Experiment-12

Name :- Aryan Langhanoja
Enrollment No :- 92200133030

Aim: Hands-on experimentation of I2C Programming with ATMEGA32 in C .

Objectives: After successfully completion of this experiment students will be able to,

- Use C language for ATMega32 microcontroller programming on AVRStudio.
- Experiment with I2C programming with ATMega32 on ATMega32 AVR Development Board.

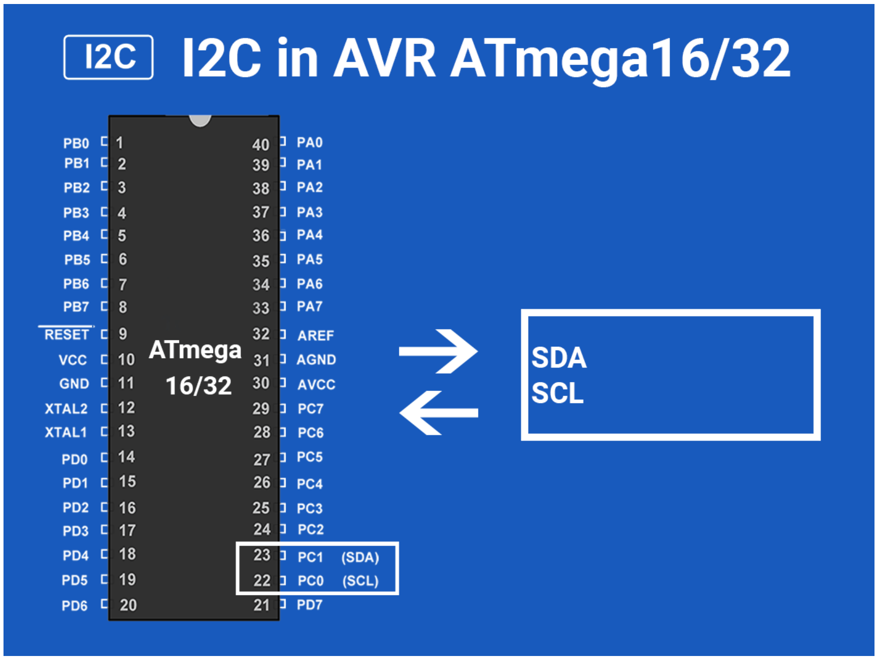
Equipment required:

- Windows7 or later based host computer
- ATMega32 Development board
- USBasp Programmer
- Jumper Wires
- Peripherals

Software required:

- AVR Studio7 installation setup
- USBasp driver installation setup

Theory:



I2C (Inter-Integrated Circuit) is a serial bus interface connection protocol. It is also called TWI (two-wire interface) since it uses only two wires for communication, that two wires called SDA (serial data) and SCL (serial clock). AVR-based ATmega16/ATmega32 has a TWI module made up of several submodules as shown in the figure.

I2C works in two modes namely,

- Master mode
- Slave mode

Let see registers in the ATmega16/32 I2C module

TWBR: TWI Bit Rate Register

TWI bit rate register used in generating SCL frequency while operating in master mode

7	6	5	4	3	2	1	0	
TWBR7	TWBR6	TWBR5	TWBR4	TWBR3	TWBR2	TWBR1	TWBR0	TWBR Register

TWCR: TWI Control Register

TWI control resistor used to control events of all I2C communication.

7	6	5	4	3	2	1	0	
TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	TWCR Register

Bit 7 – TWINT: TWI interrupt

- This bit gets set whenever TWI completes its current event (like start, stop, transmit, receive, etc).

- While I-bit in SREG and TWIE bit in TWCR is enabled then TWI interrupt vector called whenever TWI interrupt occur.
- TWI interrupt flag must be cleared by software by writing a logical one to it. This bit is not automatically cleared by hardware.

Bit 6 – TWEA: TWI enable acknowledgment bit

- This is TWI acknowledgment enable bit, it is set in receiver mode to generate acknowledgment and cleared in transmit mode.

Bit 5 – TWSTA: TWI START condition bit

- The master device set this bit to generate START condition by monitoring free bus status to take control over the TWI bus.

Bit 4 – TWSTO: TWI STOP condition bit

- The master device set this bit to generate STOP condition to leave control over the TWI bus.

Bit 3 – TWWC: TWI write collision

- This bit gets set when writing to the TWDR register before the current transmission not complete i.e. TWINT is low.

Bit 2 – TWEN: TWI enable bit

- This bit set to enables the TWI interface in the device and takes control over the I/O pins.

Bit 1 - Reserved

Bit 0 – TWIE: TWI interrupt enable

- This bit is used to enable TWI to interrupt routine while the I-bit of SREG is set as long as the TWINT flag is high.

TWSR: TWI Status Register

7	6	5	4	3	2	1	0	
TWS7	TWS6	TWS5	TWS4	TWS3	—	TWPS1	TWPS0	TWSR Register

Bit 7:Bit 3 - TWS7: TWS3: TWI status bits

- TWI status bits shows the status of TWI control and bus

Bit 1:0 - TWPS1:TWPS0: TWI pre-scaler bits

- TWI pre-scaler bits used in bit rate formula to calculate SCL frequency

TWPS1	TWPS0	Exponent	Pre-scaler value
0	0	0	1
0	1	1	4
1	0	2	16
1	1	3	64

TWDR: TWI Data Register

- TWDR contains data to be transmitted or received.
- It’s not writable while TWI is in process of shifting a byte.
- The data remains stable as long as TWINT is set.

7	6	5	4	3	2	1	0	
								TWDR Register

TWAR: TWI Address Register

- TWAR register contains the address of the TWI unit in slave mode.
- It is mostly used in the multi-master system.

7	6	5	4	3	2	1	0	
TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	TWAR Register

Bit 7:1 - TWA6: TWA0: TWI address bits

- TWI address bits contain TWI 7-bit address with which it can called by other masters in slave mode.

Bit 0 – TWGCE: TWI general call enable bit

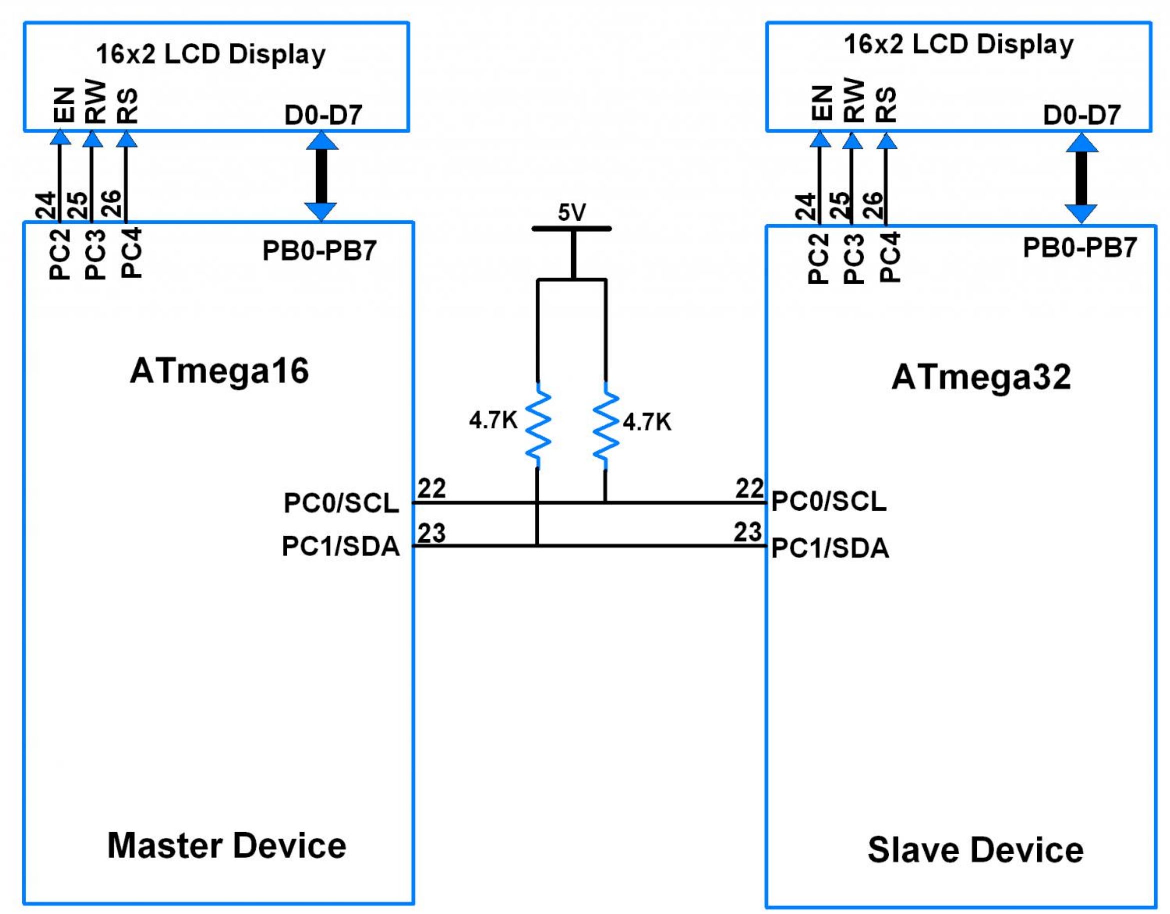
- TWI general call enable bit when set it enables recognition of general call over the TWI bus

There are four transmission modes in I2C in which the I2C device works.

- When the device is Master it works in MT and MR transmission modes.
- And when the device is Slave it works in ST and SR transmission modes.

SR No.	Transmission mode	Operation
1	Master Transmitter (MT)	Master device writes data to SDA.
2	Master Receiver (MR)	Master device read data from SDA.
3	Slave Transmitter (ST)	Slave device writes data to SDA.
4	Slave Receiver (SR)	Slave device read data from SDA.

I2C Interfacing Diagram:



Programming steps for Write operation:

1. Initialize I2C.
2. Generate START condition.
3. Send the Slave device to write address (SLA+W) and check for acknowledgment.
4. Write memory location addresses for memory devices to which we want to write.
5. Write data till the last byte.
6. Generate a STOP condition.

Programming steps for reading operation:

1. Initialize I2C.
2. Generate START condition.
3. Write device Write address (SLA+W) and check for acknowledgment.
4. Write a memory location address for memory devices.
5. Generate REPEATED START condition.
6. Read data and return acknowledgment.
7. Return Not acknowledgment for the last byte.
8. Generate a STOP condition.

ATmega32 I2C Master Program:

```
#define F_CPU 8000000UL             /* Define CPU clock Frequency 8MHz */
#include <avr/io.h>             /* Include AVR std. library file */
#include <util/delay.h>                 /* Include Delay header file */
#include <string.h>             /* Include string header file */
#include "I2C_Master_H_file.h"      /* Include I2C header file */
#include "LCD_16x2_H_file.h"        /* Include LCD header file */

#define EEPROM_Write_Addess             0xA0
#define EEPROM_Read_Addess              0xA1

int main(void)
{
   char array[10] = "test";      /* Declare array to be print */
   LCD_Init();                   /* Initialize LCD */
   I2C_Init();                   /* Initialize I2C */
   I2C_Start(EEPROM_Write_Addess);/* Start I2C with device write address */
   I2C_Write(0x00);              /* Write start memory address for data write */
   for (int i = 0; i<strlen(array); i++)/* Write array data */
        {
                I2C_Write(array[i]);
        }
   I2C_Stop();                   /* Stop I2C */
   _delay_ms(10);
   I2C_Start(EEPROM_Write_Addess);/* Start I2C with device write address */
   I2C_Write(0x00);              /* Write start memory address */
   I2C_Repeated_Start(EEPROM_Read_Addess);/* Repeat start I2C SLA+R */
   for (int i = 0; i<strlen(array); i++)/* Read data with acknowledgment */
        {
                LCD_Char(I2C_Read_Ack());
        }
   I2C_Read_Nack();              /* Read flush data with nack */
   I2C_Stop();                   /* Stop I2C */
   return 0;

}
```

ATMEGA32 Slave Program:

```
#define F_CPU 8000000UL             /* Define CPU clock Frequency 8MHz */
#include <avr/io.h>                 /* Include AVR std. library file */
#include <util/delay.h>             /* Include inbuilt defined Delay header file */
#include <stdio.h>                  /* Include standard I/O header file */
#include <string.h>                 /* Include string header file */
```

```
#include "LCD_16x2_H_file.h"        /* Include LCD header file */
#include "I2C_Slave_H_File.h"       /* Include I2C slave header file */

#define Slave_Address               0x20

int main(void)
{
    char buffer[10];
    int8_t count = 0;

    LCD_Init();
    I2C_Slave_Init(Slave_Address);

    LCD_String_xy(1, 0, "Slave Device");

    while (1)
     {
        switch(I2C_Slave_Listen())                  /* Check for SLA+W or SLA+R */
         {
           case 0:
                {
                 LCD_String_xy(2, 0, "Receiving :       ");
                 do
                 {
                   sprintf(buffer, "%d    ", count);
                   LCD_String_xy(2, 13, buffer);
                   count = I2C_Slave_Receive();       /* Receive data byte*/
                 } while (count != -1);                /* Receive until STOP/REPEATED
START */
                 count = 0;
                 break;
                }
           case 1:
                {
                 int8_t Ack_status;
                 LCD_String_xy(2, 0, "Sending :       ");
                 do
                 {
                   Ack_status = I2C_Slave_Transmit(count);/* Send data byte */
                   sprintf(buffer, "%d    ",count);
                   LCD_String_xy(2, 13, buffer);
                   count++;
```

```
                    } while (Ack_status == 0);                              /* Send until Ack is receive
*/
                    break;
                  }
            default:
                  break;
          }
    }

}
```

Circuits :