Serial communications with I²C

DRAFT VERSION - This is part of a course slide set, currently under development at:

http://mbed.org/cookbook/Course-Notes

We welcome your feedback in the comments section of the course notes cookbook page. Tell us if these slides are useful, if you would use them to help run lessons, workshops or training, and how you'd improve them.

Written by R. Toulson (Anglia Ruskin University) and
T. Wilmshurst (University of Derby), (c) ARM 2011

An accompanying textbook is also being considered if there is interest

Serial communications with I^2C

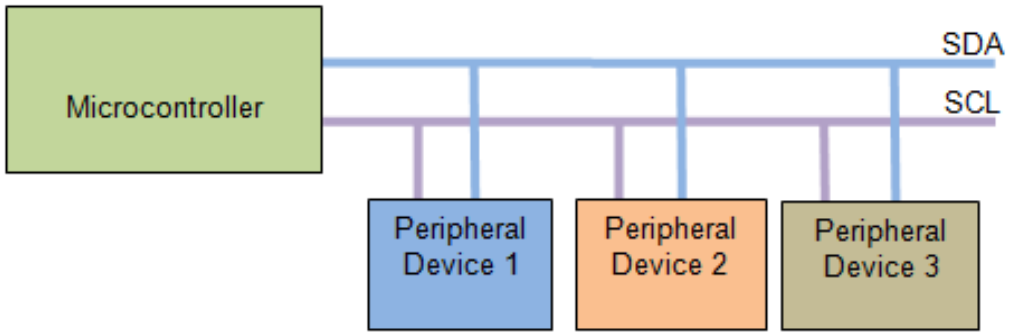
- Introducing I^2C
- Evaluating simple I^2C communications
- I^2C on the mbed
- Working with the TMP102 I2C temperature sensor
- Working with the SRF08 ultrasonic rangefinder
- Interfacing multiple devices on a single I2C bus
- Extended exercises

Introducing I^2C

- The name I^2C is shorthand for Standard Inter-Integrated Circuit bus
- I^2C is a serial data protocol which operates with a master/slave relationship
- I^2C only uses two physical wires, this means that data only travels in one direction at a time.

Introducing I²C

- The I²C protocol is a two-wire serial bus:



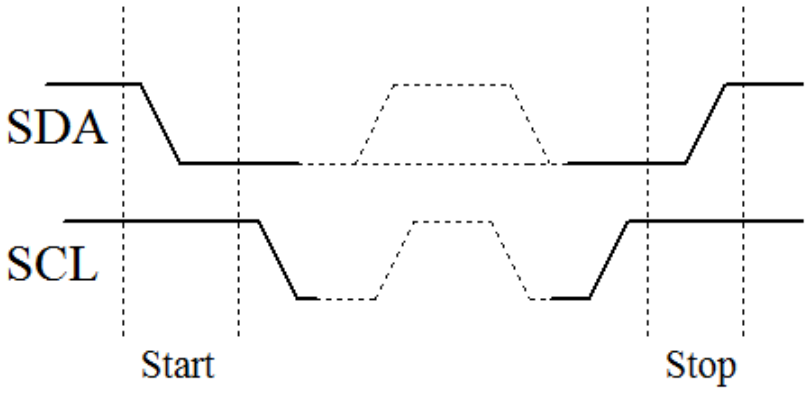
- The I²C communication signals are serial data (SDA) and serial clock (SCL)
  - These two signals make it possible to support serial communication of 8-bit data bytes, 7-bit device addresses as well as control bits
  - Using just two wires makes it cheap and simple to implement in hardware

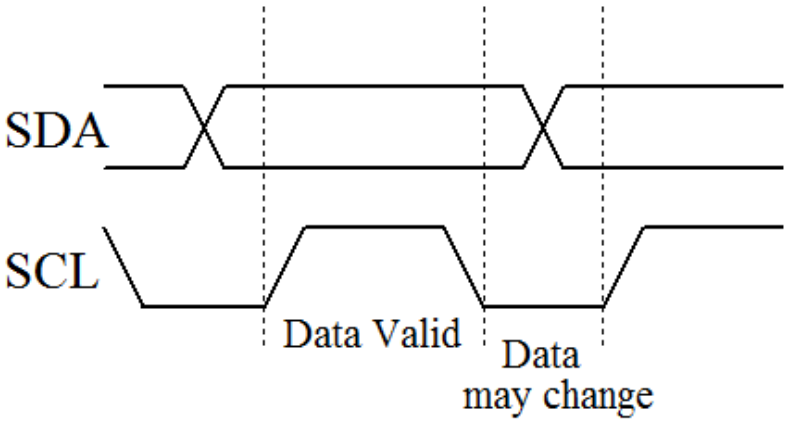
Evaluating simple I^2C communications

- I^2C has a built-in addressing scheme, which simplifies the task of linking multiple devices together.
  - In general, the device that initiates communication is termed the 'master'. A device being addressed by the master is called a 'slave'.
  - Each I^2C-compatible slave device has a predefined device address. The slaves are therefore responsible for monitoring the bus and responding only to data and commands associate with their own address.
  - This addressing method, however, limits the number of identical slave devices that can exist on a single I^2C bus, as each device must have a unique address. For some devices, only a few of the address bits are user configurable.

Evaluating simple I²C communications

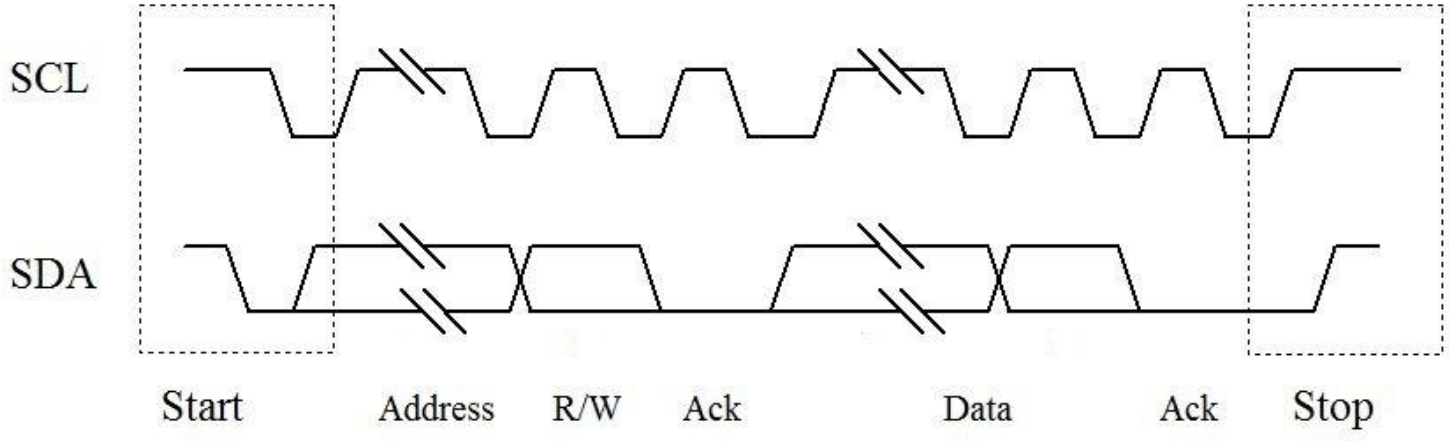
- A data transfer is made up of the Master signalling a Start Condition, followed by one or two bytes containing address and control information.
- The Start condition is defined by a high to low transition of SDA when SCL is high.
- A low to high transition of SDA while SCL is high defines a Stop condition
- One SCL clock pulse is generated for each SDA data bit, and data may only change when the clock is low.





Evaluating simple I²C communications

- The byte following the Start condition is made up of seven address bits, and one data direction bit (Read/Write)
- All data transferred is in units of one byte, with no limit on the number of bytes transferred in one message.
- Each byte must be followed by a 1-bit acknowledge from the receiver, during which time the transmitter relinquishes SDA control.



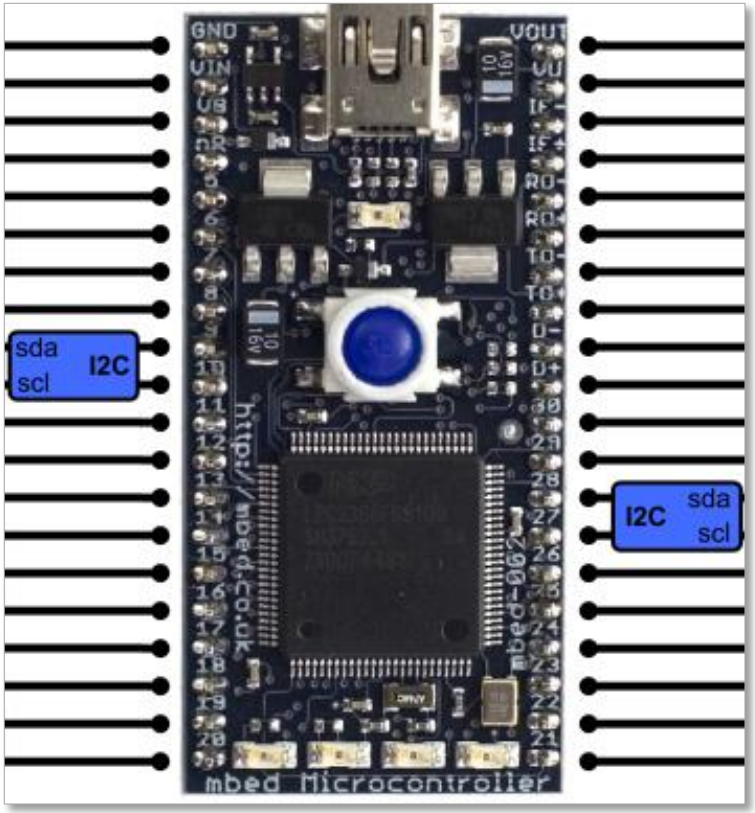
I²C on the mbed

The mbed I²C library functions are shown in the table below:

I2C	An I2C Master, used for communicating with I2C slave devices
Functions	**Usage**
I2C	Create an I²C Master interface, connected to the specified pins
frequency	Set the frequency of the I²C interface
read	Read from an I²C slave
read	Read a single byte from the I²C bus
write	Write to an I²C slave
write	Write single byte out on the I²CC bus
start	Creates a start condition on the I²C bus
stop	Creates a stop condition on the I²C bus

I²C on the mbed

- The I2C Interface can be used on mbed pins p9/p10 and p28/p27
- Note also that the SDA and SCL data signals each need to be ‘pulled up’ to 3.3V through a 2.2 kΩ resistor



Evaluating the TMP102 I²C temperature sensor

- Configuration and data register details are given in the TMP102 data sheet

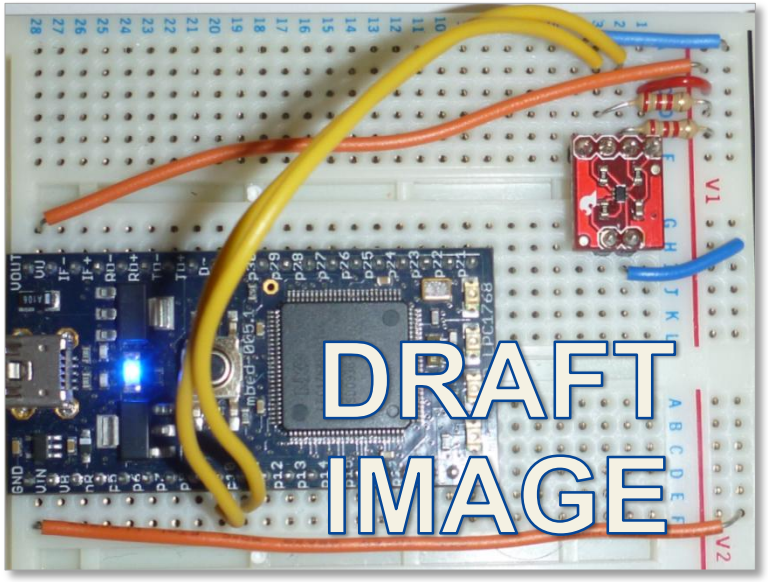
  http://focus.ti.com/lit/ds/sbos397b/sbos397b.pdf

- To configure the temperature sensor we need to:
  - Use arrays of 8-bit values for the data variables, because the I²C bus can only communicate data in one bytes.
  - Set the configuration register; we first need to send a data byte of 0x01 to specify that the Pointer Register is set to 'Configuration Register'.
  - Send two bytes of data to perform the configuration. A simple configuration value to initialise the TMP102 to normal mode operation is 0x60A0.
- To read the data register we need to:
  - To read the data register we need to set the pointer value to 0x00.
  - To print the data we need to convert the data from a 16-bit data reading to an actual temperature value. The conversion required is to shift the data right by 4 bits (its actually only 12-bit data held in two 8-bit registers) and to multiply by the 1-bit resolution which is 0.0625 degrees C per LSB.

Interfacing the TMP102 with the mbed

- The TMP102 can be connected to the mbed as shown:

Signal	TMP102 Pin	Mbed Pin	Notes
Vcc (3.3V)	1	40	
Gnd (0V)	4	1	
SDA	2	9	2.2kΩ pull-up to 3.3V
SCL	3	10	2.2kΩ pull-up to 3.3V



Working with the TMP102 I2C temperature sensor

- Exercise 1: Connect the temperature sensor to an I^2C bus. Verify that the correct data can be read by continuously display updates of temperature to the screen.
  - Test that the temperature increases when you press your finger against the sensor. You can even try placing the sensor on something warm, for example a pocket hand warmer, in order to check that it reads temperature correctly.

Working with the TMP102 I2C temperature sensor

The following program will configure the TMP102 sensor, read data, convert data to degrees Celsius and then display values to the screen every second:

```
#include "mbed.h"
I2C tempsensor(p9, p10); //sda, sc1
Serial pc(USBTX, USBRX); //tx, rx
const int addr = 0x90;
char config_t[3];
char temp_read[2];
float temp;
int main() {
    config_t[0] = 0x01;                    //set pointer reg to 'config register'
    config_t[1] = 0x60;                    // config data byte1
    config_t[2] = 0xA0;                    // config data byte2
    tempsensor.write(addr, config_t, 3);
    config_t[0] = 0x00;                    //set pointer reg to 'data register'
    tempsensor.write(addr, config_t, 1);   //send to pointer 'read temp'
    while(1) {
        wait(1);
        tempsensor.read(addr, temp_read, 2);       //read the two-byte temp data
        temp = 0.0625 * (((temp_read[0] << 8) + temp_read[1]) >> 4);  //convert data
        pc.printf("Temp = %.2f degC\n\r", temp);
    }
}
```

Evaluating the SRF08 ultrasonic rangefinder

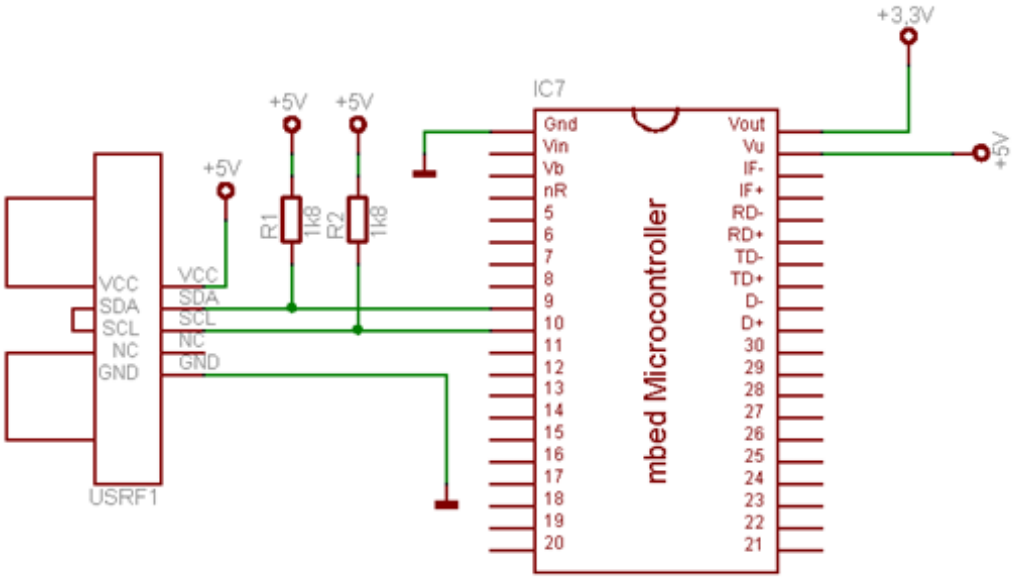
- Configuration and data register details are given in the SRF08 data sheet:

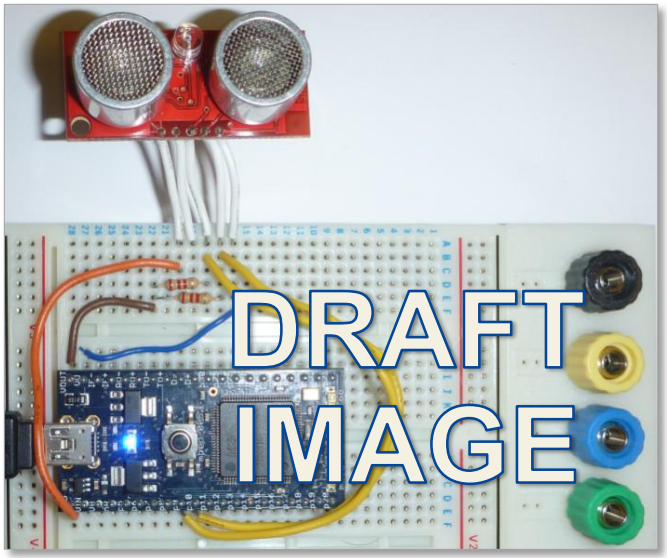
  http://www.rapidonline.com/netalogue/specs/78-1086.pdf

- The following information summarises the configuration and data read procedures for the SRF08:
  - The rangefinder I^2C address is 0xE0.
  - The pointer value for the command register is 0x00.
  - A data value of 0x51 to the command register initialises the range finder to operate and return data in cm.
  - A pointer value of 0x02 prepares for 16-bit data (i.e. two bytes) to be read.

Interfacing the SRF08 with the mbed

- The SRF08 ultrasonic range finder is an I^2C device which measures distance and proximity of items
- The SRF08 can be connected to the mbed as shown:





Working with the SRF08 ultrasonic rangefinder

- Exercise 2: Configure the SRF08 ultrasonic rangefinder to update distance data to the screen. The following code will perform this operation:

```
#include "mbed.h"
I2C rangefinder(p9, p10); //sda, sc1
Serial pc(USBTX, USBRX); //tx, rx
const int addr = 0xE0;
char config_r[2];
char range_read[2];
float range;
int main() {
    while (1) {
        config_r[0] = 0x00;                         //set pointer reg to 'cmd register'
        config_r[1] = 0x51;                         // config data byte1
        rangefinder.write(addr, config_r, 2);
        wait(0.07);
        config_r[0] = 0x02;                         //set pointer reg to 'data register'
        rangefinder.write(addr, config_r, 1);       //send to pointer 'read range'
        rangefinder.read(addr, range_read, 2);      //read the two-byte range data
        range = ((range_read[0] << 8) + range_read[1]);
        pc.printf("Range = %.2f cm\n\r", range);  //print range on screen
        wait(0.05);
    }
}
```

Working with the SRF08 ultrasonic rangefinder

- Exercise 3:

  Evaluate the SRF08 datasheet to understand the different control setup and data conversion options.

  Now configure the SRF08 to accurately return data in inches measured to a maximum range of 10 feet.

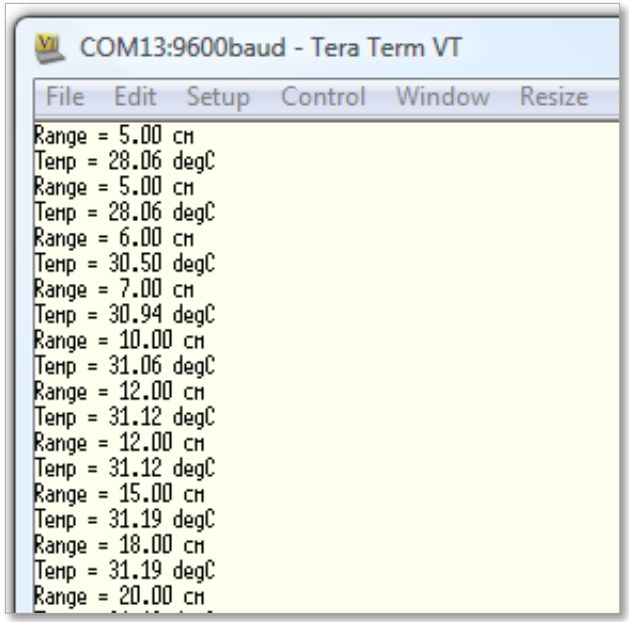
  Note:

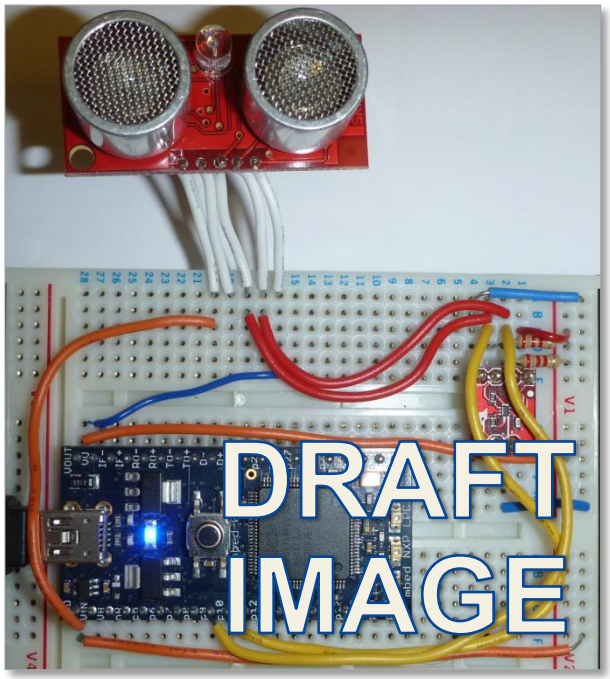
  1 inch = 25.4 mm

  1 foot = 12 inches

Interfacing multiple devices on an I²C bus

- Exercise 4: Connect both the temperature sensor and the range finder to **the same** I²C bus and verify that the correct data can be read by addressing commands appropriately. Update your screen readout to continuously display updates of temperature and detected range.





Extended Exercises

- Exercise 5: Use an ultrasonic range finder to control the position of a servo motor. When items are distant, the servo will take a 0 degrees position, but as objects come within range of the SRF08, the servo will move towards a 180 degrees position.

- Exercise 6: Communicate between two mbeds on an I2C bus. Program the master mbed to count a value from 0 to 15 and to broadcast this count value on the I^2C bus. The slave mbed should be programmed to read the I^2C data at regular intervals and set the onboard LEDs to display the binary count value.

Summary

- Introducing I^2C
- Evaluating simple I^2C communications
- I^2C on the mbed
- Working with the TMP102 I2C temperature sensor
- Working with the SRF08 ultrasonic rangefinder
- Interfacing multiple devices on a single I2C bus
- Extended exercises