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:

George L. Engel, D.Sc, Chair

Bradley Noble, D.Sc

Timothy York, PhD

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, D.Sc

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) and gated integrators 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. The IC will have an average power dissipation of 220 mW and occupy an area of 2.4 x 3.5 mm. The jitter in the trailing edge of the output timing pulse will be less than 5 nsec (for pulse width of 50 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 implemented in a 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 whom I've been privileged to work with on this project as well. Pohan Wang, Prarthana Jani, Sneha Edula, Anil Korkmaz, Sri Kandula, and I all worked together to make CFD16C possible. 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, (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 "toolbox" 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`).

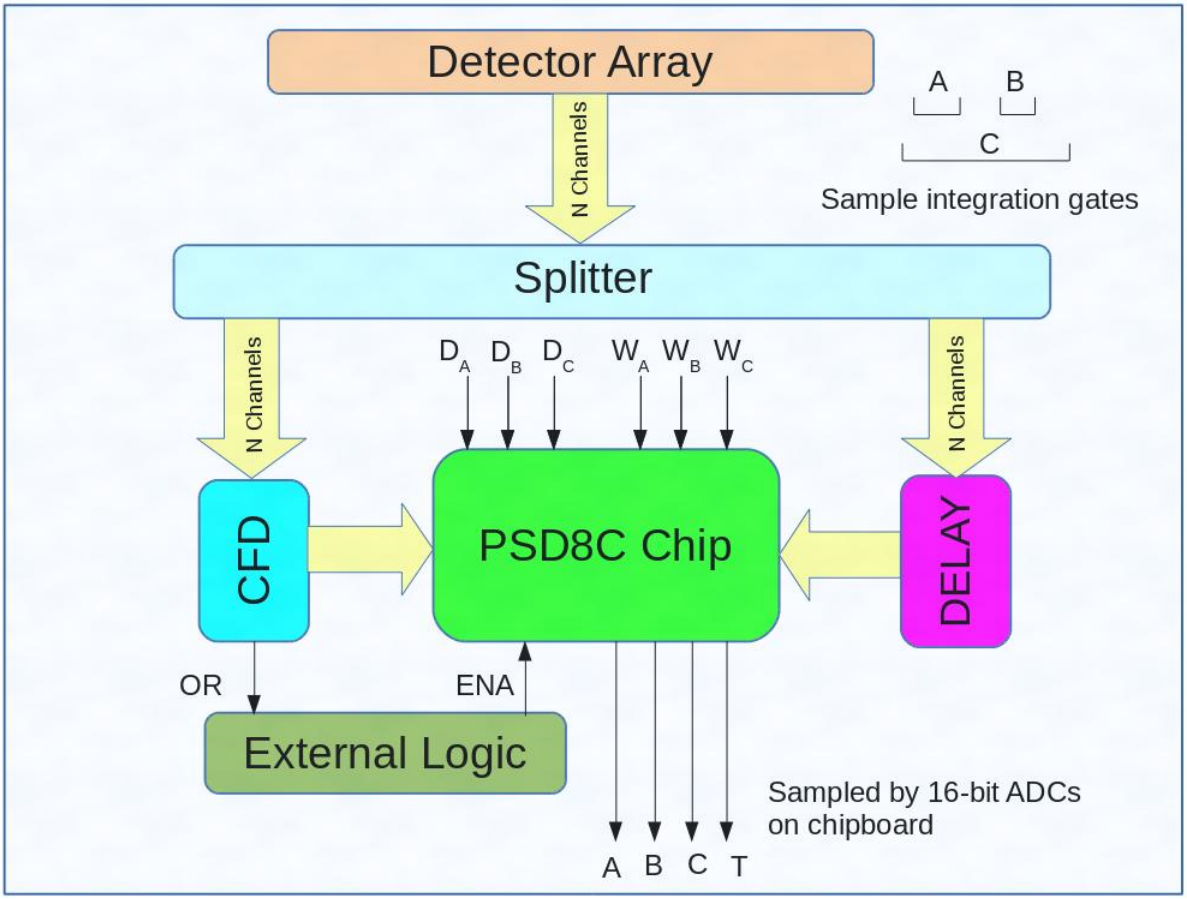


Figure 1.1: Block diagram of a typical PSD system.

Figure 1.1 shows a typical 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 associated 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. The 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;

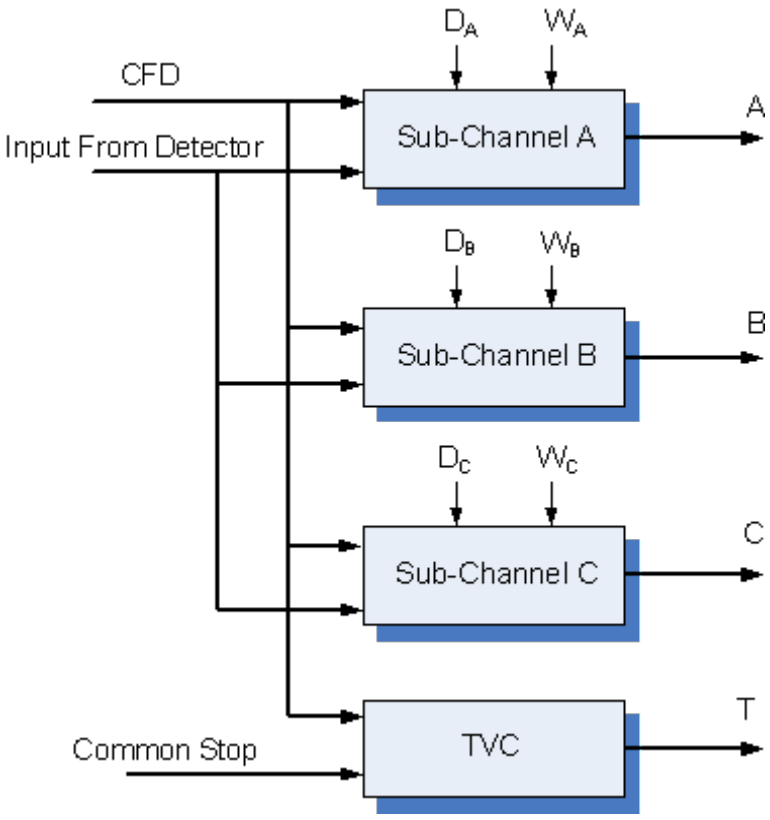
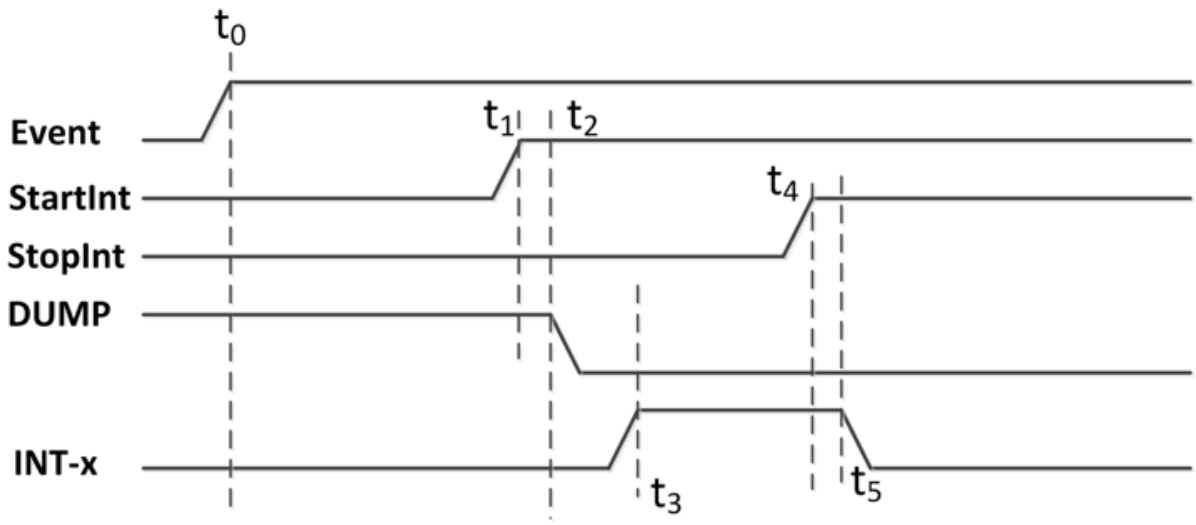


Figure 1.2: PSD Channel



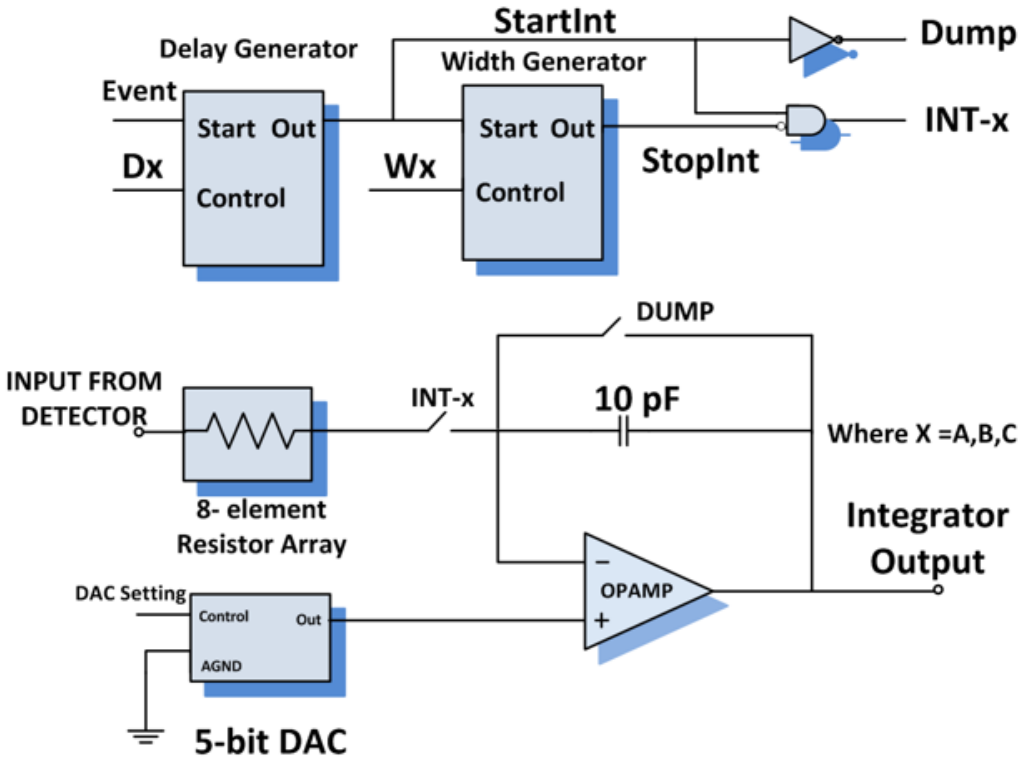


Figure 1.3: PSD Sub-channel.

- 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 (Time-to-Voltage Converter) with two time ranges: 500 nsec and 2 msec;
- 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 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 and is packaged in a 14 by 14 mm, 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 attempted to correct several minor problems. First, the TVC circuit could inadvertently be restarted. In Rev. 2, once the rising edge of the "common stop" signal is detected, the TVC cannot restart 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 was 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. These level translators convert 5 V logic level outputs into 3.3 V logic levels to be used safely by a 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. Aslo, in the latest version of the chip, the TVC circuits start on the "falling" rather than "rising" edge of the CFD input timimg pulse. This change has profound implications for the the design presented in this thesis. Most notably, the timimg pulses produced by the CFD chip described herein must possess low jitter (stocahistic timimg uncertainty) for both edges of the output pulses.

1.3 Need for an Integrated Circuit

Although 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 a multi-channel integrated circuit which can generate the timing signals for a pair of PSD chips.

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], suffered from 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

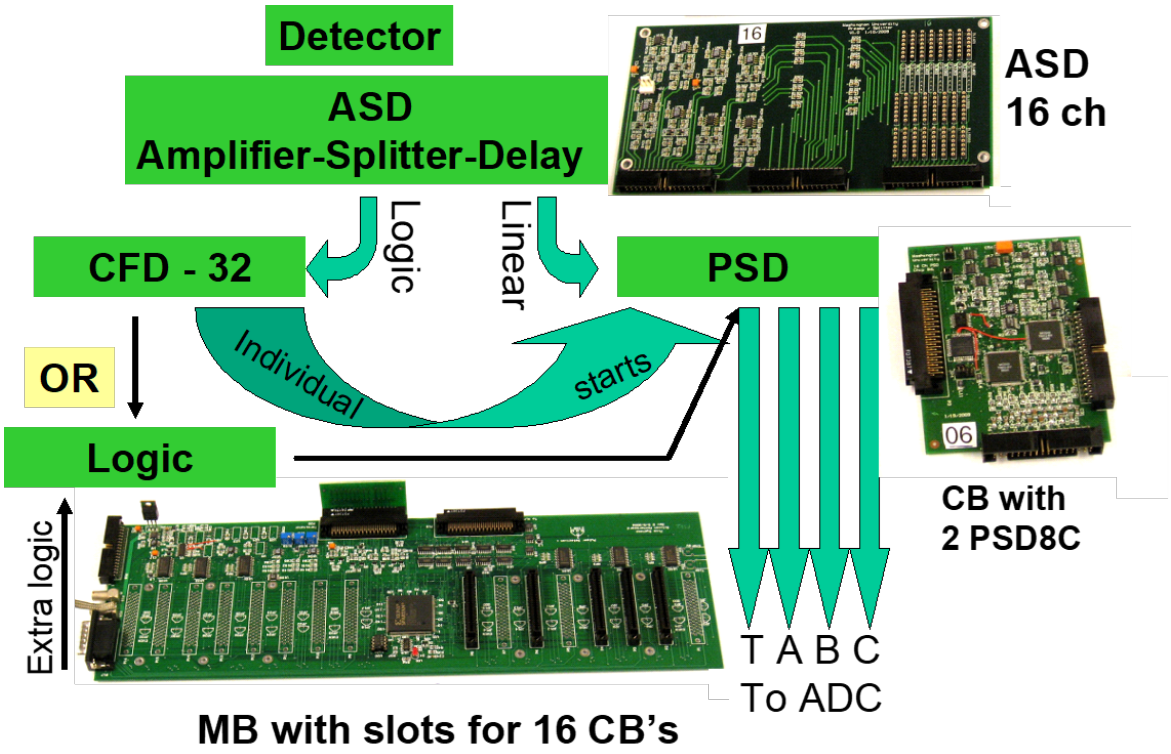


Figure 1.4: PSD system using board-level CFD electronics

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 4 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. The 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 disabled as per the needs of the user. Additionally, should it be required, 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.

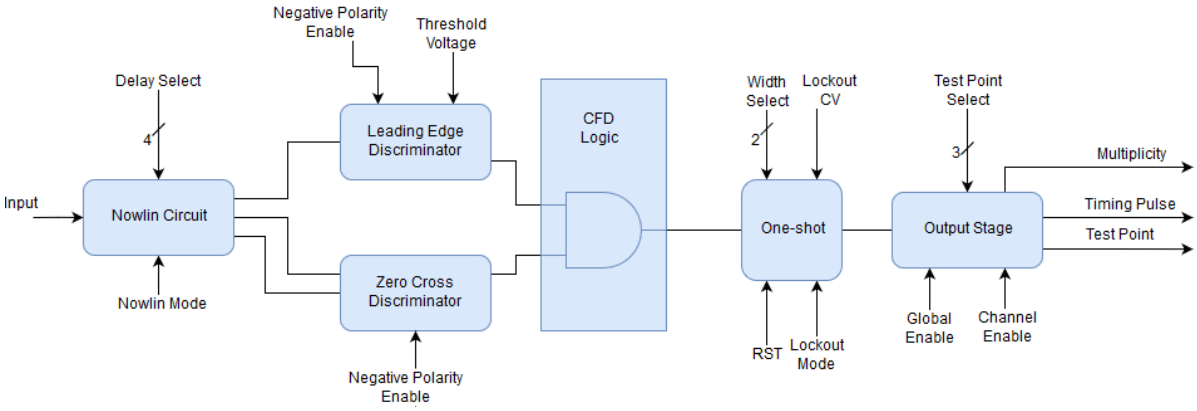


Figure 2.1: System level overview for one channel of the CFD16C

An analog input pulse will arrive at the input stage of the channel in the form of an exponential voltage pulse with a rise time approximately 10 to 100 times faster than its fall time. This stage contains a Nowlin circuit that creates a differential output and a high pass filtered output (often referred to in the literature as the "fast shaper" 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 presented in Figure 2.2 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. In order 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 mode value and the purpose of each of the bits in the configuration registers.

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

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

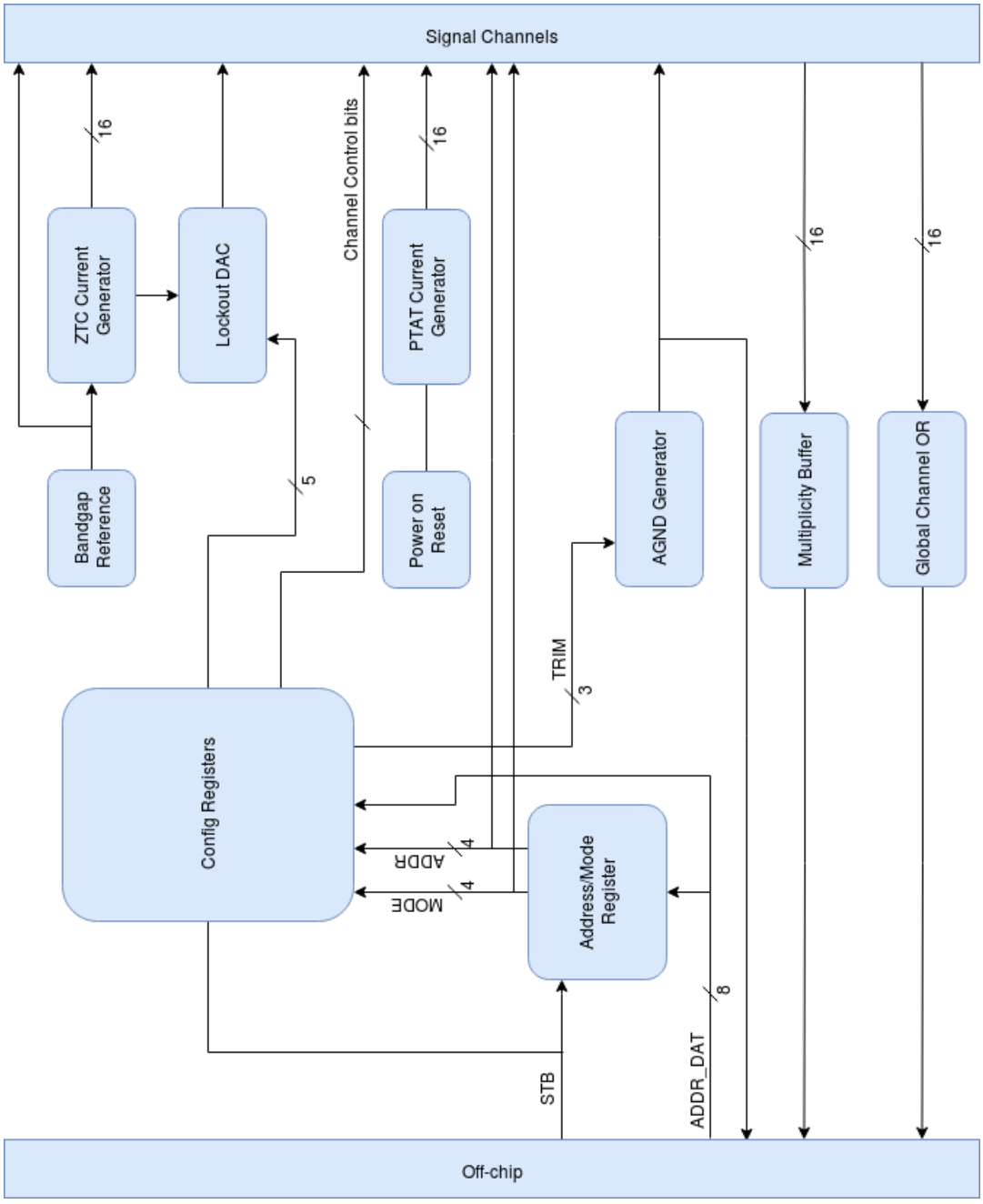


Figure 2.2: System level diagram of the common channel

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

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 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 circuit 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].

A dynamic 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.

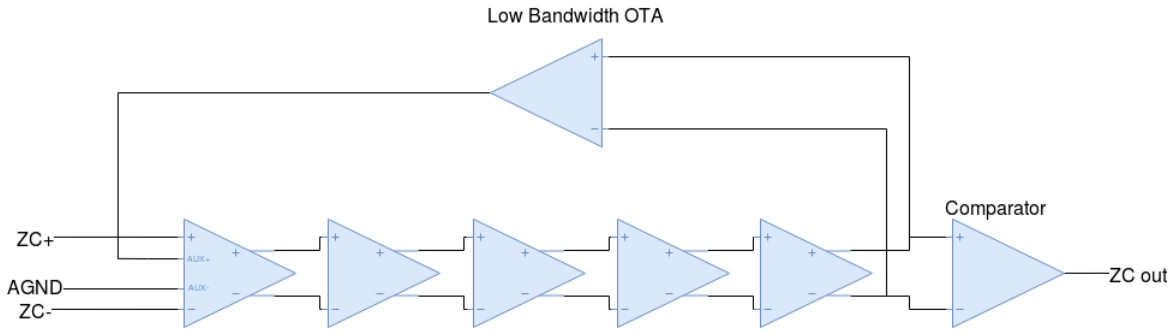


Figure 2.4: Zero cross discriminator with DC offset cancellation

The input to the leading-edge discriminator comes from the high-pass filter, or "fast shaper", in the Nowlin circuit. The leading-edge discriminator (Figure 2.5) 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 (3.3 V) to the *NEG_POL* input pin. The output of the leading edge discriminator is then used to qualify the zero-cross detector.

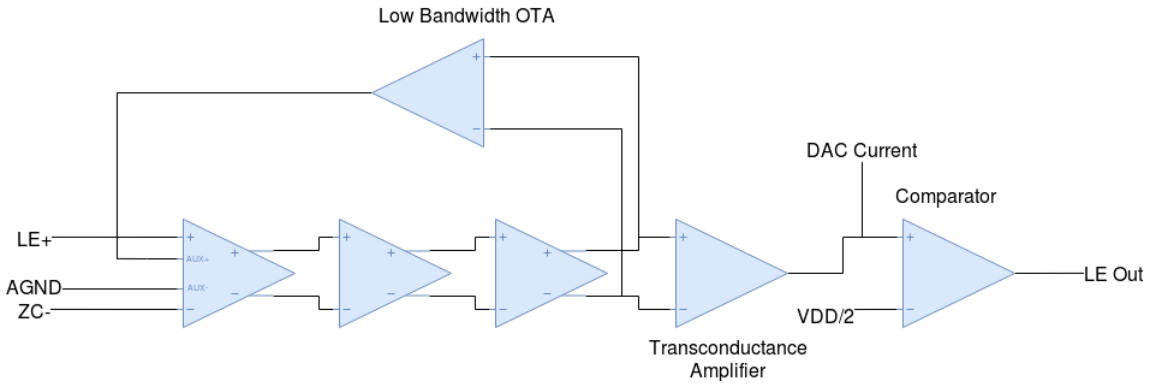


Figure 2.5: Leading edge discriminator with DC offset cancellation

The 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 one-shot circuit creates the channel's output timing pulse. It is provided with a two-bit pulse-width selection 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.

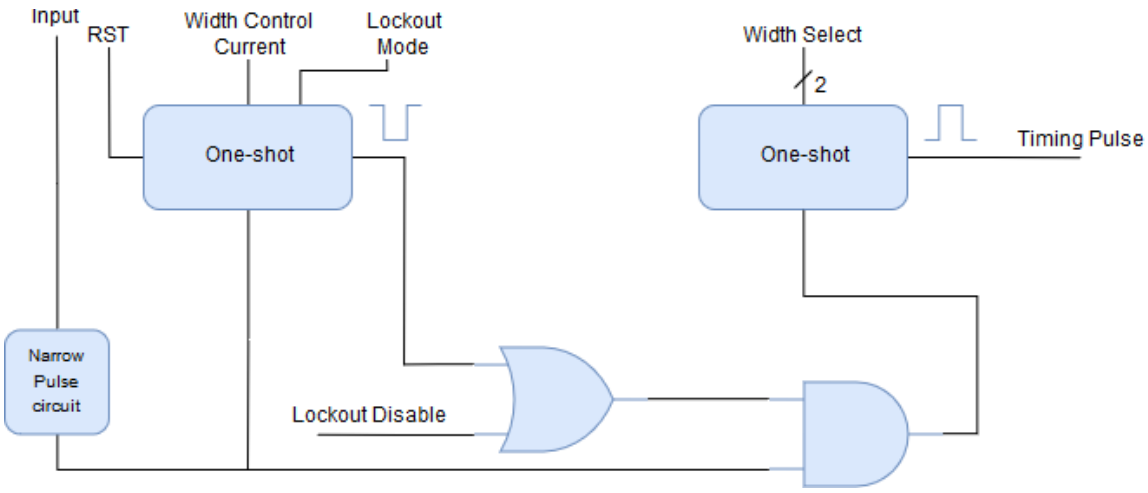


Figure 2.6: System level diagram of one-shot stage

A final output generation stage is used to qualify the timing pulse as well and create other useful channel outputs. The timing pulse for a channel 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. Therefore 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.

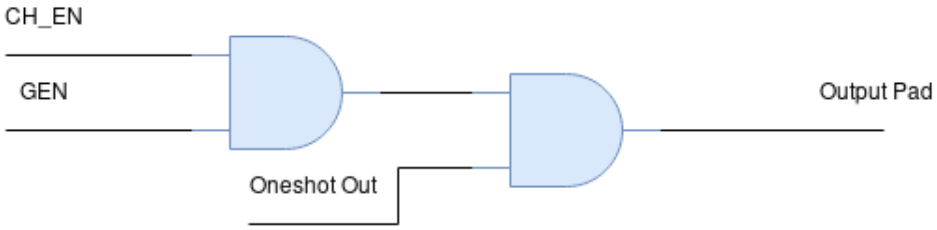


Figure 2.7: Timing pulse output qualification

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).

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 a 0.35 micron n-well process. The process supports two poly and 4 metal layers. Double poly capacitors, BJTs (Bipolar Junction Transistors) and a high-resistance layer are all available to the designer. NFET (N-type Field Effect Transistor) device properties are given in Table 3.1 while PFET (P-type Field Effect Transistor) device properties are available in Table 3.2. These parameters are provided to the reader because they may prove useful for understanding transistor-level circuits and evaluating design decisions discussed later in this chapter.

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

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

circuitry that is common to all of the signal channels. These include a power on reset circuit, a signal ground generator, a bandgap reference, and a pair of bias current generators.

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 designed by selecting digital standard cells provided in our PDK (Process Design Kit). The registers are D-registers possessing an enable input. The enable signal is active when a specified register's address and mode have 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 logic is presented 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 are set to match the address of the channel the register is located in, as well as the mode that should select the specific register. The output of the *ADDR* decoder circuit is then ORed with the MSB of the *MODE* field so

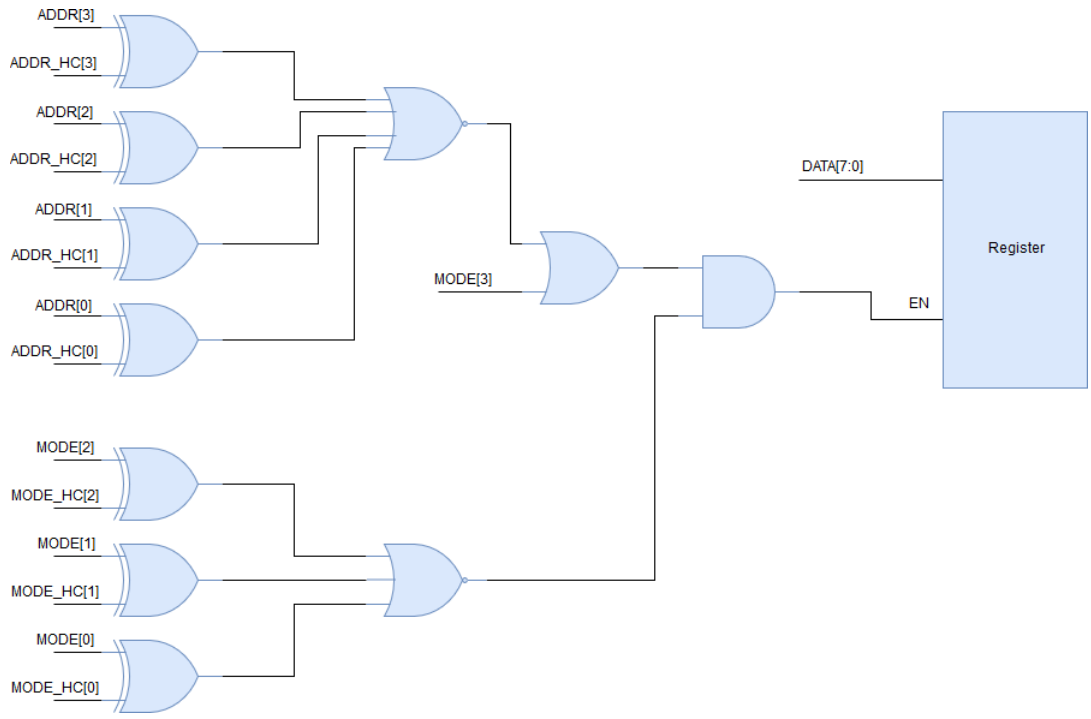


Figure 3.1: Register address and mode decoding

that all registers of a given mode can be selected if the global mode bit is set.

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 PDK. 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 (typically 3.5 μsec). 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 [Hastings, 2001], 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 referenced to a analog signal ground. This signal ground, called *AGND*, and must be at a potential half way between *AVDD* and *AVSS*. An analog ground generator circuit was provided in our PDK analog cell library so we used it. 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 according to Table 3.3.

Trim Bits	AGND voltage
000	1.36 V
001	1.43 V
010	1.49 V
011	1.56 V
100	1.63 V
101	1.69 V
110	1.77 V
111	1.84 V

Table 3.3: Signal ground generator trim values

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 selected from the PDK's analog cells library, and is a known working design. The bandgap voltage reference is used to provide a 1.2 V reference that is independent of temperature or power supply noise. The bandgap voltage is created by generating a PTAT current and then passing it through a resistor which in turn is placed in series with 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 reference voltage with a temperature coefficient of only -87 $\frac{\mu V}{^\circ C}$ (over the range of temperatures that the IC is expected to operate) as can be seen in Figure 3.2.

This bandgap circuit 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 also selected from the analog cells in our PDK. This current reference produces a bias current between 7.3 μA and 17.8 μA (across process corners) 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 transconductance linearly proportional to bias current but inversely proportional 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.

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 (or at least very little) temperature dependence. This zero temperature coefficient (ZTC) current was generated using the op-amp 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 op-amp 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-

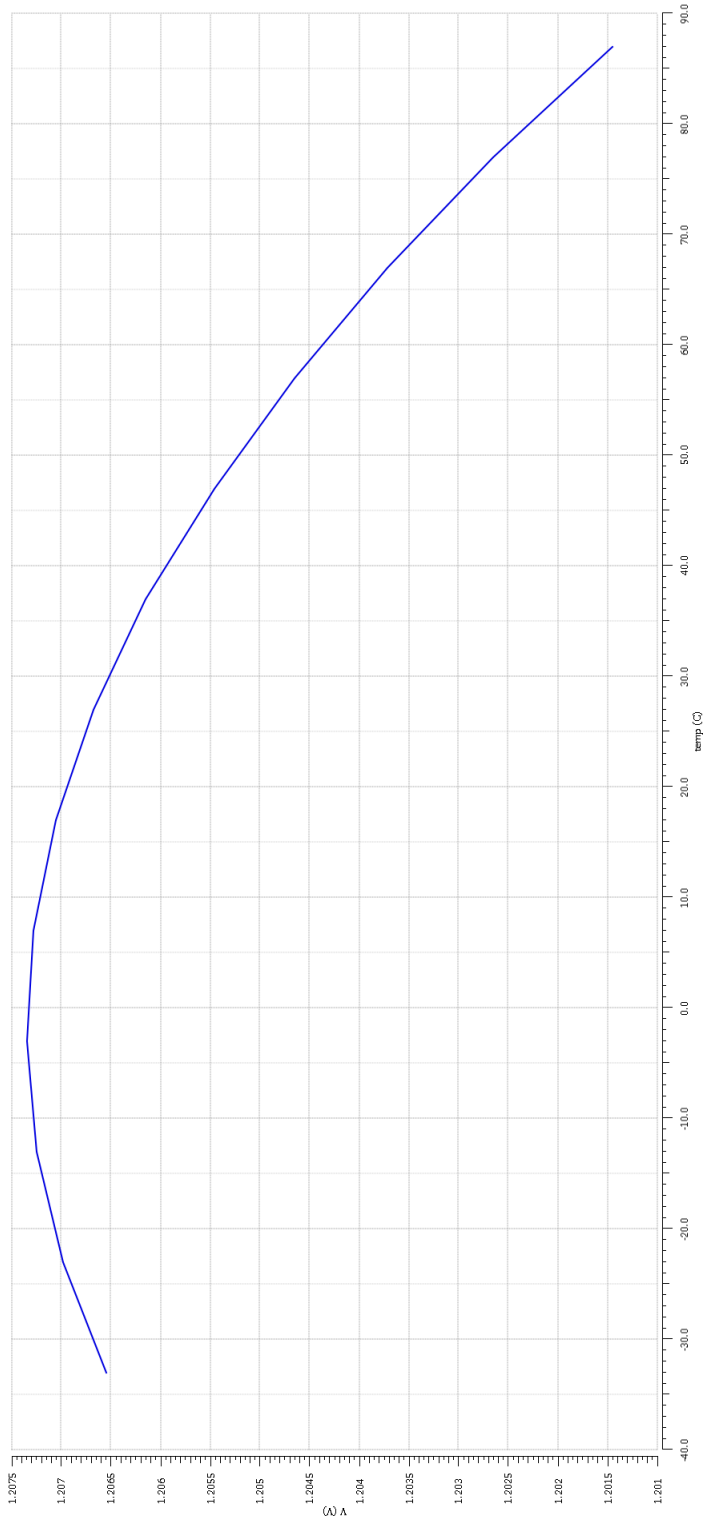


Figure 3.2: Bandgap temperature dependence.

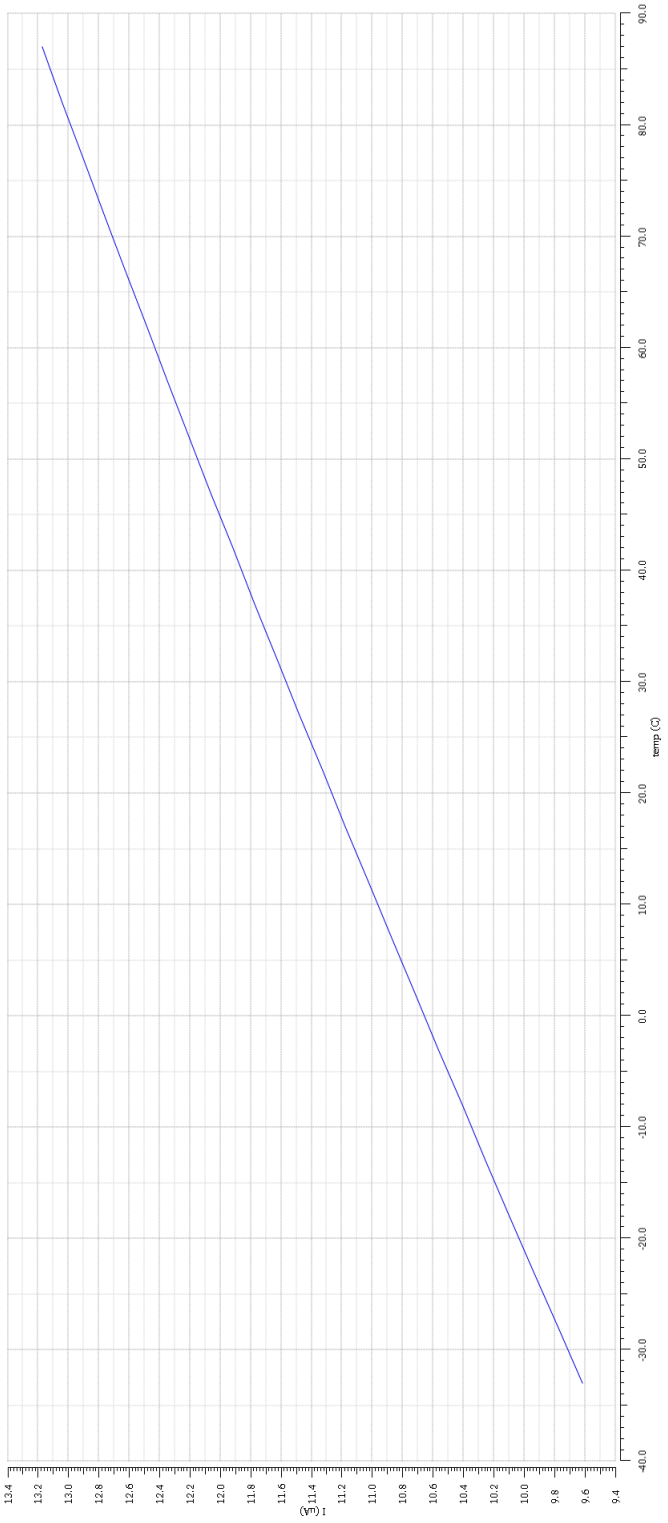


Figure 3.3: PTAT current reference temperature dependence.

inverting terminals of the opamp [Baker, 2010]. 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 does not depend on temperature.

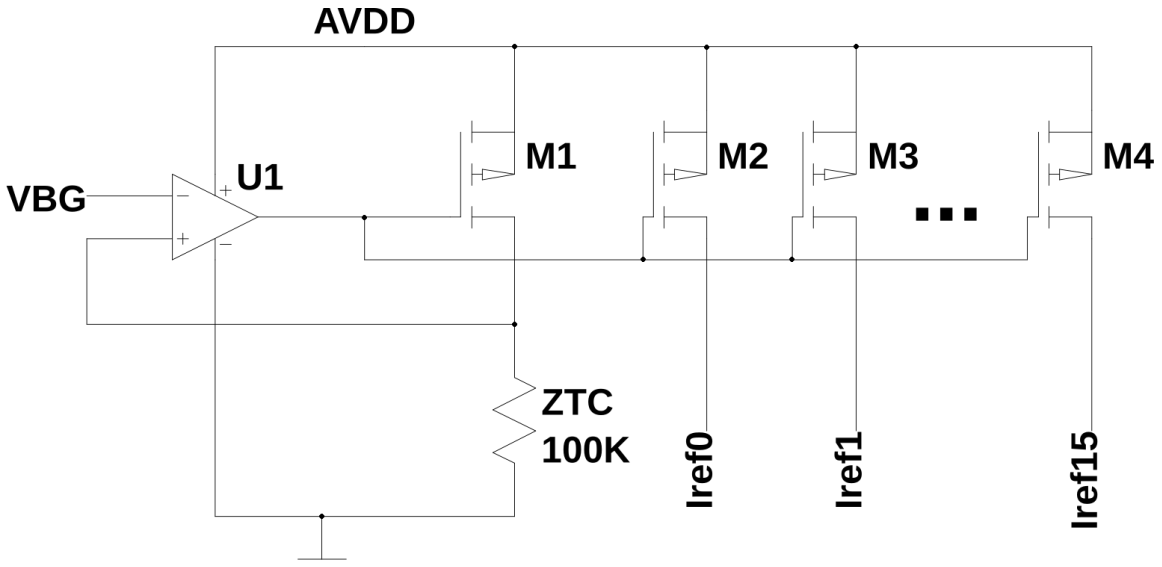


Figure 3.4: Zero temperature coefficient current generator.

	W (µm)	L (µm)	ng	I_B (µA)	g_m (µ℧)	V_{eff} (mV)
M1-M15	6	20	1	12	15.94	1345

Table 3.4: Zero tempco current generator device sizes.

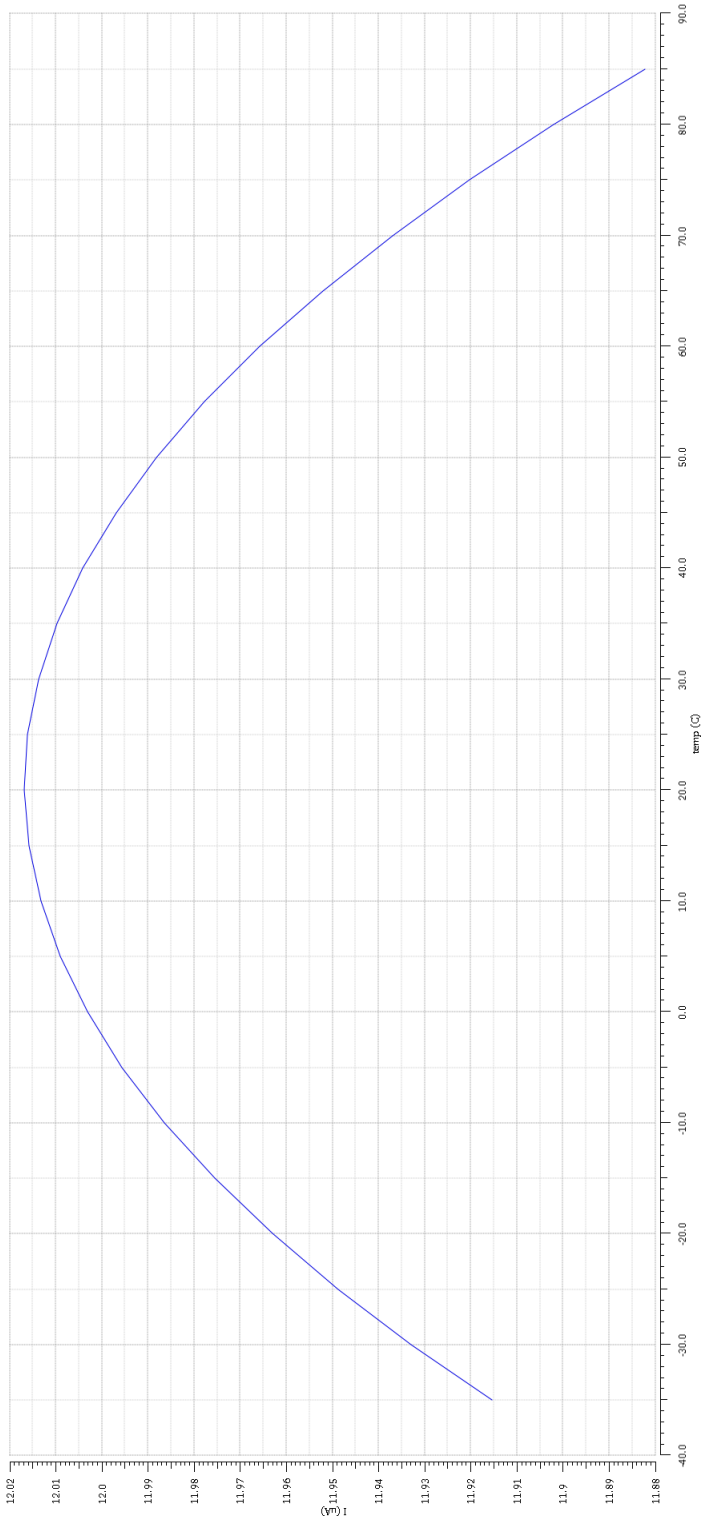


Figure 3.5: Zero temperature coefficient current temperature dependence.

3.2.7 Lockout DAC

It is desirable to prevent the one-shot circuit in the signal channels from re-firing for a given amount of time after they have fired. The length of this lockout time may need to change depending on the nature of the experiment. To do this, a lockout time can be set that is inversely proportional to the current at the output of a 6-bit digital to analog converter (DAC).

The DAC is implemented as a current scaling DAC using a R2R ladder topology [Baker, 2010] 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 technique relies on the excellent matching characteristics of FETs.

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 gives an extremely high output impedance of

$$Z_o = r_{ds7} \cdot g_{m7} \cdot r_{ds6} \tag{3.1}$$

A terminator circuit is used to ensure the current splits in half for the last stage (Stage0).

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, denoting negative polarity is enabled, then the NFET current mirror will

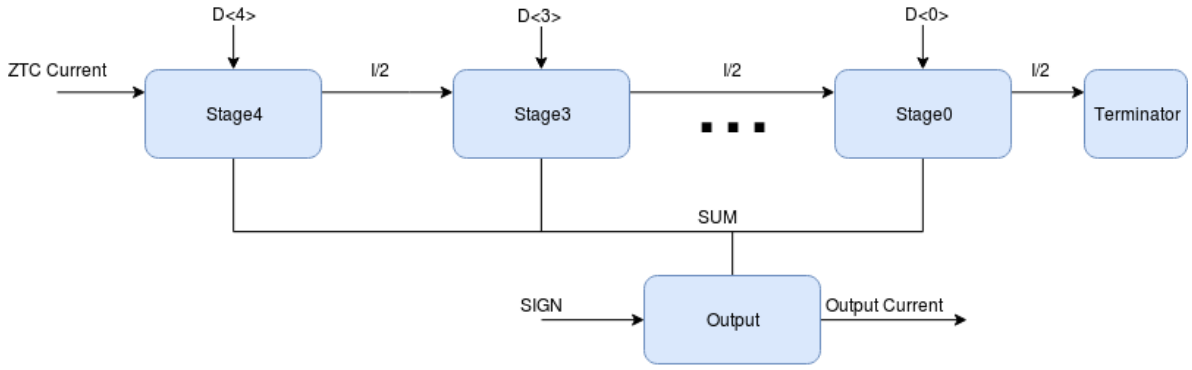


Figure 3.6: 6-bit bipolar DAC

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 discriminator threshold, and must be bipolar in that design.

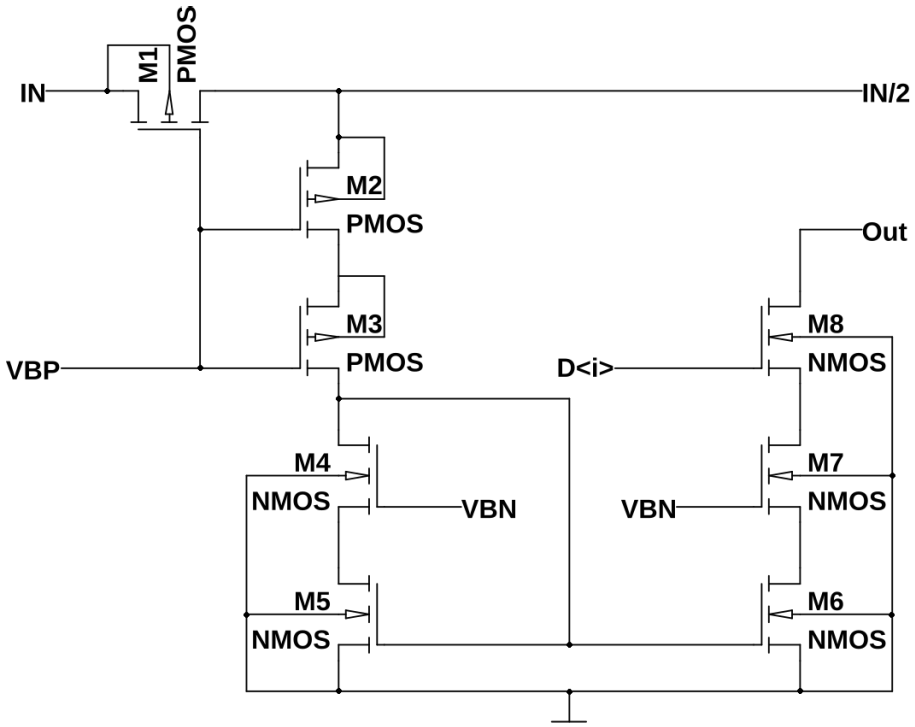


Figure 3.7: One stage of DAC using R2R ladder

	W (μm)	L (μm)	ng	I_B (μA)
M1	8	2	1	I_{in}
M2	8	2	1	I_{in} * 0.5
M3	8	2	1	I_{in} * 0.5
M4	4	8	1	I_{in} * 0.5
M5	4	8	1	I_{in} * 0.5
M6	4	8	1	I_{in} * 0.5
M7	4	8	1	I_{in} * 0.5
M8	7	0.35	1	I_{in} * 0.5

Table 3.5: Device sizes for single stage of current scaling DAC

	W (μm)	L (μm)	ng	I_B (μA)
M1	11.3	8	1	I_{sum}
M2	5.65	4	1	I_{sum}
M3	13.6	8	1	I_{sum}
M4	5.65	4	1	I_{sum}
M5	22.6	8	2	$2*I_{sum}$
M6	11.3	4	2	$2*I_{sum}$
M7	4	8	1	I_{sum}
M8	4	8	1	I_{sum}
M9	4	0.35	1	$2*I_{sum}$
M10	2	0.35	1	$2*I_{sum}$
M11	8	8	2	$2*I_{sum}$
M12	8	8	2	$2*I_{sum}$

Table 3.6: Current scaling DAC output device sizes

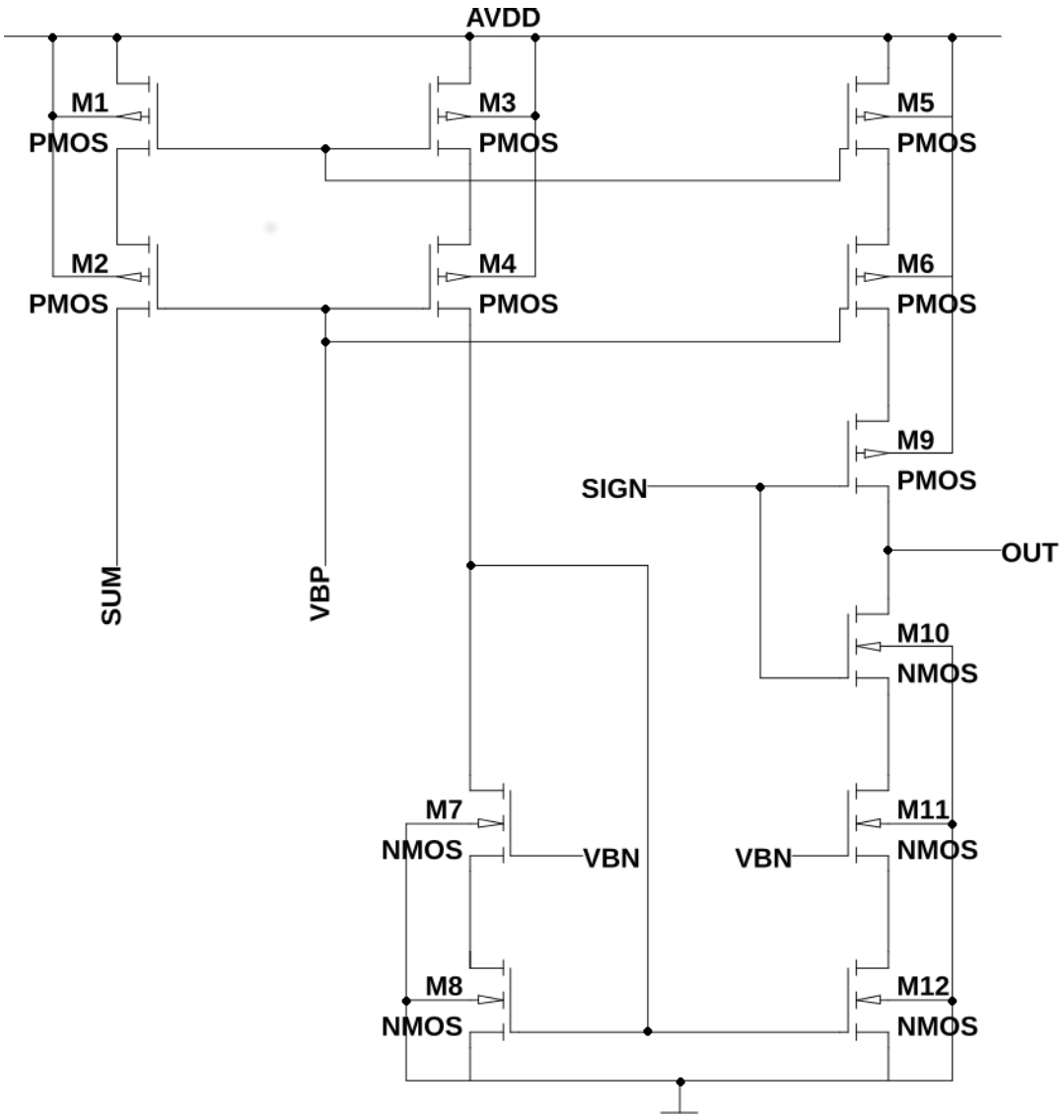


Figure 3.8: DAC output current stage

3.2.8 Multiplicity output buffer

One of the outputs of the CFD16C chip 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 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 $\frac{1}{g_{m2}}$ (here about 230 Ω).

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. The circuit can accommodate load capacitance up to approximately 30 pF with a settling time of less than 50 ns. The capacitor C1 provides compensation and ensures that the circuit has real poles so that the step response displays little or no ringing.

	W (µm)	L (µm)	ng	I_B (µA)	g_m (µ℧)	V_{eff}
M1	186.4	0.7	8	1000	4336	440
M2	186.4	0.7	8	1000	4336	440
M3	186.4	0.7	8	1000	4336	450

Table 3.7: Multiplicity buffer device sizes

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

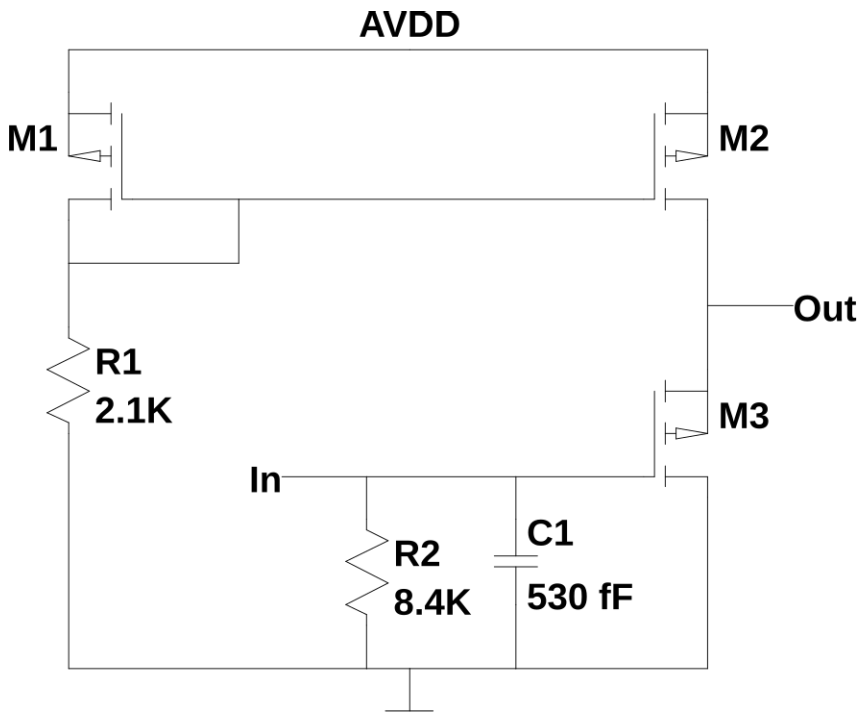


Figure 3.9: Multiplicity output buffer.

circuit, and an output generation circuit.

3.3.1 Programmable Nowlin circuit

In order to meet our walk and jitter specifications over such a large range of rise times (2 nsec to 192 nsec), the proposed Nowlin circuit must be highly programmable. To accomplish this, a "coarse" and "fine" grain adjustment of the Nowlin circuit was used. To do this, two modes are provided for the Nowlin circuit: a "long" mode (coarse) and "short" mode (fine). In the short mode the circuit must handle rise time constants in the range from 1 nsec to 16 nsec (in 1 nsec increments) while in long mode the IC will accommodate risetimes from 12 nsec to 192 nsec (in 12 nsec increments).

The fine adjustment is accomplished through the use of a programmable capacitor array. In other words, 10 - 90 % ristimes in the range of 2 nsec to 420 nsec will be supported by the IC. A 4-bit digital code allows the user to select one of 16 capacitor values from 0.5 pF to 8 pF (in 0.5 pF increments), thereby providing the fine adjustment.

The circuit for the "short" mode is presented in Figure 3.10 while the circuit for the long mode is given in Figure 3.11.

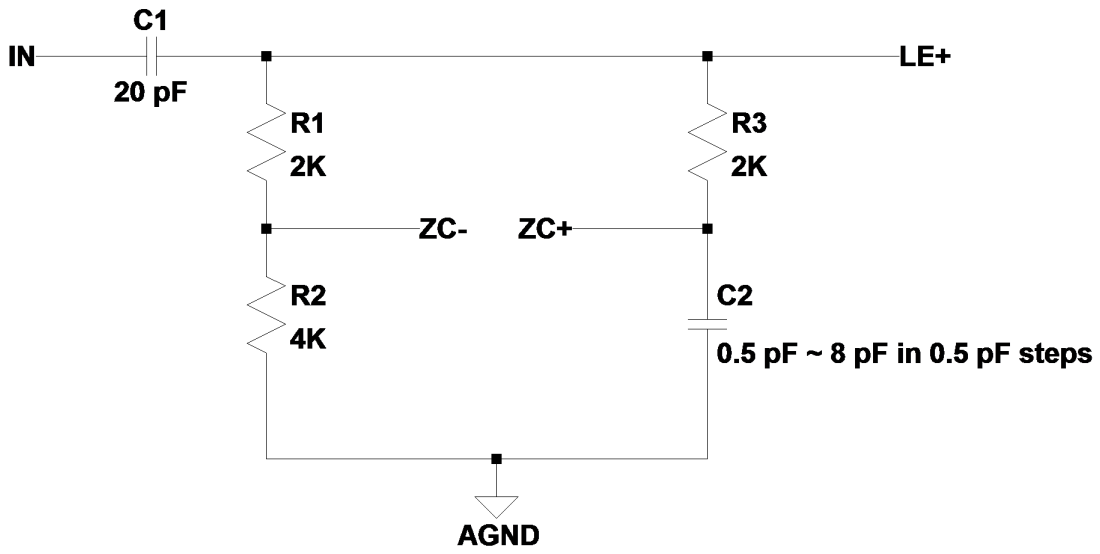


Figure 3.10: Nowlin circuit for short mode

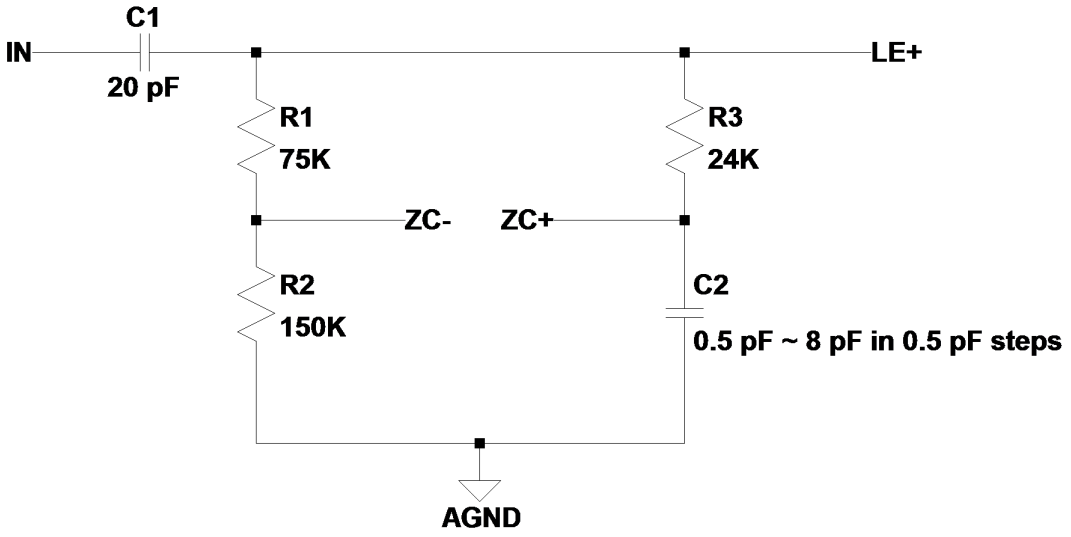


Figure 3.11: Nowlin circuit for long mode

The fraction $K = \frac{R2}{R1+R2}$ helps determine the amount of underdrive. A typical value for K is $\frac{2}{3}$ and is what was chosen for this application. It is important that the comparator

in the zero-cross circuit be driven low first and then high since it is the low-high transition that marks the onset of the pulse in an amplitude independent manner. It is important that the comparator underdrive and overdrive be comparable.

While the input to the Nowlin circuit is an exponential unipolar pulse with amplitude, ≈A, and risetime constant, τ_r. The signal associated with the differential output (i.e. difference in voltage on the ZC+ and ZC- nodes) from the Nowlin circuit, however, is a bipolar pulse centered around *AGND*. The bipolar signal crosses through zero when the attenuated input signal and the delayed input signal are equal to one another. In other words

$$k \cdot A \cdot \left[1 - e^{\frac{-t}{\tau_r}}\right] = A \cdot \left[1 - e^{\frac{-(t-t_o)}{\tau_r}}\right] \tag{3.2}$$

Notice that the amplitude, A, cancels out in the above expression and the time at which this bipolar pulse crosses through zero is independent of pulse amplitude. The delay, t_o, is approximately equal to the Nowlin time constant, $\tau_n \approx \tau_r$. This results in a zero-crossing time

$$t_z \approx \tau_r \cdot ln\left[\frac{e-k}{1-k}\right] \approx 1.8 \cdot \tau_r. \tag{3.3}$$

We shall refer to the slope of this signal, when crossing through zero, as the slew rate (SR) given by

$$SR \approx \frac{A}{4 \cdot \tau_r}. \tag{3.4}$$

For the minimum pulse amplitude, A, of 15 mV (and r value of 100) and a rise time constant, τ_r, this results in a SR_{max} of 1.7 $\frac{V}{\mu s}$. One can see from Figure 3.12 that it is important that the Nowlin delay constant be matched to the risetime constant. If the Nowlin delay time constant is much too short, we have insufficient underdrive to the comparator to force it to the low state. If the delay time constant is much too big, then the underdrive is large but the slew rate suffers. Setting the Nowlin delay time constant approximately equal to the rise time constant of the input waveform provides a good compromise between large underdrive and high slew rate.

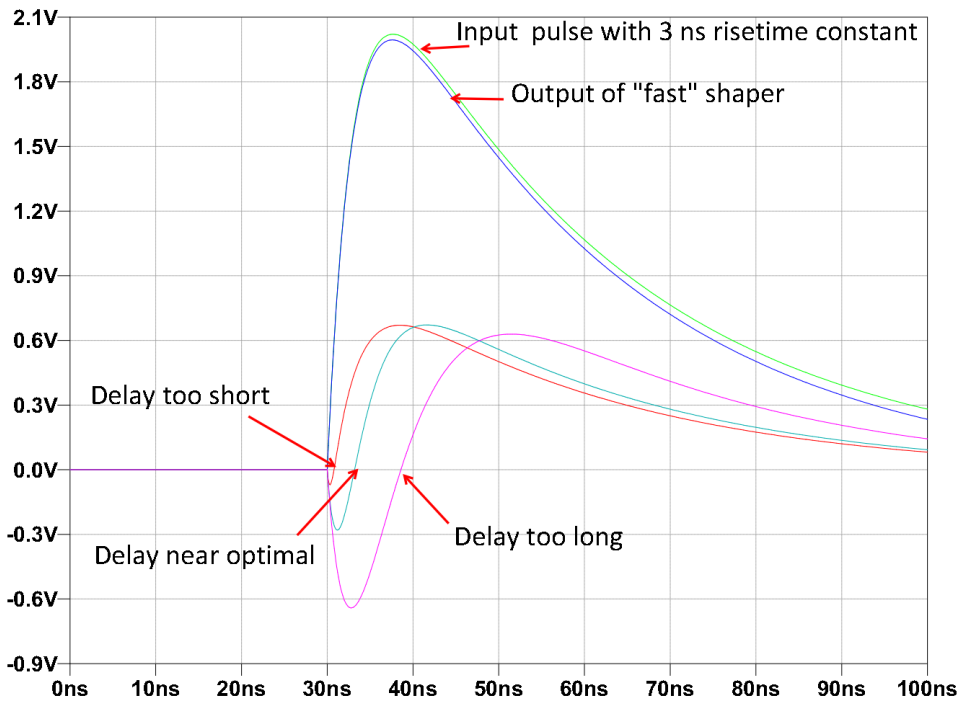


Figure 3.12: Input signal with 3 nsec risetime constant showing Nowlin delay effects

C1 and the series combination of R1 and R2 create a high pass filter (or "fast shaper") whose output is called LE+. The corner frequency of this high pass filter is given by $f_c = \frac{1}{2\pi \cdot (R1+R2) \cdot C1}$. This gives a corner frequency of ≈1.3 MHz in the short mode and ≈35 KHz in the long mode. This high pass filter output is used as input to the the leading edge discriminator.

The resistor R3 along with the programmable capacitor C2 form a delay circuit. The delay time is given by $\tau_n = R2 \cdot C2$ and is referred to as the Nowlin delay constant. The programmable capacitor in the Nowlin circuit is created by using switches (i.e. transmission gates) to connect individual capacitors in and out of circuit. Two transmission gates (t-gates) per capacitor are used to accomplish this. The programmable capacitor circuit can be seen in Figure 3.13.

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 3.13, 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 3.8 shows the bit order of these capacitor elements. Referencing this table shows that the total capacitance at the ZC+ node will be

$$C2 = D \cdot 0.5pF + 0.5pF \tag{3.5}$$

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 3.8 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 3.8: Programmable capacitor values and time constants.

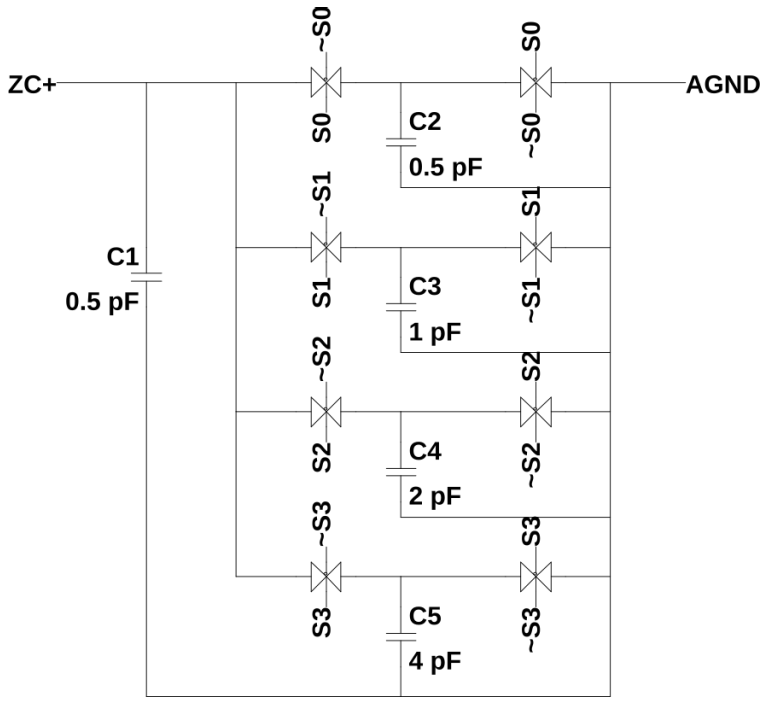


Figure 3.13: Programmable capacitor circuit.

3.3.2 Dynamic offset cancellation loop

For both the zero-cross discriminator and leading-edge discriminator to work properly DC offset voltages need to be canceled out before an input pulse arrives. This is achieved by placing a very slow amplifier in the feedback path. For the purposes of this amplifier we will model the forward path using a single pole model.

$$A(s) = \frac{K_1}{1 + \frac{s}{\omega_1}} \tag{3.6}$$

We will do the same for the slow amplifier in the feedback path.

$$B(s) = \frac{K_2}{1 + \frac{s}{\omega_2}} \tag{3.7}$$

The closed-loop response is then given by

$$A_f(s) = \frac{A(s)}{1 + A(s) \cdot B(s)} \tag{3.8}$$

Using the expressions for A(s) and B(s) given above and assuming $K = K_1 \cdot K_2$ is large, then the closed-loop gain is approximately given by the expression

$$A_f(s) \approx \frac{1}{K_2} \cdot \frac{1 + \frac{s}{\omega_2}}{1 + \frac{s}{K \cdot \omega_2} + \frac{s^2}{K \cdot \omega_1 \cdot \omega_2}}. \tag{3.9}$$

In the given application, we will choose $\omega_2 \gg \omega_1$. Under this assumption, the lower and upper corner frequencies, ω_L and ω_H, become

$$\omega_L \approx K \cdot \omega_2 \text{ and } \omega_H \approx \omega_1 \tag{3.10}$$

One observes that since K is large, the value of ω_2 must be at an extremely low frequency if f_L is to be less than 10 kHz. The frequency ω_2 is set by the time constant formed by the output resistance of the amplifier in the feedback path and a double-poly capacitor placed on the output of the amplifier. The value of the capacitor is 10 pF in the zero-cross circuit but 2 pF in the leading-edge circuit.

The key component of the DC offset cancellation is the slow amplifier in the feedback path. Since the amplifier only needs to drive a capacitive load, a Symmetric Miller OTA (Operational Transconductance Amplifier) was utilized. The circuit is given in Figure 3.14. The circuit makes use of source degeneration (resistive PFETs M_{11} and M_{12}) to reduce the transconductance of the OTA. The transconductance is further reduced by reducing the bias currents flowing through the input devices M_1 and M_2. Bias current is diverted to ground using PFETs M_5 and M_6. The gain of the circuit is quite low (around 30 dB). The design is not fully complete at the present time so device sizes for this OTA will not be given.

3.3.3 Zero-cross discriminator

The purpose of the zero cross detector is to produce a digital signal which marks the onset of the analog input pulse [Simpson et al., 1994]. It is important that this time be independent of pulse amplitude, A. The zero-cross circuit is constructed by cascading N

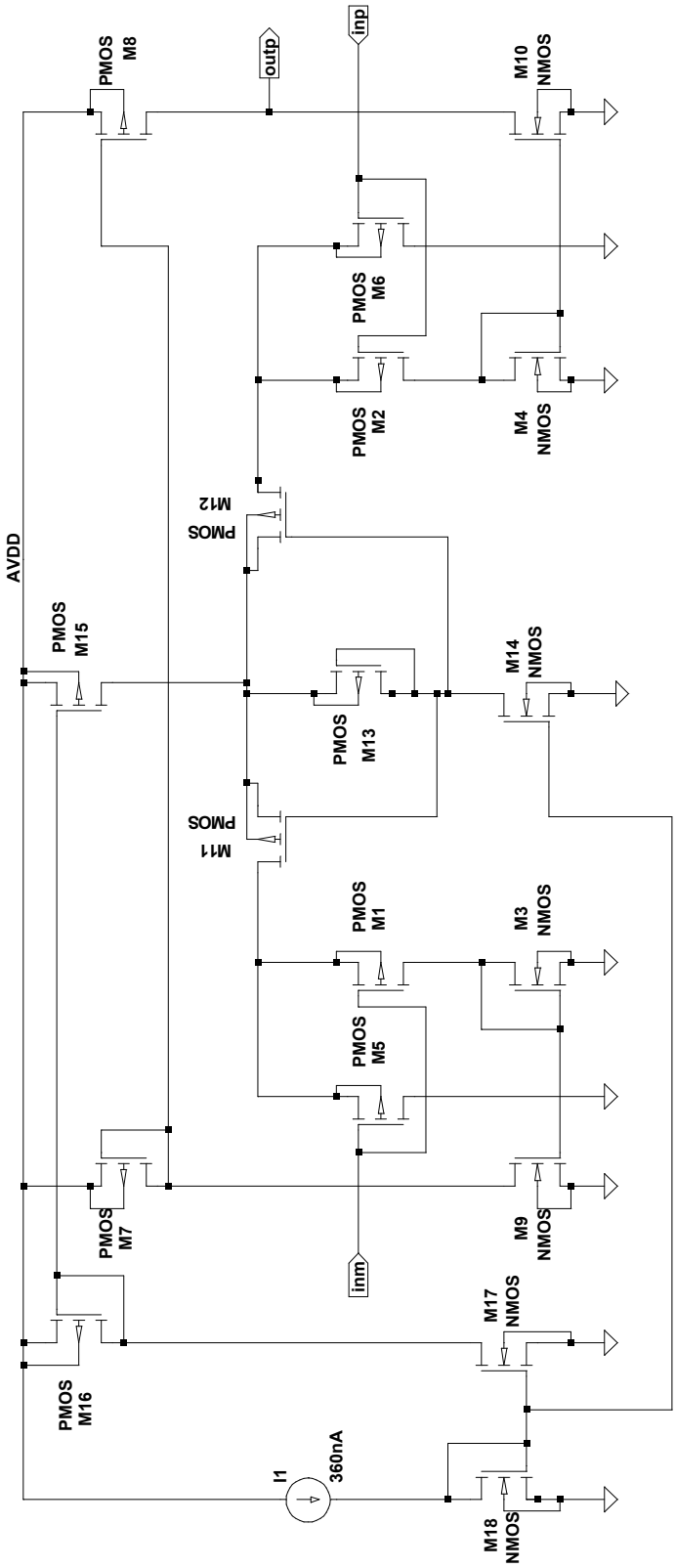


Figure 3.14: Schematic of low bandwidth, low gain OTA.

differential amplifier stages (where we have chosen N equal to 5) each possessing a very wide bandwidth, but relatively low-gain (≈ 4.5) with the final output stage driving a simple, yet very fast analog comparator.

The purpose of cascading a relatively large number of high-bandwidth but low-gain stages is to force linear operation where delay is independent of amplitude. In short, the cascaded amplifier, which we shall refer to as the "pre-amplifier" serves as a slew rate enhancer. For a first-order system, it is well-known that if the input risetime is not to be severely degraded, the bandwidth, BW, of the amplifier must obey the following equation

$$BW > \frac{0.35}{t_{10-90}} \approx 80 \text{ MHz} \tag{3.11}$$

In the equation above we have assumed the shortest risetime constant that we must accommodate, *i.e.* 2 nsec. Since we are cascading N (here N is 5) stages, the BW of each stage must be increased by $\sqrt{N} \approx 2.2$ resulting in a differential amplifier bandwidth of 180 MHz if the overall BW is to be at least 80 MHz. If the stage gain, G, is approximately 4.5, then the GBW of a single differential amplifier stage must exceed 800 MHz. The overall gain of the pre-amplifier is

$$Gain = G^N = 4.5^5 \approx \text{ 2000 or 66 dB.} \tag{3.12}$$

Thus, the highest rate of change at the output of the pre-amplifier is approximately 2000 times larger than the input worst case rate of 1.8 $\frac{V}{\mu s}$. Hence, the slew rate of the differential amplifier must be at least 3.6 $\frac{V}{nsec}$ if the pre-amplifier is to be linear for signals near the zero-crossing point. In other words, CFD walk performance will be limited by how large a slew rate for the differential amplifier can be achieved. Increasing the effective voltage of the input devices maximizes slew rate but how large of an effective voltage that can be used is ultimately limited by the supply voltage.

The heart of the zero cross discriminator is a wide-bandwidth low-gain differential amplifier. The amplifier design used is shown in Figure 3.15. The device sizes and bias

currents are given in Table 3.9. The value *ng* represents the number of fingers (i.e. gates) to use in the layout.

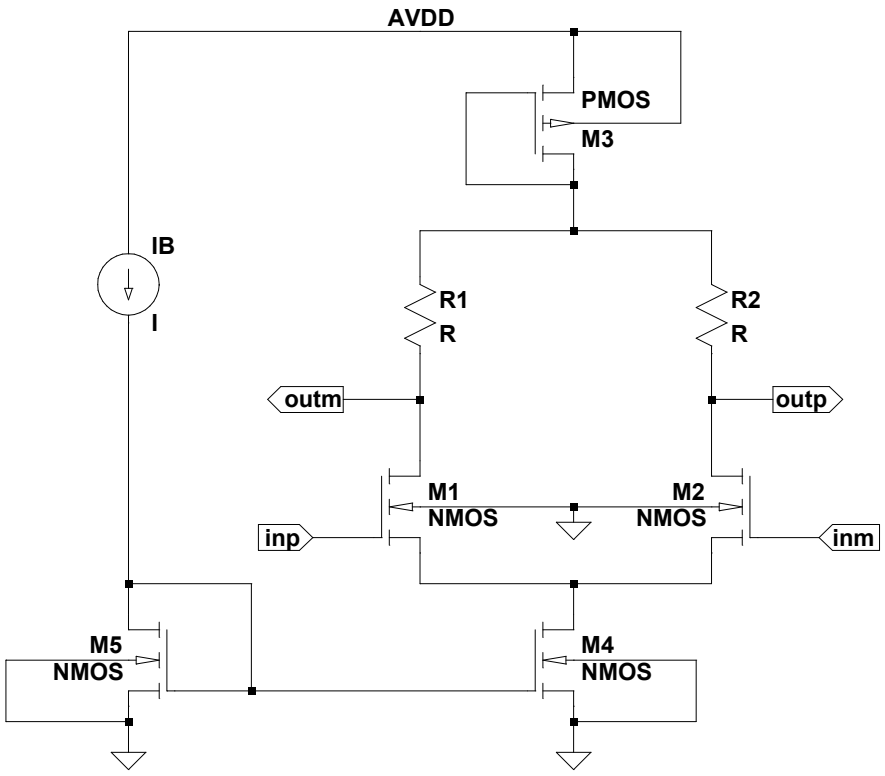


Figure 3.15: Schematic of low-gain high-bandwidth differential amplifier.

	W (µm)	L (µm)	ng	I_B (µA)	g_m (µ℧)	V_{eff} (mV)
M1	6.5	0.35	1	23	290	160
M2	6.5	0.35	1	23	290	160
M3	18	3	2	46	161	554
M4	32	4	2	46	307	300
M5	32	4	2	46	307	300

Table 3.9: Device Sizes for Zero-Cross Detector Differential Amplifier

The value of the resistors, R_1 and R_2, is 17 kΩ and the pair of matched resistors are implemented using a high resistance poly 2 layer. Transistor M_3 was added so that the output common-voltage could be set to approximately $\frac{V_{DD}}{2}$). Note, the current flowing through transistor M_3 is the same as the current, I_B, flowing through transistor M_4.

Hence the drain of M_3 is a virtual ground in much the same way that the drain of M_4 is a virtual ground. Hence,the addition of M_3 has no impact on the speed of the amplifier. The low-frequency gain of stage is

$$A_0 = g_{m1} \cdot R \approx 4.6 \tag{3.13}$$

and the bandwidth of the stage is

$$BW = \frac{1}{2 \cdot \pi \cdot R \cdot C_L} \tag{3.14}$$

where C_L is the total capacitive load on the output node.

The output swing for the differential amplifier is $2 \cdot R \cdot I_B$, centered around $\frac{V_{DD}}{2}$, where I_B is the tail current or 46 μA. Transistor M_5 mirrors the PTAT primary current of 11.5 μA and hence the value of 4 for the "ng" parameter in Table 3.9

Also crucial to the zero cross detector is a very fast high-bandwidth comparator. This circuit is shown in Figure 3.16. Transistors M_1 - M_4 make up a simple symmetric Miller OTA. The input common-mode voltage is assumed to be approximately half the supply voltage. The DC bias current through each of the 4 transistors is approximately 200 μA. The resulting GBW exceeds 3 GHz while the low-frequency open-loop gain is 26 dB.

Device sizes and bias currents are given in Table 3.10.

	W (µm)	L (µm)	ng	I_B (µA)	g_m (µ℧)	V_{eff} (mV)
M1	1.4	0.35	1	177	308	1150
M2	1.4	0.35	1	177	308	1150
M3	6.8	0.7	1	177	382	925
M4	6.8	0.7	1	177	382	925
M5	1	0.35	1			
M6	2.8	0.35	1			
M7	3	0.35	1			
M8	8.4	0.35	1			

Table 3.10: Device Sizes for Zero-Cross comparator

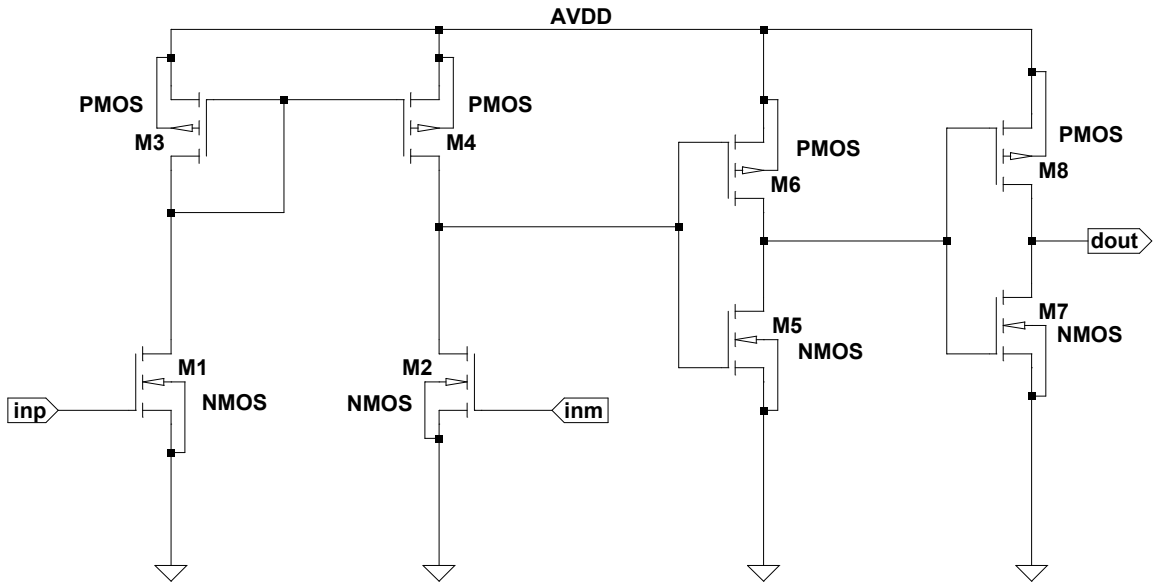


Figure 3.16: Schematic of high-bandwidth, very fast comparator.

The performance is identical for both positive and negative polarities. This is because for the negative polarity, transmission gates are used to flip the inputs to the comparator. This makes negative polarity pulses functionally identical to positive polarity pulses as seen from the input of the comparator, simplifying the design.

3.3.4 Leading-edge discriminator

The leading-edge circuit is very similar to the zero-cross circuit. The differential output voltage from the leading-edge pre-amplifier must be converted to a single-ended current. This is done using the linear transconductor shown in Figure 3.17. The transconductance of the stage is $\frac{1}{R}$ where R is 17 kΩ.

The leading-edge comparator is presented in Figure 3.18. It is very similar to the zero-cross comparator except two resistive voltage dividers have been added, one on the inverting and one on the non-inverting inputs. All resistors are 34 $k\Omega$ so that each divider can be represented by a Thevenin voltage equal to one-half the supply voltage and a Thevenin resistance of 17 $k\Omega$. The divider on the non-inverting input is used to convert the sum of the differential current output from the pre-amplifier and the current

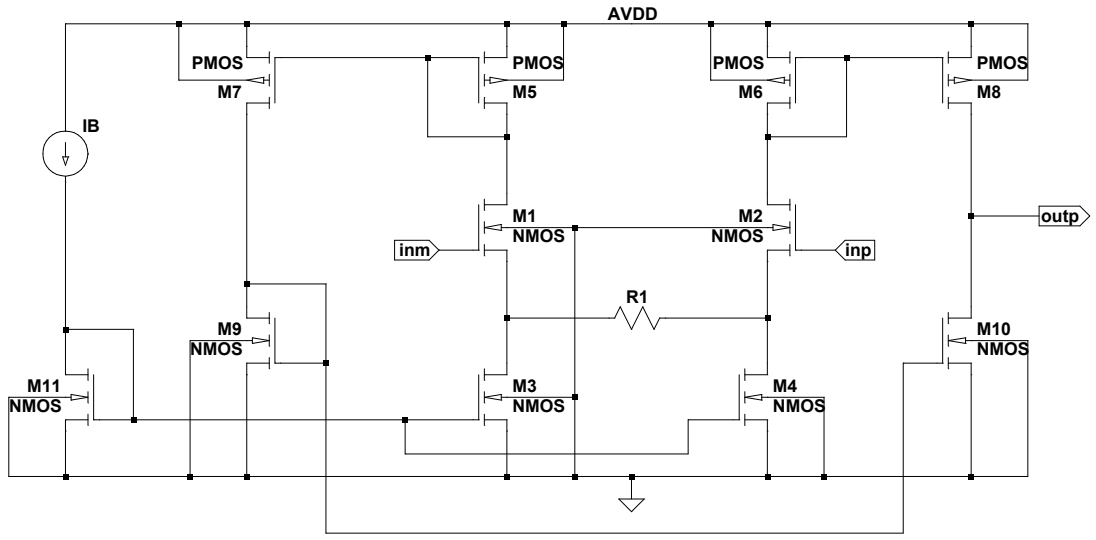


Figure 3.17: Schematic of differential voltage to single-ended current converter.

	W (µm)	L (µm)	ng	I_B (µA)	g_m (µ℧)	V_{eff} (mV)
M1	60	0.7	2	46	751	123
M2	60	0.7	2	46	751	123
M3	23	4	2	46	261	352
M4	23	4	2	46	261	352
M5	45	3	2	46	263	350
M6	45	3	2	46	263	350
M7	45	3	2	46	263	350
M8	45	3	2	46	263	350
M9	23	4	2	46	261	352
M10	23	4	2	46	261	352
M11	23	4	2	46	261	352

Table 3.11: Differential to single ended amplifier device sizes

output of the DAC into a voltage which is then compared to $\frac{V_{DD}}{2}$. Just as with the zero-cross comparator, the inputs of the leading edge comparator are swapped in the negative polarity mode.

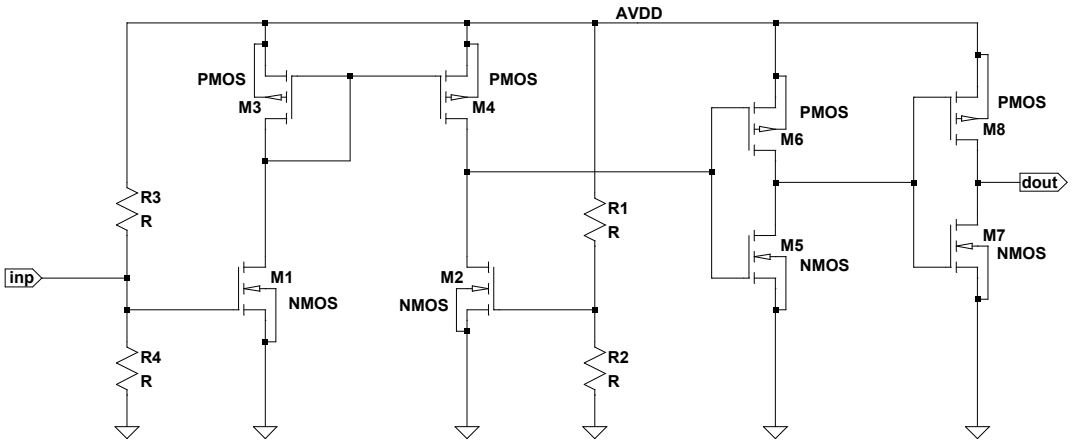


Figure 3.18: Schematic of leading edge comparator.

	W (µm)	L (µm)	ng	I_B (µA)	g_m (µ℧)	V_{eff} (mV)
M1	1.4	0.35	1	177	308	1150
M2	1.4	0.35	1	177	308	1150
M3	6.8	0.7	1	177	382	925
M4	6.8	0.7	1	177	382	925
M5	1	0.35	1			
M6	2.8	0.35	1			
M7	3	0.35	1			
M8	8.4	0.35	1			

Table 3.12: Leading-edge comparator device sizes

3.3.5 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 pulses used to start the gated integrators and the TVCs

on the PSD8C chip. Recall that the gating of the integrators is relative to the rising edge of the one-shot output while the TVC is started by the trailing edge of the one-shot output. Thus, both edges of these one-shot output pulses must exhibit low timing jitter. The one-shot circuit design is shown in Figure 3.19.

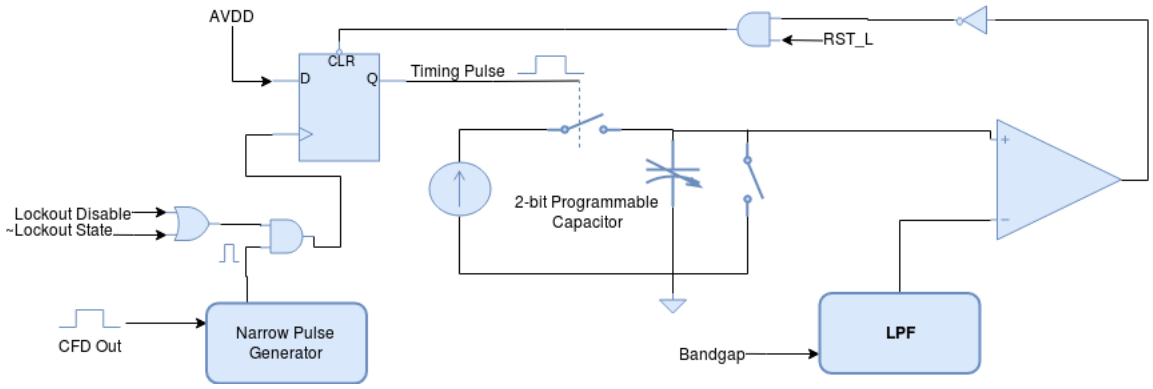


Figure 3.19: One-shot circuit with lockout

The one-shot circuit works by utilizing a D-type flip flop with an asynchronous clear input. When the CFD circuit (i.e. 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 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 3.20.

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 outputs a logic high. The comparator output triggers the *CLR* input on the D-flip flop to go low, forcing

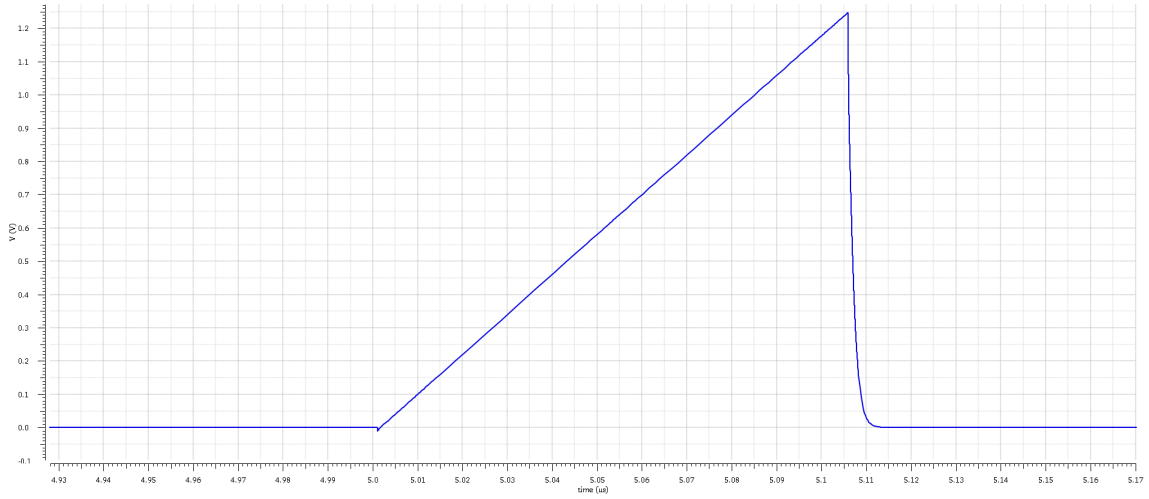


Figure 3.20: Comparator ramp input

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 (i.e.. 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 steps 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 time steps 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.

3.3.6 Final output generation

All of the signal channel outputs 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 (fast) 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 3.13. 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 3.13: 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 3.21, 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).

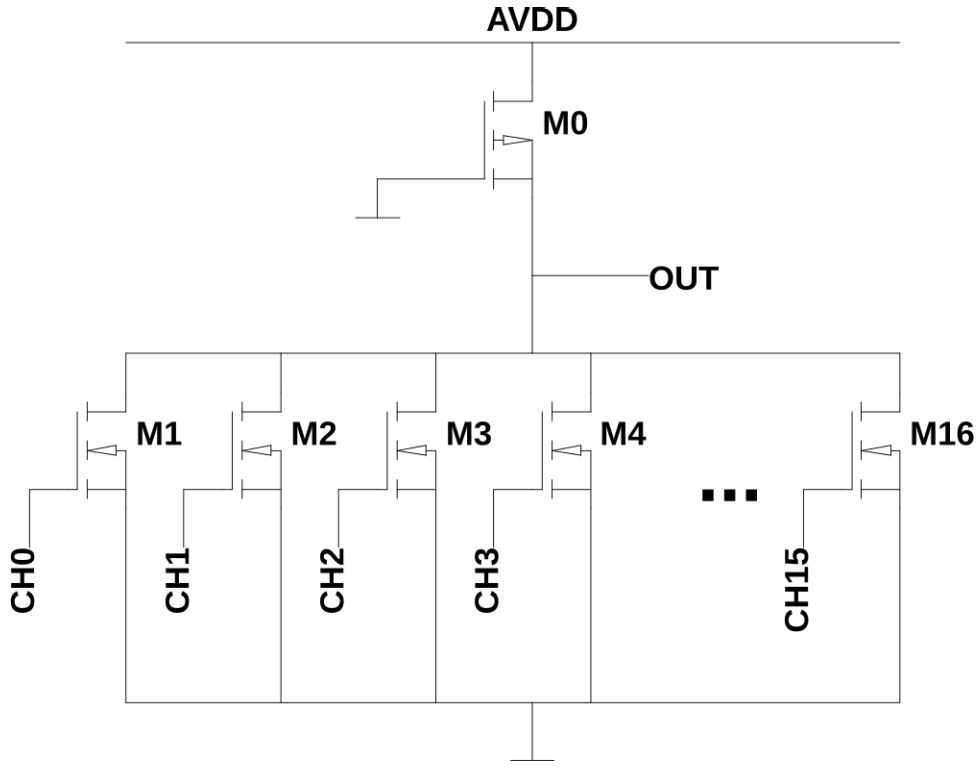


Figure 3.21: Fast pseudo-NMOS NOR

CHAPTER 4

SIMULATION RESULTS

4.1 Verification of Circuits in Common Channel

The common channel contains configuration registers and biasing circuitry. In order to test the configuration, a System Verilog test fixture was used to load each configuration register with data. Setting the configuration registers agreed with the expected behavior from each of the bits outlined in Table 2.1. The biasing circuitry was verified by observing the output of the circuits being biased in the signal channels. The channel circuitry worked as expected as well confirming that the biasing circuitry was functioning within specification.

4.2 Walk Characteristics of CFD Circuit

Time walk is is used to describe systematic variation in the rising edge of the discriminator output signal as a function of pulse height. If we assume a single-pole response for the analog comparator with a gain-bandwidth-product of GBW_c, it is not difficult to show that the variation in propagation delay, Δt_{pd}, due to input pulse amplitude is

$$\Delta t_{pd} \approx \sqrt{\frac{V_{DD}}{2 \cdot \pi \cdot GBW_c \cdot G^N \cdot SR_{min}}} \tag{4.1}$$

where V_{DD} is the supply voltage, G is the gain of a single differential amplifier stage, N is the number of stages, and SR_{min} is the SR with a pulse amplitude of 15 mV. Table 4.1 summarizes the expected walk performance (over the dynamic range of 15 mV - 1.5 V) for three representative risetime constants.

In reality, the time walk may increase at low amplitudes because of residual offset. Ultimately, how well the discriminator performs at low amplitude will depend on the effectiveness of the dynamic offset cancellation loop. It can be shown that residual offset

alters the time at which one crosses zero as given by

$$\Delta t_{cross} \approx \tau_r \cdot \left[\frac{V_{os}}{(1-k) \cdot A} \right] \tag{4.2}$$

where V_{os} is the residual input referred offset and A is the amplitude of the input pulse. It is important that the residual offset be driven to the sub-mV level. The "slow" DC offset cancellation loop has to be "slow" enough so as not to interfere with the pulse as it passes through the zero-cross discriminator but yet fast enough so that pulse repetition rates as high as 1 kHz can be accommodated.

τ_R (ns)	Time Walk (ps)
3	300
30	980
150	2200

Table 4.1: Time walk as function of risetime constant, τ_r for $G = 4.5$, $N = 5$ and $GBW_c = 3GHz$.

The values given in Table 4.1 assume no residual offset. For the case of $\tau_r = 3$ nsec, A = 15 mV, and a residual offset of just 200 μV, the effect of the residual offset is to add an additional 130 psec of walk resulting in a value of 430 psec. The typical simulated walk performance for a signal with a rise time constant of 3 nsec is presented in Figure 4.1. Two hundred Monte Carlo runs were performed and the results of this simulation is presented in Figure 4.2.

As can be seen from Figure 4.2, the walk across all process and mismatch is centered between 400 psec and 600 psec for a three sigma design. The expected walk performance is thus within the 500 psec or less design specification. Moreover, given the monotonic and systematic behavior of the response, calibration techniques can be used to compensate for the effects of walk.

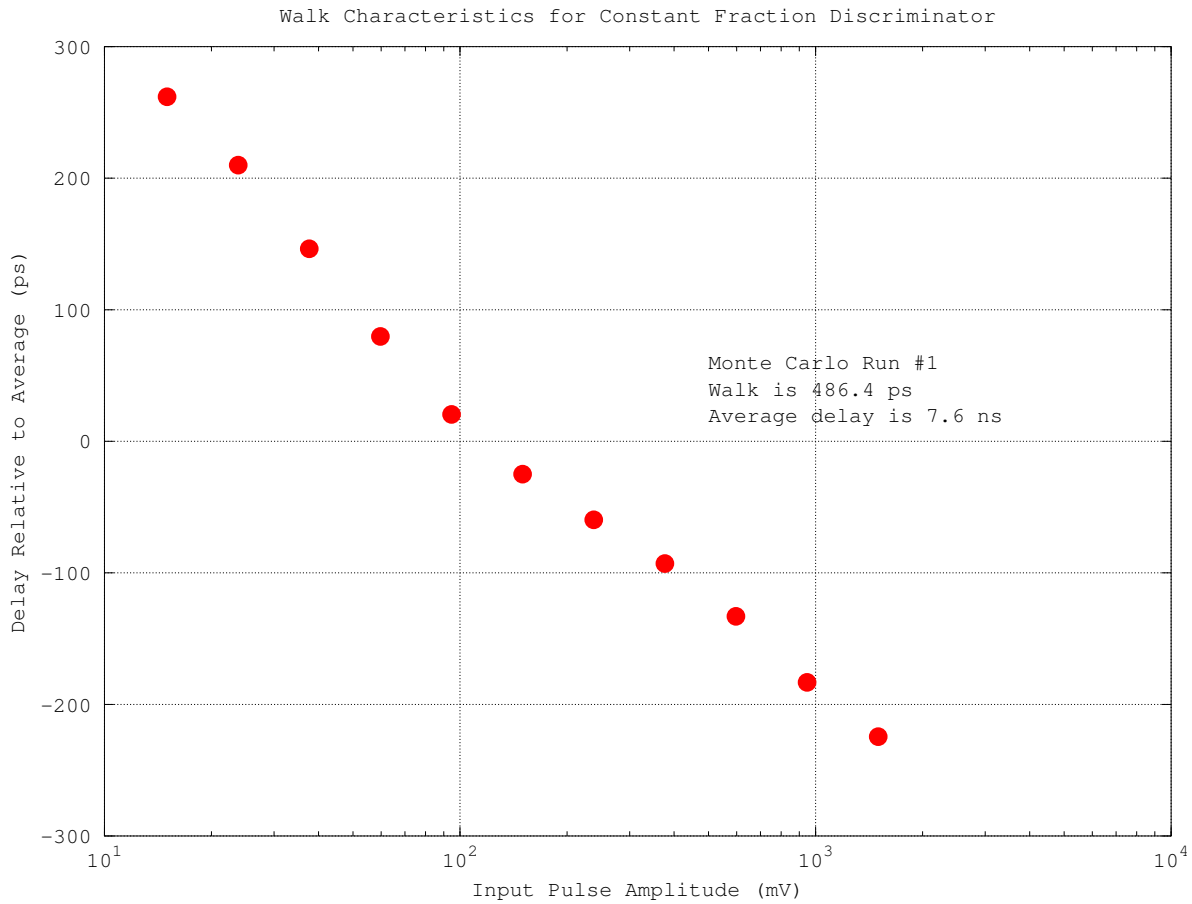


Figure 4.1: Typical walk performance with a a signal whose rise time constant is 3 nsec.

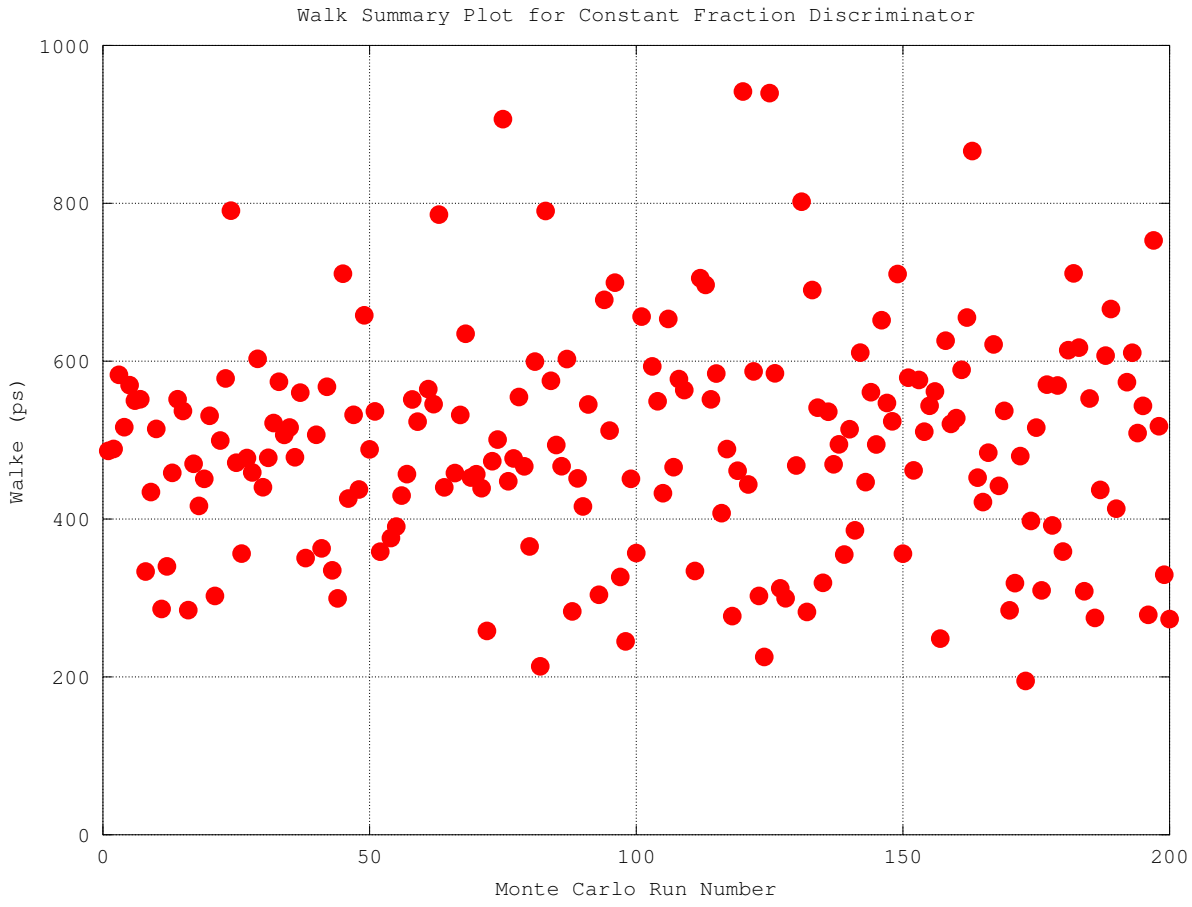


Figure 4.2: Summary walk performance with a signal whose rise time constant is 3 nsec.

4.3 Jitter Performance

Jitter is a consequence of the electronics noise associated with the Nowlin circuit as well as the noise of the *first* differential amplifier in the zero-cross discriminator circuit (where we assume the noise of succeeding stages when referred to the input is negligible). Worst case timing jitter can be estimated by taking the total integrated noise at the input to the zero-cross pre-amplifier and then dividing by the worst case SR given above. The noise voltage is due to the thermal noise of the input devices on the first differential amplifier stage, the thermal noise of the gain-setting resistors in that stage, and the thermal noise of the resistors in the Nowlin circuit. Jitter can be minimized by biasing the input FETs at high current levels and keeping all resistor values small.

$$\sigma_t = \frac{\sigma_v}{SR} \tag{4.3}$$

The theoretically predicted jitter performance plot for the short range mode is given in Figure 4.3 and the expected performance in the long range mode is provided in Figure 4.4. One observes that for the short range mode, the jitter (as a percentage of the 10-90 risetime of the input signal is less than about 3 percent. For the long range mode this value only increases to about 5 percent. These values are generally considered to be "excelllent".

4.4 Verification of One-Shot

The output of the 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 mis-firings. For the lockout feature, a wide distribution of programmable lockout times was desired (*i.e.* a a logarithmic distribution). Figure 4.5 shows the distribution of the lockout times available in both the short and long modes. As can be seen from the plots the distribution is nearly logarithmic, with the lockout time being given by

$$T_{lockout} = \frac{K}{D \cdot \Delta I} \tag{4.4}$$

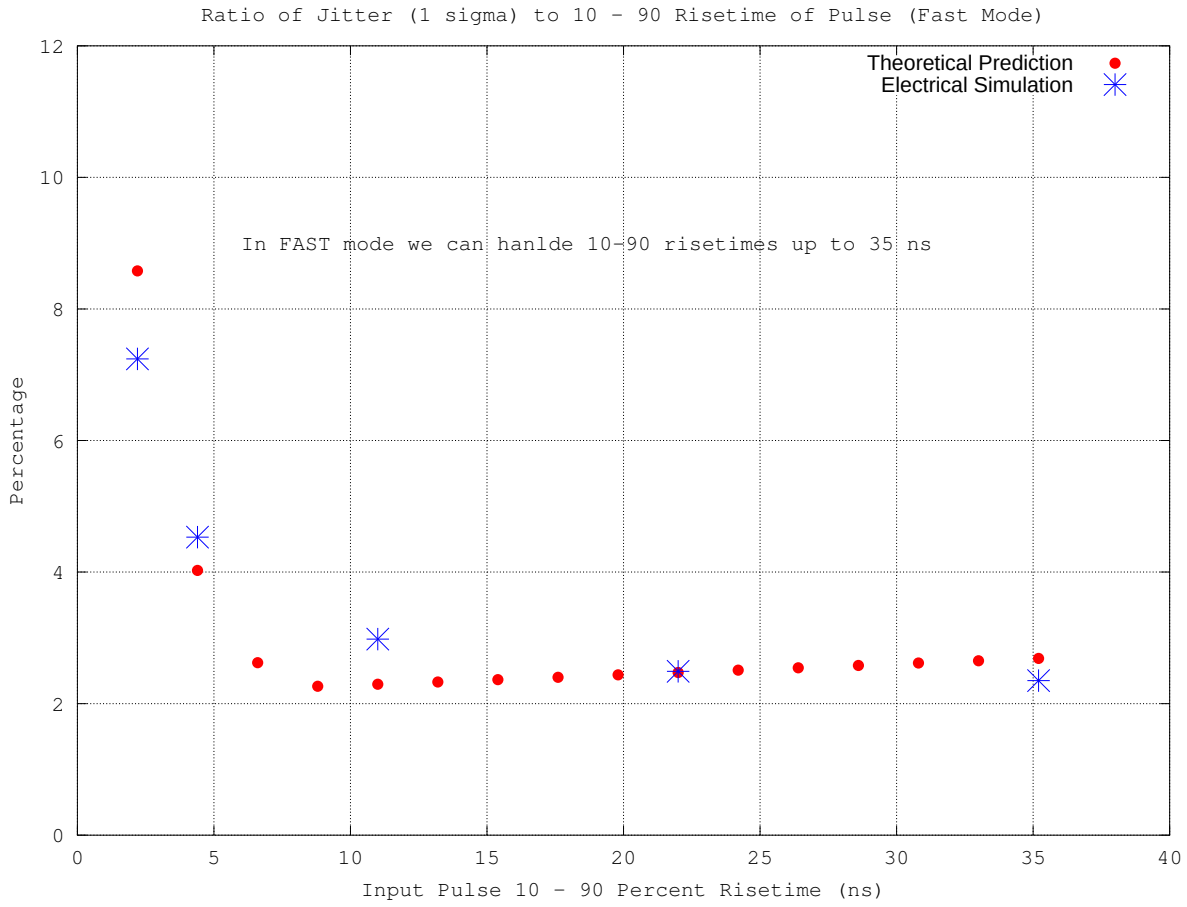


Figure 4.3: Jitter performance for short time constant mode.

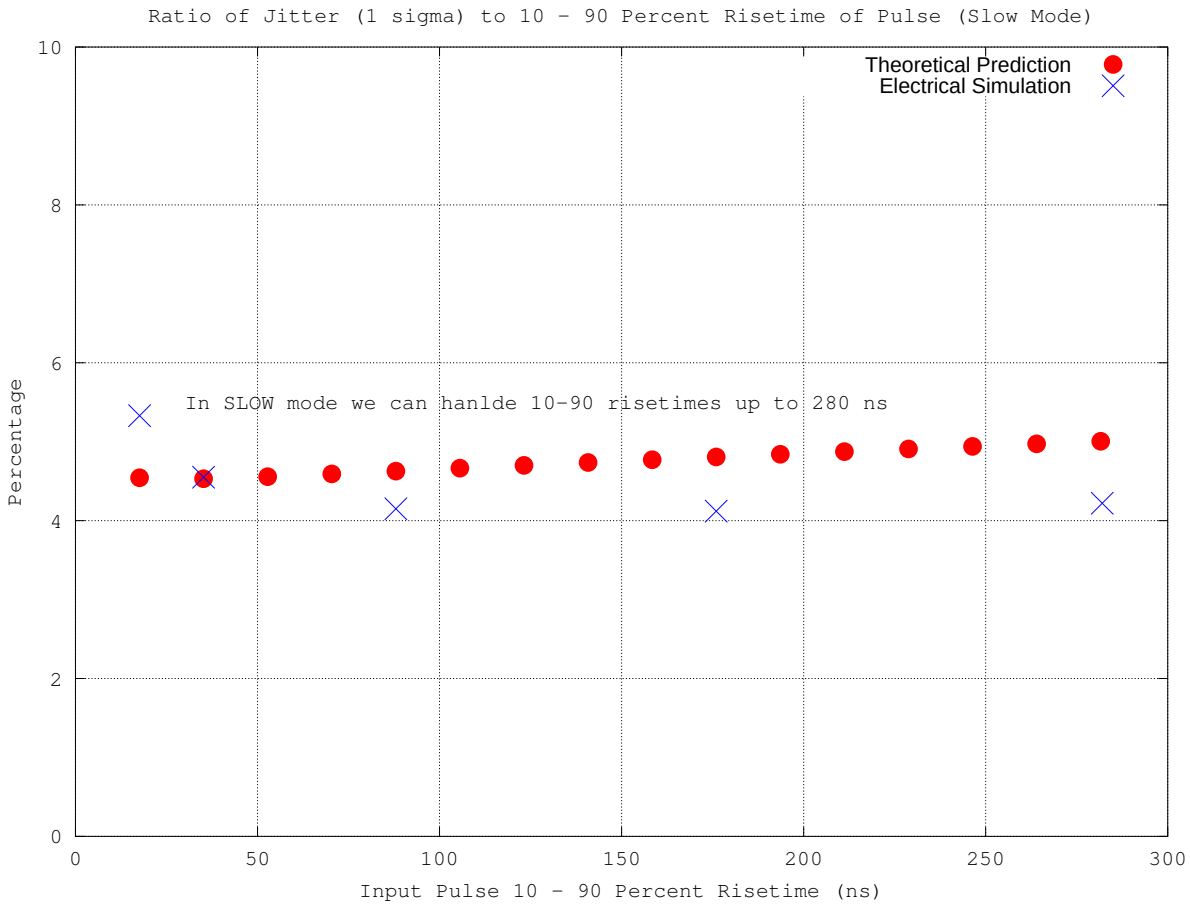


Figure 4.4: Jitter performance for short time constant mode.

where K is given by the the expression $K = C \cdot V_C$ (C is the value of the capacitor in the lockout one-shot *i.e.* 0.25 pF), D is the digital code word given to the lockout DAC, and ΔI is the step-size of the lockout DAC. V_C is peak amplitude of the voltage ramp generated within the lockout circuit. The ΔI term will change depending on the lockout mode, being 380 nA in short mode and 38 nA in long mode.

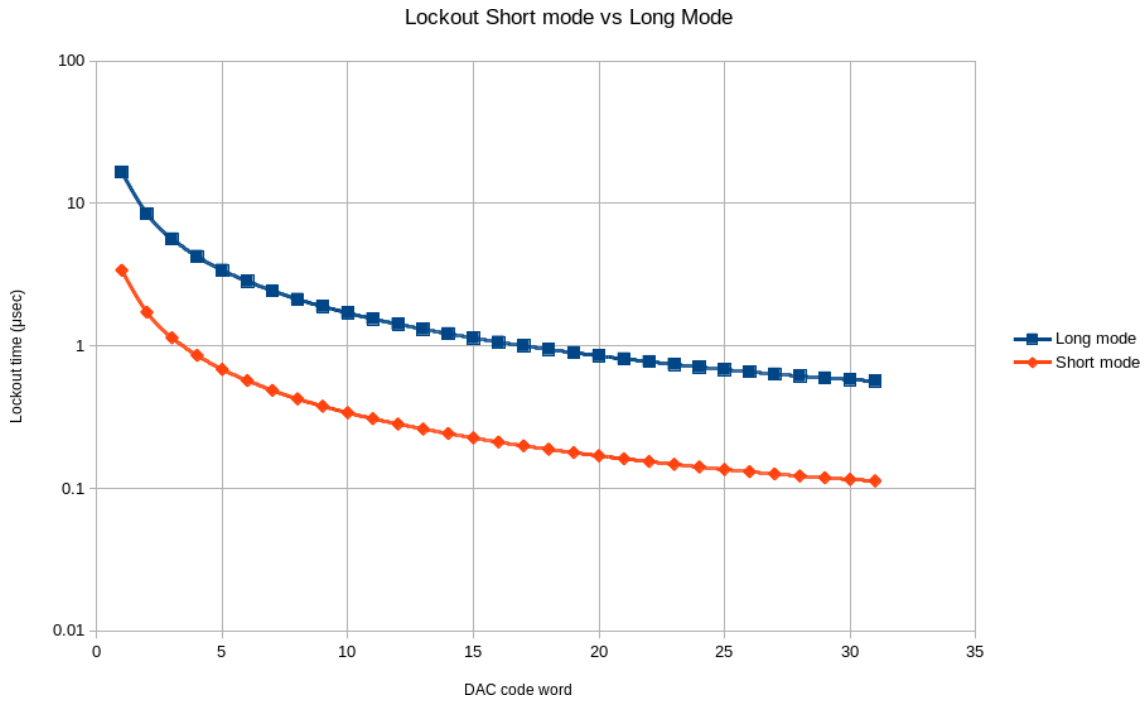


Figure 4.5: Lockout times for short and long modes

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 4.2). 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.

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 $\sigma_j = \frac{\sigma_V}{V_{BG}} \cdot T_{oneshot}$, where σ_V is the noise voltage in the current source (PFET current mirror) charging the one-shot capacitor in Figure 3.19 and V_{BG} is the bandgap

voltage. We estimate the jitter to be approximatlely 8 ps for the 50 ns wide output pulse while the 500 ns wide output pulse will display 25 ps of jitter. The noise associated with the current source is not 10 time worse (but rather just $\sqrt{10}$ time worse) as one might expect because a larger capacitor, C, was used (5 pF versus 0.5 pF for the 50 ns wide pulse). The use of the larger capacitor value reduces σ_V.

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 4.2: 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 differential non-linearity error (DNL) of less than 0.5 least significant bits (LSBs). Differential 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 (1 LSB) 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 4.6–4.7 show the results of 200 Monte Carlo simulations. With each separate Monte Carlo simulation, the values of the process parameters such as 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 exceptions. It can also be seen that the worst case INL and DNL performance occurred during simulation number 150.

Figure 4.8 shows the worst case run out of all 200 simulations. It can easily be seen

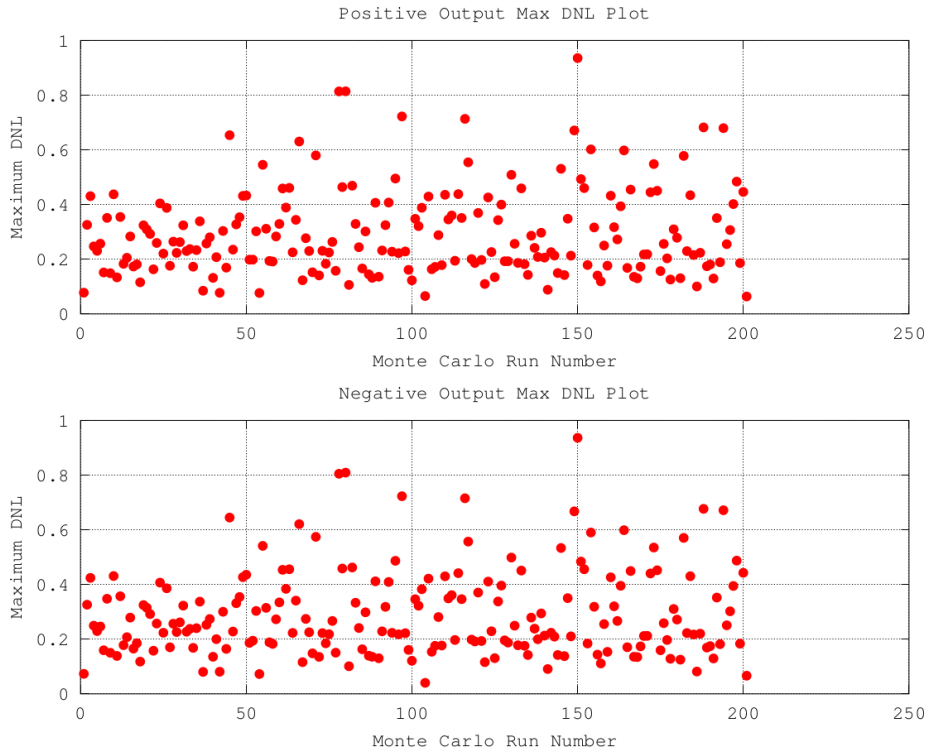


Figure 4.6: 6-bit DAC DNL error summary

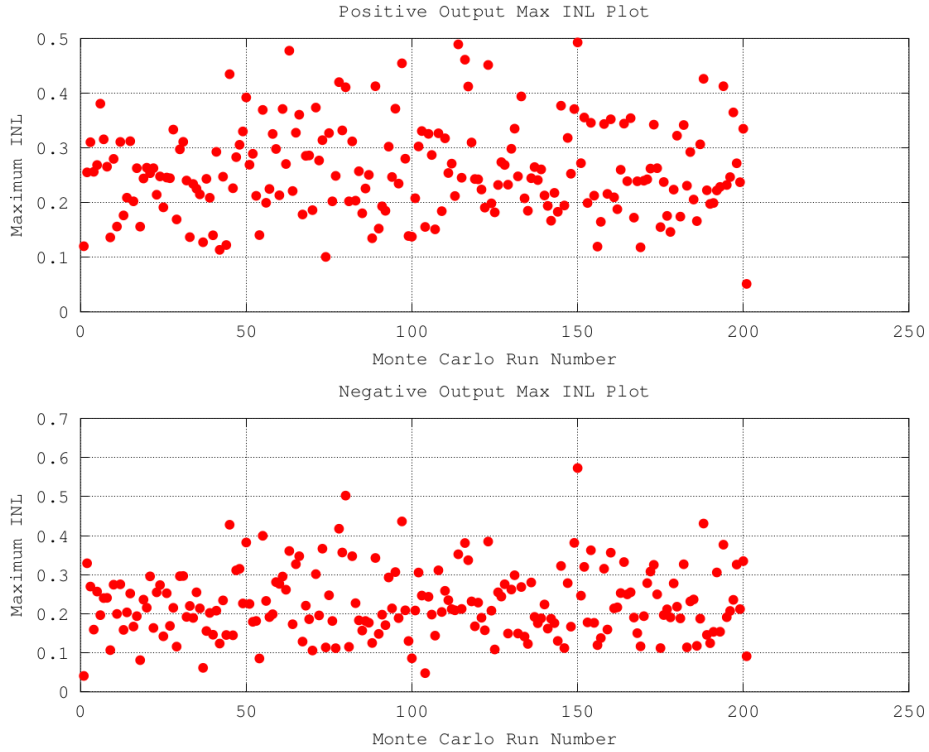


Figure 4.7: 6-bit DAC INL error summary

here than the DNL is well below 0.5 LSB for all increments of the code word except at the half way point. An average case simulation result is shown below in Figure 4.9, showing that on average the INL and DNL errors are very good.

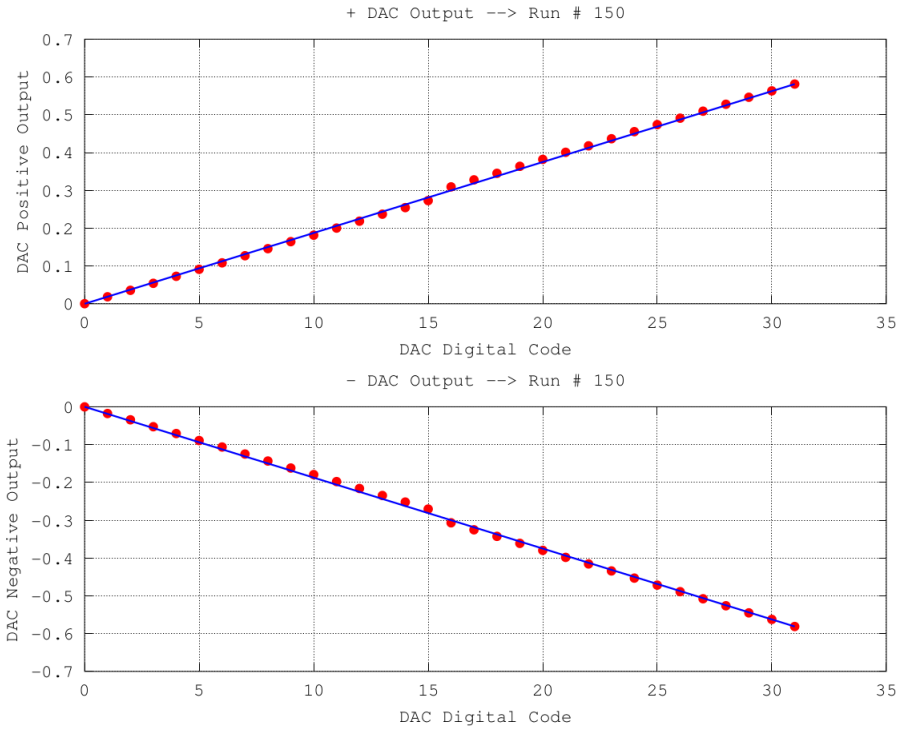


Figure 4.8: 6-bit DAC worst case error

4.6 Chip-Level Verification

A large System Verilog testbench was used to test the performance of the chip as a whole. Every channel was hit with a pulse of increasing amplitudes between 15 mV and 1.5 V. Registers were configured between pulses to adjust circuit parameters and verify correctness. Full chip simulations have already been completed to test the functionality of the following:

- Walk characteristics of the output timing pulse
- Both Nowlin time constant modes

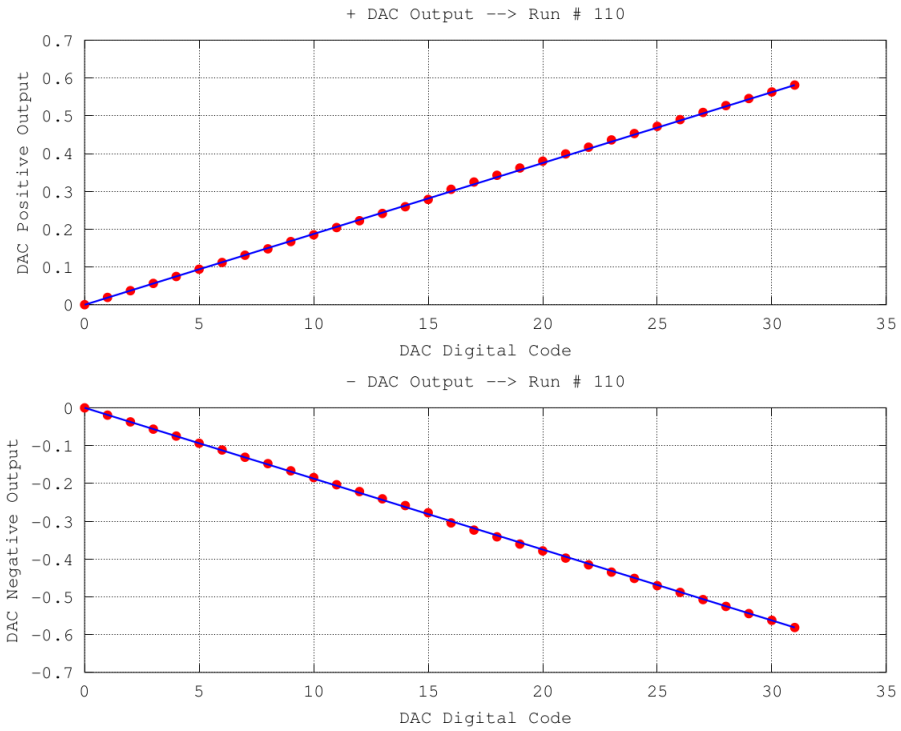


Figure 4.9: 6-bit DAC average case error

- Both lockout time modes
- Multiple input pulse amplitudes and risetime constants (both polarities)
- Multiple lockout times (matching the predicted values from Eq 4.4)
- Global mode bit to load all registers of a given mode
- Global enable control to disable all 16 channels
- Power on from a cold start
- All four oneshot pulse widths and their tolerance
- Individual channel enable bits to disable specific channels

The walk times were more thoroughly tested using a separate simulation of the CFD and oneshot circuits; however, since every channel is hit with a different amplitude pulse,

the walk perfromance can be observed in a full chip simulation. Walk is easily visualized in Figure 4.10, which shows the rising edge of an output timing pulse from each channel after hitting each channel with an input pulse of increasing amplitudes (15 mV on Channel 0, 1.5 V on Channel 15).

4.6.1 Testing Procedure

To verify that the all of the circuits in the IC were functioning properly, a System Verilog test fixture was created to do the following:

- Pulse power on to simulate the chip turning on from a cold start.
- Set the *ADDRDAT* bus to the appropriate address/mode and data values on the proper *STB* edge.
- Load each configuration register with data to test every possible configuration value.
- Hit every channel with a pulse of different amplitude and risetime to verify channel circuits are working.

The test fixture consists of three parts: a tasks file that specifies routines for setting the power, data, and control lines, a driver that specifies which routines to execute and with what parameters at given times, and a testbench instantiation that is used to output the driver data to a value change dump file (VCD). Once the testbench was simulated and a VCD file was generated, a python script was used to convert the VCD data to piecewise-linear (PWL) which could be instantiated as a voltage source using Verilog-A. Figure 4.11 details the process of setting up a simulation for the IC.

The tasks file specifies high level routines to manipulate power, data, and control lines on the IC. The first task to be called in any test driver is the *init* task, which simply sets all lines to 0 V to simulate a chip that has been powered off for a long period of time. Another task, *power_on*, can then be used to simulate power being applied to the chip.

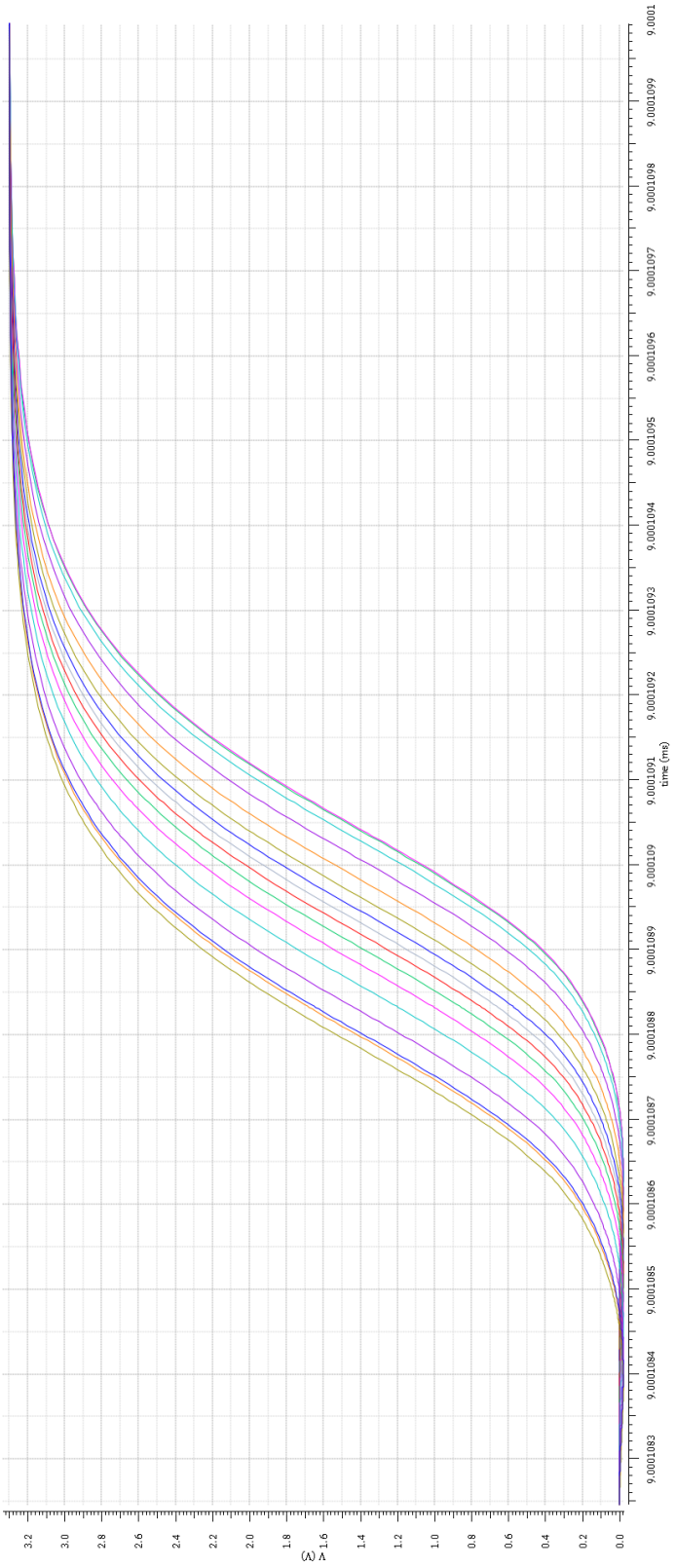


Figure 4.10: Sixteen output timing pulses overlayed to demonstrate walk characteristics

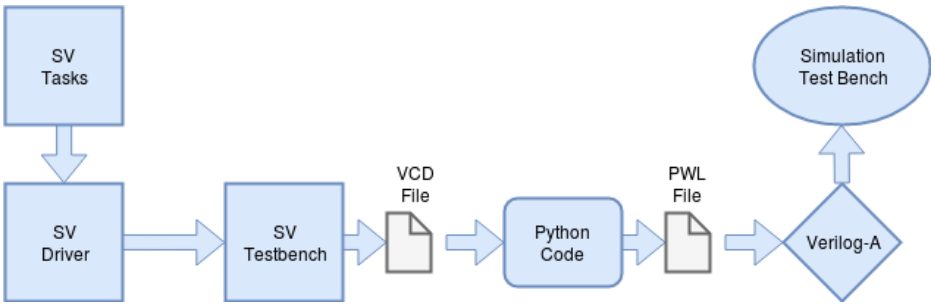


Figure 4.11: Full chip testing procedure

This allows the IC to be simulated from a cold start rather than from a steady sate. Other tasks were written to set/clear each of the pin mounted control lines (see Table 2.2), as well as the *STB* line to latch address, mode, and data. High level tasks were used to set mode by allowing the user to specify a named mode (e.g. PCAP, ONESHOT, etc.) rather than having to remember the correct mode bits from Table 2.1. Similarily, data can be set by using a task to set a specific function. For example, the one-shot output pulse width can be set by *set_os_width(<2-bit data>)*. Finally, tasks are also provided to set the amplitude, risetime, and firing time of an expoenential input pulse. These analog lines are provided to a Verilog-A test fixture which can generate the pulse and record data.

The driver file is used to specify the parameters of an experiment. The driver is best understood as a timeline of events that will happen during a simulation. For example, the general structure of the driver used for testing the CFD16C was the following:

1. Begin with cold start (i.e. call *init* task)
2. Power the chip (i.e. call *power_on*)
3. Configure chip appropriateley for incoming pulses (i.e. call tasks to set registers)
4. Enable/disable relevant control lines
5. Generate analog pulses

6. Repeat 3-5 for input pulses with different parameters

Once a driver is prepared, it is instantiated in a testbench fixture. The testbench is used to create a driver object that can be simulated and captures the waveform data in VCD files. A an example VCD file is shown below.

```
$date
    Jul  2, 2018  10:36:01
$end
$version
    TOOL:   ncsim 09.20-s038
$end
$timescale
    1 ps
$end
$scope module verilog_driver_tb $end
$var real      64 !    SIG_A  $end
$var wire       1 $    SIG_B  $end
$var wire       8 %    SIG_C  $end
$upscope $end
$enddefinitions $end
$dumpvars
r0 !
b0$
b00000000%
$end
#6000
r3.3 !
b1$
b10100101%
#8000
```

The VCD file header first gives a date and version number for the software that created the file. The file then specifies the timescale being used. In the example above the timescale of picoseconds (ps) is used. Next, the variables from the testbench fixture are given symbols to be referenced by in the *$dumpvars* section. In the example above there are three signals: an analog, or *real*, signal called SIG_A, a digital wire called SIG_B, and a digital bus called SIG_C. The symbols given to these are !, $, and % respectively. The *$dumpvars* section contains information about how the variables change over time. Timestamps are denoted by a number preceded by a # (e.g. #6000 above). Between timestamps are the variables that changed at that time and the value that they changed to, with the value given first followed by the symbol and no space seperating. For analog signals the value is preceded with an **r**. The timestamp at the end of a VCD file represents

the end time of a simulation.

While the VCD file contains all of the information needed to create a voltage source, there is no way to use it directly in the Cadence software tools. The simulation tools used for the design of this IC are, however, able to create voltage sources that are piecewise-linear in nature by providing a PWL file that describes them. For this a python script was used to parse the VCD file and create a PWL file with the same data contents. The script can be modified to change the rise/fall times, logic levels, and scale factors for analog signals. Once these values are set, running the script will create a PWL file for each of the symbols defined in the VCD file. A PWL file is similar to a VCD file but only defines timestamps and values for one signal line instead of many. The standard PWL format specifies the x-axis (time) first, followed by the y-axis value (voltage) for that x-input seperated by a space. An exaple is shown below.

```
0 0
5301000p 0
5302000p 3.3
5921000p 3.3
5922000p 0
6201000p 0
6202000p 3.3
6801000p 3.3
6802000p 0
7101000p 0
7102000p 3.3
7501000p 3.3
7502000p 0
```

Once the PWL files are generated Verilog-A can be used to create software voltage sources that are useable by the simulator used for testing the IC. The Verilog-A test fixture also creates the exponential pulses from the analog voltages created in the System Verilog driver, and records timing data to a file.

4.6.2 Power dissipation

Power usage will vary depending on if the chip is idle or being hit by a pulse. For this reason it is worthwhile to look at both average power and peak power dissipation in the IC. Figure 4.12 shows a breakdown of the average power usage of the various subcircuits

in the IC, while Figure 4.13 shows the breakdown peak power usage that occurs when 16 input pulses arrive at the same time. The IC has a peak power usage for the core of ≈360 mW (very briefly during output generation) and an average power usage of ≈220 mW.

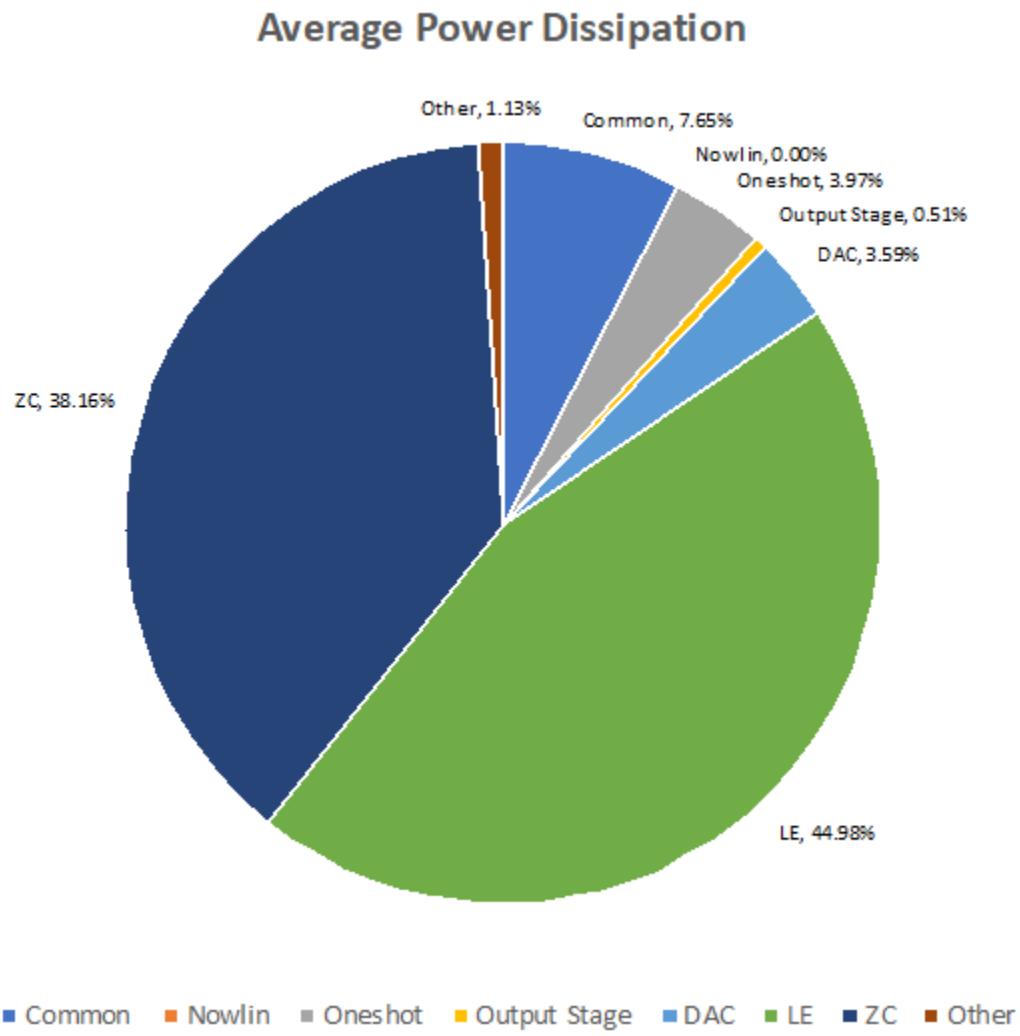


Figure 4.12: Breakdown of average power dissipation

4.6.3 Chip area usage

Chip area usage needed to be minimized due to the cost of working in this 0.35 micron NWELL process. Figure 4.14 shows how the area was utilized in the IC. Figure 4.15

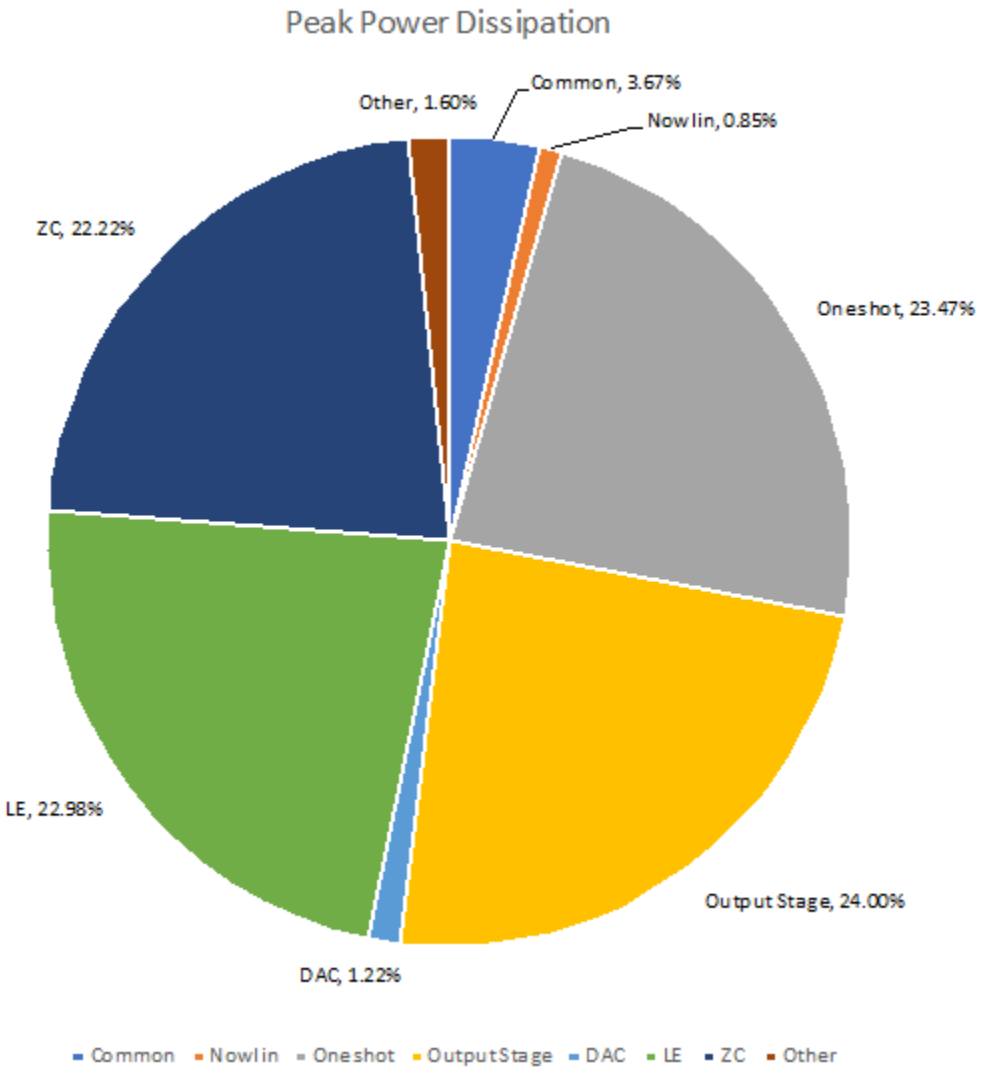


Figure 4.13: Breakdown of peak power dissipation

shows the full IC layout. The entire chip die uses an area of 2.4 mm x 3.5 mm giving a total area usage of 8.4 mm^2, only slightly larger than the 8 mm^2 target.

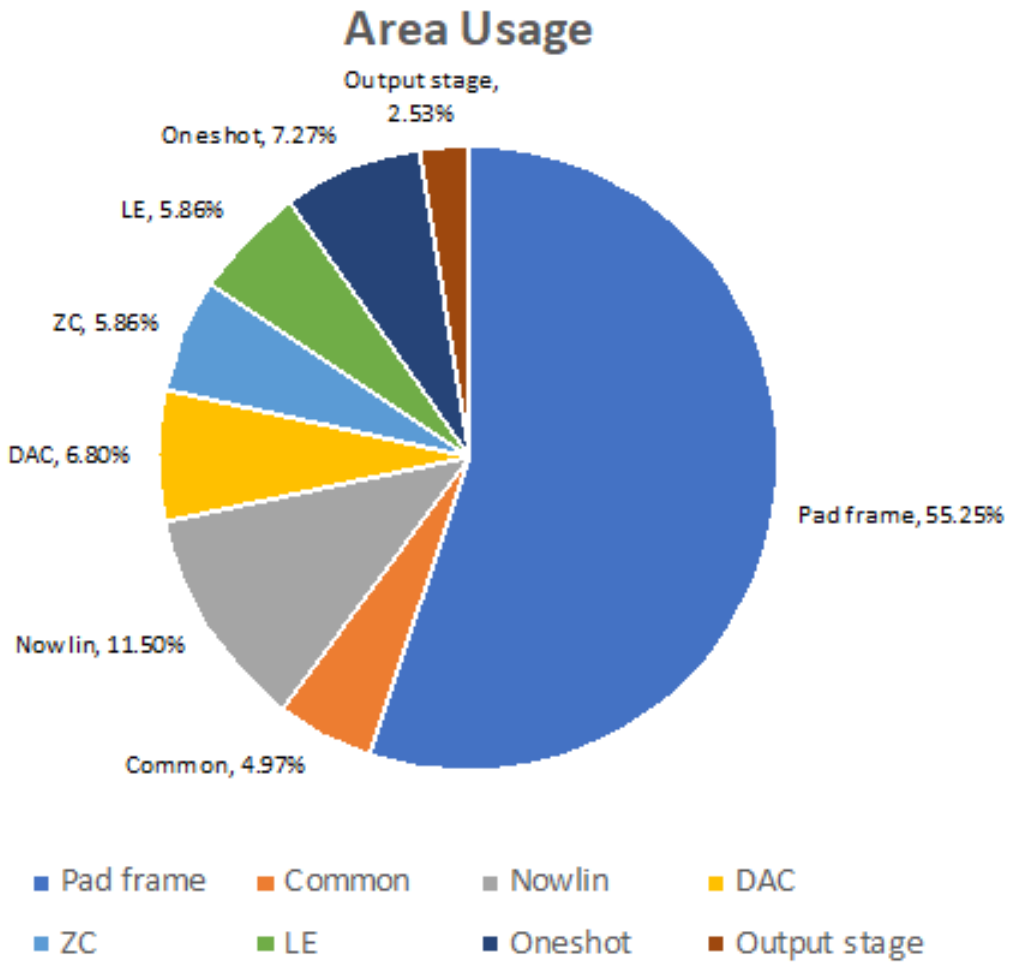


Figure 4.14: Breakdown of IC area usage

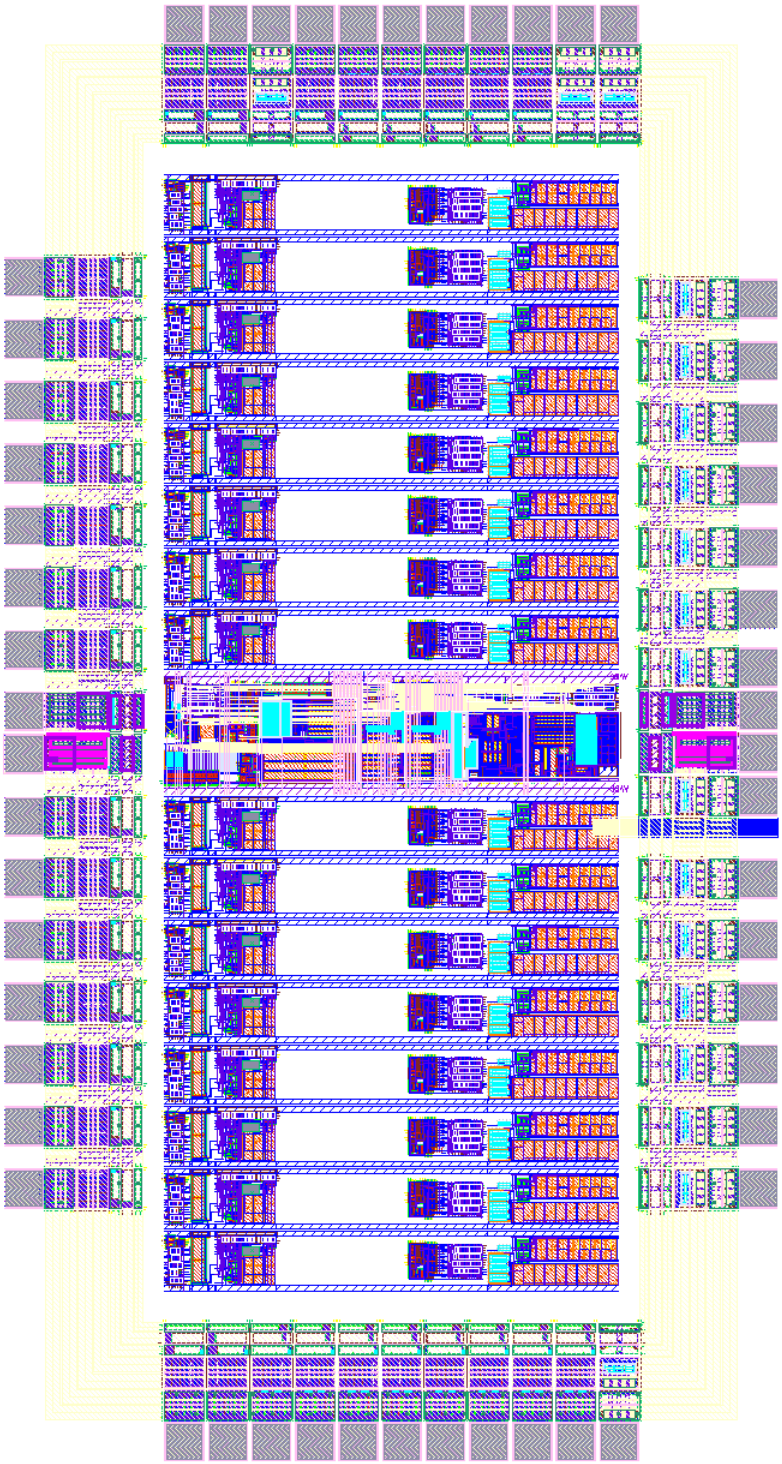


Figure 4.15: CFD16C full layout

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 (falling edge) and gated integrators (rising edge) on the PSD8C. The chip is highly configurable, allowing the following circuit parameters to be adjusted to an experiments needs:

- Timing pulse width control with four choices: 50, 100, 200 and 500 nsec
- Leading edge threshold 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 voltage

The CFD16C contains sixteen signal channels allowing one chip to support two PSD8Cs. Each signal channel is comprised of an input Nowlin circuit with two modes. A short mode to provide support for input pulse time constants between 2 nsec and 16 nsec (1 nsec steps), and a long mode to cover a range from 12 nsec to 192 nsec (12 nsec steps). Next is a zero cross and leading edge discriminator with DC offset cancellation loops that can support pulse repetition rates of up to 1 kHz. Finally the signal channel has an output one-shot circuit used to create the timing pulse. This one-shot has lockout

capabilities with two modes. A short mode provides 31 selectable lockout times from 100 nsec to 1.6 μsec, and the long mode provides 31 selectable lockout times from 600 nsec to 16 μsec. Both lockout modes have lockout times that are inversely proportional to the 5-bit digital code word that controls them.

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 (100 mV per channel). Each signal channel can support input pulse amplitudes between 15 mV and 1.5V (both positive and negative). A single common channel is used to contain bias circuitry and chip configuration registers, using a single 8-bit wide bus for both data and adress.

Simulated performance indicates that the IC performs as expected. There is low variation (due to process and mismatch) in the falling edge of the timing pulse, being $\approx$10% for all one-shot pulse width modes. The jitter in the rising edge of the timing pulse is also very good, being around 8 psec for the 50 nsec wide pulse width mode, and 25 psec for the 500 nsec wide pulse width mode. The IC exhibits excellent walk characteristics, having a walk time of only $\approx$430 psec for input rise time constants of 3 nsec (and less than 8% of the risetime constant for the longest supported rise time constant). There is an average power dissipation of 220 mW and a peak power dissipation of 360 mW for the core of the IC. The IC occupies a die area of approximately 2.4 mm x 3.5 mm and will be fabricated in a 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 the performance simulations show it will meet or exceed all design specifications. The final 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. Smulations for the following must be completed:

- Disabling the internal signal ground generator and providing an external signal ground reference
- Increasing/decreasing the *AVDD* supply voltage and trimming the internal *AGND* to match
- Test each channel with a different leading edge threshold value
- Chip level simulations including parasitic extraction

Layout of the CFD circuits (i.e. leading edge and zero cross discriminators) needs to be completed, as well as the top level interconnect routing. 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 any discrepancies.

Finally, we are looking into adding capabilities of doing pulse shape discrimination using a time over threshold (TOT) method. With this technique, the PSD8C would not be needed as the 16 leading edge discriminator outputs would be brought out to pins on the chip package. Since we cannot add 16 more pins and still use the same 64 pin package, the 8-bit *ADDRDAT* bus would be converted to a bidirectional bus and a pin would be added to control the direction (input during chip configuration, output during an experiment). Doing this allows only 9 pins to be added (8 for LE outputs, 1

for direction control on the 8-bit bus). This technique would involve setting a very low leading edge threshold and measuring the width of the LE output pulse to determine how long the input pulse was above the threshold. Particles with longer decay times (e.g. neutrons) will how a longer LE pulse width while those with shorter decay times (e.g. gamma particles) will have a shorter pulse width. This was a feature proposed late in the design of the CFD16C and is currently being investigated.

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.

APPENDICES

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
```

APPENDICES

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
```

APPENDICES

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;
```

APPENDICES

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
```

APPENDICES

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
```

APPENDICES

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
```

APPENDICES

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 ""
```