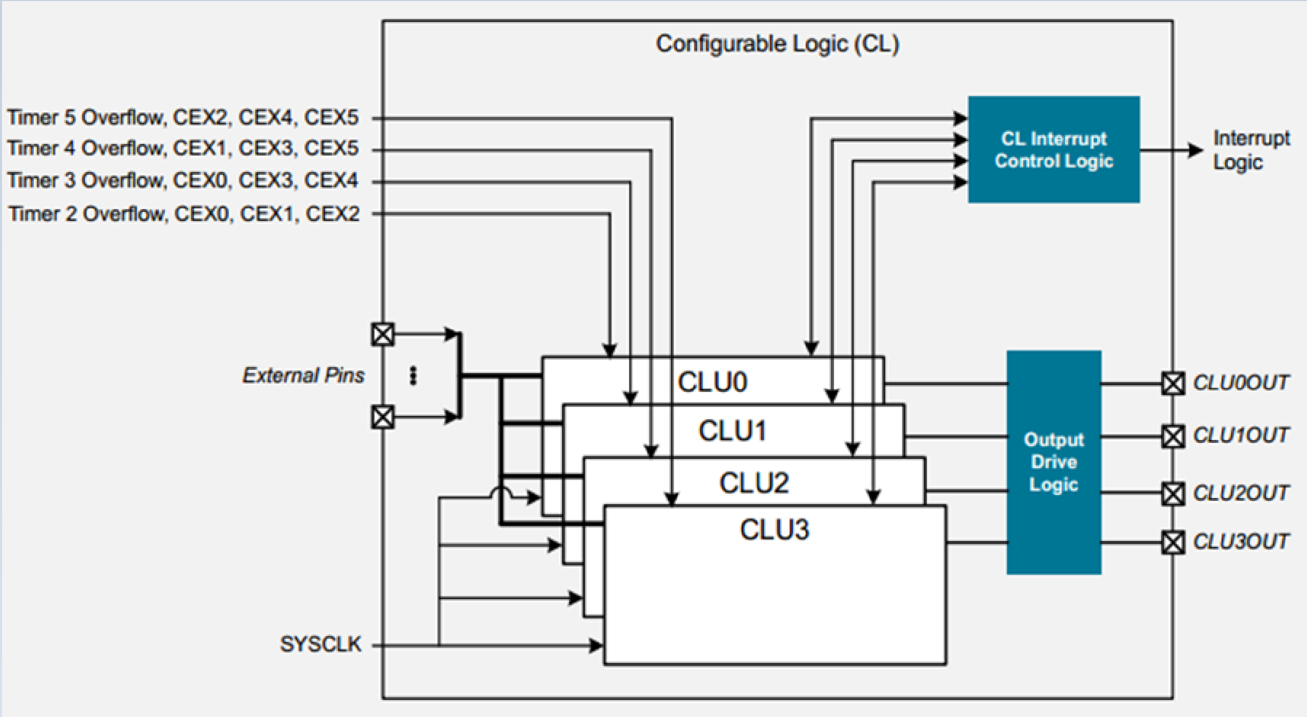
EFM8LB1 – Configurable Logic Unit (CLU)

18 NOV 2015

Agenda

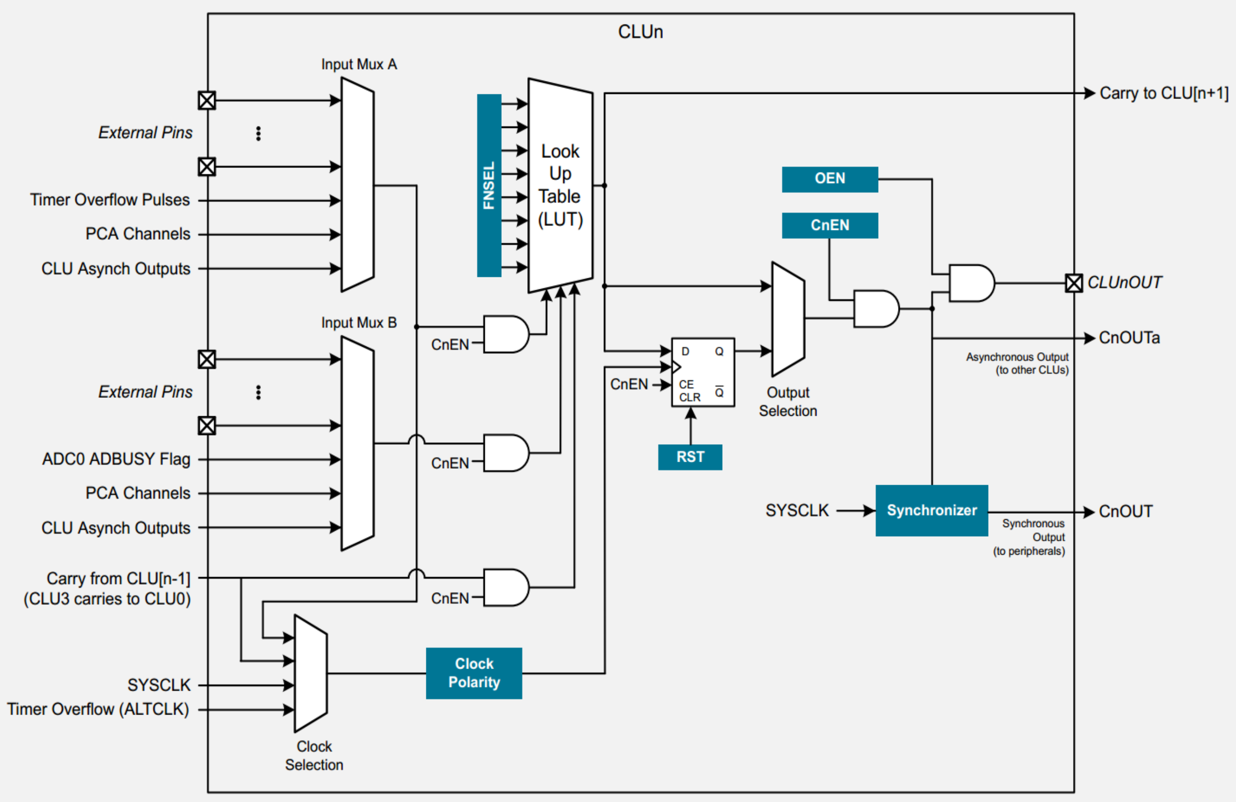
- CLU Block Overview
- Input Multiplexer Selection
- Output configuration
- LUT configuration
- How to Configure CLU
- Demos & Software Examples
  - Demo on STK (AND, NAND, OR, NOR, XOR, XNOR)
  - Peripheral Driver
- App Note AN921
  - SR Latch
  - D Latch
  - Button Debounce
  - Manchester Encoder/Decoder
  - Biphase Mark Code Encoder/Decoder

CLU Block Overview



- Configurable Logic Units(CLUs) provides user-programmed digital logic.
- Four CLUs
- Three inputs for each CLU
- Internal and external signals as inputs to each CLU
- Output to Port IO pins.
- Output to a peripheral input.

Individual CLU Overview



- Look up table (LUT) supports 256 different logic functions with three inputs (MXA, MXB, Carry)
- CLUnFN value implements desired logic.
- May be cascaded together to perform more complicated logic functions.
- A D flip-flop whose input from LUT output.
- Output
  - Fixed output pin assignment
    - CLU0-3 output pins are P0.2, P1.0, P2.2 and P2.5
  - Asynchronous output may be used as other CLU input.
  - Synchronous output triggers ADC, DAC, Timers, etc.
- Interrupt
  - CLU output rising edge
  - CLU output falling edge

Input Multiplexer Selection

- Each CLU has two primary logic inputs (A and B) and a carry input (C).
- The A and B inputs are selected by the MXA and MXB fields in the CLUnMX register.
- MXA and MXB input may be one of many different internal and external signals.
- The carry input, C, is the LUT output of the previous CLU.
  - CLU0's LUT output is CLU1 carry input.
  - CLU3's LUT output is CLU0 carry input.

CLUnA Input Selection

CLUnMX.MXA	CLU0A	CLU1A	CLU2A	CLU3A
0000	C0OUTa	C0OUTa	C0OUTa	C0OUTa
0001	C1OUTa	C1OUTa	C1OUTa	C1OUTa
0010	C2OUTa	C2OUTa	C2OUTa	C2OUTa
0011	C3OUTa	C3OUTa	C3OUTa	C3OUTa
0100	Timer2 Overflow	Timer3 Overflow	Timer4 Overflow	Timer5 Overflow
0101	CEX0	CEX0	CEX1	CEX2
0110	CEX1	CEX3	CEX3	CEX4
0111	CEX2	CEX4	CEX5	CEX5
1000	P0.0	P0.4	P0.0	P0.2
1001	P0.2	P0.5	P0.1	P0.3
1010	P0.4	P1.0	P1.0	P0.6
1011	P0.6	P1.2	P1.1	P0.7
1100	P1.0	P1.4	P1.6	P1.2
1101	P1.2	P1.5	P1.7	P1.3
1110	P1.4	P2.0	P2.0	P2.2
1111	P1.6	P2.2	P2.1	P2.3

CLUnB Input Selection

CLUnMX.MXB	CLU0B	CLU1B	CLU2B	CLU3B
0000	C0OUTa	C0OUTa	C0OUTa	C0OUTa
0001	C1OUTa	C1OUTa	C1OUTa	C1OUTa
0010	C2OUTa	C2OUTa	C2OUTa	C2OUTa
0011	C3OUTa	C3OUTa	C3OUTa	C3OUTa
0100	ADBUSY	ADBUSY	ADBUSY	ADBUSY
0101	CEX3	CEX1	CEX0	CEX0
0110	CEX4	CEX2	CEX2	CEX1
0111	CEX5	CEX5	CEX4	CEX3
1000	P0.1	P0.6	P0.2	P0.0
1001	P0.3	P0.7	P0.3	P0.1
1010	P0.5	P1.1	P1.2	P0.4
1011	P0.7	P1.3	P1.3	P0.5
1100	P1.1	P1.6	P1.4	P1.0
1101	P1.3	P1.7	P1.5	P1.1
1110	P1.5	P2.1	P2.2	P2.0
1111	P1.7	P2.3	P2.3	P2.1

Output Configuration

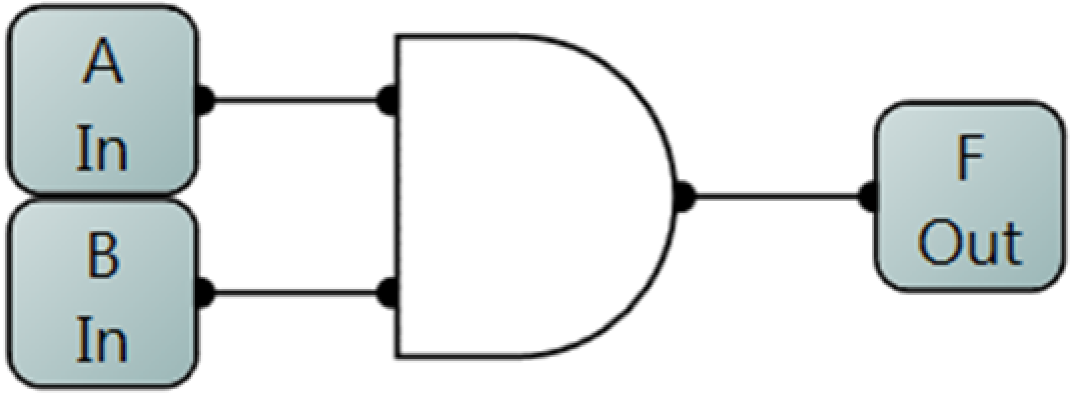
- CLU presents a sync and an async output to the system.
- The CLU output may be present on selected pins
- CLU outputs may be derived directly from the LUT.
- Or from D flip-flop output, whose input from LUT output.
- The D flip-flop clock may be configured from one of four sources
  - CARRY_IN
  - MXA_INPUT
  - SYSCLK
  - ALTCLK (Timer overflows)
- The D flip-flop may be reset to logic 0 by writing 1 to the RST bit in CLUnCF

Look Up Table(LUT) configuration

- The LUT determines the Boolean logic function in each CLU.
- LUT may be changed by programming FNSEL field in register CLUnFN.
- The output of the LUT is selected by the combination of the A, B and C inputs.
- A truth table is used to determine the FNSEL value for a given logic function.

Truth Table

- Truth table shows output of a logic circuit responds to various combinations of the inputs.
- Using logic 1 for true and logic 0 for false.
- A truth table for two inputs AND logic circuit is shown as follows



A	B	Out
0	0	0
0	1	0
1	0	0
1	1	1

Converting Truth Tables Into Boolean Expressions

- Sum of products(SOP) method
  - Write an AND term for each input combination that produces a 1 output
    - Write the input variable if its value is 1; Write its complement otherwise
  - OR the AND terms to get the final expression.
- Product of sum(POS) method
  - Write an OR term for each input combination that produces a 0 output
    - Write the input variable if its value is 0; Write its complement otherwise
  - AND the OR terms to get the final expression
- Ex: Boolean expression for given Truth Table
  - SOP: F = A’B’C’ + A’BC’ + AB’C’
  - POS: F = (A+B+C’)(A+B’+C’)(A’+B+C’)(A’+B’+C)(A’+B’+C’)

A	B	C	Out
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Simplifying Boolean Expression

- Some useful Boolean Laws for simplification
  - **Commutative**: A • B = B • A , A + B = B + A
  - **Associative**: (A • B) • C = A • (B • C), (A + B) + C = A + (B + C)
  - **Identity**: A + 0 = A, A • 1 = A
  - **Distributive**: A • (B + C) = A • B + A • C, A + (B • C) = (A + B) • (A + C)
  - **Complements**: A • A' = 0, A + A' = 1
  - **Double Complement**: (A')' = A
  - **Idempotence**: A • A = A, A + A = A
  - **Dominance**: A • 0 = 0, A + 1 = 1
  - **Absorption Law**: A + (A • B) = A , A • (A + B) = A
  - **De Morgan's theorem**: (A + B)' = A' • B‘, (A • B)' = A' + B'

LUT Configuration example

MXA	MXB	Carry	LUT Output
1	1	1	FNSEL.7
1	1	0	FNSEL.6
1	0	1	FNSEL.5
1	0	0	FNSEL.4
0	1	1	FNSEL.3
0	1	0	FNSEL.2
0	0	1	FNSEL.1
0	0	0	FNSEL.0

- LUT can realize 3 inputs Boolean logic function.
- MXA = 0xF0, MXB=0xCC, Carry = 0xAA
- One input example: F = NOT B. Others don't care
  - NOT B = 0x33
  - FNSEL should be programmed to 0x33.
- Two inputs example: F = A & B. Other doesn't care
  - A & B = 0xF0 & 0xCC = 0xC0
  - FNSEL should be programmed to 0xC0.
- Three Inputs example: F = A & B | C
  - A & B | C = 0xC0 | 0xAA = 0xEA
  - FNSEL should be programmed to 0xEA.

How to Configure – Peripheral Driver

- Peripheral Driver has SI_LUT.h
  - Macro definitions for inputs

```
#define SI_LUT_A    0xf0
#define SI_LUT_B    0xcc
#define SI_LUT_C    0xaa
```

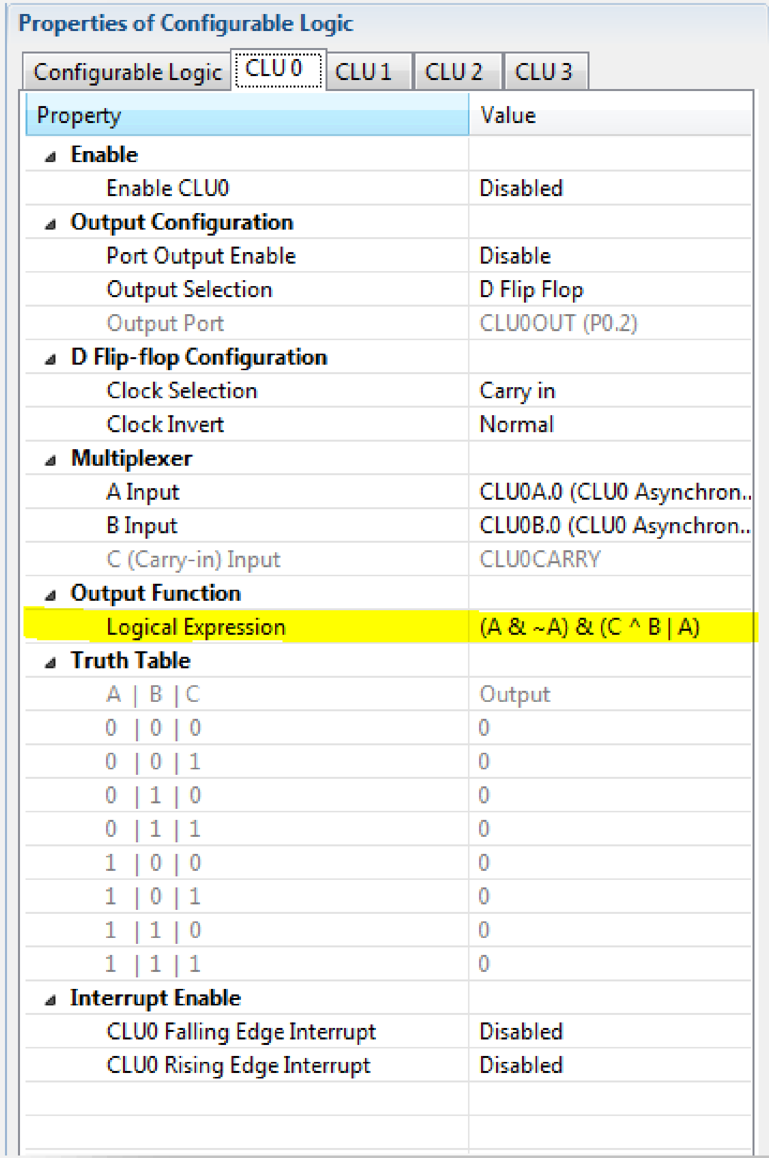
  - Macros for logic functions

```
#define LUT_NOT(a)       ~(a)
#define LUT_AND(a,b)     ((a) & (b))
#define LUT_OR(a,b)      ((a) | (b))
#define LUT_NAND(a, b)   LUT_NOT(LUT_AND(a,b))
#define LUT_NOR(a, b)    LUT_NOT(LUT_OR(a,b))
#define LUT_XOR(a,b)     LUT_OR(LUT_AND(a, LUT_NOT(b)), LUT_AND(LUT_NOT(a), b))
#define LUT_XNOR(a,b)    LUT_NOT(LUT_XOR(a,b))
```

  - Examples

```
CLU0FN = LUT_XOR (SI_LUT_A, SI_LUT_B);
CLU0FN = LUT_NOR (SI_LUT_A, LUT_XOR(SI_LUT_B, SI_LUT_C));
```

How to Configure – Configurator



Properties of Configurable Logic

Configurable Logic | CLU 0 | CLU 1 | CLU 2 | CLU 3

Property	Value
Enable	
Enable CLU0	Disabled
Output Configuration	
Port Output Enable	Disable
Output Selection	D Flip Flop
Output Port	CLU0OUT (P0.2)
D Flip-flop Configuration	
Clock Selection	Carry in
Clock Invert	Normal
Multiplexer	
A Input	CLU0A.0 (CLU0 Asynchron..
B Input	CLU0B.0 (CLU0 Asynchron..
C (Carry-in) Input	CLU0CARRY
Output Function	
Logical Expression	(A & ~A) & (C ^ B | A)
Truth Table	
A | B | C	Output
0 | 0 | 0	0
0 | 0 | 1	0
0 | 1 | 0	0
0 | 1 | 1	0
1 | 0 | 0	0
1 | 0 | 1	0
1 | 1 | 0	0
1 | 1 | 1	0
Interrupt Enable	
CLU0 Falling Edge Interrupt	Disabled
CLU0 Rising Edge Interrupt	Disabled

- Configurator is the easiest method
- Use a logical expression
- Examples
  - AND: A & B
  - OR: A | B
  - XOR: A ^ B
  - NAND: ~(A & B)
  - NOR: ~(A | B)
  - XNOR: ~(A ^ B)

Demo and Software Examples

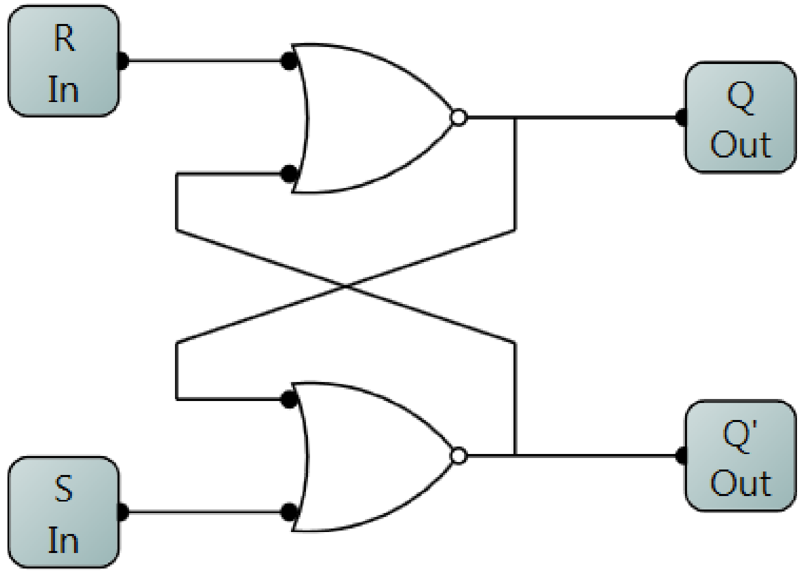
- Demo uses LCD and implements AND, NAND, OR, NOR, XOR, XNOR
- Software Examples do not use LCD
  - AND (without peripheral driver)
  - OR (without peripheral driver)
  - Peripheral Driver
    - AND
    - OR

Application Note AN921

- The AN921 demonstrates how to use CLUs to implement following functions:
  - SR Latch
  - D Latch
  - Button debounce
  - Manchester encoder/decoder
  - Biphase Mark Code encoder/decoder

SR Latch

- SR(SET-RESET) latch, constructed from a pair of cross coupled NOR gates.
- From the structure, we can easily know:
  - When S=R=0, the feedback maintains the Q and Q’ output state
  - When R=0, S=1, then the Q output is forced high.
  - When R=1, S=0, then the Q output is forced low.
  - When R=S=1, both NOR gates then output zero, it is called a forbidden state.



SR Latch - Truth Table and Boolean Equation

- The SR Latch Truth Table and Boolean Equation.
  - The SR latch contains three inputs and one output.

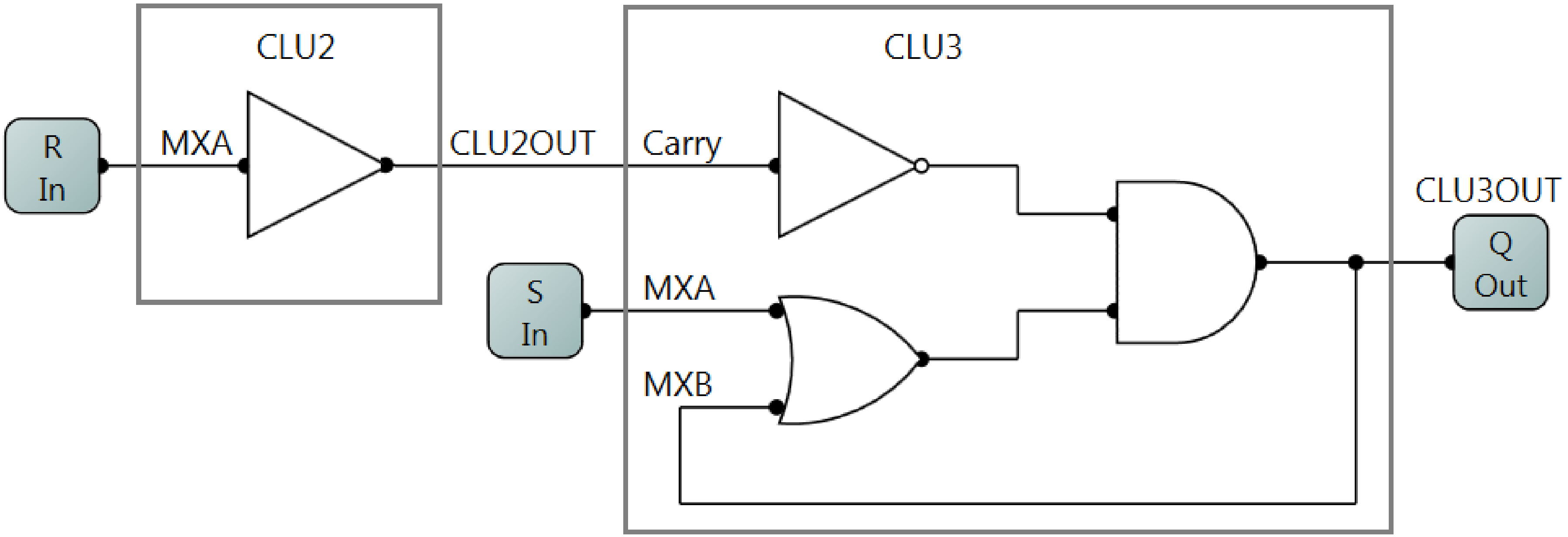
S	R	Q	Q*
1	1	1	0
1	0	1	1
1	1	0	0
1	0	0	1
0	1	1	0
0	0	1	1
0	1	0	0
0	0	0	0

$$
\begin{aligned}
Q^* &= SR'Q+SR'Q'+S'R'Q \\
&= SR'(Q+Q')+S'R'Q \\
&= SR'+S'R'Q \\
&= R'(S+S'Q) \\
&= R'((S+S')(S+Q)) \\
&= R'(S+Q)
\end{aligned}
$$

Q* represents Q next

SR Latch - Implementation

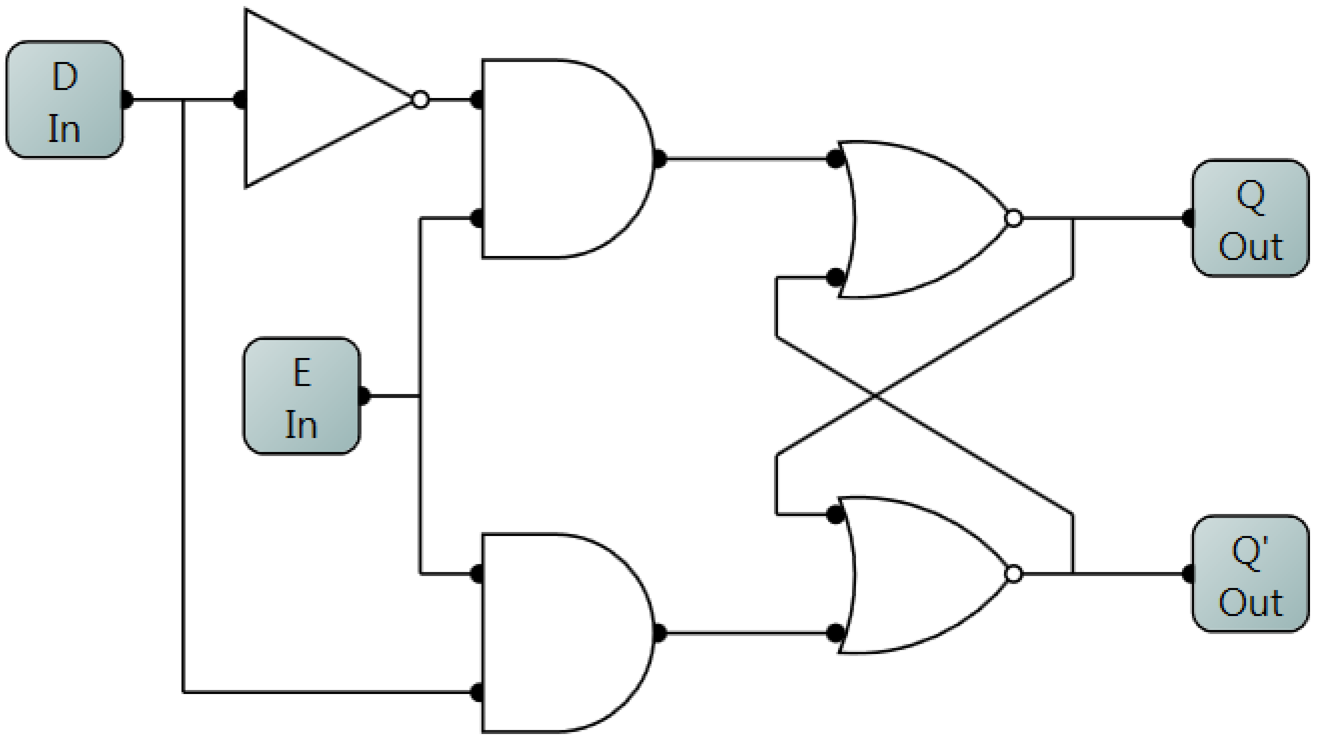
- Using two CLUs for three inputs.




```
CLU2MX = CLU2MX_MXA__CLU2A11 | CLU2MX_MXB__CLU2B0;
CLU2CF = CLU2CF_OUTSEL__LUT;
CLU2FN = SI_LUT_A;  // Buffer MXA
CLU3MX = CLU3MX_MXA__CLU3A10 | CLU3MX_MXB__CLU3B3; // MXB as CLU3out,
CLU3CF |= CLU3CF_OEN__ENABLE | CLU3CF_OUTSEL__LUT;
CLU3FN = LUT_AND(LUT_NOT(SI_LUT_C),LUT_OR(SI_LUT_A,SI_LUT_B)); // Q* = R’(S+Q)
```

D Latch

- The D Latch is also known as transparent latch, data latch or gated latch
- From the structure, we can easily know
  - When E=0, the Q and Q’ output doesn’t change.
  - When E=1, the output Q next captures the D input value.



D Latch - Truth Table and Boolean Equation

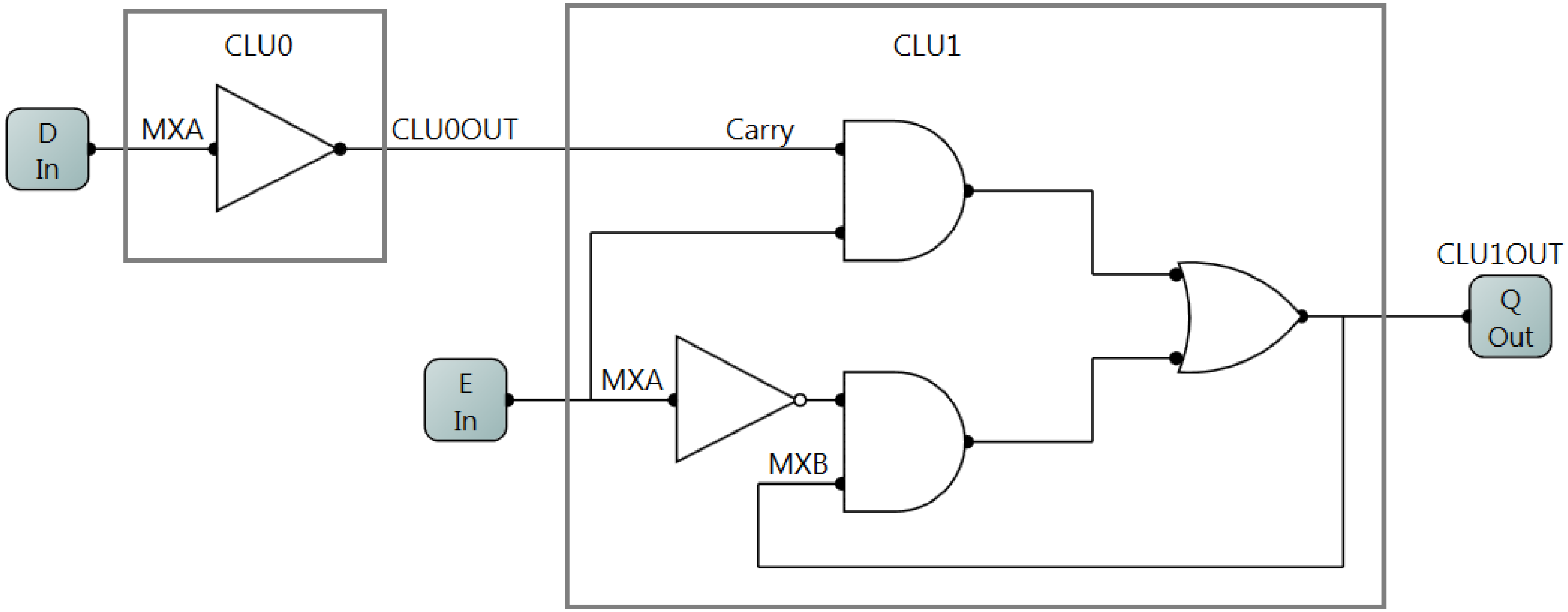
- The D Latch Truth Table and Boolean Equation.
  - The D latch contains three inputs and one output.

E	D	Q	Q*
1	1	1	1
1	0	1	0
1	1	0	1
1	0	0	0
0	1	1	1
0	0	1	1
0	1	0	0
0	0	0	0

```
Q* = EDQ+EDQ’+E’DQ+E’D’Q
   = ED(Q+Q’)+E’Q(D+D’)
   = ED+E’Q

Q* represents Q next
```

D Latch - Implementation

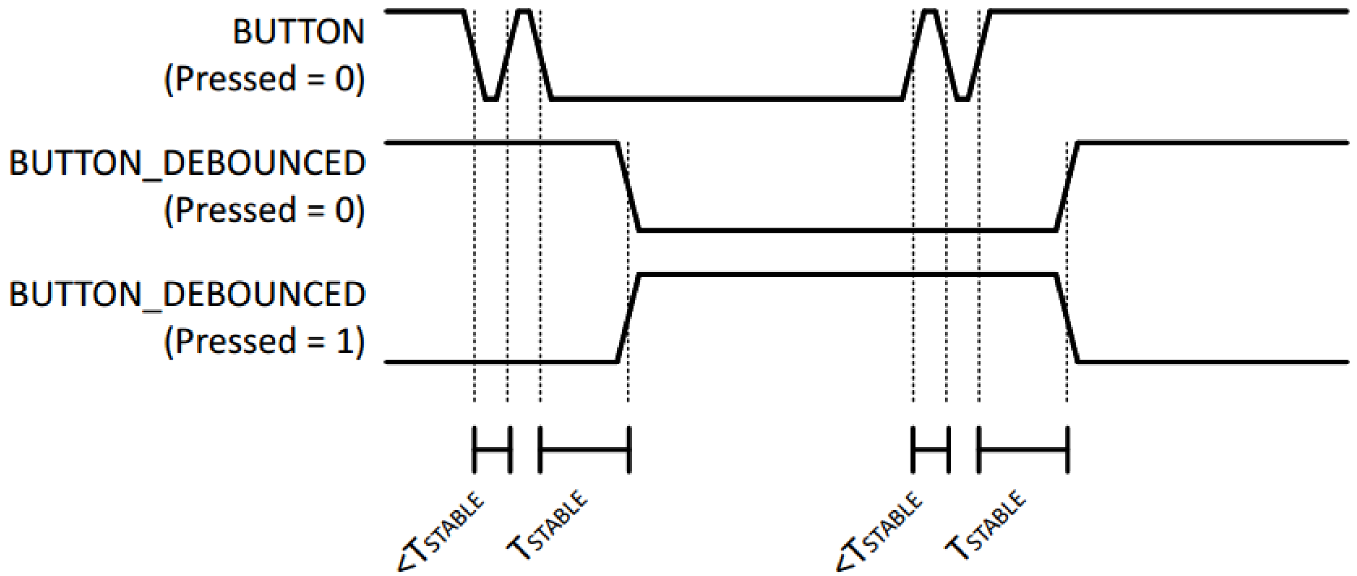
- Using two CLUs for three inputs.




```
CLU0MX = CLU0MX_MXA__CLU0A9 | CLU0MX_MXB__CLU0B0;
CLU0CF = CLU0CF_OUTSEL__LUT;
CLU0FN = SI_LUT_A; // Buffer MXA
CLU1MX = CLU1MX_MXA__CLU1A8 | CLU1MX_MXB__CLU1B1;
CLU1CF = CLU1CF_OEN__ENABLE | CLU1CF_OUTSEL__LUT;
CLU1FN=LUT_OR(LUT_AND(SI_LUT_A,SI_LUT_C),LUT_AND(LUT_NOT(SI_LUT_A),SI_LUT_B));//Q*=ED+E'Q
```

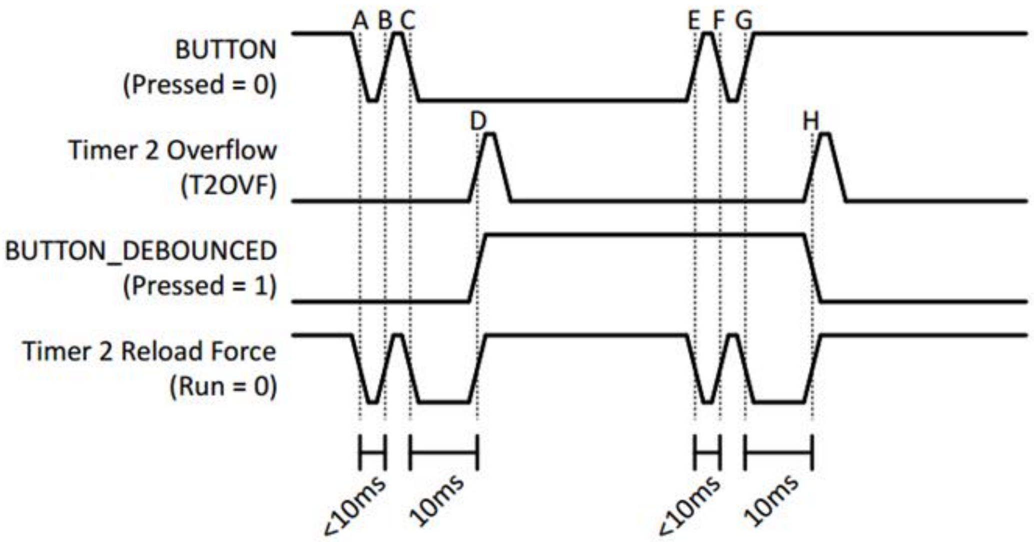
Button Debounce

- Mechanical push buttons generate multiple pulses when pressed or released.
- Typically, the debounce task is performed in firmware.
- CLU can be used to debounce the button without firmware resources.



Timing Diagram of Debounced Button (Simplified)

Button Debounce - Method



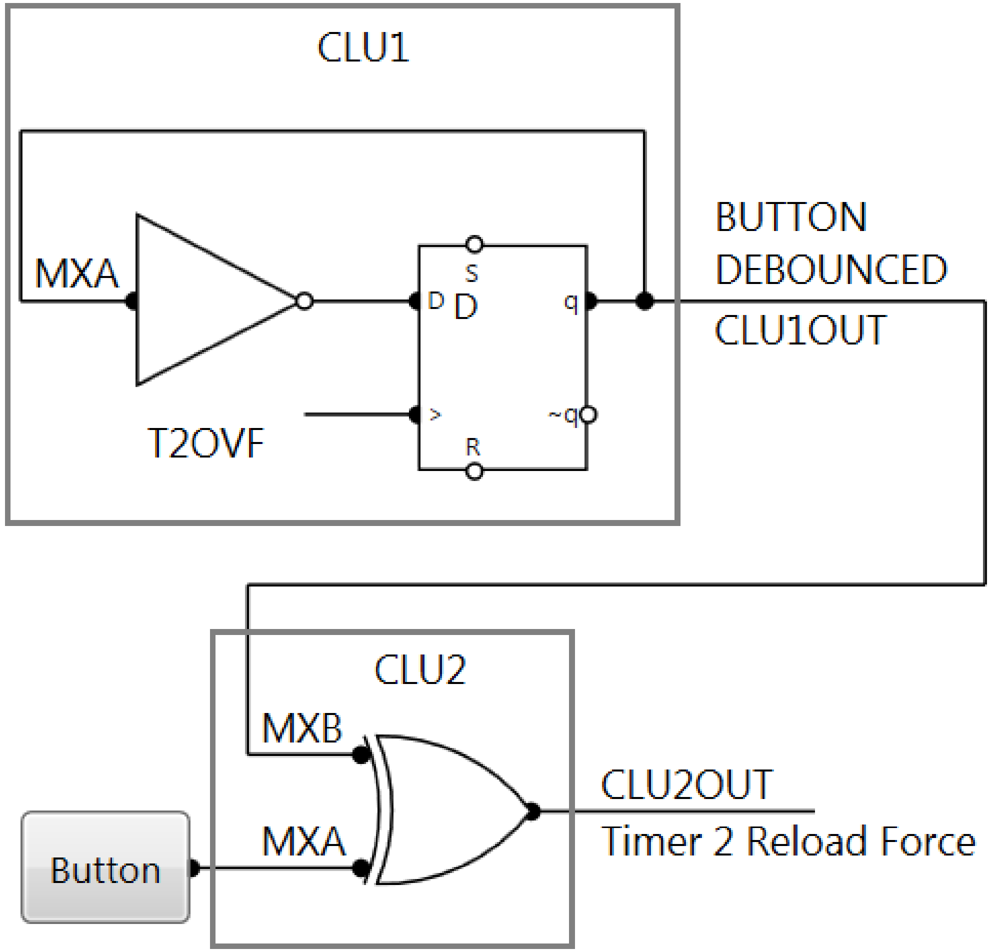
Timing Diagram of Debounced Button (Expanded)

- Button press triggers Timer 2 to run.
- Button bounces high reload Timer 2
- Button keeps low for more than 10 MS
- Timer 2 overflows, triggers debounce output high, and Timer 2 stops.
- Button release high triggers Timer 2 to run.
- Button keeps high for more than 10 MS
- Timer 2 overflows, triggers debounce output low, and Timer 2 stops.

Button Debounce - Implementation

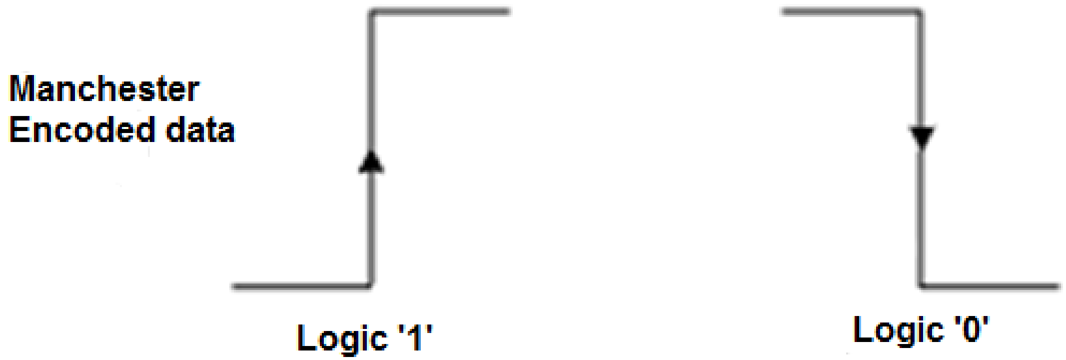
- The CLU1OUT is the debounce button signal.
- No button pressed, Button = 1, CLU1OUT = 0, the CLU2OUT is high, Timer 2 stops
- Button pressed, T2OVF rising edge triggers CLU1OUT = 1, Timer 2 stops.
- Button released, T2OVF rising edge triggers CLU1OUT = 0. Timer2 stops.

```
CLU1MX = CLU1MX_MXA__CLU1A1 | CLU1MX_MXB__CLU1B1;
/* select D flip-flop,  T2OVF as D flip-flop CLK*/
CLU1CF = CLU1CF_OEN__ENABLE | CLU1CF_CLKSEL__ALTCLK;
CLU1FN = LUT_NOT(SI_LUT_A); // Inverse MXA
CLU2MX = CLU2MX_MXA__CLU2A1 | CLU2MX_MXB__CLU2B14;
CLU2CF = CLU2CF_OUTSEL__LUT;
CLU2FN = LUT_XOR(SI_LUT_A, SI_LUT_B);
```



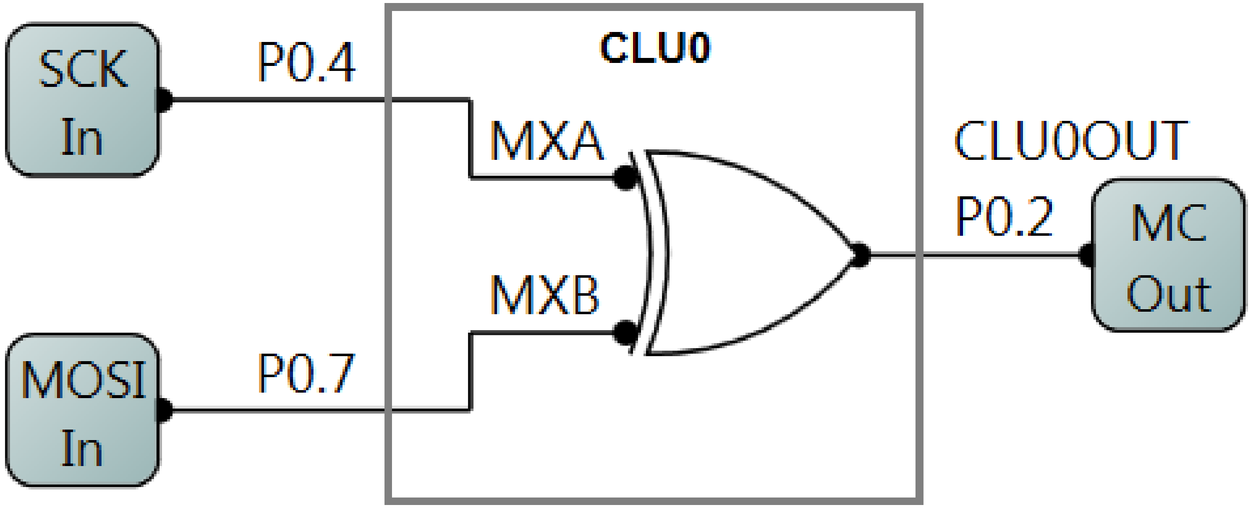
Manchester Code

- Manchester Code is a line code, conveys the data and clock information.
- The encoding of each bit is either low then high, or high then low, of equal time.
  - ‘1’ is represented by a rising edge. (IEEE802.3 standard)
  - ‘0’ is represented by a failing edge. (IEEE802.3 standard)
- It has no DC component, and is self-clocking.
- It is widely used(e.g., in 10BASE-T Ethernet(IEEE802.3))



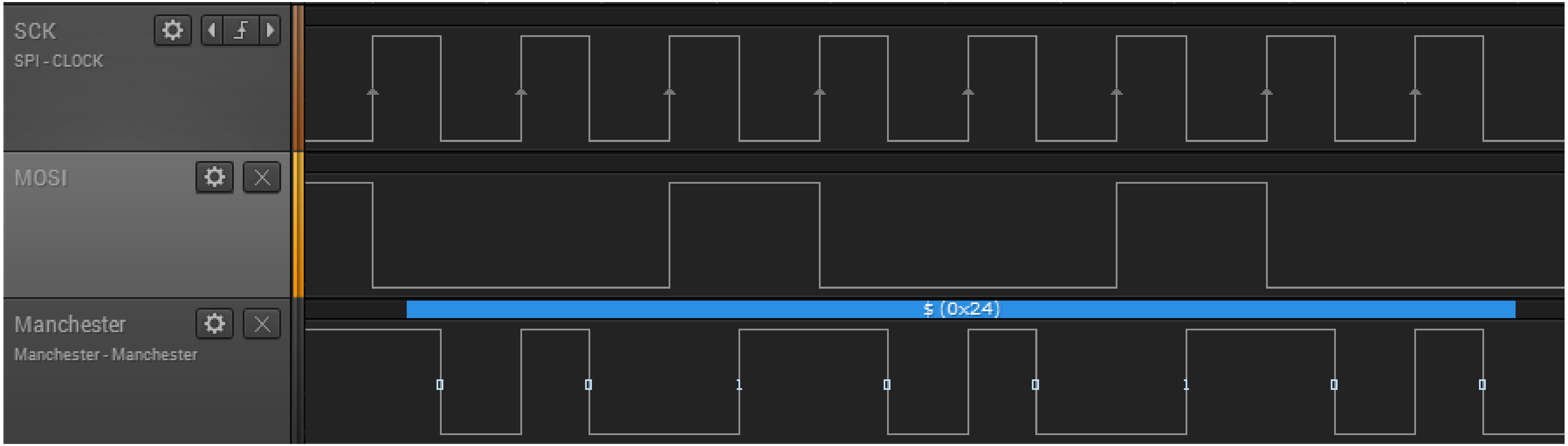
Manchester Encoder - Implementation

- The SPI master and an XOR gate generate Manchester Encoded data
- SCK phase and polarity setting (CKPOL = 0, CKPHA = 1)
- MOSI XOR SCK to obtain Manchester Encoded data



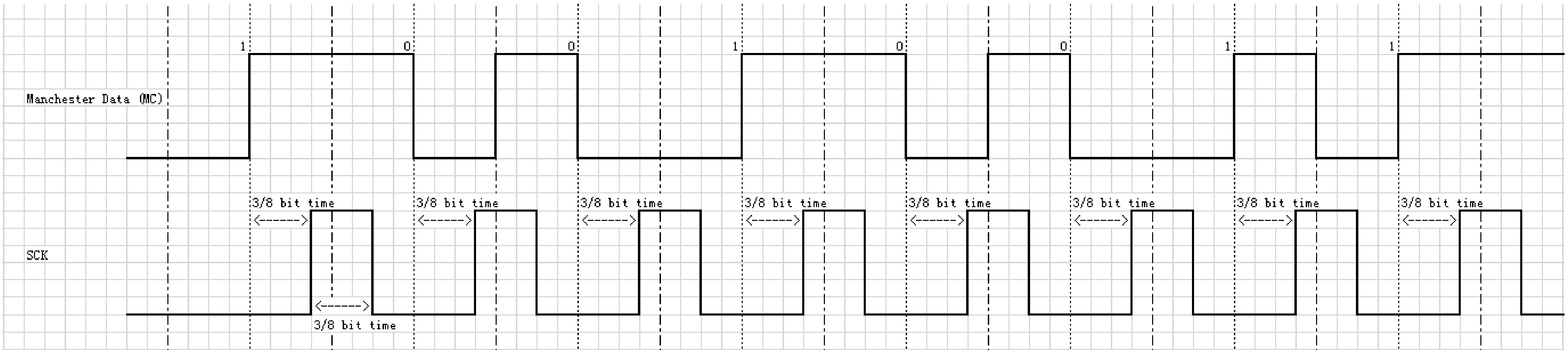

```
SFRPAGE = 0x20;
/* MXA as P0.4, MXB as P0.7*/
CLU0MX = 0xAB;
CLU0FN = LUT_XOR(SI_LUT_A, SI_LUT_B);
CLU0CF = CLU0CF_OEN__ENABLE |
         CLU0CF_OUTSEL__LUT;
CLEN0 |= CLEN0_C0EN__ENABLE;
SFRPAGE = 0;
```

Manchester Encoder - Waveform



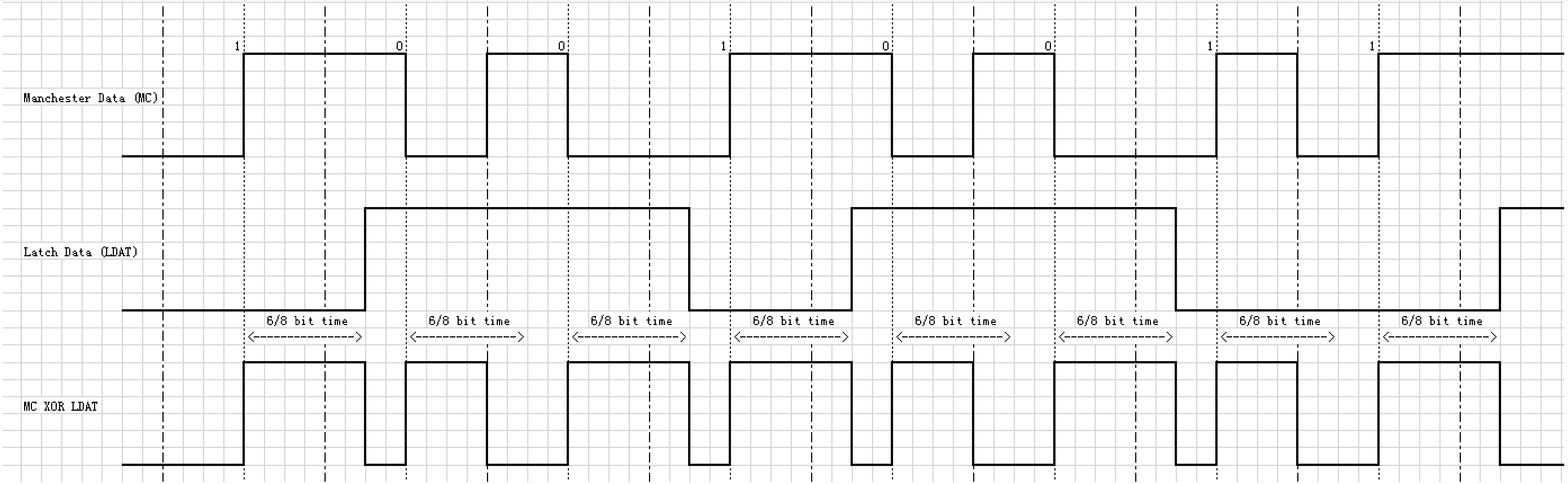
- MOSI = 0, the XORed output follows SCK, it is failing edge = “0”.
- MOSI = 1, the XORed output is an inverted SCK, it is rising edge = “1”

Manchester Decoder – Clock Generation



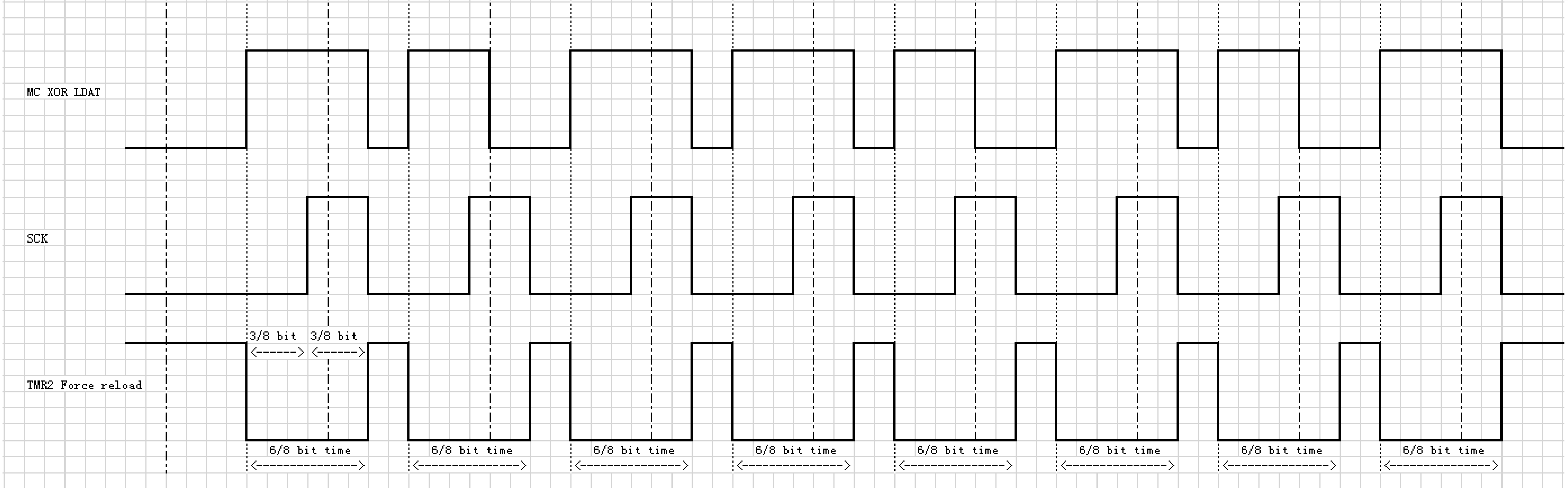
- Manchester bit value is presented in the second half of each bit time.
- The transition in middle of each bit triggers timer with 3/8 bit time.
- Generating SCK rising edge when timer overflow.
- Generating SCK failing edge when timer overflow again and stop the timer.
- Repeat above steps for rest bits.

Manchester Decoder – Clock Trigger Signal Generation



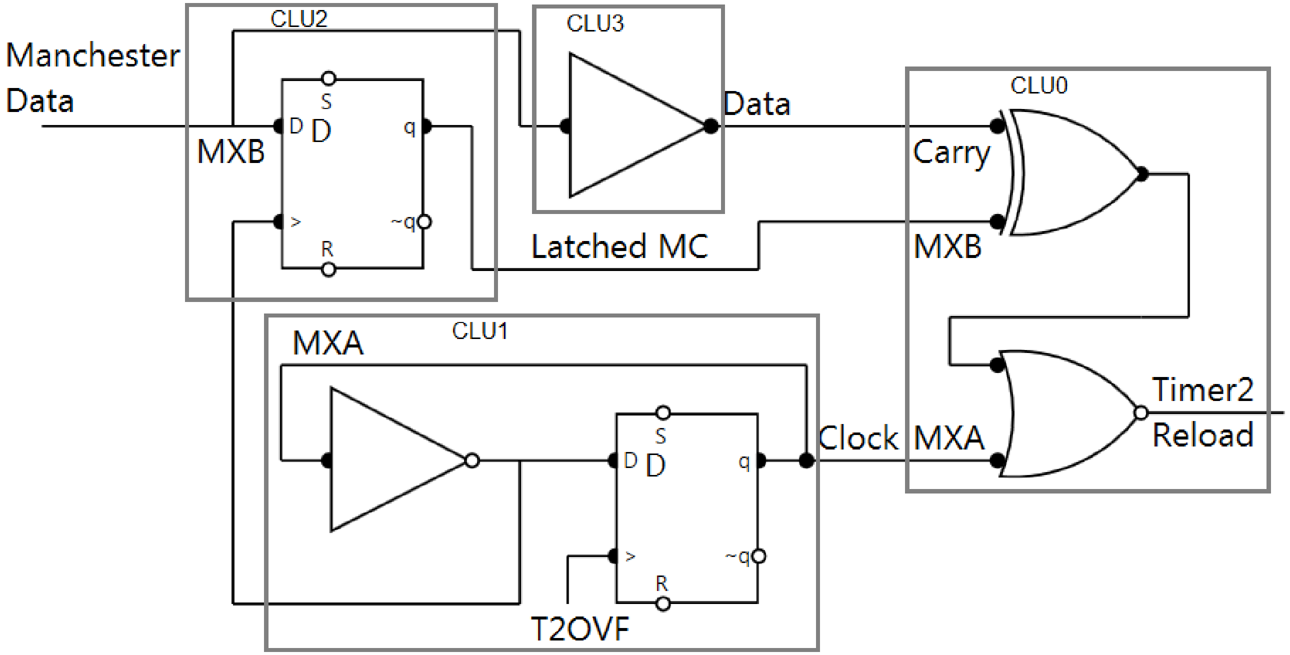
- Generating Latch data by capturing the Manchester data at 6/8 bit time.
- Manchester data XOR Latch data to get rising edge at middle transition.
  - The XOR result at 6/8 bit time must be '0', because Latch data has same value as Manchester data.
  - And then at middle of bit transition, the XOR result change to '1" and generate rising edge.
- The rising edge can triggers a Timer with 3/8 bit time.

Manchester Decoder – Timer Control



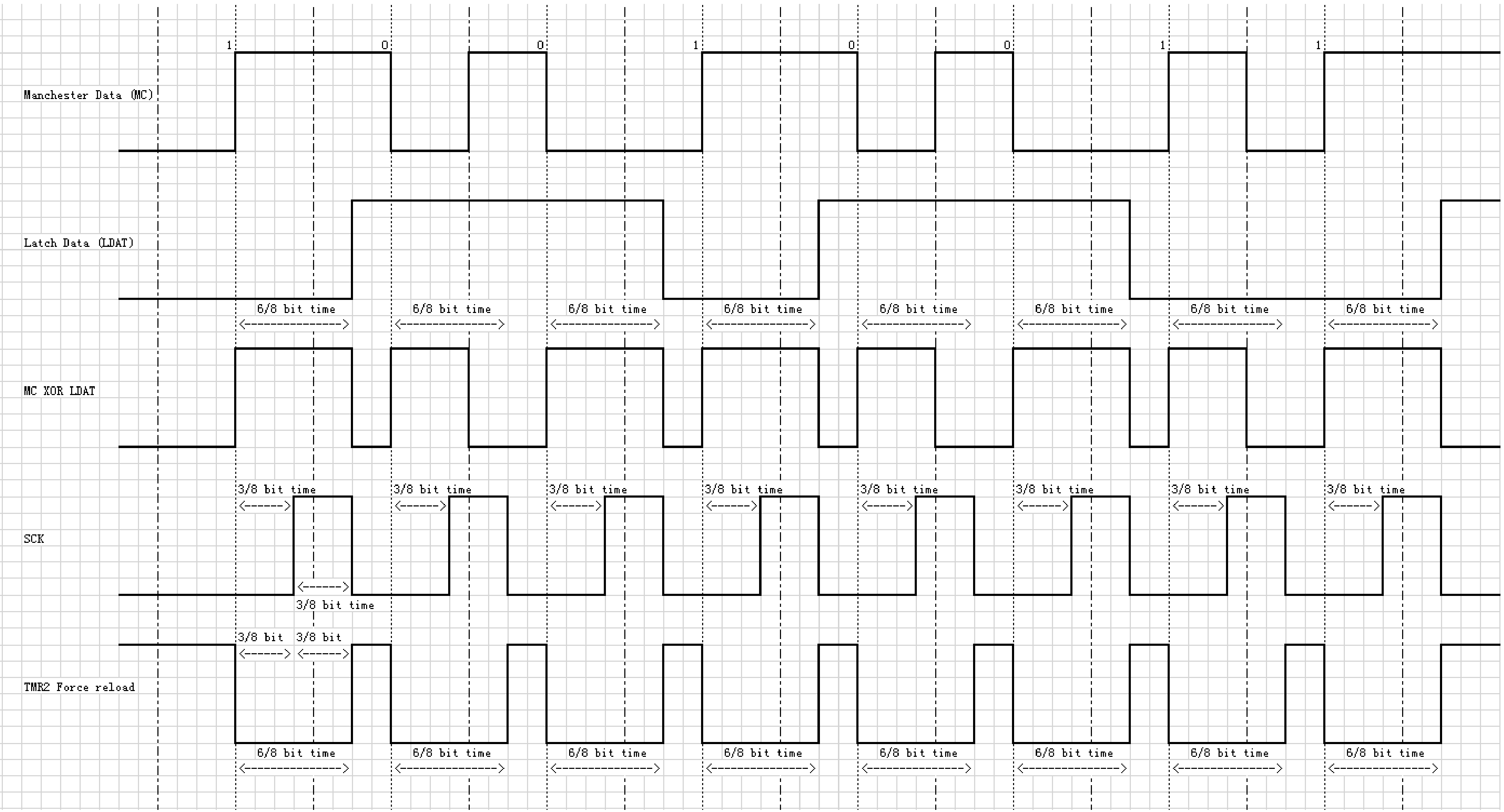
- The MC XOR LDAT rising edge start timer. Timer stops after 6/8 bit time.
- From the observation, the Boolean Equation is F = A NOR B.
  - The A represents MC XOR LDAT, B represents SCK.
  - The F represents TMR2 Force reload

Manchester Decoder - Implementation



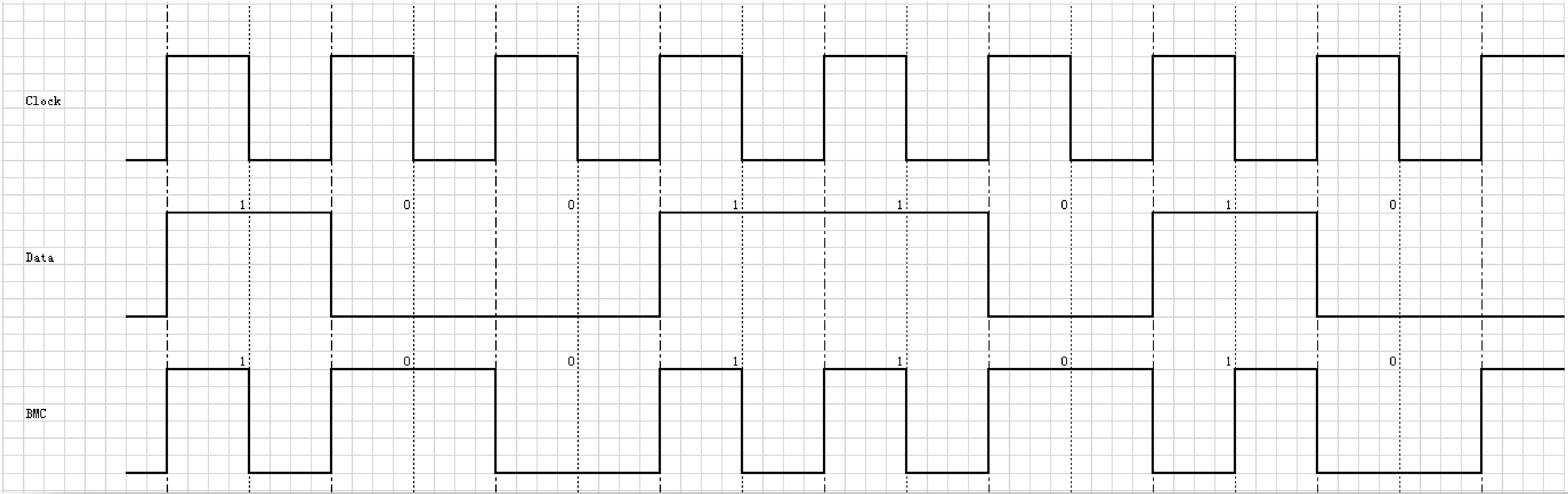

```
CLU0MX = CLU0MX_MXA__CLU0A1 | CLU0MX_MXB__CLU0B2;
CLU0CF = CLU0CF_OUTSEL__LUT | CLU0CF_OEN__ENABLE;
CLU0FN = LUT_NOR(SI_LUT_A, LUT_XOR(SI_LUT_B, SI_LUT_C));

CLU1MX = CLU1MX_MXA__CLU1A1 | CLU1MX_MXB__CLU1B1;
CLU1CF = CLU1CF_OEN__ENABLE | CLU1CF_CLKSEL__ALTCLK;
CLU1FN = LUT_NOT(SI_LUT_A);

CLU2MX = CLU2MX_MXA__CLU2A0 | CLU2MX_MXB__CLU2B8;
CLU2CF = CLU2CF_CLKSEL__CARRY_IN | CLU2CF_OEN__ENABLE;
CLU2FN = SI_LUT_B;

CLU3MX = 0x00;
CLU3CF = CLU3CF_OUTSEL__LUT | CLU3CF_OEN__ENABLE;
CLU3FN = SI_LUT_C;

CLEN0   = 0x0F; // enable CLU0, CLU1, CLU2, CLU3
```

Manchester Decoder - Waveform



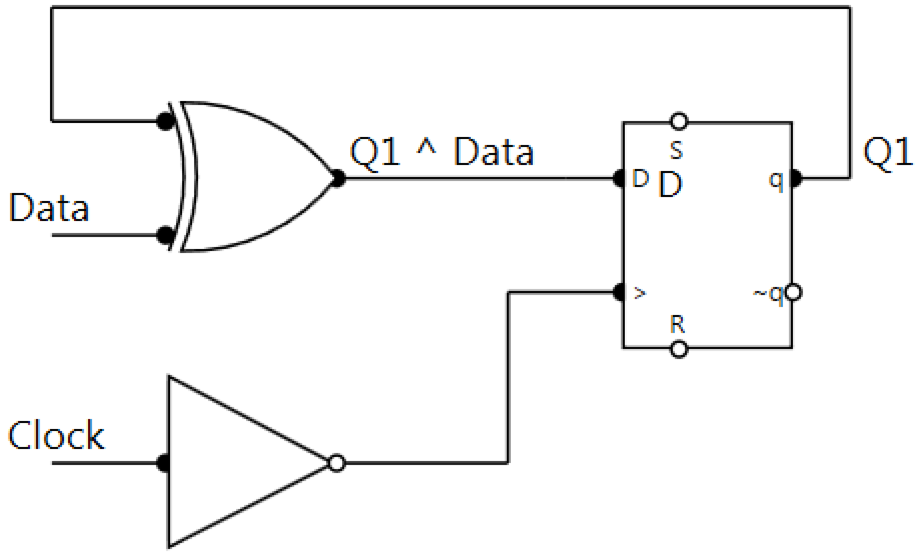
Biphase Mark Code(BMC)

- Biphase Mark Code(BMC) is a line code ,conveys the data and clock information.
- Using the presence or absence of transitions to indicate logical value.
- BMC transitions on every positive edge of the clock signal
- BMC transitions on negative edge of the clock signal when the data is a 1.

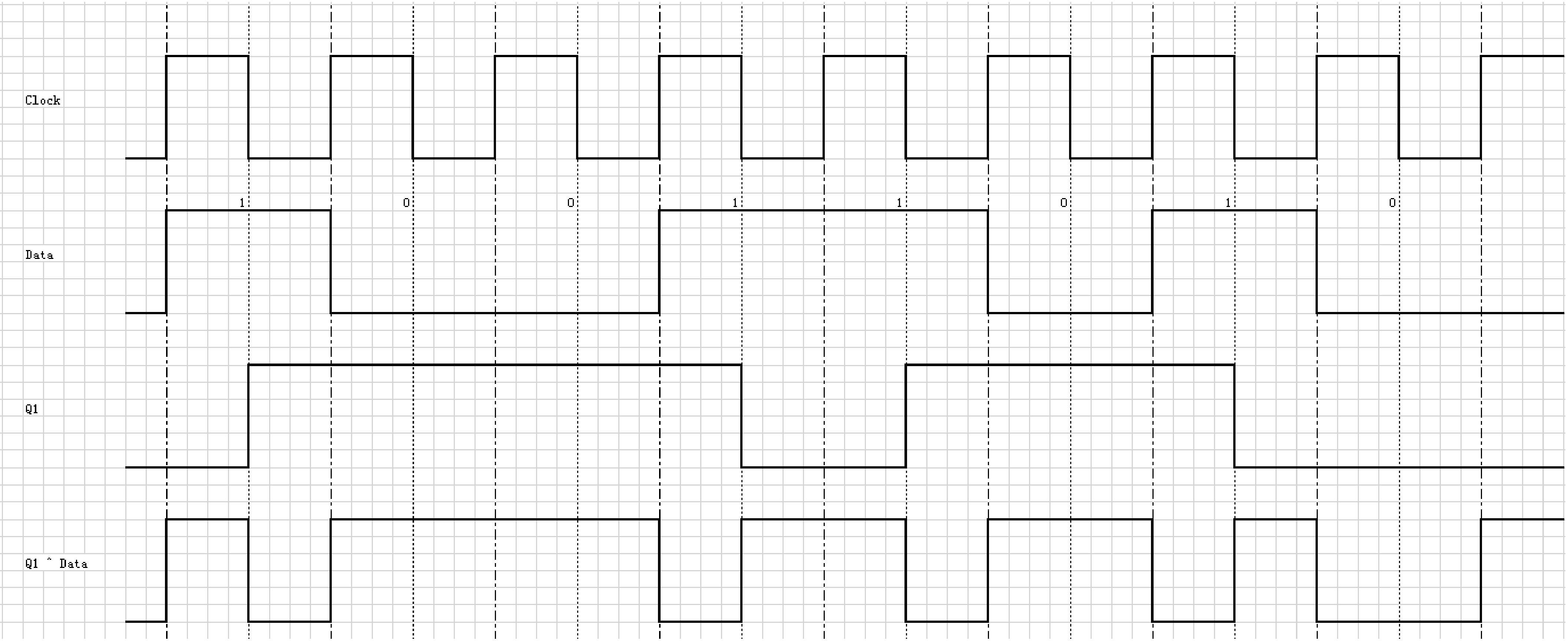


BMC Encoder – Transitions on Data 1

- The Data XOR Q1 as input of D flip-flop.
  - When the Data is 1:
    - At first half bit time, the Q1 XOR 1 = Not Q1
    - At failing edge of clock, the Q1 captures Not Q1 and transition happens.
  - When the Data is 0, the Q1 keeps unchanged, no transition happens.



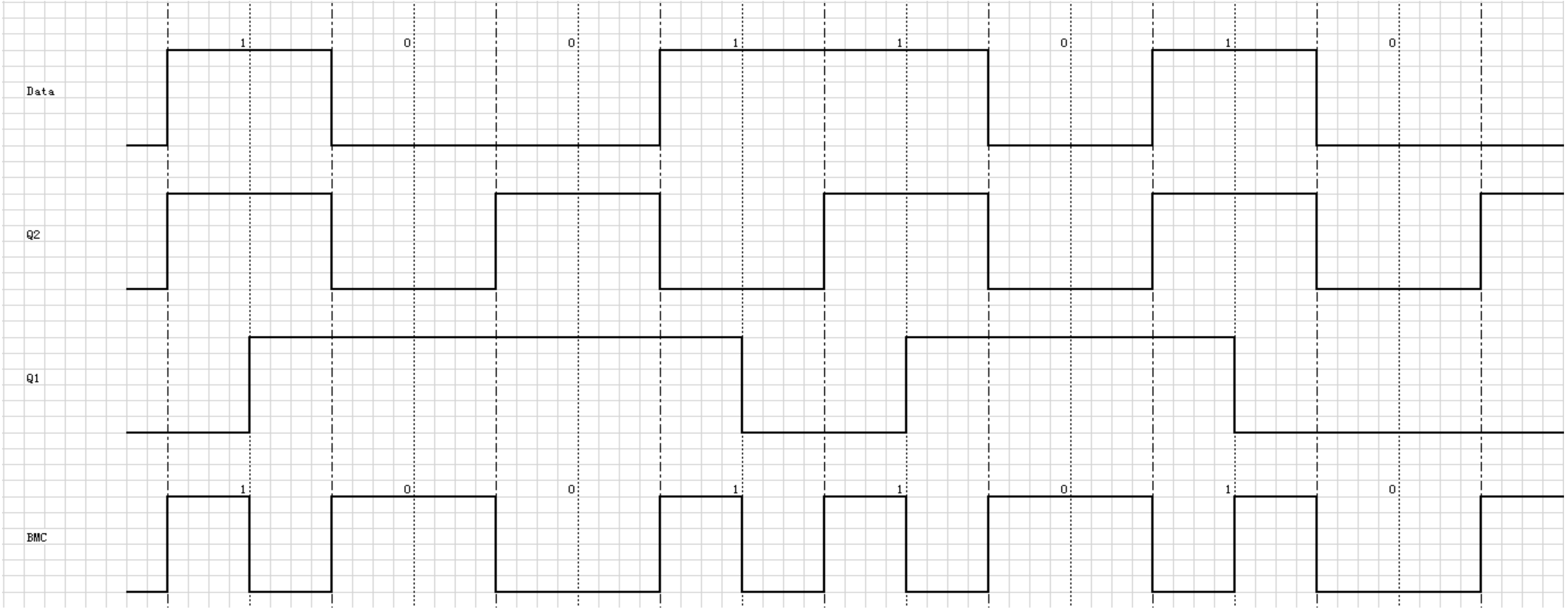
BMC Encoder – Waveform of Transitions on Data 1



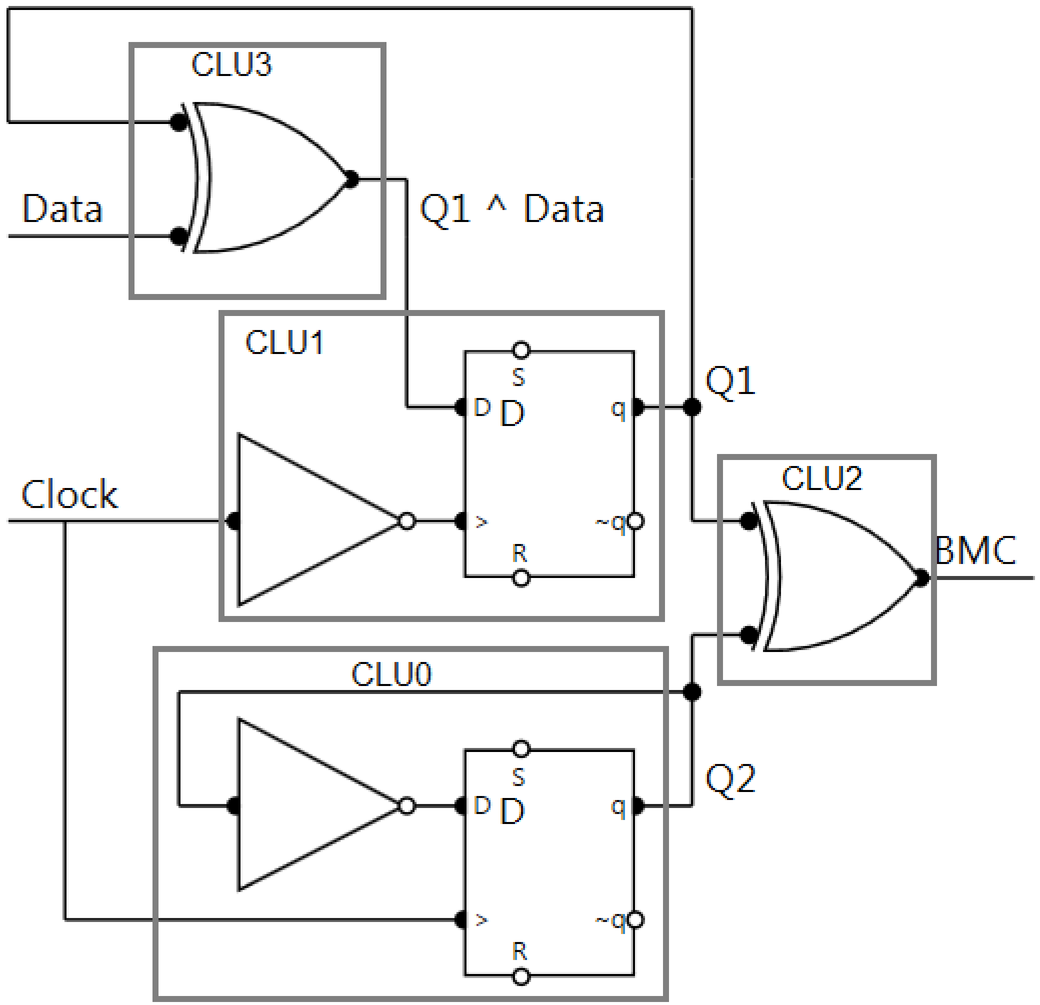
- The D flip-flop captures Q1 ^ Data at middle of bit time.
- The D flip-flop output value keeps unchanged at the beginning of each bit.

BMC Encoder – Transitions at Beginning of Every Bit

- Using D flip-flop to generate Q2 which is the clock divided by 2.
- Using Q2 XOR Q1 to generate BMC data.
  - When Q2 is 1, the XOR results transition.
  - When Q2 is 0, the XOR changes nothing.



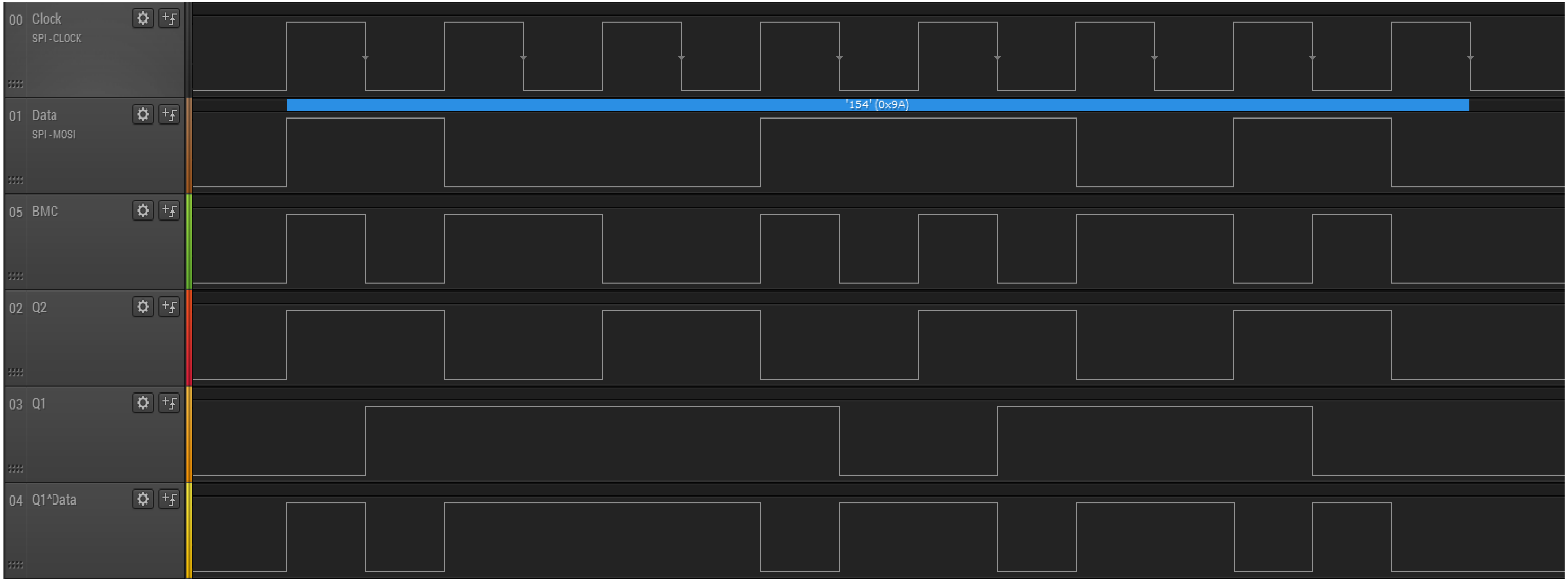
BMC Encoder - Implementation




```
CLU0MX = 0xA0; // MXA as P0.4, MXB as CLU0OUT(P0.2)
CLU0FN = LUT_NOT(SI_LUT_B);
CLU0CF = CLU0CF_OEN__ENABLE | CLU0CF_CLKSEL__MXA_INPUT;

CLU1MX = 0x83; // MXA as P0.4, MXB as CLU3OUT(P2.5)
CLU1FN = SI_LUT_B; // MXB buffer
CLU1CF = CLU1CF_OEN__ENABLE | CLU1CF_CLKSEL__MXA_INPUT |
           CLU1CF_CLKINV__INVERT;

CLU2MX = 0x01; // MXA as CLU0OUT, MXB as CLU1 output
CLU2FN = LUT_XOR(SI_LUT_A, SI_LUT_B);
CLU2CF = CLU2CF_OUTSEL__LUT | CLU2CF_OEN__ENABLE;

CLU3MX = 0xA1; // MXA as P0.6, MXB as CLU1OUTPUT(P1.0)
CLU3FN = LUT_XOR(SI_LUT_A, SI_LUT_B);
CLU3CF = CLU3CF_OUTSEL__LUT | CLU3CF_OEN__ENABLE;

CLEN0   = 0x0F; // enable CLU0, CLU1, CLU2, CLU3
```

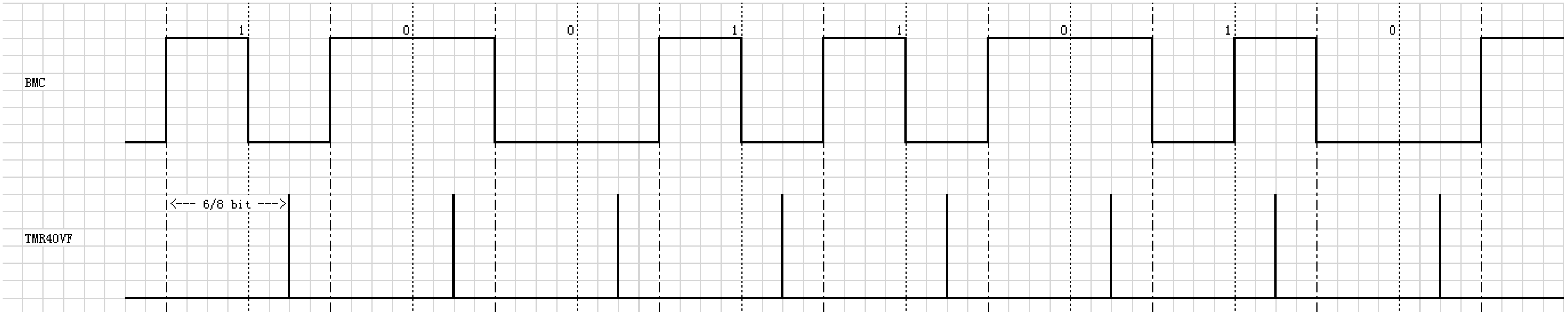
BMC Encoder - Waveform

- All related signals are captured as follows:



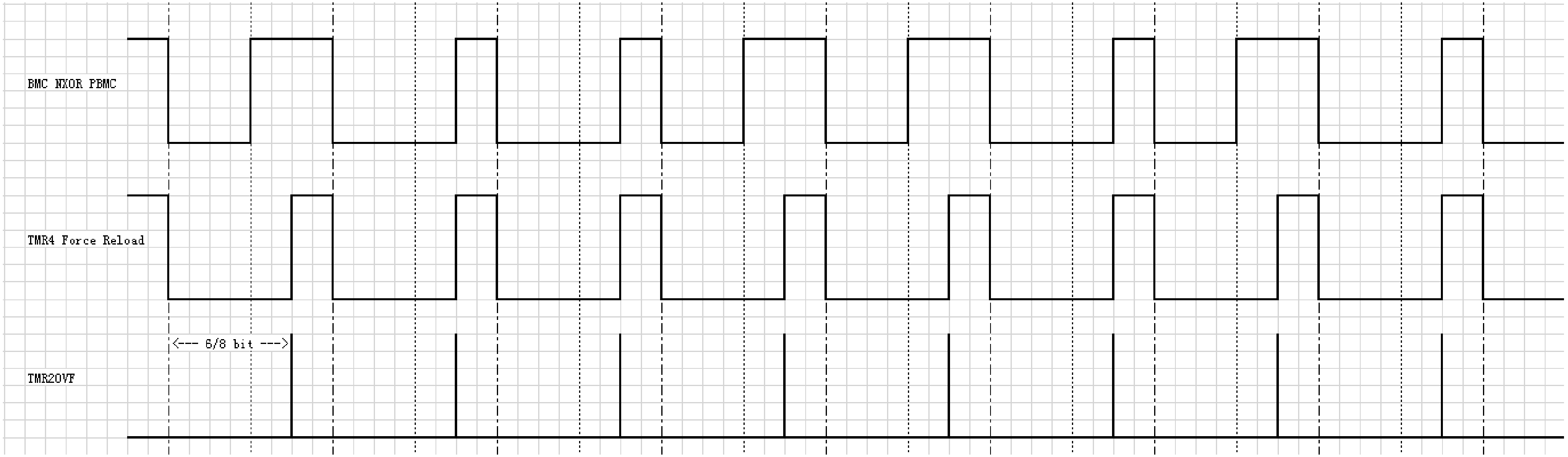
BMC Decoder - Method

- The BMC transition at beginning of each bit or middle of bit when data = 1.
- From observation, we capture the data at 6/8 bit time.
  - When S1 = S2, that means second bit is “1”
  - When S1 != S2, that means second bit is “0”
- Base on above analysis
  - Using XNOR two sample points value to generate Data.
  - Using a Timer to generate SCK which rising edge at beginning of each bit.



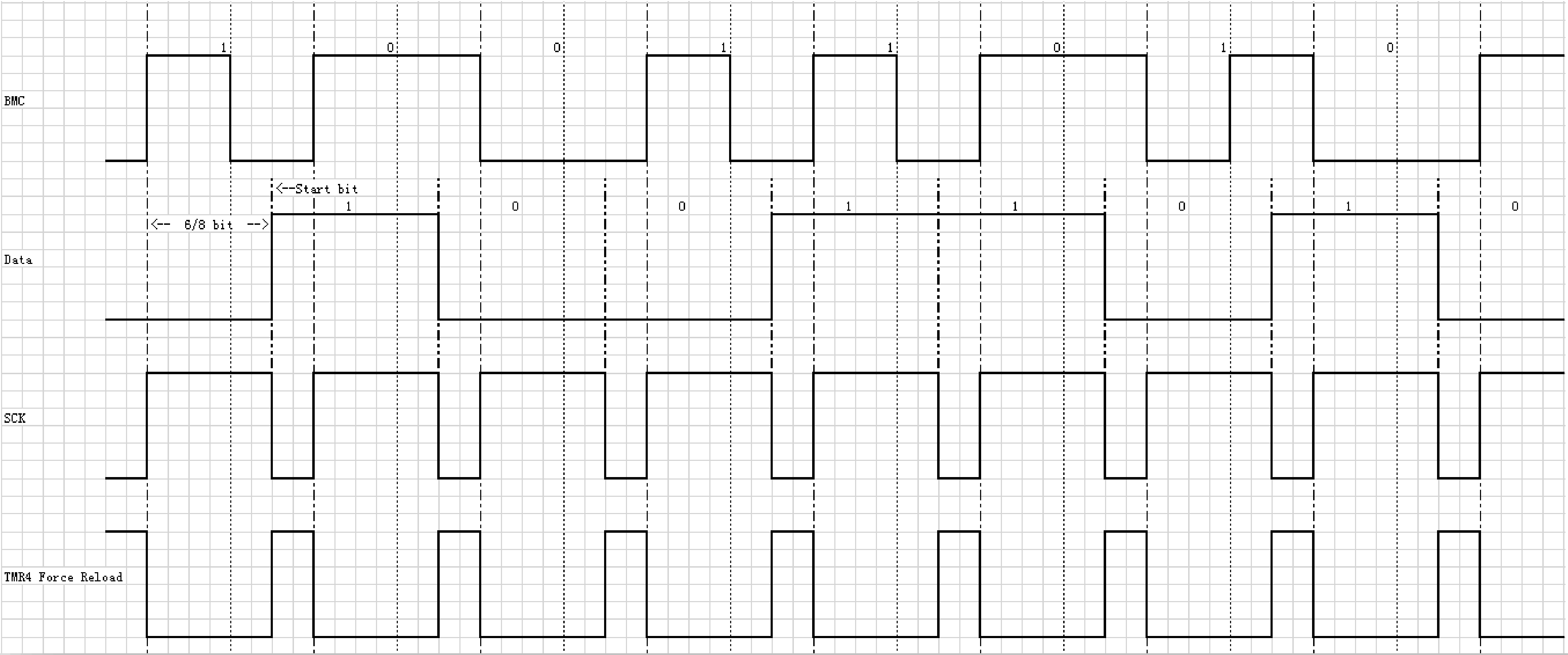
BMC Decoder – Timer Control

- The Timers are used to capture the data and generate clock.
  - Timer starts from beginning of each bit.
  - Timer stops at 6/8 bit time.
- Using Timer OVF and BMC XNOR prior BMC at 6/8 bit time to control Timer.
  - The Timer OVF is low at beginning of the bit. It can be use to start timer.
  - The XNOR result start from 6/8 bit must be "1". It can be use to stop timer.



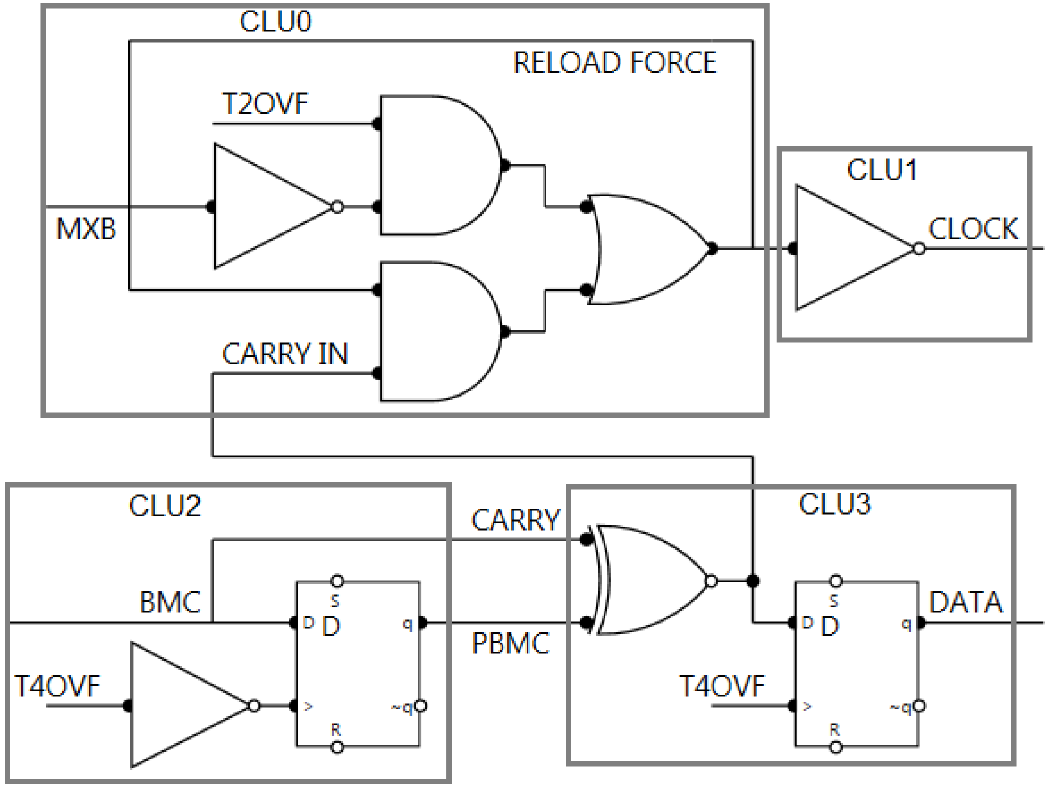
BMC Decoder – Data and Clock Generation

- BMC data XNOR prior BMC value at 6/8 bit time to generate Data.
- The Timer force reload signal inversed as clock.

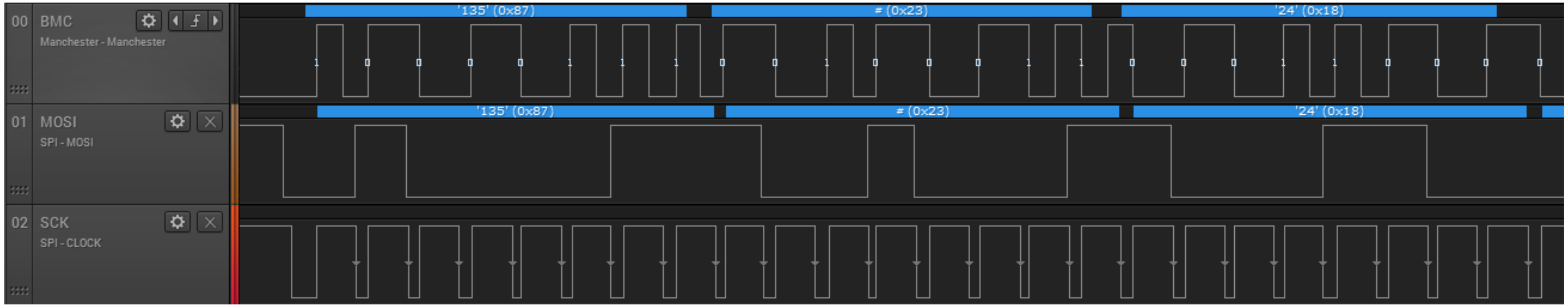


BMC Decoder - Implementation

```
CLU0MX = 0x40; // MXA as T2OVF, MXB as CLU0 output
CLU0CF = CLU0CF_OUTSEL__LUT;
CLU0FN = LUT_OR(LUT_AND(SI_LUT_B, SI_LUT_C),
            LUT_AND(LUT_NOT(SI_LUT_B), SI_LUT_A));

CLU1MX = 0x00; // MXA as CLU0 output
CLU1CF = CLU1CF_OEN__ENABLE | CLU1CF_OUTSEL__LUT;
CLU1FN = LUT_NOT(SI_LUT_A);

CLU2MX = 0x48; // MXA as T4OVF, MXB as P0.2
CLU2CF = CLU2CF_CLKINV__INVERT | CLU2CF_CLKSEL__MXA_INPUT;
CLU2FN = SI_LUT_B;

CLU3MX = 0x22; // MXA as CLU2 output;
CLU3CF = CLU3CF_OEN__ENABLE | CLU3CF_CLKSEL__ALTCLK;
CLU3FN = LUT_XNOR(SI_LUT_A, SI_LUT_C);

CLEN0   = 0x0F; // enable CLU0, CLU1, CLU2, CLU3
```



BMC Decoder - Waveform



- The Data start from 6/8 bit time of BMC data.
- The Data is valid on second edge of SCK period. (PHA = 1)

Software examples

- Software examples are available in Simplicity Studio.
  - SR and D Latches
  - Button Debounce
  - Manchester Encoder
  - Manchester Decoder
  - Biphase Mark Encoder
  - Biphase Mark Decoder

Thank you!