Unit 1: Microprocessor I/O

Revised March 9, 2017
This manual applies to Unit 1.

1 Introduction

The goal for this series of laboratory designs is to teach not only how to manage the PIC32 processor to solve engineering design problems, but also how to produce dependable (validated) and sustainable (verified) software using best design practices. To that end, Unit 1 will begin by introducing engineering concepts that are important in developing these best practices. Although some of these concepts may seem trivial at first, they are important in measuring the ability of a system to meet design requirements and specifications, and will be implemented throughout Units 1-7.

2 Objectives

1. How to develop a plan for a software-based microprocessor design.
2. Understand the management of basic microprocessor digital inputs and outputs.
3. Write an application that utilizes the Basys MX3 processor platform to perform basic calculator functions and display the results on a 4-digit 7-segment LED display.
4. Become familiar with the circuits included on the Basys MX3 PCB.

3 Basic Knowledge

1. Fundamentals of digital combinational and sequential logic.
2. How to interpret a schematic diagram and electric circuits.
3. How to write a computer program using the C language.
4. How to launch a Microchip MPLAB X project.

4 Equipment List

4.1 Hardware

1. Basys MX3 trainer board
2. Micro USB cable
3. Workstation computer running Windows 10 or higher, MAC OS, or Linux

In addition, we suggest the following instruments:
4. **Digilent Analog Discovery 2**

### 4.2 Software

The following programs must be installed on your development workstation:

1. **Microchip MPLAB X® v3.35 or higher**
   a. Installation help
   b. Getting started
2. **XC32 Cross Compiler**
3. **PLIB Peripheral Library**

### 5 Project Takeaways

1. Know how to generate a microprocessor development project using MPLAB X.
2. Know how to generate a config_bits file for the PIC32MX370 processor.
3. Know how to configure the Microchip PIC32 processor pins as either digital inputs or outputs.
4. Understand the drive capability of a processor digital output.
5. Know how to write a C program for a specific application using an embedded system.
6. Know how to make speed performance measure on a processor-based system.

### 6 Fundamental Concepts

#### 6.1 Project Planning

A software-based system consists of a collection of electrically interconnected electronic hardware components, some of which require programming using a computer language. **Software-based** system design is an engineering effort and requires a good process to obtain good results. The **life cycle** of system development uses a series of activities that, when considered in proper order, minimize design effort and achieve maximum benefit using the available resources of the software-based system.

Coding is one of the last phases when developing software-based systems. There are many methods and tools available today to assist in code development planning. Not all software applications require exotic planning tools, although all applications do require some degree of planning. After my 25 years of teaching a course on microcontrollers, the most common mistake I have seen students make is writing code before they completely understand the problem. Even a simple plan helps to guide the developer to design a system that meets the stated requirements.

Table 6.1 provides an outline of minimal tasks to complete the design process as is commonly done in industry. The first step is to have a complete understanding of the requirements or specifications. In other words, state in common human language what it is that the final design will do. Although the life cycle process appears to be a single pass, the final phase often results in cycling through the entire process numerous times to ensure that the delivered system meets the customer’s expectations.
Table 6.1. Life cycle of a software-based system design plan.¹

| Phase     | Activity                                             | Output                                                        | Measure                                      |
|-----------|------------------------------------------------------|                                                              |                                             |
| Concept   | Understanding the purpose, use, or objectives        | Problem statement                                            | 0 = Not adequate 10 = very precise          |
| Requirements | Define specific characteristics and capabilities – what the design will do. | Validated detailed requirements and specifications          | 0 = Incomplete 10 = Complete                |
| Design    | Describe how the requirements will be met and compliance be measured. | A series of concept, circuit diagrams, plans identifying required resources. | 0 = Incomplete 10 = Complete                |
| Construction | Build hardware components and write software code | A software based system                                      | 0 = Incomplete 10 = Complete                |
| Test      | Verify system meets requirements                     | Test compliance report                                        | Rating on compliance                         |
| Evaluation | Evaluation of reliability and functionality          | Recommendations for improvements                              | Cost – benefit analysis                      |

In all lab exercises of this series, the purposes and requirements will be defined by a problem statement. In most labs, the required hardware is provided on the Basys MX3 trainer board. In some instances, additional hardware circuits will need to be constructed and connected to the trainer board.

Software development will be the focus of the design phase for this series of labs. A significant amount of effort will be needed to understand the required resources of the PIC32MX370 processor along with the functionality of sensors, actuators, displays, and controls used in each lab, as well as the interconnection of these I/O devices to the PICMX370 processor. This includes application specific hardware, interconnection drawings, development tools, and testing instrumentation. Part of processor resource allocation is a table that designates the processor I/O pins and special functions such as timers, communications, and analog I/O. During the design phase, the developer must consider how the system will be tested or validated. Frequently, this requires identifying hardware instrumentation test points and test code that must be integrated with the system code.

6.1.1 Concepts Planning Using Graphical Representations

Concept maps are used as a mechanism for describing design plans. Figures 6.1 and 6.2 represent two common concept maps used in software engineering. Figure 6.1 is a data flow diagram (DFD) that describes the system as a collection of elements that transform inputs and outputs. The DFD is very useful in system partitioning to help identify minimum dependency and interaction between portioned elements.

6.1.2 Concepts Planning Using Partitioning

Software and hardware engineering use the divide and conquer approach that divides a large problem into a series of small tasks that are easier to complete. For software engineering, this requires partitioning the problem into single-purpose functions that can be tested independently of other single-purpose functions. There are two common approaches to partitioning: those based on hardware such as I/O devices or processor peripherals and partitions based on software functionality so that minimum interaction with or dependency on other partitions is necessary.

A control flow diagram (CFD), as the one shown in Fig. 6.2, describes the order in which the transactions occur. The CFD is useful when integrating the portioned elements in defining the order in which they are executed. One should be able to determine where each line of software code is executed on the CFD graph. The CFD is also useful during the testing phase to ensure that a scenario can be generated that causes the execution to transverse each path through the CFD.

### 6.1.3 Concept Planning: Performance Evaluation – When and How to Test

Using the divide and conquer approach, each element in the partitioned project must be completely tested and validated before integrating it with another element. You don’t want to design new code based on the functionality of incorrectly functioning old code. It is a waste of time.

After integration of all partitioned elements, system tests determine how well your design effort satisfies the stated requirements. One of the most straightforward ways is to make a table of the specified requirements to serve as a check-off list to include in your final documentation.

### 7 Problem Statement

There are two lab assignments associated with this unit. Lab 1a is very limited in scope and complexity and focuses on defining the PIC32MX370 I/O pins as digital inputs or outputs in a tutorial-like fashion. Lab 1b is more open-ended and asks the student to extend the knowledge gained in Lab 1a by using the slide switches, push buttons, and the 4-digit 7-segment LED display on the Basys MX3 trainer board to implement a basic calculator. Both labs emphasize the engineering approach to embedded system design.
8 Background Information

8.1 The Development Environment

The reader is expected to have acquired the basic knowledge of programming a computer in C. But as a review, we offer web links to an online C Tutorial and Microchip MPLAB X IDE. Figure 8.1 illustrates a typical hardware configuration for microcontroller development. The workstation computer can be running any Windows®, MAC®, or Linux® operating system. The developer writes the program code on this workstation using the editor provided by the MPLAB X IDE or a third party editor such as NotePad++. A cross compiler converts the C programs into a binary representation of the code that the PIC processor can execute. The file generated by the compiler is downloaded to the target application processor with the aid of a programmer/debugger module (included on-board the Basys MX3).

![Figure 8.1. Integrated Development Hardware Diagram.](image)

The diagram in Fig. 8.1 shows the Basys MX3 unit. The Basys MX3 has a built-in programmer/debugger that allows a direct connection from the development workstation computer to the Basys MX3 board. If the programmer/debugger is not built into the hardware platform, a separate programmer is required, such as the PICkit 3 or a chipKIT Programmer.

8.2 General Notes of Interest

1. For optimal processor performance, include the instruction `SYSTEMConfig(SystemClock, SYS_CFG_WAIT_STATES | SYS_CFG_PCACHE);` in the initialization portion of the main function. The parameter, `SystemClock`, is set to 80,000,000 by the “config_bits” file.

2. In order to access all pins on PORT A as digital I/O, include the following statement in the initialization portion of the main function. `DDPCONbits.JTAGEN = 0;`
3. If you are using “plib.h” functions, XC32 rev. 1.4 will generate numerous warnings concerning the obsolescence of the plib functions. These warnings can be suppressed by adding the following definitions to the top of the main.c file.

```c
#ifndef _SUPPRESS_PLIB_WARNING
#define _SUPPRESS_PLIB_WARNING
#endif

#ifndef _DISABLE_OPENADC10_CONFIGPORT_WARNING
#define _DISABLE_OPENADC10_CONFIGPORT_WARNING
#endif
```

4. Table 8.1 below, reproduced from the [MPLAB XC32 C/C++ Compilers User’s Guide](https://www.digilentinc.com/resources/user-guides/mplab-xc32-c-c-compiler-user-guide), notes that the legacy library option is automatically selected for new projects. If using MPLABX 3.40 or later, when starting a new project, this option must **not** be checked, as shown in Fig. 8.2.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Command Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use legacy lib</td>
<td>Check to use the Standard C library in the format before XC32 v1.12. Uncheck to use the HTC LIBC version. The legacy libc is the default for new projects created in MPLAB X v3.15 and later.</td>
<td>-legacy-libc &amp; -no-legacy-libc</td>
</tr>
</tbody>
</table>

![Figure 8.2. Disable "Use Legacy libc" to use plib.h.](image-url)
5. Only the 8 individual LEDs labeled LD0 through LD7 are connected to contiguous I/O port pins (see Table E.1 for Port A pins 0 through 7). These are the only I/O pins that may be set as a group. All other pins must be individually assigned. The dispersion of pins and ports for common elements such as switches, buttons, and segment LEDs requires a bit-banging approach rather than reading or setting a group of bits with a single instruction.

8.3 Microcontroller Resources

Microcontrollers have three major resources: time access to the central processing unit (CPU execution time), program and data memory, and input and output (I/O). Frequently, an application requires more of one of these three resources than the selected processor can provide. Microprocessor resource management involves compensating for a lack of one of these resources using excess capability in one or both of the other two. For example, if an application requires more I/O than the processor has I/O pins, the designer can implement a parallel pin multiplexing scheme that reduces the number of I/O pins but requires additional memory and CPU time to execute the code. This concept will be explored when we interface the PIC32 processor with a 4-digit 7-segment LED display.

9 References

4. PIC32MX330/350/370/430/450/470 Family Data Sheet
Appendix A: Starting a New MPLAB X Project

The following three steps are used to launch a new project. New MPLAB X users are also referred to Reference 6 above.

1. Generate Lab 1 Project:
   a. Open the MPLAB X application on your workstation computer.
   b. Start a new Microchip Embedded Standalone Project.
   c. Select Device: PIC32MX370F512L
   d. Select Tool: Licensed Debugger MCU Alpha One
   e. Select Toolchain: XC32 compiler
   f. Select Project Name and Folder:
      i. Project Name: LABx
      ii. Project Location: to be determined
   g. Click on “Finish”

2. Generate a config_bits header file
   a. Right-click on the Header Files and select “New” -> “XC32 Header.h”. Name this file “config_bits.h”.
   b. Click on “Windows” -> “PIC Memory Views” -> “Configuration Bits”.
   c. Set the options as shown in Listing C.2 in the Appendix.
   d. Click on “Generate Source Code to Output”.
   e. Enter “CTRL A” to highlight the text in the output window and “CTRL C” to copy the text.
   f. Double-click on “config_bits” under the Header Files tab to open that file in the editor window. Delete lines 21 through 173 of the automatically generated code.
   g. Change the three instances of the text “_EXAMPLE_FILE_NAME_H” to “_CONFIG_BITS_H”.
   h. Position the cursor to the blank line – line 21.
   i. Enter “CRTL V” to paste the config_bits text to the config_bits.h file. Enter “CTR_S” to save the newly generated file to the project folder.
   j. Make the appropriate edits to the automatically generated header comments.
   k. Once a config_bits.h file is generated, it can be copied to other projects without needing to complete steps a through j.

3. Generate a main function: All projects must contain a function called “main”.
   a. Right-click on Source Files and select “New” -> “C Main File..”. Name this file “main.” and the extension set to “c”, so it will be “main.c”.
   b. The main.c file must contain the following elements:
      i. #include “config_bits.h”. Other header files may be required.
      ii. Declare global variables.
      iii. Declare local functions other than the function main.
      iv. Add a main function code as shown in Listing A.1. The return denotes an EXIT FAILURE because embedded systems code should never exit from the function main because it inherently has no place to return to.

Listing A.1. Format of a typical main function.

```c
int main(int argc, char** argv)
{
    // Initialization code goes here

    while(1)
    {
```
// User code in infinite loop
}
return (EXIT_FAILURE);  // Failure if code executes
               // this line of code.
} // End of main.c

i. Complete the project requirements by adding header and source files as elements of building blocks. Each source file should have a header file that declares constants and software function prototypes that are used as interfaces with other project files. The project work is divided into independently testable units that can serve as libraries for other projects that use the same hardware resources. For example, writing a decimal number to a 4-digit 7-segment display is a common operation. Hence, a set of source-header files will be written to display a 4-digit number.
Appendix B: Configuration Settings for the PIC32MX370F512L

Listing B.1. Configuration bits Settings for Basys MX3 Processor Board

/* ************************************************************************** */
/** Descriptive File Name

Company: DigilentInc

File Name: config_bits.h

Summary: Set configuration bits for the PIC32MX370F512L processor.

Description: The configurations settings set the processor to have a core frequency of 80MHz and a peripheral clock speed of 10MHz.
*/
/* ************************************************************************** */

#ifndef _CONFIG_BIYS_H_
#define _CONFIG_BIYS_H_

// PIC32MX370F512L Configuration Bit Settings

// 'C' source line config statements

#include <xc.h>

// DEVCFG3
// USERID = No Setting
#pragma config FSRSSSEL = PRIORITY_7 // Shadow Register Set Priority Select
// (SRS Priority 7)
#pragma config PMDL1WAY = OFF // Peripheral Module Disable
// Configuration (Allow multiple
// reconfigurations)
#pragma config IOL1WAY = OFF // Peripheral Pin Select Configuration
// (Allow multiple reconfigurations)

// DEVCFG2
#pragma config FPLLIDIV = DIV_2 // PLL Input Divider (2x Divider)
#pragma config FPLLMUL = MUL_20 // PLL Multiplier (20x Multiplier)
#pragma config FPLL0DIV = DIV_1 // System PLL Output Clock Divider
// (PLL Divide by 1)

// DEVCFG1
#pragma config FNOSC = PRIPLL // Oscillator Selection Bits (Primary
// Osc w/PLL (XT+,HS+,EC+PLL))
#pragma config FSOSCCEN = OFF // Secondary Oscillator Enable
// (Disabled)
#pragma config IESO = OFF // Internal/External Switch Over
// (Disabled)
#pragma config POSCMOD = XT // Primary Oscillator Configuration
// (XT osc mode)
#pragma config OSCIOFNC = OFF // CLKO Output Signal Active on the
// OSCO Pin (Disabled)
#pragma config FPBDIV = DIV_8 // Peripheral Clock Divisor (Pb_Clk is
// Sys_Clk/8)
#pragma config FCKSM = CSDCMD // Clock Switching and Monitor Selection

*/
#pragma config WDTPS = PS1048576    // Watchdog Timer Postscaler (1:1048576)
#pragma config WINDIS = OFF         // Watchdog Timer Window Enable
// (Watchdog Timer in Non-Window Mode)
#pragma config FWDTEN = OFF          // Watchdog Timer Enable (WDT Disabled
// (SWDTEN Bit Controls))
#pragma config FWDTWINSZ = WINSZ_25 // Watchdog Timer Window Size (Window
// Size is 25%)

// DEVCFG0
#pragma config DEBUG = OFF           // Background Debugger Enable (Debugger
// is Disabled)
#pragma config JTAGEN = OFF          // JTAG Enable (JTAG Disabled)
#pragma config ICESEL = ICS_PGx1     // ICE/ICD Comm Channel Select
// (Communicate on PGEC1/PGED1)
#pragma config PWP = OFF             // Program Flash Write Protect (Disable)
#pragma config BWP = OFF             // Boot Flash Write Protect bit
//(Protection Disabled)
#pragma config CP = OFF              // Code Protect (Protection Disabled)

#endif /* _CONFIG_BIYS_H_ */

/* *************************************************** End of File ***************************************************/
Appendix C: Basys MX3 Hardware Initialization

Listings C.1 and C.2 that are provided below initialize the Basys MX3 hardware inputs. Development time will be reduced by including these two files in future projects. The hardware.h header file contains an extensive use of macro instructions. Macro instructions are text replacement compiler directives that can result in better documentation and all but eliminate magic numbers, which are a well-known programming bad practice. Macro instructions can obfuscate code and can make it difficult when debugging; however, when properly used, macro instructions can improve documentation and facilitate code portability. The macro instructions used in the hardware.h header file are simple definitions, conditional compilation, and ternary operations all designed to improve readability of the code.

Listing C.1. Hardware Initialization Header File

```c
/* ************************************************************************** */
/** Descriptive File Name
 * @ Author
 *     Richard Wall
 * @ Date
 *     April 30, 2016
 */

@ Revised
    December 10, 2016

@Company
    Digilent Inc.

@File Name
    hardware.h

@Summary
    Definition of constants and macro routines for the Basys MX3 processor board

@Description
    The #define statements and macro C code provide high level access to the Basys MX3 trainer boards switches, push buttons, and LEDs.

    /* Conditional inclusion prevents multiple definition errors */
    ifndef _HARDWARE_H_
        #define _HARDWARE_H_

        ifndef _SUPPRESS_PLIB_WARNING /* Suppress plib obsolesce warnings */
            #define _SUPPRESS_PLIB_WARNING
        endif

        ifndef _DISABLE_OPENADC10_CONFIGPORT_WARNING
            #define _DISABLE_OPENADC10_CONFIGPORT_WARNING
        endif

        /* Comment out the following define statement when programmer is NOT used to allow using BTNL and BTNU as user inputs. */
        ifndef DEBUG_MODE
            #define DEBUG_MODE /* Inputs from push buttons BTNL and BTNU are not useable */
```
/* This included file provides access to the peripheral library functions and
must be installed after the XC32 compiler. See

#include <plib.h>

/* The following definitions are for IO assigned on the Digilent Basys MX3
processor board. */

/* The ANSELx register has a default value of 0xFFFF; therefore, all pins that
* share analog functions are analog (not digital) by default. All pins are
* initially set be digital followed be setting A_POT for the ANALOG INPUT
* CONTROL and A_MIC for the microphone input back to being analog input pins.*/
define ALL_DIGITAL_IO() (ANSELA=0,ANSELB=0,ANSELC=0,ANSELD=0,ANSELF=0,ANSELF=0)
define SET_MIC_ANALOG() ANSELBbits.ANSB4 = 1
#define SET_POT_ANALOG() ANSELBbits.ANSB2 = 1

/* Macros to configure PIC pins as inputs to sense switch settings */
/* BIT definitions are defined in port.h which is provided with the plib
Library. */
define SW0_bit BIT_3 /* RF3 - 1<<3 */
define SW1_bit BIT_5 /* RF5 - 1<<5 */
define SW2_bit BIT_4 /* RF4 - 1<<4 */
define SW3_bit BIT_15 /* RD15 - 1<<15 */
define SW4_bit BIT_14 /* RD14 - 1<<14 */
define SW5_bit BIT_11 /* RB11 - 1<<11 */
define SW6_bit BIT_10 /* RB10 - 1<<10 */
define SW7_bit BIT_9 /* RB9 - 1<<9 */

/* The following macro instructions set switches as inputs. */
define Set_SW0_in() TRISFbits.TRISF3 = 1
define Set_SW1_in() TRISFbits.TRISF5 = 1
define Set_SW2_in() TRISFbits.TRISF4 = 1
define Set_SW3_in() TRISDbits.TRISD15 = 1
define Set_SW4_in() TRISDbits.TRISD14 = 1
define Set_SW5_in() TRISBbits.TRISB11 = 1
define Set_SW6_in() TRISBbits.TRISB10 = 1
define Set_SW7_in() TRISBbits.TRISB9 = 1

/* The following macro instruction sets the processor pins for all 8 switch inputs */
define Set_All_Switches_In(); ( Set_SW0_in(); Set_SW1_in(); Set_SW2_in(); \
Set_SW3_in(); Set_SW4_in(); Set_SW5_in(); \
Set_SW6_in(); Set_SW7_in(); )

/* The following macro instructions provide for reading the position of the 8 switches. */
define SW0() PORTFbits.RF3
define SW1() PORTFbits.RF5
define SW2() PORTFbits.RF4
define SW3() PORTDbits.RD15
define SW4() PORTDbits.RD14
define SW5() PORTBbits.RB11
define SW6() PORTBbits.RB10
define SW7() PORTBbits.RB9

/* Organize the SW bits into a unsigned integer */

/* Macro instructions to define the bit values for push button sensors */
define BTN1_bit BIT_0 /* RB0 1 << 0 */
define BTN2_bit BIT_8 /* RB8 1 << 8 */
define BTN3_bit BIT_1 /* RB1 1 << 1 */
define BTN4_bit BIT_15 /* RA15 1 << 15 */
/* See http://umassamherstm5.org/tech-tutorials/pic32-tutorials/pic32mx220-tutorials/internal-pull-up-down-resistors */
/* Macro instructions to set the push buttons as inputs */
#define Set_BTNL_in() (TRISBbits.TRISB0 = 1, CNPDBbits.CNPDB0 = 1)
#define Set_BTNR_in() TRISBbits.TRISB8 = 1
#define Set_BTNR_out() TRISBbits.TRISB8 = 0
#define Set_BTNU_in() (TRISBbits.TRISB1 = 1, CNPDBbits.CNPDB1 = 1)
#define Set_BTNU_out() TRISBbits.TRISB1 = 0
#define Set_BTND_in() TRISAbits.TRISA15 = 1
#define Set_BTND_out() TRISAbits.TRISA15 = 0
#define Set_BTNC_in() TRISFbits.TRISF0 = 1
#define Set_BNTL_in() (TRISBbits.TRISB0 = 1, CNPDBbits.CNPDB0 = 1)
#define Set_BNTU_in() (TRISBbits.TRISB1 = 1, CNPDBbits.CNPDB1 = 1)
#define Set_BNTD_in() TRISAbits.TRISA15 = 1
#define Set_BNTD_out() TRISAbits.TRISA15 = 0
#define Set_BNTC_in() TRISFbits.TRISF0 = 1
#define Set_All_PBs_Input() (SetBTN_in(), Set_BTNR_in(), Set_BTNU_in(), Set_BTND_in(), Set_BTNC_in())
#ifdef DEBUG_MODE
#define Set_All_PBs_Input() (Set_BTNR_in(), Set_BTND_in(), Set_BTNC_in())
#endif
/* Macro instructions to read the button position values. 1 = button pressed */
/* Include BNTL and BTNU only if NOT in debug mode */
#ifdef DEBUG_MODE
#define BNTL() PORTBbits.RB0
#define BNTU() PORTBbits.RB1
#define BNTR() PORTBbits.RB8
#define BNTD() PORTAbits.RA15
#define BNTC() PORTFbits.RF0
#endif
#define LED0_bit BIT_0 /* RA0 */
#define LED1_bit BIT_1 /* RA1 */
#define LED2_bit BIT_2 /* RA2 */
#define LED3_bit BIT_3 /* RA3 */
#define LED4_bit BIT_4 /* RA4 */
#define LED5_bit BIT_5 /* RA5 */
#define LED6_bit BIT_6 /* RA6 */
#define LED7_bit BIT_7 /* RA7 */
#define All_LED_bits 0xff /* Set all LEDs off
#define Set_LED0_out() TRISAbits.TRISA0 = 0
#define Set_LED1_out() TRISAbits.TRISA1 = 0
#define Set_LED2_out() TRISAbits.TRISA2 = 0
#define Set_LED3_out() TRISAbits.TRISA3 = 0
#define Set_LED4_out() TRISAbits.TRISA4 = 0
#define Set_LED5_out() TRISAbits.TRISA5 = 0
#define Set_LED6_out() TRISAbits.TRISA6 = 0
#define Set_LED7_out() TRISAbits.TRISA7 = 0
#define Set_All_LEDs_Output() TRISACLR = All_LED_bits
#define setLED0(a) { if(a) LATASET = LED0_bit; else LATACLR = LED0_bit; }
#define setLED1(a) { if(a) LATASET = LED1_bit; else LATACLR = LED1_bit; }
#define setLED2(a) { if(a) LATASET = LED2_bit; else LATACLR = LED2_bit; }
#define setLED3(a) { if(a) LATASET = LED3_bit; else LATACLR = LED3_bit; }
#define setLED4(a) { if(a) LATASET = LED4_bit; else LATACLR = LED4_bit; }
#define setLED5(a) { if(a) LATASET = LED5_bit; else LATACLR = LED5_bit; }
#define setLED6(a) { if(a) LATASET = LED6_bit; else LATACLR = LED6_bit; }
#define setLED7(a) { if(a) LATASET = LED7_bit; else LATACLR = LED7_bit; }
#define setLED7(a) {if(a) LATASET = LED7_bit; else LATACLR = LED7_bit;}
#define Set_All_LEDs_On() LATASET = All_LED_bits // Set all LEDs on
#define Set_All_LEDs_Off() LATACLR = All_LED_bits // Set all LEDs off

/* Macros to invert the output to the board LEDs */
#define invLED0() LATAINV = LED0_bit
#define invLED1() LATAINV = LED1_bit
#define invLED2() LATAINV = LED2_bit
#define invLED3() LATAINV = LED3_bit
#define invLED4() LATAINV = LED4_bit
#define invLED5() LATAINV = LED5_bit
#define invLED6() LATAINV = LED6_bit
#define invLED7() LATAINV = LED7_bit

/* Based upon setting in config_bits.h These directly influence timed
 * events using the Tick module. They also are used for UART I2C, and SPI
 * baud rate generation. */
#define XTAL (80000000UL) /* 8 MHz Xtal on Basys MX3 */
#define GetSystemClock() (80000000UL) /* Instruction frequency */
#define SYSTEM_FREQ (GetSystemClock())
#define GetCoreClock() (GetSystemClock()/2) /* Core clock frequency */
#define GetPeripheralClock() (GetSystemClock()/8) /* PCLK set for 10 MHz */

/* Used in core timer software delay */
#define CORE_MS_TICK_RATE (unsigned int) (GetCoreClock()/1000UL)

#endif /* End of _HARDWARE_H */

/* Declare Hardware setup for global access */
void Hardware_Setup(void);
unsigned int switch2Binary(void);

Listing C.2. Hardware Configurations for the Basys MX3 Trainer Board

/* **************************************************************************
 * Descriptive File Name
 * @ Author
 * Richard Wall
 * @ Date
 * April 30, 2016
 *
 * @ Revised
 * December 10, 2016
 *
 * @Company
 * Digilent
 *
 * @File Name
 * PICmx370.c
 *
 * @Summary
 * Definition of constants and macro routines
 *
 * @Description
 */
The #define statements and macro C code provide high level access to the "Trainer" boards switches, push buttons, and LEDs.

*/
/* **************************************************************************/

/* This included file provides access to the peripheral library functions and must be installed after the XC32 compiler. See

#include "hardware.h"
#include "switches.h"
#include <plib.h>

// ****************************************************************************

_UNSIGNED▸
@Function
void Hardware_Setup(void);

@Summary
Initializes PIC32 pins commonly used for IO on the Trainer processor board.

@Description
Initializes PIC32 digital IO pins to provide functionality for the switches, push buttons, and LEDs

@Precondition
"config_bits* must be included in the project

@Parameters
None

@Returns
None

@Remarks
* Returned error flag indicates the value of either x or y is out of
* range 0 through 15.
*/
void Hardware_Setup(void)
{
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* Statement configure cache, wait states and peripheral bus clock
* Configure the device for maximum performance but do not change the PBDIV
* Given the options, this function will change the flash wait states, RAM
* wait state and enable prefetch cache but will not change the PBDIV.
* The PBDIV value is already set via the pragma FPBDIV option above..
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
// Allow RA0, RA1, RA4 and RA5 to be used as digital IO
SYSTEMConfig(GetSystemClock(), SYS_CFG_WAIT_STATES | SYS_CFG_PCACHE);

ALL_DIGITAL_IO(); // Set all IO pins to digital */
SET_MIC_ANALOG();  // Set microphone input to analog */
SET_POT_ANALOG();   // Set ANALOG INPUT CONTROL pot input to analog */
Set_All_LEDS_Output(); // Configure all Basys MX3 LED0 ~ LED7 as outputs */
Set_All_LEDS_Off();  // Set all Basys MX3 LED0 ~ LED7 off */
Set_All_Switches_Input();  /* Configure all Basys MX3 slide switches as inputs */
Set_All_PBs_Input(); /* Configure all Basys MX3 push buttons as inputs */
}
} /* End of hardware_setup */
/**
@Function
unsigned int Switch2Binary( void )

@Summary
Generates an unsigned integer value from the switch settings using binary weighting. The Basys MX3 slide switches are initialized as inputs to a disparate port assignments. This function collects all the switch settings into a single variable.
*/
unsigned int switch2Binary(void)
{
  int value;

  value = (int) SW0() << 0;
  value += (int) SW1() << 1;
  value += (int) SW2() << 2;
  value += (int) SW3() << 3;
  value += (int) SW4() << 4;
  value += (int) SW5() << 5;
  value += (int) SW6() << 6;
  value += (int) SW7() << 7;
  return value;
}
Appendix D: Essential Elements of the PIC32 Microprocessor I/O^2

This section is not required to complete Labs 1a and 1b; however, it contains information important to forming a more complete view of microprocessor I/O.

Digital input and output is one of the most basic functions as a microprocessor can perform. A simplified block diagram of the control of an I/O processor pin is shown in Fig. D.1. The operation of the I/O pin is configured by bit values in the following four registers: the output drain control (ODC), the output latch (LAT), the output tri-state (TRIS), and, for pins that can be used for analog inputs, the analog-to-digital input configuration (AD1PCFG).

Minimally, the TRIS registers must be set to configure a processor I/O pin as an input or output by setting the bit in the register to 0 for output or 1 for input. By default, all pins that can serve as an analog input are configured to be so. Any analog input pin that is to be used for digital I/O must also be designated to be a digital I/O pin by clearing the appropriate analog select pin. The PIC32MX370 processor has five registers that are used to designate dual-functioning pins as digital or analog inputs. For example, the instruction to clear the analog select pin for PORT D bit 1 (RD1) is "ANSELDbits.ANSD1 = 0; ".

If the bit in the TRIS register is set low to make the I/O pin to function as an output, the voltage pin can be set high by writing a 1 to the appropriate bit in the LAT register. Setting the LAT register bit to a zero sets the output pin low (0V). The actual voltage at the output pin depends on the setting in the ODC register. The default configuration

---

^2 PIC32MX330/350/370/430/450/470 family data sheet, Chapter 12
has all bits in the ODC register set to zero, which means the output can both source (supply output current) and sink (pull outputs low) current. If the bit in the ODC register is set high, then the output pin functions as open drain, which can sink current but not source current.

The maximum current capability of each conventional I/O pin is 15mA (sink or source), while the combined current of all I/O pins is 200mA subject to total power constraints. I/O pins have both open drain and active source output capability. The open drain capability provided by the ODC register is useful when interfacing with switch array keypads (see Pmod KYPD for an example keypad device). The outputs that drive the four LEDs on the Basys MX3 board are limited to approximately 5mA.

The PORT register allows the state of pin to be read regardless if the pin is configured in the TRIS register to be an input or an output. If the voltage on the pin is above the high input threshold, the PORT bit is read as a logic one. Various processor pins have different thresholds.

If you are interested, refer to Table 30-8 of PIC32MX330/350/370/430/450/470 data sheet to determine the high and low thresholds for the pins being used. All digital input-only pins are 5V tolerant, meaning that a device that outputs 5V for logic high can be connected to digital input pins and will not damage the PIC32MX processor. For the PIC32MX7370, the maximum voltage that can be applied to I/O pins that can be used as either analog or digital I/O is \( V_{DD} + 0.3V \), or 3.6V. Exceeding this voltage limit will damage the processor.

The following listings are two functionally equivalent examples of code that uses the input state of pin RD0 to set the output on RD1.

### Listing D.1. Basic I/O Program

```c
#include "config_bits.h" // Configure PIC32MX370F512L
int main()
{
    // Initialization
    ANSELDbits.ANSD1 = 0; // RD1 set to digital I/O
    TRISDbits.TRISD1 = 0; // RD1 set to output
    TRISDbits.TRISD0 = 1; // RD2 set to input

    // loop
    while(1)
    {
        LATDbits.LATD1 = PORTDbits.RD0; // Copy state of RD0 to RD1
    }
}
```

### Listing D.2. Alternate Basic I/O Program

```c
#include "config_bits.h" // Configure PIC32MX370F512L
int main()
{
    // Initialization
    ANSELDCLR = 0x02; // RD1 set to digital I/O
    TRISDCLR = 0x02; // RD1 set to output
    TRISDSET = 0x01; // RD0 set to input

    // loop
    while(1)
    {
        if(PORTD & 0x01) // Read RD0 pin
            LATDSET = 0x02; // Set RD1 pin high
    }
}
else
    LATDCLR = 0x02;  // Set RD1 pin low

}
## Appendix E: PIC32MX370F512L Processor Pin Assignments for Basys MX3

Table E.1. Processor I/O Assignments.

<table>
<thead>
<tr>
<th>CPU pin</th>
<th>Port</th>
<th>ALT</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>RB4</td>
<td>AN4/C1INB/RB4</td>
<td>A_MIC Microphone</td>
</tr>
<tr>
<td>43</td>
<td>RB14</td>
<td>AN14/RPB14/PMA1/CTED5/RB14</td>
<td>A_OUT Speaker</td>
</tr>
<tr>
<td>23</td>
<td>RB2</td>
<td>PGE3/C2INB/RB2/CTED13/RB2</td>
<td>A_POT Pot</td>
</tr>
<tr>
<td>90</td>
<td>RG0</td>
<td>RPOG/RMD8/RG0</td>
<td>ACL_INT2 i2C - Accelerometer</td>
</tr>
<tr>
<td>22</td>
<td>RB3</td>
<td>PGED3/C2INA/RPB3/RB3</td>
<td>AIN1 Stepper motor</td>
</tr>
<tr>
<td>18</td>
<td>RE8</td>
<td>RPE8/RE8</td>
<td>AIN2 Stepper motor</td>
</tr>
<tr>
<td>41</td>
<td>RB12</td>
<td>AN12/PMA11/RB12</td>
<td>AN0 4 digit 7 segment LED</td>
</tr>
<tr>
<td>42</td>
<td>RB13</td>
<td>AN13/RB13</td>
<td>AN1 &quot;</td>
</tr>
<tr>
<td>28</td>
<td>RA9</td>
<td>VREF-/CVREF-/PMA7/RA9</td>
<td>AN2 &quot;</td>
</tr>
<tr>
<td>29</td>
<td>RA10</td>
<td>VREF+/CVREF+/PMA6/RA10</td>
<td>AN3 &quot;</td>
</tr>
<tr>
<td>19</td>
<td>RE9</td>
<td>RPE9/RE9</td>
<td>BIN1 Stepper motor</td>
</tr>
<tr>
<td>20</td>
<td>RB5</td>
<td>ANS/C1INA/RPB5/RB5</td>
<td>BIN2 Stepper motor</td>
</tr>
<tr>
<td>87</td>
<td>RF0</td>
<td>RPOF/PMD11/RF0</td>
<td>BTNC Push Button</td>
</tr>
<tr>
<td>67</td>
<td>RA15</td>
<td>RPA15/RA15</td>
<td>BTND/S1_PWM Push Button / servo mtr</td>
</tr>
<tr>
<td>32</td>
<td>RB8</td>
<td>AN8/RPB8/CTED10/RB8</td>
<td>BTNR/SP_PWM Push Button / servo mtr</td>
</tr>
<tr>
<td>96</td>
<td>RG12</td>
<td>TRD1/RG12</td>
<td>CA 4 digit 7 segment LED</td>
</tr>
<tr>
<td>66</td>
<td>RA14</td>
<td>RPA14/RA14</td>
<td>CB &quot;</td>
</tr>
<tr>
<td>83</td>
<td>RD6</td>
<td>RDD6/PMD14/RA6</td>
<td>CC &quot;</td>
</tr>
<tr>
<td>97</td>
<td>RG13</td>
<td>TRD0/RG13</td>
<td>CD &quot;</td>
</tr>
<tr>
<td>1</td>
<td>RG15</td>
<td>CNG15/RG15</td>
<td>CE &quot;</td>
</tr>
<tr>
<td>84</td>
<td>RD7</td>
<td>RPD7/PMD15/RD7</td>
<td>CF &quot;</td>
</tr>
<tr>
<td>80</td>
<td>RD13</td>
<td>RPD13/RD13</td>
<td>CG &quot;</td>
</tr>
<tr>
<td>95</td>
<td>RD14</td>
<td>TRD2/RD14</td>
<td>CP &quot;</td>
</tr>
<tr>
<td>93</td>
<td>RE0</td>
<td>PMDO/RE0</td>
<td>DB0 Character LCD data</td>
</tr>
<tr>
<td>94</td>
<td>RE1</td>
<td>PMD1/RE1</td>
<td>DB1 &quot;</td>
</tr>
<tr>
<td>98</td>
<td>RE2</td>
<td>AN20/PMD2/RE2</td>
<td>DB2 &quot;</td>
</tr>
<tr>
<td>99</td>
<td>RE3</td>
<td>RPE3/PMD3/RE3</td>
<td>DB3 &quot;</td>
</tr>
<tr>
<td>100</td>
<td>RE4</td>
<td>AN21/PMD4/RE4</td>
<td>DB4 &quot;</td>
</tr>
<tr>
<td>3</td>
<td>RE5</td>
<td>AN22/RPE5/PMD5/RE5</td>
<td>DB5 &quot;</td>
</tr>
<tr>
<td>4</td>
<td>RE6</td>
<td>AN23/PMD6/RE6</td>
<td>DB6 &quot;</td>
</tr>
<tr>
<td>5</td>
<td>RE7</td>
<td>AN27/PMD7/RE7</td>
<td>DB7 &quot;</td>
</tr>
<tr>
<td>81</td>
<td>RD4</td>
<td>RPD4/PWR/RD4</td>
<td>DISP_EN Character LCD ctrl</td>
</tr>
<tr>
<td>82</td>
<td>RDS</td>
<td>RPD5/PMS/RS/RD5</td>
<td>DISP_R/W &quot;</td>
</tr>
<tr>
<td>44</td>
<td>RB15</td>
<td>AN15/RPB15/PMA0/CTED6/RB15</td>
<td>DISP_RS &quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>89</td>
<td>RG1</td>
<td>RPG1/PMD9/RG1</td>
<td>IR_PDOWN</td>
</tr>
<tr>
<td>26</td>
<td>RB6</td>
<td>PGE2/CAN/CAN/RB6</td>
<td>IR_RX</td>
</tr>
<tr>
<td>27</td>
<td>RB7</td>
<td>PGE2/CAN/CAN/RB7</td>
<td>IR_TX</td>
</tr>
<tr>
<td>7</td>
<td>RC2</td>
<td>RPC2/RPC2</td>
<td>JA1</td>
</tr>
<tr>
<td>6</td>
<td>RC1</td>
<td>RPC1/RPC1</td>
<td>JA2</td>
</tr>
<tr>
<td>9</td>
<td>RC4</td>
<td>RPC4/CTED3/RC4</td>
<td>JA3</td>
</tr>
<tr>
<td>10</td>
<td>RG6</td>
<td>AN16/C1N/C1N/RG6</td>
<td>JA4</td>
</tr>
<tr>
<td>8</td>
<td>RC3</td>
<td>RPC3/RC3</td>
<td>JA7</td>
</tr>
<tr>
<td>11</td>
<td>RG7</td>
<td>AN17/C1N/C1N/RG7</td>
<td>JA8</td>
</tr>
<tr>
<td>12</td>
<td>RG8</td>
<td>AN18/C2N/C2N/RG8</td>
<td>JA9</td>
</tr>
<tr>
<td>14</td>
<td>RG9</td>
<td>AN19/C2N/C2N/RG9</td>
<td>JA10</td>
</tr>
<tr>
<td>69</td>
<td>RD9</td>
<td>RPD9/RD9</td>
<td>JB1</td>
</tr>
<tr>
<td>71</td>
<td>RD11</td>
<td>RPD11/PMCS1/RD11</td>
<td>JB2</td>
</tr>
<tr>
<td>70</td>
<td>RD10</td>
<td>RPD10/PMCS2/RD10</td>
<td>JB3</td>
</tr>
<tr>
<td>68</td>
<td>RD8</td>
<td>RPD8/RTCC/RD8</td>
<td>JB4</td>
</tr>
<tr>
<td>74</td>
<td>RC14</td>
<td>SOSC1/PC14/T1CK/RC14</td>
<td>JB7</td>
</tr>
<tr>
<td>72</td>
<td>RD0</td>
<td>RPD0/RD0</td>
<td>JB8</td>
</tr>
<tr>
<td>76</td>
<td>RD1</td>
<td>AN24/RD1/RD1</td>
<td>JB9</td>
</tr>
<tr>
<td>73</td>
<td>RC13</td>
<td>SOSC/PC13/RC13</td>
<td>JB10</td>
</tr>
<tr>
<td>17</td>
<td>RA0</td>
<td>TMS/CTED1/RA0</td>
<td>LED0</td>
</tr>
<tr>
<td>38</td>
<td>RA1</td>
<td>TCK/CTED2/RA1</td>
<td>LED1</td>
</tr>
<tr>
<td>58</td>
<td>RA2</td>
<td>SCL2/RA2</td>
<td>LED2</td>
</tr>
<tr>
<td>59</td>
<td>RA3</td>
<td>SDA2/RA3</td>
<td>LED3</td>
</tr>
<tr>
<td>60</td>
<td>RA4</td>
<td>TDI/CTED9/RA4</td>
<td>LED4</td>
</tr>
<tr>
<td>61</td>
<td>RA5</td>
<td>TDO/RA5</td>
<td>LED5</td>
</tr>
<tr>
<td>91</td>
<td>RA6</td>
<td>TRCLK/RA6</td>
<td>LED6</td>
</tr>
<tr>
<td>92</td>
<td>RA7</td>
<td>TD3/CTED8/RA7</td>
<td>LED7</td>
</tr>
<tr>
<td>78</td>
<td>RD3</td>
<td>AN26/RPD3/RD3</td>
<td>LED8_B</td>
</tr>
<tr>
<td>79</td>
<td>RD12</td>
<td>RPD12/PMD12/RD12</td>
<td>LED8_G</td>
</tr>
<tr>
<td>77</td>
<td>RD2</td>
<td>AN25/RPD2/RD2</td>
<td>LED8_R</td>
</tr>
<tr>
<td>88</td>
<td>RF1</td>
<td>RPF1/PMD10/RF1</td>
<td>MODE</td>
</tr>
<tr>
<td>25</td>
<td>RB0</td>
<td>PGE1/AN/AN/RB0</td>
<td>P32_PGC/BTNU</td>
</tr>
<tr>
<td>24</td>
<td>RB1</td>
<td>PGC1/AN/RB1/CTED12/RB1</td>
<td>P32_PGD/BTNL</td>
</tr>
<tr>
<td>57</td>
<td>RG2</td>
<td>SCL1/RG2</td>
<td>SCL</td>
</tr>
<tr>
<td>56</td>
<td>RG3</td>
<td>SDA1/RG3</td>
<td>SDA</td>
</tr>
<tr>
<td>53</td>
<td>RF8</td>
<td>RPF8/RF8</td>
<td>SPI_CE</td>
</tr>
<tr>
<td>55</td>
<td>RF6</td>
<td>RPF6/SCK1/INT0/RF6</td>
<td>SPI_SCK</td>
</tr>
<tr>
<td>52</td>
<td>RF2</td>
<td>RPF2/RF2</td>
<td>SPI_SI</td>
</tr>
<tr>
<td>54</td>
<td>RF7</td>
<td>RPF7/RF7</td>
<td>SPI_SO</td>
</tr>
<tr>
<td>51</td>
<td>RF3</td>
<td>RPF3/RF3</td>
<td>SW0</td>
</tr>
<tr>
<td>50</td>
<td>RF5</td>
<td>RPF5/PMA8/RF5</td>
<td>SW1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>49</td>
<td>RF4</td>
<td>RPF4/PMA9/RF4</td>
<td>SW2</td>
</tr>
<tr>
<td>48</td>
<td>RD15</td>
<td>RPD15/RD15</td>
<td>SW3</td>
</tr>
<tr>
<td>47</td>
<td>RD14</td>
<td>RPD14/RD14</td>
<td>SW4</td>
</tr>
<tr>
<td>35</td>
<td>RB11</td>
<td>AN11/PMA12/RB11</td>
<td>SW5</td>
</tr>
<tr>
<td>34</td>
<td>RB10</td>
<td>CVREFOUT/AN10/RPB10/PMA13/CTED11/RB10</td>
<td>SW6</td>
</tr>
<tr>
<td>33</td>
<td>RB9</td>
<td>AN9/RPB9/CTED4/RB9</td>
<td>SW7</td>
</tr>
<tr>
<td>39</td>
<td>RF13</td>
<td>RPF13/RF13</td>
<td>UART_RX</td>
</tr>
<tr>
<td>40</td>
<td>RF12</td>
<td>RPF12/RF12</td>
<td>UART_TX</td>
</tr>
<tr>
<td>63</td>
<td>RC12</td>
<td>CLKI/RC12/OSC1</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>RC15</td>
<td>CLKO/RC15/OSC2</td>
<td></td>
</tr>
</tbody>
</table>