NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

SYMMETRICAL RESIDUE-TO-BINARY CONVERSION ALGORITHM, PIPELINED FPGA IMPLEMENTATION, AND TESTING LOGIC FOR USE IN HIGH-SPEED FOLDING DIGITIZERS

by

Ross Alan Monta

December 2005

Thesis Advisor:	Phillip E. Pace
Thesis Co Advisor:	Douglas Fouts

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** December 2005	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: Symmetrical Residue-to-Binary Conversion Algorithm, Pipelined FPGA Implementation, and Testing Logic for Use in High-Speed Folding Digitizers.			**5. FUNDING NUMBERS**
6. AUTHOR(S) Ross Alan Monta			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) Office of Naval Research (Code 313) Arlington, VA 22203-1995			**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**

13. ABSTRACT (maximum 200 words)

The robust symmetrical number system (RSNS) can play a significant role in the reduction of encoding errors within a low-power folding analog-to-digital converter (ADC). A key part of this ADC design is the logic block that converts the symmetrical residues from each channel into a more convenient binary output. This thesis describes a robust symmetrical residue-to-binary conversion algorithm for moduli $m_1 = 7$, $m_2 = 8$ and $m_3 = 9$ (ADC dynamic range $\hat{M} = 126$). Also described is a pipelined digital logic implementation for use in high speed programmable logic or application specific integrated circuits. To verify correct outputs of the robust symmetrical residue-to-binary conversion algorithm, a digital test circuit is described that generates the thermometer code (symmetrical residues) for the 3-channel ADC design.

14. SUBJECT TERMS ADC, Robust Symmetrical Numbering System (RSNS), Xilinx,			**15. NUMBER OF PAGES** 85
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

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.

SYMMETRICAL RESIDUE-TO-BINARY CONVERSION ALGORITHM, PIPELINED FPGA IMPLEMENTATION, AND TESTING LOGIC FOR USE IN HIGH-SPEED FOLDING DIGITIZERS

Ross A. Monta
Major, United States Marine Corps
B.S., Carnegie Mellon University, 1995

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 2005

Author: Ross Alan Monta

Approved by: Phillip E. Pace
Thesis Advisor

Douglas Fouts
Co-Advisor

Jeffrey B. Knorr
Chairman, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The robust symmetrical number system (RSNS) can play a significant role in the reduction of encoding errors within a low-power folding analog-to-digital converter (ADC). A key part of this ADC design is the logic block that converts the symmetrical residues from each channel into a more convenient binary output. This thesis describes a robust symmetrical residue-to-binary conversion algorithm for moduli $m_1 = 7$, $m_2 = 8$ and $m_3 = 9$ (ADC dynamic range $\widehat{M} = 126$). Also described is a pipelined digital logic implementation for use in high speed programmable logic or application specific integrated circuits. To verify correct outputs of the robust symmetrical residue-to-binary conversion algorithm, a digital test circuit is described that generates the thermometer code (symmetrical residues) for the 3-channel ADC design.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

During the writing of this Thesis, the easiest tasks became difficult and time slipped away faster than was expected. This fact made me appreciate the little things in life that much more. I owe my dear wife, Dru, and my five children, Blake, Jordan, Chandler, Dominic, and Rylie, my most sincere appreciation for putting up with the long hours and the stress.

I would also like to thank my Advisors for their tutelage, and support during this most enlightening endeavor. Their assistance in both technical and professional discussion of this subject matter, made it both educational and enjoyable.

This work was supported by the Office of Naval Research. (ONR code 313).

In closing, I would like to thank my Family, both spouse and children, along with my parents and siblings for the support they have shown me throughout my life. With them and the support and blessings of my God, I have and will continue to perform my military and family duties to the best of my ability.

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

The goal of this thesis is to expand on already existing theory in order to create a high resolution, high speed, low power, Analog to Digital Converter (ADC). Once implemented, this ADC will have many uses to include incorporation into miniaturized sensor networks, system-on-a-chip (SOC), and many other areas where size and power consumption are limiting factors.

The most recent work in this area was done by Brian Luke in a Dissertation for his Doctorate at the Naval Postgraduate School [1]. This work demonstrated the theory behind a 3 channel folding ADC that utilized the RSNS conversion in order to reduce the power consumption and size of a system. The focus in his study was for SOC applications. The power and size savings of the design were significant; however it did not achieve a high enough resolution.

Utilizing the theory developed in the above dissertation, a project for a higher resolution folding ADC was initiated. In this thesis, the focus was to expand on the folding ADC's RSNS digital processing to achieve higher resolution from the overall circuit being developed.

The contribution of the research contained in this thesis was two fold. First, to verify the theory contained in [1] for a three channel RSNS folding ADC could be expanded to achieve higher resolution. Second, to design and test a circuit using the expanded, higher resolution equations produced and to verify the actual results match the expected theoretical results.

In order to accomplish the expansion verification, the first decision to be made was to establish the moduli for the three channels of the system. Initially, 5, 6 and 7 were considered. However, they did not sufficiently increase the resolution to the required value. Moduli 7, 8, and 9 for channels 1, 2, and 3, respectively, were chosen and found to accomplish an acceptable resolution for theoretical validation of the digital system.

This thesis demonstrated and verified the theoretical expansion of the folding circuit by implementing the equations in MATLAB. The results of the MATLAB

simulation demonstrated a useful dynamic range of 126, or 7-bits. The MATLAB also verified the proper test vectors and the operational range of the system.

Once the theory was verified, a circuit was designed using Xilinx Project Navigator and Mentor Graphics' Model Sim programs. The schematic capture functionality of Xilinx was utilized, and due to the extensive amount of wiring, took several months to complete.

In order to test the design captured in Xilinx, a Model Sim waveform had to be generated. Due to limitations of the waveform generator associated with Model Sim, additional logic to produce a proper signal to test the system had to be designed.

Utilizing the test vectors verified with MATLAB, a waveform was generated and processed through the Xilinx system. The result of the testing was that a dynamic range of 126 was achieved and theoretical and actual results matched.

The next step is to convert the Xilinx design over to application specific integrated circuit (ASIC) software for simulation with the folding analog portion of the ADC. Once completed, this system will have far reaching application to the DoD in Electronic Warfare, Sensor Systems, Unmanned Aerial Vehicle (UAV), and any other area where power and size constraints are relevant.

I. INTRODUCTION

A. FOLDING ANALOG TO DIGITAL CONVERTER BRIEF HISTORY

Analog to Digital Converters (ADC) are integral parts of almost all communication and detection systems available today. These devices allow a smooth flow into the digital processing capabilities of today's high speed digital computing from our analog surroundings to process information.

Folding circuits added a new dimension to the ADC implementation. The folding of the analog signal allows the repetitive use of comparators thereby reducing the die size and increasing the current that can be used for each comparator, increasing the analog band-width.[2] This significant attribute of lower power consumption and smaller die space has led to research in this area for use in data collection with unmanned aerial vehicles (UAVs), and System on a Chip applications that are of increasing interest to the DoD.

Currently, most folding ADCs are using 64 comparators for an 8-bit resolution [2], [9], [10]. There are some ADCs that use 26 comparators for the 8-bit resolution. However, the error correction logic associated with this pipelined implementation creates an exorbitant amount of logic overhead [7]. The limitations of these current systems led to this thesis research.

B. PRINCIPAL CONTRIBUTIONS

In [1], it was shown theoretically that a three channel folding ADC of fold moduli 3, 4, and 5 could be designed using the Robust Symmetrical Number System (RSNS). In this thesis, the equations developed in [1] were scrutinized and the general equations for a larger moduli three channel folding ADC were expanded. The moduli chosen for this thesis were 7, 8, and 9. The equations resulting from this expansion were verified utilizing software analysis.

Using the above moduli, a seven bit ADC can be designed using only 24 comparators vice 64, which is the average number for an eight bit ADC [2, 3]. This

demonstrates the savings made possible in using the RSNS folding ADC model for on chip designs for significantly more complicated applications in terms of energy and space savings due to comparator savings.

The major contribution of the research contained in this thesis is two fold. First, it shows verification of the general equations for a three channel RSNS folding ADC. Second, it shows the testing of circuits designed using these equations and that they produce the expected theoretical output results.

The first contribution was accomplished using MATLAB. The fist step was to expand the equations given in [1], then convert them into MATLAB syntax for modeling and simulation. The major portion of the work using MATLAB was creating the test vectors for the equations. Once the code was written and executed, the result was a seven bit output with a useful dynamic range of 126.

The second contribution was accomplished using Xilinx Project Navigator and Mentor Graphics' Model Sim programs. The first step was to convert the equations utilized for the MATLAB portion to NAND, NOR, INVERTER, and XOR gates for implementation in hardware. The rewritten equations were then implemented using the schematic capture functionality of Xilinx Project Navigator. As with the MATLAB code, the harder portion was creating the test signal for the system. Using the Mentor Graphic Model Sim program that has a symbiotic relationship with the Xilinx Project Navigator, a test-bench waveform was created that produced the expected results and that were exactly the same as the MATLAB results.

The thesis work was carried out by first accomplishing the MATLAB verification so that significant resources would not be brought to bear on this subject unless the equations and analysis were verified. Once the MATLAB verification was completed, the schematic capture portion was started. Due to the complexity and extensive amount of wiring, the implementation of the equations in logic took several months to complete.

Once the schematics of the RSNS-to-binary system were completed, a test-bench waveform needed to be created to test the system using Model Sim. Model Sim has a function generator for this purpose; however it did not possess the function needed to easily create the needed test-bench for this system. The available functions were

evaluated and a resetting counter function was chosen. Utilizing this function, additional logic was designed to convert a resetting counter input to the desired function needed for the RSNS-to-binary converter.

The new logic was front loaded to the RSNS-to-binary logic and the appropriate waveform was processed through the system. The resulting data demonstrated the circuit functioned properly.

C. THESIS OUTLINE

Chapter II is a brief summary of the RSNS and is an expansion of the equations used for the main body of work in this thesis. This chapter will be used to reassert the model in [1] for the 3, 4, and 5 moduli ADC. Along with the 3, 4, and 5 moduli, the general equations for each major component will be given and expanded to address moduli 7, 8, and 9. This chapter will delineate all the equations used for the following chapters for design of the RSNS-to-Binary conversion circuit. All equations in this chapter and all following chapters will be Boolean expression using a binary numbering system unless otherwise specified. Table 1 is a list of the most commonly used variables in this thesis. The bit order will use the highest number as the most significant bit (MSB) and zero as the least significant bits (LSB) as denoted by the bit order subscript.

Variable	Description	Subscript i	Subscript k
m_i	Modulus	Channel Number	
s_{ik}	Thermometer Code Input from Comparators	Channel Number	Bit Order
p_{ik}	Position bit conversion	Channel Number	Bit Order
n_i	Dynamic Range Filtered Output	Bit Order	
g_i	Encoder Output	Bit Order	
h_i	System Output	Bit Order	

Table 1. List of Variables

Chapter III will outline the process used to convert the equations developed in Chapter II from general Boolean equations into MATLAB code and the appropriate least product term equations for the Xilinx Project Navigator schematic capture. The chapter is broken into two major sections; the first is on the implementation of the general equations from Chapter II to MATLAB code and the second section the implementation for the Xilinx schematic capture. Each section breaks down the conversion by the major components for the RSNS-to-binary converter.

Chapter IV is a detailed description of how the testing of the system had to be set up in order to verify the proper functioning of the RSNS-to-binary conversion logic. The first section of this chapter focuses on the development of the creation of inputs for the system. There are specific requirements placed on the inputs for the system and the second portion of this chapter ensures that these requirements are satisfied.

Chapter V shows the combined results of the test vectors and waveforms created in Chapter IV and the implementing of them via the program and circuit design in Chapter III.

Chapter VI wraps up the conclusions of the work and future work to be accomplished.

THIS PAGE INTENTIONALLY LEFT BLANK

II. RSNS RESIDUE-TO-BINARY CONVERSION

A. GENERAL BACKGROUND

The Robust Symmetrical Number System (RSNS) utilizes modulo arithmetic to decompose an integer into one or more symmetrical residues. Unlike other numbering systems, such as the Residue Numbering System (RNS) or Symmetrical Numbering System (SNS), the RSNS contains built-in redundancy to eliminate the need for extensive error correction [1]. In [1], three methods for converting RSNS residues to binary were discussed and this thesis utilizes the third method in which an algorithmic approach is used that is based on the underlying RNS structure of RSNS.

B. APPLICATION TO ADC

Information obtained for this chapter follows the previous work in [1] and utilizes the equations given therein to generate the equations for the basic design demonstrated in Chapter III. This Chapter attempts to summarize the information in [1] and use that information to develop the equations for this thesis.

The RSNS-to-binary portion developed in this thesis takes logical 1's and 0's from the outputs of a bank of 24 comparators at the end of an analog folding circuit. These comparator outputs are separated into 3 channels of moduli 7, 8, and 9 with the same number of comparator outputs as the moduli number for each channel. These comparator signals are the input to the RSNS-to-binary conversion system and are labeled s_{ij}, with each channel labeled with m_i. The three channels are broken down as follows; $m_1 = 7 \Rightarrow [s_{16}, s_{15}, s_{14}, s_{13}, s_{12}, s_{11}, s_{10}]$, $m_2 = 8 \Rightarrow [s_{27}, s_{26}, s_{25}, s_{24}, s_{23}, s_{22}, s_{21}, s_{20}]$, and $m_3 = 9 \Rightarrow [s_{38}, s_{37}, s_{36}, s_{35}, s_{34}, s_{33}, s_{32}, s_{31}, s_{30}]$ representing the thermometer code for each channel. A block diagram of the thermometer code to binary conversion algorithm is shown in Figure 1.

The dynamic range for the 3-channel system, with pair-wise relatively prime moduli, is found using equation (1) from [4].

$$\widehat{M}_{RSNS} = \frac{3}{2}m_1^2 + \frac{15}{2}m_1 + 7 \tag{1}$$

With $m_1 = 7$, the maximum dynamic range for the $m_1 = 7$, $m_2 = 8$, and $m_3 = 9$ system designed in this thesis is $\widehat{M}_{RSNS} = \frac{3}{2}(7^2) + \frac{15}{2}(7) + 7 = 133$.

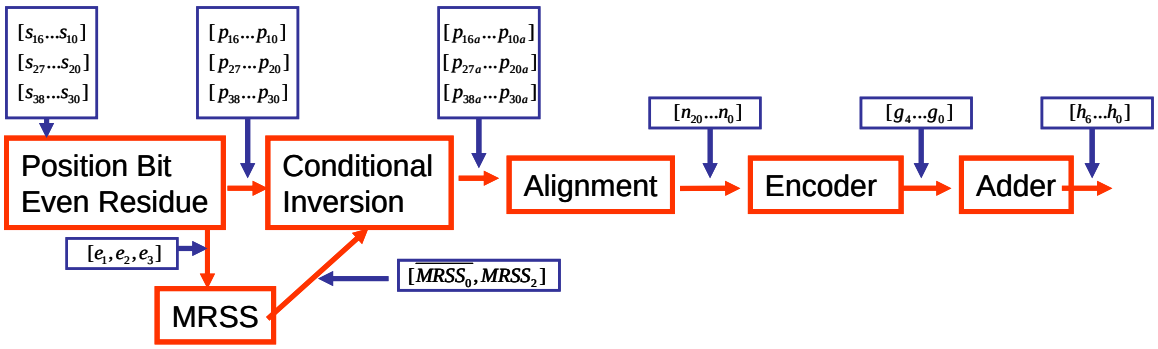


Figure 1. Block Diagram of the Symmetrical Residue-to-Binary Converter

1. Position Bit Equations

The Thermometer code inputs $[s_{16}...s_{10}]$, $[s_{27}...s_{20}]$, and $[s_{38}...s_{30}]$ are first converted to RSNS Position Bits. Position Bits are moduli dependent and are decomposed modulo residues of the Thermometer code inputs. Position Bits map from the Thermometer code inputs in accordance with the moduli of the channel. $[s_{16}...s_{10}]$, $[s_{27}...s_{20}]$, and $[s_{38}...s_{30}]$ map to $[p_{16}...p_{10}]$, $[p_{27}...p_{20}]$, and $[p_{38}...p_{30}]$ for $m_1 = 7$, $m_2 = 8$, and $m_3 = 9$. For $m_1 = 3$, $m_2 = 4$, and $m_3 = 5$, $[s_{12}...s_{10}]$, $[s_{23}...s_{20}]$, and $[s_{34}...s_{30}]$ map to $[p_{12}...p_{10}]$, $[p_{23}...p_{20}]$, and $[p_{34}...p_{30}]$.

The following equations were taken from [1] and used in the development of this thesis work. The following are the derived position bit equations utilizing binary thermometer code inputs from the analog circuit for moduli three, four, and five:

$$\begin{aligned} p_{10} &= \overline{s_{11}}, \\ p_{11} &= s_{11}, \\ p_{12} &= s_{10}\overline{s_{12}}, \end{aligned} \tag{2}$$

for $m_1 = 3$,

$$\begin{aligned} p_{20} &= \overline{s_{21}}, \\ p_{21} &= s_{21}\overline{s_{23}}, \\ p_{22} &= s_{22}, \\ p_{23} &= s_{20}\overline{s_{22}}, \end{aligned} \tag{3}$$

for $m_2 = 4$,

$$\begin{aligned} p_{30} &= \overline{s_{31}}, \\ p_{31} &= s_{31}\overline{s_{33}}, \\ p_{32} &= s_{33}, \\ p_{33} &= s_{32}\overline{s_{34}}, \\ p_{34} &= s_{30}\overline{s_{32}}, \end{aligned} \tag{4}$$

for $m_3 = 5$.

Using truth tables and Boolean algebra, the general forms of the above equations for even and odd moduli were shown to be:

$$\begin{aligned} p_{i0} &= \overline{s_{i1}}, \\ p_{i1} &= s_{i1}\overline{s_{i3}}, \\ p_{i2} &= s_{i3}\overline{s_{i5}}, \\ &\vdots \\ p_{i\left(\frac{m_i}{2}-1\right)} &= s_{i(m_i-3)}\overline{s_{i(m_i-1)}}, \\ p_{i\left(\frac{m_i}{2}\right)} &= s_{i(m_i-2)}, \\ p_{i\left(\frac{m_i}{2}+1\right)} &= s_{i(m_i-4)}\overline{s_{i(m_i-2)}}, \\ &\vdots \\ p_{i(m_i-3)} &= s_{i4}\overline{s_{i6}}, \\ p_{i(m_i-2)} &= s_{i2}\overline{s_{i4}}, \\ p_{i(m_i-1)} &= s_{i0}\overline{s_{i2}}, \end{aligned} \tag{5}$$

for even moduli, and

$$
\begin{aligned}
p_{i0} &= \overline{s_{i1}}, \\
p_{i1} &= s_{i1}\overline{s_{i3}}, \\
p_{i2} &= s_{i3}\overline{s_{i5}}, \\
&\vdots \\
p_{i\left(\left\lfloor \frac{m_i}{2} \right\rfloor -1\right)} &= s_{i(m_i-4)}\overline{s_{i(m_i-2)}}, \\
p_{i\left(\left\lfloor \frac{m_i}{2} \right\rfloor\right)} &= s_{i(m_i-2)}, \\
p_{i\left(\left\lfloor \frac{m_i}{2} \right\rfloor +1\right)} &= s_{i(m_i-3)}\overline{s_{i(m_i-1)}}, \\
&\vdots \\
p_{i(m_i-3)} &= s_{i4}\overline{s_{i6}}, \\
p_{i(m_i-2)} &= s_{i2}\overline{s_{i4}}, \\
p_{i(m_i-1)} &= s_{i0}\overline{s_{i2}},
\end{aligned}
\tag{6}
$$

for odd moduli. The term $\left\lfloor \frac{m_i}{2} \right\rfloor$ is defined as the greatest positive integer less than or equal to $\frac{m_i}{2}$, i.e., if the result is 3.54, then $\lfloor 3.54 \rfloor = 3$.

Equations (5) and (6) above were used to generate the signal bit mapping to position bits for the higher moduli used in this thesis. The equations are as follows:

$$
\begin{aligned}
p_{10} &= \overline{s_{11}}, \\
p_{11} &= s_{11}\overline{s_{13}}, \\
p_{12} &= s_{13}\overline{s_{15}}, \\
p_{13} &= s_{15}, \\
p_{14} &= s_{14}\overline{s_{16}}, \\
p_{15} &= s_{12}\overline{s_{14}}, \\
p_{16} &= s_{10}\overline{s_{12}},
\end{aligned}
\tag{7}
$$

for $m_1 = 7$, and

$$
\begin{aligned}
p_{20} &= \overline{s_{21}}, \\
p_{21} &= s_{21}\overline{s_{23}}, \\
p_{22} &= s_{23}\overline{s_{25}}, \\
p_{23} &= s_{25}\overline{s_{27}}, \\
p_{24} &= s_{26}, \\
p_{25} &= s_{24}\overline{s_{26}}, \\
p_{26} &= s_{22}\overline{s_{24}}, \\
p_{27} &= s_{20}\overline{s_{22}},
\end{aligned}
\tag{8}
$$

for $m_2 = 8$, and

$$
\begin{aligned}
p_{30} &= \overline{s_{31}}, \\
p_{31} &= s_{31}\overline{s_{33}}, \\
p_{32} &= s_{33}\overline{s_{35}}, \\
p_{33} &= s_{35}\overline{s_{37}}, \\
p_{34} &= s_{37}, \\
p_{35} &= s_{36}\overline{s_{38}}, \\
p_{36} &= s_{34}\overline{s_{36}}, \\
p_{37} &= s_{32}\overline{s_{34}}, \\
p_{38} &= s_{30}\overline{s_{32}},
\end{aligned}
\tag{9}
$$

for $m_3 = 9$. An example of the above equations would be $[s_{16}...s_{10}]$= [0,0,1,1,1,1,1] would map to $[p_{16}...p_{10}]$ = [0,0,1,0,1,0,0].

2. Even Residue Flags

After position bit conversion, the next step is to determine if the Thermometer code input to the three channels is even or odd. This is done in by counting the number of input bits that are one for each of the Thermometer code channels and asserting the even signal if the number is even and not asserting the signal if the number is odd. That is, for channel two $(m_2 = 8)$ if the Thermometer code input bits are $[s_{20}...s_{27}] = [1,1,1,1,0,0,0,0]$, then the even residue flag for channel two would be asserted since there are four ones in the thermometer code input.

The equations for an even residue with $m_1 = 3$, $m_2 = 4$, and $m_3 = 5$, from [1], are shown in equation (10). The e represents an even bit flag, and will be asserted when the channel is even, with the subscript designating the channel.

$$\begin{aligned} e_1 &= \overline{s_{10}} + s_{11}\overline{s_{12}}, \\ e_2 &= \overline{s_{20}} + s_{21}\overline{s_{22}} + s_{23}, \\ e_3 &= \overline{s_{30}} + s_{31}\overline{s_{32}} + s_{33}\overline{s_{34}}, \end{aligned} \tag{10}$$

In general, the equations for even moduli (11) and for odd moduli (12) can be generated as

$$e_i = \overline{s_{i0}} + s_{i1}\overline{s_{i2}} + s_{i3}\overline{s_{i4}} + \cdots + s_{i(m_i-1)}, \tag{11}$$

$$e_i = \overline{s_{i0}} + s_{i1}\overline{s_{i2}} + s_{i3}\overline{s_{i4}} + \cdots + s_{i(m_i-2)}\overline{s_{i(m_i-1)}}, \tag{12}$$

This thesis chose to use equations (11) and (12) to get the following equations for the even flags for $m_1 = 7$, $m_2 = 8$, and $m_3 = 9$.

$$\begin{aligned} e_1 &= \overline{s_{10}} + s_{11}\overline{s_{12}} + s_{13}\overline{s_{14}} + s_{15}\overline{s_{16}}, \\ e_2 &= \overline{s_{20}} + s_{21}\overline{s_{22}} + s_{23}\overline{s_{24}} + s_{25}\overline{s_{26}} + s_{27}, \\ e_3 &= \overline{s_{30}} + s_{31}\overline{s_{32}} + s_{33}\overline{s_{34}} + s_{35}\overline{s_{36}} + s_{37}\overline{s_{38}}, \end{aligned} \tag{13}$$

Following the example above for $m_2 = 8$, $e_2 = 1$ would be asserted.

3. Modulus Residue Sub-Sequence Flags

The Modulus Residue Sub-Sequence (MRSS) flags are used to determine if the position bits of a channel $\left(p_{ij} \right)$ must have the bit order reversed. The discussion of this in [1] explains how these flags effect the conditional bit reversal of channel two and three of the system. Equation (14) shows the general equations from [1], with the sub-script N being the number of channels in the system.

$$
\begin{aligned}
MRSS_{N-1} &= e_1 \oplus e_2, \\
MRSS_{N-2} &= e_2 \oplus e_3, \\
&\vdots \\
MRSS_2 &= e_{N-2} \oplus e_{N-1}, \\
MRSS_1 &= e_N \oplus e_{N-1}, \\
\overline{MRSS_0} &= e_N \oplus e_1,
\end{aligned}
\tag{14}
$$

The three-channel circuit utilized in this thesis uses equation (14) to generate equation (15), which is used in the next section to perform the conditional inversion of the position bits.

$$
\begin{aligned}
\overline{MRSS_0} &= e_3 \oplus e_1, \\
MRSS_1 &= e_3 \oplus e_2, \\
MRSS_2 &= e_2 \oplus e_1,
\end{aligned}
\tag{15}
$$

For example, for $e_3 = 1$, $e_2 = 1$, and $e_1 = 0$, $MRSS_2 = 1$ and $\overline{MRSS_0} = 1$

4. Conditional Bit Reversal

Conditional bit reversal of the position bits is based on the MRSS's, where each channel is reversed depending on a different MRSS flag. When a channel is reversed, its bit order is reversed, i.e., if the signal $\left[p_{20,} p_{21} ... p_{27} \right]$ met the conditions to be inverted it would be transposed to $\left[p_{27,} p_{26} ... p_{20} \right]$. In order to keep continuity of variable naming, all position bits after the conditional bit reversal will have a subscript of "a" added to delineate them from the bits before the conditional bit reversal logic, i.e., channel two position bit labels would be changed to $\left[p_{20a,} p_{21a} ... p_{27a} \right]$

Channel one is never reversed. Channel two is bit reversed if $MRSS_2$ is asserted. Channel three is reversed if $\overline{MRSS_0}$ is asserted. This can be accomplished with simple multiplexers. The logic and equations used for this will be shown in Chapter III. For our 3-channel case, $MRSS_1$ is not used.

5. Alignment Logic

The RSNS relies on least positive solution (LPS) positional alignment to align the congruence equations resulting from the incoming RSNS symmetrical residue vectors. It can be shown that the RNS residue position bits for the $m_1 = 3$, $m_2 = 4$, and $m_3 = 5$ system can be aligned using eight 3-input NAND gates. Using 3-input NAND gates the $m_1 = 7$, $m_2 = 8$, and $m_3 = 9$ system can accomplish the alignment using twenty-one gates. Utilizing the code in the appendices of [1], the start point for the alignment logic is $\overline{p_{14a} p_{23a} p_{36a}}$. Equation (16) shows all twenty-one of the gate equations needed for the system.

$$
\begin{aligned}
n_0 &= \overline{p_{14a} p_{23a} p_{36a}}, \\
n_1 &= \overline{p_{15a} p_{24a} p_{37a}}, \\
n_2 &= \overline{p_{16a} p_{25a} p_{38a}}, \\
n_3 &= \overline{p_{10a} p_{26a} p_{30a}}, \\
n_4 &= \overline{p_{11a} p_{27a} p_{31a}}, \\
n_5 &= \overline{p_{12a} p_{20a} p_{32a}}, \\
n_6 &= \overline{p_{13a} p_{21a} p_{33a}}, \\
n_7 &= \overline{p_{14a} p_{22a} p_{34a}}, \\
n_8 &= \overline{p_{15a} p_{23a} p_{35a}}, \\
n_9 &= \overline{p_{16a} p_{24a} p_{36a}}, \\
n_{10} &= \overline{p_{10a} p_{25a} p_{37a}}, \\
n_{11} &= \overline{p_{11a} p_{26a} p_{38a}}, \\
n_{12} &= \overline{p_{12a} p_{27a} p_{30a}}, \\
n_{13} &= \overline{p_{13a} p_{20a} p_{31a}}, \\
n_{14} &= \overline{p_{14a} p_{21a} p_{32a}}, \\
n_{15} &= \overline{p_{15a} p_{22a} p_{33a}}, \\
n_{16} &= \overline{p_{16a} p_{23a} p_{34a}}, \\
n_{17} &= \overline{p_{10a} p_{24a} p_{35a}}, \\
n_{18} &= \overline{p_{11a} p_{25a} p_{36a}}, \\
n_{19} &= \overline{p_{12a} p_{26a} p_{37a}}, \\
n_{20} &= \overline{p_{13a} p_{27a} p_{38a}},
\end{aligned}
\tag{16}
$$

This range determination of n is the longest vector, where the combinations of the positions bits $\left[p_{1ja}, p_{2ka}, p_{3la} \right]$ does not have a repeated combination of j, k, and l.

Within the dynamic range of the system, only one NAND gate output from equation (16) will be active at a time. This unique attribute of the RSNS-to-binary algorithm is where the usefulness of RSNS is realized.

6. Encoder

The outputs from the alignment logic are encoded into a 5-bit output because only one NAND gate is active at one time and 21 can be encoded in binary in 5 bits. The equations for the encoding logic are not difficult and will not be included in this section. They are shown graphically in Chapter III.

7. Adder

The adder designed for this thesis follows the equation shown in Figure 1 and is for the $m_1 = 3$, $m_2 = 4$, and $m_3 = 5$ system [1]. The RSNS-to-binary implementation in this thesis is three-channels and uses the same basic equations. However, the equations are expanded in this thesis so that the inputs to the adder are consistent with the 5-bit output from the Encoder described in the previous section. This is shown in Figure 3.

$$\begin{array}{cccccccc} g_3 & g_2 & g_1 & g_0 & \overline{e_1} & \overline{MRSS_0} & \leftarrow MRSS_2 \text{ carry-in} \\ & g_3 & g_2 & g_1 & g_0 & \overline{e_1} & + \\ \hline h_5 & h_4 & h_3 & h_2 & h_1 & h_0 & \end{array}$$

Figure 2. Adder implementation for RSNS-to-binary for [1].

$$\begin{array}{ccccccccc} g_4 & g_3 & g_2 & g_1 & g_0 & \overline{e_1} & \overline{MRSS_0} & \leftarrow MRSS_2 \text{ carry-in} \\ & g_4 & g_3 & g_2 & g_1 & g_0 & \overline{e_1} & + \\ \hline h_6 & h_5 & h_4 & h_3 & h_2 & h_1 & h_0 & \end{array}$$

Figure 3. Adder implementation of RSNS-to-binary for this thesis.

The chapter contained an explanation of the concept for the Symmetrical Residue-to-binary conversion algorithm. This information will be the basis for the following chapters and will aid the process of designing the code and logic in the next chapter.

III. LOGIC DESIGN FOR ROBUST SYMMETRICAL NUMBERING SYSTEM LOGIC BLOCK

An overview of the RSNS circuit for processing the thermometer code produced by the analog folding circuits is shown in Figure 4. The 6 major levels of logic were developed from the theory and equations discussed in Chapter II.

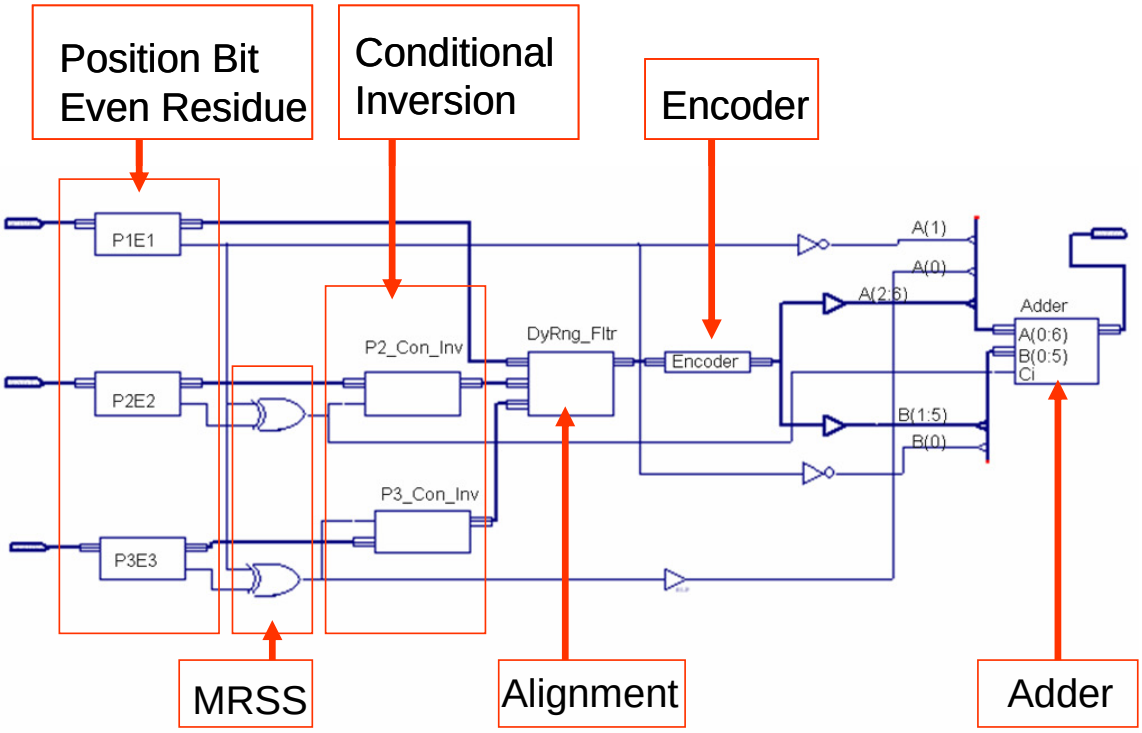


Figure 4. Hierarchical View of RSNS-to-binary converter

A. MATLAB REALIZATION

Programming the RSNS-to-binary logic equations in MATLAB, for verification of the equations derived in Chapter II, was the next step in developing the $m_1 = 7$, $m_2 = 8$, and $m_3 = 9$ converter. This section walks through the basic algorithm to convert the binary logic equations to MATLAB logical equations. The code developed during this phase was based on the assumption that the thermometer code, or symmetrical

residues, would be passed as a parameter to the system. Appendix B shows the code created by using the algorithm shown in the following sections.

First the algorithm maps the input to the proper variable naming for the system, i.e., it takes the twenty-four binary input bits and renames them $\left[S_{10}...S_{16}\right]$, $\left[S_{20}...S_{27}\right]$, and $\left[S_{30}...S_{38}\right]$. The twenty-four bits are then processed as described in the remainder of this section.

1. Thermometer Code to Position Bit Conversion

Equations (7), (8), and (9) show the general Position Bit conversion equations for the RSNS-to-binary converter developed in this thesis. Changing these equations from standard Boolean format to MATLAB syntax we would get, for example, $p_{31} = s_{31}\overline{s_{33}}$ mapped to P31=and(S31,not(S33)). Apply this mapping to equations (7), (8), and (9), gives the position bit conversion for $m_1 = 7$, $m_2 = 8$, and $m_3 = 9$.

2. Even Residue Flags

Equation (13) shows the specific Even Residue Flag equations. A simple mapping to MATLAB syntax from equation (13) would map $e_1 = \overline{s_{10}} + s_{11}\overline{s_{12}} + s_{13}\overline{s_{14}} + s_{15}\overline{s_{16}}$ to

E1=or(not(S10),or(and(S11,not(S12)),or(and(S13,not(S14)),and(S15,not(S16))))).

Applying this mapping to the remainder of equation (13), gives the even residue flags for $m_1 = 7$, $m_2 = 8$, and $m_3 = 9$.

3. MRSS

Equation (15) shows the MRSS flag generation. The MATLAB syntax for $MRSS_1 = e_3 \oplus e_2$ is MRSS1=xor(E3,E2). Apply this syntax to the remaining portion of (15) gives the MRSS flags for $m_1 = 7$, $m_2 = 8$, and $m_3 = 9$.

4. Conditional Bit Reversal

Conditional bit reversal is accomplished by use of a multiplexer. Multiplexers are implemented using an OR of the AND of a signal and its compliment with the two outputs, i.e., P20a is either P27 or P20, depending on MRSS2 being asserted or not asserted, respectively. The Boolean expression for this multiplexer portion would be $p_{20a} = (p_{27} MRSS_2 + p_{20} \overline{MRSS_2})$, which is then converted and mapped in MATLAB to P20a=or(and(P27,MRSS2),and(P20,not(MRSS2))). Applying this logic to all the positions bits of channel two and three generates the conditional bit reversal for $m_1 = 7$, $m_2 = 8$, and $m_3 = 9$. Figure 5 shows a 2-to-1 multiplexer where the output is input 1 if the control signal is asserted and input 2 otherwise.

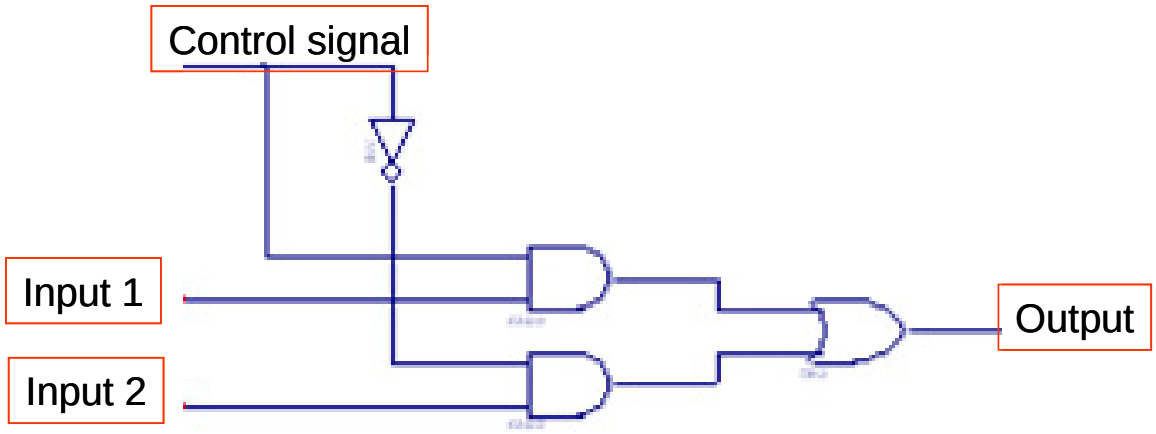


Figure 5. 2-to-1 Multiplexer

5. Alignment Logic

The theory presented in Chapter II called for 3-input NAND gates to be used in the alignment logic. However, MATLAB does not have NAND commands. Therfore, modifications to the system had to be made to the alignment logic to allow the use of an AND gate in the MATLAB code. This was accomplished by following which signal should be asserted and adjusting accordingly to allow proper use of MATLAB coding. Equation (16) shows all equations for the alignment, by mapping $n_0 = \overline{p_{14a} p_{23a} p_{36a}}$ to n0=(and (P14a,and(P23a,P36a))); and continuing this for the rest of (16) we get the Alignment Logic portion.

6. Encoder

The Encoder expresses the subscript number of the asserted AND gate and generates a five bit output, [g4,g3,g2,g1,g0]. For example, if n14 was the active AND gate, then the output of the encoder would be [0,1,1,1,0]. The equations utilized in Appendix B were derived from the truth table shown in Table 2. Each output bit of the Encoder is the OR of all the logic 1s in the column below the respective output designator in Table 2. For example, if we look at g4, there are logic 1s at n16, n17, n18, n19 and n20. If any of these gates are active then g4 will be asserted. Thus, $g4 = n16 + n17 + n18 + n19 + n20$ (where + represents OR) which converts to MATLAB of g4=or(n16,or(n17,or(n18,or(n19,n20)))). Following this we get the Encoder from Table 2.

Active AND gate		g4	g3	g2	g1	g0
n0		0	0	0	0	0
n1		0	0	0	0	1
n2		0	0	0	1	0
n3		0	0	0	1	1
n4		0	0	1	0	0
n5		0	0	1	0	1
n6		0	0	1	1	0
n7		0	0	1	1	1
n8		0	1	0	0	0
n9		0	1	0	0	1
n10		0	1	0	1	0
n11		0	1	0	1	1
n12		0	1	1	0	0
n13		0	1	1	0	1
n14		0	1	1	1	0
n15		0	1	1	1	1
n16		1	0	0	0	0
n17		1	0	0	0	1
n18		1	0	0	1	0
n19		1	0	0	1	1
n20		1	0	1	0	0

Table 2. Encoder Truth Table

7. Adder

The adder is a standard 7 bit adder. Once the inputs are arranged according to Figure 3, the equations for the adder can be produced as shown in Appendix B. The output is given by 7 matrices, h0, h1, h2, h3, h4, h5 and h6. Since the binary output is not easily verified using MATLAB plots, a matrix called decimal was created for easier visual verification of the output and is the decimal conversion of the h6 to h0 binary output. Verification and testing for this code will be discussed in Chapters IV and V.

B. XILINX REALIZATION

This section shows the schematics created in Xilinx Project Navigator to create the RSNS-to-binary conversion logic. Xilinx was chosen for this because it was a readily available program with schematic capture functions, along with a good test-bench simulator for verification of proper circuit operation. The equations used were modifications of those in Chapter II. The only changes were the application of DeMorgan's Theorem to those equations in order to predominantly use NAND, NOR, and INVERTER gates only. A few XORs were used for the adder and MRSS logic blocks. Note, in the following schematic diagrams, the labels are as in Xilinx Project Navigator.

1. Thermometer Code to Position Bit Conversion and Even Residue Flags

For $m_1 = 7$, equation (7) and (13) are used, and DeMorgan's Theorem was applied to get equation (17) and (18), and when drawn in Xilinx, results in Figure 6.

$$
\begin{aligned}
p_{10} &= \overline{s_{11}}, \\
p_{11} &= s_{11}\overline{s_{13}} = \overline{\overline{s_{11}} + s_{13}}, \\
p_{12} &= s_{13}\overline{s_{15}} = \overline{\overline{s_{13}} + s_{15}}, \\
p_{13} &= s_{15}, \\
p_{14} &= s_{14}\overline{s_{16}} = \overline{\overline{s_{14}} + s_{16}}, \\
p_{15} &= s_{12}\overline{s_{14}} = \overline{\overline{s_{12}} + s_{14}}, \\
p_{16} &= s_{10}\overline{s_{12}} = \overline{\overline{s_{10}} + s_{12}}
\end{aligned}
\qquad (17)
$$

$$e_1 = \overline{s_{10}} + s_{11}\overline{s_{12}} + s_{13}\overline{s_{14}} + s_{15}\overline{s_{16}} = \overline{s_{10}\left(\overline{s_{11}\overline{s_{12}}}\right)\left(\overline{s_{13}\overline{s_{14}}}\right)\left(\overline{s_{15}\overline{s_{16}}}\right)} \tag{18}$$

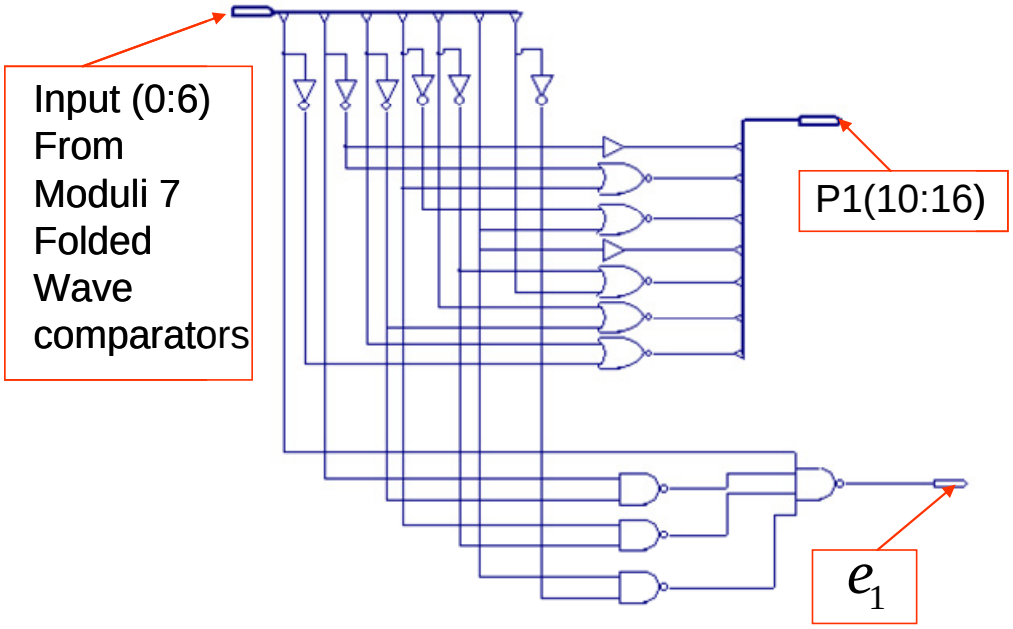


Figure 6. Channel 1 Position Bit and Even Bit Flag Schematic

For $m_2 = 8$, equation (8), and (13) are used, and DeMorgan's Theorem applied to get equation (19) and (20), and when drawn in Xilinx, results in Figure 7.

$$\begin{aligned}
p_{20} &= \overline{s_{21}}, \\
p_{21} &= s_{21}\overline{s_{23}} = \overline{\overline{s_{21}} + s_{23}}, \\
p_{22} &= s_{23}\overline{s_{25}} = \overline{\overline{s_{23}} + s_{25}}, \\
p_{23} &= s_{25}\overline{s_{27}} = \overline{\overline{s_{25}} + s_{27}}, \\
p_{24} &= s_{26}, \\
p_{25} &= s_{24}\overline{s_{26}} = \overline{\overline{s_{24}} + s_{26}}, \\
p_{26} &= s_{22}\overline{s_{24}} = \overline{\overline{s_{22}} + s_{24}}, \\
p_{27} &= s_{20}\overline{s_{22}} = \overline{\overline{s_{20}} + s_{22}},
\end{aligned} \tag{19}$$

$$e_2 = \overline{s_{20}} + s_{21}\overline{s_{22}} + s_{23}\overline{s_{24}} + s_{25}\overline{s_{26}} + s_{27} = \overline{s_{20}\left(s_{21}\overline{s_{22}}\right)\left(s_{23}\overline{s_{24}}\right)\left(s_{25}\overline{s_{26}}\right)\overline{s_{27}}} \qquad (20)$$

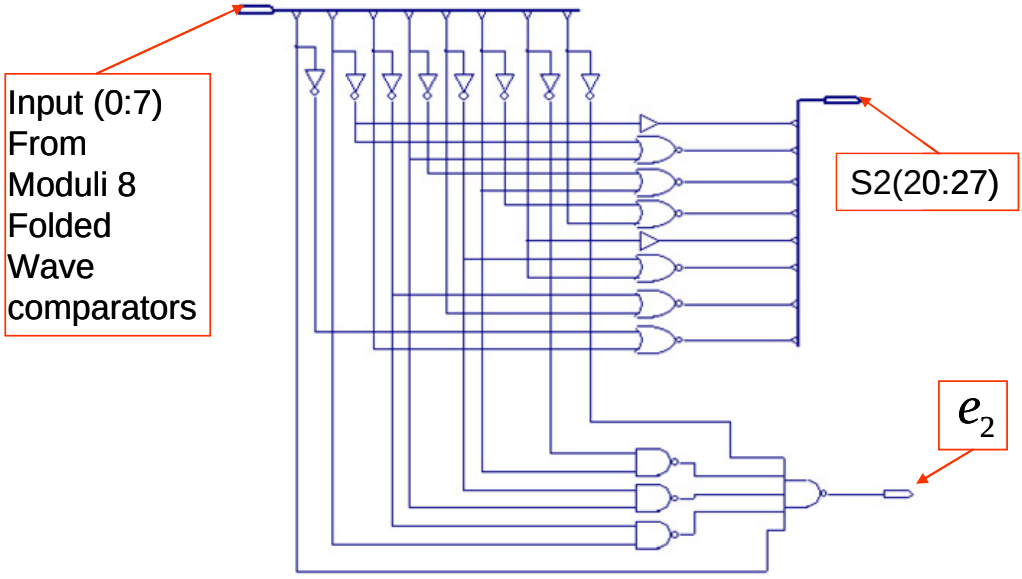


Figure 7. Channel 2 Position Bit and Even Bit Flag Schematic

For $m_3 = 9$, equation (9) and (13) are used, and DeMorgan's Theorem applied to get equation (19) and (22), and when drawn in Xilinx, results in Figure 8.

$$
\begin{aligned}
p_{30} &= \overline{s_{31}}, \\
p_{31} &= s_{31}\overline{s_{33}} = \overline{\overline{s_{31}} + s_{33}}, \\
p_{32} &= s_{33}\overline{s_{35}} = \overline{\overline{s_{33}} + s_{35}}, \\
p_{33} &= s_{35}\overline{s_{37}} = \overline{\overline{s_{35}} + s_{37}}, \\
p_{34} &= s_{37}, \\
p_{35} &= s_{36}\overline{s_{38}} = \overline{\overline{s_{36}} + s_{38}}, \\
p_{36} &= s_{34}\overline{s_{36}} = \overline{\overline{s_{34}} + s_{36}}, \\
p_{37} &= s_{32}\overline{s_{34}} = \overline{\overline{s_{32}} + s_{34}}, \\
p_{38} &= s_{30}\overline{s_{32}} = \overline{\overline{s_{30}} + s_{32}},
\end{aligned}
\tag{21}
$$

$$
e_3 = \overline{s_{30}} + s_{31}\overline{s_{32}} + s_{33}\overline{s_{34}} + s_{35}\overline{s_{36}} + s_{37}\overline{s_{38}} = \overline{s_{30}\overline{\left(s_{31}\overline{s_{32}}\right)}\overline{\left(s_{33}\overline{s_{34}}\right)}\overline{\left(s_{35}\overline{s_{36}}\right)}\overline{\left(s_{37}\overline{s_{38}}\right)}}
\tag{22}
$$

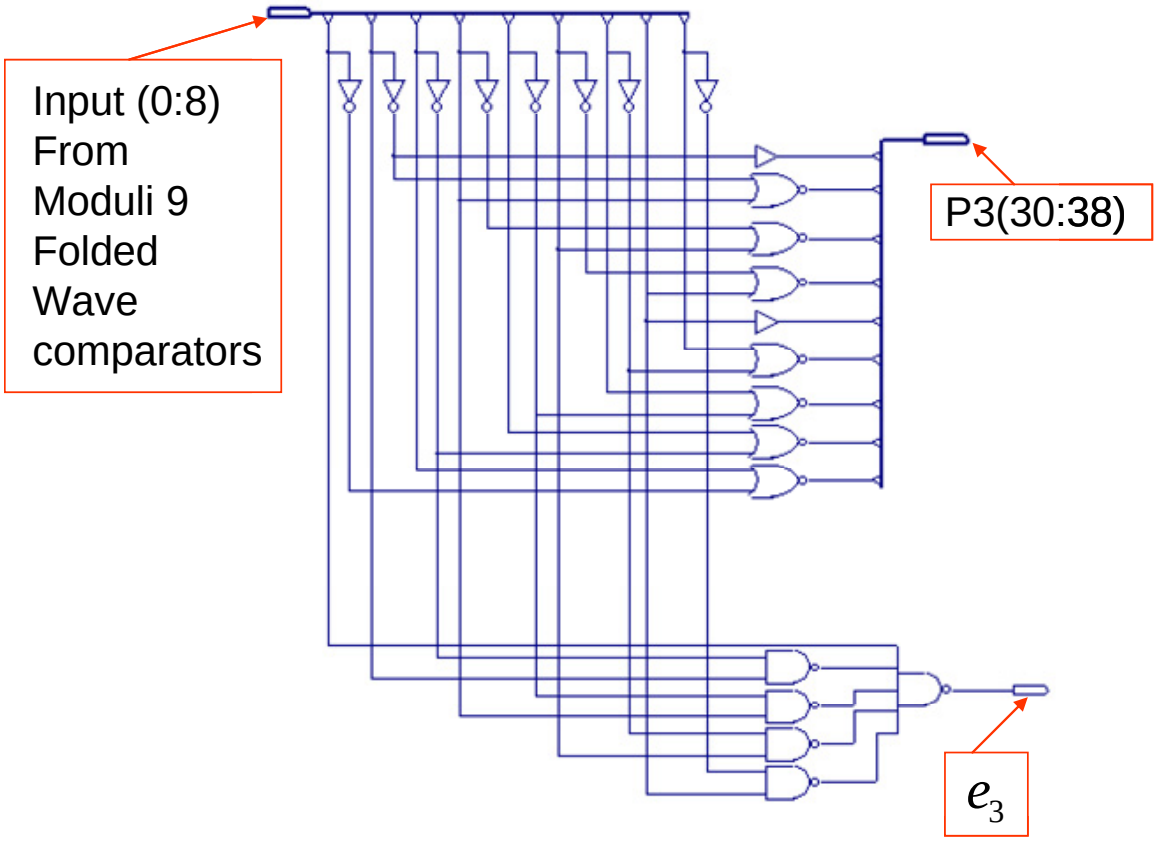


Figure 8. Channel 3 Position Bit and Even Bit Flag Schematic

2. Modulus Residue Sub-Sequence Flags

Equation (15) is used to get Figures 9 and 10.

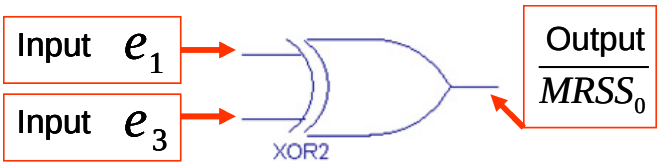


Figure 9. $\overline{MRSS_0}$ Xilinx Schematic

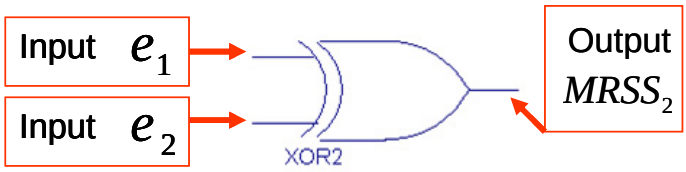


Figure 10. $MRSS_2$ Xilinx Schematic

3. Conditional Bit Reversal

Channel 1 is never reversed. Therefore, in Xilinx, P1(6:0) will be a single bus line to the next block of logic. Channel 2, which is an 8-bit to 8-bit multiplexer with $MRSS_2$ as the control signal, is shown in Figure 11. Again, this reverses the Position Bit order when $MRSS_2$ is asserted.

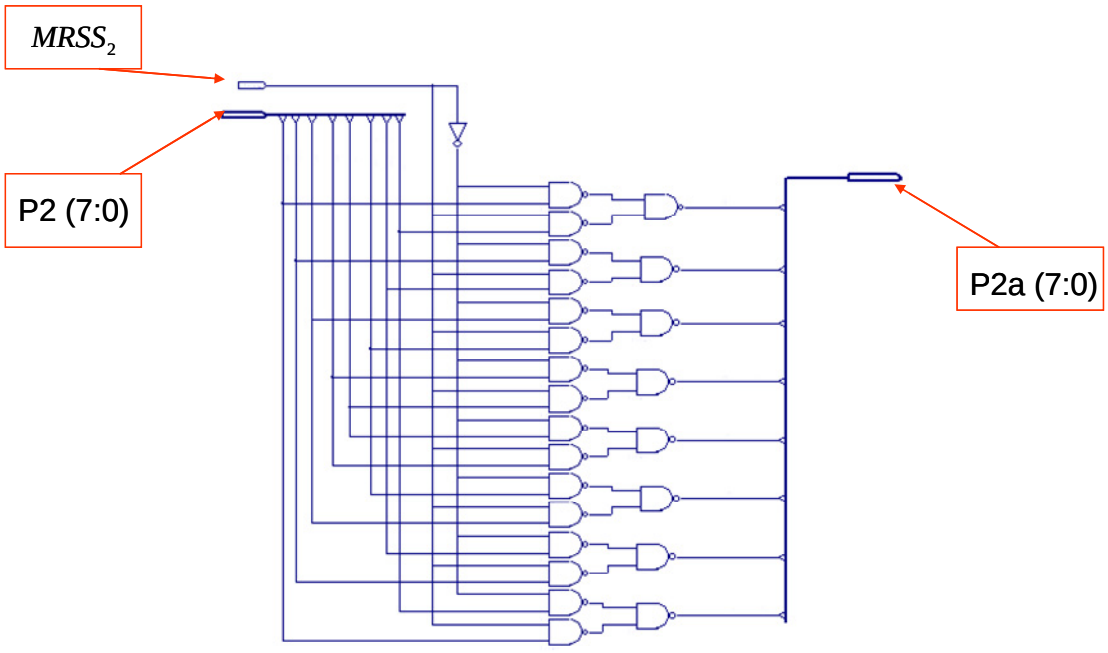


Figure 11. Channel 2 Conditional Bit Reversal Schematic

Channel 3, which is a 9-bit to 9-bit multiplexer with $\overline{MRSS_0}$ as the control signal, as shown in Figure 12. Again, this inverts the Position Bit order when $\overline{MRSS_0}$ is asserted.

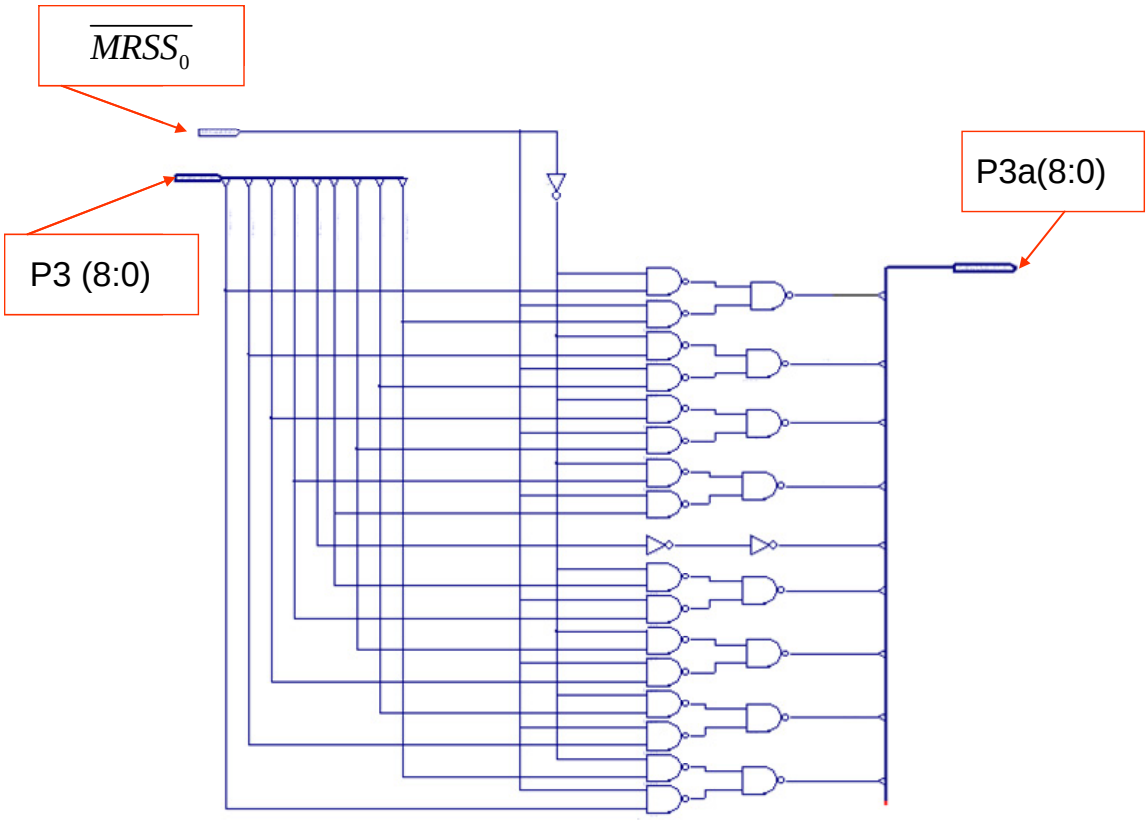


Figure 12. Channel 3 Conditional Bit Reversal Schematic

4. Alignment Logic

Figure 13 is a direct mapping of equation (16). The only modification is the insertion of inverters for proper functioning of the follow on encoder.

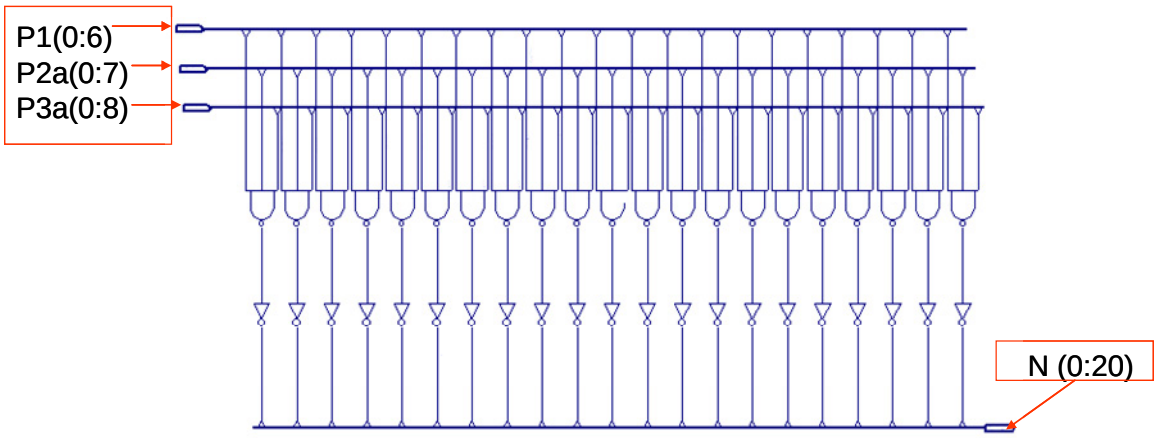


Figure 13. Alignment Logic for Position Bits after Inversion

5. Encoder

The component equations in Equation (23) are the logical binary expressions for a 21 to 5 bit encoder. The circuit is shown in Figure 14. This simple logic is all that is required for this section because only one NAND in the Alignment Logic can be active in the Dynamic Range at one time.

$$
\begin{aligned}
g_0 &= \overline{(n_1+n_3+n_5)(n_7+n_9+n_{11})(n_{13}+n_{15})(n_{17}+n_{19})} \\
g_1 &= \overline{(n_2+n_3+n_6)(n_7+n_{10}+n_{11})(n_{14}+n_{15})(n_{18}+n_{19})} \\
g_2 &= \overline{(n_4+n_5+n_6)(n_7+n_{12}+n_{13})(n_{14}+n_{15}+n_{20})} \\
g_3 &= \overline{(n_8+n_9+n_{10})(n_{11}+n_{12}+n_{13})(n_{14}+n_{15})} \\
g_4 &= \overline{(n_{16}+n_{17}+n_{18})(n_{19}+n_{20})}
\end{aligned}
\tag{23}
$$

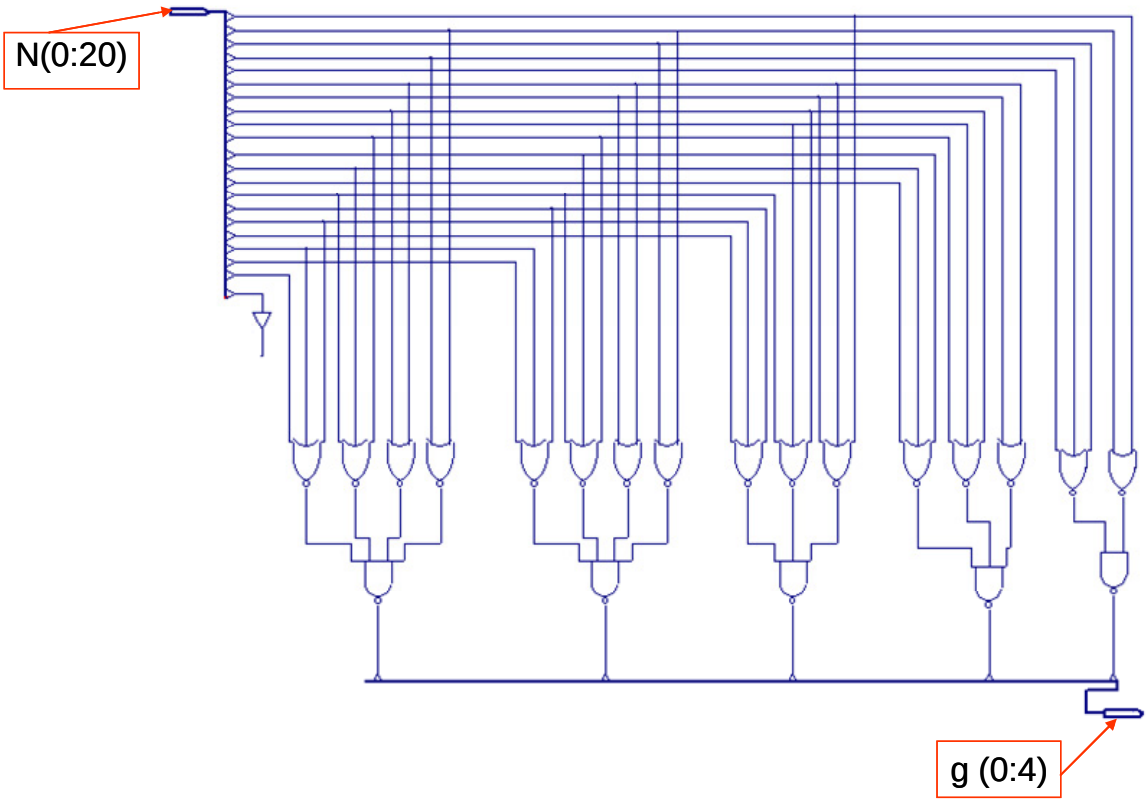


Figure 14. Binary 21 to 5 Bit Encoder Schematic

6. Adder

A standard adder design was selected for this thesis vice a carry-look-ahead adder since the logic involved in the design of the RSNS-to-binary logic will be faster than the analog circuit providing the inputs. Figure 15 shows the adder circuit and Figure 3 shows the proper bit alignment for the inputs.

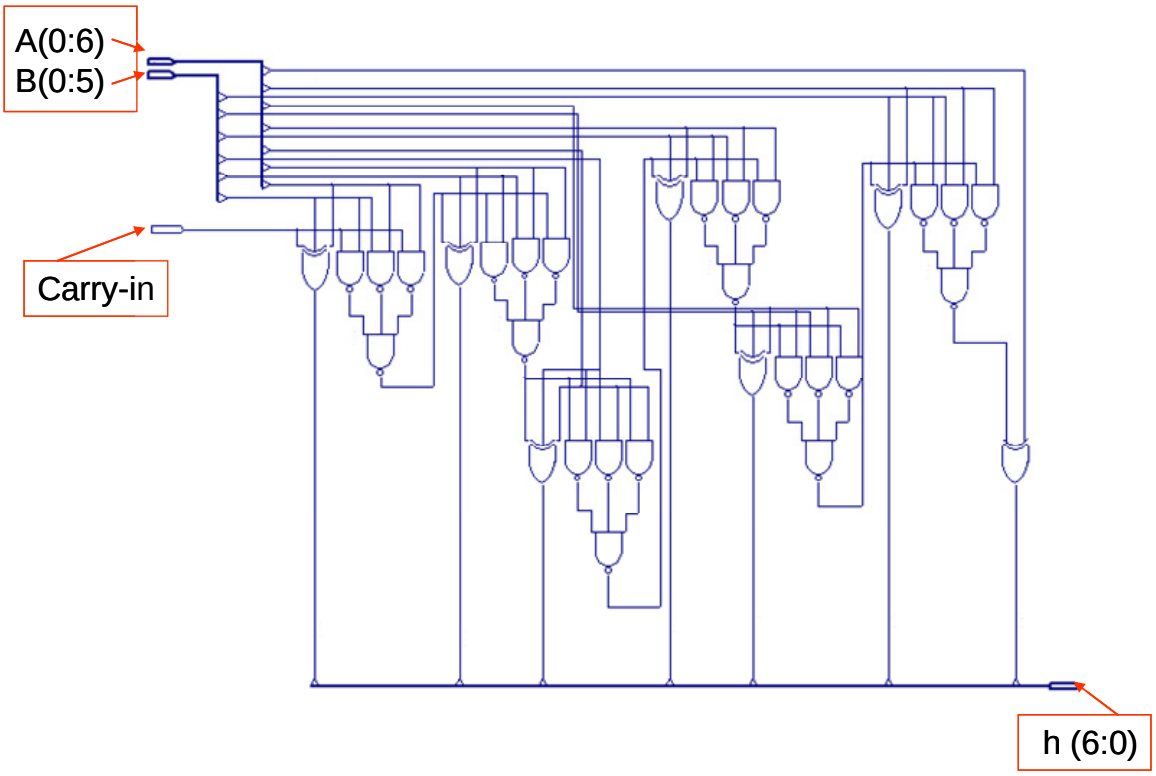


Figure 15. 7 Bit Adder Schematic

This chapter has shown the implementation of the theory and design of the previous chapter. The next step is to ensure the inputs to these systems meets the RSNS residue requirements of the Residue-to-Binary converter.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. TEST DEVELOPMENT AND VERIFICATION

A. THERMOMETER CODE DEVELOPMENT

The inputs for the RSNS-to Binary logic must be the three thermometer codes for each channel, having RSNS Gray code properties for all 24 inputs. Table 3 demonstrates a small section of the thermometer code for $m_1 = 7$ and shows that higher-order bits can only be asserted as 1 after all the less significant bits are asserted to 1. The bits fill and ebb in bit order from lower to higher without skipping any bits. The thermometer code property is present in all three RSNS channels.

Bit number	t = 1	t = 2	t = 3	t = 4	t = 5	t = 6	t = 7	t = 8
P16	0	0	1	0	0	0	0	0
P15	0	1	1	1	0	0	0	0
P14	1	1	1	1	1	0	0	0
P13	1	1	1	1	1	1	0	0
P12	1	1	1	1	1	1	1	0
P11	1	1	1	1	1	1	1	1
P10	1	1	1	1	1	1	1	1

Table 3. Channel 1 Thermometer Code

Gray code sequences only change 1 binary bit in the code group when going from one sample period to the next [5]. Table 4 shows the Gray code scheme used for the RSNS-to-Binary logic. The number in the channel rows are the number of 1's asserted for that channel. As shown in Table 4, only one bit in all channels changes at any time sampling. In terms of the circuit, this means only one of twenty-four comparator inputs change at any one sample time. Another fact demonstrated in Table 4 is that each

channel will hold a value for three sample periods before changing. This is to assist in ensuring the Gray code property of the inputs.

Channel		t=1	t=2	t=3	t=4	t=5	t=6	t=7	t=8
1		6	6	6	5	5	5	4	4
2		6	6	7	7	7	8	8	8
3		6	5	5	5	4	4	4	3

Table 4. RSNS Gray Code Property Showing the Number of Comparators for each channel

Another property of the inputs is that the channels are time shifted. Channels two and three are shifted left one and two sample periods, respectively, as is shown in Table 4. This is critical to the conversion process and assists in the Gray code functionality of the system.

The development of the thermometer code for testing both the MATLAB code and the Xilinx Project Navigator schematic turned out to be much more difficult than initially expected because of these unique input requirements. The following section will elaborate on how this was accomplished for the testing and ultimate verification of the RSNS-to-binary algorithm.

1. MATLAB

Creating a thermometer code with Gray code properties in MATLAB was a tedious task. The first task was to create twenty-four matrices to represent the twenty-four inputs from the comparators. To accomplish this, the matrices were sorted by channel, that is, utilizing the fact that there are seven matrices in channel 1, $m_1 = 7$, eight matrices representing channel 2, $m_2 = 8$, and nine matrices representing channel 3, $m_3 = 9$. Each channel holds a signal for three sampling periods. One full cycle of the thermometer code would be from 0 to m_i down to 1, i.e., with m_1=7, one full cycle would be[0, 1, 2, 3, 4, 5, 6, 7, 6, 5, 4, 3, 2, 1]. With the 3 sample hold and thermometer

code properties applied, it can be seen that the size of a matrix for a RSNS moduli is $3(2m_i - 1)$. Applying this equation to moduli $m_1 = 7$, $m_2 = 8$, and $m_3 = 9$, it is found that the matrices for the channels are 42, 48, and 54, respectfully. Utilizing these numbers, the MATLAB code in Appendix A generates matrices for each comparator output of the appropriate length for one full thermometer cycle and include the appropriate shifts for channels two and three.

The next step is to make all the matrices equal size in order to cover all possible combinations of the shifted channels. This was accomplished by using iterative 'for' loops to expand the matrices. Equation (24) was used to establish the number of copies of each matrix that must be concatenated together in order facilitate proper alignment.

$$\begin{aligned} m_1 &\Rightarrow [3(2m_2 - 1) * 3(2m_3 - 1)] - 1, \\ m_2 &\Rightarrow [3(2m_1 - 1) * 3(2m_3 - 1)] - 1, \\ m_3 &\Rightarrow [3(2m_1 - 1) * 3(2m_2 - 1)] - 1, \end{aligned} \tag{24}$$

Once the first two sections of MATLAB code in Appendix A are executed, twenty-four matrices of equal length, and encompassing all combinations of alignment, are produced and ready to be passed to the code in Appendix B for processing.

2. Field Programmable Gate Array Schematic Capture

Xilinx test-bench software does not have a thermometer code production function, but it does have a resetting counter. The resetting counter will not produce the sequence [0, 1, 2, 3, 2, 1, 0, 1…] but will produce [0, 1, 2, 3, 0, 1, 2, 3, 0…]. Therefore to get thermometer code inputs for the system, some digital logic to convert the resetting counter to a thermometer code had to be designed. The outputs of reset counters are the binary equivalent of the decimal value and the input to the thermometer code generators described in the following paragraphs and tables. As an example, Table 5 shows the inputs to the thermometer code generator for $m_1 = 7$ as [I13, I12, I11, I10].

Table 5 shows a truth table for channel 1. The information in Table 5 is then put into Karnaugh maps for bits S10 thru S16 as shown in Figures 16, 17, 18, and 19. These

Karnaugh maps produce the least sum equations shown in Equation(25). These equations were then utilized to implement the channel 1 thermometer code generator in Xilinx, as shown in Figure 20.

Decimal	Therm		I13	I12	I11	I10		S16	S15	S14	S13	S12	S11	S10
0	0		0	0	0	0		0	0	0	0	0	0	0
1	1		0	0	0	1		0	0	0	0	0	0	1
2	2		0	0	1	0		0	0	0	0	0	1	1
3	3		0	0	1	1		0	0	0	0	1	1	1
4	4		0	1	0	0		0	0	0	1	1	1	1
5	5		0	1	0	1		0	0	1	1	1	1	1
6	6		0	1	1	0		0	1	1	1	1	1	1
7	7		0	1	1	1		1	1	1	1	1	1	1
8	6		1	0	0	0		0	1	1	1	1	1	1
9	5		1	0	0	1		0	0	1	1	1	1	1
10	4		1	0	1	0		0	0	0	1	1	1	1
11	3		1	0	1	1		0	0	0	0	1	1	1
12	2		1	1	0	0		0	0	0	0	0	1	1
13	1		1	1	0	1		0	0	0	0	0	0	1

Table 5. Channel 1 ($m_1 = 7$) Waveform Truth Table

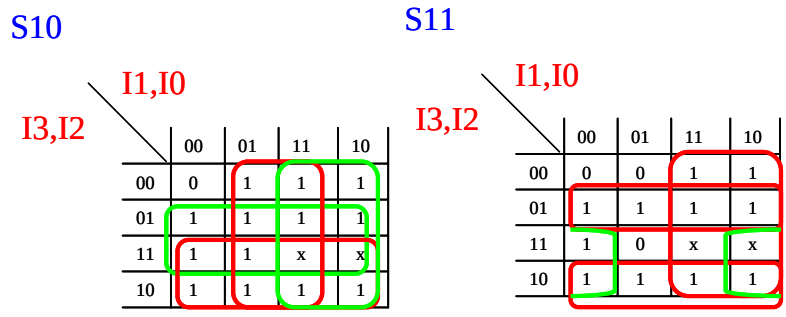


Figure 16. Karnaugh maps for Channel 1 ($m_1 = 7$)Thermometer code, bits S10 and S11

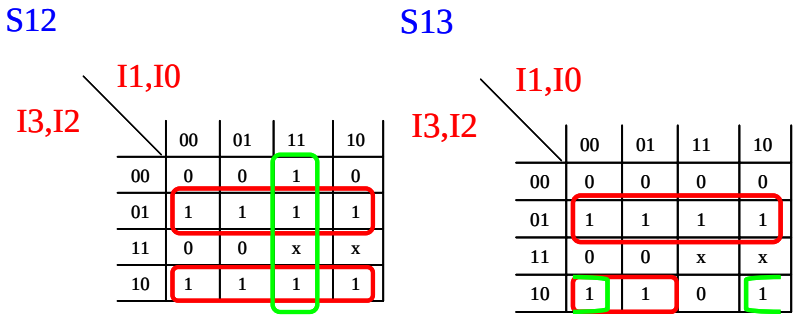


Figure 17. Karnaugh maps for Channel 1 ($m_1 = 7$)Thermometer code, bits S12 and S13

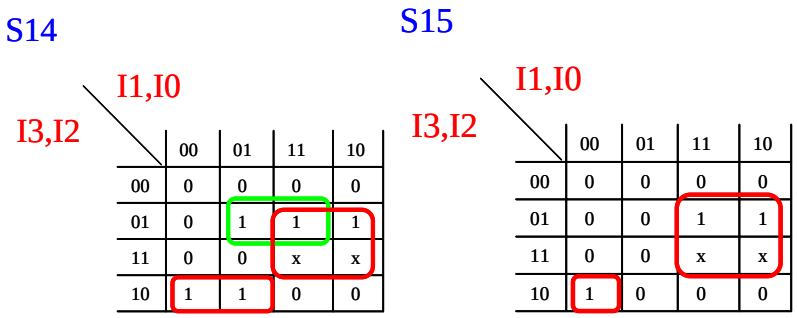


Figure 18. Karnaugh maps for Channel 1 ($m_1 = 7$) Thermometer code, bits S14 and S15

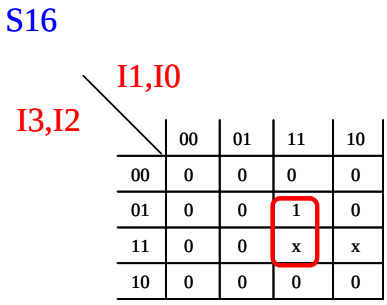


Figure 19. Karnaugh map for Channel 1 ($m_1 = 7$) Thermometer code, bit S16

$$
\begin{aligned}
s_{10} &= I_3 + I_2 + I_1 + I_0 = \overline{\overline{I_3 + I_2 + I_1 + I_0}} \\
s_{11} &= I_1 + \overline{I_3} I_2 + I_3 \overline{I_2} + I_3 \overline{I_0} = \overline{\overline{I_1}\left(\overline{\overline{I_3} I_2}\right)\left(\overline{I_3 \overline{I_2}}\right)\left(\overline{I_3 \overline{I_0}}\right)} \\
s_{12} &= \overline{I_3} I_2 + I_3 \overline{I_2} + I_1 I_0 = \overline{\left(\overline{\overline{I_3} I_2}\right)\left(\overline{I_3 \overline{I_2}}\right)\left(\overline{I_1 I_0}\right)} \\
s_{13} &= \overline{I_3} I_2 + I_3 \overline{I_2 I_1} + I_3 \overline{I_2 I_0} = \overline{\left(\overline{\overline{I_3} I_2}\right)\left(\overline{I_3 \overline{I_2 I_1}}\right)\left(\overline{I_3 \overline{I_2 I_0}}\right)} \\
s_{14} &= I_2 I_1 + \overline{I_3} I_2 I_0 + I_3 \overline{I_2 I_1} = \overline{\left(\overline{I_2 I_1}\right)\left(\overline{\overline{I_3} I_2 I_0}\right)\left(\overline{I_3 \overline{I_2 I_1}}\right)} \\
s_{15} &= I_2 I_1 + I_3 \overline{I_2 I_1 I_0} = \overline{\left(\overline{I_2 I_1}\right)\left(\overline{I_3 \overline{I_2 I_1 I_0}}\right)} \\
s_{16} &= I_2 I_1 I_0 = \overline{\overline{I_2} + \overline{I_1} + \overline{I_0}}
\end{aligned}
\tag{25}
$$

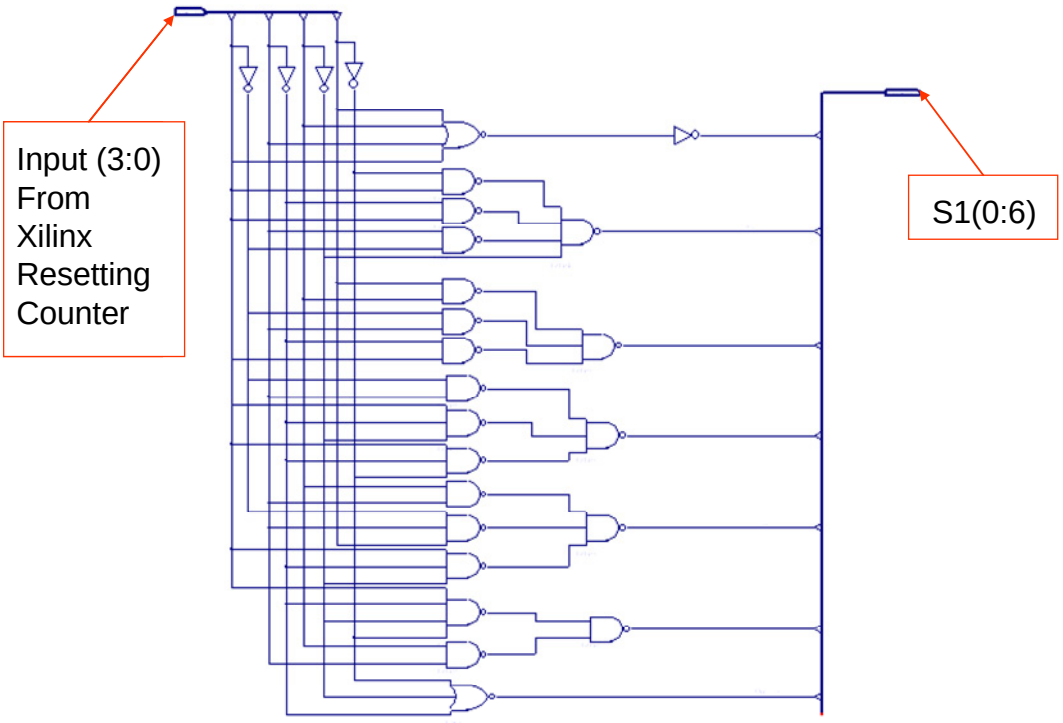


Figure 20. Thermometer Code 7-Bit Generator

Table 6 shows a truth table for channel 2 ($m_2 = 8$). The information in Table 6 is then put into Karnaugh map for bits S20 thru S27, as seen in Figures 21, 22, 23, and 24. These Karnaugh maps produce the least sum equations shown in Equation(26). These

equations were then utilized to implement the channel 2 ($m_2 = 8$) thermometer code generator in Xilinx, as shown in Figure 25.

Decimal	Therm		I23	I22	I21	I20		S27	S26	S25	S24	S23	S22	S21	S20	
0			0	0	0	0		0	0	0	0	0	0	0	0	
1			0	0	0	1		0	0	0	0	0	0	0	1	
2			0	0	1	0		0	0	0	0	0	0	1	1	
3			0	0	1	1		0	0	0	0	0	1	1	1	
4			0	1	0	0		0	0	0	0	1	1	1	1	
5			0	1	0	1		0	0	0	1	1	1	1	1	
6			0	1	1	0		0	0	1	1	1	1	1	1	
7			0	1	1	1		0	1	1	1	1	1	1	1	
8			1	0	0	0		1	1	1	1	1	1	1	1	
9			1	0	0	1		0	1	1	1	1	1	1	1	
10			1	0	1	0		0	0	1	1	1	1	1	1	
11			1	0	1	1		0	0	0	1	1	1	1	1	
12			1	1	0	0		0	0	0	0	1	1	1	1	
13			1	1	0	1		0	0	0	0	0	1	1	1	
14			1	1	1	0		0	0	0	0	0	0	1	1	
15			1	1	1	1		0	0	0	0	0	0	0	1	

Table 6. Channel 2 ($m_2 = 8$) Waveform Truth Table

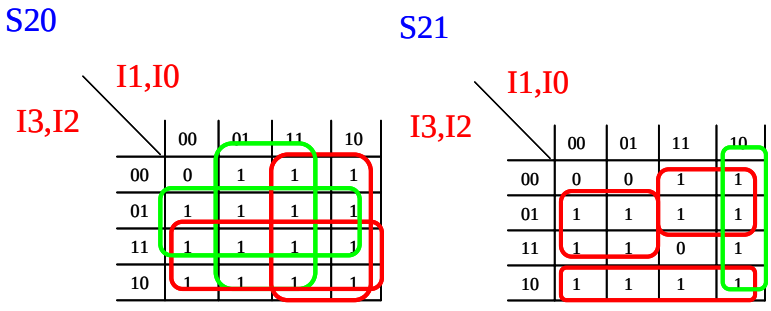
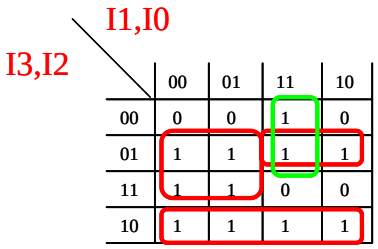


Figure 21. Karnaugh maps for Channel 2 ($m_2 = 8$) Thermometer code, bits S20 and S21

S22



S23

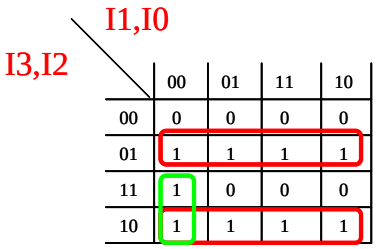
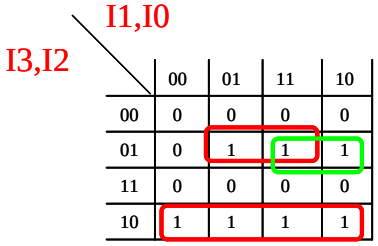


Figure 22. Karnaugh maps for Channel 2 ($m_2 = 8$) Thermometer code, bits S22 and S23

S24



S25

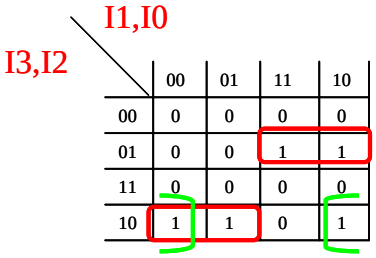
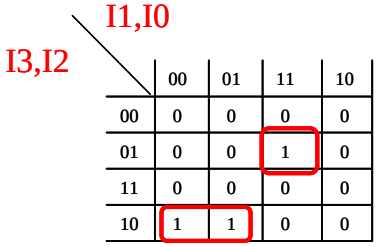


Figure 23. Karnaugh maps for Channel 2 ($m_2 = 8$) Thermometer code, bits S24 and S25

S26



S27

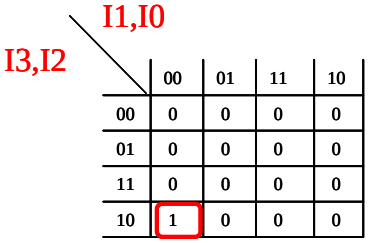


Figure 24. Karnaugh maps for Channel 2 ($m_2 = 8$) Thermometer code, bits S26 and S27

$$
\begin{aligned}
s_{20} &= I_3 + I_2 + I_1 + I_0 = \overline{\overline{I_3 + I_2 + I_1 + I_0}} \\
s_{21} &= \overline{I}_3 I_2 + I_3 \overline{I}_1 + \overline{I}_2 I_1 + I_1 \overline{I}_0 = \overline{\left(\overline{\overline{I}_3 I_2}\right)\left(\overline{I_3 \overline{I}_1}\right)\left(\overline{\overline{I}_2 I_1}\right)\left(\overline{I_1 \overline{I}_0}\right)} \\
s_{22} &= \overline{I}_3 I_2 + I_3 \overline{I}_0 + \overline{I}_3 I_1 I_0 + I_2 \overline{I}_1 = \overline{\left(\overline{\overline{I}_3 I_2}\right)\left(\overline{I_3 \overline{I}_0}\right)\left(\overline{\overline{I}_3 I_1 I_0}\right)\left(\overline{I_2 \overline{I}_1}\right)} \\
s_{23} &= \overline{I}_3 I_2 + I_3 \overline{I}_2 + I_3 \overline{I}_2 \overline{I}_0 = \overline{\left(\overline{\overline{I}_3 I_2}\right)\left(\overline{I_3 \overline{I}_2}\right)\left(\overline{I_3 \overline{I}_2 \overline{I}_0}\right)} \\
s_{24} &= I_3 \overline{I}_2 + \overline{I}_3 I_2 I_0 + \overline{I}_3 I_2 I_1 = \overline{\left(\overline{I_3 \overline{I}_2}\right)\left(\overline{\overline{I}_3 I_2 I_0}\right)\left(\overline{\overline{I}_3 I_2 I_1}\right)} \\
s_{25} &= \overline{I}_3 I_2 I_1 + I_3 \overline{I}_2 \overline{I}_1 + I_3 \overline{I}_2 \overline{I}_0 = \overline{\left(\overline{\overline{I}_3 I_2 I_1}\right)\left(\overline{I_3 \overline{I}_2 \overline{I}_1}\right)\left(\overline{I_3 \overline{I}_2 \overline{I}_0}\right)} \\
s_{26} &= I_3 \overline{I}_2 \overline{I}_1 + \overline{I}_3 I_2 I_1 I_0 = \overline{\left(\overline{I_3 \overline{I}_2 \overline{I}_1}\right)\left(\overline{\overline{I}_3 I_2 I_1 I_0}\right)} \\
s_{27} &= I_3 \overline{I}_2 \overline{I}_1 \overline{I}_0 = \overline{I_3 + \overline{I}_2 + \overline{I}_1 + \overline{I}_0}
\end{aligned}
\tag{26}
$$

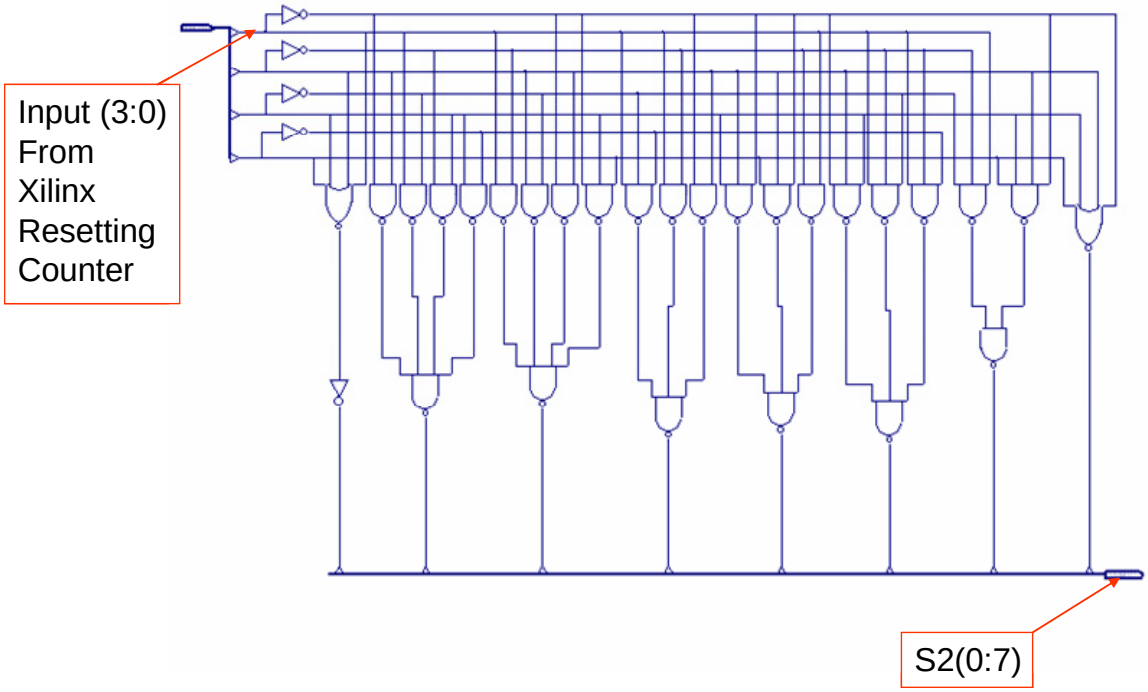


Figure 25. Thermometer Code 8 Bit Generator

Table 7 shows a truth table for channel 3($m_3 = 9$). The information in Table 7 is then put into Karnaugh maps for bits S30 thru S38, as shown in Figures 26, 27, 28, 29, 30, 31, 32, 32, and 34. These Karnaugh maps produce the least sum equations shown in Equation (27). These equations were then utilized to implement the channel 3 thermometer code generator in Xilinx, as shown in Figure 35.

Decimal	Therm		I34	I33	I32	I31	I30		S38	S37	S36	S35	S34	S33	S32	S31	S30
0	0		0	0	0	0	0		0	0	0	0	0	0	0	0	0
1	1		0	0	0	0	1		0	0	0	0	0	0	0	0	1
2	2		0	0	0	1	0		0	0	0	0	0	0	0	1	1
3	3		0	0	0	1	1		0	0	0	0	0	0	1	1	1
4	4		0	0	1	0	0		0	0	0	0	0	1	1	1	1
5	5		0	0	1	0	1		0	0	0	0	1	1	1	1	1
6	6		0	0	1	1	0		0	0	0	1	1	1	1	1	1
7	7		0	0	1	1	1		0	0	1	1	1	1	1	1	1
8	8		0	1	0	0	0		0	1	1	1	1	1	1	1	1
9	9		0	1	0	0	1		1	1	1	1	1	1	1	1	1
10	8		0	1	0	1	0		0	1	1	1	1	1	1	1	1
11	7		0	1	0	1	1		0	0	1	1	1	1	1	1	1
12	6		0	1	1	0	0		0	0	0	1	1	1	1	1	1
13	5		0	1	1	0	1		0	0	0	0	1	1	1	1	1
14	4		0	1	1	1	0		0	0	0	0	0	1	1	1	1
15	3		0	1	1	1	1		0	0	0	0	0	0	1	1	1
16	2		1	0	0	0	0		0	0	0	0	0	0	0	1	1
17	1		1	0	0	0	1		0	0	0	0	0	0	0	0	1

Table 7. Channel 3 ($m_3 = 9$) Waveform Truth Table

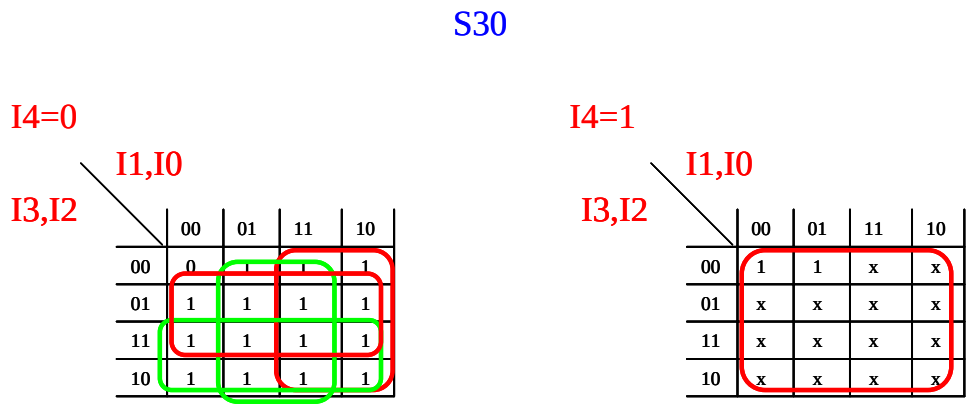


Figure 26. Karnaugh map for Channel 3 ($m_3 = 9$) Thermometer code, bit S30

S31

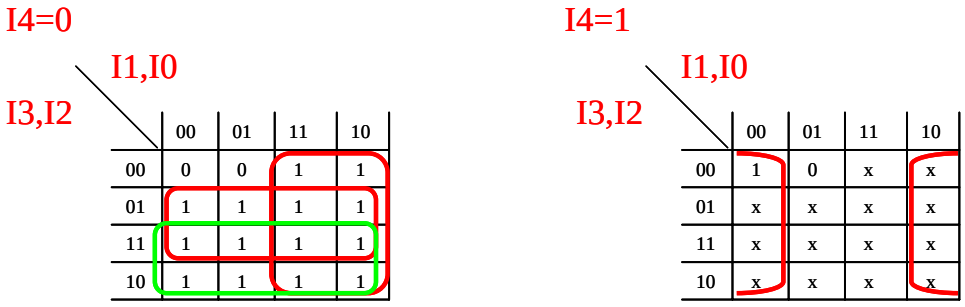


Figure 27. Karnaugh map for Channel 3 ($m_3 = 9$) Thermometer code, bit S31

S32

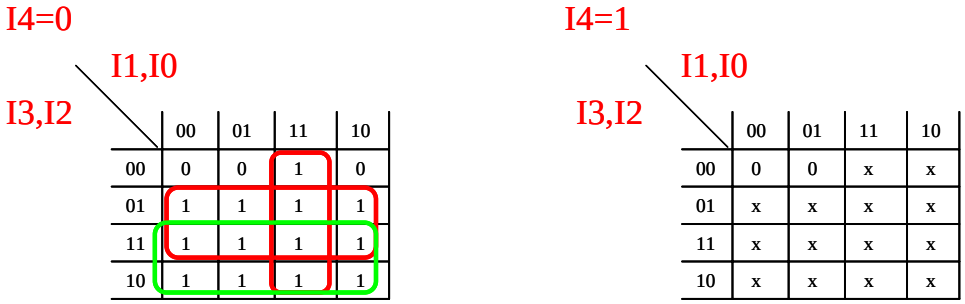


Figure 28. Karnaugh map for Channel 3 ($m_3 = 9$) Thermometer code, bit S32

S33

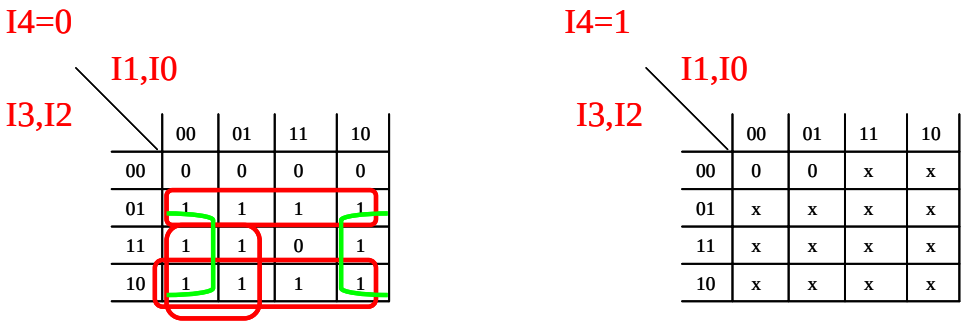


Figure 29. Karnaugh map for Channel 3 ($m_3 = 9$) Thermometer code, bit S33

S34

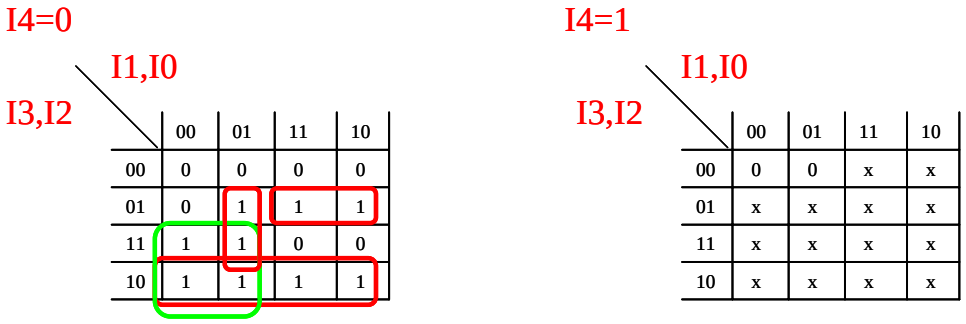


Figure 30. Karnaugh map for Channel 3 ($m_3 = 9$) Thermometer code, bit S34

S35

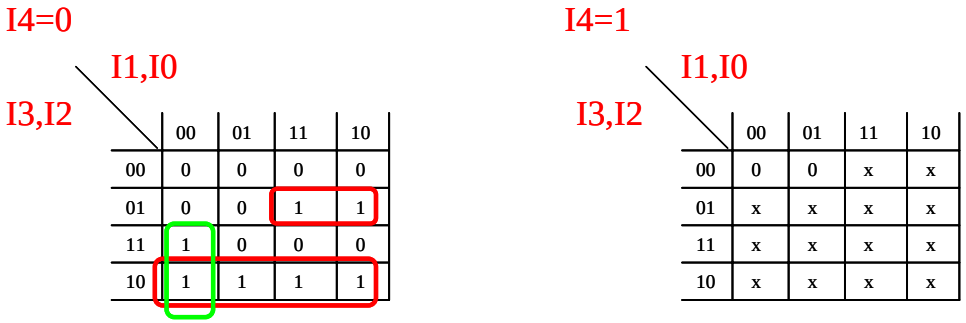


Figure 31. Karnaugh map for Channel 3 ($m_3 = 9$) Thermometer code, bit S35

S36

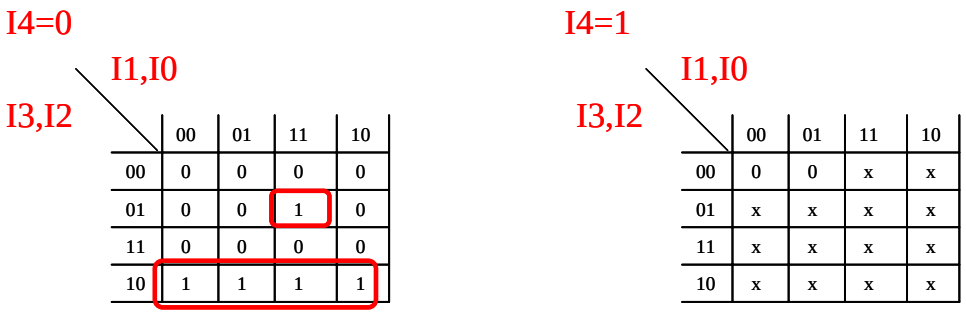


Figure 32. Karnaugh map for Channel 3 ($m_3 = 9$) Thermometer code, bit S36

S37

I4=0

I3,I2 \ I1,I0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	0	0	0
10	1	1	0	1

I4=1

I3,I2 \ I1,I0	00	01	11	10
00	0	0	x	x
01	x	x	x	x
11	x	x	x	x
10	x	x	x	x

Figure 33. Karnaugh map for Channel 3 ($m_3 = 9$) Thermometer code, bit S37

S38

I4=0

I3,I2 \ I1,I0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	0	0	0
10	0	1	0	0

I4=1

I3,I2 \ I1,I0	00	01	11	10
00	0	0	x	x
01	x	x	x	x
11	x	x	x	x
10	x	x	x	x

Figure 34. Karnaugh map for Channel 3 ($m_3 = 9$) Thermometer code, bit S38

$$s_{30} = I_4 + I_3 + I_2 + I_1 + I_0 = \overline{\left(\overline{I_4 + I_3 + I_2}\right)\left(\overline{I_1 + I_0}\right)}$$

$$s_{31} = I_4\overline{I_0} + I_3 + I_2 + I_1 = \overline{\left(\overline{I_4\overline{I_0}}\right)\left(\overline{I_3 + I_2 + I_1}\right)}$$

$$s_{32} = I_3 + I_2 + I_1I_0 = \overline{\left(\overline{I_3 + I_2}\right)\left(\overline{I_1I_0}\right)}$$

$$s_{33} = \overline{I_3}I_2 + I_3\overline{I_1} + I_3\overline{I_2} + I_3\overline{I_0} = \overline{\left(\overline{\overline{I_3}I_2}\right)\left(\overline{I_3\overline{I_1}}\right)\left(\overline{I_3\overline{I_2}}\right)\left(\overline{I_3\overline{I_0}}\right)}$$

$$s_{34} = I_3\overline{I_2} + I_3\overline{I_1} + I_2\overline{I_1}I_0 + \overline{I_3}I_2I_1 = \overline{\left(\overline{I_3\overline{I_2}}\right)\left(\overline{I_3\overline{I_1}}\right)\left(\overline{I_2\overline{I_1}I_0}\right)\left(\overline{\overline{I_3}I_2I_1}\right)} \qquad (27)$$

$$s_{35} = I_3\overline{I_2} + I_3\overline{I_1}\,\overline{I_0} + \overline{I_3}I_2I_1 = \overline{\left(\overline{I_3\overline{I_2}}\right)\left(\overline{I_3\overline{I_1}\,\overline{I_0}}\right)\left(\overline{\overline{I_3}I_2I_1}\right)}$$

$$s_{36} = \overline{I_3}I_2I_1I_0 + I_3\overline{I_2} = \overline{\left(\overline{\overline{I_3}I_2I_1I_0}\right)\left(\overline{I_3\overline{I_2}}\right)}$$

$$s_{37} = I_3\overline{I_2}\,\overline{I_1} + I_3\overline{I_2}\,\overline{I_0} = \overline{\left(\overline{I_3\overline{I_2}\,\overline{I_1}}\right)\left(\overline{I_3\overline{I_2}\,\overline{I_0}}\right)}$$

$$s_{38} = I_3\overline{I_2}\,\overline{I_1}I_0 = \overline{I_3 + \overline{I_2} + \overline{I_1} + I_0}$$

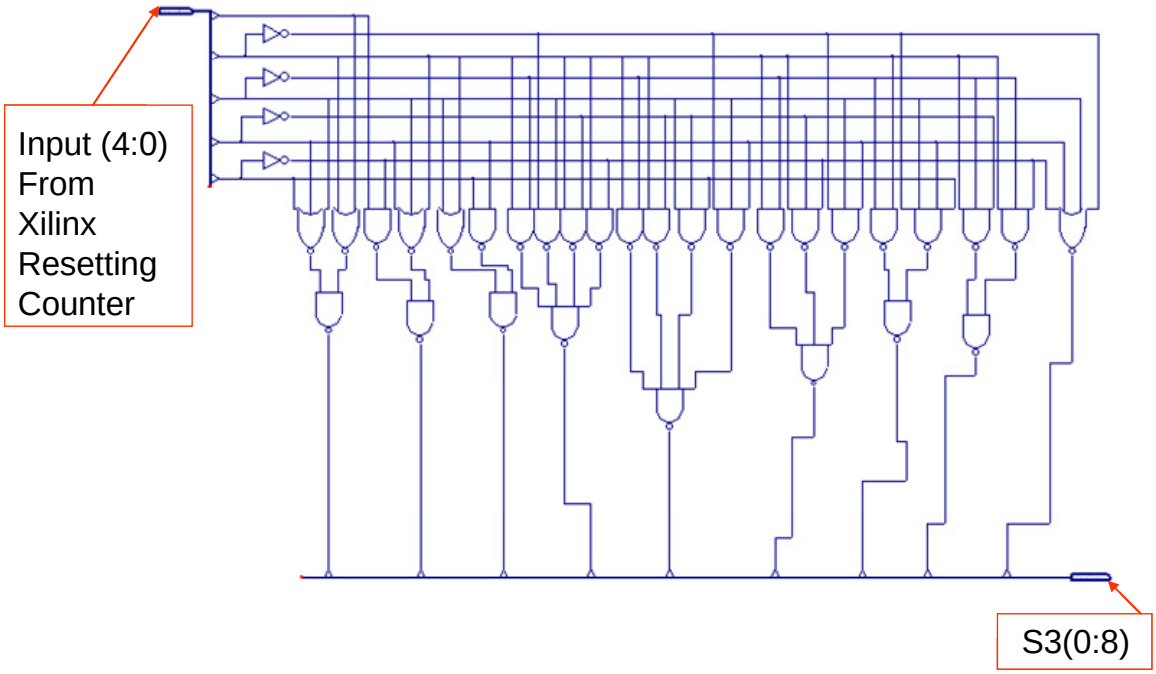


Figure 35. Thermometer Code 9 Bit Generator

These logic blocks will enable the production of a test-bench waveform that will allow the testing of the logic designed in Chapter III.

B. THERMOMETER CODE VERIFICATION

The next step is to verify the thermometer code with Gray code properties was actually generated using the methods described in Section A. Without proper inputs, the functioning of the symmetrical residue thermometer code-to-binary conversion circuit cannot be verified to be operating correctly.

1. MATLAB

Figure 36 is the graph of the outputs of the MATLAB code in Appendix A, sorted by channel. As shown, Channel 1 goes from 0 to 7, Channel 2 goes from 0 to 8, and Channel 3 goes from 0 to 9, showing that each channel has its individual thermometer code properties. Figure 36 also shows the Gray code and shifting properties that are required for the inputs of the system. The figure also shows that in any one sampling period, the signal of only one channel changes.. The left shift of one and two for Channels 1 and 2, respectively, can be seen if we look at Channel 1 being the base. In this case, Channel 2 changes one cycle before, and Channel 3 changes 2 cycles before Channel 1.

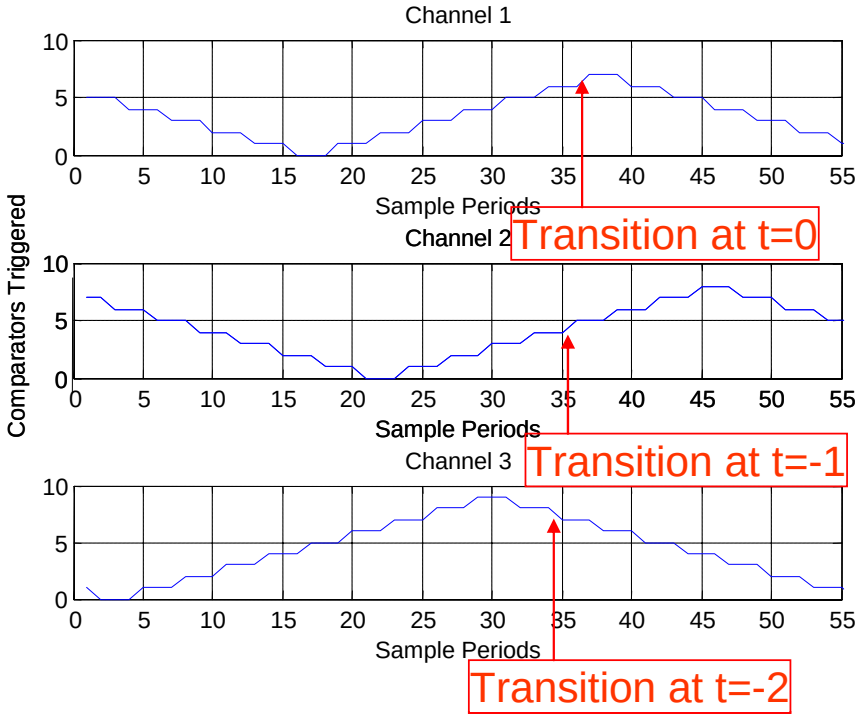


Figure 36. MATLAB Thermometer Code sample

2. Field Programmable Gate Array Schematic Capture

In order to test the RSNS-to-binary thermometer code generating circuits, the generators first needed to be incorporated into the main system as shown in Figure 37. I1, I2, and I3 are the reset counter inputs generated by the Xilinx test-bench waveform. The Verification Outputs, S1, S2, and S3, were included to ensure the generators functioned properly and that the RSNS-to-binary circuit receives the proper signals. Once the system is wired together, a test-bench waveform with the properties discussed above for testing the system and the thermometer code generators was executed.

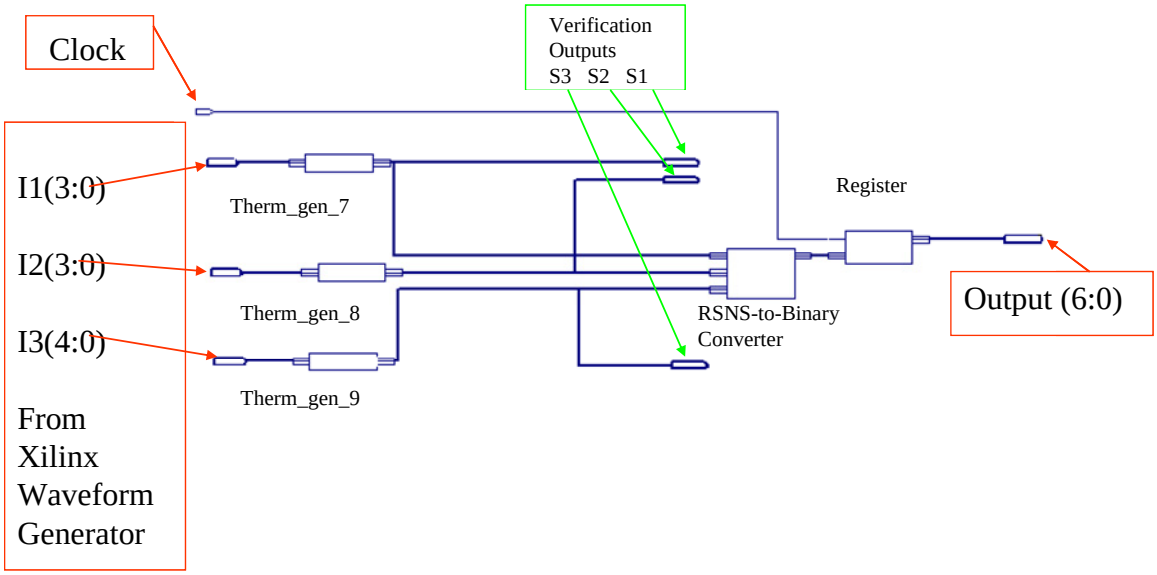


Figure 37. Hierarchal Schematic of Additional testing circuits.

The result of executing the test-bench waveform through the thermometer code generators is shown in Figure 38. I1, I2, and I3 are the decimal equivalents of the binary inputs, being executed by the test-bench waveform for channels 1, 2, and 3, respectively. S1, S2, and S3 are the binary thermometer outputs resulting from the Inputs I1, I2, and I3 respectively. S1, S2, and S3 are all shown with the LSB being to the left most bit and the MSB to the right.

The Gray code property needed for the RSNS-to-binary system is also verified by looking at Figure 38. Remembering that channel 2 is shifted one left and channel 3 is shifted two left, we see channel 2 changing one sample time prior to channel 1 and channel 3 changing two time samples prior to channel 1. The further Gray code property

of only one bit changing can be seen by looking at the bits in S1, S2, and S3 and seeing that only one bit changes at each time sampling.

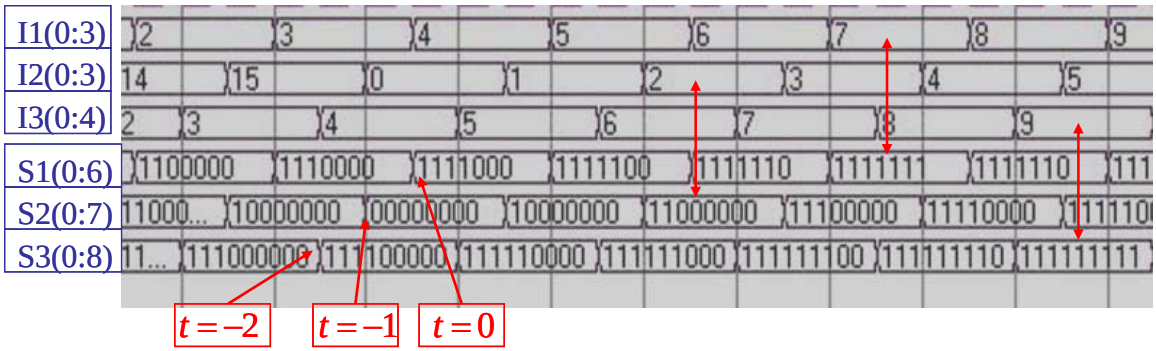


Figure 38. Test-Bench Inputs and Outputs from Thermometer Code Generators

This chapter verified that both the MATLAB and Xilinx programs were generating an appropriate test signal that met all the requirements for the RSNS-to-binary converter. The next chapter will use these verified signals to test and verify the functioning of the symmetrical residue-to-binary algorithm.

THIS PAGE INTENTIONALLY LEFT BLANK

V. VERIFICATION OF TEST RESULTS

A. MATLAB CODE

Chapter IV showed that the MATLAB code in Appendix A generates a thermometer code for the three channels of the RSNS-to-binary circuit. It also showed that the MATLAB code was properly shifted and possessed Gray code properties as needed by the system.

When the MATLAB code in Appendix A is executed, the matrices generated represent the input bits to the MATLAB code in Appendix B. At the conclusion of the execution of both sets of code, Figure 37 is produced.

The desired output of the RSNS-to-binary system is a linear region of length 126. Figure 39 shows the output for the test vector passed to Appendix B. There is a linear region of 126, demonstrating that the desired output for the RSNS-to-binary system was accomplished with the MATLAB code.

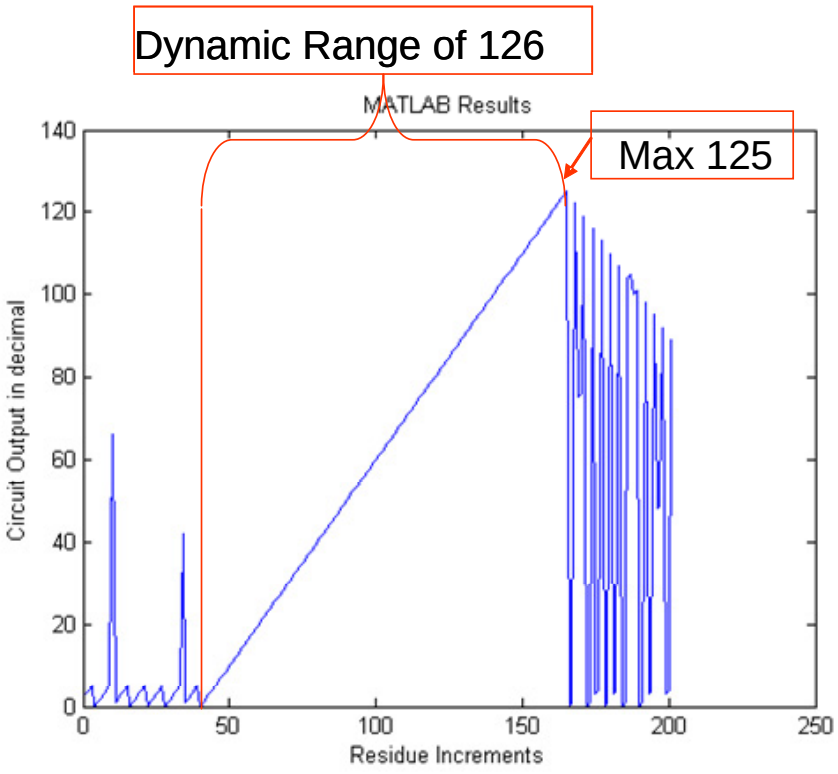


Figure 39. Graph of Output of MATLAB Logic code.

The other critical portion that was found with the MATLAB code was the beginning vector for the dynamic range. The vector that produces the zero at the beginning of the dynamic range of the system is [6, 6, 6]. That is, the thermometer codes from Channel 1, Channel 2, and Channel 3 are [1111110], [11111100], and [111111000] respectively. There are 2^3 possible occurrences of this vector in the three channel vector of inputs as each of the three channels can be either increasing or decreasing. Therefore, the beginning vector description has to be more specific. It was found that the beginning vector is [6, 6, 6], with Channel 1 decreasing, Channel 2 increasing, and Channel 3 decreasing.

B. FPGA DESIGN

Chapter IV showed that the thermometer code generators for the Xilinx schematic RSNS-to-binary circuit functioned properly. Utilizing the starting vector found after the execution of the MATLAB code, the output shown in Figure 40 is achieved from the Xilinx RSNS-to-binary circuit. The starting point of the system is shown and each transition of the decimal output coincides with the positive edge of the clock due to the use of an output register made with positive edge triggered D flip flops. The output depends on the twenty-four inputs that are active at the positive edge of the clock signal for the three channels, i.e., as shown in Figure 38, the transition from 5 to 0 depends on Channel 1, 2, and 3 having values of [1111110], [11111100], and [111111000] respectively.

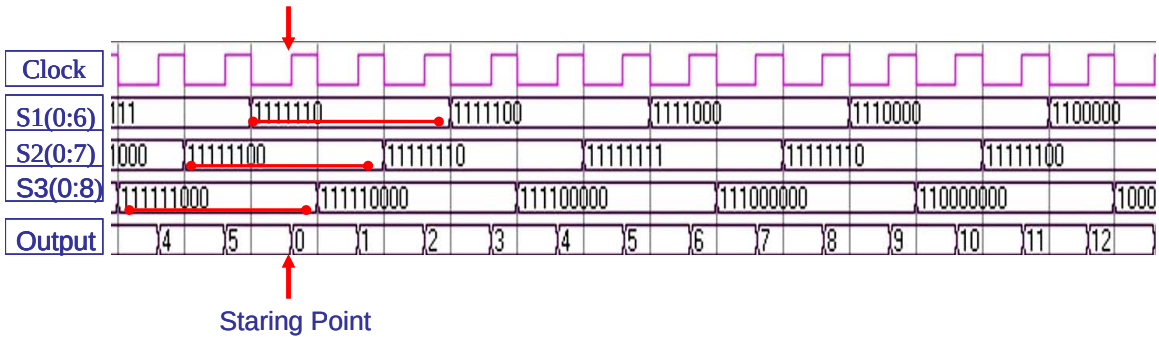


Figure 40. RSNS-to-Binary Waveform Outputs for Dynamic Range Values 0:12

Figures 41 thru 50 show the remaining 113 outputs of the dynamic range of the system. Therefore, we know the RSNS-to-binary logic design functions correctly, as designed.

Figure 41. RSNS-to-Binary Waveform Outputs for Dynamic Range Values 12:24

Figure 42. RSNS-to-Binary Waveform Outputs for Dynamic Range Values 24:36

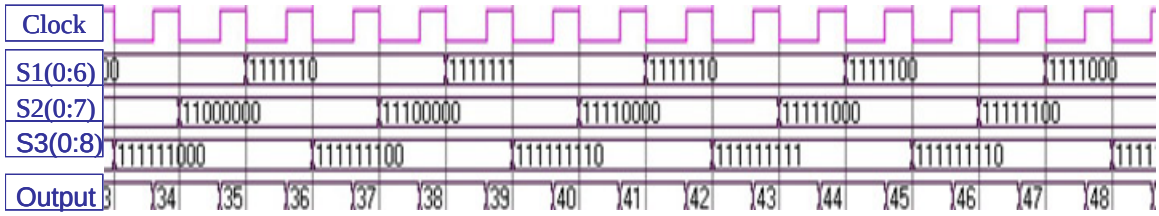


Figure 43. RSNS-to-Binary Waveform Outputs for Dynamic Range Values 36:48

Figure 44. RSNS-to-Binary Waveform Outputs for Dynamic Range Values 48:60

Figure 45. RSNS-to-Binary Waveform Outputs for Dynamic Range Values 60:72

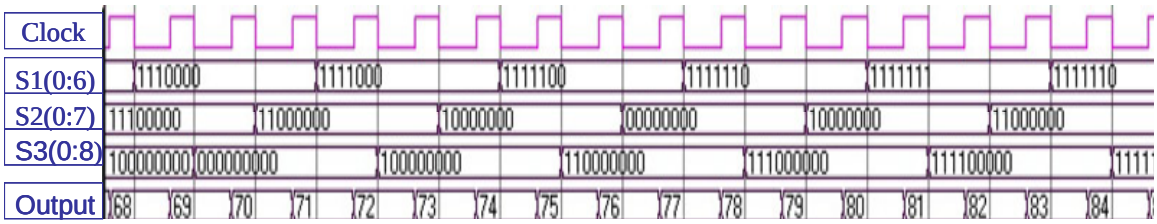


Figure 46. RSNS-to-Binary Waveform Outputs for Dynamic Range Values 72:84

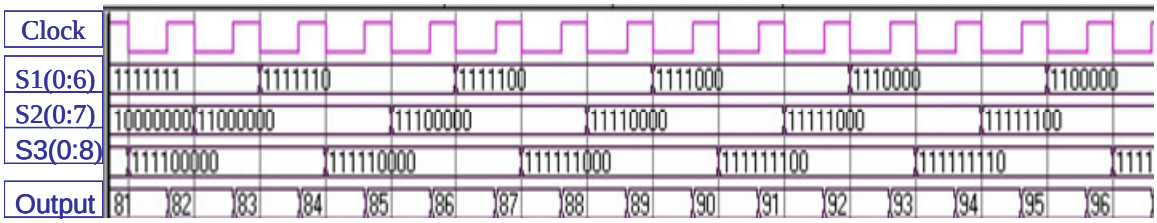


Figure 47. RSNS-to-Binary Waveform Outputs for Dynamic Range Values 84:96

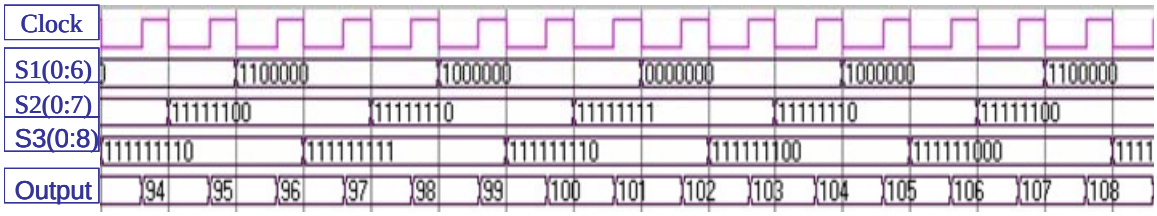


Figure 48. RSNS-to-Binary Waveform Outputs for Dynamic Range Values 96:108

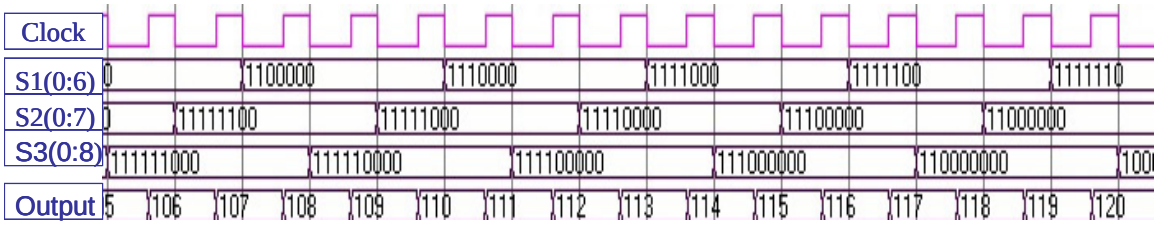


Figure 49. RSNS-to-Binary Waveform Outputs for Dynamic Range Values 108:120

Figure 50. RSNS-to-Binary Waveform Outputs for Dynamic Range Values 120:125

VI. CONCLUSION AND RECOMMENDATIONS FOR FUTURE WORK

A. CONCLUSIONS

The goal of this thesis was to create a logic block that would effectively convert twenty-four RSNS comparator outputs into a useful binary output that had a linear range of operation. Verification of the symmetric residue-to-binary algorithm was accomplished by utilizing two separate software applications.

As shown in Chapter II, the equations for the Position Bit conversion along with the sorting algorithm for the alignment logic, were not difficult to expand. Given the general equations, the more difficult task was to verify functionality of the expanded version.

The MATLAB code generated for this thesis, and shown in Appendix A and B, fully expanded the region of possible combinations of outputs from the analog circuit. This data was then searched and the dynamic range located. Once this range was located, the code was modified to focus only on the functional range of the system. As shown in Chapter V, the output of Appendix B had a linear region of length 126.

Once the equations were verified utilizing the MATLAB code in the appendices, the next step was to create actual logic that could implement the system. The RSNS-to-binary logic created in this thesis did perform as expected and produced the results needed for further use.

The ADC assumed design was not bipolar. Therfore, an actual analog signal would have to be biased so that the mean amplitude would correspond to the center of the operating region of the RSNS-to-binary converter in order for the negative swing of the amplitude to be accurately converted to a digital signal. The theory here is much like that of light emitting diode (LED) analog signal generation, the diode has to have a dc bias or it will not transit the negative amplitudes of the signal.

This RSNS-to-binary system has been demonstrated to work and produces a seven-bit output for the cost of only 24 comparators and with relatively simple logic. The savings in energy and size due to the reduced number of comparators and the relatively

small amount of logic will make this design appealing for low power and small size applications which are becoming more and more sought after. Once implemented with the analog processing portion this design, this system will be ideal for unmanned aerial vehicles (UAVs), both for sensors and flight control, along with many other systems.

B. RECOMMENDATIONS FOR FUTURE RESEARCH

This thesis is only the digital portion of a larger ADC research project that also involved the design of the analog folding circuits. There is still work to be done on this project in several areas.

1. Program FPGA in Order to Test Performance of the Design.

The RSNS-to-binary system has been tested and verified in [1] for the smaller moduli and this thesis verified the conversion for much larger moduli in software simulation only. One possible extension of this research would be to program an FPGA with the data developed in this thesis and perform detailed testing of the design in hardware.

2. Implement the RSNS Digital Processing Portion of the ADC and Test in ASIC Simulator Software.

Currently, the code developed in the Xilinx Project Navigator system is being converted to netlist files for implementation into an application specific integrated circuit (ASIC). There are several areas where work on this project is continuing along this line. The final place and route layout will need to be complete, along with the link to the analog portion of the system

Once the ASIC chip is designed and tested in software, it will need to be fabricated and hardware testing will need to be conducted. Along this line, integration of this system into other research projects at the NPS is already being considered.

APPENDIX A. MATLAB CODE FOR GENERATING THERMOMETER CODE INPUTS

```
%Ross A. Monta
%Created 5 Jan 2004

%Thermometer code establishment.
%This code was written to generate thermometer code for input into the
%Residue-to-binary logic written in file "Logic1.m".

%Moduli 7 shifted 0

clear all

%S10 to S16
for i=1:42
if i>3 & i<43
   S10(i)=1;
else
   S10(i)=0;
end
if i>6 & i<40
   S11(i)=1;
else
   S11(i)=0;
end
if i>9 & i<37
   S12(i)=1;
else
   S12(i)=0;
end
if i>12 & i<34
   S13(i)=1;
else
   S13(i)=0;
end
if i>15 & i<31
   S14(i)=1;
else
   S14(i)=0;
end
 if i>18 & i<28
   S15(i)=1;
else
   S15(i)=0;
end
if i>21 & i<25
   S16(i)=1;
else
   S16(i)=0;
end
```

```
end   %end of moduli 7 for loop
%The following vector was created for ease of visual verification of above
%code.
S1=S10+S11+S12+S13+S14+S15+S16;

%Moduli 8 shifted left 1

%S20 to S27
for i=1:48
if i>2 & i<48
   S20(i)=1;
else
   S20(i)=0;
end
if i>5 & i<45
   S21(i)=1;
else
   S21(i)=0;
end
if i>8 & i<42
   S22(i)=1;
else
   S22(i)=0;
end
if i>11 & i<39
   S23(i)=1;
else
   S23(i)=0;
end
if i>14 & i<36
   S24(i)=1;
else
   S24(i)=0;
end
 if i>17 & i<33
   S25(i)=1;
else
   S25(i)=0;
end
if i>20 & i<30
   S26(i)=1;
else
   S26(i)=0;
end
if i>23 & i<27
   S27(i)=1;
else
   S27(i)=0;
end
end   %end of moduli 8 for loop

%The following vector was created for ease of visual verification of above
%code.
S2=S20+S21+S22+S23+S24+S25+S26+S27;
```

```
%Moduli 9 shifted 2 left

%S30 to S38
for i=1:54
if i>1 & i<53
   S30(i)=1;
else
   S30(i)=0;
end
if i>4 & i<50
   S31(i)=1;
else
   S31(i)=0;
end
if i>7 & i<47
   S32(i)=1;
else
   S32(i)=0;
end
if i>10 & i<44
   S33(i)=1;
else
   S33(i)=0;
end
if i>13 & i<41
   S34(i)=1;
else
   S34(i)=0;
end
 if i>16 & i<38
   S35(i)=1;
else
   S35(i)=0;
end
if i>19 & i<35
   S36(i)=1;
else
   S36(i)=0;
end
if i>22 & i<32
   S37(i)=1;
else
   S37(i)=0;
end
if  i>25 & i<29
   S38(i)=1;
else
   S38(i)=0;
end

end   %end of moduli 9 for loop

%The following vector was created for ease of visual verification of above
%code.
S3=S30+S31+S32+S33+S34+S35+S36+S37+S38;
```

```
% Vector expansion to get the range of numbers for a right shifted
% sequence.  Starting with the Mod 8 shifted 1 left, and Mod 9 shifted 2
% left. For i = 1  the line up will have all elements equal to zero.

%Mod 7 expansion
T10=S10;
T11=S11;
T12=S12;
T13=S13;
T14=S14;
T15=S15;
T16=S16;
for i=1:2591
   T10=[T10,S10];
   T11=[T11,S11];
   T12=[T12,S12];
   T13=[T13,S13];
   T14=[T14,S14];
   T15=[T15,S15];
   T16=[T16,S16];
end
%The following vector was created for ease of visual verification of above
%code.
T1=(T10+T11+T12+T13+T14+T15+T16);

%Mod 8 expansion
T20=S20;
T21=S21;
T22=S22;
T23=S23;
T24=S24;
T25=S25;
T26=S26;
T27=S27;
for i = 1:2267
   T20=[T20,S20];
   T21=[T21,S21];
   T22=[T22,S22];
   T23=[T23,S23];
   T24=[T24,S24];
   T25=[T25,S25];
   T26=[T26,S26];
   T27=[T27,S27];
end
%The following vector was created for ease of visual verification of above
%code.
T2=(T20+T21+T22+T23+T24+T25+T26+T27);

%Mod 9 Expansion

T30=S30;
T31=S31;
T32=S32;
```

```
T33=S33;
T34=S34;
T35=S35;
T36=S36;
T37=S37;
T38=S38;
for i= 1:2015
   T30=[T30,S30];
   T31=[T31,S31];
   T32=[T32,S32];
   T33=[T33,S33];
   T34=[T34,S34];
   T35=[T35,S35];
   T36=[T36,S36];
   T37=[T37,S37];
   T38=[T38,S38];
end
%The following vector was created for ease of visual verification of above
%code.
T3=(T30+T31+T32+T33+T34+T35+T36+T37+T38);

%The following vector was created for ease of visual verification of the
%required shifts of the moduli.
T=[T1;T2;T3];


%Set Input vector size
%The range of the entire system is to large to effectively run through the
%"Logic1.m" file, therefore, this next section was used to parcel the
%output of "Therm.m" into smaller more usable sections.
%The section of T sent to "Logic1.m" is the correct portion for the dynamic
%range of this system.
%Values were picked of i to represent dynamic range or larger for data to
%input to "Logic1.m" function.

count=0;
for i=700:900
   count=count+1;

   IT1(count)=T1(i);
   IT2(count)=T2(i);
   IT3(count)=T3(i);

   I10(count)=T10(i);
   I11(count)=T11(i);
   I12(count)=T12(i);
   I13(count)=T13(i);
   I14(count)=T14(i);
   I15(count)=T15(i);
   I16(count)=T16(i);

   I20(count)=T20(i);
   I21(count)=T21(i);
   I22(count)=T22(i);
```

```
    I23(count)=T23(i);
    I24(count)=T24(i);
    I25(count)=T25(i);
    I26(count)=T26(i);
    I27(count)=T27(i);

    I30(count)=T30(i);
    I31(count)=T31(i);
    I32(count)=T32(i);
    I33(count)=T33(i);
    I34(count)=T34(i);
    I35(count)=T35(i);
    I36(count)=T36(i);
    I37(count)=T37(i);
    I38(count)=T38(i);
end
%The following vector was created for ease of visual verification of above
%code.
IT=[IT1;IT2;IT3];
```

APPENDIX B. CODE FOR IMPLEMENTATION OF RESIDUE-TO-BINARY IN MATLAB

```
%Ross A. Monta
%Created 5 Jan 2005
%Program to evaluate logic equations for 3 Level Folding ADC using Moduli,
%7,8,9 respectively.

%S10 to S16, S20 to S27, and S30 to S38 would all be Thermometer
%Code inputs from the analog portion of the system.
%I10 to I16, I20 to I27, and I30 to I38 are all variables passed from the
%execution of "Therm.m" file.

S10=I10;
S11=I11;
S12=I12;
S13=I13;
S14=I14;
S15=I15;
S16=I16;

S20=I20;
S21=I21;
S22=I22;
S23=I23;
S24=I24;
S25=I25;
S26=I26;
S27=I27;

S30=I30;
S31=I31;
S32=I32;
S33=I33;
S34=I34;
S35=I35;
S36=I36;
S37=I37;
S38=I38;

%Channel 1 Moduli 7
%Position Bit Conversion

P10=not(S11);
P11=and(S11,not(S13));
P12=and(S13,not(S15));
P13=S15;
P14=and(S14,not(S16));
P15=and(S12,not(S14));
P16=and(S10,not(S12));

%Channel 2 Moduli 8
```

```
%Position Bit Conversion

P20= not(S21);
P21= and(S21,not(S23));
P22=and(S23,not(S25));
P23=and(S25,not(S27));
P24=S26;
P25=and(S24,not(S26));
P26=and(S22,not(S24));
P27=and(S20,not(S22));

%Channel 3 Moduli 9
%Position Bit Conversion

P30= not(S31);
P31= and(S31,not(S33));
P32=and(S33,not(S35));
P33=and(S35,not(S37));
P34=S37;
P35=and(S36,not(S38));
P36=and(S34,not(S36));
P37=and(S32,not(S34));
P38=and(S30,not(S32));

%Testing Channels for even or odd%%%%%%%%%%%%%%%%%%%%%%

%Channel 1 even ?

E1=or(not(S10),or(and(S11,not(S12)),or(and(S13,not(S14)),and(S15,not(S16)))));

%Channel 2 even ?

E2=or(not(S20),or(and(S21,not(S22)),or(and(S23,not(S24)),or(and(S25,not(S26)),S27))));

%Channel 3 even ?

E3=or(not(S30),or(and(S31,not(S32)),or(and(S33,not(S34)),or(and(S35,not(S36)),and(S37,not(S
38))))));

%Inversion Controls

E=[E1;E2;E3];


MRSS0not= xor(E3,E1);
MRSS1=xor(E3,E2);
MRSS2=xor(E2,E1);

MRS=[not(MRSS0not);MRSS1;MRSS2];

%Signal Inversions%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%Channel 1 is never inverted.  For sake of consistent variables this
%conversion is included.

P10a=P10;
P11a=P11;
P12a=P12;
P13a=P13;
P14a=P14;
P15a=P15;
P16a=P16;

%Channel 2

P20a=or(and(P20,not(MRSS2)),and(P27,MRSS2));
P21a=or(and(P21,not(MRSS2)),and(P26,MRSS2));
P22a=or(and(P22,not(MRSS2)),and(P25,MRSS2));
P23a=or(and(P23,not(MRSS2)),and(P24,MRSS2));
P24a=or(and(P24,not(MRSS2)),and(P23,MRSS2));
P25a=or(and(P25,not(MRSS2)),and(P22,MRSS2));
P26a=or(and(P26,not(MRSS2)),and(P21,MRSS2));
P27a=or(and(P27,not(MRSS2)),and(P20,MRSS2));

%Channel 3

P30a=or(and(P30,not(MRSS0not)),and(P38,MRSS0not));
P31a=or(and(P31,not(MRSS0not)),and(P37,MRSS0not));
P32a=or(and(P32,not(MRSS0not)),and(P36,MRSS0not));
P33a=or(and(P33,not(MRSS0not)),and(P35,MRSS0not));
P34a=P34;
P35a=or(and(P35,not(MRSS0not)),and(P33,MRSS0not));
P36a=or(and(P36,not(MRSS0not)),and(P32,MRSS0not));
P37a=or(and(P37,not(MRSS0not)),and(P31,MRSS0not));
P38a=or(and(P38,not(MRSS0not)),and(P30,MRSS0not));

%Nand Logic%%%%%%%%%%%%%%%%%%
%This logic acts as a filter to use the dynamic range of this system

n0=(and(P14a,and(P23a,P36a)));
n1=(and(P15a,and(P24a,P37a)));
n2=(and(P16a,and(P25a,P38a)));
n3=(and(P10a,and(P26a,P30a)));
n4=(and(P11a,and(P27a,P31a)));
n5=(and(P12a,and(P20a,P32a)));
n6=(and(P13a,and(P21a,P33a)));
n7=(and(P14a,and(P22a,P34a)));
n8=(and(P15a,and(P23a,P35a)));
n9=(and(P16a,and(P24a,P36a)));
n10=(and(P10a,and(P25a,P37a)));
n11=(and(P11a,and(P26a,P38a)));
n12=(and(P12a,and(P27a,P30a)));
n13=(and(P13a,and(P20a,P31a)));
n14=(and(P14a,and(P21a,P32a)));
n15=(and(P15a,and(P22a,P33a)));
n16=(and(P16a,and(P23a,P34a)));
```

```
n17=(and(P10a,and(P24a,P35a)));
n18=(and(P11a,and(P25a,P36a)));
n19=(and(P12a,and(P26a,P37a)));
n20=(and(P13a,and(P27a,P38a)));

N=[n0;n1;n2;n3;n4;n5;n6;n7;n8;n9;n10;n11;n12;n13;n14;n15;n16;n17;n18;n19;n20];


% 21 bit to 5 bit Encoder%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

g0=(or(n1,or(n3,or(n5,or(n7,or(n9,or(n11,or(n13,or(n15,or(n17,n19)))))))));
g1=(or(n2,or(n3,or(n6,or(n7,or(n10,or(n11,or(n14,or(n15,or(n18,n19)))))))));
g2=(or(n4,or(n5,or(n6,or(n7,or(n12,or(n13,or(n14,or(n15,n20)))))))));
g3=(or(n8,or(n9,or(n10,or(n11,or(n12,or(n13,or(n14,n15)))))));
g4=(or(n16,or(n17,or(n18,or(n19,n20)))));

%7 bit adder%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

h0=xor(MRSS2,xor(MRSS0not,not(E1)));
%c0=carry out of h0
c0=or(and(MRSS2,MRSS0not),or(and(MRSS2,not(E1)),and(MRSS0not,not(E1))));

h1=xor(c0,xor(not(E1),g0));
c1=or(and(c0,g0),or(and(c0,not(E1)),and(g0,not(E1))));

h2=xor(c1,xor(g0,g1));
c2=or(and(c1,g0),or(and(c1,g1),and(g0,g1)));

h3=xor(c2,xor(g2,g1));
c3=or(and(c2,g2),or(and(c2,g1),and(g2,g1)));

h4=xor(c3,xor(g2,g3));
c4=or(and(c3,g2),or(and(c3,g3),and(g2,g3)));

h5=xor(c4,xor(g4,g3));
c5=or(and(c4,g4),or(and(c4,g3),and(g4,g3)));

h6=xor(g4,c5);

%The following vector was generated to make visual verification of the
%output easier.
decimal=h0+(2*h1)+(4*h2)+(8*h3)+(16*h4)+(32*h5)+(64*h6);

plot(decimal)
title('MATLAB Results')
xlabel('Residue Increments')
ylabel('Circuit Output in decimal')
```

LIST OF REFERENCES

[1] B. L. Luke, "Architecture of an Integrated Microelectronic Warfare System on a Chip and Design of Key Components" Doctoral dissertation, Naval Postgraduate School, Monterey, California, Dec. 2004.

[2] R.Van De Plassche, and P. Baltus, "An 8b 100MHz ADC." *Proceedings of the IEEE 1988 International Solid-State Circuit Conf*. pp. 222-278, 19 Feb. 1988

[3] Z Wang, H. Pan, C. Chang, H. Yu, and M. F. Chang, "A 600 MSPS 8-bit Folding ADC in 0.18um CMOS", 2004 IEEE Symposium on VLSI Circuits Digest of Technical Papers, pp. 424-427

[4] P.E. Pace, D. Styer and I. A. Akin, "A folding ADC Preprocessing Architecture Employing a Robust Symmetrical Number System with Gray-code Properties", *IEEE Transactions on Circuits and Systems-II; Analog and Digital Signal Processing*, vol. 47, pp. 462-467 , May 2000.

[5] M. M. Mano, *Digital Design*, Prentice Hall, New Jersey 2002

[6] J. F. Wakerly, *Digital Design Principles and Practices*, 3rd ed. Prentice Hall, New Jersey 2001

[7] M. J. Choe, B. S. song, and K Bacrania, "An 8-b 100-MSample/s CMOS Piplined Folding ADC", *IEEE Journal of Solid-state Circuits*, vol. 32, pp. 184-194, Feb. 2001

[8] D. Styer and R. E. Pace, "Two-Channel RSNS Dynamic Range", *IEEE Transactions on Circuits and Systems-I*, vol. 49, pp. 395-397, March 2002

[9] H. Pan and A. A. Abidi, "Signal folding in A/D Converters, *IEEE Transactions on Circuits and Systems-I*, vol. 51, pp. 3-14, Jan. 2004

[10] Z. Wang, H. Pan, C. Chang, H. Yu, and M. F. Chang, "a 600 MSPS 8-bit Folding ADC in 0.18um CMOS", *IEEE Symposium on VLSI Circuits Digest of Technical Papers*, pp. 424-427

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Marine Corps Representative
   Naval Postgraduate School
   Monterey, California

4. Director, Training and Education, MCCDC, Code C46
   Quantico, Virginia

5. Director, Marine Corps research Center, MCCDC, Code C40RC
   Quantico, Virginia

6. Marine Corps Tactical systems Support Activity (Attn: Operations Officer)
   Camp Pendleton, California

7. Prof. Phillip Pace
   Naval Postgraduate School
   Monterey, California

8. Prof. Douglas Fouts
   Naval Postgraduate School
   Monterey, California

9. CDR Brian Luke
   NIOC Suitland
   Suitland, Maryland

10. Dr. Peter Craig
    Office of Naval Research
    Arlington, Virginia