High speed DSP and A to D in an Ice Radar logging application.

by

Andrew K. Brocklesby, B.Eng.
(Andrew Kenneth)

Submitted in fulfilment of the requirements
for the degree of

Master of Engineering Science

University of Tasmania

June, 1999

Declaration

This thesis contains no material which has been accepted for a degree or diploma by the University or any other institution, except by way of background information and duly acknowledged in the Thesis, and to the best of my knowledge and belief no material previously published or written by another person except where due acknowledgement is made in the text of the Thesis.

Andrew K. Brocklesby

Authority of access

This thesis may be available for loan and limited copying in accordance with the *Copyright Act* 1968.

Andrew K. Brocklesby

S76 Helicopter and author, Antarctica.

Abstract

For some time the Glaciology section of the Australian Antarctic Division has operated a land-based radar to measure the depth of glacial ice in Antarctica. The radar was mounted on a sled and towed around the Lambert glacier by bulldozer at 5km per hour. High power radio frequency pulses are transmitted down into the glacial ice, propagate through the ice and reflect off the bedrock below. The reflected pulses are picked up by a receiver, amplified and passed on to a signal processing section. The waveforms from the signal processing section are logged and displayed. The time taken from pulse transmission to detection, which is proportional to ice depth, may then be obtained. Signal averaging capabilities were required to improve the system signal to noise ratio and enable ice depths of over 3km to be observed. The ground based display and logging system used a digital oscilloscope with signal averaging capabilities to digitise and process incoming radar return echoes. Up to 256 radar return echo waveforms were averaged, the result being displayed and downloaded to an IBM PC via an IEEE488 link. Future ice depth measurements will involve airborne operations increasing the travelling speed from 5km per hour to 180km per hour. This speed increase will require greater data rates of the display, and logging system. In addition the airborne antenna, being carried by a helicopter, is physically smaller than the antenna mounted on the 60 tonne overland system. A smaller antenna leads to smaller antenna effective area, which in turn reduces the radar range. The high-speed digital signal processor (DSP) and high-speed analogue to digital converter (A to D) circuit board forms part of the ice radar display and logging system, replacing the slower digital oscilloscope system. The DSP will have sufficient speed to keep pace with the data rates required for airborne operation and provide improved signal processing power to compensate for reduced radar antenna gain.

As no suitable 'off the shelf' product was found the new system would utilise a custom designed high-speed DSP and high-speed A to D converter electronic circuit board. This document concerns the design of a high-speed DSP and A to D circuit board for an ice radar application.

Table of contents

Chapter 1 Introduction.

1.1 Description

This document concerns the development of a high-speed digital signal processor (DSP) and analogue to digital converter (A to D) circuit board. This board was produced in response to the requirement for an airborne ice radar system, used to measure the depth of glacial ice.

1.2 Background information

For some time the Glaciology section of the Australian Antarctic Division has operated a land-based radar to measure the depth of glacial ice in Antarctica. The radar was mounted on a sled and towed around the Lambert glacier by bulldozer at 5km per hour.

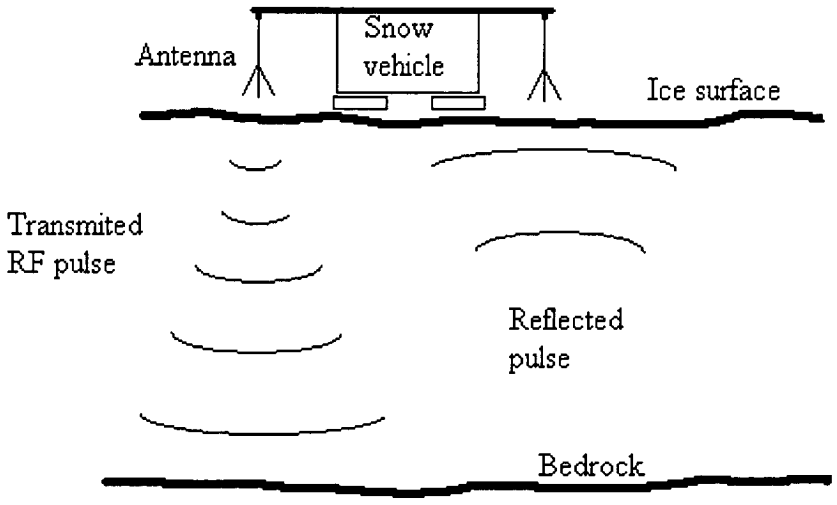


Figure 1.1. Sled mounted radar.

High power radio frequency (RF) pulses are transmitted down into the glacial ice, propagate through the ice and reflect off the bedrock below. The reflected pulses are picked up by a receiver, amplified and passed on to a signal processing section. The waveforms from the signal processing section are logged and displayed. The time taken from pulse transmission to detection, which is proportional to ice depth, may then be obtained. See Australian Antarctic Division Technical Note 12: Radar Ice Sounding at 100MHz. by I. Bird, B. Morton and A. Robinson and Radio-Glaciology by V.V. Bogorodsky, C.R.Bentley. P.E. Gudmandsen.

The figure 1.2 below shows a block diagram of the radar system.

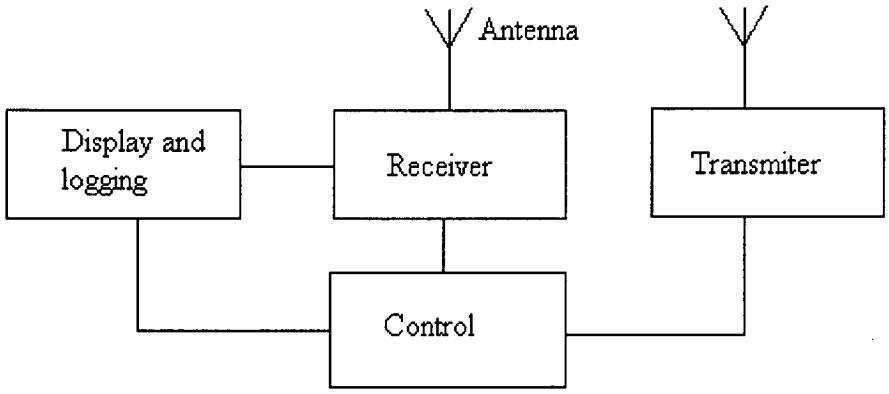


Figure 1.2. Block diagram of the Ice Radar system.

The 1990/95 ground based display and logging system used a digital oscilloscope with signal averaging capabilities to digitise and process incoming radar return echoes. Up to 256 waveforms were averaged, the result being displayed and downloaded to an IBM PC via an IEEE488 link. Signal averaging capabilities were required to improve the system signal to noise ratio, refer Radar Handbook by M Skolnik, Information, Transmission, Modulation and Noise by M. Schwartz. Using signal averaging, depths of over 3km where observed. The paper, Ice Radar Digital Recording, Data processing and Results from the Lambert Glacier Basin Traverses, M. Higham *et al* gives a description of the Antarctic Division ice radar expeditions 1990 to 1995. Subsequent work will involve an airborne ice radar system. (Refer ANARE Helicopter Operations handbook for details on airborne operations.)

1.3 Function of the DSP and A to D

Airborne operations increase the travelling speed from 5km per hour to 180km per hour, which demands greater data rates of the display, and logging system. In addition the airborne antenna, being carried by a helicopter, is physically smaller than the antenna mounted on the 60 tonne overland system. A smaller antenna leads to smaller antenna effective area, which in turn reduces the radar range. Refer to Radar Handbook by M. Skolnik.

The high speed DSP and A to D circuit board forms part of the display and logging section, replacing the slower digital oscilloscope system. The DSP will have sufficient speed to keep pace with the data rates

required for airborne operation and provide signal processing power to increase the radar system signal to noise ratio.

The High speed A to D and DSP board will provide the following essential functions for radar operation:

- digitise the incoming ice radar return echoes.
- provide signal processing on the return echoes.
- interface directly with the existing ice radar and IBM PC logging platform.
- carry out the above functions at sufficient speed to allow full speed airborne operations.
- improve signal processing capabilities to compensate for reduced radar antenna gain.

The DSP and A to D will have performance in excess of that required for the ice radar in an effort to anticipate future A to D and DSP requirements.

A survey of ice radars, in particular the waveform processing section, was carried out.

Both the Chinese Polar Research Institute and the Russian Polar Research Institute ice radar logging systems use a film recording method similar to that used in the 1975 Australian Antarctic Division ice radar, Bird *et al.* The radar return echo intensity modulates the x scan of a cathode ray tube display. The display is continuously photographed by a slow moving 35mm film. Several ice radar return echoes illuminated the same area on the film; the film therefore integrates these return echoes. Integrating return echoes in this manor

increases the system signal to noise ratio. Refer to Australian Antarctic Division Technical Note 12: Radar Ice Sounding at 100MHz. by I. Bird *et al.* Several examples of the film recording method can be found in Radio-Glaciology by V.V. Bogorodsky *et al.* The Antarctic Division replaced this system with a digital oscilloscope method in 1990 as described above. Refer to Ice Radar Recording, Data processing and Results from the Lambert Glacier Basin Traverses. Higham, *et al.*

The British Antarctic Survey (BAS) airborne ice radar digital system digitised radar return echoes using a digitising card with an ISA bus interface. The digitised data was passed to a processing card via the ISA bus. The processed data was then passed to a host computer via an Ethernet link. Refer to Description of British Antarctic Survey (BAS) Ice radar, 1996, System by Hugh Corr. Data throughput was limited by the digitiser to processor ISA bus link. This system was also large and expensive. The British Antarctic survey system is flown in a Twin Otter aeroplane.

The German Institute for Polar Research ice radar digital system records digitised radar return echoes directly to Exabyte data tapes. The operation is controlled by a 68000 and transputer microprocessor system. Radar return echo data is post-processed, requiring the storage of large amounts of raw data. This system was to be replaced by a storage oscilloscope method. The ice radar measured ice depth of less than 2km. See Description of German Institute for Polar Research Ice radar System by Dr Ludwig Hempel. Again the German system was relatively large and expensive. The German system is flown in a Dornier D0228 aeroplane, a large aircraft.

The University of British Columbia ice radar, designed to find ice depths of up to 1km, recorded its ice radar return echo data directly onto

storage tape without processing. Narod, B.B. and Clarke, G.K.C., 1994. Miniature high-power impulse transmitter for radio-echo sounding, J. Glac., 40(134), p.190-194. This system was small and compact but lacked the data rates and processing capabilities required. The Australian Antarctic Division aircraft would be a S76 helicopter with future work possibly carried out in the smaller Squirrel helicopter. See ANARE Helicopter Operations Handbook. Both helicopters have less equipment room than the British and German aircraft.

It was decided to investigate the possibility of obtaining a smaller cheaper and less complicated high-speed digital system than those discussed above.

Chapter 2 Ice Radar System Design Information

2.1 Ice Radar System Performance Information.

The high speed DSP and A to D board is designed primarily for the ice radar but where possible should have sufficient performance for other possible applications within the Antarctic Division.

The ice radar airborne system has the following performance requirements:

- Ice depth, GPS position and barometric pressure information is required every 50m travelled by the helicopter. The helicopter cruises at a speed of 180km per hour or 50m/s leading to an ice radar logging rate of once per second.
- The minimum ice depth resolution is 10m. The speed of electromagnetic radiation in ice is approximately 170m/μs, 10m travel through ice will therefore take 69ns.
- The maximum ice depth is 5km. For 5km ice depth the transmitted pulse will take 59μs to return to the radar receiver.
- An operator will monitor the processed radar return echoes, the GPS position and pressure measurements during the flight
- The operator may choose the logging period, the number of waveforms to be processed per log, the number of samples per waveform and the A to D rate

- The DSP system will take into account future Antarctic Division requirements were possible
- The IBM PCs have an ISA bus interface with a transfer rate of approximately 500kb/s
- The radar pulse repetition rate is 10kHz, i.e. 0.1ms period.

2.2 Mechanical Characteristics

As the space within the helicopter is limited the DSP and A to D system should take up minimal space. Placing the board within the logging and display PC will minimise system space requirements and provide physical protection for the board. The existing IBM PCs have extended ISA bus slots. One slot is allocated for the DSP and A to D board, dictating the real estate available. Low temperature design will not be necessary as the radar system will be powered on at least half an hour before use, allowing the electronics to warm up above 0C. All equipment installed in a helicopter is required to conform to Civil Aviation Authority regulations.

Chapter 3 Methods

3.1 Purchased DSP system versus in-house designed system

Initially it was hoped that a commercially available DSP and A to D system could be purchased. A survey of "off the shelf" DSP and A to D PC plug in cards was made. Although many DSP cards and A to D cards exist, matching the A to D speed and DSP requirements on a single board proved difficult. Most DSP cards have slow speed A to D's associated with them (aimed at the audio market) or have digital interfaces requiring an additional A to D board. A two board (A to D board and DSP board) system has severe cost, system complexity and possibly higher data transfer overheads. Many of the boards had suitable DSPs but had timing overheads associated with transfer of data from the A to D to the DSP and from the DSP to the PC.

As no suitable product was found it was decided to produce an in house designed board. This has the following advantages:

- there will be no reliance on third party suppliers
- the board can be customised directly to our needs
- after the initial R and D costs the cost of in house produced boards are favourable
- software and hardware compatibility is guaranteed.

3.2 Requirements of an in-house designed DSP and A to D board

The requirements of the DSP for the radar system are outlined in section 1.3. A DSP system that fulfils those requirements and have a general-purpose nature would have the following characteristics:

- high-speed computation.
- be programmable
- high A to D to DSP transfer rates.
- high DSP to PC transfer rates.
- be able to accommodate varying A to D requirements.
- the board would have an IBM PC ISA bus interface

3.3 Design methods

3.3.1 The DSP design method.

As the DSP algorithms may vary a programmable rather than a dedicated fixed algorithm solution was chosen. This leads to the choice of a DSP chip or programmable state machine as opposed to fixed discrete computational units. Programmable state machines were considered but did not have significant advantages over the DSP chip solutions and had real estate disadvantages. A DSP integrated circuit (IC) solution was therefore chosen.

3.3.2 Fabrication

The DSP and A to D system would utilise surface mount IC technology where possible to achieve maximum component densities. Reworking high-density surface mount components becomes extremely difficult with a high pin count device. A major requirement was the need to avoid hardware reworking. The printed circuit boards are very expensive and the high-speed nature of the design along with wide buses precludes rough printed circuit board (PCB) modifications. By using a DSP chip and in-circuit programmable logic (ISP) minor errors in design could be rectified in software without the need for a second expensive printed circuit board. Considerable thought was given to the inputs and outputs of the programmable logic devices in an attempt to anticipate any requirements arising at a later date. This led to some redundancy in the ISP device IO but was at an acceptable minimum. The A to D section involves analogue design where the physical layout of the PCB may affect the design performance. The performance of the analogue design cannot be truly evaluated until it is built and tested. This leads to a high probability that a second PCB design may be needed. The cost and time for this second PCB was included in the time and money budget.

3.3.3 DSP to PC interface

The DSP was to be programmed and controlled via the PC. The PC would need to write data to program memory and data memory, be interrupted by the DSP and have data transferred to it without slowing

the DSP. The DSP would need to have interrupt facilities from the PC and be able to receive and send data without incurring speed penalties. A two way signalling process would be required.

3.3.4 A to D section

The A to D section would be on a piggyback board, allowing for different piggyback boards to be used with the same DSP. See figure 3.1 below.

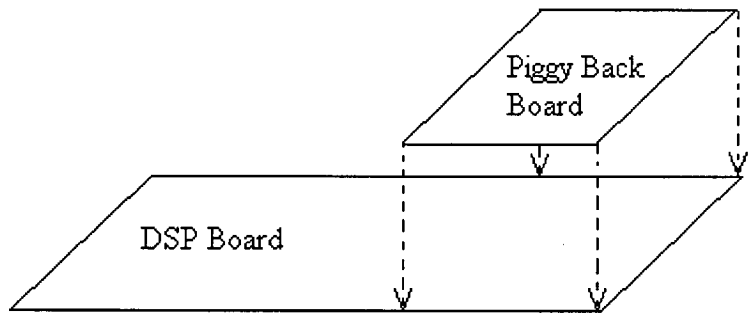


Figure 3.1. Piggyback system

The A to D section would need to send data to the DSP at a sufficient rate to ensure full speed DSP operation.

3.3.5 Software

The DSP algorithms would be written in assembler rather than a higher level language to obtain speed efficient code. With assembly language programs the designer has full control over the DSP operation making it possible to create faster run time code than with a higher level

program. Analog Devices also recommends assembly language programs for speed critical operations and supply many speed efficient assembly program routines such as FFTs etc. See Analog Devices ADSP-21000 Applications Handbook, Volume 1. C programs could be considered for larger programs but would depend on the speed optimisation abilities of the compiler. The PC logging and control software would be written in C for speed (compared to other high level languages) and its ubiquitousness.

Chapter 4. DSP and A to D System Design.

4.1 DSP and A to D system

The DSP and A to D system would have the following characteristics:

- The DSP board would be designed using a programmable DSP IC
- The DSP would be programmed via the IBM PC
- The A to D would be built on an interchangeable "front end" A to D Piggyback board.

4.1.1 Technology

Market survey shows most DSP utilise TTL compatible technology.
A single technology will reduce the system complexity, therefore where possible TTL compatible technology would be chosen.

4.1.2 The DSP section.

The DSP design is dependent on the DSP IC chosen. The choice of IC depends on the processing speed required, the dynamic range and often on the type of algorithm to be computed. Many manufacturers quote benchmarks for speed but those benchmarks can often be tailored to a particular IC. Although the processor clock speed is important it is not the definitive reference for speed. A high-speed processor has the following characteristics:

- able to sustain single cycle instructions such as multiply and accumulate.
- efficiently access data to sustain single cycle instructions.
- effect zero program looping overheads.
- maintain circular data buffers.
- have sufficiently large dynamic range to avoid overflows.

Once the hardware characteristics of the IC are taken into account some practical aspect of design are considered:

- cost, not only of the DSP chip but the cost of support products such as an emulator, software emulators and software support. As we did not intend to go into production the cost of the IC was not a major consideration.
- support availability.
- hardware availability.

A market survey revealed that most manufacturers were concentrating on 32 bit wide, fixed point, data bus DSP IC development. The fastest DSPs available were therefore 32 bit. For the ice radar application the DSP will process 8 bit A to D data. Averaging 1024 waveforms will require an 18 bit wide (255 x 1024) data bus, a 32 bit data bus will therefore meet ice radar requirements. For these reasons a 32 bit fixed point DSP was chosen.

A survey of the large number of manufacturers led to a comparison between Texas instruments and Analog devices DSP ICs.

The final two processors under scrutiny were the Texas instruments TMS320C30 and the Analog devices ADSP21020. The survey lead to the following comparisons:

- Both are capable of 32 bit fixed point and floating point operations. The 21020 is capable of 40 bit floating point operations.
- The 21020 can generate interrupts on ALU overflows and other operations whereas the C30 requires a conditional branch.
- The two computational architectures are different. The 21020 architecture is much more versatile than the C30, particularly with the barrel shifter.
- Comparison of the processor instructions shows the C30 functions are at best a subset of the 21020.
- The 21020 program sequencing and external memory accessing was superior to the C30. In each case the C30 had higher overheads resulting in slower overall computational times.
- Analog devices produces a few bench mark comparisons of the 21020 and the C30. In the benchmarks investigated the 21020 is significantly faster overall than the C30.

Refer also to Analog Devices Application Note AN-235, Considerations for Selecting a DSP Processor.
The 21020 was chosen on the basis of its superior performance and prompt reply from an easily accessible help line.

4.1.2.1 The DSP IC

Figure 4.1 below shows a block diagram of the ADSP21020.

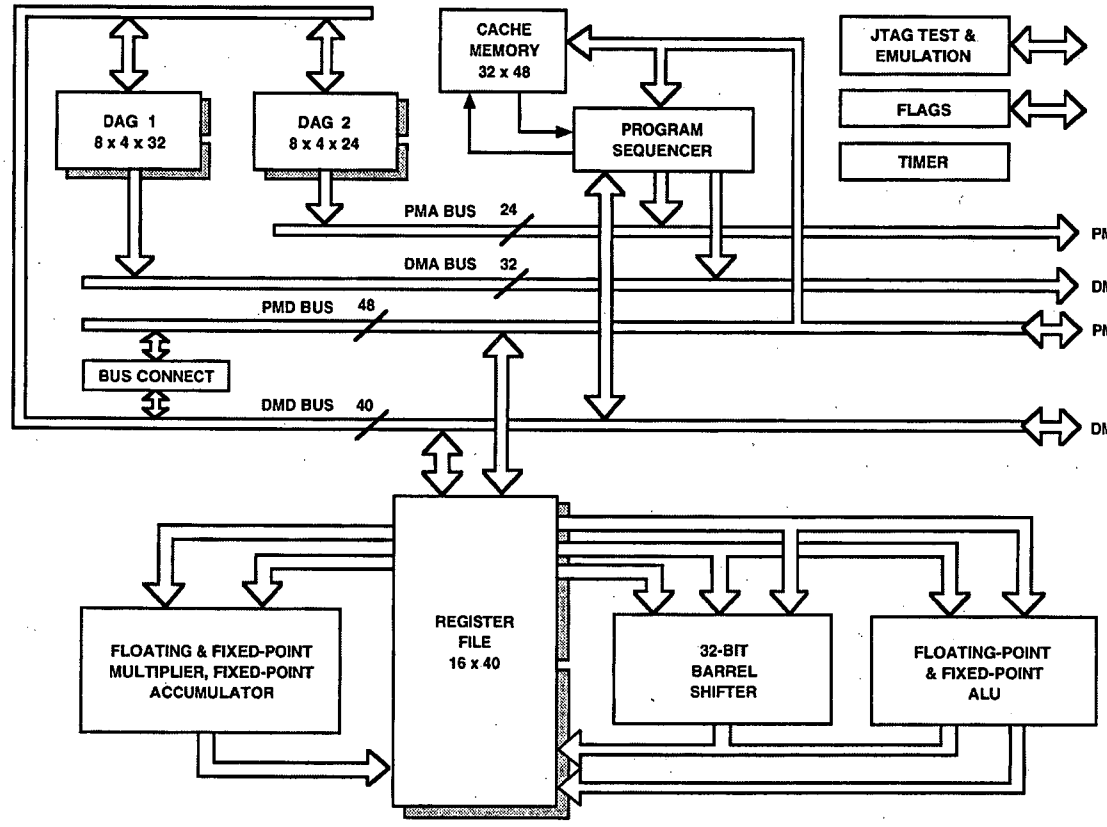


Figure 4.1. Block diagram of the ADSP21020.

The 21020 has a Harvard architecture, i.e. a separate program and data bus. It is capable of simultaneous, single cycle, program memory access, data memory access and arithmetic operations. It also has internal program memory cache. This allows the program memory to be used for data storage without reducing processor speeds. The user manual gives a detailed discussion of the 21020 architecture.

4.1.2.2 Interfacing to the ADSP21020

Figure 4.2 below shows a basic system configuration.

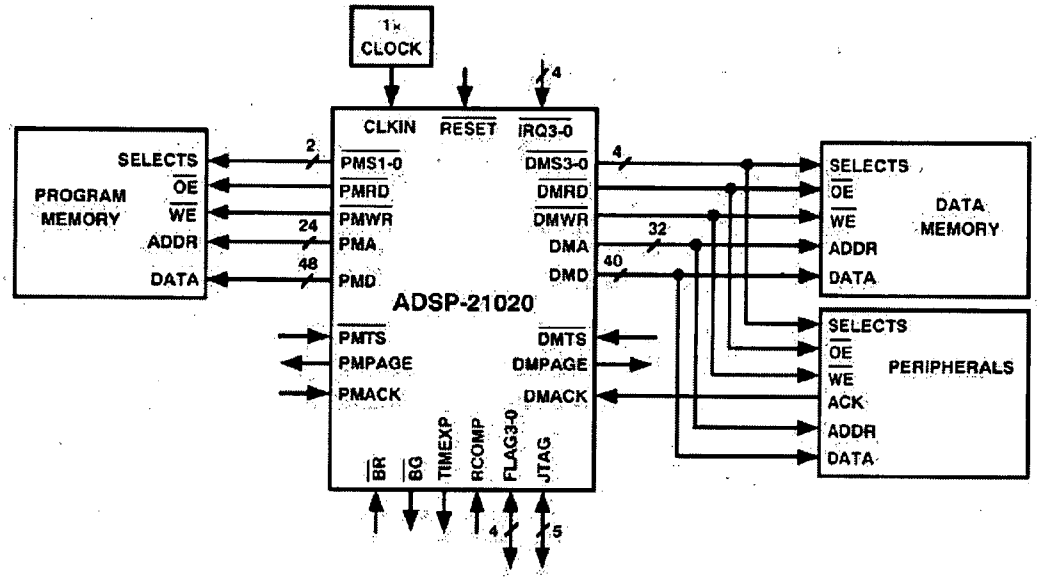


Figure 4.2. Basic system configuration.

The ADSP21020 data sheet gives a full pin description. The 21020 has the following to support system interfacing:

- The external memory interface supports memory mapped peripherals. Memory mapped select pins, DMS0-3 and PMS0-1, can be used as chip selects for the corresponding memory mapped peripheral. The memory map position of these select pins are software programmable.
- Bus request, BR, and bus grant, BG, signals. If an external processor wishes to gain control of the memory busses it asserts BR.

On receiving a BR the 21020 puts the memory busses and control pins in a high impedance state. It then asserts BG to indicate it is no longer driving the memory busses.

- 4 bi-directional flag pins allowing single bit signalling between the 21020 and peripherals.
- 4 DSP hardware interrupts.

Figure 4.3 below shows the 21020 signalling interface for our system.

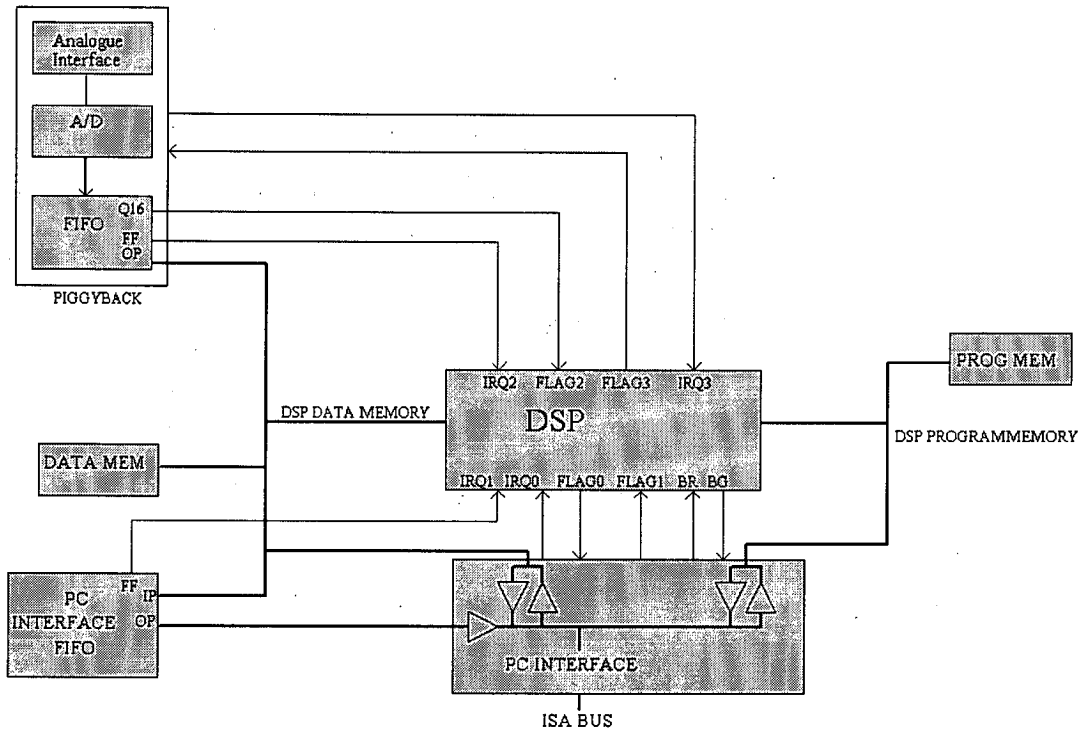


Figure 4.3. ADSP21020 system signal interface.

The DSP interfaces to program and data memory, an A to D piggyback board including an A to D FIFO and the PC ISA bus including a PC FIFO.

The DSP interrupts are connected to the FIFO full flags, FF, on the A to D piggyback board and the PC interface. Connecting the FIFO full flags to an interrupt alerts the DSP if data is being lost due to either the DSP not emptying the A to D data fast enough or the PC not reading DSP data fast enough. The interfaces will now be discussed in greater detail.

4.1.2.3 DSP external memory requirements

The 21020 requires separate program and data memory. The data and program memory address range is 4G words and 16M words respectively. A survey of the high speed static RAM IC market revealed the 128k by 8 bit wide configuration to be an industry standard. Several manufacturers supported pin for pin compatible products. The assembly language ice radar data logging programs initially written were of the order of 10k bytes. Two 6k word waveform samples are stored in memory for the logging programs. This led to the conclusion that 128k words of program and data memory would be sufficient.

The DSP program memory is connected to PC interface transceivers and RAM. These transceivers are used for DSP and PC data interfacing. The DSP data memory is connected to the A to D FIFO output, the PC interface FIFO input, RAM and the PC interface transceivers. These transceivers are used for DSP and PC data interfacing.

Figure 4.4 below shows the memory bus interfaces and memory select pin usage.

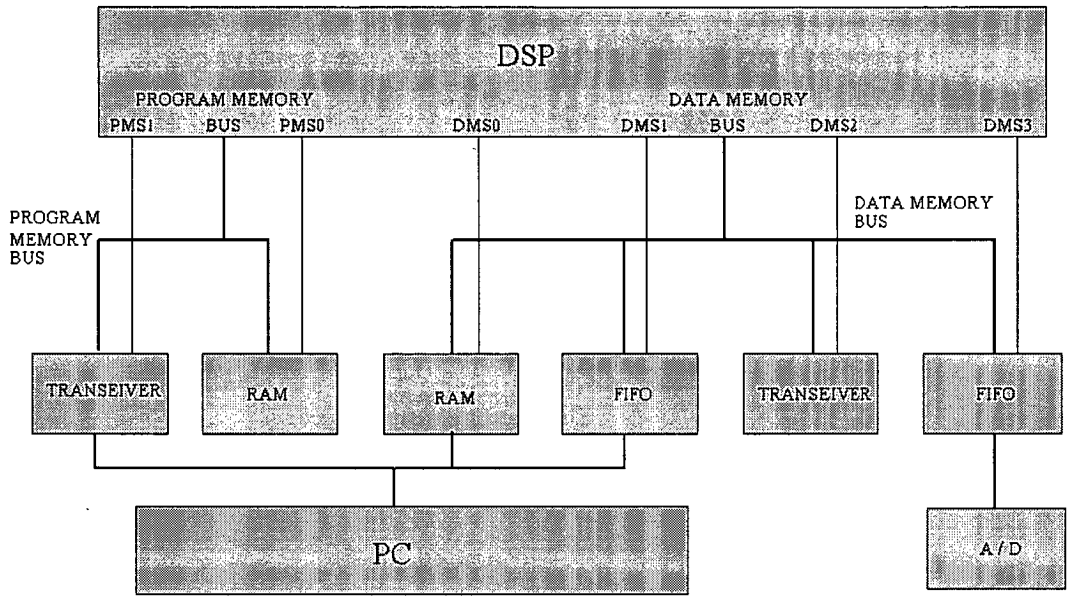


Figure 4.4. DSP memory bus connections.

The DSP is able to read data from, and download data to, the PC ISA bus via its program and data memory busses.
The FIFO's facilitate high-speed asynchronous transfer of blocks of data to and from the DSP.

4.2 DSP to Piggy back board interface

Three separate power supplies are sent to the piggyback board, a +5v digital supply, a +8v analogue supply and a –8v analogue supply. The +5v digital power supply is taken directly from the ISA bus. The +12v and - 12v supply from the ISA bus are filtered and regulated down to 8v to present a clean supply for analogue applications. Two physically

separate grounds are routed to the piggyback board one for analogue applications and one for digital.

The signals routed to the piggyback board from the DSP were chosen to give the interface a high degree of versatility.

The Piggyback board is data memory mapped via the data memory select 3 (DMS3) line and data memory address lines 0 to 16 from the DSP. The table below shows the complete set of signals to the piggyback board.

Signal interface to the piggyback board.

CLK1	DSP clock
DMACK	Data memory acknowledge signal
DMS3	Data memory select 3 line
DMA 0 – 16	Data memory address
DMD 8- 39	Data memory data
DMW	Data memory write strobe
DMR	Data memory read strobe
IRQ3	Interrupt request
IRQ2	Interrupt request
FLAG2	Flag (bi-directional)
FLAG3	Flag (bi-directional)
TIMEXP	Counter time out signal from the DSP.
RESET	Reset signal from PC.

4.3 DSP to PC ISA bus interface.

The DSP to PC interface is shown in figures 4.3 and 4.4 above.
The DSP card is given PC ISA bus IO addresses 300 hex to 31F hex, reserved by IBM for users. See PC Instrumentation for the 90s, Boston Technology and PC programmers Bible, P Norton. The ISA bus data width is 16 bits on even addresses or 8 bits on odd and even addresses. The PC controls the DSP board operation by downloading instruction words and data to the DSP board. Communication between the PC and DSP can be polled or interrupt driven.
Data can be passed between the PC and DSP via four avenues:

1. The PC FIFO connected to the data memory bus. Data is written to the FIFO from the DSP at full speed. The PC reads the FIFO via the ISA bus. The two operations are asynchronous.
2. Via transceivers connected to the data memory bus. The DSP and PC can latch data into, and read from, the data memory bus transceivers. The two operations are asynchronous.
3. Via transceivers connected to the program memory bus. The DSP and PC can latch data into, and read from, the program memory bus transceivers. The two operations are asynchronous
4. The PC writes and reads data directly to and from program memory. The PC gains control of the DSP memory busses via the DSP BR and BG signals as described below.

To write to the DSP program memory RAM the PC first sends a BR signal. The DSP responds by tristating its memory bus pins and asserts

BG. On receipt of the BG signal the PC downloads the 17-bit program memory address and program memory select PMS0 to the PC interface ISP logic device. See figure 4.5 below.

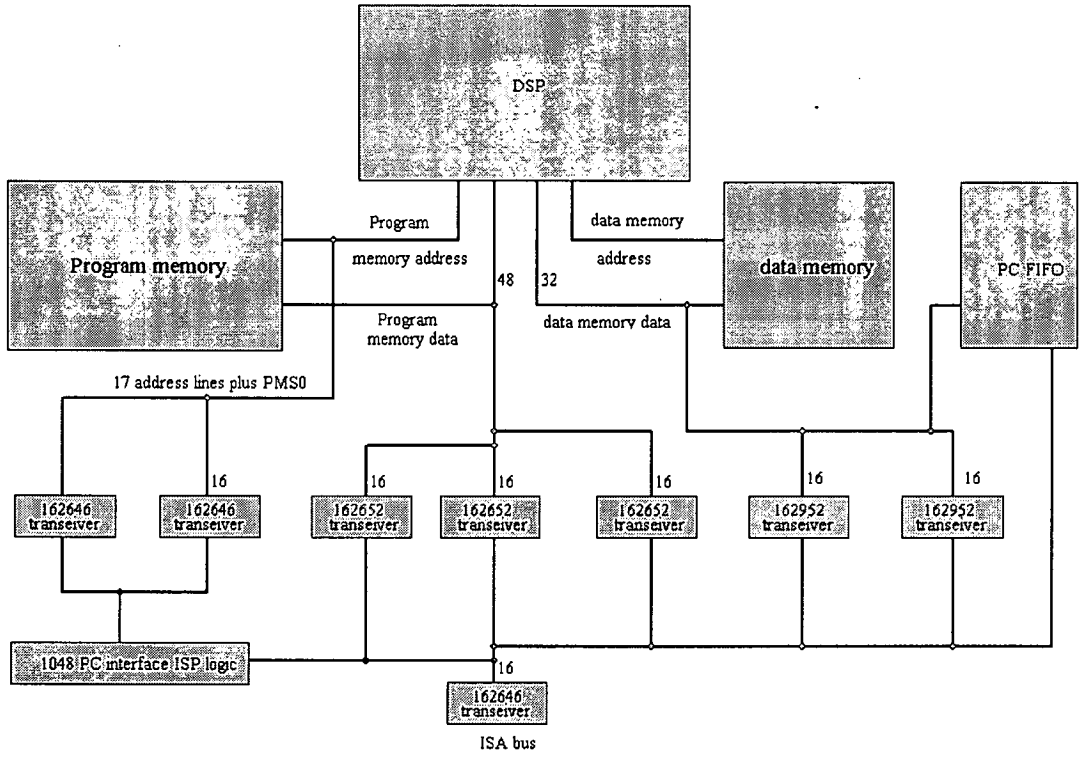


Figure 4.5. DSP to PC interface.

The PMS0 and upper address bit, bit 16, are latched into one of two memory address bus transceivers. The lower 16 bit of the memory address, bit 0 to 15, can be supplied by the PC direct or generated in the 1048 ISP logic device. Generating the lower 16 bits of the address within the ISP logic device reduces the number of PC to DSP board data transfers. If a block of data is to be transferred the lower 16 bits of the address can be supplied by a counter within the ISP device that is incremented after each write to RAM. Thus given a start address and number of memory locations, a block of contiguous RAM locations can

be written without the PC supplying each RAM address location. The 48 bit program memory data is written, 16 bits at a time, to 3 latches connected to program memory data lines. A write strobe is provided via the ISA bus. Reading the RAM is similar, except the 48-bit word is provided from the DSP program memory RAM. See figure 4.5. In this way the DSP program and data coefficients can be downloaded to the program memory and verified. The DSP data memory RAM cannot be accessed directly by the PC. Data can be placed in DSP data memory by transferring it from the data memory bus transceivers or by transferring from the DSP program memory, each requiring the intervention of the DSP.

FLAG 0 and 1 are used for communication between the DSP and PC. FLAG 0 may be used to interrupt the PC via the PC IRQ15 hardware interrupt line (allocated to the DSP board by the user) or its status read for polling operations. When used to interrupt the PC, FLAG 0 sets a flip-flop within the PC interface ISP logic device. The output of this flip-flop controls the state of the PC IRQ15 signal. The PC interrupt routine must reset this flip-flop. The PC drives DSP interrupt IRQ 0. DSP interrupt IRQ 1 is driven by the PC FIFO full signal to alert the DSP that the PC is not reading the FIFO fast enough.

DSP set-up and control signals from the PC are stored in one of two latches within the PC interface 1048 ISP device. Status signals from the DSP board are passed to the ISA bus via the PC interface ISP device. The table below shows the function of each bit in the control latches and status buffer.

Control latch 1

bit	Function
0	Drives DSP BR signal (active low)\
1	Drives DSP FLAG 1 input
2	Interrupt DSP via IRQ0
3	Resets PC FIFO

Control latch 2

bit	Function
1	Software reset DSP (active low)
3	DSP FLAG 0 to PC mode, when set to 1 FLAG 0 causes PC interrupt

Status buffer

Bit	State of signal
0	BG
1	DSP FLAG 0
2	PC FIFO empty flag
3	DSP Timexp signal

The ISA bus IO space map is shown below.

Address (hex)	PC Operation	Comments
300	Read PMD bits 0-15	PC takes control of DSP bus

301		
302	Read PMD bits 16 - 31	PC takes control of DSP bus
303		
304	Read PMD bits 32- 47	PC takes control of DSP bus
305		
306	Read PC FIFO data	
307		
308	Read DSP data memory bits 8 - 23	
309		
30A	Read DSP data memory bits 24 - 39	
30B		
30C		
30D		
30E	Read DSP status latch	
30F		
310	Write 16-bit program memory address word	
311	Reset DSP	
312	write PMD 0=15	
313	Write to control latch 1	
314	write PMD 16-31	
315	write to control latch 2	
316	write PMD 32-47	
317	Output a PMW from PC	
318		
319		
31A		

31B	reset PC IRQ latch
31C	write DMD bits 8-23
31D	Latch program memory address bit 16 and PMS0 from PC
31E	write DMD bits 24-39
31F	

4.4 A to D piggy back board interface

Refer to figure 4.3 on page 26 and figure 4.4 on page 28. DMS3 and data memory address 0 to 16 are used to memory map the A to D piggyback board. Control data is up loaded to the A to D using the upper 16 bits of the data memory data bus and the A to D output is downloaded to the DSP on the lower 16 bits. The A to D FIFO full signal is routed to DSP IRQ2 alerting the DSP if it is not keeping up with data flow from the A to D piggyback board. FLAG2 and FLAG3 are used to communicate with the A to D piggyback board. FLAG2 is connected to bit 16 of the A to D, 18 bit wide, FIFO output. This bit originates in the A to D logic and can be used to tag A to D data, (see section 4.9.4). FLAG 3 is routed to the A to D control logic and may be used to initiate the start of an A to D sequence. IRQ3 is derived from the A to D logic to facilitate interrupt of the DSP by the A to D operation.

4.5 A to D converter system

The digitising oscilloscope used in the original land based ice radar system digitised to 8 bits. The new A to D system will also digitise to 8

bits. (The user can alter the ice radar base band amplifier gain thereby changing the system dynamic range).

The minimum sampling frequency for any design is the Nyquist frequency. One problem with sampling close to the Nyquist frequency is that it puts a heavy demand on the front end anti aliasing filter, the input filter to the A to D. The ideal filter has very sharp cut-off at the Nyquist frequency, that is, the wanted signal frequencies below the Nyquist are un-attenuated and the unwanted frequencies above the Nyquist rejected. To achieve a practical filter with these characteristics would be extremely difficult. The project would become a filter project. A poor filter with a poor role off and sufficient attenuation at the Nyquist frequency will attenuate the wanted frequencies, distorting the signal. Sampling at a rate greater than twice the highest signal frequency of interest, i.e. over sampling, relaxes the filter requirements. (See discussion in section 4.6). In addition, a signal digitised at the Nyquist rate could not be displayed immediately and give a true representation of what the A to D "saw". For example, sampling a sine wave, with an A to D converting at this sine waves Nyquist frequency, then displaying the A to D output would show a square wave not a sine wave without further processing. As the sampling frequency is increased the fidelity of the displayed sign wave is increased. The disadvantage of over sampling is that more memory is required to store the "extra" samples.

The A to D speed was chosen to be as fast as possible without the need to go to special technologies. The faster the speed the more applications the A to D could encompass. The highest speed solid state A to D converters use ECL technology. An ECL device was not chosen for the following reasons:

- An ECL A to D would mean mixing logic technologies, leading to greater system complexity.
- As the DSP is TTL compatible ECL to TTL converters would be required which in turn require additional PCB real estate. The market is spars on these ECL to TTL converters as most modern designs use custom ECL to TTL devices, an expensive process.
- The ECL devices would require their own –5.2v power supply.
- A survey of TTL compatible A to Ds revealed that sufficiently fast TTL A to Ds were available.

For the ice radar project a minimum of 8-bit precision is required. To achieve a high speed TTL A to D converter an Analog Devices dual A to D, each capable of 50MSPS conversion rate, was used in an interleaved configuration. See figure 4.6 below.

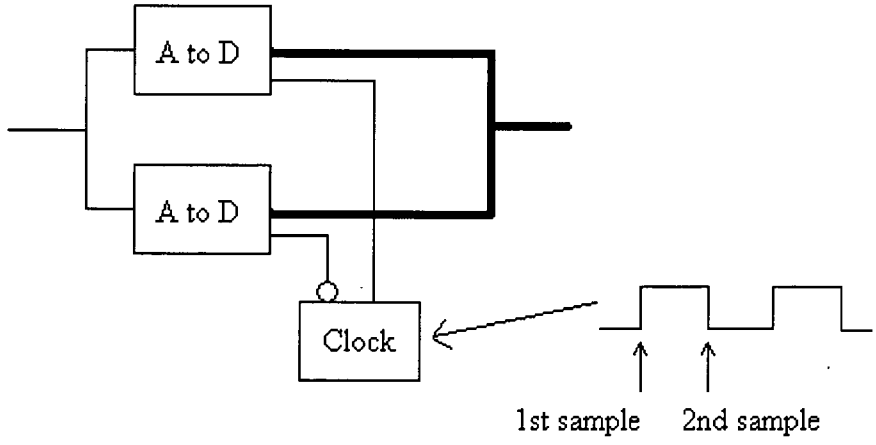


Figure 4.6. Interleave A to D system.

Using two A to D in this configuration gives a system A to D with double the individual A to D conversion rate. This would lead to

100MSPS A to D while still having a 50MHz A to D clock and TTL components. Although two A to D converters are used the additional PCB real estate required is minimal as the two A to D and a voltage reference are in the same IC package. This system had some unexpected results and was replaced by a non interlaced system. See later discussion in chapter 6.

4.6 Front end A to D anti aliasing filter.

An active front end anti aliasing filter was chosen to minimise the real estate required. The number of poles of the filter would be dictated by the A to D speed and the space available for it. An anti aliasing filter is required to attenuate frequencies above the Nyquist frequency to below the least significant bit (lsb) of the A to D. An anti aliasing filter for a 100MSPS 8 bit A to D would therefore require an attenuation of at least 256 or 48.2dB at 50MHz.

Any practical filter will distort the signal it is filtering. The type of filter chosen, and the degree of distortion introduced, depends on the signal characteristics being measured. The design must take into account the effect of the filter transient and phase response, as well as frequency response, on the signal. Refer to chapter 2 in 'Electronic Filter Design Handbook' by Williams and Taylor.

The slope of the filter frequency response in the stop-band will depend on the filter type and the number of poles of the filter. We may consider the 3dB point of a low pass filter as being the upper boundary of the pass-band. The A to D sampling rate and accuracy determines the Nyquist point in the stop band. The 3dB point is found by

'working back' from this point in the stop-band. For example: A five pole Bessel low pass filter would give 50dB attenuation at 50MHz satisfying the Nyquist requirements for an 8 bit 100MSPS A to D. Working backwards the 3dB attenuation point, the pass-band limit, would be 10Mhz. Notice that the 100 MSPS A to D would be said to over sample a 10MHz signal. Over sampling reduces the requirements of the anti aliasing filter.

4.7 Designing high speed logic circuits

Most electronic engineers are familiar with the techniques used in slow speed (below 10MHz) digital design such as with LS technology. Much of the design consists of a system approach were building blocks are linked together. The designer thinks in terms of fixed propagation delays and clean well-behaved signals that give a good binary model. Voltage and currents are usually thought of only in terms of power supply requirements. With high-speed logic, design techniques normally associated with high-speed analogue and RF design have to be applied. Transmission line effects, distributed (PCB) components etc, have to be taken into account. For example, at low frequencies to create a short between two pins a piece of wire is placed between the pins. At high frequencies the same wire may appear as an inductor and not function as a short, the designer has to take the physical properties of the wire into account. The first point to establish in a high frequency design is how high a frequency do we need to consider. Consider the experiment where a flip-flop is clocked at a rate of F_{clock} and has an input toggling randomly between 0 and 1. The flip-flop

output rise and fall times are less than 1% of the clock period. The flip-flop output is now input to a spectrum analyser to reveal its spectral power density. Its spectrum would look like that shown in figure 4.7 below.

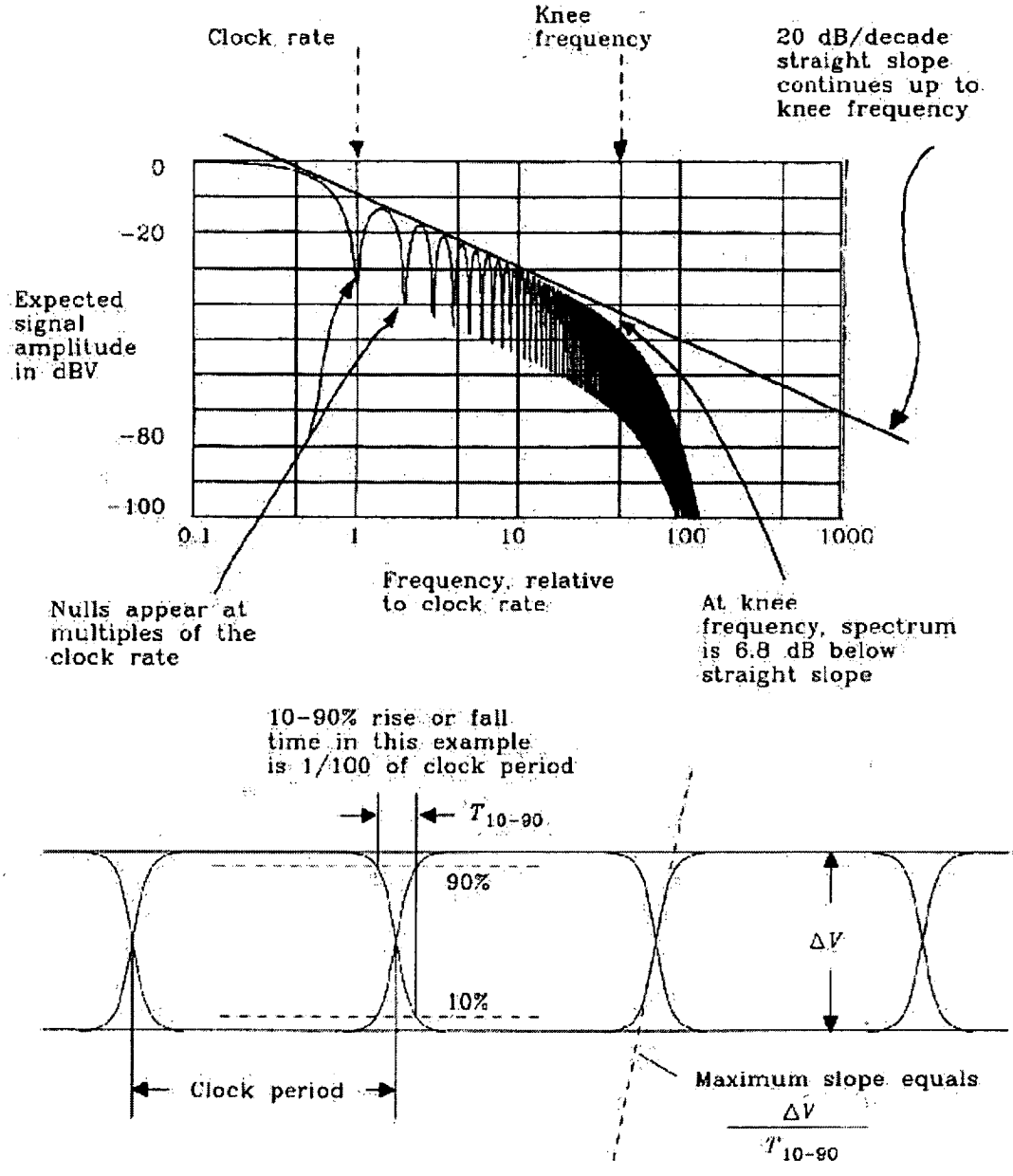


Figure 4.7. Power spectrum of a random digital waveform.

The spectrum nulls at multiples of the clock frequency and has a –20dB/decade slope up until the frequency, F_{knee}. Beyond F_{knee} the

spectrum rolls off much faster. The knee frequency, F_{knee}, is related to the rise and fall times of the signal not the clock rate. At F_{knee} the spectral amplitude is down by half, (6dB). A circuit with a frequency response up to F_{knee} will pass a digital signal with minimal distortion. Frequency responses above F_{knee} of a circuit will have minimal effect on the processing of a digital signal. F_{knee} is therefore considered the upper frequency in a high speed design.

The above discussion shows that not only are low propagation delay devices considered high speed devices but devices with fast rise and fall times. The highest frequency of interest may be taken as:

$F_{knee} = 0.5/Tr$, where Tr is the rise or fall time.

This can be used to give a guide for the high frequencies involved in the design.

Typical rise and fall times for the 21020 read and write strobes are about 1ns depending on load capacitance. This yields a F_{knee} frequency of about 500MHz. To obtain a good low distortion digital model the circuit components, including the PCB, must have a favourable frequency response up to 500MHz.

The next problem is to decide at what point do we need to consider a PCB track a transmission line. To do this we need to calculate the effective length of the fastest signals. The effective length of a digital signal is the length of conductor required for a signal to traverse the logic levels. See figure 4.8 on the next page.

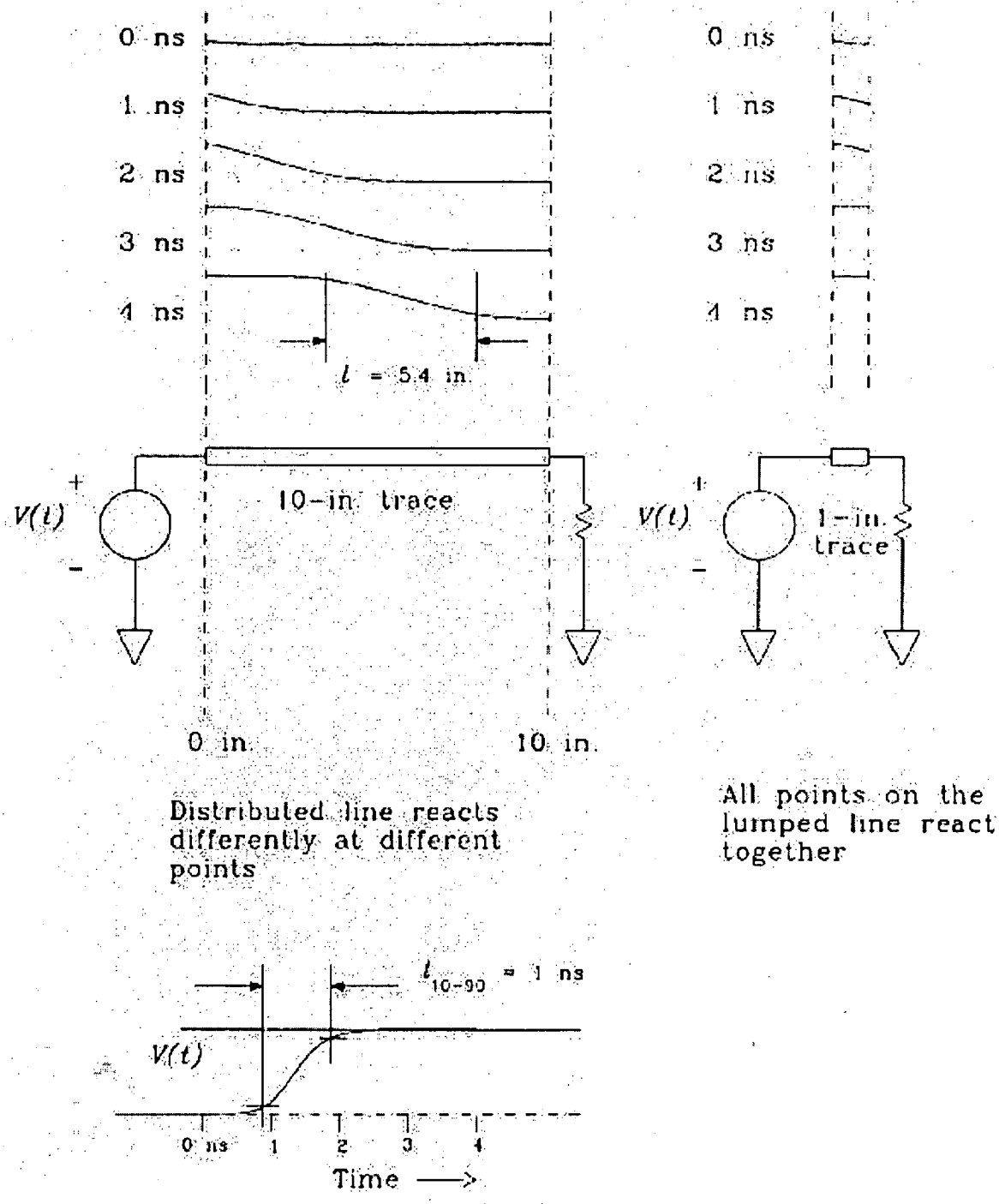


Figure 4.8. "Snapshot" in time of a digital signal voltage on a transmission line.

The effective length, L, is given by

$$L = Tr/D$$

Tr = rise time

D = Propagation delay (the inverse of propagation velocity)

D is proportional to the square route of the PCB dielectric and can be obtained from the PCB manufacturer. For a 1ns rise time and a propagation delay of 180ps/inch (dielectric constant of 4.5) the effective length, L, is about 5.5inches. For a track length greater than 5.5 inches the rising edge of a pulse will be seen to propagate along the track, the potential is not uniform at all points on the track. At these lengths voltage reflections and therefore transmission line properties will need to be considered. At track lengths much shorter than 5.5inches, say 1 inch, the voltage can be considered as uniform along the track. In this case the track does not need to be considered a transmission line.

Not only do the signal tracks need to be considered but the ground returns also. If a ground path is poor voltage differences will exist along the ground. For example, if a narrow track is used as a ground return it may be inductive. Fast changes in current will cause a voltage difference along this track. For high-speed work a large solid ground plane is preferable. Note that high-speed signals follow the path of least inductance not least resistance. A low inductance path is as important as a low resistance path.

The final point to consider is cross-talk caused by mutual inductance and mutual capacitance. By ensuring there is enough physical

separation between signal carriers and considering the signal timing, cross-talk problems can be eliminated. The amount of cross-talk introduced may be determined if the mutual capacitance and mutual inductance between tracks, IC legs and components is known. The difficulty is that both these quantities cannot be calculated until the layout is at least near completion (by which time its too late). The rule of thumb is to use the amount of PCB space available and determine the cross-talk empirically if need be. (Some PCB layout techniques are still considered art-work). A discussion of cross-talk is given in 'High Speed Digital Design, A Handbook of Black Magic' by Johnson and Graham.

When high speed digital signal timings are determined the load capacitance of a signal must be estimated. At high speeds the rise and fall times of a signal are of a significant length to affect timing requirements. For example, in writing to RAM the address hold from end of write may be 0ns. If the address line capacative load is significantly less than that of the write strobe, the address lines switch much faster. See figure 4.9 on the next page.

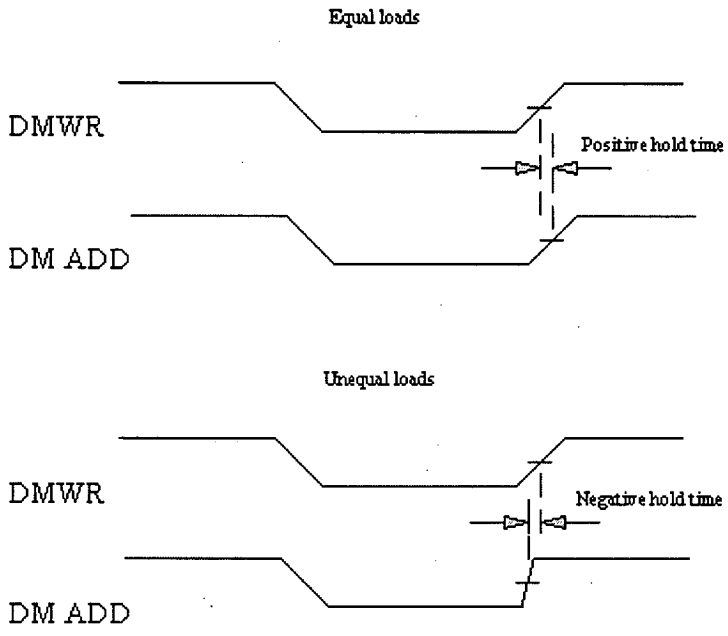


Figure 4.9. Example of the effect of capacative loading on timing.

This being the case the address hold time is violated. The 21020 data sheets specify rise, fall and output delay times for a given load capacitance. PCB layout and system topology will effect load capacitance.

The above discussion can be condensed into the following as far as high-speed TTL and CMOS designs are concerned:

- Signal tracks greater than 1 inch (25mm) are to be considered as transmission lines. This means that incorrectly terminated signal

lines will cause ringing, overshoot and undershoot. These will introduce signal distortion. Undershoot may cause device damage.

- Currents flowing through a poor ground will cause potential differences between the grounds of different devices. Resistance will cause DC power currents to introduce a ground potential difference. Inductance will cause potential difference in grounds when rapidly changing currents pass through them.

- Adjacent signals may couple together creating cross-talk.

- Propagation delays may cause significant timing changes. For example keeping the track lengths in a data bus fixed and changing the length of the write strobe line will change the timing between them.

- Device timings is load dependent, i.e. changing the load capacitance will change the switching times. The 21020 data sheets give output timing versus load capacitance graphs. The 21020 Users Manual, section 9, page 27, gives an example of the effects of uneven memory bus loading.

Most of these effects can be accommodated with careful topology and PCB layout. The PCB layout is as important part as any other part of the hardware design.

4.8 The 21020 DSP software.

The 21020 has a large and powerful instruction set. See ADSP21020 Users manual for full details. Efficient use of the 21020 architecture greatly speeds up the system processing power. Efficient code is organised to take advantage of the 21020 parallel operations and hardware capabilities. For example a multiply, an addition, a move of data from program memory to a register, increment of a data pointer and a move of data from data memory to register can take place in one clock cycle. The user must organise the code to make use of these parallel facilities.

Assembly code gives greatest control over the DSP operation and therefore generates the fastest possible programs. Section 7.2 gives an example of 21020 assembly code. This example, the DSP radar logging program, shows many of the 21020s features and strengths.

4.9 The system implementation for the ice radar.

4.9.1 Signal processing.

In order to increase the ice radar system signal to noise ratio the number of return echo waveforms to average was increased from 256 (the maximum number possible with the existing digital oscilloscope system) to 1024. An ice radar return echo waveform could be up to 60μs long (as noted in section 2.1). This leads to 6000 samples per waveform at an A to D sampling rate of 100MSPS. The clock period of the ADSP21020 is 33ns. To average 1024 waveforms at 6000 samples

per waveform in less than 1 second, 4 DSP clock periods per sample would be available (leading to a total DSP processing time of approximately 0.8s).

4.9.2 Transfer of data to the PC

The PC ISA bus can transfer 500kb per second and can be 8 or 16 bit wide. See PC Instrumentation for the 90s. Boston Technology. The averaged waveform from the DSP would be limited to 16 bit data, therefore one 6000 point averaged waveform would take approximately 24ms to transfer to the PC. The transfer of data to the PC does not require the DSP intervention if the PC FIFO is used.

4.9.3 Front end anti aliasing filter

The filter would need to have a linear phase change response to preserve the shape of a radar pulse. A 5 pole Bessel filter (chosen for its phase characteristics) would give 50dB attenuation at 50MHz satisfying the Nyquist requirements for an 8 bit A to D. The 3dB bandwidth would be 10Mhz. This would pass a low fidelity pulse with a width of 100ns and high fidelity pulse of width 200ns.

4.9.4 A to D for the radar application

To display a reasonable fidelity pulse without signal processing at least 10 samples of the pulse would be required. 100MSPS A to D would give reasonable display of a 100ns pulse and a high fidelity display of a 200ns pulse without post processing. For a 5km radar range the system, sampling at 100MSPS, would need to sample 6000 points starting as the radar transmits. A slower sampling option is available to reduce the data storage requirements when using longer radar pulse widths. The radar provides a trigger pulse, the rising edge of which occurs just before the radar transmits its RF pulse. The A to D may start sampling on the rising edge of the trigger pulse. (For versatility a choice of A to D triggering mechanism is available, an external trigger, the DSPs TIMEXP signal or the DSP FLAG3 signal). To ensure data synchronisation the digitised radar return echo waveform is tagged at a known point, for example at the first sample of the waveform, see figure 4.10 below.

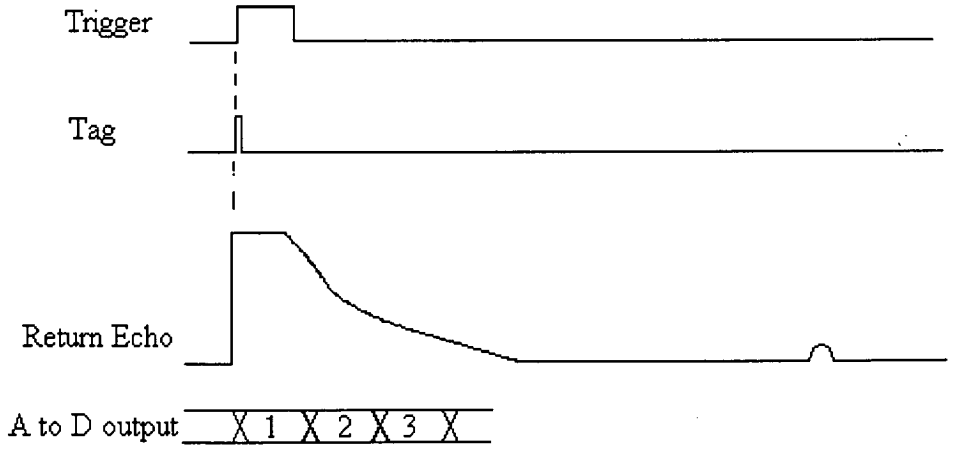


Figure 4.10. Example of radar return echo tagging.

For the transfer of data from the A to D to the DSP, FLAG2 is tagged, that is, FLAG2 is set high when the first sample of the waveform is transferred to the DSP. FLAG2 may then be used in a conditional instruction to ensure synchronisation without incurring overheads, such as additional lines of code to test a data bit. For example the following program will repeat the 'DO' loop, code1 to code4, until FLAG 2 input goes high.

```
DO label UNTIL FLAG2_IN;
            code1;
            code2;
            code3;
label:      code4;
```

A counter clocked by the A to D sampling clock would be used to determine the number of samples recorded.

Two A to Ds systems were considered for this design. The first uses the interleaved system discussed in section 4.5 and was later replaced by a second non interleaved system. See discussion on chapter 6.

A to D 1

The first A to D system utilised the Analog Devices AD9058 dual A to D, each capable of 50MSPS conversion rate, in an interleaved configuration. See figures 4.11 and 4.12 on the next page.

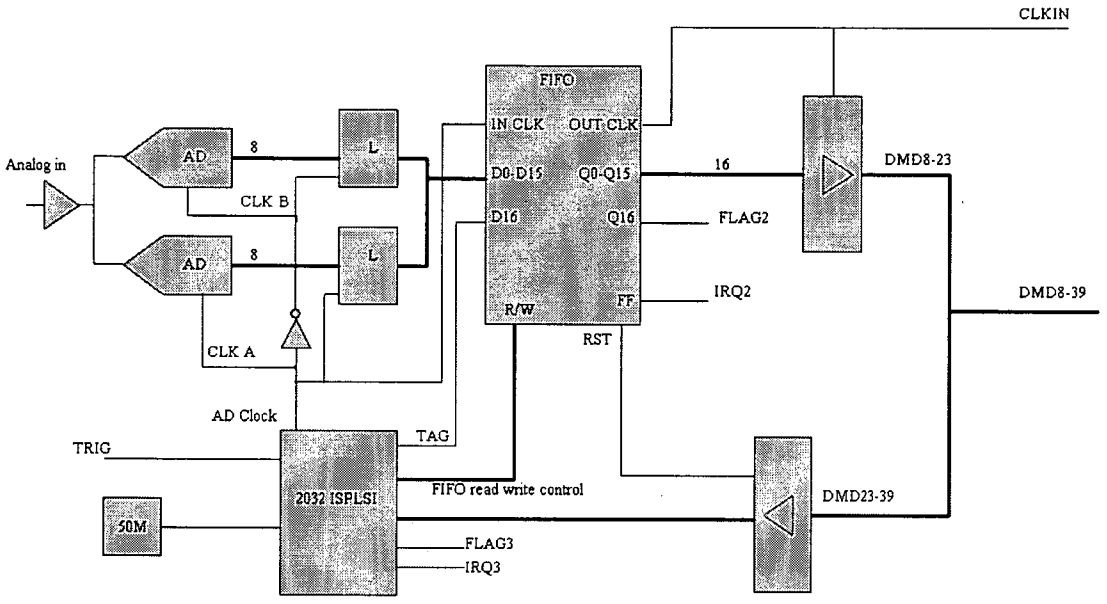


Figure 4.11. Block diagram of dual A to D system.

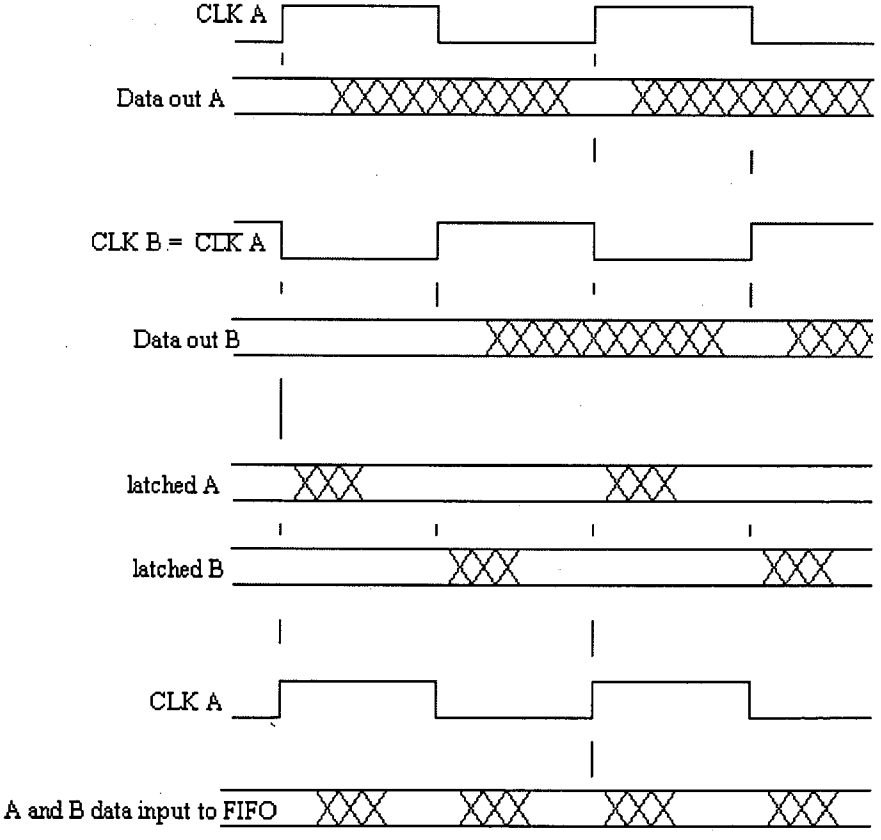


Figure 4.12. Timing diagram for dual A to D system.

The two A to D encode pins are driven 180 out of phase effectively giving a sample every 1/2 clock period. A 50MHz clock is used to give a 100MSPS A to D system. The advantages of this system over a direct 100MSPS sampling rate being:

- a slower clock speed simplifying the design timing constraints,
- TTL component use,
- the layout of a PCB containing a 50MHz clock is simpler than a layout containing a 100MHz clock
- A 100MHz system is generally more complex than a 50MHz system.

The two 8 bit A to D outputs are paralleled to form a 16-bit word that is stored in the 18 bit wide A to D FIFO at bits 0 to 15.
This A to D system operates as follows: On system start up the A to D board is reset. On receipt of a radar transmit trigger a 12 bit down counter (allowing for up to 2 x 4k A to D samples) is loaded with the number of A to D samples to be stored for the input waveform.
A corresponding tag bit is stored in the FIFO, at bit 16, to mark the trigger point data. The same clock that drives the A to D clocks the counter. As the counter counts down the FIFO fills. When the counter reaches zero the counter halts and the FIFO stops filling. On receiving a FLAG3 signal from the DSP the counter is able to reload on the receipt of another radar trigger signal. In this manner the DSP is able to request another radar waveform using FLAG 3. The DSP reads the FIFO and is alerted to the start of a new waveform via FLAG2 connected to FIFO output bit 16. The tag is placed on the FLAG2 signal not a memory data line; this enables the FLAG test instructions

(discussed above) of the 21020 to be used. The tag may also cause a DSP interrupt, via DSP IRQ 2 that may be masked if not required.

The A to D requirements are loaded in the upper 16 bits of the data memory data bus as: -

DMD24 to DMD33	Number of samples required. This number is used to load the upper 10 bits of a 12 bit down counter the lower 2 bits are set to 0 on the load.
DMD 34	Tag mode
DMD35,36	Trigger mode
DMD37	Sampling rate
DMD38	Logic reset
DMD39	A to D FIFO reset.

This system gave some unanticipated results when used with signal averaging. This is discussed later chapter 6. New higher speed TTL A to D converters, FIFOs and clock driver circuits entered the market allowing a true 100MSPS A to D with TTL technology to be designed. The dual A to D, AD9058, system was therefore replaced.

A to D 2

The second A to D used a true 100MSPS A to D IC, the Analog Devices AD9012. See block diagram figure 4.13 below.

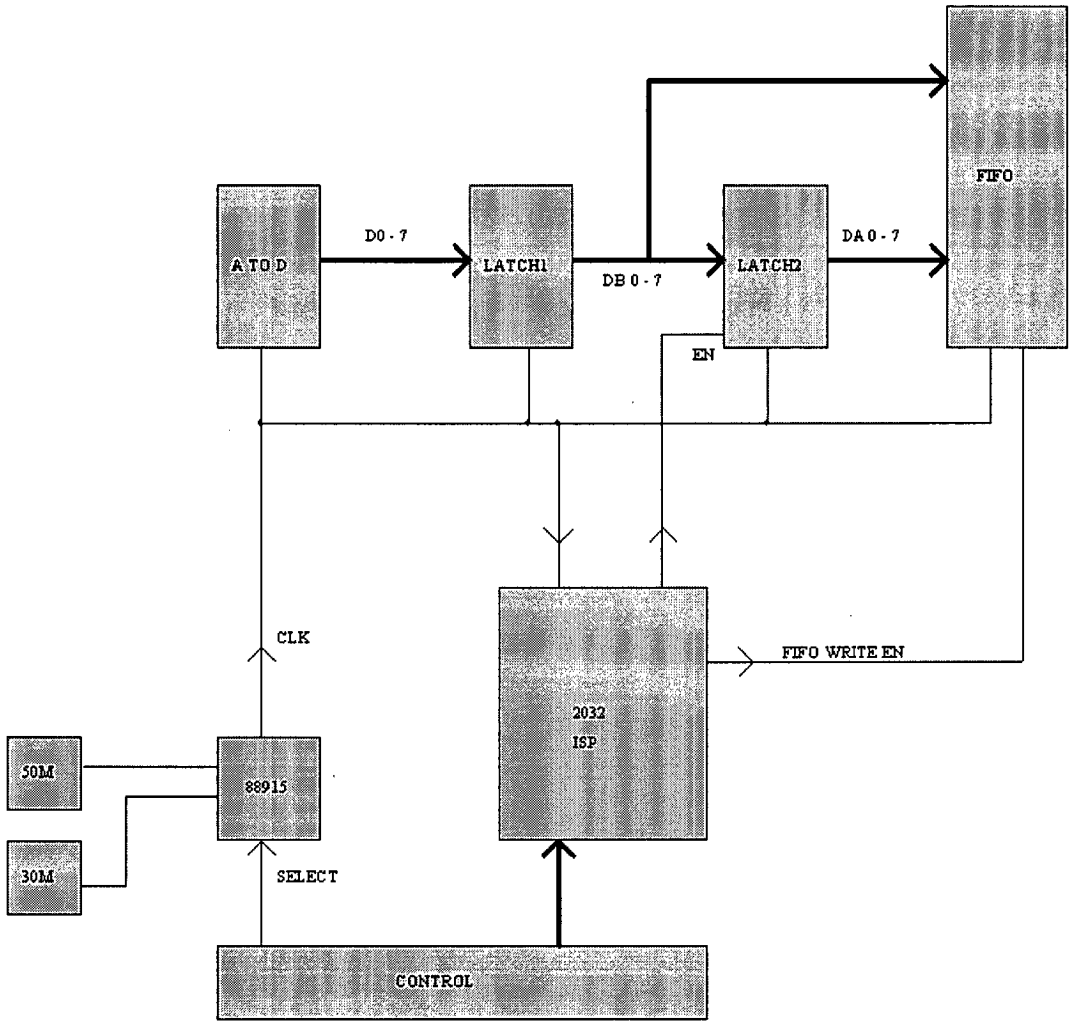


Figure 4.13. Block diagram of true 100MSPS A to D system.

The A to D control system is similar to the first interleaved A to D system. The A to D output data rate was slowed to half speed by paralleling two 8 bit A to D outputs. The 16 bit word containing the

two A to D outputs is stored in the 18 bit wide FIFO. See block diagram above and timing diagram figure 4.14 below.

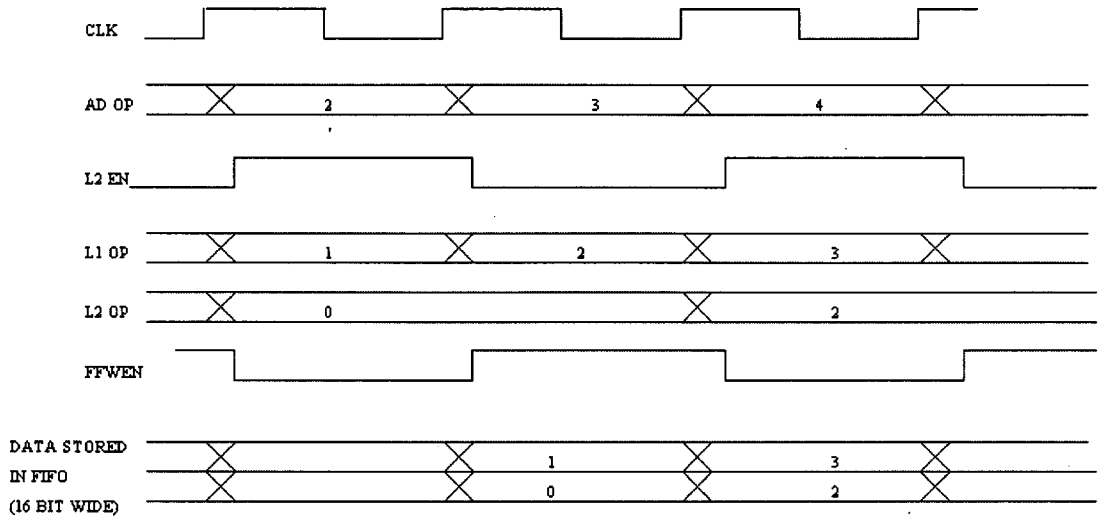


Figure 4.14. Timing diagram of 100MSPS A to D system.

The counter containing the number of samples is now 13 bits (maximum of 8k samples) due to the use of a single A to D. The board now contains a 100MHz A to D clock, increasing the speed complexities. The main impacts of the faster clock being the tighter restrictions on the control ISP logic 2032 and transmission line effects of the high speed clock. The A to D requirements are loaded in the upper 16 bits of the data memory data bus as: -

DMD24 - DMD34	number of samples required. This number is used to load the upper 10 bits of a 13 bit counter the lower 3 bits are set to zero on load.
DMD 34	Tag mode

DMD35,36	Trigger mode
DMD37	Sampling rate
DMD38	Logic reset
DMD39	A to D FIFO reset.

Chapter 5. Circuit Design.

5.1 Component choice.

5.1.1 IC components

The digital devices are all TTL compatible were possible. Surface mount components are used to minimise PCB real estate requirements.

5.1.2 Decoupling capacitors and inductors

Low effective series resistance (ESR) capacitors are mandatory along with the selection of capacitor materials and component values to ensure all noise frequencies of interest are suppressed. Typically for each IC a 1nF and 0.1µF npo type capacitor is used for decoupling. Type npo capacitors were chosen for their high frequency characteristics. Every capacitor will have series resonance characteristics. A lower capacitance value capacitor will resonate at a higher frequency than a higher capacitance value capacitor. Above the capacitor resonant frequency the capacitor behaves inductively and is a poor decoupler. The 0.1µF values will decouple lower frequencies and the 1nF capacitors will decouple higher frequencies. The IC lead inductance also helps decouple most of the ICs internally generated high frequency noise. A PCB power and ground plane form an excellent decoupling capacitor. The power to the board is filtered with low impedance special polymer 4µ7 and 10µ electrolytic capacitors and 'lossy' ferrites.

5.1.3 Glue logic.

The glue logic chosen was in-circuit programmable logic, (ISPL). The greatest problem with prototyping is the removal and addition of surface mount ICs. Using in-circuit programmable logic would enable the IC to be reprogrammed without being removed. There are a number of producers of in circuit programmable logic, each manufacturer tends to specialise in a particular area such as size, power or speed. Our first priority was for speed, the second non-volatile programming, the third density and fourth cost. The Lattice semiconductor devices were the fastest (comparable to the standard advanced Schotky speeds), were EEROM based and had reasonably high densities. There were a wide range of programming choices, with a wide range of costs. (Lattice could provide a budget programming version for $2000, compared to Xilink $5000).

The ISP logic devices used are manufactured by Lattice and were chosen on the basis of speed and programming software costs. These devices are programmed using the Lattice development software (pDS). The logic devices can be optimised for routing efficiency or speed. The devices were roughly routed first to ensure the pin selection was valid and the required logic would "fit". The pin allocations could then be fixed and the PCB sent for manufacture while the logic software was being completed. Refer to Lattice user manual for detailed descriptions.

Two Lattice ISP devices were used the 1048 and the 2032.

The lattice ISP2032 logic device.

This device is small in density but has very high speed, comparable to AS TTL. Figure 5.1 below shows a representation of the device logic.

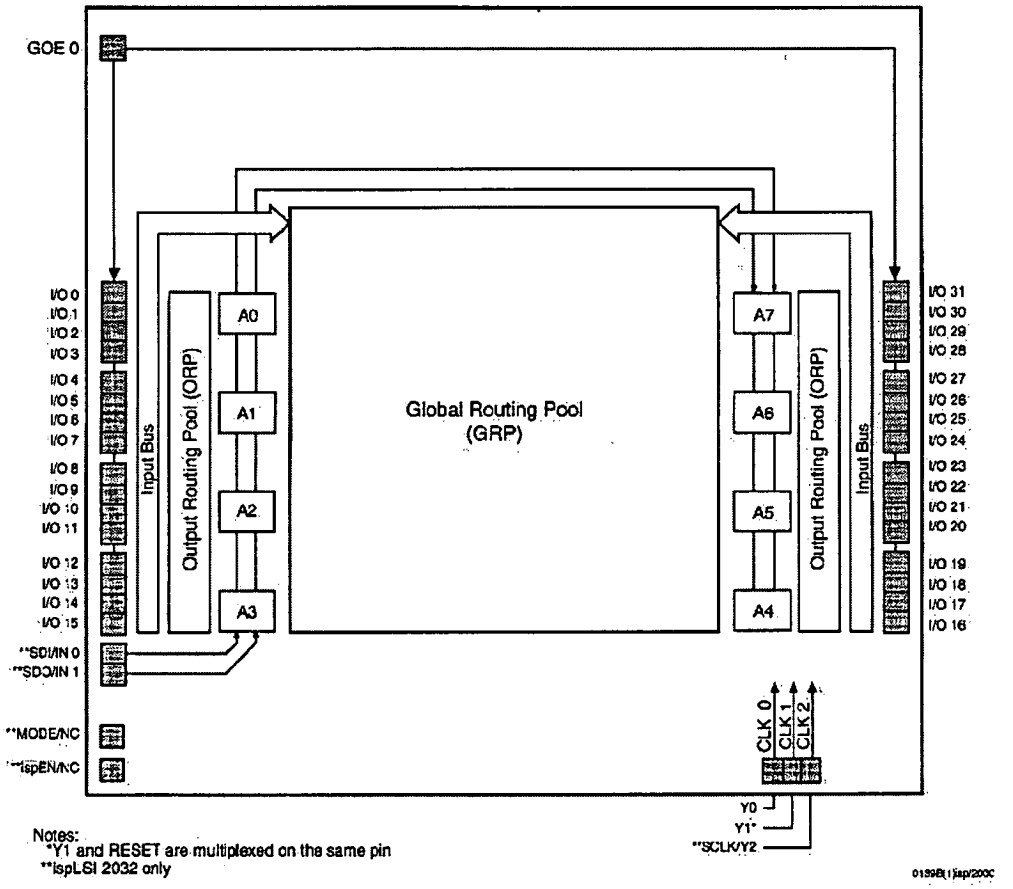


Figure 5.1. 2032 ISP functional block diagram.

Each of the blocks A0 to A7 represents a generic logic block (GLB). The logic block description is shown pictorially in figure 5.2 on the next page.

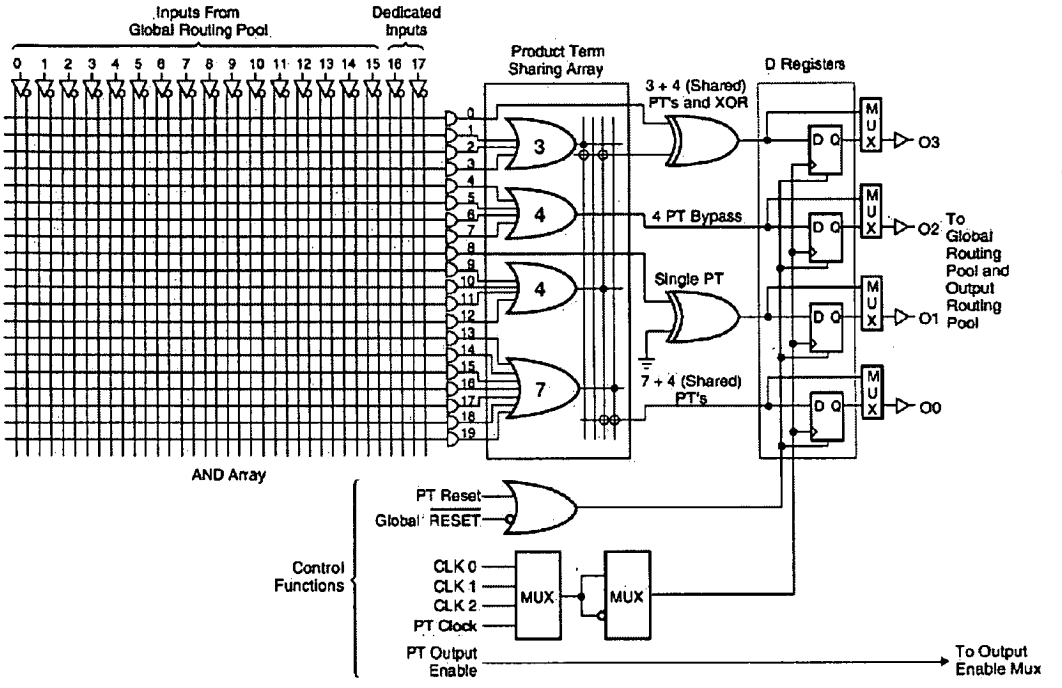


Figure 5.2. Example GLB configuration.

This block has 17 inputs and 4 outputs. The example configuration of the block above shows that the output product terms can be summed with other product terms, exclusive ORed with other product terms or simply output without addition logic. The global routing pool (figure 5.1) can route any input to any output. The outputs from the logic blocks can be routed to any of the IO blocks, IO0 to IO31, via the output routing pool (figure 5.1). Similarly the inputs from the IO blocks can be routed to the global routing pool. The IO blocks can be configured as inputs, outputs or bi-directional with tristate control. The output routing pool and the product term sharing arrays may be bypassed for high-speed operation.

The main problems associated with the ISP logic design was getting the logic to 'fit' into the device while having control over the device routing. The routing of the device effects the logic timing. For example, if the term A + B + C + D + E is required the first four terms could be produced by one GLB by simply ORing the terms A to D. The ORed output would then be ORed with E. This would create two GLB timing delays. By creating !A.!B.!C.!D.!E in one GLB and inverting the result on the IO block one GLB timing delay is introduced leading to a faster logic solution.

The ISP 1048 logic device.

The 1048 device is slower than the 2032 but contains much more logic. Its architecture is similar to the 2032 but contains forty-eight GLB's and 96 IOs.

5.2 DSP circuit design.

The DSP circuit diagrams are shown in figures 5.3 to 5.9 on pages 62 to 68. A discussion of the main sections of the circuit follows.

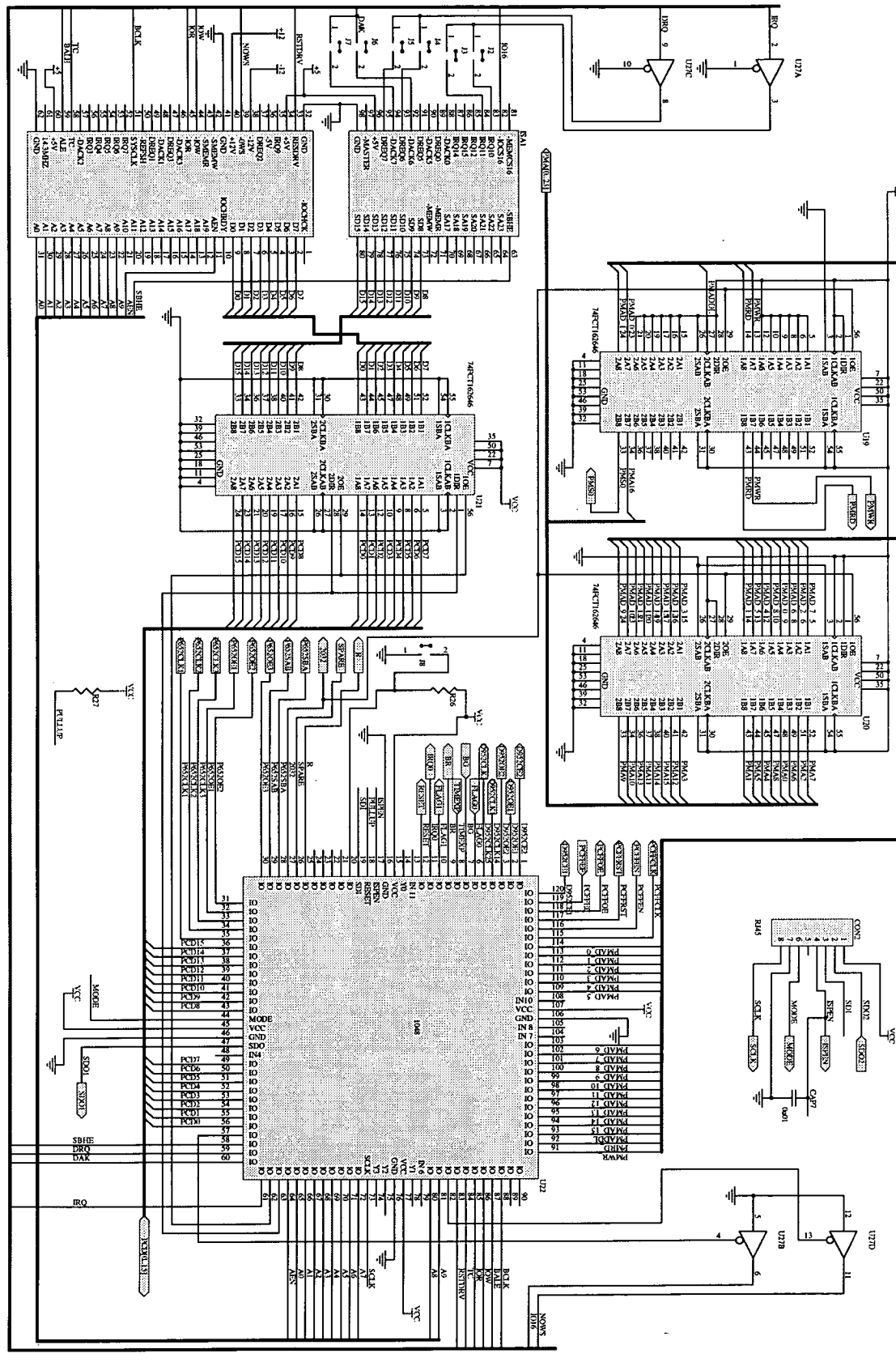


Figure 5.3 DSP Circuit Part 1

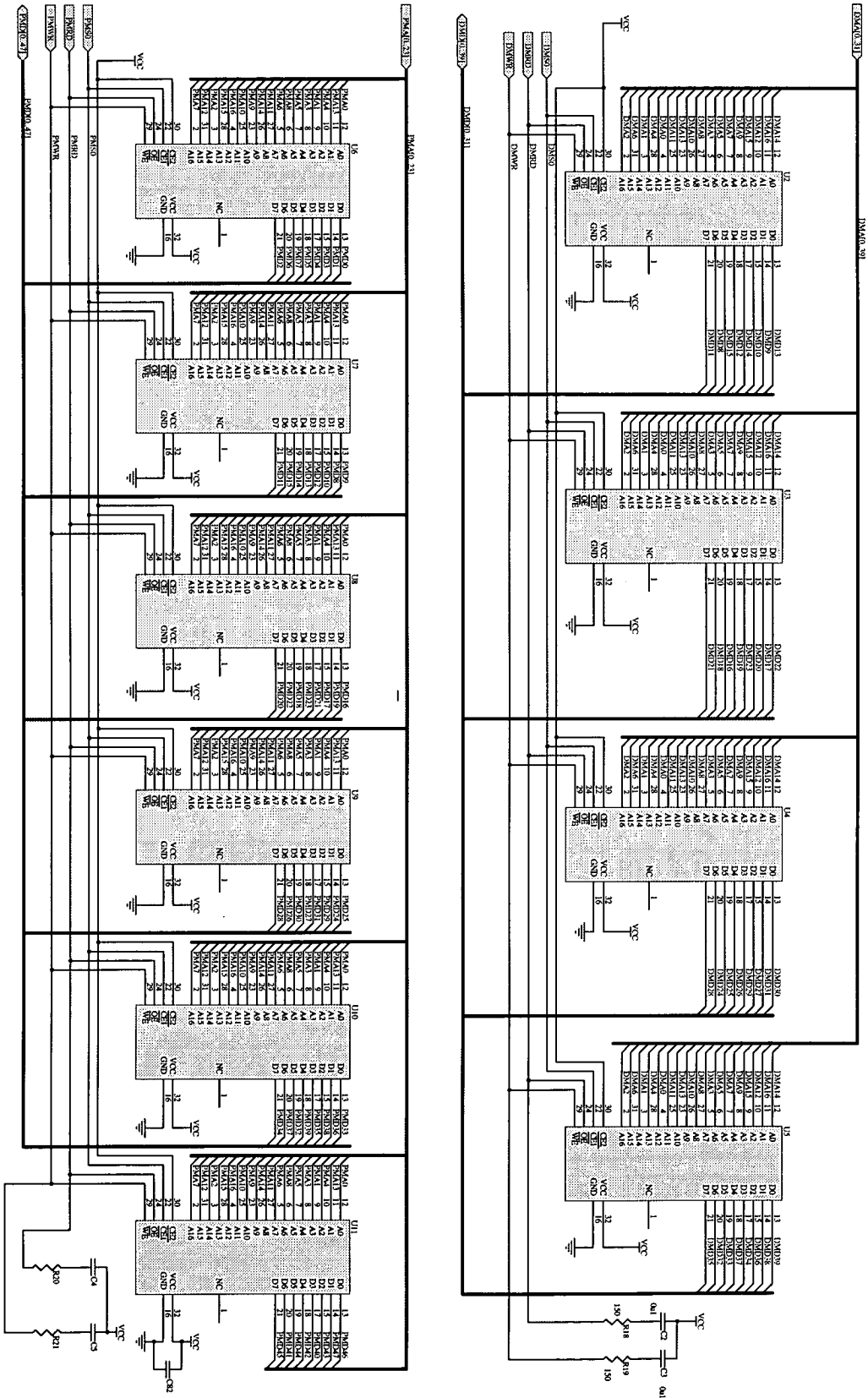


Figure 5.4 DSP Circuit Part 2

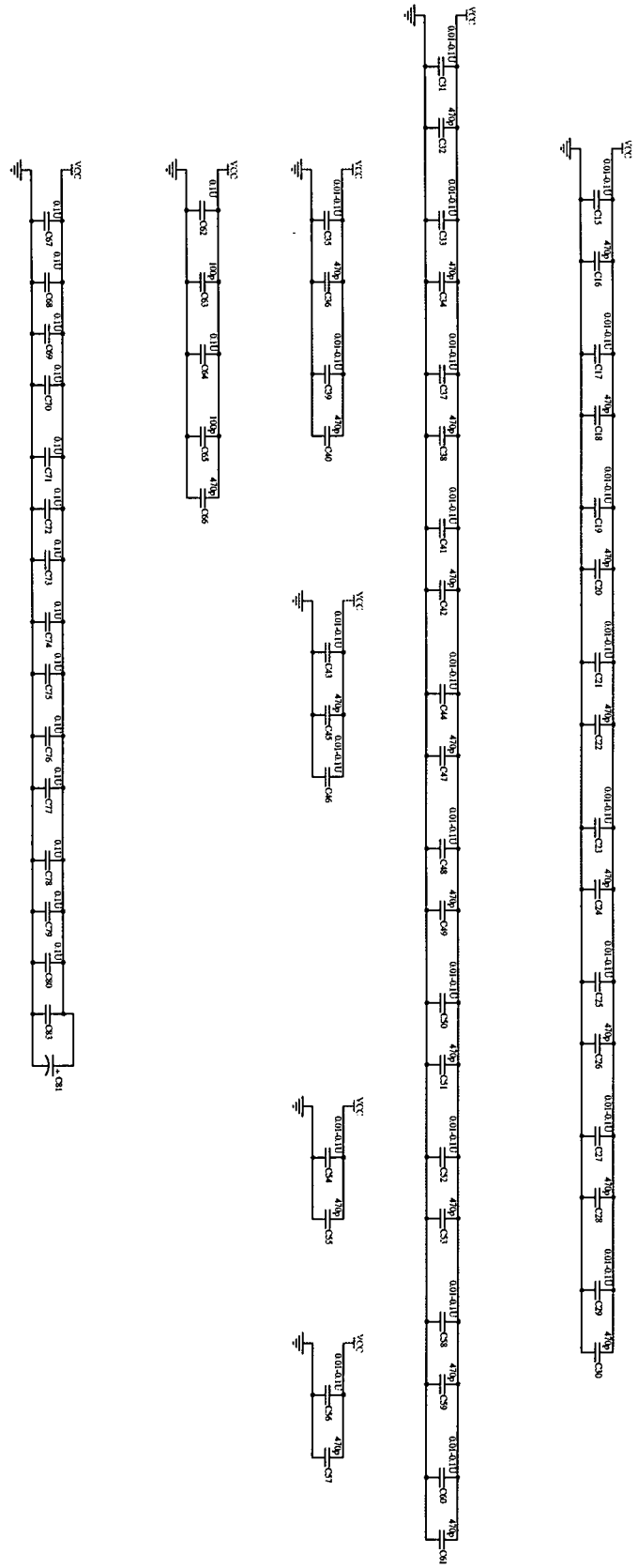


Figure 5.5 DSP Circuit Part 3

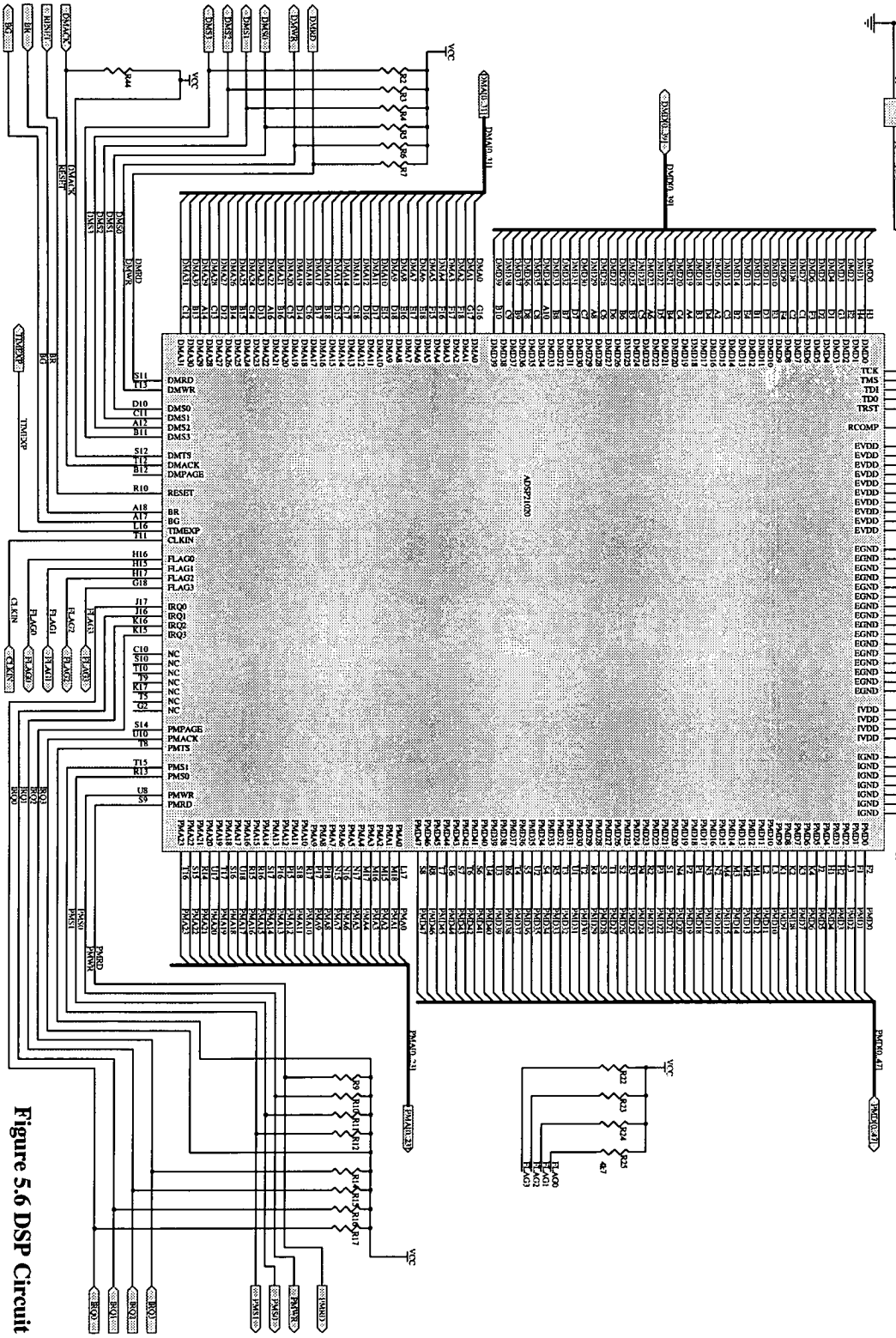


Figure 5.6 DSP Circuit Part 4

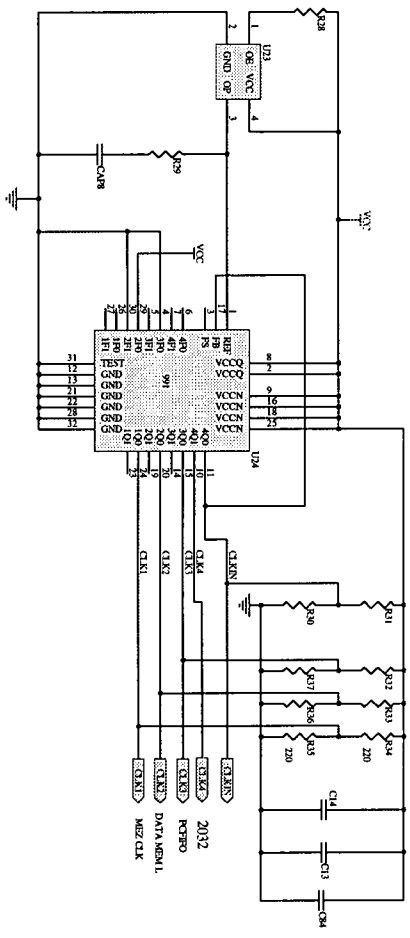


Figure 5.7. DSP Circuit Part 5

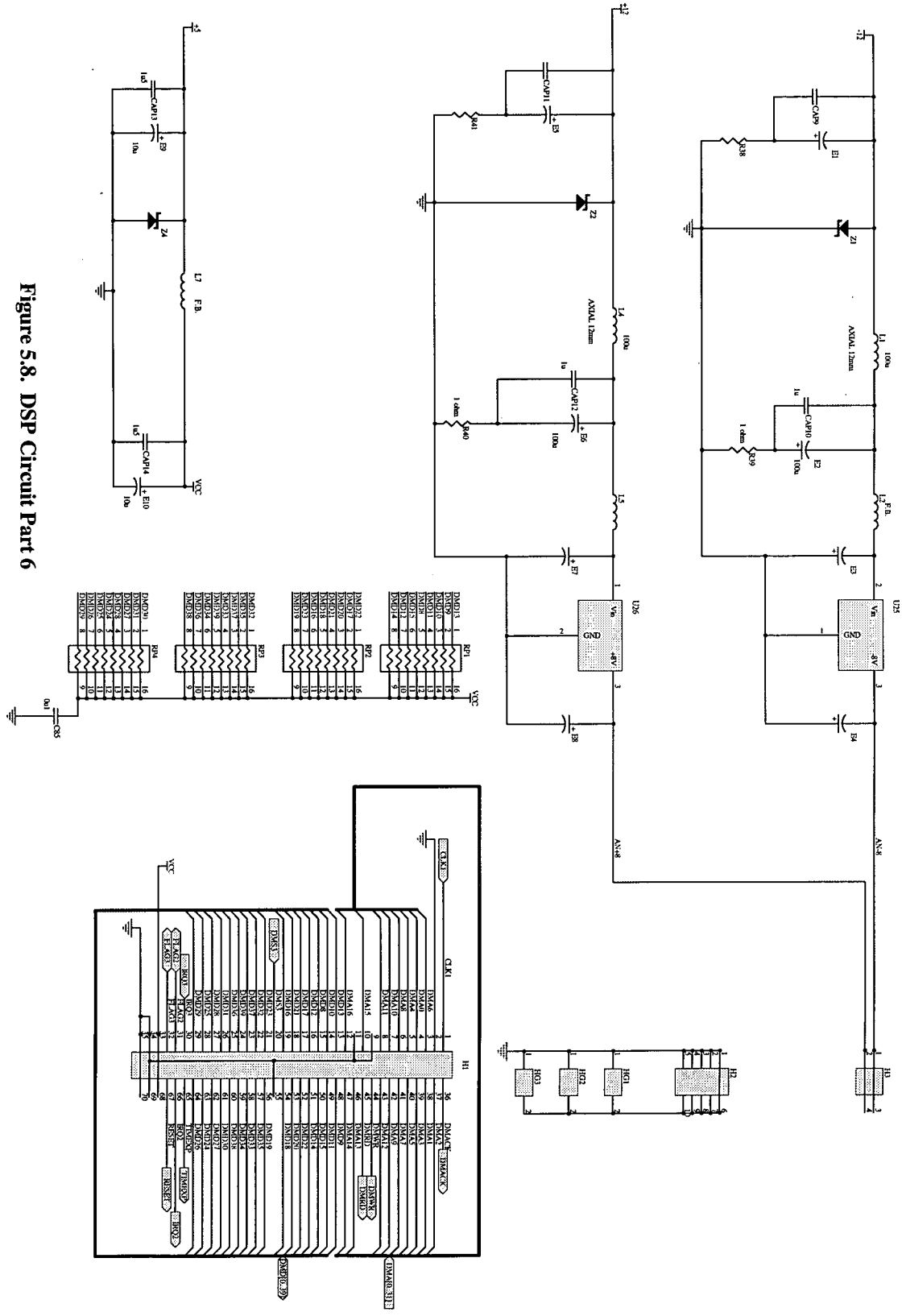


Figure 5.8. DSP Circuit Part 6

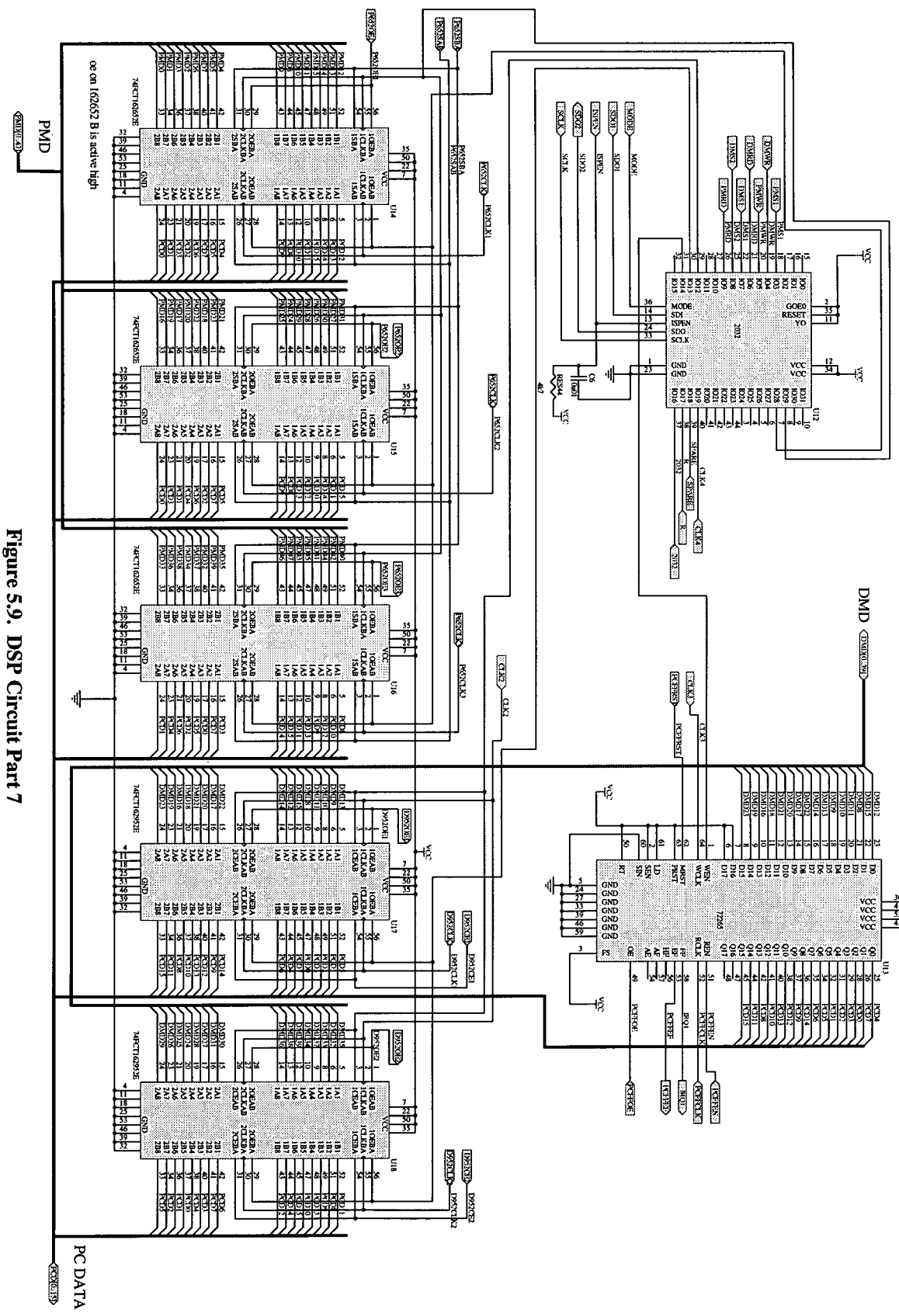


Figure 5.9. DSP Circuit Part 7

5.2.1 DSP Memory

The DSP has 128k of program and data memory as shown in figure 5.10 below.

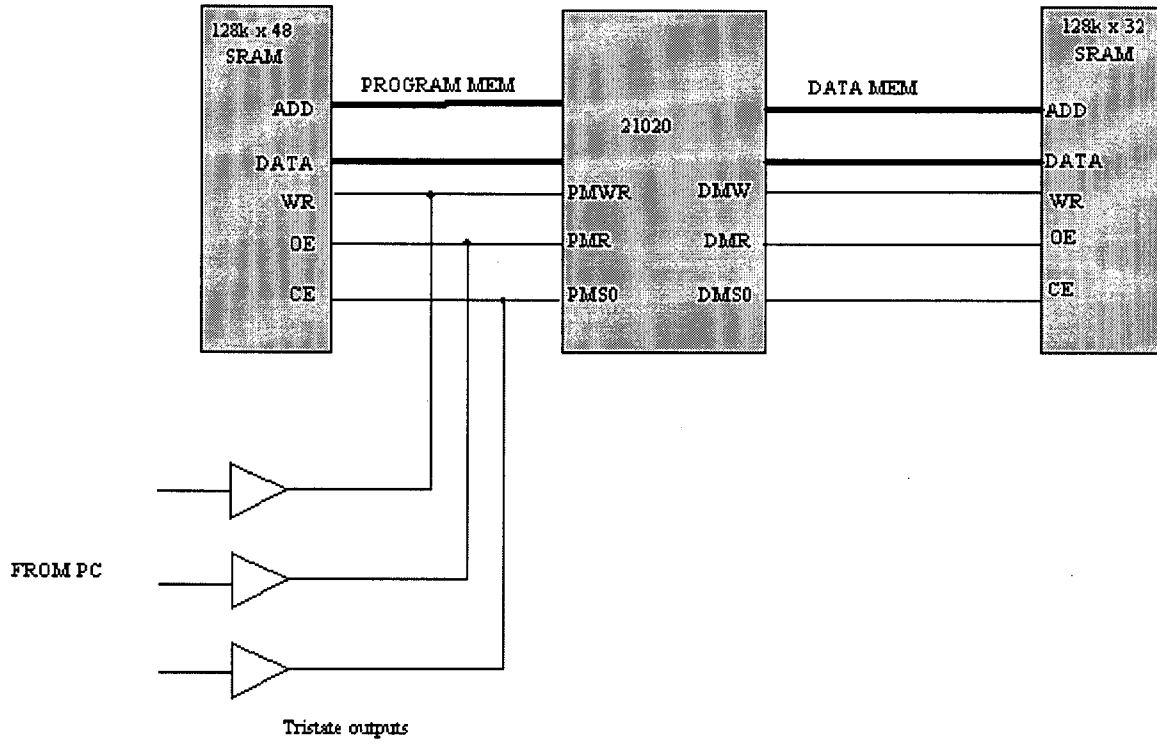


Figure 5.10. Block diagram of DSP RAM usage.

Circuit diagram, figure 5.4, shows how IC U2 to U11, the IDT70124, 128k x 8 RAM ICs are interconnected to form the DSP program and data memory. These RAMs, manufactured by IDT, have 15ns access times. DSP signals, PMS0 and DMS0 are used as chip enables to select the memory block. DSP program memory can be written too and read by the PC. Due to the tight timing constraints the memory

read and write pulses from the DSP cannot be gated, as the introduced timing delays would be unacceptable, see timing diagram, figure 5.11 below.

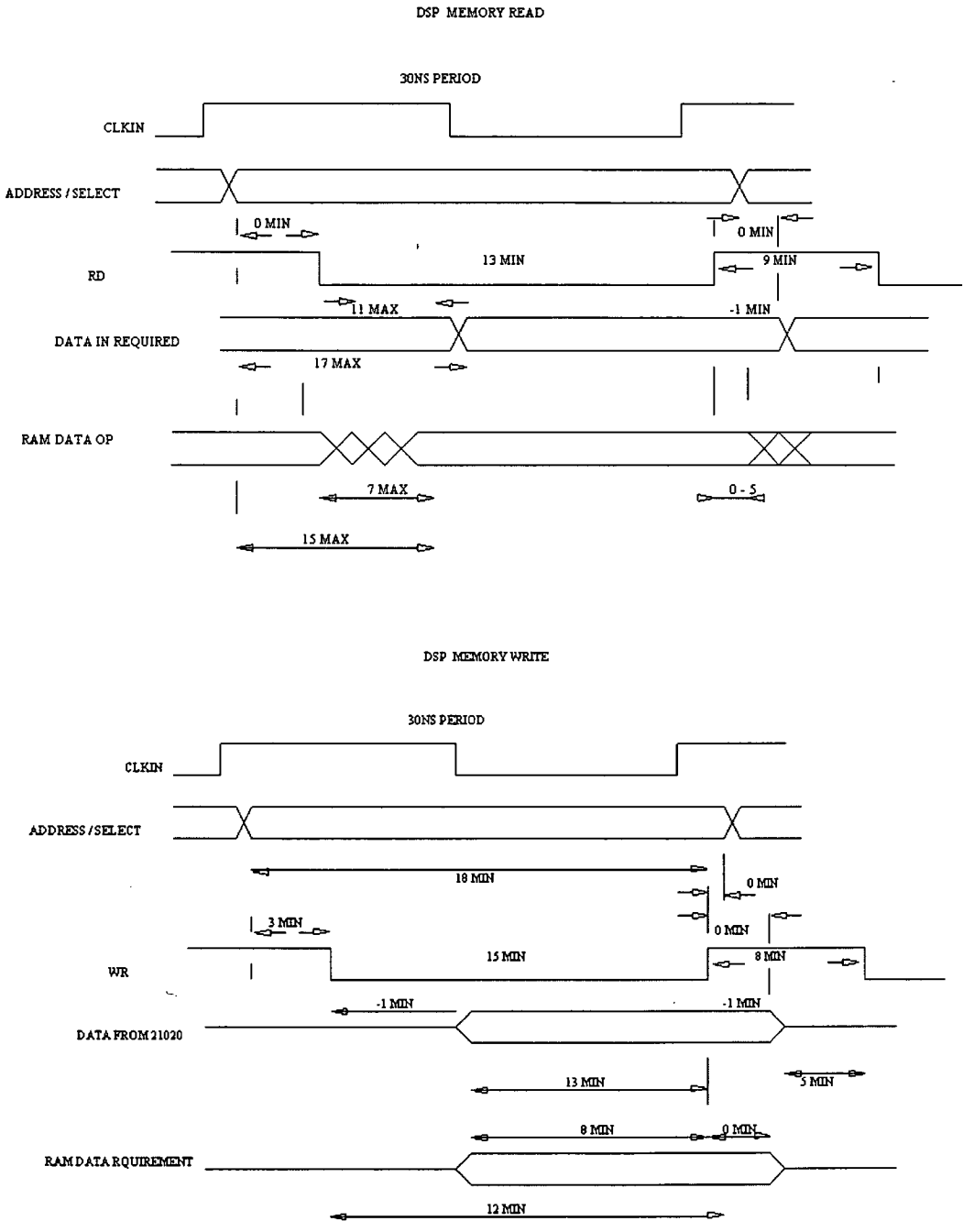


Figure 5.11. DSP memory timing requirements.

Notice that the DSP address hold time after WR high is 0 min. The RAM requires a hold time of 0ns. Therefore the WR pulse cannot be gated. Careful layout will ensure the write to hold time is not violated. Also the write pulse width is 15ns min. The logic can switch in 0 to 5.5ns. The RAM requires a pulse width of 12ns. Therefore if the high to low and visa-versa switching times are not symmetrical the write pulse can possibly be shortened below 12ns. For RAM read operation the RAM read strobe high to RAM output tristated time could be 5ns max and gating the read strobe could take 5.5ns leading to a strobe to RAM tristate time of 10.5ns. The strobe signal could go low again in 9ns. Other devices attached to the data pins could go low impedance in 1ns leading to a short circuit time of 0.5ns. This period although small may cause "spikes" which may in turn cause some error in the circuit. It is important to note that, although these spikes are of short time interval compared to RAM write operations for example, they may cause some RAM operation. The RAM (and other) timings are given to guarantee a particular operation, it does not mean that pulses outside these specifications will not cause a RAM operation. Therefore as the DSP memory bus can be tristated with a bus request signal, BR, the PMWR signal from the PC is sent via a tristatable transceiver, see figure 4.14. The active PMWR, PMS0 and PMRD signal is provided by the device not tristated. The extra load of the tristated gate is acceptable and layout minimises the effects of extra PCB tracks. Notice in the write timing that the data hold after the DSP write strobe is de-asserted states -1 minimum. The Ram requires a hold time of 0ns. The DSP does not change the data but tristates the data pins. The hold times are met as the data is held longer by the RAM chip capacitances. E.g.

For a 2p load at 10μA leakage current the voltage would droop by 0.5v in 0.1 us. 10μA is a typical leakage current and 2p is 1/4 of maximum capacitive load for one RAM pin.

The memory bus control signals have pull up resistors to ensure they are de-asserted when the bus is tristated and are in a defined logic state. The pull-ups are placed at the end of the long tracks. This lowers the load resistance at the end of the track and improves matching and noise immunity. The memory read and write lines are terminated using an AC biased terminating resistor. The 21020 does not have sufficient DC drive to drive the resistor direct (DC coupled). The capacitor is made large to make the RC time constant large compared to the clock period. The resistor is at the approximate impedance of the PCB track.

5.2.2 DSP to PC FIFO

The DSP and PC has a FIFO data interface as shown in the block diagram, figure 5.12, on the next page.

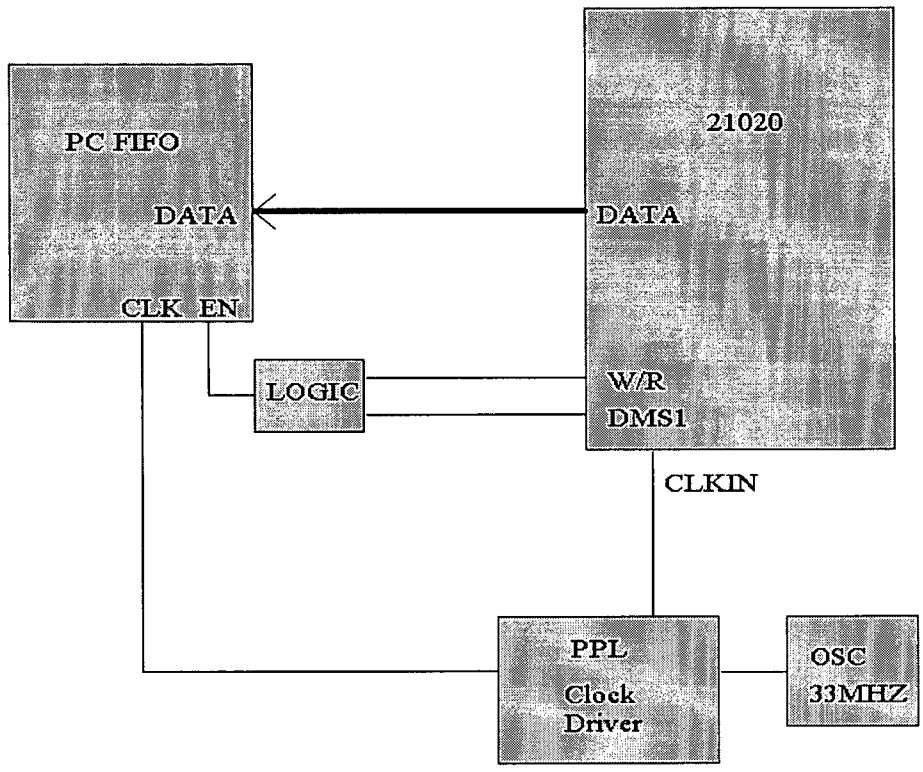


Figure 5.12. PC FIFO system diagram.

To obtain the high speeds required a synchronous FIFO is used. The FIFO is driven in a similar manner to a register. Data is input to the FIFO when the write enable signal is active and the write clock is strobed. Data is output when the read enable is active and the read clock strobed. The FIFO used is the IDT 72265, IC U13 in figure 5.8. The FIFO is 32k deep by 18 bits wide chosen for its speed and depth. It is written too by the DSP and read by the PC. (See 72265 data sheet for detailed information). Figure 5.13 on the next page shows the FIFO timings required.

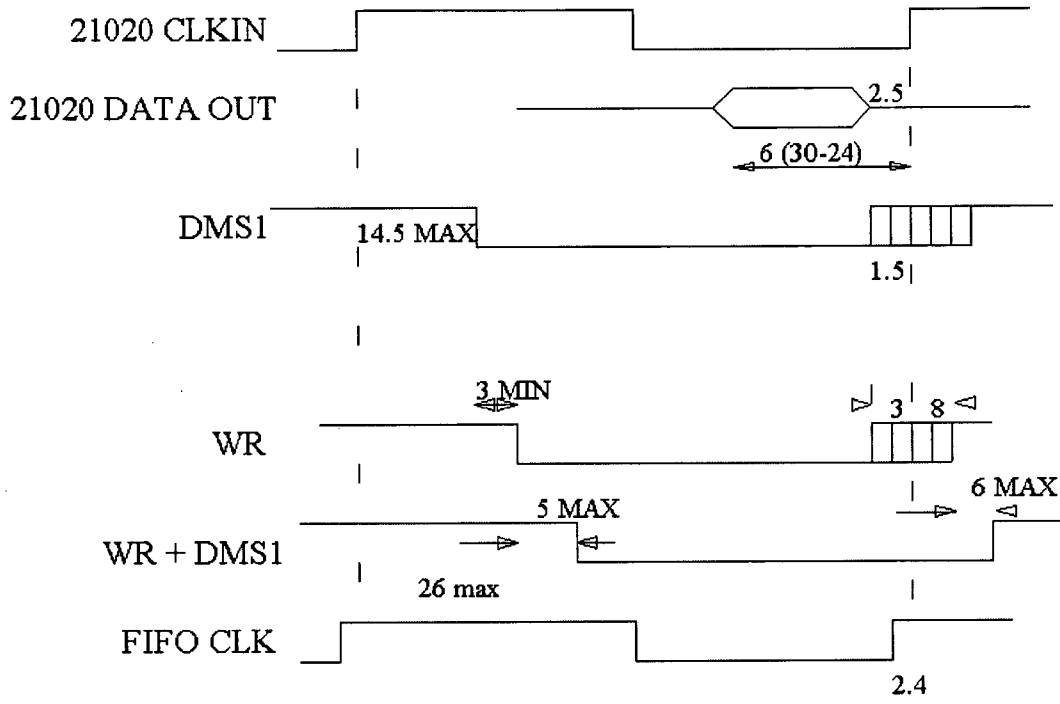


Figure 5.13. PC FIFO timing requirements.

The FIFO write clock input is driven by a time shifted copy of the DSP clock, see section 5.2.3. (The DSP clock timing is not on the 21020 data sheet but can be obtained from Analog devices on request. The timings used cannot be derived by summing other timings as stated by Analog Devices). The FIFO write enable, WEN, signal is derived from DMS1 and DMWR by the 2032 in circuit programmable logic device, IC U12. See timing waveform WR + DMS1 in figure 5.13 above. The PC reads the PC FIFO and drives the FIFO output. The FIFO read enable signal, PCFFEN, is derived from the decoded ISA bus IO address. The FIFO read clock is derived from the decoded ISA bus IO address and the ISA bus IO read signal, IOR. The FIFO output enable is derived from the decoded ISA bus IO address and the IO read signal. The timing for the PC read operations are relatively slow and are

generated within the 1048 ISP logic devise, U22. The FIFO read and write functions are independent thus the DSP does not have to wait while the PC accesses the FIFO data. The FIFO Full flag is connected to the DSP interrupt 4, IRQ4, to signal that the PC is not emptying the FIFO fast enough. The DSP may then, if necessary, signal the PC to let it know that it is not keeping up. The FIFO empty flag is used as a status signal to the PC to let the PC know if the FIFO is empty.

5.2.3 DSP system clocks

Figure 5.6 shows the DSP clock generator circuit. An oscillator IC, U23, generates the DSP system clock frequency. It provides the reference for the clock driver IC, U24. This IC is a phase locked loop system and can output multiple clocks with varying phase differences. Four clocks are generated, for the PC FIFO, data memory latch, 2032 ISP device and piggyback board. The production of multiple clocks is useful for clock distribution and signal timing. The outputs can be 50 ohm terminated for transmission line driving to ensure a "clean" clock.

5.2.4 DSP bus latches and buffers.

The DSP can communicate with the PC via the program memory RAM, the PC FIFO and transceivers. Transceivers are utilised as these devices can be configured as latches or buffers, have high drive capability, are 16 bit wide, have high density packages and are high

speed. The DSP section is linked to the ISA bus via a set of transceivers.

Figure 4.5 above shows the transceiver interface system. Circuit diagram, figure 5.8, shows how the transceivers are interconnected. IC U21, an IDT74FCT162646 drives the PC ISA bus. This transceiver is 16 bits wide, can pass data in either direction, can be registered or unregistered, can be tristated and has a high power symmetrical output drive suitable for high speed busses. Figure 5.14 below gives a functional diagram of the transceiver. The 74FCT162646 data sheet gives detailed information.

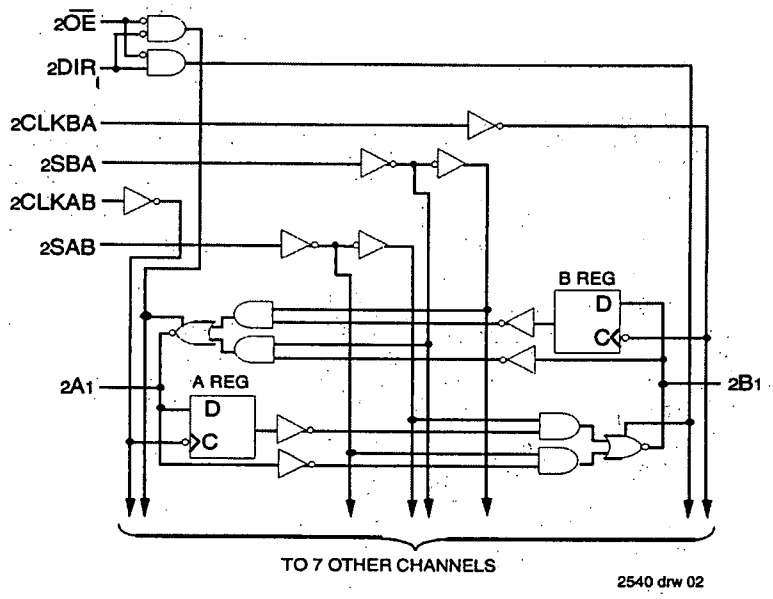


Figure 5.14. Functional diagram of the 74FCT162646 transceiver.

The Direction and output enable of this transceiver is supplied by decoding the ISA bus IO address. The buffered ISA bus data lines are routed to: the 1048 ISP logic device, IC U22, three 162652

transceivers, IC U14 to U16, two 162952 transceivers, IC U17 and U18 and the PC FIFO, ICU13, output data pins.

The three 162652 transceivers are connected to the DSP 48 bit wide program memory data bus. These transceivers are very high speed and can keep pace with the DSP. The 62652 transceiver is similar to the 162646 but has independent output enable, OE, signals for each data direction and no direction signal, see figure 5.15 below.

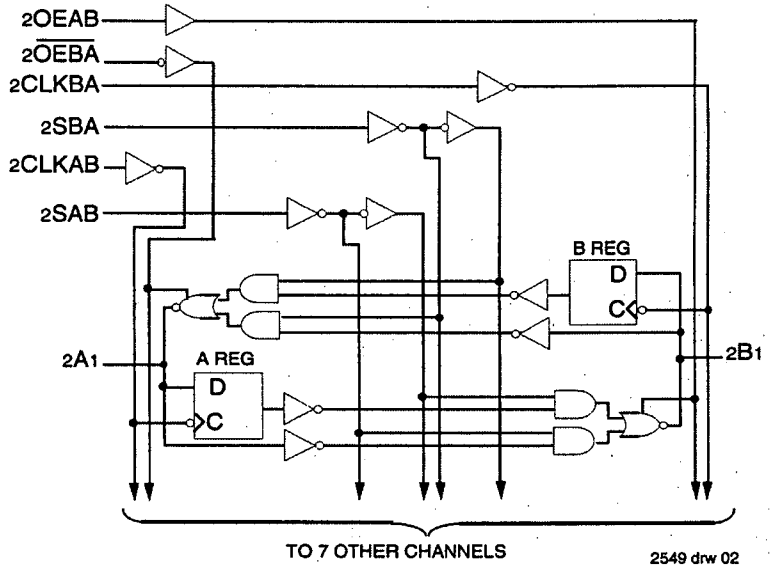


Figure 5.15. Functional diagram of the 74FCT162652 transceiver.

The two 162952 transceivers are connected to the upper 32 bits of the DSP 40 bit wide data memory data bus. These transceivers are very high speed and can keep pace with the DSP. The 62952 transceiver is a latch with latch enables, see figure 5.16 on the next page.

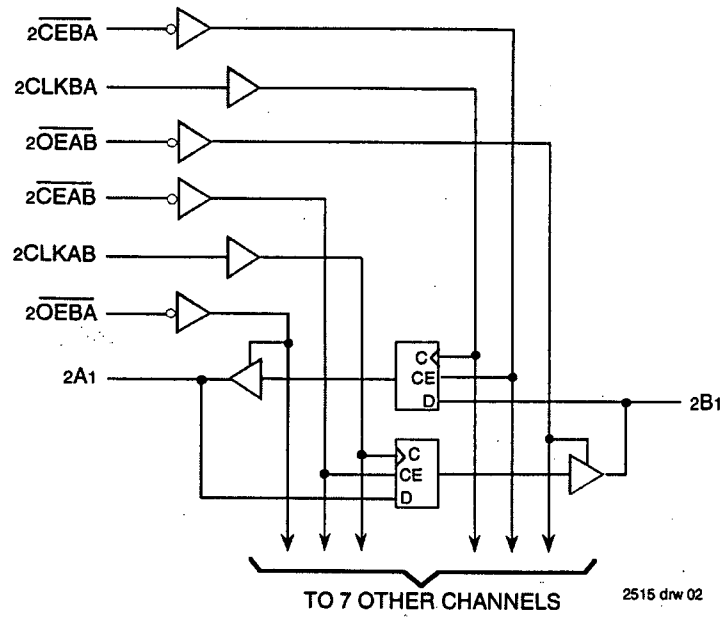


Figure 5.16. Functional diagram of the 74FCT162952 transceiver.

The transceiver functions are controlled by the 1048 and DSP 2032 ISP devices as shown in figure 5.17 on the next page and noted in section 4.3. A more detailed description of the ISP to transceiver interconnect will be given in section 5.2.5.

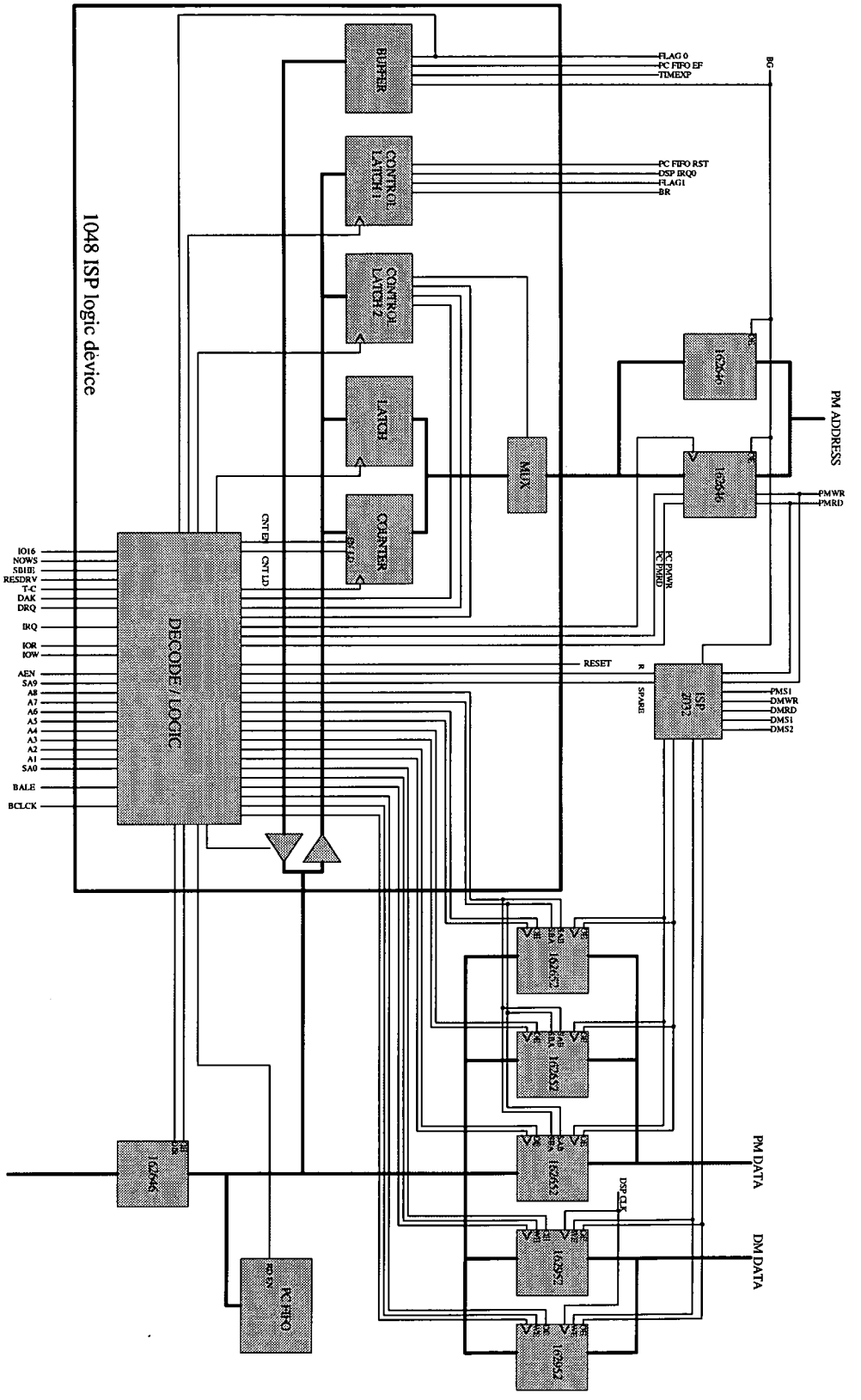


Figure 5.17 Block diagram of ISP logic interconect

The DSP to transceiver high speed link is shown in the block diagram, figure 5.18 below.

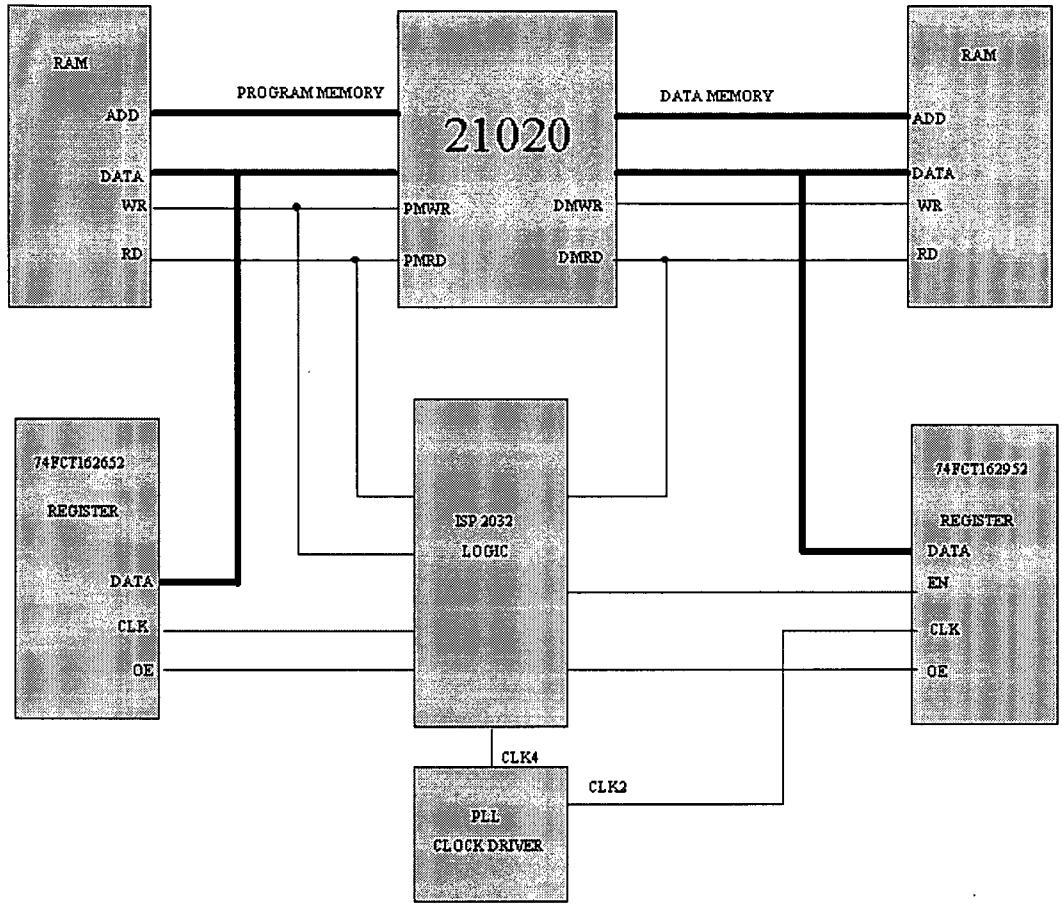


Figure 5.18. DSP to transceiver block diagram.

Using the transceivers the DSP can communicate with the PC while running at full speed and have data and code written to its program memory by the PC.

The operations involving the transceivers are as follows:

- DSP data memory bus read of 162952. The OE is formed by gating DMS2 and DMRD in the 2032 ISP. The timing diagram is shown in figure 5.19 below.

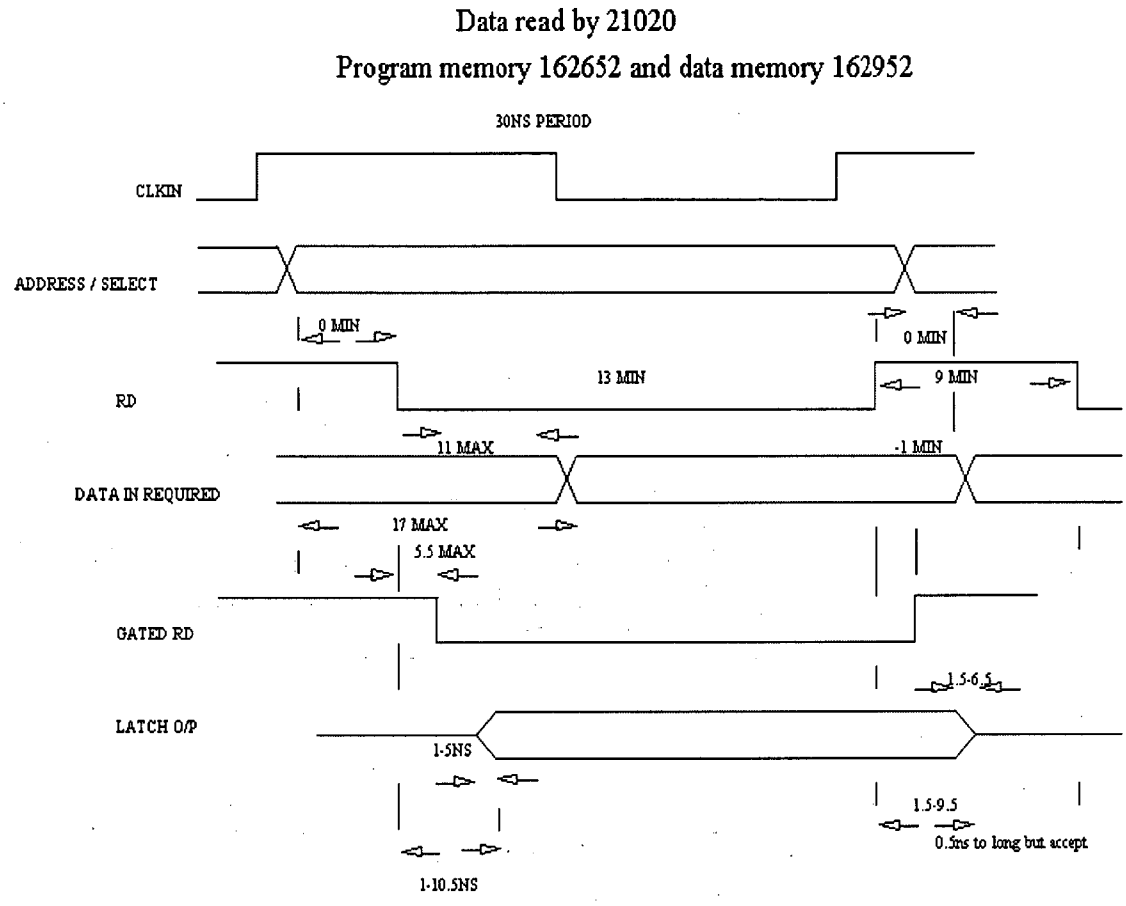


Figure 5.19. Read of 162952 transceiver by 21020.

- DSP data memory bus write to 162952. The latch enable signal is formed by gating data memory write strobe, DMWR and data memory select 2, DMS2. The latch is clocked by a phase changed copy of the system clock. This ensures that the set up and hold times are guaranteed. See timing diagram figure 5.20 on the next page.

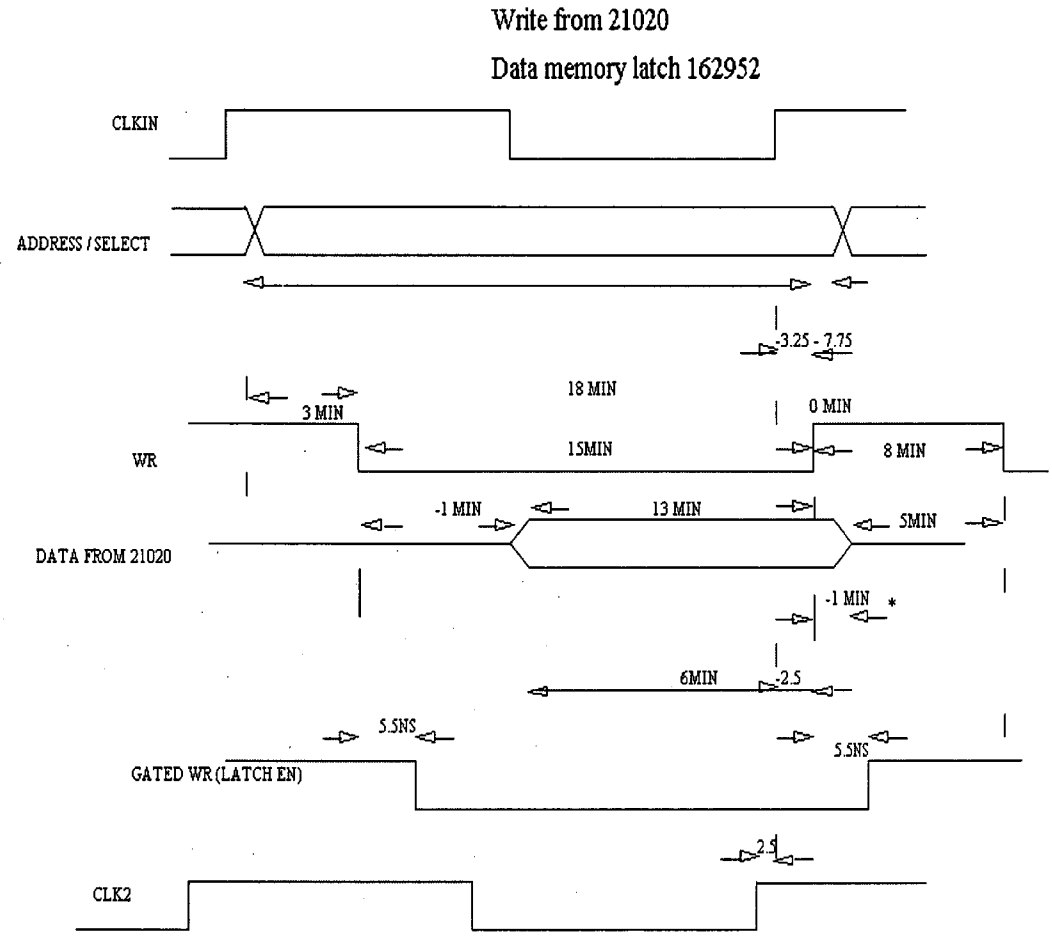


Figure 5.20. Write to data memory 162952 by 21020.

- DSP program memory bus write to 162652. The Write pulse was produced by gating, program memory select 1, PMS1, and program memory write, PMWR. The timing diagram is shown in figure 5.21 on the next page.

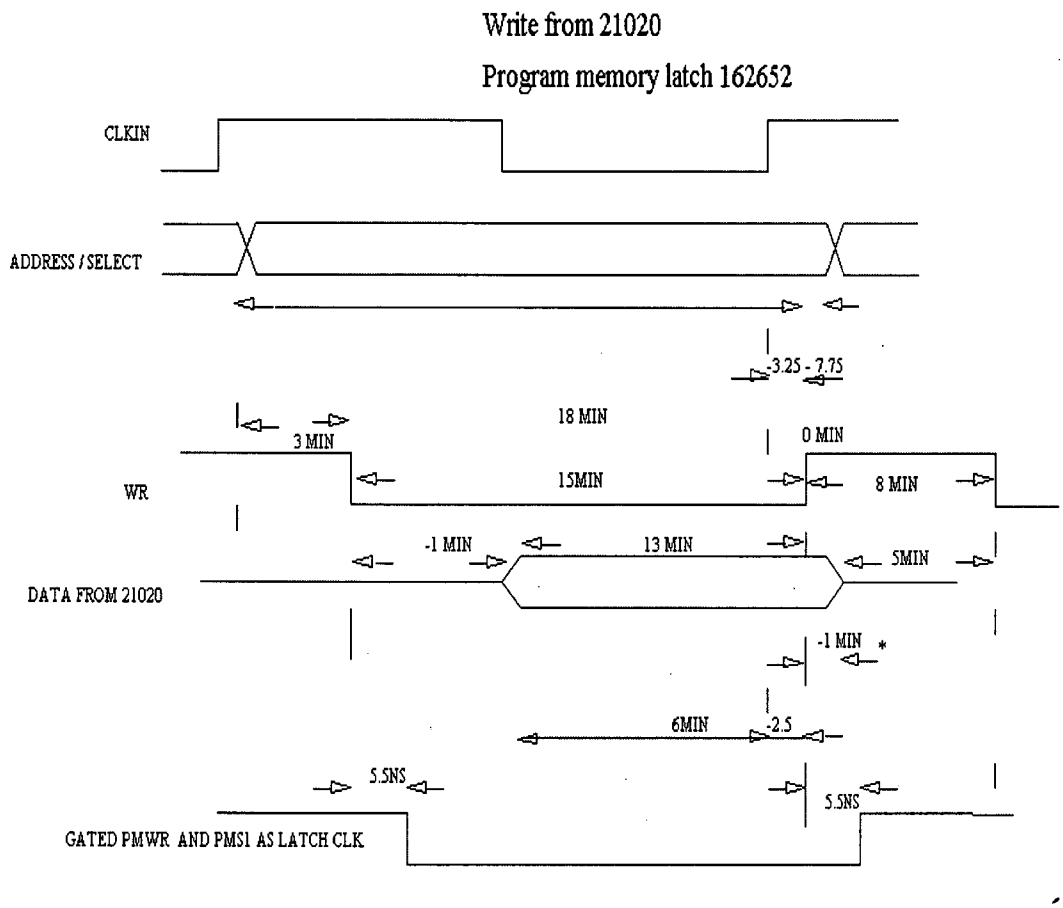


Figure 5.21. Write to data memory transceiver 162652 by 21020.

Notice that the data hold time may be 6.5ns worse case into the 21020 data output tristated period. The data hold time is satisfied by the decay time of the data lines being long enough. Throughout the design it must be noted that 21020 data sheet timings are given for specific loads. These loads are not known until the circuit is built. This leads to some uncertainty at the design stage and the designer must make a worse case estimate. 162652 are used rather than 162952 because of

their non-registered transmission option allowing direct read from program memory by the PC.

- PC read from and write to DSP program memory. The PC first gains control of the DSP memory busses by sending a BR signal. The DSP responds by tristating its memory bus pins and asserts BG. On receipt of the BG signal the PC downloads a PMS0 signal and upper address bit, bit16, to transceiver U19, via the ISP1048 (see discussion in 5.2.3), were it is latched. The lower 16 bit of the memory address, bit 0 to 15, are passed to the program memory address bus, via the ISP1048, to transceiver U20. The 48 bit data is written, 16 bits at a time, to 3 transceiver latches, U14, U15 and U16, connected to the program memory data lines. A write strobe is provided via the ISA bus. Reading the RAM is similar, except the 48-bit word is provided from the RAM.

5.2.5 DSP glue logic

Figure 5.17, page 79, shows a block diagram of the ISP logic device interconnect. The 1048 acts as an interface to the ISA bus control signals. Its functions are:

- Decode the ISA bus IO addresses
- Buffer control signals from the DSP and PC FIFO
- Control transceiver operation
- Latch DSP board control signal
- Provide program memory address. The lower 16 bit of the program memory address are latched or may be provided by a counter. (This

option is not implemented but has space allocated for future requirements).

The ISP2032 provides the high speed logic required by the DSP related operations.

5.3 The DSP PCB

The DSP board is laid out as a high-speed digital board. It is an 8-layer board including a +5v power and ground plane. Power and ground to the ICs are considered first. Good signal routing is of no use to a poorly supplied high-speed device. Power is connected from the power planes to the IC by large vias placed close to the IC power pins to reduce impedance. Next the high speed signals are laid out. The high-speed digital signals are considered transmission lines. The PCB tracks and power planes form microstrip and stripline transmission lines, see figure 5.22 below.

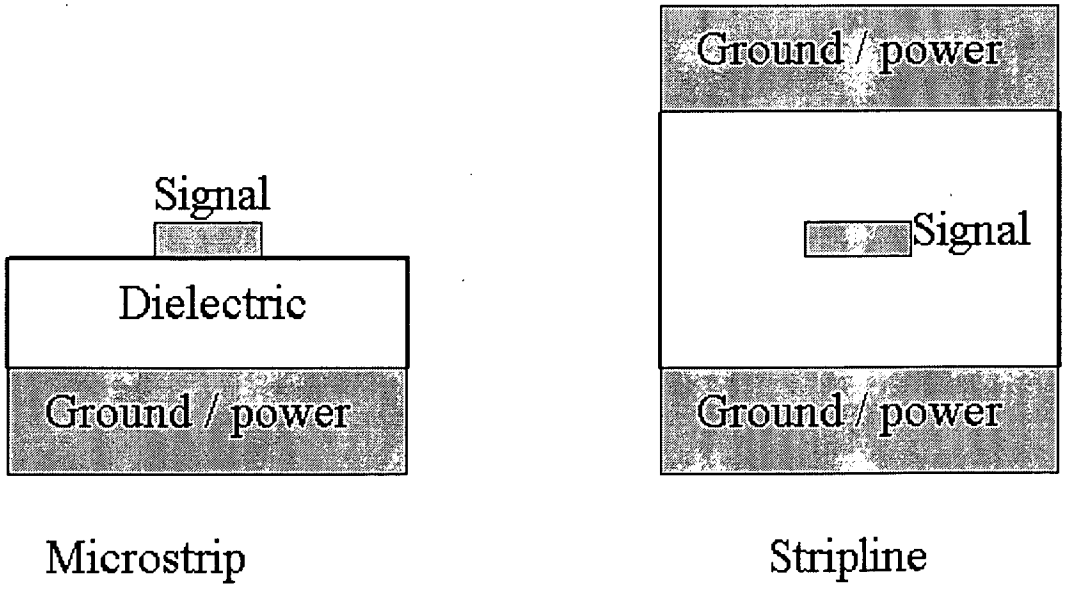


Figure 5.22. Cross section of PCB transmission line.

The impedance of the transmission line can be calculated from the dielectric thickness, thickness of the signal track copper and width of the signal track. The dielectric and copper thickness' can be supplied by the PCB manufacturer. Figure 5.23 below shows graphs of transmission line impedance and capacitance.

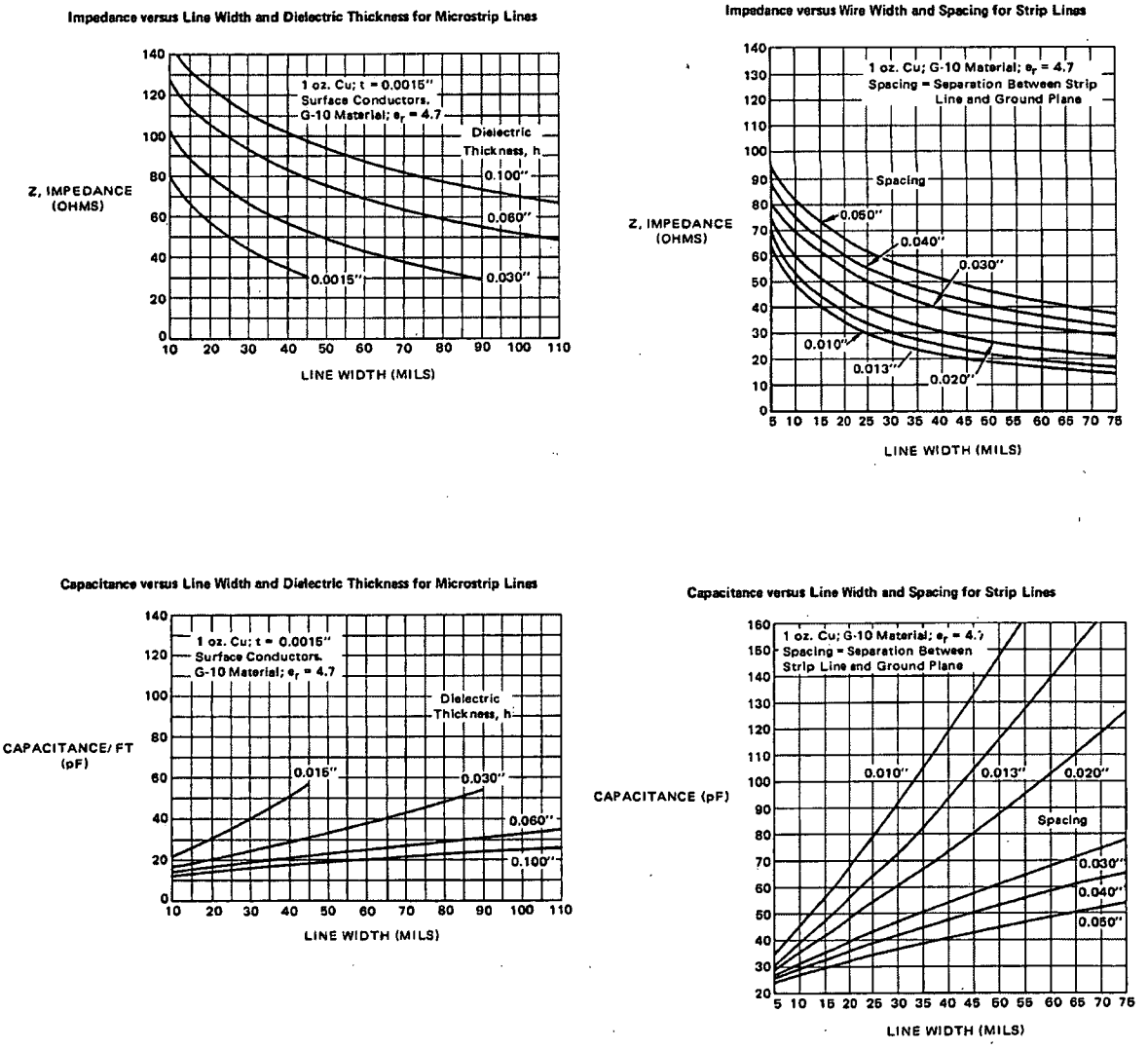


Figure 5.23. An excerpt from the 'MECL system design handbook' by Motorola.

The impedance does not need to be known exactly as the signal reflections need to be minimised not expunged. Graphical techniques are adequate for this. See figure 5.23 above. The narrower the track widths the higher the transmission line impedance. Standard manufacturing techniques for PCBs limit the minimum track widths to 8mil. (Thinner tracks are available but at increased cost). This leads to impedances of the order of 100 ohms. Long lines are terminated at the end of the line were possible to reduce reflections. The transmission lines are continuous and do not change PCB layer. This reduces reflections caused by discontinuities in the tracks. Some reflections are inevitable due to discontinuities but are minimised by careful routing. When a signal is routed to many loads, one continuous transmission line is employed. The loads are connected to the transmission line by short stubs, see figure 5.24 below.

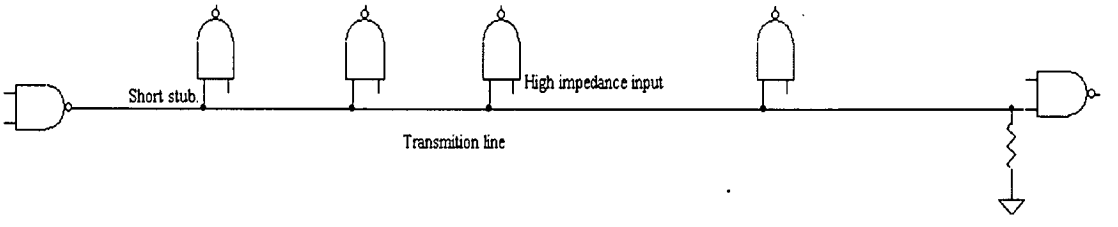


Figure 5.24. Example of multi-load transmission line.

Where possible vias are used only at the source or load point of a transmission line. When placing decoupling capacitors the path of the supply current needs to be considered. Surface mount decoupling

capacitors are placed under or close to the IC they are decoupling. The capacitor positioning is important and is shown in figure 5.25 below.

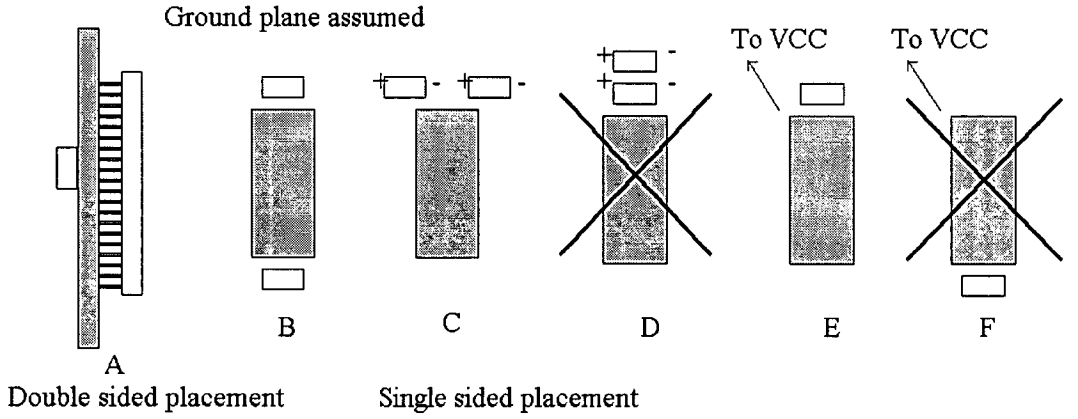


Figure 5.25. Decoupling capacitor positioning.

The preferred placing is on the reverse side of the board under the IC. Figure 5.25A. For single sided placement the capacitor should be placed at the power entry point. Figure 5.25E and F. When using capacitors of different values and types in close proximity to one another it is possible for the two capacitors to interact with one another. With different frequency responses, the two capacitors will respond with different timings to voltage spikes with a result being an instantaneous difference in internal voltages between the two capacitors. The two capacitors can then begin oscillating as they pass charge between each other. Capacitors of different values and types must not be placed in close proximity to each other. Figure 5.25C and D. After the transmission lines are laid out and the decoupling capacitors placed the slower ISA bus signals are routed. Transmission line effects are ignored for these slower signals. The area under the piggyback board is kept free of digital signals were possible to eliminate cross-talk to the piggyback board which may contain low

noise analogue signals. The DSP PCB file is on the attached floppy and can be viewed using Protel PCB design software.

5.4 DSP to piggyback board interface.

The piggyback board is mounted onto the main DSP board by square pin header connectors. See figure 5.26 below.

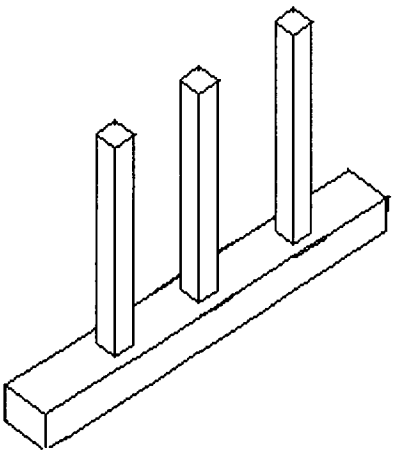


Figure 5.26. Header pins.

These connectors are not designed for high-speed transmission line connections. If placed mid way along a transmission line they will cause a large discontinuity and are therefore placed at the end of the transmission line. The headers were chosen for their ease of purchase and cost. Although not ideal, by keeping the PCB track distances from the header pins on the piggyback board to the connecting components short (less than 25mm) this connector will suffice. (A similar

technique is used for the PC PCI bus connectors). True transmission line connectors are very expensive and difficult to obtain. Pull up resistors on the control strobes are also placed close to this connector to form a crude transmission line terminator.
The board is supplied with a digital +5v power supply and a +8v and -8v supply derived from the ISA bus +12v and - 12v supply. These 12v supplies are noisy and are therefore filtered and regulated to 8v.

5.5 The A to D Converter

Two A to D converters were designed. The second was designed after testing of the first revealed some unexpected results. These results will be discussed in chapter 6.

5.5.1 A to D anti aliasing filter

The PCB real estate available and the type of signal to be filtered dictates the characteristics of the filter chosen. For the ice radar project, were the signal is a pulse, a filter with good transient behaviour is required to minimise distortion of the pulse. A Bessel filter is selected for its linear phase characteristics. A 5 pole Bessel filter would give 50dB attenuation at 50MHz satisfying the Nyquist requirements for an 8 bit A to D. The 3dB bandwidth would be 10Mhz, wide enough to retain the fidelity of a 200ns pulse and pass a recognisable 100ns pulse. (Refer to Information, Transmission, Modulation and noise by

Schwartz). An active filter was chosen to reduce the PCB space required. Figure 5.27 below shows a 5 pole anti-aliasing filter.

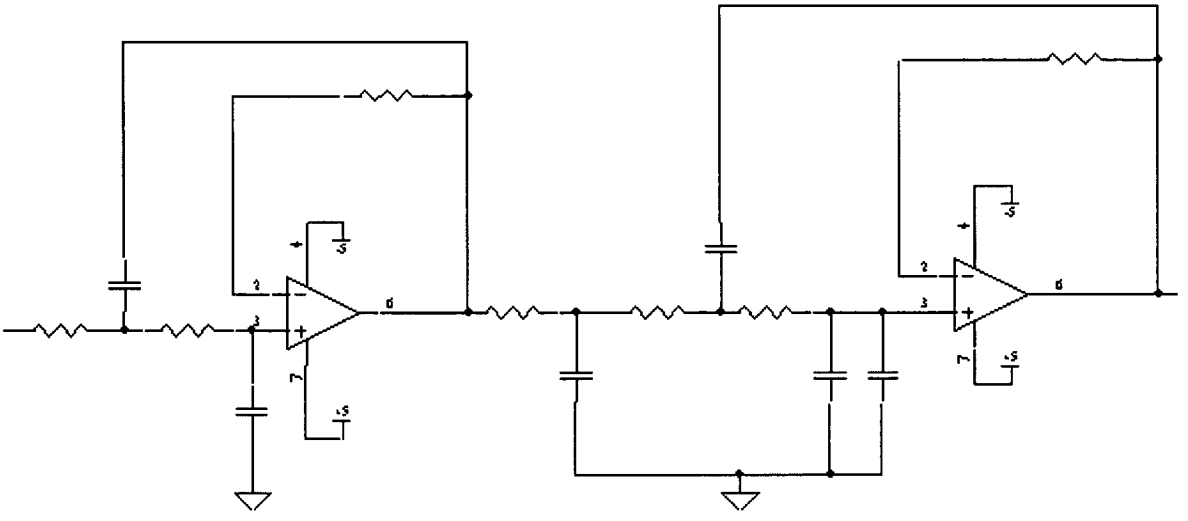


Figure 5.27. Active 5 pole filter.

High speed OP amps were required. Although several Opamps exist with gain bandwidths of the order of 50 to 75MHz this is not necessarily sufficient for our filter. The characteristics required are:

- High gain at the highest frequency of interest
- Low phase shifts at highest frequency of interest
- High input impedance at the highest frequency of interest
- Low output impedance at the highest frequency of interest
- Fast transient response
- Low drift

The analogue 5th order Bessel input filter stage, input amplifier and A to D driver was implemented using the AD8001 current feedback amplifier, chosen for their high bandwidth, low distortion, high drive

capability and fast transient response, see AD8001 data sheet. The filter capacitors are npo ceramic capacitors suitable for high frequency circuits with sufficient temperature and long term stability. Figure 5.28, page 93, shows the anti aliasing filter. The layout is particularly critical as discussed in the A to D PCB section 5.5.3.

5.5.2 A to D 1

Figure 4.11, on page 51, gives a block diagram of this A to D system and the schematic diagram is shown in figures 5.28 to 5.31 on pages 93 to 96. This design is based around the Analog devices 9058 dual A to D. The 9058 was chosen as the dual A to D and onboard voltage reference in one IC package simplified the design. The +8v and -8v analogue power supplies are regulated down to 5v for the A to D. The two levels of regulation, from 12v to 8v on the DSP board followed by 8v to 5 volt on the A to D board, provide a clean analogue power supply. Care is taken to isolate the A to D digital and analogue power supplies. Decoupling for the A to D is a priority and the decoupling capacitors are placed as close to the A to D IC power supply pins as possible. Three npo type capacitor values are chosen to decouple the IC 0.1μF, 1nF and 100pF to ensure noise at all frequencies is suppressed. The A to D inputs need to be driven by a low distortion amplifier with high drive capability. The Analog Devices AD8001 current feedback amplifier was chosen to drive the A to D input (as for the analogue 5^{th} order Bessel input anti aliasing filter stage and board input amplifier). The feedback resistor for the AD8001 OPamps is place directly under the IC and connected directly to the IC pins to ensure that there are no

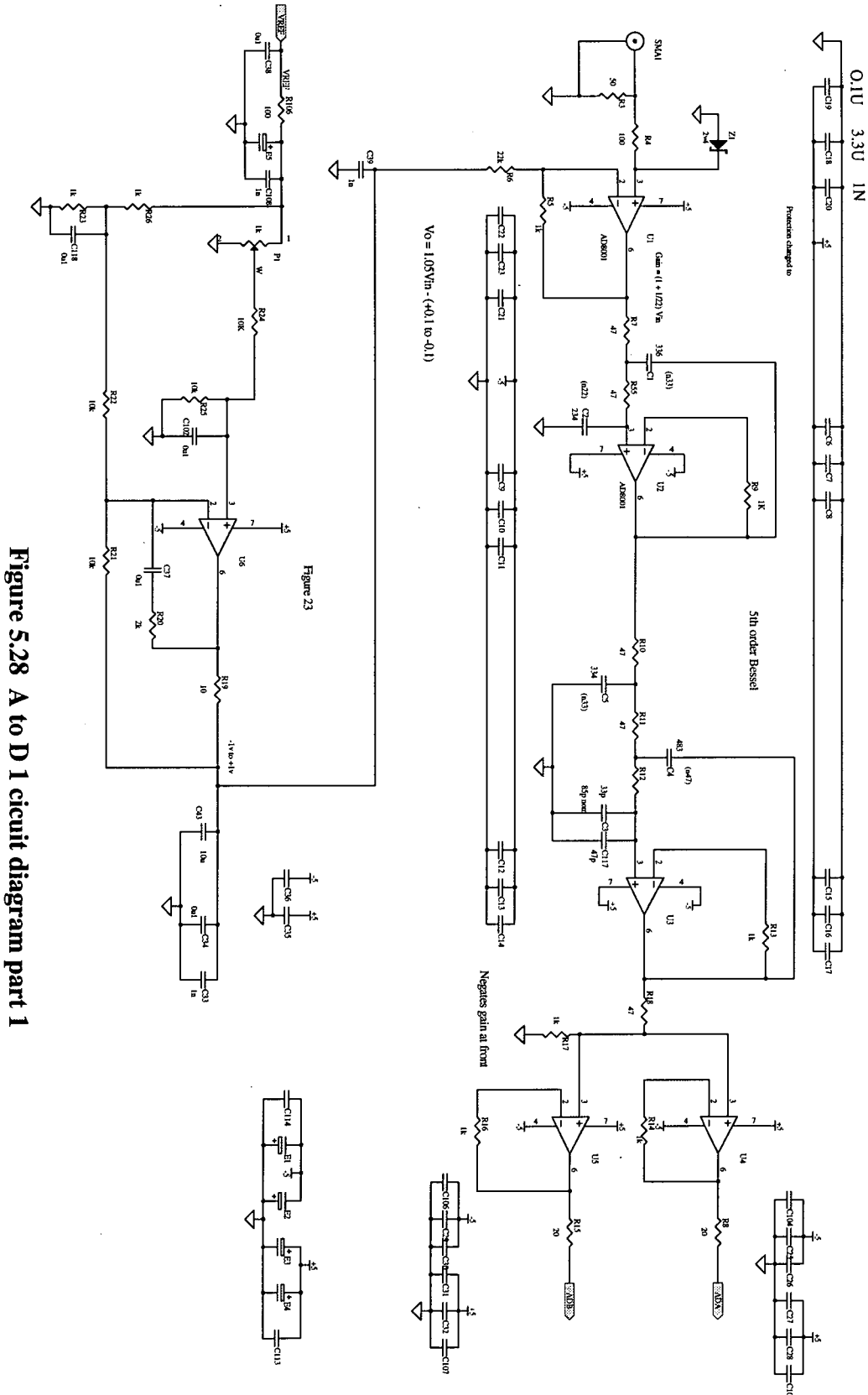


Figure 5.28 A to D 1 cicuit diagram part 1

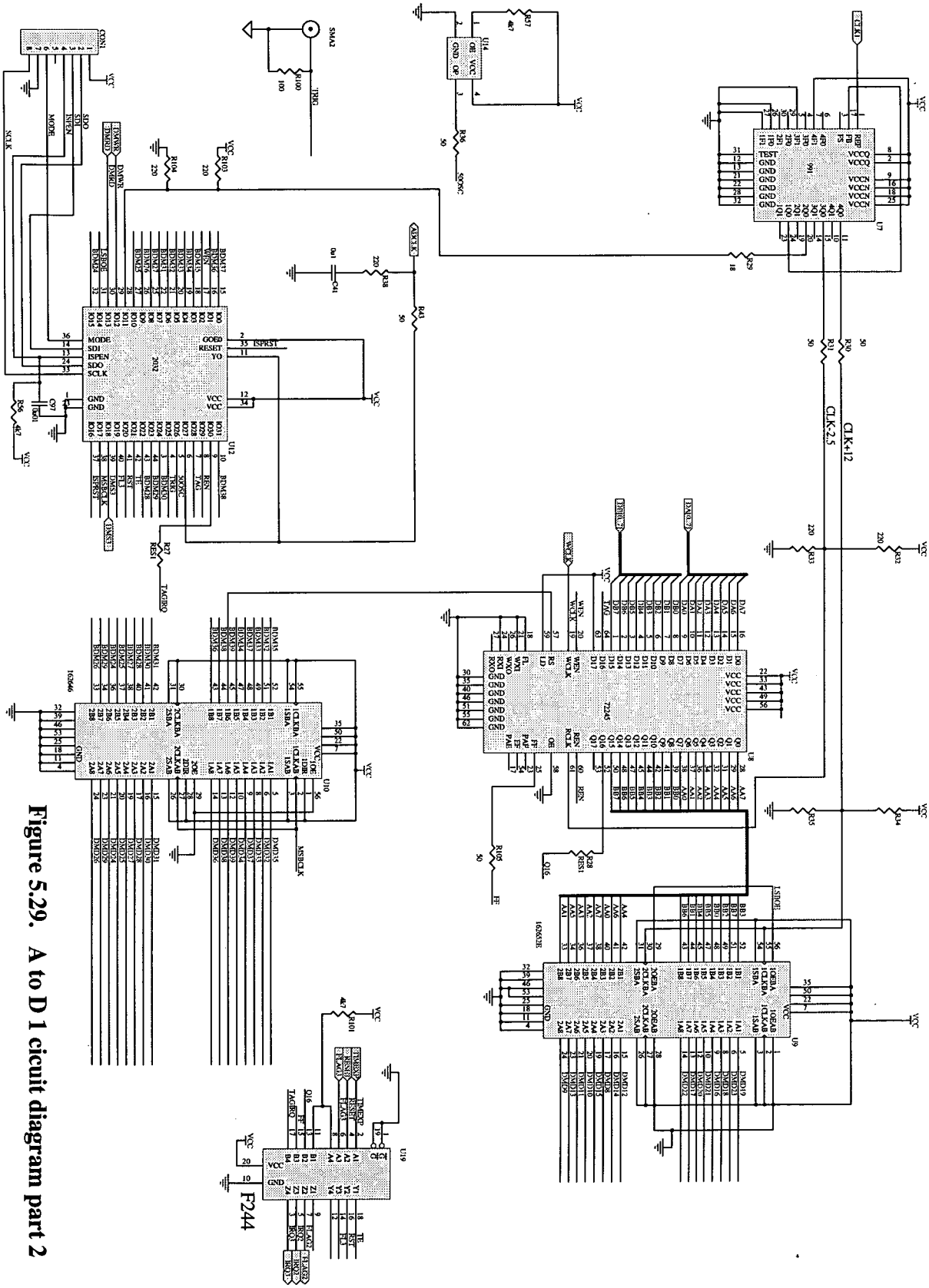


Figure 5.29. A to D 1 cicuit diagram part 2

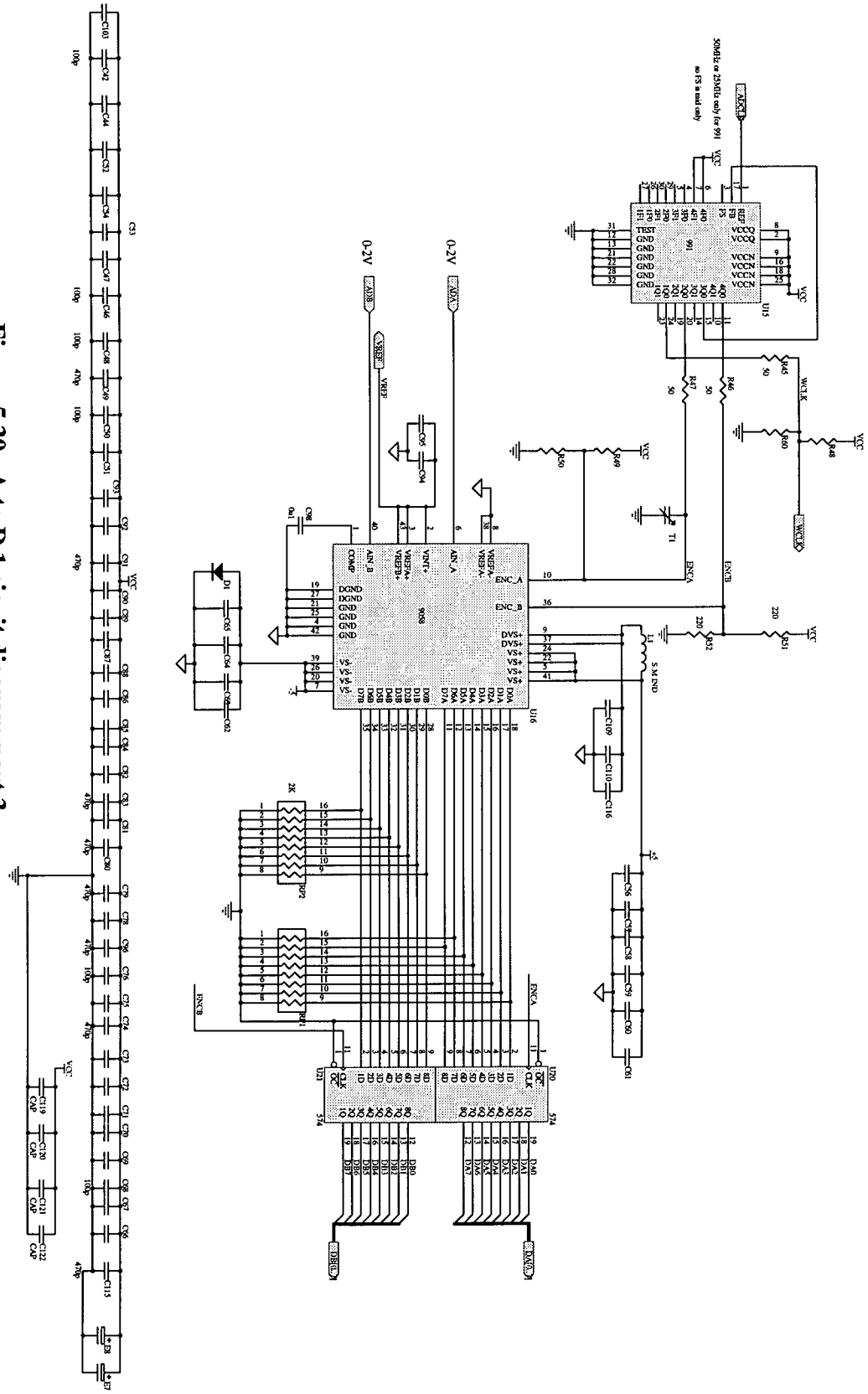


Figure 5.30. A to D 1 cicuit diagram part 3

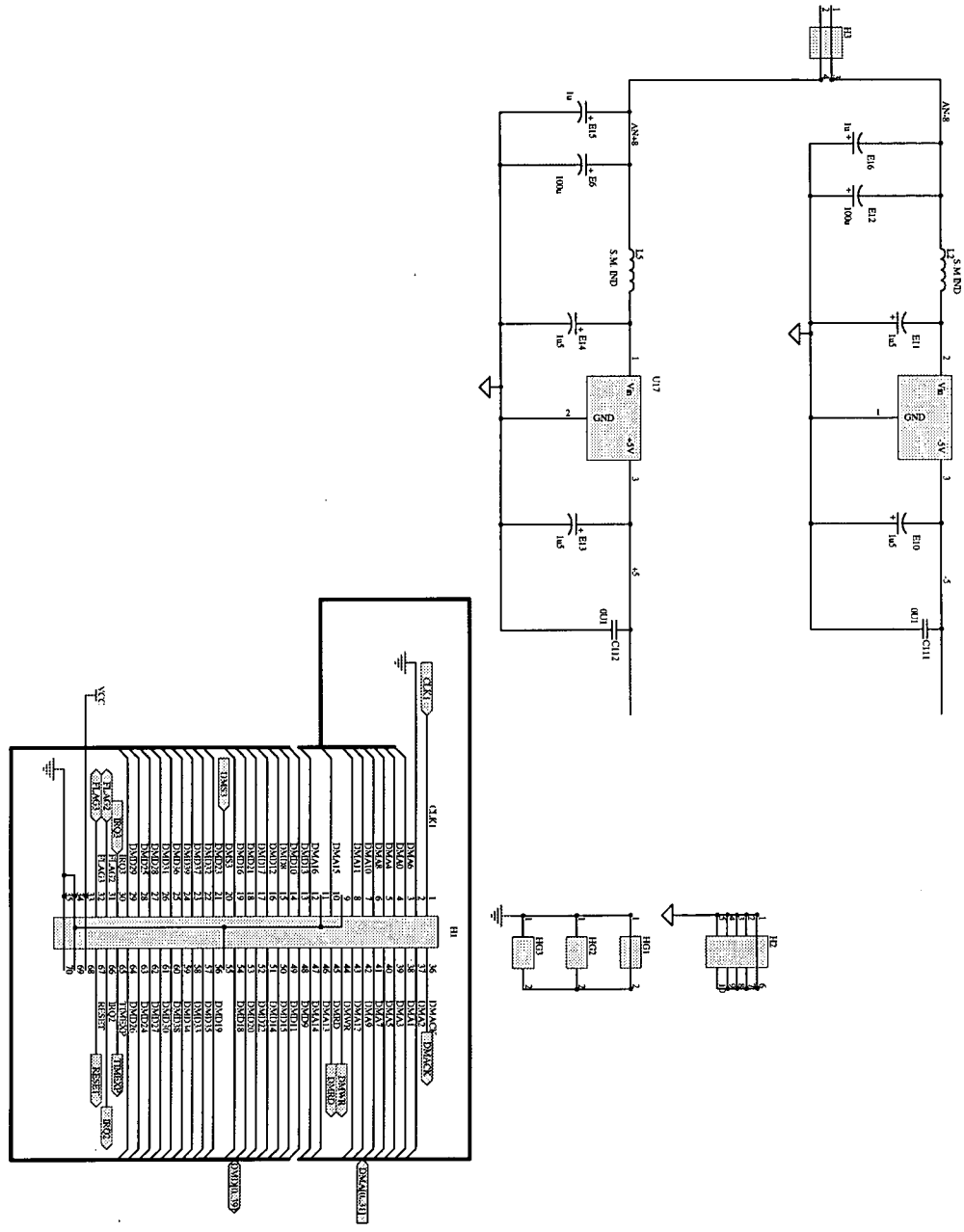


Figure 5.31. A to D 1 cicuit diagram part 4

significant parasitic components that may cause instability. Surface mount components and careful layout reduces unwanted inductance and parasitics. To reduce parasitic capacitance there is no ground or power plane directly under the 8001. A DC offset was added to the signal input to compensate for any offset introduced by the 8001's. This DC offset was generated by IC U6, figure 5.28, and associated components. The input to the board presents a 50 ohm load.

The Clock drive for the A to D's is provided from a CY7B991 clock driver, U15, figure 5.30. This phased locked loop, PLL, driver can supply phase shifted clock outputs and the inverted equal mark space ratio clock required. The clock output has series source resisters, matched to the signal line impedance, to absorb reflected signals and ensure the clock drive is 'clean'. An option for end terminating resistors was available. Terminating resistors were not used as the increased power requirements may in turn increase the risk of introducing noise and clock jitter. To ensure the A to D encode signal timings are exactly 180^0 out of phase the encode input to one A to D has a variable capacitor to adjust the timing. The A to D outputs are latched by the 574s, U20 and 21, figure 5.30. These latches are placed physically close to A to D. This reduces the drive requirements on the digital section of the A to D IC improving the noise immunity. The latches drive the A to D FIFO input, U8, figure 5.29. The FIFO used is an IDT72245 synchronous FIFO. The 72245 is similar to the 72265 used in the DSP section. It is a newer FIFO and will supersede the 72265. It is not as deep as the 72265 but future, pin for pin compatible, designs will be. It is second sourced by Cypress who produced a faster, deeper part to supersede the 72265 at a later date.

The FIFO is reset by the DSP using the DSP data line, DMD39. The FIFO write clock is provided from the 991 PLL clock driver. The FIFO write enable, WEN, is derived in the A to D board 2032 ISP logic device. The WEN signal is active when the 12 bit counter within the 2032 is counting down. On reaching zero WEN goes inactive preventing further write to the FIFO. (See discussion in section 4.9.4). The 12 bit counter within the 2032 can be loaded thus initiating another FIFO write sequence. The load is inhibited until the DSP toggles FLAG3, indicating it is ready for another set of samples. On receipt of a trigger from the radar the counter is loaded. As the radar trigger is asynchronous to the A to D and DSP it may repeat before the required number of A to D samples are gained. The radar triggers are therefore inhibited from generating another load until FLAG3 is again toggled. See A to D sample operation discussion below.

The 9058 A to D clock is derived from a 50MHz oscillator, U14, figure 5.29. The 50MHz clock is fed into the 2032 where it is divided by two to produce a 25MHz A to D clock option. The DSP FLAG3, the radar trigger and the A to D clock are asynchronous. All signals within the 2032 are 'synced up' to the selected A to D clock. By syncing everything to the A to D clock timing errors due to coincident signals etc are omitted. The FLAG3 signal may be high for 33ns, the DSP clock period, whereas the 25MHz A to D clock option has a 40ns period. The FLAG3 signal is therefore "stretched" to ensure it is 'captured' by the slower A to D clock. To do this the FLAG3 signal, routed to the 2032, is first synced up to the 50MHz clock negative edge. The synced signal is then 'stretched' and synced to the A to D clock rising edge, see figure 4.32 on the next page.

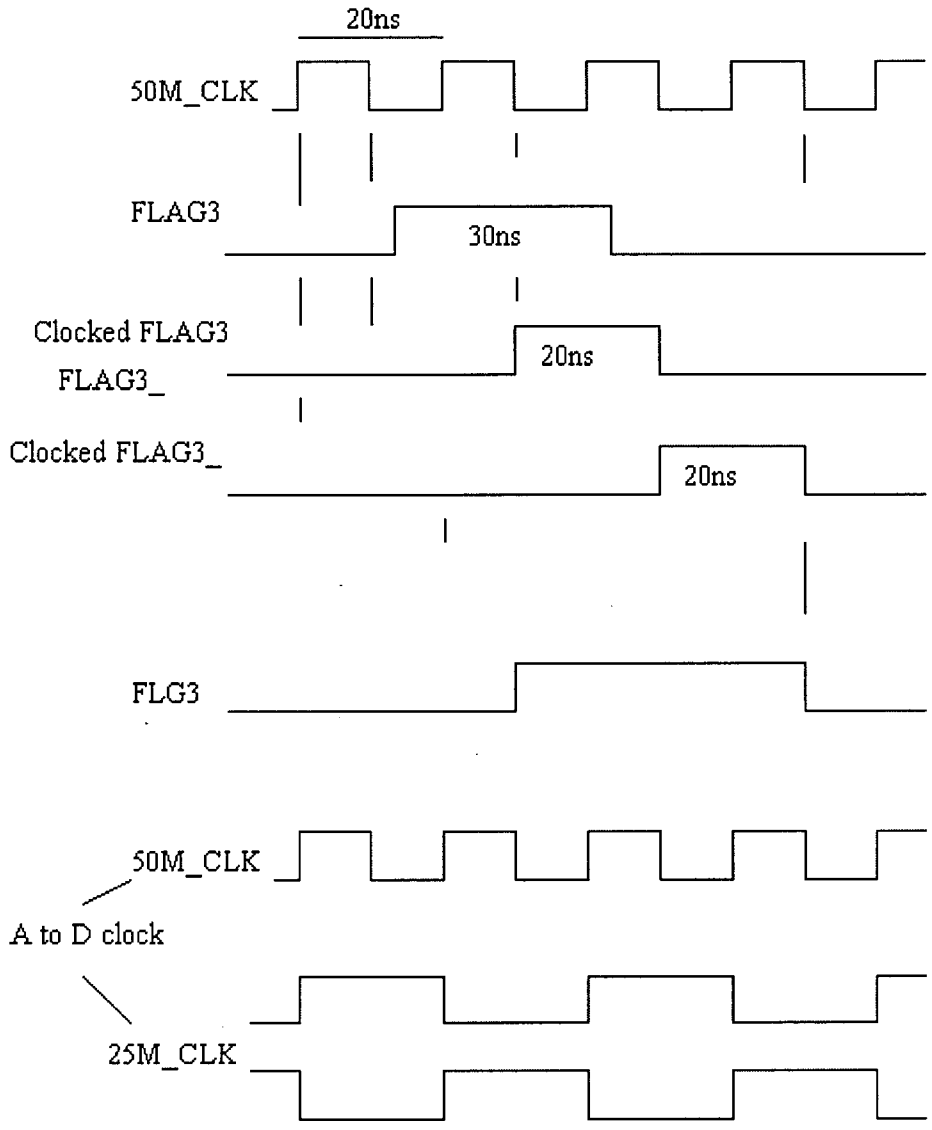


Figure 4.32. "Stretching" and synchronisation of FLAG3.

The A to D operation is controlled by the 2032 ISP logic device, U12, figure 5.29. The logic equation listings for this 2032 are given in the attachments. The logic control operation is based around a 12 bit synchronous down counter. Refer to block diagram of the control logic, figure 5.33 on the next page.

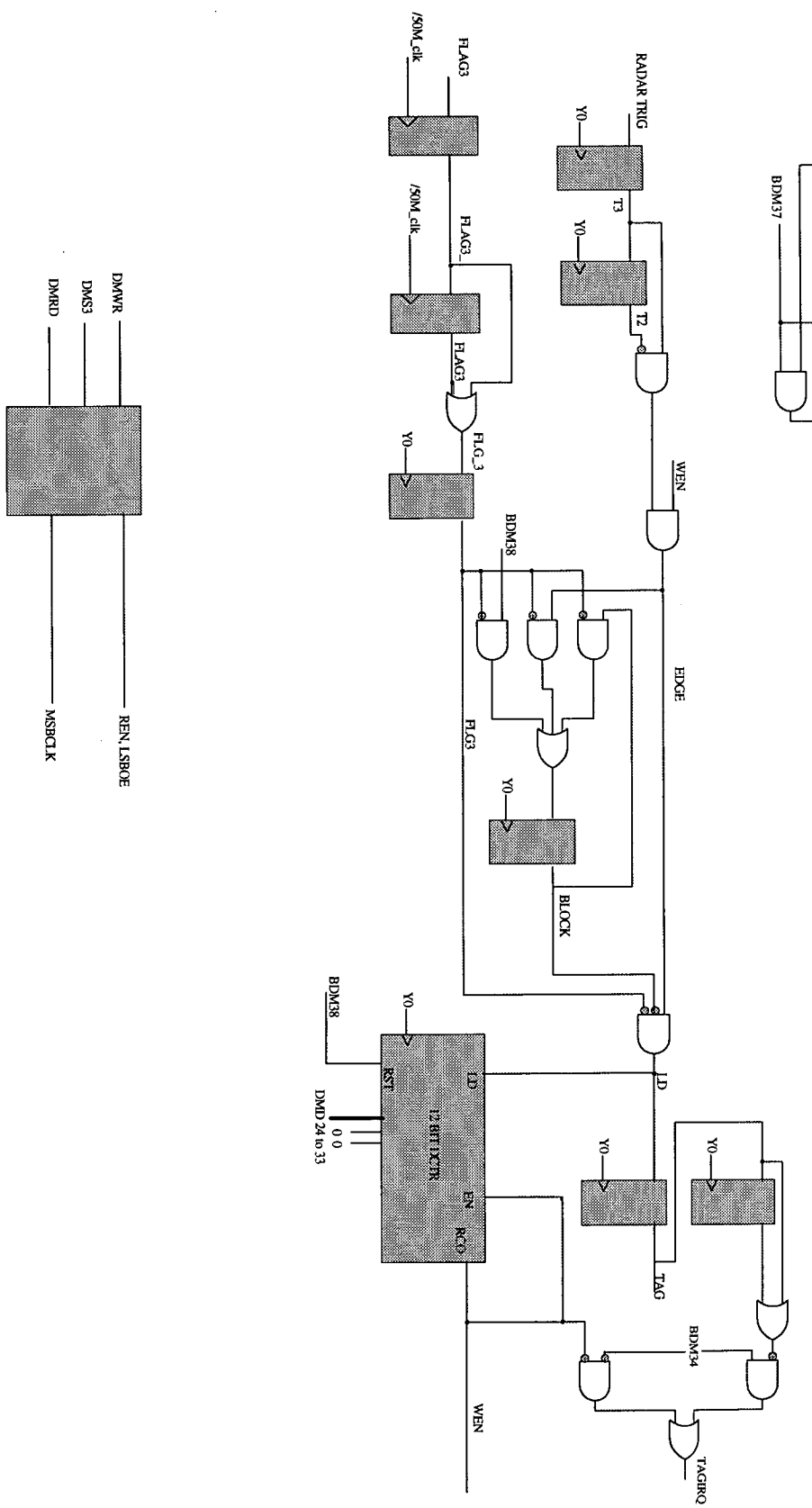


Figure 5.33. 2032 A to D 1 control logic

This counter may be reset, loaded and count enabled. The signal WEN generated by the counter is high when the counter has counted down to zero, (it is similar to the ripple counter output, RCO, in some counter ICs). WEN is inverted and drives the A to D FIFO write enable input. Thus, the FIFO stores data when the counter is counting down. On receipt of a radar transmit trigger signal a pulse one A to D clock period wide, EDGE, is formed from the rising edge of the trigger signal. This EDGE signal causes the down counter to load and count down. WEN is fed back to inhibit further active EDGE signals being generated thereby inhibiting further loads until the counter has counted down to zero. A further counter load is also inhibited by a BLOCK signal, see figure 5.33 on the previous page. BLOCK is generated from the EDGE signal. To initiate another sampling period the DSP sends FLAG3 high. The BLOCK signal is cleared by FLG3, generated from FLAG3 rising edge. If the counter has reached zero (WEN is high) and BLOCK is cleared (via FLAG3) a radar trigger will cause the counter to load and a new set of waveform samples is stored in the A to D FIFO.

This operation is shown in figure 5.34 below

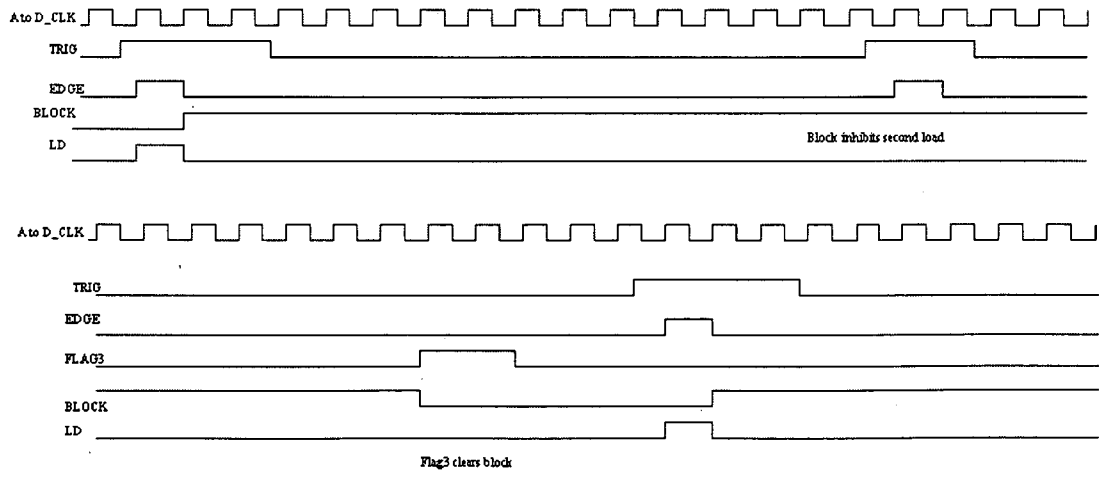


Figure 5.34. A to D 1 ISP2032 control function.

In the first attempt to create this control logic FLAG3 was not synced to the A to D clock. On testing the system 'extra' samples were taken for some waveforms, for example: if 128 samples per waveform was chosen, at random, a waveform with greater than 128 samples would appear. The reason is that if BLOCK was cleared and a FLAG3 signal arrived coincident with an EDGE signal the counter would load but BLOCK would not be set. The counter would count down and the next radar trigger would cause a second, unwanted, load to take place. Syncing FLAG3 with the A to D clock ensured that a BLOCK signal would accompany every counter load.
The terms within the logic expressions for the counter are rather large. Care has to be taken to ensure that the delays through the ISP 2032 GLB cells do not add up and violate timing restrictions. Excessive daisy chaining of GLBs to obtain the logic would cause unacceptably large timing delays. Trying to pass large numbers of signals between GLBs causes routing problems. The design is a compromise between the two.

The tag signal is selectable and can be either the WEN signal or the counter load signal. Using the counter load places a tag on the first sample of the waveform stored. The tag generated from the load signal is 'stretch' to create a DSP interrupt signal if required. The DSP samples interrupt and FLAG signals on the rising edge of its clock, stretching ensure the DSP 'sees' the interrupt.

The DSP reads the FIFO data via transceiver, U9 in figure 5.29, a 162652. Figure 5.35 below shows a block diagram of the A to D FIFO read system.

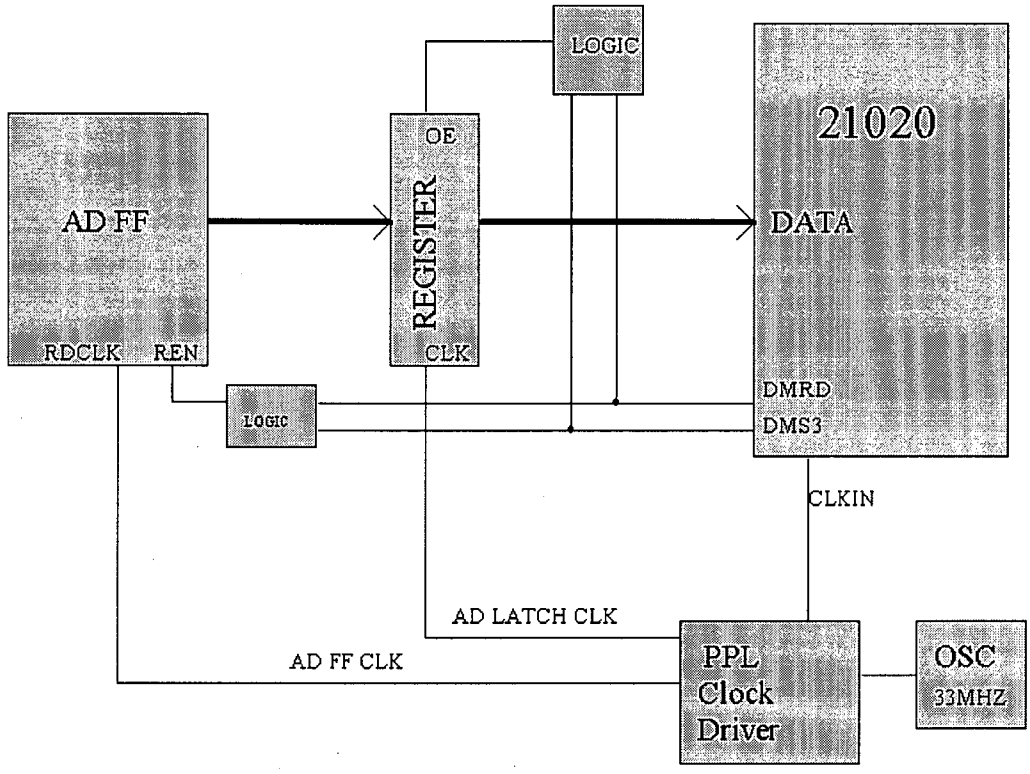


Figure 5.35 Block diagram of the A to D FIFO read system.

The FIFO read clock and transceiver is driven by a time-shifted version of the DSP clock. The FIFO read enable and transceiver output enable are formed from the DSP DMS3 and DMRD signals. Figure 5.36 on the next page shows the A to D FIFO read system timing.

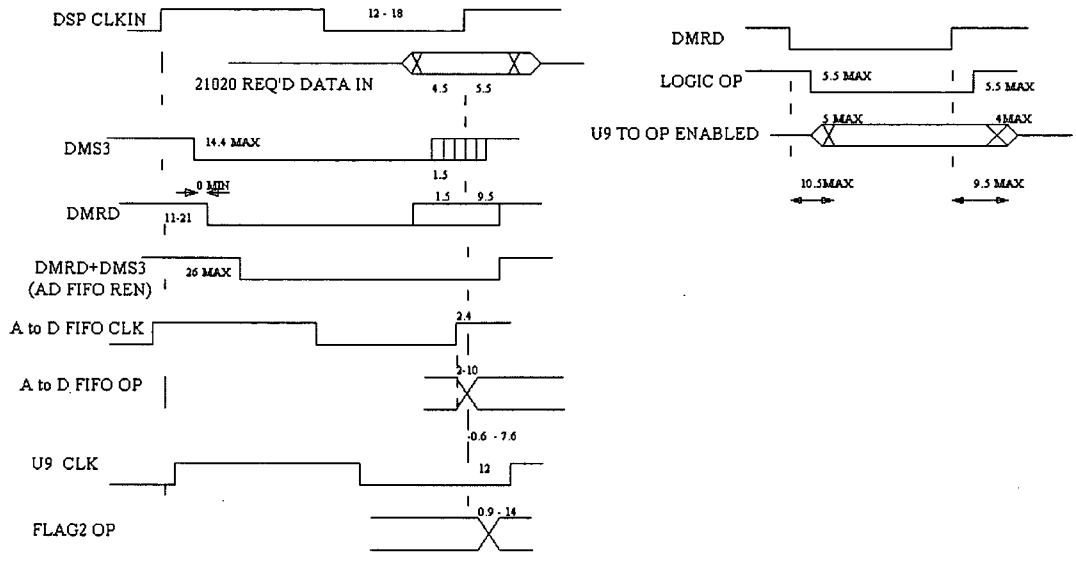


Figure 5.36. A to D FIFO read system timing.

5.5.3 A to D 1 PCB.

The PCB layout is an integral part of the design process. Often a circuit has to be modified due to PCB constraints. For example a filter response is defined by a lumped capacitor value plus an amount due to stray capacitance. This stray capacitance is a function of the PCB layout. Due to the high speeds involved the A to D circuits may include "components" that are not present on the circuit diagram, but are present because of the physical properties of the circuit board. A ground plane is mandatory to negate introduced PCB 'components'. Two grounds are used, one for digital, one for analogue. The ground PCB connections are made through multiple pins to reduce impedance. One of the problems with A to D PCB layout is that the digital section is capable of injecting noise into the analogue section. The best noise

reduction technique for reducing this noise injection is physical separation. The analogue signals are physically separated from the digital section of the board as far as possible. The A to D is treated as an analogue component. Often the A to D data sheets suggest joining the digital and analogue grounds at the A to D. In many systems this is not practical. What the manufacturer is saying is that the ground for the A to D needs to be as good as possible. The A to D has digital and analogue sections kept separate within the IC to avoid coupling of digital signals into the analogue section. Noise from the digital ground will be capacitively coupled to the analogue ground within the IC. A poor external digital ground will cause more digital noise to be capacitively coupled to the analogue ground. Figure 5.37 below shows a representation of the A to D silicon layout.

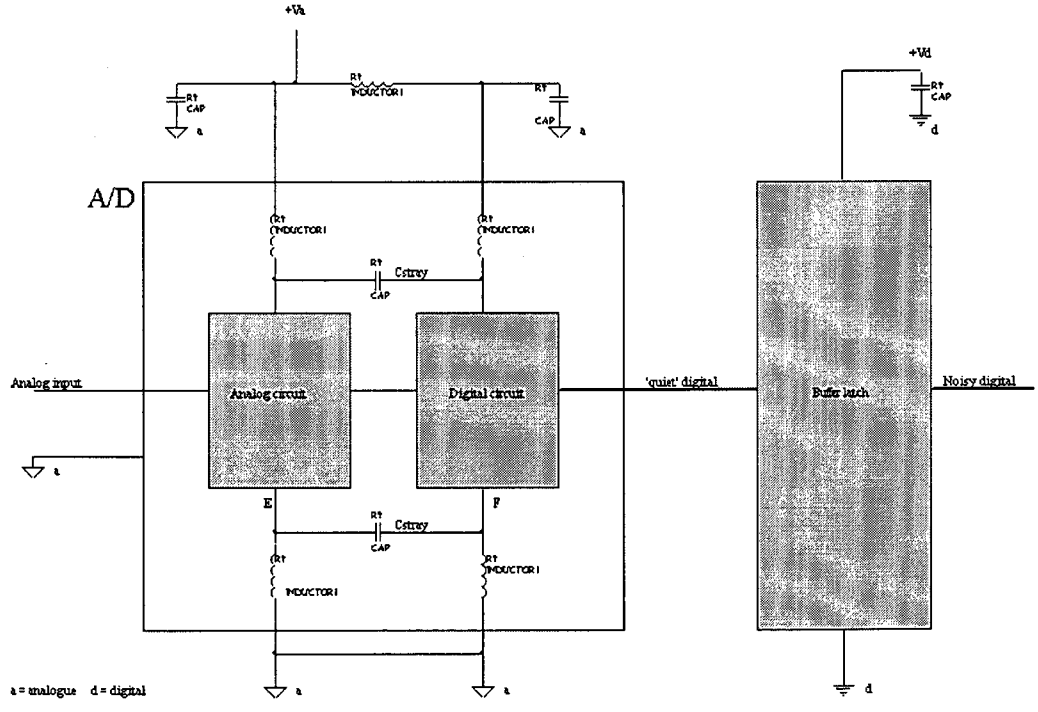


Figure 5.37. A to D converter showing the digital and analog sections.

Rapidly changing digital currents at F in figure 5.37 will cause a voltage that will couple into point E of the analogue circuit through the stray capacitance. There is also about 0.2pF of capacitance between the IC legs. Therefore the digital and analogue grounds are joined together externally with minimum lead lengths to reduce extra impedance. Any extra impedance at the digital ground will cause more digital noise to develop at point E, coupling more noise into the analogue circuit. Digital noise from the digital section of the A to D is introduced onto the analogue ground but is minimised by ensuring the A to D does not drive large fan outs. Surface mount capacitors are located next to the supply pins. Ferrite beads are also used to isolate power supplies.

The A to D clock is grounded to the digital ground. For A to D accuracy's greater than 8 bits the analogue ground would be used. This would reduce the phase noise of the clock and improve the SNR. The A to D clock is well decoupled and placed in a less noisy section of the digital ground. Separate analogue and digital power supplies are used. In the digital section transmission line techniques are employed in the PCB to control track impedances and reduce ringing etc. The area on the DSP motherboard under the piggyback board is free of high speed digital signals and is shielded by power and power planes on the piggyback board to reduce cross-talk from the DSP board. Where possible multiple power and ground pins are used on the digital connector. Ground pins physically separate some digital signals in an effort to reduce cross-talk. The analogue power and ground connectors are separated from the digital power and ground connectors. The digital signals from the DSP are kept to a minimum length to reduce transmission line effects. A four-layer board is used. One layer is

dedicated to ground. The power supply takes large portions of the other planes. See ad.pcb on the attached floppy disk. Throughout the design careful consideration to the return path currents is adhered too ensuring that large transient currents do not flow through the ground plane near sensitive areas. On the analogue section large single vias are used to connect power to reduce impedance and eliminate current loops.

5.5.4 A to D 2

The second A to D circuit uses a single A to D, the AD9012, capable of 100MSPS conversion rates. Section 4.5 outlines the system design. Figures 5.38 to 5.41 on pages 108 to 111 show the schematic diagram. The DSP interface is unchanged. The AD9012 requires an external voltage reference. This was produced from a low noise voltage reference, the Analog Devices REF43, IC U52, figure 5.41. This reference is filtered and conditioned to be in the –2v range required for the AD9012. An OP27 OPamp was chosen for the reference conditioning. The OP27 is a general purpose OPamp with good long term and temperature drift characteristics. The A to D clock is produced by a PLL frequency doubler, the MC88915, driven by a 50MHz crystal oscillator, IC U14, and the 33MHz DSP clock see figure 5.41. The clock to be doubled is selectable within the MC88915 PLL IC. Figure 4.13, page 54, shows the method used to reduce the A to D output data clocking rate. The data clocking rate is halved and the data word width is doubled, thus the data rate is retained. The logic functions are again supplied by the 2032 ISP logic device. The logic equation listings for this 2032 are given in the attachments.

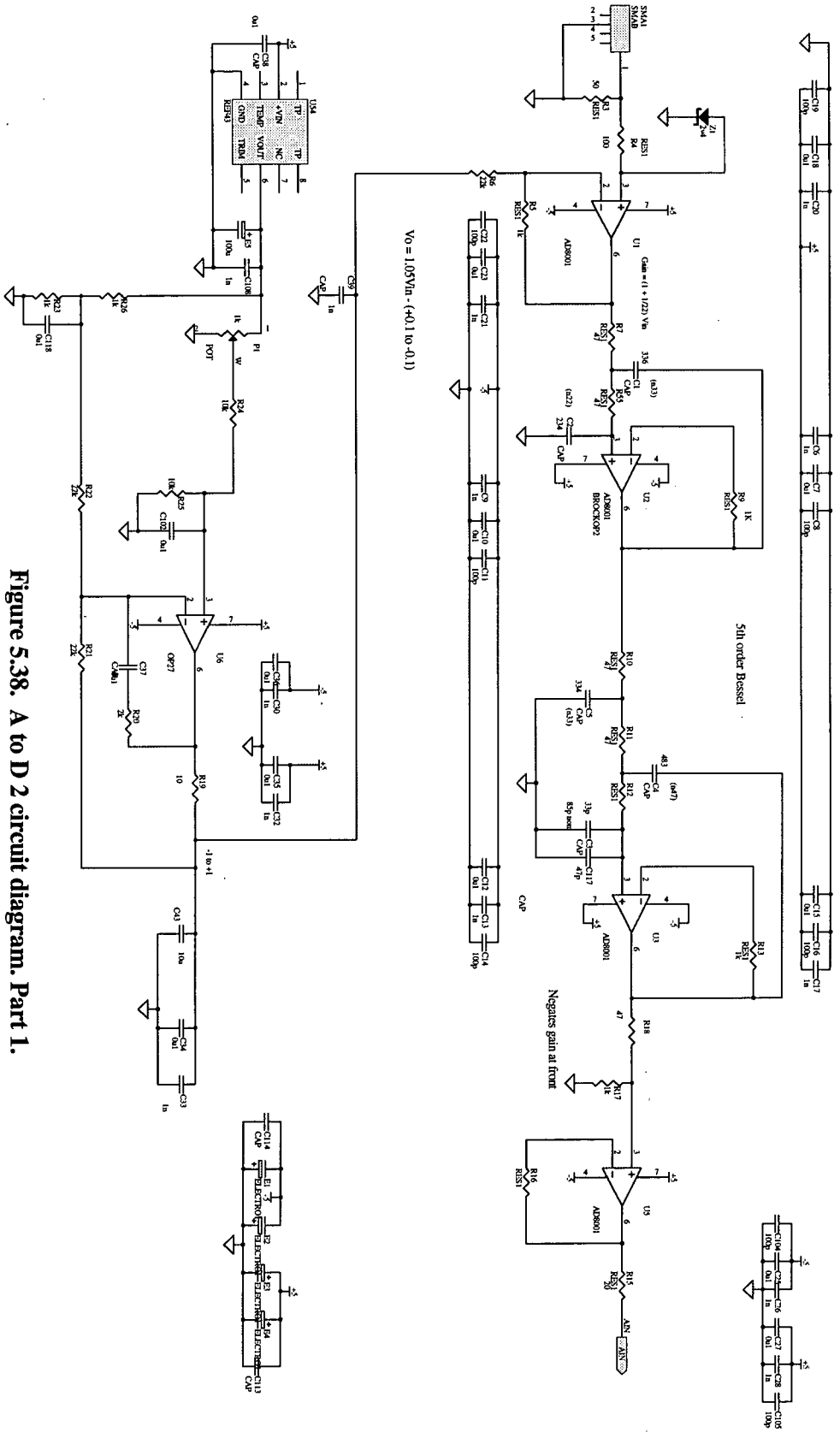


Figure 5.38. A to D 2 circuit diagram. Part 1.

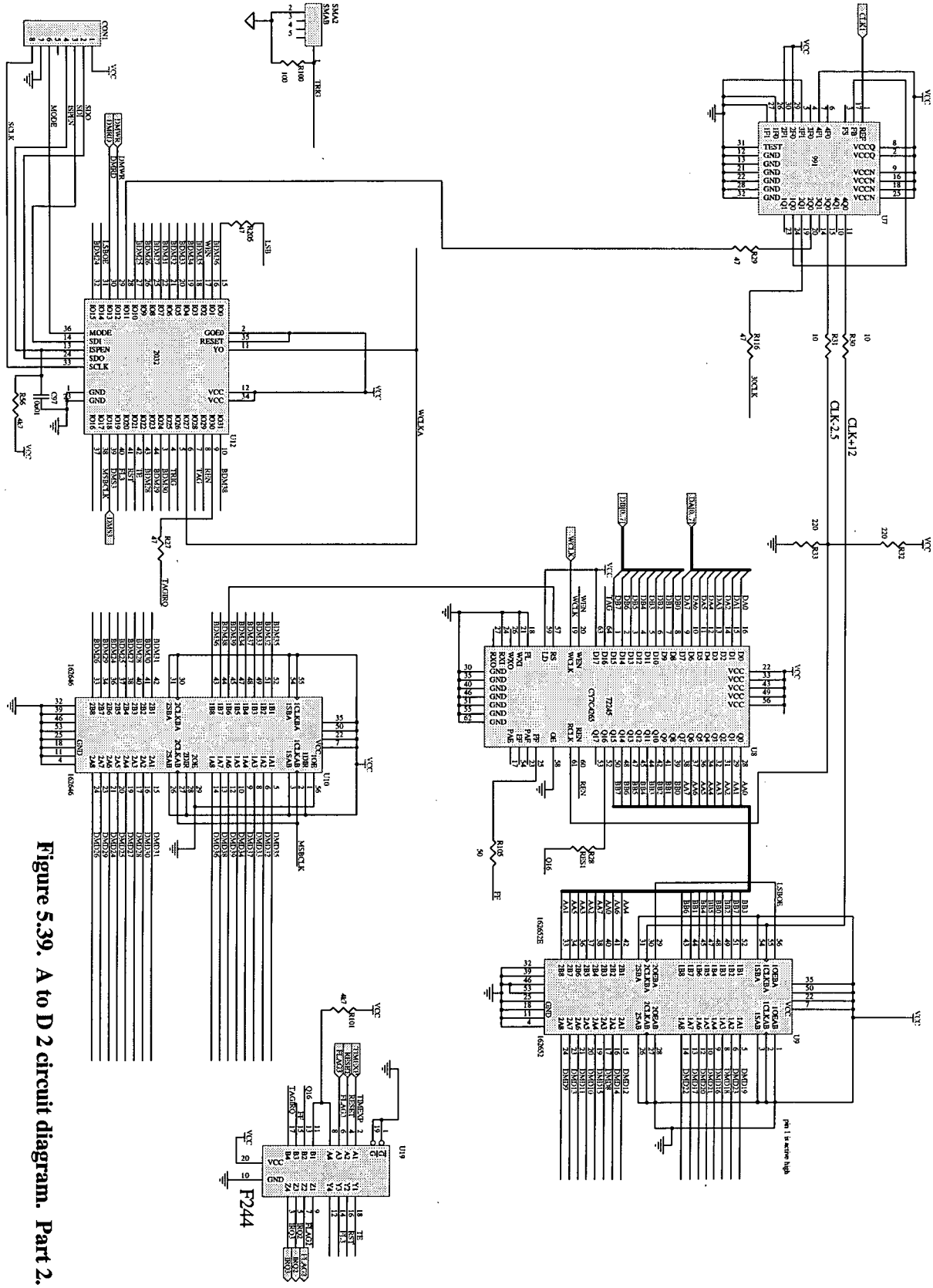


Figure 5.39. A to D 2 circuit diagram. Part 2.

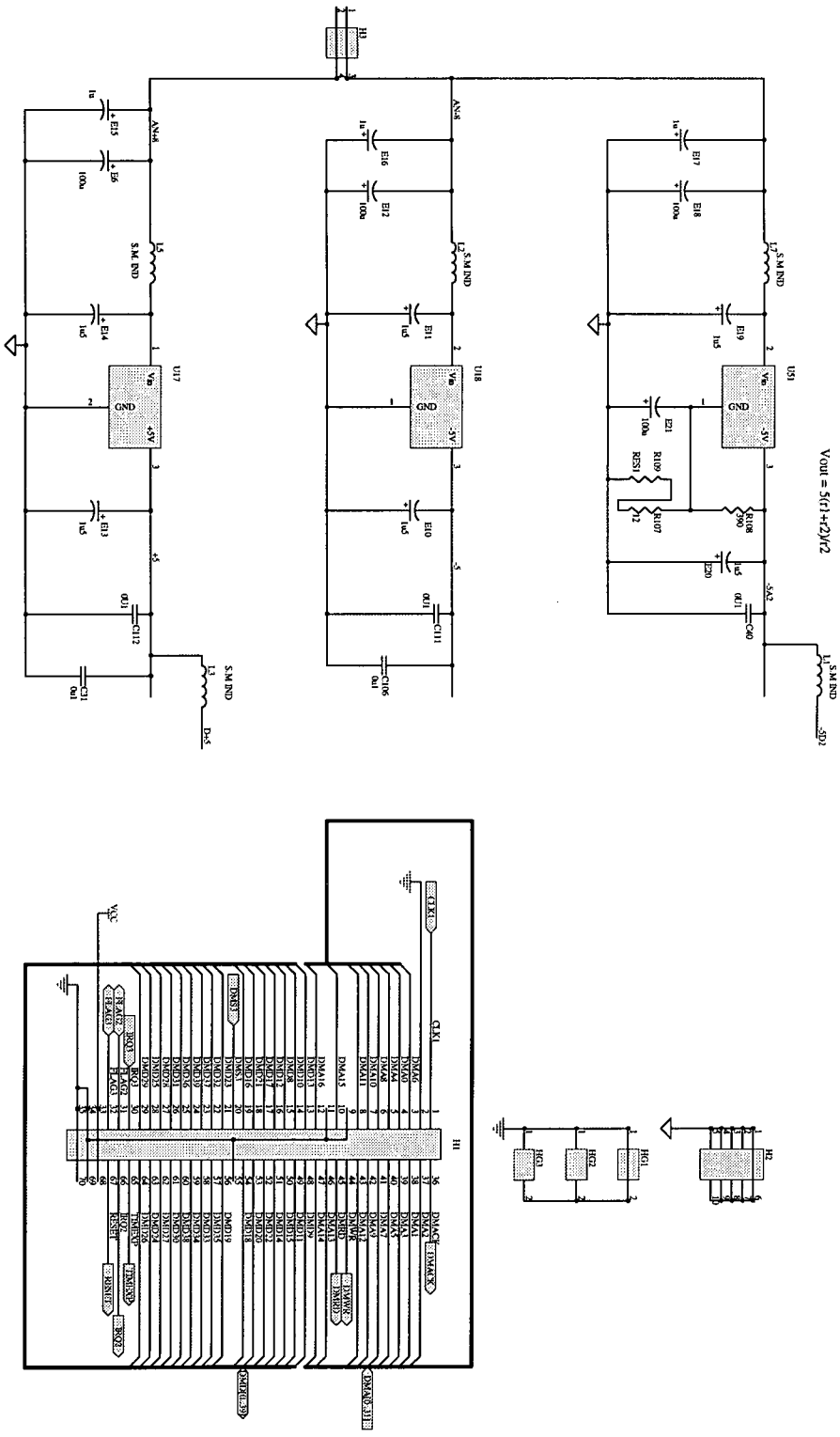


Figure 5.40. A to D 2 circuit diagram. Part 3.

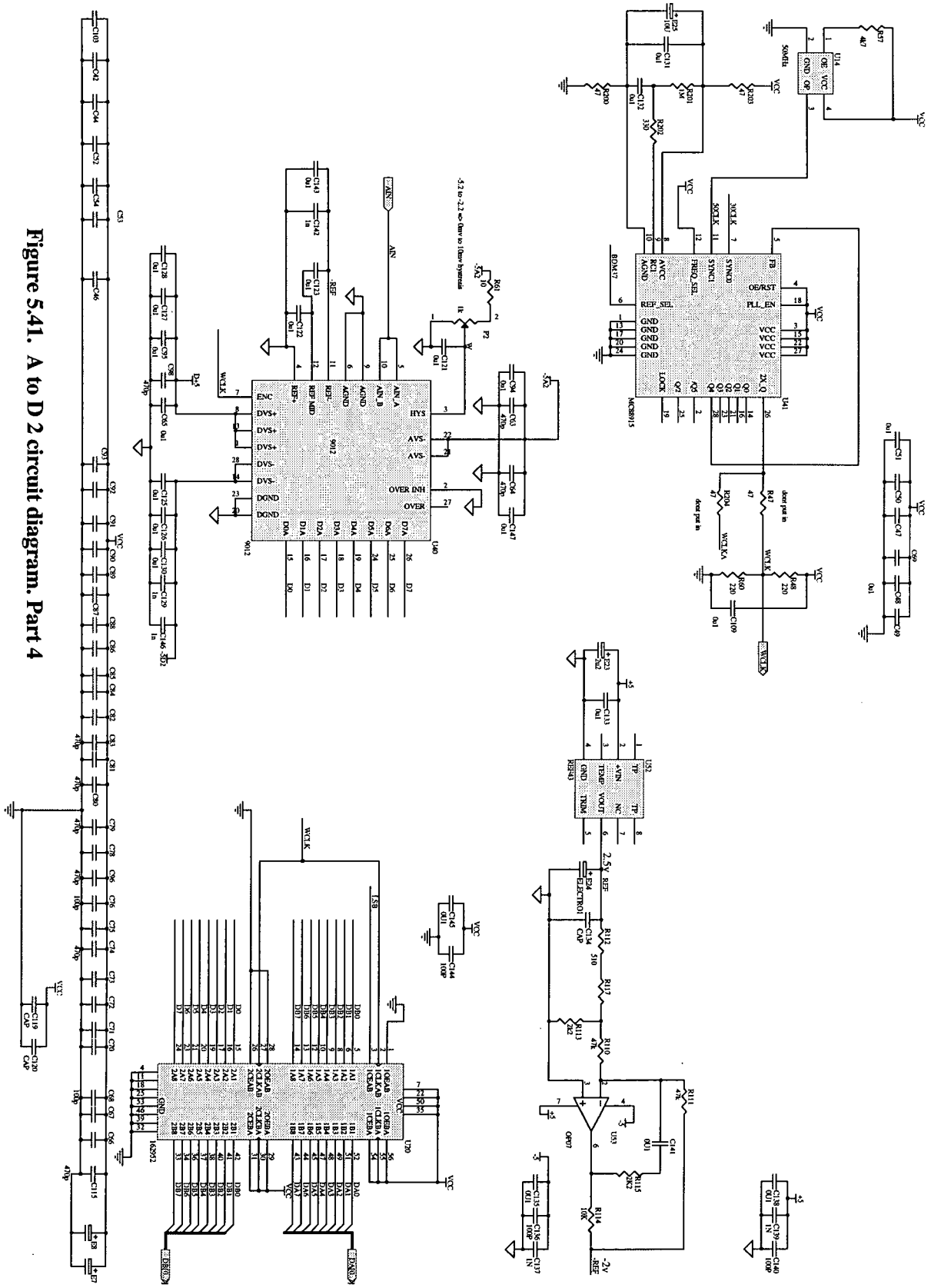


Figure 5.41. A to D 2 circuit diagram. Part 4

The logic control is based around a 13 bit synchronous down counter (allowing 8k samples per waveform) as shown in figure 5.42 on the next page. The radar trigger and FLAG signals are synced up to the A to D clock, (100MHz or 66MHz). As this clock is faster than the DSP clock the input signal did not need to be stretched, however this means the output signals to the DSP do. The time period for the A to D logic functions when using the 100MHz A to D clock is now 10ns, thus as the 2032 propagation through the device is 5.5ns, only one level of GLB cascading is allowed. This second A to D board design was able to take advantage of a new fast, deep FIFO, the Cypress CY7C4265, a 16k x 18 with 10ns access time. Figure 4.13, page 54, gives a block diagram of the A to D system and 5.43 below shows the A to D FIFO write timing.

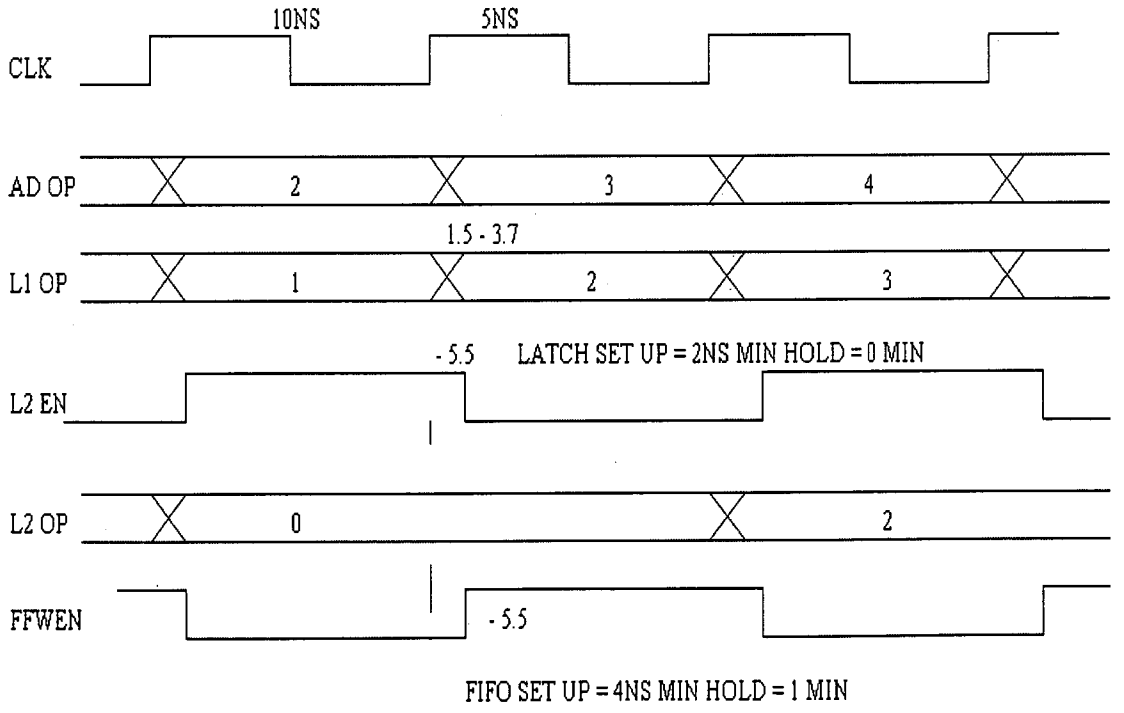


Figure 5.43 A to D latch and FIFO write timing.

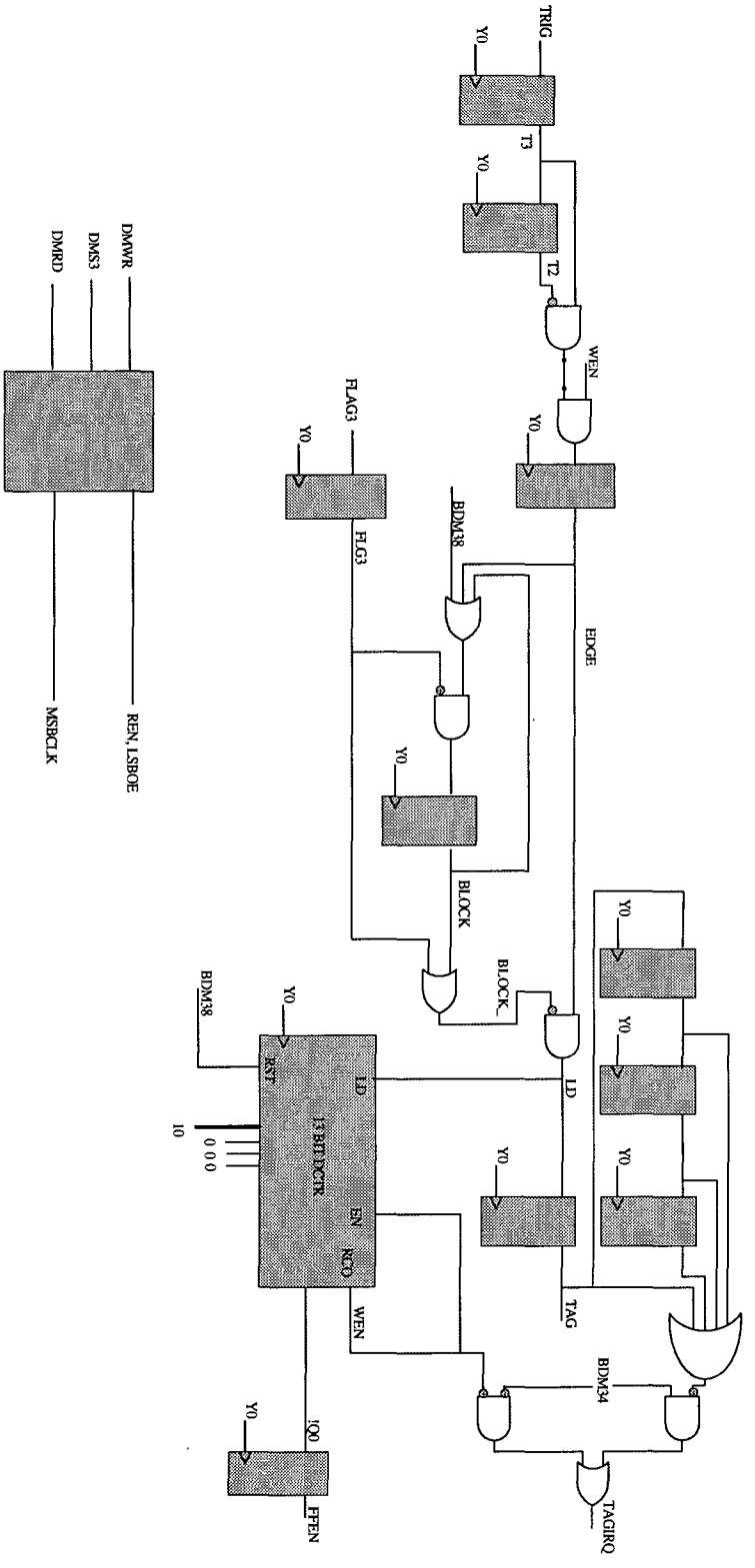


Figure 5.42. 2032 A to D 2 control logic

5.5.5 A to D 2 PCB

The layout constraints for the 9012 circuit were similar to the 9058 design. There was the added complexity of the 100MHz clock, whose track length was necessarily longer than the 50MHz clock in the previous design. This signal is terminated at the end of the 100MHz clock transmission line to minimise reflections near the A to D.

Chapter 6. Construction and Testing.

One of the problems with high-speed circuits is how to measure and observe what is happening on the signal lines. For example, at slow speed a normal oscilloscope probe could be placed on a digital write strobe and the operator would have confidence that what he saw on the CRO screen was a good representation of what was happening on the track. With high-speed signals this is not the case. Any significant capacitance or inductance of the probe would severely distort the signal to be measured. A typical probe has a capacitance of about 10pF and a long, inductive, earth lead as well as relatively slow rise and fall times. The 10pF probe capacitance alone may stop a circuit from working by inserting it. Special low capacitance FET input probes are required. These probes have sub pF input capacitance and very fast rise and fall times. Two such probes were purchased for used with a Tektronix TDS 1GSPS digital oscilloscope. A Tektronix 3001GPX logic analyser was also used. This analyser can sample using its internal clock at 1GSPS. Once the measuring equipment is acquired the next problem is connecting it to the physically small devices. Surface mount IC clips are very expensive so small flying leads were soldered to the IC legs and the measuring equipment connected to theses flying leads. These methods gave a good enough approximation of the circuit operation being measured. A crude technique for detecting periodic pulsed digital signals is using a normal oscilloscope probe and connecting the earth to the probe to form a loop. This acts as an antenna and can detect changing digital signals when placed in close proximity to them.

The DSP board was populated and tested piecemeal. First the ISA bus interface was placed and tested, followed by the RAM and finally the DSP. At each stage a short test program was written and the response of the DSP board observed. A list of test programs is given in the attachments. To test the ISA bus interface
the ISA bus transceivers and 1048 were placed on the PCB and a short PC test program exercised the ISA bus lines while the response was monitored. After verifying the 1048 and transceiver operation the RAM and 2032 were added and a short PC test program exercised the ISA bus lines and DSP memory busses and control lines. The final stage was the insertion of the DSP itself. Short PC and DSP test programs were written and the response of the DSP board monitored. The programmable logic devices were reprogrammed in circuit as necessary reducing the need for PCB modifications. To avoid any bus contentions while programming the ISP logic devices, the DSP is placed in IDLE mode. This mode tristates the DSP memory bus pins and halts processor operation. The ISP device IO pins are tristated and pulled high by internal 100k pull-ups when being programmed. All pins going to device output enable, OE, pins were pulled high by external resistors to avoid bus conflicts. Before downloading the DSP programs, program memory RAM is first filled with zeros. This is done as the first few memory locations in the DSP are reserved by Analog Devices. Anything other than 0, equivalent to a NOP instruction, in these locations causes the DSP to crash.

The A to D board was built up in a similar piecemeal manor. The analogue filter section was tested separately with a network analyser. The A to D output is read in to the PC via the DSP board.

To test the A to D and DSP system a signal generator and oscilloscope was used. The signal generator output a narrow pulse to simulate the radar return echoes. The oscilloscope was used to view the signal going into the A to D in order to compare it with the signal displayed on the PC monitor. As the generator signal amplitude was reduced the number of waveforms stacked (averaged) was increased. With sufficient number of waveforms stacked the signal into the A to D could still be seen even though it was well below the amplitude of a single A to D bit, i.e. a signal of 2mV could clearly be seen even though the quantisation levels of the A to D is 8mv. When using the dual A to D 9058 board, the two stacked waveforms formed from each A to D output had a dc offset between them and had different amplitudes. No amount of circuit modification could remove these differences. The reason is discussed below.

Noise is associated with the A to D signal input as shown in figure 6.1 below.

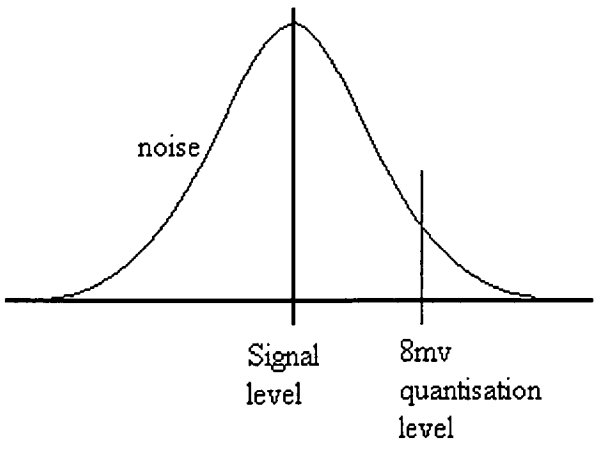


Figure 6.1. Diagram of signal and associated noise.

For a fixed signal the absolute voltage will vary due to the noise on the signal. From the figure it can be seen that a quantisation threshold can be passed due to noise. The closer the signal is to the quantisation level the more crossings will take place. Therefore the number of quantisation threshold crossings is proportional to the voltage level; therefore a representation of the signal below the quantisation level is possible when many samples are stacked. The reason that the stacked outputs from each A to D within the 9058 produced different waveforms is that the local noise around each A to D is different due to the physical differences within the A to D. This led to different A to D outputs. It may be useful to go below the bit level to see a representation of the signal. The 'double' image would be confusing therefore a different A to D system was designed using the 9012.

The AD9012 gave similar results but as only one A to D is used there is no offsets between adjacent samples.

Chapter 7 Radar Logging Project.

7.1 Radar system set-up

The DSP system was designed to process and record return echo information from an ice radar. This ice radar was flown in a helicopter over the Lambert glacier in Antarctica. The radar return echo output, a zero to -2 volt signal is input to the A to D and DSP system. The DSP averages together several waveforms to increase the signal to noise ratio. The improvement in signal to noise ratio is proportional to the square root of the number of waveforms averaged. As there is no additional information gained from dividing the summed waveforms by the number of waveforms summed, the waveforms are simply added, i.e. stacked. The stacked waveform is output to the PC and displayed on the PC monitor along with GPS positional and barometer pressure information. The DSP stacking program is downloaded to the DSP from the PC. The user can interact with the system operation to select the number of waveforms averaged, the number of samples per waveform, the logging period, and the A to D conversion rate.

7.2 The DSP logging software

The aim of the DSP program is to average, or more correctly stack, up to 1024 return radar echo waveforms in less than 1second. The discussion below highlights sections of the program to demonstrate some of the DSPs powers. The program, logger.asm, is shown attached starting at page 161 and is on the accompanying floppy disk.

This DSP program was written in assembly language to obtain speed efficient code. Assembly language programs run faster than higher level programs and the designer has full control over the DSP operation. Analog Devices, the DSP IC manufacturer, recommends assembly language for speed critical applications and has a list of speed optimised functions available. (A 'C' compiler is available if required from Analog Devices). The DSP program reads digitised radar return echo waveforms from the A to D piggyback board and stacks them together. The DSP reads the data memory transceiver latches, loaded from the PC, to obtain set-up information for the logging operation. The 32 bit word in the data memory transceiver latches contain the number of waveforms to average, the A to D clock speed and number of samples to be taken per waveform. The number of samples per waveform and A to D clock speed are passed to the A to D board in the upper 16 bits of the 32 bit data memory. This data is latched on the A to D piggyback board and used to set-up the A to D operation.

The DSP stacking program consists of four main sections:

1. Set up the DSP and A to D board
2. Grab the first waveform
3. Stack the first and following waveforms
4. Add the final waveform to the stacked waveform and output the results to the PC FIFO.

If only two waveforms are to be stacked section 3 is jumped. If no waveforms are stacked, i.e. simply display one waveform, the program

jumps to a separate program that simply outputs the A to D FIFO contents to the PC.

These sections are now discussed in greater detail.

As mentioned above the first section sets up the DSP registers and A to D control ready for the main section of the program. The DSP has two data address generators, DAGs, one for program memory one for data memory. The DAGs maintain pointers into memory allowing the processor to address memory indirectly. Each DAG can keep track of up to eight address pointers to memory. (Chapter 4 of the 21020 users manual). The DAGs are initialised in the set-up section as shown in the example below.

```
i0=atod;
m0=0;
l0=0;
b0=atod;
```

The register i0 points to memory location 'atod', the memory mapped A to D board, m0 contains the increment value for advancing the pointer set to zero as i0 is to continually point at the A to D board, b0 holds the base address of a circular buffer and l0 holds the number of locations in the circular buffer. In our example above circular buffering is not used therefore l0 is set to zero. In the instruction:

```
R0 = DM(i0,m0);
```

register R0 is loaded with the data pointed to by i0. Pointer i0 is initialised to point to the data memory mapped A to D FIFO. After each access the pointer, i0, is incremented (by the data memory DAG) by an amount m0. In this case m0 is equal to zero thus i0 always points to the A to D FIFO.

The second section of the program causes the DSP to grab the initial waveform to be stacked and places it in an array in program memory. A start code, 0xFC, 0x10, 0xFC, 0xFC is written to the PC FIFO. This marks the start of the stacked (averaged) waveform for the PC. If the system is placed in a noisy environment it is possible that a FIFO be corrupted (e.g. a spike causing an extra write). The FIFO may eventually fill with bad data. The PC therefore checks for the start of waveform code to validate the data. To instruct the A to D to grab a waveform FLAG3 is toggled.

```
BIT SET ASTAT FLG3;
BIT CLR ASTAT FLG3;
```

(ASTAT is a control register in the DSP).

The A to D FIFO is then read and FLAG 2 checked for the A to D start of waveform tag, (generated in the A to D logic).

```
          DO check1 UNTIL FLAG2;
          R0 = DM(i0,m0);
Check1:   R3 = R0 AND 0xFF;
```

The above code forms a loop that is repeated until FLAG2 is found to be active. The 21020 has a three level instruction pipeline, fetch, decode and execute. The processor tests the loop termination condition (in our case FLAG2 high) before the end of the loop so that the next fetch either exits

the loop or returns to the top based on the test result. The condition is tested when the instruction two locations before the last instruction in the loop is executed. In the example code below the condition is tested when code2 is executed.

```
DO label UNTIL FLAG2_IN;
         code1;
         code2;
         code3;
label:   code4;
         code5:
```

Depending on the test result either the code5 or the code1 instruction is fetched. Short loops (as in our logging loop) terminate in a special way. For a two instruction loop the condition is checked during the last instruction. If the condition is true during the last instruction the loop exits after one more loop pass. If the condition is true during the first instruction two more loop passes occur before the loop exit. In our logging code example:

```
         DO check1 UNTIL FLAG2;
         R0 = DM(i0,m0);
Check1:  R3 = R0 AND 0xFF;
         next;
```

FLAG2 becomes active when RO is loaded and the exits as follows:

```
         DO check1 UNTIL FLAG2;
         R0 = DM(i0,m0);        FLAG2 becomes active
Check1:  R3 = R0 AND 0xFF;      FLAG2 tests true
```

```
R0 = DM(i0,m0);
R3 = R0 AND 0xFF;
next;
```

The first two samples held in R0 are therefore lost, being overwritten by the next A to D read. This is acceptable as it is consistent and, due to the radar trigger positioning, the first two samples occur before the radar transmits. No radar echo waveform information is therefore lost.

The A to D output data is held in the lower 16 bits of the 32 bit data memory and consists of two A to D eight-bit bytes. See figure 7.1 below.

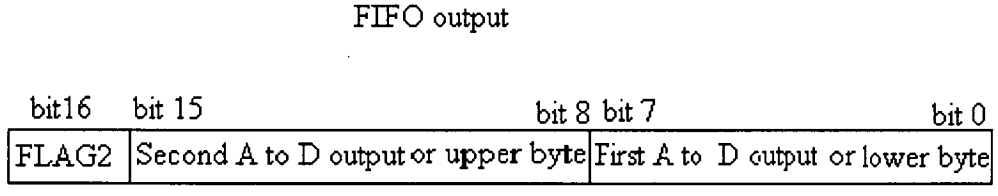


Figure 7.1. Position of A to D data in DSP data memory word.

Register R3 is loaded with the first byte (bits 0 - 7) i.e. one A to D output by ANDing the 16 bit FIFO output, held in R0, with FF hex.

On finding the tag active, which corresponds to the first sample of the waveform, the DSP exits the loop (as described above) and reads in the waveform samples from the A to D. See code below.

```
LCNTR=R13, do first1 until lce;
          R5 = FEXT R0 BY R7,R0=DM(i0,m0), PM(i9,m9)=R3;
first1:  R3=R0 AND 0xFF, PM(i9,m9)=R5;
```

The first line, `LCNTR= .....`, sets up the loop. The loop is counter based, the counter being implemented in hardware within the 21020. As this loop is counter based the 21020 can exit the loop without overheads provided the loop is executed at least twice. If the loop is executed less than twice the 21020 inserts two NOP (no operation) instructions. See 21020 user manual, program sequencing, chapter 3.

Register R13 holds the number of loops to be executed (equal to the half number of samples in the waveform, half because each A to D FIFO word has two samples) minus 999. (The minimum number of samples requested by the PC is 3000). This first line, LCNTR= ....., is a one clock cycle loop overhead. The last line of the loop is labelled first1. The loop is 'shortened' by 999 in order to toggle FLAG3 thereby requesting another waveform before the present one has been processed. In doing this the DSP will have the next waveform available when it has processed the present waveform and will not have to halt operations until a new waveform is received. Remember the radar is asynchronous to the DSP and the period between waveforms is 0.1ms, (section 2). If the waveform is grabbed immediately after toggling FLAG3 the A to D FIFO is deep enough to store the new incoming waveform and the remainder of the previous waveform. If the waveform is not grabbed immediately (waiting for a radar trigger) the DSP is not slowed down as it is still processing a waveform. In the second line of the loop five operations are executed in parallel in one instruction clock cycle:

- The second A to D byte , (bits 8 to 15), is extracted from register R0 using the DSP barrel shifter, `FEXT R0 BY R7`.
- R0 is loaded with the new, two byte, A to D outputs, `R0=DM(i0,m0)`.

- Register R3 (containing the A to D lower byte) is placed in program memory, `PM(i9,m9)=R3`.
- The program and data memory pointers are incremented by the DAGs.

The area in program memory used for data storage is pointed to by i9. The program memory DAG controls the pointer i9. I9 is incremented by an amount m9, in this case set to one, after the access. The pointer is therefore incremented and points to the next location in program memory. Program memory is used to store the A to D data in the loop thus freeing up the data memory and allowing two memory operations to execute in parallel. Once in a loop the program can execute the loop using the program memory cache within the 21020. Notice that R0 is read on the first half DSP clock cycle and written too in the second half DSP clock cycle.

In the final line the new lower A to D byte is stored in R3 and the extracted upper A to D byte (held in R5) is stored in program memory.

FLAG3 is toggles after this first loop.

The remainder of the waveform is now emptied from the A to D FIFO (which may now be filling also) by the loop code shown below.

```
LCNTR=995, do secound1 until lce;
          R5 = FEXT R0 BY R7, R0=DM(i0,m0), PM(i9,m9)=R3;
secound1:    R3=R0 AND R6, PM(i9,m9)=R5;
```

In this manor the A to D output bytes are stored in a program memory array.

This second section code above is shown in figure 7.2 pictorially on the next page.

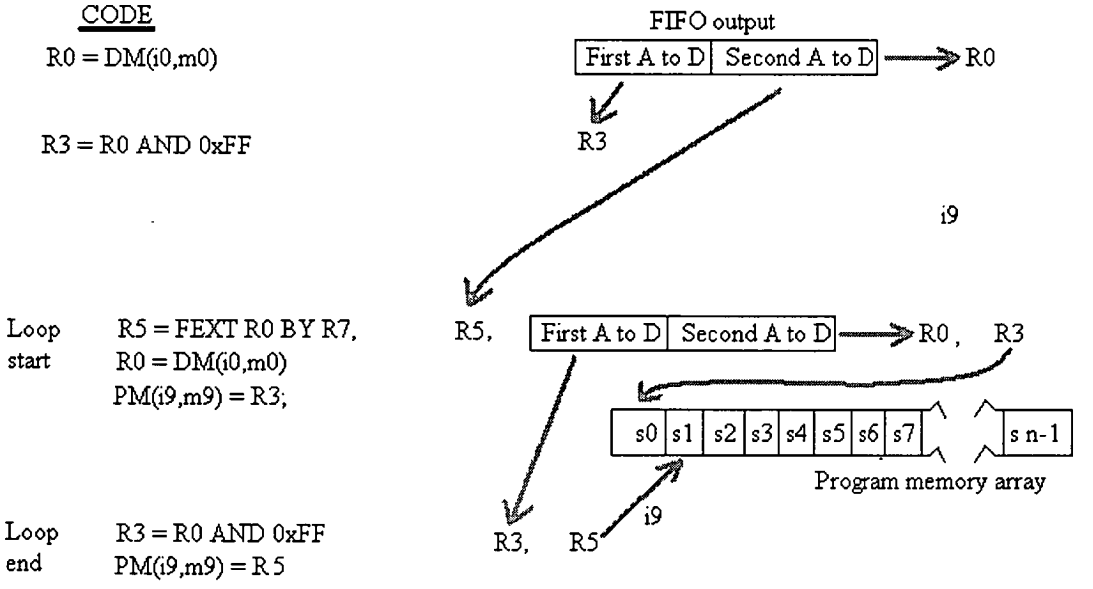


Figure 7.2. Pseudo code for second section of DSP code showing data movement.

The third section is slightly more complicated. A new waveform is grabbed and stacked with previously stacked waveforms. The new sample is simply added to the sum of the other samples taken at the same point on the waveform.

This section consists of a nested inner and outer loop. The inner loop is concerned with the stacking of the new sample and is iterated for each sample of the waveform. The outer loop is executed for the number of waveforms to be stacked.

Pointer i8 and i9 are initialised to point to the beginning of the program memory data array containing the waveform. See figure 7.3 on the next page.

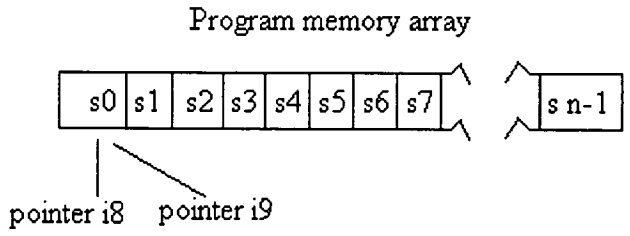


Figure 7.3. Diagram of program memory array showing data

The A to D FIFO is read and FLAG 2 checked for the A to D start of waveform tag as described previously. The program is shown below.

```
DO check2 UNTIL FLAG2_IN;
          R0=DM(i0,m0);
check2:   R3=R0 AND 0xFF;
          R1=PM(i8,m8);
```

The above code forms a loop that is repeated until FLAG2 is found to be active. The A to D FIFO output is grabbed and the first A to D byte placed in R3. R1 is loaded with the contents of the first memory location of the stacked waveform array, pointed to by i8 and i8 is incremented.
Once the start of waveform is found the program enters the main loop:

```
  LCNTR=R13, do main until lce;
          R4=R3+R1, R1=PM(i8,m8);
          R5 = FEXT R0 BY R7, R0=DM(i0,m0),PM(i9,m9)=R4;
          R4=R5+R1, R1=PM(i8,m8);
main:    R3=R0 AND R6, PM(i9,m9)=R4;
```

The first line above sets up the loop, the code repeating when the line labelled 'main' is executed. The second line adds the low A to D byte held in R3 to the sum array held in R1 and places the result in R4. The next sum along the waveform is then placed in R1. Notice that R1 is read on the first half clock cycle and written to in the second half. In the next line, the upper byte from the A to D is extracted from the 32 bit data memory word, a new A to D word is grabbed and placed in R0 and the result of the previous sum placed in program memory, pointed to by i9. As stated, i8 and i9 are initially set equal and point to the data storage area in program memory. Before the loop is entered i8 is used to read the program memory. This means that within the loop i8 points to a memory location one step ahead of i9. I9 places the new sum in memory while i8 get the previous sum.

The final two lines add the A to D high byte, held in R5, to the old waveform sum array, held in R1, retrieves the next waveform sum in the array and places it in R1, places the new A to D low byte in R3 and stores the result, held in R4, in the program memory array. Again each line in the program takes 1 clock cycles and shows how parallel operations take place.

The third section program is shown pictorially in figure 7.4 on the next page.

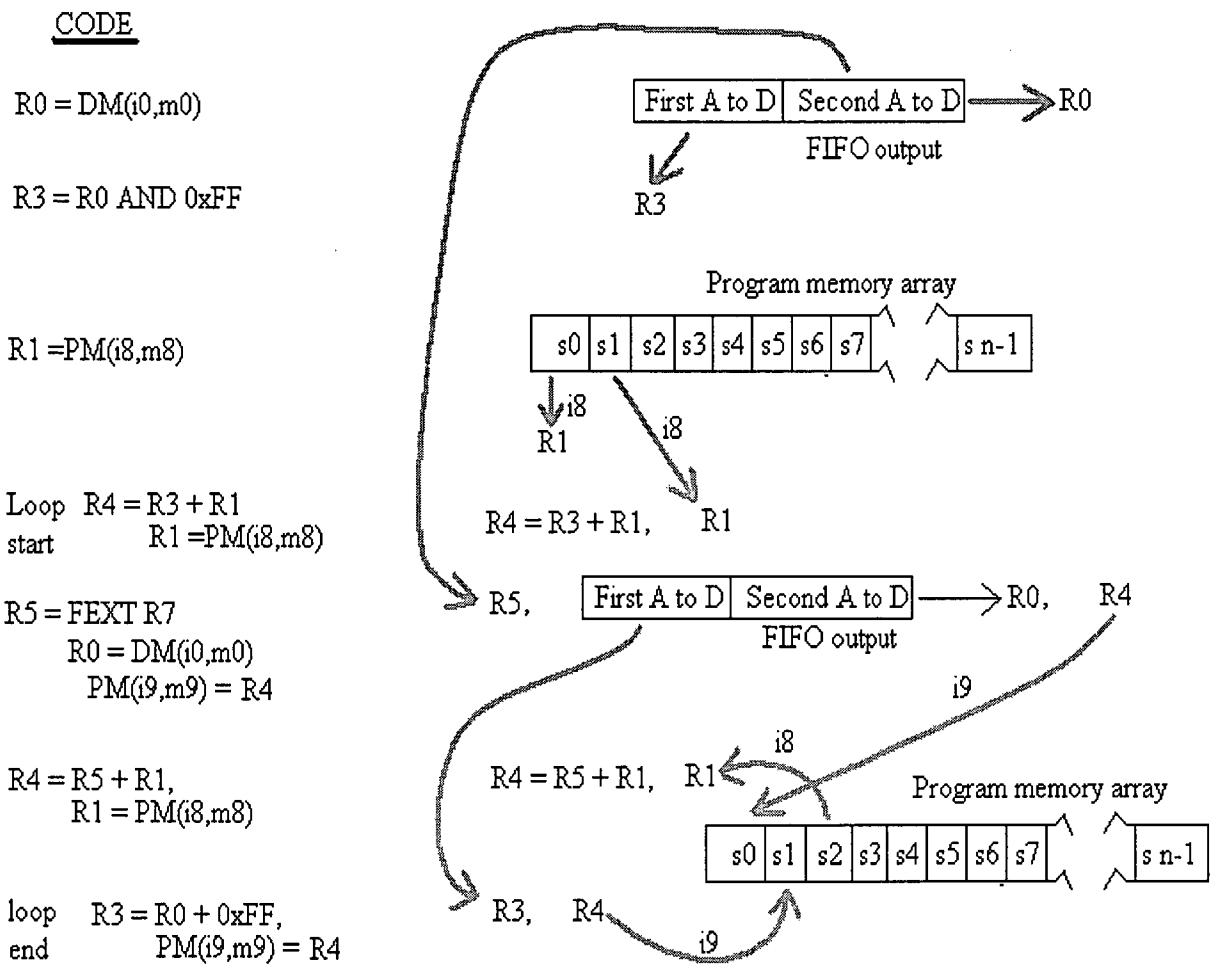


Figure 7.4. Pseudo code for third section of DSP code showing data movement.

Section four of the program adds the last waveform to be stacked in a similar manner as section three but this time the result is passed to the PC FIFO.

The outer loop in the program simply repeats the inner loop for the number of waveforms to be stacked.

The program discussion above shows some of the power of the 21020 i.e. simultaneous program execution from program memory cache, data memory access, program memory access, computational unit operation and loop control functions all in a single DSP clock cycle. The DSP operation timing is determined by counting the number of DSP clock periods. (The software simulator provides the number of cycles for each stage of the program).

7.3 PC logging software

An IBM PC controls the DSP and A to D operation. The PC was used as the logging platform and the radar user interface. The DSO operating system was chosen above a Windows operating system for the following reasons:

- The application was for a real-time logging operation. One program was required to run continuously until halted by the user. Multitasking the logging application with other programs may cause logging system timing problems.
- It is much easier to write programs under DOS that control PC hardware, such as IO ports, than with Windows.
- DOS programs require less system resources, such as memory and generally run faster then Windows programs.
- Multitasking was not required.

For the ice radar logging system positional and pressure information is logged along with the return ice radar echoes. For positional information a Trimble GPS receiver with a RS232 interface was

utilised. This unit outputs positional information in a binary format once per second. The barometer also has a RS232 interface and outputs pressure measurements in an ASCII format once per second. These two instruments were connected to the two RS232 ports of the PC.
The logging software, written in C, controlled the PC and DSP operation. The PC onboard clocks were used to control logging and display rates. As this was a real time application data logging was of primary importance. Data was displayed for the user to monitor and control the radar operation. The logged data was post-processed to obtain ice depths.

The timing of the DSP operations could be calculated knowing the number of DSP clock cycles required to complete the DSP program. As the maximum DSP program time was calculated to be less than one second the PC could pole the DSP and expect a new stacked waveform at least once a second. DSP FLAG0 was poled by the PC at the logging period selected by the user. Polling was chosen over interrupts for simplicity. The DSP can interrupt the PC and the PC could interrupt the DSP if error conditions occurred.
The receipt of RS232 data from the GPS or barometer caused an interrupt of the PC that responded by storing and displaying the data. The PC interrupt routines are kept small as large interrupt routines take too much time off the PC system causing the PC to crash.
Each waveform received from the DSP was from 3000 to 6000 samples long. As the screen for the display was only 600 points wide two display choices were available, a compressed version where groups of samples were averaged together and a display of a section of the waveform.

To start the logging program the user simply enters "logger" from the command prompt. On entry to the PC logging software the user can enter a filename of up to four letters long. A count is appended to this filename along with a .dat extension. When the logging file gets to 1Mbytes in size the file is closed and a new file is opened with the same four letters but the count is incremented. For example, watt0.dat, watt1.dat, watt3.dat etc. This way the files didn't get two large and could be transported by floppy disk if necessary. If a file count of 9999 is reached a default file of zzzz is started. It is unlikely in this application that 9999 files of the same name is required. After selecting the file name the user can enter header information which gives some information on the file. E.g. Lambert glacier from Davis base. The PC time and date and the header is logged once per file. The number of samples per waveform, the number of waveforms stacked, the logging period and sampling rate are entered by the user. This information is also logged along with GPS position and time and air pressure for each waveform.

Typically the radar was run for about 3 hours the time being limited by helicopter logistics. Number of samples per waveform was usually set to 3000, leading to 178 waveforms per 1 Mbyte file and about 65M of data per run. The data was transferred to a removable hard disk (a Jazz drive) after each flight.

The data collected was 'played back', using software written in C, to test the data collected. The format of the playback software is similar to the logging software. On entry to the playback software the operator enters the data file to be viewed, e.g. watt0.dat. PC time and date of the logging run is displayed along with a list of display options for the

header and waveform information. Once a waveform is displayed the ice depth can be obtained by placing a cursor at the bedrock echo point.

Most crashes during the writing of the PC programs were caused by one of two methods:

1. interrupts taking too much time off the system, for example an interrupt 'hogging' the PC while the PC system needs to refresh PC DRAM.
2. The amount of memory allocated to data arrays in the C program was not large enough. As the program exceeded the memory allocated to an array it would then overwrite the next location it came too. This would cause program or system memory to be overwritten causing a crash.

The logging and playback software programs, logger.c and retrieve.c and their executables are in the attached floppy. Both logging and retrieve software can be run from the floppy disk drive for inspection.

7.4 Radar results and example data

The ice radar was run successfully on the 1997/8 Australian Antarctic Division expedition to the Lambert Glacier region. The return echoes showed a bedrock reflection in ice as deep as 4km.
The picture, figure 7.5 on the next page, shows an example ice radar return echo.

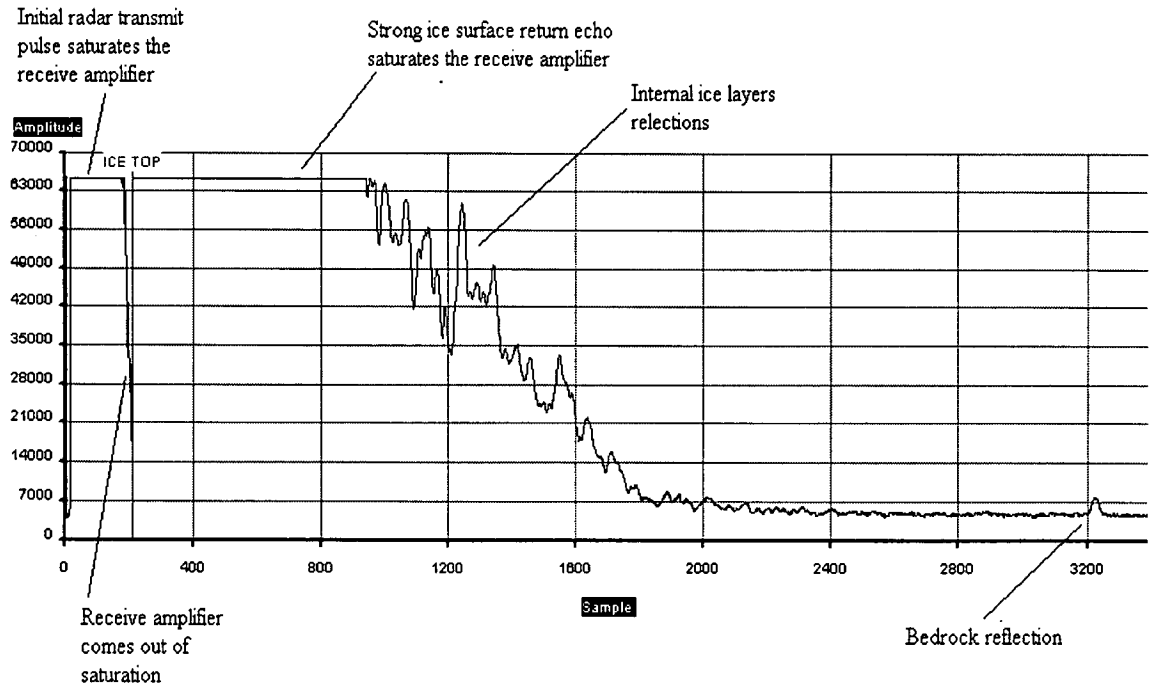


Figure 7.5. An example ice radar return echo.

The Y axis has stacked waveform amplitude and the X axis has the sample number (equivalent to time and therefore depth).

As the radar transmits the radar receivers are saturated and can take several micro seconds to come out of saturation. The transmitted pulse hits the ice surface, is partially reflected and as sufficient energy is reflected the receiver again saturates. It is important that the receiver be given time to come out of saturation after the radar transmission as the ice surface is the ice depth reference point. The operator must ensure the helicopter pilot is flying at sufficient height to allow the receiver to come out of saturation after radar transmission. The transmitted pulse propagates through the ice and is partially reflected at internal ice layer boundaries, producing the internal peaks as shown in figure 7.5. The pulse propagates as far as the ice-bedrock boundary and is reflected causing a relatively strong peak. The strength of the bedrock return

echo is a function of, not only depth, but also ice temperature, ice layering and other ice effects. Some external noise does enter the radar system but this was not unexpected, as a modern helicopter tends to generate significant amounts of EM noise caused by HF and RF communication systems, motors, digital systems, etc. This noise shows up as dc drifts and ac noise on the radar return echoes. The operator 'fine tunes' the system during the flight: shallow ice (1km) gives strong return echoes therefore fewer waveforms are stacked and fewer samples are required to capture the bedrock echo. Remember that the DSP output is limited to 16 bits. Stacking fewer waveforms will ensure the DSP does not 'saturate' (limit at FFFF hex) and lose internal ice layer reflections. As discerning bedrock echoes is a priority as the ice depth increases the number of waveforms stacked is increased to improve system signal to noise ratio.

Figure 7.6 on the next page shows several waveforms displayed together to give a bedrock profile. The Y axis has the sample number (equivalent to time and therefore depth) and the X axis has the waveform number. 145 waveforms are shown, representing about 3 minutes flight. These radar echoes were obtained on the deepest section of the glacier. The bottom echo cannot always be seen, but as shown is not necessarily due to ice depth. 1024 radar return echoes are stacked to produce each waveform.

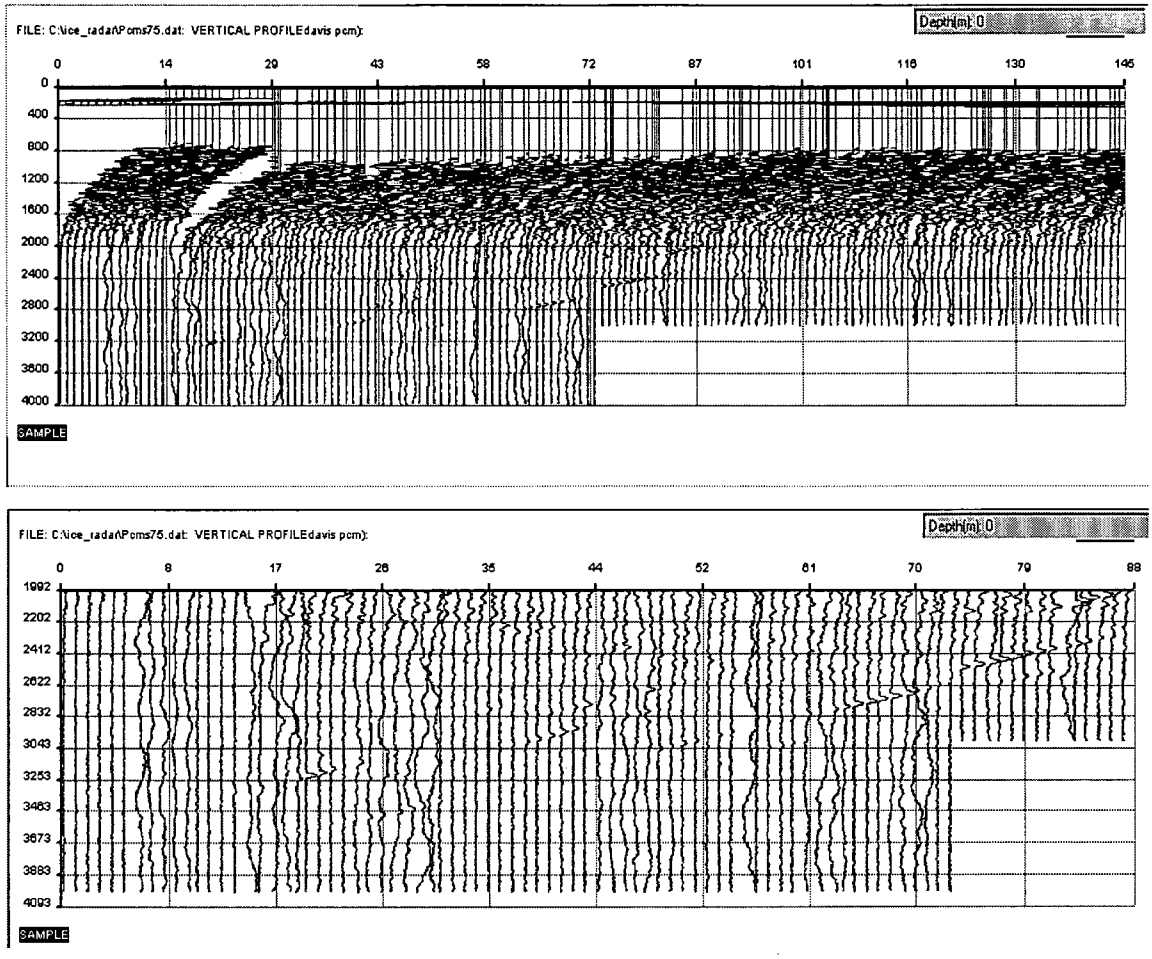


Figure 7.6. Ice radar bedrock profile. Full waveforms and zoomed-in section.

The greatest depth is at waveform 21. Notice that the return bedrock echo is stronger than the echo at waveform 25, a shallower section. After waveform 72 the user has chosen to reduce the number of samples recorded per waveform. The ac ripple and dc drift noise can also be seen.

Figure 7.7 on the next page shows a typical bedrock profile where the depth is approximately 1.5km. Here only 128 radar return echoes are stacked to produce each waveform.

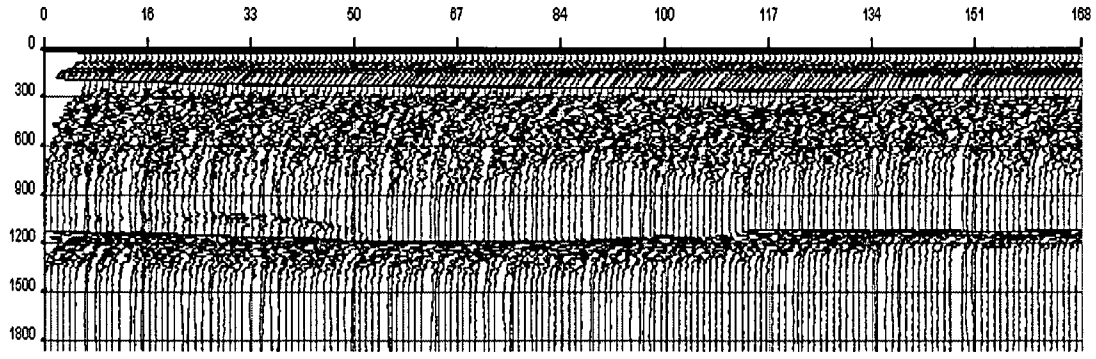


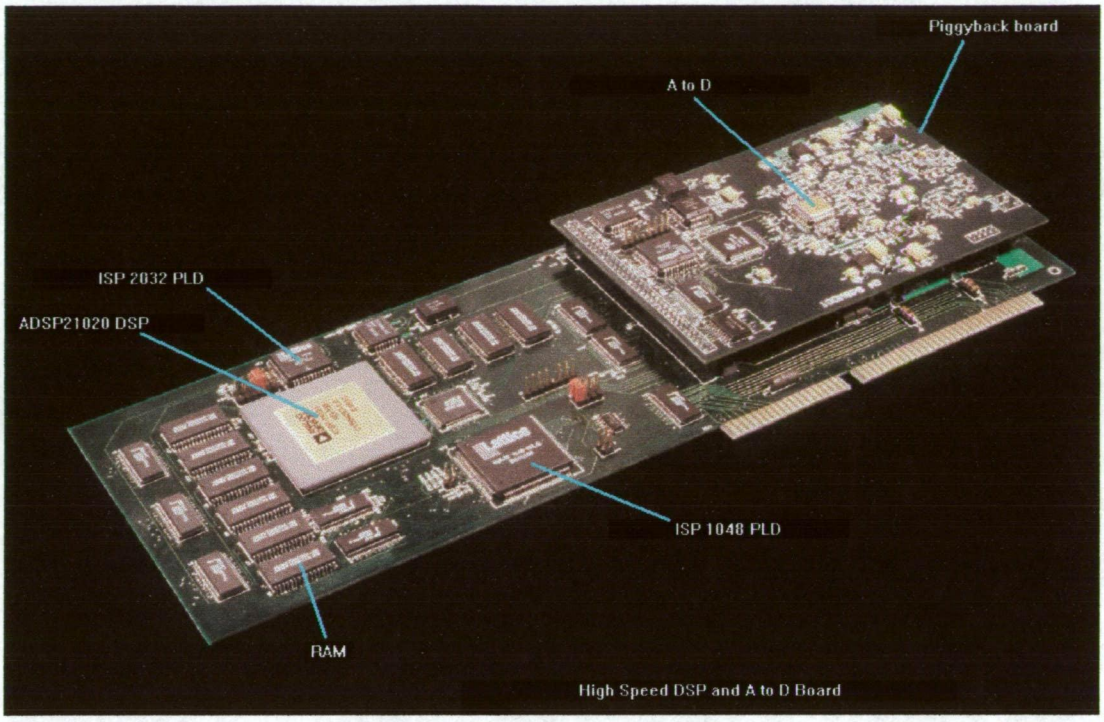
Figure 7.7. Ice radar bedrock profile.

The ice radar return stacked echoes, stored in files PCMS75.dat and PCMS25.dat, may be viewed with the playback software, retrive.exe. These files are on the attached floppy disk.

Chapter 8 Summary

High speed DSP and A to D can be used effectively in real time logging applications. The DSP can interface to different piggyback boards for different interface applications such as high accuracy A to D, digital IO and D to A. The ISA bus can be used to transfer data from the DSP to a PC platform without incurring DSP speed penalties.

The DSP and PC markets have moved on with larger DSP ICs becoming available and the PC PCI bus looking like superseding the ISA bus. The latest DSP microprocessor from Analog Devices, the 21060 has all the functions of the ADSP21020 but incorporates on board RAM and interfacing hardware suitable for multiprocessor applications. Improvements in the DSP market have tended towards using multiple processors to increase data computation rates. The same techniques as discussed above can be utilised with these new devices and standards to create new designs.



High speed DSP and A to D board as used in the ice radar logging application.

References

Narod, B.B. and Clarke, G.K.C., 1994. Miniature high-power impulse transmitter for radio-echo sounding, J. Glac., 40(134), p.190-194.

Radio-Glaciology. V.V. Bogorodsky, C.R.Bentley. P.E.Gudmandsen.

Description of British Antarctic Survey (BAS) Ice radar System. Hugh Corr

Description of German Institute for Polar Research Ice radar System. Dr Ludwig Hempel.

Ice Radar Recording, Data processing and Results from the Lambert Glacier Basin Traverses. Higham, Reynolds, Brocklesby and Allison. Terra Antartic, 1995, 2(1) 23-32.

Australian Antarctic Division Technical Note 12: Radar Ice Sounding at 100MHz. I. Bird, B. Morton and A. Robinson.

Radar Handbook. M Skolnik.

ANARE Helicopter Operations Handbook

ADSP 21020 Users manual. Analog Devices

ADSP21020 data sheet. Analog Devices

Analog Devices Application Note AN-235

MECL system design handbook. Motorola

High Speed Digital Design, A Handbook of Black Magic. Johnson and Graham.

Electronic Filter Design Handbook. Williams and Taylor.

Information, Transmission, Modulation and Noise. M. Schwartz

ADSP-210XX_ Applications Handbook. Analog Devices.

The IBM PC From the Inside Out. M. Sargent and R.L. Shoemaker.

PC programmers Bible. P. Norton.

PC Intern System Programming. M. Tischer.

PC Instrumentation for the 90s. Boston Technology.

Analog Devices Application Seminar Notes. Analog Devices.

Lattice Semiconductor application notes and data book. Lattice Semiconductor

Attachments

Logger.C PC logging program source codes on floppy disk

Retrieve.exe PC ice radar playback program executable on floppy disk

egavga.bgi graphics driver on floppy disk

Retrieve.C PC ice radar playback program source code on floppy disk

Logger.asm DSP logging software printout and on floppy disk

pcms75.dat Ice Radar sample data on floppy disk

Protel PCB files on floppy disk

PLD logic equations

List of test programs

PLD logic equations.

******DSP 2032 cell equations**********

GLB A1
CELL EQUATIONS:

BGG = BG_;

GLB A2
CELL EQUATIONS:

NOTBG = (!BGG) # (!BG_);
BG = (BGG) # (BG_);

GLB A3
CELL EQUATIONS:

DLCE = (DMS2 & BG) # (NOTBG) # (DMWR & BG);
DLOE = (DMRD & BG) # (DMS2 & BG) # (NOTBG);
FFWEN = (NOTBG) # (DMWR & BG) # (DMS1 & BG);

GLB A7
CELL EQUATIONS:

PLWR = (PMWR & BG) # (NOTBG) # (PMS1 & BG);
PLOE = (R & NOTBG) # (!PMRD & !PMS1 & BG);

**** PC interface 1048 cell equations **********

GLB A2
CELL EQUATIONS:

IRQ.D = FLGOP_CTL;
IRQ.PTCLK = FLAG0;
IRQ.RE = IRQRST;

```
  GLB A4
  CELL EQUATIONS:

R = A0 & A1 & A2 & !A3 & A4 & BOARD;
SPARE = RESDRV;

  GLB A5
  CELL EQUATIONS:

PSAB = VCC;
PSBA = OUT0;
POE3 = !IOR & !A0 & !A1 & A2 & !A3 & !A4 & BOARD;

  GLB A6
  CELL EQUATIONS:

PW2 = !IOW & !A0 & !A1 & A2 & !A3 & A4 & BOARD;
PW1 = !IOW & !A0 & A1 & !A2 & !A3 & A4 & BOARD;
POE1 = !IOR & !A0 & !A1 & !A2 & !A3 & !A4 & BOARD;

  GLB A7
  CELL EQUATIONS:

OUT0_ = OUT0;

  GLB A5_part1
  CELL EQUATIONS:

PW3 = !IOW & !A0 & A1 & A2 & !A3 & A4 & BOARD;

  GLB A6_part1
  CELL EQUATIONS:

POE2 = !IOR & !A0 & A1 & !A2 & !A3 & !A4 & BOARD;

  GLB B0
  CELL EQUATIONS:
IN3_ = IN3;
IN2_ = IN2;
```

```
IN1_ = IN1;
IN0_ = IN0;

  GLB B1
  CELL EQUATIONS:

FFRST.D = IN3_;
FFRST.PTCLK = !(!IOW & A0 & A1 & !A2 & !A3 & A4 & BOARD);
DSPIRQ.D = IN2_;
DSPIRQ.PTCLK = !(!IOW & A0 & A1 & !A2 & !A3 & A4 & BOARD);
FLAG1.D = IN1_;
FLAG1.PTCLK = !(!IOW & A0 & A1 & !A2 & !A3 & A4 & BOARD);

  GLB B2
  CELL EQUATIONS:

S0_1.D = IN1_;
S0_1.PTCLK = !(!IOW & A0 & !A1 & A2 & !A3 & A4 & BOARD);
FIFO_CTL.D = IN0_;
FIFO_CTL.PTCLK = !(!IOW & A0 & !A1 & A2 & !A3 & A4 & BOARD);
SFT_RST.D = IN2_;
SFT_RST.PTCLK = !(!IOW & A0 & !A1 & A2 & !A3 & A4 & BOARD);
S0.D = IN1_;
S0.PTCLK = !(!IOW & A0 & !A1 & A2 & !A3 & A4 & BOARD);

  GLB B3
  CELL EQUATIONS:

DOE = !A0 & A1 & A2 & A3 & !A4 & !A5 & !A6 & !A7 & A8 & A9 & !AEN;

  GLB B4
  CELL EQUATIONS:

B12.D = IN12_;
B12.PTCLK = !(!IOW & !A0 & !A1 & !A2 & !A3 & A4 & BOARD);
B13.D = IN13_;
B13.PTCLK = !(!IOW & !A0 & !A1 & !A2 & !A3 & A4 & BOARD);
B14.D = IN14_;
B14.PTCLK = !(!IOW & !A0 & !A1 & !A2 & !A3 & A4 & BOARD);
```

```
B15.D = IN15_;
B15.PTCLK = !(!IOW & !A0 & !A1 & !A2 & !A3 & A4 & BOARD);

  GLB B5
  CELL EQUATIONS:

B8.D = IN8_;
B8.PTCLK = !(!IOW & !A0 & !A1 & !A2 & !A3 & A4 & BOARD);
B9.D = IN9_;
B9.PTCLK = !(!IOW & !A0 & !A1 & !A2 & !A3 & A4 & BOARD);
B10.D = IN10_;
B10.PTCLK = !(!IOW & !A0 & !A1 & !A2 & !A3 & A4 & BOARD);
B11.D = IN11_;
B11.PTCLK = !(!IOW & !A0 & !A1 & !A2 & !A3 & A4 & BOARD);

  GLB B6,
  CELL EQUATIONS:

B0.D = IN0_;
B0.PTCLK = !(!IOW & !A0 & !A1 & !A2 & !A3 & A4 & BOARD);
B1.D = IN1_;
B1.PTCLK = !(!IOW & !A0 & !A1 & !A2 & !A3 & A4 & BOARD);
B2.D = IN2_;
B2.PTCLK = !(!IOW & !A0 & !A1 & !A2 & !A3 & A4 & BOARD);
B3.D = IN3_;
B3.PTCLK = !(!IOW & !A0 & !A1 & !A2 & !A3 & A4 & BOARD);

  GLB B7
  CELL EQUATIONS:

B7.D = IN7_;
B7.PTCLK = !(!IOW & !A0 & !A1 & !A2 & !A3 & A4 & BOARD_1);
B6.D = IN6_;
B6.PTCLK = !(!IOW & !A0 & !A1 & !A2 & !A3 & A4 & BOARD_1);
B5.D = IN5_;
B5.PTCLK = !(!IOW & !A0 & !A1 & !A2 & !A3 & A4 & BOARD_1);
B4.D = IN4_;
B4.PTCLK = !(!IOW & !A0 & !A1 & !A2 & !A3 & A4 & BOARD_1);
```

```
  GLB B1_part1
  CELL EQUATIONS:

BR.D = IN0_;
BR.PTCLK = !(!IOW & A0 & A1 & !A2 & !A3 & A4 & BOARD);

  GLB B2_part1
  CELL EQUATIONS:

FLGOP_CTL.D = IN3_;
FLGOP_CTL.PTCLK = !(!IOW & A0 & !A1 & A2 & !A3 & A4 & BOARD);

  GLB C0
  CELL EQUATIONS:

DRQ = !A1 & !A2 & A3 & A4 & BOARD & A0;
IO16 = !A0 & !A5 & !A6 & !A7 & A8 & A9 & !AEN;

  GLB C1
  CELL EQUATIONS:

DIRIO = !A4;
OEIO = (!A5 & !A6 & !A7 & A8 & A9 & !AEN & A4) # (!IOR & BOARD & !A4);

  GLB C2
  CELL EQUATIONS:

IN4_ = IN4;

  GLB C3
  CELL EQUATIONS:

IN11_ = IN11;
IN10_ = IN10;
IN9_ = IN9;
IN8_ = IN8;

  GLB C4
```

```
  CELL EQUATIONS:

IN15_ = IN15;
IN14_ = IN14;
IN13_ = IN13;
IN12_ = IN12;

   GLB C5
  CELL EQUATIONS:

BOARD_1 = !A5 & !A6 & !A7 & A8 & A9 & !AEN;
BOARD = !A5 & !A6 & !A7 & A8 & A9 & !AEN;

   GLB C6
  CELL EQUATIONS:

FFCNTRST.D = IN4_;
FFCNTRST.PTCLK = !(!IOW & A0 & A1 & !A2 & !A3 & A4 & BOARD_1);

  GLB C7
  CELL EQUATIONS:

IRQRST = (!IOW & A0 & A1 & !A2 & A3 & A4 & BOARD) # (RESDRV);

  GLB C0_part1,
  CELL EQUATIONS:

NOWS = VCC;

   GLB C2_part1
  CELL EQUATIONS:

IN7_ = IN7;
IN6_ = IN6;
IN5_ = IN5;


 GLB D0
  CELL EQUATIONS:
```

```
CLK = !A0 & A1 & !A2 & !A3 & A4 & BOARD & !BALE_;
CD = !A0 & A1 & !A2 & A3 & A4 & BOARD;
EN = !A0 & !A1 & A2 & !A3 & !A4 & BOARD;

  GLB D1
  CELL EQUATIONS:

BALE_ = BALE;

  GLB D7
  CELL EQUATIONS:

PMWR = !OUT0 & !IOW & A0 & A1 & A2 & !A3 & A4 & BOARD;
ADD_W = !IOW & A0 & !A1 & A2 & A3 & A4 & BOARD;
PMRD = (!OUT0 & !IOR & !A0 & !A1 & !A3 & !A4 & BOARD) # (!OUT0 & !IOR & !A0
& A1 & !A2 & !A3 & !A4 & BOARD);

  GLB E0
  CELL EQUATIONS:

ADD3 = (!S0 & Q3) # (S0 & B3);
ADD2 = (!S0 & Q2) # (S0 & B2);
ADD1 = (!S0 & Q1) # (S0 & B1);
ADD0 = (!S0 & Q0) # (S0 & B0);

  GLB E1
  CELL EQUATIONS:

ADD7 = (!S0 & Q7) # (S0 & B7);
ADD6 = (!S0 & Q6) # (S0 & B6);
ADD5 = (!S0 & Q5) # (S0 & B5);
ADD4 = (!S0 & Q4) # (S0 & B4);


 GLB E2
  CELL EQUATIONS:

ADD11 = (!S0 & Q11) # (S0 & B11);
```

```
ADD10 = (!S0 & Q10) # (S0 & B10);
ADD9 = (!S0 & Q9) # (S0 & B9);
ADD8 = (!S0 & Q8) # (S0 & B8);

 GLB E3
  CELL EQUATIONS:

ADD15 = (!S0 & Q15) # (S0 & B15);
ADD14 = (!S0 & Q14) # (S0 & B14);

 GLB E4
  CELL EQUATIONS:

Q3.D = (Q2.Q & Q1.Q & Q0.Q & EN) $$ (Q3.Q);
Q3.CLK = CLK;
Q3.RE = CD;
Q2.D = (Q1.Q & Q0.Q & EN) $$ (Q2.Q);
Q2.CLK = CLK;
Q2.RE = CD;
Q1.D = (Q0.Q & EN) $$ (Q1.Q);
Q1.CLK = CLK;
Q1.RE = CD;
Q0.D = (Q0.Q) $$ (EN);
Q0.CLK = CLK;
Q0.RE = CD;

  GLB E5
  CELL EQUATIONS:

Q7.D = (Q6.Q & Q5.Q & Q4.Q & Q3 & Q2 & Q1 & Q0 & EN) $$ (Q7.Q);
Q7.CLK = CLK;
Q7.RE = CD;
Q6.D = (Q5.Q & Q4.Q & Q3 & Q2 & Q1 & Q0 & EN) $$ (Q6.Q);
Q6.CLK = CLK;
Q6.RE = CD;
Q5.D = (Q4.Q & Q3 & Q2 & Q1 & Q0 & EN) $$ (Q5.Q);
Q5.CLK = CLK;
Q5.RE = CD;
Q4.D = (Q3 & Q2 & Q1 & Q0 & EN) $$ (Q4.Q);
```

```
Q4.CLK = CLK;
Q4.RE = CD;

 GLB E6
  CELL EQUATIONS:

Q11.D = (Q10.Q & Q9.Q & Q8.Q & Q7 & Q6 & Q5 & Q4 & Q3 & Q2 & Q1 & Q0 & EN)
$$ (Q11.Q);
Q11.CLK = CLK;
Q11.RE = CD;
Q10.D = (Q9.Q & Q8.Q & Q7 & Q6 & Q5 & Q4 & Q3 & Q2 & Q1 & Q0 & EN) $$
(Q10.Q);
Q10.CLK = CLK;
Q10.RE = CD;
Q9.D = (Q8.Q & Q7 & Q6 & Q5 & Q4 & Q3 & Q2 & Q1 & Q0 & EN) $$ (Q9.Q);
Q9.CLK = CLK;
Q9.RE = CD;
Q8.D = (Q7 & Q6 & Q5 & Q4 & Q3 & Q2 & Q1 & Q0 & EN) $$ (Q8.Q);
Q8.CLK = CLK;
Q8.RE = CD;

  GLB E7
  CELL EQUATIONS:

Q15.D = (Q14.Q & Q13.Q & Q12.Q & Q11 & Q10 & Q9 & Q8 & Q7 & Q6 & Q5 & Q4 &
Q3 & Q2 & Q1 & Q0 & EN) $$ (Q15.Q);
Q15.CLK = CLK;
Q15.RE = CD;
Q14.D = (Q13.Q & Q12.Q & Q11 & Q10 & Q9 & Q8 & Q7 & Q6 & Q5 & Q4 & Q3 & Q2
& Q1 & Q0 & EN) $$ (Q14.Q);
Q14.CLK = CLK;
Q14.RE = CD;
Q13.D = (Q12.Q & Q11 & Q10 & Q9 & Q8 & Q7 & Q6 & Q5 & Q4 & Q3 & Q2 & Q1 &
Q0 & EN) $$ (Q13.Q);
Q13.CLK = CLK;
Q13.RE = CD;
Q12.D = (Q11 & Q10 & Q9 & Q8 & Q7 & Q6 & Q5 & Q4 & Q3 & Q2 & Q1 & Q0 & EN)
$$ (Q12.Q);
Q12.CLK = CLK;
```

```
Q12.RE = CD;

  GLB E3_part1
  CELL EQUATIONS:
ADD12 = (!S0_1 & Q12) # (S0_1 & B12);
ADD13 = (!S0_1 & Q13) # (S0_1 & B13);

  GLB F0
  CELL EQUATIONS:

OUT3 = BG;
OUT2 = FFEF;
OUT1 = FLAG0;
OUT0 = BG;

  GLB F1
  CELL EQUATIONS:

OUT7 = BG;
OUT6 = EFC;
OUT5 = EFB;
OUT4 = EFA;

  GLB F2
  CELL EQUATIONS:

OUT11 = BG;
OUT10 = BG;
OUT9 = BG;
OUT8 = BG;

  GLB F3
  CELL EQUATIONS:

OUT15 = BG;
OUT14 = BG;
OUT13 = BG;
OUT12 = BG;
```

```
GLB F4
CELL EQUATIONS:

DW2 = !IOW & !A0 & A1 & A2 & A3 & A4 & BOARD;
DW1 = !IOW & !A0 & !A1 & A2 & A3 & A4 & BOARD;
DOE2 = !IOR & !A0 & A1 & !A2 & A3 & !A4 & BOARD;
DOE1 = !IOR & !A0 & !A1 & !A2 & A3 & !A4 & BOARD;

GLB F5
CELL EQUATIONS:

PCFFCLK_1 = (!DAK & !FIFO_CTL & !IOR) # (FIFO_CTL & !IOR & !A0 & A1 & A2 &
!A3 & !A4 & BOARD);
PCFFEN = (!DAK & !FIFO_CTL) # (FIFO_CTL & !A0 & A1 & A2 & !A3 & !A4 &
BOARD);
PCFFCLK = (!DAK & !FIFO_CTL & !IOR) # (FIFO_CTL & !IOR & !A0 & A1 & A2 &
!A3 & !A4 & BOARD);

GLB F6
CELL EQUATIONS:

EFA.D = (EFB.Q & EFA.Q & EFC.Q & !FFEF) # (!EFA.Q & !FFEF) # (EFA.Q &
FFEF);
EFA.PTCLK = PCFFCLK;
EFA.RE = FFCNTRST;
EFB.D = !((!EFB.Q & !EFA.Q) # (!EFB.Q & FFEF) # (EFB.Q & EFA.Q & !EFC.Q &
!FFEF));
EFB.PTCLK = PCFFCLK;
EFB.RE = FFCNTRST;
EFC.D = (EFB.Q & EFA.Q & !FFEF) # (EFC.Q);
EFC.PTCLK = PCFFCLK;
EFC.RE = FFCNTRST;

GLB F7
CELL EQUATIONS:

DCE2 = !BALE & !A0 & A1 & A2 & A3 & A4 & BOARD;
DCE1 = !BALE & !A0 & !A1 & A2 & A3 & A4 & BOARD;
PCFFOE = (!DAK & !FIFO_CTL & !IOR) # (FIFO_CTL & !IOR & !A0 & A1 & A2 & !A3
```

```
& !A4 & BOARD);

  GLB F7_part1
  CELL EQUATIONS:

DSP_RST = SFT_RST & !RESDRV;



********* A to D 1 2032 cell equations **********

  GLB A0
  CELL EQUATIONS:

WEN    =      !Q9.Q & !Q10.Q & !Q11.Q & CO & !Q8;
Q9.D   =      (Q9.Q & !LD) $$ ((BDM31 & LD) # (Q11.Q & CO & !Q8 & !LD) #
                        (Q10.Q & CO & !Q8 & !LD) # (Q9.Q & CO & !Q8 & !LD));
Q9.CLK =      I_CLK;
Q9.RE  =      BDM38;
Q10.D =       (Q10.Q & !LD) $$ ((!Q9.Q & Q11.Q & CO & !Q8 & !LD) #
                        (!Q9.Q & Q10.Q & CO & !Q8 & !LD) #  (BDM32 & LD));
Q10.CLK =     I_CLK;
Q10.RE =      BDM38;
Q11.D =       (Q11.Q & !LD) $$ ((!Q9.Q & !Q10.Q & Q11.Q & CO & !Q8 & !LD) #
                        (BDM33 & LD));
Q11.CLK =     I_CLK;
Q11.RE =      BDM38;

  GLB A1
  CELL EQUATIONS:

Q4.D   =      (Q4.Q & !LD) $$ ((!Q0 & !Q1 & !Q2 & !Q3 & !WEN & !LD) #
                 (BDM26 & LD));
Q4.CLK =      I_CLK;
Q4.RE =       BDM38;
Q5.D =        (Q5.Q & !LD) $$ ((!Q4.Q & !Q0 & !Q1 & !Q2 & !Q3 & !WEN & !LD) #
                        (BDM27 & LD));
Q5.CLK =      I_CLK;
```

```
Q5.RE =         BDM38;
Q6.D =          (Q6.Q & !LD) $$
                ((!Q4.Q & !Q5.Q & !Q0 & !Q1 & !Q2 & !Q3 & !WEN & !LD) #
                        (BDM28 & LD));
Q6.CLK =        I_CLK;
Q6.RE =         BDM38;
Q7.D =          Q7.Q & !LD) $$ ((!Q4.Q & !Q5.Q & !Q6.Q & !Q0 & !Q1 & !Q2 & !Q3 & !WEN
& !LD)                  # (BDM29 & LD));
Q7.CLK =        I_CLK;
Q7.RE =         BDM38;

  GLB A2
  CELL EQUATIONS:

Q0.D =          (Q0.Q & !LD) $$ (!WEN & !LD);
Q0.CLK =        I_CLK;
Q0.RE =         BDM38;
Q1.D =          (Q1.Q & !LD) $$ (!Q0.Q & !WEN & !LD);
Q1.CLK =        I_CLK;
Q1.RE =         BDM38;
Q2.D =          (Q2.Q & !LD) $$ ((!Q0.Q & !Q1.Q & !WEN & !LD) # (BDM24 & LD));
Q2.CLK =        I_CLK;
Q2.RE =         BDM38;
Q3.D =          (Q3.Q & !LD) $$ ((!Q0.Q & !Q1.Q & !Q2.Q & !WEN & !LD) # (BDM25 &
LD));
Q3.CLK =        I_CLK;
Q3.RE =         BDM38;

  GLB A3
  CELL EQUATIONS:

EDGE =          T3 & !T2 & WEN;
!LSBOE =        !DMRD & !DMS3;
Q8.D =          (Q8.Q & !LD) $$ ((CO & !WEN & !LD) # (BDM30 & LD));
Q8.CLK =        I_CLK;
Q8.RE =         BDM38;
FLG3.D =        FLG_3;
FLG3.CLK =      I_CLK;
FLG3.RE =       BDM38;
```

```
  GLB A4
  CELL EQUATIONS:

!FLG_3 =          !FLAG3__.Q & !FLAG3_.Q;
!MSBCLK =         !DMWR & !DMS3;
FLAG3_.D =        FLAG3;
FLAG3_.PTCLK =          !OSC50;
FLAG3__.D =       FLAG3_.Q;
FLAG3__.PTCLK =         !OSC50;

   GLB A5
  CELL EQUATIONS:

BLOCK.D =         (!FLG3 & EDGE) # (!FLG3 & BDM38) # (BLOCK.Q & !FLG3);
BLOCK.CLK =       I_CLK;
T3.D =            TRIG;
T3.CLK =          I_CLK;
T2.D =            T3.Q;
T2.CLK =          I_CLK;
LD =              !BLOCK.Q & !FLG3 & EDGE;

  GLB A6
  CELL EQUATIONS:

ADCLK =           (BDM37 & OSC50) # (OSC25.Q & !BDM37);
OSC25.D =         !OSC25.Q;
OSC25.PTCLK =OSC50;
TAGIRQ =          (!WEN & !BDM34) # (!TAG_ & !TAG & BDM34);

  R GLB A7
  CELL EQUATIONS:

TAG.D =           LD;
TAG.CLK =         I_CLK;
CO =              !Q0 & !Q1 & !Q2 & !Q3 & !Q4 & !Q5 & !Q6 & !Q7;
!REN =            !DMRD & !DMS3;
TAG_.D =          TAG.Q;
TAG_.CLK =        I_CLK;
```

*****A to D 2 2032 cell equations**********

```
GLB A0
CELL EQUATIONS:

FFEN.D =        Q0;
FFEN.CLK =      I_CLK;
Q9.D =          (Q9.Q & !LD & !BDM38) $$ ((!Q8 & !CO2 & CO & !LD & !BDM38) #
                                        (BDM30 & LD & !BDM38));
Q9.CLK =        I_CLK;
Q10.D =         (Q10.Q & !LD & !BDM38) $$ ((!Q9.Q & !Q8 & !CO2 & CO & !LD & !BDM38)
#  (BDM31 & LD & !BDM38));
Q10.CLK =       I_CLK;
Q11.D =         (Q11.Q & !LD & !BDM38) $$ ((!Q9.Q & !Q10.Q & !Q8 & !CO2 & CO & !LD &
!BDM38) #  (BDM32 & LD & !BDM38));
Q11.CLK =       I_CLK;

GLB A1
CELL EQUATIONS:

Q4.D =          (Q4.Q & !LD & !BDM38) $$ ((!Q0 & !Q1 & !Q2 & !Q3 & !CO & !LD &
!BDM38) #   (!Q0 & !Q1 & !Q2 & !Q3 & !CO2 & !LD & !BDM38) #  (BDM25 & LD &
!BDM38));
Q4.CLK =        I_CLK;
Q5.D =          (Q5.Q & !LD & !BDM38) $$ ((!Q4.Q & !Q0 & !Q1 & !Q2 & !Q3 & !CO & !LD &
!BDM38) #   (!Q4.Q & !Q0 & !Q1 & !Q2 & !Q3 & !CO2 & !LD & !BDM38) #  (BDM26 & LD &
!BDM38));
Q5.CLK =        I_CLK;
Q6.D =          (Q6.Q & !LD & !BDM38) $$
                ((!Q4.Q & !Q5.Q & !Q0 & !Q1 & !Q2 & !Q3 & !CO & !LD & !BDM38) #
(!Q4.Q & !Q5.Q & !Q0 & !Q1 & !Q2 & !Q3 & !CO2 & !LD & !BDM38) #
 (BDM27 & LD & !BDM38));
Q6.CLK =        I_CLK;
Q7.D =          (Q7.Q & !LD & !BDM38) $$
((!Q4.Q & !Q5.Q & !Q6.Q & !Q0 & !Q1 & !Q2 & !Q3 & !CO & !LD & !BDM38) #
(!Q4.Q & !Q5.Q & !Q6.Q & !Q0 & !Q1 & !Q2 & !Q3 & !CO2 & !LD & !BDM38) #
```

```
 (BDM28 & LD & !BDM38));
Q7.CLK =        I_CLK;

  GLB A2
  CELL EQUATIONS:

Q0.D =          (Q0.Q & !LD & !BDM38) $$ ((!CO2 & !LD & !BDM38) # (!CO & !LD &
!BDM38));
Q0.CLK =        I_CLK;
Q1.D =          (Q1.Q & !LD & !BDM38) $$ ((!Q0.Q & !CO2 & !LD & !BDM38) #
                        (!Q0.Q & !CO & !LD & !BDM38));
Q1.CLK =        I_CLK;
Q2.D =          (Q2.Q & !LD & !BDM38) $$ ((!Q0.Q & !Q1.Q & !CO2 & !LD & !BDM38) #
                        (!Q0.Q & !Q1.Q & !CO & !LD & !BDM38));
Q2.CLK =        I_CLK;
Q3.D =          (Q3.Q & !LD & !BDM38) $$ ((!Q0.Q & !Q1.Q & !Q2.Q & !CO & !LD &
!BDM38) #  (!Q0.Q & !Q1.Q & !Q2.Q & !CO2 & !LD & !BDM38) #  (BDM24 & LD &
!BDM38));
Q3.CLK =        I_CLK;

  GLB A3
  CELL EQUATIONS:

EDGE.D =        T3 & !T2 & CO2 & CO;
EDGE.CLK =      I_CLK;
                !LSBOE = !DMRD & !DMS3;
Q8.D =          (Q8.Q & !LD & !BDM38) $$ ((!CO2 & CO & !LD & !BDM38) # (BDM29 & LD
& !BDM38));
Q8.CLK =        I_CLK;
Q12.D =         (Q12.Q & !LD & !BDM38) $$
                ((!Q8.Q & !Q9 & !Q10 & !Q11 & !CO2 & CO & !LD & !BDM38) #
                        (BDM33 & LD & !BDM38));
Q12.CLK =       I_CLK;

  GLB A4
  CELL EQUATIONS:

CO =            !Q0 & !Q1 & !Q2 & !Q3 & !Q4 & !Q5 & !Q6 & !Q7;
!MSBCLK =       !DMWR & !DMS3;
```

```
FLG3.D =        FLAG3;
FLG3.CLK =      I_CLK;

  GLB A5
  CELL EQUATIONS:

BLOCK.D =       (!FLG3 & EDGE) # (!FLG3 & BDM38) # (BLOCK.Q & !FLG3);
BLOCK.CLK =     I_CLK;
T3.D =          TRIG;
T3.CLK =        I_CLK;
T2.D =          T3.Q;
T2.CLK =        I_CLK;
LD =            !BLOCK.Q & !FLG3 & EDGE;

    GLB A6
    CELL EQUATIONS:

TAG_.D =        TAG.Q;
TAG_.CLK =      I_CLK;
TAG.D =                 LD;
TAG.CLK =       I_CLK;
TAG___.D =      TAG__;
TAG___.CLK =    I_CLK;
TAGIRQ =        (!TAG___.Q & !TAG_.Q & !TAG.Q & !TAG__ & BDM34) # (CO2 & CO &
!BDM34);

  GLB A7
  CELL EQUATIONS:

TAG__.D =       TAG_;
TAG__.CLK =     I_CLK;
CO2 =           !Q8 & !Q9 & !Q10 & !Q11 & !Q12;
!REN =          !DMRD & !DMS3;
```

Test Programs

DWNLD.C	Downloads a program to DSP
NOPIDL.C	Downloads DSP idle program.
IOTST2.C	Used to set IO lines to test 1048
PCIOTST.C	Used to exercise IO lines.
PCPMDS.C	Exercises ISA bus IO. DSP held reset and BR asserted
PMFLE2.C	Downloads a DSP file entered from command line
PMTST4.C	Downloads a DSP file with NOPS downloaded
PM_DM2.C	Downloads program and looks at DM latch as well as PM latch (with test7).
FFTST.C	Used to test FIFO.
MEZTST.C	Used to test piggyback board
INT.C	Used to test PC interrupt by DSP
DSPINT.C	Tests interrupt to DSP.
test1.asm	Adds 2 numbers (2, 3) from PM and ops to PM latch
test2.asm	Adds 2 numbers from (2, 3) PM and ops to PM
test3.asm	Adds 2 numbers from PM (4, 7) and ops to PM latch
test4.asm	Ops 2 numbers from PM to DM reads back add and op result to PM latch
test5.asm	Ops 2 numbers from PM to DM reads back add and op result to PM latch
test6.asm	Ops 2 numbers from PM to DM reads back add and op result to PM latch
and	DM latch
test7.asm	Ops 2 numbers from PM to DM reads back, adds and op result to PM latch and left shifted version to DM latch. (0x01020304, 0x05060708). answer 0608 0A0C, C101418
test8.asm	Out puts first, second then first number to FIFO
test9.asm	Out puts first, second and third number to FIFO (abcd, ef12, 0000).
test10.asm	Toggles flag0.
test11.asm	Tests mezzanine AD.
idle.asm	sets DSP in IDLE.
test15.asm	tests IRQ0

ADSP21020 DSP logging program

```
/*                          AD and SIGNAL AVERAGING                    */
/*************************************************************/
/*                                                                     */
/*  ADSP21020 assembler program to drive A to D and average/stack      */
/*  radar digitised waveforms.                                         */
/*                                                                     */
/*************************************************************/


.SEGMENT   /PM     pm_code;

#include "def21020.h"

/************************* set-up ********************************/

begin:    pmwait=0x0021;
          dmwait=0x8421;
          BIT SET mode2 FLG3O;  /*  Set flag3 as op */
          BIT CLR ASTAT FLG3;   /*  flag3 = 0 */
          BIT SET mode2 FLG0O;  /*  Set flag0 as op */
          BIT CLR ASTAT FLG0;   /*  flag0 = 0 */
          BIT CLR mode2 TIMEN;  /*  disable timer */



          i2=dio;          /* i2 = data memory to PC latch */
          m2=0;
          l2=0;
          b2=dio;

          i0=atod;         /*  set up DAG1_0 for control op */
          m0=0;            /*  i0 = piggyback */
          l0=0;            /*  needs to be set to 0 to avoid circular buffering */
          b0=atod;
```

```
i1=doutff;        /* i1 = DSP to PC FIFO */
m1=0;
l1=0;
b1=doutff;

/* set up constants */
R6=0XFF;
R7=0X208;
R12=0x80000000;
R13=0x40000000;
R3=2;
R9=999;

/* read PC instructions and reset the A to D controls */
R15 =DM(dio);          /* Read PC comms latch.  set up AD = (0,0, ad set-up) */
R14 =R15 OR R13;       /* R13 = 40000000,  R14 = 01,adsetup  */
DM(i0,m0)=R14;         /* clear fifo & 2032  */
R14 =R14 OR R12;       /* R12 = 80000000,  R14 = 11,adsetup  */
DM(i0,m0)=R14;         /* release fifo clear  */
R14 =R15 OR R12;       /* R14 = 10,adsetup  */
DM(i0,m0)=R14;         /* release 2032 clear  */
R11=0x30;              /* message 3.  PC not keeping up  */

/* extract number of samples (min 2500) per waveform info */
/* from R15 (= PC instructions) */
R12 = FEXT R15 BY 16:10;   /* number of samples for DSP routine  */
R12=LSHIFT R12 BY 2;       /* number of samples/2 */
R13 = R12 - R9;            /* number of samples/2 - 999*/
R9 = 4;
R10 = R12 - R9;            /* number of samples/2 - 4 */
R12=LSHIFT R12 BY 1;       /* R12 = 12 bit number of samples  */

i8=pmdata;                 /* set PM pointer to PM data area */
m8=1;
l8=6200;        /* maximum number of samples = maximum loop size*/
b8=pmdata;

i9=pmdata;      /* i8 and i9 are the same PM memory locations */
```

```
            m9=1;
            l9=6200;          /* max number of samples */
            b9=pmdata;

            R9 = 0xFFFF;

            /* extract number of waveforms to average/stack */
            R8 = R15 AND R9;        /* number of waveforms, gives DMD 7 - 23 */
            R8 = R8 - R3;           /* Number of waveforms - 2*/
            IF LT JUMP noave;       /* 1 - 2 = -1   If no averaging */
                                    /* required jump to no averaging program */


/* delay for Clock driver PPL */
            LCNTR = 100000;
            DO delay UNTIL LCE;
            nop;
            nop;
            nop;
delay:      nop;


/******************** end of set up************************/


   /******* toggle flag 3 for first waveform *******/

tm_start:   BIT SET ASTAT FLG3;
            BIT CLR ASTAT FLG3;


 log_loop:  DM(i1,m1) = 0x10; /* Output averaged waveforms start code to pcff */
            DM(i1,m1) = 0xfc;
            DM(i1,m1) = 0x10;
            DM(i1,m1) = 0xfc;

/************** get first waveform ******************************/
```

```
            i9=b9;                          /* maximum number of samples = maximum loop size*/
            i8=b8;                          /* maximum number of samples = maximum loop size*/
            DO check1 UNTIL FLAG2_IN;  /*  Look for Flag 2 set = start of waveform.  */
                                            /*flag2 is tested as per user man pg 3:17 */
                                            /*pipeline effect on short loops. */
                                            /*Also sampling Flag in */
            R0=DM(i0,m0);                   /* input a to d data 2 lots in 1 word*/
check1:     R3=R0 AND R6;                   /* R6 = 0xFF, separated lower a to d */


            LCNTR=R13, do first1 until lce;
                                            /* R13 = number of samples / 2 -999   */
            R5 = FEXT R0 BY R7, R0=DM(i0,m0), PM(i9,m9)=R3;
                                            /* extract upper 8 bits = a to d output upper*/
first1:     R3=R0 AND R6, PM(i9,m9)=R5;


            BIT SET ASTAT FLG3;             /* toggle flag for next waveform  */
            BIT CLR ASTAT FLG3;             /* before finished getting this one */

            LCNTR=995, do second1 until lce;  /* 1 (lost) + 1  + nosamples/2 -999 + (999-4) */
            R5 = FEXT R0 BY R7, R0=DM(i0,m0), PM(i9,m9)=R3;
second1:    R3=R0 AND R6, PM(i9,m9)=R5;

            R5 = FEXT R0 BY R7;
            PM(i9,m9)=R5;

/***************** end of get first waveform ********************/
/***************** middle waveforms get and average *************/

            R0 = R8 - 1;     /* see if no averages is two */
            IF LT jump last;

            LCNTR=R8, DO outer_loop UNTIL LCE;          /* R8 = (no waveforms - 2).  */
                                                /* start of outer loop        */

            i9=b9;                              /* set i9 to start of array */
```

```
            i8=b8;                                  /* set i8 to start of array */

            DO check2 UNTIL FLAG2_IN;
            R0=DM(i0,m0);
check2:     R3=R0 AND R6;
            R1=PM(i8,m8);

            LCNTR=R13, do main until lce;
            R4=R3+R1, R1=PM(i8,m8);
            R5 = FEXT R0 BY R7, R0=DM(i0,m0), PM(i9,m9)=R4;
            R4=R5+R1, R1=PM(i8,m8);
 main:      R3=R0 AND R6, PM(i9,m9)=R4;

            BIT SET ASTAT FLG3;  /* toggle flag for next */
            BIT CLR ASTAT FLG3;

            LCNTR=995, do rest until lce;

            R4=R3+R1, R1=PM(i8,m8);
            R5 = FEXT R0 BY R7, R0=DM(i0,m0), PM(i9,m9)=R4;
            R4=R5+R1, R1=PM(i8,m8);
 rest:      R3=R0 AND R6, PM(i9,m9)=R4;

            R4=R3+R1, R1=PM(i8,m8);
            R5 = FEXT R0 BY R7, PM(i9,m9)=R4;
            R4=R5+R1;
outer_loop: PM(i9,m9)=R4;

/*** This is the end of the averaging of the middle waveforms to the others */

/* The bit below does the last waveform to be averaged, */
/* its the same as above but result is op to PCff */

last:       i9=b9;                  /* set i9 to start of array */
            i8=b8;                  /* set i8 to start of array */

            DO check3 UNTIL FLAG2_IN;  /*  flag2 is tested on  */
            R0=DM(i0,m0);                   /*  the fetch cycle i.e. 3 cycles before end*/
check3:     R3=R0 AND R6;
```

```
            R1=PM(i8,m8);



            LCNTR=R10, do last3 until lce;   /* number of samples/2 - 4*/

            R4=R3+R1, R1=PM(i8,m8);
            R12 = R4-R9;
            IF GE R4 = R9;          /*Limit output to 16 bits i.e. 0xFFFF*/
            R5 = FEXT R0 BY R7, DM(i1,m1)=R4;
            R4=R5+R1, R0=DM(i0,m0), R1=PM(i8,m8);
            R12 = R4-R9;
            IF GE R4 = R9;

last3:      R3=R0 AND R6, DM(i1,m1)=R4;

            R4=R3+R1, R1=PM(i8,m8);
            R12 = R4-R9;
            IF GE R4 = R9;
            R5 = FEXT R0 BY R7, DM(i1,m1)=R4;
            R4=R5+R1;
            R12 = R4-R9;
            IF GE R4 = R9;
            DM(i1,m1)=R4;           /* output to PC FIFO */



            DM(i1,m1) = 0x00;  /* end code */
            DM(i1,m1) = 0x00;

            IF NOT FLAG1_IN  DM(i2,m2) = R11;
          /* R11 = 3, message that the DSP ready to do next waveform but PC hasn't read */
          /* FIFO yet (since it didn't set flag1 to 0) so do something */

                /*wait for PC to set FLAG1 = ready for interrupt */
                /*continuous loop, PC can interrupt to break the loop*/
            DO waite_loop1 UNTIL FLAG1_IN;
waite_loop1: NOP;
            BIT SET ASTAT FLG0;                         /* interrupt PC */
```

```
            DO waite_loop2 UNTIL NOT FLAG1_IN;
                /*wait for PC to download to DM latch and clear  FLAG1*/
waite_loop2: NOP;               /*continuous loop, PC can interrupt to break the loop*/
            BIT CLR ASTAT FLG0;     /* When pc OP's flag1 clear flag0 as PC can read it */

                /* toggle flag for first waveform of new average */
                /* Here because pc is involved above and may delay */
                /*things therefore wont get latest waveform */
            BIT SET ASTAT FLG3;
            BIT CLR ASTAT FLG3;

            JUMP log_loop;

/******************  single waveform ****************************/

noave:    R9=995;
            R13=R13 + R9;
            BIT SET ASTAT FLG3;
            BIT CLR ASTAT FLG3;


log_loop2: DM(i1,m1) = 0x10;      /* waveforms start code to pcff */
            DM(i1,m1) = 0xfc;
            DM(i1,m1) = 0x10;
            DM(i1,m1) = 0xfc;

            i9=b9;
            i8=b8;

            DO this UNTIL FLAG2_IN;  /*  flag2 is tested as per user man pg 3:17 */
            R0=DM(i0,m0);
this:     R3=R0 AND R6;

            LCNTR=R13, do that until lce;
            R5 = FEXT R0 BY R7, R0=DM(i0,m0);
            DM(i1,m1)=R3;
that:     R3=R0 AND R6, DM(i1,m1)=R5;
```

```
            R5 = FEXT R0 BY R7;
            DM(i1,m1)=R3;
            DM(i1,m1)=R5;



            DM(i1,m1) = 0x00;
            DM(i1,m1) = 0x00;   /* end code = 0,0 */

            IF NOT FLAG1_IN  DM(i2,m2) = R11;
          /* R11 = 3, message that the DSP ready to do next waveform but PC hasnt read */
          /* FIFO yet (since it didn't set flag1 to 0) so do something */

            DO waite_loop3 UNTIL FLAG1_IN;
waite_loop3: NOP;                   /*continuous loop, PC can interrupt to break the loop*/
            BIT SET ASTAT FLG0;
            DO waite_loop4 UNTIL NOT FLAG1_IN;

waite_loop4: NOP;                   /*continuous loop, PC can interrupt to break the loop*/
            BIT CLR ASTAT FLG0;      /* When pc OP's flag1 clear flag0 as PC can read it */

            BIT SET ASTAT FLG3;
            BIT CLR ASTAT FLG3;



            JUMP log_loop2;


.ENDSEG;


.SEGMENT   /PM     rst_svc;

            jump begin;
.ENDSEG;


.SEGMENT   /PM     tmzl_svc;
```

```
            jump tm_start;
.ENDSEG;



.SEGMENT   /PM    pm_data;

.VAR pmdata[6200];

.ENDSEG;

.SEGMENT   /DM    dm_pcfifo;

.PORT doutff;

.ENDSEG;


.SEGMENT   /DM    dm_mezanine;

.PORT atod;

.ENDSEG;


.SEGMENT   /DM    dm_latch;

.PORT dio;

.ENDSEG;
```