Timers and LCD

Solderless breadboard

In the following exercises, you need to wire an LCD to your LPCXpresso. For wiring we use solderless breadboard and solid wires.

Get a PSoC programmer from the library. You don't the programmer for anything but you need the other parts of the kit, breadboard and lcd.

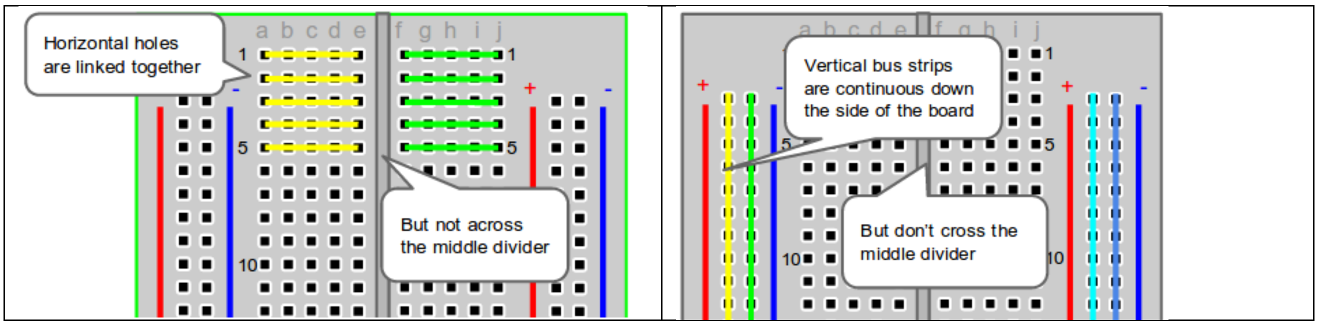
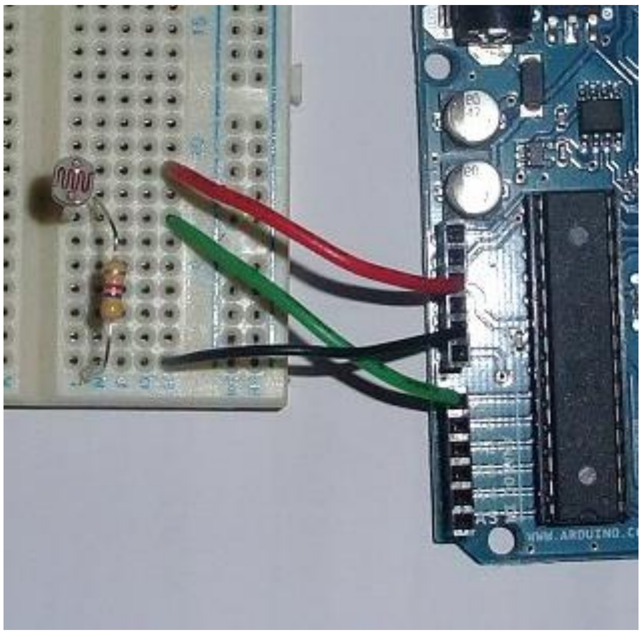

Horizontal holes are linked together
But not across the middle divider
Vertical bus strips are continuous down the side of the board
But don't cross the middle divider

Figure 1 Breadboard strip connections



Picture 1 Example of breadboard wiring

Exercise 1

Systick timer is meant for time keeping, typically with millisecond accuracy. Some devices require small delays (micro or nanoseconds granularity) at irregular intervals which calls for other type of implementation. Your task is to implement microsecond resolution wait function using Repetitive Interrupt Timer (RIT) and to use it in an LCD class.

First read chapter 20 of LPC1549 user manual (UM10736.pdf). Then study ritimer_15xx.h in lpc_chip_15xx/inc. The header contains declarations of functions and control bits that are needed to manage RIT. The control block to pass to the functions is called LPC_RITIMER. For example to initialize RIT you call:

```
Chip_RIT_Init(LPC_RITIMER); // initialize RIT (enable clocking etc.)
```

You must call the function above after the board initialization in main() to enable clock signal to the timer component. By default, the clock of RIT is disabled to save power.

RIT timer can trigger an interrupt when the timer expires and interrupt flag is set. In this case the interrupt will not be enabled. Instead the wait function will poll the interrupt flag. When the interrupt flag is set then we stop the timer and the wait ends.

Modify delayMicroseconds function to calculate RIT compare value from Chip_Clock_GetSystemClockRate() and the requested wait time. Note that the compare value is a 64-bit unsigned integer so you must use 64-bit variables for calculations. Use uint64_t for 64-bit integers.

```
void delayMicroseconds(unsigned int us)
{
// calculate compare value
// disable RIT – compare value may only be changed when RIT is disabled
// set compare value to RIT
// clear RIT counter (so that counting starts from zero)
// enable RIT
// wait until RIT Int flag is set
// disable RIT
// clear RIT Int flag
}
```

Prove that your timer works by writing a program that sends the following pulses (in a loop) to PIO0_10 (Digital 4):

- 55 us high
- 35 us low
- 40 us high
- 20 us low

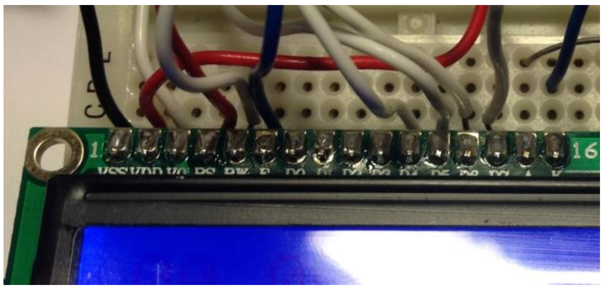
Ask instructor to verify timing with an oscilloscope. You need an accurate delay for proper operation of the LCD. With too short delay, the display will not work at all and a too long delay adds an awful lot latency to the program.

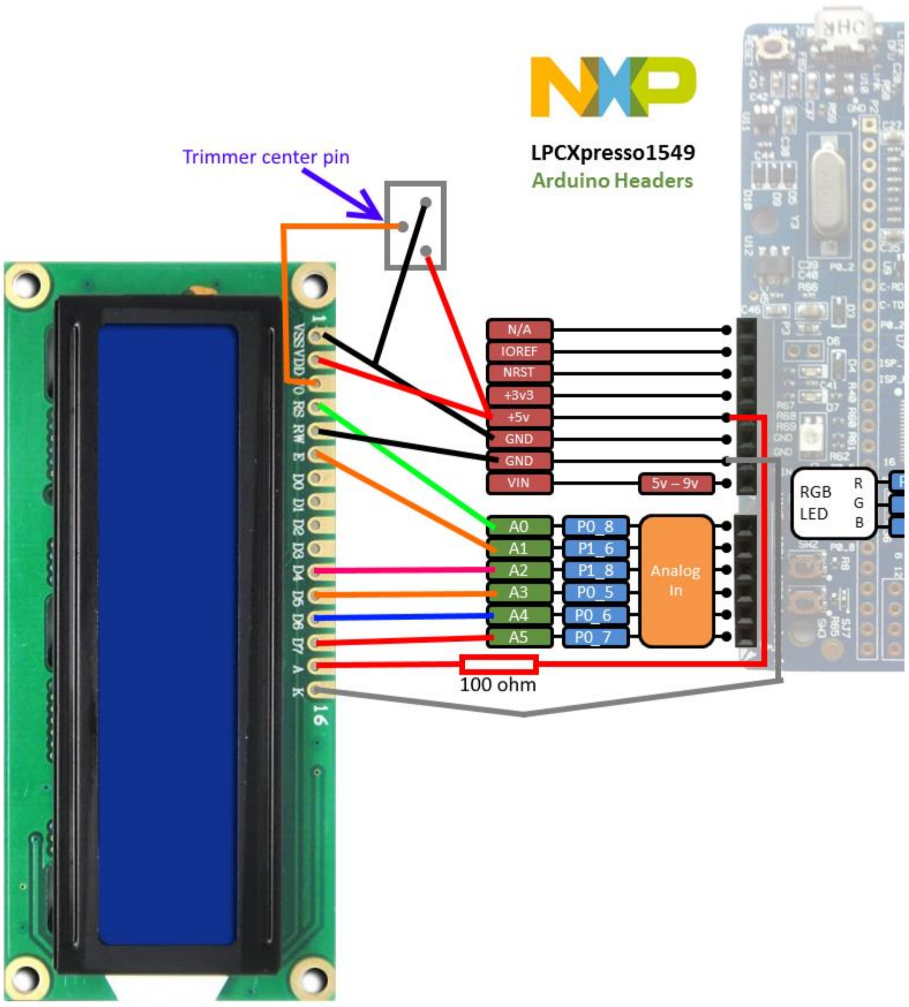
Wire the LCD to your LPCXpresso. Connect the LCD to the breadboard. Use a trimmer for contrast adjustment. The center pin of the trimmer is connected to pin 3 (V0) of the LCD and the other trimmer pins are connected to GND and 5V.

LPCXpresso pin	LCD pin
A5	14 (DB7)
A4	13 (DB6)
A3	12 (DB5)
A2	11 (DB4)
A1	6 (E)
GND	5 (R/!W)

A0	4 (RS)
(Trimmer center)	3 (V0)
5V	2 (VDD)
GND	1 (VSS)

The wiring diagram also shows how to wire display back light. Some LCD modules don't have a back light. The backlight must always have a 100 ohm resistor to limit backlight current.





Picture 2 LCD wiring diagram

Test and adjust the LCD

Create an LPCOpen C++ project.

Add LiquidCrystal.h and LiquidCrystal.cpp to the project. They are originally part of Arduino LCD library and have been adapted to use DigitalIoPin class instead of Arduino pin number plus some other minor changes. You will also need the microsecond delay function from exercise 1.

See https://www.arduino.cc/en/Reference/LiquidCrystal for library documentation.

To test and adjust the contrast of the lcd, write a program that creates a LiquidCrystal object and prints a single “A” on the screen. See Arduino Hello World example for details. Note that objects should not be created global. You will need to setup the timer before creating LiquidCrystal object! Since we don’t have full Arduino library you can only print one character at a time using write() method.

Now that you have printed “A” on the screen we can adjust the contrast to see if the library works.

Turn the trimmer until you see the text. The proper trimmer setting is fairly near the limit of the trimmer and it is very easy to turn past the proper setting. Turn the trimmer carefully! If you cannot finds a trimmer setting that brings “A” to the display then check your pin assignments, wiring and your source code. If you are still having trouble the text ask the instructor for help.

A typical software problem is that the delay function does not work properly. If the delay is too short, the display can’t keep up with the data the library is sending and will not display anything.

Improve LCD functionality

Add Print method to the library. There should be two variants of the method:

```
void LiquidCrystal::Print(std::string const &s)
void LiquidCrystal::Print(const char *s)
```

Both should print the given string on the LCD.

Write a program that reads the state of the buttons and prints state of the buttons on the LCD.

Your program should print the button states in the following format:

Example 1:

```
B1    B2    B3
UP    DOWN  DOWN
```

Must always be aligned!

Example 2:

```
B1    B2    B3
UP    UP    UP
```

On the top line there must be four spaces between button numbers. The value on the lower line must be “UP” or “DOWN” according to the state of the corresponding button (down = pressed). The text must be aligned with the corresponding button number regardless of the number of buttons pressed.

Exercise 2

Write a program that prints the real time clock (hours, minutes, seconds) on the LCD.

Example 1:

```
15:23:59
```

A leading space is printed if hours have only one digit

Example 2:

```
 9:05:01
```

Implement a real time clock class that has a member function (method) called **tick** that is called from the Systick interrupt. Since **tick** is called by an ISR all data members that tick uses must be declared **volatile**. See the example below.

```
class RealTimeClock {
public:
            void tick();
private:
            volatile int hour;
            volatile int min;
            volatile int sec;
};
```

Volatile keyword tells the compiler that the value may change outside the observable flow of program, the value may change even if the current code does not modify the value.

The C++ standard library does not provide a proper date type. C++ inherits the structs and functions for date and time manipulation from C. To access date and time related functions and structures, you would need to include <ctime> header file in your C++ program.

The constructor of the class must always take systick rate as a parameter. Systick rate is needed to know the number of tick() calls is makes per second. The class must also provide a constructor that takes the current time as a parameter (in addition to systick rate). If no

```
static volatile RealTimeClock *clock;

#ifdef __cplusplus
extern "C" {
#endif
/**
 * @brief   Handle interrupt from SysTick timer
 * @return  Nothing
 */
void SysTick_Handler(void)
{
    if(clock != NULL) clock->tick();
}
#ifdef __cplusplus
}
#endif
```

Set this to point to your object so that ISR can call tick().

Example code

Critical section functions and some usage examples.

```
// returns the interrupt enable state before entering critical section
bool enter_critical(void)
{
    uint32_t pm = __get_PRIMASK();
    __disable_irq();
    return (pm & 1) == 0;
}

// restore interrupt enable state
void leave_critical(bool enable)
{
    if(enable) __enable_irq();
}


// example of critical section usage
void RealTimeClock::gettime(struct tm *now){
    bool irq = enter_critical();
    now->tm_hour = hours;
    // do same for minutes, secs, years, months
    leave_critical(irq);
}
```

This must to be inside a critical section to ensure that time is not updated during read. An update in the middle of reading would leave us with an inconsistent time.

Exercise 3

Implement an alarm clock with the following features:

- User can set the time
- User can set the alarm time
- User can enable/disable alarm
- When the alarm goes off user can acknowledge the alarm
- Alarm is indicated by blinking a led