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

The AT89LP51RC2 Microcontroller System

Introduction

This document introduces a microcontroller system using Atmel/Microchip's AT89LP51RC2 microcontroller. The AT89LP51RC2 IC is an 8-bit 8051 compatible microcontroller, 24MHz, 32KB Flash, in a DIP40 package.

Recommended documentation

AT89LP51RC2 User Manual: http://ww1.microchip.com/downloads/en/DeviceDoc/doc3722.pdf

Assembling the Microcontroller System

Figure 1 shows the circuit schematic of the AT89LP51RC2 microcontroller system used in ELEC291/ELEC292. Table 1 below lists the components used to assemble the circuit. Figure 2 shows the AT89LP51RC2 assembled in a bread board.

Quantity	Description
2	0.1uF capacitors
1	330Ω resistor
1	LED 5MM GREEN
1	22.1184MHZ Crystal
1	BO230XS USB adapter
1	AT89LP51RC2 IC 80C51 MCU 32K FLASH 40-DIP
2	Push button switch

Table 1. Parts required to assemble the AT89LP51RC2 system.

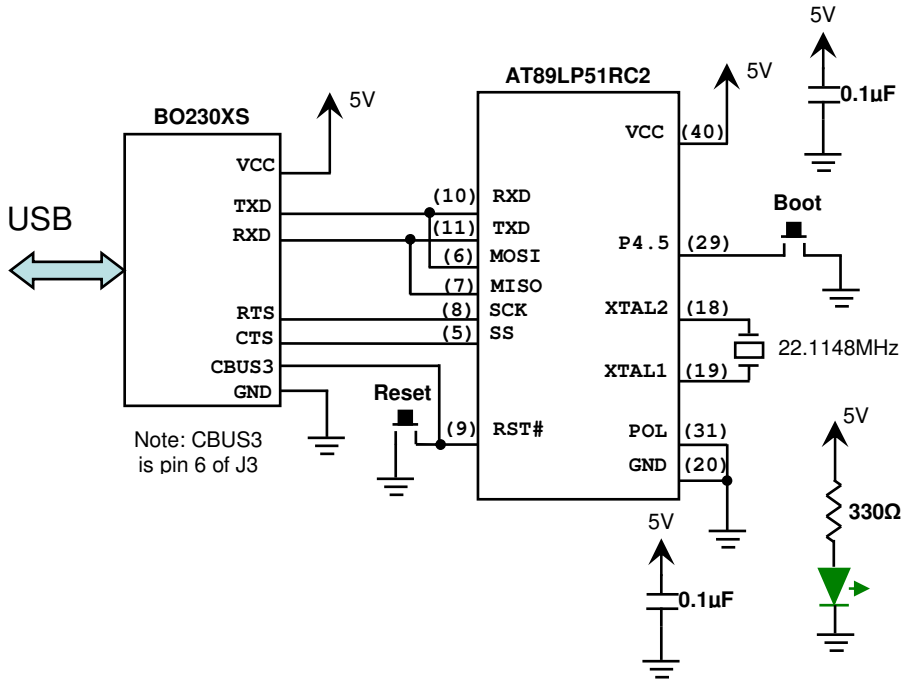


Figure 1. Circuit schematic of the AT89LP51RC2 microcontroller system.

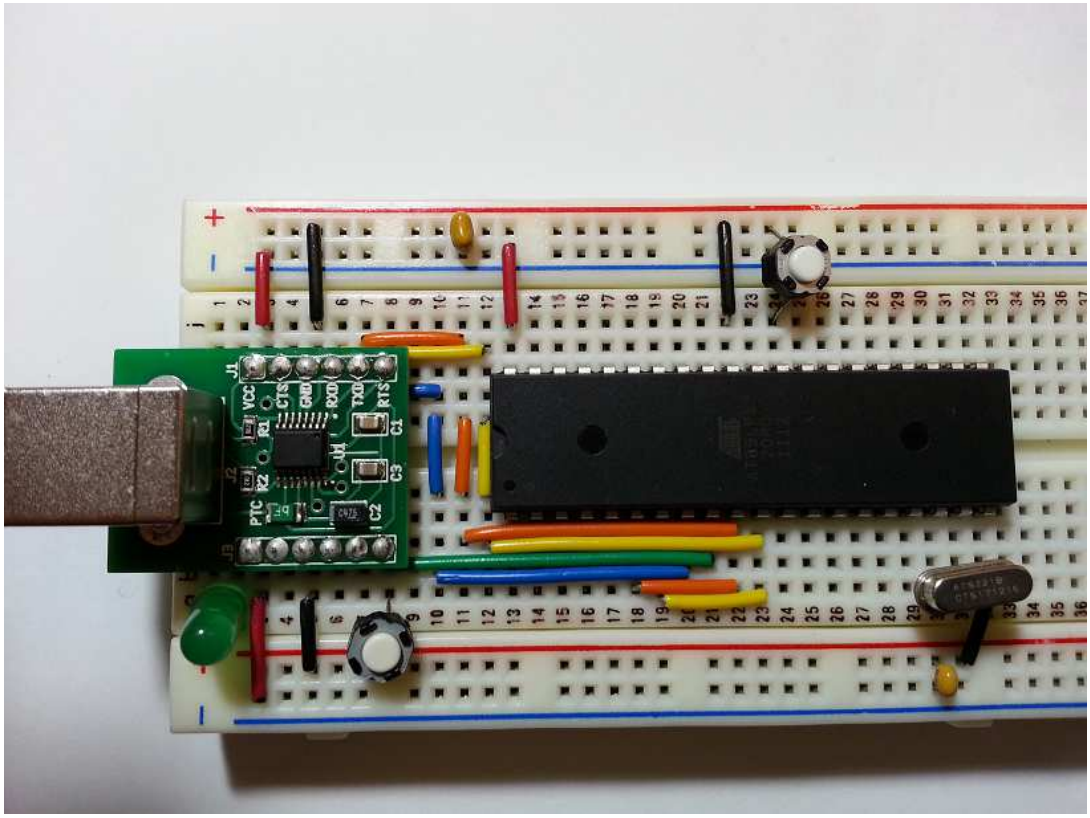


Figure 2. AT89LP51RC2 microcontroller system assembled in a bread board.

Setting up the Development Environment

To establish a workflow for the AT89LP51RC2 we need to install and configure the following three packages:

1. **CrossIDE V2.26 (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. **CALL51 Toolchain for Windows.**

   CALL51[2] (C compiler, Assembler, Linker, Librarian for the 8051) is included with the installation of CrossIDE. The "bin" folder of the CALL51 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:\CrossIDE\Call51", then the folder "C:\CrossIDE\Call51\Bin" must be added at the end of the environment variable "PATH" as described here[1].

3. **AT89LP51RC2 Flash Loader: Pro89lp.**

   Available in the web page for the course is the program "Pro89lp.zip" developed by the author. Download and decompress the archive file "Pro89lp.zip" somewhere in your hard drive.

   The folder of the AT89LP51RC2 flash loader must be added to the environment variable "PATH" in order for the workflow described bellow to operate properly. For example, if the AT89LP51RC2 flash loader is installed in the folder "C:\CrossIDE\Pro89lp", then the folder "C:\CrossIDE\Pro89lp" must be added at the end of the environment variable "PATH" as described here[1]. Alternatively, the full path of the Pro89lp.exe can be provided in the Makefiles.

4. **PuTTy**

   PuTTy is used to display information sent from the serial port of the microcontroller. The Makefiles provided assume that the PuTTy is available in the "PATH" environment variable. To add PuTTy to the "PATH" follow the instructions here[1].

---

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

[2] Derived work from SDCC. The author used to contribute to this project but branched off after some disagreements with other developers. The assembler, linker, and librarian, as well as many of the C library functions were developed by the author.

Workflow.

The workflow for the AT89LP51RC2 microcontroller includes the following steps.

1. Creation and Maintenance of Makefiles.

CrossIDE version 2.24 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=c51
# Object files to link
OBJS=Blinky.obj

# The default 'target' (output) is Blinky.hex and 'depends' on
# the object files listed in the 'OBJS' assignment above.
# These object files are linked together to create Blinky.hex.
Blinky.hex: $(OBJS)
        $(CC) $(OBJS)
        @echo Done!

# The object file Blinky.o depends on Blinky.c. Blinky.c is compiled
# to create Blinky.o.
Blinky.obj: Blinky.c
        $(CC) -c Blinky.c

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

# Target 'FlashLoad' is used to load the hex file to the microcontroller
# using the flash loader.  If the folder of the flash loader has been
# added to 'PATH' just 'Prog89lp' is needed.  Otherwise, a valid path
# must be provided as shown below.
LoadFlash:
        Pro89lp -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".

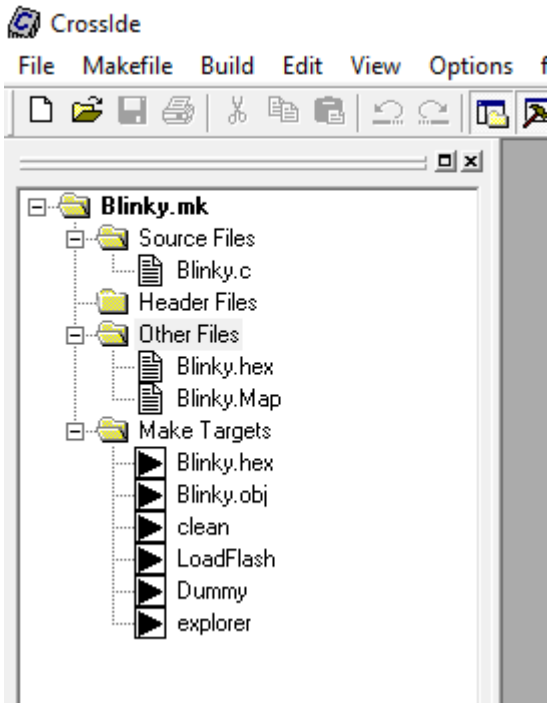
Makefiles are an industry standard. Information about using and maintaining Makefiles is widely available on the internet. For example, these links show how to create and use simple Makefiles.

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

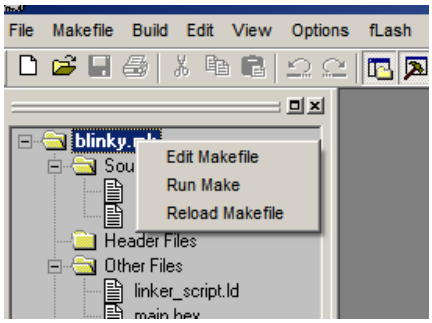
https://en.wikipedia.org/wiki/Makefile

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 -----------------
c51 -c Blinky.c
c51 Blinky.obj
Done!
```

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 -----------------
Pro89lp.exe -p -v Blinky.hex
AT89LP ISP/SPI programmer using the BO230X board. (C) Jesus Calvino-Fraga (2016-2017)
Blinky.hex: 176 bytes loaded
Connected to COM131
AT89LP51RC2 detected.
Erasing... Done.
Loading flash memory: ### Done.
Verifiying flash memory: ### Done.
Writing fuses... Done.
Actions completed in 0.2 seconds.
```

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 COM131 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 P3.7. This is the same project used in the examples above.

BlinkyISR: Similar to "Blinky" but instead of using a delay loop, it uses a timer interrupt. Timer 0 and its corresponding interrupt service routine (ISR) are used in this example.

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

LCD: Shows how to configure and use a Hitachi compatible 16 x 2 LCD in four bit mode.

SPI_ADC: Shows how to communicate to the MCP3008 ADC using SPI. The program prints the resulting voltage using printf() and floating point.