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

# LAB 1

# Introductory Serial Communication

## Contents

## 1 Introduction

This lab aims to implement two types of serial link, applying the Nucleo F401RE board. The specific objectives are:

- to apply SPI to transfer serial data from the Nucleo board to a shift register (SR);
- to implement a Liquid Crystal Display (LCD), linking it to the parallel output of the SR;
- to apply a UART link to transfer data to the host PC;
- to use a PC-based terminal to receive data from the Nucleo board;
- to develop your skills in hardware and software development.

We will proceed incrementally, building up a circuit and accompanying program in stages, so that each stage can be tested and successfully commissioned before moving on.

## 2 Resources

In this lab you can use either the Mbed Studio, or the on-line compiler (or both). Both were introduced in Lab 0. When programming, remember the on-line and print references which were introduced in Lab 0; they are repeated as References 3-9.

The hardware elements needed for this lab are listed in Table 1. Items which are in grey were used in Lab 0. Those in black are new for this lab.

| Item | Qty. |
|---|---|
| STM32F401 Nucleo-64 Development Board | 1 |
| Bread Board | 1 |
| Jumper Wires (kit) | 1 |
| LED with internal current-limiting resistor | 3 |
| 74HC595N Shift Register | 1 |
| Newhaven LCD. NHD-0420H1Z-FSW-GBW-33V3 | 1 |
| 10kΩ potentiometer | 1 |

*Table 1: List of Required Parts*

**Formatted Table**

## 3 Reviewing the Nucleo F401RE Board

The Nucleo F401RE Board was introduced in Lab 0. Connection information is repeated for convenience in Figure 1.

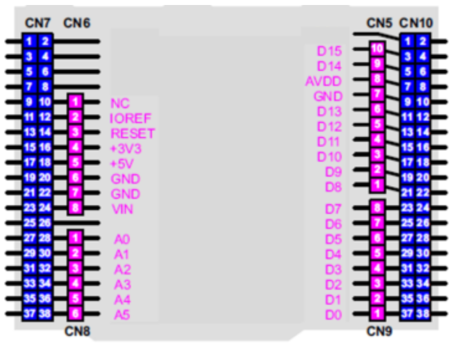


*Figure 1: Connections on Nucleo Board, Arduino Connectors Only*

## 4 Configuring a SPI Circuit

We will build our circuit in stages. Although we use one of the available SPI outputs of the Nucleo board as the Master, it is interesting to note that we use a general-purpose SR as the Slave. This emphasises the simplicity of the SPI protocol. The SR is type 74HC595N, whose pin connections are shown in Figure 2, and whose data sheet is given in Reference 1. If you are of an electronics background, you will find this data sheet interesting to look at, to see how the SR is implemented in this context.

### 4.1 Implementing the Shift Register

Viewing the connections shown in Figure 2, connect the SR to the Nucleo board. The notch in the shift register indicates the top or pin 1. An example build is shown in Figure 3. Notice how the Ground and 3.3 V connections are taken from the Nucleo board to the power distribution lines one side of the breadboard, and then linked across to the far side of the board. A green LED is linked across the Ground and 3.3 V lines, for power indication. Note that the LED specified in Table 1 has an internal current-limiting resistor; to put an unprotected LED across these two power lines would damage it irreversibly. As the data flow is from Nucleo to SR only, the MISO (Master In Slave Out) connection on the Nucleo is not used. Green and red test LEDs are shown connected to pins 2 and 3 of the SR, with their other terminal going to ground. You can add more LEDs to any of pins QA to QH if you wish, ensuring of course that all LEDs are connected the right way round (longer lead is positive).

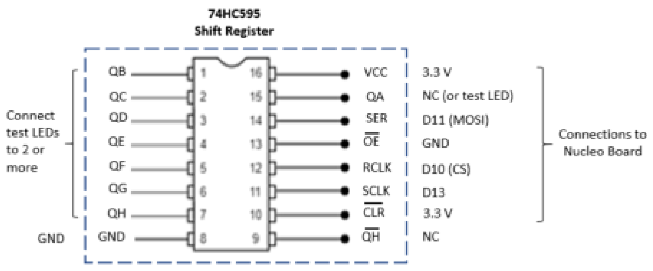


*Figure 2: Connections for the Shift Register*

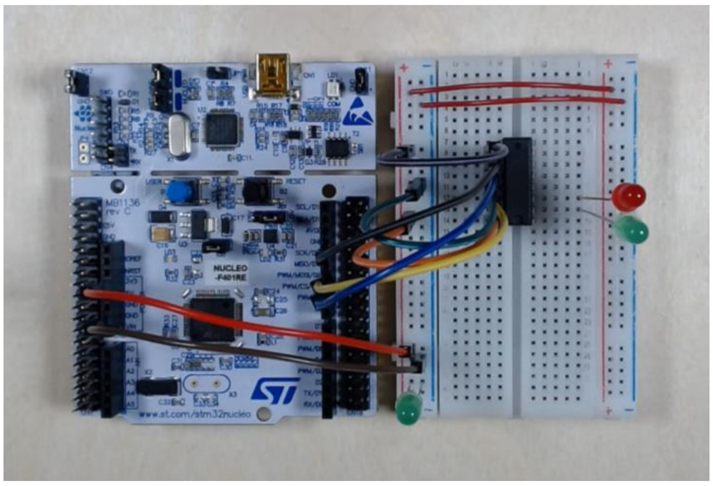


*Figure 3: Example Build for Nucleo Board Connected to Shift Register*

Program Example 1 is intended to test this simple circuit. It configures pins D11-13 as SPI port, using the Mbed API functions shown in Table 2. The default settings of the SPI interface are 1MHz, 8-bit, Mode 0; the configuration code lines in this program are thus not strictly needed, but are included for illustration. The program selects D10 to be used as chip select, hence configuring it as a digital output. The while loop then sends the words 0xAA (binary 10101010) and 0x55 (binary 01010101) continuously from the Nucleo SPI port to the SR; thus every bit changes state for every new word received. There is a half second pause between each transmission.

| Function name | Description |
|---|---|
| **SPI (PinName mosi, PinName miso, PinName sclk, PinName _unused=NC)** | Create a SPI master connected to the specified pins |
| **void format (int bits, int mode=0)** | Configure the data transmission format |
| **void frequency (int hz=1000000)** | Set the SPI bus clock frequency |
| **virtual int write (int value)** | Write to the SPI Slave and return the response |

*Table 2: Mbed API for SPI*

Using Mbed Studio or the on-line compiler, enter the code of Program Example 1, and compile and download it to your Nucleo Device.

```
/* Trial SPI Master: Sends two words in turn to external Shift Register.
   */

#include "mbed.h"
DigitalOut CS(D10);

SPI ser_port(D11, D12, D13); // mosi, miso, sclk
char switch_word;            //word we will send

int main() {
  ser_port.format(8,0);        // Set up the SPI for 8 bit data,
                                    //Mode 0 operation
  ser_port.frequency(1000000); // Clock frequency is 1MHz
  CS=1;                        //Chip Select idles high
  while (1){
    switch_word=0xAA;            //set up word to be transmitted
    CS=0;                        //Select Slave device
    ser_port.write(switch_word); //send switch_word
    CS=1;
    thread_sleep_for (500);
    switch_word=0x55;            //set up word to be transmitted
    CS=0;
    ser_port.write(switch_word); //send switch_word
    CS=1;
    thread_sleep_for (500);
    }
}
```

*Program Example 1: Testing the Shift Register Connection*

If your circuit is connected correctly, you should find your two test LEDs lighting in turn (as long as neighbouring SR outputs are chosen). If you have access to an oscilloscope, you may wish to reduce the wait times between each SPI transmission (say to 2 ms), and observe the resulting square wave output on the 'scope. Depending on how many traces you have, you could also observe the CS, SCLK and MOSI lines.

Don't break up this circuit, we are about to add the LCD to it!

## 4.2 Introducing the LCD

A popular form of LCD is the character display; the one we have chosen is seen in Figure 4, wired and ready to use. These are widely available from one line of characters to four or more, and are commonly seen on many domestic and office items, such as photocopiers, burglar alarms or DVD players. Driving this complex array of tiny LCD dots is far from simple, so such displays contain a hidden microcontroller, customised to drive the display. Our chosen LCD has 4 lines with 20 characters each. Its on-board microcontroller is the ST7066U, whose data can be found in Reference 2.

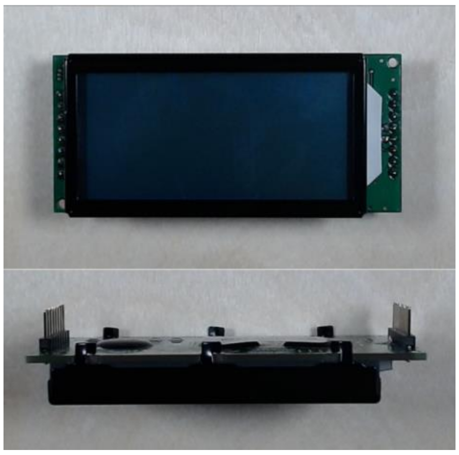


*Figure 4: The LCD, ready for Connection*

The ST7066U contains an 80-byte volatile memory array to hold the user display data, and a non-volatile memory for generating the characters. It has a simple instruction set, including instructions for initialisation, cursor control (moving, blanking, blinking), and clearing the display. Communication with the controller is made via an 8-bit data bus, 3 control lines – RS (Register Select), R/|W (Read/Write), Busy Flag - and an enable/strobe line (E). These are itemised in Table 3.

Data written to the controller is interpreted either as instruction or as display data, depending on the state of the RS line. An important use of reading data back from the LCD is to check the controller status via the Busy flag. As some instructions take finite time to implement (for example a minimum of 40 us is required for the LCD to receive one character code), it is sometimes useful to be able to read the Busy flag, and wait until the LCD controller is ready to receive further data.

The controller can be set up to operate in 8-bit or 4-bit mode. In the latter mode only the four most significant bits of the data bus are used, and two write cycles are required to send a single byte.

| **Line** | **Function** |
|---|---|
| RS | Register Select: 0 = Instruction register 1 = Data Register |
| R/\|W | Selects read (when 1) or write (when 0) |
| E | Synchronises read and write operations |
| DB4 - DB7 | Higher order bits of data bus; DB7 also used as Busy flag |
| DB0 - DB3 | Lower order bits of data bus; not used for 4-bit operation |

*Table 3: Interfacing with the ST7066U LCD Driver*

## 4.3 Linking the LCD display

We move now to wiring the LCD to the SR, applying the LCD in 4-bit mode. Note that the display could be connected direct to the Nucleo board, but inclusion of the SR allows a useful demonstration of using SPI, and limits the number of I/O connections required of the Nucleo.

Carefully connect the LCD to your previous SR circuit, following the diagram of Figure 5. We will be adding a potentiometer that will control the contrast of the LCD. If you don't see anything on the LCD screen when running the program, ensure to try twisting the potentiometer fully both ways to ensure the contrast is set correctly. If you turn the potentiometer to max it should display the text on the screen. Notice how the 4-bit data connection is made, and the use of the control lines. As we don't intend at this stage to read from the LCD, the R/|W line is permanently tied to ground. A completed build is seen in Figure 6.

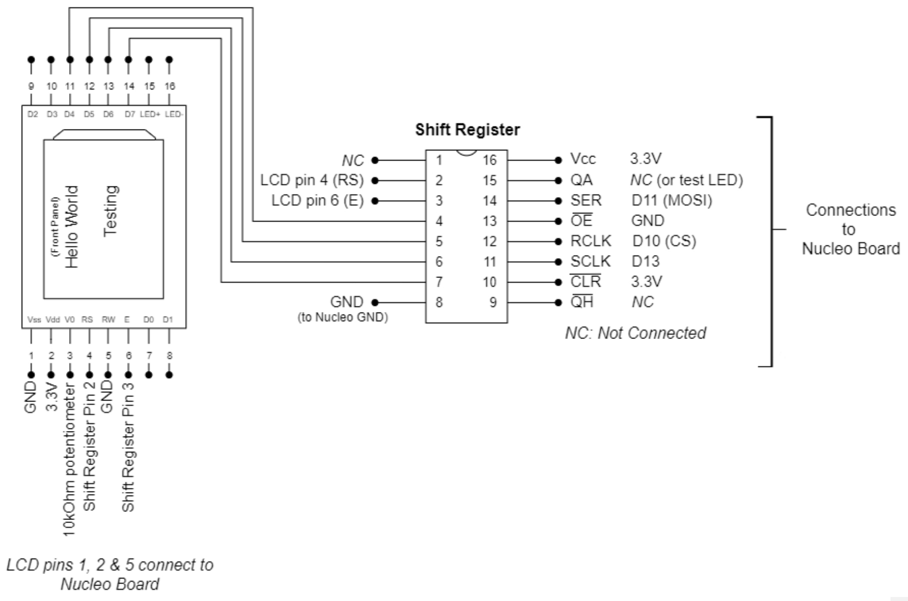


*Figure 5: Connection Diagram for LCD Screen*

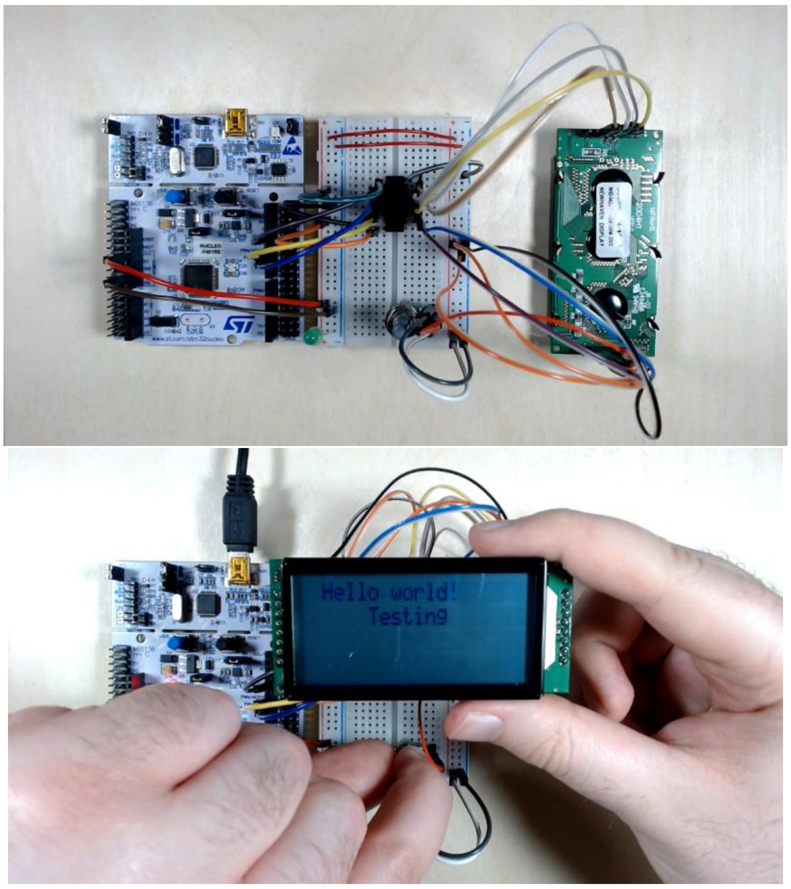


*Figure 6: The completed LCD Circuit and its output*

## 4.4 Writing to the LCD – Example Program

Program Example 2 runs on the hardware we have just connected. While there are a number of libraries available to drive components like the LCD, this program doesn't use them. There are two reasons for this: first, such libraries tend to hide the details of how the program works and second, they leave one dependent on changes to the library which might be introduced by another programmer. This program is hence entirely self-contained. For more advanced programs use of libraries will however become an essential feature.

The program initialises the LCD, and writes two lines of characters to it, applying the requirements of the ST7066U LCD driver (for example the initialisation procedure is found on page 25 of Reference 2). The Busy Flag is not invoked, small delays are inserted into the program when needed to ensure all instructions have adequate time to complete. Remember that the user LED is connected to pin

D13, so we can't use it, as we've committed this pin to SPI. The full mbed API listing can be found in Reference 4, it's always worth having this ready for more detailed reading.

```
/* LCD demo for Lab 1. Sends character strings to LCD
 * LCD is operated in 4-bit mode.  */

#include "mbed.h"

#define ENABLE 0x08         //Will be ORed in to data value to strobe E bit
#define COMMAND_MODE 0x00   //used to clear RS line to 0, for command transfer
#define DATA_MODE 0x04      //used to set RS line to 1, for data transfer

DigitalOut CS(D10);
SPI ser_port(D11, D12, D13); // Initialise SPI, using default settings

//Function Prototypes
void clr_lcd(void);
void init_lcd(void);
void print_lcd(const char *string);
void shift_out(int data);
void write_cmd(int cmd);
void write_data(char c);
void write_4bit(int data);

  //----------- MAIN function ---------------//
int main() {
  CS=1;
  init_lcd();  //initialise the LCD
  clr_lcd();   //Clear the LCD
  //while (true){
    print_lcd("  Hello world!");
    write_cmd(0xc0);   //set cursor to second line
    wait_us(40);
    print_lcd("      Testing");
    while (1) {        //idle in permanent loop
        thread_sleep_for (100);
        }
}       //end of Main

  //----------- Other functions --------------//
void init_lcd(void) {   //follow designated procedure in data sheet
    thread_sleep_for (40);
    shift_out(0x30);    //function set 8-bit
    wait_us(37);
    write_cmd(0x20);    //function set
    wait_us(37);
    write_cmd(0x20);    //function set
    wait_us(37);
    write_cmd(0x0C);    //display ON/OFF
    wait_us(37);
    write_cmd(0x01);    //display clear
    wait_us(1520);
    write_cmd(0x06);    //entry-mode set
    wait_us(37);
    write_cmd(0x28);    //function set
    wait_us(37);
}
void write_4bit(int data, int mode) {  //mode is RS line, ie 0 for cmd, 1 for
data
    int hi_n;
    int lo_n;
    hi_n = (data & 0xF0);           //form the two 4-bit nibbles that will be
sent
    lo_n = ((data << 4) &0xF0);
    shift_out(hi_n | ENABLE | mode); //send each word twice, strobing the
Enable line
    wait_us(1);
    shift_out(hi_n & ~ENABLE);
    shift_out(lo_n | ENABLE | mode);
    wait_us(1);
    shift_out(lo_n & ~ENABLE);
}
void shift_out(int data) {     //Invokes SPI
    CS = 0;
    ser_port.write(data);
    CS = 1;
}
void write_cmd(int cmd) {
    write_4bit(cmd, COMMAND_MODE);
}
void write_data(char c) {
    write_4bit(c, DATA_MODE);    //1 for data mode
}
void clr_lcd(void) {
    write_cmd(0x01);    //display clear
    wait_us(1520);
}
void print_lcd(const char *string) {
    while(*string){
        write_data(*string++);
            wait_us(40);
    }
}
```

*Program Example 2: Writing to the LCD*

Check carefully through the program, and try to grasp its structure and how it works. The comments should aid your understanding. Copy it into Mbed Studio or the on-line compiler, and compile and download to your hardware. Check carefully that the bi-colour on-board LED flickers red-green as download occurs. When the program runs, you should see the message shown in Figure 6, displayed on the LCD.

If the program does not run, press Reset, and/or disconnect and reconnect the USB. This type of LCD is famous for its sensitivity to start-up conditions. Also check all connections very carefully – it's easy to make a mistake with these, and even a single error can cause complete non-functioning of the circuit.

Once you have the program running successfully, try writing different text messages on the screen.

# 5 Applying Asynchronous Serial Transfer and the UART

We turn now to a simple application of data transfer by UART (Universal Asynchronous Receiver/ Transmitter), sending text from the Nucleo F401RE Board to the PC. This is simple, as no extra hardware is required, apart from the Nucleo board.

The Mbed API applies the "Serial" label to asynchronous UART-based communication, even though the Mbed environment makes use of many other serial protocols. Recall that a UART-based serial link has two unidirectional channels, one for sending and one for receiving. As the link is asynchronous, both ends of the data link must be configured to use the same settings, e.g. baud rate and use of parity bit.

The Mbed API distinguishes between buffered and unbuffered use of the UART, with a set of functions for each. Some of the Buffered serial ones are shown in Table 4.

| Function Name | Description |
|---|---|
| **BufferedSerial**(PinName tx, PinName rx, int baud) | Create a Serial port connected to the specified transmit and receive pins, with specified baud rate |
| ssize_t **write**(const void *buffer, size_t length) | Write the contents of a buffer to a file. |
| ssize_t **read** (void *buffer, size_t length) | Read the contents of a file into a buffer. |
| void **set_baud**(int_baud) | Set the baud rate |
| void **set_format**(int bits=8, Parity parity=BufferedSerial::None, int stop_bits=1) | Set the transmission format used by the serial port |

*Table 4: Mbed API for "Serial"*

A very simple application of the mbed serial capability provides a link back to the host computer through the USB cable, using a UART port reserved for that purpose. In current versions of the API, this is now automatically configured. A simple example is shown in Program Example 3.

```
/*UART demo. Sends message to PC terminal
 */

#include "mbed.h"
BufferedSerial pc(USBTX, USBRX,9600);
int cycle=0;

int main() {
  while(true){
    printf("Hello World. This is loop %i \n\r",cycle);
    thread_sleep_for(500);
    cycle++;
  }
}
```

*Program Example 3: Implementing a Simple Link with PC*

As usual, enter this program into Mbed Studio or the on-line compiler, compile and download. To view the output, you need an on-screen virtual serial terminal running on the host computer. Mbed Studio provides this automatically. A number of terminals are available as free download; one which works readily for this application is Tera Term, which can be downloaded from Tera Term - Download (lo4d.com) . An alternative is PuTTY, which can be downloaded from PuTTY - Download (softonic.com) . If you are developing with Mbed Studio, but then wish to use Tera Term or PuTTY, you will need to close down Studio; it is likely otherwise to block access to any other virtual terminal.

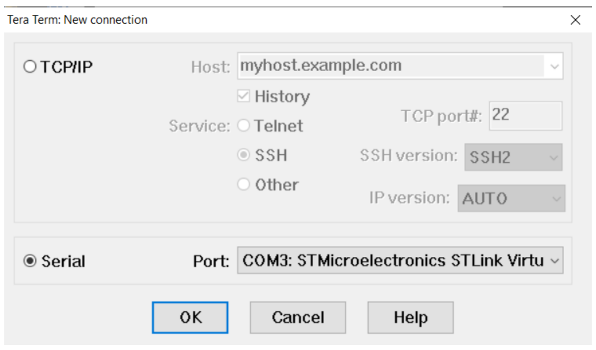
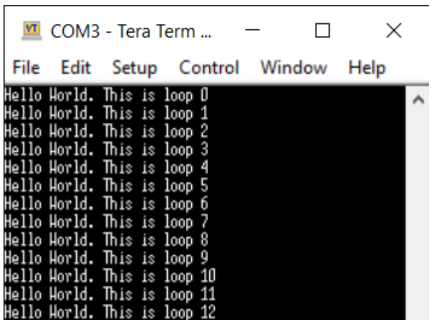


Figure 7: The Tera Term Configuration Screen

The Tera Term Configuration Screen is shown in Figure 7. Select **Serial**, and if your Nucleo is plugged in, you should see it identified in the **Port:** window, as shown. Then click OK, to move to the screen shown in Figure 8. Once Program Example 3 is running it should start showing the output displayed.



```
Hello World. This is loop 0
Hello World. This is loop 1
Hello World. This is loop 2
Hello World. This is loop 3
Hello World. This is loop 4
Hello World. This is loop 5
Hello World. This is loop 6
Hello World. This is loop 7
Hello World. This is loop 8
Hello World. This is loop 9
Hello World. This is loop 10
Hello World. This is loop 11
Hello World. This is loop 12
```

*Figure 8: Tera Term Screen, and Output from Program Example 3*

# 6 Conclusion

You have now experienced building up a useful microcontroller-based sub-system, applying hardware and software of some complexity. While designs have been provided for you, you should try developing simple variations on hardware and/or software, moving towards building up original ideas of your own. All ideas and techniques that have been presented here will be built on in future on-line teaching and labs. Make sure your understanding is clear, so that you have a firm foundation on which to build!

Don't break up the circuit you have built, we develop it further in Lab 2, adding a temperature sensor to produce a fully-functioning and useful temperature measurement system.

# 7 References

1. 74HC595N shift register datasheet

   http://www.nxp.com/documents/data_sheet/74HC_HCT595.pdf

2. ST7066U LCD driver datasheet
   http://www.newhavendisplay.com/app_notes/ST7066U.pdf

*The five web pages below are significant landmarks of the on-line Mbed manual*

3. Introduction to ARM Mbed OS6
   Introduction - Introduction to Mbed OS 6 | Mbed OS 6 Documentation
4. Full Mbed API listing
   Full API list - API references and tutorials | Mbed OS 6 Documentation
5. Mbed Tutorials and Examples
   Tutorials and official examples - Tutorials and examples | Mbed OS 6 Documentation
6. Mbed Components
   Components | Mbed
7. Mbed Forums
   Arm Mbed OS support forum - Get support for Arm Mbed OS from our community and support team

*These books are excellent reference points while programming in C and/or C++*

8. Peter Prinz and Ulla Kirch-Prinz. (2002). *C Pocket Reference*. O'Reilly. ISBN 0-596-00436-2.

9. Kyle Loudon. (2003). *C++ Pocket Reference*. O'Reilly. ISBN 978-0-596-00496-5.