Building a High-Speed Data Acquisition System For Spectroscopy and Other Applications

A thesis submitted in partial fulfillment of the requirement
for the degree of Bachelor of Science with Honors in
Physics from the College of William and Mary in Virginia,

by

Gregory Millington Jones

Accepted for _____
(Honors, High Honors, or Highest Honors)

_____
Director: Dr. Cooke

_____
Dr. Griffioen

_____
Dr. Kossler

_____
Dr. Spitkovsky

Williamsburg, Virginia
May 2001

Table of Contents:

Abstract

This paper describes the design and construction of a high-speed data acquisition system interfaced to a personal computer. This research has involved first hand experience with high-speed electronics architecture, chip manufactures and their products, circuit board design and fabrication, circuit board construction and testing, microprocessor programming, and front-end software programming. This paper deals with the specific issues that arose while trying to adhere to our architectural goals and to bring this system from the early conceptual stages to a functioning viable product. It follows our progress from the earliest preliminary stages all the way to a successful completion, and outlines the further work that might be done refining and increasing the speed of our data acquisition board.

I. Introduction:

Recent development of high-speed chips has been driven by the cell phone and computer industries. The competition between personal computer companies and cell phone companies to meet the massive demands of their relative markets has caused incredibly fast paced growth in these areas. Cell phone and computer companies are being forced to constantly develop cutting edge technology at very low prices to stay competitive in their market place.

Conversely, the smaller laboratory environments of research and development have driven the production of high-speed data acquisition systems. The companies that produce these systems are forced to deal with high overhead costs, a small, specialized market, and low demand. Data acquisition system manufacturers have compensated for large overhead and small clientele by elevating their high-speed data acquisition system prices to tens of thousands of dollars. As a result of reduced spending, the development of high-speed data acquisition boards has been slowed.

Our research began based on the belief that we could build a high-speed data acquisition system for a fraction of the market price by using chips that have already been designed for the computer and cell phone industry to suit other purposes. The only element missing was the architecture necessary to combine the chips so that they could communicate with each other in the necessary ways. It was our belief that by using the latest surface mount soldering techniques and an architecture of our own design that we would be able to create a real, operating, lab-worthy system.

Thus, we developed a specific list of architectural criteria to use as research goals. We wanted a simple design, so our ideal architecture should have the fewest number of

chips and smallest number of connections possible. Using only a few chips also would help us meet our goal of producing a low cost system. Adaptability is important in the lab, so we decided to try to make our data acquisition board one that could easily be reprogrammed to work under different operating conditions. Finally, we wanted an architecture that is easily upgradeable with very little turnaround time. That way we could upgrade to the latest chips and keep up with, or even ahead of the state of the art data acquisition board manufactures, while still producing our system for a fraction of the commercial purchase price.

II. Architecture:

The main goal of any high-speed data acquisition system linked to a P.C. is to gather data at very fast speeds and transfer that data to a P.C. without data loss. It is illustrative to think of our architecture as a high-speed buffer. At each buffering stage our system decreases the speed of the data until it is slow enough to transfer to the P.C. To accomplish high-speed data acquisition we needed certain functional components. Although these components may be built from many different types of microchips they can be categorized as several specific universal functions.

Every digital electronic system that records analog data must convert that data to digital information. This step requires a component called an analog to digital converter. An A/D takes analog voltage levels as an input and samples the voltage level at a certain rate. Each sample is compared to a reference voltage and categorized as a digital number proportional to the input's amplitude at the time of the sample. To get an accurate picture representation of the analog input the A/D sample rate must be very fast. The purpose of a high-speed data acquisition system is to achieve the most accurate representation of the input as possible, hence the motivation for faster and faster systems.

Every high-speed data acquisition system also needs a microprocessor to handle certain functions like controlling the data taking process, performing data manipulation, and transferring data to a P.C. All microprocessors have a set number of lines that can be used for data input and output (I/O) as well as lines that can be set to control the digital logic built into the system.

High-speed data acquisition systems need a method to temporarily store data until that data can be retrieved by the processor and sent to the P.C. There are two primary

types of digital memory to choose from: RAM (random access memory) and FIFO (first in first out) memory. Each different type of memory has advantages and disadvantages that must be weighed against the specific requirements of the system.

All high-speed data acquisition systems need an on/off switch, or "trigger," that tells the system when to start taking data and when to stop. A trigger is usually a system of several logic devices and can vary in complexity. There are two basic types of triggers, internal and external. The most complex systems are built with both.

The following block diagram illustrates the basic way that data flows throughout my system.

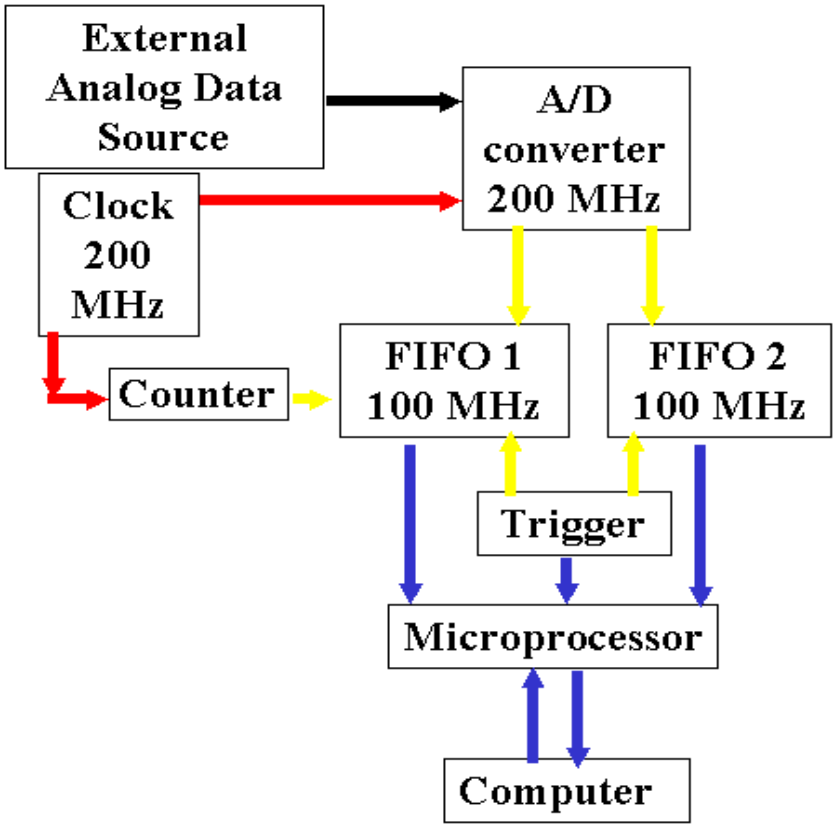


Figure 1. 200 MHz Architecture: Block Diagram of Data Flow

The A/D receives analog information from an external analog voltage source and converts the data to a digital signal. At the same time the A/D automatically halves the

signal speed and sends it into two memory chips to be stored. When the system receives a trigger signal, the memory begins to fill and when it finishes, the memory signals that it is done taking data. Once the memory chips are full, the microprocessor reads out the data stored in memory, manipulates it, and sends it on to the P.C.

III. Chip Selection and Acquisition:

To choose the proper chips for any electronics device it is necessary to compare the exact specifications of many different chips. Features such as timing, pin layout, operation, and power specifications are all considered. In the past chip manufactures published books full of information on microchips. Each chip was described at great length in a data-sheet. Now manufactures primarily distribute data-sheets through their websites and use of the data-sheets is free to the public. Web site distribution benefits the consumer because the information is much more current and readily available. Data sheets are the primary source material that I used throughout my research. See the appendix for a partial example of a data sheet.

We began the project intending to use two specific chips in our initial design: the P.C. interface chip, EZ-USB 8051, and the analog to digital converter, AD9054A. The EZ-USB 8051 can transfer data to a P.C. using a standard computer USB port. USB stands for Universal Serial Bus and is a recently created standard for communication with a P.C. One benefit to using the EZ-USB 8051 is that former graduate student Jon Curley already designed a board to run it and Dr. Cooke was already familiar with how to control and program it. The other benefit of using the 8051 was that we already owned several, so we didn't have to waste time acquiring them.

The AD9054A is produced by Analog Devices and can convert data at a maximum frequency of 200MHz. 200MHz corresponds to a speed of one sample every 5 nanoseconds. The AD9054A has a feature built in called a demultiplexer that halves the digital data output rate by sending data synchronously to two different output ports.

First, we began to look for a specific memory chip to store our data in the interim time between its digital conversion and eventual computer transfer. We decided that a FIFO would be a better choice than RAM, because a FIFO runs in a cycle. The major benefit to using RAM is that it can recall information in any order, from any data address in memory. But, this feature was unnecessary for our system because the purpose of our board is to simply record a data stream. On the other hand, a FIFO reads data out of memory in the same order that it was written into memory, on a **F**irst byte **I**n **F**irst byte **O**ut basis. The major benefit of using a FIFO is that instead of having to track data addresses like with a RAM chip, it is only necessary to cycle the FIFO memory to the first data point of interest. Once there, the whole sample can be removed in the correct order, at any desired speed. In our case that speed is dictated by the 8051 microprocessor.

I decided on a FIFO made by a company called Cypress. Cypress produces a few models of FIFOs that run at a maximum speed of 100 MHz (CY7C4292) and suit our architectural goals well, because they are loaded with special functions, helping reduce the need for additional chips. One such function is a programmable flag that can be set to signal the microprocessor when the FIFO has taken a specific amount of data. We thought that programmability would simplify our data taking process. In the end though we used the standard full and empty flags built into all FIFOs because programming the other flags increased the design overhead by requiring many more input lines.

Purchasing the FIFOs was difficult because the FIFO we chose turned out to be a specialty item. Chip manufacturers sell microchips through independent distributors, and most distributors are looking for big sales. They don't want to waste their time with small

time developers. Even the company policies of the chip manufacturers work against the small developer, because these companies have rules that they will only stock a certain chip if a minimum number, usually 100 or more, are ordered by the distributor. Distributors will only order CY7C4292s if they know they are going to sell the full minimum, in this case the minimum was 125. We only needed 2 or 3 FIFOs and the chips cost on the order of $75 to $100 making it impossible for us to order the minimum. Thus, we needed to find a distributor that had overstocked our chip when a big order came in. I finally found some CY7C4292s that ran at the 100 MHz speed level and we ordered three even though our system only needed two, in case one of them got damaged or destroyed during the board assembly process.

After our difficulty ordering the FIFOs, we added a new chip selection goal to our already growing list of architectural criteria: whenever possible look at largely used products first, because these products are much easier to obtain. The rest of the microchips that we used in the board design were made by Toshiba and were not nearly as difficult to purchase because they are standard digital logic that distributors routinely stock.

While searching the web we found another product that seemed like it would help streamline our architecture called the USBSIMMS board. The USBSIMMS is a pre-manufactured board that combines our EZ-USB 8051 microprocessor with external RAM and EEPROM memory and provides output connections via a 30 pin SIMMS socket.

At the time we had hardwired circuitry into our design that was very similar to the USBSIMMS board. But, we began to consider using the USBSIMMS instead, because a pre-manufactured board would greatly simplify our building and testing phases. By

purchasing the USBSIMMS board, instead of building one ourselves, we knew that we had a product that worked right out of the box. We didn't have to waste time building it and we knew that the USBSIMMS' half of our circuitry was 100% reliable, allowing us to narrow our focus if problems arose in testing.

Another benefit of using the USBSIMMS is that in addition to the microprocessor, the USBSIMMS also contains built in onboard memory. The onboard memory makes our board much more powerful, allowing it to do more onboard data processing before transporting that data to the computer. In essence it makes the brain of our system much larger.

Therefore, the USBSIMMS board became a motherboard to control our data acquisition system circuitry as a daughterboard. Since our circuitry is not hardwired together as a single unit, our design is more versatile and inexpensive over the long run. For example, in the motherboard/daughterboard case, if the USBSIMMS fails a new USBSIMMS easily replaces it. Conversely if the daughter board fails (even during construction) the daughterboard can be replaced. The separation of the motherboard and daughterboard is possible because the motherboard and daughterboard communicate at the slow rate of 1-4 MHz through 20 parallel lines configured into a custom bus.

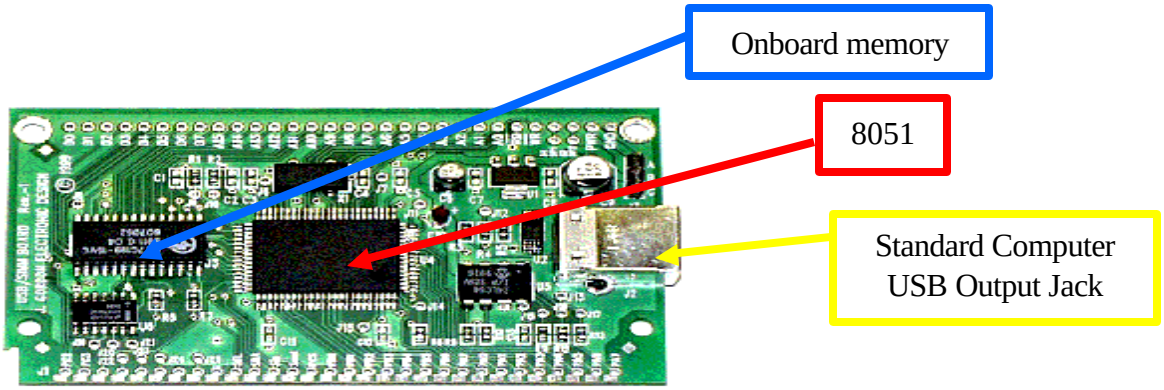


Figure 2: USBSIMM Board removed from the Daughterboard

Just like any other electronics product today, the USBSIMMS has a data sheet. It seemed at first glance that the USBSIMMS board would not work easily with our architecture, because it only has 20 I/O (input output) lines to work with. The raw USB8051 has 24 accessible I/O lines but the USBSIMMS uses four of these to control the onboard SIMMS memory chip. At this stage our architecture relied on 23 I/O lines and the loss of four lines prevented us from using the USBSIMMS board. So we decided to reduce the number of I/O lines our system sent to the 8051.

The current architecture at that time required 16 I/O lines for data input leaving only four control lines if we used the USBSIMMS. We needed seven control lines. To attempt to make the current architecture work with the USBSIMMS board we investigated the possibility of using a multiplexer. A multiplexer is a digital logic chip that switches simultaneously and quickly, connecting either of two input sources to its output lines. When a multiplexer is employed with proper timing, the number of lines can be reduced while still retaining complete data flow.

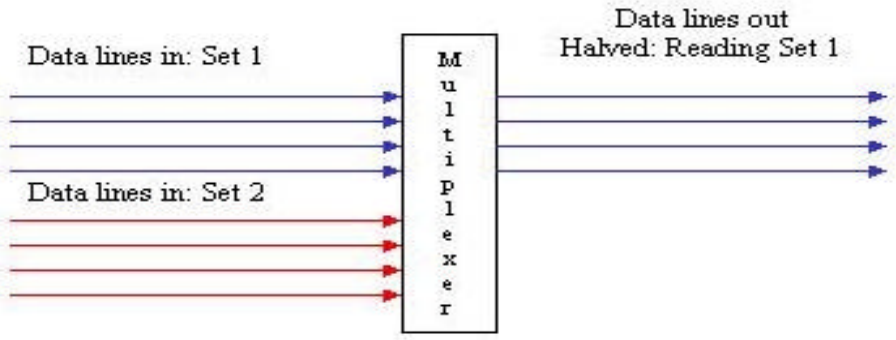


Figure 3a. Multiplexer State 1: Unswitched

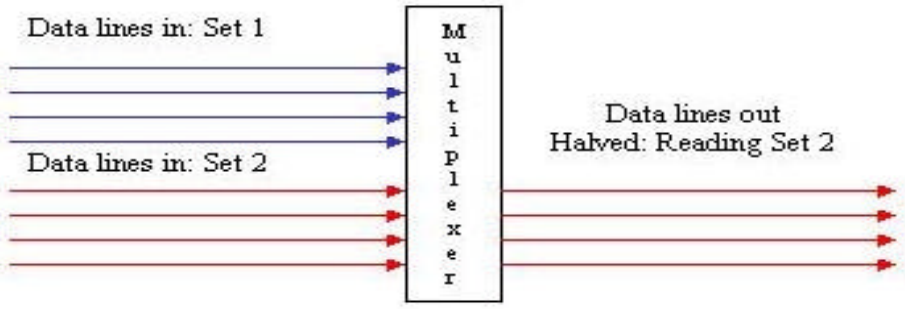


Figure 3b. Multiplexer State 2: Switched

Using a multiplexer would halve the number of data lines being dumped into the USBSIMMS to 8, leaving 12 lines for control, thus allowing us to use the USBSIMMS board in our design. But, using a multiplexer is contrary to our design goal of reducing our system to the least number of chips because the addition of the multiplexer adds chips and complicates the system. Using a multiplexer means we have to carefully control the timing of the multiplexer, in addition to all of the other components in our system, to ensure that no data is lost.

Throughout the project, sticking points like the question of whether or not to use a multiplexer, led to major changes in the architecture. The CY7C4292 has a feature called tri-state output mode. Tri-state output allows the user to tie the output lines of tri-state chips together without causing the chip to malfunction. In effect this feature allows us to tie two or more FIFO's data output lines together without a multiplexer. The main difficulty with tri-state output is that only one chip at a time may be enabled to read out data or data loss is caused. Using the tri-state feature allows us to use the USBSIMMS board and is a great solution in terms of our architectural goals because it reduces the number of chips in our system by 1 and the number of lines by 8.

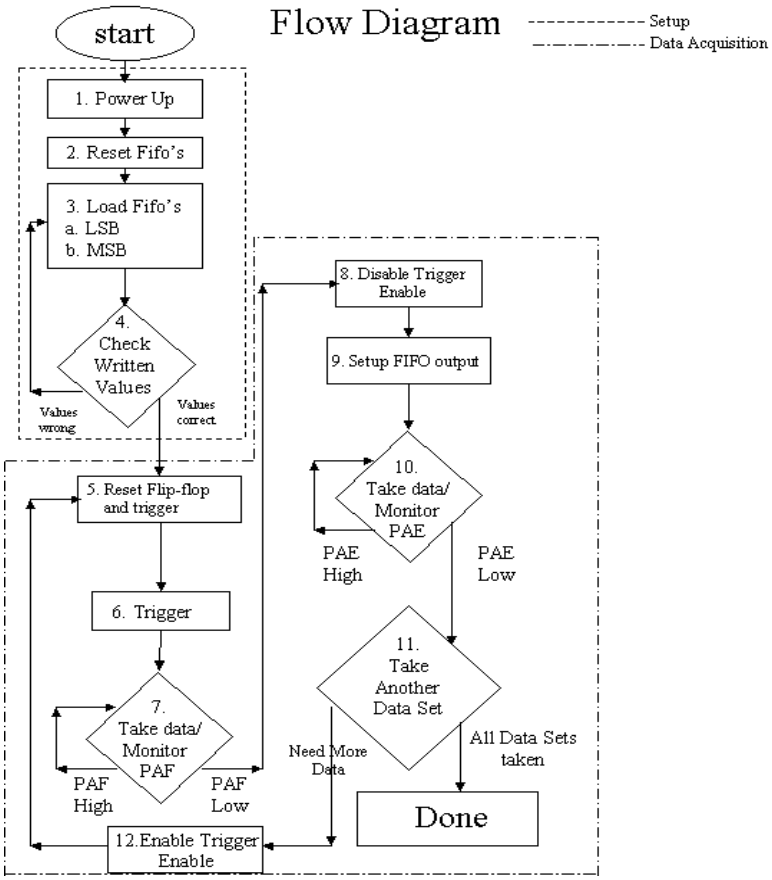


Figure 4: First Flow Diagram to Clarify Operational Procedure and Evaluate Data Line Usage

To make the final decision on whether or not to use the USBSIMMS board, I developed an operational flow chart. Each box on the flow chart contains a short description of the action performed at that step as well as a numerical label. The numerical label corresponded to a matching number on another sheet followed by a detailed description of what pins on the FIFO we need to manipulate to make the system perform each specific function. Using this technique I was able to see if our plan exceeded the 20 available control and I/O lines. From this flow chart we determined that we could step through our architectural plan and do everything that we intended using

fewer then 20 I/O control pins from the USB SIMMS if we used the tri-state output feature.

We also tested the USBSIMMS it in the lab because the 8051 built into USBSIMMS is a slightly newer version of the EZ-USB 8051 then we had used in the past. We needed to make sure that the new 8051 actually works the same way as our old 8051's because later in the project, we knew that our bank of accumulated programming knowledge would be important and we wanted to make sure our old programs still worked reliably with the new version.

IV. Board Design

The first steps to board design began during the chip selection phase. To get an idea of how the chips I chose fit together as well as to figure out how to obtain the smallest distance between chips, I photocopied and enlarged the pin diagrams of the fast buffering components (A/D and FIFOs) and taped them to a white board. The lines were connected using wipe board marker so that mistakes could be corrected and so that revisions could be made.

To design our board, we used the CAD (Computer Assisted Design) program Express PCB. Express PCB was written by Engineering Express, who also produce customized double-sided surface mount electronics boards. Express PCB was developed so that small-scale electronics enthusiasts and academic institutions could easily and cheaply design surface mount prototyping boards. Express PCB is free, however it comes with limited design features. For example, it has no simulation features, so we had to be very careful to avoid system-timing mistakes. Another limitation of the Express PCB program is that it does not contain nearly as many preprogrammed state of the art microchip pad designs as most professional CAD programs do. Express PCB's limitations coupled with a steep learning curve caused our original board design to contain several unintentional errors. In one case, the wrong pin spacing was originally used for the FIFOs because Express PCB did not have the proper spacing available as a built in feature.

The CY7C4292's pin spacing is much smaller then the pad programmed into Express PCB. In the end it was necessary to design a custom pad. In fact we were not even sure that Express PCB's resolution would allow lines to be drawn as close together

as our FIFO's pad required. The pins on the CY7C4292 FIFO are spaced .5mm apart with 16 pins to a side, and the program's minimum line spacing seemed slightly too big to even custom draw a FIFO pad for our CY7C4292.

We decided to try to correct for Express PCB's seeming lack of precision by making a correction every 3 pins to the line spacing. This way all the pins would be a little off, but the line spacing error would only compound over 3 pins instead of 16. Still, some uncertainty remained about whether or not this fix would work when it came time to solder the chips to the board. If too much inherent error still remained, it might cause solder bridges. A solder bridge is a tiny solder connection that unintentionally forms during soldering and attaches two pins together. After getting to know the program better, I finally found a solution.

Originally we were trying to lay down lines for the FIFO pad using the program's grid spacing feature and we were only able to be sure that the line precision was correct to one tenth of an inch. In Express PCB I discovered a way to preset line spacing rather than grid spacing. The preset line spacing feature allows the program to draw parallel lines whose distance is set with the precision of up to one thousandth of an inch. Using this discovery I was able to easily draw up a custom pad for the FIFOs that would be used to solder them onto on the daughter board with virtually no line spacing error.

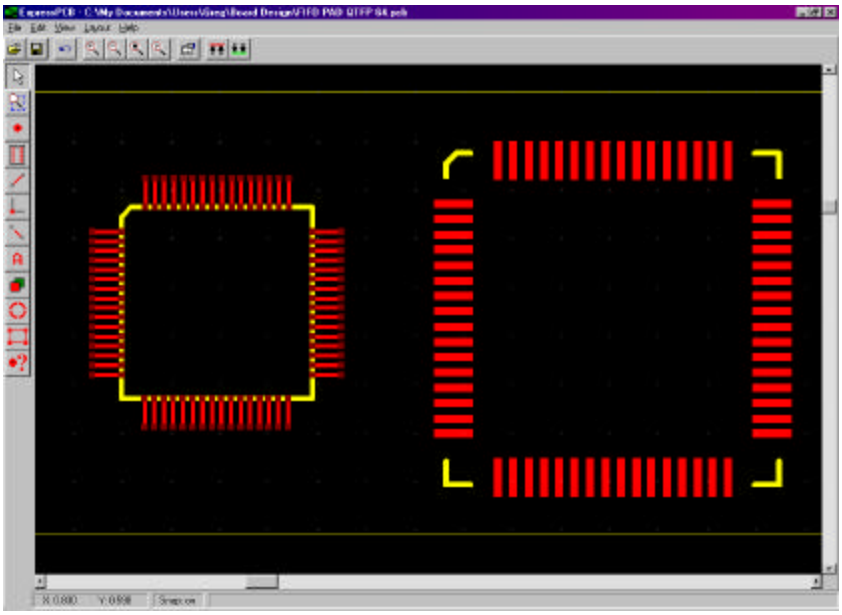


Figure 5: FIFO Pad I designed (left) and the wrong pad (right)

No built-in SIMM socket pad was included in Express PCB so I designed one using the functions I discovered while creating the FIFO pads.

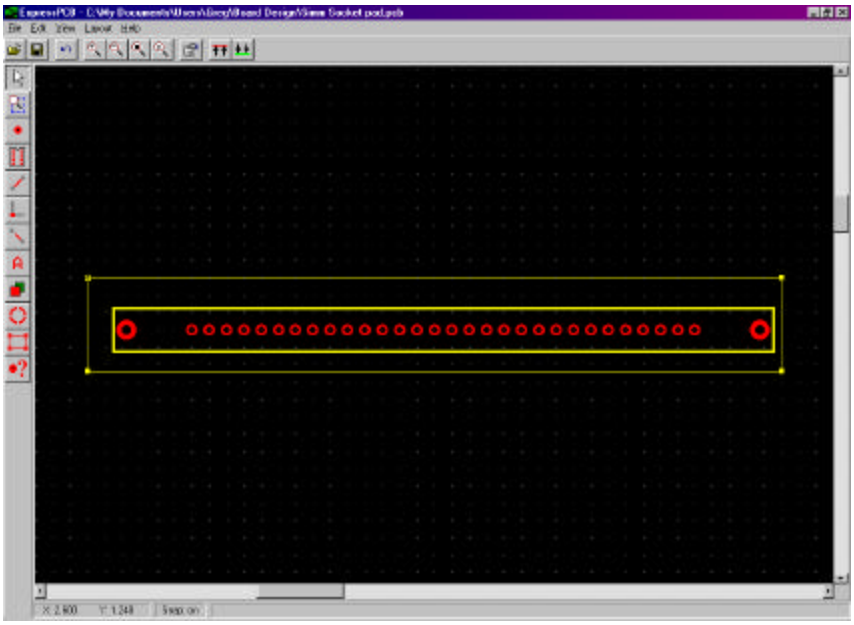


Figure 6: My SIMM Socket Design, Another Customized PAD in Express PCB

The next step in the design process was to develop a trigger for the data acquisition board. Although the concept of an on/off switch is simple, the trigger must be carefully developed, otherwise it can cause major problems. For example, unpredictable results might occur if the board was triggered while in the process of taking data.

There are two ways to design a trigger, internal and external. An external trigger relies mainly on external trigger signal generation while an internal trigger depends on an internally generated trigger signal. We chose to use an external trigger design because an external trigger fits better with current lab needs. However, one drawback to using an external trigger is that multiple triggering or missed triggers might occur. To combat this problem, we used a flip-flop as a gate to allow or disallow trigger signals.

When the flip-flop's enable line !PR is held low then it switches to its PRESET mode and sets its output high. The ! sign in front of the PR line indicates that PR is activated when held low, and PR is short for PRESET. When !PR is asserted high the flip-flop is enabled but waiting for a trigger signal. The flip-flop is able to switch its output level to low, but remains set to a high output level. When a trigger signal arrives at the CK pin (short for ClocK), in the form of a low to high TTL clock transition, the output line on the flip-flop switches to a low output level. The output line of the flip-flop is connected to the FIFO's Write Enable line (!WEN), and the FIFOs can only write data into memory when Write Enable is asserted low. Figure 7 illustrates this more explicitly.

$\overline{CLR}$	$\overline{PR}$	D	CK	Q	
H	L	X	X	H	Preset (Disabled)
H	H	Gnd.	Trans. Up	L	Enabled

Figure 7: FIFO Truth Table Conditions for Enabled or Disabled Trigger

An added bonus of designing the trigger this way is that the control line from the USBSIMMS can also act as a hard reset/abort. If at any time during the data taking process something goes wrong, the user could ask the program to de-initialize the !PR line, and the FIFOs would not be able to write any more data.

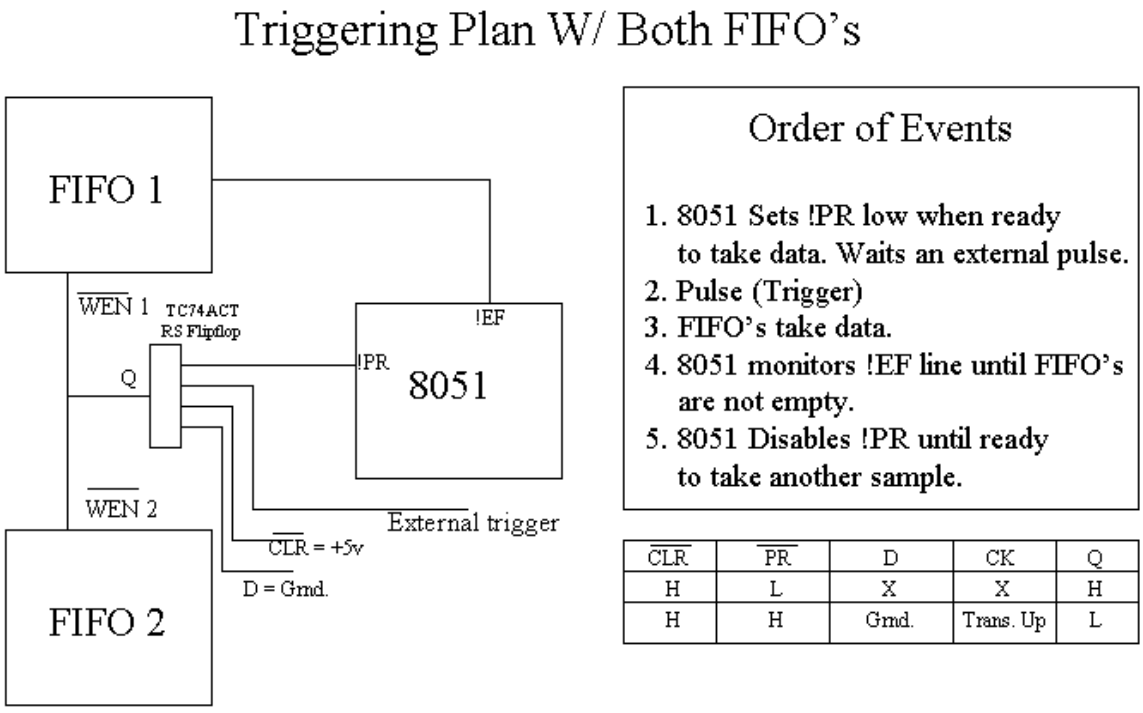


Figure 8: Trigger Block Diagram w/ Operational Explanation

While examining the logic for the trigger, a design problem arose. Programming the FIFO flags involves inputting 8-bit flag numbers to each FIFO. Sending this number to the FIFOs is a problem because it must be sent to the same pins on the FIFO that are used for our fast data lines. To send this number meant interrupting our fast data lines with more chips to multiplex the 16 lines together. The problem with interrupting the fast data lines at speeds of 200 MHz is that the high-speed input data could be messed up if the multiplexer caused even a short lag. Plus, adding unnecessary chips runs counter to the original design goal of using the fewest chips possible.

Rather then use a multiplexer we decided to modify the trigger design. The FIFOs run an order of magnitude faster then the 8051 microprocessor, so the time lost by allowing them to completely fill is inconsequential. In fact, both 128K FIFOs fill in just over 2 milliseconds. Because the first data written into the FIFO is also the first data removed, the last data is discarded making the transfer of the data to the computer more efficient. In essence we completely eliminated the programmable flag use from the system in favor of using the simpler full and empty flags.

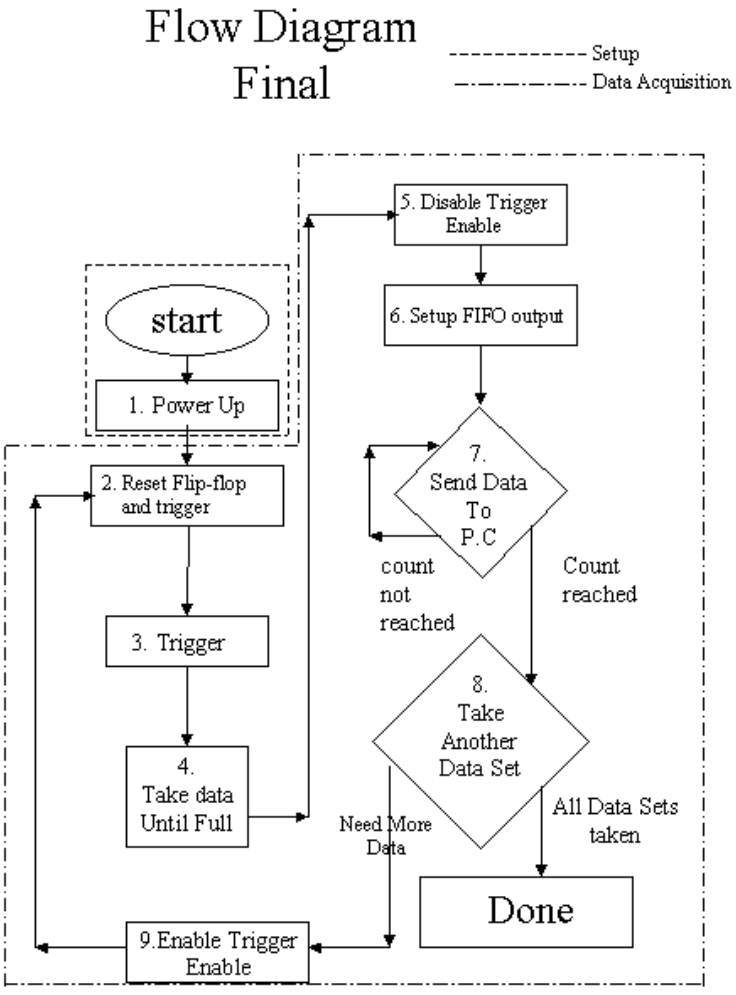


Figure 9: Final Architectural Flow Diagram To Clarify System Operation

This solution satisfies our architectural goals because it reduces the number of necessary chips and lines, as well as making our system simpler. When viewing the final flow chart it is easy to see that the system is simpler. Three unnecessary steps are cut out of the setup process virtually eliminating it all together, as well as making the data capture operation (step 4) much easier to program later during the programming stage of the project.

I made the finishing touches to the first prototype board by shortening line lengths. Large loops can cause noise so I eliminated them by moving small portions of the power line to the top surface of the board and connecting the tri-state output lines of the FIFOs closer to the FIFOs themselves before feeding them up to the USB socket. The "snap to" function in the Express PCB program helped optimize the distances between horizontal lines so that the lines were very close together, further shrinking my lines. I also laid more ground plane to reduce the amount of noise that might be present in the system due to the high-speed chips.

After all this, I still detected some last minute problems. For example, what I originally thought was a clock output line from the A/D that could be used to drive the FIFOs instead turned out to be an input line on the A/D. The FIFOs require a dock at one half the speed of the A/D to operate, and halving the frequency is usually performed by a chip known as a "divide by 2" counter. Fortunately for us, a counter we were using at one point in our trigger could also perform this function. In the design program I made the minor corrections necessary to accommodate the change.

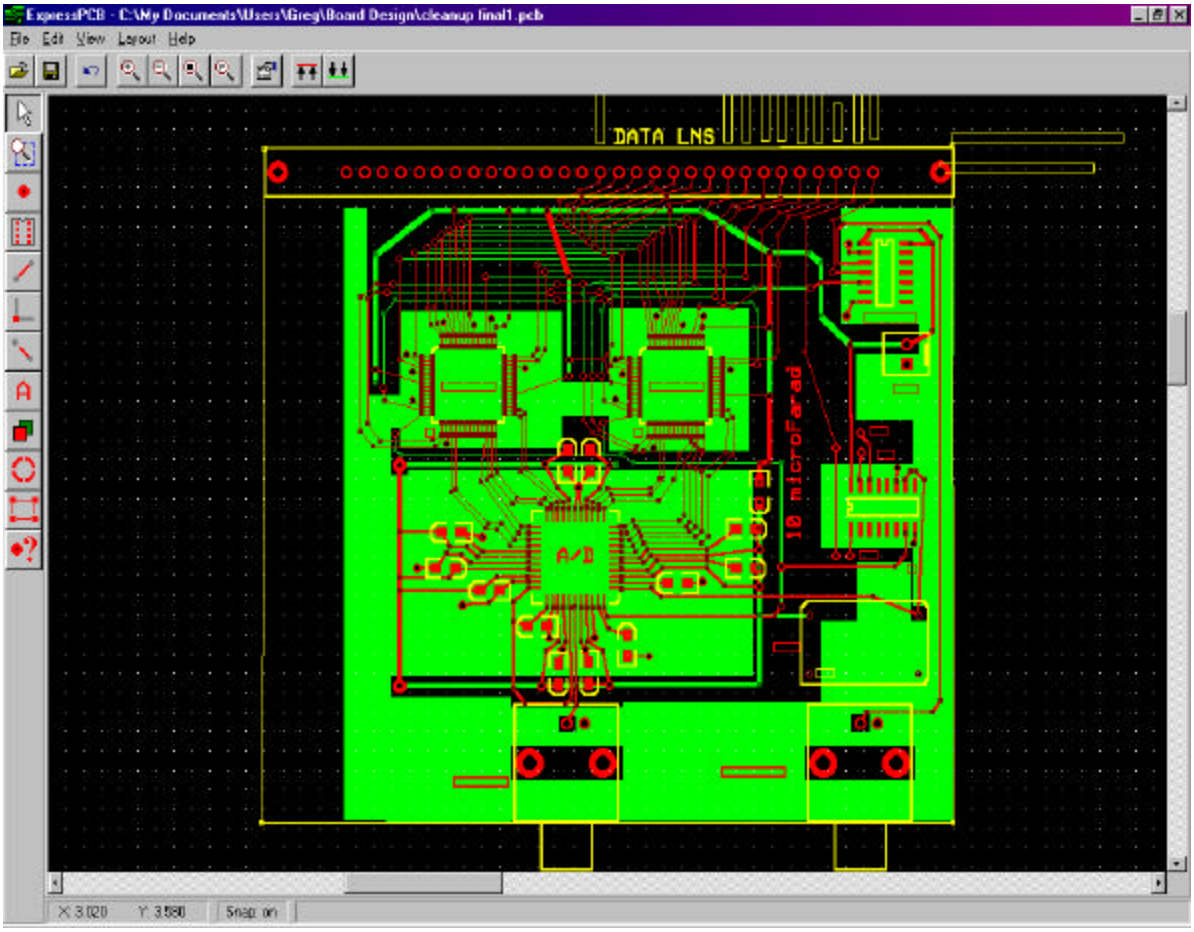


Figure 10: The Mark I Board Design, used to create the first prototype board

V. Construction and Testing

To construct state of the art microchip boards a technique called surface mount soldering is necessary. In the past chips were made in packages that had large pins and could be easily soldered onto into place because the pins were placed through holes in the circuit board. This design and chips that rely on it are known as thru-hole. As microchip speeds have increased historically they have shrunk in size. The very newest state of the art chips are so complicated that they are not even produced in thru-hole form. Instead these chips, such as our FIFOs are created in a form called surface mount. They attach to circuit boards that are flat plastic with thin solder traces etched onto it where the chips connect. The chip is laid on top of these traces and the traces are heated, creating a physical connection between the pin and the board. With small chips surface mounting is essential because traces can be etched onto the board at much smaller spacing then holes can be drilled.

Major electronics manufactures now perform surface mount soldering on the large scale with machines called Wave Soldering Stations. These machines place and attach all of the components at once with incredible precision. They treat the boards so that solder will only stick where it is supposed to and then run liquid solder over the entire board attaching all of the pieces at once. On the other hand, the small time electronics enthusiast when prototyping must painstakingly solder each part by hand, using a soldering iron to attach the parts and a magnifying glass to be able to see the parts clearly, because they are so small.

We chose to solder our components in an order based on two main criteria, the availability/cost of the chip and the difficulty level of soldering the part. For example the

FIFOs have the smallest pin spacing and are the most expensive so we decided to solder them last. This method also gave me plenty of easy practice while I geared up to solder on the difficult parts. While we were attaching parts we also testing and continuously checking for errors that might cause us a lot of trouble if overlooked until later.

We initially ordered two Mark I prototype boards. By the time they arrived, we had all of the necessary components to start production on the Mark I prototype. Every component on the board requires a 5V power supply so the first step I took was to solder into place the power junction used to power the board. I checked with a voltmeter to make sure that all of the component's power lines had power. They did not. A major power line and some secondary power lines had not been run properly.

Inevitably during prototyping, some layout errors will be caught that were not caught prior to production. To save time these errors are corrected manually on the prototype and later fixed in the CAD program for future iterations. Our power problem is a case in point.

Standard procedure for clean power supply calls for double bypass circuitry of one 10 microfarad capacitor and multiple 0.1 microfarad capacitors to be placed as near as possible to the A/D. In our initial design I mistakenly had not connected two of the .1-microfarad capacitors to ground and had used the 10-microfarad capacitor as a power feed rather then power by-pass. We drilled some small holes through the surface of the board to run jumpers to ground from the ground plane on the opposite side of the board, correcting the problems with the .1-microfarad capacitors. To correct the power feed problem, we jumpered the disconnected power line together and placed the 10-microfarad capacitor in a different area on the board.

Soldering the clock into place was not difficult because it was designed as a thru-hole device and only consisted of 4 pins. A slow 10 MHz clock was selected to make our system easier to test and we replaced it on the Mark II board with a much faster clock. After soldering the clock, I tested that it was operational by using a scope probe to examine the output. It worked perfectly and we saw a good square wave output at 10 MHz.

Next we chose to solder on the counter because it has only 16 pins and they only cost about a dollar per chip. Soldering the counter was easy and to test it I checked the solder connections for continuity with a voltmeter and then examined the counter's output with one of the lab's oscilloscopes. We could see that each pin on the counter was outputting the correct frequency square wave.

Soldering the flip-flop (TC74ACT74FN) was also easy but I was concerned about testing it, because we needed a more complex setup than was necessary to test the counter. We wired up a breadboard so that we could use its built in switches to control voltage on several different lines and attached these lines to the lines on our prototype board that run to the flip-flop. We asserted the lines to the PRESET levels and checked the output. As expected it read High. We enabled the flip-flop and the line still read high. Finally, we sent CK a low to high transition from the breadboard and the output made a successful transition from high to low.

The A/D was the first difficult chip that we soldered on. The A/D must be powered cleanly so that it digitizes the analog input properly. So, before soldering on the actual A/D I had to solder on 11 capacitors to regulate the power supply. At this point I made my first soldering mistake.

Capacitance is additive. Once soldered onto a circuit board, each component's capacitance cannot be checked individually. I mistakenly soldered on all of the tiny .1-microfarad capacitors before realizing that the only way to check if they were functional was to check that the board capacitance increased .1-microfarad after each was soldered into place. After soldering them all on there was no way to verify with any certainty that I had successfully attached them all. However the total capacitance did seem to be in the right ballpark. We corrected this error on the Mark II prototype.

I soldered the A/D onto the board and preliminarily tested my work using the voltmeter to check for continuity and to make sure that I had not created solder bridges between pins. I tested the A/D by looking at its output lines to see if they were making transitions when an input signal was applied. The AD9054A requires an input voltage range between 2-3 volts so I used a voltage divider and the variable resistance pot on one of our breadboards to wire up a makeshift variable voltage supply. Turning the pot changed the voltage smoothly between 2 and 3 volts. I ran a BNC cable from our daughter board input to the voltage supply and we set up the lab's fast scope to watch the output lines from the A/D. Twisting the pot varied the input voltage causing clear transitions between high and low on all the output lines of the A/D. Although we couldn't be sure exactly what the numbers the A/D was sending out, it at least appeared operational.

Lastly we soldered on one of the FIFOs. We only soldered on one because only one was necessary to check that the board architecture and timing was working correctly. Soldering the FIFOs was the most concerning step in the board construction process mainly due to the FIFO's 0.5mm pin spacing. No one in the lab had ever soldered

anything smaller then a 1.2mm pin spacing before, and we were still not sure that it would even be possible to solder the FIFOs by hand. With such a small pin spacing solder bridges are much more likely to occur and much harder to detect by sight. Plus, we weren't even sure that all of our FIFOs still worked anymore because they had been stored since the summer and CMOS chips are very sensitive to static discharge.

Soldering the FIFO was harder than any of the other logic. Even getting the chip into position was difficult. Because the leads on the board are raised slightly, the chip would slip off of the leads and to the side to rest on the plastic board between the leads. Flux, which is used to clean the leads so that the solder will stick, also usually helps to keep the chip from moving out of position because it is sticky. But in this case, it actually made the chip harder to position by making it difficult to slide the chip up onto the pad. Several times I was able to get the FIFO in place, only to accidentally push it off the pad as I tried to apply pressure to the top of the chip, to keep it from moving while I soldered. Finally I secured the chip with top pressure and tacked a few of the non-functioning legs to the board. These legs are called N/C or "no connects" because they don't connect to anything inside the chip, but they remain as part of the standard chip package design.

Once I had soldered all of the pins, I checked all of the pin connections for continuity and found that I had only successfully connected about half of the legs. I spent even more time soldering and testing with the voltmeter until finally it said that all of the legs were connected. However, the FIFO still needed to be tested further because the voltmeter could not tell us whether or not the chip was actually operational, and the possibility for capacitive coupling, pressure connections, and cold solder joints still existed as well.

To test the FIFO we decided to program the 8051 microprocessor to control the FIFO while we checked that it was operating correctly. Programming the microprocessor requires an algorithm describing the functions performed by the microprocessor in clear, step-by-step order. We created an algorithm using our flow chart that described every step in order. At each step we determined what level, either high or low, that each control pin must be set to. From the pin settings we created a single binary number. In the end our list of numbers was the basis of our algorithm. For each major algorithmic step we created a function in the program that sent out the binary numbers that we created. We were then able to watch the FIFO's full flag (!FF) and empty flag (!EF) as indicators of whether or not our program worked as anticipated.

Empty		Partially Full		Full	
!EF	!FF	!EF	!FF	!EF	!FF
0	1	1	1	1	0

Figure 11: FIFO Empty and Full Flag Operation

The truth table in Figure 11 illustrates the different stages of data acquisition and the resulting levels of the FIFO's flag lines. When reset, the FIFO enters the empty state. To test that the FIFO was actually reading in data, we checked that the flags had switched to either the Partially Full state or the Full State. In the Mark II version of the prototype, we sent the !EF and !FF lines to the USBSIMMS so that we could check their operation with the computer instead of with the scope.

For about five minutes our Mark 1 board operated in a way indicating that it was fully functional. We left the variable voltage supply attached to the board input and were successfully reading in data with our board, but the levels we read out of the board didn't seem to make sense. We realized that we had reversed the lines going into the FIFO and

that we were reading the hex numbers in reverse, so in the microprocessor program, we wrote a small function to resort the data. For Mark II we corrected this problem by physically reversing the lines in the design.

Next we realized another problem. One of the bits seemed to be sticking so that it sometimes read high when it should read low. We would reset the chip clearing all of the memory and then read in data. If we read in the maximum voltage of three volts then all of the lines went high. However, if we then changed the voltage back to the minimum input voltage all of the lines should have returned to the low state, but one would stay high consistently. Soon multiple bits were sticking and finally the whole chip just stopped working completely. Our conclusion was that one FIFO had gone bad. In the Mark II board this problem never occurred so we never had to fix it.

After the Mark I board failed, we built the Mark II and for a day, also believed its FIFOs were not operating properly. When reading the empty and full flags the FIFO seemed to be locked in a state that was not allowed on our FIFO logic diagrams. The FIFO would reset properly but when told to take data, it would enter a state where both !EF and !FF lines were low. We thought that this state was impossible because the FIFO could not be completely full and completely empty at the same time. After carefully reviewing the FIFO data sheet, we realized that we made a programming error, causing both FIFOs to act improperly. The !EF flag will only be updated after the FIFO receives a read clock (RCLK) pulse, so this state was occurring because we were not updating the flags. Once we added this clock signal into the program we corrected problem and the FIFO began to operate in the manner we expected. Our board was running successfully!

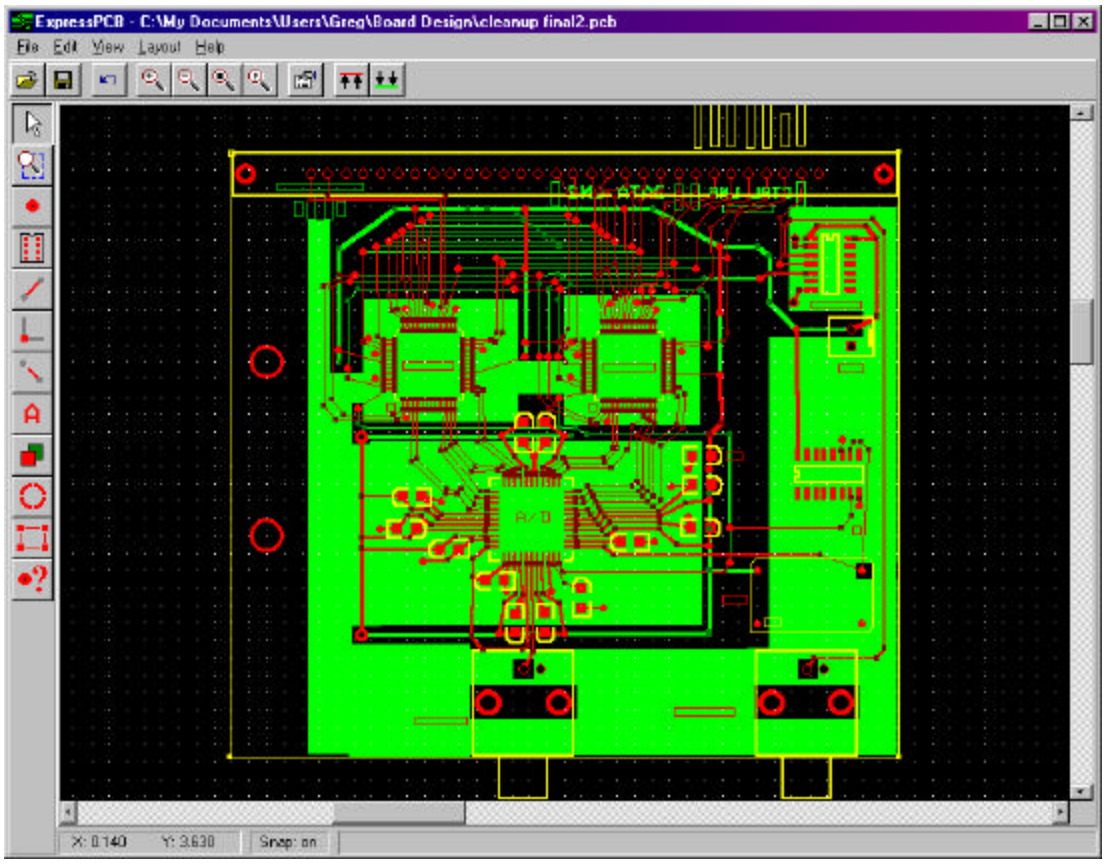


Figure 12: Mark II Board Design

VI. Programming the Front-End:

A front-end program that runs the data acquisition board is vital to evaluating the results of my work. It acts as the supreme acid test by performing all of the operational steps at once, just as is necessary in a laboratory experiment. The data is sent out to the computer and the front-end program plots it in graphical form for examination. Although, programming the front-end in its full complexity is actually beyond the scope of my experiment, I participated in programming a simple front-end to evaluate my device.

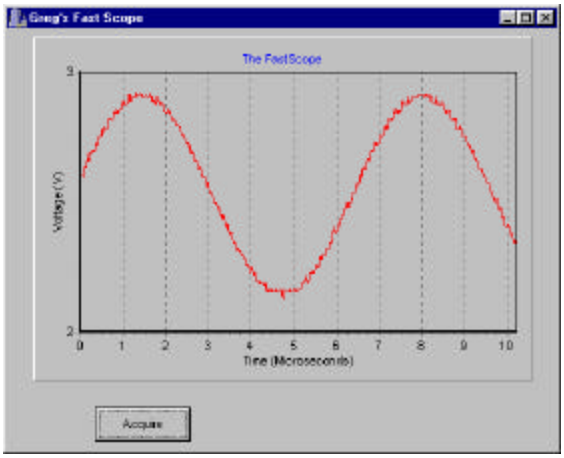
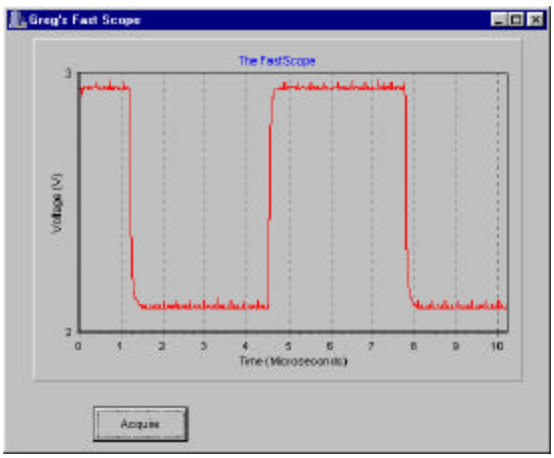


Figure 13: A Function Generated Sine Wave taken by our Data Acquisition System at 100 MHz



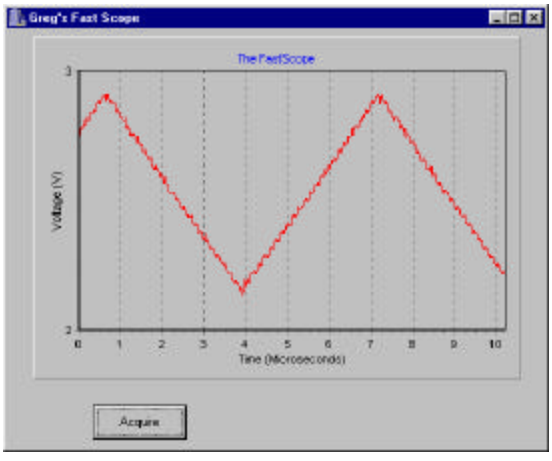


Figure 14 and 15: Sampled Square and Triangle Waves at 100 MHz

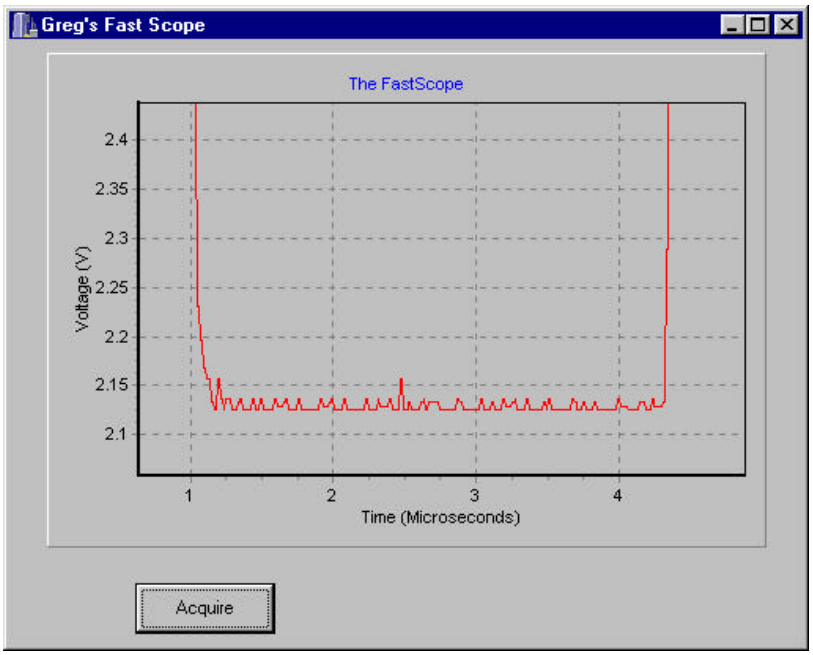


Figure 16: Zoomed In View of the Square Wave Picture Taken by Our System

In all of the figures some digital error is visible. With any digital system a digital error of 1 bit is to be expected. However, when zooming in much farther, as in Figure 16, a two-bit error is visible in our system. As of this point in time we are not sure why. Possible causes may include cross talk from the high-speed clock, noise in the power supply to the A/D, or noise on the input line.

VII. Conclusion:

This project was largely successful. We created an architecture that in future projects could easily be modified to work with faster and faster chips as they are developed. Our current design can be quickly modified to produce a new state of the art scope using very little extra effort beyond the groundwork that I set up in this project. Using our architecture and the dropping prices of chips to our advantage should allow us to design data acquisition boards in the future that could be built by subcontractors for prices on the order of a few hundred dollars, rather than the tens of thousands of dollars that the industry charges for comparable devices. Most importantly however, we have proved that prototyping this architecture is possible and the board works just how we expected it too. Plus, with a little testing and extra electronics design, this board could actually be turned into a working product for our laboratory.

Another important result of my project is that in setting the groundwork for my architecture, I also helped to lay the groundwork for revision of the digital electronics lab curriculum at William and Mary. My work on understanding the USBSIMMS has shown us that it the USBSIMMS board would be a fantastic tool to work with in the lab, and it would make a whole range of new experiments accessible to the undergraduate physics students here.

I have made it very convenient for someone to follow my work by creating a step-by-step guide to producing my system or one similar to it. That way, if someone follows up my project they will waste much less time getting started because the learning curve will be diminished and they will not have to relive every problem I faced during this research project. The elements of every data acquisition system are universal so even in

the eventuality that someone were to take a completely different approach with the next speed level, they could still learn from my mistakes and incorporate my learning into their own ideas.

As with all experiments, my project could be followed up by another student for further research. Work must be done to test our board and resolve the source of the extra bit level noise by examining the power supply and input sources more carefully. A pre-amp and internal power supply needs to be added to make our system more robust and help reduce noise on the input line while running the board in non-optimum operating conditions. Also the front-end programming could be expanded to include more features. Testing might also be done on my board to see what its actual maximum speed is before timing errors occur. Right now the board runs at 100 MHz, but theoretically it should work at up to 200 MHz just by adding a faster clock. Finally, some future student might want to produce a Mark III version of my board that resolves the extra bit error and runs at the full 200 MHz. This Mark III board could then take the place of the lab's current data acquisition board.

References:

Analog Devices: Data Sheet AD9054A 8-Bit A/D converter.
http://products.analog.com/products/info.asp?product=AD9054A
Revised 04/01

Cypress Semiconductor: 128k x9 Deep Sync FIFOs with Retransmit and Depth Expansion
http://www.cypress.com/cypress/prodgate/fifo/cy7c4282.htm
Revised 11/06/ 97

J. Gordan Electronic Design: Data Sheet USBSIMM
http://usbsimm.home.att.net/info/info.html
Revised 09/22/00

Toshiba: Data Sheet TC74ACT74FN Dual D-Type Flip Flop with preset and clear.
http://www.toshiba.com
Revised 08/08/97

Toshiba: Data Sheet TC74VHC4040 12 stage Ripple-Carry Binary Counter
http://www.toshiba.com
Revised 01/30/97