DESIGN PROJECT REPORT

NAME : NAWARATHNA K.G.I.S.
REG. NO : E/17/219
DEPARTMENT : COMPUTER ENGINEERING
DATE:13/06/2020

INTRODUCTION

In digital communication, errors are introduced during the transmission of data from the transmitter to receiver due to noise or environmental interface. These errors can become a serious problem when considering the accuracy of the system. Therefore error detection and correction is a viral part in data transmission.

There are some methods to detect errors in a data transmission such as two dimenssional parity check, checksum and cyclic redundancy check (CRC). However ~~thesen~~ these methods only detect errors. They can not be used to correct detected errors. Therefore some other methods must be introduced to correct such errors. One of the most common method for achiving this goal is the Hamming Code.

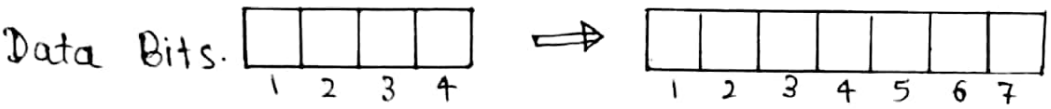
In the hamming code k number of additional bits are generated for n number of data bits. Therefore (n+k) number of bits are used to generate the Hamming code. These additional bits are called parity bits and they are used to check whether the received data is correct or not. If data bits are found correct, then they can be used to whaterver the process without undergoing any correction process. Otherwise data bits are corrected.

The main pupose of this project is to design the circuit for implementing the hamming code, using various hardware components in digital electronics. First transmitter and receiver will be degigned for 4 data bits and then this circuit will be developed to be working with any number of data bits.

IMPLEMENTATION OF THE DATA TRANSMITTER

BASIC FUNCTIONALITY

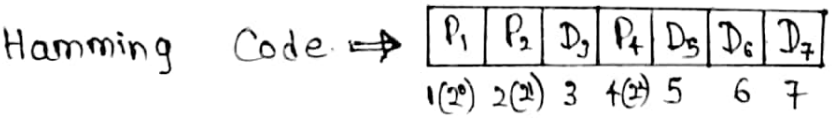
This implementation is done for 4 data bits. Additional 3 bits (parity bits) are generated for error calculations. Therefore Hammig Code contains 7 bits.



Hamming Code.

- Generating Parity Bits :

Additional number of bits called parity bits are generated for error calculations. They are placed in the 2^n (n=0, 1, 2) th position of the hamming code. Rest of the positions are consisted with data bits.



P = Parity Bits.
D = Data Bits.

- How to generate parity bits ? :

- P_1 checks the positions of,

  { 011 , 101 , 111 }

  - They are the positions of 3, 5 and 7.

- P_2 checks the positions of,

  {011, 110 , 111 }

  - They are the positions of 3, 6 and 7.

- P_3 checks the positions of,

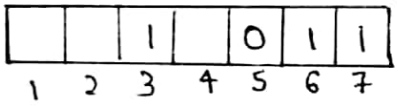
  { 101, 110, 111 }

  - They are the positions of 5, 6 and 7.

P_1 = XOR of (3, 5, 7).
P_2 = XOR of (3, 6, 7).
P_3 = XOR of (5, 6, 7.).

- Example :

1011 are considered as the data bits.



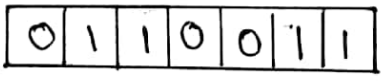
Generating parity bits

$P_1 = (1 \oplus 0 \oplus 1) = 0$,

$P_2 = (1 \oplus 1 \oplus 1) = 1$,

$P_4 = (0 \oplus 1 \oplus 1) = 0$,

Hamming Code

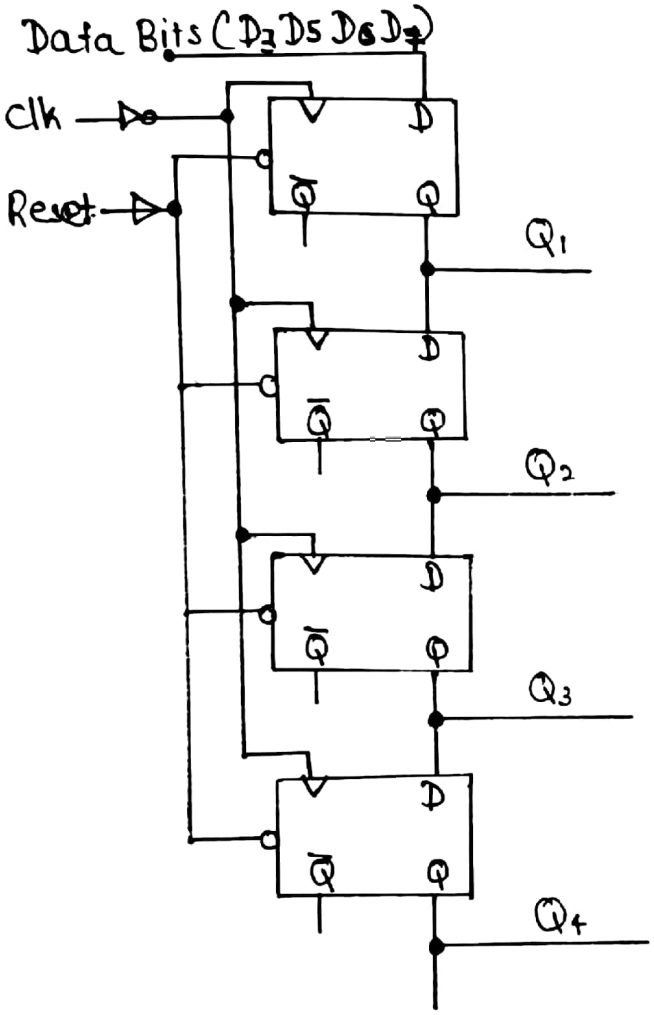


This generated hamming code can be transmitted by using a transfering media to the receiver.

IMPLEMENTATION

① Inputs of data bits.

first of all data bits are loaded into a 4 bit register as following.



All of these flip flops are negative edge triggered flip flops. An inverter is connected to the clock. This is done in order to minimize the leakage current from the clock generator.

First of all values stored in flip flops are reset to 0. Then data bits are loaded in to the register. The sequence is $D_3 D_5 D_6 D_{47}$ This is a 4 bit serial in parallel out D flip flop register. Therefore After 4 clock Pulses we can get data bits D_1, D_2, P_3 and D_4 to Q_1, Q_2, Q_3 and Q_4. This register is sensitive to the input when clock pulse is a negative edge.

Clk.	Input	Q_1	Q_2	Q_3	Q_4
0	D_7	0	0	0	0
1	D_6	D_7	0	0	0
2	D_5	D_6	D_7	0	0
3	D_3	D_5	D_6	D_7	0
4	0	D_3	D_5	D_6	D_7

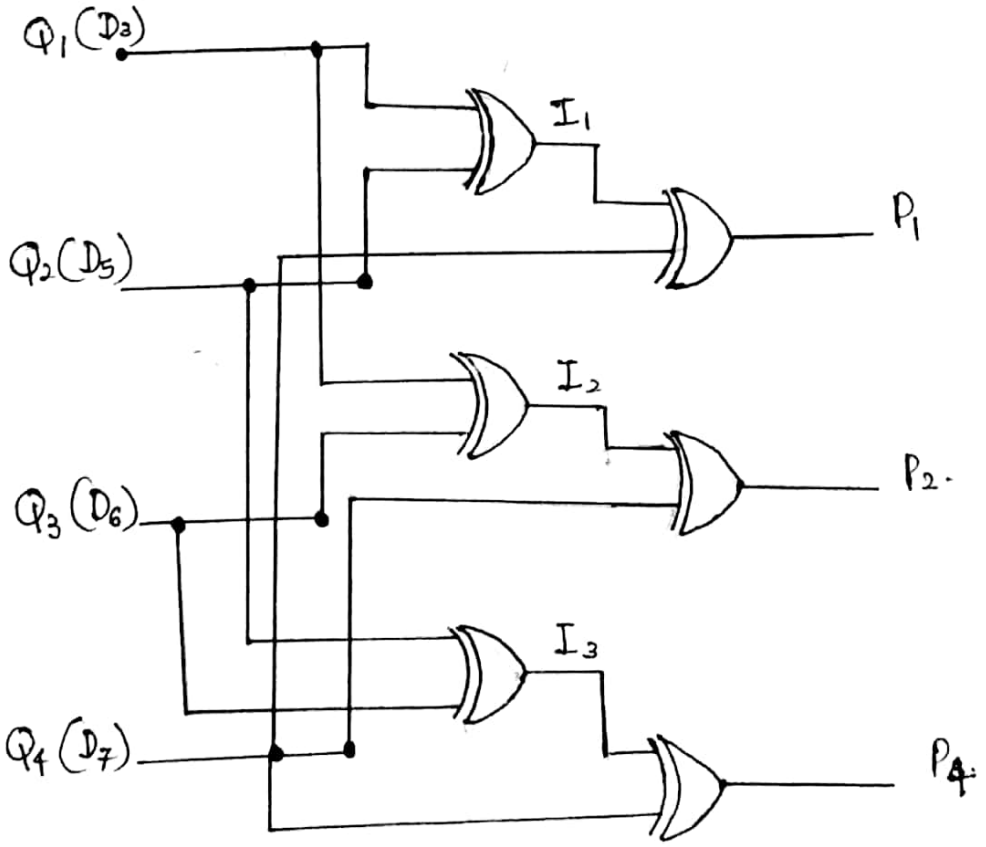
Since the sequence is $D_3 D_5 D_6 D_7$, after 4 clock pulses,

$$Q_1 = D_3$$
$$Q_2 = D_5$$
$$Q_3 = D_6$$
$$Q_4 = D_7$$

After that we give an enable to the clock circuit and keep this values stable.

② Parity bit generator.

After loading data bits to the register, Outputs of the parallel bit register are now connected to the parity bit generator. Parity bit generator is consisted with 6 XOR 2 inputs gates. The circuit is shown here.



$$I_1 = D_3 \oplus D_5$$
$$P_1 = I_1 \oplus D_7 = D_3 \oplus D_5 \oplus D_7$$

$$I_2 = D_3 \oplus D_6$$
$$P_2 = I_2 \oplus D_7 = D_3 \oplus D_6 \oplus D_7$$

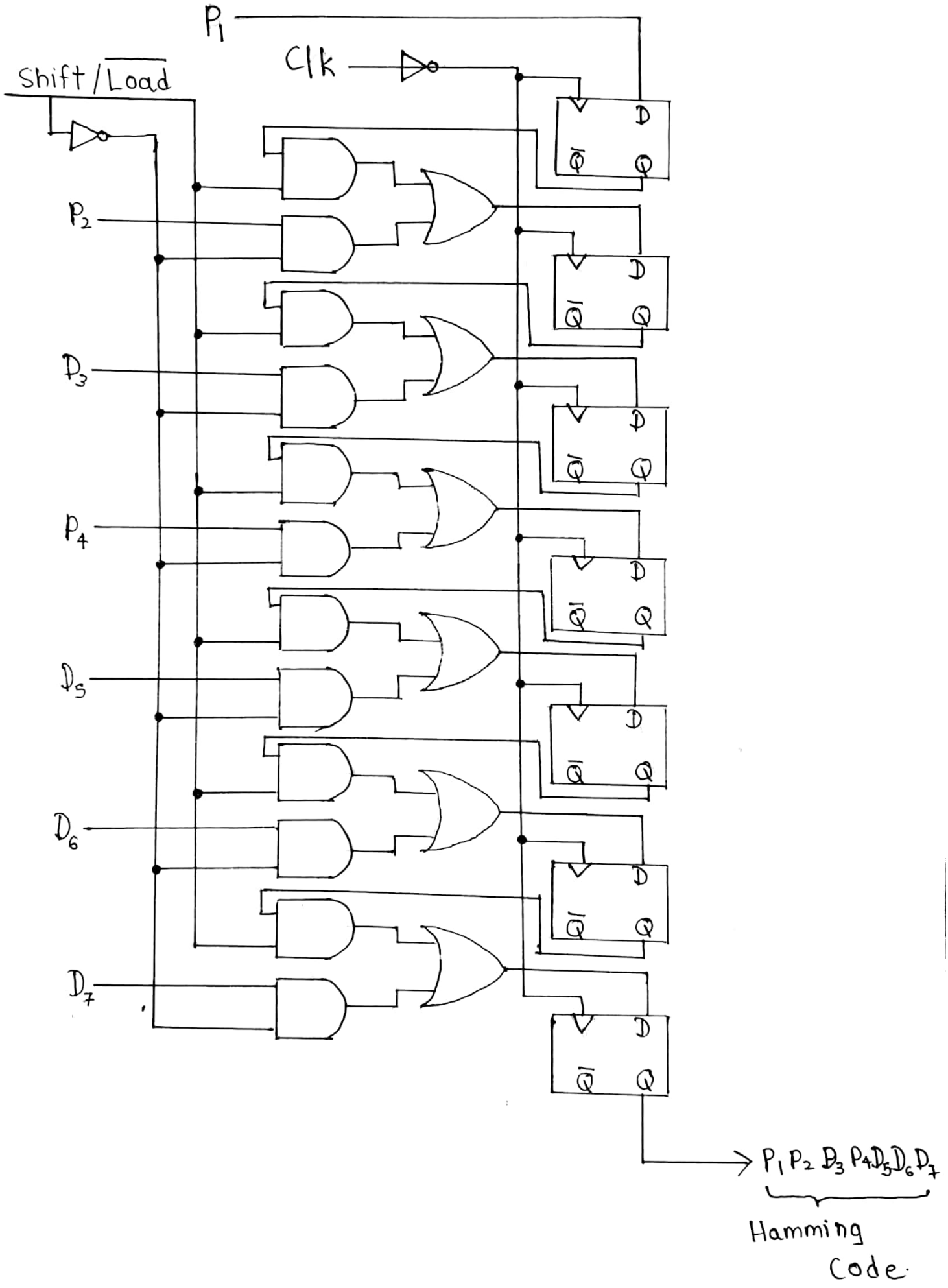
$$I_3 = D_5 \oplus D_6$$
$$P_4 = I_3 \oplus D_7 = D_5 \oplus D_6 \oplus D_7$$

③. Output of the transmitter.

Since parity bits are generated now, all the bits that are parity bits and data bits now needed to be combined in order to make the hamming code. This is done by a parallel to serial shift register.

This register is consisted with 7 D flip flops. All of these flip flops are sensitive to the falling edge of the clock pulse since they are falling edge triggered flip flops.

This register has two modes. Those are loading mode and shift mode. In loading mode we are whatever the values of the inputs in flip flops. In shift mode the movement of the bits that are stored in the flip flop occurs. Therefore in shift mode after 7 clock pulses all of the data bits are output to the transfering media. The circuit diagram of the register is shown below.



After parity bits has generated, all of the 7 bits including parity bits are now loaded into the 7 D flip flops. To do this register must be configured to Load Mode.

① . Load Mode.

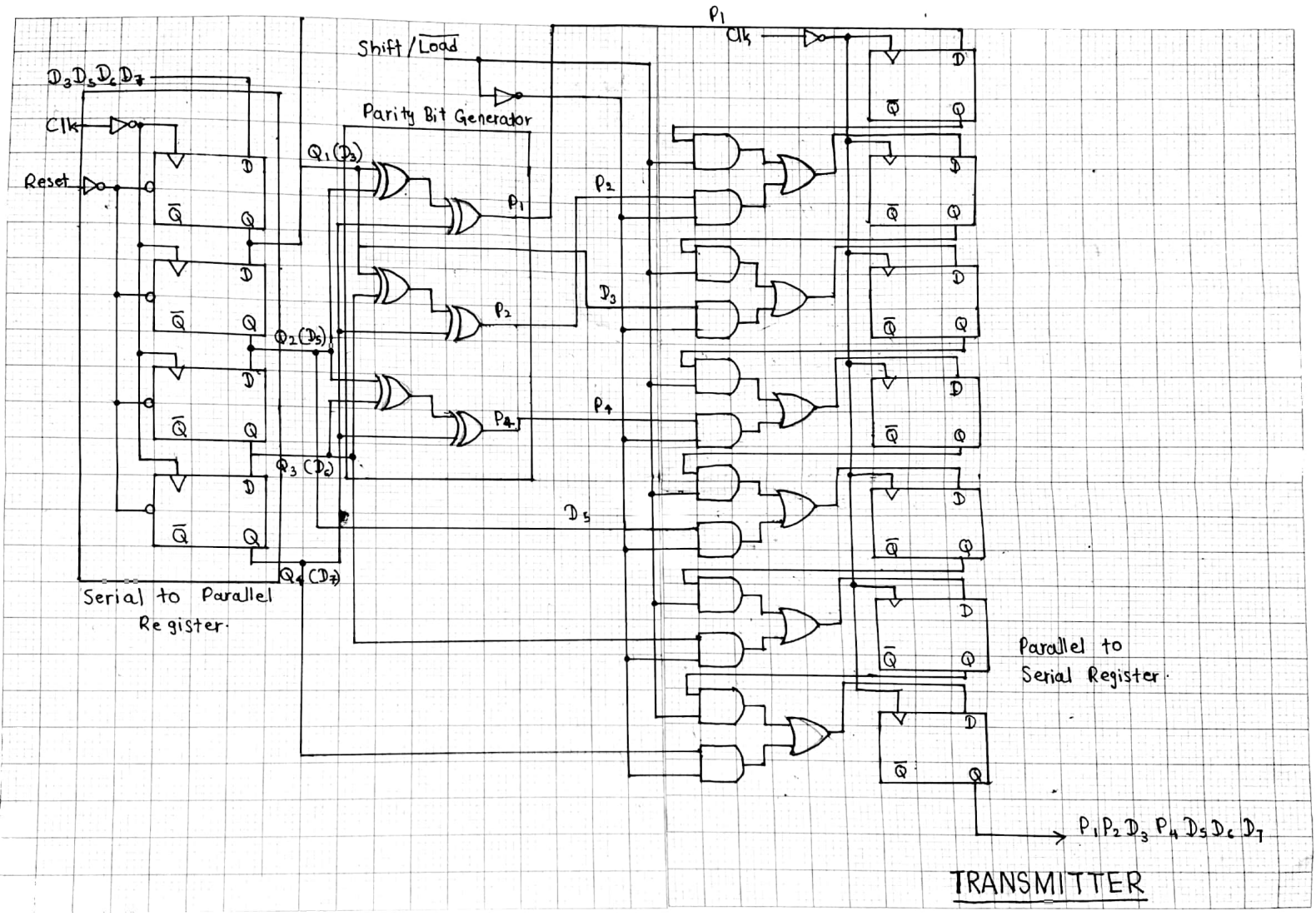
- By making the shift ($\overline{Load}$) input low, register can be configured to the loading mode. In loading mode, Combinational circuits are sensitive to the parity bits and data bits inputs. Therefore in loading mode after the 1st active clock pulse all the values of $P_1, P_2, D_3, P_4, D_5, D_6, D_7$ are stored in the flip flops.

After this, since bits are stored in flip flops now it is time to take the hamming code as the output of the register. In order to do this register must be configured to the shift mode.

② . Shift Mode.

- By making the shift ($\overline{Load}$) input high, register can be configured to the shif mode. In shift mode combinational circuits are sensitive to the output of the flip flops. Therefore in shift mode output is generated as following.

Active clk	Input.	Output.	
0	D_7	x	
1	D_6	D_7	
2	D_5	D_6	D_7
3	P_4	D_5	$D_6\ D_7$
4	D_3	P_4	$D_5\ D_6\ D_7$
5	P_2	D_3	$P_4\ D_5\ D_6\ D_7$
6	P_1	P_2	$D_3\ P_4\ D_5\ D_6\ D_7$.
7	x	P_1	$P_2\ D_3\ P_4\ D_5\ D_6\ D_7$



PROOF OF THE TRANSMITTER

First 4 data bits $D_3 D_5 D_6 D_7$ are loaded into the Serial to Parallel 4 bit D flip flop register. Then after 4 negative edge clock pulses $D_3 D_5 D_6 D_7$ can be taken from the register as the output of Q_1, Q_2, Q_3 and Q_4. The we make the clock enable so that outputs doesn't change.

$$Q_1 = D_3$$
$$Q_2 = D_5$$
$$Q_3 = D_6$$
$$Q_4 = D_7$$

Then we input these bits to the parity bit generator. From the output of the parity bit generator, we can take,

$P_1 = I_1 \oplus D_7 = D_3 \oplus D_5 \oplus D_7$ = XOR of bit (3, 5, 7).
$P_2 = I_2 \oplus D_7 = D_3 \oplus D_6 \oplus D_7$ = XOR of bit (3, 6, 7).
$P_3 = I_3 + D_7 = D_5 \oplus D_6 \oplus D_7$ = XOR of bits (5, 6, 7).

After this we ~~make the~~ load all of D_3, D_5, D_6, D_7 and P_1, P_2, P_3 into the parallel input serial output 7 bit D flip flop register.

First we make the shif input low and load all of the data bits and parity bits into the register. Then we shift input high can configure the register to shift mode. After 7 negative edges we can get the hamming code as the output.

Therefore this is a good implementation of the transmitter of the hamming code.

Therefore after 7 clock pulses, intended hamming code is generated. Now this code should be sent to the receiver using a transmitting media.

IMPLEMENTATION OF THE RECEIVER

BASIC FUNCTIONALITY

When 7 bits of the hamming code are read from the memory they are checked again for errors. For checking purposes, check bits are gene rated over the same combination of bits ncluding the parity bit.

- How to generate checker bits? :
  - First checker bit (C_1) checks the positions of,

    {001, 011, 101, 111}
    - They are the positions of 1, 3, 5, 7.
  - Second checker bit (C_2) checks the positions of,

    {010, 011, 110, 111}
    - They are the positions of 2, 3, 6, 7.
  - Third checke bit (C_3) checks the positions of,

    {100, 101, 110, 111}
    - They are the positions of 4, 5, 6, 7.

C_1 = XOR of (1, 3, 5, 7)
C_2 = XOR of (2, 3, 6, 7)
C_3 = XOR of (4, 5, 6, 7).

Then ~~folling~~ following number which is called syndrome ~~if~~ is formed by combining $C_3C_2C_1$.

$$C = C_3 C_2 C_1.$$

- If $C = 000$ then that means there is no errors.

- If $C = 011$ then ⇒

$$011 \rightarrow 2^0 \times 1 + 2^1 \times 1 + 2^2 \times 0 = 1 + 2 = 3.$$

- 3rd bit is wrong. It is a data bit. Therefore it should be complemented.

- If $C = 010$ then ⇒

$$010 \rightarrow 2^0 \times 0 + 2^1 \times 1 + 2^2 \times 0 = 2.$$

- 2nd bit is wrong. It is a parity bit. That means data bits are not incorrect.

- Example:

- Previous ~~error~~ ~~is~~ example is considered here as well.

Data ⇒ 1 0 1 1

Hamming code generated by the transmitter. ⇒

0	1	1	0	0	1	1
1	2	3	4	5	6	7

- If there is an error in the 6th bit after transmitting,

Hamming code received by the receiver ⇒

0	1	1	0	0	0	1
1	2	3	4	5	6	7

C_1 = XOR of (1, 3, 5, 7) = $0 \oplus 1 \oplus 0 \oplus 1 = 0$.

C_2 = XOR of (2, 3, 6, 7) = $1 \oplus 1 \oplus 0 \oplus 1 = 1$

C_3 = XOR of (4, 5, 6, 7) = $0 \oplus 0 \oplus 0 \oplus 1 = 1$

$C = C_3 C_2 C_1 = 110$.

- Since $C \neq 000$, there must be an error occured during transmission.

$$C = 110 = 2^0 \times 0 + 2^1 \times 1 + 2^2 \times 1 = 6$$

- 6th bit is not correct. Therefore it should be complemented. After complementing,

Received Code after correction $\Rightarrow$

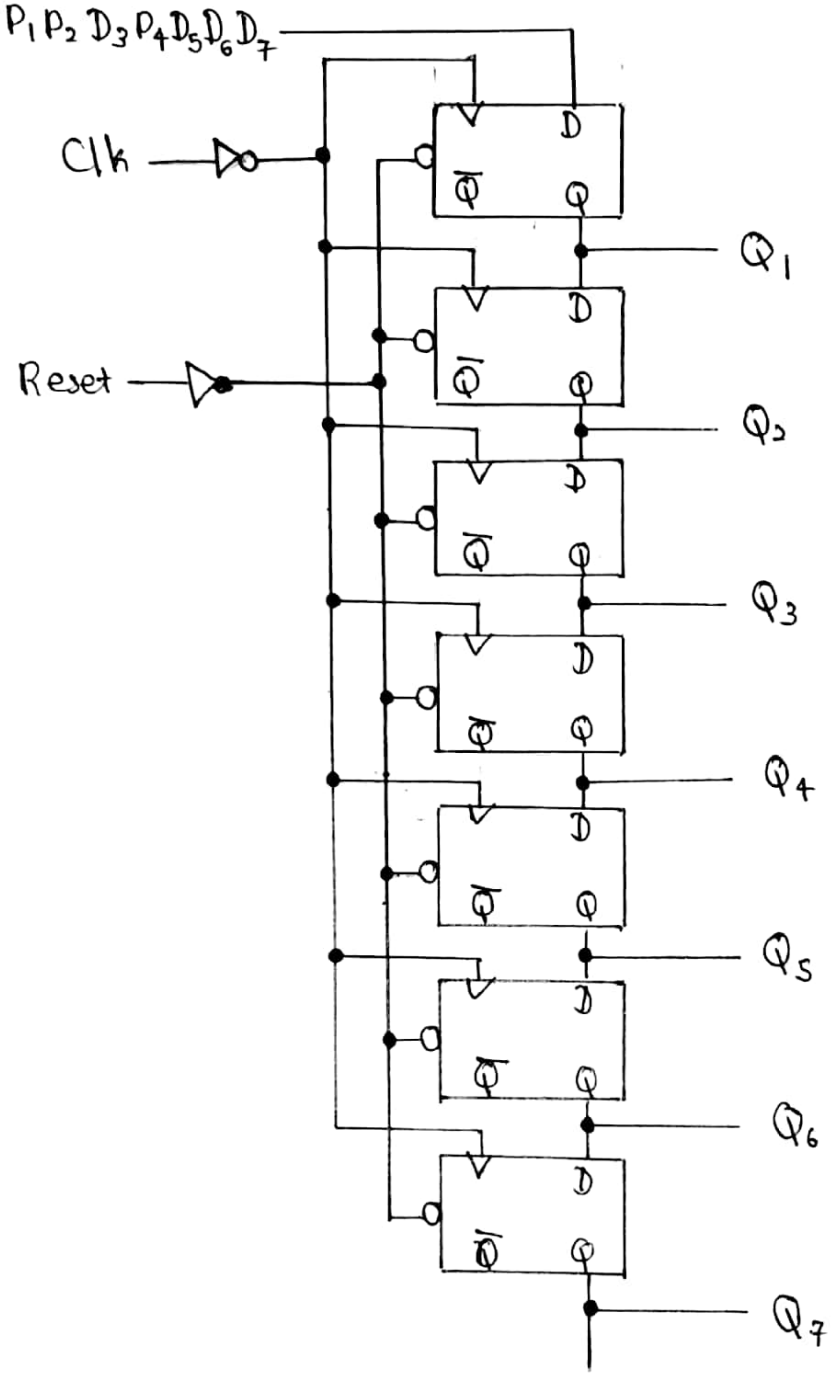
0	1	1	0	0	1	1

- Therefore, Received Hamming Code = Transmitted Hamming code.

- This is how the error calculation is done by the receiving end.

IMPLEMENTATION

① Inputs of the received hamming code.



As the first step of the receiver, received hamming code is loaded into a 7 bit D flip flop serial input parallel output shift register. Since an inverter is connected to the clock, it is a negative edge triggered flip flop. Inverter is directely connected to the clock, rather than connecting an inverter for each individual flip flop. This is done in order to

minimize the leakage current from the clock generator. This is also the case for Reset as well.

Before loading all flip flops are reset to 0. Then bits are loaded into the register. The sequence of bits is $P_1\ P_2\ D_3\ P_4\ D_5\ D_6\ D_7$. After 7 active clock pulses input code is stored in $Q_1, Q_2, Q_3, Q_4, Q_5, Q_6, Q_7$ as following.

Clk.	Input	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7
0	D_7	0	0	0	0	0	0	0
1	D_6	D_7	0	0	0	0	0	0
2	D_5	D_6	D_7	0	0	0	0	0
3	P_4	D_5	D_6	D_7	0	0	0	0
4	D_3	P_4	D_5	D_6	D_7	0	0	0
5	P_2	D_3	P_4	D_5	D_6	D_7	0	0
6	P_1	P_2	D_3	P_4	D_5	D_6	D_7	0
7	X	P_1	P_2	D_3	P_4	D_5	D_6	D_7

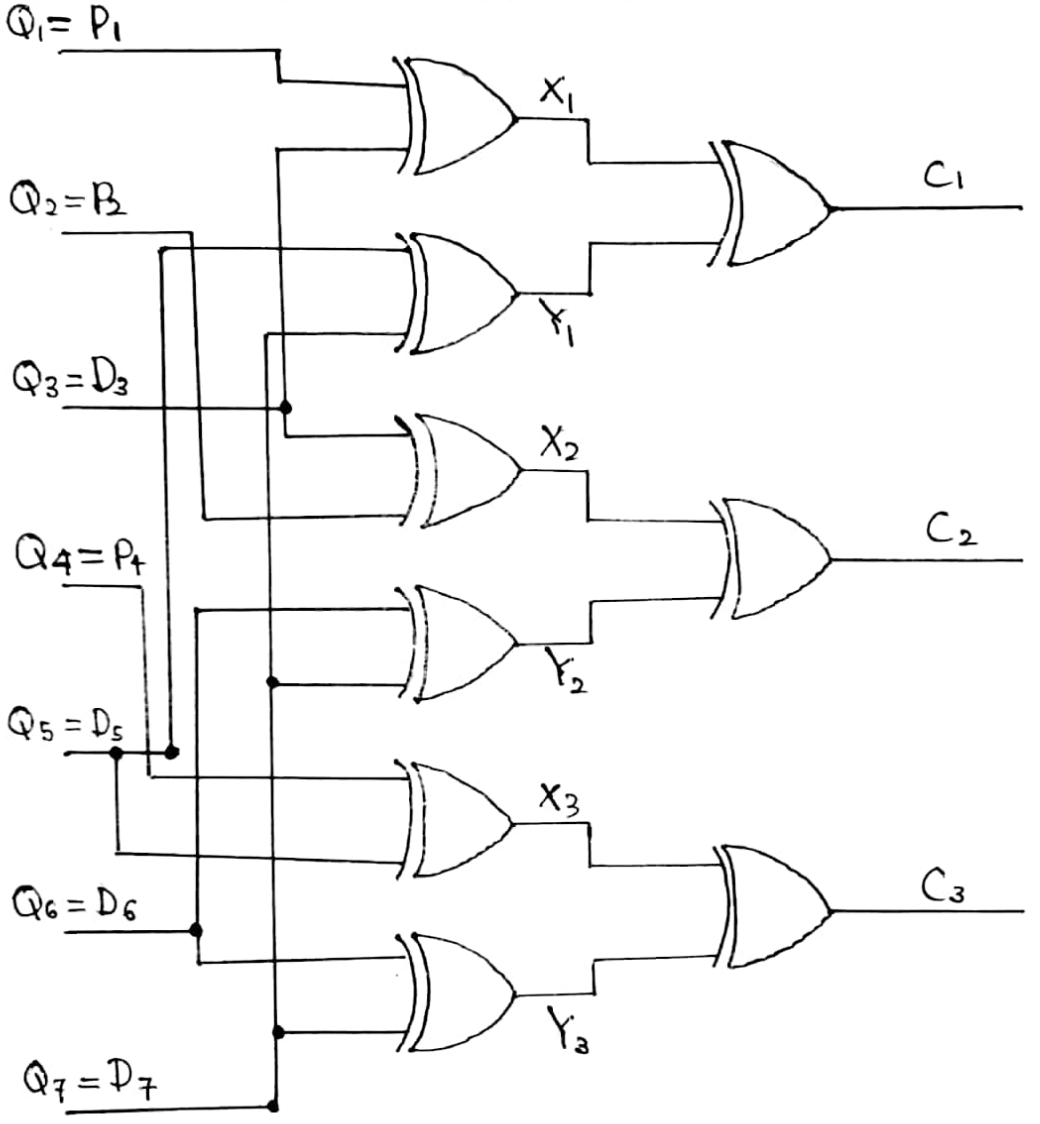
Now,

$$Q_1 = P_1$$
$$Q_2 = P_2$$
$$Q_3 = D_3$$
$$Q_4 = P_4$$
$$Q_5 = D_5$$
$$Q_6 = D_6$$
$$Q_7 = D_7$$

After this we give an enable to the clock pulses inorder to keep the output values unchanged.

(29). Checker bit generator.

After loading data bits and parity bits to the register, outputs of the register is now connected to the checker bit generator. Checker bit generator contains 9 XOR 2 input gates. The circuit is shown here.



$X_1 = P_1 \oplus D_3$, $Y_1 = D_5 \oplus D_7$, $C_1 = X_1 \oplus Y_1$

$C_1 = P_1 \oplus D_3 \oplus D_5 \oplus D_7$ = XOR of (1, 3, 5, 7).

Similarly, $X_2 = D_3 \oplus P_2$, $Y_2 = D_7 \oplus D_6$, $C_2 = X_2 \oplus Y_2$

$C_2 = D_3 \oplus D_2 \oplus D_6 \oplus D_7$ = XOR of (2, 3, 6, 7).

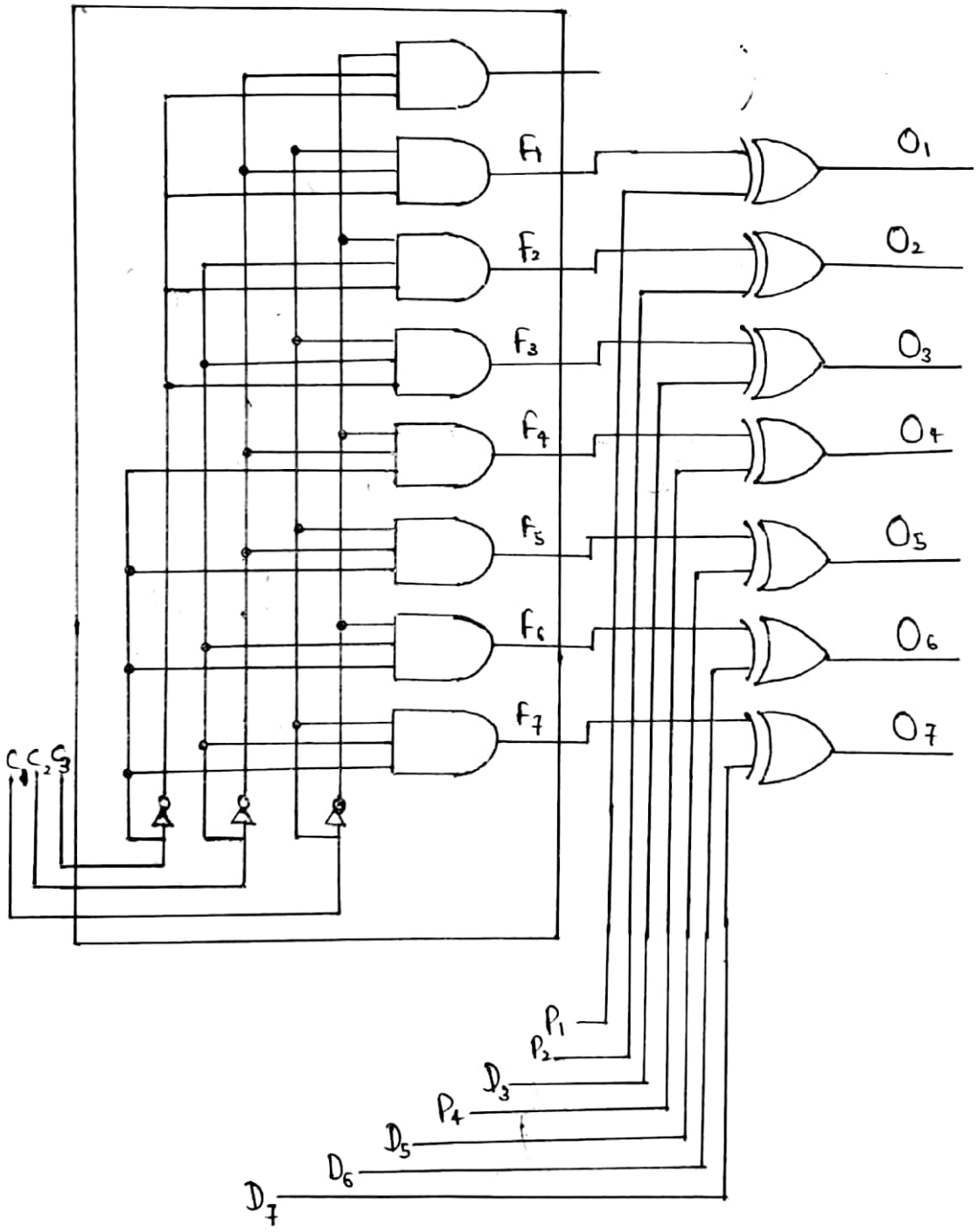
Similarly, $X_3 = P_4 \oplus D_5$, $Y_3 = D_6 \oplus D_7$.

$$C_3 = X_3 \oplus Y_3 = P_4 \oplus D_5 \oplus D_6 \oplus D_7$$

C_3 = XOR of (4, 5, 6, 7).

Therefore checker bites are generated as intended.

③. Error Calculation.



Error calculation part of the circuit contains 2 main parts.

① 3 to 8 decoder.

② Circuit with XOR gates.

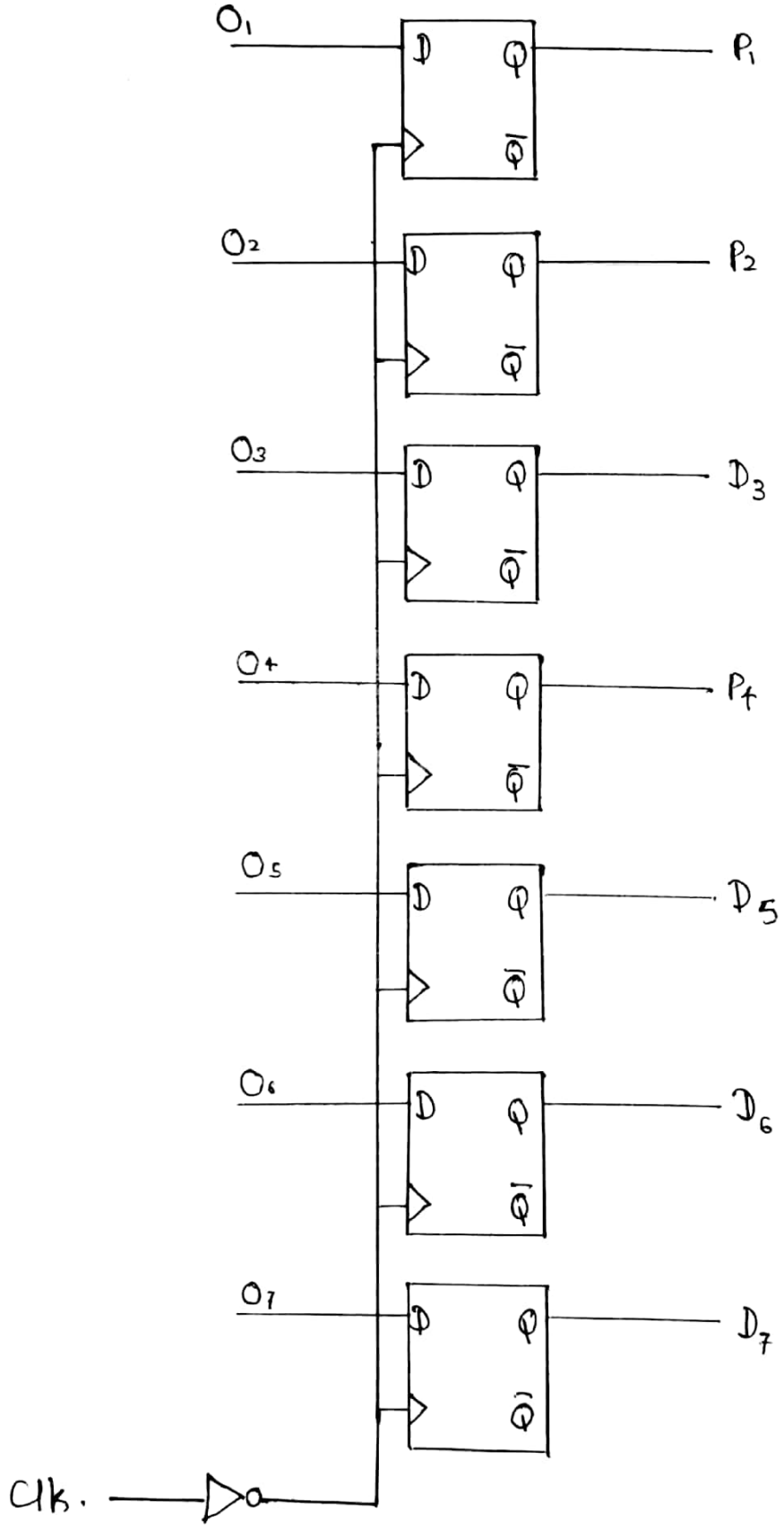
Outputs of the decoder are $F_0, F_1, F_2, F_3, F_4, F_5, F_6, F_7$ Among those F_0 is ignored because it doesn't contribute to the error calculation. All of the other outputs connected to a XOR gate with the corresponding bits as shown in the diagram.

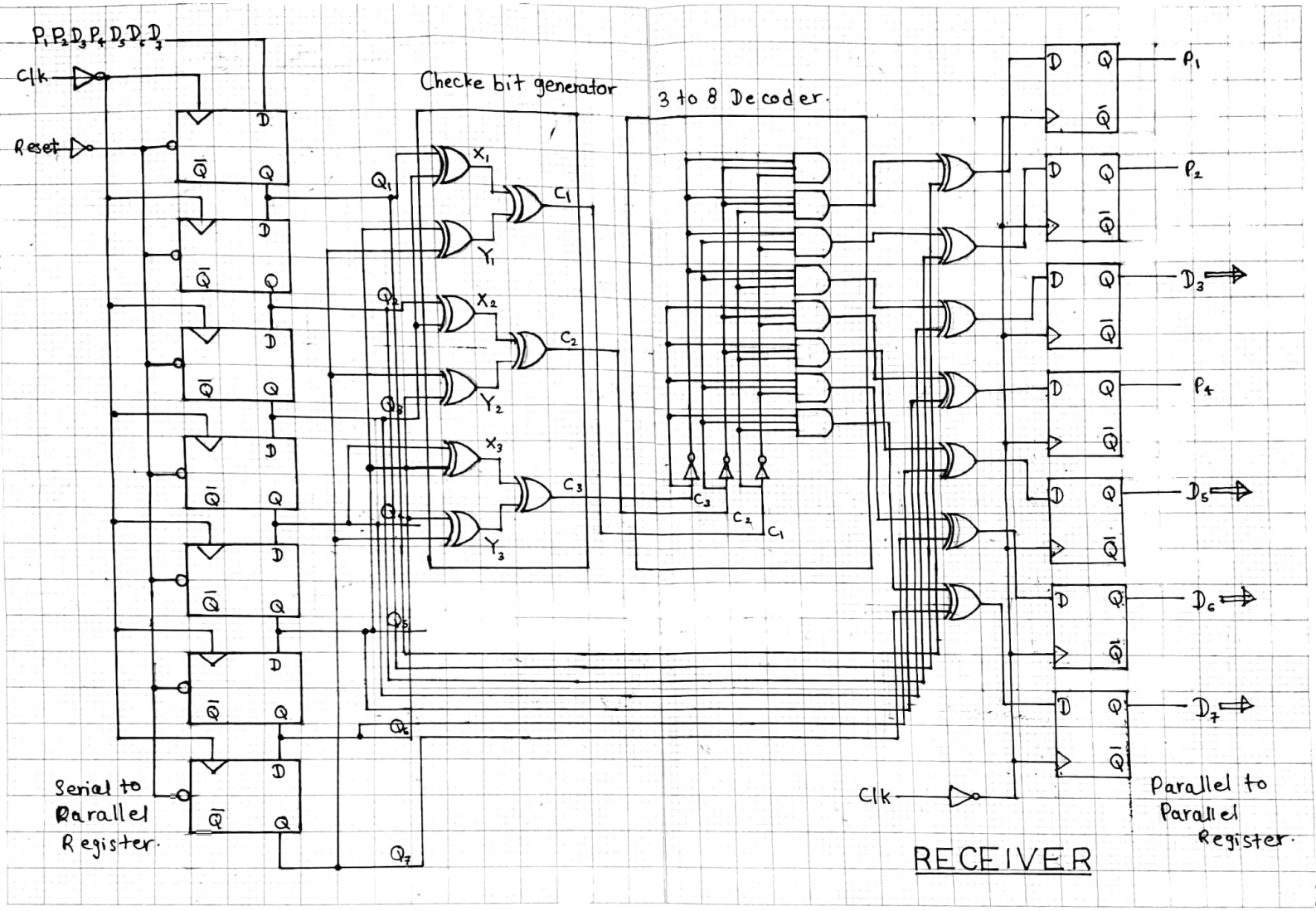
As the output of the XOR gates, corrected values of the received hamming code can be collected. The proof that this implementation works will be available in the " proof of the transmitter " part.

After the error correction, $O_1, O_2, O_3, O_4, O_5, O_6, O_7$ terminals are connected to a parallel to parallel register.

④. Final output of the receiver.

After doing the error calculations outputs are first stored in a 7 bit register.





PROOF OF THE RECEIVER

① . Proof of the checker bit generator.

$$X_1 = D_3 \oplus P_1 \;, \quad Y_1 = D_5 + D_7 \;, \quad C_1 = X_1 \oplus Y_1$$

$$C_1 = P_1 \oplus D_3 \oplus D_5 \oplus D_7 = \text{XOR of } (1, 3, 5, 7).$$

Similarly, $C_2 = \text{XOR of } (2, 3, 6, 7)$ — ①

$C_3 = \text{XOR of } (4, 5, 6, 7)$ — ②

① and ② equations were proven in the checker bit generator part.

② . Proof of the rest of the circuit.

Outputs of the decoder are,

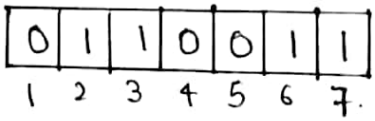
$$F_1 = C_3' . C_2' . C_1$$
$$F_2 = C_3' . C_2 . C_1'$$
$$F_3 = C_3' . C_2 . C_1$$
$$F_4 = C_3 . C_2' . C_1'$$
$$F_5 = C_3 . C_2' . C_1$$
$$F_6 = C_3 . C_2 . C_1'$$
$$F_7 = C_3 . C_2 . C_1$$

Outputs of the XOR gates are,

$$O_1 = \text{XOR}(F_1, P_1)$$
$$O_2 = \text{XOR}(F_2, P_2)$$
$$O_3 = \text{XOR}(F_3, D_3)$$
$$O_4 = \text{XOR}(F_4, P_4)$$
$$O_5 = \text{XOR}(F_5, D_5)$$
$$O_6 = \text{XOR}(F_6, D_6)$$
$$O_7 = \text{XOR}(F_7, D_7).$$

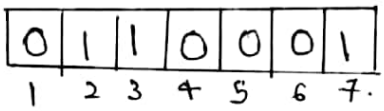
- To prove this, previous example is considered here.

Transmitted Hamming code (correct). ⇒



- If there is an error in the 6th bit while transfering,

Received Hamming Code (incorrect). ⇒



- In checker bit generator,

$$C_1 = \text{XOR}(1,3,5,7) = 0 \oplus 1 \oplus 0 \oplus 1 = 0$$
$$C_2 = \text{XOR}(2,3,6,7) = 1 \oplus 1 \oplus 0 \oplus 1 = 1$$
$$C_3 = \text{XOR}(4,5,6,7) = 0 \oplus 0 \oplus 0 \oplus 1 = 1.$$

- Outputs of the decoder,

$$F_1 = C_3' \cdot C_2' \cdot C_1 = 0 \cdot 0 \cdot 1 = 0.$$
$$F_2 = C_3' \cdot C_2 \cdot C_1' = 0 \cdot 1 \cdot 1 = 0.$$
$$F_3 = C_3' \cdot C_2 \cdot C_1 = 0 \cdot 1 \cdot 0 = 0.$$
$$F_4 = C_3' \cdot C_2' \cdot C_1' = 0 \cdot 0 \cdot 1 = 0.$$
$$F_5 = C_3 \cdot C_2' \cdot C_1 = 1 \cdot 0 \cdot 0 = 0$$
$$F_6 = C_3 \cdot C_2 \cdot C_1' = 1 \cdot 1 \cdot 1 = 1$$
$$F_7 = C_3 \cdot C_2 \cdot C_1 = 1 \cdot 1 \cdot 0 = 0$$

- Note that, the outputs of the decoder gives 1 to the bit which is the incorrect bit.

- Outputs of the XOR gates.

$O_1 = XOR(F_1, P_1) = 0 \oplus 0 = 0$
$O_2 = XOR(F_2, P_2) = 0 \oplus 1 = 1$
$O_3 = XOR(F_3, D_3) = 0 \oplus 1 = 1$
$O_4 = XOR(F_4, P_4) = 0 \oplus 0 = 0$
$O_5 = XOR(F_5, D_5) = 0 \oplus 0 = 0$
$O_6 = XOR(F_6, D_6) = 1 \oplus 0 = 1$
$O_7 = XOR(F_7, D_7) = 0 \oplus 1 = 1$

There fore output code = | 0 | 1 | 1 | 0 | 0 | 1 | 1 |

Therefore transmitted hamming code is equal to the output of the receiver.

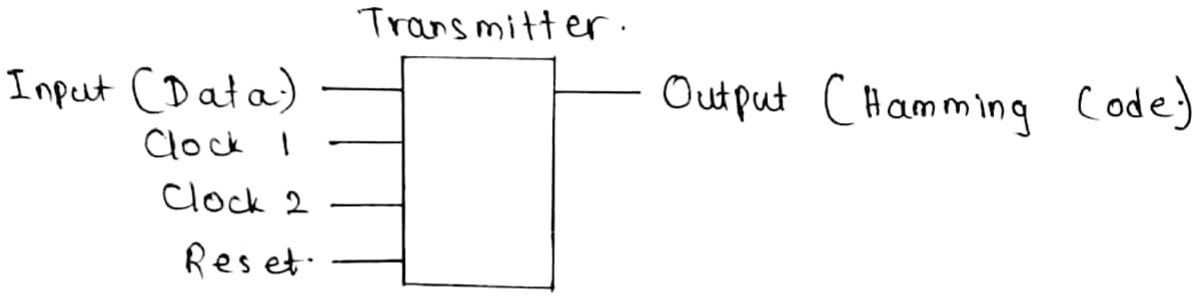
Moreover, this poof can be shown for any 4 bit combination.

Futher more, as the final conclusion, implementation of the receiver is prored to be accurater.

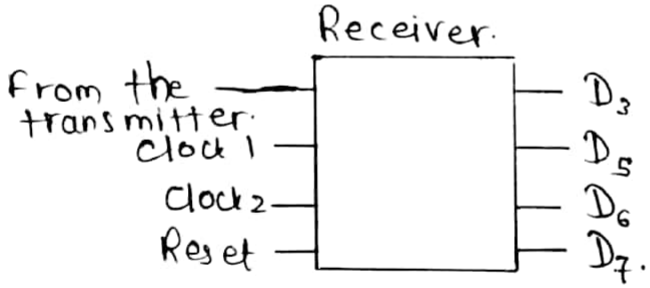
Developing the hamming code for any number of bits.

For the sake of simplicity, transmitter and receiver are represented by following symbols.

- Transmitter has 2 clock signals for each register and a reset input.



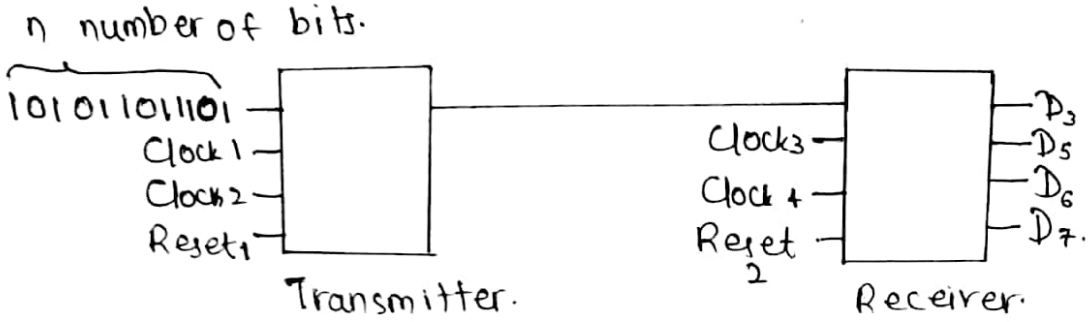
- Receiver has 2 clock signals for each register and a reset input.



- If we want to build this for any number of bits we need to consider this as a modular component. When considering n number of bits, we can talk about 2 versions.
  ①. n number of bits at T time.
  ②. n number of bits at the same time.

① n number of bits in T time.

- Suppose if we want to make hamming code for n number of bits (n is a multiple of 4.) in T time. Then we only have to use this modular component once.



- If we want to use hamming code circuit for n number of bits in T time period, first we have to talk about the time for a 4 bit data word.

- In transmiter,

  Serial to parallel register = 4 clock pulses.

- After that we make clock 1 enable.

  Load in parallel to serial register = 1 clock pulse.

  Shift in parallel to serial register = 7 clock pulses.

- After that we make clock 2 enable.

  Clock pulses
  Total ~~time~~ for 4 bits in transmitter = 12 clock pulses ~~+ propagation delay~~.

- In receiver,

  Serial to parallel register = 7 clock pulses.

- After that we make clock 3 enable.

  Parallel to parallel register = 1 clock pulse.

  Total ~~time~~ for 4 bits in receiver = 8 clock pulses.

- Total clock pulses for transmitter and receiver = 8 + 12 = 20 clock pulses.

- Therefore total Clock pulses for n number of bits = $20 \times \frac{n}{4}$ Clock pulses.

//

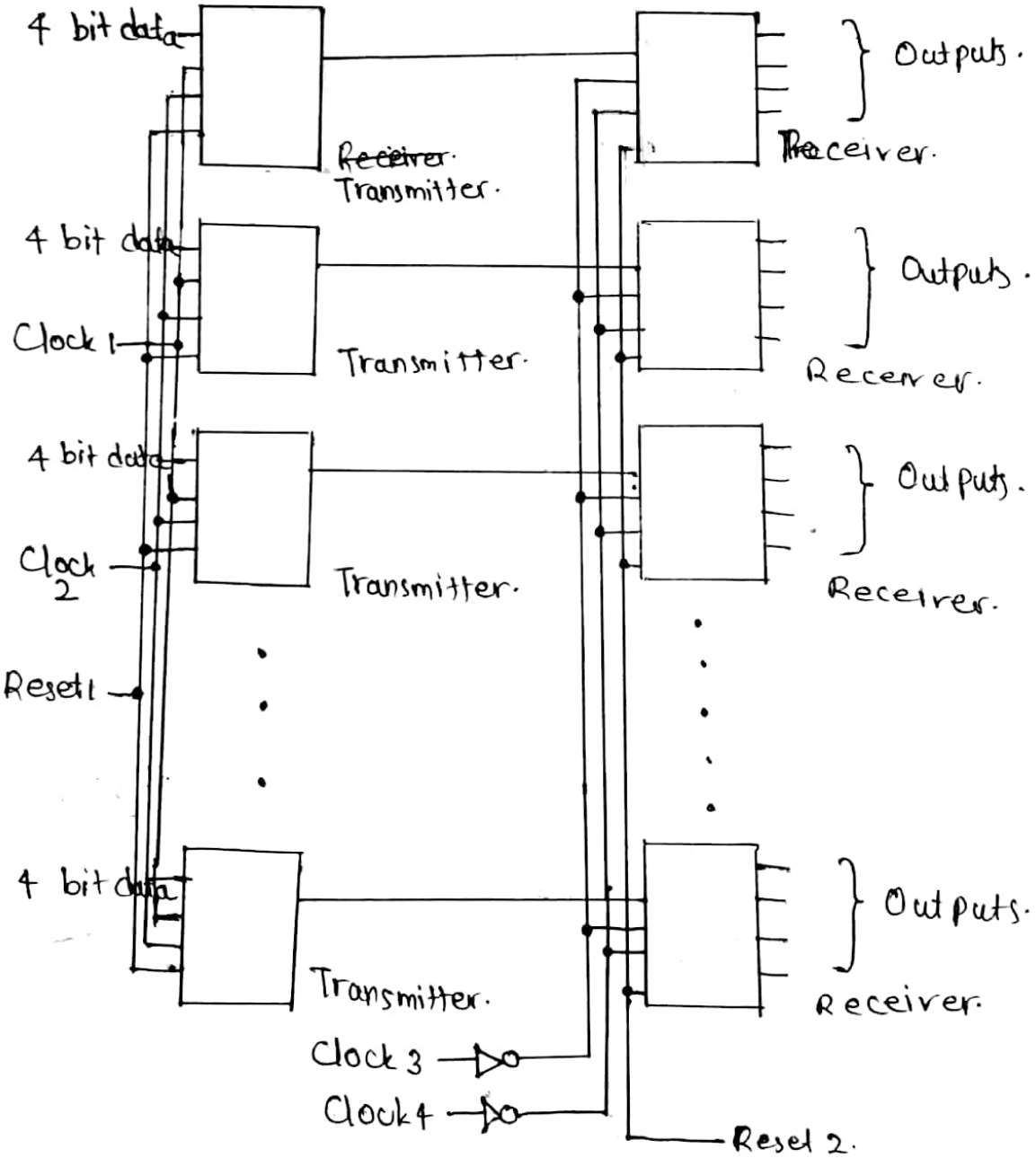
- As we can see, this is a weak implementation. Because,
  1. We can not implement this for n number of bits at the same time.
  2. We have to make clock 1, clock 2, clock 3 and clock 4 enable accordingly.
  3. There will be a lot of time consuming task since it takes a lot of clock pulses for 4 bits.

- Therefore it's better to use the version 2.

② · n number of bits at the same time.

- Note that n is a ~~number~~ multiple of 4.
- We can use this to convert any number of bits to hamming code at the same time. we have to use 4 clock generaters for this. This implementation is shown below.

Number of modular components $= \frac{n}{4}$ //



- In this implementation we can convert n number of data bits in to hamming code using only n/4 transmitter- receiver pairs.
- There will take only 20 clock pulses to generate ~~the~~ the hamming code.