Quaid-i-Azam University, Islamabad

Lab Report By:

Mr. Faizan Saleem Siddiqui

(BS-5th, Fall22, Dept of Electronics)

Submitted to:

Dr. Muhammad Zia

Lab Report on:

Design & Implementation of a Synchronous Parallel Data Transmission Channel between two ESP-32 Boards.

Student Name:	**Faizan Saleem Siddiqui**
Student ID:	**04102013033**
Subject:	**Microcontroller Interfacing**
Course Code:	**EL-344**
Class:	**BS-5th**
Session:	**Fall2022**
Department:	**Electronics**

Date: ______________________ Sig: ________________________________

Table of Contents

Introduction

This Lab report is based on an experiment of design of a parallel communication channel between the two ESP-32 boards. In this report a detail working procedure of this experiment is discussed and all the challenges that were met in its design have been overcome. Furthermore, the requirements and other necessary things have been also mentioned and discussed.

Based on these divisions, this Lab Report has following sections.

Overview Section: In this section, detail understanding of the problem is presented. Such as what is the main requirement asked by the examiner of what actually our challenge is? As we saw the first one-line statement in beginning of this document, we have texted a detail paraphrase of our problem statement and limitations put on us by the examiner. We have also presented in detail the other practical feasibilities in this section for the better understanding of problem statement.

Solution & Procedure: In this section a detail theoretical solution of the mentioned problem is discussed. The detail of procedure of writing data on transmission line is discussed. Afterwards, the receiving procedure is also presented in detail, that how data will be received at other end.

Apparatus & Requirements: In this section, all the physical requirements and components are listed and their role is also mentioned.

Wiring Connections: A schematic view of wiring connection is presented in this section.

State & Timing Diagrams: All the timing diagrams and state diagrams are presented and explained in this section.

Arduino Codes: In this section the cades written in C++ language is presented for both receiver and transmitter end.

Problem Statement

Design & Implement a Communication channel that:

1. Transmits 8-bits of data in parallel from one ESP-32 boars to another
2. In Asynchronous Mode Simplex, Half-Duplex and Full Duplex
3. Without using any port, wi-fi, Bluetooth or any other physical medium.

You can only use:

1. timers interrupts,
2. External Interrupts & interrupts service routines
3. And GPIOs in high or low state and INPUT or OUTPUT mode.

You can use jumper wires between GPIOs for connection.

Overview

In this problem we are asked to design a data transmitting channel that should transmit 8-bits of data in parallel from one ESP board to other. We are prohibited from using any port (UART, I2C, Wi-fi or Bluetooth). We can only use GPIOs in their specified modes INPUT or OUTPUT. Furthermore, we are asked to make an Asynchronous channel, hence we shall not use any clock or timed signal for the transmission. Rather we will use a timer ISR to transmit data synchronously so that we do not face any data contention.

In addition to all this, we are allowed to use interrupts, both external and internal so that we will also use interrupt ISR at the receiving end.

Our second biggest challenge is to write 8-bit of data on the transmission line. For this purpose, we need to have data first. We will assume any ASCII character or any 8-bit number from 0-255 and we will transmit it on the transmission/data lines. So we will first convert this character into 8-bit binary and toggle the each data line to HIGH if the binary character is 1 and to LOW if the binary character is 0. And same concept will be used in the receiving section. Where digitally read value will be 1 if the data line is at high (3.3V) and will be zero if data line is at 0V.

We are also asked to perform this experiment for both Half-Duplex and Full duplex mode. So, we will first design half duplex and then move towards the second challenge.

So, as we have highlighted all the requirements and available ingredients, now we move towards the solution.

Solution & Procedure:

Decimal & Binary interconversion:

First of all, we will declare a variable that contains the data to be transmitted. As we are making an 8-bit channel, we can only send numbers from 0-to-255. Secondly we convert that specific 8-bit integer into binary.

For this purpose we will use in-built structure in C++ called bitset, embedded using

#include <bitset> header file.

Following C++ command will convert any bit of integer data into binary:

std::bitset<8> Data(x);

Were index 0 of above variable points to LSB and index 7 points to MSB

Similarly at Receiving end, we will declare a variable with same syntax as:

std::bitset<8> RecData;

Here, we will receive MSB at 7 index, and LSB 0th index. After we have received and stored relevant bits at the specified positions or bit location in the RecData variable, now we have to convert this binary number back into decimal. For this purpose, we will use an in-built function as follows:

int RData;

RData=RecData.to_ulong();

So now, our data is sent and received in integer form. We can print it and perform any relevant operation on it.

Timer & Timer ISR:

As we know, we are asked to make Asynchronous transmission channel, but we actually have to make it synchronous. For this purpose we will use a timer hardware inside ESP 32.

We will define, a time lapse in such a way that our data is sent and received synchronously and without contention. Each time a timer is executed, an ISR will be called. All the transmission and receiving happens in this ISR.

Timer ISR is an Interrupt Service routine or in short a specified function that will be called every time the timer reaches its counting number. In this timer ISR, we first write data on the data lines and that we will define a control signal CS, that will trigger the receiver to start receiving that data. High or low state of this CS signal will determine the mode of receiver.

After the receiver has received the data, now the receiver will convert the data into integer as mentioned.

One important fact to consider is the pin mode. In unidirectional, pin Modes are specified in the setup function and all pins will be output mode. However CS pin will be permanently in OUTPUT mode.

Interrupt & Interrupt ISR:

Rather than using timers at receiver end we will use interrupt or more technically called external interrupts. For this we define a RS pin on the receiver end and attach it to an ISR where will receive the data. Whenever a CS signal changes its state from LOW to HIGH, an attached interrupt on that RS pin (in Rising edge configuration) and ISR related to that interrupt will be executed and all the data will be received.

One important fact to consider is the pin mode. In unidirectional, pin Modes are specified in the setup function and all pins will be in INPUT mode. However, RS pin will be in INPUT_PULLUP mode because it is supposed to receive the interrupt signal.

Writing & Reading Data:

For writing data we use the following Arduino function:

digitalWrite(PinNo. , State)

For binary 1, we pull the specified pin to HIGH and for Binary 0 we pull the specified pin to LOW.

Similarly for reading data we use the following Arduino function:

Bool wr= digitalRead(Pin)

Where wr is Boolean type variable. Digital Read function will return 1 if the pin is HIGH such as 3.3V and returns 0 if the pin is low such as 0V.

In this way we can write and read binary data on the wires and pins. After this we can convert the received data back into decimal, using the methods mentioned above.

Making it Bi-directional:

To make this transmission bi-directional, we will simply use another control signal CS2 at sender end and receive this signal at RS2 at the receiver end. At RS2, we will attach an interrupt that will set the pins mode to OUTPUT and send all data back to the sender ESP.

We may face many challenges in doing so such as we have to make the sending & receiving timed and synchronous with each other. We have explained in detail in the upcoming sections how we will achieve this.

Apparatus & Requirements:

Following are the requirements for this experiment:

- Two ESP-32 WROOM kit boards
- 10 male-to-male jumper wires.
- Two micro-USB cables
- Two Windows or UNIX systems with Arduino 2.0 or above installed on them
- In Arduino all ESP models should be installed and embedded.
- CP210xVCP drivers should be installed on your System.
- A 10uF & 10V Electrolyte Capacitor in case if ESP does not boot.
- Two 4.7K ohm Resistors
- Two external LEDs to check the generation of interrupts.

Wiring & Connections:

We have 15 pins on each side of the ESP boards as shown in the figure. We will use same pins on both ESP 32 for data transmission.

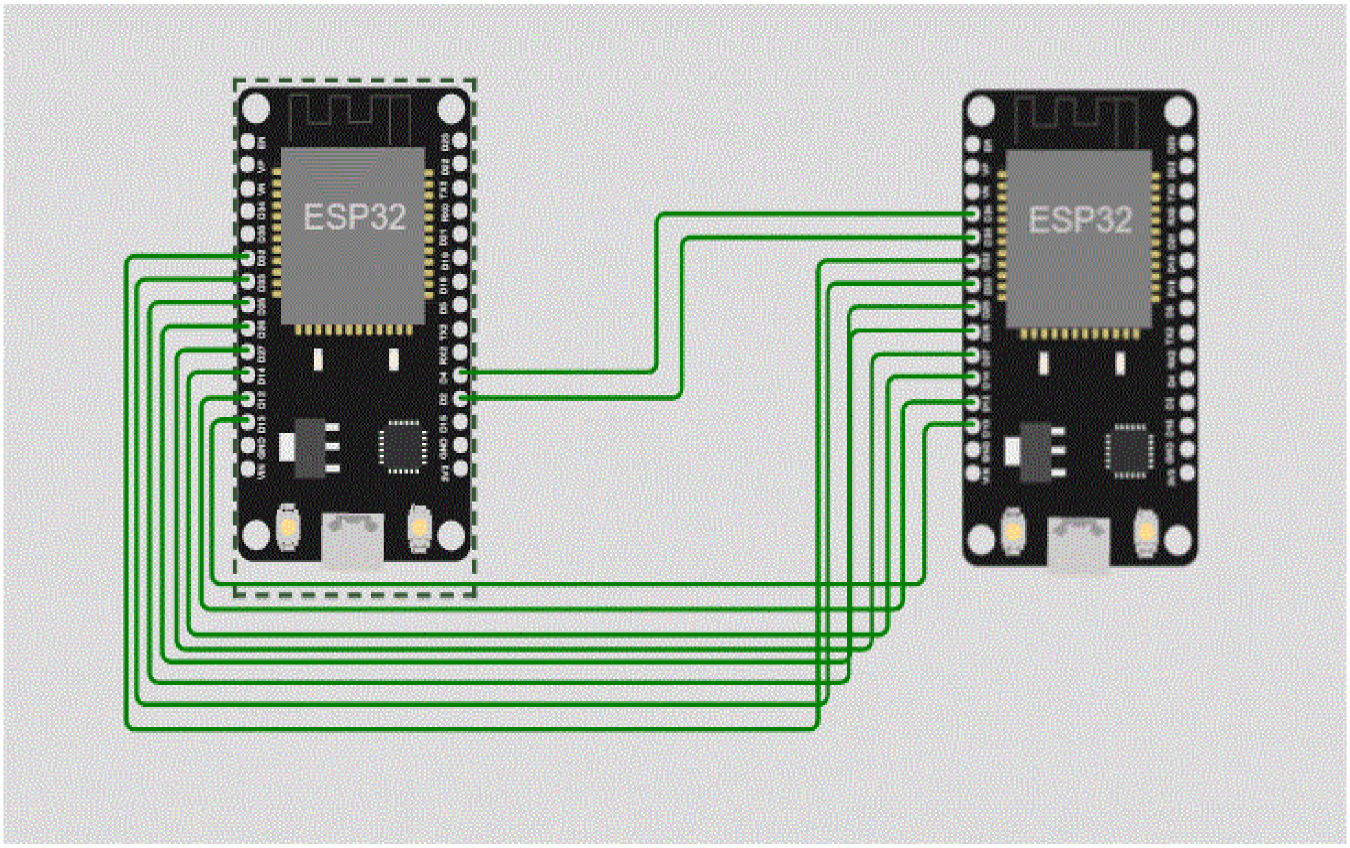
For data transmission we use 8 pins whose numbers are:

- GPIO32
- GPIO33
- GPIO25
- GPIO26
- GPIO27
- GPIO14
- GPIO12
- GPIO13

And for control signal we use GPIO2 and GPIO4 for CS1 and CS2 respectively. At receiver end we use GPIO34 and GPIO35 for RS1 and RS2 signals respectively.

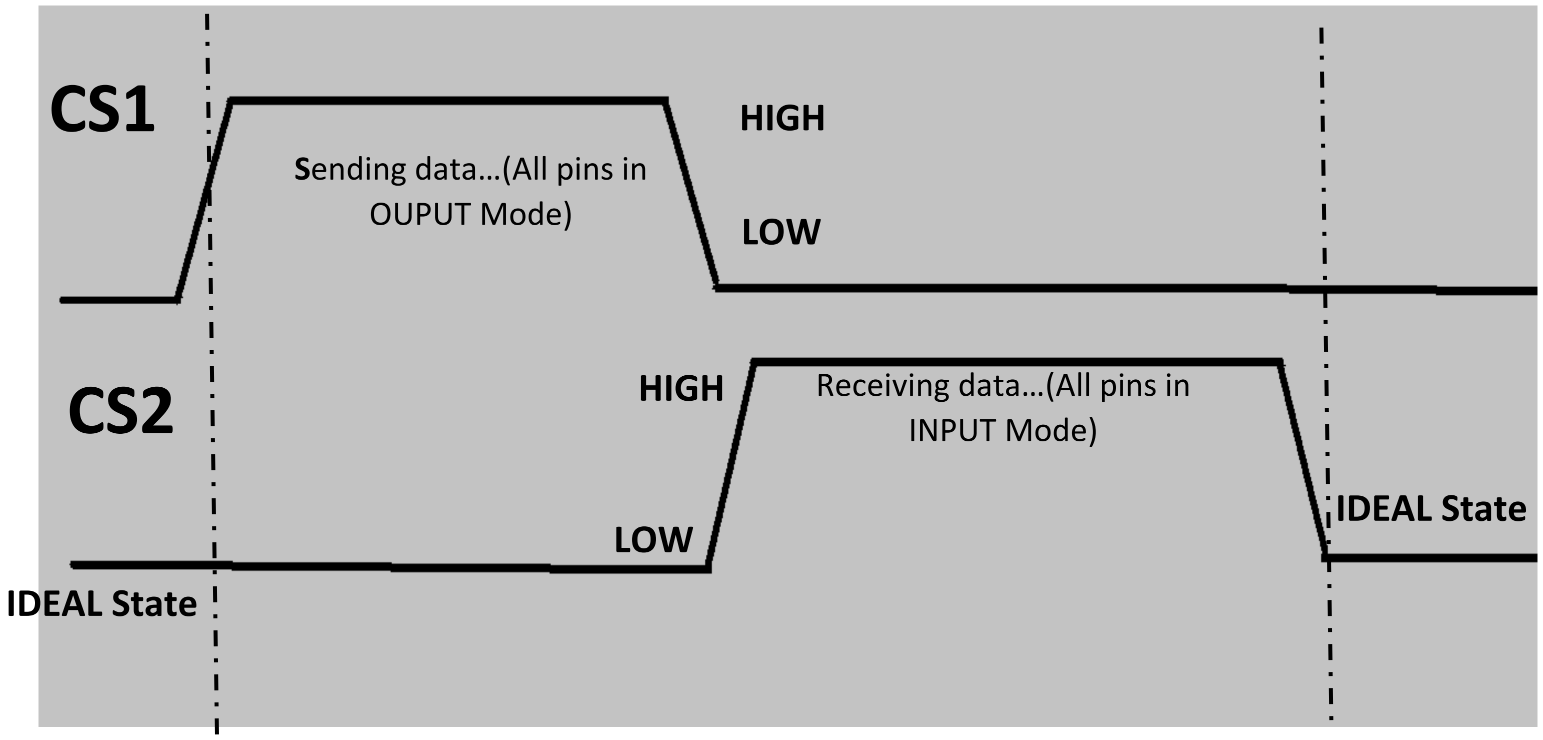
Following figure explains the connection:

On left is ESP-1 which is configured to sender by default and on right is ESP-2 which is receiver by default.

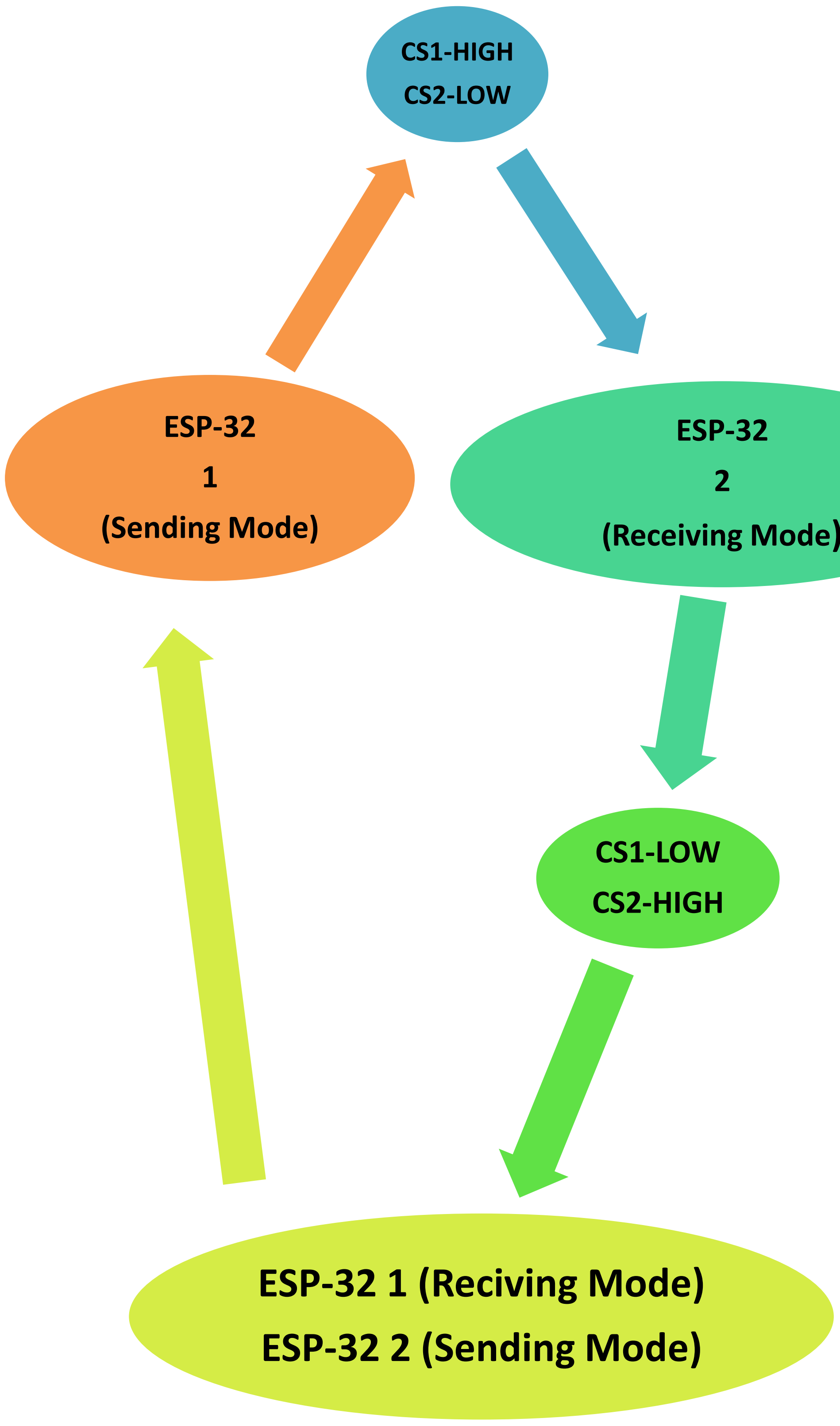


Timing & State Diagrams:

Timing Diagram:


CS1
HIGH
Sending data…(All pins in OUPUT Mode)
LOW
CS2
HIGH
Receiving data…(All pins in INPUT Mode)
LOW
IDEAL State
IDEAL State

State Diagram:



Arduino Codes:

For Uni-directional Simplex Transmission:

Sender Code:

```
#include <bitset>
#include <bits.h>
//Timer Definition
hw_timer_t *SenderT=NULL;


//PinsDefinition
int CSPin=2;  //CS Pin for transmitter
int Data=55;
std::bitset<8>  DataSet(Data);
void IRAM_ATTR Transmitter()
{
digitalWrite(CSPin , HIGH);
delay(100);
pinMode(13 , OUTPUT);   //LSB 1
delay(50);
pinMode(12 , OUTPUT);   //B 2
delay(50);
pinMode(14 , OUTPUT);   //B 4
delay(50);
pinMode(27 , OUTPUT);    //B 8
delay(50);
pinMode(26 , OUTPUT);     //B 16
delay(50);
pinMode(25 , OUTPUT);     //B 32
delay(50);
pinMode(33 , OUTPUT);     //B 64
delay(50);
pinMode(32 , OUTPUT);     //B 128
delay(50);
wr=DataSet[0];
digitalWrite(13 , wr);
wr=DataSet[1];
digitalWrite(12 , wr);
wr=DataSet[2];
digitalWrite(14 , wr);
wr=DataSet[3];
digitalWrite(27 , wr);
wr=DataSet[4];
digitalWrite(26 , wr);
wr=DataSet[5];
digitalWrite(25 , wr);
wr=DataSet[6];
digitalWrite(33 , wr);
wr=DataSet[7];
digitalWrite(32 , wr);

Serial.print("The Data is: ");
Serial.println(Data);
}

void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  pinMode(CSPin , OUTPUT);
  SenderT=timerBegin(0 , 80 , true);
  timerAttachInterrupt(SenderT , Transmitter , true);
  timerAlarmWrite(SenderT , 3000000 , true);  //3 Seconds
  timerAlarmEnable(SenderT);  //

}
```

```
#include <bitset>
#include <bits.h>
#define RS 34
int Recover;
```

```
RecData[0]=digitalRead(13);
delay(50);
RecData[1]=digitalRead(12);
delay(50);
RecData[2]=digitalRead(14);
```

Bidirectional/Half Duplex Code:

Bi-directional / Half Duplex:

Sender Code:

```
//Sender
#include <bitset>
#include <bits.h>
#define CS1 4    //to 34
#define CS2 2   //to 35
hw_timer_t *SenderT=NULL;
int i;
int x=55;
int Rec;
bool wr;
std::bitset<8> DataSet(x);
std::bitset<8> RecData;
void IRAM_ATTR Sender()
{
  Serial.println("We are pleased to print Data");
  Serial.println("---------SenderTimer Data--------");
  Serial.println("I am in Sender Timer ISR");
  pinMode(32 , OUTPUT);
  pinMode(33 , OUTPUT);
  pinMode(25 , OUTPUT);
  pinMode(26 , OUTPUT);
  pinMode(27 , OUTPUT);
  pinMode(14 , OUTPUT);
  pinMode(12 , OUTPUT);
  pinMode(13 , OUTPUT);
```

```
digitalWrite(CS1 , HIGH);  //indicates the coming of data
  wr=DataSet[0];
  digitalWrite(32 , wr);
  wr=DataSet[1];
  digitalWrite(33 , wr);
  wr=DataSet[2];
  digitalWrite(25 , wr); //LSB
  wr=DataSet[3];
  digitalWrite(26 , wr);
  wr=DataSet[4];
  digitalWrite(27 , wr);
  wr=DataSet[5];
  digitalWrite(14 , wr); //MSB
  wr=DataSet[6];
  digitalWrite(12 , wr);
  wr=DataSet[7];
  digitalWrite(13 , wr); //MSB
  Serial.println("Writtern Data is : ");
  Serial.println("Written Bits");
  for (i=0;i<8;i++)
  Serial.println(DataSet[i]);
Serial.println("The Data is :");
Serial.println(x);
Serial.println("--------------------");
delay(500);
  digitalWrite(CS1 , HIGH);  //indicates the coming of data
  delay(250);
  digitalWrite(CS1 , LOW);
  delay(250);
//Receiving
   delay(500);
  digitalWrite(CS2 , HIGH);  //indicates the coming of data
  delay(250);
```

```
Serial.println("************----Receiver Timer Data----------*******");
 pinMode(32 , INPUT);
 pinMode(33 , INPUT);
 pinMode(25 , INPUT);
 pinMode(26 , INPUT);
 pinMode(27 , INPUT);
 pinMode(14 , INPUT);
 pinMode(12 , INPUT);
 pinMode(13 , INPUT);
 RecData[0]=digitalRead(32);
 delay(150);
 RecData[1]=digitalRead(33);
 delay(150);
 RecData[2]=digitalRead(25);
 delay(150);
 RecData[3]=digitalRead(26);
 delay(150);
 RecData[4]=digitalRead(27);
 delay(150);
 RecData[5]=digitalRead(14);
 delay(150);
 RecData[6]=digitalRead(12);
 delay(150);
 RecData[7]=digitalRead(13);
 delay(150);
```

```
Serial.println("Recover Bit is :");
for(i=0; i<8;i++)
{
Serial.println(RecData[i]);
}

Rec=RecData.to_ullong();
Serial.print("The Recieved data is : ");
Serial.println(Rec);
Serial.println(" ");
  digitalWrite(CS1 , LOW);
  delay(250);
}
void setup() {
  Serial.begin(115200);
  pinMode(CS1 , OUTPUT);
  pinMode(CS2 , OUTPUT);
  SenderT=timerBegin(0,80 , true);
  timerAttachInterrupt(SenderT , Sender , true);
  timerAlarmWrite(SenderT , 5000000 , true);
  timerAlarmEnable(SenderT);
}

void loop() {
  // put your main code here, to run repeatedly:
}
```

Receiver Code:

```
//Receiver
#include <bitset>
#include <bits.h>
#define RS1 34  //to CS1 4
#define RS2 35 //to CS2 2
int i=0;
int x2=4;
int Rec2;
bool wr;
std::bitset<8> RecData2;
std::bitset<8> DataSet2(x2);

void IRAM_ATTR InterruptSender()
{
  //Serial.println("*****----NOW Sending Back Data---*****");
  //Serial.println("We are pleased to send Data from Interrupt");
  //Serial.println("I am in Sender Interrupt ISR");
  Serial.println("***********----Sender Interrupt Data----------*******");
  pinMode(32 , OUTPUT);
  pinMode(33 , OUTPUT);
  pinMode(25 , OUTPUT);
  pinMode(26 , OUTPUT);
  pinMode(27 , OUTPUT);
  pinMode(14 , OUTPUT);
  pinMode(12 , OUTPUT);
  pinMode(13 , OUTPUT);
```

```
void IRAM_ATTR InterruptReciever()
{
  delay(250);
  //Serial.println("We are pleased to Recieved  Data");
  //Serial.println("I am in Reciver Interrupt ISR");
  Serial.println("************----Reciever Interrupt Data----------*******");
   pinMode(32 , INPUT);
  pinMode(33 , INPUT);
  pinMode(25 , INPUT);
  pinMode(26 , INPUT);
  pinMode(27 , INPUT);
  pinMode(14 , INPUT);
  pinMode(12 , INPUT);
  pinMode(13 , INPUT);
  RecData2[0]=digitalRead(32);
  delay(150);
  RecData2[1]=digitalRead(33);
  delay(150);
  RecData2[2]=digitalRead(25);
  delay(150);
  RecData2[3]=digitalRead(26);
  delay(150);
  RecData2[4]=digitalRead(27);
  delay(150);
  RecData2[5]=digitalRead(14);
  delay(150);
  RecData2[6]=digitalRead(12);
  delay(150);
  RecData2[7]=digitalRead(13);
  delay(150);
wr=DataSet2[0];
 digitalWrite(32 , wr);
 wr=DataSet2[1];
 digitalWrite(33 , wr);
 wr=DataSet2[2];
 digitalWrite(25 , wr); //LSB
 wr=DataSet2[3];
 digitalWrite(26 , wr);
 wr=DataSet2[4];
 digitalWrite(27 , wr);
 wr=DataSet2[5];
 digitalWrite(14 , wr); //LSB
 wr=DataSet2[6];
 digitalWrite(12 , wr);
 wr=DataSet2[7];
 digitalWrite(13 , wr);




 Serial.println("Written Bits in External Interrupt are : ");
 for (i=0;i<8;i++)
 Serial.println(DataSet2[i]);
 Serial.print("The Sent Data is :");
 Serial.println(x2);



}
```

```
Serial.println("Recover Interrupt Bit is :");
for(i=0; i<3;i++)
{
Serial.println(RecData2[i]);
}

Rec2=RecData2.to_ullong();
Serial.print("The Recieved data is : ");
Serial.println(Rec2);
Serial.println(" ");

}



void setup() {
  // put your setup code here, to run once:
  pinMode(RS1 , INPUT_PULLUP);
  pinMode(RS2 , INPUT_PULLUP);
 attachInterrupt(RS1 , InterruptReciever , RISING);
  attachInterrupt(RS2 , InterruptSender , RISING);

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Ended here