Laboratory #3
PIC32 Basic Digital I/O Port Interfacing

Objectives:

- Use the PIC32 I/O ports for interfacing simple digital I/O devices such as pushbutton switches and LEDs.

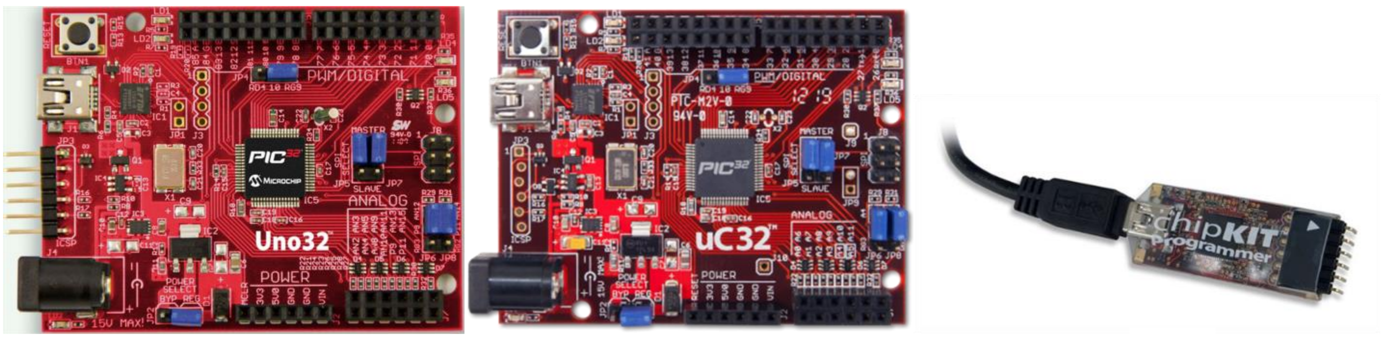
Files provided on Blackboard:

- TLights.h
- TLights.c (incomplete)
- Main.c (incomplete)
- TLights.X.production.hex

Introduction:

We will be using the chipKIT UNO32 or uC32 development boards, from Digilent Inc, for the practical lab assignments in this course. Both boards use the PIC32 microcontroller running at 80 MHz clock frequency. However, the amounts of available program flash and data RAM memory are what distinguish the two systems apart. The chipKIT UNO32 is based on the PIC32MX320F128H processor with 128 KBytes of program flash memory and 16 KBytes of SRAM data memory; whereas the chipKIT uC32 is based on the PIC32MX340F512H with 512 KBytes of program flash memory and 32KBytes of SRAM data memory.

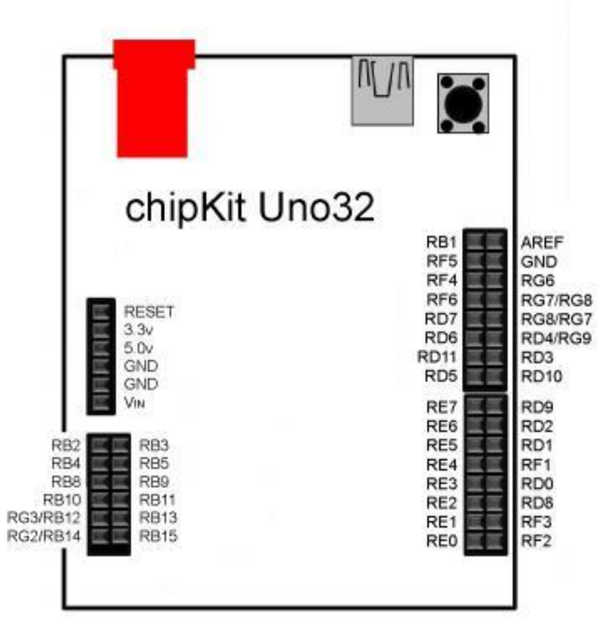
Both of these boards have Arduino Uno form factor and are claimed to be compatible with many Arduino shields. Although these boards can be programmed using Arduino IDE or an Arduino-like programming environment, called MPIDE, for the CE-420 class we will not be using those tools. Arduio like tools are good for beginner programmers and provides lots of functions that simplify the task for the end user. However, it does not give you the power to do in-circuit debugging. Moreover, as part of the learning experience in CE-420, students will be developing their own device drivers and functions to acquire intimate knowledge in the workings of the microcontroller. Thus, for the lab activities we will be using the MPLABX development environment and a low-cost programmer/debugger device called chipKIT PGM Programmer/Debugger, from Digilent Inc.



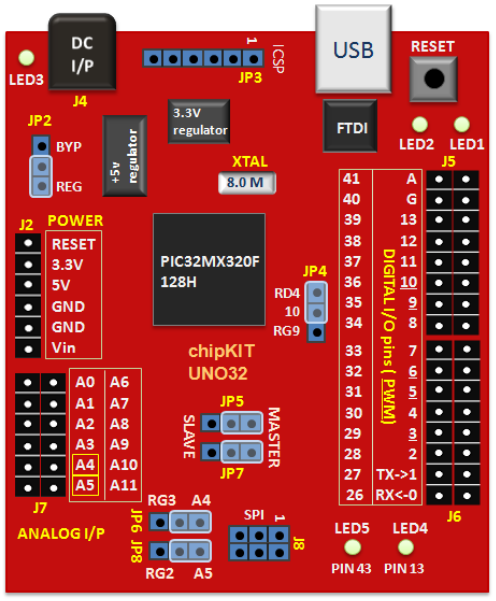
(a) (b) (c)

Figure 1: a) chipKIT UNO32, (b) chipKIT uC32, and (c) chipKIT PGM Programmer/Debugger

Both the chipKIT UNO32 and uC32 boards provide 42 I/O pins that support a number of peripheral functions, such as UART, SPI, and I2C ports and pulse width modulated outputs. Twelve of the I/O pins can be used as analog inputs or as digital inputs/outputs.



(a)



(b)

Figure 2: a) I/O pins of chipKIT Uno32, and (b) Arduino style pinout of chipKIT UNO32

For this course, you will be using the chipKIT UNO32 or uC32 board with the custom built I/O shield shown below in Figure 3. The I/O shield comes with some quite a few I/O modules, such as pushbutton switches, LEDs, seven-segment displays, potentiometer, and other on-board devices such as light sensor, temperature sensor, microphone, buzzer, accelerometer, gyro, EEPROM, and header pins for mounting keypad, OLED, ultrasonic sensor, and Bluetooth module.

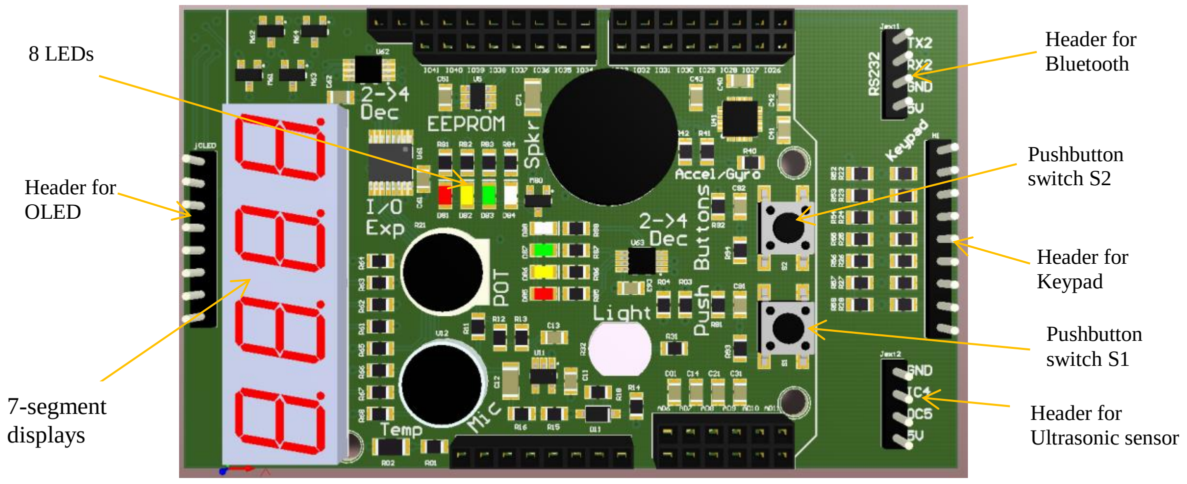


Figure 3: I/O shield for the chipKIT UNO32 and uC32

Task:

In this lab assignment you will be implementing a program, with multiple functional modules, that mimics a simple traffic light controller with different operating modes as specified below:

a) MODE1: in this mode the two sets of four LEDs, Red, Yellow, and Green behave just like a simple traffic light at an intersection with the lights cycling through the three colors in the order given here:
   a. Direction 1: Green and Direction 2: Red for 4 seconds
   b. Direction 1: Yellow and Direction 2: Red for 1 second
   c. Direction 1: Red and Direction 2: Green for 4 seconds
   d. Direction 1: Red and Direction 2: Yellow for 1 second

b) MODE2: This mode behaves live a flashing Red light mode in which the Red LEDs in both directions flash on and off for 500 msec each; i.e. 500 msec on and then 500 msec off.

c) MODE3: This mode is used to activate pedestrian crossing lights (using the white LEDs). When a user activates this mode, the following sequences of steps need to be performed:
   a. Complete one cycle of MODE1 (to let any cars currently near the intersection pass through it before activating the pedestrian crossing signals)
   b. Turn on the Red LEDs in both directions for 1000 msec.
   c. Turn on the White LEDs in both directions for 5 seconds, while the Red LEDs are still on in both directions, to notify pedestrians that it is safe to cross the streets.
   d. Flash the White LEDs on and off for five times while keeping the Red LEDs on.
   e. Return to MODE1

The mode selection is to be made using the two pushbutton switches. Initially, when the program starts running or after reset, the lights start in MODE1. Activating pushbutton SW1 allows the user to toggle between MODE1 and MODE2, while SW2 is used exclusively for activating MODE3.

The pushbutton switches are wired in active low mode. That is when you trigger a switch by pressing it down, it sends a logic low (0) signal to the microcontroller pin it is connected to. The signal stays low until the switch is released.

Since you are not expected to use interrupts at this point for interfacing the devices in this lab, you would need to keep holding the pushbutton until the CPU is able to finish whatever it was busy doing and gets to your button read function to detect your input. You also need to implement some logic for a software based switch debouncing. To notify the user that you have detected the button pressed, you need to turn off all LEDs while you are inside a loop waiting for the user to release the button. When all the LEDs are off the user knows that the input is detected and can then release the button.

The schematic diagram for the LED driver and the pushbutton circuits are given in Figure 4. As you can see in the schematic diagram, 2-to-4 decoders are used to control which of the four LEDs to turn on for each group of LEDs. The decoder inputs are controlled by the PIC32 GPIO pins as follows:

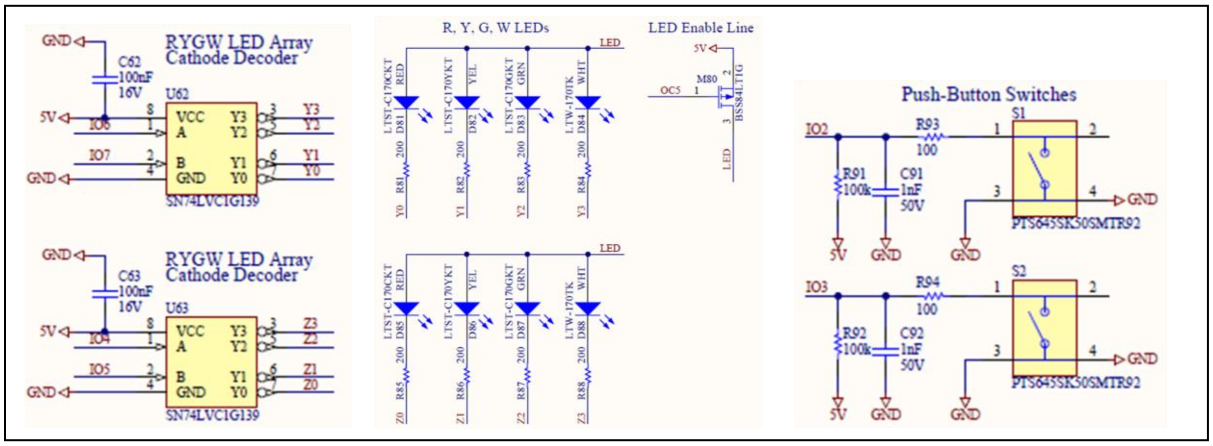


Figure 4: Schematic diagram of the cathode decoder circuits, the RYGW LEDs, and the pushbutton switches.

- IO6 & IO7 for the top decoder (that selects among Y0, Y1, Y2, Y3). Y0, Y1, Y2, Y3 correspond to the signals that drive the first (vertical) group of Red, Yellow, Green, and White LEDs, respectively. These signals are active-low and directly drive the LEDs.
- IO4 & IO5 for the bottom decoder (that selects among Z0, Z1, Z2, Z3). Z0, Z1, Z2, Z3 correspond to the signals that drive the second (horizontal) group of the Red, Yellow, Green, and White LEDs, respectively. These are active-low and directly drive the LEDs.

The two pushbuttons SW1 & SW2 are connected to the chipKIT pins IO2 & IO3, respectively.

The Anode input to all the eight LEDs is controlled by the MOSFET switch (LED Enable Line), whose gate input in turn is controlled by pin 4 of PORTD, which is RD4, and is shared with OC5. It is on IO pin 10 of the chipKIT Uno32 / uC32 board. To enable the power to the LEDs, drive this pin to low (0). You can disable all the eight LEDs by driving this pin to high (1).

Here is a summary of the pins that are used by the devices in this lab assignment:

Peripheral device	PIC32 Port	Pin #
SW1	PORTD	8
SW2	PORTD	0
LED Enable	PORTD	4
Decoder 1 bit 0	PORTD	2
Decoder 1 bit 1	PORTD	9
Decoder 2 bit 0	PORTF	1
Decoder 2 bit 1	PORTD	1

A header file (`TLights.h`) that defines the PIC32 I/O pins for the device interfacing of this project is provided on Blackboard for your use. There is also an incomplete C program file (`TLights.c`) that you could use as a template for creating your code that is necessary to drive the lights. To put all the program pieces together, you also need to write (complete) the `main.c` program file that implements the high-level functionality for the complete system.

An executable file (in ".hex" file format) is also provided on Blackboard. It will show you the expected behavior of the traffic light emulator circuit. To test it on your board first download the file ("TLights.X.production.hex") from Blackboard and then import it into an MPLABX project. To import, follow the command: "**File -> Import -> Hex/ELF … (Prebuilt) file**", then select the **Prebuilt Filename** by browsing your file system and choosing the ".hex" file that you downloaded. Select the correct MCU device and Hardware tool (**Licensed Debugger … chipKIT Programmer**) before proceeding. Then in the final step, choose your project location and then click finish.

Once you have the executable ".hex" imported you can use it to program the chip by clicking the "**Make and Program Device**" command button that has a symbol of a downward pointing green arrow on a chip. After the successfully flushing of the board, the program starts running right away. You can observe the behavior of the lights and interact with the application using the two pushbuttons to change the operation mode of the traffic lights.

Notes:

- When you create your project remember to select the correct device:
    - PIC32MX320F128H, for the chipKIT UNO32
    - PIC32MX340F512H, for the chipKIT uC32
- Use the "File -> Project Properties" menu to choose the "Licensed Debugger" as your debugger from the Hardware Tools option, and the XC32 from the Compiler Toolchains.
- If you get error messages when MPLABX tries to connect to the ChipKit Debugger, or during the programming or debugging process, try to reset the hardware (by unplugging the USB cable from your PC and plugging it back). If you keep getting errors, you can try to exit MPLAX, after saving your project, and restart it fresh.

Check-off and report:

Demonstrate your working program to the lab instructor and submit, via Blackboard, a lab report that follows the format and structure guidelines given in "Laboratory and Project Report Guidelines" available on Blackboard. Your report needs to include: Title page, Table of Contents, Objectives, Hardware, Program Source Code, Disassembly Listing, and Conclusions.