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.

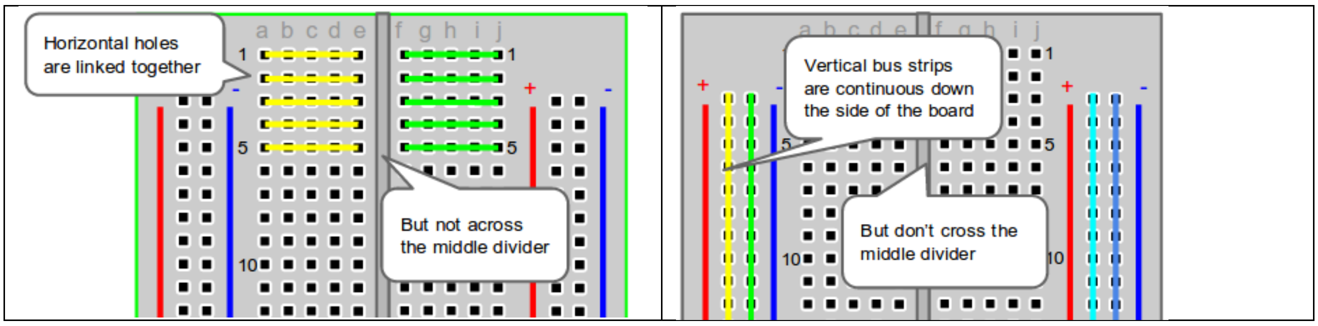
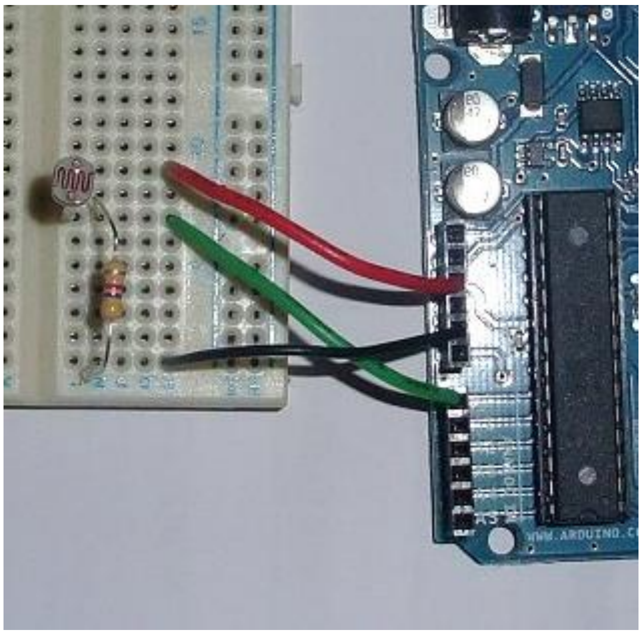

Horizontal holes are linked together
But not across the middle divider
Vertical bus strips are continuous down the side of the board
But don't cross the middle divider

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 signal 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
}
```

You need to implement all the things listed in the comments. Stick to the order given here.

Poll the flag in a loop

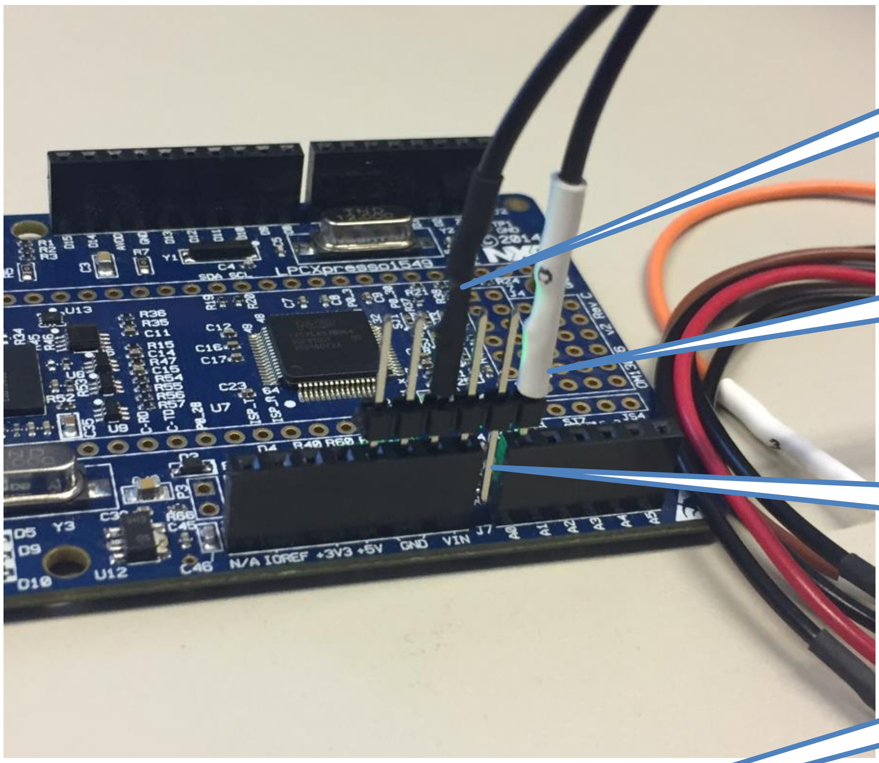
Note that you do not need to write an ISR or enable RIT interrupts. The delay is implemented by polling the RIT interrupt flag.

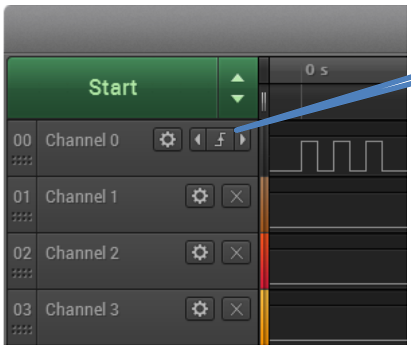
Prove that your timer works by writing a program that sends the following pulses (in a loop) to A0 (Port 0, Pin 8):

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

Verify timing with a logic analyzer or an oscilloscope. Show the result to instructor before wiring the display to LPCXpresso.

Note that entering and exiting a function adds some overhead to execution time. There is overhead from entry/exit to delayMicroseconds and also IO-pin object member functions. Typical overhead is in the range of 5 – 25 us. You should see the same overhead in each signal length. If the overhead is X, the times you measure will be 55+X, 35+X, etc. For example, with 10 us overhead you will get 65 us, 45 us, etc. You don’t need to compensate this overhead.


Connect logic analyzer ground to GND
Connect Channel 0 to A0 (Port0, Pin8).
One pin goes between connectors.
Capture on rising edge.



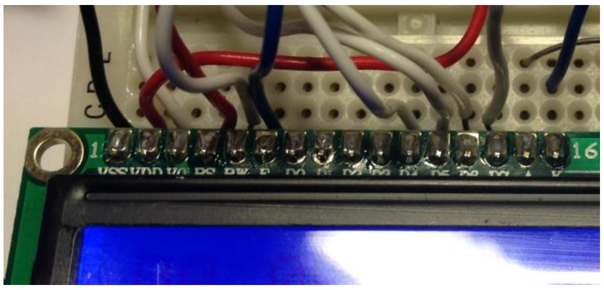
Zoom out and measure each pulse by moving cursor to the pulses. When cursor is between edges, you can see time measurement.

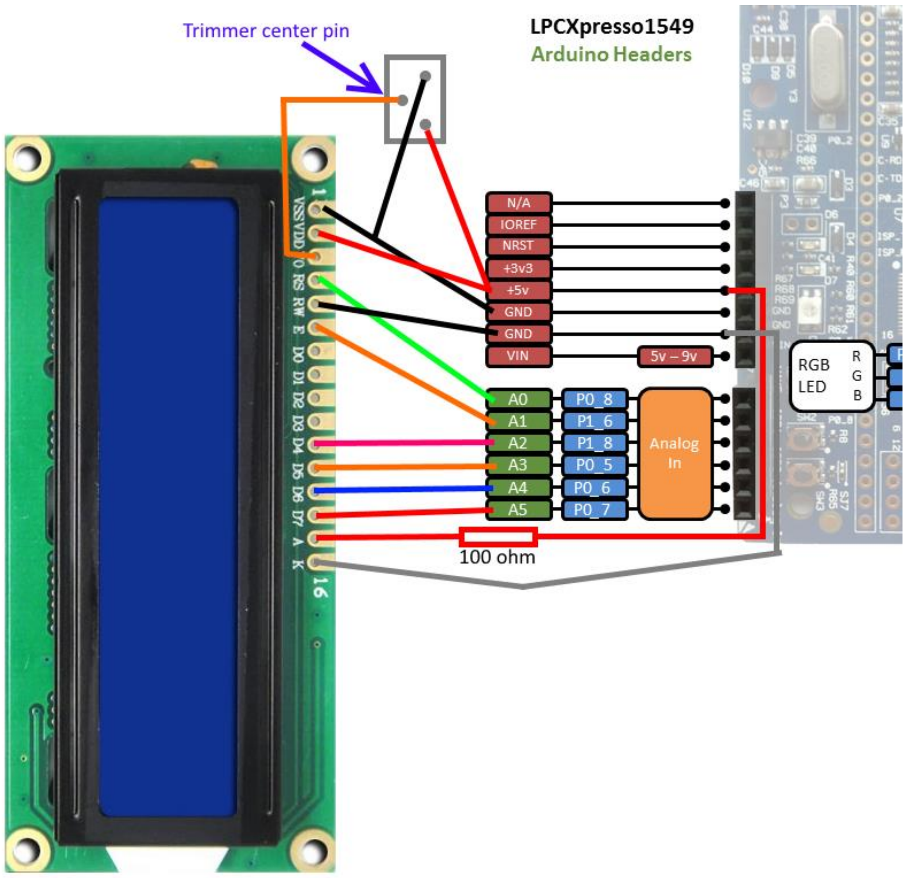
You need an accurate delay for proper operation of the LCD. With too short delay, the display will not work at all and a too long delay adds an awful lot latency to the program.

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 numbers plus some other minor changes. You will also need the microsecond delay function you just implemented and tested.

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.

```
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);
// configure display geometry
lcd.begin(16, 2);
// set the cursor to column 0, line 1
// (note: line 1 is the second row, since counting begins with 0):
lcd.setCursor(0, 1);
// Print a message to the LCD.
lcd.write('A');
```

Pointers to IO-pins

Note that objects may not be created global. You will need to setup the RIT timer before creating LiquidCrystal object! Since we don’t have working implementation of print() in the 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 seeing the letter 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 member function to the library. There should be two variants of the function:

```
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
```

Must always be aligned!

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 must be 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**. Year, month, day, hour, etc. must be stored in separate private member variables.

See the example below.

```
class RealTimeClock {
public:
    // You must also provide a copnstructor variant with systick rate and current time
    RealTimeClock(int ticksPerSecond);
    void gettime(tm *now);
    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 provides chrono library for time management however, we can also use time manipulation functions and structures from C standard library. To access date and time related functions and structures from C standard library, you will 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 ISR makes per second. The class must also provide a constructor variant that takes the current time as a parameter (in addition to systick rate). If only systick rate is given then the time must be set to 23:58.35.

```
static RealTimeClock *rtc;

#ifdef __cplusplus
extern "C" {
#endif
/**
 * @brief   Handle interrupt from SysTick timer
 * @return  Nothing
 */
void SysTick_Handler(void)
{
    if(rtc != NULL) rtc->tick();
    // you may add you own stuff here, for example the sleep counter
}
#ifdef __cplusplus
}
#endif
```

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

Create the ohject in main() and assign the address of your RTC object here before you start systick timer.

Example code

Critical section functions and some usage examples.

```
// header of interrupt locking “mutex”
#ifndef IMUTEX_H_
#define IMUTEX_H_

class Imutex {
public:
    Imutex();
    ~Imutex();
    void lock();
    void unlock();
private:
    bool enable;
};

#endif /* IMUTEX_H_ */

// implementation of interrupt locking “mutex”
#include "chip.h"

#include "Imutex.h"

Imutex::Imutex() : enable(false)
{
}

Imutex::~Imutex()
{
}

void Imutex::lock()
{
    enable = (__get_PRIMASK() & 1) == 0;
    __disable_irq();
}

void Imutex::unlock()
{
    if(enable) __enable_irq();
}

// example of critical section usage (not a complete class)
// class declaration
class RealTimeClock {
    void gettime(tm *now);
    void tick();
    // add more members as needed
private:
    Imutex guard;
};
```

```
#include <mutex>
#include "Imutex.h"

// Get current time
void RealTimeClock::gettime(tm *now){
    std::lock_guard<Imutex> lock(guard);
    now->tm_hour = hour;
    now->tm_min = min;
    // do same for seconds, years, months
    // you may not do any kind of processing or printing here
    // all processing must be done outside the critical section (outside this function)
    // Interrupts are enabled/restored at the end by lock_guard destructor
}
```

Constructor enters critical section and saves previous critical section state.

Destructor is called automatically when we exit the function. Destructor exits critical section (restores previous state).

Exercise 3

You need to study chapter 12 of LPC1549 user manual and pinint15xx.h. The headers can be found in the chip library (lpc_chip_15xx/inc).

There are eight pin interrupts available. Their interrupt numbers are defined in cmsis.h. The numbers are: *`PIN_INT0_IRQn - PIN_INT7_IRQn`*.

Interrupt handlers: **`void PIN_INT0_IRQHandler(void)`** - **`void PIN_INT7_IRQHandler(void)`**. Remember that interrupt handlers must use C-naming convention.

Study example periph_pinit in the library/examples zip file (lpcopen_2_20_lpcxpresso_nxp_lpcxpresso_1549.zip)

In this exercise you need to implement a GPIO isr that increments a global variable when the button is pressed. The main first performs necessary initializations and enters an infinite loop where the program sleeps for 100 ms and reads the global variable. If the variable value is different from previous read the program prints the difference.

For example:

```
Button was pressed: 1 times

Button was pressed: 1 times

Button was pressed: 2 times
```

The main program needs to do the following initializations:

- Configure SW1 as input with pullup
- Initialize pin interrupt hardware
- Configure and enable pin interrupt in the pin interrupt hardware
    - Configure **pin interrupt 0** generate interrupt when it sees a falling edge on SW1
- Enable pin interrupt 0 in NVIC

The pin interrupt handler must do the following:

- Acknowledge the interrupt (clear interrupt status)
- Increment counter