12-2017

Design of a Single Precision Floating Point Divider and Multiplier with Pipelined Architecture

Mayuresh Vijay Keni
mvk3029@rit.edu

Design Of A Single Precision Floating Point Divider And Multiplier With Pipelined Architecture

by
Mayuresh Vijay Keni

Graduate Paper

Submitted in partial fulfillment
of the requirements for the degree of
Master of Science
in Electrical Engineering

Approved by:

Mr. Mark A. Indovina, Lecturer
Graduate Research Advisor, Department of Electrical and Microelectronic Engineering

---

Dr. Sohail A. Dianat, Professor
Department Head, Department of Electrical and Microelectronic Engineering

Department of Electrical and Microelectronic Engineering
Kate Gleason College of Engineering
Rochester Institute of Technology
Rochester, New York
December 2017

To my Mom, Sister and Friends, for all of their endless love, support, and encouragement throughout my career at Rochester Institute of Technology.

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this paper are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other University. This paper is the result of my own work and includes nothing which is the outcome of work done in collaboration, except where specifically indicated in the text.

Mayuresh Vijay Keni

December 2017

Acknowledgements

I would like to thank my adviser, professor, and mentor, Mark A. Indovina, for all of his guidance and feedback throughout the entirety of this project. He is the reason for my love of digital hardware design and drove me to pursue it as a career path. He has been a tremendous help and a true friend during my graduate career at RIT.

Another professor I would like to thank is Dr. Dorin Patru. He cleared my computer architecture knowledge and always provided helpful feedback for my random questions.

I would also like to thank Akshay, Mohit, Surya and Saurabhi for guiding me and provide great company throughout many sleepless nights spent on this project.

Finally, I would like to thank my best friends, Vishal, Aniket, Prashant and Shishir, without their advice, love and guidance my career at RIT wouldn't have been possible.

Abstract

High speed computation is the need of today's generation of Processors. To accomplish this major task, many functions are implemented inside the hardware of the processor rather than having software computing the same task. Majority of the operations which the processor executes are Arithmetic operations which are widely used in many applications that require heavy mathematical operations such as scientific calculations, image and signal processing. Especially in the field of signal processing, multiplication division operation is widely used in many applications. The major issue with these operations in hardware is that many iteration's are required which results in slow operation while fast algorithms require complex computations within each cycle. The result of a Division operation results in a either in Quotient and Remainder or a Floating point number which is the major reason to make it more complex than Multiplication operation. The work described in this paper includes design and verification of a floating point divider and multiplier. The inputs of both the Multiplier and Divider and also the output are designed using the single precision IEEE Standard for floating point numbers.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The most commonly used floating point formats are Single precision and Double precision floating point format. Both of these formats are defined by the IEEE 754-2008 standard where single precision format consists of a 32 bit binary number and the double precision format consists of a 64 bit binary number. Both of these numbers are divided into different sections which together represents a floating point value. In this paper, a 32 bit Single Precision Floating Point Divider and Multiplier is designed using pipelined architecture. A 32 bit floating point value represented using single precision format is divided into 3 sections. The first bit in the single precision format is the Sign bit, the next 8 bits represents the Exponent, and the remaining 23 bits represents the Mantissa. The base of the exponent bit is 2 and thus the range of the exponent is -127 to 128.

Both the Divider and Multiplier uses a pipelined architecture to carry out the operation. The Taylor series is used to develop an algorithm for pipelining the divider which is explained in Section 3.2 of this paper while the Multiplier uses a Wallace Tree Algorithm. Additionally, a Look up Table and Multipliers are included in the Divider architecture. The Wallace Tree Algorithm enables pipelining the multiplier by calculating all the partial products in the first clock cycle and then adding them using a set of half and full adders. Both of these algorithms are described in the later Section 3.2 and Section 4.1 of this paper. The goal of this project is build a Single precision floating point Divider and Multiplier using pipelined architecture where once the pipeline is filled, the divider and multiplier will generate an output at every clock cycle, thus reducing the latency.

1.1 Organization

Chapter 2 discusses different Divider Algorithms and provides background research. Chapter 3 presents the design and implementation of the Single Precision Floating point Dividers and the Architecture. Chapter 4 discusses the design and implementation of a Floating Point Multiplier using Wallace Tree Algorithm. Chapter 5 shows the tests and Result of the implementation of Single Precision Floating Point Divider and Multiplier and compares it's results implemented on different technologies. Finally, Chapter 6 discusses about the Possible future work and concludes the paper.

Chapter 2

Background Research

Dividers are an important part of today's Graphics Processor Units (GPU) and a good amount of work has been done by previous researchers in this field of study. Most of the Dividers fall into two categories namely Slow Dividers and Fast Dividers. Slow Divider's are the one's which are designed using the basic divider algorithm which is the shift and subtract algorithm discussed in the Section 2.1 of this chapter. While Fast Divider's can be designed using various algorithms like the Newton-Raphson Algorithm, Goldschmidt [6], etc. and pipelining stages as described in this chapter. A divider for handling complex number is described in [7]. Different Dividers have been implented using Multiplication operations described in [8] and [9]. Double precision floating point dividers have been described in [10] and [11]. Dividers can be further categorized into Fixed-point or Floating point Dividers.

2.1 Shift and Subtract Divider

This the simplest and basic divider designed by subtracting the divisor from the dividend and then comparing the result with the dividend. If the subtraction does not result into a negative number, the quotient is incremented by 1 and then the divisor is again subtracted from the dividend. This procedure is carried out till a negative result is achieved and then stored as a remainder [12]. The algorithm of this divider is shown in Figure 2.1. This algorithm is easy to implement and takes less hardware but it is extremely slow and thus cannot be used in today's fast GPU processors. Also this algorithm is very difficult to implement for floating point numbers. Due to these two major drawbacks, various different Algorithms were developed which are described in the Section 2.2 and Section 2.3 of this chapter.

2.2 Newton-Raphson Algorithm

Initially this method was developed in Numerical Analysis by Isaac Newton and Joseph Raphson, for finding the roots of a real-valued function. It was later developed for division operation's. Different Mathematical operations such as Multiplication and Subtraction are required in the Newton-Raphson's computational algorithm. At the start of the algorithm, the reciprocal of the Denominator is calculated and then it is multiplied with the Numerator to find the final Quotient. This multiplicative inverse of the denominator is calculated using iterative process and then the calculated multiplicative inverse is multiplied with the dividend to compute the final Quotient.

In the design mentioned in [13], the division algorithm is implemented using a 32 Bit floating point Multiplier and Subtractor Module. Scaling of the Divisor is carried out to achieve minimal and maximal relative error where the scaling is kept within the interval

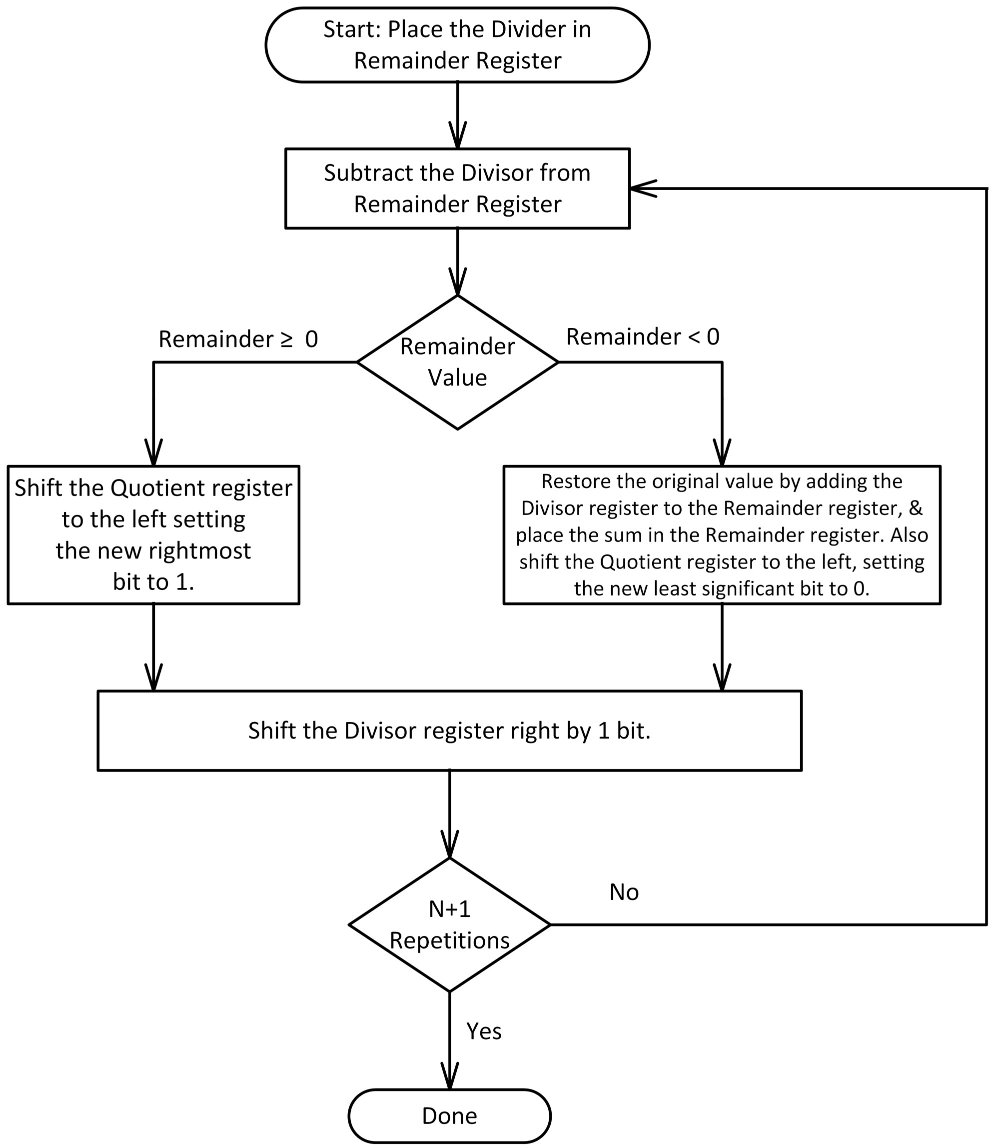

Figure 2.1: Basic Shift Divider[1]

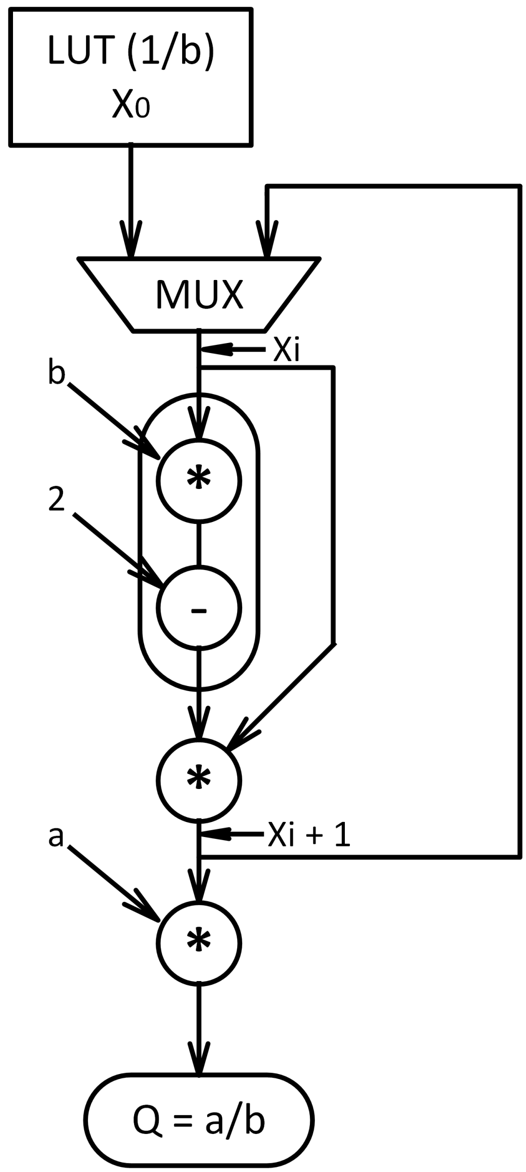

Figure 2.2: Newton - Raphson's Divider [2]

(0.5, 1). This scaling of the divisor is achieved by using shift operators. The algorithm is designed in such a way that to achieve higher accuracy, more iterations are required. The Newton-Raphson Algorithm converges faster for a single iteration and thus requires several Multiplier and Subtractor module's to compute continuous iterations [14]. Figure 2.2 shows the flowchart of a Floating Point Division using Newton_Raphson Algorithm. For a single iteration in this algorithm, two Multiplier and a Subtractor module are used. In [13] 3 iterations are computed in one cycle to get the required division result. The result can be improved by refining the multiplicative inverse.

The major disadvantage of the Newton_Raphson computational algorithm is computational failure if certain conditions of convergence are not met. This limits the use of this Algorithm as reliability of results being a major issue. Also the use of the multiplicative inverse results in the divider requiring a larger area compare to other algorithms [15]. Another disadvantage with the standard form of the Newton- Raphson divider is that the multiplications in the iteration are dependent to the previous stage and thus it must be performed serially. Hence two or more multiplications are required in each iteration [2].

2.3 Liddicoat and Flynn Divider

The Liddicoat and Flynn divider uses the first three terms of Newton and Raphson Divider [15]. The number of multiplication iterations is reduced in this divider and thus resulting in reduced latency [16]. The quotient can be directly expressed as a product of the dividend and the k^{th} order Newton-Raphson divider. The value of k decides the accuracy and the size of the Divider; the reater the value of k, the higher the accuracy and higher the complexity and area of the Divider [17]. This creates a trade-off between the size and the computational complexity of the algorithm.

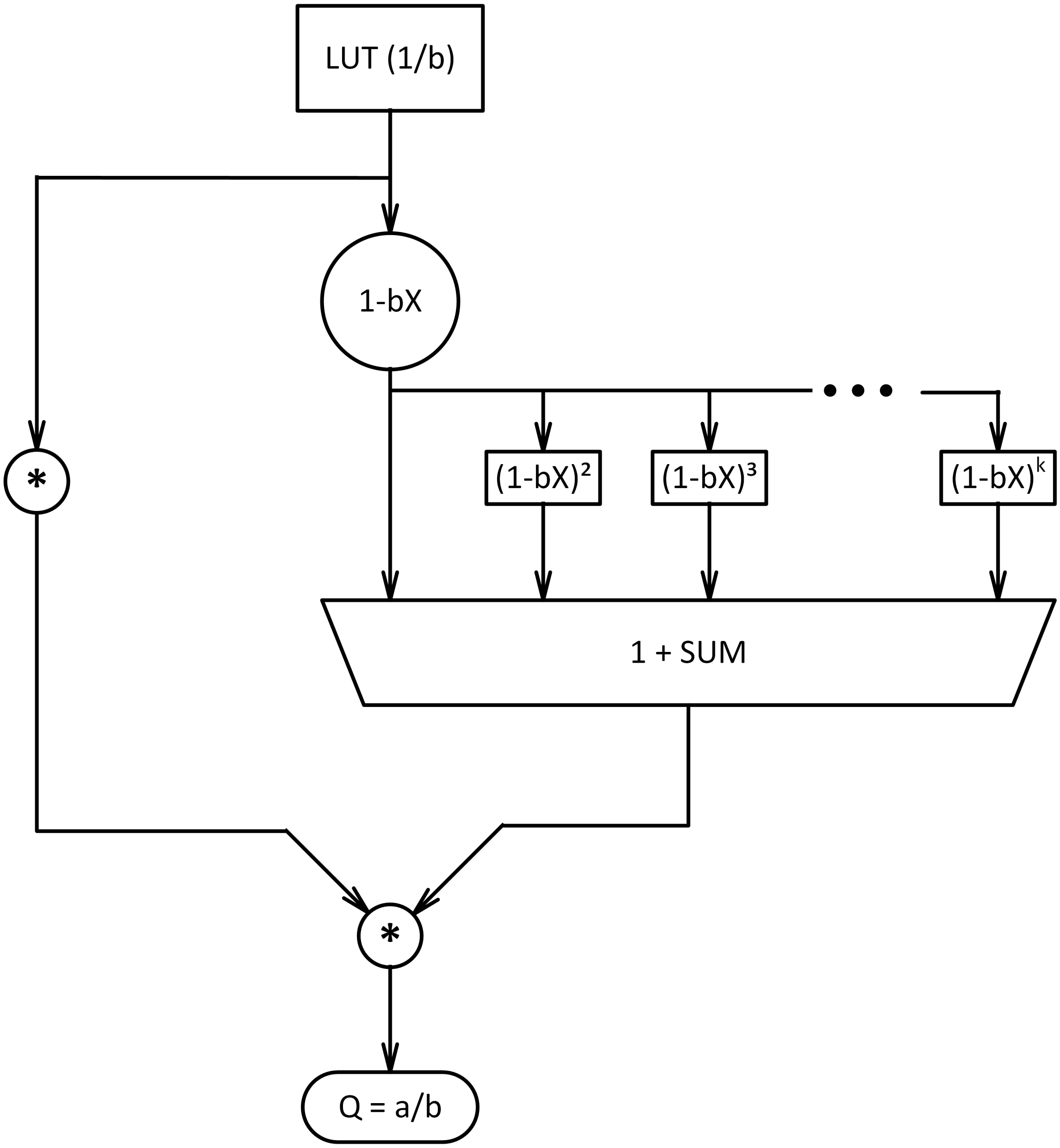

Figure 2.3: Liddicoat and Flynn Divider [2]

As seen from Figure 2.3, the value of 1/b is stored in a look up table. The selection of the size of the register b implies a trade-off between the lookup table size and error induced in computation. The next step is to express the quotient directly as the product of the dividend and the Newton-Raphson reciprocal approximation which is found by Eqn. 2.1.

$$aX(1 + (1 - bX)^1 + (1 - bX)^2 + ... + (1 - bX)^k) \tag{2.1}$$

Here the value of X is the initial estimation of the dividend which is found in the look up table. This equation is achieved by using Binomial Expansion series from the Newton-Raphson Divider [13]. As the number of terms increases, the error in the quotient reduces resulting in increase in the complexity for implementation.

As only the MSB part of the dividend is stored in the Lookup table to reduce the size of it, the value of X generally contains an error due to approximation. The number of computation cycles required for an n-bit reciprocal stored in the Lookup Table containing only the MSB part required in a single iteration will be $\frac{n}{k+1}$ bits for a k^{th} order approximation. Also the number of bits of precision of X are increased approximately by a factor of $k + 1$ for a k^{th} order number[2], [18].

For divider shown in Figure 2.3, the divider unit can be fully pipelined by using two small multipliers, a powering unit for computing then^{th}order Newton_Raphson coefficient and a full multiplier to get the final product. The small multipliers are used to calculate the intermediate coefficients $(1 - bX)$, as X being calculated using the Lookup Table[2].

This divider architecture can be implemented using a fully pipelined version and thus can produce divide instruction per clock cycle. This divider utilizes the higher-order Newton-Raphson reciprocal approximation, this limits the accuracy to the number of coefficients used in the implementation. This is the major disadvantage of the Liddicoat and

Flynn Divider as the accuracy is limited to the number of Newton-Raphson divider and thus higher accuracy leads to increase in the complexity of the divider [15].

Chapter 3

Design and Algorithm

This chapter discusses the Design and Architecture of the Single Precision Floating point Divider using pipeline approach.

3.1 P. Hung's Divider using Taylor Series

P. Hung proposed a new high radix division algorithm which was based on Taylor Series expansion. With a slight modification in the Taylor Series, P. Hung showed a way to pipeline this algorithm. The previous algorithms described in chapter 2 considered each term of the Taylor series as a separate element and thus large look-up tables were needed to store the values of the coefficients thus making the algorithms more complicated [19]. Also reducing the size of the lookup table would make the results more error prone and thus had this major drawback.

P. Hung proposed an algorithm which would combine the first two terms of the Taylor Series and together, this would drastically reduce the size of the lookup table required to implement the algorithm and would provide more accurate results. As seen from Figure

3.1, the algorithm achieves a faster division by a getting the value from the lookup table and at the same time calculating $(y_h - y_l)$. This saves one clock cycle and thus reduces the latency of the division operation. This value of $\mathrm{y_h}$ and $\mathrm{y_l}$ is derived from the Dividend which is considered as Y which is described in section 3.1.1.

3.1.1 P. Hung's Algorithm

Consider the Dividend X and Divisor Y to be two 2m bit numbers between one and two. These two numbers can be defined by the following equation.

$$X = 1 + 2^{-1}X_1 + 2^{-2}X_2 ...... + 2^{-(2m-1)}X_{2m-1} \tag{3.1}$$

$$Y = 1 + 2^{-1}Y_1 + 2^{-2}Y_2 ...... + 2^{-(2m-1)}Y_{2m-1} \tag{3.2}$$

These two equations 3.1 3.2 have been derived from Taylor Series Expansion [3]. From the Equations 3.1 and 3.2, the divisor Y is separated into two parts called y_h and y_l where the y_h contains the MSB part of Y which is $(m + 1)$ bits wide while y_l contains the LSB part of Y which is$(m - 1)$ bits wide. Thus the division equation can be written as

$$\frac{X}{Y} = \frac{X}{(Y_h + Y_l)} \tag{3.3}$$

$$\frac{X(Y_h - Y_l)}{(Y_h + Y_l)(Y_h - Y_l)} \tag{3.4}$$

$$\frac{X(Y_h - Y_l)}{Y_h^2} \tag{3.5}$$

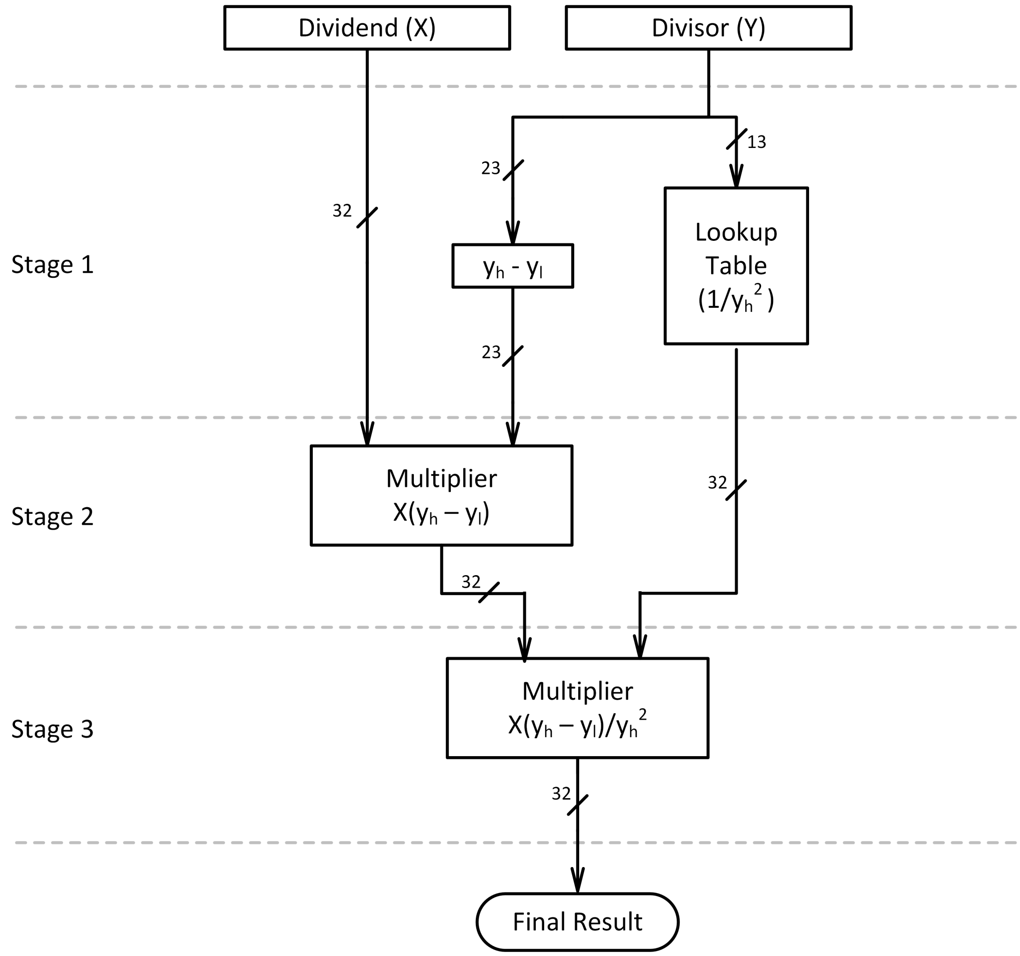

Figure 3.1: P. Hung's Divider Algorithm using Taylor Series [3]

Equation 3.5 is the final equation for Hung's divider. According to Equation 3.5, the division operation can be performed by mean's of multiplying the Dividend X with $(Y_h - Y_l)$ and also with the$1/(Y_h)^2$. As seen in Figure 3.1, the divider can be thus implemented using 3 stages of Pipeline. In the first stage the value of $(Y_h - Y_l)$ is calculated. Also the value of $1/(Y_h)^2$ is taken from the lookup table. Using a multiplier, X is then multiplied with $(Y_h - Y_l)$ term in the second stage of the pipeline. In the 3$^{\text{rd}}$ stage the result of the second stage is multiplied using a multiplier with $1/(Y_h)^2$ term to get the final result.

3.1.2 Error Analysis of P. Hung's Algorithm

The term $1/(Y_h)^2$ is stored in a Look up table and thus plays an important role in the overall size of the Divider. The divider size can be reduced by decreasing the bit width of Y_h. For every one bit decrease in the but width of Y_h, the lookup table size reduces by 2 times and vice-versa. But this decrease is size of the lookup table comes at a cost of increase in the error of the final result [20]. Apart from this, there are 3 other sources of error in this Division Algorithm namely Taylor Series Approximation error, Rounding error due to first multiplication, and Rounding error due to Second multiplication [3]. The total error can be expressed using the following equation 3.6.

$$E_T = E_L + E_{TS} + E_{M1} + E_{M2} \tag{3.6}$$

Where,

E_T= Total Error

E_L = Lookup table rounding error

E_{TS}= Taylor Series Approximation Error

E_{M1}= Rounding error due to first Multiplication

E_{M2}= Rounding error due to second Multiplication

For minimizing the error, the divider needs to carefully designed such that $E_L \leq 0$, $E_{TS} \leq 0$, $E_{M1} \leq 0$ and $E_{M2} \geq 0$ [3]. To achieve this, the result of the first multiplication should be truncated to 2m +2 bits of MSB thus resulting in value of Y_h should also be $2m + 2$ bits. The result of the second multiplication should be truncated to $2m$ bits of MSB which will be the final result. The total error generated by all of this factors is less than 1ulp, out of which the lookup table and first multiplication generates an error of less than 1/4 Unit of Least Precision (ulp) while the second multiplication generates an error of less than 1ulp.

3.2 Single Precision Floating Point Divider Algorithm

The division algorithm mentioned in Section 3.1 requires a lookup table whose size can be decreased by approximation. But even after this reduction, the size of the lookup table is still 13KB for a single precision division [15]. Thus, this size of the lookup table constitutes a larger area compare to that of iterative dividers. Also it is impossible to implement this algorithm for a double precision number considering the large amount of lookup table size. Also as discussed in Section 3.1.2, the error induced in the Divider of Double Precision would greatly increased and thus this algorithm proves to be unsatisfactory for a Double Precision Floating point Division.

This paper shows an Algorithm described in [15] where the size of the Lookup Table used in this Paper is just 2KB compare to the large size of Lookup table used in [3]. The algorithm mentioned in [3] is also slightly modified to reduce the error. A course and a fine quotient is found and then added together to get the final result. This algorithm is explained in detail in the later sections of this chapter. Section 3.2.1 describes the IEEE-754 standard for Single Precision Floating Point number. Section 3.2.2 describes the Algorithm and Architecture used in this project while Section 3.2.3 shows the Error Analysis and Error induced due to this Algorithm.

3.2.1 IEEE-754 Standard for Floating Point Numbers

The IEEE-754 Standard was developed specially for Computers to recognize and perform operations on Floating point numbers developed by the Institute of Electrical and Electronics Engineers in 1985 which was later modified in June 2008. It provides a standardize method for performing computation in floating point point numbers. The major advantage of this standard is to be able perform operations on floating point numbers independent of it being on a software or hardware platform or even combination of the two.

The IEEE 754 standard allows the user to specify the floating point number into various different formats such as Single Precision, Double Precision, Extended and Extendable Precision according to user requirements. This standard includes formats to define both Binary and decimal floating point formats, performing Arithmetic operations on them, conversion of Binary Floating point to Decimal Floating point and vice-versa and also Exception handling for numbers who comes in the category of Not A Number (NaN) [21] defined in Section 3.2.1.2. The floating point formats in Single and Double Precision format is discussed in the below Section 3.2.1.1

3.2.1.1 Floating Point Format

IEEE 754 standard allows the user to define a floating point number into various different formats such as Single Precision, Double Precision, Extended and Extendable Precision. In this section, Single and Double Precision format for representing floating point numbers is discussed. Both of the these formats are divided in three parts namely Sign Bit, Exponent Bit and Mantissa or Significand Bit, the difference being in the size of the Single Precision and Double Precision format shown in Figure 3.2 and Figure 3.3

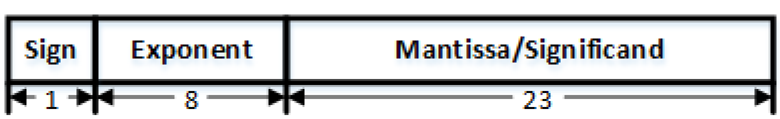

Figure 3.2: Single Precision Floating Point Format[4]

. The Single Precision format consists of 32 bits whereas Double Precision format consists of 64 bit where the MSB 1 bit defines the Sign of the number in both the format.

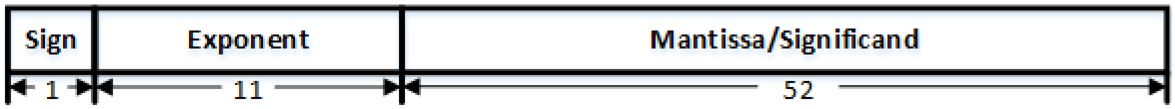

Figure 3.3: Double Precision Floating Point Representation[4]

The Exponent part in the Single precision format is stored in excess 127 format where the range of the exponent part can be between 128 to -127. Similarly for the double precision format, the exponent part is represented in excess 2048 format and the range of it is between 1023 to -1022. Thus a very small number can be defined using both of these formats. The Mantissa/Significand part is a bit tricky in both of these formats. The mantissa is always stored in normalized form which means 1 is considered to be always present in the MSB part of the Mantissa also called as implicit 1 in the MSB. As this 1 is always considered to be present in the MSB this 1 in the MSB is never stored in Mantissa part of both Single and Double Precision format but it is always used if any operation considering it to be present. It can be explained by the following example.

Eg. Number 4.5 can be written in Floating point Binary as

$$4.5 = 100.1$$

The result in the normalised form is represented as

$$4.5 = 1.001 \text{ X } 2^2$$

To store this number in the Single Precision form, the Mantissa is stored as 001 while the exponent is incremented by 2 to take care of the normalized form.Therefore 4.5 in single precision format is stored as 0x4090_0000. Due to this normalization, storing the Mantissa part becomes a slightly complex operation due to the implicit hidden 1 in the Mantissa Part. If the value of the Mantissa part is less than 1.0, then a left shift operation is performed till the MSB bit is 1. The number of times this left shift operation is performed, the Exponent bit is decreased by the same amount. Similarly, if the number is greater than

2.0, then a right shift operation is performed till the value is than 2.0 and the exponent bit is increase by the same amount [22]. This operation is applicable to both the Single and Double Precision format.

Processing on the Double Precision Floating Point format is slower than it's counter part Single Precision Floating Point form due to its size. Thus Single Precision format is used in most of the processors. Majority of the times, the Double Precision format is used for Parallel computing in the GPU's.

3.2.1.2 Not A Number (NaN)

IEEE 754 standard has an inbuilt way to define not defined numbers. Operations such as a number divided by 0 will result in infinity and thus result in exception case of number overflow. Also operations such as taking a square root of a negative number will result in an Imaginary number which will trigger the exception handling of this format number. The IEEE 754 standard defines such exception numbers as Not a Number (NaN). It should be taken into account that NaN numbers are not equal to infinity, it is just a technique to represent exceptional cases in operation [21], [4]. IEEE - 754 format supports two types of NaN namely Signaling and Quiet in all floating point operations [21].

3.2.2 Algorithm and Architecture

Equation 3.5 can be considered as the Course Coefficient of the Quotient. As discussed in Section 3.1, Hung's divider induces an error in the final result. This error can be reduced by introducing adding Course and Fine Coefficients of the Quotients. The course coefficient from Equation 3.5 can be defined as

$$\widetilde{Y} = \frac{X(Y_h - Y_l)}{Y_h^2} \tag{3.7}$$

Where $\widetilde{Y}$ is defined as the course coefficient.

From [15], the subdividend $\widetilde{X}$ can be defined as,

$$\widetilde{X} \approx X - Y \cdot \tilde{Q}$$

$$= X - Y \cdot \frac{X(Y_h - Y_l)}{Y_h^2}$$

$$= X\left(1 - Y \cdot \frac{(Y_h - Y_l)}{Y_h^2}\right)$$

$$= X\left(1 - \frac{(Y_h + Y_l)(Y_h - Y_l)}{Y_h^2}\right)$$

$$= X\left(1 - \frac{Y_h^2 - Y_l^2)}{Y_h^2}\right)$$

$$= X\left(\frac{Y_h^2 - Y_h^2 + Y_l^2)}{Y_h^2}\right)$$

$$\therefore \widetilde{X} = X\left(\frac{Y_l^2)}{Y_h^2}\right) \tag{3.8}$$

This Subdividend $\widetilde{X}$ can be utilized to find the fine coefficient $\tilde{\tilde{Q}}$. This can be found using the value of $\widetilde{X}$ from equation 3.8 in equation 3.7 which would give us.

$$\tilde{\tilde{Q}} = \frac{\widetilde{X}(Y_h - Y_l)}{Y_h^2} \tag{3.9}$$

According to [15], the Final Quotient can be derived by adding Equation 3.7 and 3.9,

$$Q = \tilde{Q} + \tilde{\tilde{Q}}$$

$$= \frac{X(Y_h - Y_l)}{Y_h^2} + \frac{\widetilde{X}(Y_h - Y_l)}{Y_h^2}$$

$$= \frac{(X + \widetilde{X})(Y_h - Y_l)}{Y_h^2} \tag{3.10}$$

$$= (X + \widetilde{X})A$$

$$= (X + X - Y \cdot \tilde{Q})A$$

$$= (2X - Y \cdot AX)A$$

$$= (2 - AY)AX \tag{3.11}$$

where,

$$A = \frac{(Y_h - Y_l)}{Y_h^2} \tag{3.12}$$

Equation 3.11 refers to the final equation of the Single Precision Floating Point Divider. Equation 3.11 can also be implemented for Double Precision floating point division by increasing the size of the Multiplier and Lookup table[23]. This algorithm can be implemented using the Pipeline Architecture shown in Figure 3.4.

In the first stage, the dividend Y is separated into two terms Y_h and Y_l where the Y_h is 12 bits MSB, while $Y_l$is 11 bits LSB. Y_h is then concatenated with the implicit 1 of the Mantissa in Floating Point Representation. The value of Y_h thus becomes 13 bits which is then subtracted with Y_l.

The lookup table is constructed in the form of ROM implementation and the input Y_h is considered to be the address of the memory location. The memory location the value of $1/(Y_h)^2$ is stored in Single precision floating point format. This storage in the single precision format is important because the equation $1/(Y_h)^2$ would result in a floating point number. Thus it needs to be stored in a standardized format so that the resultant value from the Lookup table can be easily utilized in the later stages of the Pipeline. At the same time, value of $(Y_h - Y_l)$ is also calculated. As mentioned above, Y_h is concatenated with 1 at the MSB, therefore the result of this subtraction is never a negative number. This value is also converted into Single Precision Floating point format to simplify the further operations.

These results achieved in first stage of pipeline are then multiplied with each other to obtain the value of A described by Equation 3.12. A 32 bit floating point multiplier is used to perform this operation. As discussed in Section 3.2.1, special care needs to be taken for floating point multiplication. The 32 bit number in Single Precision format is first separated into its Sign, Exponent and Mantissa bits. This number is then checked with all the NaN conditions and then if this test passes then the Mantissa is unpacked from its Normalized form [22]. The 23 bit Mantissa is converted into a 24 bit number and then multiplication

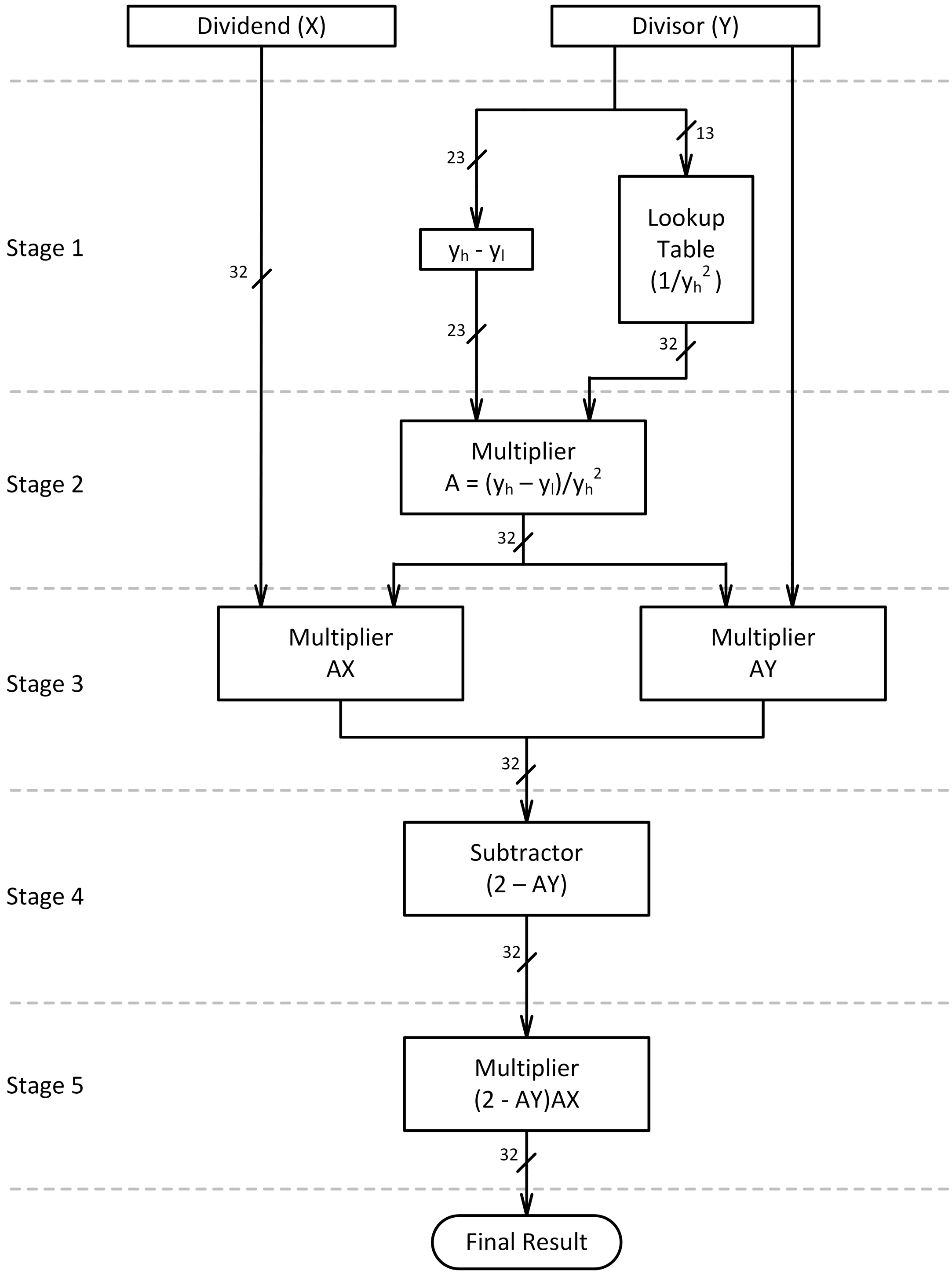

Figure 3.4: Architecture of the Divider

of the Mantissa is carried out [5]. The exponent bit of the two inputs are added while an Ex-OR operation is carried out with the sign bit of both inputs to the Multiplier [24]. The architecture of the Multiplier used in this operation is described in Chapter 4. The result of the multiplication operation is truncated to 24 bits and then a packing operation is carried out on to Normalize the Mantissa. Finally, the Sign, Exponent and Mantissa bits are then concatenated to get the final result of Floating point multiplication. The value of A achieved in the second stage is multiplied with the Dividend X and Divisor Y to get the product AX and AY. Similar multiplication operation is carried out in the third stage as to the second stage.

In the fourth stage of Pipeline, the$(2 - AY)$ operation is carried out using a floating point subtracter module. Similar to multiplication operation, the mantissa needs to be unpacked for Subtraction too. But here, the exponent of both the inputs first needs to be made equal and the Matissa bits are adjusted accordingly. After unpacking the subtraction operation is carried out on the Mantissa bits. The same algorithm can be used for addition by just Adding the Mantissa bits instead of Subtracting. The Sign, Exponent and Matissa bits are then concatenated to get the result of Pipeline Stage four. This result is then multiplied with the product of AX using the same multiplier algorithm explained above to get the final result of Division operation.

3.2.3 Error Analysis

Similar to Section 3.1.2, even this algorithm induces an error. There are three types of error induced in the algorithm which are error due to the lookup table, rounding error in Multiplication operation and rounding error in Subtraction operation. Four Floating point multipliers are used in the overall algorithm and the total error in the algorithm is as follows,

$$E_T = E_{LT} + E_S + E_{m1} + E_{m2} + E_{m3} + E_{m4} \tag{3.13}$$

Where,

E_T= Total Error

E_{LT} = Lookup table rounding error

E_S= Rounding error due to Subtraction

E_{M1}= Rounding error due to first Multiplication

E_{M2}= Rounding error due to second Multiplication

E_{m3}= Rounding error due to third Multiplication

E_{m4}= Rounding error due to fourth Multiplication

The lookup table error is due to limiting the storage of the $1/(Y_h)^2$ term. This is done to reduce the size of the lookup table. This reduction in the bit width of the Y results in the Lookup table error. The multipliers used in this algorithm are 24 bit multipliers which results in a 48 bit result. Thus the result of this multiplication operation is truncated to 24 bits MSB which results in the rounding error. All the errors can be minimized similar to the method discuss in Section 3.1.2. In [15] an error due to limitation of entries in the lookup table is also introduced as they restrict the number of entries to reduce the size of the lookup table which is not the case in the algorithm used in this paper.

Chapter 4

Wallace Tree Multiplier

Similar to Division, Multiplication is also an important arithmetic operation in today's processors. The basic multiplier consists of Adders, Shifters and AND gate to perform the multiplication shown in figure 4.1. A basic 4 bit multiplier is shown in the Figure 4.1 which results in an 8bit product. PPO, PP1, PP2 and PP3 are the partial products generated using the AND gate in the multiplication operation. A sum of this partial product is calculate using ADDER to generate the final result of Multiplication. The number of partial products generated increases with respect to the Multiplier's bit size. For a 16 bit adder, the 256 partial products are generated. This number can be reduce by using Booth's Encoding which is a technique to reduce the number of Partial products generated during a multiplication operation [24].

Booth's Algorithm is one of the most commonly used algorithm for Multiplications operation used for both signed and unsigned numbers. It uses Shift operation which are faster than Adders, to perform multiplication. A high speed piplined divider in discussed in [24] where they use booth encoding to reduce the number of partial products. Figure 4.2 shows Booth's Multiplier Algorithm for a 32 bit Multiplier. It checks the 2 bit's of

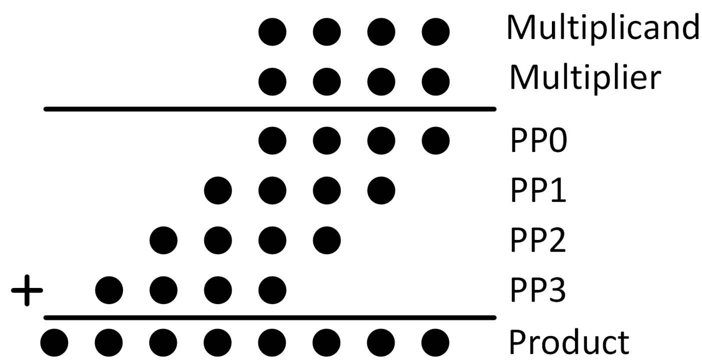

Figure 4.1: Basic Multiplier [5]

the multiplicand and does the operation accordingly. As seen from Figure 4.2 it performs No Operation if these 2 bits are 00 or 11 while it performs an Addition or Subtraction operation depending on these two bits.

Booth encoding is used in [5] where the generated partial products are later added using Carry Look Ahead Adder. Booth's encoding is a technique used to reduce the number of Partial Products generated which are the main agent of time required for Multiplication operation. Booth's encoding can reduce these number of Partial products generated to almost half by using Radix-4 technique.

As seen from Figure 4.3 a Carry Look Ahead Adder is used to calculate the sum of the partial product. This CLA's have a huge drawback of propagation delay due to the fact that every addition operation has to wait for the previous Carry bit to be generated creating a huge delay in operation. As the CLA is used in every stage, the propagation delay induced in this Multiplier would huge and may not meet the timing requirement. To reduce the propagation delay induced due to this CLA adders, a Wallace Tree Algorithm described in Section 4.1 can be used.

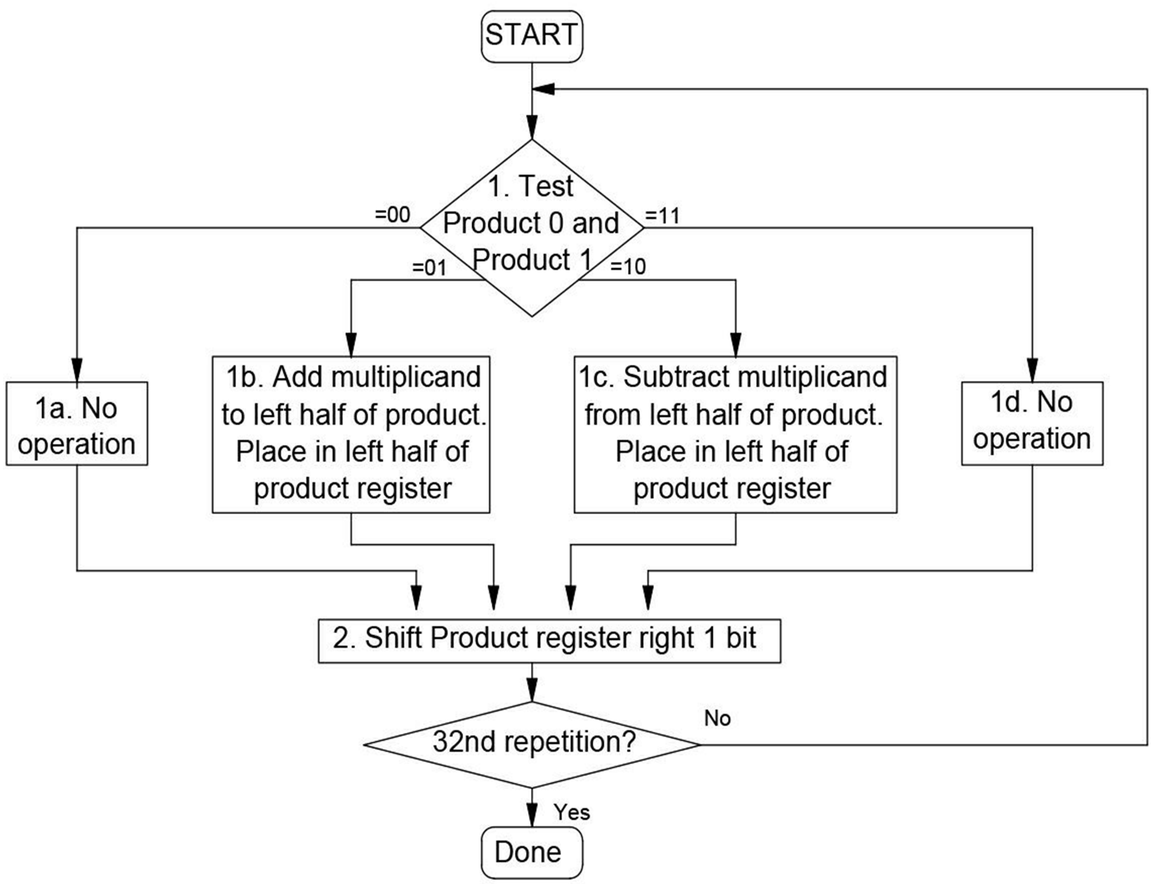

Figure 4.2: Booth's Algorithm

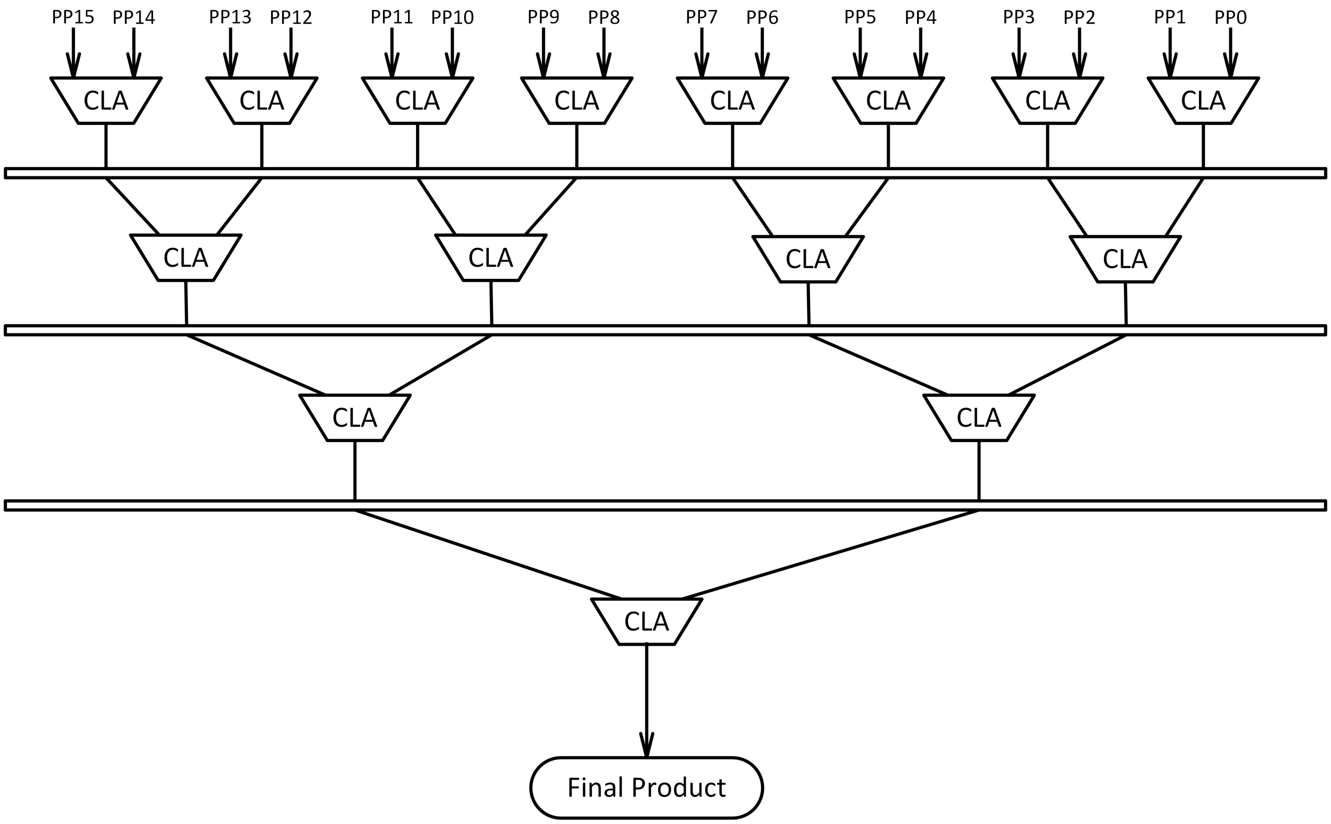

Figure 4.3: 8 Bit Multiplier Using CLA Adder [5]

4.1 Algorithm

The major advantage Wallace Tree Algorithm provides compared to the traditional CLA adders in the Multiplier is that the Adder operation does not have to wait for the previous stage Carry. It only considers the previous stage generated Carry at a later stage instead of waiting it during every Adder operation. This reduces the propagation delay drastically as we increase the size of the multiplier.

Figure 4.4 shows a Wallace Tree algorithm for an 8bit Multiplier. In Wallace Tree Algorithm, all the Partial products of the Multiplication are calculated in the first stage. These partial products are calculated using an AND gate where each bit of Multiplicand is ANDed with every bit of the Multiplier. Sum of partial products is calculated using Half

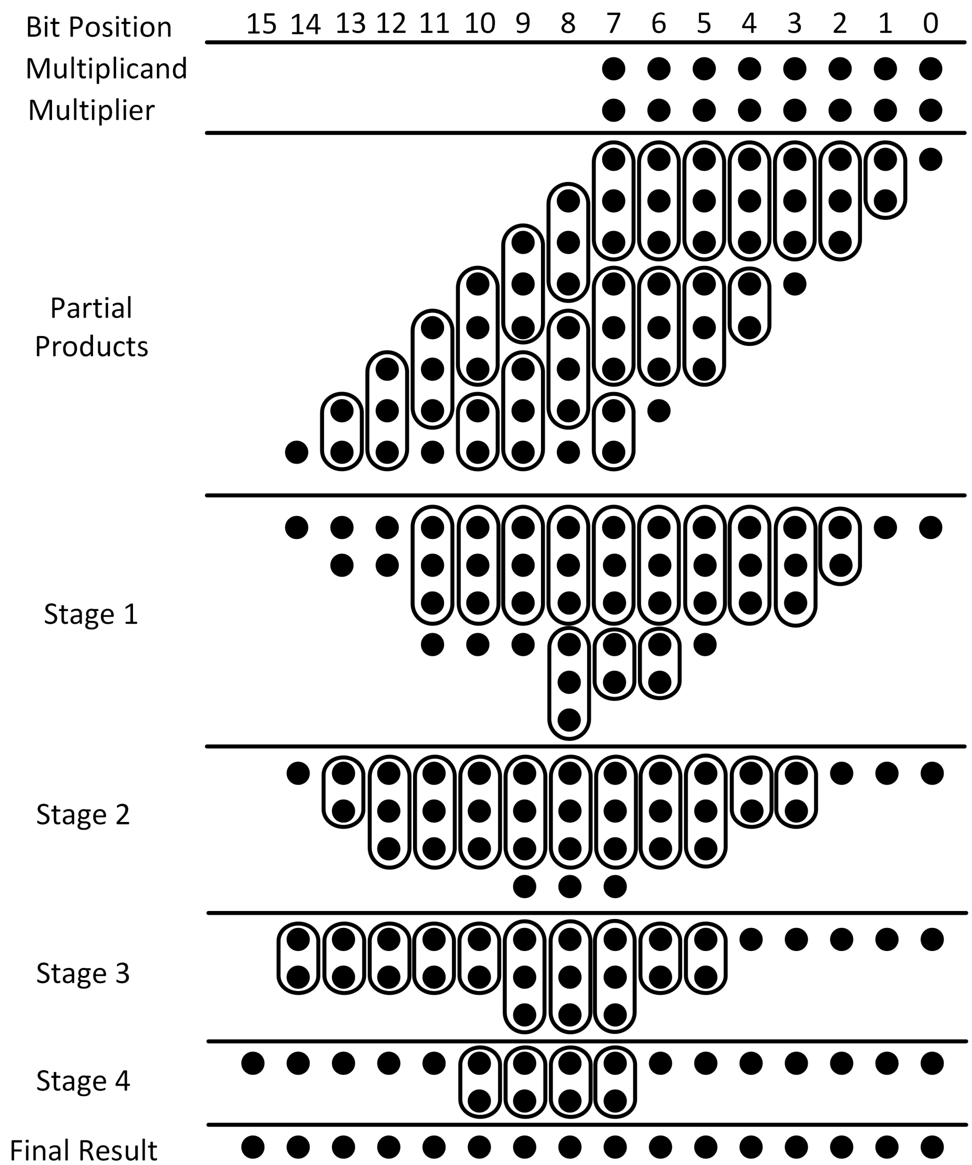

Figure 4.4: 8 Bit Wallace Tree Multiplier

and Full Adders according to the number of Partial products in each column. Whenever an Addition operation is performed in a column, the generated carry is placed in the neighboring column similar to CLA adder but the addition in the neighboring column is independent of this generated carry in that stage [25]. In the next stage, this generated carry is used in the addition operation and thus the addition operations are not restricted because of the generated carry. As seen from figure 4.4, addition of the partial products is carried out in different stages, thus a pipeline architecture can be implemented to on Wallace tree multiplier.

An entirely combinational circuit is used to implement the Wallace Tree Multiplier in this project. All the partial products were calculated by using AND gate between every bit of Multiplicand and Multiplier. Approximately a 576 partial products were generated in the first stage on which addition operation was carried out in 10 stages. Half of the bits of the final results are generated by the 8^{th} stage of the addition. This is achieved by adding according to the remaining partial products as the algorithm progresses. This further reduces the propagation delay as only half of the final result needs to be calculated in the last stage, thus resulting in a CLA adder only for half of the bits in the final stage.

Chapter 5

Results

This chapter discusses the results from the implementation of the Single Precision Divider and Multiplier.

5.1 Floating Point Divider

The Single Precision Floating Point Divider was implemented using the Architecture described in Chapter 3.2.2. It was synthesized on different technologies using two pass synthesis strategy: RTL logic synthesis followed by Design for Test (DFT) synthesis. A full scan methodology was used in the synthesis for test structure insertion, which inserts scan chains throughout the design for testing. A clock of 50Mhz was used during the testing of the entire design.

5.2 Floating Point Multiplier

As discussed in Chapter 4, the Multiplier was implemented using a Wallace Tree Algorithm. Similar to the Floating point Divider, the Multiplier was also Synthesized on a different

Table 5.1: Area, Power, Timing, and DFT Coverage of Divider

		32nm	**65nm**	**180nm**
Area	Sequential Area	23901	31325	250607
	Combinational Area	84598	80222	563934
	Buf/Inv Area	4982	4060	15750
	Total Area	145245	111548	814542
Power	Internal Power	1.533E3 µW	1.99 mW	14.4372 mW
	Switching Power	65.6268 µW	0.1134 mW	2.3339 mW
	Leakage Power	1.025E10pW	5.2629e03 nW	3.807E3 nW
	Total Power	1.167E5 µW	2.1161 mW	16.7749 mW
Timing	Slack	14.845ns	14.065ns	14.392ns
DFT Coverage		98.38%	99.54%	99.57%
Latency (Clock Cycles)		41	41	41

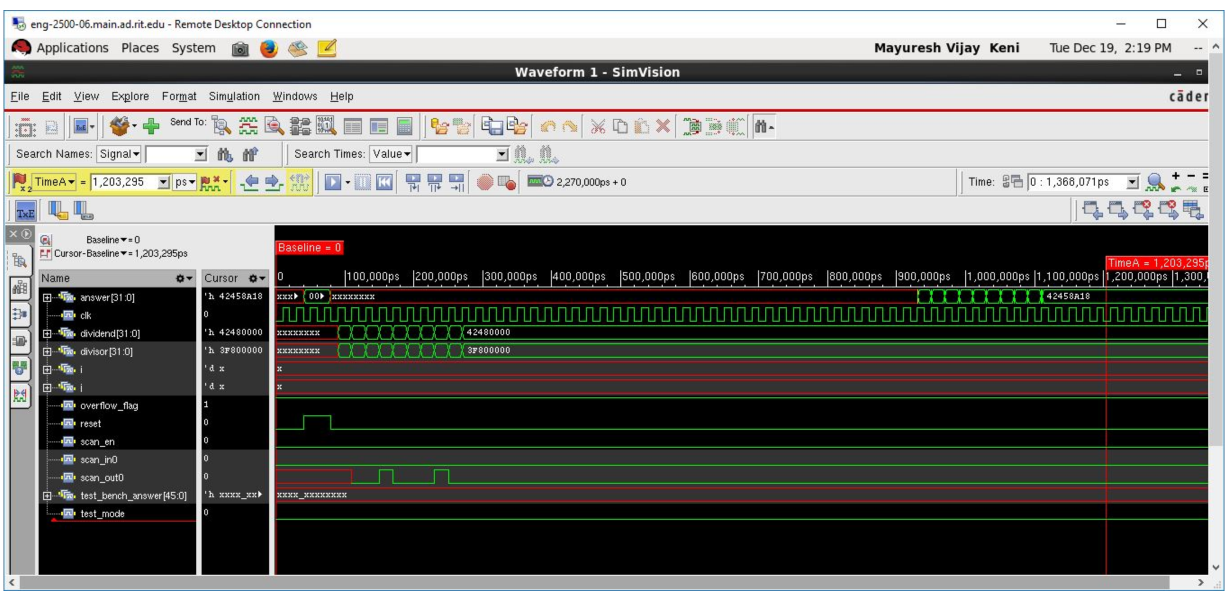

Figure 5.1: 32nm Divider Result

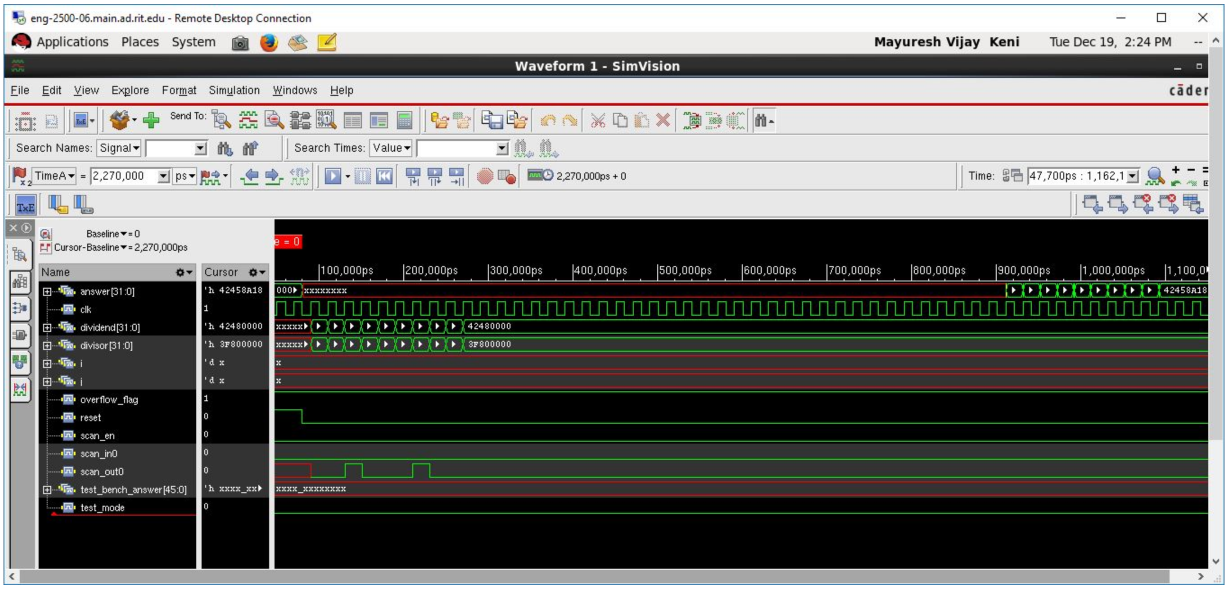

Figure 5.2: 65nm Divider Result

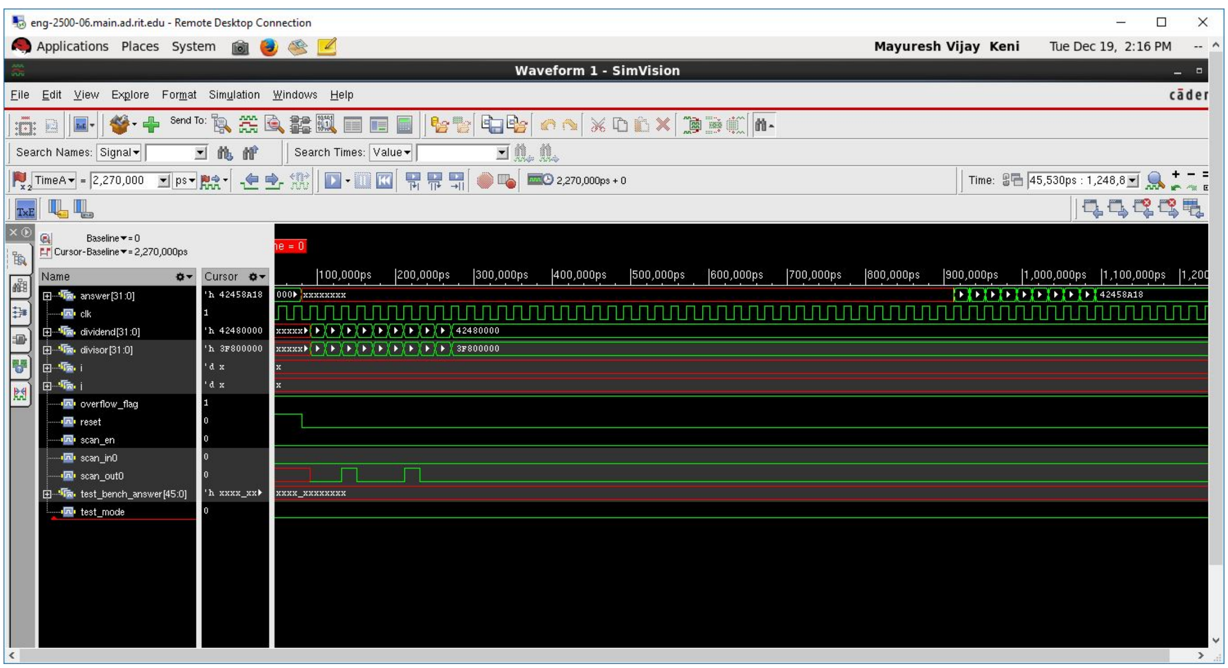

Figure 5.3: 180nm Divider Result

Table 5.2: Area, Power, Timing, and DFT Coverage of Multiplier

		32nm	**65nm**	**180nm**
Area	Sequential Area	2855	3507	28866
	Combinational Area	12178	11741	83991
	Buf/Inv Area	637	416	2125
	Total Area	19370	15249	112858
Power	Internal Power	197.876 µW	0.2797 mW	2.169 mW
	Switching Power	19.6011 µW	2.877E-02 mW	0.5953 mW
	Leakage Power	1.4197E09 pW	668.708 nW	591.9263 nW
	Total Power	1.6372E03 µW	0.3092 mW	2.7649 mW
Timing	Slack	14.4904ns	14.923ns	16.5276ns
DFT Coverage		99.44%	99.42%	99.33%
Latency (Clock Cycles)		9	9	9

technologies using two different synthesis options, RTL logic synthesis and DFT Synthesis with a full scan methodology.

Chapter 6

Conclusions

This chapter discusses conclusions from this project as well as future work that could be completed.

6.1 Conclusions

This paper describes implementation of a Floating point Divider and Multiplier which was benchmarked on different technologies. A standard 31 stage pipeline was implemented on the Divider while a 9 stage Pipeline was implemented on the Multiplier. The goal of this project was to implement a Divider and Multiplier using pipeline architecture so that once the pipeline is filled, there will be an output at the end of every clock cycle.

6.2 Future Work

The multiplier and the divider can always be improved and optimized. This paper discusses about the implementation of a Single Precision Multiplier and Divider. In future the same algorithm can be implemented on a Double Precision floating point number. In this project, the Multiplier has a latency of 9 clock cycles while the Divider has a latency of 31 clock cycles. In future, work can be done to reduce the latency of both of them. The number of partial products generated in the multiplier can be reduced by using Booth's Encoding.

References

[1] D. B. Najib Ghatte, Shilpa Patil, "Single precision floating point division," in *Fifth IRF International Conference*, vol. 13, 2014, pp. 294–298.

[2] A. A. Liddicoat and M. J. Flynn, "High-performance floating point divider," in *Digital Systems Design, 2001. Proceedings. Euromicro Symposium*, 2001.

[3] P. H. H. F. O. M. M. J. Flynn, "Fast division algorithm with a small lookup table," in *Signals, Systems, and Computers, 1999. Conference Record of the Thirty-Third Asilomar Conference*, 1999, pp. 137–150.

[4] B. Parhami, *Computer Arithematic.* Oxford University Press, 1999, vol. 42.

[5] P. D. Smruti Bokade, "Cla based 32-bit signed pipelined multiplier," in *Communication and Signal Processing (ICCSP), 2016 International Conference*, 2016.

[6] G. E. P.-M. W. E.Ferguson, "A parametric error analysis of goldschmidt's division algorithm," in *Journal of Computer and System Sciences*, vol. 70 (2005) 118-139, Aug. 2014.

[7] L. L. Dong Wang, Pengju Ren, "A high-throughput fixed-point complex divider for fpga," in *IEICE Electronics Express*, vol. 10, no. 4, 2013.

[8] P. MalÃk, “High throughput floating-point dividers implemented in fpga,” in *Design and Diagnostics of Electronic Circuits & Systems (DDECS), 2015 IEEE 18th International Symposium*, 2015.

[9] I. Rust and T. G. Noll, “A radix-4 single-precision floating point divider based on digit set interleaving,” in *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium*, 2010.

[10] X. Fang and M. Leeser, “Vendor agnostic, high performance, double precision floating point division for fpgas,” in *High Performance Extreme Computing Conference (HPEC), 2013 IEEE*, 2013.

[11] V. S. R. D. M. V. H. Neto, “Multiplier-based double precision floating point divider according to the ieee-754 standard,” in *Springer-Verlag Berlin Heidelberg*, vol. 262-267, 2008, 2008.

[12] S. O. . M. Flynn, “Minimizing the complexity of srt tables,” in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 6, 1998, pp. 141–149.

[13] N. Singh and T. N. Sasamal, “Design and synthesis of single precision floating point division based on newton-raphson algorithm on fpga,” in *4th International Conference on Advancements in Engineering & Technology (ICAET-2016)*, 2016.

[14] W. G. . E. Swartzlander, “Fault-tolerant newton-raphson and goldschmidt dividers using time shared tmr,” in *IEEE Transactions on Computers*, vol. 49, 2000, pp. 588–595.

[15] J.-C. J. . W.-C. P. . W. J. . T.-D. H. . M.-K. Lee, “A cost-effective pipelined divider with a small lookup table,” in *IEEE Transactions on Computers*, vol. 53, no. 4, Apr. 2004, pp. 490–496.

[16] T. J. K. Taek Jun Kwon, "Floating-point division and square root using a taylor-series expansion algorithm," in *Microelectronics Journal*, vol. 40, Apr. 2009, pp. 1601–1605.

[17] A. L. . M. Flynn, "Parallel square and cube computations," in *Signals, Systems and Computers, 2000. Conference Record of the Thirty-Fourth Asilomar Conference*, vol. 24, 2002, pp. 1645–1667.

[18] T.-J. K. . J. Draper, "Floating-point division and square root implementation using a taylor-series expansion algorithm with reduced look-up tables," in *Circuits and Systems, 2008. MWSCAS 2008. 51st Midwest Symposium*, 2008.

[19] T.-J. K. J.-S. M. J. S. J. Draper, "A 0.18ÎŒm implementation of a floating-point unit for a processing-in-memory system," in *Circuits and Systems, 2004. ISCAS '04. Proceedings of the 2004 International Symposium*, 2004.

[20] J.-C. J. W. J. H.-J. W. S.-H. K. W.-C. P. M.-K. L. T. don Han;, "A new pipelined divider with a small lookup table," in *ASIC, 2002. Proceedings. 2002 IEEE Asia-Pacific Conference*, 2002.

[21] I. C. S. M. S. C. I. of Electrical and E. E. I.-S. S. Board., *IEEE Standard for Floating-Point Arithmetic*, revision of ieee std 754-1985 ed. Institute of Electrical and Electronics Engineers, 2008.

[22] J. L. H. Patterson, D. A., *Computer Organisation and Design. The Hardware/Software Interface*, 5th ed. Morgan Kaufmann, 2013.

[23] S. K. N. Sandeep B. Singh, Jayanta Biswas, "A cost effective pipelined divider for double precision floating point number," in *Application-specific Systems, Architectures and Processors, 2006. ASAP '06. International Conference*, Dec. 2006.

[24] A. B. Qingzheng LI, Guixuan LIANG, "A high-speed 32-bit signed/unsigned pipelined multiplier," in *Fifth IEEE International Symposium on Electronic Design, Test & Applications*, 2010.

[25] R. D. K. A. A. S. V. P. Jayakrishnan, "Implementation of pipelined booth encoded wallace tree multiplier architecture," in *International Conference on Green Computing, Communication and Conservation of Energy (ICGCE)*, 2013.

Appendix I

Source Code

I.1 Floating Point Divider

```
module mvk_div (
                reset,
                clk,
                scan_in0,
                scan_en,
                test_mode,
                dividend,
                divisor,
                answer,
                overflow_flag,
                scan_out0
          );

input
      reset,                              // system reset
      clk;                                // system clock

input
      scan_in0,                           // test scan mode data input
      scan_en,                            // test scan mode enable
      test_mode;                          // test mode select
```

```

output
    scan_out0;                    // test scan mode data output

//inputs and outputs
input
    [31:0] dividend, divisor;

output
    [31:0] answer;

output reg
    overflow_flag;

reg
    stall_mc,
    stall_mc0,
    stall_mc1,
    stall_mc1_1,
    stall_mc2,
    stall_mc3,
    stall_mc4;

reg
    [11:0] yh_1,
           yh_0;

reg
    [10:0] yl_1,
           yl_0;

reg
    [11:0] address;

wire
    [29:0] lut_output;

wire
    [31:0] dividend_3,
           divisor_3;
reg
```

```
       [31:0] final_mult_1,
              final_mult_2,
              subtractor_input_1,
              subtractor_input_2,
              Ax_input_1,
              Ax_input_2,
              Ay_input_1,
              Ax1_input_2,
              Ay1_input_2,
              Ay_input_2,
              delay_in,
              dividend_0,
              divisor_0,
              dividend_1,
              divisor_1,
              dividend_2,
              divisor_2,
              dividend_4,
              divisor_4,
              a_input_1,
              a_input_2;

wire
       [31:0] final_output,
              subtractor_out_1,
              Ax_output,
              Ay_output,
              a_output,
              delay_out,
              sub_module_output;

mvk_stage_1 stage11(reset,clk, scan_in0, scan_en, test_mode,
    address, lut_output, scan_out0);     //Calculate value from LUT
mvk_subtractor subtract11(reset, clk, scan_in0, scan_en,
    test_mode, yh_1, yl_1, sub_module_output, scan_out0);

mvk_float_mult mult11(reset, clk, scan_in0, scan_en, test_mode,
    a_input_1, a_input_2, a_output, scan_out0);     //Calculate A

mvk_float_mult mult12(reset, clk, scan_in0, scan_en, test_mode,
    Ax_input_1, Ax_input_2, Ax_output, scan_out0);     //Calculate
```

```
AX
mvk_float_mult mult13(reset , clk , scan_in0 , scan_en , test_mode , Ay_input_1 , Ay_input_2 , Ay_output , scan_out0);    //Calculate AY

mvk_stage_2 stage21(reset ,clk , scan_in0 , scan_en , test_mode , delay_in , delay_out , scan_out0);

mvk_stage_2 stage22(reset ,clk , scan_in0 , scan_en , test_mode , dividend_2 , dividend_3 , scan_out0);

mvk_stage_2 stage23(reset ,clk , scan_in0 , scan_en , test_mode , divisor_2 , divisor_3 , scan_out0);

mvk_sub sub11(reset , clk , scan_in0 , scan_en , test_mode , scan_out0 , subtractor_input_1 , subtractor_input_2 , subtractor_out_1);

mvk_float_mult mult14(reset , clk , scan_in0 , scan_en , test_mode , final_mult_1 , final_mult_2 , final_output , scan_out0);    // Calculate AY

assign answer = (reset == 1'b1) ? 32'd0 : final_output;

always @(posedge clk or posedge reset)
begin
    if(reset)
    begin
        overflow_flag <= 1'b1;
        stall_mc <= 1'b1;
        stall_mc0 <= 1'b0;
        stall_mc1 <= 1'b0;
        stall_mc1_1 <= 1'b0;
        stall_mc2 <= 1'b0;
        stall_mc3 <= 1'b0;
        stall_mc4 <= 1'b0;
    end
    else
    begin
        if(stall_mc4 == 1'b1)    //We calculate (2-Ay)Ax which is the final Answer
        begin
```

```
            final_mult_1 <= subtractor_out_1;
            final_mult_2 <= delay_out;
        end

        if(stall_mc3 == 1'b1)     //We calculate 2-Ay
        begin
            subtractor_input_1 <= 32'h4000_0000;
            subtractor_input_2 <= Ay_output;
            delay_in <= Ax_output;
            stall_mc4 <= 1'b1;
        end

        if(stall_mc2 == 1'b1)     //We calculate Ax and Ay
        begin
            Ax_input_1 <= dividend_3;
            Ax_input_2 <= a_output;
            Ay_input_1 <= divisor_3;
            Ay_input_2 <= a_output;
            stall_mc3 <= 1'b1;
        end

        if(stall_mc1 == 1'b1)     //We calculate A
        begin
            a_input_1 <= {1'b0, lut_output};             //We call Multiplier Module
            a_input_2 <= sub_module_output;     //This is the second input to mult module
            dividend_2 <= dividend_1;
            divisor_2 <= divisor_1;
            stall_mc2 <= 1'b1;
        end

        if(stall_mc0 == 1'b1)     //In this stage we calculate (yh - yl) and also Get value from LUT
        begin
            yh_1 <= yh_0;         //Give input to yh - yl module
            yl_1 <= yl_0;
            stall_mc1 <= 1'b1;
            dividend_1 <= dividend_0;
            divisor_1 <= divisor_0;
        end
```

```

          if(stall_mc == 1'b1)     //In this stage we calculate (yh
             - yl) and also Get value from LUT
          begin
              dividend_0 <= dividend;
              divisor_0 <= divisor;
              yh_0 <= divisor[22:11];        //Give input to yh - yl
                  module
              yl_1 <= divisor[10:0];
              address <= divisor[22:11];     //This the address we
                 give to find 1/yh^2
              stall_mc0 <= 1'b1;
          end

      end
end

endmodule // mvk_div
```

I.2 Floating Point Multiplier

```
module mvk_float_mult (
           reset,
           clk,
           scan_in0,
           scan_en,
           test_mode,
           input_a,
           input_b,
           output_z,
           scan_out0
           );

input
    reset,                         // system reset
    clk;                           // system clock

input
```

```
      scan_in0,                        // test scan mode data input
      scan_en,                         // test scan mode enable
      test_mode;                       // test mode select

output
      scan_out0;                       // test scan mode data output

input
      [31:0]  input_a;

input
      [31:0]  input_b;

output
      [31:0]  output_z;

reg
      [3:0]  state;

reg
      [8:0]xyz;

parameter
      get_a             = 4'd0,
      get_b             = 4'd1,
      unpack            = 4'd2,
      check_nan_case    = 4'd3,
      normalise_a       = 4'd4,
      normalise_b       = 4'd5,
      multiply_0        = 4'd6,
      multiply_1        = 4'd7,
      normalise_1       = 4'd8,
      normalise_2       = 4'd9,
      round             = 4'd10,
      pack              = 4'd11,
      put_z             = 4'd12;

reg
      [31:0]  a, b, z;

reg
```

```
      [23:0] a_m_1, b_m_1, a_m_2, b_m_2, a_m_3, b_m_3;

reg
      [23:0] z_m_1, z_m_2, z_m_3, z_m_4, z_m_5;

reg
      [9:0] a_e_1, b_e_1, a_e_2, b_e_2, a_e_3, b_e_3;

reg
      [9:0] z_e_1, z_e_2, z_e_3, z_e_4, z_e_5;

reg
      a_s_1, b_s_1, a_s_2, b_s_2, a_s_3, b_s_3, z_s_1, z_s_2, z_s_3
          , z_s_4, z_s_5;

reg
      guard_2, round_bit_2, sticky_2, guard_3, round_bit_3,
          sticky_3, guard_4, round_bit_4, sticky_4;

reg
      [49:0] product;

wire
      product_guard,
      product_round,
      product_sticky;

reg
      [31:0] s_output_z;

reg
      [23:0]multiplicand, multiplier;

reg
      stall_mc,
      stall_mc0,
      stall_mc1,
      stall_mc2,
      stall_mc3,
      stall_mc4,
      stall_mc5,
```

```
    stall_mc6,
    stall_mc7,
    stall_mc8,
    stall_mc9;

reg
    special;

//reg
//      [49:0] product_1;

//mvk_mult mult11(reset, clk, scan_in0, scan_en, test_mode, multiplicand, multiplier, product, product_guard, product_round, product_sticky, scan_out0);

//mvk_mult mult11(reset, clk, scan_in0, scan_en, test_mode, multiplicand, multiplier, product, scan_out0);

assign output_z = s_output_z;


always @(posedge clk)
begin
    if (reset == 1)
    begin
        state <= unpack;
        xyz = 0;
        stall_mc <= 1'b0;
        stall_mc0 <= 1'b1;
        stall_mc1 <= 1'b1;
        stall_mc2 <= 1'b1;
        stall_mc3 <= 1'b1;
        stall_mc4 <= 1'b1;
        stall_mc5 <= 1'b1;
        stall_mc6 <= 1'b1;
        stall_mc7 <= 1'b1;
        stall_mc8 <= 1'b1;
        stall_mc9 <= 1'b1;
        a_m_1 <= 0;
        b_m_1 <= 0;
        a_m_2 <= 0;
```

```
            b_m_2 <= 0;
            a_m_3 <= 0;
            b_m_3 <= 0;
            a_e_1 <= 0;
            b_e_1 <= 0;
            a_e_2 <= 0;
            b_e_2 <= 0;
            a_e_3 <= 0;
            b_e_3 <= 0;
            a_s_1 <= 0;
            b_s_1 <= 0;
            a_s_2 <= 0;
            b_s_2 <= 0;
            a_s_3 <= 0;
            b_s_3 <= 0;
        end
        else
        begin
            if(stall_mc8 == 1'b0)
            begin
                s_output_z[22 : 0] <= z_m_5[22:0];
                s_output_z[30 : 23] <= z_e_5[7:0] + 127;
                s_output_z[31] <= z_s_5;
                if ($signed(z_e_5) == -126 && z_m_5[23] == 0)
                begin
                    s_output_z[30 : 23] <= 0;
                end
                //if overflow occurs, return inf
                if ($signed(z_e_5) > 127)
                begin
                    s_output_z[22 : 0] <= 0;
                    s_output_z[30 : 23] <= 255;
                    s_output_z[31] <= z_s_5;
                end
                stall_mc9 <= 1'b0;
            end     //End stall_mc8 block

            if(stall_mc7 == 1'b0)
            begin
/*
                if (guard_4 && (round_bit_4 | sticky_4 | z_m_4[0]))
```

```
                begin
                    z_m_5 <= z_m_4 + 1;
                    if (z_m_4 == 24'hffffff)
                    begin
                        z_e_5 <=z_e_4 + 1;
                    end
                    else
                    begin
                        z_e_5 <= z_e_4;
                    end
                    z_s_5 <= z_s_4;
                end
                else
                begin
*/
                    z_m_5 <= z_m_4;
                    z_e_5 <= z_e_4;
                    z_s_5 <= z_s_4;
//              end
//              state <= pack;
                stall_mc8 <= 1'b0;
            end     //End stall_mc7 block

            if(stall_mc6 == 1'b0)
            begin
                    if ($signed(z_e_3) < -126)
                    begin
                        xyz = -126 - $signed(z_e_3);
                        case(xyz)
                        1:
                        begin
                            z_e_4 <= z_e_3 + 1;
                            z_m_4 <= z_m_3 >> 1;
                            guard_4 <= z_m_3[0];
                            round_bit_4 <= guard_3;
                            sticky_4 <= sticky_3 | round_bit_3;
                        end
                        2:
                        begin
                            z_e_4 <= z_e_3 + 2;
                            z_m_4 <= z_m_3 >> 2;
```

```
                    guard_4 <= z_m_3[1];
                    round_bit_4 <= z_m_3[0];
                    sticky_4 <= sticky_3 | z_m_3[1];
                end
                3:
                begin
                    z_e_4 <= z_e_3 + 3;
                    z_m_4 <= z_m_3 >> 3;
                    guard_4 <= z_m_3[2];
                    round_bit_4 <= z_m_3[1];
                    sticky_4 <= sticky_3 | z_m_3[2];
                end
                4:
                begin
                    z_e_4 <= z_e_3 + 4;
                    z_m_4 <= z_m_3 >> 4;
                    guard_4 <= z_m_3[3];
                    round_bit_4 <= z_m_3[2];
                    sticky_4 <= sticky_3 | z_m_3[3];
                end
                5:
                begin
                    z_e_4 <= z_e_3 + 5;
                    z_m_4 <= z_m_3 >> 5;
                    guard_4 <= z_m_3[4];
                    round_bit_4 <= z_m_3[3];
                    sticky_4 <= sticky_3 | z_m_3[4];
                end
                6:
                begin
                    z_e_4 <= z_e_3 + 6;
                    z_m_4 <= z_m_3 >> 6;
                    guard_4 <= z_m_3[5];
                    round_bit_4 <= z_m_3[4];
                    sticky_4 <= sticky_3 | z_m_3[5];
                end
                7:
                begin
                    z_e_4 <= z_e_3 + 7;
                    z_m_4 <= z_m_3 >> 7;
                    guard_4 <= z_m_3[6];
```

```
                round_bit_4 <= z_m_3[5];
                sticky_4 <= sticky_3 | z_m_3[6];
            end
            8:
            begin
                z_e_4 <= z_e_3 + 8;
                z_m_4 <= z_m_3 >> 8;
                guard_4 <= z_m_3[7];
                round_bit_4 <= z_m_3[6];
                sticky_4 <= sticky_3 | z_m_3[7];
            end
            9:
            begin
                z_e_4 <= z_e_3 + 9;
                z_m_4 <= z_m_3 >> 9;
                guard_4 <= z_m_3[8];
                round_bit_4 <= z_m_3[7];
                sticky_4 <= sticky_3 | z_m_3[8];
            end
            10:
            begin
                z_e_4 <= z_e_3 + 10;
                z_m_4 <= z_m_3 >> 10;
                guard_4 <= z_m_3[9];
                round_bit_4 <= z_m_3[8];
                sticky_4 <= sticky_3 | z_m_3[9];
            end
            11:
            begin
                z_e_4 <= z_e_3 + 11;
                z_m_4 <= z_m_3 >> 11;
                guard_4 <= z_m_3[10];
                round_bit_4 <= z_m_3[9];
                sticky_4 <= sticky_3 | z_m_3[10];
            end
            12:
            begin
                z_e_4 <= z_e_3 + 12;
                z_m_4 <= z_m_3 >> 12;
                guard_4 <= z_m_3[11];
                round_bit_4 <= z_m_3[10];
```

```
                sticky_4 <= sticky_3 | z_m_3[11];
            end
            13:
            begin
                z_e_4 <= z_e_3 + 13;
                z_m_4 <= z_m_3 >> 13;
                guard_4 <= z_m_3[12];
                round_bit_4 <= z_m_3[11];
                sticky_4 <= sticky_3 | z_m_3[12];
            end
            14:
            begin
                z_e_4 <= z_e_3 + 14;
                z_m_4 <= z_m_3 >> 14;
                guard_4 <= z_m_3[13];
                round_bit_4 <= z_m_3[12];
                sticky_4 <= sticky_3 | z_m_3[13];
            end
            15:
            begin
                z_e_4 <= z_e_3 + 15;
                z_m_4 <= z_m_3 >> 15;
                guard_4 <= z_m_3[14];
                round_bit_4 <= z_m_3[13];
                sticky_4 <= sticky_3 | z_m_3[14];
            end
            16:
            begin
                z_e_4 <= z_e_3 + 16;
                z_m_4 <= z_m_3 >> 16;
                guard_4 <= z_m_3[15];
                round_bit_4 <= z_m_3[14];
                sticky_4 <= sticky_3 | z_m_3[15];
            end
            17:
            begin
                z_e_4 <= z_e_3 + 17;
                z_m_4 <= z_m_3 >> 17;
                guard_4 <= z_m_3[16];
                round_bit_4 <= z_m_3[15];
                sticky_4 <= sticky_3 | z_m_3[16];
```

```
            end
            18:
            begin
                z_e_4 <= z_e_3 + 18;
                z_m_4 <= z_m_3 >> 18;
                guard_4 <= z_m_3[17];
                round_bit_4 <= z_m_3[16];
                sticky_4 <= sticky_3 | z_m_3[17];
            end
            19:
            begin
                z_e_4 <= z_e_3 + 19;
                z_m_4 <= z_m_3 >> 19;
                guard_4 <= z_m_3[18];
                round_bit_4 <= z_m_3[17];
                sticky_4 <= sticky_3 | z_m_3[18];
            end
            20:
            begin
                z_e_4 <= z_e_3 + 20;
                z_m_4 <= z_m_3 >> 20;
                guard_4 <= z_m_3[19];
                round_bit_4 <= z_m_3[18];
                sticky_4 <= sticky_3 | z_m_3[19];
            end
            21:
            begin
                z_e_4 <= z_e_3 + 21;
                z_m_4 <= z_m_3 >> 21;
                guard_4 <= z_m_3[20];
                round_bit_4 <= z_m_3[19];
                sticky_4 <= sticky_3 | z_m_3[20];
            end
            22:
            begin
                z_e_4 <= z_e_3 + 22;
                z_m_4 <= z_m_3 >> 22;
                guard_4 <= z_m_3[21];
                round_bit_4 <= z_m_3[20];
                sticky_4 <= sticky_3 | z_m_3[21];
            end
```

```
                    23:
                    begin
                        z_e_4 <= z_e_3 + 23;
                        z_m_4 <= z_m_3 >> 23;
                        guard_4 <= z_m_3[22];
                        round_bit_4 <= z_m_3[21];
                        sticky_4 <= sticky_3 | z_m_3[22];
                    end
                    endcase
                end
                else
                begin
                    z_e_4 <= z_e_3;
                    z_m_4 <= z_m_3;
                    guard_4 <= guard_3;
                    round_bit_4 <= round_bit_3;
                    sticky_4 <= sticky_3;
                end
//                state <= round;
                z_s_4 <= z_s_3;
                stall_mc7 <= 1'b0;
        end     //End stall_mc6 block

        if(stall_mc5 == 1'b0)
        begin
                casex(z_m_2)
                    24'b1xxx_xxxx_xxxx_xxxx_xxxx_xxxx:
                    begin
//                          state <= normalise_2;
                        z_m_3 <= z_m_2;
                        z_e_3 <= z_e_2;
                        stall_mc6 <= 1'b0;
                    end
                    24'b01xx_xxxx_xxxx_xxxx_xxxx_xxxx:
                    begin
                        z_e_3 <= z_e_2 - 1;
                        z_m_3 <= z_m_2 << 1;
                        z_m_3[0] <= guard_2;
                        guard_3 <= round_bit_2;
                        round_bit_3 <= 0;
                    end
```

```
24'b001x_xxxx_xxxx_xxxx_xxxx_xxxx:
begin
    z_e_3 <= z_e_2 - 2;
    z_m_3 <= z_m_2 << 2;
    z_m_3[0] <= round_bit_2;
    guard_3 <= 0;
    round_bit_3 <= 0;
end
24'b0001_xxxx_xxxx_xxxx_xxxx_xxxx:
begin
    z_e_3 <= z_e_2 - 3;
    z_m_3 <= z_m_2 << 3;
    z_m_3[0] <= 0;
    guard_3 <= 0;
    round_bit_3 <= 0;
end
24'b0000_1xxx_xxxx_xxxx_xxxx_xxxx:
begin
    z_e_3 <= z_e_2 - 4;
    z_m_3 <= z_m_2 << 4;
    z_m_3[0] <= 0;
    guard_3 <= 0;
    round_bit_3 <= 0;
end
24'b0000_01xx_xxxx_xxxx_xxxx_xxxx:
begin
    z_e_3 <= z_e_2 - 5;
    z_m_3 <= z_m_2 << 5;
    z_m_3[0] <= 0;
    guard_3 <= 0;
    round_bit_3 <= 0;
end
24'b0000_001x_xxxx_xxxx_xxxx_xxxx:
begin
    z_e_3 <= z_e_2 - 6;
    z_m_3 <= z_m_2 << 6;
    z_m_3[0] <= 0;
    guard_3 <= 0;
    round_bit_3 <= 0;
end
24'b0000_0001_xxxx_xxxx_xxxx_xxxx:
```

```
                begin
                    z_e_3 <= z_e_2 - 7;
                    z_m_3 <= z_m_2 << 7;
                    z_m_3[0] <= 0;
                    guard_3 <= 0;
                    round_bit_3 <= 0;
                end
                24'b0000_0000_1xxx_xxxx_xxxx_xxxx:
                begin
                    z_e_3 <= z_e_2 - 8;
                    z_m_3 <= z_m_2 << 8;
                    z_m_3[0] <= 0;
                    guard_3 <= 0;
                    round_bit_3 <= 0;
                end
                24'b0000_0000_01xx_xxxx_xxxx_xxxx:
                begin
                    z_e_3 <= z_e_2 - 9;
                    z_m_3 <= z_m_2 << 9;
                    z_m_3[0] <= 0;
                    guard_3 <= 0;
                    round_bit_3 <= 0;
                end
                24'b0000_0000_001x_xxxx_xxxx_xxxx:
                begin
                    z_e_3 <= z_e_2 - 10;
                    z_m_3 <= z_m_2 << 10;
                    z_m_3[0] <= 0;
                    guard_3 <= 0;
                    round_bit_3 <= 0;
                end
                24'b0000_0000_0001_xxxx_xxxx_xxxx:
                begin
                    z_e_3 <= z_e_2 - 11;
                    z_m_3 <= z_m_2 << 11;
                    z_m_3[0] <= 0;
                    guard_3 <= 0;
                    round_bit_3 <= 0;
                end
                24'b0000_0000_0000_1xxx_xxxx_xxxx:
                begin
```

```
                        z_e_3 <= z_e_2 - 12;
                        z_m_3 <= z_m_2 << 12;
                        z_m_3[0] <= 0;
                        guard_3 <= 0;
                        round_bit_3 <= 0;
                    end
                    24'b0000_0000_0000_01xx_xxxx_xxxx:
                    begin
                        z_e_3 <= z_e_2 - 13;
                        z_m_3 <= z_m_2 << 13;
                        z_m_3[0] <= 0;
                        guard_3 <= 0;
                        round_bit_3 <= 0;
                    end
                    24'b0000_0000_0000_001x_xxxx_xxxx:
                    begin
                        z_e_3 <= z_e_2 - 14;
                        z_m_3 <= z_m_2 << 14;
                        z_m_3[0] <= 0;
                        guard_3 <= 0;
                        round_bit_3 <= 0;
                    end
                    24'b0000_0000_0000_0001_xxxx_xxxx:
                    begin
                        z_e_3 <= z_e_2 - 15;
                        z_m_3 <= z_m_2 << 15;
                        z_m_3[0] <= 0;
                        guard_3 <= 0;
                        round_bit_3 <= 0;
                    end
                    24'b0000_0000_0000_0000_1xxx_xxxx:
                    begin
                        z_e_3 <= z_e_2 - 16;
                        z_m_3 <= z_m_2 << 16;
                        z_m_3[0] <= 0;
                        guard_3 <= 0;
                        round_bit_3 <= 0;
                    end
                    24'b0000_0000_0000_0000_01xx_xxxx:
                    begin
                        z_e_3 <= z_e_2 - 17;
```

```
                z_m_3 <= z_m_2 << 17;
                z_m_3[0] <= 0;
                guard_3 <= 0;
                round_bit_3 <= 0;
            end
            24'b0000_0000_0000_0000_001x_xxxx:
            begin
                z_e_3 <= z_e_2 - 18;
                z_m_3 <= z_m_2 << 18;
                z_m_3[0] <= 0;
                guard_3 <= 0;
                round_bit_3 <= 0;
            end
            24'b0000_0000_0000_0000_0001_xxxx:
            begin
                z_e_3 <= z_e_2 - 19;
                z_m_3 <= z_m_2 << 19;
                z_m_3[0] <= 0;
                guard_3 <= 0;
                round_bit_3 <= 0;
            end
            24'b0000_0000_0000_0000_0000_1xxx:
            begin
                z_e_3 <= z_e_2 - 20;
                z_m_3 <= z_m_2 << 20;
                z_m_3[0] <= 0;
                guard_3 <= 0;
                round_bit_3 <= 0;
            end
            24'b0000_0000_0000_0000_0000_01xx:
            begin
                z_e_3 <= z_e_2 - 21;
                z_m_3 <= z_m_2 << 21;
                z_m_3[0] <= 0;
                guard_3 <= 0;
                round_bit_3 <= 0;
            end
            24'b0000_0000_0000_0000_0000_001x:
            begin
                z_e_3 <= z_e_2 - 22;
                z_m_3 <= z_m_2 << 22;
```

```
                        z_m_3[0] <= 0;
                        guard_3 <= 0;
                        round_bit_3 <= 0;
                    end
                    24'b0000_0000_0000_0000_0000_0001:
                    begin
                        z_e_3 <= z_e_2 - 23;
                        z_m_3 <= z_m_2 << 23;
                        z_m_3[0] <= 0;
                        guard_3 <= 0;
                        round_bit_3 <= 0;
                    end
                endcase
//                  state <= normalise_2;
                z_s_3 <= z_s_2;
                sticky_3 <= sticky_2;
                stall_mc6 <= 1'b0;
        end    //End stall_mc5 block

        if(stall_mc4 == 1'b0)
        begin
            z_m_2 <= product[49:26];
            guard_2 <= product[25];
            round_bit_2 <= product[24];
            sticky_2 <= (product[23:0] != 0);
            z_e_2 <= z_e_1;
            z_s_2 <= z_s_1;
//              state <= normalise_1;
            stall_mc5 <= 1'b0;
        end    //End stall_mc4 block

        if(stall_mc3 == 1'b0)
        begin
            z_s_1 <= a_s_3 ^ b_s_3;
            z_e_1 <= a_e_3 + b_e_3 + 1;
            product <= a_m_3 * b_m_3 * 4;
//              multiplicand <= a_m_3;
//              multiplier <= b_m_3;
//              state <= normalise_1;
            stall_mc4 <= 1'b0;
        end    //End stall_mc3 block
```

```

        if(stall_mc2 == 1'b0)
        begin
                casex(a_m_2)

                        24'b1xxx_xxxx_xxxx_xxxx_xxxx_xxxx:
                        begin
//                              state <= multiply_0;
                                a_m_3 <= a_m_2;
                                a_e_3 <= a_e_2;
                        end

                        24'b01xx_xxxx_xxxx_xxxx_xxxx_xxxx:
                        begin
                                a_m_3 <= a_m_2 << 1;
                                a_e_3 <= a_e_2 - 1;
                        end
                        24'b001x_xxxx_xxxx_xxxx_xxxx_xxxx:
                        begin
                                a_m_3 <= a_m_2 << 2;
                                a_e_3 <= a_e_2 - 2;
                        end
                        24'b0001_xxxx_xxxx_xxxx_xxxx_xxxx:
                        begin
                                a_m_3 <= a_m_2 << 3;
                                a_e_3 <= a_e_2 - 3;
                        end
                        24'b0000_1xxx_xxxx_xxxx_xxxx_xxxx:
                        begin
                                a_m_3 <= a_m_2 << 4;
                                a_e_3 <= a_e_2 - 4;
                        end
                        24'b0000_01xx_xxxx_xxxx_xxxx_xxxx:
                        begin
                                a_m_3 <= a_m_2 << 5;
                                a_e_3 <= a_e_2 - 5;
                        end
                        24'b0000_001x_xxxx_xxxx_xxxx_xxxx:
                        begin
                                a_m_3 <= a_m_2 << 6;
                                a_e_3 <= a_e_2 - 6;
```

```
            end
            24'b0000_0001_xxxx_xxxx_xxxx_xxxx:
            begin
                a_m_3 <= a_m_2 << 7;
                a_e_3 <= a_e_2 - 7;
            end
            24'b0000_0001_xxxx_xxxx_xxxx_xxxx:
            begin
                a_m_3 <= a_m_2 << 7;
                a_e_3 <= a_e_2 - 7;
            end
            24'b0000_0000_1xxx_xxxx_xxxx_xxxx:
            begin
                a_m_3 <= a_m_2 << 8;
                a_e_3 <= a_e_2 - 8;
            end
            24'b0000_0000_01xx_xxxx_xxxx_xxxx:
            begin
                a_m_3 <= a_m_2 << 9;
                a_e_3 <= a_e_2 - 9;
            end
            24'b0000_0000_001x_xxxx_xxxx_xxxx:
            begin
                a_m_3 <= a_m_2 << 10;
                a_e_3 <= a_e_2 - 10;
            end
            24'b0000_0000_0001_xxxx_xxxx_xxxx:
            begin
                a_m_3 <= a_m_2 << 11;
                a_e_3 <= a_e_2 - 11;
            end
            24'b0000_0000_0000_1xxx_xxxx_xxxx:
            begin
                a_m_3 <= a_m_2 << 12;
                a_e_3 <= a_e_2 - 12;
            end
            24'b0000_0000_0000_01xx_xxxx_xxxx:
            begin
                a_m_3 <= a_m_2 << 13;
                a_e_3 <= a_e_2 - 13;
            end
```

```
        24'b0000_0000_0000_001x_xxxx_xxxx:
        begin
            a_m_3 <= a_m_2 << 14;
            a_e_3 <= a_e_2 - 14;
        end
        24'b0000_0000_0000_0001_xxxx_xxxx:
        begin
            a_m_3 <= a_m_2 << 15;
            a_e_3 <= a_e_2 - 15;
        end
        24'b0000_0000_0000_0000_1xxx_xxxx:
        begin
            a_m_3 <= a_m_2 << 16;
            a_e_3 <= a_e_2 - 16;
        end
        24'b0000_0000_0000_0000_01xx_xxxx:
        begin
            a_m_3 <= a_m_2 << 17;
            a_e_3 <= a_e_2 - 17;
        end
        24'b0000_0000_0000_0000_001x_xxxx:
        begin
            a_m_3 <= a_m_2 << 18;
            a_e_3 <= a_e_2 - 18;
        end
        24'b0000_0000_0000_0000_0001_xxxx:
        begin
            a_m_3 <= a_m_2 << 19;
            a_e_3 <= a_e_2 - 19;
        end
        24'b0000_0000_0000_0000_0000_1xxx:
        begin
            a_m_3 <= a_m_2 << 20;
            a_e_3 <= a_e_2 - 20;
        end
        24'b0000_0000_0000_0000_0000_01xx:
        begin
            a_m_3 <= a_m_2 << 21;
            a_e_3 <= a_e_2 - 21;
        end
        24'b0000_0000_0000_0000_0000_001x:
```

```
                    begin
                        a_m_3 <= a_m_2 << 22;
                        a_e_3 <= a_e_2 - 22;
                    end
                    24'b0000_0000_0000_0000_0000_0001:
                    begin
                        a_m_3 <= a_m_2 << 23;
                        a_e_3 <= a_e_2 - 23;
                    end
                endcase

/*
                if (b_m[23])
                begin
                    state <= multiply_0;
                end
                else
                begin
                    b_m <= b_m << 1;
                    b_e <= b_e - 1;
                end
*/
                casex(b_m_2)

                    24'b1xxx_xxxx_xxxx_xxxx_xxxx_xxxx:
                    begin
//                          state <= multiply_0;
                        b_m_3 <= b_m_2;
                        b_e_3 <= b_e_2;
                    end

                    24'b01xx_xxxx_xxxx_xxxx_xxxx_xxxx:
                    begin
                        b_m_3 <= b_m_2 << 1;
                        a_e_3 <= a_e_2 - 1;
                    end
                    24'b001x_xxxx_xxxx_xxxx_xxxx_xxxx:
                    begin
                        b_m_3 <= b_m_2 << 2;
                        a_e_3 <= a_e_2 - 2;
                    end
```

```
24'b0001_xxxx_xxxx_xxxx_xxxx_xxxx:
begin
    b_m_3 <= b_m_2 << 3;
    a_e_3 <= a_e_2 - 3;
end
24'b0000_1xxx_xxxx_xxxx_xxxx_xxxx:
begin
    b_m_3 <= b_m_2 << 4;
    a_e_3 <= a_e_2 - 4;
end
24'b0000_01xx_xxxx_xxxx_xxxx_xxxx:
begin
    b_m_3 <= b_m_2 << 5;
    a_e_3 <= a_e_2 - 5;
end
24'b0000_001x_xxxx_xxxx_xxxx_xxxx:
begin
    b_m_3 <= b_m_2 << 6;
    a_e_3 <= a_e_2 - 6;
end
24'b0000_0001_xxxx_xxxx_xxxx_xxxx:
begin
    b_m_3 <= b_m_2 << 7;
    a_e_3 <= a_e_2 - 7;
end
24'b0000_0001_xxxx_xxxx_xxxx_xxxx:
begin
    b_m_3 <= b_m_2 << 7;
    a_e_3 <= a_e_2 - 7;
end
24'b0000_0000_1xxx_xxxx_xxxx_xxxx:
begin
    b_m_3 <= b_m_2 << 8;
    a_e_3 <= a_e_2 - 8;
end
24'b0000_0000_01xx_xxxx_xxxx_xxxx:
begin
    b_m_3 <= b_m_2 << 9;
    a_e_3 <= a_e_2 - 9;
end
24'b0000_0000_001x_xxxx_xxxx_xxxx:
```

```
begin
    b_m_3 <= b_m_2 << 10;
    a_e_3 <= a_e_2 - 10;
end
24'b0000_0000_0001_xxxx_xxxx_xxxx:
begin
    b_m_3 <= b_m_2 << 11;
    a_e_3 <= a_e_2 - 11;
end
24'b0000_0000_0000_1xxx_xxxx_xxxx:
begin
    b_m_3 <= b_m_2 << 12;
    a_e_3 <= a_e_2 - 12;
end
24'b0000_0000_0000_01xx_xxxx_xxxx:
begin
    b_m_3 <= b_m_2 << 13;
    a_e_3 <= a_e_2 - 13;
end
24'b0000_0000_0000_001x_xxxx_xxxx:
begin
    b_m_3 <= b_m_2 << 14;
    a_e_3 <= a_e_2 - 14;
end
24'b0000_0000_0000_0001_xxxx_xxxx:
begin
    b_m_3 <= b_m_2 << 15;
    a_e_3 <= a_e_2 - 15;
end
24'b0000_0000_0000_0000_1xxx_xxxx:
begin
    b_m_3 <= b_m_2 << 16;
    a_e_3 <= a_e_2 - 16;
end
24'b0000_0000_0000_0000_01xx_xxxx:
begin
    b_m_3 <= b_m_2 << 17;
    a_e_3 <= a_e_2 - 17;
end
24'b0000_0000_0000_0000_001x_xxxx:
begin
```

```
                        b_m_3 <= b_m_2 << 18;
                        a_e_3 <= a_e_2 - 18;
                    end
                    24'b0000_0000_0000_0000_0001_xxxx:
                    begin
                        b_m_3 <= b_m_2 << 19;
                        a_e_3 <= a_e_2 - 19;
                    end
                    24'b0000_0000_0000_0000_0000_1xxx:
                    begin
                        b_m_3 <= b_m_2 << 20;
                        a_e_3 <= a_e_2 - 20;
                    end
                    24'b0000_0000_0000_0000_0000_01xx:
                    begin
                        b_m_3 <= b_m_2 << 21;
                        a_e_3 <= a_e_2 - 21;
                    end
                    24'b0000_0000_0000_0000_0000_001x:
                    begin
                        b_m_3 <= b_m_2 << 22;
                        a_e_3 <= a_e_2 - 22;
                    end
                    24'b0000_0000_0000_0000_0000_0001:
                    begin
                        b_m_3 <= b_m_2 << 23;
                        a_e_3 <= a_e_2 - 23;
                    end
                endcase
//                state <= multiply_0;
                a_s_3 <= a_s_2;
                b_s_3 <= b_s_2;
                stall_mc3 <= 1'b0;

        end     //End stall_mc2 block

        if(stall_mc1 == 1'b0)
        begin
            if ((a_e_1 == 128 && a_m_1 != 0) || (b_e_1 == 128 &&
              b_m_1 != 0))
            begin
```

```
                    z[31] <= 1;
                    z[30:23] <= 255;
                    z[22] <= 1;
                    z[21:0] <= 0;
//                    state <= put_z;
                    stall_mc2 <= 1'b0;
                    special <= 1'b1;
                    //if a is inf return inf
                end
                else if (a_e_1 == 128)
                begin
                    z[31] <= a_s_1 ^ b_s_1;
                    z[30:23] <= 255;
                    z[22:0] <= 0;
//                    state <= put_z;
                    stall_mc2 <= 1'b0;
                    special <= 1'b1;
                    //if b is zero return NaN
                    if ($signed(b_e_1 == -127) && (b_m_1 == 0))
                    begin
                        z[31] <= 1;
                        z[30:23] <= 255;
                        z[22] <= 1;
                        z[21:0] <= 0;
//                        state <= put_z;
                    special <= 1'b1;
                    stall_mc2 <= 1'b0;
                    end
                    //if b is inf return inf
                end
                else if (b_e_1 == 128)
                begin
                    z[31] <= a_s_1 ^ b_s_1;
                    z[30:23] <= 255;
                    z[22:0] <= 0;
//                    state <= put_z;
                    special <= 1'b1;
                    stall_mc2 <= 1'b0;
                //if a is zero return zero
                end
                else if (($signed(a_e_1) == -127) && (a_m_1 == 0))
```

```
        begin
            z[31] <= a_s_1 ^ b_s_1;
            z[30:23] <= 0;
            z[22:0] <= 0;
//            state <= put_z;
            special <= 1'b1;
            stall_mc2 <= 1'b0;
        //if b is zero return zero
        end
        else if (($signed(b_e_1) == -127) && (b_m_1 == 0))
        begin
            z[31] <= a_s_1 ^ b_s_1;
            z[30:23] <= 0;
            z[22:0] <= 0;
//            state <= put_z;
            special <= 1'b1;
            stall_mc2 <= 1'b0;
        end
        else
        begin
        //Denormalised Number
            if ($signed(a_e_1) == -127)
            begin
                a_e_2 <= -126;
                a_m_2 <= a_m_1;
                a_s_2 <= a_s_1;
            end
            else
            begin
                a_m_2[23] <= 1;
                a_m_2[22:0] <= a_m_1[22:0];
                a_e_2 <= a_e_1;
                a_s_2 <= a_s_1;
            end
            //Denormalised Number
            if ($signed(b_e_1) == -127)
            begin
                b_e_2 <= -126;
                b_m_2 <= b_m_1;
                b_s_2 <= b_s_1;
            end
```

```
                else
                begin
                    b_m_2[23] <= 1;
                    b_m_2[22:0] <= b_m_1[22:0];
                    b_e_2 <= b_e_1;
                    b_s_2 <= b_s_1;
                end
//                  state <= normalise_a;
                special <= 1'b0;
                stall_mc2 <= 1'b0;
            end
        end     //End stall_mc1 block

        if(stall_mc0 == 1'b0)
        begin
            a_m_1 <= input_a[22 : 0];
            b_m_1 <= input_b[22 : 0];
//              a_e_1 <= input_a[30 : 23] - 127;
//              b_e_1 <= input_b[30 : 23] - 127;
            a_s_1 <= input_a[31];
            b_s_1 <= input_b[31];
            stall_mc1 <= 1'b0;
        end     //End stall_mc0 block

        if(stall_mc == 1'b0)
        begin
            a_e_1 <= input_a[30 : 23] - 127;
            b_e_1 <= input_b[30 : 23] - 127;
            stall_mc0 <= 1'b0;
        end     //End stall_mc0 block


    end     //End Else block of reset
end     //End always block

endmodule
```

I.3 Wallace Tree Multiplier

```
module mvk_mult (
            reset,
            clk,
            scan_in0,
            scan_en,
            test_mode,
            multiplicand,
            multiplier,
            product,
            scan_out0
        );

input
    reset,                          // system reset
    clk;                            // system clock

input
    scan_in0,                       // test scan mode data input
    scan_en,                        // test scan mode enable
    test_mode;                      // test mode select

output
    scan_out0;                      // test scan mode data output

//inputs and outputs
input
    [23:0] multiplicand, multiplier;

output
    [49:0] product;

wire S11,C11, S12,C12, S13,C13, S114,C114, S115,C115, S116,C116,
   S117,C117, S118,C118, S119,C119, S1110,C1110_inst, S1111,
   C1111_inst, S1112,C1112_inst, S1113,C1113_inst, S1114,
   C1114_inst, S1115,C1115_inst, S1116,C1116_inst, S1117,
   C1117_inst, S1118,C1118_inst, S1119,C1119_inst, S1120,
   C1120_inst, S1121,C1121_inst, S1122,C1122_inst, S1123,
   C1123_inst, S1124,C1124_inst, S1125,C1125_inst, S1126,
```

```
    C1126_inst, S1127,C1127_inst, S1128,C1128_inst, S1129,
    C1129_inst, S1130,C1130_inst, S1131,C1131_inst, S1132,
    C1132_inst, S1133,C1133_inst, S1134,C1134_inst, S1135,
    C1135_inst, S1136,C1136_inst, S1137,C1137_inst, S1138,
    C1138_inst, S1139,C1139_inst, S1140,C1140_inst, S1141,
    C1141_inst, S1142,C1142_inst, S1143,C1143_inst;    //Contains
    all the Sum's partial product

wire S124,C124, S125,C125, S126,C126, S127,C127, S128,C128, S129,
    C129, S1210,C1210_inst, S1211,C1211_inst, S1212,C1212_inst,
    S1213,C1213_inst, S1214,C1214_inst, S1215,C1215_inst, S1216,
    C1216_inst, S1217,C1217_inst, S1218,C1218_inst, S1219,
    C1219_inst, S1220,C1220_inst, S1221,C1221_inst, S1222,
    C1222_inst, S1223,C1223_inst, S1224,C1224_inst, S1225,
    C1225_inst, S1226,C1226_inst, S1227,C1227_inst, S1228,
    C1228_inst, S1229,C1229_inst, S1230,C1230_inst, S1231,
    C1231_inst, S1232,C1232_inst, S1233,C1233_inst, S1234,
    C1234_inst, S1235,C1235_inst, S1236,C1236_inst, S1237,
    C1237_inst, S1238,C1238_inst, S1239,C1239_inst, S1240,
    C1240_inst;

wire S137,C137, S138,C138, S139,C139, S1310,C1310_inst, S1311,
    C1311_inst, S1312,C1312_inst, S1313,C1313_inst, S1314,
    C1314_inst, S1315,C1315_inst, S1316,C1316_inst, S1317,
    C1317_inst, S1318,C1318_inst, S1319,C1319_inst, S1320,
    C1320_inst, S1321,C1321_inst, S1322,C1322_inst, S1323,
    C1323_inst, S1324,C1324_inst, S1325,C1325_inst, S1326,
    C1326_inst, S1327,C1327_inst, S1328,C1328_inst, S1329,
    C1329_inst, S1330,C1330_inst, S1331,C1331_inst, S1332,
    C1332_inst, S1333,C1333_inst, S1334,C1334_inst, S1335,
    C1335_inst, S1336,C1336_inst, S1337,C1337_inst;

wire S1410,C1410_inst, S1411,C1411_inst, S1412,C1412_inst, S1413,
    C1413_inst, S1414,C1414_inst, S1415,C1415_inst, S1416,
    C1416_inst, S1417,C1417_inst, S1418,C1418_inst, S1419,
    C1419_inst, S1420,C1420_inst, S1421,C1421_inst, S1422,
    C1422_inst, S1423,C1423_inst, S1424,C1424_inst, S1425,
    C1425_inst, S1426,C1426_inst, S1427,C1427_inst, S1428,
    C1428_inst, S1429,C1429_inst, S1430,C1430_inst, S1431,
    C1431_inst, S1432,C1432_inst, S1433,C1433_inst, S1434,
    C1434_inst;
```

```

wire S1513,C1513_inst, S1514,C1514_inst, S1515,C1515_inst, S1516,
    C1516_inst, S1517,C1517_inst, S1518,C1518_inst, S1519,
    C1519_inst, S1520,C1520_inst, S1521,C1521_inst, S1522,
    C1522_inst, S1523,C1523_inst, S1524,C1524_inst, S1525,
    C1525_inst, S1526,C1526_inst, S1527,C1527_inst, S1528,
    C1528_inst, S1529,C1529_inst, S1530,C1530_inst, S1531,
    C1531_inst;

wire S1616,C1616_inst, S1617,C1617_inst, S1618,C1618_inst, S1619,
    C1619_inst, S1620,C1620_inst, S1621,C1621_inst, S1622,
    C1622_inst, S1623,C1623_inst, S1624,C1624_inst, S1625,
    C1625_inst, S1626,C1626_inst, S1627,C1627_inst, S1628,
    C1628_inst;

wire S1719,C1719_inst, S1720,C1720_inst, S1721,C1721_inst, S1722,
    C1722_inst, S1723,C1723_inst, S1724,C1724_inst, S1725,
    C1725_inst, S1822,C1822_inst;

wire S22,C22, S23,C23, S24,C24, S25,C25, S216,C216, S217,C217,
    S218,C218, S219,C219, S2110,C2110_inst, S2111,C2111_inst,
    S2112,C2112_inst, S2113,C2113_inst, S2114,C2114_inst, S2115,
    C2115_inst, S2116,C2116_inst, S2117,C2117_inst, S2118,
    C2118_inst, S2119,C2119_inst, S2120,C2120_inst, S2121,
    C2121_inst, S2122,C2122_inst, S2123,C2123_inst, S2124,
    C2124_inst, S2125,C2125_inst, S2126,C2126_inst, S2127,
    C2127_inst, S2128,C2128_inst, S2129,C2129_inst, S2130,
    C2130_inst, S2131,C2131_inst, S2132,C2132_inst, S2133,
    C2133_inst, S2134,C2134_inst, S2135,C2135_inst, S2136,
    C2136_inst, S2137,C2137_inst, S2138,C2138_inst, S2139,
    C2139_inst, S2140,C2140_inst, S2141,C2141_inst;

wire S226,C226, S227,C227, S228,C228, S229,C229, S2210,C2210_inst
    , S2211,C2211_inst, S2212,C2212_inst, S2213,C2213_inst, S2214,
    C2214_inst, S2215,C2215_inst, S2216,C2216_inst, S2217,
    C2217_inst, S2218,C2218_inst, S2219,C2219_inst, S2220,
    C2220_inst, S2221,C2221_inst, S2222,C2222_inst, S2223,
    C2223_inst, S2224,C2224_inst, S2225,C2225_inst, S2226,
    C2226_inst, S2227,C2227_inst, S2228,C2228_inst, S2229,
    C2229_inst, S2230,C2230_inst, S2231,C2231_inst, S2232,
    C2232_inst, S2233,C2233_inst, S2234,C2234_inst, S2235,
```

```
    C2235_inst, S2236,C2236_inst, S2237,C2237_inst, S2238,
    C2238_inst;

wire S2311,C2311_inst, S2312,C2312_inst, S2313,C2313_inst, S2314,
    C2314_inst, S2315,C2315_inst, S2316,C2316_inst, S2317,
    C2317_inst, S2318,C2318_inst, S2319,C2319_inst, S2320,
    C2320_inst, S2321,C2321_inst, S2322,C2322_inst, S2323,
    C2323_inst, S2324,C2324_inst, S2325,C2325_inst, S2326,
    C2326_inst, S2327,C2327_inst, S2328,C2328_inst, S2329,
    C2329_inst, S2330,C2330_inst, S2331,C2331_inst, S2332,
    C2332_inst, S2333,C2333_inst, S2334,C2334_inst, S2335,
    C2335_inst;

wire S2415,C2415_inst, S2416,C2416_inst, S2417,C2417_inst, S2418,
    C2418_inst, S2419,C2419_inst, S2420,C2420_inst, S2421,
    C2421_inst, S2422,C2422_inst, S2423,C2423_inst, S2424,
    C2424_inst, S2425,C2425_inst, S2426,C2426_inst, S2427,
    C2427_inst, S2428,C2428_inst, S2429,C2429_inst;

wire S2520,C2520_inst, S2521,C2521_inst, S2522,C2522_inst, S2523,
    C2523_inst, S2524,C2524_inst, S2525,C2525_inst, S2526,
    C2526_inst;

wire S33,C33, S35,C35, S36,C36, S37,C37, S38,C38, S319,C319,
    S3110,C3110_inst, S3111,C3111_inst, S3112,C3112_inst, S3113,
    C3113_inst, S3114,C3114_inst, S3115,C3115_inst, S3116,
    C3116_inst, S3117,C3117_inst, S3118,C3118_inst, S3119,
    C3119_inst, S3120,C3120_inst, S3121,C3121_inst, S3122,
    C3122_inst, S3123,C3123_inst, S3124,C3124_inst, S3125,
    C3125_inst, S3126,C3126_inst, S3127,C3127_inst, S3128,
    C3128_inst, S3129,C3129_inst, S3130,C3130_inst, S3131,
    C3131_inst, S3132,C3132_inst, S3133,C3133_inst, S3134,
    C3134_inst, S3135,C3135_inst, S3136,C3136_inst, S3137,
    C3137_inst, S3138,C3138_inst, S3139,C3139_inst, S3140,
    C3140_inst, S3141,C3141_inst, S3142,C3142_inst, S3143,
    C3143_inst, S3144,C3144_inst;

wire S329,C329, S3210,C3210_inst, S3211,C3211_inst, S3212,
    C3212_inst, S3213,C3213_inst, S3214,C3214_inst, S3215,
    C3215_inst, S3216,C3216_inst, S3217,C3217_inst, S3218,
    C3218_inst, S3219,C3219_inst, S3220,C3220_inst, S3221,
```

```
    C3221_inst, S3222,C3222_inst, S3223,C3223_inst, S3224,
    C3224_inst, S3225,C3225_inst, S3226,C3226_inst, S3227,
    C3227_inst, S3228,C3228_inst, S3229,C3229_inst, S3230,
    C3230_inst, S3231,C3231_inst, S3232,C3232_inst, S3233,
    C3233_inst, S3234,C3234_inst, S3235,C3235_inst, S3236,
    C3236_inst;

wire S3316,C3316_inst, S3317,C3317_inst, S3318,C3318_inst, S3319,
    C3319_inst, S3320,C3320_inst, S3321,C3321_inst, S3322,
    C3322_inst, S3323,C3323_inst, S3324,C3324_inst, S3325,
    C3325_inst, S3326,C3326_inst, S3327,C3327_inst, S3328,
    C3328_inst, S3329,C3329_inst, S3330,C3330_inst, S3423,
    C3423_inst;

wire S44,C44, S45,C45, S46,C46, S47,C47, S48,C48, S49,C49, S410,
    C410, S411,C411, S412,C412, S413,C413, S4114,C4114_inst, S4115
    ,C4115_inst, S4116,C4116_inst, S4117,C4117_inst, S4118,
    C4118_inst, S4119,C4119_inst, S4120,C4120_inst, S4121,
    C4121_inst, S4122,C4122_inst, S4123,C4123_inst, S4124,
    C4124_inst, S4125,C4125_inst, S4126,C4126_inst, S4127,
    C4127_inst, S4128,C4128_inst, S4129,C4129_inst, S4130,
    C4130_inst, S4131,C4131_inst, S4132,C4132_inst, S4133,
    C4133_inst, S4134,C4134_inst, S4135,C4135_inst, S4136,
    C4136_inst, S4137,C4137_inst, S4138,C4138_inst, S4139,
    C4139_inst;

wire S4214,C4214_inst, S4215,C4215_inst, S4216,C4216_inst, S4217,
    C4217_inst, S4218,C4218_inst, S4219,C4219_inst, S4220,
    C4220_inst, S4221,C4221_inst, S4222,C4222_inst, S4223,
    C4223_inst, S4224,C4224_inst, S4225,C4225_inst, S4226,
    C4226_inst, S4227,C4227_inst, S4228,C4228_inst, S4229,
    C4229_inst, S4230,C4230_inst, S4231,C4231_inst, S4232,
    C4232_inst, S4324,C4324_inst;

wire S57,C57, S58,C58, S59,C59, S510,C510, S511,C511, S512,C512,
    S513,C513, S514,C514, S515,C515, S516,C516, S517,C517, S518,
    C518, S519,C519, S520,C520, S5121,C5121_inst, S5122,C5122_inst
    , S5123,C5123_inst, S5124,C5124_inst, S5125,C5125_inst, S5126,
    C5126_inst, S5127,C5127_inst, S5128,C5128_inst, S5129,
    C5129_inst, S5130,C5130_inst, S5131,C5131_inst, S5132,
    C5132_inst, S5133,C5133_inst, S5134,C5134_inst, S5135,
```

```
    C5135_inst, S5136,C5136_inst, S5137,C5137_inst;

wire S5221,C5221_inst, S5222,C5222_inst, S5223,C5223_inst, S5224,
    C5224_inst, S5225,C5225_inst, S5226,C5226_inst;

wire S610,C610, S611,C611, S612,C612, S613,C613, S614,C614, S615,
    C615, S616,C616, S617,C617, S618,C618, S619,C619, S620,C620,
    S621,C621, S622,C622, S623,C623, S624,C624, S625,C625, S626,
    C626, S627,C627, S628,C628, S629,C629, S630,C630, S631,C631,
    S632,C632, S633,C633, S634,C634, S635,C635, S636,C636, S637,
    C637, S638,C638, S639,C639, S640,C640, S641,C641, S642,C642,
    S643,C643, S644,C644;

wire S715,C715, S716,C716, S717,C717, S718,C718, S719,C719, S720,
    C720, S721,C721, S722,C722, S723,C723, S724,C724, S725,C725,
    S726,C726, S727,C727, S728,C728, S729,C729, S730,C730, S731,
    C731, S732,C732, S733,C733, S734,C734, S735,C735, S736,C736,
    S737,C737, S738,C738, S739,C739, S740,C740, S741,C741, S742,
    C742, S743,C743, S744,C744, S745,C745;

wire S822,C822, S823,C823, S824,C824, S825,C825, S826,C826, S827,
    C827, S828,C828, S829,C829, S830,C830, S831,C831, S832,C832,
    S833,C833, S834,C834, S835,C835,  S836,C836, S837,C837, S838,
    C838, S839,C839, S840,C840, S841,C841, S842,C842, S843,C843,
    S844,C844, S845,C845;

wire S923,C923_inst, S924,C924_inst, S925,C925_inst, S926,
    C926_inst, S927,C927_inst, S928,C928_inst, S929,C929_inst,
    S930,C930_inst, S931,C931_inst, S932,C932_inst, S933,C933_inst
    , S934,C934_inst, S935,C935_inst, S936,C936_inst, S937,
    C937_inst, S938,C938_inst, S939,C939_inst, S940,C940_inst,
    S941,C941_inst, S942,C942_inst, S943,C943_inst, S944,C944_inst
    , S945,C945_inst, S946,C946_inst;

wire S1024,C1024_inst, S1025,C1025_inst, S1026,C1026_inst, S1027,
    C1027_inst, S1028,C1028_inst, S1029,C1029_inst, S1030,
    C1030_inst, S1031,C1031_inst, S1032,C1032_inst, S1033,
    C1033_inst, S1034,C1034_inst, S1035,C1035_inst, S1036,
    C1036_inst, S1037,C1037_inst, S1038,C1038_inst, S1039,
    C1039_inst, S1040,C1040_inst, S1041,C1041_inst, S1042,
    C1042_inst, S1043,C1043_inst, S1044,C1044_inst, S1045,
```

```
    C1045_inst, S1046,C1046_inst, S1047,C1047_inst;

wire [47:0]P0, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12,
     P13, P14, P15, P16, P17, P18, P19, P20, P21, P22, P23;

wire sign;
wire [7:0]exponent,
          exponent_1;

wire [7:0]multi_exponent;
wire [7:0]multiplier_exponent;

wire sum0, sum1, sum2, sum3, sum4, sum5, sum6, sum7;
wire carry0, carry1, carry2, carry3, carry4, carry5, carry6,
    carry7;

wire
     [49:0]product_2;

assign product = (reset == 1'b1) ? 48'b0 : {S1047, S1046, S1045,
    S1044, S1043, S1042, S1041, S1040, S1039, S1038, S1037, S1036,
     S1035, S1034, S1033, S1032, S1031, S1030, S1029, S1028, S1027
    , S1026, S1025, S1024, S923, S822, S721, S720, S719, S718,
    S717, S716, S715, S614, S613, S612, S611, S610, S59, S58, S57,
     S46, S45, S44, S33, S22, S11, P0[0]};

assign P0 = {24'd0, multiplicand[23] & multiplier[0],
    multiplicand[22] & multiplier[0], multiplicand[21] &
    multiplier[0], multiplicand[20] & multiplier[0], multiplicand
    [19] & multiplier[0], multiplicand[18] & multiplier[0],
    multiplicand[17] & multiplier[0], multiplicand[16] &
    multiplier[0], multiplicand[15] & multiplier[0], multiplicand
    [14] & multiplier[0], multiplicand[13] & multiplier[0],
    multiplicand[12] & multiplier[0], multiplicand[11] &
    multiplier[0], multiplicand[10] & multiplier[0], multiplicand
    [9] & multiplier[0], multiplicand[8] & multiplier[0],
    multiplicand[7] & multiplier[0], multiplicand[6] & multiplier
    [0], multiplicand[5] & multiplier[0], multiplicand[4] &
    multiplier[0], multiplicand[3] & multiplier[0], multiplicand
    [2] & multiplier[0], multiplicand[1] & multiplier[0],
    multiplicand[0] & multiplier[0]};
```

```

assign P1 = {23'd0, multiplicand[23] & multiplier[1], multiplicand[22] & multiplier[1], multiplicand[21] & multiplier[1], multiplicand[20] & multiplier[1], multiplicand[19] & multiplier[1], multiplicand[18] & multiplier[1], multiplicand[17] & multiplier[1], multiplicand[16] & multiplier[1], multiplicand[15] & multiplier[1], multiplicand[14] & multiplier[1], multiplicand[13] & multiplier[1], multiplicand[12] & multiplier[1], multiplicand[11] & multiplier[1], multiplicand[10] & multiplier[1], multiplicand[9] & multiplier[1], multiplicand[8] & multiplier[1], multiplicand[7] & multiplier[1], multiplicand[6] & multiplier[1], multiplicand[5] & multiplier[1], multiplicand[4] & multiplier[1], multiplicand[3] & multiplier[1], multiplicand[2] & multiplier[1], multiplicand[1] & multiplier[1], multiplicand[0] & multiplier[1], 1'b0};

assign P2 = {22'd0, multiplicand[23] & multiplier[2], multiplicand[22] & multiplier[2], multiplicand[21] & multiplier[2], multiplicand[20] & multiplier[2], multiplicand[19] & multiplier[2], multiplicand[18] & multiplier[2], multiplicand[17] & multiplier[2], multiplicand[16] & multiplier[2], multiplicand[15] & multiplier[2], multiplicand[14] & multiplier[2], multiplicand[13] & multiplier[2], multiplicand[12] & multiplier[2], multiplicand[11] & multiplier[2], multiplicand[10] & multiplier[2], multiplicand[9] & multiplier[2], multiplicand[8] & multiplier[2], multiplicand[7] & multiplier[2], multiplicand[6] & multiplier[2], multiplicand[5] & multiplier[2], multiplicand[4] & multiplier[2], multiplicand[3] & multiplier[2], multiplicand[2] & multiplier[2], multiplicand[1] & multiplier[2], multiplicand[0] & multiplier[2], 1'b0, 1'b0};

assign P3 = {21'd0, multiplicand[23] & multiplier[3], multiplicand[22] & multiplier[3], multiplicand[21] & multiplier[3], multiplicand[20] & multiplier[3], multiplicand[19] & multiplier[3], multiplicand[18] & multiplier[3], multiplicand[17] & multiplier[3], multiplicand[16] & multiplier[3], multiplicand[15] & multiplier[3], multiplicand[14] & multiplier[3], multiplicand[13] & multiplier[3], multiplicand[12] & multiplier[3], multiplicand[11] &
```

```
    multiplier[3], multiplicand[10] & multiplier[3], multiplicand
    [9] & multiplier[3], multiplicand[8] & multiplier[3],
    multiplicand[7] & multiplier[3], multiplicand[6] & multiplier
    [3], multiplicand[5] & multiplier[3], multiplicand[4] &
    multiplier[3], multiplicand[3] & multiplier[3], multiplicand
    [2] & multiplier[3], multiplicand[1] & multiplier[3],
    multiplicand[0] & multiplier[3], 3'd0};

assign P4 = {20'd0, multiplicand[23] & multiplier[4],
    multiplicand[22] & multiplier[4], multiplicand[21] &
    multiplier[4], multiplicand[20] & multiplier[4], multiplicand
    [19] & multiplier[4], multiplicand[18] & multiplier[4],
    multiplicand[17] & multiplier[4], multiplicand[16] &
    multiplier[4], multiplicand[15] & multiplier[4], multiplicand
    [14] & multiplier[4], multiplicand[13] & multiplier[4],
    multiplicand[12] & multiplier[4], multiplicand[11] &
    multiplier[4], multiplicand[10] & multiplier[4], multiplicand
    [9] & multiplier[4], multiplicand[8] & multiplier[4],
    multiplicand[7] & multiplier[4], multiplicand[6] & multiplier
    [4], multiplicand[5] & multiplier[4], multiplicand[4] &
    multiplier[4], multiplicand[3] & multiplier[4], multiplicand
    [2] & multiplier[4], multiplicand[1] & multiplier[4],
    multiplicand[0] & multiplier[4], 4'd0};

assign P5 = {19'd0, multiplicand[23] & multiplier[5],
    multiplicand[22] & multiplier[5], multiplicand[21] &
    multiplier[5], multiplicand[20] & multiplier[5], multiplicand
    [19] & multiplier[5], multiplicand[18] & multiplier[5],
    multiplicand[17] & multiplier[5], multiplicand[16] &
    multiplier[5], multiplicand[15] & multiplier[5], multiplicand
    [14] & multiplier[5], multiplicand[13] & multiplier[5],
    multiplicand[12] & multiplier[5], multiplicand[11] &
    multiplier[5], multiplicand[10] & multiplier[5], multiplicand
    [9] & multiplier[5], multiplicand[8] & multiplier[5],
    multiplicand[7] & multiplier[5], multiplicand[6] & multiplier
    [5], multiplicand[5] & multiplier[5], multiplicand[4] &
    multiplier[5], multiplicand[3] & multiplier[5], multiplicand
    [2] & multiplier[5], multiplicand[1] & multiplier[5],
    multiplicand[0] & multiplier[5], 5'd0};
```

```
assign P6 = {18'd0, multiplicand[23] & multiplier[6],
    multiplicand[22] & multiplier[6], multiplicand[21] &
    multiplier[6], multiplicand[20] & multiplier[6], multiplicand
    [19] & multiplier[6], multiplicand[18] & multiplier[6],
    multiplicand[17] & multiplier[6], multiplicand[16] &
    multiplier[6], multiplicand[15] & multiplier[6], multiplicand
    [14] & multiplier[6], multiplicand[13] & multiplier[6],
    multiplicand[12] & multiplier[6], multiplicand[11] &
    multiplier[6], multiplicand[10] & multiplier[6], multiplicand
    [9] & multiplier[6], multiplicand[8] & multiplier[6],
    multiplicand[7] & multiplier[6], multiplicand[6] & multiplier
    [6], multiplicand[5] & multiplier[6], multiplicand[4] &
    multiplier[6], multiplicand[3] & multiplier[6], multiplicand
    [2] & multiplier[6], multiplicand[1] & multiplier[6],
    multiplicand[0] & multiplier[6], 6'd0};

assign P7 = {17'd0, multiplicand[23] & multiplier[7],
    multiplicand[22] & multiplier[7], multiplicand[21] &
    multiplier[7], multiplicand[20] & multiplier[7], multiplicand
    [19] & multiplier[7], multiplicand[18] & multiplier[7],
    multiplicand[17] & multiplier[7], multiplicand[16] &
    multiplier[7], multiplicand[15] & multiplier[7], multiplicand
    [14] & multiplier[7], multiplicand[13] & multiplier[7],
    multiplicand[12] & multiplier[7], multiplicand[11] &
    multiplier[7], multiplicand[10] & multiplier[7], multiplicand
    [9] & multiplier[7], multiplicand[8] & multiplier[7],
    multiplicand[7] & multiplier[7], multiplicand[6] & multiplier
    [7], multiplicand[5] & multiplier[7], multiplicand[4] &
    multiplier[7], multiplicand[3] & multiplier[7], multiplicand
    [2] & multiplier[7], multiplicand[1] & multiplier[7],
    multiplicand[0] & multiplier[7], 7'd0};

assign P8 = {16'd0, multiplicand[23] & multiplier[8],
    multiplicand[22] & multiplier[8], multiplicand[21] &
    multiplier[8], multiplicand[20] & multiplier[8], multiplicand
    [19] & multiplier[8], multiplicand[18] & multiplier[8],
    multiplicand[17] & multiplier[8], multiplicand[16] &
    multiplier[8], multiplicand[15] & multiplier[8], multiplicand
    [14] & multiplier[8], multiplicand[13] & multiplier[8],
    multiplicand[12] & multiplier[8], multiplicand[11] &
    multiplier[8], multiplicand[10] & multiplier[8], multiplicand
```

```
[9] & multiplier[8], multiplicand[8] & multiplier[8],
multiplicand[7] & multiplier[8], multiplicand[6] & multiplier
[8], multiplicand[5] & multiplier[8], multiplicand[4] &
multiplier[8], multiplicand[3] & multiplier[8], multiplicand
[2] & multiplier[8], multiplicand[1] & multiplier[8],
multiplicand[0] & multiplier[8], 8'd0};

assign P9 = {15'd0, multiplicand[23] & multiplier[9],
multiplicand[22] & multiplier[9], multiplicand[21] &
multiplier[9], multiplicand[20] & multiplier[9], multiplicand
[19] & multiplier[9], multiplicand[18] & multiplier[9],
multiplicand[17] & multiplier[9], multiplicand[16] &
multiplier[9], multiplicand[15] & multiplier[9], multiplicand
[14] & multiplier[9], multiplicand[13] & multiplier[9],
multiplicand[12] & multiplier[9], multiplicand[11] &
multiplier[9], multiplicand[10] & multiplier[9], multiplicand
[9] & multiplier[9], multiplicand[8] & multiplier[9],
multiplicand[7] & multiplier[9], multiplicand[6] & multiplier
[9], multiplicand[5] & multiplier[9], multiplicand[4] &
multiplier[9], multiplicand[3] & multiplier[9], multiplicand
[2] & multiplier[9], multiplicand[1] & multiplier[9],
multiplicand[0] & multiplier[9], 9'd0};

assign P10 = {14'd0, multiplicand[23] & multiplier[10],
multiplicand[22] & multiplier[10], multiplicand[21] &
multiplier[10], multiplicand[20] & multiplier[10],
multiplicand[19] & multiplier[10], multiplicand[18] &
multiplier[10], multiplicand[17] & multiplier[10],
multiplicand[16] & multiplier[10], multiplicand[15] &
multiplier[10], multiplicand[14] & multiplier[10],
multiplicand[13] & multiplier[10], multiplicand[12] &
multiplier[10], multiplicand[11] & multiplier[10],
multiplicand[10] & multiplier[10], multiplicand[9] &
multiplier[10], multiplicand[8] & multiplier[10], multiplicand
[7] & multiplier[10], multiplicand[6] & multiplier[10],
multiplicand[5] & multiplier[10], multiplicand[4] & multiplier
[10], multiplicand[3] & multiplier[10], multiplicand[2] &
multiplier[10], multiplicand[1] & multiplier[10], multiplicand
[0] & multiplier[10], 10'd0};

```

```
assign P11 = {13'd0, multiplicand[23] & multiplier[11],
    multiplicand[22] & multiplier[11], multiplicand[21] &
    multiplier[11], multiplicand[20] & multiplier[11],
    multiplicand[19] & multiplier[11], multiplicand[18] &
    multiplier[11], multiplicand[17] & multiplier[11],
    multiplicand[16] & multiplier[11], multiplicand[15] &
    multiplier[11], multiplicand[14] & multiplier[11],
    multiplicand[13] & multiplier[11], multiplicand[12] &
    multiplier[11], multiplicand[11] & multiplier[11],
    multiplicand[10] & multiplier[11], multiplicand[9] &
    multiplier[11], multiplicand[8] & multiplier[11], multiplicand
    [7] & multiplier[11], multiplicand[6] & multiplier[11],
    multiplicand[5] & multiplier[11], multiplicand[4] & multiplier
    [11], multiplicand[3] & multiplier[11], multiplicand[2] &
    multiplier[11], multiplicand[1] & multiplier[11], multiplicand
    [0] & multiplier[11], 11'd0};

assign P12 = {12'd0, multiplicand[23] & multiplier[12],
    multiplicand[22] & multiplier[12], multiplicand[21] &
    multiplier[12], multiplicand[20] & multiplier[12],
    multiplicand[19] & multiplier[12], multiplicand[18] &
    multiplier[12], multiplicand[17] & multiplier[12],
    multiplicand[16] & multiplier[12], multiplicand[15] &
    multiplier[12], multiplicand[14] & multiplier[12],
    multiplicand[13] & multiplier[12], multiplicand[12] &
    multiplier[12], multiplicand[11] & multiplier[12],
    multiplicand[10] & multiplier[12], multiplicand[9] &
    multiplier[12], multiplicand[8] & multiplier[12], multiplicand
    [7] & multiplier[12], multiplicand[6] & multiplier[12],
    multiplicand[5] & multiplier[12], multiplicand[4] & multiplier
    [12], multiplicand[3] & multiplier[12], multiplicand[2] &
    multiplier[12], multiplicand[1] & multiplier[12], multiplicand
    [0] & multiplier[12], 12'd0};

assign P13 = {11'd0, multiplicand[23] & multiplier[13],
    multiplicand[22] & multiplier[13], multiplicand[21] &
    multiplier[13], multiplicand[20] & multiplier[13],
    multiplicand[19] & multiplier[13], multiplicand[18] &
    multiplier[13], multiplicand[17] & multiplier[13],
    multiplicand[16] & multiplier[13], multiplicand[15] &
    multiplier[13], multiplicand[14] & multiplier[13],
```

```
    multiplicand[13] & multiplier[13], multiplicand[12] &
    multiplier[13], multiplicand[11] & multiplier[13],
    multiplicand[10] & multiplier[13], multiplicand[9] &
    multiplier[13], multiplicand[8] & multiplier[13], multiplicand
    [7] & multiplier[13], multiplicand[6] & multiplier[13],
    multiplicand[5] & multiplier[13], multiplicand[4] & multiplier
    [13], multiplicand[3] & multiplier[13], multiplicand[2] &
    multiplier[13], multiplicand[1] & multiplier[13], multiplicand
    [0] & multiplier[13], 13'd0};

assign P14 = {10'd0, multiplicand[23] & multiplier[14],
    multiplicand[22] & multiplier[14], multiplicand[21] &
    multiplier[14], multiplicand[20] & multiplier[14],
    multiplicand[19] & multiplier[14], multiplicand[18] &
    multiplier[14], multiplicand[17] & multiplier[14],
    multiplicand[16] & multiplier[14], multiplicand[15] &
    multiplier[14], multiplicand[14] & multiplier[14],
    multiplicand[13] & multiplier[14], multiplicand[12] &
    multiplier[14], multiplicand[11] & multiplier[14],
    multiplicand[10] & multiplier[14], multiplicand[9] &
    multiplier[14], multiplicand[8] & multiplier[14], multiplicand
    [7] & multiplier[14], multiplicand[6] & multiplier[14],
    multiplicand[5] & multiplier[14], multiplicand[4] & multiplier
    [14], multiplicand[3] & multiplier[14], multiplicand[2] &
    multiplier[14], multiplicand[1] & multiplier[14], multiplicand
    [0] & multiplier[14], 14'd0};

assign P15 = {9'd0, multiplicand[23] & multiplier[15],
    multiplicand[22] & multiplier[15], multiplicand[21] &
    multiplier[15], multiplicand[20] & multiplier[15],
    multiplicand[19] & multiplier[15], multiplicand[18] &
    multiplier[15], multiplicand[17] & multiplier[15],
    multiplicand[16] & multiplier[15], multiplicand[15] &
    multiplier[15], multiplicand[14] & multiplier[15],
    multiplicand[13] & multiplier[15], multiplicand[12] &
    multiplier[15], multiplicand[11] & multiplier[15],
    multiplicand[10] & multiplier[15], multiplicand[9] &
    multiplier[15], multiplicand[8] & multiplier[15], multiplicand
    [7] & multiplier[15], multiplicand[6] & multiplier[15],
    multiplicand[5] & multiplier[15], multiplicand[4] & multiplier
    [15], multiplicand[3] & multiplier[15], multiplicand[2] &
```

```
multiplier[15], multiplicand[1] & multiplier[15], multiplicand[0] & multiplier[15], 15'd0};

assign P16 = {8'd0, multiplicand[23] & multiplier[16], multiplicand[22] & multiplier[16], multiplicand[21] & multiplier[16], multiplicand[20] & multiplier[16], multiplicand[19] & multiplier[16], multiplicand[18] & multiplier[16], multiplicand[17] & multiplier[16], multiplicand[16] & multiplier[16], multiplicand[15] & multiplier[16], multiplicand[14] & multiplier[16], multiplicand[13] & multiplier[16], multiplicand[12] & multiplier[16], multiplicand[11] & multiplier[16], multiplicand[10] & multiplier[16], multiplicand[9] & multiplier[16], multiplicand[8] & multiplier[16], multiplicand[7] & multiplier[16], multiplicand[6] & multiplier[16], multiplicand[5] & multiplier[16], multiplicand[4] & multiplier[16], multiplicand[3] & multiplier[16], multiplicand[2] & multiplier[16], multiplicand[1] & multiplier[16], multiplicand[0] & multiplier[16], 16'd0};

assign P17 = {7'd0, multiplicand[23] & multiplier[17], multiplicand[22] & multiplier[17], multiplicand[21] & multiplier[17], multiplicand[20] & multiplier[17], multiplicand[19] & multiplier[17], multiplicand[18] & multiplier[17], multiplicand[17] & multiplier[17], multiplicand[16] & multiplier[17], multiplicand[15] & multiplier[17], multiplicand[14] & multiplier[17], multiplicand[13] & multiplier[17], multiplicand[12] & multiplier[17], multiplicand[11] & multiplier[17], multiplicand[10] & multiplier[17], multiplicand[9] & multiplier[17], multiplicand[8] & multiplier[17], multiplicand[7] & multiplier[17], multiplicand[6] & multiplier[17], multiplicand[5] & multiplier[17], multiplicand[4] & multiplier[17], multiplicand[3] & multiplier[17], multiplicand[2] & multiplier[17], multiplicand[1] & multiplier[17], multiplicand[0] & multiplier[17], 17'd0};

assign P18 = {6'd0, multiplicand[23] & multiplier[18], multiplicand[22] & multiplier[18], multiplicand[21] & multiplier[18], multiplicand[20] & multiplier[18], multiplicand[19] & multiplier[18], multiplicand[18] &
```

```
    multiplier[18], multiplicand[17] & multiplier[18],
    multiplicand[16] & multiplier[18], multiplicand[15] &
    multiplier[18], multiplicand[14] & multiplier[18],
    multiplicand[13] & multiplier[18], multiplicand[12] &
    multiplier[18], multiplicand[11] & multiplier[18],
    multiplicand[10] & multiplier[18], multiplicand[9] &
    multiplier[18], multiplicand[8] & multiplier[18], multiplicand
    [7] & multiplier[18], multiplicand[6] & multiplier[18],
    multiplicand[5] & multiplier[18], multiplicand[4] & multiplier
    [18], multiplicand[3] & multiplier[18], multiplicand[2] &
    multiplier[18], multiplicand[1] & multiplier[18], multiplicand
    [0] & multiplier[18], 18'd0};

assign P19 = {5'd0, multiplicand[23] & multiplier[19],
    multiplicand[22] & multiplier[19], multiplicand[21] &
    multiplier[19], multiplicand[20] & multiplier[19],
    multiplicand[19] & multiplier[19], multiplicand[18] &
    multiplier[19], multiplicand[17] & multiplier[19],
    multiplicand[16] & multiplier[19], multiplicand[15] &
    multiplier[19], multiplicand[14] & multiplier[19],
    multiplicand[13] & multiplier[19], multiplicand[12] &
    multiplier[19], multiplicand[11] & multiplier[19],
    multiplicand[10] & multiplier[19], multiplicand[9] &
    multiplier[19], multiplicand[8] & multiplier[19], multiplicand
    [7] & multiplier[19], multiplicand[6] & multiplier[19],
    multiplicand[5] & multiplier[19], multiplicand[4] & multiplier
    [19], multiplicand[3] & multiplier[19], multiplicand[2] &
    multiplier[19], multiplicand[1] & multiplier[19], multiplicand
    [0] & multiplier[19], 19'd0};

assign P20 = {4'd0, multiplicand[23] & multiplier[20],
    multiplicand[22] & multiplier[20], multiplicand[21] &
    multiplier[20], multiplicand[20] & multiplier[20],
    multiplicand[19] & multiplier[20], multiplicand[18] &
    multiplier[20], multiplicand[17] & multiplier[20],
    multiplicand[16] & multiplier[20], multiplicand[15] &
    multiplier[20], multiplicand[14] & multiplier[20],
    multiplicand[13] & multiplier[20], multiplicand[12] &
    multiplier[20], multiplicand[11] & multiplier[20],
    multiplicand[10] & multiplier[20], multiplicand[9] &
    multiplier[20], multiplicand[8] & multiplier[20], multiplicand
```

```
[7] & multiplier[20], multiplicand[6] & multiplier[20], multiplicand[5] & multiplier[20], multiplicand[4] & multiplier[20], multiplicand[3] & multiplier[20], multiplicand[2] & multiplier[20], multiplicand[1] & multiplier[20], multiplicand[0] & multiplier[20], 20'd0};

assign P21 = {3'd0, multiplicand[23] & multiplier[21], multiplicand[22] & multiplier[21], multiplicand[21] & multiplier[21], multiplicand[20] & multiplier[21], multiplicand[19] & multiplier[21], multiplicand[18] & multiplier[21], multiplicand[17] & multiplier[21], multiplicand[16] & multiplier[21], multiplicand[15] & multiplier[21], multiplicand[14] & multiplier[21], multiplicand[13] & multiplier[21], multiplicand[12] & multiplier[21], multiplicand[11] & multiplier[21], multiplicand[10] & multiplier[21], multiplicand[9] & multiplier[21], multiplicand[8] & multiplier[21], multiplicand[7] & multiplier[21], multiplicand[6] & multiplier[21], multiplicand[5] & multiplier[21], multiplicand[4] & multiplier[21], multiplicand[3] & multiplier[21], multiplicand[2] & multiplier[21], multiplicand[1] & multiplier[21], multiplicand[0] & multiplier[21], 21'd0};

assign P22 = {2'd0, multiplicand[23] & multiplier[22], multiplicand[22] & multiplier[22], multiplicand[21] & multiplier[22], multiplicand[20] & multiplier[22], multiplicand[19] & multiplier[22], multiplicand[18] & multiplier[22], multiplicand[17] & multiplier[22], multiplicand[16] & multiplier[22], multiplicand[15] & multiplier[22], multiplicand[14] & multiplier[22], multiplicand[13] & multiplier[22], multiplicand[12] & multiplier[22], multiplicand[11] & multiplier[22], multiplicand[10] & multiplier[22], multiplicand[9] & multiplier[22], multiplicand[8] & multiplier[22], multiplicand[7] & multiplier[22], multiplicand[6] & multiplier[22], multiplicand[5] & multiplier[22], multiplicand[4] & multiplier[22], multiplicand[3] & multiplier[22], multiplicand[2] & multiplier[22], multiplicand[1] & multiplier[22], multiplicand[0] & multiplier[22], 22'd0};

```

```
assign P23 = {1'd0, multiplicand[23] & multiplier[23], multiplicand[22] & multiplier[23], multiplicand[21] & multiplier[23], multiplicand[20] & multiplier[23], multiplicand[19] & multiplier[23], multiplicand[18] & multiplier[23], multiplicand[17] & multiplier[23], multiplicand[16] & multiplier[23], multiplicand[15] & multiplier[23], multiplicand[14] & multiplier[23], multiplicand[13] & multiplier[23], multiplicand[12] & multiplier[23], multiplicand[11] & multiplier[23], multiplicand[10] & multiplier[23], multiplicand[9] & multiplier[23], multiplicand[8] & multiplier[23], multiplicand[7] & multiplier[23], multiplicand[6] & multiplier[23], multiplicand[5] & multiplier[23], multiplicand[4] & multiplier[23], multiplicand[3] & multiplier[23], multiplicand[2] & multiplier[23], multiplicand[1] & multiplier[23], multiplicand[0] & multiplier[23], 23'd0};

assign S11 = P0[1] ^ P1[1];
assign C11 = P0[1] & P1[1];
assign S12 = P0[2] ^ P1[2] ^ P2[2];
assign C12 = (P0[2] & P1[2]) | (P1[2] & P2[2]) | (P0[2] & P2[2]);
assign S13 = P0[3] ^ P1[3] ^ P2[3];
assign C13 = (P0[3] & P1[3]) | (P1[3] & P2[3]) | (P0[3] & P2[3]);
assign S114 = P0[4] ^ P1[4] ^ P2[4];
assign C114 = (P0[4] & P1[4]) | (P1[4] & P2[4]) | (P0[4] & P2[4]);
assign S115 = P0[5] ^ P1[5] ^ P2[5];
assign C115 = (P0[5] & P1[5]) | (P1[5] & P2[5]) | (P0[5] & P2[5]);
assign S116 = P0[6] ^ P1[6] ^ P2[6];
assign C116 = (P0[6] & P1[6]) | (P1[6] & P2[6]) | (P0[6] & P2[6]);
assign S117 = P0[7] ^ P1[7] ^ P2[7];
assign C117 = (P0[7] & P1[7]) | (P1[7] & P2[7]) | (P0[7] & P2[7]);
assign S118 = P0[8] ^ P1[8] ^ P2[8];
assign C118 = (P0[8] & P1[8]) | (P1[8] & P2[8]) | (P0[8] & P2[8]);
assign S119 = P0[9] ^ P1[9] ^ P2[9];
assign C119 = (P0[9] & P1[9]) | (P1[9] & P2[9]) | (P0[9] & P2[9]);
```

```
assign S1110 = P0[10] ^ P1[10] ^ P2[10];
assign C1110_inst = (P0[10] & P1[10]) | (P1[10] & P2[10]) | (P0[10] & P2[10]);
assign S1111 = P0[11] ^ P1[11] ^ P2[11];
assign C1111_inst = (P0[11] & P1[11]) | (P1[11] & P2[11]) | (P0[11] & P2[11]);
assign S1112 = P0[12] ^ P1[12] ^ P2[12];
assign C1112_inst = (P0[12] & P1[12]) | (P1[12] & P2[12]) | (P0[12] & P2[12]);
assign S1113 = P0[13] ^ P1[13] ^ P2[13];
assign C1113_inst = (P0[13] & P1[13]) | (P1[13] & P2[13]) | (P0[13] & P2[13]);
assign S1114 = P0[14] ^ P1[14] ^ P2[14];
assign C1114_inst = (P0[14] & P1[14]) | (P1[14] & P2[14]) | (P0[14] & P2[14]);
assign S1115 = P0[15] ^ P1[15] ^ P2[15];
assign C1115_inst = (P0[15] & P1[15]) | (P1[15] & P2[15]) | (P0[15] & P2[15]);
assign S1116 = P0[16] ^ P1[16] ^ P2[16];
assign C1116_inst = (P0[16] & P1[16]) | (P1[16] & P2[16]) | (P0[16] & P2[16]);
assign S1117 = P0[17] ^ P1[17] ^ P2[17];
assign C1117_inst = (P0[17] & P1[17]) | (P1[17] & P2[17]) | (P0[17] & P2[17]);
assign S1118 = P0[18] ^ P1[18] ^ P2[18];
assign C1118_inst = (P0[18] & P1[18]) | (P1[18] & P2[18]) | (P0[18] & P2[18]);
assign S1119 = P0[19] ^ P1[19] ^ P2[19];
assign C1119_inst = (P0[19] & P1[19]) | (P1[19] & P2[19]) | (P0[19] & P2[19]);
assign S1120 = P0[20] ^ P1[20] ^ P2[20];
assign C1120_inst = (P0[20] & P1[20]) | (P1[20] & P2[20]) | (P0[20] & P2[20]);
assign S1121 = P0[21] ^ P1[21] ^ P2[21];
assign C1121_inst = (P0[21] & P1[21]) | (P1[21] & P2[21]) | (P0[21] & P2[21]);
assign S1122 = P0[22] ^ P1[22] ^ P2[22];
assign C1122_inst = (P0[22] & P1[22]) | (P1[22] & P2[22]) | (P0[22] & P2[22]);
assign S1123 = P1[23] ^ P2[23] ^ P3[23];
```

```
assign C1123_inst = (P1[23] & P2[23]) | (P2[23] & P3[23]) | (P1[23] & P3[23]);
assign S1124 = P2[24] ^ P3[24] ^ P4[24];
assign C1124_inst = (P2[24] & P3[24]) | (P3[24] & P4[24]) | (P2[24] & P4[24]);
assign S1125 = P3[25] ^ P4[25] ^ P5[25];
assign C1125_inst = (P3[25] & P4[25]) | (P4[25] & P5[25]) | (P3[25] & P5[25]);
assign S1126 = P4[26] ^ P5[26] ^ P6[26];
assign C1126_inst = (P4[26] & P5[26]) | (P5[26] & P6[26]) | (P4[26] & P6[26]);
assign S1127 = P5[27] ^ P6[27] ^ P7[27];
assign C1127_inst = (P5[27] & P6[27]) | (P6[27] & P7[27]) | (P5[27] & P7[27]);
assign S1128 = P6[28] ^ P7[28] ^ P8[28];
assign C1128_inst = (P6[28] & P7[28]) | (P7[28] & P8[28]) | (P6[28] & P8[28]);
assign S1129 = P7[29] ^ P8[29] ^ P9[29];
assign C1129_inst = (P7[29] & P8[29]) | (P8[29] & P9[29]) | (P7[29] & P9[29]);
assign S1130 = P8[30] ^ P9[30] ^ P10[30];
assign C1130_inst = (P8[30] & P9[30]) | (P9[30] & P10[30]) | (P8[30] & P10[30]);
assign S1131 = P9[31] ^ P10[31] ^ P11[31];
assign C1131_inst = (P9[31] & P10[31]) | (P10[31] & P11[31]) | (P9[31] & P11[31]);
assign S1132 = P10[32] ^ P11[32] ^ P12[32];
assign C1132_inst = (P10[32] & P11[32]) | (P11[32] & P12[32]) | (P10[32] & P12[32]);
assign S1133 = P11[33] ^ P12[33] ^ P13[33];
assign C1133_inst = (P11[33] & P12[33]) | (P12[33] & P13[33]) | (P11[33] & P13[33]);
assign S1134 = P12[34] ^ P13[34] ^ P14[34];
assign C1134_inst = (P12[34] & P13[34]) | (P13[34] & P14[34]) | (P12[34] & P14[34]);
assign S1135 = P13[35] ^ P14[35] ^ P15[35];
assign C1135_inst = (P13[35] & P14[35]) | (P14[35] & P15[35]) | (P13[35] & P15[35]);
assign S1136 = P14[36] ^ P15[36] ^ P16[36];
assign C1136_inst = (P14[36] & P15[36]) | (P15[36] & P16[36]) | (P14[36] & P16[36]);
```

```
assign S1137 = P15[37] ^ P16[37] ^ P17[37];
assign C1137_inst = (P15[37] & P16[37]) | (P16[37] & P17[37]) | (P15[37] & P17[37]);
assign S1138 = P16[38] ^ P17[38] ^ P18[38];
assign C1138_inst = (P16[38] & P17[38]) | (P17[38] & P18[38]) | (P16[38] & P18[38]);
assign S1139 = P17[39] ^ P18[39] ^ P19[39];
assign C1139_inst = (P17[39] & P18[39]) | (P18[39] & P19[39]) | (P17[39] & P19[39]);
assign S1140 = P18[40] ^ P19[40] ^ P20[40];
assign C1140_inst = (P18[40] & P19[40]) | (P19[40] & P20[40]) | (P18[40] & P20[40]);
assign S1141 = P19[41] ^ P20[41] ^ P21[41];
assign C1141_inst = (P19[41] & P20[41]) | (P20[41] & P21[41]) | (P19[41] & P21[41]);
assign S1142 = P20[42] ^ P21[42] ^ P22[42];
assign C1142_inst = (P20[42] & P21[42]) | (P21[42] & P22[42]) | (P20[42] & P22[42]);
assign S1143 = P21[43] ^ P22[43];
assign C1143_inst = P21[43] & P22[43];

assign S124 = P3[4] ^ P4[4];
assign C124 = P3[4] & P4[4];
assign S125 = P3[5] ^ P4[5] ^ P5[5];
assign C125 = (P3[5] & P4[5]) | (P4[5] & P5[5]) | (P3[5] & P5[5]);
assign S126 = P3[6] ^ P4[6] ^ P5[6];
assign C126 = (P3[6] & P4[6]) | (P4[6] & P5[6]) | (P3[6] & P5[6]);
assign S127 = P3[7] ^ P4[7] ^ P5[7];
assign C127 = (P3[7] & P4[7]) | (P4[7] & P5[7]) | (P3[7] & P5[7]);
assign S128 = P3[8] ^ P4[8] ^ P5[8];
assign C128 = (P3[8] & P4[8]) | (P4[8] & P5[8]) | (P3[8] & P5[8]);
assign S129 = P3[9] ^ P4[9] ^ P5[9];
assign C129 = (P3[9] & P4[9]) | (P4[9] & P5[9]) | (P3[9] & P5[9]);
assign S1210 = P3[10] ^ P4[10] ^ P5[10];
assign C1210_inst = (P3[10] & P4[10]) | (P4[10] & P5[10]) | (P3[10] & P5[10]);
```

```
assign S1211 = P3[11] ^ P4[11] ^ P5[11];
assign C1211_inst = (P3[11] & P4[11]) | (P4[11] & P5[11]) | (P3[11] & P5[11]);
assign S1212 = P3[12] ^ P4[12] ^ P5[12];
assign C1212_inst = (P3[12] & P4[12]) | (P4[12] & P5[12]) | (P3[12] & P5[12]);
assign S1213 = P3[13] ^ P4[13] ^ P5[13];
assign C1213_inst = (P3[13] & P4[13]) | (P4[13] & P5[13]) | (P3[13] & P5[13]);
assign S1214 = P3[14] ^ P4[14] ^ P5[14];
assign C1214_inst = (P3[14] & P4[14]) | (P4[14] & P5[14]) | (P3[14] & P5[14]);
assign S1215 = P3[15] ^ P4[15] ^ P5[15];
assign C1215_inst = (P3[15] & P4[15]) | (P4[15] & P5[15]) | (P3[15] & P5[15]);
assign S1216 = P3[16] ^ P4[16] ^ P5[16];
assign C1216_inst = (P3[16] & P4[16]) | (P4[16] & P5[16]) | (P3[16] & P5[16]);
assign S1217 = P3[17] ^ P4[17] ^ P5[17];
assign C1217_inst = (P3[17] & P4[17]) | (P4[17] & P5[17]) | (P3[17] & P5[17]);
assign S1218 = P3[18] ^ P4[18] ^ P5[18];
assign C1218_inst = (P3[18] & P4[18]) | (P4[18] & P5[18]) | (P3[18] & P5[18]);
assign S1219 = P3[19] ^ P4[19] ^ P5[19];
assign C1219_inst = (P3[19] & P4[19]) | (P4[19] & P5[19]) | (P3[19] & P5[19]);
assign S1220 = P3[20] ^ P4[20] ^ P5[20];
assign C1220_inst = (P3[20] & P4[20]) | (P4[20] & P5[20]) | (P3[20] & P5[20]);
assign S1221 = P3[21] ^ P4[21] ^ P5[21];
assign C1221_inst = (P3[21] & P4[21]) | (P4[21] & P5[21]) | (P3[21] & P5[21]);
assign S1222 = P3[22] ^ P4[22] ^ P5[22];
assign C1222_inst = (P3[22] & P4[22]) | (P4[22] & P5[22]) | (P3[22] & P5[22]);
assign S1223 = P4[23] ^ P5[23] ^ P6[23];
assign C1223_inst = (P4[23] & P5[23]) | (P5[23] & P6[23]) | (P4[23] & P6[23]);
assign S1224 = P5[24] ^ P6[24] ^ P7[24];
```

```
assign C1224_inst = (P5[24] & P6[24]) | (P6[24] & P7[24]) | (P5[24] & P7[24]);
assign S1225 = P6[25] ^ P7[25] ^ P8[25];
assign C1225_inst = (P6[25] & P7[25]) | (P7[25] & P8[25]) | (P6[25] & P8[25]);
assign S1226 = P7[26] ^ P8[26] ^ P9[26];
assign C1226_inst = (P7[26] & P8[26]) | (P8[26] & P9[26]) | (P7[26] & P9[26]);
assign S1227 = P8[27] ^ P9[27] ^ P10[27];
assign C1227_inst = (P8[27] & P9[27]) | (P9[27] & P10[27]) | (P8[27] & P10[27]);
assign S1228 = P9[28] ^ P10[28] ^ P11[28];
assign C1228_inst = (P9[28] & P10[28]) | (P10[28] & P11[28]) | (P9[28] & P11[28]);
assign S1229 = P10[29] ^ P11[29] ^ P12[29];
assign C1229_inst = (P10[29] & P11[29]) | (P11[29] & P12[29]) | (P10[29] & P12[29]);
assign S1230 = P11[30] ^ P12[30] ^ P13[30];
assign C1230_inst = (P11[30] & P12[30]) | (P12[30] & P13[30]) | (P11[30] & P13[30]);
assign S1231 = P12[31] ^ P13[31] ^ P14[31];
assign C1231_inst = (P12[31] & P13[31]) | (P13[31] & P14[31]) | (P12[31] & P14[31]);
assign S1232 = P13[32] ^ P14[32] ^ P15[32];
assign C1232_inst = (P13[32] & P14[32]) | (P14[32] & P15[32]) | (P13[32] & P15[32]);
assign S1233 = P14[33] ^ P15[33] ^ P16[33];
assign C1233_inst = (P14[33] & P15[33]) | (P15[33] & P16[33]) | (P14[33] & P16[33]);
assign S1234 = P15[34] ^ P16[34] ^ P17[34];
assign C1234_inst = (P15[34] & P16[34]) | (P16[34] & P17[34]) | (P15[34] & P17[34]);
assign S1235 = P16[35] ^ P17[35] ^ P18[35];
assign C1235_inst = (P16[35] & P17[35]) | (P17[35] & P18[35]) | (P16[35] & P18[35]);
assign S1236 = P17[36] ^ P18[36] ^ P19[36];
assign C1236_inst = (P17[36] & P18[36]) | (P18[36] & P19[36]) | (P17[36] & P19[36]);
assign S1237 = P18[37] ^ P19[37] ^ P20[37];
assign C1237_inst = (P18[37] & P19[37]) | (P19[37] & P20[37]) | (P18[37] & P20[37]);
```

```
assign S1238 = P19[38] ^ P20[38] ^ P21[38];
assign C1238_inst = (P19[38] & P20[38]) | (P20[38] & P21[38]) | (P19[38] & P21[38]);
assign S1239 = P20[39] ^ P21[39] ^ P22[39];
assign C1239_inst = (P20[39] & P21[39]) | (P21[39] & P22[39]) | (P20[39] & P22[39]);
assign S1240 = P21[40] ^ P22[40];
assign C1240_inst = P21[40] & P22[40];

assign S137 = P6[7] ^ P7[7];
assign C137 = P6[7] & P7[7];
assign S138 = P6[8] ^ P7[8] ^ P8[8];
assign C138 = (P6[8] & P7[8]) | (P7[8] & P8[8]) | (P6[8] & P8[8]);
assign S139 = P6[9] ^ P7[9] ^ P8[9];
assign C139 = (P6[9] & P7[9]) | (P7[9] & P8[9]) | (P6[9] & P8[9]);
assign S1310 = P6[10] ^ P7[10] ^ P8[10];
assign C1310_inst = (P6[10] & P7[10]) | (P7[10] & P8[10]) | (P6[10] & P8[10]);
assign S1311 = P6[11] ^ P7[11] ^ P8[11];
assign C1311_inst = (P6[11] & P7[11]) | (P7[11] & P8[11]) | (P6[11] & P8[11]);
assign S1312 = P6[12] ^ P7[12] ^ P8[12];
assign C1312_inst = (P6[12] & P7[12]) | (P7[12] & P8[12]) | (P6[12] & P8[12]);
assign S1313 = P6[13] ^ P7[13] ^ P8[13];
assign C1313_inst = (P6[13] & P7[13]) | (P7[13] & P8[13]) | (P6[13] & P8[13]);
assign S1314 = P6[14] ^ P7[14] ^ P8[14];
assign C1314_inst = (P6[14] & P7[14]) | (P7[14] & P8[14]) | (P6[14] & P8[14]);
assign S1315 = P6[15] ^ P7[15] ^ P8[15];
assign C1315_inst = (P6[15] & P7[15]) | (P7[15] & P8[15]) | (P6[15] & P8[15]);
assign S1316 = P6[16] ^ P7[16] ^ P8[16];
assign C1316_inst = (P6[16] & P7[16]) | (P7[16] & P8[16]) | (P6[16] & P8[16]);
assign S1317 = P6[17] ^ P7[17] ^ P8[17];
assign C1317_inst = (P6[17] & P7[17]) | (P7[17] & P8[17]) | (P6[17] & P8[17]);
```

```
assign S1318 = P6[18] ^ P7[18] ^ P8[18];
assign C1318_inst = (P6[18] & P7[18]) | (P7[18] & P8[18]) | (P6[18] & P8[18]);
assign S1319 = P6[19] ^ P7[19] ^ P8[19];
assign C1319_inst = (P6[19] & P7[19]) | (P7[19] & P8[19]) | (P6[19] & P8[19]);
assign S1320 = P6[20] ^ P7[20] ^ P8[20];
assign C1320_inst = (P6[20] & P7[20]) | (P7[20] & P8[20]) | (P6[20] & P8[20]);
assign S1321 = P6[21] ^ P7[21] ^ P8[21];
assign C1321_inst = (P6[21] & P7[21]) | (P7[21] & P8[21]) | (P6[21] & P8[21]);
assign S1322 = P6[22] ^ P7[22] ^ P8[22];
assign C1322_inst = (P6[22] & P7[22]) | (P7[22] & P8[22]) | (P6[22] & P8[22]);
assign S1323 = P7[23] ^ P8[23] ^ P9[23];
assign C1323_inst = (P7[23] & P8[23]) | (P8[23] & P9[23]) | (P7[23] & P9[23]);
assign S1324 = P8[24] ^ P9[24] ^ P10[24];
assign C1324_inst = (P8[24] & P9[24]) | (P9[24] & P10[24]) | (P8[24] & P10[24]);
assign S1325 = P9[25] ^ P10[25] ^ P11[25];
assign C1325_inst = (P9[25] & P10[25]) | (P10[25] & P11[25]) | (P9[25] & P11[25]);
assign S1326 = P10[26] ^ P11[26] ^ P12[26];
assign C1326_inst = (P10[26] & P11[26]) | (P11[26] & P12[26]) | (P10[26] & P12[26]);
assign S1327 = P11[27] ^ P12[27] ^ P13[27];
assign C1327_inst = (P11[27] & P12[27]) | (P12[27] & P13[27]) | (P11[27] & P13[27]);
assign S1328 = P12[28] ^ P13[28] ^ P14[28];
assign C1328_inst = (P12[28] & P13[28]) | (P13[28] & P14[28]) | (P12[28] & P14[28]);
assign S1329 = P13[29] ^ P14[29] ^ P15[29];
assign C1329_inst = (P13[29] & P14[29]) | (P14[29] & P15[29]) | (P13[29] & P15[29]);
assign S1330 = P14[30] ^ P15[30] ^ P16[30];
assign C1330_inst = (P14[30] & P15[30]) | (P15[30] & P16[30]) | (P14[30] & P16[30]);
assign S1331 = P15[31] ^ P16[31] ^ P17[31];
```

```
assign C1331_inst = (P15[31] & P16[31]) | (P16[31] & P17[31]) | (
    P15[31] & P17[31]);
assign S1332 = P16[32] ^ P17[32] ^ P18[32];
assign C1332_inst = (P16[32] & P17[32]) | (P17[32] & P18[32]) | (
    P16[32] & P18[32]);
assign S1333 = P17[33] ^ P18[33] ^ P19[33];
assign C1333_inst = (P17[33] & P18[33]) | (P18[33] & P19[33]) | (
    P17[33] & P19[33]);
assign S1334 = P18[34] ^ P19[34] ^ P20[34];
assign C1334_inst = (P18[34] & P19[34]) | (P19[34] & P20[34]) | (
    P18[34] & P20[34]);
assign S1335 = P19[35] ^ P20[35] ^ P21[35];
assign C1335_inst = (P19[35] & P20[35]) | (P20[35] & P21[35]) | (
    P19[35] & P21[35]);
assign S1336 = P20[36] ^ P21[36] ^ P22[36];
assign C1336_inst = (P20[36] & P21[36]) | (P21[36] & P22[36]) | (
    P20[36] & P22[36]);
assign S1337 = P21[37] ^ P22[37];
assign C1337_inst = P21[37] & P22[37];

assign S1410 = P9[10] ^ P10[10];
assign C1410_inst = P9[10] & P10[10];
assign S1411 = P9[11] ^ P10[11] ^ P11[11];
assign C1411_inst = (P9[11] & P10[11]) | (P10[11] & P11[11]) | (
    P9[11] & P11[11]);
assign S1412 = P9[12] ^ P10[12] ^ P11[12];
assign C1412_inst = (P9[12] & P10[12]) | (P10[12] & P11[12]) | (
    P9[12] & P11[12]);
assign S1413 = P9[13] ^ P10[13] ^ P11[13];
assign C1413_inst = (P9[13] & P10[13]) | (P10[13] & P11[13]) | (
    P9[13] & P11[13]);
assign S1414 = P9[14] ^ P10[14] ^ P11[14];
assign C1414_inst = (P9[14] & P10[14]) | (P10[14] & P11[14]) | (
    P9[14] & P11[14]);
assign S1415 = P9[15] ^ P10[15] ^ P11[15];
assign C1415_inst = (P9[15] & P10[15]) | (P10[15] & P11[15]) | (
    P9[15] & P11[15]);
assign S1416 = P9[16] ^ P10[16] ^ P11[16];
assign C1416_inst = (P9[16] & P10[16]) | (P10[16] & P11[16]) | (
    P9[16] & P11[16]);
assign S1417 = P9[17] ^ P10[17] ^ P11[17];
```

```
assign C1417_inst = (P9[17] & P10[17]) | (P10[17] & P11[17]) | (
    P9[17] & P11[17]);
assign S1418 = P9[18] ^ P10[18] ^ P11[18];
assign C1418_inst = (P9[18] & P10[18]) | (P10[18] & P11[18]) | (
    P9[18] & P11[18]);
assign S1419 = P9[19] ^ P10[19] ^ P11[19];
assign C1419_inst = (P9[19] & P10[19]) | (P10[19] & P11[19]) | (
    P9[19] & P11[19]);
assign S1420 = P9[20] ^ P10[20] ^ P11[20];
assign C1420_inst = (P9[20] & P10[20]) | (P10[20] & P11[20]) | (
    P9[20] & P11[20]);
assign S1421 = P9[21] ^ P10[21] ^ P11[21];
assign C1421_inst = (P9[21] & P10[21]) | (P10[21] & P11[21]) | (
    P9[21] & P11[21]);
assign S1422 = P9[22] ^ P10[22] ^ P11[22];
assign C1422_inst = (P9[22] & P10[22]) | (P10[22] & P11[22]) | (
    P9[22] & P11[22]);
assign S1423 = P10[23] ^ P11[23] ^ P12[23];
assign C1423_inst = (P10[23] & P11[23]) | (P11[23] & P12[23]) | (
    P10[23] & P12[23]);
assign S1424 = P11[24] ^ P12[24] ^ P13[24];
assign C1424_inst = (P11[24] & P12[24]) | (P12[24] & P13[24]) | (
    P11[24] & P13[24]);
assign S1425 = P12[25] ^ P13[25] ^ P14[25];
assign C1425_inst = (P12[25] & P13[25]) | (P13[25] & P14[25]) | (
    P12[25] & P14[25]);
assign S1426 = P13[26] ^ P14[26] ^ P15[26];
assign C1426_inst = (P13[26] & P14[26]) | (P14[26] & P15[26]) | (
    P13[26] & P15[26]);
assign S1427 = P14[27] ^ P15[27] ^ P16[27];
assign C1427_inst = (P14[27] & P15[27]) | (P15[27] & P16[27]) | (
    P14[27] & P16[27]);
assign S1428 = P15[28] ^ P16[28] ^ P17[28];
assign C1428_inst = (P15[28] & P16[28]) | (P16[28] & P17[28]) | (
    P15[28] & P17[28]);
assign S1429 = P16[29] ^ P17[29] ^ P18[29];
assign C1429_inst = (P16[29] & P17[29]) | (P17[29] & P18[29]) | (
    P16[29] & P18[29]);
assign S1430 = P17[30] ^ P18[30] ^ P19[30];
assign C1430_inst = (P17[30] & P18[30]) | (P18[30] & P19[30]) | (
    P17[30] & P19[30]);
```

```
assign S1431 = P18[31] ^ P19[31] ^ P20[31];
assign C1431_inst = (P18[31] & P19[31]) | (P19[31] & P20[31]) | (
    P18[31] & P20[31]);
assign S1432 = P19[32] ^ P20[32] ^ P21[32];
assign C1432_inst = (P19[32] & P20[32]) | (P20[32] & P21[32]) | (
    P19[32] & P21[32]);
assign S1433 = P20[33] ^ P21[33] ^ P22[33];
assign C1433_inst = (P20[33] & P21[33]) | (P21[33] & P22[33]) | (
    P20[33] & P22[33]);
assign S1434 = P21[34] ^ P22[34];
assign C1434_inst = P21[34] & P22[34];

assign S1513 = P12[13] ^ P13[13];
assign C1513_inst = P12[13] & P13[13];
assign S1514 = P12[14] ^ P13[14] ^ P14[14];
assign C1514_inst = (P12[14] & P13[14]) | (P13[14] & P14[14]) | (
    P12[14] & P14[14]);
assign S1515 = P12[15] ^ P13[15] ^ P14[15];
assign C1515_inst = (P12[15] & P13[15]) | (P13[15] & P14[15]) | (
    P12[15] & P14[15]);
assign S1516 = P12[16] ^ P13[16] ^ P14[16];
assign C1516_inst = (P12[16] & P13[16]) | (P13[16] & P14[16]) | (
    P12[16] & P14[16]);
assign S1517 = P12[17] ^ P13[17] ^ P14[17];
assign C1517_inst = (P12[17] & P13[17]) | (P13[17] & P14[17]) | (
    P12[17] & P14[17]);
assign S1518 = P12[18] ^ P13[18] ^ P14[18];
assign C1518_inst = (P12[18] & P13[18]) | (P13[18] & P14[18]) | (
    P12[18] & P14[18]);
assign S1519 = P12[19] ^ P13[19] ^ P14[19];
assign C1519_inst = (P12[19] & P13[19]) | (P13[19] & P14[19]) | (
    P12[19] & P14[19]);
assign S1520 = P12[20] ^ P13[20] ^ P14[20];
assign C1520_inst = (P12[20] & P13[20]) | (P13[20] & P14[20]) | (
    P12[20] & P14[20]);
assign S1521 = P12[21] ^ P13[21] ^ P14[21];
assign C1521_inst = (P12[21] & P13[21]) | (P13[21] & P14[21]) | (
    P12[21] & P14[21]);
assign S1522 = P12[22] ^ P13[22] ^ P14[22];
assign C1522_inst = (P12[22] & P13[22]) | (P13[22] & P14[22]) | (
    P12[22] & P14[22]);
```

```
assign S1523 = P13[23] ^ P14[23] ^ P15[23];
assign C1523_inst = (P13[23] & P14[23]) | (P14[23] & P15[23]) | (
    P13[23] & P15[23]);
assign S1524 = P14[24] ^ P15[24] ^ P16[24];
assign C1524_inst = (P14[24] & P15[24]) | (P15[24] & P16[24]) | (
    P14[24] & P16[24]);
assign S1525 = P15[25] ^ P16[25] ^ P17[25];
assign C1525_inst = (P15[25] & P16[25]) | (P16[25] & P17[25]) | (
    P15[25] & P17[25]);
assign S1526 = P16[26] ^ P17[26] ^ P18[26];
assign C1526_inst = (P16[26] & P17[26]) | (P17[26] & P18[26]) | (
    P16[26] & P18[26]);
assign S1527 = P17[27] ^ P18[27] ^ P19[27];
assign C1527_inst = (P17[27] & P18[27]) | (P18[27] & P19[27]) | (
    P17[27] & P19[27]);
assign S1528 = P18[28] ^ P19[28] ^ P20[28];
assign C1528_inst = (P18[28] & P19[28]) | (P19[28] & P20[28]) | (
    P18[28] & P20[28]);
assign S1529 = P19[29] ^ P20[29] ^ P21[29];
assign C1529_inst = (P19[29] & P20[29]) | (P20[29] & P21[29]) | (
    P19[29] & P21[29]);
assign S1530 = P20[30] ^ P21[30] ^ P22[30];
assign C1530_inst = (P20[30] & P21[30]) | (P21[30] & P22[30]) | (
    P20[30] & P22[30]);
assign S1531 = P21[31] ^ P22[31];
assign C1531_inst = P21[31] & P22[31];

assign S1616 = P15[16] ^ P16[16];
assign C1616_inst = P15[16] & P16[16];
assign S1617 = P15[17] ^ P16[17] ^ P17[17];
assign C1617_inst = (P15[17] & P16[17]) | (P16[17] & P17[17]) | (
    P15[17] & P17[17]);
assign S1618 = P15[18] ^ P16[18] ^ P17[18];
assign C1618_inst = (P15[18] & P16[18]) | (P16[18] & P17[18]) | (
    P15[18] & P17[18]);
assign S1619 = P15[19] ^ P16[19] ^ P17[19];
assign C1619_inst = (P15[19] & P16[19]) | (P16[19] & P17[19]) | (
    P15[19] & P17[19]);
assign S1620 = P15[20] ^ P16[20] ^ P17[20];
assign C1620_inst = (P15[20] & P16[20]) | (P16[20] & P17[20]) | (
    P15[20] & P17[20]);
```

```
assign S1621 = P15[21] ^ P16[21] ^ P17[21];
assign C1621_inst = (P15[21] & P16[21]) | (P16[21] & P17[21]) | (
    P15[21] & P17[21]);
assign S1622 = P15[22] ^ P16[22] ^ P17[22];
assign C1622_inst = (P15[22] & P16[22]) | (P16[22] & P17[22]) | (
    P15[22] & P17[22]);
assign S1623 = P16[23] ^ P17[23] ^ P18[23];
assign C1623_inst = (P16[23] & P17[23]) | (P17[23] & P18[23]) | (
    P16[23] & P18[23]);
assign S1624 = P17[24] ^ P18[24] ^ P19[24];
assign C1624_inst = (P17[24] & P18[24]) | (P18[24] & P19[24]) | (
    P17[24] & P19[24]);
assign S1625 = P18[25] ^ P19[25] ^ P20[25];
assign C1625_inst = (P18[25] & P19[25]) | (P19[25] & P20[25]) | (
    P18[25] & P20[25]);
assign S1626 = P19[26] ^ P20[26] ^ P21[26];
assign C1626_inst = (P19[26] & P20[26]) | (P20[26] & P21[26]) | (
    P19[26] & P21[26]);
assign S1627 = P20[27] ^ P21[27] ^ P22[27];
assign C1627_inst = (P20[27] & P21[27]) | (P21[27] & P22[27]) | (
    P20[27] & P22[27]);
assign S1628 = P21[28] ^ P22[28];
assign C1628_inst = P21[28] & P22[28];

assign S1719 = P18[19] ^ P19[19];
assign C1719_inst = P18[19] & P19[19];
assign S1720 = P18[20] ^ P19[20] ^ P20[20];
assign C1720_inst = (P18[20] & P19[20]) | (P19[20] & P20[20]) | (
    P18[20] & P20[20]);
assign S1721 = P18[21] ^ P19[21] ^ P20[21];
assign C1721_inst = (P18[21] & P19[21]) | (P19[21] & P20[21]) | (
    P18[21] & P20[21]);
assign S1722 = P18[22] ^ P19[22] ^ P20[22];
assign C1722_inst = (P18[22] & P19[22]) | (P19[22] & P20[22]) | (
    P18[22] & P20[22]);
assign S1723 = P19[23] ^ P20[23] ^ P21[23];
assign C1723_inst = (P19[23] & P20[23]) | (P20[23] & P21[23]) | (
    P19[23] & P21[23]);
assign S1724 = P20[24] ^ P21[24] ^ P22[24];
assign C1724_inst = (P20[24] & P21[24]) | (P21[24] & P22[24]) | (
    P20[24] & P22[24]);
```

```
assign S1725 = P21[25] ^ P22[25];
assign C1725_inst = P21[25] & P22[25];

assign S1822 = P21[22] ^ P22[22];
assign C1822_inst = P21[22] & P22[22];

//Second Stage

assign S22 = S12 ^ C11;
assign C22 = S12 & C11;
assign S23 = S13 ^ C12 ^ P3[3];
assign C23 = (S13 & C12) | (C12 & P3[3]) | (S13 & P3[3]);
assign S24 = S114 ^ C13 ^ S124;
assign C24 = (S114 & C13) | (C13 & S124) | (S114 & S124);
assign S25 = S115 ^ C114 ^ C124;
assign C25 = (S115 & C114) | (C114 & C124) | (S115 & C124);
assign S216 = S116 ^ C115 ^ C125;
assign C216 = (S116 & C115) | (C115 & C125) | (S116 & C125);
assign S217 = S117 ^ C116 ^ C126;
assign C217 = (S117 & C116) | (C116 & C126) | (S117 & C126);
assign S218 = S118 ^ C117 ^ C127;
assign C218 = (S118 & C117) | (C117 & C127) | (S118 & C127);
assign S219 = S119 ^ C118 ^ C128;
assign C219 = (S119 & C118) | (C118 & C128) | (S119 & C128);
assign S2110 = S1110 ^ C119 ^ C129;
assign C2110_inst = (S1110 & C119) | (C119 & C129) | (S1110 &
    C129);
assign S2111 = S1111 ^ C1110_inst ^ C1210_inst;
assign C2111_inst = (S1111 & C1110_inst) | (C1110_inst &
    C1210_inst) | (S1111 & C1210_inst);
assign S2112 = S1112 ^ C1111_inst ^ C1211_inst;
assign C2112_inst = (S1112 & C1111_inst) | (C1111_inst &
    C1211_inst) | (S1112 & C1211_inst);
assign S2113 = S1113 ^ C1112_inst ^ C1212_inst;
assign C2113_inst = (S1113 & C1112_inst) | (C1112_inst &
    C1212_inst) | (S1113 & C1212_inst);
assign S2114 = S1114 ^ C1113_inst ^ C1213_inst;
assign C2114_inst = (S1114 & C1113_inst) | (C1113_inst &
    C1213_inst) | (S1114 & C1213_inst);
assign S2115 = S1115 ^ C1114_inst ^ C1214_inst;
```

```
assign C2115_inst = (S1115 & C1114_inst) | (C1114_inst & C1214_inst) | (S1115 & C1214_inst);
assign S2116 = S1116 ^ C1115_inst ^ C1215_inst;
assign C2116_inst = (S1116 & C1115_inst) | (C1115_inst & C1215_inst) | (S1116 & C1215_inst);
assign S2117 = S1117 ^ C1116_inst ^ C1216_inst;
assign C2117_inst = (S1117 & C1116_inst) | (C1116_inst & C1216_inst) | (S1117 & C1216_inst);
assign S2118 = S1118 ^ C1117_inst ^ C1217_inst;
assign C2118_inst = (S1118 & C1117_inst) | (C1117_inst & C1217_inst) | (S1118 & C1217_inst);
assign S2119 = S1119 ^ C1118_inst ^ C1218_inst;
assign C2119_inst = (S1119 & C1118_inst) | (C1118_inst & C1218_inst) | (S1119 & C1218_inst);
assign S2120 = S1120 ^ C1119_inst ^ C1219_inst;
assign C2120_inst = (S1120 & C1119_inst) | (C1119_inst & C1219_inst) | (S1120 & C1219_inst);
assign S2121 = S1121 ^ C1120_inst ^ C1220_inst;
assign C2121_inst = (S1121 & C1120_inst) | (C1120_inst & C1220_inst) | (S1121 & C1220_inst);
assign S2122 = S1122 ^ C1121_inst ^ C1221_inst;
assign C2122_inst = (S1122 & C1121_inst) | (C1121_inst & C1221_inst) | (S1122 & C1221_inst);
assign S2123 = S1123 ^ C1122_inst ^ C1222_inst;
assign C2123_inst = (S1123 & C1122_inst) | (C1122_inst & C1222_inst) | (S1123 & C1222_inst);
assign S2124 = S1124 ^ C1123_inst ^ C1223_inst;
assign C2124_inst = (S1124 & C1123_inst) | (C1123_inst & C1223_inst) | (S1124 & C1223_inst);
assign S2125 = S1125 ^ C1124_inst ^ C1224_inst;
assign C2125_inst = (S1125 & C1124_inst) | (C1124_inst & C1224_inst) | (S1125 & C1224_inst);
assign S2126 = S1126 ^ C1125_inst ^ C1225_inst;
assign C2126_inst = (S1126 & C1125_inst) | (C1125_inst & C1225_inst) | (S1126 & C1225_inst);
assign S2127 = S1127 ^ C1126_inst ^ C1226_inst;
assign C2127_inst = (S1127 & C1126_inst) | (C1126_inst & C1226_inst) | (S1127 & C1226_inst);
assign S2128 = S1128 ^ C1127_inst ^ C1227_inst;
assign C2128_inst = (S1128 & C1127_inst) | (C1127_inst & C1227_inst) | (S1128 & C1227_inst);
```

```
assign S2129 = S1129 ^ C1128_inst ^ C1228_inst;
assign C2129_inst = (S1129 & C1128_inst) | (C1128_inst &
    C1228_inst) | (S1129 & C1228_inst);
assign S2130 = S1130 ^ C1129_inst ^ C1229_inst;
assign C2130_inst = (S1130 & C1129_inst) | (C1129_inst &
    C1229_inst) | (S1130 & C1229_inst);
assign S2131 = S1131 ^ C1130_inst ^ C1230_inst;
assign C2131_inst = (S1131 & C1130_inst) | (C1130_inst &
    C1230_inst) | (S1131 & C1230_inst);
assign S2132 = S1132 ^ C1131_inst ^ C1231_inst;
assign C2132_inst = (S1132 & C1131_inst) | (C1131_inst &
    C1231_inst) | (S1132 & C1231_inst);
assign S2133 = S1133 ^ C1132_inst ^ C1232_inst;
assign C2133_inst = (S1133 & C1132_inst) | (C1132_inst &
    C1232_inst) | (S1133 & C1232_inst);
assign S2134 = S1134 ^ C1133_inst ^ C1233_inst;
assign C2134_inst = (S1134 & C1133_inst) | (C1133_inst &
    C1233_inst) | (S1134 & C1233_inst);
assign S2135 = S1135 ^ C1134_inst ^ C1234_inst;
assign C2135_inst = (S1135 & C1134_inst) | (C1134_inst &
    C1234_inst) | (S1135 & C1234_inst);
assign S2136 = S1136 ^ C1135_inst ^ C1235_inst;
assign C2136_inst = (S1136 & C1135_inst) | (C1135_inst &
    C1235_inst) | (S1136 & C1235_inst);
assign S2137 = S1137 ^ C1136_inst ^ C1236_inst;
assign C2137_inst = (S1137 & C1136_inst) | (C1136_inst &
    C1236_inst) | (S1137 & C1236_inst);
assign S2138 = S1138 ^ C1137_inst ^ C1237_inst;
assign C2138_inst = (S1138 & C1137_inst) | (C1137_inst &
    C1237_inst) | (S1138 & C1237_inst);
assign S2139 = S1139 ^ C1138_inst ^ C1238_inst;
assign C2139_inst = (S1139 & C1138_inst) | (C1138_inst &
    C1238_inst) | (S1139 & C1238_inst);
assign S2140 = S1140 ^ C1139_inst ^ C1239_inst;
assign C2140_inst = (S1140 & C1139_inst) | (C1139_inst &
    C1239_inst) | (S1140 & C1239_inst);
assign S2141 = S1141 ^ C1140_inst ^ C1240_inst;
assign C2141_inst = (S1141 & C1140_inst) | (C1140_inst &
    C1240_inst) | (S1141 & C1240_inst);

assign S226 = S126 ^ P6[6];
```

```
assign C226 = S126 & P6[6];
assign S227 = S127 ^ S137;
assign C227 = S127 & S137;
assign S228 = C137 ^ S128 ^ S138;
assign C228 = (C137 & S128) | (S128 & S138) | (C137 & S138);
assign S229 = C138 ^ S129 ^ S139;
assign C229 = (C138 & S129) | (S129 & S139) | (C138 & S139);
assign S2210 = C139 ^ S1210 ^ S1310;
assign C2210_inst = (C139 & S1210) | (S1210 & S1310) | (C139 & S1310);
assign S2211 = C1310_inst ^ C1410_inst ^ S1211;
assign C2211_inst = (C1310_inst & C1410_inst) | (C1410_inst & S1211) | (C1310_inst & S1211);
assign S2212 = C1311_inst ^ C1411_inst ^ S1212;
assign C2212_inst = (C1311_inst & C1411_inst) | (C1411_inst & S1212) | (C1311_inst & S1212);
assign S2213 = C1312_inst ^ C1412_inst ^ S1213;
assign C2213_inst = (C1312_inst & C1412_inst) | (C1412_inst & S1213) | (C1312_inst & S1213);
assign S2214 = C1313_inst ^ C1413_inst ^ C1513_inst;
assign C2214_inst = (C1313_inst & C1413_inst) | (C1413_inst & C1513_inst) | (C1313_inst & C1513_inst);
assign S2215 = C1314_inst ^ C1414_inst ^ C1514_inst;
assign C2215_inst = (C1314_inst & C1414_inst) | (C1414_inst & C1514_inst) | (C1314_inst & C1514_inst);
assign S2216 = C1315_inst ^ C1415_inst ^ C1515_inst;
assign C2216_inst = (C1315_inst & C1415_inst) | (C1415_inst & C1515_inst) | (C1315_inst & C1515_inst);
assign S2217 = C1316_inst ^ C1416_inst ^ C1516_inst;
assign C2217_inst = (C1316_inst & C1416_inst) | (C1416_inst & C1516_inst) | (C1316_inst & C1516_inst);
assign S2218 = C1317_inst ^ C1417_inst ^ C1517_inst;
assign C2218_inst = (C1317_inst & C1417_inst) | (C1417_inst & C1517_inst) | (C1317_inst & C1517_inst);
assign S2219 = C1318_inst ^ C1418_inst ^ C1518_inst;
assign C2219_inst = (C1318_inst & C1418_inst) | (C1418_inst & C1518_inst) | (C1318_inst & C1518_inst);
assign S2220 = C1319_inst ^ C1419_inst ^ C1519_inst;
assign C2220_inst = (C1319_inst & C1419_inst) | (C1419_inst & C1519_inst) | (C1319_inst & C1519_inst);
assign S2221 = C1320_inst ^ C1420_inst ^ C1520_inst;
```

```
assign C2221_inst = (C1320_inst & C1420_inst) | (C1420_inst &
    C1520_inst) | (C1320_inst & C1520_inst);
assign S2222 = C1321_inst ^ C1421_inst ^ C1521_inst;
assign C2222_inst = (C1321_inst & C1421_inst) | (C1421_inst &
    C1521_inst) | (C1321_inst & C1521_inst);
assign S2223 = C1322_inst ^ C1422_inst ^ C1522_inst;
assign C2223_inst = (C1322_inst & C1422_inst) | (C1422_inst &
    C1522_inst) | (C1322_inst & C1522_inst);
assign S2224 = C1323_inst ^ C1423_inst ^ C1523_inst;
assign C2224_inst = (C1323_inst & C1423_inst) | (C1423_inst &
    C1523_inst) | (C1323_inst & C1523_inst);
assign S2225 = C1324_inst ^ C1424_inst ^ C1524_inst;
assign C2225_inst = (C1324_inst & C1424_inst) | (C1424_inst &
    C1524_inst) | (C1324_inst & C1524_inst);
assign S2226 = C1325_inst ^ C1425_inst ^ C1525_inst;
assign C2226_inst = (C1325_inst & C1425_inst) | (C1425_inst &
    C1525_inst) | (C1325_inst & C1525_inst);
assign S2227 = C1326_inst ^ C1426_inst ^ C1526_inst;
assign C2227_inst = (C1326_inst & C1426_inst) | (C1426_inst &
    C1526_inst) | (C1326_inst & C1526_inst);
assign S2228 = C1327_inst ^ C1427_inst ^ C1527_inst;
assign C2228_inst = (C1327_inst & C1427_inst) | (C1427_inst &
    C1527_inst) | (C1327_inst & C1527_inst);
assign S2229 = C1328_inst ^ C1428_inst ^ C1528_inst;
assign C2229_inst = (C1328_inst & C1428_inst) | (C1428_inst &
    C1528_inst) | (C1328_inst & C1528_inst);
assign S2230 = C1329_inst ^ C1429_inst ^ C1529_inst;
assign C2230_inst = (C1329_inst & C1429_inst) | (C1429_inst &
    C1529_inst) | (C1329_inst & C1529_inst);
assign S2231 = C1330_inst ^ C1430_inst ^ C1530_inst;
assign C2231_inst = (C1330_inst & C1430_inst) | (C1430_inst &
    C1530_inst) | (C1330_inst & C1530_inst);
assign S2232 = C1331_inst ^ C1431_inst ^ C1531_inst;
assign C2232_inst = (C1331_inst & C1431_inst) | (C1431_inst &
    C1531_inst) | (C1331_inst & C1531_inst);
assign S2233 = C1332_inst ^ C1432_inst ^ S1233;
assign C2233_inst = (C1332_inst & C1432_inst) | (C1432_inst &
    S1233) | (C1332_inst & S1233);
assign S2234 = C1333_inst ^ C1433_inst ^ S1234;
assign C2234_inst = (C1333_inst & C1433_inst) | (C1433_inst &
    S1234) | (C1333_inst & S1234);
```

```
assign S2235 = C1334_inst ^ C1434_inst ^ S1235;
assign C2235_inst = (C1334_inst & C1434_inst) | (C1434_inst &
    S1235) | (C1334_inst & S1235);
assign S2236 = C1335_inst ^ S1236 ^ S1336;
assign C2236_inst = (C1335_inst & S1236) | (S1236 & S1336) | (
    C1335_inst & S1336);
assign S2237 = C1336_inst ^ S1237 ^ S1337;
assign C2237_inst = (C1336_inst & S1237) | (S1237 & S1337) | (
    C1336_inst & S1337);
assign S2238 = C1337_inst ^ S1238 ^ P22[38];
assign C2238_inst = (C1337_inst & S1238) | (S1238 & P22[38]) | (
    C1337_inst & P22[38]);

assign S2311 = S1311 ^ S1411;
assign C2311_inst = S1311 & S1411;
assign S2312 = S1312 ^ S1412 ^ P12[12];
assign C2312_inst = (S1312 & S1412) | (S1412 & P12[12]) | (S1312
    & P12[12]);
assign S2313 = S1313 ^ S1413 ^ S1513;
assign C2313_inst = (S1313 & S1413) | (S1413 & S1513) | (S1313 &
    S1513);
assign S2314 = S1214 ^ S1314 ^ S1414;
assign C2314_inst = (S1214 & S1314) | (S1314 & S1414) | (S1214 &
    S1414);
assign S2315 = S1215 ^ S1315 ^ S1415;
assign C2315_inst = (S1215 & S1315) | (S1315 & S1415) | (S1215 &
    S1415);
assign S2316 = S1216 ^ S1316 ^ S1416;
assign C2316_inst = (S1216 & S1316) | (S1316 & S1416) | (S1216 &
    S1416);
assign S2317 = C1616_inst ^ S1217 ^ S1317;
assign C2317_inst = (C1616_inst & S1217) | (S1217 & S1317) | (
    C1616_inst & S1317);
assign S2318 = C1617_inst ^ S1218 ^ S1318;
assign C2318_inst = (C1617_inst & S1218) | (S1218 & S1318) | (
    C1617_inst & S1318);
assign S2319 = C1618_inst ^ S1219 ^ S1319;
assign C2319_inst = (C1618_inst & S1219) | (S1219 & S1319) | (
    C1618_inst & S1319);
assign S2320 = C1619_inst ^ C1719_inst ^ S1220;
```

```
assign C2320_inst = (C1619_inst & C1719_inst) | (C1719_inst & S1220) | (C1619_inst & S1220);
assign S2321 = C1620_inst ^ C1720_inst ^ S1221;
assign C2321_inst = (C1620_inst & C1720_inst) | (C1720_inst & S1221) | (C1620_inst & S1221);
assign S2322 = C1621_inst ^ C1721_inst ^ S1222;
assign C2322_inst = (C1621_inst & C1721_inst) | (C1721_inst & S1222) | (C1621_inst & S1222);
assign S2323 = C1622_inst ^ C1722_inst ^ C1822_inst;
assign C2323_inst = (C1622_inst & C1722_inst) | (C1722_inst & C1822_inst) | (C1622_inst & C1822_inst);
assign S2324 = C1623_inst ^ C1723_inst ^ S1224;
assign C2324_inst = (C1623_inst & C1723_inst) | (C1723_inst & S1224) | (C1623_inst & S1224);
assign S2325 = C1624_inst ^ C1724_inst ^ S1225;
assign C2325_inst = (C1624_inst & C1724_inst) | (C1724_inst & S1225) | (C1624_inst & S1225);
assign S2326 = C1625_inst ^ C1725_inst ^ S1226;
assign C2326_inst = (C1625_inst & C1725_inst) | (C1725_inst & S1226) | (C1625_inst & S1226);
assign S2327 = C1626_inst ^ S1227 ^ S1327;
assign C2327_inst = (C1626_inst & S1227) | (S1227 & S1327) | (C1626_inst & S1327);
assign S2328 = C1627_inst ^ S1228 ^ S1328;
assign C2328_inst = (C1627_inst & S1228) | (S1228 & S1328) | (C1627_inst & S1328);
assign S2329 = C1628_inst ^ S1229 ^ S1329;
assign C2329_inst = (C1628_inst & S1229) | (S1229 & S1329) | (C1628_inst & S1329);
assign S2330 = S1230 ^ S1330 ^ S1430;
assign C2330_inst = (S1230 & S1330) | (S1330 & S1430) | (S1230 & S1430);
assign S2331 = S1231 ^ S1331 ^ S1431;
assign C2331_inst = (S1231 & S1331) | (S1331 & S1431) | (S1231 & S1431);
assign S2332 = S1232 ^ S1332 ^ S1432;
assign C2332_inst = (S1232 & S1332) | (S1332 & S1432) | (S1232 & S1432);
assign S2333 = S1333 ^ S1433;
assign C2333_inst = S1333 & S1433;
assign S2334 = S1334 ^ S1434;
```

```
assign C2334_inst = S1334 & S1434;
assign S2335 = S1335 ^ P22[35];
assign C2335_inst = S1335 & P22[35];

assign S2415 = S1515 ^ P15[15];
assign C2415_inst = S1515 & P15[15];
assign S2416 = S1516 ^ S1616;
assign C2416_inst = S1516 & S1616;
assign S2417 = S1417 ^ S1517 ^ S1617;
assign C2417_inst = (S1417 & S1517) | (S1517 & S1617) | (S1417 & S1617);
assign S2418 = S1418 ^ S1518 ^ S1618;
assign C2418_inst = (S1418 & S1518) | (S1518 & S1618) | (S1418 & S1618);
assign S2419 = S1419 ^ S1519 ^ S1619;
assign C2419_inst = (S1419 & S1519) | (S1519 & S1619) | (S1419 & S1619);
assign S2420 = S1320 ^ S1420 ^ S1520;
assign C2420_inst = (S1320 & S1420) | (S1420 & S1520) | (S1320 & S1520);
assign S2421 = S1321 ^ S1421 ^ S1521;
assign C2421_inst = (S1321 & S1421) | (S1421 & S1521) | (S1321 & S1521);
assign S2422 = S1322 ^ S1422 ^ S1522;
assign C2422_inst = (S1322 & S1422) | (S1422 & S1522) | (S1322 & S1522);
assign S2423 = S1223 ^ S1323 ^ S1423;
assign C2423_inst = (S1223 & S1323) | (S1323 & S1423) | (S1223 & S1423);
assign S2424 = S1324 ^ S1424 ^ S1524;
assign C2424_inst = (S1324 & S1424) | (S1424 & S1524) | (S1324 & S1524);
assign S2425 = S1325 ^ S1425 ^ S1525;
assign C2425_inst = (S1325 & S1425) | (S1425 & S1525) | (S1325 & S1525);
assign S2426 = S1326 ^ S1426 ^ S1526;
assign C2426_inst = (S1326 & S1426) | (S1426 & S1526) | (S1326 & S1526);
assign S2427 = S1427 ^ S1527 ^ S1627;
assign C2427_inst = (S1427 & S1527) | (S1527 & S1627) | (S1427 & S1627);
```

```
assign S2428 = S1428 ^ S1528 ^ S1628;
assign C2428_inst = (S1428 & S1528) | (S1528 & S1628) | (S1428 &
    S1628);
assign S2429 = S1429 ^ S1529 ^ P22[29];
assign C2429_inst = (S1429 & S1529) | (S1529 & P22[29]) | (S1429
    & P22[29]);

assign S2520 = S1620 ^ S1720;
assign C2520_inst = S1620 & S1720;
assign S2521 = S1621 ^ S1721 ^ P21[21];
assign C2521_inst = (S1621 & S1721) | (S1721 & P21[21]) | (S1621
    & P21[21]);
assign S2522 = S1622 ^ S1722 ^ S1822;
assign C2522_inst = (S1622 & S1722) | (S1722 & S1822) | (S1622 &
    S1822);
assign S2523 = S1523 ^ S1623 ^ S1723;
assign C2523_inst = (S1523 & S1623) | (S1623 & S1723) | (S1523 &
    S1723);
assign S2524 = S1624 ^ S1724;
assign C2524_inst = S1624 & S1724;
assign S2525 = S1625 ^ S1725;
assign C2525_inst = S1625 & S1725;
assign S2526 = S1626 ^ P22[26];
assign C2526_inst = S1626 & P22[26];

//Third Stage_inst

assign S33 = S23 ^ C22;
assign C33 = S23 & C22;
assign S35 = S25 ^ S125 ^ C24;
assign C35 = (S25 & S125) | (S125 & C24) | (S25 & C24);
assign S36 = S216 ^ S226 ^ C25;
assign C36 = (S216 & S226) | (S226 & C25) | (S216 & C25);
assign S37 = S217 ^ S227 ^ C216;
assign C37 = (S217 & S227) | (S227 & C216) | (S217 & C216);
assign S38 = S218 ^ S228 ^ C217;
assign C38 = (S218 & S228) | (S228 & C217) | (S218 & C217);
assign S319 = S219 ^ S229 ^ C218;
assign C319 = (S219 & S229) | (S229 & C218) | (S219 & C218);
assign S3110 = S2110 ^ S2210 ^ C219;
```

```
assign C3110_inst = (S2110 & S2210) | (S2210 & C219) | (S2110 & C219);
assign S3111 = S2111 ^ S2211 ^ S2311;
assign C3111_inst = (S2111 & S2211) | (S2211 & S2311) | (S2111 & S2311);
assign S3112 = S2112 ^ S2212 ^ S2312;
assign C3112_inst = (S2112 & S2212) | (S2212 & S2312) | (S2112 & S2312);
assign S3113 = S2113 ^ S2213 ^ S2313;
assign C3113_inst = (S2113 & S2213) | (S2213 & S2313) | (S2113 & S2313);
assign S3114 = S2114 ^ S2214 ^ S2314;
assign C3114_inst = (S2114 & S2214) | (S2214 & S2314) | (S2114 & S2314);
assign S3115 = S2115 ^ S2215 ^ S2315;
assign C3115_inst = (S2115 & S2215) | (S2215 & S2315) | (S2115 & S2315);
assign S3116 = S2116 ^ S2216 ^ S2316;
assign C3116_inst = (S2116 & S2216) | (S2216 & S2316) | (S2116 & S2316);
assign S3117 = S2117 ^ S2217 ^ S2317;
assign C3117_inst = (S2117 & S2217) | (S2217 & S2317) | (S2117 & S2317);
assign S3118 = S2118 ^ S2218 ^ S2318;
assign C3118_inst = (S2118 & S2218) | (S2218 & S2318) | (S2118 & S2318);
assign S3119 = S2119 ^ S2219 ^ S2319;
assign C3119_inst = (S2119 & S2219) | (S2219 & S2319) | (S2119 & S2319);
assign S3120 = S2120 ^ S2220 ^ S2320;
assign C3120_inst = (S2120 & S2220) | (S2220 & S2320) | (S2120 & S2320);
assign S3121 = S2121 ^ S2221 ^ S2321;
assign C3121_inst = (S2121 & S2221) | (S2221 & S2321) | (S2121 & S2321);
assign S3122 = S2122 ^ S2222 ^ S2322;
assign C3122_inst = (S2122 & S2222) | (S2222 & S2322) | (S2122 & S2322);
assign S3123 = S2123 ^ S2223 ^ S2323;
assign C3123_inst = (S2123 & S2223) | (S2223 & S2323) | (S2123 & S2323);
```

```
assign S3124 = S2124 ^ S2224 ^ S2324;
assign C3124_inst = (S2124 & S2224) | (S2224 & S2324) | (S2124 & S2324);
assign S3125 = S2125 ^ S2225 ^ S2325;
assign C3125_inst = (S2125 & S2225) | (S2225 & S2325) | (S2125 & S2325);
assign S3126 = S2126 ^ S2226 ^ S2326;
assign C3126_inst = (S2126 & S2226) | (S2226 & S2326) | (S2126 & S2326);
assign S3127 = S2127 ^ S2227 ^ S2327;
assign C3127_inst = (S2127 & S2227) | (S2227 & S2327) | (S2127 & S2327);
assign S3128 = S2128 ^ S2228 ^ S2328;
assign C3128_inst = (S2128 & S2228) | (S2228 & S2328) | (S2128 & S2328);
assign S3129 = S2129 ^ S2229 ^ S2329;
assign C3129_inst = (S2129 & S2229) | (S2229 & S2329) | (S2129 & S2329);
assign S3130 = S2130 ^ S2230 ^ S2330;
assign C3130_inst = (S2130 & S2230) | (S2230 & S2330) | (S2130 & S2330);
assign S3131 = S2131 ^ S2231 ^ S2331;
assign C3131_inst = (S2131 & S2231) | (S2231 & S2331) | (S2131 & S2331);
assign S3132 = S2132 ^ S2232 ^ S2332;
assign C3132_inst = (S2132 & S2232) | (S2232 & S2332) | (S2132 & S2332);
assign S3133 = S2133 ^ S2233 ^ S2333;
assign C3133_inst = (S2133 & S2233) | (S2233 & S2333) | (S2133 & S2333);
assign S3134 = S2134 ^ S2234 ^ S2334;
assign C3134_inst = (S2134 & S2234) | (S2234 & S2334) | (S2134 & S2334);
assign S3135 = S2135 ^ S2235 ^ S2335;
assign C3135_inst = (S2135 & S2235) | (S2235 & S2335) | (S2135 & S2335);
assign S3136 = S2136 ^ S2236 ^ C2135_inst;
assign C3136_inst = (S2136 & S2236) | (S2236 & C2135_inst) | (S2136 & C2135_inst);
assign S3137 = S2137 ^ S2237 ^ C2136_inst;
```

```
assign C3137_inst = (S2137 & S2237) | (S2237 & C2136_inst) | (S2137 & C2136_inst);
assign S3138 = S2138 ^ S2238 ^ C2137_inst;
assign C3138_inst = (S2138 & S2238) | (S2238 & C2137_inst) | (S2138 & C2137_inst);
assign S3139 = S2139 ^ C2138_inst ^ C2238_inst;
assign C3139_inst = (S2139 & C2138_inst) | (C2138_inst & C2238_inst) | (S2139 & C2238_inst);
assign S3140 = S2140 ^ C2139_inst ^ S1240;
assign C3140_inst = (S2140 & C2139_inst) | (C2139_inst & S1240) | (S2140 & S1240);
assign S3141 = S2141 ^ C2140_inst ^ P22[41];
assign C3141_inst = (S2141 & C2140_inst) | (C2140_inst & P22[41]) | (S2141 & P22[41]);
assign S3142 = S1142 ^ C1141_inst ^ C2141_inst;
assign C3142_inst = (S1142 & C1141_inst) | (C1141_inst & C2141_inst) | (S1142 & C2141_inst);
assign S3143 = S1143 ^ C1142_inst ^ C3142_inst;
assign C3143_inst = (S1143 & C1142_inst) | (C1142_inst & C3142_inst) | (S1143 & C3142_inst);
assign S3144 = P22[44] ^ C1143_inst ^ C3143_inst;
assign C3144_inst = (P22[44] & C1143_inst) | (C1143_inst & C3143_inst) | (P22[44] & C3143_inst);

assign S329 = C228 ^ P9[9];
assign C329 = C228 & P9[9];
assign S3210 = C229 ^ S1410;
assign C3210_inst = C229 & S1410;
assign S3211 = C2110_inst ^ C2210_inst;
assign C3211_inst = C2110_inst & C2210_inst;
assign S3212 = C2111_inst ^ C2211_inst ^ C2311_inst;
assign C3212_inst = (C2111_inst & C2211_inst) | (C2211_inst & C2311_inst) | (C2111_inst & C2311_inst);
assign S3213 = C2112_inst ^ C2212_inst ^ C2312_inst;
assign C3213_inst = (C2112_inst & C2212_inst) | (C2212_inst & C2312_inst) | (C2112_inst & C2312_inst);
assign S3214 = C2113_inst ^ C2213_inst ^ C2313_inst;
assign C3214_inst = (C2113_inst & C2213_inst) | (C2213_inst & C2313_inst) | (C2113_inst & C2313_inst);
assign S3215 = S2415 ^ C2114_inst ^ C2214_inst;
```

```
assign C3215_inst = (S2415 & C2114_inst) | (C2114_inst & C2214_inst) | (S2415 & C2214_inst);
assign S3216 = S2416 ^ C2115_inst ^ C2215_inst;
assign C3216_inst = (S2416 & C2115_inst) | (C2115_inst & C2215_inst) | (S2416 & C2215_inst);
assign S3217 = S2417 ^ C2116_inst ^ C2216_inst;
assign C3217_inst = (S2417 & C2116_inst) | (C2116_inst & C2216_inst) | (S2417 & C2216_inst);
assign S3218 = S2418 ^ C2117_inst ^ C2217_inst;
assign C3218_inst = (S2418 & C2117_inst) | (C2117_inst & C2217_inst) | (S2418 & C2217_inst);
assign S3219 = S2419 ^ C2118_inst ^ C2218_inst;
assign C3219_inst = (S2419 & C2118_inst) | (C2118_inst & C2218_inst) | (S2419 & C2218_inst);
assign S3220 = S2420 ^ S2520 ^ C2119_inst;
assign C3220_inst = (S2420 & S2520) | (S2520 & C2119_inst) | (S2420 & C2119_inst);
assign S3221 = S2421 ^ S2521 ^ C2120_inst;
assign C3221_inst = (S2421 & S2521) | (S2521 & C2120_inst) | (S2421 & C2120_inst);
assign S3222 = S2422 ^ S2522 ^ C2121_inst;
assign C3222_inst = (S2422 & S2522) | (S2522 & C2121_inst) | (S2422 & C2121_inst);
assign S3223 = S2423 ^ S2523 ^ C2122_inst;
assign C3223_inst = (S2423 & S2523) | (S2523 & C2122_inst) | (S2423 & C2122_inst);
assign S3224 = S2424 ^ S2524 ^ C2123_inst;
assign C3224_inst = (S2424 & S2524) | (S2524 & C2123_inst) | (S2424 & C2123_inst);
assign S3225 = S2425 ^ S2525 ^ C2124_inst;
assign C3225_inst = (S2425 & S2525) | (S2525 & C2124_inst) | (S2425 & C2124_inst);
assign S3226 = S2426 ^ S2526 ^ C2125_inst;
assign C3226_inst = (S2426 & S2526) | (S2526 & C2125_inst) | (S2426 & C2125_inst);
assign S3227 = S2427 ^ C2126_inst ^ C2226_inst;
assign C3227_inst = (S2427 & C2126_inst) | (C2126_inst & C2226_inst) | (S2427 & C2226_inst);
assign S3228 = S2428 ^ C2127_inst ^ C2227_inst;
assign C3228_inst = (S2428 & C2127_inst) | (C2127_inst & C2227_inst) | (S2428 & C2227_inst);
```

```
assign S3229 = S2429 ^ C2128_inst ^ C2228_inst;
assign C3229_inst = (S2429 & C2128_inst) | (C2128_inst & C2228_inst) | (S2429 & C2228_inst);
assign S3230 = C2129_inst ^ C2229_inst ^ C2329_inst;
assign C3230_inst = (C2129_inst & C2229_inst) | (C2229_inst & C2329_inst) | (C2129_inst & C2329_inst);
assign S3231 = C2130_inst ^ C2230_inst ^ C2330_inst;
assign C3231_inst = (C2130_inst & C2230_inst) | (C2230_inst & C2330_inst) | (C2130_inst & C2330_inst);
assign S3232 = C2131_inst ^ C2231_inst ^ C2331_inst;
assign C3232_inst = (C2131_inst & C2231_inst) | (C2231_inst & C2331_inst) | (C2131_inst & C2331_inst);
assign S3233 = C2132_inst ^ C2232_inst ^ C2332_inst;
assign C3233_inst = (C2132_inst & C2232_inst) | (C2232_inst & C2332_inst) | (C2132_inst & C2332_inst);
assign S3234 = C2133_inst ^ C2233_inst ^ C2333_inst;
assign C3234_inst = (C2133_inst & C2233_inst) | (C2233_inst & C2333_inst) | (C2133_inst & C2333_inst);
assign S3235 = C2134_inst ^ C2234_inst ^ C2334_inst;
assign C3235_inst = (C2134_inst & C2234_inst) | (C2234_inst & C2334_inst) | (C2134_inst & C2334_inst);
assign S3236 = C2235_inst ^ C2335_inst;
assign C3236_inst = C2235_inst & C2335_inst;

assign S3316 = C2315_inst ^ C2415_inst;
assign C3316_inst = C2315_inst & C2415_inst;
assign S3317 = C2316_inst ^ C2416_inst;
assign C3317_inst = C2316_inst & C2416_inst;
assign S3318 = C2317_inst ^ C2417_inst ^ P18[18];
assign C3318_inst = (C2317_inst & C2417_inst) | (C2417_inst & P18[18]) | (C2317_inst & P18[18]);
assign S3319 = C2318_inst ^ C2418_inst ^ S1719;
assign C3319_inst = (C2318_inst & C2418_inst) | (C2418_inst & S1719) | (C2318_inst & S1719);
assign S3320 = C2219_inst ^ C2319_inst ^ C2419_inst;
assign C3320_inst = (C2219_inst & C2319_inst) | (C2319_inst & C2419_inst) | (C2219_inst & C2419_inst);
assign S3321 = C2220_inst ^ C2320_inst ^ C2420_inst;
assign C3321_inst = (C2220_inst & C2320_inst) | (C2320_inst & C2420_inst) | (C2220_inst & C2420_inst);
assign S3322 = C2221_inst ^ C2321_inst ^ C2421_inst;
```

```
assign C3322_inst = (C2221_inst & C2321_inst) | (C2321_inst & C2421_inst) | (C2221_inst & C2421_inst);
assign S3323 = C2222_inst ^ C2322_inst ^ C2422_inst;
assign C3323_inst = (C2222_inst & C2322_inst) | (C2322_inst & C2422_inst) | (C2222_inst & C2422_inst);
assign S3324 = C2223_inst ^ C2323_inst ^ C2423_inst;
assign C3324_inst = (C2223_inst & C2323_inst) | (C2323_inst & C2423_inst) | (C2223_inst & C2423_inst);
assign S3325 = C2224_inst ^ C2324_inst ^ C2424_inst;
assign C3325_inst = (C2224_inst & C2324_inst) | (C2324_inst & C2424_inst) | (C2224_inst & C2424_inst);
assign S3326 = C2225_inst ^ C2325_inst ^ C2425_inst;
assign C3326_inst = (C2225_inst & C2325_inst) | (C2325_inst & C2425_inst) | (C2225_inst & C2425_inst);
assign S3327 = C2326_inst ^ C2426_inst ^ C2526_inst;
assign C3327_inst = (C2326_inst & C2426_inst) | (C2426_inst & C2526_inst) | (C2326_inst & C2526_inst);
assign S3328 = C2327_inst ^ C2427_inst;
assign C3328_inst = C2327_inst & C2427_inst;
assign S3329 = C2328_inst ^ C2428_inst;
assign C3329_inst = C2328_inst & C2428_inst;
assign S3330 = C2429_inst ^ S1530;
assign C3330_inst = C2429_inst & S1530;

assign S3423 = C2522_inst ^ P22[23];
assign C3423_inst = C2522_inst & P22[23];

//Fourth Stage

assign S44 = S24 ^ C23 ^ C33;
assign C44 = (S24 & C23) | (C23 & C33) | (S24 & C33);
assign S45 = S35 ^ C44;
assign C45 = S35 & C44;
assign S46 = S36 ^ C35 ^ C45;
assign C46 = (S36 & C35) | (C35 & C45) | (S36 & C45);
assign S47 = S37 ^ C36 ^ C226;
assign C47 = (S37 & C36) | (C36 & C226) | (S37 & C226);
assign S48 = S38 ^ C37 ^ C227;
assign C48 = (S38 & C37) | (C37 & C227) | (S38 & C227);
assign S49 = S319 ^ S329 ^ C38;
assign C49 = (S319 & S329) | (S329 & C38) | (S319 & C38);
```

```
assign S410 = S3110 ^ S3210 ^ C319;
assign C410 = (S3110 & S3210) | (S3210 & C319) | (S3110 & C319);
assign S411 = S3111 ^ S3211 ^ C3110_inst;
assign C411 = (S3111 & S3211) | (S3211 & C3110_inst) | (S3111 & C3110_inst);
assign S412 = S3112 ^ S3212 ^ C3111_inst;
assign C412 = (S3112 & S3212) | (S3212 & C3111_inst) | (S3112 & C3111_inst);
assign S413 = S3113 ^ S3213 ^ C3112_inst;
assign C413 = (S3113 & S3213) | (S3213 & C3112_inst) | (S3113 & C3112_inst);
assign S4114 = S3114 ^ S3214 ^ C3113_inst;
assign C4114_inst = (S3114 & S3214) | (S3214 & C3113_inst) | (S3114 & C3113_inst);
assign S4115 = S3115 ^ S3215 ^ C3114_inst;
assign C4115_inst = (S3115 & S3215) | (S3215 & C3114_inst) | (S3115 & C3114_inst);
assign S4116 = S3116 ^ S3216 ^ S3316;
assign C4116_inst = (S3116 & S3216) | (S3216 & S3316) | (S3116 & S3316);
assign S4117 = S3117 ^ S3217 ^ S3317;
assign C4117_inst = (S3117 & S3217) | (S3217 & S3317) | (S3117 & S3317);
assign S4118 = S3118 ^ S3218 ^ S3318;
assign C4118_inst = (S3118 & S3218) | (S3218 & S3318) | (S3118 & S3318);
assign S4119 = S3119 ^ S3219 ^ S3319;
assign C4119_inst = (S3119 & S3219) | (S3219 & S3319) | (S3119 & S3319);
assign S4120 = S3120 ^ S3220 ^ S3320;
assign C4120_inst = (S3120 & S3220) | (S3220 & S3320) | (S3120 & S3320);
assign S4121 = S3121 ^ S3221 ^ S3321;
assign C4121_inst = (S3121 & S3221) | (S3221 & S3321) | (S3121 & S3321);
assign S4122 = S3122 ^ S3222 ^ S3322;
assign C4122_inst = (S3122 & S3222) | (S3222 & S3322) | (S3122 & S3322);
assign S4123 = S3123 ^ S3223 ^ S3323;
assign C4123_inst = (S3123 & S3223) | (S3223 & S3323) | (S3123 & S3323);
```

```
assign S4124 = S3124 ^ S3224 ^ S3324;
assign C4124_inst = (S3124 & S3224) | (S3224 & S3324) | (S3124 &
    S3324);
assign S4125 = S3125 ^ S3225 ^ S3325;
assign C4125_inst = (S3125 & S3225) | (S3225 & S3325) | (S3125 &
    S3325);
assign S4126 = S3126 ^ S3226 ^ S3326;
assign C4126_inst = (S3126 & S3226) | (S3226 & S3326) | (S3126 &
    S3326);
assign S4127 = S3127 ^ S3227 ^ S3327;
assign C4127_inst = (S3127 & S3227) | (S3227 & S3327) | (S3127 &
    S3327);
assign S4128 = S3128 ^ S3228 ^ S3328;
assign C4128_inst = (S3128 & S3228) | (S3228 & S3328) | (S3128 &
    S3328);
assign S4129 = S3129 ^ S3229 ^ S3329;
assign C4129_inst = (S3129 & S3229) | (S3229 & S3329) | (S3129 &
    S3329);
assign S4130 = S3130 ^ S3230 ^ S3330;
assign C4130_inst = (S3130 & S3230) | (S3230 & S3330) | (S3130 &
    S3330);
assign S4131 = S3131 ^ S3231 ^ C3130_inst;
assign C4131_inst = (S3131 & S3231) | (S3231 & C3130_inst) | (
    S3131 & C3130_inst);
assign S4132 = S3132 ^ S3232 ^ C3131_inst;
assign C4132_inst = (S3132 & S3232) | (S3232 & C3131_inst) | (
    S3132 & C3131_inst);
assign S4133 = S3133 ^ S3233 ^ C3132_inst;
assign C4133_inst = (S3133 & S3233) | (S3233 & C3132_inst) | (
    S3133 & C3132_inst);
assign S4134 = S3134 ^ S3234 ^ C3133_inst;
assign C4134_inst = (S3134 & S3234) | (S3234 & C3133_inst) | (
    S3134 & C3133_inst);
assign S4135 = S3135 ^ S3235 ^ C3134_inst;
assign C4135_inst = (S3135 & S3235) | (S3235 & C3134_inst) | (
    S3135 & C3134_inst);
assign S4136 = S3136 ^ S3236 ^ C3135_inst;
assign C4136_inst = (S3136 & S3236) | (S3236 & C3135_inst) | (
    S3136 & C3135_inst);
assign S4137 = S3137 ^ C3136_inst ^ C3236_inst;
```

```
assign C4137_inst = (S3137 & C3136_inst) | (C3136_inst & C3236_inst) | (S3137 & C3236_inst);
assign S4138 = S3138 ^ C3137_inst ^ C2237_inst;
assign C4138_inst = (S3138 & C3137_inst) | (C3137_inst & C2237_inst) | (S3138 & C2237_inst);
assign S4139 = S3139 ^ C3138_inst ^ S1239;
assign C4139_inst = (S3139 & C3138_inst) | (C3138_inst & S1239) | (S3139 & S1239);

assign S4214 = C3213_inst ^ S1514;
assign C4214_inst = C3213_inst & S1514;
assign S4215 = C3214_inst ^ C2314_inst;
assign C4215_inst = C3214_inst & C2314_inst;
assign S4216 = C3115_inst ^ C3215_inst;
assign C4216_inst = C3115_inst & C3215_inst;
assign S4217 = C3116_inst ^ C3216_inst ^ C3316_inst;
assign C4217_inst = (C3116_inst & C3216_inst) | (C3216_inst & C3316_inst) | (C3116_inst & C3316_inst);
assign S4218 = C3117_inst ^ C3217_inst ^ C3317_inst;
assign C4218_inst = (C3117_inst & C3217_inst) | (C3217_inst & C3317_inst) | (C3117_inst & C3317_inst);
assign S4219 = C3118_inst ^ C3218_inst ^ C3318_inst;
assign C4219_inst = (C3118_inst & C3218_inst) | (C3218_inst & C3318_inst) | (C3118_inst & C3318_inst);
assign S4220 = C3119_inst ^ C3219_inst ^ C3319_inst;
assign C4220_inst = (C3119_inst & C3219_inst) | (C3219_inst & C3319_inst) | (C3119_inst & C3319_inst);
assign S4221 = C3120_inst ^ C3220_inst ^ C3320_inst;
assign C4221_inst = (C3120_inst & C3220_inst) | (C3220_inst & C3320_inst) | (C3120_inst & C3320_inst);
assign S4222 = C3121_inst ^ C3221_inst ^ C3321_inst;
assign C4222_inst = (C3121_inst & C3221_inst) | (C3221_inst & C3321_inst) | (C3121_inst & C3321_inst);
assign S4223 = S3423 ^ C3122_inst ^ C3222_inst;
assign C4223_inst = (S3423 & C3122_inst) | (C3122_inst & C3222_inst) | (S3423 & C3222_inst);
assign S4224 = C3123_inst ^ C3223_inst ^ C3323_inst;
assign C4224_inst = (C3123_inst & C3223_inst) | (C3223_inst & C3323_inst) | (C3123_inst & C3323_inst);
assign S4225 = C3124_inst ^ C3224_inst ^ C3324_inst;
```

```
assign C4225_inst = (C3124_inst & C3224_inst) | (C3224_inst &
    C3324_inst) | (C3124_inst & C3324_inst);
assign S4226 = C3125_inst ^ C3225_inst ^ C3325_inst;
assign C4226_inst = (C3125_inst & C3225_inst) | (C3225_inst &
    C3325_inst) | (C3125_inst & C3325_inst);
assign S4227 = C3126_inst ^ C3226_inst ^ C3326_inst;
assign C4227_inst = (C3126_inst & C3226_inst) | (C3226_inst &
    C3326_inst) | (C3126_inst & C3326_inst);
assign S4228 = C3127_inst ^ C3227_inst ^ C3327_inst;
assign C4228_inst = (C3127_inst & C3227_inst) | (C3227_inst &
    C3327_inst) | (C3127_inst & C3327_inst);
assign S4229 = C3128_inst ^ C3228_inst ^ C3328_inst;
assign C4229_inst = (C3128_inst & C3228_inst) | (C3228_inst &
    C3328_inst) | (C3128_inst & C3328_inst);
assign S4230 = C3129_inst ^ C3229_inst ^ C3329_inst;
assign C4230_inst = (C3129_inst & C3229_inst) | (C3229_inst &
    C3329_inst) | (C3129_inst & C3329_inst);
assign S4231 = C3230_inst ^ C3330_inst ^ S1531;
assign C4231_inst = (C3230_inst & C3330_inst) | (C3330_inst &
    S1531) | (C3230_inst & S1531);
assign S4232 = C3231_inst ^ P22[32];
assign C4232_inst = C3231_inst & P22[32];

assign S4324 = C3423_inst ^ C2523_inst;
assign C4324_inst = C3423_inst & C2523_inst;

//Fifth Stage

assign S57 = S47 ^ C46;
assign C57 = S47 & C46;
assign S58 = S48 ^ C47 ^ C57;
assign C58 = (S48 & C47) | (C47 & C57) | (S48 & C57);
assign S59 = S49 ^ C48 ^ C58;
assign C59 = (S49 & C48) | (C48 & C58) | (S49 & C58);
assign S510 = S410 ^ C49 ^ C329;
assign C510 = (S410 & C49) | (C49 & C329) | (S410 & C329);
assign S511 = S411 ^ C410 ^ C3210_inst;
assign C511 = (S411 & C410) | (C410 & C3210_inst) | (S411 &
    C3210_inst);
assign S512 = S412 ^ C411 ^ C3211_inst;
```

```
assign C512 = (S412 & C411) | (C411 & C3211_inst) | (S412 & C3211_inst);
assign S513 = S413 ^ C412 ^ C3212_inst;
assign C513 = (S413 & C412) | (C412 & C3212_inst) | (S413 & C3212_inst);
assign S514 = S4114 ^ S4214 ^ C413;
assign C514 = (S4114 & S4214) | (S4214 & C413) | (S4114 & C413);
assign S515 = S4115 ^ S4215 ^ C4114_inst;
assign C515 = (S4115 & S4215) | (S4215 & C4114_inst) | (S4115 & C4114_inst);
assign S516 = S4116 ^ S4216 ^ C4115_inst;
assign C516 = (S4116 & S4216) | (S4216 & C4115_inst) | (S4116 & C4115_inst);
assign S517 = S4117 ^ S4217 ^ C4116_inst;
assign C517 = (S4117 & S4217) | (S4217 & C4116_inst) | (S4117 & C4116_inst);
assign S518 = S4118 ^ S4218 ^ C4117_inst;
assign C518 = (S4118 & S4218) | (S4218 & C4117_inst) | (S4118 & C4117_inst);
assign S519 = S4119 ^ S4219 ^ C4118_inst;
assign C519 = (S4119 & S4219) | (S4219 & C4118_inst) | (S4119 & C4118_inst);
assign S520 = S4120 ^ S4220 ^ C4119_inst;
assign C520 = (S4120 & S4220) | (S4220 & C4119_inst) | (S4120 & C4119_inst);
assign S5121 = S4121 ^ S4221 ^ C4120_inst;
assign C5121_inst = (S4121 & S4221) | (S4221 & C4120_inst) | (S4121 & C4120_inst);
assign S5122 = S4122 ^ S4222 ^ C4121_inst;
assign C5122_inst = (S4122 & S4222) | (S4222 & C4121_inst) | (S4122 & C4121_inst);
assign S5123 = S4123 ^ S4223 ^ C4122_inst;
assign C5123_inst = (S4123 & S4223) | (S4223 & C4122_inst) | (S4123 & C4122_inst);
assign S5124 = S4124 ^ S4224 ^ S4324;
assign C5124_inst = (S4124 & S4224) | (S4224 & S4324) | (S4124 & S4324);
assign S5125 = S4125 ^ S4225 ^ C4124_inst;
assign C5125_inst = (S4125 & S4225) | (S4225 & C4124_inst) | (S4125 & C4124_inst);
assign S5126 = S4126 ^ S4226 ^ C4125_inst;
```

```
assign C5126_inst = (S4126 & S4226) | (S4226 & C4125_inst) | (S4126 & C4125_inst);
assign S5127 = S4127 ^ S4227 ^ C4126_inst;
assign C5127_inst = (S4127 & S4227) | (S4227 & C4126_inst) | (S4127 & C4126_inst);
assign S5128 = S4128 ^ S4228 ^ C4127_inst;
assign C5128_inst = (S4128 & S4228) | (S4228 & C4127_inst) | (S4128 & C4127_inst);
assign S5129 = S4129 ^ S4229 ^ C4128_inst;
assign C5129_inst = (S4129 & S4229) | (S4229 & C4128_inst) | (S4129 & C4128_inst);
assign S5130 = S4130 ^ S4230 ^ C4129_inst;
assign C5130_inst = (S4130 & S4230) | (S4230 & C4129_inst) | (S4130 & C4129_inst);
assign S5131 = S4131 ^ S4231 ^ C4130_inst;
assign C5131_inst = (S4131 & S4231) | (S4231 & C4130_inst) | (S4131 & C4130_inst);
assign S5132 = S4132 ^ S4232 ^ C4131_inst;
assign C5132_inst = (S4132 & S4232) | (S4232 & C4131_inst) | (S4132 & C4131_inst);
assign S5133 = S4133 ^ C4132_inst ^ C4232_inst;
assign C5133_inst = (S4133 & C4132_inst) | (C4132_inst & C4232_inst) | (S4133 & C4232_inst);
assign S5134 = S4134 ^ C4133_inst ^ C3233_inst;
assign C5134_inst = (S4134 & C4133_inst) | (C4133_inst & C3233_inst) | (S4134 & C3233_inst);
assign S5135 = S4135 ^ C4134_inst ^ C3234_inst;
assign C5135_inst = (S4135 & C4134_inst) | (C4134_inst & C3234_inst) | (S4135 & C3234_inst);
assign S5136 = S4136 ^ C4135_inst ^ C3235_inst;
assign C5136_inst = (S4136 & C4135_inst) | (C4135_inst & C3235_inst) | (S4136 & C3235_inst);
assign S5137 = S4137 ^ C4136_inst ^ C2236_inst;
assign C5137_inst = (S4137 & C4136_inst) | (C4136_inst & C2236_inst) | (S4137 & C2236_inst);

assign S5221 = C4220_inst ^ C2520_inst;
assign C5221_inst = C4220_inst & C2520_inst;
assign S5222 = C4221_inst ^ C2521_inst;
assign C5222_inst = C4221_inst & C2521_inst;
assign S5223 = C4222_inst ^ C3322_inst;
```

```
assign C5223_inst = C4222_inst & C3322_inst;
assign S5224 = C4123_inst ^ C4223_inst;
assign C5224_inst = C4123_inst & C4223_inst;
assign S5225 = C4224_inst ^ C4324_inst ^ C2524_inst;
assign C5225_inst = (C4224_inst & C4324_inst) | (C4324_inst & C2524_inst) | (C4224_inst & C2524_inst);
assign S5226 = C4225_inst ^ C2525_inst;
assign C5226_inst = C4225_inst & C2525_inst;

//Sixth Stage

assign S610 = S510 ^ C59;
assign C610 = S510 & C59;
assign S611 = S511 ^ C510 ^ C610;
assign C611 = (S511 & C510) | (C510 & C610) | (S511 & C610);
assign S612 = S512 ^ C511 ^ C611;
assign C612 = (S512 & C511) | (C511 & C611) | (S512 & C611);
assign S613 = S513 ^ C512 ^ C612;
assign C613 = (S513 & C512) | (C512 & C612) | (S513 & C612);
assign S614 = S514 ^ C513 ^ C613;
assign C614 = (S514 & C513) | (C513 & C613) | (S514 & C613);
assign S615 = S515 ^ C514 ^ C4214_inst;
assign C615 = (S515 & C514) | (C514 & C4214_inst) | (S515 & C4214_inst);
assign S616 = S516 ^ C515 ^ C4215_inst;
assign C616 = (S516 & C515) | (C515 & C4215_inst) | (S516 & C4215_inst);
assign S617 = S517 ^ C516 ^ C4216_inst;
assign C617 = (S517 & C516) | (C516 & C4216_inst) | (S517 & C4216_inst);
assign S618 = S518 ^ C517 ^ C4217_inst;
assign C618 = (S518 & C517) | (C517 & C4217_inst) | (S518 & C4217_inst);
assign S619 = S519 ^ C518 ^ C4218_inst;
assign C619 = (S519 & C518) | (C518 & C4218_inst) | (S519 & C4218_inst);
assign S620 = S520 ^ C519 ^ C4219_inst;
assign C620 = (S520 & C519) | (C519 & C4219_inst) | (S520 & C4219_inst);
assign S621 = S5121 ^ S5221 ^ C520;
assign C621 = (S5121 & S5221) | (S5221 & C520) | (S5121 & C520);
```

```
assign S622 = S5122 ^ S5222 ^ C5121_inst;
assign C622 = (S5122 & S5222) | (S5222 & C5121_inst) | (S5122 & C5121_inst);
assign S623 = S5123 ^ S5223 ^ C5122_inst;
assign C623 = (S5123 & S5223) | (S5223 & C5122_inst) | (S5123 & C5122_inst);
assign S624 = S5124 ^ S5224 ^ C5123_inst;
assign C624 = (S5124 & S5224) | (S5224 & C5123_inst) | (S5124 & C5123_inst);
assign S625 = S5125 ^ S5225 ^ C5124_inst;
assign C625 = (S5125 & S5225) | (S5225 & C5124_inst) | (S5125 & C5124_inst);
assign S626 = S5126 ^ S5226 ^ C5125_inst;
assign C626 = (S5126 & S5226) | (S5226 & C5125_inst) | (S5126 & C5125_inst);
assign S627 = S5127 ^ C5126_inst ^ C5226_inst;
assign C627 = (S5127 & C5126_inst) | (C5126_inst & C5226_inst) | (S5127 & C5226_inst);
assign S628 = S5128 ^ C5127_inst ^ C4227_inst;
assign C628 = (S5128 & C5127_inst) | (C5127_inst & C4227_inst) | (S5128 & C4227_inst);
assign S629 = S5129 ^ C5128_inst ^ C4228_inst;
assign C629 = (S5129 & C5128_inst) | (C5128_inst & C4228_inst) | (S5129 & C4228_inst);
assign S630 = S5130 ^ C5129_inst ^ C4229_inst;
assign C630 = (S5130 & C5129_inst) | (C5129_inst & C4229_inst) | (S5130 & C4229_inst);
assign S631 = S5131 ^ C5130_inst ^ C4230_inst;
assign C631 = (S5131 & C5130_inst) | (C5130_inst & C4230_inst) | (S5131 & C4230_inst);
assign S632 = S5132 ^ C5131_inst ^ C4231_inst;
assign C632 = (S5132 & C5131_inst) | (C5131_inst & C4231_inst) | (S5132 & C4231_inst);
assign S633 = S5133 ^ C5132_inst ^ C3232_inst;
assign C633 = (S5133 & C5132_inst) | (C5132_inst & C3232_inst) | (S5133 & C3232_inst);
assign S634 = S5134 ^ C5133_inst ^ C633;
assign C634 = (S5134 & C5133_inst) | (C5133_inst & C633) | (S5134 & C633);
assign S635 = S5135 ^ C5134_inst ^ C634;
```

```
assign C635 = (S5135 & C5134_inst) | (C5134_inst & C634) | (S5135 & C634);
assign S636 = S5136 ^ C5135_inst ^ C635;
assign C636 = (S5136 & C5135_inst) | (C5135_inst & C635) | (S5136 & C635);
assign S637 = S5137 ^ C5136_inst ^ C636;
assign C637 = (S5137 & C5136_inst) | (C5136_inst & C636) | (S5137 & C636);
assign S638 = S4138 ^ C5137_inst ^ C637;
assign C638 = (S4138 & C5137_inst) | (C5137_inst & C637) | (S4138 & C637);
assign S639 = S4139 ^ C4138_inst ^ C638;
assign C639 = (S4139 & C4138_inst) | (C4138_inst & C638) | (S4139 & C638);
assign S640 = S3140 ^ C4139_inst ^ C639;
assign C640 = (S3140 & C4139_inst) | (C4139_inst & C639) | (S3140 & C639);
assign S641 = S3141 ^ C3140_inst ^ C640;
assign C641 = (S3141 & C3140_inst) | (C3140_inst & C640) | (S3141 & C640);
assign S642 = S3142 ^ C3141_inst ^ C641;
assign C642 = (S3142 & C3141_inst) | (C3141_inst & C641) | (S3142 & C641);
assign S643 = S3143 ^ C642;
assign C643 = S3143 & C642;
assign S644 = S3144 ^ C643;
assign C644 = S3144 & C643;

//Seventh Stage

assign S715 = S615 ^ C614;
assign C715 = S615 & C614;
assign S716 = S616 ^ C615 ^ C715;
assign C716 = (S616 & C615) | (C615 & C715) | (S616 & C715);
assign S717 = S617 ^ C616 ^ C716;
assign C717 = (S617 & C616) | (C616 & C716) | (S617 & C716);
assign S718 = S618 ^ C617 ^ C717;
assign C718 = (S618 & C617) | (C617 & C717) | (S618 & C717);
assign S719 = S619 ^ C618 ^ C718;
assign C719 = (S619 & C618) | (C618 & C718) | (S619 & C718);
assign S720 = S620 ^ C619 ^ C719;
```

```
assign C720 = (S620 & C619) | (C619 & C719) | (S620 & C719);
assign S721 = S621 ^ C620 ^ C720;
assign C721 = (S621 & C620) | (C620 & C720) | (S621 & C720);
assign S722 = S622 ^ C621 ^ C5221_inst;
assign C722 = (S622 & C621) | (C621 & C5221_inst) | (S622 &
    C5221_inst);
assign S723 = S623 ^ C622 ^ C5222_inst;
assign C723 = (S623 & C622) | (C622 & C5222_inst) | (S623 &
    C5222_inst);
assign S724 = S624 ^ C623 ^ C5223_inst;
assign C724 = (S624 & C623) | (C623 & C5223_inst) | (S624 &
    C5223_inst);
assign S725 = S625 ^ C624 ^ C5224_inst;
assign C725 = (S625 & C624) | (C624 & C5224_inst) | (S625 &
    C5224_inst);
assign S726 = S626 ^ C625 ^ C5225_inst;
assign C726 = (S626 & C625) | (C625 & C5225_inst) | (S626 &
    C5225_inst);
assign S727 = S627 ^ C626 ^ C4226_inst;
assign C727 = (S627 & C626) | (C626 & C4226_inst) | (S627 &
    C4226_inst);
assign S728 = S628 ^ C627;
assign C728 = S628 & C627;
assign S729 = S629 ^ C628 ^ C728;
assign C729 = (S629 & C628) | (C628 & C728) | (S629 & C728);
assign S730 = S630 ^ C629 ^ C729;
assign C730 = (S630 & C629) | (C629 & C729) | (S630 & C729);
assign S731 = S631 ^ C630 ^ C730;
assign C731 = (S631 & C630) | (C630 & C730) | (S631 & C730);
assign S732 = S632 ^ C631 ^ C731;
assign C732 = (S632 & C631) | (C631 & C731) | (S632 & C731);
assign S733 = S633 ^ C632 ^ C732;
assign C733 = (S633 & C632) | (C632 & C732) | (S633 & C732);
assign S734 = S634 ^ C733;
assign C734 = S634 & C733;
assign S735 = S635 ^ C734;
assign C735 = S635 & C734;
assign S736 = S636 ^ C735;
assign C736 = S636 & C735;
assign S737 = S637 ^ C736;
assign C737 = S637 & C736;
```

```
assign S738 = S638 ^ C737;
assign C738 = S638 & C737;
assign S739 = S639 ^ C738;
assign C739 = S639 & C738;
assign S740 = S640 ^ C739;
assign C740 = S640 & C739;
assign S741 = S641 ^ C740;
assign C741 = S641 & C740;
assign S742 = S642 ^ C741;
assign C742 = S642 & C741;
assign S743 = S643 ^ C742;
assign C743 = S643 & C742;
assign S744 = S644 ^ C743;
assign C744 = S644 & C743;
assign S745 = C644 ^ C744;
assign C745 = C644 & C744;

//Eighth Stage

assign S822 = S722 ^ C721;
assign C822 = S722 & C721;
assign S823 = S723 ^ C722 ^ C822;
assign C823 = (S723 & C722) | (C722 & C822) | (S723 & C822);
assign S824 = S724 ^ C723 ^ C823;
assign C824 = (S724 & C723) | (C723 & C823) | (S724 & C823);
assign S825 = S725 ^ C724 ^ C824;
assign C825 = (S725 & C724) | (C724 & C824) | (S725 & C824);
assign S826 = S726 ^ C725 ^ C825;
assign C826 = (S726 & C725) | (C725 & C825) | (S726 & C825);
assign S827 = S727 ^ C726 ^ C826;
assign C827 = (S727 & C726) | (C726 & C826) | (S727 & C826);
assign S828 = S728 ^ C727 ^ C827;
assign C828 = (S728 & C727) | (C727 & C827) | (S728 & C827);
assign S829 = S729 ^ C828;
assign C829 = S729 & C828;
assign S830 = S730 ^ C829;
assign C830 = S730 & C829;
assign S831 = S731 ^ C830;
assign C831 = S731 & C830;
assign S832 = S732 ^ C831;
assign C832 = S732 & C831;
```

```
assign S833 = S733 ^ C832;
assign C833 = S733 & C832;
assign S834 = S734 ^ C833;
assign C834 = S734 & C833;
assign S835 = S735 ^ C834;
assign C835 = S735 & C834;
assign S836 = S736 ^ C835;
assign C836 = S736 & C835;
assign S837 = S737 ^ C836;
assign C837 = S737 & C836;
assign S838 = S738 ^ C837 ^ C4137_inst;
assign C838 = (S738 & C837) | (C837 & C4137_inst) | (S738 &
    C4137_inst);
assign S839 = S739 ^ C838;
assign C839 = S739 & C838;
assign S840 = S740 ^ C839 ^ C3139_inst;
assign C840 = (S740 & C839) | (C839 & C3139_inst) | (S740 &
    C3139_inst);
assign S841 = S741 ^ C840;
assign C841 = S741 & C840;
assign S842 = S742 ^ C841;
assign C842 = S742 & C841;
assign S843 = S743 ^ C842;
assign C843 = S743 & C842;
assign S844 = S744 ^ C843;
assign C844 = S744 & C843;
assign S845 = S745 ^ C844 ^ C3144_inst;
assign C845 = (S745 & C844) | (C844 & C3144_inst) | (S745 &
    C3144_inst);

//Nineth Stage

assign S923 = S823 ^ P23[23] ^ P0[23];
assign C923_inst = (S823 & P23[23]) | (P23[23] & P0[23]) | (S823
   & P0[23]);
assign S924 = S824 ^ P23[24] ^ P1[24];
assign C924_inst = (S824 & P23[24]) | (P23[24] & P1[24]) | (S824
   & P1[24]);
assign S925 = S825 ^ P23[25] ^ P2[25];
assign C925_inst = (S825 & P23[25]) | (P23[25] & P2[25]) | (S825
   & P2[25]);
```

```
assign S926 = S826 ^ P23[26] ^ P3[26];
assign C926_inst = (S826 & P23[26]) | (P23[26] & P3[26]) | (S826
    & P3[26]);
assign S927 = S827 ^ P23[27] ^ P4[27];
assign C927_inst = (S827 & P23[27]) | (P23[27] & P4[27]) | (S827
    & P4[27]);
assign S928 = S828 ^ P23[28] ^ P5[28];
assign C928_inst = (S828 & P23[28]) | (P23[28] & P5[28]) | (S828
    & P5[28]);
assign S929 = S829 ^ P23[29] ^ P6[29];
assign C929_inst = (S829 & P23[29]) | (P23[29] & P6[29]) | (S829
    & P6[29]);
assign S930 = S830 ^ P23[30] ^ P7[30];
assign C930_inst = (S830 & P23[30]) | (P23[30] & P7[30]) | (S830
    & P7[30]);
assign S931 = S831 ^ P23[31] ^ P8[31];
assign C931_inst = (S831 & P23[31]) | (P23[31] & P8[31]) | (S831
    & P8[31]);
assign S932 = S832 ^ P23[32] ^ P9[32];
assign C932_inst = (S832 & P23[32]) | (P23[32] & P9[32]) | (S832
    & P9[32]);
assign S933 = S833 ^ P23[33] ^ P10[33];
assign C933_inst = (S833 & P23[33]) | (P23[33] & P10[33]) | (S833
    & P10[33]);
assign S934 = S834 ^ P23[34] ^ P11[34];
assign C934_inst = (S834 & P23[34]) | (P23[34] & P11[34]) | (S834
    & P11[34]);
assign S935 = S835 ^ P23[35] ^ P12[35];
assign C935_inst = (S835 & P23[35]) | (P23[35] & P12[35]) | (S835
    & P12[35]);
assign S936 = S836 ^ P23[36] ^ P13[36];
assign C936_inst = (S836 & P23[36]) | (P23[36] & P13[36]) | (S836
    & P13[36]);
assign S937 = S837 ^ P23[37] ^ P14[37];
assign C937_inst = (S837 & P23[37]) | (P23[37] & P14[37]) | (S837
    & P14[37]);
assign S938 = S838 ^ P23[38] ^ P15[38];
assign C938_inst = (S838 & P23[38]) | (P23[38] & P15[38]) | (S838
    & P15[38]);
assign S939 = S839 ^ P23[39] ^ P16[39];
```

```
assign C939_inst = (S839 & P23[39]) | (P23[39] & P16[39]) | (S839 & P16[39]);
assign S940 = S840 ^ P23[40] ^ P17[40];
assign C940_inst = (S840 & P23[40]) | (P23[40] & P17[40]) | (S840 & P17[40]);
assign S941 = S841 ^ P23[41] ^ P18[41];
assign C941_inst = (S841 & P23[41]) | (P23[41] & P18[41]) | (S841 & P18[41]);
assign S942 = S842 ^ P23[42] ^ P19[42];
assign C942_inst = (S842 & P23[42]) | (P23[42] & P19[42]) | (S842 & P19[42]);
assign S943 = S843 ^ P23[43] ^ P20[43];
assign C943_inst = (S843 & P23[43]) | (P23[43] & P20[43]) | (S843 & P20[43]);
assign S944 = S844 ^ P23[44] ^ P21[44];
assign C944_inst = (S844 & P23[44]) | (P23[44] & P21[44]) | (S844 & P21[44]);
assign S945 = S845 ^ P23[45] ^ P22[45];
assign C945_inst = (S845 & P23[45]) | (P23[45] & P22[45]) | (S845 & P22[45]);
assign S946 = C845 ^ C745 ^ P23[46];
assign C946_inst = (C845 & C745) | (C745 & P23[46]) | (C845 & P23[46]);

//Tenth Stage

assign S1024 = S924 ^ C923_inst;
assign C1024_inst = S924 & C923_inst;
assign S1025 = S925 ^ C924_inst ^ C1024_inst;
assign C1025_inst = (S925 & C924_inst) | (C924_inst & C1024_inst) | (S925 & C1024_inst);
assign S1026 = S926 ^ C925_inst ^ C1025_inst;
assign C1026_inst = (S926 & C925_inst) | (C925_inst & C1025_inst) | (S926 & C1025_inst);
assign S1027 = S927 ^ C926_inst ^ C1026_inst;
assign C1027_inst = (S927 & C926_inst) | (C926_inst & C1026_inst) | (S927 & C1026_inst);
assign S1028 = S928 ^ C927_inst ^ C1027_inst;
assign C1028_inst = (S928 & C927_inst) | (C927_inst & C1027_inst) | (S928 & C1027_inst);
assign S1029 = S929 ^ C928_inst ^ C1028_inst;
```

```
assign C1029_inst = (S929 & C928_inst) | (C928_inst & C1028_inst)
    | (S929 & C1028_inst);
assign S1030 = S930 ^ C929_inst ^ C1029_inst;
assign C1030_inst = (S930 & C929_inst) | (C929_inst & C1029_inst)
    | (S930 & C1029_inst);
assign S1031 = S931 ^ C930_inst ^ C1030_inst;
assign C1031_inst = (S931 & C930_inst) | (C930_inst & C1030_inst)
    | (S931 & C1030_inst);
assign S1032 = S932 ^ C931_inst ^ C1031_inst;
assign C1032_inst = (S932 & C931_inst) | (C931_inst & C1031_inst)
    | (S932 & C1031_inst);
assign S1033 = S933 ^ C932_inst ^ C1032_inst;
assign C1033_inst = (S933 & C932_inst) | (C932_inst & C1032_inst)
    | (S933 & C1032_inst);
assign S1034 = S934 ^ C933_inst ^ C1033_inst;
assign C1034_inst = (S934 & C933_inst) | (C933_inst & C1033_inst)
    | (S934 & C1033_inst);
assign S1035 = S935 ^ C934_inst ^ C1034_inst;
assign C1035_inst = (S935 & C934_inst) | (C934_inst & C1034_inst)
    | (S935 & C1034_inst);
assign S1036 = S936 ^ C935_inst ^ C1035_inst;
assign C1036_inst = (S936 & C935_inst) | (C935_inst & C1035_inst)
    | (S936 & C1035_inst);
assign S1037 = S937 ^ C936_inst ^ C1036_inst;
assign C1037_inst = (S937 & C936_inst) | (C936_inst & C1036_inst)
    | (S937 & C1036_inst);
assign S1038 = S938 ^ C937_inst ^ C1037_inst;
assign C1038_inst = (S938 & C937_inst) | (C937_inst & C1037_inst)
    | (S938 & C1037_inst);
assign S1039 = S939 ^ C938_inst ^ C1038_inst;
assign C1039_inst = (S939 & C938_inst) | (C938_inst & C1038_inst)
    | (S939 & C1038_inst);
assign S1040 = S940 ^ C939_inst ^ C1039_inst;
assign C1040_inst = (S940 & C939_inst) | (C939_inst & C1039_inst)
    | (S940 & C1039_inst);
assign S1041 = S941 ^ C940_inst ^ C1040_inst;
assign C1041_inst = (S941 & C940_inst) | (C940_inst & C1040_inst)
    | (S941 & C1040_inst);
assign S1042 = S942 ^ C941_inst ^ C1041_inst;
assign C1042_inst = (S942 & C941_inst) | (C941_inst & C1041_inst)
    | (S942 & C1041_inst);
```

```
assign S1043 = S943 ^ C942_inst ^ C1042_inst;
assign C1043_inst = (S943 & C942_inst) | (C942_inst & C1042_inst)
    | (S943 & C1042_inst);
assign S1044 = S944 ^ C943_inst ^ C1043_inst;
assign C1044_inst = (S944 & C943_inst) | (C943_inst & C1043_inst)
    | (S944 & C1043_inst);
assign S1045 = S945 ^ C944_inst ^ C1044_inst;
assign C1045_inst = (S945 & C944_inst) | (C944_inst & C1044_inst)
    | (S945 & C1044_inst);
assign S1046 = S946 ^ C945_inst ^ C1045_inst;
assign C1046_inst = (S946 & C945_inst) | (C945_inst & C1045_inst)
    | (S946 & C1045_inst);
assign S1047 = C946_inst ^ C1046_inst;
assign C1047_inst = C946_inst & C1046_inst;

endmodule // mvk_mult
```

I.4 Floating Point Subtractor

```
module mvk_sub (
           reset,
           clk,
           scan_in0,
           scan_en,
           test_mode,
           scan_out0,
           input_a,
           input_b,
           output_z
       );

input
    reset,                      // system reset
    clk;                        // system clock

input
    scan_in0,                   // test scan mode data input
    scan_en,                    // test scan mode enable
```

```
    test_mode;                    // test mode select

output
    scan_out0;                    // test scan mode data output

input
    [31:0] input_a;

input
    [31:0] input_b;

output reg
    [31:0] output_z;

reg
    [31:0] z_8;
reg
    [26:0] a_m_1, a_m_2, b_m_1, b_m_2;

reg
    [23:0] z_m_7, z_m_6, z_m_5, z_m_4;
reg
    [9:0] a_e_1, a_e_2, b_e_1, b_e_2, z_e_7, z_e_6, z_e_5, z_e_4, z_e_3;

reg
    a_s_1, a_s_2, b_s_1, b_s_2, z_s_7, z_s_6, z_s_5, z_s_4, z_s_3;

reg
    guard_6, guard_5, guard_4, round_bit_6, round_bit_5, round_bit_4, sticky_6, sticky_5, sticky_4;

reg
    [27:0] sum_3;

reg
    [8:0]abc;

reg
    [8:0]xyz;
```

```

reg stall_mc10 , stall_mc1 , stall_mc2 , stall_mc3 , stall_mc4 ,
    stall_mc5 , stall_mc6 , stall_mc7 , stall_mc8 , stall_mc9;

always @(posedge clk or posedge reset)
begin
    if (reset == 1)
    begin
        output_z <= 32'h0000_0000;
        abc = 0;
        xyz = 0;
        stall_mc1 = 1'b1;
        stall_mc2 = 1'b0;
        stall_mc3 = 1'b0;
        stall_mc4 = 1'b0;
        stall_mc5 = 1'b0;
        stall_mc6 = 1'b0;
        stall_mc7 = 1'b0;
        stall_mc8 = 1'b0;
        stall_mc9 = 1'b0;
        stall_mc10 = 1'b0;
        a_m_1 <= 0;
        b_m_1 <= 0;
        a_e_1 <= 0;
        b_e_1 <= 0;
        a_s_1 <= 0;
        b_s_1 <= 0;
        a_m_1[26] <= 0;
    end

    else
    begin

        if(stall_mc9 == 1'b1)
        begin
            output_z <= z_8;
        end
        else
        begin
            stall_mc10 = 1'b1;
        end
```

```

        if(stall_mc8 == 1'b1)
        begin
            z_8[22 : 0] <= z_m_7[22:0];
            z_8[30 : 23] <= z_e_7[7:0] + 127;
            z_8[31] <= z_s_7;
            if ($signed(z_e_7) == -126 && z_m_7[23] == 0)
            begin
                z_8[30 : 23] <= 0;
            end
            //if overflow occurs, return inf
            if ($signed(z_e_7) > 127)
            begin
                z_8[22 : 0] <= 0;
                z_8[30 : 23] <= 255;
                z_8[31] <= z_s_7;
            end
            stall_mc9 = 1'b1;
        end

        if(stall_mc7 == 1'b1)
        begin
            if (guard_6 && (round_bit_6 | sticky_6 | z_m_6[0]))
            begin
                z_m_7 <= z_m_6 + 1;
                if (z_m_6 == 24'hffffff)
                begin
                    z_e_7 <=z_e_6 + 1;
                end
                else
                begin
                    z_e_7 <= z_e_6;
                end
            end
            else
            begin
                z_m_7 <= z_m_6;
                z_s_7 <= z_s_6;
                z_e_7 <= z_e_6;
            end
            z_s_7 <= z_s_6;
```

```
                stall_mc8 = 1'b1;
            end

            if(stall_mc6 == 1'b1)
            begin
                if ($signed(z_e_5) < -126)
                begin
                    xyz = -126 - $signed(z_e_5);
                    case(xyz)
                        1:
                        begin
                            z_e_6 <= z_e_5 + 1;
                            z_m_6 <= z_m_5 >> 1;
                            guard_6 <= z_m_5[0];
                            round_bit_6 <= guard_5;
                            sticky_6 <= sticky_5 | round_bit_5;
                        end
                        2:
                        begin
                            z_e_6 <= z_e_5 + 2;
                            z_m_6 <= z_m_5 >> 2;
                            guard_6 <= z_m_5[1];
                            round_bit_6 <= z_m_5[0];
                            sticky_6 <= sticky_5 | z_m_5[1];
                        end
                        3:
                        begin
                            z_e_6 <= z_e_5 + 3;
                            z_m_6 <= z_m_5 >> 3;
                            guard_6 <= z_m_5[2];
                            round_bit_6 <= z_m_5[1];
                            sticky_6 <= sticky_5 | z_m_5[2];
                        end
                        4:
                        begin
                            z_e_6 <= z_e_5 + 4;
                            z_m_6 <= z_m_5 >> 4;
                            guard_6 <= z_m_5[3];
                            round_bit_6 <= z_m_5[2];
                            sticky_6 <= sticky_5 | z_m_5[3];
                        end
```

```
5:
begin
    z_e_6 <= z_e_5 + 5;
    z_m_6 <= z_m_5 >> 5;
    guard_6 <= z_m_5[4];
    round_bit_6 <= z_m_5[3];
    sticky_6 <= sticky_5 | z_m_5[4];
end
6:
begin
    z_e_6 <= z_e_5 + 6;
    z_m_6 <= z_m_5 >> 6;
    guard_6 <= z_m_5[5];
    round_bit_6 <= z_m_5[4];
    sticky_6 <= sticky_5 | z_m_5[5];
end
7:
begin
    z_e_6 <= z_e_5 + 7;
    z_m_6 <= z_m_5 >> 7;
    guard_6 <= z_m_5[6];
    round_bit_6 <= z_m_5[5];
    sticky_6 <= sticky_5 | z_m_5[6];
end
8:
begin
    z_e_6 <= z_e_5 + 8;
    z_m_6 <= z_m_5 >> 8;
    guard_6 <= z_m_5[7];
    round_bit_6 <= z_m_5[6];
    sticky_6 <= sticky_5 | z_m_5[7];
end
9:
begin
    z_e_6 <= z_e_5 + 9;
    z_m_6 <= z_m_5 >> 9;
    guard_6 <= z_m_5[8];
    round_bit_6 <= z_m_5[7];
    sticky_6 <= sticky_5 | z_m_5[8];
end
10:
```

```
            begin
                z_e_6 <= z_e_5 + 10;
                z_m_6 <= z_m_5 >> 10;
                guard_6 <= z_m_5[9];
                round_bit_6 <= z_m_5[8];
                sticky_6 <= sticky_5 | z_m_5[9];
            end
            11:
            begin
                z_e_6 <= z_e_5 + 11;
                z_m_6 <= z_m_5 >> 11;
                guard_6 <= z_m_5[10];
                round_bit_6 <= z_m_5[9];
                sticky_6 <= sticky_5 | z_m_5[10];
            end
            12:
            begin
                z_e_6 <= z_e_5 + 12;
                z_m_6 <= z_m_5 >> 12;
                guard_6 <= z_m_5[11];
                round_bit_6 <= z_m_5[10];
                sticky_6 <= sticky_5 | z_m_5[11];
            end
            13:
            begin
                z_e_6 <= z_e_5 + 13;
                z_m_6 <= z_m_5 >> 13;
                guard_6 <= z_m_5[12];
                round_bit_6 <= z_m_5[11];
                sticky_6 <= sticky_5 | z_m_5[12];
            end
            14:
            begin
                z_e_6 <= z_e_5 + 14;
                z_m_6 <= z_m_5 >> 14;
                guard_6 <= z_m_5[13];
                round_bit_6 <= z_m_5[12];
                sticky_6 <= sticky_5 | z_m_5[13];
            end
            15:
            begin
```

```
                        z_e_6 <= z_e_5 + 15;
                        z_m_6 <= z_m_5 >> 15;
                        guard_6 <= z_m_5[14];
                        round_bit_6 <= z_m_5[13];
                        sticky_6 <= sticky_5 | z_m_5[14];
                    end
                    16:
                    begin
                        z_e_6 <= z_e_5 + 16;
                        z_m_6 <= z_m_5 >> 16;
                        guard_6 <= z_m_5[15];
                        round_bit_6 <= z_m_5[14];
                        sticky_6 <= sticky_5 | z_m_5[15];
                    end
                    17:
                    begin
                        z_e_6 <= z_e_5 + 17;
                        z_m_6 <= z_m_5 >> 17;
                        guard_6 <= z_m_5[16];
                        round_bit_6 <= z_m_5[15];
                        sticky_6 <= sticky_5 | z_m_5[16];
                    end
                    18:
                    begin
                        z_e_6 <= z_e_5 + 18;
                        z_m_6 <= z_m_5 >> 18;
                        guard_6 <= z_m_5[17];
                        round_bit_6 <= z_m_5[16];
                        sticky_6 <= sticky_5 | z_m_5[17];
                    end
                    19:
                    begin
                        z_e_6 <= z_e_5 + 19;
                        z_m_6 <= z_m_5 >> 19;
                        guard_6 <= z_m_5[18];
                        round_bit_6 <= z_m_5[17];
                        sticky_6 <= sticky_5 | z_m_5[18];
                    end
                    20:
                    begin
                        z_e_6 <= z_e_5 + 20;
```

```
                        z_m_6 <= z_m_5 >> 20;
                        guard_6 <= z_m_5[19];
                        round_bit_6 <= z_m_5[18];
                        sticky_6 <= sticky_5 | z_m_5[19];
                    end
                    21:
                    begin
                        z_e_6 <= z_e_5 + 21;
                        z_m_6 <= z_m_5 >> 21;
                        guard_6 <= z_m_5[20];
                        round_bit_6 <= z_m_5[19];
                        sticky_6 <= sticky_5 | z_m_5[20];
                    end
                    22:
                    begin
                        z_e_6 <= z_e_5 + 22;
                        z_m_6 <= z_m_5 >> 22;
                        guard_6 <= z_m_5[21];
                        round_bit_6 <= z_m_5[20];
                        sticky_6 <= sticky_5 | z_m_5[21];
                    end
                    23:
                    begin
                        z_e_6 <= z_e_5 + 23;
                        z_m_6 <= z_m_5 >> 23;
                        guard_6 <= z_m_5[22];
                        round_bit_6 <= z_m_5[21];
                        sticky_6 <= sticky_5 | z_m_5[22];
                    end
                endcase
                z_s_6 <= z_s_5;
                stall_mc7 = 1'b1;
            end
            else
            begin
                z_e_6 <= z_e_5;
                z_m_6 <= z_m_5;
                guard_6 <= guard_5;
                round_bit_6 <= round_bit_5;
                sticky_6 <= sticky_5;
                z_s_6 <= z_s_5;
```

```
                stall_mc7 = 1'b1;
            end
        end

        if(stall_mc5 == 1'b1)
        begin
                casex(z_m_4)
                    24'b1xxx_xxxx_xxxx_xxxx_xxxx_xxxx:
                    begin
                        z_m_5 <= z_m_4;
                        z_e_5 <= z_e_4;
                        z_m_5[0] <= z_m_4[0];
                        guard_5 <= guard_4;
                        round_bit_5 <= round_bit_4;
                        z_s_5 <= z_s_4;
                        sticky_5 <= sticky_4;
                        stall_mc6 = 1'b1;
                    end
                    24'b01xx_xxxx_xxxx_xxxx_xxxx_xxxx:
                    begin
                        z_m_5 <= z_m_4 << 1;
                        z_e_5 <= z_e_4 - 1;
                        z_m_5[0] <= guard_4;
                        guard_5 <= round_bit_4;
                        round_bit_5 <= 0;
                    end
                    24'b001x_xxxx_xxxx_xxxx_xxxx_xxxx:
                    begin
                        z_m_5 <= z_m_4 << 2;
                        z_e_5 <= z_e_4 - 2;
                        z_m_5[0] <= round_bit_4;
                        guard_5 <= 0;
                        round_bit_5 <= 0;
                    end
                    24'b0001_xxxx_xxxx_xxxx_xxxx_xxxx:
                    begin
                        z_m_5 <= z_m_4 << 3;
                        z_e_5 <= z_e_4 - 3;
                        z_m_5[0] <= 0;
                        guard_5 <= 0;
                        round_bit_5 <= 0;
```

```
                    end
                    24'b0000_1xxx_xxxx_xxxx_xxxx_xxxx:
                    begin
                        z_m_5 <= z_m_4 << 4;
                        z_e_5 <= z_e_4 - 4;
                        z_m_5[0] <= 0;
                        guard_5 <= 0;
                        round_bit_5 <= 0;
                    end
                    24'b0000_01xx_xxxx_xxxx_xxxx_xxxx:
                    begin
                        z_m_5 <= z_m_4 << 5;
                        z_e_5 <= z_e_4 - 5;
                        z_m_5[0] <= 0;
                        guard_5 <= 0;
                        round_bit_5 <= 0;
                    end
                    24'b0000_001x_xxxx_xxxx_xxxx_xxxx:
                    begin
                        z_m_5 <= z_m_4 << 6;
                        z_e_5 <= z_e_4 - 6;
                        z_m_5[0] <= 0;
                        guard_5 <= 0;
                        round_bit_5 <= 0;
                    end
                    24'b0000_0001_xxxx_xxxx_xxxx_xxxx:
                    begin
                        z_m_5 <= z_m_4 << 7;
                        z_e_5 <= z_e_4 - 7;
                        z_m_5[0] <= 0;
                        guard_5 <= 0;
                        round_bit_5 <= 0;
                    end
                    24'b0000_0000_1xxx_xxxx_xxxx_xxxx:
                    begin
                        z_m_5 <= z_m_4 << 8;
                        z_e_5 <= z_e_4 - 8;
                        z_m_5[0] <= 0;
                        guard_5 <= 0;
                        round_bit_5 <= 0;
                    end
```

```
            24'b0000_0000_01xx_xxxx_xxxx_xxxx:
            begin
                z_m_5 <= z_m_4 << 9;
                z_e_5 <= z_e_4 - 9;
                z_m_5[0] <= 0;
                guard_5 <= 0;
                round_bit_5 <= 0;
            end
            24'b0000_0000_001x_xxxx_xxxx_xxxx:
            begin
                z_m_5 <= z_m_4 << 10;
                z_e_5 <= z_e_4 - 10;
                z_m_5[0] <= 0;
                guard_5 <= 0;
                round_bit_5 <= 0;
            end
            24'b0000_0000_0001_xxxx_xxxx_xxxx:
            begin
                z_m_5 <= z_m_4 << 11;
                z_e_5 <= z_e_4 - 11;
                z_m_5[0] <= 0;
                guard_5 <= 0;
                round_bit_5 <= 0;
            end
            24'b0000_0000_0000_1xxx_xxxx_xxxx:
            begin
                z_m_5 <= z_m_4 << 12;
                z_e_5 <= z_e_4 - 12;
                z_m_5[0] <= 0;
                guard_5 <= 0;
                round_bit_5 <= 0;
            end
            24'b0000_0000_0000_01xx_xxxx_xxxx:
            begin
                z_m_5 <= z_m_4 << 13;
                z_e_5 <= z_e_4 - 13;
                z_m_5[0] <= 0;
                guard_5 <= 0;
                round_bit_5 <= 0;
            end
            24'b0000_0000_0000_001x_xxxx_xxxx:
```

```
            begin
                z_m_5 <= z_m_4 << 14;
                z_e_5 <= z_e_4 - 14;
                z_m_5[0] <= 0;
                guard_5 <= 0;
                round_bit_5 <= 0;
            end
            24'b0000_0000_0000_0001_xxxx_xxxx:
            begin
                z_m_5 <= z_m_4 << 15;
                z_e_5 <= z_e_4 - 15;
                z_m_5[0] <= 0;
                guard_5 <= 0;
                round_bit_5 <= 0;
            end
            24'b0000_0000_0000_0000_1xxx_xxxx:
            begin
                z_m_5 <= z_m_4 << 16;
                z_e_5 <= z_e_4 - 16;
                z_m_5[0] <= 0;
                guard_5 <= 0;
                round_bit_5 <= 0;
            end
            24'b0000_0000_0000_0000_01xx_xxxx:
            begin
                z_m_5 <= z_m_4 << 17;
                z_e_5 <= z_e_4 - 17;
                z_m_5[0] <= 0;
                guard_5 <= 0;
                round_bit_5 <= 0;
            end
            24'b0000_0000_0000_0000_001x_xxxx:
            begin
                z_m_5 <= z_m_4 << 18;
                z_e_5 <= z_e_4 - 18;
                z_m_5[0] <= 0;
                guard_5 <= 0;
                round_bit_5 <= 0;
            end
            24'b0000_0000_0000_0000_0001_xxxx:
            begin
```

```
                z_m_5 <= z_m_4 << 19;
                z_e_5 <= z_e_4 - 19;
                z_m_5[0] <= 0;
                guard_5 <= 0;
                round_bit_5 <= 0;
            end
            24'b0000_0000_0000_0000_0000_1xxx:
            begin
                z_m_5 <= z_m_4 << 20;
                z_e_5 <= z_e_4 - 20;
                z_m_5[0] <= 0;
                guard_5 <= 0;
                round_bit_5 <= 0;
            end
            24'b0000_0000_0000_0000_0000_01xx:
            begin
                z_m_5 <= z_m_4 << 21;
                z_e_5 <= z_e_4 - 21;
                z_m_5[0] <= 0;
                guard_5 <= 0;
                round_bit_5 <= 0;
            end
            24'b0000_0000_0000_0000_0000_001x:
            begin
                z_m_5 <= z_m_4 << 22;
                z_e_5 <= z_e_4 - 22;
                z_m_5[0] <= 0;
                guard_5 <= 0;
                round_bit_5 <= 0;
            end
            24'b0000_0000_0000_0000_0000_0001:
            begin
                z_m_5 <= z_m_4 << 23;
                z_e_5 <= z_e_4 - 23;
                z_m_5[0] <= 0;
                guard_5 <= 0;
                round_bit_5 <= 0;
            end
        endcase
        z_s_5 <= z_s_4;
        sticky_5 <= sticky_4;
```

```
            stall_mc6 = 1'b1;
        end

        if(stall_mc4 == 1'b1)
        begin
            if (sum_3[27])
            begin
                z_m_4 <= sum_3[27:4];
                guard_4 <= sum_3[3];
                round_bit_4 <= sum_3[2];
                sticky_4 <= sum_3[1] | sum_3[0];
                z_e_4 <= z_e_3 + 1;
            end
            else
            begin
                z_m_4 <= sum_3[26:3];
                z_e_4 <= z_e_3;
                guard_4 <= sum_3[2];
                round_bit_4 <= sum_3[1];
                sticky_4 <= sum_3[0];
            end
            z_s_4 <= z_s_3;
            stall_mc5 = 1'b1;
        end

        if(stall_mc3 == 1'b1)
        begin
            z_e_3 <= a_e_2;
            if (a_s_2 == b_s_2)
            begin
                sum_3 <= a_m_2 - b_m_2;
                z_s_3 <= a_s_2;
            end
            else
            begin
                if (a_m_2 >= b_m_2)
                begin
                    sum_3 <= a_m_2 - b_m_2;
                    z_s_3 <= a_s_2;
                end
                else
```

```
                    begin
                        sum_3 <= b_m_2 - a_m_2;
                        z_s_3 <= b_s_2;
                    end
                end

                stall_mc4 = 1'b1;
            end

            if(stall_mc2 == 1'b1)
            begin
                if ($signed(a_e_1) > $signed(b_e_1))
                begin
                    abc = a_e_1 - b_e_1;
                    case(abc)
                        8'd1: b_m_2 <= b_m_1 >> 1;
                        8'd2: b_m_2 <= b_m_1 >> 2;
                        8'd3: b_m_2 <= b_m_1 >> 3;
                        8'd4: b_m_2 <= b_m_1 >> 4;
                        8'd5: b_m_2 <= b_m_1 >> 5;
                        8'd6: b_m_2 <= b_m_1 >> 6;
                        8'd7: b_m_2 <= b_m_1 >> 7;
                        8'd8: b_m_2 <= b_m_1 >> 8;
                        8'd9: b_m_2 <= b_m_1 >> 9;
                        8'd10: b_m_2 <= b_m_1 >> 10;
                        8'd11: b_m_2 <= b_m_1 >> 11;
                        8'd12: b_m_2 <= b_m_1 >> 12;
                        8'd13: b_m_2 <= b_m_1 >> 13;
                        8'd14: b_m_2 <= b_m_1 >> 14;
                        8'd15: b_m_2 <= b_m_1 >> 15;
                        8'd16: b_m_2 <= b_m_1 >> 16;
                        8'd17: b_m_2 <= b_m_1 >> 17;
                        8'd18: b_m_2 <= b_m_1 >> 18;
                        8'd19: b_m_2 <= b_m_1 >> 19;
                        8'd20: b_m_2 <= b_m_1 >> 20;
                        8'd21: b_m_2 <= b_m_1 >> 21;
                        8'd22: b_m_2 <= b_m_1 >> 22;
                        8'd23: b_m_2 <= b_m_1 >> 23;
                        8'd24: b_m_2 <= b_m_1 >> 24;
                        8'd25: b_m_2 <= b_m_1 >> 25;
                        8'd26: b_m_2 <= b_m_1 >> 26;
```

```
                default:
                begin
                    b_m_2 <= 1;
                    b_e_2 <= a_e_1;
                end
            endcase
            a_m_2 <= a_m_1;
            b_e_2 <= a_e_1;
            a_e_2 <= a_e_1;
            stall_mc3 = 1'b1;
        end
        else if ($signed(a_e_1) < $signed(b_e_1))
        begin
            abc = b_e_1 - a_e_1;
            case(abc)
                8'd1: a_m_2 <= a_m_1 >> 1;
                8'd2: a_m_2 <= a_m_1 >> 2;
                8'd3: a_m_2 <= a_m_1 >> 3;
                8'd4: a_m_2 <= a_m_1 >> 4;
                8'd5: a_m_2 <= a_m_1 >> 5;
                8'd6: a_m_2 <= a_m_1 >> 6;
                8'd7: a_m_2 <= a_m_1 >> 7;
                8'd8: a_m_2 <= a_m_1 >> 8;
                8'd9: a_m_2 <= a_m_1 >> 9;
                8'd10: a_m_2 <= a_m_1 >> 10;
                8'd11: a_m_2 <= a_m_1 >> 11;
                8'd12: a_m_2 <= a_m_1 >> 12;
                8'd13: a_m_2 <= a_m_1 >> 13;
                8'd14: a_m_2 <= a_m_1 >> 14;
                8'd15: a_m_2 <= a_m_1 >> 15;
                8'd16: a_m_2 <= a_m_1 >> 16;
                8'd17: a_m_2 <= a_m_1 >> 17;
                8'd18: a_m_2 <= a_m_1 >> 18;
                8'd19: a_m_2 <= a_m_1 >> 19;
                8'd20: a_m_2 <= a_m_1 >> 20;
                8'd21: a_m_2 <= a_m_1 >> 21;
                8'd22: a_m_2 <= a_m_1 >> 22;
                8'd23: a_m_2 <= a_m_1 >> 23;
                8'd24: a_m_2 <= a_m_1 >> 24;
                8'd25: a_m_2 <= a_m_1 >> 25;
                8'd26: a_m_2 <= a_m_1 >> 26;
```

```
                    default:
                    begin
                        a_m_2 <= 1;
                        a_e_2 <= b_e_1;
                    end
                endcase
                b_m_2 <= b_m_1;
                a_e_2 <= b_e_1;
                b_e_2 <= b_e_1;
                stall_mc3 = 1'b1;
            end
            else
            begin
                a_m_2 <= a_m_1;
                b_m_2 <= b_m_1;
                a_e_2 <= a_e_1;
                b_e_2 <= b_e_1;
                stall_mc3 = 1'b1;
            end
            a_s_2 <= a_s_1;
            b_s_2 <= b_s_1;
        end

        if(stall_mc1 == 1'b1)
        begin
            a_m_1 <= {input_a[22 : 0], 3'd0};
            b_m_1 <= {input_b[22 : 0], 3'd0};
            a_e_1 <= input_a[30 : 23] - 127;
            b_e_1 <= input_b[30 : 23] - 127;
            a_s_1 <= input_a[31];
            b_s_1 <= input_b[31];
            a_m_1[26] <= 1;

            if(input_b[30:23] == 0)
            begin
                b_e_1 <= -126;
            end
            else
            begin
                b_m_1[26] <= 1;
            end
```

```
                stall_mc2 = 1'b1;
            end
        end
end
endmodule
```

I.5 Subtracter Module

```
module mvk_subtractor (
            reset ,
            clk ,
            scan_in0 ,
            scan_en ,
            test_mode ,
            number_1 ,
            number_2 ,
            answer ,
            scan_out0
        );

input
    reset ,                     // system reset
    clk;                        // system clock

input
    scan_in0 ,                  // test scan mode data input
    scan_en ,                   // test scan mode enable
    test_mode;                  // test mode select

output
    scan_out0;                  // test scan mode data output

input
    [11:0] number_1;

input
    [10:0] number_2;

```

```
output
      [31:0]answer;

wire
      [23:0]inverse_2;

wire
      [23:0]sum;

wire
      [23:0]carry;

wire
      [23:0]big_number;

wire
      [23:0]number_2_inverse;

wire
      [7:0]exponent;

wire
      [22:0]a_m;

//big number contains the yh part.
//The input number is padded with 10'b0 at the end
assign big_number = {1'b1, number_1[11], number_1[10], number_1[9], number_1[8], number_1[7], number_1[6], number_1[5], number_1[4], number_1[3], number_1[2], number_1[1], number_1[0], 11'b00000000000};

//inverse_2 contains the inverse of input number 2 which is yl and padded with 1's at the start.
assign inverse_2 = {2'b00, number_2, 11'b00000000000};

assign answer = (reset == 1'b1) ? 32'h0000_0000 : {1'b0, exponent, a_m};

assign sum = big_number - inverse_2;
assign a_m = (sum <= 24'b0000_0000_0000_0000_0000_0000) ?  24'd0 :
```

```
(sum <= 24'b0000_0000_0000_0000_0000_0001) ? sum << 23 :
(sum <= 24'b0000_0000_0000_0000_0000_0011) ? sum << 22 :
(sum <= 24'b0000_0000_0000_0000_0000_0111) ? sum << 21 :
(sum <= 24'b0000_0000_0000_0000_0000_1111) ? sum << 20 :
(sum <= 24'b0000_0000_0000_0000_0001_1111) ? sum << 19 :
(sum <= 24'b0000_0000_0000_0000_0011_1111) ? sum << 18 :
(sum <= 24'b0000_0000_0000_0000_0111_1111) ? sum << 17 :
(sum <= 24'b0000_0000_0000_0000_1111_1111) ? sum << 16 :
(sum <= 24'b0000_0000_0000_0001_1111_1111) ? sum << 15 :
(sum <= 24'b0000_0000_0000_0011_1111_1111) ? sum << 14 :
(sum <= 24'b0000_0000_0000_0111_1111_1111) ? sum << 13 :
(sum <= 24'b0000_0000_0000_1111_1111_1111) ? sum << 12 :
(sum <= 24'b0000_0000_0001_1111_1111_1111) ? sum << 11 :
(sum <= 24'b0000_0000_0011_1111_1111_1111) ? sum << 10 :
(sum <= 24'b0000_0000_0111_1111_1111_1111) ? sum << 9 :
(sum <= 24'b0000_0000_1111_1111_1111_1111) ? sum << 8 :
(sum <= 24'b0000_0001_1111_1111_1111_1111) ? sum << 7 :
(sum <= 24'b0000_0011_1111_1111_1111_1111) ? sum << 6 :
(sum <= 24'b0000_0111_1111_1111_1111_1111) ? sum << 5 :
(sum <= 24'b0000_1111_1111_1111_1111_1111) ? sum << 4 :
```

```
                    (sum <= 24'b0001_1111_1111_1111_1111_1111) ? sum << 3 :
                    (sum <= 24'b0011_1111_1111_1111_1111_1111) ? sum << 2 :
                    (sum <= 24'b0111_1111_1111_1111_1111_1111) ? sum << 1 :
                    (sum <= 24'b1111_1111_1111_1111_1111_1111) ? sum << 0 : 24'd0;

assign exponent = (sum <= 24'b0000_0000_0000_0000_0000_0000) ? 8'd0 :
                          (sum <= 24'b0000_0000_0000_0000_0000_0001) ? 8'd150 :
                          (sum <= 24'b0000_0000_0000_0000_0000_0011) ? 8'd149 :
                          (sum <= 24'b0000_0000_0000_0000_0000_0111) ? 8'd148 :
                          (sum <= 24'b0000_0000_0000_0000_0000_1111) ? 8'd147 :
                          (sum <= 24'b0000_0000_0000_0000_0001_1111) ? 8'd146 :
                          (sum <= 24'b0000_0000_0000_0000_0011_1111) ? 8'd145 :
                          (sum <= 24'b0000_0000_0000_0000_0111_1111) ? 8'd144 :
                          (sum <= 24'b0000_0000_0000_0000_1111_1111) ? 8'd143 :
                          (sum <= 24'b0000_0000_0000_0001_1111_1111) ? 8'd142 :
                          (sum <= 24'b0000_0000_0000_0011_1111_1111) ? 8'd141 :
                          (sum <= 24'b0000_0000_0000_0111_1111_1111) ? 8'd140 :
                          (sum <= 24'b0000_0000_0000_1111_1111_1111) ? 8'd139 :
                          (sum <= 24'b0000_0000_0001_1111_1111_1111) ? 8'd138 :
                          (sum <= 24'b0000_0000_0011_1111_1111_1111) ? 8'd137 :
                          (sum <= 24'b0000_0000_0111_1111_1111_1111) ? 8'd136 :
```

```
                        (sum <= 24'b0000_0000_1111_1111_1111_1111) ? 8'd135 :
                        (sum <= 24'b0000_0001_1111_1111_1111_1111) ? 8'd134 :
                        (sum <= 24'b0000_0011_1111_1111_1111_1111) ? 8'd133 :
                        (sum <= 24'b0000_0111_1111_1111_1111_1111) ? 8'd132 :
                        (sum <= 24'b0000_1111_1111_1111_1111_1111) ? 8'd131 :
                        (sum <= 24'b0001_1111_1111_1111_1111_1111) ? 8'd130 :
                        (sum <= 24'b0011_1111_1111_1111_1111_1111) ? 8'd129 :
                        (sum <= 24'b0111_1111_1111_1111_1111_1111) ? 8'd128 :
                        (sum <= 24'b1111_1111_1111_1111_1111_1111) ? 8'd127 : 8'd0;

endmodule // mvk_subtractor
```