Two Wire Interface

Another popular serial peripheral interface bus
- More flexible than SPI
- Master and slave modes supported
- 7-bit slave address
- 400khz data xfer speed
- Bidirectional, open-drain bus (device pulls down, resistors pull up)
- Two wires, SCL (clock) and SDA (data)

Typical TWI bus configuration
Two Wire Interface

A TWI transmission consists of
- A Start condition
- An address packet consisting of
  - Read/Write indication and
  - Slave acknowledge, \((SLA+RW)\)
- One or more data packets
- A Stop condition

A *Start* condition initiates a transmission by a master.

Between *Start* and *Stop* conditions, the bus is busy and no other masters should try to initiate a transfer.

A *Start* condition is signaled by a falling edge of SDA while SCL is high.
Two Wire Interface

Address packet
- Address packet is 9 bits long
- MSB first
  - Address “000 0000” is reserved for broadcast mode
- 7 address bits (driven by master)
  - 1 read/write control bit (driven by master)
  - 1 acknowledge bit (driven by addressed slave)
Two Wire Interface

Data packet
- All data packets are 9 bits long
  - MSB first
  - One data byte plus an acknowledge

-During a transfer, Master generates SCL, Receiver acknowledges

-Acknowledge (ACK): Slave pulls down SDA in the 9\textsuperscript{th} SCL cycle

-Not Acknowledge (NACK): Slave does not pull down SDA in 9\textsuperscript{th} cycle
Two Wire Interface

STOP condition

- A **Stop** condition completes a transmission by a master.

- A **Stop** condition is signaled by a rising edge of SDA while SCL is high.
Two Wire Interface

Special cases

-A **Repeated Start** occurs when the master initiates a new transfer without relinquishing control of the bus. Otherwise, this is just like another Start.

-Clock stretching: If a Master is issuing a clock that is too fast, the Slave can extend the clock by extending the low period of SCL.

-Multiple packets: Several data bytes may be sent between $SLA+RW$ and Stop conditions.
Two Wire Interface

TWI Registers

-TWBR (bit rate register)
-Controls the period of SCL when the TWI module is operating in Master mode

-TWAR (address register)
-Used when TWI module is receiving data to identify its address.

-TWCR (control register)
-Controls operation of the TWI unit
-Used to generate START, STOP, ACK pulse
-Also enables TWI operation including interrupt enables

-TWSR (status register)
-Reflects the status of the TWI logic bus via codes.
-Holds the prescale value for the TWI SCL pulse generator

-TWDR (data register)
-In transmit mode, it holds the data to send
-In receive mode, it holds the data received
Two Wire Interface

TWI Interrupts

-The TWINT flag in the control register must be cleared by SW
  - it is not cleared by the ISR like other interrupt sources

-TWINT indicates that software needs to take care of something on the TWI bus.

-One cycle after TWINT asserts, TWSR will hold a status code identifying the event that caused the interrupt.

-While TWINT is set, SCL is stretched low and the TWI bus is stalled to allow software time to take care of business.

-When TWINT is reset, the bus begins operations again.

-When TWAR, TWSR or TWDR are changed, TWINT must be logic “1”.
Two Wire Interface

TWI Interrupts (cont)

-TWINT is set when the TWI unit:
  - finishes sending a Start/Repeated Start
  - finishes sending a SLA+RW
  - finishes sending an address byte
  - looses arbitration
  - has been addressed by its slave address or a general call
  - has received a data byte
  - receives a Stop or Repeated Start while being addressed as a slave
  - has a bus error due to illegal Start or Stop conditions

-TWINT is not set after the TWI unit:
  - sends the STOP condition
Two Wire Interface

Using TWI (assuming its enabled)

```c
while(!(TWCR & (1<<TWINT)))

if((TWSR & 0xF8) != START)
    error();
TWCR = (1<<TWINT) | (1<<TSWTA);
TWDR = SLA_W;
TWCR = (1<<TWINT)

if((TWSR & 0xF8) != MT_SLA_ACK)
    error();
TWCR = (1<<TWINT)

if((TWSR & 0xF8) != START)
    error();
TWCR = (1<<TWINT) | (1<<TWSTO);
```

---

**Diagram:**

1. Application writes to TWCR to initiate transmission of START.
2. TWINT set. Status code indicates START condition sent.
3. Check TWSR to see if START was sent. Application loads SLA+W into TWDR, and loads appropriate control signals into TWCR, making sure that TWINT is written to one, and TWSTA is written to zero.
4. TWINT set. Status code indicates SLA+W sent, ACK received.
5. Check TWSR to see if SLA+W was sent and ACK received. Application loads data into TWDR, and loads appropriate control signals into TWCR, making sure that TWINT is written to one.
6. TWINT set. Status code indicates data sent, ACK received.
7. Check TWSR to see if data was sent and ACK received. Application loads appropriate control signals to send STOP into TWCR, making sure that TWINT is written to one.

**TWI bus:**

- START
- SLA+W
- A
- Data
- A
- STOP

**TWI Hardware Action:**

- while(!(TWCR & (1<<TWINT)))
- while(!(TWCR & (1<<TWINT)))
- while(!(TWCR & (1<<TWINT)))

**Indicates TWINT set:**
Two Wire Interface

A typical TWI device:

The device is address according to the manufacturers fixed address plus the address pin.

<table>
<thead>
<tr>
<th>Part Number</th>
<th>Address Pin</th>
<th>Device Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>LM73-0</td>
<td>Float</td>
<td>1001 000</td>
</tr>
<tr>
<td></td>
<td>Ground</td>
<td>1001 001</td>
</tr>
<tr>
<td></td>
<td>V_DD</td>
<td>1001 010</td>
</tr>
<tr>
<td>LM73-1</td>
<td>Float</td>
<td>1001 100</td>
</tr>
<tr>
<td></td>
<td>Ground</td>
<td>1001 101</td>
</tr>
<tr>
<td></td>
<td>V_DD</td>
<td>1001 110</td>
</tr>
</tbody>
</table>

The pointer register is used to read or write the other registers.

Writing to the chip is sometimes a two step process. But, the LM73 pointer register remembers the last pointed to register.
Two Wire Interface

LM73: one byte write

- send device address
- send pointer address
- send data value to register

<table>
<thead>
<tr>
<th>P7</th>
<th>P6</th>
<th>P5</th>
<th>P4</th>
<th>P3</th>
<th>P2</th>
<th>P1</th>
<th>P0</th>
<th>Bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7:3</td>
<td>Not Used</td>
<td>Must write zeros only.</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2:0</td>
<td>Register Select</td>
<td>Pointer address. Points to desired register. See table below.</td>
</tr>
</tbody>
</table>

- **P2 P1 P0 REGISTER (Note 13)**
  - 0 0 0: Temperature
  - 0 0 1: Configuration
  - 0 1 0: $T_{HIGH}$
  - 0 1 1: $T_{LOW}$
  - 1 0 0: Control / Status
  - 1 1 1: Identification

**Data**

- 0x90
- 0x00 - temp
- 0x01 - config
- 0x04 - ctrl/stat

(a) Typical 1-Byte Write
Two Wire Interface

LM73: two byte read with preset pointer

NACK by master indicates end of reading.

(a) Typical Read from a 2-Byte Register with Preset Pointer
Two Wire Interface

Gotcha’s with the LM73:

<table>
<thead>
<tr>
<th>t_TIMEOUT</th>
<th>SMBDAT and SMBCLK Time Low for Reset of Serial Interface (Note 11)</th>
<th>PULL-UP = 3.9 kΩ</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>15 ms (min)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>45 ms (max)</td>
</tr>
</tbody>
</table>

Don’t take a long time doing stuff while TWINT is asserted.

<table>
<thead>
<tr>
<th>t_BUF</th>
<th>SMBus Free Time Between Stop and Start Conditions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.2</td>
<td>μs (min)</td>
<td></td>
</tr>
</tbody>
</table>

You may need a short pause between stop an start conditions. When using 400khz clock, I didn’t. I did with slower clocks!
Two Wire Interface

TWI coding

Strongly suggest using twi.h:  

```c
#include <utils/twi.h>
```

The #defines there will help make your code readable.

```c
//start condition transmitted
#define TW START 0x08
//repeated start condition transmitted
#define TW_REP_START 0x10
//SLA+W transmitted, ACK received
#define TW_MT_SLA_ACK 0x18
//SLW_W transmitted, NACK received
#define TW_MT_SLA_NACK 0x20
```

.............
Two Wire Interface

TWI coding

Strongly suggest using your own defines too:

```c
//LM73 Addresses on the I2C bus
//Using LM73-0, address pin floating (datasheet pg. 9)
#define LM73_ADDRESS 0x90
#define LM73_WRITE (LM73_ADDRESS | TW_WRITE) //LSB is a zero to write
#define LM73_READ  (LM73_ADDRESS | TW_READ)  //LSB is a one to read

//define the codes for actions to occur
#define TWCR_START  0xA4 //send start condition
#define TWCR_STOP   0x94 //send stop condition
#define TWCR_RACK   0xC4 //receive byte and return ack to slave
#define TWCR_RNACK  0x84 //receive byte and return nack to slave
#define TWCR_SEND   0x84 //pokes the TWINT flag in TWCR and TWEN

#define LM73_PTR_TEMP         0x00  //LM73 temperature address
#define LM73_PTR_CONFIG       0x01  //LM73 configuration address
#define LM73_PTR_CTRL_STATUS  0x04  //LM73 ctrl and stat register
```
Two Wire Interface

TWI coding
Strongly suggest well structured code with inline debug as needed:

```c
TWCR = TWCR_START;             //send start condition
while(!(TWCR & (1<<TWINT))){}  //wait for start condition to transmit
if(!(TW_STATUS == TW_START)){
    string2lcd(error1); return(1);}//check start status

TWDR = LM73_WRITE;               //send device addr, write bit set
TWCR = TWCR_SEND;                //poke TWINT to send address
while(!(TWCR & (1<<TWINT))){}    //wait for LM73 address to go out
if(TW_STATUS != TW_MT_SLA_ACK){
    string2lcd(error2); return(1);}//check status reg for SLA+W ACK

TWDR = LM73_PTR_TEMP;            //send pointer address to LM73
TWCR = TWCR_SEND;                //poke TWINT to send address
while(!(TWCR & (1<<TWINT))){}    //wait for LM73 data byte 1 to go out
if(TW_STATUS != TW_MT_DATA_ACK){
    string2lcd(error3); return(1);}//check ack status
TWCR = TWCR_STOP;                 //finish transaction
return(0);                        //return success value
```
Two Wire Interface

Reading the temperature is a bit funny...

```c
uint8_t rd_temp(){
    TWCR = TWCR_START;           //send start condition
    TWDR = LM73_READ;            //send device addr, read bit set
    TWCR = TWCR_RACK;                 //receive data byte, return ACK
    while(!(TWCR & (1<<TWINT))){}     //wait for data byte to come in
    if(TW_STATUS != TW_MR_DATA_ACK){return(1);}  //byte 1 read failure
    lm73_temp_high = TWDR;            //store temp high byte

    TWCR = TWCR_RNACK;                //recv temp low byte, return NACK
    while(!(TWCR & (1<<TWINT))){}     //wait for data byte to come in
    if(TW_STATUS != TW_MR_DATA_NACK){return(1);} //byte 2 read failure
    lm73_temp_low = TWDR;             //store temp low byte
    TWCR = TWCR_STOP;                 //conclude transaction
    return(0);                        //return success value
}
```