TORNADO: A Power-Aware Scalable Radix-4 Montgomery Multiplier

Hee-Kwan Son
Multimedia Lab, SoC R&D Center
Samsung Electronics Co., Korea
+82-31-209-9781
martin.son@samsung.com

Sang-Geun Oh
Multimedia Lab, SoC R&D Center
Samsung Electronics Co., Korea
+81-31-209-7040
stephen.oh@samsung.com

Abstract

Modular exponentiation is a critical but costly operation for a variety of public-key cryptosystems. An effective and probably the most powerful algorithm for modular exponentiation is Montgomery multiplication method. In this paper, we introduce a new Montgomery multiplier named TORNADO that is very efficient in view of operation performance, power consumption, and scalability (i.e., flexibility).

1. Introduction

Public-key cryptography algorithms such as RSA encryption scheme [1] and the Diffie-Hellman key agreement scheme [2] are very powerful methods to construct secure communication channel between two end-point participants. The throughput of these schemes, however, is very low since they are based on the long integer modular exponentiation. Nowadays Montgomery's algorithm [3] is considered as one of the most efficient methods to implement the modular exponentiation. The algorithm uses simple division by a power of two instead of division by a modulus.

The security parameters used in cryptosystems do not stay in fixed bit-length forever. As computers' processing power increases, to provide the same "effective" security level, cryptosystems require security parameters to be increased as well. To be able to cope with such coming requests, the architecture of cryptographic hardware has to be scalable.

It is well known that applying the modified Booth recoding scheme to the radix-4 multiplication is very efficient since it can almost halve the number of clock cycles, costing small amount of extra hardware. The Booth recoding scheme makes the calculation of a partial product simple. However, contrary to normal multiplication, Montgomery multiplication needs one more vector besides the partial product. Since the Booth recoding scheme cannot simplify the calculation of the vector, an additional scheme has to be devised.

Numbers used in public-key cryptosystems usually have very large bit-lengths like 1024-bit or 2048-bit. Thus, if we add such big numbers, allowing the carry to propagate through the whole bits, clock's cycle time might be very large. To overcome this problem, carry-save addition (CSA) architecture can be used. Carry propagation path of it is very short and is independent on the bit-length of input numbers.

For any hardware device used in the mobile applications such as a Smart Card, power consumption is as much as important as processing performance. Reducing spurious transitions and Expected Switching Activity (ESA) appeared at high fan-out signals are ultimately effective to decrease power consumption. And to our knowledge, serial multipliers show better power-delay product characteristic than parallel multipliers.

This paper presents a systematic design methodology for low power, radix-4, and scalable Montgomery multiplier. The organization of this paper is as follows. Section 2 explains the digit-serial radix-4 Montgomery multiplication algorithm that uses only full-precision numbers. In this section, we also introduce a specialized recoding scheme as well as the well-known Booth recoding scheme. Section 3 explains a method that allows our hardware to be scalable. Section 4 presents the structure of accumulator in detail. Section 5 describes adopted low-power techniques. The block diagram of data path is also shown in this section. Finally section 6 concludes the paper.

2. Radix-4 Montgomery Multiplication

The basic full-precision algorithm for a radix-4 digit-serial interleaved Montgomery multiplication [4], [5] is given below:

[Inputs]

```
  $k$ : digit-length of $M$
  $M = (m_{k-1},…,m_1,m_0)_4$ : $2 < M < 4^k$,
                          $M$ is odd
  $A = (a_{k-1},…,a_1,a_0)_4$ : $0 \le A < M$
  $B = (b_{k-1},…,b_1,b_0)_4 : 0 \le B < M$
[Output]
  $S = (s_{k-1},…,s_1,s_0)_4 : 0 \le S < M$
[Method]
  $S \leftarrow 0$
  for i=0 to (k-1)
    $q_i \leftarrow ((s_0 + b_i a_0) m_0') \bmod 4$
    $S \leftarrow (S + b_i A + q_i M) / 4$
  endfor
  if $(S \ge M)$ $S \leftarrow (S - M)$ endif
```

where $R = 4^k$ and $m_0' = -m_0^{-1} \bmod 4$. Inputs A, B, and M denote multiplicand, multiplier, and modulus of the modular multiplication, respectively. The calculated result S is expressed mathematically as $S \equiv (AB + QM)R^{-1} \equiv ABR^{-1}$ (mod M). In this algorithm, q_i is the i-th significant digit of the so-called Montgomery quotient $Q = (q_{k-1}, …,q_1,q_0)_4$. Its value is calculated in such a way as to make the least significant digit of $(S + b_iA + q_iM)_4$ zero at each iteration. From now on, we use PP and MM to represent a partial product (b_iA) and a multiple of modulus (q_iM), respectively.

In the case of radix-4, b_i and q_i are 2-bit numbers. Thus, the value sets of PP and MM are as follows:

$$PP \in \{0, A, 2A, 3A\},\ MM \in \{0, M, 2M, 3M\}.$$

To calculate $3A$ and $3M$ on the fly, we need two extra adders. By using a modified Booth recoding scheme, the extra adder to calculate $3A$ can be replaced with inverters. Let the $b_{i,1}$ and $b_{i,0}$ are the two bits in the i-th significant digit of B. The radix-4 modified Booth recoding scheme takes a bit stream $(b_{i,1},b_{i,0},b_{i-1,1})_2$ as its input and generates a recoded PP according to Table 1, where $b_{-1,1}$ is defined to be 0 and qa_i is the recoded quotient digit for a PP at the i-th iteration.

TABLE 1
Booth Recoding Scheme

Triple input bits			Recoded quotient for PP	Recoded PP
$b_{i,1}$	$b_{i,0}$	$b_{i-1,1}$	qa_i	PP
0	0	0	0	0
0	0	1	+1	+A
0	1	0	+1	$+A$
0	1	1	+2	$+2A$
1	0	0	-2	$-2A$
1	0	1	-1	$-A$
1	1	0	-1	$-A$
1	1	1	0	0

Booth recoding scheme transforms the value set of PP into $\{-2A, -A, 0, +A, +2A\}$. All elements in the transformed value set are calculated by simple operations such as bit-inversion and/or bit-shift. However, $3M$ still remains in the value set of MM, and the Booth recoding scheme alone cannot solve this problem. In this paper, we adopt a specialized method named 'Montgomery recoding scheme' to transform the original value set of MM into the one comprising easily obtainable elements only. Let $(sp_{0,1},sp_{0,0})_2$ be the 2 bits in the least significant digit (LSD) of $SP = S + PP$ and $(m_{0,1},m_{0,0})_2$ be the 2 bits in the LSD of M. According to the input condition that M has to be odd, $m_{0,0}$ has to be always '1'. Then, Montgomery recoding scheme takes a bit stream $(sp_{0,1},sp_{,0,0},m_{0,1})_2$ as its input and generates a recoded MM according to Table 2, where qm_i is the recoded quotient digit for a MM at the i-th iteration.

TABLE 2
Montgomery Recoding Scheme

Three input bits			Recoded quotient for MM	Recoded MM
$sp_{0,1}$	$sp_{0,0}$	$m_{0,1}$	qm_i	MM
0	0	0	0	0
0	0	1	0	0
0	1	0	-1	$-M$
0	1	1	+1	$+M$
1	0	0	+2	$+2M$
1	0	1	+2	$+2M$
1	1	0	+1	$+M$
1	1	1	-1	$-M$

Montgomery recoding scheme transforms the value set of MM into $\{-M, 0, +M, +2M\}$. Due to the two recoding schemes, it becomes easy to calculate all the elements in the value sets of PP and MM.

3. Scalability by Multiple Precision

The basic algorithm in Section 2 is based on the full-precision arithmetic, thus it is not scalable. In addition, because our Montgomery multiplier exchanges input/output data with a host such as CPU or DSP via a shared data memory, we have to consider data bus width of the memory. To make our algorithm scalable through multiple-precision operation and compatible with the data memory, we implement it by constructing a 4-fold nested loop structure. Every operand in multiple-precision arithmetic has to be equally divided into p chunks, where p is an integer specifying the precision (1 for single precision, 2 for double precision, etc.). To help our explanation, several unit bit-lengths are introduced:

n : bit-length of modulus M
c : bit-length of a chunk
w : bit-length of a word

where $n = pc$, c is a multiple of w, and w is the data bus width of the memory. Since input/output operands excluding M have sign bits as explained in Section 2, the bit-lengths of them are extended to $n' \geq n+1$ and $n' = pc' = p(c+x)$, where c' ($= c+x$) is the bit-length of an extended chunk and x is a number decided by the memory's data bus configuration. In the n' bits, $n'-n = p(c'-c)$ bits at the most significant side are sign bits. If the memory has 32-bit data bus and its transferable data units are 8-bit, 16-bit, and 32-bit, then $w = 32$ and $x = 8$, 16, or 32. Choosing smaller x leads to better performance but greater complexity in hardware. Following the trade-off rule, we choose $x = w/2$ in this paper.

To prevent the performance degradation resulting from carry propagation through the entire bits, the accumulator in our Montgomery multiplier has been designed using Carry-Save Adder (CSA) architecture. The accumulator sums up vectors produced at each iteration. We name this "*accumulation-by-CSA*" process. Because of the CSA architecture, the accumulator's output is not in a conventional representation (CR) but in a redundant representation (RR) where a number is expressed as a sum of two numbers. Any type of Carry-Propagate Adder (CPA) can be used to convert a RR number into CR number by summation. We name this "*conversion-by-CPA*" process.

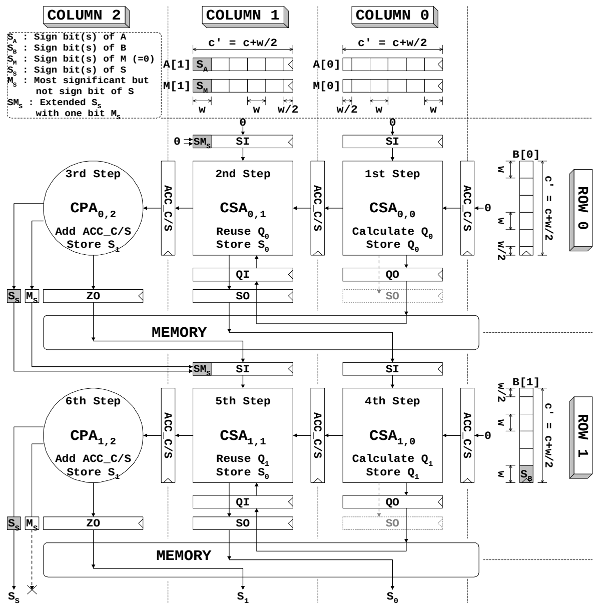

Fig 1. Processing Matrix

The operation flow diagram is drawn at Fig 1. To simplify explanation, although p can be any positive integer, we use $p = 2$ (i.e., double-precision case). We name the diagram "*processing matrix*" since it looks like a matrix with p rows and $p+1$ columns. The i-th row uses the i-th extended chunk of B as well as full data of A and M. The j-th column other than the last one in a row uses the j-th extended chunk of A and M. The last row outputs the final result. In each row, *conversion-by-CPA* process is executed at the last column and *accumulation-by-CSA* process is executed at the other columns. At the first column in a row, an extended chunk of the row's Montgomery quotient Q is calculated and stored into a memory. At each column except for the first and the last ones in a row, an extended chunk of the row's result S is calculated and stored into the memory. The extended chunk of Q stored at the first column of the corresponding row is read and reused in calculating each extended chunk of S. Every non-first row reloads the previous row's result SI and uses it at the *accumulation-by-CSA* processes.

When we generate PP and MM, we use only simple operations such as bit-shift and bit-inversion. Thus, if PP is $-A$ or $-2A$ and/or MM is $-M$, to represent the numbers correctly, "increment by 1" operation should be performed by the accumulator. In addition, one digit of SI is serially extracted and then is also summed up by the accumulator. Let's use NEG_PP and NEG_MM to indicate that PP and MM are produced by inversion, respectively. And SI_DGT is used to represent the serially extracted digit of SI. By summing up NEG_PP, NEG_MM, and SI_DGT, we generate a new 4-bit vector TT. Fig. 2 shows the circuit diagram of TT generator. The most significant bit (MSB) of SI_DGT used at the last *accumulate-by-CSA* process of a row represents the sign bit of SI. Thus, determining the sign bit (i.e., MSB) of TT depends on whether present SI_DGT has the sign bit of SI or not. Along with PP and MM, TT is also involved in the *accumulation-by-CSA* process as a third input vector.

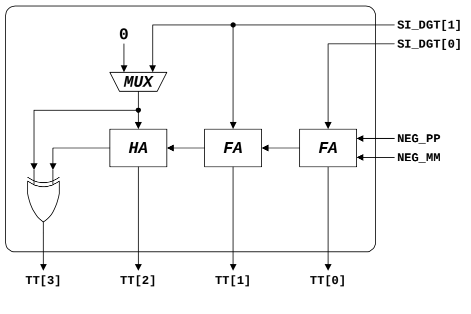

Fig 2. Circuit Diagram of TT Generator. (HA, FA, and MUX denote half-adder, full-adder, and multiplexer, respectively.)

The actual data processing operation represents a 4-fold nested loop and can be explained by a pseudo code as follows:

```
/* iteration for each ext. chunk of $B$ */
**for** $row\_idx$ = 0 **to** ($p$-1)
    $clr\_acc();$
    /* iteration for each ext. chunk of $A$ and $M$ */
    **for** $col\_idx$ = 0 **to** ($p$-1)
        /* iteration for each word of ext. chunk of $B$ */
        **for** $wrd\_idx$ = 0 **to** ($c$/$w$)
            /* iteration for each digit of  a word of $B$ */
            **for** $dgt\_idx$ = 0 **to** ($w$/2-1)
                $booth\_rec();$
                $pp\_gen();$
                $sp\_gen();$
                $montg\_rec();$
                $mm\_gen();$
                $tt\_gen();$
                $csa\_accum();$
            **endfor**
        **endfor**
    **endfor**
    /* iteration for each word of $S$ */
    **for** $wrd\_idx$ = 0 **to** ($c$/$w$-1)
        $cpa\_conv()$;
    **endfor**
**endfor**
```

where *row_idx* and *col_idx* are the row index and the column index in the *processing matrix*, respectively; *wrd_idx* and *dgt_idx* are the index of a word in an extended chunk and the index of a digit in a word, respectively. In the pseudo code above, we use several data processing functions: *clr_acc()* clears the accumulator's registers, *booth_rec()* executes the Booth recoding, *montg_rec()* executes the Montgomery recoding, *sp_gen()* generates *SP* that has 2 bits, *pp_gen(), mm_gen(), tt_gen()* generate *PP* vector, *MM* vector, *TT* vector, respectively, *csa_accum()* adds up the three vectors through *accumulation-by-CSA* process, *cpa_conv()* converts an RR number stored in the accumulator's registers into a CR number through *conversion-by-CPA* process. Because data bus of the memory has *w*-bit width, we employ a *w*-bit pipelined CPA for this operation.

4. Structure of Accumulator

The accumulator consists of 5-2 compressors as well as two registers, ACC_C and ACC_S, to store its value. ACC_C and ACC_S have *c*'+2 bits and *c*'+1 bits, respectively. One 5-2 compressor is constructed by stacking up 3 full-adders (FAs) as shown in Fig. 3 below.

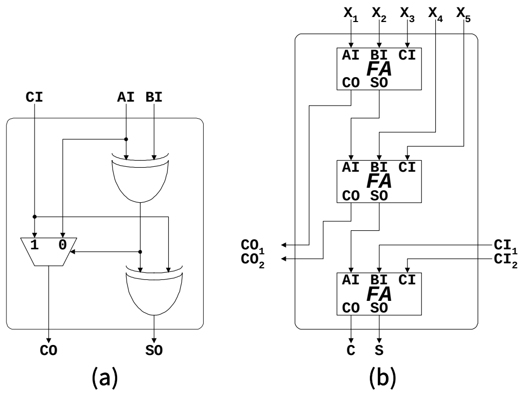

Fig 3. Circuit Diagram of (a) Full-Adder and (b) 5-2 Compressor

The structure of the accumulator is illustrated in Fig. 4. Two compressors at the highest bit positions use the sign bits (i.e., MSBs) of *PP* and *MM*. Because *TT* has only 4 bits, the MSB of *TT* is extended through the upper bit positions of accumulator. The sign bits of ACC_C and ACC_S are suitably extended when they are fed back. One digit of a row's result, SO_DGT, is generated and outputted from the two lowest compressors at each cycle of the non-first and non-last columns. The values in ACC_C and ACC_S registers are signed numbers.

5. Low Power Techniques

From the basis of fundamental idea explained at Section 2 to 4, we further improve our hardware to dissipate less power than the one implemented directly. Inevitably all digital devices have spurious transitions or glitches internally due to unbalanced path delays, and they cause worthless dynamic power dissipation. Furthermore, if fan-outs of the glitchy signals are big, then the amount of worthlessly dissipated power is significant. Such signals identified by us are the outputs from the two circuit modules in charge of the Booth and Montgomery recodings. Because the modules comprise only combinational logic circuits according to Table 1 and Table 2, their outputs must have glitches. And the fan-outs of the outputs are *c*'+2 that is usually very large number. To reduce the glitching power dissipation, we put in some latches and force the outputs to pass through latches. If all flip-flops and registers capture their inputs at the clock's rising edge, then the latches are transparent when the clock is in a low state. If the outputs of the two recoding modules can reach their stable values before the clock's falling edge, none of the glitches can propagate to the fan-out modules driven by the outputs. We name these latches "*glitch blockers*". The *glitch blockers* are also very effective for reducing the glitches appearing in the accumulator since they synchronize the arrival of *PP* and *MM* at the accumulator's inputs.

Fig. 5 shows the block diagram of data path. PP generator makes *PP* by modifying an extended chunk of *A* according to the Booth recoder's outputs, SEL_PP and EN_PP. MM generator makes *MM* by modifying an extended chunk of *M* according to the Montgomery recoder's outputs, SEL_MM and EN_MM. Several *glitch blockers* are located at the outputs of the two recoders. Meanings of the two recoders' outputs are as follows:

Booth Recoder's outputs

SEL_PP:	selects *PP*'s value from {–2*A*,–*A*,+*A*,+2*A*}
EN_PP:	sets *PP*'s value to 0 or not
NEG_PP:	indicates that *PP*'s value is –2*A* or –*A*

Montgomery Recoder's outputs

SEL_MM:	selects *MM*'s value from {–*M*,+*M*,+2*M*}
EN_MM:	sets *MM*'s value to 0 or not
NEG_MM:	indicates that *MM*'s value is –*M*

When *PP* is zeroed by EN_PP, *PP* outputted from the PP generator does not depend on SEL_PP. Thus, keeping SEL_PP frozen at that time is effective for reducing power dissipation. Same reasoning also applies to SEL_MM. We place two 2-bit flip-flops and construct feedback loops for SEL_PP and SEL_MM to implement this idea.

Booth recoding scheme has a feature that +2*A* cannot follow +*A* or +2*A*, and –2*A* cannot follow –*A* or –2*A*. Utilizing this feature to lower power consumption, we suggest a binary coding method for SEL_PP as follows:

SEL_PP for +*A* = ~(SEL_PP for +2*A*)
SEL_PP for –*A* = ~(SEL_PP for –2*A*)

where '~' denotes bit-inversion. This binary coding method minimizes the ESA of SEL_PP.

The whole *c'* bits of an extended chunks of *A* and *M* are used at every cycle in the non-last columns. And neighboring columns require different *c'* bits of *A* and *M*. To maximize the processing performance, we implement the *double buffering* scheme by preparing a pair of *c'*-bit registers for both *A* and *M* (AX_REG, AY_REG, MX_REG, and MY_REG in Fig. 5). While *c'* bits of *A* (and *M*) in one register is used for a column operation, the next *c'* bits of *A* (and *M*) is pre-loaded into the other register for the next column operation. Since all bits in an extended chunk of A and M are simultaneously used, registers for *A* and *M* are parallel-in-parallel-out (PIPO) registers. To the contrary, one digit of *B*, *Q*, and *S* is needed and/or generated at each cycle. Thus we prepare three *w*-bit parallel-in-serial-out (PISO) shift registers for inputs *B*, *Q*, and *S* (BI_REG, QI_REG, and SI_REG in Fig. 5), and two *w*-bit serial-in-parallel-out (SIPO) shift registers for outputs *Q* and *S* (QO_REG and SO_REG in Fig. 5). Those five shift registers perform 2-bit right shift at a clock cycle.

At the beginning of the last column in a row, higher significant bits of the row's result remain in the accumulator's registers without being stored into data memory. We implement a module that contains a pipelined *w*-bit CPA. It undertakes three kinds of tasks: summing up the bits remaining in the accumulator's registers by the iterated *conversion-by-CPA* processes, storing lower significant *cp-c'* (*p*-1) bits of the sum (i.e., *ZO*) into the memory in *w* bits a cycle, and registering next higher significant two bits of the sum (i.e., SIGN_S and MS1B_S) at flip-flops.

At each cycle of the first column in a row, Montgomery recoder calculates a digit of *Q* according to Table 2. This calculation requires a 2-bit number, *SP*. Its value is determined by summing up four kinds of digits as shown in the following pseudo code below.

```
SP[1:0] = (PP[1:0] + SI_DGT[1:0]) +
          (ACC_C[1:0] + ACC_S[1:0]);
```

where PP[1:0] is the LSD of *PP*, ACC_C[1:0] and ACC_S[1:0] are the LSDs of ACC_C and ACC_S, respectively, SI_DGT[1:0] is the serially extracted digit of the previous row's result *SI*.

6. Conclusion

We propose an efficient VLSI architecture and implementation methodology for a Montgomery multiplier. Scalability is achieved through the multiple-precision operation. High performance is achieved through a radix-4 multiplier architecture based on CSA accumulation. Applying the Montgomery recoding scheme as well as the Booth recoding scheme reduces the amount of hardware resources and the length of critical path. Power optimization is achieved through decreasing the glitches and the ESA of high fan-out signals. Due to its low-power and high-performance features, the presented Montgomery multiplier can be applicable to mobile devices such as Smart Card IC and flash card IC successfully.

7. References

[1] R.L. Rivest, A. Sharmir, and L. Adleman, “A Method of Obtaining Digital Signature and Public-Key Cryptosystems,” *Comm. ACM.*, vol. 21, no. 2, pp. 120-126, 1982

[2] W. Diffie and M.E. Hellman, "New Directions in Cryptography," *IEEE Trans. Information Theory*, vol. 22, pp. 644-654, 1976

[3] P.L. Montgomery, "Modular Multiplication without Trial Division," *Math. Computing*, vol. 44, no. 170, pp. 519-521, Apr. 1985

[4] J. Menezes, P.C. van Oorschot, and S.A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997

[5] H. Orup, “Simplifying Quotient Determination in High - Radix Modular Multiplication,” *Proc. 12th Symp. Computer Arithmetic*, pp. 193-199, July 1995

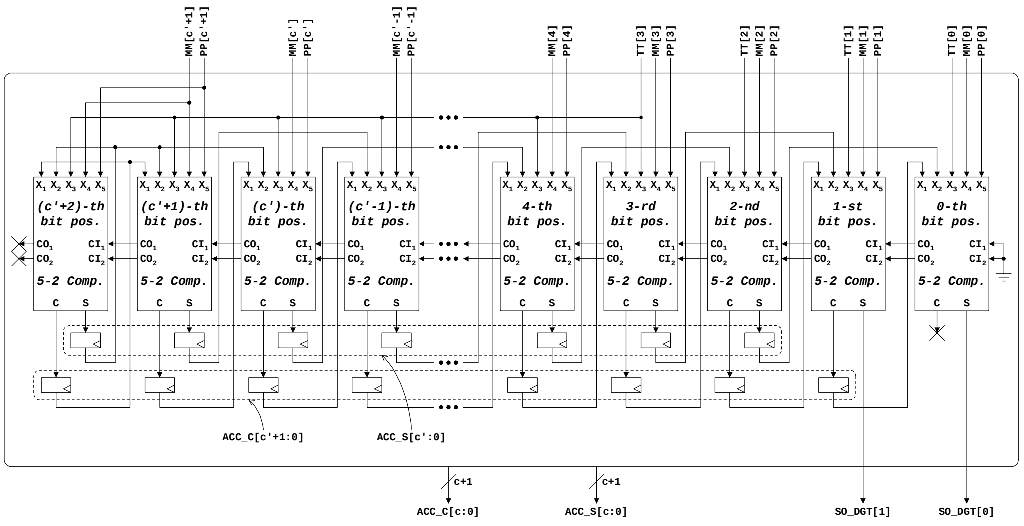

Fig 4. Structure of Accumulator

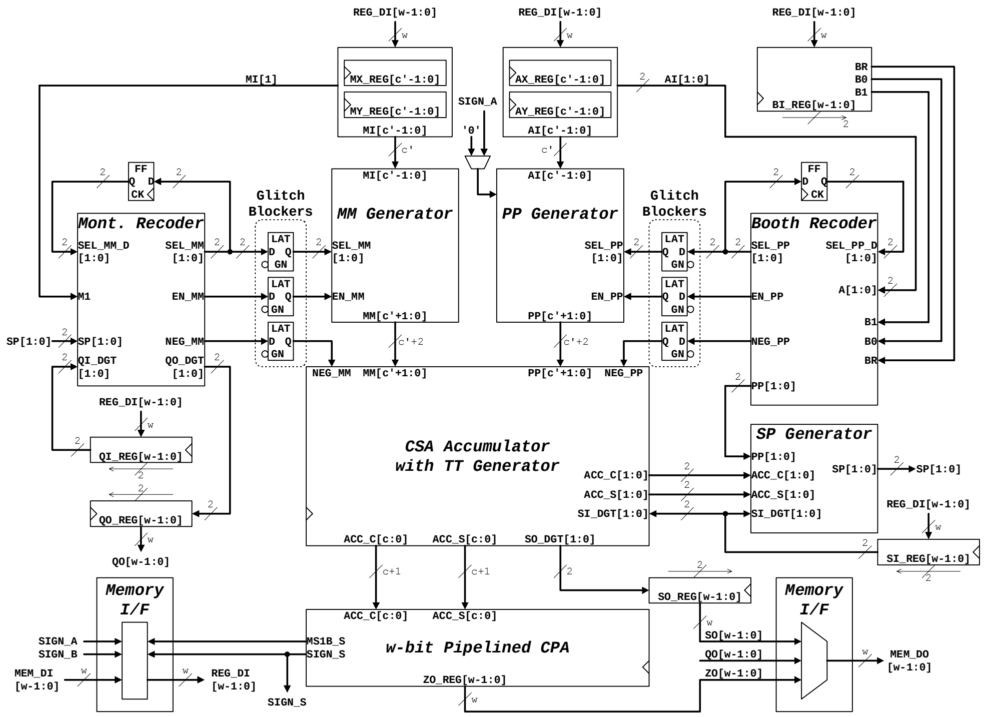

Fig 5. Block Diagram of Data Path