# Towards Implementing Carry-Free Primitives for Prime Field

- *Vikram Voleti, KU Leuven, Summer 2013*

## 1. ABSTRACT

The implementation of Prime Field Arithmetic Primitives for Elliptic Curve Cryptography comprises of the design of circuits for addition, subtraction, multiplication and modular reduction. A new method has been used to perform Addition in such a way as to eliminate the use of 'carry.' This new way of addition has been incorporated in other arithmetic operations viz. subtraction, multiplication, and modular reduction.

## 2. ADDITION WITH CARRY PROPAGATION

School-book addition, although it does not consume much area, is quite time-intensive. The Synthesis results of performing this addition on two 256-bit numbers in Xilinx 9 on a Virtex-5 FPGA are:

**Maximum combinational path delay: 71.306ns**
Number of Slice LUTs: 384 out of 19200 2%
Number of LUT Flip Flop pairs used: 384

## 3. RECODED BINARY SIGNED DIGIT NUMBERS

The input binary numbers are converted to a "Recoded Binary Signed Digit" [RBSD] format that utilizes three digits to express numbers: -1, 0, 1. The inclusion of the third digit -1 allows for fast computation of sum, and has been effectively used to eliminate the notorious 'carry' which is the major cause of delays in computations.

### 3.1 Binary to RBSD Conversion

The conversion from binary to RBSD is carried out using a simple truth table, which takes as input two consecutive bits in the original binary number, and produces the RBSD value to replace the more significant of the two input bits. Since this process need only utilize all the bits independently, it can be parallelized to significantly reduce computation time.

The truth table used to convert binary numbers to RBSD numbers is:

| $A_i$ | $A_{i-1}$ | $X_i$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | -1 |
| 1 | 1 | 0 |

Table 1

For example, the binary number 10110 is converted to $1\bar{1}10\ \bar{1}0$. As can be seen the RBSD number contains one bit more than the original binary number.

The conversion to RBSD takes place such that the value of the number remains the same, but there are no two consecutive 1's or -1's in the number. This, in effect, eliminates the possibility of a Carry beyond one position.

## 3.2 BSD to RBSD Conversion

The conversion of a Binary Signed Digit (BSD) number to its RBSD form is a more complex process, because a binary number has only two possible digits: 0 and1, while a BSD number has three possible inputs: -1, 0 and 1.

Table 2[1] has been followed to convert any BSD number to an RBSD number:

| $A_i$ | $A_{i-1}$ | $A_{i-2}$ | $A_{i-3}$ | $X_i$ |
|---|---|---|---|---|
| -1 | -1 | -1 | X | 0 |
| | | 0 | -1 | 0 |
| | | 0 | 0 | 1 |
| | | 0 | 1 | 1 |
| | | 1 | X | 1 |
| -1 | 0 | -1 | X | 1 |
| | | 0 | X | -1 |
| | | 1 | X | -1 |
| -1 | 1 | -1 | X | -1 |
| | | 0 | -1 | -1 |
| | | 0 | 0 | 0 |
| | | 0 | 1 | 0 |
| | | 1 | X | 0 |
| 0 | -1 | -1 | X | -1 |
| | | 0 | -1 | -1 |
| | | 0 | 0 | 0 |
| | | 0 | 1 | 0 |
| | | 1 | X | 0 |
| 0 | 0 | X | X | 0 |
| 0 | 1 | -1 | X | 0 |
| | | 0 | -1 | 0 |
| | | 0 | 0 | 1 |
| | | 0 | 1 | 1 |
| | | 1 | X | 1 |
| 1 | -1 | -1 | X | 0 |
| | | 0 | -1 | 0 |
| | | 0 | 0 | 1 |
| | | 0 | 1 | 1 |
| | | 1 | X | 1 |
| 1 | 0 | -1 | X | 1 |
| | | 0 | X | -1 |
| | | 1 | X | -1 |
| 1 | 1 | -1 | X | -1 |
| | | 0 | -1 | -1 |
| | | 0 | 0 | 0 |
| | | 0 | 1 | 0 |
| | | 1 | X | 0 |

Table 2

Table 2 has been given in Behrooz Parhami's paper "Carry-Free Addition of Recoded Binary Signed-Digit Numbers", which is the main basis for the work carried out. Table 1 has been logically designed by simplifying this table to include only 0 and 1 as input digits.

It is important to note that the field of RBSD numbers is not closed under addition. This means that the addition of two RBSD numbers shall result in a BSD sum, but which is not necessarily an RBSD number. Thus, a Carry-Free Addition module, of binary numbers or of RBSD numbers, needs to be followed by a BSD-to-RBSD Converter.

## 3.3 BSD to Binary Conversion

The following table is used to convert a BSD number, could be RBSD, into its Binary form. The process involves carry propagation, and the table details the conversion of one bit of the BSD digit depending on the carry bit at its position (0 for the least significant) into one bit at the same position in its Binary form, and a carry bit for the next digit.

| BSD (Z) | Carry_in (c) | Bin (B) | Carry_out (C) |
|---|---|---|---|
| -1 | 0 | 1 | 1 |
| -1 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |

Table 3

Zs is the sign bit, and Zv is the value bit of Z.

B = (~Zv & c) | (Zv & ~c)

C = (~Zv & c) | Zs

## 4. CARRY-FREE ADDITION OF RBSD NUMBERS

Once a number has been converted to RBSD format, addition is carried out using the following truth table, which can be easily transformed into a Product-of-Sum form to implement a circuit. The table used to add two RBSD numbers takes as input two consecutive digits each from the two numbers to add, and produces the digit to place in the more significant position in the sum.

| $X_i$ | $Y_i$ | $X_{i-1}$ | $Y_{i-1}$ | $S_i$ |
|---|---|---|---|---|
| $\bar{1}$ | $\bar{1}$ | 0 | 0 | 0 |
| | | 0 | 1 | 0 |
| | | 1 | 0 | 0 |
| | | 1 | 1 | 1 |
| | | | | |
| $\bar{1}$ | 0 | 0 | X | $\bar{1}$ |
| | | 1 | $\bar{1}$ | $\bar{1}$ |
| | | 1 | 0 | $\bar{1}$ |
| | | 1 | 1 | 0 |
| | | | | |
| $\bar{1}$ | 1 | 0 | $\bar{1}$ | 0 |
| | | 0 | 0 | 0 |
| | | 1 | $\bar{1}$ | 0 |
| | | 1 | 0 | 0 |
| | | | | |

| | | | | |
|---|---|---|---|---|
| 0 | $\bar{1}$ | X | 0 | $\bar{1}$ |
| | | $\bar{1}$ | 1 | $\bar{1}$ |
| | | 0 | 1 | $\bar{1}$ |
| | | 1 | 1 | 0 |
| | | | | |
| 0 | 0 | $\bar{1}$ | $\bar{1}$ | $\bar{1}$ |
| | | X | 0 | 0 |
| | | $\bar{1}$ | 1 | 0 |
| | | 0 | X | 0 |
| | | 1 | $\bar{1}$ | 0 |
| | | 1 | 1 | 1 |
| | | | | |
| 0 | 1 | $\bar{1}$ | $\bar{1}$ | 0 |
| | | X | 0 | 1 |
| | | 0 | $\bar{1}$ | 1 |
| | | 1 | $\bar{1}$ | 1 |

| | | | | |
|---|---|---|---|---|
| | | | | |
| 1 | $\bar{1}$ | $\bar{1}$ | 0 | 0 |
| | | 1 | 1 | 0 |
| | | 0 | 0 | 0 |
| | | 0 | 1 | 0 |
| | | | | |
| 1 | 0 | $\bar{1}$ | $\bar{1}$ | 0 |
| | | $\bar{1}$ | 0 | 1 |
| | | $\bar{1}$ | 1 | 1 |
| | | 0 | X | 1 |
| | | | | |
| 1 | 1 | $\bar{1}$ | $\bar{1}$ | $\bar{1}$ |
| | | $\bar{1}$ | 0 | 0 |
| | | 0 | $\bar{1}$ | 0 |
| | | 0 | 0 | 0 |

Table 4

The above table is useful for the addition of two RBSD numbers. The same table can be written in POS form to ease the implementation of an electronic circuit. The most optimized Product-of-Sum form of the above table is:

$$S0 = \bar{X}_i^s + Y_i^s + \bar{Y}_i^v$$

$$S1 = \bar{X}_i^v + Y_i^v + \bar{X}_{i-1}^s + \bar{Y}_{i-1}^s$$

$$S2 = X_i^v + \bar{Y}_i^v + \bar{X}_{i-1}^s + \bar{Y}_{i-1}^s$$

$$S3 = \bar{X}_i^s + Y_i^s + \bar{X}_{i-1}^v + Y_{i-1}^s + \bar{Y}_{i-1}^v$$

$$S4 = X_i^v + \bar{Y}_i^s + X_{i-1}^s + \bar{X}_{i-1}^v + \bar{Y}_{i-1}^v$$

$$S_i^s = S0.S1.S2.S3.S4.(\bar{X}_i^v + \bar{Y}_i^s).(X_i^s + Y_i^s + X_{i-1}^s).(X_i^s + Y_i^s + Y_{i-1}^s)$$

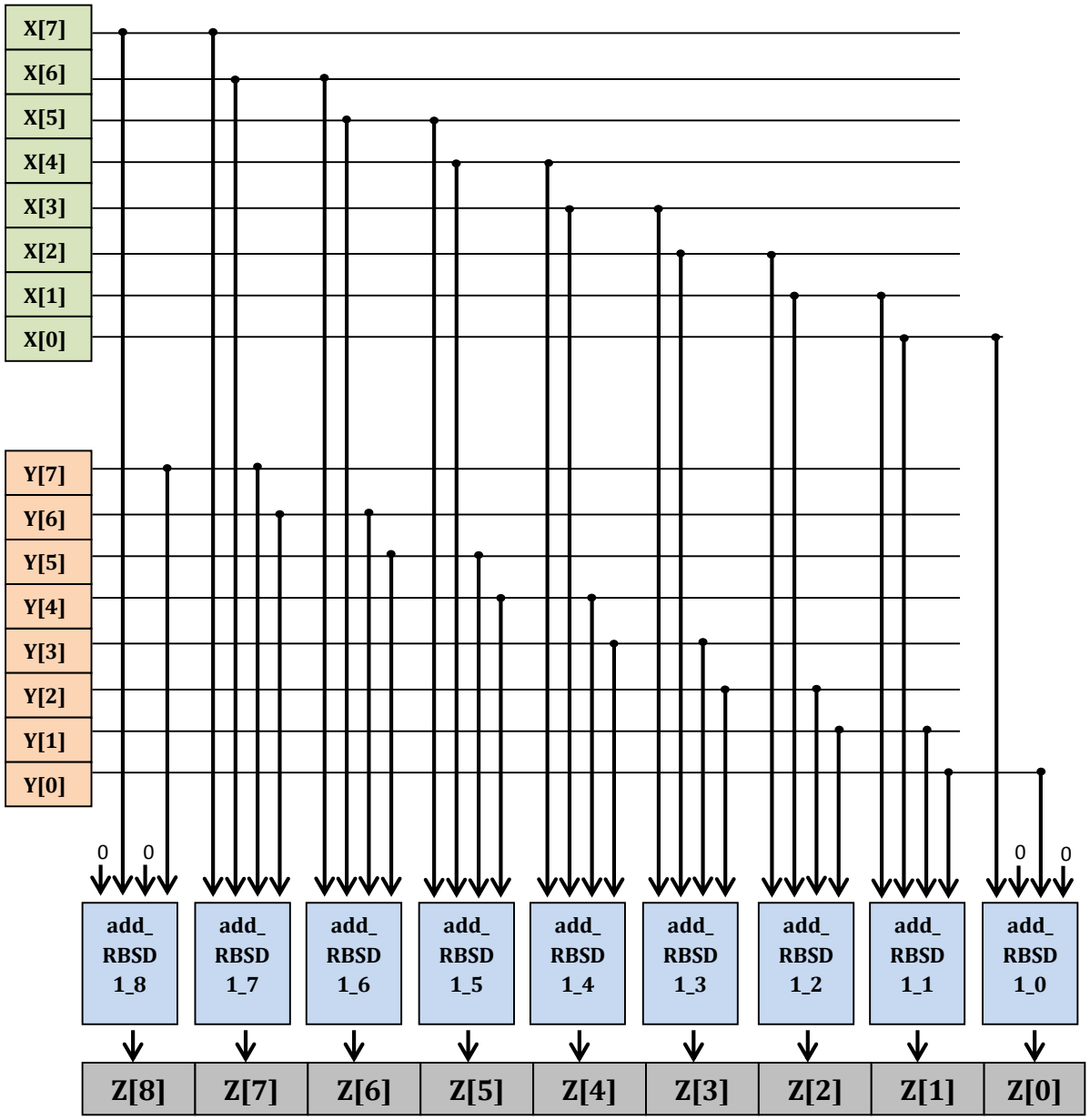
$$S_i^v = S0.S1.S2.S3.S4.(X_i^s + \bar{X}_i^v + \bar{Y}_i^s).(\bar{X}_i^v + \bar{Y}_i^v + X_{i-1}^v).(\bar{X}_i^v + \bar{Y}_i^v + Y_{i-1}^v).$$
$$(X_i^v + Y_i^v + X_{i-1}^v).(X_i^v + Y_i^v + Y_{i-1}^v).(X_i^v + Y_i^v + \bar{X}_{i-1}^s + Y_{i-1}^s).(X_i^v + Y_i^v + X_{i-1}^s + \bar{Y}_{i-1}^s)$$

The Synthesis results of performing Carry-Free Addition on two 256-bit RBSD numbers, in Xilinx 9 on a Virtex-5 FPGA, are:

**Maximum combinational delay:**
**69-bit Carry-Free RBSD Addition: 4.784ns**
**133-bit Carry-Free RBSD Addition: 4.784ns**

For Carry-Free addition of two 132-bit (each of sign and value) RBSD numbers:
Selected Device : 5vlx30ff324-3 (Virtex-5)
Slice Logic Utilization: Number of Slice LUTs: 928 out of 19200 4%
Slice Logic Distribution: Number of LUT Flip Flop pairs used: 928

As an example, consider the addition of two 8-bit RBSD number X and Y into the 9-bit sum Z:




To produce one bit of the output BSD number Z, 4 bits are required from the input, 2 from each of the two input numbers. A series of such adders that produce one bit of the output can produce the entire sum, with the delay of only one of those adders since all of them act in parallel. Thus, Carry-Free Addition is achieved.

## Carry-Free Subtraction of Two RBSD Numbers

The negative of an RBSD number can be obtained by interchanging the 1's and -1's in it. For example, RBSD [1 0 -1] is decimal 3, while RBSD [-1 0 1] is decimal -3. Thus, by carrying out Carry-Free Addition of the negative of the number to subtract and the second number, Carry-Free Subtraction is achieved.

# 5. CARRY FREE ADDITION OF BINARY NUMBERS

The addition of two binary numbers can be carried out by simply connecting a Binary-to-RBSD Converter module and a Carry-Free Adder for RBSD Numbers module in cascade. Although this is functionally correct, a better approach would be to combine the truth tables of these two modules to make a single truth table that takes as input binary digits, and produces the sum.

Thus, a truth table was constructed by merging the above two truth tables. This new table takes as input three consecutive bits in a binary number, and produces the value to be placed in the most significant position among the three input positions, in the sum.

The Synthesis results of performing Carry-Free Addition on two 256-bit Binary numbers, in Xilinx 9 on a Virtex-5 FPGA, are:

**Maximum combinational delay:**
**Carry_Free_Add_Bin32: 4.028ns**
**Carry_Free_Add_Bin132: 4.028ns**

For Carry-Free addition of two 132 bit binary numbers:
Selected Device : 5vlx30ff324-3 (Virtex-5)
Slice Logic Utilization: Number of Slice LUTs: 265 out of 19200 1%
Slice Logic Distribution: Number of LUT Flip Flop pairs used: 265

As can be observed, there is a significant change in combinational path delay in school-book addition and Carry-Free Addition; while the former is estimated to take around 71.3ns for the addition of 256-bit numbers, an optimized Carry-Free Adder of two RBSD numbers is estimated to take only 4.7ns, while an optimized Carry-Free Adder of two binary numbers is estimated to take an even lesser 4ns.

The reason for a smaller combinational delay for binary addition can be attributed to the fact that the overall number of inputs of the binary adder is lesser than that of the RBSD adder. This can be reasoned upon, keeping in mind that the RBSD number requires twice the number of digits in it in order to account for the 'sign' and 'value' of every digit.

The truth table followed to implement Carry-Free Addition of two binary numbers is shown in the next page, along with its Product-of-Sum form.

The combination of the binary->RBSD conversion table and Carry-Free Addition of Two RBSD Numbers table was made by taking inputs as two binary numbers, and finding out their sum by the two-step process of:

1) Conversion of the two binary numbers into their RBSD equivalents.
2) Adding up the two RBSD numbers thus found.

The conversion from binary to RBSD requires two consecutive binary bits to give one RBSD bit at the higher significant bit position. The addition of two RBSD numbers requires two consecutive RBSD bits from each of the two input RBSD numbers to result in one BSD bit of the Sum. Thus, a total of three consecutive binary bits from each of the two binary input numbers are required to produce a Sum bit at the most significant bit position.

Table 3 has been followed to carry out addition of two binary numbers using RBSD addition.

| $A_i$ | $A_{i-1}$ | $A_{i-2}$ | $B_i$ | $B_{i-1}$ | $B_{i-2}$ | $S_i$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | 0 | 0 | 1 | 0 |
| | | | 0 | 1 | 0 | 1 |
| | | | 0 | 1 | 1 | 1 |
| | | | 1 | 0 | 0 | $\bar{1}$ |
| | | | 1 | 0 | 1 | $\bar{1}$ |
| | | | 1 | 1 | 0 | 0 |
| | | | 1 | 1 | 1 | 0 |
| | | | | | | |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| | | | 0 | 0 | 1 | 1 |
| | | | 0 | 1 | 0 | 1 |
| | | | 0 | 1 | 1 | 1 |
| | | | 1 | 0 | 0 | $\bar{1}$ |
| | | | 1 | 0 | 1 | 0 |
| | | | 1 | 1 | 0 | 0 |
| | | | 1 | 1 | 1 | 0 |
| | | | | | | |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| | | | 0 | 0 | 1 | 1 |
| | | | 0 | 1 | 0 | $\bar{1}$ |
| | | | 0 | 1 | 1 | 0 |
| | | | 1 | 0 | 0 | 0 |
| | | | 1 | 0 | 1 | 0 |
| | | | 1 | 1 | 0 | 0 |
| | | | 1 | 1 | 1 | 1 |
| | | | | | | |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| | | | 0 | 0 | 1 | 1 |
| | | | 0 | 1 | 0 | 0 |
| | | | 0 | 1 | 1 | 0 |
| | | | 1 | 0 | 0 | 0 |
| | | | 1 | 0 | 1 | 0 |
| | | | 1 | 1 | 0 | 1 |
| | | | 1 | 1 | 1 | 1 |
| | | | | | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | $\bar{1}$ |
| | | | 0 | 0 | 1 | $\bar{1}$ |
| | | | 0 | 1 | 0 | 0 |
| | | | 0 | 1 | 1 | 0 |
| | | | 1 | 0 | 0 | 0 |
| | | | 1 | 0 | 1 | 0 |
| | | | 1 | 1 | 0 | $\bar{1}$ |
| | | | 1 | 1 | 1 | $\bar{1}$ |
| | | | | | | |
| 1 | 0 | 1 | 0 | 0 | 0 | $\bar{1}$ |
| | | | 0 | 0 | 1 | 0 |
| | | | 0 | 1 | 0 | 0 |
| | | | 0 | 1 | 1 | 0 |
| | | | 1 | 0 | 0 | 0 |
| | | | 1 | 0 | 1 | 1 |
| | | | 1 | 1 | 0 | $\bar{1}$ |
| | | | 1 | 1 | 1 | $\bar{1}$ |
| | | | | | | |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| | | | 0 | 0 | 1 | 0 |
| | | | 0 | 1 | 0 | 0 |
| | | | 0 | 1 | 1 | 1 |
| | | | 1 | 0 | 0 | $\bar{1}$ |
| | | | 1 | 0 | 1 | $\bar{1}$ |
| | | | 1 | 1 | 0 | $\bar{1}$ |
| | | | 1 | 1 | 1 | 0 |
| | | | | | | |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| | | | 0 | 0 | 1 | 0 |
| | | | 0 | 1 | 0 | 1 |
| | | | 0 | 1 | 1 | 1 |
| | | | 1 | 0 | 0 | $\bar{1}$ |
| | | | 1 | 0 | 1 | $\bar{1}$ |
| | | | 1 | 1 | 0 | 0 |
| | | | 1 | 1 | 1 | 0 |

Table 5

$$S0 = A_i + A_{i-1} + \bar{B}_i + \bar{B}_{i-1}$$

$$S1 = A_i + A_{i-1} + \bar{A}_{i-2} + \bar{B}_i + B_{i-1} + \bar{B}_{i-2}$$

$$S2 = A_i + \bar{A}_{i-1} + A_{i-2} + B_i + \bar{B}_{i-1} + \bar{B}_{i-2}$$

$$S3 = \bar{A}_i + A_{i-1} + B_i + \bar{B}_{i-1}$$

$$S4 = \bar{A}_i + A_{i-1} + \bar{A}_{i-2} + B_i + B_{i-1} + \bar{B}_{i-2}$$

$$S5 = \bar{A}_i + \bar{A}_{i-1} + A_{i-2} + \bar{B}_i + \bar{B}_{i-1} + \bar{B}_{i-2}$$

$$S6 = \bar{A}_i + \bar{A}_{i-1} + \bar{A}_{i-2} + \bar{B}_i + \bar{B}_{i-1}$$

$$S_i^s = S0.S1.S2.S3.S4.S5.S6.(A_i + A_{i-1} + B_i).(A_i + \bar{A}_{i-1} + A_{i-2} + B_i + B_{i-1})$$
$$.(A_i + \bar{A}_{i-1} + A_{i-2} + \bar{B}_i).(A_i + \bar{A}_{i-1} + \bar{A}_{i-2}).(\bar{A}_i + A_{i-1} + \bar{B}_i + B_{i-1}).(\bar{A}_i + \bar{A}_{i-1} + B_i)$$

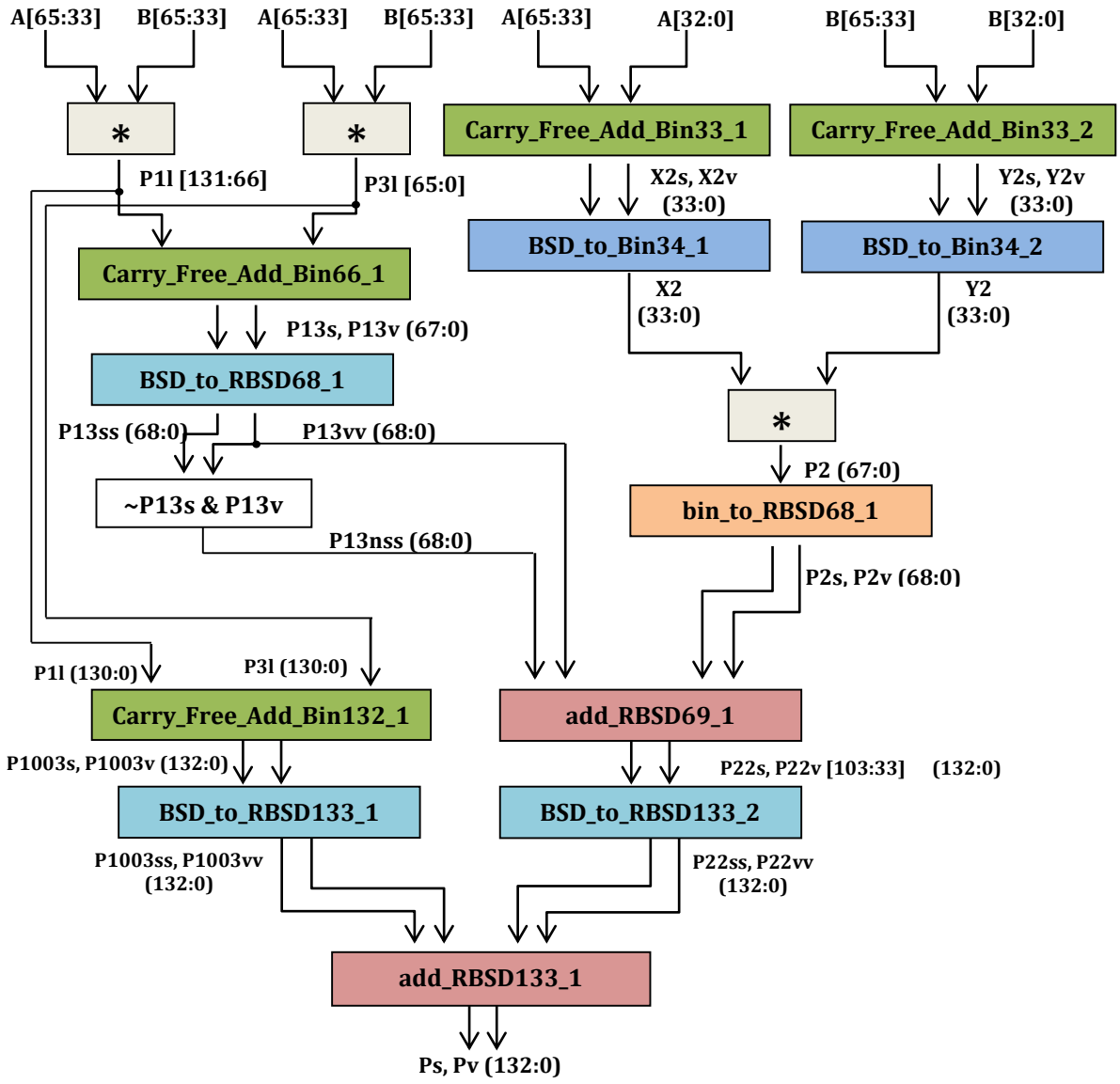
$$S_i^v = S0.S1.S2.S3.S4.S5.S6.(A_i + A_{i-1} + A_{i-2} + B_i + B_{i-1}).(A_i + A_{i-1} + \bar{A}_{i-2} + B_i + B_{i-1})$$
$$.(A_i + \bar{A}_{i-1} + \bar{B}_i + B_{i-1}).(A_i + A_{i-1} + A_{i-2} + \bar{B}_i + \bar{B}_{i-1} + B_{i-2}).(A_i + \bar{A}_{i-1} + \bar{A}_{i-2} + B_i + \bar{B}_{i-1})$$
$$.(\bar{A}_i + A_{i-1} + A_{i-2} + \bar{B}_i + B_{i-1}).(\bar{A}_i + A_{i-1} + \bar{A}_{i-2} + \bar{B}_i + B_{i-1} + B_{i-2}).(\bar{A}_i + \bar{A}_{i-1} + B_i + B_{i-1})$$
$$.(\bar{A}_i + \bar{A}_{i-1} + A_{i-2} + B_i + \bar{B}_{i-1} + B_{i-2})$$

## 6. MULTIPLICATION USING CARRY-FREE ADDITION

Integer multiplication forms another very important part of integer arithmetic used in ECC. The basic (school-book) multiplication algorithm is time-intensive ($O(n^2)$), and so is not a preferable option.

Karatsuba Algorithm for multiplication proves to be a fast algorithm to be used for ECC applications, or any application for that matter ($O(n^{Log_2 3})$). Hence, this algorithm has been chosen to implement integer multiplication. The above methods for Carry-Free Addition have been incorporated into this multiplication algorithm.

The following diagram is the circuit implementation of the Karatsuba Algorithm, for the multiplication of two 66-bit binary numbers A and B:




Synthesis results of the above circuit in Xilinx 9 on a Virtex-5 FGPA are:

**Maximum combinational delay: ks_mul_Bin66: 24.9728ns**
For multiplication of two 66 bit binary numbers:
Selected Device : 5vlx30ff324-3 (Virtex-5)
Slice Logic Utilization: Number of Slice LUTs: 2223 out of 19200 11%
Slice Logic Distribution: Number of LUT Flip Flop pairs used: 2223

## 7. MODULAR REDUCTION USING CARRY-FREE ADDITION

The algorithm used for Modular Reduction has been found in the book "Guide to Elliptic Curve Cryptography" by Darrel Hankerson et al[2] to carry out reduction modulo the 256-bit prime p256:

**Fast reduction modulo $p256 = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$**

INPUT: An integer $c = (c15, \ldots, c2, c1, c0)$ in base $2^{32}$ with $0 \leq c < p256^2$.

OUTPUT: $c \bmod p256$.

1. Define 256-bit integers:

$s1 = (c7, c6, c5, c4, c3, c2, c1, c0)$,

$s2 = (c15, c14, c13, c12, c11, 0, 0, 0)$,

$s3 = (0, c15, c14, c13, c12, 0, 0, 0)$,

$s4 = (c15, c14, 0, 0, 0, c10, c9, c8)$,

$s5 = (c8, c13, c15, c14, c13, c11, c10, c9)$,

$s6 = (c10, c8, 0, 0, 0, c13, c12, c11)$,

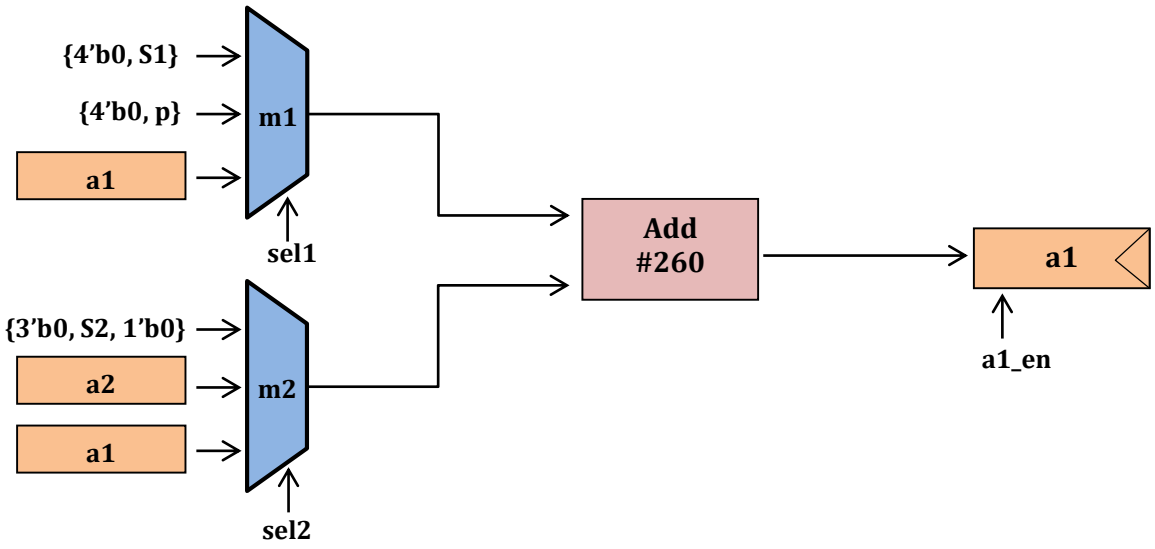
$s7 = (c11, c9, 0, 0, c15, c14, c13, c12)$,

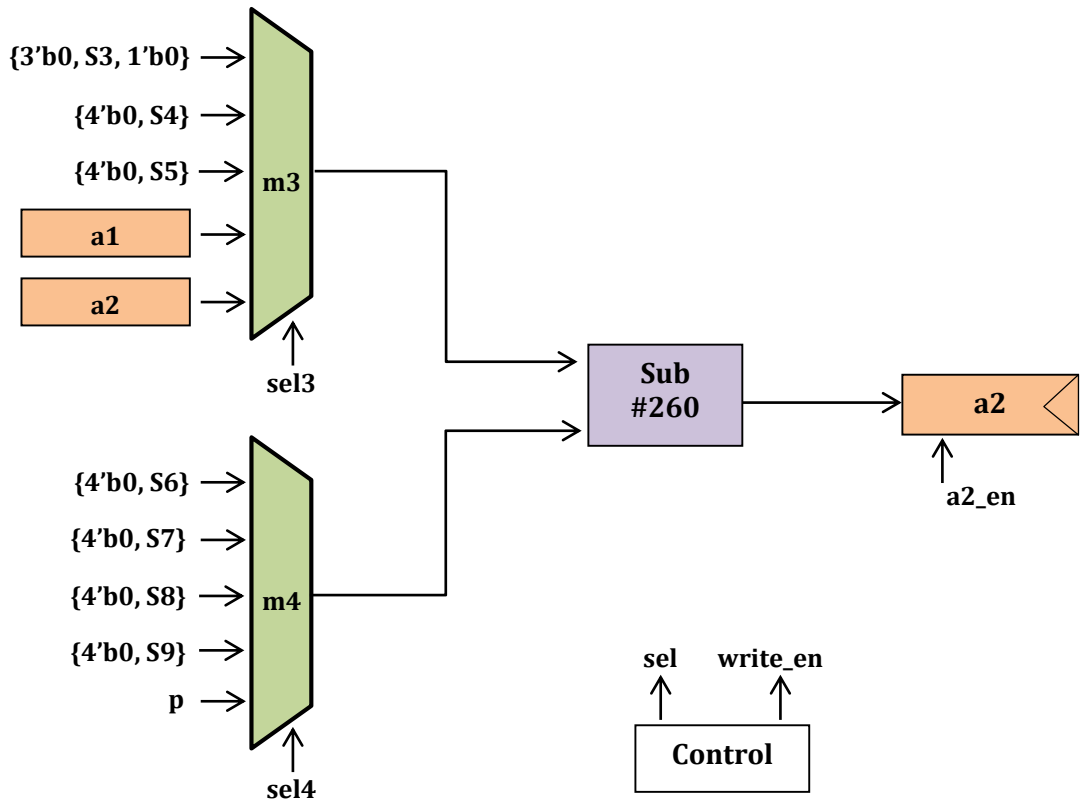
$s8 = (c12, 0, c10, c9, c8, c15, c14, c13)$,

$s9 = (c13, 0, c11, c10, c9, 0, c15, c14)$.

2. Return $(s1 + 2s2 + 2s3 + s4 + s5 - s6 - s7 - s8 - s9 \bmod p256)$.

A sequential circuit was designed to carry out this modular reduction:






This sequential circuit takes 10 clock cycles to complete the operation. The adder and subtracter circuits have been designed to accommodate a 260-bits input number, since the addition and subtraction operations could increase the resultant bit size from 256.

The Synthesis results are:

**Minimum period: 5.215ns (Maximum Frequency: 191.755MHz)**
**Minimum input arrival time before clock: 5.880ns**
**Maximum output required time after clock: 7.229ns**
**Maximum combinational path delay: 7.895ns**

Selected Device: 5vlx30ff324-3 (Virtex-5)
Slice Logic Utilization: Number of Slice Registers: 1051 out of 19200 5%
Slice Logic Distribution: Number of Slice LUTs: 10612 out of 19200 55%

The above method of implementation of Modular Reduction is sequential in nature, and is, therefore, expected to carry unnecessary delay. A combinational approach can be pursued in implementing this circuit without any delay, by simplifying the operation equation to a Sum-of-Products or a Product-of-Sums form, similar to the circuits of the other arithmetic operations shown previously. Such an implementation shall not only save time, but shall also cut down on storage space.

# 8. PROBLEMS

## 8.1 Binary-to-RBSD Conversion

It is assumed that the binary number being considered is positive. In this case, there is an increase in the bit size from n-bits of the binary number to 2*(n+1) bits of the RBSD number.

However, if the binary number considered is in its 2's complement form, then there would be no need for the extra digit in Binary to RBSD conversion, and the number of bits would increase to just 2*n. This remains to be implemented in all the functions that have been written.

## 8.2 Use of 2's Complement binary numbers

Throughout the project, binary numbers have been assumed to be positive. However, the incorporation of negative numbers has not been explored, i.e. the possible use of binary numbers as being in their 1's complement or 2's complement forms.

The good news here is that this incorporation is particularly easy. If the binary number being used is assumed to be in its 2's complement form, then there shall be no increase in number of bits in its conversion from binary to RBSD. If the number is negative, the most significant bit of its RBSD form shall inevitably turn out to be -1. Thus, Binary to RBSD conversion shall then entail only the doubling of number of bits to take care of the sign bits, and no extra bit addition.

## 8.3 Carry Free Addition

The field of RBSD numbers is not closed. Hence, the addition of two RBSD numbers, carry-free or with carry propagation, shall result in a BSD number that is not necessarily RBSD.

Fortunately, the conversion of a BSD number to RBSD is combinational, and hence shall not contribute much to the delay of the circuit. Thus, every time Carry-Free addition has taken place, a module for the corresponding bit size has been attached to its output that functions as a BSD-to-RBSD converter.

It can be argued that just as in the case of Carry-Free Addition of Binary Numbers, where instead of performing Binary-to-RBSD Conversion followed by Carry-Free Addition of RBSD Numbers a new table was created that directly provides the sum given the binary input, a combined table for Carry-Free Addition (Binary or RBSD) and BSD-to-RBSD converter could prove to make a faster circuit. However, making a concise table has many hurdles.

It can be observed that the BSD-to-RBSD Converter takes 4 consecutive digits of the input BSD number and produces one output digit at the most significant digit position. It can also be observed that a Carry-Free Adder takes 2 consecutive RBSD input digits, or 3 consecutive binary input bits, to produce one output digit at the most significant position. Since these act as inputs to the BSD-to-RBSD Converter, 5 consecutive input RBSD digits in case of RBSD Carry-Free Adder, and 6 consecutive bits in case of Binary Carry-Free Adder are required to produce one output digit at the most significant position. This shall result in a table having 3^5 = 243 rows in case of RBSD Adder, and 2^6 = 64 rows in case of Binary adder. It is practically infeasible to manually construct a Sum-of-Products or Product-of-Sum form for the resulting truth table.

## 8.4 BSD-to-Binary Conversion

BSD-to-Binary conversion involves carry propagation. Although, since the conversion from Binary to RBSD in done so as to eliminate carry propagation, it can be expected that the opposite direction would involve carry propagation. Unfortunately, it cannot be avoided. In the future, if a method is devised to eliminate the need for carry propagation in this conversion, then instead of using it here that method should directly be applied for the addition of two numbers.

## 8.5 Karatsuba Multiplication with Carry-Free Addition Incorporated

### 8.3.1 Splitting results in Carry Propagation

The first step of Karatsuba multiplication requires splitting the input number into two parts. In case of BSD numbers, however, simply splitting the number at the middle is mathematically incorrect, and shall lead to erroneous results. Moreover, mathematically sound splitting involves carry propagation, which dissolves our goal of fast computation.

### 8.3.2 Use of 34-bit RBSD-to-Binary converter which involves carry propagation

Two 34-bit RBSD-to-Binary converters are used in the multiplier, which are the major contributors to the path delay of this circuit since they involve carry propagation. It can be argued that the use of 17-bit RBSD-to-Binary converters can help decrease the path delay. However, this would involve splitting the RBSD number, which, as discussed above, results in some carry propagation of its own.

### 8.3.3 Integer Multiplication

Ultimately, multiplication operation is performed using integer arithmetic, which involves carry propagation. It can be argued that the numbers can be split into two 17-bit numbers and integer multiplication can be carried out over them instead of the 34-bit numbers. However, this is equivalent to adding one more level to the Karatsuba multiplication being carried out. This shall result in thrice the number of multiplications involved. It needs to be checked through experimentation whether this incurs lesser delay.

# 9. FUTURE WORK

- Implement 2's complement binary number to RBSD conversion
- Implement carry-free integer multiplication
- Check the performance of 17-bit integer multipliers in the Karatsuba Carry-Free 66-bit Binary Multiplier instead of the 34-bit ones currently used
- Incorporate splitting and concatenation of BSD and RBSD numbers in carry-free arithmetic
- Implement Modular Reduction in a combinational circuit
- Compare performance of arithmetic circuits with state of the art arithmetic modules
- Design an ECC Processor that works with Carry-Free arithmetic:
    - design of arithmetic modules that work completely in the BSD number domain, without conversions to and from binary

# 10. REFERENCES


1) Behrooz Parhami, "Carry-Free Addition of Recoded Binary Signed-Digit Numbers", IEEE TRANSACTIONS ON COMPUTERS, Vol. 37, No. 11, November 1988
2) Behrooz Parhami, "Generalized Signed-Digit Number Systems: A Unifying Framework for Redundant Number Representations", IEEE TRANSACTIONS ON COMPUTERS, Vol. 39, No. 1, January 1990
3) Darrel Hankerson, Alfred Menezes, Scott Vanstone, "Guide to Elliptic Curve Cryptography"