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

The AVR Microcontroller System

Copyright © 2017, Jesus Calvino-Fraga. Not to be copied, used, or revised without explicit written permission from the copyright owner.

Introduction

This document introduces a minimal microcontroller system using Atmels's ATmega328P microcontroller. The ATmega328P IC is an 8-bit microcontroller, featuring 32-Bit, 40MHz, 64KB (64K x 8) FLASH, in a DIP28 package.

Recommended documentation

ATmega328P User Manual

ATmega Family Data Sheet

Assembling the Microcontroller System

Figure 1 shows the circuit schematic of the ATmega328P 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
3	BC1148CT-ND	0.1uF ceramic capacitors
1	1.0KQBK-ND	1kΩ resistor
2	330QBK-ND	330Ω resistor
1	67-1102-ND	LED 5MM RED
1	67-1108-ND	LED 5MM GREEN
1	CTX1085-ND	Crystal 16.0000MHZ 18PF T/H
1	N/A	BO230XS USB adapter
1	ATMEGA328P-PU-ND	ATMEGA328P
1	P8070SCT-ND	Push button switch

Table 1. Parts required to assemble the ATmega328P microcontroller system.

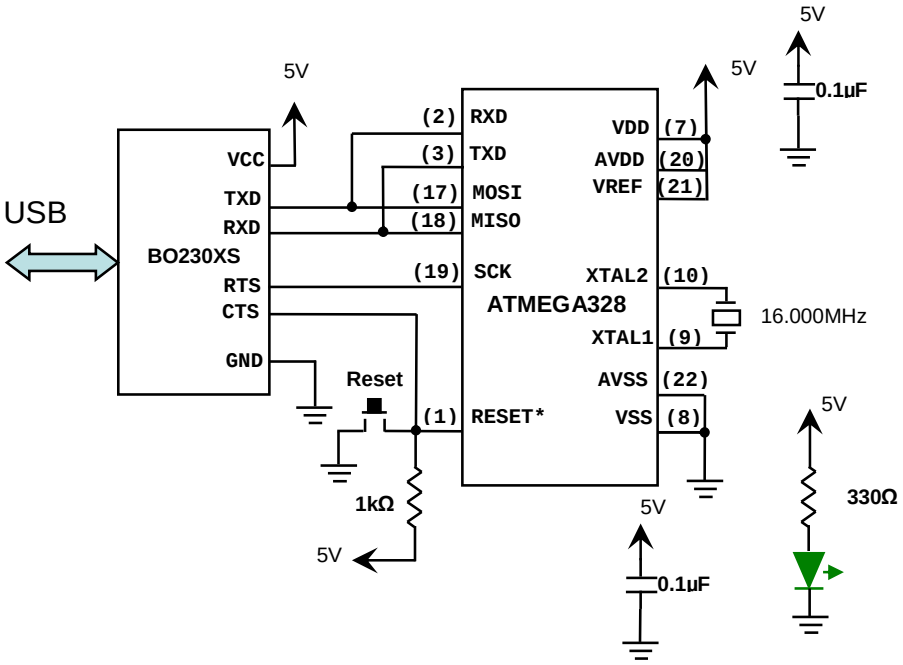


Figure 1. Circuit schematic of the microcontroller system.

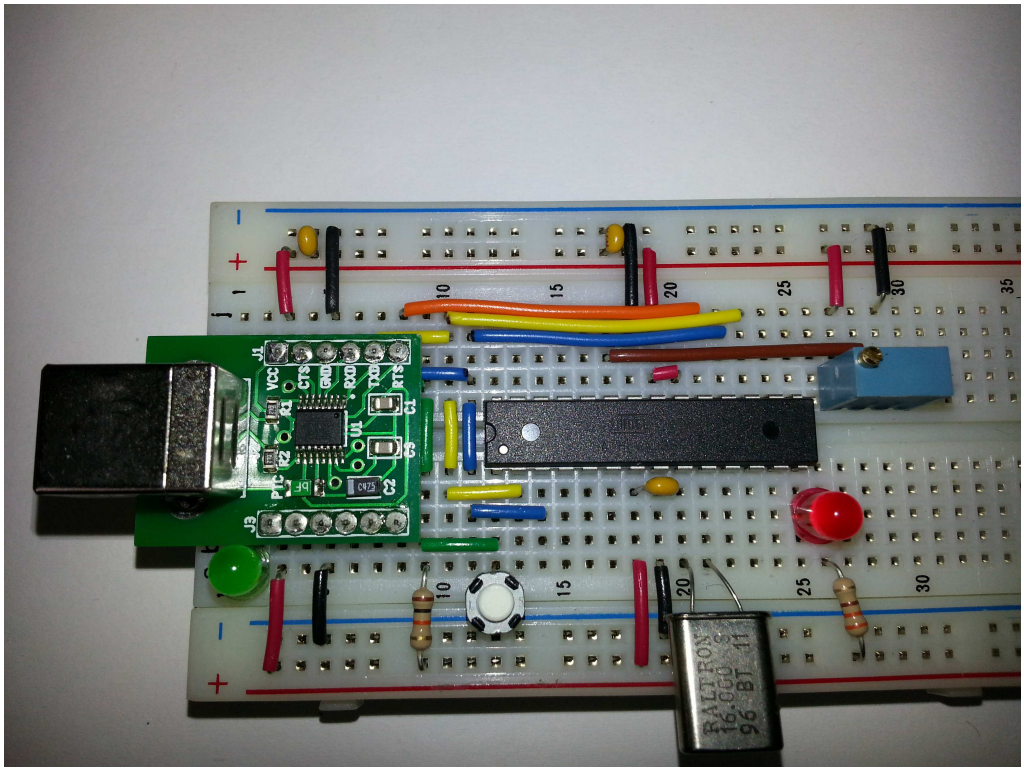


Figure 2. Bread boarded ATmega328P microcontroller system. The red LED is connected in series with a 330Ω resistor to pin 14 (PB6) as needed by the 'Blinky' example below. The 1kΩ potentiometer serves as an adjustable voltage divider (5V to GND) to provide an analog input to the 'ADCTest' example. The middle pin of the potentiometer is connected to pin 23 (ADC0).

Setting up the Development Environment

To establish a workflow for the ATMEGA328P 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 in: http://www.computerhope.com/issues/ch000549.htm

   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. **Atmel AVR 8-bit Toolchain for Windows.**

   Download the AVR 8-bit Embedded Toolchain from:

   http://www.atmel.com/tools/ATMELAVRTOOLCHAINFORWINDOWS.aspx

   Once the download is complete, decompress/install the archive somewhere in your computer. The "bin" folder of the AVR 8-bit 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\avr8-gnu-toolchain", then the folder "C:\Programs\avr8-gnu-toolchain\bin" must be added at the end of the environment variable "PATH" as described here.

3. **ATmega328P Flash Loader: spi_atmega328.**

   Available in the web page for the course is the program "spi_atmega328.zip" developed by the author. Download and decompress the archive file "spi_atmega328.zip" somewhere in your hard drive.

   The folder of the ATmega328P flash loader must be added to the environment variable "PATH" in order for the workflow described bellow to operate properly. For example, if the ATmega328P flash loader is installed in the folder "C:\CrossIDE\spi_atmega328", then the folder "C:\CrossIDE\spi_atmega328" must be added at the end of the environment variable "PATH" as described here.

Workflow.

The workflow for the ATmega328P 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:

```
# Specify the compiler to use
CC= avr-gcc
# Specify the microcontroller
CPU=-mmcu=atmega328p
# C compiler options
COPT= -g -Os $(CPU)
# Object files to link
OBJS= 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 aver-objcopy.
blinky.elf: $(OBJS)
	avr-gcc $(CPU) -Wl,-Map,blinky.map $(OBJS) -o blinky.elf
	avr-objcopy -j .text -j .data -O ihex blinky.elf blinky.hex
	@echo done!

# The object file blinky.o depends on blinky.c. blinky.c is compiled
# to create blinky.o.
blinky.o: blinky.c
	avr-gcc $(COPT) -c blinky.c

# Target 'clean' is used to remove all object files and executables
# associated wit this project
clean:
	@del *.hex *.elf *.o 2> nul

# Target 'FlashLoad' is used to load the hex file to the microcontroller
# using the flash loader.
FlashLoad:
	spi_atmega328 -CRYSTAL -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
	@echo Hello dummy!

explorer:
	explorer .
```

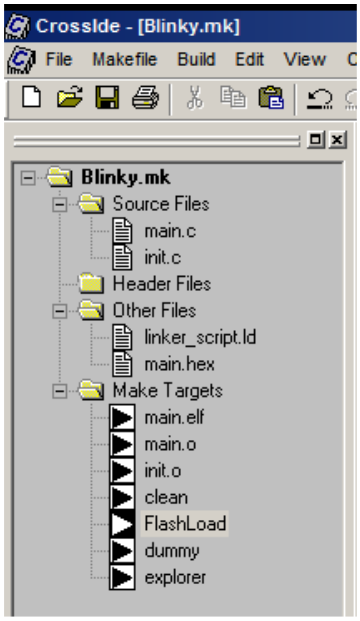
The preferred extension used by CrossIDE Makefiles is ".mk". For example, the file above is named "blinky.mk".

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.

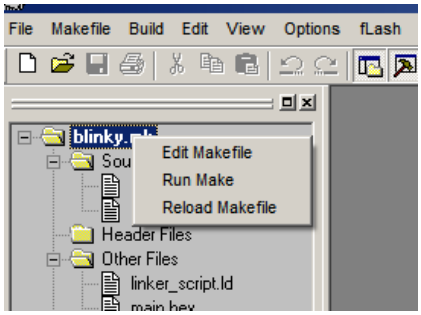
https://www.gnu.org/software/make/manual/make.html
http://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/
https://www.cs.swarthmore.edu/~newhall/unixhelp/howto_makefiles.html
https://en.wikipedia.org/wiki/Makefile

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 -----------------
avr-gcc -g -Os -mmcu=atmega328p -c blinky.c
avr-gcc -mmcu=atmega328p -Wl,-Map,blinky.map blinky.o -o blinky.elf
avr-objcopy -j .text -j .data -O ihex blinky.elf blinky.hex
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 -----------------
spi_atmega328 -CRYSTAL -p -v blinky.hex
Atmega328 programmer using BO230X board. (C) Jesus Calvino-Fraga (2016)
Connected to COM105
Atmega328P detected.
Erasing flash memory: # Done.
Loading flash memory: ## Done.
Verifying flash memory: ## Done.
Writing configuration fuses: ### Done.
Actions completed in 0.4 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 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 14 (PB0). This is the same project used in the examples above.

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

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

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 23, ADC0) and displays the result using printf() via PuTTY using floating point numbers. This program uses 4358 bytes of flash memory.

ADCtestEff1: Reads a channel of the built in ADC (pin 23, ADC0) and displays the result using printf() via PuTTY, using only integers. This program uses 2286 bytes of flash memory.

ADCtestEff2: Reads a channel of the built in ADC (pin 23, ADC0) and displays the result using serial functions only via PuTTy. printf() is not used in this example. This program uses 1060 bytes of flash memory.