Method for hardware implementation of a convolutional turbo code interleaver and a sub-block interleaver

Disclosed is a method for hardware implementation of a convolutional turbo code interleaver and a sub-block interleaver. Benefits include improved functionality and improved performance.

Background

Convolutional turbo code (CTC) is an error correction technology. It is standardized in the 802.16e-2005 specification titled “IEEE Standard for Local and Metropolitan Area Networks Part 16: Air Interface for Fixed and Mobile Broadband Wireless Access Systems Amendment for Physical and Medium Access Control Layers for Combined Fixed and Mobile Operation in Licensed Bands”, published December 7, 2005 by the Institute of Electrical and Electronics Engineers, Inc. The name 802.16e is a registered trademark of the Institute of Electrical and Electronics Engineers, Inc. The name IEEE is a trademark of the Institute of Electrical and Electronics Engineers, Inc.

CTC interleaver

The CTC interleaver is a two step interleaver and is defined for a particular block size, designated as *N*. Parameters for the block size, modulation, and coding rate are designated as P0, P1, P2, and P3. For example, for QPSK modulation, a block size of 6 bytes (uncoded) and a code rate of ½ results in $N = 24$, $P0 = 5$, $P1 = 0$, $P2 = 0$, $P3 = 0$. A pseudocode example illustrates the standardized CTC interleaver code (see Figure 1).

The conventional hardware implementation generates an address (see Figure 2).

Processing begins by initializing the storage element to a value of 1. For an increasing value of *J*, the addition amount can be dynamically changed. The addition amount depends on the least significant bits of *J* and select the difference between $P(j+1)$ and $P(j)$. For example, if the pseudo code is executing case 2, for the next interleaver address, case 3 is executed. The difference in the values of $P(j+1)$ and $P(j)$ is the case 3 input to the multiplexer, mux1.

A *mod N* operation can be performed. For example, if $(x+y) \bmod N$ must be calculated, two simultaneous calculations are performed:

$x + y$

$x + y - N$

If x and y are in the range of $0 <= x, y < N$, then the sum $(x + y)$ is in the range of:

$0 <= x + y < 2*N$

As a result, the value of $–N$ is as follows:
$-N <= x + y –N < N$

The following test is performed: If $(x+y-N)$ is negative, then $(x + y) \bmod N = (x + y)$; else $(x + y) \bmod N = x + y – N$.

The check for whether $(x + y –N)$ is negative is simple and can be calculated by checking the most significant bit (MSB) of the sum $(x + y – N)$.

The critical path is 2 muxes, 1 adder, and 1 carry save adder (CSA). If the less-than operator is assumed to be as complex as an adder, the approximate critical path is 3 adders.

The constant values for parameters *P0*, *P1*, *P2*, *P3* for different block sizes *N* as specified in the 802.16e standard are listed (see Figure 3).

Sub-block interleaver

The conventional method for implementing a general block interleaver is to use an interleaver look-up table. Each entry of the table is the offset to the location in the data buffer from which the current value should be read. The data is read by using the base address of the data buffer and the offset retrieved from the interleaver table. However, this operation is memory intensive when a large number of block sizes must be supported.

A very effective forward error correction (FEC) technique encodes a number of codewords and shuffles (interleaves) the encoded bits. Interleaving reorders the symbols in a group of transmitted codewords so that adjacent symbols in the data stream are not from the same codeword. The sub-block interleaver (SBI) for 802.16 modems interleaves the outputs of the CTC encoder before the output is digitally modulated.

The address generator used for the SBI can be reused to interleave six encoded sub-blocks and deinterleave six depunctured link-level retry (LLR) sub-blocks. Corresponding entries of the sub-blocks can be packed to minimize the number of memory-move operations.

The sub-block interleaver processing in 802.16e occurs after the CTC encoder and before the puncturing block. The sub-block deinterleaver processing occurs after the depuncturing block and before the CTC decoder. The same address generator is used for interleaving and deinterleaving, including the two systematic blocks (*A*, *B*) and the four parity blocks (*Y1*, *Y2*, *W1*, *W2*). The generated address depends on the two parameters (*m*, *J*) specified by the sub-block size *N* . The sub-block interleaver parameters for different sub-block sizes are listed (see Figure 4).

The address mapping is standardized. During the interleaving of each of the six sub-blocks, the bit at address T_k in the interleaver memory is moved to the bit location *k* where $k=0,1,...N-1$. For interleaving, the write into memory is linear, and the read out of memory is based on the nonlinear address T_k. Given a sub-block size *N* with parameters {*m*, *J*}, the mapping from *k* to T_k is standardized using the following steps:

1. Initialize:
$k=0; i = 0$
2. Form a tentative address:
$T_k = 2^m (k \bmod J) + \text{Bit reverse(m-bit version of floor}(k/J))$ [Equation 1]
3. If $T_k < N$, then T_k is a valid address, increase k, i; else discard T_k, increase k, regenerate T_k, using step 2.
4. Repeat steps 2 and 3 until all N interleaver output addresses are generated.

A conventional hardware implementation generates an address. The m least significant bit (LSB) address bits are generated for each k-th bit by performing the following steps:
1. Increment an m-bit forward counter every J cycles.
2. Bit reverse the count in hardware (B_k).
3. Create three candidates $\{B_k\ B_{k+1}\ B_{k+2}\}$, corresponding to k, $k+1$, and $k+2$.
4. Select the valid address, T_k, from among $\{B_k\ B_{k+1}\ B_{k+2}\}$, which is the first entry less than N.

This method has the drawbacks of hardware complexity and latency.

General description

The disclosed method includes hardware implementations of a convolutional turbo code interleaver and a sub-block interleaver. The critical path of the CTC interleaver is comprised of two muxes and one adder, if a carry save adder is used. If the less-than operator is as complex as an adder, the approximate critical path is 1 adder. The sub-block interleaver generates an address using a concatenation of the bit outputs of a 2-bit forward counter (FC) and a bit-reversed counter (BRC).

Advantages

The disclosed method provides advantages, including:
- Improved functionality due to providing hardware implementations of a convolutional turbo code interleaver and a sub-block interleaver
- Improved performance due to reducing latency in the address generation of the CTC interleaver
- Improved performance due to simplifying the critical processing paths
- Improved performance due to enabling faster cycle times

Detailed description

The disclosed method includes hardware implementations of a convolutional turbo code interleaver and a sub-block interleaver.

CTC interleaver

The disclosed method is a hardware implementation of the CTC interleaver. For example, the method generates an address (see Figure 5).

The critical path is comprised of two muxes and one adder (if the CSA is used). If the less-than operator is assumed to be as complex as an adder, the approximate critical path is 1 adder.

The modified parameter values, designated as *A0*, *A1*, *A2*, and *A3*, are input to mux1 (see Figure 6).

Sub-block interleaver

The disclosed method includes a hardware implementation of the sub-block interleaver. The generated address (T_k) is formed as a concatenation of the bit outputs of a 2-bit forward counter (FC) and *m*-bit bit-reversed counter (BRC). The address T_k is represented by the binary word (f_1 f_0 b_{m-1} b_{m-2}... b_1 b_0), with *m* ranging from 3 to 10 (see Figure 7).

The 6-bit address generation for a sub-block size $N = 36$ with parameters $m = 4$ and $J = 3$ is represented in a table (see Figure 8).

The *FCm*, which accounts for the term involving mod *J* (in Equation 1), requires 2 bits, as $ceil(\log_2(J)) \leq 2$. The ripple counter FC is incremented (*FC++*) every clock cycle *k* (k = 0,1,…N-1), while the BRC is incremented (*BRC(++)*) when $Tk \geq N$. With the FC, the increment of 1 is added to the LSB (f_0) of the current count and the resultant carry bits are propagated from right to the left. The BRC is a transposed (flip-left-to-right) version of the FC. With the BRC, the increment of 1 is added to its MSB (b_{m-1}), and the resultant carry bits are propagated from left to right. The bit reversing can be implemented with no extra hardware.

The disclosed method is an efficient hardware implementation of the logic for SBI address generation. The BRC is a 10-bit bit-reversed counter to accommodate the full range of *m*. Depending on *N*, the bit selector picks the *m* MSB bits of the BRC from the left, and T_k is formed by the binary word (f_1 f_0 b_{m-1} b_{m-2}... b_1 b_0). The forward counter, FC, and the reverse counter BRC operate in parallel. FC is incremented every cycle, unless the resultant address is greater than *N* as indicated by the sign bit of the comparison. In that case, FC is reset. The bit-reversed counter is incremented whenever the forward counter is reset. For the bit-reversed counter, a simple incrementor is illustrated, but in actual implementation, a look-ahead structure is used (see Figure 9).

The most significant $m+2$ bits of the resultant address must be compared against *N* (the block size). The disclosed method supports the worst case of *N* (2400) for which 10 bits are required for the BRC. For all other cases of block size *N*, the complete 12 bits (10 for bit reversed counter, 2 for forward counter) are compared against a value of *N** (see Figure 4):
$N^* = N << 10 - m$

For 1-cycle address generation, the operations of comparison, resetting/incrementing and ($m+2$) bit selection must be performed in 1 cycle. To prevent latency, the critical path can be split by performing the comparison in the previous cycle by performing $T_{k-1} \geq (N^* - 2^{10})$. Alternatively, an advanced version of T_k, which is available at the D-input of the flip-flop in the previous cycle, can be compared against *N*.

Further simplification of the comparison hardware is possible by creating a state machine that operates directly on the counter bits to set FC and BRC appropriately when $T_k \geq N$. Assuming 12 gates per flip-flop, 10 gates per bit for comparison, and 6 bits per bit for selection, the approximate gate count of the implementation is:
*(M+2) * (12 +10 + 6) = 28*M + 44* gates

For *M = 10*, the number of gates is 324.

```
Step 1: for j = 0 .. N – 1
   If (j mod 2 = 1)
      Then, (B, A) = (A, B) (switch the couple)

Step 2: Interleaved vector (j) = Original Vector (P(j))
   For j = 0, 1, .. N-1
      Switch j mod 4:
      Case 0: P(j) = (P0 * j + 1) mod N
      Case 1: P(j) = (P0 * j + 1 + N/2 + P1) mod N
      Case 2: P(j) = (P0 * j + 1 + P2) mod N
      Case 3: P(j) = (P0 * j + 1 + N/2 + P3) mod N
```

Fig. 1

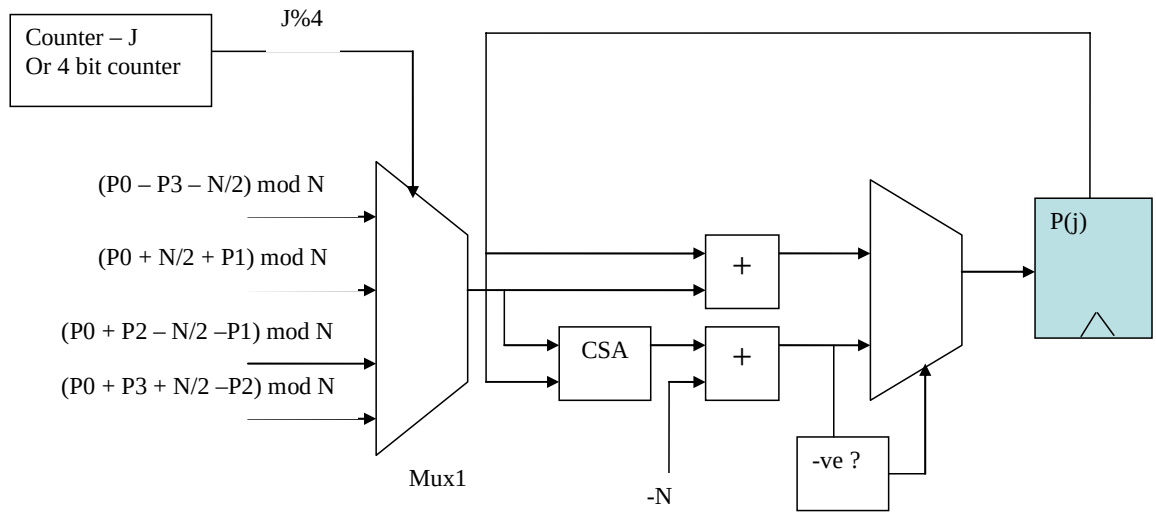

Fig. 2

N	P0	P1	P2	P3
24	5	0	0	0
36	11	18	0	18
48	13	24	0	24
72	11	6	0	6
96	7	48	24	72
108	11	54	56	2
120	13	60	0	60
144	17	74	72	2
180	11	90	0	90
192	11	96	48	144
216	13	108	0	108
240	13	120	60	180

Fig. 3

Operational modes	Data size bits	N = sub-block size	m	J
Normal & HARQ	48	24	3	3
Normal	72	36	4	4
Normal & HARQ	96	48	4	3
Normal & HARQ	144	72	5	3
Normal & HARQ	192	96	5	3
Normal	216	108	6	3
Normal	240	120	6	2
Normal & HARQ	288	144	6	3
Normal & HARQ	384	192	6	3
Normal	432	216	6	4
Normal & HARQ	480	240	7	2
HARQ	960	480	8	2
HARQ	1920	960	9	2
HARQ	2880	1440	9	3
HARQ	3840	1920	10	2
HARQ	4800	2400	10	3

Fig. 4

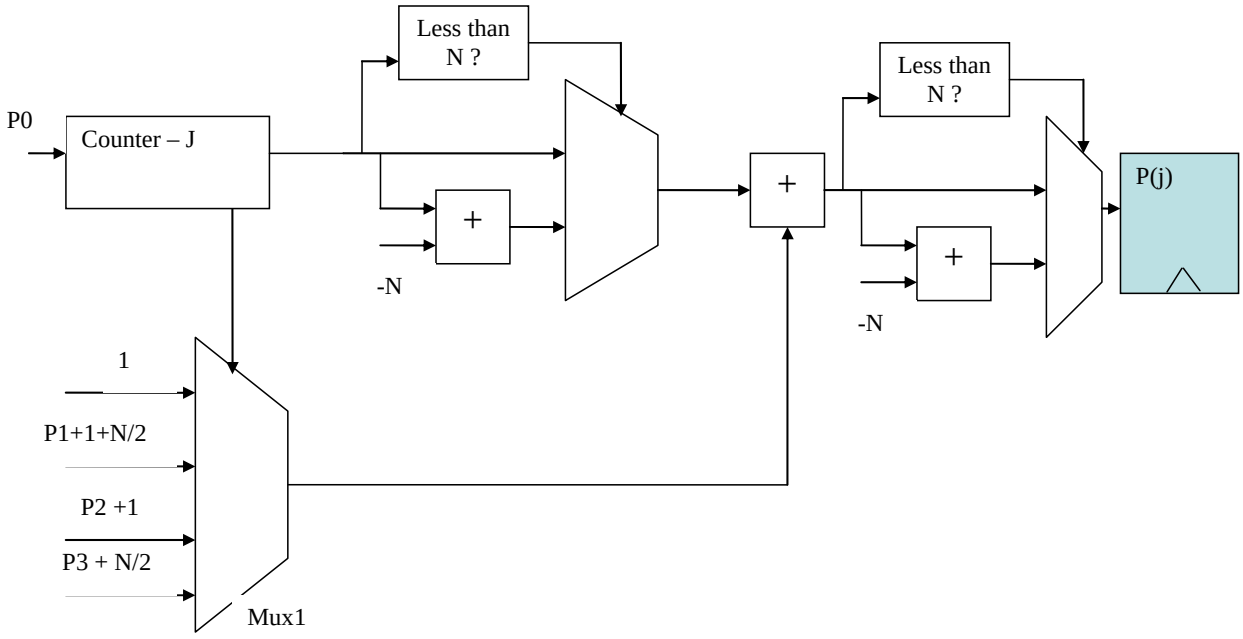

Fig. 5

N	A0	A1	A2	A3
24	17	17	17	17
36	11	11	11	11
48	13	13	13	13
72	41	53	41	53
96	79	7	31	7
108	63	11	67	11
120	13	13	13	13
144	87	19	87	19
180	11	11	11	11
192	155	11	59	11
216	13	13	13	13
240	193	13	73	13

Fig. 6

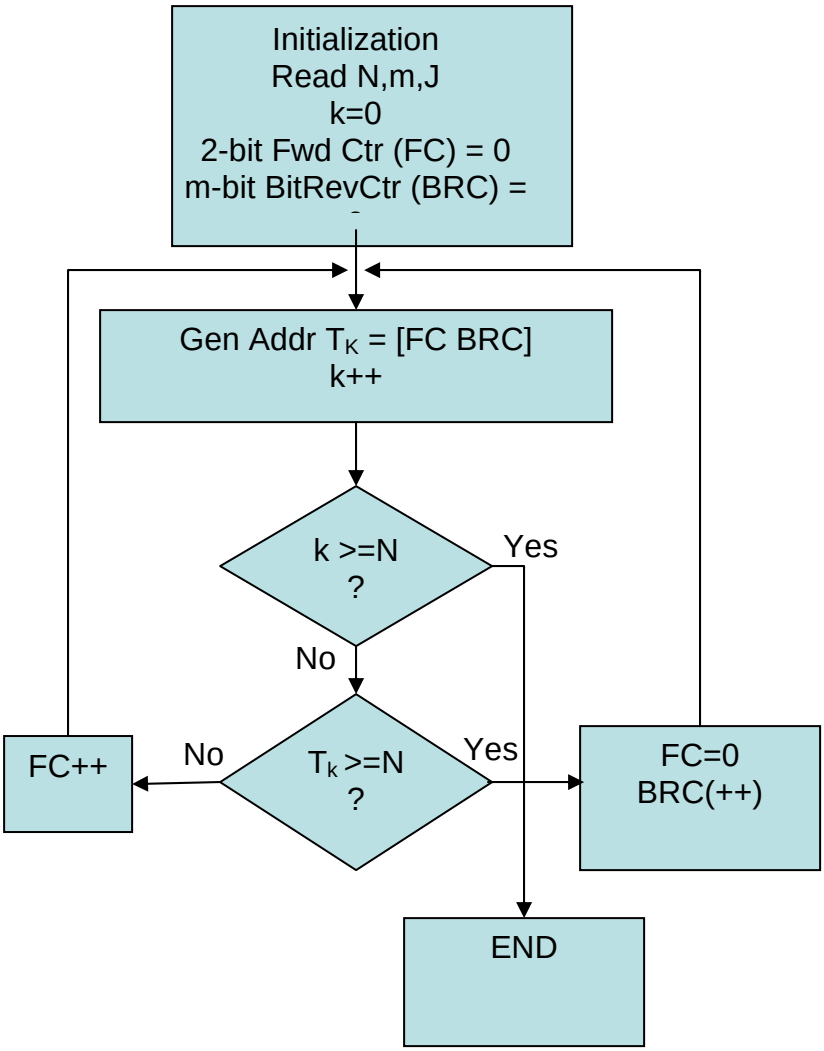
Initialization
Read N,m,J
k=0
2-bit Fwd Ctr (FC) = 0
m-bit BitRevCtr (BRC) =
Gen Addr T_K = [FC BRC]
k++
k >=N
?
Yes
No
T_k >=N
?
No
Yes
FC++
FC=0
BRC(++)
END

Fig. 7

Illustration of SBI address generation for N = 36; m = 4, J = 3

Cycl k	FwdCtr (FC) [f_1 f_0]	BitRevCtr (BRC) [b_{m-1}..b_1 b_0]	Addr T_k= [f_1..b_0] (deci)	T_k >= N-2^m	Action for k+1 cycle (Note: N-2^m = 10 0100 - 01 0000 = 01 0100)
0	0 0	0 0 0 0	0		Incr FC
1	0 1		16		Incr FC
2	1 0		32	Yes	Set FC = 0, Incr BRC
3	0 0	1 0 0 0	8		Incr FC
4	0 1		24	Yes	Set FC = 0, Incr BRC
5	0 0	0 1 0 0	4		Incr FC
6	0 1		20	Yes	Set FC = 0, Incr BRC
7	0 0	1 1 0 0	12		Incr FC
8	0 1		28	Yes	Set FC = 0, Incr BRC
9	0 0	0 0 1 0	2		Incr FC
10	0 1		18		Incr FC
11	1 0		34	Yes	Set FC = 0, Incr BRC
12	0 0	1 0 1 0	10		Incr FC
13	0 1		26	Yes	Set FC = 0, Incr BRC
14	0 0	0 1 1 0	6		Incr FC
15	0 1		22	Yes	Set FC = 0, Incr BRC
16	0 0	1 1 1 0	14		Incr FC
17	0 1		30	Yes	Set FC = 0, Incr BRC
18	0 0	0 0 0 1	1		Incr FC
19	0 1		17		Incr FC
20	1 0		33	Yes	Set FC = 0, Incr BRC
21	0 0	1 0 0 1	9		Incr FC
22	0 1		25	Yes	Set FC = 0, Incr BRC
23	0 0	0 1 0 1	5		Incr FC
24	0 1		21	Yes	Set FC = 0, Incr BRC
25	0 0	1 1 0 1	13		Incr FC
26	0 1		29	Yes	Set FC = 0, Incr BRC
27	0 0	0 0 1 1	3		Incr FC
28	0 1		19		Incr FC
29	1 0		35	Yes	Set FC = 0, Incr BRC
30	0 0	1 0 1 1	11		Incr FC
31	0 1		27	Yes	Set FC = 0, Incr BRC
32	0 0	0 1 1 1	7		Incr FC
33	0 1		23	Yes	Set FC = 0, Incr BRC
34	0 0	1 1 1 1	15		Incr FC
35	0 1		31	Yes	Set FC = 0, Incr BRC

Fig. 8

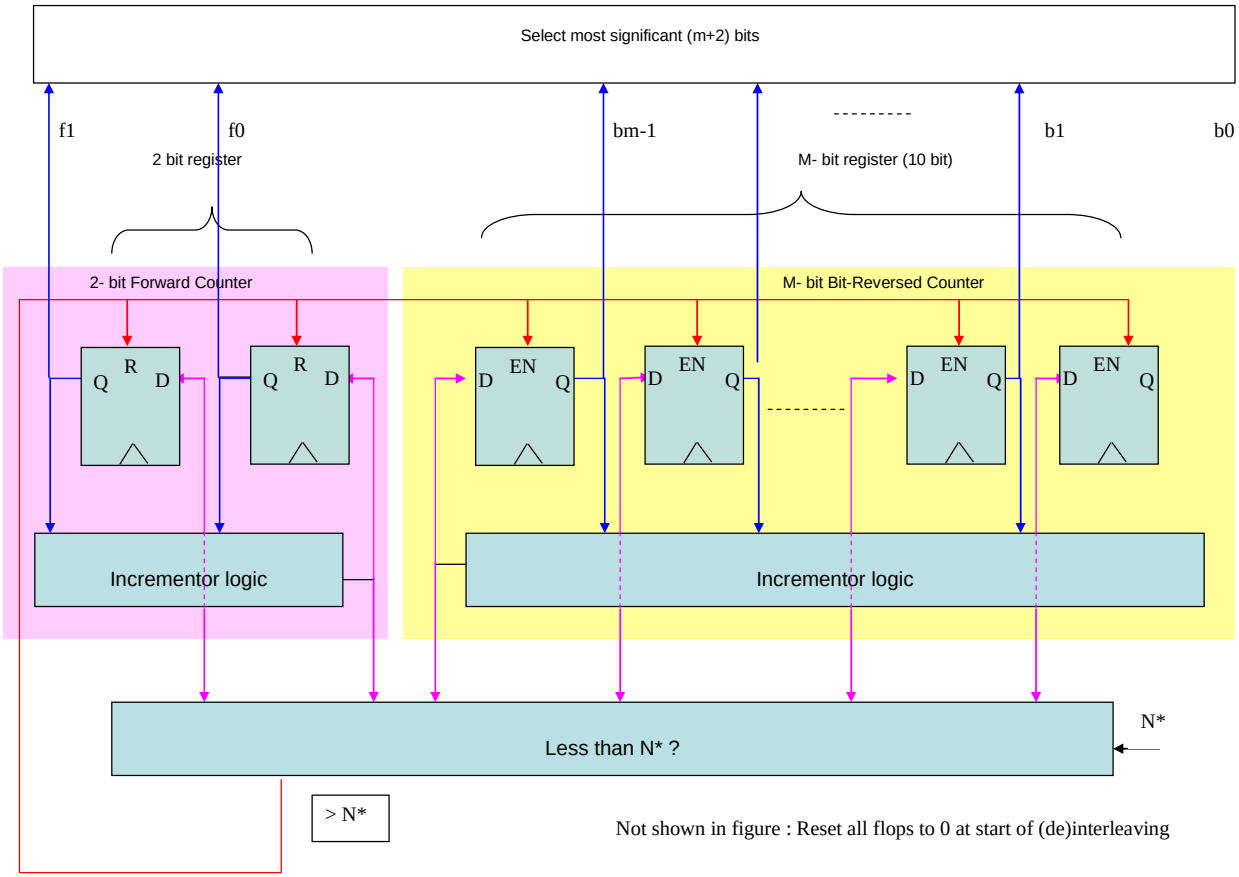

Fig. 9

Disclosed anonymously