EEL3020: Digital Systems Project

On

Implement the I2C (Inter-Integrated Circuit) protocol in bare-metal programming for the STM32F412 microcontroller

by

Team Members:

Priyam Gaurav (B21EE092)
Rahul Gurjar (B21EE055)

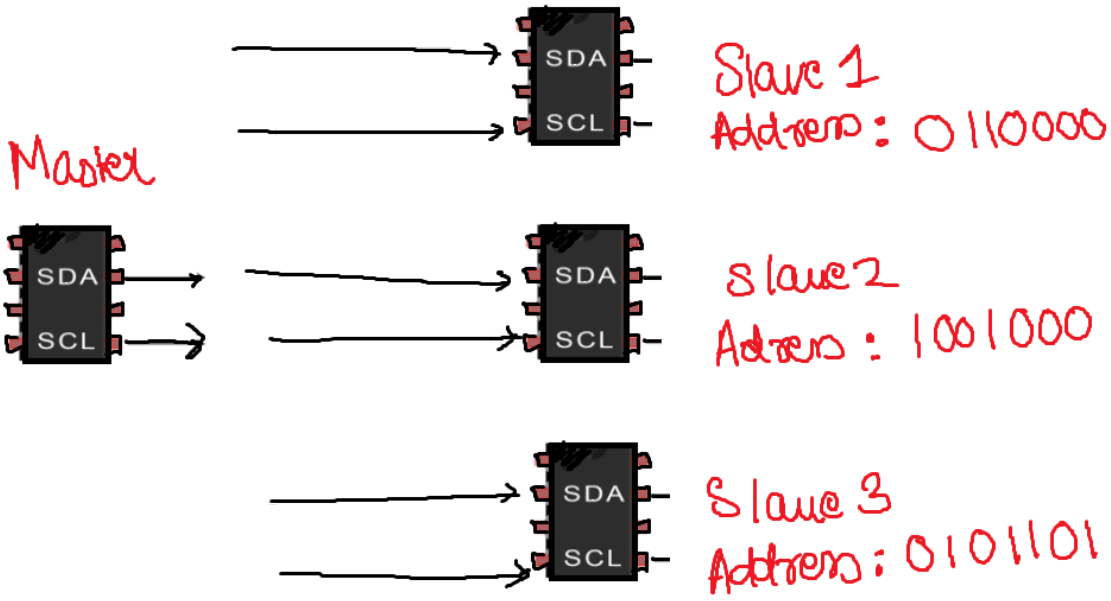
➢Objective:

In this project, we have implemented the I2C (Inter-Integrated Circuit) protocol in bare-metal programming for the STM32F412 microcontroller without utilizing any external libraries. Then the differences between the I2C protocol and other communication protocols is discussed. Also, we have highlighted the application areas where the I2C protocol is particularly advantageous.

➢Theory:

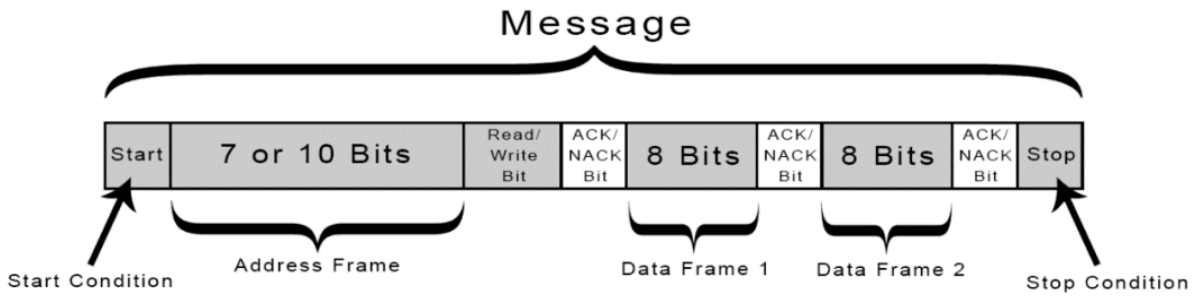
I2c (Inter-Integrated Circuit) is a synchronous protocol, with a master and multiple slaves. It is a single ended and serial communication, it is widely used for low speed peripherals, short distance connection on board.

A simple I2C circuit consists of a Master device and a slave device. The master device generally communicates through the slave device through two lines. The SDA and SCL. SDA stands for Serial Data and SCL stands for Serial Clock, they both are bidirectional synchronous lines. In the case of I2C we generally do serial communication. We transfer bit by bit. A master device can also be connected to multiple slave devices, these slave devices are distinguished through the slave address. The slaves are identified using these addresses, used by the master to identify which slave it wants to talk to.



Steps for I2C Communication :

In I2C communication, the master controls the clock. The master sends data, which are broken in frames or packets. Each packet looks like this.



The message packet includes the start condition, the slave address followed by a read or write bit,an acknowledge or negative/not acknowledgement. It is followed by bits of data, each ending with an ACK/NACK and in the end a stop condition.
For the start condition as the SCL line is high, the SDA switches from high to low.
It is followed by a generally a 7 bit slave address, through which the master decides which slave it wants to communicate with incase of multiple slaves present.It is followed by a Read/Write which is sent with the slave address, to determine what kind of operation the master wants to perform. The bit '0' represents the master wants to write, while '1' represents the master device wants to read.
It is followed by either a NACK/ACK for an acknowledgement, while pulling the SDA low and an additional clock pulse is generated on the SCL.
It is followed by our 8 bits of data, which the master either wants to write or read from the slave, each 8 bits of data is followed by an ACK/NACK.
In the end, the communication ends with a stop bit, which is given while the SCL switches from low to high line, the SDA switches from low to high.
These are the steps of how exactly an I2C communication protocol works, how exactly a master communicates with the different slave devices.
Compared to other communication protocols like SPI and CAN. Let us see how exactly SPI works,in the case of SPI we have 4 pins through which a master and a slave have a full duplex communication. We have a Serial Clock, which basically carries the clock signal generated by the master to different slaves through a serial bus.We have a MOSI i.e Master Output Slave Input, it carries the master output to the slave input on the serial bus, also we have MISO i.e Master Input Slave output, which carries the output data from the slave to the master. In the end we have a SS i.e Slave Select , as there are multiple slaves we can use this to select the different slaves we want to communicate to.
If we observe carefully SPI and I2C communication have some similarities, but differences too. In I2c slaves are identified from addresses while in SPI it is done through Select line, making SPI preferable for fast communication, while I2C is well suited for protocols with many devices and also offering flow and error control. Now let us compare it to CAN. CAN is an asynchronous communication protocol while our I2C is a synchronous communication protocol, CAN is a

controller area network which allows communication between internal systems without a central computer, while in I2C communication, we have a master which facilitates communication, but intercommunication b/w slaves are not allowed directly.

➢Code Explanation and Analysis:

Firstly we are using the STM32F429ZG for this implementation. The program is loaded on the discovery board and the PIN 6 of PORT B is connected to the SCL of LED and PIN 7 of PORT B is connected to the SDA of LED. Also other than this the VCC of LED is supplied with the 5V and its GND is connected to the GND of the microcontroller board.
The code explanation is as follows:

```
int main(void){
  // ON CLOCK FOR PORT B
  RCC->AHB1ENR |= 0x2;

  // SET MODER FOR PIN PB6 - SCL AND PB7 - SDA AS AF MODE
  GPIOB->MODER |= (2<<(6*2));
  GPIOB->MODER |= (2<<(7*2));

  // SELECT ALTERNATE FUNCTION (AF4) AS I2C
  GPIOB->AFR[0] |= 4<<(6*4);
  GPIOB->AFR[0] |= 4<<(7*4);

  // SETTING AT HIGH SPEED
  GPIOB->OSPEEDR |= (2<<(6*2));
  GPIOB->OSPEEDR |= (2<<(7*2));

  // SETTING AS PULL UP
  GPIOB->PUPDR |= (1<<(6*2));
  GPIOB->PUPDR |= (1<<(7*2));

  // SETTING AS OPEN DRAIN
  GPIOB->OTYPER |=(1<<6) ;
  GPIOB->OTYPER |=(1<<7);
```

In this code for I2C Protocol based communication firstly we have enabled the clock for Port B. This is done because we are using the Port B PIN 6 and PIN 7 as the SCL and SDA of LEDs respectively. The mode for these pins is set to the alternate function mode. The open drain and pull up are set because it is a requirement for the I2C to work. Also high speed is reliable for I2C communication.

```
// Enable I2C1 clock
RCC->APB1ENR |= 1UL<<21;

// RESETTING I2C BY SWSRT AT 15
I2C1->CR1 |= 1<<15;
I2C1->CR1 &= ~(1<<15);

// SET I2C CLK AT 16MHZ BY FREQ[5:0] VALUE
I2C1->CR2 |= 16<<0;

// Needs to be set high by software for I2C
I2C1->OAR1 |= 1<<14;

// SET SC1 AT 100KHZ BY CCR[11:0] VALUE
I2C1->CCR |= 0x50;

// SET RISE TIME AS 1000NS BY TRISE[5:0] VALUE
I2C1->TRISE |= 17;

// ENABLE I2C1
I2C1->CR1 |= 1;
```

Then here we enable the clock for the I2C1 from RCC->AP1ENR. Then we reset the I2C using the SWSRT at the PIN 15 of I2C1->CR1. Then the clock of I2C is set at 16MHz using the control register 2 (I2C1->CR2). Thenwe set the ADDMODE bit in the I2C1 own address register 1 (I2C1->OAR1), which is required for I2C communication. Also then we set the I2C clock control register (CCR) to configure the I2C clock speed to 100 kHz, which is one of the common speeds for I2C communication with LCD. Then we set the rise time to 1000ns and then using the control register 1 (CR1) bit 1 we enable the I2C1.

```
  TIM4_Delay();
  Cmd_WRITE(LCD_ADDRESS,0x30);
  TIM4_Delay();
  //SETTING OUR LED TO 4 BIT MODE
  Cmd_WRITE(LCD_ADDRESS,0x20);
  TIM4_Delay();

  Write_Data_Function(LCD_ADDRESS, 0x50); // MAPPING VALUES IN LCD TO SHOW 'P' USING ROM PATTERN
  Write_Data_Function(LCD_ADDRESS, 0x52); // MAPPING VALUES IN LCD TO SHOW 'R' USING ROM PATTERN
  Write_Data_Function(LCD_ADDRESS, 0x49); // MAPPING VALUES IN LCD TO SHOW 'I' USING ROM PATTERN
  Write_Data_Function(LCD_ADDRESS, 0x59); // MAPPING VALUES IN LCD TO SHOW 'Y' USING ROM PATTERN
  Write_Data_Function(LCD_ADDRESS, 0x41); // MAPPING VALUES IN LCD TO SHOW 'A' USING ROM PATTERN
  Write_Data_Function(LCD_ADDRESS, 0x4D); // MAPPING VALUES IN LCD TO SHOW 'M' USING ROM PATTERN
  TIM4_Delay();
  while(1){}
}
```

Then in these lines we firstly wait for a certain time by introducing delay. Then we have sent commands to the LCD module using the Cmd_WRITE function. The first command 0x30 is a function set command that prepares the LCD for further initialization. The second command 0x20 sets the LCD to 4-bit mode since four of the bits for the LCD are carrying the data, the rest four carry the information for lighting the backlight, enable bit, mode bit(either read or write) and one bit is for distinguishing between data and instruction .

```
void TIM4_Delay(void){
  RCC->APB1ENR |= 0x4;
  TIM4->PSC = 15999;
  TIM4->ARR = 100;
  TIM4->CNT = 0;
  TIM4->CR1 |= 1;


  while(!(TIM4->SR & 0x1)){}
  TIM4->SR &= ~(0x1);
}
```

This function is for introducing the delay to the points wherever it is being called. To generate this delay it uses the Timer where we set the prescaler resistor to 15000 hence scaling the 16MHz system clock to 1kHz . Setting ARR to 100 makes the delay go to 100ms. In the control register we are enabling the delay and then using the status resistor we get the instant wherever the one cycle of delay is completed.

```
void I2C_Write(uint8_t var){

  // CHECKING TXE BIT FOR DATA REGISTER EMMPTY OR NOT
  while(!(I2C1->SR1 & (1<<7))){}

  // DATA TO BE WRITTEN
  I2C1->DR = var;

  // CHECK BTF BIT TRANSFER FINISHED
  while(!(I2C1->SR1 & (1<<2))){}
}
```

In this function we check the status of the Transmit Data Register Empty (TXE) bit in the I2C Status Register 1 (SR1). This bit is set when the data register (DR) is empty and ready to receive new data. The loop waits until the TXE bit is set, or we can say that the data register is ready to accept new data. Then we write the data in variable var to the I2C Data Register (DR). Then in the loop we check the status of the Byte Transfer Finished (BTF) bit in the I2C Status Register 1 (SR1). The BTF bit is set when the data byte has been transmitted, and the I2C peripheral is ready for new data. The BTF bit is set indicates that the data byte has been successfully transmitted. This function is responsible for transmission of individual data bytes.

```
void I2C_Start(void){
  // ENALBE ACK BIT
  I2C1->CR1 |= 1<<10;

  // SET START BIT TO START COMMUNICATION
  I2C1->CR1 |= 1UL<<8;

  // TO MAKE SURE COMMUNICATION HAS STARTED
  while(!(I2C1->SR1 & 0x1)){}
}
```

In this we set the acknowledgement bit and the start bit in the control resistor. The start bit is set to start the communication. To check if the communication has been successfully started we check the Start Bit (SB) flag in the I2C Status Register 1 (SR1). This function is essential to be called before each data transmission.

```
void Cmd_WRITE(uint8_t Device_Addr, uint8_t data){
  //IN D1 AND D2 WE ARE STORING THE UPPER 4 BITS OF DATA
  //WE ARE SETTING EN=1,BACKLED IS ON
  uint8_t d1 = (data & 0xF0) | 0x0C;
  //HERE EN = 0
  uint8_t d2 = (data & 0xF0) | 0x08;

  //IN D3 AND D4 WE ARE STORING THE UPPER 4 BITS OF DATA
  //WE ARE SETTING EN=1,BACKLED IS ON
  uint8_t d3 = ((data << 4) & 0xF0)| 0x0C;
  //HERE EN = 0
  uint8_t d4 = ((data << 4) & 0xF0)| 0x08;

  I2C_Start();
  //ADDRESS TO BE SENT
  I2C1->DR = Device_Addr;

  // ADDRESS TRANSMITTED ADDR BIT
  while(!(I2C1->SR1 & 0x2)){}

  // CLEARING THE ADDR BIT
  clc = (I2C1->SR1 | I2C1->SR2);

  I2C_Write(d1);
  I2C_Write(d2);
  I2C_Write(d3);
  I2C_Write(d4);

  // STOP COMMUNCATION
  I2C1->CR1 |= 1<<9;
}
```

Here in this function, four bytes (d1, d2, d3, d4) are prepared to send the command data to the LCD module. The LCD operates in a 4-bit mode, where data is sent in two halves (upper 4 bits and lower 4 bits). The control bits (0x0C and 0x08) are combined with the data bits to indicate the state of the Enable (EN) and Backlight (BACKLED) signals for each half of the data transfer. To understand this the configuration of the pins are as follows:

1. P0 - Register Select -- distinguishes between instruction and data being sent, 0 indicates the data
2. P1 - Read/Write -- direction of data transfer, 0 indicates data is transfered to LED
3. P2 - Enable
4. P3 - Back Light
5. P4 - Data Pin1
6. P5 - Data Pin2
7. P6 - Data Pin3
8. P7 - Data Pin4

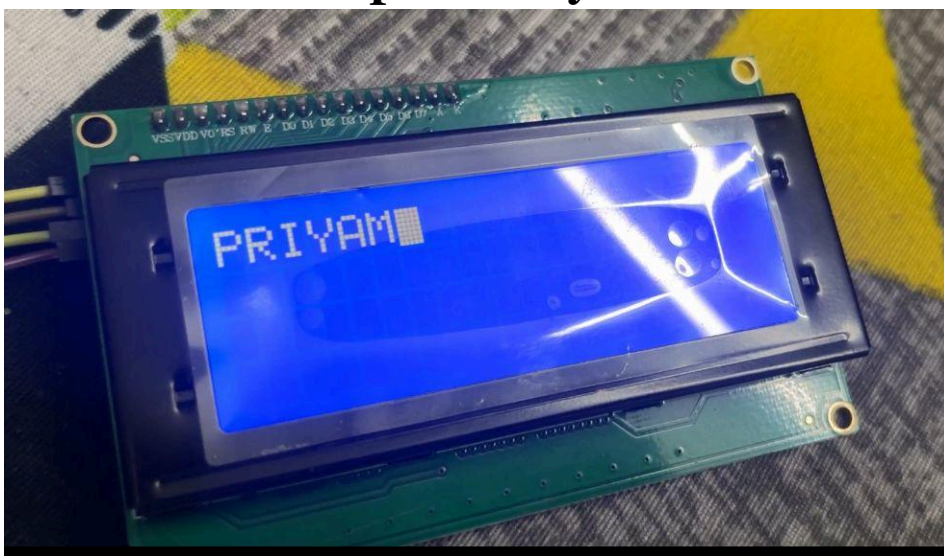
Then we call the I2C_Start function to initialize the communication. Then, the slave device address (Device_Addr) is written to the I2C Data Register (DR), which will be transmitted after the start condition. The function then waits for the ADDR bit to be set in the I2C Status Register 1 (SR1), indicating that the slave device address has been transmitted. After that, the function clears the ADDR flag. The four bytes (d1, d2, d3, d4) prepared earlier are transmitted over the I2C bus using the I2C_Write function. Then to stop the communication the Stop (STOP) bit in the I2C Control Register 1 (CR1) is set.

```
void Write_Data_Function(uint8_t Device_Addr, uint8_t data){

  //IN D1 AND D2 WE ARE STORING THE UPPER 4 BITS OF DATA
  //WE ARE SETTING EN=1,BACKLED IS ON
  uint8_t d1 = (data & 0xF0) | 0x0D;
  //HERE EN = 0
  uint8_t d2 = (data & 0xF0) | 0x09;

  //IN D3 AND D4 WE ARE STORING THE UPPER 4 BITS OF DATA
  //WE ARE SETTING EN=1,BACKLED IS ON
  uint8_t d3 = ((data << 4) & 0xF0)| 0x0D;
  //HERE EN = 0
  uint8_t d4 = ((data << 4) & 0xF0)| 0x09;

  // ENALBE ACK BIT
  I2C1->CR1 |= 1<<10;

  // SET START BIT TO START COMMUNICATION
  I2C1->CR1 |= 1UL<<8;

  // TO MAKE SURE COMMUNICATION HAS STARTED
  while(!(I2C1->SR1 & 0x1)){}

  //ADDRESS TO BE SENT
  I2C1->DR = Device_Addr;

  // ADDRESS TRANSMITTED ADDR BIT
  while(!(I2C1->SR1 & 0x2)){}

  // CLEARING THE ADDR BIT
  clc = (I2C1->SR1 | I2C1->SR2);

  I2C_Write(d1);
  I2C_Write(d2);
  I2C_Write(d3);
  I2C_Write(d4);

  // STOP COMMUNCATION
  I2C1->CR1 |= 1<<9;
}
```

This function is similar to the command write function written above but here the actual data is transferred. However, the control bits (0x0D and 0x09) are different from those used in Cmd_WRITE. These control bits indicate that the data being sent is character data rather than a command. Then we start the I2C communication and wait until we make sure that the communication has been started using the status register. Then similar to Cmd_WRITE the data is written to the I2C Data Register (DR). The function then waits for the ADDR bit to be set in the I2C Status Register 1 (SR1), indicating that the slave device address has been transmitted. After that, the function clears the ADDR flag. The four bytes

(d1, d2, d3, d4) prepared earlier are transmitted over the I2C bus using the I2C_Write function. Then to stop the communication the Stop (STOP) bit in the I2C Control Register 1 (CR1) is set.

➢Results and Output Analysis:



From our result we can confirm that our communication is working, as through our code we are sending the string 'PRIYAM' through our I2C communication protocol which is being displayed on the LCD display that we have used as our Slave device here.

➢Conclusion:

From our implementation of I2C communication protocol from scratch, we can conclude the things we discussed in theory. I2c is a half -duplex communication protocol. Generally it has a single master which can be connected to multiple slaves.The communication in I2C is serial, as we only send certain bits of data at a single instant. For our implementation we have used a LCD display as a slave device, as discussed for I2C communication we implemented the same steps for our communication. Through our result we could verify that our I2C communication was working.

➢References:

1.I2C Resource pdf (https://classroom.google.com/u/0/c/NjQ2ODI0ODI2NDk0/a/NjYwODI4NzAwMTA4/details)
2.https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/
3.https://medium.com/@jchrysaphiades/stm32-bare-metal-programming-i2c-4b1f9ed66f53
4.https://www.totalphase.com/blog/2021/07/i2c-vs-spi-protocol-analyzers-differences-and-similarities/