fil

VLA Interference Memo #17

NRAO Clock Recovery Test Fixture
Senior Design Project

Francis Martinez
Matthew Nunez
Steven J. Padilla

Revision 3
May 9, 2001

Contents:

Appendices:

Introduction

The output of the Very Large Baseline Array (VLBA) tape recorders is made up of 24 differential digital data signals plus a differential clock signal for each of the channels. The operational bandwidth of these signals is from 100Khz to 4.5Mhz and is transmitted over 100 feet of cable. The VLBA correlator uses the clock signal from each channel to decode the data signal into a clean serial data stream. The signals come from the headstack, the collection of magnetic heads that play and record the data, then are amplified and filtered and sent to the clock recovery/driver board. An accurate operation of the VLBA recording/playback system depends on the correlator's ability to decode the data stream with the clock recovery board.

This project consists of designing and building a test fixture to characterize the performance of the clock recovery/driver board in the lab. Two of the 32 channels need to be compared to one another. Given the same data input, each of the 32 channels should have a similar output, (within the present error-rates). If the channels have some sort of difference between them, a counter will count that as an error. An indicator of error rate difference between the channels will be provided. A maximum error rate of 10^{-3} (approximately 22 kHz of the 4.5 MHz signal) is acceptable. Once this error rate has been detected, an indicator will flag telling the user that an unacceptable error rate has been detected. It can be deduced from this information that one of the two data streams are incorrect and that one of the two clock recovery circuits may not be functioning properly. All circuit designed must be able to function properly within the bandwidth of the

data signals. A method of simulating the 100' long cable impedence must also be designed and implemented.

A computer interface including a Graphical User Interface (GUI) will also be designed to allow the operator to select the two channels to be compared. A real time error rate will be displayed at all times to allow the operator to analyze the error rate at any moment in time. Another option will be available to allow the user to compare each channel to all the other 31 channels. A report file will be generated containing the data generated from the comparison of all 32 channels.

Clock Recovery/VME Card

The clock recovery board is a significant subsystem to the entire correlator system at the NRAO. A functioning clock recovery circuit is crucial to the operation of the VLBA recording/playback system. Because of this, it is necessary to test the clock recovery circuits to determine if they are functioning properly.

The purpose of the clock recovery board is to take a serial data stream, which is out of sync, and to recover a clock signal from it. Once the new clock has been recovered, the clock is used to re-sync the existing data. This allows a "noisy" and un-clocked data signal to be input to the clock recovery board and the data signal will be output from the board re-clocked with minimal noise.

The comparator circuit and all its associated parts are to be wire-wrapped on a standard VME wire-wrap card. The VME card is being used to house the test circuit, but its bus is not being utilized. The purpose of the VME card is to house the test circuit in a concise and organized manner and to provide the test circuit with VCC and ground. Therefore, the VME card had to be equipped with the circuit components and all necessary components to interface it with the computer.

To interface the test circuit with the computer, a GPIB controller was used, and thus, this had to be mounted to the VME card. However, it was mounted directly to the HC12 evaluation board which was mounted to the VME card. To interface the test circuit to the NRAO circuitry, a set of clock recovery connectors was necessary to input the clock recovery data to the test circuit. To input the

test data stream into the NRAO circuitry, the data has to be input to the parallel introduce card. This also requires the necessary connectors for the parallel reproduce. The VME card was also equipped with the 3 Altera chips that were used to house the multiplexors and the comparator circuit.

The final components that were installed on the VME card were an array of differential line receivers. Before the data can be compared using the comparator circuit, the data must be manipulated so that it can be manipulated by the comparator circuit. The difficulty arises from the fact that the output from the clock recovery card is differential. Due to this, a differential line receiver must be used to take this differential data and convert it to single-ended data. This The line receivers also required D flip-flops, pull up resistors and terminating resistors to allow them to function correctly. Once the receivers were complete, the data was single-ended, and thus, able to be compared using the comparator circuit.

Comparator Circuit

To determine the error rate of the two data signals, an Altera signal generator was used to produce a data stream. This data stream was injected into all 32 data signals through the parallel reproduce board which takes the data stream and amplifies it to be fed into the clock recovery board. Since the same data stream is to be input to the parallel reproduce board, all 32 differential data streams coming out of the clock recovery board should be identical. Because of this fact, the data exiting the differential line receivers are also identical and can be checked for errors. Theoretically all 32 data streams should be identical and clocked together. However, if one of the clock recovery circuits is not operating correctly, the data could be out of sync and incorrect.

Because of the identical data streams, it is possible to do a bit-by-bit comparison to determine if the output data is identical from all 32 channels. To perform this analysis, the most appropriate and most obvious method seemed to be to compare the two data streams directly. It appeared that an XOR gate could be used to perform this task since the output for an XOR gate is high when a 0 or 1 and 1 or 0 were present. This would allow us to identify a non-identical pair of data.

The core of the comparator circuit is the XOR gate, however there is some extra discrete logic necessary to generate an error rate. A 2^17-bit counter (total bit counter) is used to count the total number of sets of bits that have been compared. The 17^{th} bit is used as an "overflow bit" that will tell the HC12 that it has counted that high. As the total bit counter is counting, a second counter, the

error counter, is also performing a count. This counter is being driven by the "highs" being output by the XOR gate. The error counter is counting the total number of errors in the two streams of data. At the output of the error counter is also a 16-bit latch. When the 17^{th} bit of the total counter overflows, it tells the latch at the error counter to latch the value in the error counter. The HC12 can then retrieve this value and determine an error using the value in the error counter and the highest count the total counter will count to before overflowing. The HC12 then resets all necessary counters and latches and the process repeats itself.

The HC12 is to used memory-mapped I/O to access the different components of the test circuit. Because of this, the comparator circuit has also been given an address range to call it up without having to confuse it with the other components of the system. Because of this, a 3:8 decoder was used to do this. To turn the decoder on, the E-clock has to be high and the read line has to be low. The A, B, and C bits on the decoder also correspond to address bits A15, A14, and A13 from the HC12's address bus. The combination that gives 0, 1, 0 for A15, A14, A13 respectively enables the data to be read by the HC12. This combination allows the comparator circuit to be enabled with an address range of $4000 to $4FFF from the HC12. This combination will write a high to the output pin 2 of the decoder chip. This bit is tied to an array of tri-state buffers that are connected at the output of the latch. This allows the data to be kept off of the bus until the HC12 asks for it to be read.

The comparator circuit poses many timing issues that must be accounted for to allow the comparator circuit to function correctly. If the HC12 tried to ask for the data before it was correctly latched, the data the HC12 read might have been incorrect. The error latch had to be equipped with a way to let the HC12 know that it was latching the data and was not ready to be read. To do this, a 16-bit counter was used, with the clock input being one of the recovered clock signals from the clock recovery board and the high bit on the counter being used as an overflow line. When the total error counter overflows and tells the latch to latch, the overflow line resets the counter which has been counting and has a high at its high bit (overflow bit). This high bit is what is being used to tell the HC12 that he data is ready. Once the overflow bit comes from the total bit counter, the "busy" counter begins to count. Once the high bit on the counter goes high, the HC12 knows that the data is ready to be read. The high bit is also tied through an inverter to the enable on the busy counter which disables the counter when it goes high but keeps its last count on the output pins of the counter. Therefore, the high bit (or busy bit) on the busy counter stays high whenever the data is valid. The HC12 can retrieve the data at any time that the line is high. Once the total bit counter overflows again and tells the latch to latch, it again resets the busy counter. When the busy counter is reset, the high bit is turned low which enables the counter again due to the enable line seeing a high through the inverter. The process is continually occurring and thus the HC12 knows only to retrieve the data when the busy line is high.

The total counter overflow line is also used as a way to 'write' the latch and reset the error counter. However, the error counter must be cleared after the latch has been latched. To do this, both the error counter and the latch 'write' line were tied to the total counter overflow line. However, the line that went to the error counter reset branched from the total counter overflow line and was equipped with a series of Altera LCELLS between the overflow line and the reset line. This causes a small delay between when the latch has been written and the error counter has been reset. This allows the latch to be written before the error counter is reset so the data is in the counter long enough to be latched.

The latch must also eventually be reset. The method in which this is performed is by using the HC12. When the HC12 polls the latch for the data, the address values to access the data contain the bit pattern '010.' Because of this, the 1 can be used to reset the latch. This means the HC12 will read in the data from the error counter latch and then it will reset the latch using the same command from the HC12.

Altera Design

One of the requirements of this project was to build a multiplexor to select two channels, out of thirty-four to compare to each other. One of the options that presented itself involved a circuit that contained many pieces of discrete logic interconnected to each other on the back of the wire-wrapped VME board. Because of the complexity involved and the probability of error involved with wire wrapping each of the chips to each other, it was easy to decide to look at other options. The option that we felt fit best for the project was the use of Altera programmable logic. By using programmable logic, the complexity of the multiplexors was gone and the size was whatever was needed. After discussion, it was also decide that there were other parts of the project that would be much easier to accommodate into an Altera chip. These parts of the project included the circuit that makes up the comparator circuit and the random bit generator. In this section we shall discuss the design steps taken in the design of the Altera portion of the circuit.

The first part of design of which Altera was chosen to implement were the multiplexors. We were provided with a circuit that could be used as a multiplexor for all 32 channels. Using Altera to design a new multiplexor as a single circuit seemed as the best option because of the time it would save and the fact that it is much less complex than wiring together digital logic. The multiplexors were written as a text design file in Altera following a basic form and then applying this form for the rest of the design. Each multiplexor is made up of thirty-four inputs,

six select lines, which select the correct channel, and one output. The basic form of the multiplexor looks like this:

```
IF select[5..0] == B"000000" THEN Output 0 = Input 0;
ELSIF select[5..0] == B"000001" THEN Output 0 = Input 1;
```

The first line states that if the six select lines contain a binary low, all zeros then the output is whatever is at the first input (Input 0). Next, if there is a one at the lowest bit of the select lines (Select 0), then the output is whatever signal is at the second input signal (Input 1). There are thirty-two more lines like the second one that serve to select the correct channel.

The next step in the design of the multiplexors was to set two of them up to be ready to accept data. Each differential signal contains both a channel and clock signal. To get two different outputs to compare to each other, two multiplexors were placed together on a single chip with the same inputs going to each. Next, a new device was built to select first the multiplexor to be used and next, select the channel to be used. This device has seven input lines with twelve output lines. If the first input line is low then the top mux gets the information from the lower six input lines through the first six output lines. If the first input line is high, then the second mux gets the select data from the lower six input lines through the last six output lines of the select device. The basic form of this multiplexor design is illustrated in figure 1.

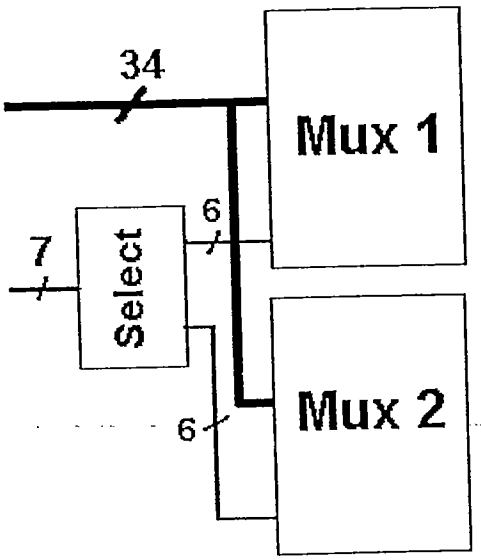


Figure 1: Basic Altera Block Diagram

This basic form was followed for both the channel and clock multiplexors with the names of the inputs changed to match the clock.

During the design of the select device there were problems with keeping the data from the non-current bits (the lower six if the first input is high and the higher six if the first input is low) valid. What was happening was whenever the first input switched from either low to high or high to low, the data that was current before the change was no longer valid once the first input changed. At first, we were unable to get anything at the output of the non-current bits. With experimentation we were able to get something that looked to work by setting the non-current bits to their previous value when they were not selected. Unfortunately this data oscillated between low and high rapidly between the actual changes in the data. The solution to our problem came by using a D flip flop as a latch to keep the data in the non-current output bits as valid. The way that the circuit now looks like in an Altera .gdf file is 32 inputs going into two 32-

channel multiplexors. There is a third device that is the select box that controls what each output is.

Two other parts of the project were also implemented in Altera. The first of these was the comparator circuit, which is connected to the outputs of the multiplexors. The design of the comparator circuit proved itself to be one of much complexity with regard to the necessary amount of discrete logic needed to implement it. Because of this, Altera was used to implement the circuit in an organized and concise manner. The error and total bit counters were written in Altera using a .tdf format. A symbol was created for each to simplify the construction of the circuit in Altera by the use of a .gdf (graphical design file). This allowed us to easily draw the circuit and insert discrete logic parts wherever necessary. The latch for the error counter was also written as a .tdf format and then turned into a symbol. The Altera format allowed for a complete redesign of the circuit if it did not work. This was possible because of Altera's simulation package. This meant the circuit could be tested before it was built. This also meant that incorrectly wired circuits could be re-wired in the Altera package with minimal effort and time. The use of Altera for the comparator circuit minimized the complexity and design time of the comparator circuit.

The next part of the project that was designed in Altera was the random pattern generator. The test pattern generator for this project is a pseudo-random bit generator. The output of the test pattern generator is a pseudo random pattern 256 bits long. After producing the 256-bit pattern the test pattern generator will repeat this pattern indefinitely. To allow the test pattern to be

viewed on an oscilloscope a trigger pulse is produced every time the test pattern repeats. The test pattern generator is implemented using an 8-bit shift register with taps taken from the outputs of the third and eighth flip-flops, XORed and fed back into the input of the shift register. In order for this circuit to run the shift register is loaded with 80 hex upon reset. Once the reset is unasserted, the external clock runs the shift register producing the test pattern. Every time contents of the register become 80 hex, the trigger pulse goes high until the contents of the register are changed with the next clock pulse. The ALTERA design, the simulation and the actual output of test pattern generator are contained in the appendices.

Now that everything for the project was designed, it was time to put everything together and get it to work. The first step in this process was to select which device to use for the project and then, how many of these devices it would take to implement the project. At first the total number of input and output pins were considered to be the basis in deciding which device to use. In the beginning of the project the design was much simpler and it was totaled to be about 91 inputs and 2 outputs. After the actual design, the size of the circuit came out to actually be _ inputs and _ outputs. Using this information, we needed an Altera chip with a lot of input/output pins and looking through the available devices we felt that we would need an Altera chip with 160 I/O pins. After thinking about it, we decided that the best option would be to split up the project into several smaller chips. Based on the facts that we had used them before and that the programming adapter was available for them, we went with 84 pin Altera chips.

The first device that was attempted was the EPM7128ALC84 chip that was used in EE 308, a familiar chip. When trying to get the project to fit onto the device it kept on failing and refusing to work no matter how many variations were tried. After with consulting with one of the Professors in the EE department, Dr. Stephen Bruder, it was evident that the device chosen was not the best fit for the project. Using Dr. Bruder's advice to choose a device from the Altera flex family, the new device chosen was the EPF8282ALC84-4 chip from the Altera Flex 8000 family. Using this chip though would require difficult programming steps using JTAG plus the Flex chip is a destructible SRAM device. In the end, the chip chosen was the EPM7128SLC84 chip. Simulating the project in Altera and setting the device setting to the new chip and making sure that it would compile tested the fit of the new device. This was a very convenient way to test the fit of the project without actually having the chip. Now it was easier to order the correct device, knowing what device fit correctly under the specification of 84 pins.

After deciding on the device that we were going to use, the project could now be split up onto several of the 84 pin Altera devices. Because of the amount of input pins required and the select control lines and the outputs it was decided to implement each multiplexor on separate chips. The pins on each chip were forced to set values in order to be able to change the project without any change in chip layout. The comparator and the random pattern generator were implemented onto a third chip.

GPIB Interface

The GPIB interface provides a standardized interface between to the clock recovery board test set and the computer. The GPIB interface was chosen due to design requirement that any interface used for the project should be able use a computer system assembled from commonly available parts. The customer was particularly concerned that the interface card should be commonly available and be able to be controlled by commonly available software. A GPIB interface lends itself well to filling these requirements. The GPIB interface card can be installed in any modern computer because it uses a PCI slot and is compatible with any common operating system. Another point in favor of the GPIB interface is that Labview has tools that allow the interface to be easily programmed for the project's needs. The use of the use of GPIB allows for future expansion of the test set's capabilities as part of a computerized workbench. Other options considered instead of if GPIB were a DAQ card and a RS232 interface. The DAQ card was not chosen because it was not standardized the way that the GPIB is. The RS232 interface was not chosen because although it a very standard interface, it is very slow as compared to the possible 1Mb/s transfer rate of the GPIB interface.

The design of the GPIB interface is divided into areas of hardware and software. The hardware portion of the interface consists of a Motorola HC12 and a National Instruments NAT7210 GBIP controller and its associated interface circuitry. The HC12 is the central controller of the test set. It selects the two channels to be compared using the MUX, gathers error count data and controls

the GPIB interface. The HC12 is interfaced to the rest of the test set using memory mapped I/O. The MUX, the error counter, and the GPIB controller are read from and written to as if they were memory addresses. In this way the NAT7210 is configured and controlled using its internal registers.

The hardware setup for the GPIB interface consists of an interface between the HC12 and the NAT7210. The GPIB controller's internal registers are selected using three register select lines that are feed directly into the chip. The register select line values from 0-7, read, write, and chip select control the access to the internal registers. The internal registers configure the GPIB controller and control what the chip does. The HC12 has multiplexed data and memory lines on the higher order address lines on port A. This makes it necessary to latch the address data on these lines before the data is output to memory. The latch is set using the rising edge of the E-clock. The E-clock is a special clock signal that dictates the timing of the various portions of a memory access. The output of the latched address lines is then put through a decoder to generate an active low chip select signal. Any latched address from 1000h to 1700h will generate a chip select signal. Address lines A10-A8 are fed directly into the NAT7210 register select lines. The R/W line of the HC12 must be fed into some logic in order to produce separate active low read and write signals for the GPIB controller. The active low reset signal for the HC12 is inverted so that its can reset the GPIB controller. In order to simplify the wiring of the circuit and speed up development, the logic was implemented using ALTERA.

the test set. The GPIB bus transceivers were wired to the bus controller as shown in the documentation for the controller chip. The interface between the controller chip and the HC12 was tested. It was confirmed that the HC12 could change the values of the internal registers of the controller chip and the values of these registers could be read back. The wiring for the bus transceivers was checked thoroughly but proper operation of the transceivers could not be checked until the controller chip was programmed. An initialization and test program for the GPIB interface was written for the HC12 using the instructions contained in chapter 5 of the documentation for the NAT7210 GPIB controller. Unfortunately the GPIB interface did not operate correctly. The computer was not able to detect the NAT7210 as listener on the bus. Possible causes for this problem could include faulty bus transceiver circuitry, improper configuration of the computer, or a bug in the HC12 firmware. Due to time constraints we were unable to troubleshoot the interface. Instead we switched to a serial port interface between the computer and the test set.

In order to save time the serial port interface was used. The HC12 has a serial port built in which made it unnecessary build any new hardware. Assistance with use of the serial port and sample code were easily obtained from Dr. Rison. On the other hand the assistance with the GPIB interface was very hard to come by. There was nobody on campus or at the NRAO who had any experience programming the NAT7210. Technical support at National Instruments refused to provide source code and would only answer specific programming questions. The serial port was able to provide all the same

functionality as the GPIB with only the loss of speed and the ability to include the test set with other GPIB controlled instruments as part of a computer controlled workbench. Labview also supports serial interfacing with easy to use tools similar to the one used by the GPIB. This combination of circumstances made the serial port interface the best overall choice. To conclude, after having dealt with the GPIB on this project, a DAQ card or serial communication would have made a simpler and quicker solution.

RS 232 Interface

As already listed in the previous section about GPIB, it was decided near the end of the project that using the onboard serial port of the HC12 would be a more viable option than designing an entire GPIB controller for PC control of the test interface. One question that could possibly arise is why was the switch so late and why wasn't RS 232 an option in the beginning? The Usage of GPIB as the PC interface was decided before the microcontroller to control it. Most microcontrollers do not have the added features built in as the HC12 evaluation board that was chosen for the project. In this section the usage of the RS 232 and code from the firmware will be discussed further.

On the Motorola HC12 there are two features meant for use in serial communications. For use in the project, the SCI (serial communications interface), which uses the onboard serial port and is compatible with a standard PC serial port was used. To use the HC12's SCI, it first needs to be set up. To do this a zero needs to be written to the SC0BDH register (all register addresses are contained in the header file hc12.h). Next, the baud rate for the port is to be set up. This is done writing to the SBR12 –0 bits of the SC0BDH and SC0BDL registers. First, set SB12-0 equal to 8Mhz (HC12's clock speed) divided by 16 * desired baud rate. For this project a baud rate of 9600 was chosen. Using the previous equation it is determined that the number to be written to the registers needs to be (8Mhz/(16*9600)) = 52.083 ~ 52. Now that the baud rate is set up both the transmitter and receiver need to be enabled by writing to the SC0CR2 register. This register also needs to be written to determine interrupts. For this

project, no interrupts were chosen. Lastly a hex 00 is written to the SC0CR1 register to enable normal mode with 8bit communications and no parity, standard serial port communications standards. The following set up code is as follows:

```
SC0BDH = 0;                                   /* Setup Serial Subsystem */
SC0BDL = 52;                                  /* 9600 baud */
SC0CR2 = 0x0C;  /* Enable transmitter and receiver, no interrupts */
SC0CR1 = 0x00;                                /* Normal mode, 8bits, no parity */
```

After the SCI is setup there are two sets of code to communicate through the RS 232 serial port. The first is the set of code that enables the HC12 to transmit data over the port. The first step in transmitting data is to write the data to be sent to the SC0DRL register. In this project the data sent is being sent as a character. After writing the data to SC0DRL a while loop needs to be written to wait for the transmit data register empty (TDRE, 8^{th} bit of SC0SR1) to be written. If the transfer is interrupted, there is an error and ones are present to the lower four bits of the SC0SR1 register. In the program for this project there is an if loop to check the SC0SR1 register to identify an error.

The next piece of code that is used to communicate through the HC12's serial port is the code to receive data from the serial port. As with the transmitting portion of this project, the receive code is written to receive data in character form. To receive the data, a while loop is written to wait for the data register full (RDRF, bit 6 of SC0CR1) to be cleared. Next the data can be read from the SC0DRL register. The following piece of code features a character written to the serial port and then two lines to receive a character from the serial port. The last lines of code are the error identifier.

```
                    Char character = c;
                     /* Transmit Data */
SC0DRL = character;  /* Send the letter that is stored in character */
 while ((SC0SR1 & 0x80) == 0) ;                    /* Wait for TDRE */
                     /* Receive Data */
while ((SC0SR1 & 0x20) == 0) ;       /* Wait for received character */

 character = SC0DRL;                 /* Read Character from SC0DRL */

                    if ((SC0SR1 & 0x0f) != 0){
 DBug12FNP->printf("Error Detected\n");        /* Error Identifier */}
```

The firmware, written in C, for the HC12 using the SCI was written using pieces from the above code. The first part was the setup code. After this the pieces of code used for transmitting data and for receiving data are used where they are needed to receive commands from the PC or to send results to the PC. The next section of this paper features a discussion of the PC side of the software, which was written in LabView.

LabView GUI Software

After choosing a type of interface to communicate with the test fixture the control software for the PC was to be written. This control software was written using National Instrument's LabView graphical software. LabView was chosen because it has built in functions to control different kinds of interfaces plus it is something that the group, giving the group familiarity with the way that it works.

Early in the project LabView was being used to control the project via the GPIB interface. A test program was written to first initialize the GPIB bus, setting the PC as the controller with the test set as either talker or listener based on what was needed at the time. This first GPIB program was written using commands from the GPIB tools out of the communications toolbox of LabView. GPIB standard 488 was chosen as opposed to 488.2 because it was more straightforward and there was no need for the added functionality of the 488.2 standard. The tools used in the test program were GPIB initialize, GPIB read and GPIB write. These three commands were to be used together with other LabView tools to build up the rest of the function until the switch to the RS 232 interface.

Fortunately the switch to RS 232 serial communications was no too much of a switch in the LabView portion of the programming. The basic tools to accomplish RS 232 are in the same communications toolbox in LabView as part of the RS 232 tools. The tools that were used included write to serial port, read from serial port and bytes at serial port. The last mentioned tool was used to know how many bytes to read from the serial port, and writing that number to the read from serial port. Using this tool, the amount of bytes read from the serial

port does not have to be limited or wasted by guessing a number to represent the number of bytes.

The first program written was for the sole purpose of establishing communications with the serial communications interface of the HC12. This first program was an 'echo' program. First it writes a character to the serial port. The HC12 software receives this character and stores it into a character and lastly sends the character back to the serial port. LabView now reads the character at the serial port as it is 'echoed' back from the HC12. This program was then changed to provide different outputs based on the character sent to it.

After the test program was written it was time to write the main program in LabView. The first interface that was originally written for use with GPIB was still good. In the graphical portion of the project there are two boxes with arrows to select channels A and B to be compared to each other. Using LabView the numbers were limited to be between 2 and 34 (the usable channels from the recorder). These numbers are then written to the serial port and interpreted by the HC12. The HC12 now uses the memory mapped IO to write the address of the Multiplexors. The data is now sent to the comparator circuit of the project. Once the comparator circuit completes receiving its error rate it lets the HC12 know by raising a bit. Next the HC12 takes the error rate stored in memory and sends it through the serial port to LabView where it is displayed in another box.

The instructions for using the LabView Software are listed on the User Instruction Manual. All of the information regarding the usage for the LabView functions such as the RS 232 Write and RS 232 Read functions are included in

the appendix section. Additional information about the LabView GUI is included in the User Instructions portion of this paper, which follows the conclusion.

Conclusion

Overall the completion rate of the project was satisfactory. What was finished turned out to be:

- VME Board Wrapping complete
- HC12 Modifications Complete
- Altera Logic Written
- Altera JTAG Programmer Complete
- All Individual Logic Tested and Functional
- HC12 SCI Initialization Complete
- LFSR Logic Design Complete

The parts left to be completed for the project include:

- LabView Software Started
- Altera Chips to be programmed
- All Devices to be integrated and tested

There were parts of the project that perhaps could have been better completed using different approaches. One of these of course was the choice for the computer interface. One of the possibilities in the beginning was the usage of a Data Acquisition Card controlled by LabView. The customer required the usage of a GPIB controller in order to have a standard communications device. The GPIB controller chip was chosen before the HC12 microcontroller so the obvious option of using it's included serial communications interface did not present itself until later.

Another aspect of the project which could have been better conceived was the Altera implementation. The selection of devices for the project is where problems arose. The first chip that was to be used could not handle the type of logic that was required. Going by the advice of a professor the chip was changed to the Flex family. This chip was selected without much knowledge of how it worked. The Flex device ended up being useless because it proved very difficult to program and plus it was built up of destructible SRAM. In the end we found success with the use of a chip from the MAX 7000S family, almost the same as the chip that we originally chose, from the MAX 7000A family.

On a positive note, large portions of the project were completed and tested. Portions of the project required many hours of work. These included the wire wrapping of the VME board, circuit design and construction, software writing and design and Altera design. There were many things to learn from this project from integrating different aspects of electrical engineering to learning about designing the device to work at 4.5Mhz and learning how data is managed for scientific use.

Astronomy Observatory and includes references to necessary equipment located at the NRAO's Array Operations Center in Socorro, New Mexico. The senior design group is not responsible for changes made to the program or available equipment not prepared by them.

Section I: LabView Controller version 1.0

The PC controller for the NRAO Test Fixture is controlled using National Instruments LabView software as an interactive graphical user interface using the Serial Communications Interface RS 232 Bus of the HC12 as data acquisition lines and as a control bus.

Starting the Program:

To start the program, first open the folder with the program file, NRAOTF, in it. Next double click the file, LabView will automatically start up with the program. The front panel of the program will appear and is now ready to be used.

Usage:

Now that the program is ready, the two channels to be tested against each other can now be selected using the two boxes marked Channel A and Channel B using the arrows to the right of the number. The box is limited to numbers between 2 and 33, which are the channels associated with the recorder. Once the channels are selected the program will execute testing once the arrow button at the upper left of the LabView window is pressed. When the testing is finished,

the output sample of the error is listed to the left of the channel selectors. A graphical histogram of the error is an option that may be graphed just below the error. The selection of the channels to be tested against each other can be changed at any time and then set to run again using the arrow button. A graphic of the software panel is below as figure 2.

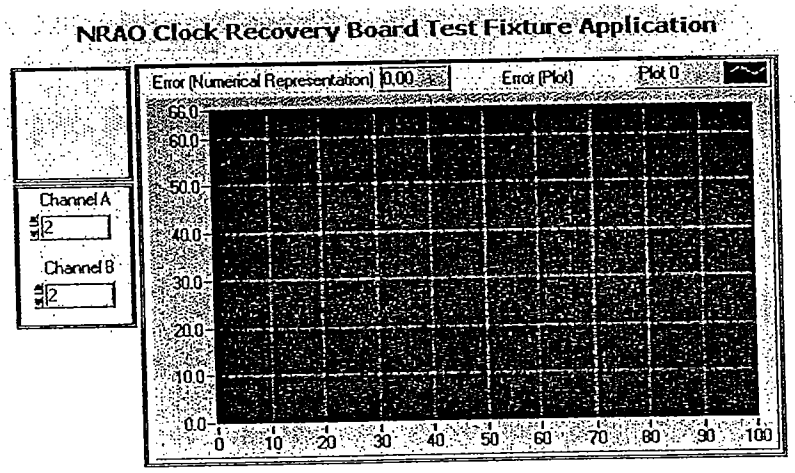


Figure 2: software front panel

Section II: Altera Programmable Logic

The NRAO test fixture uses three 84 pin Altera programmable logic chips to run the multiplexors and comparator circuit that select the channels to be tested and then sample the data to compute the error. These three devices come already affixed to the VME board that is inserted into the testing rack. The purpose of this portion of the manual is for the unlikely purpose that one of the chips either needs to be replaced or re-programmed. This part of the manual goes through the process of replacing the chip as well as how to re-program a device.

Removing and Inserting the device(s):

All three of the Altera Programmable logic devices are 84-pin chips inserted into 84-pin sockets. They are in a row and numbered starting from the closest chip to the digital logic chips on the board as U1, U2 and U3. Each socket has an arrow pointing to where the top of the chip needs to be for proper operation. There is a small dot at the top-center of the chip as well as a small indentation in the top-left corner of the chip to aid in lining up the chip with the socket. The chip should be removed using a chip puller in order to minimize the chance of physical damage to the chip and the socket. The chip should be inserted by first aligning the chip with the arrow at the top of the socket and then gently pushing the chip into place by pushing it gently at the center.

Programming a Device:

Included with the CDROM disc for this project is the programming files required to re-program each one of the three devices that make up the project. The three program files are located on the Altera directory on the CDROM disc. The file for the chip U1 is titled u1_chanmux.pof and is the programmer file for the channel multiplexor. The file for chip U2 is u2_clkmux.pof and is the file for the clock multiplexor. The programmer file for U3 is u3_comparator.pof and is the file associated with the comparator circuit and random signal generator portion of the project. To program a device, plug it into the provided JTAG programming device. In the Altera MAX+plus II software go to file and then open to select the correct .pof file. Next select file again and go to name and select set name to

current file. Lastly, go to the Max_plus II menu option and select programmer. Make sure that the JTAG device is plugged in and connected via the parallel port of the machine. Now the program button can be pressed and MAX+plus II software can be closed. It is important that each Altera program not be mistaken for another. They are nearly identical but have differing pin layouts and if installed in the incorrect socket will cause the test fixture to malfunction.

Maintenance Manual

Visual Inspection:

Place the VME board on a piece of foam material to protect the pins on the underside.

Look for bent pins and loose or missing hardware.

Ensure all components are properly inserted in the VME card

Preparing for Operation:

Take care when inserting male connectors to female counterparts on the VME card; improper care can damage the connectors or the VME card.

Ensure that the VME card is correctly seated in the VME bus.

When using extender card, take care when inserting extender card into VME bus.

During Operation:

Do not move any external probes around the board during operation due to potential shorting.

After Operation:

Take care when removing connectors from the VME card; aggressive removal of connectors may damage them or their point of installation on the VME card.

Take care when removing extender card and VME card from the VME bus.

Storage and Handling:

Avoid any direct pressure on the underside of the VME card; extreme pressure can cause damage to the pins increasing the possibility for shorting the circuit or other permanent damage to the card.

When setting down the VME card, ensure it is sitting on a piece of foam to prevent damage to the pins.

DO NOT place anything on top of the VME card. This can cause extreme damage to the card.

When storing the VME card, place a piece of foam to the underside of the card to prevent damage to the pins during storage.

Routine Maintenance:

When replacing logic devices from the VME card, use proper chip pulling tools; the lack of use of proper tools may damage the logic devices and the card.

When removing Altera chips for programming, use proper PLD removal tools; using proper tools will minimize any damage to the Altera chips and Altera sockets.

In the case of a missing logic device, always consult the schematics as to the proper installation location on the VME card. **DO NOT** install any device on the VME card without consulting the schematics first.

Trouble-shooting Guide

If the circuit seems to be working incorrectly (if data rates are consistently high or error rates are very inconsistent):

Probe the card when the power is turned on to ensure there is a voltage of 5V at the VCC pins and GND at the gnd pins.

If there are incorrect voltage readings on the VME card or no voltage readings on the VME card:

Ensure the VME card and VME extender cards are properly seated in the VME bus to ensure power is being supplied to the circuitry.

If the voltages are correct on the board but the circuitry is not working properly:

Hook an oscilloscope to the "LSFR" and "Trigger" BNC connectors to display the output of the LSFR "random signal generator." The output should look similar to:

Picture

If the output from the LSFR appears to be correct, all the Altera chips should be functioning correctly. This is concluded because the LSFR is housed in U3 which takes inputs from U2 and U1. Therefore, if U2 and U1 were not working, U3 would not be giving a correct output. Static shock to an Altera chips can erase its memory. When the Altera chip is shocked, usually it completely stops functioning and not just specific functions. The differential line-receivers should also be functioning correctly; this can be concluded since a "clean" clock signal is required for the LSFR to function properly.

If the output of the LFSR appears to be incorrect:

Using an oscilloscope, monitor the output of any one of the differential line receivers. The output from the clock pin should resemble the following:

Picture

The output from the data pin should resemble the following:

Picture

If the output from the differential line-receiver appears to be incorrect, refer to the schematics and ensure that it is properly wired.

If the line-receiver is correctly wired:

Look at the inputs to the line receiver and ensure that they resemble the following:

Clock Input

Picture

Data Input

Picture

If the input appears to be correct:

Replace the differential line-receiver and reinstall a new one while referring to the schematics for correct installation location.

Once this has been corrected, proceed to check the remainder of the differential line receivers and correct any errors found in the wiring and/or replace any damaged line-receivers.

If the output to the line-receivers is correct:

It may be possible that the Altera chips are not functioning correctly. To check this, Altera chips U1 and U2 must be checked first. The reason for this is they are both inputs to U3. So if U1 and U2 are not functioning, U3 will not display a correct output. Monitor the clock and data output from Altera chips U1 and U2. This can be one by monitoring the following pins: Data pins: U1, pins 78 and 81; Clock pins: U2, pins 78 and 81. The output to these pins should resemble the following:

Data pins

Picture

Clock pins

Picture

If the output appears incorrect, reprogram the appropriate Altera chip(s) (U1 or U2) using the instructions as detailed in the instruction manual.

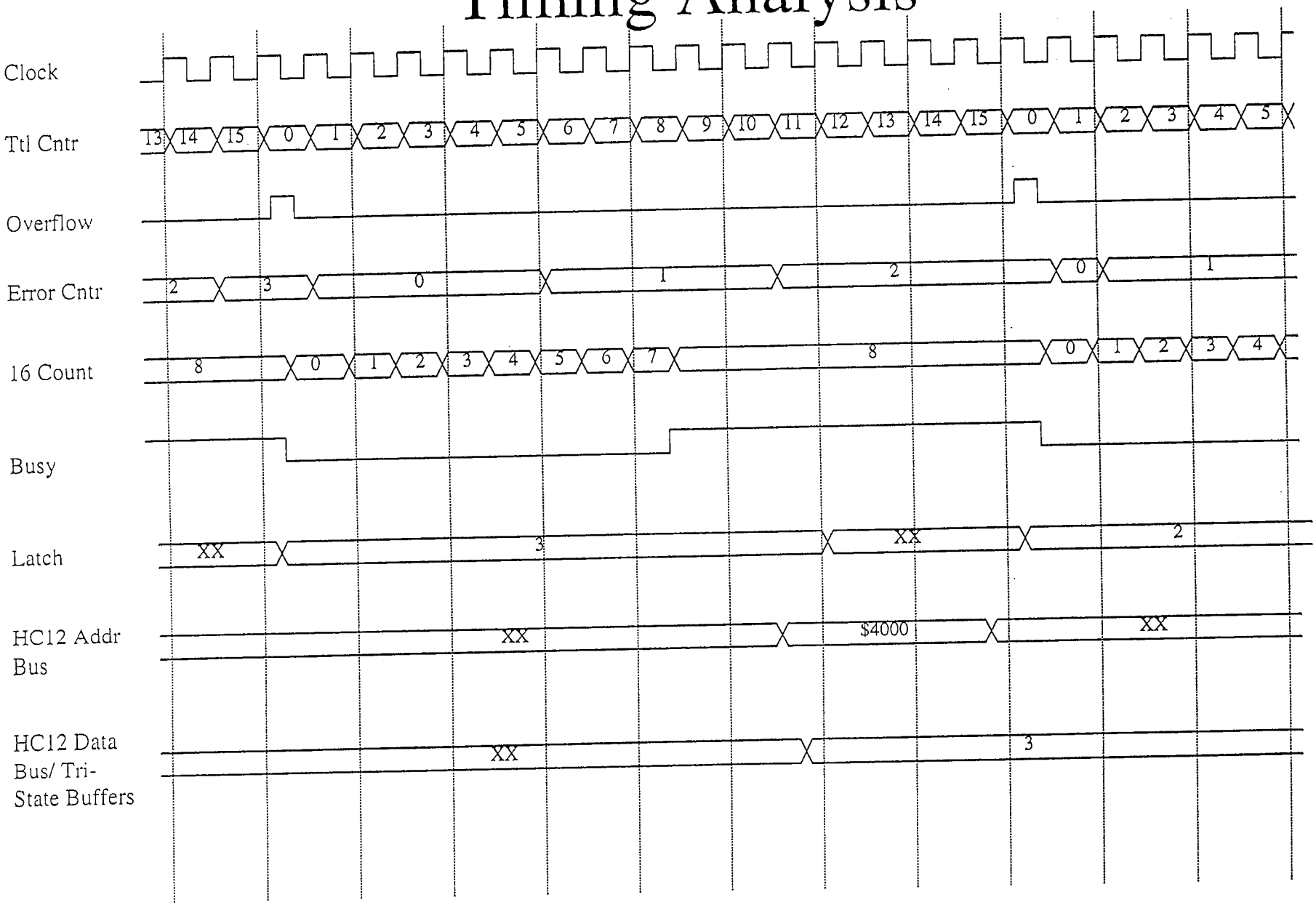
If the output of U2 and U2 are correct:

It can be concluded that U3 is not functioning correctly. Reprograms U3 using the using the instructions as detailed in the instruction manual.

If there are still problems with the circuit:

Check that all connections (including all connections to the computer) are working properly, all discrete logic chips have Vcc and ground, or try to replace and reprogram the Altera chips.

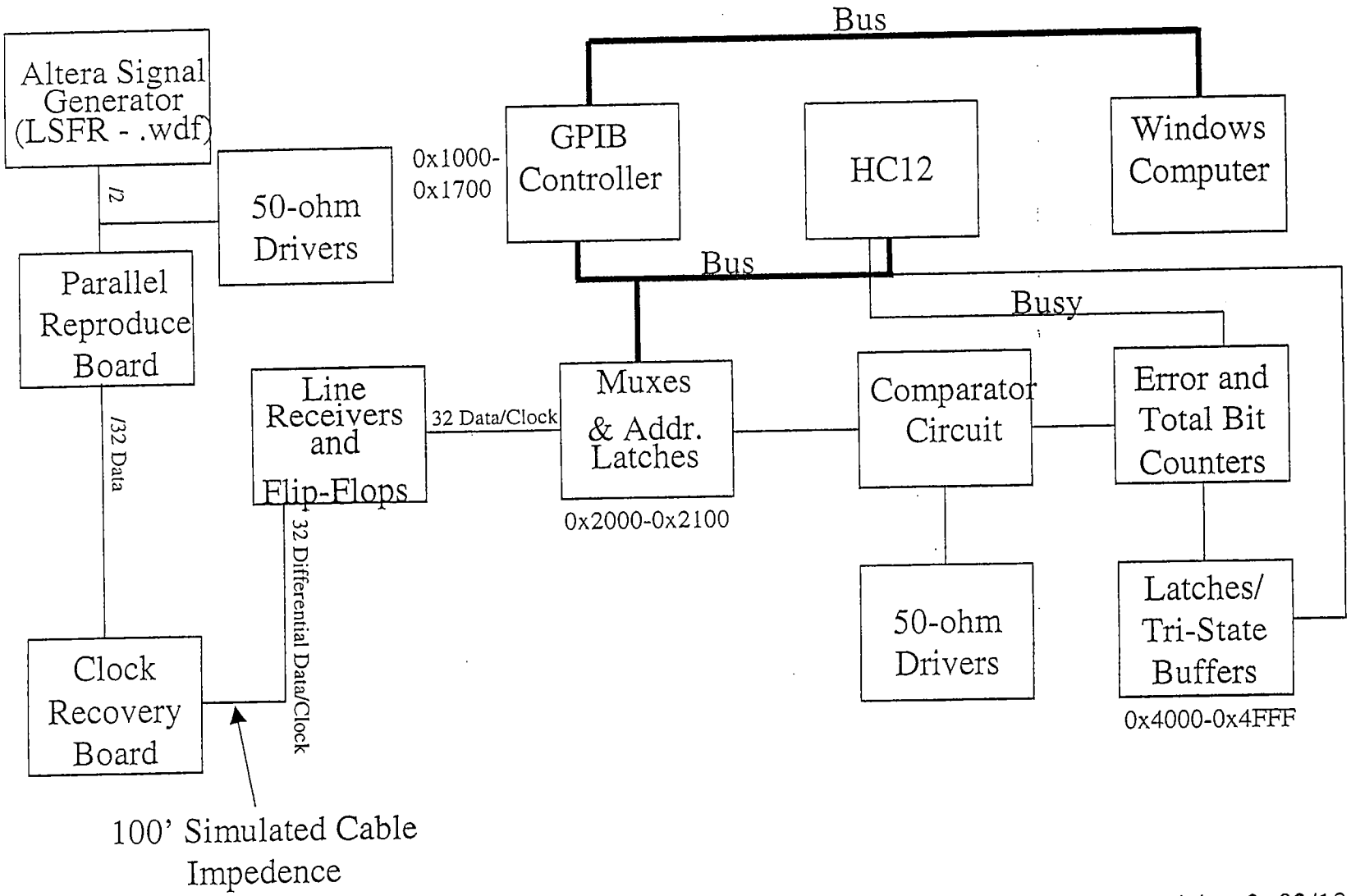
Timing Analysis



*A 16 bit total error counter and 8 bit "busy" counter were used here to allow the reader to see two overflow cycles which would be impossible to show using the real counters

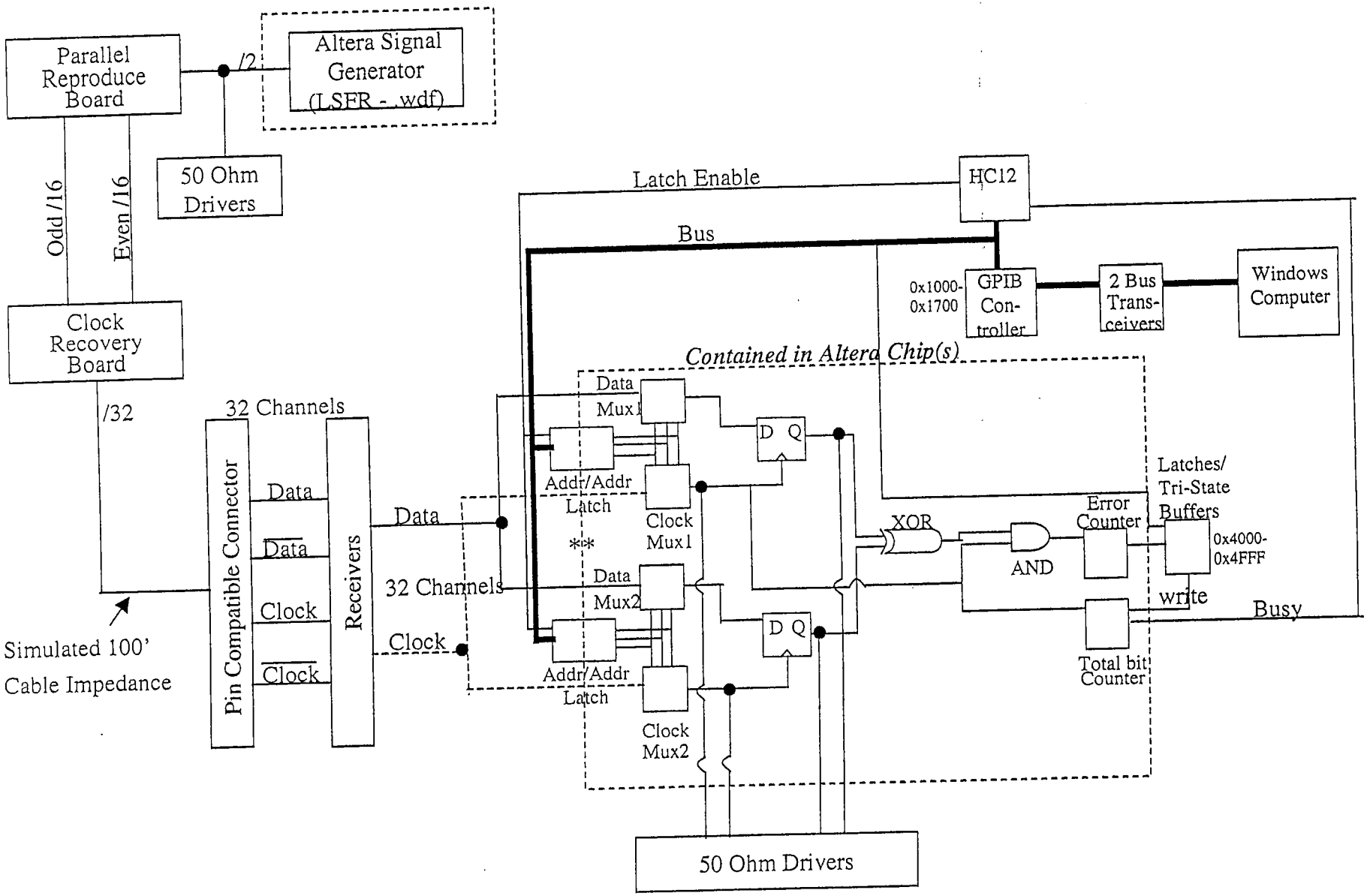
Revision 0: 03/18/01

System Integration



Revision 3: 03/18/01

System Integration

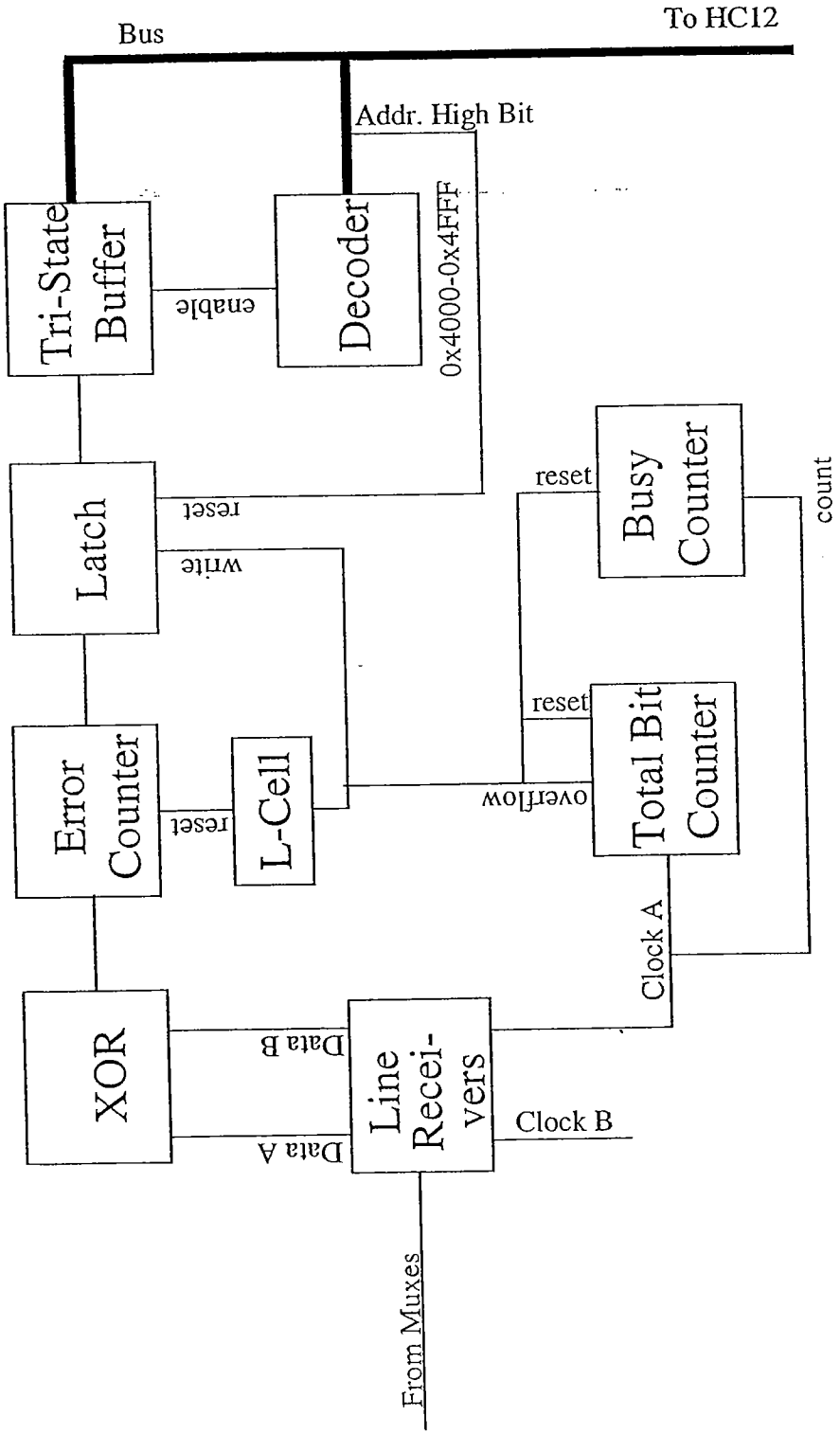


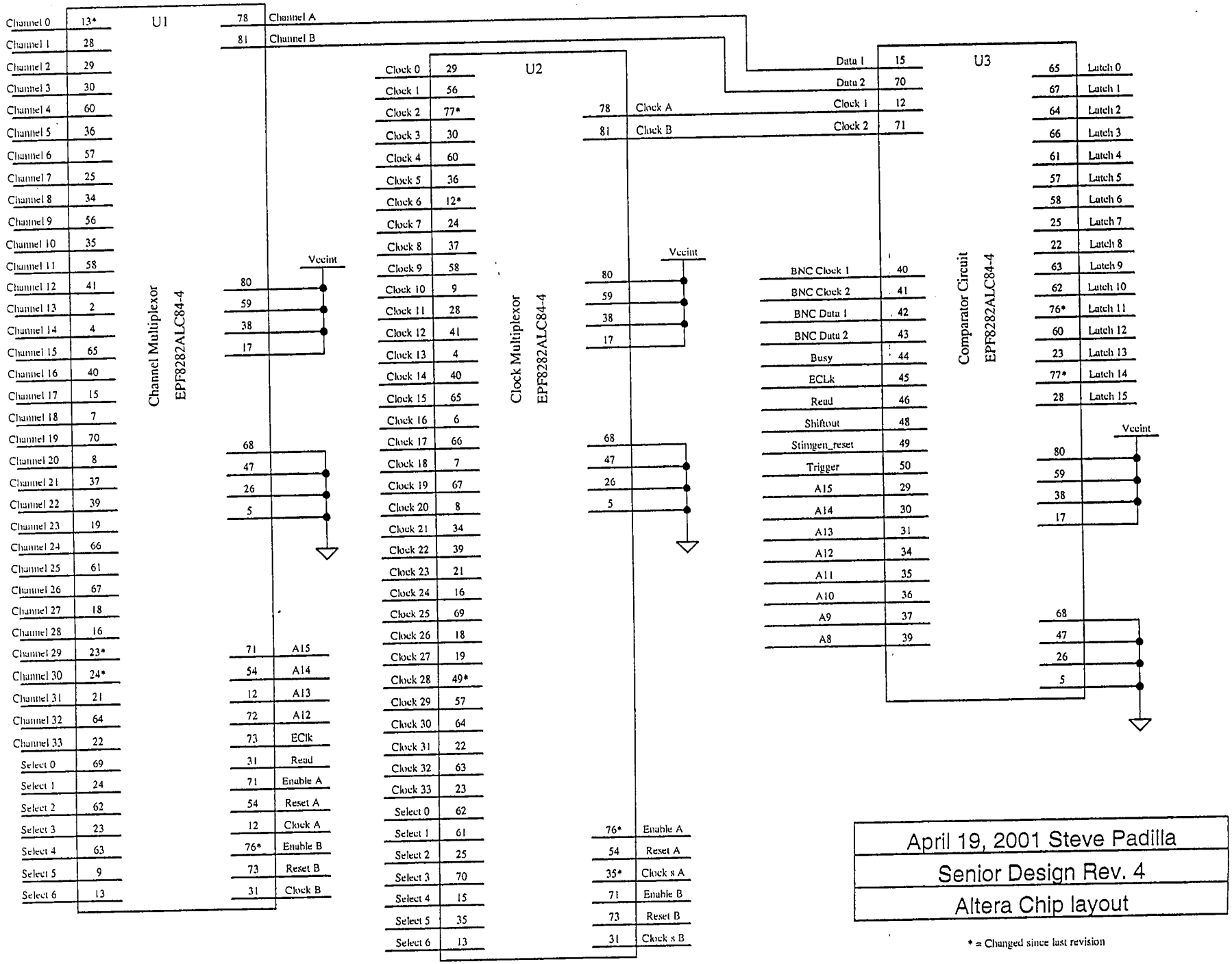
** The muxes are also equipped with decoder circuitry (0x2000-0x2100)
*The circuit is to be assembled and wire wrapped to a VME bus card

Revision 6: 03/18/01

B.

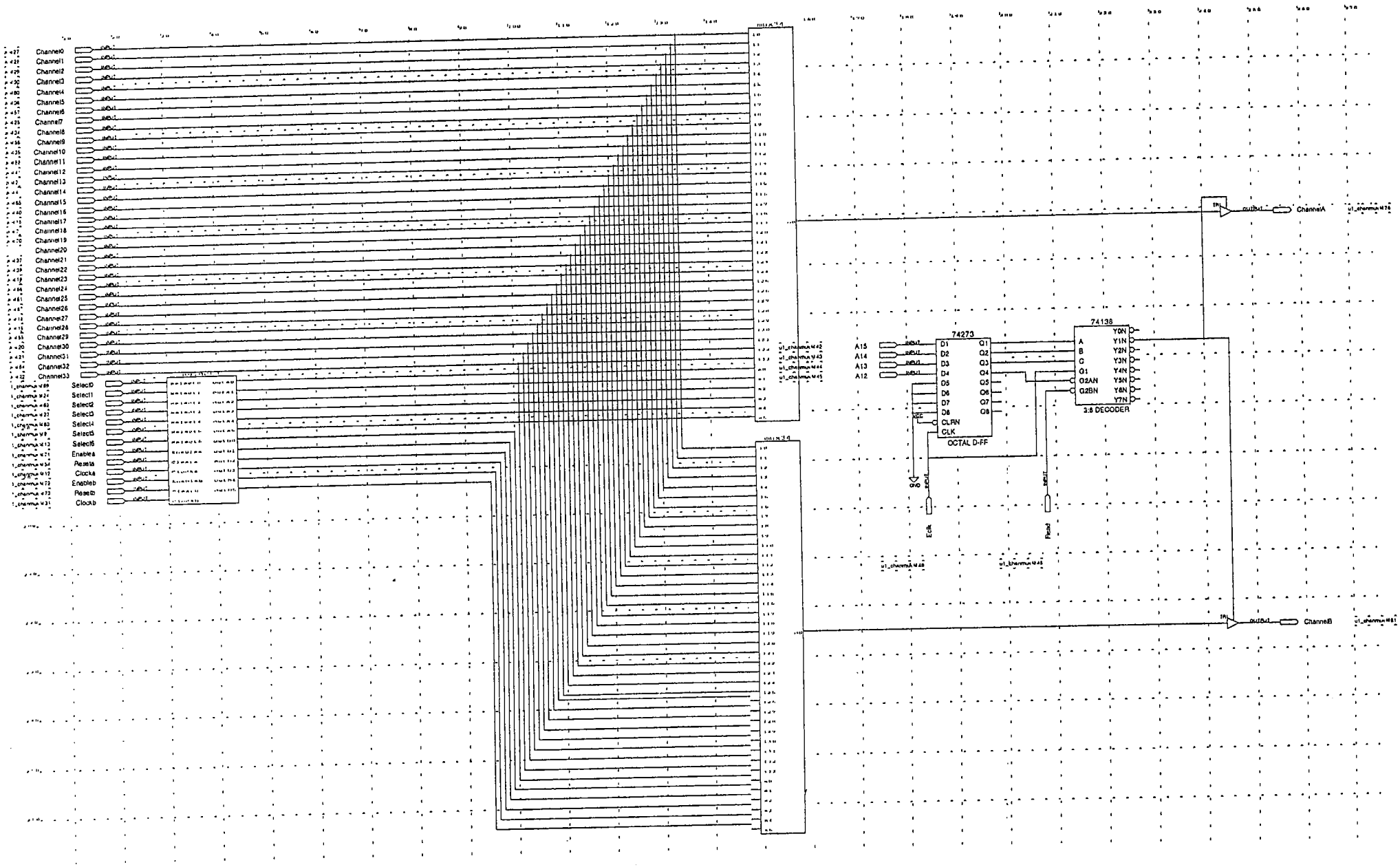
Comparator Circuit Block Diagram



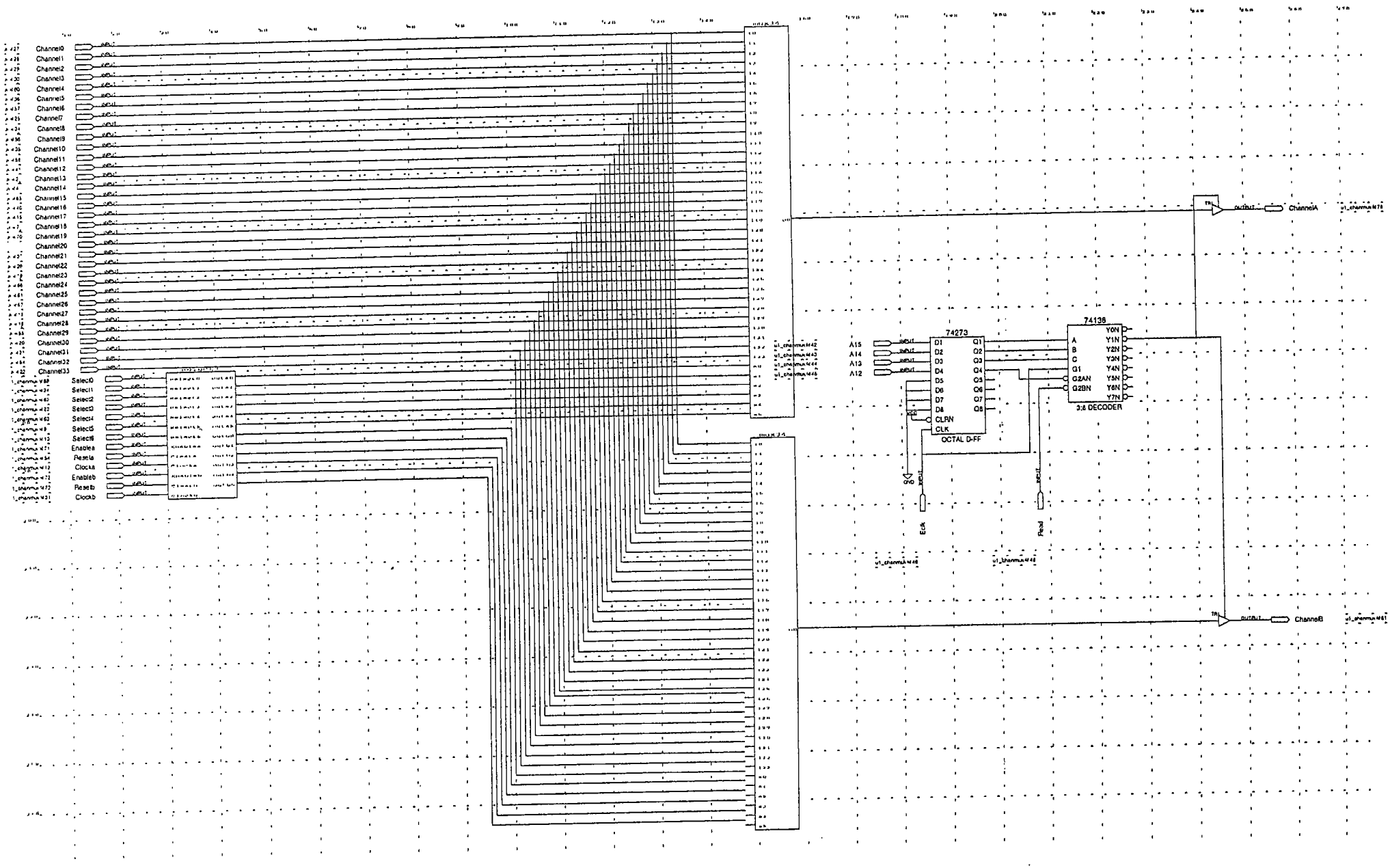

U1
Channel Multiplexor
EPF8282ALC84-4
Channel 0 13*
Channel 1 28
Channel 2 29
Channel 3 30
Channel 4 60
Channel 5 36
Channel 6 57
Channel 7 25
Channel 8 34
Channel 9 56
Channel 10 35
Channel 11 58
Channel 12 41
Channel 13 2
Channel 14 4
Channel 15 65
Channel 16 40
Channel 17 15
Channel 18 7
Channel 19 70
Channel 20 8
Channel 21 37
Channel 22 39
Channel 23 19
Channel 24 66
Channel 25 61
Channel 26 67
Channel 27 18
Channel 28 16
Channel 29 23*
Channel 30 24*
Channel 31 21
Channel 32 64
Channel 33 22
Select 0 69
Select 1 24
Select 2 62
Select 3 23
Select 4 63
Select 5 9
Select 6 13
78 Channel A
81 Channel B
Vccint
80
59
38
17
68
47
26
5
71 A15
54 A14
12 A13
72 A12
73 EClk
31 Read
71 Enable A
54 Reset A
12 Clock A
76* Enable B
73 Reset B
31 Clock B
U2
Clock Multiplexor
EPF8282ALC84-4
Clock 0 29
Clock 1 56
Clock 2 77*
Clock 3 30
Clock 4 60
Clock 5 36
Clock 6 12*
Clock 7 24
Clock 8 37
Clock 9 58
Clock 10 9
Clock 11 28
Clock 12 41
Clock 13 4
Clock 14 40
Clock 15 65
Clock 16 6
Clock 17 66
Clock 18 7
Clock 19 67
Clock 20 8
Clock 21 34
Clock 22 39
Clock 23 21
Clock 24 16
Clock 25 69
Clock 26 18
Clock 27 19
Clock 28 49*
Clock 29 57
Clock 30 64
Clock 31 22
Clock 32 63
Clock 33 23
Select 0 62
Select 1 61
Select 2 25
Select 3 70
Select 4 15
Select 5 35
Select 6 13
78 Clock A
81 Clock B
Vccint
80
59
38
17
68
47
26
5
76* Enable A
54 Reset A
35* Clock s A
71 Enable B
73 Reset B
31 Clock s B
U3
Comparator Circuit
EPF8282ALC84-4
Data 1 15
Data 2 70
Clock 1 12
Clock 2 71
BNC Clock 1 40
BNC Clock 2 41
BNC Data 1 42
BNC Data 2 43
Busy 44
ECLk 45
Read 46
Shiftout 48
Stimgen_reset 49
Trigger 50
A15 29
A14 30
A13 31
A12 34
A11 35
A10 36
A9 37
A8 39
65 Latch 0
67 Latch 1
64 Latch 2
66 Latch 3
61 Latch 4
57 Latch 5
58 Latch 6
25 Latch 7
22 Latch 8
63 Latch 9
62 Latch 10
76* Latch 11
60 Latch 12
23 Latch 13
77* Latch 14
28 Latch 15
Vccint
80
59
38
17
68
47
26
5
April 19, 2001 Steve Padilla
Senior Design Rev. 4
Altera Chip layout
* = Changed since last revision

```
TITLE "34 Channel Mux design for Senior Design 7 December 2000";

SUBDESIGN mux34
(
    i0                  : INPUT = GND;
    i1                  : INPUT = GND;
    i2                  : INPUT = GND;
    i3                  : INPUT = GND;
    i4                  : INPUT = GND;
    i5                  : INPUT = GND;
    i6                  : INPUT = GND;
    i7                  : INPUT = GND;
    i8                  : INPUT = GND;
    i9                  : INPUT = GND;
    i10                 : INPUT = GND;
    i11                 : INPUT = GND;
    i12                 : INPUT = GND;
    i13                 : INPUT = GND;
    i14                 : INPUT = GND;
    i15                 : INPUT = GND;
    i16                 : INPUT = GND;
    i17                 : INPUT = GND;
    i18                 : INPUT = GND;
    i19                 : INPUT = GND;
    i20                 : INPUT = GND;
    i21                 : INPUT = GND;
    i22                 : INPUT = GND;
    i23                 : INPUT = GND;
    i24                 : INPUT = GND;
    i25                 : INPUT = GND;
    i26                 : INPUT = GND;
    i27                 : INPUT = GND;
    i28                 : INPUT = GND;
    i29                 : INPUT = GND;
    i30                 : INPUT = GND;
    i31                 : INPUT = GND;
    i32                 : INPUT = GND;
    i33                 : INPUT = GND;
    s0                  : INPUT = GND;
    s1                  : INPUT = GND;
    s2                  : INPUT = GND;
    s3                  : INPUT = GND;
    s4                  : INPUT = GND;
    s5                  : INPUT = GND;
    o0                  : OUTPUT;
)

BEGIN
    IF s[5..0] == B"000000" THEN o0 = i0;
    ELSIF s[5..0] == B"000001" THEN o0 = i1;
    ELSIF s[5..0] == B"000010" THEN o0 = i2;
    ELSIF s[5..0] == B"000011" THEN o0 = i3;
    ELSIF s[5..0] == B"000100" THEN o0 = i4;
    ELSIF s[5..0] == B"000101" THEN o0 = i5;
    ELSIF s[5..0] == B"000110" THEN o0 = i6;
    ELSIF s[5..0] == B"000111" THEN o0 = i7;
    ELSIF s[5..0] == B"001000" THEN o0 = i8;
    ELSIF s[5..0] == B"001001" THEN o0 = i9;
    ELSIF s[5..0] == B"001010" THEN o0 = i10;
    ELSIF s[5..0] == B"001011" THEN o0 = i11;
    ELSIF s[5..0] == B"001100" THEN o0 = i12;
    ELSIF s[5..0] == B"001101" THEN o0 = i13;
    ELSIF s[5..0] == B"001110" THEN o0 = i14;
    ELSIF s[5..0] == B"001111" THEN o0 = i15;
    ELSIF s[5..0] == B"010000" THEN o0 = i16;
    ELSIF s[5..0] == B"010001" THEN o0 = i17;
    ELSIF s[5..0] == B"010010" THEN o0 = i18;
    ELSIF s[5..0] == B"010011" THEN o0 = i19;
    ELSIF s[5..0] == B"010100" THEN o0 = i20;
    ELSIF s[5..0] == B"010101" THEN o0 = i21;
    ELSIF s[5..0] == B"010110" THEN o0 = i22;
```

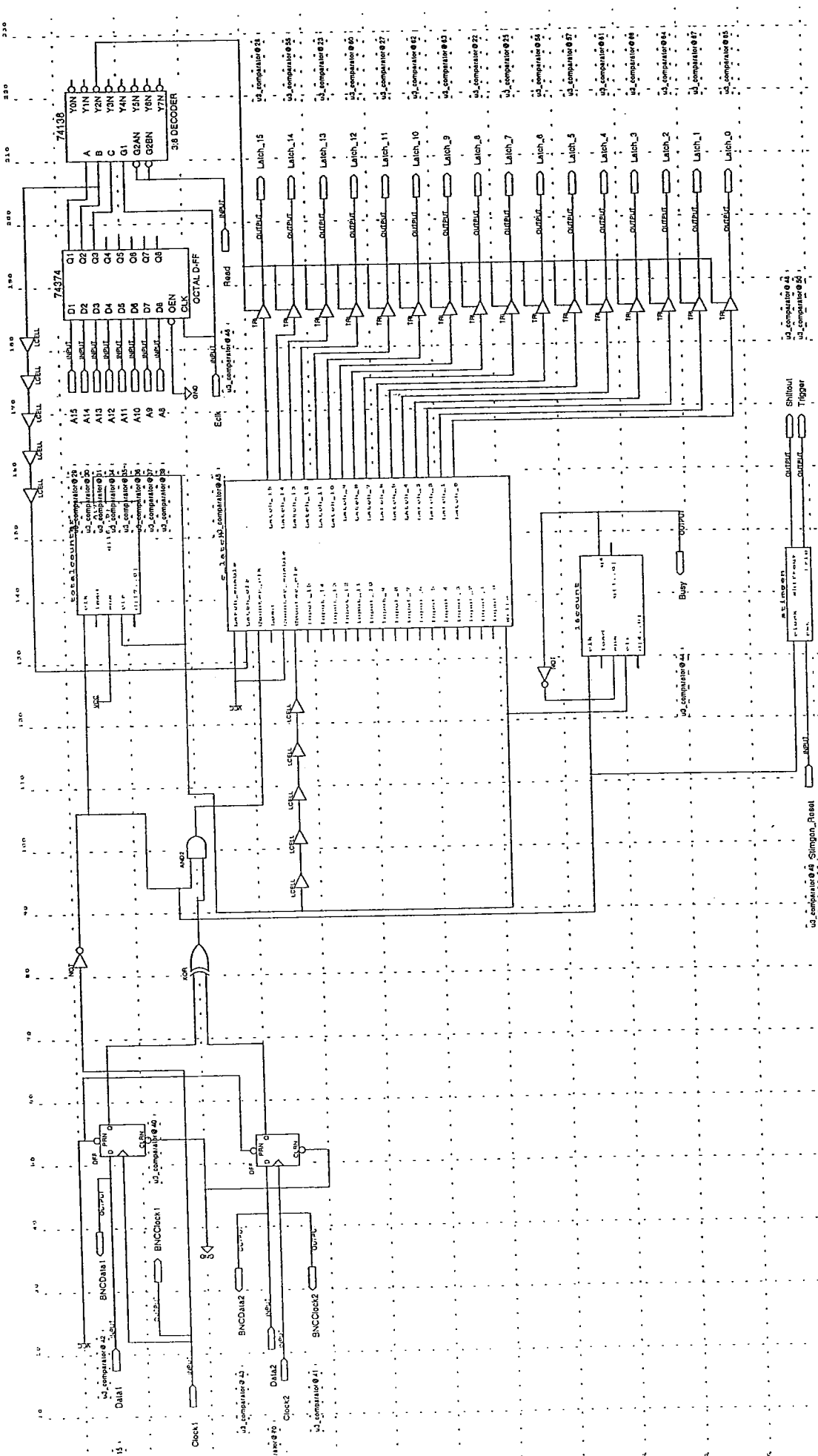
```
    ELSIF s[5..0] == B"010111" THEN o0 = i23;
    ELSIF s[5..0] == B"011000" THEN o0 = i24;
    ELSIF s[5..0] == B"011001" THEN o0 = i25;
    ELSIF s[5..0] == B"011010" THEN o0 = i26;
    ELSIF s[5..0] == B"011011" THEN o0 = i27;
    ELSIF s[5..0] == B"011100" THEN o0 = i28;
    ELSIF s[5..0] == B"011101" THEN o0 = i29;
    ELSIF s[5..0] == B"011110" THEN o0 = i30;
    ELSIF s[5..0] == B"011111" THEN o0 = i31;
    ELSIF s[5..0] == B"100000" THEN o0 = i32;
    ELSIF s[5..0] == B"100001" THEN o0 = i33;
 ND IF;
ND;
```



Data Multiplexor (contained in U1)



Clock Multiplexor (contained in U2)



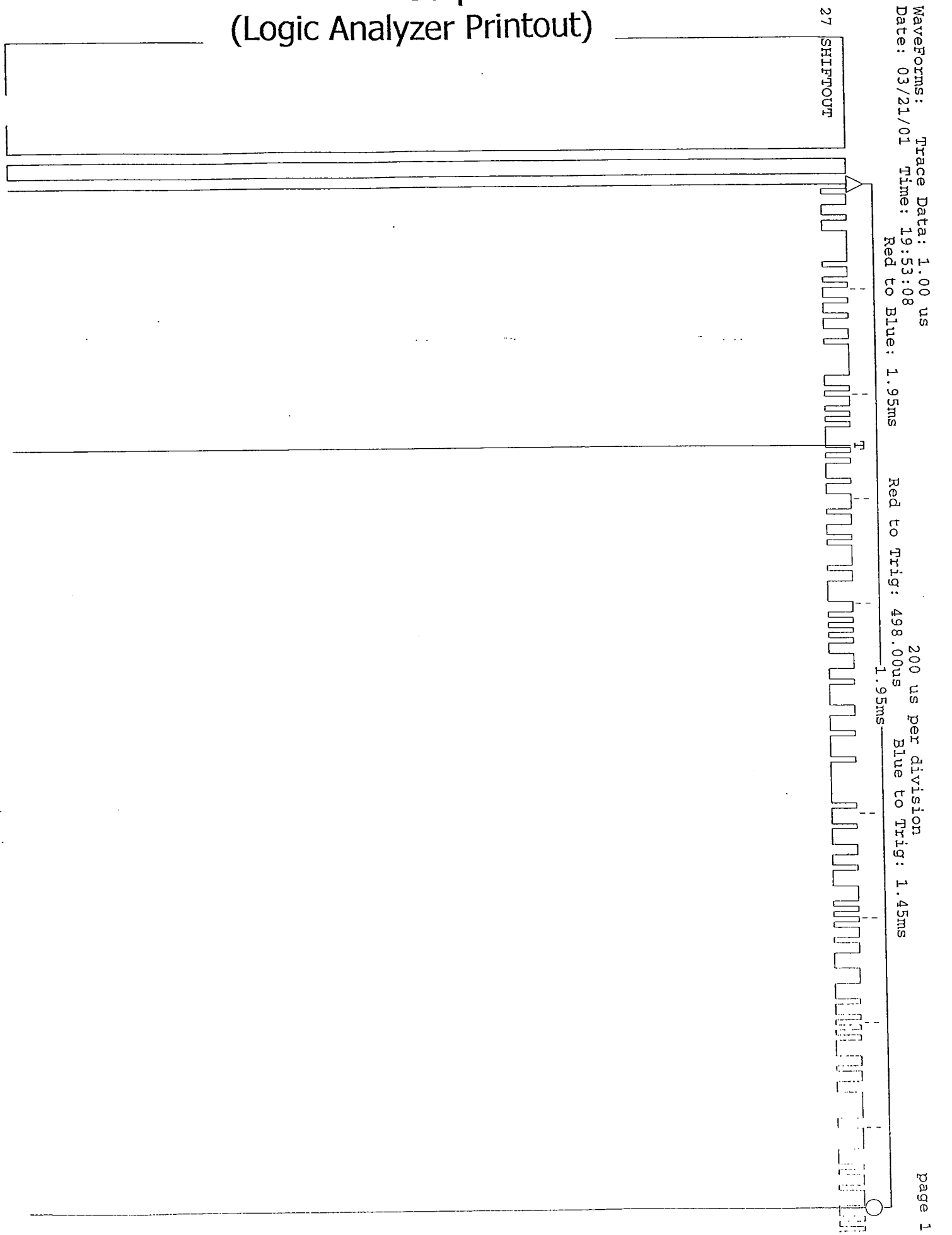
Comparator Circuit (contained in U3)

LFSR 'text' Design

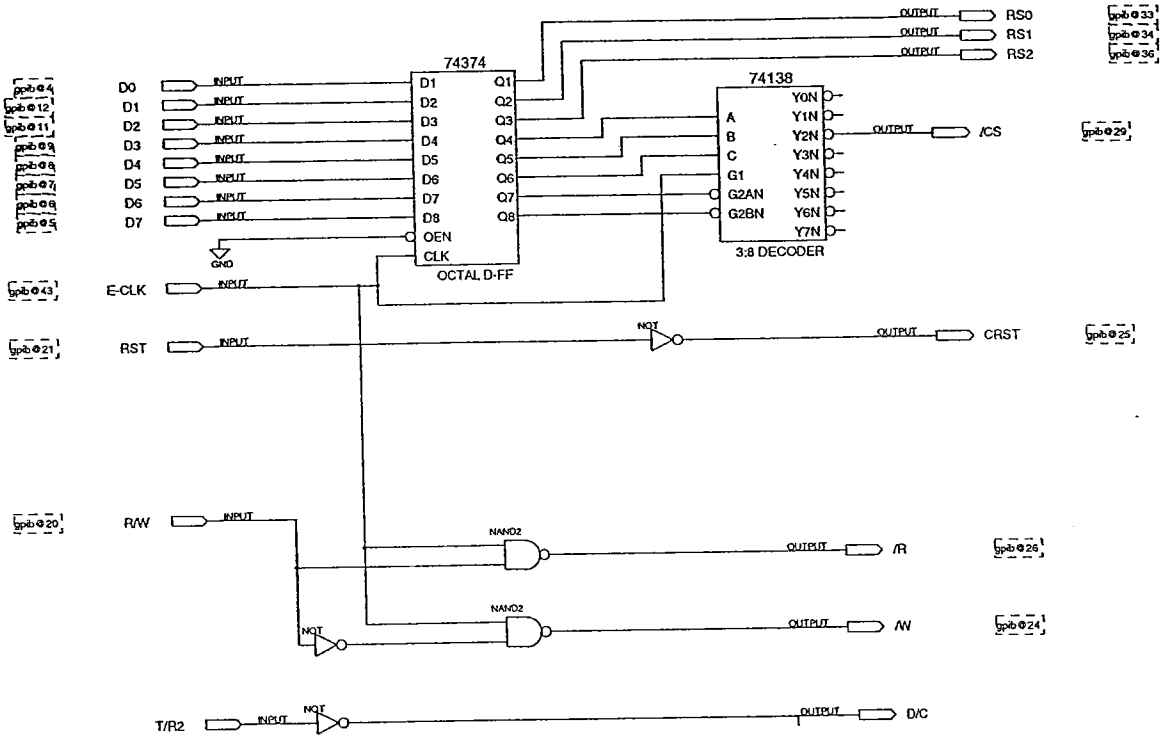
```
SUBDESIGN stimgen
(
clock      : input;
rst        : input;
  iftout   : output;
trig       : output;
)
VARIABLE
ff[8..0]      : DFFE;
 shiftin    :lcell;

BEGIN
%ff[].prn = rst;%
ff[].clk = clock;
ff[8].d=shiftin;
shiftout = ff[0].q;

IF !rst THEN
      ff[].d = B"100000000";
ELSE
shiftin=GND XOR shiftout;
ff[7].d=ff[8].q;
ff[6].d=ff[7].q;
ff[5].d=ff[6].q;
ff[4].d=ff[5].q;
ff[3].d=shiftout XOR ff[4].q;
ff[2].d=ff[3].q;
ff[1].d=ff[2].q;
ff[0].d=ff[1].q;
END IF;

trig=!ff[0].q AND !ff[1].q AND !ff[2].q AND !ff[3].q AND !ff[4].q AND !ff[5].q AND !ff[6].q
AND !ff[7].q
 AND ff[8].q;

  D;
```

LFSR Logic Test
(Wave Simulation)

LFSR Output
(Logic Analyzer Printout)



LFSR '.GDF' File



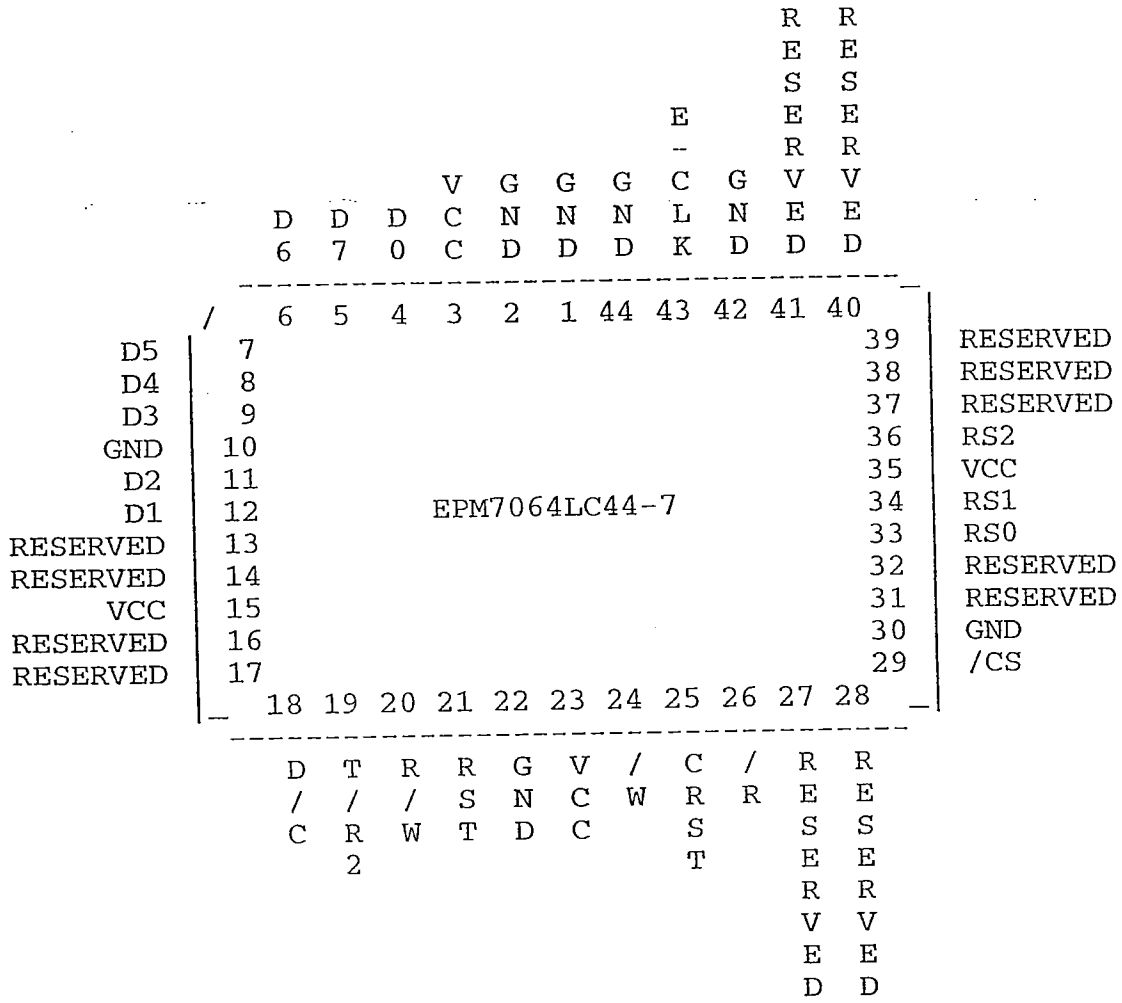
LSFR Altera Test Pinout

Untitled

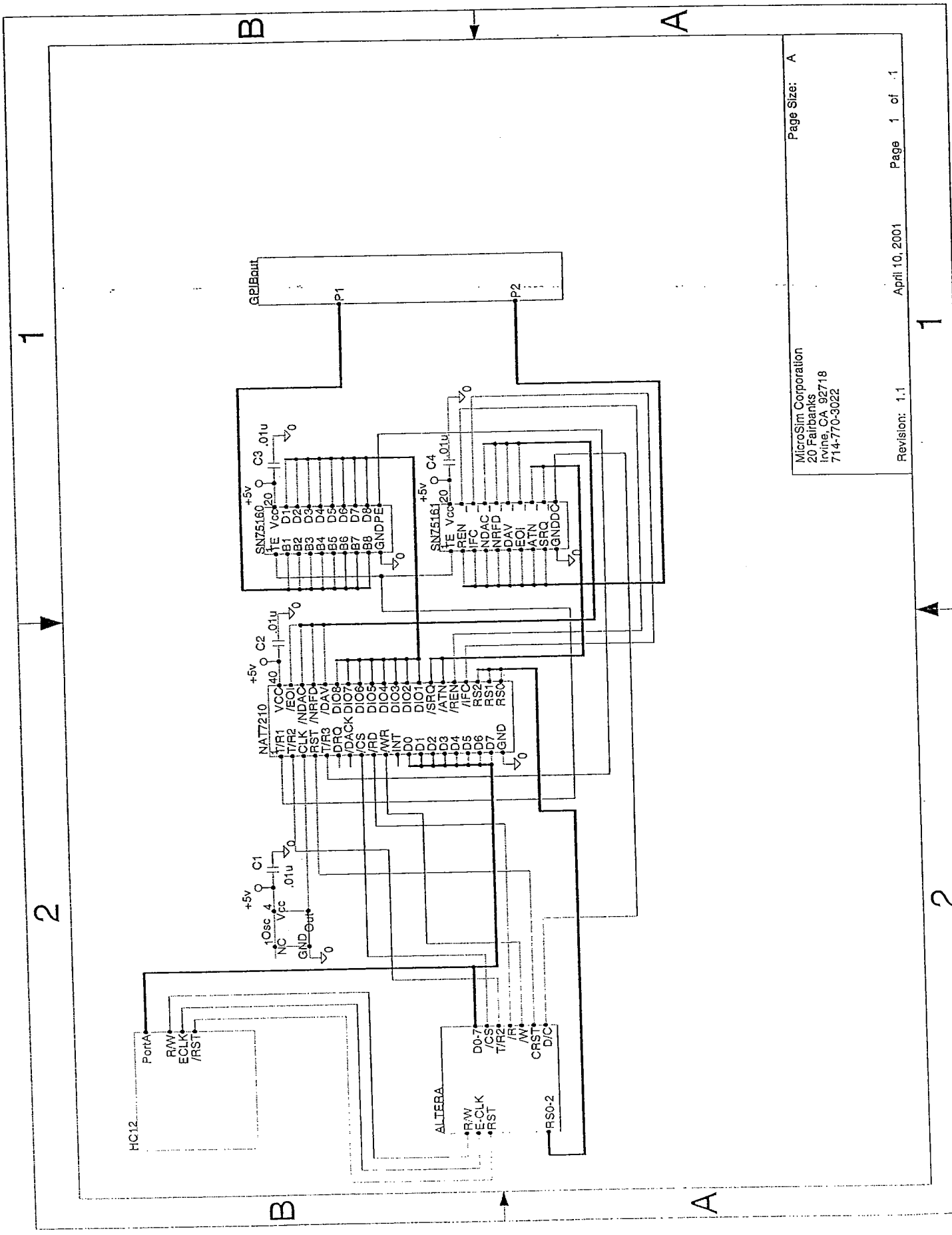
Device: EPM7064LC44-7

Device Options:

Turbo Bit	= ON
Security Bit	= OFF



Memory Decoder Schematic



ByteBlasterMV 10-Pin Female Plug Dimensions

Dimensions are shown in inches. The spacing between pin centers is 0.1 inch.

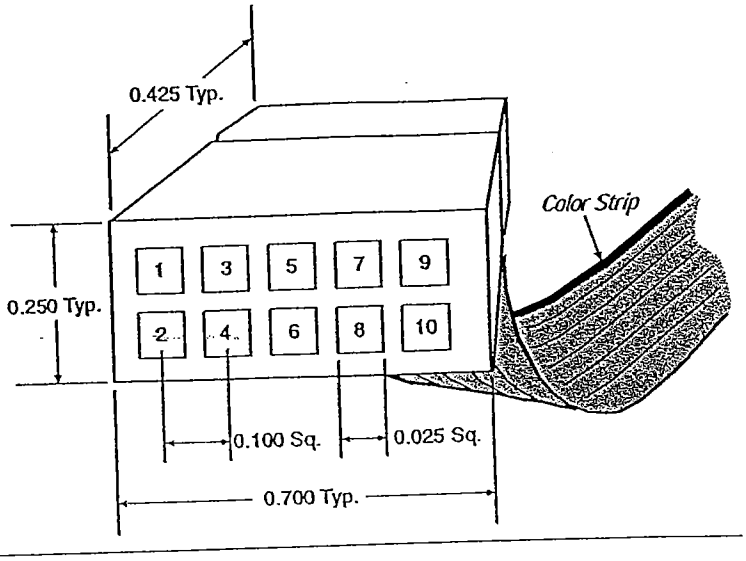


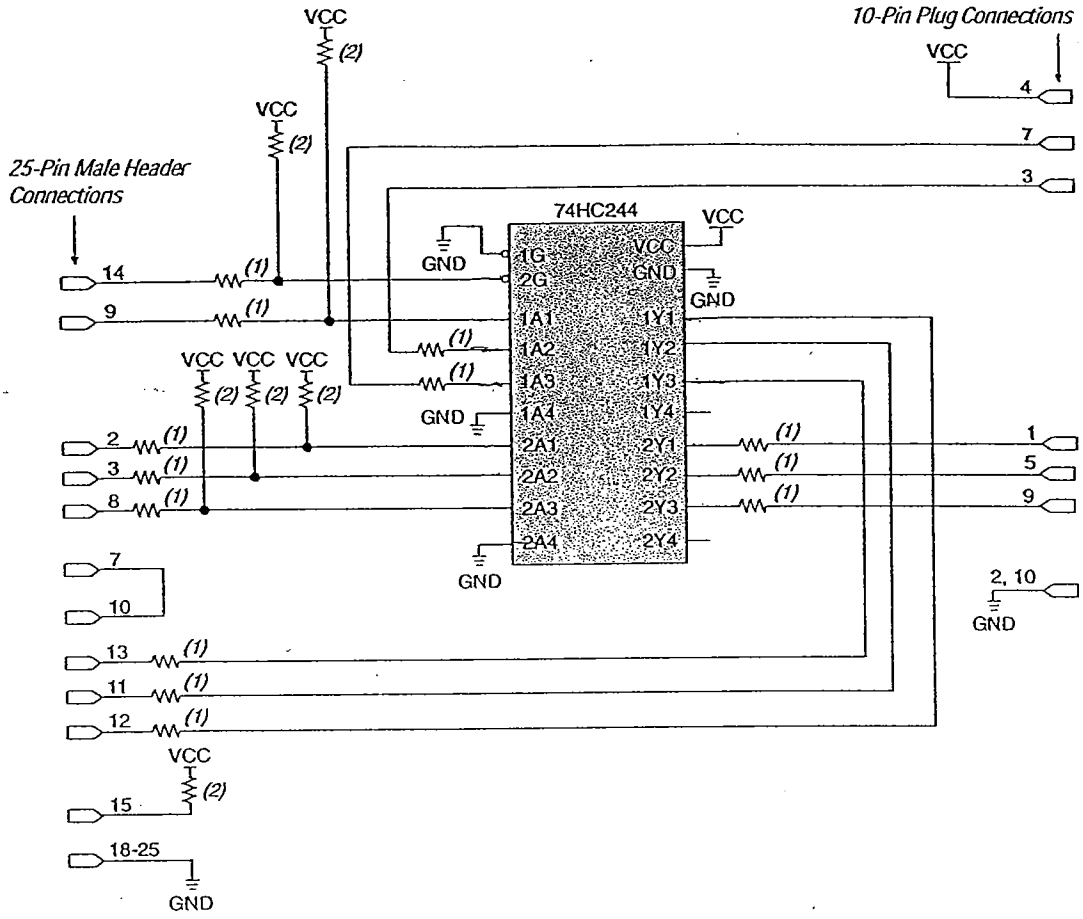
Table 2 identifies the 10-pin female plug's pin names for the corresponding download mode.

Table 2. ByteBlasterMV Female Plug's Pin Names & Download Modes

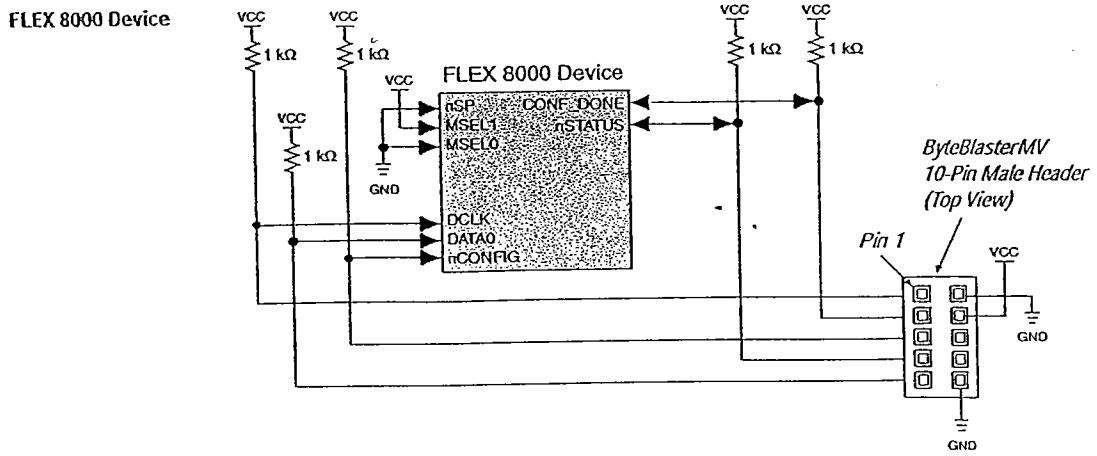
Pin	PS Mode		JTAG Mode	
	Signal Name	Description	Signal Name	Description
1	DCLK	Clock signal	TCK	Clock signal
2	GND	Signal ground	GND	Signal ground
3	CONF_DONE	Configuration control	TDO	Data from device
4	VCC	Power supply	VCC	Power supply
5	nCONFIG	Configuration control	TMS	JTAG state machine control
6	–	No connect	–	No connect
7	nSTATUS	Configuration status	–	No connect
8	–	No connect	–	No connect
9	DATA0	Data to device	TDI	Data to device
10	GND	Signal ground	GND	Signal ground

☞ The circuit board must supply V_{CC} and ground to the ByteBlasterMV cable.

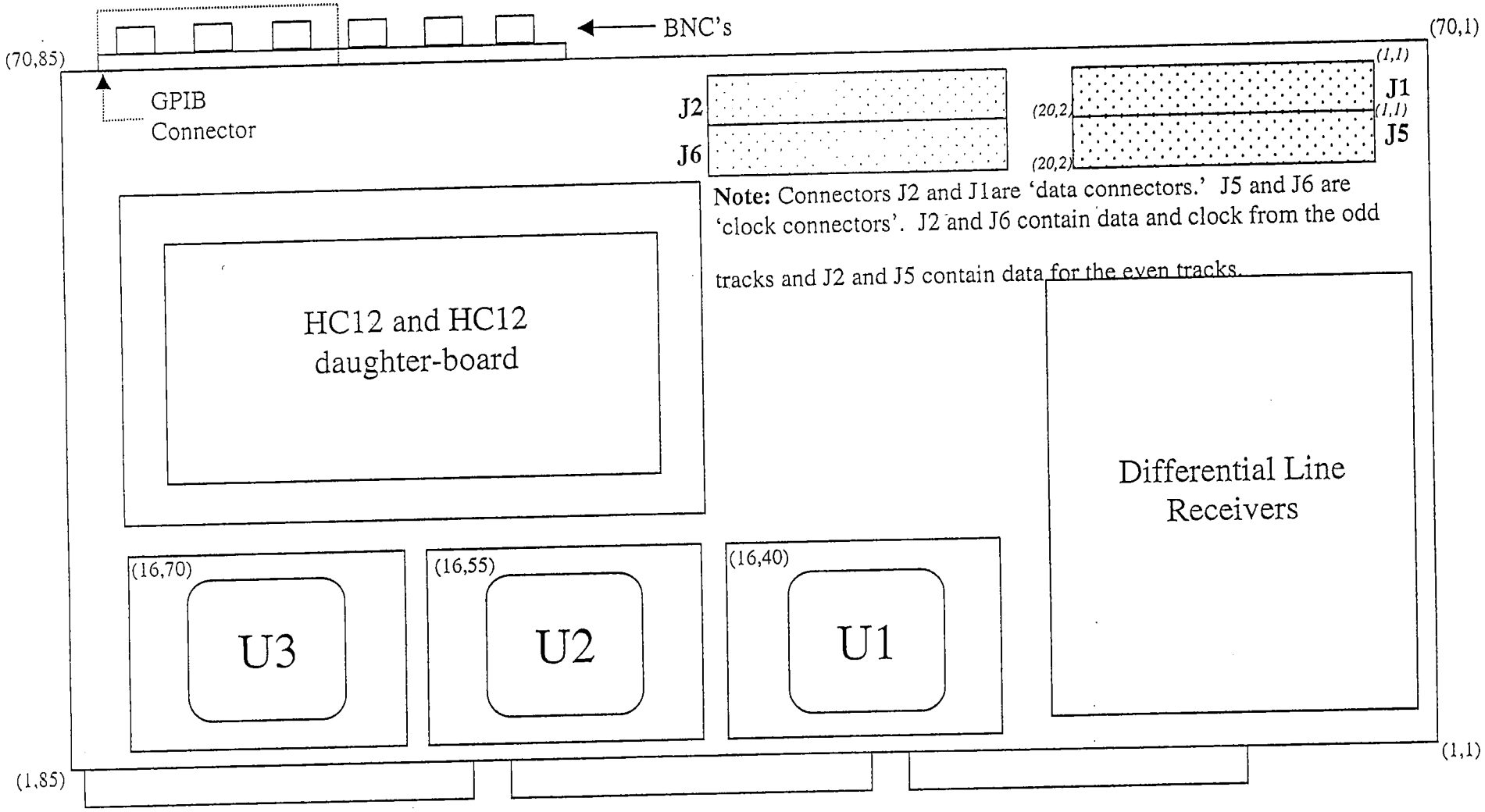
ByteBlasterMV Parallel Port Download Cable Data Sheet



Single FLEX Device Configuration with the ByteBlasterMV Cable

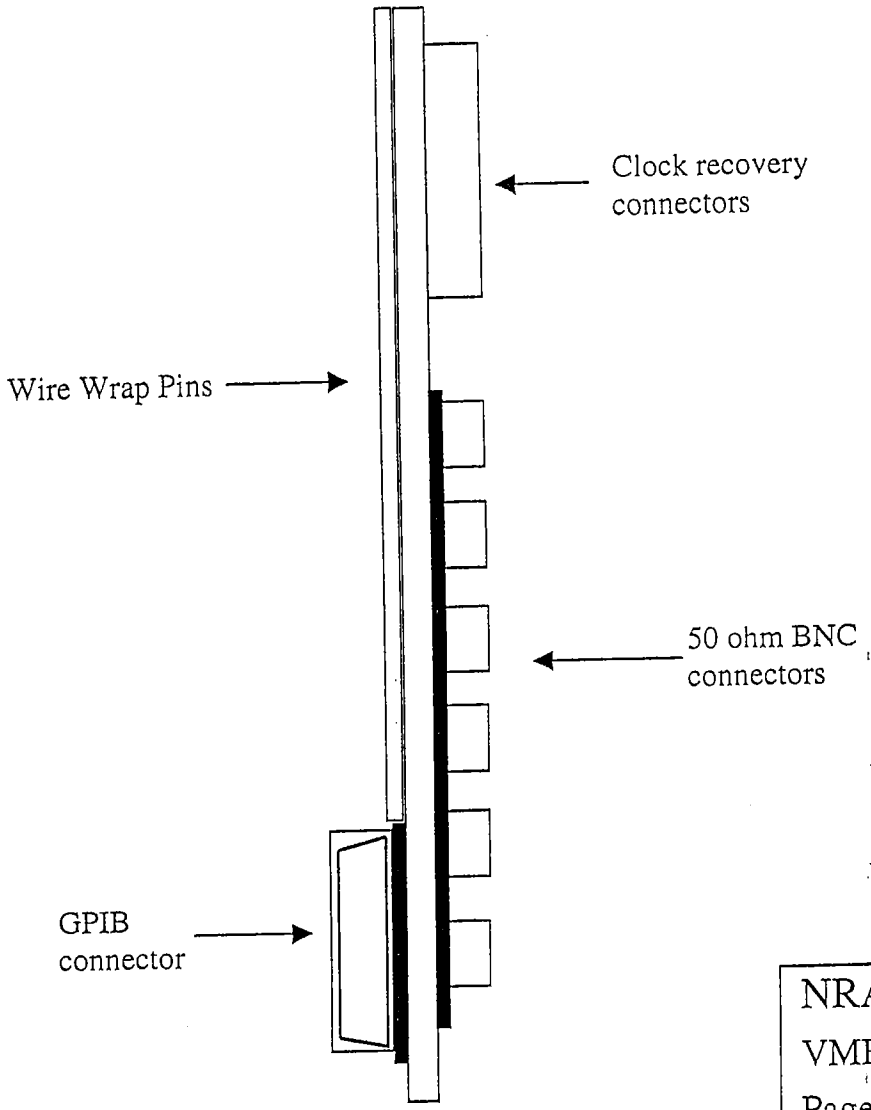


VME Card - General Layout (Top View)



NRAO Test Fixture	
VME Card - General Layout	
Page 1 of 2	Francis Martinez
April 4, 2001	Revision 0

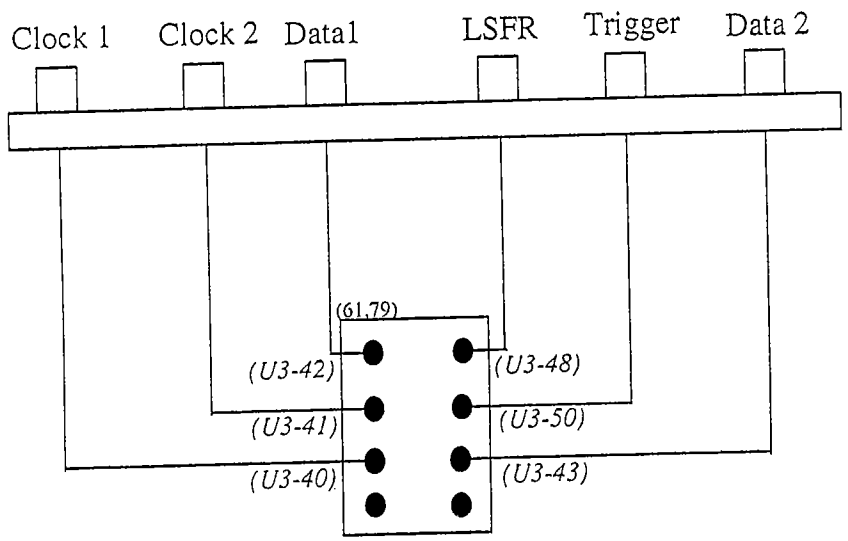
VME Card - General Layout (Side View)



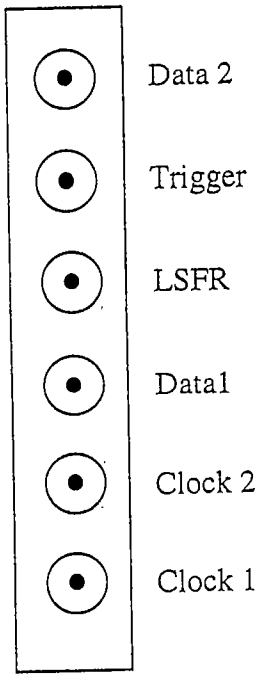
NRAO Test Fixture
VME Card - General Layout
Page 2 of 2 Francis Martinez
April 4, 2001 Revision 0

VME Card - 50 ohm drivers

Top of Card



Profile View

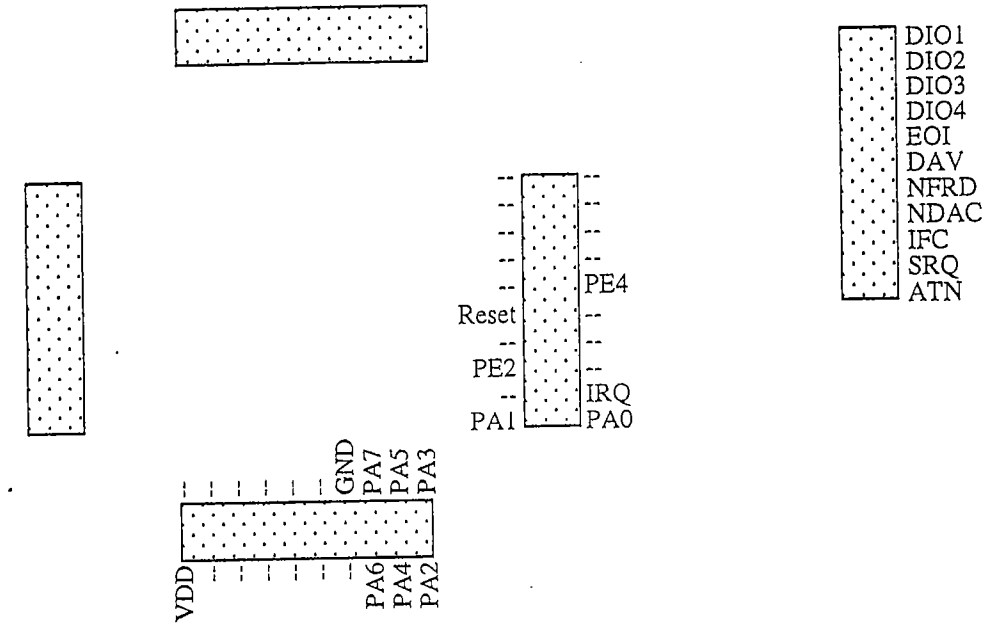


Italics correspond to the signal source

connection on the Altera chips

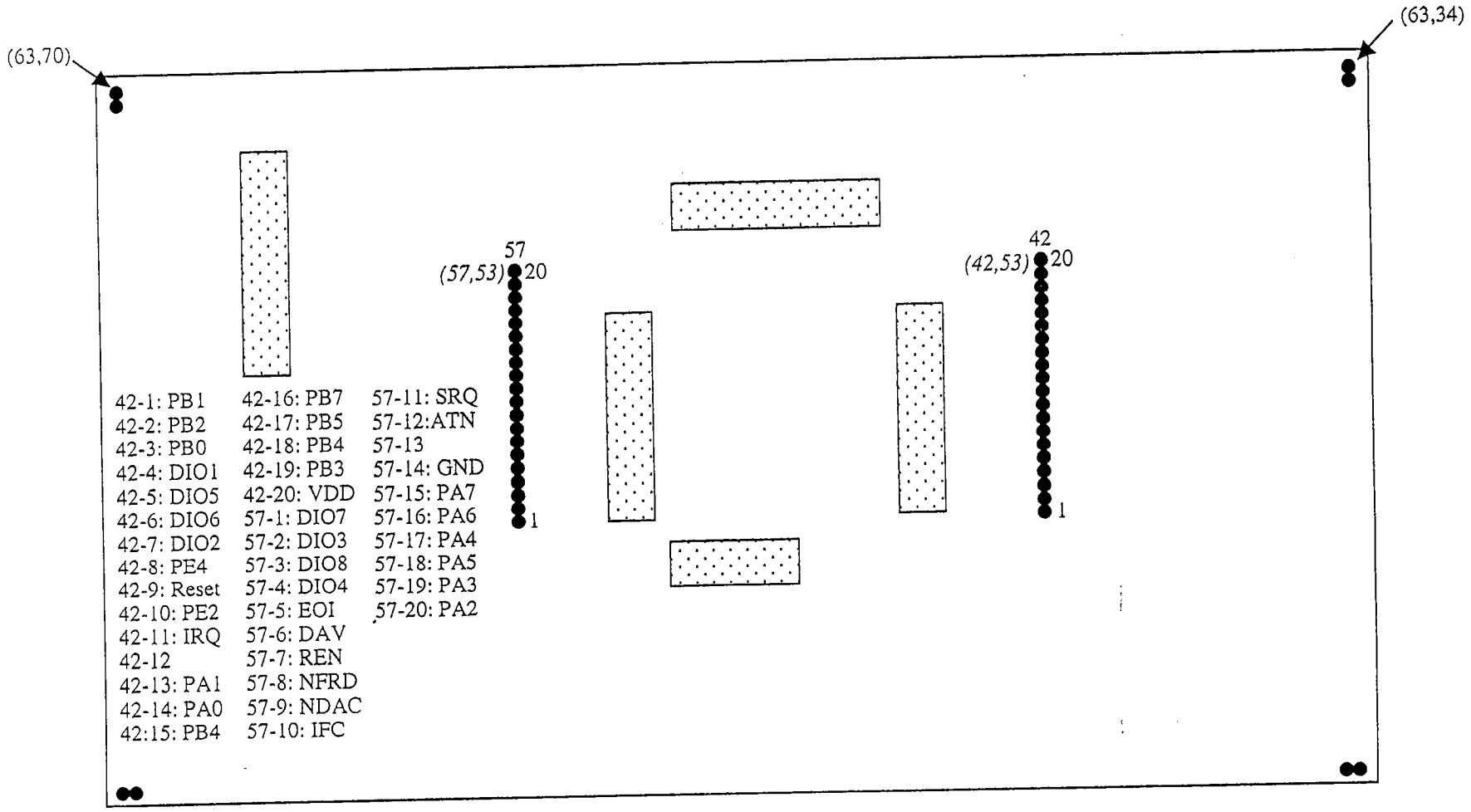
NRAO Test Fixture
50 ohm drivers
Page 1 of 1 Francis Martinez
April 4, 2001 Revision 0

VME Card - HC12 Daughter-board (underside of HC12)

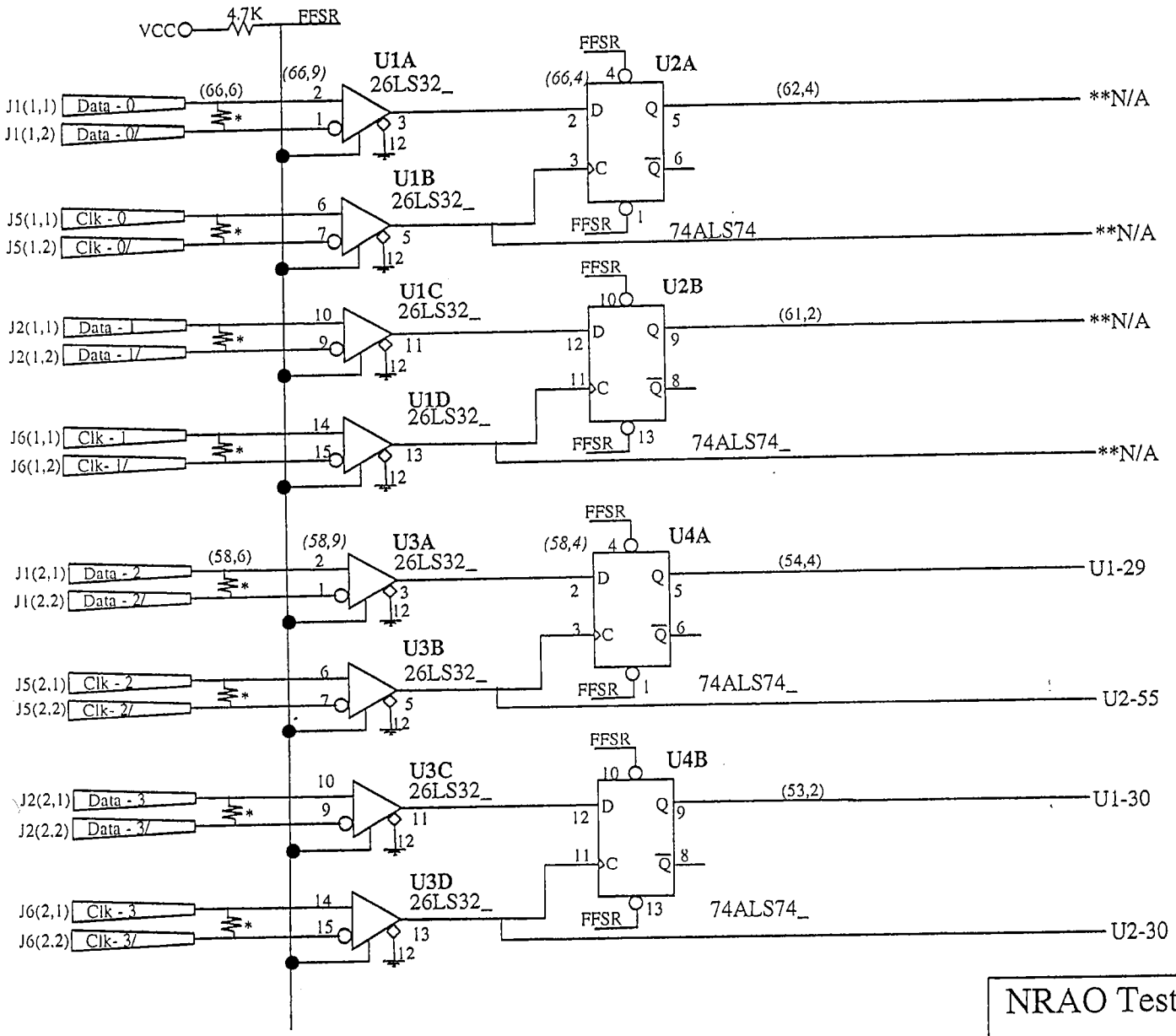


NRAO Test Fixture
VME Card - HC12 Daughter-board
Page 1 of 2 Francis Martinez
April 7, 2001 Revision 0

VME Card - HC12 Daughter-board



NRAO Test Fixture
VME Card - HC12 Daughter-board
Page 2 of 2 Francis Martinez
April 7, 2001 Revision 1



* = *8 pin 100ohm DIP resistor*

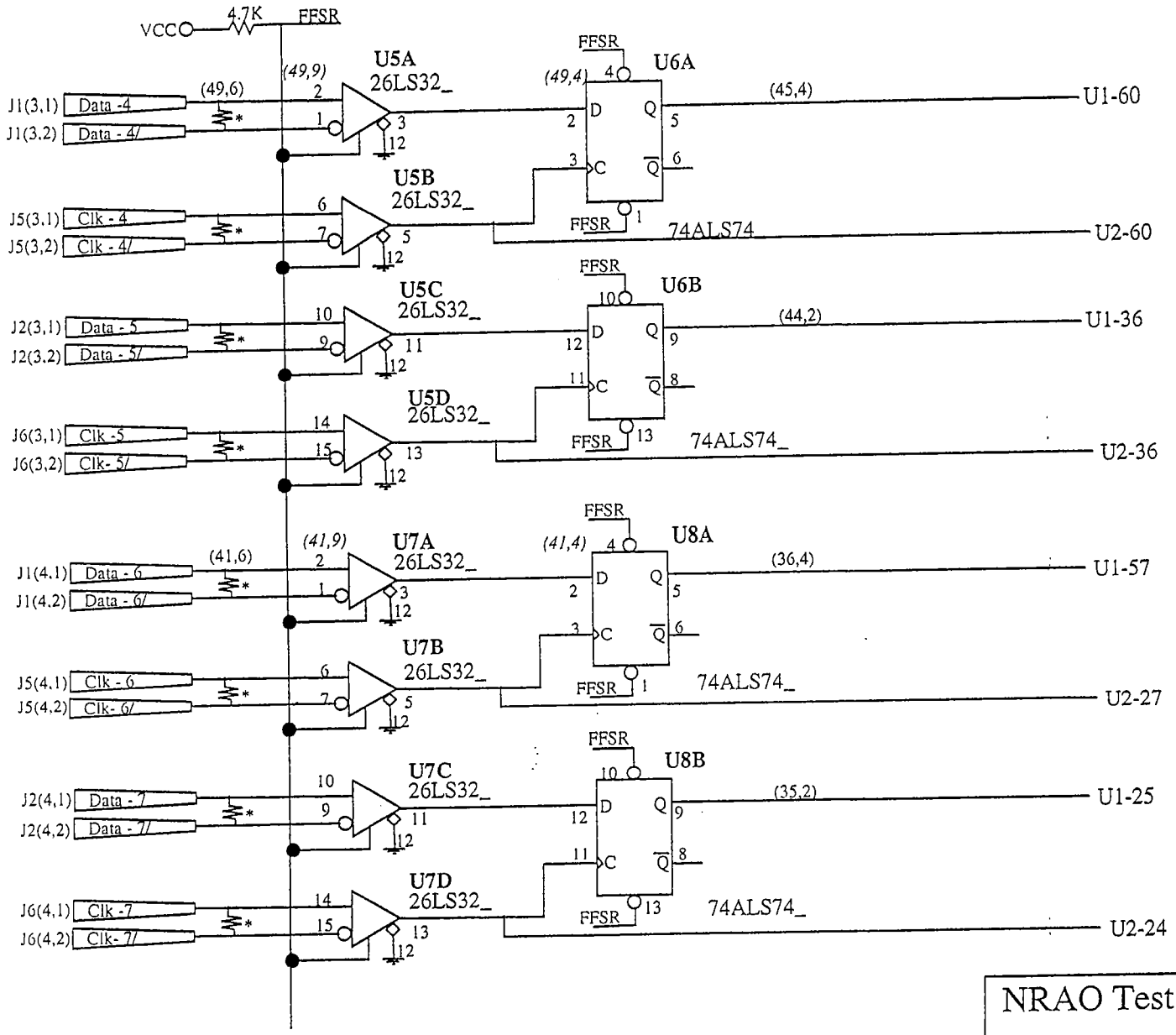
** = The NRAO told us these are not to be tested; therefore, they are not wrapped to anything.

NRAO Test Fixture

'Front-end' differential line-receivers

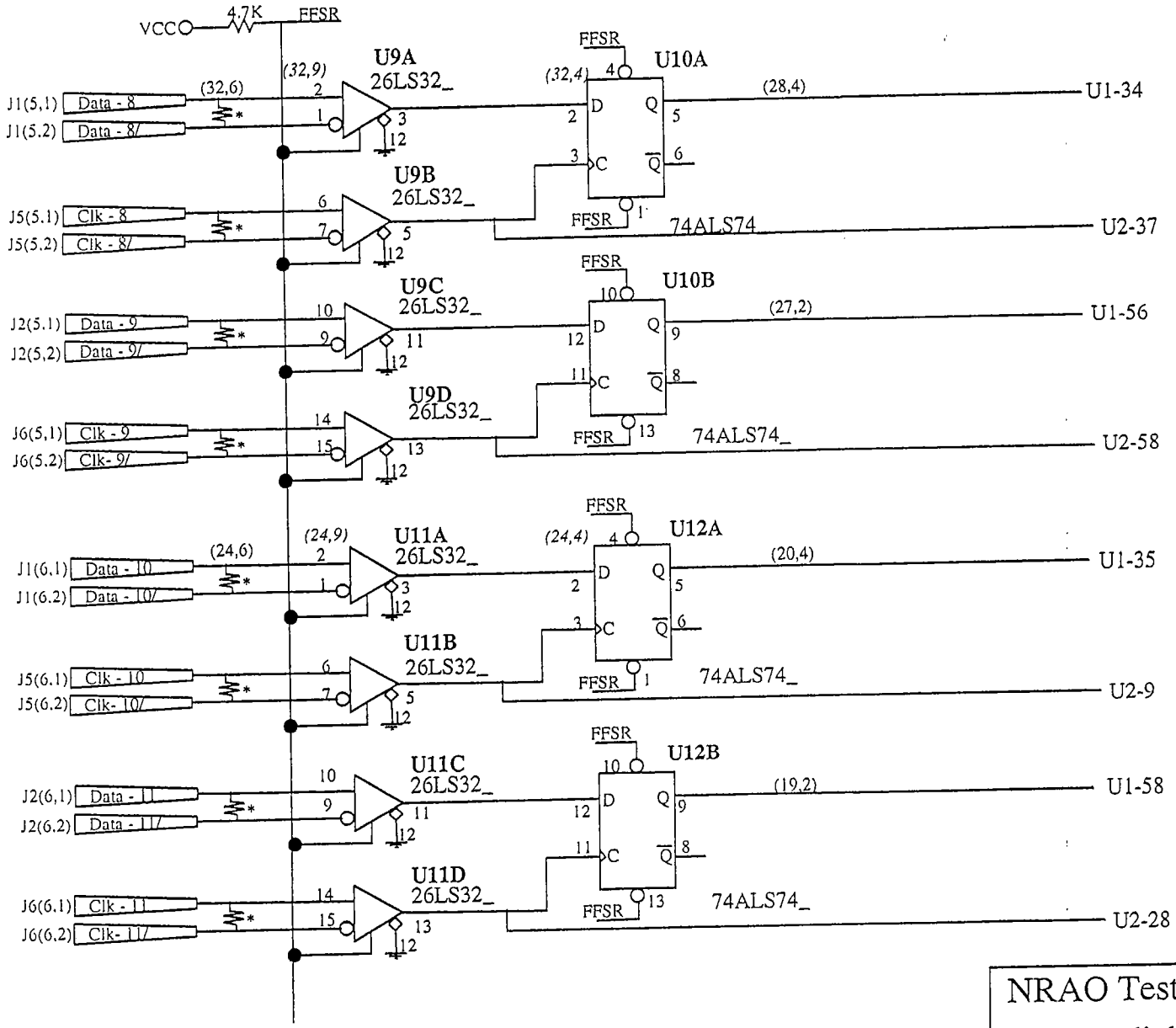
Page 1 of 9 Francis Martinez

April 1, 2001 Revision 2



** = 8 pin 100ohm DIP resistor*

NRAO Test Fixture
'Front-end' differential line-receivers
Page 2 of 9 Francis Martinez
April 1, 2001 Revision 2



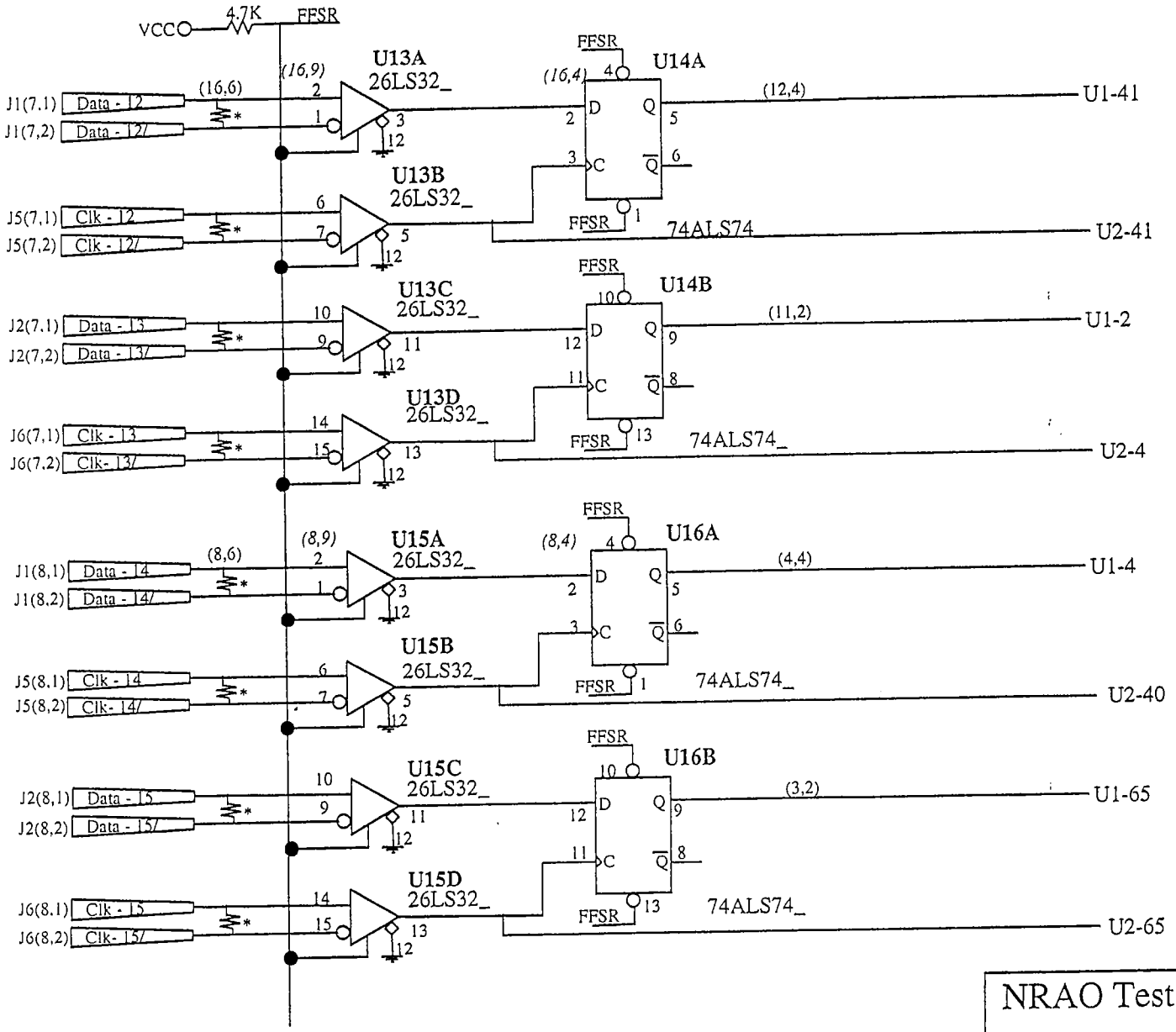
* = *8 pin 100ohm DIP resistor*

NRAO Test Fixture

'Front-end' differential line-receivers

Page 3 of 9 Francis Martinez

April 1, 2001 Revision 2



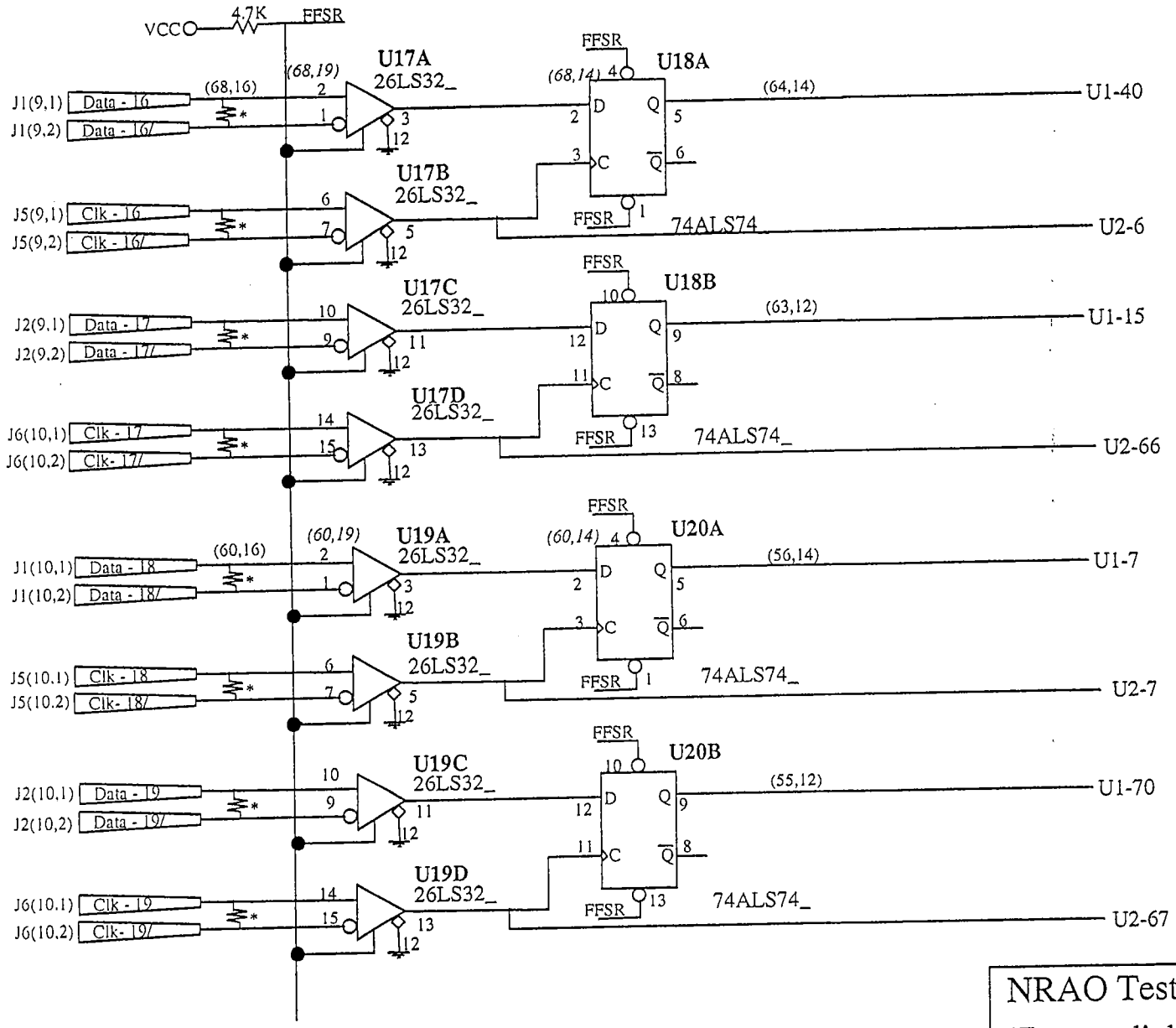
* = *8 pin 100ohm DIP resistor*

NRAO Test Fixture

'Front-end' differential line-receivers

Page 4 of 9 Francis Martinez

April 1, 2001 Revision 2



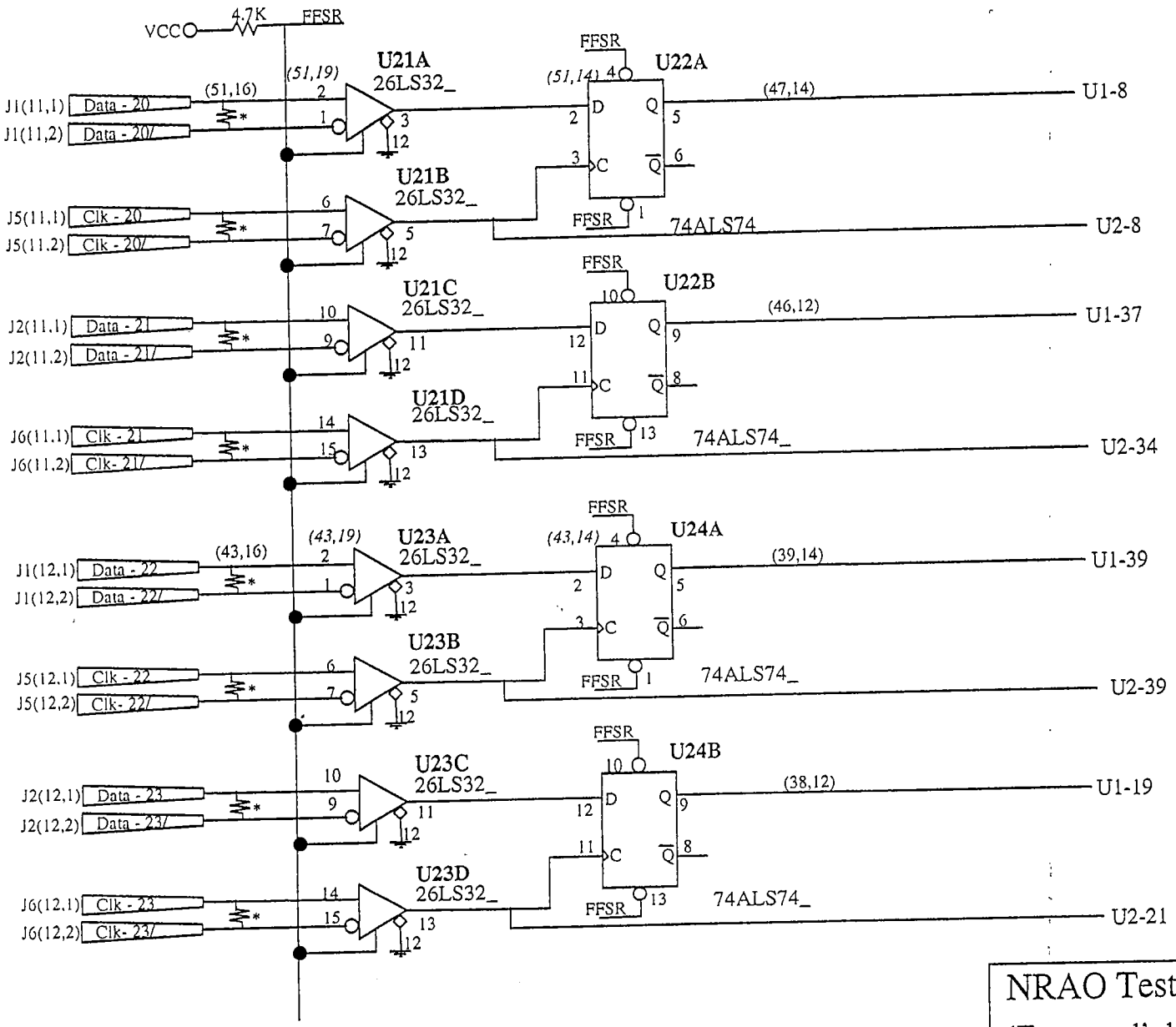
* = *8 pin 100ohm DIP resistor*

NRAO Test Fixture

'Front-end' differential line-receivers

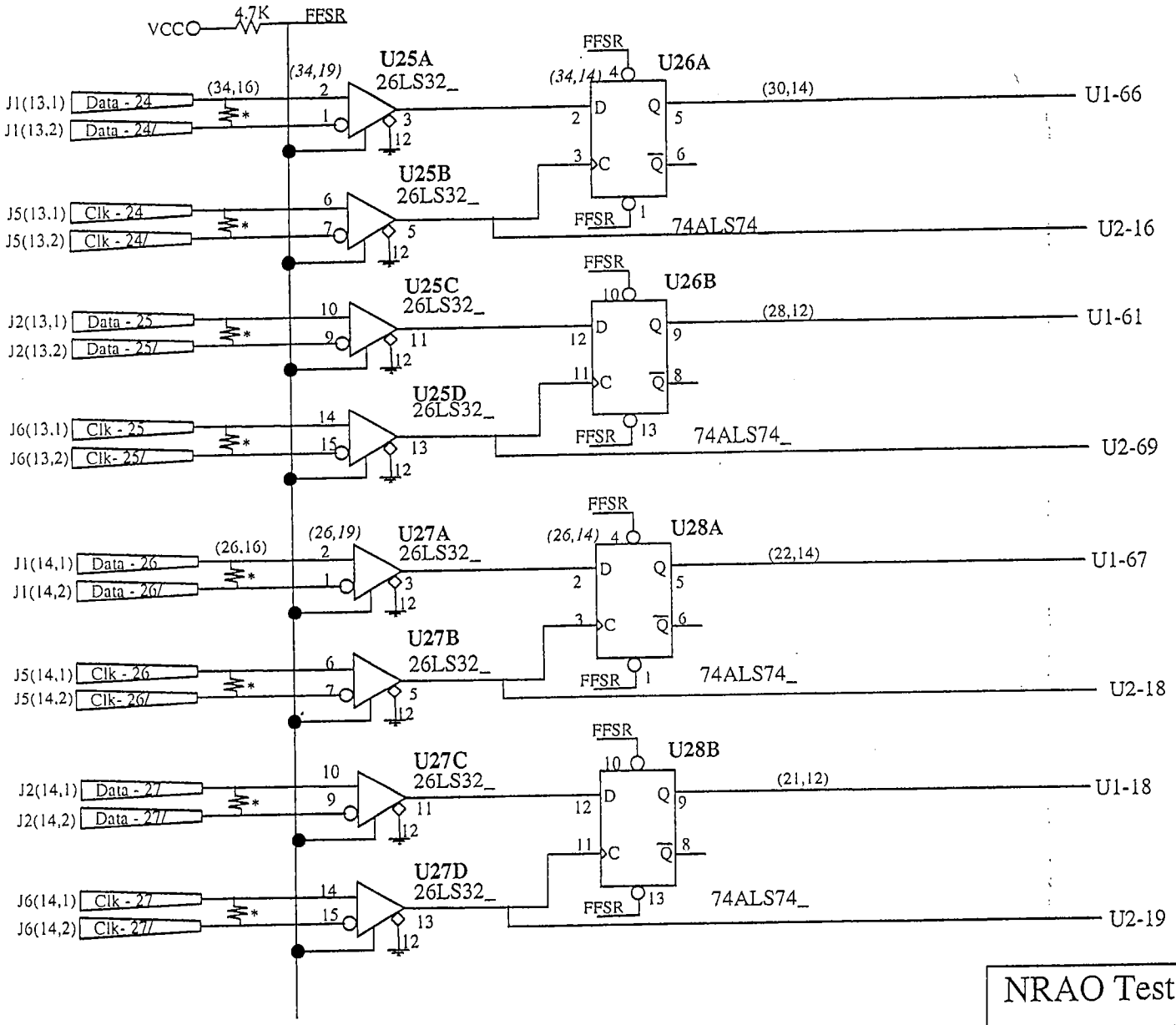
Page 5 of 9 Francis Martinez

April 1, 2001 Revision 2



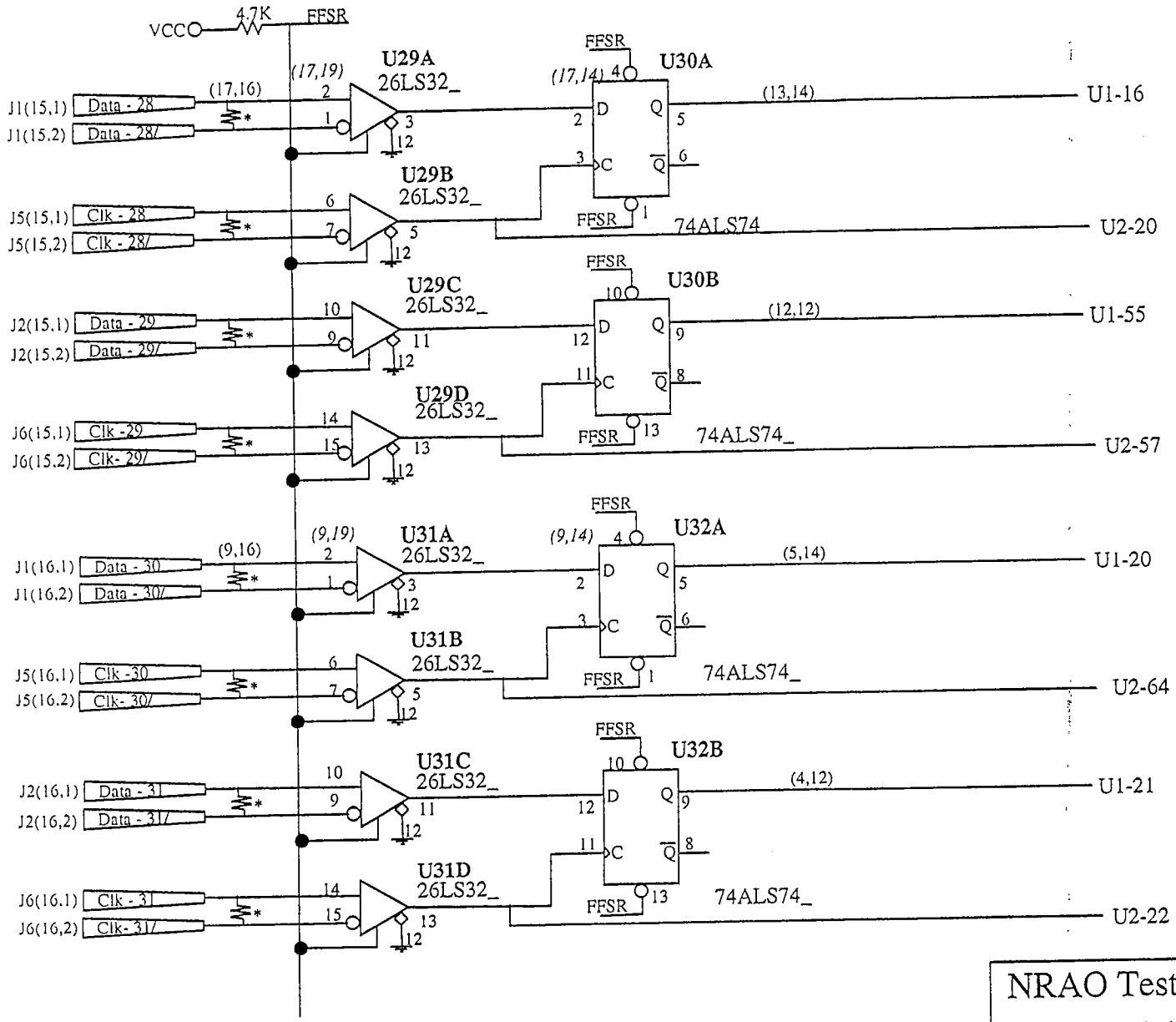
** = 8 pin 100ohm DIP resistor*

NRAO Test Fixture
'Front-end' differential line-receivers
Page 6 of 9 Francis Martinez
April 1, 2001 Revision 2



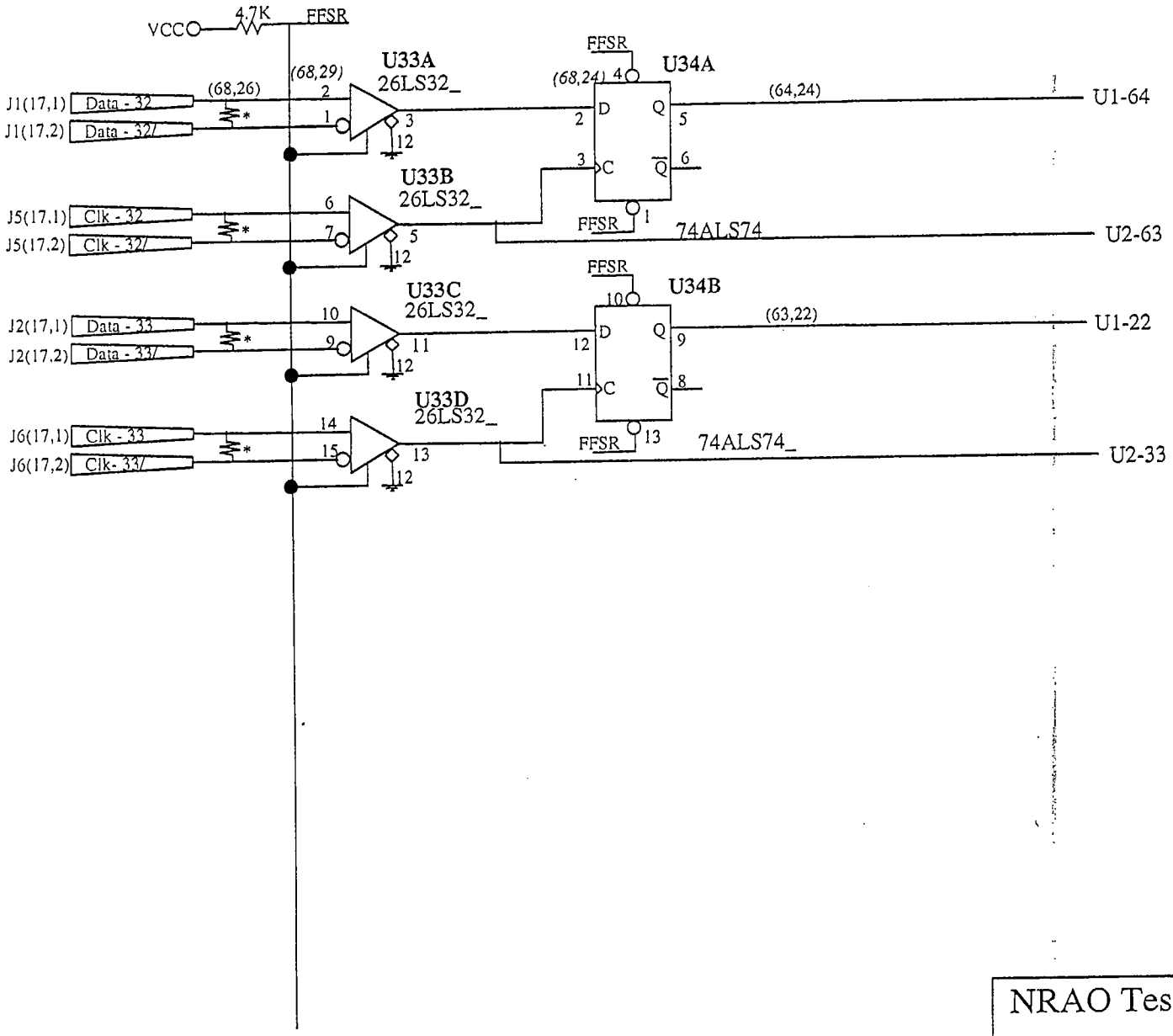
* = *8 pin 100ohm DIP resistor*

NRAO Test Fixture
'Front-end' differential line-receivers
Page 7 of 9 Francis Martinez
April 1, 2001 Revision 2



** = 8 pin 100ohm DIP resistor*

NRAO Test Fixture
'Front-end' differential line-receivers
Page 8 of 9 Francis Martinez
April 1, 2001 Revision 2



* = *8 pin 100ohm DIP resistor*

NRAO Test Fixture
'Front-end' differential line-receivers
Page 9 of 9 Francis Martinez
April 1, 2001 Revision 2

Description of NAT7210 GPIB Controller Initialization

1. Ch. 3 contains description of the internal registers and what their bits do.
2. Ch. 5 contains a step by step description of how to initialize and program the NAT7210.
3. The registers are at offsets 0-7 which are mapped to 0x1000 to 0x1700 in memory (see 3-2 for register map)

Breakdown of initialization Function

Note: The register at 0x1500 or offset 5 is control register (AUXMR) into which command codes are written that control certain activities of the NAT7210.

One of the commands that the AUXMR register accepts is a page-in command. This allows access to hidden registers (shown in bold boxes on the register map) to be accessed. (For other functions of the AUXMR register see 3-19 to 3-26.)

In addition to commands that are written to the AUXMR register, there are sub-registers in AUXMR (using the same offset) these registers are selected via the high order bits of AUXMR. These registers are AUXRA, AUXRB, AUXRE, AUXRF, AUXRG, AUXRI, and ICR whose functions are detailed on 3-27 to 3-36.

1. Write 0x02 to AUXMR this the command to reset the chip and put it into a mode
   where it is idle so that it can be programmed.

2. Write 0x50 to AUXMR this is the page-in command that allows for access to the
   hidden registers.

3. Write 0x81 to ICR2 which is a hidden register at offset 3, 0x1300 this register puts
   the ICR into a mode that allows the NAT7210 to be set to use a 10MHz clock speed.
   The bit set is bit 0 which is called MICR. The high bit 8 is supposed to be set for the
   same reason. See 3-44.

4. Write 0x25 to the ICR at offset 5 at AUXMR with the MICR bit set in ICR2.
   Using the table on 3-43 a clock frequency of 10MHz is chosen.

5. Write 0x31 to ADMR at offset 4 or 0x1400 this hex code puts the NAT7210 in
   a mode where it can implement one or two logical devices.

6. Write 0x00 to ADR at offset 6, which is the address, register (see 3-8) the highest

order bit 7 determines whether sub-registers ADR0 or ADR1 are written to by a write to this register. Bit 6 and bit 5 determine if the device is able to be a listener, talker or both. At the address given by bits 4-bit 0. This is the device address. In this case the primary address is set to be 1 with talking and listening enabled.

7. Write 0xE0 to ADR. Since the high bit is set, ADR1 is written to. Bits 6, 5 are set disabling talking and listening of the secondary address contained in bits 4-0. See 3-8 and 3-10.

8. Write 0x00 to SPMR, the serial pole mode register at offset 3, 0x1300. This register holds the serial poll status byte that is sent over when the device receives a serial poll. Bit 6 is zero because a serial poll is not being requested and the other bits are zero for no other reason than its easier that way. See 3-62.

9. Write 0x70 to the PPR at offset 5, 0x1500. PR is a sub-register of AUXMR. The highest three bits selects PPR. Bit 4 is set in order to disable the NAT7210 from responding to serial polls. See 3-58

10,11 see table 5-2 on page 5-10 this table shows the bits that need to be set in the AUXRB and AUXRI registers to produce various delays for the T1 delay. The HSTS definition is 0 for the first byte transferred and 1 for every byte there after. T1 delay is set to be 2000ns for all bytes so that USTD and TRI do not have to be changed based on HSTS. For a 2000ns delay TRI=0 and USTD=0.

10. Write 0xA0 to AUXRB sub-register of AUXMR at offset 5, 0x1500. This clears the TRI bit which it at bit 2.

11. Write 0xE0 to AUXRI sub-register of AUXMR at offset 5, 0x1500. This clears the USTD bit which is at bit 3.

12. Write 0x82 to AUXRA sub-register of AUXMR at offset 5, 0x1500. Clearing bits 4-2 keeps stuff for an EOS from being used. Setting bit 1 and clearing causes mode to be selected where there is a RFD hold off after a byte with an EOI is sent. This asserts the NRFD signal to keep data from being transmitted to it until a command is sent to the AUXMR to clear it and allow data transmission. If this gives trouble write 0x80 instead of 0x82 to cause it not to do RFD hold off. See table 3-28.

13. Write 0x00 to AUXMR at offset 5, 0x1500. This takes the NAT7210 out of idle mode and its ready for action.

Program Function main( )

The program consists of a main loop that runs forever and a series of *if* statements that determine if the DIR register should be read at offset 0, 0x1000 and its contents put into a variable.

First a 0x03 is written to AUXMR to take off the RFD hold off if there is one from the last byte received.

The first *if* statement tests for if bit 2 of ADSR register at offset 4, 0x1400 is set. This bit tells if the NAT7210 has been addressed to be an active listener.
The next *if* statement tests for if bit 0 of ISR1 at offset 1, 0x1100 has been set. This bit tells if data has been received.

GPIB.h

```
/* GPIB header file that defines the internal
 * registers of the the NAT7210 GPIB controller */

#ifndef __GPIB_H
#define __GPIB_H    1
/* the registers are mapped to extenal memory of the HC12
 * microcontroller from 0x1000 to 0x1800 */
#define _OFFSET     0x1000
#define GPIB0   (* (unsigned char *)(_OFFSET+0x0000))
#define GPIB1   (* (unsigned char *)(_OFFSET+0x0100))
#define GPIB2   (* (unsigned char *)(_OFFSET+0x0200))
#define GPIB3   (* (unsigned char *)(_OFFSET+0x0300))
#define GPIB4   (* (unsigned char *)(_OFFSET+0x0400))
#define GPIB5   (* (unsigned char *)(_OFFSET+0x0500))
#define GPIB6   (* (unsigned char *)(_OFFSET+0x0600))
#define GPIB7   (* (unsigned char *)(_OFFSET+0x0700))
#endif
```

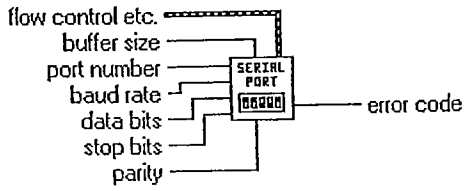
```
#include <hc12.h>
#include <DBug12.h>
#include <GPIB.h>
void initialize(void);
void main()
  {
   char data;
   initialize();
   while (1)
    {
    GPIB5=0x03; /* finish handshake */
     if ((GPIB4 & 0x04) == 1) /* listener active */
      DBug12FNP->printf("I am in the first if loop\n");     /*FRANCIS ADDED THIS LINE
*/
       {
         if ((GPIB1 & 0x01) == 1) /* data received */
           DBug12FNP->printf("I am in the second if loop\n");     /*FRANCIS ADDED THIS
 LINE*/
           {
            data=GPIB0;
            DBug12FNP->printf("data=%h\r\n",data);
           }
        }
     }
  }

void initialize()
  {

  1.GPIB5=0x02; /* write to AUXMR to make sure PON is asserted to put NAT7210
                 * in an idle state*/
  2.GPIB5=0x50; /* put in page-in mode in order to write to hidden register ICR2*/
  3.GPIB3=0x81; /* write to ICR2 to set the MICR bit to allow ICR to be set so as
                 * configure the NAT7210 to run at 10MHz*/
  4.GPIB5=0x25; /* write to ICR in order to configure the NAT7210 to run at 10MHz*/
  5.GPIB4=0x31; /* write to ADMR to set NAT7210 to be in normal dual addressing mode
*/
  6.GPIB6=0x01; /* set device address to be lucky number 1 */
  7.GPIB6=0xE0; /* disable secondary addresing */
  8.GPIB3=0x00; /* set serial poll status byte to be 0x00 */
  9.GPIB5=0x70; /* write the hidden PPR register in order to disable parallel poll
                 * response */

                /* set T1 delay to be 2000ns for all bytes sent */
  10.GPIB5=0xA0; /* clear TRI bit of the AUXRB register */
  11.GPIB5=0xE0; /* clear USTD bit of the AUXRI register */

  GPIB5=0x49; /*set CHES and NTNL */
  GPIB5=0x51; /*issue hldi command*/
  12.GPIB5=0x80; /* normal handshake mode */
  13.GPIB5=0x00; /* unassert PON in order to enguage controller */



  }
```


H

Serial Port Init

Initializes the selected serial port to the specified settings.

Click the parameters for more information.



flow control etc.

flow control etc. contains the following parameters.

Input XON/XOFF

See the Handshaking Modes topic for more information.
Default value: FALSE

Input HW Handshake

On the PC and SPARCstation, this parameter corresponds to Request To Send (RTS) handshaking.
Default value: FALSE

Input alt HW Handshake

On the PC, this parameter corresponds to Data Terminal Ready (DTR) handshaking. On the SPARCstation, this parameter is ignored.
Default value: FALSE

Output XON/XOFF

See the Handshaking Modes topic for more information.
Default value: FALSE

Output HW Handshake

On the PC and SPARCstation, this parameter corresponds to Clear to Send (CTS) handshaking.
Default value: FALSE

Output alt HW Handshake

On the PC, this parameter corresponds to Data Set Ready (DSR) handshaking. On the SPARCstation, this parameter is ignored.
Default value: FALSE

XOFF byte

XOFF byte is the byte used for XOFF (^S).
Default value: 0x13

XON byte

XON byte is the byte used for XON (^Q).
Default value: 0x11E

U16 Parity Error Byte

If the high byte is non-zero, the low byte is the character that is used to replace any parity errors found when **parity** is enabled.
Default value: 0

U16 buffer size

buffer size indicates the size of the input and output buffers the VI allocates for communication through the specified port. You may need to use larger buffers for large data transfers. The **buffer size** is in bytes.
Default value: 0

I32 port number

See the Port Number topic for a list of valid port numbers.
Default value: 0

U32 baud rate

baud rate is the rate of transmission.
Default value: 9600

U16 data bits

data bits is the number of bits in the incoming data. The value of **data bits** is between five and eight.
Default value: 8

U16 stop bits

stop bits is 0 for one stop bit, 1 for one-and-a-half **stop bits**, or 2 for two **stop bits**.
Default value: 1 bit

U16 parity

parity is 0 for no parity, 1 for odd parity, 2 for even parity, 3 for mark parity, or 4 for space parity.
Default value: 0

I32 error code

error code is -1 if **baud rate, data bits, stop bits, parity,** or **port number** are out of range, or if the serial port could not be initialized. Check the values of **baud rate, data bits, stop bits, parity,** and **port number**. If these values are valid, verify that the serial port has been initialized. Refer to the Error Code topic, for a list of error codes.

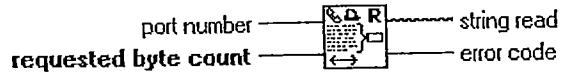
You can connect **error code** to one of the error handler VIs. These VIs can describe the error and give you options on how to proceed when an error occurs. For more information on using the error handler VIs, refer to Error Handler VIs.

Some error codes returned by the serial port VIs are platform-specific. Please refer to your system documentation for a list of error codes.

Serial Port Read

Reads the number of characters specified by **requested byte count** from the serial port indicated in **port number.**

Click the parameters for more information.



I32 port number

See the Port Number topic for a list of valid port numbers.

U32 requested byte count

requested byte count specifies the number of characters to be read. If you want to read all of the characters currently at the serial port, first execute the Bytes at Serial Port VI to determine the exact number of bytes ready to be read. Then use the **byte count** output of that VI as the **requested byte count** input to the Serial Port Read VI.

abc string read

The VI returns the bytes read in **string read.**

I32 error code

If **error code** is non-zero, an error occurred Refer to the Error Code topic, for a list of error codes.

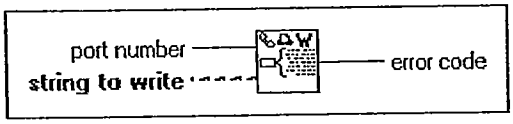
You can connect **error code** to one of the error handler VIs. These VIs describe the error and give you options on how to proceed when an error occurs. For more information on using the error handler VIs, refer to Error Handler VIs.

Some error codes returned by the serial port VIs are platform-specific. Please refer to your system documentation for a list of error codes.

Serial Port Write

Writes the data in **string to write** to the serial port indicated in **port number**.

Click the parameters for more information.



port number The parameters for serial **port numbers** depend on the platform that you use: Windows, Macintosh, or UNIX. Refer to Windows Serial Port Numbers, Macintosh Serial Port Numbers, or UNIX Serial Port Numbers for more information about each platform.

string to write is the data to be written to the serial port. If the number of characters in **string to write** is greater than the **buffer size** specified in Serial Port Init, the number of characters equal to the **buffer size** will be written.

If **error code** is non-zero, an error occurred.

You can connect **error code** to one of the error handler VIs, which describe the error and give you options on how to proceed when an error occurs.

Some error codes returned by the serial port VIs are platform-specific. Please refer to your system documentation for a list of error codes.