SPI and I^2C

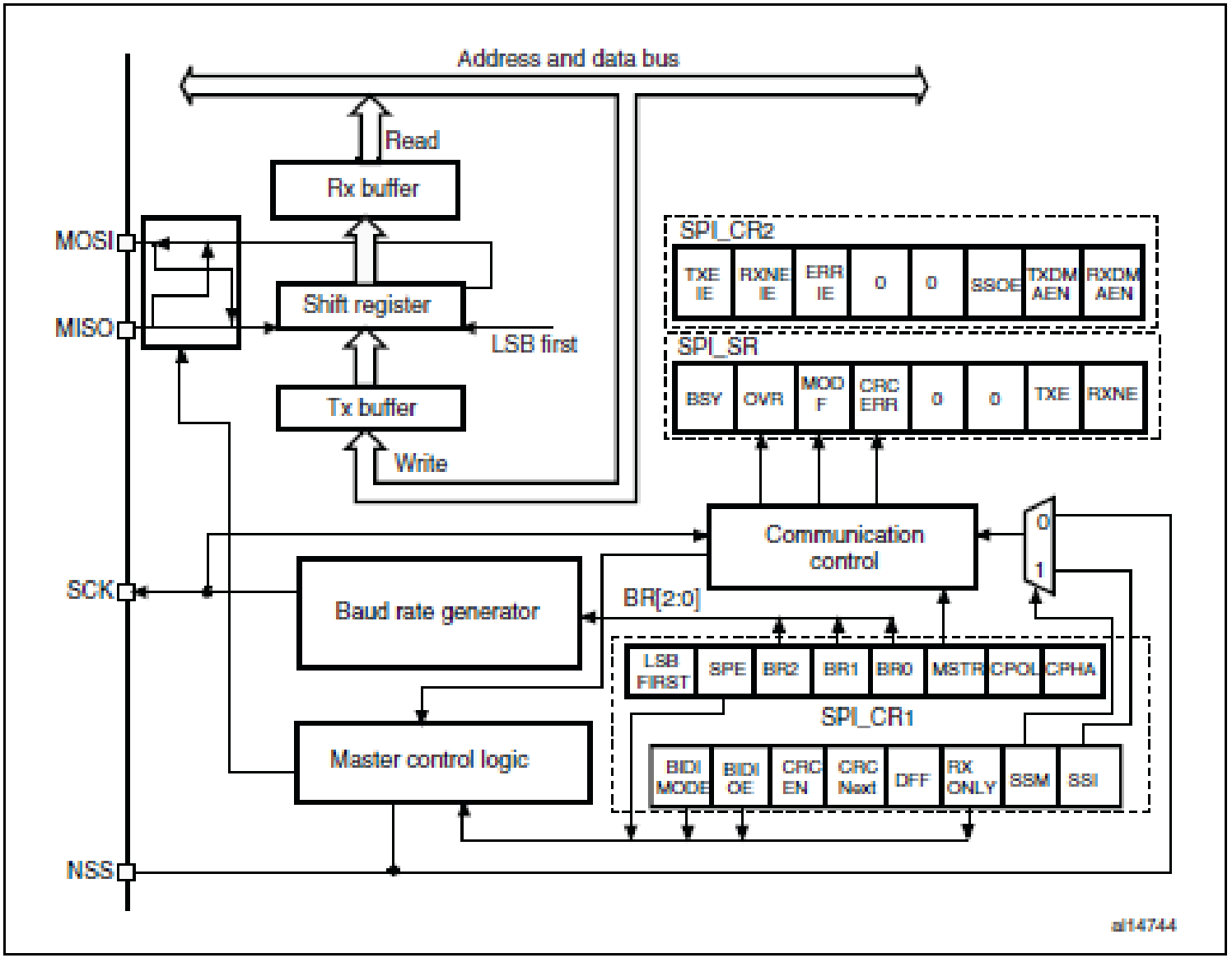
Serial Peripheral Interface (SPI)

- SPI allow half/full duplex, synchronous, serial communication with external devices
- The interface can be configured as master for MCU board
- Communication clock (SCLK) is also provided for synchronous communication

SPI features

- Full duplex synchronous transfer on 3 lines
- Simplex synchronous transfer on 2 lines
- Master or slave operation
- Multi-master mode support

SPI block diagram

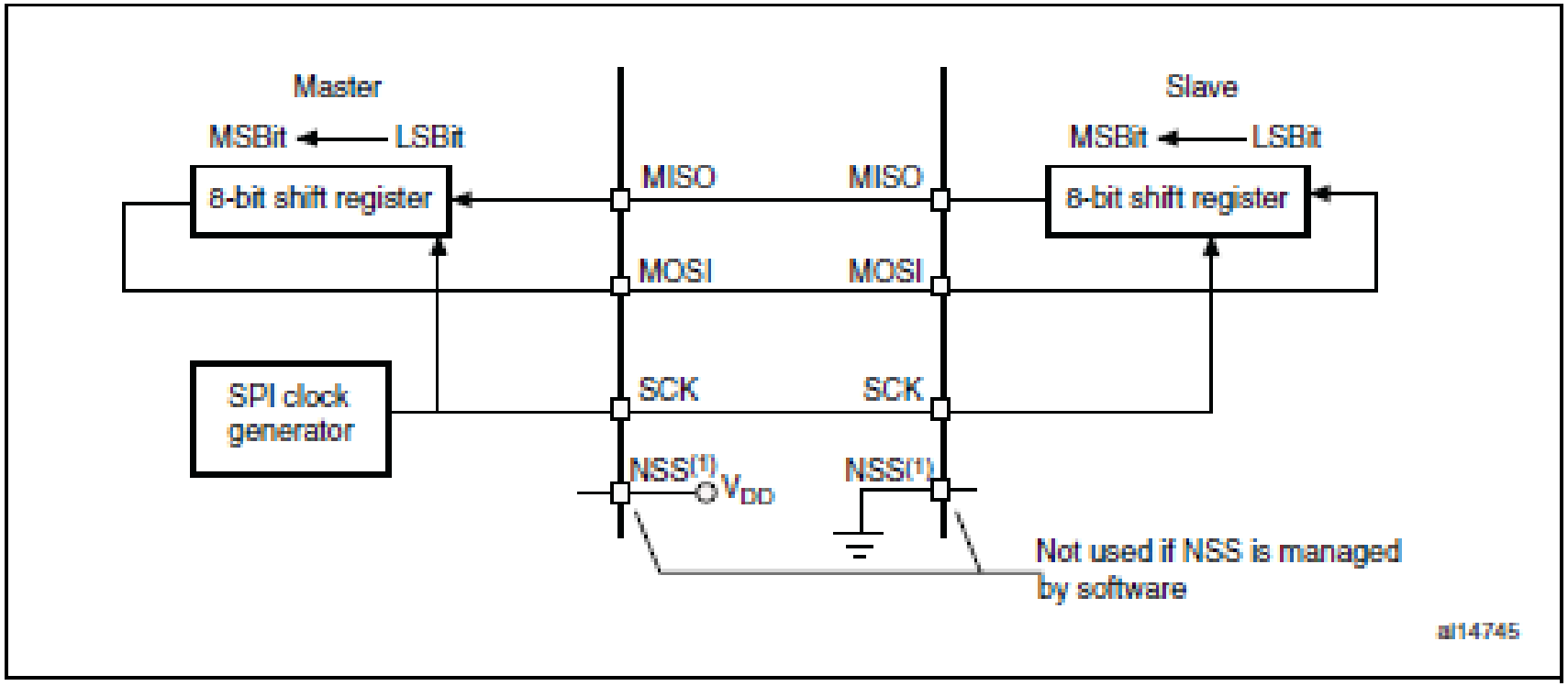


Basic SPI configuration

Usually, SPI is connected using 4 pins

- MISO (Master In/Slave Out): transmit data in slave mode, and received data in master mode
- MOSI (Master Out/Slave In): transmit data in master mode, and received data in slave mode
- SCLK: Clock
- NSS (slave select): to select a slave device

Single Master/Single Slave config



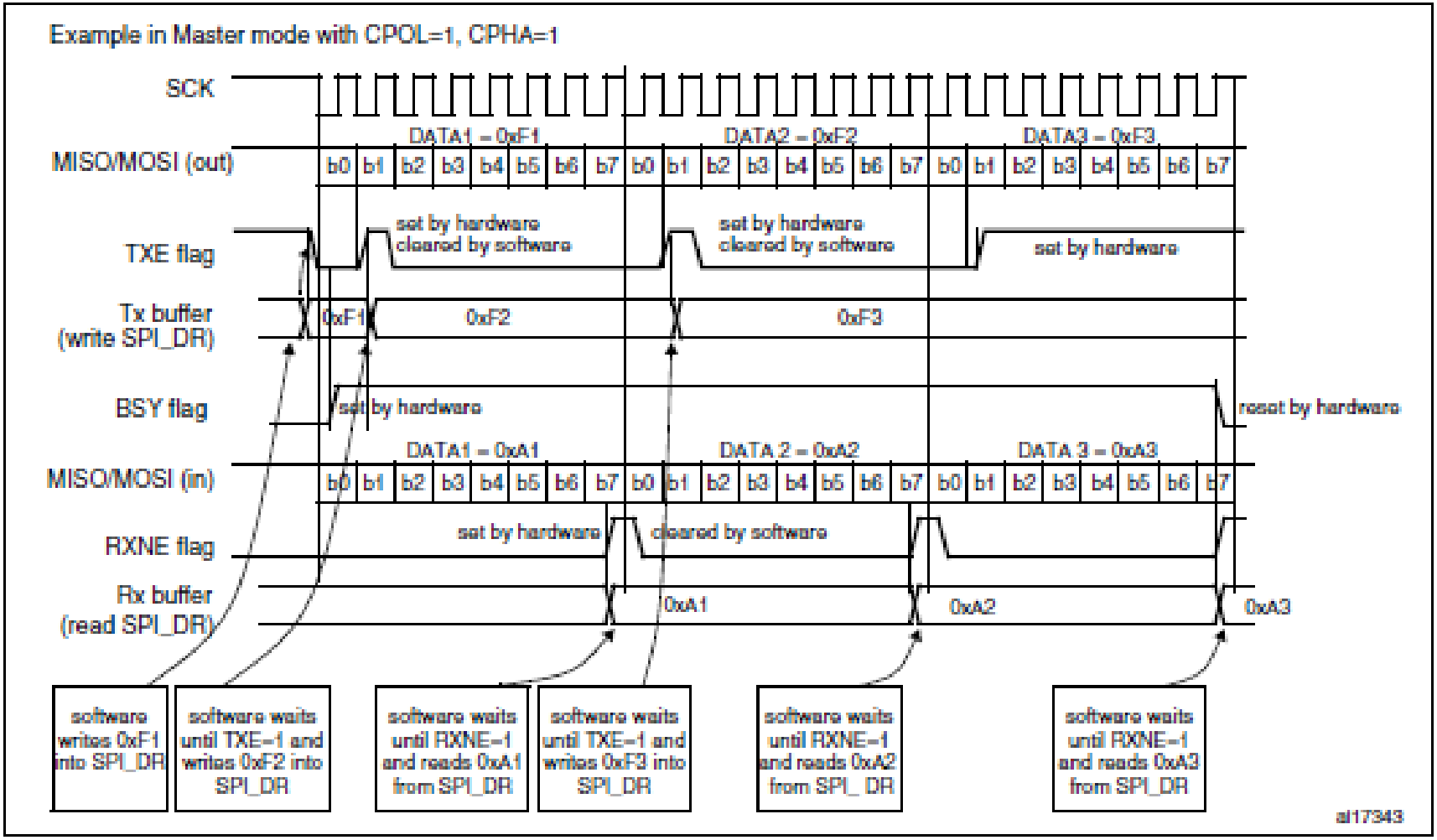
Configuring SPI in Master mode

- Select BR[2:0] to define baud rate
- Select CPOL and CPHA to define data transfer and serial clock relationship
- Set DFF bit to define 8 or 16 bit data transfer
- Configure LSBFirst bit in SPI_CR1 to define frame format
- Connect NSS if required
- MSTR and SPE must be set

Configuring SPI in Slave mode

- Select CPOL and CPHA to define data transfer and serial clock relationship
- Set DFF bit to define 8 or 16 bit data transfer
- LSBFirst bit in SPI_CR1 must be the same as Master
- Connect NSS if required
- Clear MSTR and set SPE bit

Timing diagram



I2C

- Inter-Integrated bus (I2C)
- Interface between microcontroller and I2C bus
- Provide multi-master support

I2C feature

- Master master configuration
- I2C master feature
  - Clock generation
  - Start and Stop generation
- I2C slave feature
  - programmable I2C address detection
  - dual address capabilities
  - Stop bit detection
- 2 interrupt vectors

I2C function

- There are 2 pins, data pin (SDA) and clock (SCL)

- It can be connected with standard speed (up to 100KHz) or fast mode (up to 400KHz)

I2C function description

Mode selection:

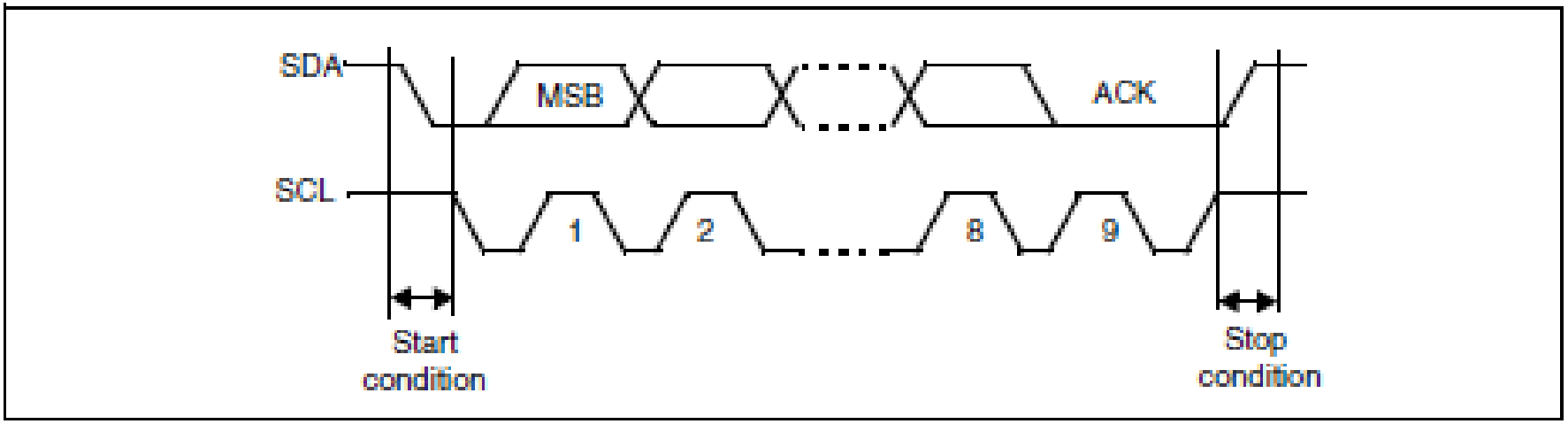
- Slave transmit
- Slave receiver
- Master transmit
- Master receiver

By default, it operates in Slave mode. It automatic switch once it receive a START condition from master to slave

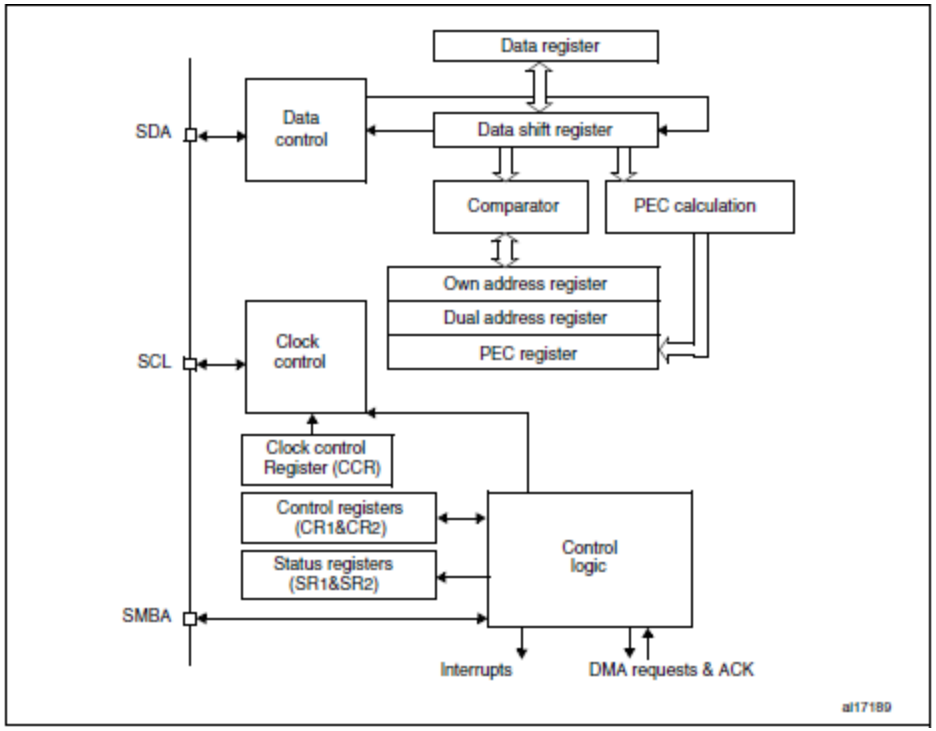
Communication flow

- In master mode, I2C interface initiates the data transfer and the clock signal.
- A serial data transfer always begins with a start condition and ends with a stop condition
- In slave mode, the interface is capable of recognizing its own address (7-10 bits)
- Data are transmitted as 8 bit bytes, MSB first.
- A 9^{th} clock pulse, receiver must send an acknowledge

I2C data transfer



I2C block diagram



I2c master mode

- Program the peripheral input clock in I2C_CR2 register to generate correct timing
- Configure the clock control register
- Configure the rise time register
- Program I2C_CR1 register to enable the peripheral
- Set the start bit in I2C_CR1 to generate a Start condition

Example

```
int main(){
  RCC_setup();
  GPIO_setup();
  I2C_setup();
  while(1){
    if(I2C_LM75_Status()==SUCCESS){
        temp = LM75_readTemp();
    }
  }
}
void GPIO_setup(){
  GPIO_InitTypeDef GPIO_InitStructure;
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB|RCC_APB2Periph_AFIO,ENABLE);
  GPIO_InitStructure.GPIO_Pin   =  GPIO_Pin_6 | GPIO_Pin_7;
  GPIO_InitStructure.GPIO_Speed =  GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode  =  GPIO_Mode_AF_OD;
  GPIO_Init(GPIOB, &GPIO_InitStructure);
}
```

```
void I2C_setup(){
  I2C_InitTypeDef   I2C_InitStructure;
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C1,ENABLE);
  I2C_DeInit(I2C1);
  I2C_InitStructure.I2C_Mode = I2C_Mode_I2C;
  I2C_InitStructure.I2C_DutyCycle = I2C_DutyCycle_2;
  I2C_InitStructure.I2C_OwnAddress1 = 0x00;
  I2C_InitStructure.I2C_Ack = I2C_Ack_Enable;
  I2C_InitStructure.I2C_AcknowledgedAddress = I2C_AcknowledgeAddress_7bit;
  I2C_InitStructure.I2C_ClockSpeed = 200000;
  I2C_Init(I2C1, &I2C_InitStructure);
  I2C_Cmd(I2C1, ENALBE);   // enable I2C1
}
```

```
ErrorStatus I2C_LM75_Status(){
  unsigned int I2C_TimeOut = 100000;
  I2C_ClearFlag(I2C1, I2C_FLAG_AF);
  I2C_AcknowledgeConfig(I2C1, ENABLE);
  I2C_GenrateSTART(I2C1, ENABLE);
  while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT); // wait for start
I2C_Send7BitAddress(I2C1, LM75_ADDR, I2C_Direction_Transmitter); //send slave addr
while(!I2C_CheckEvent(I2C1,I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTE
      D)) && I2C_TimeOut){  // get status ready
    I2C_TimeOut--;
  }
  if(I2C_GetFlagStatus(I2C1,I2C_FLAG_AF != 0x0){ return ERROR;
  }else return SUCCESS;
}
unsigned int LM75_readTemp(){
  unsigned int RegValue = 0;
  I2C_AcknowledgeConfig(I2C1, ENABLE);
  I2C_GenerateStart(I2C1, ENABLE);
 //  wait for START to complete
  while(I2C_CheckEvent(I2C1, I2CEVENT_MASTER_MODE_SELECT));
  I2C_Send7bitAddress(I2C1,  LM75_ADDR, I2C_Direction_Transmitter);
```

```
  //  wait for data transfer complete
 while(I2C_CheckEvent(I2C1, I2CEVENT_MASTER_RECEIVER_MODE_SELECTED));
 // send data pointer
I2C_SendData(I2C1,LM75_TEMP_REG);
//  wait  for data transfer complete
 while(I2C_CheckEvent(I2C1, I2CEVENT_MASTER_BYTE_TRANSMITTED));
 I2C_GenerateStart(I2C1, ENABLE);
 //  wait for START to complete
  while(I2C_CheckEvent(I2C1, I2CEVENT_MASTER_MODE_SELECT));
  I2C_Send7bitAddress(I2C1,  LM75_ADDR, I2C_Direction_Receiver);
 while(I2C_CheckEvent(I2C1, I2CEVENT_MASTER_RECEIVER_MODE_SELECTED));
 while(I2C_CheckEvent(I2C1, I2CEVENT_MASTER_BYTE_RECEIVED));
  // store data
  RegValue = I2C_ReceiveData(I2c1) << 8;
  // disable acknowledgement
  I2CAcknowledgeConfig(I2C1, DISABLE);
  // send I2C1 stop condition
  I2c_GenerateStop(I2C1, ENABLE);
  // test on EV7 and clear
 while(I2C_CheckEvent(I2C1, I2CEVENT_MASTER_BYTE_RECEIVED));
 Regvalue != I2C_ReceiveData(I2C1);
 return RegValue >> 7;
}
```

Questions?