University of British Columbia
Electrical and Computer Engineering
Electrical and Biomedical Engineering Design Studio
ELEC291/ELEC292

The PIC32 Microcontroller System

Introduction

This document introduces a minimal microcontroller system using Microchip's PIC32MX130F064B microcontroller. The PIC32MX130 IC is a 32-bit RISC MIPS32 M4K Core microcontroller, featuring 32-Bit, 40MHz, 64KB (64K x 8) FLASH, in a DIP28 package.

Recommended documentation

PIC32MX Family Reference Manual:
http://ww1.microchip.com/downloads/en/DeviceDoc/61132B.pdf

PIC32MX Family Data Sheet:
http://ww1.microchip.com/downloads/en/DeviceDoc/60001168J.pdf

Assembling the Microcontroller System

Figure 1 shows the circuit schematic of the PIC32MX130 microcontroller system used in ELEC291/ELEC292. It can be assembled using a bread board. Table 1 below lists the components needed to assemble the circuit.

Quantity	Digi-Key Part #	Description
2	BC1148CT-ND	0.1uF ceramic capacitors
2	BC1157CT-ND	1uF ceramic capacitors
2	330QBK-ND	330Ω resistor
1	67-1102-ND	LED 5MM RED
1	67-1108-ND	LED 5MM GREEN
1	MCP1700-3302E/TO-ND	MCP17003302E 3.3 Voltage Regulator
1	N/A	BO230XS USB adapter
1	PIC32MX130F064B-I/SP-ND	PIC32MX130F064B
1	493-1548-ND	100uF/25V electrolytic capacitor
2	P8070SCT-ND	Push button switch

Table 1. Parts required to assemble the PIC32MX130 microcontroller system.

Notice that the PIC32MX130 works with a VDD voltage between 2.3V and 3.6V. For this reason a voltage regulator is required to convert the USB adapter 5V to 3.3V.

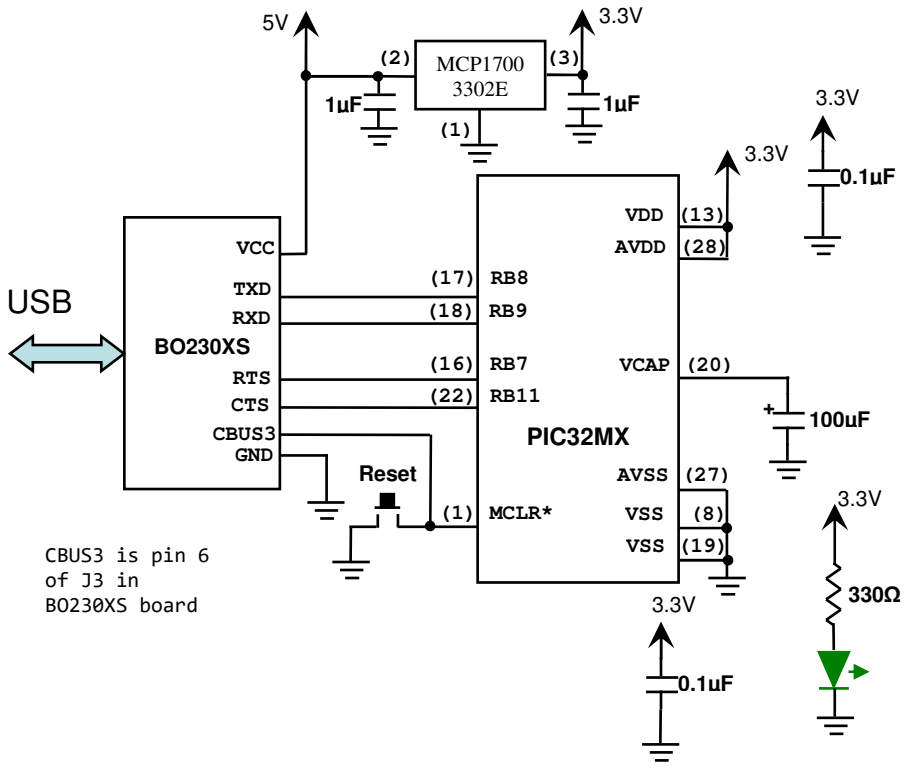


Figure 1. Circuit schematic of the microcontroller system.

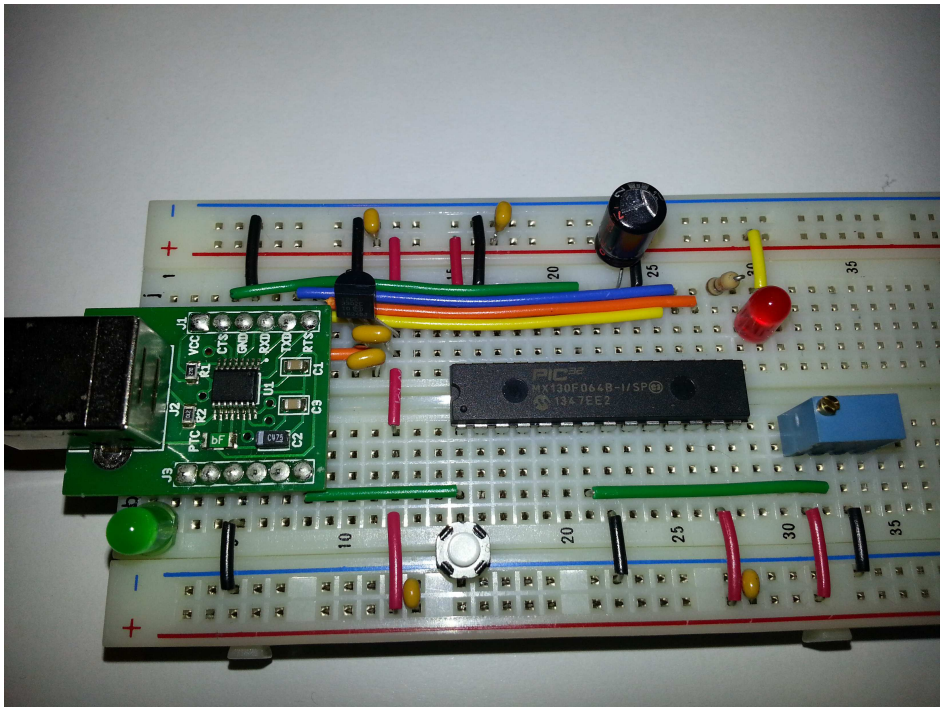


Figure 2. Bread boarded PIC32MX130 microcontroller system. The red LED is connected in series with a 330Ω resistor to pin 15 (RB6) as needed by the 'Blinky' example below. The 1kΩ potentiometer serves as an adjustable voltage divider (3.3V to GND) to provide an analog input to the 'ADCTest' example. The middle pin of the potentiometer is connected to pin 7 (AN5).

Setting up the Development Environment

To establish a workflow for the PIC32MX130 we need to install the following three packages:

1. CrossIDE V2.24 (or newer) & GNU Make V4.2 (or newer)

Download CrossIDE from: http://ece.ubc.ca/~jesusc/crosside_setup.exe and install it. Included in the installation folder of CrossIDE is GNU Make V4.2 (make.exe, make.pdf). GNU Make should be available in one of the folders of the PATH environment variable in order for the workflow described bellow to operate properly. For example, suppose that CrossIDE was installed in the folder "C:\crosside"; then the folder "C:\crosside" should be added at the end of the environment variable "PATH" as described here[1].

Some of the Makefiles used in the examples below may use a "wait" program developed by the author. This program (and its source code) can be downloaded from the course web page and must be copied into the CrossIDE folder or any other folder available in the environment variable "PATH".

2. GNU ARM Embedded Toolchain.

Download the XC32 Embedded Toolchain from:

http://courses.ece.ubc.ca/281/2017/XC32_1.42.zip

Once the download is complete, decompress the archive somewhere in your computer. The "bin" folder of the XC32 Embedded Toolchain must be added to the environment variable "PATH" in order for the workflow described bellow to operate properly. For example, if the toolchain is installed in the folder "C:\Programs\xc32\v1.42", then the folder "C:\Programs\xc32\v1.42\bin" must be added at the end of the environment variable "PATH" as described here[1].

3. PIC32 Flash Loader: PRO32.

Available in the web page for the course is the program "pro32.zip" developed by the author[2]. Download and decompress the archive file "pro32.zip" somewhere in your hard drive.

The folder of the PIC32 Flash loader must be added to the environment variable "PATH" in order for the workflow described bellow to operate properly. For example, if the PIC32MX130 Flash loader is installed in the folder "C:\CrossIDE\pro32", then the folder "C:\CrossIDE\pro32" must be added at the end of the environment variable "PATH" as described here[1].

[1] http://www.computerhope.com/issues/ch000549.htm

[2] The author wrote the program to flash all the members of the PIC32MX family, but he only tested it with the PIC32MX130F064B and PIC32MX170F256B microcontrollers so far.

Workflow.

The workflow for the PIC32MX130 microcontroller includes the following steps.

1. Creation and Maintenance of Makefiles.

CrossIDE version 2.25 or newer supports project management using **simple** Makefiles by means of GNU Make version 4.2 or newer. A CrossIDE project Makefile allows for easy compilation and linking of multiple source files, execution of external commands, source code management, and access to microcontroller flash programming. The typical Makefile is a text file, editable with the CrossIDE editor or any other editor, and looks like this:

```
# Since we are compiling in windows, select 'cmd' as the default shell.  This
# is important because make will search the path for a linux/unix like shell
# and if it finds it will use it instead.  This is the case when cygwin is
# installed.  That results in commands like 'del' and echo that don't work.
SHELL=cmd
# Specify the compiler to use
CC = xc32-gcc
# Specify the .elf to .hex converter
OBJCPY = xc32-bin2hex
# Specify the device we are using
ARCH = -mprocessor=32MX130F064B
# List the object files involved in this project
OBJ = Blinky.o

# The default 'target' (output) is Blinky.elf and 'depends' on
# the object files listed in the 'OBJS' assignment above.
# These object files are linked together to create Blinky.elf.
# The linked file is converted to hex using program $(OBJCPY).
Blinky.elf: $(OBJ)
	$(CC) $(ARCH) -o Blinky.elf Blinky.o -mips16 -DXPRJ_default=default \
		-legacy-libc -Wl,-Map=Blinky.map
	$(OBJCPY) Blinky.elf
	@echo Success!

# The object file Blinky.o depends on Blinky.c. Blinky.c is compiled
# to create Blinky.o.
Blinky.o: Blinky.c
	$(CC) -g -x c -mips16 -Os -c $(ARCH) -MMD -o Blinky.o Blinky.c \
		-DXPRJ_default=default -legacy-libc

# Target 'clean' is used to remove all object files and executables
# associated wit this project
clean:
	@del $(OBJS) *.elf *.hex *.map *.d 2>NUL

# Target 'LoadFlash' is used to load the hex file to the microcontroller
# using the flash loader.
LoadFlash:
	@Taskkill /IM putty.exe /F 2>NUL | wait 500
	pro32 -p -v Blinky.hex

# Phony targets can be added to show useful files in the file list of
# CrossIDE or execute arbitrary programs:
dummy: Blinky.hex Blinky.map

explorer:
	explorer .
```

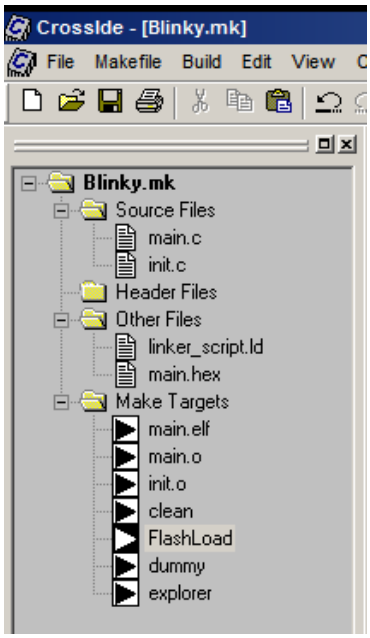
The preferred extension used by CrossIDE Makefiles is ".mk". For example, the file above is named "Blinky.mk".

Makefiles are an industry standard. Information about using and maintaining Makefiles is widely available on the internet. For example, these links show how to create and use simple Makefiles.

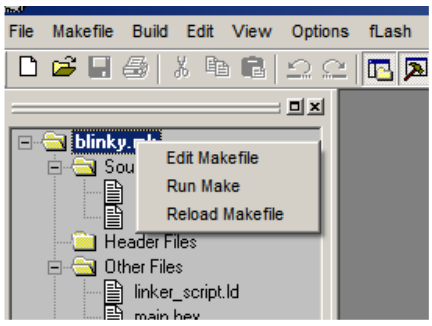
https://www.gnu.org/software/make/manual/make.html
http://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/
https://www.cs.swarthmore.edu/~newhall/unixhelp/howto_makefiles.html
https://en.wikipedia.org/wiki/Makefile

2. Using Makefiles with CrossIDE: Compiling, Linking, and Loading.

To open a Makefile in CrossIDE, click "Makefile"→"Open" and select the Makefile to open. For example "Blinky.mk". The project panel is displayed showing all the targets and source files:



Double clicking a source file will open it in the source code editor of CrossIDE. Double clicking a target 'makes' that target. Right clicking the Makefile name shows a pop-up menu that allows for editing, running, or reloading of the Makefile:



Additionally, the Makefile can be run by means of the Build menu or by using the Build Bar:

Clicking the 'wall' with green 'bricks' makes only the files that changed since the last build. Clicking the 'wall' with colored 'bricks' makes all the files. Clicking the 'brick' with an arrow, makes only the selected target. You can also use F7 to make only the files that changed since the last build and Ctrl+F7 to make only the selected target.

Compiling & Linking

After clicking the build button this output is displayed in the report panel of CrossIDE:

```
----------------- CrossIde - Running Make -----------------
xc32-gcc -g -x c -mips16 -Os -c -mprocessor=32MX130F064B -MMD -o Blinky.o Blinky.c \
        -DXPRJ_default=default -legacy-libc
xc32-gcc -mprocessor=32MX130F064B -o Blinky.elf Blinky.o -mips16 -DXPRJ_default=default \
        -legacy-libc -Wl,-Map=Blinky.map
xc32-bin2hex Blinky.elf
Success!
```

Loading the Hex File into the Microcontroller's Flash Memory

To load the flash memory to the microcontroller, double click the 'FlashLoad' target. This output is then displayed in the report panel of CrossIDE:

```
----------------- CrossIde - Running Make -----------------
..\pro32\pro32 -p -v Blinky.hex
Connected to COM4
IDCode=0x4d07053 : PIC32MX130F064B

Erasing flash memory.
Loading Program Flash: ################## Done (2304 bytes).
Loading Boot Flash: ####### Done (896 bytes).
Done 3200 bytes in 3.5 seconds. (903.7 bytes/sec)

Verifiying program flash: ################## Done (2304 bytes).
Verifiying boot flash: ####### Done (896 bytes).
Done verifying 3200 bytes in 5.5 seconds. (577.8 bytes/sec)
```

A file named "COMPORT.inc" is created after running the flash loader program. The file contains the name of the port used to load the program, for example, in the example above COM4 is stored in the file. "COMPORT.inc" can be used in the Makefile to create a target that starts a PuTTy serial terminal session using the correct serial port:

```
PORTN=$(shell type COMPORT.inc)
.
.
putty:
        @Taskkill /IM putty.exe /F 2>NUL | wait 500
        c:\putty\putty.exe -serial $(PORTN) -sercfg 115200,8,n,1,N -v
```

For more details about using "COMPORT.inc" check the project examples below.

Project Examples

The following Project examples are available in the web page of the course.

Blinky: 'blinks' an LED connected to pin 15 (RB6). This is the same project used in the examples above.

SerialPort: Configures the serial port and uses it to receive/transmit serially to PuTTy.

HelloWorld: Uses printf() to display "Hello, World!" using the serial port and PuTTy.

StdioTest: Adds support for serial input routines via stdio. The function gets() is used in this program to receive information serially.

TimerIRQ: Similar to "Blinky" but instead of using a delay loop, it uses a timer interrupt. Timer 1 and its corresponding interrupt are used in this example.

ADCtest: Reads a channel of the built in ADC (pin 7, AN5) and displays the result using printf() via PuTTY.