

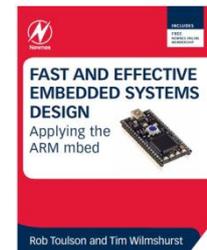
# Embedded Systems Design Course

## Applying the mbed microcontroller

### Parallel data and communication

These course notes are written by R.Toulson (Anglia Ruskin University) and T.Wilmshurst (University of Derby). (c) ARM 2012

These course notes accompany the textbook “Fast and effective embedded system design : Applying the ARM mbed”



# Parallel data and communication

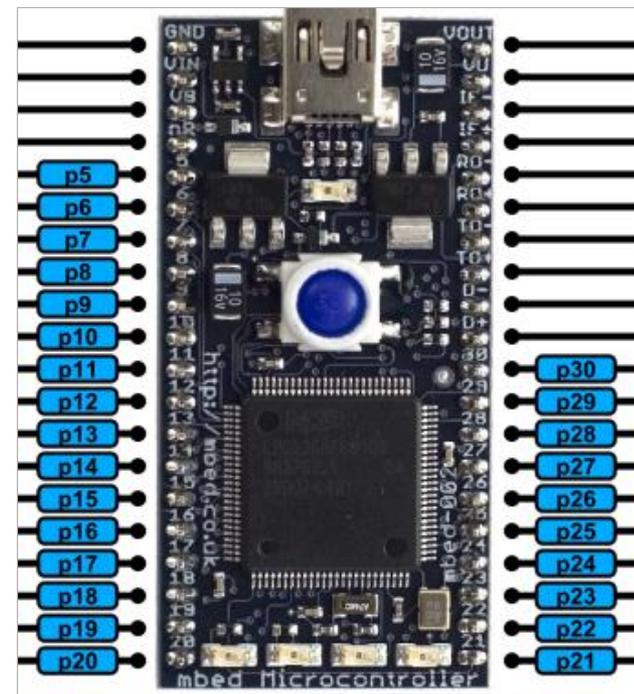
- Using parallel digital outputs with the BusOut object
- Working with a parallel LCD display
- Hardware integration
- Using modular coding to interface the LCD display
- Initialising the LCD display
- Sending parallel display data to the LCD
- Adding data to a specified location
- Using the mbed TextLCD library
- Displaying variable data on the LCD

# Using parallel digital outputs with BusOut

- A digital bus is a system to transfer data between components, or between controllers.
- Buses can be parallel, carrying data on multiple wires, or serial, carrying data sequentially on a single wire.
- Conventional PCI, Parallel ATA and PCMCIA are examples of parallel buses.
- Examples of serial interfaces include Ethernet, FireWire, USB, I<sup>2</sup>C, and SPI.
- The BusOut interface is used to create a number of parallel DigitalOut pins that can be written as one numeric value.
- The BusOut interface can be used to set the state of the output pins, and also read back the current output state.

# Using parallel digital outputs with BusOut

- On the mbed, the four on-board LEDs can be used to create a BusOut interface and have been specially configured to operate with no need for extra wires or connections.
- The mbed also has 26 digital IO pins (pins 5-30) which can be configured as a digital bus for use with the BusOut and BusIn interfaces.



# Using parallel digital outputs with BusOut

The BusOut library functions are shown in the table below

<b>BusOut</b>	<b>A digital output bus, used for setting the state of a collection of pins</b>
BusOut	Create a BusOut object, connected to the specified pins
write	Write the value to the output bus
read	Read the value currently output on the bus
operator=	A shorthand for write
operator int()	A shorthand for read

- We can use digital outputs to switch the on-board LEDs in a specific order.
- For example, we can use the code shown to produce a light that moves horizontally across the 4 on-board LEDs.
- The code is very simple, but it could become quite complex if we require more outputs, or to perform more on/off configurations of the LEDs.

```
#include "mbed.h"

DigitalOut led1(LED1);
DigitalOut led2(LED2);
DigitalOut led3(LED3);
DigitalOut led4(LED4);

int main() {
    while(1) {
        led1 = 1;
        led2 = 0;
        led3 = 0;
        led4 = 0;
        wait(0.25);
        led1 = 0;
        led2 = 1;
        led3 = 0;
        led4 = 0;
        wait(0.25);
        led1 = 0;
        led2 = 0;
        led3 = 1;
        led4 = 0;
        wait(0.25);
        led1 = 0;
        led2 = 0;
        led3 = 0;
        led4 = 1;
        wait(0.25);
    }
}
```

# Using parallel digital outputs with BusOut

- Exercise 1: Using digital outputs, create a program to produce a “Knightrider” LED sweep effect with the on-board LEDs.

```
#include "mbed.h"
DigitalOut led1(LED1);
DigitalOut led2(LED2);
DigitalOut led3(LED3);
DigitalOut led4(LED4);

int main() {
    while(1) {
        led1 = 1; led2 = 0; led3 = 0; led4 = 0;
        wait(0.25);
        led1 = 0; led2 = 1; led3 = 0; led4 = 0;
        wait(0.25);
        led1 = 0; led2 = 0; led3 = 1; led4 = 0;
        wait(0.25);
        led1 = 0; led2 = 0; led3 = 0; led4 = 1;
        wait(0.25);
        led1 = 0; led2 = 0; led3 = 1; led4 = 0;
        wait(0.25);
        led1 = 0; led2 = 1; led3 = 0; led4 = 0;
        wait(0.25);
    }
}
```

# Using parallel digital outputs with BusOut

- Exercise 2: Using the BusOut object, create a program to produce a “Knightrider” sweep effect with the on-board LEDs.
- Verify that this program behaves the same as the previous exercise.

```
#include "mbed.h"

BusOut myleds(LED4, LED3, LED2, LED1);
char x=1;
int main() {
    while(1) {
        for(int i=0; i<3; i++) {
            x = x << 1;           // x = a << b then x = a*2^b;
                                // x=1,2,4,8 or x=0001,0010,0100,1000
            myleds=x;           // sweep left
            wait(0.2);
        }
        for(int i=0; i<3; i++) {
            x = x >> 1;           // x = a >> b then x = a/2^b;
                                // x=8,4,2,1 or x=1000,0100,0010,0001
            myleds=x;           // sweep right
            wait(0.2);
        }
    }
}
```

- Note: Shift operators, << and >>, are used to multiply and divide by two.

# Working with a parallel LCD display

- We will use the 2x16 character Powertip PC1602F LCD, though a number of similar LCD displays can be found with the same hardware configuration and functionality.



- The following must be achieved in order to interface the LCD:
  - Hardware integration: we will need to connect the LCD to the correct mbed pins.
  - Modular coding: as there are many processes that need to be completed, it makes sense to define LCD functions in modular files.
  - Initialising the LCD: a specific sequence of control signals must be sent to the LCD in order to initialise it.
  - Outputting data: we will need to understand how the LCD converts control data into legible display data.

# Hardware integration

- The PC1602F display is a 2x16 character display with an on board data controller chip and an integrated backlight.
- The LCD display has 16 connections as shown here.

Pin Number	Pin Name	Function
1	V <sub>SS</sub>	Power supply (GND)
2	V <sub>DD</sub>	Power supply (5V)
3	V <sub>0</sub>	Contrast adjust
4	RS	Register select signal
5	R/W	Data read / write
6	E	Enable signal
7	DB0	Data bus line bit 0
8	DB1	Data bus line bit 0
9	DB2	Data bus line bit 0
10	DB3	Data bus line bit 0
11	DB4	Data bus line bit 0
12	DB5	Data bus line bit 0
13	DB6	Data bus line bit 0
14	DB7	Data bus line bit 0
15	A	Power supply for LED back light (5V)
16	K	Power supply for LED back light (GND)

# Hardware integration

- The Powertip PC1602F datasheet is available from here:

<http://www.rapidonline.com/netalogue/specs/57-0911.pdf>

- Operation and interfacing the LCD is summarised as follows:
  - The display is initialised by sending control instructions to the relevant configuration registers in the LCD. This is done by setting RS, R/W and E all low, then sending the correct data through bits DB0-DB7.
  - We will use the LCD in 4-bit mode which means that we only need to use the final 4-bits of the data bus (DB4-DB7). This means we can control the LCD with only 7 digital lines, rather than 11 lines which are required for 8-bit mode.
  - After each data byte has been sent, the Enable bit must be toggled on then off again, this tells the LCD that data is ready and should be processed.
  - Once the LCD has been initialised, display data can be sent by setting the RS bit. Again, after each byte of display data has been sent, the Enable bit should be toggled to process the data.

# Hardware integration

- We obviously need a digital mbed pin to attach to each of the LCD data pins. We need 4 digital outputs to send the 4-bit instruction and display data and 3 digital outputs to manipulate the RS, R/W and E control flags.

- We can connect the PC1602F to the mbed using the following interface configuration:

mbed Pin number	LCD Pin Number	LCD Pin Name	Power Connections
1	1	V <sub>SS</sub>	0V
39	2	V <sub>DD</sub>	5V
1	3	V <sub>0</sub>	0V
19	4	RS	
1	5	R/W	0V
20	6	E	
21	11	DB4	
22	12	DB5	
23	13	DB6	
24	14	DB7	
39	15	A	5V
1	16	K	0V

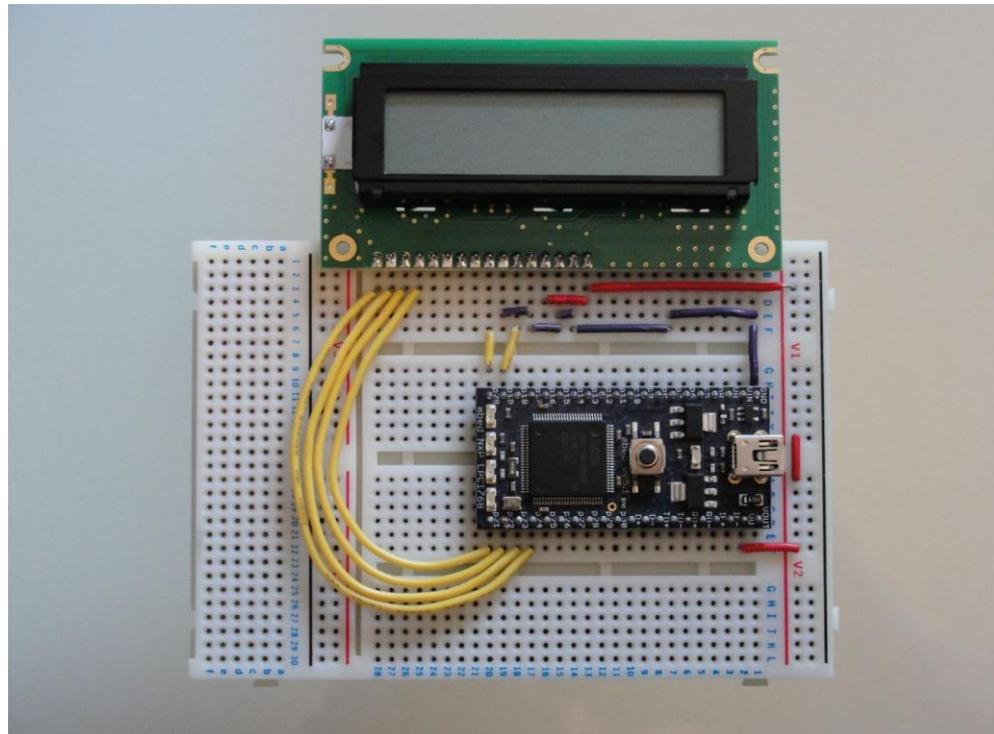
- Note: in general, we only use the LCD in write mode, so we tie R/W permanently to ground (mbed pin 1).

# Hardware integration

- Note that the PC1602F has a non-conventional pin layout which reads from left to right:

14	13	12	11	10	9	8	7	6	5	4	3	2	1	16	15
DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	E	R/W	RS	V <sub>0</sub>	V <sub>DD</sub>	V <sub>SS</sub>	A	K

The hardware setup is as shown.



# Using modular coding to interface the LCD

- We will use three files for the LCD application:
  - A main code file (main.cpp) which can call functions defined in the LCD feature file.
  - An LCD feature file (LCD.cpp) which will include all the functions for initialising and sending data to the LCD.
  - An LCD header file (LCD.h) which will be used to declare data and function prototypes.

# Using the LCD display

- We will declare the following functions in our LCD header file
  - toggle\_enable - function to toggle the enable bit
  - LCD\_init - function to initialise the LCD
  - display\_to\_LCD - function to display characters on the LCD
- Our LCD.h file should therefore be as follows:

```
// LCD.h file

#ifndef LCD_H
#define LCD_H

#include "mbed.h"

void display_to_LCD(char value);    //function to display characters on the LCD
void toggle_enable(void);          //function to toggle the enable bit
void LCD_init(void);               //function to initialise the LCD

#endif
```

# Initialising the LCD display

- In LCD.cpp:
  - We define the digital IO classes for the mbed. We need one digital output for each of RS and E and we will use the mbed BusOut class for the 4-bit data.
  - We send a number of 4-bit data packets to the LCD in order to initialise it and to display alphanumeric characters on the display.
  - After each data packet has been sent, the LCD requires the Enable bit to be toggled (i.e. sent high and then low with a pause in between). This is done by the **toggle\_enable** function.

```
// define mbed objects
DigitalOut RS(p19);
DigitalOut E(p20);
BusOut data(p21, p22, p23, p24);
```

```
/** toggle enable function
void toggle_enable(void) {
    E=1;
    wait(0.001);
    E=0;
    wait(0.001);
}
```

# Initialising the LCD display

- A specific initialisation procedure must be followed in order for the PC1602F display to operate correctly. Please refer to the PC1602F datasheet for more specific configuration details.
- We will code the initialisation routine using the **LCD\_init** function:
  - In order to initialise we first need to wait a short period (approximately 20ms), set the RS and E lines to zero and then send a number of configuration messages to set up the LCD functionality.
  - To set the LCD **function mode** we send a binary value of 0010 1000 (0x28 hex) to the LCD data pins, we define 4-bit mode, 2 line display and 5x7 dot characters.
  - Note that as we are using 4-bit mode, we need to send two pieces of 4-bit data for each instruction; effectively setting one 8-bit register by sending two 4-bit packets of data.

```
// Function Mode
data=0x2;
toggle_enable();
data=0x8;
toggle_enable();
```

# Initialising the LCD display

- The LCD **display mode** control register must also be set during initialisation.
- Here we need to send a command to switch the display on, and to determine the cursor function. Value 0x0F will switch the display on with a blinking cursor.
- Before data can be written to the display, the display must first be cleared and the cursor reset to the first character in the first row. This is done by the **clear display** command with a value 0x01.

```
// Display Mode  
data=0x0;  
toggle_enable();  
data=0xF;  
toggle_enable();
```

```
// Clear display  
data=0x0;  
toggle_enable();  
data=0x1;  
toggle_enable();
```

# Sending parallel display data to the LCD

- Display data is sent to the LCD screen by the `display_to_LCD` function. This function performs the following tasks:
  - Setting the RS flag to 1 (data setting).
  - Sending a data byte describing the ascii character to be displayed.
  - Toggle the enable flag.
  - The character displayed to the LCD is described by an 8-bit hexadecimal ascii value. The complete ascii table is included with the LCD datasheet.

```
/**** display ***/  
void display_to_LCD(char value ){  
  
    RS=1;  
  
    /***** display character  
  
    data=value>>4;        // upper 4 bits  
    toggle_enable();  
    data=value&0x0F;     // lower 4 bits  
    toggle_enable();  
  
}
```

# Sending parallel display data to the LCD

- If we wish to display the word “HELLO”, for example, the hexadecimal ascii values required are as follows: 0x48, 0x45, 0x4C, 0x4C and 0x4F.
- Other ascii values can be found in the table below:

		LSB															
		0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xA	0xB	0xC	0xD	0xE	0xF
MSB	0x0																
	0x1																
	0x2		!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
	0x3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
	0x4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
	0x5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
	0x6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
	0x7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	

## Digital IO and LCD functions are therefore defined in LCD.cpp as below

```
// LCD.cpp file

#include "LCD.h"

DigitalOut RS(p19);
DigitalOut E(p20);
BusOut data(p21, p22, p23, p24);

void toggle_enable(void){
    E=1;
    wait(0.001);
    E=0;
    wait(0.001);
}

//initialise LCD function
void LCD_init(void){
    wait(0.02);
    RS=0;
    E=0;

    //function mode
    data=0x2;
    toggle_enable();
    data=0x8;
    toggle_enable();

    // continued...
```

```
//... LCD.cpp continued...

//display mode
data=0x0;
toggle_enable();
data=0xF;
toggle_enable();

//clear display
data=0x0;
toggle_enable();
data=0x1;
toggle_enable();
}

//display function
void display_to_LCD(char value ){
    RS=1;
    data=value>>4;
    toggle_enable();
    data=value&0x0F;
    toggle_enable();
}
```

# Using the LCD display

- Exercise 3: Connect the LCD to an mbed and construct a new program with the files main.cpp, LCD.cpp and LCD.h ,as described in the previous slides.

Add the following code to your main.cpp file and compile and run on the mbed:

```
#include "LCD.h"

int main() {
    LCD_init();
    display_to_LCD(0x48); // 'H'
    display_to_LCD(0x45); // 'E'
    display_to_LCD(0x4C); // 'L'
    display_to_LCD(0x4C); // 'L'
    display_to_LCD(0x4F); // 'O'
}
```

Verify that the word 'HELLO' is correctly displayed with a flashing cursor.

# Using the mbed TextLCD library

- The mbed TextLCD library is more advanced than the simple functions we have created.
  - The TextLCD library performs the laborious LCD setup routine for us
  - The TextLCD definition also tells the LCD object which pins are used for which functions

- The pin definition is defined in the following manner:

```
TextLCD lcd(int rs, int e, int d0, int d1, int d2, int d3);
```

- We need to ensure that our pins are defined in the same order. For our hardware setup this will be:

```
TextLCD lcd(p19, p20, p21, p22, p23, p24);
```

- We use printf statements to display characters on the LCD screen.

# Using the mbed TextLCD library

- Exercise 4: Compile a “Hello World” example using the mbed library, which makes use of an alphanumeric LCD much simpler and quicker to program.

```
#include "mbed.h"
#include "TextLCD.h"
TextLCD lcd(p19, p20, p21, p22, p23, p24); //rs,e,d0,d1,d2,d3

int main() {
    lcd.printf("Hello World!");
}
```

- Import the mbed TextLCD.h library file to your project (right click and select ‘import library’).
- This library is effectively a file full of specific LCD functions already written for you. Use the following link for the library file:

<http://mbed.org/users/simon/libraries/TextLCD/livod0>

# Using the mbed TextLCD library

- The cursor can be moved to a chosen position to allow you to choose where to display data, for example

```
lcd.locate(3,1);
```

- The display is laid out as 2 rows (0-1) of 16 columns (0-15). The locate function defines the column first followed by the row.
  - The above example moves the cursor to the 4<sup>th</sup> column and 2<sup>nd</sup> line
  - Any printf statements after the locate command will be printed at the new cursor location.
- We can also clear the screen with the following command:

```
lcd.cls();
```

# Using the mbed TextLCD library

- We display data on the screen using the standard printf statement too. If we want to display the value of an integer to the screen, we need to:
  - declare the variable
  - give the variable a value
  - display the variable to the screen by using the printf statement,

```
x = 1028  
lcd.printf("%i", x);
```

- Note that the “%i” command is used to indicate that an integer is to be output, and the integer name follows.

# Displaying variables on the LCD

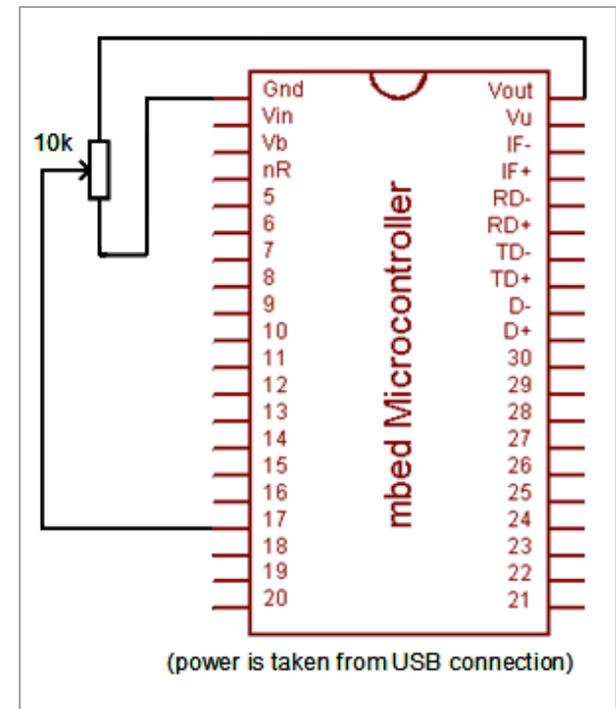
- Exercise 5: display a continuous count variable on the LCD display by implementing the following code:

```
#include "mbed.h"
#include "TextLCD.h"
TextLCD lcd(p19, p20, p21, p22, p23, p24); // rs, e, d0, d1, d2, d3
int x=0;
int main() {
    lcd.printf("LCD Counter");
    while (1) {
        lcd.locate(5,1);
        lcd.printf("%i",x);
        wait(1);
        x++;
    }
}
```

- Don't forget to import the TextLCD library!
- Increase the speed of the counter and investigate how the cursor position changes as the count value increases.

# Displaying analog input data on the LCD

- Exercise 6: Display the analog value to the screen.
  - You will need to use a potentiometer to provide an analog input.
  - The analog input variable has a floating point value between 0 and 1, where 0 is 0V and 1 represents 3.3V.
  - We will multiply the analog input value by 100 to make it a percentage between 0-100%.
  - An infinite loop is used so that the screen continuously updates automatically. To do this it is necessary to clear the screen and add a delay to set the update frequency.



# Displaying analog input data on the LCD

- Add the following to your main.cpp . Your code should now compile and run:

```
#include "mbed.h"
#include "TextLCD.h"

TextLCD lcd(p19, p20, p21, p22, p23, p24); //rs,e,d0, d1,d2,d3
AnalogIn Ain(p17);
int percentage;

int main() {
    while(1){
        percentage=Ain*100;
        lcd.printf("%i",percentage);
        wait(0.002);
        lcd.cls();
    }
}
```

- The analog value should change as you move the position of the potentiometer.

# Extended task

- Exercise 7: Create a program to make the mbed and display act like a standard voltmeter. Potential difference should be measured between 0 - 3.3 V and display this to the screen similar to that shown below:



- Note of the following:
  - You will need to convert the 0.0 - 1.0 analog input value to a value which represents 0 - 3.3 Volts.
  - You will need to use an infinite loop to allow the voltage value to continuously update as the potentiometer position changes.
  - Check your display with the reading from an actual voltmeter – is it accurate?

# Summary

- Using parallel digital outputs with the BusOut object
- Working with a parallel LCD display
- Hardware integration
- Using modular coding to interface the LCD display
- Initialising the LCD display
- Sending parallel display data to the LCD
- Adding data to a specified location
- Using the mbed TextLCD library
- Displaying variable data on the LCD