SECTION 18.5: TWI PROGRAMMING WITH CHECKING STATUS REGISTER

In this section we discuss TWI programming with checking the value of status register. By checking the value of the status register you can monitor the TWI module current state and operation. This helps you to detect an error when it happens and resolve it at the same time. This is an advanced topic and used only if you are connecting I2C to multiple masters.

As we mentioned before, there are four modes of operation: master transmitter, master receiver, slave transmitter, and slave receiver. We will discuss each mode separately because each mode has its own special status codes. For each mode of operation there is a flowchart that shows the sequence of steps in each mode and also a figure that summarizes most of the status values for each mode in a single table.

Programming of the AVR TWI in master transmitter operating mode

Figure 18-18 shows the steps of programming the AVR TWI in master transmitter mode. Here we focus on each step in more detail:

Initialization

To initialize the TWI module to operate in master operating mode, we should do the following steps:

1. Set the TWI module clock frequency by setting the values of the TWBR register and the TWPS bits in the TWSR register.
2. Enable the TWI module by setting the TWEN bit in the TWCR register to one.

Transmit START condition

To start data transfer in master operating mode, we must transmit a START condition. To transmit a START condition we should do the following steps:

1. Set the TWEN, TWSTA, and TWINT bits of TWCR to one. Setting the TWEN bit to one enables the TWI module. Setting the TWSTA bit to one tells the TWI to initiate a START condition when the bus is free, and setting the TWINT bit to one clears the interrupt flag to initiate operation of the TWI module to transmit a START condition.
2. Poll the TWINT flag in the TWCR register to see when the START condition is completely transmitted.
3. When the TWINT flag is set to one, check the value of the status register to see if the START condition transmitted successfully. Notice that you have to mask the two LSB bits of the status register to get ride of prescalers. If the status value is 0x08 it indicates that the START condition has been transmitted successfully.

Send SLA + W

To send SLA + W, after transmitting the START condition, we should do the following steps:

1. Copy SLA + W to the TWDR.

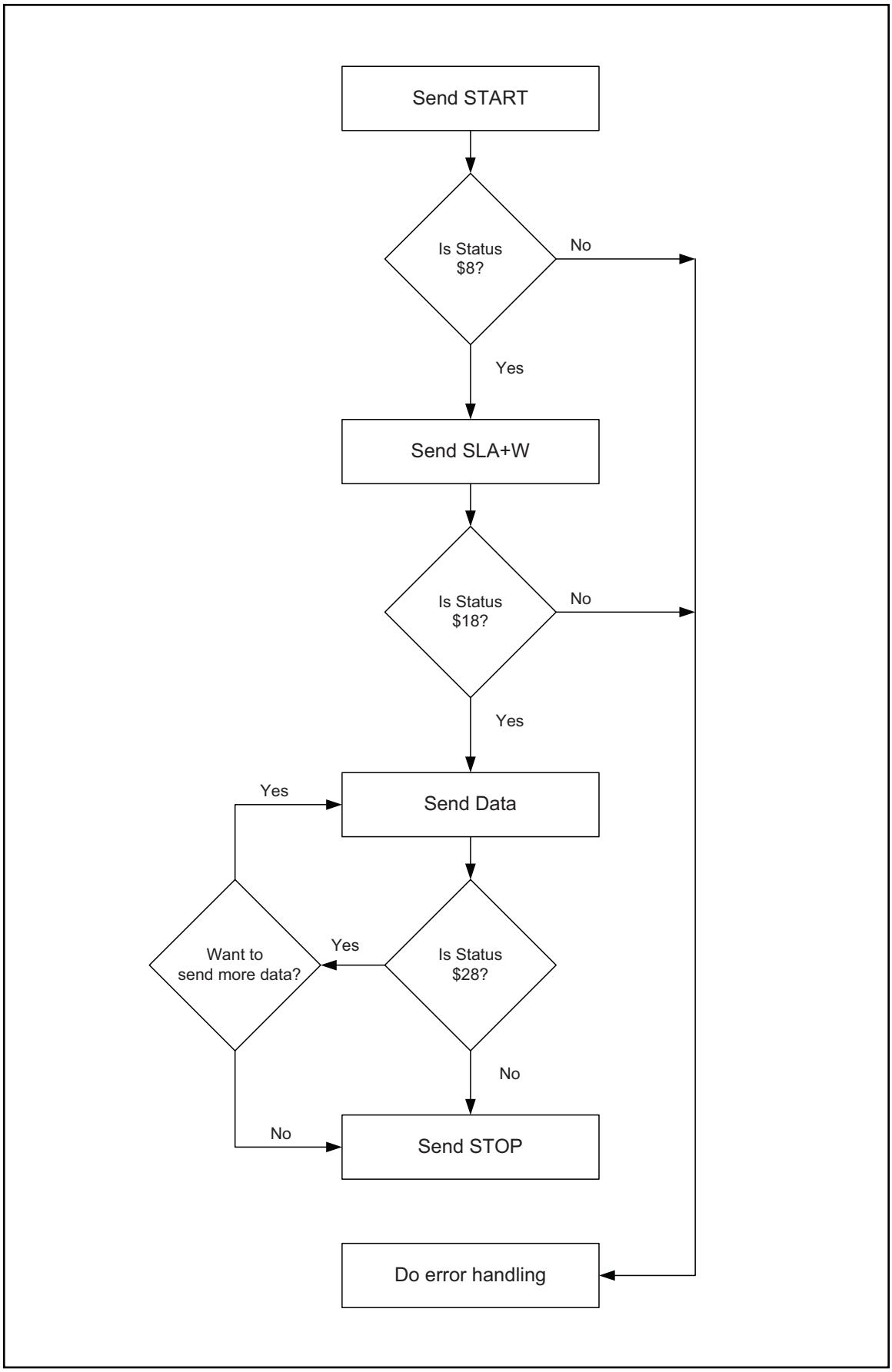

Figure 18-18. Programming Steps of Master Transmitter Mode with Checking of Flags

2. Set the TWEN and TWINT bits of the TWCR register to one to start sending the byte.
3. Poll the TWINT flag in the TWCR register to see when the byte is completely transmitted.
4. When the TWINT flag is set to one, you should check the value of the status register to see if the SLA + W is transmitted successfully. If the status value is 0x18, it indicates that the SLA + W has been transmitted and ACK received successfully.

Send data

To send data, after transmitting of SLA + W, we should do the following steps:

1. Copy the byte of data to the TWDR.
2. Set the TWEN and TWINT bits of the TWCR register to one to start sending the byte.
3. Poll the TWINT flag in the TWCR register to see whether the byte is completely transmitted.
4. When the TWINT flag is set to one, you should check the value of the status register to see if the data has been transmitted successfully and the value of ACK was as expected. Notice that NACK does not necessarily indicate an error; it may indicate that no more data needs to be transmitted. If the status value indicates that ACK is received (0x28) you can either transmit a STOP condition or repeat this function (Send Data) to transmit more data; otherwise, you should transmit a STOP condition.

Transmit STOP condition

To stop data transfer, we must transmit a STOP condition. This is done by setting the TWEN, TWSTO, and TWINT bits of the TWCR register to one. Notice that we cannot poll the TWINT flag after transmitting a STOP condition.

Figure 18-19 shows the meanings of the different values of the status register and possible responses to each of them.

Initialization:	Set values of TWBR register and prescaler bits TWCR = 0x04 TWCR = (1<<TWEN)|(1<<TWINT)|(1<<TWSTA)	Enable TWI Transmit START condition

Status	Meaning	Your Response	Next Action By TWI module
$8	START condition has been transmitted	TWDR = SLA+W TWCR =(1<<TWEN)|(TWINT)	SLA + W will be transmitted ACK or NACK will be returned
$18	SLA + W transmitted. ACK has been received	TWDR = DATA TWCR =(1<<TWEN)|(TWINT)	DATA byte will be Transmitted ACK or NACK will be returned
$20	SLA + W transmitted. NACK has been received	TWCR =(1<<TWEN)|(TWINT)|(TWSTO)	STOP condition will be transmitted
$28	Data byte has been transmitted. ACK has been received.	TWDR = DATA TWCR =(1<<TWEN)|(TWINT) OR TWCR =(1<<TWEN)|(TWINT)|(TWSTO)	DATA byte will be Transmitted ACK or NACK will be returned STOP condition will be transmitted
$30	Data transmitted. NACK received	TWCR =(1<<TWEN)|(TWINT)|(TWSTO)	STOP condition will be transmitted

Figure 18-19. TWSR Register Values for Master Transmitter

Program 18-14 shows how a master writes 11110000 on a slave with address 1101000. The program checks the value of the status register in each step of the operation.

```
.INCLUDE "M32DEF.INC"

     LDI   R21,HIGH(RAMEND);set up stack
     OUT   SPH,R21
     LDI   R21,LOW(RAMEND)
     OUT   SPL,R21

     CALL  I2C_INIT         ;initialize TWI module
     CALL  I2C_START        ;transmit START condition
     CALL  I2C_READ_STATUS  ;read status register
     CPI   R26, 0x08        ;was START transmitted correctly?
     BRNE  ERROR            ;else jump to error function
     LDI   R27, 0b11010000  ;SLA (11010000) + W(0)
     CALL  I2C_WRITE        ;write R27 to I2C bus
     CALL  I2C_READ_STATUS  ;read status register
     CPI   R26, 0x18        ;was SLA+W transmitted, ACK received?
     BRNE  ERROR            ;else jump to error function
     LDI   R27, 0b11110000  ;data to be transmitted
     CALL  I2C_WRITE        ;write R27 to I2C bus
     CALL  I2C_READ_STATUS  ;read status register
     CPI   R26, 0x28        ;was data transmitted, ACK received?
     BRNE  ERROR            ;else jump to error function
     CALL I2C_STOP          ;transmit STOP condition
HERE: RJMP  HERE            ;wait here forever
ERROR:                      ;you can type error handler here
     LDI   R21,0xFF
     OUT   DDRA,R21         ;Port A is output
     OUT   PORTA,R26        ;send error code to Port A
     RJMP  HERE             ;some error code
;*********************************************************
I2C_INIT:
     LDI   R21, 0
     OUT   TWSR,R21         ;set prescaler bits to zero
     LDI   R21, 0x48        ;move 0x48 into R21
     OUT   TWBR,R21         ;clock frequency is 50k (XTAL=50MHZ)
     LDI   R21, (1<<TWEN)   ;move 0x04 into R21
     OUT   TWCR,R21         ;enable the TWI
     RET

;*********************************************************
I2C_START:
     LDI   R21, (1<<TWINT)|(1<<TWSTA)|(1<<TWEN)
     OUT   TWCR,R21         ;transmit a START condition
WAIT1:
     IN    R21, TWCR        ;read control register into R21
     SBRS  R21, TWINT       ;skip next line if TWINT is 1
```

Program 18-14: Writing a Byte in Master Mode with Status Checking

```
      RJMP  WAIT1           ;jump to WAIT1 if TWINT is 0
      RET
;*********************************************************
I2C_WRITE:
      OUT   TWDR, R27       ;move the byte into TWDR
      LDI   R21, (1<<TWINT)|(1<<TWEN)
      OUT   TWCR, R21       ;configure TWCR to send TWDR
WAIT3:
      IN    R21, TWCR       ;read control register into R21
      SBRS  R21, TWINT      ;skip next line if TWINT is 1
      RJMP  WAIT3           ;jump to WAIT3 if TWINT is 0
      RET
;*********************************************************
I2C_STOP:
      LDI   R21, (1<<TWINT)|(1<<TWSTO)|(1<<TWEN)
      OUT   TWCR, R21       ;transmit STOP condition
      RET
;*********************************************************
I2C_READ_STATUS:
      IN    R26, TWSR       ;read status register into R21
      ANDI  R26, 0xF8       ;mask the prescaler bits
      RET
```

Program 18-14: Writing a Byte in Master Mode with Status Checking *(cont. from prev. page)*

Program 18-15 is the C version of Program 18-10 and shows how a master writes 11110000 to a slave with address `1101000`. The program checks the value of the status register in each step of the operation.

```
#include <avr/io.h>

void i2c_write(unsigned char data)
{
  TWDR = data ;
  TWCR = (1<< TWINT)|(1<<TWEN);
  while ((TWCR & (1 <<TWINT)) == 0);
}
//*********************************************************
void i2c_start(void)
{
  TWCR = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);
  while ((TWCR & (1 << TWINT)) == 0);
}
//*********************************************************
void i2c_showError(unsigned char er)
{
  DDRA = 0xFF;
  PORTA = er;
}
```

Program 18-15: Writing a Byte in Master Mode with Status Checking in C

```
//*************************************************************
unsigned char i2c_readStatus(void)
{
  unsigned char i = 0;
  i = TWSR & 0xF8;
  return i;
}
//*************************************************************
void i2c_stop()
{
  TWCR = (1<< TWINT)|(1<<TWEN)|(1<<TWSTO);
}
//*************************************************************
void i2c_init(void)
{
  TWSR=0x00;                //set prescaler bits  to zero
  TWBR=0x48;                //SCL frequency is 50K for XTAL = 8M
  TWCR=0x04;                //enable the TWI module
}
//*************************************************************

int main (void)
{
  unsigned char s = 0;
  i2c_init();
  i2c_start();              //transmit START condition
  s = i2c_readStatus();
  if (s != 0x08)
  {
     i2c_showError(s);
     return 0;
  }
  i2c_write(0b11010000);    //transmit SLA + W(0)
  s = i2c_readStatus();
  if (s != 0x18)
  {
     i2c_showError(s);
     return 0;
  }
  i2c_write(0b11110000);    //transmit data
  s = i2c_readStatus();
  if (s != 0x28)
  {
     i2c_showError(s);
     return 0;
  }
  i2c_stop();               //transmit STOP condition
  while(1);                 //stay here forever
  return 0;
}
```

Program 18-15: Writing a Byte in Master Mode with Status Checking in C *(continued)*

Programming of the AVR TWI in master receiver operating mode

The steps to program the AVR TWI to operate in master receiver mode are somewhat similar to the steps for programming for master transmitter mode. Figure 18-20 shows the steps for programming of the AVR TWI in master receiver mode. Here we focus on each step in more detail:

Initialization

To initialize the TWI module to operate in master operating mode, we should do the following steps:

1. Set the TWI module clock frequency by setting the values of the TWBR register and the TWPS bits in the TWSR register.
2. Enable the TWI module by setting the TWEN bit in the TWCR register to one.

Transmit START condition

To start data transfer in master operating mode, we must transmit a START condition. To transmit a START condition we should do the following steps:

1. Set the TWEN, TWSTA, and TWINT bits of TWCR to one. Setting the TWEN bit to one enables the TWI module. Setting the TWSTA bit to one tells the TWI module to initiate a START condition when the bus is free, and setting the TWINT bit to one clears the interrupt flag to initiate operation of the TWI module to transmit a START condition.
2. Poll the TWINT flag in the TWCR register to see when the START condition is completely transmitted.
3. When the TWINT flag is set to one, check the value of the status register to see if the START condition was successfully transmitted. Notice that you have to mask the two LSB bits of the status register to get rid of prescalers. If the status value is 0x08 it indicates that the START condition was successfully transmitted.

Send SLA + R

To send SLA + R, after transmitting a START condition, we should do the following steps:

1. Copy SLA + R to the TWDR.
2. Set the TWEN and TWINT bits of the TWCR register to one to start sending the byte.
3. Poll the TWINT flag in the TWCR register to see whether the byte has completely transmitted.
4. When the TWINT flag is set to one, you should check the value of status register to see if the SLA + R transmitted successfully. 0x40 means that the SLA + R transmitted and ACK was successfully received.

Receive data return NACK

If we want to receive only one byte of data, we should receive data and return NACK by doing the following steps:

1. Set the TWEN and TWINT bits of the TWCR register to one to start receiving a byte.
2. Poll the TWINT flag in the TWCR register to see whether a byte was com-

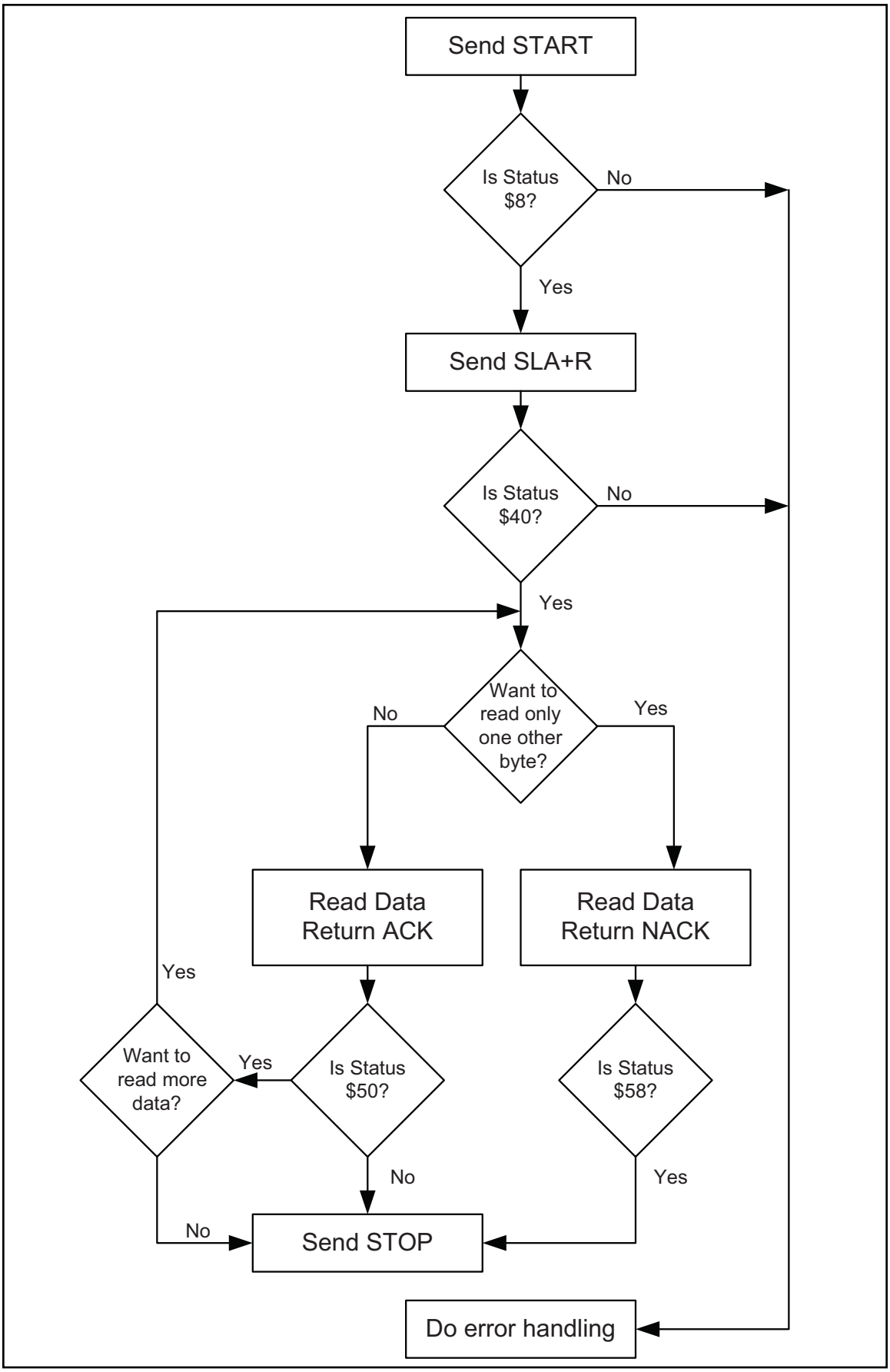

Figure 18-20. TWI Programming Steps of Master Receiver Mode with Checking of Flags

pletely received.
3. Copy the received byte from the TWDR.
4. When the TWINT flag is set to one, you should check the value of the status register to see if the byte was received successfully. 0x58 means that a byte of data was received and NACK returned successfully. After this step we should transmit a STOP condition.

Receive data and return ACK

If we want to receive more than one byte of data, we should receive data and return ACK by doing the following steps:

1. Set the TWEN, TWINT, and TWEA bits of the TWCR register to one to receive a byte of data and return ACK.
2. Poll the TWINT flag in the TWCR register to see when a byte has been received completely.
3. Copy the received byte from the TWDR.
4. When the TWINT flag is set to one, you should check the value of the status register to see if the byte was received successfully. 0x50 means that a byte of data was received and ACK returned successfully. Now you can repeat this step to receive one or more bytes of data, or you can run the "Receive Data Return NACK" function to receive only one other byte of data. Also, you can transmit a STOP condition to finish receiving data.

Transmit STOP condition

To stop data transfer, we must transmit a STOP condition. This is done by setting the TWEN, TWSTO, and TWINT bits of the TWCR register to one. Notice that we cannot poll the TWINT flag after transmitting a STOP condition.

Figure 18-21 shows the meanings of different values of the status register and possible responses to each of them in master receiver operating mode.

Initialization:	TWCR = 0x04 TWCR = (1<<TWEN)|(1<<TWINT)|(1<<TWSTA)	Enable TWI Transmit START condition.

Status	Meaning	Your Response	Next Action By TWI module
$8	START condition has been transmitted	TWDR = SLA + R (1) TWCR =(1<<TWEN)|(TWINT)	SLA + R will be transmitted ACK or NACK will be returned
$40	SLA + R has been transmitted. ACK has been received	TWCR =(1<<TWEN)|(TWINT)|(TWEA) OR TWCR =(1<<TWEN)|(TWINT)	DATA byte will be received ACK will be returned DATA byte will be received NACK will be returned
$48	SLA + R transmitted. NACK received	TWCR =(1<<TWEN)|(TWINT)|(TWSTO)	STOP condition will be transmitted
$50	Data byte has been received. ACK has been returned.	DATA = TWDR TWCR =(1<<TWEN)|(TWINT)|(TWEA) OR DATA = TWDR TWCR =(1<<TWEN)|(TWINT)	Another DATA byte will be received ACK will be returned Another DATA byte will be received NACK will be returned
$58	Data byte received. NACK ACK returned.	DATA = TWDR TWCR =(1<<TWEN)|(TWINT)|(TWSTO)	STOP condition will be transmitted

Figure 18-21. TWSR Register Values for Master Receiver Operating Mode

Program 18-15 shows how a master reads a byte from a slave with address `1101000` and displays the result on Port A. The program checks the value of the

status register in each step of the operation.

```
.INCLUDE "M32DEF.INC"
     LDI   R21,HIGH(RAMEND);set up stack
     OUT   SPH,R21
     LDI   R21,LOW(RAMEND)
     OUT   SPL,R21
     LDI   R21,0xFF
     OUT   DDRA,R21         ;Port A is output
     CALL  I2C_INIT         ;initialize TWI module
     CALL  I2C_START        ;transmit START condition
     CALL  I2C_READ_STATUS  ;read status register
     CPI   R26, 0x08        ;was start transmitted correctly?
     BRNE  ERROR            ;else jump to error function
     LDI   R27, 0b11010001 ;SLA (11010000) + R(1)
     CALL  I2C_WRITE        ;write R27 to I2C bus
     CALL  I2C_READ_STATUS  ;read status register
     CPI   R26, 0x40        ;was SLA+R transmitted, ACK received?
     BRNE  ERROR            ;else jump to error function
     CALL  I2C_READ
     CALL  I2C_READ_STATUS  ;read status register
     CPI   R26, 0x58        ;was data transmitted, ACK received?
     BRNE  ERROR            ;else jump to error function
     OUT   PORTA,R27
     CALL I2C_STOP          ;transmit STOP condition
HERE: RJMP  HERE            ;wait here forever
ERROR:RJMP HERE             ;you can type error handler here
;*********************************************************
I2C_INIT:
     LDI   R21, 0
     OUT   TWSR,R21         ;set prescaler bits to zero
     LDI   R21, 0x48        ;move 0x48 into R21
     OUT   TWBR,R21         ;SCL freq. is 50k for 8 MHz XTAL
     LDI   R21, (1<<TWEN)   ;move 0x04 into R21
     OUT   TWCR,R21         ;enable the TWI
     RET
;*********************************************************
I2C_START:
     LDI   R21, (1<<TWINT)|(1<<TWSTA)|(1<<TWEN)
     OUT   TWCR,R21         ;transmit a START condition
WAIT1:
     IN    R21, TWCR        ;read control register into R21
     SBRS  R21, TWINT       ;skip next line if TWINT is 1
     RJMP  WAIT1            ;jump to WAIT1 if TWINT is 0
     RET
;*********************************************************
I2C_WRITE:
     OUT   TWDR, R27        ;move the byte into TWDR
     LDI   R21, (1<<TWINT)|(1<<TWEN)
     OUT   TWCR, R21        ;configure TWCR to send TWDR
```

Program 18-16: TWI Reading a Byte in Master Mode with Status Checking

```
W3:   IN    R21, TWCR             ;read control register into R21
      SBRS  R21, TWINT            ;skip next line if TWINT is 1
      RJMP  W3                    ;jump to W3 if TWINT is 0
      RET
;*****************************************************
I2C_READ:
      LDI   R21,(1<<TWINT)|(1<<TWEN)
      OUT   TWCR, R21
W2:   IN    R21, TWCR             ;read control register into R21
      SBRS  R21, TWINT            ;skip next line if TWINT is 1
      RJMP  W2                    ;jump to W2 if TWINT is 0
      IN    R27, TWDR             ;read received data into R21
      RET
;*****************************************************
I2C_STOP:
      LDI   R21, (1<<TWINT)|(1<<TWSTO)|(1<<TWEN)
      OUT   TWCR, R21             ;transmit STOP condition
      RET
;*****************************************************
I2C_READ_STATUS:
      IN    R26, TWSR             ;read status register into R21
      ANDI  R26, 0xF8             ;mask the prescaler bits
      RET
```

Program 18-16: TWI Reading a Byte in Master Mode with Status Checking *(continued)*

Program 18-17 is the C version of Program 18-16.

```
#include <avr/io.h>
void i2c_showError(unsigned char er)
{
  DDRA = 0xFF;
  PORTA = er;
} //*********************************************************
unsigned char i2c_readStatus(void)
{
  unsigned char i = 0;
  i = TWSR & 0xF8;
  return i;
} //*********************************************************
void i2c_init(void)
{
  TWSR=0x00;               //set prescaler bits to zero
  TWBR=0x48;               //SCL frequency is 50K for XTAL=8M
  TWCR=0x04;               //enable the TWI module
} //*********************************************************
void i2c_start(void)
{
  TWCR = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);
  while ((TWCR & (1 << TWINT)) == 0);
}  //*********************************************************
```

Program 18-17: TWI Reading a Byte in Master Mode with Status Checking in C

```
void i2c_write(unsigned char data)
{
  TWDR = data;
  TWCR = (1<< TWINT)|(1<<TWEN);
  while ((TWCR & (1 <<TWINT)) == 0);
} //************************************************************
unsigned char i2c_read(unsigned char isLast)
{
  if (isLast == 0)          //if want to read more than 1 byte
    TWCR = (1<< TWINT)|(1<<TWEN)|(1<<TWEA);
  else                      //if want to read only one byte
    TWCR = (1<< TWINT)|(1<<TWEN);
  while ((TWCR & (1 <<TWINT)) == 0);
  return TWDR;
} //************************************************************
void i2c_stop()
{
  TWCR = (1<< TWINT)|(1<<TWEN)|(1<<TWSTO);
} //************************************************************
int main (void)
{
  DDRA = 0xFF;              //Port A is output
  unsigned char s,i;
  i2c_init();
  i2c_start();              //transmit START condition
  s = i2c_readStatus();
  if (s != 0x08)
  {
     i2c_showError(s);
     return 0;
  }
  i2c_write(0b11010001);    //transmit SLA + R(1)
  s = i2c_readStatus();
  if (s != 0x40)
  {
     i2c_showError(s);
     return 0;
  }
  i=i2c_read(1);
  s = i2c_readStatus();
  if (s != 0x58)
  {
     i2c_showError(s);
     return 0;
  }
  PORTA= i;                 //show the byte on Port A
  i2c_stop();               //transmit STOP condition
  while(1);                 //stay here forever
  return 0;
}
```

Program 18-17: TWI Reading a Byte in Master Mode with Status Checking in C *(continued)*

Programming of the AVR TWI in slave transmitter operating mode

Before programming the AVR to operate in slave mode, there are some points that we must pay attention to. As we mentioned before, the slave device, regardless of whether it is receiver or transmitter, does not generate the clock pulse. To control the clock rate and let the software to complete its job, the slave device uses clock stretching. The slave device does not start or stop a transmission; it listens to the bus and replies when it is addressed by a master device.

In the slave transmitter mode, one or more bytes of data are transmitted from the slave to a master receiver. The following steps show the transmission of one or more bytes of data in slave transmitter mode.

Initialization

To initialize the TWI module to operate in slave operating mode, we should do the following steps:

1. Set the TWAR. As we mentioned before, the upper seven bits of TWAR are the slave address. It is the address to which the TWI will respond when addressed by a master. The eighth bit is TWGCE. If you set this bit to one, the TWI will respond to the general call address ($00); otherwise, it will ignore the general call address.
2. Enable the TWI module by setting the TWEN bit in the TWCR register to one.
3. Set the TWEN and TWEA bits of TWCR to one to enable the TWI and acknowledge generation.

Wait to be addressed for read

In slave mode, the TWI hardware waits until it is addressed by its own slave address (or the general call address, if enabled) followed by the R/W bit, and then sets the TWINT flag and updates the status register. If the R/W bit is zero (write), it means that the slave should operate in slave receiver mode; otherwise, the slave should operate in slave transmitter mode. Notice that you can not directly read the value of the R/W bit. Instead you should read the value of the status register. Next, we will show how to wait to be addressed by a master device.

1. Poll the TWINT flag in the TWCR register to see whether a byte has received completely.
2. When the TWINT flag is set to one, you should check the value of the status register to see if the SLA + R is received successfully. $A8 means that the SLA + R was received and ACK returned successfully.

Now if you want to transmit only one byte of data you should run the "Send Data and Wait for NACK" function. Otherwise, if you want to send more than one byte of data you should run the "Send Data and Wait for ACK" function. Next we will examine each function in detail.

Send data and wait for ACK

In slave transmitter mode, if you want to transmit more than one byte of data you should send a byte of data and wait for ACK by doing the following steps:

1. Copy the byte of data to the TWDR.
2. Set the TWEN, TWINT, and TWEA bits of the TWCR register to one to send

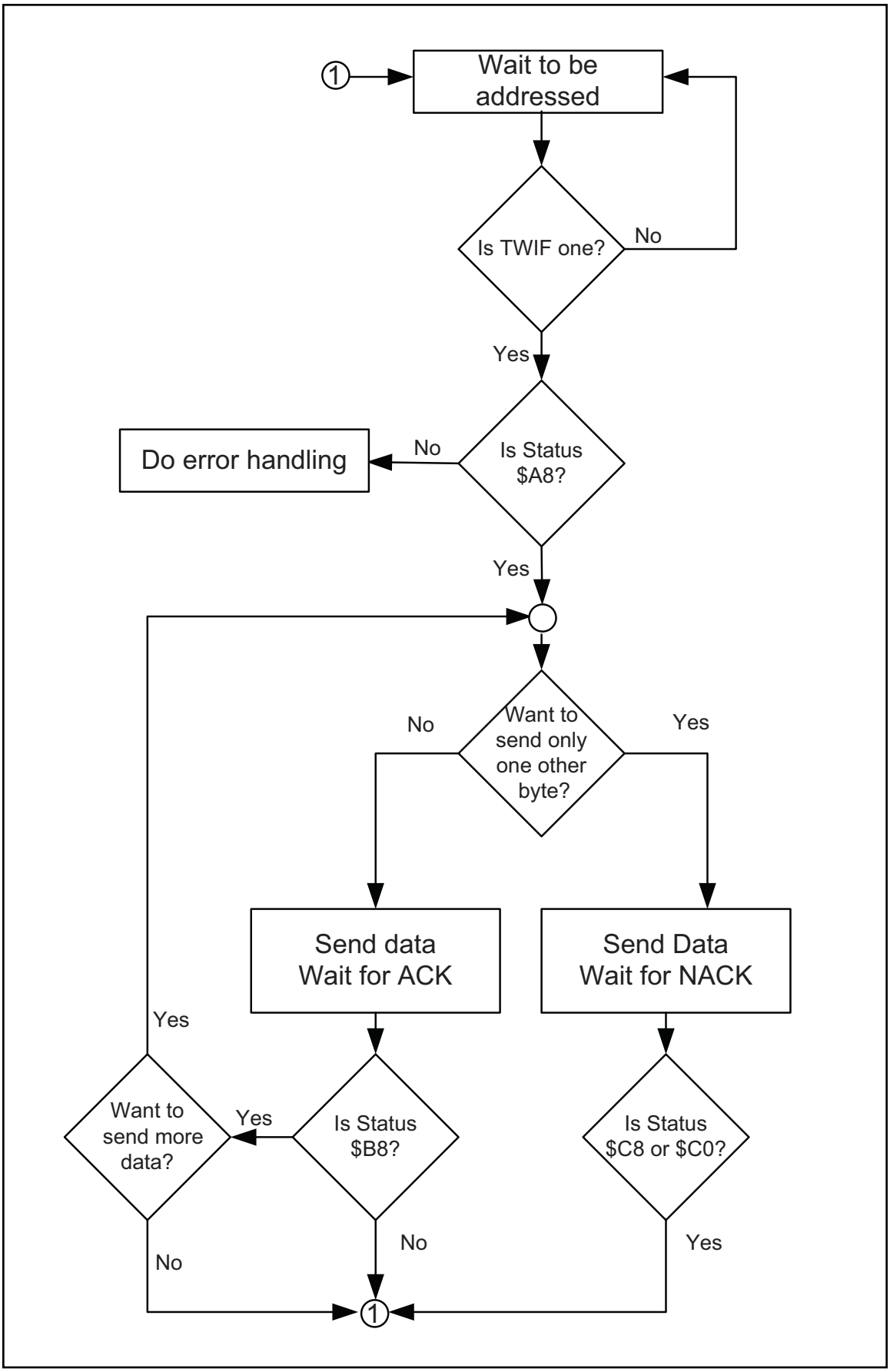

Figure 18-22. TWI Programming Steps of Slave Transmitter Mode with Checking of Flags

a byte of data and wait for ACK.

3. Poll the TWINT flag in the TWCR register to see whether the byte transmitted completely.
4. When the TWINT flag is set to one, you should check the value of the status register to see if the data transmitted successfully and the value of ACK was as expected. Notice that NACK does not necessarily indicate an error; it may indicate that no more data needs to be transmitted. If the status value indicates that NACK was received ($0C), it means that the current transmission section is finished and you should start from the beginning. If the status value indicates that ACK was received (0xC8), you can either repeat this function to transmit more than one byte of data or you can run the "Send Data and Wait for NACK" function to transmit only one byte of data.

Send data and wait for NACK

In slave transmitter mode, to transmit another byte of data you should send a byte of data and wait for NACK by doing the following steps:

1. Copy the byte of data to the TWDR.
2. Set the TWEN and TWINT bits of the TWCR register to one to send a byte and wait for NACK.
3. Poll the TWINT flag in the TWCR register to see when the byte has been transmitted completely.
4. When the TWINT flag is set to one, you should check the value of the status register. If the status value is $0C, it indicates that NACK has been received. If the value of status register is $C8, it means that ACK was received. In both cases you have to go to the "Wait to be addressed" mode because you have not set the TWEA bit in step 2 saying that you want to transmit only one other byte of data.

Notice that in most applications you can use the "Send Data and Wait for ACK" function instead of the "Send Data and Wait for NACK" function. We rec-

Initialization:	TWCR = 0x04 TWAR = the address of Slave TWCR = (1<<TWEN)|(1<<TWIF)|(1<<TWEA)	Enable TWI Set the slave address Enable Acknowledging by slave

Status	Meaning	Your Response	Next Action By TWI module
$A8	Own SLA+R received ACK returned	TWDR = DATA TWCR =(1<<TWEN)|(TWINT)|(TWEA)	DATA byte will be transmitted Wait for ACK
		OR	
		TWDR = DATA TWCR =(1<<TWEN)|(TWINT)	DATA byte will be transmitted Wait for NACK
$B8	Data has been transmitted ACK received	TWDR = DATA TWCR =(1<<TWEN)|(TWINT)|(TWEA)	DATA byte will be transmitted Wait for ACK
		OR	
		TWDR = DATA TWCR =(1<<TWEN)|(TWINT)	DATA byte will be transmitted Wait for NACK
$C0	Data has been transmitted NACK received	TWCR =(1<<TWEN)|(TWINT)|(TWEA)	Start from beginning and wait to be addressed
		OR	
		TWCR =(1<<TWEN)|(TWINT)	Start from beginning but do not respond to Its address (Sleep)
$C8	Data transmitted ACK received but you wanted NACK (TWEA was 0 in last command)	TWCR =(1<<TWEN)|(TWINT)|(TWEA)	Start from beginning and wait to be addressed
		OR	
		TWCR =(1<<TWEN)|(TWINT)	Start from beginning but do not respond to Its address (Sleep)

Figure 18-23. TWSR Register Values for Slave Transmitter Operating Mode

ommend that you use the first one.

Program 18-18 shows how to initialize the TWI module to operate in slave transmitter mode. In this program the TWI module listens to the bus and waits to be addressed by a master device. Then it transmits the letter 'G' to the master device.

```
.INCLUDE "M32DEF.INC"

     LDI   R21,HIGH(RAMEND);set up stack
     OUT   SPH,R21
     LDI   R21,LOW(RAMEND)
     OUT   SPL,R21

     CALL  I2C_INIT         ;initialize the TWI module as slave
     CALL  I2C_LISTEN       ;listen to the bus to be addressed
     CALL I2C_READ_STATUS   ;read the status value into R26
     CPI   R26, 0xA8        ;addressed as slave tranmitter ?
     BRNE  ERROR            ;else jump to error function
     LDI   R27, 'G'         ;load 'G' into R21
     CALL  I2C_WRITE
     CALL I2C_READ_STATUS   ;read the status value into R26
     CPI   R21, 0xc0        ;was data transmitted, NACK received?
     BRNE  ERROR            ;else jump to error function

HERE:
     RJMP  HERE             ;wait here forever
ERROR:                      ;you can type error handler here
     LDI   R21,0xFF
     OUT   DDRA,R21         ;Port A is output
     OUT   PORTA,R26
     RJMP  HERE
;*********************************************************

I2C_INIT:
     LDI   R21, 0x10        ;load 00010000 into R21
     OUT   TWAR,R21         ;set address register
     LDI   R21, (1<<TWEN)   ;move 0x04 into R21
     OUT   TWCR,R21         ;enable the TWI
     LDI   R21, (1<<TWINT)|(1<<TWEN)|(1<<TWEA)
     OUT   TWCR,R21         ;enable TWI and ACK(can't be ignored)
     RET
;*********************************************************

I2C_LISTEN:
W1:
     IN    R21, TWCR        ;read control register into R21
     SBRS  R21, TWINT       ;skip next intruction if TWINT is 1
     RJMP  W1               ;jump to W1 if TWINT is 0
     RET
```

Program 18-18: Writing a Byte in Slave Mode with Status Checking

```
;*********************************************************
I2C_WRITE:

     OUT   TWDR, R27        ;move R21 to TWDR
     LDI   R21, (1<<TWINT)|(1<<TWEN)
     OUT   TWCR, R21        ;configure TWCR to send TWDR
W2:
     IN    R21, TWCR        ;read control register into R21
     SBRS  R21, TWINT ;skip next intruction if TWINT is 1
     RJMP  W2               ;jump to W2 if TWINT is 0
     RET

;*********************************************************
I2C_READ_STATUS:
     IN    R26, TWSR        ;read status register into R21
     ANDI  R26, 0xF8        ;mask the prescaler bits
     RET
```

Program 18-18: Writing a Byte in Slave Mode with Status Checking *(cont. from prev. page)*

Program 18-19 is the C version of Program 18-18. Program 18-19 shows how to initialize the TWI module to operate in slave transmitter mode. In Program 18-19 the TWI module listens to the bus and waits to be addressed by a master device. Then it transmits the letter 'G' to the master device.

```
#include <avr/io.h>                    //standard AVR header

void i2c_showError(unsigned char er)
{
  DDRA = 0xFF;
  PORTA = er;
} //*************************************************************

unsigned char i2c_readStatus(void)
{
  unsigned char i = 0;
  i = TWSR & 0xF8;
  return i;
} //*************************************************************

void i2c_initSlave(unsigned char slaveAddress)
{
  TWCR = 0x04;                         //enable TWI module
  TWAR = slaveAddress;                 //set the slave address
  TWCR = (1<<TWINT)|(1<<TWEN)|(1<<TWEA);//init TWI module
}
```

Program 18-19: Writing a Byte in Slave Mode with Status Checking in C

```
//*************************************************************

void i2c_send(unsigned char data)
{
  TWDR = data;                          //copy data to TWDR
  TWCR = (1<< TWINT)|(1<<TWEN);         //start transmission
  while ((TWCR & (1 <<TWINT))==0);      //wait to complete
}

//*************************************************************

void i2c_listen()
{
  while ((TWCR & (1 <<TWINT))==0);      //wait to be addressed
}

//*************************************************************

int main (void)
{
  i2c_initSlave(0x10);                  //init TWI module as
                                        //slave with address
                                        //0b0001000 and do not
                                        //accept general call
  i2c_listen();                         //listen to be addressed

  unsigned char s,i;
  s = i2c_readStatus();
  if (s != 0xA8)
  {
     i2c_showError(s);
     return 0;
  }
  i2c_send('G');
  s = i2c_readStatus();
  if (s != 0xC0)
  {
     i2c_showError(s);
     return 0;
  }

  while(1);                             //stay here forever
  return 0;

}
```

Program 18-19: Writing a Byte in Slave Mode with Status Checking in C *(continued)*

Programming of the AVR TWI in slave receiver operating mode

In the slave receiver mode, one or more bytes of data are transmitted from a master transmitter to the slave receiver. The following steps show the functions needed to receive one or more bytes of data in slave receiver mode.

Initialization

To initialize the TWI module to operate in slave operating mode, we should do the following steps:

1. Set the TWAR. As we mentioned before, the upper seven bits of TWAR are the slave address. It is the address to which the Two-wire Serial Interface will respond when addressed by a master. The eighth bit is TWGCE. If you set this bit to one, the TWI will respond to the general call address ($00); otherwise, it will ignore the general call address.
2. Enable the TWI module by setting the TWEN bit in the TWCR register to one.
3. Set the TWEN and TWEA bits of TWCR to one to enable the TWI and acknowledge generation.

Wait to be addressed for write

In slave mode, we should do the following steps to wait to be addressed by a master for a write operation.

1. Poll the TWINT flag in the TWCR register to see when a byte has been received completely.
2. When the TWINT flag is set to one, we should check the value of the status register to see if the SLA + W was received successfully. $60 or $70 (for general call) means that the SLA + W was received and ACK returned successfully.

Now if you want to receive only one byte of data you should run the "Receive Data and Return NACK" function. Otherwise, if you want to send more than one byte of data you should run the "Receive Data and Return ACK" function. Next, we will examine each function in detail.

Receive data and Return ACK

In slave receiver mode, if you want to receive more than one byte of data you should receive a byte of data and return ACK by doing the following steps:

1. Set the TWEN, TWINT, and TWEA bits of the TWCR register to one to receive a byte and return ACK.
2. Poll the TWINT flag in the TWCR register to see when a byte has been received completely.
3. When the TWINT flag is set to one, you should check the value of the status register to see if the data was received successfully and ACK was returned. If the status value is $80 or $90 (for general call), it means that a byte of data has been received and ACK was returned. You can either repeat this function to receive more than one bytes of data or you can run the "Receive Data and Return NACK" function to receive only one byte of data.
4. Copy the received byte from the TWDR.

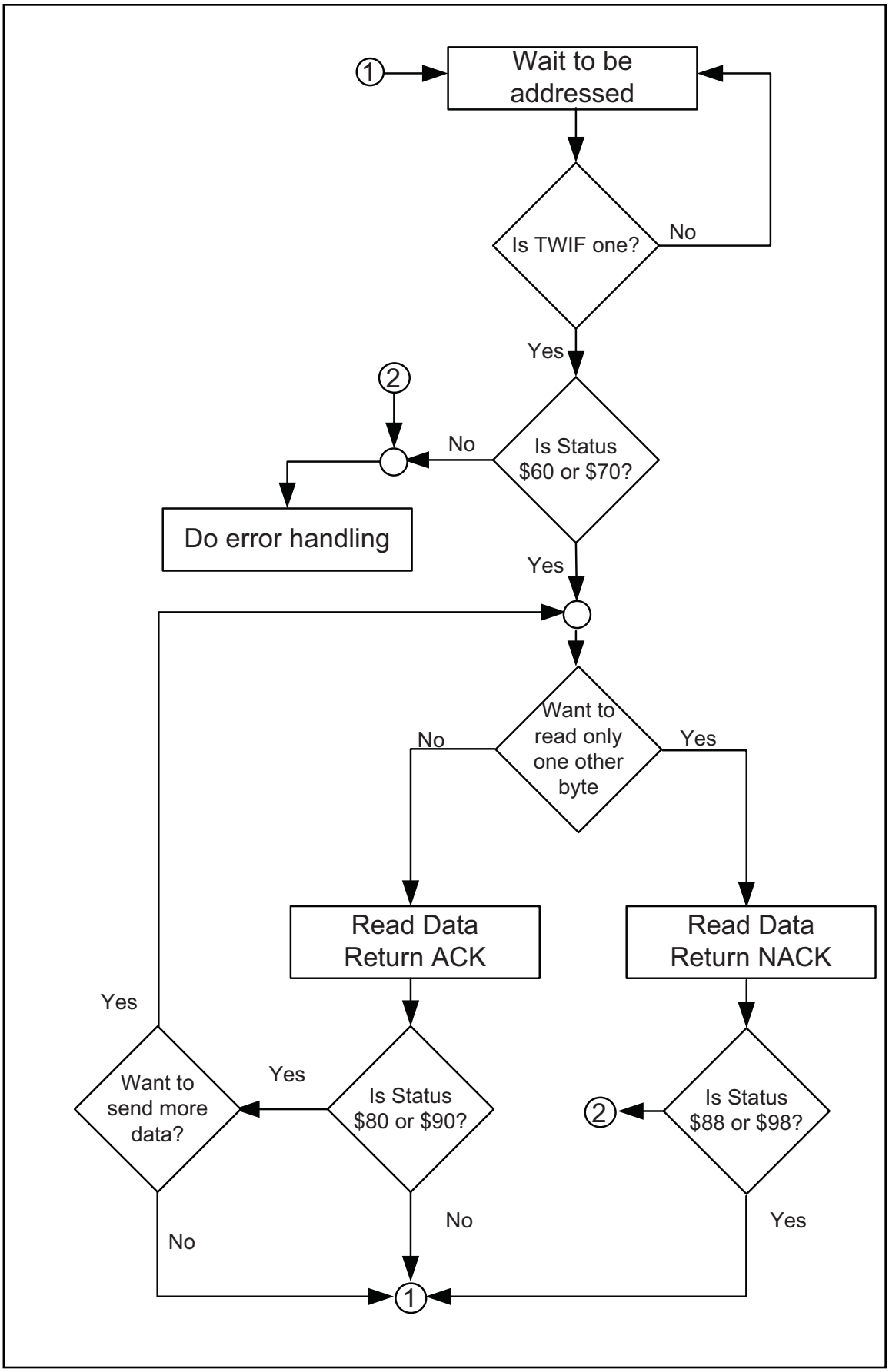

Figure 18-24. TWI Programming Steps of Slave Receiver Mode with Checking of Flags

Receive data and return NACK

In slave receiver mode, if you want to receive one byte of data you should receive the byte of data and return NACK by doing the following steps:

1. Set the TWEN and TWINT bits of the TWCR register to one to receive a byte and return NACK.
2. Poll the TWINT flag in the TWCR register to see when a byte has been received completely.
3. When the TWINT flag is set to one, you should check the value of the status register to see if the data was received successfully and NACK was returned. If the status value is $88 or $98 (for general call), it means that a byte of data was received and NACK was returned.
4. Copy the received byte from the TWDR.

Initialization:	TWAR = the address of Slave TWCR = 0x04 TWCR = (1<<TWEN)|(1<<TWIF)|(1<<TWEA)	Enable TWI Set the slave address Enable Acknowledging by slave

Status	Meaning	Your Response	Next Action By TWI module
$60 ($70 for General Call)	Own SLA+W received ACK returned	TWCR =(1<<TWEN)|(TWINT)|(TWEA)	DATA byte will be received ACK will be returned
		OR	
		TWCR =(1<<TWEN)|(TWINT)	DATA byte will be received NACK will be returned
$80 ($90 for General Call)	Data has been received ACK returned	DATA = TWDR TWCR =(1<<TWEN)|(TWINT)|(TWEA)	DATA byte will be received ACK will be returned
		OR	
		DATA = TWDR TWCR =(1<<TWEN)|(TWINT)	DATA byte will be received NACK will be returned
$88 ($98 for General Call)	Data has been received NACK returned	DATA = TWDR TWCR =(1<<TWEN)|(TWINT)|(TWEA)	Start from beginning and wait to be addressed
		OR	
		DATA = TWDR TWCR =(1<<TWEN)|(TWINT)	Start from beginning but do not respond to Its address (Sleep)
$A0	STOP or REPEATED START condition has been received	TWCR =(1<<TWEN)|(TWINT)|(TWEA)	Start from beginning and wait to be addressed
		OR	
		TWCR =(1<<TWEN)|(TWINT)	Start from beginning but do not respond to Its address (Sleep)

Figure 18-25. TWSR Register Values for Slave Receiver Operating Mode

Program 18-20 shows how to initialize the TWI module to operate in slave receiver mode. This program receives a byte of data and displays it on Port A after being addressed by a master device.

```
.INCLUDE "M32DEF.INC"

     LDI   R21,HIGH(RAMEND);set up stack
     OUT   SPH,R21
     LDI   R21,LOW(RAMEND)
     OUT   SPL,R21

     LDI   R21, 0xFF        ;move 0xFF into R21
     OUT   DDRA,R21         ;set PORTA as ouput

     CALL  I2C_INIT         ;initialize the TWI module as slave
```

Program 18-20: Reading a Byte in Slave Mode with Status Checking

```
      CALL  I2C_LISTEN       ;listen to the bus to be addressed
      CALL  I2C_READ_STATUS
      CPI   R26, 0x60        ;addressed as slave receiver?
      BRNE  ERROR            ;else jump to error function
      CALL  I2C_READ         ;read a byte and copy it to R27
      CALL  I2C_READ_STATUS
      CPI   R26, 0x80        ;addressed as slave receiver?
      BRNE  ERROR            ;else jump to error function
      OUT   PORTA,R27        ;copy R27 to PORTA

HERE:
      RJMP  HERE             ;wait here forever
ERROR:
      RJMP HERE
;*********************************************************

I2C_INIT:
      LDI   R21, 0x10        ;load 00010000 into R21
      OUT   TWAR,R21         ;set address register
      LDI   R21, (1<<TWEN)   ;move 0x04 into R21
      OUT   TWCR,R21         ;enable the TWI
      LDI   R21, (1<<TWINT)|(1<<TWEN)|(1<<TWEA)
      OUT   TWCR,R21         ;enable TWI and ACK(can't be ignored)
      RET

;*********************************************************

I2C_LISTEN:
W1:
      IN    R21, TWCR        ;read control register into R21
      SBRS  R21, TWINT       ;skip next intruction if TWINT is 1
      RJMP  W1               ;jump to W1 if TWINT is 0
      RET

;*********************************************************

I2C_READ:
      LDI   R21, (1<<TWINT)|(1<<TWEN)|(1<<TWEA)
      OUT   TWCR, R21        ;configure TWCR to receive TWDR
W2:   IN    R21, TWCR        ;read control register into R21
      SBRS  R21, TWINT       ;skip next line if TWINT is 1
      RJMP  W2               ;jump to W2 if TWINT is 0
      IN    R27,TWDR         ;move received data into R21
      RET

;*********************************************************
I2C_READ_STATUS:
      IN    R26, TWSR        ;read status register into R21
      ANDI  R26, 0xF8        ;mask the prescaler bits
      RET
```

Program 18-20: Reading a Byte in Slave Mode with Status Checking *(cont. from prev. page)*

Program 18-21 is the C version of Program 18-20. This program receives a byte of data and displays it on Port A after being addressed by a master device.

```
#include <avr/io.h>                    //standard AVR header

void i2c_showError(unsigned char er)
{
  DDRA = 0xFF;
  PORTA = er;
}

//*********************************************************

unsigned char i2c_readStatus(void)
{
  unsigned char i = 0;
  i = TWSR & 0xF8;
  return i;
}

//*********************************************************

void i2c_initSlave(unsigned char slaveAddress)
{
  TWCR = 0x04;                         //enable TWI module
  TWAR = slaveAddress;                 //set the slave address
  TWCR = (1<<TWINT)|(1<<TWEN)|(1<<TWEA);//init. TWI module
}
//*********************************************************

unsigned char i2c_receive(unsigned char isLast)
{
  if (isLast == 0)          //if want to read more than 1 byte
    TWCR = (1<< TWINT)|(1<<TWEN)|(1<<TWEA);
  else                      //if want to read only one byte
    TWCR = (1<< TWINT)|(1<<TWEN);

  while ((TWCR & (1 <<TWINT))==0);    //wait to complete
  return (TWDR);
}

//*********************************************************

void i2c_listen()
{
  while ((TWCR & (1 <<TWINT))==0);    //wait to be addressed
}

//*********************************************************
```

Program 18-21: Reading a Byte in Slave Mode with Status Checking in C

```
int main (void)
{
  DDRA = 0xFF;
   i2c_initSlave(0x10);                //init. TWI module as
                                       //slave with address
                                       //0b0001000 and do not
                                       //accept general call
  i2c_listen();                        //listen to be addressed

  unsigned char s,i;
  s = i2c_readStatus();
  if (s != 0x60)
  {
     i2c_showError(s);
     return 0;
  }
  i=i2c_receive(0);
  s = i2c_readStatus();
  if (s != 0x80)
  {
     i2c_showError(s);
     return 0;
  }
  PORTA = i;
  while(1);                            //stay here forever
  return 0;
}
```

Program 18-21: Reading a Byte in Slave Mode with Status Checking in C *(continued)*

Review Questions

1. True or false. We can ignore checking the status register when there is more than one master on the bus.
2. True or false. We can enable the TWI module and generate aSTART condition at the same time.
3. How can a slave device read the value of the R/W bit when it is being addressed by a master device?
4. True or false. We can check the status register to see if a STOP condition has been transmitted successfully.
5. What is the value of the status register when SLA + W is received and ACK has been returned?
6. What is the value of the status register when SLA + W is transmitted and ACK has been received?
7. What is the value of the status register when SLA + R is received and ACK has been returned?
8. What is the value of the status register when SLA + W is transmitted and ACK has been received?