



# **Calhoun: The NPS Institutional Archive**

# **DSpace Repository**

Theses and Dissertations

1. Thesis and Dissertation Collection, all items

2018-06

# IMPLEMENTATION OF THE FAST FOURIER TRANSFORM ONBOARD CFTP-7 SPACE EXPERIMENT

Walker, Alan A. III

Monterey, CA; Naval Postgraduate School

http://hdl.handle.net/10945/59613

Downloaded from NPS Archive: Calhoun



Calhoun is a project of the Dudley Knox Library at NPS, furthering the precepts and goals of open government and government transparency. All information contained herein has been approved for release by the NPS Public Affairs Officer.

> Dudley Knox Library / Naval Postgraduate School 411 Dyer Road / 1 University Circle Monterey, California USA 93943

http://www.nps.edu/library



# NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

# THESIS

# IMPLEMENTATION OF THE FAST FOURIER TRANSFORM ONBOARD CFTP-7 SPACE EXPERIMENT

by

Alan Walker

June 2018

Thesis Advisor: Co-Advisor: Herschel H. Loomis James H. Newman

Approved for public release. Distribution is unlimited.

REPORT	DOCUMENTATION PAGE			rm Approved OMB No. 0704-0188		
instruction, searching existing dat information. Send comments r suggestions for reducing this burg	bllection of information is estimated to a sources, gathering and maintaining th egarding this burden estimate or an den, to Washington headquarters Servio 204, Arlington, VA 22202-4302, and to DC 20503.	e data needed, and c y other aspect of t ces, Directorate for I	ompleting an his collection nformation C	d reviewing the collection of n of information, including Operations and Reports, 1215		
1. AGENCY USE ONLY (Leave blank)	<b>2. REPORT DATE</b> June 2018	3. REPORT TY	TYPE AND DATES COVERED Master's thesis			
CFTP-7 SPACE EXPERIMENT	E FAST FOURIER TRANSFORM	I ONBOARD	5. FUNDI	ING NUMBERS		
6. AUTHOR(S) Alan Walker						
7. PERFORMING ORGAN Naval Postgraduate School Monterey, CA 93943-5000	ZATION NAME(S) AND ADDF	RESS(ES)	8. PERFO ORGANI NUMBEI	ZATION REPORT		
9. SPONSORING / MONITO ADDRESS(ES) N/A	D	MONITO	ISORING / DRING AGENCY NUMBER			
	<b>TES</b> The views expressed in this t the Department of Defense or the U.		he author ar	nd do not reflect the		
<b>12a. DISTRIBUTION / AVA</b> Approved for public release.			12b. DIST	<b>FRIBUTION CODE</b> A		
(CFTP) was designed at the Gate Array (FPGA), which satellite. Experimentation of conducted for use within the pipelined FFT in which far utilized to synthesize behavior	as a testbed for experiments s as a testbed for experiments s be Naval Postgraduate School. h may be reprogrammed by p of a high-speed pipelined and he CFTP. In this thesis, we de ault tolerance can be applied avioral Verilog to program an nonstrate functionality. Launch	This processor of receiving a signa fault tolerant Fa etail the develop at a later opport FPGA. Xilinx	consists of al from a ust Fourier ment and t cunity. Xili Vivado IS	a Field Programmable source external to the Transform (FFT) was testing of a high-speed inx Vivado ISE® was SE's® simulation suite		
<b>14. SUBJECT TERMS</b> FPGA, NPSat-1, satellite, faul MidStar-1, CFTP-1, CFTP-7,	t tolerance, FFT, DFT, reprogramn Parseval's theorem	nable computers, C	EFTP,	15. NUMBER OF PAGES 117		
· · · · · · · · · · · · · · · · · · ·			ŀ	16. PRICE CODE		
<b>17. SECURITY</b> <b>CLASSIFICATION OF</b> <b>REPORT</b> Unclassified	<b>18. SECURITY</b> <b>CLASSIFICATION OF THIS</b> <b>PAGE</b> Unclassified	<b>19. SECURITY</b> <b>CLASSIFICAT</b> <b>ABSTRACT</b> Unclassified	ION OF	20. LIMITATION OF ABSTRACT UU		

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. 239-18

#### Approved for public release. Distribution is unlimited.

### IMPLEMENTATION OF THE FAST FOURIER TRANSFORM ONBOARD CFTP-7 SPACE EXPERIMENT

Alan A. Walker III Lieutenant, United States Navy BS, U.S. Naval Academy, 2009

Submitted in partial fulfillment of the requirements for the degree of

#### MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

### NAVAL POSTGRADUATE SCHOOL June 2018

Approved by: Herschel H. Loomis Advisor

> James H. Newman Co-Advisor

R. Clark Robertson Chair, Department of Electrical and Computer Engineering

# ABSTRACT

A satellite to be used as a testbed for experiments such as the Configurable Fault Tolerant Processor (CFTP) was designed at the Naval Postgraduate School. This processor consists of a Field Programmable Gate Array (FPGA), which may be reprogrammed by receiving a signal from a source external to the satellite. Experimentation of a high-speed pipelined and fault tolerant Fast Fourier Transform (FFT) was conducted for use within the CFTP. In this thesis, we detail the development and testing of a high-speed pipelined FFT in which fault tolerance can be applied at a later opportunity. Xilinx Vivado ISE® was utilized to synthesize behavioral Verilog to program an FPGA. Xilinx Vivado ISE's® simulation suite produced waveforms to demonstrate functionality. Launch of CFTP is planned for FY18 aboard NPSat-1.

# TABLE OF CONTENTS

I.	INT	RODUCTION	.1
	А.	OBJECTIVES	.1
	В.	DEFINING THE PROBLEM	.3
	C.	ORGANIZATION	.5
	D.	ADDITIONAL DOCUMENTATION	.5
II.	BAC	KGROUND AND PRIOR WORK	.7
	А.	CFTP HISTORY	.7
		1. Midshipmen Space Technology Applications Research-1	.7
		2. Naval Postgraduate School Satellite-1	.8
	В.	LITERATURE REVIEW	.9
		1. Academic Advisors	9
		2. Research History1	.0
III.	DES	IGN AND IMPLEMENTATION1	3
	A.	SYSTEM SPECIFICATIONS1	3
		1. Two's Complement1	3
		2. Fixed-Point1	4
		3. Complex Numbers1	5
	В.	FAST FOURIER TRANSFORM ALGORITHMS1	.6
		1. Bit Reversal1	.6
		2. The 8-Point DFT1	7
	C.	HIGH-SPEED PIPELINED FFT ARCHITECTURE1	8
		1. Radix-2 Pipeline Butterfly Machine Architecture1	9
		2. 2-Stage Pipelined Complex Multiplier Architecture2	:0
IV.	TES	TING AND EVALUATION2	3
	A.	TESTING PLAN2	3
		1. System Test Plan2	3
		2. Component Test Plan2	4
		3. Component Timing2	25
	B.	TEST INPUTS2	7
	C.	FFT SYSTEM AND COMPONENT TESTING2	8
		1. Bit Reversed Ping-Pong Buffer2	9
		2. Radix-2 Pipeline Butterfly Machine Sub-Component	
		Testing3	3

V.	ENI	) TO END TESTING / INTEGRATION TESTING	51
	A.	ALL TEST VECTOR ANALYSIS	51
	B.	TEST VECTOR ANALYSIS	55
		1. Bit Reversed Ping Pong Buffer	55
		2. First-Stage BFM and Ping Pong Buffer	
		3. Second-Stage BFM and Ping-Pong Buffer	
		4. Third-Stage BFM and Ping-Pong Buffer	
	C.	TIMING ERROR WHILE INTEGRATING	
VI.	CON	NCLUSION	63
	A.	ACADEMIC VALUE	
	B.	THESIS SUMMARY	
	C.	<b>RECOMMENDATIONS FOR FUTURE WORK</b>	63
	D.	CLOSING REMARKS	
APP	ENDIX	X A. FILE STRUCTURE	65
APP	ENDIX	K B. SOURCE CODE	67
APP	ENDIX	X C. HARDWARE CONSTRAINTS FILE	91
LIST	r of r	EFERENCES	95
INIT	TAL D	DISTRIBUTION LIST	97

# LIST OF FIGURES

Figure 1.	Implementation of Parseval's Theorem with Two Duplicate FFTs. Source: [2]
Figure 2.	External View of MidStar-1 Showing the Location of CFTP-1. Adapted from [6]8
Figure 3.	Expanded View of NPSat-1 That Shows Location of CFTP-7. Adapted from [7]9
Figure 4.	Demonstration of Two's Complement Operation14
Figure 5.	Fixed-Point Signed Binary Representation. Binary Point Place After Bit 014
Figure 6.	Flow Graph of Simplified Butterfly Machine Computation Requiring Only One Complex Multiplication. Source: [4]
Figure 7.	Bit Reversed Demonstration17
Figure 8.	Flow Graph of 8-point DFT Using the Butterfly Machine Computation. Source: [4]17
Figure 9.	Rearrangement of Figure 1 to Allow Each Stage to Have a Constant Geometry Permitting Sequential Data Accessing and Storage. Source: [4]
Figure 10.	Basic Pipelined FFT Structure. $N = 2^{1}$ , Where N Is the FFT Word Size. Adapted from [15]
Figure 11.	Detailed High Speed Pipelined FFT Structure. Adapted from [15]19
Figure 12.	Pipeline Radix-2 Butterfly Machine (BFM) Architecture, Level <i>q</i> . Adapted from [15]20
Figure 13.	Two-Stage Complex Multiplier21
Figure 14.	System Level Test Plan for $N = 8$ FFT24
Figure 15.	Component Level Test Plan for $N = 8$ BFM. Adapted from [15]25
Figure 16.	Butterfly Machine Timing Diagram Part A. Adapted from [15]26
Figure 17.	Butterfly Timing Machine Diagram Part B. Adapted from [15]

Figure 18.	Injected $X_q(t)$ Inputs in Binary	27
Figure 19.	Bit-Reversed Ping-Pong Buffer Variables and Arrays	29
Figure 20.	Bit-Reversed Ping-Pong Buffer Loading, Code Snippet	30
Figure 21.	Bit-Reversed Ping-Pong Buffer Output $X_q(t)$	31
Figure 22.	Variable Indexbr transposed from Counter[2:0]	31
Figure 23.	Pong Buffer Loading in Bit Reversed Order, Ping Empty	32
Figure 24.	Startup Delay Between Bit-Reversed Ping-Pong Buffer Output and BFM	32
Figure 25.	Ping-Buffer Loading in Bit Reversed Order, Pong Full	33
Figure 26.	Butterfly Machine Variable Declaration Code Snippet	33
Figure 27.	"Top" and "Bottom" Multiplexer Code Snippet	34
Figure 28.	Delay into Both Multiplexer Code Snippet	34
Figure 29.	Internal Multiplexer Code Snippet	35
Figure 30.	Top Multiplexer Half Scale	35
Figure 31.	Top Multiplexer Three-Clock-Cycle Delay	36
Figure 32.	Top Multiplexer Output, Hexadecimal	36
Figure 33.	Top Multiplexer Output, Fixed-Point Binary	37
Figure 34.	Top Multiplexer Output, Binary	37
Figure 35.	Multiplexer Internal Code Snippet	37
Figure 36.	Bottom Multiplexer Half-scale Code Snippet	38
Figure 37.	Bottom Multiplexer Output with Multiplier as a Black Box	38
Figure 38.	Test Plan for Multiplier and Sub-Component testing	39
Figure 39.	Multiplier Variables Code Snippet	40
Figure 40.	Code Snippet for <i>twoComp</i> Module Call	40
Figure 41.	Code Snippet for <i>twoComp</i> Module Internals	41

Figure 42.	Sign Bit Extraction Code Snippet	41
Figure 43.	Module Call for Behavioral Multiplication on Four Unsigned Complex Numbers Code Snippet	42
Figure 44.	Unsigned Multiplication Module Code Snippet	42
Figure 45.	Behavioral Multiplication Code Snippet	43
Figure 46.	Output Truncation Code Snippet	43
Figure 47.	Exclusive-Or of Sign-Bit Code Snippet	44
Figure 48.	Code Snippet for twoCompRedo Module Call	44
Figure 49.	Module twoCompRedo Code Snippet	44
Figure 50.	Adder/Subtracter for Complex Multiplication Code Snippet	45
Figure 51.	Complex Multiplication Module Testing	45
Figure 52.	Conversion of Multiplier Inputs to Signed Magnitudes	46
Figure 53.	Multiplier Test Points. Fixed Point	47
Figure 54.	Adder / Subtracter Variables Code Snippet	47
Figure 55.	Adder / Subtracter Code Snippet	48
Figure 56.	Adder / Subtracter Adds on Even Clock-Cycles	48
Figure 57.	First-Stage Ping-Pong Buffer Variables Code Snippet	48
Figure 58.	First-Stage Ping Buffer Input Code Snippet	49
Figure 59.	First-Stage Ping Buffer Input Code Snippet (Continued)	49
Figure 60.	First-Stage Ping-Pong Buffer Simulation	50
Figure 61.	Generic Constant Geometric FFT. Adapted from [4]	52
Figure 62.	Test Vector-1 {1, 1, 1, 1, 1, 1, 1}. Adapted from [4]	53
Figure 63.	Test Vector-2 {1, 0, 0, 0, 0, 0, 0, 0}. Adapted from [4]	53
Figure 64.	Test Vector-3 {0, 0, 0, 0, 0, 0, 0, 0}. Adapted from [4]	54

Figure 65.	Test Vector-4 $\{0, 0, 0, 1, 0, 0, 0, 0\}$ Displayed after Bit-Reversed to $\{0, 0, 0, 0, 0, 0, 1, 0\}$ . Adapted from [4]	55
Figure 66.	Bit-Reversed Ping-Pong Buffer Initializing	56
Figure 67.	Bit-Reversed Ping-Pong Buffer Output	56
Figure 68.	First-Stage BFM Output and First-Stage Ping-Pong Buffer Initialization	57
Figure 69.	First-Stage Ping-Pong Buffer Outputting and Second-Stage BFM Initializing	58
Figure 70.	Second-Stage BFM Outputting and Second-Stage Ping-Pong Buffer Initializing	58
Figure 71.	Second-Stage Ping-Pong Buffer Outputting and Third-Stage BFM Initializing	59
Figure 72.	Third-Stage BFM Outputting and Third-Stage Ping-Pong Initializing	60
Figure 73.	Third-Stage Ping-Pong Buffer With Final Scaled Result {1, 0, 0, 0, 0 0, 0, 0}	60
Figure 74.	Highlighted Timing Variables within the BFM. Adapted from [15]	61
Figure 75.	Parseval's Theorem Implementation Illustration. Source: [2]	64

# LIST OF TABLES

Table 1.	State Diagram for Parseval's Theorem Implemented with Two Redundant FFTs as Pictured in Figure 1	3
Table 2.	Comparison of DFT Calculation Size to FFT Calculation Size to Demonstrate Efficiency of Algorithm	4
Table 3.	18-Bit Fixed-Point Binary Number with a 16-bit Radix Point Examples	15
Table 4.	$X_q(t)$ Inputs Injected into FFT	28
Table 5.	Test Vector Input and Expected Output	51

# LIST OF ACRONYMS AND ABBREVIATIONS

BFM	Butterfly machine
CFTP	Configurable Fault Tolerant Processor
COTS	commercial off-the-shelf
DFT	discrete Fourier transform
DSP	digital signal processing
FFT	fast Fourier transform
FPGA	field-programmable gate array
HDL	Hardware Description Language
LTE	Long-Term Evolution
MidSTAR-1	Midshipman Space Technology Applications Research-1
MSB	most significant bit
MUX	multiplexer
NPS	Naval Postgraduate School
NPSAT-1	Naval Postgraduate School Satellite-1
OFDM	orthogonal frequency-division multiplexer
RF	radio frequency
RPR	reduced precision redundancy
SDR	software defined radio
SEU	single-event upset
TMR	triple-modular redundancy
VHDL	VHSIC Hardware Description Language
WiMAX	Worldwide Interoperability for Microwave Access

## ACKNOWLEDGMENTS

I thank the dedicated faculty and staff of the Naval Postgraduate School. I give a special thank you to my primary thesis advisor, Professor Herschel Loomis, as he spent hours coaching me through code development and explaining the historical context. I am fortunate that I had the opportunity to take part in a project that has been developing for over 20 years, culminating Dr. Loomis's career with one final launch and a retirement.

Thank you to my parents, Alan and Claudette Walker, and to my resource sponsor. My time here has included some of my most enjoyable experiences while serving within the United States Navy. To all who may come after, continue to believe in yourself, set goals, get mentors, dream, have fun, and always enjoy the moment.

# I. INTRODUCTION

The Naval Postgraduate School (NPS) worked with government sponsors to design and build a small satellite that is scheduled to be launched into space in June 2018. Because of radiation effects, computer systems that are utilized in space must be able to detect and/ or correct digital bit errors. The Configurable Fault-Tolerant Processor (CFTP) experiment is a field-programmable gate array (FPGA) implementation of a digital processor to test means for correcting radiation-induced faults in digital processors.

In 2005, Coudeyras [1] tested and demonstrated that radiation in the space environment can cause single-event upsets (SEU) that can have unknown, sometimes unrecoverable, effects on electrical systems. A fast Fourier transform (FFT) that can be used to implement Parseval's theorem in a way that corrects single bit errors [2] is designed and will be implemented on the CFTP as a space experiment on the Naval Postgraduate School Satellite One (NPSAT-1). This implementation will detect and correct SEUs caused by radiation in space.

Specifically, in this thesis, we detail the development and testing of a high speed pipelined FFT. Caleb Humberd performed similar research during his time at NPS in 2011 [2]. His research differed in that he implemented a proprietary Xilinx Integrated Synthesis Environment® (ISE) designed FFT that was based on proprietary intellectual property [2]. The FFT code developed in this thesis is open source, easily modified, and documented, which has important pedagogical utility. Sample data was processed through the FFT code to verify functionality.

#### A. **OBJECTIVES**

In thesis, we expand upon the engineering of the fault-tolerant FFT design, implementation, and execution. The FFT can be utilized to compress radio signals to reduce buffer memory and downlink bandwidth. Use of Parseval's theorem enables the FFT to perform error detection and correction as shown in Figure 1, increasing the reliability of the signal.

A sequence x[n] flows into identical and parallel FFT A and FFT B, and the power is calculated in the frequency-domain. In addition, the sequence x[n] has the power calculated from the time-domain. Parseval's theorem states that the FFT's frequencydomain power must equal the time-domain power. The output signal defaults to FFT A unless the FFT A power differs from the reference time-domain power by a threshold deviation. If the threshold deviation is detected, FFT B becomes the output signal. Detailed within Table 1 are the output options that may result based on the results of implementing Parseval's theorem. This will eventually be an experiment on the memory of the CFTP-1 at launch.

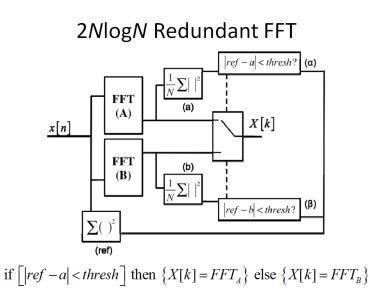


Figure 1. Implementation of Parseval's Theorem with Two Duplicate FFTs. Source: [2].

ref	FFT (A)	FFT (B)	(a)	(b)	ref – a	<b>ref</b> – b	Output
							X[k]
Correct	Correct	Correct	Correct	Correct	Zero	Zero	FFT (A)
Correct	Correct	Incorrect	Correct	Incorrect	Zero	Not Zero	FFT (A)
Correct	Incorrect	Correct	Incorrect	Correct	Not Zero	Zero	FFT (B)
Correct	Incorrect	Incorrect	Incorrect	Incorrect	Not Zero	Not Zero	FFT (A)
Incorrect	Correct	Correct	Correct	Correct	Not Zero	Not Zero	FFT (A)
Incorrect	Correct	Incorrect	Correct	Incorrect	Not Zero	Not Zero	FFT (A)
Incorrect	Incorrect	Correct	Incorrect	Correct	Not Zero	Not Zero	FFT (B)
Incorrect	Incorrect	Incorrect	Incorrect	Incorrect	Not Zero	Not Zero	FFT (A)

Table 1. State Diagram for Parseval's Theorem Implemented with Two Redundant FFTs as Pictured in Figure 1.

In this thesis, a behavioral Verilog definition of a pipeline eight-point FFT was developed, simulated, and implemented in a Xilinx Kintex-7 FPGA. This FFT is the basic element for a test of the Parseval's theorem-protected SEU-tolerant FFT.

#### **B. DEFINING THE PROBLEM**

The discrete Fourier transform (DFT) is an integral component in digital signal processing (DSP). Understanding signals is important due to their prevalence everywhere.

Social communications between people, physical communications between people and machines, or machine to machine communications are done through signals. Signals present themselves in nature as continuous-time analog quantities. Once digitized and converted by sampling to discrete time they become sequences. A sequence is "a continuous or connected series such as a set of elements ordered so that they can be labeled with positive integers" [3]. Discrete-time samples are roughly comparable to digital signals and are treated the same in our case. The system produces outputs at the same rate at which the continuous signal is sampled, producing a real-time system. The Fourier transform of the sampled signal gives the frequency-domain representation of the time-domain sequence. The DFT is a representation of the Fourier transform that is bounded in time and is defined by the Fourier-transform-pair [4]

$$X_{k} = \sum_{n=0}^{N-1} x_{n} e^{-j\left(\frac{2\pi}{N}\right)kn} = \sum_{n=0}^{N-1} x_{n} W_{N}^{kn}$$
$$x_{n} = \frac{1}{N} \sum_{n=0}^{N-1} X_{k} e^{j\left(\frac{2\pi}{N}\right)kn} = \sum_{n=0}^{N-1} X_{k} W_{N}^{-kn}$$
(1)

where  $W_N^{kn} = e^{-j(2\pi/N)kn}$  is known as a twiddle factor,  $X_k$  is the calculated time to frequency,  $x_n$  is the calculate frequency to time, n is the current sample, and k is a constant.

The DFT calculation requires  $N^2$  calculations where *N* is the sample size. The DFT is one of the most important equations for DSP. It is useful in an orthogonal frequencydivision multiplexer (OFDM) demodulators and modulators. Long-Term Evolution (LTE) signal processing, WIFI, and Worldwide Interoperability for Microwave Access (WiMax) signals are examples of signals that the DFT is utilized to analyze. Efficient algorithms have been developed to calculate the DFT. The fast Fourier transform is the signal processing algorithm that is used to reproduce to perform signal processing in this research. The FFT requires  $N \log_2 N$  multiply-add operations [4]. To demonstrate the efficiency of the FFT over the DFT, Table 2 is provided.

	Ν	1000	10 <sup>6</sup>	10 <sup>9</sup>	10 <sup>12</sup>
DFT Calculation	$N^2$	10 <sup>6</sup>	10 <sup>12</sup>	10 <sup>18</sup>	10 <sup>24</sup>
FFT Calculation	Nlog <sub>2</sub> N	10 <sup>4</sup>	20 × 10 <sup>6</sup>	30 × 10 <sup>9</sup>	$40 \times 10^{12}$

Table 2. Comparison of DFT Calculation Size to FFT Calculation Size toDemonstrate Efficiency of Algorithm

The FFT is loaded into an FPGA as part of the Configurable Fault Tolerant Processor (CFTP) for operation within the space environment. Since electronics in space must be able to produce the correct calculations in spite of the risk of SEUs, protection of the FFT is accomplished by implementing Parseval's theorem.

## C. ORGANIZATION

Background on Configurable Fault Tolerant Processor research performed at NPS is covered in Chapter II. First, an introduction of Academic Advisors that have facilitated the research is given, following with a review of literature. In Chapter III, we discuss the implementation of the FFT. In addition, the design methodology of the FFT is also included in Chapter III. In Chapter IV, we discuss the test vectors that were utilized to confirm proper operation of the FFT. In Chapter V, we summarize and draw conclusions from the thesis research as well as provide recommendations for future work.

#### D. ADDITIONAL DOCUMENTATION

The Verilog source code for the FFT is included in appendixes.

## II. BACKGROUND AND PRIOR WORK

The United States Navy (USN), United States Air Force (USAF), and Department of Defense (DoD) agencies have been designing, purchasing, manufacturing, launching, and operating space systems since 1957. These systems are designed and operated to support the warfighter and strategic decision makers. The acquisition life cycle time and cost for a constellation of proprietary and military specification satellites is greater than DoD leadership likes. In addition, space and the aerospace industry have been commercialized enough to allow the U.S. government to decrease risk and cost by purchasing entire systems or constructing custom systems utilizing commercial-off-theshelf (COTS) components. The utilization of an FPGA to perform signal processing is becoming routine. The use of a reconfigurable processor, an antenna, a demodulator, an analog-to-digital convertor, and a modulator can be utilized as one system named a software-defined radio (SDR). In short, a SDR is a reconfigurable signal processor.

The reprogrammable nature of an SDR makes it convenient for utilization in space. Satellite operators and engineers now have the ability to add or change capability by uploading software while a satellite is on orbit as long as the receiving antenna, analog-todigital converter, FPGA, and downlink antenna are installed prior to launch. The control module also needs a connection to load the FPGA.

Space is a challenging environment for electronics to operate. The space environment can cause digital bits to flip, thus being read in error. Coudeyras confirmed this through research on FPGAs at Crooker Research Laboratory [1]. Prior research has been performed to allow an FPGA to detect errors and/or correct errors. Triple modular redundancy (TMR) and reduced precision redundancy (RPR) are two techniques that accomplish error correction. This is discussed in the literature review.

#### A. CFTP HISTORY

#### 1. Midshipmen Space Technology Applications Research-1

The U.S. Naval Academy designed Midshipmen Space Technology Application Research-1 (MidSTAR) to incorporate a test bed for fault-tolerant techniques applied to FPGAs developed by NPS. This consisted of two FPGAs, which were called the Configurable Fault-Tolerant Processor (CFTP). In September 2006, the United States Air Force launched CFTP into Low Earth Orbit (LEO) onboard the host Space Test Program (STP-1) satellite. CFTP detected seven single-event upsets, mainly while flying through the south-Atlantic Anomaly, during its 492 km, 46 degree inclination orbit [1]. The location of CFTP is displayed in Figure 2.

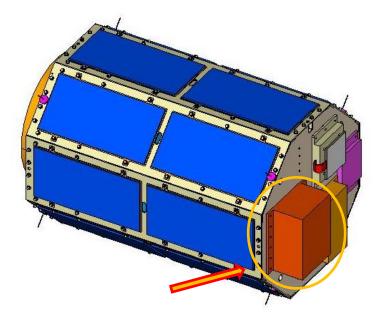


Figure 2. External View of MidStar-1 Showing the Location of CFTP-1. Adapted from [6].

#### 2. Naval Postgraduate School Satellite-1

NPS's effort on CFTP-7, which is currently in production, makes MidSTAR CFTP-1 obsolete. CFTP-7 is designed to be fault tolerant due to the use of TMR techniques. Its increased capability includes memory and processor improvements. The utilization of partial reconfigurations also increases capability. Four experiments will be preloaded into CFTP-7 at launch. This research project details one of the four experiments. Naval Postgraduate School Satellite-1 (NPSat-1) is scheduled to fly at 560 km at 35.4 degrees inclination and is scheduled to launch in FY18. An expanded view of the satellite is displayed in Figure 3.

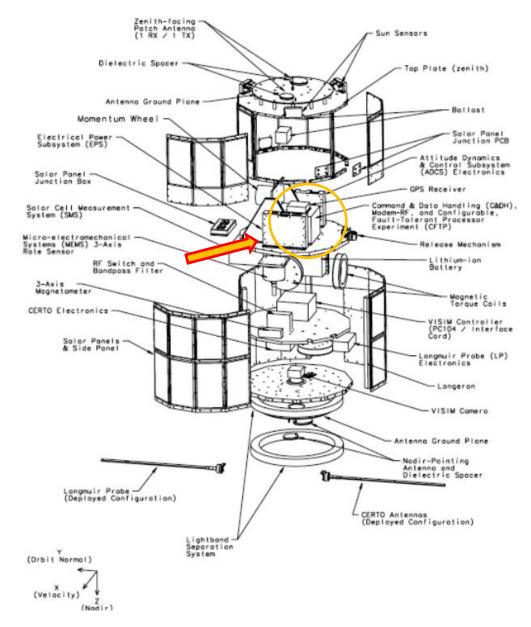


Figure 3. Expanded View of NPSat-1 That Shows Location of CFTP-7. Adapted from [7].

#### **B.** LITERATURE REVIEW

#### 1. Academic Advisors

Dr. Alan A. Ross and Dr. Herschel H. Loomis have been the primary advisors for the CFTP project, completing a combined total of twenty theses and three dissertations. Dr. Ross (Lt. Col., USAF, retired) earned his PhD from the University of California, Davis, in 1978 and recently retired as a professor in computer engineering. Dr. Loomis earned his PhD from Massachusetts Institute of Technology (MIT) in 1963. Dr. James H. Newman, chair, Space Systems Academic Group (SSAG), has participated as an advisor recently on CFTP projects. He earned his PhD from Rice University in 1984 and served as an astronaut from 1990–2008.

#### 2. Research History

In June 2003, Dean Ebert researched the design trade offs for CFTP-1's initial concepts. In 2005, James Coudeyras completed his research in partnership with the Crocker Nuclear Laboratory in Davis, California in which he executes his radiation test plan utilizing their proton radiation beam [1]. He proved that space radiation has an impact on electronics, and these errors are known as SEUs. Also in December 2005, Peter Majewicz completed his research of a fault-tolerance technique called Triple Modular Redundancy. TMR instantiates three modules in parallel. They utilize a majority voter to correct errors in, at most, one component [8]. In December 2006, Gerald Caldwell completed his research on the design challenges present while utilizing two FPGAs [9]. In September 2008, David Dwiggins, Jr., redesigned the X1 control FPGA to be fault tolerant and added a microcontroller to manage internal components [10].

In December 2008, Margaret Sullivan completed her research implementing and analyzing a new method of fault tolerance called Reduced Precision Redundancy (RPR) [11]. She concluded that "RPR provided very good recovery from errors caused by SEU in spacecraft systems" [11]. To be clear, RPR protects a satellite's arithmetic module from SEU just like TMR. In addition, RPR has a lower power cost than TMR. RPR was developed by PhD student Josh Snodgrass in September 2006. He performed research on fault tolerance by means of reducing the precision of the redundant copies of a precise number used for error detection and correction. He named this method RPR. RPR applies only to arithmetic operations, and he proved this technique as viable using live proton radiation testing [12].

In December 2009, Jeremy Livingston completed his design to compress a wideband radio signal into a narrowband signal [13]. In December 2011, Caleb Humberd

completed his research on a FFT based compression algorithm and discovered a way to use Parseval's theorem to correct single-component errors in an FFT. In March 2016, Andrew Jackson completed his design of a TMR embedded MIPS processor architecture with a majority-output voter to combat single-event upsets for NPSat-1 on orbit [14].

The most important prior works directly related to this thesis are a thesis written by Michael Zimmer and a dissertation by Raymond Bernstein. Zimmer designed a radix-4 FFT that operated at 45 MHz, had a floating point multiplier and adder, and consisted of 20-bit words [15]. Bernstein utilized a FFT to design a vector-processing computer. He discovered that the structure of the memory system for a vector-based computer favored the butterfly machine (BFM) operation [3]. The BFM is a visible representation of the FFT computation and is described in detail in Chapter III.

The FFT described in this thesis is a product of 20 years of space and computer research. As we continue to operate complex electronics in space, the understanding of the impact of space radiation, how to mitigate those impacts in a cost effective manner, and to continue to improve every aspect of a computer system means research like this will continue to build on itself. In Chapter III, the FFT development and design choices are presented and matured.

## **III. DESIGN AND IMPLEMENTATION**

Chapter III is organized into three major sections. Specifications are given in Section A for the end system. The FFT is described from a mathematical perspective in Section B. The architecture that must be implemented to realize the specified FFT is detailed in Section C. This algorithm was coded in the Verilog design language utilizing the Xilinx Vivado<sup>®</sup> design suite. It builds on known block diagrams and structures to realize the code.

#### A. SYSTEM SPECIFICATIONS

The FFT design consists of the 18-bit signed two's complement number system, a fixed point rational number representation with the binary point between the  $16^{\text{th}}$  and  $17^{\text{th}}$  bits. In addition, two 18-bit words representing the real and imagery part of a complex number system are utilized. Sample size *N* is equal to 8. Sub-sections are organized to elaborate on these details.

#### 1. Two's Complement

Two's complement is a popular number system because it allows for the representation of negative numbers within a binary adder which may perform addition or subtraction. For an N-bit word, the range of values can be represented as  $-2^{N-1}$  to  $2^{N-1}$ -1.Commonly, the most significant bit (MSB) is utilized to determine if the number is positive or negative. A "0" in the MSB represents a positive number, and a "1" in the MSB represents a negative number. The value of an N-bit word when utilizing two's complement can be determined as [15]

$$V_{2's} = (-b_{N-1})(2^{N-1}) + \sum_{i=0}^{N-2} b_i 2^i$$
(2)

A simpler way to compute the two's complement of a number is to invert all bits in a binary number and then add 1. This converts a positive number to a negative number representation. An example of how this operation works is shown in Figure 4. 0101 (5) 1010 + 1 1011 (-5)

Figure 4. Demonstration of Two's Complement Operation

#### 2. Fixed-Point

When utilizing a fixed-point rational number representation, the binary point is established by the user to satisfy a design goal. Unlike the integer representation where, the binary point is to the right of the least-significant bit (LSB) [15], the binary point is located anywhere to the left of the LSB. Within this design, the binary point is after bit 0 as shown in Figure 5. This bounds values to be less than two and equal to or greater than negative two.

s 01 -2 -3	-4 -5	-6 -7 -8	-9 -10 -11	-12 -13 -14	-15 -16
------------	-------	----------	------------	-------------	---------

Figure 5. Fixed-Point Signed Binary Representation. Binary Point Place After Bit 0

The formula to determine a value when a fixed point two's complement representation is used is given by

$$V_{FixedPoint} = (-b_0) (2^{\overline{N}-1}) + \sum_{i=1}^{\overline{N}-2} b_i 2^{\overline{N}-1-i}, \qquad b_i \in \{0,1\},$$
(3)

where  $b_i$  represents the value of the *i* numbered bit,  $b_0$  represents the value of the 0<sup>th</sup> bit,  $\overline{N}$  represents the total number of bits. Examples of equivalent radixes are shown in Table 3.

Binary	Hex	Integer	Fixed-Point	Actual	Exponential
			Binary		
0111111111111111111	1FFFF	131071	1.99998474121	<2	-
0100000000000000000	10000	65536	1.00000000000	1	2 <sup>1</sup>
0010000000000000000	8000	32768	0.50000000000	1/2	2 <sup>-1</sup>
0001000000000000000	4000	16384	0.25000000000	1/4	2 <sup>-2</sup>
000010000000000000	2000	8192	0.12500000000	1/8	2 <sup>-3</sup>
000001000000000000	1000	4096	0.06250000000	1/16	2 <sup>-4</sup>
00000100000000000	0800	2048	0.03125000000	1/32	2 <sup>-5</sup>
0000001000000000	0400	1024	0.01562500000	1/64	2 <sup>-6</sup>
0000000100000000	0200	512	0.00781250000	1/128	2 <sup>-7</sup>
0000000010000000	0100	256	0.00390625000	1/256	2 <sup>-8</sup>
0000000001000000	0080	128	0.00195312000	1/512	2 <sup>-9</sup>
00000000001000000	0040	64	0.00097656000	1/1024	2 <sup>-10</sup>
00000000000100000	0020	32	0.00048828125	1/2048	2 <sup>-11</sup>
00000000000010000	0010	16	0.00024414062	1/4096	2 <sup>-12</sup>
00000000000001000	0008	8	0.00012207031	1/8192	2 <sup>-13</sup>
00000000000000100	0004	4	0.00006103515	1/16384	2 <sup>-14</sup>
000000000000000000000000000000000000000	0002	2	0.00003051757	1/32768	2 <sup>-15</sup>
000000000000000000000000000000000000000	0001	1	0.00001525878	1/65536	2 <sup>-16</sup>
0001010101010101010	2AAB	10923	0.33334351	.33333	-
001011010011111110	5A7F	23167	.70700073	$\sqrt{2}/2$	-
000000000000010100	000A	10	.00030518	π	2 <sup>-14</sup>
					$+ 2^{-16}$

Table 3. 18-Bit Fixed-Point Binary Number with a 16-bit Radix Point Examples

## **3.** Complex Numbers

In implementing the arithmetic to compute the discrete Fourier transform, there is need to represent complex numbers. When transposing from the time domain to the frequency domain,  $W_N^{kn}$  introduces an imaginary component. A complex number is a + jbwhere *a* is the real component of a complex number, *jb* is the imaginary component of a complex number, and  $\sqrt{-1} = j$ . In performing complex multiplication, two complex numbers can be expanded to produce

$$(a+jb)(c+jd) = (ac-bd) + (ad+bc).$$
(4)

#### **B.** FAST FOURIER TRANSFORM ALGORITHMS

DSP progressed greatly when Cooley and Tukey discovered the FFT algorithm in 1965 [4]. The simplified BFM computation demonstrated in Figure 6 is a visual representation of the smallest element of a decomposed FFT calculation [4] and is in fact the two-point DFT. This element is used to construct the constant geometry decimation-intime algorithm this research implements [4]. The twiddle factor,  $W_N^{kn}$ , seen in equation (1), is within this diagram. Simply, the twiddle factor is a predictable multiplier needed to correctly calculate the FFT that is factor of sample size, time, and BFM stage. The twiddle  $W_N^{kn}$  can be expanded to  $e^{-j(2\pi/N)kn}$  [4]. In addition, odd samples are multiplied by the twiddle factor.

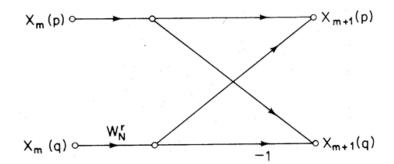


Figure 6. Flow Graph of Simplified Butterfly Machine Computation Requiring Only One Complex Multiplication. Source: [4].

## 1. Bit Reversal

The first step to utilizing the FFT within a DSP algorithm is to perform a bit reversal on the sample data. Bit reversal is a misnomer. Nothing is being done to the internal bits of the data; however, the data is being stored within alternating working buffers with the index transformed by reversing its bits as demonstrated in Figure 7.

$X_0[000] = x[000] = x_m[p]$	$X_0[0] = x[0]$
$X_0[001] = x[100] = x_m[q]$	$X_0[1] = x[4]$
$X_0[010] = x[010] = x_m[p]$	$X_0[2] = x[2]$
$X_0[011] = x[110] = x_m[q]$	$X_0[3] = x[6]$
$X_0[100] = x[001] = x_m[p]$	$X_0[4] = x[1]$
$X_0[101] = x[101] = x_m[q]$	$X_0[5] = x[5]$
$X_0[110] = x[011] = x_m[p]$	$X_0[6] = x[3]$
$X_0[111] = x[111] = x_m[q]$	$X_0[7] = x[7]$

Figure 7. Bit Reversed Demonstration

## 2. The 8-Point DFT

Once the equation for the FFT is decomposed into odd and even sets, Figure 6 can be utilized to create Figure 8 to yield an algorithm for an 8-point DFT, which shows the bit-reversal applied to the input sequence. This algorithm is detailed in depth within *Discrete-Time Signal Processing* by Oppenheim and Schafer [4]. To accommodate the needs of satellite-based DSP, a high speed pipelined FFT is desired. Constructing an FFT for pipelined operation is demonstrated in Figure 8. Data is loaded sequentially into the butterfly machine (BFM) computation alternating between  $x_m(p)$  and  $x_m(q)$ .

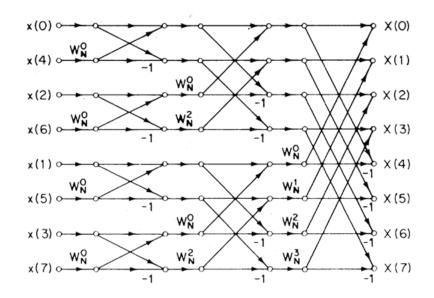


Figure 8. Flow Graph of 8-point DFT Using the Butterfly Machine Computation. Source: [4].

The algorithm being implemented is detailed in Figure 9. The constant geometry FFT is organized to take advantage of repeating patterns from stage to stage within the FFT structure. These patterns ease construction of a pipelined algorithm.

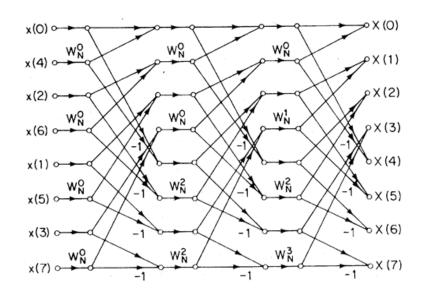


Figure 9. Rearrangement of Figure 1 to Allow Each Stage to Have a Constant Geometry Permitting Sequential Data Accessing and Storage. Source: [4]

## C. HIGH-SPEED PIPELINED FFT ARCHITECTURE

A high speed pipelined FFT is demonstrated in Figure 10. Specifically, the image depicts a digit-reversed block, alternating memory buffers, and the BFM computations. There are three memory buffer/BFM pairs within a N=8 FFT.

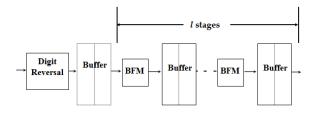


Figure 10. Basic Pipelined FFT Structure.  $N = 2^{1}$ , Where N Is the FFT Word Size. Adapted from [15].

Although similar to Figure 10, Figure 11 has added timing details to ensure that x(i) is placed into the correct slot of the output BFM ping-pong buffer. While the first buffer is filled according to the bit reversed ordering discussed in this chapter, successive buffers are filled to allow for efficient pipelined operation. They are filled according to  $k = 0, \frac{N}{2}, 1, \frac{N}{2} + 1, 2, \frac{N}{2} + 2, 3, \frac{N}{2} + 3, 4, \frac{N}{2} + 1, ..., N - 1$  [15]. For N = 8, k = 0, 4, 1, 5, 2, 6, 3, 7. This order allows x(i) to be fully utilized in both calculations in which it is involved before moving on to x(i+1).

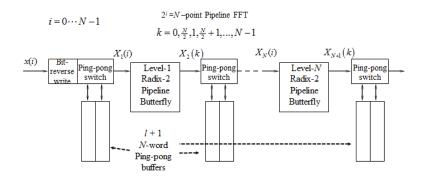


Figure 11. Detailed High Speed Pipelined FFT Structure. Adapted from [15].

## 1. Radix-2 Pipeline Butterfly Machine Architecture

A BFM that can be coded utilizing smaller blocks is displayed in Figure 12. It depicts two memory delays one clock-cycle long, two 2-input multiplexers, one memory delay  $d_m + 1$  cycles long, a complex multiplier, and a complex adder / subtracter. Additionally, timing details and twiddle factors are calculated within the image. This entire circuit is made complex by utilizing separate registers for the real and imaginary portion of the complex number.

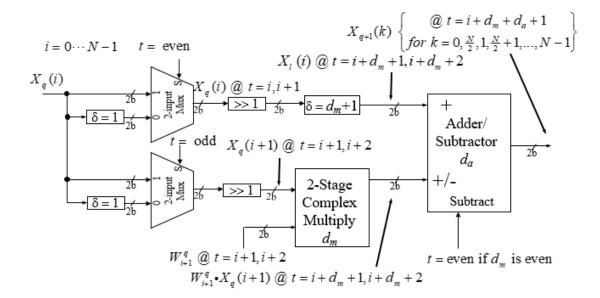


Figure 12. Pipeline Radix-2 Butterfly Machine (BFM) Architecture, Level *q*. Adapted from [15].

### 2. 2-Stage Pipelined Complex Multiplier Architecture

A one-stage pipeline real multiplier and a one-stage pipeline real adder is utilized to implement a two-stage pipeline complex multiplier. This is depicted in Figure 13. The multiplication must be performed on unsigned positive numbers. A module is inserted to perform a two's complement computation on negative numbers and strip off the sign bit, reducing numbers within the multiplier to 17 bits. After multiplication, there is an expansion to 34 bits, and a truncation is required to strip off the 17 least significant bits (LSBs). The truncated output has the sign added back and is two's complemented into a negative number if the exclusive-or of the signs of the multiplicand represent a negative product. An addition or subtraction is required to complete the complex product formation as a signed two's complement representation of the real and imaginary parts of the product.

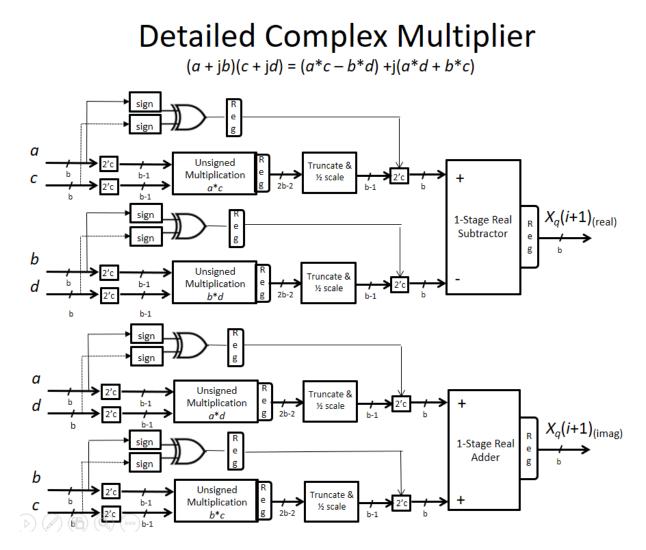


Figure 13. Two-Stage Complex Multiplier

Timing is important throughout the development of this design. Accessing a memory element, in this case a register, requires one clock cycle. This results in a one clock cycle delay in which the data is available for use. Testing must be done to ensure that multiplication and addition occur among the proper data elements. The delay of the multiplier  $d_m$  is equal to two clock cycles. The delay of the adder  $d_a$  is equal to one clock cycle. Timing is analyzed with simulation throughout the testing of the algorithm. This gives a visual depiction of where specific data elements align with the clock-cycle.

This design was instantiated in a Xilinx Artix-7 FPGA with Verilog hardware description language (HDL). The implementation and testing are discussed in Chapter IV.

THIS PAGE INTENTIONALLY LEFT BLANK

# IV. TESTING AND EVALUATION

Three major concepts are presented in Chapter IV. Planning and managing the testing of the FFT is in Section A. Here we find visual testing plans and major interfaces within the FFT that were tested. Sample inputs that were tested are listed in Section B. They showed up in hexadecimal, binary, and signed decimal throughout testing. Finally, the testing that was performed is demonstrated and displayed. The code that produced the test is described in detail, and simulations that were produced are analyzed. Verification of the code confirms that what was designed in Chapter III was produced and is usable for the intended space experiment purpose.

### A. TESTING PLAN

This section is divided into three sub-sections. The test plan for the FFT is detailed in Sub-section 1. Here the FFT was considered as an entire system. Interfaces between code blocks, henceforth known as "modules," were labeled to allow for correlation within simulations. The BFM module within the FFT is detailed in Sub-section 2. Being a smaller portion of the larger system FFT, it was labeled a component due to it consisting of multiple modules itself. This component performs multiplication on two signed binary numbers. The timing associated with this clocked systems is described and analyzed within Subsection 3. Timing analysis ensures that data arrives for processing at the right time and location.

#### 1. System Test Plan

A systematic approach to software testing was important due to the complex nature of the project. A system level test plan can be seen in Figure 14. Test points for simulation inputs and outputs are marked by numbers. Boxes represent the aforementioned components of the FFT. Test point S0 yields the initial vector that was sent to get digit reversed and stored. Test point S2 gives the output. Test point S3 is the outcome of the first stage BFM computation. Test point S4 is located at the end of stage 1.

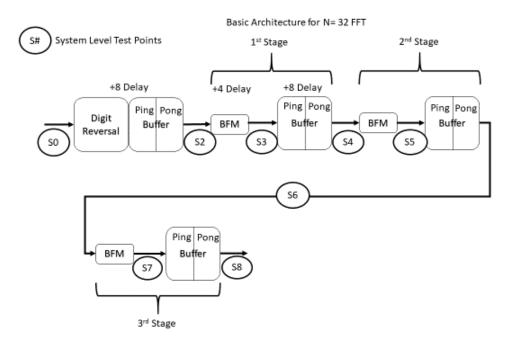


Figure 14. System Level Test Plan for N = 8 FFT

#### 2. Component Test Plan

The component-level test plan for the BFM can be seen in Figure 15. The numbers on the BFM diagram represent input/output points that were examined to verify results and timing against expected values. Point C0 allows the observation of inputs into the BFM. From C1, we get the result of the data after a one clock-cycle delay. Point C2 allows us to observe the result of a 2-to-1 multiplexer (mux) that was selected on even time, and C3 allows us to observe the results of a 2-to-1 mux selected on odd time. Points C4 and C6 allow observation of the result of a shift right by one operation which results in a scale-byhalf operation necessary to prevent the overflow in the BFM output adder / subtracter. This effectively divides the fixed-point binary number by two. Point C5 allows observation of the result of  $d_m + 1$ , where  $d_m$  is the delay of the multiplier and is equal to two. Point C7 allows observational of the result of the multiplication between Point C4 and the twiddle factor discussed earlier. Point C8 allows observation of the result of addition or subtraction of Points C5 and C7. The operand was chosen by a selector set by the time. Subtraction occurred on even time if  $d_m$  was even. The results were sent to the first, second, and third ping pong buffer respective to the BFM stage as shown in Figure 14, S3, S5, and S7, respectively. Those ping pong buffers satisfy the constant-geometry reordering seen in Figure 9.

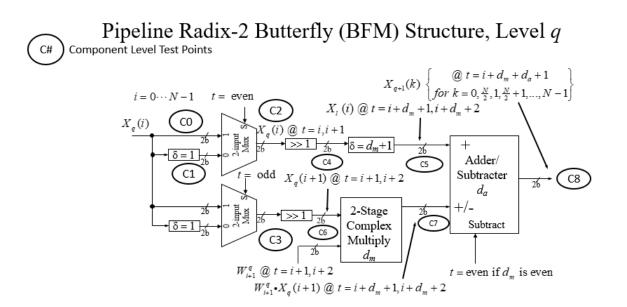


Figure 15. Component Level Test Plan for N = 8 BFM. Adapted from [15].

## **3.** Component Timing

The FPGA is connected to an internal clock running at 100 MHz. Variable CLK100MHZ was used to reflect this within the FFT code and waveforms. This physical clock was reduced to a 50 MHz clock and seen as clk\_50 within the FFT code and waveforms. This 50 MHz clock drives multiple code snippets. *Registers* are triggered to load on the rising edge of the clock. This is where the clock transitions from a binary 0 to binary 1.

The processor operates in a pipelined fashion as demonstrated in Figure 16 and 17. This means that multiple processing elements occur in parallel. The first eight samples processed through the BFM are displayed in Figure 16 with a generic  $X_q(t)$ .

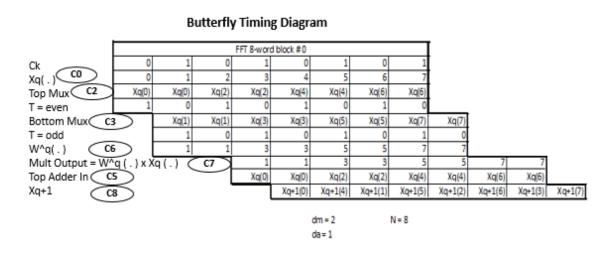


Figure 16. Butterfly Machine Timing Diagram Part A. Adapted from [15].

The next eight samples that are processed through the BFM are displayed in Figure 17. Within  $X_q(t)$ , t is less than 8 and greater than or equal to 0. The  $W^q(t)$  listed here was for test purposes only and is not a real value. Each successive *N*-word block was processed by the BFM and generated corresponding *N*-frequency components.

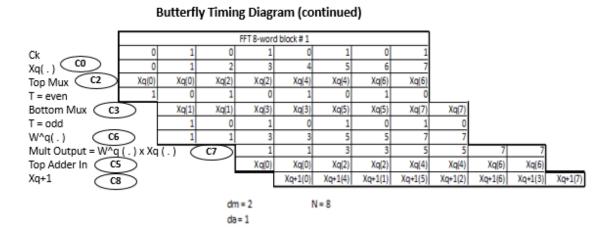


Figure 17. Butterfly Timing Machine Diagram Part B. Adapted from [15].

Figures 16 and 17 were integral to the analysis of the FFT timing. Timing was synchronized by staggering the initialization of each BFM to accommodate the pipelined delays in the data input to the FFT.

## **B.** TEST INPUTS

To verify outputs, there needs to be a known input for which expected outputs can be calculated. Starting with fixed point number 1.0,  $2^{-16}$  was subtracted from each iteration to generate identifiable inputs. During some of the simulations, we see a hexadecimal representation. Figure 18 shows displays in binary within the simulation environment.



Figure 18. Injected  $X_q(t)$  Inputs in Binary

Injected inputs have been listed in Table 4 in three possible formats to enhance readability. The stream continues without repeating; however, only the first sixteen have been listed.

$X_q(t)$	Binary	Fixed-Point Binary	Hexadecimal
$X_q(0)$	010000000000000000	1.0	10000
$X_q(1)$	0011111111111111111	.99998474	Offff
$X_q(2)$	001111111111111111	.99996948	Offfe
$X_{q}(3)$	001111111111111111111	.9999542236	Offfd
$X_{q}(4)$	001111111111111100	.9999389648	Offfc
$X_{q}(5)$	0011111111111111111111	.9999237060	Offfb
$X_q(6)$	0011111111111111010	.999908447	Offfa
$X_q(7)$	001111111111111001	.9998931	Offf9
$X_{q}(8)$	001111111111111000	.999877929	Offf8
$X_q(9)$	0011111111111111111111	.9998626708	Offf7
$X_q(a)$	0011111111111110110	.999847412	0fff6
$X_q(b)$	001111111111110101	.99983215	Offf5
$X_q(c)$	001111111111110100	.999816894	0fff4
$X_q(d)$	001111111111110011	.9998016357	Offf3
$X_q(e)$	001111111111110010	.99978637	Offf2
$X_q(f)$	001111111111110001	.99978484	Offf1

Table 4.  $X_q(t)$  Inputs Injected into FFT

## C. FFT SYSTEM AND COMPONENT TESTING

In this section, we detail each portion of the FFT code in snippets. The FFT is coded in behavioral Verilog. Vivado ISE's® simulation tool was utilized to provide the simulation waveforms that demonstrated the functionality of the major components of Figures 11 and 12. Xilinx Vivado ISE® synthesized the behavioral Verilog into a hardware definition that detailed the interconnections of gates and registers. The Xilinx Vivado synthesizer produced a realization that was instantiated in an FPGA to perform successive 8-point FFTs in a pipelined fashion on 18-bit signal samples. Timing challenges as a result of the pipelined nature of the design were present; however, timing diagrams were utilized within the testing process to ensure synchronization of data and verification of results.

#### 1. Bit Reversed Ping-Pong Buffer

The Bit-Reversed Ping-Pong Buffer provides internal storage for the data to be delivered in bit-reversed order to the BFM and allows data to flow in "blocks" based on the FFT's designed *N*. This Bit-Reversed Ping-Pong Buffer receives inputs in a sequential fashion, stores in a shuffled order, and outputs in a sequential order after N = 8 clock-cycles. A switch then occurs. The other half of the buffer receives inputs in a sequential fashion, while the previously filled buffer outputs in sequential order for N = 8 clock-cycles. This cycle repeats indefinitely.

The variables that are needed to code the Bit-Reversed Ping-Pong buffer are presented in Figure 19. *XqPing\_Real*, *XqPing\_Imag*, *XqPong\_Real*, and *XqPong\_Imag* are two-dimensional arrays consisting of eight-eighteen bit words with the MSB in the leftmost digit. Variables transpose and indexbr are both three bits wide, and the MSB is the left-most digit. Variables declared as *reg*, register, are not loaded with data until after a low-to-high clock-cycle. Variables declared as *wire are* placed immediately, but there is no storage mechanism for data.

<pre>reg [17:0] XqPing_Real [7:0];</pre>	<pre>// register for Ping real component</pre>
<pre>reg [17:0] XqPing_Imag [7:0];</pre>	<pre>// register for Ping imag component</pre>
<pre>reg [17:0] XqPong_Real [7:0];</pre>	<pre>// register for Pong real component</pre>
<pre>reg [17:0] XqPong_Imag [7:0];</pre>	<pre>// register for Pong imag component</pre>
<pre>wire [2:0] tranpose;</pre>	<pre>// wire used to tranpose counter</pre>
<pre>reg [2:0] indexbr;</pre>	<pre>// index utilized for bit reversal</pre>

Figure 19. Bit-Reversed Ping-Pong Buffer Variables and Arrays

To implement the bit reversal, an indexed value was created, shown as variable *indexbr* in Figure 20. Variable *indexbr* was used to transpose {*transpose*[2], *transpose*[1], *transpose*[0]} into {*transpose*[0], *transpose*[1], *transpose*[2]}. *Indexbr* was then used to address the arrays  $XqPing\_Real, XqPing\_Imag, XqPong\_Real, and <math>XqPong\_Imag$  to direct the input stream  $X_a(t)$  into the correct memory element. Bit counter[3], which is the MSB

of a four-bit counter, switched between even and odd every eight clock-cycles and triggered the *if/else* statement to load the Bit-Reversed Ping-Buffer or the Bit-Reversed Pong-Buffer. Variables *ping\_loading* or *pong\_loading* are set to high to indicate the active storage buffer on the simulations. A code snippet of the Bit-Reversed Ping-Pong Buffer loading into memory is shown in Figure 20.

```
76 🖯 always@(posedge clk)
77 🤅
       begin
78
             // counter transposed due to bit reversal
79
             indexbr <= {tranpose[0],tranpose[1],tranpose[2]};</pre>
80
81
            //The Pong buffer is outputed when the Ping buffer is
             //being filled up. Ping and Pong then alternate
82
83
             //The MSB of counter is utilized as
84
             //my switch bit. Buffers are complex numbers.
85
86
             //Alternating loading Ping and Pong buffers
87
             //based on MSB counter. Results in 16
88
             //cycle period.
89 🖻
            if (counter[3] == 1'bl) begin
                 XqPing_Real[indexbr] <= XqIn_PingPong_Real;</pre>
90
91
                 XqPing Imag[indexbr] <= XqIn PingPong Imag;</pre>
92
                 ping loading <= 1'bl;</pre>
                 pong_loading <= 1'b0;</pre>
93
94 🖞
                 end
95 🖯
                 else begin
96
                 XqPong_Real[indexbr] <= XqIn_PingPong_Real;</pre>
97
                 XqPong_Imag[indexbr] <= XqIn_PingPong_Imag;</pre>
98
                 pong loading <= 1'bl;</pre>
99
                ping_loading <= 1'b0;
100 合
                 end
101 🍐
         end
```

Figure 20. Bit-Reversed Ping-Pong Buffer Loading, Code Snippet

The Bit-Reversed Ping-Pong buffer outputs the reordered  $X_q(t)$  in variables  $XqOut\_PingPongw\_Real$  and  $XqOut\_PingPongw\_Imag$ . A conditional statement triggered on the MSB of the four-bit counter controls which buffer loads. Synthesized designs connect wires from the data of the Bit-Reversed Ping buffer or Bit-Reversed Pong buffer to the output variables depending on the result of the conditional statement. A low signal connects the Bit-Reversed Ping buffer, and a high signal connects the Bit-Reversed Pong buffer. The code snippet in Figure 21 displays the Bit-Reversed Ping-Pong buffer output process.

64 assign XqOut\_PingPongw\_Real = (counter[3]==1'b0) ? XqPing\_Real[counter[2:0]] : XqPong\_Real[counter[2:0]]; 65 assign XqOut\_PingPongw\_Imag = (counter[3]==1'b0) ? XqPing\_Imag[counter[2:0]] : XqPong\_Imag[counter[2:0]];

Figure 21. Bit-Reversed Ping-Pong Buffer Output  $X_{q}(t)$ 

The bit-reversed index, seen as *indexbr* within Figure 22, displays the bit reversal. Variable *indexbr* is a reversal of the transpose bits. The display is listed in binary; however, a conversion to decimal reveals the relationship discussed in Figure 7.

Figure 22. Variable *Indexbr* transposed from Counter[2:0]

Next, simulation waveforms were used to demonstrate the proper function of the code. Clock-cycles, buffer-memory, and data values can be observed producing the expected results in Figure 23. In this image Pong was loaded in the sequential order indicated by the binary high bit in the *pong\_loading* register. In addition, Figure 23 contains the successful Bit-Reversed Pong-Buffer instantiation and displays the  $X_q(t)$  {10000, 0ffff, 0fffe, 0fffd, 0fffc, 0fffb, 0fffa, 0fff9} being stored into the two-dimensional memory element as {10000, 0fffc, 0fffe, 0fffa, 0ffff, 0fffb, 0fffa, 0fffb, 0fffb, 0fffb, 0fffb, 0fffb, 0fffb, 0fffb, 0fffd}. This is the expected bit reversed order. Eight clock-cycles after  $X_q(t)$  was received as an input into the Bit-Reversed Ping-Pong Buffer, the input sequence outputs in the bit-reversed order.

									1,385.000 n	12
Name Value		11 240 58	1,260 ns	1,280 ns	1,300 ns	1,320 ns	1,340 ns	11 360 pc 11 3	30 mr	1 400 55
CLK100MHZ 1		1,240 ns	7,000	*****				1,360 ns 1,3	80 ns	1,400 ns
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1										
■ W counter[3:0] 1000				X Y (PR)	V N OLDO	Valde	Value		1000	X 1001
■ MXq_TopLevel[17:0] ← Input	X (0)				X Older			$X X_{g_{11}}^{011}(7)$	81110	01117
A Main PingPong Real (17:0)	10000	X 01111	X Offfe	01110	V Offic	01110	X Offfa	e1110	01118	X 0fff7
XqIn_PingPong_Imag[17:0] 00001						001				
■	0000	X 0001	X 0010	X 0011	X 0100	X 0101	X 0110	X 0111	1000	X 1001
III indexbr[2:0] 000	000	X 100	X 010	X 110	X 001	X 101	X 011	111	000	X 100
lindk 1										
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	×			200000,200000,20	0000,000000,0000	ek,x00000K,X00000K,	0000			X00000K,0
XqPong_Real[7:0][17:0] 0fff9,0fffd,0fffb,0	300000,300000		003 ( 200000, 2000	000000,0000	000000,011	an X 200000, 011	an ( )000000,0 ff	an ( x00000,0 ff fan	1110,01110	d,0fffb,0fD
🖪 📢 [7][17:0]					000000				$X_{a}(7)$	££9
■-12 [6][17:0] Offfd			00000		<u>X a (6)</u>		0	1114		
🖬 📲 [5][17:0] Offfb			V (A)	000000			$X_{a}(5)$	01110		
are [4][17:0] Active offf		000000	<u>X q</u> (4)			0	****			
Buffer Offa				>00000(				X <sub>a</sub> (3)	Offfa	
🖬 📲 [2][17:0] Offfe		300000		X <sub>a</sub> (2)		V (4)	Offfe			
🖬 🖏 [1][17:0] Offfe		V (0)	200003			$X_q(1)$		Offic		
International and a state of the state o	>000000	X <sub>q</sub> (0)				10000				
lie ping_loading 0										
la pong_loading 1										
	D D									
	Pong B	uffer load	ing begins	S						

Figure 23. Pong Buffer Loading in Bit Reversed Order, Ping Empty

The Bit-Reversed Pong-Buffer load required the eight clock-cycles shown in Figure 24 to get the first  $X_q(t)$  block of data to the BFM. This startup-delay only occurs once. It is required for every BFM stage.

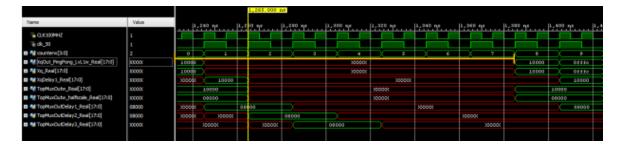


Figure 24. Startup Delay Between Bit-Reversed Ping-Pong Buffer Output and BFM

The Bit-Reversed Ping-Pong buffer switched loads as shown in Figure 25. The Bit-Reversed Pong buffer was filled as shown in Figure 23 and simultaneously outputted data in a sequential order, while Bit-Reversed Ping buffer was filled in bit reversed order. Input data, known as  $X_q(t)$ , holds the values of {0fff8, 0fff7, 0fff6, 0fff5, 0fff4, 0fff3, 0fff2, 0fff1 } and is stored in the two-dimensional Bit-Reversed Ping buffer as {0fff8, 0fff4, 0fff6, 0fff2, 0fff7, 0fff3, 0fff5, 0fff1 }. These values represent the second block of data outputted to the first BFM.

										1,545.000 n	8
Name	Value		1,400 ns	1,420 ns	1,440 ns	1,460 ns	1,480 ns	1,500 ns	1,520 ns  1,5	10 ns	1,560 ns
CLK100MHZ	1										
₩ dk_50	1				14 (2)	14 101		10 10	14 1 1		
🖬 📲 counter[3:0]	0000	$\chi X_q(0)$	$X_q$ (1)	$\times X_{q^1}(2)$	X q 63	$X_q$ (4)	XX <sub>q1</sub> (ठ)	$X_{q_1}(6)$	$X_q(7)$	0000	0001
🛚 📲 Xq_TopLevel[17:0] 🔶   nput	🗹 SO 📜	01110	X 01117	01110 X	X 01115	01110	C1110	21110 X	X 0fff1	01110	1stef X
XqIn_PingPong_Real[17:0]	offfo	X 01110	01117	01110	01115	01110	C1110	21110 X	01111	01110	Offef
	00001					00	001				
	0000	1000	1001	1010	1011	1100	1101	1110	1111	0000	0001
	000	X 000	X 100	010	110	X 001	X 101	X 011	X 111	000	100
ិ <mark>e</mark> dk	1										
	offf1,0fff5,0fff3,0	300000,300000	,в Хэооос,зооо	bai X x00000, x000	000,000	000000,011	150 X 200000,011	50 X 300000, 0 f f 2	50 ( )00000,0 ###50		
	Offf1					$X_{a}$ (6				X <sub>q</sub> (7) <sub>61</sub>	ffl
	OfffS			000000			/				
Activo	Offf3				00000			⊃X <sub>q</sub> (5)	01113		
	Offf7 Offf2		00000	_∕X <sub>q</sub> (4)	>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>		0	1117	X _ (3)	01112	
Dunci	0fff6		300000		$X_{a}(2)$			01116		01112	
□- ₩ [1][17:0]	Offf4			>00000	<u></u>		$X_{a}(1)$	01116	41110		
	Offf8	10000	$\rightarrow X_{a}(0)$						OTTIN		
XqPong_Real[7:0][17:0] Output		Y	^^ q (0)		01119.01114.0	teeb.orere.ore	fa, Offfe, Offfc	10000			0,63330
16 ping_loading	52 00										
B pong_loading	0		1								
	-							Pong Buffe	riuli		
		Ping Bu	ffer loadi	ng begins							

Figure 25. Ping-Buffer Loading in Bit Reversed Order, Pong Full

## 2. Radix-2 Pipeline Butterfly Machine Sub-Component Testing

The BFM is the primary processing element of the FFT. BFMs operate on blocks of data and are designed specifically for the *N*-block they are processing. This BFM is operating on N = 8 clock-cycles. Variables for the BFM were declared as *wires* with exception of the registers needed to introduce required delays as shown in Figure 26. These delays are needed for data synchronization within the BFM. All needed variables are 18 bits wide with the MSB to the left. Variables are paired to form the real and imaginary portions of a complex number.

34	34 (* dont_touch = "true" *)reg [17:0] TopMuxOutDelayl_Real, To	pMuxOutDelay2_Real, TopMuxOutDelay3_Real;
35	<pre>35 (* dont_touch = "true" *)reg [17:0] TopMuxOutDelayl_Imag, To</pre>	pMuxOutDelay2_Imag, TopMuxOutDelay3_Imag;
36	<pre>36 (* dont_touch = "true" *)wire [17:0] TopMuxOutw_Real, BottomM</pre>	uxOutw_Real;
37	<pre>37 (* dont_touch = "true" *)wire [17:0] TopMuxOutw_Imag, BottomM</pre>	luxOutw_Imag;
38	<pre>38 (* dont_touch = "true" *)wire [17:0] TopMuxOutw_halfscale_Rea</pre>	1, BottomMuxOutw_halfscale_Real;
39	<pre>39 (* dont_touch = "true" *)wire [17:0] TopMuxOutw_halfscale_Ima</pre>	g, BottomMuxOutw_halfscale_Imag;
40	<pre>40 (* dont_touch = "true" *)reg [17:0] XqDelay1_Real;</pre>	
41	<pre>41 (* dont_touch = "true" *)reg [17:0] XqDelay1_Imag;</pre>	
42	<pre>42 (* dont_touch = "true" *)wire [17:0] MultOutw_Real;</pre>	
43	<pre>43 (* dont_touch = "true" *)wire [17:0] MultOutw_Imag;</pre>	
44	<pre>44 (* dont_touch = "true" *)wire [17:0] AdderOutw_Real;</pre>	
45	<pre>45 (* dont_touch = "true" *)wire [17:0] AdderOutw_Imag;</pre>	

Figure 26. Butterfly Machine Variable Declaration Code Snippet

Two multiplexers are needed. One multiplexer, visibly the "top" multiplexer within the architecture diagram (Figure 15), selects on  $X_q(t)$  or  $X_q(t-1)$  based on even time

while the "bottom" multiplexer selects  $X_q(t)$  or  $X_q(t-1)$  based on odd time. This results in data falling on even time slots passing through the "top" multiplexer and data falling on odd time slots passing through the "bottom" multiplexer. Within the code, a single module is instantiated for both "top and "bottom" multiplexer; however, the input for the *clk* are opposed between multiplexers. Specifically, when the "top" multiplexer receives a high *clk* signal, the "bottom" multiplexer receives a low *clk* signal. Both continue to cycle between low and high *clk* signals. The code snippet for both multiplexers are shown in Figure 27.

```
47
        // ----- 2 to 1 Multiplexer Instantiated for top -----
48
       two2oneMux two2oneMux_Top
49
       (
           .inl_real(Xq_Real),
50
51
          .inl_imag(Xq_Imag),
          .in2_real(XqDelay1_Real),
52
          .in2_imag(XqDelay1_Imag),
53
          .clk(~counterw[0]),
54
55
           .out real(TopMuxOutw Real),
56
           .out_imag(TopMuxOutw_Imag)
57
       );
58
       // ----- 2 to 1 Multiplexer Instantiated for bottom -----
59
60
       two2oneMux two2oneMux Bottom
61
       (
62
          .inl_real(Xq_Real),
63
           .inl_imag(Xq_Imag),
         .in2_real(XqDelayl_Real),
64
65
           .in2_imag(XqDelay1_Imag),
         .clk(counterw[0]),
66
67
          .out_real(BottomMuxOutw_Real),
68
           .out imag(BottomMuxOutw Imag)
69
       );
```

Figure 27. "Top" and "Bottom" Multiplexer Code Snippet

The delay producing  $X_q(t-1)$  into both multiplexers is coded by sending the signal into a register. By loading after a clock-cycle, we introduce a delay into the data. This is an expected, and required, delay and is shown coded in Figure 28.

111	XqDelay1_Real <= Xq_Real;
112	XqDelayl_Imag <= Xq_Imag;

Figure 28. Delay into Both Multiplexer Code Snippet

#### a. Top Multiplexer, Half Scale, and 3-Cycle-Delay Sub-Component Test

The internals of the multiplexer is a conditional statement selecting on even and odd time. If time is even, selection is on  $X_q(t-1)$ . If time is odd, selection is on  $X_q(t)$ . Variables are 18 bits with the MSB to the left. The code snippet for the internals of the multiplexer is shown in Figure 29.

```
22 module two2oneMux (
23
       input wire signed [17:0] inl_real, // a
24
        input wire signed [17:0] inl_imag, // jb
25
       input wire signed [17:0] in2 real, // c
26
       input wire signed [17:0] in2_imag, // jd
27
       input wire
                                 clk,
28
       output wire signed [17:0] out_real, // c or a
29
       output wire signed [17:0] out imag // jd or jb
30
       );
31
32
        assign out_real = (clk==1'b0) ? in2_real : in1_real; //Real Component
        assign out_imag = (clk==l'b0) ? in2_imag : inl_imag; //Imag Component
33
34 🛆 endmodule
```

Figure 29. Internal Multiplexer Code Snippet

An arithmetic shift right by one moves the binary point one bit to the left. When performed on the data, outputs of the "top" multiplexer" are reduced by a factor of-one-half. The code in Figure 30 was utilized for this shift operation.

98 assign TopMuxOutw\_halfscale\_Real = TopMuxOutw\_Real >> 1; 99 assign TopMuxOutw\_halfscale\_Imag = TopMuxOutw\_Imag >> 1;

Figure 30. Top Multiplexer Half Scale

Half-scaled data from the "top" multiplexer was then delayed by three clock-cycles. Implementation of this was performed utilizing registers. Each register store operation introduces a one clock-cycle delay. The code can be seen in Figure 31.

105	TopMuxOutDelay1_Real <= TopMuxOutw_halfscale_Real;
106	TopMuxOutDelayl_Imag <= TopMuxOutw_halfscale_Imag;
107	TopMuxOutDelay2_Real <= TopMuxOutDelay1_Real;
108	TopMuxOutDelay2_Imag <= TopMuxOutDelay1_Imag;
109	TopMuxOutDelay3_Real <= TopMuxOutDelay2_Real;
110	TopMuxOutDelay3_Imag <= TopMuxOutDelay2_Imag;

Figure 31. Top Multiplexer Three-Clock-Cycle Delay

BFM test point C0 indicates the input of the BFM. The delay into the multiplexer can be read at test point C1. Test point C2 presents the output of the "top" multiplexer, C4 shows the data after the half-scale operation, and C5 displays the output to the top multiplexer after a three clock-cycle delay. Half-scaling data results in a loss of significance while truncating. The multiplexer and the delayed outputs are shown in Figures 32, 33, and 34. Functionally, they are the same waveform; however, the radix has been changed to support readability from different perspectives. Hexadecimal, fixed-point binary, and binary views are present. The delay being observed was needed to synchronize data with the bottom multiplexer due to the multiplier delay. Registers store data internal to the multiplier operation resulting in a two clock-cycle delay. Test point C5 demonstrates the delay of C2 after the half-scale operation seen in C4.



Figure 32. Top Multiplexer Output, Hexadecimal

A simulation view for fixed-point binary, 16-bit radix notation can be seen in Figure 33. This allows the decimal number to be read. Recall that values have been normalized to  $-2 \le X_a(t) < 2$ .

1 CLK100MHZ	1																		
14n dk_50	1																		
III - Via counterw [3:0]	0	8		9		( a		b		e		d				) E		0	
🖬 💐 XqOut_PingPong_Lvl.1w_Real[17:0]	0.9998779296875	1.0		0.99993896	40	0.9999694	20	0.9999084	470	0.99998474	10	0.99992370	60	0.9999542	30	0.99989318	80	0.9998779	290
🖬 📲 Xq_Real[17:0] 🔍 CO	0.9998779296875	1.0		0.99993896	40	0.9999694	20	0.9999084	70	0.9999847	10	0.99992370	60	0.9999542	230	0.99989318	80	0.9998779	290
1 - Value (17:0) 1 - C1	0.999893188476563	0.0		1.0		0.9999389	640	0.9999694	02D	0.99990844	70	0.99998474	10	0.9999237	eп	0.99995422	30	0.9998931	880
II M TopMuxOutw_Real[17:0] C2	0.9998779296875		1	.0		0.995	969	482421875		0.999	984	741210938		0.999	954	223632813		0.9998	792
TopMuxOutw_halfscale_Real[17:0]	C4 3895484375		0.	. 5		Χ		0.499	984	741210938				0.499	969	482421875		0.49993	8964
TopMuxOutDelay1_Real[17:0]	0.499969482421875	0.0			0	. \$		χ		0.499	984	741210938				0.4999	169	82421875	
TopMuxOutDelay2_Real[17:0]	0.499969482421875		0.	0			0	.5				0.499	984	741210938				0.49996	482
TopMuxOutDelay3_Real[17:0]	C5 34741210938			0.0					0.	. 5		×		0.499	984	741210938			

Figure 33. Top Multiplexer Output, Fixed-Point Binary

A binary view has also been provided and can be seen in Figure 34. Each view has utility based on what information is desired.

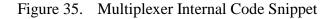
													_			
Le CLK100MHZ	1															
\} <sub>0</sub> dk_50	1															
🖬 📲 counterw[3:0]	c		8		$\square$	9			a			ъ			e	
XqOut_PingPong_LvL1w_Real[17:0]	00111111111111111	01	000000000000000	00	00	111111111111111	00	00	111111111111111	10	00	111111111111110	10	00111	1111111111	111
🖬 📲 Xq_Real[17:0] 🛛 CO	001111111111111111	01	000000000000000	00	00	111111111111111	0	00	1111111111111111	10	00	111111111111110	10	00111	1111111111	111
🛛 📲 XqDelay1_Real[17:0] 🛛 C1	001111111111111111	300	00000000000000000	×	01	0000000000000000	po	00	111111111111111	po	00	1111111111111111	10	00111	1111111111	010
TopMuxOutw_Real[17:0] C2	00111111111111111		01	0000000	0000000	00			00	1111111	1111111	10		00111	1111111111	111
TopMuxOutw_halfscale_Real[17:0]	C4		00	1000000		00					0001	111111111111111				
TopMuxOutDelay1_Real[17:0]	000111111111111111	010	000000000000000000000000000000000000000	×		00	1000000	0000000	00		$\sim$	0001	111111	111111		
TopMuxOutDelay2_Real[17:0]	000111111111111111		000	0000000		x			00	1000000	0000000	bo		00011	1111111111	111
TopMuxOutDelay3_Real[17:0]	C5 0000000000				0)0	000000000000000	50					0010	000000	000000		

Figure 34. Top Multiplexer Output, Binary

## b. Bottom Multiplexer Sub-Component Test

The "bottom" multiplexer utilizes the same module call to *two2oneMux*. The input change representing odd time, binds odd data points to this multiplexer as shown in Figure 35.

```
22 module two2oneMux (
23
        input wire signed [17:0] inl_real, // a
        input wire signed [17:0] inl_imag, // jb
24
25
        input wire signed [17:0] in2 real, // c
        input wire signed [17:0] in2_imag, // jd
26
27
        input wire
                                  clk,
28
        output wire signed [17:0] out real, // c
                                                   or
                                                        a
29
        output wire signed [17:0] out imag // jd or
                                                      jb
30
        );
31
32
        assign out real = (clk==1'b0) ? in2 real : inl real;
                                                             //Real Component
33
        assign out_imag = (clk==l'b0) ? in2_imag : inl_imag; //Imag Component
34 🔆 endmodule
```



Data in the "bottom" multiplexer is half scaled utilizing the same methodology as the "top" multiplexer. The output to the multiplexer is shifted right by one binary place utilizing the code in Figure 36.

100 assign BottomMuxOutw\_halfscale\_Real = BottomMuxOutw\_Real >> 1; 101 assign BottomMuxOutw\_halfscale\_Imag = BottomMuxOutw\_Imag >> 1;

Figure 36. Bottom Multiplexer Half-scale Code Snippet

The bottom multiplexer simulation is displayed in Figure 37 and outputs its  $X_q(t)$  at odd *ts*. We can see this at test point C3. Test point C6 is the half-scale. Test point C7 is the output of the multiplier. The multiplier in this instance is viewed via a black box analysis. This means that only the inputs and outputs are analyzed.



Figure 37. Bottom Multiplexer Output with Multiplier as a Black Box

### c. Rate\_OneHalf\_complex\_Multiply Sub-Component Test

In section, we provide insight into the internal workings of the multiplier. Performing complex multiplication is not a trivial operation, and this code consists of many module calls. Each module call added interfaces and processing elements that had to be tested.

As part of performing multiplication within hardware, data manipulation is required to get the data into a format that behavioral Verilog can process. To start, negative binary numbers are stored in signed two's complement. The multiplication operation produces the anticipated result if operating on unsigned magnitude numbers; thus, signed two's complement numbers are first converted to unsigned magnitude. Unsigned multiplication is performed utilizing a behavioral operator. The sign is operated on separately and appended to the final resultant.

The final number must add the negative or positive sign that resuls from normal multiplication of signed numbers. The signs of the multiplicands are exclusive-or'd with each other to produce the correct sign. The new sign triggers a two's complement operation on the output of the truncated multiplication if the sign results in a negative number. The sign bit is then concatenated as the MSB to the truncated result. The multiplier test plan is provided in Figure 38 to guide a systematic way to test the multiplier and create common nomenclature for testable points.

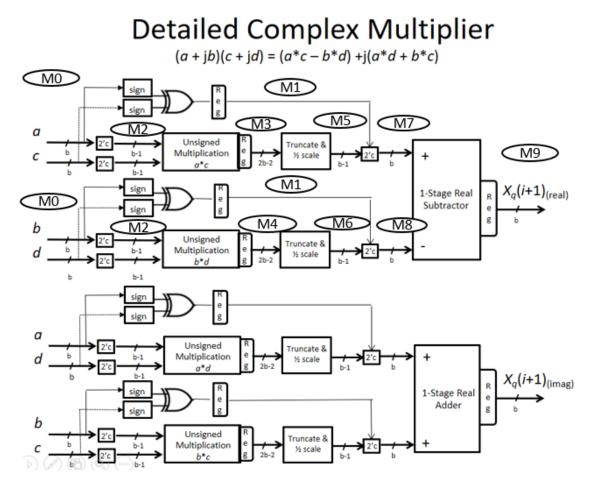


Figure 38. Test Plan for Multiplier and Sub-Component testing

The *wire* and register variables required for the multiplier coding are listed in Figure 39. Registers *sign\_mult\_left\_Real*, *sign\_mult\_right\_Real*, *sign\_mult\_left\_Imag*, and *sign\_mult\_right\_Imag* are utilized to store the result of complex multiplication on signed data. Four wires, one bit wide, indicate the sign of each portion of the complex number. Four wires, 18 bits wide, hold the concatenated 17-bit wide *unsigned\_a*, *unsigned\_jb*, *unsigned\_c*, and *unsigned\_jd* and their respective 1-bit signs.

11	(* dont touch = "true" *) reg sign mult left Real, sign mult right Real, sign mult left Imag, sign mult right Imag;
12	(* dont touch = "true" *) wire sign a;
13	(* dont_touch = "true" *) wire sign_jb;
14	(* dont_touch = "true" *) wire sign_c;
15	(* dont_touch = "true" *) wire sign_jd;
16	(* dont_touch = "true" *) wire [16:0] unsigned_a; //
17	(* dont_touch = "true" *) wire [16:0] unsigned_jb; //
18	(* dont_touch = "true" *) wire [16:0] unsigned_c; //
19	(* dont_touch = "true" *) wire [16:0] unsigned_jd; //
20	(* dont_touch = "true" *) wire [16:0] trunc_left_Real; //(a*c)
21	(* dont_touch = "true" *) wire [16:0] trunc_right_Real; //(jb*jd)
22	(* dont_touch = "true" *) wire [16:0] trunc_left_Imag; //(a*jd)
23	(* dont_touch = "true" *) wire [16:0] trunc_right_Imag; //(jb*c)
24	(* dont_touch = "true" *) wire [17:0] signed_left_Real; //(a*c)
25	(* dont_touch = "true" *) wire [17:0] signed_right_Real; //(jb*jd)
26	(* dont_touch = "true" *) wire [17:0] signed_left_Imag; //(a*jd)
27	(* dont_touch = "true" *) wire [17:0] signed_right_Imag; //(jb*c)

Figure 39. Multiplier Variables Code Snippet

This code segment details four calls to *twoComp* module. Each portion of the complex number is individually passed into the module as an 18-bit binary number. A 17-bit unsigned magnitude number was returned as an output and connected to a wire as seen in Figure 40.

52	//2'c operator prior to multiplication. returns unsigned number. returns 17 bits
53	<pre>twoComp insta(.In(a),.Out(unsigned_a)); //extracts data portion a</pre>
54	<pre>twoComp instb(.In(jb),.Out(unsigned_jb)); //extracts data portion jb</pre>
55	<pre>twoComp instc(.In(c),.Out(unsigned_c)); //extracts data portion c</pre>
56	<pre>twoComp instd(.In(jd),.Out(unsigned_jd)); //extracts data portion jd</pre>

Figure 40. Code Snippet for twoComp Module Call

Internal to the *twoComp* module is a conditional statement that selects an output based on the value of the MSB. If the MSB is a digital high, the input is inverted and incremented by one bit; while a digital low returns a copy of the input. This can be seen

coded in Figure 41. The MSB was truncated off so an unsigned magnitude number could be returned to the parent module.

```
23 module twoComp(
24 input wire [17:0] In,
25 output wire [16:0] Out);
26
27 assign Out = In[17]?~In+1:In;
```

Figure 41. Code Snippet for twoComp Module Internals

The MSB that was truncated within the twoComp module was also stored for use later within the multiplier module. The coding for the multiplier module store operation is displayed in Figure 42.

58	<pre>// saves sign bit. returns 1 bit</pre>
59	<pre>assign sign_a = a[17]; //extracts sign of a</pre>
60	<pre>assign sign_jb = jb[17]; //extracts sign of jb</pre>
61	<pre>assign sign_c = c[17]; //extracts sign of c</pre>
62	<pre>assign sign_jd = jd[17]; //extracts sign of jd</pre>

Figure 42. Sign Bit Extraction Code Snippet

Unsigned numbers in the one's complement form can successfully pass through a behavioral multiplier and produce expected results. A module was called to perform behavioral multiplication on the 17-bit binary numbers seen in Figure 43. Four 17-bit binary numbers are returned.

34	mult18x18 unsigned_multiply
35	(
36	.clk(clk),
37	.a(unsigned_a),
38	.jb(unsigned_jb),
39	.c(unsigned_c),
40	.jd(unsigned_jd),
41	.Out_left_Real(trunc_left_Real),
42	.Out_right_Real(trunc_right_Real),
43	.Out_left_Imag(trunc_left_Imag),
44	.Out_right_Imag(trunc_right_Imag));

Figure 43. Module Call for Behavioral Multiplication on Four Unsigned Complex Numbers Code Snippet

The unsigned multiplier module creates registers to store the output of the multiplication. Registers *mult\_left\_Real*, *mult\_right\_Real*, *mult\_left\_Imag*, and *mult\_right\_Imag* are declared as 35-bit registers. Storage must be available to accommodate the largest possible multiple. The coding of these variables is displayed in Figure 44.

34	//Complex Multiplier		
35	5 // Real	Ima	g
36	5 //(a+jb)(c+jd)= (a*c - 1	b*d) + j(b*c +	a*d)
37	(* dont_touch = "true"	*) <b>reg</b> [34:0] m	ult_left_Real;
38	(* dont_touch = "true"	*) <b>reg</b> [34:0] m	ult_right_Real;
39	<pre>(* dont_touch = "true"</pre>	*) <b>reg</b> [34:0] m	ult_left_Imag;
40	(* dont_touch = "true"	*) <b>reg</b> [34:0] m	ult_right_Imag;

Figure 44. Unsigned Multiplication Module Code Snippet

Complex multiplication can be seen coded in Figure 45. This code snippet is within the unsigned multiplier module. Multiplications are performed at the positive edge of every clock-cycle, and the results are stored within one of the four 35-bit registers. When two 17-bit binary numbers multiply, they produce one 34-bit binary number.

```
50 🖯
       always@(posedge clk)
51 🖯
         begin
52
           //Calculate real portion
           mult left Real <= a * c;</pre>
53
           mult_right_Real <= jb * jd;</pre>
54
55
56
           //Calculate imag portion
            mult_left_Imag <= a * jd;</pre>
57
           mult_right_Imag <= jb * c;</pre>
58
59 📄
            end
```

Figure 45. Behavioral Multiplication Code Snippet

The parent module functions with 17-bit binary numbers. The 17 leading bits contain the relevant data. The trailing 17 bits add precision to the binary number; however, truncation is required to return product to the original 17-bit binary number format. The truncation code snippet is displayed in Figure 46.

42	<pre>assign Out_left_Real[16:0] = mult_left_Real[32:16]; // (a*c)</pre>
43	<pre>assign Out_right_Real[16:0] = mult_right_Real[32:16]; // (jb*jd)</pre>
44	<pre>assign Out_left_Imag[16:0] = mult_left_Imag[32:16]; // (a*jd)</pre>
45	<pre>assign Out_right_Imag[16:0] = mult_right_Imag[32:16]; // (jb*c)</pre>

Figure 46. Output Truncation Code Snippet

The exclusive-or operation is performed on the sign bits within the multiplier parent module on the rising edge of the clock-cycle to compute the new sign. This can be seen coded in Figure 47. If the sign of the product is negative, the result is a concatenation of the computed sign and the two's complement of the magnitude of the product. If the sign of the product is positive, the result is a concatenation of the computed sign and the unsigned magnitude of the product.

```
72 // Statements below will be executed on every clock rising edge
73
74 always@(posedge clk)
75 begin
76 sign_mult_left_Real <= sign_a ^ sign_c;
77 sign_mult_right_Real <= sign_jb ^ sign_jd;
78 sign_mult_left_Imag <= sign_a ^ sign_jd;
79 sign_mult_right_Imag <= sign_jb ^ sign_c;
80 end</pre>
```

Figure 47. Exclusive-Or of Sign-Bit Code Snippet

Module *twoCompRedo* receives the new sign bit and the truncated output of the multiplier and produces the two's complement 18-bit numbers. The code snippet for the module call is displayed in Figure 48.

46	//2'c operator following multiplication. Concatenates the sign bit with a 2'c number. returns 18 bits
47	<pre>twoCompRedo twoC_leftReal(.signbit(sign_mult_left_Real),.number(trunc_left_Real),.sign_number(signed_left_Real));</pre>
48	<pre>twoCompRedo twoC_rightReal(.signbit(sign_mult_right_Real),.number(trunc_right_Real),.sign_number(signed_right_Real));</pre>
49	<pre>twoCompRedo twoC_leftImag(.signbit(sign_mult_left_Imag),.number(trunc_left_Imag),.sign_number(signed_left_Imag));</pre>
50	twoCompRedo twoC_rightImag(.signbit(sign_mult_right_Imag),.number(trunc_right_Imag),.sign_number(signed_right_Imag));

Figure 48. Code Snippet for twoCompRedo Module Call

Peering inside of *twoCompRedo*, we see that module *twoCompRedo* consists of a conditional statement selected with the sign bit. If the sign bit is a binary one, every bit in the binary number is inverted and then one bit is added. That binary number is then concatenated with the sign-bit before being sent to a *wire* for output. If the sign bit is zero, the original binary number is outputted after being concatenated with the sign bit. The *twoCompRedo* module code snippet is displayed in Figure 49.

```
23 module twoCompRedo(
24 input wire signbit,
25 input wire [16:0] number,
26 output wire [17:0] sign_number
27 );
28
29 assign sign_number = signbit?{signbit,~number+1}:{signbit,number};
30 endmodule
```

Figure 49. Module twoCompRedo Code Snippet

The final stage consists of adding or subtracting the signed multiplier output. The real component of the complex number is subtracted to get the final real number. The imaginary component of the complex number is added to get the final imaginary number. Both numbers are sent to a wire for output to the BFM module and the code is displayed in Figure 50.

64 assign Xqout\_real = signed\_left\_Real - signed\_right\_Real; 65 assign Xqout\_imag = signed\_left\_Imag + signed\_right\_Imag;



Two sets of complex numbers were tested through simulation. The input and output of (1+.5j)(0.25+0.125j) and (-1-0.5j)(1+j) is shown in Figure 51 as a simulation output.

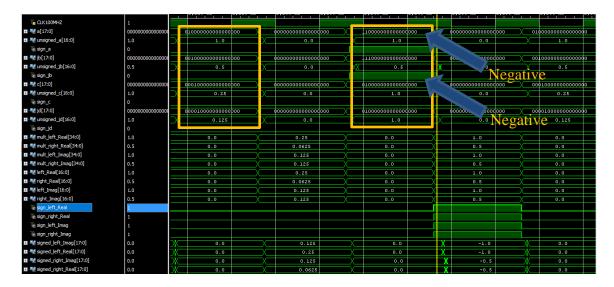


Figure 51. Complex Multiplication Module Testing

As mentioned previously, behavioral multiplication functions correctly on unsigned numbers. The seventeenth bit of data is the sign bit and must be stripped off. The circles within Figure 52 highlights the signal for the sign bit. Also, all negative numbers must be two's complemented to return to their magnitude in preparation for multiplication. Inputs can be viewed against outputs in Figure 52. Point M2 demonstrates the magnitude.

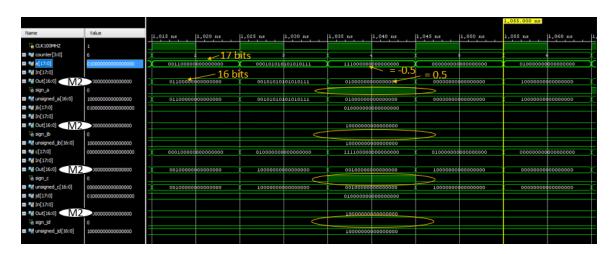


Figure 52. Conversion of Multiplier Inputs to Signed Magnitudes

Once all negative numbers are two's complemented, seen as test point M2, a behavioral multiplier multiplies the magnitudes. The multiplier produces a 34-bit unsigned number that is displayed as test points M3 and M4. The data we need are in the upper 17-bits of the data. The data was truncated to extract what was needed and the rest discarded. Test points M5 and M6 show the truncated data.

The original signs of the data were exclusive-or'd and can be seen at test point M1. The new sign was appended to the truncated multiplication output. If the sign indicates a negative number, the multiplication output is inverted and one bit is added. This is reflected as test points M7 and M8. Prior to being sent as an output to the adder / subtracter, test point M9 can be used to view the resultant value. The simulation labeled with applicable test points is displayed in Figure 53.

																			1,125.0	00 ns	
Name	Value	11.0	050 ns	: 1	1,060	ns	11.070		$l_m = 1_{80}$	ne		0 ne	11.10	00 ns	11	110 n	e	11.120 7	e	1,130 ns	
1% CLK100MHZ	1					<u></u>	1,070	1 Clock	-Cycle D	elay d	" = 2							77777			
🖬 🔜 counter[3:0]	d	5	=	_		<u>x</u>	7	x -	8	_	-			=	ъ						
🖬 📲 unsigned_a[16:0]	0.0	0.0	Ξŵ	1	0	ĵ —		0.5		2 Clock	-Cycle	Delay	1 0			=	=		0.0		
	0.0					1			1.0	_^						=	$\equiv$		0.0		
unsigned_c[16:0]	0.0	1.0	X	0.	0		0	. 25		-x	1.0		0.0	_χ_	1.0	=	_		0.0		
🖪 🐳 unsigned_id[16:0] 🛛	0.0								1.0	_				_		=			0.0		
🛚 📲 mult_left_Real[34:0] M3	0.0	0.12	s )(		(	0.0			(	0.125			1.0	_X_	0.0	=	1	0		0.0	
🖪 🖬 mult_right_Real[34:0]	0.0									1.0										0.0	
🖬 🖬 mult_left_Imag[34:0] 🛛 M3	0.0	0.5	X	0.	0	X	1.0			0.5					1.0					0.0	
🛛 📲 mult_right_Imag[34:0] M4	0.0	0.25	s X	1.	0		0.0			0.25			1.0		0.0		1	0		0.0	
🔓 sign_mult_left_Real 🤍 M1	0																				_
Dut_left_Real[16:0]	0.0	0.12	5 X		(	0.0				0.125			1.0		0.0	— X	1	0		0.0	
III - M trunc_left_Real[16:0] M5	0.0	0.12	:5 X		(	0.0				0.125			1.0		0.0		1	0		0.0	
lia sign_mult_right_Real M1	0																				
Out_right_Real[16:0]	0.0									1.0										0.0	
III M trunc_right_Real[16:0] M6	0.0									1.0										0.0	
1& sign_mult_left_Imag M1	0																				
III- Mag [16:0]	0.0	0.5		0.	0	$\mathbf{x}$	1.0			0.5		$\supset$			1.0					0.0	
🖬 📲 trunc_left_Imag[16:0] M5	0.0	0.5		0.	0		1.0			0.5		$\supset$			1.0					0.0	
🔓 sign_mult_right_tmag 🛛 M1	0																				
Dut_right_Imag[16:0]	0.0	0.25	5 X	1.	0	$\mathbf{x}$	0.0			0.25		$\rightarrow$	1.0		0.0		1	0		0.0	
n 👷 trunc_right_Imag[16:0] M6	0.0	0.25	5 )(	1.	0		0.0			0.25		$\supset$	1.0		0.0		1	0		0.0	
III M7		0.12	5 X		(	0.0			-	0.125		$\supset$	-1.0		0.0		1	0		0.0	
■ ¥ signed_right_Real[17:0] M8 M7	0.0									1.0										0.0	
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII	0.0	0.5	5 X	0.	0		1.0		-0.5		0.5	$\supset$	1.0		-1.0		1	0		0.0	
IVI8	0.0	-0.2	:5 )(	1.	0	X	0.0		0.25		-0.25		-1.0		0.0		1	0		0.0	
Xqout_real[17:0]	0.0	-0.66	60	-0.	875	X	-	1.0				-1.125			-2.0			.0		0.0	
🖬 📲 Xqout_imag[17:0]	-2.0	1.333	30 X	-0.	75	х —		L.O			-0.25		0.25		0.0		-	. 0	-2	. • X	0.0

Figure 53. Multiplier Test Points. Fixed Point

## d. Adder / Subtracter Component Sub-Component Test

The adder / subtracter combines the common results of the multiplication. The variables needed consist of two registers, *out\_real* and *out\_imag*, both 18 bits wide, and can be seen coded in Figure 54.

33	(* dont_touch = "true" *) reg [17:0] out_real;
34	(* dont_touch = "true" *) reg [17:0] out_imag;
35	
36	//Complex Addition/Subtraction
37	// Real Imag
38	//(a+jb)+(c+jd) = a + c , j(b+d)
39	//(a+jb)-(c+jd) = a - c , j(b-d)
40	<pre>assign Xqout_real = out_real; //Real Component</pre>
41	assign Xqout_imag = out_imag; //Imag Component

Figure 54. Adder / Subtracter Variables Code Snippet

The adder / subtracter is the last component of the BFM. On the positive edge of every clock-cycle, a selection was made to either add or subtract. This selection is based on the value of *counter*. A digital high results in addition, while a digital low results in subtraction as seen in Figure 55.

```
43 calways@(posedge clk)
44 begin
45 out_real <= (counterw==1'bl) ? TopIn_real + BottomIn_real : TopIn_real - BottomIn_real; //Real Component
46 out_imag <= (counterw==1'bl) ? TopIn_imag + BottomIn_imag : TopIn_imag - BottomIn_imag; //Imag Component
47 calways@(posedge clk)
43 begin
44 begin
45 end
46 out_imag <= (counterw==1'bl) ? TopIn_imag + BottomIn_imag : TopIn_imag - BottomIn_imag; //Imag Component
47 begin
48 begin
49 begin
49 begin
40 begin
40 begin
40 begin
40 begin
41 begin
41 begin
42 begin
43 begin
44 begin
45 out_real <= (counterw==1'bl) ? TopIn_real + BottomIn_real : TopIn_real - BottomIn_real; //Real Component
46 begin
47 begin
47 begin
47 begin
47 begin
48 begin
49 begin
49 begin
49 begin
40 b
```

Figure 55. Adder / Subtracter Code Snippet

The simulation for the adder / subtracter is displayed in Figure 56. The top multiplexer input, seen as M7, was delayed by  $d_m + 1$ . The inputs were added on even clock cycles and subtracted on odd clock cycles, with the results being viewed at test point M9.



Figure 56. Adder / Subtracter Adds on Even Clock-Cycles

# e. 2<sup>nd</sup> Stage Ping Pong Buffer Sub-Component Test

Four two-dimensional registers are utilized to store the real and imaginary components of the First-Stage Ping-Pong buffers. Variables *XqPing\_Real*, *XqPing\_Imag*, *XqPing\_Real*, and *XqPing\_Imag* are declared as 18-bit wide words. In addition, a 4-bit register *buffer\_counter* was utilized to count. These can be seen in Figure 57.

58	(* dont_touch = "true" *) reg	<pre>[17:0] XqPing_Real [7:0];</pre>	<pre>// register for Ping real component</pre>
59	(* dont_touch = "true" *) reg	<pre>[17:0] XqPing_Imag [7:0];</pre>	<pre>// register for Ping imag component</pre>
60	(* dont_touch = "true" *) reg	[17:0] XqPong_Real [7:0];	<pre>// register for Pong real component</pre>
61	(* dont_touch = "true" *) reg	[17:0] XqPong_Imag [7:0];	<pre>// register for Pong imag component</pre>
62	(* dont_touch = "true" *) reg	<pre>[3:0] buffer_counter;</pre>	

Figure 57. First-Stage Ping-Pong Buffer Variables Code Snippet

The 4-bit *buffer\_counter* was utilized to produce sixteen timeslots. Eight timeslots are needed for the First-Stage Ping buffer and eight timeslots are needed for the First-Stage

Pong buffer to fill according to  $k = 0, \frac{N}{2}, 1, \frac{N}{2} + 1, 2, \frac{N}{2} + 2, 3, \frac{N}{2} + 3, 4, \frac{N}{2} + 1, ..., N - 1$ . Since N = 8, the fill order is k = 0, 4, 1, 5, 2, 6, 3, 7 and can be seen in Figures 58 and 59.

```
114 🗄
                  3'b0100: //Xq(4)
115 🗄
                     begin
                      XqPing_Real[0] <= XqIn_PingPong_Real;</pre>
116
117
                      XqPing_Imag[0] <= XqIn_PingPong_Imag;</pre>
118
                     buffer_counter = 4'd0;
119 🍐
120 🖯
                      end
                3'b0101: //Xq(5)
121 🖯
                     begin
122
                      XqPing Real[4] <= XqIn PingPong Real;</pre>
                      XqPing_Imag[4] <= XqIn_PingPong_Imag;
123
                      buffer_counter = 4'dl;
124
125
                      end
126 🗟
                3'b0110: //Xq(6)
127 🖯
                      begin
                      XqPing_Real[1] <= XqIn_PingPong_Real;</pre>
128
129
                      XqPing_Imag[1] <= XqIn_PingPong_Imag;</pre>
                      buffer_counter = 4'd2;
130
131 🍐
                      end
                 3'b0111: //Xg(7)
132 🤅
133
                      begin
                      XqPing_Real[5] <= XqIn_PingPong_Real;</pre>
134
                  XqPing_Imag[5] <= XqIn_PingPong_Imag;</pre>
135
136
                      buffer_counter = 4'd3;
137 🛓
                      end
                 4'b1000: //Xq(8)
138 🖯
139 🗄
                     begin
140
                      XqPing_Real[2] <= XqIn_PingPong_Real;</pre>
141
                      XqPing_Imag[2] <= XqIn_PingPong_Imag;</pre>
                      buffer counter = 4'd4;
142
143 🍐
                      end
```

Figure 58. First-Stage Ping Buffer Input Code Snippet

```
144 🖯
                  4'b1001: //Xq(9)
145 🖯
                     begin
146
                      XqPing_Real[6] <= XqIn_PingPong_Real;</pre>
147
                     XqPing_Imag[6] <= XqIn_PingPong_Imag;</pre>
                     buffer_counter = 4'd5;
148
149 👌
150 🗟
                      end
                4'b1010: //Xq(10)
151 🖯
                      begin
152
                      XqPing_Real[3] <= XqIn_PingPong_Real;</pre>
153
                      XqPing_Imag[3] <= XqIn_PingPong_Imag;</pre>
                      buffer_counter = 4'd6;
154
155 🍐
                      end
156 🖯
                4'b1011: //Xq(11)
157 🖯
                      begin
158
                      XqPing_Real[7] <= XqIn_PingPong_Real;</pre>
159
                      XqPing Imag[7] <= XqIn PingPong Imag;
160
                      buffer_counter = 4'd7;
161 🍐
                      end
```

Figure 59. First-Stage Ping Buffer Input Code Snippet (Continued)

Component level testing of the First-Stage Ping-Pong buffer demonstrates that inputs flow into the appropriate location at the desired time. The loading process of both the First-Stage Ping buffer and the First-Stage Pong buffer is annotated in Figure 60. The Pong buffer is filled prior to filling Ping buffer, which then alternates every eight clock-cycles. All subsequent stages of Ping-Pong buffer are identical and consistent with the  $k = 0, \frac{N}{2}, 1, \frac{N}{2} + 1, 2, \frac{N}{2} + 2, 3, \frac{N}{2} + 3, 4, \frac{N}{2} + 1, ..., N - 1$  loading scheme seen in Figure 59.

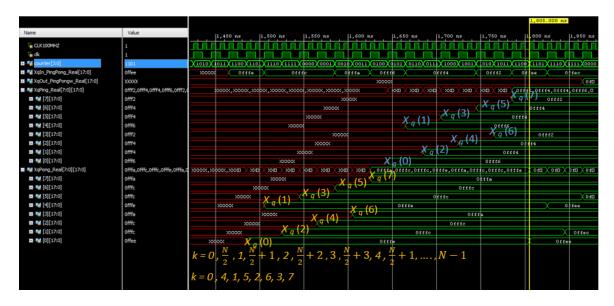


Figure 60. First-Stage Ping-Pong Buffer Simulation

The code of the FFT and simulations to verify operation at a component and subcomponent level was described in this chapter. The integration of the FFT as a whole system is discussed in Chapter V.

## V. END TO END TESTING / INTEGRATION TESTING

The operability of the Fast Fourier transform as a system is confirmed in this chapter. The four test vectors seen in Table 5 are utilized. How real  $X_q(t)$  inputs into a FFT produce both real and imaginary outputs is shown in Table 5. Separate columns exist for real and imaginary portions of the complex number.

Real Input	Real Output	Imaginary	Scaled	Scaled
1	Ĩ	Output	Real Output	Imaginary Output
$\{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1$	$\{8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,$	$\{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0$	$\{1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,$	$\{0, 0, 0, 0, 0, 0, 0, 0, 0\}$
1, 1, 1}	0, 0}	$0, 0, 0\}$	$0, 0, 0\}$	
$\{1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,$	$\{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1$	$\{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0$	$\left\{\frac{1}{8}, \frac{1}{8}, \frac{1}{8},$	$\{0, 0, 0, 0, 0, 0, 0, 0, 0\}$
0, 0, 0}	1, 1}	$0, 0, 0\}$	t	
			$\left[\frac{1}{8}, \frac{1}{8}, \frac{1}{8}\right]$	
$\{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0$	$\{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0$	$\{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0$	$\{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0$	$\{0, 0, 0, 0, 0, 0, 0, 0, 0\}$
$0, 0, 0\}$	0, 0}	$0, 0, 0\}$	$0, 0, 0\}$	
$\{0, 0, 0, 1, 0$	{1, -0.707, 0,	{0, -0.707, 1, -	$\frac{1}{8}, -\frac{0.707}{8}, 0,$	$(0, -\frac{0.707}{8}, \frac{1}{8}),$
,0, 0, 0}	0.707, -1, 0.707,	0.707, 0,		l
	0,707}	0.707, -1,	$\frac{0.707}{8}, -\frac{1}{8},$	$-\frac{0.707}{8}, 0, \frac{0.707}{8},$
		0.707}	$\frac{.707}{8}, 0, \frac{.707}{8}$	$-\frac{1}{8}, \frac{0.707}{8}$

Table 5. Test Vector Input and Expected Output

Three major sections are present in this chapter. A higher level perspective that shows an analysis utilizing the constant geometry FFT architecture is provided in Section A. The detailed analysis of the first test vector in Table 5 is detailed in Section B. The flow of data is demonstrated through annotated simulations. The results of integration and a hypothesis as to why are captured in Section C.

#### A. ALL TEST VECTOR ANALYSIS

The generic FFT architecture utilized to implement the coding is being reinserted as Figure 61. Numbers in sequence multiply with each other, and numbers that are in parallel add where paths converge. Unlabeled arrowheads imply a multiplication by one. In addition, scaling is not displayed within the butterfly diagram.

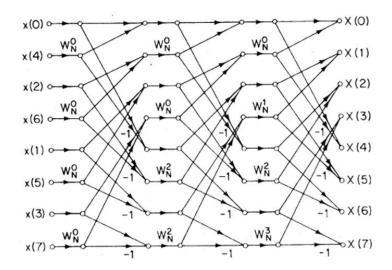


Figure 61. Generic Constant Geometric FFT. Adapted from [4].

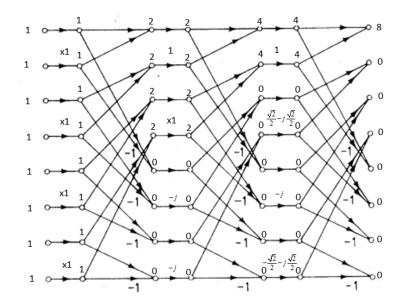


Figure 62. Test Vector-1 {1, 1, 1, 1, 1, 1, 1, 1}. Adapted from [4].

Test Vector-2 inputs {1, 0, 0, 0, 0, 0, 0, 0}. After the first stage processing, {1, 0, 0, 0, 1, 0, 0, 0} resulted. Second stage processing produced {1, 0, 1, 0, 1, 0, 1, 0}. Third stage processing results in {1, 1, 1, 1, 1, 1, 1} and is scaled as { $\frac{1}{8}$ ,  $\frac{$ 

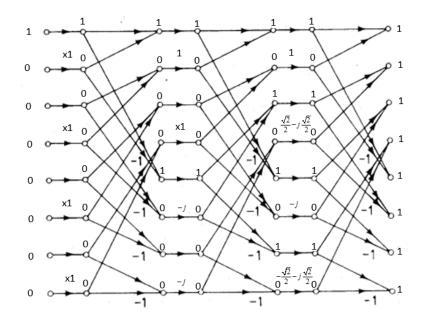


Figure 63. Test Vector-2 {1, 0, 0, 0, 0, 0, 0, 0}. Adapted from [4]. 53

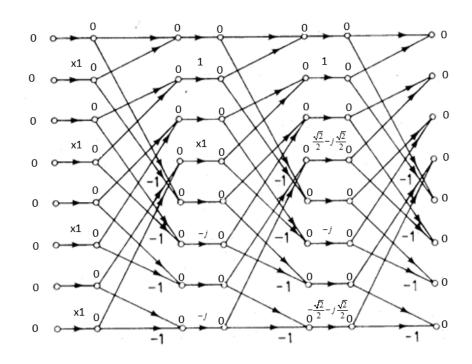


Figure 64. Test Vector-3 {0, 0, 0, 0, 0, 0, 0, 0}. Adapted from [4].

Test Vector-4 began with {0, 0, 0, 1, 0, 0, 0, 0}. First stage processing resulted in {0, 0, 0, 1, 0, 0, 0, 1,}. Second stage processing produced {0, 1, 0, -*j*, 0, -1, 0, *j*}. Third stage processing resulted in {1,  $-\frac{\sqrt{2}}{2} - j\frac{\sqrt{2}}{2}$ , *j*,  $\frac{\sqrt{2}}{2} - j\frac{\sqrt{2}}{2}$ , *-1*,  $\frac{\sqrt{2}}{2} + j\frac{\sqrt{2}}{2}$ , *-j*,  $-\frac{\sqrt{2}}{2} + j\frac{\sqrt{2}}{2}$ }. This is the only test vector which yields a non-zero imaginary component. Test Vector-4 can be seen in Figure 65.

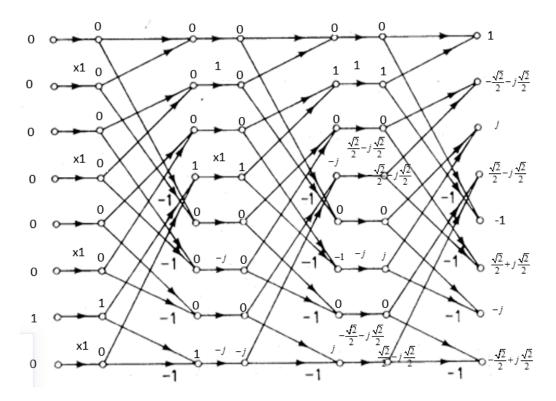


Figure 65. Test Vector-4 {0, 0, 0, 1, 0, 0, 0, 0} Displayed after Bit-Reversed to {0, 0, 0, 0, 0, 0, 0, 1, 0}. Adapted from [4].

#### **B.** TEST VECTOR ANALYSIS

A detailed analysis of annotation simulations on Test Vector-1 is given in this section. This is done by looking at the simulation of each component of the FFT and following how the test data is processed throughout. Each BFMs require a counter initialization to zero. Each stage has been synchronized on an individual counter to allow for proper execution of the FFT at a system level.

#### 1. Bit Reversed Ping Pong Buffer

Test Vector-1 was received as an input into the FFT algorithm. The test vector was driven from a test bench. In a real-life application, the signal is received from an analog-to-digital converter after receipt by an antenna, and the first half of the Bit-Reversed Ping-Pong buffer loads the data. The second half of the Bit-Reversed Ping Pong buffer will not yet have valid data as demonstrated in Figure 66.

CLK 100MHZ							ť		<u></u>			<u>i i p</u>		14		Ľ.		-	يبين		10		
1/2 clc100442	1		Щ.		_	<u> </u>				_	_								_				
iig dk_50 iig dk 200				+	<mark>8 c</mark>	ycleş	, Ð	igit Rev	ersa	al P	ing Po	ong	; Buffer	ini	tializing				-		6		
(c) CK_200 ■ Counter[4:0]	8		0				-	X 2			3							-					
counter [4:0]		<b>-</b> €			<u>~</u>	1	-	<u></u>		<u>~</u>		<u> </u>	<u> </u>		5		6		$\rightarrow$			8	$\equiv$
counter1(+:0)     Counter2[4:0]	0		24	+	ऱ_	25	+	X 26	-	<u>×</u>	27	<u>الج</u>	28	⊨	29	Ē	30	-	<b>⇒</b> ≻	31		0	
	3	l∛=∓	27 estady		X	28	rh -	29		Å,	30	<u>اللا</u>	31	⊨		$\equiv$	1	-	<u>الجنب</u>	2		3	$\blacksquare$
□ w counter3[4:0]	6	<u>-</u>	35 ladv	ect	QI.	L ⇒≷ :	4	<u>X, I, OL</u>		1, 1	<u>L, Ш</u> )	X	2		(з	X	4		X	5		6	
Xq_TopLevel[17:0]	1.0	_					-						1.0				/		_				
XqOut_PingPong_LvL1w_Real[17:0]	1.0	-							0.	0						Х	1.0		X.	0.0		(	1.0
XqOut_PingPong_LvL1w_Imag[17:0]	1.52587890625e-0					- Di		Poweres	J D:	na	nong	bu	1.525878	9062	izing, nq		toute		dia				
XqOut_PingPong_LvL2w_Real[17:0]	0.0		0.0		Х	0.5	5 <sup>IL</sup>	Meversa		пg	pong	bu			izing, nq	.90	itputs	ve	inc				
XqOut_PingPong_LvL2w_Imag[17:0]	0.0												0	0									
XqOut_PingPong_LvL3w_Real[17:0]	0.0	$\Sigma$		6	0.0			Х	1	. 0								0	. 0				
XqOut_PingPong_LvL3w_Imag[17:0]	0.0		0.0		Х	0.25		χ							(	.0							
XqOut_PingPong_LvL4w_Real[17:0]	1.32565307617188										0.0								$\square X$	0.125		1.3256530	0760
XqOut_PingPong_LvL4w_Imag[17:0]	1.32565307617188											0.0										1.3256530	0760
XqOut_BFM_LvL1w_Real[17:0]	0.0												0	0									
XqOut_BFM_LvL1w_Imag[17:0]	0.0												0	0									
XqOut_BFM_LvL2w_Real[17:0]	0.0		0.0		X		0	.25		X		0.0	0			0.2	25					0.0	
XqOut_BFM_LvL2w_Imag[17:0]	0.0												0	0									
XqOut_BFM_LvL3w_Real[17:0]	0.0	χ1.	325653	0760	χ-1	.325653	3070	χ	0	. 0		Ξx	0.0883636	470	-0.0883636	40 )	1.0		-X-			0.0	
XqOut_BFM_LvL3w_Imag[17:0]	0.0	X1.	325653	0760	X -1	.325653	3070	-1.87	5	X	1.875	Ξx	-0.088363	640	0.08836364	70 🗡					0.0		
W_M1_Real[17:0]	1.0												1	0									
W_W1_Imag[17:0]	0.0												0	0									
W_W2_Real[17:0]	1.0					1.0				X				0.	. 0							1.0	
W_ivi2_imag[17:0]	0.0					0.0				X				-1	0				Ξx			0.0	
🖬 📲 W_M3_Real[17:0]	0.0	$\overline{\chi}$		0	).0			X -0.7	0701	5991:	210938	X		1.	. 0	X	0.7	7070	01599	1210938			0.0
🖪 📲 W_W3_Imag[17:0]	-1.0	Γχ=		-	1.0			X -0.7	0701	5991:	210938			0.	. 0	Ex	-0.	707	0159	91210938			-1.0

Figure 66. Bit-Reversed Ping-Pong Buffer Initializing

The Bit-Reversed Ping-Pong buffer outputted  $\{1, 1, 1, 1, 1, 1, 1, 1, 1\}$  as seen in Figure 67. Since this is a pipelined algorithm, the first stage BFM receives data while simultaneously processing data. Four clock-cycles are required for the BFM to completely process the data. The startup delay is equivalent to initialization of the BFM.

CLK100MHZ	1					+++		<u>.</u>						+++						÷
₩ dk_50				_									_						_	
1 dk_200		<b>−</b> +{	3 cyc	les,	Digit	Rev	ersal Pii	ig f	ong <mark>B</mark> u	ffe	r outpu	ttin	g n h	h						
🛛 😼 counter[4:0]	16		8	X	9		10	X					13				15		16	
🗉 📲 counter 1[4:0]	8	13	0	X	1		2	X	3		4		5		6		7		8	
counter2[4:0]	11		1 cvc	lok	1st St	ano	<b>BEM</b> 5pr	bca	ssing te	oct.	voctor '	E	8	Ħ	9	5	10		11	
counter3[4:0]	14	Γχ.	6	X	7	Be		Fχ	aanig u		10	B	11	Ħ	12	Ē	13		14	
🛛 📲 Xq_TopLevel[17:0]	0.0	1	0	=7=		F				F			0.0	F						
XqOut_PingPong_LvL1w_Real[17:0]	1.0										1.0									
XqOut_PingPong_LvL1w_Imag[17:0]	1.525878906256		<b>N</b>	0		<b>b</b> .	0	- c	с.		1.525878	9062	Se-05 4 4	4.1						
XqOut_PingPong_LvL2w_Real[17:0]	0.5		Digit	. Ke	/ersal	Pin	g Pong	Bur	ter out	8.6	ting {1,	,±,	1,1,1,1,1,	, <del> </del> }					0.5	
XqOut_PingPong_LvL2w_Imag[17:0]	0.0											.0								
XqOut_PingPong_LvL3w_Real[17:0]	0.25			0.0			0.25	Εx	0.0		0.25		0.0		0.25		0.0		0.25	
XqOut_PingPong_LvL3w_Imag[17:0]	0.0										0	.0								
XqOut_PingPong_LvL4w_Real[17:0]	0.0	1.32	565307	60	0.0	$\square$	0.0883636	70 )	-0.125		-1.325653	070	0.0	$\square$	-0.0883636	40	1.0			0.0
🛛 📲 XqOut_PingPong_LvL4w_Imag[17:0]	0.0	1.32	565307	60 🔪	-1.875		-0.088363	640 )	0.0		-1.325653	070	1.875	$\square$	0.08836364	70			0.0	
XqOut_BFM_LvL1w_Real[17:0]	1.0	-		0.0		$\square$		0.	5		1.0		0.0	$\square$	1.0		0.0		1.0	
XqOut_BFM_LvL1w_Imag[17:0]	0.0				- 1	. c.	age BFN	. :	tet al tata		0	9								
XqOut_BFM_LvL2w_Real[17:0]	0.0				- 15	ιsι	age briv	1 111	manzin	<b>в,</b> Г	io outp	i s	valiu							
XqOut_BFM_LvL2w_Imag[17:0]	0.0										C	0								
XqOut_BFM_LvL3w_Real[17:0]	0.125							0.0						$\square$			0	.125		
XqOut_BFM_LvL3w_Imag[17:0]	0.0										0	.0								
🛛 📲 W_lvl1_Real[17:0]	1.0										1	0								
🛛 📲 W_lvl1_Imag[17:0]	0.0										C	0								
🛛 📲 W_lvl2_Real[17:0]	1.0				1.0			Бx				0.	0						1.0	
🛯 🔜 W_lvl2_Imag[17:0]	0.0				0.0							-1.	. 0						0.0	
🛛 📲 W_lvl3_Real[17:0]	0.0	X		0.0			-0.70	0155	991210938			1.	0		0.707	0159	91210938			0.0
W [v]3 Imag[17:0]	-1.0	~		-1.0					991210938			0.	_		0.001	0.15	991210938			-1.1

Figure 67. Bit-Reversed Ping-Pong Buffer Output

## 2. First-Stage BFM and Ping Pong Buffer

After a four clock-cycle initialization, the First-Stage BFM outputs  $\{1, 1, 1, 1, 0, 0, 0, 0, 0\}$  as can be seen in Figure 68. These outputs were sent into the First-Stage Ping-Pong buffer. No valid outputs were present from the First-Stage Ping-Pong buffer until after initialization.

CLK100MHZ											•••	1.1.1		1.		<u>i</u>			1		1		
₩ dk_50	1	_	ч.			_												_		_			
ua_ dk_ 200	1		- 6				6									F		. 6	H		-		
Counter[4:0]	20		12			13	H				5		16		17	H		18	H	19		20	
counter [ 10]	12	× –	4		<del>\$</del>	5	⊨	6		<u> </u>	,	=	8	Ħ	9	+	$\geq$	10	ŧ	11		12	Ħ
counter2[4:0]	15	⊧⊱⊨	2		÷	8	⊨	( a		Δ	0		11	╞	12	ŧ	-	13	÷	14		15	=
counter3[4:0]	18	l≎=	10		∻=	11	≓	12	, =		3	≓≎	14	Ħ	15	⊨≎	$\succ$	16	⊨	17	F	18	
Xq_TopLevel[17:0]	0.0	<u> </u>	10		<u>~</u>		ŧ		-	^·				10	- 10	Þ		10	Ľ	17	F		Ŧ
XqOut_PingPong_LvL1w_Real[17:0]	0.0							1.0							/				-	0.0	+		
XqOut_PingPong_LvL1w_Imag[17:0]	1.52587890625e-0	_						1.0					1.525878	0.62						0.0	⊨		
	1.525070500230-0						0.0						0.5	0002	/			0.0				0.5	
XqOut_PingPong_LvL2w_Imag[17:0]	0.0	1	at Ct		D:-						~ .				lid			0.0			F	0.3	=
XqOut_PingPong_LvL3w_Real[17:0]	0.0	- <u></u> -	0.25	age	<u>, 1911</u>	ig po	PΒ.	builter			<b>57</b>	10,0	outputs	v a	jia					0.0	+		
XqOut PingPong LvL3w Imag[17:0]	0.0		0.23	,	<u> </u>	0.0	F	0.2		<u> </u>		_^		10	<u> </u>					0.0	+		
XqOut_PingPong_LvL4w_Real[17:0]	0.0			_		0.0			_	1									0.0				
XqOut_PingPong_LvL4w_Imag[17:0]	0.0	_	+8 cv		e 1	<sup>t</sup> Sta		REM	utni	t hol	4 in	15	Stago	Pin	g Pong	Ruf	for	ac f	1 1	1 1 0	n	0.0}	
XqOut_BFM_LvL1w_Real[17:0]	0.5		1.0	_	<u>,                                    </u>	0.0				X 0	0		1.0		0.0		/	1.0	1	0.0			0.5
XqOut_BFM_LvL1w_Imag[17:0]	0.0		1.0		~	0.0	F	1.	0	<u>^ 0</u>	•			lo	0.0	<u> </u>		1.0	-	0.0			0.0
Agout_sth_create_mag_shop	-0.25									0.0									ŧ,	0.25		-0.25	
XqOut_BFM_LvL2w_Imag[17:0]	0.0									0.1	_			0					1	0.25	F	0.23	E
Adout_or in_creating in agric in a state of the state	0.125			0.	0		ŧ								0	125							
XqOut_BFM_LvL3w_Imag[17:0]	0.0						E						0	0	•	120							
	1.0			=										0					-		+		
W_V1 Imag[17:0]	0.0																						
W lvl2 Real[17:0]	0.0					0.0				v				1.	0				E,			0.0	-
W_lvl2_Imag[17:0]	-1.0					-1.0				$\sim$				0.					ŧ			-1.0	
W_V[vl3_Real[17:0]	1.0	$\overline{}$			1.0	-1-0	Ē	0	202015	A 9912109:	0	-v		0.		<b>—</b>	_	-0.20	2015	991210938		/	1.0
W_W_VI3_Imag[17:0]	0.0	$\left  \right\rangle =$			0.0		Ħ			59912109		$\equiv$		-1		⊨⇒				991210938		$\succ$	0.0
	0.0	<u> </u>			0.0		P	-0.		55512109	30	_^		-1		$\vdash$		-0.70	0135	91210938		·	1.0

Figure 68. First-Stage BFM Output and First-Stage Ping-Pong Buffer Initialization

Outputs can be seen leaving the First-Stage Ping-Pong buffer in Figure 69. Also shown is the Second-Stage BFM initialization. Variable *counter2* drove the Second-Stage BFM; although, seen to be on count 16, the BFM interprets this as  $16 \mod(8)$ , which is interpreted as zero.

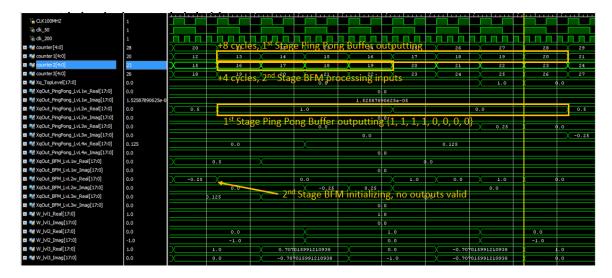


Figure 69. First-Stage Ping-Pong Buffer Outputting and Second-Stage BFM Initializing

## 3. Second-Stage BFM and Ping-Pong Buffer

Variable *counter2* was drove the Second-Stage BFM and Ping-Pong buffer. Starting at  $20 \mod 8$ , which equates to four, the Second-Stage BFM outputted {1, 1, 0, 0, 0, 0, 0, 0}. Logically, values are released and rearranged as explained in Chapter III, Paragraph C to maintain a constant geometry FFT. See Figure 70 for a demonstration.

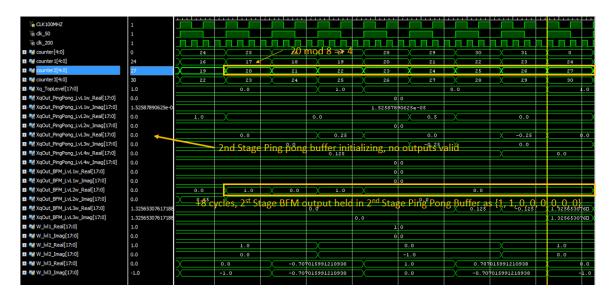


Figure 70. Second-Stage BFM Outputting and Second-Stage Ping-Pong Buffer Initializing

The Second-Stage Ping-Pong buffer outputted  $\{1, 1, 0, 0, 0, 0, 0, 0, 0\}$ , and the Third-Stage BFM took four clock-cycles to initialize and process the values as displayed in Figure 71. Variable *counter3* was initialized to zero to synchronize with the Third-Stage BFM operation.

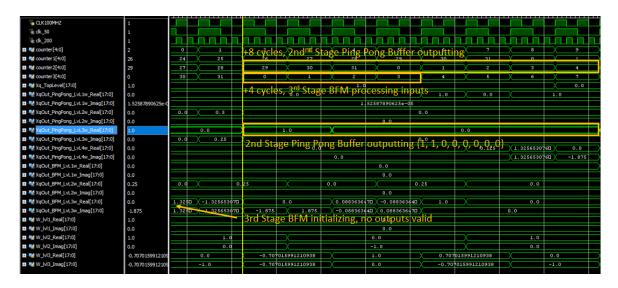


Figure 71. Second-Stage Ping-Pong Buffer Outputting and Third-Stage BFM Initializing

## 4. Third-Stage BFM and Ping-Pong Buffer

The Third-Stage BFM outputted {1, 0, 0, 0, 0, 0, 0, 0, 0}. This is the expected result for Test Vector-1 and is demonstrated in Figure 72. Also note that this came four clockcycles after the data was received into the BFM and twelve clock- cycles after this block of data was loaded into the Third-Stage Ping-Pong Buffer. Since the Forth-Stage Ping-Pong Buffer is loading its initial data, it does not output valid yet data in Figure 72

CLK100MHZ	1																				
¼a dk_50	1																				
ી∰ dk_200	1			uп	hr	uпп	hπ	ΠП				Т		hr				h			nni
🔲 📲 counter[4:0]	13	X 5	Éх	6	Éх	7	БX	8		9		Эx	10	Ξx	11	$\square$	12		13		14
a-w counter1[4:0]	5	29		30	Εx	31		0		1		X	2		3	$\square$	4		5	Ð	6
counter2[4:0]	8	χo	Ξx	1	Εx	2	Ex	3	Ex	- 4		-x	5		6	Ð	7		8	Ð	9
🖬 🌃 counter3[4:0]	11	Х 3		4	ΓX	5	Γ X	6	X	7		Х	8	X	9	$ \rightarrow $	10		11		12
Xq_TopLevel[17:0]	0.0				1.0											0.0					
XqOut_PingPong_LvL1w_Real[17:0]	1.0	0.0		1.0		0.0									1.0						
XqOut_PingPong_LvL1w_Imag[17:0]	1.52587890625e-0									1.5258	37890	625	e-05								
XqOut_PingPong_LvL2w_Real[17:0]	0.0										0.0										
XqOut_PingPong_LvL2w_Imag[17:0]	0.0										0.0										
XqOut_PingPong_LvL3w_Real[17:0]	0.0					0.0							0.25		0.0	$\square$	0.25		0.0		0.25
XqOut_PingPong_LvL3w_Imag[17:0]	0.0										0.0										
XqOut_PingPong_LvL4w_Real[17:0]	0.0		0.0			0.125		3256530	60	0.0	)		0.08836364	70 )	-0.125		-1.325653	070	0.0		-0.0880
XqOut_PingPong_LvL4w_Imag[17:0]	1.875		h Sta	age Dir	o n	on <mark>g b</mark> u	ffer	13254539	9h	a nn	2611	ŧň	uts val	<del>1</del> 1	0.0		-1.325653	070	1.875		0.08830
XqOut_BFM_LvL1w_Real[17:0]	0.0		1 30	JEC I II	6 P	0.0	lici	miniai	2	6, 110	- u	X		0.5		$\square$	1.0		0.0		1.0
XqOut_BFM_LvL1w_Imag[17:0]	0.0										0.0										
XqOut_BFM_LvL2w_Real[17:0]	0.0	X	0.2	5									(	0.0							
XqOut_BFM_LvL2w_imag[17:0]	0.0										olo										
XqOut_BFM_LvL3w_Real[17:0]	0.0	-0.088363	3640	1.0									0.0								0.125
XqOut_BFM_LvL3w_Imag[17:0]	0.0	0.0883636	5470 X		Cto		4	to the last		:- 20	1 0+		offing	Dor					0, 0, 0,	0	01
W_M1_Real[17:0]	1.0	+0	ycie	es, aru	Sta	ge briv	n ou	ւթու ո	eiu	111.2.	16	ag	e Ping	PUI	ig bull	FL I	as (1, 0	, v,	0, 0, 0,	υ,	0}
W_V1_Imag[17:0]	0.0										0.0										
🖬 📲 W_M2_Real[17:0]	0.0		0.0						1.	0								<b>0</b> .0			
W_lvl2_Imag[17:0]	-1.0		-1.0						0.	0								-1.0			
🖬 📢 W_M3_Real[17:0]	1.0	1.0		0.701	01599	1210938			0.	0			-0.70	0159	91210938			1	0		0.70700
🖬 📲 W_lvl3_Imag[17:0]	0.0	0.0		-0.70	01599	91210938	Бх		-1.	0			-0.70	0159	91210938			0	0		-0.7070

Figure 72. Third-Stage BFM Outputting and Third-Stage Ping-Pong Initializing

A test vector of  $\{1, 1, 1, 1, 1, 1, 1, 1\}$  should have a scaled result of  $\{1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\}$ . Shown in Figure 73 the correct result is demonstrated for Test Vector-1 processed through the designed FFT.

CLK100MHZ 1												
16 dk_50 1												
1 dk_200					nnn							
- counter[4:0] 22		) 14		15	16	17	18	19	X 20	21	22	23
- w counter 1[4:0] 14		χ 6		vele	s 3rd Stag	e Ping Por	ng Buffer c	utputting	12	13	14	15
- w counter 2[4:0] 17		χ 9						urburning	15	X 16	17	18
20 counter3(4:0)		X 12		13	X 14	X 15	X 16	X 17	X 18	X 19	20	21
Xq_TopLevel[17:0] 0.0				_				t u				
XqOut_PingPong_LvL1w_Real[17:0] 0.0			1.	0		X			0.0			
XqOut_PingPong_LvL1w_Imag[17:0] 1.525	87890625e-0						1.525878	90625e-05				
XqOut_PingPong_LvL2w_Real[17:0] 1.0			0.0		0.5	X	0.0		X 0.5	X	1.0	
XqOut_PingPong_LvL2w_Imag[17:0] 0.0							0	. 0				
XqOut_PingPong_LvL3w_Real[17:0] 0.0		0.25	X	.0	0.25	X			0.0			
XqOut_PingPong_LvL3w_Imag[17:0] 0.0							0	. 0				
XqOut_PingPong_LvL4w_Real[17:0] 0.0		X-0.0883636	\$4D 1	0	Х			0.0				0.125
		0.08836364	70	<u></u>	<b>D</b> ' <b>D</b>	D ((		60 B.O.				
XqOut_BFM_LvL1w_Real[17:0] 0.0		χ 1.0	4th	Sta	ge Ping Pc	ng Butter	outpuțținț	g {1, 0, 0, 0	<b>υ, υ, υ, υ, ι</b>	0.5		.0
XqOut_BFM_LvL1w_Imag[17:0] 0.0							0	0				
XqOut_BFM_LvL2w_Real[17:0] 0.0					0.0			0.25	-0.25	X	0.0	
- XqOut_BFM_LvL2w_Imag[17:0] 0.0							0.0					-0.25
XqOut_BFM_LvL3w_Real[17:0] 0.0		X					0.125					. 0
- XqOut_BFM_LvL3w_Imag[17:0] 0.0							0	. 0				
- W_lvl1_Real[17:0] 1.0							1	. 0				
- W_lvl1_imag[17:0] 0.0							0	. 0				
W W2 Real[17:0] 0.0		0.0				1.0		X		0.0		1.0
						0.0		T Y		-1.0		0.0
		-1.0	X									
-1.0	0159912109		0159912109	38	X	0.0	X -0.70	015991210938	X	1.0	0.70701	991210938

The FFT was demonstrated successfully with Test Vector-1 in Chapter V. Timing was key to successful synchronization. Multiple counters were utilized to synchronize the stages of the FFT.

## C. TIMING ERROR WHILE INTEGRATING

An analysis of Test Vector 3 proved unsuccessful. While individually analyzing the processing elements of the FFT proved successful, synchronizing the timing between the three levels of the BFM failed. Highlighted in Figure 74 are a few locations where incorrect timing can skew processing of data. This improperly processed data then replicates through the successive stages. With the error being localized to within a timing element, initialization of the BFM and setting the selectors on the multiplexer or adder / subtracter can be as simple as a one-bit inversion; however, isolating the exact bit has proven to be a challenge. A complete overhaul of the timing within the FFT is the recommended corrective action. This has the greatest chance of success while also producing a code that is usable by others.

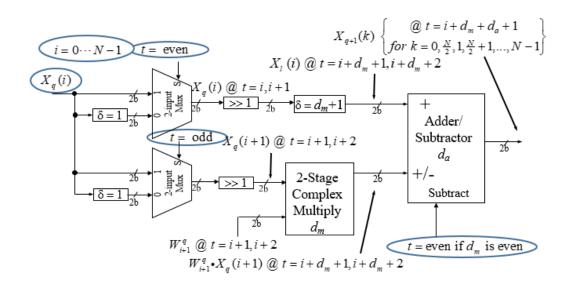


Figure 74. Highlighted Timing Variables within the BFM. Adapted from [15].

THIS PAGE INTENTIONALLY LEFT BLANK

## VI. CONCLUSION

With 20 years of research into satellite development, in this thesis research we are closer to creating a rapidly deployable and reconfigurable signal processor that is space capable. Signal processing may be the basis for a mission element or an enabling component of a larger objective. It also has utility in every day communications.

## A. ACADEMIC VALUE

An understanding of many academic areas was required for this research. A foundation in digital signal processing required a solid foundation in the Fast Fourier Transform theorems, principles, and mathematics. In addition, to conceptualize the final product, an understanding of the electronics needed to digitize an analog signal is required. With space being the intended application of the end product, knowledge of the space environment and its impact on electronics is also fundamental. A working knowledge of producing software code and an additional knowledge specifically in Verilog HDL was acquired throughout this research.

#### **B.** THESIS SUMMARY

The objective of this thesis was to give access to an open source signal processing algorithm that could eventually be implemented in an error detecting / error correcting algorithm for use within the space environment. Success was achieved in programming major portions of the highspeed pipelined FFT. The major modules needed to instantiate the FFT have been designed and tested. The FFT fails to integrate correctly. Timing and synchronization of data is the suspected cause. Once timing and synchronization is corrected, implementing Parseval's theorem will make this code space capable.

## C. RECOMMENDATIONS FOR FUTURE WORK

Follow on work should consist of synchronizing the timing within the FFT and then utilization of the FFT to implement Parseval's theorem. Parseval's theorem makes this code space capable and is required before loading on to the FPGA for launch. Parseval's theorem is illustrated once more in Figure 75.

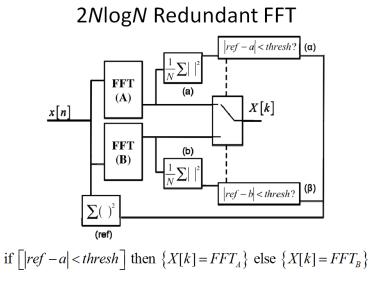


Figure 75. Parseval's Theorem Implementation Illustration. Source: [2].

In addition, capability can be added to the FFT by increasing the sample size from N = 8 to N = 32. This will allow a more granular level of processing. This FFT will also need to be loaded onto NPSat-1's FPGA with operability testing to follow.

#### D. CLOSING REMARKS

In clocked systems, timing and synchronization of data is mandatory. Processing elements must have the right data at the right time to produce the correct results. When debugging software, have a plan and work in a systematic process. This research project was built on 20 years of work. A literature review was helpful and there are many documents to educate oneself.

## **APPENDIX A. FILE STRUCTURE**

top\_level (top\_level.v) clk mod-clk module (clk module.xci) clk\_module (clk\_module.v) inst - clk\_module\_clk\_wiz (clk\_module\_clk\_wiz.v) PingPong\_SwitchLevel1 – pingpong (pingpong.v) BFMlevelOne - Radix2\_BFM (Radix2\_BFM.v) two2oneMux\_Top - two2oneMux (two2oneMux.v) two2oneMux\_Bottom - two2oneMux (two2oneMux.v) ComplexMultiply2Stage – multiply\_complex (multiply\_complex.v) Unsigned multiply – mult18x18 (mult18x18.v) twoC\_leftReal - twoCompRedo (twoCompRedo.v) twoC\_rightReal - twoCompRedo (twoCompRedo.v) twoC\_leftImag - twoCompRedo (twoCompRedo.v) twoC\_rightImag - twoCompRedo (twoCompRedo.v) insta – twoComp (twoComp.v) instb - twoComp (twoComp.v) instc - twoComp (twoComp.v) instd - twoComp (twoComp.v) AdderandSubtractor1Stage - addsub18x18 (addsub18x18.v) PingPong SwitchLevel2 – pingpongK (pingpongK.v) BFMlevelTwo – Radix2\_BFM (Radix2\_BFM.v) ... as shown above PingPong\_SwitchLevel3 - pingpongK (pingpongK.v) BFMlevelThree – Radix2\_BFM (Radix2\_BFM.v) ... as shown above

PingPong\_SwitchLevel4 - pingpongK (pingpongK.v)

THIS PAGE INTENTIONALLY LEFT BLANK

## **APPENDIX B. SOURCE CODE**

```
module top_level(
  16
                         input wire CLK100MHZ
 3);
4
          /*
(* dont_touch = "true" *)wire clk_50, clk_200, clk_400;
(* dont_touch = "true" *)reg [4:0] counter, counter], counter3;
(* dont_touch = "true" *)reg [17:0] driver;
(* dont_touch = "true" *)reg [17:0] XqJopLev1;
(* dont_touch = "true" *)wire [17:0] XqJopLev1;
(* dont_touch = "true" *)reg [17:0] W_1V1] Real; M_1V1] Imag;
(* dont_touch = "true" *)reg [17:0] W_1V1] Real; M_1V1] Imag;
(* dont_touch = "true" *)reg [17:0] W_1V1] Real; M_1V1] Imag;
(* dont_touch = "true" *)reg [17:0] W_1V1] Real; M_1V1] Imag;
(* dont_touch = "true" *)reg [17:0] W_1V1] Real; M_1V1] Imag;
(* dont_touch = "true" *)reg [17:0] W_1V1] Real; M_1V1] Imag;
(* dont_touch = "true" *)reg [17:0] W_1V1] Real; M_1V1] Imag;
(* dont_touch = "true" *)reg [17:0] W_1V1] Real; M_1V1] Imag;
(* dont_touch = "true" *)reg [17:0] W_1V1] Real; M_1V1] Imag;
(* dont_touch = "true" *)reg [17:0] W_1V1] Real; M_1V1] Imag;
(* dont_touch = "true" *)reg [17:0] W_1V1] Real; M_1V1] Imag;
(* dont_touch = "true" *)reg [17:0] W_1V1] Real; M_1V1] Imag;
(* dont_touch = "true" *)reg [17:0] W_1V1] Real; M_1V1] Imag;
(* dont_touch = "true" *)reg [17:0] W_1V1] Real; M_1V1] Imag;
(* dont_touch = "true" *)reg [17:0] W_1V1] Real; M_1V1] Imag;
(* dont_touch = "true" *)reg [17:
   5
// The following must be inserted into your Verilog file for this
// core to be instantiated. Change the instance name and port connections
                          // (in parentheses) to your own signal names.
                           //----- Clock INSTANTIATION ------
                               clk_module clk_mod
                                  (
// Clock in p
                                     .clk_inl(CLK100MHZ), // input clk
                                   .clk_inl(CLK100HB2), // input clk
// Clock out ports
.clk_50(clk_50), // output clk_50
.clk_200(clk_400), // output clk_200
.clk_400(clk_400), // output clk_400
// Status and control signals
                                     .reset(reset),
.locked(locked)
                         );
// ----- End Clock INSTANTIATION ------
                                           ---- Begin Ping-pong switch Level 1 -----
                                   pingpong PingPong_SwitchLevell
                                                (
.XqIn_PingPong_Real(Xq_TopLevel),
                                                .XqIn_PingPong_Imag(1),
                                                .clk(clk 50),
                                                  .counter(counter).
                                                  .XqOut_PingPongw_Real(XqOut_PingPong_LvLlw_Real),
.XqOut_PingPongw_Imag(XqOut_PingPong_LvLlw_Imag)
                                               );
--- End Ping-pong switch Level 1 ---
                                                  -- Begin Radix2_BFM Level Q = 1 ------
                                   Radix2_BFM BFMlevelOne
                                                (
.Xq_Real(XqOut_PingPong_LvLlw_Real),
.Xq_Imag(XqOut_PingPong_LvLlw_Imag),
.clk(clk_50),
.W_Real(W_lvll_Real),
                                                 .W_Imag(W_lvll_Imag),
                                                 .counterw(counterl),
.Radix2_BFMOut_Real(XqOut_BFM_LvLlw_Real),
.Radix2_BFMOut_Imag(XqOut_BFM_LvLlw_Imag)
                                                );
                                                          End Radix2_BFM Level Q = 1 -----
                                  ----- Begin Ping-pongK switch Level 2 -----
pingpongK PingPong_SwitchLevel2
                                                );
                                                   - End Ping-pongK switch Level 2 -----
                                                   - Begin Radix2 BFM Level Q = 2 ------
                                    Radix2_BFM BFMlevelTwo
                                                 .Xq_Real(XqOut_PingPong_LvL2w_Real),
.Xq_Imag(XqOut_PingPong_LvL2w_Imag),
                                                .Rq_lmag(XqOut_PingPong_LVLDw_lmag),
.clk(clt_S0),
.W_Beal(W_LV12_Real),
.W_Imag(W_LV12_Inag),
.counterw(counter2),
.Radix2_BFMOut_Real(XqOut_BFM_LVL2w_Real),
.Radix2_BFMOut_Rag(XqOut_BFM_LVL2w_lmag)
                                                 );
```

3	// Begin Ping-pong switch Level 3
4 5	pingpongK PingPong_SwitchLevel3
6	.xqIn_PingPong_Real(XqOut_BFM_LvL2w_Real), // input Xq real component
7	.XqIn_PingPong_Imag(XqOut_BFM_LvL2w_Imag), // input Xq imag component
	.clk(clk 50), // input clock .counter(counter2), // input counter
	.XqOut_FingPongw_Real(XqOut_FingPong_LvL3w_Real), // output Xq real component on a wire
	.XqOut_PingPongw_Imag(XqOut_PingPong_LvL3w_Imag) // output Xq imag component on a vire
	); // End Ping-pong switch Level 3
	// ind Fing-pong Bvitch Level 3
	// Begin Radix2_BFM Level Q = 3
	Radix2_BFM BFMlevelThree
	( .Xq_Real(XqOut_PingFong_LvL3v_Real),
	.Xq_keat.kqdvu_r=hingtong_vvsw_keat), .Xq_mag(Xqdvu_r=hingtong_vvsw_keat),
	.clk(clk_50),
L	.W_Real(W_lv13_Real),
	.W Imag(W lvl3 Imag), .counterw(counter3).
4	.Radix2_BFMout_Real(XqOut_BFM_LvL3w_Real),
5	.Radix2_BFMOut_Imag(XqOut_BFM_LvL3w_Imag)
	);
	// Begin Radix_EFM Level Q = 3
•	// Begin Ping-pong switch Level 4
	pingpongK PingFong_SwitchLevel4
	( Vie DiseRess Ball/Sobre EDV (right Ball) // journe Ve wal semanant
	.XqIn_PingPong_Real(XqOut_BFM_LvL3w_Real), // input Xq real component .XqIn_PingPong_Imag(XqOut_BFM_LvL3w_Imag), // input Xq imag component
	.clk(clk_50), // input clock
5	counter(counter), // input counter
5	.XqOut_PingPongw_Real(XqOut_PingPong_LvL4w_Real), // output Xq real component on a vire .XqOut_PingPongw_Imag(XqOut_PingPong_LvL4w_Imag) // output Xq imag component on a vire
	); };
	// End Ping-pong switch Level 4
	( Tribial black (Main summad and the simulation shoul)
÷	/ Initial block (Only executed once when simulation start) initial begin
1	counter = -1;
	counter1 = -9;
	counter2 = -6; counter3 = -3;
	INIT = 0;
3	reset = 0;
	Xq_TopLevel = 18'b010000000000000;
	W_lvllReal = 1; W lvllmag = 1;
2	W_1v12_Real = 1;
3	W_1v12_Imag = 1;
5	W_1v13_Real = 1; W_1v13_Imag = 1;
	end
7	
11	/ Statements below vill be executed on every clock rising edge
	lways8(posedge clk_50)
ιė	begin
÷ 🗄 –	case (counter[4:0])
e e	5'b00000: //Xq(0)
Y	begin Xq_TopLevel <= 18'b0100000000000000; // 1
5 Å	end
2	5'b00001: //Xq(1)
3 († 9 (	begin Xq_TopLevel <= 18'bb10000000000000; // 1
6	Ad topLevel <= 10-Dollowowowowowowow // 1 end
ιψ	5'b00010: //Xq(2)
ę	begin
	Xq_TopLevel <= 18'b010000000000000000000000000000000000
¢.	5'b00011: //Xq(3)
ę.	begin
	Xq_TopLevel <= 18'b010000000000000000000000000000000000
8	ena 5'b00100: //Xq(4)
ψ.	begin
	Xq_TopLevel <= 18'b01000000000000000; // 1
	end
зĘ	5'b0101: //Xq(5) becin
2 合 3 同 4 同 5	5'b0101: //Xq(5) begin Xq_TopLevel <= 18'b0100000000000000; // 1
	begin Xg_TopLevel <= 18'b010000000000000; // 1 end
	begin Xq_TopLevel <= 18*b010000000000000000000000000000000000
HO-D	begin Xg_TopLevel <= 18'b010000000000000; // 1 end

5'b00111: //Xg(7)	
begin	
Xq_TopLevel <= 18'b01000000000000000; // 1 end	
5'b01000: //Xq(8)	
begin	
Xq_TopLevel <= 18'b0000000000000000; // 0	
end	
5'b01001: //Xq(9)	
begin	
Xq_TopLevel <= 18'b00000000000000000; // 0 end	
5'b01010: //Xg(10)	
begin	
Xq_TopLevel <= 18'b00000000000000000; // 0	
end	
5'b01011: //Xq(11)	
begin	
Xq_TopLevel <= 18'b000000000000000000000000000000000000	
5'b01100: //Xq(12)	
begin	
Xq_TopLevel <= 18'b00000000000000000; // 0	
end	
5'b01101: //Xq(13)	
begin Xq_TopLevel <= 18'b00000000000000000; // 0	
<pre>kd_lopLevel &lt;= 18.B00000000000000000; // 0 end</pre>	
5'b01111: //Xq(15)	
begin	
Xq_TopLevel <= 18'b0000000000000000; // 0	
end	
5'b10000: //Xq(16)	
begin Xq TopLevel <= 18'b00000000000000000; // 0	
end	
5'b10001: //Xq(17)	
begin	
Xq_TopLevel <= 18'b00000000000000000; // 0	
end	
5'b10010: //Xq(18) begin	
Xq_TopLevel <= 18'b000000000000000000; // 0	
end	
end 5'b10011: //Xq(19)	
end 5'b10011: //Xq(19) begin	
end 5'blobl1: //Xq(19) begin Xq_TopLevel <= 18'b00000000000000000; // 0	
end 5'b10011: //Xq(19) begin Xq_TopLevel <= 18'b000000000000000; // 0 end	
<pre>end 5'bl0011: //Xq(19) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end 5'b10100: //Xq(20) begin</pre>	
<pre>end 5'bl0011: //Xq(19) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end 5'b10100: //Xq(20) begin</pre>	
<pre>end S'blo01: //Xq(19) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end S'bl010: //Xq(20) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end</pre>	
<pre>end 5'b10011: //Xq(19) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'b1010: //Xq(20) begin Xq_TopLevel &lt;= 18'b0000000000000; // 0 end 5'b10101: //Xq(21)</pre>	
<pre>end 5'bl001: //Xq(19) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end Xq_TopLevel &lt;= 18'b00000000000000; // 0 begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl0010: //Xq(21) begin</pre>	
<pre>end 5'b1001: //Xq(19) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end 5'b1010: //Xq(20) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'b10101: //Xq(21) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0</pre>	
<pre>end 5'bl0011: //Xq(19) bg1n Xq_TopLevel &lt;= 18'b000000000000000; // 0 end 5'bl0100: //Xq(20) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end 5'bl0101: //Xq(21) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end</pre>	
<pre>end 5'bl0011: //Xq(19) bg1n Xq_TopLevel &lt;= 18'b000000000000000; // 0 end 5'bl0100: //Xq(20) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end 5'bl0101: //Xq(21) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl0101: //Xq(22) begin</pre>	
<pre>end 5'bl001: //Xq(19) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl0101: //Xq(21) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl0101: //Xq(22) begin Xq_TopLevel &lt;= 18'b0000000000000; // 0</pre>	
<pre>end 5'b10011; //Xq(19) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end 5'b10101; //Xq(20) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'b10101; //Xq(21) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'b10101; //Xq(22) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end</pre>	
<pre>end 5'bl001: //Xq(19) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl0101: //Xq(21) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl0110: //Xq(22) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl0111: //Xq(23)</pre>	
<pre>end 5'b1001: //Xq(19) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end Xq_TopLevel &lt;= 18'b0000000000000000; // 0 end 5'b10101: //Xq(21) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'b1010: //Xq(22) begin Xq_TopLevel &lt;= 18'b0000000000000; // 0 end 5'b1011: //Xq(23) begin</pre>	
<pre>end 5'b1001: //Xq(19) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end Xq_TopLevel &lt;= 18'b000000000000000; // 0 end 5'b1010: //Xq(21) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end 5'b1010: //Xq(22) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'b1011: //Xq(23) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'b1011: //Xq(23) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end </pre>	
<pre>end 5'b10011; //Xq(19) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end 5'b10101; //Xq(20) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end 5'b10101; //Xq(21) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'b10101; //Xq(23) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end 5'b10111; //Xq(23) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'b10111; //Xq(23) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'b10111; //Xq(23) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'b10111; //Xq(23) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end 5'b10111; //Xq(23) begin Xq_TopLevel &lt;= 18'b0000000000000000; // 0</pre>	
<pre>end 5'bl001: //Xq(19) begin Xq_TopLevel &lt;= 18'b0000000000000000; // 0 end Xq_TopLevel &lt;= 18'b000000000000000; // 0 end 5'bl0101: //Xq(21) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end 5'bl010: //Xq(22) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl011: //Xq(23) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl010: //Xq(24) begin </pre>	
<pre>end 5'bl001: //Xq(19) begin % x_TopLevel &lt;= 18'b000000000000000; // 0 end % TopLevel &lt;= 18'b000000000000000; // 0 end 5'bl010: //Xq(21) begin % TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl011: //Xq(22) begin % X_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl011: //Xq(23) begin % X_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl011: //Xq(24) begin % X_TopLevel &lt;= 18'b00000000000000; // 0 end</pre>	
<pre>end 5'bl001: //Xq(19) kqInp </pre> %2_TopLevel <= 18'b000000000000000; // 0 end %2_TopLevel <= 18'b000000000000000; // 0 end 5'bl010: //Xq(21) begin Xq_TopLevel <= 18'b00000000000000; // 0 end 5'bl010: //Xq(22) begin Xq_TopLevel <= 18'b00000000000000; // 0 end 5'bl011: //Xq(23) begin Xq_TopLevel <= 18'b00000000000000; // 0 end 5'bl010: //Xq(24) begin Xq_TopLevel <= 18'b00000000000000; // 0 end 5'bl000: //Xq(24) begin Xq_TopLevel <= 18'b00000000000000; // 0 end 5'bl000: //Xq(24) begin Xq_TopLevel <= 18'b00000000000000; // 0 end 5'bl000: //Xq(24) begin Xq_TopLevel <= 18'b00000000000000; // 0 end	
<pre>end 5'bl001: //Xq(19) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end Xq_TopLevel &lt;= 18'b000000000000000; // 0 end 5'bl0101: //Xq(21) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl010: //Xq(22) begin Xq_TopLevel &lt;= 18'b0000000000000; // 0 end 5'bl011: //Xq(23) begin Xq_TopLevel &lt;= 18'b0000000000000; // 0 end 5'bl011: //Xq(24) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl0100: //Xq(24) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl0100: //Xq(24) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl000: //Xq(24) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end 5'bl000: //Xq(25)</pre>	
<pre>end 5'bl001: //Xq(19) kqInp </pre> %2_TopLevel <= 18'b000000000000000; // 0 end %2_TopLevel <= 18'b000000000000000; // 0 end 5'bl010: //Xq(21) begin Xq_TopLevel <= 18'b00000000000000; // 0 end 5'bl010: //Xq(22) begin Xq_TopLevel <= 18'b00000000000000; // 0 end 5'bl011: //Xq(23) begin Xq_TopLevel <= 18'b00000000000000; // 0 end 5'bl010: //Xq(24) begin Xq_TopLevel <= 18'b00000000000000; // 0 end 5'bl000: //Xq(24) begin Xq_TopLevel <= 18'b00000000000000; // 0 end 5'bl000: //Xq(24) begin Xq_TopLevel <= 18'b00000000000000; // 0 end 5'bl000: //Xq(24) begin Xq_TopLevel <= 18'b00000000000000; // 0 end	
<pre>end 5'bl001: //Xq(19) begin Xq_TopLevel &lt; 18'b00000000000000; // 0 end Xq_TopLevel &lt; 18'b00000000000000; // 0 end 5'bl001: //Xq(21) begin Xq_TopLevel &lt; 18'b00000000000000; // 0 end 5'bl010: //Xq(22) begin Xq_TopLevel &lt; 18'b00000000000000; // 0 end 5'bl010: //Xq(23) begin Xq_TopLevel &lt; 18'b00000000000000; // 0 end 5'bl100: //Xq(23) begin Xq_TopLevel &lt; 18'b00000000000000; // 0 end 5'bl100: //Xq(25) begin Xq_TopLevel &lt; 18'b00000000000000; // 0 end 5'bl100: //Xq(25) begin Xq_TopLevel &lt; 18'b00000000000000; // 0 end 5'bl100: //Xq(25) begin Xq_TopLevel &lt; 18'b00000000000000; // 0 end 5'bl100: //Xq(25) begin Xq_TopLevel &lt; 18'b00000000000000; // 0 end 5'bl100: //Xq(25) begin Xq_TopLevel &lt; 18'b00000000000000; // 0 end 5'bl100: //Xq(25) begin Xq_TopLevel &lt; 18'b00000000000000; // 0 end 5'bl100: //Xq(25) begin Xq_TopLevel &lt; 18'b00000000000000; // 0 end 5'bl100: //Xq(25) begin Xq_TopLevel &lt; 18'b00000000000000; // 0 end 5'bl100: //Xq(25) begin Xq_TopLevel &lt; 18'b000000000000000; // 0 end 5'bl100: //Xq(25) begin Xq_TopLevel &lt; 18'b000000000000000; // 0 end 5'b1000000000000000000000; // 0 end 5'b1000000000000000000000000000000000000</pre>	
<pre>end 5'bl001: //Xq(19) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end Xq_TopLevel &lt;= 18'b000000000000000; // 0 end 5'bl1001: //Xq(21) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end 5'bl101: //Xq(22) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl1011: //Xq(23) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl1001: //Xq(24) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl1001: //Xq(25) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl1001: //Xq(25) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end</pre>	
<pre>end 5'bl001: //Xq(19) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end Xq_TopLevel &lt;= 18'b000000000000000; // 0 end 5'bl0101: //Xq(21) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl0110: //Xq(22) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl0101: //Xq(23) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl1001: //Xq(23) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl1001: //Xq(25) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl1001: //Xq(25) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl1001: //Xq(25) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl1001: //Xq(25) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end 5'bl1001: //Xq(25) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end 5'bl1001: //Xq(25) begin Xq_TopLevel &lt;= 18'b00000000000000000; // 0 end 5'bl1001: //Xq(25) begin Xq_TopLevel &lt;= 18'b00000000000000000; // 0 end 5'bl1001: //Xq(25) begin Xq_TopLevel &lt;= 18'b00000000000000000; // 0 end 5'bl1001: //Xq(25) begin Xq_TopLevel &lt;= 18'b000000000000000000000000000000000000</pre>	
<pre>end 5'bl001: //Xq(19) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end Yq_TopLevel &lt;&lt; 18'b000000000000000; // 0 end 5'bl101: //Xq(21) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl101: //Xq(22) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl11: //Xq(23) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl1001: //Xq(24) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl1001: //Xq(25) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl1001: //Xq(25) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl1001: //Xq(25) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl1001: //Xq(25) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl1001: //Xq(25) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end 5'bl1001: //Xq(25) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end 5'bl1001: //Xq(25) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end 5'bl1001: //Xq(25) begin Xq_TopLevel &lt;= 18'b00000000000000000; // 0 end 5'bl1001: //Xq(25) begin Xq_TopLevel &lt;= 18'b00000000000000000; // 0 end 5'bl1001: //Xq(25) begin Xq_TopLevel &lt;= 18'b0000000000000000; // 0 end 5'bl1001: //Xq(25) begin Xq_TopLevel &lt;= 18'b0000000000000000; // 0 end 5'bl1001: //Xq(25) begin Xq_TopLevel &lt;= 18'b0000000000000000; // 0 end 5'bl1001: //Xq(25) begin Xq_TopLevel &lt;= 18'b00000000000000000; // 0 end 5'bl1001: //Xq(25) begin Xq_TopLevel &lt;= 18'b0000000000000000000; // 0 end 5'bl1001: //Xq(25) begin Xq_TopLevel &lt;= 18'b000000000000000000000; // 0 end 5'bl1001: //Xq(25) begin Xq_TopLevel &lt;= 18'b000000000000000000000000000000000000</pre>	
<pre>end 5'bl001: //Xq(19) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end %X_TopLevel &lt;= 18'b000000000000000; // 0 end 5'bl001: //Xq(21) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl010: //Xq(22) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl010: //Xq(23) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl010: //Xq(24) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl100: //Xq(25) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl100: //Xq(26) begin Xq_TopLevel &lt;= 18'b0000000000000; // 0 end 5'bl100: //Xq(26) begin Xq_TopLevel &lt;= 18'b0000000000000; // 0 end 5'bl100: //Xq(26) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl100: //Xq(26) begin Xq_TopLevel &lt;= 18'b00000000000000; // 1 end 5'bl100: //Xq(26) begin Xq_TopLevel &lt;= 18'b000000000000000; // 1 end 5'bl100: //Xq(26) begin Xq_TopLevel &lt;= 18'b000000000000000; // 1 end 5'bl100: //Xq(26) begin Xq_TopLevel &lt;= 18'b000000000000000; // 1 end 5'bl100: //Xq(26) begin Xq_TopLevel &lt;= 18'b0000000000000000; // 1 end 5'bl100: //Xq(26) begin Xq_TopLevel &lt;= 18'b000000000000000000000000000000000000</pre>	
<pre>end 5'bl001: //Xq(13) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end Xq_TopLevel &lt;= 18'b000000000000000; // 0 end 5'b10101: //Xq(21) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'b10101: //Xq(22) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'b10101: //Xq(23) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'b10101: //Xq(25) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'b10101: //Xq(26) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'b10101: //Xq(26) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'b10101: //Xq(26) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'b10101: //Xq(27) begin Xq_TopLevel &lt;= 18'b00000000000000; // 1 end 5'b10101: //Xq(27) begin</pre>	
<pre>end 5'bl001: //Xq(19) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end Xq_TopLevel &lt;= 18'b000000000000000; // 0 end 5'bl0101: //Xq(21) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end 5'bl0101: //Xq(22) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl0101: //Xq(22) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl0001: //Xq(23) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl1001: //Xq(25) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl1001: //Xq(25) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl1001: //Xq(25) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl1001: //Xq(25) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl1011: //Xq(25) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl1011: //Xq(25) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end 5'bl1011: //Xq(25) begin Xq_TopLevel &lt;= 18'b0000000000000000; // 0 end 5'bl1011: //Xq(25) begin Xq_TopLevel &lt;= 18'b0000000000000000; // 0 end 5'bl1011: //Xq(25) begin Xq_TopLevel &lt;= 18'b00000000000000000; // 0 end 5'bl1011: //Xq(25) begin Xq_TopLevel &lt;= 18'b000000000000000000; // 0 end 5'bl1011: //Xq(25) begin Xq_TopLevel &lt;= 18'b00000000000000000000000000; // 0 end 5'bl1011: //Xq(25) begin Xq_TopLevel &lt;= 18'b000000000000000000000; // 0 end 5'bl1011: //Xq(25) begin Xq_TopLevel &lt;= 18'b000000000000000000000000000000000000</pre>	
<pre>end 5'bl001: //Xq(19) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'b1001: //Xq(20) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'b10101: //Xq(22) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'b10111: //Xq(23) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'b10101: //Xq(23) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'b10101: //Xq(23) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'b1001: //Xq(24) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'b1101: //Xq(27) begin Xq_TopLevel &lt;= 18'b010000000000000; // 1 end 5'b1101: //Xq(27) begin Xq_TopLevel &lt;= 18'b010000000000000; // 1 end 5'b1101: //Xq(27) begin Xq_TopLevel &lt;= 18'b010000000000000; // 1 end 5'b1101: //Xq(27) begin Xq_TopLevel &lt;= 18'b010000000000000; // 0 end 5'b11011: //Xq(27) begin Xq_TopLevel &lt;= 18'b010000000000000; // 0 end 5'b11011: //Xq(27) begin Xq_TopLevel &lt;= 18'b0100000000000000; // 0 end 5'b11011: //Xq(27) begin Xq_TopLevel &lt;= 18'b010000000000000000; // 0 end 5'b11011: //Xq(27) begin Xq_TopLevel &lt;= 18'b01000000000000000; // 0 end 5'b11011: //Xq(27) begin Xq_TopLevel &lt;= 18'b01000000000000000; // 0 end 5'b11011: //Xq(27) begin Xq_TopLevel &lt;= 18'b01000000000000000000000000; // 0 end 5'b11011: //Xq(27) begin Xq_TopLevel &lt;= 18'b010000000000000000000000000000000000</pre>	
<pre>end 5'bl001: //Xq(19) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end %q_TopLevel &lt;= 18'b000000000000000; // 0 end 5'bl010: //Xq(21) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl010: //Xq(22) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl010: //Xq(23) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl100: //Xq(24) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl100: //Xq(25) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl100: //Xq(26) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl100: //Xq(26) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl100: //Xq(26) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl100: //Xq(26) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl100: //Xq(27) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end 5'bl100: //Xq(27) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end 5'bl100: //Xq(27) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end 5'bl100: //Xq(27) begin Xq_TopLevel &lt;= 18'b0000000000000000; // 0 end 5'bl100: //Xq(27) begin Xq_TopLevel &lt;= 18'b0000000000000000; // 0 end 5'bl100: //Xq(27) begin Xq_TopLevel &lt;= 18'b0000000000000000000; // 0 end 5'bl100: //Xq(27) begin Xq_TopLevel &lt;= 18'b0000000000000000000; // 0 end 5'bl100: //Xq(28) </pre>	
<pre>end 5'bl001: //Xq(19) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end Xq_TopLevel &lt;= 18'b000000000000000; // 0 end 5'bl1001: //Xq(21) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end 5'bl101: //Xq(22) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end 5'bl1011: //Xq(23) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end 5'bl1001: //Xq(23) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end 5'bl1001: //Xq(24) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end 5'bl1001: //Xq(25) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end 5'bl1001: //Xq(25) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end 5'bl1001: //Xq(25) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl1001: //Xq(25) begin Xq_TopLevel &lt;= 18'b0100000000000000; // 0 end 5'bl1011: //Xq(25) begin Xq_TopLevel &lt;= 18'b0100000000000000; // 0 end 5'bl1011: //Xq(25) begin Xq_TopLevel &lt;= 18'b0100000000000000; // 0 end 5'bl1011: //Xq(25) begin Xq_TopLevel &lt;= 18'b0100000000000000; // 0 end 5'b11011: //Xq(25) begin Xq_TopLevel &lt;= 18'b01000000000000000; // 0 end 5'b11011: //Xq(25) begin Xq_TopLevel &lt;= 18'b0100000000000000; // 0 end 5'b1101: //Xq(25) begin Xq_TopLevel &lt;= 18'b0100000000000000; // 0 end 5'b11011: //Xq(25) begin Xq_TopLevel &lt;= 18'b0100000000000000; // 0 end 5'b11011: //Xq(25) begin Xq_TopLevel &lt;= 18'b01000000000000000; // 0 end 5'b11011: //Xq(25) begin Xq_TopLevel &lt;= 18'b01000000000000000; // 0 end 5'b11011: //Xq(25) begin Xq_TopLevel &lt;= 18'b01000000000000000; // 0 end 5'b11011: //Xq(25) begin Xq_TopLevel &lt;= 18'b01000000000000000; // 0 end 5'b11011: //Xq(27) begin Xq_TopLevel &lt;= 18'b000000000000000000000000000000000000</pre>	
<pre>end 5'bl001: //Xq(19) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end Xq_TopLevel &lt;&lt; 18'b000000000000000; // 0 end 5'bl0101: //Xq(21) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl1011: //Xq(22) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl1001: //Xq(23) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl1001: //Xq(23) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl1001: //Xq(24) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl1001: //Xq(25) begin Xq_TopLevel &lt;= 18'b00000000000000; // 1 end 5'bl1001: //Xq(25) begin Xq_TopLevel &lt;= 18'b00000000000000; // 1 end 5'bl1001: //Xq(25) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl1001: //Xq(25) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl1001: //Xq(27) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl1001: //Xq(28) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl1001: //Xq(28) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end 5'bl1001: //Xq(28) begin Xq_TopLevel &lt;= 18'b000000000000000000000; // 0 end 5'bl1001: //Xq(28) begin Xq_TopLevel &lt;= 18'b000000000000000000000000000000000000</pre>	
<pre>end 5'b1001: //Xq(13) begin Xq_TopLevel &lt; 18'b000000000000000; // 0 end %Tyb1001: //Xq(20) begin Xq_TopLevel &lt; 18'b000000000000000; // 0 end 5'b1010: //Xq(22) begin Xq_TopLevel &lt; 18'b00000000000000; // 0 end 5'b1010: //Xq(23) begin Xq_TopLevel &lt; 18'b00000000000000; // 0 end 5'b1100: //Xq(24) begin Xq_TopLevel &lt; 18'b00000000000000; // 0 end 5'b1100: //Xq(25) begin Xq_TopLevel &lt; 18'b00000000000000; // 0 end 5'b1100: //Xq(25) begin Xq_TopLevel &lt; 18'b0000000000000; // 0 end 5'b1100: //Xq(27) begin Xq_TopLevel &lt; 18'b00000000000000; // 0 end 5'b1100: //Xq(27) begin Xq_TopLevel &lt; 18'b00000000000000; // 0 end 5'b1100: //Xq(27) begin Xq_TopLevel &lt; 18'b00000000000000; // 0 end 5'b1100: //Xq(27) begin Xq_TopLevel &lt; 18'b00000000000000; // 0 end 5'b1100: //Xq(27) begin Xq_TopLevel &lt; 18'b00000000000000; // 0 end 5'b1100: //Xq(27) begin Xq_TopLevel &lt; 18'b00000000000000; // 0 end 5'b1100: //Xq(27) begin Xq_TopLevel &lt; 18'b00000000000000; // 0 end 5'b1100: //Xq(27) begin Xq_TopLevel &lt; 18'b000000000000000; // 0 end 5'b1100: //Xq(27) begin Xq_TopLevel &lt; 18'b000000000000000; // 0 end 5'b1100: //Xq(27) begin Xq_TopLevel &lt; 18'b000000000000000; // 0 end 5'b1100: //Xq(27) begin Xq_TopLevel &lt; 18'b000000000000000; // 0 end 5'b1100: //Xq(27) begin Xq_TopLevel &lt; 18'b000000000000000; // 0 end 5'b1100: //Xq(27) begin Xq_TopLevel &lt; 18'b0000000000000000; // 0 end 5'b1100: //Xq(27) begin Xq_TopLevel &lt; 18'b00000000000000000; // 0 end 5'b1100: //Xq(27) begin Xq_TopLevel &lt; 18'b00000000000000000; // 0 end 5'b1100: //Xq(28) </pre>	
<pre>end 5'bl001: //Xq(19) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end Xq_TopLevel &lt;&lt; 18'b000000000000000; // 0 end 5'bl0101: //Xq(21) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl1011: //Xq(22) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl1001: //Xq(23) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl1001: //Xq(23) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl1001: //Xq(24) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl1001: //Xq(25) begin Xq_TopLevel &lt;= 18'b00000000000000; // 1 end 5'bl1001: //Xq(25) begin Xq_TopLevel &lt;= 18'b00000000000000; // 1 end 5'bl1001: //Xq(25) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl1001: //Xq(25) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl1001: //Xq(27) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl1001: //Xq(28) begin Xq_TopLevel &lt;= 18'b00000000000000; // 0 end 5'bl1001: //Xq(28) begin Xq_TopLevel &lt;= 18'b000000000000000; // 0 end 5'bl1001: //Xq(28) begin Xq_TopLevel &lt;= 18'b000000000000000000000; // 0 end 5'bl1001: //Xq(28) begin Xq_TopLevel &lt;= 18'b000000000000000000000000000000000000</pre>	

271	Xq_TopLevel <= 18'b0000000000000000; // 0
272 ≙ 273 ₽	end
273 🖯	5'b11110: //Xq(30)
274 👳	begin
275	Xq_TopLevel <= 18'b0000000000000000; // 0
276	end
276 A	
211 9	///
278	begin
279	Xq_TopLevel <= 18'b0100000000000000; // 1
280	end
281	endcase
282	
	vel 1 Tviddle
284 UP 285 UP	case (counter1[2:0])
285 👳	3'b000: //Xq(0)
286 👳	begin
287	W_lv11_Real <= 18'b0100000000000000; // 1
288	W_lvll_Imag <= 18'b0000000000000000; // -j0
289	end
289 👌 290 🖯	3'b001: //Xq(1)
291	
291 🖯	begin
292	W_lvll_Real <= 18'b0100000000000000; // 1
293	W_lvll_Imag <= 18'b0000000000000000; // -j0
294 🛆	end
294 占 295 🖯	3'b010: //Xq(2)
296 🖯	begin
301	begin
302	
302	W_lvll_Real <= 18'b0100000000000000; // 1
	W_lvll_Imag <= 18'b0000000000000000; // -j0
304 ≙ 305 ⊖	end
305 👳	3'b100: //Xq(4)
306 👳	begin
307	W_lvll_Real <= 18'b0100000000000000; // 1
308	W_lvll_Real <= 18*b0100000000000000; // 1 W_lvll_Imag <= 18*b0000000000000000; // -j0
	end
309 Å 310 ⊕	3'b101: //Xq(5)
310 7	
311 🗄	begin
312	W_lv11_Real <= 18'b01000000000000000; // 1
313	W_lvll_Imag <= 18"b0000000000000000; // -j0
314 🛆	end
314 👌 315 🖯	3'b110: //Xq(6)
316 👳	begin
317	W_lvll_Real <= 18'b0100000000000000; // 1
318	W_1v11_Imag <= 18'b0000000000000000; // -j0
319 占 320 👳	end
320 🖓	3'b111: //Xq(7)
321 🗄	begin
322	W_lvll_Real <= 18'b0100000000000000; // 1
323	W_lvll_Imag <= 18'b0000000000000000; // -j0
324	end
325	endcase
326	chadabe
327 // Le	vel 2 Twiddle
328 👳	case (counter2[3:0])
329 🖗	4'b0000:
330 🗄	begin
331	
331	<pre>W_lvl2_Real &lt;= 18'b000000000000000000; // 0 W_lvl2_Imag &lt;= 18'b11000000000000000; // -j1</pre>
	W_1v12_Imag <= 18'b1100000000000000; // -j1
333 🖨 334 🛡	end
334 🤤	4'b0001:
335 👳	begin
336	W_lv12_Real <= 18'b01000000000000000; // 1
337	W_1v12_Imag <= 18'b00000000000000000; // -j0
	end
338 A 339 🖶	4'b0010:
222 7	
340 🖯	begin
341	<pre>W_lvl2_Real &lt;= 18'b01000000000000000; // 1</pre>
342	W_lv12_Imag <= 18'b000000000000000; // -j0
343 📄	end
343 🛆 344 🛡	4'b0011:
345 🔅	begin
346	W_lvl2_Real <= 18'b01000000000000000; // 1
	W 1v12_Real <= 18'B01000000000000000; // 1
347	
347 348 🖨 349 🖻	end
	4'b0100:
350 👳	begin
351	W 1v12 Real <= 18'b0100000000000000; // 1
352	<pre>W_lvl2_Real &lt;= 18'b01000000000000000; // 1</pre>
	w_1v12_1mag <= 18*b000000000000000000000000000000000000
353 🖕 354 👳	
354	4'b0101:
355 👳	begin
356	W_1v12_Real <= 18'b000000000000000000000000000000000000
357	W_lv12_Imag <= 18'b110000000000000; // -j1
0.00	end
358	4'b0110:
200 H	
360	begin

361	W_lv12_Real <= 18'b0000000000000000; // 0
362	W_lvl2_Imag <= 18'b11000000000000000; // -j1
363 ≙ 364 ⊕	end
	4'b0111:
365 🖯	begin
366	W_lv12_Real <= 18'b0000000000000000; // 0
367	W_1v12_Imag <= 18'b11000000000000000; // -j1
368 🖕 369 🖶	end 4'b1000:
370 ¢	4 biolo: begin
371	N 1212 Deal - 181b00000000000000000000000000000000000
372	W_lvl2_Real <= 18'b0000000000000000; // 0 W_lvl2_Imag <= 18'b100000000000000; // -j1
	end
373 🛓 374 🛡	4'b1001:
375 🔶	begin
376	W_1v12_Real <= 18'b0100000000000000; // 1
377	W_lv12_Imag <= 18'b00000000000000000; // -j0
378 占 379 🗟	end
379 🗄	4'b1010:
380 🗄 👘	begin
381	<pre>W_lvl2_Real &lt;= 18'b01000000000000000; // 1 W_lvl2_Imag &lt;= 18'b0000000000000000; // -j0</pre>
382	W_lv12_Imag <= 18'b00000000000000000; // -j0
383 🖕 384 🖶	end
384 🤤	4'b1011:
385 🖗	begin
386	W_1v12_Real <= 18'b010000000000000000; // 1 W_1v12_Real <= 18'b01000000000000000; // 1 W_1v12_Imag <= 18'b0000000000000000; // -j0
391	W_1v12_Real <= 18'b0100000000000000; // 1
392	W_lv12_Imag <= 18'b000000000000000; // -j0
393 ≙ 394 🖯	end
394 -	4'b1101:
395 🖯	begin
396 397	<pre>W_lvl2_Real &lt;= 18'b0000000000000000; // 0 W_lvl2_Imag &lt;= 18'b110000000000000; // -j1</pre>
	w_ivi2_imag <= 18.bi1000000000000000000000000000000000000
398 Å 399 €	4'b1110:
400 0	begin begin
401	
402	<pre>W_lvl2_Real &lt;= 18'b0000000000000000; // 0 W_lvl2_Imag &lt;= 18'b110000000000000; // -j1</pre>
	end
403 Å 404 ⊕	4'b1111:
405 🗄	begin
406	W_lvl2_Real <= 18'b0000000000000000; // 0
407	W_lv12_Imag <= 18'b1100000000000000; // -j1
408	end
409 🚖	endcase
410	
411 // L	vel 3 Twiddle
412	case (counter3[3:0])
412	case (counter3[3:0]) 4'b0000:
412 0 413 0 414 0	case (counter3[3:0]) 4'b0000: begin
412 0 413 0 414 0 415	<pre>case (counter3[3:0]) 4'b0000:     begin     #_uv13_Real &lt;= 18'b110100101100000001; // -sgrt(2)/2</pre>
412 0 413 0 414 0 415 415 416	<pre>case (counter3[3:0]) 4'b0000; begin W_ivi3_Real &lt;= 18'b110100101100000001; // -sqrt(2)/2 W_ivi3_Imag &lt;= 18'b110100101100000001; // -jaqrt(2)/2</pre>
412 0 413 0 414 0 415 415 416	<pre>case (counter3[3:0]) 4'b0000; begin W_iv13_Beal &lt;= 18'b11010010100000001; // -sqrt(2)/2 W_iv13_Imag &lt;= 18'b110100101100000001; // -jsqrt(2)/2 end</pre>
412 0 413 0 414 0 415 4 416 4 417 0 418 0	<pre>case (counter3[3:0]) 4'bb000; begin W_lv13_Real &lt;= 18'bl1010010100000001; // -sqrt(2)/2 W_lv13_Imag &lt;= 18'bl1010010100000001; // -jsqrt(2)/2 end 4'b0001;</pre>
412 0 413 0 414 0 415 4 416 4 417 0 418 0 419 0	<pre>case (counter3[3:0]) 4'b0000:     begin     W_ivl3_Real &lt;= 18'bil010010100000001; // -sqrt(2)/2     W_ivl3_Imag &lt;= 18'bil0100101100000001; // -jsqrt(2)/2     end 4'b0001:     begin</pre>
412 0 413 0 414 0 415 4 416 4 417 4 418 0 419 0 420	<pre>case (counter3[3:0]) 4'b0000:     begin     W_ivl3_Real &lt;= 18'bil010010100000001; // -sqrt(2)/2     W_ivl3_Imag &lt;= 18'bil0100101100000001; // -jsqrt(2)/2     end 4'b0001:     begin</pre>
412 0 413 0 414 0 415 4 416 4 417 0 418 0 419 0 420 4 421	<pre>case (counter3[3:0]) 4'b0000; begin W_Lvl3_Real &lt;= 18'b10100101100000001; // -sqrt(2)/2 W_Lvl3_Imag &lt;= 18'b10100101100000001; // -sqrt(2)/2 end 4'b0001; begin W_Lvl3_Real &lt;= 18'b0100000000000000; // 1 W_Lvl3_Imag &lt;= 18'b0100000000000000; // -j0</pre>
412 0 413 0 414 0 415 4 416 4 417 0 418 0 419 0 420 4 421 4 422 0 423 0	<pre>case (counter3[3:0]) 4'b0000:     begin     W_ivl3_Real &lt;= 18'bil010010100000001; // -sqrt(2)/2     W_ivl3_Imag &lt;= 18'bil0100101100000001; // -jsqrt(2)/2     end 4'b0001:     begin</pre>
412 0 413 0 414 0 415 4 416 4 417 0 418 0 419 0 420 4 421 4 422 0 423 0	<pre>case (counter3[3:0]) 4'b0000; begin W_lvl3_Real &lt;= 18'bli0100101100000001; // -sgrt(2)/2 W_lvl3_Imag &lt;= 18'bli0100101000000001; // -jsgrt(2)/2 end 4'b0001; begin W_lvl3_Real &lt;= 18'b0100000000000000; // 1 W_lvl3_Imag &lt;= 18'b010000000000000; // -j0 end 4'b0010;</pre>
412 0 413 0 414 0 415 416 416 417 0 418 0 419 0 420 1 422 0 423 0 423 0 423 0 423 0	<pre>case (counter3[3:0]) 4'b0000; begin W_lvl3_Real &lt;= 18'b10100101100000001; // -sgrt(2)/2 W_lvl3_Imag &lt;= 18'b101001010000000000; // -jsgrt(2)/2 end 4'b0001; begin W_lvl3_Imag &lt;= 18'b01000000000000000; // 1 W_lvl3_Imag &lt;= 18'b0100000000000000; // -j0 end 4'b0010; begin W_lvl3_Real &lt;= 18'b0100000000000000; // 1</pre>
412 0 413 0 414 0 415 416 416 417 0 418 0 419 0 420 1 421 422 0 423 0 424 0 423 0 424 0	<pre>case (counter3[3:0]) 4'b0000; begin W_lvl3_Real &lt;= 18'b10100101100000001; // -sgrt(2)/2 W_lvl3_Imag &lt;= 18'b101001010000000000; // -jsgrt(2)/2 end 4'b0001; begin W_lvl3_Imag &lt;= 18'b01000000000000000; // 1 W_lvl3_Imag &lt;= 18'b0100000000000000; // -j0 end 4'b0010; begin W_lvl3_Real &lt;= 18'b0100000000000000; // 1</pre>
412 0 413 0 414 0 415 4 416 4 417 0 419 0 421 422 0 421 422 0 423 0 424 4 425 426 4 425 426 4 427 0 427 0 427 0 427 0 427 0 428 0 429 0 420 0 400 0 40	<pre>case (counter3[3:0]) 4'b0000; begin W_lvl3_Real &lt;= 18'b11010010100000001; // -sqrt(2)/2 entl3_Imag &lt;= 18'b110100101100000001; // -jsqrt(2)/2 entl4'b0001; begin W_lvl3_Real &lt;= 18'b010000000000000; // 1 W_lvl3_Imag &lt;= 18'b010000000000000; // -j0 entl4'b0010; begin</pre>
412 0 413 0 414 0 415 416 417 0 418 0 420 420 420 422 0 422 0 423 0 423 0 425 426 0 427 0 428 0	<pre>case (counter3[3:0]) 4'b0000; begin W_lvl3_Beal &lt;= 18'b11010010100000001; // -sgrt(2)/2 w_lvl3_Imag &lt;= 18'b11010010100000001; // -sgrt(2)/2 end 4'b0001; begin W_lvl3_Beal &lt;= 18'b0100000000000000; // 1 W_lvl3_Imag &lt;= 18'b0100000000000000; // -j0 end 4'b0010; begin W_lvl3_Beal &lt;= 18'b0100000000000000; // 1 W_lvl3_Imag &lt;= 18'b0100000000000000; // 1)</pre>
$\begin{array}{c} 412 \\ + 12 \\ + 13 \\ + 14 \\ + 14 \\ + 15 \\ + 14 \\ + 15 \\ + 415 \\ + 415 \\ + 415 \\ + 415 \\ + 416 \\ + 417 \\ + 419 \\ + 422 \\ + 422 \\ + 422 \\ + 422 \\ + 422 \\ + 422 \\ + 425 \\ + 426 \\ + 428 \\ + 99 \\ +$	<pre>case (counter3[3:0]) 4'bomot: begin W_iv13_Real &lt;= 18'bil010010100000001; // -sqrt(2)/2 W_iv13_Imag &lt;= 18'bil010010100000000; // -jsqrt(2)/2 end 4'b001: begin W_iv13_Imag &lt;= 18'b0100000000000000; // 1 W_iv13_Real &lt;= 18'b010000000000000; // 1 4'b0010; begin W_iv13_Real &lt;= 18'b010000000000000; // 1 W_iv13_Real &lt;= 18'b010000000000000; // 1 W_iv13_Real &lt;= 18'b0100000000000000; // 1 W_iv13_Real &lt;= 18'b010000000000000000; // 1 W_iv13_Real &lt;= 18'b01000000000000000; // 1 W_iv13_Real &lt;= 18'b0100000000000000; // 1 W_iv13_Real &lt;= 18'b01000000000000000; // 1 W_iv13_Real &lt;= 18'b010000000000000000000; // 1 W_iv13_Real &lt;= 18'b0100000000000000000; // 1 W_iv13_Real &lt;= 18'b01000000000000000000000; // 10 W_iv13_Real &lt;= 18'b0100000000000000000000; // 10 W_iv13_Real &lt;= 18'b010000000000000000000000000000000000</pre>
$\begin{array}{c} 412 \\ + 12 \\ + 13 \\ + 14 \\ + 14 \\ + 15 \\ + 14 \\ + 15 \\ + 415 \\ + 415 \\ + 415 \\ + 415 \\ + 416 \\ + 417 \\ + 419 \\ + 422 \\ + 422 \\ + 422 \\ + 422 \\ + 422 \\ + 422 \\ + 425 \\ + 426 \\ + 428 \\ + 99 \\ +$	<pre>case (counter3[3:0]) 4'bomot: begin W_iv13_Real &lt;= 18'bil010010100000001; // -sqrt(2)/2 W_iv13_Imag &lt;= 18'bil010010100000000; // -jsqrt(2)/2 end 4'b001: begin W_iv13_Imag &lt;= 18'b0100000000000000; // 1 W_iv13_Real &lt;= 18'b010000000000000; // 1 4'b0010; begin W_iv13_Real &lt;= 18'b010000000000000; // 1 W_iv13_Real &lt;= 18'b010000000000000; // 1 W_iv13_Real &lt;= 18'b0100000000000000; // 1 W_iv13_Real &lt;= 18'b010000000000000000; // 1 W_iv13_Real &lt;= 18'b01000000000000000; // 1 W_iv13_Real &lt;= 18'b0100000000000000; // 1 W_iv13_Real &lt;= 18'b01000000000000000; // 1 W_iv13_Real &lt;= 18'b010000000000000000000; // 1 W_iv13_Real &lt;= 18'b0100000000000000000; // 1 W_iv13_Real &lt;= 18'b01000000000000000000000; // 10 W_iv13_Real &lt;= 18'b0100000000000000000000; // 10 W_iv13_Real &lt;= 18'b010000000000000000000000000000000000</pre>
412 0 413 0 414 1 415 4 415 4 415 4 417 0 418 0 422 0 422 0 422 0 422 0 422 0 422 0 422 0 422 0 425 4 425 4 426 0 427 0 428 0 428 0 428 0 431 4 431 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4	<pre>case (counter3[3:0]) 4'bomot: begin W_iv13_Real &lt;= 18'bil010010100000001; // -sqrt(2)/2 W_iv13_Imag &lt;= 18'bil010010100000000; // -jsqrt(2)/2 end 4'b001: begin W_iv13_Imag &lt;= 18'b0100000000000000; // 1 W_iv13_Real &lt;= 18'b010000000000000; // 1 4'b0010; begin W_iv13_Real &lt;= 18'b010000000000000; // 1 W_iv13_Real &lt;= 18'b010000000000000; // 1 W_iv13_Real &lt;= 18'b0100000000000000; // 1 W_iv13_Real &lt;= 18'b010000000000000000; // 1 W_iv13_Real &lt;= 18'b01000000000000000; // 1 W_iv13_Real &lt;= 18'b0100000000000000; // 1 W_iv13_Real &lt;= 18'b01000000000000000; // 1 W_iv13_Real &lt;= 18'b010000000000000000000; // 1 W_iv13_Real &lt;= 18'b0100000000000000000; // 1 W_iv13_Real &lt;= 18'b01000000000000000000000; // 10 W_iv13_Real &lt;= 18'b0100000000000000000000; // 10 W_iv13_Real &lt;= 18'b010000000000000000000000000000000000</pre>
412 0 413 0 414 1 415 4 415 4 415 4 417 0 418 0 422 0 422 0 422 0 422 0 422 0 422 0 422 0 422 0 425 4 425 4 426 0 427 0 428 0 428 0 428 0 431 4 431 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4	<pre>case (counter3[3:0]) 4'b0000; begin W_lvl3_Real &lt;= 18'b1010010100000001; // -sgrt(2)/2 w_lvl3_Imag &lt;= 18'b1010010100000000; // -jsgrt(2)/2 end 4'b0001; begin W_lvl3_Imag &lt;= 18'b0100000000000000; // 1 W_lvl3_Imag &lt;= 18'b0100000000000000; // -j0 end 4'b001; begin W_lvl3_Imag &lt;= 18'b010000000000000; // -j0 end 4'b001; begin W_lvl3_Imag &lt;= 18'b01011010111111; // sgrt(2)/2 W_lvl3_Imag &lt;= 18'b01010100000000; // -jsgrt(2)/2 w_lvl3_Imag &lt;= 18'b01010100000000; // -jsgrt(2)/2 w_lvl3_Imag &lt;= 18'b010101010000000; // -jsgrt(2)/2 w_lvl3_Imag &lt;= 18'b010101010000000; // -jsgrt(2)/2 w_lvl3_Imag &lt;= 18'b010101010000000; // -jsgrt(2)/2 w_lvl3_Imag &lt;= 18'b01010101010000000; // -jsgrt(2)/2 w_lvl3_Imag &lt;= 18'b0101010100000000; // -jsgrt(2)/2 w_lvl3_Imag &lt;= 18'b010101010000000; // -jsgrt(2)/2 w_lvl3_Imag &lt;= 18'b0101010100000000; // -jsgrt(2)/2 w_lvl3_Imag &lt;= 18'b010101010000000; // -jsgrt(2)/2 w_lvl3_Imag &lt;= 18'b0101010100000000; // -jsgrt(2)/2 w_lvl3_Imag &lt;= 18'b0101010100000000; // -jsgrt(2)/2 w_lvl3_Imag &lt;= 18'b01010101000000000; // -jsgrt(2)/2 w_lvl3_Imag &lt;= 18'b0101010100000000; // -jsgrt(2)/2 w_lvl3_Imag &lt;= 18'b010000000000000000; // -jsgrt(2)/2 w_lvl3_Imag &lt;= 18'b010000000000000000000000000000000000</pre>
412 0 413 0 413 0 415 4 415 4 415 4 416 4 419 4 419 4 419 4 422 0 423 0 423 0 423 0 425 4 425 4 426 0 427 0 429 0 431 4 429 0 439 0 439 0	<pre>case (counter3[3:0]) 4'bomot; begin W_lv13_Beal &lt;= 18'bil010010100000001; // -sqrt(2)/2 end 4'b0001; begin W_lv13_Beal &lt;= 18'b010000000000000; // 1 W_lv13_Beal &lt;= 18'b0100000000000000; // 1 W_lv13_Beal &lt;= 18'b010000000000000; // 2 end 4'b0101; begin W_lv13_Beal &lt;= 18'b01010101010111111; // sqrt(2)/2 end 4'b0100; </pre>
412 0 413 0 413 0 414 1 415 4 416 4 419 0 421 4 422 0 422 0 422 0 422 0 422 0 423 0 425 4 425 4 425 4 426 4 429 0 429 0 420 0 400 00	<pre>case (counter3[3:0]) 4'bomot; begin W_lv13_Real &lt;= 18'b11010010100000001; // -sqrt(2)/2 W_lv13_Imag &lt;= 18'b1010010100000000; // -jsqrt(2)/2 end 4'b0010; begin W_lv13_Real &lt;= 18'b0100000000000000; // 1 W_lv13_Real &lt;= 18'b010000000000000; // 1 W_lv13_Imag &lt;= 18'b0100000000000000; // 1 W_lv13_Real &lt;= 18'b0100000000000000; // 1 W_lv13_Real &lt;= 18'b010010110001111111; // sqrt(2)/2 end 4'b0010; begin W_lv13_Real &lt;= 18'b010101010000000; // -jsqrt(2)/2 end 4'b0010; begin </pre>
412 0 413 0 413 0 413 0 415 1 416 1 416 0 417 0 410 1 421 0 422 0 423 0 425 1 425 0 425 0	<pre>case (counter3[3:0]) 4'bomo: begin W_lv13_Beal &lt;= 18'bil010010100000001; // -sqrt(2)/2 end 4'b0001 begin W_lv13_Beal &lt;= 18'b0100000000000000; // 1 W_lv13_Beal &lt;= 18'b010000000000000; // 2 end 4'b0101; begin W_lv13_Beal &lt;= 18'b001011010111111; // sqrt(2)/2 end 4'b0100; begin W_lv13_Beal &lt;= 18'b010101010000000; // -jsqrt(2)/2 end 4'b0100; begin W_lv13_Beal &lt;= 18'b0010110100111111; // sqrt(2)/2 end 4'b0100; begin W_lv13_Beal &lt;= 18'b0010110100111111; // sqrt(2)/2 end 4'b0100; begin W_lv13_Beal &lt;= 18'b010110100111111; // sqrt(2)/2</pre>
$\begin{array}{c} 412 \\ + 1$	<pre>case (counter3[3:0]) 4'booo: begin W_lvl3_Real &lt;= 18'bil010010100000001; // -sqrt(2)/2 W_lvl3_Imag &lt;= 18'bil010010100000000; // -jaqrt(2)/2 end 4'b0001: begin W_lvl3_Imag &lt;= 18'b0100000000000000; // 1 W_lvl3_Real &lt;= 18'b0100000000000000; // 1 W_lvl3_Real &lt;= 18'b01000000000000000; // 1 W_lvl3_Real &lt;= 18'b0100000000000000; // 20 end 4'b0011: begin W_lvl3_Real &lt;= 18'b01010100000000; // -j0 end 4'b0010: begin W_lvl3_Real &lt;= 18'b010101010000000; // -j0 end 4'b0010: begin W_lvl3_Real &lt;= 18'b010101010000000; // -j0 end 4'b0010: begin W_lvl3_Real &lt;= 18'b010101010000000; // -j0qrt(2)/2 w_lvl3_Imag &lt;= 18'b010110100111111; // sqrt(2)/2 w_lvl3_Real &lt;= 18'b010101010000000; // -jaqrt(2)/2 end 4'b010: begin W_lvl3_Real &lt;= 18'b01001010000000; // -jaqrt(2)/2 end 4'b010: begin W_lvl3_Real &lt;= 18'b01001010000000; // -jaqrt(2)/2 end 4'b010: begin W_lvl3_Real &lt;= 18'b01001010000000; // -jaqrt(2)/2 end 4'b010: begin W_lvl3_Real &lt;= 18'b0100000000; // -jaqrt(2)/2 end 4'b0100: begin W_lvl3_Real &lt;= 18'b0100000000; // -jaqrt(2)/2 end 4'b01000000000; // -jaqrt(2)/2 W_lvl3_Real &lt;= 18'b010000000000000000000000000000000000</pre>
$\begin{array}{c} 412 \\ + 1$	<pre>case (counter3[3:0]) 4'b0000; begin W_lvl3_Real &lt;= 18'b1010010100000001; // -sgrt(2)/2 w_lvl3_Imag &lt;= 18'b1010010100000000; // -jsgrt(2)/2 end 4'b0001; begin W_lvl3_Imag &lt;= 18'b000000000000000; // 1 W_lvl3_Real &lt;= 18'b00000000000000; // -j0 end 4'b0010; begin W_lvl3_Imag &lt;= 18'b00000000000000; // -j0 end 4'b0011; bggin W_lvl3_Imag &lt;= 18'b010101010111111; // sgrt(2)/2 w_lvl3_Imag &lt;= 18'b010101010000000; // -jsgrt(2)/2 end 4'b010; begin W_lvl3_Imag &lt;= 18'b010101010000000; // -jsgrt(2)/2 end 4'b010; begin W_lvl3_Imag &lt;= 18'b010101010111111; // sgrt(2)/2 w_lvl3_Imag &lt;= 18'b010101010000000; // -jsgrt(2)/2 end 4'b010; begin W_lvl3_Imag &lt;= 18'b010101010000000; // -jsgrt(2)/2 end 4'b010; begin W_lvl3_Imag &lt;= 18'b010101010000000; // -jsgrt(2)/2 w_lvl3_Imag &lt;= 18'b010101010000000; // -jsgrt(2)/2 w_lvl3_Imag &lt;= 18'b0101010000000; // -jsgrt(2)/2 w_lvl3_Imag &lt;= 18'b010101010000000; // -jsgrt(2)/2 w_lvl3_Imag &lt;= 18'b0101010000000; // -jsgrt(2)/2 w_lvl3_Imag &lt;= 18'b0101010000000; // -jsgrt(2)/2 w_lvl3_Imag &lt;= 18'b01001010000000; // -jsgrt(2)/2 w_lvl3_Imag &lt;= 18'b01000000000; // -jsgrt(2)/2 w_lvl3_Imag &lt;= 18'b01000000000; // -jsgrt(2)/2 w_lvl3_Imag &lt;= 18'b0100000000; // -jsgrt(2)/2 w_lvl3_Imag &lt;= 18'b010000000000; // -jsgrt(2)/2 w_lvl3_Imag &lt;= 18'b01000000000; // -jsgrt(2)/2 w_lvl3_Imag &lt;= 18'b00000000000; // -jsgrt(2)/2 w_lvl3_Imag &lt;= 18'b0100000000; /</pre>
$\begin{array}{c} 412 \\ + 1$	<pre>case (counter3[3:0]) 4'blood; begin W_Jv13_Real &lt;= 18'bl1010010100000001; // -sqrt(2)/2 w_lv13_Real &lt;= 18'bl101001010000000; // 1 W_lv13_Real &lt;= 18'b0100000000000000; // 2) end 4'b0011; begin W_lv13_Real &lt;= 18'b010101010111111; // sqrt(2)/2 w_lv13_Real &lt;= 18'b010101010000001; // -sqrt(2)/2 end 4'b0100; begin W_lv13_Real &lt;= 18'b010101010000001; // -sqrt(2)/2 end 4'b010; begin W_lv13_Real &lt;= 18'b01011010111111; // sqrt(2)/2 end 4'b010; begin W_lv13_Real &lt;= 18'b01011010111111; // sqrt(2)/2 end 4'b010;</pre>
412 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	<pre>case (counter3[3:0]) 4'b0000; begin W_lvl3_Real &lt;= 18'b1010010100000001; // -sgrt(2)/2 w_lvl3_Imag &lt;= 18'b1010010100000000; // -jsgrt(2)/2 end 4'b0001; begin W_lvl3_Imag &lt;= 18'b0100000000000000; // 1 W_lvl3_Imag &lt;= 18'b010000000000000; // -j0 end 4'b001; begin W_lvl3_Imag &lt;= 18'b010010100000000; // -j0 end 4'b001; begin W_lvl3_Real &lt;= 18'b010101010111111; // sgrt(2)/2 w_lvl3_Real &lt;= 18'b0101010000000; // -jsgrt(2)/2 end 4'b010; begin W_lvl3_Real &lt;= 18'b0101010000000; // -jsgrt(2)/2 end 4'b010; begin W_lvl3_Real &lt;= 18'b010110000000; // -jsgrt(2)/2 end 4'b010; begin W_lvl3_Real &lt;= 18'b01010101000000; // -jsgrt(2)/2 end 4'b010; begin W_lvl3_Real &lt;= 18'b0101010000000; // -jsgrt(2)/2 end 4'b010; begin </pre>
412 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	<pre>case (counter3[3:0]) 4'b0000; begin W_Jv13_Real &lt;= 18'b1101001010000001; // -sqrt(2)/2 w_Jv13_Real &lt;= 18'b1010010100000000; // -jaqrt(2)/2 end 4'b0001; begin W_Jv13_Real &lt;= 18'b0100000000000000; // 1 W_Jv13_Real &lt;= 18'b0100000000000000; // 1 W_Jv13_Real &lt;= 18'b0100000000000000; // -j0 end 4'b0011; begin W_Jv13_Real &lt;= 18'b010101010111111; // sqrt(2)/2 end 4'b0100; begin W_Jv13_Real &lt;= 18'b010101010000000; // -j0qrt(2)/2 end 4'b010; begin W_Jv13_Real &lt;= 18'b0010110000000; // -j0qrt(2)/2 end 4'b010; begin W_Jv13_Real &lt;= 18'b001001010000000; // -j0qrt(2)/2 end 4'b010; begin W_Jv13_Real &lt;= 18'b001001010000000; // -j0qrt(2)/2 end 4'b010; begin W_Jv13_Real &lt;= 18'b000000000000000000000000; // 0</pre>
412 中 413 中 413 中 413 中 414 415	<pre>case (counter3[3:0]) 4'bomot; begin W_lvl3_Real &lt;= 18'bl1010010100000001; // -sgrt(2)/2 W_lvl3_Imag &lt;= 18'bl1010010100000000; // -jsgrt(2)/2 end 4'bomot; begin W_lvl3_Real &lt;= 18'b0100000000000000; // 1 W_lvl3_Real &lt;= 18'b0100000000000000; // -j0 end 4'b0010; begin W_lvl3_Real &lt;= 18'b0100101000000000000; // -j0 end 4'b0011; begin W_lvl3_Real &lt;= 18'b0101010101111111; // sgrt(2)/2 W_lvl3_Imag &lt;= 18'b01010100000000; // -jagrt(2)/2 end 4'b0101: begin W_lvl3_Real &lt;= 18'b010101010000000; // -jagrt(2)/2 end 4'b0101; begin W_lvl3_Real &lt;= 18'b01010101000000; // -jagrt(2)/2 end 4'b0101; begin W_lvl3_Real &lt;= 18'b0101101001111111; // sgrt(2)/2 end 4'b010; begin W_lvl3_Real &lt;= 18'b01011010000000; // -jagrt(2)/2 end 4'b010; begin W_lvl3_Real &lt;= 18'b01011010000000; // -jagrt(2)/2 end 4'b010; begin W_lvl3_Real &lt;= 18'b0100110000000; // -jagrt(2)/2 end 4'b010; begin W_lvl3_Real &lt;= 18'b01001010000000; // -jagrt(2)/2 end 4'b010; begin W_lvl3_Real &lt;= 18'b0100101000000; // -jagrt(2)/2 end 4'b010; begin W_lvl3_Real &lt;= 18'b01000000000000; // -jagrt(2)/2 end 4'b010; begin W_lvl3_Real &lt;= 18'b01000000000000; // -jagrt(2)/2 end 4'b010; begin W_lvl3_Real &lt;= 18'b010000000000000; // -jagrt(2)/2 end 4'b010; begin W_lvl3_Real &lt;= 18'b010000000000000; // -jagrt(2)/2 end 4'b010; begin W_lvl3_Real &lt;= 18'b01000000000000; // -jagrt(2)/2 end 4'b010; begin W_lvl3_Real &lt;= 18'b010000000000000; // -jagrt(2)/2 end 4'b010; begin W_lvl3_Real &lt;= 18'b01000000000000; // -jagrt(2)/2 end 4'b000; begin W_lvl3_Real &lt;= 18'b010000000000000; // -jagrt(2)/2 end 4'b000; begin W_lvl3_Real &lt;= 18'b00000000000000; // -jagrt(2)/2 end 4'b000; begin W_lvl3_Real &lt;= 18'b00000000000000; // -jagrt(2)/2 end 4'b000; begin W_lvl3_Real &lt;= 18'b000000000000000; // -jagrt(2)/2 end 4'b000; b0000; b0000000000000; // -j1 b0000; b0000000000000000; // -j1</pre>
412 中 413 中 413 中 413 中 414 415	<pre>case (counter3[3:0]) {*booo:     wein     wijv13.Real &lt;= 18*bil010010100000001; // -sqrt(2)/2     end     wijv13.Real &lt;= 18*bil01001100000000; // 1     wijv13.Real &lt;= 18*b00000000000000; // 1     wijv13.Real &lt;= 18*b00000000000000; // 1     wijv13.Real &lt;= 18*b00000000000000; // 1     wijv13.Real &lt;= 18*b0000000000000; // 1     wijv13.Real &lt;= 18*b0000000000000; // 1     wijv13.Real &lt;= 18*b0000000000000; // 2     end     4*b0001;     begin     wijv13.Real &lt;= 18*b001011010111111; // sqrt(2)/2     end     4*b010;     begin     wijv13.Real &lt;= 18*b0010110100111111; // sqrt(2)/2     end     4*b010;     begin     wijv13.Real &lt;= 18*b0010110100111111; // sqrt(2)/2     end     4*b010;     begin     wijv13.Real &lt;= 18*b0010110000000; // -jsqrt(2)/2     end     4*b010;     begin     wijv13.Real &lt;= 18*b001011010000000; // -jsqrt(2)/2     end     wijv13.Real &lt;= 18*b00000000000000; // -jsqrt(2)/2     end     wijv13.Real &lt;= 18*b0000000000000000; // -jsqrt(2)/2     end     wijv13.Real &lt;= 18*b00000000000000000; // -jsqrt(2)/2     end     wijv13.Real &lt;= 18*b000000000000000000000000000000000000</pre>
412 0 0 413 0 0 413 0 0 414 1 415 0 414 1 415 0 414 1 417 0 421 0 422 0 423 0 420 0 420 420 0 420 0 400 0 40	<pre>case (counter3[3:0]) 4'bomo: begin W_lvl3_Real &lt;= 18'bil010010100000001; // -sqrt(2)/2 W_lvl3_Imag &lt;= 18'bil010010100000000; // -jsqrt(2)/2 end 4'bomo: begin W_lvl3_Imag &lt;= 18'b0100000000000000; // 1 W_lvl3_Real &lt;= 18'b0100000000000000; // 1 W_lvl3_Real &lt;= 18'b0100000000000000; // 1 W_lvl3_Real &lt;= 18'b010010100000000; // 1 W_lvl3_Real &lt;= 18'b010101010111111; // sqrt(2)/2 W_lvl3_Real &lt;= 18'b010101010000001; // -jsqrt(2)/2 end 4'b000: begin W_lvl3_Real &lt;= 18'b010101010000001; // -jsqrt(2)/2 end 4'b010: begin W_lvl3_Real &lt;= 18'b010101010000001; // -jsqrt(2)/2 end 4'b010: begin W_lvl3_Real &lt;= 18'b010101010000001; // -jsqrt(2)/2 end 4'b010: begin W_lvl3_Real &lt;= 18'b010101010000001; // -jsqrt(2)/2 end 4'b0101: begin W_lvl3_Real &lt;= 18'b01001010000000; // 0 W_lvl3_Real &lt;= 18'b0100000000000000; // 0 W_lvl3_Real &lt;= 18'b010000000000000000; // 0 W_lvl3_Real &lt;= 18'b01000000000000000; // 0 W_lvl3_Real &lt;= 18'b01000000000000000; // 0 W_lvl3_Real &lt;= 18'b010000000000000000; // 0 W_lvl3_Real &lt;= 18'b01000000000000000000; // 0 W_lvl3_Real &lt;= 18'b010000000000000000; // 0 W_lvl3_Real &lt;= 18'b010000000000000000; // 0 W_lvl3_Real &lt;= 18'b010000000000000000; // 0 W_lvl3_Real &lt;= 18'b0100000000000000000; // 0 W_lvl3_Real &lt;= 18'b010000000000000000000; // 0 W_lvl3_Real &lt;= 18'b010000000000000000; // 0 W_lvl3_Real &lt;= 18'b0100000000000000000; // 0 W_lvl3_Real &lt;= 18'b0100000000000000000; // 0 W_lvl3_Real &lt;= 18'b0100000000000000000; // 0 W_lvl3_Real &lt;= 18'b01000000000000000000; // 0 W_lvl3_Real &lt;= 18'b01000000000000000000; // 0 W_lvl3_Real &lt;= 18'b0100000000000000000000; // 0 W_lvl3_Real &lt;= 18'b010000000000000000000000000000000000</pre>
412 0 0 413 0 414 0 415 0 414 0 415 0 416 0 416 0 417 0 416 0 410	<pre>case (counter3[3:0]) {*booos: begin</pre>
412 0 0 413 0 0 413 0 0 414 1 415 4 415 4 417 0 0 422 0 0 422 0 0 422 0 0 423 0 0 423 0 0 423 0 0 423 0 0 423 0 0 433 0 0 433 0 0 433 0 0 433 0 0 434 422 0 0 434 423 0 435 4 436 0 444 0	<pre>case (counter3[3:0]) 4'bomo: begin W_lvl3_Real &lt;= 18'bil010010100000001; // -sqrt(2)/2 W_lvl3_Imag &lt;= 18'bil010010100000000; // -jsqrt(2)/2 end 4'b001: begin W_lvl3_Imag &lt;= 18'b0100000000000000; // 1 W_lvl3_Real &lt;= 18'b0100000000000000; // 1 W_lvl3_Real &lt;= 18'b0100000000000000; // 1 W_lvl3_Real &lt;= 18'b010010100000000; // 20 end 4'b0010: begin W_lvl3_Real &lt;= 18'b010101010000000; // -j0 end 4'b0100: begin W_lvl3_Real &lt;= 18'b01010101000000; // -j0qrt(2)/2 end 4'b0100: begin W_lvl3_Real &lt;= 18'b01010101000000; // -jqrt(2)/2 end 4'b0100: begin W_lvl3_Real &lt;= 18'b01010101000000; // -jqrt(2)/2 end 4'b0100: begin W_lvl3_Real &lt;= 18'b01001010000000; // -jqrt(2)/2 end 4'b0100: begin W_lvl3_Real &lt;= 18'b0100000000000000; // -j1 end 4'b0100: begin W_lvl3_Real &lt;= 18'b0100000000000000; // -j1 end 4'b0100: begin W_lvl3_Real &lt;= 18'b0100000000000000; // 0 W_lvl3_Real &lt;= 18'b010000000000000; // 0 W_lvl3_Real &lt;= 18'b0100000000000000; // 0 W_lvl3_Real &lt;= 18'b000000000000000; // 0 W_lvl3_Real &lt;= 18'b0000000000000000; // 0 W_lvl3_Real &lt;= 18'b0000000000000000; // 0 W_lvl3_Real &lt;= 18'b0000000000000000; // 0 W_lvl3_Real &lt;= 18'b00000000000000000; // 0 W_lvl3_Real &lt;= 18'b000000000000000000; // 0 W_lvl3_Real &lt;= 18'b000000000000000000; // 0 W_lvl3_Real &lt;= 18'b00000000000000000; // 0 W_lvl3_Real &lt;= 18'b00000000000000000; // 0 W_lvl3_Real &lt;= 18'b00000000000000000; // 0 W_lvl3_Real &lt;= 18'b0000000000000000; // 0 W_lvl3_Real &lt;= 18'b00000000000000000000; // 0 W_lvl3_Real &lt;= 18'b0000000000000000000; // 0 W_lvl3_Real &lt;= 18'b00000000000000000; // 0 W_lvl3_Real &lt;= 18'b000000000000000000; // 0 W_lvl3_Real &lt;= 18'b00000000000000000000; // 0 W_lvl3_Real &lt;= 18'b0000000000000000000000; // 0 W_lvl3_Real &lt;= 18'b000000</pre>
412	<pre>case (counter3[3:0]) 4'bomo: begin # [v13]Real &lt;= 18'bil010010100000001; // -sqrt(2)/2 end 4'b000: begin # [v13]Real &lt;= 18'b0100000000000000; // 1 # [v13]Real &lt;= 18'b01001010111111; // sqrt(2)/2 end 4'b0101; begin # [v13]Real &lt;= 18'b01010101010111111; // sqrt(2)/2 end 4'b0100; begin # [v13]Real &lt;= 18'b010101010000000; // -jaqrt(2)/2 end 4'b0100; begin # [v13]Real &lt;= 18'b0100101000000; // -jaqrt(2)/2 end 4'b0100; begin # [v13]Real &lt;= 18'b0100101000000; // -jaqrt(2)/2 end 4'b0100; begin # [v13]Real &lt;= 18'b01001010000000; // 0 # [v13]Real &lt;= 18'b00000000000000; // -j1 end 4'b010; begin # [v13]Real &lt;= 18'b00000000000000; // -j1 end 4'b010; begin # [v13]Real &lt;= 18'b00000000000000; // -j1 end 4'b010; begin # [v13]Real &lt;= 18'b00000000000000; // 0 # [v13]Real &lt;= 18'b00000000000000; // -j1 end 4'b010; begin # [v13]Real &lt;= 18'b00000000000000; // 0 # [v13]Real &lt;= 18'b00000000000000; // 0 # [v13]Real &lt;= 18'b000000000000000; // -j1 end 4'b010; begin # [v13]Real &lt;= 18'b000000000000000; // 0 # [v13]Real &lt;= 18'b00000000000000; // 0 # [v13]Real &lt;= 18'b000000000000000; // -j1 end 4'b010; begin # [v13]Real &lt;= 18'b0000000000000000; // 0 # [v13]Real &lt;= 18'b000000000000000; // 0 # [v13]Real &lt;= 18'b0000000000000000; // -j1 end 4'b000; begin # [v13]Real &lt;= 18'b0000000000000000; // 0 # [v13]Real &lt;= 18'b000000000000000; // 0 # [v13]Real &lt;= 18'b000000000000000; // -j1 end # b00000000000000000; // -j1</pre>
412	<pre>case (counter3[3:0]) 4'blood: begin W_lv13_Real &lt;= 18'bl1010010100000001; // -agrt(2)/2 w_lv13_Imag &lt;= 18'bl101001100000000; // 1 W_lv13_Imag &lt;= 18'b0100000000000000; // 1 W_lv13_Real &lt;= 18'b0100000000000000; // 1 W_lv13_Real &lt;= 18'b0100000000000000; // 1 W_lv13_Real &lt;= 18'b010010100000000; // 1 W_lv13_Real &lt;= 18'b01001010000000; // 20 end 4'b0101; begin W_lv13_Real &lt;= 18'b0101110001111111; // agrt(2)/2 w_lv13_Real &lt;= 18'b01010101000000; // agrt(2)/2 end 4'b010; begin W_lv13_Real &lt;= 18'b01010101000000; // agrt(2)/2 end 4'b010; begin W_lv13_Real &lt;= 18'b0100101000000; // agrt(2)/2 end 4'b010; begin W_lv13_Real &lt;= 18'b0100101000000; // agrt(2)/2 end 4'b010; begin W_lv13_Real &lt;= 18'b0100101000000; // agrt(2)/2 end 4'b010; begin W_lv13_Real &lt;= 18'b00000000000000; // agrt(2)/2 end 4'b010; begin W_lv13_Real &lt;= 18'b000000000000000; // agrt(2)/2 end 4'b010; begin W_lv13_Real &lt;= 18'b000000000000000; // agrt(2)/2 end 4'b010; begin W_lv13_Real &lt;= 18'b000000000000000; // agrt(2)/2 end 4'b010; begin W_lv13_Real &lt;= 18'b0000000000000000; // agrt(2)/2 end 4'b010; begin W_lv13_Real &lt;= 18'b0000000000000000; // agrt(2)/2 end 4'b010; begin W_lv13_Real &lt;= 18'b000000000000000; // agrt(2)/2 end 4'b010; begin W_lv13_Real &lt;= 18'b000000000000000; // agrt(2)/2 end 4'b010; begin W_lv13_Real &lt;= 18'b0000000000000000; // agrt(2)/2 end 4'b010; begin W_lv13_Real &lt;= 18'b000000000000000; // agrt(2)/2 end 4'b010; begin W_lv13_Real &lt;= 18'b000000000000000; // agrt(2)/2 end 4'b010; begin W_lv13_Real &lt;= 18'b0000000000000000; // agrt(2)/2 end 4'b010; begin W_lv13_Real &lt;= 18'b000000000000000; // agrt(2)/2 end 4'b010; begin W_lv13_Real &lt;= 18'b0000000000000000; // agrt(2)/2 end 4'b010; begin W_lv13_Real &lt;= 18'b00000000000000000; // agrt(2)/2 end 4'b010; begin W_lv13_Real &lt;= 18'b0000000000000</pre>
412 0 0 413 0 414 0 415 0 414 0 415 0 414 0 419 0 410	<pre>case (counter3[3:0]) 4'bomot; begin W_lvl3_Real &lt;= 18'b1010010100000001; // -sgrt(2)/2 W_lvl3_Imag &lt;= 18'b1010010100000000; // -jsgrt(2)/2 end 4'b0010; begin W_lvl3_Real &lt;= 18'b0100000000000000; // 1 W_lvl3_Real &lt;= 18'b010000000000000; // 1 W_lvl3_Real &lt;= 18'b0100101000000000000; // -j0 end 4'b0010; begin W_lvl3_Real &lt;= 18'b01010101010111111; // sgrt(2)/2 W_lvl3_Real &lt;= 18'b010101010000000; // -jsgrt(2)/2 end 4'b0100; begin W_lvl3_Real &lt;= 18'b01010101000000; // -jsgrt(2)/2 end 4'b0100; begin W_lvl3_Real &lt;= 18'b01010101000000; // -jsgrt(2)/2 end 4'b0100; begin W_lvl3_Real &lt;= 18'b010101010000000; // -jsgrt(2)/2 end 4'b0100; begin W_lvl3_Real &lt;= 18'b01001010000000; // -jsgrt(2)/2 end 4'b0100; begin W_lvl3_Real &lt;= 18'b00000000000000; // 0 W_lvl3_Real &lt;= 18'b00000000000000; // 0 W_lvl3_Real &lt;= 18'b000000000000000; // 0 W_lvl3_Real &lt;= 18'b000000000000000; // 0 W_lvl3_Real &lt;= 18'b00000000000000; // 0 W_lvl3_Real &lt;= 18'b000000000000000; // 0 W_lvl3_Real &lt;= 18'b0000000000000000; // 0 W_lvl3_Real &lt;= 18'b0000000000000000; // 0 W_lvl3_Real &lt;= 18'b0000000000000000; // 0 W_lvl3_Real &lt;= 18'b00000000000000000; // 0 W_lvl3_Real &lt;= 18'b0000000000000000; // 0 W_lvl3_Real &lt;= 18'b00000000000000000; // 0 W_lvl3_Real &lt;= 18'b0000000000000000; // 0 W_lvl3_Real &lt;= 18'b00000000000000000; // 0 W_lvl3_Real &lt;= 18'b000000000000000000; // 0 W_lvl3_Real &lt;= 18'b000000000000000000; // 0 W_lvl3_Real &lt;= 18'b000000000000000000; // 0 W_lvl3_Real &lt;= 18'b000000000000000000000; // 0 W_lvl3_Real &lt;= 18'b000000000000000000; // 0 W_lvl3_Real &lt;= 18'b000000000000000000000; // 0 W_lvl3_Real &lt;= 18'b00000</pre>
412 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	<pre>case (counter3[3:0]) {*booo: begin W_Jv13_Real &lt;= 18*b1101001010000001; // -agrt(2)/2 w_Jv13_Real &lt;= 18*b1101001100000000; // -jngrt(2)/2 end 4*b0001: begin W_Jv13_Real &lt;= 18*b0100000000000000; // 1 W_Jv13_Real &lt;= 18*b010000000000000; // 1 W_Jv13_Real &lt;= 18*b010000000000000; // -j0 end 4*b0001: begin W_Jv13_Real &lt;= 18*b0101010000000; // -j0 end 4*b0101: begin W_Jv13_Real &lt;= 18*b0101010000000; // -j0 end 4*b0100: begin W_Jv13_Real &lt;= 18*b0101010000000; // -j0; end 4*b0101: begin W_Jv13_Real &lt;= 18*b0100101000000; // -j0; end 4*b0101: begin W_Jv13_Real &lt;= 18*b0101010000000; // -j0; end 4*b0101: begin W_Jv13_Real &lt;= 18*b01001010000000; // -j1; end 4*b0101: begin W_Jv13_Real &lt;= 18*b010000000000000; // -j1; end 4*b0101: begin W_Jv13_Real &lt;= 18*b010000000000000; // -j1; end 4*b0101: begin W_Jv13_Real &lt;= 18*b01000000000000; // 0 W_Jv13_Real &lt;= 18*b0100000000000; // -j1; end 4*b0101: begin W_Jv13_Real &lt;= 18*b010000000000000; // -j1; end 4*b0101: begin W_Jv13_Real &lt;= 18*b0100000000000000; // -j1; end 4*b0101: begin W_Jv13_Real &lt;= 18*b010000000000000; // -j1; end 4*b0101: begin W_Jv13_Real &lt;= 18*b01000000000000000; // -j1; end 4*b0101: begin W_Jv13_Real &lt;= 18*b0100000000000000; // -j1; end 4*b0101: begin W_Jv13_Real &lt;= 18*b01000000000000000; // 0 W_Jv13_Real &lt;= 18*b0100000000000000; // -j1; end 4*b0101: begin W_Jv13_Real &lt;= 18*b010000000000000000000; // -j1; end 4*b0101: begin W_Jv13_Real &lt;= 18*b0100000000000000000; // 0 W_Jv13_Real &lt;= 18*b01000000000000000; // 0 W_Jv13_Real &lt;= 18*b010000000000000000; // 0 W_Jv13_Real &lt;= 18*b0100000000000000000000; // 0 W_Jv13_Real &lt;= 18*b0100000000000000000000; // 0 W_Jv13_Real &lt;= 18*b01000000000000000000; // 0 W_Jv13_Real &lt;= 18*b010000000000000000000; // 0 W_Jv13_Real &lt;= 18*b010000000000000000000000; // 0 W_Jv13_Real &lt;= 18*b010000000000000000000000000000000000</pre>
412 ( - - - - - - - - - - - - -	<pre>case (counter3[3:0]) 4'bomot; begin W_lvl3_Real &lt;= 18'b1010010100000001; // -sgrt(2)/2 W_lvl3_Imag &lt;= 18'b1010010100000000; // -jsgrt(2)/2 end 4'b0010; begin W_lvl3_Real &lt;= 18'b0100000000000000; // 1 W_lvl3_Real &lt;= 18'b010000000000000; // 1 W_lvl3_Real &lt;= 18'b0100101000000000000; // -j0 end 4'b0010; begin W_lvl3_Real &lt;= 18'b01010101010111111; // sgrt(2)/2 W_lvl3_Real &lt;= 18'b010101010000000; // -jsgrt(2)/2 end 4'b0100; begin W_lvl3_Real &lt;= 18'b01010101000000; // -jsgrt(2)/2 end 4'b0100; begin W_lvl3_Real &lt;= 18'b01010101000000; // -jsgrt(2)/2 end 4'b0100; begin W_lvl3_Real &lt;= 18'b010101010000000; // -jsgrt(2)/2 end 4'b0100; begin W_lvl3_Real &lt;= 18'b01001010000000; // -jsgrt(2)/2 end 4'b0100; begin W_lvl3_Real &lt;= 18'b00000000000000; // 0 W_lvl3_Real &lt;= 18'b00000000000000; // 0 W_lvl3_Real &lt;= 18'b000000000000000; // 0 W_lvl3_Real &lt;= 18'b000000000000000; // 0 W_lvl3_Real &lt;= 18'b00000000000000; // 0 W_lvl3_Real &lt;= 18'b000000000000000; // 0 W_lvl3_Real &lt;= 18'b0000000000000000; // 0 W_lvl3_Real &lt;= 18'b0000000000000000; // 0 W_lvl3_Real &lt;= 18'b0000000000000000; // 0 W_lvl3_Real &lt;= 18'b00000000000000000; // 0 W_lvl3_Real &lt;= 18'b0000000000000000; // 0 W_lvl3_Real &lt;= 18'b00000000000000000; // 0 W_lvl3_Real &lt;= 18'b0000000000000000; // 0 W_lvl3_Real &lt;= 18'b00000000000000000; // 0 W_lvl3_Real &lt;= 18'b000000000000000000; // 0 W_lvl3_Real &lt;= 18'b000000000000000000; // 0 W_lvl3_Real &lt;= 18'b000000000000000000; // 0 W_lvl3_Real &lt;= 18'b000000000000000000000; // 0 W_lvl3_Real &lt;= 18'b000000000000000000; // 0 W_lvl3_Real &lt;= 18'b000000000000000000000; // 0 W_lvl3_Real &lt;= 18'b00000</pre>

451	W_lv13_Imag <= 18'b110100101100000001; // -sqrt(2)/2
452 🍐	end
453 🖯	4'b1000:
454 🖯	begin
455	W 1v13 Real <= 18'b110100101100000001; // -sqrt(2)/2
456	W 1v13 Imag <= 18'b110100101100000001; // -sgrt(2)/2
457	end
458 🗄	4'b1001:
459 🗄	begin
460	W lv13 Real <= 18'b01000000000000000; // 1
461	W lv13 Imag <= 18'b00000000000000000; // -j0
462	end
463 🗄	4'b1010:
464	begin
465	W 1v13 Real <= 18'b01000000000000000; // 1
466	W 1v13_Real <= 18 b0000000000000000; // -i0
467	end
468 🖯	4'b1011:
469 🖯	4 bioir: begin
470	
471	W_lv13_Real <= 18'b001011010011111111; // sqrt(2)/2 W lv13 Imag <= 18'b110100101100000001; // -jsqrt(2)/2
	w_ivi3_imag <= 18*bii0100101100000001; // -jsqrt(2)/2 end
472 A	ena 4'b1100:
473 🗟	
474 🖻	begin
475	W_lvl3_Real <= 18'b001011010011111111; // sqrt(2)/2
476 481	W_lvl3_Imag <= 18'b110100101100000001; // -isgrt(2)/2 W lvl3 Imag <= 18'b1100000000000000; // -j1
482	w_1013_1mag <= 18*b11000000000000000; // -j1 end
483 🖨	4'b1110:
484 🖯	4 billo: begin
485	
	W_lv13_Real <= 18'b000000000000000000; // 0
486	<pre>W_lv13_Imag &lt;= 18'b1100000000000000; // -j1 end</pre>
487 🔶	
488 🖻	4'b1111:
489 🖗	begin
490	W_1v13_Real <= 18'b110100101100000001; // -sqrt(2)/2
491	W_lvl3_Imag <= 18'b110100101100000001; // -jsqrt(2)/2
492 🍦	end
493 🍦	endcase
494	
495	counter <= counter + 1;
496	<pre>counterl &lt;= counterl + 1;</pre>
497	<pre>counter2 &lt;= counter2 + 1;</pre>
498	counter3 <= counter3 + 1;
499 🍙	end
500 ( em	imodule

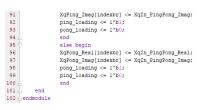
6 // This file contains confidential and proprietary information 7 // of Xilinx, Inc. and is protected under U.S. and 8 // international copyright and other intellectual property 9 // laws. 10 // 11 // DISCLAIMER 12 // This foleclaimer is not a license and does not grant any 13 // rights to the matherials distributed herwith. Except as 14 // otheriae provided in a valid license issued to you by 15 // Xilinx, and to the maximum extent permitted by applicable 16 // Vill L FAULTS MATERIALS ARE MADE AVAILABLE "AS IS" AND 17 // WITH ALL FAULTS, MAIL MAILTS MATERIALS ARE MADE AVAILABLE "AS IS" AND 17 // WITH ALL FAULTS. MAIL MAILTS MATERIALS LARE MADE AVAILABLE TAS IS" AND 17 // WITH ALL FAULTS. MAIL MAILTS MAIL AURTES	
4 // (c) Copyright 2008 - 2013 Xilinx, Inc. All rights reserved. 5 // This file contains confidential and proprietary information 7 // of Xilinx, Inc. and is probected under 0.5. and 8 // International copyright and other intellectual property 9 // lavs. 10 // Vist relational copyright and other intellectual property 11 // DISCLAIMER 12 // This file contains is not a license and does not grant any 13 // rights to the materials distributed herewith. Except as 14 // otherwise provided in a valid license issued to you by 15 // Kilm, and to the maximum extent permitted by applicable 16 // lavt. (1) THESE MATERIALS ARE MADE AVAILABLE "AS 15° AND 17 // WITH AL FADUTS, AND VILINX HERERY DISCLAIMER LL & & & AND 17 // WITH AL FADUTS, AND VILINX HERERY DISCLAIME ALL & & & & & & & & & & & & & & & & & &	
<pre>5 // // if file contains confidential and proprietary information 7 // of Xlinx, Inc. and is protected under 0.5. and 7 // of Xlinx, Inc. and is protected under 0.5. and 7 // of Xlinx, Inc. and is protected under 0.5. and 7 // of Xlinx, Inc. and is protected under 0.5. and 7 // of Xlinx, Inc. and is protected under 0.5. and 7 // of Xlinx and to the intellectual property 9 // lav: 1 // DiscLaIMER 12 // Trights to the materials distributed herewith. Except as 14 // otherwise provided in a valid license issued to you by 15 // Xlinx, and to the maximum extent permitted by applicable 16 // lav: (1) THESE MATERIALS ARE MADE AVAILABLE "AS IS" AND 17 // WITH ALL FADITS, MAYINA HEREPS USCLAIMES ALL #ARRANTES</pre>	
6 // This file contains confidential and proprietary information 7 // of Xilinx, Inc. and is protected under U.S. and 8 // international copyright and other intellectual property 9 // laws. 10 // Usoclatter 11 // Disclatter 12 // This fieloclainer is not a license and does not grant any 13 // Trig fieloclainer is not a license and does not grant any 14 // otherwise provided in a valid license issued to you by 15 // Xilinx, and to the maximum extent permitted by applicable 16 // law: (1) THESE MATERIALS ARE MADE AVAILABLE "AS 15" AND 17 // WITH L FAULTS, MAILTAN LEWERPUSCLAIMER LL & WARANTIES	
7 // of Xilinx, Enc. and is protected under U.S. and 8 // international copyright and other intellectual property 9 // laws. 10 // laws. 11 // DISCLAIMER 12 // File disclaimer is not a license and does not grant any 13 // rights to the materials distributed herewith. Except as 14 // otherwise provided in a valid license issued to you by 15 // Xilinx, and to the maximum extent permitted by applicable 16 // laws (1) THESE MATERIALS ARE MARE AVAILABLE "AS IS" AND 17 // WITH AL FAULTS, AND XILINX HEREBY DISCLAIME ALL WAREANTES	
<ul> <li>// lavs.</li> <li>// DISCLAIMER</li> <li>// This disclaimer is not a license and does not grant any</li> <li>// Trights to the materials distributed herewith. Except as</li> <li>// rights to the materials distributed herewith. Except as</li> <li>// vishts to the materials distributed herewith. Except as</li> <li>// vishts to the maximum extent permitted by applicable</li> <li>// lavs. (1) THESE MATERIALS ARE MADE AVAILABLE "AS 15" AND</li> <li>// VISHT LAURTS, MAILINA HEREMENT DISCLAIME ALL WARRANTES</li> </ul>	
<pre>10 // 11 // DISCLAIMER 12 // Tip disclaimer is not a license and does not grant any 13 // Tiphs to the materials distributed herewith. Except as 14 // otherwise provided in a valid license issued to you by 15 // Kilinx, and to the maximum extent permitted by applicable 16 // lav: (i) THESE MATERIALS ARE MADE AVAILABLE "AD IS" AND 17 // WITH ALL FAULTS, MAY LILNW REMERP DISCLAIMES ALL WARRANTES 17 // WITH ALL FAULTS, MAY ENDING AND ALL WARRANTES 18 // KILNA CONTINUE AND ALL WARRANTES 19 // WITH ALL FAULTS, MAY ENDING AND ALL WARRANTES 19 // WITH ALL FAULTS, MAY ENDING AND ALL WARRANTES 10 // WITH ALL FAULTS, MAY ENDING AND ALL WARRANTES 10 // WITH ALL FAULTS, MAY ENDING AND ALL WARRANTES 11 // WITH ALL FAULTS, MAY ENDING AND ALL WARRANTES 11 // WITH ALL FAULTS, MAY ENDING AND ALL WARRANTES 11 // WITH ALL FAULTS, MAY ENDING AND ALL WARRANTES 11 // WITH ALL FAULTS, MAY ENDING AND ALL WARRANTES 11 // WITH ALL FAULTS, MAY ENDING AND ALL WARRANTES 12 // WITH ALL FAULTS, MAY ENDING AND ALL WARRANTES 13 // WITH ALL FAULTS, MAY ENDING AND ALL WARRANTES 14 // WITH ALL FAULTS, MAY ENDING AND ALL WARRANTES 15 // WITH ALL FAULTS, MAY ENDING AND ALL WARRANTES 15 // WITH ALL FAULTS, MAY ENDING AND ALL WARRANTES 15 // WITH ALL FAULTS, MAY ENDING AND ALL WARRANTES 15 // WITH ALL FAULTS, MAY ENDING AND ALL WARRANTES 15 // WITH ALL FAULTS, MAY ENDING AND ALL WARRANTES 15 // WITH ALL FAULTS, MAY ENDING AND ALL WARRANTES 15 // WITH ALL FAULTS, MAY ENDING AND ALL WARRANTES 15 // WITH ALL MAY AND ALL WARRANTES 15 // WITH ALL MAY AND ALL WARRANTES 15 // WITH ALL W</pre>	
<ol> <li>// DISCLAIMER</li> <li>// TBIG direlaimer is not a license and does not grant any</li> <li>// Tgights to the materials distributed herewith. Except as</li> <li>// rights to the materials distributed herewith. Except as</li> <li>// vinherwise provided in a valid license issued to you by</li> <li>// Nilm, and to the maximum extent permitted by applicable</li> <li>// Nilm, and to the maximum extent permitted by a IS<sup>6</sup> ANN</li> <li>// VIITH L FAULTS, ANN VILINK HEREBY DISCLAIMES ALL BARRANTIES</li> </ol>	
12 // This disclaimer is not a license and does not grant any 13 // Triphs to the materials distributed herewith. Except as 14 // otherwise provided in a valid license issued to you by 15 // Xilinx, and to the maximum extent permitted by applicable 16 // lar: (1) THESE MATERIALS ARE MADE AVAILABLE "AS IS" AND 17 // WITH ALL FAULTS, AND XILINX HEREBY DISCLAIMS ALL WARRANTES	
13 // rights to the materials distributed herewith. Except as 14 // otherwise provided in a valid license issued to you by 15 // Xilina, and to the maximum extent permitted by applicable 16 // law: (1) THESE MATERIALS ARE MADE AVAILABLE "AS IS" AND 17 // WITH ALL FAULTS, ANY CHILINA HEREBY DISCLAHES ALL WARRANTES	
15 // Xilinx, and to the maximum extent permitted by applicable 16 // lev: (1) THESE MATERIALS ARE MADE AVAILABLE "AS IS" AND 17 // WITH ALL FAULTS, AND XILINN HEREBY DISCLAIMS ALL WARRANTIES	
16 // law: (1) THESE MATERIALS ARE MADE AVAILABLE "AS IS" AND 17 // WITH ALL FAULTS, AND XILINX HEREBY DISCLAIMS ALL WARRANTIES	
17 // With All FAults, And Xilinx Hereby disclaims All WARRANTIES	
18 // AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING	
19 // BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-	
19 // INFIGUENEMENT, OF FITTINESS FOR ANY PARTICULAR FURPOSE; and 0 // INFIGUENEMENT, OF FITTINESS FOR ANY PARTICULAR FURPOSE; and	
1 // (2) Xilinx shall not be liable (whether in contract or tort,	
22 // including negligence, or under any other theory of	
23 // liability) for any loss or damage of any kind or nature	
24 // related to, arising under or in connection with these	
25 // materials, including for any direct, or any indirect, 26 // special, incidental, or consequential loss or danace	
20 // incluental, incluental, or consequential loss or damage 27 // including loss of data, profils, goodvill, or any type of	
// loss or damage suffered as a result of any action brought	
29 // by a third party) even if such damage or loss was	
30 // reasonably foreseeable or Xilinx had been advised of the	
1 // possibility of the same.	
32 // CRITICAL APPLICATIONS	
33 // Killad arealcallos 34 // Killaw products are not designed or intended to be fail-	
35 // safe, or for use in any application requiring fail-safe	
36 // performance, such as life-support or safety devices or	
37 // systems, Class III medical devices, nuclear facilities,	
38 // applications related to the deployment of airbags, or any	
99 // other applications that could lead to death, personal 00 // injury, or severe property or environmental damage	
<pre>// injust; vi sovie projectory i controlational damage 4 // (injustical and collectively, "Critical 4)</pre>	
42 // Applications"). Customer assumes the sole risk and	
43 // liability of any use of Xilinx products in Critical	
44 // Applications, subject only to applicable laws and	
45 // regulations governing limitations on product liability. 46 //	
/ // THIS COPYRIGHT NOTICE AND DISCLAIMER MUST BE RETAINED AS	
48 // Part of this file at all times.	
49 //	
50 //	
51 // User entered comments	
Sa // None	
54 //	
55 //	
56 // Output Output Phase Duty Cycle FX-to-FX Phase 57 // Clock Freq (MHz) (degrees) (%) Jitter (ps) Error (ps)	
// caoca red (may (addeed) (s) accor (ba) are (ba)	
<b>59</b> //clk_5050.0000.00050.0167.017114.212	
60 //_clk_200_200.0000.00050.0126.455114.212	
El //Lk_40400.0000.00050.0111.164114.212	
62 // 63 //-	
64 // Input Clock Freq (MHz) Input Jitter (UI)	
65 //	
66 // _primary100.0000.010	
67 68 'timescale lps/lps	
(* CORE_GENERATION_INFO = "clk_module,clk_wiz_v5_3_1, {component_name=clk_module,use_phase_alignment=true,use_min_o_jitter=false,use_max_i_jitter=false,use_dyn_phase_shift	t=false,use
11 Z module clk module	
73 (	
74 // Clock in ports	
75 input clk_inl,	
76 // Clock out ports // 70 output clk 50,	
77 output clk_50, 78 output clk_200,	
7 output clk 400,	
80 // Status and control signals	
81 input reset,	
22 output locked 93 ):	
83 ); 94	
5 clk_module_clk_wiz inst	
86 (	
87 // Clock in ports	
88 .clk_inl(clk_inl), 89 // Clock out ports	
89 // Clock outp ports of .clk_Solck_50),	
<ul> <li>c.t20(c.tk_20),</li> <li>c.tk_20(c.tk_20),</li> </ul>	
2 (a) (a) (a) (a) (a) (a)	
3 // Status and control signals	
94 .reset(reset),	
95 Jocked (Jocked)	
96 );	
97	
97   50   endmodule	

1	
	// file: clk module.v
2	// ile: Cik_module.v
4	
5	
6	// This file contains confidential and proprietary information
7	// of Xilinx, Inc. and is protected under U.S. and
8	// international copyright and other intellectual property
9	// laws.
	1/
11	// DISCLAIMER
12	// This disclaimer is not a license and does not grant any
13	// rights to the materials distributed herewith. Except as
14	// otherwise provided in a valid license issued to you by
15	// Xilinx, and to the maximum extent permitted by applicable
16	// lav: (1) THESE MATERIALS ARE MADE AVAILABLE "AS IS" AND
17	// with all faults, and xilinx hereby disclaims all warranties
	// AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING
	// BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-
20	// INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and
21	// (2) Xilinx shall not be liable (whether in contract or tort,
	// including negligence, or under any other theory of
23	// liability) for any loss or damage of any kind or nature
24	// related to, arising under or in connection with these
	// materials, including for any direct, or any indirect,
26	
	// (including loss of data, profits, goodwill, or any type of
28	// loss of damage suffered as a result of any cycle of a company of the cycle of th
	// by a third party) even if such damage or loss vas
	// reasonably foreseeable or Xilinx had been advised of the
31	
32	
	// CRITICAL APPLICATIONS
34	// Xilinx products are not designed or intended to be fail-
35	// safe, or for use in any application requiring fail-safe
	// performance, such as life-support or safety devices or
37	// systems, Class III medical devices, nuclear facilities,
38	
	// other applications that could lead to death, personal
40	// injury, or severe property or environmental damage
	// (individually and collectively, "Critical
	// Applications"). Customer assumes the sole risk and
43	// liability of any use of Xilinx products in Critical
44	// Applications, subject only to applicable laws and
45	// regulations governing limitations on product liability.
46 47	
48	// THIS COPYRIGHT NOTICE AND DISCLAIMER MUST BE RETAINED AS // PART OF THIS FILE AT ALL TIMES.
49	// FARL OF ITIS FILE AI ALL TIMES.
50	//
51	// User entered comments
52	
53	
54	// None
	// None //
55	
55 56	// //
	//
56 57 58	// //- Output Dutput Phase Duty Cycle Fk-to-Fk Phase // Clock Freq (MHz) (degrees) (1) Jitter (ps) Error (ps)
56 57 58 59	// //- Output Dutput Phase Duty Cycle Fk-to-Fk Phase // Clock Freq (MHz) (degrees) (1) Jitter (ps) Error (ps)
56 57 58 59 60	// // Output Output Phase Duty Cycle Pi-to-Pk Phase // Clock Freq (Mis) (degrees) (i) Jitter (ps) Error (ps) //
56 57 58 59	// // Output Output Phase Duty Cycle Pi-to-Pk Phase // Clock Freq (Mis) (degrees) (i) Jitter (ps) Error (ps) //
56 57 58 59 60 61 62	// // Output Output Phase Duty Cycle PK-to-Pk Phase // Clock Freq (MHz) (degrees) (1) Jitter (ps) Error (ps) //
56 57 58 59 60 61 62 63	// // Output Output Phase Duty Cycle FX-to-FX Phase // Clock Freq (MHz) (degrees) (1) Jitter (ps) Error (ps) //
56 57 58 59 60 61 62 63 64	// //- // Output Output Phase Duty Cycle Fk-to-Fk Phase // Clock Freq (MfHz) (degrees) (1) Jitter (ps) Error (ps) //
56 57 58 59 60 61 62 63 64 65	// // Output Output Phase Duty Cycle EX-to-EX Phase // Clock Freq (MHz) (degrees) (1) Jitter (ps) Error (ps) //
56 57 58 59 60 61 62 63 64 65 66	// //- // Output Output Phase Duty Cycle Fk-to-Fk Phase // Clock Freq (MfHz) (degrees) (1) Jitter (ps) Error (ps) //
56 57 58 59 60 61 62 63 64 65 66 67	// // Clock Freq (Miz) (degrees) (1) Jitter (ps) Error (ps) // Clock Freq (Miz) (degrees) (1) Jitter (ps) Error (ps) // _clk_5050.0000.00050.0124.455114.212 // _clk_200_200.0000.00050.0124.455114.212 // _clk_400_400.0000.00050.0111.164114.212 // _retwork freq (Miz) Input Jitter (UI) // _primary100.0000.010
56 57 58 59 60 61 62 63 64 65 66	// // Output Output Phase Duty Cycle EX-to-EX Phase // Clock Freq (MHz) (degrees) (1) Jitter (ps) Error (ps) //
56 57 58 59 60 61 62 63 64 65 66 67 68	// // Clock Freq (Miz) (degrees) (1) Jitter (ps) Error (ps) // Clock Freq (Miz) (degrees) (1) Jitter (ps) Error (ps) // _clk_5050.0000.00050.0124.455114.212 // _clk_200_200.0000.00050.0124.455114.212 // _clk_400_400.0000.00050.0111.164114.212 // _retwork freq (Miz) Input Jitter (UI) // _primary100.0000.010
56 57 58 59 60 61 62 63 64 65 66 67 68 69	// //
56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72	// // Output Output Phase Duty Cycle EN-to-FK Phase // Clock Freq (NHz) (degrees) (1) Jitter (ps) Error (ps) //
56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73	// //
56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74	// // Clock Freq (MEz) (degrees) (1) Jitter (ps) Error (ps) // Clock Social Conditions (17, 017, 114.212) // clk 200 500.000 0.000 50.0 114.645 114.212 // clk 200 500.000 0.000 50.0 111.164 114.212 // clk 200 400.000 0.000 50.0 111.164 114.212 // clk 200 400.000 0.000 50.0 111.164 114.212 // clk 200 400.000 0.000 0.000 0.000 0.000 50.0 111.164 114.212 // clk 200 400.000 0.000 0.000 0.000 0.000 0.000 50.0 111.164 114.212 // clk 200 400.000 0.000 0.000 0.000 0.000 0.000 50.0 111.164 114.212 // clk 200 400.000 0.000 0.000 0.000 0.000 0.000 50.0 111.164 114.212 // clk 200 400.000 0.000 0.000 0.000 0.000 0.000 50.0 111.164 114.212 // clk 200 400.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 50.0 111.164 114.212
56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75	// // Output Output Phase Duty Cycle EX-to-EX Phase // Clock Freq (MHz) (degrees) (1) Jitter (p2) Error (pc) //
56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76	// // Output Output Phase Duty Cycle EX-to-EX Phase // Clock Freq (MHz) (degrees) (1) Jitter (p2) Error (pc) //
56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77	// // Output Output Phase Duty Cycle Ek-to-Ek Phase // Clock Freq (MHz) (degrees) (1) Jitter (p2) Error (pc) //
56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 172 73 74 75 76 77 78	// // Clock Freq (MEz) (degrees) (1) Jitter (ps) Error (ps) // Clt 50 50:00 0.000 50.0 124.212 // clt 20 50:000 0.000 50.0 124.4212 // clt 200 200.000 0.000 50.0 111.164 114.212 // clt 400 400.000 0.000 50.0 111.164 114.212 // clt 400 400.000 0.000 50.0 101.11.64 // Input Clock Freq (MEz) Input Jitter (UI) //
56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 5 76 77 78 79	// // Clock Freq (MEz) (degrees) (1) Jitter (ps) Error (ps) // Clt 50 50:00 0.000 50.0 124.212 // clt 20 50:000 0.000 50.0 124.4212 // clt 200 200.000 0.000 50.0 111.164 114.212 // clt 400 400.000 0.000 50.0 111.164 114.212 // clt 400 400.000 0.000 50.0 101.11.64 // Input Clock Freq (MEz) Input Jitter (UI) //
56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 57 68 77 77 78 980	//
56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 80 81	<pre>//</pre>
56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 9 80 81 82	//
56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 80 81	<pre>// // clock Preq (MED) (degrees) (1) Jitter (ps) Error (ps) // clt_s0_50.0000.00050.0147.017_114.212 // clt_s0_200.0000.00050.0111.164_114.212 // clt_s0_400.0000.00050.0111.164_114.212 // clt_s0_400.0000.00050.0111.164_114.212 // primary100.0000.010 *timescale lps/lps module clt_module_clt_wiz (// Clock in ports input clt_s1, // Clock of ports output clt_s0, output locked }; // Tput buffering //</pre>
56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 5 76 77 78 79 80 81 82 83	<pre>//</pre>
56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 79 80 81 82 83 84	<pre>// // clock Preq (MED) (degrees) (1) Jitter (ps) Error (ps) // clt_s0_50.0000.00050.0147.017_114.212 // clt_s0_200.0000.00050.0111.164_114.212 // clt_s0_400.0000.00050.0111.164_114.212 // clt_s0_400.0000.00050.0111.164_114.212 // primary100.0000.010 *timescale lps/lps module clt_module_clt_wiz (// Clock in ports input clt_s1, // Clock of ports output clt_s0, output locked }; // Tput buffering //</pre>
56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 273 74 75 76 77 78 80 81 82 83 84 85	<pre>//</pre>
56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 576 77 78 90 81 82 83 84 85 86	<pre>//</pre>
56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 677 78 79 80 82 83 84 85 68 83 84 85 86 83	<pre>//</pre>

91	// Clocking P	TMTTTIÆ
92	//	1011105
92		
93	// Testantist	on of the MACH DETMITIE
94	// Instantiat	on of the MMCM PRIMITIVE
95	// * Unuse	inputs are tied off
	// * Unuse	outputs are labeled unused
97		
98	wire c	k_50_clk_module;
99	wire c	k_200_clk_module;
100	wire c	k_400_clk_module;
101	wire c	k_out4_clk_module;
102	wire c	k_out5_clk_module;
103	wire c	k_out6_clk_module;
104	wire c	k_out7_clk_module;
105		
106	wire [15:0] d	unused;
107		dy_unused;
108		done_unused;
109		cked_int;
110		kfbout_clk_module;
111		kfbout_buf_clk_module;
112		kfboutb_unused;
113	wire clkout	
114	wire clkoutl	
115	wire clkout2	
	wire clkout3	
116 121	wire cikoucs	kfbstopped_unused;
122		kinstopped_unused;
123		set_high;
124		
125	MMCME2_ADV	
126	#(.BANDWIDTH	("OPTIMIZED"),
127	.CLKOUT4_CA	
128	.COMPENSATI	N ("ZHOLD"),
129	.STARTUP_WA	
130	.DIVCLK_DIV	
131		
131	.CLKFBOUT_M	LT_F (8.000), IASE (0.000).
132	.CLKFBOUT_P	ASE (0.000),
133	.CLKFBOOI_0	E_FINE_PS ("FALSE"),
134	.CLKOUTO_DI	
	.CLKOUT0_PH	SE (0.000),
136	.CLKOUTO_DU	"
137 138	.CLKOUTO_US	FINE_PS ("FALSE"),
139	.CLKOUT1_PH	SE (0.000),
140	.CLKOUT1_DU	Y_CYCLE (0.500), FINE_PS ("FALSE"),
141	.CLKOUT1_US	_FINE_PS ("FALSE"),
142	.CLKOUT2_DI	TIDE (2),
L43	.CLKOUT2_PH	SE (0.000),
44	.CLKOUT2_DU	Y_CYCLE (0.500),
45	.CLKOUT2_US	_FINE_PS ("FALSE"),
L46	.CLKIN1_PER	OD (10.0))
147	mmcm_adv_inst	
L48	// Output c	.ocks
L49	(	
.50	.CLKFBOUT	(clkfbout_clk_module),
.51	.CLKFBOUTB	(clkfboutb_unused),
152	.CLKOUTO	(clk 50 clk module),
153	.CLKOUTOB	(clkout0b_unused),
154	.CLKOUT1	(clk_200_clk_module),
155	.CLKOUT1B	(clkoutlb_unused),
156	.CLKOUT2	(clk_400_clk_module),
157	.CLKOUT2B	(clkout2b_unused),
158	.CLKOUT3	(clkout3_unused),
159	.CLKOUT3B	(clkout3b_unused),
160	.CLKOUT4	(clkout4 unused),
161	.CLKOUT5	<pre>(clkout4_unused), (clkout5_unused),</pre>
162	.CLKOUT6	(clkout6_unused),
163		ock control
163	.CLKFBIN	(clkfbout_buf_clk_module),
165	.CLKIN1	<pre>(clkibbut_but_clk_module), (clk_inl_clk_module),</pre>
165	.CLKINI	(libo)
		(1'b0),
167	// Tied to	always select the primary input clock
168	.CLKINSEL	(1'bl),
169	// Ports fo	dynamic reconfiguration
	.DADDR	(7'h0),
		(1'b0),
171	.DCLK	
171 172	.DEN	(1'b0),
171 172 173	.DEN .DI	(16'h0),
171 172 173 174	.DEN .DI .DO	(16'h0), (do_unused),
171 172 173 174 175	.DEN .DI .DO .DRDY	<pre>(16'h0), (do_unused), (drdy_unused),</pre>
171 172 173 174 175	.DEN .DI .DO	<pre>(16'h0), (do_unused), (drdy_unused),</pre>
171 172 173 174 175	.DEN .DI .DO .DRDY .DWE // Ports fo	<pre>(16'h0), (do_unused), (drdy_unused), (1'b0), ' dynamic phase shift</pre>
171 172 173 174 175 176 177	.DEN .DI .DO .DRDY .DWE // Ports fo .PSCLK	<pre>(16'h0), (do_unused), (drdy_unused), (1'b0), ' dynamic phase shift</pre>
171 172 173 174 175 176 177 178	.DEN .DI .DO .DRDY .DWE // Ports fo .PSCLK	<pre>(l6<sup>+</sup>h<sup>0</sup>), (dc_unused), (drdy_unused), (l<sup>+</sup>b<sup>0</sup>), <sup>+</sup> dynamic phase shift (l<sup>+</sup>b<sup>0</sup>),</pre>
178 179	.DEN .DI .DO .DRDY .DWE // Ports fo .PSCLK .PSEN	(16 %h0), (do_unused), (drdy unused), (1 %b0), (1 %b0), (1 %b0), (1 %b0),
171 172 173 174 175 176 177 178	.DEN .DI .DO .DRDY .DWE // Ports fo .PSCLK	<pre>(16'h0), (dc_unused), (drdy_unused), (1'b0), ' dynamic phase shift (1'b0),</pre>

<pre>181 .FSDOME (padone_unused), 183 // Other control and status signals 184 .CCKED (locked_int), 185 .CCKED (locked_int), 186 .CCKEDSTOPED (clkinstopped_unused), 187 .RST (reset_high); 188 assign reset_high = reset; 199 assign locked = locked_int; 199 // Clock Monitor clock assigning 199 // Clock Monitor clock assigning 190 // Clock Monitor clock assigning 191 // Clock Monitor clock assigning 192 // Clock Monitor clock assigning 193 // Clock Monitor clock assigning 194 // Output Duffering 195 // Clock Monitor clock assigning 195 // Clock Monitor clock assigning 196 (.O (clkfbout_buf_clk_module), 197 BUFS clkutl_buf clk_module), 198 (.O (clkfbout_plk_module)); 201 202 203 204 (.O (clkfbout_buf_clk_module)); 205 205 .I (clkfbout_buf 206 .I (clk_50, 207 208 209 209 209 201 202 203 204 (.O (clk_400, 204 .I (clk_400_clk_module)); 205 205 206 207 208 209 209 209 209 209 200 200 200</pre>			
<pre>13 .LOCKED (locked_int), 14 .CLKINSTOPFED (clkinstopped_unused), 15 .CLKINSTOPFED (clkinstopped_unused), 16 .CLKINSTOPFED (clkinstopped_unused), 17 .RST (reset_high); 18 assign reset_high = reset; 19 assign locked = locked_int; 19 //</pre>	181	. PSDONE	(psdone_unused),
<pre>164CLMINSTORED (clkinatopped_unused), 165CLMINSTORED (clkinatopped_unused), 167CLMINSTORED (l'hoi), 167</pre>	182	<pre>// Other control and</pre>	status signals
<pre>155</pre>	183	. LOCKED	(locked_int),
186       .FWEDENN (1'b0), ''''''''''''''''''''''''''''''''''''	184	.CLKINSTOPPED	
187       (reset_high));         188       assign reset_high = reset;         189       assign locked = locked_int;         180       issign locked = locked_int;         181       assign locked = locked_int;         182       // Clock Monitor clock assigning         184       // Output buffering         185       //	185	.CLKFBSTOPPED	(clkfbstopped_unused),
<pre>les</pre>	186	. PWRDWN	(1'b0),
<pre>199 assign reset_high = reset; 190 assign locked = locked_int; 192 // Clock Monitor clock assigning 193 //</pre>	187	.RST	(reset_high));
<pre>100 101 assign locked = locked_int; 102 // Clock Monitor clock assigning 103 // 104 // Output buffering 105 // 106 107 BUFS clkf_buf 108 (.0 (clkfbout_clk_module), 100 100 201 202 203 BUFG clkoutl_buf 204 (.0 (clk_50), 1 (clk_50), 1 (clk_50, 1 (clk_s0, 1 (cl</pre>	188		
<pre>19. assign locked = locked_int; 192 // Clock Monitor clock assigning 193 //</pre>	189	assign reset_high = res	et;
<pre>122 // Clock Monitor clock assigning 133 // 134 // Output buffering 135 // 14 147 BUFG clkr[buf 149 (.0 (clkFbout_clk_module), 149 .1 (clkfbout_clk_module)); 140 140 (.0 (clk_50), 141 (clk_50,clk_module)); 141 141 150 BUFG clkout3_buf 142 142 BUFG clkout3_buf 144 .1 (clk_400_clk_module)); 145 145 145 145 145 145 145 145 145 145</pre>	190		
<pre>193 // 194 // Output buffering 194 // Output buffering 195 // 196 197 BUFG clk1 buf 198 (.0 (clk1bout_buf_clk_module), 199 .I (clk1bout_clk_module)); 201 202 203 BUFG clkout1_buf 204 (.0 (clk_50, 205 .I (clk_50,k_module)); 205 201 202 203 204 204 204 204 204 204 204 204 204 204</pre>			int;
<pre>194 // Output buffering 195 // 196 BUFG clkf_buf 197 BUFG clkf_buf 198 (.0 (clkfbout_buf_clk_module), 199 . I (clkfbout_clk_module)); 201 202 203 BUFG clkoutl_buf 203 UFG clkoutl_buf 204 (.0 (clk_50, 205 . I (clk_50,clk_module)); 205 201 201 202 203 BUFG clkoutl_buf 203 UFG clkoutl_buf 203 UFG clkoutl_buf 203 UFG clkoutl_buf 204 (.0 (clk_400), 204 . I (clk_400_clk_module)); 205 205 207 207 207 207 207 207 207 207 207 207</pre>	192	// Clock Monitor clock as	signing
<pre>195 //</pre>	193	//	
<pre>196 197 BUFG elkf_buf 197 197 197 197 197 197 197 197 197 197</pre>			
<pre>197 BUFG clkf.buf 198 (.0 (clkfbout_cbf_clk_module), 199 .1 (clkfbout_clk_module)); 200 201 203 BUFG clkout_buf 204 (.0 (clk_50), 205 .1 (clk_50_clk_module)); 211 212 213 BUFG clkout3_buf 213 (.0 (clk_400, dlk_module)); 214 .1 (clk_400_clk_module)); 215 217</pre>	195	//	
<pre>198 (.0 (clkEbout_buf_clk_module), 199 .I (clkEbout_clk_module)); 201 202 203 BUFS clkoutl_buf 204 (.0 (clk_50), 205 .I (clk_50,tk_module)); 206 211 212 BUFS clkout3_buf 213 (.0 (clk_400), 214 .I (clk_400_clk_module)); 215 217 </pre>	196		
<pre>199 I (clkfbout_clk_module)); 200 201 202 203 BUFG clkoutl_buf 204 (.0 (clk_50), 205 I (clk_50,clk_module)); 211 212 212 BUFG clkout3_buf 213 (.0 (clk_400, dlk_module)); 214 I (clk_400_clk_module)); 215 216 217 </pre>	197	BUFG clkf_buf	
200 201 202 203 BUFG clkoutl_buf 204 (.0 (clk_50, 205 .I (clk_50,elk_module)); 206 211 212 BUFG clkout3_buf 213 (.0 (clk_400, dlk_module)); 214 .I (clk_400_clk_module)); 215 217	198	(.0 (clkfbout_buf_clk_	module),
<pre>201 202 203 BUFG clkouti_buf 204 (.0 (clk_50), 205 .I (clk_50_clk_module)); 206 201 202 202 203 C.0 (clk_400, 203 (.0 (clk_400_clk_module)); 204 .I (clk_400_clk_module)); 205 207 205 207 207 207 207 207 207 207 207 207 207</pre>	199	.I (clkfbout_clk_modu	le));
203 BUFG clkoutl_buf 204 (.0 (clk_50), 205 .I (clk_50,clk_module)); 207 208 209 209 209 209 209 209 209 209	200		
<pre>203 BUFG clouti_baf 204 (.0 (clk_50), 205 .I (clk_50,clk_module)); 206 201 202 BUFG clout3_baf 203 (.0 (clk_400, 204 .I (clk_400_clk_module)); 205 206 207</pre>	201		
<pre>204 (.0 (clk_50), .1 (clk_50_clk_module)); 206 211 212 BUFG clkout3_buf 213 (.0 (clk_400), 214 .I (clk_400_clk_module)); 215 216 217</pre>	202		
<pre>205 .I (clk_50_clk_module)); 206 211 212 BUFG clkout1_buf 213 (.0 (clk_400), 214 .I (clk_400_clk_module)); 215 216 217</pre>	203	BUFG clkoutl_buf	
206 tr (te	204	(.0 (clk_50),	
211 212 BUFG clkout1_baf 213 (.0 (clk_400), 214 .I (clk_400_clk_module)); 215 216 217	205	.I (clk_50_clk_modu	le));
<pre>212 BUFG clkout3_baf 213 (.0 (clk_400), 214 .I (clk_400_clk_module)); 215 216 217</pre>			
213 (.0 (clk_400), 214 .I (clk_400_clk_module)); 215 216 217			
<pre>214 .I (ol_400_olk_module)); 215 216 217</pre>			
215 216 217			
216 217		.I (clk_400_clk_mod	ule));
217			
218 endmodule			
	218	endmodule	

```
3
4
5
6
7
8
                   Company:
              // Engineer:
               /
// Create Date: 04/19/2017 04:41:54 PM
              // Design Name:
// Module Name: pingpong
             // Project Name:
// Target Devices:
// Tool Versions:
// Description:
  9
10
11
12
13
14
15
               / Dependencies:
              // Revision:
// Revision 0.01 - File Created
// Additional Comments:
  16
17
18
19
20
21
22
23
            24
25
26
27
28
29
30
31
32
33
34
35
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
            // complex input = (x + jz)
// complex output = (x + jz)
           module pingpong(
                   uile pingpong(
input wire [17:0] XqIn_PingPong_Real, // input Xq real component
input wire [17:0] XqIn_PingPong_Imag, // input Zq inag component
input wire clk, // input clock
input wire [3:0] counter, // input counter
output wire [17:0] XqOut_PingPongw_Real, // output Xq real component on a wire
output wire [17:0] XqOut_PingPongw_Imag // output Xq imag component on a wire
                   );
  52
53
54
55
56
57
58
59
60
61
62
63
64
65
                                                                                         // register for Ping real component
// register for Ping imag component
// register for Pong imag component
// register for Pong imag component
// register used to tranpose counter
// index milliar for bit
                   reg [17:0] XqPing_Real [7:0];
reg [17:0] XqPing_Imag [7:0];
reg [17:0] XqPong_Real [7:0];
reg [17:0] XqPong_Imag [7:0];
wire [2:0] tranpose;
                                                                                                        // index utilized for bit reversal
                    reg [2:0] indexbr;
                   wire pingpongstatus;
reg ping_loading;
reg pong_loading;
          assign XqOut_PingPongw_Real = (counter[3]==1'b() ? XqPing_Real[counter[2:0]] : XqPong_Real[counter[2:0]];
assign XqOut_PingPongw_Imag = (counter[3]==1'b() ? XqPing_Imag[counter[2:0]] : XqPong_Imag[counter[2:0]];
assign pingpongstatus = counter[3];
assign transpose = counter[4];
//always statement that occurs at every rising
//edge of the clock
                                                                                                                                                                                                                                                                       //XqOut_PingPong_Real; // register for output Xq real component
//XqOut PingPong Imag; // register for output Xq imag component
  // Initial block (Only executed once when simulation start)
                    initial begin
                          indexbr = 0;
                   end
           always@(posedge clk)
                    begin
                                    counter transposed due to bit re
                            indexbr <= {tranpose[0],tranpose[1],tranpose[2]};</pre>
                            //The Pong buffer is outputed when the Fing buffer is
//being filled up. Fing and Pong then alternate
//The NSB of counter is utilized as
//my switch bit. Buffers are complex numbers.
                            //Alternating loading Ping and Pong buffers
//based on MSB counter. Results in 16
                                      vele period.
                             if (counter[3] == 1'bl) begin
    XqPing_Real[indexbr] <= XqIn_PingPong_Real;</pre>
```



```
4
5
6
7
                     / Engineer:
                   // Create Date: 04/08/2017 10:24:56 AM
                   // Design Name:
                 // Design Name:
// Module Name: Radix2_BFM
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
  10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
                     / Dependencies:
                      /
/ Revision:
/ Revision 0.01 - File Created
/ Additional Comments:
              Ű
               ondule Radix2_BFM(
    input wire [17:0] Xq_Real,
    input wire [17:0] Xq_Imag,
    input wire [17:0] W_Real,
    input wire [17:0] W_Imag,
    input wire [17:0] Radix2_BFMOut_Real,
    output wire [17:0] Radix2_BFMOut_Imag
    );
                             );
                           (* dont_touch = "true" *)reg [17:0] TopMuxOutDelayl_Real; TopMuxOutDelay2_Real; TopMuxOutDelay3_Real;
(* dont_touch = "true" *)reg [17:0] TopMuxOutDelayl_Imag, TopMuxOutDelay2_Imag, TopMuxOutDelay3_Imag;
(* dont_touch = "true" *)wire [17:0] TopMuxOut_Real, BottomMuxOutw_Imag;
(* dont_touch = "true" *)wire [17:0] TopMuxOut_Malfscale_Real, BottomMuxOutw_Imalfscale_Real;
(* dont_touch = "true" *)wire [17:0] TopMuxOut_Malfscale_Real, BottomMuxOutw_Imalfscale_Real;
(* dont_touch = "true" *)wire [17:0] TopMuxOutw_Imalfscale_Real, BottomMuxOutw_Imalfscale_Real;
(* dont_touch = "true" *)wire [17:0] XuDelay1_Real;
(* dont_touch = "true" *)wire [17:0] XuDelay1_Real;
(* dont_touch = "true" *)wire [17:0] MultOutw_Real;
(* dont_touch = "true" *)wire [17:0] MultOutw_Real;
(* dont_touch = "true" *)wire [17:0] AdderOutw_Real;
(* dont_touch = "true" *)wire [17:0] AdderOutw_Real;
(* dont_touch = "true" *)wire [17:0] AdderOutw_Imag;
2 to 1 Multiplexer Instantiated for top -----
                             two2oneMux two2oneMux_Top
                                        .inl_real(Xq_Real),
.inl_imag(Xq_Imag),
.in2_real(XqDelayl_Real),
.in2_imag(XqDelayl_Imag),
.in2_imag(XqDelayl_Imag),
                                        .clk(~counterw[0]),
.out_real(TopMuxOutw_Real),
.out_imag(TopMuxOutw_Imag)
                             );
                                                           2 to 1 Multiplexer Instantiated for bottom ---
                             two2oneMux two2oneMux_Bottom
                                        .inl_real(Xq_Real),
                                        .inl_real(Xq_Real),

.inl_inag(Xq_Inag),

.in2_real(XqDelay1_Real),

.in2_imag(XqDelay1_Imag),

.clk(counterw[0]),

.out_real(BottomMuxOutw_Real),

.out_imag(BottomMuxOutw_Imag)
                           );
                             // ----- 18 bit by 18 bit Multiplier Instantiated ------
multiply_complex ComplexMultiply2Stage
                                         .clk(counterw[0]),
.a(BottomMuxOutw_halfscale_Real),
.jb(BottomMuxOutw_halfscale_Imag),
                                          .c(W Real),
                                        .jd(W_Imag),
.Xqout_real(MultOutw_Real),
.Xqout_imag(MultOutw_Imag)
                            );
                             // ----- 18 bit by 18 bit Adder Instantiated -----
addsub18x18 AdderandSubtractor1Stage
                                        .TopIn_real(TopMuxOutDelay3_Real),
.TopIn_imag(TopMuxOutDelay3_Imag),
.BottomIn_real(MultOutw_Real),
.BottomIn_imag(MultOutw_Imag),
                                          .clk(clk),
```

91	.counterw(counterw[0]),
92	.Xqout real(AdderOutw Real),
93	.Xgout imag(AdderOutw Imag)
94	);
95	
96	assign Radix2_BFMOut_Real = AdderOutw_Real;
97	assign Radix2_BFMOut_Imag = AdderOutw_Imag;
98	assign TopMuxOutw_halfscale_Real = TopMuxOutw_Real >> 1;
99	assign TopMuxOutw_halfscale_Imag = TopMuxOutw_Imag >> 1;
100	assign BottomMuxOutw_halfscale_Real = BottomMuxOutw_Real >> 1;
101	assign BottomMuxOutw_halfscale_Imag = BottomMuxOutw_Imag >> 1;
102	
103 5	always@(posedge clk)
104 🤅	begin
105	TopMuxOutDelayl_Real <= TopMuxOutw_halfscale_Real;
106	TopMuxOutDelayl_Imag <= TopMuxOutw_halfscale_Imag;
107	TopMuxOutDelay2_Real <= TopMuxOutDelay1_Real;
108	TopMuxOutDelay2_Imag <= TopMuxOutDelay1_Imag;
109	TopMuxOutDelay3_Real <= TopMuxOutDelay2_Real;
110	TopMuxOutDelay3_Imag <= TopMuxOutDelay2_Imag;
111	XqDelay1_Real <= Xq_Real;
112	XqDelay1_Imag <= Xq_Imag;
113 🤅	end
114	
115	endmodule

1	timescale lns / lps
2	
3	// Company:
4	// Engineer:
5	//
6	// Create Date: 04/05/2017 03:43:36 PM
7	// Design Name: Complex Mutiplexer
8	// Module Name: two2oneMux
9	// Project Name:
10	// Target Devices:
11	// Tool Versions:
12	// Description:
13	//
14	// Dependencies:
15	//
16	// Revision:
17	// Revision 0.01 - File Created
18	// Additional Comments:
19	//
20	///////////////////////////////////////
21	
22 8	module two2oneMux (
23	input wire signed [17:0] inl_real, // a
24	input wire signed [17:0] inl_imag, // jb
25	input wire signed [17:0] in2_real, // c
26	input wire signed [17:0] in2_imag, // jd
27	input wire clk,
28	output wire signed [17:0] out_real, // c or a
29	output wire signed [17:0] out_imag // jd or jb
30	);
31	
32	assign out real = (clk==1'b0) ? in2 real : in1 real; //Real Component
33	assign out imag = (clk==1'b0) ? in2 imag : inl imag; //Imag Component
34 (	
	r

10	module multiply complex(
2	input wire clk.
3	input wire [17:0] a,
4	
	input wire [17:0] jb,
5	input wire [17:0] c,
6	input wire [17:0] jd,
7	output wire [17:0] Xgout_real,
8	output wire [17:0] Xgout_imag
9	);
10	
11	(* dont_touch = "true" *) reg sign_mult_left_Real, sign_mult_right_Real, sign_mult_left_Imag, sign_mult_right_Imag;
12	<pre>(* done_couch = "true" * / wire sign a; (* done_touch = "true" * / wire sign a;</pre>
13	(* dont_touch = "true" *) wire sign_jb;
14	(* dont_touch = "true" *) wire sign_C;
15	(* dont_touch = "true" *) wire sign_jd;
16	(* dont_touch = "true" *) wire [16:0] unsigned a; //
17	(* dont touch = "true" *) wire [16:0] unsigned jb; //
18	(* dont touch = "true" *) wire [16:0] unsigned c; //
19	(* dont touch = "true" *) wire [16:0] unsigned jd; //
20	(* done_touch = "true" *) wire [16:0] trunc_left_Real; //(a*c)
21	(* dont_touch = "true" *) wire [16:0] trunc_right_Real; //(jb/jd)
22	(* dont_touch = "true" *) wire [16:0] trunc_left_Imag; //(a'jd)
23	(* dont_touch = "true" *) wire [16:0] trunc_right_Imag; //(jb*c)
24	(* dont_touch = "true" *) wire [17:0] signed_left_Real; //(a*c)
25	(* dont_touch = "true" *) wire [17:0] signed_right_Real; //(jb/jd)
26	(* dont_touch = "true" *) wire [17:0] signed_left_Imag; //(a*jd)
27	(* dont_touch = "true" *) wire [17:0] signed_right_Imag; //(jb*c)
28	
29	
30	//performs unsigned fixed point multiplication. truncates output reducing precision.
31	//returns 17 bits. Inputs are a clock, 17 bit unsigned fixed point numbers labeled as
32	<pre>//unsigned_a, unsigned_jb, unsigned_c, and unsigned_jd. Outputs are 17 bit unsigned fixed point</pre>
33	//numbers labeled as left Real
34	mult18x18 unsigned_multiply
35	
36	.clk(clk),
37	.a(unsigned a),
38	-jb(unsigned_jb),
39	.c(unsigned_c),
40	-id(unsigned_),
41	.Ju(umargher_Ju), .Out_left_Real(trunc_left_Real),
42	
	.Out_right_Real(trunc_right_Real),
43	.Out_left_Imag(trunc_left_Imag),
44	.Out_right_Imag(trunc_right_Imag));
45	
46	//2'c operator following multiplication. Concatenates the sign bit with a 2'c number. returns 18 bits
46 47	<pre>twoCompRedo twoC_leftReal(.signbit(sign_mult_left_Real),.number(trunc_left_Real),.sign_number(signed_left_Real));</pre>
46 47 48	<pre>twoCompRedo twoC_leftReal(.signbit(sign_mult_left_Real),.number(trunc_left_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_rightReal(.signbit(sign_mult_right_Real),.number(trunc_right_Real),.sign_number(signed_right_Real));</pre>
46 47 48 49	<pre>twoCompRedo twoC_leftReal(.signbit(sign_mult_left_Real),.number(trunc_left_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_ightReal(.signbit(sign_mult_gint_Real),.number(trunc_ight_Real),.sign_number(signed_ight_Real)); twoCompRedo twoC_leftImag(.signbit(sign_mult_gint_Imag),.mumber(trunc_left_Imag),.sign_number(signed_left_Imag));</pre>
46 47 48	<pre>twoCompRedo twoC_leftReal(.signbit(sign_mult_left_Real),.number(trunc_left_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_rightReal(.signbit(sign_mult_right_Real),.number(trunc_right_Real),.sign_number(signed_right_Real));</pre>
46 47 48 49 50 51	<pre>twoCompRedo twoC_lettReal(signbit(sign_mult_left_Real),.number(trunc_left_Real),sign_number(signed_left_Real)); twoCompRedo twoC_inthReal(signbit(sign_mult_right_Real),.number(trunc_left_Real),sign_number(signed_left_Real)); twoCompRedo twoC_leftImag(signbit(sign_mult_left_Imag),.number(trunc_left_Imag),sign_number(signed_right_Imag)); twoCompRedo twoC_inghtImag(signbit(sign_mult_right_Imag),.number(trunc_right_Imag),.sign_number(signed_right_Imag));</pre>
46 47 48 49 50	<pre>twoCompRedo twoC_leftReal(.signbit(sign_mult_left_Real),.number(trunc_left_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_ightReal(.signbit(sign_mult_gint_Real),.number(trunc_ight_Real),.sign_number(signed_ight_Real)); twoCompRedo twoC_leftImag(.signbit(sign_mult_gint_Imag),.mumber(trunc_left_Imag),.sign_number(signed_left_Imag));</pre>
46 47 48 49 50 51	<pre>twoCompRedo twoC_lettReal(signbit(sign_mult_left_Real),.number(trunc_left_Real),sign_number(signed_left_Real)); twoCompRedo twoC_inthReal(signbit(sign_mult_right_Real),.number(trunc_left_Real),sign_number(signed_left_Real)); twoCompRedo twoC_leftImag(signbit(sign_mult_left_Imag),.number(trunc_left_Imag),sign_number(signed_right_Imag)); twoCompRedo twoC_inghtImag(signbit(sign_mult_right_Imag),.number(trunc_right_Imag),.sign_number(signed_right_Imag));</pre>
46 47 48 49 50 51 52	<pre>twoCompRedo twoC_leftReal(signbit(sign_mult_left_Real),.number(trunc_left_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_iftReal(signbit(sign_mult_left_Real),.number(trunc_left_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_leftImag(.signbit(sign_mult_left_Imag),.number(trunc_left_Imag),.sign_number(signed_left_Imag)); twoCompRedo twoC_iftRimag(.signbit(sign_mult_left_Imag),.number(trunc_left_Imag),.sign_number(signed_right_Imag)); twoCompRedo twoC_iftRimag(.signbit(sign_mult_right_Imag),.number(trunc_left_Imag),.sign_number(signed_right_Imag)); twoCompRedo twoC_iftRimag(.signbit(sign_mult_right_Imag),.number(trunc_right_Imag),.sign_number(signed_right_Imag)); //2'c operatof prior to multiplication. returns unsigned number. returns 17 bits twoComp insts(.In(s),.Out(unsigned_s)); //extracts data portion a</pre>
46 47 48 49 50 51 52 53 53	<pre>twoCompRedo twoC_leftReal(.signbit(sign_mult_left_Real),.number(trunc_left_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_rightReal(.signbit(sign_mult_right_Real),.number(trunc_right_Real),.sign_number(signed_right_Real)); twoCompRedo twoC_erightRead(.signbit(sign_mult_right_Imag),.number(trunc_left_Imag),.sign_number(signed_right_Reag)); twoCompRedo twoC_rightImag(.signbit(sign_mult_right_Imag),.number(trunc_right_Imag),.sign_number(signed_right_Imag)); twoCompRedo twoC_rightImag(.signbit(sign_mult_right_Imag),.number(trunc_right_Imag),.sign_number(signed_right_Imag)); //2'c operatof prior to multiplication. returns unsigned number. returns 17 bits twoComp Inste(.in(a),.Out(unsigned_j)); //extracts data portion a twoComp Inste(.in(a)).Out(unsigned_j)); //extracts data portion jb</pre>
46 47 48 49 50 51 52 53 54 55	<pre>twoCompRedo twoC_leftReal(signbit(sign_mult_left_Real),.number(trunc_left_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_rightReal(signbit(sign_mult_left_Real),.number(trunc_left_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_leftImsq(signbit(sign_mult_left_Imsq),.number(trunc_left_Imsq),.sign_number(signed_left_Imsq)); twoCompRedo twoC_rightImsq(.signbit(sign_mult_right_Imsq),.number(trunc_left_Imsq),.sign_number(signed_right_Imsq)); twoCompRedo twoC_rightImsq(.signbit(sign_mult_right_Imsq),.number(trunc_right_Imsq),.sign_number(signed_right_Imsq)); //2'c operatof prior to multiplication. returns unsigned number. returns 17 bits twoComp_inste(.in(s),.Out(unsigned_B)); //extracts data portion a twoComp_inste(.in(s)).Out(unsigned_c)); //extracts data portion o</pre>
46 47 48 49 50 51 52 53 54 55 55	<pre>twoCompRedo twoC_leftReal(.signbit(sign_mult_left_Real),.number(trunc_left_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_rightReal(.signbit(sign_mult_right_Real),.number(trunc_right_Real),.sign_number(signed_right_Real)); twoCompRedo twoC_erightRead(.signbit(sign_mult_right_Imag),.number(trunc_left_Imag),.sign_number(signed_right_Reag)); twoCompRedo twoC_rightImag(.signbit(sign_mult_right_Imag),.number(trunc_right_Imag),.sign_number(signed_right_Imag)); twoCompRedo twoC_rightImag(.signbit(sign_mult_right_Imag),.number(trunc_right_Imag),.sign_number(signed_right_Imag)); //2'c operatof prior to multiplication. returns unsigned number. returns 17 bits twoComp Inste(.in(a),.Out(unsigned_j)); //extracts data portion a twoComp Inste(.in(a)).Out(unsigned_j)); //extracts data portion jb</pre>
46 47 48 49 50 51 52 53 54 55 56 57	<pre>twoCompRedo twoC_leftReal(signbit(sign_mult_left_Real),.number(trunc_left_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_rightReal(signbit(sign_mult_lift_Real),.number(trunc_right_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_rightImag(signbit(sign_mult_lift_Imag),.number(trunc_left_Imag),.sign_number(signed_right_Real)); twoCompRedo twoC_rightImag(signbit(sign_mult_right_Imag),.number(trunc_right_Imag),.sign_number(signed_right_Imag)); twoCompRedo twoC_rightImag(signbit(sign_mult_right_Imag),.number(trunc_right_Imag),.sign_number(signed_right_Imag)); //2/o operatof prior to multiplication. returns unsigned number. returns 17 bits twoComp_insts(.in(s),.Out(unsigned_s)); //extracts data portion a twoComp_insts(.in(s)).Out(unsigned_g)); //extracts data portion o twoComp_insts(.in(s),.Out(unsigned_jd)); //extracts data portion o d</pre>
46 47 48 49 50 51 52 53 54 55 56 57 58	<pre>twoCompRedo twoC_lettReal(signbit(sign_mult_left_Real),.number(trunc_left_Real),.sign_mumber(signed_left_Real)); twoCompRedo twoC_leftReal(signbit(sign_mult_left_Real),.number(trunc_left_Imag),.sign_number(signed_left_Real)); twoCompRedo twoC_leftImag(.signbit(sign_mult_left_Imag)number(trunc_left_Imag),.sign_number(signed_idpt_Imag)); twoCompRedo twoC_leftImag(.signbit(sign_mult_left_Imag),.number(trunc_right_Imag),.sign_number(signed_right_Imag)); twoCompRedo twoC_leftImag(.signbit(sign_mult_right_Imag),.number(trunc_right_Imag),.sign_number(signed_right_Imag)); //2'c operatof prior to multiplication. returns unsigned number. returns 17 bits twoComp insta(.In(s),.Out(unsigned_])); //extracts data portion a twoComp insta(.In(s),.Out(unsigned_])); //extracts data portion jb twoComp insta(.In(s),.Out(unsigned_d)); //extracts data portion c twoComp insta(.In(s)).cout(unsigned_d)); //extracts data portion jd // saves sign bit. returns 1 bit</pre>
46 47 48 49 50 51 52 53 54 55 56 57 58 59	<pre>twoCompRedo twoC_lettReal(.signbit(sign_mult_left_Real),.number(trunc_left_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_leftReal(.signbit(sign_mult_right_Real),.number(trunc_left_Real),.sign_number(signed_right_Real)); twoCompRedo twoC_rightRead(.signbit(sign_mult_right_Imag),.number(trunc_left_Imag),.sign_number(signed_right_Real)); twoCompRedo twoC_rightImag(.signbit(sign_mult_right_Imag),.number(trunc_right_Imag),.sign_number(signed_right_Imag)); twoCompRedo twoC_rightImag(.signbit(sign_mult_right_Imag),.number(trunc_right_Imag),.sign_number(signed_right_Imag)); //S'c operatof prior to multiplication. returns unsigned number. returns 17 bits twoComp inate(.In(s),.Out(unsigned_s)); //extracts data portion s twoComp inate(.In(s),.Out(unsigned_s)); //extracts data portion o twoComp inate(.In(s),.Out(unsigned_s)); //extracts data portion o twoComp instd(.In(s),.Out(unsigned_s)); //extracts data p</pre>
46 47 48 49 50 51 52 53 54 55 56 57 58 59 60	<pre>twoCompRedo twoC_lettReal(signput(sign_mult_left_Real),.number(trunc_left_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_leftReal(signput(sign_mult_lift_Real),.number(trunc_left_Imag).sign_number(signed_left_Imag)); twoCompRedo twoC_leftImag(.signbit(sign_mult_lift_Real),.number(trunc_left_Imag),.sign_number(signed_left_Imag)); twoCompRedo twoC_leftImag(.signbit(sign_mult_lift_Imag),.number(trunc_left_Imag),.sign_number(signed_right_Real)); twoCompRedo twoC_leftImag(.signbit(sign_mult_lift)_Imag),.number(trunc_left_Imag),.sign_number(signed_right_Imag)); //2'c operator prior to multiplication. returns unsigned number.returns 17 bits twoComp_insta(.In(s),.Out(unsigned_s)); //extracts data portion a twoComp_insta(.In(s)).Out(unsigned_s)); //extracts data portion jb twoComp_insta(.In(s),.Out(unsigned_jb)); //extracts data portion jc twoComp_insta(.In(s),.Out(unsigned_jb)); //extracts data portion jc twoComp_insta(.In(s)).Out(unsigned_jb); //extracts data portion jc itwoComp_insta(.In(s)).Out(unsigned_jb); //extracts data portion jc itwoComp_insta(.In(s)).(j).(j).(j).(j).(j).(j).(j).(j).(j).(</pre>
46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61	<pre>twoCompRedo twoC_leftReal(.signbit(sign_mult_left_Real),.number(trunc_left_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_leftReal(.signbit(sign_mult_right_Real),.number(trunc_right_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_ightRead(.signbit(sign_mult_right_Imag),.number(trunc_right_Imag),.sign_number(signed_right_Real)); twoCompRedo twoC_ightImag(.signbit(sign_mult_right_Imag),.number(trunc_right_Imag),.sign_number(signed_right_Imag)); //2/c operatof prior to multiplication. returns unsigned number. returns 17 bits twoComp Insta(.In(a),.Out(unsigned_a)); //extracts data portion a twoComp Insta(.In(a),.Out(unsigned_jb)); //extracts data portion a twoComp Instd(.In(g),.Out(unsigned_jb)); //extracts data portion c twoComp Instd(.In(a),.Out(unsigned_jb)); //extracts data portion f twoComp Instd(.In(a),.Out(unsigned_jb)); //extracts data portion f twoComp Instd(.In(a),.Out(unsigned_d)); //extracts data portion f twoComp Instd(.In(a), Int(unsigned_d)); //extracts data portion f twoComp Instd(.In(a), Int(unsigned_d)); //extracts data portion f assign sign_a = a[17]; //extracts sign of a assign sign_a = a[17]; //extracts sign of fb assign sign_a = clard(assign of a assign sign_a = clard(assign of c); //extracts sign of c</pre>
46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62	<pre>twoCompRedo twoC_lettReal(signput(sign_mult_left_Real),.number(trunc_left_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_leftReal(signput(sign_mult_lift_Real),.number(trunc_left_Imag).sign_number(signed_left_Imag)); twoCompRedo twoC_leftImag(.signbit(sign_mult_lift_Real),.number(trunc_left_Imag),.sign_number(signed_left_Imag)); twoCompRedo twoC_leftImag(.signbit(sign_mult_lift_Imag),.number(trunc_left_Imag),.sign_number(signed_right_Real)); twoCompRedo twoC_leftImag(.signbit(sign_mult_lift)_Imag),.number(trunc_left_Imag),.sign_number(signed_right_Imag)); //2'c operator prior to multiplication. returns unsigned number.returns 17 bits twoComp_insta(.In(s),.Out(unsigned_s)); //extracts data portion a twoComp_insta(.In(s)).Out(unsigned_s)); //extracts data portion jb twoComp_insta(.In(s),.Out(unsigned_jb)); //extracts data portion jc twoComp_insta(.In(s),.Out(unsigned_jb)); //extracts data portion jc twoComp_insta(.In(s)).Out(unsigned_jb); //extracts data portion jc itwoComp_insta(.In(s)).Out(unsigned_jb); //extracts data portion jc itwoComp_insta(.In(s)).(j).(j).(j).(j).(j).(j).(j).(j).(j).(</pre>
46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63	<pre>twoCompRedo twoC_lettReal(signpit(sign_mult_left_Real),.number(trunc_left_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_leftReal(signpit(sign_mult_right_Real),.number(trunc_left_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_leftImag(signbit(sign_mult_right_Real),.number(trunc_left_Imag),.sign_number(signed_right_Real)); twoCompRedo twoC_leftImag(signbit(sign_mult_right_Imag),.number(trunc_left_Imag),.sign_number(signed_right_Imag)); //2/c operatof prior to multiplication. returns unside(trunc_right_Imag),.sign_number(signed_right_Imag)); //2/c operatof prior to multiplication. returns unside data portion a twoComp insta(.In(s),.out(unsigned_s)); //extracts data portion a twoComp insta(.In(s),.out(unsigned_s)); //extracts data portion o twoComp insta(.In(s),.out(unsigned_s)); //extracts data portion o twoComp insta(.In(s),.out(unsigned_s)); //extracts data portion o twoComp insta(.In(s)).out(unsigned_s)); //extracts data portion o twoComp insta(.In(s))./out(unsigned_s)); //extracts data portion o assign sign_s] = a[17]; //extracts sign of a assign sign_s] = a[17]; //extracts sign of jb assign sign_s] = a[17]; //extracts sign of jd</pre>
46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62	<pre>twoCompRedo twoC_leftReal(.signbit(sign_mult_left_Real),.number(trunc_left_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_leftReal(.signbit(sign_mult_right_Real),.number(trunc_right_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_ightRead(.signbit(sign_mult_right_Imag),.number(trunc_right_Imag),.sign_number(signed_right_Real)); twoCompRedo twoC_ightImag(.signbit(sign_mult_right_Imag),.number(trunc_right_Imag),.sign_number(signed_right_Imag)); //2/c operatof prior to multiplication. returns unsigned number. returns 17 bits twoComp Insta(.In(a),.Out(unsigned_a)); //extracts data portion a twoComp Insta(.In(a),.Out(unsigned_jb)); //extracts data portion a twoComp Instd(.In(g),.Out(unsigned_jb)); //extracts data portion c twoComp Instd(.In(a),.Out(unsigned_jb)); //extracts data portion f twoComp Instd(.In(a),.Out(unsigned_jb)); //extracts data portion f twoComp Instd(.In(a),.Out(unsigned_c)); //extracts data portion f twoComp Instd(.In(a), Int(unsigned_c)); //extracts data portion f twoComp Instd(.In(a), Int(unsigned_c)); //extracts data portion f assign sign_a = a[17]; //extracts sign of a assign sign_a = a[17]; //extracts sign of fb assign sign_a = clard(assign of a assign sign_a = clard(assign of c); //extracts sign of c</pre>
46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63	<pre>twoCompRedo twoC_lettReal(signpit(sign_mult_left_Real),.number(trunc_left_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_leftReal(signpit(sign_mult_right_Real),.number(trunc_left_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_leftImag(signbit(sign_mult_right_Real),.number(trunc_left_Imag),.sign_number(signed_right_Real)); twoCompRedo twoC_leftImag(signbit(sign_mult_right_Imag),.number(trunc_left_Imag),.sign_number(signed_right_Imag)); //2/c operatof prior to multiplication. returns unside(trunc_right_Imag),.sign_number(signed_right_Imag)); //2/c operatof prior to multiplication. returns unside data portion a twoComp insta(.In(s),.out(unsigned_s)); //extracts data portion a twoComp insta(.In(s),.out(unsigned_s)); //extracts data portion o twoComp insta(.In(s),.out(unsigned_s)); //extracts data portion o twoComp insta(.In(s),.out(unsigned_s)); //extracts data portion o twoComp insta(.In(s)).out(unsigned_s)); //extracts data portion o twoComp insta(.In(s))./out(unsigned_s)); //extracts data portion o assign sign_s] = a[17]; //extracts sign of a assign sign_s] = a[17]; //extracts sign of jb assign sign_s] = a[17]; //extracts sign of jd</pre>
46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64	<pre>twoCompRedo twoC_leftReal(signbit(sign_mult_left_Real),.number(trunc_left_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_rightReal(signbit(sign_mult_left_Enag),.number(trunc_left_Real),.sign_number(signed_left_Enag)); twoCompRedo twoC_rightImag(.signbit(sign_mult_left_Imag),.number(trunc_left_Imag),.sign_number(signed_right_Real)); twoCompRedo twoC_rightImag(.signbit(sign_mult_right_Imag),.number(trunc_left_Imag),.sign_number(signed_right_Imag)); twoCompRedo twoC_rightImag(.signbit(sign_mult_right_Imag),.number(trunc_right_Imag),.sign_number(signed_right_Imag)); //2'c operatof prior to multiplication. returns unsigned number. returns 17 bits twoComp_inste(.in(s),.Out(unsigned_B)); //extracts data portion a twoComp_inste(.in(s),.Out(unsigned_g)); //extracts data portion o twoComp_inste(.in(s),.Out(unsigned_g)); //extracts data portion jd // saves sign bit. returns 1 bit assign sign_g = a[17]; //extracts sign of jb assign sign_g = c[17]; //extracts sign of jb assign sign_g = c[17]; //extracts sign of jd assign sign_g = a[17]; //extracts sign of jd assign sign_g = a[17]; //extracts sign of jd</pre>
46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65	<pre>twoCompRedo twoC_leftReal(signbit(sign_mult_left_Real),.number(trunc_left_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_rightReal(signbit(sign_mult_left_Enam),.number(trunc_left_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_rightRead(.signbit(sign_mult_left_Imag),.number(trunc_left_Imag),.sign_number(signed_right_Real)); twoCompRedo twoC_rightRead(.signbit(sign_mult_left_Imag),.number(trunc_left_Imag),.sign_number(signed_right_Imag)); twoCompRedo twoC_rightRead(.signbit(sign_mult_right_Imag),.number(trunc_right_Imag),.sign_number(signed_right_Imag)); //2'c operatof prior to multiplication. returns unsigned number. returns 17 bits twoComp_inste(.in(s),.Out(unsigned_a)); //extracts data portion a twoComp_inste(.in(s),.Out(unsigned_a)); //extracts data portion o twoComp_inste(.in(s)).Out(unsigned_jo)); //extracts data portion o twoComp_inst(.in(s)).Out(unsigned_jo)); //extracts data portion jd // saves sign bit. returns 1 bit assign sign_a = a(17); //extracts sign of jb assign sign_d = jb(17); //extracts sign of jb assign sign_d = b(17); //extracts sign of jd assign sign_d = signed_left_Real = signed_right_Real; assign Xgout_real = signed_left_Real = signed_right_Real; assign Xgout_imag = signed_left_Imag + signed_right_Imag;</pre>
46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66	<pre>twoCompRedo twoC_leftReal(signbit(sign_mult_left_Real),.number(trunc_left_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_idptReal(signbit(sign_mult_left_Real),.number(trunc_left_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_leftImag(.signbit(sign_mult_right_Real),.number(trunc_left_Imag),.sign_number(signed_left_Imag)); twoCompRedo twoC_idptImag(.signbit(sign_mult_right_Imag),.number(trunc_left_Imag),.sign_number(signed_right_Real)); twoCompRedo twoC_idptImag(.signbit(sign_mult_right_Imag),.number(trunc_right_Imag),.sign_number(signed_right_Imag)); //2'c operatof prior to multiplication. returns unsigned number. returns 17 bits twoComp_inste(.In(b)).Out(unsigned_a)); //extracts data portion a twoComp_inste(.In(b)).Out(unsigned_a)); //extracts data portion a twoComp_inste(.In(b),.Out(unsigned_jb)); //extracts data portion c twoComp_inste(.In(b),.Out(unsigned_jb)); //extracts data portion c twoComp_inste(.In(b),.Out(unsigned_jb)); //extracts data portion c twoComp_inste(.In(b)).(Jut(unsigned_jb)); //extracts data portion c assign sign_g_= = c[17]; //extracts sign of a assign sign_g_= = c[17]; //extracts sign of c assign sign_g_= = c[17]; //extracts sign of jd assign sign_treal = signed_left_Real = signed_right_Real; assign Xgout_imag = signed_left_Read = signed_right_Imag; // Initial block (Only executed once when simulation start)</pre>
46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67	<pre>twoCompRedo twoC_leftReal(signbit(sign_mult_left_Real),.number(trunc_left_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_rightReal(signbit(sign_mult_left_Enam),.number(trunc_left_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_leftImag(signbit(sign_mult_left_Imag),.number(trunc_left_Imag),.sign_number(signed_left_Imag)); twoCompRedo twoC_rightImag(signbit(sign_mult_left_Imag),.number(trunc_left_Imag),.sign_number(signed_right_Real)); //2/c operatof prior to multiplication. returns unsigned number. returns 17 bits twoComp Insta(.in(s),.Out(unsigned_a)); //extracts data portion a twoComp Insta(.in(s),.Out(unsigned_a)); //extracts data portion o twoComp Insta(.in(s),.Out(unsigned_a)); //extracts data portion o twoComp Insta(.in(s),.Out(unsigned_jb)); //extracts data portion o twoComp Insta(.in(s),.Out(unsigned_jb)); //extracts data portion o twoComp Insta(.in(s)).Out(unsigned_jb); //extracts data portion o twoComp Insta(.in(s)).(Nutunsigned_jb)); //extracts data portion o twoComp Insta(.in(s)).(Nutunsigned_jb); //extracts data portion o twoComp Insta(.in(s)).(Nutunsigned_jb)); //extracts data portion jd // saves sign bit. returns 1 bit assign sign_a = a[17]; //extracts sign of jb assign_sign_a = a[17]; //extracts sign of jb assign_sign_a = a[17]; //extracts sign of jd assign Xgout_real = signed_left_Real = signed_right_Real; assign Xgout_real = signed_left_Imag + signed_right_Imag; // Initial block (Only executed once when simulation start) initial</pre>
46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68	<pre>twoCompRedo twoC_leftReal(signpult(sign_mult_left_Real),.number(trunc_left_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_idptReal(signbit(sign_mult_left_Emag),.number(trunc_left_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_leftImag(.signbit(sign_mult_right_Real),.number(trunc_left_Imag),.sign_number(signed_right_Real)); twoCompRedo twoC_idptImag(.signbit(sign_mult_right_Imag),.number(trunc_left_Imag),.sign_number(signed_right_Imag)); twoCompRedo twoC_idptImag(.signbit(sign_mult_right_Imag),.number(trunc_right_Imag),.sign_number(signed_right_Imag)); //2'c operatof prior to multiplication. returns unsigned number. returns 17 bits twoComp_inst(.In(s),.Out(unsigned_s)); //extracts data portion a twoComp_inst(.In(s),.Out(unsigned_g)); //extracts data portion o twoComp_inst(.In(s),.Out(unsigned_j)); //extracts data portion o twoComp_inst(.In(s),.Out(unsigned_j)); //extracts data portion o twoComp_inst(.In(s)),.Out(unsigned_j)); //extracts data portion o data assign_sign_jb = b[1]? //extracts sign of a assign_sign_jb = b[1]? //extracts sign of c assign_sign_jd = signed_left_Imag = signed_right_Real; assign_sign_id = signed_left_Imag = signed_right_Imag; /// Initial block (Only executed once when simulation start) initial begin</pre>
46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70	<pre>twoCompRedo twoC_leftReal(signpult(sign_mult_left_Real),.number(trunc_left_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_idptReal(signbit(sign_mult_left_Emag),.number(trunc_left_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_leftImag(.signbit(sign_mult_right_Real),.number(trunc_left_Imag),.sign_number(signed_right_Real)); twoCompRedo twoC_idptImag(.signbit(sign_mult_right_Imag),.number(trunc_left_Imag),.sign_number(signed_right_Imag)); twoCompRedo twoC_idptImag(.signbit(sign_mult_right_Imag),.number(trunc_right_Imag),.sign_number(signed_right_Imag)); //2'c operatof prior to multiplication. returns unsigned number. returns 17 bits twoComp_inst(.In(s),.Out(unsigned_s)); //extracts data portion a twoComp_inst(.In(s),.Out(unsigned_g)); //extracts data portion o twoComp_inst(.In(s),.Out(unsigned_j)); //extracts data portion o twoComp_inst(.In(s),.Out(unsigned_j)); //extracts data portion o twoComp_inst(.In(s)),.Out(unsigned_j)); //extracts data portion o data assign_sign_jb = b[1]? //extracts sign of a assign_sign_jb = b[1]? //extracts sign of c assign_sign_jd = signed_left_Imag = signed_right_Real; assign_sign_id = signed_left_Imag = signed_right_Imag; /// Initial block (Only executed once when simulation start) initial begin</pre>
46 47 48 49 50 51 52 53 55 56 57 58 59 60 61 62 63 64 65 66 67 68 99 70 71	<pre>twoCompRedo twoC_leftReal(signbit(sign_mult_left_Real),.number(trunc_left_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_idtReal(signbit(sign_mult_left_Imag),.number(trunc_left_Real),.sign_number(signed_left_Imag)); twoCompRedo twoC_leftImag(.signbit(sign_mult_left_Imag),.number(trunc_left_Imag),.sign_number(signed_right_Real)); twoCompRedo twoC_idtRimag(.signbit(sign_mult_right_Imag),.number(trunc_left_Imag),.sign_number(signed_right_Imag)); twoCompRedo twoC_idtRimag(.signbit(sign_mult_right_Imag),.number(trunc_left_Imag),.sign_number(signed_right_Imag)); twoComp insta(.In(s),.Out(unsigned_s)); //extracts data portion a twoComp insta(.In(s),.Out(unsigned_s)); //extracts data portion a twoComp insta(.In(s),.Out(unsigned_j)); //extracts data portion o twoComp insta(.In(s),.Out(unsigned_j)); //extracts data portion o twoComp insta(.In(s))Out(unsigned_j)); //extracts data portion o twoComp insta(.In(s))Out(unsigned_j); //extracts data portion o dassign_sign_s = c[17]; //extracts sign of s assign_sign_sign_sign_sign_sign_sign_sign</pre>
46 47 48 49 50 51 52 53 55 56 57 58 59 60 61 62 63 64 65 66 67 68 99 70 71	<pre>twoCompRedo twoC_leftReal(signpult(sign_mult_left_Real),.number(trunc_left_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_idthReal(signbit(sign_mult_left_Emag),.number(trunc_left_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_leftImag(.signbit(sign_mult_right_Real),.number(trunc_left_Imag),.sign_number(signed_right_Real)); twoCompRedo twoC_idthImag(.signbit(sign_mult_right_Imag),.number(trunc_left_Imag),.sign_number(signed_right_Imag)); twoCompRedo twoC_idthImag(.signbit(sign_mult_right_Imag),.number(trunc_right_Imag),.sign_number(signed_right_Imag)); //2'c operatof prior to multiplication. returns unsigned number. returns 17 bits twoComp_inst(.In(s),.Out(unsigned_s)); //extracts data portion a twoComp_inst(.In(s),.Out(unsigned_g)); //extracts data portion o twoComp_inst(.In(s),.Out(unsigned_j)); //extracts data portion o twoComp_inst(.In(s),.Out(unsigned_j)); //extracts data portion o twoComp_inst(.In(s)),.Out(unsigned_j)); //extracts data portion o data assign_sign_jb = b[1]? //extracts sign of a assign_sign_jb = b[1]? //extracts sign of c assign_sign_jd = signed_left_Imag = signed_right_Real; assign_sign_id = signed_left_Imag = signed_right_Imag; /// Initial block (Only executed once when simulation start) initial begin</pre>
46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 66 67 68 970 71 72 73	<pre>twoCompRedo twoC_lettReal(signpult(sign_mult_left_Real),.number(trunc_left_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_inthReal(signpult(sign_mult_left_Imag), number(trunc_left_Real),.sign_number(signed_left_Imag)); twoCompRedo twoC_lettImag(signbit(sign_mult_right_Real),.number(trunc_left_Imag),.sign_number(signed_left_Imag)); twoCompRedo twoC_inthImag(signbit(sign_mult_right_Imag), number(trunc_left_Imag),.sign_number(signed_left_Imag)); twoCompRedo twoC_orightImag(signbit(sign_mult_right_Imag), number(trunc_left_Imag),.sign_number(signed_right_Imag)); //2'c operatof prior to multiplication. returns unsigned number. returns 17 bits twoComp_insts(.In(s),.Out(unsigned_B)); //extracts data portion a twoComp_insts(.In(s),.Out(unsigned_g)); //extracts data portion o twoComp_insts(.In(s),.Out(unsigned_g)); //extracts data portion o twoComp_insts(.In(s), //extracts sign of a assign sign_a = ell7]; //extracts sign of s assign sign_b = jb[17]; //extracts sign of jb assign sign_b = jb[17]; //extracts sign of jd assign sign_d = clf1?] //extracts sign of jd assign Xgout_real = signed_left_Real - signed_right_Real; assign Xgout_real = signed_left_Tag + signed_right_Real; assign Xgout_real = signed_left_Tag + signed_right_Real; assign Xgout_imag = signed_left_Tag + signed_right_Tag; // Initial block (Only executed once when simulation start) initial begin end // Statements below will be executed on every clock rising edge</pre>
46 47 48 49 50 51 52 53 54 55 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74	<pre>twoCompRedo twoC_lettReal(signbit(sign_mult_ife(F_Real),.number(trunc_left_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_leftReal(signbit(sign_mult_ife), number(trunc_left_Imag),.sign_number(signed_left_Imag)); twoCompRedo twoC_leftImag(.signbit(sign_mult_ife), number(trunc_left_Imag),.sign_number(signed_left_Imag)); twoCompRedo twoC_leftImag(.signbit(sign_mult_ife), Imag), .number(trunc_left_Imag),.sign_number(signed_left_Imag)); twoCompRedo twoC_leftImag(.signbit(sign_mult_ife), Imag), .number(trunc_left_Imag),.sign_number(signed_right_Imag)); twoCompIntat(.In(s), Cout(unsigned)); //extracts data portion a twoComp Insta(.In(s), Cout(unsigned_jb)); //extracts data portion a twoComp Insta(.In(s), Cout(unsigned_jb)); //extracts data portion c twoComp Insta(.In(s), Cout(unsigned_jb)); //extracts data portion c twoComp Insta(.In(s)), Cout(unsigned_jb); //extracts data portion c twoComp Insta(.In(s)), Cout(unsigned_jb); //extracts data portion c twoComp Insta(.In(s)); //extracts sign of a assign sign_jb = bpl(1); //extracts sign of jb assign sign_jd = c(1)]; //extracts sign of jd assign sign_jd = c(1)]; //extracts sign of jd assign sign_jd = jd[1]; //extracts sign of jd assign sign_jd = signed_left_Real = signed_right_Real; assign Xqout_Imag = signed_left_Real = signed_right_Imag; /// Initial block (Colly executed once when simulation start) initial begin end // Statements below will be executed on every clock rising edge always@(posedge clk)</pre>
46 47 48 49 50 52 53 54 55 57 58 59 60 61 62 63 64 65 66 67 70 71 72 73 74 75	<pre>twoCompRedo twoC_leftReal(signpult(sign_mult_left_Real),.number(trunc_left_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_liftReal(signpult(sign_mult_left_Imag),.number(trunc_left_Real),.sign_number(signed_left_Imag)); twoCompRedo twoC_liftImag(signpult(sign_mult_left_Imag),.number(trunc_left_Imag),.sign_number(signed_left_Imag)); twoCompRedo twoC_liftImag(signpult(sign_mult_left_Imag),.number(trunc_left_Imag),.sign_number(signed_left_Imag)); twoCompRedo twoC_liftImag(signpult(sign_mult_left_Imag),.number(trunc_liftImag),.sign_number(signed_right_Imag)); //2'c operatof prior to multiplication. returns unsigned number, returns 17 bits twoComp_inst(.In(s),.Out(unsigned_B)); //extracts data portion a twoComp_inst(.In(s),.Out(unsigned_g)); //extracts data portion o twoComp_inst(.In(s),.Out(unsigned_g)); //extracts data portion o twoComp_inst(.In(s),.Out(unsigned_g)); //extracts data portion o twoComp_inst(.In(s)).Cut(unsigned_g)); //extracts data portion o twoComp_inst(.In(s)).Cut(unsigned_g)]; //extracts disc of jd assign Xigout_real = signed_left_Imag + signed_right_Imag; // Initial block (Only executed once when simulation start) initial begin end // Statements below will be executed on every clock rising edge always@(posedge clk) begin</pre>
46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 76	<pre>twoCompRedo twoC_lettReal(signbit(sign_mult_left_Real),.number(trunc_left_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_leftInsg(.signbit(sign_mult_left_Insg),.number(trunc_left_Insg),.sign_number(signed_left_Insg)); twoCompRedo twoC_leftInsg(.signbit(sign_mult_right_Real),.number(trunc_left_Insg),.sign_number(signed_left_Insg)); twoCompRedo twoC_leftInsg(.signbit(sign_mult_right_Insg),.number(trunc_left_Insg),.sign_number(signed_right_Insg)); twoCompRedo twoC_leftInsg(.signbit(sign_mult_right_Insg),.number(trunc_right_Insg),.sign_number(signed_right_Insg)); twoComp insta(.In(s),Cout(unsigned_s)); //extracts data portion a twoComp insta(.In(s),Cout(unsigned_s)); //extracts data portion o twoComp insta(.In(s),Cout(unsigned_g)); //extracts data portion o twoComp insta(.In(s),Cout(unsigned_g)); //extracts data portion o twoComp insta(.In(s)).Cout(unsigned_g)); //extracts data portion o twoComp insta(.In(s)).Cout(unsigned_g)); //extracts data portion o twoComp insta(.In(s)).(Nut(unsigned_g)); //extracts data portion o end // Statements below vill be executed on every clock rising</pre>
46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 66 66 66 66 66 66 70 71 72 73 74 77 77	<pre>twoCompRedo twoC_leftReal(signbit(sign_mult_left_Real),.number(trunc_left_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_iftReal(signbit(sign_mult_left_Imag),.number(trunc_left_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_liftImag(signbit(sign_mult_left_Imag),.number(trunc_left_Imag),.sign_number(signed_left_Imag)); twoCompRedo twoC_ightImag(signbit(sign_mult_left_Imag),.number(trunc_left_Imag),.sign_number(signed_left_Imag)); twoCompRedo twoC_uphtImag(signbit(sign_mult_left_Imag),.number(trunc_left_Imag),.sign_number(signed_right_Imag)); //2'c operatof prior to multiplication. returns unsigned number, returns 17 bits twoComp_inste(.In(s),.Out(unsigned_B)); //extracts data portion a twoComp_inste(.In(s),.Out(unsigned_g)); //extracts data portion o twoComp_inste(.In(s),.Out(unsigned_g)); //extracts data portion o twoComp_inste(.In(s),.Out(unsigned_g)); //extracts data portion o twoComp_inste(.In(s)).Cout(unsigned_g)); //extracts data portion o twoComp_inste(.In(s))./vextracts sign of jb assign_sign_a = a(17); //extracts sign of jb assign_sign_a = a(17); //extracts sign of jb assign_sign_d = signed_left_Real = signed_right_Real; assign Xgout_imag = signed_left_Imag + signed_right_Real; assign_Xgout_imag = signed_left_Imag + signed_right_Real; assign_Xgout_imag = signed_left_Real = signed_right_Real; assign_Xgout_imag = signed_left_Imag + signed_right_Imag; // Initial block (Only executed once when simulation start) initial begin end // Statements below vill be executed on every clock rising edge always8(posedge clk) begin</pre>
46 47 48 49 51 52 53 55 55 55 55 55 55 55 55 55 55 55 55	<pre>twoCompRedo twoC_lettReal(signbit(sign_mult_left_Real),.number(trunc_left_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_leftReal(signbit(sign_mult_left_Imag),.number(trunc_left_Imag),.sign_number(signed_left_Imag)); twoCompRedo twoC_leftImag(.signbit(sign_mult_left_Imag),.number(trunc_left_Imag),.sign_number(signed_left_Imag)); twoCompRedo twoC_leftImag(.signbit(sign_mult_right_Imag),.number(trunc_left_Imag),.sign_number(signed_left_Imag)); twoCompRedo twoC_leftImag(.signbit(sign_mult_right_Imag),.number(trunc_left_Imag),.sign_number(signed_right_Imag)); twoComp intat(.In(s),Cout(unaigned_s)); //extracts data portion a twoComp intat(.In(s),Cout(unaigned_s)); //extracts data portion o twoComp intat(.In(s),Cout(unaigned_s)); //extracts data portion o twoComp intat(.In(s),Cout(unaigned_s)); //extracts data portion o twoComp intat(.In(s)),.Cout(unaigned_s)); //extracts data portion o twoComp intat(.In(s)),.Cout(unaigned_s); //extracts data portion o isasign_sign_s = cliN; //extracts sign of c assign sign_s = cliN; //extracts sign of jd assign xqout_rimag = signed_left_Imag + signed_right_Imag; // Initial block (Conly executed once when simulation start) initial begin end // Statements below will be executed on every clock rising edge always8(posedge clk) begin sign_mult_left_Real &lt;= sign_s isgn_jd; sign_mult_left_Real &lt;= sign_s isgn_jd;</pre>
46 47 48 50 51 52 53 55 55 55 55 55 55 55 55 55 55 55 55	<pre>twoCompRedo twoC_leftReal(signbit(sign_mult_left_Real),.number(trunc_left_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_rightReal(signbit(sign_mult_left_Imag),.number(trunc_left_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_rightRead(.signbit(sign_mult_left_Imag),.number(trunc_left_Imag),.sign_number(signed_left_Imag)); twoCompRedo twoC_rightRead(.signbit(sign_mult_left_Imag),.number(trunc_left_Imag),.sign_number(signed_left_Imag)); twoCompRedo twoC_rightRead(.signbit(sign_mult_left_Imag),.number(trunc_left_Imag),.sign_number(signed_right_Imag)); //2/c operatof prior to multiplication. returns unsigned number. returns 17 bits twoComp inst(.in(s),.Out(unsigned_B)); //extracts data portion a twoComp inst(.in(s),.Out(unsigned_G)); //extracts data portion o twoComp inst(.in(s),.Out(unsigned_g); assign_sign_g = signed_left_Enal = signed_right_Enag; // Initial block (Only executed once when simulation start) initial begin end // Statements below vill be executed on every clock rising edge always8(posedge clk) begin sign_mult_left_Enag (&lt; sign_g) a 'sign_jdi sign_mult_left_Enag (&lt;</pre>
46 47 48 50 51 52 53 55 56 57 58 59 60 61 263 64 65 667 68 69 70 71 72 73 74 57 76 77 78 9 80	<pre>twoCompRedo twoC_leftReal(signbit(sign_mult_left_Real),.number(trunc_left_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_leftInsg(.signbit(sign_mult_left_Insg),.number(trunc_left_Insg),.sign_number(signed_left_Insg)); twoCompRedo twoC_leftInsg(.signbit(sign_mult_left_Insg),.number(trunc_left_Insg),.sign_number(signed_left_Insg)); twoCompRedo twoC_leftInsg(.signbit(sign_mult_right_Insg),.number(trunc_left_Insg),.sign_number(signed_right_Insg)); twoCompRedo twoC_leftInsg(.signbit(sign_mult_right_Insg),.number(trunc_left_Insg),.sign_number(signed_right_Insg)); twoComp insts(.In(s),.Out(unsigned_s)); //extracts data portion a twoComp insts(.In(s),.Out(unsigned_s)); //extracts data portion o twoComp insts(.In(s),.Out(unsigned_j)); //extracts data portion o twoComp insts(.In(s)).//extracts sign of a assign sign_s = a[17]; //extracts sign of f assign sign_s = c[17]; //extracts sign of jb assign sign_s = c[17]; //extracts sign of jd assign sign_s = c[17]; //extracts sign of jd assign sign_s = signed_left_Enal - signed_right_Real; assign Xqout_real = signed_left_Insg + signed_right_Insg; // Initial block (Only executed once when simulation start) initial</pre>
46 47 48 50 51 52 53 55 56 57 58 59 60 61 263 64 65 667 68 69 70 71 72 73 74 57 76 77 78 9 80	<pre>twoCompRedo twoC_leftReal(signbit(sign_mult_left_Real),.number(trunc_left_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_rightReal(signbit(sign_mult_left_Imag),.number(trunc_left_Real),.sign_number(signed_left_Real)); twoCompRedo twoC_rightRead(.signbit(sign_mult_left_Imag),.number(trunc_left_Imag),.sign_number(signed_left_Imag)); twoCompRedo twoC_rightRead(.signbit(sign_mult_left_Imag),.number(trunc_left_Imag),.sign_number(signed_left_Imag)); twoCompRedo twoC_rightRead(.signbit(sign_mult_left_Imag),.number(trunc_left_Imag),.sign_number(signed_right_Imag)); //2/c operatof prior to multiplication. returns unsigned number. returns 17 bits twoComp inst(.in(s),.Out(unsigned_B)); //extracts data portion a twoComp inst(.in(s),.Out(unsigned_G)); //extracts data portion o twoComp inst(.in(s),.Out(unsigned_g); assign_sign_g = signed_left_Enal = signed_right_Enag; // Initial block (Only executed once when simulation start) initial begin end // Statements below vill be executed on every clock rising edge always8(posedge clk) begin sign_mult_left_Enag (&lt; sign_g) a 'sign_jdi sign_mult_left_Enag (&lt;</pre>

1	`timescale lns / lps
2	
3	// Company:
4	// Engineer:
5	//
6	// Create Date: 03/25/2017 11:43:54 AM
7	// Design Name: Complex Multiplication
8	// Module Name: mult18x18
9	// Project Name:
10	// Target Devices:
11	// Tool Versions:
12	// Description:
13	//
	// Dependencies:
	//
	// Revision:
	// Revision 0.01 - File Created
18	// Additional Comments:
19	//
20	
21	
22	
23 0	module multisti8( interest.
29	input wire Cik, in the life) a, //a
26	input wire [16:0] sb, //sb
27	input wife [16:0] ju, //ju
28	input wire [16:0] jd, //jd
29	august wire [16:0] Gut left Real, //(a*c)
30	output wire [16:0] Out right Real, //(b/d)
31	output wire [16:0] Out left Imag, //(a <sup>1</sup> jd)
32	output wire [16:0] Out right mag; //(c/jb)
33	cather are frach and rain and he has
34	//Complex Multiplier
35	// Real Imag
36	$//(a+jb)(c+jd) = (a^{\pm}c - b^{\pm}d) + j(b^{\pm}c + a^{\pm}d)$
37	(* dont_touch = "true" *)reg [34:0] mult_left_Real;
38	(* dont_touch = "true" *)reg [34:0] mult_right_Real;
39	(* dont_touch = "true" *)reg [34:0] mult_left_Imag;
40	(* dont_touch = "true" *)reg [34:0] mult_right_Imag;
41	
42	assign Out_left_Real[16:0] = mult_left_Real[32:16]; // (a*c)
43	assign Out_right_Real[16:0] = mult_right_Real[32:16]; // (jb*jd)
44	assign Out_left_Imag[16:0] = mult_left_Imag[32:16]; // (a'jd)
45	assign Out_right_Imag[16:0] = mult_right_Imag[32:16]; // (jb*c)
46	)initial begin
48	
40 6	end
50 E	always8(posedge clk)
<b>51</b> E	
52	//Calculate real portion
53	mult left Real <= a * c;
54	mult right Real <= jb * jd;
55	
56	//Calculate imag portion
57	<pre>mult_left_Imag &lt;= a * jd;</pre>
58	<pre>mult_right_Imag &lt;= jb * c;</pre>
<b>59</b> 🤅	end
60	
<b>61</b> (	endmodule

	1 ** 00
1	`timescale lns / lps
2	
3	// Company:
4	// Engineer:
5	//
6	// Create Date: 05/23/2017 08:08:50 AM
7	// Design Name:
8	// Module Name: twoCompRedo
9	// Project Name:
10	// Target Devices:
11	// Tool Versions:
12	// Description:
13	1/
14	// Dependencies:
	//
	// Revision:
17	// Revision 0.01 - File Created
	// Additional Comments:
	//
20	1
21	
22	
	module twoCompRedo(
	input wire signbit,
	input wire [16:0] number,
26	output wire [17:0] sign_number
27	);
28	
29	assign sign_number = signbit?{signbit,~number+1}:{signbit,number};
30 (	endmodule

\*TWO

/ timescale lns / lps
// Company:
// Design Rame:
// Design Rame:
// Design Rame:
// Project Numme:
// Project Project Numme

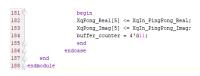
1	`timescale lns / lps
2	
3	// Company:
4	// Engineer:
5	11
6	// Create Date: 04/08/2017 09:41:42 AM
7	// Design Name: Complex Addition / Subtraction
8	// Module Name: addsub18x18
9	// Project Name:
10	// Target Devices:
11	// Tool Versions:
12	// Description:
13	1/
14	// Dependencies:
15	1/
16	// Revision:
17	// Revision 0.01 - File Created
18	// Additional Comments:
19	17
20	
21	
22	
	module addsub18x18(
24	input wire [17:0] TopIn_real, // a
25	input wire [17:0] TopIn_imag, // jb
26	input wire [17:0] BottomIn_real, // c
27	input wire [17:0] BottomIn_imag, // jd
28	input wire clk,
29	input wire counterw,
30	output wire [17:0] Xgout_real, // a + b or a - b
31	output wire [17:0] Xqout_imag ); // j(b + d) or j(b - d)
32	
33	(* dont_touch = "true" *) reg [17:0] out_real;
34	(* dont_touch = "true" *) reg [17:0] out_imag;
35	
36	//Complex Addition/Subtraction
37	// Real Imag
38	//(a+jb)+(c+jd) = a + c , $j(b+d)$
39	//(a+jb) - (c+jd) = a - c , $j(b-d)$
40	assign Xgout_real = out_real; //Real Component
41	assign Xqout_imag = out_imag; //Imag Component
42	
	always@(posedge clk)
<b>44</b> E	
45	out_real <= (counterw==1'b1) ? TopIn_real + BottomIn_real : TopIn_real - BottomIn_real; //Real Component
46	<pre>out_imag &lt;= (counterw==1'bl) ? TopIn_imag + BottomIn_imag : TopIn_imag - BottomIn_imag; //Imag Component</pre>
47 6	
48 6	endmodule

1	`timescale lns / lps
2 3	
3 4	//////////////////////////////////////
5	// Engineer:
6	// // Create Date: 04/19/2017 04:41:54 FM
8	// Lesign back 69/15/20/ 94/41/34 FM
9	// Module Name: pingpong
10 11	// Project Name: // Target Devices:
	/ Tol Vesions:
	// Description:
14 15	// // Dependencies:
16	//
	// Revision: // Revision 0.01 - File Created
18	// Wetulion 0.01 - File Created // Additional Comments:
20	//
21 22	
23	// for N=8
24 25	//k = 0, N/2, 1, N/2+1, 2, N/2+2, 3, N/2+3;;
25	//k = (0, 4, 1, 5, 2, 6, 3, 7)
27	
	//Ping//Fong/ //One Register is being utilized for real component
30	// 0 // 0 // //One Register is being utilized for imag component ////////////////i.e. Fing has a real and imaginery register
	// 1 // 1 // //Pong has a real and imaginery register
32 33	//////////////////////////////////////
34	$/////////////////////x[000] = x[000] \text{ or } x[0] \Rightarrow x[0]$
35	// 3 // //X[001] = x[100] or X[1] => x[4] //////////////X[010] = x[001] or X[2] => x[1]
36 37	///////// //K[01] = x[001] or X[2] => x[1] //4 //4 // //K[011] = x[101] or X[3] => x[5]
38	///////////////X[100] = x[010] or X[4] => x[2]
	// 5 // 5 // //X[101] = x[110] or X[5] => x[6] ///////////////X[110] = x[011] or X[6] => x[3]
41	//6 //6 // //X[111] = x[111]  or  X[7] => x[7]
43 44	
45	
46 47	// complex input = (x + jz) // complex output = (x + jz)
48	
	module pingpongK(
50 51	input wire [17:0] XqIn_PingPong_Real, // input Xq real component input wire [17:0] XqIn_PingPong_Imag, // input Xq imag component
52	input wire clk, // input clock
53 54	input wire [3:0] counter, // input counter output wire [17:0] XqOut_PingPongw_Real, // output Xq real component on a wire
55	output wire [17:0] XqOut_PingPongw_Imag // output Xq imag component on a wire
56 57	);
58	(* dont_touch = "true" *) reg [17:0] XqBing_Real [7:0]; // register for Ping real component
59	(* dont_touch = "true" *) reg [17:0] XgPing_Imag [7:0]; // register for Ping imag component
60 61	(* dont_touch = "true" *) reg [17:0] XqPong_Real [7:0]; // register for Pong real component (* dont_touch = "true" *) reg [17:0] XqPong_Imag [7:0]; // register for Pong imag component
62	(* dont_touch = "true" *) reg [3:0] buffer_counter;
63 64	// (* dont_touch = "true" *) vire [3:0] select;
65	
	<pre>assign XqOut_PingPongw_Real = (buffer_counter[3]==1'bl) ? XqPing_Real[buffer_counter[2:0]] : XqPong_Real[buffer_counter[2:0]]; //XqOut_PingPong_Real; // register for output Xq rea assign XqOut_PingPongw_Imag = (buffer_counter[3]==1'bl) ? XqPing_Imag[buffer_counter[2:0]] : XqPong_Imag[buffer_counter[2:0]]; //XqOut_PingPong_Imag; // register for output Xq ima </pre>
68	//assign select = buffer_counter + 1;
	//alvays statement that occurs at every rising //edge of the clock
70	//eage of the clock
	initial
73 Ċ	begin
75 6	end
76	alvavs8(cosedge clk)
78	
79	//The Pong buffer is outputed when the Ping buffer is
80 81	//being filled up. Fing and Fong then alternate //The MSB of counter is utilized as
82	//my switch bit. Buffers are complex numbers.
83	// Temph code to Ding and Energine buffaue
84 85	// Input code to Ping and Pong buffers // based on k = 0, N/2, 1, N/2+1, 2, N/2+2, 3, N/2+3;
86	// k = {0, 4, 1, 5, 2, 6, 3, 7};
87 88	// Results in 16 cycle period.
89 🗄	case (counter)
90 🕴	31b0000: //Xq(0)

XqPong\_Real[2] <= XqIn\_PingPong\_Real; XqPong\_Imag[2] <= XqIn\_PingPong\_Imag; buffer\_counter = 4\*d12; end buffer\_counter = < \_\_\_\_\_end end 3'b0010: //Xq(2) begin XqFoom\_Real[3] <= XqIn\_PingFong\_Real; XqFoom\_Imag[3] <= XqIn\_PingFong\_Imag; buffer\_counter = 4'd14; end buffer\_counter = 4\*d14; end 3\*bo01: //Xq(3) begin XqPong\_Heal(7) << XqIn\_PingPong\_Real; XqPong\_Heal(7) << XqIn\_PingPong\_Real; yaqPong\_Hang(7) << XqIn\_PingPong\_Imag; buffer\_counter = 4\*d15; end bullfer\_counct . .... end 3'b0100; //Xq(4) begin XqPing\_Real[0] <= XqIn\_PingPong\_Real; XqPing\_Imag[0] <= XqIn\_PingPong\_Imag; buffer\_counter = 4'd0; end end 3'b0101: //Xq(5) begin begin Xq2ing\_Real(4) <= XqIn\_PingPong\_Real; Xq2ing\_Imag(4) <= XqIn\_PingPong\_Imag; buffer\_counter = 4\*d1; end 3\*b0110; //Xq(6) begin begin
XqPing\_Real[1] <= XqIn\_PingPong\_Real;
XqPing\_Imag[1] <= XqIn\_PingPong\_Imag;
buffer\_counter = 4\*d;
end
3\*b0111: //Xq(7)
begin
XvPine Bax[5] <= VaTa\_DispRes\_Real;
</pre> begin XqFing\_Real(5) <= XqIn\_PingPong\_Real; XqPing\_Imag(5) <= XqIn\_PingPong\_Imag; baffer\_counter = 4\*d; end 4\*b1000: //Xq(3) begin XqPing\_Real[2] <= XqIn\_PingPong\_Real; XqPing\_Imag(2) <= XqIn\_PingPong\_Imag; baffer\_counter = 4\*d4; end buffer\_counter = 4\*dd; end 4\*biO01: //Xq(9) begin XqPing\_Real(6) <= XqIn\_PingPong\_Real; XqPing\_Imag(6) <= XqIn\_PingPong\_Imag; buffer\_counter = 4\*dd; end end 4'b1010: //Xg(10) 1010: //Aq(10) begin XqPing\_Real(3) <= XqIn\_PingPong\_Real; XqPing\_Imag(3) <= XqIn\_PingPong\_Imag; buffer\_counter = 4'd6; ---4 putrer\_counter = 4'do; end 4'bi0l1: //Xq(1) begin XqPing\_Real[7] <= XqIn\_PingPong\_Real; XqPing\_Imag(7) <= XqIn\_PingPong\_Imag; buffer\_counter = 4'd7; end end 4'b1100: //Xq(12) begin begin XqPong\_Real[0] <= XqIn\_PingPong\_Real; XqPong\_Imag[0] <= XqIn\_PingPong\_Imag; buffer\_counter = 4\*d0; end end 4'b1101: //Xq(13) begin XqPong\_Real(4) <= XqIn\_PingPong\_Real; XqPong\_Imag(4) <= XqIn\_PingPong\_Imag; buffer\_counter = 4'd0; end 4'bill0: //Xq(14) begin XqPong\_Real(1) < " - "</pre> pegin
XqPong\_Real[1] <= XqIn\_PingPong\_Real;
XqPong\_Imag[1] <= XqIn\_PingPong\_Imag;
buffer\_counter = 4'dl0;
end</pre>

 $\begin{array}{c} 151 \\ 152 \\ 153 \\ 154 \\ 155 \\ 156 \\ 157 \\ 158 \\ 159 \\ 160 \\ 161 \\ 162 \\ 163 \\ 164 \\ 165 \\ 166 \\ 167 \\ 168 \\ 167 \\ 170 \\ 171 \\ 172 \\ 173 \\ 174 \\ 175 \\ 177 \\ 178 \\ 179 \\ 179 \\ 179 \\ 179 \\ 179 \\ 170 \\ 179 \\ 170 \\ 170 \\ 177 \\ 178 \\ 179 \\ 180 \\$ 

end 4'b1111: //Xq(15)



THIS PAGE INTENTIONALLY LEFT BLANK

## **APPENDIX C. HARDWARE CONSTRAINTS FILE**

#### 

1 f# This file is a general .xdc for the Nexys4 DDR Rev. C 2 f# To use it in a project: 3 f# - uncomment the lines corresponding to used pins 4 f# - rename the used ports (in each line, after get\_ports) according to the top level signal names in the project 5 6 ## Clock signal 7 set\_property -dict (PACKAGE\_PIN E3 IOSTANDARD LVCMOS33) [get\_ports CLK100MHZ]
8 create\_clock -add -name sys\_clk\_pin -period 10.00 -waveform [0 5] [get\_ports (CLK100MHZ)]; 11 ##Switches 11 ##olicaes 12 14 set\_property -dict ( PACKAGE PIN J15 IOSTANDARD LVCMOS33 ) [get\_ports ( SW[0) ]); #10\_L44N\_T3\_R50\_15 Sch=sv[0] 14 set\_property -dict ( PACKAGE PIN L16 IOSTANDARD LVCMOS33 ) [get\_ports ( SW[1] )]; #10\_L5N\_T0\_D02\_RMCLK\_14 Sch=sv[1] 15 set\_property -dict ( PACKAGE PIN N3 IOSTANDARD LVCMOS33 ) [get\_ports ( SW[1] )]; #10\_L5N\_T0\_D02\_RMCLK\_14 Sch=sv[2] 16 set\_property -dict ( PACKAGE PIN N3 IOSTANDARD LVCMOS33 ) [get\_ports ( SW[4] )]; #10\_L1N\_T1\_MCC\_14 Sch=sv[3] 16 set\_property -dict ( PACKAGE PIN N3 IOSTANDARD LVCMOS33 ) [get\_ports ( SW[4] )]; #10\_L1N\_T1\_MCC\_14 Sch=sv[4] 18 set\_property -dict ( PACKAGE PIN N3 IOSTANDARD LVCMOS33 ) [get\_ports ( SW[4] )]; #10\_L1N\_T1\_MCC\_14 Sch=sv[4] 18 set\_property -dict ( PACKAGE PIN N3 IOSTANDARD LVCMOS33 ) [get\_ports ( SW[4] )]; #10\_L1N\_T1\_MCC\_14 Sch=sv[4] 10 set\_property -dict ( PACKAGE PIN N3 IOSTANDARD LVCMOS33 ) [get\_ports ( SW[4] )]; #10\_L1N\_T1\_MCC\_14 Sch=sv[4] 21 set\_property -dict ( PACKAGE PIN N3 IOSTANDARD LVCMOS33 ) [get\_ports ( SW[4] )]; #10\_L5N\_T0\_D0\_14 Sch=sv[6] 21 set\_property -dict ( PACKAGE PIN N3 IOSTANDARD LVCMOS33 ) [get\_ports ( SW[1] )]; #10\_L5N\_T0\_D0\_14 Sch=sv[6] 23 set\_property -dict ( PACKAGE PIN N3 IOSTANDARD LVCMOS13 ) [get\_ports ( SW[1] )]; #10\_L5N\_T0\_D0\_2 MS\_2N\_R B\_14 Sch=sv[1] 24 set\_property -dict ( PACKAGE PIN N3 IOSTANDARD LVCMOS33 ) [get\_ports ( SW[1] )]; #10\_L5N\_T0\_D0\_2 MS\_2N\_R B\_14 Sch=sv[1] 24 set\_property -dict ( PACKAGE PIN N3 IOSTANDARD LVCMOS33 ) [get\_ports ( SW[1] )]; #10\_L1SN\_T1\_A0\_L1S\_14 Sch=sv[1] 25 set\_property -dict ( PACKAGE PIN N4 IOSTANDARD LVCMOS33 ) [get\_ports ( SW[1] )]; #10\_L1SN\_T1\_A0E\_D1\_14 Sch=sv[1] 26 set\_property -dict ( PACKAGE PIN N5 IOSTANDARD LVCMOS33 ) [get\_ports ( SW[1] )]; #10\_L1SN\_T1\_A0E\_D1\_14 Sch=sv[1] 27 set\_property -dict ( PACKAGE PIN N1 IOSTANDARD LVCMOS33 ) [get\_ports ( SW[1] )]; #10\_L1SN\_T1\_A0E\_D2\_14 Sch=sv[1] 28 set\_property -dict ( PACKAGE PIN N1 IOSTANDARD LVCMOS33 ) [get\_ports ( SW[1] )]; #10\_L1SN\_T1\_A0E\_D2\_14 Sch=sv[1] 29 set\_property -dict ( PACKAGE PIN N6 IOSTANDARD LVCMOS33 ) [get\_ 30 31 ## LEDs 31 # EAGS 32 33 # dest\_property -dict ( PACKAGE\_PIN H17 IOSTANDARD LVCMOS33 ) [get\_ports ( LED[0] )]; #IO\_LISP\_T2\_A24\_IS Sch-led[0] 34 # dest\_property -dict ( PACKAGE\_PIN H17 IOSTANDARD LVCMOS33 ) [get\_ports ( LED[1] )]; #IO\_LISP\_T2\_A24\_IS Sch-led[1] 35 # dest\_property -dict ( PACKAGE\_PIN H17 IOSTANDARD LVCMOS33 ) [get\_ports ( LED[1] )]; #IO\_LISP\_T2\_A15\_IS Sch-led[2] 36 # dest\_property -dict ( PACKAGE\_PIN H18 IOSTANDARD LVCMOS33 ) [get\_ports ( LED[2] )]; #IO\_LISP\_T2\_A14\_D2\_I4 Sch-led[3] 37 # des\_property -dict ( PACKAGE\_PIN H18 IOSTANDARD LVCMOS33 ) [get\_ports ( LED[4] )]; #IO\_LISP\_T2\_A14\_D2\_I4 Sch-led[4] 38 # dest\_property -dict ( PACKAGE\_PIN H18 IOSTANDARD LVCMOS33 ) [get\_ports ( LED[4] )]; #IO\_LISP\_T2\_A14\_D2\_I4 Sch-led[5] 39 # dest\_property -dict ( PACKAGE\_PIN H18 IOSTANDARD LVCMOS33 ) [get\_ports ( LED[6] )]; #IO\_LISP\_T2\_A14\_D2\_I4 Sch-led[6] 39 # dest\_property -dict ( PACKAGE\_PIN H18 IOSTANDARD LVCMOS33 ) [get\_ports ( LED[6] )]; #IO\_LISP\_T2\_A14\_D2\_I4 Sch-led[6] 40 # dest\_property -dict ( PACKAGE\_PIN H16 IOSTANDARD LVCMOS33 ) [get\_ports ( LED[6] )]; #IO\_LIAP\_T2\_A14\_D2\_I4 Sch-led[6] 41 # dest\_property -dict ( PACKAGE\_PIN H16 IOSTANDARD LVCMOS33 ) [get\_ports ( LED[6] )]; #IO\_LIAP\_T2\_A14\_D2\_I4 Sch-led[1] 42 # dest\_property -dict ( PACKAGE\_PIN H16 IOSTANDARD LVCMOS33 ) [get\_ports ( LED[1] )]; #IO\_LIAP\_T2\_A14\_D2\_LIA Sch-led[10] 44 # dest\_property -dict ( PACKAGE\_PIN H16 IOSTANDARD LVCMOS33 ) [get\_ports ( LED[11] )]; #IO\_LIAP\_T2\_DSCL 4 Sch-led[10] 45 # dest\_property -dict ( PACKAGE\_PIN H16 IOSTANDARD LVCMOS33 ) [get\_ports ( LED[11] )]; #IO\_LIAP\_T2\_DSCL 4 Sch-led[10] 45 # dest\_property -dict ( PACKAGE\_PIN H16 IOSTANDARD LVCMOS33 ) [get\_ports ( LED[11] )]; #IO\_LIAP\_T2\_DSCL 4 Sch-led[10] 45 # dest\_property -dict ( PACKAGE\_PIN H16 IOSTANDARD LVCMOS33 ) [get\_ports ( LED[11] )]; #IO\_LISP\_T2\_DSCL 4 Sch-led[10] 45 # dest\_property -dict ( PACKAGE\_PIN H16 IOSTANDARD LVCMOS33 ) [get\_ports ( LED[11] )]; #IO\_LISP\_T2\_DSCL 4 Sch-led[11] 45 # dest\_property -dict ( PACKAGE\_PIN H16 IOSTANDARD LVCMOS33 ) [get\_p 49 69 #set\_property -dict ( PACKAGE\_PIN RL2 IOSTANDARD LVCMOS33 ) [get\_ports ( LEDL6 B )]; #IO\_LSP\_TO\_DO6\_14 Sch=ledl6 b 51 #set\_property -dict ( PACKAGE\_PIN ML6 IOSTANDARD LVCMOS33 ) [get\_ports ( LEDL6 G ]]; #IO\_LLSP\_TI\_DI6\_14 Sch=ledl6 g 52 #set\_property -dict ( PACKAGE\_PIN GL4 IOSTANDARD LVCMOS33 ) [get\_ports ( LEDL7 B ]]; #IO\_LLSP\_TI\_SOC\_14 Sch=ledl6 g 53 #set\_property -dict ( PACKAGE\_PIN GL4 IOSTANDARD LVCMOS33 ) [get\_ports ( LEDL7 B ]]; #IO\_LLSP\_TI\_SOC\_14 Sch=ledl7 g 54 #set\_property -dict ( PACKAGE\_PIN GL4 IOSTANDARD LVCMOS33 ) [get\_ports ( LEDL7 B ]]; #IO\_LLSP\_TI\_SOC\_14 Sch=ledl7 g 54 #set\_property -dict ( PACKAGE\_PIN NL6 IOSTANDARD LVCMOS33 ) [get\_ports ( LEDL7 G ]]; #IO\_LLN7 I\_SNC\_14 Sch=ledl7 g 55 #set\_property -dict ( PACKAGE\_PIN NL6 IOSTANDARD LVCMOS33 ) [get\_ports ( LEDL7 B ]]; #IO\_LLN7 I\_SNC\_14 Sch=ledl7 g 55 #set\_property -dict ( PACKAGE\_PIN NL6 IOSTANDARD LVCMOS33 ) [get\_ports ( LEDL7 B ]]; #IO\_LLN7 I\_SNC\_14 Sch=ledl7 g 55 #set\_property -dict ( PACKAGE\_PIN NL6 IOSTANDARD LVCMOS33 ) [get\_ports ( LEDL7 B ]]; #IO\_LLN7 I\_SNC\_14 Sch=ledl7 g 55 #set\_property -dict ( PACKAGE\_PIN NL6 IOSTANDARD LVCMOS33 ) [get\_ports ( LEDL7 B ]]; #IO\_LLN7 I\_SNC\_14 Sch=ledl7 g 55 #set\_property -dict ( PACKAGE\_PIN NL6 IOSTANDARD LVCMOS33 ) [get\_ports ( LEDL7 B ]]; #IO\_LLN7 I\_SNC\_14 Sch=ledl7 g 55 #set\_property -dict ( PACKAGE\_PIN NL6 IOSTANDARD LVCMOS33 ) [get\_ports ( LEDL7 B ]]; #IO\_LLN7 I\_SNC\_14 Sch=ledl7 g 55 #set\_property -dict ( PACKAGE\_PIN NL6 IOSTANDARD LVCMOS33 ) [get\_ports ( LEDL7 B ]]; #IO\_LLN7 I\_SNC\_14 Sch=ledl7 g 55 #set\_property -dict ( PACKAGE\_PIN NL6 IOSTANDARD LVCMOS33 ) [get\_ports ( LEDL7 B ]]; #IO\_LLN7 I\_SNC\_14 Sch=ledl7 g 55 #set\_property -dict ( PACKAGE\_PIN NL6 IOSTANDARD LVCMOS33 ) [get\_ports ( LEDL7 B ]]; #IO\_LN7 I\_SNC\_14 Sch=ledl7 g 55 #set\_property -dict ( PACKAGE\_PIN NL6 IOSTANDARD LVCMOS33 ) [get\_ports ( LEDL7 B ]]; #IO\_LN7 I\_SNC\_14 Sch=ledl7 B ]]; #IO\_LN7 I\_SNC\_14 Sch=ledl7 B ]]; #IO\_LN7 I\_SNC\_14 Sch=ledl7 B ]]]; #IO\_LN7 I\_SNC\_14 Sch=ledl7 B ]]]; #IO\_LN7 I\_SNC\_14 Sch=ledl7 B ]]]]]]]]]]]]]]]] 57 58 #77 segment display 59 60 #set\_property -dict ( PACKAGE\_PIN TIO IOSTANDARD LVCMOS33 ) [get\_ports ( CA )]; #IO\_L24N\_T3\_A00\_D16\_14 Sch=ca 60 #set\_property -dict ( PACKAGE\_PIN TIO IOSTANDARD LVCMOS33 ) [get\_ports ( CA )]; #IO\_25 14 Sch=cb 60 #see\_property -dict ( PACHAGE PIN 110 IOSTANDARD LVCM0533 ) [get\_ports ( Ca )]; #10\_118 PI\_1 = 000\_116\_1 \* Sci 61 #see\_property -dict ( PACHAGE PIN 110 IOSTANDARD LVCM0533 ) [get\_ports ( CC )]; #10\_2514 Sch=cb 62 #see\_property -dict ( PACHAGE PIN 116 IOSTANDARD LVCM0533 ) [get\_ports ( CC )]; #10\_118 PI\_1 #AC 14 Sch=cb 64 #see\_property -dict ( PACHAGE PIN 115 IOSTANDARD LVCM0533 ) [get\_ports ( CC )]; #10\_118 PI\_1 #AC 14 Sch=cb 64 #see\_property -dict ( PACHAGE PIN 115 IOSTANDARD LVCM0533 ) [get\_ports ( CC )]; #10\_118 PI\_1 #AC 14 Sch=cb 66 #see\_property -dict ( PACHAGE PIN 115 IOSTANDARD LVCM0533 ) [get\_ports ( CG )]; #10\_118 PI\_1 #AC 14 Sch=cb 66 #see\_property -dict ( PACHAGE PIN 115 IOSTANDARD LVCM0533 ) [get\_ports ( CG )]; #10\_14PI\_10 De1( 4 Sch=cc) 66 #see\_property -dict ( PACHAGE PIN 115 IOSTANDARD LVCM0533 ) [get\_ports ( CG )]; #10\_14PI\_10 De1( 4 Sch=cc) 66 #see\_property -dict ( PACHAGE PIN 115 IOSTANDARD LVCM0533 ) [get\_ports ( CG )]; #10\_14PI\_10 L4P\_10 Do4( 4 Sch=cc) 66 #see\_property -dict ( PACHAGE PIN 115 IOSTANDARD LVCM0533 ) [get\_ports ( CG )]; #10\_14PI\_10 L4P\_10 Do4( 4 Sch=cc) 66 #see\_property -dict ( PACHAGE PIN 115 IOSTANDARD LVCM0533 ) [get\_ports ( CG )]; #10\_14PI\_10 L4P\_10 Do4( 4 Sch=cc) 66 #see\_property -dict ( PACHAGE PIN 115 IOSTANDARD LVCM053 ) [get\_ports ( CG )]; #10\_14PI\_10 L4P\_10 Do4( 4 Sch=cc) 66 #see\_property -dict ( PACHAGE PIN 115 IOSTANDARD LVCM053 ) [get\_ports ( CG )]; #10\_14PI\_10 L4P\_10 Do4( 4 Sch=cc) 66 #see\_property -dict ( PACHAGE PIN 115 IOSTANDARD LVCM053 ) [get\_ports ( CG )]; #10\_14PI\_10 L4P\_10 Do4( 4 Sch=cc) 66 #see\_property -dict ( PACHAGE PIN 115 IOSTANDARD LVCM053 ) [get\_ports ( CG )]; #10\_14PI\_10 L4PI\_10 Sch=cd Sch 66 #see\_property -dict ( PACHAGE PIN 115 IOSTANDARD LVCM053 ) [get\_ports ( CG )]; #10\_14PI\_10 L4PI\_10 Sch=cd Sch 66 #see\_property -dict ( PACHAGE PIN 115 IOSTANDARD LVCM053 ) [get\_ports ( CG )]; #10\_14PI\_10 Sch 4 Sch=cd Sch 66 #see\_property -dict ( PACHAGE PIN 115 IOSTANDARD LVCM053 ) [get\_ports ( CG )]; #10\_14PI\_10 Sch 4 Sch=cd Sch 66 #see\_property -dic 68 #set property -dict ( PACKAGE PIN H15 IOSTANDARD LVCMOS33 ) [get ports ( DP )]; #IO L19N T3 A21 VREF 15 Sch-dp 69 70 #set\_property -dict ( PACKAGE\_PIN J17 IOSTANDARD LVCMOS33 ) [get\_ports ( AN[0] )]; #IO\_L23P\_T3\_FOE\_B\_15 Sch=an[0] 70 /set\_property -dict / PACKAGE\_PIN J17 71 /set\_property -dict / PACKAGE\_PIN J18 72 /setproperty -dict / PACKAGE\_PIN J14 74 /setproperty -dict / PACKAGE\_PIN J14 74 /setproperty -dict / PACKAGE\_PIN T14 76 /set\_property -dict / PACKAGE\_PIN T14 76 /setproperty -dict / PACKAGE\_PIN T14 IOSTMUNARD LVCMOS33 ) [ge\_ports ( AN[0] )]; 410\_L325\_73\_702 m\_15\_Schman[0] IOSTMUNARD LVCMOS33 ) [ge\_ports ( AN[1] )]; 410\_L325\_73\_778 m\_15\_15\_Schman[1] IOSTMUNARD LVCMOS33 ) [ge\_ports ( AN[2] )]; 410\_L426\_73\_A0\_L017\_14\_Schman[1] IOSTMUNARD LVCMOS33 ) [ge\_ports ( AN[4] )]; 410\_L80\_T1\_D12\_14\_Schman[1] IOSTMUNARD LVCMOS33 ) [ge\_ports ( AN[4] )]; 410\_L80\_T1\_D12\_14\_Schman[1] IOSTMUNARD LVCMOS33 ) [ge\_ports ( AN[4] )]; 410\_L80\_T1\_D12\_14\_Schman[4] IOSTMUNARD LVCMOS33 ) [ge\_ports ( AN[4] )]; 410\_L80\_T1\_D12\_14\_Schman[4] IOSTMUNARD LVCMOS33 ) [ge\_ports ( AN[4] )]; 410\_L80\_T1\_D12\_15\_Schman[6] ret property =arct ( FACKAGE FIN K2 = 1051AMDARD LVCM0535 ) [get ports ( AN(5) )]; #10 225F 15 35 5cm=an(5) rest property =dict ( FACKAGE FIN U13 = 1051AMDARD LVCM0533 ) [get ports ( AN(7) )]; #10 223N T3 A02 D18 14 Sch=an(7) R0 ##Buttons 81 82 #set\_property -dict ( PACKAGE\_PIN C12 IOSTANDARD LVCMOS33 ) [get\_ports ( CFU\_RESEIN )]; #IO\_L3P\_TO\_DQS\_ADIP\_15 Sch=cpu\_resetn 84 #set property -dict ( PACKAGE PIN N17 IOSTANDARD LVCMOS33 ) [get ports ( BINC )]; #IO L9P T1 DQS 14 Sch=btnc 6 #set\_property -dict ( PACKAGE\_PIN P18 IOTANDARD LVCMOS3 ) [get\_ports [ DIN ]); #10\_LHT\_DD\_14 Schwebra 66 #set\_property -dict ( PACKAGE\_PIN P17 IOTANDARD LVCMOS3 ) [ get\_ports [ DIN ]); #10\_LHT\_DD\_14 Schwebra 66 #set\_property -dict ( PACKAGE\_PIN P17 IOTANDARD LVCMOS3 ) [ get\_ports [ DIN ]); #10\_LINT\_LDS\_14 Schwebra 80 #set\_property -dict ( PACKAGE\_PIN P18 IOTANDARD LVCMOS3 ) [ get\_ports [ DIN ]); #10\_LNT\_LDS\_14 Schwebra 80 #set\_property -dict ( PACKAGE\_PIN P18 IOTANDARD LVCMOS3 ) [ get\_ports ( DIN ]); #10\_LNT\_LDS\_14 Schwebra 80 #set\_property -dict ( PACKAGE\_PIN P18 IOTANDARD LVCMOS3 ) [ get\_ports ( DIN ]); #10\_LNT\_LDS\_14 Schwebra 80 #set\_property -dict ( PACKAGE\_PIN P18 IOTANDARD LVCMOS3 ) [ get\_ports ( DIN ]); #10\_LNT\_LDS\_14 Schwebra 80 #set\_property -dict ( PACKAGE\_PIN P18 IOTANDARD LVCMOS3 ) [ get\_ports ( DIN ]); #10\_LNT\_LDS\_14 Schwebra 80 #set\_property -dict ( PACKAGE\_PIN P18 IOTANDARD LVCMOS3 ) [ get\_ports ( DIN ]); #10\_LNT\_LDS\_14 Schwebra 80 #set\_property -dict ( PACKAGE\_PIN P18 IOTANDARD LVCMOS3 ) [ get\_ports ( DIN ]); #10\_LNT\_LDS\_14 Schwebra 80 #set\_property -dict ( PACKAGE\_PIN P18 IOTANDARD LVCMOS3 ) [ get\_ports ( DIN ]); #10\_LNT\_LDS\_14 Schwebra 80 #set\_property -dict ( PACKAGE\_PIN P18 IOTANDARD LVCMOS3 ) [ get\_ports ( DIN ]); #10\_LNT\_LDS\_14 Schwebra 80 #set\_property -dict ( PACKAGE\_PIN P18 IOTANDARD LVCMOS3 ) [ get\_ports ( DIN ]); #10\_LNT\_LDS\_14 Schwebra 80 #set\_property -dict ( PACKAGE\_PIN P18 IOTANDARD LVCMOS3 ) [ get\_ports ( DIN ]); #10\_LNT\_LDS\_14 Schwebra 80 #set\_property -dict ( PACKAGE\_PIN P18 IOTANDARD LVCMOS3 ) [ get\_ports ( DIN ]); #10\_LNT\_LDS\_14 Schwebra 80 #set\_property -dict ( PACKAGE\_PIN P18 IOTANDARD LVCMOS3 ) [ get\_ports ( DIN ]); #10\_LNT\_LDS\_14 Schwebra 80 #set\_property -dict ( PACKAGE\_PIN P18 IOTANDARD LVCMOS3 ) [ get\_ports ( DIN ]); #10\_LNT\_LDS\_14 Schwebra 80 #set\_property -dict ( PACKAGE\_PIN P18 IOTANDARD LVCMOS3 ) [ get\_ports ( DIN ]] [ get\_ports ( DIN

91 ##Pmod Headers 92 93 94 ##Pmod Header JA 95 
 95

 96 #set\_property -dict ( PACKAGE\_PIN C17 IOSTANDARD LVCNOS33 ) [get\_ports ( JA[1] )]; #10 L20N T3 A19\_15 Sch=ja[1]

 97 #set\_property -dict ( PACKAGE\_PIN B18 IOSTANDARD LVCNOS33 ) [get\_ports ( JA[1] )]; #10 L21N T3 D05\_18 Sch=ja[3]

 96 #set\_property -dict ( PACKAGE\_PIN B18 IOSTANDARD LVCNOS33 ) [get\_ports ( JA[3] )]; #10 L21N T3 D05\_18 Sch=ja[3]

 96 #set\_property -dict ( PACKAGE\_PIN B18 IOSTANDARD LVCNOS33 ) [get\_ports ( JA[4] )]; #10 L21N T3 D05\_18 Sch=ja[4]

 100 #set\_property -dict ( PACKAGE\_PIN B17 IOSTANDARD LVCNOS33 ) [get\_ports ( JA[4] )]; #10 L14N T2 A3\_15 Sch=ja[4]

 101 #set\_property -dict ( PACKAGE\_PIN B17 IOSTANDARD LVCNOS33 ) [get\_ports ( JA[4] )]; #10 L14N T2 A3\_15 Sch=ja[4]

 101 #set\_property -dict ( PACKAGE\_PIN T17 IOSTANDARD LVCNOS33 ) [get\_ports ( JA[4] )]; #10 L14N T2 A3\_15 Sch=ja[4]

 102 #set\_property -dict ( PACKAGE\_PIN T17 IOSTANDARD LVCNOS33 ) [get\_ports ( JA[4] )]; #10 L14N T2 A3\_15 Sch=ja[4]

 103 #set\_property -dict ( PACKAGE\_PIN T18 IOSTANDARD LVCNOS33 ) [get\_ports ( JA[4] )]; #10 L14N T2 A3\_15 Sch=ja[4]

 103 #set\_property -dict ( PACKAGE\_PIN T18 IOSTANDARD LVCNOS33 ) [get\_ports ( JA[1] )]; #10 L22N T3 A16\_15 Sch=ja[6]

 103 #set\_property -dict ( PACKAGE\_PIN G18 IOSTANDARD LVCNOS33 ) [get\_ports ( JA[10] )]; #10 L22N T3 A16\_15 Sch=ja[6]

 104
 IOSTANDARD LVCNOS33 ) [get\_ports ( JA[10] )]; #10 L22N T3 A16\_15 Sch=ja[10]
 105 106 ##Pmod Header JB 107
108 #set\_property -dict ( PACKAGE\_PIN D14 IOSTANDARD LVCMOS33 ) [get\_ports ( JB[1) ]); #IO\_LIP\_TO\_ADOP\_IS\_Sch=jb[1]
109 #set\_property -dict ( PACKAGE\_PIN F16 IOSTANDARD LVCMOS33 ) [get\_ports ( JB[2) ]); #IO\_LIM\_T2\_SCC\_IS\_Sch=jb[2]
110 #set\_property -dict ( PACKAGE\_PIN F16 IOSTANDARD LVCMOS33 ) [get\_ports ( JB[2) ]); #IO\_LIM\_T2\_MCC\_LIS\_Sch=jb[4]
111 #set\_property -dict ( PACKAGE\_PIN F16 IOSTANDARD LVCMOS33 ) [get\_ports ( JB[2) ]); #IO\_LIM\_T2\_MCC\_LIS\_Sch=jb[4]
112 #set\_property -dict ( PACKAGE\_PIN F16 IOSTANDARD LVCMOS33 ) [get\_ports ( JB[2) ]); #IO\_LIM\_T2\_MCC\_LIS\_Sch=jb[4]
113 #set\_property -dict ( PACKAGE\_PIN F16 IOSTANDARD LVCMOS33 ) [get\_ports ( JB[2) ]); #IO\_LIM\_T1\_SCC\_LIS\_Sch=jb[4]
114 #set\_property -dict ( PACKAGE\_PIN F16 IOSTANDARD LVCMOS33 ) [get\_ports ( JB[2) ]); #IO\_LIM\_T1\_SCC\_LIS\_Sch=jb[4]
114 #set\_property -dict ( PACKAGE\_PIN F16 IOSTANDARD LVCMOS33 ) [get\_ports ( JB[2) ]); #IO\_LIM\_T1\_SCC\_LIS\_Sch=jb[4]
115 #set\_property -dict ( PACKAGE\_PIN F16 IOSTANDARD LVCMOS33 ) [get\_ports ( JB[1) ]); #IO\_LIM\_T1\_SCC\_LIS\_Sch=jb[4]
115 #set\_property -dict ( PACKAGE\_PIN F16 IOSTANDARD LVCMOS33 ) [get\_ports ( JB[1) ]); #IO\_LIM\_T1\_SCC\_LIS\_Sch=jb[4]
115 #set\_property -dict ( PACKAGE\_PIN F16 IOSTANDARD LVCMOS33 ) [get\_ports ( JB[10) ]); #IO\_LIM\_T1\_SCC\_LIS\_Sch=jb[4]
115 #set\_property -dict ( PACKAGE\_PIN F16 IOSTANDARD LVCMOS33 ) [get\_ports ( JB[10) ]); #IO\_LIM\_T2\_Sch=jb[4]
115 #set\_property -dict ( PACKAGE\_PIN F16 IOSTANDARD LVCMOS33 ) [get\_ports ( JB[10) ]); #IO\_LIM\_T3\_Sch=jb[4]
115 #set\_property -dict ( PACKAGE\_PIN F16 IOSTANDARD LVCMOS33 ) [get\_ports ( JB[10) ]); #IO\_LIM\_T3\_Sch=jb[4]
116 #set\_property -dict ( PACKAGE\_PIN H16 IOSTANDARD LVCMOS33 ) [get\_ports ( JB[10) ]); #IO\_LIM\_T3\_Sch=jb[4]
116 #set\_property -dict ( PACKAGE\_PIN H16 IOSTANDARD LVCMOS33 ) [get\_ports ( JB[10) ]); #IO\_LIM\_T3\_Sch=jb[4]
116 #set\_property -dict ( PACKAGE\_PIN H16 IOSTANDARD LVCMOS33 ) [get\_ports ( JB[10) ]); #IO\_LIM\_T3\_Sch=jb[4]
116 #set\_property -dict ( PACKAGE\_PIN H16 IOSTANDARD LVCMOS33 ) [get\_ports ( JB[10) ]); #IO\_LIM\_T3\_Sch=jb[4 
 IE
 Available uvinkods / [get\_ports ( J8[10] )]; #IO\_LISP\_T2\_MRCC\_15 Sch=jb[10]

 121 #set\_property -dict ( PACKAGE\_PIN F6
 IOSTANDARD UVINKOdS / [get\_ports ( JC[2] )]; #IO\_LISP\_T2\_MRCC\_15 Sch=jb[10]

 122 #set\_property -dict ( PACKAGE\_PIN F6
 IOSTANDARD UVINKOdS / [get\_ports ( JC[2] )]; #IO\_LISP\_T3 Sch=jc[2]

 123 #set\_property -dict ( PACKAGE\_PIN F6
 IOSTANDARD UVINKOdS / [get\_ports ( JC[2] )]; #IO\_LISP\_T3 Sch=jc[2]

 124 #set\_property -dict ( PACKAGE\_PIN F6
 IOSTANDARD UVINKOS3 ) [get\_ports ( JC[1] )]; #IO\_LISP\_T3 Sch=jc[2]

 124 #set\_property -dict ( PACKAGE\_PIN 73
 IOSTANDARD UVINKOS3 ) [get\_ports ( JC[2] )]; #IO\_LISP\_T3 Sch=jc[2]

 125 #set\_property -dict ( PACKAGE\_PIN 74
 IOSTANDARD UVINKOS3 ) [get\_ports ( JC[2] )]; #IO\_LISP\_T3 Sch=jc[2]

 126 #set\_property -dict ( PACKAGE\_PIN 74
 IOSTANDARD UVINKOS3 ) [get\_ports ( JC[2] )]; #IO\_LISP\_T3 Sch=jc[2]

 127 #set\_property -dict ( PACKAGE\_PIN 74
 IOSTANDARD UVINKOS3 ) [get\_ports ( JC[10] )]; #IO\_LISP\_T3 Sch=jc[2]

 127 #set\_property -dict ( PACKAGE\_PIN 74
 IOSTANDARD UVINKOS3 ) [get\_ports ( JC[10] )]; #IO\_LISP\_T0 ADI3P\_35 Sch=jc[10]

 128
 IOSTANDARD UVINKOS3 ) [get\_ports ( JC[10] )]; #IO\_LISP\_T0 ADI3P\_35 Sch=jc[10]
 129 130 ##Pmod Header JD 
 131

 132
 #set\_property -dict ( PACKAGE\_PIN #4
 IOSTANDARD LVCMOS33 ) [get\_ports ( JD[1] )]; #IO\_L1N\_T3\_D95\_35 Sch=jd[1]

 133
 #set\_property -dict ( PACKAGE\_PIN #1
 IOSTANDARD LVCMOS33 ) [get\_ports ( JD[1] )]; #IO\_L1N\_T2\_35 Sch=jd[2]

 134
 #set\_property -dict ( PACKAGE\_PIN #1
 IOSTANDARD LVCMOS33 ) [get\_ports ( JD[3] )]; #IO\_L1N\_T2\_35 Sch=jd[3]

 135
 #set\_property -dict ( PACKAGE\_PIN #1
 IOSTANDARD LVCMOS33 ) [get\_ports ( JD[3] )]; #IO\_L1N\_T2\_35 Sch=jd[3]

 135
 #set\_property -dict ( PACKAGE\_PIN #2
 IOSTANDARD LVCMOS33 ) [get\_ports ( JD[1] )]; #IO\_L1N\_T3\_35 Sch=jd[4]

 136
 #set\_property -dict ( PACKAGE\_PIN #2
 IOSTANDARD LVCMOS33 ) [get\_ports ( JD[7] )]; #IO\_L1N\_T3\_S5 Sch=jd[7]

 137
 #set\_property -dict ( PACKAGE\_PIN #2
 IOSTANDARD LVCMOS33 ) [get\_ports ( JD[7] )]; #IO\_L1N\_T2\_D5 Sch=jd[6]

 138
 #set\_property -dict ( PACKAGE\_PIN #2
 IOSTANDARD LVCMOS33 ) [get\_ports ( JD[7] )]; #IO\_L1N\_T2\_D5 Sch=jd[6]

 138
 #set\_property -dict ( PACKAGE\_PIN #2
 IOSTANDARD LVCMOS33 ) [get\_ports ( JD[0] )]; #IO\_L1N\_T2\_D5 Sch=jd[6]

 139
 #set\_property -dict ( PACKAGE\_PIN #2
 IOSTANDARD LVCMOS33 ) [get\_ports ( JD[0] )]; #IO\_L1N\_T2\_MRCC\_35 Sch=jd[10]

 140
 #set\_property -dict ( PACKAGE\_PIN #3
 IOSTANDARD LVCMOS3 ) [get\_ports ( JD[10] )]; #IO\_L1N\_T2\_MRCC\_35 Sch=jd[10]
 140 141 142 ##Pmod Header JXADC 14 faile 143 144 set property -dict ( PACKAGE PIN A14 IOSTANDARD LVDS 145 fast\_property -dict ( PACKAGE PIN A13 IOSTANDARD LVDS 146 fast\_property -dict ( PACKAGE\_PIN A16 IOSTANDARD LVDS ) [get\_ports ( XA\_N[1] )]; #IO\_LSM\_TI\_DOS\_ADJN\_IS Sch=xa\_n[1] ) [get\_ports ( XA\_P[1] )]; #IO\_LSM\_TI\_DOS\_ADJP\_IS Sch=xa\_n[1] ) [get\_ports ( XA\_N[2] )]; #IO\_LSM\_TI\_ADION\_IS Sch=xa\_n[2] ) [get\_ports ( XA\_N[3] )]; #IO\_LSM\_TI\_ADIOP\_IS Sch=xa\_n[3] ) [get\_ports ( XA\_N[3] )]; #IO\_LTM\_TI\_ADIN\_IS Sch=xa\_n[3] ) [get\_ports ( XA\_N[4] )]; #IO\_LTM\_TI\_ADIN\_IS Sch=xa\_n[4] ) [get\_ports ( XA\_P[4] )]; #IO\_LIN\_TI\_ADIN\_IS Sch=xa\_n[4] ) [get\_ports ( XA\_P[4] )]; #IO\_LIN\_TI\_ADIN\_IS Sch=xa\_n[4] 140 #set property -dict ( PACKAGE\_PIN ALE IOSTANDARD LVDS 147 #set property -dict ( PACKAGE PIN BL7 IOSTANDARD LVDS 148 #set property -dict ( PACKAGE PIN BL7 IOSTANDARD LVDS 150 #set property -dict ( PACKAGE PIN BL8 IOSTANDARD LVDS 151 #set\_property -dict ( PACKAGE PIN BL8 IOSTANDARD LVDS 152 153 154 ##VGA Connector 156 #Jose Connector 155 156 #Jose property -dict ( PACKAGE PIN A3 157 #Jose property -dict ( PACKAGE PIN B4 158 #Jose property -dict ( PACKAGE PIN C5 IOSTANDARD LVCMOS33 ) [get\_ports ( VGA\_R[0] )]; #IO\_LSM\_TI\_AD14N\_35 Sch=vqa\_r[0] IOSTANDARD LVCMOS33 ) [get\_ports ( VGA\_R[1] )]; #IO\_LNT\_TI\_AD6N\_35 Sch=vqa\_r[1] IOSTANDARD LVCMOS33 ) [get\_ports ( VGA\_R[3] )]; #IO\_LLI\_TO\_AD4N\_35 Sch=vqa\_r[3] IOSTANDARD LVCMOS33 ) [get\_ports ( VGA\_R[3] )]; #IO\_LST\_TI\_AD14P\_35 Sch=vqa\_r[3] 159 #set property -dict ( PACKAGE PIN A4 160 160 161 #set\_property -dict ( PACKAGE\_PIN C6 162 #set\_property -dict ( PACKAGE\_PIN A5 163 #set\_property -dict ( PACKAGE\_PIN B6 164 #set\_property -dict ( PACKAGE\_PIN A6 IOSTANDARD LVCMO333 ) [get\_ports ( V&A\_G[0] )]; #IO\_LIP\_TO\_AD4P\_35 Sch=vqa\_g[0] IOSTANDARD LVCMO333 ) [get\_ports ( V&A\_G[1] )]; #IO\_S31\_TO\_DO5\_AD5N\_35 Sch=vqa\_g[2] IOSTANDARD LVCMO333 ) [get\_ports ( V&GA\_G[2] )]; #IO\_LST\_TO\_AD03 Sch=vqa\_g[2] IOSTANDARD LVCMO333 ) [get\_ports ( V&A\_G[3] )]; #IO\_LSP\_TO\_DO5\_AD5P\_35 Sch=vqa\_g[3] 165
166 #set\_property -dict ( PACKAGE\_PIN B7
167 #set\_property -dict ( PACKAGE\_PIN C7
168 #set\_property -dict ( PACKAGE\_PIN D7
169 #set\_property -dict ( PACKAGE\_PIN D8
170 IOSTANDARD LVCMOS33 ) [get\_ports ( VGA\_B[0] )]; #IO\_L2P\_TO\_AD12P\_35 Sch=vga\_b[0] IOSTANDARD LVCMOS33 ) [get\_ports ( VGA\_B[1] )]; #IO\_LAN\_TO\_35 Sch=vga\_b[1] IOSTANDARD LVCMOS33 ) [get\_ports ( VGA\_B[2] )]; #IO\_LAN\_TO\_VREF\_35 Sch=vga\_b[2] IOSTANDARD LVCMOS33 ) [get\_ports ( VGA\_B[3] )]; #IO\_L4P\_T0\_35 Sch=vga\_b[3] 174 175 ##Micro SD Connector 176 176 #set\_property -dict ( PACKAGE\_PIN E2 178 #set\_property -dict ( PACKAGE\_PIN A1 179 #set\_property -dict ( PACKAGE\_PIN B1 180 #set\_property -dict ( PACKAGE\_PIN C1 IOSTANDARD LVCMOS33 ) [get\_ports ( SD\_RESET )]; #IO\_L14P\_T3\_SRCC\_35 Sch=sd\_reset IOSTANDARD LVCMOS33 ) [get\_ports ( SD\_CC) ]; #IO\_LNN T1\_DGS\_ADTN\_35 Sch=sd\_cd IOSTANDARD LVCMOS33 ) [get\_ports ( SD\_SCK )]; #IO\_LNP\_T1\_DGS\_ADTP\_35 Sch=sd\_ck IOSTANDARD LVCMOS33 ) [get\_ports ( SD\_CMD )]; #IO\_L16N\_T2\_35 Sch=sd\_cmd

 181 #set\_property -dict ( PACKAGE\_PIN C2
 IOSTANDARD LVCMOS33 ) [get\_ports ( SD\_DAT[0) ]); #TO\_LI6P\_T2\_S5 Sch=ed\_dat[0]

 182 #set\_property -dict ( PACKAGE\_PIN E1
 IOSTANDARD LVCMOS33 ) [get\_ports ( SD\_DAT[1) ]); #TO\_LINT T2\_S5 Sch=ed\_dat[1]

 183 #set\_property -dict ( PACKAGE\_PIN E1
 IOSTANDARD LVCMOS33 ) [get\_ports ( SD\_DAT[1) ]); #TO\_LENT T2\_S5 Sch=ed\_dat[1]

 184 #set\_property -dict ( PACKAGE\_PIN E2
 IOSTANDARD LVCMOS33 ) [get\_ports ( SD\_DAT[1] ]); #TO\_LINT T2\_S5CC\_S5 Sch=ed\_dat[3]

 184 #set\_property -dict ( PACKAGE\_PIN D2
 IOSTANDARD LVCMOS33 ) [get\_ports ( SD\_DAT[3] ]); #TO\_LINT T2\_S5CC\_S5 Sch=ed\_dat[3]

 185 185 187 ##Accelerometer 188 189 #set\_property -dict ( PACKAGE\_PIN E15 IOSTANDARD LVCN0533 ) [get\_ports ( ACL\_MISO )]; #IO\_LIIP\_TL\_SRCC\_15 Schwacl\_miso
180 #set\_property -dict ( PACKAGE\_PIN F14 IOSTANDARD LVCN0533 ) [get\_ports ( ACL\_MOSI )]; #IO\_LINP\_TL\_SRCC\_15 Schwacl\_mosi
181 #set\_property -dict ( PACKAGE\_PIN F15 IOSTANDARD LVCN0533 ) [get\_ports ( ACL\_SCL )]; #IO\_LINP\_TL\_RCC\_15 Schwacl\_acl
182 #set\_property -dict ( PACKAGE\_PIN L5 IOSTANDARD LVCN0533 ) [get\_ports ( ACL\_SCL )]; #IO\_LINP\_TL\_RCC\_15 Schwacl\_acl
183 #set\_property -dict ( PACKAGE\_PIN L5 IOSTANDARD LVCN0533 ) [get\_ports ( ACL\_SCL )]; #IO\_LINP\_TL\_RCC\_15 Schwacl\_and[
184 #set\_property -dict ( PACKAGE\_PIN L5 IOSTANDARD LVCN0533 ) [get\_ports ( ACL\_SCL )]; #IO\_LINP\_TQ\_ASC\_15 Schwacl\_int[1]
184 #set\_property -dict ( PACKAGE\_PIN C16 IOSTANDARD LVCN0S33 ) [get\_ports ( ACL\_SCL )]; #IO\_LINP\_TQ\_ASC\_15 Schwacl\_int[2]
185 195 196 197 ##Temperature Sensor 198 199 #set\_property -dict ( FACHAGE\_FIN C14 IOSTANDARD LVCNOS33 ) [get\_ports ( TMP\_SCL )]; #IO\_LIN\_TO\_ADON\_IS Sch=tmp\_scl 200 #set\_property -dict ( FACHAGE\_FIN C15 IOSTANDARD LVCNOS33 ) [get\_ports ( TME\_SCA )]; #IO\_LIN\_TI\_MRCC\_IS Sch=tmp\_scl 201 #set\_property -dict ( FACHAGE\_FIN 13 IOSTANDARD LVCNOS33 ) [get\_ports ( TME\_INT )]; #IO\_LAN\_TO\_KRES\_ISS.ch=tmp\_int 202 #set\_property -dict ( FACHAGE\_FIN B14 IOSTANDARD LVCNOS33 ) [get\_ports ( TMP\_CT )]; #IO\_LAN\_TO\_ADON\_IS Sch=tmp\_int 204 ##Omnidirectional Microphone 206 #set property -dict / PACKAGE PIN J5 IOSTANDARD LVCMOS33 \ [det ports / M CLK \]: #IO 25 35 Sch=m clk
211 ##PWM Audio Amplifier 214 #set\_property -dict ( PACKAGE\_PIN All IOSTANDARD LVCMOS33 ) [get\_ports ( AUD\_PWM )]; #IO\_L4N\_T0\_15 Sch=aud\_pvm
214 #set\_property -dict ( PACKAGE\_PIN D12 IOSTANDARD LVCMOS33 ) [get\_ports ( AUD\_SD )]; #IO\_L6P\_T0\_15 Sch=aud\_sd 215 216 217 ##USR\_RS232 Interface 217 #MUSE-RSJ2 interrace 318 219 #set property -dict ( FACKAGE\_FIN C4 IOSTANDARD LVCNOS33 ) [get\_ports ( UART\_TXD\_IN )]; #IO\_L7P\_T1\_AD6F\_35 Sch=uart\_txd\_in 200 #set property -dict ( FACKAGE\_FIN D4 IOSTANDARD LVCNOS33 ) [get\_ports ( UART\_RXD\_OUT )]; #IO\_L1N\_T1\_RSC\_35 Sch=uart\_txd\_out 221 #set\_property -dict ( FACKAGE\_FIN D3 IOSTANDARD LVCNOS33 ) [get\_ports ( UART\_CTS )]; #IO\_L1N\_T1\_NSC\_35 Sch=uart\_txs 222 #set\_property -dict ( FACKAGE\_FIN E5 IOSTANDARD LVCNOS33 ) [get\_ports ( UART\_RTS )]; #IO\_L5N\_T0\_AD13N\_35 Sch=uart\_txs \*\*\* 224 ##USB HID (PS/2) 225 226 føst\_property -dict ( PACKAGE\_PIN F4 IOSTANDARD LVCMOS33 ) [get\_ports ( PS2\_CLK )]; #IO\_L13P\_T2\_MRCC\_35\_Sch=ps2\_clk 227 føst\_property -dict ( PACKAGE\_PIN B2 IOSTANDARD LVCMOS33 ) [get\_ports ( PS2\_DATA )]; #IO\_L10N\_T1\_ADI5N\_35\_Sch=ps2\_data 238 229 230 ##SMSC Ethernet PHY 131
132 #set\_property -dict ( PACKAGE\_PIN 69
10STANDARD LVCNOS3 ) [get\_ports ( ETH\_NDC )]; #IO\_LIP\_TI\_SRCC\_16 Scheeth\_mdc
133 #set\_property -dict ( PACKAGE\_PIN 89
10STANDARD LVCNOS3 ) [get\_ports ( ETH\_RDC )]; #IO\_LIP\_TI\_SRCC\_16 Scheeth\_mdi
133 #set\_property -dict ( PACKAGE\_PIN 89
10STANDARD LVCNOS3 ) [get\_ports ( ETH\_RDC )]; #IO\_LINT\_TSRCC\_16 Scheeth\_rest
133 #set\_property -dict ( PACKAGE\_PIN 89
10STANDARD LVCNOS3 ) [get\_ports ( ETH\_RDC )]; #IO\_LINT\_TSRCC\_16 Scheeth\_rest
133 #set\_property -dict ( PACKAGE\_PIN 109
10STANDARD LVCNOS3 ) [get\_ports ( ETH\_RDC )]; #IO\_LINT\_TSRCC\_16 Scheeth\_rest
133 #set\_property -dict ( PACKAGE\_PIN 109
10STANDARD LVCNOS3 ) [get\_ports ( ETH\_RDC )]; #IO\_LINT\_TSRCC\_16 Scheeth\_rest
133 #set\_property -dict ( PACKAGE\_PIN 100
10STANDARD LVCNOS3 ) [get\_ports ( ETH\_RDC )]; #IO\_LINT\_TSRCC\_16 Scheeth\_rest
140 #set\_property -dict ( PACKAGE\_PIN 40
10STANDARD LVCNOS3 ) [get\_ports ( ETH\_RDC )]; #IO\_LINT\_TSRCC\_16 Scheeth\_rest
140 #set\_property -dict ( PACKAGE\_PIN 40
10STANDARD LVCNOS3 ) [get\_ports ( ETH\_RDC )]; #IO\_LINT\_TSRCC\_16 Scheeth\_rest
141 #set\_property -dict ( PACKAGE\_PIN 40
10STANDARD LVCNOS3 ) [get\_ports ( ETH\_RDC )]; #IO\_LINT\_TSRCC\_16 Scheeth\_rest
142 #set\_property -dict ( PACKAGE\_PIN 40
10STANDARD LVCNOS3 ) [get\_ports ( ETH\_RTNC )]; #IO\_LINT\_TSRCC\_16 Scheeth\_rest
143 #set\_property -dict ( PACKAGE\_PIN 40
10STANDARD LVCNOS3 ) [get\_ports ( ETH\_RTNC )]; #IO\_LINT\_TSRCC\_16 Scheeth\_rest
143 #set\_property -dict ( PACKAGE\_PIN 40
10STANDARD LVCNOS3 ) [get\_ports ( ETH\_RTNC )]; #IO\_LINT\_TSRCC\_16 Scheeth\_rest
143 #set\_property -dict ( PACKAGE\_PIN 40
10STANDARD LVCNOS3 ) [get\_ports ( ETH\_RTNC )]; #IO\_LINT\_TSRCC\_16 Scheeth\_rest
143 #set\_property -dict ( PACKAGE\_PIN 40
10STANDARD LVCNOS3 ) [get\_ports ( ETH\_RTNC )]; #IO\_LINT\_TSRCC\_16 Scheeth\_rest
143 #set\_property -dict ( PACKAGE\_PIN 40
10STANDARD LVCNOS3 ) [get\_ports ( ETH\_RTNC )]; #IO\_LINT\_TSRCC\_16 Scheeth\_rest
143 #set\_property -dict ( PACKAGE\_PIN 40
10STANDARD LVCNOS3 ) [get\_ports ( ETH\_RTNC )]; #IO\_LINT\_TSRCC\_16 Scheeth\_rest
143 #set\_property -di 244 245 246 ##Quad SPI Flash 247 247 248 #set\_property -dict ( PACKAGE\_PIN K17 IOSTANDARD LVCNOS33 ) [get\_ports ( OSFI\_D0[0] )]; #IO\_LIP\_T0\_D00\_MOSI\_14 Sch=qspi\_dq[0] 249 #set\_property -dict ( PACKAGE\_PIN K18 IOSTANDARD LVCNOS33 ) [get\_ports ( OSFI\_D0[1] )]; #IO\_LIP\_T0\_D01\_H Sch=qspi\_dq[1] 250 #set\_property -dict ( PACKAGE\_PIN L14 IOSTANDARD LVCNOS33 ) [get\_ports ( OSFI\_D0[2] )]; #IO\_LZP\_T0D\_D01\_H Sch=qspi\_dq[2] 251 #set\_property -dict ( PACKAGE\_PIN L14 IOSTANDARD LVCNOS33 ) [get\_ports ( OSFI\_D0[3] )]; #IO\_LZP\_T0D\_D01\_H Sch=qspi\_dq[3] 252 #set\_property -dict ( PACKAGE\_PIN L14 IOSTANDARD LVCNOS33 ) [get\_ports ( OSFI\_D0[3] )]; #IO\_LZP\_T0\_D03\_H Sch=qspi\_dq[3] 252 #set\_property -dict ( PACKAGE\_PIN L13 IOSTANDARD LVCNOS33 ) [get\_ports ( OSFI\_CSN )]; #IO\_LGP\_T0\_FS\_B\_14 Sch=qspi\_csn

THIS PAGE INTENTIONALLY LEFT BLANK

## LIST OF REFERENCES

- [1] J. C. Coudeyras, "Radiation testing of the Configurable Fault Tolerant Processor (CFTP) for space-based applications," M.S. thesis, Dept. of Elec. Eng., NPS, Monterey, CA, USA, 2005.
- [2] C. J. Humberd, "A compression algorithm for field programmable gate arrays in the space environment," M.S. thesis, Dept. of Elec. Eng., NPS, Monterey, CA, USA, 2011.
- [3] R. F. Bernstein, Jr. "A pipelined vector processing and memory architecture for cyclostationary processing," Ph.D. dissertation, Dept. of Elec. Eng., NPS, Monterey, CA, USA, 1995.
- [4] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, USA, 1989.
- [5] D. A. Ebert, "Design and development of a Configurable Fault-Tolerant Processor (CFTP) for space applications," M.S. thesis, Dept. of Elec. Eng., NPS, Monterey, CA, USA, 2003.
- [6] MidSTAR. (n.d.). [Online]. Available FTP: \\it154512\CFTP Directory: DesignDocs\Midstar File: midstar\_external\_view
- [7] D Sakoda. (n.d.). NPSAT-1 Spacecraft architecture and technology demostration satellite [Online]. Available FTP: \\it154512\CFTP Directory: NPSat1\ Sponsor091806 File: NPSAT1\_Brief
- [8] P. J. Majewicz, "Implementation of a Configurable Fault Tolerant Processor (CFTP) using Internal Triple Modular Redundancy (TMR)," M.S. thesis, Dept. of Elec. Eng., NPS, Monterey, CA, USA, 2005.
- [9] G. W. Caldwell, "Implementation of Configurable Fault Tolerant Processor (CFTP) experiments," M.S. thesis, Dept. of Elec. Eng., NPS, Monterey, CA, USA, 2006.
- [10] D. E. Dwiggins, "Fault tolerant microcontroller for the configurable Fault Tolerant Processor," M.S. thesis, Dept. of Elec. Eng., NPS, Monterey, CA, USA, 2008.
- [11] M. A. Sullivan, "Reduced precision redundancy applied to arithmetic operations in field programmable gate arrays for satellite control and sensor systems," M.S. thesis, Dept. of Elec. Eng., NPS, Monterey, CA, USA, 2008.

- [12] J. D. Snodgrass, "Low-power fault tolerance for spacecraft FPGA-based numberical computing," M.S. thesis, Dept. of Elec. Eng., NPS, Monterey, CA, USA, 2009.
- [13] J. V. Livingston, "A field programmable gate array based software defined radio design for the space environment," M.S. thesis, Dept. of Elec. Eng., NPS, Monterey, CA, USA, 2009.
- [14] A. S. Jackson, "Implementation of the configurable fault tolerant system experiment on NPSAT-1," M.S. thesis, Dept. of Elec. Eng., NPS, Monterey, CA, USA, March 2016.
- [15] M. L. Zimmer, "A VLSI design of a radix-4 floating point FFT butterfly," M.S. thesis, Dept. of Elec. Eng., NPS, Monterey, CA, USA, 1991.

# **INITIAL DISTRIBUTION LIST**

- 1. Defense Technical Information Center Ft. Belvoir, Virginia
- 2. Dudley Knox Library Naval Postgraduate School Monterey, California