Design and Analysis of a Multi-Channel Discriminator Integrated Circuit for Use in Nuclear Physics Experiments

by Bryan Orabutt, Bachelor of Science

A Thesis Submitted in Partial
Fulfillment of the Requirements
for the Degree of
Master of Science
in the field of Electrical Engineering

Advisory Committee:

Dr. George L. Engel, Chair

Dr. Bradley Noble

Dr. Timothy York

Graduate School
Southern Illinois University Edwardsville

August, 2018

ABSTRACT

DESIGN AND ANALYSIS OF A MULTI-CHANNEL DISCRIMINATOR INTEGRATED CIRCUIT FOR USE IN NUCLEAR PHYSICS EXPERIMENTS

by

BRYAN ORABUTT

Chairperson: Professor George L. Engel

This thesis presents the design and simulation of a multi-channel integrated circuit (IC) that will be used in nuclear physics experiments. The chip is being designed as a companion chip for another IC used in particle identification called PSD8C. The IC described in this thesis is used to create precise timing pulses for starting time-to-voltage converters (TVCs) on the PSD8C. These timing pulses are created using a technique called Constant Fraction Discrimination (CFD). Each of the sixteen channels in the IC contains a Nowlin circuit, leading-edge discriminator, zero-cross discriminator, and a one shot circuit to generate the output.

The IC will support input pulse amplitudes between 15 mV and 1.5 V (both positive and negative), and input pulse rise times between 2 nsec and 192 nsec. The IC will feature a programmable output pulse width between 50 nsec and 500 nsec. Most importantly the output pulse firing time variation will be independent of the input amplitude, having a time walk of only 500 psec or less (for input pulse rise time constants of 2 nsec). The IC has been named CFD16C and the design presented is using the AMS-AG 0.35 micron NWELL process.

ACKNOWLEDGEMENTS

I would first like to thank Dr. George Engel for being a continuous source of guidance through all my time working on this project. I would also like to thank Dr. Bradley Noble for encouraging me to investigate challenging problems and for being a source of guidance both in the classroom and out. I would also thank Dr. Timothy York for introducing me to IC design, without him I likely would never have gone to graduate school. I am grateful to Dr. Lee Sobotka and Mr. Jon Elson, department of chemistry, Washington University Saint Louis, for their help during the various stages of this project. My special thanks to all of the faculty and staff of ECE department for their direct and indirect support without which I simply could not have progressed with my work. Additionally, Dr. Gary Mayer of the Computer Science department has helped me expand knowledge beyond the skills learned in the classroom, and I am forever thankful.

I owe a debt of gratitude to my fellow graduate students I've been privileged to work with on this project as well. Pohan Wang, Prarthana Jani, Sneha Edula, and Anil Korkmaz, Sri Kandula, and I all worked together to make CFD16C possible and they have helped make this project a pleasure to work on.

I would not have gotten this far without my friends Jack White, Jared Charter, Andrew Quirin, Nelly Sanchez, and Shana Mankouski who have offered support during all of my endeavors in graduate school. I am forever grateful to my family for being a constant source of encouragement. My mother Marsha Orabutt, brother Sean Orabutt, and Vicki Kern, have all been there for me and I know I could not have come this far without them.

Special thanks to the National Science Foundation (NSF) for funding the work presented in this thesis under NSF Grant#1625499.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

This chapter will introduce the reader to the field of radiation monitoring and describe how custom multi-channel integrated circuits are helping to re-shape this field. The IC described in this thesis, called CFD16C (Constant Fraction Discriminator–16 Channels), is the newest addition to the family of ICs which are being developed by the IC Design Research Laboratory at Southern Illinois University Edwardsville (SIUE) in collaboration with researchers from the Nuclear Reactions Group at Washington University (WUSTL).

1.1 Research Background

The Integrated Circuits Design Research Laboratory at SIUE and the Nuclear Reactions Group at WUSTL have been working (since 2001) on a family of multi-channel custom integrated circuits. The group became interested in developing a family of microchips for use in the detection and measurement of ionizing radiation because: (1) the need for high-density signal processing in the low- and intermediate-energy nuclear physics community is widespread, and (2) no commercial chips were identified that were capable of doing what the researchers wanted, and (3) the scientists deemed it necessary for the experimenter to be in the designer's seat. The goal is to develop a tool box of circuits, useful for researchers working with radioactive ion beams, which can be composed in different ways to meet the researchers' evolving needs and desires.

The group's first success was an analog shape and peak sensing chip with on-board constant-fraction discriminators and sparsified readout. This chip is designed for use with arrays of Si strip detectors of medium scale (with the number of channels ranging from a few hundred to a few thousand) and is known as Heavy-Ion Nuclear Physics–16 Channel (HINP16C).

The second chip, christened Pulse Shape Discrimination–8 Channel(PSD8C), was

designed to logically complement (in terms of detector types) the HINP16C chip. PSD8C performs pulse shape discrimination (PSD), and thus particle identification, if the time dependence of the light output of the scintillator depends on particle type. Moreover, PSD8C uses almost all the same supporting hardware as the HINP16C chip. Both ICs were fabricated in the ON-Semiconductor (formerly AMI) 0.5 mm n-well process (C5N), available through MOSIS (see `www.mosis.com`).

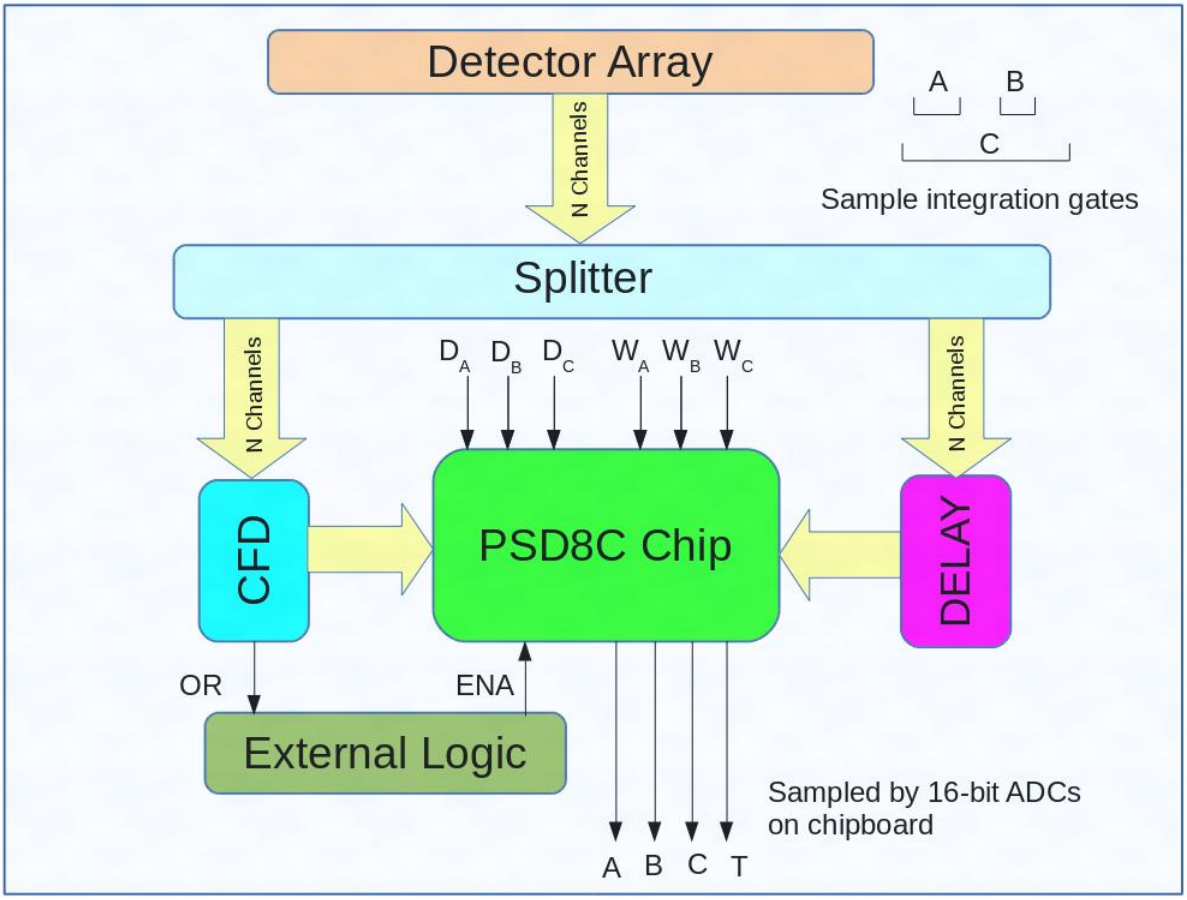


Figure 1.1: Block diagram of typical PSD system.

Figure 1.1 shows a typcial PSD system using the PSD8C IC. The outputs of a detector array are split in two so that a copy of each signal can be sent to both the CFD circuit and the PSD8C. The signals sent to the PSD8C must be delayed to match the propagation delay of the CFD circuit. The CFD logic signals are ANDed with a global enable signal

to provide channel enables for the PSD8C. For each delayed detector signal (and it's assosciated CFD logic signal), three integrations (called A, B, C) will be performed with start times referenced to the CFD signals. An additional amplitude, T, is produced which is proportional to the time difference between the CFD firing and an external common stop reference, which eliminated the need for VME TDCs.

The integrators' starting time delay (D_A, D_B, D_C) and the integration window widths (W_A, W_B, W_C) are controlled by the user. In Figure 1.1, D_A, D_B, D_C are voltages that are converted to times on-chip, along with the widths W_A, W_B, W_C.

1.2 PSD8C IC

Our PSD8C chip greatly simplifies the pulse-processing electronics needed for large arrays of scintillation detectors. Each channel(see Figure 1.2) possesses 3 sub-channels. The sub-channels are referred to as A, B, and C. A sub-channel consists of an integrator and a gate generator. External control voltages (DX, WX) determine the gate delay and the gate width. The structure of a single PSD8C sub-channel is illustrated in Figure 1.3. Because PSD8C employs (user-controlled) multi-region charge integration, particle identification is incorporated into the basic design. Each channel on the chip also contains a TVC that provides relative time information. The pulse height integrals and the relative time are all stored on capacitors and are either reset, after a user-controlled time, or sequentially read out if acquisition of the event is desired (in a manner similar to that of HINP16C).

Features of the first generation PSD8C (Rev. 1) chip include:

- eight independent channels per IC;
- on-chip data sparsification;
- each channel automatically resets itself after a user programmable delay time;

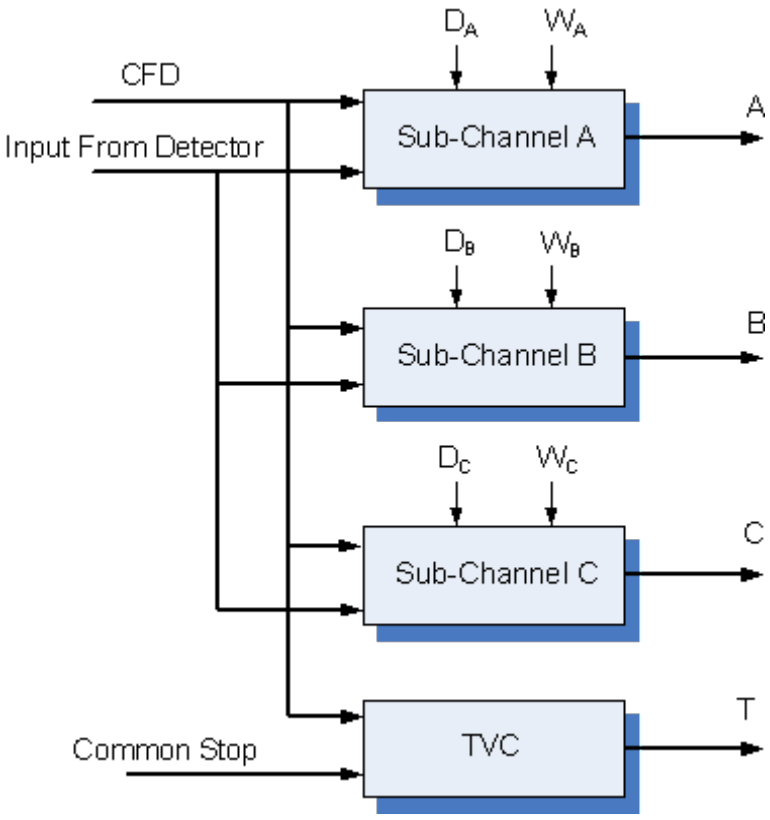
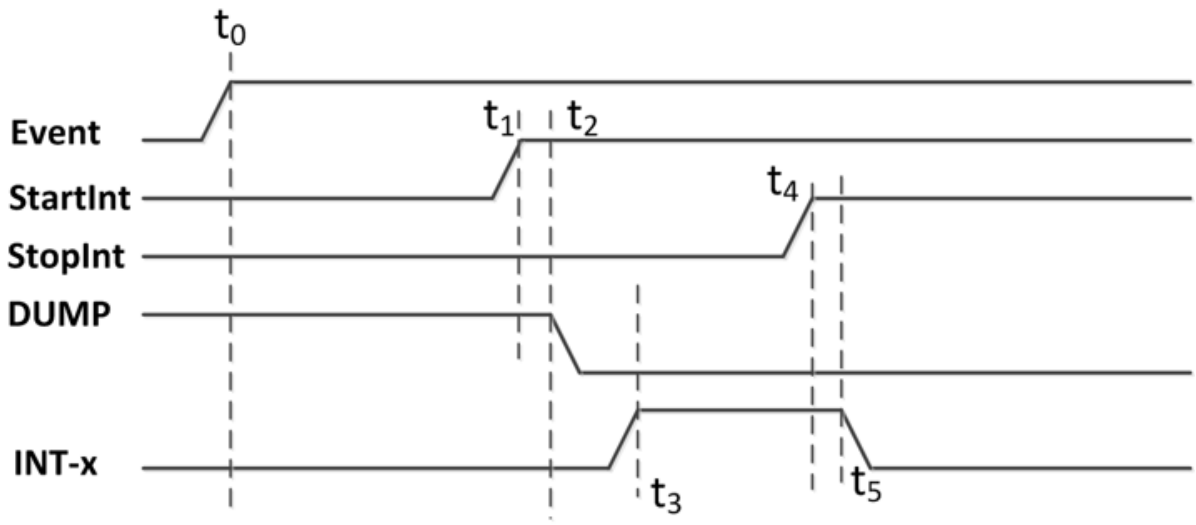


Figure 1.2: PSD Channel



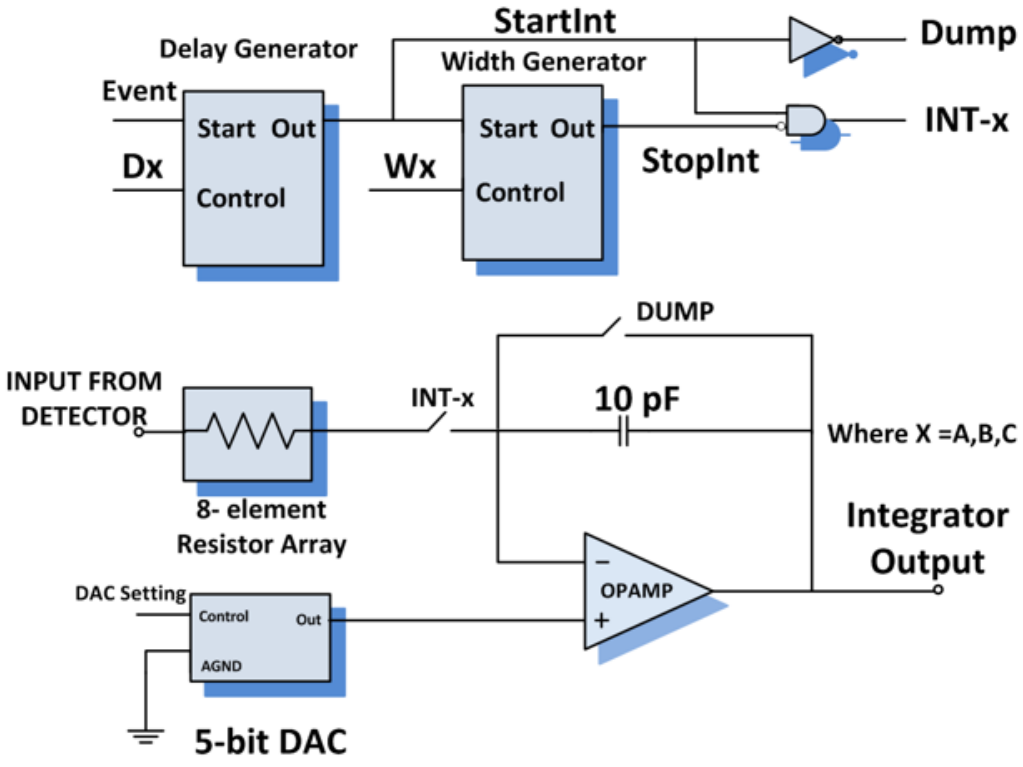


Figure 1.3: PSD Sub-channel.

- three (3) integration regions each with: (a) independent control of time offset (beginning), (b) width (ending) of the integration window, and (c) a menu of eight (8) charging rates;
- each channel possesses a TVC with two time ranges: 500 ns and 2 ms;
- three triggering modes;
- fast logical OR-gate and an analog multiplicity output to aid in trigger decisions;
- two power modes to facilitate use with fast and slow detectors and to thus allow for a more modest power budget for the latter;
- and CFD circuits are not on-chip so as to provide greater flexibility.

PSD8C is described in detail in [Proctor, 2007] and [Hall, 2007]. PSD8C is 2.25 by 5.7 mm^2 and is packaged in a 14 by 14 mm^2, 128 lead thin quad flat pack. Power consumption is 65 mW (low-bias mode)and 150 mW (high-bias mode). A second version (Rev. 2) of PSD8C was submitted for fabrication in May 2010. Rev. 2 attempts to correct several minor problems. First, the TVC circuit could inadvertently be re-started. In Rev. 2, once the rising edge of the "common stop" signal is detected, the TVC cannot re-start until the channel is reset. Second, undesirable temperature dependence (1 $\frac{nsec}{^\circ C}$) in the TVC circuit was identified and traced to the local channel buffer. The buffer was redesigned, and the TVC temperature sensitivity has been greatly reduced (5 $\frac{psec}{^\circ C}$ in the 500 nsec mode, 40 $\frac{psec}{^\circ C}$ in the 2 msec mode). Third, some TVC crosstalk issues were identified and remedied. Fourth, additional shielding was added to the integrator circuits. Finally, at the system level, the chip-boards (printed-circuit boards) were redesigned to include on-board analog to digital converters, or ADCs. (one for each of the chip's analog output pulse trains).

In the latest revision of PSD8C level translators were added to all of the digital pads on the chip. This level translators convert 5 V logic level outputs to 3.3 V logic levels to

be used safely by an field programmable gate array (FPGA). The input level translators perform the opposite function, converting 3.3 V signals into 5 V logic levels so the FPGA can reliably provide data to the PSD8C.

1.3 Need for an Integrated Circuit

While not including the timing circuits on PSD made it more flexible, those circuits are needed. Currently, a large complex board with many ICs produce the timing signals required by the PSD chip. This thesis describes the design of multi-channel integrated circuit which can generate the timing signals for a pair of PSD chips.

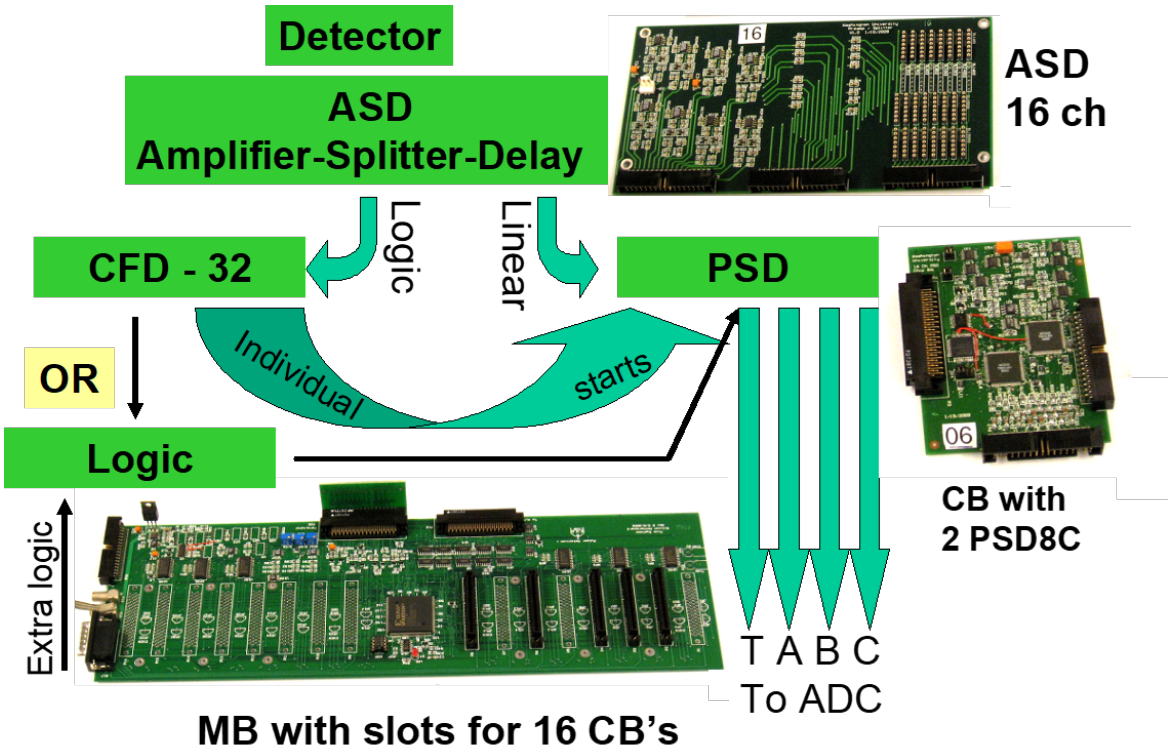


Figure 1.4: PSD system using board-level CFD electronics

1.4 Sample Applications

To focus the reader's attention on what would be possible with the PSD chip complemented by the CFD chip described in this thesis, consider a highly granular discrete element

array for neutron detection using the recently developed inorganic [B.S. Budden, 2015] and plastic [N. Zaitseva, 2012] scintillators with PSD. Such a large array would open the n-rich side up to the kind of high-precision work the Washington University group has done on the p-rich side. (The existing work on the neutron-rich side, done at high energy and with detectors such as MONA-LISA [T. Baumanna, 2005], while providing provocative data on such cases as ^{16}Be [A. Spyrou, 2012] and ^{26}O [Kohley Z., 2015], sufferer for poor statistics and, compared to the proton-rich side, poor resolution.) An array to be deployed at low (reaccelerated beam) energies with thousands of optically isolated PSD elements made from the new generation of plastics, would revolutionize the study of multiple n-decay from what are generally high-isospin states. (The problem of detector-to-detector scattering cross-talk can also be improved with discrete pixilation rather than using large bars by corrugating the detectors in the same way as the conventional discrete array DEMON has [I. Tilquin, 1995]).

While we are enamored with the above idea, it is premature to propose such an array before the ground-work for scalable timing electronics, as we describe in this thesis, is successfully completed. (In fact the coupling of the scintillator from Eljen to the new blue sensitive SiPMs from SensL is simple compared to the development of the scalable electronics.) To this end however, we plan to develop a circuit board using the PSD and CFD chips to process signals from the new generation of PSD-capable plastic scintillators [N. Zaitseva, 2012].

The CFD chip described herein with its programmable Nowlin cirucit will allow the WUSTL Nuclear Reactions Group to work with variety of scintillators (LaBr:Ce to CsI:Tl or :Na to standard plastics and, for what might be the most interesting untapped opportunity, the new class of PSD capable plastics [N. Zaitseva, 2012]).

1.5 Object and Scope of Work

The object of this thesis work was to create a multi-channel integrated circuit capable of constant fraction discrimination. This thesis is composed of five chapters. The system level architecture is presented in Chapter 2. Chapter 3 describes the circuit level design of the many sub-circuits that compose the CFD16C. Chapter 4 details the simulated performance of the CFD16C to show that it performs within the intended design specification. Finally Chapter 5 provides a summary, conclusions, and details future work to be done on the CFD16C.

CHAPTER 2

SYSTEM ARCHITECTURE

This chapter will attempt to describe the CFD16C integrated circuit at the system-level. We will start with a detailed list of system requirements and then will describe the high-level architecture of the IC.

2.1 System Specifications

The success of our group over the past 20 years lies on the close working relationship that the IC Design Research Laboratory at Southern Illinois University Edwardsville (SIUE) has had with the Nuclear Reactions Group at Washington University in St. Louis(WUSTL) led by Dr. Lee Sobotka. The IC group here at SIUE and the Nuclear Reactions Group at WUSTL, after lengthy discussions, drafted the following specifications for the IC described here in this thesis.

- The IC should support 16 detectors.
- It should support analog pulses of both polarities (relative to analog signal ground).
- It should accommodate analog exponentially shaped pulses with rise time constants ranging from 2 nsec to 192 nsec.
- It must exhibit "excellent" walk and jitter characteristics for input pulse amplitudes ranging from 15 mV to 1.5 V. The adjective "excellent" will be quantified in a later chapter of this thesis.
- Pulse repetition rates up to 1 KHz must be accommodated.
- The discriminator in each of the 16 channels should be of the constant fraction type (CFD). In CFD discriminators an attenuated version of the input is subtracted from

a delayed version of input waveform and the time at which the difference between the two is equal to zero is used to mark the pulse arrival time. This results in output timing signals independent of pulse amplitude.

- Each channel should have a leading-edge threshold.
- While the chip must support signals with rise time constants ranging from 2 nsec to 192 nsec, performance will be optimized for the shorter time constants.
- The output pulse width from a channel should be programmable.
- The IC should operate from a single 3.3 Volt supply.
- Power consumption of the 16 channel IC should not exceed 350 mW *i.e.* 20 mW per channel with 30 mW budgeted for the circuits common to all channels.
- The IC is not to occupy an area greater than roughly 2 mm x 3 mm. The chip should be packaged in a 64-pin plastic package.

2.2 Features

In order to achieve the intended system design specification many of the analog circuity in the chip is user configurable. Nowlin delay time, leading edge threshold, one-shot pulse width, and lockout times are all able to be configured to the user's needs. Writing to configuration registers is performed using a signal 8-bit wide bus to provide address, mode, and data information. Using the user-controlled *STB* line, address and mode can be presented on the rising edge and then data will be registered into the selected configuration register on the falling edge. Each channel can be individually enabled or disables as per the needs of the user. Additionally, should it be reuqired, all sixteen channels can be disabled with a single global enable pin available to the user. Finally, a test point is provided to give the user feedback about how some of the digital circuit

elements within the channel are performing. This test point can come from eight different nodes within a specified signal channel.

2.3 System-Level Description

The CFD16C is designed in a 0.35 micron CMOS process. The chip is designed to act as a multi-channel constant fraction discriminator with very low jitter and time walk in the output timing pulse. The chip contains sixteen signal channels that are driven by a detector, and a single common channel that contains circuitry used by all of the signal channels. A system level block diagram of a single signal channel can be seen in Figure 2.1.

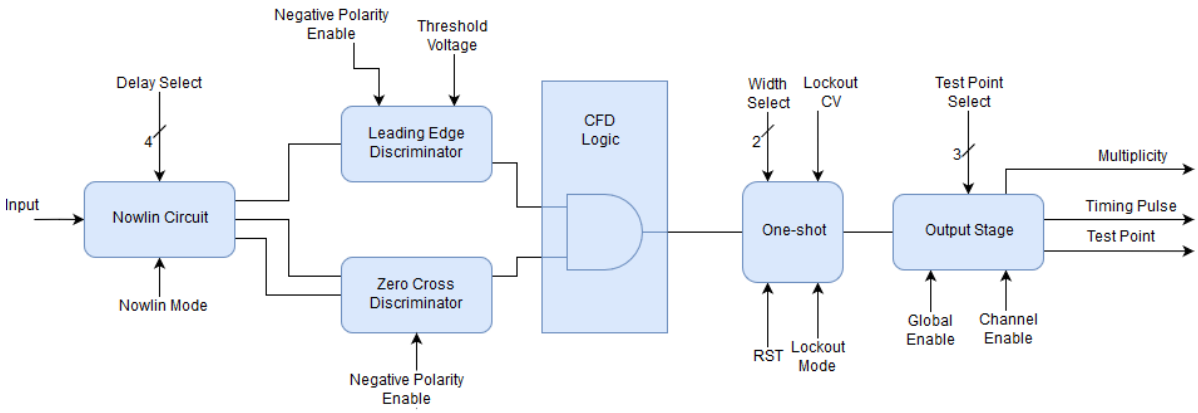


Figure 2.1: System level overview for one channel of the CFD16C

An analog input pulse will arrive at the input stage of of the channel in the form of an exponential voltage pulse with a rise time ≈10 times faster than its fall time. This stage contains a Nowlin circuit that creates a differential output and a high pass filtered output from the input pulse. The differential output is used as input to a zero crossing discriminator while the high pass filtered output is used as input to a leading edge discriminator. The outputs of the two discriminator channels are ANDed together to provide input to a one shot that creates the output timing pulse. Additional outputs such as a test point and multiplicity output are generated in a final output stage of the

channel.

2.3.1 Common Channel

The common channel contains configuration registers to change the performance of the various analog circuits in the signal channels. There are a total of three configuration registers in the common channel and one on the signal channel. These registers can be individually selected and loaded by using an special address and mode scheme. Each channel is assigned a 4-bit address from 0000 to 1111 and each register is assigned a 3-bit mode. To load any specific register the correct mode and address must be provided. A fourth bit of mode, the MSB, is used to select all registers of a given mode regardless of what address is provided. Table 2.1 shows the modes and usage for each of the registers.

While the registers need to be provided with data, address, and mode information, only a single 8-bit bus is used to provide this. On the positive edge of the *STB* input address and mode information are stored in a special purpose register that drives the internal *ADDR* and *MODE* busses (see Figure 2.3). On the negative edge of *STB* data is then stored in all enabled registers.

The common channel also contains biasing circuitry for many of the analog circuits in the signal channels. Bias currents and reference voltages are generated here and distributed to each of the signal channels. More information on these circuits is presented in Chapter 3.

2.3.2 Signal Channel

The input to a signal channel comes from a detector in the form of a pulse with an exponential rise in voltage and an exponential decay. The programmable Nowlin circuit acts as the input stage to the signal channel. The Nowlin circuit is used to create a differential output from this single ended input pulse. One leg of this differential signal is composed of a constant fraction of the input pulse. The other leg of this output is a

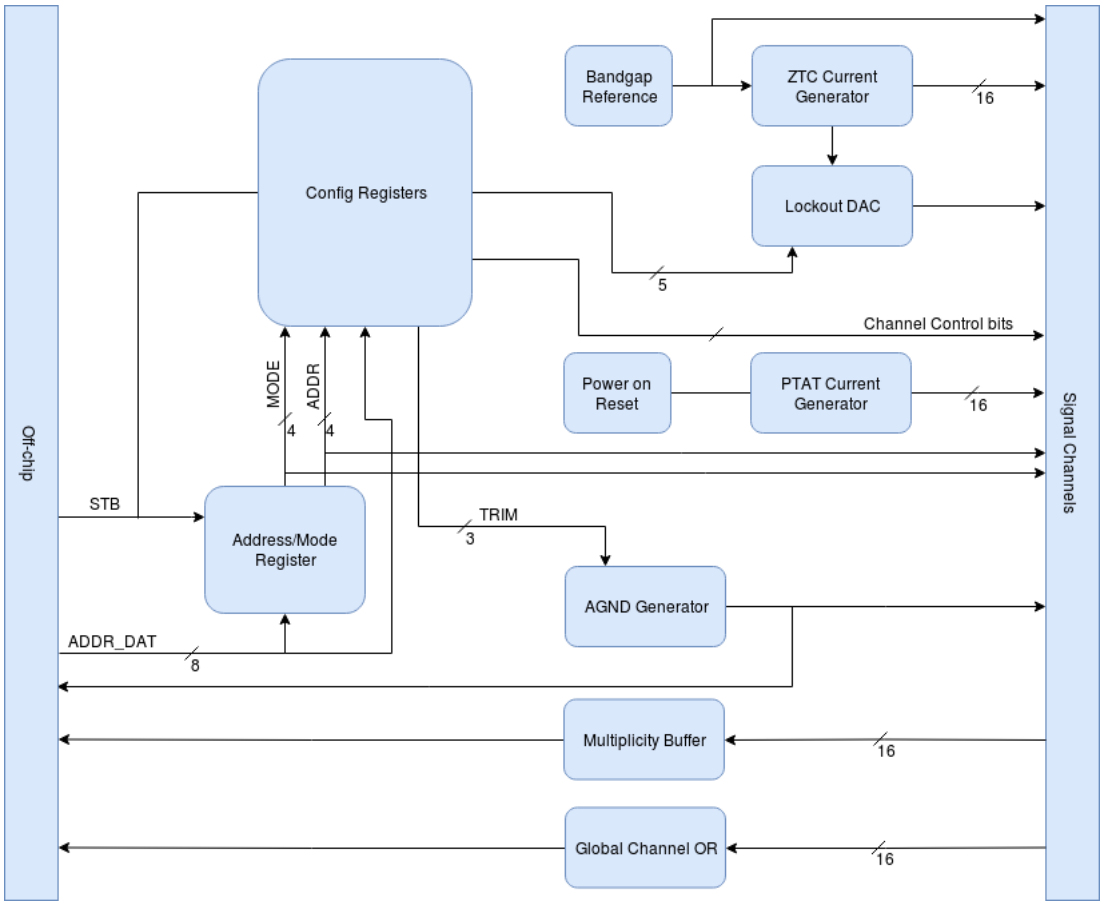


Figure 2.2: System level diagram of the common channel

Figure 2.3: The address, mode, & data shared bus scheme

delayed version of the input pulse. This delay time is determined by an RC time constant that is configurable by changing the value of a programmable capacitor. A high pass filter output is provided by the Nowlin as well, and is used by the leading-edge discriminator circuit.

The differential outputs of the Nowlin circuit are used as inputs to a zero-crossing discriminator (Figure 2.4). This discriminator is created by cascading several amplifiers together and connecting the final output to a high speed comparator. This circuit will provide a digital output that is a logic high (3.3V) when the two differential output voltages from the Nowlin cross the 0V threshold when referenced to analog ground ($AGND$). This will allow the circuit to produce an output independent of the input pulse amplitude [Engel, 2016]. To achieve amplitude independence it is important that the delay time set in the Nowlin circuit is optimal, as seen in Figure 2.5. With a short delay time there is very little under drive in the output of the final differential amplifier which could prevent the comparator from firing. With too much delay there is not enough slew rate to accommodate the fast timings that are needed.

A DC offset cancellation loop is used to remove the effects of systematic DC offsets. Without this DC compensation loop, the output comparator would be permanently stuck in one state regardless of input from the Nowlin circuit. This same DC cancellation loop is also used in the Leading-edge discriminator.

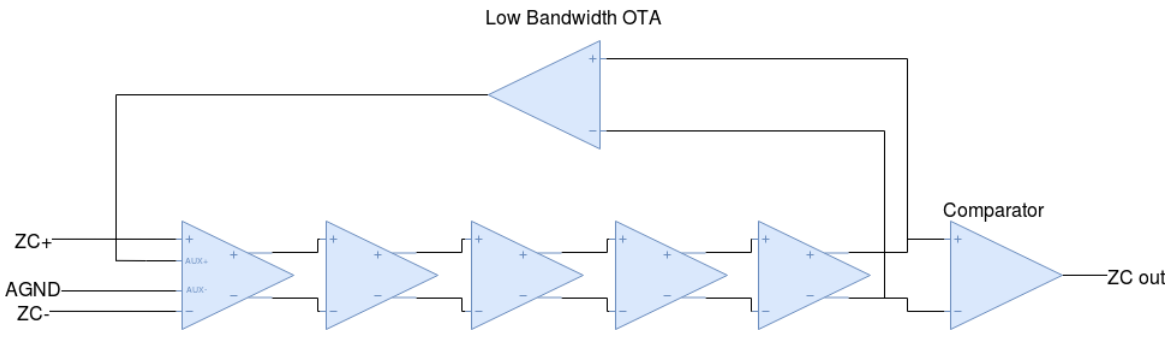


Figure 2.4: Zero cross discriminator with DC cancellation

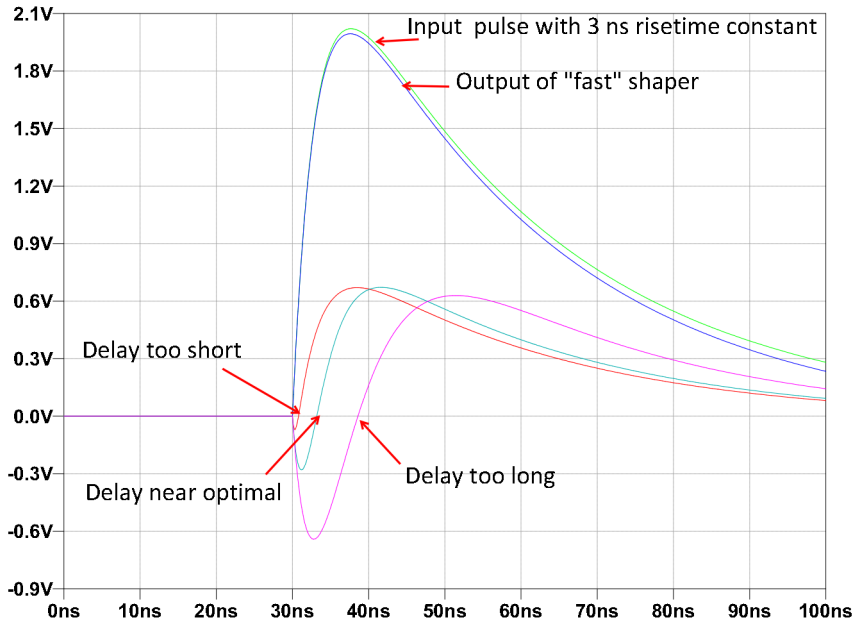


Figure 2.5: Input signal with 3 ns risetime constant showing Nowlin delay effects

The input to the leading-edge discriminator comes from the high-pass filter, or "fast shaper", in the Nowlin circuit. The leading-edge discriminator has a user controllable threshold that is compared against the input from the Nowlin circuit. This threshold will be set just above the noise floor, ensuring that the comparator will only fire in response to a real pulse coming off of a detector and not just inherent noise in the circuit. This threshold can be made negative by applying a logic high (3.3V) to the *NEG_POL* input pin. The output of the leading edge discriminator is then used to qualify the zero-cross detector.

Qualified zero-cross discriminator pulse is used as input for a narrow pulse circuit that triggers the one-shot. Figure 2.6 shows this in more detail. The oneshot circuit creates the channel's output timing pulse. It is provided with a two bit pulse width seleciton bus that allows the user to configure the output width of this pulse between 50 nsec and 500 nsec. There is another one-shot circuit used to create a lockout period which will prevent the creation of an output timing pulse regardless of the presence of any stimuli from the

narrow pulse generator. This lockout time is also user configurable with a control voltage provided by a 5-bit DAC in the common area. The lockout feature can also be completely turned off by the user if desired.

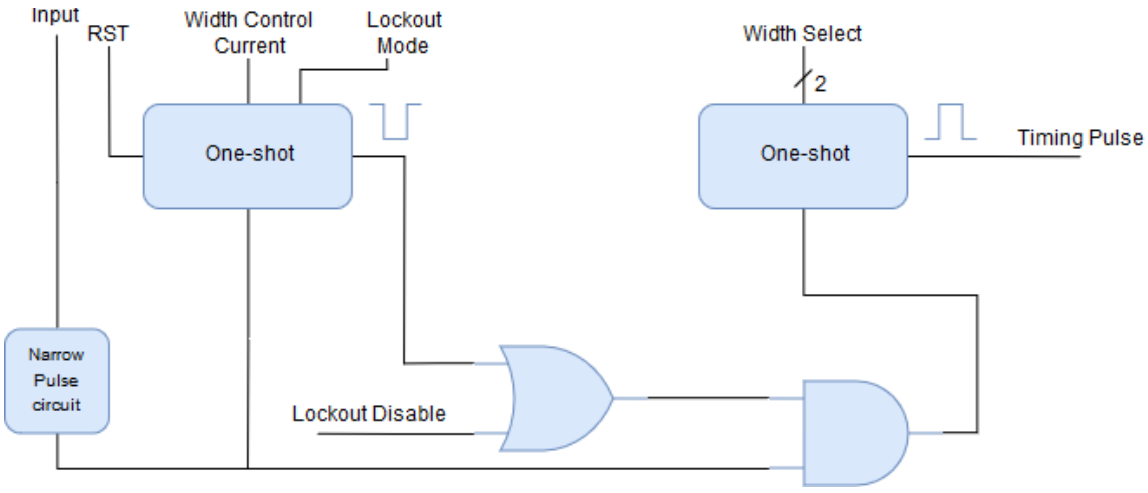


Figure 2.6: System level diagram of one-shot stage

A final output generation stage is used to qualify the timing pulse as well and create other useful channel outputs. The timing pulse should not be present on the pin of the chip package if the global enable signal is not present, or the channel enable bit is not set. Therefor the timing pulse is ANDed with these two signals before going off-chip as seen in Figure 2.7. The digital test point, multiplicity, and global channel OR outputs are also created in this stage and explained in more detail in the next chapter.

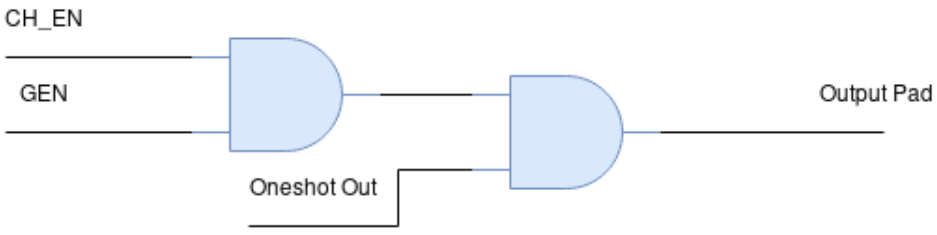


Figure 2.7: Timing pulse output qualification

2.4 Chip Pinout

The CFD16C will be packaged in a 64-pin QFN plastic package. The pinout is detailed in Table 2.2. Pins with no electrical connection to the chip die are labeled as *NC* (no connection).

Mode	Register Bits		
0000	7: Nowlin Mode	6-4: Test point signal select	3-0: Programmable capacitor bus
0001	5: Lockout Mode	4-2: AGND voltage trim	1-0: oneshot width select
0010	N/A		
0011	N/A		
0100	N/A		
0101	6: unused	5: Lockout enable	4-0: Lockout DAC input
0110*	6: Channel enable	5: Leading edge DAC polarity	4-0: Leading edge DAC input
0111	N/A		
1xxx	sends data to all registers of mode 'xxx' regardless of address		
	*A register of this mode is located in each signal channel		

Table 2.1: Register modes and usage

Pin Number	Pin Name	Functionality
1	DOUT0	Channel 0 output
2	CH_OR	Global Channel OR
3-4		NC
5*-12	ADDR_DAT7*-ADDR_DAT0	AD7*-0
13-15		NC
16*-22	AIN0*-AIN6	Channel 0*-6 inputs
23	AGND	Analog reference voltage
24	AVSS	Analog circuit ground
25	AVDD	Analog 3.3V supply
26*-34	AIN7*-AIN15	Channel 7*-15 inputs
35-37		NC
38	NEG_POL	Negative Polarity enable
39	AGND_INT_EN	Internal AGND enable
40	RST_L	Low active reset
41	STB	User controlled strobe
42	GEN	Global enable
43		NC
44	MULT	Multiplicity
45	TP	Test point
46-47		NC
48*-55	DOUT15*-DOUT8	Channel 15*-8 output
56	DVDD	Digital 3.3V supply
57	DGND	Digital circuit ground
58*-64	DOUT7*-DOUT1	Channel 7*-1 output
* used to indicate mapping between pin number and bit/channel number		

Table 2.2: Pinout of CFD16C

CHAPTER 3

ELECTRICAL LEVEL DESIGN

3.1 Fabrication Process

The IC described in this thesis will be fabricated in the AMS-AG 0.35 micron NWELL process. The process supports two poly and 4 metal layers. Double poly capacitors, BJTs, and a high-resistance layer are all available to the designer. NFET device properties are given in Table 3.1 while PFET device properties are available in Table 3.2.

Threshold Voltage	V_{TN}	0.5 V
Transconductance Parameter	K_{PN}	170 $\frac{\mu A}{V^2}$
Bulk Modulation Factor	γ_N	0.6 $V^{\frac{1}{2}}$
Early Voltage per Unit Length	V_{EN}	21.1 $\frac{V}{\mu m}$
Gate Oxide Thickness	t_{ox}	7.6 nm
Gate Oxide Capacitance per Unit Area	C_{ox}	4.5 $\frac{fF}{\mu m^2}$
Threshold Voltage Matching Coefficient	A_{VTN}	9.4 mV $\cdot$ μm
Transconductance Matching Coefficient	A_{KPN}	0.7 % $\cdot$ μm

Table 3.1: NMOS Parameters

3.2 Common Channel

The CFD16C is composed of sixteen signal channels and a single larger common channel. As the name implies, the common channel contains configuration and biasing circuitry that is common to all of the signal channels. These include a power on reset circuit, signal ground generator, bandgap reference, and bias current generators.

Threshold Voltage	V_{TP}	-0.7 V
Transconductance Parameter	K_{PP}	60 $\frac{\mu A}{V^2}$
Bulk Modulation Factor	γ_P	0.4 $V^{\frac{1}{2}}$
Early Voltage per Unit Length	V_{EP}	17.7 $\frac{V}{\mu m}$
Gate Oxide Thickness	t_{ox}	7.6 nm
Gate Oxide Capacitance per Unit Area	C_{ox}	4.5 $\frac{fF}{\mu m^2}$
Threshold Voltage Matching Coefficient	A_{VTP}	14.5 mV $\cdot$ μm
Transconductance Matching Coefficient	A_{KPP}	1.0 % $\cdot$ μm

Table 3.2: PMOS Parameters

3.2.1 Configuration registers

There are three registers in the common channel as well as one in each of the signal channels. These registers were made using digital standard cells provided in the AMS design kit. The registers are D-registers with an enable input. The enable signal is active when a specified register's address and mode has been selected. A single 8-bit wide bus is used to present address, mode, and data to each of the registers. An 8-bit register in the common channel will register *ADDR* and *MODE* on the rising edge of *STB*, with *ADDR* being in the upper four bits. The 8-bits of data are then registered on the falling edge of *STB*. This decoding topology can be seen in Figure 3.1.

The *ADDR* and *MODE* bits are compared against a hard coded value using the XOR and NOR gates. The hard coded values will be set to match the address of the channel the register is in, as well as the mode that should select the specific register. The *ADDR* decoder circuit is then ORed with the MSB of *MODE* so that all registers of a given mode can be selected if the global mode bit is set.

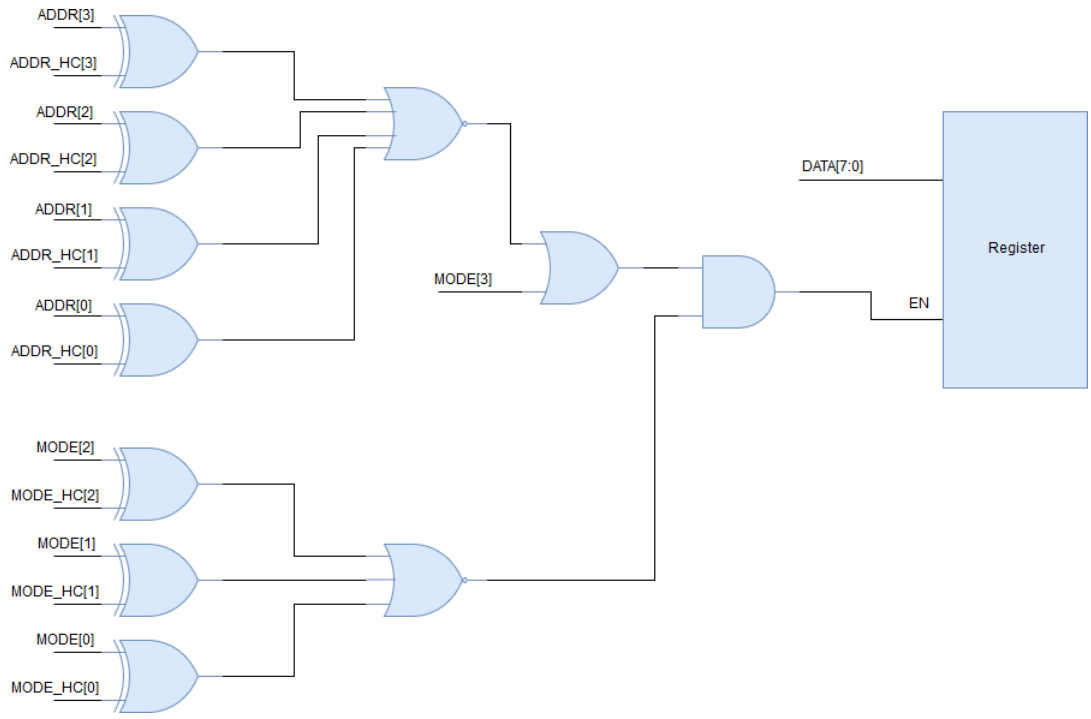


Figure 3.1: Register address and mode decoding

3.2.2 Power on reset circuit

A power on reset circuit is used to start the PTAT bias current generator circuit. The POR circuit was used from the provided analog cells library with the AMS design kit. When power is applied to the CFD16C, the power on reset (POR) circuit generates a single low active reset pulse. This reset pulse is guaranteed to be at least 2 μsec long. The POR signal pulse is used to start the PTAT current reference. This PTAT current reference circuit can converge on a 11.5 μA or 0 A output current when the CFD16C first receives power, but the 0 A solution is not useful. This long reset pulse guarantees that the PTAT current generator will start correctly and provide an 11.5 μA bias current.

3.2.3 Signal ground generator

In each of the signal channels the analog input circuitry is all referenced to a separate signal ground. This signal ground, called *AGND*, must be at a potential half way between

AVDD and *AVSS*. An analog ground generator circuit was provided in an analog cell library from AMS. The signal ground generator circuit can be trimmed to a specific voltage using three trim bits. The signal ground generator has a nominal reference voltage of 1.63 V and can be trimmed by ±250 mV.

3.2.4 Bandgap voltage reference

A bandgap voltage reference is a circuit that provides a temperature and power supply independent voltage [Allen, 2012]. The bandgap voltage reference used in this design was provided in an analog cells library, and is a known working design. The bandgap voltage reference is used to provide a 1.2 V reference point that is independent of temperature or power supply noise. The bandgap voltage is created by generating a PTAT series resistor and a diode-connected parasitic bipolar PNP transistor [Allen, 2012]. The PTAT current has a positive temperature coefficient while the PNP transistor has a negative temperature coefficient creating a nodal voltage with a temperature coefficient of only -87 $\frac{\mu V}{^\circ C}$ as can be seen in Figure 3.2.

This bandgap topology produces an output voltage of $\approx 1.2V$ with near zero temperature independence. The bandgap reference is further filtered by an RC lowpass filter before being used by any other circuits. This is mostly due to the sensitivity of the circuits that use the bandgap reference to noise, rather than an inherently noisy bandgap reference.

3.2.5 PTAT current reference

A PTAT current reference was used from the analog cells library provided by AMS. This current reference produces a bias current between 7.3 μA and 17.8 μA with a nominal value of 11.5 μA. This bias current is proportional to absolute temperature and is used to bias all of the amplifiers on the chip. A weakly or moderately inverted FET has a tranconductance linearly proportional to bias current but inversely proportional

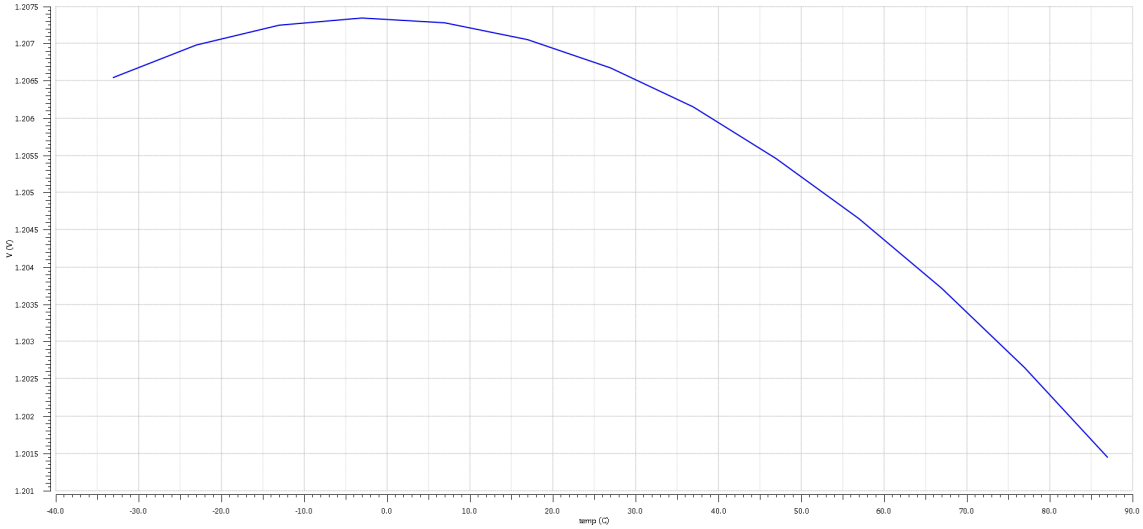


Figure 3.2: Bandgap temperature dependence.

to absolute temperature. Thus by using PTAT currents for the moderately inverted FETs in the amplifier designs, the transconductance of the FETs becomes independent of temperature [Allen, 2012]. Figure 3.3 shows the temperature dependence of the PTAT current reference circuit.

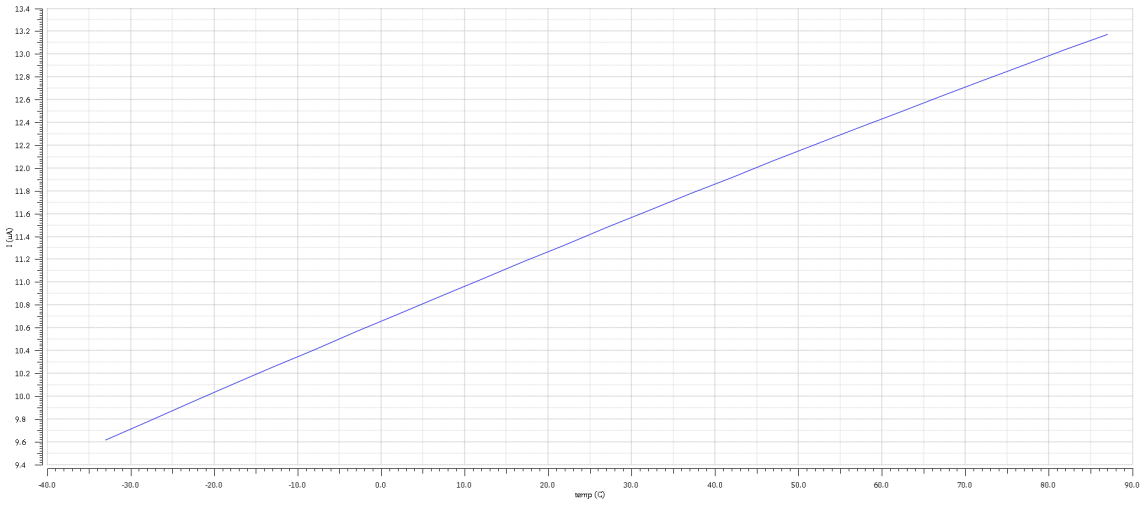


Figure 3.3: PTAT current reference temperature dependence.

3.2.6 Zero-tempco current reference

In order to function correctly, some of the circuitry in the signal channel needs to be biased with a current that has no temperature dependence. This zero temperature coefficient (ZTC) current was generated using an opamp circuit shown in Figure 3.4. In this circuit, the temperature independent bandgap voltage is applied to the inverting terminal of an opamp. The output of the opamp connects to the gate of a PFET whose drain is connected to a zero temperature coefficient 100 kΩ resistor. A feedback connection to the non-inverting terminal of the opamp is made to the drain of the PFET as well.

Because of this feedback connection, the opamp will drive the gate of the PFET to a voltage that ensures there is no potential difference between the inverting and non-inverting terminals of the opamp. This means the voltage drop across the ZTC resistor has to be equal to the bandgap voltage and so the current through the PFET has to be 12 μA. This ZTC current reference produces a current with a temperature dependence of only 2.07 $\frac{nA}{^\circ C}$ as seen in Figure 3.5.

The ZTC resistor is made from a resistor with positive temperature coefficient (rpoly2), and one that has a negative temperature coefficient (rpolyh). The ratio to achieve temperature independence in this process is ≈0.56 rpoly2 to 0.44 rpolyh. Since a 12 μA ZTC current is desired, a 100 kΩ resistor was made using a 56 kΩ rpoly2 resistor and a 44 kΩ rpolyh resistor. This ZTC current is replicated using 17 current mirrors to provide one for each channel, as well as one for the lockout DAC in the common channel. The ZTC currents are used to give the DAC circuits an output that doesn't depend on temperature.

3.2.7 Lockout DAC

It is desirable to prevent the one-shot circuit in the signal channels from firing for a set amount of time after it has already fired. The length of this lockout time may need to

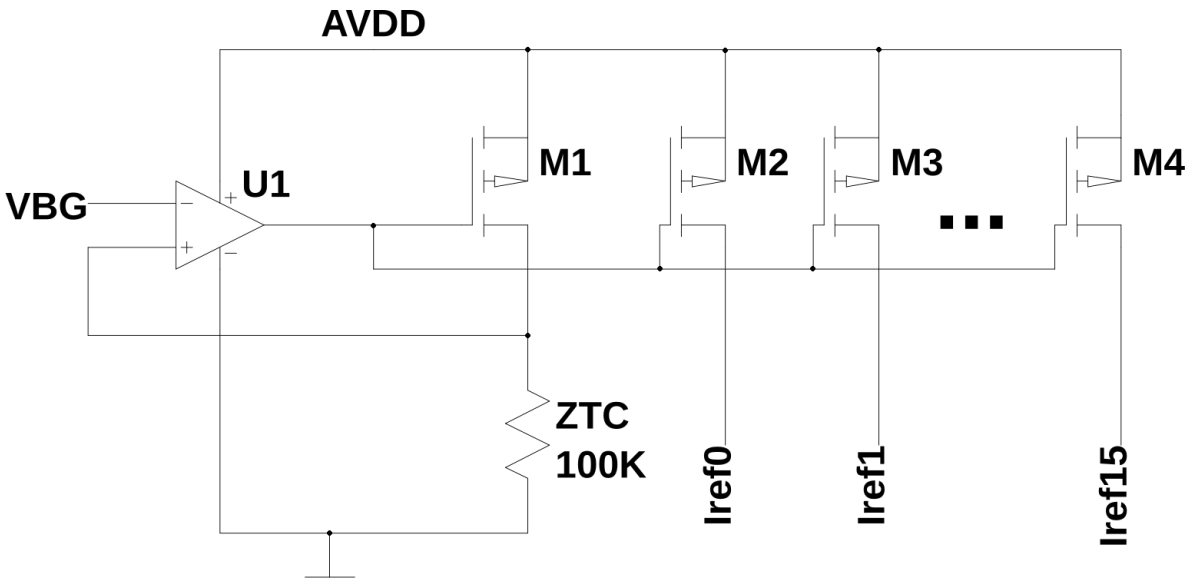


Figure 3.4: Zero temperature coefficient current generator.

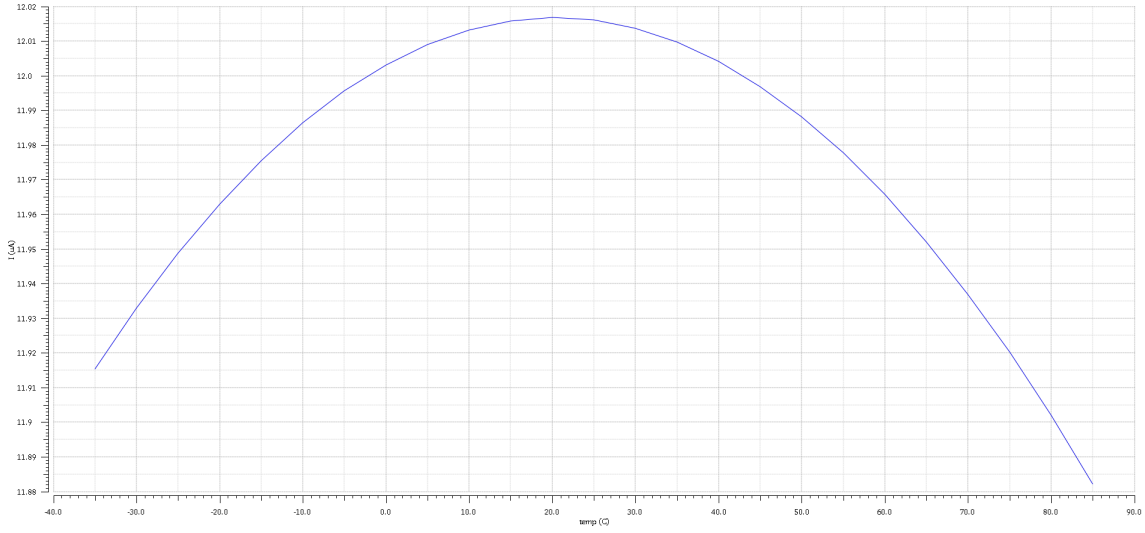


Figure 3.5: Zero temperature coefficient current temperature dependence.

change depending on the nature of the experiment. To do this, a lockout time can be set that is proportional to the voltage at the output of a 6-bit digital to analog converter (DAC).

The DAC itself is implemented as a current scaling DAC using an R2R ladder topology that makes use of MOSFETs as resistors. While MOSFET resistors are generally non-linear and not suitable for use in DAC design, a special technique was used in this design to ensure linearity for current division [Bult and Geelen, 1992]. This technique allowed for the R2R ladder to be made much smaller than using traditional poly resistors, while having very good current division capabilities.

The input to the DAC is a ZTC current coming from the common channel. The R2R ladder equally divides the input current in half and uses this divisor current as the input to the next stage of the DAC, as seen in Figure 3.6. The other half of the current is mirrored using a cascode current mirror to create part of the output current of the DAC (Figure 3.7). The use of a cascode current mirror an extremely high output impedance of $r_{ds7} \cdot g_{m7} \cdot r_{ds6}$. A terminator circuit is used to ensure the current splits in half for the last stage (Stage0).

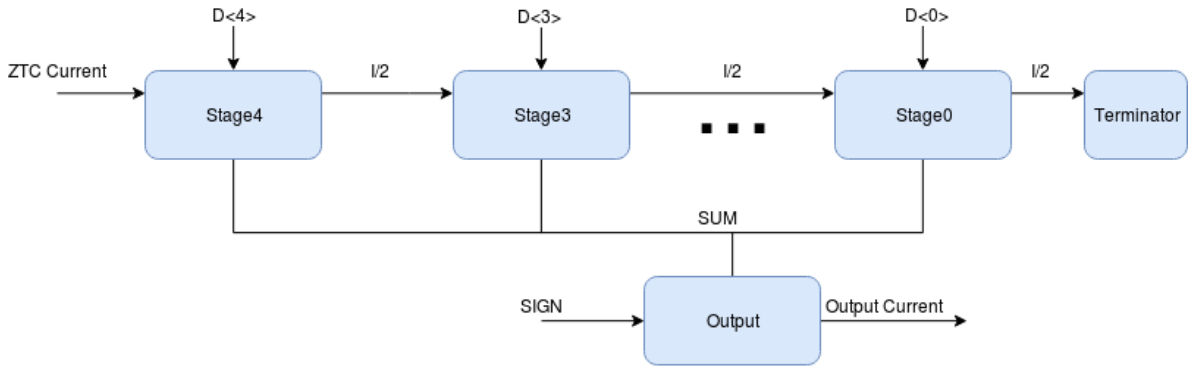


Figure 3.6: 6-bit bipolar DAC

The NFET *M8* in Figure 3.7 acts as a switch to only allow current to flow when the control bit, $D¡i¿$, is a logic 1. The currents are summed up and mirrored with another

cascode current mirror in a final output stage of the DAC (Figure 3.8). Either a PFET current mirror or NFET current mirror will be used depending on the sign bit. If the sign bit is high, meaning negative polarity is enabled, then the NFET current mirror will sink current from the DAC output. Conversely, if the sign bit is high, then the PFET current mirror sources current to the output.

For the lockout voltage, the DAC will only source current to the output where it is passed through a diode connected NFET to create a voltage. The same DAC design is also used for the leading-edge dsicriminator threshold, and must be bipolar in that design.

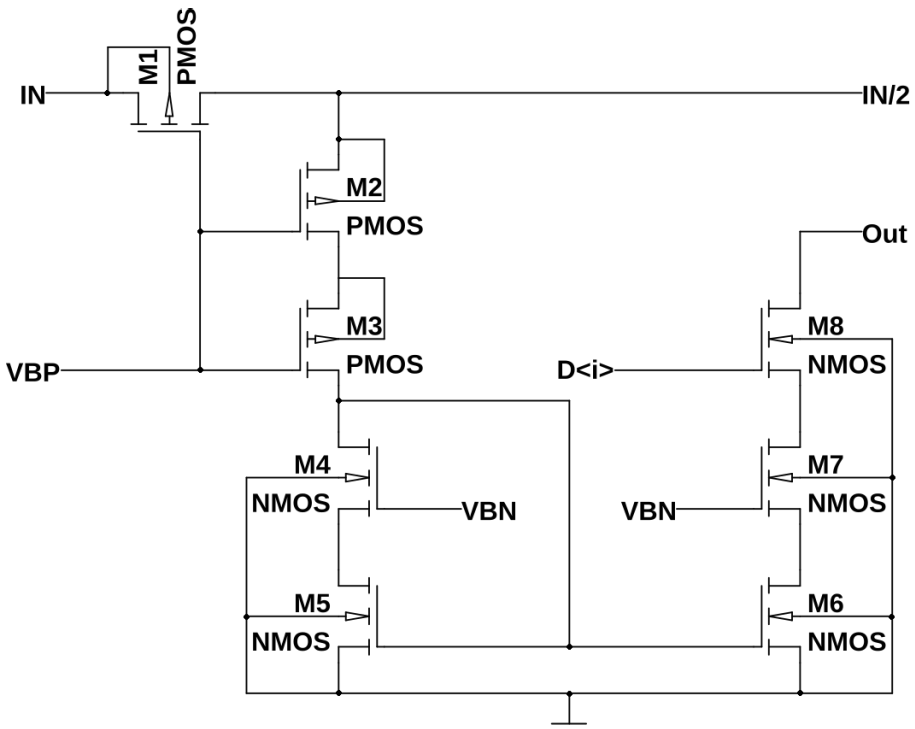


Figure 3.7: One stage of DAC using R2R ladder

3.2.8 Multiplicity output buffer

One of the outputs of the CFD16C is an analog voltage that is proportional to the number of channels that have fired. To create this output each of the signal channels outputs a copy of the PTAT current but only when the channel has fired. All of these

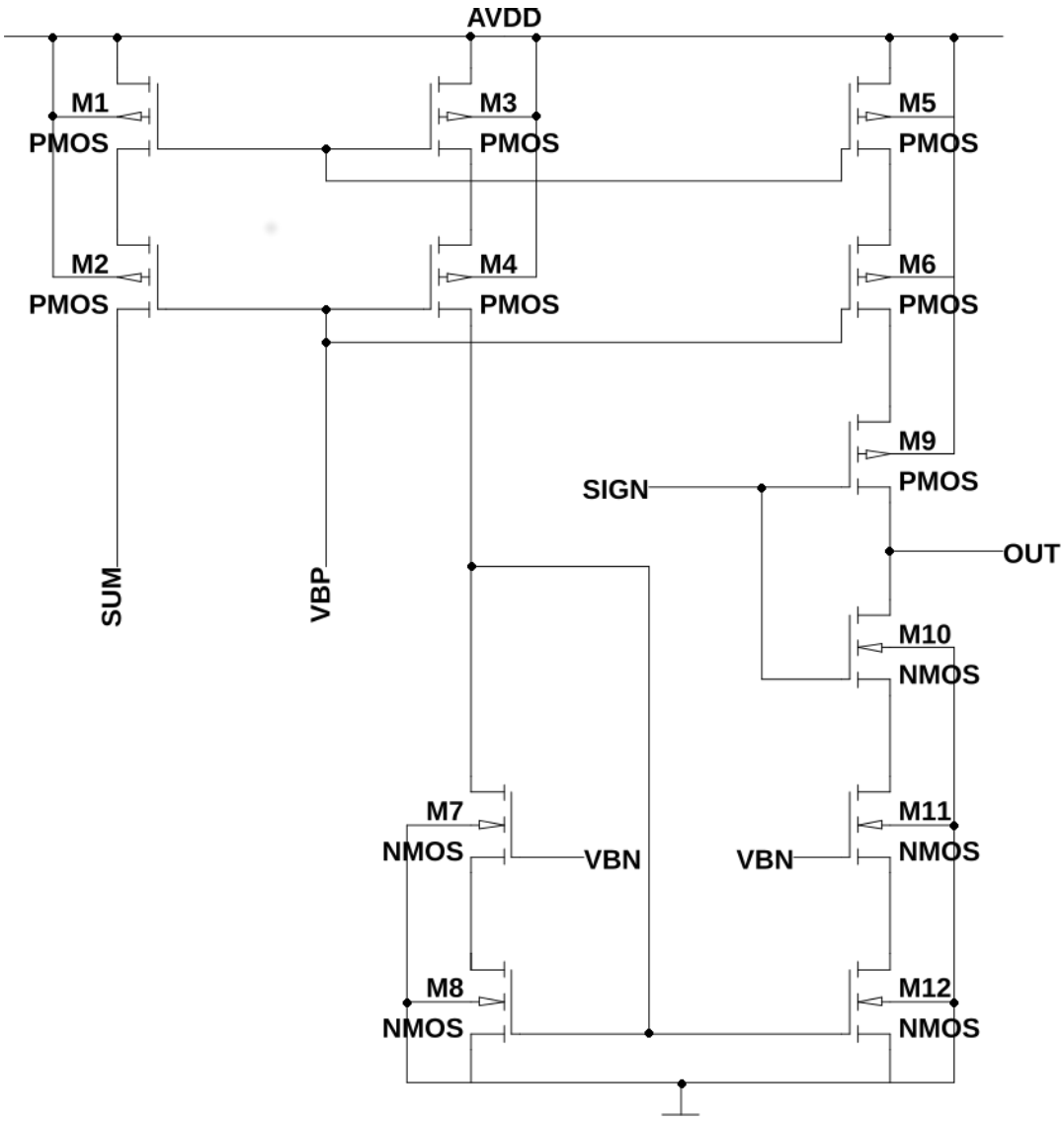


Figure 3.8: DAC output current stage

PTAT currents are summed up using a resistor to create a voltage. However, it is necessary to buffer this output voltage before sending it off chip. For this a source follower output buffer is used to present a high input impedance, but low output impedance of $\frac{1}{g_{mM_2}}$.

The source follower circuit, shown in Figure 3.9, is biased with a 1 mA current using resistor R1. The signal channel multiplicity currents are summed up using R2 creating an output voltage on the source of M3. This output voltage is 1.15 V when no channels have fired and 2.7 V when all sixteen channels have fired.

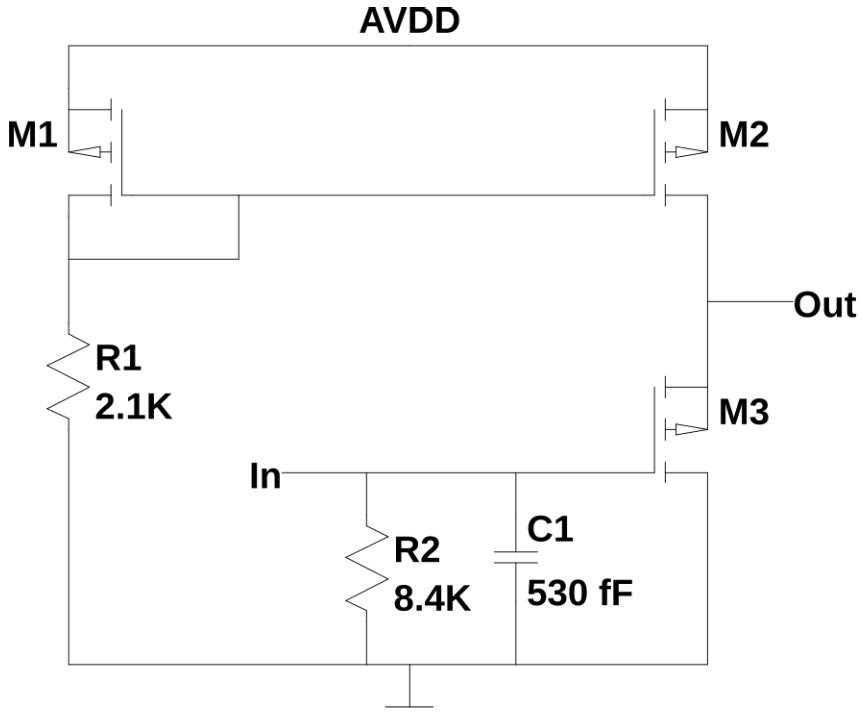


Figure 3.9: Multiplicity output buffer.

3.3 Signal Channel

The CFD16C is made up of sixteen signal channels capable of producing a precise output pulse. Each of the signal channels is identical but contain configurable analog blocks that can be changed on a per channel basis. The signal channel consists of a Nowlin circuit, leading-edge detector circuit, zero-crossing detector circuit, a one-shot circuit, and an output generation circuit.

3.3.1 Programmable Nowlin circuit

The Nowlin circuit is used at the input stage of each of the signal channels and is shown in Figure 3.10. The Nowlin circuit turns a single ended input pulse into a differential signal for use in the zero crossing discriminator. This is achieved by taking a fraction, in this case $\frac{2}{3}$, of the input voltage using a resistor voltage divider (ZC-). The other leg (ZC+) of the differential output is a delayed version of the input pulse. The delay comes from resistor R3 in Figure 3.10 and the programmable capacitor C2. This creates a configurable RC time constant to delay the input signal.

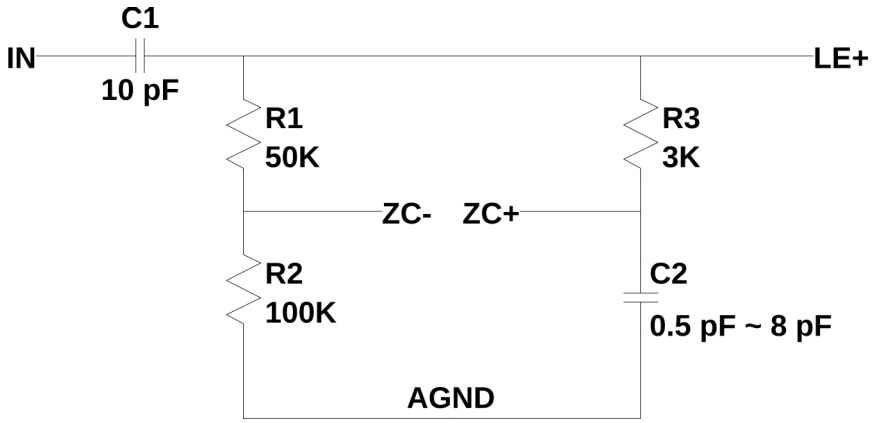


Figure 3.10: Nowlin circuit.

C1 and the series combination of R1 and R2 create a high pass filter whose output is called LE+. The corner frequency of this high pass filter is given by $f_c = \frac{1}{2\pi \cdot (\frac{R1 \cdot R2}{R1+R2}) \cdot C2} \approxeq 475kHz$. This high pass filter output is used as input the the leading edge discriminator. The reason for high pass filtering is to remove any low frequency and DC noise present on the input pin as the actual signal pulse will always be a high frequency exponential pulse.

The programmable capacitor (C2) in the Nowlin circuit is created by using switches, in this case transmission gates, to connect individual capacitors in and out of circuit with the Nowlin. Two transmission gates (t-gates) per capacitor are used to accomplish this. The programmable capacitor circuit can be seen in Figure 3.11.

In this configuration when a bit from the programmable capacitor bus is on (ie. 3.3V), the capacitor connects in parallel with C1 from Figure 3.11, adding more capacitance to the ZC+ node in the Nowlin circuit. If the control signal is off (ie. 0V) then the capacitor is shorted out to $AGND$ discharging it and removing capacitance from the ZC+ node. In total there are four of these capacitor circuits, one for each bit of the programmable capacitor bus. Each capacitor is binary weighted with the following values: 0.5 pF, 1 pF, 2 pF, and 4 pF. This gives a total in circuit capacitance of 8 pF and a minimum in circuit capacitance of 0.5 pF. Table 3.3 shows the bit order of these capacitor elements. Referencing this table shows that the total capacitance at the ZC+ node will be $C_{total} = D \cdot 0.5pF + 0.5pF$ where D is the decimal value of the 4-bit programmable capacitor bus.

The programmable capacitor must be set properly in order to ensure proper operation of the signal channel. Using Table 3.3 an appropriate time constant must be set. This time constant should be chosen such that it is as close as possible to the rise time constant expected from the exponential input pulses coming into the Nowlin circuit.

P-CAP Bus	Capacitance	Time Constant (short)	Time Constant (long)
0000	0.5 pF	1.0 nsec	12.0 nsec
0001	1.0 pF	2.0 nsec	24.0 nsec
0010	1.5 pF	3.0 nsec	36.0 nsec
0011	2.0 pF	4.0 nsec	48.0 nsec
0100	2.5 pF	5.0 nsec	60.0 nsec
0101	3.0 pF	6.0 nsec	72.0 nsec
0110	3.5 pF	7.0 nsec	84.0 nsec
0111	4.0 pF	8.0 nsec	96.0 nsec
1000	4.5 pF	9.0 nsec	108.0 nsec
1001	5.0 pF	10.0 nsec	120.0 nsec
1010	5.5 pF	11.0 nsec	132.0 nsec
1011	6.0 pF	12.0 nsec	144.0 nsec
1100	6.5 pF	13.0 nsec	156.0 nsec
1101	7.0 pF	14.0 nsec	168.0 nsec
1110	7.5 pF	15.0 nsec	180.0 nsec
1111	8.0 pF	16.0 nsec	192.0 nsec

Table 3.3: Programmable capacitor values and time constants.

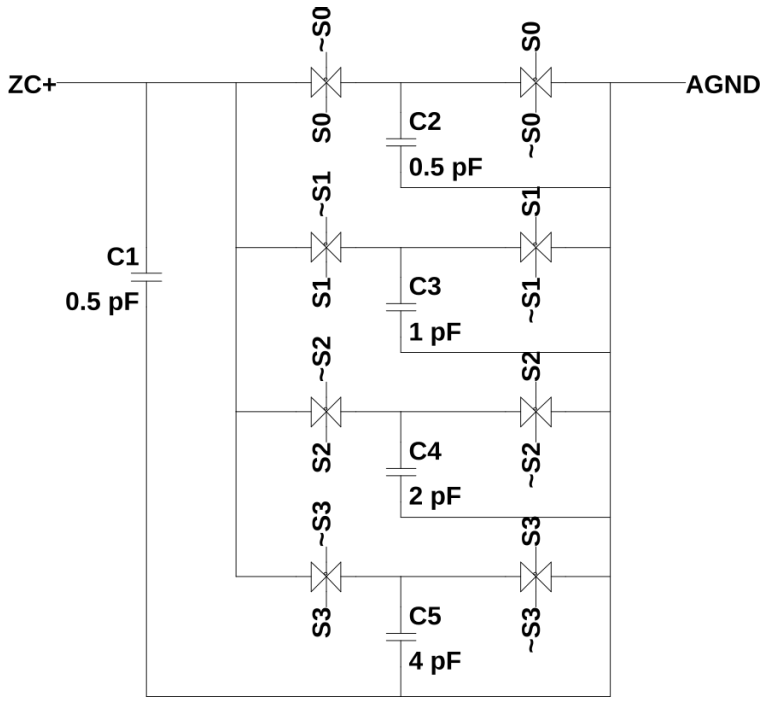


Figure 3.11: Programmable capacitor circuit.

3.3.2 Leading-edge discriminator

3.3.3 Zero-cross discriminator

3.3.4 Output one-shot with lockout features

The output timing pulse for each of the channels is generated from a one-shot circuit with lockout capabilities. A one-shot circuit is a circuit that creates a fixed width output pulse in response to a narrow input pulse. The one-shot circuit in the signal channels is what produces the output timing pulse used to start the TVCs on the PSD8C. The one-shot circuit design is shown in Figure 3.12.

The one-shot circuit works but utilizing a D-type flip flop with an asynchronous clear input. When the CFD circuit (ie. Leading-edge and Zero-cross discriminators) produces an output, the pulse width can be variable depending on the conditions in the experiment. Because of this a narrow pulse generator is used to produce a fixed 5 nsec wide pulse

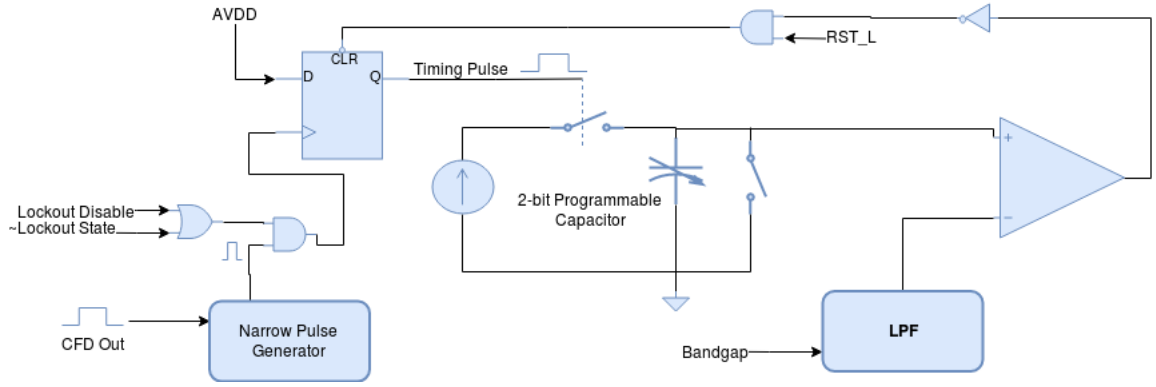


Figure 3.12: One-shot circuit with lockout

in response to any width CFD output. This narrow pulse is used as a clock signal for this D-flip flop, provided lockout is disabled or the circuit is not in a lockout state. Since the input to the flip flop is tied to *AVDD* (logic high) the output will transition to a logic high state in response to the CFD circuit output. This causes a switch to close allowing a current source to charge a capacitor at the input of a comparator. Charging this capacitor produces a ramp voltage on the non-inverting input of the comparator, as seen in Figure 3.13.

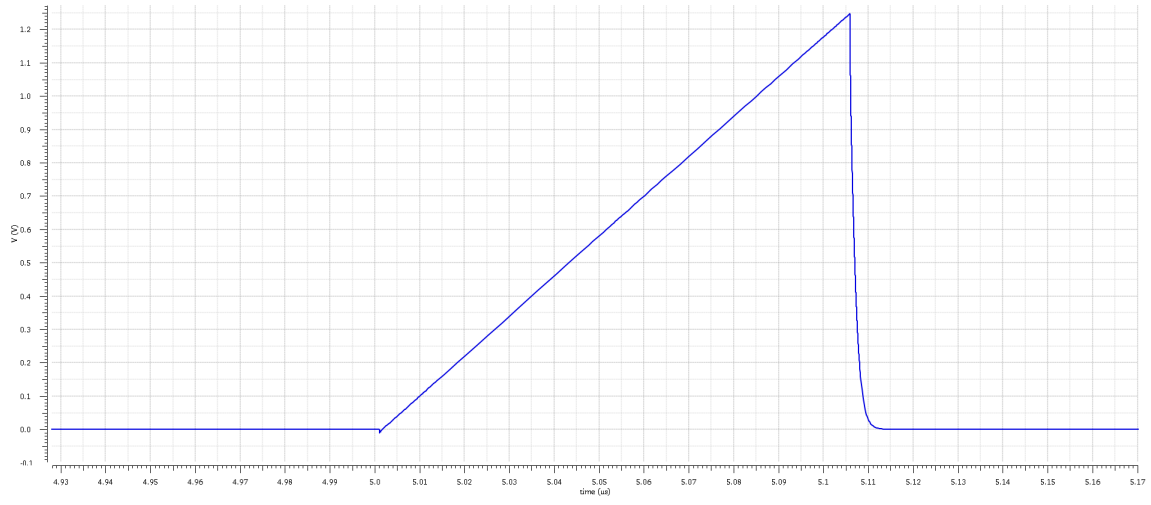


Figure 3.13: Comparator ramp input

The inverting input to the comparator is connected to a low pass filtered bandgap

voltage. Thus, when the ramp input reaches 1.2 V the comparator output a logic high. The comparator output triggers the *CLR* input on the D-flip flop to go low, forcing the output of the flip flop into a logic low state. This causes the switch controlling the current source to open and the switch in parallel with the capacitor to close, shorting the capacitor out and discharging it. The *CLR* input on the flip flop can also be triggered by a master reset provided by the *RST_L* pin on the CFD16C.

By changing the value of the programmable capacitor the charge time changes, effectively changing how long the D-flip flop will be in the logic high state (ie. it sets the width of the timing pulse). The lockout one-shot circuit works on a similar principle but has a non-programmable capacitor (250 fF) with a programmable charge current. Changing the charge current allows the capacitor to charge faster or slower, changing the lockout time. Two lockout modes are also available by writing to a mode bit as seen in Table 2.1. When Lockout Mode is a logic high, shorter ≈110 nsec time incremented are provided for a total lockout time of 3.4 μsec. When the Lockout Mode is a logic low, a long mode is used providing ≈565 nsec increments for a total lockout time of 16.6 μsec. The lockout one-shot provides an active low output as opposed to the active high output of the timing pulse one-shot.

3.3.5 Final output generation

All of the signal channel output are generated in a final output generation stage. In this stage the timing pulse, multiplicity current, global OR, and test point outputs are produced. While the timing pulse is generated in the previous stage by the one-shot circuit, it is necessary to further qualify this output before sending it off chip. Each signal channel can be individually enabled or disabled using a configuration bit, or the whole CFD16C chip can be disabled using a global enable input pin. If the global enable signal or the channel enable bit are not asserted then the timing pulse from the one shot will not be present on the output pin and will not trigger the global OR output.

A number of test point nodes from within the channel can be selected to be routed to a pin on the chip package. These available test point nodes are detailed in Table 3.4. A multiplexer at the end of each channel controls which test point is used. To prevent all sixteen channels from trying to drive the test point pin at once, a tri-state buffer with enable is used. This enable signal will only be active for the channel whose address is currently selected on the *ADDR* bus, and all other channels will have their test point outputs put into a high impedance state.

Test Point Sel	Test Point Node
000	AVSS
001	lockout pulse
010	leading edge detector pulse
011	oneshot input
100	AVSS
101	oneshot pulse
110	zero crossing detector pulse
111	AVSS

Table 3.4: Test point multiplexer outputs

The multiplicity current is produced in response to the timing pulse output firing. Once the timing pulse has fired and been qualified, a PFET switch is turned on and allows a copy of the PTAT current to be sourced to the multiplicity buffer in the common channel. All of the channel multiplicity current output are sourced to the same node so that the voltage across the resistor in the multiplicity buffer will be determined by the sum of all of the currents from the fired channels.

The global OR signal is produced by the timing pulse as well. This output is generated using a pseudo-NMOS NOR gate. As shown in Figure 3.14, in a pseudo-NMOS NOR gate a PFET acts as a pullup for parallel connected NFETs [Weste, 2006]. Thus if any one NFET is turned on then the output node will be pulled to *AVSS*. Each channel contains one of these NFETs with the PFET and a CMOS inverter being in the common channel. This allows for a very fast active high logic OR of all of the channels with

reduced complexity since the NOR gate is distributed across all channels (reducing the amount of interconnect needed).

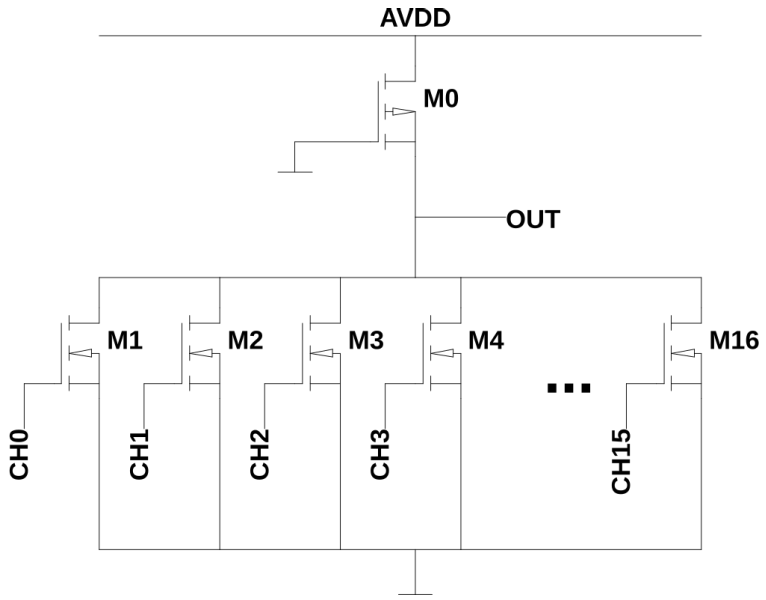


Figure 3.14: Fast pseudo-NMOS NOR

CHAPTER 4

SIMULATION RESULTS

The output one-shot provides two important features to the CFD16C: generation of the timing pulse that starts the TVCs on PSD8C, and a lockout functionality to prevent misfirings. For the lockout feature a wide spread of lockout times was desired, with a logarithmic spread. Figure 4.1 shows the spread of the lockout times available in both the short and long modes. As can be seen from the plots the spread is nearly logarithmic, with the lockout time being given by $T_{lockout} = \frac{K}{D \cdot \Delta I}$, where K is determined by $K = C \cdot V_C$ (C is capacitor in lockout one-shot), D is the digital code word given to the lockout DAC, and ΔI is the change in current from one step of the lockout DAC. The ΔI term will change depending on the lockout mode, being 380 nA in short mode and 3.8 nA in long mode.

The primary one-shot that creates the output timing pulse needs very low jitter and low variance from process and mismatch. The jitter proved to be accurately modeled by the formula $\sigma_j = \frac{\sigma_V}{VBG} \cdot T_{oneshot}$, where σ_V is the noise voltage in the current source (PFET current mirror) charging the one-shot capacitor in Figure 3.12 and V_{BG} is the bandgap voltage. For process variance and mismatch, 20 Monte Carlo simulations were run and showed acceptable variance in the trailing edge of the one-shot timing pulse (Table 4.1). This result is within two sigma, and it is expected that across all process and mismatch a variance of no more than $\pm$5 nsec in the trailing edge of the one-shot pulse is expected.

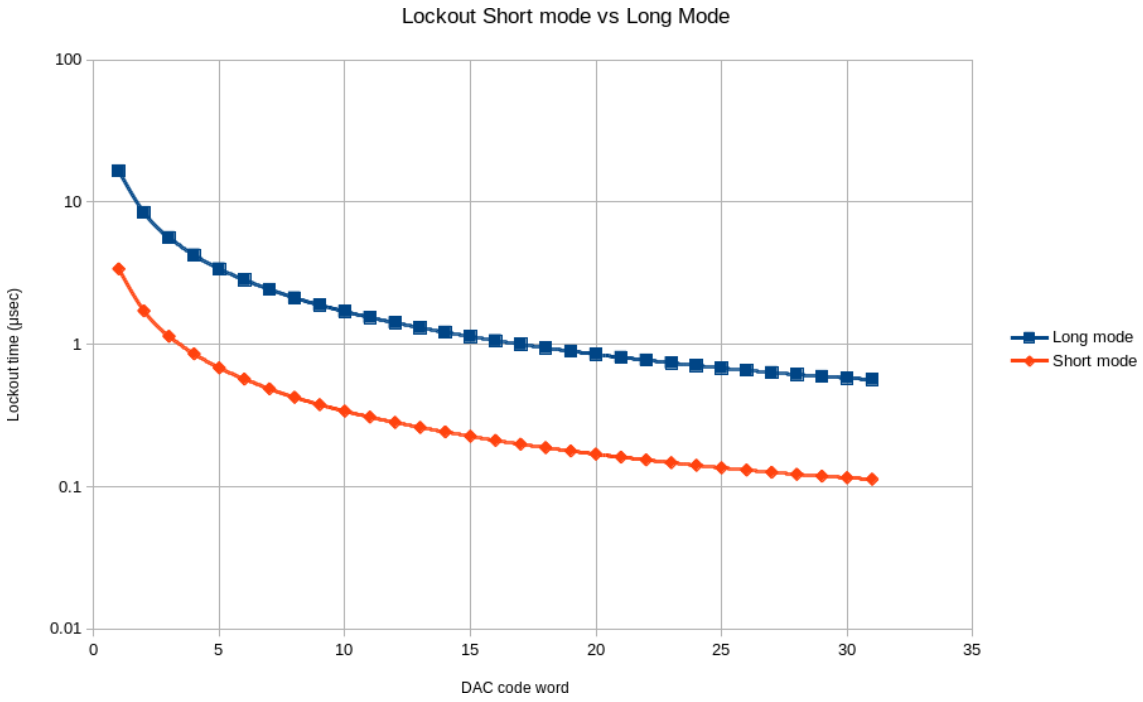


Figure 4.1: Lockout times for short and long modes

Pulse Width	Variation in Trailing edge
50 nsec	±3.8 nsec
100 nsec	±9.2 nsec
200 nsec	±18.3 nsec
500 nsec	±45 nsec

Table 4.1: One-shot pulse width variation from process and mismatch

4.5 Performance Characterization of DAC

The 6-bit bipolar DAC needed to have excellent integral non-linearity error (INL) and derivative non-linearity error (DNL) of less than 0.5 least significant bits (LSBs). Derivative non-linearity describes how much the output changes for each increment of the digital code word for the DAC [Allen, 2012]. In an ideal DAC this change would be exactly the same for each increment but in reality there are minor differences between each increment. The accumulation of all of these DNL errors over the whole range of the DAC is the INL error [Allen, 2012].

Figures 4.2–4.3 show the results of 200 Monte Carlo simulations. With each separate Monte Carlo simulation, the values of the process parameters such offset voltages, threshold voltages, etc. will be assigned differently allowing for process variation from chip to chip to be examined. As can be seen the INL and DNL errors are almost all below 0.5 LSBs with only a few outliers. It can also be seen that the worst case INL and DNL occured on simulation 150.

Figure 4.4 shows the worst case run out of all 200 simulations. It can easily be seen here than the DNL is well below 0.5 LSB for all increments of the code word except at the half way point. Examining other simulation plots show that this is the case for the other outliers as well, showing that the INL and DNL errors are in general very good and more than acceptable for this application. An average case simulation result is shown below in Figure 4.5, showing that on average the INL and DNL errors are very good.

4.6 Chip-Level Verification

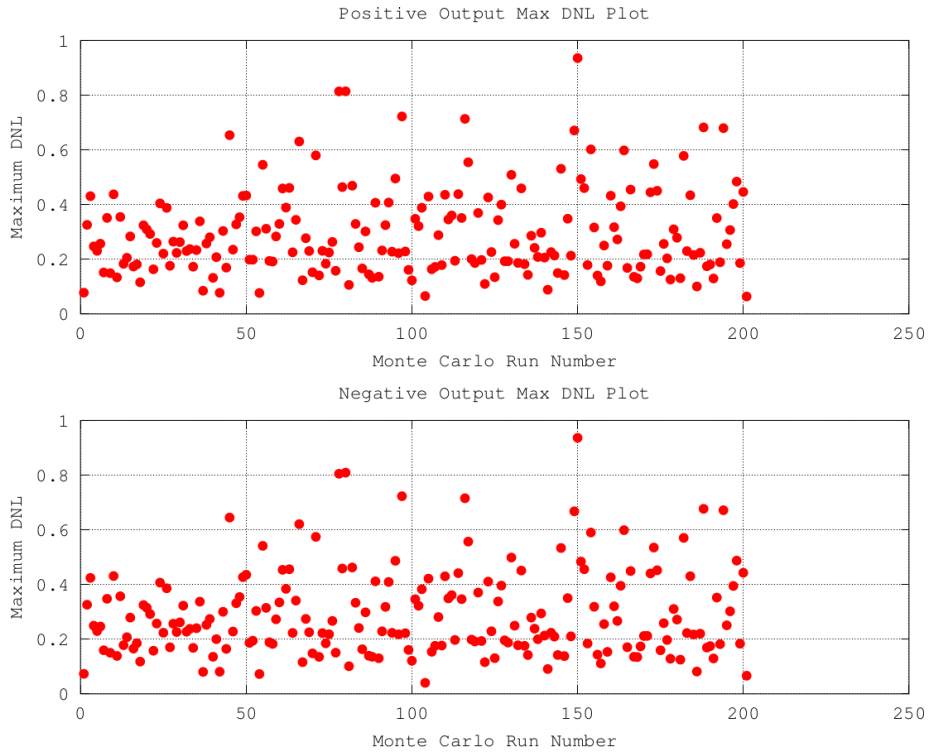


Figure 4.2: 6-bit DAC DNL error summary

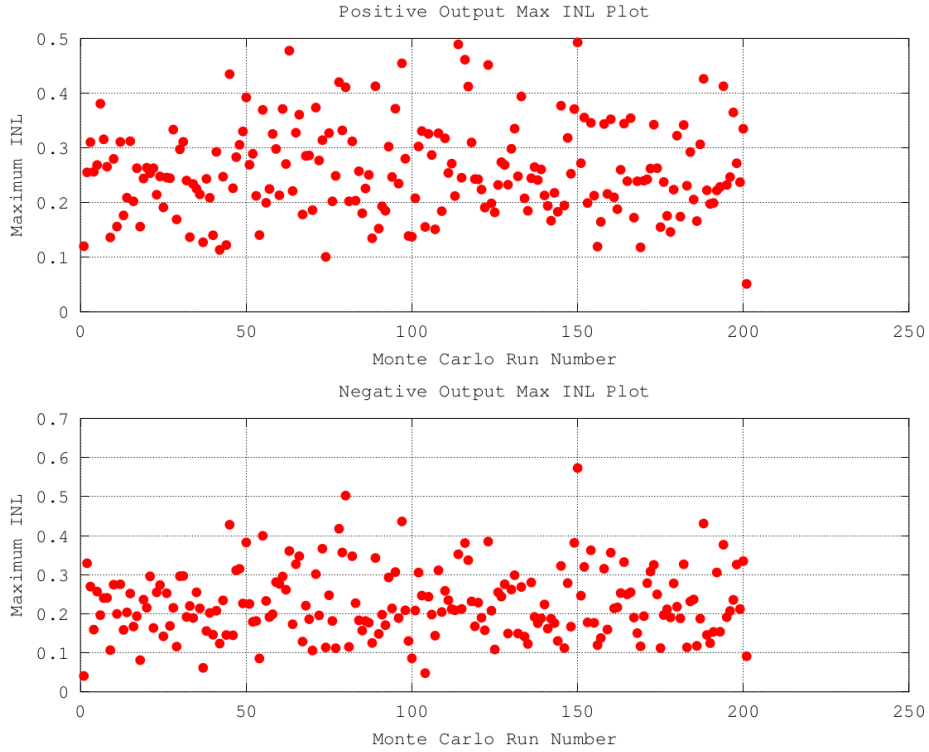


Figure 4.3: 6-bit DAC INL error summary

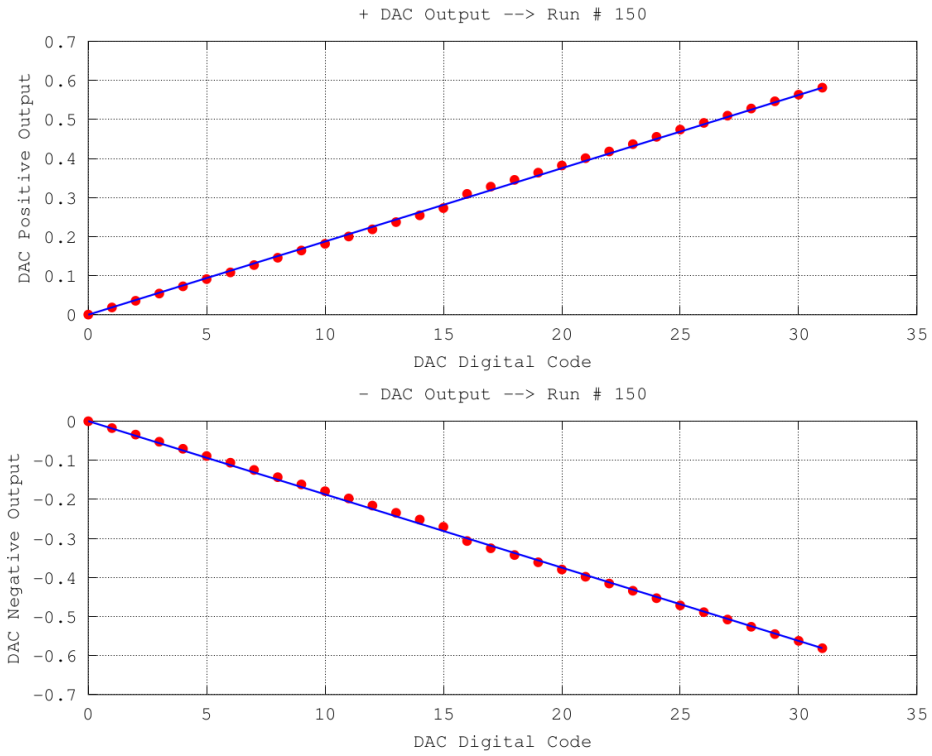


Figure 4.4: 6-bit DAC worst case error

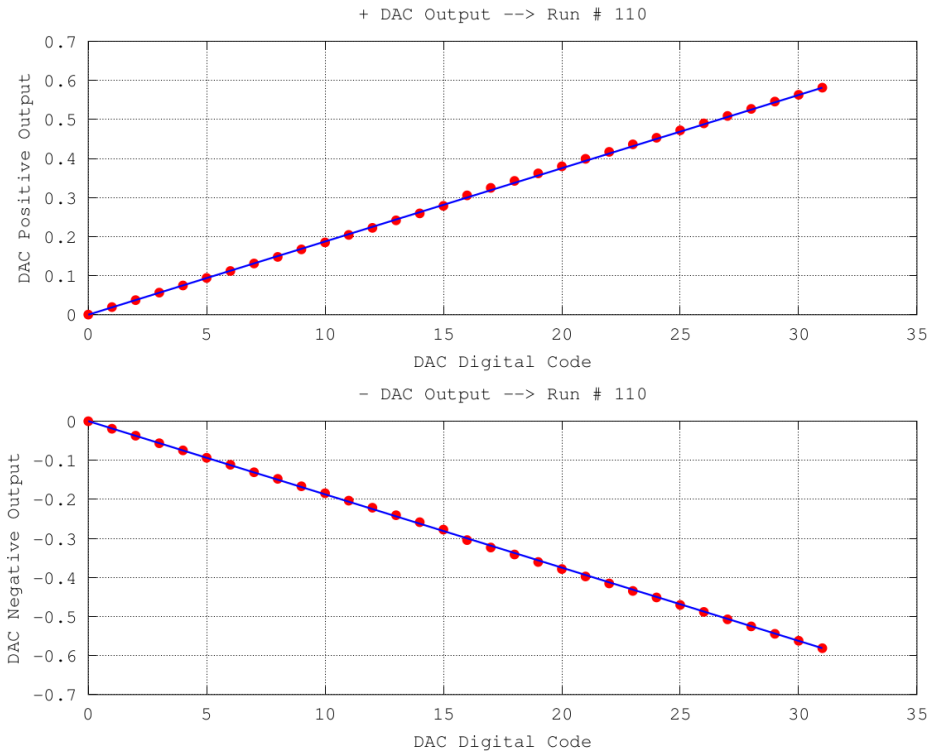


Figure 4.5: 6-bit DAC average case error

CHAPTER 5

SUMMARY, CONCLUSIONS, AND FUTURE WORK

5.1 Summary

The CFD16C was designed as a companion chip for the pulse shape discriminating IC called PSD8C. CFD16C will allow new experiments to be done with PSD8C that were previously not possible. The CFD16C provides a precise timing pulse to start time-to-voltage converters on the PSD8C. The chip is highly configurable, allowing the following circuit parameters to be adjusted to an experiments needs:

- Timing pulse width control between 50 nsec and 500 nsec
- Leading edge theshold controllable via 6-bit bi-polar DAC
- Lockout time adjustable between 100 nsec and 16.8 μsec
- Test point output providing one of five selectable digital signals
- Controllable Nowlin delay to allow input risetimes between 2 nsec and 192 nsec
- On chip signal ground generator trimmable to match supply

The CFD16C contains sixteen signal channels allowing one chip to support two PSD8Cs. Each signal channel is comprised of an input Nowlin circuit, leading edge discriminator, zero cross discriminator, and an output one-shot circuit with lockout capabilities. The CFD16C provides a timing pulse output from each of the 16 signal channels, as well as a global logical OR of all channels and an analog multiplicity voltage proportional to the number of channels that have fired. The IC occupies a die area of approximately 1.9 mm x 3.5 mm and will be fabricated in the AMS-AG 0.35 micron NWELL process.

5.2 Conclusions

The chip level simulations show that the CFD16C will work very well in new scintillator experiments accompanied with the PSD8C. The design of the IC is complete and layouts will be finished in time for a late 2018 submission.

5.3 Future Work

While the bulk of the design for CFD16C has been completed and the simulated performance looks promising, more robust and detailed simulations still need to be ran before the chip is ready to be sent out for fabrication. It is expected the chip will be sent for fabrication in November 2018. Once the chip has been fabricated a performance analysis on the physical IC will be done and any discrepancies from the simulated performance will be identified. More simulations will then be run to find the cause of these discrepancies.

REFERENCES

[A. Spyrou, 2012] A. Spyrou, Z. Kohley, T. B. (2012). First observation of ground state dineutron decay: ^{16}Be. *Physical Review Letters*.

[Allen, 2012] Allen, P. E. (2012). *CMOS analog circuit design*. Oxford University Press, USA, New York Oxford.

[Baker, 2010] Baker, R. J. (2010). *CMOS: Circuit Design, Layout, and Simulation*. Wiley, Hoboken, New Jersey, 3rd edition.

[Binkley, 1994] Binkley, D. M. (1994). Performance of non-delay-line constant-fraction discriminator timing circuits. *IEEE Transactions on Nuclear Science*, 41(4):1169–1175.

[Binkley and Casey, 1988] Binkley, D. M. and Casey, M. E. (1988). Performance of fast monolithic ecl voltage comparators in constant-fraction discriminators and other timing circuits. *IEEE Transactions on Nuclear Science*, 35(1):226–230.

[Binkley et al., 1991] Binkley, D. M., Simpson, M. L., and Rochelle, J. M. (1991). A monolithic, 2 mu;m cmos constant-fraction discriminator for moderate time resolution systems. *IEEE Transactions on Nuclear Science*, 38(6):1754–1759.

[B.S. Budden, 2015] B.S. Budden, L.C.Stonehill, A. (2015). Handheld readout electronics to fully exploit the particle discrimination capabilities of elpasolite scintillators. *Nuclear Instruments and Methods in Physics*.

[Bult and Geelen, 1992] Bult, K. and Geelen, G. (1992). An inherently linear and compact most-only current-division technique. In *1992 IEEE International Solid-State Circuits Conference Digest of Technical Papers*, pages 198–199.

[Engel, 2016] Engel, G. (2016). A multi-channel discriminator ic. `http://www.siue.edu/~gengel/ece584WebStuff/CFDchip.pdf`. accessed : 2017-1-10.

[G. Engel, 2007] G. Engel, M. Sadasivam, M. N. (2007). Multi-channel integrated circuit for use in low and intermediate energy nuclear physics - hinp16c. *Nuclear Instruments and Methods in Physics*.

[Hall, 2007] Hall, M. J. (2007). Design considerations in systems employing multiple charge integration for the detection of ionizing radiation. Master's thesis, Southern Illinois University Edwardsville.

[Hastings, 2001] Hastings, A. (2001). *The ART of ANALOG LAYOUT*. Prentice Hall, Upper Saddle River,NJ07458.

[Helmuth, 2005] Helmuth, S. (2005). *Semiconductor Detector Systems*. Oxford University Press, New York.

[I. Tilquin, 1995] I. Tilquin, Y. El Masri, M. P. (1995). Detection efficiency of the neutron modular detector demon and related characteristics. *Nuclear Instruments and Methods in Physics*.

[Jackson et al., 1996] Jackson, R. G., Blalock, T. V., Simpson, M. L., Wintenberg, A. L., and Young, G. R. (1996). Integrated constant-fraction discriminator shaping techniques for the phenix lead-scintillator calorimeter. In *1996 IEEE Nuclear Science Symposium. Conference Record*, volume 1, pages 51–55 vol.1.

[Kohley Z., 2015] Kohley Z., Baumann T., C. G. (2015). Three-body correlations in the ground-state decay of ^{26}O. *Physical Review C*.

[N. Zaitseva, 2012] N. Zaitseva, B. L. Rupert, I. P. (2012). Plastic scintillators with efficient neutron/gamma pulse shape discrimination author links open overlay panel. *Nuclear Instruments and Methods in Physics*.

[Proctor, 2007] Proctor, J. (2007). Design of a multi-channel integrated circuit for use in nuclear physics experiments where particle identification is required. Master's thesis, Southern Illinois University Edwardsville.

[Rabaey, 2003] Rabaey, J. (2003). *Digital integrated circuits : a design perspective*. Pearson Education, Upper Saddle River, N.J.

[Razavi, 2001] Razavi, B. (2001). *Design of analog CMOS integrated circuits*. McGraw-Hill, Boston, MA.

[Sadasivam, 2002] Sadasivam, M. (2002). A multi-channel integrated circuit for use with silicon strip detectors in experiments in low and intermediate physics. Master's thesis, Southern Illinois University Edwardsville.

[Simpson et al., 1994] Simpson, M. L., Britton, C. L., Wintenberg, A. L., and Young, G. R. (1994). An integrated, cmos, constant-fraction timing discriminator for multichannel detector systems. In *Nuclear Science Symposium and Medical Imaging Conference, 1994., 1994 IEEE Conference Record*, volume 1, pages 320–324 vol.1.

[Simpson et al., 1995] Simpson, M. L., Young, G. R., Jackson, R. G., and Xu, M. (1995). A monolithic, constant-fraction discriminator using distributed r-c delay line shaping. In *1995 IEEE Nuclear Science Symposium and Medical Imaging Conference Record*, volume 1, pages 292–296 vol.1.

[Singh et al., 2012] Singh, A. P., Pandey, S. K., and Kumar, M. (2012). Operational transconductance amplifier for low frequency application.

[T. Baumanna, 2005] T. Baumanna, J. Boike, J. B. (2005). Construction of a modular large-area neutron detector for the nscl. *Nuclear Instruments and Methods in Physics*.

[Tsividis, 2011] Tsividis, Y. (2011). *Operation and modeling of the MOS transistor*. Oxford University Press, New York.

[Weste, 2006] Weste, N. (2006). *CMOS VLSI design : a circuits and systems perspective*. Pearson Education, Delhi.

APPENDIX A

Verilog-A pulse generator

```
// VerilogA for Lib, pulser, veriloga
`include "constants.vams"
`include "disciplines.vams"
`define  CHANNELS 16
module pulser(TRIG, PEAK, RISE);
   output[`CHANNELS-1:0] TRIG, PEAK;
   output RISE;
   electrical[`CHANNELS-1:0] TRIG, PEAK;
   electrical gnd, RISE;
   ground gnd;
   /* Pulse peak amplitude lines */
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//PEAK0.pwl")) V_peak0(PEAK[0], gnd);
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//PEAK1.pwl")) V_peak1(PEAK[1], gnd);
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//PEAK2.pwl")) V_peak2(PEAK[2], gnd);
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//PEAK3.pwl")) V_peak3(PEAK[3], gnd);
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//PEAK4.pwl")) V_peak4(PEAK[4], gnd);
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//PEAK5.pwl")) V_peak5(PEAK[5], gnd);
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//PEAK6.pwl")) V_peak6(PEAK[6], gnd);
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//PEAK7.pwl")) V_peak7(PEAK[7], gnd);
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//PEAK8.pwl")) V_peak8(PEAK[8], gnd);
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//PEAK9.pwl")) V_peak9(PEAK[9], gnd);
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//PEAK10.pwl")) V_peak10(PEAK[10],
       gnd);
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//PEAK11.pwl")) V_peak11(PEAK[11],
       gnd);
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//PEAK12.pwl")) V_peak12(PEAK[12],
       gnd);
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//PEAK13.pwl")) V_peak13(PEAK[13],
       gnd);
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//PEAK14.pwl")) V_peak14(PEAK[14],
       gnd);
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//PEAK15.pwl")) V_peak15(PEAK[15],
       gnd);
   /* Pulse trigger lines */
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//TRIG0.pwl")) V_trig0(TRIG[0], gnd);
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//TRIG1.pwl")) V_trig1(TRIG[1], gnd);
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//TRIG2.pwl")) V_trig2(TRIG[2], gnd);
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//TRIG3.pwl")) V_trig3(TRIG[3], gnd);
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//TRIG4.pwl")) V_trig4(TRIG[4], gnd);
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//TRIG5.pwl")) V_trig5(TRIG[5], gnd);
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//TRIG6.pwl")) V_trig6(TRIG[6], gnd);
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//TRIG7.pwl")) V_trig7(TRIG[7], gnd);
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//TRIG8.pwl")) V_trig8(TRIG[8], gnd);
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//TRIG9.pwl")) V_trig9(TRIG[9], gnd);
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//TRIG10.pwl")) V_trig10(TRIG[10],
       gnd);
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//TRIG11.pwl")) V_trig11(TRIG[11],
       gnd);
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//TRIG12.pwl")) V_trig12(TRIG[12],
       gnd);
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//TRIG13.pwl")) V_trig13(TRIG[13],
       gnd);
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//TRIG14.pwl")) V_trig14(TRIG[14],
       gnd);
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//TRIG15.pwl")) V_trig15(TRIG[15],
       gnd);
```

```
    /* Risetime constant line */
    vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//RISE.pwl")) V_rise(RISE, gnd);
endmodule
```

APPENDIX B

Verilog-A Test fixture

```
// VerilogA for Lib, CFD_test, veriloga
`include "constants.vams"
`include "disciplines.vams"
`define     DATABITS 8
`define     CHANNELS 16
module Channel_Tester(IN, PEAK, TRIG, AGND, RISE, DATA, AVDD, AVSS, DVDD, DGND, GEN,
    NEG_POL, RST_L, AGND_INT_DISABLE, STB, SIG);
   output AVDD, AVSS, DVDD, DGND, RST_L, STB, GEN, AGND_INT_DISABLE;
   output[`DATABITS-1:0] DATA;
   input AGND, RISE;
   input[`CHANNELS-1:0] TRIG, PEAK, IN;
   output[`CHANNELS-1:0] SIG;
   electrical AVDD, AVSS, DVDD, DGND, RST_L, STB, GEN, AGND_INT_DISABLE, AGND, RISE;
   electrical[`DATABITS-1:0] DATA;
   electrical[`CHANNELS-1:0] TRIG, PEAK;
   electrical[`CHANNELS-1:0] SIG, IN;
   electrical gnd;
   ground gnd;
   parameter real time_tol = 100p from[1p:100n]; //time tolerance for transitions and
       timers
   parameter real load_cap = 10p; //load capacitance on the DOUT lines
   real vpeak[`CHANNELS-1:0]; //peak amplitude of exp pulse
   real exp_pulse[`CHANNELS-1:0]; //variable to hold current value of pulse
   real t0[`CHANNELS-1:0]; //time the last pulse started
   real t; //current time (used for calculating value of pulse)
   genvar j;
   real time_fall, time_rise;
   real vth;
   real tau_r;
   integer time_flag;
   integer fid;
   real start[`CHANNELS-1:0];
   /* Supply voltage lines */
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//AVDD.pwl")) V_avdd(AVDD, gnd);
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//AVSS.pwl")) V_avss(AVSS, gnd);
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//DVDD.pwl")) V_dvdd(DVDD, gnd);
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//DGND.pwl")) V_dgnd(DGND, gnd);
   /* Data lines  */
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//DATA0.pwl")) V_data0(DATA[0], gnd);
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//DATA1.pwl")) V_data1(DATA[1], gnd);
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//DATA2.pwl")) V_data2(DATA[2], gnd);
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//DATA3.pwl")) V_data3(DATA[3], gnd);
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//DATA4.pwl")) V_data4(DATA[4], gnd);
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//DATA5.pwl")) V_data5(DATA[5], gnd);
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//DATA6.pwl")) V_data6(DATA[6], gnd);
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//DATA7.pwl")) V_data7(DATA[7], gnd);
   /* Control Lines  */
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//STB.pwl")) V_stb(STB, gnd);
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//GEN.pwl")) V_gen(GEN, gnd);
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//AGND_INT_DISABLE.pwl"))
       V_agnd_int_disable(AGND_INT_DISABLE, gnd);
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//NEG_POL.pwl")) V_neg_pol(NEG_POL,
       gnd);
   vsource #(.type("pwl"), .file("~/cds/CFDtest/vcd//RST_L.pwl")) V_rst_l(RST_L, gnd);
   analog begin
      @(initial_step) begin
```

```
            time_flag = 0;
            fid = $fopen("~/cds/CFD/walk_results.csv");
            tau_r = 3e-9;
        end
        vth = (V(DVDD) + V(AVSS))/2.0;
        t = $abstime;
        for(j = 0; j < `CHANNELS; j = j + 1) begin
            @(cross(V(TRIG[j]) - vth, 1, time_tol)) begin
                vpeak[j] = V(PEAK[j]);
                tau_r = V(RISE);
                t0[j] = $abstime;
                time_flag = 1;
                time_fall = $abstime + 20*tau_r;
                start[j] = t0[j]; //record pulse time when trigger line goes high
                if(j == 0) $fwrite(fid, "\n\n");
            end
            exp_pulse[j] = exp(-(t-t0[j])/(10.0*tau_r)) - exp(-(t-t0[j])/tau_r);
            V(SIG[j], AGND) <+ vpeak[j]*exp_pulse[j]*1.435;
            @(cross(V(IN[j])-vth, 1, time_tol)) begin
                $fstrobe(fid, "channel\t%d\tamp\t%g\tstart\t%g\tout\t%g", j, vpeak[j], start
                    [j], $abstime); //record peak amplitude and output time when channel
                    output goes high.
            end
        end
        for(j = 0; j < `CHANNELS; j = j + 1) begin
            I(IN[j]) <+ load_cap*ddt(V(IN[j]));
        end
        /* After 20 time constants, clear the time flag */
        @(timer(time_fall)) begin
            time_flag = 0;
        end
        /* While time flag is set, steps will be smaller to accurately model pulse */
        if(time_flag) begin
            $bound_step(tau_r/25.0);
        end
        @(final_step) begin
            $fclose(fid);
        end
    end
endmodule
```

APPENDIX C

System Verilog global defines

```
/* Pulse characteristics   */
localparam PW = 100ns;
localparam TRF = 1ns;
localparam TPD = 5ns;
localparam STBPERIOD = 1us; // time between stb pulses when creating pulse train
/* Bit widths  */
localparam CVBITS = 2;
localparam LOCKOUTBITS = 5;
localparam ADDRBITS = 4;
localparam DATABITS = 8;
localparam MODEBITS = 4;
localparam SBITS = 4;
localparam LEBITS = 6;
localparam TRIMBITS = 3;
localparam TPBITS = 3;
localparam CHANNELS = 16;
/* Mode values    */
localparam PCAPMODE = 3'h0; //used for programmable capacitor
localparam TPMODE = 3'h0; //used for test point
localparam ONESHOTMODE = 3'h1; //used for oneshot
localparam TRIMMODE = 3'h1; //used for AGND trim
localparam LOCKOUTMODE = 3'h5; //used for lockout DAC
localparam DACMODE = 3'h6; //used for LE DAC
localparam COMMON = 4'h0; //address for common area devices
/* TP MUX selectors  */
localparam TPAVSS = 3'h0;
localparam TPLOCK = 3'h1;
localparam TPLE   = 3'h2;
localparam TPOSIN = 3'h3;
localparam TPOSOUT = 3'h5;
localparam TPZC = 3'h6;
/* Simulation parameters   */
localparam EPSILON = 1e-15;
localparam SIMDELAY = 225us;
/* Event schedule */
localparam POWER = 5ns;
localparam CONTROL_SIGS = 1us;
localparam CONFIG_ALL = 5us;
localparam GEN_ON = 6ms;
localparam HIT_ALL = 9ms;
localparam TEST_LOCKOUT = 9001us;
localparam HIT_ALL_100us = 11ms;
localparam CONFIG_LONG = 12ms;
localparam GEN_OFF = 12ms;
localparam GEN_ON2 = 12.5ms;
localparam HIT_LONG = 13ms;
localparam GEN_OFF2 = 14ms;
localparam NEG_TEST = 15ms;
```

APPENDIX D

System Verilog tasks

```
//
    ##############################################################################################

//
// Initialization task
//
//
    ##############################################################################################

task init;
   integer i;
   AVDD = 0;   //Analog supply
   AVSS = 0;   //Analog reference
   DVDD = 0;   //Digital supply
   DGND = 0;   //Digital reference
   GEN = 0; //Global enable
   NEG_POL = 0; //Negative polarity enable
   AGND_INT_DISABLE = 0; //Internal AGND generator enable
   RST_L = 0; //Low active RST
   STB  = 0; //Data/addr/mode latch signal
   RISE = 0; //Rise time constant line
   /* Set data bus low */
   for(i = 0; i < DATABITS; i = i + 1) begin
      DATA[i] = 0;
   end
   /* Set pulse trigger and peak lines low */
   for(i = 0; i < CHANNELS; i = i + 1) begin
      TRIG[i] = 0;
      PEAK[i] = 0;
   end
endtask
//
    ##############################################################################################

//
// Global Enable task
//
//
    ##############################################################################################

task gen(input state);
   if(state > 1 || state < 0)
      state = 0;
   GEN = state;
endtask
//
    ##############################################################################################

//
// Power on task
//
//
    ##############################################################################################

task power_on;
   #(TRF)
```

```
   AVDD = 3.3;
   DVDD = 3.3;
endtask
//
    ##########################################################################################

//b
// Set ADDR and MODE on the shared 8-bit bus
//
//
    ##########################################################################################

task set_addr_mode(input[ADDRBITS-1:0]addr, input[MODEBITS-1:0] mode);
   #(TRF)
   DATA[7:4] = addr;
   DATA[3:0] = mode;
endtask
//
    ##########################################################################################

//
// Set the One Shot lockout enable
//
//
    ##########################################################################################

task set_lockout_en(input data);
   if(data > 1 || data < 0)
      data = 0;
   #(TRF)
   DATA[5] = data;
endtask
//
    ##########################################################################################

//
// Set the lockout mode (0 long; 1 short)
//
//
    ##########################################################################################

task set_lockout_mode(input data);
   if(data > 1 || data < 0)
      data = 0;
   #(TRF)
   DATA[5] = data;
endtask
//
    ##########################################################################################

//
// Set the One Shot lockout pulse width control voltage DAC.
//
//
    ##########################################################################################

task set_lockout_cv(input[LOCKOUTBITS-1:0] data);
   #(TRF)
   DATA[4:0] = data;
endtask
//
    ##########################################################################################
```

```
//
// Set the One Shot pulse width control bus
//
//
    ##############################################################################################

task set_os_width(input[CVBITS-1:0] data);
   #(TRF)
   DATA[1:0] = data;
endtask
//
    ##############################################################################################

//
// Set the Leading Edge descriminator threshold DAC
//
//
    ##############################################################################################

task set_le_dac(input[LEBITS-1:0] data);
   #(TRF)
   DATA[LEBITS-1:0] = data;
endtask
//
    ##############################################################################################

//
// Set the AGND trim bits
//
//
    ##############################################################################################

task set_trim(input[TRIMBITS-1:0] data);
   #(TRF)
   DATA[4:2] = data;
endtask
//
    ##############################################################################################

//
// Set the AGND_INT_DISABLE line (internal AGND generator disable)
//
//
    ##############################################################################################

task set_agnd_int_disable(input val);
   if(val > 1 || val < 0)
      val = 0;
   #(TRF)
   AGND_INT_DISABLE = val;
endtask
//
    ##############################################################################################

//
// Set the channel enable bit for a given channel
//
//
    ##############################################################################################

task set_channel_enable(input val);
   if(val > 1 || val < 0)
      val = 0;
```

```
   #(TRF)
   DATA[6] = val;
endtask
//
    ##########################################################################################

//
// Set the test point select mux
//
//
    ##########################################################################################

task set_tp_mux(input[TPBITS-1:0] data);
   #(TRF)
   DATA[6:4] = data;
endtask
//
    ##########################################################################################

//
// Set the programmable capacitor bus
//
//
    ##########################################################################################

task set_prog_cap(input[SBITS-1:0] data);
   #(TRF)
   DATA[SBITS-1:0] = data;
endtask
//
    ##########################################################################################

//
// Set the Nowlin Circuit mode (1 = fast, 0 = slow)
//
//
    ##########################################################################################

task set_nowlin_mode(input data);
   if(data > 1 || data < 0)
      data = 0;
   #(TRF)
   DATA[7] = data;
endtask
//
    ##########################################################################################

//
// Set the negative polarity bit
//
//
    ##########################################################################################

task set_neg_pol(input val);
   if(val > 1 || val < 0)
      val = 0;
   #(TRF)
   NEG_POL = val;
endtask
//
    ##########################################################################################

//
```

```
// Set RST_L line
//
//
    ##########################################################################################

task set_rst_l(input val);
   if(val > 1 || val < 0)
      val = 0;
   #(TRF)
   RST_L = val;
endtask
//
    ##########################################################################################

//
// Produces n clock cycles on the STB line
//
//
    ##########################################################################################

task pulse_stb(integer npulses);
   integer i;
   for(i = 0; i < npulses; i = i + 1) begin
      #(STBPERIOD/2.0)
      STB = 1;
      #(STBPERIOD/2.0)
      STB = 0;
   end
endtask
//
    ##########################################################################################

//
// Set STB line either high or low
//
//
    ##########################################################################################

task set_stb(input val);
   if(val > 1 || val < 0)
      val = 0;
   #(TRF)
   STB = val;
endtask
//
    ##########################################################################################

//
// Sets the exponential pulse peak line, and creates a trigger pulse to begin the output
//
//
    ##########################################################################################

task trigger_pulse(real level[CHANNELS-1:0], real delay[CHANNELS-1:0], real rise);
   integer i;
   begin
      fork
         begin
            RISE = EPSILON;
            #(TRF)
            RISE = rise;
            #(PW*2)
            RISE = rise + EPSILON;
```

```
   #(TRF)
   RISE = 0;
end
begin
   /* Create exp pulse peak voltage pulse */
   for(i = 0; i < CHANNELS; i = i + 1) begin
      PEAK[i] = EPSILON;
   end
   #(TRF)
   for(i = 0; i < CHANNELS; i = i + 1) begin
      PEAK[i] = level[i];
   end
   #(PW)
   for(i = 0; i < CHANNELS; i = i + 1) begin
      PEAK[i] = level[i] + EPSILON;
   end
   #(TRF)
   for(i = 0; i < CHANNELS; i = i + 1) begin
      PEAK[i] = 0;
   end
end
/* Trigger an exp pulse on each channel if a peak voltage is set */
begin
   if(level[0] != 0) begin
   #(delay[0]) TRIG[0] = 1;
   #(PW)      TRIG[0] = 0;
   end
end
begin
   if(level[1] != 0) begin
   #(delay[1]) TRIG[1] = 1;
   #(PW)      TRIG[1] = 0;
   end
end
begin
   if(level[2] != 0) begin
   #(delay[2]) TRIG[2] = 1;
   #(PW)      TRIG[2] = 0;
   end
end
begin
   if(level[3] != 0) begin
   #(delay[3]) TRIG[3] = 1;
   #(PW)      TRIG[3] = 0;
   end
end
begin
   if(level[4] != 0) begin
   #(delay[4]) TRIG[4] = 1;
   #(PW)      TRIG[4] = 0;
   end
end
begin
   if(level[5] != 0) begin
   #(delay[5]) TRIG[5] = 1;
   #(PW)      TRIG[5] = 0;
   end
end
begin
   if(level[6] != 0) begin
   #(delay[6]) TRIG[6] = 1;
   #(PW)      TRIG[6] = 0;
   end
```

```
            end
            begin
               if(level[7] != 0) begin
               #(delay[7]) TRIG[7] = 1;
               #(PW)       TRIG[7] = 0;
               end
            end
            begin
               if(level[8] != 0) begin
               #(delay[8]) TRIG[8] = 1;
               #(PW)       TRIG[8] = 0;
               end
            end
            begin
               if(level[9] != 0) begin
               #(delay[9]) TRIG[9] = 1;
               #(PW)       TRIG[9] = 0;
               end
            end
            begin
               if(level[10] != 0) begin
               #(delay[10]) TRIG[10] = 1;
               #(PW)       TRIG[10] = 0;
               end
            end
            begin
               if(level[11] != 0) begin
               #(delay[11]) TRIG[11] = 1;
               #(PW)       TRIG[11] = 0;
               end
            end
            begin
               if(level[12] != 0) begin
               #(delay[12]) TRIG[12] = 1;
               #(PW)       TRIG[12] = 0;
               end
            end
            begin
               if(level[13] != 0) begin
               #(delay[13]) TRIG[13] = 1;
               #(PW)       TRIG[13] = 0;
               end
            end
            begin
               if(level[14] != 0) begin
               #(delay[14]) TRIG[14] = 1;
               #(PW)       TRIG[14] = 0;
               end
            end
            begin
               if(level[15] != 0) begin
               #(delay[15]) TRIG[15] = 1;
               #(PW)       TRIG[15] = 0;
               end
            end
         join
      end
   endtask
```

APPENDIX E

System Verilog test fixture

```
`include "localparams.vh"
`define  NEGATIVE
module verilog_driver(
   output reg [DATABITS-1:0] DATA,
   output reg [CHANNELS-1:0] TRIG,
   output reg NEG_POL,
   output reg AGND_INT_DISABLE,
   output reg RST_L,
   output reg STB,
   output reg GEN,
   output real AVDD,
   output real AVSS,
   output real DVDD,
   output real DGND,
   output real PEAK[CHANNELS-1:0],
   output real RISE
   );
   `include "verilog_driver_tasks.sv"
   integer i;
   reg gmode;
   real level[CHANNELS-1:0];
   real delay[CHANNELS-1:0];
   initial begin
      fork
         gmode = 1'b1;
         /* Initialize the chip */
         init;
         /* Apply power (AVDD, DVDD) */
         #(POWER) begin fork
            power_on;
         join end
         /* Set pin mapped control lines */
         #(CONTROL_SIGS) begin fork
            set_rst_l(1);
            set_neg_pol(0);
            set_agnd_int_disable(0);
         join end
         /* Cofigure all channels identically */
         #(CONFIG_ALL) begin fork
            set_addr_mode(4'h0, {gmode, PCAPMODE}); //system verilog for some reason
                does not set DATA correctly without this
            #10ns
            set_addr_mode(4'h0, {gmode, PCAPMODE}); //change addr to 0 and mode to
                select programmable capacitor register
            #150ns
            set_stb(1); //register addr/mode
            #300ns
            set_prog_cap(4'h1); //set programmable capacitor for 2ns rise
            #310ns
            set_tp_mux(TPLOCK); //set testpoint to see lockout pulse
            #320ns
            set_nowlin_mode(1); //set to short mode
            #450ns
            set_stb(0); //register data
            #600ns
            set_addr_mode(4'h0, {gmode, ONESHOTMODE}); //set mode to select oneshot
```

```
        register
    #750ns
    set_stb(1);
    #900ns
    set_trim(3'h4); //set AGND trim to nominal
    #910ns
    set_lockout_mode(1); //short mode
    #920ns
    set_os_width(2'h2); //set oneshot width to 500 ns
    #1.05us
    set_stb(0);
    #1.2us
    set_addr_mode(4'h0, {gmode, LOCKOUTMODE}); //set mode to select lockout dac
        register
    #1.35us
    set_stb(1);
    #1.5us
    set_lockout_cv(5'h01); //set lockout to 3.4 uS
    #1.51us
    set_lockout_en(0); //enable lockout
    #1.65us
    set_stb(0);
    #1.8us
    set_addr_mode(4'h0, {gmode, DACMODE}); //set mode to select threshold DAC
    #1.95us
    set_stb(1);
    #2.1us
    set_le_dac(6'h2f); //set threshold value
    #2.11us
    set_channel_enable(1); //enable channel
    #2.25us
    set_stb(0);
    #2.5us
    set_addr_mode(4'h0, 4'h0); //set address for test point
join end
/* Set global enable */
#(GEN_ON) begin
    gen(1);
end
#(HIT_ALL)
begin
    /* Trigger the exp pules */
        for(i = 0; i < CHANNELS; i = i + 1) begin
            delay[i] = 100ns;
            level[i] = (i == 0) ? 0.015:level[i-1]*1.36;
        end
        trigger_pulse(level, delay, 2e-9);
end
/* Fire shortly after HIT_ALL to test lockout */
#(TEST_LOCKOUT)
begin
    /* Trigger the exp pules */
        for(i = 0; i < CHANNELS; i = i + 1) begin
            delay[i] = 100ns;
            level[i] = 0.15;
        end
        trigger_pulse(level, delay, 2e-9);
end
/* Hit all of the channels with the same pulses, and hit them again
    100us later to test how quick DC offset recovery is */
#(HIT_ALL_100us) begin fork
    /* Trigger the exp pules */
        for(i = 0; i < CHANNELS; i = i + 1) begin
```

```
            delay[i] = 100ns;
            level[i] = (i == 0) ? 0.015:level[i-1]*1.36;
        end
        trigger_pulse(level, delay, 2e-9);
        #100us
        trigger_pulse(level, delay, 2e-9);
join end
#(GEN_OFF) begin
    gen(0);
end
/* Configure for long rise time constants */
#(CONFIG_LONG) begin fork
    set_addr_mode(4'h0, {gmode, PCAPMODE});
    #10ns
    set_addr_mode(4'h0, {gmode, PCAPMODE});
    #150ns
    set_stb(1);
    #300ns
    set_prog_cap(4'hf); //set PCAP at maximum value (192 ns)
    #310ns
    set_tp_mux(TPLOCK);
    #320ns
    set_nowlin_mode(0); //set to long mode
    #450ns
    set_stb(0);
    #600ns
    set_addr_mode(4'h0, {gmode, ONESHOTMODE});
    #750ns
    set_stb(1);
    #900ns
    set_trim(3'h4);
    #910ns
    set_lockout_mode(0); //long mode
    #920ns
    set_os_width(2'h0);
    #1.05us
    set_stb(0);set_addr_mode(4'h0, {gmode, LOCKOUTMODE}); //set mode to select
        lockout dac register
    #1.2us
    set_stb(1);
    #1.35us
    set_lockout_cv(5'h01); //set lockout to 16.8 uS
    #1.36us
    set_lockout_en(0); //enable lockout
    #1.5us
    set_stb(0);
    #1.65us
    set_addr_mode(4'h0, 4'h0); //set address for test point
join end
#(GEN_ON2) begin
    gen(1);
end
/* Hit channels with long rise time pulses */
#(HIT_LONG) begin
    /* Trigger the exp pules */
        for(i = 0; i < CHANNELS; i = i + 1) begin
            delay[i] = 100ns;
            level[i] = (i == 0) ? 0.015:level[i-1]*1.36;
        end
        trigger_pulse(level, delay, 192e-9);
    end
`ifdef NEGATIVE
#(GEN_OFF2) begin
```

```
   gen(0);
end
/* Configurate for negative pulses */
#(NEG_TEST + CONFIG_ALL) begin fork
   set_addr_mode(4'h0, {gmode, PCAPMODE}); //system verilog for some reason
       does not set DATA correctly without this
   #10ns
   set_addr_mode(4'h0, {gmode, PCAPMODE}); //change addr to 0 and mode to
       select programmable capacitor register
   #150ns
   set_stb(1); //register addr/mode
   #300ns
   set_prog_cap(4'h1); //set programmable capacitor for 2ns rise
   #310ns
   set_tp_mux(TPLOCK); //set testpoint to see lockout pulse
   #320ns
   set_nowlin_mode(1); //set to fast mode
   #450ns
   set_stb(0); //register data
   #600ns
   set_addr_mode(4'h0, {gmode, DACMODE}); //set mode to select threshold DAC
   #750ns
   set_le_dac(6'h04); //set threshold value
   #760ns
   set_channel_enable(1); //enable channel
   #900ns
   set_stb(0);
   #1us
   set_addr_mode(4'h0, 4'h0); //set address for test point
join end
/* Enable control lines */
#(NEG_TEST + 100us) begin fork
   gen(1);
   set_neg_pol(1);
join end
/* Hit with negative pulses 16ns rise */
#(NEG_TEST + 500us) begin
   /* Trigger the exp pules */
      for(i = 0; i < CHANNELS; i = i + 1) begin
         delay[i] = 100ns;
         level[i] = (i == 0) ? -0.015:level[i-1]*1.36;
      end
      trigger_pulse(level, delay, 2e-9);
end
/* Test lockout with negative pulses */
#(NEG_TEST + 501us) begin
   /* Trigger the exp pules */
      for(i = 0; i < CHANNELS; i = i + 1) begin
         delay[i] = 100ns;
         level[i] = -0.15;
      end
      trigger_pulse(level, delay, 2e-9);
end
/* Hit all of the channels with the same negative pulses, and hit them again
   100us later to test how quick DC offset recovery is */
#(NEG_TEST + 2.5ms) begin fork
   /* Trigger the exp pules */
      for(i = 0; i < CHANNELS; i = i + 1) begin
         delay[i] = 100ns;
         level[i] = (i == 0) ? -0.015:level[i-1]*1.36;
      end
      trigger_pulse(level, delay, 2e-9);
      #100us
```

```
            trigger_pulse(level, delay, 2e-9);
        join end
        #(NEG_TEST + 4.5ms) begin fork
            gen(0);
        join end
        /* Configure for long negative pulses */
        #(NEG_TEST + 6.5ms) begin fork
            set_addr_mode(4'h0, {gmode, PCAPMODE});
            #10ns
            set_addr_mode(4'h0, {gmode, PCAPMODE});
            #150ns
            set_stb(1);
            #300ns
            set_prog_cap(4'hf); //set PCAP at maximum value (192 ns)
            #310ns
            set_tp_mux(TPLOCK);
            #320ns
            set_nowlin_mode(0); //set to slow mode
            #450ns
            set_stb(0);
            #500ns
            set_addr_mode(4'h0, 4'h0); //set address for TP
        join end
        #(NEG_TEST + 5ms) begin
            gen(1);
        end
        /* Hit channels with long negative pulses */
        #(NEG_TEST + 5.5ms) begin
            /* Trigger the exp pules */
            for(i = 0; i < CHANNELS; i = i + 1) begin
                delay[i] = 100ns;
                level[i] = (i == 0) ? -0.015:level[i-1]*1.36;
            end
            trigger_pulse(level, delay, 192e-9);
        end
    `endif
    join
  end
endmodule
```

APPENDIX F

System Verilog instantiation

```
`timescale 1ns/1ps
`define NEGATIVE
`ifdef NEGATIVE
   `define LEN_OF_SIM      22ms
`else
   `define LEN_OF_SIM      13.5ms
`endif
`define     ADDRBITS    4
`define     DATABITS    8
`define     MODEBITS    4
`define     CHANNELS    16
module verilog_driver_tb;
   // Need to create a VCD (Value Change Dump) file
   initial begin
      $dumpfile("/home/CFD/cds/CFDtest/vcd/verilog_driver.vcd") ;
      $dumpvars(1, AVDD, AVSS, DVDD, DGND, NEG_POL, AGND_INT_DISABLE,
              RST_L, STB, DATA0, DATA1, DATA2, DATA3, DATA4,
              DATA5, DATA6, DATA7, PEAK0, PEAK1, PEAK2, PEAK3,
              PEAK4, PEAK5, PEAK6, PEAK7, PEAK8, PEAK9, PEAK10, PEAK11,
              PEAK12, PEAK13, PEAK14, PEAK15, TRIG0, TRIG1, TRIG2, TRIG3,
              TRIG4, TRIG5, TRIG6, TRIG7, TRIG8, TRIG9, TRIG10, TRIG11,
              TRIG12, TRIG13, TRIG14, TRIG15, GEN, RISE) ;
   end
   real AVDD, AVSS, DVDD, DGND, RISE;
   wire STB;
   wire[`DATABITS-1:0] DATA;
   real PEAK[`CHANNELS-1:0];
   wire[`CHANNELS-1:0] TRIG;
   wire DATA0, DATA1, DATA2, DATA3, DATA4, DATA5, DATA6, DATA7;
   real PEAK0, PEAK1, PEAK2, PEAK3, PEAK4, PEAK5, PEAK6, PEAK7;
   real PEAK8, PEAK9, PEAK10, PEAK11, PEAK12, PEAK13, PEAK14, PEAK15;
   wire TRIG0, TRIG1, TRIG2, TRIG3, TRIG4, TRIG5, TRIG6, TRIG7;
   wire TRIG8, TRIG9, TRIG10, TRIG11, TRIG12, TRIG13, TRIG14, TRIG15;
   wire GEN, AGND_INT_DISABLE, RST_L, NEG_POL;
   verilog_driver dut(      .AVDD(AVDD),
                .AVSS(AVSS),
               .DVDD(DVDD),
               .DGND(DGND),
               .RISE(RISE),
               .DATA(DATA),
               .PEAK(PEAK),
               .TRIG(TRIG),
               .STB(STB),
               .RST_L(RST_L),
               .NEG_POL(NEG_POL),
               .AGND_INT_DISABLE(AGND_INT_DISABLE),
               .GEN(GEN) );
   assign DATA0 = DATA[0];
   assign DATA1 = DATA[1];
   assign DATA2 = DATA[2];
   assign DATA3 = DATA[3];
   assign DATA4 = DATA[4];
   assign DATA5 = DATA[5];
   assign DATA6 = DATA[6];
   assign DATA7 = DATA[7];
   assign TRIG0 = TRIG[0];
```

```
    assign TRIG1 = TRIG[1];
    assign TRIG2 = TRIG[2];
    assign TRIG3 = TRIG[3];
    assign TRIG4 = TRIG[4];
    assign TRIG5 = TRIG[5];
    assign TRIG6 = TRIG[6];
    assign TRIG7 = TRIG[7];
    assign TRIG8 = TRIG[8];
    assign TRIG9 = TRIG[9];
    assign TRIG10 = TRIG[10];
    assign TRIG11 = TRIG[11];
    assign TRIG12 = TRIG[12];
    assign TRIG13 = TRIG[13];
    assign TRIG14 = TRIG[14];
    assign TRIG15 = TRIG[15];
    assign PEAK0 = PEAK[0];
    assign PEAK1 = PEAK[1];
    assign PEAK2 = PEAK[2];
    assign PEAK3 = PEAK[3];
    assign PEAK4 = PEAK[4];
    assign PEAK5 = PEAK[5];
    assign PEAK6 = PEAK[6];
    assign PEAK7 = PEAK[7];
    assign PEAK8 = PEAK[8];
    assign PEAK9 = PEAK[9];
    assign PEAK10 = PEAK[10];
    assign PEAK11 = PEAK[11];
    assign PEAK12 = PEAK[12];
    assign PEAK13 = PEAK[13];
    assign PEAK14 = PEAK[14];
    assign PEAK15 = PEAK[15];
    // Run simulation for a bit and then finish
    initial begin
          # (`LEN_OF_SIM)      $finish ;
    end
endmodule
```

APPENDIX G

VCD to PWL python script

```
#!/usr/bin/python
#
#  GLE: 6 June 2017
#
# Fixed several bugs!!!
# It now finds floating point numbers correctly
# Also fixed the bug tha Po discovered
#
# Python script to convert vcd file to a series of pwl files
#
#
# This script will read and parse a VCD (Value Change Dump) file
# produced by a Verilog simulation
#
# A SPICE piece-wise-linear description is created for each signal in the VCD file
#
# Modfied on September 14, 2014 to support real variables
# Need system calls
import  sys ;
# Need command line arguments from operating system
from sys import argv ;
# Need the regular expression package
import   re ;
# Set some electrical parameters
HI = 3.3 ;        # Electrical level for a logical 1
LO = 0.0 ;        # Electrical value for a logical 0
TRF = 1000  ;     # Rise/fall time (in ps) i.e. 1 ns
REAL_SCALE = 1.0; # Scale factor for the real valued signals
# Creates symbol_table
symbol_table = {} ;
#
# Create a global variable token_table
# Token is the key with the pattern as the value
#
token_table = {"DATE" : "^\$date" ,
   "VER" : "^\$version" ,
   "TIME" : "^\$timescale",
   "SCOPE" : "^\$scope",
   "DUMP"  : "^\$dumpvars" ,
   "VAR" : "^\$var",
   "END" : "^\$end",
   "UPDATE" : "^\#[\d]+",
   "VCD" : "^[01]",
   "VCDR" : "^r" } ;
# Create a function that parses a line and returns the appropriate token
def parser(line):
   global token_table ;                  # token_table is a global variable
   keys = token_table.keys() ;
   token = "NULL" ;                       # The NULL token is our default
   for key in keys :
      pattern = token_table[key] ;         # Pattern we are attempting to match
      match = re.match(pattern, line) ;
      if (match) :
         token = key ;                     # If there is a match set token equal to the
             key
   return token ;                         # Return the token!
```

```
# User is expected to provide name of vcd file to read
# Expecting exactly two command line arguments
if (len(sys.argv) == 2) :
   cmd, vcd_file_name = argv ;
   print "" ;
   print "Reading file:  %s" % vcd_file_name ;
   print "" ;
else :
   print "" ;
   print "Usage: vcd2pwl <filename.vcd>" ;
   print ""
   sys.exit ;
# Open up the file for reading
try:
   vcd_fid = open(vcd_file_name, "r") ;
except IOError:
   print "Could not open file for reading!" ;
# Read one line at a time from the vcd file
# Read in first line from file
line = vcd_fid.readline() ;
# Keep reading lines from the file until EOF reached
while line :
#
# Send line off to be parsed ... comes back with a token
#
   token = parser(line) ;
#
# If we have a timescale directive then read the next line and pick off the multiplier
#
   if (token == "TIME") :
      line = vcd_fid.readline() ;          # Read in the next line
      fields = line.split() ;           # Split the line up into fields
      value = float(fields[0]) ;           # value (first field)
      unit = fields[1] ;             # unit ... ps, ns, us etc (second field)
      if (unit == "ns") :            # Determine what our time base multiplier is
         multiplier = 1e-9  * value ;
      elif (unit == "ps") :
         multiplier = 1e-12 * value ;
      elif (unit == "us") :
         multiplier = 1e-6 * value ;
      else :
         multiplier = 1.0 * value ;
      print "Mutliplier is %g.\n" % multiplier ;
#
# If we have a var directive then pick off signal name and symbol to be used to
    represent the signal
#
   elif (token == "VAR") :
      fields = line.split() ;           # Split line up into fields
      symbol = fields[3] ;              # Symbol used to represent signal (4th field)
      signal = fields[4] ;              # Signal name (5th field)
      symbol_table[symbol] = signal ;         # Build our dictionary of symbols and
          signal names
#
# If we have a dump directive then open up a bunch of files for writing
# and then get the initial conditions
#
   elif (token == "DUMP") :
      time = 0 ;               # Set time to 0.0
      keys = symbol_table.keys() ;         # The keys are the symbols
      fid = {} ;               # Create a dictionary
      for key in keys :           # Build the dictionary
         signal_name = symbol_table[key] ;
```

```
         file_name = signal_name + ".pwl" ;
         fid[key] =  open(file_name, "w") ;  # Opening a .pwl file for each signal
#
# Keep reading lines for the DUMP state until we get the END token
# For real valued signals the line will begin with a r
#
      line = vcd_fid.readline() ;          # Read next line from the file
      while (parser(line) != "END") :
         value = line[0] ;        # First character is the value of the signal
         if (value == '0') :         # Convert to an electrical level
            voltage = LO ;
            symbol = line[1] ;       # Second character is the symbol used for the signal
         elif (value == '1') :
            voltage = HI ;
            symbol = line[1] ;       # Second character is the symbol used for the signal
         elif (value == 'r') :
            fields = line.split() ;     # Split the line up into sapce delimited fields
#           m = re.search('[\d.]+', fields[0]); # Find the float
            m = re.search('[-+]?(\d+(\.\d*)?|\.\d+)([eE][-+]?\d+)?', fields[0]); # Find
                the float
            voltage = float(m.group(0)) ; # Convert to float
            voltage *= REAL_SCALE ;
            symbol = fields[1] ;
         else :
            pass ;
         line_out = "%g %g\n" % (time, voltage) ;
         fid[symbol].write(line_out) ;     # Write out inital values at time t=0
         line = vcd_fid.readline() ;
#
# Need to compute our new time ... UPDATE state
#
   elif (token == "UPDATE") :
      time = int(line[1:len(line)]) ;        # Strip off first character which is a
          pound sign
#
# Here is what we do when a value changes and is dumped (VCD)
#
   elif (token == "VCD") :
      value = line[0] ;           # First character is the NEW value either a 0 or a 1
      symbol = line[1] ;             # Next character is the symbol
      if (value == '0') :            # Convert to an electrical level
         voltage = HI;            # Looks wrong but not
      else :                   # Need to write out old value first
         voltage = LO ;
      line_out = "%dp %g\n" % (time, voltage) ;     # time came from the UPDATE state
      fid[symbol].write(line_out) ;
#     time += TRF ;                # Increment time by a rise/fall time
      if (value == '0') :            # Compute new electrical levels
         voltage = LO;
      else :
         voltage = HI ;
      line_out = "%dp %g\n" % (time + TRF, voltage) ; # Write out "new" (time, voltage)
          pair
      fid[symbol].write(line_out) ;
   elif (token == "VCDR") :
      fields = line.split() ;           # Split the line up into sapce delimited fields
#     m = re.search('[\d.-]+', fields[0]);       # Find the float
      m = re.search('[-+]?(\d+(\.\d*)?|\.\d+)([eE][-+]?\d+)?', fields[0]);     # Find the
           float
      voltage = float(m.group(0)) ;        # Convert to float
      voltage *=  REAL_SCALE ;
      symbol = fields[1] ;
      line_out = "%dp %g\n" % (time, voltage) ;     # time came from the UPDATE state
```

```
      fid[symbol].write(line_out) ;
#
# Go read the next line from the file and go back to start of while loop
#
   line = vcd_fid.readline() ;
# ************************************************************************
#
# Close up all of the files
#
vcd_fid.close() ;
keys = symbol_table.keys() ;              # The keys are the symbols
for key in keys :
   name = symbol_table[key] ;
   name += ".pwl" ;
   print  "Successfully created: %s" % name ;
   fid[key].close() ;
print ""
```