NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

FIELD PROGRAMMABLE GATE ARRAY HIGH CAPACITY TECHNOLOGY FOR RADAR AND COUNTER-RADAR DRFM SIGNAL PROCESSING

by

Hawken L. Grubbs

June 2018

Thesis Advisor: Phillip E. Pace
Co-Advisor: Steven J. Iatrou

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** June 2018	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE
FIELD PROGRAMMABLE GATE ARRAY HIGH CAPACITY TECHNOLOGY FOR RADAR AND COUNTER-RADAR DRFM SIGNAL

5. FUNDING NUMBERS

6. AUTHOR(S) Hawken L. Grubbs

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
CENTER FOR JOINT SERVICES ELECTRONIC WARFARE
Naval Postgraduate School, Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
CRUSER, Monterey, CA 93943

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release. Distribution is unlimited.

12b. DISTRIBUTION CODE
A

13. ABSTRACT (maximum 200 words)

Radar systems often use low power, continuous waveform radio frequency (RF) modulations and require high-speed adaptive signal processors to provide the necessary processing gain to detect small radar cross-section targets in clutter on range-Doppler maps. Counter-radar technologies include digital RF memories (DRFMs) that attempt to provide multiple, structured false targets with clutter, for example, using a pipelined, finite impulse response arrangement of complex range bin processors. This thesis investigates high-capacity field-programmable gate array (FPGA) technology to enable on-the-fly flexibility and reconfigurability for both radar signal processing and DRFM electronic attack using a Xilinx Virtex Ultrascale+. A three-stage range, Doppler, post-detection integration radar modulation compression circuit is designed and quantified. A range compression circuit with a peak power consumption of 6.100W and a post-implementation utilization of 11% was designed. The Doppler filter bank was designed at 400 MHz with a peak power consumption of 2.688W and a post-implementation utilization of 9%. A coherent integration processor at 400 MHz had a peak power consumption of 2.517W and a post-implementation utilization of 9%. In addition, a DRFM complex range bin processor was designed and quantified at 500 MHz and had a peak power 2.543W with a post-implementation utilization of 11%.

14. SUBJECT TERMS
LPI radar, low probability of intercept radar, FPGA, field programmable gate array, Virtex Ultrascale+, Vivado, Simulink, DSP, digital signal processing, DRFM, Xilinx

15. NUMBER OF PAGES 149

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

FIELD PROGRAMMABLE GATE ARRAY HIGH CAPACITY TECHNOLOGY FOR RADAR AND COUNTER-RADAR DRFM SIGNAL PROCESSING

Hawken L. Grubbs
Captain, United States Marine Corps
BS, University of Oklahoma, 2008

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN INFORMATION WARFARE SYSTEMS ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
June 2018

Approved by: Phillip E. Pace
Advisor

Steven J. Iatrou
Co-Advisor

Dan C. Boger
Chair, Department of Information Sciences

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Radar systems often use low power, continuous waveform radio frequency (RF) modulations and require high-speed adaptive signal processors to provide the necessary processing gain to detect small radar cross-section targets in clutter on range-Doppler maps. Counter-radar technologies include digital RF memories (DRFMs) that attempt to provide multiple, structured false targets with clutter, for example, using a pipelined, finite impulse response arrangement of complex range bin processors. This thesis investigates high-capacity field-programmable gate array (FPGA) technology to enable on-the-fly flexibility and reconfigurability for both radar signal processing and DRFM electronic attack using a Xilinx Virtex Ultrascale+. A three-stage range, Doppler, post-detection integration radar modulation compression circuit is designed and quantified. A range compression circuit with a peak power consumption of 6.100W and a post-implementation utilization of 11% was designed. The Doppler filter bank was designed at 400 MHz with a peak power consumption of 2.688W and a post-implementation utilization of 9%. A coherent integration processor at 400 MHz had a peak power consumption of 2.517W and a post-implementation utilization of 9%. In addition, a DRFM complex range bin processor was designed and quantified at 500 MHz and had a peak power 2.543W with a post-implementation utilization of 11%.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

ADC	analog-to-digital converter
ASIC	application specific integrated circuit
ASSP	application-specific standard parts
BPF	band-pass filter
CLB	configurable logic block
CMT	clock management tiles
CPLD	complex programmable logic device
CW	continuous wave
DAC	digital-to-analog converter
DCM	digital clock manager
DDS	direct digital synthesis
DIS	digital imaging synthesizer
DP	data processing
DRFM	digital radio frequency memory
DSB	double sideband
DSP	digital signal processing
EA	electronic attack
EW	electronic warfare
FF	flip-flop
FFT	fast Fourier transform
FMCW	frequency modulated CW
FPGA	field programmable gate array
FSK	frequency shift keying
Go-CFAR	greatest of (GO) constant false alarm rate (CFAR)
GPP	general purpose processor
I	in-phase
IF	intermediate frequency
IFFT	inverse fast Fourier transform
I/O	input and output
IOB	input/output block

IP	internet protocol
ISAR	inverse synthetic aperture radar
LO	local oscillator
LPF	low pass filter
LPI	low probability of intercept
LUT	lookup table
MAC	multiply and accumulate/add
MATLAB	matrix laboratory software
MIMO	minimum input minimum output
PLD	programmable logic device
PLL	phase locked loop
PROM	programmable read-only memory
PSK	polyphase shift keying
Q	quadrature
RAM	random access memory
RF	radio frequency
ROM	read only memory
RSNS	robust symmetrical number system
RSNS-P4	robust symmetrical number system – P4
SNR	signal-to-noise ratio
SoC	system on a chip
SRL	shift register
SSB	single sideband
VHDL	Verilog hardware description language
WHS	worst hold slack
WNS	worst negative slack

ACKNOWLEDGMENTS

I would like to take the time to thank my thesis advisor, Dr. Pace, for his leadership, mentorship, and guidance over the past year. This thesis experience has been an endeavor in which I have personally learned more than I could have ever imagined in this field of expertise. I would also like to thank Max Hainz for the countless hours he spent helping me understand the pre-existing MATLAB simulations and further developing the required Simulink models for this thesis. Without his time and dedication, this thesis would not have been possible. I would like to thank Steve Iatrou for his support as a co-advisor and guiding me through the IWSE curriculum while understanding that with my knowledge learned from NPS, that I could indeed take on such a demanding thesis topic. Additionally, many thanks are also aimed toward CRUSER personnel, who through their generosity and sponsorship, made it possible for me to explore thesis topic ideas and eventually attend the required courses to broaden my knowledge and understanding of the presented material.

I would like to thank my parents, Kenneth Brad and Rae Nita Grubbs, for instilling in me the absolute importance of an education as a young child and to never forget who I am and where I come from. Furthermore, I would like to thank my children, Colt, Levi, Chloe, Cheyenne, and Conner, for always being the light in my life. Each of you continually make me proud, especially with how well each of you handled the difficulty of family separation for the past two years. Last but certainly not least, I would like to thank the love of my life, Rachael, for the love and support you have given me over the past thirteen years. Without that love and support, I would certainly not be the man I am today. You are the strongest woman I know for being able to serve as a single mother in my absence while earning your undergraduate degree. You are a shining example for our children; may the future bring many more wonderful memories for us to hold as a family.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. RADAR SYSTEMS AND COUNTER-RADAR DIGITAL RF MEMORIES: THE NEED FOR HIGH CAPACITY EMBEDDED PROCESSORS

Low probability of intercept (LPI) radar systems are vital in many regions around the world requiring force protection through surveillance. These radar systems often use low power, continuous waveform (CW) modulations to remain undetected while retaining the "capability to detect and track targets in clutter" [1]. The transmitted CW signal provides a processing gain based on the radar's periodic modulation. The radar return waveform from the target is then distinguished using coherent compression. Many periodic modulation techniques are used for digital CW radar systems such as frequency shift keying (FSK), e.g., Costas sequencing, frequency modulated CW (FMCW), noise and noise modulated waveforms. Also used are polyphase shift keying (PSK) with codes such as the Frank and P4 codes. The PSK techniques have inherently low sidelobes and are also compatible with digital sidelobe suppression techniques. Other CW modulation techniques such as hybrid PSK/FSK combinations have also been recently a topic of active investigation [1].

Lately, much attention has been given to PSK due to the many digital CW modulations available. LPI characteristics are then achieved with a periodic autocorrelation function that has inherently low time sidelobes. The PSK modulations have a code period T containing limited amounts of range bins, each containing differing phase values. The subcode width t_b is related to both the waveform's range resolution and the 3-dB bandwidth while the "processing gain (PG) of the radar is equal to the number of subcodes" [1]. One important example of a PSK technique is the P4. The P4 code exhibits a perfect periodic autocorrelation in that it has zero sidelobes. Using a polyphase code for CW modulation however, presents a major limitation in that the modulation code period greatly confines the unambiguous range.

Extending the unambiguous range through the use of the Robust Symmetrical Number System (RSNS) has been investigated. By combining the RSNS with the P4

polyphase modulation, the unambiguous range can be divided up into different *moduli*, with each modulus relating to a different P4 code period. Transmitting the moduli (P4 code periods) in series and recombining the ambiguous detection results can resolve the targets range at considerably longer ranges than the unambiguous range pertaining to only 1 code period. In addition, the RSNS-P4 has Gray code properties that can serve to detect target range errors.

A digital RF memory (DRFM) is a device that can receive, store, modulate and retransmit, an RF signal. The DRFM is arguably one of the most important technologies for electronic warfare (EW) since it can retransmit an intercepted radar waveform and can put complex modulations on the return signal such that when integrated within the radar, can deceive for example, the track loop. To create a "structured" false target that has shape or amplitude in range-Doppler space, a standard DRFM kernel is not sufficient. That is, this type of target cannot be accomplished with a DRFM kernel alone. Hence, an augmented kernel must be used such as a high-speed, high capacity Field Programmable Gate Array (FPGA).

The circuit designs in this thesis are two-fold. First, the circuit design for a post analog-to-digital converter (ADC) compression process for a RSNS-P4 radar signal processor is designed for use within a Multifunction Sensor System for which the antenna is shown in Figure 1. A range compression processor is followed by a Doppler filter bank. This is then followed by a range-Doppler post-detection integration processor. A range compression circuit with a peak power consumption of 6.100W and a post-implementation utilization of 11% was designed. The Doppler filter bank was designed at 400 MHz with a peak power consumption of 2.688W and a post-implementation utilization of 9%. A coherent integration processor at 400 MHz had a peak power consumption of 2.517W and a post-implementation utilization of 9%. The second circuit design concept in this thesis is to design a DRFM single complex range bin processor that accepts the sampled input imaging radar waveform's phase value to first produce the range bin's desired Doppler profile using an accumulator and then the desired radar cross section using a multiplier. The final stage contains the range bin delay where the range bin consists of an adder. This

DRFM processor runs at 500 MHz and is quantified with a peak power 2.543W with a post-implementation utilization of 11%.

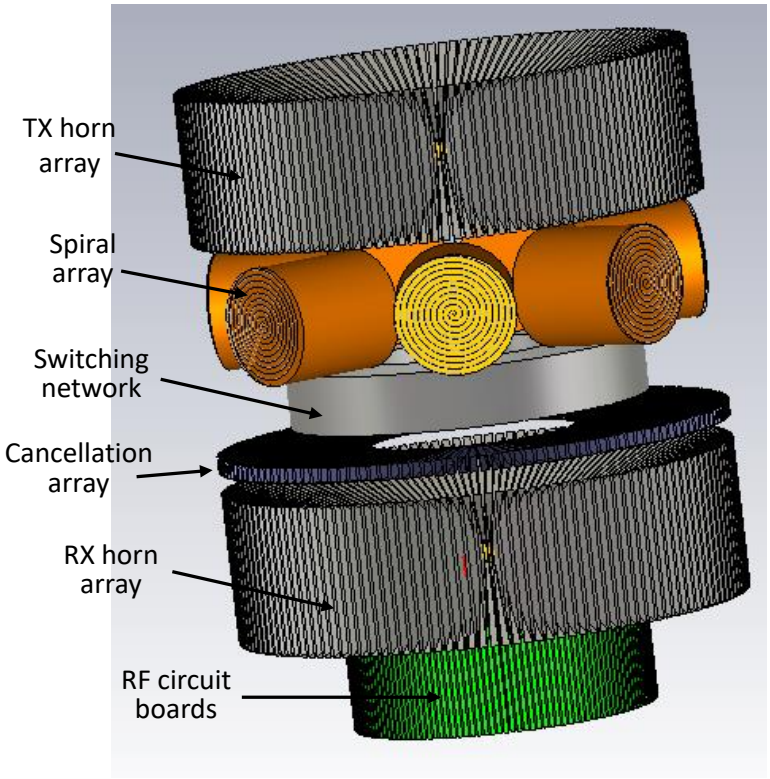

Figure 1. Multifunction Sensor System. Source: [2].

B. PRINCIPAL CONTRIBUTIONS

For this thesis, the first step was to investigate the concepts involved in radar signal processing. Detailed within the outline, the idea was to grasp an understanding from macroscopic to microscopic. This included the design requirements of an LPI radar, the specific modulation types used within the MATLAB code, and the three-stage compression which was the ultimate task at hand: to replicate the MATLAB version of the three-stage compression within Simulink. These investigations led into the study of DRFMs, which use many of the same design requirements and modulation techniques. The only difference was that DRFMs require an additional microprocessor to ensure false targets or images are injected into a return signal. The first DRFM designs focused purely on false target returns

whereas DRFM designed used today focus on the deception of high range resolution profiling radar and range Doppler imaging sensors.

Subsequently, the use of FPGAs was examined. This study began through a MathWorks sponsored course titled “DSP for FPGAs” in Natick, Massachusetts. It was here that the fundamentals of Xilinx Vivado, DSP regarding FPGA synthesis and implementation, and the overall impact of area, power, and timing were introduced. Afterwards, familiarity with the specific FPGA purchased for this thesis was required. Through this familiarity of the FPGA, many design features could be rectified without need of compiling or synthesis/implementation. This in turn saved much time, which in the world of FPGAs, is extremely scarce.

Finally, the process of modeling, synthesis, and implementation of each digital system was investigated. This included many hours of Simulink tutorials and phone calls to MathWorks to ensure that the MATLAB derived results matched precisely with that of Simulink. This proved to be a challenge due to Simulink’s reliance on power of twos and HDL specific blocks. Many concepts were used to overcome these requirements such as the periodic ambiguity function, which not only performed the required target detection but also was built in a way that greatly increased the speed at which the FPGA could process the bit streams. Once the models were complete, the synthesis and implementation within Vivado began. About eight runs per model were constructed. Four of the models are considered sub models of the overall three-stage compression model. The idea was to test the subcomponents and relate it to the overall scheme. The relationships were summarized within graphs and subsequently compared.

C. THESIS OUTLINE

In Chapter II, the LPI Radar concepts are discussed. This chapter delves into the requirements of LPI Radars, the specifics of how to generate LPI modulation, in addition to the RSNS-P4 Three-stage Compression. Chapter III discusses DRFM architecture as well as the need for augmentation in Electronic Attack (EA). Chapter IV discusses FPGA history and uses, digital signal processing (DSP) on FPGAs, and the specifics of the XILINX Virtex Ultrascale+ FPGA utilized during this research. The modeling, synthesis,

and implementation results are then analyzed and discussed within Chapter V. Chapter VI concludes and provides recommendations for follow-on research.

THIS PAGE INTENTIONALLY LEFT BLANK

II. LOW PROBABILITY OF INTERCEPT RADAR

A. REQUIREMENTS

The safest sensor on the modern battlefield is one that can perform surveillance and conduct operations without being identified as doing such. The most capable sensor that can operate in smoke, rain, day or night is the radio frequency sensor that transmits electromagnetic energy for target detection. If the radar transmits high-energy pulses, it can detect targets at long ranges. However, their high peak power, pulsed radiated output signals are easily intercepted by the adversary's non-cooperative intercept receiver. This often leads to them executing an immediate counter-attack using several options with the deadliest being an anti-radiation missile targeting the radar system and operators.

Arguably the most important development in radar system technology is the LPI radar that "uses a special emitted waveform intended to prevent the non-cooperative intercept receiver from intercepting the radiation" as discussed in [3]. The LPI radar emission is a continuous waveform (CW) with low power. Typical output power ranges from one to twenty milliwatts and is usually transmitted from a solid-state array. To perform target measurements, the CW carrier is periodically modulated with a bandwidth that depends on the range resolution. The range resolution also determines the bandwidth. A larger bandwidth can be achieved making it difficult for narrowband receivers to detect and intercept the signal. This technique is much like what is used in telecommunication and radio communications, which is called, spread spectrum. The idea is to deliberately spread the signal in the frequency domain causing it to increase in bandwidth while also lowering its overall peak power. If spread spectrum is used effectively, it can essentially hide the signal within noise making it difficult to intercept.

Another consideration for achieving LPI radar conditions also include using frequency variations that may propagate well for an intended target but cannot effectively be acknowledged by a passive or intercept receiver due to atmospheric absorption. Finally, an antenna design requirement for an LPI radar is to have ultra-low side lobes. The return signal to the radar is, for all intents and purposes, considered feint. Large side lobes would

cause enough interference to deafen the return signal defeating the overall purpose of the LPI radar.

As advancements continue with LPI radars, so come advancements to intercept receivers. Near-peer adversaries are increasingly interested in the ability to perform electronic attack on friendly radar or communication assets. “To See and Not Be Seen” is the first line of defense in countering these intentions [3].

B. GENERATING LPI WAVEFORMS

In [4], Paepolshiri states that

> since pure CW waveforms cannot resolve the target’s range, periodic modulation techniques are used, such as frequency modulated CW, frequency shift keying, noise modulation, PSK, as well as hybrids of these techniques.” Paepolshiri found that the initial action to developing CW radar systems using periodic modulation compression is deciding on the necessary range resolution. According to Paepolshiri, this in turn “sets the transmitted bandwidth of the waveform for the above techniques (except for frequency shift keying where the range resolution is dependent to the duration of each frequency).

Paepolshiri goes on to say that

> due to the advances in high-speed processing and direct digital synthesis modules, the use of PSK techniques in CW radar is highly advantageous. CW radars that transmit and receive PSK signals can result in LPI radar systems with small range resolution cells and are ideally suited for many sensor applications for situational awareness including minimum input minimum output (MIMO) configurations. [4]

Additionally, PSK allows for RSNS-P4 generation, as depicted in Figure 2, which is the modulated signal utilized within this research.

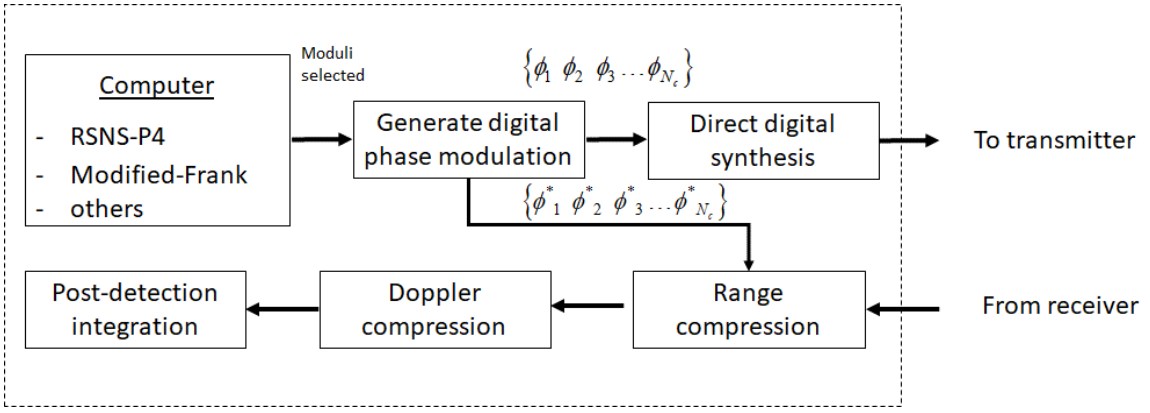

Figure 2. LPI Waveform Generation and Compression.

1. Polyphase Modulation (Polyphase Shift Keying)

The covert nature of polyphase modulation (or polyphase shift keying) is due to the code not being available to the receiver. "The unambiguous range of the radar is limited by the code period. That is, the unambiguous range is limited by the number of subcodes within the code period" [1]. The P4 polyphase code is unique due to it being a perfect code with zero periodic autocorrelation sidelobes. "The phase sequence of a P4 signal is given by

$$\phi_k = \frac{\pi}{N_c}(k-1)^2 - \pi(k-1) \tag{2.1}$$

where k is the subcode index and N_c is the number of subcodes within the code period" [1]. The phase distribution is both symmetrical and parabolic. It is these properties that allow the P4 modulation to lend itself nicely to being compatible with the RSNS described next.

2. Robust Symmetrical Number System (RSNS)

"The RSNS is a modular system consisting of $N \geq 2$ integer sequences with each sequence associated with a coprime modulus m_i." Details on the RSNS are given in [1]. An example for $N = 3$ is

$$\begin{bmatrix} 0\,0\,0\,1\,1\,1\,2\,2\,2 \ldots m_1\ m_1\ m_1\ \ldots\ 1\ \ 1\ \ 1\ \ 0\,0\,0\,1\,1\,1 \ldots \\ 1\,0\,0\,0\,1\,1\,1\,2\,2\,2\ \ldots\ \ldots\ m_2\ m_2\ m_2\ \ldots\ \ 1\ \ 1\,1\,0\,0\,0\,1 \ldots \\ 1\,1\,0\,0\,0\,1\,1\,1\,2\,2\ \ 2\ \ \ldots\ \ \ldots\ \ \ldots\ \ m_3\ m_3\ m_3 \ldots 1\,1\,1\,0\,0 \ldots \end{bmatrix}.$$

The above sequences "exhibit an integer Gray code property making the RSNS well suited for radar signal processing applications which can benefit from the inherent error detection and correction capability. To use the RSNS for radar signal processing, it is only necessary to know the greatest length of combined sequences without ambiguities, known as the dynamic range $\hat{M}$ and its position." As described in [1], combining the P4 with the RSNS (RSNS-P4) the phase relationship is

$$\phi_{m_i,k} = \left[\frac{\pi}{2m_i} \left(RS_{m_i,k} - m_i \right)^2 \right] - \frac{\pi}{2} m_i \ , \tag{2.2}$$

"where, $RS_{m_i,k}$ is the symmetrical residue, and $k \in \{1, 2, ..., N_{c_i}\}$ is the phase index. The code length is given as $N_{c_i} = NN_c = 2Nm_i$."

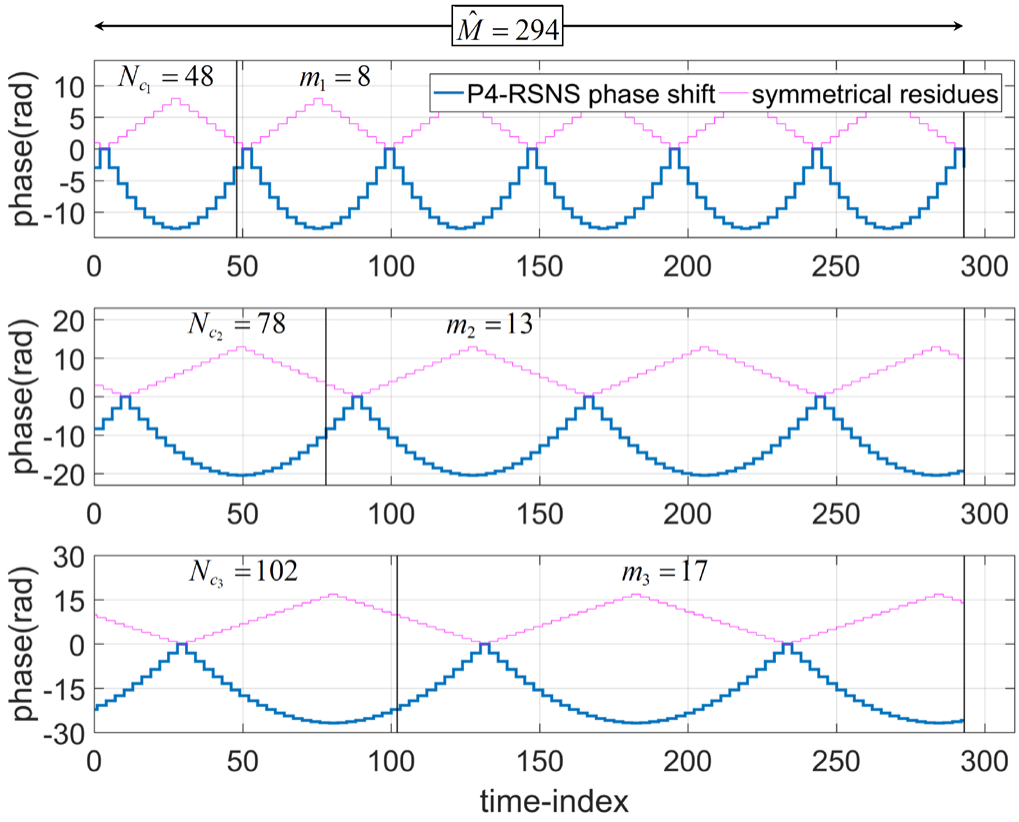

Figure 3. "P4-RSNS Channels and Symmetrical Residues." Source: [1].

C. RSNS-P4 THREE-STAGE COMPRESSION

The polyphase is transmitted, reflected from the target and upon receive, down converted and digitized, as shown in Figure 4. The digitized RSNS-P4 waveform is strobed into memory for receiver processing. The receive processing consists of range compression, Doppler processing and range-Doppler integration. The total processing gain PG_T is then $PG_T = PG_R + PG_D + PG_I$ which is determined from the required SNR_{Ri} input, the maximum detection range R_{Rmax}, and the "greatest of constant false alarm rate" (GOCFAR). The range compression determines PG_R and is determined from the maximum unambiguous detection range and range resolution while the "Doppler filtering process determines PG_D from the max target velocity at the unambiguous range during the process

sync time. The range-Doppler map is shown in Figure 5. The coherent integration of the range-Doppler" output is used to derive PG_I over the process sync time [1].

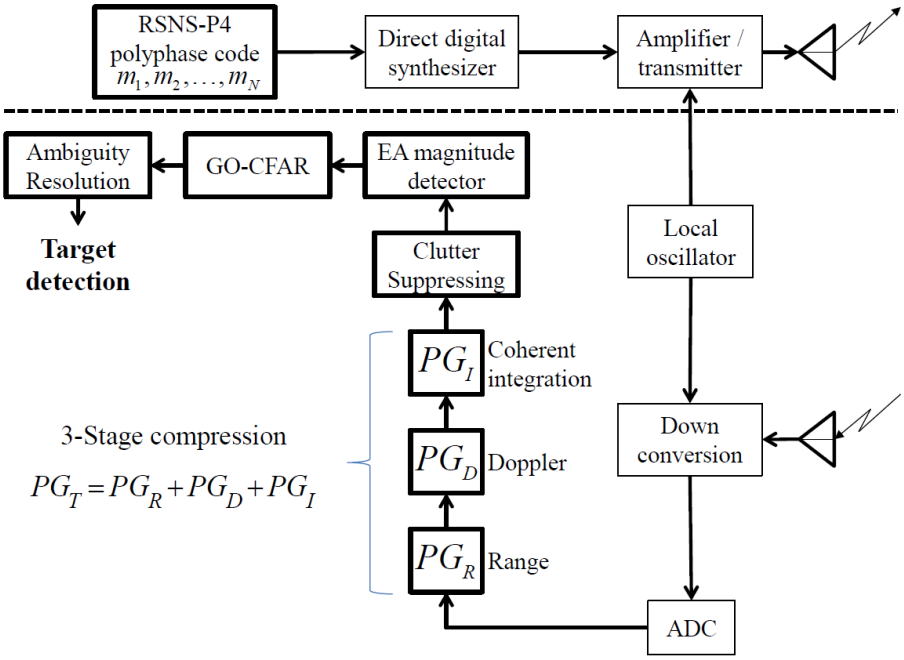

Figure 4. "RSNS-P4 CW Radar - Block Diagram Indicating the Processing Gain Steps." Source: [1].

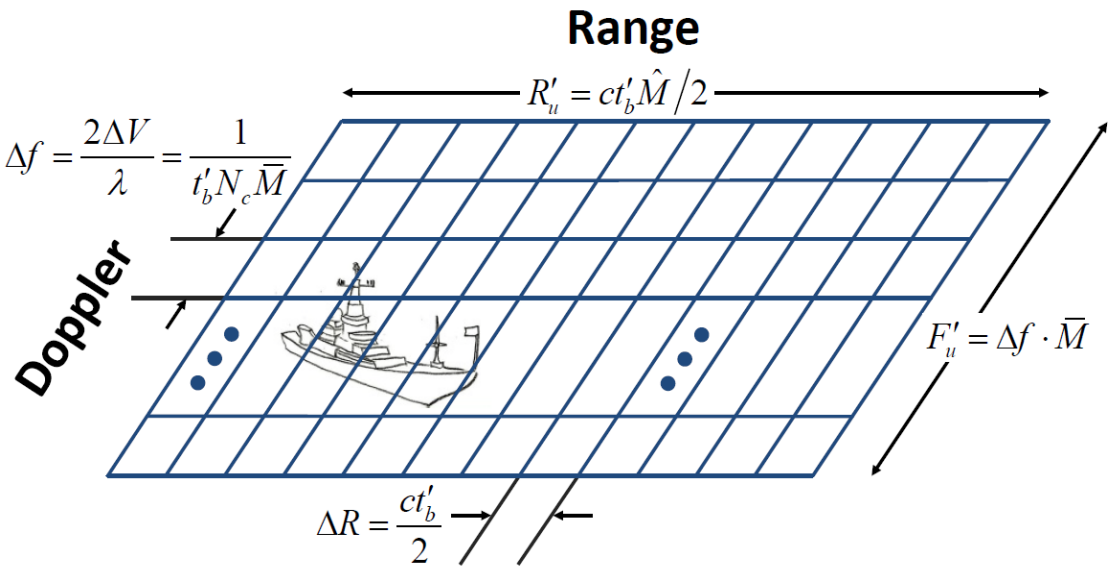

Figure 5. "RSNS-P4 CW Radar - Range-Doppler Detection Map." Source: [1].

1. Range Compression

The range compression multiplies the "fast Fourier transform (FFT) of the complex received signal by the FFT of a reference (transmitted RSNS-P4 phase) waveform and then taking the inverse fast Fourier transform (IFFT). Described in [1], the processing gain from the range compression is

$$PG_R = 10\log\left(N_{c_i}\right)." \tag{2.3}$$

2. Doppler Filtering

The Doppler filters for a particular range bin are calculated by executing the FFT algorithm on all the range bins collected during the number of code periods M_i, described in [1] as

$$M_i = 1/\left(t_b N_{c_i} \Delta f\right) \tag{2.4}$$

and is a function of the Doppler resolution Δf, the subcode width t_b and the code period for the modulus N_{c_i}. The estimated *PG* from the Doppler filtering is described in [1] as

$$PG_D = 10\log\left(M_i\right). \tag{2.5}$$

3. Integration

Coherent integration provides a means to enhance the SNR and increase the processing gain, which is described in [1] as

$$PG_I = 10\log\left(N_i\right) \tag{2.6}$$

where N_i is the number of maps that are averaged together coherently. The range-Doppler maps after the coherent integration are shown in Figure 6 “for all three RSNS-P4 channels or moduli (note the different range scales). Targets that have significant velocity separation from the clutter can be detected” [1].

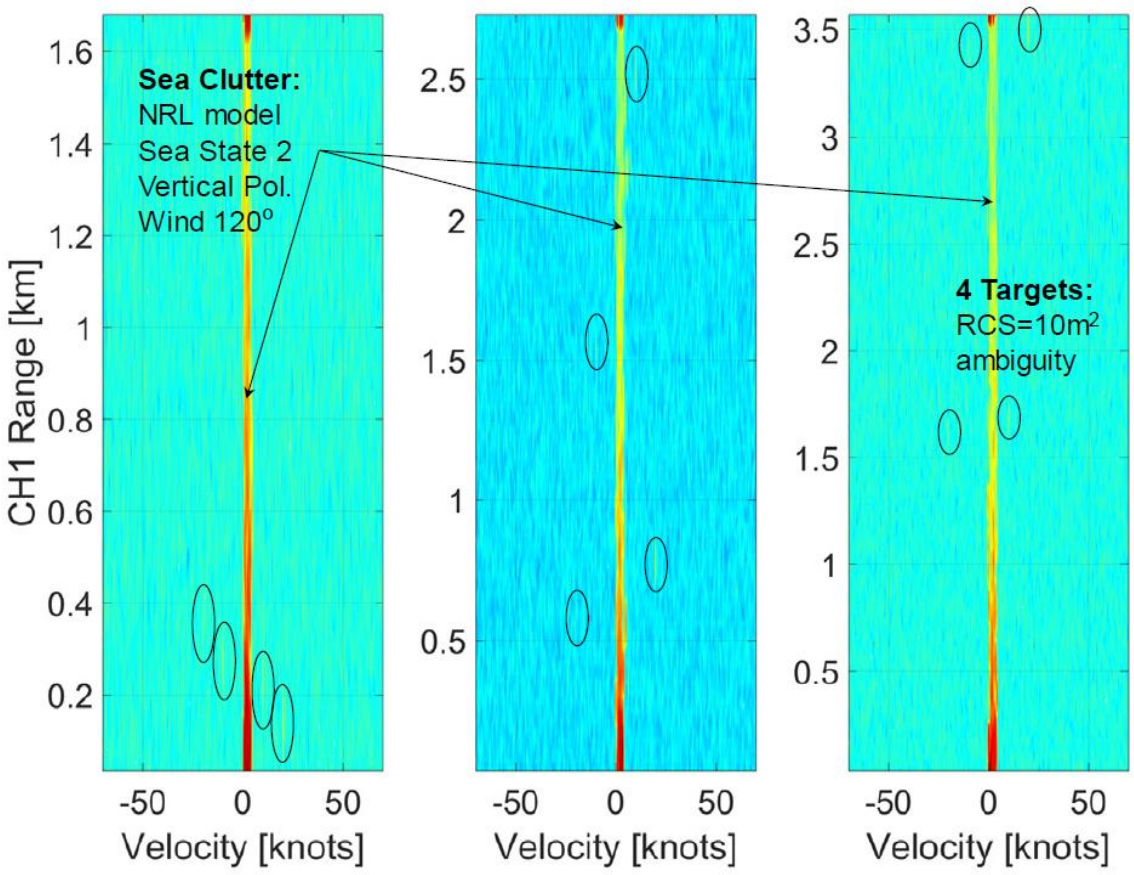

Figure 6. "Range-Velocity Maps for Each of the Three Channels. Target Moves at Relative Speeds Sufficiently Different from the Sea Clutter Spectra." Source: [1].

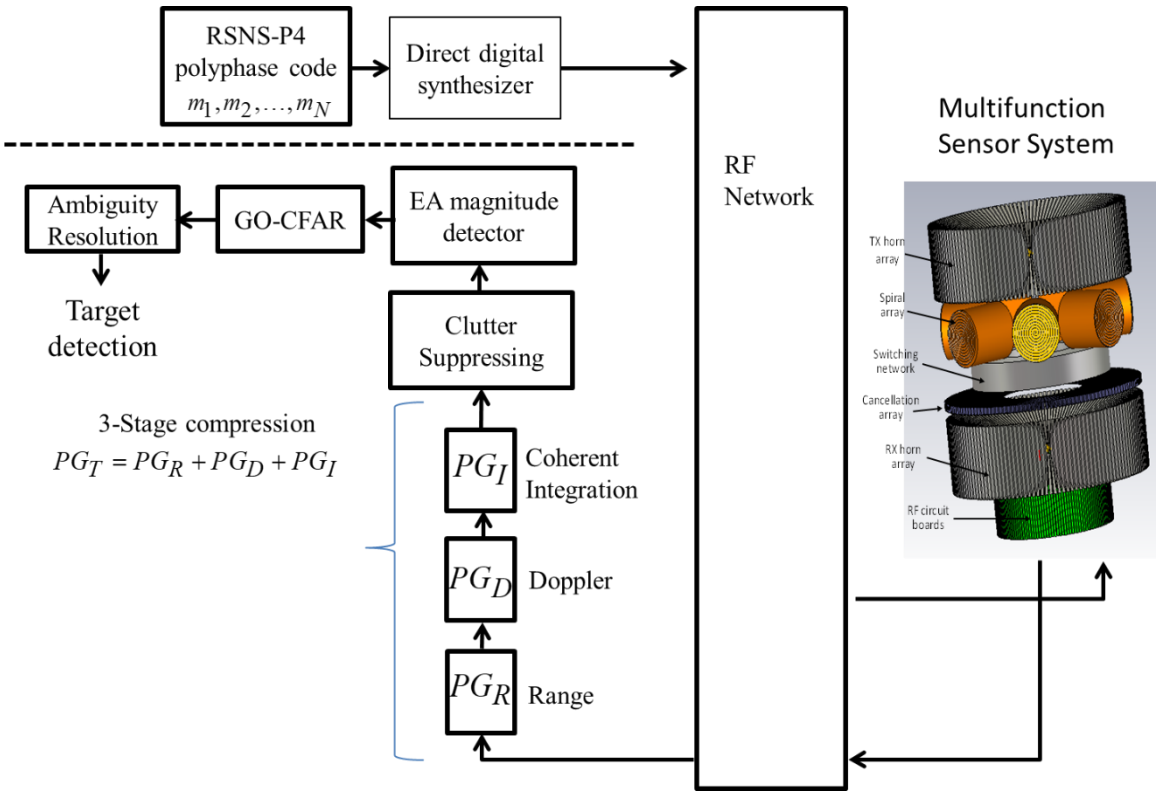

Figure 7. LPI Signaling Environment and Integration with the Multifunction Sensor System.

D. CHAPTER SUMMARY

In this chapter, the requirements for LPI radar were presented. PSK modulation as an ideal modulation for LPI radars, the concept of RSNS, and advantages of using RSNS-P4 were also presented. Additionally, the RSNS-P4 three-stage compression was presented which broke down the properties of range compression, Doppler filtering, and coherent integration. The next chapter presents the concept of Digital Radio Frequency Memory and its continuing evolution to deceive current and future radar signatures.

III. DRFM

A. DRFM ARCHITECTURE

The DRFM provides the ability to capture radiated emissions and generate precise, coherent replicas, making them important in applications such as signal jamming, deception of covert communications, SIGINT operations, decoys, radar transmitters, simulations, and test equipment as discussed in [5]. A block diagram of single sideband DRFM is shown in Figure 8. At the receive antenna, a bandpass filter is used to pass only the signals of interest. A Local Oscillator (LO) is used to tune the DRFM to intercept the desired signal in the down conversion process. A Low Pass Filter (LPF) removes the components above Nyquist (anti-aliasing) along with unwanted mixer products. This configuration gives good rejection of spurious signals while retaining all the advantages of a conventional superheterodyne. At the output of the LPF, the signal is digitized by an ADC with resolution typically on the order of one to eight bits depending on the DRFM throughput. The higher the resolution, the slower the conversion process. After digitization, the samples are strobed into memory. High-speed, dual-ported memory is often used so the stored digital signal is captured and replayed simultaneously through memory control. "Dual-ported memory usually requires a serial-to-parallel and parallel-to-serial circuitry to achieve the necessary data-rate conversion to match the dual-port memory's input/output bandwidth" [5]. With the use of multi-ported memory, recording and multiple replays can occur simultaneously. The retrieved digital signal is strobed from memory to a DAC to reconstitute the signal back into an analog waveform. After lowpass filtering, the baseband signal is mixed with the LO to reconstruct the radio frequency (RF) version (typically a single sideband modulator). A band-pass filter (BPF) at the output serves to transmit only the desired frequencies [5].

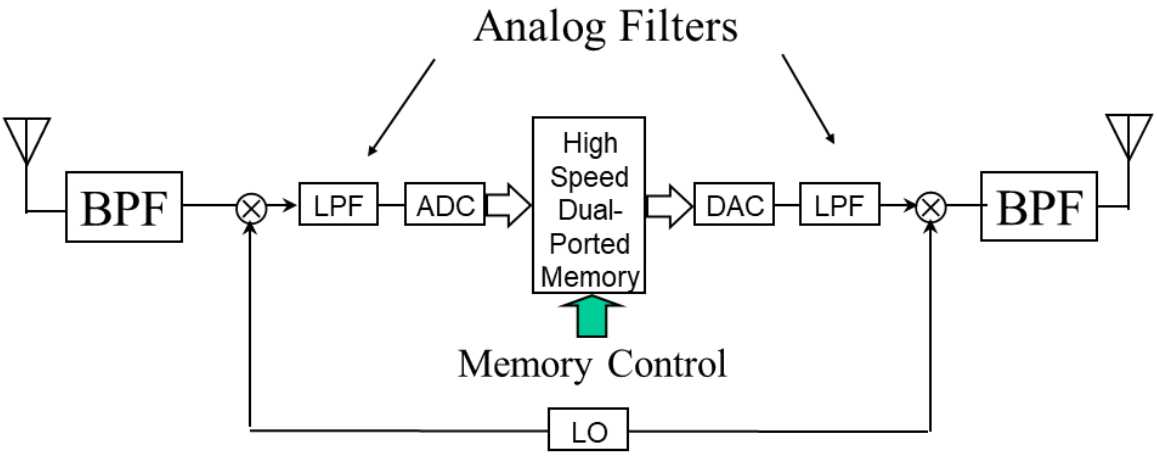

Figure 8. Block Diagram of a Single Sideband DRFM. Source: [5].

Figure 9 shows a block diagram of a double sideband DRFM. As discussed by Dr. Pace in [5], this architecture is similar to Figure 8 except that the phase of the DRFM signal is retained throughout the digitization process using both an I and Q channel. These are produced by the signals Q intermediate frequency (IF) modulator at the input. The Q IF modulator also down converts the input RF signal. Also shown in this configuration is the capability to retrieve the stored digital signal for further, more complicated signal processing using FPGAs or DSPs. This additional processing power can be used to create a variety of complex waveforms. For example, digital image synthesis for inverse synthetic aperture radar (ISAR) counter targeting applications require a level of signal processing that cannot be accomplished with simple memory recall and bit manipulation. The synthesis of the image requires focusing the Doppler frequency at each range bin, and amplitude modulating the output correctly such that the proper image is constructed [5].

Dr. Pace goes on to say that the double sideband (DSB) architecture requires the lowest sampling rate and has the capability of retaining the phase of the signal. The single sideband (SSB) is a less complex architecture; however, a higher sampling rate is required, and the phase of the intercepted signal is not retained. Due to this higher sampling rate, the bit resolution is less [5].

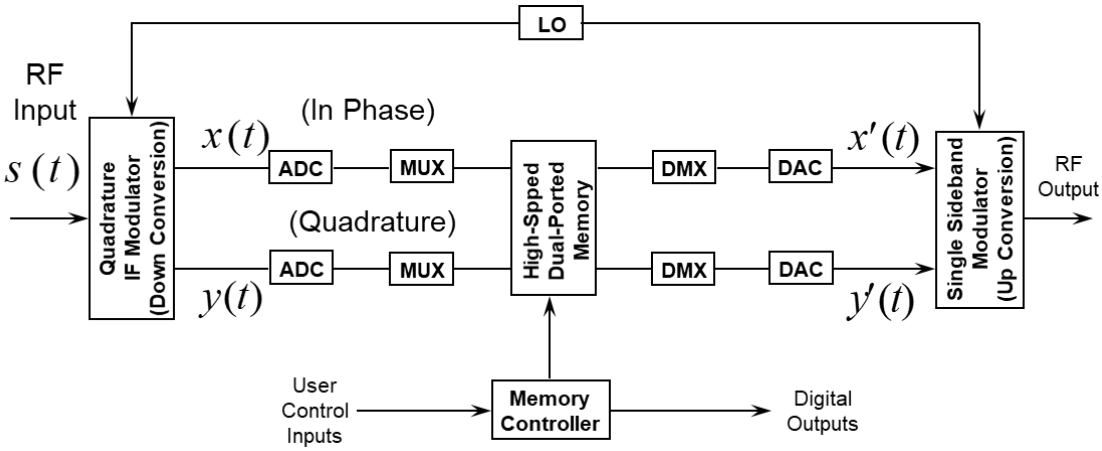

Figure 9. Block Diagram of a Double Sideband DRFM. Source: [5].

Dr. Pace also notes that to process the signals of interest, the DRFM uses a memory controller. The memory controller simply provides the memory address and control signals for the signal storage and recall operations. Several user controls are also available. The storage enable function is used to designate the pulse to be stored. The store address designates where the leading edge of the pulse is to be stored. The recall initiates a recall cycle at the next word clock time. The recall output triggers must be synchronized with the DRFM clock for coherent output. The recall address is the address where recall begins, and the delay time is the throughput delay impressed upon the stored signal [5].

Lastly, Dr. Pace determines that the development of a common DRFM kernel interface specification has been of interest in the Electronic Warfare (EW) community. The purpose of this specification is to define a common narrowband DRFM kernel that meets the jamming requirements of the Department of Defense. This kernel can be utilized in existing and future countermeasure systems. The interface architecture provides potential for efficient upgrades and simplifies DRFM-based system development, leading to a cost savings during development [5].

B. NEED FOR AUGMENTATION IN ELECTRONIC ATTACK SYSTEMS

In [6], the authors propose that “the need for coherent countering of ISAR imaging sensors remains a high priority for many electronic warfare systems.” They state that with the development of an “all-digital image synthesizer (DIS), multiple false-target images

can be generated from a series of intercepted ISAR chirp pulses to provide a novel counter targeting and counter lock-on capability." The authors comment that the DIS "can be also be deployed for Suppression of Enemy Air Defense and any operation that encounters interrogating ISAR imaging sensors." The authors then conclude that the "device can be deployed on aircraft, ships, unmanned air or surface vehicles to provide a superior imaging decoy and deception capability" [6].

In [7], Pak Ang examines and shows the concept of embedding an I/Q Phase Converter and DIS into a DRFM in Figure 10. In Ang's thesis, this DRFM can intercept and store RF waveforms as well as retransmit them subsequently. His thesis adds that upon capturing an ISAR waveform, the DRFM uses a local oscillator to down convert the signal to an intermediate frequency; and that these signals are separated into I and Q components and then digitized by the ADCs into digital samples that are stored in a high-speed memory. An I/Q phase converter extracts phase information from the digitized waveforms to generate phase samples for the Digital Imaging Synthesizer (DIS) to process. Ang continues that after modulation by the DIS is complete, the DRFM converts the processed signal back into an analog form. Finally, the DRFM transmits the analog signal back to the ISAR [7].

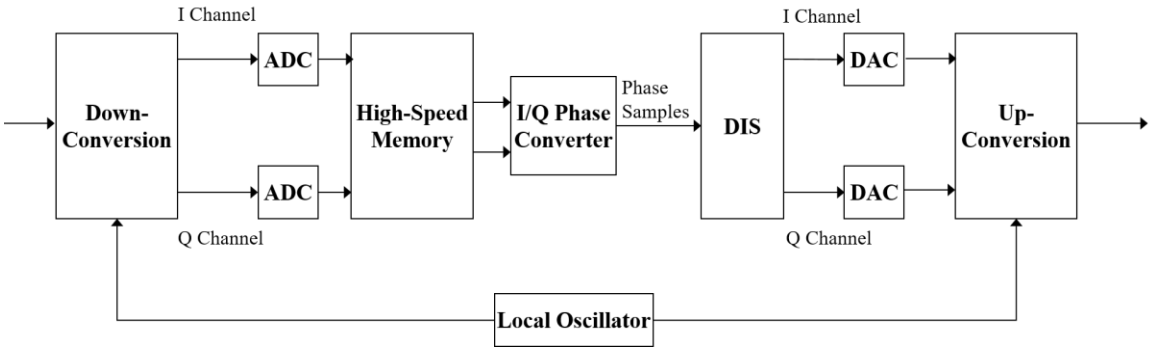

Figure 10. Simplified Block Diagram of the DRFM Integrated with the I/Q Phase Converter and DIS. Source: [7].

In [6], the authors remark that the "position of the false target in range can be controlled by delaying in time, the read-out samples going to the image synthesizer." They state that the

> image synthesizer performs the complex modulations to synthesize the temporal lengthening and amplitude modulation due to the many recessed and reflective surfaces of the desired false target and generates a realistic Doppler profile for each surface. The FPGA contains a parallel array of identical digital modulators with one modulator for each false target range bin. That is, each modulator synthesizes the part of the overall image that is within the false-target range bin associated with that modulator. [6]

The authors state further that "each complex output pulse *I(m,n)* is the superposition of N_r copies of the intercepted pulse, each delayed with respect to one another by the delay within the modulator, scaled differently by the gains $2^{g(r,n)}$ and phase rotated by $\phi_{inc}(r,n)$ described in [6] as

$$I(m,n) = \sum_{r=0}^{N_r-1} 2^{g(r,n)} e^{j\phi(m-r,n)+\phi_{inc}(r,n)} \tag{3.1}$$

where *m* represents the sample number within the chirp pulse, *n* is the pulse number index, and *r* represents the range bin modulator index. The target extent, amplitude, and target motion are controlled by the gain and phase increment coefficients applied to the FPGA" [6].

C. CHAPTER SUMMARY

In this chapter, the current and proposed DRFM processes were presented to include the need for an augmentation update within electronic attack systems. This augmentation would require an I/Q Converter and DIS to be integrated into the current DRFM architecture. This addition would ensure that not only traditional radars are deceived but ISAR as well. The next chapter presents an overview of FPGAs, DSP utilizing FPGAs, the specific Xilinx FPGA used during this research, and the simulation process used to recreate the results in the follow-on chapter.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. FIELD PROGRAMMABLE GATE ARRAY

A. OVERVIEW OF THE FPGA

In [8], the training course states that FPGAs are a fully reconfigurable resource: the implemented functionality is defined and programmed after manufacture, and this process can be repeated indefinitely. The programmable resources of the FPGA are configured into a desired digital circuit by downloading the user created design as a bitstream. Several steps are involved in creating, verifying, and preparing a design for download to an FPGA [8]. Table 1 provides a list of related technologies like FPGAs.

Historically, Application Specific Integrated Circuits (ASICs) have been faster, extra energy efficient, and typically achieved much more performance than their FPGA counterparts. In [9], the authors observed that implemented FPGA designs "need an average of 40 times as much area, draw 12 times as much dynamic power, and run at one third the speed of corresponding ASIC implementations." In [9], the training course determines that

> much more recently, FPGAs including the Xilinx Virtex-7 or maybe the Altera Stratix-5 came to rival corresponding ASIC and application-specific standard parts (ASSP) through the process of delivering considerably minimal power consumption, improved speeds, lower overall production costs, less implementation utilization, and improved options for 'on-the-fly' re-configuration. [9]

Previously, implemented designs requiring and incorporating six to ten ASICs can now produce the same results using a single FPGA [8].

In [8], the training course illustrates that

> advantages of FPGAs include the ability to reprogram in the field to fix bugs and may include a shorter time to market and lower non-recurring engineering costs. Vendors can also take a middle road by developing their hardware on ordinary FPGAs but manufacture their final version as an ASIC so that it can no longer be modified after the design has been committed. [8]

Due to ASIC complexity, revenue losses driving higher production costs, and slow time-to-market trends, FPGAs have become a much-needed solution for higher-volume

applications. Mentioned later in this chapter, many FPGAs can perform partial reconfiguration, which allows one section of the unit to be reprogrammed while various other regions keep on running.

Table 1. Related Technologies to FPGAs. Source: [8].

Application Specific Integrated Circuits (ASICs)	Unlike FPGAs, the function of an ASIC is defined at manufacture, and it cannot be reconfigured. However, ASICs are generally smaller, lower power and when manufactured in high volume, cheaper to produce. The time for design and manufacture of ASICs is longer than for FPGAs, which is a consideration where fast time-to-market is important.
Digital Signal Processors (DSPs)	DSPs historically had one processing engine of fixed wordlength, although modern devices may have several cores. In comparison, FPGAs have parallel processing capabilities, and the designer is not restricted to pre-specified wordlengths. As DSPs and FPGAs offer such different capabilities, they are often used for different tasks within a larger system.
General Purpose Processors (GPPs)	While DSP processors are optimized for fast arithmetic, multiply-accumulate type operations, GPPs have the flexibility to deal with a variety of applications but are not suited to the fast arithmetic demanded by DSP.
Processor Arrays / Sea of Processors	This type of device exhibits some of the characteristics of an FPGA (parallelism, interconnects) with the processor architecture of a DSP. One of the challenges associated with this type of device is efficiently programming them.
Complex Programmable Logic Devices (CPLDs)	CPLDs are like FPGAs in the sense that they are parallel and reconfigurable, but they are smaller and far less sophisticated. CPLDs have very low power consumption and are suited to "glue logic" type applications.
Structured ASIC	Structured ASICs offer a compromise between ASICs and FPGAs, and there are several slightly different architectures that fall into this category. The benefits of lower power, and cheaper high-volume production than FPGAs.
FPGA-hardening services	The major FPGA companies offer a path to high-volume production based on an FPGA prototype. These are appropriate when the full reprogrammability of an FPGA is not required, are cheaper than standard FPGAs, and faster to produce than ASICs.

1. History

In [10], the article states that the FPGA industry began from “programmable read-only memory (PROM) and programmable logic devices (PLDs).” Both PROMs and PLDs

had the ability to be uniformly and “mass-programmed either in a factory or in the field by the user (field programmable).” The programmable logic of the initial FPGA design was “hard-wired between logic gates” [10].

The article adds that in the late 1980s, Steve Casselman proposed to the Naval Surface Warfare Center that a personal computer might be created that could apply 600,000 reprogrammable array gates. The system was funded as well as in 1992 a patent was given because of the program eventually naming Casselman as “the expert in the field of virtual computing” [10].

As found in [11] and [12], David W. Page and LuVerne R. Peterson were awarded with patents in 1985. These patents evolved into several of the “industry's foundational principles for programmable logic blocks, arrays, and gates.”

In [13], Ron Wilson claims Altera was founded in 1983, In 1984, Altera delivered the EP300, which was the company’s first product. This reprogrammable logic device “offered erasability by shining a UV lamp through the window above the die.” The article states that the logic device maintained a quartz window allowing the UV lamp to penetrate the die, erasing the “EPROM cells that held the device configuration” [13].

In [14] and [15], the articles state that Xilinx co-founders Ross Freeman and Bernard Vonderschmitt invented the XC2064 in 1985, which was the first commercially viable FPGA. The sources explain that the XC2064 began a new technology and marketable demand using “programmable gates and interconnects between gates.” The XC2064 had “64 configurable logic blocks (CLBs), with two three-input lookup tables (LUTs).” As explained in [16] and [17], after two decades, “Freeman was entered into the National Inventors Hall of Fame for his invention.”

Unchallenged, Altera and Xilinx expanded through the latter part of the 1980s to the mid-1990s. At this point, competition began to form within the technological market causing Altera and Xilinx shares to decrease rapidly. In [15], the article explains that by 1993, “Actel (now Microsemi) was serving about 18 percent of the market.” In [18], the article reports that by 2013, “Altera (31 percent), Actel (10 percent) and Xilinx (36 percent) together represented approximately 77 percent of the FPGA market.”

The ability to mass produce as well as the sophistication of circuitry within FPGAs made significant leaps during the 1990s. The overall use of FPGAs transitioned from being primarily used by telecommunications and networking to be utilized in everyday consumer, automotive, and industrial applications by the early 2000s.

2. Current Uses

A trend has formed within the past decade to develop a "complete system on a programmable chip" as seen on the Xilinx website in [19]. This idea of having a system on a chip (SoC) involves the combination and assembling of logic blocks and interconnects of previous generations of FPGAs as well as fixed microprocessors and associated peripherals. The website states that the Zynq-7000 All Programmable SoC is an example of such hybrid technology and is depicted in Figure 11. The website adds that this chip "includes a 1.0 GHz dual-core ARM Cortex-A9 MPCore processor embedded within the FPGA's logic fabric." The website concludes that his hybrid technology uses specific high-performance processors and pair them "with programmable logic architectures or multi-channel ADC and DAC analog peripherals to their flash-based FPGA fabric" [19].

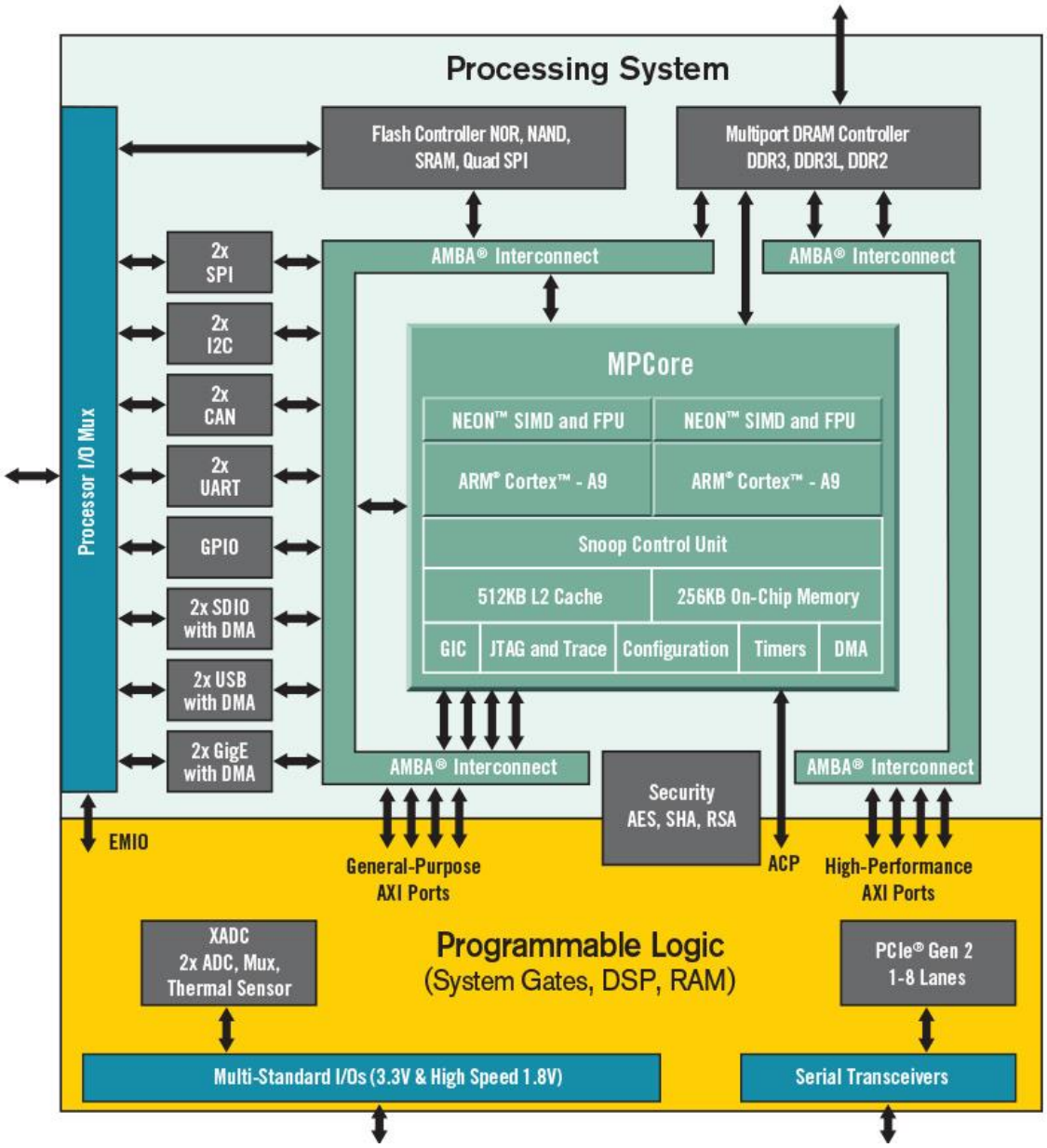

Figure 11. "A Xilinx Zynq-7000 All Programmable System on a Chip." Source: [19].

Implemented within the FPGA logic are soft processor cores, which provide a hard-macro processor alternate approach. Examples of such technology with soft processor cores are the Nios II, MicroBlaze and Mico32. Reconfigurable computing or systems is the idea of programming current-day FPGAs prior to or during use which allows CPUs to meet any task by reconfiguring themselves. Additionally, non-FPGA technology is emerging.

Microprocessors, such as the Stretch S5000, are software configurable and maintain FPGA-like programmable cores, which, in turn, provide an array of processor cores all within the same chip.

In [20], the article states that "companies like Microsoft have started to use FPGAs to accelerate high-performance, computationally intensive systems (like the data centers that operate their Bing search engine), due to the performance per watt advantage FPGAs deliver." Table 2 provides a list of current FPGA Applications per specified category.

Table 2. Current FPGA Applications. Adapted from [8].

Aerospace and Defense	Audio	Broadcast	Automotive
Avionics/DO-254	Connectivity Solutions	Real-Time Video Engine	High Resolution Video
Communications	Portable Electronics	EdgeQAM	Image Processing
Missiles & Munitions	Software-Defined Radio	Encoders	Vehicle Networking and Connectivity
Secure Solutions	Digital Signal Processing (DSP)	Displays	Automotive Infotainment
Space	Speech Recognition	Switches and Routers	

Medical	High Performance Computing	Industrial	Integrated Circuit Design
Ultrasound	Servers	Industrial Imaging	ASIC Prototyping
CT Scan	Super Computers	Industrial Networking	Computer Hardware Emulation
MRI	SIGINT Systems	Motor Control	
X-ray	High-end RADARs		
PET	High-end Beam Forming Systems		
Surgical Systems	Data Mining Systems		

Data Center	Consumer Electronics	Security	Wired Communications
Servers	Digital Displays	Industrial Imaging	Optical Transport Networks
Security	Digital Cameras	Secure Solutions	Network Processing
Hardware security module	Multi-function Printers	Hardware security module	Connectivity Interfaces
Routers	Portable Electronics	Image Processing	
Switches	Set-top Boxes		
Gateways	Flash Cartridges		
Load Balancing			

Video & Image Processing	Scientific Instruments	Wireless Communications	Bioinformatics
High Resolution Video	Lock-in amplifiers	Baseband	
Video Over IP Gateway	Boxcar averagers	Connectivity Interfaces	
Digital Displays	Phase-locked loops	Mobile Backhaul	
Industrial Imaging	Radio Astronomy	Radio	
Computer Vision			

B. DIGITAL SIGNAL PROCESSING ON A FPGA

In [8], the training course states that from a DSP perspective, the resources available on FPGAs have evolved significantly over the last 25 years. Early FPGAs comprised a general-purpose array of CLBs and routing resources, surrounded by IOBs at the edge of the chip. Over time, integrated memory blocks and fast arithmetic slices have been introduced. The functionality of these arithmetic components has increased too. Initially, embedded multipliers were provided, and now DSP engineers have access to an integrated tile containing an adder, multiplier and accumulator. In terms of system integration, embedded processors and communications interfaces have become standard. Naturally, the speed and size of FPGAs has increased also, as have the sophistication of clock management resources [8].

1. Introduction to DSP

In the 1980s, the arrival of microprocessors such as Intel 8086 and Rockwell 6502 triggered the so-called Microprocessor Revolution, as stated in [8]. This resulted in commonly accessible and cost-effective computer equipment. Besides several early 1980s home computer systems as well as video gaming machines, the primary development was the IBMPC in 1980 as well as the Macintosh in 1984. This proliferation of computer systems in the workplace, in the market, and the house, entirely altered business processes along with the way info is stored as well as processed. By the 1990s,

> multimedia PC was essentially enabled by DSP technology, devices, and algorithms. Additionally, the processing power of DSP (micro-) processors was increased by an order of magnitude with a decrease in price. The digital reliability, repeatability, and programmability of DSP has widely displaced analog systems in both consumer and industrial markets. [8]

Table 3 is a list of the individual DSP technology and components of early 1990 machines and Table 4 is a list of current DSP Applications per specified category.

Table 3. 1990s Multimedia PC Enabled by DSP Technologies. Source: [8].

Windows 95 OS	Facilities for speech coding / compression (ADPCM, LPC, GSM etc.). Digital filtering, FFT, correlation facilities all within Microsoft Excel spreadsheet.
Video Acquisition Card	Fast ADC / DAC technology and DSP video coding algorithms for MPEG etc. FFTs, DCTs, sub band coding etc.
Disk Drive	Most modern disk Speech synthesis, speech recognition. Drives now include a DSP processor for control purposes.
Teleconferencing Card	Enabled by DSP coding for audio and video, and adaptive acoustic echo cancellation.
Sound-card	16-bit sound technology sampling up to 48kHz. Sigma delta technology allows low cost implementation; DSP processor implements algorithms for decimation, interpolation, mixing, filtering, coding etc.
Speech Processing	DSP enabled digital recording answering machine.
FAX-modem	Enabled by adaptive signal processing algorithms for echo cancellation, data equalization

Furthermore, a distinction must be made between data processing (DP) and DSP according to the training course. DP and DSP are both

> ideally performed by high speed computers which have very fast numerical capabilities. DP is the arithmetic processing of (sampled) stored integer numerical quantities (accounts, salary spreadsheets and so on); fast processing of data is desirable but not essential. DSP is concerned with the arithmetic processing of numerical representations of real world analog quantities. Real time performance is necessary, such that processed outputs are produced as fast as input data is available. For both, a suitable sampling rate must be chosen (not too high and not too low). [8]

In short, DSP means real-time arithmetic operations and DP means non-real time arithmetic operations [8].

Additionally, the training course provides the following DSP Strategies:

a. Linear Filtering

Removing high frequency background noise from speech. Linear filtering strategies can be used in any application where it is known that two signals can be discriminated by the frequency bands they occupy. [8]

b. Signal Transforms

Signal component analysis, signal detection etc. Transforming a signal into a different domain often allows a signal to be more conveniently analyzed and viewed. For example, transformation into the s-domain (Laplace) allows more straightforward mathematical manipulation. Transforming into the frequency domain allows the frequency variation of a signal to be seen more easily than in the time domain [8]

c. Non-linear Signal Enhancement / Filtering

Removing of impulse noise by median / order type filtering. Some signals may often benefit from non-linear filtering. A well-known audio non-linear filter is for impulsive noise, whereby a signal is contaminated by impulses. Given that an impulse essentially contains all frequencies, frequency or phase discriminating filter is not of use. Hence, a median filter may be used whereby the N most recent samples are ordered and the median value is chosen. Hence, very large magnitude outliers are likely to be ignored if the duration of the N samples is somewhat longer than the duration of the impulsive noise [8].

d. Signal Analysis / Interpretation / Classification

Designed for ECGs, speech recognition, image recognition, etc. The aim here is to compare known 'patterns' with input signals to recognize the input signal and output some parameterized information. [8]

e. Compression / Coding:

Hi-fidelity audio (Minidisc), mobile telecoms, videoconferencing, ECG signal compression and so on. Compression is one of the most important areas within the audio and telecommunication business at present. Compressed formats such as MP3 are on the rise in the high-fidelity audio market. For telecommunications speech requires to be coded into as small a bandwidth as possible, but while maintaining sufficient signal quality. With each new mobile generation, the availability of more DSP processing power allows the bit rate to reduce but while still maintaining good intelligible quality. [8]

Table 4. DSP Applications. Source: [8].

Telecommunications	Mobile (GSM, CDMA, IS-95); Digital / Video Telephony; Data Modems; ADSL
Digital Audio	CD; CDI; DAT; DCC; Surround Sound, MPEG; MiniDisc; Dolby Prologic
Digital Video/Imaging	High Definition TV (HDTV); MPEG; Medical Imaging; JPEG; DVD
Speech Based Systems	Speech Recognition; Speech Coding / Compression; Speech Synthesis
Multimedia	PC FAX / Modem / Graphics / Audio; Teleconferencing; Software Radio
Biomedical Systems	ECG Electrocardiograph; EEG - Electroencephalograph; Hearing Aids
Industrial	Motor Control; Disk Drives; Process Controllers; Noise Cancellation
Defense	Guidance Systems; Sonar; Radar; Secure Communications
Automotive	GPS Navigation; Engine Management; Digital Comms / Audio Systems

2. Introduction to DSP for FPGAs

In [8], the training course points out that a DSP algorithm or problem is often specified in terms of its "multiply and accumulates/add" (MAC) requirements. When comparing two algorithms, if both perform the same job but one with less MACs, then the other would clearly would be "cheaper" and the best choice. However, this implies some assumptions: one is that the required MACs are the same. In traditional DSP processor-based situations, it is likely that a 16-bit device that will be processing 16-bit inputs will be using a 16-bit digital filter coefficient. With FPGAs, this constraint is removed due to the ability to use as many, or as few, bits as are required. Therefore, optimization and scheduled DSP algorithms can be chosen and implemented in completely different ways [8].

The training course states further that standards are constantly evolving which means that devices, like FPGAs, that can be reconfigured and upgraded become seamless base-stations, access points, etc. DSP enables many aspects of everyday life but even though they all have different specific requirements, usually the low-level processing is similar i.e., filters, transforms, sine wave synthesis, adaptive filtering, and sampling rate changes [8].

The training course finds that DSP is built upon arithmetic; therefore, special attention must be given to the implementation of arithmetic operations. Most important are addition and multiplication in which the arithmetic wordlength must increase to prevent arithmetic overflow from input to output. FPGAs allow wordlengths to be specified with

complete flexibility, and without any computational overhead. Before FPGAs arrived, DSP circuits could be constructed but with less flexibility such as integrated circuits and gate arrays but had fixed wordlengths. This also caused operations with shorter wordlengths to be just as computationally expensive due to requiring one execution. DSPs are still useful, particularly for sequential processing, and often complement FPGAs in a DSP system, especially when FPGAs allow the user to choose exactly the required wordlength [8].

3. DSP-FPGA Design Fundamentals

a. General FPGA Architecture

In [8], the training course describes that most of the FPGA area is logic fabric, which is the building blocks of combinational and sequential elements connected by local and long-distance wires. Input / Output Blocks (IOBs) allow signals to be routed into and out of the FPGA. The logic fabric is made up from CLBs and signals are routed between CLBs using programmable interconnects. The programmable aspect of the interconnections is partly realized by the Switch Matrices located beside each CLB. CLBs contain slices that contain LUTs, Flip-Flops (FFs), and a few logic gates [8]. The functions implemented by these resources are programmable, as depicted by the design process, as depicted in Figure 12.

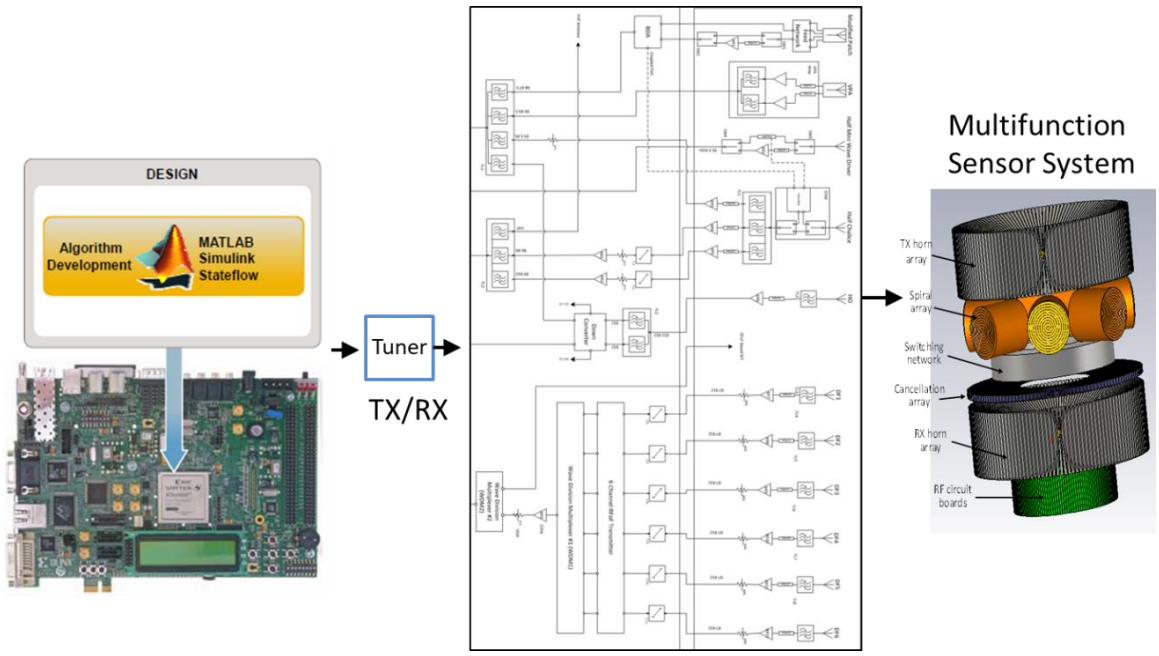

Figure 12. Model-Design Environment that Provides an Integrated Workflow for Faster Waveform Engineering.

The training course agrees that logic units differ in size, composition, and name. However, in all cases, their Logic Blocks include both combinational logic and registers, and routing resources are required for connecting blocks together. LUTs can be utilized in four modes: to implement a combinatorial logic function, as Read Only Memory (ROM), as Random-Access Memory (RAM), or as shift registers [8].

In more recent FPGAs, the training course states that the grouping of slices into CLBs has changed to 2 slices per CLB. From a high-level perspective, a slice from one of the 4-slice CLBs consists of 2 LUTs and 2 FFs. However, in newer devices, the grouping of LUTs and FFs into slices differs. As mentioned previously, LUTs can also be configured as RAM, i.e., memory that can be written to, as well as read from. LUTs can also act as shift registers (SRLs). A 4-input LUT can provide up to 16 bits of memory and implement up to length 16 shift registers, whereas a 6-input LUT can provide up to 64 bits of memory and implement up to length 32 shift registers. When building shift registers or RAMs using distributed resources (i.e., LUTs), several LUTs can be combined as necessary to build

larger sizes. Likewise, RAMs capable of storing more bits can be created by combining LUTs (although for very large memories, Block RAMs are a more appropriate choice) [8].

As well as implementing 16 or 32 (i.e., maximum length) shift registers, they can also be programmed to implement any length up to that as stated by the training course. The slice FF at the output of the LUT can be used to add another one-clock cycle delay, which is beneficial for timing performance. Hence, with one LUT-FF pair, a delay of up to 17 clock cycles can be realized. In other types of FPGAs, with SRL32s and 2 FFs per LUT, up to 34 cycles can be implemented [8].

The training course goes on to say that IOBs feature a pad which allows a signal on the FPGA to be connected to an external signal (input/output direction is defined by multiplexers). IOB FFs may optionally register signals as they enter or depart the chip; registering inputs and outputs is generally recommended and beneficial for timing performance. Ultimately, one IOB is required for each 1-bit signal [8].

The training course determines that each FPGA device is normally associated with two or more packages. The choice of package defines which of the IOB pads on the FPGA are physically bonded to pins on the package, and therefore which IOBs are available for use. Therefore, some of the IOBs may not be connected to a pin. Some of the IOBs may be connected to pins which are used for reserved purposes (like power provision, programming etc.), and hence are not available to the user. Depending on the package chosen, many fewer usable pins may be available than the maximum number the FPGA device would support. To clarify, the term bonded IOBs is given in resource utilization reports and unbonded IOBs are not available when using that particular package [8].

b. Memory

In [8], the training course reveals that columns of dedicated memories (Block RAMs), and high-speed DSP functional blocks (DSP48x slices) are also integrated into the array. The spacing of Block RAMs and DSP48s is sparser than CLBs. Block RAMs are located in columns close to the edges of the FPGA (to buffer input and output data from the chip) and next to DSP blocks (for enhanced DSP performance). Additionally, Block

RAMs can be combined to form larger RAMs or be subdivided into smaller ones while also being implemented as first-in first-out buffers [8].

Additionally, these arrangements allow for DSP48 to become dedicated high-speed arithmetic blocks and Block RAMs to become dedicated high-speed memory. DSP48 comprise a multiplier, accumulator, and in most cases, a pre-adder. They also have several internal registers provided such that the DSP48 may be configured to perform several different functions. Block RAMs are more suitable than distributed RAM (i.e., using LUTs as memory elements) for larger memories. The width of Xilinx memories is 18-bits. However, they can also be configured to operate in several different dimensions. For example, the number of entries can be doubled if the storage wordlength is shortened to nine bits [8].

c. Communications Interfaces

In [8], the training course states that Xilinx supports high-speed connectivity applications by embedding hard Internet Protocol (IP) blocks into its devices. Like DSP48 slices, these are dedicated blocks of silicon. They have a compact footprint and consume much less power than an equivalent design implemented in the logic fabric. Alternatively, soft IP blocks are also available. These are provided as Verilog hardware description language (VHDL) source and may be incorporated into a design if required. Soft cores give the customer the flexibility to include or omit an interface in any give design, but do not achieve the power savings of a hard core [8].

Additionally, RocketIO GTP Transceivers are low power and support a variety of protocols and standards. RocketIO GTX and GTH Transceivers are higher speed alternatives. Ethernet MAC Controllers and PCI Express interfaces are also available. By providing commonly used, standards-based communications interfaces as hard or soft cores, Xilinx reduces the design effort required by its customers [8].

d. Clocking Resources

In [8], the training course notes that Digital Clock Managers (DCMs) undertake several functions. One of these is to provide deskewed clocks, i.e., a different copy of the

synthesized clock is created for each clock region of the FPGA, appropriately phased to account for the delays in distributing the clocks throughout the device. Clock circuits and buffers are used to convey these different clocks to the various resources and regions of the FPGA. Clock buffers also act as clock enables, and this helps to reduce power consumption (i.e., each FPGA resource is only clocked and enabled when required). Another benefit of DCMs is that they reduce jitter present on the input clock. More recent devices also include a Phase Locked Loop (PLL), which helps to reduce jitter further [8].

Additionally, Clock Management Tiles (CMTs) were introduced in the Virtex-5 series, with each CMT comprising 2 DCMs and 1 PLL. Each of these components can be used in isolation or in cascade with another element. A common such usage is to precede the DCM with a PLL in order that the PLL filters jitter from the input reference signal [8].

Finally, Mixed Mode Clock Managers (MMCMs) first appeared in the Virtex-6 and Spartan-6 series, also based around the DCM and PLL, but offer enhanced options. For example, instead of the four coarse clock output phases provided in the Virtex-5 (0^o, 90^o, 180^o, and 270^o), the MMCM provides 8 (at 0^o, 45^o, 90^o, 135^o, 180^o, 225^o, 270^o, and 315^o). Fine-grained phase shifting is also possible, and the achievable resolution depends on the clock frequency [8].

e. Critical Path and Clock Frequency

Signals experience logic and routing delays through all logic paths as they propagate from one clocked register to the next. The critical path of a system is defined by [8] as "the longest combinatorial logic path between two clocked registers." It is "longest" in terms of propagation time. The critical path delay is the delay along the critical path, i.e., the longest combinatorial propagation delay through the circuit. However, the critical path delay relates to any group of combinatorial logic that could be simple gates or more complex operations such as arithmetic calculations [8].

Additionally, when considering arithmetic, the training course notes that the signals are not single 1-bit wires, but buses formed from several bits, in accordance with the arithmetic wordlengths. The last bit of the result (the most significant bit, in the case of addition and subtraction) is not available until the calculation is complete, and all the

necessary carries have propagated from the least significant bit to the most significant bit. Therefore, the longer the arithmetic wordlengths involved, the longer the critical path [8].

The training course also notes that when the logical operations are significant, the associated logic delays dominate over the routing delays. The term "routing delay" describes the time it takes signals to travel along the wires through, switch matrices and so on, between the logic elements in the circuit. The "logic delay" describes the time it takes a signal to pass through the logic elements in the circuit. DSP-FPGA designers must have a greater influence over logic delays in terms of how the design is set for implementation. Routing delays are largely managed by the design tool (specifically the place and route process), or at a more advanced level, by manual intervention in this process. As described in [8], this leads to the critical path delay (τ_{CPD}), which is the sum of several logic and routing delays therefore

$$\tau_{CPD} = \underbrace{\tau_{NAND} + \tau_{XOR} + \tau_{AND}}_{\text{logic delays}} + \underbrace{\tau_{route1} + \tau_{route2} + ... + \tau_{routeN}}_{\text{routing delays}} . \tag{4.1}$$

The critical path is significant because it restricts the maximum frequency at which the design can be clocked as described within the training course. As shown in (4.1), the critical path is the sum of several logic and routing delays which directly relates to the maximum clock frequency $\left(f_{clk_{\max}} \right)$ as described in [8] as

$$f_{clk_{\max}} = \frac{1}{\tau_{CPD}} \tag{4.2}$$

Therefore, if the total τ_{CPD} is 1.6 ns within the design, then

$$f_{clk_{\max}} = \frac{1}{\tau_{CPD}} = \frac{1}{1.6\text{ns}} = 625 \text{ MHz} \tag{4.3}$$

This means that if the clock frequency applied is less than 625MHz, then a signal leaving one register arrives at the next register within on clock period. This is vital to preserve the integrity of the circuit's functionality [8].

For the design to be valid, the output from the combinatorial logic must become valid during the same clock period as the input. If this were not the case, then it would be as if the combinatorial contained a 1-clock delay, which of course it cannot. From a mathematical perspective, this would fundamentally change the implemented algorithm, thus making it incorrect.

Signal changes occurring during the setup and hold times of the registers must be avoided as well, i.e., a short period before and after the active clock transition. Hence the true minimum clock period is slightly longer than the critical path delay, more accurately described in [8] as

$$T_{clk} > \tau_{CPD} + t_{setup} + t_{hold} . \tag{4.4}$$

The FPGA design tools (e.g., Xilinx ISE) will take these into account when analyzing the minimum clock period.

Having noted the dependence of clock frequency on length of the combinatorial logic path within the training course, additional registers would be inserted to break the combinatorial logic path to an acceptable size. This process is referred to as pipelining. By dividing the combinatorial logic path into two equal sections, a pipeline register increases the maximum clock frequency by a factor of 2. However, for every pipeline register, the latency is increased by one clock cycle [8].

In many cases, increasing the latency by one or two clock cycles to improve timing performance is an acceptable outcome. However, one notable exception is in feedback loops where each new calculation cannot start until the result of the last one has been computed; in this instance, delaying the readiness of the output is not helpful. More generally, pipeline registers must be placed within a design according to a formal method to preserve the integrity of the DSP function implemented by the circuit [8].

f. FPGA Power Consumption

In [8], the training course describes that FPGA power consumption comprises two components: static and dynamic power. FPGA companies are driven to reduce both. Static power consumption is associated with the circuitry that maintains the programmed

configuration of the FPGA. This is ultimately affected by transistor leakage current, which tends to rise with smaller process geometries and it is effectively constant for any given device. Dynamic power consumption is attributed to the switching of logic elements due to the actual processing of data by the configured design. In contrast, this tends to reduce with shrinking process geometries.

Additionally, FPGA companies can reduce static power consumption via the design of their devices, in terms of both process and architecture. By replacing commonly used functionality with integrated silicon blocks, they can reduce both static and dynamic power. Dynamic power consumption varies linearly with capacitance and frequency and with the square of the voltage described in [8] as

$$DynamicPower = CV^2 f \ . \tag{4.5}$$

A DSP engineer can influence dynamic power consumption through intelligent design. For example, polyphase filters can be used to implement rate changes in multi-rate designs while clocking each circuit element at the lowest possible rate [8].

g. Partial Reconfiguration

In [8], the training course mentions that partial reconfiguration is a relatively new technique whereby selected partitions of an FPGA device are allocated with two or more different modules but configured with only one of them at any given time. These modules can be used to reconfigure the selected partition without affecting the rest of the FPGA design while maintaining multiple such regions on one device.

Additionally, reconfiguration is accomplished by downloading the bitstream of the desired module onto the FPGA. This replaces the existing configuration of that partition, effectively swapping the old design out and inserting the new one in its place. Partial reconfiguration is therefore ideal when the design can be separated into functional blocks occupying the same area on the FPGA and when only one of these is required at any one time.

Subsequently, the training course states that there are several potential benefits associated with partial reconfiguration. Firstly, from a commercial perspective, using this

technique may mean that a significantly smaller FPGA device can be specified, thus leading to cost savings. As the design consumes fewer FPGA resources, it will also expend less static power than an equivalent design in which all modules are programmed on the device simultaneously. And finally, from a development perspective, partitioning the design in this manner may be more suitable for an engineering team undertaking a large and complex FPGA design.

Partial reconfiguration is also an enabler of software-defined radio, wherein the functionality of high-speed DSP subsystems running on the FPGA are defined at runtime, via software [8].

h. Implementation Metrics

The training course mentions that to tell if a design is good given the architecture of an FPGA, the goal is to optimize one of the following metrics or to achieve a desirable balance between them:

(1) Resource Utilization / Area – The amounts of the various FPGA resources required to implement the user design. This primarily includes slices, DSP48s, and Block RAMs

(2) Timing Performance – The maximum frequency at which the circuit can be clocked. This is affected by the critical path.

(3) Power Consumption – Overall FPGA power consumption is affected by the dimensions of the FPGA, the number of circuit elements being clocked, and their frequency of operation.

Additionally, there are certain interactions between these three metrics. For example, often the technique of pipelining is used to reduce the critical path of a design and hence increase the maximum frequency at which it can be clocked. Pipelining involves the insertion of registers, which obviously increase the overall resource cost. However, with the increased clock frequency, improved resource sharing may be possible, hence reducing resource cost, etc. The implications of the design choices are not always straightforward. Therefore, achieving the best balance requires intelligent DSP design based on knowledge of the device architecture [8].

C. XILINX VIRTEX ULTRASCALE+

1. Overview

a. Product Description

In [21], the Xilinx webpage states that the "Virtex® UltraScale+™ FPGA VCU118 Evaluation Kit is the ideal development environment for evaluating the cutting edge Virtex UltraScale+ FPGAs. Virtex UltraScale+ devices provide the highest performance and integration capabilities in a FinFET node, including both the highest serial I/O and signal processing bandwidth, as well as the highest on-chip memory density" [21].

The webpage continues stating that this "kit is ideal for prototyping applications ranging from 1+ Tb/s networking and data center to fully integrated radar/early-warning systems" [21].

b. Key Features and Benefits

Presented in [21], the following key features and benefits are listed accordingly:

(1) Dual 80-bit DDR4 Component Memory

(2) RLDRAM3 (2x36-bit) Memory

(3) Dual QSFP28 Interfaces

(4) PCIe Gen3 x16 (V_{CCINT} = 0.85V)

(5) VITA 57.4 FMC+ Interface

(6) VITA 57.1 FMC Interface

(7) Samtec FireFly Interface [21]

2. Hardware

Depicted in Figure 13 is the Virtex UltraScale+ board and chipset that was utilized throughout the research conducted for this thesis. Table 5 is an additional list of features that are broken down according to functionality. Table 6 provides a list of total available resources for area utilization or logic fabric utilization.

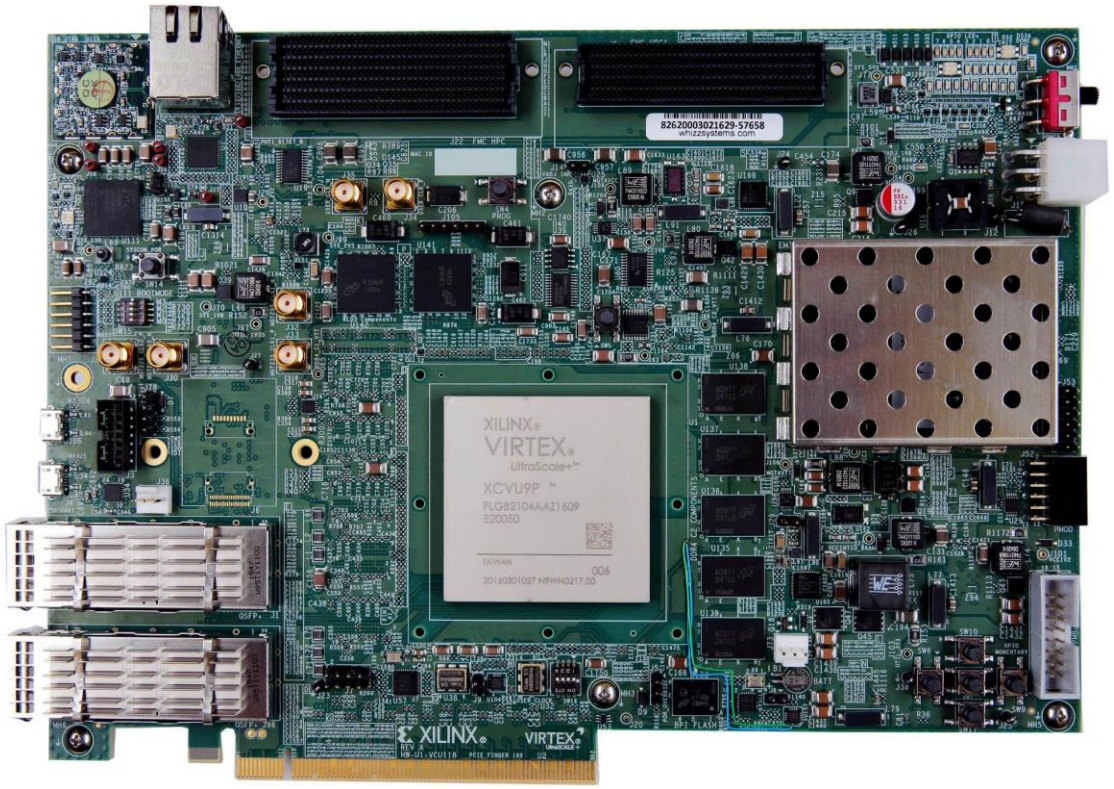

Figure 13. Virtex UltraScale+ VCU 118 Evaluation Board. Source: [21].

Table 5. Virtex UltraScale+ VCU 118 Evaluation Board Features. Source: [21].

Communications & Networking	Clocking	Expansion Connectors	Configuration
10/100/1000 Mbps Ethernet (SGMII)	SI5335A Quad Clock Generator	FMC+ HSPC connector (24 – 28Gbps GTY Transceivers, 80 differential user defined pairs)	Onboard JTAG configuration circuitry to enable configuration over USB
Dual 4x28Gbps QSFP28 cages	Si570 IIC Programmable LVDS Clock Generator	FMC HPC1 connector (58 differential user defined pairs)	JTAG header provided for use with Xilinx download cables such as the Platform Cable USB II
Samtec FireFly 4x28Gbps Interface	SI5328C Clock Multiplier and Jitter Attenuator	PMOD header	QSPI flash memory
Dual USB-to-UART Bridge with mico-B USB connector	2x SMA MGT Reference Clock inputs	IIC	
RJ45 Ethernet connector	1 SMA User Clock input		
PCI Express endpoint Gen3 x 16			
Control & I/O	**Memory**	**Display**	**Power**
User Push Buttons (x5)	Two 4 GB DDR4 component memory interfaces (five [256 Mb x 16] devices each)	Users & Status LEDs	12V wall adapter or ATX
User DIP Switch (4-position)	4 MB RLD3 component memory interfaces (five [256 Mb x 16] devices each) IIC EEPROM: 8Kb		
PMBUS & System Controller MSP430 for power, clocks, SD-Card and I2C bus switching	Micro Secure Digital (SD) connector 1Gb Quad SPI Flash		

Table 6. Total Resources for Project Part: xcvu9p-plga2104-2L-e-es1.

Resource	Total Available
LUT	1182240
LUTRAM	591840
FF	2364480
BRAM	2160
DSP	6840
IO	832
BUFG	1800

D. SIMULATION PROCESS

To produce the results in the following chapter, a specific process involving MathWorks' Simulink and Xilinx's Vivado must be utilized. These steps will be supplied within this section.

First and foremost, the correct versions of MATLAB and Vivado must be installed for the HDL Workflow Advisor (that will be discussed later) to work correctly. The correct pairings are MATLAB R2017b and Vivado 2016.4 and for future iterations, MATLAB R2018a would be compatible with Vivado 2017.1+. In Table 7, a full list of toolboxes and add-ons is provided. Of those, the essentials are the Communications Toolbox, DSP System Toolbox, Filter Design HDL Coder, Fixed-Point Designer, Fuzzy Logic Toolbox, HDL Coder, HDL Verifier, RF Blockset, RF Toolbox, Signal Processing Toolbox, and any of the FPGA for Simulink Toolboxes / Add-ons that are available at the time.

Table 7. List of Installed MATLAB/Simulink Toolboxes and Add-ons.

Aerospace Blockset	Version	3.2	(R2017b)
Aerospace Toolbox	Version	2.2	(R2017b)
Antenna Toolbox	Version	3	(R2017b)
Audio System Toolbox	Version	1.3	(R2017b)
Automated Driving System Toolbox	Version	1.1	(R2017b)
Bioinformatics Toolbox	Version	4.9	(R2017b)
Communications System Toolbox	Version	6.5	(R2017b)
Computer Vision System Toolbox	Version	8	(R2017b)
Control System Toolbox	Version	10.3	(R2017b)
Curve Fitting Toolbox	Version	3.5.6	(R2017b)
DSP System Toolbox	Version	9.5	(R2017b)
Data Acquisition Toolbox	Version	3.12	(R2017b)
Database Toolbox	Version	8	(R2017b)
Datafeed Toolbox	Version	5.6	(R2017b)
Econometrics Toolbox	Version	4.1	(R2017b)
Embedded Coder	Version	6.13	(R2017b)
Filter Design HDL Coder	Version	3.1.2	(R2017b)
Financial Instruments Toolbox	Version	2.6	(R2017b)
Financial Toolbox	Version	5.1	(R2017b)
Fixed-Point Designer	Version	6	(R2017b)
Fuzzy Logic Toolbox	Version	2.3	(R2017b)
GPU Coder	Version	1	(R2017b)
Global Optimization Toolbox	Version	3.4.3	(R2017b)
HDL Coder	Version	3.11	(R2017b)
HDL Verifier	Version	5.3	(R2017b)
Image Acquisition Toolbox	Version	5.3	(R2017b)
Image Processing Toolbox	Version	10.1	(R2017b)
Instrument Control Toolbox	Version	3.12	(R2017b)
LTE HDL Toolbox	Version	1	(R2017b)
LTE System Toolbox	Version	2.5	(R2017b)
MATLAB Coder	Version	3.4	(R2017b)
MATLAB Compiler	Version	6.5	(R2017b)
MATLAB Compiler SDK	Version	6.4	(R2017b)
MATLAB Report Generator	Version	5.3	(R2017b)
Mapping Toolbox	Version	4.5.1	(R2017b)
Model Predictive Control Toolbox	Version	6	(R2017b)
Model-Based Calibration Toolbox	Version	5.3	(R2017b)
Neural Network Toolbox	Version	11	(R2017b)
OPC Toolbox	Version	4.0.4	(R2017b)
Optimization Toolbox	Version	8	(R2017b)
Parallel Computing Toolbox	Version	6.11	(R2017b)
Partial Differential Equation Toolbox	Version	2.5	(R2017b)
Phased Array System Toolbox	Version	3.5	(R2017b)

Polyspace Bug Finder	Version	2.4	(R2017b)
Polyspace Code Prover	Version	9.8	(R2017b)
Powertrain Blockset	Version	1.2	(R2017b)
RF Blockset	Version	6.1	(R2017b)
RF Toolbox	Version	3.3	(R2017b)
Risk Management Toolbox	Version	1.2	(R2017b)
Robotics System Toolbox	Version	1.5	(R2017b)
Robust Control Toolbox	Version	6.4	(R2017b)
Signal Processing Toolbox	Version	7.5	(R2017b)
SimBiology	Version	5.7	(R2017b)
SimEvents	Version	5.3	(R2017b)
Simscape	Version	4.3	(R2017b)
Simscape Driveline	Version	2.13	(R2017b)
Simscape Electronics	Version	2.12	(R2017b)
Simscape Fluids	Version	2.3	(R2017b)
Simscape Multibody	Version	5.1	(R2017b)
Simscape Power Systems	Version	6.8	(R2017b)
Simulink 3D Animation	Version	7.8	(R2017b)
Simulink Check	Version	4	(R2017b)
Simulink Code Inspector	Version	3.1	(R2017b)
Simulink Coder	Version	8.13	(R2017b)
Simulink Control Design	Version	5	(R2017b)
Simulink Coverage	Version	4	(R2017b)
Simulink Design Optimization	Version	3.3	(R2017b)
Simulink Design Verifier	Version	3.4	(R2017b)
Simulink Desktop Real-Time	Version	5.5	(R2017b)
Simulink PLC Coder	Version	2.4	(R2017b)
Simulink Real-Time	Version	6.7	(R2017b)
Simulink Report Generator	Version	5.3	(R2017b)
Simulink Requirements	Version	1	(R2017b)
Simulink Test	Version	2.3	(R2017b)
Spreadsheet Link	Version	3.3.2	(R2017b)
Stateflow	Version	9	(R2017b)
Statistics and Machine Learning Toolbox	Version	11.2	(R2017b)
Symbolic Math Toolbox	Version	8	(R2017b)
System Identification Toolbox	Version	9.7	(R2017b)
Text Analytics Toolbox	Version	1	(R2017b)
Tracking and Sensor Fusion Toolbox	Version	1	(R2017b)
Trading Toolbox	Version	3.3	(R2017b)
Vehicle Network Toolbox	Version	3.4	(R2017b)
Vision HDL Toolbox	Version	1.5	(R2017b)
WLAN System Toolbox	Version	1.4	(R2017b)
Wavelet Toolbox	Version	4.19	(R2017b)

Once all software tools are established, an active license must be procured for whichever FPGA is being utilized. In this case, a specific license was activated within Vivado allowing synthesis and implementations to occur for the specific board that was selected i.e., Project Part: xcvu9p-plga2104-2L-e-es1. Without a valid license, the Vivado projects to be created or prior ones to be reviewed will not be allowed.

The next step is to ensure that the HDL Workflow Advisor is properly synced with Vivado. To do this, input the command shown in Figure 14. Ensure the drive path for the system at use and the Vivado versions are correct. The vivado.bat file is the key to a proper sync.

```
>> hdlsetuptoolpath('ToolName','Xilinx Vivado','ToolPath',...
'D:\Xilinx\Vivado\2016.4\bin\vivado.bat');
```

Figure 14. MATLAB / Vivado Sync Command.

If this is successful, a return shown in Figure 15 will appear.

```
Prepending following Xilinx Vivado path(s) to the system path:
D:\Xilinx\Vivado\2016.4\bin
```

Figure 15. Successful MATLAB / Vivado Sync Return.

Now that MATLAB and Vivado are synced, the Simulink design can be opened / created. Once a design is created and run with no errors, it is time to synthesize and implement the design through Simulink. To do so, go to Code >> HDL Code >> HDL Workflow Advisor… as shown in Figure 16 and at this point, a pop-up will occur shown in Figure 17. This pop-up allows the user to select which system within the design to be synthesized and implemented. Individual pieces or the entire design can be selected.

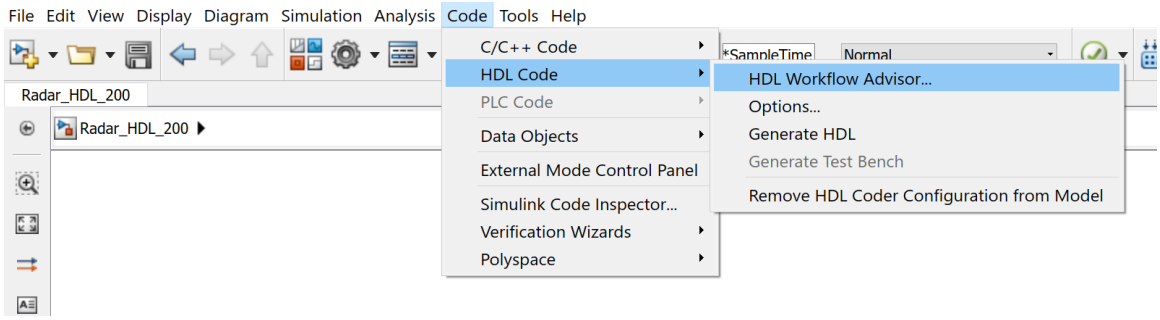

Figure 16. HDL Workflow Advisor Selection.

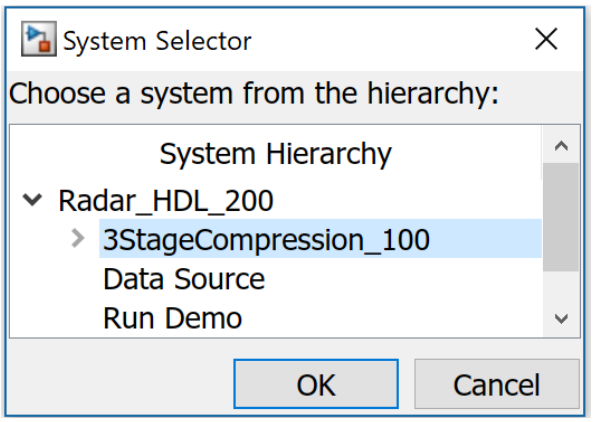

Figure 17. System Selector Pop-up.

Once selected, the HDL Workflow Advisor will pop-up. Now using the arrows to the side of each folder, open them to see the list of categories within each. Figure 18 shows the exact setup for this thesis for Section 1.1. The Target Workflow, Family, Device, and Project Folder will all require inputs. Note: always press apply after any changes in all HDL Workflow Advisor windows.

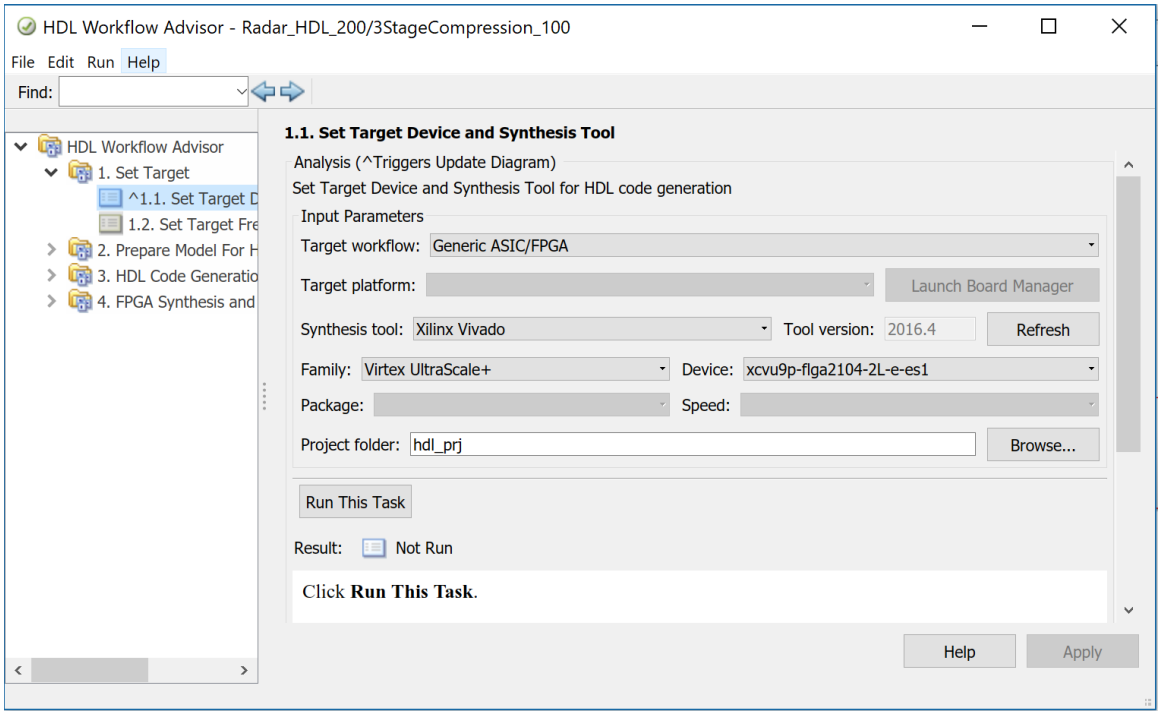

Figure 18. Section 1.1: Set Target Device and Synthesis Tool Window.

In Section 1.2, the target frequency is the frequency in which the FPGA will run the design. Normally, naming conventions are of *subsystem name*_*target frequency* is applied to the Simulink design so that when it is run in HDL Workflow Advisor, separate runs using different frequencies will be noticeable. In this example, 100 MHz is being selected.

For Section 2 and all subsections, nothing needs to be altered. In Section 3.1.1, the Language needs to be changed to "Verilog" and the boxes in front of generate traceability report, generate resource utilization report, and generate optimization report need to be checked as depicted in Figure 19.

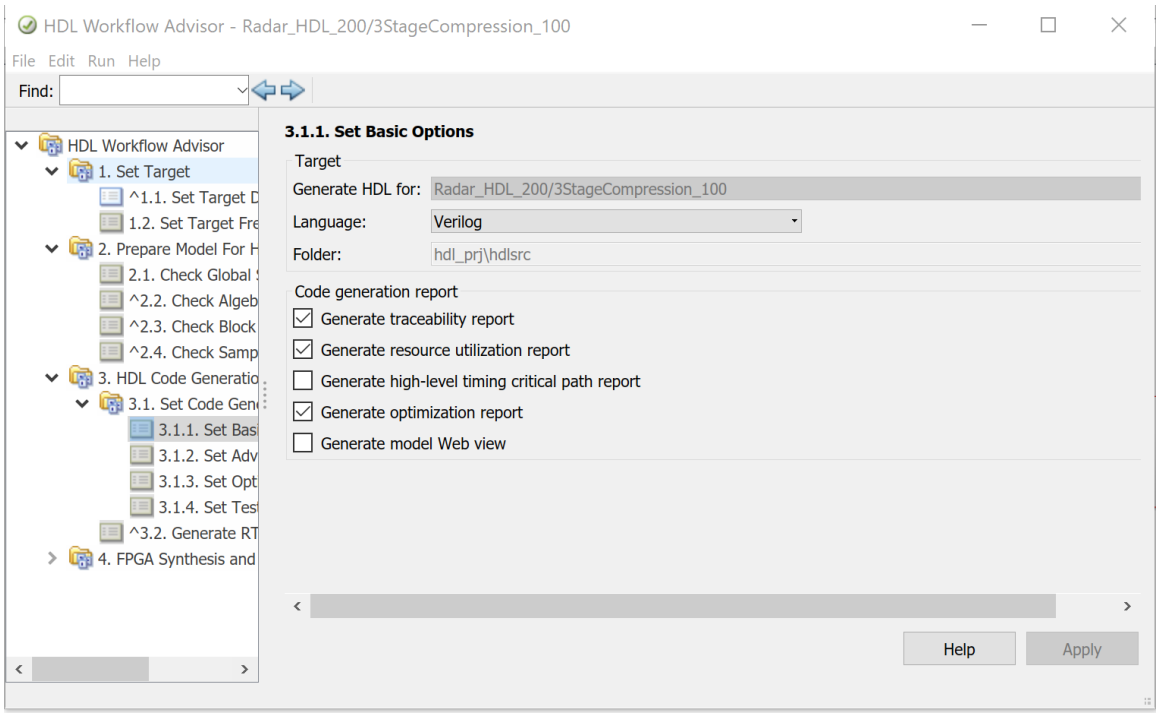

Figure 19. Section 3.1.1: Set Basic Options Window.

In Section 3.1.2, ensure that the reset type is changed to "Synchronous." Then select the ports tab in the additional settings section and ensure there is no check in front of the minimize clock enables box, as shown in Figure 20.

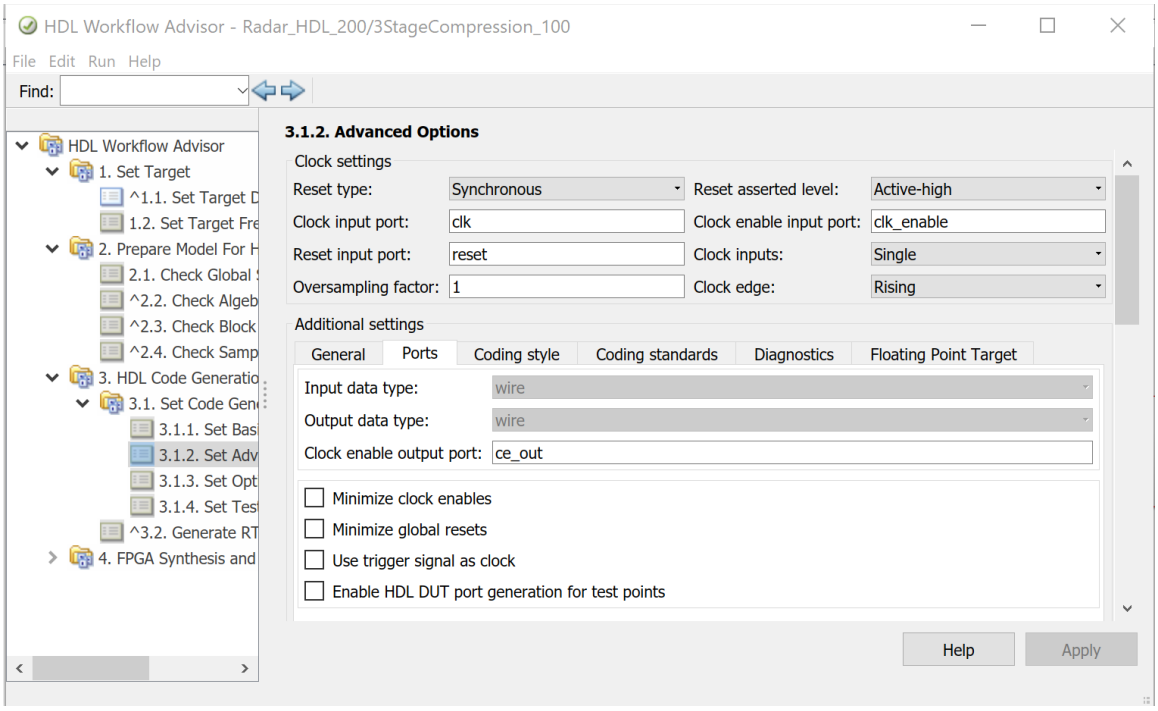

Figure 20. Section 3.1.2: Advanced Options Window.

The remainder of options within Section 3.1 will be selected based on experience with the design and software. Section 3.2 does not require any changes unless the FPGA being utilized is supported by Simulink like a Virtex 7, if so, then generate test bench can be selected as well as generate RTL code. Unfortunately, the Virtex Ultrascale+ is still not properly supported but MathWorks is aware of the demand for future patches.

In Section 4.1, a synthesis objective can be selected at this point. If none is selected, then a default synthesis will be created that can be later examined through Vivado. Within Section 4.2, ensure all boxes within the Input parameters for both 4.2.1 and 4.2.2 are clear. When all is set to the user's satisfaction, right click on 4.2.2 and select run to selected task, as depicted in Figure 21.

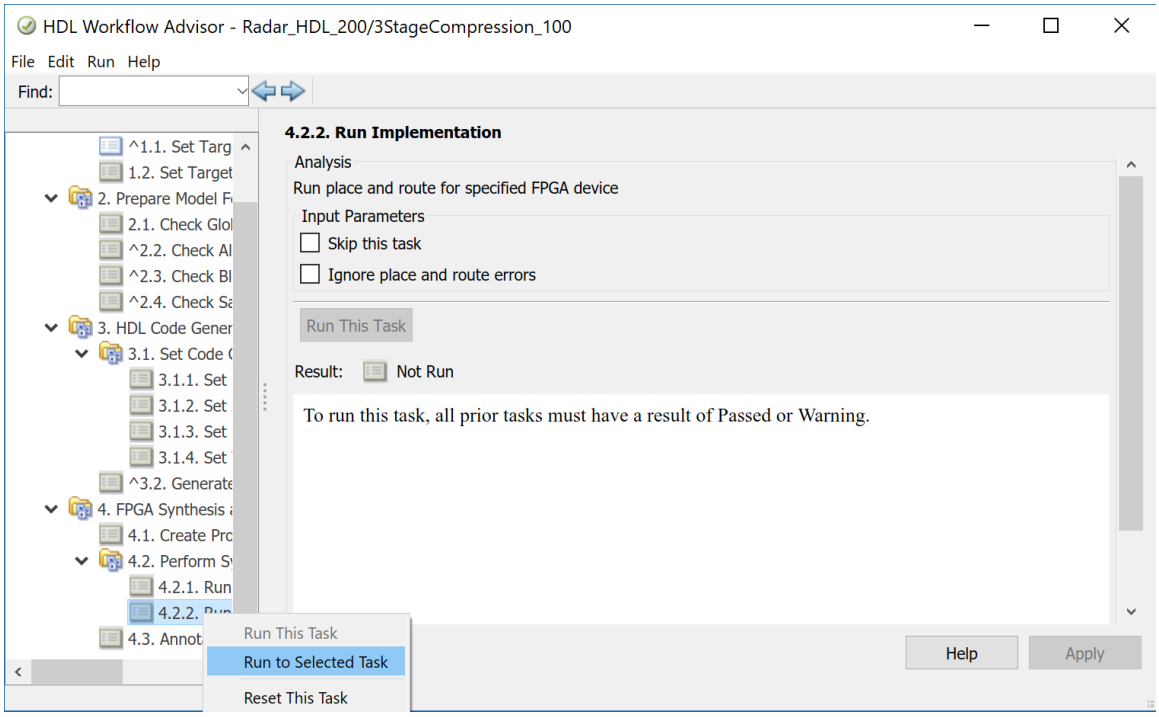

Figure 21. Section 4.2.2: Run Implementation Window.

As the program runs, check marks will begin to appear to each section and subsection. At any point a section does not pass, it will end the run with an error output. As the HDL Workflow Advisor continues to run, an HDL Generation Report Summary for the design will be created. This will provide valuable information regarding I/O bits, total utilization, and latency. Depending on the sophistication of the design, running HDL Workflow Advisor from start to implementation can take from 30 minutes up to one hour depending on the user's processing power and available RAM.

Upon completion, the user will then open the folder in which the project was saved. Within the folder, a Vivado Project File will be created with the naming convention that was previously established. Execute this file and Vivado will automatically startup and run the selected file. Once the Vivado Project File opens, the user will be able to visually analyze the outcome as depicted within next chapter's results. If additional synthesis and implementation strategies are required, click on the button at the bottom of the "Design Runs" panel that is depicted in Figure 22.

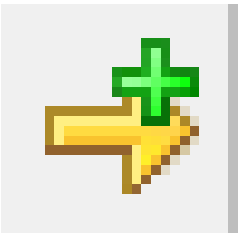

Figure 22. Create Runs button.

Once selected, navigate through the different options that pertain to the new runs. There are different strategies for both synthesis and implementation. The user can then load a queue of many different run strategies and allow them to process while the user is away from the computer. Once the queue of created runs is complete, right click on the run that requires analyzing and select "Make Active." This will update all the Vivado panels to the selected synthesis and implementation.

E. CHAPTER SUMMARY

In this chapter, it was presented that by leveraging the knowledge of DSP theory, finite wordlength effects, the architecture of the FPGA device being targeted, while adding a DSP implementation specialist's knowledge, an efficient mapping of the DSP algorithm to FPGA hardware can be achieved. A seamless and invaluable process requires a good understanding of design flow and the Electronic Design Automation Tools being utilized. Verification and testing is also an essential element in ensuring the design works as intended. Furthermore, integration is necessary to ensure the DSP design works within the larger system. The chapter finished with the steps and process necessary to replicate the simulation, synthesis, and implementation results of this thesis in the next chapter.

V. MODELING, SYNTHESIS, AND IMPLEMENTATION RESULTS

Within this chapter, the modeling, synthesis, and implementation results are presented. The time required to simulate each model through HDL Workflow Advisor is 15-30 minutes depending on the overall computing power of the user. Subsequently, the time required to synthesize and implement one run for each model is an additional 30 minutes to an hour. One run is defined as a synthesis and implementation pair chosen by the user to maximize or balance the given parameters of either utilization (area), power, or timing. The user can queue multiple runs within Vivado, but these runs will be processed simultaneously by the computer tasked to do so. Therefore, if the computer is not capable of multitasking large CPU and RAM intensive algorithms, it is highly recommended to not queue for multiple runs. However, if the computer can process an intensive workload, it is realistic to queue 8-10 runs and allow the computer to continue synthesizing and implementing the runs overnight.

The requirements for a successful DRFM and three-stage compression were never explicitly defined throughout this research. This led to an exploration of FPGA capabilities as shown through the data presented in this chapter. Through these data, it can best be served as a guide to what is to be expected once detailed parameters are set for each system. For example, if the system is going to be placed in a radar that is required to run many different systems on the same FPGA, then area or timing optimization may be most beneficial depending which is more important at the time. On the other hand, if the system was to be placed within an aircraft, then power optimization may be most beneficial to not exclude overall power resources to more vital components within the aircraft.

There are two digital systems: the DRFM and three-stage compression for an LPI radar. The three-stage compression includes a data storage component required for proper simulation results. Simulink models were all built manually, and the simulations were verified using pre-existing MATLAB code developed from past thesis results. These models on their own are not complete and shall be explained within this chapter. Figure 23 is a screenshot from Vivado giving a detailed explanation, which populated for every

model and every run. This figure is stating that although the model successfully synthesized and implemented, due to I/O pins not actually being connected either logically or physically (through the model), the implementation may cause severe harm to an FPGA. These I/O pins are essentially placeholders until the design is finished and for each bit of I/O that is left unconnected, the overall I/O bit count available on the FPGA shall be taxed. This is evident in incomplete designs because all other parameters will appear to be normal except for I/O which will appear to be utilizing a much higher number than what is required. These indicators should be treated as a guide for identifying open ports and should not be alarming unless the design is considered finalized. Once these ports are connected, the I/O utilization shall reduce significantly.

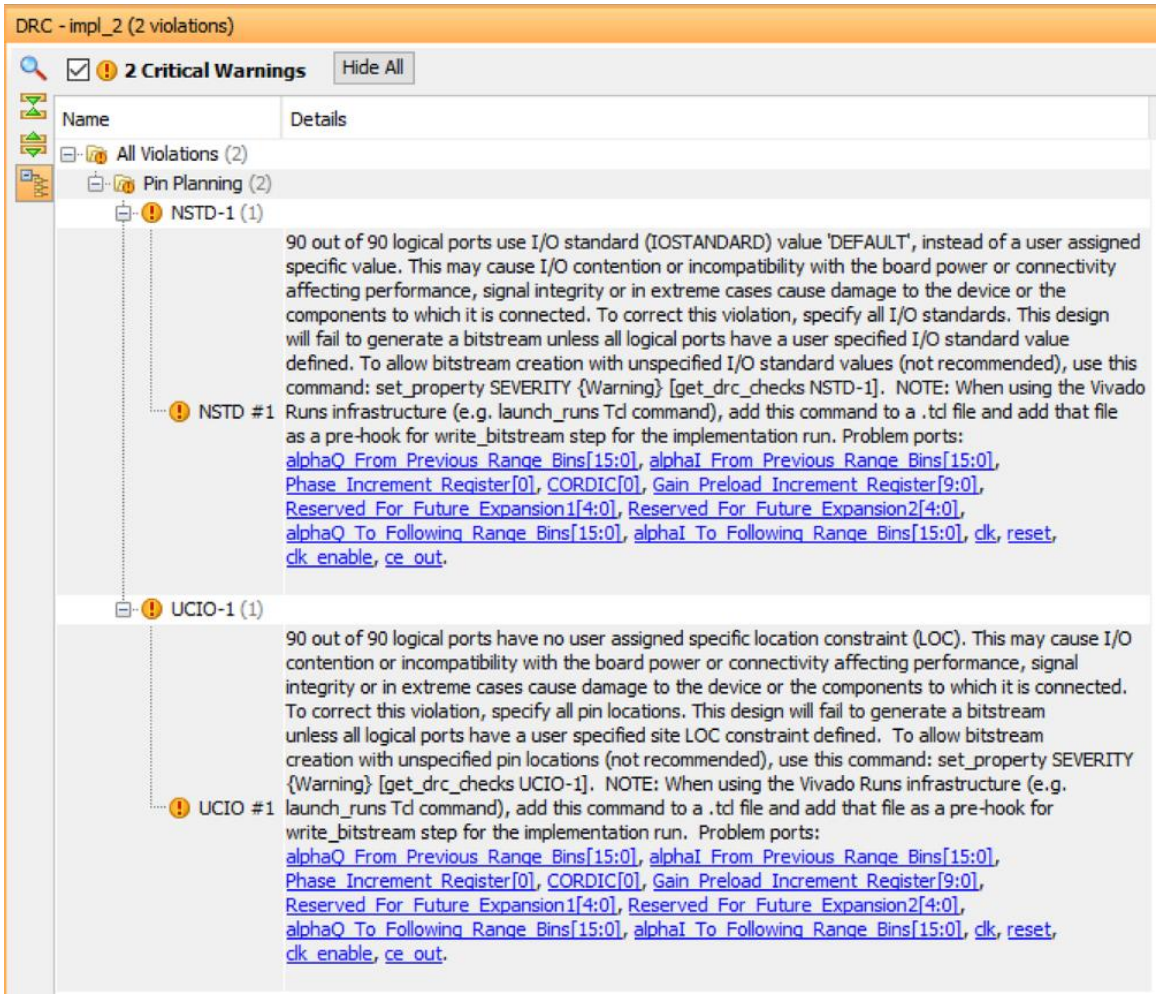

Figure 23. Implementation Critical Warnings.

After analyzing the data across all six models, there were a few observations that became common amongst all. The first was previously mentioned regarding the I/O utilization. The second observation is regarding the thermal power. It can be noted that the Junction Temperature and Thermal Margin are inversely proportional to one another. If the Junction Temperature rises by 0.1 degree Celsius, the Thermal Margin can be expected to decrease by 0.1 degree Celsius.

The third observation is regarding the Vivado Default runs. These are the runs that were created through Simulink in the HDL Workflow Advisor. In nearly all cases, these were run with no specific synthesis or implementation strategies. Therefore, the results appear to be skewed in comparison to the other results within the same model. Afterwards, the runs varied the synthesis strategy (seven different options) while maintaining the same implementation strategy (twenty-seven different options) except for the DRFM model – this model used the Performance_Explore strategy versus the Performance_Retiming strategy like the rest of the models. The DRFM model also did not run a Flow_AreaOptimized_medium synthesis strategy. The reasons for not testing twenty-seven different implementation strategies may appear to be obvious but it must be noted that the shear amount of time required for one set of synthesis and implementation runs becomes unfeasible when multiplied by twenty-seven. So, the implementation became a control within this research in which allows the user to analyze the results of differing synthesis strategies. Once design requirements are set, a few selected implementation strategies can be used in multiple variations with the synthesis strategies.

A. DIGITAL RADIO FREQUENCY MEMORY (DRFM)

Project Name: DRFMvers2_vivado

Implementation Strategy: Performance_Explore

1. Model

The DRFM Simulink model in Figure 25 was built based off the DIS-512 Block Diagram in Figure 24. The DRFM Simulink model is incomplete due to no I/O connections made to the Phase Increment Register, CORDIC, Gain Preload/Increment Register, From

Previous Range Bins for both I and Q, To Following Range Bins for both I and Q, and two Reserved for Future Expansion slots. With this many unconnected I/O ports, the I/O utilization is significantly higher. The model uses a microprocessor that establishes phase and gain coefficients. These coefficients are then preloaded and then incremented into registers. The phase increment goes through a phase adder with a 5-bit phase input data supplied from the CORDIC. The 5-bit data are then pipelined and then processed through an I/Q look-up table which splits the input data into two 8-bit streams for both the I and Q components. These data streams are once again run through pipelining and then multiplied by the information stored within the gain increment register which causes the bit stream to shift between 8 and 18 bits depending on the multiplication results. These data streams are then bit extracting the first five bits to future expansion output ports whereas the remaining thirteen bits are pipelined into another adder in which the data stream is padded in the front with three zeros. These 16-bit data streams are then added with 16-bits from previous range bins for both the I and Q components. These new 16-bit data streams from this addition are then sent to following range bins.

To ensure a working model is complete for FPGA implementation, this design must be linked to both a microprocessor feed as well as CORDIC feed and then connected accordingly to the port designations. The last output would therefore become the final output of the DRFM model. Additionally, the current DRFM model was synthesized and implemented with ease for a target frequency of 500 MHz autonomously from all other models. A hypothesis is that the target frequency would only decrease to accommodate a more complex structure.

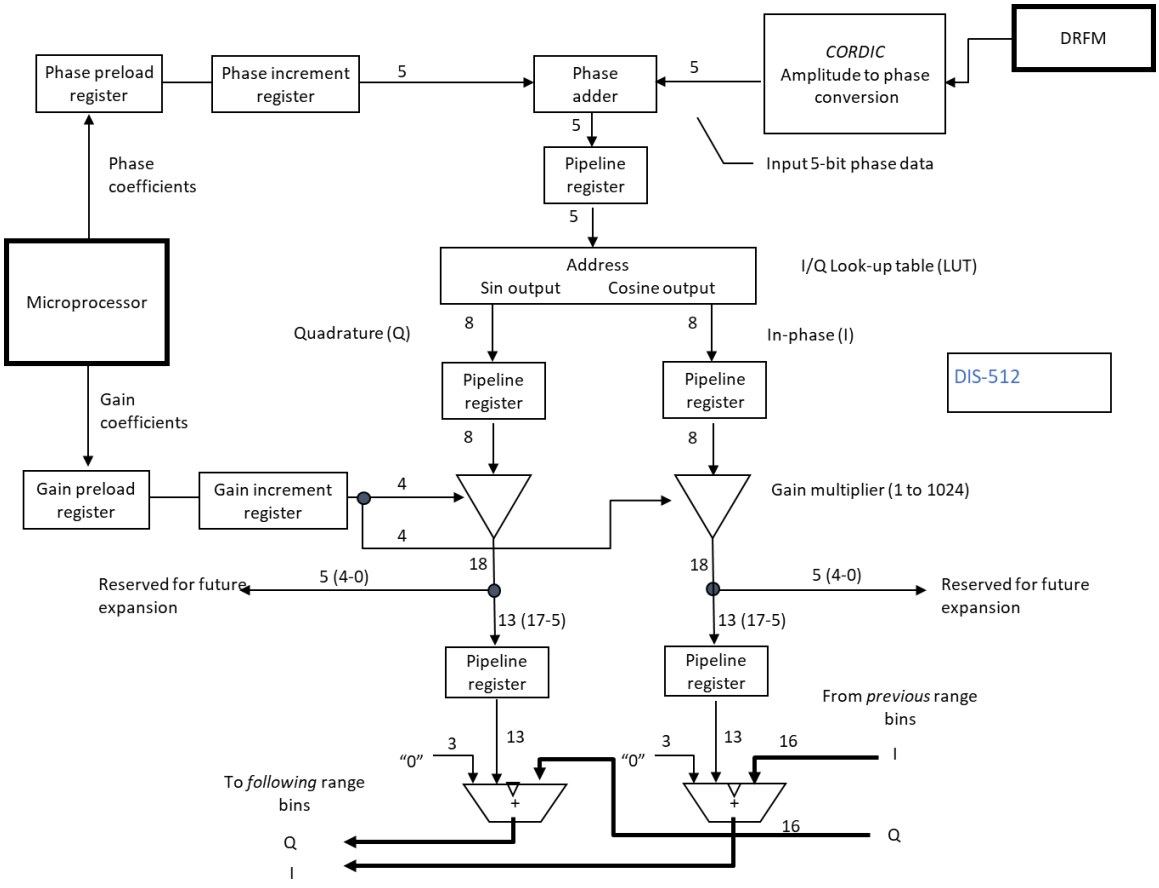

Figure 24. DIS-512 DRFM Block Diagram.

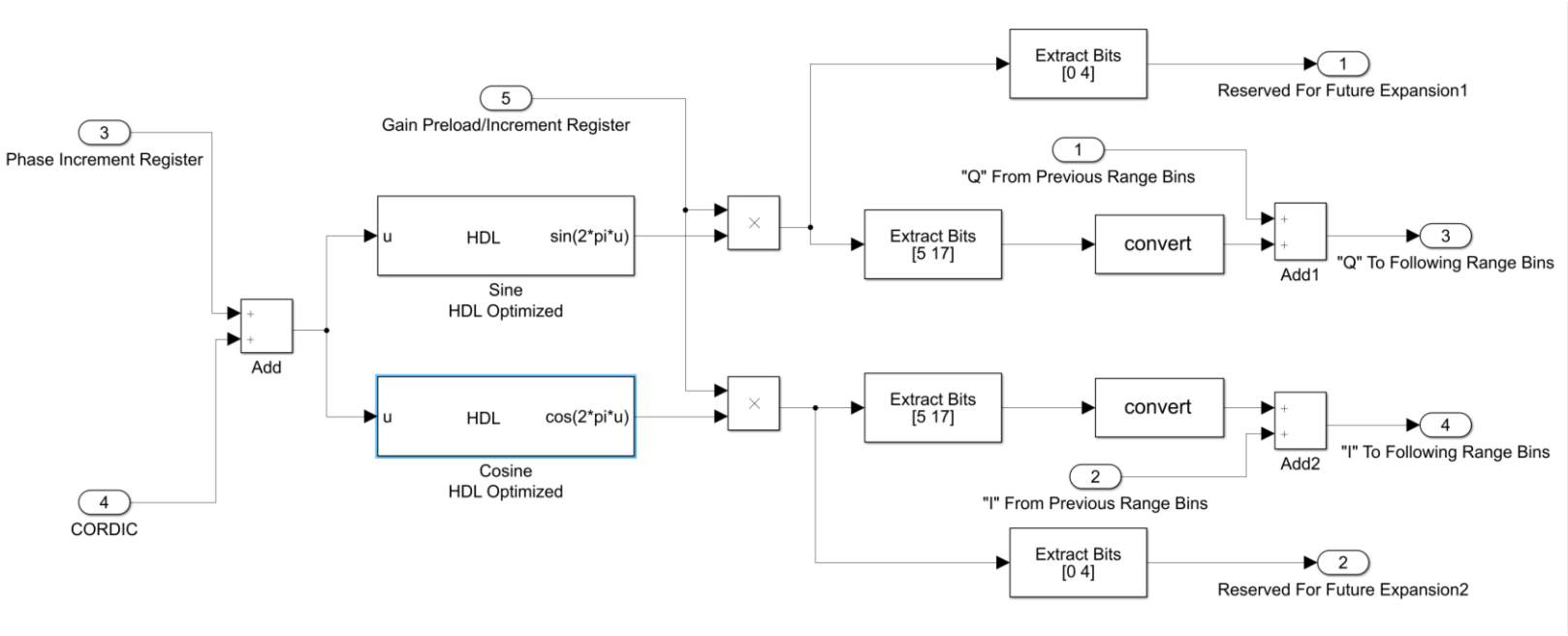

Figure 25. DIS-512 DRFM Block Diagram Built in Simulink.

2. Synthesis / Implementation

Within Figure 26, the DRFM model timing results are displayed for each synthesis type run. The graph depicts the Worst Negative Slack (WNS) and is a point in the logical path within the design having the maximum (worst) negative slack (timing). Worst Hold Slack (WHS) points to a path within the design having the maximum or worst hold (delay) in the slack. These times are both displayed in nanoseconds. The WNS and WHS resulted in positive numbers, which means that the timing of the model passed. Significant deviations in WNS are displayed depending on the run; however, the WHS stays consistent.

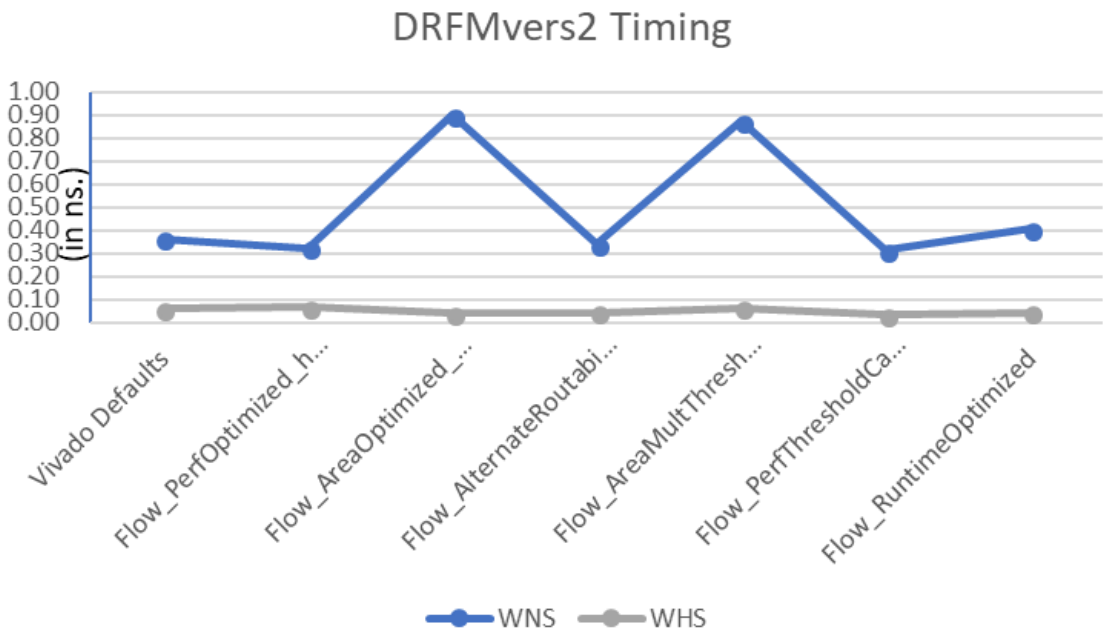

Figure 26. DRFM Model Timing Results.

Within Figure 27, the DRFM model power results are displayed for each synthesis type run. The graphs depict the total on-chip power, how each synthesis run compares to one another regarding power used per each static and dynamic component, and the thermal power associated to each run. The total on-chip power graph shows a deviation within the Flow_PerfOptimized_high synthesized run whereas the remaining runs are more consistent

with each other. Within the power utilization comparison graph, many of the power components have minimal deviations except for the DSP component, which indicates that only the Flow_AreaMultThresholdDSP and Flow_AreaOptimized_high utilize power for DSP. However, this is consistent with Figure 28 because these two runs are the only two that utilize DSP logic space. The thermal power results are identical per run.

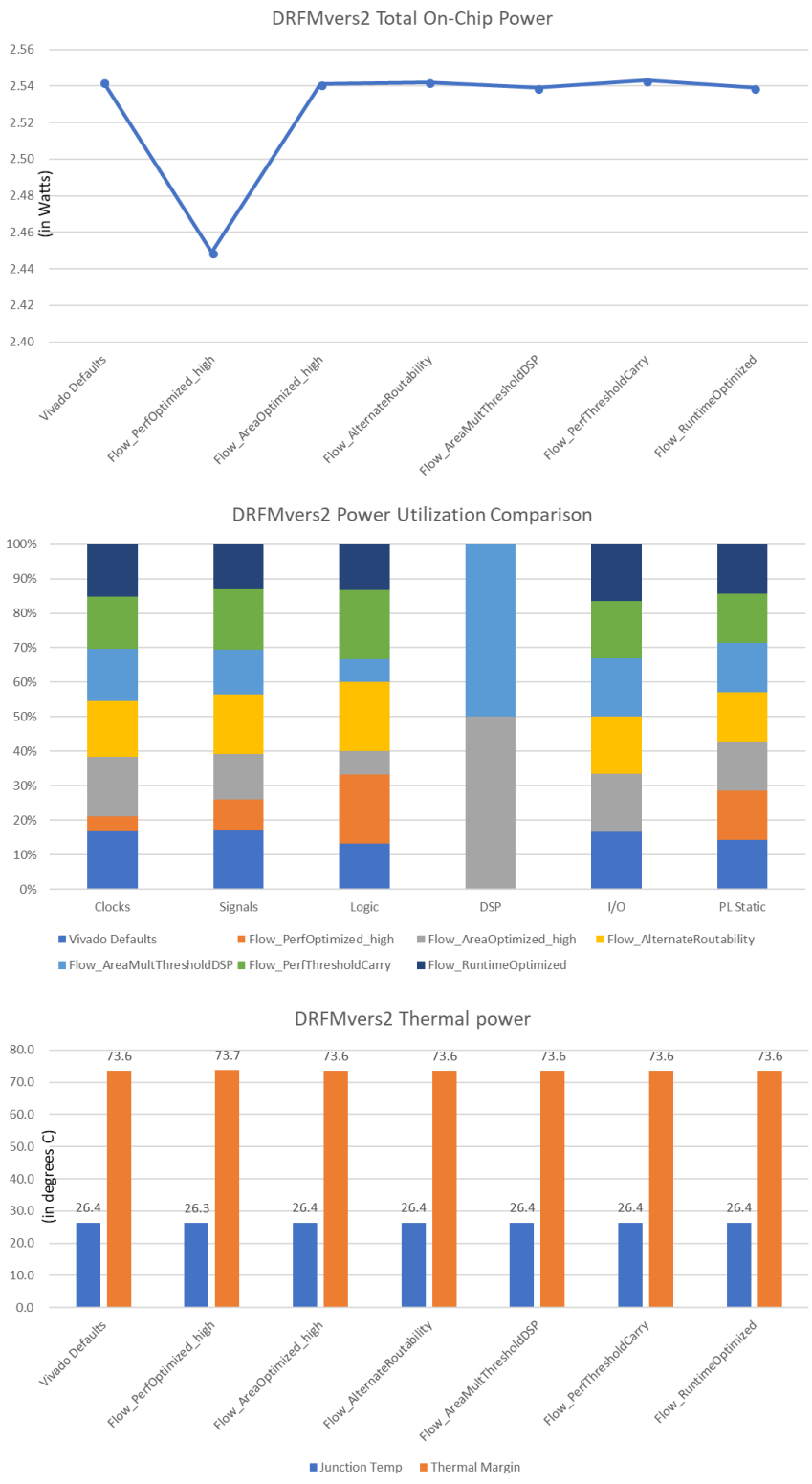

Figure 27. DRFM Model Power Results.

Within Figure 28, the DRFM model utilization results are displayed for each synthesis type run. The graphs depict the total and individual utilizations per run. To note, I/O and BUFG utilization are not provided for the Flow_PerfOptimized_high run. These are outliers in comparison to the remaining runs in which use 90 I/O slices and 1 BUFG slice. Many trade-offs can be captured through the individual utilization charts. The most prominent trade-off is the fact that the amount of LUT and FFs are significantly reduced with the use of a couple DSP as depicted through the differences between the Flow_AreaMultThresholdDSP and Flow_AreaOptimized_high and the remaining runs.

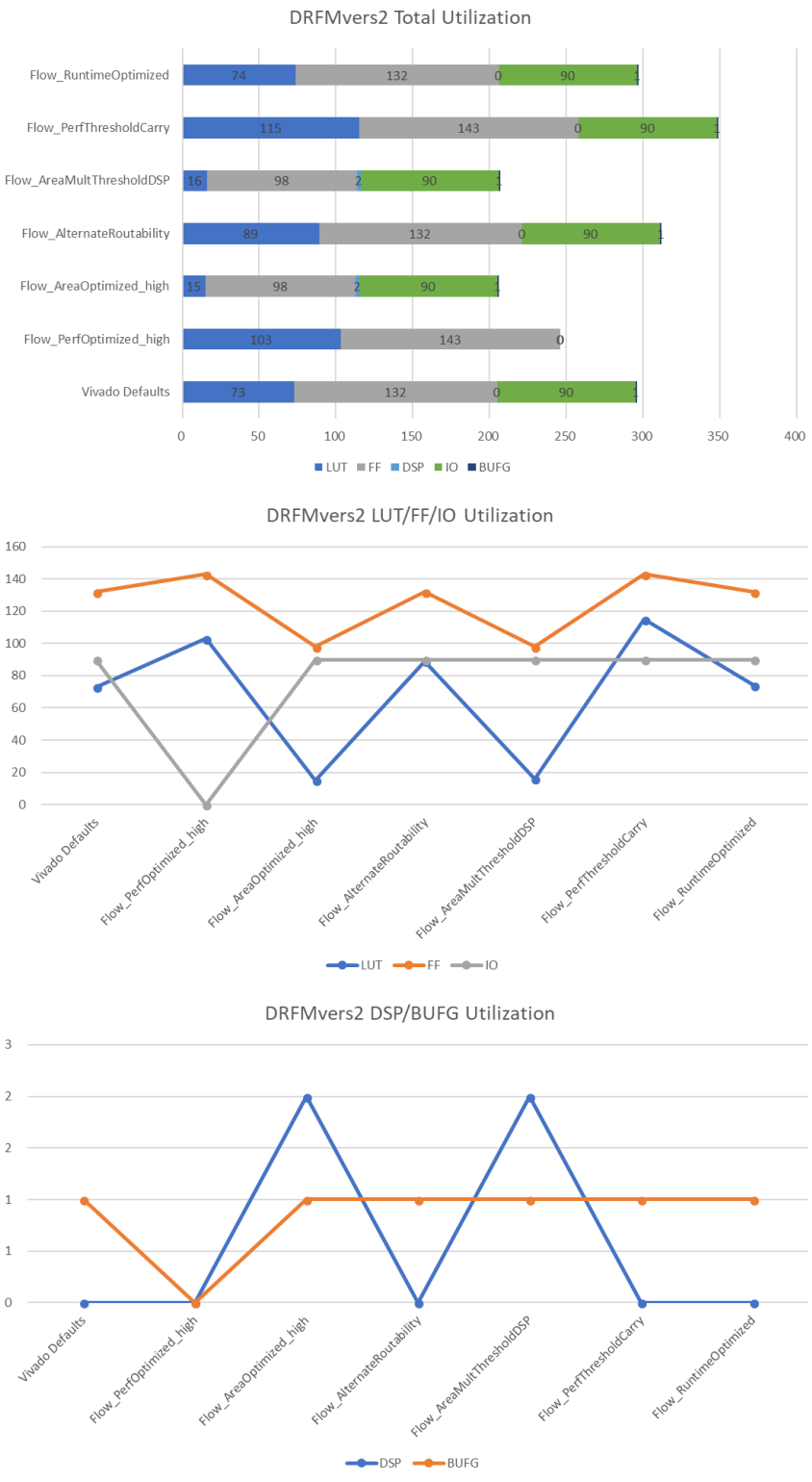

Figure 28. DRFM Model Utilization Results.

B. RANGE COMPRESSION

Project Name: Correlation_Receiver_500_vivado

Implementation Strategy: Peformance_Retiming

1. Model

The Range Compression (also known as Correlation Receiver depending on the publication) model shown in Figure 29 is the first step within the three-stage compression, which was discussed in Chapter II. This Range Compression model was synthesized and implemented with a target frequency of 500 MHz autonomously from all other models. The model implements a coherent correlation receiver of finite duration NT where $N = 2$ is the number of matched periods for the signal. $T = N_c t_b$ where the implementation N_c = 102 and $t_b = 0.133\mu s$, where N_c and t_b are the number of subcodes and the subcode period respectively. To implement the coherent correlation receiver, the signal begins by going through a tapped delay of N_c and performs a cross-correlation between the received signal and the complex conjugate of the reference signal. Then at each subcode, the single correlation value is summed over the number of subcodes, N_c.

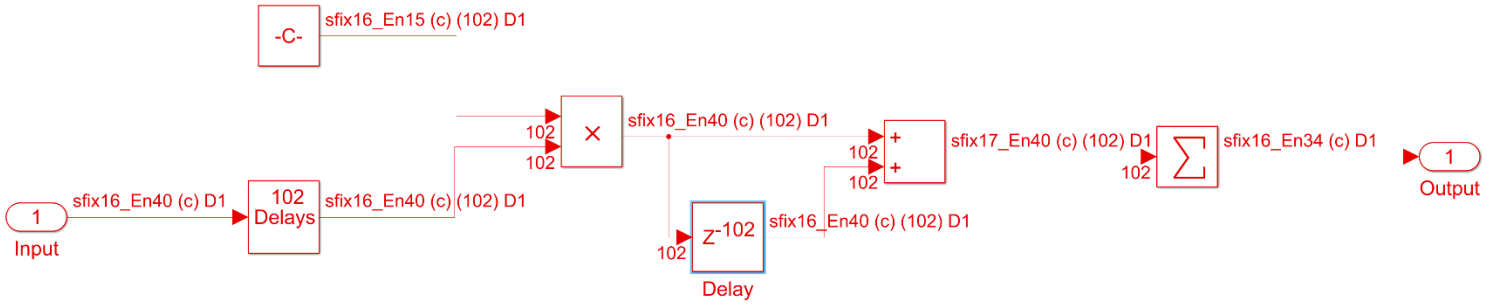

Figure 29. Range Compression Block Diagram Built in Simulink.

2. Synthesis / Implementation

Within Figure 30, the Range Compression model timing results are displayed for each synthesis type run. The graph depicts the WNS and WHS resulted in seven of eight positive numbers, which means that the timing of those models passed. However, the Flow_RuntimeOptimized run failed timing causing the entire synthesis and implementation of the model to fail. The WHS deviated by plus or minus 0.02ns throughout the graph whereas the WNS deviated by plus or minus 0.03ns until the noted failure.

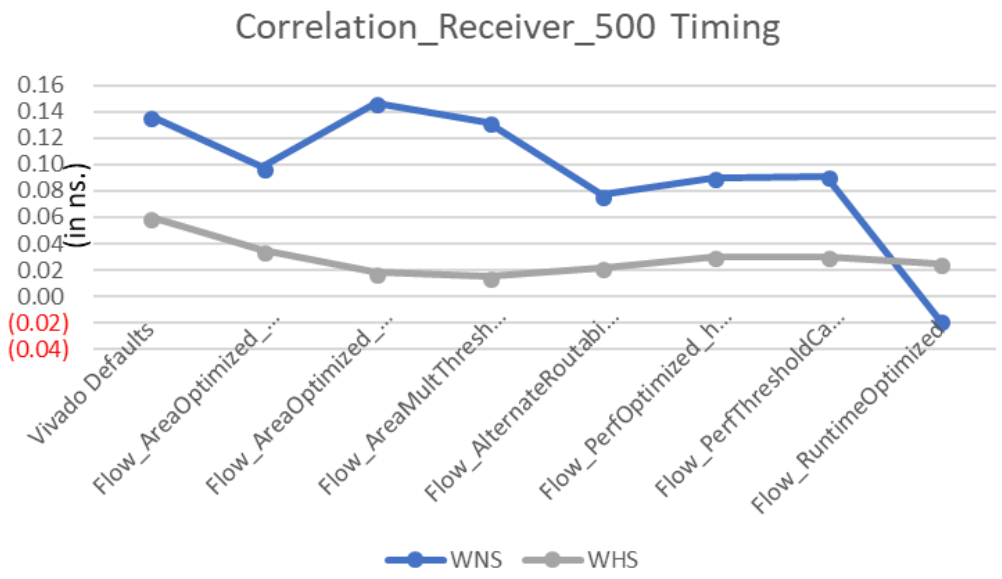

Figure 30. Range Compression Model Timing Results.

Within Figure 31, the Range Compression model power results are displayed for each synthesis type run. The total on-chip power graph shows a deviation of plus or minus 0.1 W. Within the power utilization comparison graph, many of the power components have minimal deviations except for the I/O component, which indicates that Vivado Defaults does not require power for I/O components. However, this is consistent with Figure 32 because the Vivado Defaults run does not require I/O logic space. The thermal power results are plus or minus 0.1 degrees Celsius.

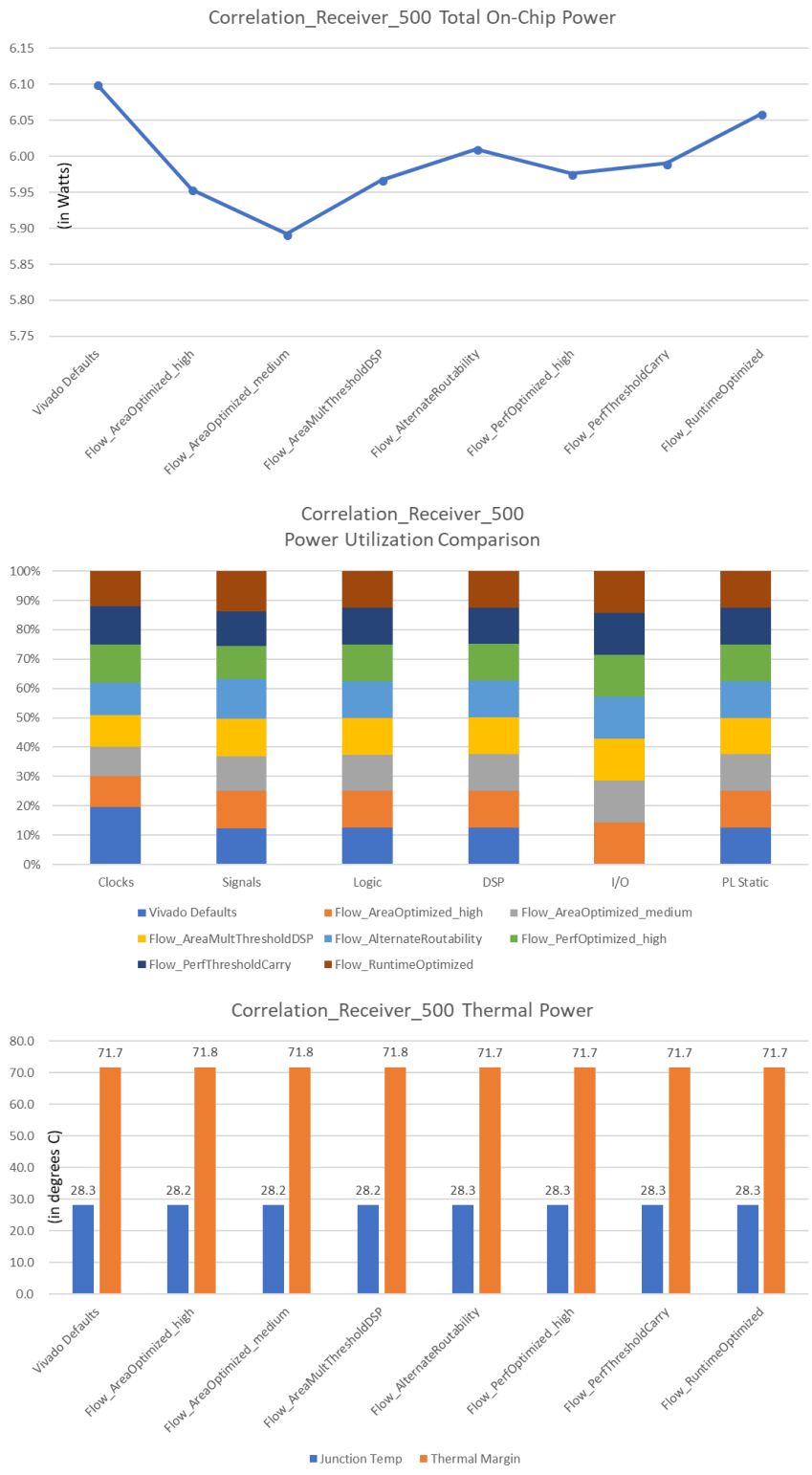

Figure 31. Range Compression Model Power Results.

Within Figure 32, the Range Compression model utilization results are displayed for each synthesis type run. The graphs depict the total and individual utilizations per run. To note, I/O and BUFG utilization are not provided for the Vivado Default run. These are outliers in comparison to the remaining runs in which use 68 I/O slices and one (up to two) BUFG slices. No correlations can be made between individual utilization graphs. The rise and fall in data can only be contributed to the actual synthesize type run. Figure 33 displays the absolute values of the Range Compression model results as simulated through MATLAB and Simulink. A delay is noted within the Simulink results due to slight modeling error.

Figure 32. Range Compression Model MATLAB and Simulink Results.

Range Compression

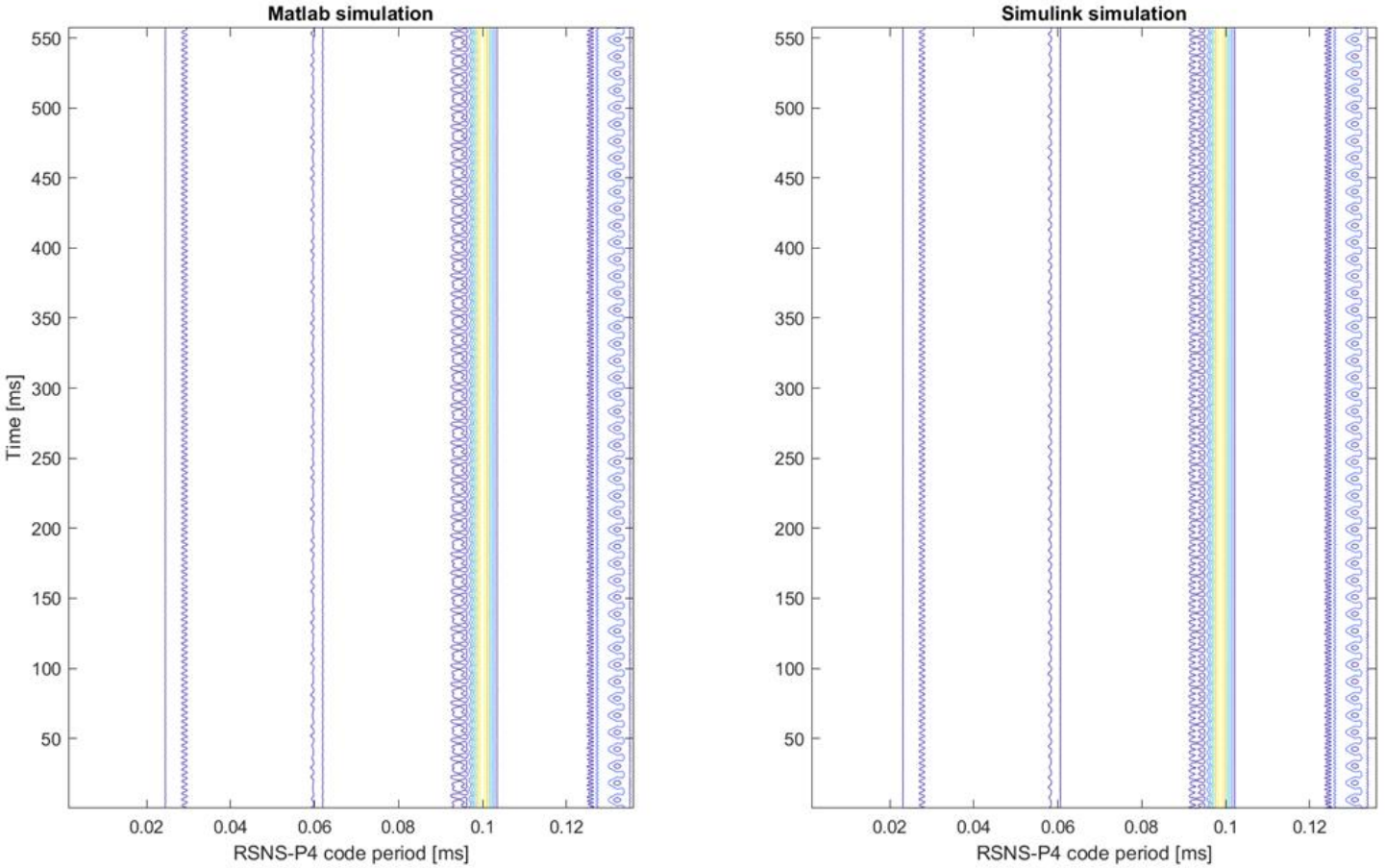

Figure 33. Range Compression Model MATLAB and Simulink Results.

C. DATA STORAGE

Project Name: DataStorage_500_vivado

Implementation Strategy: Peformance_Retiming

1. Model

The Data Storage model shown in Figure 34 is not considered an official step within the three-stage compression but it necessary within Simulink to ensure the buffered Range Compression signal is synchronized with the Doppler Filter range bins. This Data Storage model was synthesized and implemented with a target frequency of 500 MHz autonomously from all other models. This model takes the signal from the Range Compression model and then collects the data to produce a matrix and is output as a matrix transpose data stream. There are two data storages. One data storage is used as written storage for later use and the other is used as storage that is currently being read. These data storages cycle through each write or read only use accordingly after 4096 times 102 points are collected. Therefore, they can never write or read to the same storage consecutively – they must alternate.

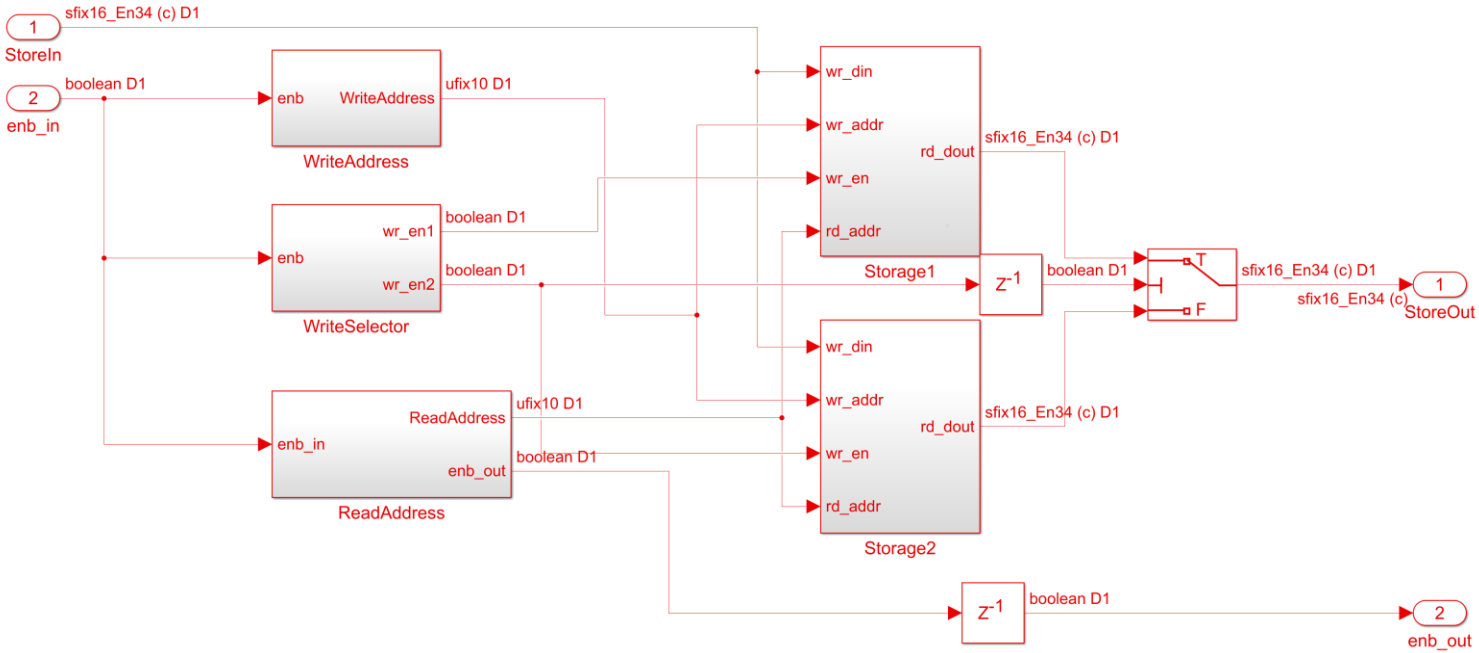

Figure 34. Data Storage Block Diagram Built in Simulink.

2. Synthesis / Implementation

Within Figure 35, the Data Storage model timing results are displayed for each synthesis type run. The graph depicts the WNS and WHS resulted in positive numbers, which means that the timing of the model passed. Both WNS the WHS remain constant with plus or minus 0.075ns and 0.025ns deviations respectively. No runs were at risk of failing timing.

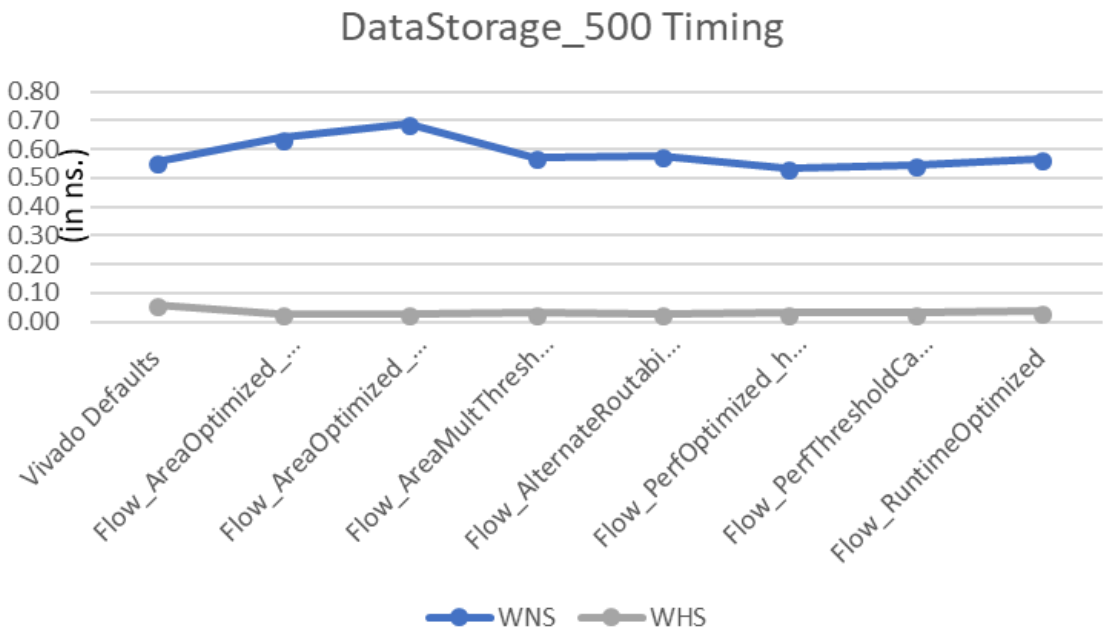

Figure 35. Data Storage Model Timing Results.

Within Figure 36, the Data Storage model power results are displayed for each synthesis type run. The total on-chip power graph shows a deviation of plus or minus 0.1 W except for Vivado Defaults which is 0.12W lower than the average of the other seven runs. Within the power utilization comparison graph, many of the power components have minimal deviations except for the I/O component which indicates that Vivado Defaults does not require power for I/O components. However, this is consistent with Figure 37 because the Vivado Defaults run does not require I/O logic space. The thermal power results are identical per run.

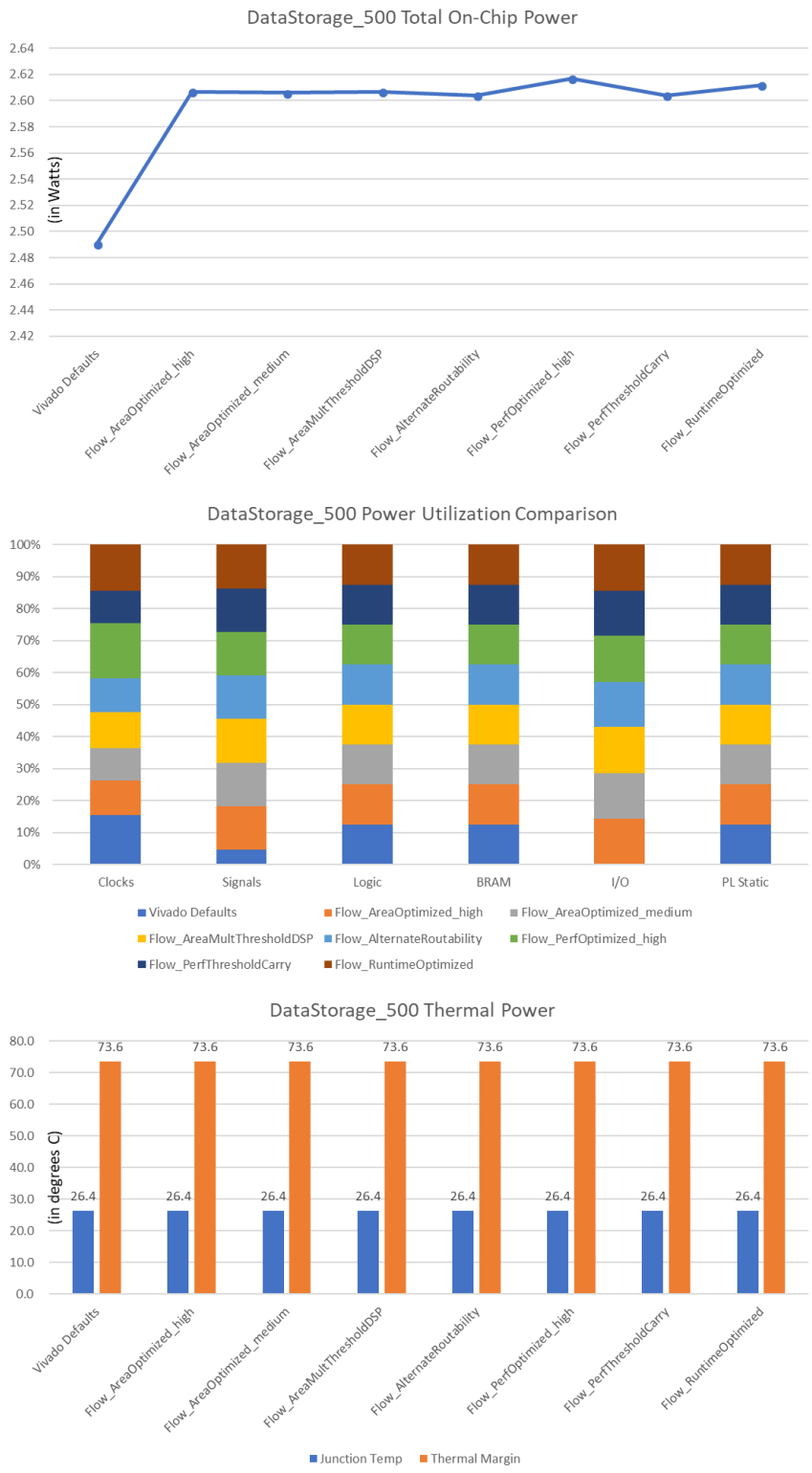

Figure 36. Data Storage Model Power Results.

Within Figure 37, the Data Storage model utilization results are displayed for each synthesis type run. The graphs depict the total and individual utilizations per run. To note, I/O and BUFG utilization are not provided for the Vivado Default run. These are outliers in comparison to the remaining runs in which use 70 I/O slices and 1 BUFG slice. There are no other variances within the Data Storage model for utilization.

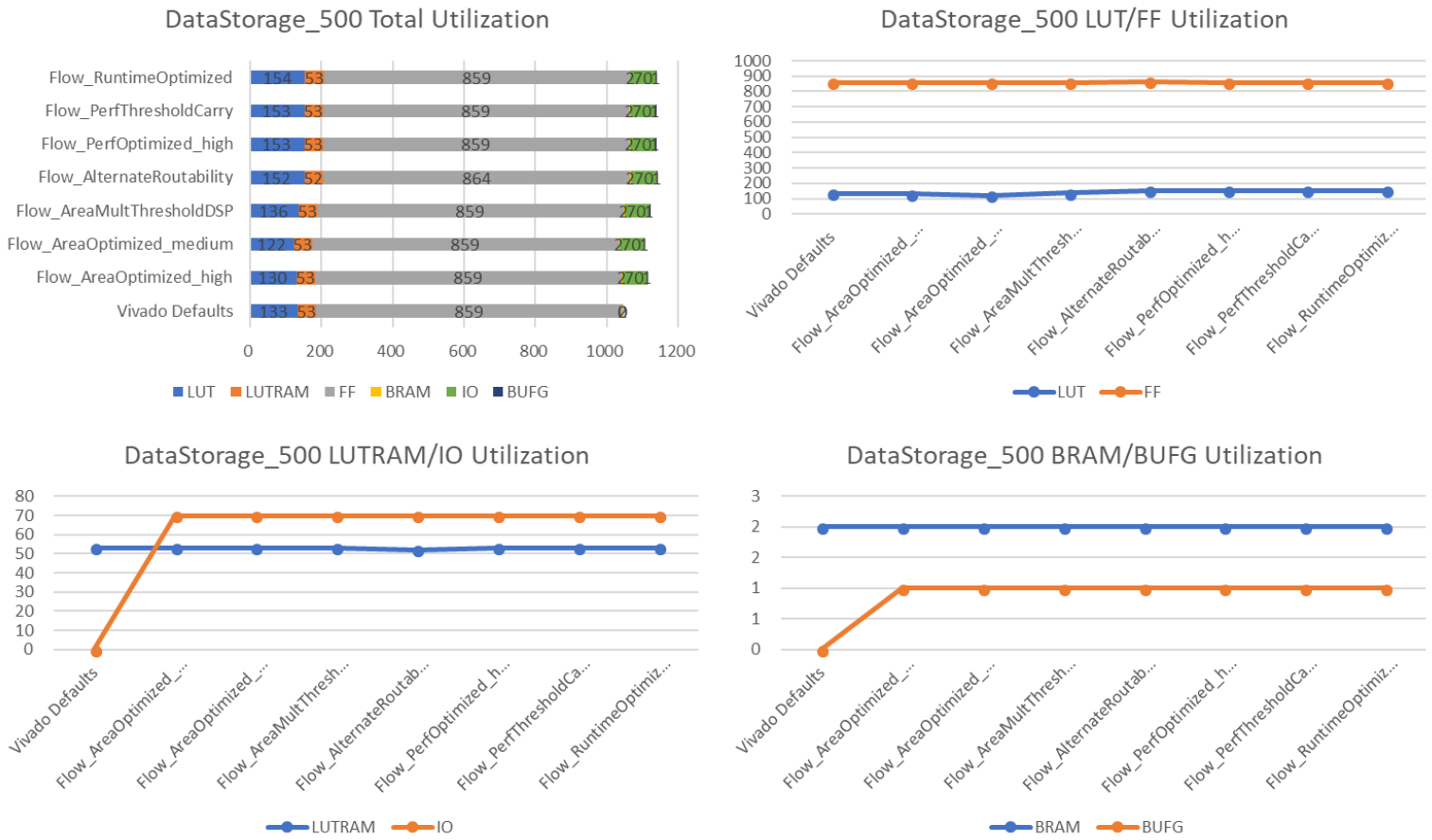

Figure 37. Data Storage Model Utilization Results.

D. DOPPLER FILTERING

Project Name: DopplerFilter_400_vivado

Implementation Strategy: Peformance_Retiming

1. Model

The Doppler Filtering model shown in Figure 38 is the second step within the three-stage compression. This Doppler Filtering model was synthesized and implemented with a target frequency of 400 MHz autonomously from all other models. This model takes the Data Storage model data stream and windows it with a Blackman window and performs an FFT to perform the Doppler filtering. The “convert” Simulink block is not needed; however, it is used to increase processing time and saves memory for the coherent integration.

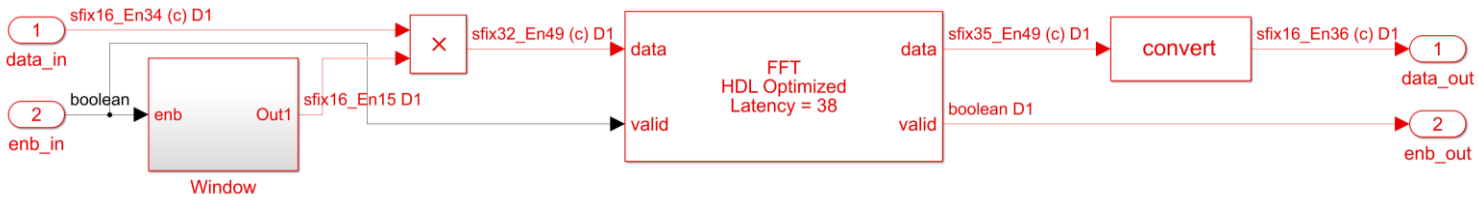

Figure 38. Doppler Filtering Block Diagram Built in Simulink.

2. Synthesis / Implementation

Within Figure 39, the Doppler Filtering model timing results are displayed for each synthesis type run. The graph depicts the WNS and WHS resulted in seven of eight positive numbers, which means that the timing of those models passed. However, the Flow_RuntimeOptimized run failed timing causing the entire synthesis and implementation of the model to fail. The WHS deviated by plus or minus 0.02ns throughout the graph whereas the WNS deviated by plus or minus 0.05ns until the noted failure.

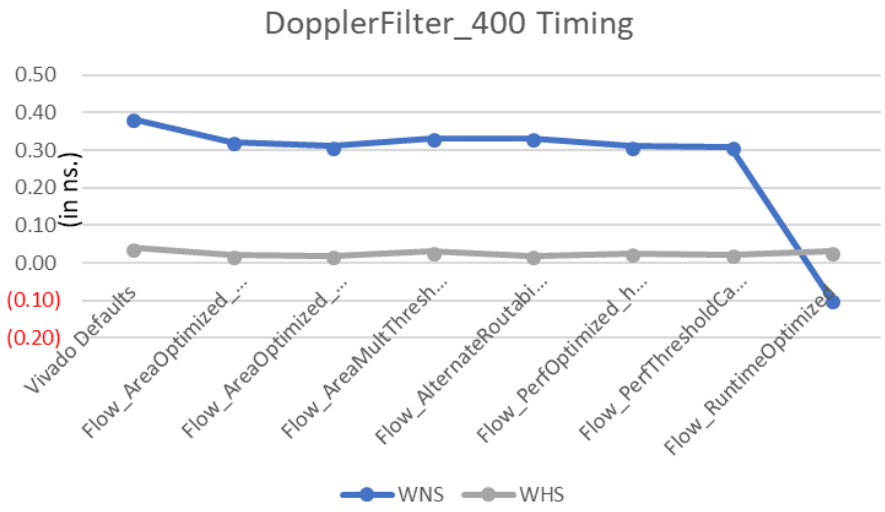

Figure 39. Doppler Filtering Model Timing Results.

Within Figure 40, the Doppler Filtering model power results are displayed for each synthesis type run. The total on-chip power graph shows a deviation of plus or minus 0.2W however extra power is required for Flow_PerfOptimized_high, Flow_PerfThresholdCarry, and Flow_RuntimeOptimized. This extra power requirement is outside the steady baseline set by the other five runs. Within the power utilization comparison graph, many of the power components have minimal deviations except for the I/O component which indicates that Vivado Defaults does not require power for I/O components. However, this is consistent with Figure 41 because the Vivado Defaults run does not require I/O logic space. The thermal power results are plus or minus 0.1 degrees Celsius.

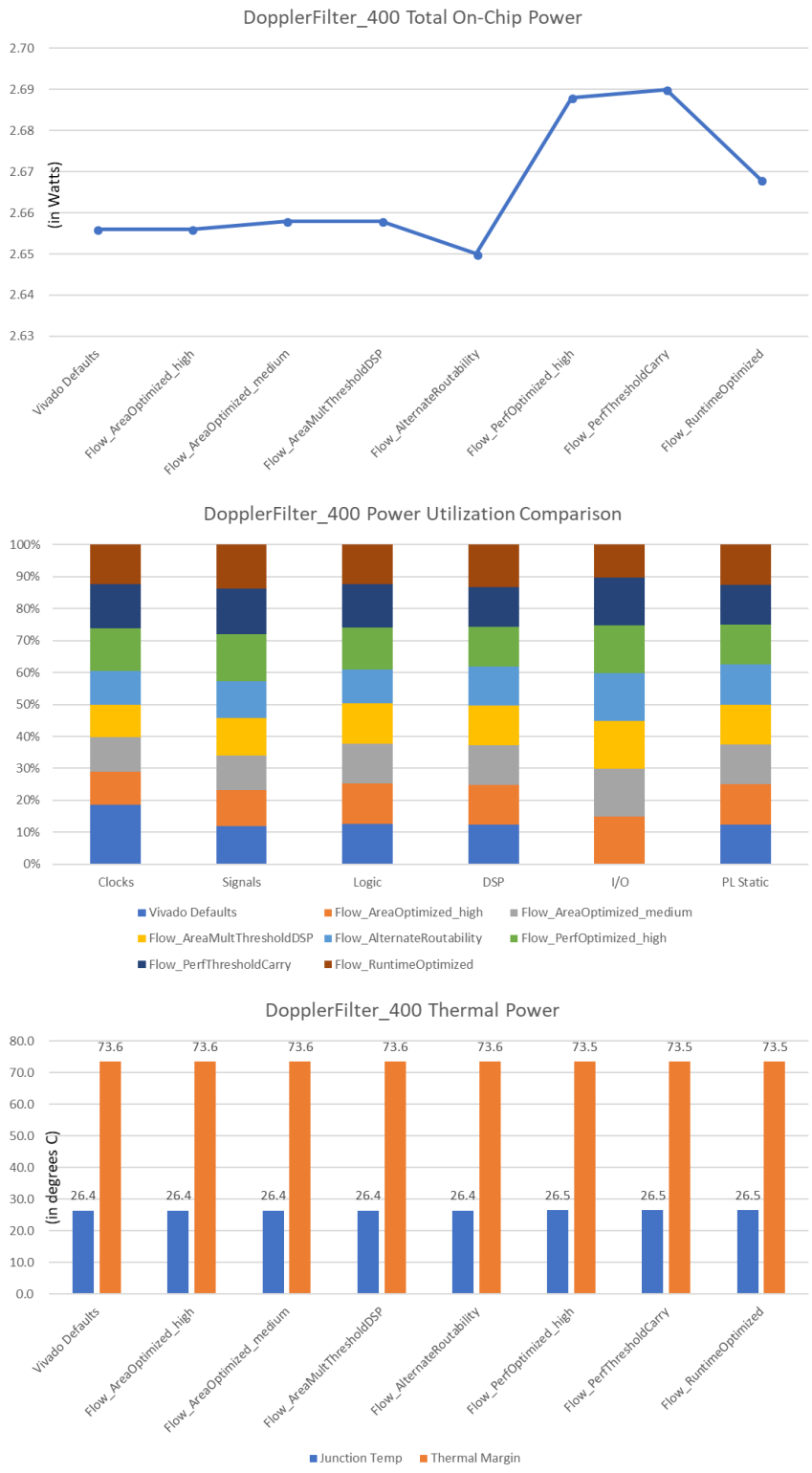

Figure 40. Doppler Filtering Model Power Results.

Within Figure 41, the Doppler Filtering model utilization results are displayed for each synthesis type run. The graphs depict the total and individual utilizations per run. To note, I/O and BUFG utilization are not provided for the Vivado Default run. These are outliers in comparison to the remaining runs in which use 70 I/O slices and one BUFG slice. A direct correlation can be made between LUTRAM, LUT, and FF. Within the graph, as less LUTRAM is used, the more LUT and FF are utilized based on the synthesis strategy. Figure 42 displays the absolute values of the Doppler Filtering model results as simulated through MATLAB and Simulink. Once again, a delay is noted within the Simulink results due to slight modeling error.

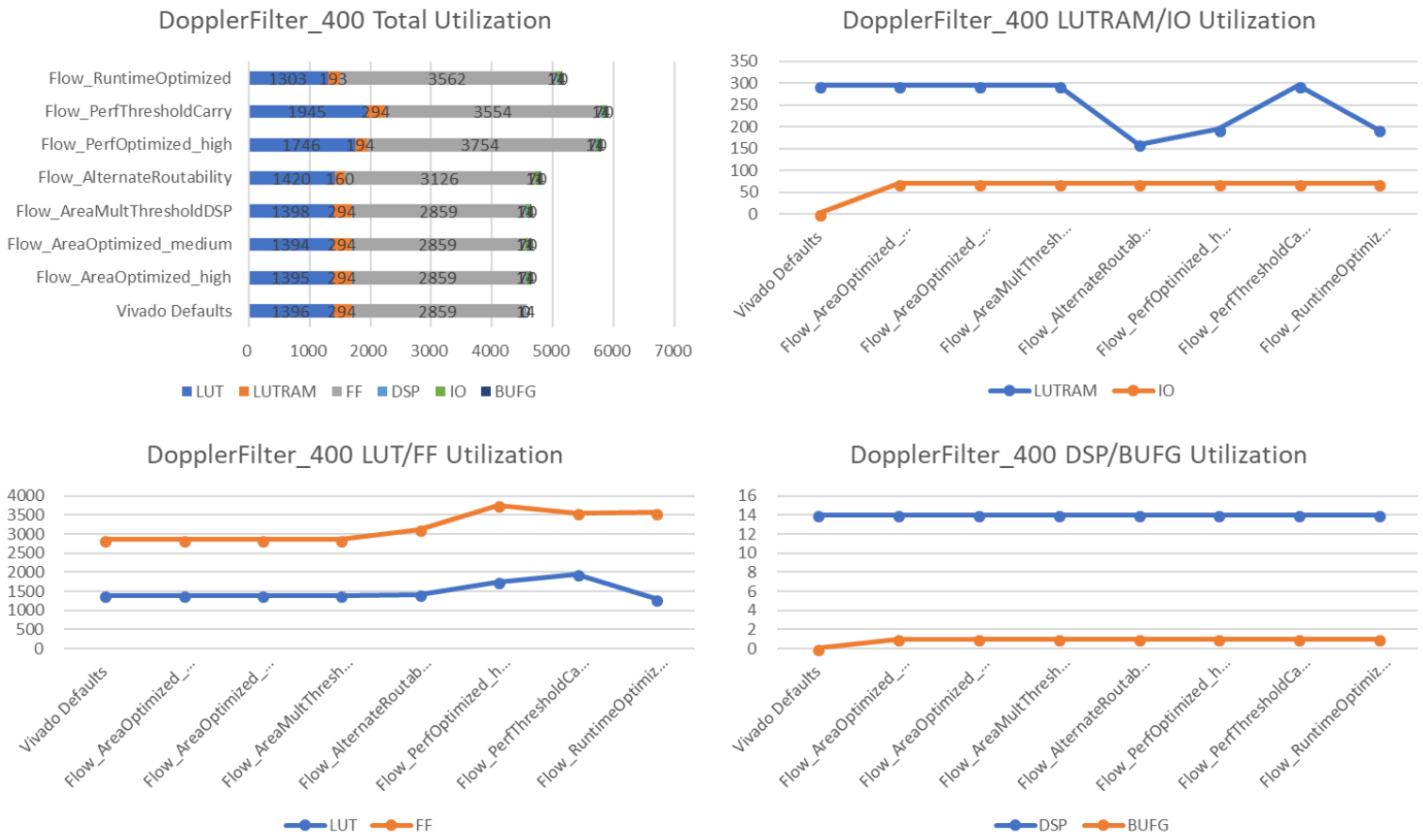

Figure 41. Doppler Filtering Model Utilization Results.

Doppler Filtering

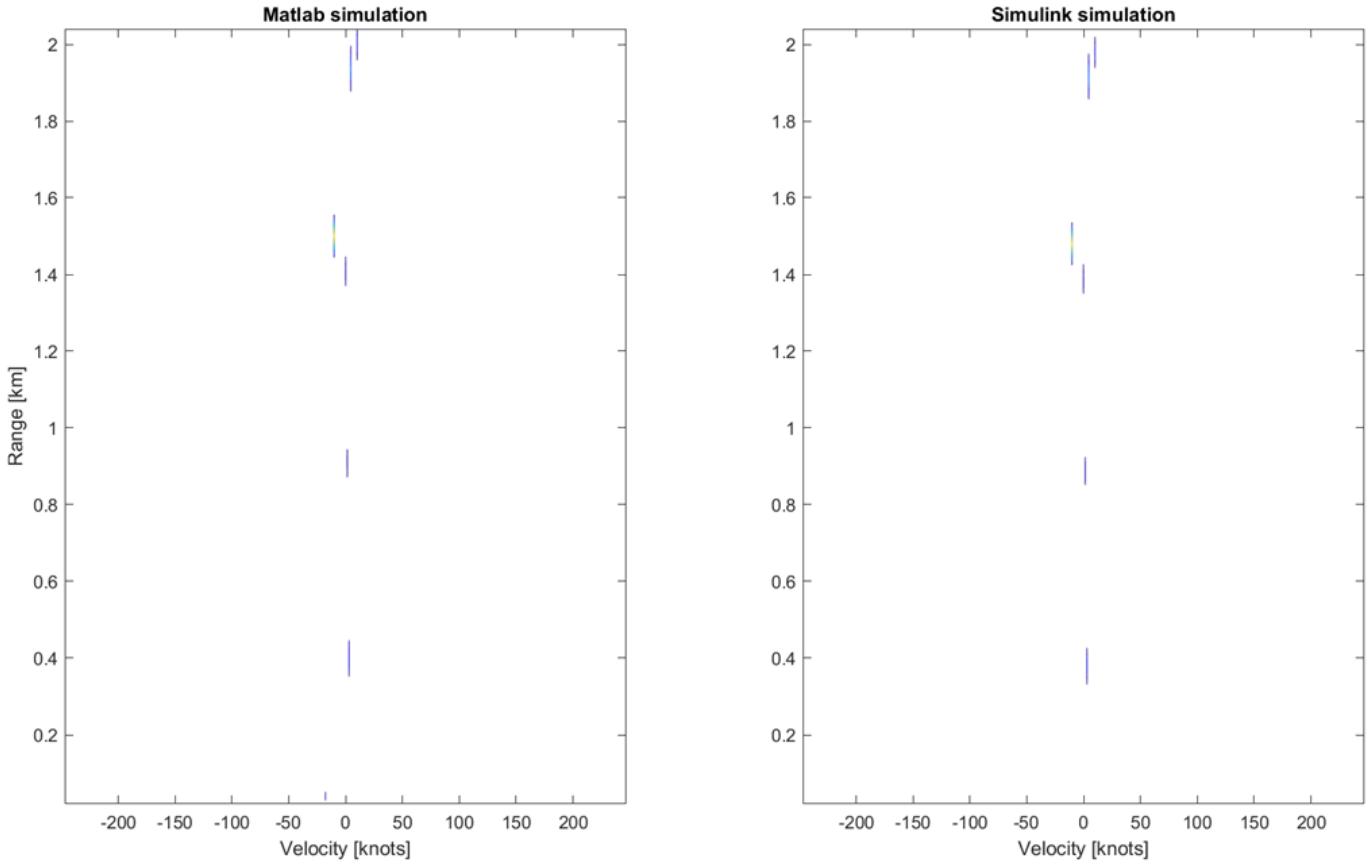

Figure 42. Doppler Filtering Model MATLAB and Simulink Results

E. COHERENT INTEGRATION

Project Name: CoherentIntegration_400_vivado

Implementation Strategy: Peformance_Retiming

1. Model

The Coherent Integration model shown in Figure 43 is the third step within the three-stage compression. This Coherent Integration model was synthesized and implemented with a target frequency of 400 MHz autonomously from all other models. This model takes the data stream from the Doppler filter and stores it into memory. Each correlating point from the next four Doppler maps are then added accordingly. The output then becomes the overall output for one moduli of the three-stage compression.

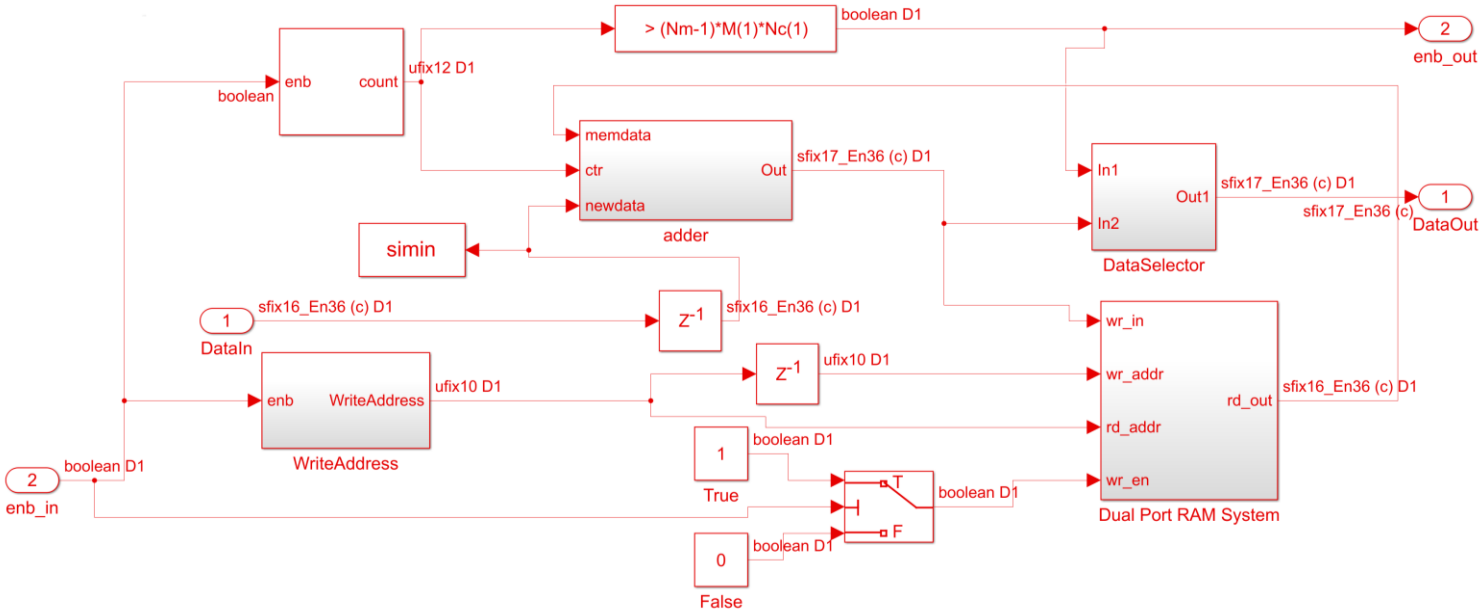

Figure 43. Coherent Integration Block Diagram Built in Simulink.

2. Synthesis / Implementation

Within Figure 44, the Coherent Integration model timing results are displayed for each synthesis type run. The graph depicts the WNS and WHS resulted in positive numbers, which means that the timing of the model passed. Both WNS the WHS remain constant with plus or minus 0.025ns deviations. No runs were at risk of failing timing.

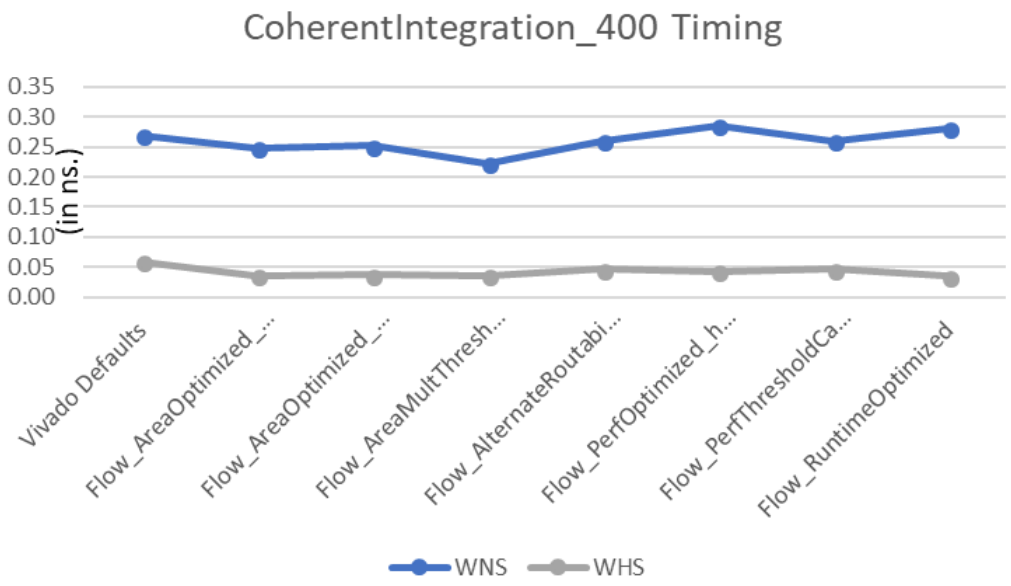

Figure 44. Coherent Integration Model Timing Results.

Within Figure 45, the Coherent Integration model power results are displayed for each synthesis type run. The total on-chip power graph shows a deviation of plus or minus 0.1W except for Vivado Defaults which is 0.06W lower than the average of the other seven runs. Within the power utilization comparison graph, many of the power components have minimal deviations except for the I/O component which indicates that Vivado Defaults does not require power for I/O components. However, this is consistent with Figure 46 because the Vivado Defaults run does not require I/O logic space. The power utilization graph also indicates that Vivado Defaults requires far less power for clock and signals components. The thermal power results are plus or minus 0.1 degrees Celsius.

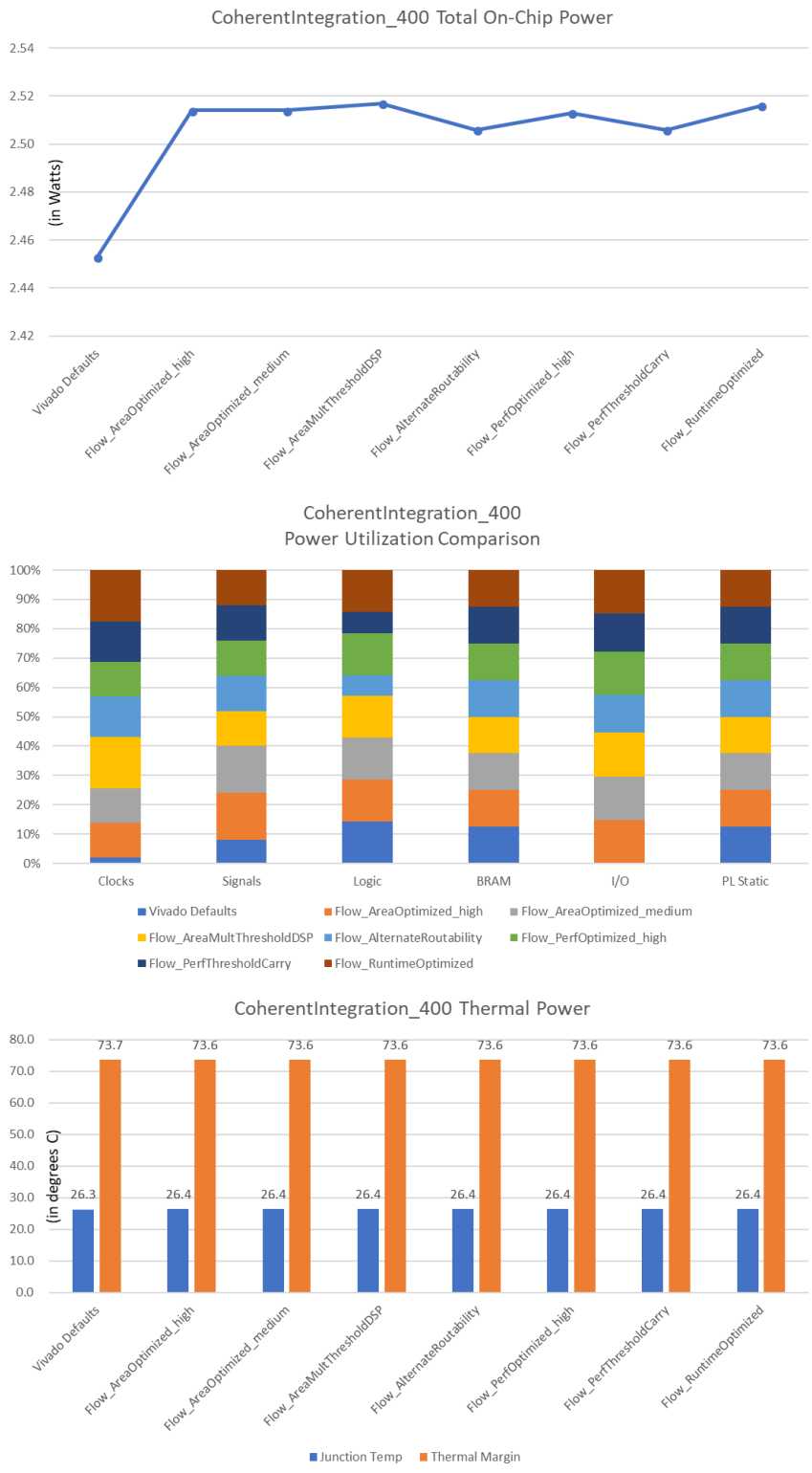

Figure 45. Coherent Integration Model Power Results.

Within Figure 46, the Coherent Integration model utilization results are displayed for each synthesis type run. The graphs depict the total and individual utilizations per run. To note, I/O and BUFG utilization are not provided for the Vivado Default run. These are outliers in comparison to the remaining runs in which use 72 I/O slices and one BUFG slice. A direct correlation can be made between LUT and the synthesis strategy. As expected, the less LUT is used as area is optimized during Flow_AreaOptimized_high and Flow_AreaOptimized_medium. Then as the strategy moves towards performance, the more resources on the FPGA are being utilized. Figure 47 displays the absolute values of the Coherent Integration model results as simulated through MATLAB and Simulink. Once again, a delay is noted within the Simulink results due to slight modeling error.

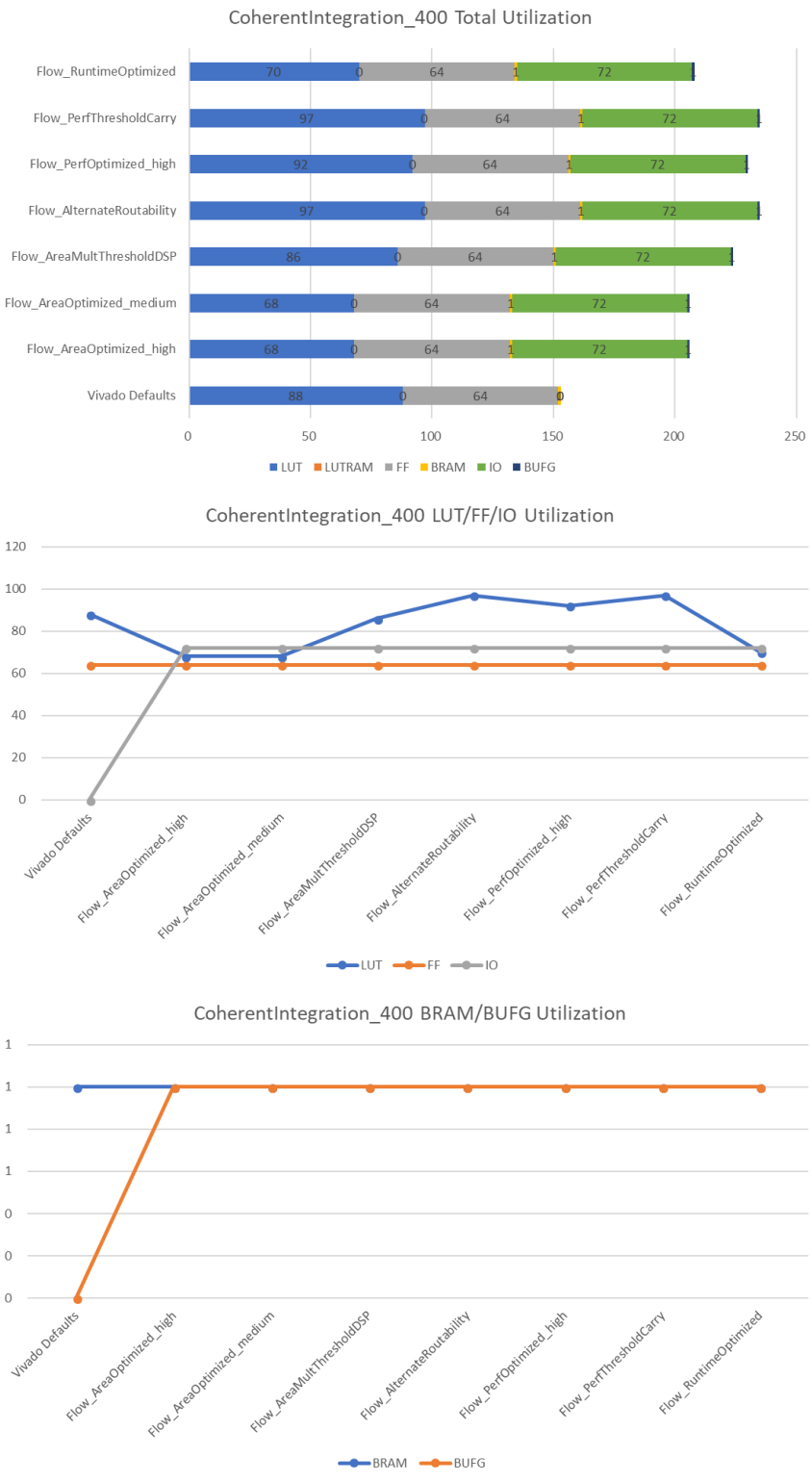

Figure 46. Coherent Integration Model Utilization Results.

Coherent Integration

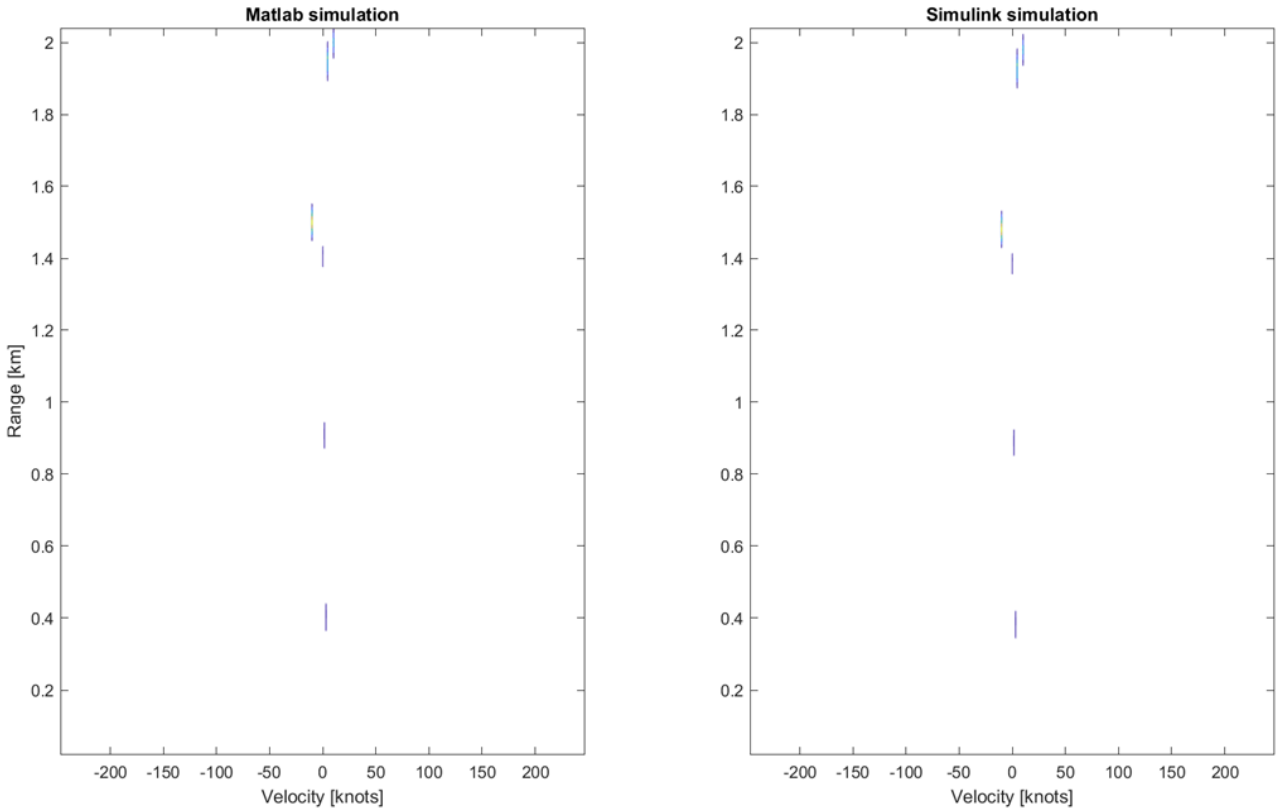

Figure 47. Coherent Integration Model MATLAB and Simulink Results

F. THREE-STAGE COMPRESSION

Project Name: alpha3StageCompression_100_vivado

Implementation Strategy: Peformance_Retiming

1. Model

The Three-stage Compression model shown in Figure 48 is the top-level view consisting of the subsystem block diagram shown in Figure 49. This subsystem comprises the Range Compression, Data Storage, Doppler Filtering, and Coherent Integration models previously discussed. This Three-stage Compression model was synthesized and implemented with a target frequency of 100 MHz autonomously from all other models. To ensure this model is adapted for LPI detection, it must be duplicated for three separate moduli running in parallel of each other. This model inputs a receiver for an RSNS-P4 signal that was scripted in MATLAB and is converted into Simulink. Therefore, a data stream is stepped through the previous models (minus DRFM) with the use of Boolean operators that act as miniature reference points that ensure the data stream remains in proper step throughout all models.

Figure 48. Top Level Block Diagram of Three-stage Compression Built in Simulink.

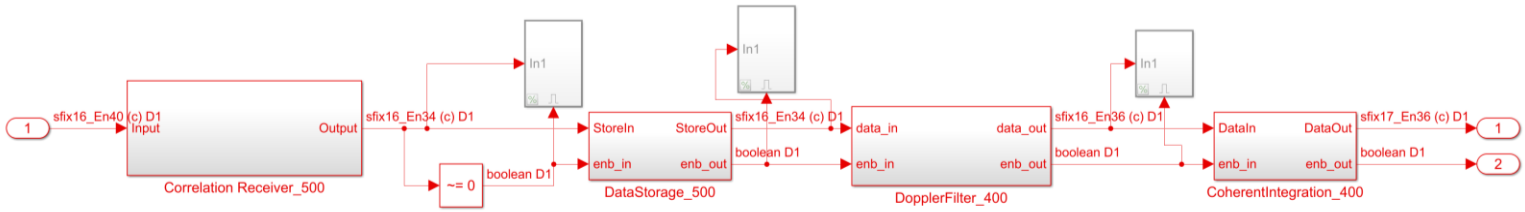

Figure 49. Three-stage Compression Block Diagram Built in Simulink.

2. Synthesis / Implementation

Within Figure 50, the Three-stage Compression model timing results are displayed for each synthesis type run. The graph depicts the WNS and WHS resulted in positive numbers which means that the timing of the model passed. Both WNS the WHS remain constant. WNS retained a plus or minus 0.4ns deviation whereas WHS shows no deviation due to its consistency across runs. No runs were at risk of failing timing. However, when this model was run at a target frequency of 200 MHz, the entire model failed timing across all runs.

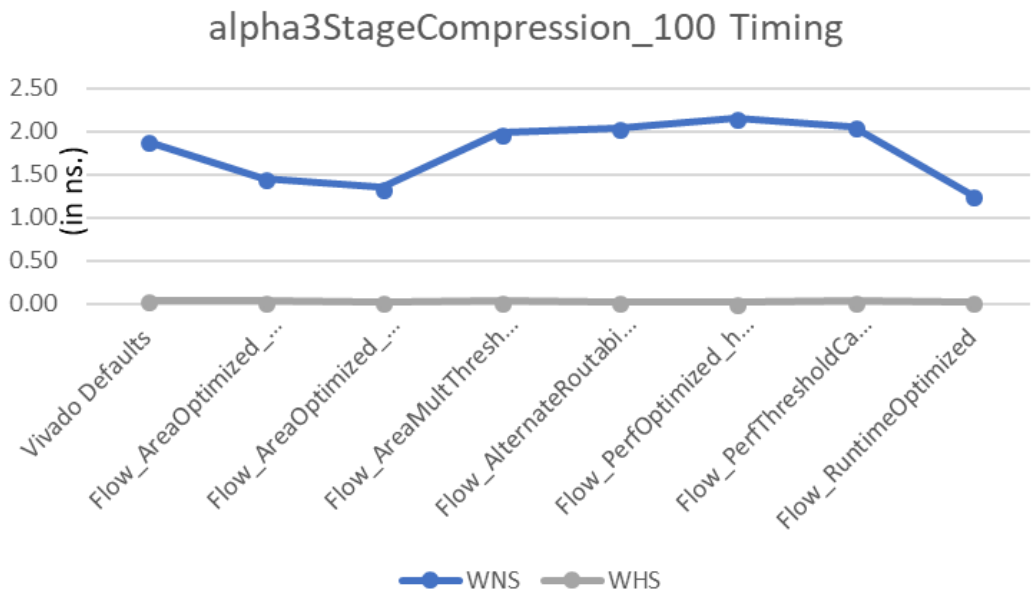

Figure 50. Three-stage Compression Model Timing Results.

Within Figure 51, the Three-stage Compression model power results are displayed for each synthesis type run. The total on-chip power graph shows a deviation of plus or minus 0.0275W. Overall, the power is consistent between runs. Within the power utilization comparison graph, many of the power components have minimal deviations except for the I/O component, which indicates that Vivado Defaults does not require power for I/O components. However, this is consistent with Figure 52 because the Vivado Defaults run does not require I/O logic space. The thermal power results are plus or minus 0.1 degrees Celsius.

Figure 51. Three-stage Compression Model Power Results.

Within Figure 52, the Three-stage Compression model utilization results are displayed for each synthesis type run. The graphs depict the total and individual utilizations per run. To note, I/O and BUFG utilization are not provided for the Vivado Default run. These are outliers in comparison to the remaining runs in which use 71 I/O slices and two BUFG slices. A direct correlation can be made between LUT and the synthesis strategy. As expected, the less LUT is used as area is optimized during Flow_AreaOptimized_high and Flow_AreaOptimized_medium. Then as the strategy moves towards performance, the more resources to include FFs as well as LUT on the FPGA are being utilized. Figure 53 displays the contour plot of the absolute values of the RSNS-P4 signal input into the Three-Stage Compression model as simulated through MATLAB and Simulink. Once again, a delay is noted within the Simulink results due to slight modeling error.

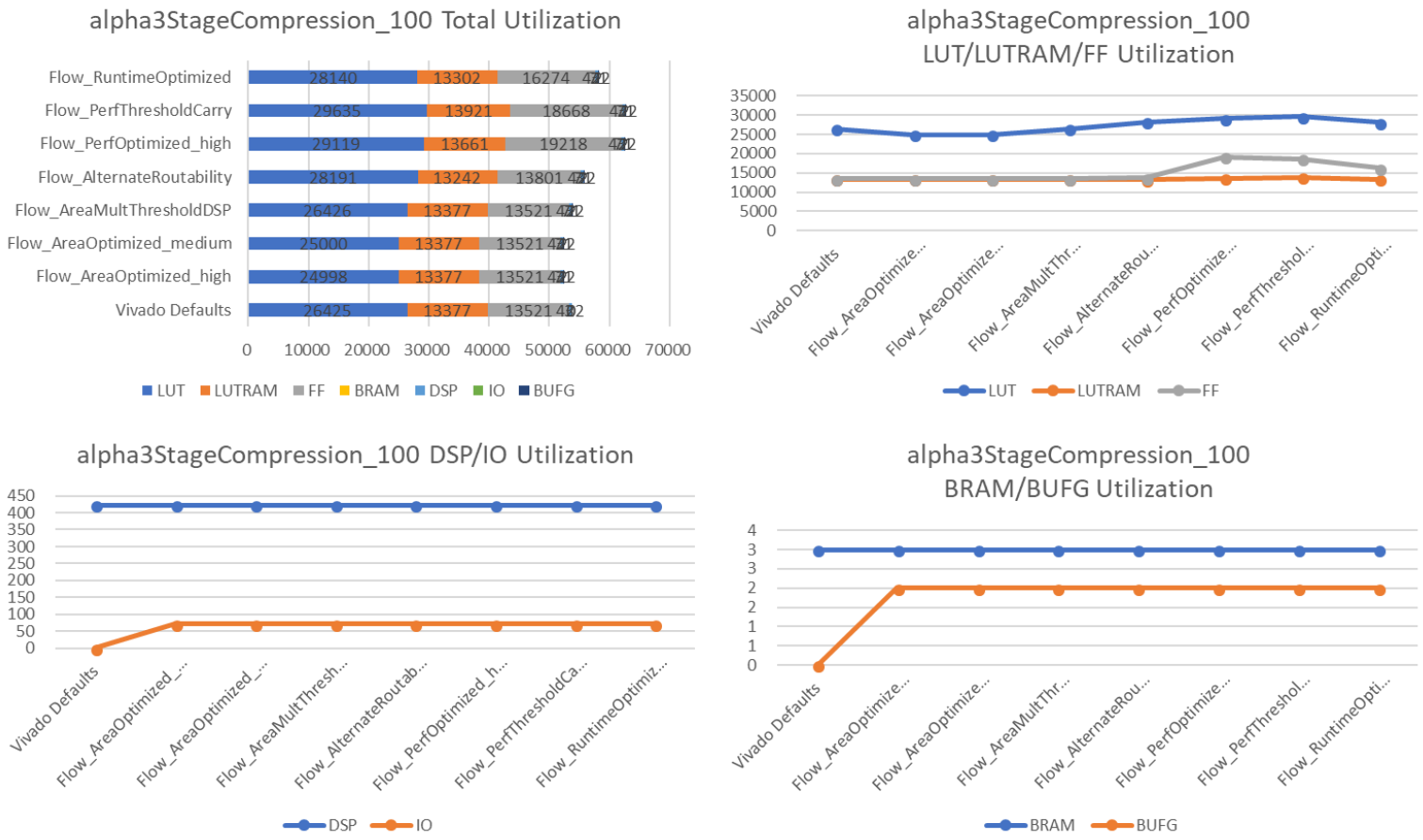

Figure 52. Three-stage Compression Model Utilization Results.

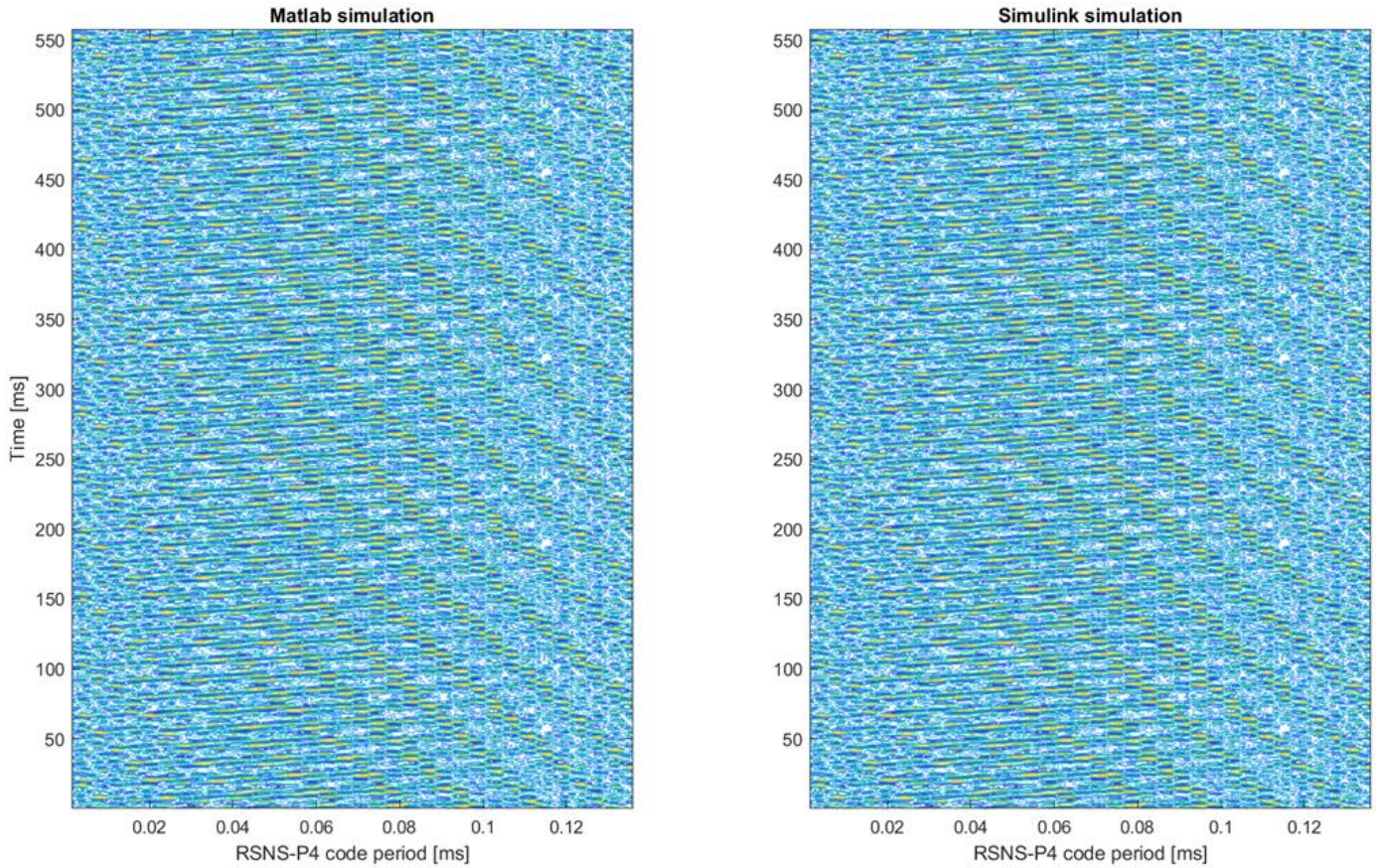

Figure 53. Input RSNS-P4 Test Signal through the MATLAB and Simulink Three-Stage Compression Model.

G. CHAPTER SUMMARY

Within this chapter, the modeling, synthesis, and implementation results were presented. These results were compiled using the Simulink modeling and Vivado synthesis and implementation tools. The data was depicted using visual graphs in which help the user pinpoint the inflections within the data for appropriate analysis. Further analysis of the data can be done using the Vivado Data Captures in the Appendix. Each model reacted differently to each synthesis strategy; however, the differences across all strategies for a single model appeared to be minimal for smaller systems such as these. The differences between a Vivado Default run and the remaining runs per model were made clear as well.

VI. CONCLUSION AND RECOMMENDATIONS

This thesis involved two sequential objectives. The first objective was to generate two Simulink digital systems: the DRFM and three-stage compression. The second objective was to synthesize and implement the given models within Xilinx Vivado software (based on area, power, or timing) to ensure the designs would precisely implement onto a Xilinx Ultrascale+ FPGA.

In this thesis, given pre-existing MATLAB code simulating the desired outputs, comparable models were designed in Simulink of two digital systems. These designed Simulink models consisted of individual components for the three-stage compression including but not limited to the range compression, Doppler filtering, and coherent integration as well as the overall designs for both the DRFM and three-stage compression. The overall designed models, however, are not complete. To finish the DRFM model, it must be integrated with a CORDIC, microprocessor, and piped (connected) with additional identical designs, which feed into one output for the desired deception outcome. To finish the three-stage compression, two additional models need to be created, a model for each moduli, and run in parallel of each other to subsequently decorrelate clutter which will maximize the unambiguous range and range resolution while suppressing the false alarm rate of returns.

The synthesis and implementation of the models with Xilinx Vivado software was also investigated in this thesis. The individual sub systems were each run separately. The concept was to compare the results of varying synthesis strategies while the implementation strategy remained the same as a form of control. This resulted in the understanding that the target frequency at which the FPGA was clocked would vary depending on the model. For sub systems, they were clocked at 400-500 MHz, however, the total three-stage compression system could only clock at 100 MHz. Through research, it was found that although the overall system clocked significantly lower, the ability to manually change the desired routing within the model can be done by an experienced FPGA designer resulting in higher clock speeds. Furthermore, it was noted that Vivado Default runs had significant differences in comparison to the controlled strategies.

Therefore, the default results should be used as a guide to overall synthesis and implementation strategy and not as a standalone end strategy.

Going forward, each model must be complete. To do so, additional training would be required for the user. Table 8 is a list of recommended training courses that would help facilitate successful designs. These courses are listed from basic to advanced and it is highly recommended that the most novice user attend all listed courses to facilitate the advanced designs required for proper DRFM and LPI radar operations. Once the models are complete, implementation onto the FPGA and testing is required for further interfacing into the analog portions of the overall systems.

Table 8. Recommended Training Courses.

MathWorks	Xilinx
MATLAB Fundamentals (3 days)	Designing FPGAs Using the Vivado Design Suite 1 (2 days)
Simulink for System and Algorithm Modeling (2 days)	Designing FPGAs Using the Vivado Design Suite 2 (2 days)
Signal Processing with Simulink (3 days)	Designing FPGAs Using the Vivado Design Suite 3 (2 days)
Verification and Validation of Simulink Models (1 day)	Designing FPGAs Using the Vivado Design Suite 4 (2 days)
Simulink Model Management and Architecture (2 days)	Designing with the UltraScale and UltraScale+ Architectures (2 days)
Generating HDL Code from Simulink (2 days)	Xilinx Partial Reconfiguration Tools & Techniques (2 days)
DSP for FPGAs (3 days)	

APPENDIX. VIVADO DATA CAPTURES

A. DRFM MODEL DATA

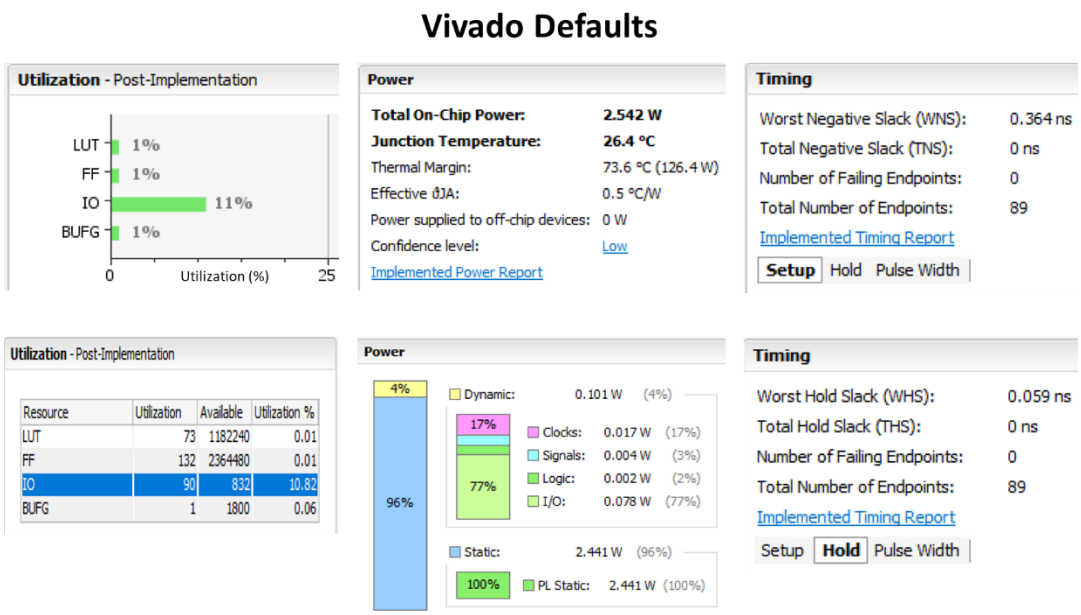

Figure 54. Vivado Default Results for DRFM Model.

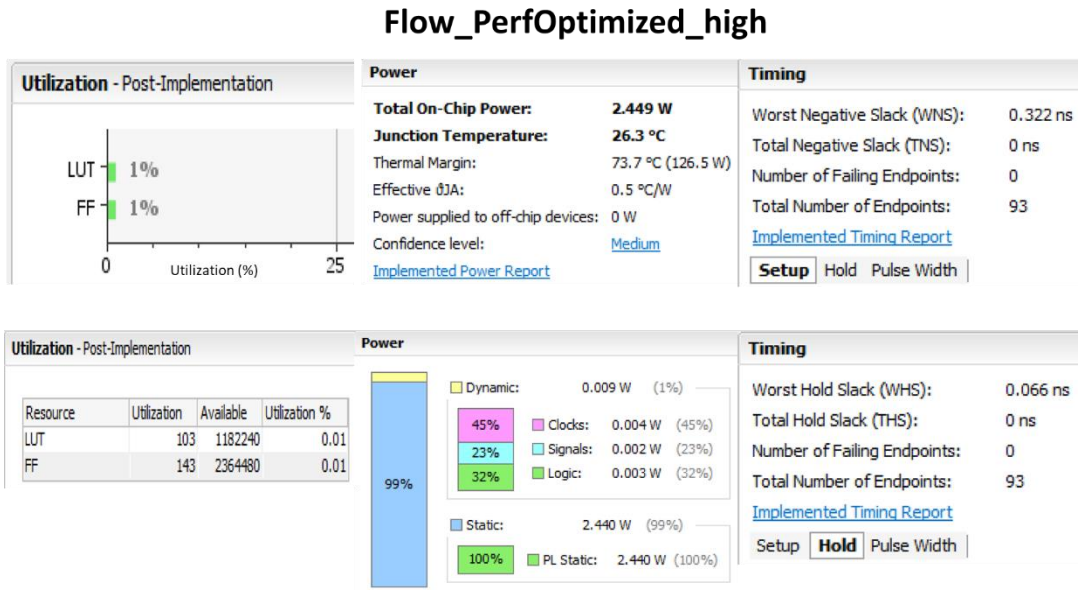

Figure 55. Vivado Flow_PerfOptimized_high Results for DRFM Model.

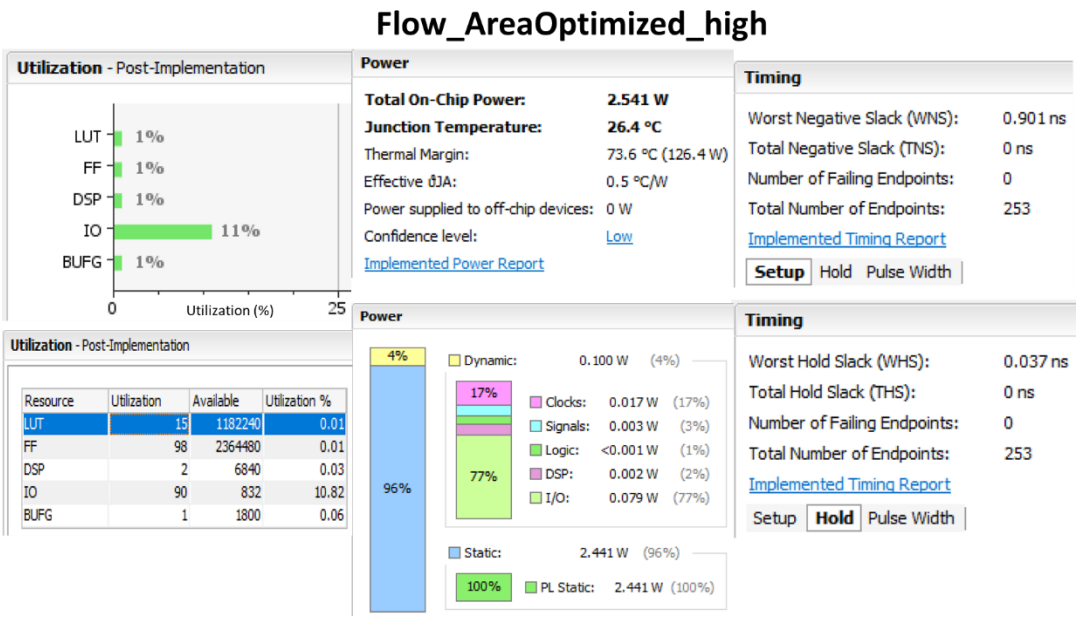

Figure 56. Vivado Flow_AreaOptimized_high Results for DRFM Model.

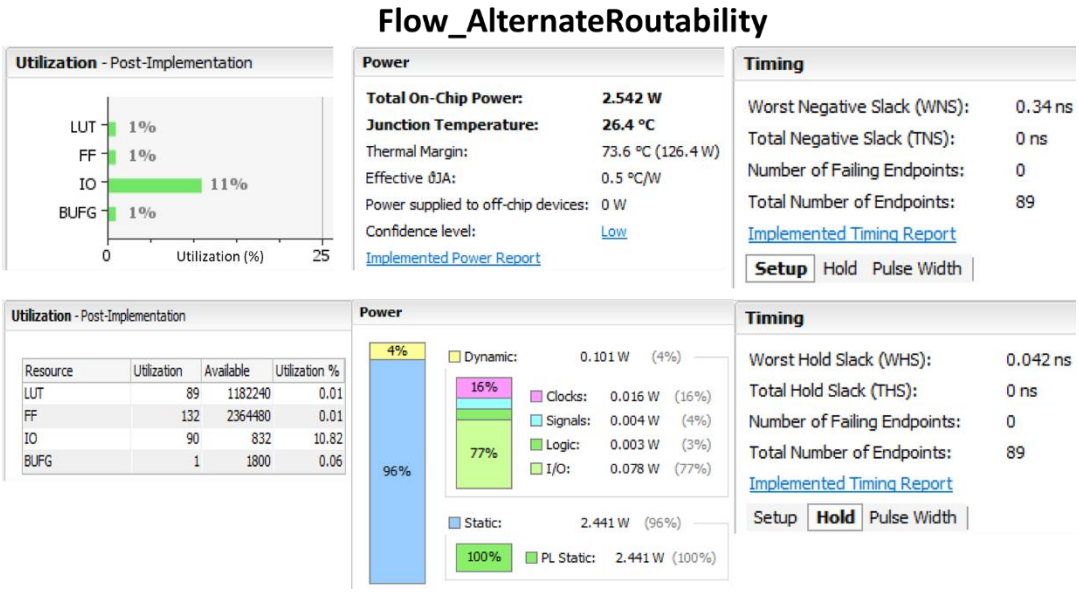

Figure 57. Vivado Flow_AlternateRoutability Results for DRFM Model.

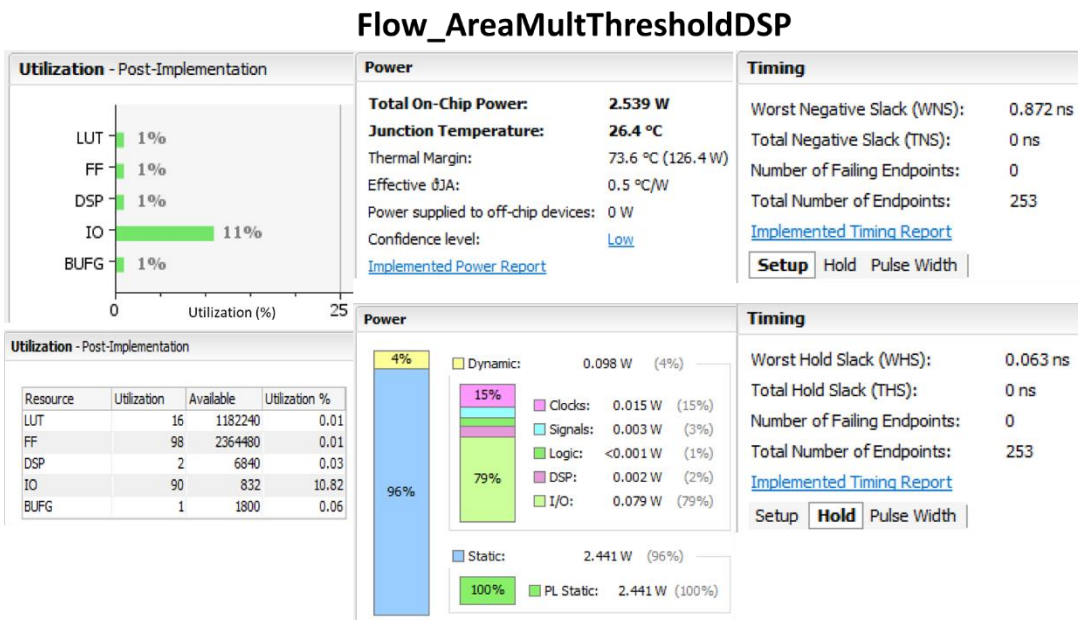

Figure 58. Vivado Flow_AreaMultThresholdDSP Results for DRFM Model.

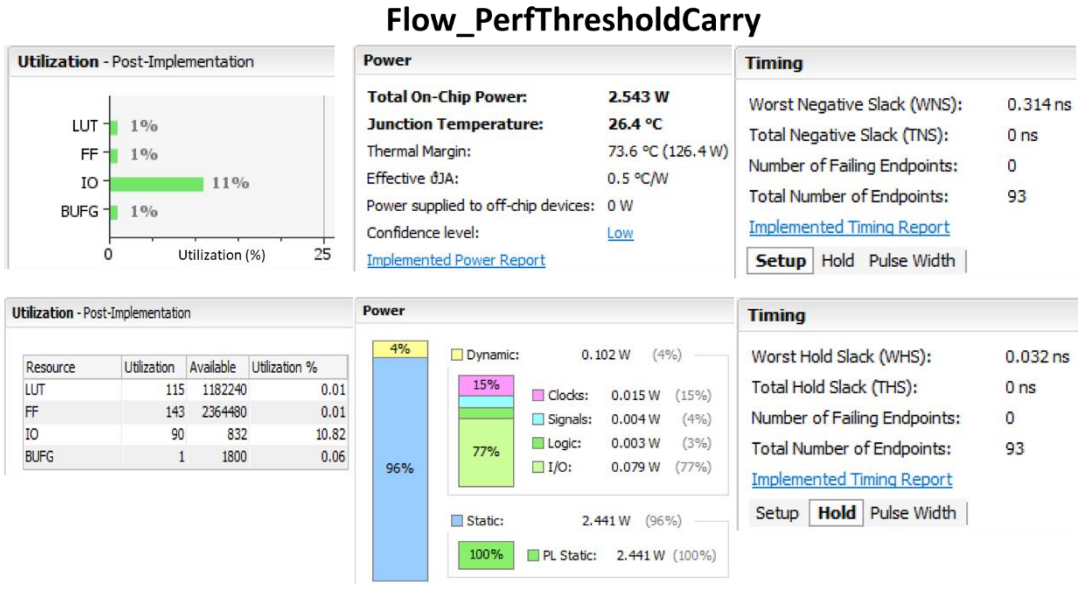

Figure 59. Vivado Flow_PerfThresholdCarry Results for DRFM Model.

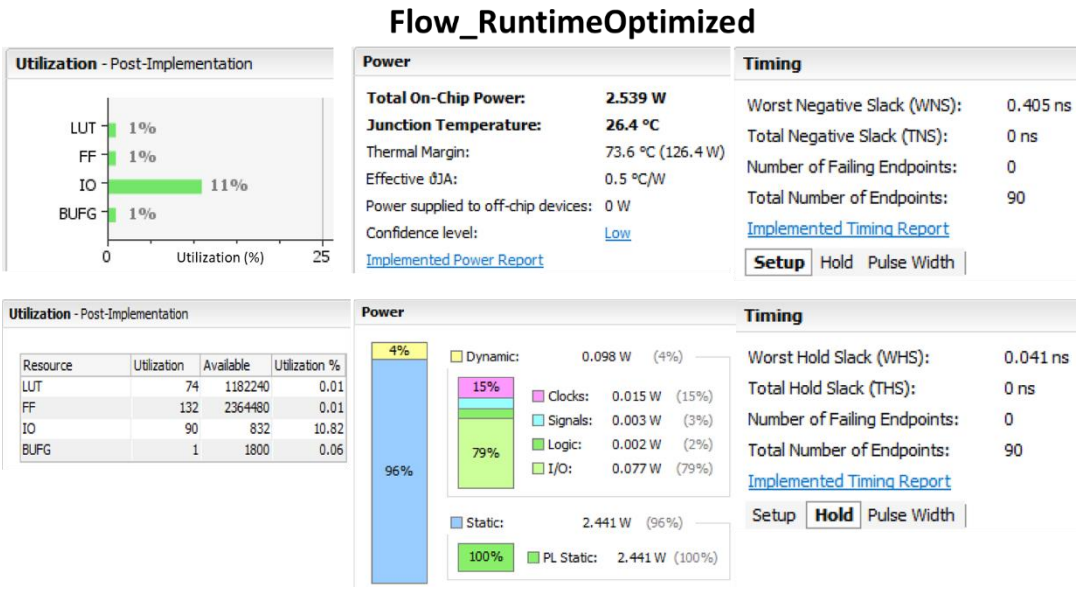

Figure 60. Vivado Flow_RuntimeOptimized Results for DRFM Model.

B. RANGE COMPRESSION MODEL DATA

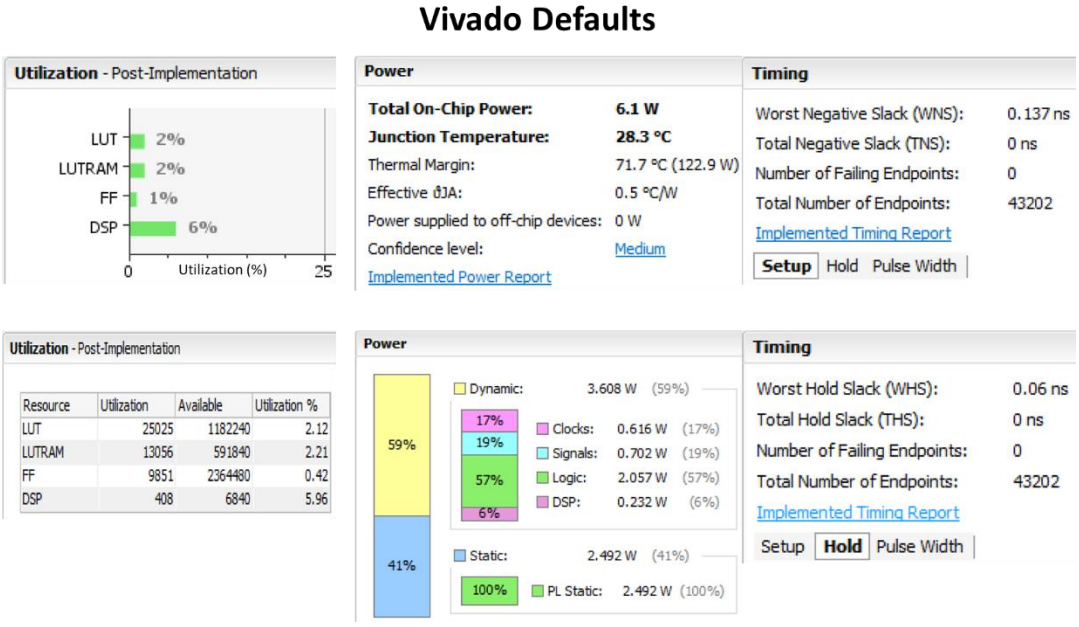

Figure 61. Vivado Default Results for Range Compression Model.

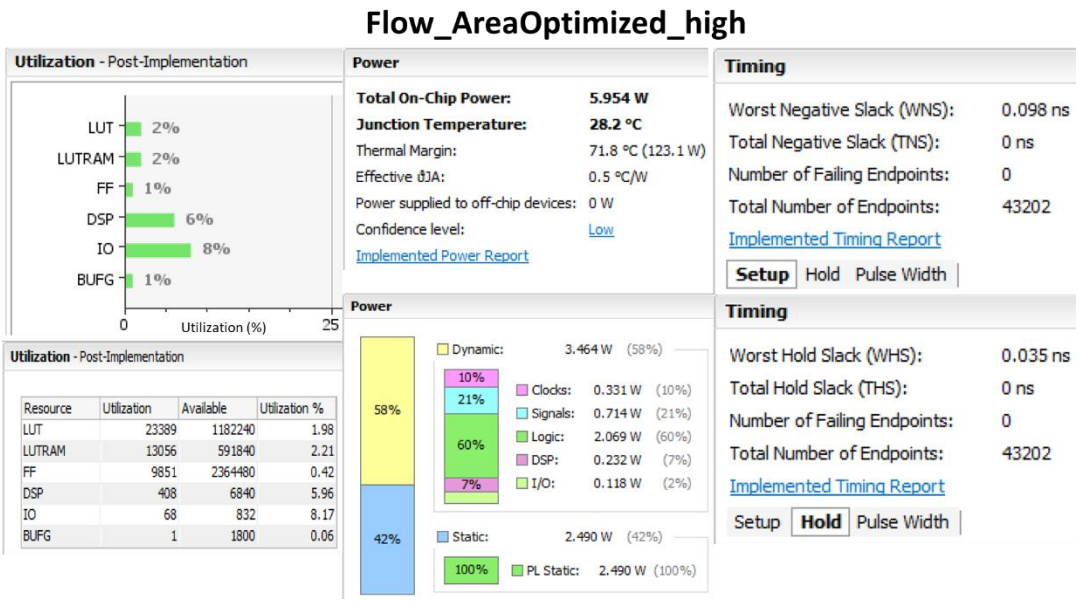

Figure 62. Vivado Flow_AreaOptimized_high Results for Range Compression Model.

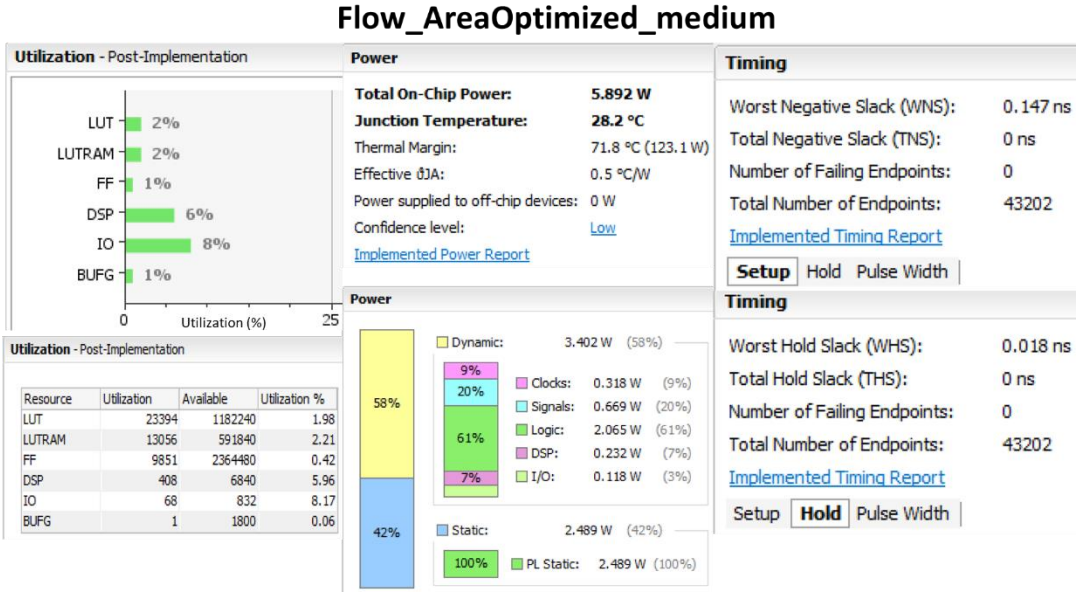

Figure 63. Vivado Flow_AreaOptimized_medium Results for Range Compression Model.

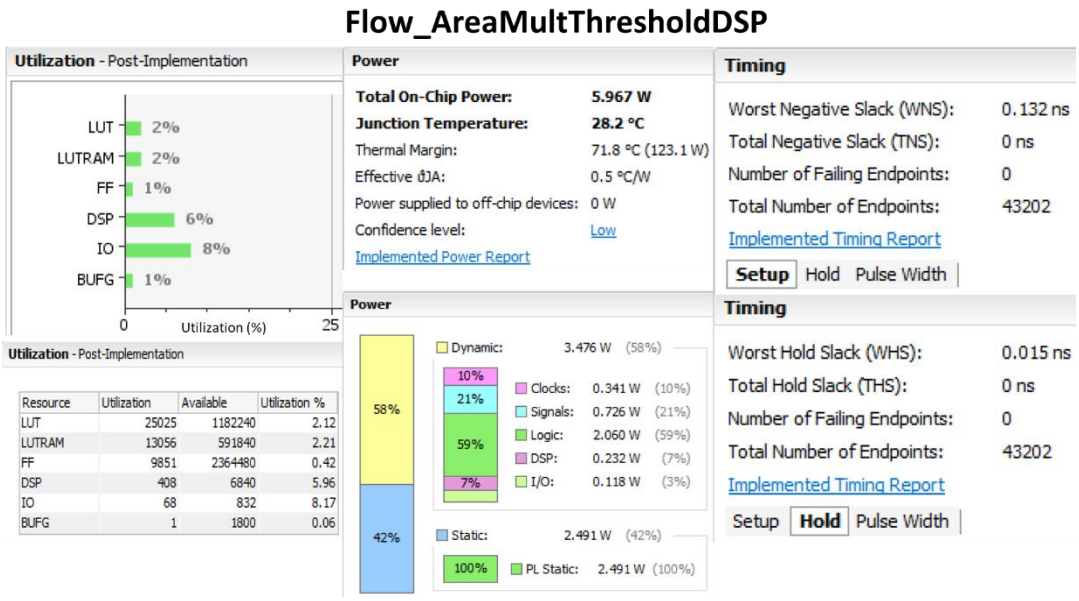

Figure 64. Vivado Flow_AreaMultThresholdDSP Results for Range Compression Model.

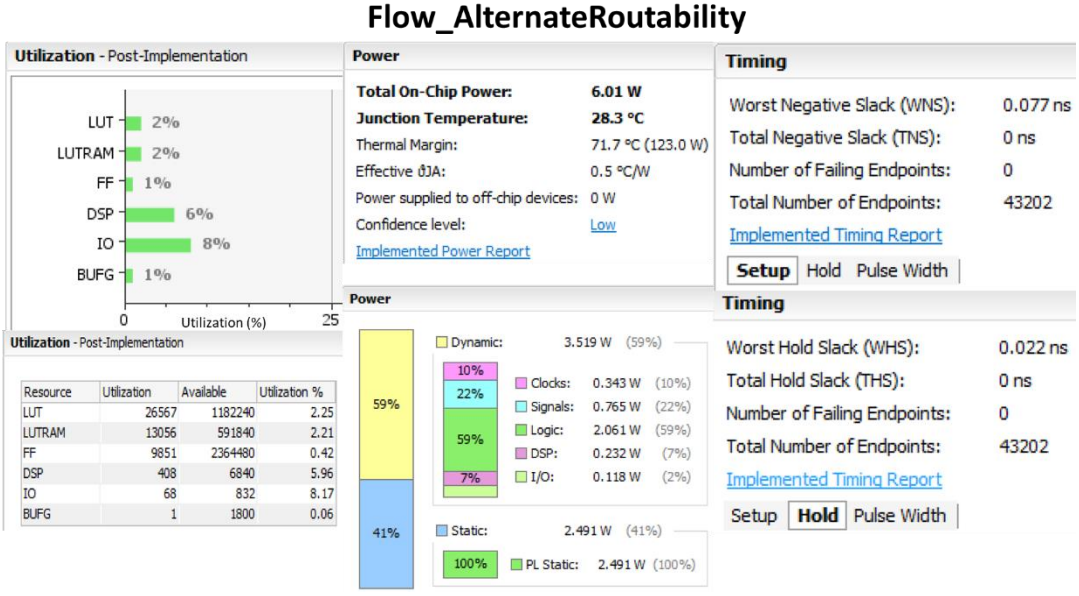

Figure 65. Vivado Flow_AlternateRoutability Results for Range Compression Model.

Flow_PerfOptimized_high

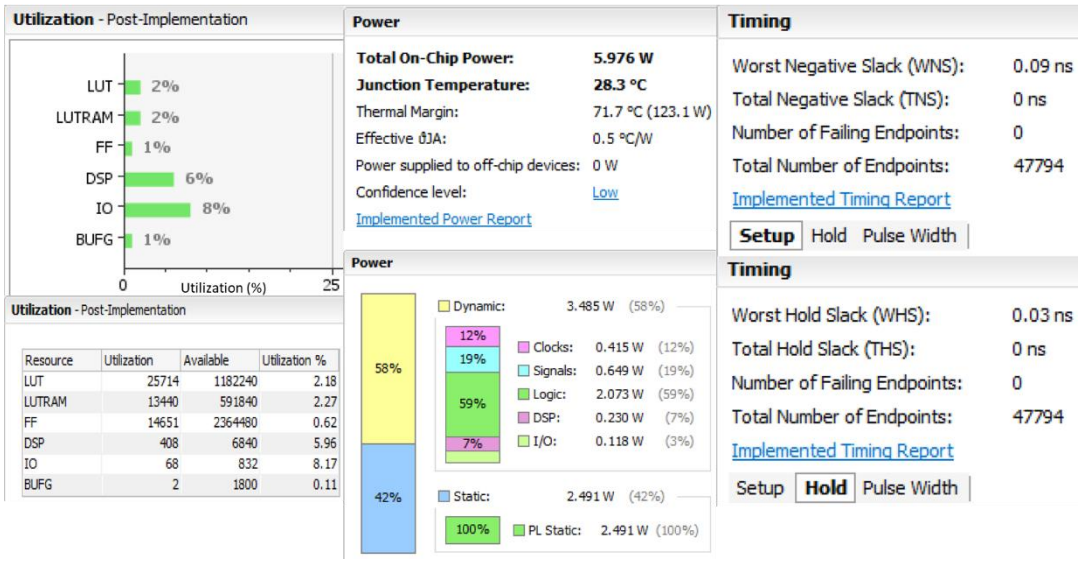

Figure 66. Vivado Flow_PerfOptimized_high Results for Range Compression Model.

Flow_PerfThresholdCarry

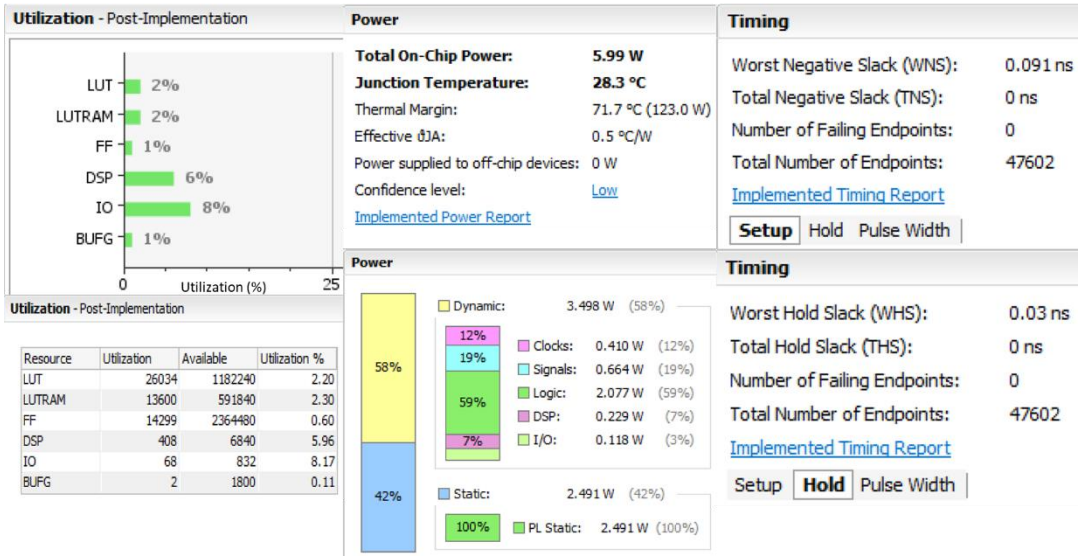

Figure 67. Vivado Flow_PerfThresholdCarry Results for Range Compression Model.

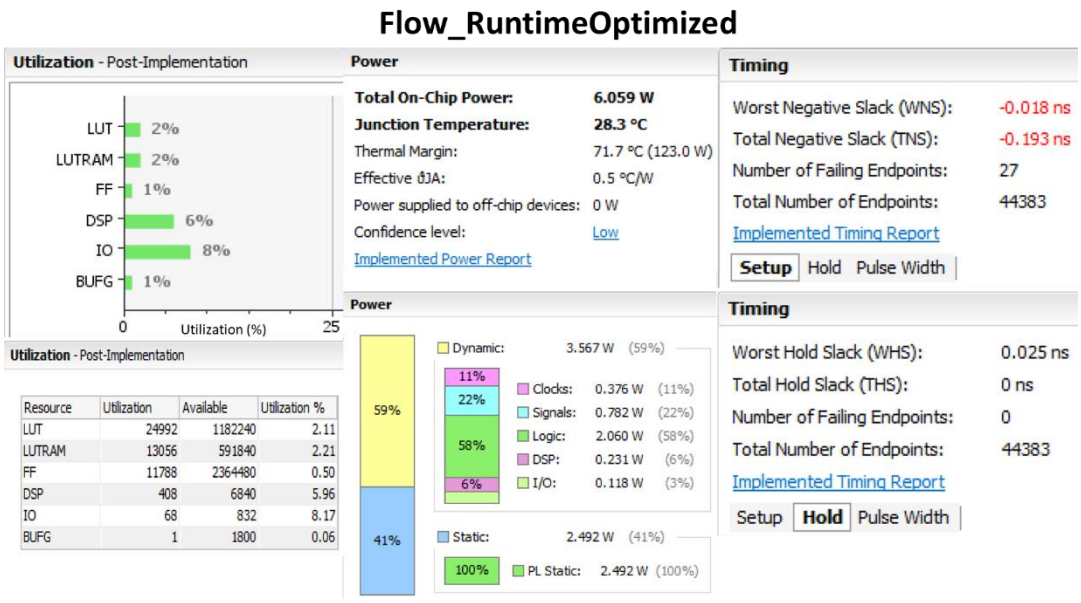

Figure 68. Vivado Flow_RuntimeOptimized Results for Range Compression Model.

C. DATA STORAGE MODEL DATA

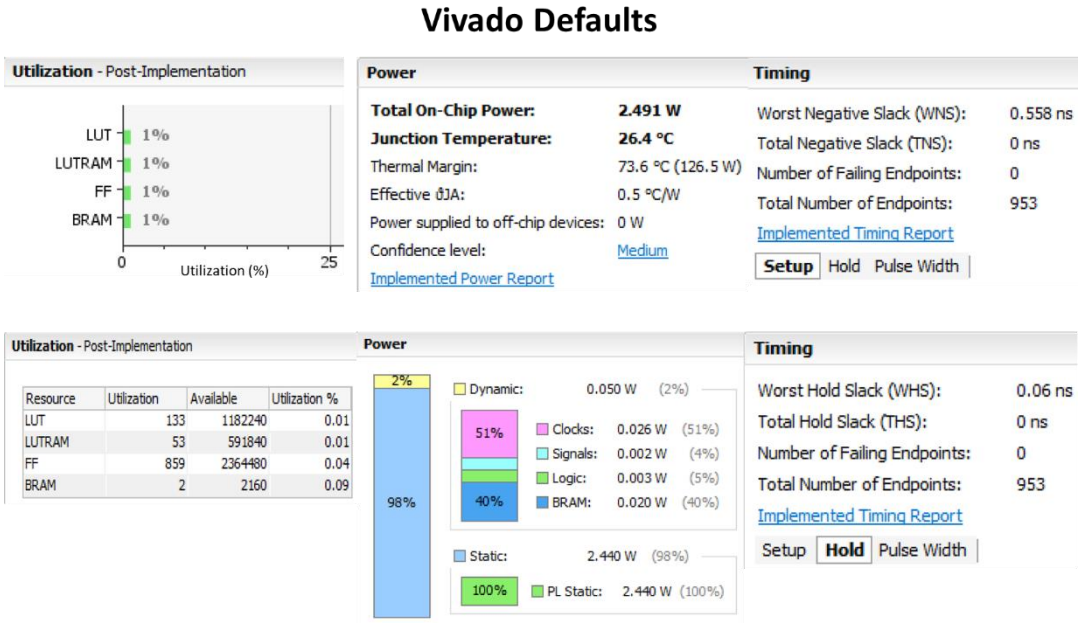

Figure 69. Vivado Default Results for Data Storage Model.

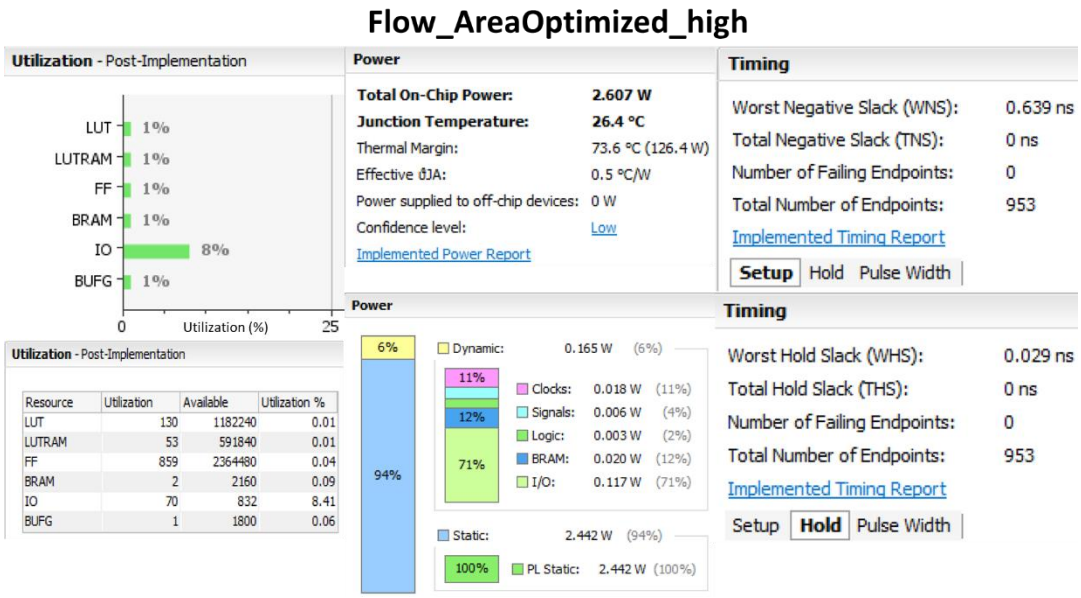

Figure 70. Vivado Flow_AreaOptimized_high Results for Data Storage Model.

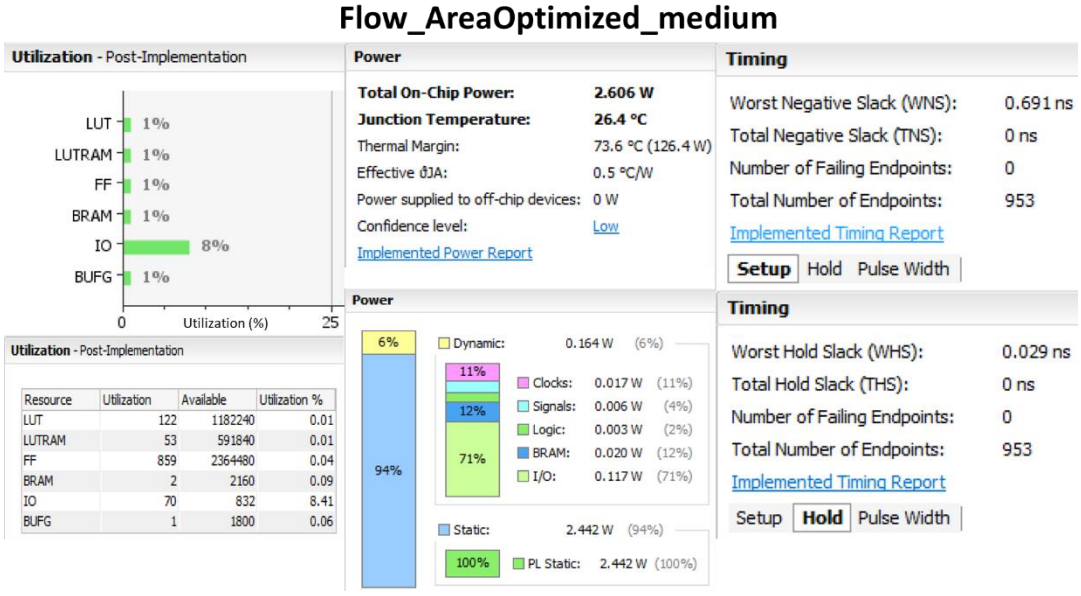

Figure 71. Vivado Flow_AreaOptimized_medium Results for Data Storage Model.

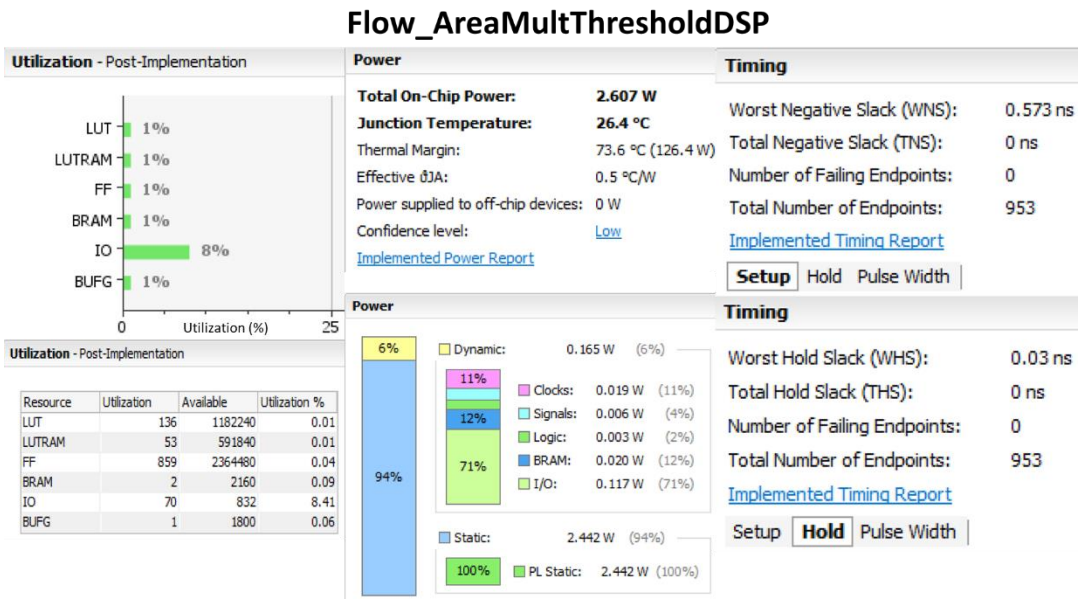

Figure 72. Vivado Flow_AreaMultThresholdDSP Results for Data Storage Model.

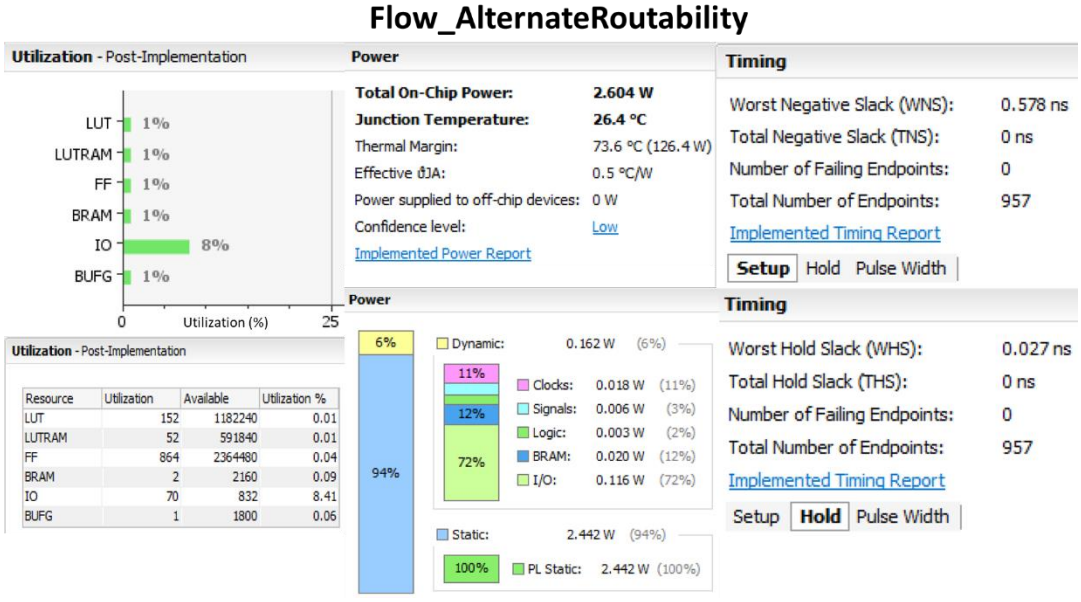

Figure 73. Vivado Flow_AlternateRoutability Results for Data Storage Model.

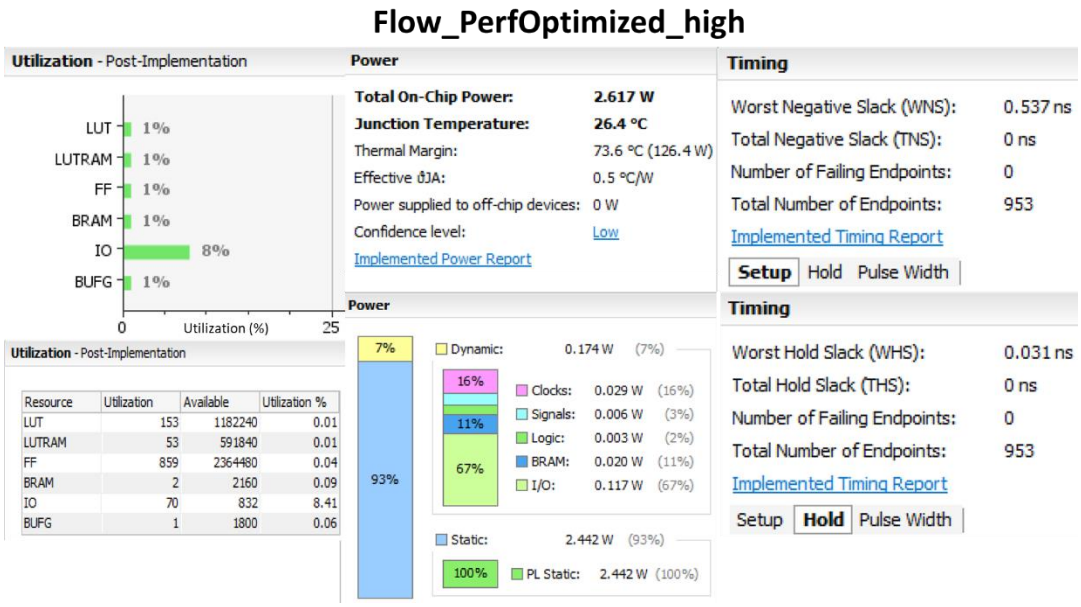

Figure 74. Vivado Flow_PerfOptimized_high Results for Data Storage Model.

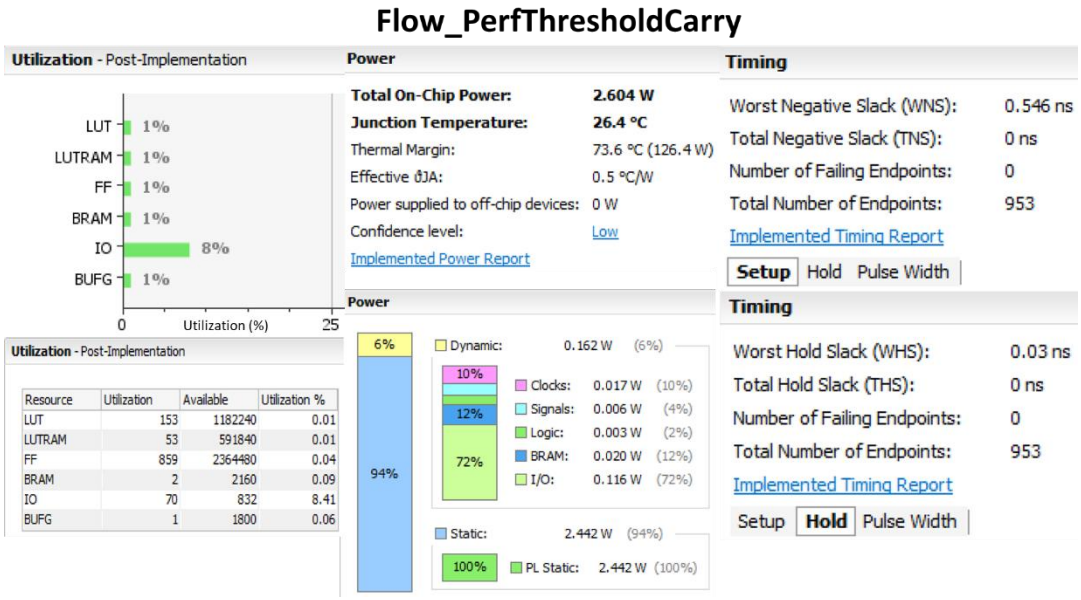

Figure 75. Vivado Flow_PerfThresholdCarry Results for Data Storage Model.

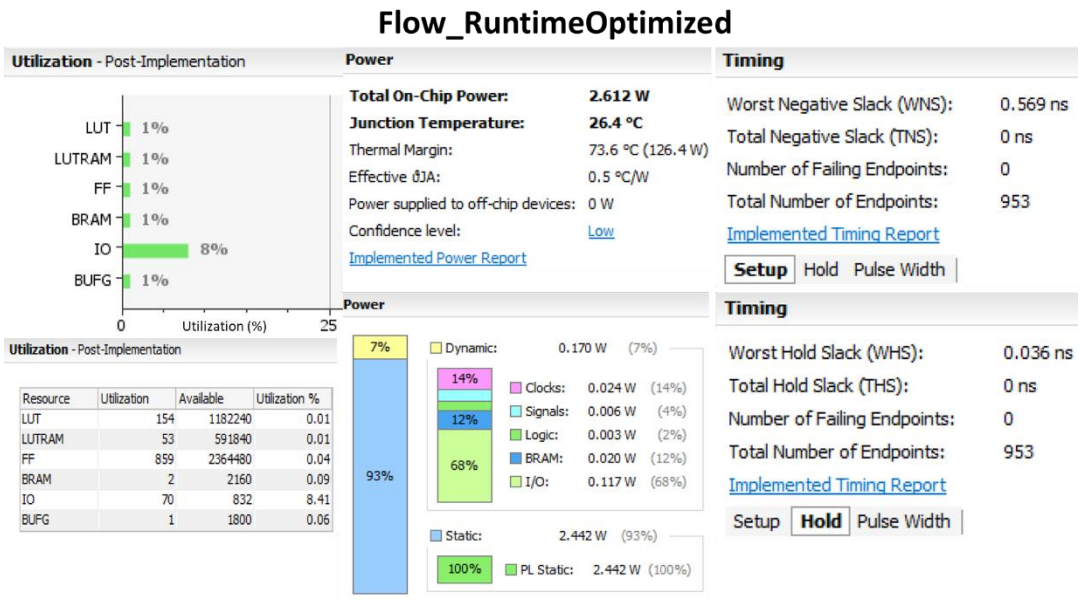

Figure 76. Vivado Flow_RuntimeOptimized Results for Data Storage Model.

D. DOPPLER FILTERING MODEL DATA

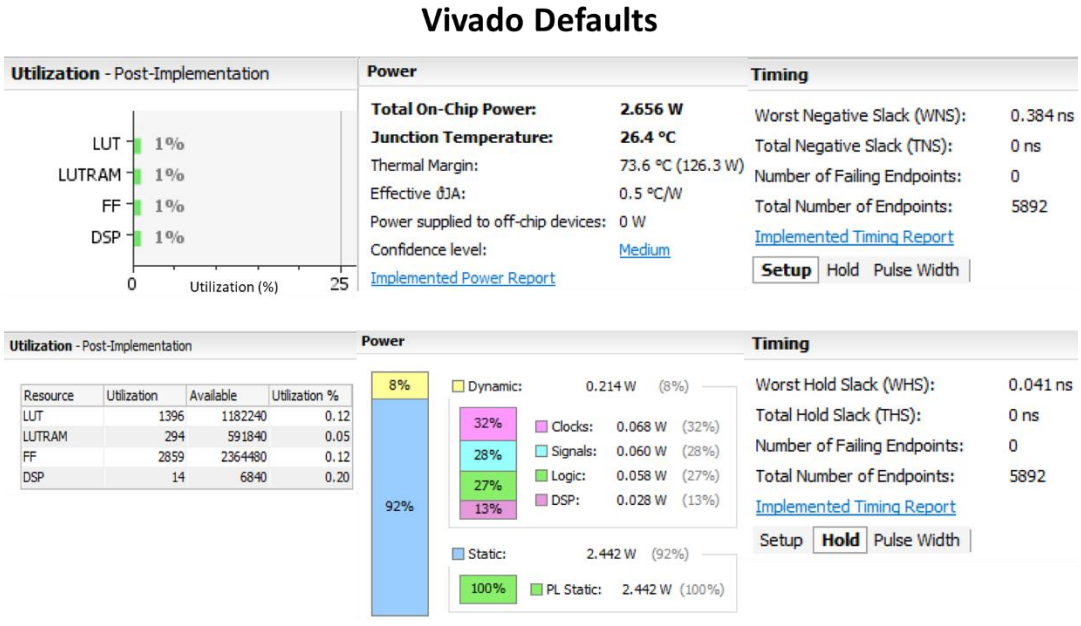

Figure 77. Vivado Default Results for Doppler Filtering Model.

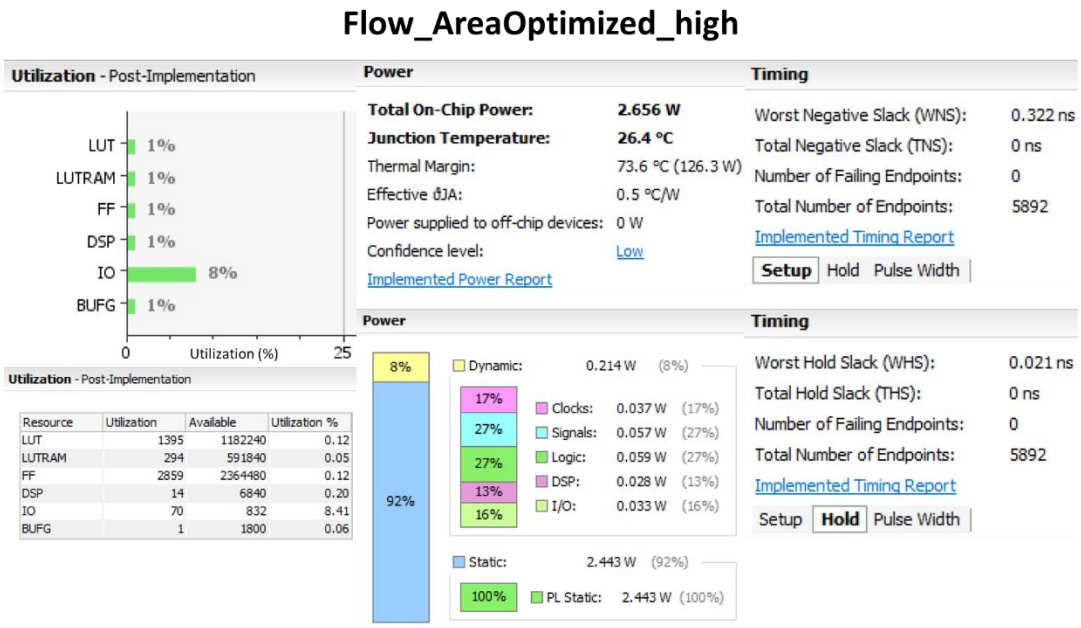

Figure 78. Vivado Flow_AreaOptimized_high Results for Doppler Filtering Model.

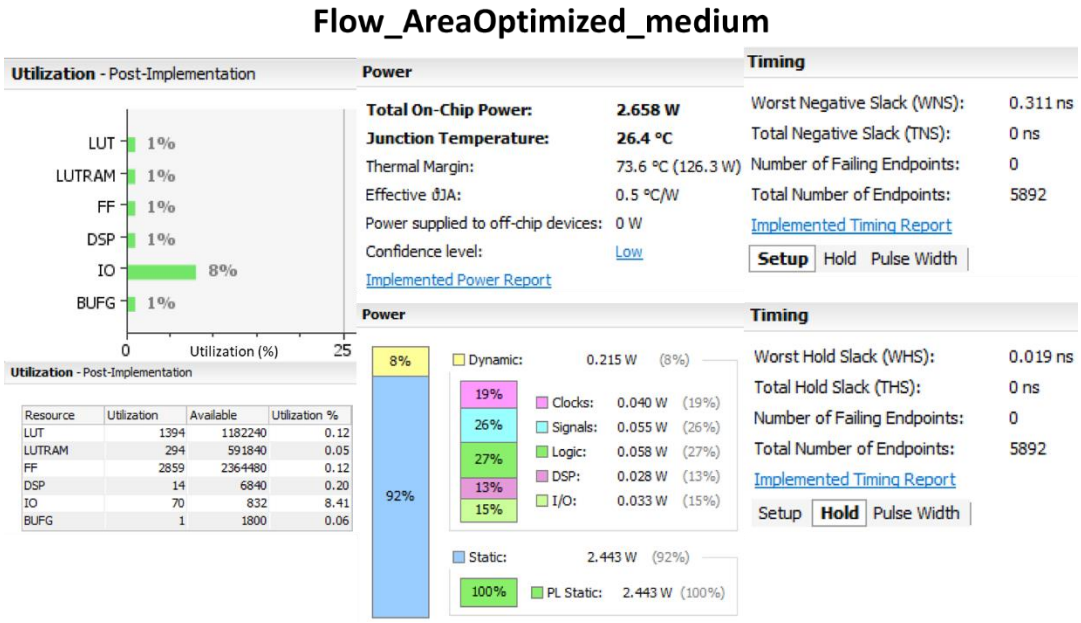

Figure 79. Vivado Flow_AreaOptimized_medium Results for Doppler Filtering Model.

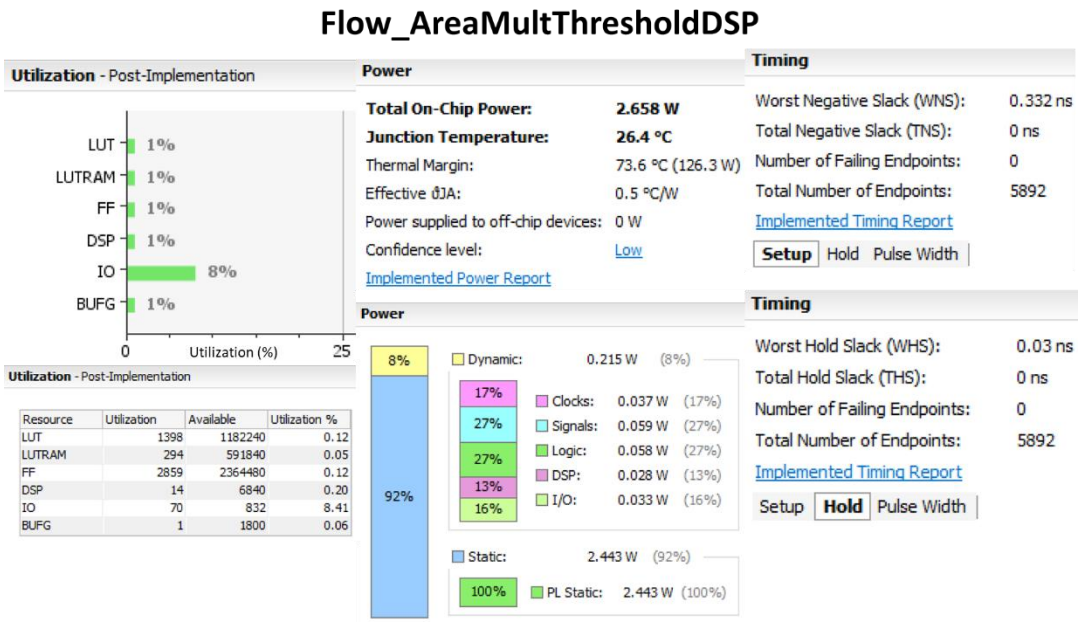

Figure 80. Vivado Flow_AreaMultThresholdDSP Results for Doppler Filtering Model.

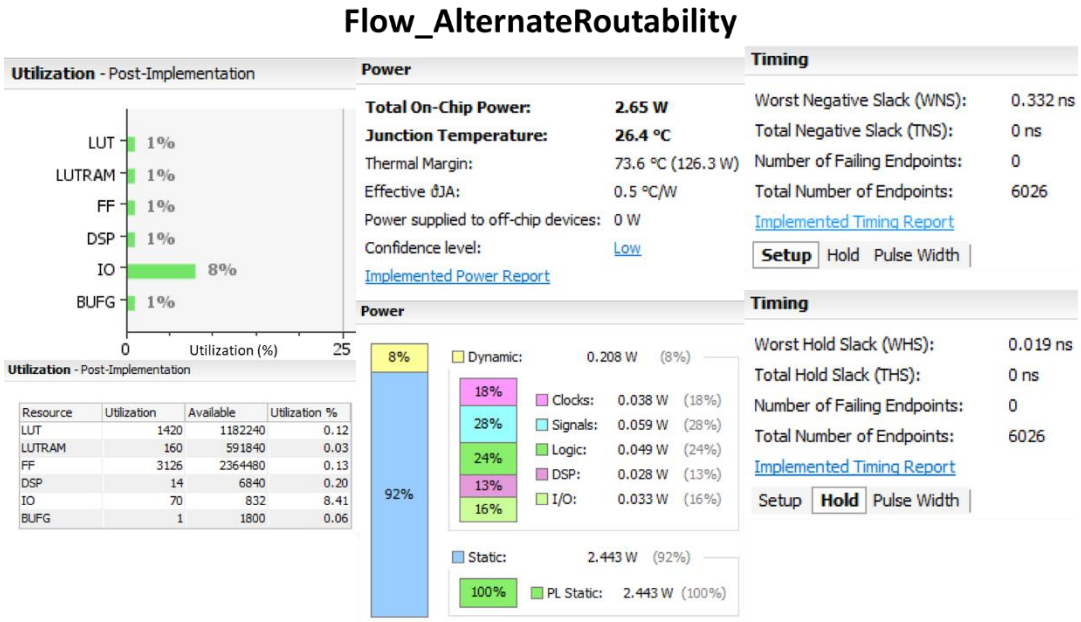

Figure 81. Vivado Flow_AlternateRoutability Results for Doppler Filtering Model.

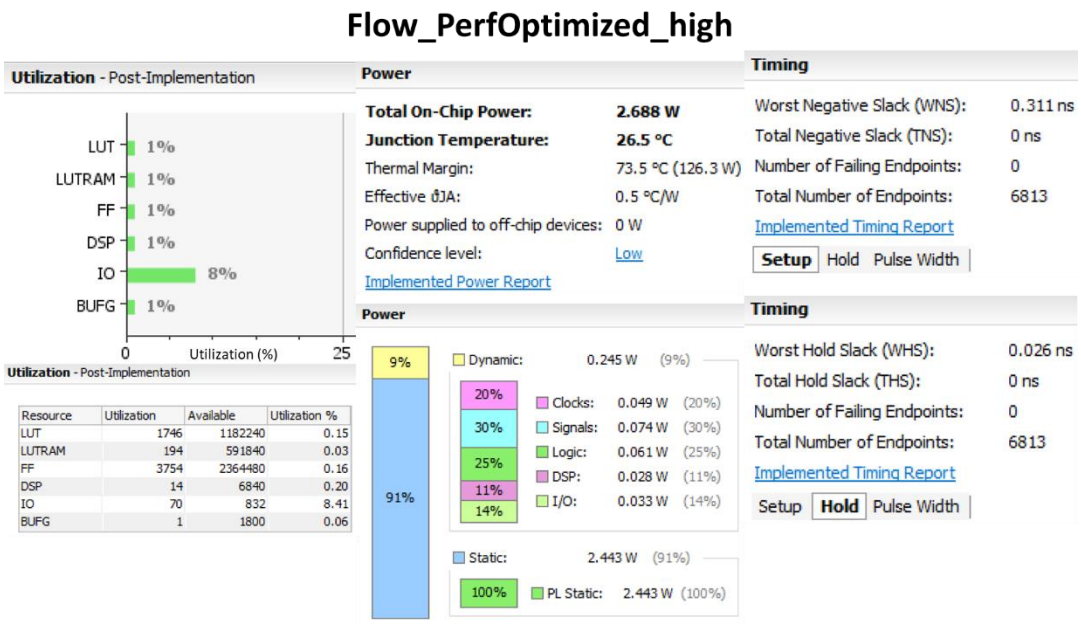

Figure 82. Vivado Flow_PerfOptimized_high Results for Doppler Filtering Model.

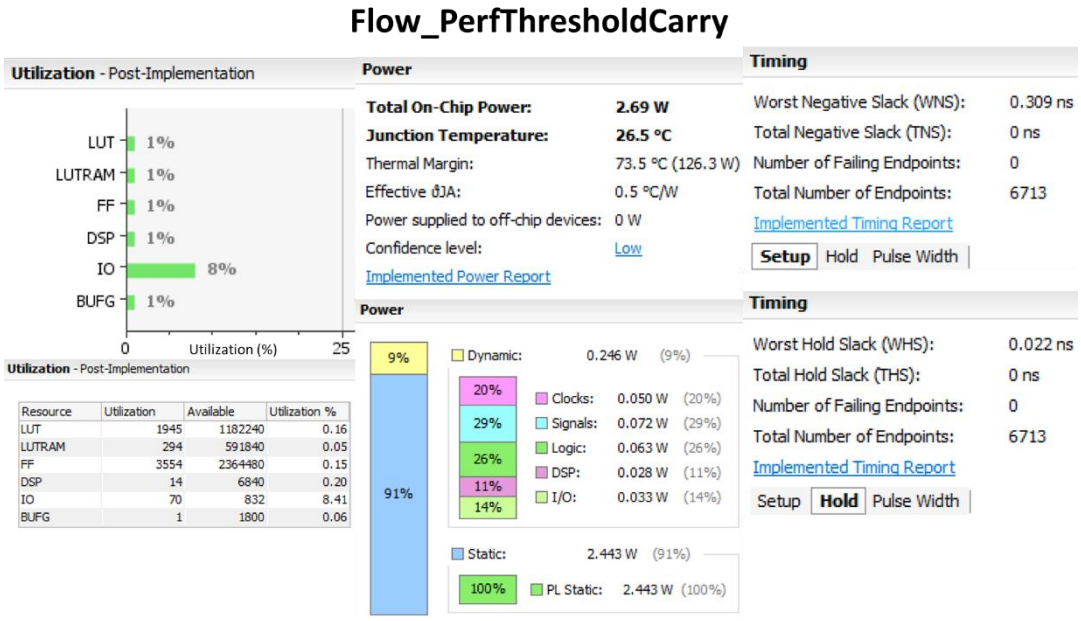

Figure 83. Vivado Flow_PerfThresholdCarry Results for Doppler Filtering Model.

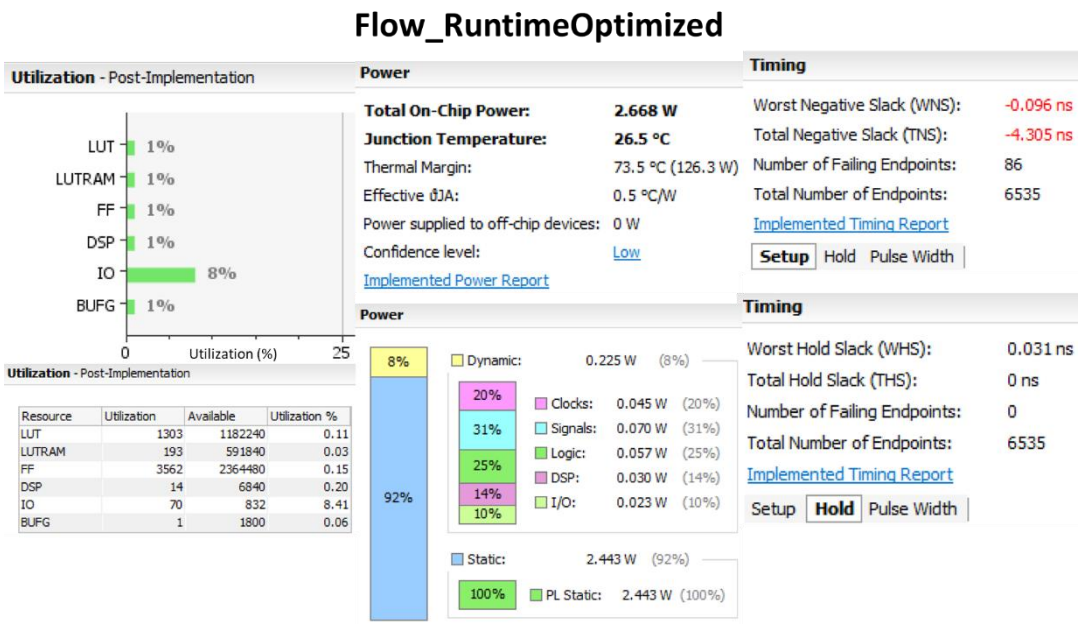

Figure 84. Vivado Flow_RuntimeOptimized Results for Doppler Filtering Model.

E. COHERENT INTEGRATION MODEL DATA

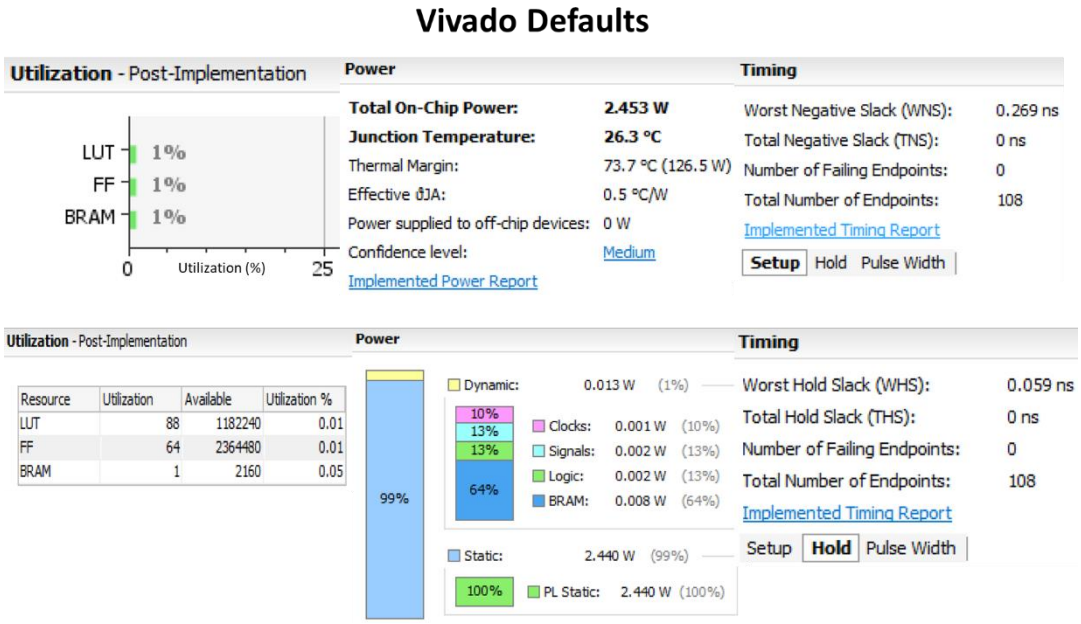

Figure 85. Vivado Default Results for Coherent Integration Model.

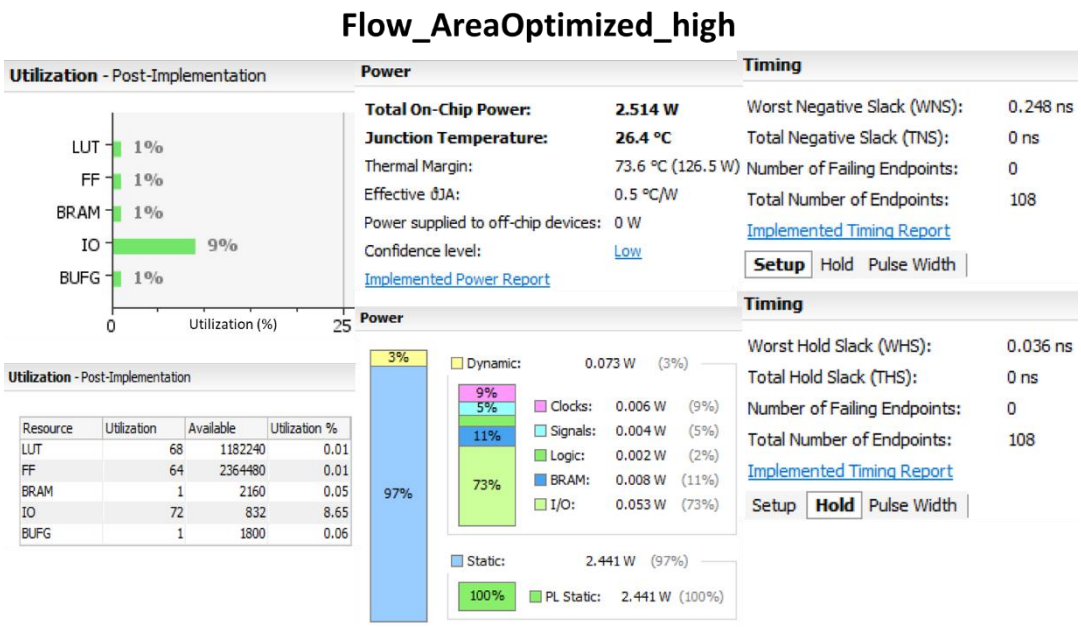

Figure 86. Vivado Flow_AreaOptimized_high Results for Coherent Integration Model.

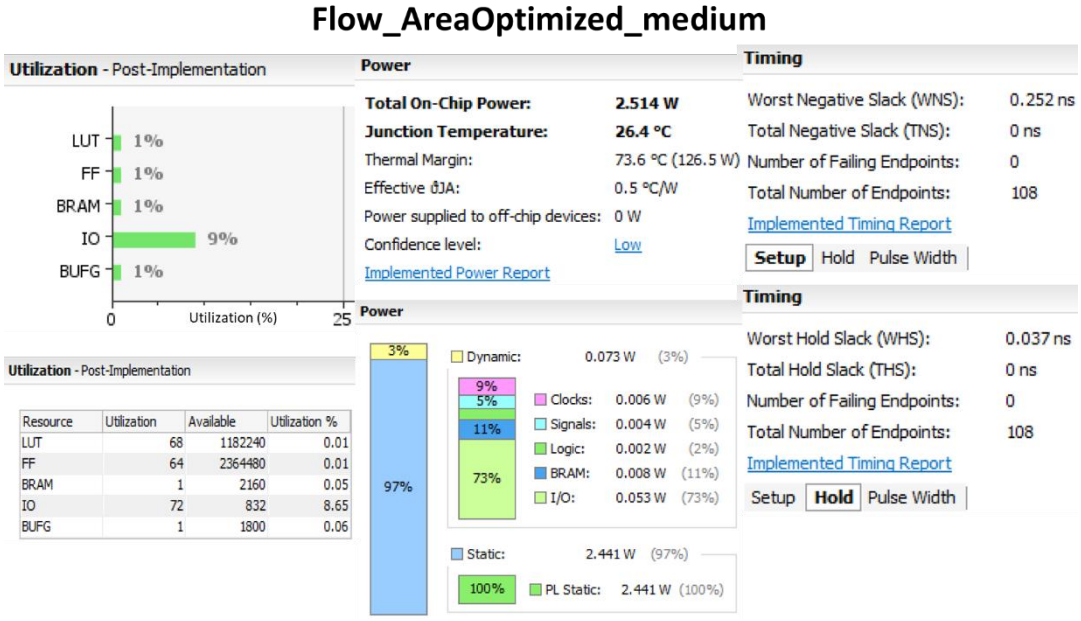

Figure 87. Vivado Flow_AreaOptimized_medium Results for Coherent Integration Model.

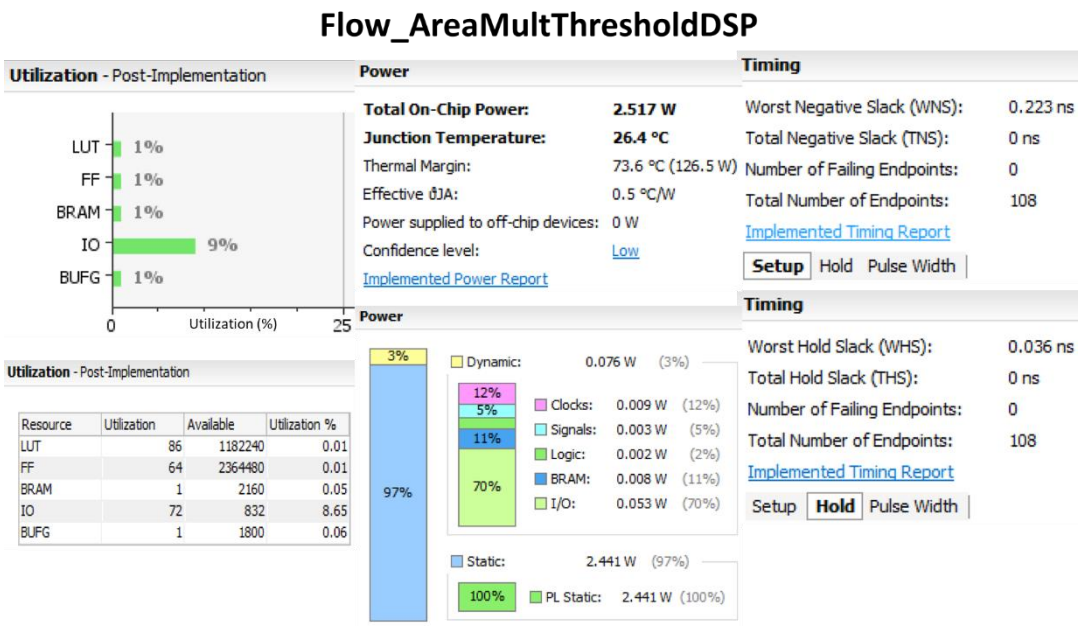

Figure 88. Vivado Flow_AreaMultThresholdDSP Results for Coherent Integration Model.

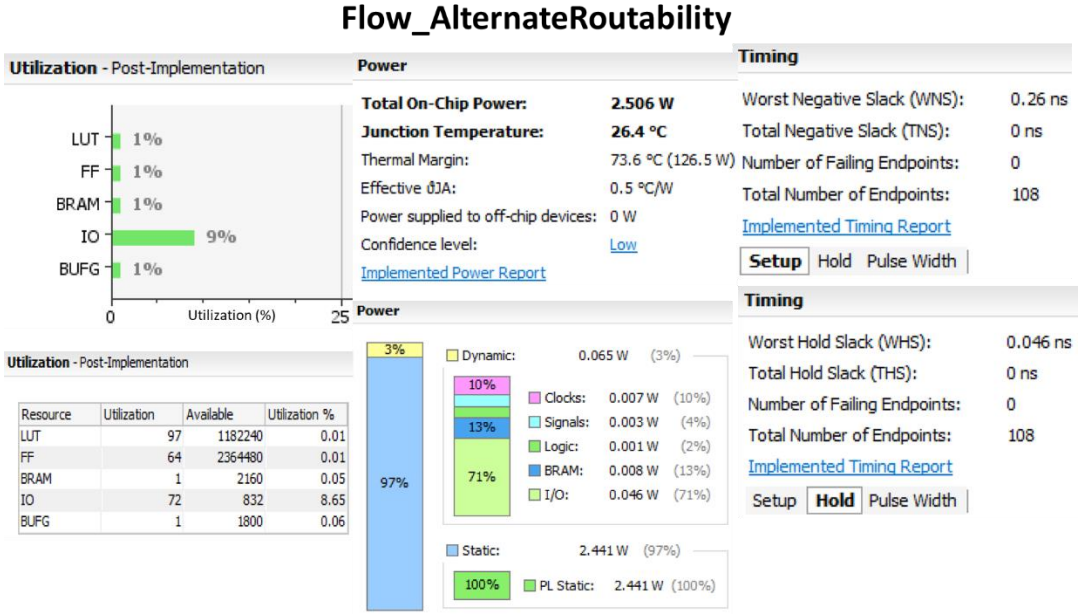

Figure 89. Vivado Flow_AlternateRoutability Results for Coherent Integration Model.

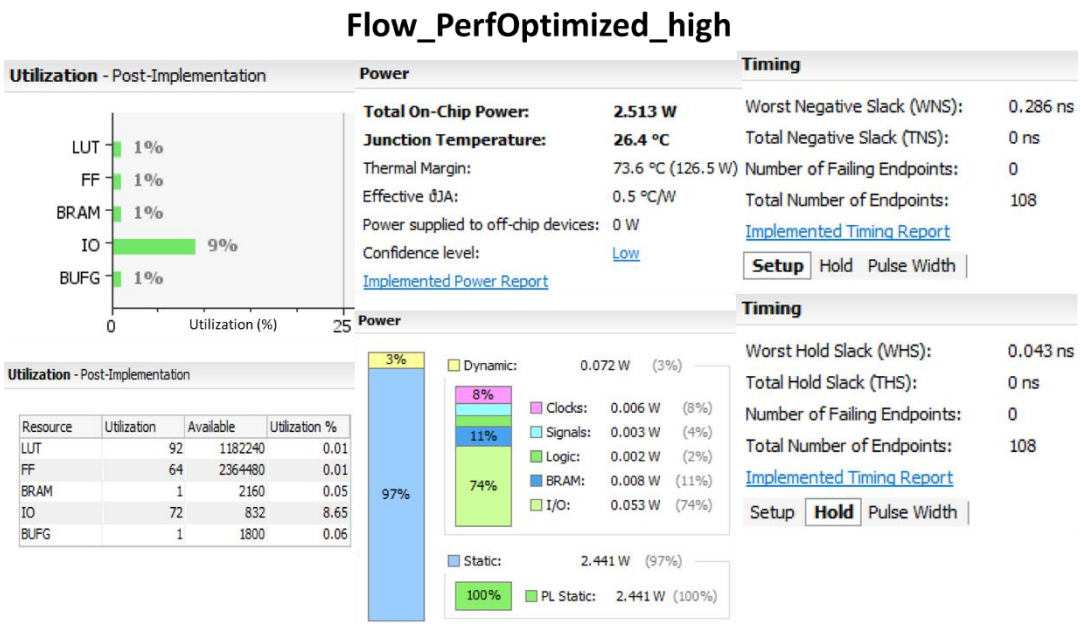

Figure 90. Vivado Flow_PerfOptimized_high Results for Coherent Integration Model.

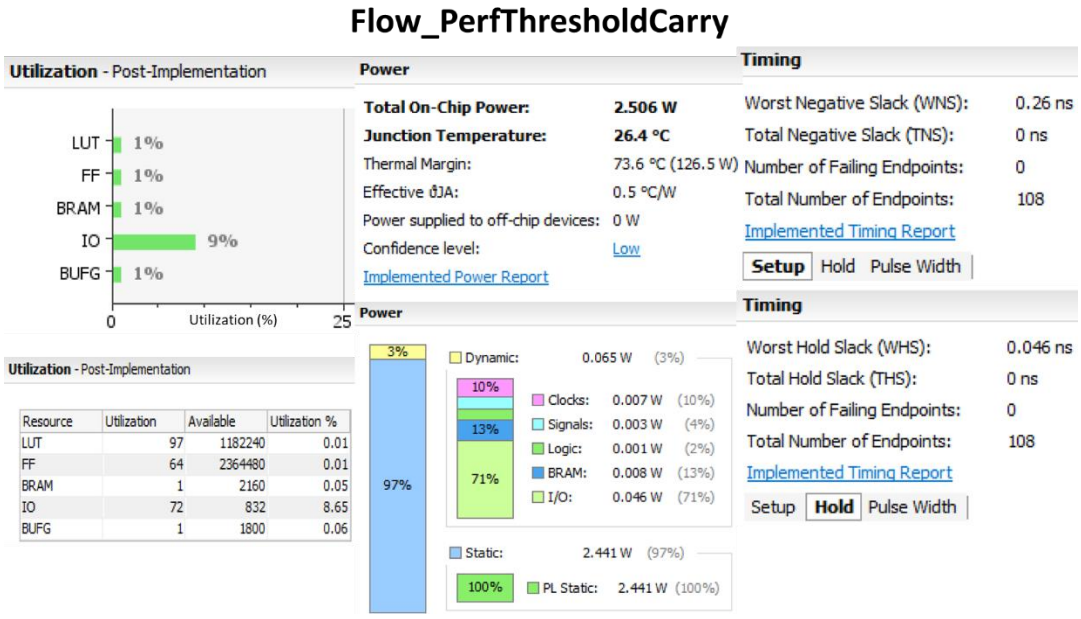

Figure 91. Vivado Flow_PerfThresholdCarry Results for Coherent Integration Model.

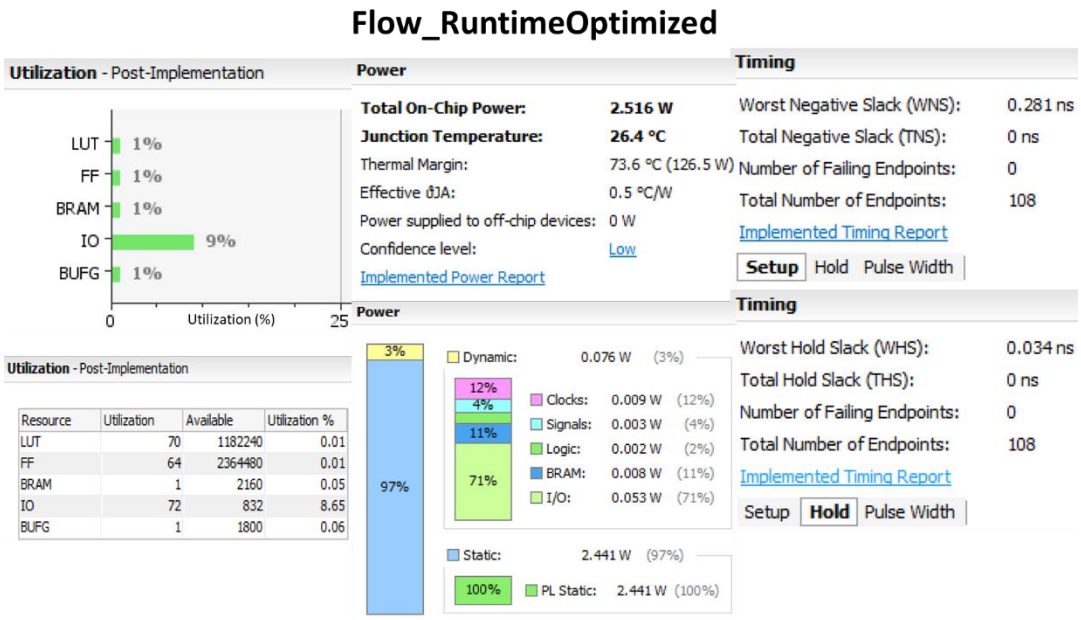

Figure 92. Vivado Flow_RuntimeOptimized Results for Coherent Integration Model.

F. THREE-STAGE COMPRESSION MODEL DATA

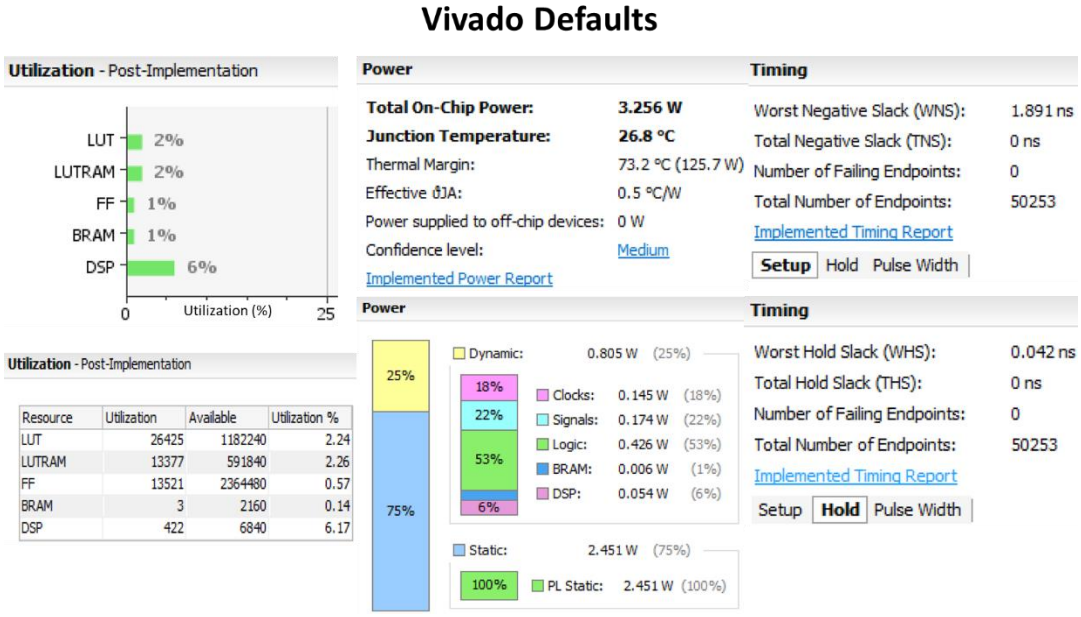

Figure 93. Vivado Default Results for Three-stage Compression Model.

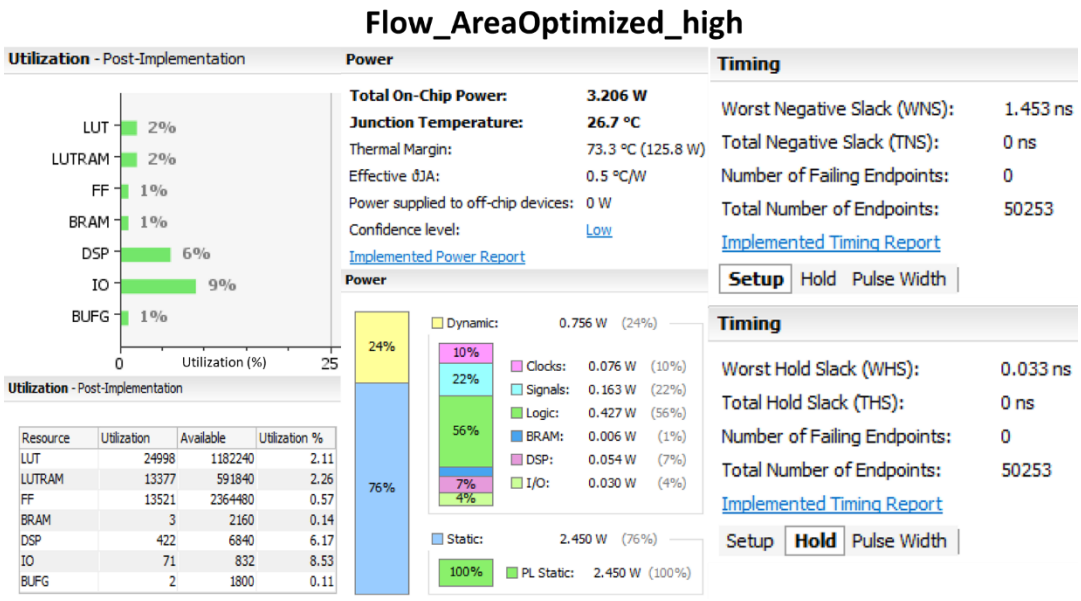

Figure 94. Vivado Flow_AreaOptimized_high Results for Three-stage Compression Model.

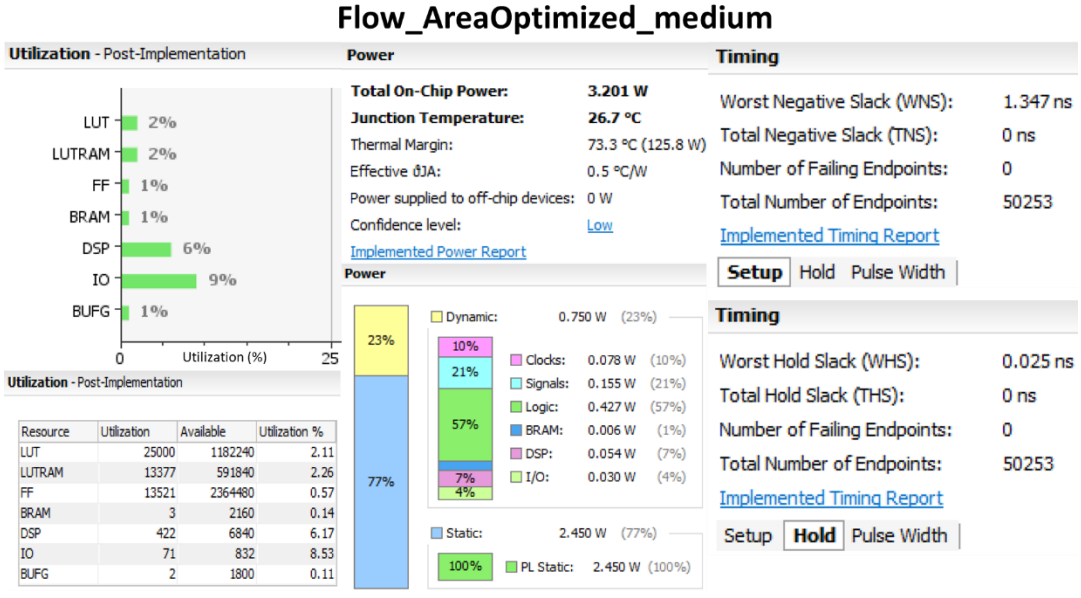

Figure 95. Vivado Flow_AreaOptimized_medium Results for Three-stage Compression Model.

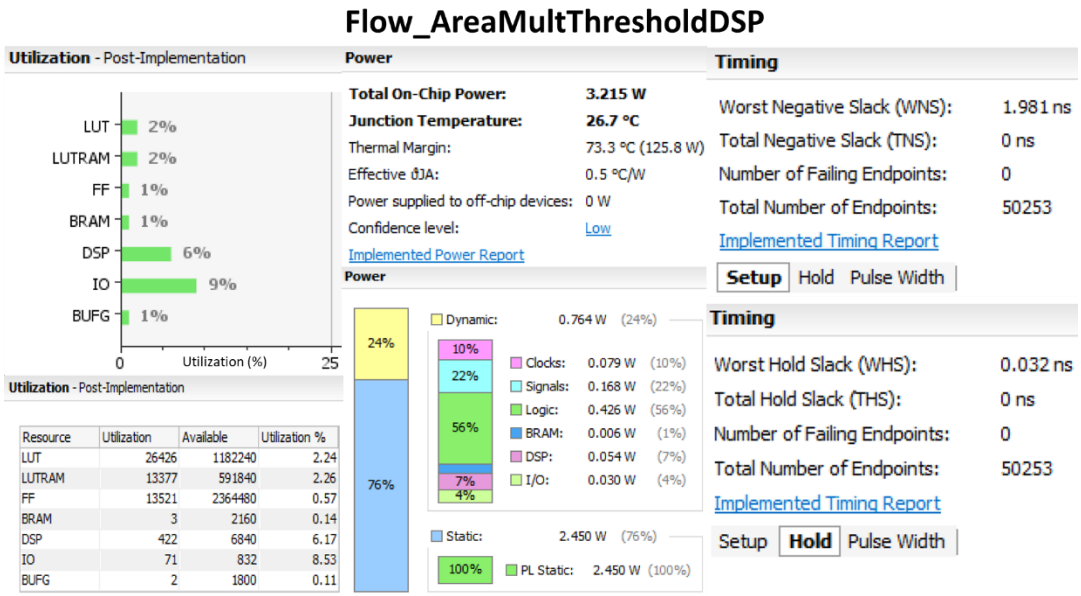

Figure 96. Vivado Flow_AreaMultThresholdDSP Results for Three-stage Compression Model.

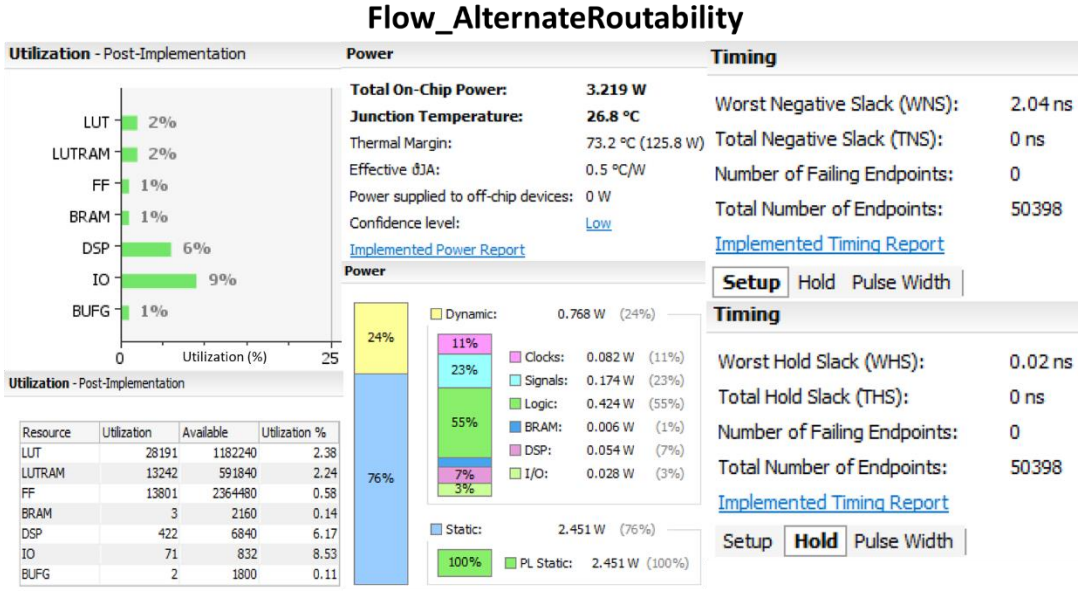

Figure 97. Vivado Flow_AlternateRoutability Results for Three-stage Compression Model.

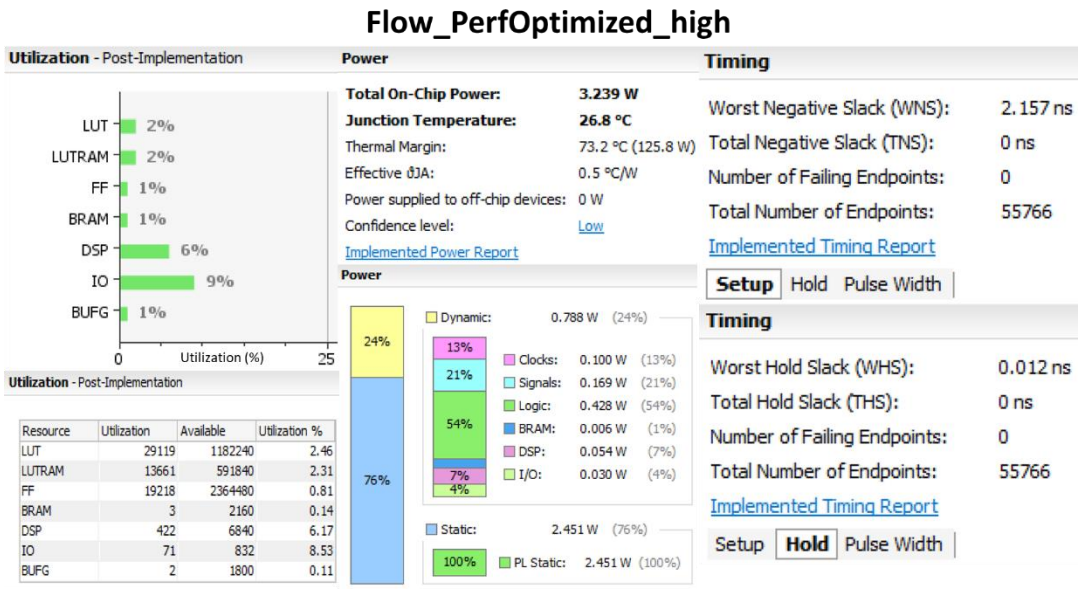

Figure 98. Vivado Flow_PerfOptimized_high Results for Three-stage Compression Model.

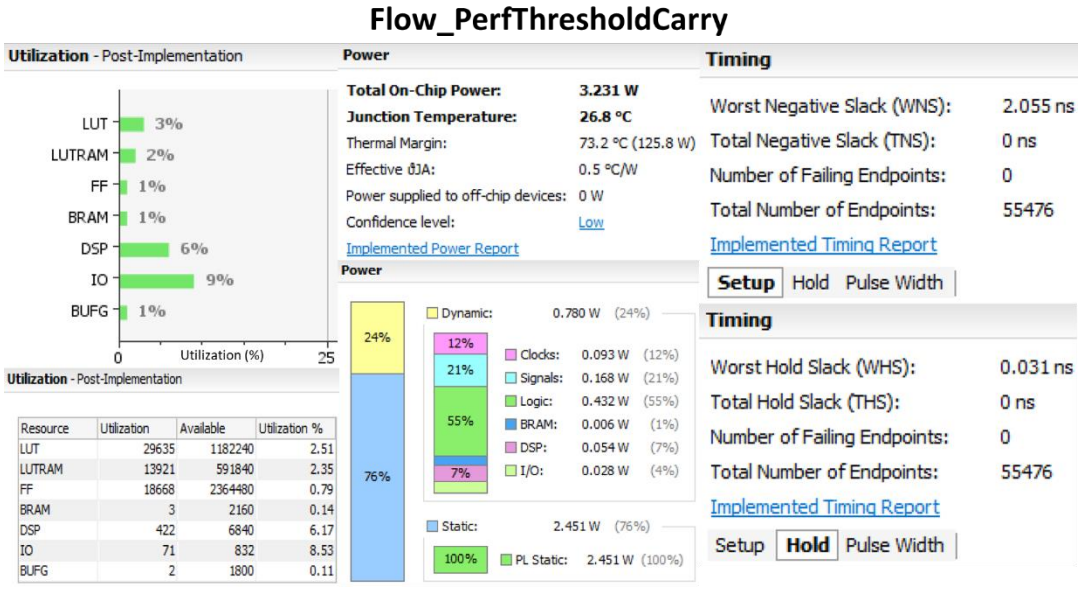

Figure 99. Vivado Flow_PerfThresholdCarry Results for Three-stage Compression Model.

Flow_RuntimeOptimized

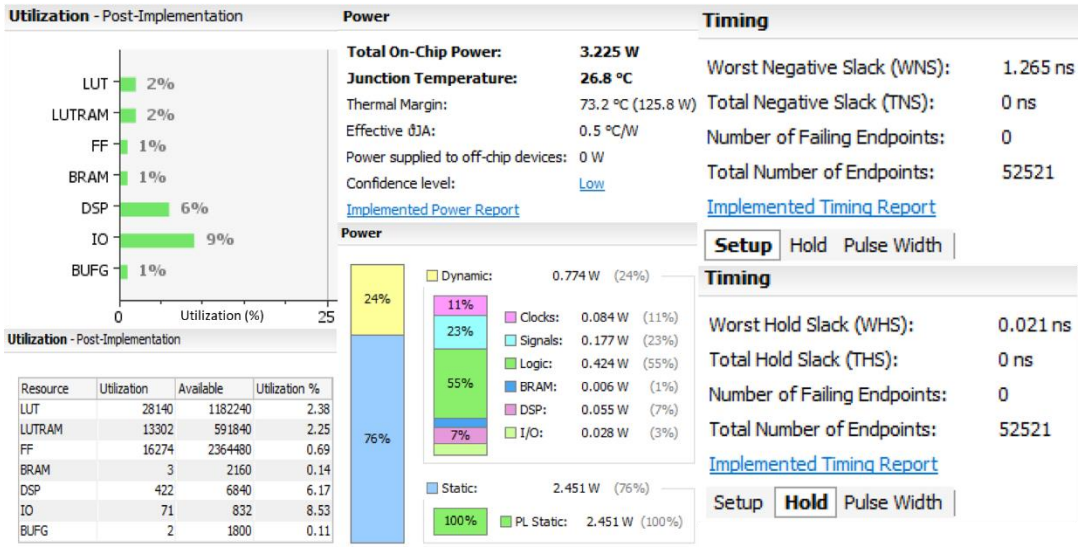

Figure 100. Vivado Flow_RuntimeOptimized Results for Three-stage Compression Model.

LIST OF REFERENCES

[1] P. E. Pace, S. Teich, O. E. Brooks, D. C. Jenn, and R. A. Romero, "Extended detection range using a polyphase CW modulation with an efficient number theoretic correlation process," in 2017 IEEE Radar Conference (RadarConf), May 2017, pp. 1669–1674.

[2] P. E. Pace, D. C. Jenn, and R. A. Romero, "Cognitive, multi-function periscope sensor with LPI radar, comms, and electronic warfare," presented at Office of Naval Research Symposium, Naval Postgraduate School, Monterey, CA, Jan. 18, 2017.

[3] P. E. Pace, *Detecting and Classifying Low Probability of Intercept Radar, 2nd ed*. Norwood, MA, USA: Artech House, 2009.

[4] N. Paepolshiri, "Extending the unambiguous range of CW polyphase radar systems using number theoretic transforms," Master's thesis. Naval Postgraduate School, Monterey, CA, 2011.

[5] P. E. Pace, *Advanced Techniques for Digital Receivers*. Norwood, MA, USA: Artech House, 2000.

[6] P. E. Pace, D. J. Fouts, and D. P. Zulaica, "Digital image synthesizers: Are enemy sensors really seeing what's there?" *IEEE Aerospace and Electronic Systems Magazine*, vol. 21, iss. 2, pp. 3-7, 2006.

[7] P. S. Ang, "DRFM CORDIC processor and sea clutter modeling for enhancing structured false target synthesis," Master's Thesis. Department of Electrical and Computer Engineering. Naval Postgraduate School, Monterey, CA, 2017.

[8] MathWorks. (n.d.). "DSP for FPGAs," in MathWorks Training Course Notebook, November 2017, pp. 2-1 – 4-16, 9-5 – 9-41, 12-5 – 12-56

[9] I. Kuon and J. Rose. "Measuring the gap between FPGAs and ASICs," Department of Electrical and Computer Engineering. University of Toronto, Toronto, ON, 2006. [Online]. Available: https://ece.gmu.edu/coursewebpages/ECE/ECE448/S09/viewgraphs/Gap_between_FPGAs_and_ASICs.pdf

[10] "History of FPGAs," Internet Archive Way Back Machine, April 12, 2017. [Online]. Available: https://web.archive.org/web/20070412183416/http://filebox.vt.edu/users/tmagin/history.htm

[11] D. W. Page and L. R. Peterson. “Re-programmable PLA,” US4508977A, 1983. Accessed May 19, 2018. [Online]. Available: https://patents.google.com/patent/US4508977?oq=4508977

[12] D. W. Page. “Dynamic data re-programmable PLA,” US4524430A, 1983. Accessed May 19, 2018. [Online]. Available: https://patents.google.com/patent/US4524430?oq=4524430

[13] R. Wilson, “In the Beginning,” Altera, 2009. [Online]. Available: https://www.altera.com/solutions/technology/system-design/articles/_2013/in-the-beginning.html

[14] “Xcell,” Xilinx, 1999. [Online]. Available: https://www.xilinx.com/publications/archives/xcell/Xcell32.pdf

[15] “Xilinx, Inc. History,” Xilinx. Accessed May 19, 2018. [Online]. Available: http://www.fundinguniverse.com/company-histories/xilinx-inc-history/

[16] C. Maxfield. “Xilinx co-founder Ross Freeman honored,” EE Times, February 12, 2009. [Online]. Available: https://www.eetimes.com/document.asp?doc_id=1243111

[17] R. H. Freeman. “Configurable electrical circuit having configurable logic elements and configurable interconnects,” US4870302A, 1989. Accessed May 19, 2018. [Online]. Available: https://worldwide.espacenet.com/publicationDetails/biblio?CC=US&NR=4870302&KC=&FT=E&locale=en_EP#

[18] “Top FPGA companies for 2013,” Source Tech 411, April 28, 2013. [Online]. Available: http://sourcetech411.com/2013/04/top-fpga-companies-for-2013/

[19] “Zynq-7000 SoC,” Xilinx. Accessed May 19, 2018. [Online]. Available: https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html

[20] R. McMillan. “Microsoft supercharges Bing search with programmable chips,” Wired, June 16, 2014. [Online]. Available: https://www.wired.com/2014/06/microsoft-fpga

[21] “Xilinx Virtex UltraScale+ FPGA VCU118 Evaluation Kit,” Xilinx. Accessed May 17, 2018. [Online]. Available: https://www.xilinx.com/products/boards-and-kits/vcu118.html#overview

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California