Digital Storage Oscilloscope

Thomas Grocutt

April 2000

Acknowledgements

I would like to thank my project supervisor Milos Kolar for his time, help, and encouragement, Philip Cupitt for his previous work and for proving the basic concepts, and the technical staff in the engineering department, particularly Ian Hutchinson and Anthony McFarlane.

I would also like to thank the programming team that created MS Office 97, for making the writing of this report so… challenging.

Summary

The objective was to design and build a low cost, high performance, dual channel Digital Storage Oscilloscope (DSO). Costs were minimised compared to conventional, commercial DSO's by utilising a personal computer to provide both the display functions and the majority of the control functions. The remaining control functions were implemented using a Field Programmable Gate Array making the system very flexible and enabling it to be customised for specific tasks in abnormal situations.

One of the major limitations of commercial DSO's is their small storage depths. To address this, the project DSO has a storage depth approximately 400 times greater than most commercially available DSO's. It contains 6Mbits of memory capable of capturing nearly 8ms when sampling at 100MHz. The DSO hardware has a low component count making extensive use of surface mount technology. This reduces cost, enhances reliability and increases the signal to noise ratio producing traces with noticeably less noise than the commercial reference DSO.

The project DSO also features a built in Logic Analyser, advanced triggering modes (such as pre-trigger) and variable sample rates together with user friendly, Windows based, software makes this system both versatile and easy to use.

Contents

Diagrams

1) Introduction

The aim of the project was to design a PC based Digital Storage Oscilloscope (DSO. A DSO built by Philip Cupitt proved the basic concepts behind the project. A standard oscilloscope displays the changes in a voltage over time, as the display is continuously updated with the current state of the input signal. A standard oscilloscope is of limited use for non-repeating signals or for observing signal glitches.

A storage scope is more useful as it captures and stores the signal. Which can then be displayed to the user. Because the screen is not continuously refreshed with the current state of the signal the scope can be used to analyse non-repeating signals and signal glitches. Both analogue storage and digital storage scopes are available, with digital scopes being by far the most common.

Normal oscilloscopes use an electron beam, which is swept across a phosphor screen, the vertical deflection of the beam being proportional to input voltage. Areas of the screen that are bombarded by the electron beam will emit light, resulting in an image that shows the waveform of the input signal. Analogue storage scopes use a specially modified cathode ray tube (CRT) to store the signal.

Unlike a conventional scope an analogue storage scope only sweeps the electron beam (the write beam) across the screen once. Then by exposing the entire screen with a low level electron beam (the flood beam) this image can then be preserved for several minutes.

Unfortunately the faster the write beam is swept across the screen the shorter the residence time of the image. This results in fast recording only being visible for a few ten’s of milliseconds. To solve this problem the write beam is scanned onto an intermediate target at high speed, then before the image decays it is transferred to the phosphor screen by a read gun. As the image is transferred to the screen at a relatively low speed it will have a high residence time.

A Digital Storage Oscilloscope (DSO) uses digital memory to store a waveform. In order to do this the incoming signal must first be digitised, once this is complete the data in the memory can be continuously replayed through a digital to analogue converter and displayed on a CRT. Unlike analogue storage scopes the captured waveform does not decay over time.

The speed and length of the recording are parameters that limit the type of signals that can be analysed by a DSO. Most commercially available DSO's allow the user to increase the length of the recording only by reducing the sampling frequency. This is not always desirable as it can lead to aliasing of the signal and the loss of small details like signal glitches that can seriously restrict the usefulness of a DSO, as engineers are often interested in these very glitches.

The memory capacity of a DSO determines the maximum length of a recording for a given sample rate. Therefore increasing the storage capacity of a DSO enables the recording to be lengthened without reducing the sample rate.

Commercial DSO's cost up to £5000. This limits the number that can be purchased and can lead to engineers sharing equipment, reducing productivity. The cost of a DSO can be dramatically reduced by using a standard PC for the display. As PC's are common place in a lab this would reduce the need to share equipment. Although PC based DSO's are currently available they often have small memory capacities and low maximum sample rates, in addition they are still quite expensive (~ £400).

Logic Analysers like DSO's capture data and then display it to the user. however instead of displaying voltage variations a Logic Analyser will simply show the Logic State of several channels in parallel. Just like DSO's, Logic Analysers suffer from the same problems of cost and low storage capacities. The only major difference between a Logic Analyser and a DSO is the display, in a PC based system this could be handled in software therefore the two could be combined with little extra cost.

The specification of the project DSO should be as follows:-

- The module must connect to a standard PC with little or no modifications, and should be easy to set-up.
- The DSO must be capable of high sample rates of around 40MHz. Lower sample rates should be user selectable from within the software.
- The DSO should have a large storage depth of 3Mbit. This would give a recording length of around 10mS at a sample rate of 40MHz.
- The cost of the final DSO must be less than £100. If the unit was commercially produced this price could be significantly reduced through bulk purchasing.
- The DSO should have 2 analogue channels with an 8Bit resolution.
- The unit should also be able to operate as a 16Bit Logic Analyser with no analogue channels, or an 8Bit Logic Analyser with 1 analogue channel.
- The unit should be able to trigger from either of the analogue channels or the logic input. The unit must also have a Pre-Trigger mode to allow the user to capture data that occurs before the trigger point.
- The software should allow the user to save captured data for future reference, it should also be able to export this data in a standard format for use in a spreadsheet.

A DSO of this specification would have the following applications:-

- Long term signal observation. As captured data could be saved to disk, several readings could be taken over time and compared.
- Debugging complex none repeating systems.
- Performance analysis of ADC's and DAC's, possible because the unit would be capable of simultaneously capturing both an analogue channel and an 8 Bit data bus.
- Tracing the source of 'signal glitches'.

1.1) Previous Work

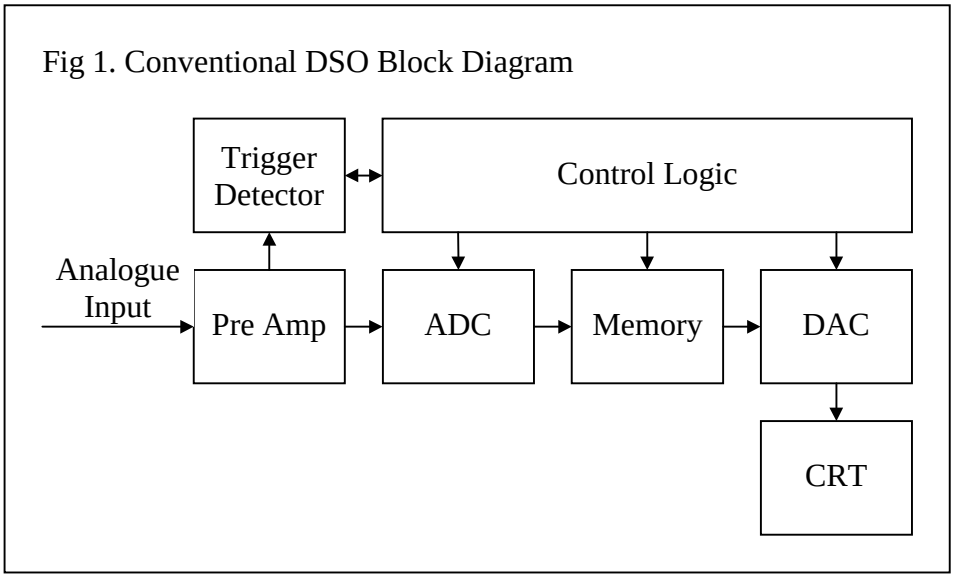
The previous DSO produced by Philip Cupitt was capable of sampling a single channel at 40MHz, it had no onboard triggering and therefore required an external trigger source. The system was based on the ‘Hitachi HM530281’ Frame Memory and the ‘Philips TDA8703’ Flash ADC, the control signals for these two chips were generated by a series of Programmable Logic Devices (PLD’s), 74 Series logic chips and a ‘PIC’ microcontroller. This system proved the concept but was not a usable tool due to the lack of onboard triggering and user selectable sample rates. These limitations were addressed by this project.

2) Theory

2.1) Digital Storage Oscilloscope Operation

A basic block diagram of a conventional DSO can be seen in fig 1.

Fig 1. Conventional DSO Block Diagram



The analogue signal being monitored is fed into a pre amp, which changes it's amplitude so that it falls within the input range of the Analogue to Digital Converter (ADC) and the trigger detector. When the resulting voltage crosses a threshold set by the user the trigger unit signals the device to start recording. The ADC samples the output of the pre amp at regular intervals and the digital output from the ADC is then stored in consecutive locations in the memory. When the memory is full the recording is stopped.

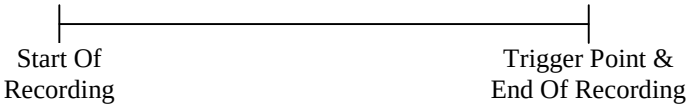
The Digital to Analogue Converter (DAC) continuously scans through the recording producing a repeating analogue signal representing the contents of the memory, which is sent to the Cathode Ray Tube (CRT) for display. This is required because the CRT image will fade away if not continuously refreshed.

If changes in the input voltage re-triggers the DSO then the memory is overwritten with a new recording unless the user puts the system into HOLD mode, Hold mode allows the user to analyse the signal trace for as long he/she requires.

As the number of samples in any recording is determined by the size of the memory, changing the sample frequency of the ADC's enables the user to alter the length of the recording.

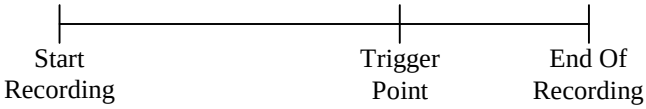
Most modern DSO's have additional trigger modes that allow the user to make better use of the available storage space. One of the most important of these is the Pre-Trigger mode, which allows the user to capture events that occur before the trigger pulse. This is achieved by the DSO continuously recording the analogue signal until the trigger pulse is detected. This leaves data from before the trigger point still in the memory, as can be seen in fig 2.

Fig 2. Pre-Trigger



By adding a delay between detecting the trigger point and terminating the recording it is possible to capture data from before and after the trigger point, as shown in fig 3. This can be very useful as it allows engineers to look at events both preceding and following an error or glitch that triggers the DSO.

Fig 3. Variable Pre-Trigger



2.2) PC Based Digital Storage Oscilloscope

Digital Storage Oscilloscopes are expensive pieces of equipment. This makes it unpractical to give to every engineer. To alleviate this problem several manufacturers have started making DSO modules for PC's. This dramatically cuts down the cost of a DSO as the PC monitor can be used for display. Additionally much of the complex control structure can be transfered into software.

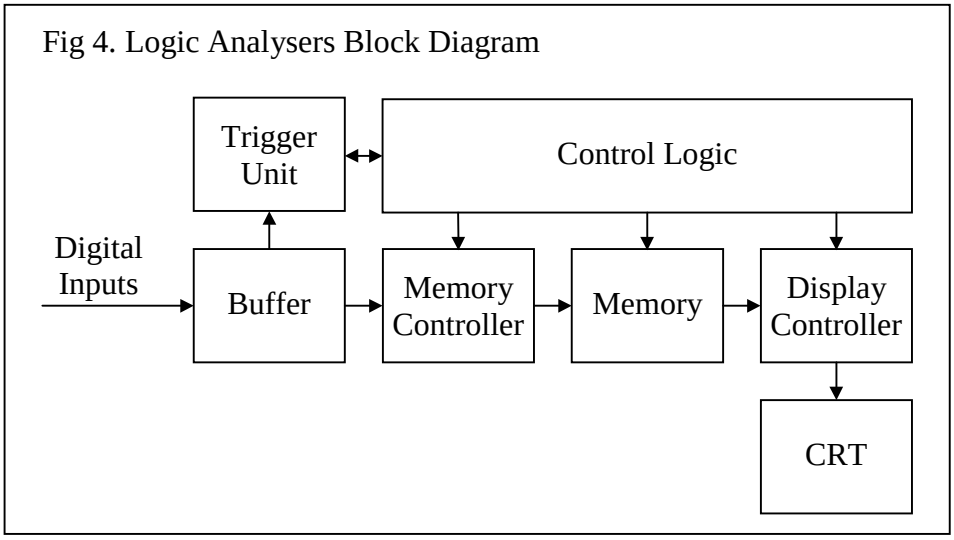
Because of the relatively low data transfer rates (~500KB/S) that are available through the parallel port and other suitable IO channels of a PC, it is not possible to transfer the data directly from the ADC's to the computer. This approach would also cause problems due to the non-real time nature of most PC operating systems. To resolve this problem PC based DSO's have an onboard memory. This memory is used to buffer the recorded data so it can be subsequently transferred to the PC at a slower speed.

A typical commercial PC based DSO costing £500 would be capable of a 2 channel recording at 25 Mega Samples Per Second (MSPS), in an 8 bit resolution with 8K sample depth. Although this is considerably cheaper than a standalone DSO of the same specification, it is still relatively expensive. As with normal DSO's the small sample depth is very limiting.

2.3) Logic Analysers

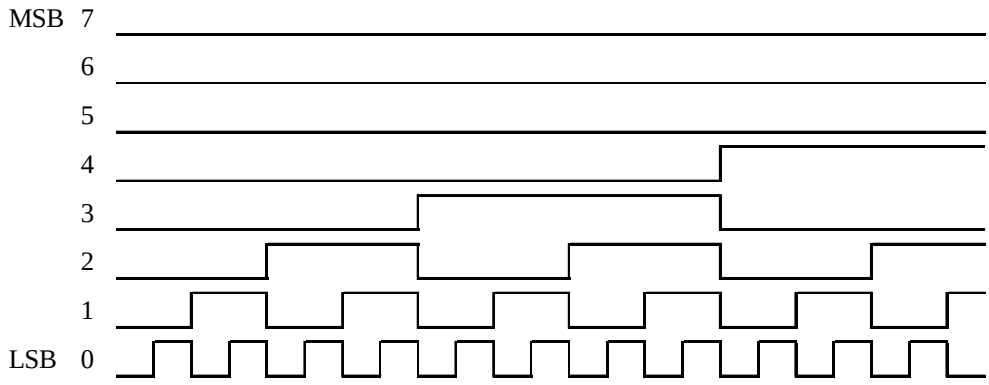
The block diagram of a typical logic analyser can be seen in fig 4.

Fig 4. Logic Analysers Block Diagram



Logic analysers record digital data from a bus. This data initially is fed into a buffer, and then to a trigger unit whish watches the data for a user defined bit pattern. When this is found the memory controller starts recording the incoming data. When the memory is full, the recording process is stopped. The display controller now reads through the stored data and displays it on the CRT as a series of lines, showing how the state of the bits change with time (See fig 5). Just like DSO's, logic analysers usually have a pre-trigger mode so that the user can capture events that occur before the trigger point. Because of their ability to capture data and display it to the user, logic analysers are very useful in fault tracing and in the evaluation of digital systems.

Fig 5. Logic Plot Of 8 Bit Counter



Due to their similarities with DSO's, logic analysers are also available as add on modules for PC's. However both the standalone and the PC based systems suffer from the same small storage depths that plague DSO's.

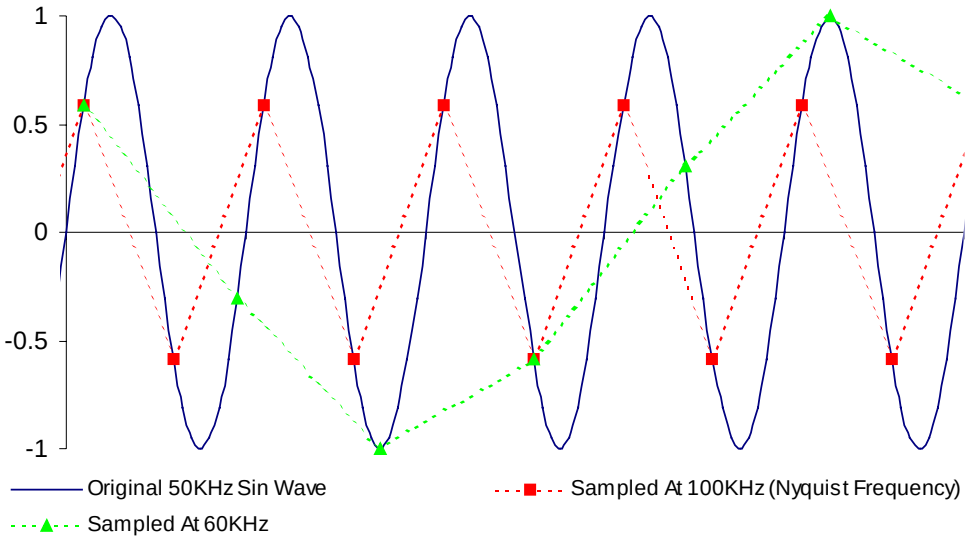
2.4) Sample Rates and Aliasing

Nyquist's Sampling Theorem states:-

> "If a signal contains no frequency components above a frequency f_{max} the signal can be uniquely represented by equally spaced samples if the sample frequency f_s is greater that twice f_{max}. That is, the sampling frequency must satisfy the inequality $f_s>2f_{max}$"

Fig 6 shows the effect of sampling a 50KHz sin wave, at the Nyquist frequency of 100KHz, and below the Nyquist frequency at 60KHz.

Fig 6. Aliasing Effects



The trace sampled at the Nyquist frequency appears to be a triangle wave at the same frequency as the original sin wave. However the trace obtained by sampling below the Nyquist frequency is misinterpreted as a triangle wave at a much lower frequency, this is caused by aliasing. In most DSO applications to correctly identify the shape of the waveform it has to be sampled around 10 times faster than the Nyquist frequency.

DSO's are often used to record complex waveforms such as video signals. These in theory, contain components of infinite frequency due to square sync pulses, suggesting that to sample them correctly you would need to sample at an infinitely high frequency. Obviously this is not possible. In practice simple signals, such as square waves can be captured relatively accurately by sampling at 10 times the fundamental frequency. Therefore complex waveforms like video signals should be sampled at roughly 10 times the frequency of the fastest event.

Most faults in digital systems are the result of signal glitches, which can be the result of cross talk (see fig 26). Unfortunately these glitches are usually only a few nano seconds in duration. A DSO which samples a waveform at intervals greater than the length of the glitch is unlikely to detect them.

Obviously the faster a DSO can sample a waveform the better the representation of the incoming signal, and the more complex the waveform that the DSO can capture. As technology and speed advances, faster and faster DSO's are required to debug the next generation of equipment.

2.5) Storage Depth

The length of a recording by a DSO is determined by the sample speed and the storage depth available and is as follows:-

$$\text{Length of Recording (Seconds)} = \frac{\text{Storage Depth (Number of Samples)}}{\text{Sample Frequency (Hz)}}$$

As discussed in section 2.4 it is advantageous to use a high sample rate. The higher the sample rate the shorter the length of the recording for any given storage depth. Engineers often need to make long recordings at a high sample rate, to stop aliasing. The solution is to provide a large storage depth, enabling the user to make long, high frequency recordings. Unfortunately most DSO's and logic analysers have only enough memory for a few thousand samples (and in some cases as low as a few hundred), leading to recording lengths as small as a few micro seconds.

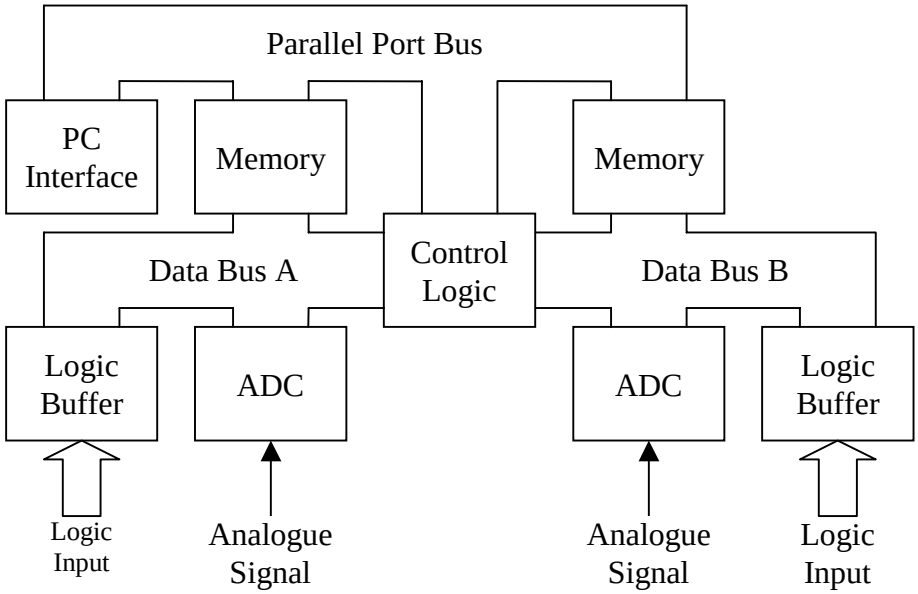
Although advanced trigger modes like Pre-Trigger and Delayed Trigger make better use of the available storage the user is still left examining fragments instead of the whole picture.

3) Design

3.1) Initial Concept

Fig 7 shows a simplified block diagram of the project DSO. (There are additional control signals from the control logic to all parts of the system, which have been left out for the sake of clarity.)

Fig 7. System Block Diagram

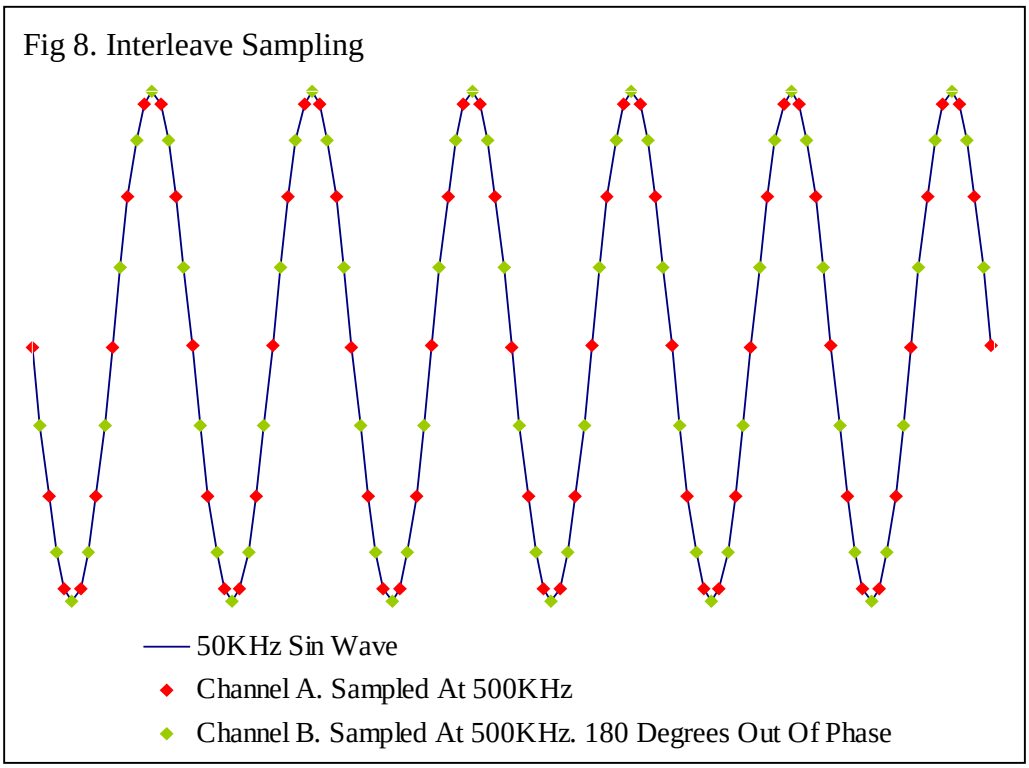


The principle of operation is relatively simple. The ADC and the logic buffer both share the same tri-state bus. The control logic selects which one of the two outputs to the bus, and therefore whether the channel is analogue or digital. The bus is then fed into the memory so that the data can be recorded. The control logic additionally monitors the data to watch for a trigger point. This approach eliminates additional circuitry that would normally be required for triggering and shortens the analogue signal path, thereby reduceing noise susceptibility. This also provides the ability to trigger off the digital signal when in logic analyser mode.

In addition to triggering, the control logic also performs all “Glue Logic” functions and advanced modes such as pre-trigger. When the memories have been filled the recording is stopped and the data is transferred to the PC via the parallel port.

Because the two channels are completely separate from each other they are able to act independently. For example channel A could be in logic analyser mode while channel B is in analogue mode. By connecting the two analogue inputs together and phase shifting the second channel by 180° it is possible to double the effective sample rate (this can be seen in fig 8). After the data is downloaded into the PC it is interleaved by the software and the waveform reconstructed. Even though the signal is sampled at twice the normal frequency the length of the recording is not reduced, as both memories are being used to store a single waveform.

Fig 8. Interleave Sampling



3.2) Design Overview

3.2.1) Memory

From the DSO specification the following requirements for the memory can be derived:-

8 Bit wide data bus

3Mbit Storage capacity

40Mbytes/Second Data Rate

Separate “read” and “write” ports

Although separate “read” and “write” ports are not required this would greatly simplify the design. The Hitachi Frame Memory as used in the last DSO (By Philip Cupitt) meets these requirements, however it is expensive and very difficult to obtain. For this reason it was decided to look for an alternative. Frame memories are mainly used in video applications, such as time base correction and picture in picture. They are ideally suited to this sort of application due to their large storage capacity, fast data rates, and separate read and write ports. In addition they are usually highly integrated, which simplifies the circuit design.

After looking at available devices the Averlogic AL422 was chosen. Like most frame memories it is based on DRAM technology. DRAM requires special controllers and provides relatively low data rates. The AL422 avoids these pitfalls by being heavily integrated. Although the external data bus is only 8 bits wide most frame memories use much wider internal bus widths. This increases the effective speed due to parallelism. High speed logic is used to split the internal bus into 8 bit segments which are then placed on to the output data bus. In addition to this the DRAM controller and the address generation logic are embedded inside the AL422.

The specific memory location that data is written to is determined by a write pointer. The value of this pointer can either be incremented or reset to zero (the start of the memory). However it can’t be moved to a specific location. As a result all write operations have to be sequential. Frame memories behave like circular buffers in that after the write pointer reaches the end of the memory it automatically resets to the

beginning and overwrites the existing data. Read operations use a similar read pointer. Because of this frame memories are often called First In First Out Buffers (FIFO's).

The AL422 uses 3 pins to control write operations. WCLK, /WE, and /WRST. Data is clocked into the memory on the rising edge of the write clock (WCLK), unless the write enable line (/WE) is high, in which case the write pointer is not incremented and the data in not captured. By pulling the write reset pin (/WRST) low and pulsing the write clock (WCLK) the write pointer is reset to the start of the memory.

Read operations are controlled by 4 pins, RCLK, /RE, /RRST and /OE. By pulsing the read clock (RCLK) the read pointer is incremented unless the read enable pin (/RE) is high, causing the read pointer to stop, and new data is not applied to the output. The read pointer can be reset to the start of the memory by holding the read reset pin (/RRST) low and pulsing RCLK. The data outputs can be tri-stated by holding the output enable pin (/OE) low, /OE is fetched on the rising edge of RCLK.

Although handled internally, the DRAM refresh is derived from RCLK or WCLK (whichever is the faster), because of this in order to maintain data integrity at least one of these signals MUST be kept running faster than 1MHz.

3.2.2) Analogue To Digital Converter

Analogue to Digital Converters (ADC's) are used to sample an analogue waveform and produce a digital representation. Because of the discrete nature of a digital signal the incoming analogue waveform is quantised to the nearest digital equivalent. The greater the resolution of the ADC the smaller the quantisation error that is introduced. For a DSO an 8bit resolution, which provides 256 levels is more than adequate.

Because of the noise problems with the previous DSO (by Philip Cupitt) it was decided to change from the Philips TDA8703 ADC to the Intersil HI5667 ADC. As well as having a higher maximum sample rate, the HI5667 allows for separate analogue and digital power supplies and differential inputs.

Most high speed ADC's utilise a Flash Conversion method, which uses a parallel array of 2^n-1 comparators (where n is the resolution of the ADC in bits). This means that an 8bit converter would require 255 comparators. This level of complexity not only increases the cost but also limits the maximum speed at which the converter can operate. The HI5667 uses a hybrid pipelined Flash Conversion method, where the conversion is split up into 7 stages with each stage using a 2bit flash converter. This reduces the number of comparators required to 28. Although this introduces a 7 clock cycle latency between the analogue signal being sampled and the data appearing on the outputs. This is acceptable for this application.

3.2.3) Buffer Amplifier

When the DSO is operating in interleave mode the analogue inputs to the two ADC's must be connected together. In addition to this the incoming analogue signal needs buffering. Both of these functions can be achieved with a video op-amp such as the Elantec EL4332C. This chip has a maximum bandwidth of 300MHz and contains 3 matched op-amps with multiplexed inputs. This makes it ideal for this application because it can perform both the buffering and the input switching. A single logic signal is used as an input select for all three op-amps. Unfortunately like most high bandwidth amplifiers the gain is fixed (in this case at 2).

3.2.4) Control Logic

The previous DSO (by Philip Cupitt) used several Programmable Logic Devices (PLD's) and a PIC microcontroller to form the control logic. Because of the increased complexity of the new DSO it was decided to use an Altera Field Programmable Gate Array (FPGA). This reduces the chip count and hence increases the reliability. Since there is only one chip that requires programming it is possible to make the DSO In System Programmable (ISP).

FPGA’s consist of a large number of identical logic cells, each cell containing a flip flop and some basic combination logic. The cells are arranged in blocks with programmable interconnects between them. By configuring both the logic cells and the interconnects it is possible to construct complex logic systems. Because of the number of programmable connections in an FPGA it is impractical to configure the device manually. Software tools (such as Altera’s MaxPlusII software) allow the functionality of a design to be separated from it’s implementation.

The first stage (design) is to define the functional blocks and the connection between them, each block can then be split into sub sections, allowing complex system to be split up into small manageable blocks. Apart from simplifying the design this enables the user to verify and simulate each section of the system independently. There are three methods that can be used to define the behaviour of a block:-

VHDL

AHDL

Schematic Capture

VHDL and AHDL are both hardware description languages that enable the user to create a hardware design by simply writing a program. This is a very powerful method and can create easily scaleable designs. Schematic Capture allows the user to enter a design by drawing the circuit diagram. To simplify this process various ‘MegaFunctions’ are available which enable the user to quickly create complex blocks such as comparitors and synchronous counters by entering the required specification.

Once the design stage is completed it is compiled and the resulting information is passed onto the fitter, which maps the configuration onto the available logic cells in the target device. Because of the large number of different combinations, random fitting techniques are used to reduce the time required, this can result in the same design having different implementations each time it is compiled. However the user does have the option to assign parts of the design to specific hardware resources, for example an output can be assigned to a specified device pin, eliminating the need to redesign the PCB every time the design is recompiled.

The final stage in this process is to simulate the design and the implementation. Because the propagation delay of a circuit is dependant on the target device and how the design is mapped onto it, it is impossible to calculate the performance of the system manually. Instead a series of test input patterns are fed into the simulator, which then generates the resulting output patterns. Because the simulator takes into account the propagation delays of both the logic cells and the interconnects between them the performance of the device can be calculated. The simulator is also a valuable tool for design verification and was used extensively throughout the development of the DSO.

The Altera chosen is EPF8282A-84, which is a member of the FLEX 8K range. It is an 84 pin PLCC chip with 68 usable I/O lines. This is more than adequate for this application and has the advantage that the chip can be socketed for easy development. Because of the nature of FPGA's the speed at which they can operate at is dependent on the design that they are implementing. Although the chip chosen has the slowest speed grade in the range, it is still capable of running large counters at over 80MHz. As all the logic that the FPGA will implement runs at 50MHz or lower this will suffice.

As the whole of the Altera FLEX range is based on SRAM technology, an external non-volatile memory is required to program the Altera from when the system powers up. Although this increases the chip count it eliminates the need for specialist programming hardware that non-volatile FPGA's often require. The cost of the external memory is offset by the reduced cost of FPGA compared to their non-volatile counterparts.

3.2.5) Software

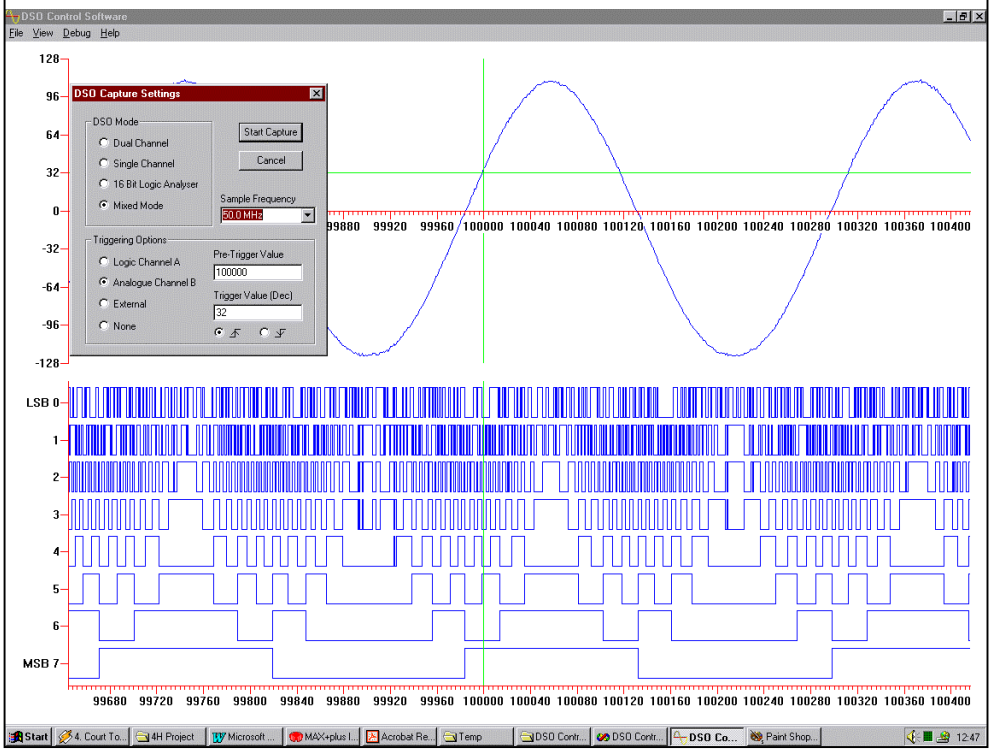
Because of the radically different approach of this new design it was not practical to adapt the software from the previous DSO. Although this meant completely re-writing the software it also provided the opportunity to produce a fully functional 32Bit Windows (Win32) application. This has the advantage that it uses the standard Windows user interface and is therefore relatively intuitive. The resolution that the

software renders the traces in is only dependant on the size of the window. As this can be resized the resolution that traces are displayed in is only limited by the computer being used, and not the software. To make examining the traces easier, the user can zoom in on any section of the waveform or pan left or right as needed.

Although the software is a Win32 application it uses direct hardware I/O in order to talk to the DSO. This is allowed under Windows 95-98 but not under Windows NT, because WinNT classifies “outp” and “inp” as privileged opt-codes. The next version of the software should use a device driver to communicate with the hardware. This would allow the DSO to be used under all versions of Windows.

All the hardware options like sample rate and trigger options are selected from a capture dialog box, which can be seen in the screen shot below (Fig 9).

Fig 9. DSO Software Screen Shot



Captured waveforms can be saved to disk in the software’s own custom file format. This has the advantage of preserving additional information like sample rate and trigger conditions. The file can be reloaded into the software at a later date for further examination. The DSO software can also export to a standard Comma Separated Variable (CSV) file for use with a spreadsheet, however most spreadsheets

including Microsoft Excel are not capable of handling the large volume of data generated, and are therefore limited to displaying a small portion of the data captured.

The software was written in ‘C’ using Visual Studio. The Win32 interface uses direct API calls for efficiency. Because of the large amount of code (approximately 3200 lines) a modular approach was adopted where related functions are grouped together. This results in several independent source files each handling a different part of the program. Each source file has a corresponding header file that contains all the necessary declarations so that functions in the source file can be called from other parts of the program. The following source files make up the software:-

Main.c
Gui.c
Gui.rc
Plot_Data.c
DSO_IO.c
P_Port_IO.c
Numeric.c
File_IO.c
Error.c

3.2.5.1) Main.c

This file contains the WinMain() function, which is called by Windows to start execution of the program. The first task that this function performs is to set up the global variable hInstance, which is the instance handler (this is used by windows to keep track of multiple instances of an application). The next step is to initialise the data storage area and preferences variables. Once this is complete the Show_Gui() function in Gui.c is called, which launches the user interface and starts the message loop.

Main.c also contains a global Quit() function, which can be called from any point in the program during execution. This function closes all open files, stops the message loop and exits.

3.2.5.2) Gui.c And Gui.rc

Graphical user interfaces (GUI's) under Win32 are based on a messaging system. When a user performs an action the operating system generates the corresponding message. For example if the user ticks a check box the application is sent a BST_CHECKED message. These messages are queued until the application can process them. Each application has a message loop that checks the queue for pending messages. Every window or dialogue box has a message handler, when the message loop finds a message in the queue it is dispatched to the appropriate message handler.

Since almost every event generates a message, the way the handler processes these messages determines the way the user interface behaves. This can be as simple as closing a dialogue box when the cancel button is pressed or as complex as drawing an object using the mouse. In addition to responding to user generated events the messaging system is used to trigger a re-draw of an applications window whenever the display area is invalidated, for example when a window is re-sized.

The Show_Gui() is the only global function in the Gui.c file, It initialises the main window of the program and loads the accelerator table, which is used to translate keyboard shortcuts into the corresponding message. After this is completed the message loop is started. It is worth noting that at this point the main window is still blank, however the user will not see this as Windows immediately sends a WM_PAINT message that triggers a redraw of the display. Because the graph plotting routines are very large they have been moved into a separate file called Plot_Data.c.

I/O operations with the hardware can take several seconds and during this time the control software will be unable to process any messages that enter the queue. This will cause the program to stop responding and it will appear to have crashed. To solve this problem a separate thread is spawned. This effectively means that both the message loop and the hardware I/O will occur in parallel. This also enables the I/O thread to send progress messages to the GUI thread which are then displayed to the user.

The Gui.rc file contains templates for the dialogue boxes, the accelerator table used for keyboard shortcuts, application icons, and several small pictures used in the GUI.

3.2.5.3) Plot_Data.c

Plot_Data.c contains all the functions required to plot graphs of the captured data. The Show_Gui() examines the Scope_Data structure to determine what mode the DSO was in when the traces were captured, and therefore how the data should be plotted.

The first step is to determine the size of the window. Once this information has been retrieved with an Application Programmers Interface (API) call the size of the graphs can be calculated. The usual approach for plotting a graph is to calculate the position of the points and then connect them with straight lines. Although this is effective if the user is viewing a small number of data points it can be extremely slow when the entire data set is viewed. To solve this problem the program uses a different plotting algorithm when large amounts of data are being displayed. This algorithm draws a series of vertical lines one pixel wide to represent the range data point within the single pixel width. Unfortunately this method is not usable when a very small range is being displayed. Therefore if the user zooms in or out the program automatically switches between the two plotting modes.

After the data has been plotted the display range and sample frequency are used to calculate the scale of the axes. The size of the window is then used to calculate how many axis labels can be drawn. A green vertical line is drawn to indicate the location of the trigger point when pre-trigger is used, similarly a horizontal line is used to represent the trigger level.

3.2.5.4) DSO_IO.c

When the user starts a new recording the program spawns off a new thread. This thread starts execution with the Capture_Thread_Func() function, which calls several local functions to initialise all the hardware registers to the correct values corresponding to the settings the user requested. When the recording is complete the download is started. During this process a series of progress messages are sent to the GUI for display.

3.2.5.5) P_Port_IO.c

P_Port_IO.c provides all the hardware I/O routines required to read and write to the parallel port registers in addition to some initialisation functions. By separating these functions from the DSO_IO.c file migration to a device driver model is greatly simplified. To enable the software to be developed under Windows NT this module is able to emulate the parallel port registers there by avoiding direct hardware I/O, this option is selected by setting a compile time parameter.

3.2.5.6) Numeric.c, File_IO.c, And Error.c

File_IO.c contains several functions that are used to save and load the captured data together with the settings that were used during capture, As all of this information is contained within a single structure this can be done by simply reading and writing the entire structure to disk. This file also contains the export function, which saves the data as a Comma Separated Variable (CSV) ASCII file. This can then be read by any standard spreadsheet. The format of the CSV file depends on the mode of the DSO, each channel is either represented by a single column of signed integers or by 8 columns of single binary digits when the channel is in logic analyser mode.

Error handling is provided by the Error() function in Error.c. Each different type of error is assigned an error number. In the event that an error occurs this number is passed to the Error() function, which translates the number into an error message that is then displayed to the user. Depending on the severity of the error the program either continues running, terminates with cleanup, or terminates without cleanup. This enables a single function call to handle everything from simple user errors to critical failures.

Numeric.c contains several common arithmetic functions that are too large to be inserted directly into the main body of the code. This file would be the ideal place to implement a Fast Fourier Transform (FFT) in the future. This would enable the system to be used as a spectrum analyser.

3.2.6) PC Interface

There are several options for interfacing the DSO to the PC:-

Serial Port

Parallel Port

USB

ISA

PCI

Because of the relatively large amount of data that is stored in the DSO the serial port was discounted because the time required to download the data to the PC would be unacceptable long. USB is not practical because of it's complexity. In addition to this not all PC's have a USB port. Although fast, PCI and ISA both require internal connection to the PC and therefore would be difficult to install. The parallel port was chosen because it has a relatively high data transfer rate, is easy to develop device for, is user friendly and is usually present on laptops, enabling the DSO to be used in the field

Unfortunately there are several parallel port operating modes, some of which are not available on old machines. For simplicity and compatibility it was decided to use the standard bi-directional mode, which has been available from the days of the 386. This allows byte wide transfer in either direction and has 5 additional input lines (Status lines) and 4 output lines (Control lines). Normally the parallel port resides at 0x378 in the PC's I/O space, although this is not always the case. The DSO software allows the user to select the appropriate address for the particular parallel port. As there are only 3 addresses that can be used this is relatively easy. This is especially useful if the PC being used has multiple ports.

3.3) Detailed Design

The circuit diagram for the system can be seen in Fig 10.

3.3.1) FPGA Configuration

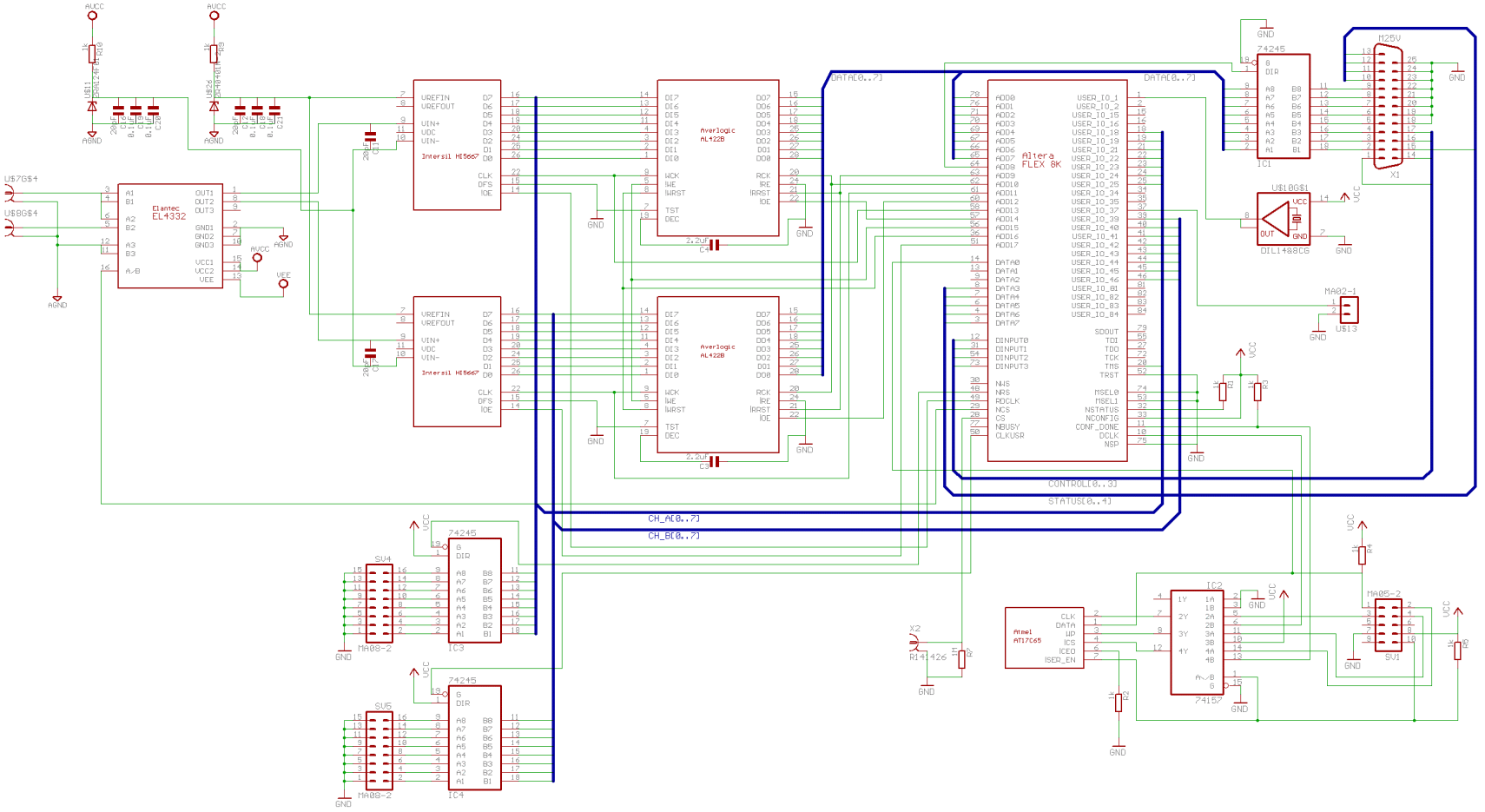
Because the FPGA is SRAM based it has to be programmed every time the system is powered up. There are several methods of accomplishing this, the most suitable for this application is the Active Serial method. This involves connecting a serial EEPROM (E^2) to the FPGA, and has the advantage that only one user I/O pin is required. The remaining 2 connections to the E^2 are dedicated configuration pins. Additional configuration pins are tied either high or low to indicate which configuration scheme is being used.

By using a multiplexor the E^2 can either be connected to the FPGA or to a programming header on the board. It is therefore possible to re-program the memory chip while it is still in the circuit. The select input to the multiplexor is pulled high by a 1K resistor and connected to the ISP header. When the programmer is plugged into the header this line is shorted to GND, this switches the E^2 from being connected to the Altera to being connected to the programmer. After programming is complete the user only has to cycle the power to the DSO to cause the Altera to read the new configuration data from the memory chip.

3.3.2) Pre-Trigger Mode

As discussed earlier the ability to record data from before the trigger point can be very useful when debugging systems. This is accomplished by making the DSO continuously record the incoming signal, terminating the recording process when a trigger pulse is detected. This leaves the memory containing data from before the trigger point. By controlling the time between receiving the trigger pulse and stopping the recording it is possible to position the trigger point anywhere in the recording.

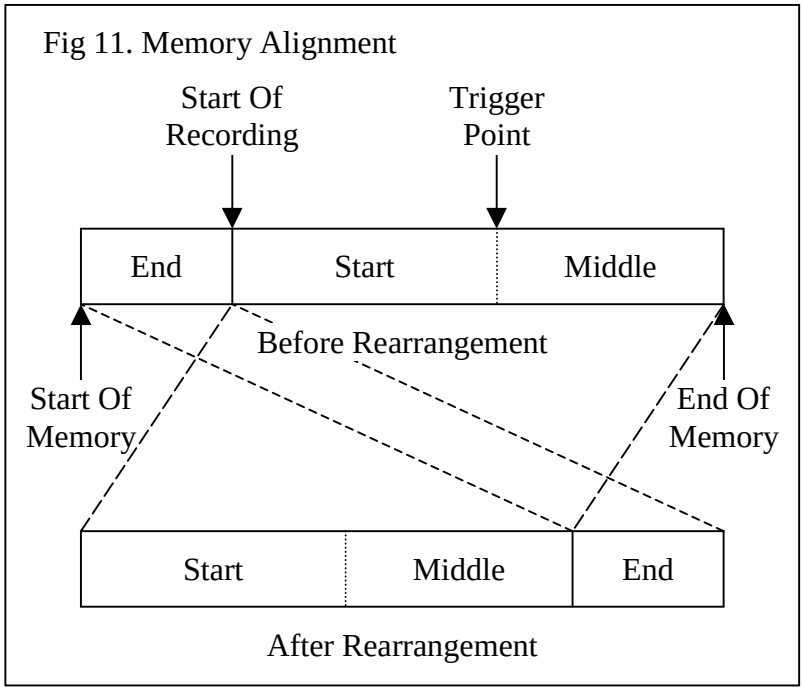
Fig 10. DSO Circuit Diagram



This is implemented with a 19 bit counter that is clocked from the same signal as the FIFO's, so that when the value in the counter is equal to the size of the memory the recording is stopped. The amount of pre-trigger required can be set by pre-loading this counter with a value between 0 (no pre-trigger) and 393216 (the size of the memory and therefore max pre-trigger). Fortunately 393216 is 0x60000 in HEX, this means that the logic only has to monitor the top two bits of the counter to know when to stop the recording.

Unfortunately this approach means that the start of the recording can be anywhere in the memory. A second 19 bit counter is used to identify the start of the recording in memory. The counter is reset at the same time as the write pointer on the FIFO's and is run from the same clock signal, so that the value stored in the counter is equal to the location of the write pointer and hence the start point of the recording. Before the data is transferred to the PC the value of this counter is read and used to re-align the captured data, as seen in fig 11.

Fig 11. Memory Alignment



The same technique of monitoring only the two most significant bits is used to reset the counter when the value equals the size of the memory. Because the counter includes zero it will count to 0x60001, therefore every time the counter resets it gets out of sync with write pointer by 1 (this is a cumulative effect). To prevent this the counter is reset to 1 instead of 0. This trick greatly reduces the complexity of the counter control logic.

Both of these counters and their associated logic are implemented inside the Altera FPGA.

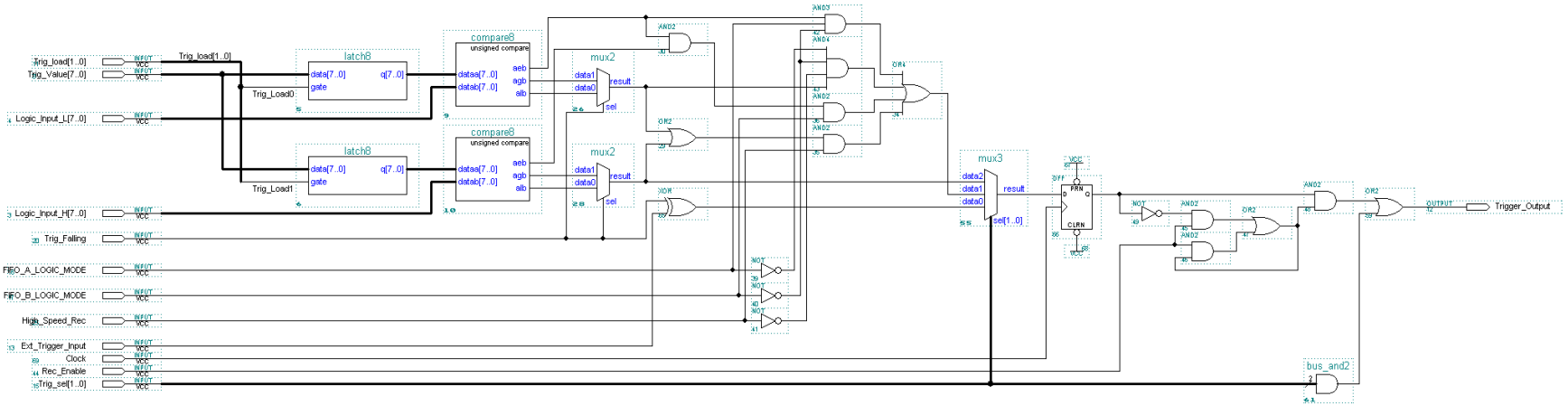
3.3.3) Trigger Detection Unit

The trigger unit, which is implemented inside the FPGA, monitors both of the channels for a valid trigger point (See fig 12). Comparitors determine if the incoming digital word is greater than, equal to, or less than the values stored in each register. By changing the values in these registers the user can alter the trigger level. The comparitors outputs are then fed into the trigger selection logic (See table 1), which forwards the correct trigger signal to the next stage.

Table 1. Trigger Selection Options

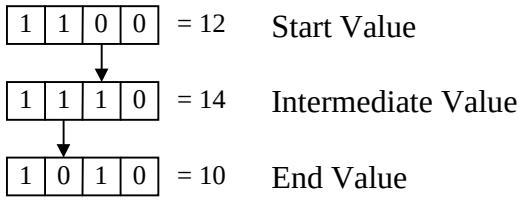
Trigger Selection Inputs					Trigger Condition	Notes
Trig Sel	Trig Falling	FIFO A Logic Mode	FIFO B Logic Mode	High Speed Rec		
0	0	X	X	X	Ext Trig = 1	External Triggering
0	1	X	X	X	Ext Trig = 0	
1	X	1	1	0	Channel A AND Channel B = Trig Value	Logic Triggering
1	X	1	0	0	Channel A = Trig Value	
1	0	0	0	1	Channel A OR Channel B > Trig Value	Analogue Triggering
1	1	0	0	1	Channel A OR Channel B < Trig Value	
1	0	0	0	0	Channel A > Trig Value	
1	1	0	0	0	Channel A < Trig Value	
2	0	X	X	X	Channel B > Trig Value	
2	1	X	X	X	Channel B < Trig Value	
3	X	X	X	X	Immediate Trigger	

Fig 12. Trigger Unit Circuit Diagram



Since neither the outputs from the ADC nor the outputs logic buffer chips are likely to change at exactly the same time, intermediate values can appearing on the bus. For example a transition from 12 to 10 could cause the value 14 to appear in the bus, as illustrated in fig 13. This can result in a miss triggering. The solution is to clock the data synchronously so that the output from the trigger selection unit is only sampled when the data on the bus is stable. This is done by placing a D-Type flip flop which is run from the system clock on the output of the trigger circuit.

Fig 13. Intermediate Bus Values



The final stage in the trigger detection unit converts the 'level triggering' behaviour produced by the first stage into 'edge triggering'. This is accomplished by only forwarding a 1 if the trigger line has first been 0, so that the output only goes high when there is a transition from low to high and hence an edge. By settings the Trig_Sel bus to 0x3 the user can disable triggering and start the recording immediately.

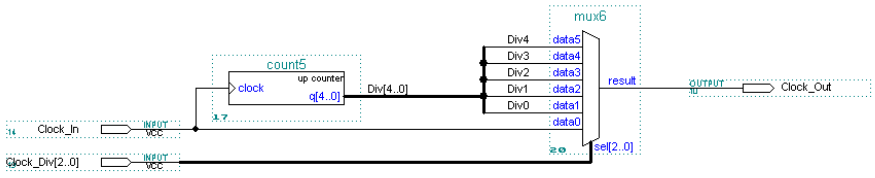
3.3.4) Clock Generation

It is useful to be able to change the sample frequency, enabling the user to trade off recording length against sample frequency. In this system the primary clock is provided by a 50MHz TTL clock generator, which is based on a quartz crystal. This is fed into the FPGA where it undergoes a programmable divide. This is achieved by feeding the clock signal into a 5 bit counter, which produces signals at $^1/_2$, $^1/_4$, $^1/_8$, $^1/_{16}$, and $^1/_{32}$ times the original frequency. A software selectable multiplexor is then used to feed the correct clock signal to the rest of the DSO, as can be seen in fig 14.

As mentioned earlier it is possible to double the effective sample rate by combining the two channels and phase shifting the second by 180°. This is done by

sending the clock signal for the second channel through a XOR gate, This programmable invert is used to provide the phase shift.

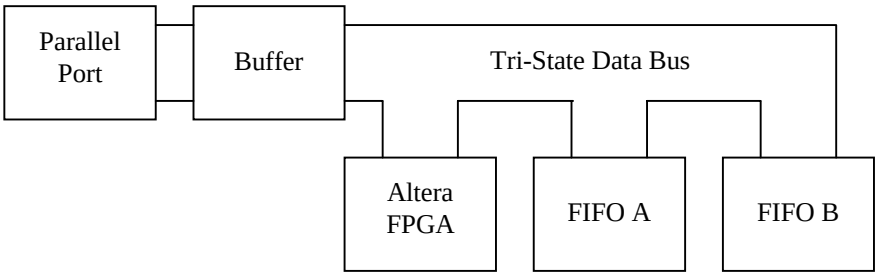
Fig 14. Clock Divider



3.3.5) Tri-State Buses

Several devices including the FPGA and both FIFO's need to be able to transfer data to the PC. This is accomplished by using a tri-state bus, as shown in fig 15. Because the FPGA controls the direction pin on the buffer chip and also the output enable (/OE) pins on the FIFO's it acts as a bus arbiter and eliminates contentions. The buffer is a standard 74LS245, which because of it's low drive capability compared to a parallel port, is incapable of causing damage to the PC in the event of a parallel port bus contention. As this buffer chip is socketed it can easily be replaced if it is damaged.

Fig 15. Data Output Bus



The tri-state buses that the ADCs and the logic buffers connect to are very similar in operation to the parallel port bus, in this case only the ADC and the logic buffer output to the bus and both the FPGA and the FIFO monitor the data on the bus. (Again the /OE lines are controlled by the Altera to avoid bus contentions.)

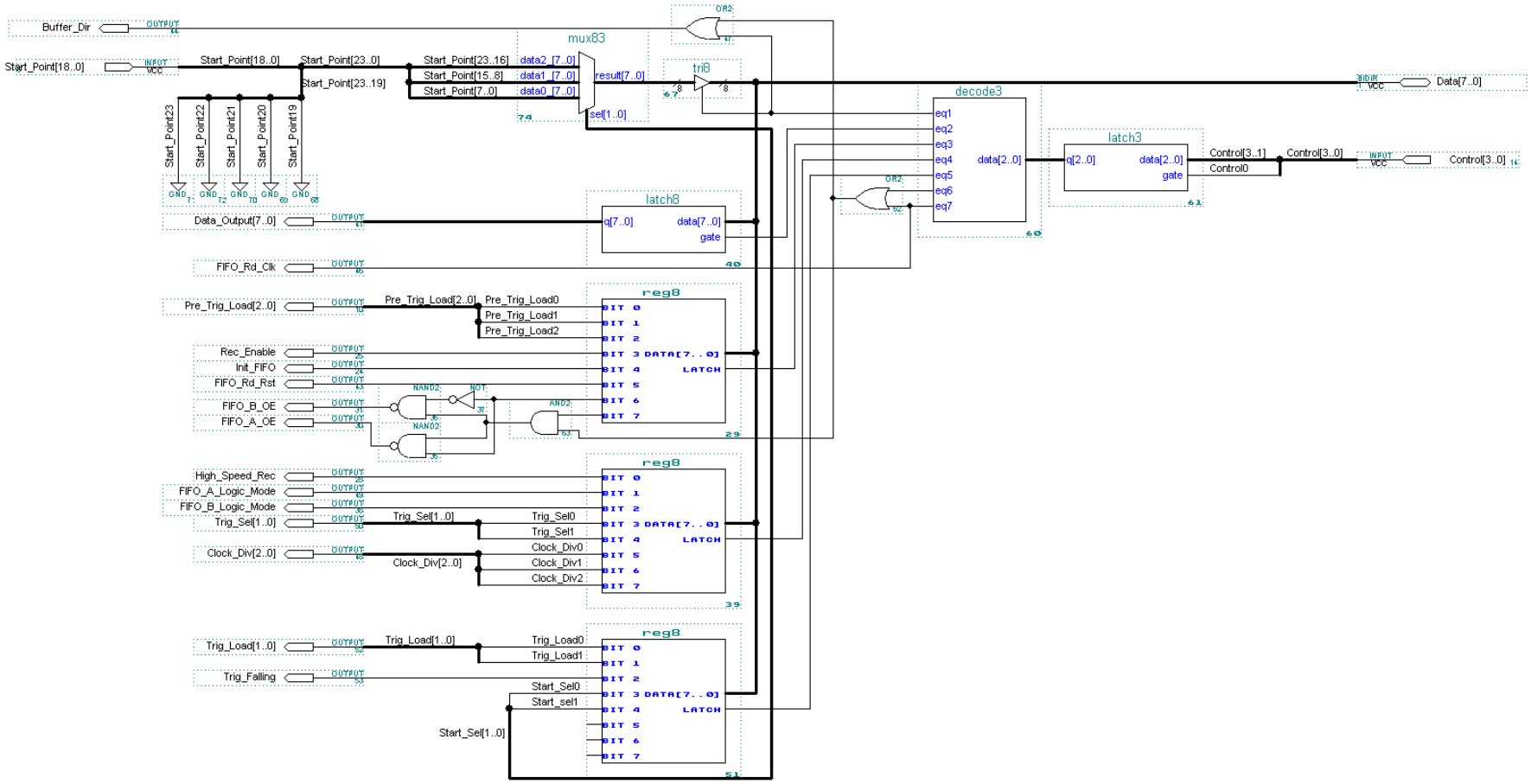
3.3.6) Parallel Port Interface Logic

The control logic requires many control lines to set clock divide, trigger points, etc. These are generated by the parallel port interface logic (See fig 16). The top 4 bits of the parallel port control bus are used as a data bus. Data on this bus is latched into the controller by pulsing the remaining control lines, this approach prevents signal glitches on the parallel port from causing abnormal behaviour in the DSO. The output of this latch is then fed into a decoder to increase the number of available lines. Some of these lines are used to latch data from the main parallel port data bus into one of four, 8 bit registers. This provides all the control lines required. The remaining lines from the decoder are used for the FIFO RCLK and to enable the bus output from the FPGA.

3.3.7) ADC Operation

When the DSO is in single channel interleave mode both of the ADC's have to output the same digital value given the same analogue input. Any difference in digital output will result in noise, as alternate samples will have different values even if there is a constant analogue input. The analogue to digital conversion is dependent on 2 reference voltages (one for scale, the other for offset). It was decided to use external references instead of the ones that are built into the ADC's, this means that the same reference voltage could be fed into both converts and therefore minimise this noise.

Fig 16. Parallel Port Interface Unit



Although the ADC's can be differentially driven it was not possible to find a suitable op-amp with differential outputs and a high enough bandwidth. For this reason it was decided to use a single ended approach. when a 2.5 volt reference fed into the ADC the full scale input voltages is V_{DC} ±0.5 volts, where V_{DC} is a DC bias. To ensure correct operation of the ADC the input voltage must be in the following range:-

$$0.5 < V_{In} < 4.5$$

Therefore V_{DC} must be in the range:-

$$1 < V_{DC} < 4$$

In this DSO V_{DC} is provided by the second reference voltage generator and has a value of 1.25volts, giving an input voltage range of:-

$$0.75 < V_{In} < 1.75$$

Fortunately this is within the output range of the buffer amplifier. As this amplifier has a fixed gain of 2 the input voltage range of the DSO is:-

$$0.375 < V_{Input} < 0.875$$

By tying the Data Format Select (DFS) pin on the ADC's either high or low the data can be outputted as 'offset binary' or 'two's complement'. Setting the output format to 'offset binary' greatly reduces the complexity of the trigger logic and therefore the number of logic cells used in it's synthesis.

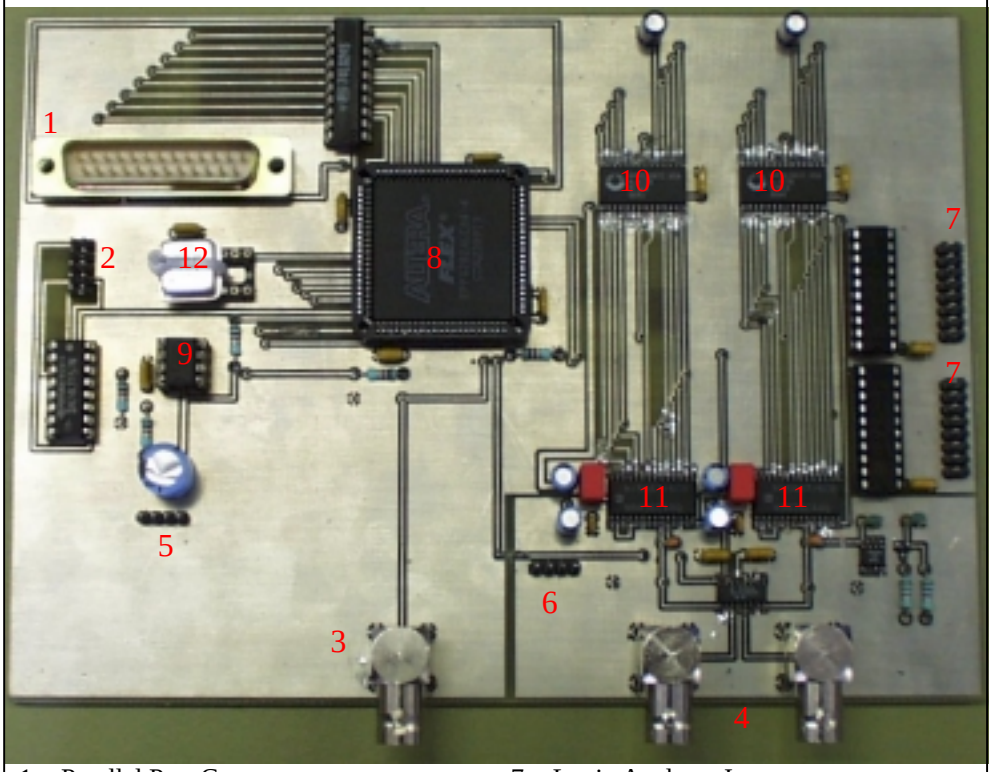
3.3.8) PCB And Power Supplies

To reduce the amount of noise that is present in the analogue signal paths it was decided to use separate analogue and digital power supplies. This creates a large amount of electrical isolation between the two parts of the DSO, as apart from a logic input to the op-amp, the only bridges between the two half's are the ADC's, which are specifically designed to operate from dual supplies. However for correct operation of the ADC's the analogue and digital grounds have to be within 0.3 volts of each other.

To achieve this with maximum noise immunity the two grounds are completely separate apart from a small link directly underneath one of the ADC's. The DSO requires a 5 volt digital power supply which can source up to 500mA, together with a ±5 volts supply for the analogue components which can source 200mA.

To further reduce noise and crosstalk in the system both sides of the PCB have ground planes, and suitable decoupling capacitors were placed both physically and electrically near all microchips. In addition to this both of the analogue inputs and the external trigger input use BNC connectors, so that screened cable can be used to connect directly to the DSO. The inputs to the logic analyser are split into two 8 bit busses and each bit is twisted with it's own ground wire to reduce crosstalk with adjacent wires. Fig 17 shows the final PCB.

Fig 17. Picture Of Project DSO



1 = Parallel Port Connector	7 = Logic Analyser Inputs
2 = ISP Connector	8 = Altera FPGA
3 = External Trigger Connector	9 = FPGA Configuration E^2
4 = Analogue Inputs	10 = Frame Memories
5 = Digital Power Connector	11 = Intersil ADC's
6 = Analogue Power Connector	12 = 50MHz TTL Clock Generator

4) Results

A Tektronix TDS-380 DSO was used as a reference when evaluating the performance of the project DSO. Several tests were performed in different operating modes and at different samples frequencies. All the graphs of waveforms from the project DSO were produced from screen shots of the DSO software.

4.1) Analogue Performance

The use of both triggering and pre-trigger modes is demonstrated in fig 18, which shows a 5KHz sin wave sampled at 1.6MHz. The green vertical line shows the location of the trigger point, where as the horizontal line shows the trigger level that was used. The two green lines intersect at a point on the waveform, indicating that the triggering system was functioning correctly.

Fig 18. 5KHz Sin Wave Sampled At 1.6MHz Using Project DSO

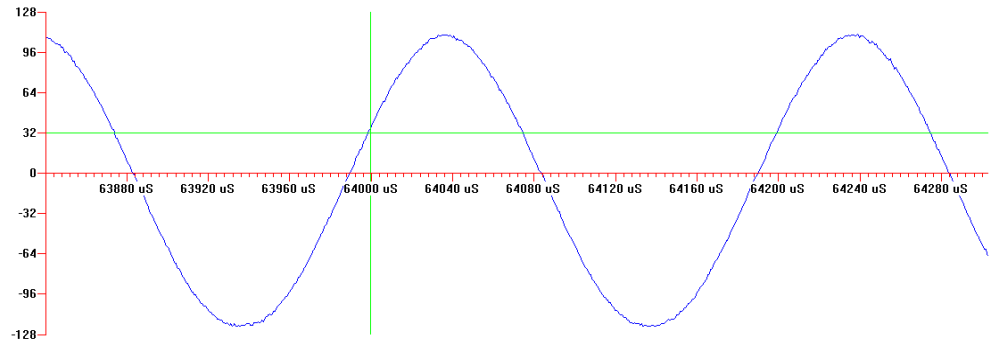
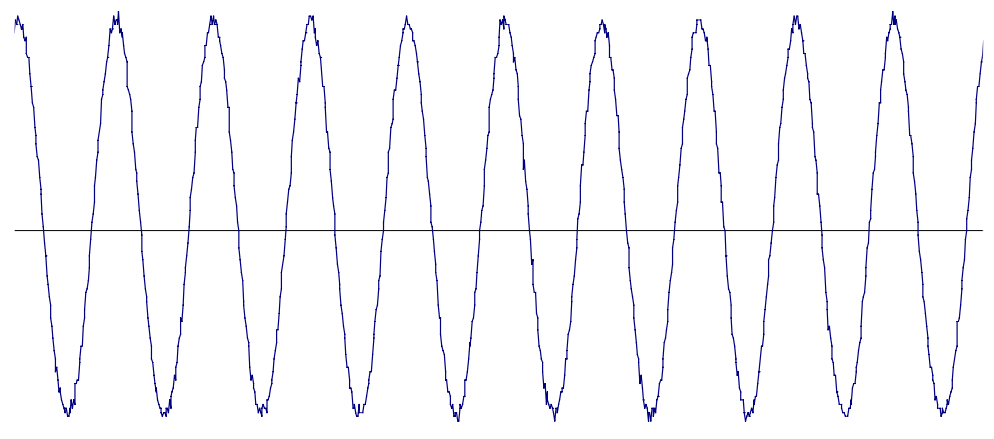


Fig 19. 5KHz Sin Wave Sampled At 500KHz With Reference DSO



For comparison the same signal was sampled using the reference DSO and the resulting trace can be seen in fig 19. There is a small amount of noise present in the trace obtained from the project DSO which is inevitable, however there is noticeably more noise present in the trace produced by the reference DSO. It is worth noting that the fig 19 represents the entire storage space of the commercial DSO, where as fig 18 is a 460µS portion of a 250mS recording.

To demonstrate the need for an increased sample depth amplitude modulated sin wave was captured, this can be seen in fig 20. To reproduce the signal correctly the sample frequency must be greater than the Nyquist frequency, which in this case is 2MHz, The waveform was sampled at 50MHz so that the shape of the carrier could be determined. Because of the limited storage depth on commercially available DSO's it is not possible to sample at a high enough frequency to avoid aliasing and still observe the modulation envelope.

Fig 20. Amplitude Modulation Of 1MHz Sin Wave, Sampled at 50MHz

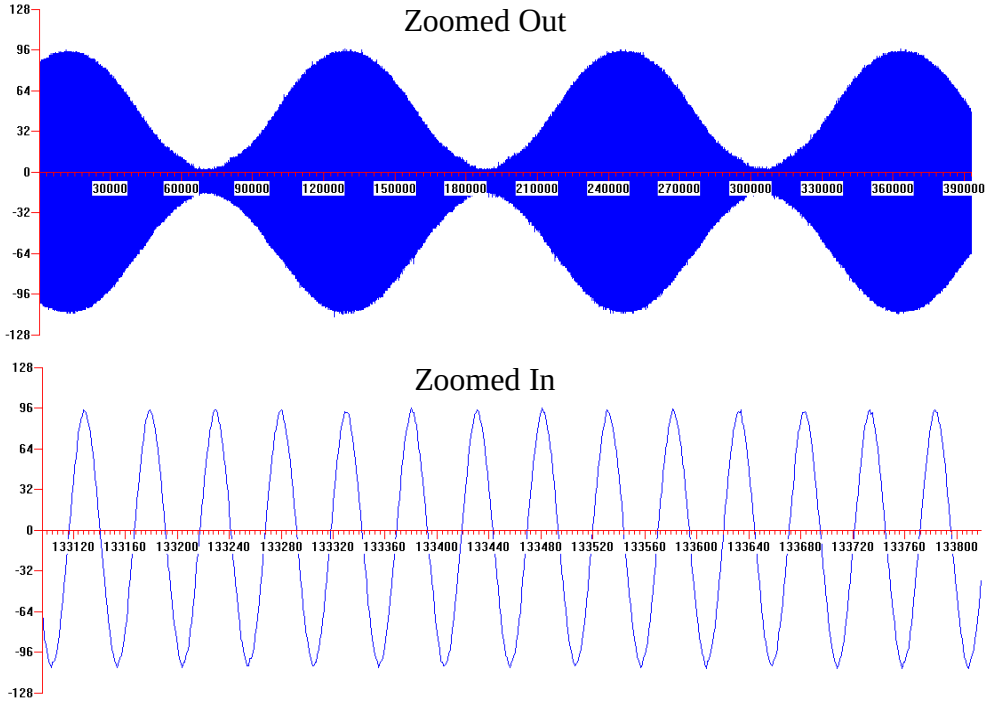
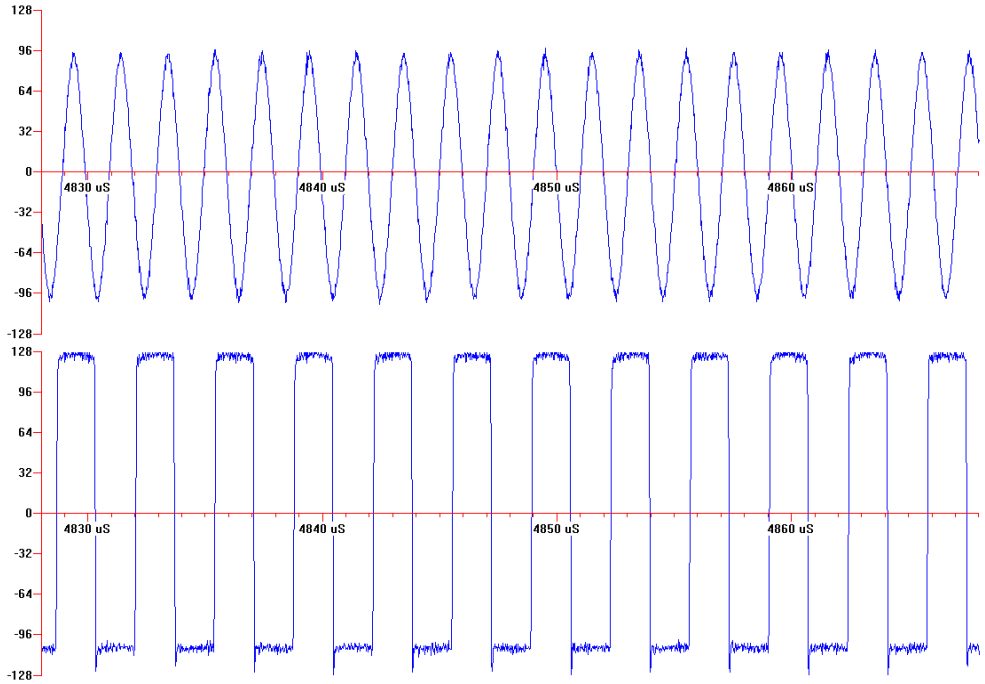


Fig 21 shows a 500KHz sin wave and 300KHz square wave sampled simultaneously at 50MHz. Unfortunately the amount of noise present in the waveform increased when using both channels. However the noise is within acceptable limits and still smaller than the noise present on the Reference DSO.

Fig 21. 500KHz Sin Wave & 300KHz Square Wave Sampled At 50MHz



A 500KHz sin wave was sampled at 100MHz to demonstrate the interleaving capability of the DSO, as shown in fig 22. This introduced a noticeable amount of noise into the captured signal. By looking at a zoomed in section of the trace it was clear that alternate samples were slightly offset, the result of two analogue channels not producing exactly the same digital value given the same analogue input. This could be the result of minor differences in the performance of the two ADC's or in the different channels of the buffer amplifier.

It is possible to re-align the data from the two channels. By averaging both channels the average DC value of each channel can be found. By shifting the whole of the second channel by the difference between two DC values, most of the noise created by the interleaving can be eliminated as seen in fig 23. This technique relies on the average DC values of both channels being the same, although this is not necessarily the case if the sample rate is close to the Nyquist frequency. In such a case

this method of re-aligning the data should not be used. As the offset between the two channels is relatively constant a low frequency signal could be used to calibrate the system before capturing data close to the Nyquist frequency.

Fig 22. 500KHz Sin Wave Sampled At 100MHz Using Interleaving

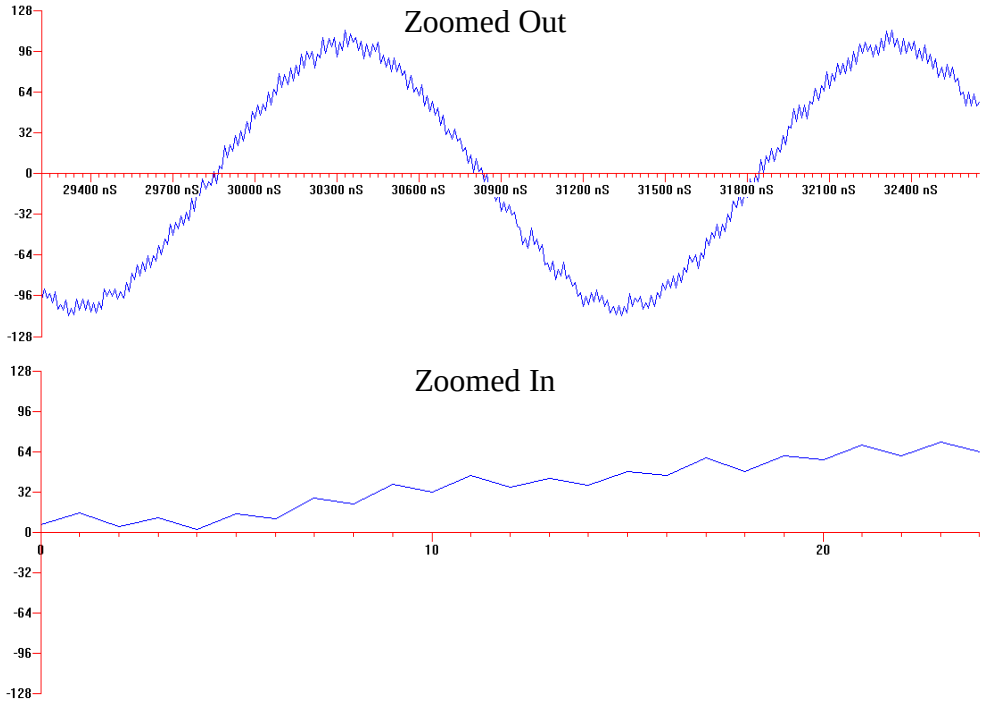
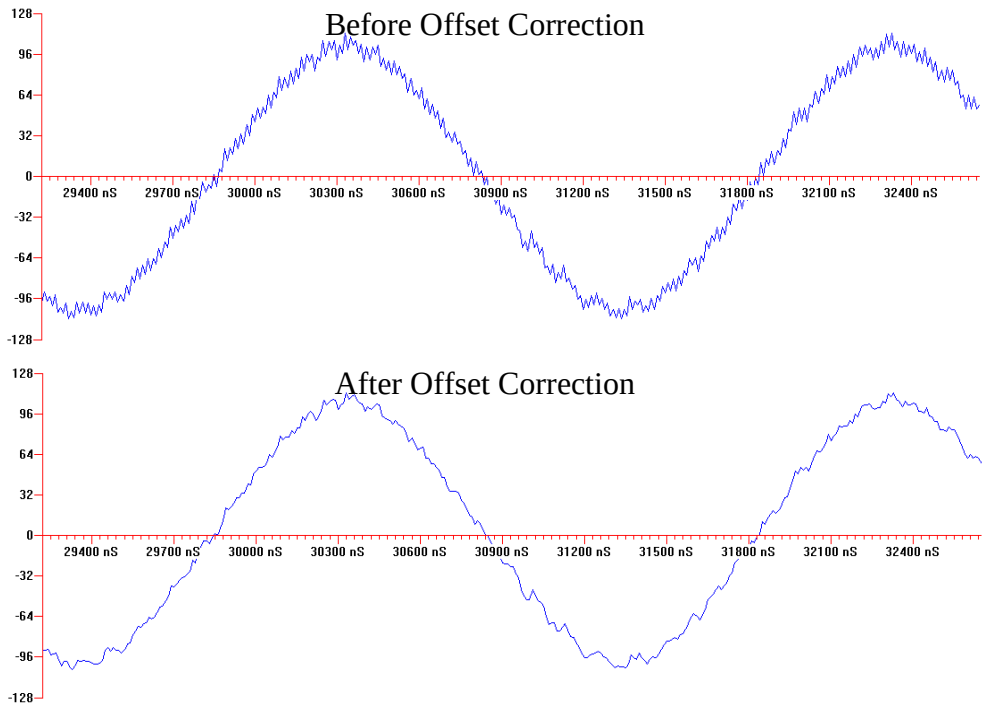


Fig 23. Affect Of Offset Correction On Interleave Sampling



4.2) Logic Analyser Performance

To test the performance of the logic analyser the 8 bit data bus of a parallel port was connected to the 8 most significant bits of the DSO, the remaining 8 bits were allowed to float high. This provided a quick and versatile method of generating complex digital signals. Fig 24 shows the parallel port data sampled at 50MHz. The DSO was set to trigger when the value 0xFF was present on the bus (as indicated by the green vertical line), a pre-trigger of 100,000 samples was also used. On closer inspection of the trace significant crosstalk was discovered, this can be seen in fig 25.

Fig 24. Logic Output From A PC Parallel Port Sampled At 50MHz

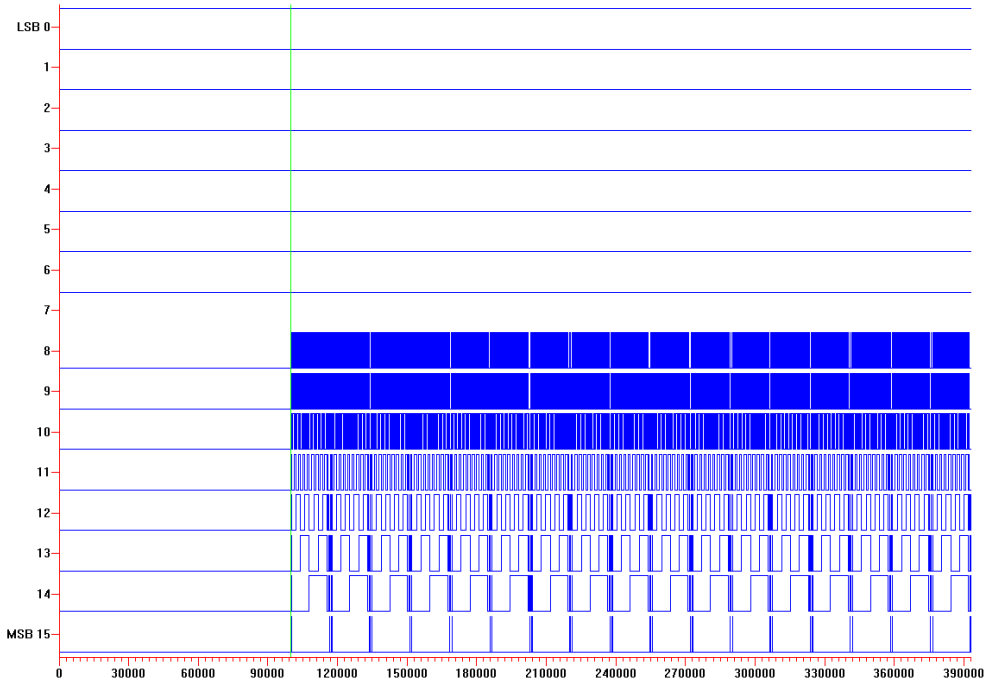
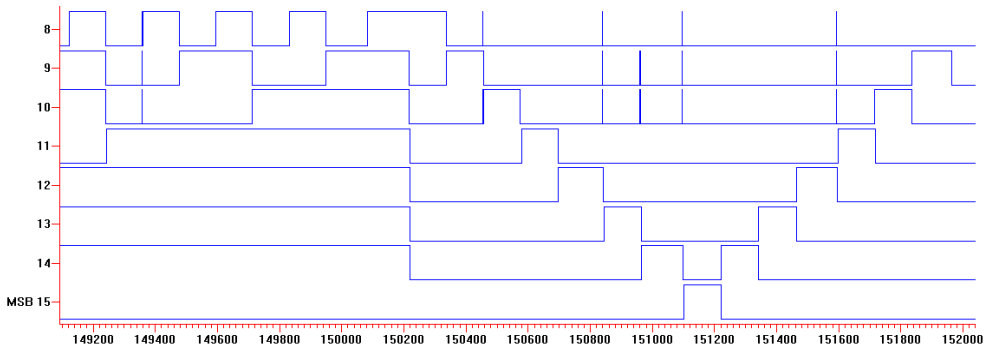
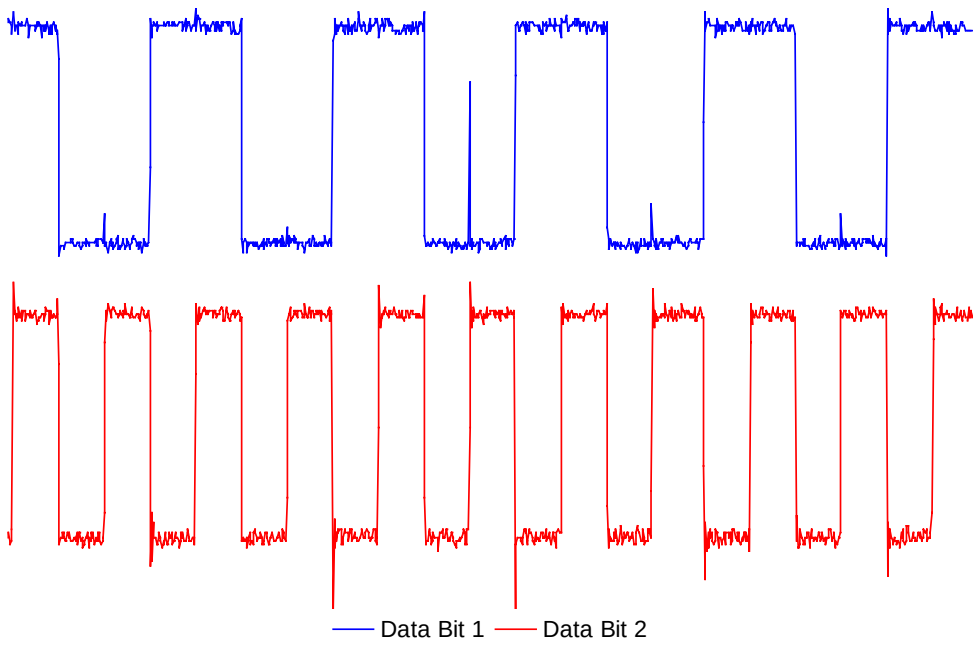


Fig 25. Crosstalk and Glitches Present In Parallel Port Data



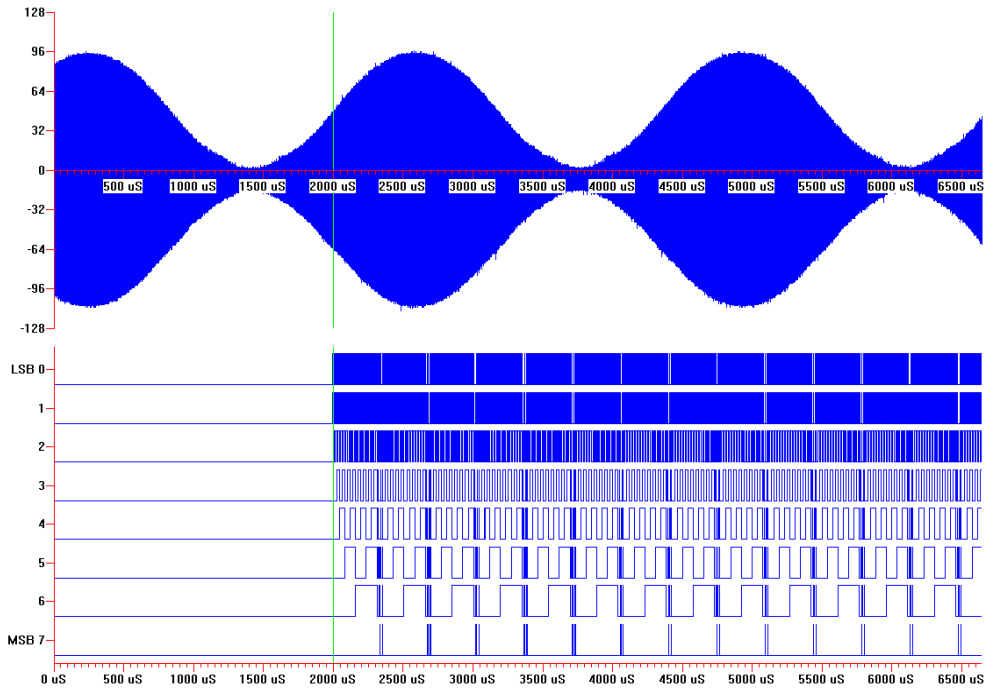
To investigate the source of the crosstalk the reference DSO was connected to the bottom 2 bits of the parallel port while the project DSO was disconnected, the resulting trace can be seen in fig 26. A considerable amount of crosstalk, undershoot and overshoot is clearly visible, as the project DSO was not connected at the time the trace was made, it could not be the cause of the glitches. After some further investigation the parallel port on the PC was found to be the source. Because the parallel port is synchronously clocked by the strobe pin, these glitches do not affect the operation of devices connected to the port. The fact that the project DSO detected these glitches proves it be a useful tool for debugging digital systems.

Fig 26. Crosstalk On Parallel Port, Captured Using Reference DSO



To demonstrate the combined analogue and logic operation of the DSO the data bus from the parallel port together with an amplitude modulated sin wave were captured at 50MHz. The DSO was triggered when 0xF was present on the parallel port, in addition to this the DSO was set to pre-trigger 100,000 samples. This can be seen in fig 27.

Fig 27. Combined Logic Analyser & Analogue Waveform



5) Discussion

The captured waveforms in section 4 show that the DSO has met the project specification laid down in section 1. In fact the system exceeds the specified maximum sample rate of 40MHz by 25% when operating in dual channel mode, and by 250% when operating in single channel mode. Table 2 shows the length of a recording for the minimum and maximum sample frequencies in each mode, along with the maximum signal frequency according to Nyquists sampling theory.

Table 2. Available Sample Frequencies And Recording Lengths

Mode	Max Sampling Frequency (MHz)	Max Signal Frequency (MHz)	Recording Length (mS)	Min Sampling Frequency (MHz)	Max Signal Frequency (KHz)	Recording Length (mS)
Single Channel	100	50	7.9	3.1	1563	252
Dual Channel	50	25	7.9	1.6	781	252

The DSO's large storage depth of 393 thousand samples, and high sample rate enables large waveforms to be captured in high detail. The system is capable of capturing nearly 8mS of a signal when sampled at 100MHz. In comparison a standard commercial DSO could capture roughly 10µS. This makes the project DSO invaluable for analysing video and other complex signals.

The only programmable component in the system is the serial E^2, which holds the configuration data for the Altera FPGA. Because this component is In System Programmable (ISP) by simply connecting a programming cable, it is easy for the end user to change the DSO hardware. This could be used to quickly and cost effectively produce bug fixes and additional features for the system. In addition to this the user could, for example re-design the triggering hardware, so that an engineer could target the DSO for a specific application or situation. This process could be further simplified if the source code for various standard controller blocks was provided. It would then be a simple matter to include the block required for a specific situation. This approach would provide an adaptable system with unparalleled flexibility,

especially as programmer used to connect to the DSO is simply a multiplexor and a connector, costing only a few pounds, a small price to pay for this level of flexibility.

The DSO connects to the PC using a standard parallel port, however the parallel port can be at different locations in the I/O space of a PC depending on the configuration, in addition to this a PC can support up to 3 ports. To resolve this the user can select which port the DSO is connected to from an easy to use drop down menu in the control software. However it would possible to make the software auto detect which port the DSO was connected to. In the present implementation only a single status bit is used to signal to the PC that the capture is complete. Connecting one of the unused status bits to one of the data bits would create a feedback path, the software could detect the presence of this, and therefore the DSO by outputting data to the data bit and monitoring the status bit. Because all standard parallel port devices only sample the data bits in the rising edge of the strobe pin this would not affect normal operation. Since all the status bits are connected to the FPGA for future expansion this could be implemented by simply reprogramming the system.

When in logic analyser mode the system is triggered when a pre-set value is present on the bus, however it is sometimes useful to be able to trigger the DSO when a sequence of values are present on the bus. For example if the system was monitoring the data bus of a processor the recording could be started when a certain sequence of code was executed. This can be implemented by storing the trigger sequence in a series of registers, which are compared to the current value of the data bus. A state machine could then be used to create a trigger pulse when the sequence was detected. The design for such a system was completed during the course of the project, however due to limited logic cell resources in the FPGA it could not be tested and is not present in the final version.

The total cost of the parts used in the construction of the DSO is approximately £75. This could significantly be reduced if the system went into volume production. Labour cost for the actual manufacturing process would be minimal due to the heavy use of surface mount components, placement of which is usually carried out by robotic equipment. In comparison a typical commercial PC based DSO module costs approximately £400, and does not posses the flexibility or capability of this system.

5.1) Possible Improvements

5.1.1) Sample Rate

To correctly refresh the DRAM either RCLK or WCLK must be greater than 1MHz. Under the current system RCLK is controlled by the PC, which can not be relied upon to provide this refresh signal, therefore WCLK must be greater than 1MHz, this imposes a minimum sample frequency. If during sampling RCLK could be controlled directly by the FPGA, WCLK could be clocked at much lower frequencies, in addition to this the ADC clock would also have to be kept running about 1MHz. If both of these criteria were met the DSO could be operated at much lower sample rates and therefore be used as a data logger. It would also be possible to implement an external clock input for the logic analyser, as this would be useful when monitoring synchronous systems.

5.1.2) Triggering

When in logic analyser mode the FPGA creates a trigger pulse when the pre-set value is present on the bus. To increase the flexibility of the system the ability to process “don’t care” states should be added. This for example, would enable the system to trigger off 10xx0x1x. To achieve this a second register could be added, which would be used as a mask for the incoming data.

It is often useful to remove high frequency components from signals before the triggering stage, this reduces the probability of triggering from signal glitches. This could be implemented by only generating a trigger pulse when the signal fulfils the trigger criteria over several samples.

The current DSO allows the user to place the trigger point anywhere in the recording by changing the initial value of the pre-trigger counter. If the size of this counter were increased then the start of the recording could be delayed. This would be useful in situations where the area of interest occurs a relatively long time after a trigger point.

5.1.3) Input Range

As mentioned previously to avoid clipping, the input signal must be in the range:-

$$0.375 < V_{Input} < 0.875$$

The next DSO should implement selectable voltage ranges and variable offsets. This would enable the DSO to be used with almost any input voltage range. In addition to this the choice between AC and DC signal coupling should be provided. This would allow the vertical scale in the DSO software to be calibrated, enabling the DSO to be used for voltage measurement.

5.1.4) Parallel Port Interface

Most of the time spent communicating with the DSO is taken up downloading the data from the FIFO's into the PC. Since the control lines are multiplexed, changing the state of the FIFO RCLK takes 4 separate writes to the parallel port registers. Therefore to read one byte from a FIFO takes 9 operations. This could be reduced to 3 by implementing a serial interface to the DSO control registers. So that one of the parallel port control bits could then be made available for direct connection to the FIFO RCLK. Although this would slow down some I/O operations this would be more than compensated for by a 3 fold increase in download speed.

5.1.5) FPGA Migration

The final design uses 86% of the available logic cells in the FPGA, before the above improvements are implemented the design should be migrated to a larger FPGA with more resources.

The Altera EPF8636A would be an ideal choice as it has over twice the number of logic cells and available in the same PLCC package as the current FPGA. This would require a small modification to the PCB as 8 of the pin assignments are different. Migration of the FPGA design would be a simple matter of recompiling the source files.

5.1.6) Software

By setting both /WRST and /RRST the FIFO could be used to transport data from the ADC directly to the parallel port. This would enable the software to provide a real time display of the signal being applied to the analogue input, this would greatly reduce the time required to set up the DSO for a specific application. Because of the flexible nature of the current design it would be possible to implement this without re-programming the FPGA or performing any hardware modifications.

In addition to acting as a DSO and a logic analyser the system could also be used as a spectrum analyser by implementing a Fast Fourier Transform (FFT) algorithm. As such algorithms are readily available this would be relatively easy to accomplish.

Because the control software uses direct hardware access to control the DSO it is not compatible with Windows NT. By converting the I/O routines to a device driver, compatibility with all known versions of Windows could be implemented. As the software is modular this could be achieved relatively easily.

6) Conclusion

The aim of the project was to design and build a PC based DSO, which has been achieved by using an FPGA. As well as simplifying the design this has enabled the following features to be added:-

Variable Pre-Trigger
Variable Sample Rates
Versatile Triggering System
Dual Channel Operation (50MHz)
High Speed Single Channel Operation (100MHz)
16 Bit Logic Analyser
Mixed Operation (1 Analogue Channel + 8Bit Logic Analyser)
3Mbit Storage Depth Per Channel
In System Programmable (ISP)
Windows Based GUI

This approach has resulted in great flexibility and expandability. Thanks to ISP, the DSO can now be targeted specifically at the system that it is monitoring.

The limited storage depth available on commercial DSO's greatly restricts their usefulness. In comparison the large storage depth of the project DSO (3MBits) allows almost 8mS to be captured at the maximum sample frequency. This enables the system to capture more than a whole video frame or to correctly sample an AM signal. The DSO has a maximum sample rate of 100MHz, which is comparable to most standalone systems and is much greater than that available on PC based systems. This allows a 50MHz signal to be sampled without aliasing, and 10MHz signals to be accurately represented. The ability to change the sample frequency enables the user can make even better use of the available storage depth, to extend the length of the recording.

As technology progresses embedded processors are becoming more and more common, which in turn means that a system that in the past would have been completely analogue now has digital components. To examine and debug these systems engineers have to use separate DSO's and Logic Analysers. Apart from the

additional cost of two separate pieces of equipment, this also creates problems in that it is not possible to completely synchronise the two systems. The project DSO solves this problem by integrating the two pieces of equipment into a single system, and combining it with a large sample depth.

Although the DSO is a useful tool in its current state, by implementing the following improvements the system could be transformed into one of the most versatile pieces of diagnostic equipment available.

Hardware Improvements:-

- Migration to larger FPGA
- Enhanced triggering
  - Don’t Care states
  - High Frequency Reject
- Input voltage range selection and calibration
- Parallel Port interface optimisation
- Creation of additional modules for the FPGA, this would allow the user to target the DSO for a specific application simply be adding the required modules.

Software Improvements:-

- Implementation of Fast Fourier Transform (FFT)
- Real time display
- Windows Device Driver

In conclusion the project has succeeded in producing a useful tool for the analysis of complex high speed analogue and digital signals.

7) References

1. ‘1210CSN’ Data Sheet
   AEL Crystals Ltd, 1997

2. ‘Configuring FLEX 8000 Devices’
   ‘Designing with FLEX 8000 Devices’
   ‘MAX + PLUS II’
   ‘Operating Requirements For Altera Devices’
   ‘Altera Programming Hardware’
   ‘Altera Device Package Information’
   ‘FLEX 8000 Programmable Logic Device Family’ Data Sheet
   Altera Corporation, 1999

3. ‘FPGA Configurator Programming Kit (Enhanced)’ Data Sheet
   ‘FPGA Configuration EEPROM Programming Specification’ Data Sheet
   ‘FPGA Configuration EEPROM Memory’ Data Sheet
   Atmel Corporation, 1999

4. ‘AL422’ Data Sheet
   Averlogic, 1999

5. Baxendale. P.R.
   Swift, J.S.
   Microprocessor Systems – 4H Course Notes

6. Cupitt, P.L.
   ‘Digital Storage Oscilloscope’
   University Of Durham, 1998

7. Gottfried, B
‘Schaum’s Outlines - Programming With C’
McGraw-Hill, 1996

8. ‘EL4332C’ Data Sheet
Elantec, 1996

9. ‘Semiconductor Data CD-Rom’
Farnell Components, 1999

10. ‘HI5667’ Data Sheet
Intersil, 1999

11. Mellor, J.E.
Advanced Digital Electronics – 4H Course Notes

12. ‘MECL system Design Handbook’
Motorola Inc. 1983

13. Microsoft Developer Network Library
Microsoft Corporation, 1998

14. Peacock, C
‘Interfacing the Standard Parallel Port’
‘Interfacing the Enhanced Parallel Port V1.0’
‘Interfacing the Extended Capabilities Port v1.0’
http://www.senet.com.au/~cpeacock

15. Rice, I.J.
‘Frame Grabber’
University of Limerick, 1998