# **Embedded Systems Essentials with Arm: Get Practical with Hardware**

## Module 2

## KV3: Using the Mbed API with the I2C and USB protocols

All Mbed-enabled devices can use I2C if they have an I2C port and there are API functions to help with that.

We can create an I2C master just using I2C. We can set the frequency, and we can do some reading and writing. We can write just one byte or the whole package, including the address, the data, the stop and the start condition. As always check the arm website for full details on any upgrades there might be.

This is an example showing the main I2C features of this program.

|  |
| --- |
| #include "mbed.h"​  I2C i2c\_port(p9, p10); //Configure a serial port, pins 9 and 10 are sda, scl​  ​  char switch\_word ; //word we will send​  char recd\_val; //value received from slave​  const int addr = 0x52; //the I2C slave address, an arbitrary even number​  ​  int main() {​  while(1) {​  switch\_word=0xaa; //set up a recognisable output pattern​  //send a single byte of data, in correct I2C package​  i2c\_port.start(); //force a start condition​  i2c\_port.write(addr); //send the address​  i2c\_port.write(switch\_word); //send one byte of data, ie switch\_word​  i2c\_port.stop(); //force a stop condition​  ... |

We can configure a microcontroller as a master or as a slave. Here we have it as a master. We use I2C nomenclature and we name it I two C underscore port. We then set up the pins we’ll use.

In this example, we’re sending a single byte of data, preceded by the address byte. We package this with start and stop and two write functions.

It’s possible to configure a microcontroller as a slave. We might use other devices as slaves, but here we have a set of functions that allow us to create that slave. It’s not necessary to set up the frequency, as the master is in charge of that. We can then set up the slave to receive, read, write and perform other I2C related activities.

Remember that a slave can’t initiate a transmission, it can only be ready to receive and wait for the master’s instruction. So here we are setting up the conditions for the slave to receive and acknowledge that it has received the data.

|  |
| --- |
| #include <mbed.h>​  I2CSlave slave(p9, p10); //Configure I2C slave​  char switch\_word ; //word we will send​  char recd\_val; //value received from master​  ​  int main() {​  slave.address(0x52);​  while (1) {​  ​  switch\_word=0xa0; //set up a recognisable output pattern​  slave.write(switch\_word); //load up word in readiness to send​  //test for I2C, and act accordingly​  int i = slave.receive();​  if (i == 3){ //slave is addressed, master will write​  recd\_val=slave.read();​  ... |

Arm has provided a set of APIs for USB in the form of a USBDevice library. A few of these are represented in this table.

|  |  |
| --- | --- |
| **Function Name​** | **Description ​** |
| USBMouse​ | Allows the mbed to emulate a USB mouse​ |
| USBKeyboard​ | Allows the mbed to emulate a USB keyboard​ |
| USBMouseKeyboard​ | A USB mouse and keyboard feature set combined in a single library​ |
| USBHID​ | Allows custom data to be sent and received from a Human Interface Device (HID) allowing custom USB features to be developed without the need for host drivers to be installed​ |
| USBSerial​ | Emulates an additional standard serial port on the mbed, through the USB connections​ |
| USBMIDI​ | Allows send and receive of MIDI messages in communication with a host PC using MIDI sequencer software​ |
| USBAudio​ | Allows the mbed to be recognised as an audio interface allowing streaming audio to be read, output or analysed and processed.​ |
| USBMSD​ | Emulates a mass storage device over USB, allowing interaction with a USB storage device.​ |

Most of them allow the Mbed to emulate a number of external devices through USB.

|  |
| --- |
| ​  #include "mbed.h" // include mbed library​  #include "USBMouse.h" // include USB Mouse library​  USBMouse mouse; // define USBMouse interface​  ​  int dx[]={40,0,-40,0}; // relative x position co-ordinates​  int dy[]={0,40,0,-40}; // relative y position co-ordinates​  ​  int main() { ​  while (1) {​  for (int i=0; i<4; i++) { // scroll through position co-ordinates​  mouse.move(dx[i],dy[i]); // move mouse to co-ordinate ​  wait(0.2);​  }​  }​  }​ |

This example shows how we can emulate a USB mouse with the Mbed.

With USBMouse it is possible to make the Mbed behave like a standard USB mouse, sending position and button press commands to the host. We can see this in this example.

This program example implements a USB mouse interface and continuously sends relative position information to move the mouse pointer around four coordinates which make up a square. These are defined by the two arrays dx and dy. ​

To compile this program it is necessary to import the USBDevice library. The Mbed-enabled device running the program needs to have USB capability, and is connected via its USB port to the PC.

Note that although this is simple, this is a complete program.

**Visit arm.com for more information.**