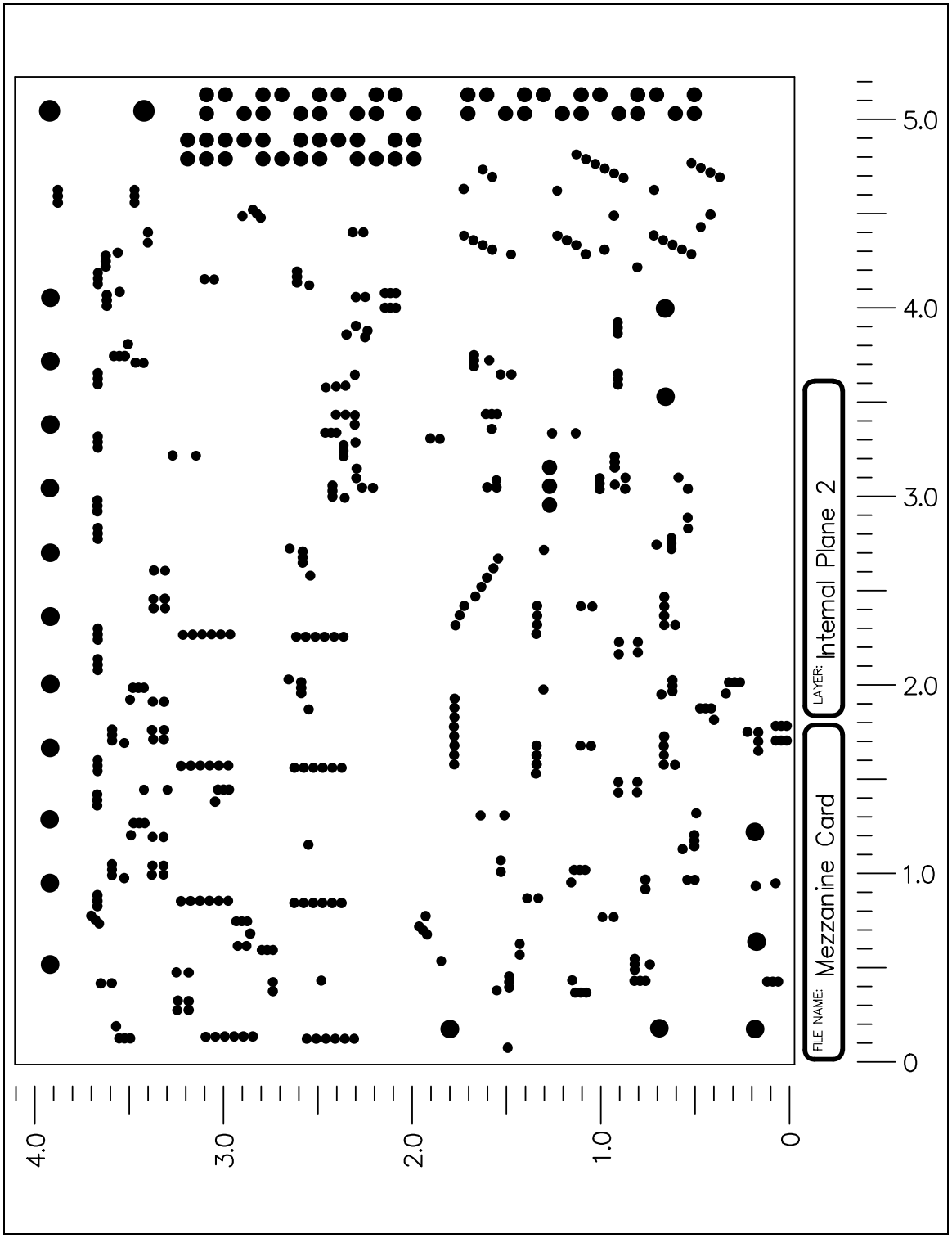

FILE NAME: Multiplexer Card
LAYER: Bottom Layer
0
1.0
2.0
3.0
4.0
5.0
5.0
4.0
3.0
2.0
1.0
0

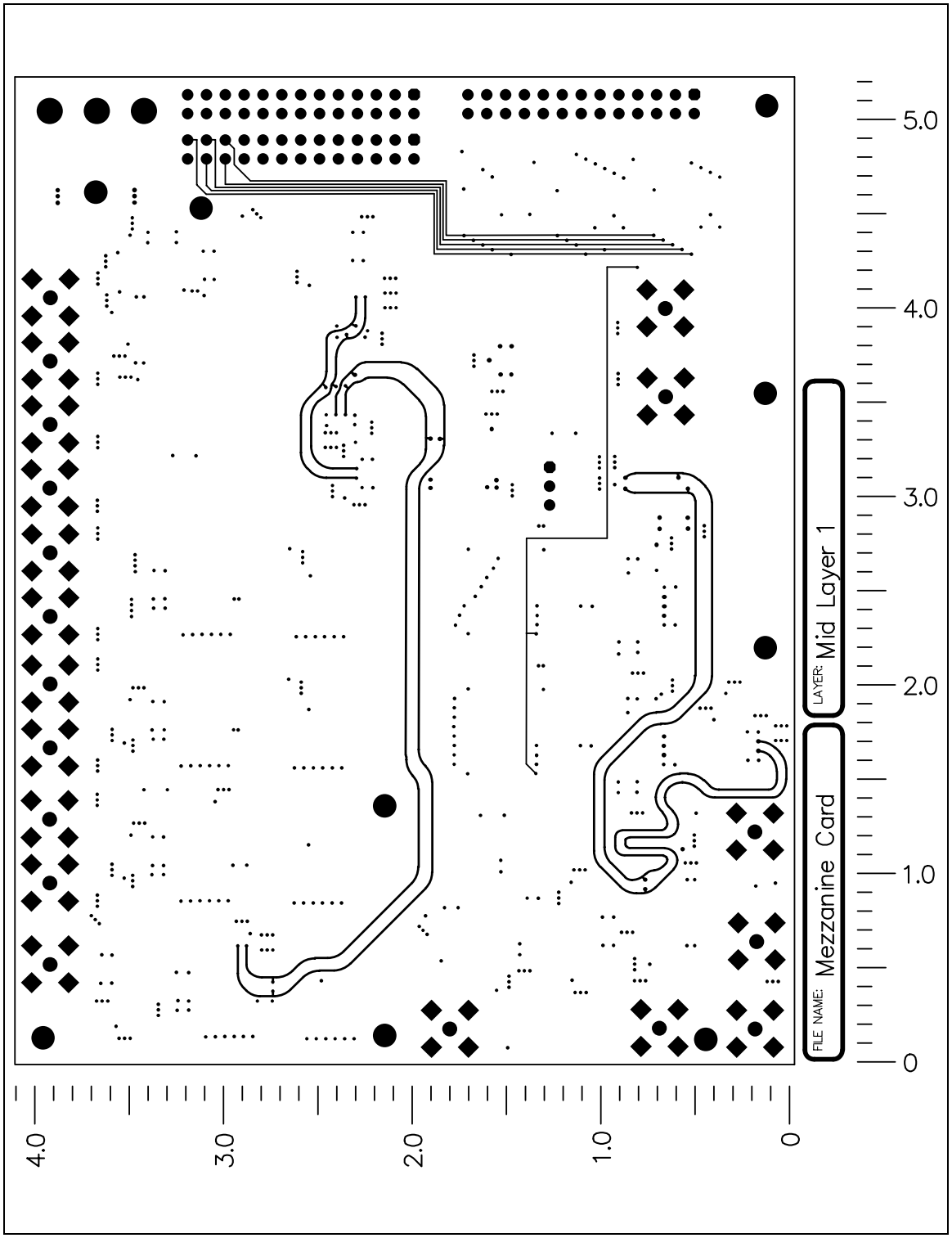
APPENDIX I TIMING MEZZANINE CARD PCB LAYOUT

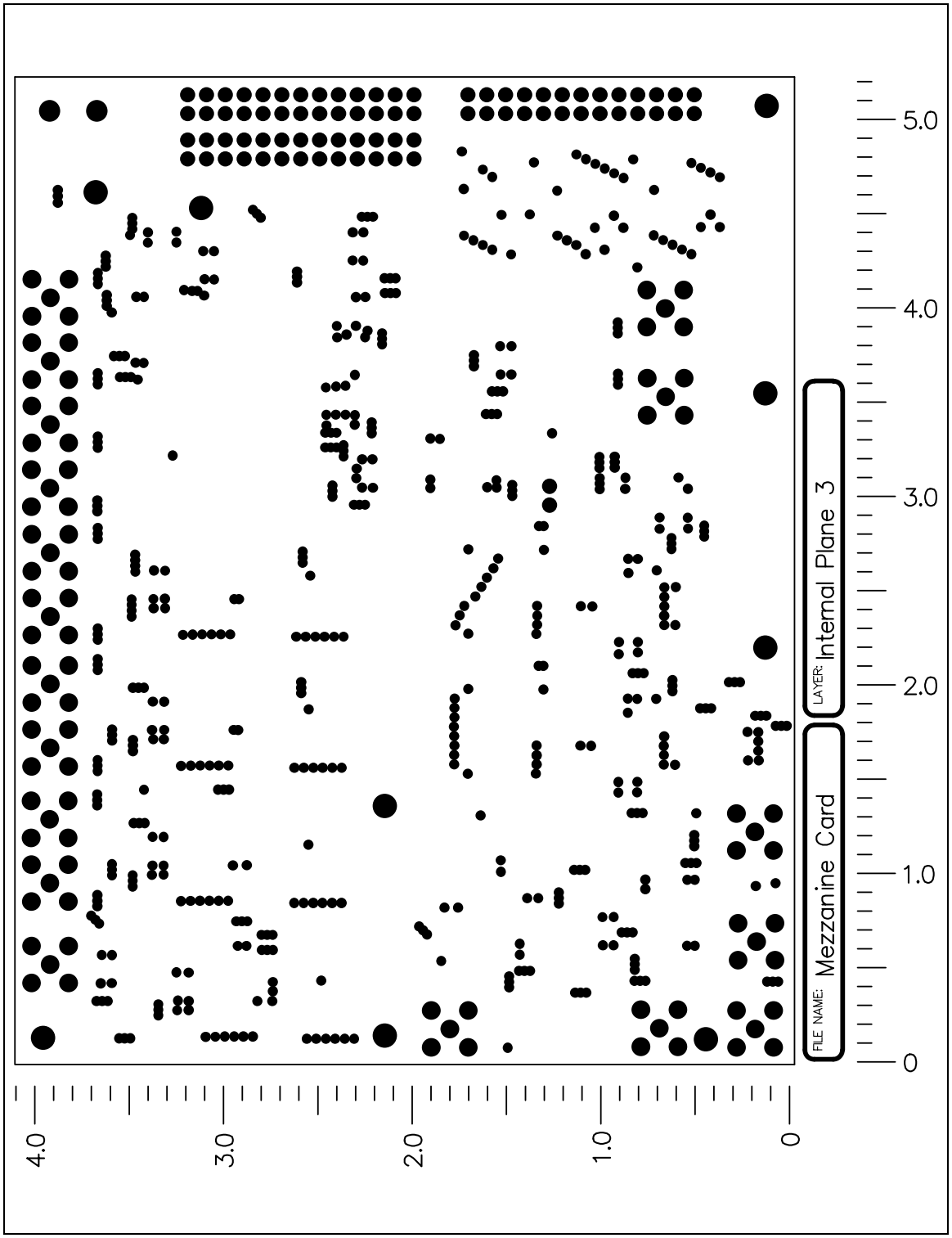

Radar Timing Mezzanine Card REV 0
KU RSL 2000
LAYER: Top Overlay
FILE NAME: Mezzanine Card
5.0
4.0
3.0
2.0
1.0
0
4.0
3.0
2.0
1.0
0

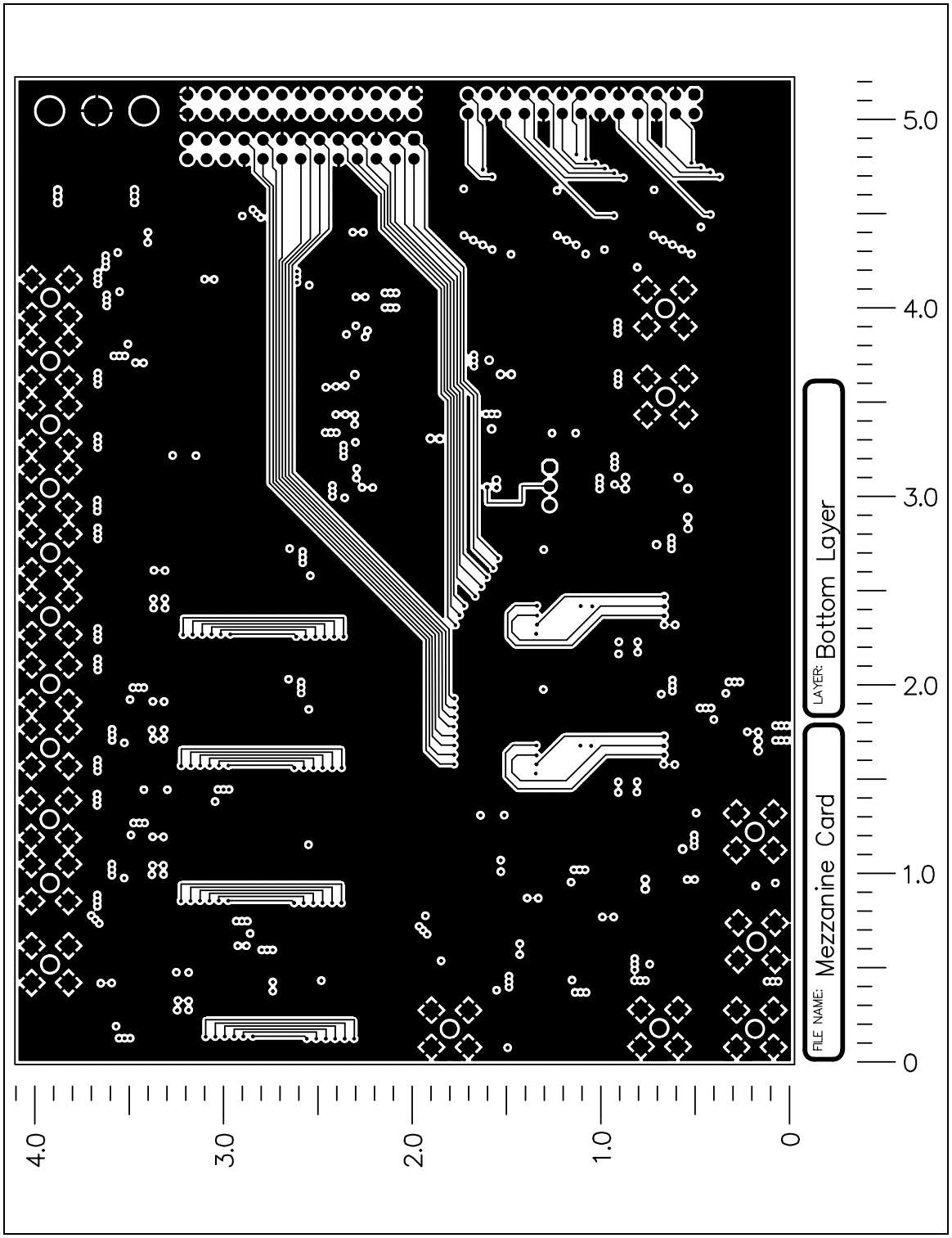

FILE NAME: Mezzanine Card
LAYER: Top Layer
0
1.0
2.0
3.0
4.0
5.0

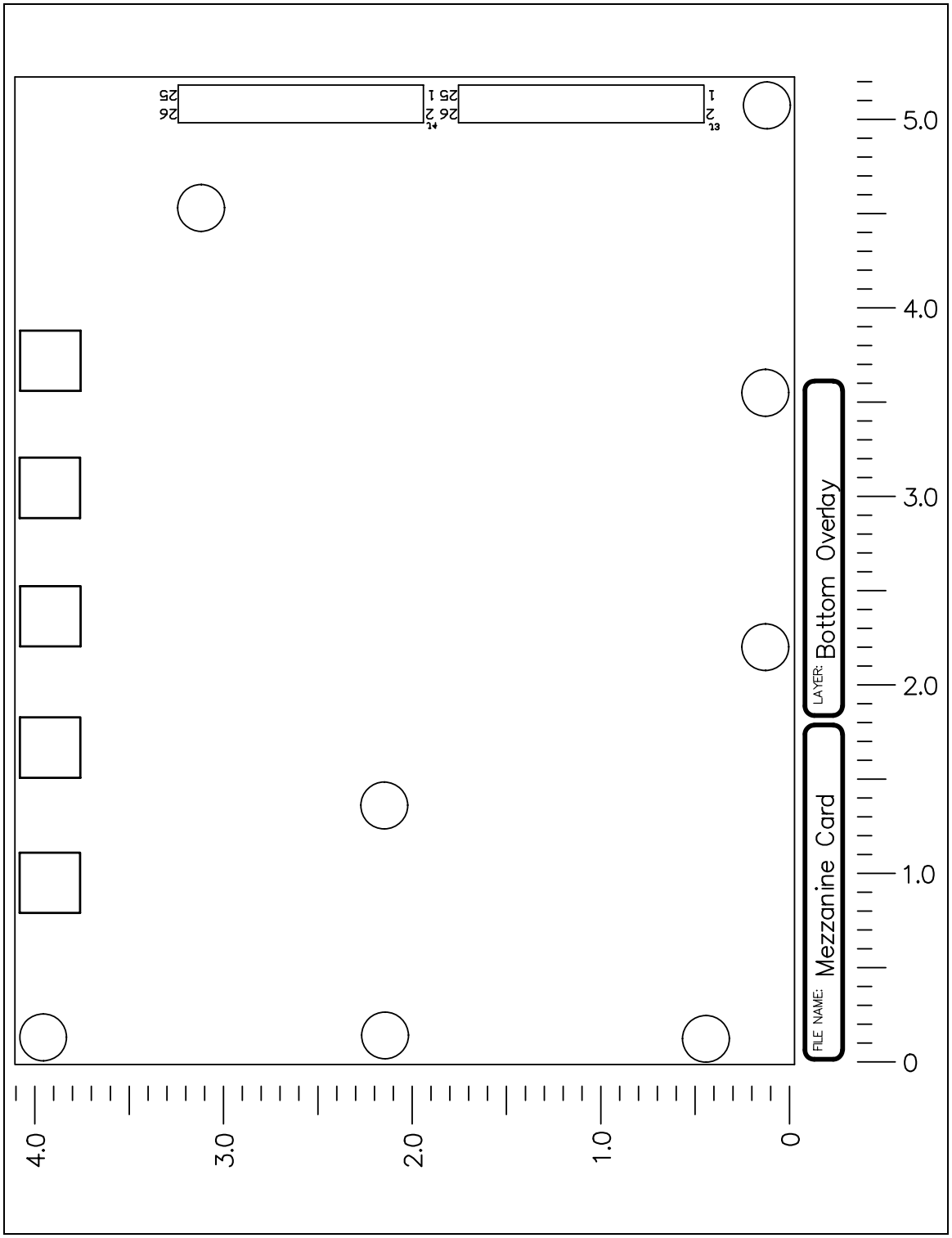

FILE NAME: Mezzanine Card
LAYER: Internal Plane 1
0
1.0
2.0
3.0
4.0
5.0
0
1.0
2.0
3.0
4.0


FILE NAME: Mezzanine Card
LAYER: Internal Plane 2
0
1.0
2.0
3.0
4.0
5.0

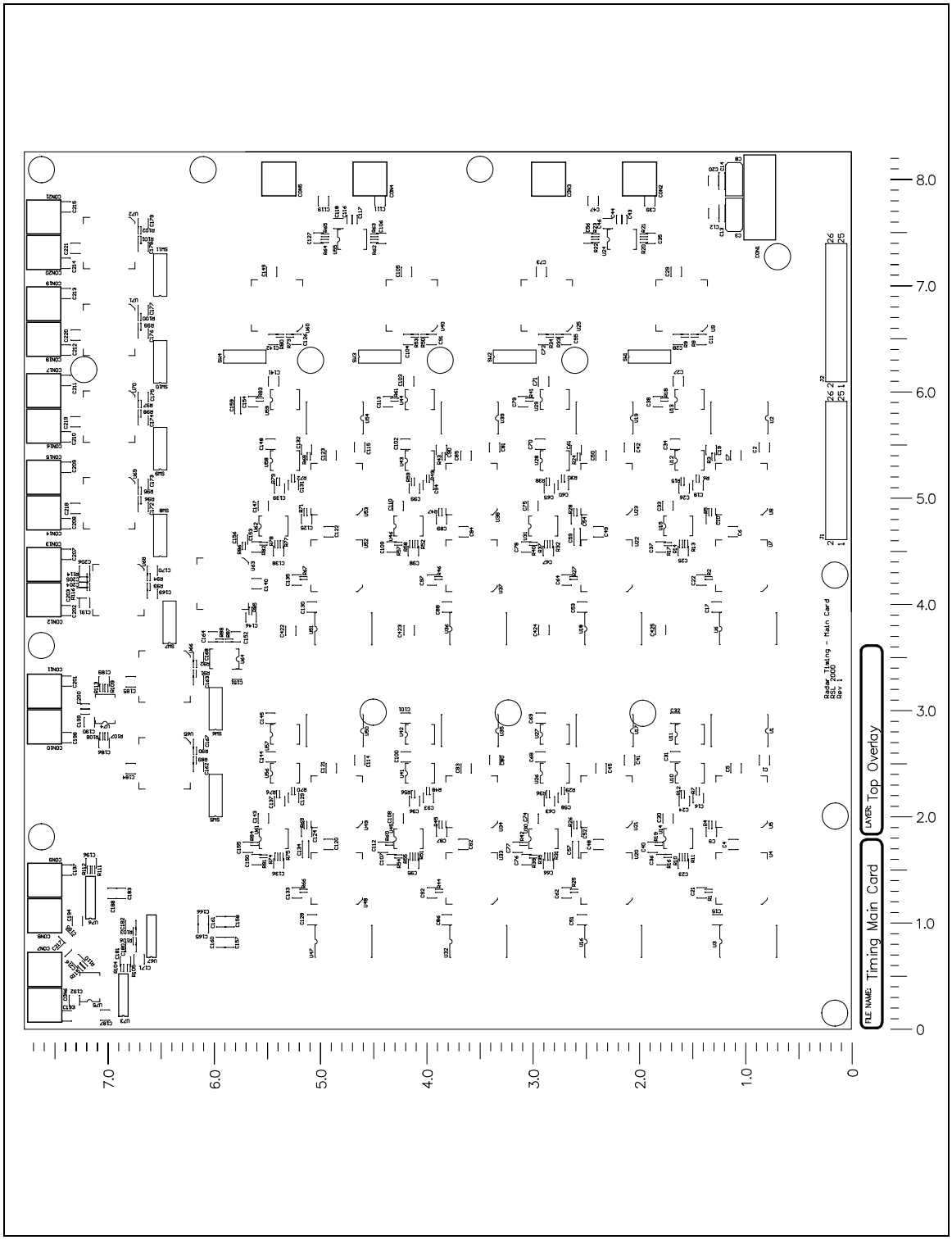

FILE NAME: Mezzanine Card
LAYER: Mid Layer 1
0
1.0
2.0
3.0
4.0
5.0

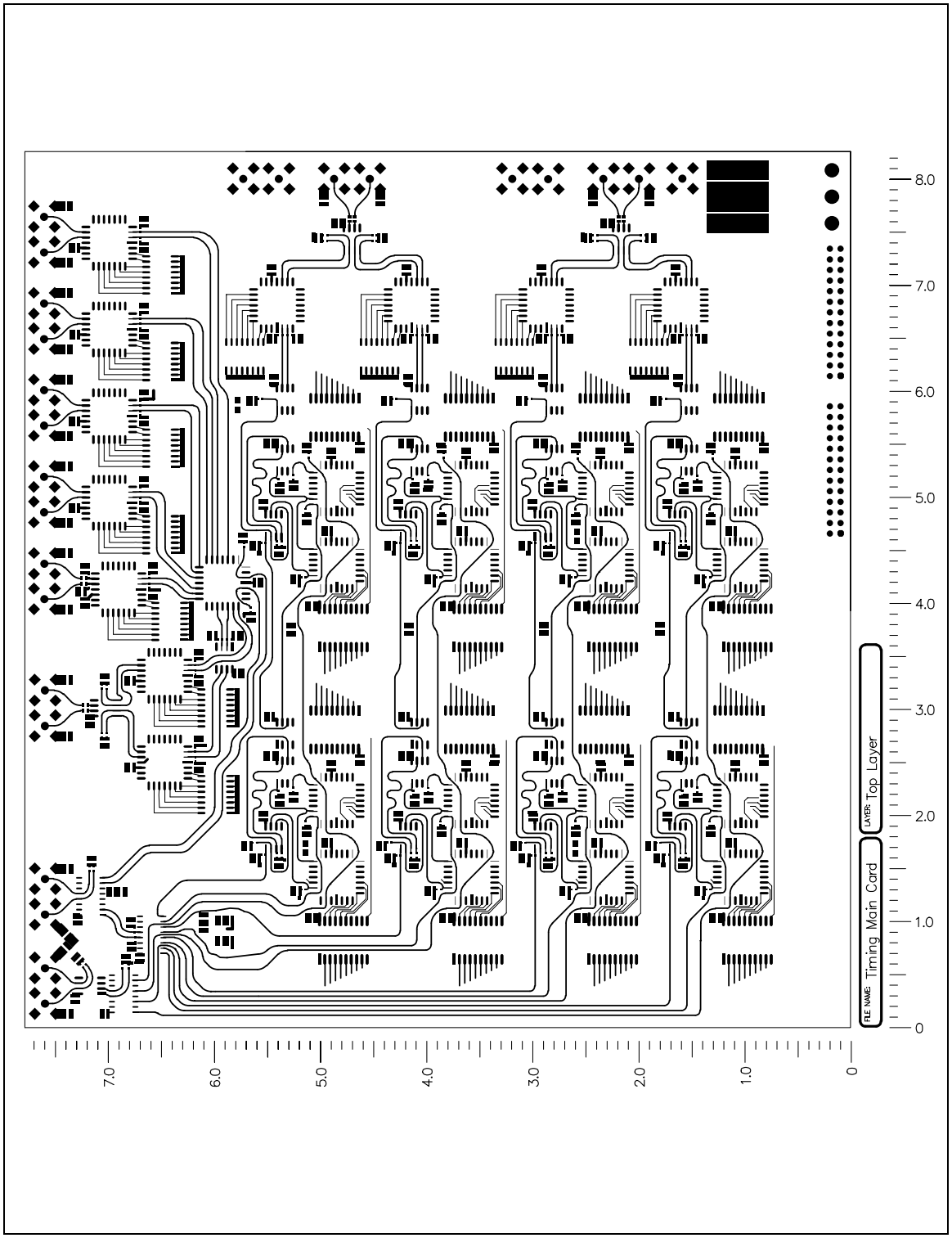

FILE NAME: Mezzanine Card
LAYER: Internal Plane 3
0
1.0
2.0
3.0
4.0
5.0

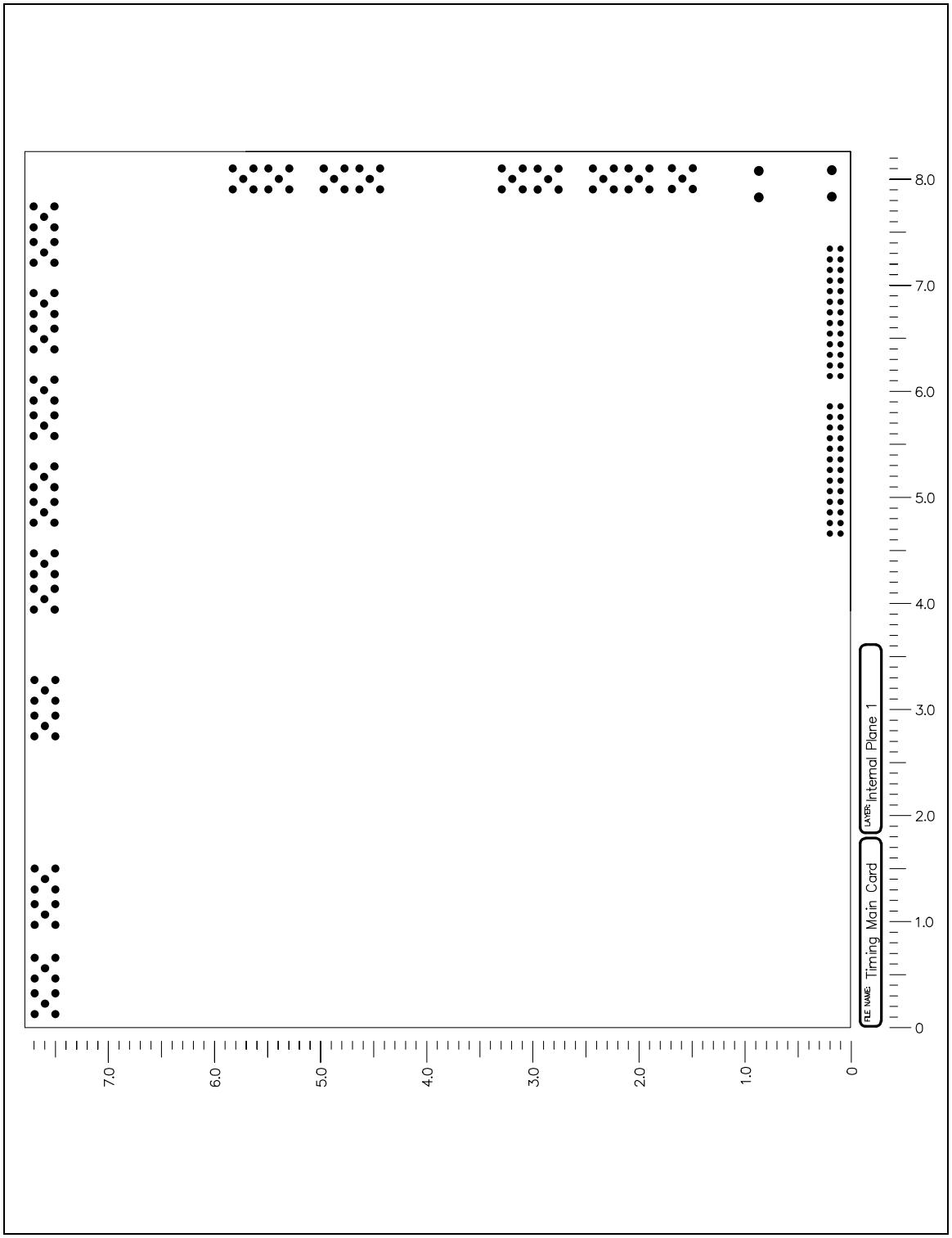

FILE NAME: Mezzanine Card
LAYER: Bottom Layer
0
1.0
2.0
3.0
4.0
5.0

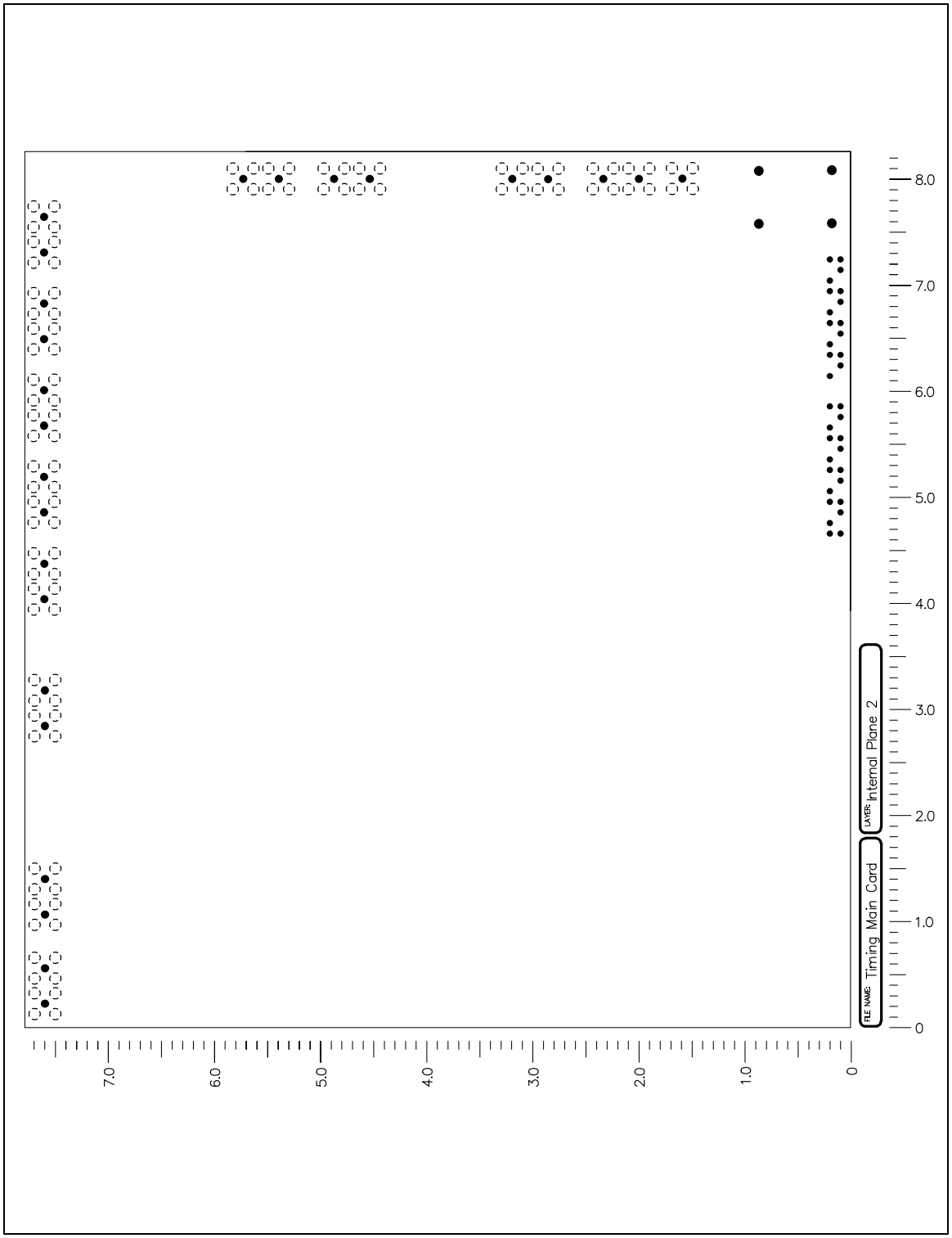

FILE NAME: Mezzanine Card
LAYER: Bottom Overlay
0
1.0
2.0
3.0
4.0
5.0

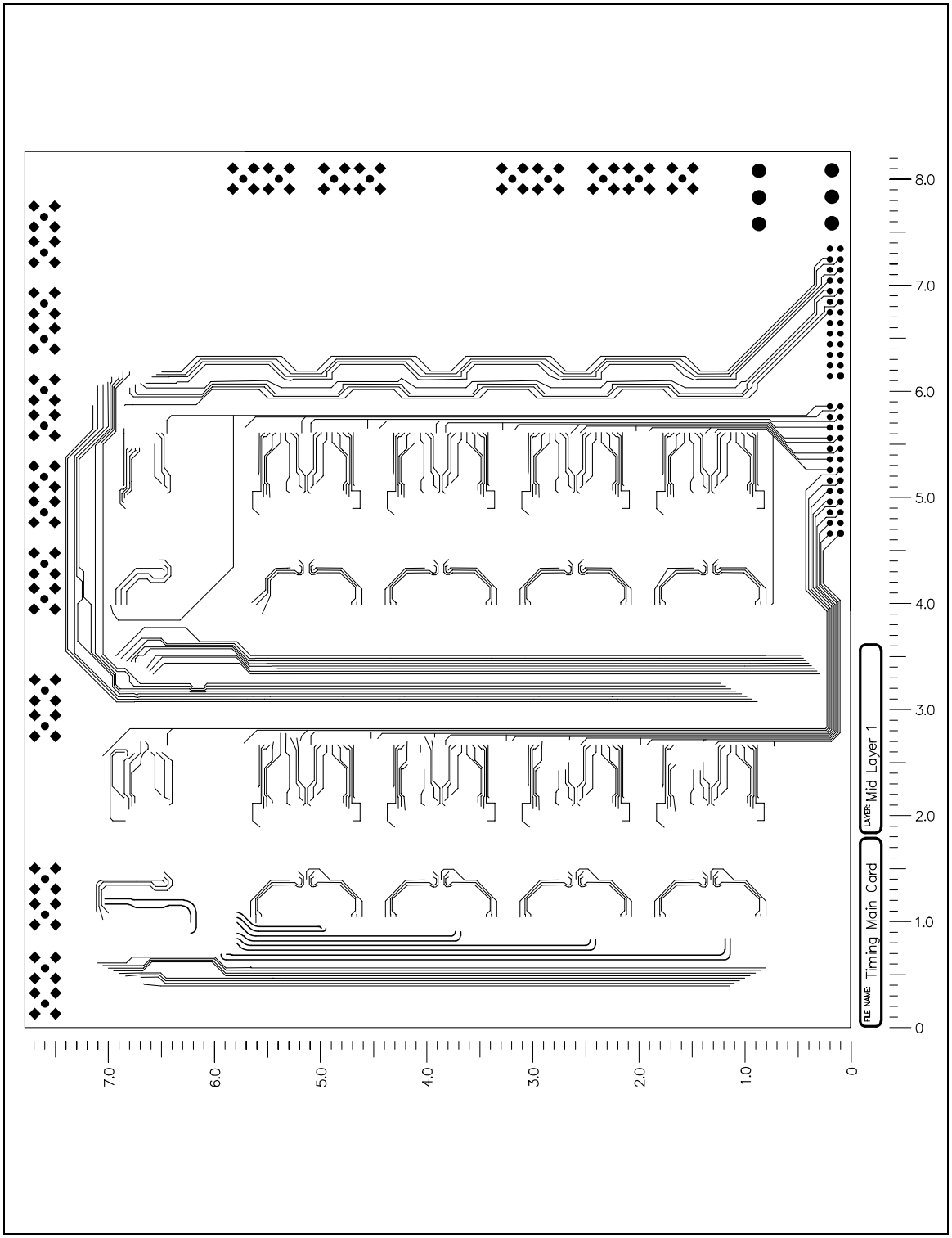
APPENDIX J TIMING MAIN CARD PCB LAYOUT

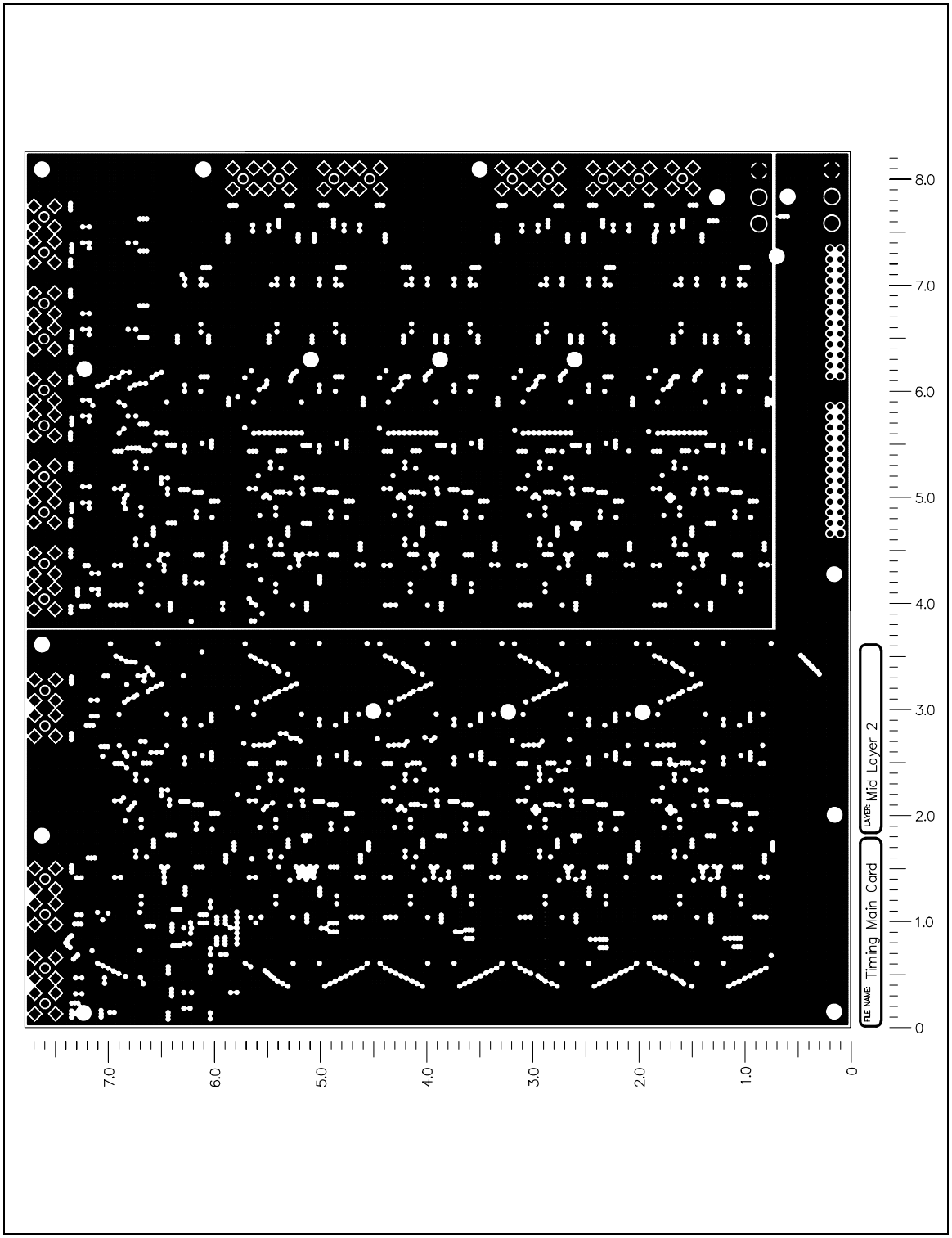

FILE NAME: Timing Main Card
LAYER: Top Overlay
Radar Timing - Main Card
RSL 2000
Rev 1
0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0

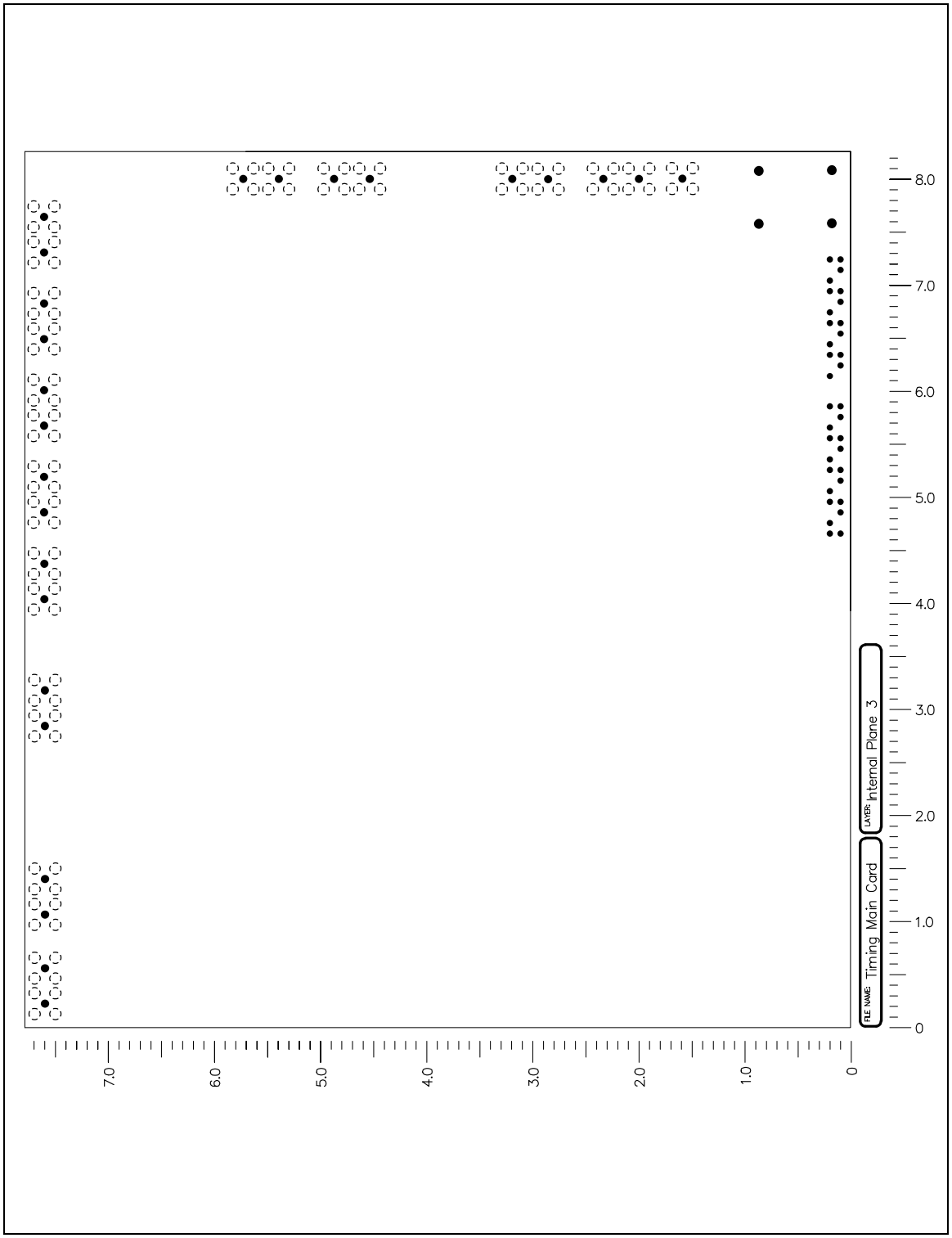

FILE NAME: Timing Main Card
LAYER: Top Layer
8.0
7.0
6.0
5.0
4.0
3.0
2.0
1.0
0

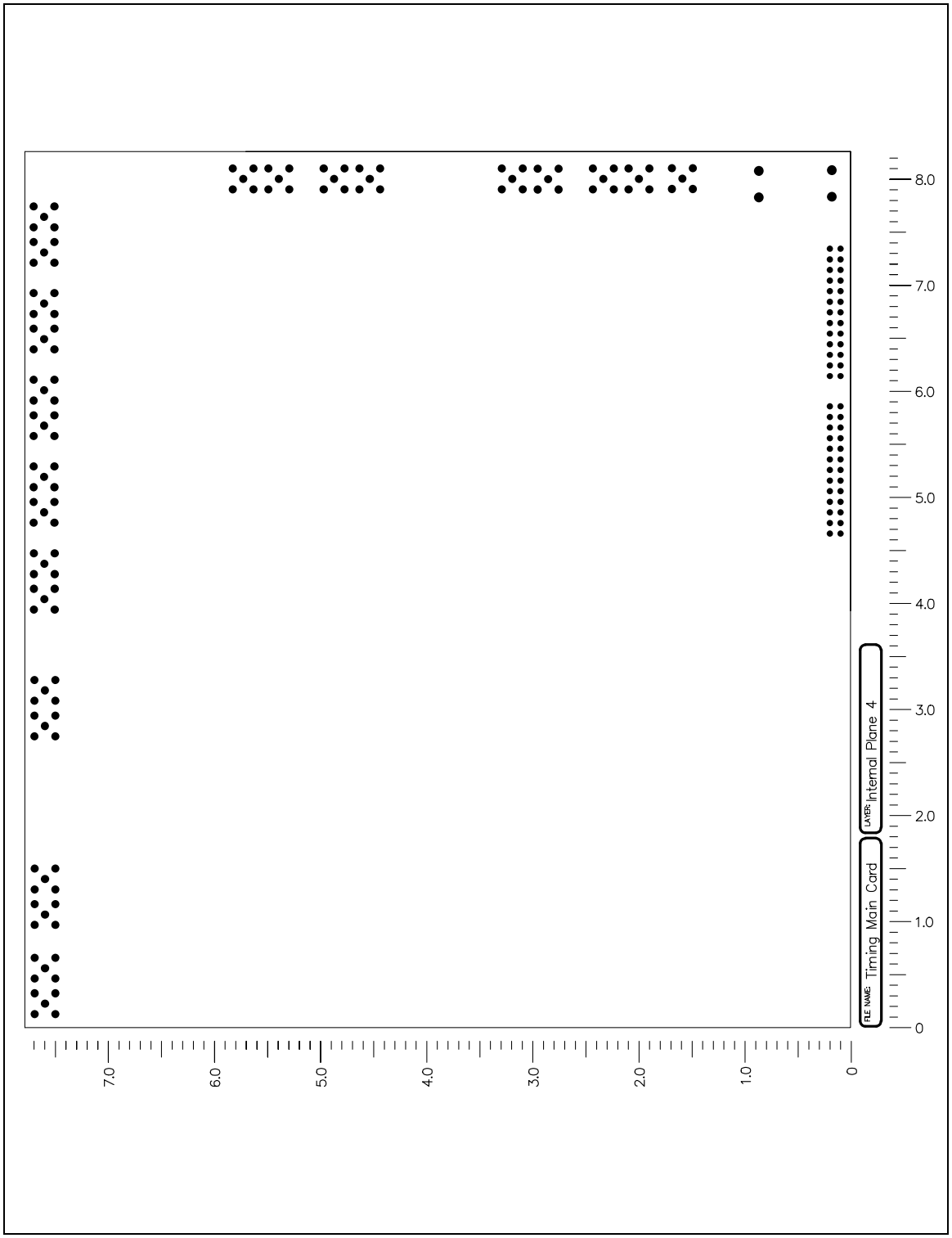

FILE NAME: Timing Main Card
LAYER: Internal Plane 1
0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0

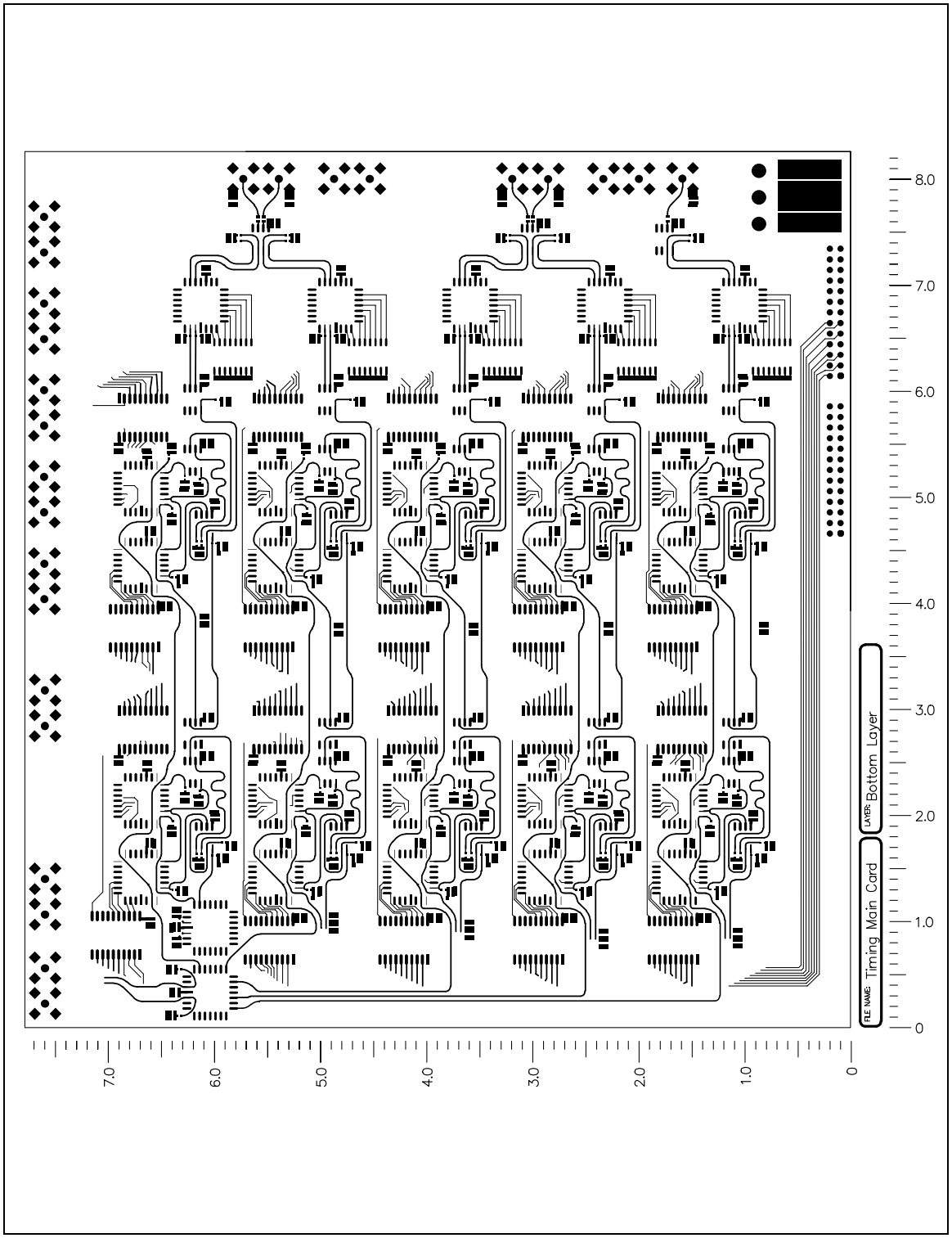

FILE NAME: Timing Main Card
LAYER: Internal Plane 2
0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0

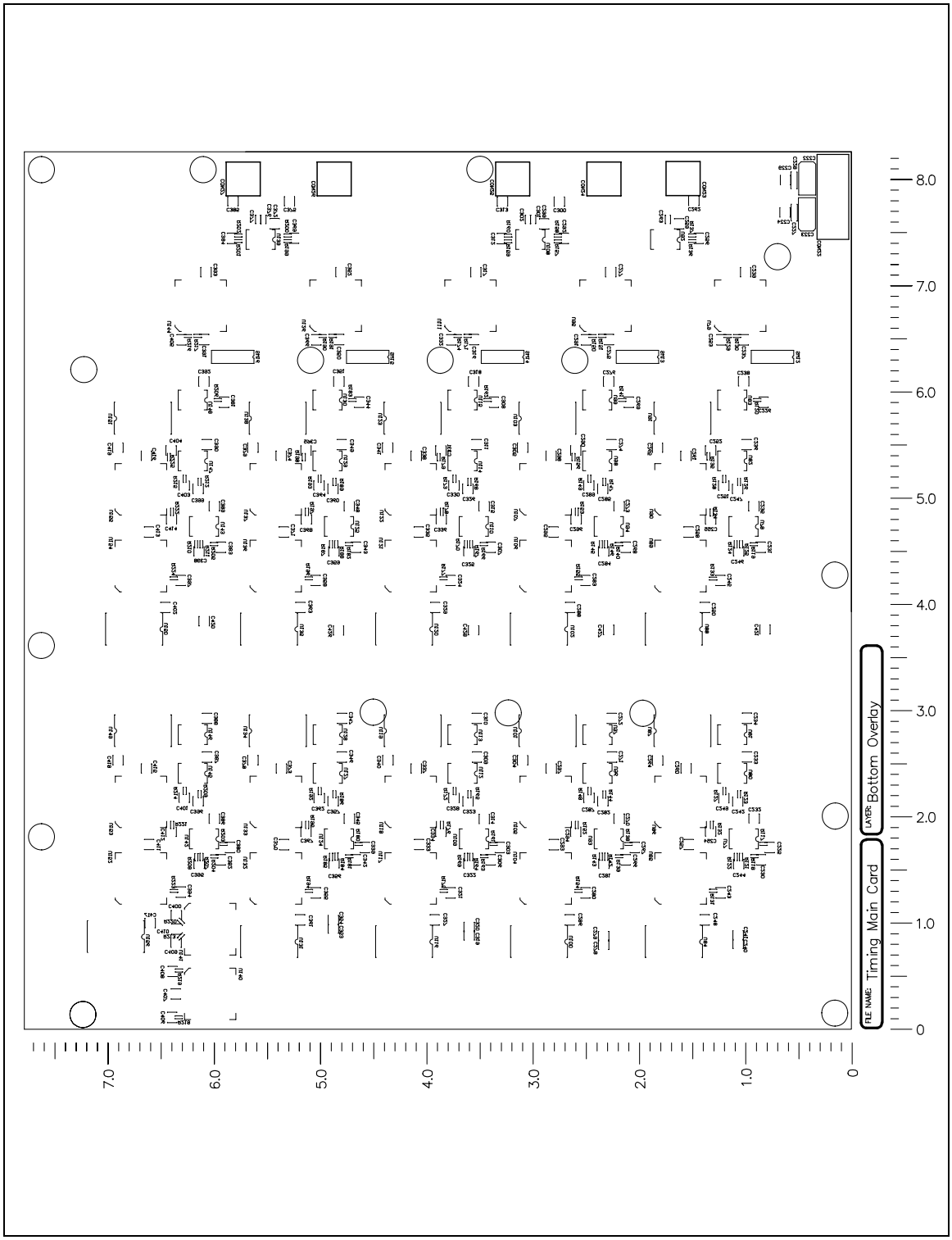

FILE NAME: Timing Main Card
LAYER: Mid Layer 1
0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0


FILE NAME: Timing Main Card
LAYER: Mid Layer 2
0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0

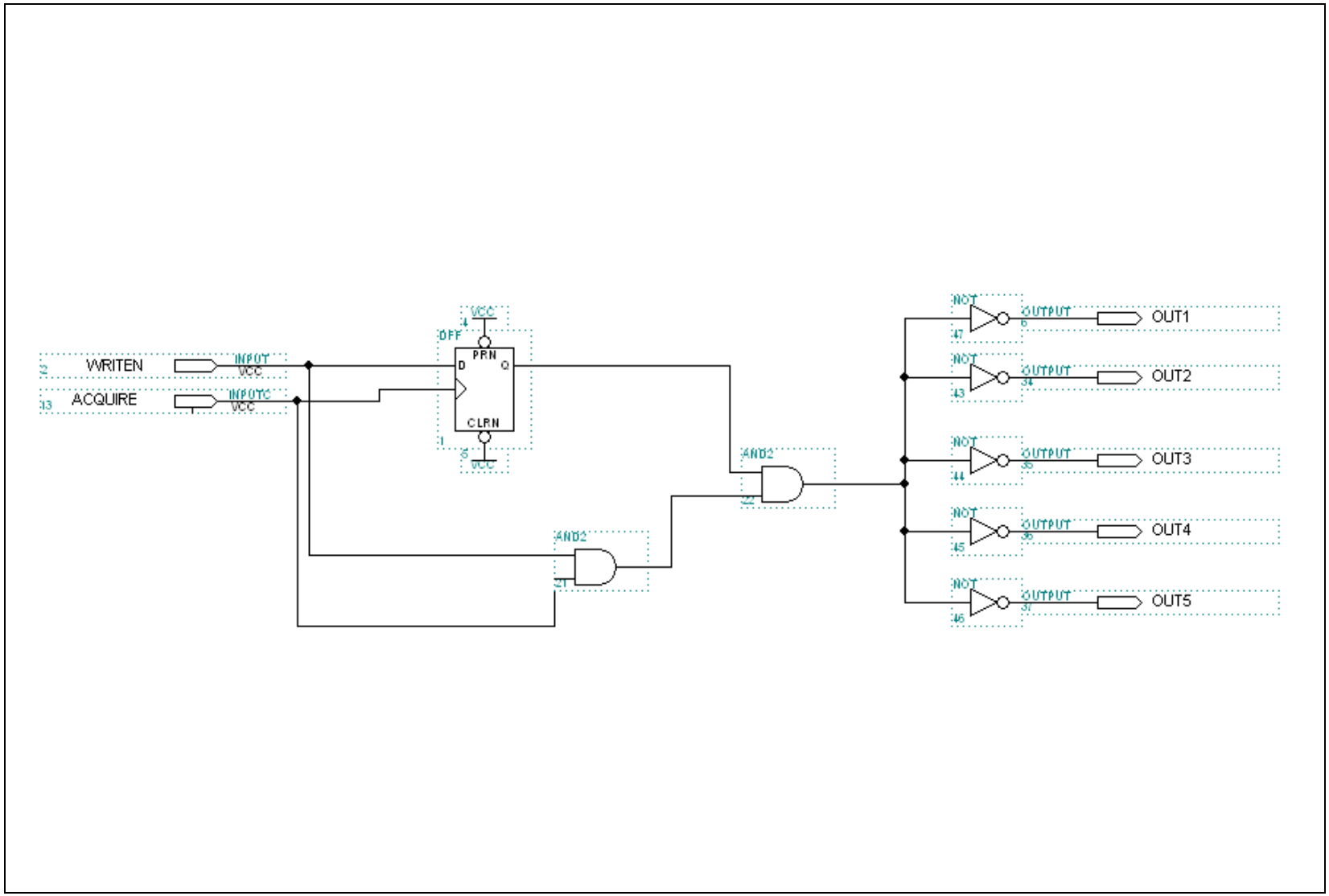

FILE NAME: Timing Main Card
LAYER: Internal Plane 3
0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0


FILE NAME: Timing Main Card
LAYER: Internal Plane 4
0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0

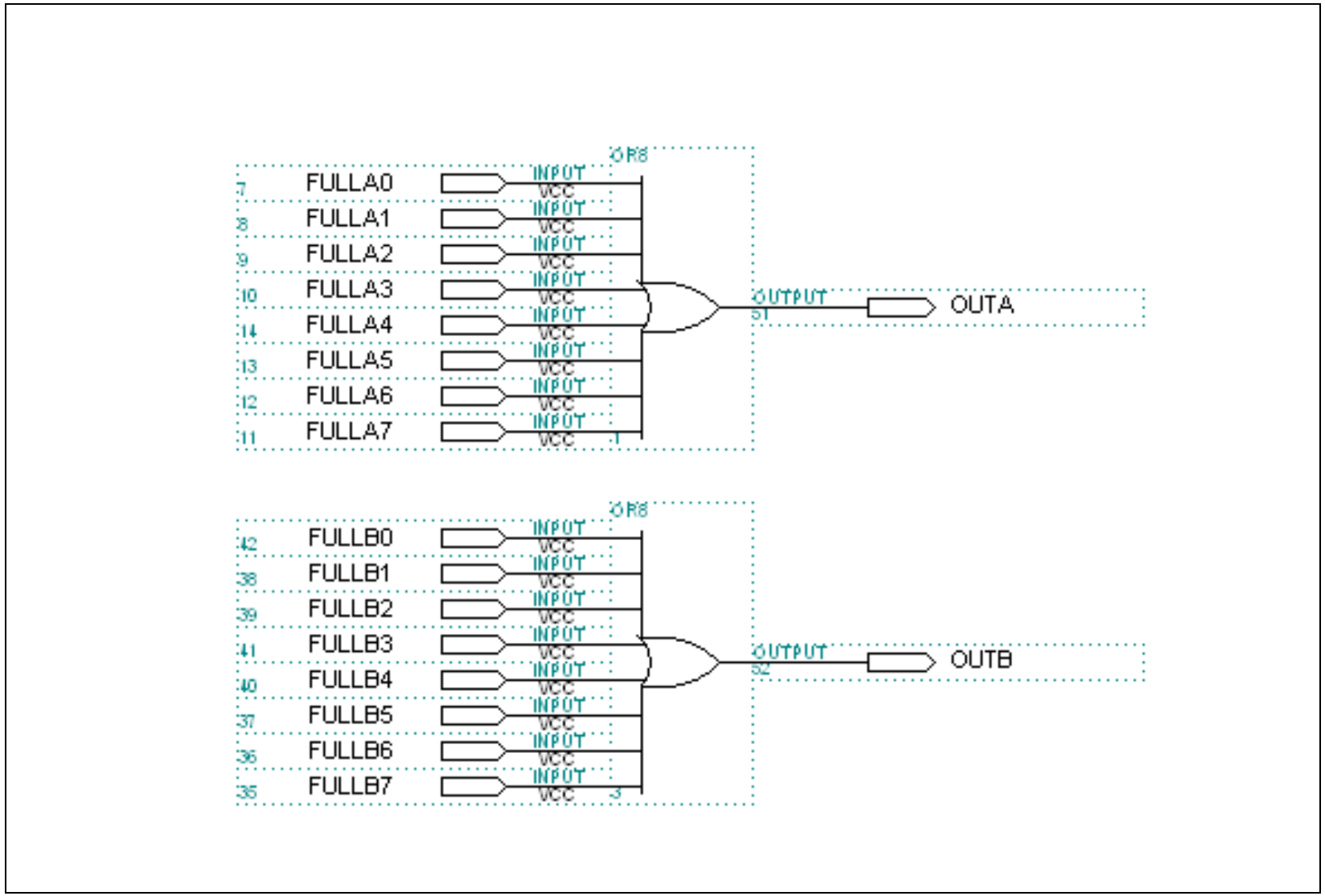

FILE NAME: Timing Main Card
LAYER: Bottom Layer
8.0
7.0
6.0
5.0
4.0
3.0
2.0
1.0
0
7.0
6.0
5.0
4.0
3.0
2.0
1.0
0


FILE NAME: Timing Main Card
LAYER: Bottom Overlay
0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0

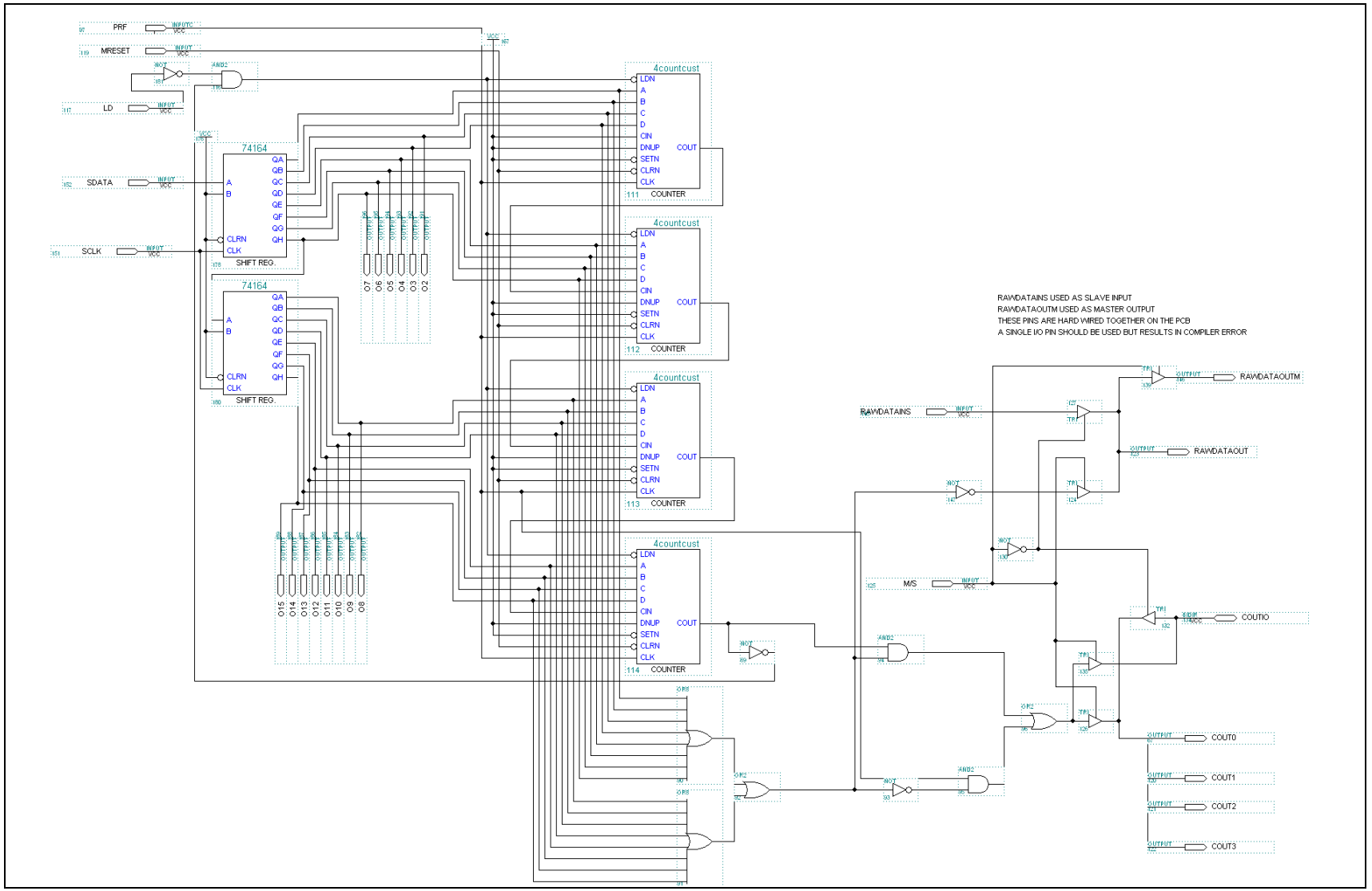
APPENDIX K PLD PROGRAMS



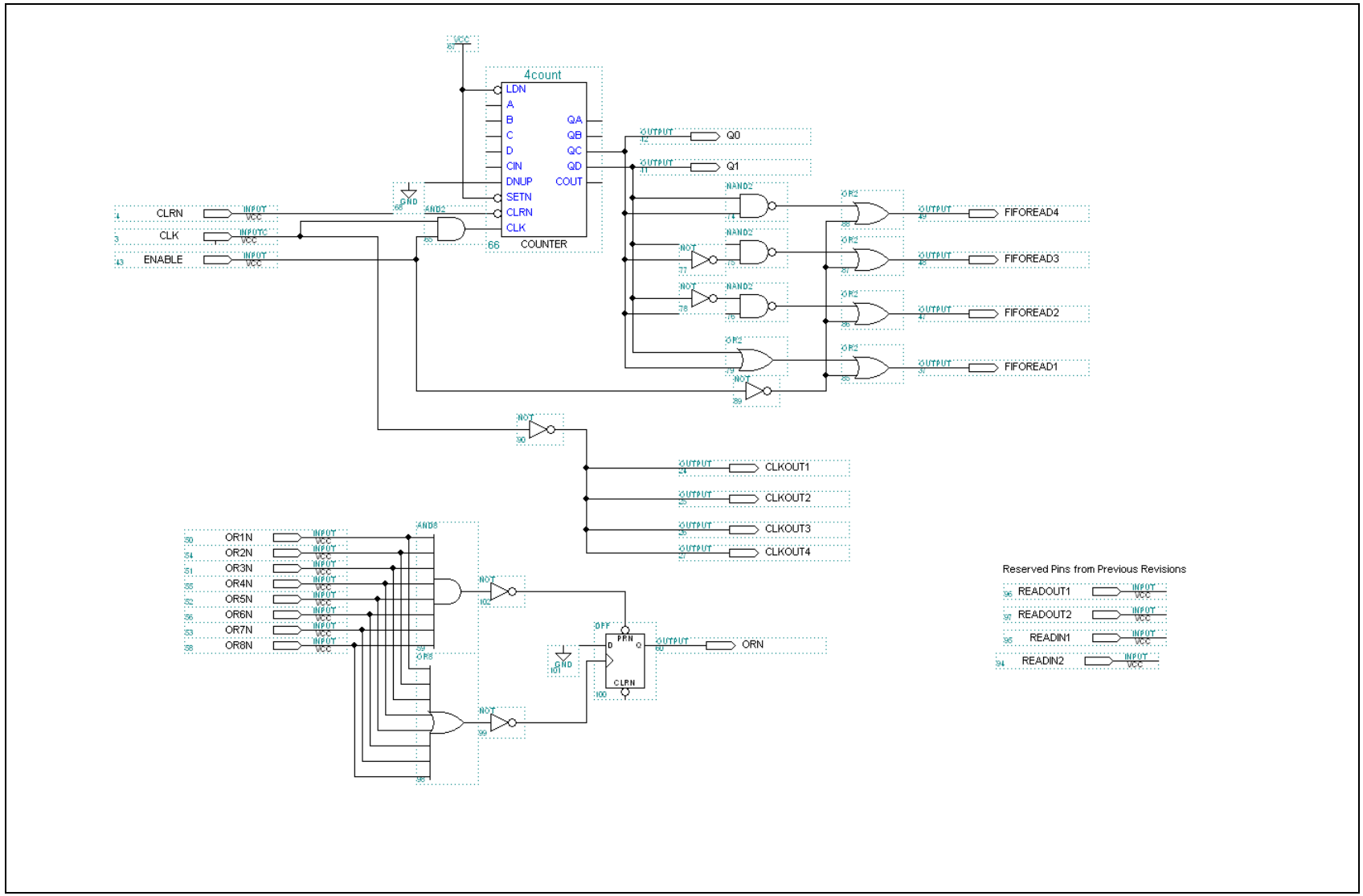
Averaging Card - Acquire Controller



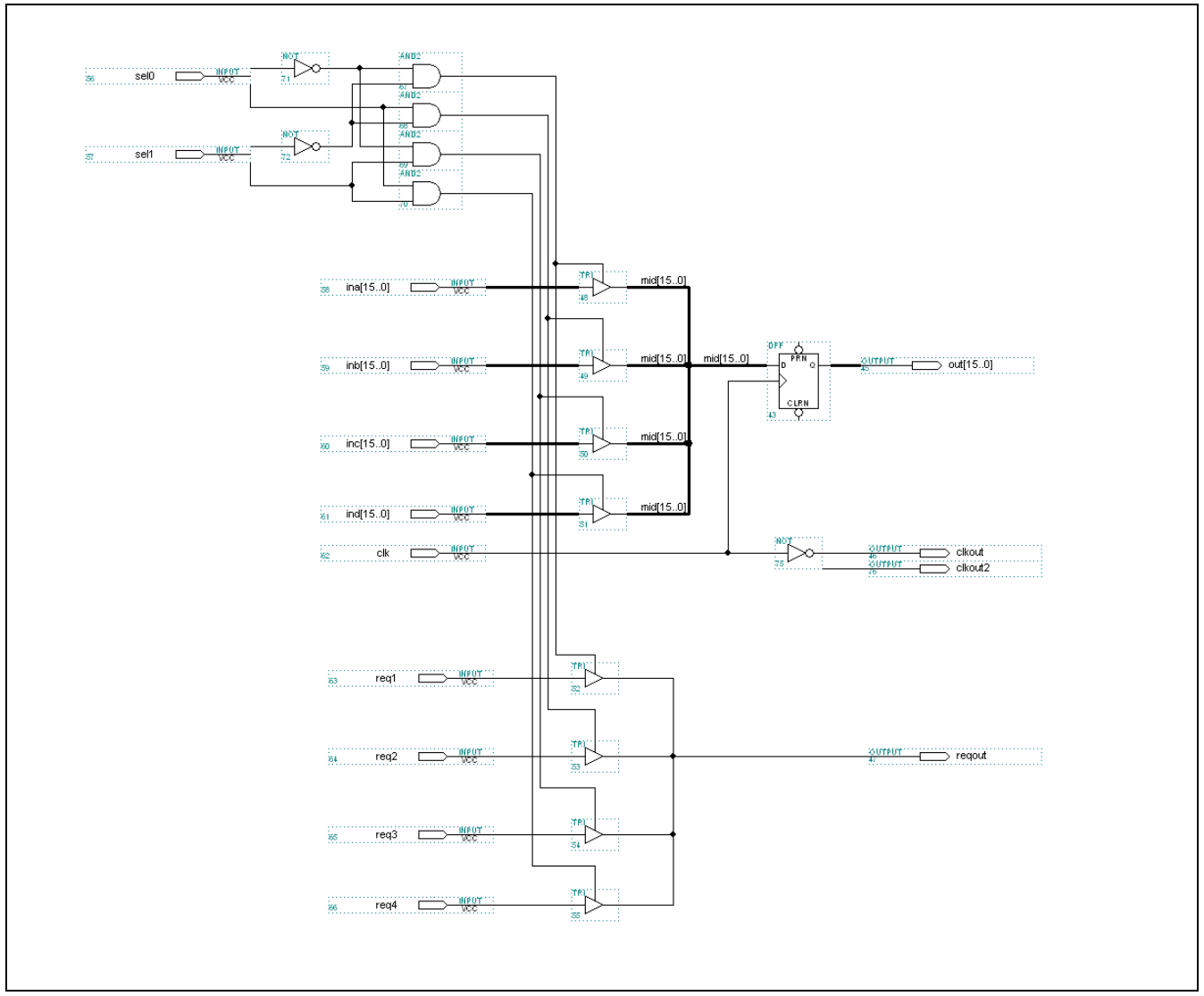
Averaging Card - Error Detector



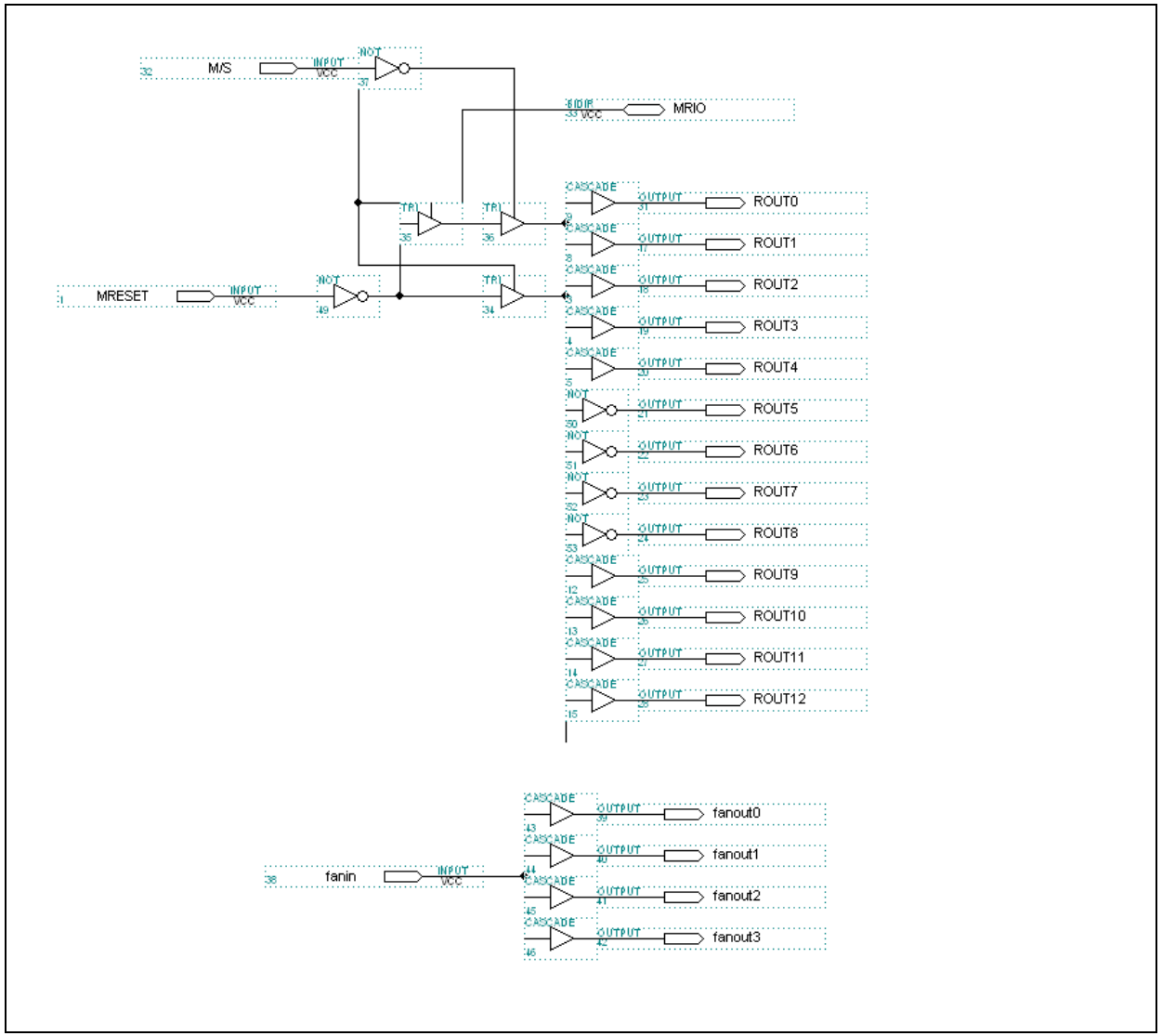
Averaging Card - Integration Counter



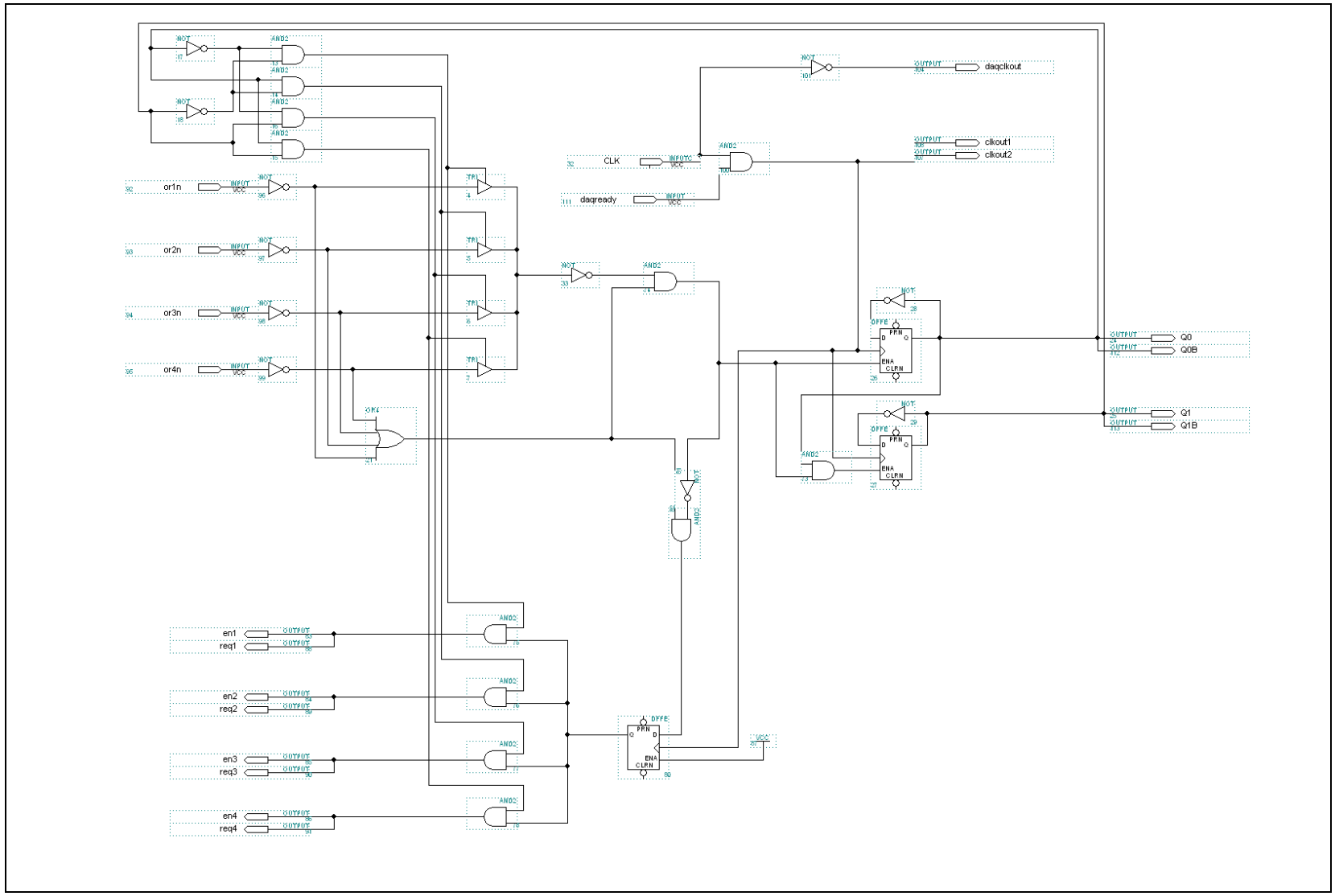
Averaging Card - MUX Controller



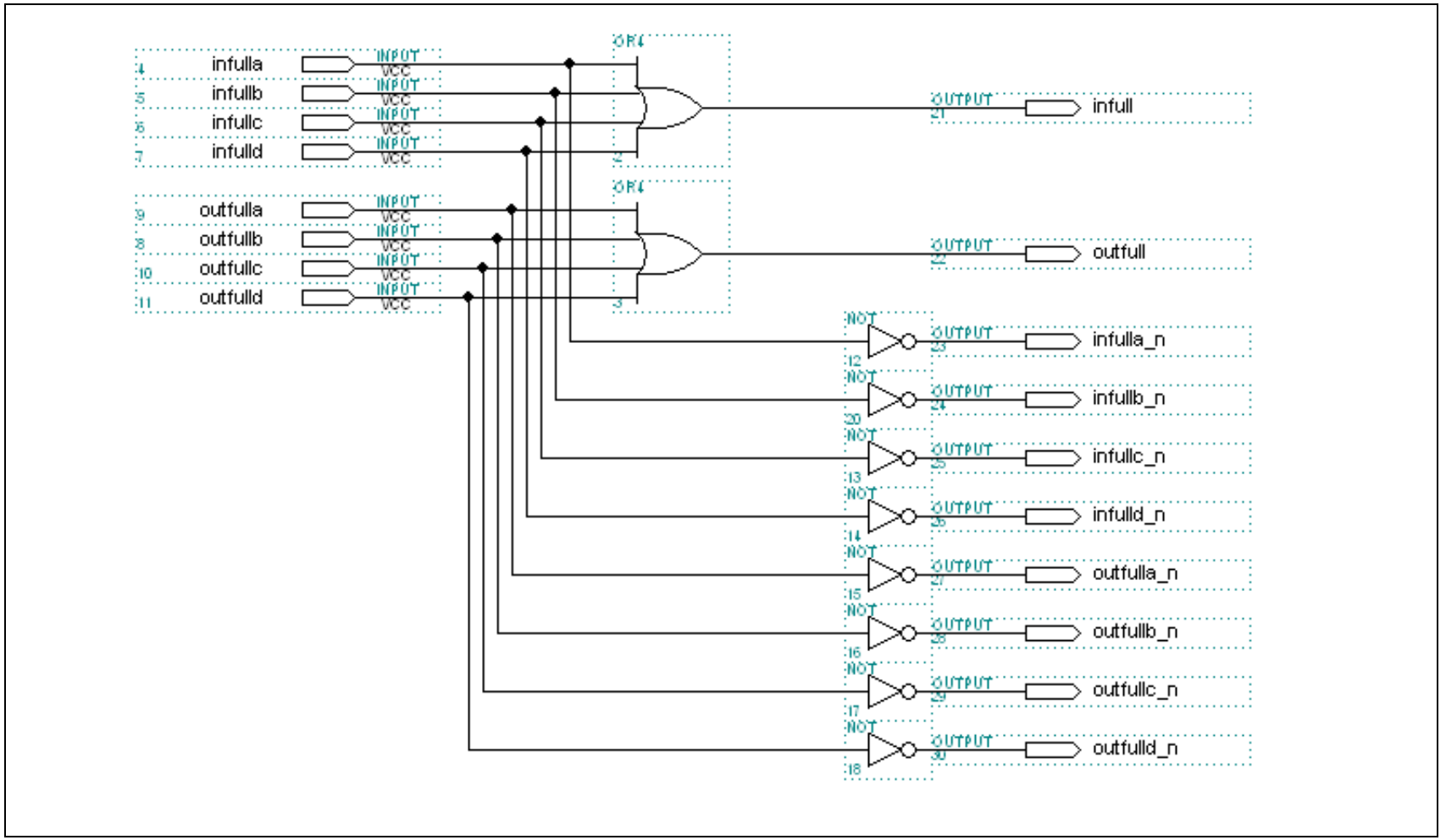
Averaging Card/MUX Card - 64:16 Multiplexer



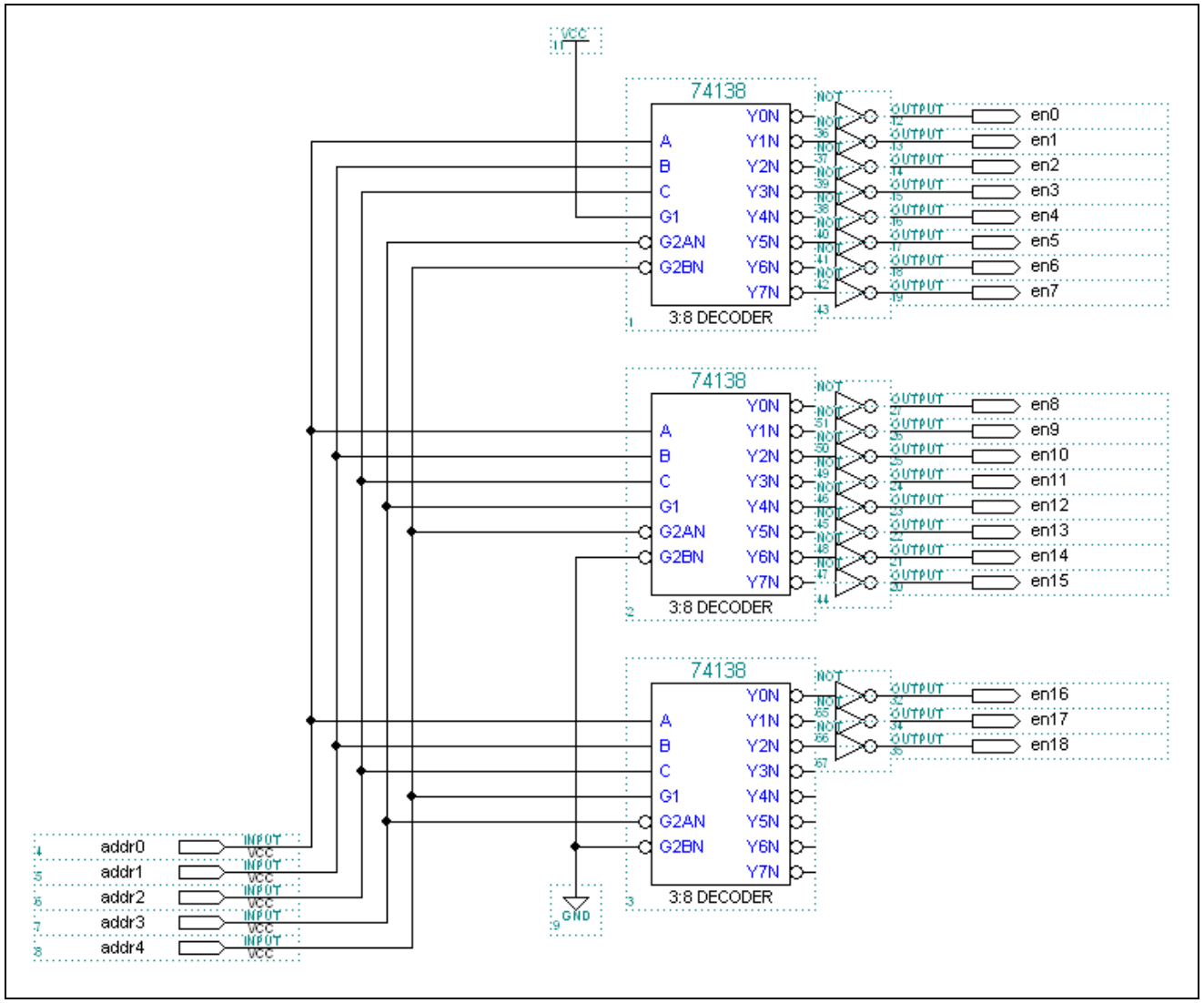
Averaging Card - Reset Controller



MUX Card - MUX Controller



MUX Card – Error Detector



Mezzanine Card – Address Decoder

APPENDIX L AVERAGING CARD DSP CODE

Averaging Card Program

```
        .title  "Average Card DSP Program"

        .data
EMIF_GLOB     .equ    0x01800000      ;address of EMIF global control reg.
EMIF_GLOB_32  .equ    0x000000FE      ;global control reg value
EMIF_CE0      .equ    0x01800008      ;address of EMIF CE0 control reg.
EMIF_CE0_32   .equ    0x00000020      ;CE0 control value (FIFO)
EMIF_CE1      .equ    0x01800004      ;address of EMIF CE1 control reg.
EMIF_CE1_32   .equ    0xFFFFFF0F      ;CE1 control value (Flash Memor
EMIF_CE2      .equ    0x01800010      ;address of EMIF CE2 control reg.
EMIF_CE2_32   .equ    0x00000020      ;CE2 control value (FIFO)
XBUS_GCR      .equ    0x01880000      ;address of XBUS global control reg.
XBUS_GCR_32   .equ    0x00007000      ;XBUS GCR mask value
XBUS_XCE0     .equ    0x01880008      ;address of XBUS CE0 control reg.
XBUS_XCE0_32  .equ    0x00000000      ;XBUS CE0 value
DMA_PC        .equ    0x01840000      ;DMA Primary Control reg.
DMA_SOURCE    .equ    0x01840010      ;DMA Source reg.
DMA_DEST      .equ    0x01840018      ;DMA destination reg.
DMA_COUNT     .equ    0x01840020      ;DMA counter reg.
TIMER0_REG    .equ    0x01940000      ;Timer 0 reg.
INT_POL       .equ    0x019C0008      ;Interrupt polarity reg.

INFIFO_ADDR   .equ    0x00400000      ;start address of input fifo on CE0
FLASH_ADDR    .equ    0x01400000      ;start address of flash memory on CE1
CE2_ADDR      .equ    0x02000000      ;start address of CE2
DATA_ADDR     .equ    0x80000000      ;address where data will be stored
FIFO_SIZE     .equ    32768           ;depth of input fifo X width in bytes
                                      ;AKA the maximum amount of 32 bit memory space
                                      ;needed to store all contents of the FIFO

        .text

;***********************************************************
;
;Setup Interrupt Service Table at address 0000h
;
;***********************************************************

RESET:                                ; reset vector
        b      main                   ; start branch to main code
        nop
        nop
        nop
        nop
        nop
        nop
        nop

NMI_ISFP:
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop

RESERVED_1:
        nop
        nop
```

```
        nop
        nop
        nop
        nop
        nop
        nop

RESERVED_2:
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop

INT_4:
        mvk    1,A1                ;set flag that averaging is complete
        b      IRP                 ;return from interrupt
        nop    4
        nop
        nop
        nop
        nop
        nop

INT_5:                             ;Interrupt for input FIFO output ready
        mvk    1,B3                ;set flag that FIFO has data
        b      IRP                 ;return from interrupt
        nop    4
        nop
        nop
        nop
        nop
        nop

INT_6:                             ;Interrupt for input FIFO empty
        mvk    0,B3                ;set flag that FIFO is empty
        b      IRP                 ;return from interrupt
        nop
        nop
        nop
        nop
        nop
        nop

main:
       ;Set EMIF global register
       mvkl    EMIF_GLOB,A4        ;EMIF_GLOB address
       mvkl    EMIF_GLOB_32,B4     ;EMIF_GLOB value
       mvkh    EMIF_GLOB,A4
       mvkh    EMIF_GLOB_32,B4
       stw     B4,*A4              ;write value to register

       ;Setup EMIF CE0 for input FIFO
       mvkl    EMIF_CE0,A4         ;EMIF_CE0 address
       mvkl    EMIF_CE0_32,B4      ;EMIF_CE0 value
       mvkh    EMIF_CE0,A4
       mvkh    EMIF_CE0_32,B4
       stw     B4,*A4              ;write value to register

       ;Setup EMIF CE1 for flash memory
       mvkl    EMIF_CE1,A4         ;EMIF_CE1 address
       mvkl    EMIF_CE1_32,B4      ;EMIF_CE1 value
       mvkh    EMIF_CE1,A4
       mvkh    EMIF_CE1_32,B4
       stw     B4,*A4

       ;Setup EMIF CE2 for fast access
       mvkl    EMIF_CE2,A4         ;EMIF_CE2 address
```

```
        mvkl    EMIF_CE2_32,B4         ;EMIF_CE2 value
        mvkh    EMIF_CE2,A4
        mvkh    EMIF_CE2_32,B4
        stw     B4,*A4                 ;write value to register

        ;Setup XBUS global control register
        mvkl    XBUS_GCR,A4            ;XBUS_GCR address
        mvkl    XBUS_GCR_32,B4         ;XBUS_GCR mask
        mvkh    XBUS_GCR,A4
        mvkh    XBUS_GCR_32,B4
        ldw     *A4,B5
        nop     5
        or      B4,B5,B5
        stw     B5,*A4

        ;Setup XBUS_CE0 for output fifo memory
        mvkl    XBUS_XCE0,A4           ;XBUS_CE0 address
        mvkl    XBUS_XCE0_32,B4        ;XBUS_CE0 value
        mvkh    XBUS_XCE0,A4
        mvkh    XBUS_XCE0_32,B4
        stw     B4,*A4

        ;preparing to clear the data memory
        mvkl    0x80000000,A3          ;Data start address
        mvkh    0x80000000,A3
        zero    A4
        zero    B6
        mvkl    FIFO_SIZE,B7           ;max memory length to clear
        mvkh    FIFO_SIZE,B7

clrmem:
        stw     A4,*A3++
        add     4,B6,B6
        cmpgt   B6,B7,B0
 [!B0]  b       clrmem
        nop     5
        nop
        ;memory clear complete

        ;put setup data/fifo addresses
        mvkl    INFIFO_ADDR,A3
        mvkh    INFIFO_ADDR,A3
        mvkl    DATA_ADDR,A8
        mvkh    DATA_ADDR,A8
        mvk      0x04,A5
        mvk     0x01,B5
        mvkl    0x80000000,A6
        mvkh    0x80000000,A6

        ;set up 8 bit masks
        mvkl    0x000000FF,A12
        mvkh    0x000000FF,A12
        mvkl    0x0000FF00,A13
        mvkh    0x0000FF00,A13
        mvkl    0x00FF0000,B12
        mvkh    0x00FF0000,B12
        mvkl    0xFF000000,B13
        mvkh    0xFF000000,B13

        ;set interrupt polarities
        mvkl    INT_POL,A4             ;interrupt polarity register address
        mvk     0x02,B4                ;interrupt polarity mask
        mvkh    INT_POL,A4
        stw     B4,*A4

        ;enable interrupt masks
        MVK     0x0073,B0              ;interrupt enable mask
        MVC     B0,IER                 ;write mask to IER

        ;enable interrupts globally
        MVC     CSR,B0
```

```
        OR      1,B0,B0
        MVC     B0,CSR

        ;Enable TOUT0 so hardware knows DSP is ready for acquisition
        mvkl    TIMER0_REG,A4
   ||   mvkl    0x00000004,B4
        mvkh    TIMER0_REG,A4
   ||   mvkh    0x00000004,B4
        stw     B4,*A4

;*********************************************************************
;Purpose:  Inifinite loop, toggling input FIFO clock line.  When FIFO
;          has data, it will trigger an interrupt to break out of this loop.
;
;*********************************************************************

        mvkl    CE2_ADDR,A0           ;EMIF CE2 start address
        mvkh    CE2_ADDR,A0
        zero    B3                    ;initial condition is to loop
        zero    A1                    ;initial condition is to average

infinite:
        ;if B3 is 1, fifo has data, else keep looping
        cmpeq   1,B3,B0
 [B0]   b       acquire_data          ;jump out of loop only if there is data
        nop     5

        ;else toggle FIFO clock, check variable, and loop again
        b       infinite
        ldw     *A0,A2
        nop     4
        nop
        nop
        nop
        nop


;*********************************************************************
;Purpose:  This loop performs the data processing.  It loops through,
;          getting 4 bytes from the input FIFO,
;          unpacking, averaging the new data with the old, and storing
;          the new results data to memory.
;          Register A0 contains the loop requirement which will be
;          updated by an interrupt routine when the fifo is empty.
;          After the FIFO is empty, the data is DMA transfered straight
;          to the output FIFO if register ?? has been updated
;          by Interrupt 5 (averaging counter), otherwise the data
;          in memory will be left for the next acquisition.
;
;*********************************************************************

acquire_data:
        mvkl    DATA_ADDR,A8
        mvkh    DATA_ADDR,A8

acquire_data_loop:
        ldw     *A3, A2               ;grab  4 bytes from fifo
        ldw     *+A8[0],A10           ;get old data byte 1
        ldw     *+A8[1],A11           ;get old data byte 2
        ldw     *+A8[2],B10           ;get old data byte 3
        ldw     *+A8[3],B11           ;get old data byte 4
        nop

        mv      A2,B2;                ;copy data into B register

        and     A2,A12,A14            ;unpack data by masking
   ||   and     A2,A13,A15            ;masked data is R's 14, 15
   ||   and     B2,B12,B14
   ||   and     B2,B13,B15

        shru    A15,8,A15             ;unpack data by shifting
```

```
   ||   shru    B14,16,B14
        shru    B15,24,B15

        add     A10,A14,A10           ;add new and old data byte 1
   ||   add     A11,A15,A11           ;add new and old data byte 2
   ||   add     B10,B14,B10           ;add new and old data byte 3
   ||   add     B11,B15,B11           ;add new and old data byte 4


        cmpeq   0,B3,B0               ;check loop condition (updated by interrupt)
 [!B0]  B       acquire_data_loop
        stw     A10,*A8++;            ;store to memory, increment data address
        stw     A11,*A8++;            ;"
        stw     B10,*A8++;            ;"
        stw     B11,*A8++;            ;"
        nop
        nop

        ;Toggle TOUT0 for testing purposes
        mvkl    TIMER0_REG,A4
   ||   zero    B4
        mvkh    TIMER0_REG,A4
        stw     B4,*A4
   ||   mvkl    0x00000004,B4
        mvkh    0x00000004,B4
        stw     B4,*A4

        ;Back to inifinite loop unless averaging is done
        cmpeq   1,A1,B0               ;check flag updated by interrupt
 [B0]   B       ave_comp              ;if flag set, averaging complete
        nop     5
        B       infinite              ;else go back to infinite loop
        nop     5

ave_comp:
        ;Averaging is complete, now DMA transfer data to the output FIFO on XBUS

        ;Setup DMA source
        mvkl    DMA_SOURCE,A4         ;DMA_SOURCE register address
   ||   mvkl    0x80000000,B4         ;DMA source address
        mvkh    DMA_SOURCE,A4
   ||   mvkh    0x80000000,B4
        stw     B4,*A4

        ;Setup DMA destination
        mvkl    DMA_DEST,A4           ;DMA destination register address
   ||   mvkl    0x40000000,B4         ;DMA destination register value
        mvkh    DMA_DEST,A4
   ||   mvkh    0x40000000,B4
        stw     B4,*A4

        ;Setup DMA counter
        mvkl    DATA_ADDR,A7
   ||   mvkl    DMA_COUNT,B4          ;Get DMA counter address
        mvkh    DATA_ADDR,A7
   ||   mvkh    DMA_COUNT,B4
        sub     A8,A7,A8              ;calculate the number of words to transfer
        shr     A8,2,A8               ;divide by 4, A8 now has number of 32 bit words
        stw     A8,*B4                ;write the number of DMA counts

        ;start DMA transfer
        mvkl    DMA_PC,A4             ;DMA control address
   ||   mvkl    0x01000011,B4         ;start and increment
        mvkh    DMA_PC,A4
   ||   mvkh    0x01000011,B4
        stw     B4,*A4

        ;clear the memory for a new acquisition
        mvkl    DATA_ADDR,A8          ;Data start address
   ||   mvkl    FIFO_SIZE,B7          ;memory length to clear
        mvkh    DATA_ADDR,A8
```

```
   ||   mvkh    FIFO_SIZE,B7
   ||   zero    A4
   ||   zero  B6

clrmem2:
        stw     A4,*A8++
        cmpgt   B6,B7,B0
 [!B0]  b       clrmem2
        add     4,B6,B6
        nop     4

        ;memory clear complete, ready to begin a new acquisition
        zero    A1                      ;clear flag to start new average

        ;Toggle TOUT0 for testing purposes
        mvkl    TIMER0_REG,A4
   ||   zero    B4
        mvkh    TIMER0_REG,A4
        stw     B4,*A4
   ||   mvkl    0x00000004,B4
        mvkh    0x00000004,B4
        stw     B4,*A4

        b       infinite                ;processing complete
        nop     5
        nop
        nop
```

Flash Write Program

```
;   Flash bootup utility for 6202 average card
;   Original Code from the TI 6211 evaluation kit.
;   Modified by Ryan Eakin, Jan-2001

        .option D,T
        .length 102
        .width  140


PAGE_SIZE      .equ    0x80          ;flash page size in byte
FLASH_START    .equ    0x01400000    ;flash start address

CODE_SIZE      .equ    0x500         ;application code size in byte
CODE_START     .equ    0x80000000    ;application code start address

FLASH_REG1     .equ    0x01415554    ;address of the flash control reg 1
FLASH_REG2     .equ    0x0140AAA8    ;address of the flash control reg 2
FLASH_KEY1     .equ    0xAA
FLASH_KEY2     .equ    0x55
FLASH_KEY3     .equ    0xA0
EMIF_GCR       .equ    0x01800000    ;EMIF global control
EMIF_CE1       .equ    0x01800004    ;address of EMIF CE1 control reg.
EMIF_CE1_8     .equ    0xffffff03    ;CE1 register value for Flash memory
TIMER0_REG     .equ    0x01940000    ;Timer 0 register


        .def  _flash_prog
        .sect ".boot_load"



_flash_prog:

        mvkl  EMIF_GCR,A4             ;EMIF_GCR address ->A4
   ||   mvkl  0x3300,B4

        mvkh  EMIF_GCR,A4
   ||   mvkh  0x3300,B4

        stw   B4,*A4

        mvk   0x100,B0
   ||   mvk   1,A1

        mvc     B0,CSR                ;CSR = 0x100

        mvc   A1,IER                  ;IER = 1

        mvk   -1,B0
   ||   mvk   0,A1                    ;couter = 0

        mvc   B0,ICR                  ;ICR = 0xffff
   ||   mvk   0,A0                    ;checksum =0

        mvkl  EMIF_CE1,A4             ;EMIF_CE1 register address ->A4
   ||   mvkl  EMIF_CE1_8,B4           ;CE1 register value

        mvkh  EMIF_CE1,A4
   ||   mvkh  EMIF_CE1_8,B4

        stw   B4,*A4
        mvk   0xff,B6

        mvkl  FLASH_START,B4          ;flash start address -> B4
        mvkh  FLASH_START,B4

        mvkl  CODE_START,A4
   ||   mvkl  CODE_SIZE,B2
```

```
        mvkh  CODE_START,A4
   ||   mvkh  CODE_SIZE,B2

        mvkl  PAGE_SIZE,A6
        mvkh  PAGE_SIZE,A6

        zero  A1

_flash_prog_loop1:
        b     _flash_page_prog          ;program a 128 Byte page
        mvkl  _flash_prog_branch1,B3    ;subroutine return address
        mvkh  _flash_prog_branch1,B3
        add   A6,A1,A1
        cmplt A1,B2,B0
        nop

;repeat prog loop1 until all pages have been written
;as indicated by B0
_flash_prog_branch1:
 [B0]   b     _flash_prog_loop1
        nop   5

        ;all pages have been written, perform success routine
        mvkl  _flash_prog_success,B3
        mvkh  _flash_prog_success,B3
        b     B3
        nop   6


;*****************************************************************************
;* FUNCTION NAME: _flash_page_prog                                           *
;*                                                                           *
;*   Regs Modified     : A3,A4,A6,B1,B4,B5,B6                                *
;*****************************************************************************
_flash_page_prog:
        ;unlock the flash memory
        MVKL    .S2     FLASH_REG1,B6        ;

        MVK     .S1     FLASH_KEY1,A3        ;
   ||   MVKH    .S2     FLASH_REG1,B6        ;

        STB     .D2T1   A3,*B6
        nop             9                    ;
        MVKL    .S2     FLASH_REG2,B6        ;

        MVK     .S1     FLASH_KEY2,A3        ;
   ||   MVKH    .S2     FLASH_REG2,B6        ;

        STB     .D2T1   A3,*B6
        nop             9                    ;

        MVKL    .S2     FLASH_REG1,B6        ;

        MVK     .S1     FLASH_KEY3,A3        ;
   ||   MVKH    .S2     FLASH_REG1,B6        ;

        STB     .D2T1   A3,*B6               ;
        nop             9

        ZERO    .L2     B5                   ;
        mvkl            4,B7
        mvkh            4,B7
        ;flash memory is now unlocked

;** --------------------------------------------------------------------------*
;loop copying 128 bytes from internal mem to flash mem
FLASH_PAGE_PROG_LOOP:
        LDB     *A4++,B6                     ;get data
        ADD     1,B5,B5                      ;increment counter
        NOP     4
```

```
       STB     .D2T2   B6,*B4                ;write data to flash
       nop             9
       add             B4,B7,B4              ;increment flash address
       CMPLT   .L2x    B5,A6,B1              ;check loop condition
 [ B1] B       .S1     FLASH_PAGE_PROG_LOOP  ;
       NOP             5

       ;calculate read address
       mvkl            FLASH_START,B8
       mvkh            FLASH_START,B8
       add             B8,A1,B8


;** --------------------------------------------------------------------------*
;wait until the last data is accessible to exit subroutine
FLASH_PAGE_PROG_WAIT:
       LDB     .D1T1   *-A4(1),A3            ;get last data from mem
   ||  LDB             *-B8(1),B5            ;try to get last data from flash
       NOP             4
       CMPEQ   .L2X    A3,B5,B1              ;compare mem and flash
 [!B1] B       .S1     FLASH_PAGE_PROG_WAIT  ;
       NOP             5
       B       .S2     B3                    ;
       NOP             5

;****************************************************************************
;* FUNCTION NAME: _flash_prog_success                                       *
;****************************************************************************
_flash_prog_success:

       mvkl  TIMER0_REG,A4                   ;Output "1" on TOUT0
       mvkh  TIMER0_REG,A4

success_loop:                                ;produces 35% positive cycle on TOUT0
       mvkl  0x00000004,B4
       mvkh  0x00000004,B4
       stw   B4,*A4

       mvkl  0x00000000,B4
       mvkh  0x00000000,B4
       stw   B4,*A4

       B     success_loop                    ;infinite loop 1,0,1,0
       nop   5
       NOP
       NOP
```

APPENDIX M MICROCONTROLLER 'C' CODE

```
/*********************************************************************
FILE:    Timing.C
AUTHOR:  Ryan Eakin
PURPOSE: This program provides a user interface for the radar timing system.

*********************************************************************/


#include <hc16.h>
#include <stdio.h>                  // printf(..)
#include <string.h>
#include "..\rtl16_h\serio.h"       // getchr(), putchr(c)...
#include "..\rtl16_h\delay.h"       // Function delay( ms )
#include "..\rtl16_h\lcd.h"         // lcd_putch(), lcd_puts()..
#include "..\rtl16_h\kpd.h"         // kpd_lookup[] array of key values..
#include "..\rtl16_h\dio.h"         // Digital I/O functions
#include "..\rtl16_h\mdefine.h"     // Defines ports and their images
#include "vectrs16.c"

#define LINE1 0                     // Line 1 of LCD
#define LINE2 0x40                  // Line 2 of LCD starts at position 0x40
#define menuitems 10                //# of menu items for data storage
#define clock_period 16             //clock period in nanoseconds
#define UPARR 0x41                  // Ret code of UP arrow key
#define DNARR 0x36                  // Ret code of DOWN arrow key
#define LARR  0x42                  // Ret code of LEFT arrow key
#define RARR  0x30                  // Ret code of RIGHT arrow key
#define CARR  0x39                  // Ret code of Center arrow key
#define LAUX  0x44                  // Ret code of Left aux key
#define RAUX  0x43                  // Ret code of Right aux key


//global variables
int menu;
int singleshot=0;
int displayupdate=0;
int prfmask=0x00;
unsigned int data[menuitems][2], address[menuitems][2], cursorline, cursorpos, key;
char display_str[20] = { 0 };       // used to assemble LCD message before
                                    // calling lcd_putstr()
char menu_str[10][4][10];           // contains all menu labels

//Function to initialize variables
void ClearVars(void)
{
       int i;

       for(i=0;i<menuitems;i++)
       {
              data[i][0]=0x0000;    //Initalize to all zeros
              data[i][1]=0x0000;
       }

       strcpy(menu_str[0][0],"PRF");
       strcpy(menu_str[0][1],"uS");
       strcpy(menu_str[0][2]," ");
       strcpy(menu_str[0][3],"kHz");
       strcpy(menu_str[1][0],"DCH1 DEL");
       strcpy(menu_str[1][1],"uS");
       strcpy(menu_str[1][2],"     WID");
       strcpy(menu_str[1][3],"uS");
       strcpy(menu_str[2][0],"DCH2 DEL");
       strcpy(menu_str[2][1],"uS");
       strcpy(menu_str[2][2],"     WID");
       strcpy(menu_str[2][3],"uS");
       strcpy(menu_str[3][0],"DCH3 DEL");
```

```
        strcpy(menu_str[3][1],"uS");
        strcpy(menu_str[3][2],"     WID");
        strcpy(menu_str[3][3],"uS");
        strcpy(menu_str[4][0],"DCH4 DEL");
        strcpy(menu_str[4][1],"uS");
        strcpy(menu_str[4][2],"     WID");
        strcpy(menu_str[4][3],"uS");
        strcpy(menu_str[5][0],"RCH1 DEL");
        strcpy(menu_str[5][1],"uS");
        strcpy(menu_str[5][2],"     WID");
        strcpy(menu_str[5][3],"uS");
        strcpy(menu_str[6][0],"RCH2 DEL");
        strcpy(menu_str[6][1],"uS");
        strcpy(menu_str[6][2],"     WID");
        strcpy(menu_str[6][3],"uS");
        strcpy(menu_str[7][0],"RCH3 DEL");
        strcpy(menu_str[7][1],"uS");
        strcpy(menu_str[7][2],"     WID");
        strcpy(menu_str[7][3],"uS");
        strcpy(menu_str[8][0],"RCH4 DEL");
        strcpy(menu_str[8][1],"uS");
        strcpy(menu_str[8][2],"     WID");
        strcpy(menu_str[8][3],"uS");
        strcpy(menu_str[9][0],"RCH5 DEL");
        strcpy(menu_str[9][1],"uS");
        strcpy(menu_str[9][2],"     WID");
        strcpy(menu_str[9][3],"uS");

        //Setup external chip addresses
        address[0][0]=0x00;    //PRF
        address[0][1]=0xFF;    //NOT USED, PREVENTS DUPLICATION OF ADDRESSES DURING SEARCH
        address[1][0]=0x01;    //DAQ CH1 Delay
        address[1][1]=0x0A;    //DAQ CH1 Width
        address[2][0]=0x03;    //DAQ CH2 Delay
        address[2][1]=0x0C;    //DAQ CH2 Width
        address[3][0]=0x05;    //DAQ CH3 Delay
        address[3][1]=0x0E;    //DAQ CH3 Width
        address[4][0]=0x07;    //DAQ CH4 Delay
        address[4][1]=0x10;    //DAQ CH4 Width
        address[5][0]=0x02;    //Radar CH1 Delay
        address[5][1]=0x0B;    //Radar CH1 Width
        address[6][0]=0x04;    //Radar CH2 Delay
        address[6][1]=0x0D;    //Radar CH2 Width
        address[7][0]=0x06;    //Radar CH3 Delay
        address[7][1]=0x0F;    //Radar CH3 Width
        address[8][0]=0x08;    //Radar CH4 Delay
        address[8][1]=0x11;    //Radar CH4 Width
        address[9][0]=0x09;    //Radar CH5 Delay
        address[9][1]=0x12;    //Radar CH5 Width
}

void WriteData(int menunumber, int linenumber)
{
        unsigned int byteaddress, lb, hb, la, ha, line;
        unsigned int latchaddress=0x27;
        unsigned int bogusaddress=0x3F;

        line=linenumber/LINE2;

        latchaddress|=prfmask;    //Apply PRF mask anytime an address is written to Port E
        bogusaddress|=prfmask;    //      "

        la=address[menunumber][line];   //get desired chip address
        ha=la;
        lb=data[menunumber][line];      //get high and low data bytes
        hb=lb;
        hb=hb>>8;                       //shift to lower 8 bits
        hb&=0x00FF;                     //sift out lower 8 bits
        lb&=0x00FF;                     //"

        //Get address ready (compensate for pin 3 missing!!!)
```

```
        la&=0x07;             //Get lower 3 bits of address
        ha&=0x18;             //Get upper 2 bits of address
        ha=ha<<1;             //Shift upper part of address 1 space to left one space
        ha&=0x30;
        //Filter upper part of address
        byteaddress=0;
        byteaddress|=la;      //Put new byteaddress together
        byteaddress|=ha;                                          //"
    byteaddress|=prfmask;     //Apply PRF mask anytime an address is written to Port E


        //latch lower byte
        poke(_REG_PG,_PORTF,lb); //write lower data byte to Port F
        delay(1);
        poke(_REG_PG,_PORTE,latchaddress);    //write latch address to Port E
        delay(1);
        poke(_REG_PG,_PORTE,bogusaddress);    //Bogus address to Port E
        delay(1);

        //place upper byte on the lines
        poke(_REG_PG,_PORTF,hb);              //write upper data byte to Port F
        delay(1);

        //place real address on lines
        poke(_REG_PG,_PORTE,byteaddress);     //write real chip address to Port E
        delay(1);

        //always return to bogus address when done
        poke(_REG_PG,_PORTE,bogusaddress);    //Bogus address to Port E
        delay(1);
}


//Display function
void DispMenu(void)
{
       unsigned int temp,i,hb,lb,period,freq, thousand, hundred, count;


       //*****************************************************
       //Calculate and display PRF period/frequency
       //PRF period is: (!LB+1)*(!HB+1)*clock_period
       //               =(!LB*!HB + !LB + !HB +1)*clock_period
       //                 where LB and HB are high and low bytes
       //We must calculate high order answer and low order answer
       //seperately due to 16 bit limitations in the HC16
       //*****************************************************

       if(menu==0)
       {
              cursorline=LINE1;                    //Only 1 line allowed on PRF
              lcd_clear(0);                        //Clear entire LCD, cursor at Home
              lcd_putstr(menu_str[menu][0]);       //write a string
              lcd_locate(18);                      //position cursor
              lcd_putstr(menu_str[menu][1]);       //write a string
              lcd_locate(LINE2+17);                //position cursor
              lcd_putstr(menu_str[menu][3]);       //write a string

              count=data[menu][0];                 //Get delay counter value
              count=~count;                        //Complement the number

              thousand=count/1000;                 //Get thousands, ten thousands places
              hundred=count-thousand*1000;         //Get lowest three digits
              hundred=(hundred+2)*clock_period;    //"
              temp=hundred/1000;                   //Compute the carry to thousands
              thousand=thousand*clock_period+temp; //Add carry
              hundred=hundred-temp*1000;           //Subtract carry amount from hundreds

              sprintf(display_str,"%3d.%3d",thousand,hundred);    //format string

              //insert zeros into string before displaying decimals
```

```
        for(i=3;i<10;i++)
        {
                if(display_str[i]==32)
                display_str[i]=48;
        }

        lcd_locate(10);               //position cursor
        lcd_putstr(display_str);      //write string
        WriteData(menu, cursorline);
        //Send data to the chips

        //format frequency string
        sprintf(display_str,"%d.%d",1000/thousand,1000%thousand*10/thousand);

        //insert zeros into string before displaying
        for(i=0;i<10;i++)
        {
                if(display_str[i]==32)
                display_str[i]=48;
        }
        lcd_locate(LINE2+12);         //position cursor
        lcd_putstr(display_str);      //write string
        lcd_locate(cursorpos);        //position cursor for user input
}

//*****************************************************
//Calculate and display period for regular channels (delay and width)
//*****************************************************
else
{
        lcd_clear(0);                   // clear entire LCD, cursor at Home
        lcd_putstr(menu_str[menu][0]);  //write a string
        lcd_locate(18);                 //position cursor
        lcd_putstr(menu_str[menu][1]);  //write a string
        lcd_locate(LINE2+0);            //position cursor
        lcd_putstr(menu_str[menu][2]);  //write a string
        lcd_locate(LINE2+18);           //position cursor
        lcd_putstr(menu_str[menu][3]);  //write a string

        //*****************************************************
        //                DELAY
        //*****************************************************

        count=data[menu][0];            //Get delay counter value
        count=~count;                   //Complement the number

        thousand=count/1000;            //Get thousands, ten thousands places
        hundred=count-thousand*1000;    //Get lowest three digits
        hundred=(hundred+1)*clock_period;     //"
        temp=hundred/1000;                    //Compute the carry to thousands
        thousand=thousand*clock_period+temp;  //Add carry
        hundred=hundred-temp*1000;            //Subtract carry amount from hundreds

        sprintf(display_str,"%3d.%3d",thousand,hundred);    //format string

        //insert zeros into string before displaying decimals
        for(i=4;i<10;i++)
        {
                if(display_str[i]==32)
                display_str[i]=48;
        }

        lcd_locate(10);                       //position cursor
        lcd_putstr(display_str);              //write string


        //*****************************************************
        //                WIDTH
        //*****************************************************

        count=data[menu][1];                  //Get delay counter value
```

```
                count=~count;                 //Complement the number
                thousand=count/1000;          //Get thousands, ten thousands places
                hundred=count-thousand*1000;  //Get lowest three digits
                hundred=(hundred+1)*clock_period;     //"
                temp=hundred/1000;                    //Compute the carry to thousands
                thousand=thousand*clock_period+temp;  //Add carry
                hundred=hundred-temp*1000;            //Subtract carry amount from hundreds

                sprintf(display_str,"%3d.%3d",thousand,hundred);    //format string

                //insert zeros into string before displaying decimals
                for(i=4;i<10;i++)
                {
                        if(display_str[i]==32)
                        display_str[i]=48;
                }

                lcd_locate(LINE2+10);                 //position cursor
                lcd_putstr(display_str);              //write string
                lcd_locate(cursorline+cursorpos);     //position cursor for user input
                WriteData(menu, cursorline);          //Send data to the chips
        }

}

void Increment_Data(void)
{
        //local vars
        unsigned int increment, old_value, line;


        //calculate value to increment data by depending on cursorposition
        switch(cursorpos)
    {
        case 10:
                increment=10000/clock_period*10;
                break;
        case 11:
                increment=10000/clock_period;
                break;
        case 12:
                increment=1000/clock_period;
                break;
        case 14:
                increment=100/clock_period;
                break;
        case 15:
                increment=10/clock_period;
                break;
        case 16:
                increment=1;
                break;
        }

        //prepare for data increment
        line=cursorline/LINE2;                        //0 for line 1, 1 for line 2
        old_value=data[menu][line];

        switch(key)
        {
        case DNARR:
                //increment data
                data[menu][line]=data[menu][line]+increment;

                //check for overflow condition
                if(old_value>data[menu][line])
                data[menu][line]=0xFFFE;
                break;

        case UPARR:
                //decrement data
```

```
                data[menu][line]=data[menu][line]-increment;

                //check for overflow condition
                if(old_value<data[menu][line])
                        data[menu][line]=0x0000;
                break;

        default:
                break;
        }
}

void SingleShot(void)
{
        int singleshotval=0x7F;
        int bogusaddress=0x3F;

        //Apply PRF mask anytime an address is written to Port E
        singleshotval|=prfmask;

        //Apply PRF mask anytime an address is written to Port E
        bogusaddress|=prfmask;

        //Toggle single shot pin
        poke(_REG_PG,_PORTE,singleshotval);          //Bringing single shot line high
        delay(1);


        //always return to bogus address when done
        poke(_REG_PG,_PORTE,bogusaddress);           //Bogus address to Port E
        delay(1);

}

void Check_Keys(void)
{
        int attempt=0;
        int tempkey=0;                               //used for signed comparisons only

        //if input from keyboard, perform action

        //loop until a value is received from the keypad or timout
        while((tempkey==0)&&(attempt<100))
        {
                tempkey=kpd_chkch();
                attempt++;
                delay(1);
        }

        //tell main program to update display if a key was pressed
        if(tempkey!=0)
                displayupdate=1;

        key=tempkey;                //write the value to key global location

        switch(key)
        {
        case LARR:
                menu--;
                if(menu<0)
                        menu=0;
            break;

        case RARR:
                menu++;
                if(menu>menuitems-1)
                        menu=menuitems-1;
                break;

        case UPARR:
                Increment_Data();
```

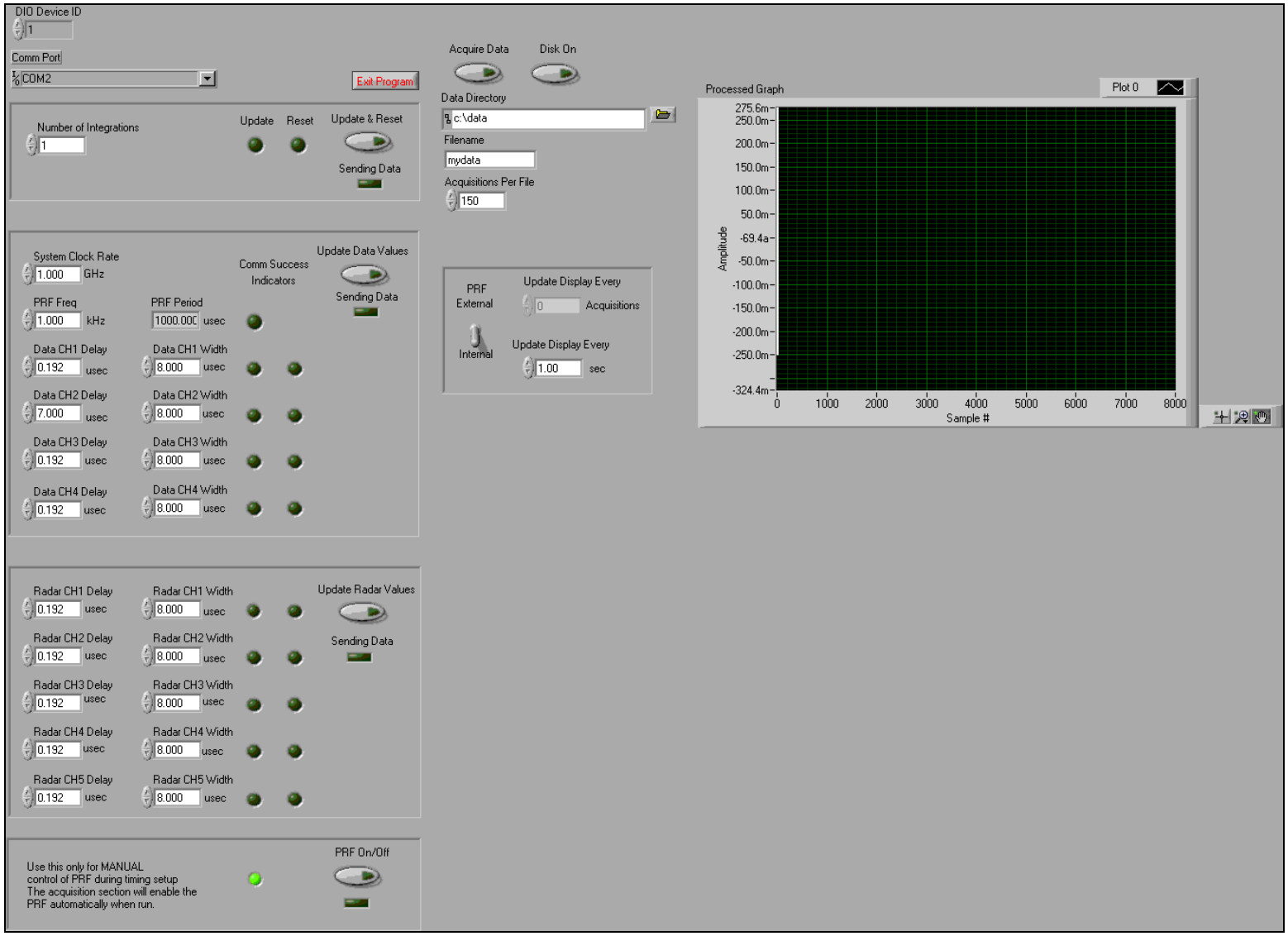
```
                break;

        case DNARR:
                Increment_Data();
                break;

        case CARR:
                if(cursorline==LINE1)
                        cursorline=LINE2;
                else
                        cursorline=LINE1;
                break;

        case LAUX:
               SingleShot();
                break;

        case RAUX:
                cursorpos++;
                if(cursorpos>16)
                        cursorpos=10;
                if(cursorpos==13)
                        cursorpos++;
                break;

        default:
                key=0;
                break;
    }
}


//*********************************************************
//Small function to read serial input with timeout
//*********************************************************

int Get1char(void)
{
        int attempt=0;
        int value=-1;

        //loop until a value is received from the serial port or timout
        while((value==-1)&&(attempt<100))
        {
                value=chkchr();
                attempt++;
                delay(1);
        }

        return(value);
}

//*********************************************************************
// Writes number of integrations through portGP as a serial connection.
// Lines are designated as follows:
//   OC1 - Device reset, not used here
//   OC2 - data line
//   OC3 - latch data line
//   OC4 - clock line
//*********************************************************************

void Write_Integration(int Intaddress, int Intdata)
{
        unsigned int tempvalue=0x00;
        unsigned int tempmask=0x8000;
        unsigned int output;
        unsigned int clockhigh=0x40;      //clockline is a 1
        unsigned int clocklow=0xBF;       //clockline is a 0
        int i;

        //start with all lines at zero
```

```
        poke(_REG_PG,_PORTGP,tempvalue);

        //loop, writing one bit of the data at a time, MSB first
        for(i=1;i<17;i++)
        {
                tempvalue=Intdata;
                tempvalue&=tempmask;            //Get only one bit
                tempvalue=tempvalue/tempmask;   //shift remaining bit to the LSB

                if(tempvalue==1)
                        output=0x10;
                else
                        output=0x00;

                poke(_REG_PG,_PORTGP,output);    //put the output bit on the data line

                output|=clockhigh;
                poke(_REG_PG,_PORTGP,output);    //toggle clock high

                output&=clocklow;
                poke(_REG_PG,_PORTGP,output);    //toggle clock low

                tempmask=tempmask>>1;            //shift the mask one bit to the right
                tempmask&=0x7FFF;                //force MSB to zero

        }

        //all data has been shifted out, now toggle the latch line
        poke(_REG_PG,_PORTGP,0x20);              //bring latch line high
        poke(_REG_PG,_PORTGP,0x00);              //bring all lines low

}

void Reset(void)
{
        poke(_REG_PG,_PORTGP,0x08);             //Bringing reset line high
        poke(_REG_PG,_PORTGP,0x00);             //Bringing reset line low
}

//*********************************************************
//Check serial port for input from the computer
//*********************************************************

void Check_Serial(void)
{
        int  first=-1;                          //1st serial port input byte
        int  second=-1;                         //2nd serial port input byte
        int  third=-1;                          //3rd serial port input byte
        int  tempmenu, templine, i, j;
        int  final=0;

        first=Get1char();                       //look for garbage
        if(first !=-1)                          //Data is coming in
        {
                first=Get1char();               //get real number
                putch('1');                     //send 1st acknowledgement

                second=Get1char();              //get garbage
                second=Get1char();              //get real number
                if(second !=-1)
                {
                        putch('2');            //send 2nd acknowledgement

                        third=Get1char();      //get garbage
                        third=Get1char();      //get real number
                        putch('3');            //send 3rd acknowledgement
                }
        }

        if(third!=-1)                           /A valid sequence was received
        {
```

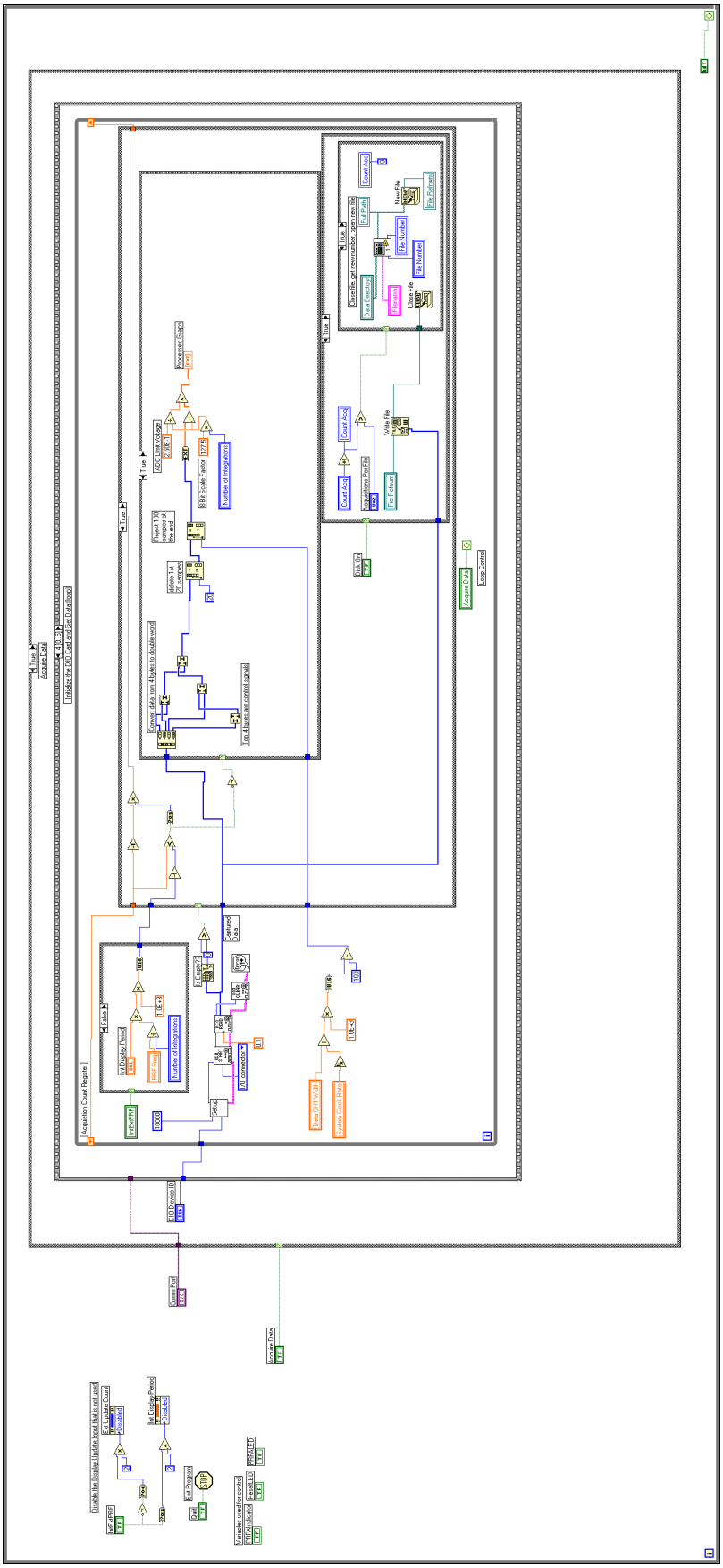
```
                displayupdate=1;

                //Input address is for the width or delay latches

                if((first>=0)&&(first<=0x12))
                {

                //loop through all latch addresses and see if
                //serial input matches one of them
                for(i=0;i<menuitems;i++)
                        for(j=0;j<2;j++)
                                if(address[i][j]==first)    //If input address is the same
as this one, get menu and line#
                                {
                                        tempmenu=i;
                                        templine=j;
                                }

                //now that address is known, concatenate the upper and lower bytes
                second=second<<8;      //Shift and filter the high order byte
                second&=0xFF00;
                final|=second;       / /combine high and low order into one
                final|=third;

                //write value to memory to be consistent with LCD interface
                data[tempmenu][templine]=final;

                //change format for consitency with LCD interface
                templine=templine*LINE2;
                WriteData(tempmenu,templine);           //send the data to the hardware
                }

                //check if single shot is being addressed
                if(first==0x20)
                        SingleShot();

                //Check if Reset is being addressed
                if(first==0x21)
                        Reset();

                //Check if integration counter is being addressed
                if(first==0x22)
                {
                        //concatenate the upper and lower bytes
                        second=second<<8;  //Shift and filter the high order byte
                        second&=0xFF00;
                        final|=second;     //combine high and low order into one
                        final|=third;

                        //Write data to the integration counter
                        Write_Integration(first,final);
                }

                //Check if PRF enable is being addressed
                if(first==0x23)
                {
                        if(second==1)
                                prfmask=0x80;
                        else
                                prfmask=0;
                }
        }
}


//Main function
void main(void)
{
        //local variables
        unsigned int serorkey;  //will select serial port or keypad operation through IC3
```

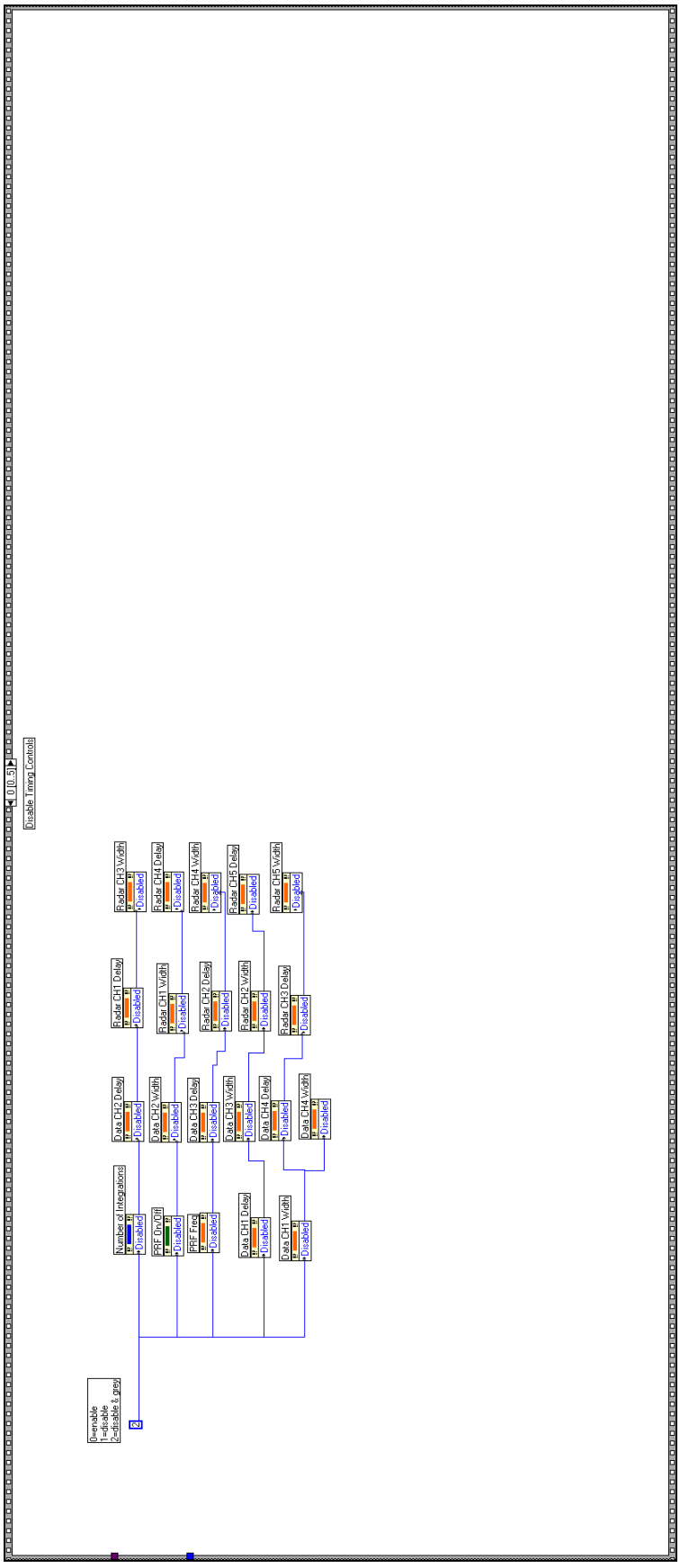
```
        //initialation
        menu=0;                         //Start at menu 0, line 1
        cursorline=LINE1;               //Initial cursor line
        cursorpos=11;                   //Initial editing cursor location


        setbaud( BAUD9600 );
        lcd_init( 2, 0, 2 );            //2 line, 5x7, cursor (#2 normal, #3 shifted)
        ClearVars();                    //Clear all values in memory
        poke(_REG_PG,_DDRE, 0xFF);      //set all Port E as output
        poke(_REG_PG,_DDRF, 0xFF);      //set all Port F as output
        poke(_REG_PG,_DDRGP,0x78);      //OC pins as output (port GP)

        //Display 1st menu
        DispMenu();
        delay(1000);

        //begin interactive loop
        while(menu<=20)
        {
                serorkey=peek(_REG_PG,_PORTGP);
                serorkey&=0x04;         //filter only IC3 pin

                if(serorkey==4)
                        Check_Keys();
                else
                        Check_Serial();

                if(displayupdate!=0)
                {
                        DispMenu();
                        displayupdate=0;
                        if(serorkey==4)
                                delay(250);     //delay if using the keypad
                }
        }
}
```

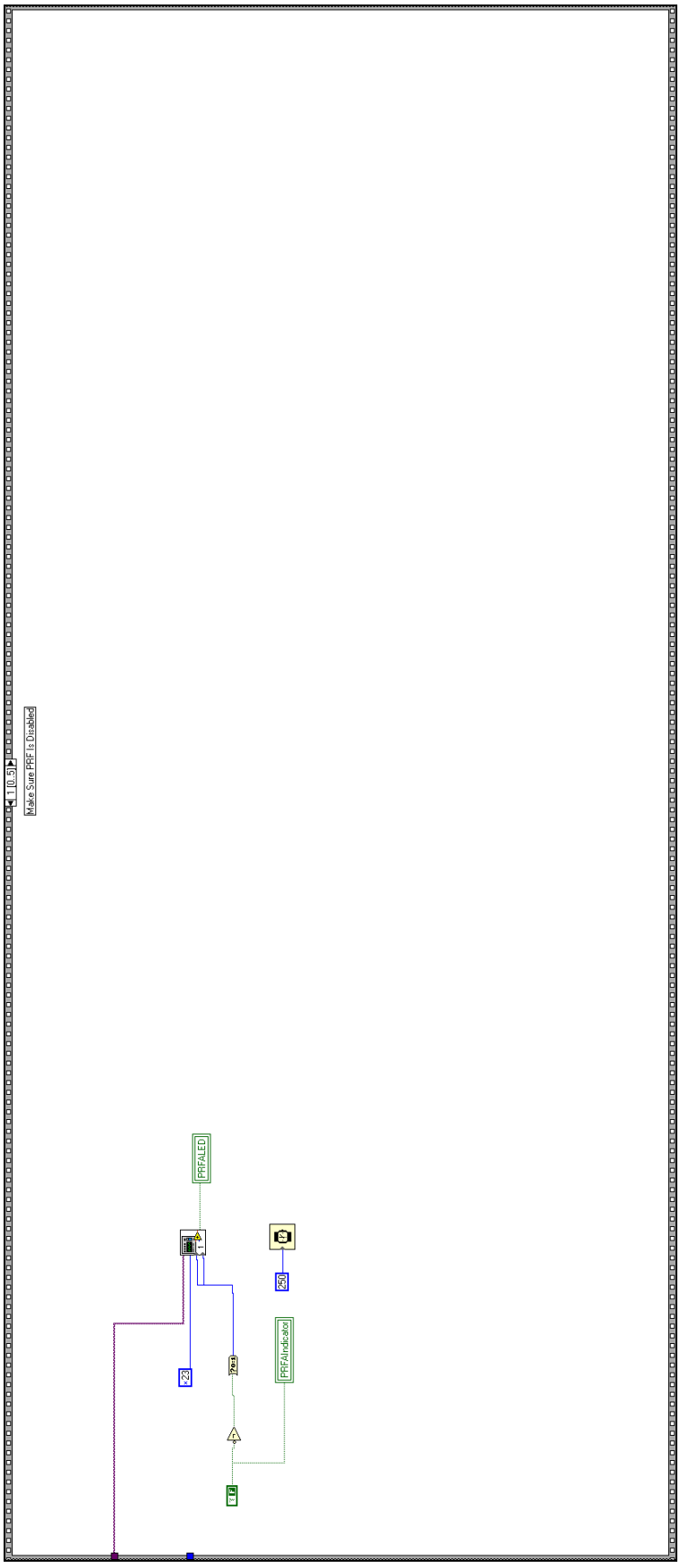
APPENDIX N LABVIEW DATA ACQUISITION PROGRAM

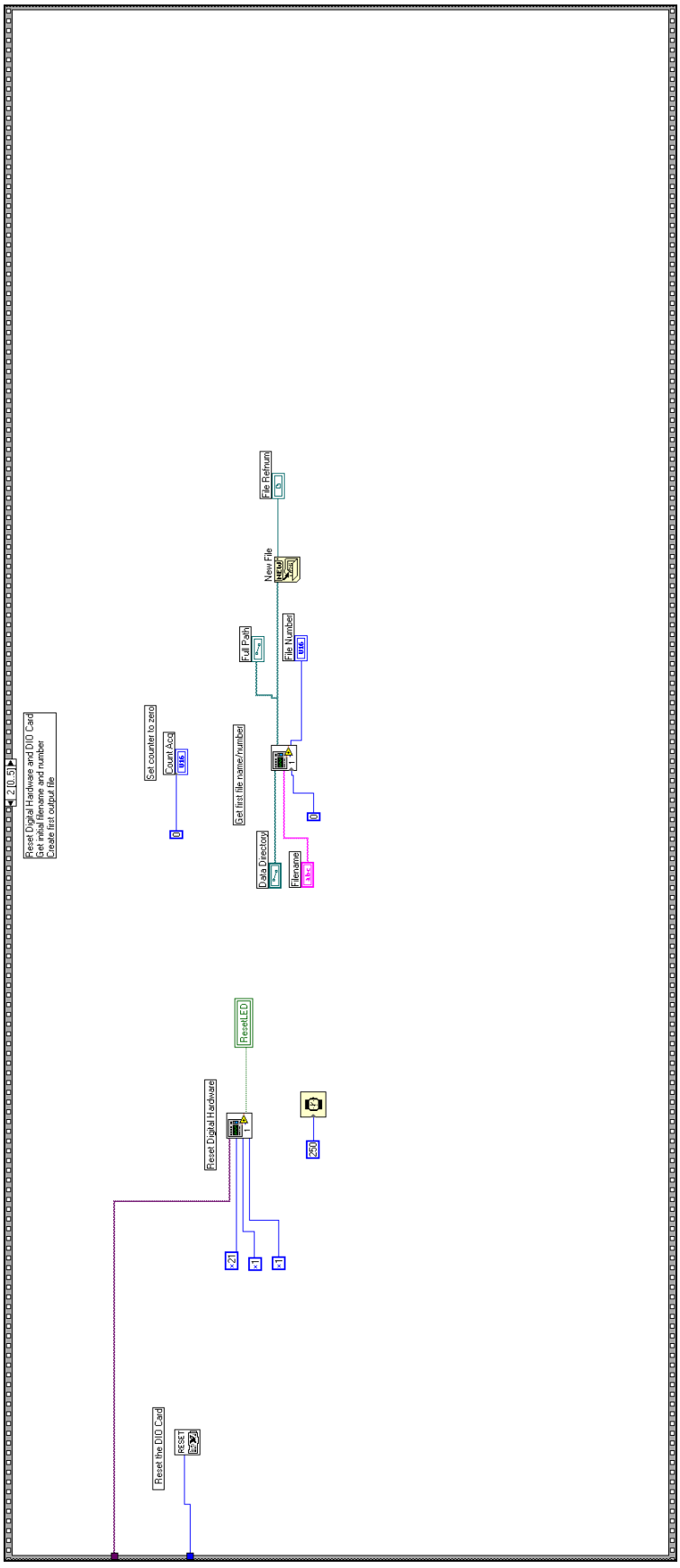


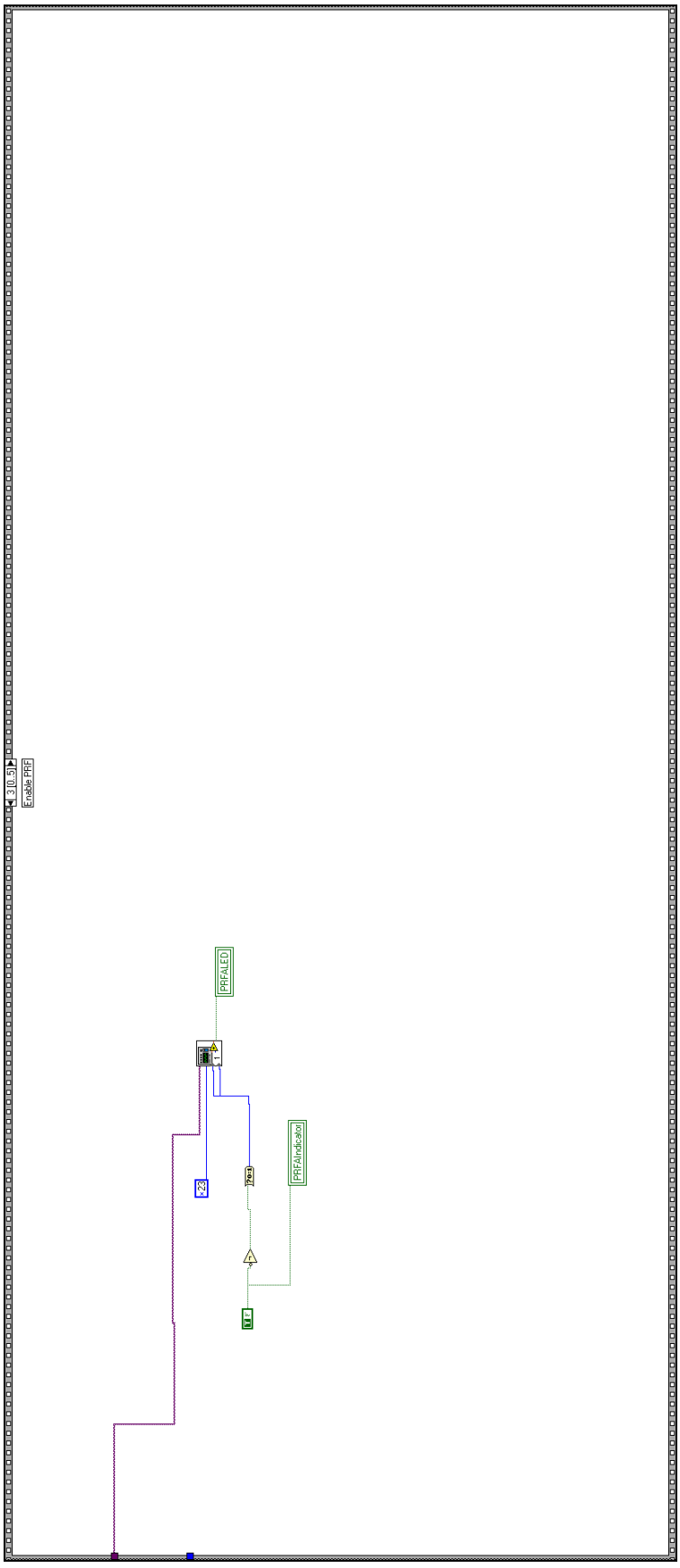
GUI Front Panel

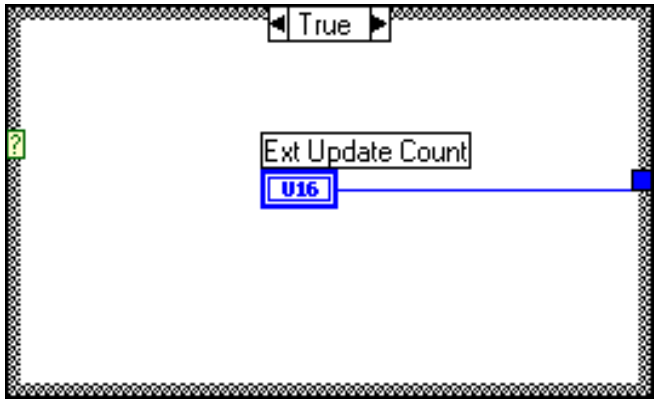


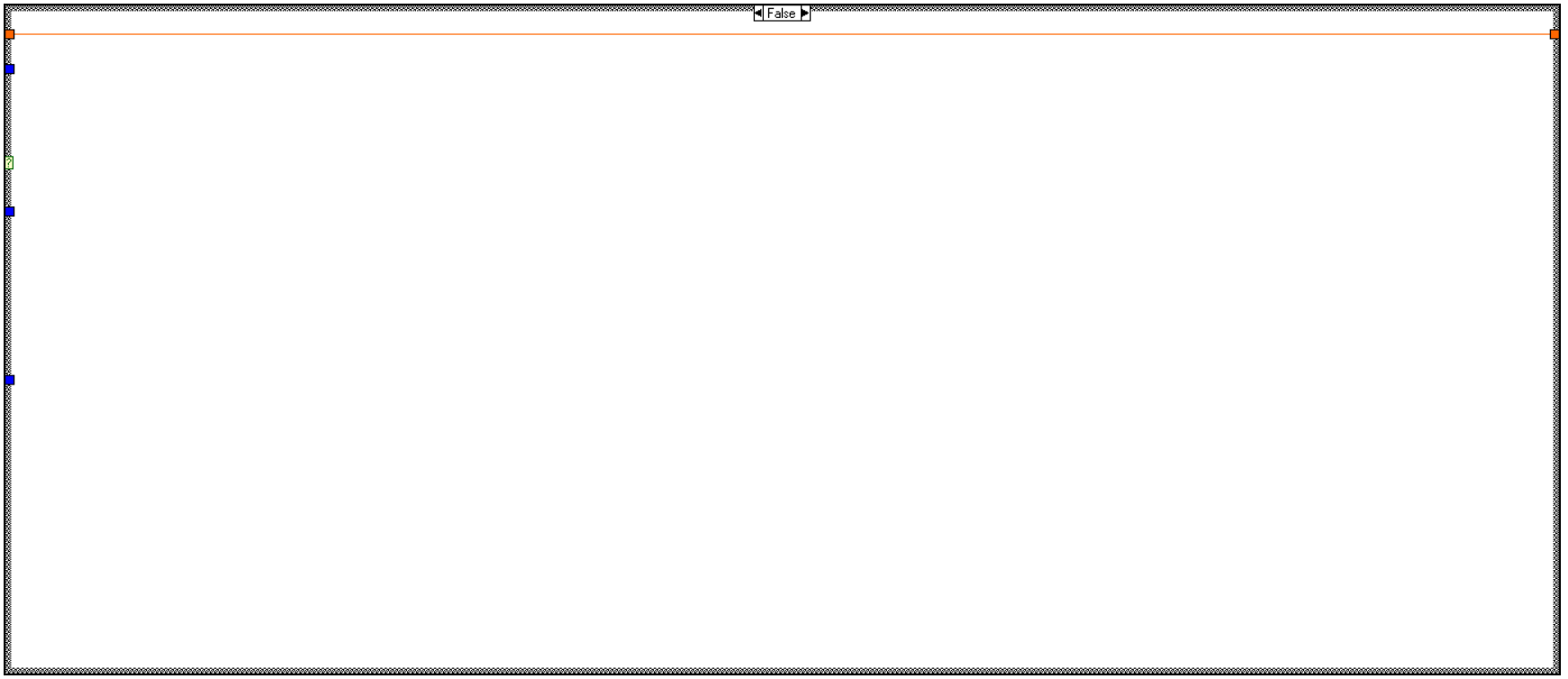

Disable Timing Controls
0 (0..5)
0=enable
1=disable
2=disable & grey
2
Number of Integrations
FRF Dwell
FRF Freq
Data CH1 Delay
Data CH1 Width
Data CH2 Delay
Data CH2 Width
Data CH3 Delay
Data CH3 Width
Data CH4 Delay
Data CH4 Width
Radar CH1 Delay
Radar CH1 Width
Radar CH2 Delay
Radar CH2 Width
Radar CH3 Delay
Radar CH3 Width
Radar CH4 Delay
Radar CH4 Width
Radar CH5 Delay
Radar CH5 Width
Disabled

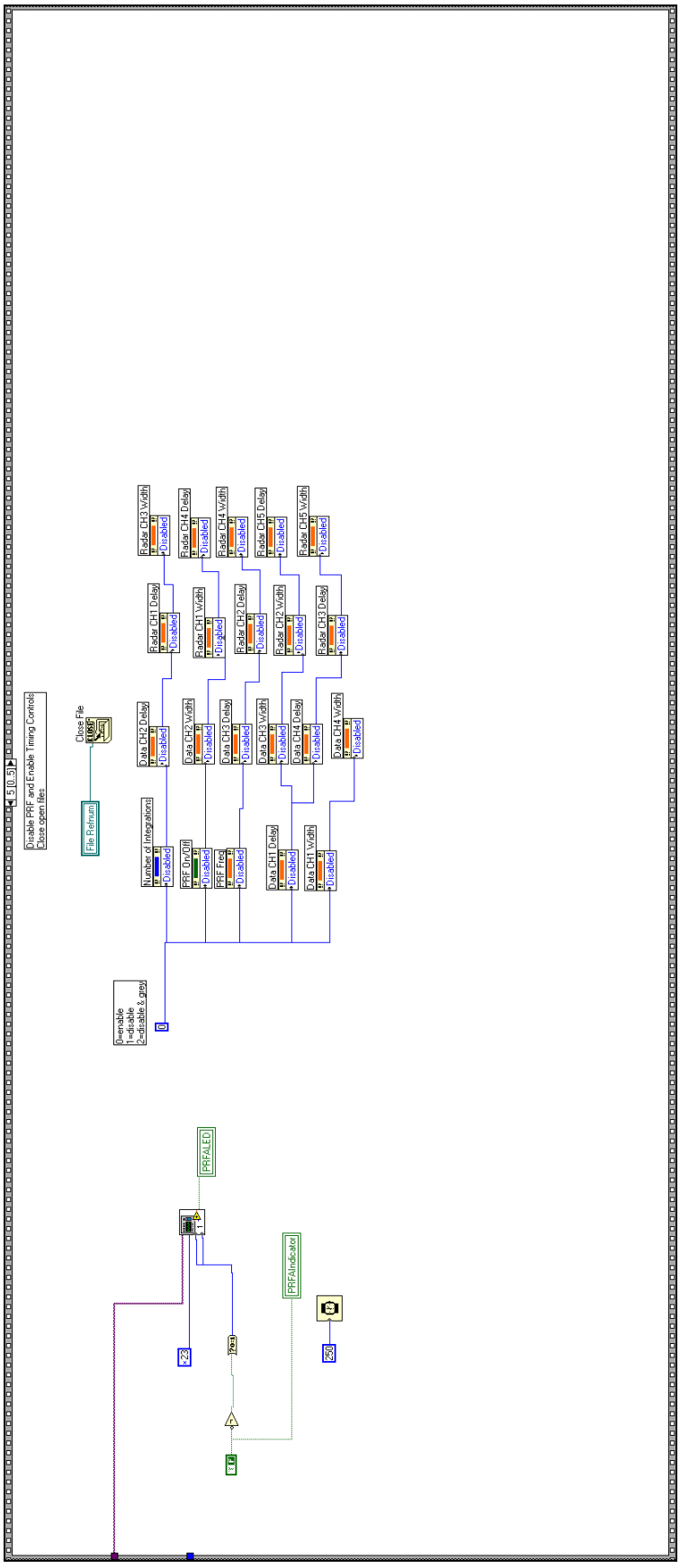

Make Sure PRF is Disabled
1 (0.5)
PRFALED
-23
250
PRFAIndicator

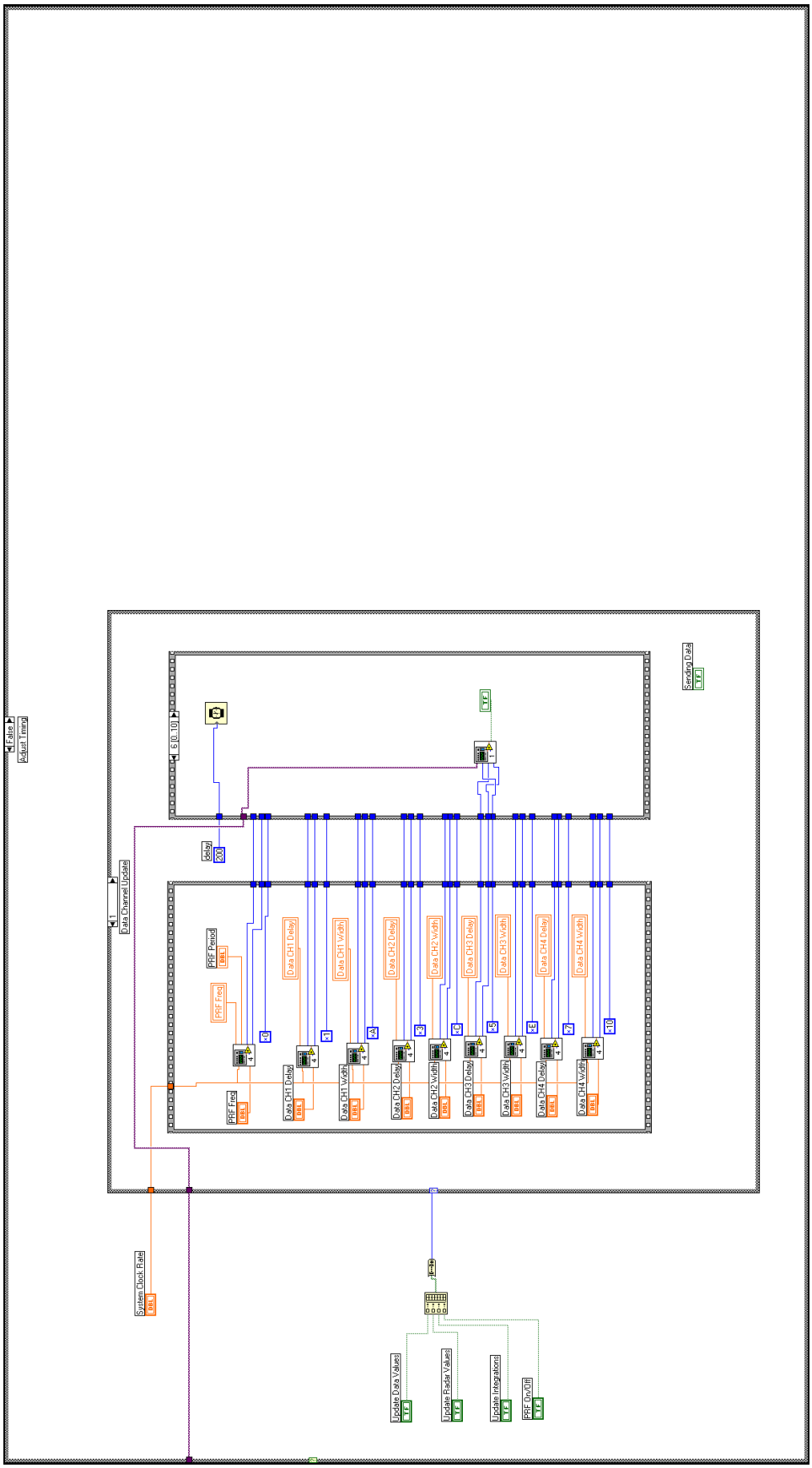

Reset Digital Hardware and DIO Card
Get initial filename and number
Create first output file
2 (0.5)
Set counter to zero
Count Accs
I16
0
Get first file name/number
Full Path
File Number
I16
File Refnum
New File
Data Directory
Filename
A16
0
ResetLED
Reset Digital Hardware
TF
250
Reset the DIO Card
RESET
TF

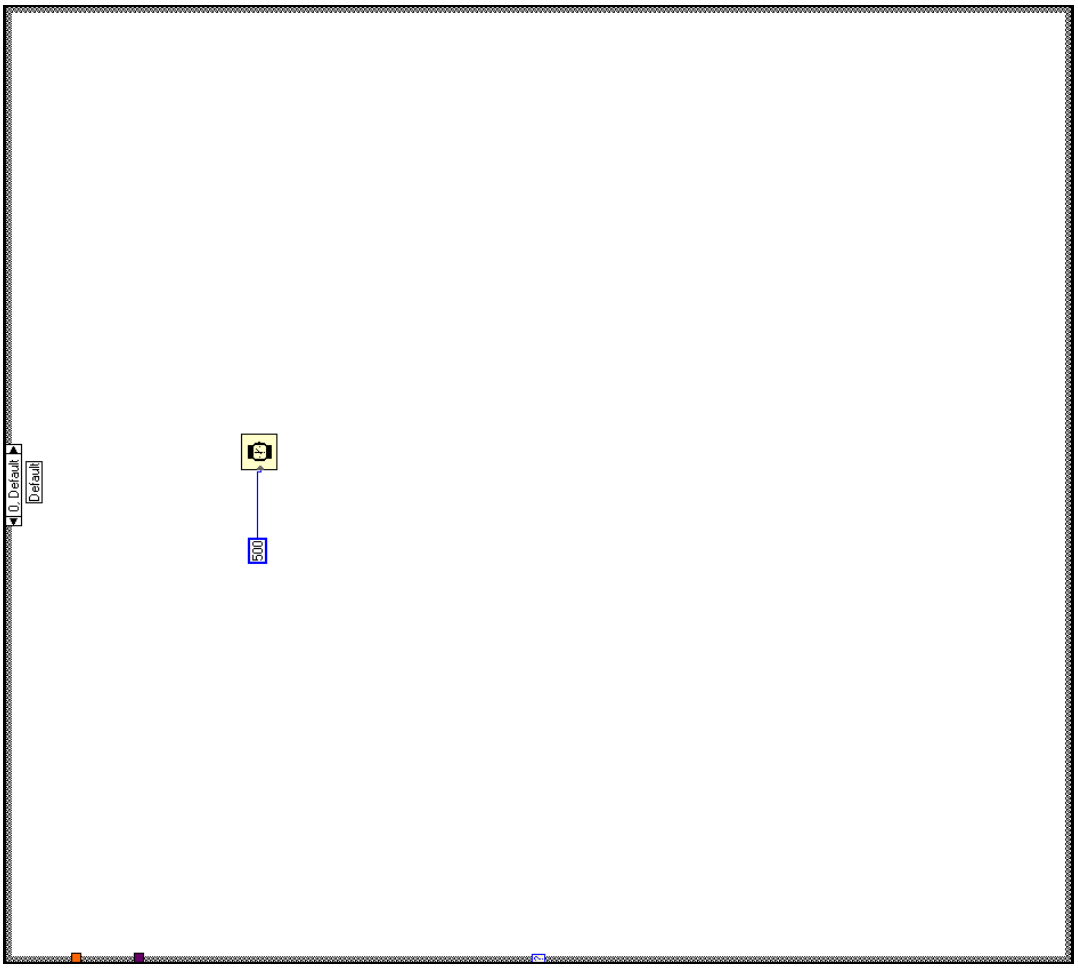

Enable PFF
PRFALED
PRFAIndicator

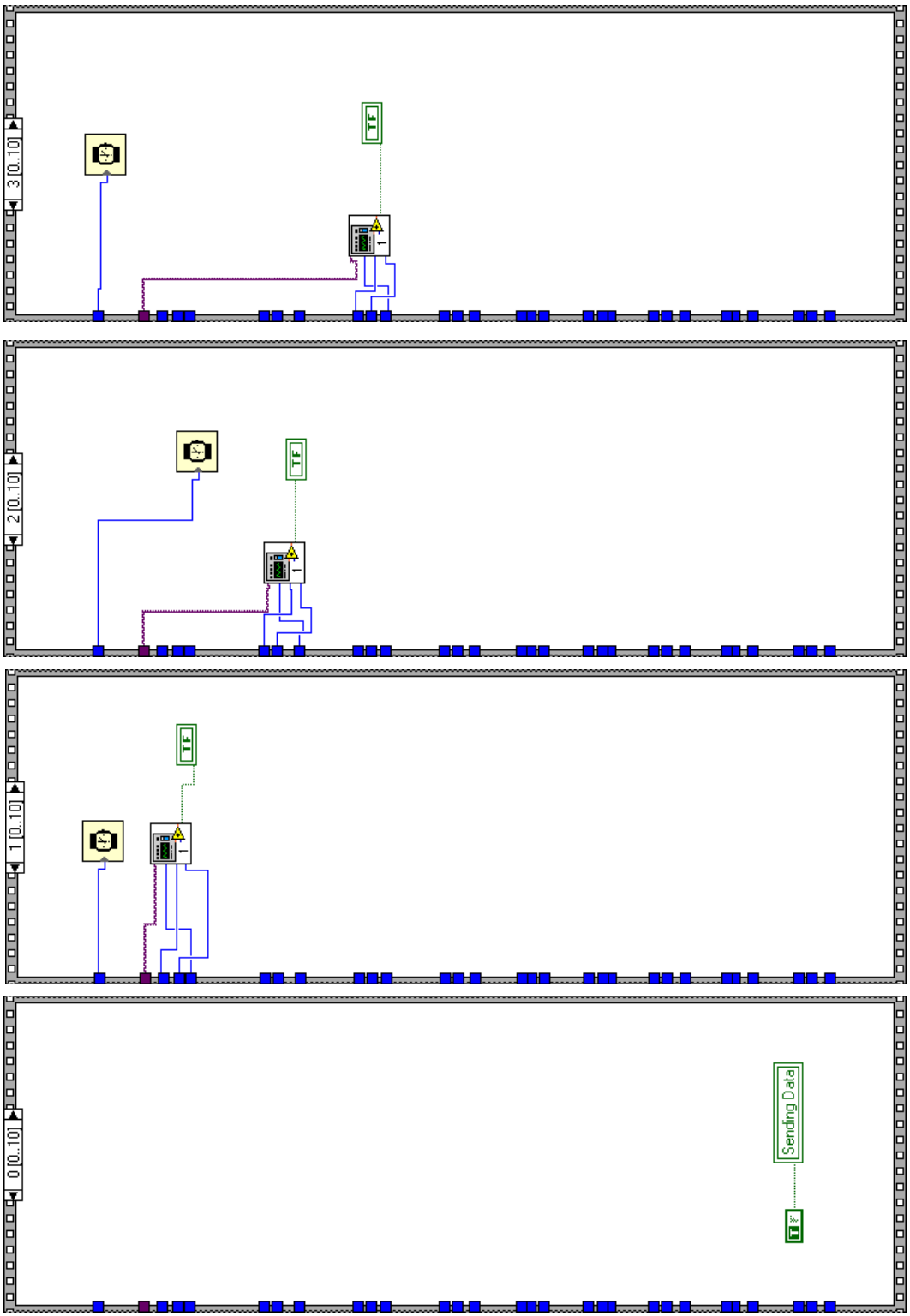

True
Ext Update Count
U16

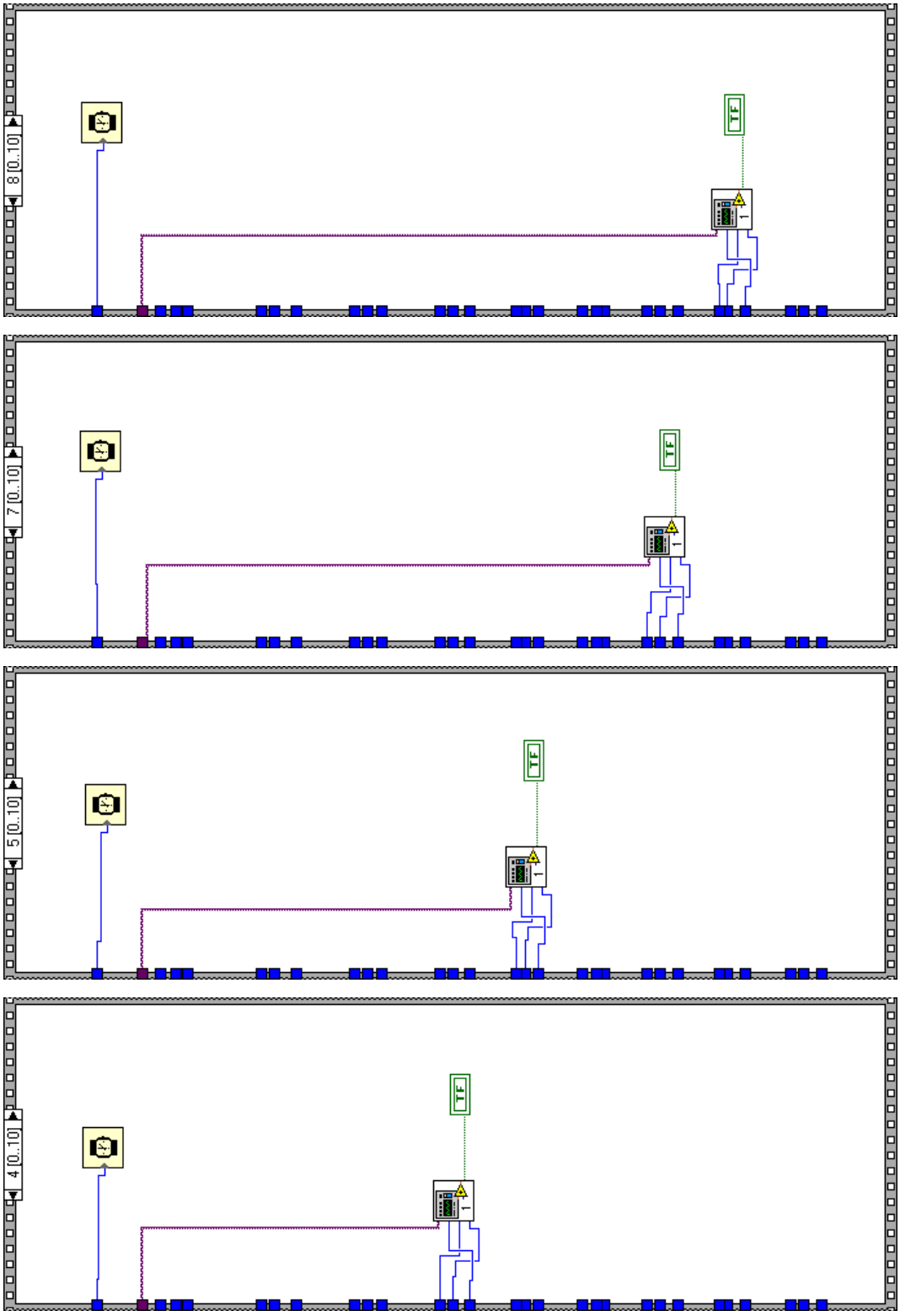

False

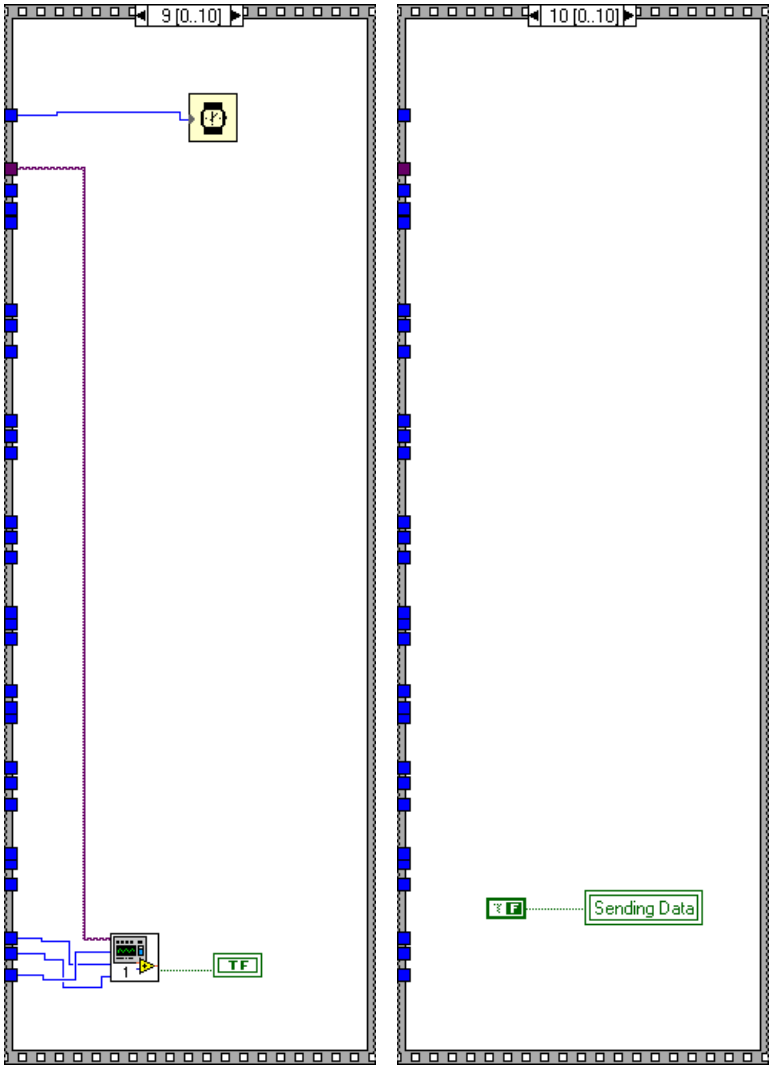

Disable PRF and Enable Timing Controls
Close open files
Close File
File Refnum
Number of Integrations
PRF Div/Dil
PRF Freq
Data CH2 Delay
Data CH2 Width
Data CH3 Delay
Data CH3 Width
Data CH4 Delay
Data CH4 Width
Data CH1 Delay
Data CH1 Width
Radar CH1 Delay
Radar CH1 Width
Radar CH2 Delay
Radar CH2 Width
Radar CH3 Delay
Radar CH3 Width
Radar CH4 Delay
Radar CH4 Width
Radar CH5 Delay
Radar CH5 Width
Disabled
Enable
1=disable
2=disable & grey
PRFALED
PRFAIndicator
23
250

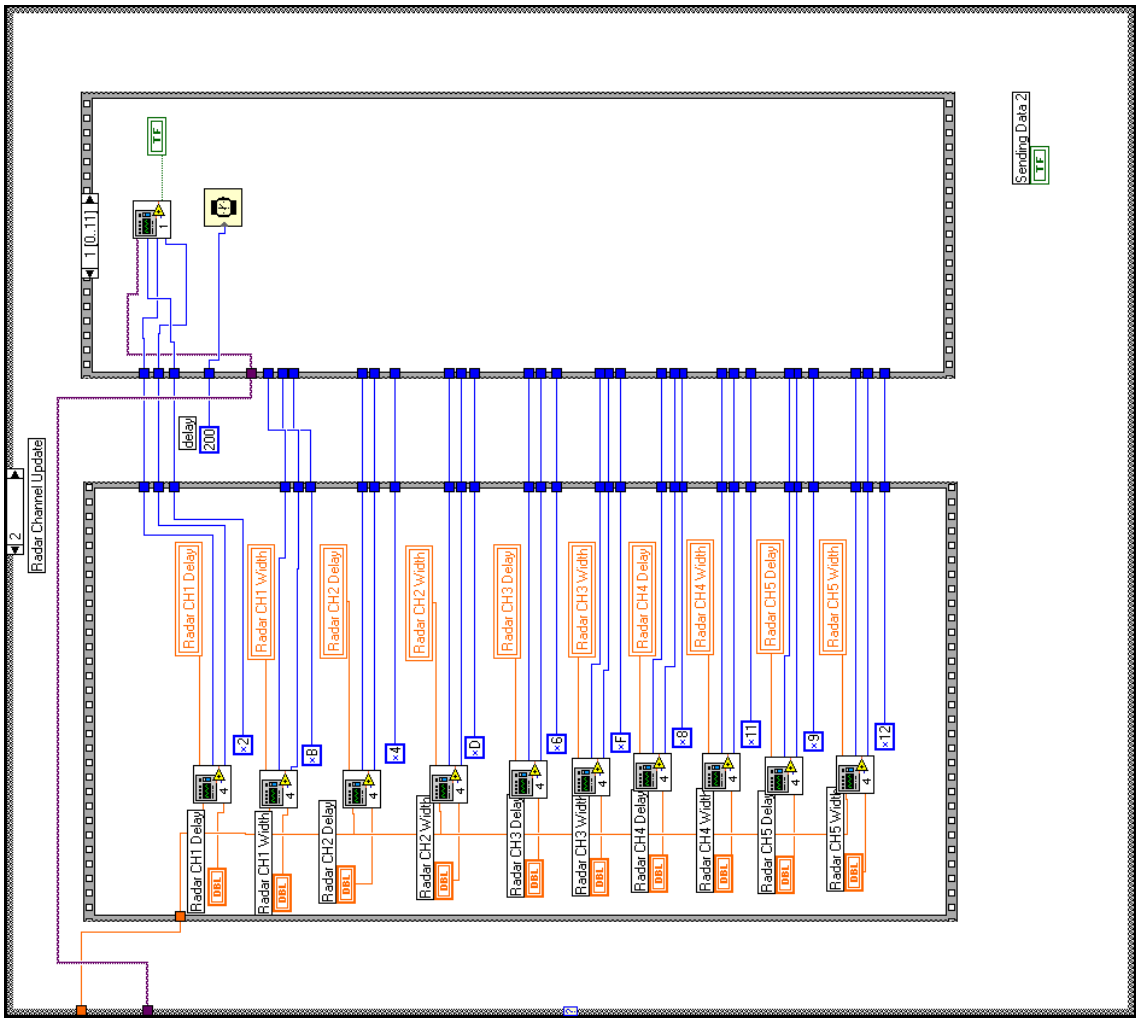

Adjust Timing
Data Channel Update
Sending Data
Delay
PRF Period
PRF Freq
Data CH1 Delay
Data CH1 Width
Data CH2 Delay
Data CH2 Width
Data CH3 Delay
Data CH3 Width
Data CH4 Delay
Data CH4 Width
System Clock Rate
Update Data Values
Update Radar Values
Update Integrations
PRF Div Diff

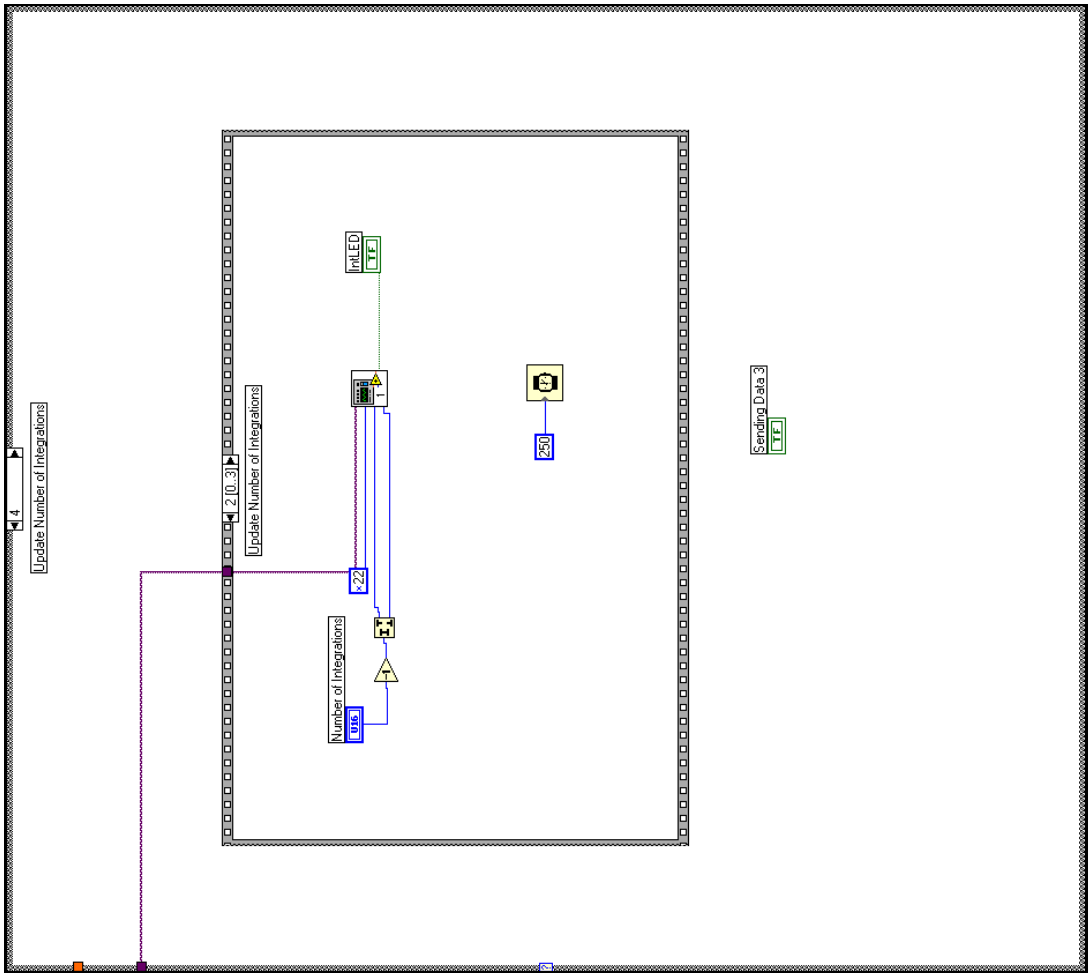

0: Default
Default
500

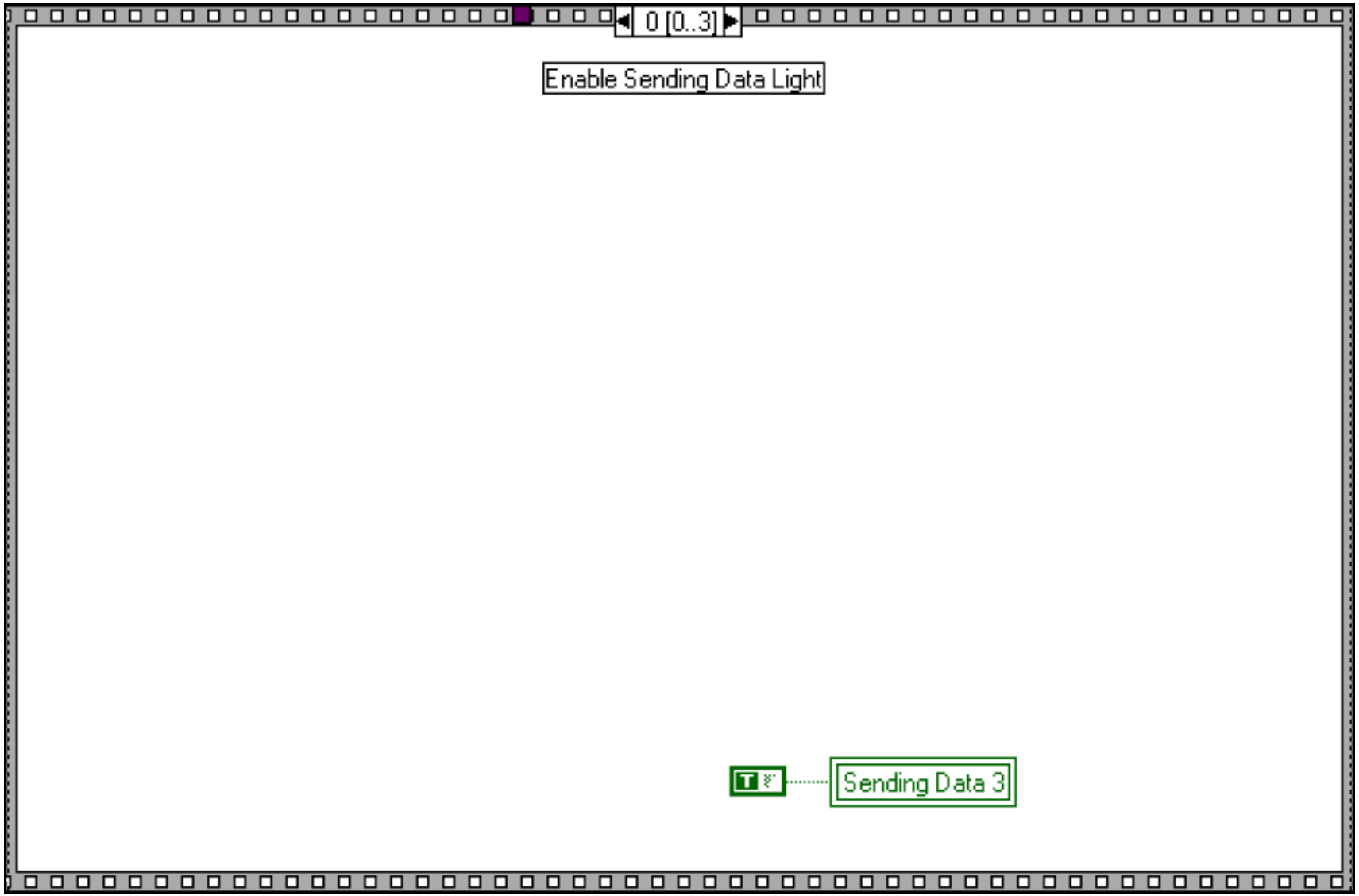

3 [0..10]
TF
1
2 [0..10]
TF
1
1 [0..10]
TF
1
0 [0..10]
Sending Data

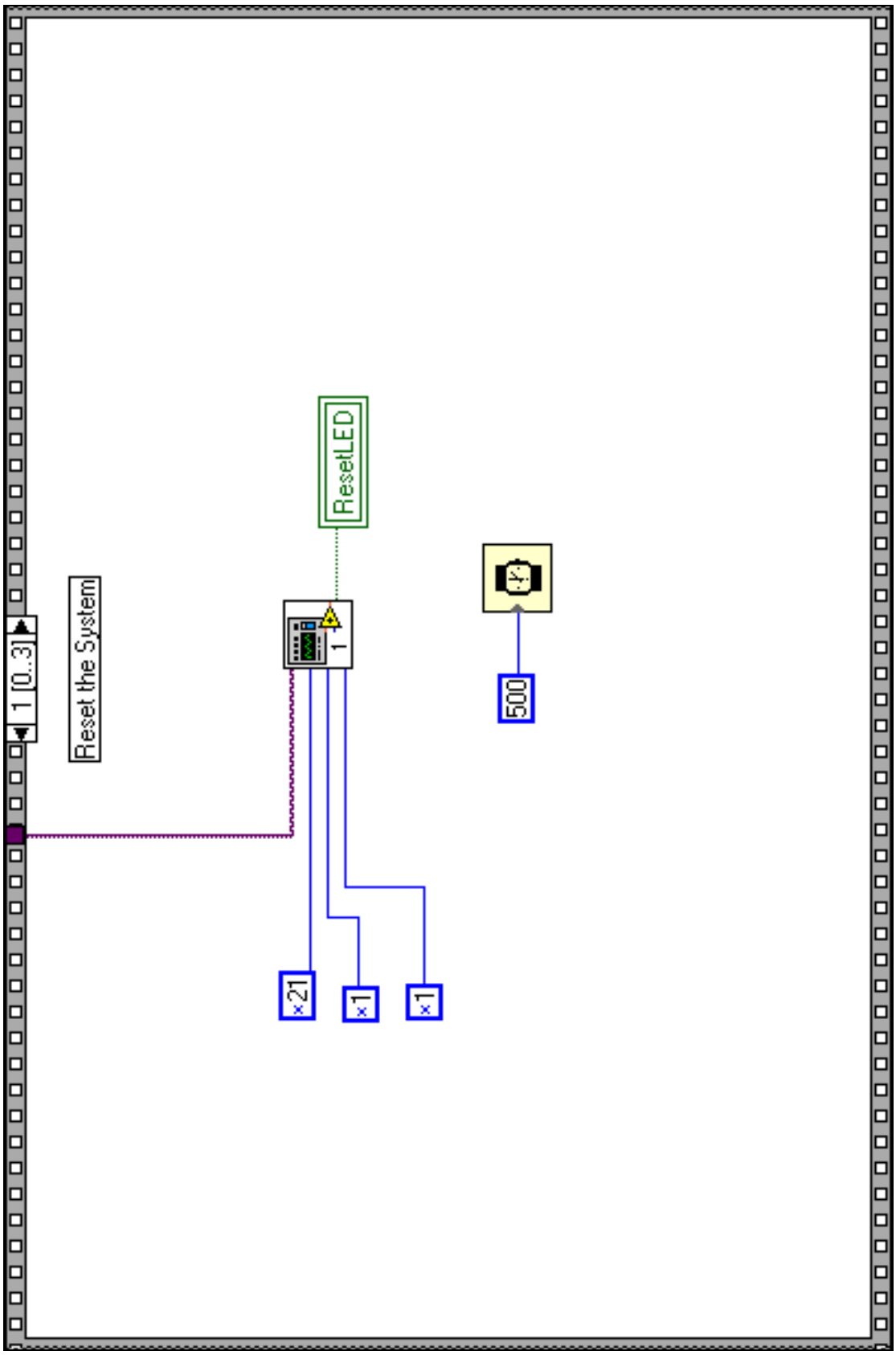

8 [0..10]
TF
1
7 [0..10]
TF
1
5 [0..10]
TF
1
4 [0..10]
TF
1

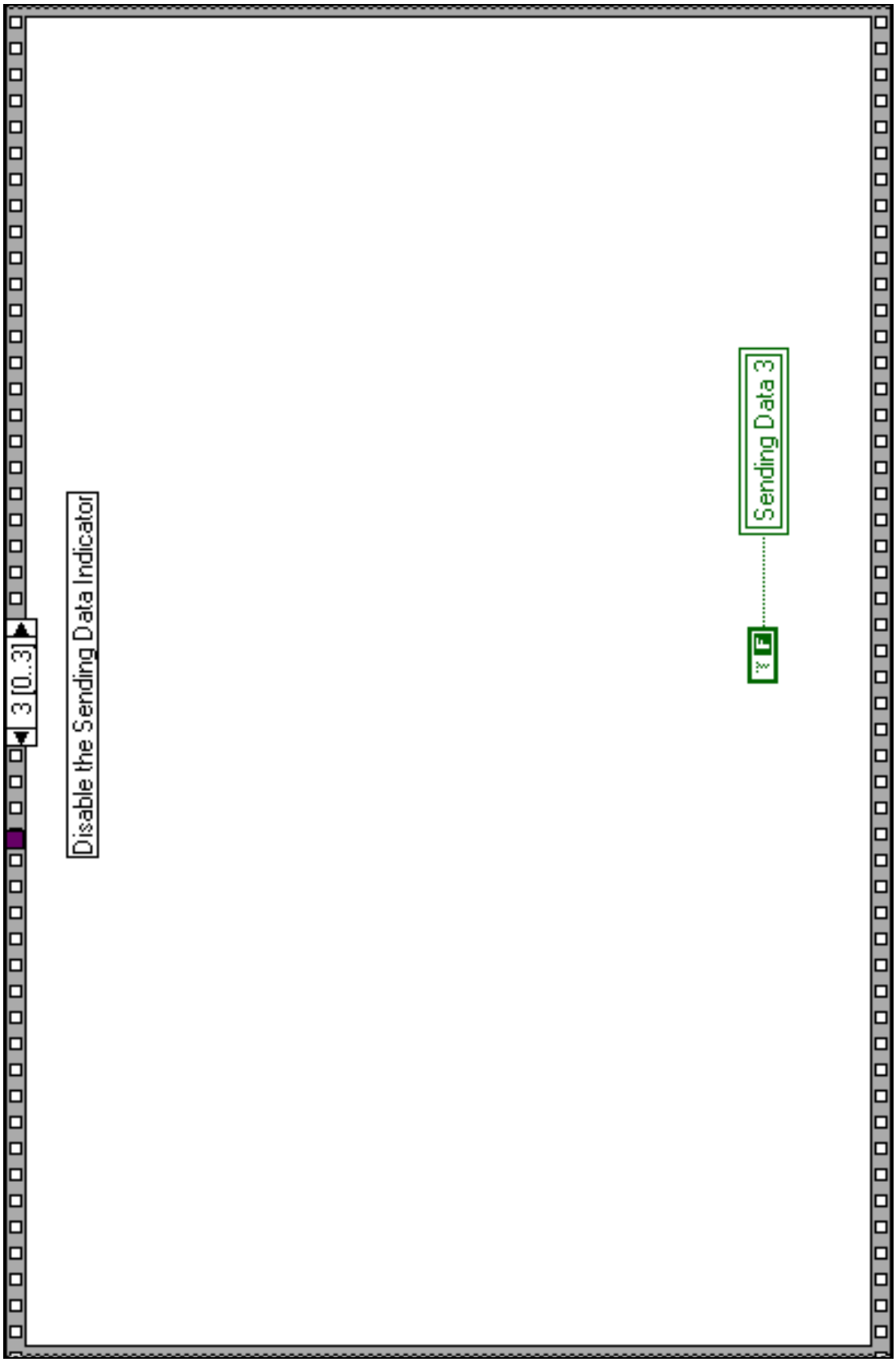

9 [0..10]
10 [0..10]
TF
Sending Data

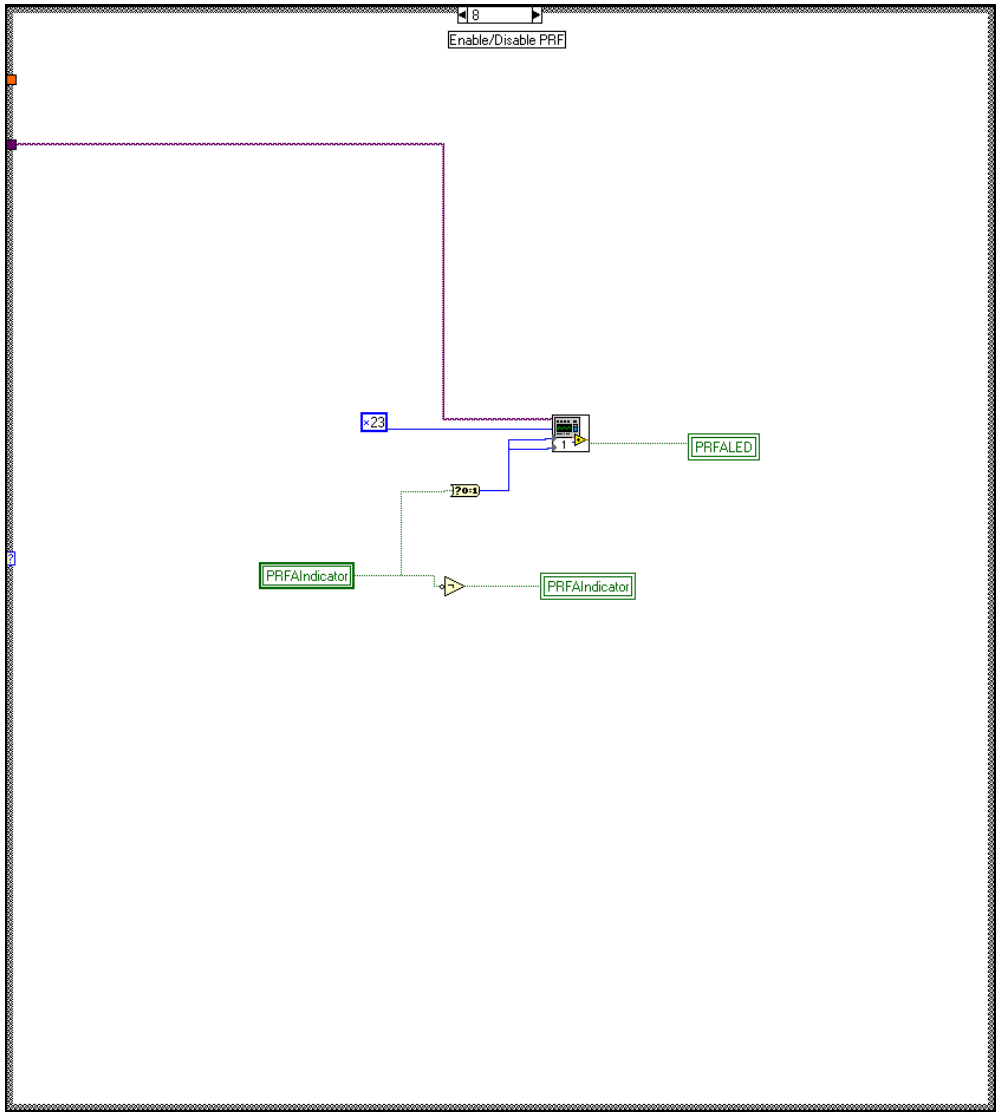

Radar Channel Update
2
delay
200
1 [0..11]
TF
Sending Data 2
Radar CH1 Delay
Radar CH1 Width
Radar CH2 Delay
Radar CH2 Width
Radar CH3 Delay
Radar CH3 Width
Radar CH4 Delay
Radar CH4 Width
Radar CH5 Delay
Radar CH5 Width
DBL


Update Number of Integrations
4
Update Number of Integrations
2 [0..3]
IntLED
TF
1
22
Number of Integrations
U16
250
Sending Data 3
TF


0 [0..3]
Enable Sending Data Light
Sending Data 3


ResetLED
Reset the System
1 [0..3]
1
500
x21
x1
x1


3 [0..3]
Disable the Sending Data Indicator
Sending Data 3


8
Enable/Disable PRF
×23
1
PRFALED
?0:1
PRFAIndicator
PRFAIndicator