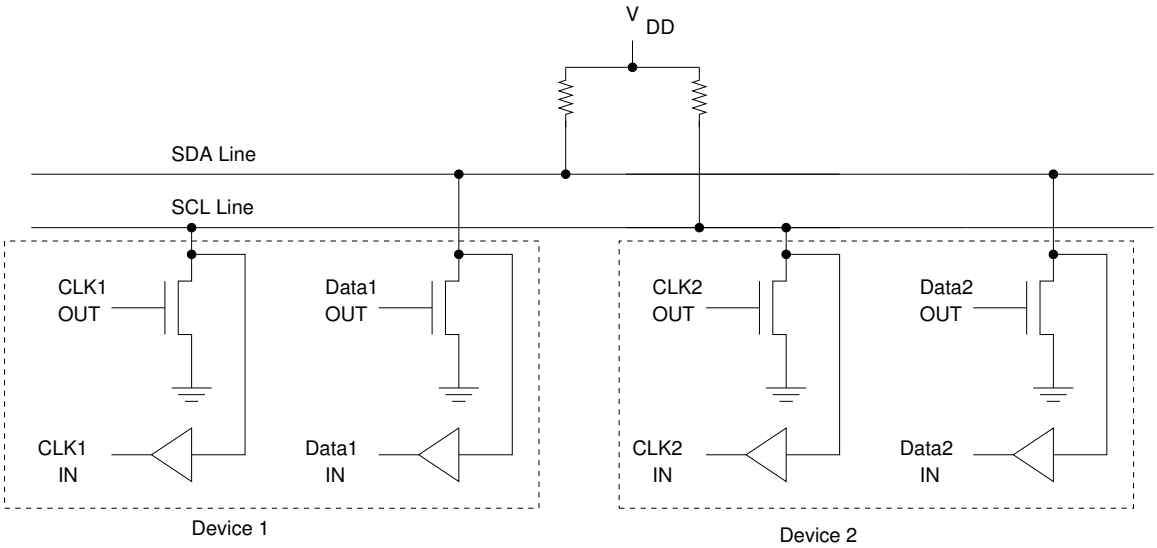
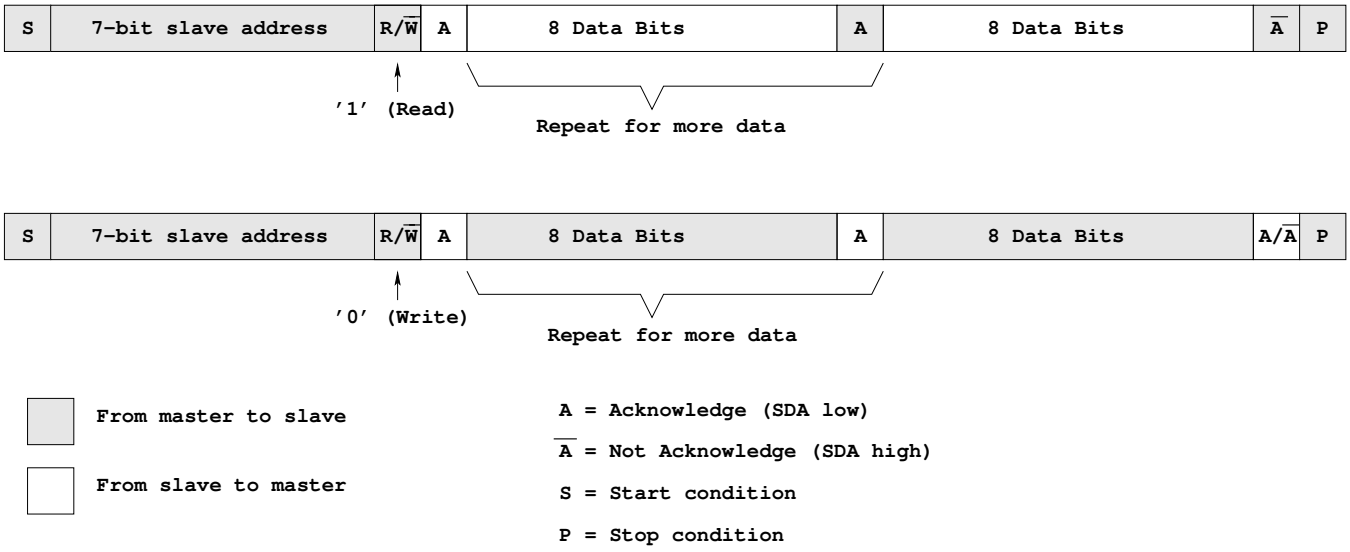
The MC9S12 IIC Interface

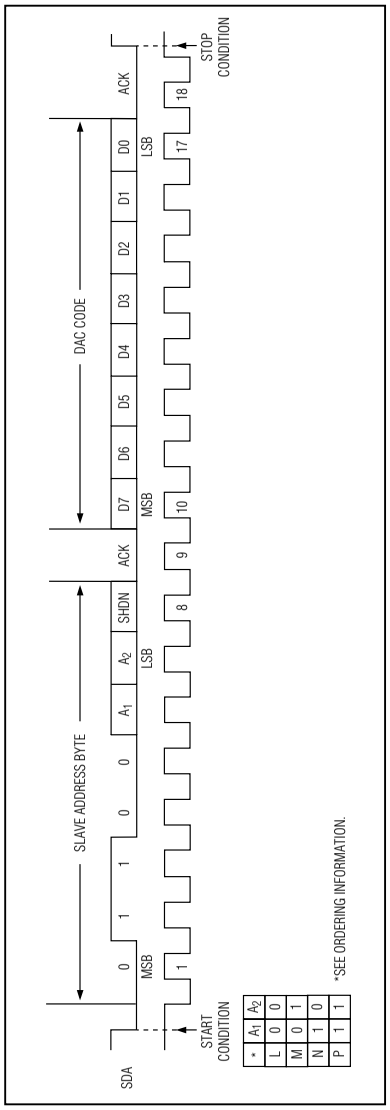
- The IIC bus can control multiple devices using only two wires
  - The two wires are Clock and Data
  - The devices connect to the wires using a *wired AND* method
  - The lines are normally high. Any device on the bus can bring them low.
- Each device on the bus has a unique address
- An IIC master starts the process by sending out a serial stream with the seven-bit address of the slave it wants to talk to, and an eighth bit indicating if it wants to write to the slave or read from the slave
- If it writes to the slave, it will continue send data on the serial data line until all the data is sent.
- If it reads from the slave, it will release the data line, and activate the clock line. The slave takes over the data line, and sends out its data in response to the clock provided by the master.
- After all the data is transfered, the master releases both the clock and the data lines



The IIC Interface

- Normally, both SDA and SCL are high.
- When transmitting data, SDA changes only when SCL is low.
- When the master wants to talk to a slave, it brings SDA low while SCL is high (the Start condition). This gives the master control of the bus
- The master sends eight bits of data on the SDA bus, and toggles the SCL bus for each bit. The eight bits are the seven-bit address of the slave, and one bit for read-write – a low indicates a write, and a high indicates a read. After the eighth data bit, the master releases the SDA line.
  - There is a ten-bit address mode discussed in the text, which we will not discuss
- The master releases the SDA bus and sends a ninth clock pulse. The adressed slave responds with an acknowledge by bringing SDA low.
  - If no slave responds, master knows not to continue
- It the master is reading data from the slave, it controls the SCL line and the slave controls the SDA line. The master sends eight clock pulses on the SCL line, and the slave transfers one bit of data on each clock pulse. At the end of the eight bits, the master takes over the SDA bus and sends an acknowledge (ACK) on all but the last byte it wants. After the last byte the master wants, the master sends an NACK (leaves SDA high).
- If the master is writing data to the slave, it continues controlling both SDA and SCL. It sends eight bits of data, releases SDA, and waits for an acknowledge from the slave
- After the transfer is complete (for both read and write), the master sends a Stop condition to indicate that it is releasing the bus. The Stop condition is indicated by bringing SDA high while SCL is high.




SDA
START CONDITION
SLAVE ADDRESS BYTE
MSB
LSB
0
1
1
0
0
A1
A2
SHDN
ACK
DAC CODE
D7
D6
D5
D4
D3
D2
D1
D0
ACK
STOP CONDITION
1
8
9
10
17
18
*SEE ORDERING INFORMATION.
* A1 A2
L 0 0
M 0 1
N 1 0
P 1 1

Using the IIC Interface

- The IIC uses five registers
  1. `IBAD` (IIC Bus Address Register)
     This is the address of the IIC when it is addressed as a slave. We will use the IIC in master mode only. Write something like 0x01 to this register (any address will work, as long as it is not the address of a slave on the bus).
  2. `IBFD` (IIC Bus Frequency Divide Register)
     This register determines the speed of the transfers. It also determines the hold times for start and stop conditions. Table 3-4 of the data sheet shows what the divide times and hold times are for all possible values of `IBFD`. The SCL Divider is the most important. If the slave device has a maximum frequency of 100 kHz, then 24 MHz/100 KHz = 240, so the SCL Divider must be greater than or equal to 240. Looking at the table, a value of 0x23 will give a divider of 256, so that is the one to use.
  3. `IBCR` (IIC Bus Control Register)
     This register controls the IIC.
     - `IBEN`: Enable the IIC Bus
     - `IBIE`: Enable interrupts
     - `MS`/$\overline{\texttt{SL}}$: Switch into master mode
     - `Tx`/$\overline{\texttt{Rx}}$: Switch between transmit and recieve
     - `TKAK`: Send an acknowledge
     - `RSTA`: Send a restart
     - `IBSWAI`: Specify whether or not the IIC clock should operate in WAIT mode
  4. `IBSR` (IIC Bus Status Register)
     This register indicates the status of the IIC, and is used to clear interrupts bits.
     - `TCF`: Transmit complete flag. Interrupt generated when `TCF` goes from low to high when `IBEN` is set.
     - `IAAS`: Addressed as slave
     - `IBB`: IIC Bus Busy
     - `IBAL`: Arbitration lost
     - `SRW`: Slave read/write
     - `IBIF`: Interrupt flag. Clear by writing a 1 to this bit
     - `RXAK`: Received Acknowledge
  5. `IBDR` (IIC Bus Data Register)
     - For both write to and read from the slave, first write to `IBDR` must be slave address plus R/$\overline{\text{W}}$ bit.
     - Write data to this register to send to slave
     - Read data from this register to receive from slave

Using the IIC Bus in Master Mode

To use the IIC bus in master mode, do the following:

- Initialize the IIC Bus. Write to the `IBFD` register to set the clock speed, write to the `IBAD` register to set a slave address, and write to the `IBCR` register to enable the IIC bus:

```
IBFD = 0x23;            // Set to 100 KHz operation
IBAD = 0x01;            // Slave address 1
IBCR = IBCR | BIT7;     // Set IBEN
```

- To write data to a slave:
  - Make sure the bus is not busy
  - Write to `IBCR` to enable XMIT and MASTER. The MC9S12 automatically emits a start condition when you do this.
  - Send data word which contains the device address and indicates a write
  - Wait for the transfer to complete
  - Make sure ACK received from slave. (If not, do something reasonable)
  - Send data, wait for ACK, repeat until all data sent
  - After all data send, send a stop condition by taking MC9S12 out of master mode.

```
/* Write data to device with address 0x68 */
while ((IBSR & BIT5) == BIT5) ; // Wait for IBB flag to clear
IBCR = IBCR | BIT4 | BIT5;      // Set XMIT and MASTER mode, emit start
IBDR = 0xD0;                    // Send device address, R/W bit = 0 => write
while ((IBSR & BIT5) == 0) ;    // Wait for IBB flag to set
while ((IBSR & BIT1) == 0) ;    // Wait for IBIF to set, indicating transfer complet
IBSR = BIT1;                    // Clear IBIF
if ((IBSR & BIT0) == BIT0) iic_stop();   // Do something if no slave answers
...                             // Repeat for more data
IBCR = IBCR & ~BIT5;            // Done; generate stop signal
```

Program to write data to an IIC slave

```
#include "hcs12.h"

void iic_init(void);
void iic_start(char address);
void iic_response(void);
void iic_stop(void);
void iic_transmit(char data);

main()
{
    /* Write to RTC */
    iic_init();
    iic_start(0xd0);
    while ((IBCR & BIT5) == 0)    // Make sure we still own bus
    iic_transmit(0);
    while ((IBCR & BIT5) == 0) ;  // Make sure we still own bus
    ...                           // Transmit more data
    iic_stop();
    asm(" swi");
}

void iic_init(void)
{
    IBFD = 0x23;            // Set to 100 KHz operation
    IBAD = 0x02;            // Slave address 1
    IBCR = IBCR | BIT7;     // Set IBEN to enable IIC bus
}

void iic_start(char address)
{
    while ((IBSR & BIT5) == BIT5) ; // Wait for IBB flag to clear
    IBCR = IBCR | BIT4 | BIT5;      // Set XMIT and MASTER mode, emit start
    IBDR = address;                 // Send device address and R/W bit
    while ((IBSR & BIT5) == 0) ;    // Wait for IBB flag to set
    iic_response();                 // Wait for response
}

void iic_response(void)
{
    while ((IBSR & BIT1) == 0) ;             // Wait for IBIF to set
    IBSR = BIT1;                             // Clear IBIF
    if ((IBSR & BIT0) == BIT0) iic_stop();   // Stop if NAK
//    DB12FNP->printf("In iic_response:  ACK received\n\r");
}
```

```
/* Release bus and stop (only if transmitting */
void iic_stop(void)
{
    IBCR = IBCR & ~BIT5;
}

void iic_transmit(char data)
{
    IBDR = data;        // Write data to device
    iic_response();     // Wait for response
}
```

Dallas Semiconductor DS1307 Real Time Clock

- The DS 1307 is a real-time clock with 56 bytes of NV (non-volatile) RAM
- It uses the IIC bus, with address 1101000_2
- It stores date and time
  - Data are stored in BCD format
- It uses a 32.768 kHz crystal to keep time
- It can generate a square wave output
  - Frequency of square wave can be 1 Hz, 4.096 kHz, 8.192 kHz or 32.768 kHz
- It uses a battery to hold the date and time when your board is not powered

Using the Dallas Semiconductor DS1307 Real Time Clock

- Find the SCL frequency
- Determine the value to write to `IBFD` to get that frequency or slower
- Make sure start condition and stop condition times met
- To set the clock, write address to clock (with R/$\overline{\text{W}}$ low), then write a 0 (to select seconds register), then send second, minute, hour, day of week, day of month, month, year, control
  - Control determines whether or not to enable square wave, and selects frequency
- To read the clock, write the address to the clock (with R/$\overline{\text{W}}$ low), then write a 0 (to select seconds register). After that, write the address to the clock (with R/$\overline{\text{W}}$ high), then read the time information.
- If you want to store some data which will remain between power cycles, you can write it to the 56 bytes of NV RAM