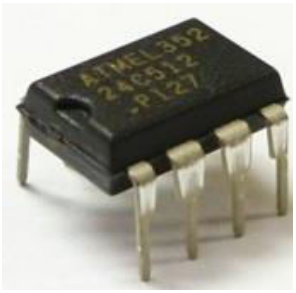
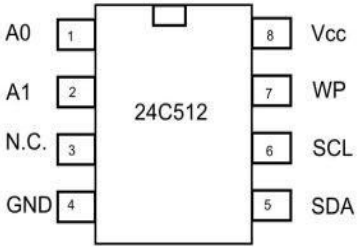
Interfacing between Arduino UNO and 24C512 EEPROM using TWI Bus

10.1 Interfacing 24C512 EEPROM with Arduino UNO using TWI Bus





Pin Name	Function
A0–A1	Address Inputs
SDA	Serial Data
SCL	Serial Clock Input
WP	Write Protect
NC	No Connect

Figure-10.1: *Pictorial View, Pin Diagram, and Pin Configuration (Google and Atmel data sheet)*

10.1.1 Introduction

24C512 is a TWI Bus compatible serial EEPROM. It exchanges data with a host processor bit-by-bit in a serial fashion. There are 65536 locations inside the memory chip; each location can hold 8-bit data. The addresses of these memory locations are 0000h, 0001, …, FFFEH, FFFFh. To write a data byte into a memory location, the following steps are executed: (a) the desired location is selected, (b) the previous content is erased, (c) data is placed, and (d) then the writing is completed within a self-timed scheme of about 5 ms.

The chip can be operated from a 2.5V to 5V power supply (Pin-8); Pin-4 is the GND point. The device has a 7-bit TWI Bus address (called device address), and it is '1 0 1 0 0 A1 A0'. A1 and A0 pins are usually connected with known logic values (LH or LL). Thus, by assigning different logic values to A1 and A0, we can simultaneously place four memory chips in the TWI Bus.

WP (Write Protect) signal (Pin-7) protects the chip from being written when it is tied to Vcc; but, reading process remains allowed. When the WP-pin connected to GND-point, both the reading and writing processes remain active.

The chip can communicate with a host microcontroller over the standard TWI Bus protocol. In TWI Bus, the SDA (Serial Data Line) line is used to exchange data between the MCU (the Master) and the EEPROM (the Slave). The SCL (Serial Clock Line) is used to carry the clock pulses that are needed to shift-in/shift-out data as needed between the Master and the Slave. SDA and SCL lines are open drain terminals, and they need have 2.2k/4.7k/10k pull-up terminations for wired-AND logic operation in a multi-master and multi-slave TWI Bus System.

10.1.2 Control Byte

Every EEPROM has its own identification code called Device Address or Slave Address. It is a 7-bit data, and it is defined as 10100 A1 A0 for the 24C512 EEPROM. When the slave address is placed in an 8-bit packet, it occupies the upper 7-bit of the packet. The lower-bit of the 8-bit packet is occupied by another bit known as R-W/ (Read-write/) bit. Putting Logic-Low at R-W/ bit indicates that the data movement occurs from Master (ATmega328 MCU) to Slave (24C512 EEPROM); whereas, putting Logic-H indicates data movement from Slave to Master.

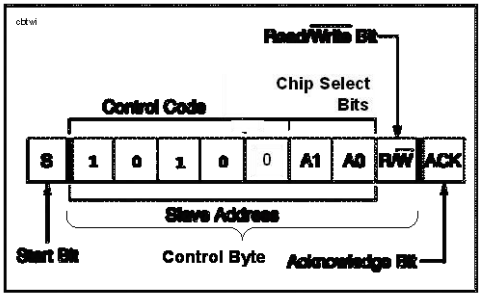


Figure-10.2: *Control Byte Structure of 24C512 EEPROM in TWI Bus Operation (Microchip data sheet)*

10.1.3 TWI Bus Connection and Single Byte Data Write into an EEPROM Location

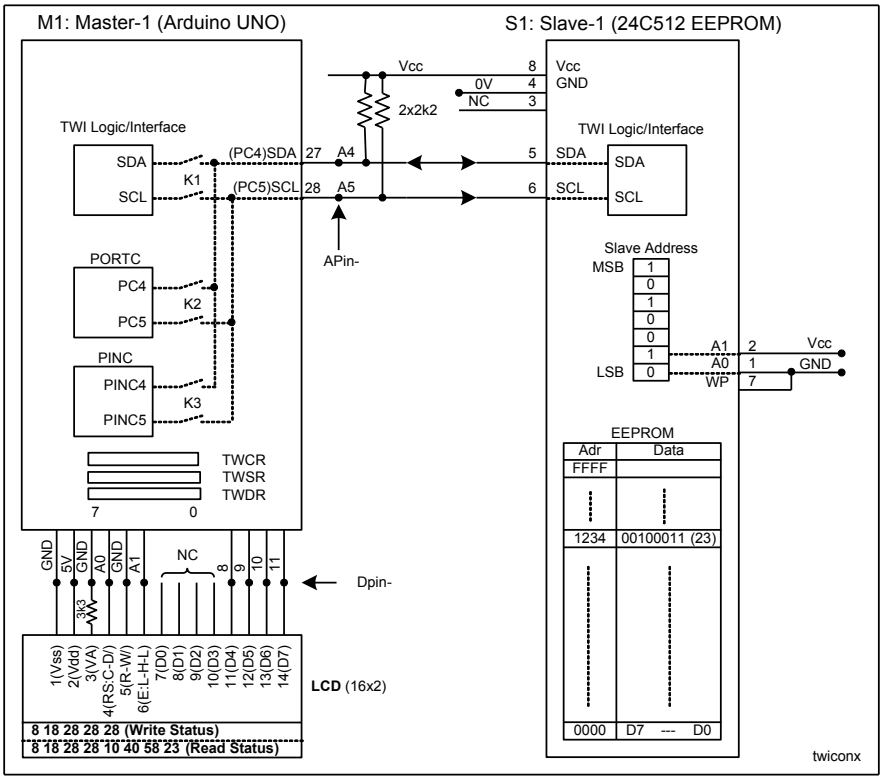


Figure-10.3: *TWI Bus connection between Arduino UNO and 24C512 EEPROM*

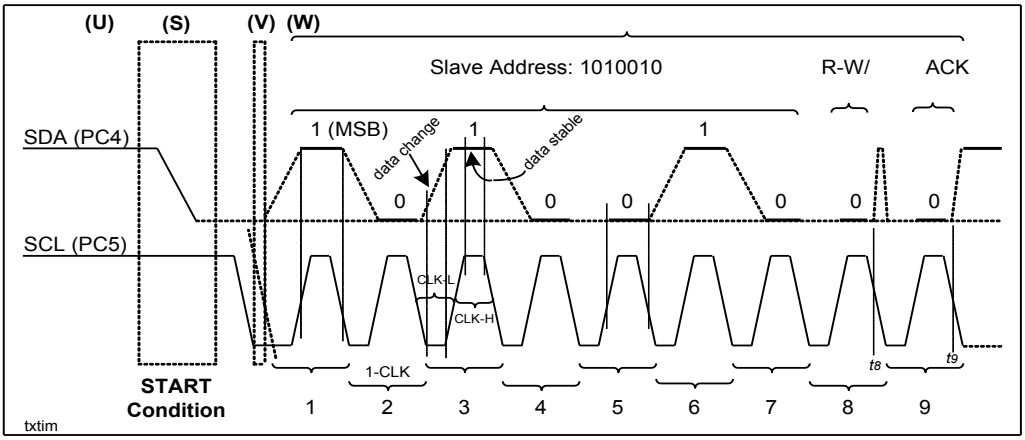


Figure-10.4: *TWI Bus Timing Diagram between Mater and Slave during device address inquiry*

10.1.3.1 TWI Bus Formation

After power up reset, Pin-27 and Pin-28 (Fig-10.3) of the ATmega328 Microcontroller (MCU) are configured as digital IO lines (PC4, PC5) and get connected with PORTC/PINC logic. Pin-27 and Pin-28 become TWI Bus when LH is placed into the TWEN-bit of the TWCR Register of the MCU and get connected with TWI Logic. Now, Pin-27 becomes SDA line and Pin-28 becomes SCL line. SDA Line and SCL Line are respectfully connected with A4 and A5 pins of the edge connector of the Arduino UNO Board. When TWEN bit becomes active, both SDA and SCL lines assume H-states (Fig-10.4, BusEvent-U). The following registers of the ATmega328 are involved in the TWI Bus operation:

twireg

	7 0								
TWBR		TWI Bus Bit Rate Register	TWBR7	TWBR6	TWBR5	TWBR4	TWBR3	TWBR2	TWBR1	TWBR0
TWCR		TWI Bus Control Register	TWINT	TWEA	TWSTA	TWST0	TWWC	TWEN	-----	TWIE
TWSR		TWI Bus Status Register	TWS7	TWS6	TWS5	TWS4	TWS3	-----	TWPS1	TWPS0
TWDR		TWI Bus Data Register	TWD7	TWD6	TWD5	TWD4	TWD3	TWD2	TWD1	TWD0
TWAR		TWI Bus (Slave) Address Register	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE

Figure-10.5: *TWI Bus Registers of ATmega328 Microcontroller*

10.1.3.2 TWI Bus Acquisition

Before any data transmission occurs between Master and Slave using the TWI Bus, the bus Master needs to acquire the control of the bus and is done by bringing L-states on both the SDA and SCL lines (Fig-10.4, BusEvent-V). The BusEvent-V occurs when the user asserts START (Fig-10.4, BusEvent-S) condition on the bus by putting LH into the TWSTA bit of the TWCR Register.

During START condition, the SDA line starts changing its state form High-to-Low (falling edge) while the SCL line is still at H-state. After a while, the SDA line assumes L-state and then the SCL line assumes L-state. If the START condition becomes a successful BusEvent-S, the upper 5-bit of the TWSR Register will hold 00001 which means that the content of the TWSR Register will appear as 0x08 (the lower 3-bit of the TWSR Register are preserved and masked).

The SDA and SCL lines will take some finite amount of time to undergo a change from BusEvent-U to BusEvent-V after the assertion of the START command. The end of the process could be detected by monitoring the TWINT bit of the TWCR Register. Initially, this bit is in the cleared condition (the TWINT bit can be cleared by writing LH into its own position), and it remains at this state until the process ends. At the end of the process, the TWINT bit assumes H-state which triggers the TWSR Register to hold 0x08. The execution of the following codes will bring the TWI Bus into BusEvent-V. Note that the Master does not generate any SCL pulses.

```
LH → TWEN                 //Pin-27 1nd Pin-28 are TWI Bus
LH → TWINT                //Clearing the TWINT bit
LH → TWSTA                //asserting START command
```

The prioritized actions of the above three lines could be achieved by the following single line:

```
TWCR = 0b10100100;        //TWI Bus is formed first; TWINT gets cleared and then START
while (bitRead(TWCR, 7) != HIGH) //checking if process is ended
      ;                          //wait until the process is completed
lcd.((TWSR & 0b11111000), HEX);  //LCD Monitor should show a status word of 0x08
```

10.1.3.3 Detecting the Presence of EEPROM in the TWI Bus

A practical TWI Bus may contain more than one device: (a) EEPROM with device address 1010010, (b) DS1307 with device address 1101000, and (c) BMP85 with device address 1110111. A particular device such as EEPROM, in this case, is detected by the following events:

(a) Master wishes to write a data byte (say, 0x23) at EEPROM location 0x1234. So, the data direction is from Master to Slave, and accordingly the R-W/ bit of the control byte will be 1010010 0 (deviceaddress, R-W/).

(b) Master puts the control byte on the TWI Bus via TWDR Register (Fig-10.4, BusEvent-W).

(c) Master pulls the TWINT bit of the TWCR Register to check that the process has ended. While pulling this bit, the 8 SCL pulses are automatically generated by the Master to shift out the bits of the control byte.

(d) At the end of 8^{th} SCL pulse (Fig-10.4), the Master releases the SDA line which immediately assumes H-state by virtue of pull-up resistor.

(e) The address (upper 7-bit of the control byte) bits are matched with the address of the EEPROM; the EEPROM accepts the address, and in recognition the EEPROM generates the ACK signal (Fig-10.4) by pulling down the SDA line.

(f) To sample the ACK signal, the Master generates the 9^{th} SCL pulse (Fig-10.4). The ACK signal triggers the generation of the appropriate status word into the TWSR Register.

(g) EEPROM is ready to accept next data byte; correct status word (0x18) in the TWSR.

All the above events could be generated by executing the following codes:

```
TWDR = 0b10100100;        //deviceaddress + 0 = SLA + W = Slave Address in Write Mode
TWCR = 0b10000100;        //TWI Bus remains enabled; TWINT gets cleared; START bit is OFF
while (bitRead(TWCR, 7) != HIGH) //checking if process is ended
      ;                          //wait until the process is completed
lcd.((TWSR & 0b11111000), HEX);  //LCD Monitor should show a status word of 0x18
```

10.1.3.4 Setting Address Counter of EEPROM

The Master wants to store an 8-bit data (0x23) into location 0x1234 of the EEPROM. The address is to be deposited first into the Address Counter of the EEPROM. This is done by executing the

following instructions where higher byte (0x12) of the address is deposited first. The bus timing diagrams are shown in Fig-10.6 and Fig-10.7.

```
TWDR = 0x12;              //higher byte of the address
TWCR = 0b10000100;        //TWI Bus remains enabled; TWINT gets cleared; START bit is OFF
while (bitRead(TWCR, 7) != HIGH) //checking if process is ended
      ;                          //wait until the process is completed
lcd.((TWSR & 0b11111000), HEX);  //LCD Monitor should show a status word of 0x28
//--------------------------------------------------------------------------------
TWDR = 0x34;              //lower byte of the address
TWCR = 0b10000100;        //TWI Bus remains enabled; TWINT gets cleared; START bit is OFF
while (bitRead(TWCR, 7) != HIGH) //checking if process is ended
      ;                          //wait until the process is completed
lcd.((TWSR & 0b11111000), HEX);  //LCD Monitor should show a status word of 0x28
```

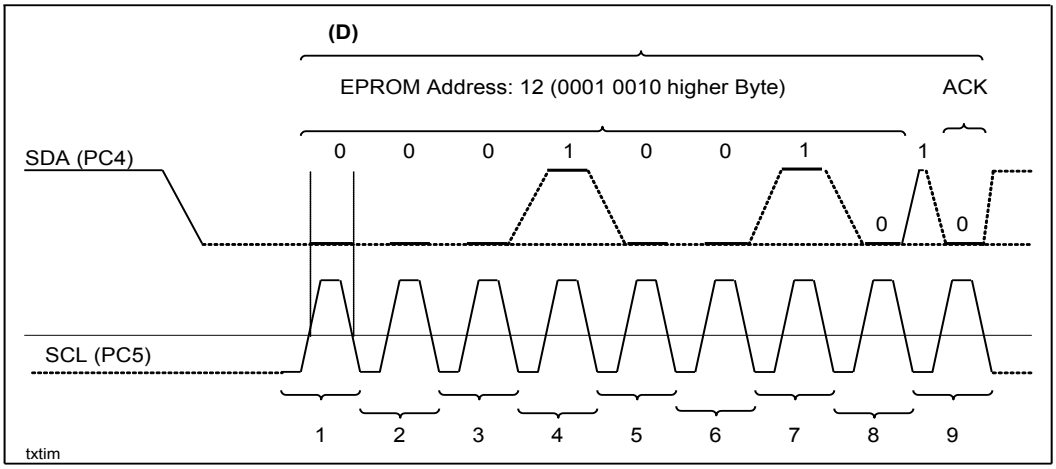


Figure-10.6: *TWI Bus timing diagram for setting upper byte of the address in the Address Counter*

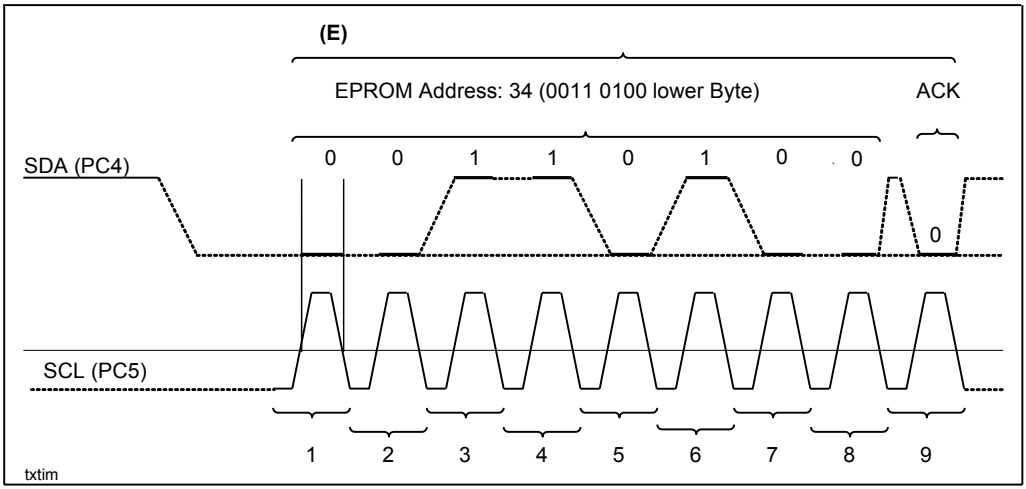


Figure-10.7: *TWI Bus timing diagram for setting lower byte of the address in the Address Counter*

10.1.3.5 Putting 8-bit Data into EEPROM Location Pointed by Address Counter

The following instructions are executed to put 0x23 into the EEPROM location pointed by the Address Counter:

```
TWDR = 0x23;              //data byte
TWCR = 0b10000100;        //TWI Bus remains enabled; TWINT gets cleared; START bit is OFF
while (bitRead(TWCR, 7) != HIGH) //checking if process is ended
      ;                          //wait until the process is completed
lcd.((TWSR & 0b11111000), HEX);  //LCD Monitor should show a status word of 0x28
```

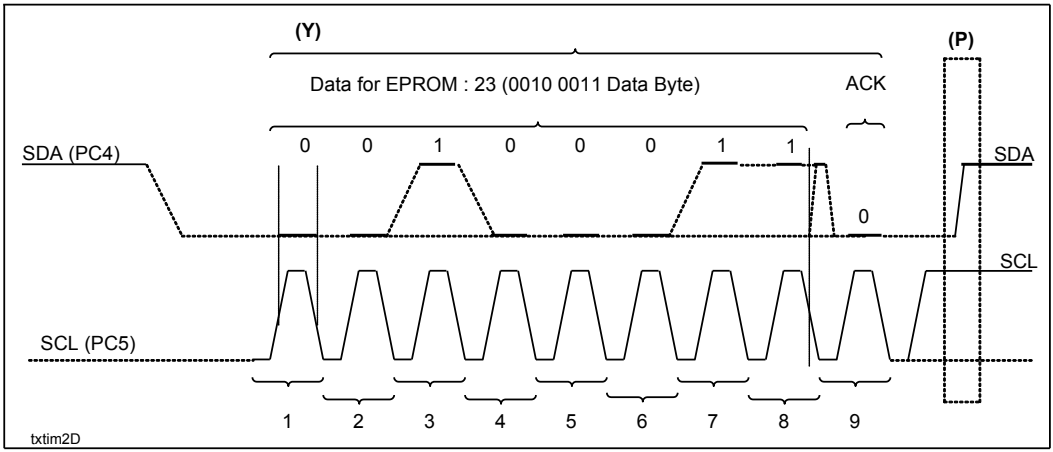


Figure-10.8: *TWI Bus timing diagram for the transmission of data byte into EEROM*

10.1.3.6 Data Byte actually getting written into EEPROM Location

The EEPROM location has received the data byte. The data byte gets written into the location in one of the following ways:

(a) Assert STOP command (Fig-10.8, BusEvent-P) on the TWI Bus to free the bus and to free the EEPROM. Once freed, the EEPROM starts a self-timed 'data write logic' which takes about 5 ms to complete the write operation. The following codes could be executed:

```
TWCR = 0b10010100;      //TWI Bus remains enabled; TWINT gets cleared; STOP command
while (bitRead(TWCR, 7) != HIGH) //checking if process is ended; hangs up in loop
   ;                    //wait until the process is completed
lcd.((TWSR & 0b11111000), HEX);  //LCD Monitor shows: 0xF8 without while() loop
delay (5);                 //wait for about 5 ms to complete the write operation
```

STOP command is issued with LH at the TWSTO bit of the TWCR Register. The SDA line makes a transition from LL to LH state while the SCL line is at H-state (Fig-10.8, BusEvent-P).

(b) Assert STOP command on the TWI Bus to free the bus and to free the EEPROM. Once freed, the EEPROM starts a self-timed 'data write logic' which takes about 5 ms to complete the write operation. During this time, the EEPROM will not respond to any TWI Bus command that involves its deviceaddress inquiry in the read mode (R-W/ bit of the

control byte should be at LH). The execution of the underneath Flow Chart allows a user to detect the end of 'data write cycle'.

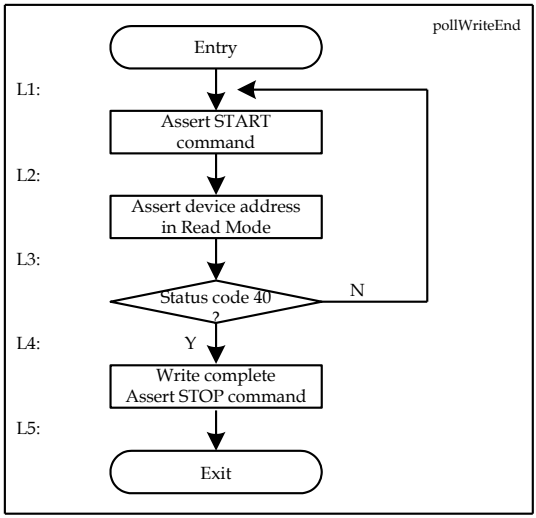


Figure-10.8: *Flow Chart describing the mechanism of detecting 'end of write' by polling status code*

The following codes could be executed in lieu of the Flow Chart of Fig-10.8:

```
//--- wait until data get written into EEPROM location-----------------------------
do
{
   TWCR = 0b10100100; //TWINT TWEA TWSTA TWSTO TWCC TWEN X TWIE  //START command
   while (bitRead(TWCR, 7) != HIGH)
      ;
   TWDR = 0b10100101;     //SLA + R     //Read Mode works; write mode do not work!
   TWCR = 0b10000100;     //Needed to generate 9 clock pulses on SCL line
   while (bitRead(TWCR, 7) != HIGH)
      ;
}

while ((TWSR & 0b11111000) != 0x40);  //correct status code
TWCR = 0b10010100;        //STOP command
//-------------------------------------------------------------------------------
```

10.1.4 Single Byte Data Read from an EEPROM Location

In Section-10.1.3, we have written 1-byte data (0x23) into EEPROM location 0x1234. We wish to read it back and show on the LCD of Fig-10.3. The procedures are:

(1) Acquire Bus control by asserting START command and checking the status code (0x08).

(2) Make link with EEPROM in Write Mode (R-W/ bit = LL) by sending the device address. Why is it Write Mode? After establishing link with the EEPROM, the user is going to write location address (0x1234) into the Address Counter of the EEPROM. The data movement direction is from the Master to the Slave. Therefore, the R-W/ bit must be set at LL.

(3) Assert upper byte (0x12) of the address on the TWI Bus.

(4) Assert lower byte (0x34) of the address on the TWI Bus.

(5) Now it is time to read data session from Slave to Master. The data movement direction must be changed by sending the device address in Read Mode (R-W/ bit = LH). How can we accomplish this task? Should we STOP the bus, and then acquire the bus and then set the Read Mode? If we STOP the bus, the bus will become free; some other device (Master-2) might seize the bus, and we will not be able to get it until it is released by Master-2. So, we will not go in STOPing the bus; rather, we will issue REPEATED START command which will allow us establishing link with EEPROM in Read Mode without losing the control of the bus. REPEATED START command is same as the normal START command, and it is asserted by putting LH at the TWSTA bit of the TWCR Register.

(6) Assert (REPEATED) START command on TWI bus and check the status code for 0x10.

(7) Now, we have to generate 9 (8+1) clock pulses on the SCL line to bring out 8-bit data (MS-bit first) from the memory location (0x1234) and stores the data into the TWDR Register of the Master. The 9th clock pulse will be required to sample the ACK (or NACK) signal which will now be generated by the Master (because it is the receiver). The ACK signal is used to advance the Address Counter of the EEPROM in order to read the data of the next location (0x1235). The ACK signal also generates known status code (0x50) which prompts the TWI Logic of the Master to generate next set of clock pulses for reading data of the next memory location. The master will generate ACK signal (pulling down the SDA line and then release it during 9th clock pulse) if the TWEA bit of the TWCR Register has already been enabled. Advancing the Address Counter of the EEPROM will be a justified action if the Master knows that there is a data it has to read, and accordingly the Master will assert next set of clock pulses on the SCL line.

In the present case, we have 1-byte data to read from location 0x1234. After reading this data byte, the Master must not generate ACK signal to advance the Address Counter. How do we do it? Generation of ACK signal will be prohibited (equivalent of the generation of NACK signal) if the TWEA bit is made inactive (disabled). NACK signal does not advance the Address Counter of the EEPROM; it generates known status code (0x58) which prevents the TWI Logic of the Master from generating the next set of clock pulses. The following codes could be executed to generate ACK and NACK signals as needed:

(a) Reading 2-byte data from Location 0x1234 and 0x1235

```
TWCR = 0b11000100;    //TWI remains enabled; TWINT is cleared; TWEA active
while(bitRead(TWCR, 7) !=HIGH)  //9 (8+1) clock pulses are being generated
       ;
dataReceive[0] = TWDR;   //data of location 0x1234 is received via TWDR
lcd.print((TWSR & 0xF8); //status code 0x50; Address Counter advances at 0x1235
//-------------------------------------------------------------------------
TWCR = 0b10000100; //TWI remains enabled; TWINT cleared; TWEA disabled for NACK
while(bitRead(TWCR, 7) !=HIGH)  //9 (8+1) clock pulses are being generated
       ;                        //wait until process ends
dataReceive[1] = TWDR;   //data of location 0x1235 is received via TWDR
lcd.print((TWSR & 0xF8;  //status code 0x58; Address Counter remains at 0x1235
```

(b) Reading 1-byte data from Location 0x1234

```
TWCR = 0b10000100; //TWI remains enabled; TWINT cleared; TWEA disabled for NACK
while(bitRead(TWCR, 7) !=HIGH)  //9 (8+1) clock pulses are being generated
       ;
dataReceive[0] = TWDR;   //data of location 0x1234 is received via TWDR
lcd.print((TWSR & 0xF8); //status code 0x58; Address Counter remains at 0x1234
```

10.1.5 Multibyte Write/Read between UNO and 24C512 EEPROM using TWI Bus

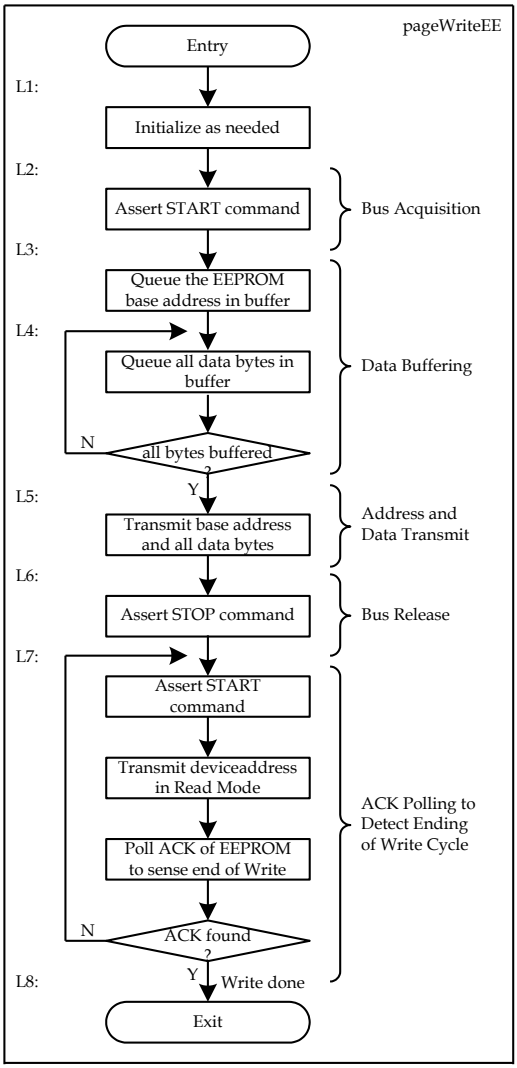


Figure-10.8: *Data write algorithm*

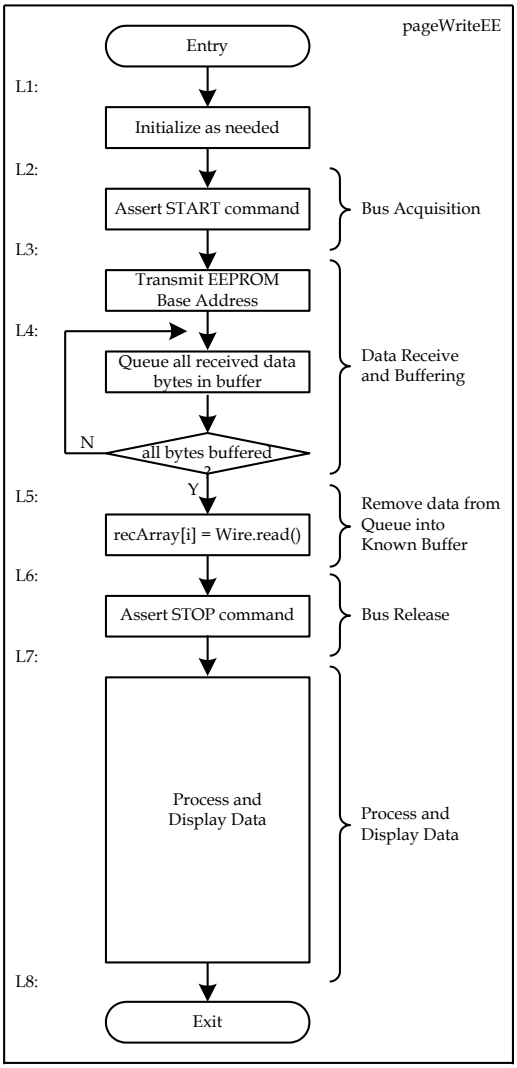


Figure-10.9: *Data read algorithm*

```
#include <Wire.h>
#define MAX 32      //32 bytes to Write/Read as Wire.requestFrom() supports only 32 byte
Byte wrArray[MAX];  //Array holds some known data that will be written into EEPROM
byte rdArray[MAX];  //Array holds data after reading from EEPROM
unsigned int locaddress = 0x1000;   //EEPROM base address

void setup()
{
  Serial.begin(9600);
```

```
  Wire.begin();
  pinMode(13, OUTPUT); //DPin-13 drives built-in LED (L) of Arduino UNO

  for (int i = 0; i<MAX; i++)    //loading known data 0, 1, …, 1F into wrArray[32]
  {
    wrArray[i] = i;
  }

  wrEEPROM();       //calling user defined function to write 32 byte data into EEPROM
  rdEEPROM();       //calling user defined function to read 32 byte data from EEPROM

  for (int k = 0; k<MAX; k++)    //showing data on the Serial Monitor (Write-Read-Display)
  {
    Serial.print(rdArray[k], HEX);
  }
}

void loop()
{
  digitalWrite(13, !digitalRead(13));  //blinking L at 2-sec interval
  delay(2000);
}

void wrEEPROM()
{
  locaddress = 0x1000;                 //EEPROM base address

  Wire.beginTransmission(0b1010010);   //EEPROM deviceaddress; START command
  Wire.write(highByte(locaddress));    //base address (upper byte) is queued in buffer
  Wire.write(lowByte(locaddress));     //base address (lower byte) is queued in buffer

  for (int n = 0; n<MAX; n++, locaddress++)   //data bytes are queued in buffer
  {
    Wire.write(wrArray[n]);
  }
  Wire.endTransmission(); //Address/data bytes are transmitted to EEPROM; STOP command

  do                      //poll ACK signal to detect end of Write Cycle
  {
   Wire.beginTransmission(0b1010010);         //START command; device address + Write Mode
   Wire.write(0x00);
  }
  while((Wire.endTransmission()) !=0x00);  //status code 0x00 indicates success
  //---alternate codes for the above do-while
  do
  {
     TWCR = 0b10100100; //TWINT TWEA TWSTA TWSTO TWCC TWEN X TWIE
     while (bitRead(TWCR, 7) != HIGH)
        ;
     TWDR = 0b10100101;   //SLA + Read     //Write Mode does not work!
     TWCR = 0b10000100;
     while (bitRead(TWCR, 7) != HIGH)
       ;
  }
  while ((TWSR & 0b11111000) != 0x40); //expected status code in Read Mode
  //--------------------------------
 }
```

```
void rdEEPROM()              //reading data from EEPROM
{
  locaddress = 0x1000;       //EEPROM base address

  Wire.beginTransmission(0b1010010);   //STATRT command and address queue
  Wire.write(highByte(locaddress));
  Wire.write(lowByte(locaddress));
  Wire.endTransmission();              //Transmit and STOP command


  Wire.requestFrom(0b1010010, MAX);    //loop until data read complete from EEPROM
  byte x = Wire.available();           //number of data bytes received from EEPROM
  for (int j = 0; j<x; j++)            //moving data from queue to known rdArray[32]
  {
    rdArray[j] = Wire.read();
  }

}
```

10.10 Problems and Solutions

1 Write down the procedures for writing 1-byte data into any location (say, 0x1234) of the 24C512 EEPROM.

(a) Assert START command and check for the generation of status code 0x08

(b) Assert SLA + W/ (0b10100100) and check for status code 0x18

(c) Assert upper byte of address (0x12) and check for status code 0x28

(d) Assert lower byte of address (0x34) and check for status code 0x28

(e) Assert data byte (say, 0x23) and check for status code 0x28

(f) Assert STOP command and no check for any status code

(g) Allow 5 ms time for the data byte to get written into the EPROM. Alternatively, poll ACK signal (status code 0x40) to detect the end of 'write cycle'. The EEPROM remains disconnected from TWI Bus as long as the write cycle is active. The polling codes:

```
//--- wait until data get written into EEPROM location----------------------
do
{
    TWCR = 0b10100100;                  //START command
    while (bitRead(TWCR, 7) != HIGH)
        ;
    TWDR = 0b10100101;     //SLA + R    //Read Mode works; write mode do not
    work!
    TWCR = 0b10000100;                  //Needed to generate 9 clock pulses
    while (bitRead(TWCR, 7) != HIGH)
        ;
}

while ((TWSR & 0b11111000) != 0x40);  //status code when ACK is received
TWCR = 0b10010100;                     //STOP command
//-------------------------------------------------------------------------
```