Inter-Integrated Circuit (I2C)

Karthik Hemmanur
ECE 480- Design Team 3
Fall 2009

Keywords:
I2C, Serial Communication, Protocol, SDA, SCL

Abstract:

Inter-Integrated Circuit, abbreviated as I2C is a serial bus short distance protocol developed by Philips Semiconductor about two decades ago to enhance communication between the core on the board and various other ICs involved around the core. This application note intends to describe the functionality of various serial buses with emphasis on I2C, and how I2C is different from other serial buses. This application note also intends to explain the functionality and working of I2C, as well as some sample code that explains how I2C is implemented.

Introduction:

The general concept of serial bus communication is shown in Figure 1. The most popular serial bus communication protocols available today in the market are, SPI, UART, I2C, CAN, USB, IEE1394, and so on.

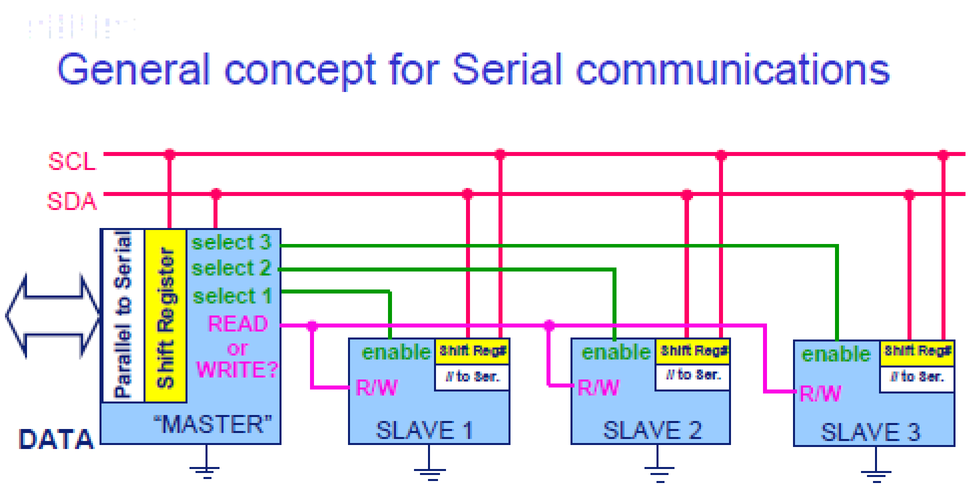

Figure 1: Serial Communication Overview

Philips originally developed I2C for communication between devices inside of a TV set. Examples of simple I2C-compatible devices found in embedded systems include EEPROMs, thermal sensors, and real-time clocks. I2C is also used as a control interface to signal processing devices that have separate, application-specific data interfaces. Philips, National Semiconductor, Xicor, Siemens, and other manufacturers offer hundreds of I^2C-compatible devices. I2C buses can typically reach speeds up to 400 Kbps.

Structure of I2C:

I^2C is appropriate for interfacing to devices on a single board, and can be stretched across multiple boards inside a closed system. An example is a host CPU on a main embedded board using I2C to communicate with user interface devices located on a separate front panel board. I2C is a two-wire serial bus, as shown in Figure 1. There's no need for chip select or arbitration logic, making it cheap and simple to implement in hardware. The two I2C signals are serial data and serial clock. Together, these signals make it possible to support serial transmission of 8-bit bytes of data-7-bit device addresses plus control bits-over the two-wire serial bus.

In a bind, an I2C slave can hold off the master in the middle of a transaction using what's called clock stretching (the slave keeps SCL pulled low until it's ready to continue). Most The I2C protocol can also support multiple masters. There may be one or more slaves on the bus. Both masters and slaves can receive and transmit data bytes.

Each I2C-compatible hardware slave device comes with a predefined device address, the lower bits of which may be configurable at the board level. The master transmits the device address of the intended slave at the beginning of every transaction. Each slave is responsible for monitoring the bus and responding only to its own address. This addressing scheme limits the number of identical slave devices that can exist on an I2C bus without contention, with the limit set by the number of user-configurable address bits.

Communication in I2C:

Figure 2 shows the communication in I2C.

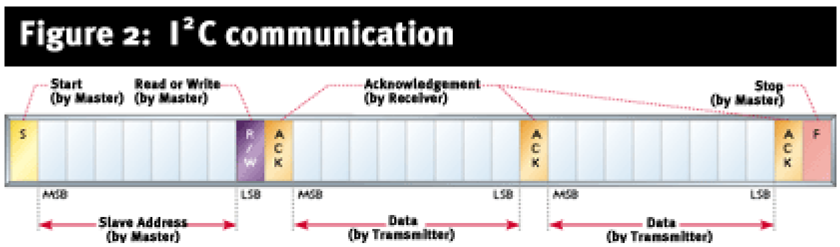

Figure 2: Communication in I2C

The I2C signaling protocol provides device addressing, a read/write flag, and a simple acknowledgement mechanism. Other elements of I2C protocol are general call (broadcast) and 10-bit extended addressing. Standard I2C devices operate up to 100Kbps, while fast-mode devices operate at up to 400Kbps. Most often, the I^2C master is the CPU or microcontroller in the system. Some microcontrollers even feature hardware to implement the I2C protocol. You can also build an all-software implementation using a pair of general-purpose I/O pins. Since the I2C master controls transaction timing, the bus protocol doesn't impose any real-time constraints on the CPU beyond those of the application. For a fixed I2C, the high and low logics are defined at 3.0 V and 1.5 V. For dependant I2C, these are defined at 0.7*Vdd and 0.3*Vdd respectively. The pull-up resistor values required for I2C are typically at 1K for 3.0V of Vdd and 1.6K for 5V of Vdd. Typical operating temperatures are between -40 degrees and +85 degrees Centigrade.

Addressing in I2C:

Figure 3 shows the SDA and SCL for I2C

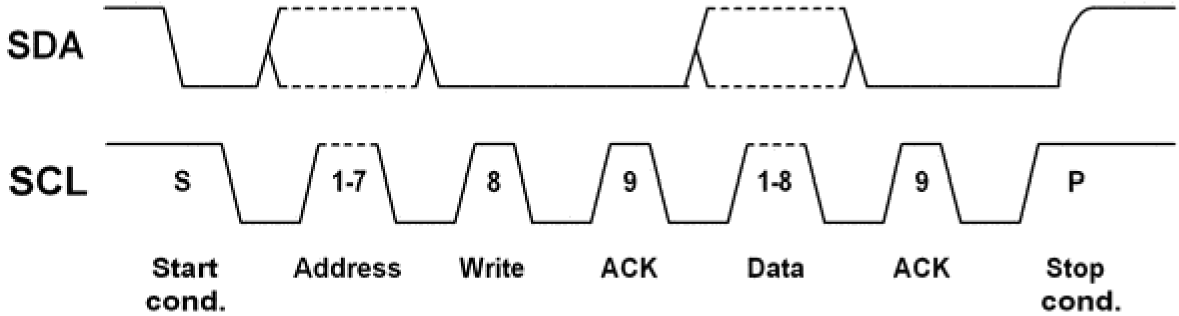

Figure 3: I2C addressing

The following table shows I2C addresses reserved for special purposes:

10 bit addresses, binary noted, MSB is left	Purpose
0000000 0	General Call
0000000 1	Start Byte
0000001 X	CBUS Addresses
0000010 X	Reserved for Different Bus Formats
0000011 X	Reserved for future purposes
00001XX X	High-Speed Master Code
11110XX X	10-bit Slave Addressing
11111XX X	Reserved for future purposes

(See Source 5 in References)

I2C Sample Code:

The following is the sample code for I2C ADC for Philips PCF8591

```
//**************************************************
// pcf8591d.c
//
// Example to read A/D values from a
// 4 channel / 8 bit AD converter PCF8591
// through I2C using the I2C driver improved by
// Geert Vancompernolle
// http://www.acmesystems.it/?id=10
//**************************************************

#include "stdio.h"
#include "stdlib.h"
#include "unistd.h"
#include "sys/ioctl.h"
#include "fcntl.h"
#include "time.h"
#include "string.h"

#include "i2c_errno.h"
#include "etraxi2c.h"


int main( int argc, char **argv ) {
  int rtc;
  int fd_i2c;
  I2C_DATA i2c_d;
  int ch;

  printf("Reading from a PCF8591 (4 chanel A/D at 8 bits with I2C
bus)\n");

  fd_i2c = open( "/dev/i2c", O_RDWR );
  if (fd_i2c<=0)     {
    printf( "Open error on /dev/i2c\n" );
    exit( 1 );
  }

  // PCF8591 address scheme
  // |  1 |  0 |  0 |  1 | A2 | A1 | A0 | R/W |
  i2c_d.slave =(0x09<<4)|(0x01<<1);

  for (ch=0;ch<=3;ch++) {
    // Select the A/D channel
    i2c_d.wbuf[0] = ch;
    i2c_d.wlen  = 1;
    if ((rtc=ioctl(fd_i2c,_IO( ETRAXI2C_IOCTYPE, I2C_WRITE),
&i2c_d))!=EI2CNOERRORS)  {
      close(fd_i2c);
      printf( "Error %d on line %d\n",rtc,__LINE__);
```

```
      return ( -1 );
    }

    i2c_d.rlen  = 3;
    if ((rtc=ioctl(fd_i2c,_IO( ETRAXI2C_IOCTYPE, I2C_READ),
&i2c_d))!=EI2CNOERRORS)  {
      close(fd_i2c);
      printf( "Error %d on line %d\n",rtc,__LINE__);
      return ( -1 );
    }

    // Show the voltage level
    printf("Chanel %d = %.2fv (%02X
hex)\n",ch,i2c_d.rbuf[2]*0.012941,i2c_d.rbuf[2]);
  }

  close(fd_i2c);
  return(0);
}
```

References:

Using the I2C bus: Sample code
< http://foxlx.acmesystems.it/?id=10>

I2C Introduction
< http://www.embedded.com/story/OEG20010718S0073>

NXP Document on I2C
< http://www.nxp.com/documents/application_note/AN10216.pdf>

NXP Standartics
< http://www.standardics.nxp.com/support/documents/i2c/pdf/an10146.pdf>

I2C Bus Website
< http://www.i2c-bus.org/addressing/>