2022

Project report

CRC HARDWARE IMPLEMENTATION

NAME: MUHAMMAD ISMAEEL BUTT

Index:

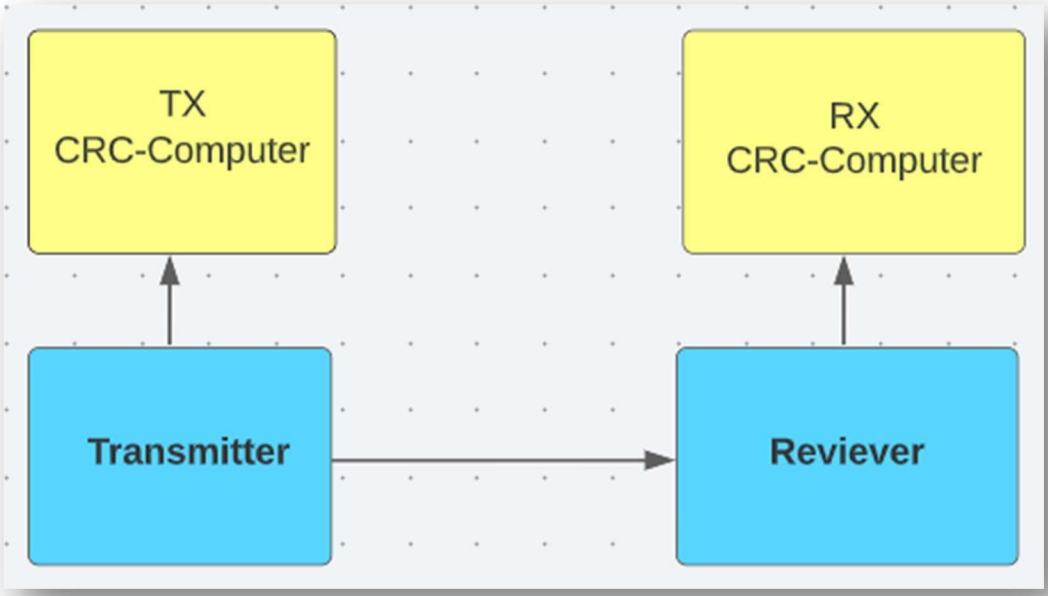
Abstract

Cyclic redundancy check (CRC) is an error detection code commonly used in digital networks and storage devices to detect random changes to digital data. Data blocks entering these systems are assigned a short check value, based on the remainder of the polynomial division on their contents. On recovery, the calculation is repeated, and if the check values do not match, corrective action can be taken for data corruption. CRC can be used to correct errors. CRC is so called because the check value (verifies the data) is redundant (it expands the message without additional information) and the algorithm is based on cyclic codes. CRCs are popular because they are easy to implement in binary hardware, easy to analyze mathematically, and are especially good at detecting common errors caused by noise in transmission channels.

Overview of Project:

CRC is used for error detection in digital data but does not necessarily make corrections to the errors detected. CRC is the polynomial division process of a binary data block by a divisor . It is used in communications protocols to send data reliably over a medium. After the remainder of the CRC division is calculated, it is appended to the data stream and transmitted. On the receiver side,the polynomial division process is performed again, and the calculated remainder is compared to the received CRC. If they're identical, it can be established that the data was received as sent and no errors were introduced by the transmission medium

Block Diagram:



Work division:

Project is divided in two halves.

1. Transmitter
2. Receiver

Transmitter is made by **Syed Muhammad Muslim** which include coding on audrino nano and complete hardware of transmitter side.

Receiver is made by **Muhammad Ismaeel Butt** which include coding on audrino nano and complete hardware of receiver side. Simulation Proteus is done by both.

Problem statement:

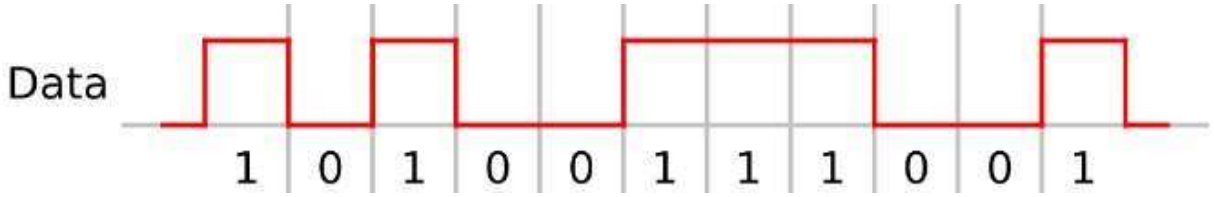
Anytime we send information from one computer to other, even a simple wire connection between transmitter and receiver there will always be a danger that data may not be received correctly depending on the connections there are a lots of ways an error can occur. So in order to detect weather the information received is correctly or not we have design a hardware (cyclic redundancy check (CRC) is an error-detecting code commonly used in digital networks and storage devices to detect accidental changes to digital data) which is based on math to configure and show the error in our data transmission.

Steps to make hardware design:

- Data transmission to 16 bit LED using Arduino Nano.
- Data transmission to receiver from transmitter
- Making similar clock for transmitter and receiver
- CRC working
- CRC calculation in hardware

Data transmission to 16 bit LED using Arduino Nano:

First we place the Arduino Nano on bread board and hook up it's digital pins with the 16 bit LED display screen and data pin is connected with a red led and display screen in order to see what data (0,1) is coming from the Arduino Nano.In this hardware Arduino Nano is used only for data and clock working which is on pin D3(data) and D2 (clock) . First we implement a simply code to see weather the 16 bit LED screen and red led is working or not then we move to the next step.So the data transfer to 16 bits display screen will look like this (some randome data):

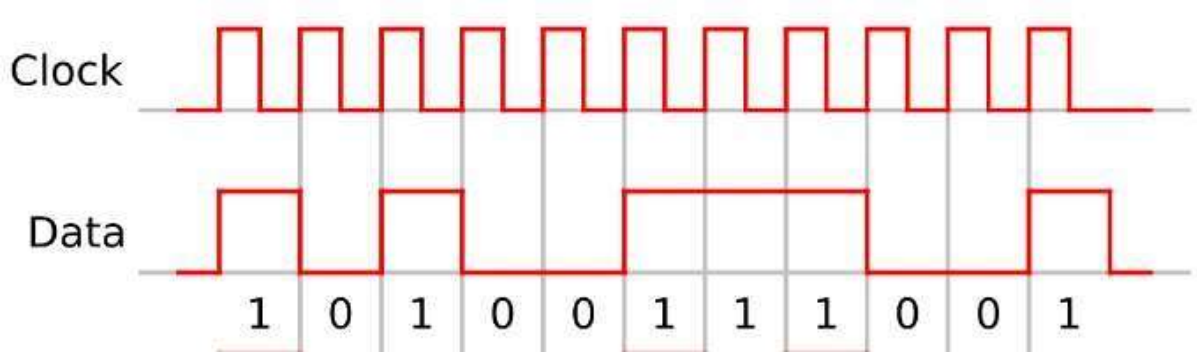


Data transmission to receiver from transmitter:

Receiver is made in same way we made the transmitter using Arduino Nano and connecting it to the 16 bit display screen and data pin of receiver is connected with display screen , green Led and with the transmitter data pin so data can be transfer from transmitter to receiver.both transmitter and receiver has common ground as well. Till this point the data is only transfer from transmitter to receiver but its not valid data transfer reason is that the receiver and transmitter has different clock beside same data pin connection we have to make both clock similar and after that we check weather the data is correct we start CRC implementation in next step.

Making similar clock for transmitter and receiver:

To make clock similar we use d2 pin to transfer clock signal from transmitter to receiver and connect it with LED to see weather it is similar or not .From this there is three connection between transmitter and receiver ground ,data connection and clock signal connection.



CRC working:

Math step: for example we want to transfer Hi!

First we see the binary and decimal equivalent of the word Hi!.

Word	H in binary	i in binary	! In binary	Decimal equivalent
Hi!	01001000	01101001	00100001	4745505

Now we transfer some extra bits at the end this change the decimal equivalent of the word.

Word	H in binary	i in binary	! In binary	extra bits	extra bits	Decimal equivalent
Hi!	01001000	01101001	00100001	00000000	00000000	311001415680

Now divide the new decimal equivalent with some decimal number assume a number is 65521.

When we divide **311001415680 with 65521** we get **26769 as a remainder**. We subtract 26769 from the 65521 so we get **38752.**

If we add 38752 in 311001215680 we will get 3110014154432 which is the then again divided by 65521 and we get remainder zero.

So if we add the binary equivalent of this number in place of extra bits and divide it by 65521 we get **zero remainder all the time.**

So if we transfer the data from transmitter to receiver and perform this process will know weather the data transfer is correct or not.

Now extra bits are binary equivalent of 38752.

Word	H in binary	i in binary	! In binary	extra bits	extra bits	Decimal equivalent
Hi!	01001000	01101001	00100001	10010111	01100000	311001454432

311001454432 divided by 65521 will give zero remainder.

If the data transfer is not correct then the remainder will not be zero.

Example:

Instead of Hi! We get Ho! Then the remainder will not be zero.

Word	H in binary	o in binary	! In binary	extra bits	extra bits	Decimal equivalent
Ho!	01001000	01101111	00100001	10010111	01100000	311102117728

When 311102117728 is divided by the number 65521 remainder is 23040 and we can detect error.

Converting message into polynomial:

We know H in binary is equivalent to so

H = 0 1 0 0 1 0 0 0 = 72 (decimal)

$0x2^7 + 1x2^6 + 0x2^5 + 0x2^4 + 1x2^3 + 0x2^2 + 0x2^1 + 0x2^0 = 72$

But in polynomial form we get

$$\mathbf{0\ x\ X^7 + 1\ x\ X^6 + 0\ x\ X^5 + 0\ x\ X^4 + 1\ x\ X^3 + 0\ x\ X^2 + 0\ x\ X^1 + 0\ x\ X^0 = X^6 + X^3}$$

SO H in polynomial from is $\mathbf{X^6 + X^3}$

Word	H in binary	i in binary	! In binary	extra bits	extra bits
Hi!	01001000	01101001	00100001	00000000	00000000

In polynomial form Hi! is

$$\mathbf{X^{38} + X^{35} + X^{30} + X^{29} + X^{27} + X^{24} + X^{21} + X^{16}}$$

now we divide is by some random polynomial like we did in above step:

Divider is $\mathbf{X^{16} + X^{12} + X^5 + 1}$

Now we use long division method to find the remainder.

So first we have to look at the field axioms and field of [0,1]

Field axioms:

Associative property:

A+(B+C) = (A+B)+C A.(B.C) = (A.B).C

Commutative property: A+B = B+A A.B = B.A

Identity: A+0 = A A.1= A

Additive inverse: A+(-A)= 0

multiplicative inverse: $A \cdot \frac{1}{A} = 1$

Distributive property: A.(B+C) = (A.B)+(A.C)

Field of [0 1]

F[0,1]:

0+0=0	**0+1=1**	**1+0=1**	**1+1=0**
0.0=0	**0.1=0**	**1.0=0**	**1.1=1**
0-0=0	**0-1=1**	**1-0=1**	**1-1=0**

$\frac{0}{1} = 1$ $\frac{1}{1} = 1$

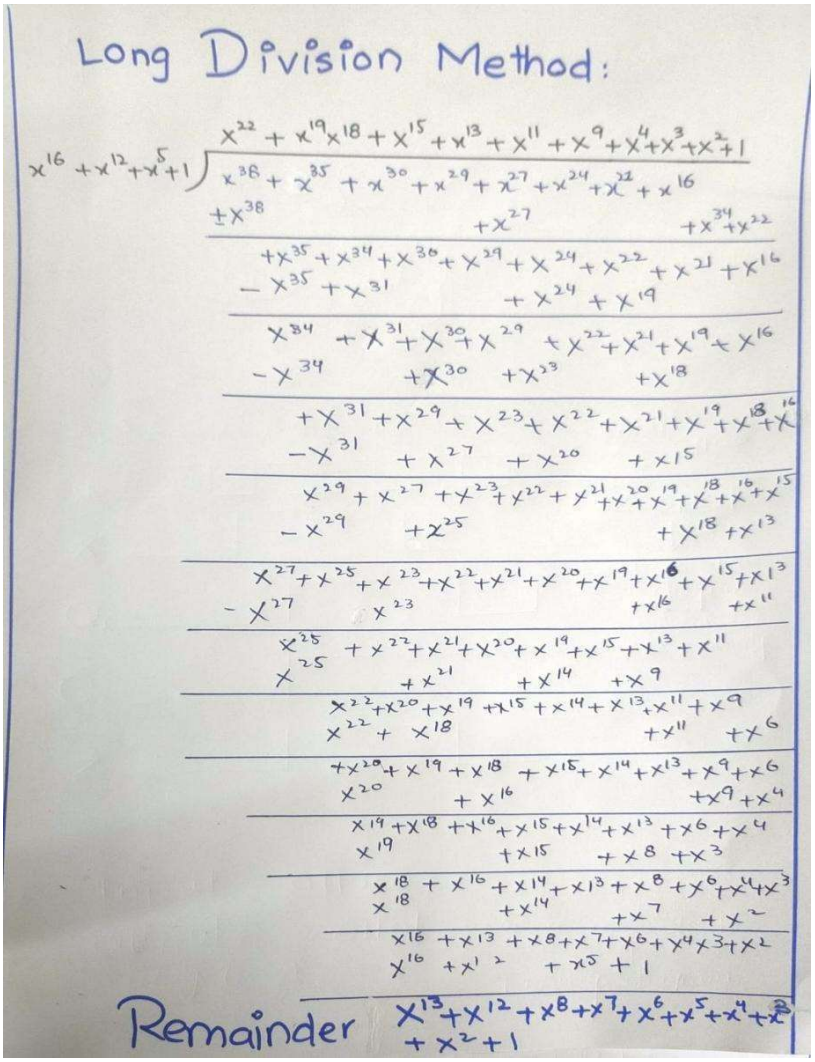
Long division method:

Hi! Polynomial is $X^{38} + X^{35} + X^{30} + X^{29} + X^{27} + X^{24} + X^{21} + X^{16}$

Divided is $X^{16} + X^{12} + X^{5} + 1$

$$X^{16} + X^{12} + X^{5} + 1 \overline{)\ X^{38} + X^{35} + X^{30} + X^{29} + X^{27} + X^{24} + X^{21} + X^{16}}$$

$$X^{22} + X^{19}\ X^{18} + X^{15} + X^{13} + X^{11} + X^{9} + X^{4} + X^{3} + X^{2}$$



Remainder is : $X^{13} + X^{12} + X^{8} + X^{7} + X^{6} + X^{5} + X^{4} + X^{3} + X^{2} + 1$

Now we convert it into the binary number

Which is **0 0 1 1 0 0 0 1 1 1 1 1 1 1 0 1**

Message polynomial = m(x)= $X^{38} + X^{35} + X^{30} + X^{29} + X^{27} + X^{24} + X^{21} + X^{16}$

generator polynomial = g(x)= $X^{16} + X^{12} + X^5 + 1$

The remainder will be added in extra bits of m(x).

T(x) = transmitted message

So t(x) = $X^{38} + X^{35} + X^{30} + X^{29} + X^{27} + X^{24} + X^{21} + X^{16}\, X^{13} + X^{12} + X^8 + X^7 + X^6 + X^5 + X^4 + X^3 + X^2 + 1$

t(x) = m(x) - [m(x) mod g(x)]

m(x) mod g(x) = 0 which is same as we did in math step to make **remainder zero.**

<u>CRC calculation in hardware:</u>

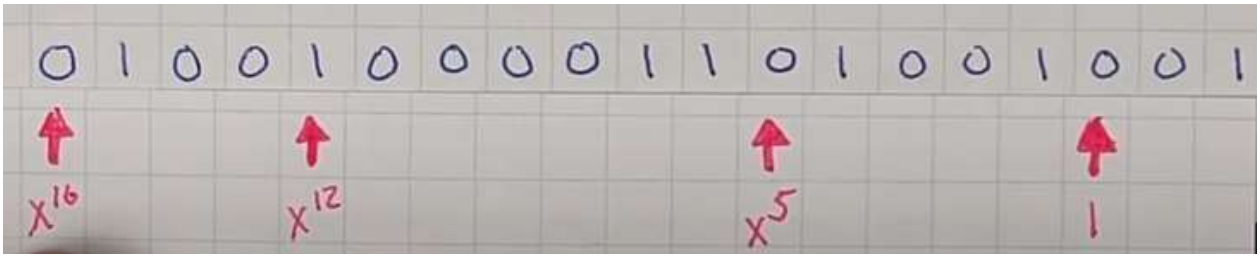
as we now message been transmitted is Hi!

Word	H in binary	i in binary	! In binary	extra bits	extra bits
Hi!	01001000	01101001	00100001	**00110001**	**11111101**

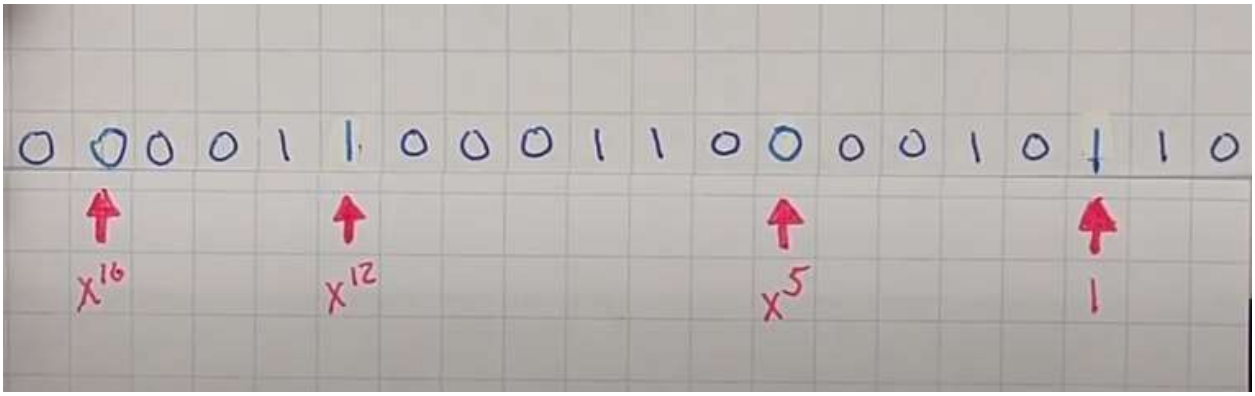
g(x)= $X^{16} + X^{12} + X^5 + 1$

Now we slides these bits on the g(x) and when the X^{16} is on 0 zero there is no flip on X^{16} , X^{12}, X^5, 1 if it is on it there will be a flip on these locations X^{16} , X^{12}, X^5, 1.

No flip case:

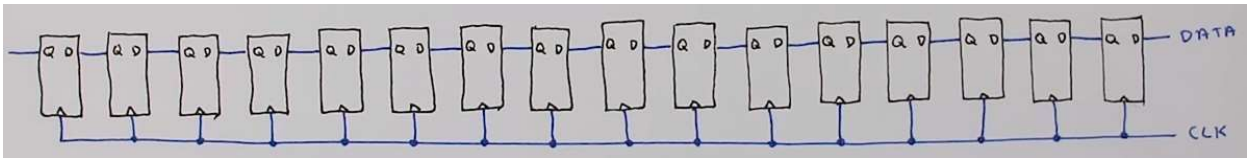


Flipping case:

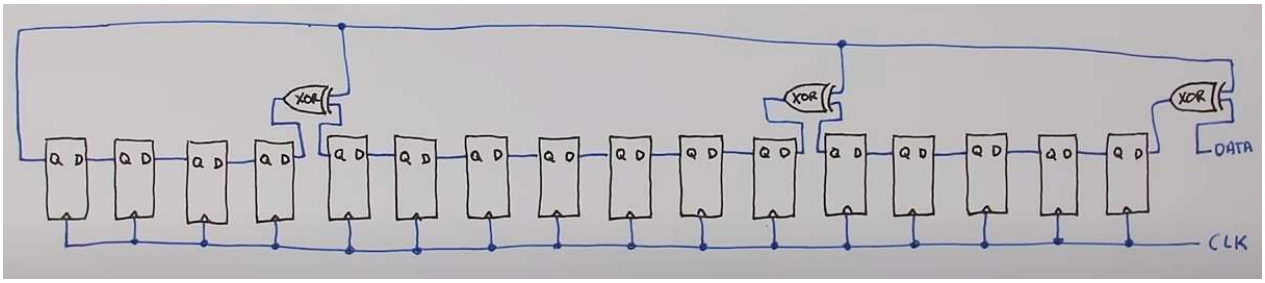


Sliding of bit continuous until the last bit of message and at the end we will get our CRC.

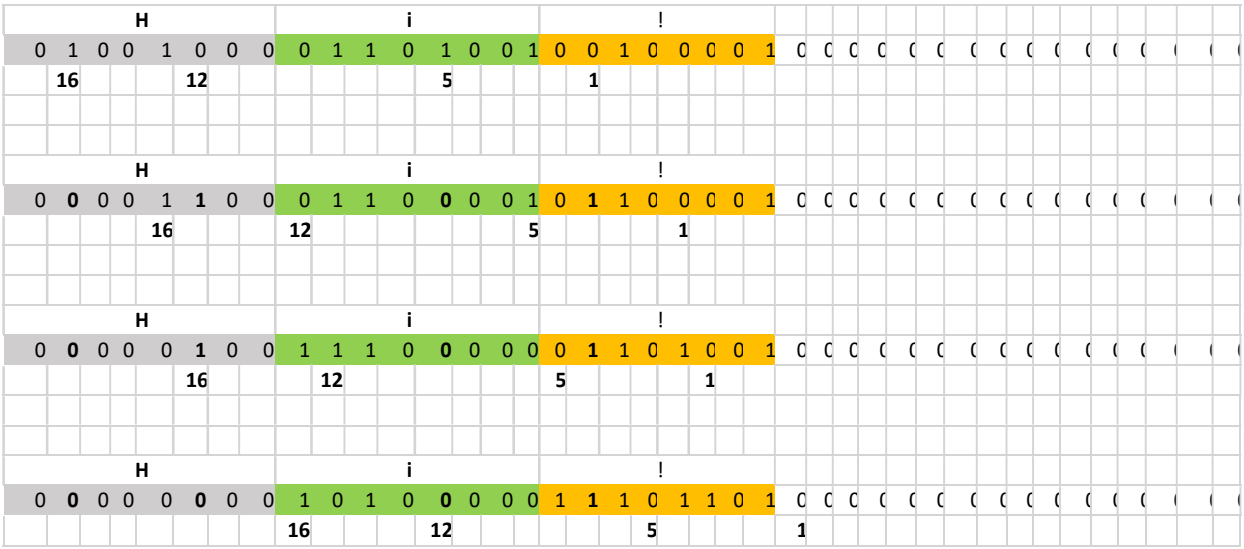
For sliding we are using d flip flop:



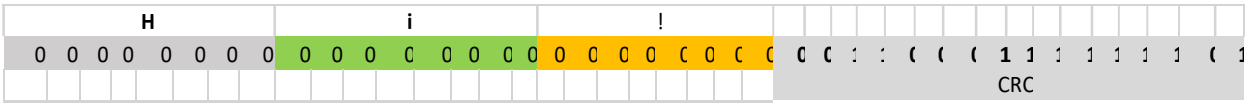
And for flipping we are using XOR gates:



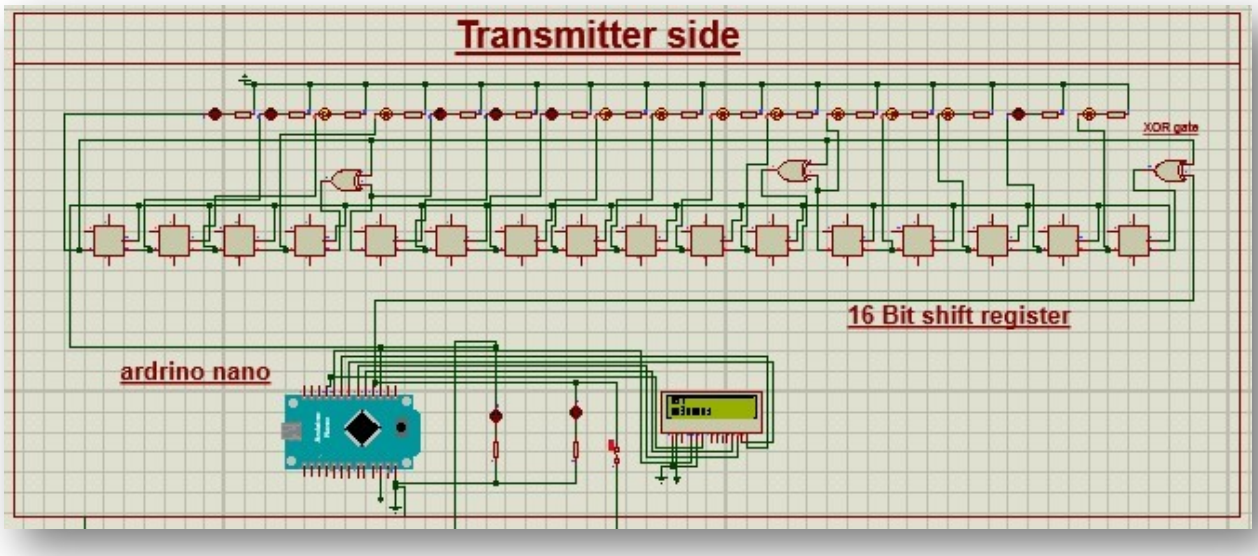
So these are the steps for hardware design.

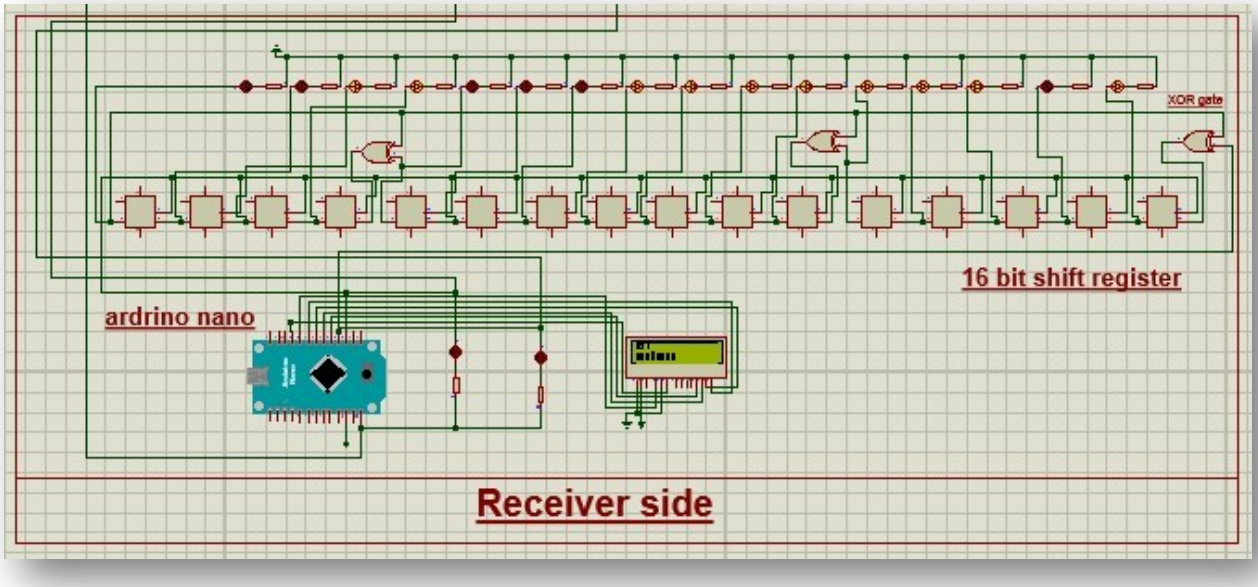


After Complete sliding and flipping:

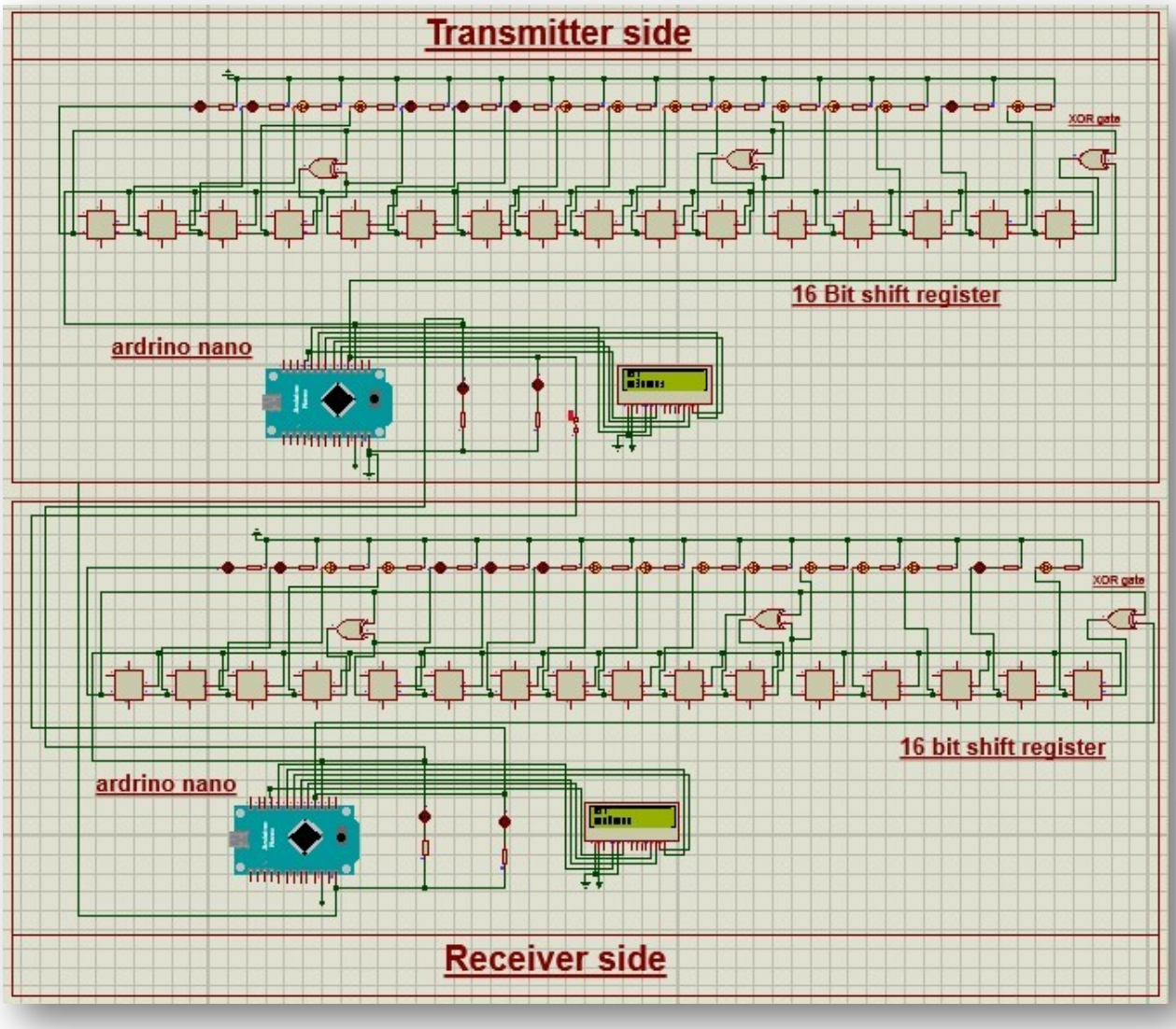


Simulation:


Transmitter side
XOR gate
16 Bit shift register
ardrino nano


XOR gate
16 bit shift register
ardrino nano
Receiver side

Complete Schematic:



Transmitter code:


```
#include <LiquidCrystal.h>
// Transmit rate in bps
#define TX_RATE 5
// Pin assignments
#define TX_CLOCK 2
#define TX_DATA 3
#define LCD_D4 4
#define LCD_D5 5
#define LCD_D6 6
#define LCD_D7 7
#define LCD_RS 8
#define LCD_EN 9
#define CRC 10
const char *message = "Hi!";
void setup() {
  pinMode(TX_CLOCK, OUTPUT);
  pinMode(TX_DATA, OUTPUT);
  pinMode(CRC, INPUT);

  // Initialize the LCD screen
  LiquidCrystal lcd(LCD_RS, LCD_EN, LCD_D4, LCD_D5, LCD_D6, LCD_D7);
  lcd.begin(16, 2);
  lcd.setCursor(0, 0);
  lcd.print(message);

  for (int byte_idx = 0; byte_idx < strlen(message); byte_idx++) {
    char tx_byte = message[byte_idx];
    // Clear the second line of the display
    lcd.noCursor();
    lcd.setCursor(0, 1);
    lcd.print("        ");
    lcd.setCursor(byte_idx, 0);
    lcd.cursor();

    for (int bit_idx = 0; bit_idx < 8; bit_idx++) {
      bool tx_bit = tx_byte & (0x80 >> bit_idx);

      digitalWrite(TX_DATA, tx_bit);
      delay((1000 / TX_RATE) / 2);

      // Update the LCD
      lcd.noCursor();
      lcd.setCursor(bit_idx, 1);
      lcd.print(tx_bit ? "1" : "0");
      lcd.setCursor(byte_idx, 0);
      lcd.cursor();

      // Pulse clock
      digitalWrite(TX_CLOCK, HIGH);
```

```
      delay((1000 / TX_RATE) / 2);
      digitalWrite(TX_CLOCK, LOW);
    }
  }
  digitalWrite(TX_DATA,LOW);
  for(int i =0 ; i<16; i++)
  {
    delay((1000 / TX_RATE) / 2);
    digitalWrite(TX_CLOCK, HIGH);
    delay((1000 / TX_RATE) / 2);
    digitalWrite(TX_CLOCK, LOW);
  }
//  digitalWrite(TX_DATA, digitalRead(PARITY));

  digitalWrite(TX_DATA, LOW);
}
```

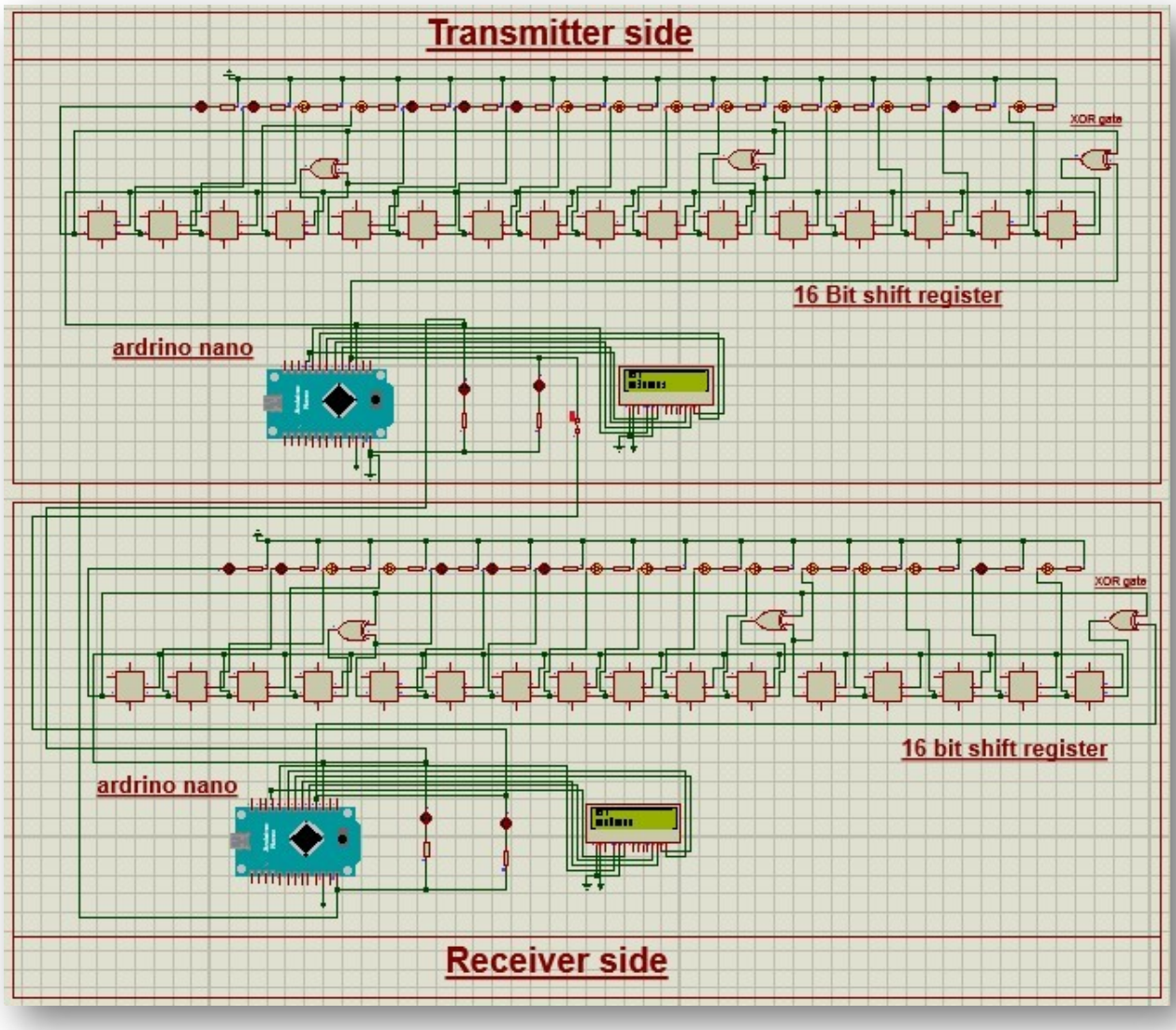
Receiver code:

```
#include <LiquidCrystal.h>
// Transmit rate in bps
#define TX_RATE 5
// Pin assignments
#define TX_CLOCK 2
#define TX_DATA 3
#define LCD_D4 4
#define LCD_D5 5
#define LCD_D6 6
#define LCD_D7 7
#define LCD_RS 8
#define LCD_EN 9
#define CRC_OUTPUT 10
#define CRC_READ 11
#define CRC_CLOCK 13
const char *message = "Hello, world!";
void setup() {
  pinMode(TX_CLOCK, OUTPUT);
  pinMode(TX_DATA, OUTPUT);
  pinMode(CRC_OUTPUT, INPUT);
  pinMode(CRC_READ, OUTPUT);
  pinMode(CRC_CLOCK, OUTPUT);
```

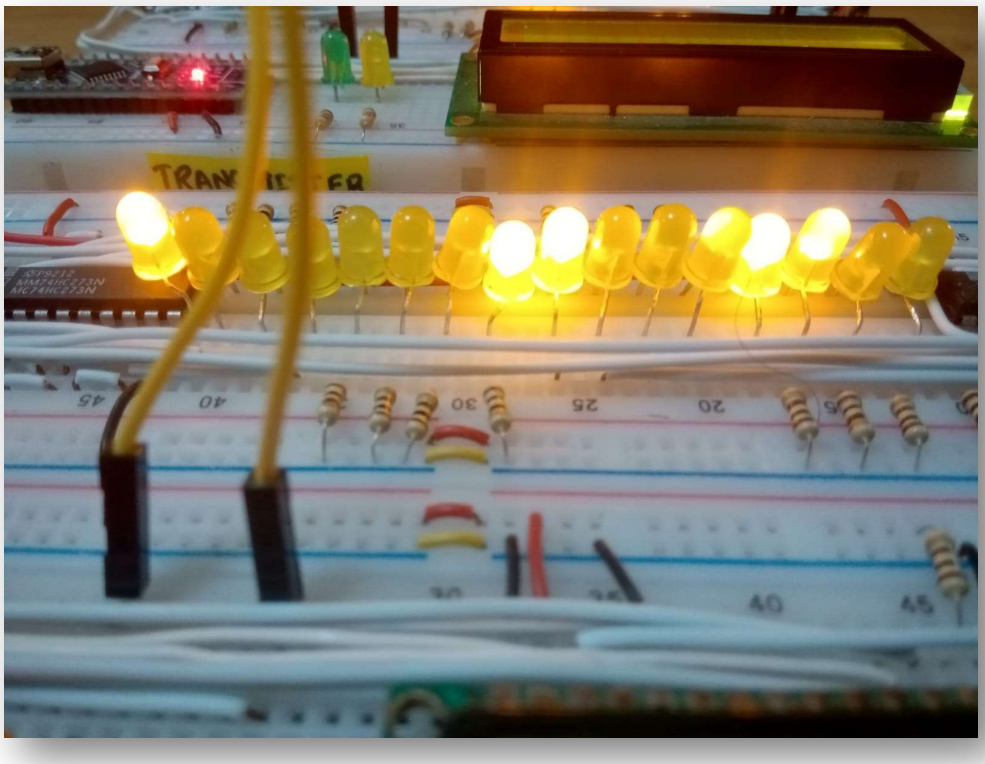
```
digitalWrite(CRC_READ, HIGH);
// Initialize the LCD screen
LiquidCrystal lcd(LCD_RS, LCD_EN, LCD_D4, LCD_D5, LCD_D6, LCD_D7);
lcd.begin(16, 2);
lcd.setCursor(0, 0);
lcd.print(message);
// Transmit data
for (int byte_idx = 0; byte_idx < strlen(message); byte_idx++) {
  char tx_byte = message[byte_idx];
  // Clear the second line of the display
  lcd.noCursor();
  lcd.setCursor(0, 1);
  lcd.print("        ");
  lcd.setCursor(byte_idx, 0);
  lcd.cursor();

  for (int bit_idx = 0; bit_idx < 8; bit_idx++) {
    bool tx_bit = tx_byte & (0x80 >> bit_idx);
    digitalWrite(TX_DATA, tx_bit);
    delay((1000 / TX_RATE) / 2);
    // Update the LCD
    lcd.noCursor();
    lcd.setCursor(bit_idx, 1);
    lcd.print(tx_bit ? "1" : "0");
    lcd.setCursor(byte_idx, 0);
    lcd.cursor();
    // Pulse clock
    digitalWrite(TX_CLOCK, HIGH);
    digitalWrite(CRC_CLOCK, HIGH);
    delay((1000 / TX_RATE) / 2);
    digitalWrite(TX_CLOCK, LOW);
    digitalWrite(CRC_CLOCK, LOW);
  }
}
// Fill the CRC circuit with 16 zeros
```

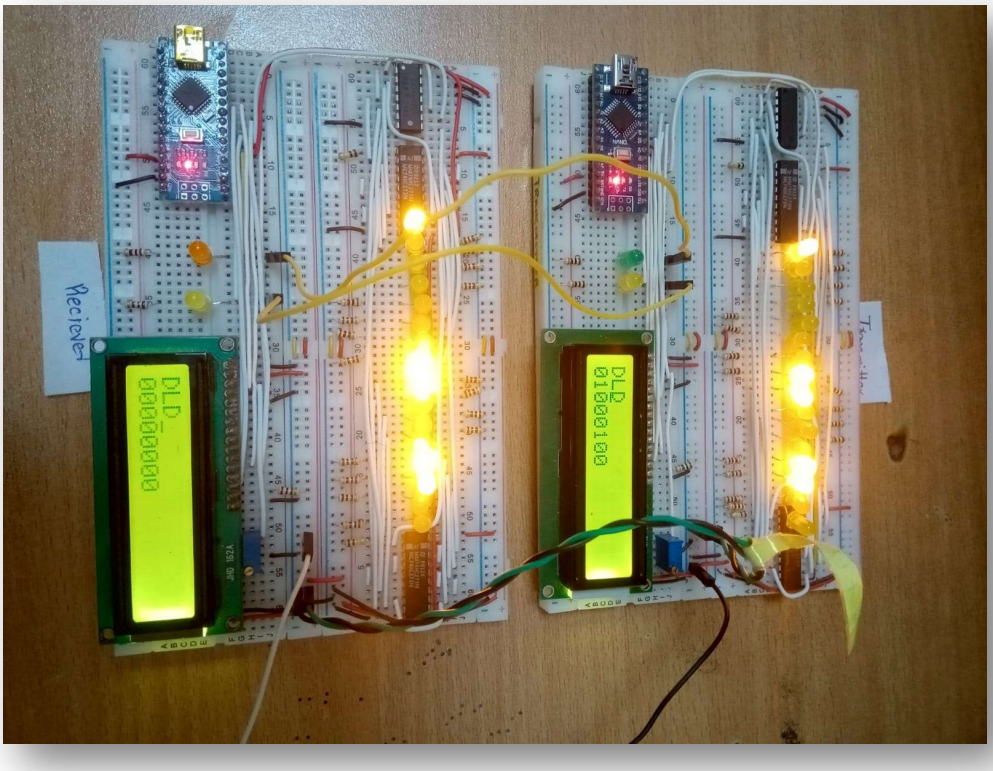
```
  digitalWrite(TX_DATA, LOW);
  for (int i = 0; i < 16; i += 1) {
    delay((1000 / TX_RATE) / 2);
    digitalWrite(CRC_CLOCK, HIGH);
    delay((1000 / TX_RATE) / 2);
    digitalWrite(CRC_CLOCK, LOW);
  }
  // Read CRC from CRC circuit and transmit it
  digitalWrite(CRC_READ, LOW);
  lcd.setCursor(0, 1);
  lcd.print("                ");
  lcd.setCursor(0, 1);
  for (int i = 0; i < 16; i += 1) {
    lcd.print(digitalRead(CRC_OUTPUT) ? "1" : "0");
    digitalWrite(TX_DATA, digitalRead(CRC_OUTPUT));
    delay((1000 / TX_RATE) / 2);
    digitalWrite(CRC_CLOCK, HIGH);
    digitalWrite(TX_CLOCK, HIGH);
    delay((1000 / TX_RATE) / 2);
    digitalWrite(CRC_CLOCK, LOW);
    digitalWrite(TX_CLOCK, LOW);
  }
  digitalWrite(TX_DATA, LOW);
}
void loop() {
  // put your main code here, to run repeatedly:
}
```

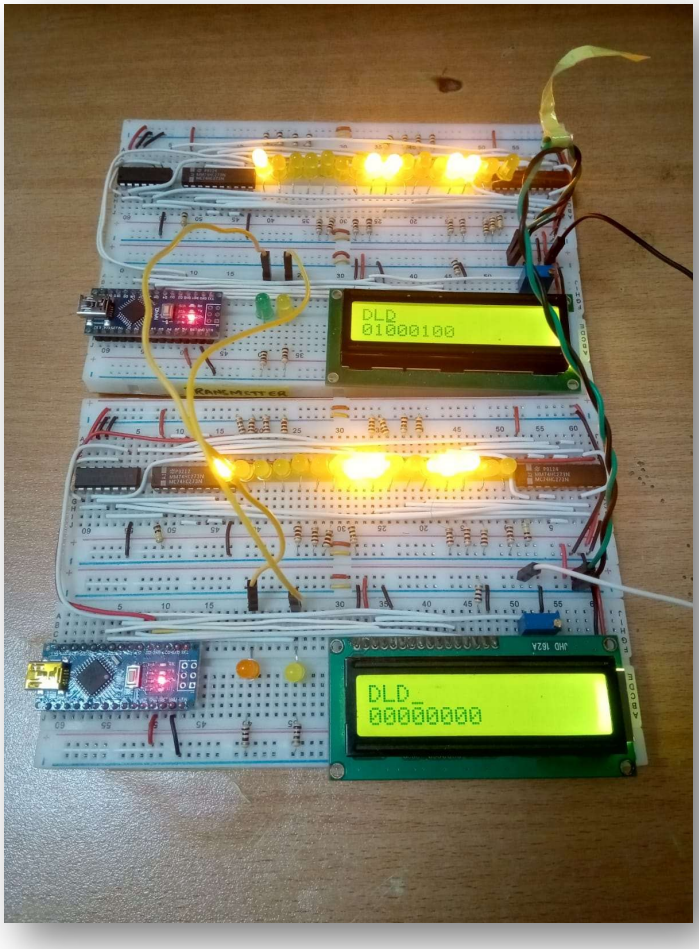
Schematic Diagram:

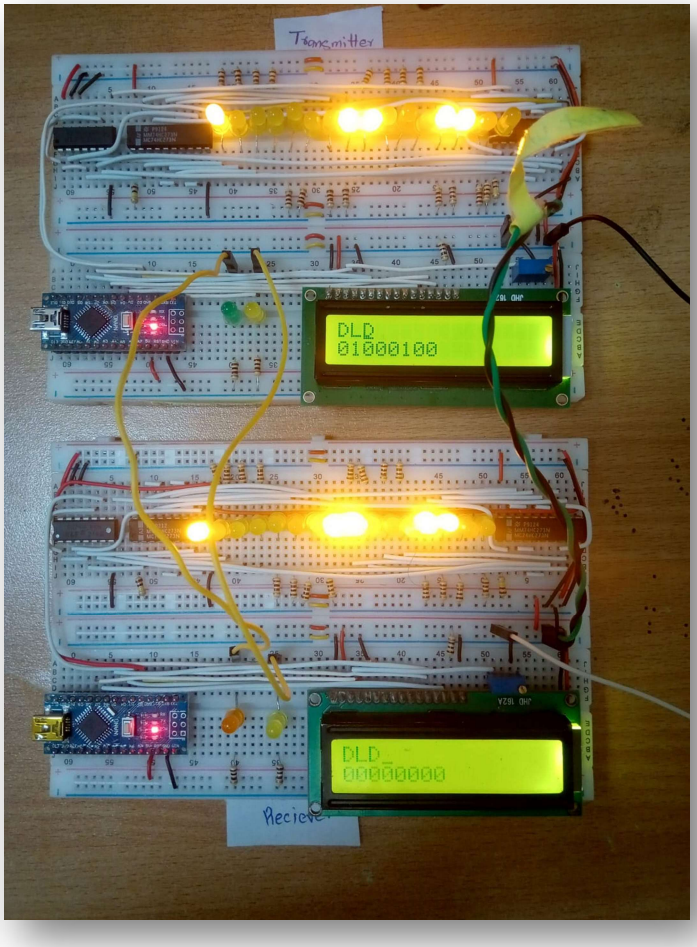


Hardware screenshot:






DLD
01000100
DLD
0000000000


Transmitter
DLD
01000100
DLD
0000000000
Reciever

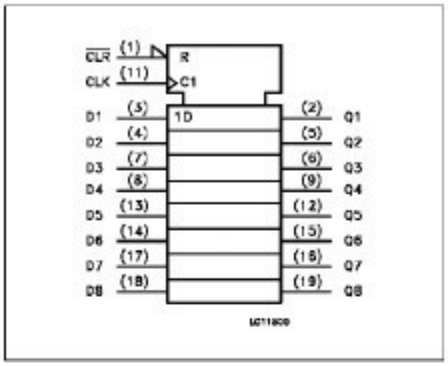
Index of ICs used:

4x 74HC273:

PIN DESCRIPTION

PIN No	SYMBOL	NAME AND FUNCTION
1	CLEAR	Master Reset Input (Active LOW)
2, 5, 6, 9, 12, 15, 16, 19	Q0 to Q7	Flip Flop Outputs
3, 4, 7, 8, 13, 14, 17, 18	D0 to D7	Data Inputs
11	CLOCK	Clock Input (LOW to HIGH, Edge Triggered)
10	GND	Ground (0V)
20	V_{CC}	Positive Supply Voltage

IEC LOGIC SYMBOL

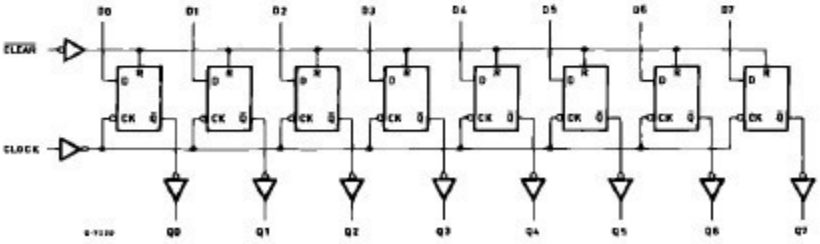


TRUTH TABLE

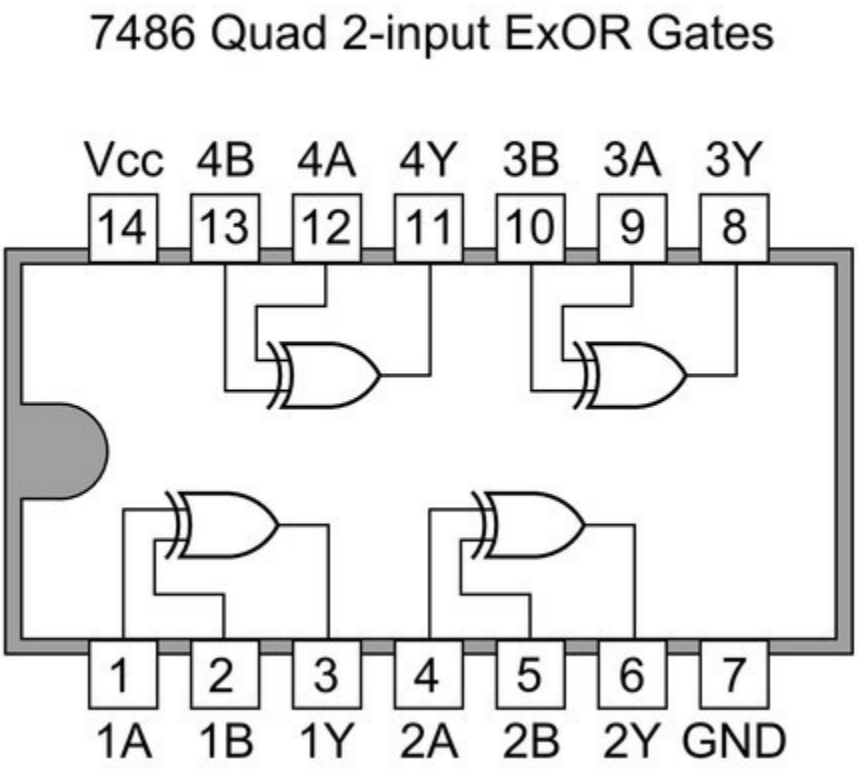
INPUTS			OUTPUS	FUNCTION
CLEAR	CLOCK	D	Q	
L	X	X	L	CLEAR
H	⎍	L	L	
H	⎍	H	H	
H	⎏	X	Qn	NO CHANGE

X: Don't Care

LOGIC DIAGRAM



2x 74HC86:



Details of Other Components used:

4x Breadboards

2x Arduino nanos (link is for a 3-pack)

2x 16x2 LCD panel

Resistors ~40x 220Ω-470Ω or so (for current-limiting LED & LCD backlight) 5x 100kΩ

32x Yellow LED

Results/ Observations:

This project helps us to build new knowledge and design and how to detail research.

From this project we made a hardware which is used to detect error in the data transfer from one transfer to receiver. We studies how to do flipping and sliding using d flip flop and XOR gates. We learn how interact two devices like we did in our project. We send message from one Arduino Nano to other one and this will help in future project.

Project Applications:

CRC is used to detect error in message transfer from the source to the receiver. It is very useful as the data now a days can be manipulated easily and CRC insure that the data transfer to the receiver remains error free or not its will tell where the error has occurred.

Future Recommendations:

CRC is been done by using different ICs and programming.In future it may have capability to detect and correct error automatically.

References:

- **https://en.wikipedia.org/wiki/Cyclic_redundancy_check**
- **https://web.archive.org/web/20110719042902/http://sar.informatik.hu-berlin.de/research/publications/SAR-PR-2006-05/SAR-PR-2006-05_.pdf**

FAIRCHILD
SEMICONDUCTOR®

September 1983
Revised May 2005

MM74HC273
Octal D-Type Flip-Flops with Clear

General Description

The MM74HC273 edge triggered flip-flops utilize advanced silicon-gate CMOS technology to implement D-type flip-flops. They possess high noise immunity, low power, and speeds comparable to low power Schottky TTL circuits. This device contains 8 master-slave flip-flops with a common clock and common clear. Data on the D input having the specified setup and hold times is transferred to the Q output on the LOW-to-HIGH transition of the CLOCK input. The CLEAR input when LOW, sets all outputs to a low state.

Each output can drive 10 low power Schottky TTL equivalent loads. The MM74HC273 is functionally as well as pin compatible to the 74LS273. All inputs are protected from damage due to static discharge by diodes to V_{CC} and ground.

Features

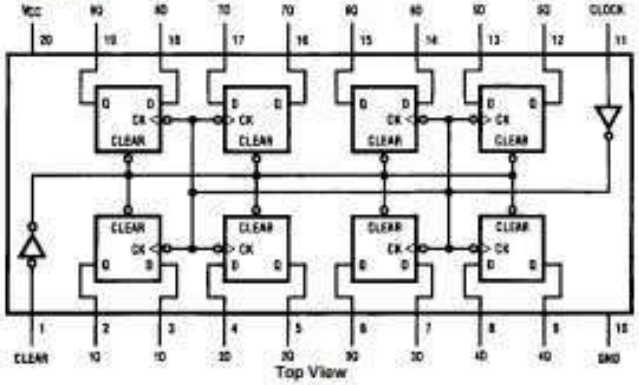
- Typical propagation delay: 18 ns
- Wide operating voltage range
- Low input current: 1 μA maximum
- Low quiescent current: 80 μA (74 Series)
- Output drive: 10 LS-TTL loads

Ordering Code:

Order Number	Package Number	Package Description
MM74HC273WM	M20B	20-Lead Small Outline Integrated Circuit (SOIC), JEDEC MS-013, 0.300" Wide
MM74HC273SJ	M20D	20-Lead Small Outline Package (SOP), EIAJ TYPE II, 5.3mm Wide
MM74HC273MTC	MTC20	20-Lead Thin Shrink Small Outline Package (TSSOP), JEDEC MO-153, 4.4mm Wide
MM74HC273N	N20A	20-Lead Plastic Dual-In-Line Package (PDIP), JEDEC MS-001, 0.300" Wide

Devices also available in Tape and Reel. Specify by appending the suffix letter "X" to the ordering code.

Connection Diagram



Top View

MM74HC273

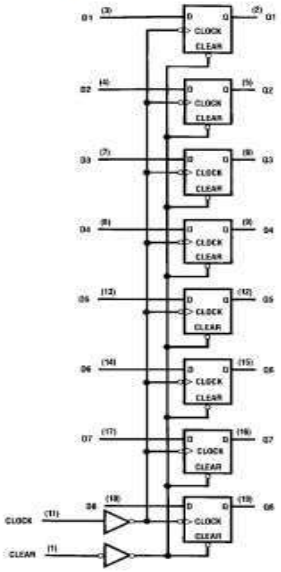
Truth Table

(Each Flip-Flop)

Inputs			Outputs
Clear	Clock	D	Q
L	X	X	L
H	↑	H	H
H	↑	L	L
H	L	X	Q_0

H = HIGH Level (Steady State)
L = LOW Level (Steady State)
X = Don't Care
↑ = Transition from LOW-to-HIGH level
Q_0 = The level of Q before the indicated steady state input conditions were established

Logic Diagram



7486 Quad 2-input ExOR Gates