A High Speed Data Acquisition System

Andrew Martens

A dissertation submitted to the Department of Electrical Engineering,

University of Cape Town, in fulfilment of the requirements

for the degree of Master of Science in Engineering.

Cape Town, February 2005

Declaration

I declare that this dissertation is my own, unaided work. It is being submitted for the degree of Master of Science in Engineering in the University of Cape Town. It has not been submitted before for any degree or examination in any other university.

Signature of Author .....

Cape Town
1 February 2005

Acknowledgements

Thanks go to the University of Cape Town for allowing me the opportunity to further my studies. I am also grateful to Openfuel for giving me time off from real work to finish areas of this dissertation. On a personal note, thanks are given to my family, girlfriend and friends for their support.

Contents

List of Figures

List of Tables

Nomenclature

ADC — Analogue to Digital Converter

ASCII — American Standard Code for Information Interchange

ASMD charts — Algorithmic State Machine and Data path charts. A means of representing a design that includes a data path and a controller [12]

B — byte or bytes

b — bit or bits

CD — Compact Disk

COTS — Commercial-off-the-Shelf

DC — Direct Current

DMA — Direct Memory Access. A means of transferring data without use of the system's main processor

dBc — Power relative to actual carrier power

dBFS — Power relative to full-scale or the maximum theoretical carrier power

dBm — Power relative to 1 mV

ENOB — Effective Number of Bits. See subsection B.2.1

FIFO — First-in-First-Out storage structure

FPGA — Field Programmable Gate Array

G — 10^9 (e.g. 1 Gb/s = 10^9bits per second) when referring to data rates. 2^{30} (e.g. 250 GB = 250×2^{30}bytes) when referring to data block size

GUI — Graphical User Interface

k — 2^{10}

M — 10^6 (e.g. 100 Mb/s = 100×10^6bits per second) when referring to data rates. 2^{20} (e.g. 10 MB = 10×2^{20} bytes) when referring to data block size

PC — Personal Computer

PCB — Printed Circuit Board

PCI — Peripheral Component Interconnect. A standard "that specifies a computer bus for attaching peripheral devices to a computer motherboard" [9].

PMC — PCI Mezzanine Card. "A PCB manufactured to the IEEE P1386.1 standard. This standard combines the electrical characteristics of the PCI bus with mechanical dimensions of the Common Mezzanie Card...format" [8].

PMC to PCI adapter card — An adapter card from the PMC form factor to PCI

PROM — Programmable Read Only Memory

Quadword — 8 bytes

RAID — Rendundant Array of Independent Disks

RAM — Random Access Memory

SNR — Signal to Noise Ratio. See subsection B.2.1

S — Samples or sample

SINAD — Signal to Noise and Distortion Ratio. See subsection B.2.1

SFDR — Spurious Free Dynamic Range. See subsection B.2.1

VHDL — VHSIC Hardware Description Language

VHSIC — Very High Speed Integrated Circuit

Word — 2 bytes

ZBT RAM — Zeros Bus Turnaround RAM. "Type of SRAM that can read or write every clock cycle to allow 100 percent bus efficiency" [7]

Chapter 1

Introduction

Digital systems pervade the world around us. The interface between analogue data sources and these digital systems are the realm of analogue to digital converters (ADCs) that acquire digital snap-shots of data for further processing. Some applications require high sampling rates or high resolution data (or both). In addition to this, certain applications require the capture of large amounts of data.

A good example of an application requiring a high sampling rate and high resolution data, is a digital spectrum analyser used to analyse large bands of a spectrum and offer precise results. Radar systems such as synthetic aperture radars use post-processing techniques on large quantities of data[1]. A developing field requiring versatile data capture systems is that of software defined radio (SDR). It is "a collection of hardware and software technologies that enable reconfigurable system architectures for wireless networks and user terminals"[10][2].

This document gives details on a project to build a high speed, high resolution data acquisition system that is capable of performing to some of the most stringent requirements. Specifically, this thesis documents the design, implementation and testing of firmware implemented in an FPGA in a commercial data capture card as part of the system. This firmware would facilitate the real-time transfer of captured data to RAM in a host PC.

1.1 Overview

This section gives an overview of the chapters to follow in this document. A brief description along with significant information and findings of each is given in the following subsections.

[1] See `http://rrsg.ee.uct.ac.za/sasar/` for an example of such a system

[2] An open source Software Defined Radio project is run by the GNU foundation. Details can be found at `http://www.gnu.org/software/gnuradio/`

1.1.1 System Design and Implementation

This chapter is split into various sections detailing the design and implementation of the final system.

The first section of this chapter gives various functional and structural system requirements. These outline a high performance data capture system capable of sampling two channels of data at 100 MHz continuously for up to 5 seconds. The system was to be composed of a commercially available data capture card plugged into a high performance PC. Data would be captured in the data capture card and be passed on via the PC's PCI bus for storage in RAM. Firmware and software would coordinate this transfer.

The following section gives information on system operation during data capture. This process is broken into a number of steps with details on the role different system components play in each step. Details on the format of the resultant data are also given.

Various system configuration options are provided in the following section. These include different data resolutions, trigger sources, clock sources and tag values to be used when marking various events during data capture.

The following section gives detailed information on hardware chosen for the system. Details are provided on the host PC, ADC card and PMC to PCI adapter card used.

Following this is a section giving basic information on the PCI device driver and software application suite implemented to configure the system and coordinate data capture.

The design of the custom firmware necessary to capture, preprocess and forward the data to the rest of the system is then described. Challenges and the design strategies implemented to overcome them are shown in Table 1.1. This section includes details of the various versions of firmware produced in the prototyping design loop used to produce the final version.

1.1.2 Testing and Results

The system was tested to check its conformance to requirements and this chapter details the results of these tests. These results are broken into two sections.

The first section details tests to verify the basic functionality of the system. This functionality and the results obtained from the tests are shown in Table 1.2.

The following section relates to the system performance in terms of data rates, captured block sizes and data quality. The results of these tests are summarized in

Table 1.1: Firmware challenges and appropriate design strategies

Challenge	Design strategy employed
Unknown performance of PCI firmware, PCI bus, operating system and software in terms of sustained data rates.	A prototype based system implementation strategy was used to design iteratively higher performance systems.
Unknown performance of PCI firmware, PCI bus, operating system and software in terms of latencies.	Buffering of data to cater for periodic stoppages was implemented. Firmware was designed to cater for and recover from occasional data loss. Buffer overflow was restricted to occur only in the custom firmware module.
Unknown capability of firmware in FPGA to perform to required clock frequency.	Firmware was designed to include heavy pipelining and reduced fan out from registers to increase the possible clock frequency.
Time constraint.	A Prototyping strategy was used to ensure a working system, even if not at the required performance level.

Table 1.3.

1.1.3 Conclusions and Recommendations

This final chapter gives conclusions reached about the project. The system is found to conform to specifications with scope for improvement in terms of functionality and performance.

Improvements envisioned for future systems are as follows;

- The use of FPGAs containing a larger amount of RAM or a more appropriate ZBT RAM module to reduce the likelihood of data overflows in the custom firmware module.
- The system should be tested with better quality signal generators and Nyquist filters.

1.1.4 High Speed Firmware Design Techniques

During design and implementation, various techniques had to be employed to enable the firmware to perform to the clock frequency required. Factors affecting the speed at which firmware can operate are discussed. This appendix provides information on methods to combat these sources of delay.

Table 1.2: System functionality test results

Functionality	Test Results
Data format.	Data format was found to be predictable. This included format in respect of the following; • The position and behaviour of synchronisation flags and the way in which data behaved in their presence. • The structure of data with the system operating in different data resolution configurations • The placement and structure of overflow markers and the way in which data behaved in their presence.
Transfer start and end.	Very large transfers (250 GB) of data were captured without disruption. Multiple (800) transfers were performed successfully with a software generated trigger.
Trigger sources.	Transfers using the various trigger sources were initiated successfully.
Overflow detection.	Overflows were forced and the system reported them predictably.
Successful capture of large data blocks.	Large blocks (2.5 GB) of test data were captured and tested for overflows in the custom firmware module. The results of the capture of 200 blocks at different data resolutions were as follows: • At $400 \times 10^6 B/s$ (14 bits per sample plus two padding bits) 8 overflows occurred. The system lost a maximum of 2072 bytes in an overflow. • At $350 \times 10^6 B/s$ (14 bits per sample) 3 overflows occurred. The system lost a maximum of 280 bytes in an overflow. • At $300 \times 10^6 B/s$ (12 bits per sample) 1 overflow occurred and the system lost 56 bytes. • At $200 \times 10^6 B/s$ (8 bits per sample) no overflows occurred.

Table 1.3: System performance test results

Performance Parameter	Test Results
Data rates.	The system was tested at data rates up to $400 \times 10^6 B/s$. The probability of data loss decreased significantly with reduced data rates and, at $200 \times 10^6 B/s$, was almost zero. In all cases data capture was to RAM on the host computer and under as close to ideal conditions as could be found.
Data block sizes.	2.5 GB data blocks were captured to the host computer's RAM, the maximum data block size possible with the RAM available.
ADC performance	Signal generators were used to generate data that was captured by the system. Due to the poor quality of this data and lack of necessary filters, these tests were not conclusive by themselves.

1.1.5 Analogue to Digital Converter Performance Testing

An important characteristic of an ADC is the quality of data captured. This involves using standard tests to quantify the quality of the data captured. Spectral analysis and related topics are explored in this appendix as these are crucial to how the tests are performed and how results are interpreted. The tests themselves are then described which include the single, multi-tone and grounded-input tests.

1.1.6 Buffer and Data Path Design

This appendix gives detailed information on the data path designed in the custom firmware. It firstly gives the functionality the data path had to implement. The various resources available to accomplish these requirements are then provided. A final design is then described that attempts to cater for all requirements with the resources supplied.

1.2 Accompanying CD

A CD is provided with this thesis. It contains the following directories as well as a version of this thesis in electronic format.

'firmware' — Files containing all modules implemented in the final system are present in this directory. Modules related to the ZBT RAM, although not used in the final system, are included for reference. These modules are implemented in VHDL and the final system synthesized and routed using Quartus II version 3.0 with service pack 1 installed from Altera.

'software' — This directory contains the source for small C programs used in debugging and testing the system. A program to convert binary data in 14 bit (with padding bits) data resolution to ASCII data usable by Matlab is also included.

'Matlab scripts' — Matlab was used in testing the data quality capabilities of the system. This directory contains the scripts used.

1.3 Conclusion

This chapter has briefly introduced the system under design and given an overview of the chapters to follow. The next chapter starts with detailed system requirements and then details the design and implementation of the system.

Chapter 2

System Design and Implementation

This chapter begins with a set of requirements the final system was to adhere to including a high level description of the required system structure. This is followed by a description of system operation during data capture for the system implemented. Included in this discussion is a description of the format of resultant data. Following are sections giving details on hardware, software and firmware system components. Various possible system configuration options are given in the final section.

2.1 System Requirements

This section gives the set of requirements for the system. These are taken from [14]. It also gives a description of the hardware to be used and the basic system structure.

2.1.1 Functional Requirements

Time to completion The system was to be finished as soon as possible

Data resolution The data resolution did not need to be very high for the target application. Better resolution data would produce better results however.

Sample rate The minimum sampling rate would be 100×10^6 samples per second per channel for two channels of data.

Time stamping Time stamping would be required to check for lost data. Data would be produced in blocks with each block having a unique identifier. A 16 bit counter placed approximately every 1000 samples would be sufficient.

Triggering The triggering instant did not have to be exact. Data would be captured and post-processing used for synchronisation. A trigger generated by

software would be sufficient. A trigger relying on the voltage level read by the ADCs could also be useful.

Length of capture This should be programmable. In this application, a block representing about five seconds of continual capture was required.

Host computer This would be determined by data capture and storage requirements.

Software compatibility The main software requirement was that resultant data should be available in a format for processing in a C program or Matlab. A means of configuring the system via software would be necessary.

Mechanical The required system did not have any mechanical constraints.

2.1.2 Structural Requirements

The system was to be designed using an ADC card that had been used in a previous project. This was the PM480 produced by Parsec (see `http://www.parsec.co.za` for the latest information on this card). Figure 2.1 shows a block diagram of the required system. The PM480 would plug into a PMC to PCI adapter card and this would plug into a PCI slot in a host computer running Linux. The PM480 contained firmware supplied by the manufacturer to perform PCI transactions. This firmware allowed data to be transferred between a PCI driver on the host computer, and custom firmware on the PM480. This custom firmware would be implemented to capture data and send it on for transfer to RAM of the host computer. The custom firmware was the responsibility of the author and is the main focus of this document although many aspects of the whole system are discussed. A colleague was to implement the PCI driver as well as other software to ease the task of system configuration and data capture.

2.2 System Operation

This section gives details on system operation. Steps taken during data capture as well as the roles played by various system components are provided in the first subsection. The second subsection provides information on the format of the resultant data.

2.2.1 Data Capture

The sequence of operations taken during the capture of data would be as follows:

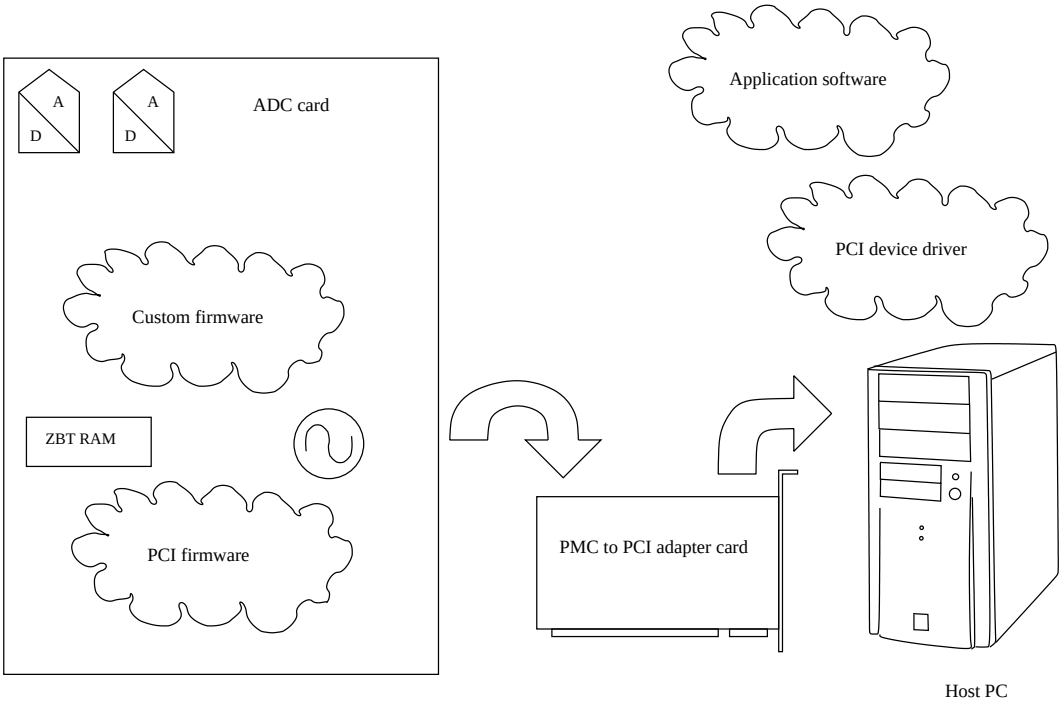


Figure 2.1: Block diagram of system

1. A user space application would be started that would prompt the user for various configuration options. This application would then configure the system for optimal performance and load the PCI device driver using appropriate arguments. The driver would use the configuration settings passed to it to configure parts of the system under its control.

2. The user space application would then wait for data transfer to complete via the device driver. Data transfer would start when the custom firmware received a trigger to start capture. The trigger source would be configurable. (See section 2.3 for information on these options). If a time to start capture was provided, the trigger would be generated by the application software at the correct time.

3. When data transfer begins, the custom firmware would take a reading from each data source every clock cycle and pass these samples on for buffering. The data would then be read out of the buffer and passed on to a buffer in the PCI firmware module for transfer on via the PCI subsystem.

4. The control unit controlling reads and writes to the custom firmware's buffer would monitor the level of this buffer and control reading from the custom firmware's buffer to prevent overflows. If the custom firmware's buffer was about to overflow, further writes would be prevented and a marker inserted into the data to indicate the location and size of the overflow.

5. The amount of data being transferred would be monitored and a marker

inserted into the data at regular intervals before passing it on to the PCI firmware.

6. The firmware transferring data to the host computer via the PCI subsystem would transfer data directly into RAM.

7. Once the required amount of data had been captured, the PCI device driver would oversee the suspension of data capture. The user space application would then transfer the data from RAM to a file on hard disk, unload the device driver, and return the system to its normal configuration.

2.2.2 Data Format

Data would be broken into 'blocks' of uniform size containing a specified number of 64 bit quadwords. A marker would be placed at the end of the specified number of quadwords. This marker would be a quadword itself and contained a 32 bit tag, that would help to identify the quadword as a marker, and a 32 bit counter identifying the block's position in the data stream. This marker is referred to as a 'block marker', 'end of block marker' or 'synchronisation word' in the text that follows.

Data from each channel would be interleaved so that the first sample from channel 'A' would be followed by the first sample from channel 'B' which would then, in turn, be followed by the second sample from channel 'A' and so on. Data would not be byte-aligned (except where this would be implicit i.e. in the 8 and 14 bit (with two bits of padding) data resolution configurations). Each sample would follow directly after the previous sample. In 14 bit (with padding) data resolution mode, two extra '0' bits would be added 'above' the most significant bit giving 16 bits per data sample. This would help to distinguish data from markers (if the appropriate bits were '1' in the marker). This would also make data extraction easier by forcing data to align to word boundaries .

In the event of an imminent overflow in the buffer in the custom firmware module, data would be discarded and a marker placed in the data stream to mark the place and extent of data loss. This marker would be placed so that there would be no partial sample before or after it (i.e. it would separate complete data samples). It would consist of the same 32 bit tag used to mark the end of a block of data, followed by a 32 bit counter indicating how many bytes had been lost. The data block containing the marker would not be truncated but still contained the correct number of quadwords (one of which would be the overflow marker).

Figure 2.2 shows a data stream of example 12 bit resolution data. The first data sample in the stream is on the left. Details of the first end of block marker and one overflow marker are shown. The beginning and end of the data are also shown.

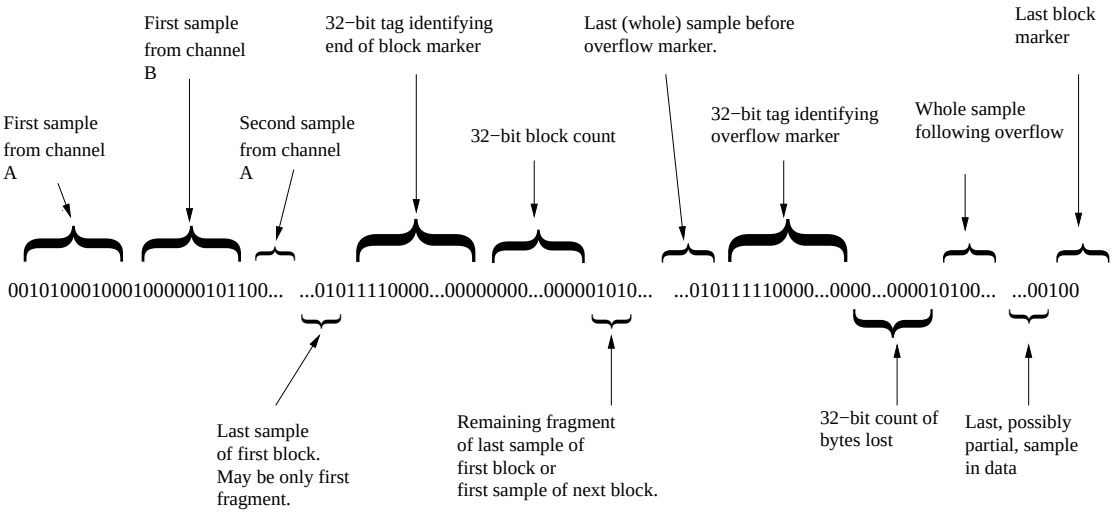


Figure 2.2: Data format of captured data

2.3 Configuration Options

Configuration options for the system were to be as follows:

- The number of quadwords in a data block could be specified.
- The insertion of a tag at the end of every data block could be suppressed.
- The unique 32 bit tag that would be used to mark the end of data blocks and data loss could be set to any value.
- The trigger source could be configured to be from software, an external source or when the data reached a specific level on a data channel.
- The data source could be the ADCs or a firmware module generating test 'ramp' data for debugging.
- The clock source for the ADCs and custom firmware could be an on-board oscillator or an external input.
- The data resolution was to be configurable as 8, 12, 14 bits or 14 bits plus two padding bits.

2.4 Hardware

The hardware used in the final system can be divided into three main parts:

- A high speed sampling card containing ADCs to sample the data stream and FPGAs to contain the custom and PCI firmware.

- A PMC to PCI adapter card.
- A computer with a large quantity of RAM, hard drive, processor and good quality motherboard supporting high data rate PCI busses.

2.4.1 The ADC Card

The PM480 wdata acquisition card from Parsec was chosen (see `http://www.parsec.co.za` for more information on the PM480). This included a PCI firmware subsystem that would allow configuration and data transfer. The version of the PM480 usfsed for this project also contained:

- Two ACEX FPGAs
- Two Analog Devices AD6645 ADCs
- A 4 MB ZBT RAM module
- A 100 MHz oscillator and a clock multiplexer allowing the choice between the onboard oscillator and an external input as the clock source
- An external input for supplying a trigger to the custom firmware

2.4.2 The PMC to PCI Adapter Card

An adapter card was needed as the ADC card we were using could not plug directly into a PCI slot as it was in the PMC form-factor. A passive adapter card (the PCI bus lines are merely extended) from another project was used in the first part of development but this proved not up to the task. The system would work only on certain PCs. In others, the entire operating system would lock up during transfer forcing a hard reboot. Certain data blocks would also be 'lost'. Many days were spent debugging and checking firmware and software. The replacement of the passive adapter card with an active one (which implements a PCI bridge on the card) solved all of these problems. The final system used a PMB-P Peritek active PMC to PCI adapter card. See `http://www.peritek.com/pmb_p_main.htm` for more information on this card.

2.4.3 The Host Computer

The host computer was chosen for its ability to handle the data rates to be generated. Hard drives did not support the data rates required (performance testing on a high-end IBM system using a RAID system yielded a sustained data rate of just over

170×2^{20} B/s when writing a large file) so capture would be to RAM. A good quality PCI chip set and the ability to support a large amount (at least 3 GB) of system RAM was required. An above-average processor was thought to be sufficient. The RAM had to be fast enough to support the data rate and the error-correcting capabilities of the modules chosen was a bonus. The final system consisted of the following:

- An Intel SE7505VB2 (Vero Beach) motherboard supporting one 64-bit, 66 MHz PCI bus.
- A single Xeon 2.4 GHz processor (512 kB cache).
- 3 GB of Transcend ECC266 DDR RAM. 500 MB of this would be allocated for use by software and 2.5 GB made into a RAM disk for data capture.
- A 120 GB 7200 rpm Seagate Barracuda hard drive.

2.5 Software

The custom built software was developed in Linux by a colleague and consisted of two main modules:

- A device driver to coordinate the capture of data and configure the system.
- A software suite to get configuration information from the user, load the driver, configure the operating system, unload the driver after capture, and copy data to a file on the hard drive after capture.

2.5.1 The PCI Device Driver

The PCI device driver was developed in parallel with the custom firmware. It was designed to be highly configurable so that various settings could be tweaked to improve performance. Two versions were developed starting with a basic, low performance version and moving to a high performance version. This software was extremely useful for testing and debugging. The final version supported the following functions:

- Configuration of the firmware and hardware on loading the module. This allowed the user to specify options like clock source, data resolution, trigger source and data block size.

- Configuration of software resources such as the size and number of internal buffers to allow tweaking.
- Access to status information obtained from various parts of the system at various stages of data capture. This data was used during debugging and tweaking.

2.5.2 The Data Capture Suite

The utility developed to aid a user during data capture served various functions:

- It allowed the user to specify a time at which capture should start if it was not to start immediately
- It allowed the user to choose from various system configuration options in a simple manner
- It automatically configured the system for optimum performance
- It loaded the device driver with appropriate configuration options removing the task of learning the various options from the user

It was implemented in two parts:

1. An executable that interfaced with the device driver to start and end capture. This executable could take arguments such as what size data block to capture and at what time to start data capture.
2. A set of scripts that helped automate the process of configuration and data capture. The set of scripts provided a GUI to obtain the settings from the user. This set of scripts then loaded and configured the device driver for optimum operating conditions. It also configured the other parts of the system for ideal system conditions during capture by performing tasks such as shutting down unneeded processes. It then called the executable with appropriate options for data capture. After capture, it copied the data block to the hard drive for further processing and returned the system to a normal configuration.

2.6 Firmware

This section details the design and implementation of the custom firmware responsible for capturing data from the ADCs and sending it on to the PCI firmware.

The various challenges associated with this firmware are introduced first along with the design strategy that would attempt to overcome each of these challenges. The actual design on the various firmware modules is then described. The prototyping process used to develop the process is described, with the various prototypes and information used in their design being described.

2.6.1 Challenges

This section gives the various challenges to be overcome when designing the custom firmware as well as the design strategies implemented in an attempt to overcome them. Various smaller challenges were encountered such as designing for metastability, debugging various error conditions, interaction between multiple state machines and others but these are not project specific and are well discussed in other sources.

Average data rates

The system to be constructed was reasonably large consisting of many components that were untested at the data rates concerned and whose operation was not under the designer's control. In addition, the entire system was not available at the time of design for testing, portions of it arriving during implementation with low performance substitutes being used in the interim as implementation could not be delayed due to time constraints. This unknown average data was the largest potential challenge. Literature could not be found on reliable transmission of large blocks of data via PCI at 400×10^6 B/s (which would be the approximate data rate required) under any conditions. A non-trivial bottleneck at any point would be difficult to cater for, except to attempt replacement with a new system component which would add time and cost to the project. System components that might not perform as needed in this regard were the following:

- The PCI system might not perform to the maximum theoretical data rate due to non-performance of any of the components in the data path. This could be a function of the PCI controllers concerned (it was found through testing that some older PC's supported a lower than theoretical maximum data rate) and the PCI subsystem provided with the ADC card.
- The operating system and application software might not be efficient enough to process the data at the required rate. The main contributor to this would be the performance of the system these would run on. This would be determined by the performance of the processor, cache, RAM and the busses interlinking these components. Another contributor would be the efficiency of software routines including interrupt service and PCI transaction routines. Combined

with these would be competition for resources from devices and processes that would be part of the system. This competition could be reduced but not totally controlled due to the necessary overhead of the operating system.

The design strategy employed to overcome this potential challenge was to develop initial low performance firmware that would not require high performance from the rest of the system. Later prototypes could add to the load placed on the rest of the system as components of the final system became available and previous versions where found to perform adequately.

Latencies

If the average data rate was sufficient, the next possible challenge would be if components in the data transfer chain added large latencies during transfer. This would cause the potential loss of data if buffers in the chain overflowed. The components whose operation was unknown could again prove to be the undoing of the system:

- The operating system and application software could have slow turn-around times between processing blocks of data. The non real time performance of the Linux kernel would mean unknown delays in servicing interrupts and time spent running application processes.
- PCI communication is non real time. A PCI controller may grant the bus to a requesting device after an unspecified amount of time [17, page 15] and may force the current bus master to release the bus if it is needed by another device even if in burst mode if the bus master conforms to the recommended PCI specification [17, page 376]. These factors could combine to introduce latencies of an unknown size during transfer.
- The performance of the PCI firmware engine supplied with the ADC card was unknown in terms of time taken to initiate and end transfers.

The strategy used to reduce the probability of buffer overflows was to use as large buffers as possible to absorb occasional disruptions in data flow. The ADC card supplied was not explicitly designed to transfer large blocks of data in real time however, and support for the buffering required was implicit. Another strategy was to design the system to ensure that the system could recover from buffer overflows and continue capturing data. The system was designed so that any overflows would occur in buffers in the custom firmware module exclusively. The custom firmware would ensure that an overflow was marked and that data capture could continue. Software processing the resultant data could check for overflows and act appropriately where they were found to occur.

FPGA performance

To capture the data being produced by the ADCs, the custom firmware had to perform at 100 MHz. Parts of the system interacting with the PCI firmware had to perform at 66 MHz. In a previous project using this ADC card, getting the firmware to run at similar clock rates had been found to be a large source of frustration if not planned for. Attempting to speed up a firmware design after implementation was found to be expensive in terms of time and effort. A change in one part of the system often caused a ripple effect of necessary changes in other logic. Each change exposed the firmware to possible bugs and the testing process to fix these wasted a lot of time.

The strategy used to combat this was to design the firmware with high speed in mind. Pipelining was heavily employed and fanout from registers was reduced by forcing redundant logic and registers. See appendix A for information on techniques to increase the possible clock rate in firmware.

Time

The customer needed the system as soon as possible. This meant that design, implementation and testing had to proceed as fast as possible. There would not be very much time for redesign in case of system non-performance. Design could also not be put off until all of the necessary components were available and fully tested.

This challenge was overcome by generating an initial low performance prototype and increasing the performance of new prototypes based on test results from previous prototypes and as system components became available. This would ensure that a system would be ready for the customer, even if it performed at a standard that was lower than optimal. This strategy also allowed core functionality to be implemented initially and features added later when basic requirements were met in previous prototypes.

2.6.2 Design

The custom firmware was firstly divided into various modules responsible for different actions. In later prototypes, modules would be reused with necessary modifications restricted to modules requiring upgrading. Consistent module interfaces would help in simulation as the same testbenches could be used to test modules from different prototypes. These modules are shown in Figure 2.3. This shows all modules from all prototype systems. Some prototypes did not include all modules. Details on the functionality provided by each module is given in Table 2.1. These modules can

be found on the attached CD in the 'firmware' directory in the file with the name given in Table 2.1 (testbenches for each module are found in '<file name>_tb.vhd' on the CD where '<file name>.vhd' is the name of the file containing the module in the table).

2.6.3 Implementation

The system was implemented as a series of prototypes. Each prototype used information provided by tests on previous systems to improve on areas identified as not performing adequately. It was implemented using Quartus II (version 3.0, service pack 1) software provided by Altera. This was installed on a Windows XP operating system.

Low performance prototype

The first firmware prototype developed was one that was very conservative on what it expected from the rest of the system so as to be almost guaranteed to operate correctly:

- Most of the basic functionality was implemented. The level trigger module was not needed for basic system operation and was not implemented for this prototype.
- A single data resolution of 8 bits was available leading to a maximum required data rate of just over 200×10^6 bytes per second with block markers inserted. This data rate could be achieved with a PCI bus supporting 64-bit, 33 MHz operation which was half of what the final system's hardware supported in terms of the maximum theoretical PCI data rate (see [17] for information on data rates possible with a PCI bus). At the time, the sampling card and PCI firmware in our possession only supported 33 MHz operation and the PCs available to us mostly only had 32 bit wide PCI data busses so this firmware was in line with the capabilities of the hardware available to us for testing.
- Buffering to cater for latencies during data flow was provided by a RAM block in the FPGA and a ZBT RAM module. The 'DataPathController' module implemented an algorithm to maximise the available buffer space while attempting to match the rate at which data was written to and read from these buffers. The 'BufferController' module and associated RAM block were not implemented at that time.

Testing the system containing this prototype took a long time. Bugs were found in the PCI device driver and in firmware. Certain problems were also found that

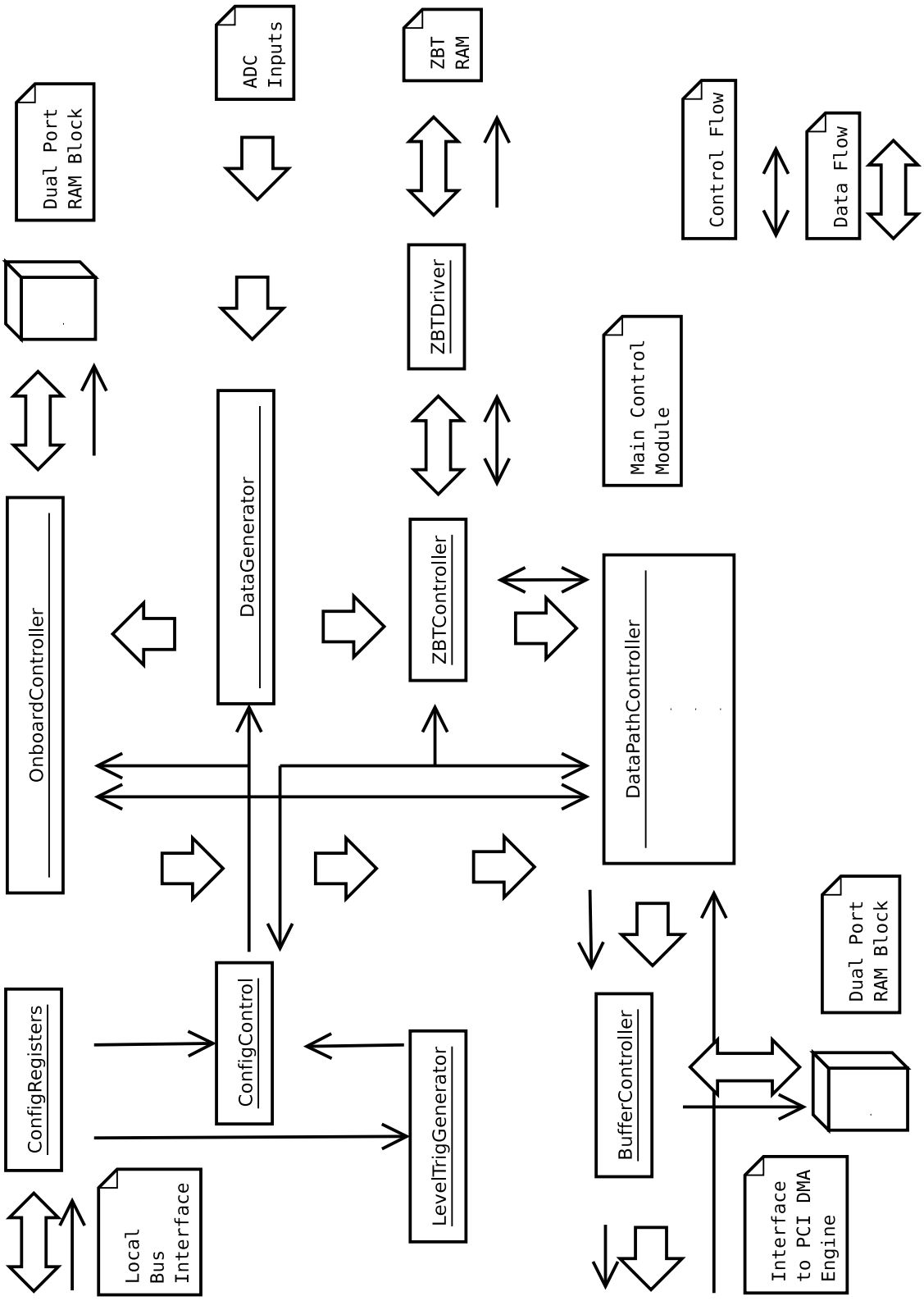


Figure 2.3: Firmware system modules

Table 2.1: Firmware module functionality

Name	Function	File Name
ConfigRegisters	Provide memory-mapped access to configuration registers and status information for the PCI device driver via the PCI firmware.	Config_ Registers.vhd
ConfigControl	Generation and control of configuration data for the system. This included clock, trigger and data sources, ensuring configuration data was updated only when appropriate (e.g. not in the middle of a transfer) and ensuring configuration signals were generated at the correct times relative to one another (e.g. triggering should occur predictably and after system configuration).	Config_ Control.vhd
DataGenerator	The data source for storage modules and LevelTrigGenerator. Should generate test ramp when appropriate and choose data format and source from configuration inputs.	Data_ Generator.vhd
LevelTrig Generator	Generate a trigger pulse when a data sample from the specified data channel had a value that was higher than the threshold value specified.	Level_Trigger_ Generator.vhd
DataPath Controller	The main control block for the data path. Controlled the start and end of data transfer. Controlled the routing of data to storage modules in the system based on configuration and feedback information. Inserted block markers into the data stream.	DataPath_ Control.vhd
Onboard Controller	Implemented a FIFO interface for a dual-port RAM module within the FPGA. Above basic FIFO functionality, it ensured reads did not overtake writes. In the first prototype it provided feedback on when to swap over to ZBT storage and, in later prototypes, prevented writes overtaking reads.	Onboard_ Controller.vhd
BufferController	This module had similar functional requirements to OnboardController except that it did not need to prevent writes overtaking reads. It was not included in the first prototype and was added to increase buffer space in later prototypes.	Buffer_ Controller.vhd
ZBTDriver	This module provided a simple interface to the ZBT RAM chip. It implemented pipelining needed to interact with the ZBT. It was only used in the first prototype.	ZBT_Driver.vhd
ZBTController	This module controlled writes and reads to the ZBT RAM chip via the ZBTDriver firmware module. It was to prevent writes overtaking reads and provide feedback on when the reader should swap over to reading from on-board storage. It was also only used in the first prototype.	ZBT_ Controller.vhd

could not be resolved and much time was spent attempting to debug them. These included:

- The system would 'lose' the last four bytes of each data block requested from the PCI subsystem. This only occurred on PCs with 32 bit wide PCI data busses.
- A more serious problem was that certain PCs would completely lock up during a transfer. Others would complete each transaction as expected. The problem was thus PC-specific.
- During testing of this prototype, metastability was encountered in the interface between the PCI and custom firmware FPGAs (see [13, 3, 18] for more on metastability). It was found that various signals would be delayed relative to others depending on how the firmware was routed. This relative delay could vary between successive routing of the same logic. This would sometimes cause incorrect values to be read. The trigger signal was found to be one of these incorrect values, and a transfer would sometimes not start and the system would stall. These errors caused a lot of confusion until debug information was inserted to evaluate the state of the firmware. Successive compiles forced registers to be sufficiently close to the input pins on the custom firmware FPGA to prevent this phenomenon.

Apart from these problems, system performance was found to be adequate. The system was tested with a PC incorporating a 64 bit PCI data bus and transfer rates were found to be more than adequate over this 33 MHz, 64 bit interface.

High performance prototype

Results of testing with the initial, low performance prototype gave the following significant results:

1. The system could support an average data rate of over 200×10^6 bytes per second. This was over a 33 MHz, 64 bit interface indicating an efficiency of over ($200 \times 10^6 bytes/s \div (33MHz \times 8bytes/cycle) =$) 75%. Assuming that the final hardware and PCI subsystem could operate at 66 MHz as specified, and, assuming that performance would scale roughly linearly with clock rate, the final system should perform as required (see [17] for information on data rates possible via PCI when in 'burst' mode).
2. Debug data was inserted into the data to indicate data flow into and out of custom firmware buffers. This data indicated that the ZBT RAM module was

unneeded at this data rate. Latencies seemed to be easily absorbed by the FPGA RAM buffer alone.

Based on these results, the next prototype aimed to improve significantly on the previous one:

- The 'LevelTrigGenerator' module was implemented and included. This brought the configuration options available up to the minimum required from the final system.
- The data resolution in this prototype was an optional 8 or 14 bits. In 14 bit resolution mode, the 14 bits were padded out to 16 as it was easy to extend the first prototype's firmware to support a 16 bit mode.
- The 'ZBTController' and 'ZBTDriver' modules were not incorporated into this prototype. The ZBT RAM module provided was a single port entity with a data bus width of 36 bits and it would be very difficult to incorporate it into the data path to support the data rate for the 14 bit mode (see appendix C). Another FIFO, consisting of the remaining on-board FPGA RAM resources, was added to the the existing buffer along with its controller ('BufferController'). The ZBT RAM module had also been found to be unnecessary in the previous prototype.

Testing of this module was helped by the arrival of an active PMC to PCI adapter card during development. A passive one had been used previously. This solved the two serious problems of the previous prototype (PC specific lock-ups and lost data). This was a great relief as testing for other bugs was hampered by these problems and much effort had been wasted in attempts to isolate their source. This also allowed the purchase of a PC for the final system as there was no PC specific problem making PC selection difficult. This PC arrived shortly before the final sampling card, whose PCI firmware could perform at 66 MHz with a 64 bit bus. These two components were also invaluable as they allowed us to test the system at high data rates.

Testing using this prototype produced the following results:

- The average data rate possible was more than sufficient. This was only under optimal or semi-optimal conditions however, where unneeded software processes where shut down and data path buffers optimised.
- Latencies were found to be a problem. The average data rate was sufficient but an occasional overflow would occur when the system was operating in 14 bit (with padding) data resolution mode.

Intermediate data rate prototypes

Testing of the second prototype showed the occasional loss of data. More prototypes were developed in an attempt to prevent this from occurring. The firmware was modified so that most of the data path components were not data resolution specific (partially completed for the high performance prototype). This allowed the development of 14 bit (non-padded) and 12 bit prototypes.

Testing of these prototypes found that reducing the data rate did reduce the likelihood of an overflow but that the firmware buffering was still not sufficient to prevent overflows in the custom firmware completely except when in 8 bit mode (see table 3.1).

Final system

Testing of the prototypes with reduced data rate requirements indicated that all functional requirements were fulfilled except that data was occasionally lost due to buffer overflows. This might be remedied by various means but each would require more time and the system was needed by the customer.

The final system used incorporated the option of choosing between 8 bit, 12 bit, 14 bit(unpadded) and 14 bit(padded) data resolutions. This would allow the user to choose which resolution best fitted their needs.

2.7 Conclusion

This chapter gave detailed system requirements and went on to describe the design and implementation of the system. The following chapter will describe tests used to verify the functionality of the system in terms of system behaviour in various configurations and under various conditions. Chapter 4 will give details of tests used to test the performance of the system.

Chapter 3

Functionality Tests and Results

This section details tests used to verify the basic functionality of the system in terms of its behaviour in various configuration modes and the ability to satisfy the required specifications.

Many of these tests used a logic block built into the firmware that outputted a ramp starting at zero and increasing linearly. A configuration option supplied to the custom firmware chose this ramp as the data source instead of the ADCs. It was then trivial to check for correct data format, alignment, corruption and loss.

3.1 Data Format Tests

These tests were used to verify that the system generated data in the expected format under all conditions and configurations. Testing involved capturing data under various system configurations and then analysing it using short C programs. The accompanying CD includes these C programs in the 'software' directory. The following aspects of data format were tested:

- A partial sample should never occur in data (except possibly the last sample in 12 or 14 bit (unpadded) data formats).
- A strict relationship between the time a trigger signal was sent and the first sample captured should be maintained.
- The alignment of data in the presence of synchronisation words and overflow markers should be predictable. This was especially important in 12 and 14 bit (unpadded) resolution modes where data was not implicitly aligned to byte boundaries when overflows occurred.

- The placement of end of block markers and the expected behaviour of their embedded counter values.

3.1.1 Method

The following tests were performed to check the data format under various configurations. Some combinations of configurations were not tested as certain configuration options would not influence results (e.g. the content of block markers was not dependent on data resolution).

- The system was configured to use different data resolutions and no synchronisation words. A 5 MB block of test data was captured for each configuration and analysed to ensure that data started at zero and increased linearly.
- The system was configured to use different data block sizes (2048, 4096, 8192 bytes) with block markers enabled. A 5 MB block of data was transferred and checked to ensure block markers were placed as expected and that the embedded counter incremented as expected.
- The system was configured to use a standard block size (8192 bytes). All permutations involving data resolution and synchronisation word were tested (i.e 8 bit, synchronisation word enabled; 8 bit, synchronisation word disabled; 12 bit, synchronisation word enabled etc). For each configuration permutation a block of 5 MB of data was captured and analysed with an appropriate C program.
- The system was configured for 14 bit (padded) resolution with synchronisation words enabled. This would result in the highest load being placed on the system in terms of the data rate required. A block of 50 MB of data was then captured with a buffer configuration that was forced to be below optimal via software. The system reported overflows and analysis with the appropriate C program verified this. Analysis showed that an appropriate overflow marker had been correctly placed wherever there was a break in the ramp data. Four overflows occurred and each checked by hand to ensure that the correct number of bytes were reported missing.
- The system was configured for 14 bit (unpadded) and 12 bit resolution. As above, a block of data was captured and overflows forced. The data was again analysed by means of an appropriate C program for overflows. The overflow markers were checked by hand to ensure that the correct number of bytes was reported lost and that no partial samples occurred before or after an overflow marker. Partial samples after an overflow marker could cause effective data

corruption in these modes as there would be no way of knowing where the next data sample started.

3.1.2 Results

The results from these tests indicated correct data format under all the usual configuration permutations. Tests to check correct data format in the presence of overflows were limited as they were done by hand and only a few examples checked. Many more overflows occurred during other tests however and the data format in all of these cases was found to be consistent.

3.2 Buffer Overflow Tests

These tests checked the performance of the system buffers under different data rate loads. This set of tests occupied a lot of time due to the time software spent processing the data to check for overflows. The test programs would typically take from 4 to 5 minutes to process a block of 2.5 GB of data, leading to about 8 hours to process 100 blocks. Tweaking was often done on one or more parameters in an attempt to further reduce the overflows and the batch tests re-run afterwards. These tests were valuable as they helped to test other areas as well (e.g. reliable software triggering, unbroken operation and format and content of overflow markers).

3.2.1 Method

Each step in the batch test involved capturing a 2.5 GB block of data (the maximum supported by the RAM in the system and 25% larger than the maximum required by the customer) and checking for data loss or corruption with small C programs. The results for each test were recorded in a log file for later reference. All of the tests were performed with the system in an optimal configuration as various buffer and system settings were found to be significant (e.g. unneeded processes should be off and buffer sizes maximised) except when testing 8 bit resolution where non-optimal configurations worked as well as optimal ones in many cases.

3.2.2 Results

The results for the different data resolution options are shown in Table 3.1. Note that these are the results of capturing 200 blocks at each resolution. When the system was capturing 12 bit resolution data, a buffer overflow only occurred once on

Table 3.1: Buffer overflow test results

Resolution	Number overflows	Max no. overflows per block	Max overflow size (bytes)
14 (padded)	8	1	2072
14	3	1	280
12	1	1	56
8	0	0	0

the 170th block so this result may have a large variance and the other results should also be considered rough estimates due to the (relatively) small sample space.

3.2.3 Discussion

The likelihood of an overflow in the custom firmware buffer can be seen to rapidly decrease as the resolution is lowered. The relationship is not linear as the decrease in data rate (linearly related to data resolution) affects the system in two crucial ways:

- Other software processes will be less likely to need resources at the same time as the data capture software simply because it needs resources for less time with a lower data rate. This lowers the likelihood of latencies caused by another process holding onto resources needed by the data capture software.
- The second way data rate affects the likelihood of an overflow is that the data capture system can now absorb larger latencies. The buffers in the system for storing data can store more samples if the samples get smaller. This means that the custom firmware buffer takes a longer time to get full and overflow if the resolution is lowered. The chance of an overflow thus decreases, assuming that the distribution governing the magnitude of latencies generated by the system is reasonably smooth without large local maxima.

3.3 Unbroken Operation Tests

An important part of testing was to ensure that a block of data could be captured successfully without disruption. The system should not enter a state where capture of a block could not complete. Even in the event of overflow in the custom firmware buffer, data capture should continue and an effort made to recover from the event gracefully.

3.3.1 Method

A large (250 GB) block of data at the highest data rate possible (just over $400 \times 10^6 B/s$) was captured. The result was discarded (redirected to /dev/null) and not checked for data loss (although a flag would be raised indicating the occurrence and the event reported by the device driver). The size of the block captured was roughly one hundred times or two orders of magnitude larger than the largest data block to be captured in normal use. It was then repeated once for lower data rates possible with the system ($350 \times 10^6 B/s$, $300 \times 10^6 B/s$, $200 \times 10^6 B/s$).

3.3.2 Results

All tests completed successfully and indicated that, once started, a block will be successfully transferred without disruption from hardware, software or firmware.

This aspect of system operation was also verified by previous tests in subsection 3.2. In these tests a total of 800 blocks were captured at different data rates. These all started and completed successfully.

3.4 Triggering Tests

These tests were performed to test the operation of the various trigger configurations available, focussing on the software trigger as this would be the one mostly used for this application. During early development, metastability could cause the occasional trigger signal to be 'lost'. After sending the trigger the software driver would then wait for data. As the firmware had not received it, the system would lock up.

3.4.1 Software Trigger

To test the software trigger functionality, a script was used to capture a small block (10 MB) of data 1000 times. All of these blocks were captured successfully. Another verification was provided by the testing performed in subsection 3.2. Here 800 blocks of data under various configurations were captured using the software trigger.

3.4.2 Level Trigger

To test the level trigger functionality implemented, the system was placed in a mode where real ADC data was exchanged for a test ramp that was generated in firmware. This ramp started at 0x000 and increased every clock cycle, wrapping around when

the maximum possible value (0x3fff for 14 bits) was reached. The system was then configured to generate a trigger when an input level of 0x1000 was exceeded on channel A. A transfer was then initiated which occurred successfully. The data generated started at 0x100d which showed the latency in clock cycles from the time a trigger was generated and the first data value captured (this value was designed top be constant regardless of trigger source). This test was repeated for channel B. This test should also have been performed using real data that might not start at 0x000 to properly test the system under normal working conditions.

3.5 Conclusion

In this chapter, various tests have been described that test the functionality of the system under various conditions and in various configurations to ensure that behaviour is consistent and correct. The following chapter gives information on tests used to quantify the performance of the system in terms of data quality, data rates and possible data quantities.

Chapter 4

Performance Tests and Results

This chapter provides details of tests used to characterize the performance of the system. The first section gives details of maximum data rates and block sizes captured with the following sections giving data quality parameters. Data quality parameters were obtained by performing the single-tone and grounded input tests as described in appendix B.

4.1 Block Sizes and Data Rates

This section overlaps with chapter 3 as the results are derived from tests detailed in that chapter.

The largest block that could be captured was restricted by the amount of RAM available. This had to be shared with the operating system and other applications and could not all be used. Up to 2.5 GB of data was captured successfully and larger blocks would have been possible if the amount of RAM were to be increased.

The highest data rate tested was just over $400 \times 10^6 B/s$ at a clock rate of 100 MHz using a 66 MHz, 64 bit PCI bus. The maximum theoretical data rate using this PCI configuration is approximately $528 \times 10^6 B/s$ [17]. However, the firmware compiler produced results showing that the custom firmware could only run reliably up to 105 MHz so testing up to the maximum theoretical data rate limit of the PCI subsystem was not possible.

4.2 Grounded Input Test

This test was used to test the maximum theoretical data resolution the system was capable of capturing. It would show the amount of noise present in the system with-

out an input signal being present. This test is described in appendices in subsection B.2.2.

4.2.1 Method

A grounded input test was performed as follows:

- The system was configured to capture data at 14 bit (padded) resolution.
- The analogue inputs to the ADCs were grounded through appropriate loads (in this case 50 Ohm) and data captured on both channels.
- The binary data captured was converted to text with the program 'pm480toascii_16' located on the attached CD in the file 'pm480toascii_16.c' in the 'software' directory.
- The data was then analysed in Matlab to determine the maximum theoretical resolution possible when using the system. The Matlab script used is in the file 'load50Ohm.m' in the 'Matlab scripts' directory on the attached CD.

4.2.2 Results

Figure 4.1 shows the power spectrum of the signal captured with the channel A ADC and Figure 4.2 the power spectrum for data captured with the channel B ADC. The power is normalised relative to the power in the maximum amplitude sinusoid the ADCs could capture. The total noise power (excluding DC) represented by each figure is -76 dB for channel A and -77 dB for channel B. This means that the maximum ENOB was approximately 12.5 bits for this system.

4.2.3 Discussion

Documentation supplied by the manufacturer for the ADC used in the system indicates a peak SNR of 75 dB [6]. The noise level found using the grounded input test thus compares favourably with that found by the manufacturer. The manufacturer's parameter was for a single tone test though and thus included noise generated by inserting a signal into the system. Also, the noise power was calculated relative to full-scale and the manufacturer's SNR was calculated relative to the signal's power which would be at around 1 dB below full-scale giving it a 1 dB disadvantage relative to the results of the grounded input test.

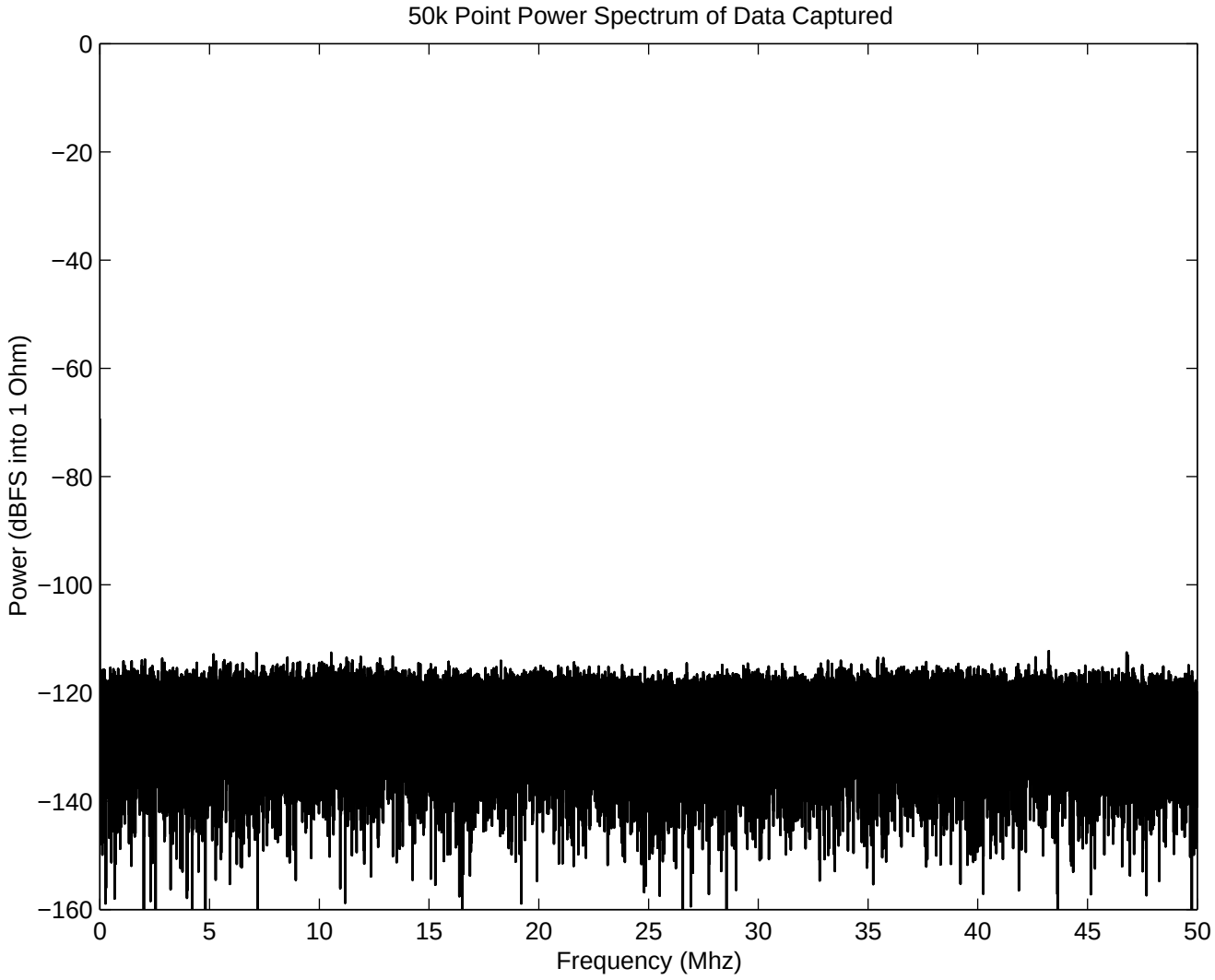


Figure 4.1: Power spectrum of data captured on channel A

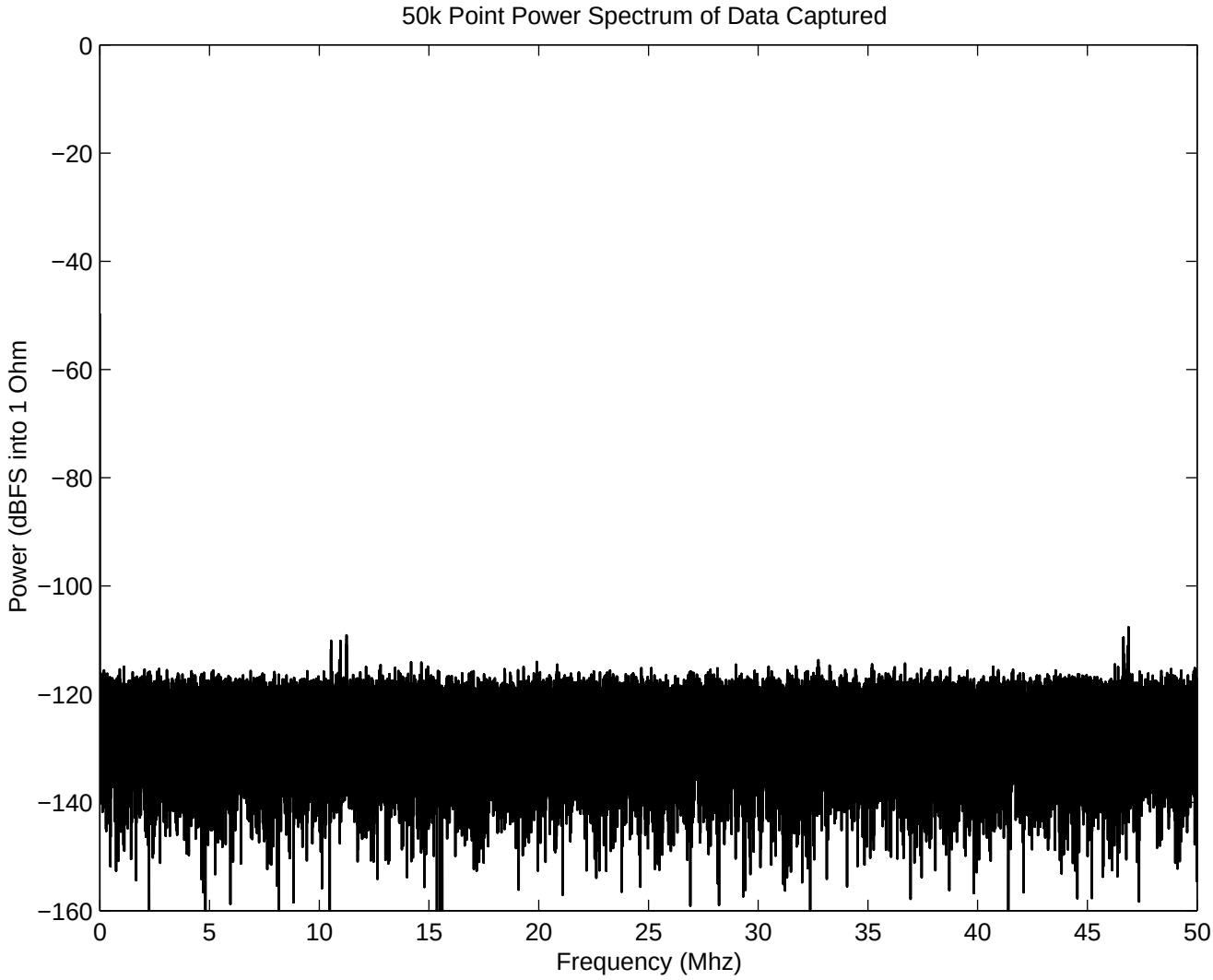


Figure 4.2: Power spectrum of data captured on channel B

4.3 Single Tone Test

This test is used by many ADC manufacturers to quantify the dynamic performance of their products. This test is described in the appendices in B.2.1.

4.3.1 Method

The single tone test was performed as follows:

The system was configured to capture maximum resolution (14 bits) data from both channels. An HP8656B signal generator was attached to the ADC of channel A and then to the ADC of channel B. Each time it was configured to output a sinusoid with an amplitude of 750 mV (this value was found by trial and error to ensure no clipping). The sinusoid frequency from the HP8656B was varied from 7.5 to 205 MHz through the bandwidth supported by the ADCs (200 MHz for IF sampling[6]).

An HP33120A was also later used when significant harmonics were seen in the data captured using the HP8656B. When using the HP33120A the frequency was swept from 2 to 15 MHz (the bandwidth supported by the signal generator).

A large block of data (20 MB) was captured for each frequency. No Nyquist filters were used to remove spurious signals from the output of the signal generators as none suitable were available during testing.

The binary data blocks were then converted to text for input to Matlab. The Matlab environment was used to calculate parameters based on this data. Figure 4.3 shows a 2 MHz signal from the HP33120A captured by the system.

4.3.2 Results

The results of this test when using the HP8656B are shown in Figure 4.4 for channel A, and Figure 4.5 for channel B. These results were obtained by using the Matlab script 'doHP8656B.m' to analyse captured data. This script is located in the 'Matlab scripts' directory on the CD included with this thesis. Results for the various parameters when using the HP33120A are shown in Figure 4.6 for channel A and Figure 4.7 for channel B. The analysis of the data from the HP33120A was performed by the Matlab script 'doHP33120A.m' found on the attached CD.

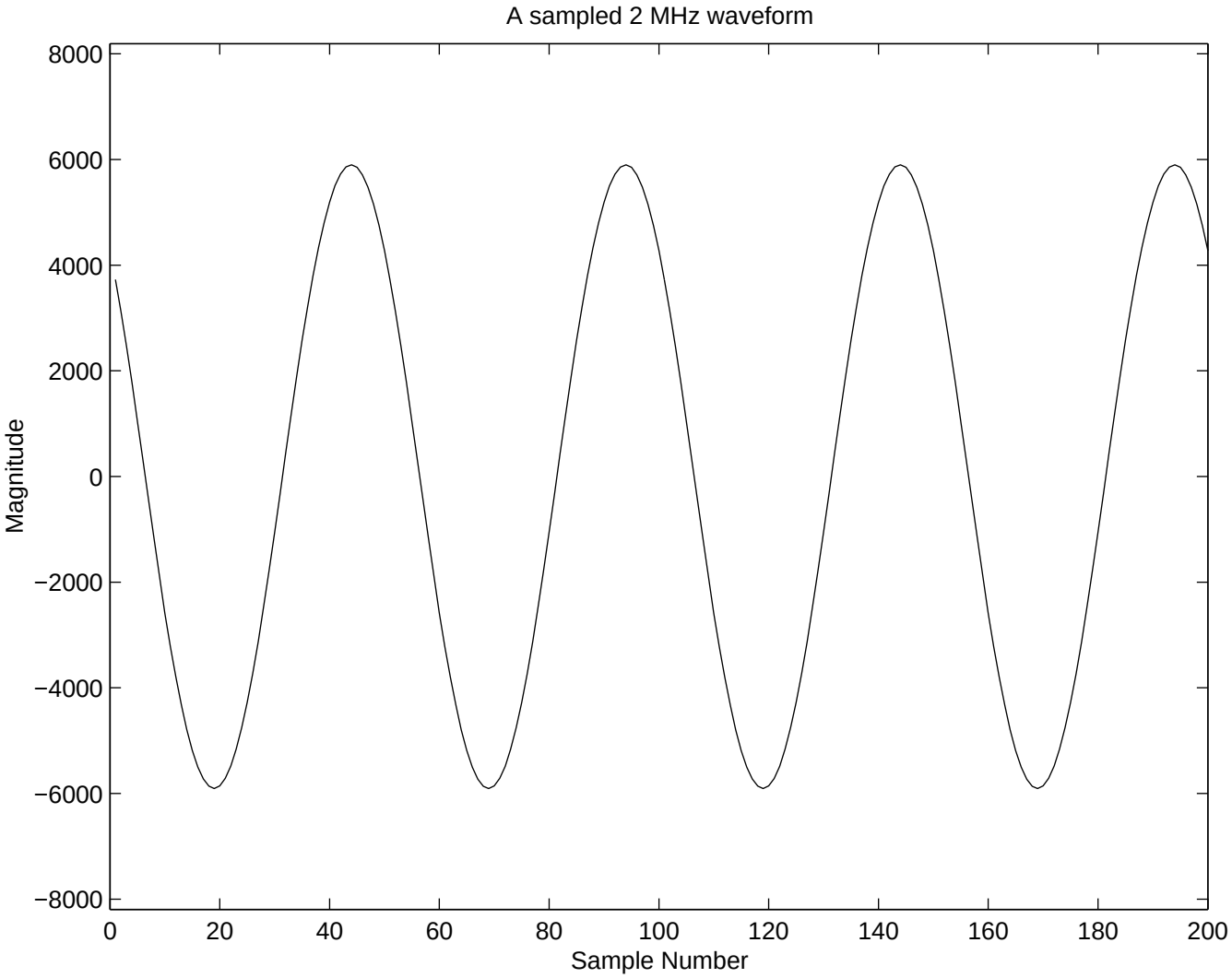


Figure 4.3: Time domain view of data captured by the system

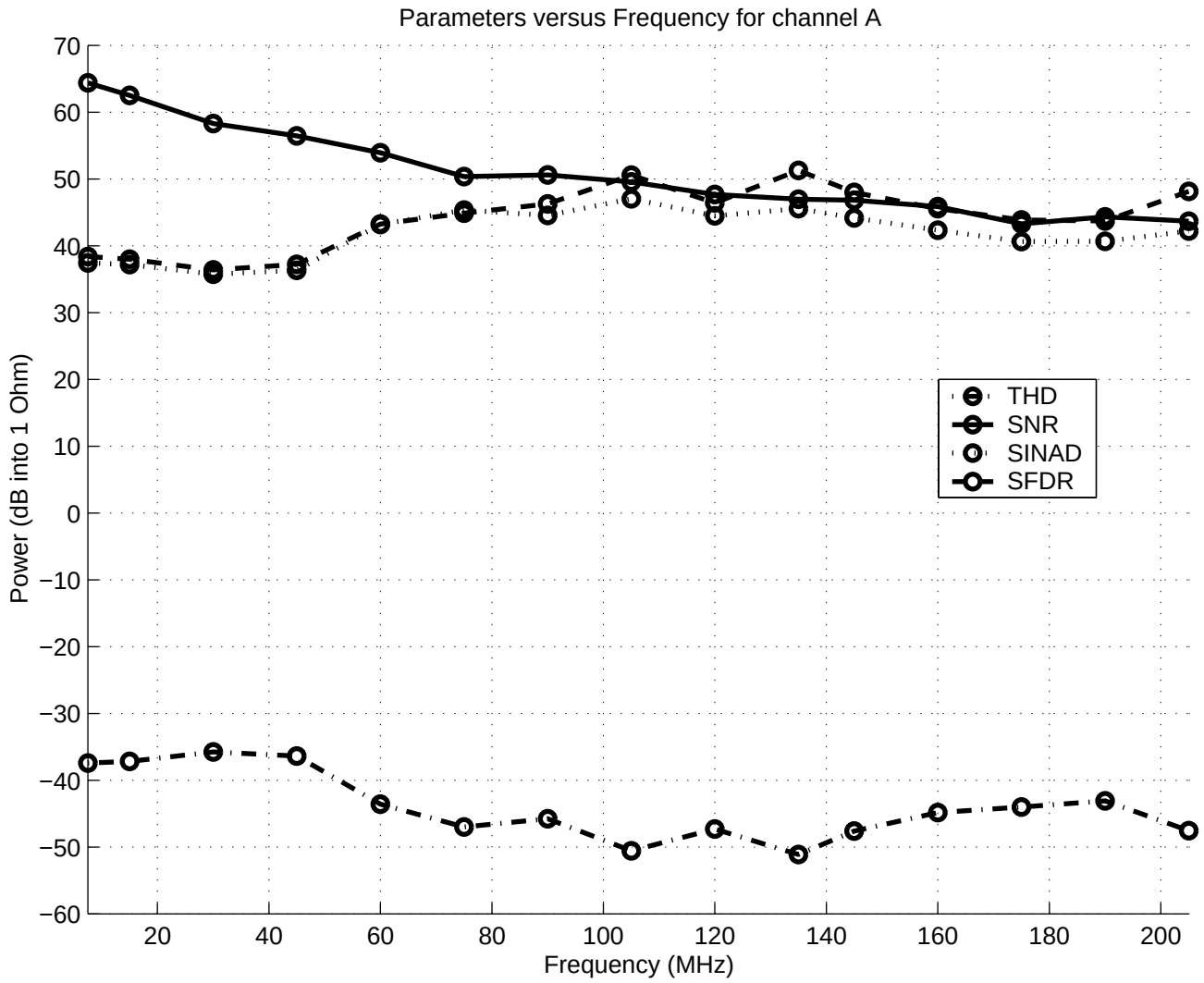


Figure 4.4: Performance parameters for channel A using the HP8656B signal generator

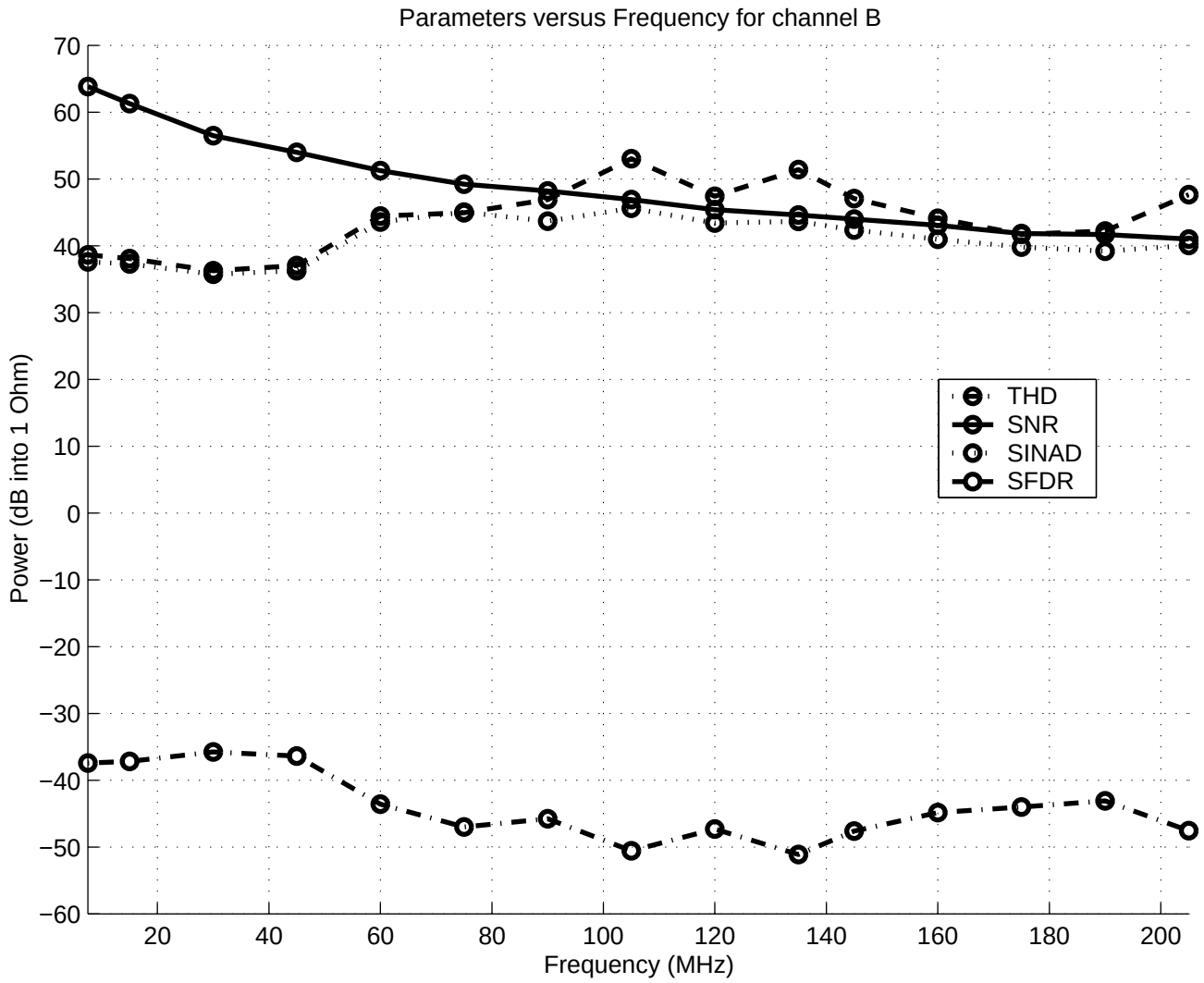


Figure 4.5: Performance parameters for channel B using the HP8656B signal generator

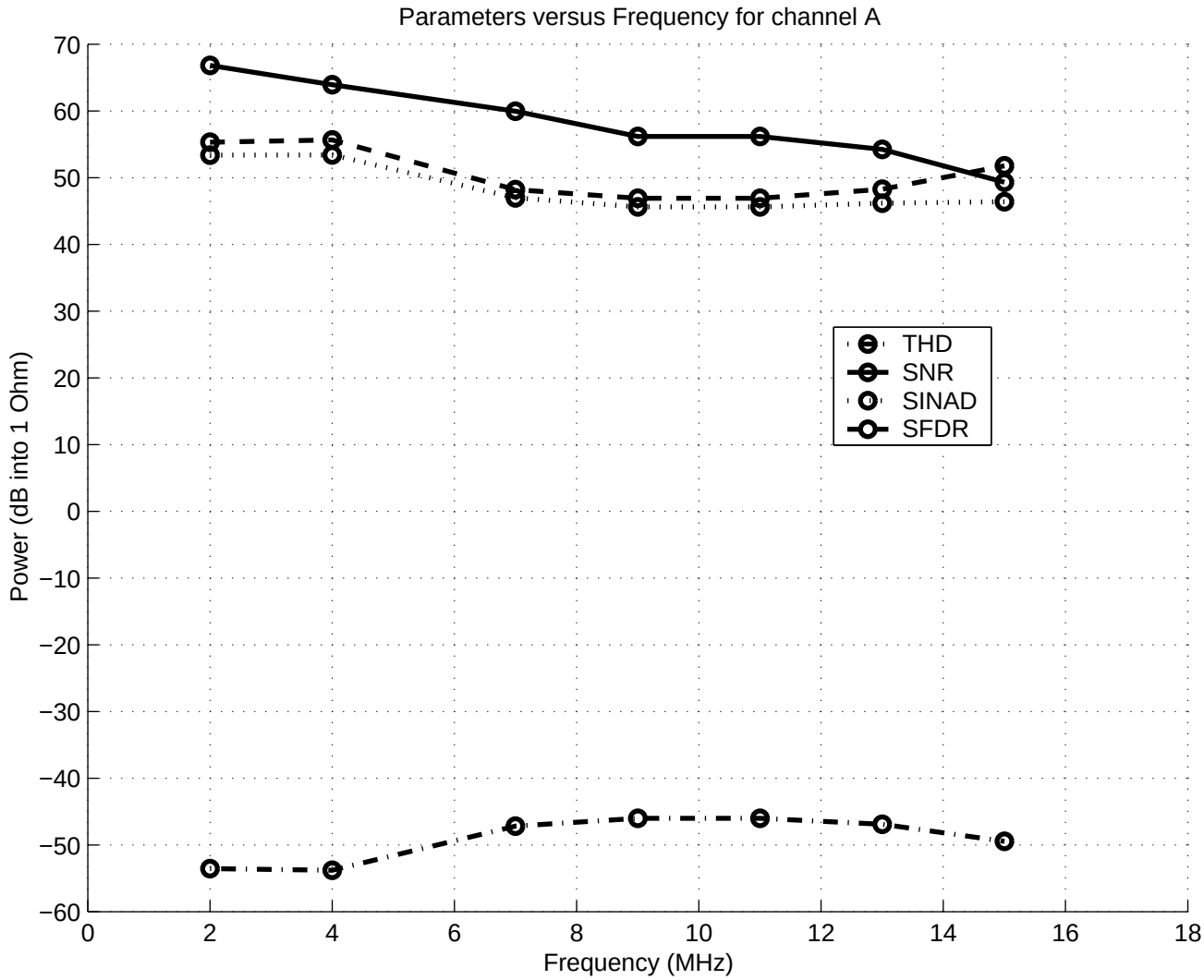


Figure 4.6: Performance parameters for channel A when using the HP33120A signal generator

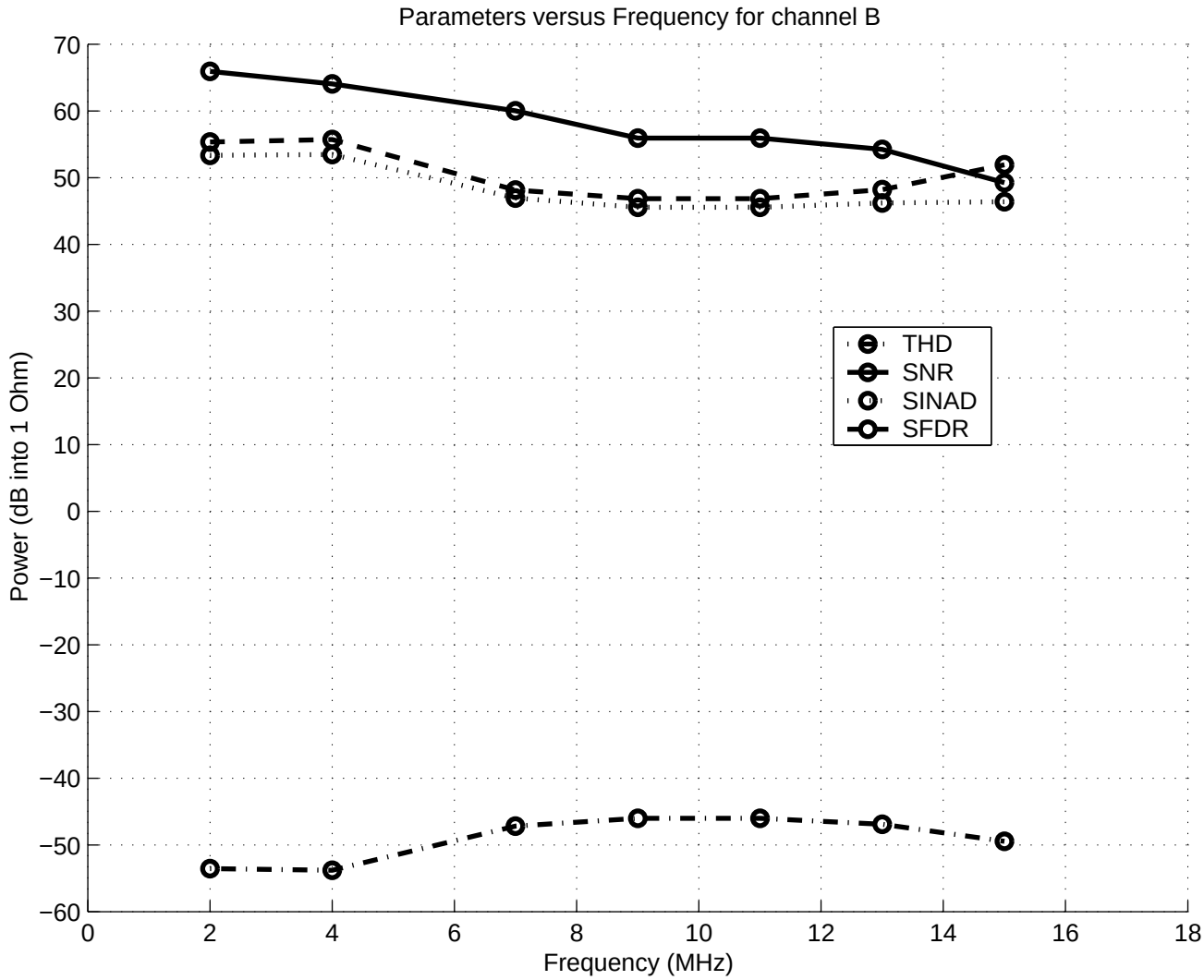


Figure 4.7: Performance parameters for channel B when using the HP33120A signal generator

4.3.3 Discussion

As can be seen from the figures, the results of these tests are far below those expected from a system producing 14 bit resolution data. Tests performed in the previous section were repeated as it was thought that data loss or corruption could be causing harmonics and noise but the same results were obtained. Test ramp data was generated using built in firmware logic and the resulting ramp tested for lost data. Visual inspection of data captured also showed no obvious missing data or corruption.

The results from the HP33120A signal generator can be seen to be much better in terms of nonlinearities than for the HP8656B where the frequencies overlap (in this case both were used to test the system at around 7 MHz). The THD, SFDR and SINAD parameters are around 10 dB better in this region for the HP33120A compared to the HP8656B. However, the SNR is about 4 dB better for the HP8656B. This indicates good linearity properties in the HP33120A compared to the HP8656B but poor on average in terms of other noise sources.

Various factors could be responsible for the worse than expected results of these tests. These are discussed in the following sub-sections.

Low performance of signal generators

The low performance of the signal generators was thought to be the largest contributor to the low performance parameters found during testing. The poor performance of the HP8656B is discussed first and thereafter that of the HP33120A.

The data sheet for the HP8656B specifies that generated harmonics should always be below -30 dBc and other noise components below -60 dBc. This seems to be consistent with data produced by the data capture system as seen in Figure 4.8 that shows a captured 205 MHz signal with harmonics marked (note how the signal appears at 5 MHz due to aliasing). Even though the noise is less than the maximum expected in the data sheet, it is far from being near the theoretical minimum noise floor for the system under test (around -86 dBFS for a 14 bit ADC [4]). Testing using this signal generator can not provide conclusive results.

Figure 4.9 shows the output of a spectrum analyser (Agilent E4407B) for a signal produced by the HP33120A and Figure 4.10 shows the same signal captured by the system. Note that the figure showing the data captured by the system has the same axis scaling and offsets as that of the spectrum analyser for easy comparison.

The magnitude and relative size of low order harmonics is similar in both figures indicating that most of the harmonic power in the signal captured by the system

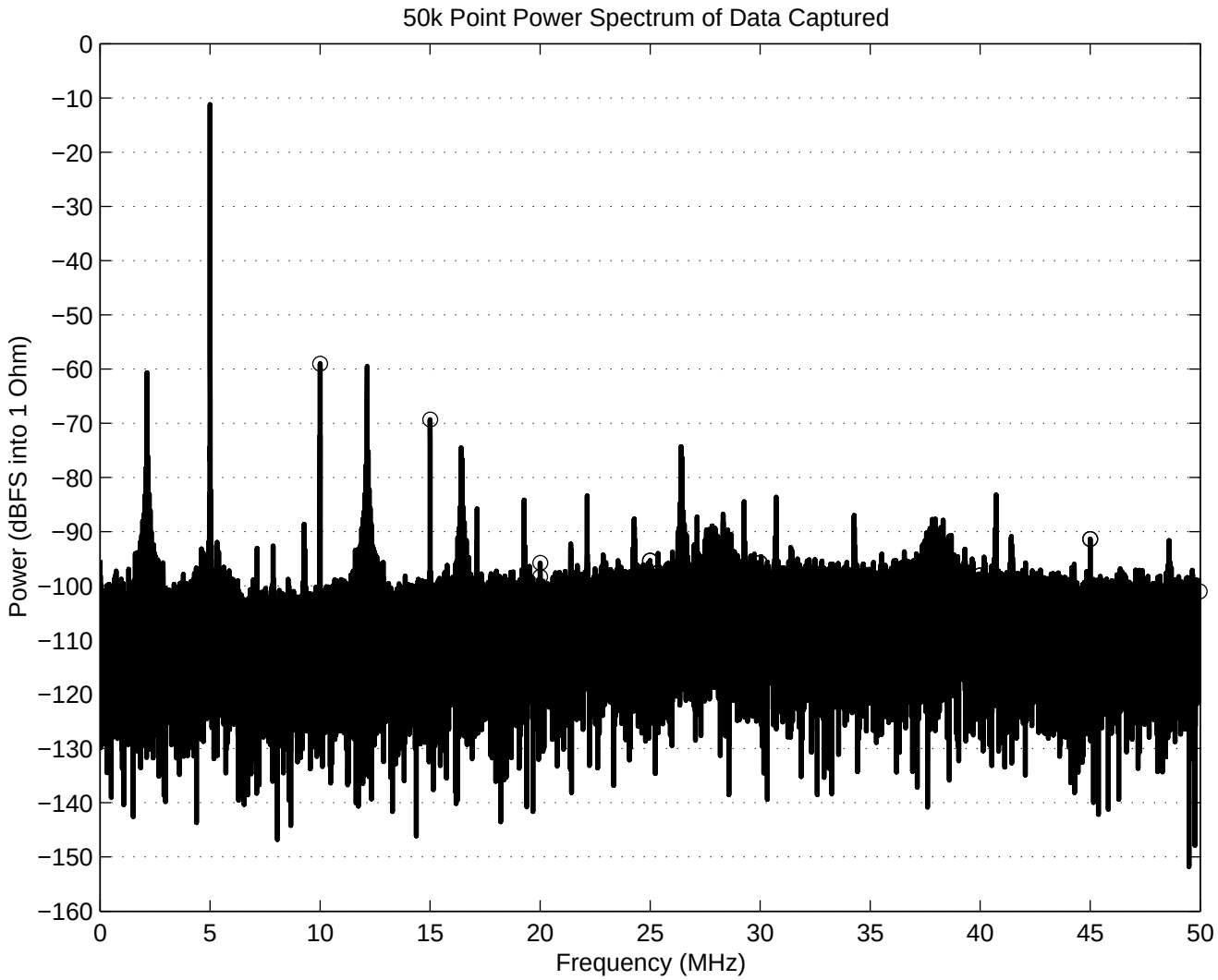


Figure 4.8: Captured spectrum of 205 MHz signal generated by HP8656B

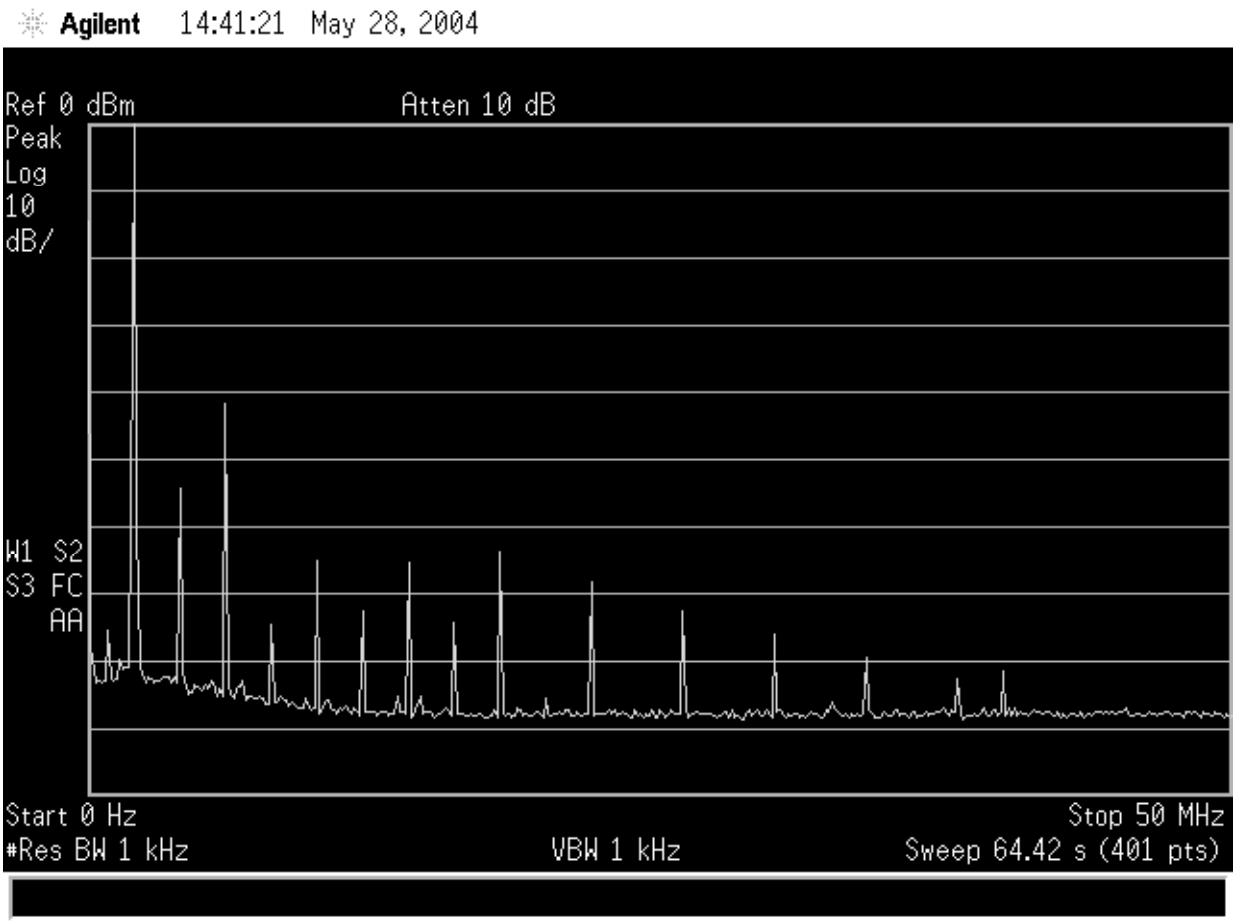


Figure 4.9: Spectrum of signal as given by Agilent E4407B

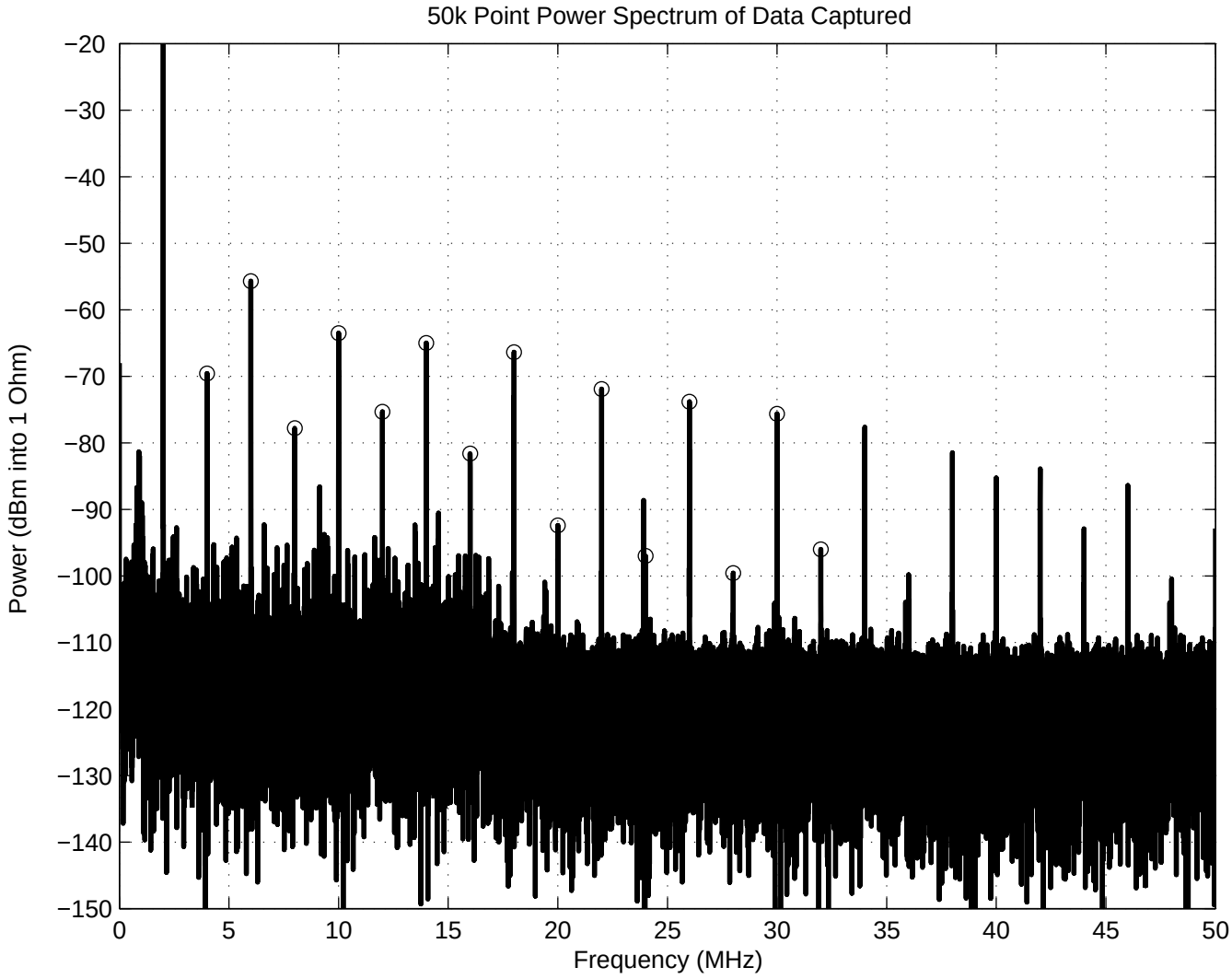


Figure 4.10: Spectrum of signal captured by system

under test is not due to non-linearities in the ADCs but due to non-linearities in the signal generator. Harmonics after the third seem to be greater in magnitude than those captured by the spectrum analyser. This is likely due to aliasing as the sampling frequency (100 MHz) in the test system was an integer multiple of the input frequency (2 MHz) so aliased harmonics above 100 MHz would add to their lower order compatriots making them seem larger which would not happen in the spectrum analyser.

The noise floor in both cases can also be seen to be similar. The spectrum analyser shows it to start at around -105 dBm at DC, decreasing and levelling off at just above -110 dBm. The spectrum of the data captured with the system under test shows a similar trend.

Another test used to check the quality of the HP33120A was to pass the output of the HP33120A through a band-pass filter before capturing it. The pass band in the filter is centred at 2 MHz. The spectrum of the captured, unfiltered signal is shown in Figure 4.11 which also shows the location of harmonics. This is shown in contrast to the spectrum of the captured, filtered signal shown in Figure 4.12. The parameters used to generate each signal were identical. As can be seen, the amplitude of harmonics is significantly reduced due to filtering although some are still evident. Many other noise sources not including harmonics have also been removed (miscellaneous noise sources from DC to 17 MHz have been reduced by at least 10 dBs).

Lack of Nyquist filters

Suitable low-pass and band-pass filters to remove spurious signals outside the Nyquist Zone under consideration were not available. Ideally a low-pass (used when sampling frequencies in the first Nyquist Zone) or band-pass (for sub-sampling in higher Nyquist Zones) filter should be used to remove noise outside the band of interest that could be aliased into the band under test. No suitable filters were available during testing and insufficient time was available to acquire any when testing started before the system was shipped to the customer.

4.4 Conclusion

This chapter has detailed the testing of the performance of the system in terms of possible data rates, data block sizes and data quality. The following chapter gives conclusions and recommendations for improvements for the system and similar future systems.

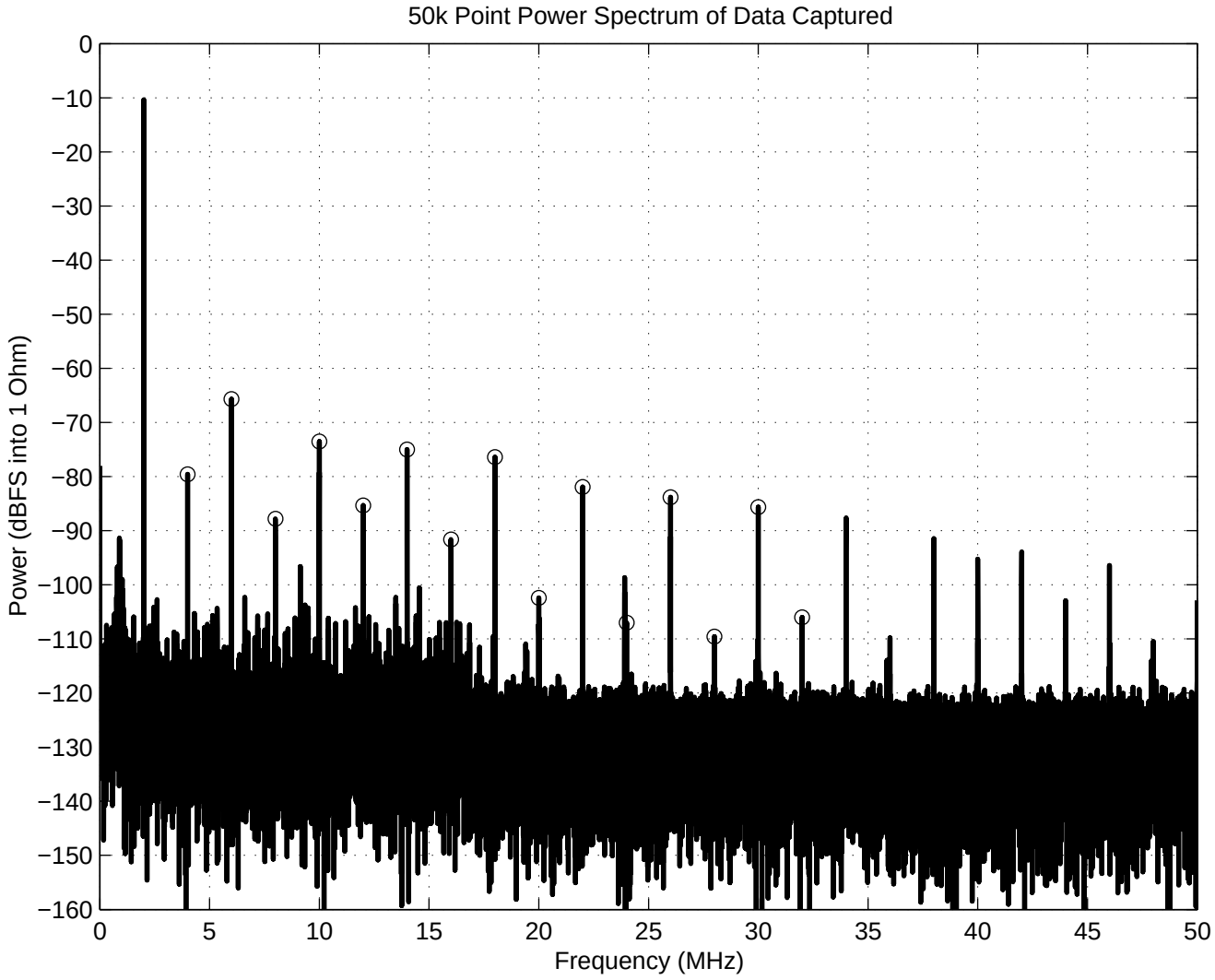


Figure 4.11: Spectrum of unfiltered 2 MHz HP33120A signal

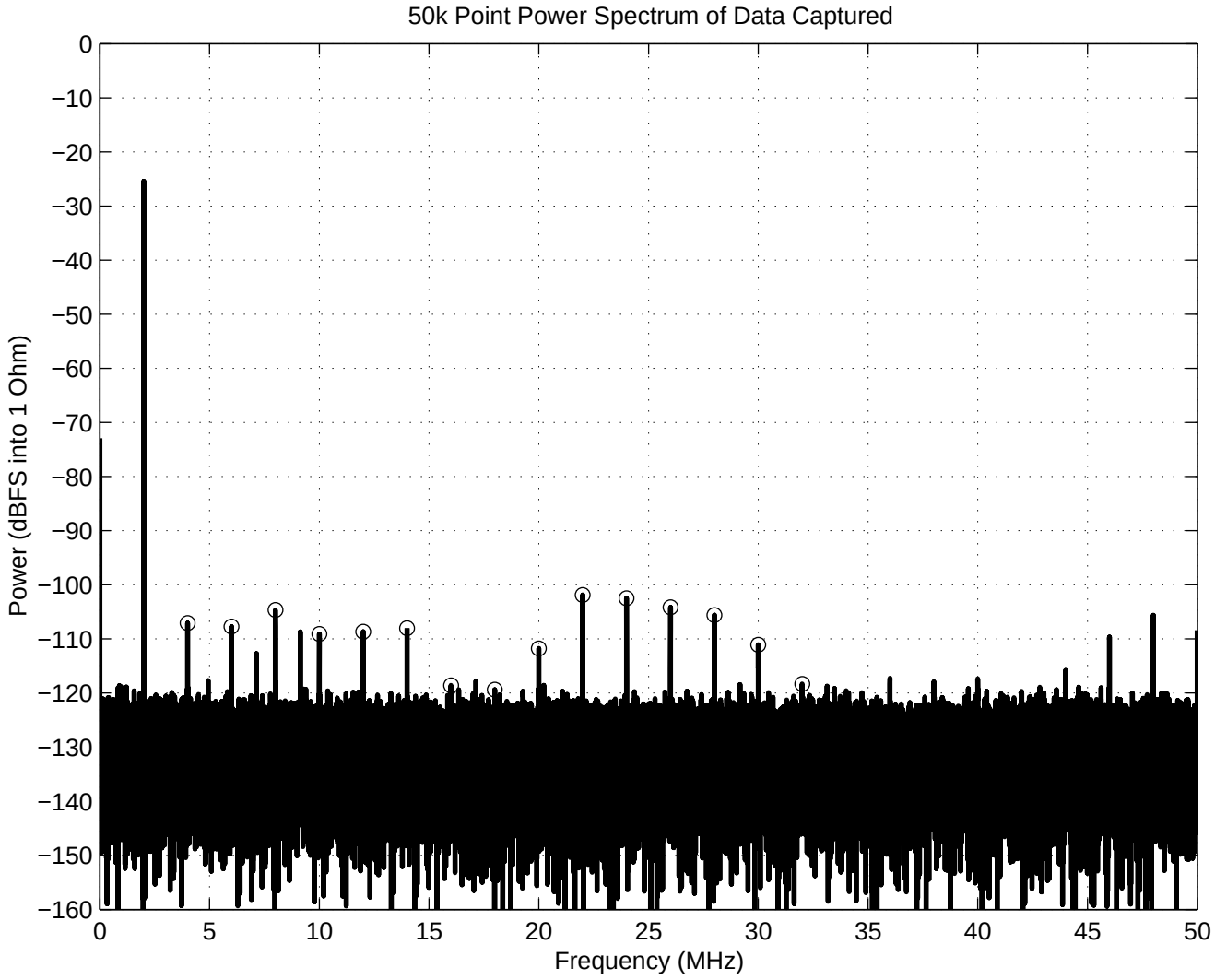


Figure 4.12: Spectrum of filtered 2 MHz HP33120A signal

Chapter 5

Conclusions and Recommendations

This chapter begins with a summary of how well the final system performed during testing. It also recommends avenues of improvement for future systems.

5.1 System Performance

The system met all of the basic functional requirements. Data format and operation under various operating conditions is predictable and correct. The performance of the system is summarized in Table 5.1. It shows performance parameters obtained in various tests.[1] The system performs to specification although there is room for improvement and more rigorous testing should be performed.

[1]Some of the data quality test results are likely to be significantly influenced by the use of signal generators of a lower standard than required. Parameters quoted are those found when using the HP8656B signal generator.

Table 5.1: Summary of system performance

Performance Parameter	Performance	Notes
SNR (best)	65 dB	With 7.5 MHz sinusoid
SNR (worst)	42 dB	With 205 MHz sinusoid
THD (best)	-51 dB	With 135 MHz sinusoid
THD (worst)	-35 dB	With 30 MHz sinusoid
SFDR (best)	48 dB	With 105 MHz sinusoid
SFDR (worst)	36	With 30 MHz sinusoid
SINAD (best)	50 dB	With 135 MHz sinusoid
SINAD (worst)	36 dB	With 30 MHz sinusoid
Number overflows	8	In 500 GB of data at $400 \times 10^6 B/s$
Number overflows	3	In 500 GB of data at $350 \times 10^6 B/s$
Number overflows	1	In 500 GB of data at $300 \times 10^6 B/s$
Number overflows	0	In 500 GB of data at $200 \times 10^6 B/s$

5.2 Recommendations for Similar Future Systems

- Future hardware's inclusion of more RAM resources in the FPGA or a more appropriate external RAM module would reduce the probability of overflow significantly.
- Testing of ADCs should be done using good quality signal signal sources to help isolate noise sources to the system under test. Anti-aliasing filters should also be used to get more accurate results.

Appendix A

High Frequency Firmware Design Techniques

This chapter gives information on various techniques used when designing with FPGAs to maximise the clock frequency the system can operate at. The first section gives information on various factors affecting possible clock frequencies in configurable logic. Following is a section outlining methods to maximise clock frequencies when designing and implementing systems using FPGAs.

A.1 Factors Affecting Clock Rates in FPGAs

FPGAs generally consist of basic computational units joined by a configurable data interconnect and a clock distribution network. Various RAM and specialist computational units are also often included. Parameters associated with these components affect the maximum frequency a certain synchronous circuit constructed from these basic units can operate at. (See data sheets for the make up of Altera FPGAs on their website at `http://www.altera.com`).

A.1.1 FPGA Make Up

FPGA manufacturers have various families of FPGAs targeting different application areas. Various parameters are optimised, sometimes at the expense of others, depending on the application at which they are aimed. Some FPGAs are meant to be used in cost-sensitive applications so that expensive manufacturing techniques must be minimised (examples here include the ACEX 1K family from Altera and Spartan-3 family from Xilinx). Others are targeted at applications requiring very high clock frequencies (examples include the Stratix II family from Altera and the Virtex-4

family from Xilinx). Within the various families, devices are sold with a variation in certain parameters such as the amount of configurable logic, pin-count and speed grade. Technology to increase possible clock rates is applied within families to produce high speed versions of generic chips (for information on the Altera family of FPGAs see `http://www.altera.com/products/devices` and for the Xilinx family see `http://www.xilinx.com/xlnx/xil_prodcat_landingpage.jsp`).

A.1.2 Fan Out

Consider a digital pulse starting at a single source and propagating to multiple loads. The signal travels down a path with occasional connections off this main path to different destinations. At each path off to a destination, a small reflection of signal occurs. The first edge of the pulse is distorted slightly leading to a longer rise time at targets further down the pulse's path. This leads to signal skew. High fan out can lead to violation of set up times for data signals and clock skew in clock signals [15, page 349]. Corruption of data and clock signals leads to lower possible clock frequencies.

If the output of a certain flip flop has many destination loads it can be assumed that the compiler may find it difficult to place all the destinations in positions within the FPGA to equalise the length of the signal paths to the destination loads. This could be exacerbated if the resources of the target device are heavily used. The propagation delay can thus be expected to be different for the different signals. This could lead to skew between the signals at the loads and a lower maximum possible clock frequency for the logic.

A.1.3 Propagation Delay

The path a data signal must traverse to get from the output of one flip flop to the input of another consists of some time spent in the following components of an FPGA's architecture:

- Asynchronous logic
- Interconnect

Both of these components delay the signal by a certain amount of time that is dependent on the logic used. Due to the limited amount of interconnect available, it can be theorised that some signals may not be routed by the shortest path to their destination. The limited amount of logic resources also means that it can be assumed that the signal destination may be placed far from the signal source. This

sub-optimal placement of logic and use of interconnect seems to become more likely as more of the resources in the FPGA are used as the possible clock frequencies reported by synthesizers decreases rapidly as designs become large. The maximum possible clock frequency is related to this propagation delay and decreases as this delay increases.

A.2 Methods to Maximise Clock Rates

A.2.1 Built-in Resources

FPGA manufacturers invariably include resources in FPGAs to optimise possible clock rates. See sections in the introductory data sheet for the Stratix II family (`http://www.altera.com/literature/hb/stx2/stx2_sii51001.pdf`) for more information on resources implemented for this family.

- Clock domain logic is generally separate from other logic with special purpose multiplexers and clock signal processing components. This ensures minimal, and predictable, clock signal skew and degradation.
- Dedicated signal paths are provided for certain signals (e.g. carry bits for use when implementing adders) so that they do not need to be routed via general interconnect resources thus allowing the adder to operate at a higher clock frequency.
- FPGAs often include built-in components such as multipliers so that these, potentially slow due to being highly coupled, systems do not have to be constructed out of normal logic and interconnect. These components can operate at a higher clock frequency than if they were constructed from configurable logic.

A.2.2 Compiler Options

The performance of a configurable logic design is largely the compiler's responsibility. Very good compilers exist that can do a good job of fitting a complex design to a particular FPGA while maximising the clock rate possible. The task can be time consuming and computationally intensive however, with large designs taking hours to place and route. (Note that this discussion is limited to the compiler included with version 3.0 of the Quartus II software package and some of the options may not be available for other compilers. The way in which the options increase possible clock rates are deduced by the author from the results produced when using them

and from software descriptions). The ways in which the compiler can be used to increase possible clock rates are as follows:

- Specify which signals are to be used (usually deduced automatically) as clock signals and at what speed they are to run. The place and route algorithm can then choose where to position logic so that the required clock rate is achieved. The compiler will often redo place and routing if adequate timing is not achieved as many configurations of logic to achieve the same result are often possible.
- Specify that the compiler should maximise the possible clock rate over other parameters during compilation. This may influence other factors such as causing the amount of logic to increase and should only be done if the possible clock rate is a problem.
- Force certain registers to be placed close together by specifying a maximum signal propagation time between them. This can be tried if certain registers are found to consistently cause problems. This could be assumed cause other registers not be optimally placed however, especially if registers are 'hand placed' by the designer within the FPGA.
- The Quartus II compiler allows signals to be grouped together by the designer into a 'clique' and then attempts to place these physically near each other to optimise timing. This is useful if the design is made of blocks of logic connected by few signals.

A.2.3 Pipelining

Pipelining is a popular means of increasing the possible clock frequency a digital design can run at. The basic idea is to take a data path and divide it into stages that each produce intermediate results. These results are fed to registers that feed the inputs of the next stage. Each stage is designed so that has a similar propagation delay for data being processed as the others. The delay for each stage per clock cycle is a lot lower than if the data had to pass through the whole data path in one clock cycle. This allows the clock rate for the data path to be increased. (See `http://en.wikipedia.org/wiki/Pipeline` for a good discussion on this technique).

Advantages of this technique are the following:

- It increases the clock frequency possible.
- It increases the possible effective data rate.

- If planned for during system design, it can be accommodated with minimum difficulty during implementation.

When considering this technique, the following possible disadvantages must be remembered:

- The added registers can use significant resources. If resources in the FPGA are scarce, this can reduce the clock rate possible as logic is not placed optimally due to lack of space.
- Pipelining is often unneeded. Using compiler options may be just as effective. Many compilers perform pipelining automatically is enabled.
- Data paths become more complex if pipelining is added. This will add to design, implementation and debugging time.
- Adding pipelining to a data path after design and implementation, especially within a complex system, often causes unanticipated effects in the rest of the design and may be very hard to implement successfully.

A.2.4 Duplicating Logic

Logic with a high fan out can reduce the possible clock frequency in a design as previously described in subsection A.1.2. In this case, duplicating the logic supplying the signal to destinations could reduce the fan out and ease the pressure on where logic is placed. This is often done by the compiler automatically if necessary but may need to be enabled. It may also need to be forced as many compilers optimise away duplicate logic automatically. (The compiler in Quartus II seems to automatically optimise away duplicate logic unless it affects the possible clock rate). A disadvantage of this technique is that extra resources are used which may actually cause a reduction of possible clock rate as routing becomes more difficult as previously described.

Appendix B

Analogue to Digital Converter Performance Testing

No analogue to digital converter (ADC) system can represent an arbitrary input analogue signal with perfect accuracy. This chapter gives details on tests performed to quantify the performance of ADC systems in this respect. All of these tests involve spectral analysis and this topic is explored before details on the tests themselves is provided.

B.1 Spectral Analysis

This section gives information on spectral analysis that relates to the testing of ADCs. A proper understanding of this topic is necessary when attempting to understand the tests used to quantify an ADC's performance. The effects of time and frequency-domain sampling are first explored, which leads on to the relationship between the discrete, finite-length Fast Fourier Transform (FFT) (used by most computer Digital Signal Processing packages) and the continuous, infinite length Continuous Fourier Transform (CFT) that is, ideally, what the FFT hopes to represent.

B.1.1 Time Domain Sampling

An ADC generates a discrete version of the input analogue signal. We would like to know to what extent this discrete version of the analogue signal accurately represents the original signal. In [11, chapter 3] this process of time domain sampling is analysed. The results of this analysis are summarized below.

Sampling an analogue signal in the time domain produces a periodic Fourier Trans-

form consisting of an infinite number of scaled and shifted copies of the Fourier Transform of the original, unsampled signal summed together. The shift and scaling factor is determined by the sampling rate in the time domain. If the sampling rate is too low relative to the original signal's bandwidth or the bandwidth is infinite, then copies overlap and information about the original signal is lost. This is known as aliasing and can be solved by band-limiting the input analogue signal and/or by increasing the sampling rate.

B.1.2 Frequency Domain Sampling

In a similar manner to time domain sampling, frequency domain sampling of a Continuous Fourier Transform (CFT) produces a scaled periodic time domain signal when the the result is represented in the time domain. This time domain signal consists of scaled, shifted copies of the original unsampled time domain signal summed together. The scaling and shifting factor is again determined by the sampling rate in the frequency domain. If the sampling rate is not sufficiently high or the original time domain signal not finite in length, aliasing in the time domain will occur [16, section 9.5].

This finding provides hope that a discrete series (as present in digital systems) can accurately represent a continuous time domain signal given appropriate conditions. The next section shows that this is indeed possible and shows such a relationship.

B.1.3 Relationship between the Continuous Fourier Transform and the Fast Fourier Transform

When performing spectral analysis one ideally would like to obtain the infinite-length, analogue CFT of the digital signal. Due to the discrete nature of digital signals and the limited storage resources in a PC, this is not possible.

Some implementation of the FFT is generally used to transform the discrete, time domain samples to a discrete frequency domain representation. The FFT is an optimised algorithm used to calculate the Discrete Fourier Transform (DFT) of a time domain series. Information on the FFT can be found in[11, chapter 9]. The following subsections show that the discrete frequency domain representation produced by the FFT can accurately represent the original signal under certain conditions.

Relationship between the CFT and FFT for time limited signals

The relationship between the FFT and the CFT of a sampled time-limited waveform is derived in [16, section 12.4]. The following is a summary of the relationship derived.

1. The waveform $x(t)$, having CFT $X(\omega)$, under consideration is non-zero between time 0 and T and zero otherwise. Further, it is sampled with period T_s giving exactly N samples between 0 and T.

2. A relationship between the DFT performed on the time domain signal formed and the CFT of the original time domain signal can be found. This relationship is shown in equation B.1.

3. The sum on the right can be seen to be the DFT of the sampled $x(t)$ multiplied by T/N. The sum on the left is a periodic function constructed from copies of the CFT of the original analogue signal shifted by multiples of ω_s (where $\omega_s = 2\pi/T_s$) and summed together (a result of time domain sampling). This periodic function is evaluated at multiples of ω_0 (where $\omega_0 = 2\pi/T$). If one period from $p\omega_s$ to $(p+1)\omega_s$ of this sampled waveform is taken, the result is the FFT.

4. The result of performing an FFT on samples of a time limited waveform is thus identical to taking one period of the waveform obtained by adding scaled, shifted copies of the CFT of the original analogue signal and sampling the result. The magnitude of the shifting and scaling depends on the time domain sampling rate, ω_s and number of samples taken, N. The various parameters are related as follows $T_s = T/N = 2\pi/\omega_s = 2\pi/N\omega_0$

$$\sum_{m=-\infty}^{\infty} X(n\omega_0 - m\omega_s) = \frac{T}{N} \sum_{k=0}^{N-1} x(kT_s) e^{-j2\pi nk/N} (\forall n \in \mathbb{Z}) \tag{B.1}$$

Relationship between the CFT and FFT for periodic time domain signals

Another useful relation is derived in [16, section 12.4]. It is very similar to, and follows from, the relation found above. This relation is between the FFT and the CFT of a sampled periodic function made from copies of the time limited pulse considered previously. The following relationship is found:

1. Create a periodic function $y(t)$ from the time limited pulse $x(t)$ considered in the previous subsection. This is done by placing an infinite number of copies of $x(t)$ next to each other in the time domain. Sample the result as before giving

N samples per copy. This new function has a Fourier Series $Y(n)$ consisting of a series of discrete values equivalent to scaling $X(\omega)$ and evaluating it at $n\omega_0$.

2. The relationship between the CFT line spectrum of $y(t)$ and the FFT taken over one copy of $x(t)$ that created the time domain signal is shown in equation B.2.

3. It can be seen that the FFT representation of the function is again equivalent to a scaled, shifted sum of copies of $y(t)$'s Fourier Series.

$$\sum_{m=-\infty}^{\infty} Y(n - mN) = \frac{1}{N} \sum_{k=0}^{N-1} y(kT_s) e^{-j2\pi nk/N} (\forall n \in \mathbb{Z}) \tag{B.2}$$

B.1.4 Number of Time Domain Samples

Assuming that the sampling period is kept constant, the number of time domain samples captured can be significant when performing an FFT.

Periodic components of signals are often of interest when spectral analysis is used, as they show the products of system non-linearities such as harmonics and inter-modulation distortion products. It is easy to tell the difference between periodic and time limited components if the FFT is used. As can be seen from equation B.2, the magnitude of an FFT sample representing a periodic signal is proportional to the number of samples N. If the number of time domain samples used doubles, the FFT component corresponding to the periodic signal components should double in size as well. If the signal component is not periodic, doubling the number of time domain samples also doubles the time T over which the FFT is evaluated and equation B.1 shows that the FFT is proportional to N/T giving no amplitude change for non-periodic components. Periodic FFT components can thus be 'lifted' above non-periodic components by increasing N, making it easy to distinguish the former from the latter when viewing the magnitude spectrum of the resultant FFT.

Another case where the number of samples is significant is the number of samples chosen for analysis when the signal contains a periodic component to be analysed. If the samples chosen for analysis do not contain an integer number of periods of the periodic signal component, the resultant FFT will not represent the CFT of the component accurately. Referring to the discussion leading to equation B.2 it can be seen that the resultant time domain periodic waveform $y(t)$ constructed will not accurately represent the original periodic signal in this case. This is because $x(t)$ does not contain a full periodic component of the signal as required. See subsection B.1.5 for related information.

A useful technique when using the FFT is known as 'zero-padding'. The discussion on this technique is taken from [11, section 11.2]. This involves padding the time domain samples of the captured waveform with zeros. The resulting FFT will have the corresponding number of extra data points allowing more detail of the CFT the FFT represents to be seen. If this process is viewed with relation to the discussion in B.1.3 it can be seen that adding the zeros does not change the CFT of the original time limited pulse (as the time-limited signal is assumed to be zero where the zeros are added) but increases the rate at which the resulting periodic CFT is sampled giving a higher FFT resolution.

The top left waveform in Figure B.1 shows 32 samples representing exactly 5 cycles of a sinusoid in the time domain. The corresponding magnitude spectrum of the FFT is shown in the waveform to its right. The waveform on the left in the line below shows the same 32 samples but with 32 extra points of zero-padding. The corresponding magnitude spectrum of the FFT is shown to its right. This new spectrum can be seen to contain all of the points contained in the spectrum of the unpadded waveform but also extra points that give more information on the CFT of the time limited signal the FFT is constructed from. The bottom line in this figure shows an extreme version of this phenomenon, with the padding now being increased to a ratio of ninety percent of the time domain samples. The spectrum to the right of this time domain sequence seems a lot different from the spectrum of the unpadded waveform but the only difference is the higher sampling rate of the CFT represented by the larger number of samples. The shape of this waveform is explained in subsection B.1.5.

B.1.5 Windowing

The following discussion is taken from [11, chapter 11]. Additions are made to relate it to previous subsections.

- In subsection B.1.3 it is shown that the FFT of a sampled, time limited signal $z(t)$ is equivalent to the sum of shifted, scaled versions of the CFT of the original time limited signal $y(t)$ evaluated periodically. If a periodic waveform $x(t)$ is constructed from this waveform, the FFT can be shown to be related to the Fourier Series of the resulting waveform in a similar manner as shown in subsection B.1.3. The Fourier Series is a scaled sampled version of the CFT $Y(\omega)$ of the original time limited signal $y(t)$ from which the periodic signal was constructed.

- The time limited, sampled signal $z(t)$ can be seen to be equivalent to taking the entire unsampled, infinite length signal $x(t)$ in the time domain, obtaining

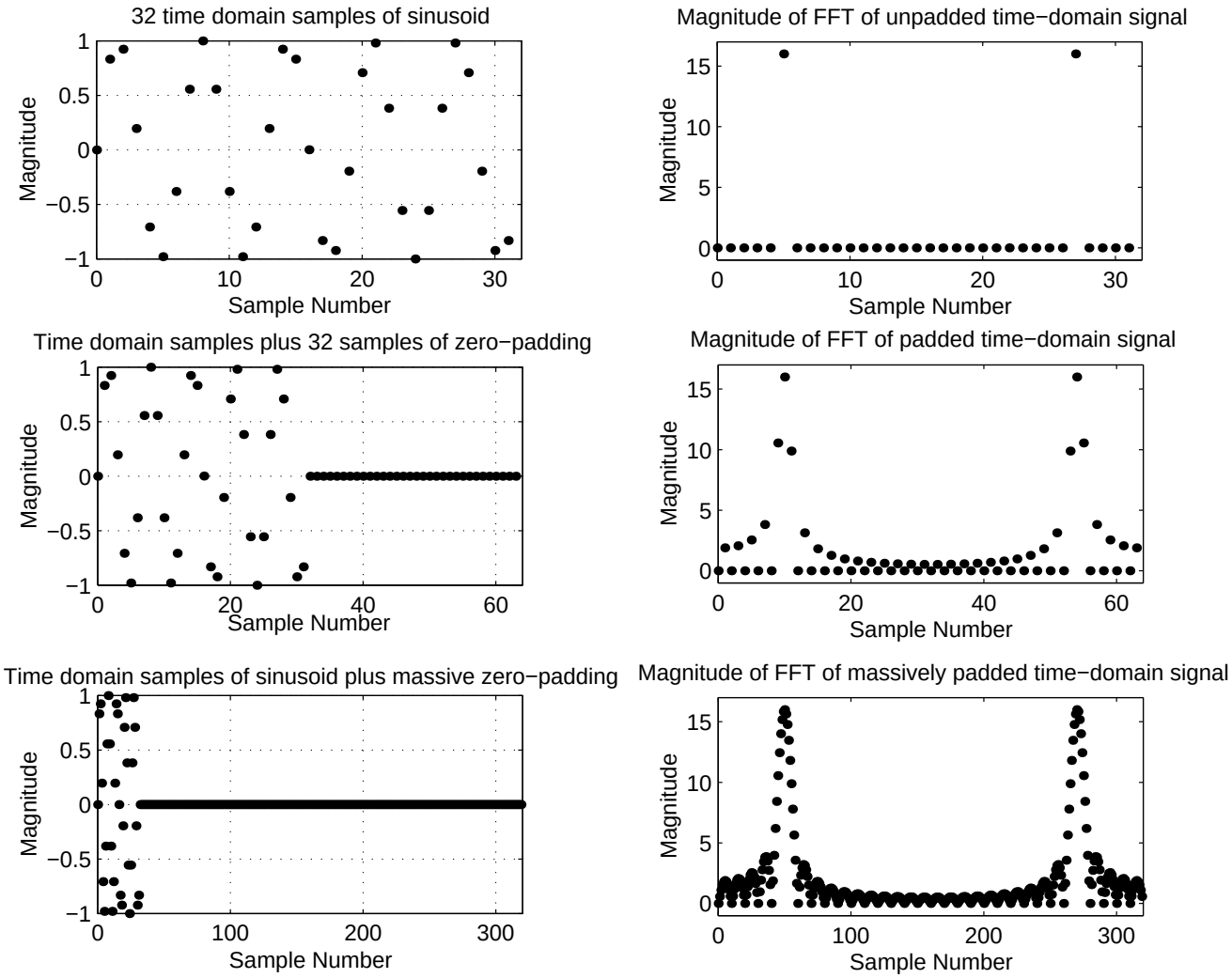


Figure B.1: Time domain and FFT magnitude representation of various degrees of zero-padding

$y(t)$ by multiplying it by a window function $w(t)$, that leaves a band from 0 to T and removes the rest, and sampling the result to give exactly N samples over the period 0 to T. In the frequency domain, $Y(\omega)$ can be seen as convolution of the CFT of the infinite, unsampled time domain signal $X(\omega)$ with the CFT of the window $W(\omega)$. $Z(\omega)$ is then the sum of scaled shifted copies of $Y(\omega)$.

- If no weighting is applied to the input samples, the window can be seen as the following function $w(t) = 1$ for $(0 < t < T)$ else 0. The CFT of this function is $W(\omega) = e^{-j\omega T/2}\frac{sin(\omega T/2)}{\omega T/2}$ whose magnitude is a sinusoidal waveform whose amplitude decays rapidly and which has zero crossings at multiples of $2\pi/T$ (see [11, page 446]) .

- If the signal $x(t)$ is periodic, convolution of $W(\omega)$ with $X(\omega)$ will thus give a waveform $Y(\omega)$ that consists of the sum of scaled, shifted copies of $W(\omega)$ centred at positions where the Fourier Series $X(\omega)$ is non-zero.

- If $y(t)$ is sampled with period T/N, this leads to a CFT $Z(\omega)$ that is periodic with period $2\pi N/T$ and consists of scaled, shifted copies of $Y(\omega)$ as discussed in subsection B.1.1 on time domain sampling. If $Z(\omega)$ is compared to the FFT of $x(t)$ as shown in equation B.2 it can be seen that the FFT is a sampled, scaled version of one period of $Z(\omega)$.

- If the Fourier Series $X(\omega)$ has elements located at multiples of $2\pi/T$ (implying an integer number of cycles in period T), the structure of $W(\omega)$ ensures that the copies of $W(\omega)$ from which $Y(\omega)$ is constructed will not interfere with each other at multiples of $2\pi/T$ as $W(\omega)$ is zero at multiples of $2\pi/T$. If, in addition, the resulting $Z(\omega)$ is scaled and evaluated at multiples of $2\pi/T$ (which would be the case when performing an FFT as the FFT represents N scaled samples of one period of $Z(\omega)$), the resulting waveform will accurately represent the Fourier Series of $x(t)$ (assuming $x(t)$ is sufficiently band-limited and periodic so that aliasing does not occur).

- If the above conditions do not exist, then $W(\omega)$ can be seen to be distorting the correct Fourier Series representation of $x(t)$. If the copies of $W(\omega)$ are not centred at multiples of $2\pi/T$, the samples of $Z(\omega)$ will have remnants of $W(\omega)$ and side-lobes of $W(\omega)$ will affect signals at multiples of $2\pi/T$. Figure B.1 illustrates this phenomenon when $y(t)$ contains an integral number of periods of a sinusoid. The top row shows the time domain sequence and its corresponding FFT magnitude spectrum and lower rows show more detail on how the samples in the FFT magnitude spectrum come from samples of the shifted window functions $W(\omega)$ summed together. As $y(t)$ contains an integral number of periods of the original $x(t)$, sampling the transform here results in the top FFT which is a correct representation of the original $x(t)$ (bar scaling

of course). This is to be contrasted with Figure B.2 where $y(t)$ does not contain an integral number of cycles (in this case 4.5) of the original $x(t)$. As can be seen in the bottom FFT, which is shown next to the time domain samples that generated it, the FFT is far from an accurate representation of the Fourier Series of $x(t)$. It can be seen that the CFT of the windowing function has been sampled incorrectly with significant distortion as a result. If this is the case, other windowing functions $W(\omega)$ should be used that have properties that make the distortion less, depending on the application.

- These, alternative, windowing functions and their properties are discussed in [11, section 7.4] and the following results given. The rectangular window consists of the narrowest main lobe but largest amplitude side lobes in the frequency domain. This leads to it being best when individual frequency components are to be represented, especially when close to one another as leakage between the two is minimal due to the narrowness of the main lobe. The large side lobes mean significant noise addition to a signal from distant signals though. Windows like the Blackman window have very low magnitude side lobes which minimise the distortion from distant signals but a very wide main lobe, causing significant distortion of individual frequency components due to those around them and difficulty in discerning between signals that are close to one another.

B.2 Standard Performance Tests

Various parameters have been devised to help quantify the performance of a specific ADC. These parameters are calculated by spectral analysis of data captured under standard test conditions.

B.2.1 Single Tone Test

In this test, a signal generator is attached to the system inputs and a sinusoid generated. This signal should be as spectrally pure as possible and should be as close to the maximum power level supported by the ADC as possible so that as much of the dynamic range of the system can be tested as possible. This test is repeated at various frequencies through the bandwidth of interest. Although differing in detail, most manufacturers use some variation of this test so that customers can easily compare products using parameters garnered from this test. A brief description of the parameters that can be calculated using this test are given below:

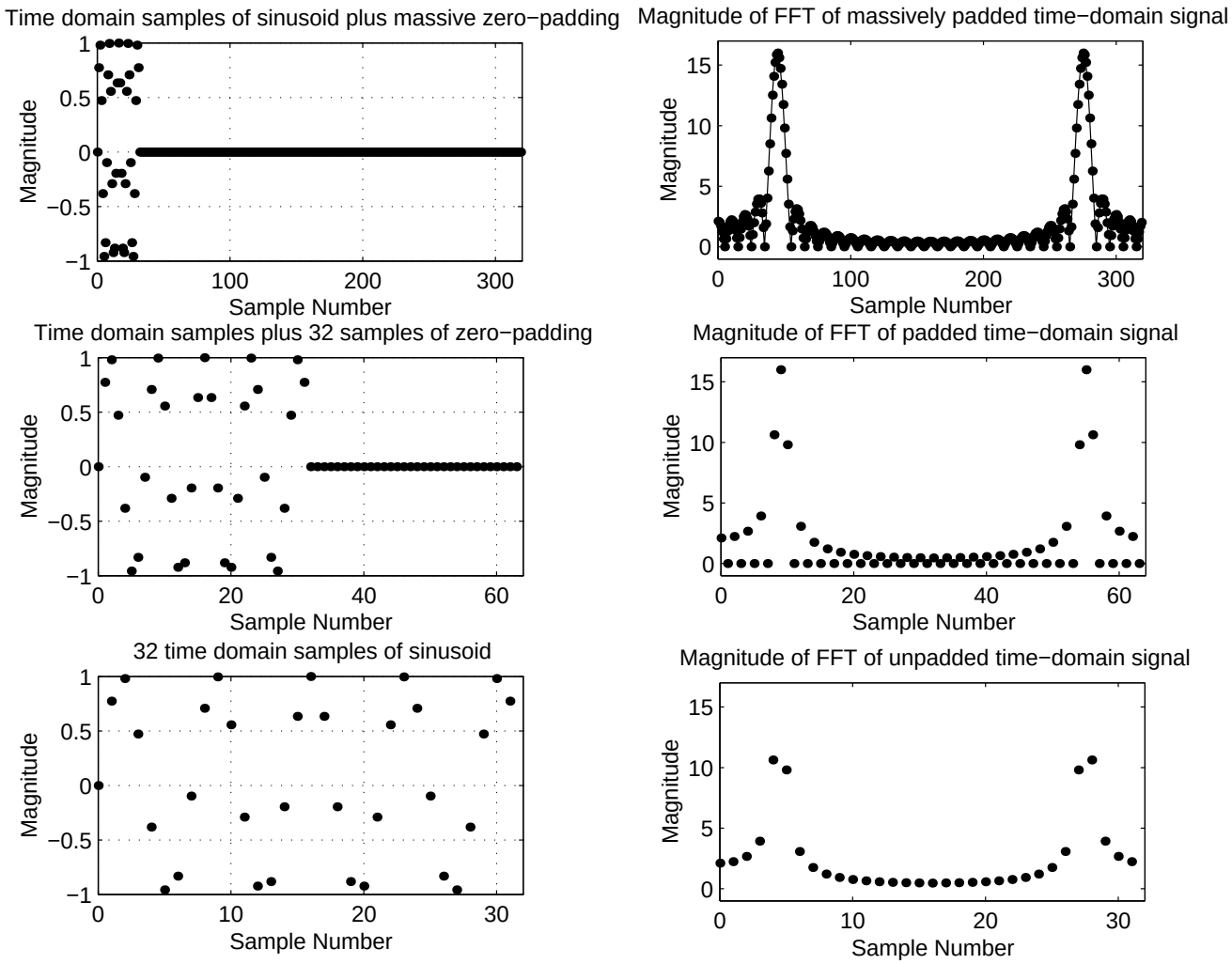


Figure B.2: Sequence illustrating distortion due to unfortunate choice of window function size

SNR Signal to Noise Ratio. The ratio of signal to noise power, excluding DC and a number of harmonics.

ENOB Effective Number of Bits. This takes the SNR and calculates the resolution of the ideal ADC that would produce the same data quality with only quantisation noise effects.

SINAD Signal to Noise and Distortion (Ratio). The ratio of signal power to all noise sources except DC.

SFDR Spurious Free Dynamic Range. The ratio of the largest noise signal, generally assumed to be a harmonic, and peak signal power.

THD Total Harmonic Distortion (Ratio). The ratio of power in a number of harmonics relative to the signal power.

The information on these parameters and the algorithm used to calculate them below is from [4] and [5]. The algorithm used to calculate the various parameters is:

1. Extract a number of samples from the waveform captured. Windowing can be avoided if the frequency of the sinusoidal input and the number of samples are chosen carefully (see subsection B.1.5 for more information).

2. Apply a window function to the time domain samples to reduce spectral leakage if necessary.

3. Perform an FFT on the data to obtain a frequency domain version of the data.

4. Use the FFT results obtained to construct the power spectrum of the signal.

5. Normalise the power spectrum so that power in frequencies are relative to the maximum possible signal power. (See Figure B.3 for the normalised power spectrum obtained from a captured 2 MHz sine wave with harmonics marked).

6. Obtain the indices of the bins containing most of the sinusoid signal's power. This normally includes a number of spectral bins around the fundamental due to distortion from the main lobe of the windowing function used (see section B.1.5). In a similar manner, obtain the indices of the bins containing most of DC power and a number of harmonics. (See Figure B.4 for the normalised power spectrum of the captured 2 MHz sine wave showing all these bands of indices). The signal, harmonics and DC bins shall be excluded from the original power spectrum when calculating the noise power.

7. The indices found can be used to calculate the power in the signal, harmonics, noise and their related parameters.

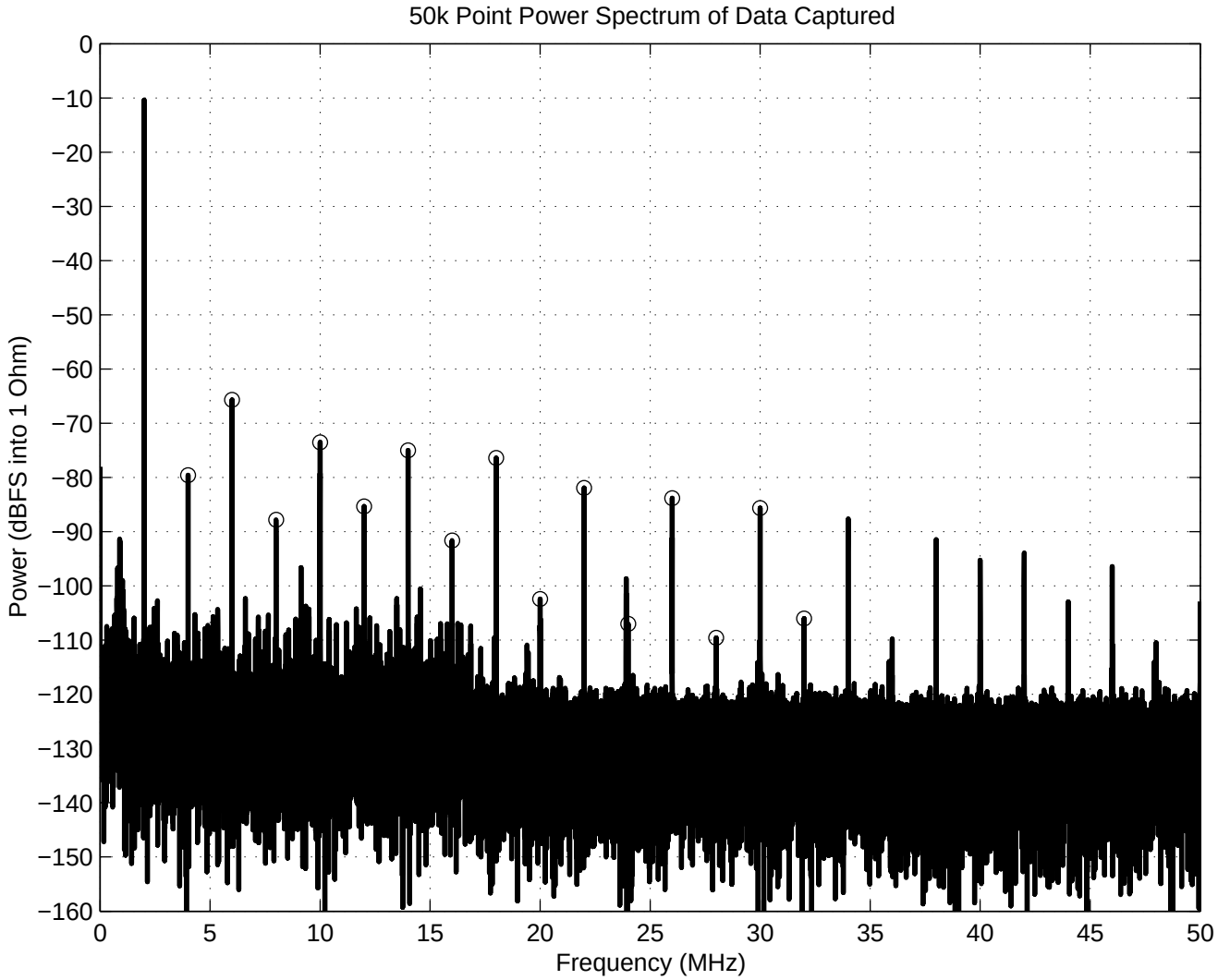


Figure B.3: Power spectrum of captured 2 MHz sine wave with harmonics marked

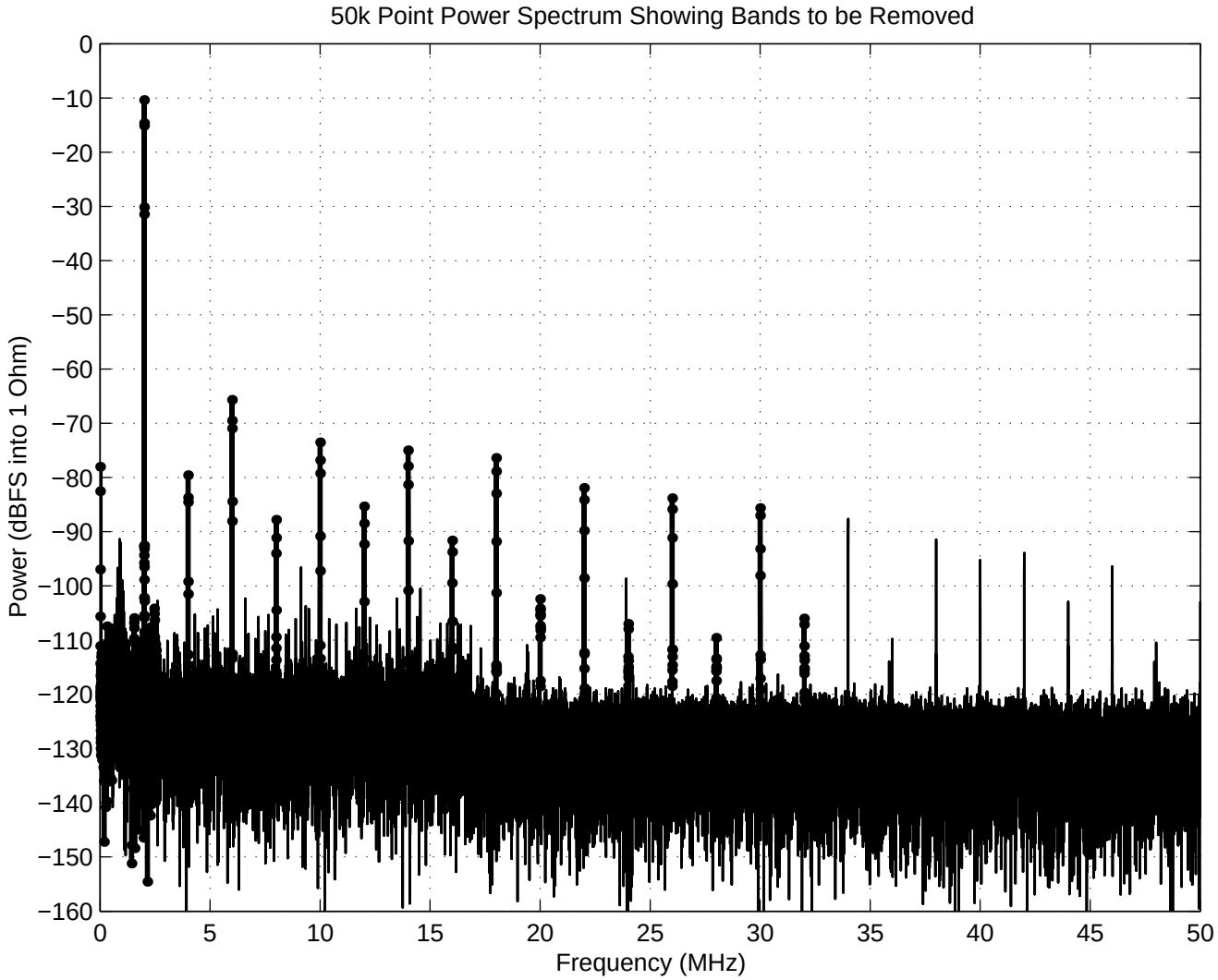


Figure B.4: Power spectrum showing bands to be excluded from noise calculation

B.2.2 Grounded Input Test

A similar test to the one above involves connecting the analogue input of the system to ground via a load that matches the impedance expected by the sampling circuit. Data is then captured and the results analysed. This test shows the noise to be expected from non data related sources (thermal, coupling from power supplies). A signal to noise ratio can be generated by comparing the noise power to a theoretical full-scale sinusoid. This ratio gives the maximum possible data resolution the system could obtain assuming no other noise sources. The algorithm to obtain this ratio is as follows:

- Acquire a large number of data samples.
- Perform an FFT on the samples.
- Generate a power spectrum using the FFT data.
- Normalise the power spectrum by dividing each value by the maximum theoretical power value.
- Remove DC and some of the adjoining bins.
- Sum up the remaining noise components from half a Nyquist Zone. This gives the total noise that would be the SNR if a spectrally pure, full-scale tone had been the input to the system.

B.2.3 Multi-tone Test

A further test described in [4] is often used to test nonlinearities in an ADC. This is identical to the single tone test except that the input analogue signal consists of two frequencies. The results are analysed to test the effects of intermodulation distortion. This test was not performed for this system due to time constraints and the fact that the effects of nonlinearities are tested in single tone tests.

Appendix C

Buffer and Data Path Design

This Appendix details the design of the firmware data path for this project. Various tasks to be performed and their priorities in the system are provided first. Following is a description of resources available to accomplish the tasks. The design of the final system is provided in the final section.

C.1 Task

The data path to be constructed had to conform to the requirements specified in Table C.1. This gives a list of requirements for the data path and extent to which the design had to enforce each of them.

C.2 Resources

The storage elements available to use in the design of the data path are given in Table C.2. Various parameters relevant to the tasks to be accomplished are also given. There proved to be a more than adequate supply of logic resources to implement

Table C.1: Data path requirements

Requirement	Importance
Maximise buffer space available to compensate for stalls	High but only as much as necessary
Allow recovery from stalls in data flow due to latencies in other parts of system	Imperative
Ensure space is always available for new data	Imperative
Configurable resolution	Nice to have
Detection of overflow in buffer and a means of communicating the location and extent of overflow.	Imperative if needed

Table C.2: Storage element resources

Description	Storage Capacity	Dual Port Capable	Output Data Bus Width (bits)	Input Data Bus Width (bits)
On-board RAM resources	6 kB	Yes	Configurable	Configurable
ZBT RAM integrated circuit	4 MB	No	36 (combined in/out)	

the pipelining and logic necessary for the data path in the FPGAs provided.

C.3 Design

At the highest level, the data path can be seen to consist of two processes operating in parallel. The one coordinates the writing of ADC data to the buffer and the other coordinates the reading of data from the buffer and outputting it to the PCI DMA firmware. The buffer thus acts as a large FIFO with the writing process pushing data into it and the reading process pulling data out of it. These processes run independently of each other except for a variable that keeps track of what part of the buffer each process is operating in relative to the other and prevents the processes 'overtaking' each other causing data loss and corruption.

The logic needed to insert a synchronisation word into data at the end of a block proved reasonably simple at the expense of some resources. Figure C.1 shows a modified ASMD chart describing logic used to insert a synchronisation word (see [12, page 195] for more on ASMD charts). This algorithm is implemented in 'Data-Path_Control.vhd' on the accompanying CD in the 'firmware' directory.

The design of the buffer, given the storage resources available, was difficult due to the many trade-offs and unknowns involved.

- The use of the ZBT RAM would increase the size of the buffer by a large factor. This would maximise the number and size of latencies that could be 'absorbed' by the system. The ZBT RAM was not a dual-port module. This would mean that reads and writes could not happen simultaneously and would have to be interleaved. The data bus width was also not sufficient to allow reading at the same rate or higher than writing for data resolutions above 9 bits. This meant that the reading process would always fall behind the writing process when using this module if higher data resolutions were used. Occasional latencies introduced by the rest of the system would exacerbate this and lead to inevitable overflows if this module were used exclusively.

- The onboard RAM resources were ideal for use in this application. FIFOs would be easy to construct and allow simultaneous reading and writing. With

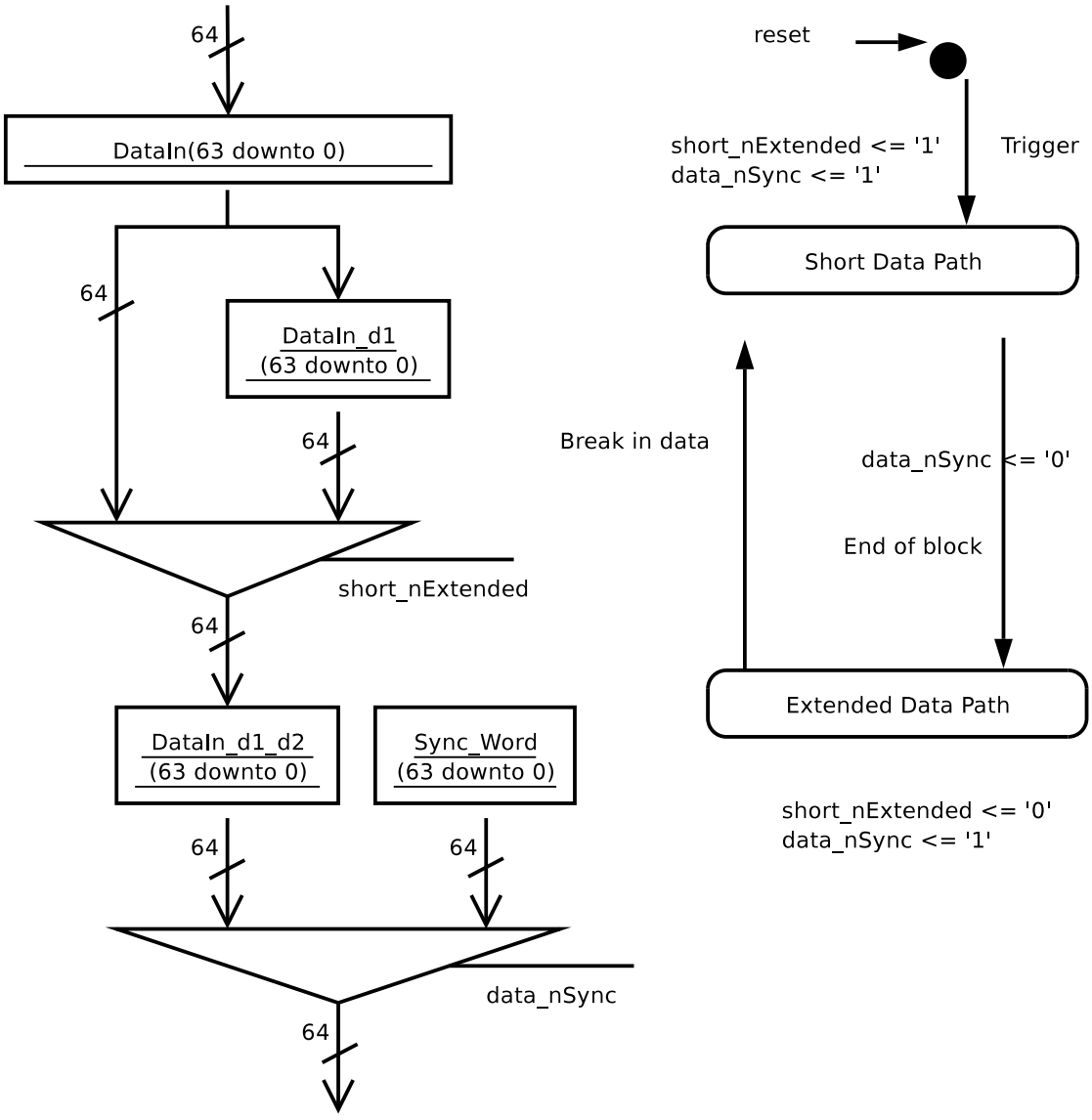


Figure C.1: ASMD chart showing synchronisation word logic

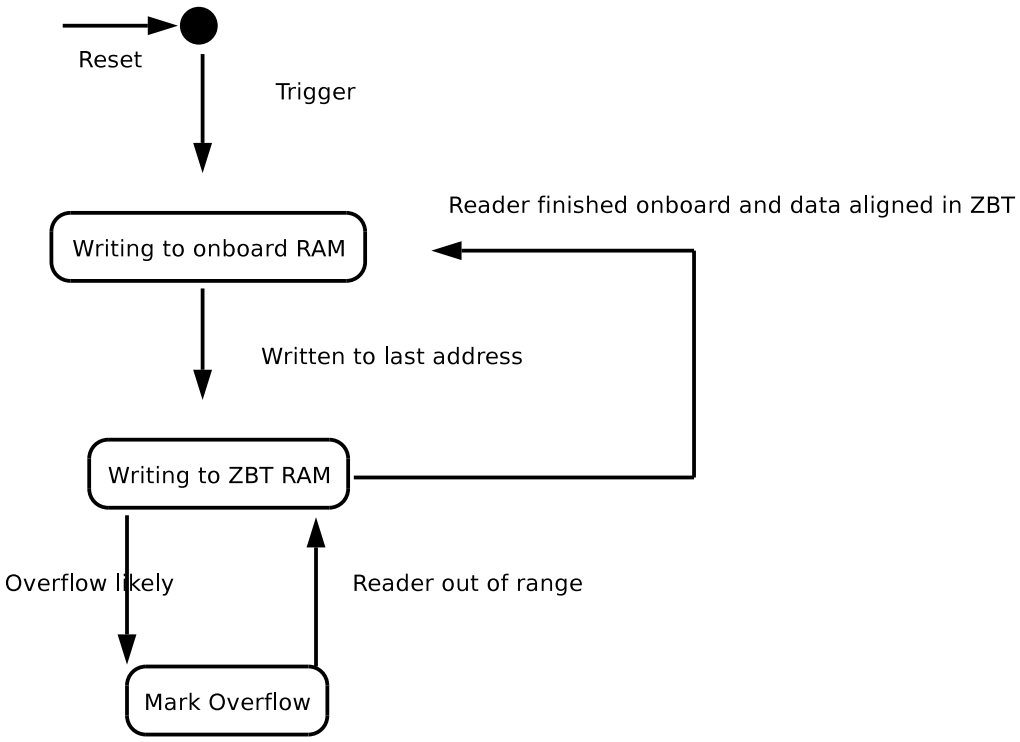


Figure C.2: Complex writer process state machine

the appropriate choice of data bus width, this would allow for the reading process to read at the same or higher rate than the writing process as the PCI DMA firmware would accept data in 64-bit words per clock cycle and data would be produced at a maximum of 32-bits per clock cycle for 14-bit (padded) resolution. However, the amount of RAM resources available was extremely small (6 kB) so that the use of these for buffering alone would decrease the buffer size significantly compared to buffering using the ZBT RAM.

- Using a buffer that combined the use of ZBT and onboard RAM would increase the complexity and size of the system (and thus implementation, testing and debugging time). The ZBT RAM needed a driver to compensate for the pipelining in its use as well as logic to swap between the two RAMs or to produce the view of a large generic RAM block. Inclusion of the ZBT module would mean an algorithm based on careful study of data flow through the PCI subsystem.

The initial 8 bit resolution prototype used both the onboard and ZBT RAM modules. Figure C.2 shows the writing process' state machine and Figure C.3 shows the reading process' state machine. This prototype was the most rugged in terms of the the number and size of latencies it could absorb. It was the most time consuming to implement and test however due to the added complexity, and did not scale easily to higher data resolutions.

Later prototypes were a lot less complex. During testing of the first prototype

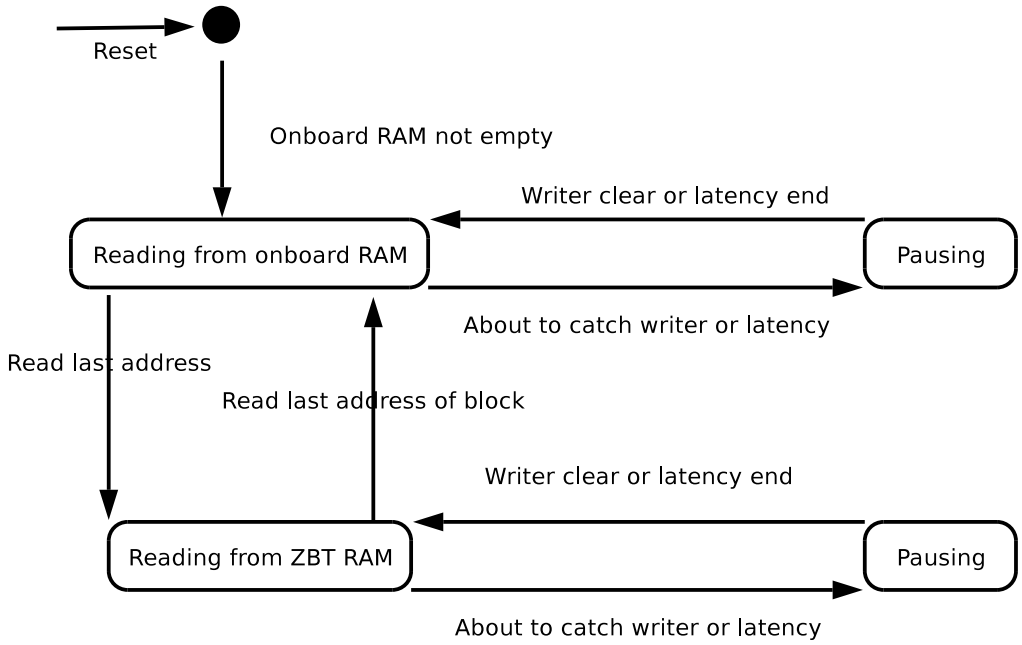
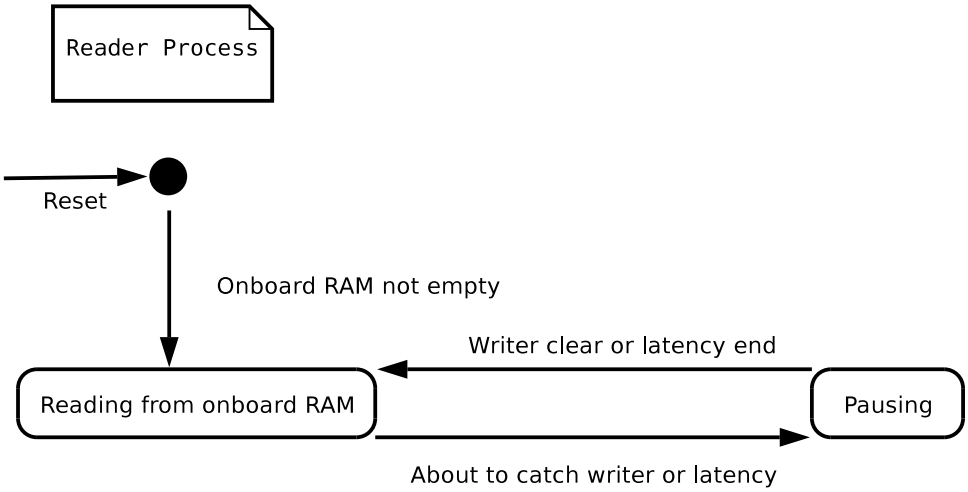


Figure C.3: Complex reader process state machine

it was found that the ZBT RAM was very lightly used and basically unnecessary and it was hoped that this trend would continue in later prototypes. The ZBT RAM was discarded along with the complex logic to control swapping between storage modules and modifications were made to ensure easy scalability between data resolutions. The state machines for the simpler reader and writer processes are shown in Figure C.4. This algorithm was implemented in 'DataPath_Control.vhd' on the accompanying CD in the 'firmware' directory.



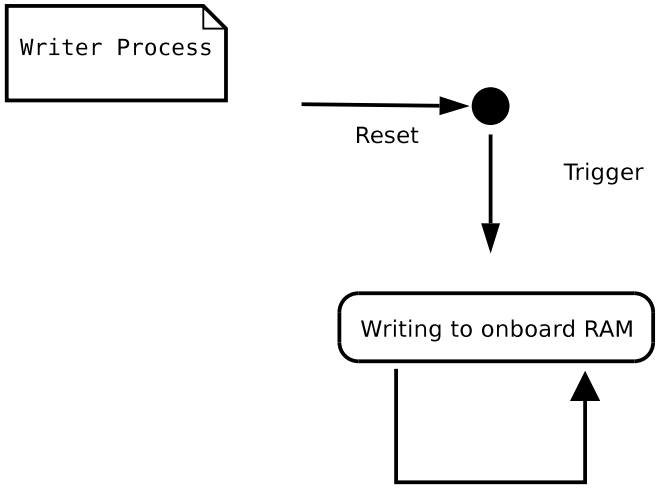


Figure C.4: Simple writer and reader process state machines

Bibliography

[1] PMC to PCI Adapter Board with On-board PCI Bridge. http://www.peritek.com/pmb_p_main.htm.

[2] Operation and Calibration Manual, HP 8656B Synthesized Signal Generator, 1984. Electronic copy obtained from unknown Internet site.

[3] Metastability in Altera Devices ver 4. Technical report, Altera Corporation, 1999.

[4] Defining and Testing Dynamic Parameters in High-Speed ADCs, Part 1. http://www.maxim-ic.com/appnotes.cfm/appnote_number/728, 2001.

[5] Dynamic Testing of High-Speed ADCs, Part 2. http://www.maxim-ic.com/appnotes.cfm/appnote_number/729, 2001.

[6] AD6645 data sheet. http://www.analog.com/productSelection/pdf/AD6645_a.pdf, 2003.

[7] Glossary. http://www.xilinx.com/esp/glossary/z.htm, 2004.

[8] PCI mezzanine card. http://www.wikipedia.org/wiki/PCI_mezzanine_card, 2004.

[9] Peripheral Component Interconnect. http://www.wikipedia.org/wiki/PCI, 2004.

[10] SDR Forum FAQs. http://www.sdrforum.irg/faq.html, 2004.

[11] R. W. S. Alan V. Oppenheim et al. *Discrete-Time Signal Processing*. Prentice Hall, Inc, Engelwood Cliffs, New Jersey 07632, 1989.

[12] M. D. Ciletti. *Advanced Digital Design with the Verilog HDL*. Prentice Hall, Upper Saddle River, New Jersey 07458, 2003.

[13] J. Ganssle. Metastability and Firmware. 2001.

[14] M. Inggs and A. Whitewood. Requirements Review and Specification for High Speed Data Acquisition System. Technical report, Radar and Remote Sensing Group, University of Cape Town, February 2004.

[15] H. Johnson and M. Graham. *High-Speed Digital Design. A Handbook of Black Magic.* Prentice Hall, Inc, Upper Saddle River, New Jersey 07458, 1993.

[16] N. Morrison. *Introduction to Fourier Analysis.* John Wiley & Sons, Inc, New York, 1994.

[17] T. Shanley and D. Anderson. *PCI System Architecture.* 1999.

[18] D. K. Tala. What is Metastability? http://www.deeps.org/tidbits/meta.html, 2001.