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

The STM32F051 Microcontroller System

Introduction

This document introduces a minimal microcontroller system using the ST Microelectronics' STM32F051 ARM Cortex-M0 microcontroller.

Recommended documentation

RM0091 Reference manual: "*STM32F0x1/STM32F0x2/STM32F0x8 advanced ARM®-based 32-bit MCUs*":

http://www.st.com/content/ccc/resource/technical/document/reference_manual/c2/f8/8a/f2/18/e6/43/96/DM00031936.pdf/files/DM00031936.pdf/jcr:content/translations/en.DM00031936.pdf

Datasheet - production data: "*STM32F051x4 STM32F051x6 STM32F051x8 ARM®-based 32-bit MCU, 16 to 64 KB Flash, 11 timers, ADC, DAC and communication interfaces, 2.0-3.6 V*":

http://www.st.com/content/ccc/resource/technical/document/datasheet/55/53/3e/86/29/61/41/d9/DM00039193.pdf/files/DM00039193.pdf/jcr:content/translations/en.DM00039193.pdf

Assembling the Microcontroller System

Figure 1 shows the circuit schematic of the STM32F051 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	270QBK-ND	270Ω resistor
1	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	497-13626-ND	STM32F051K8T6
1	1528-1065-ND	LQFP32 to DIP32 adapter board
2	A26509-16-ND	16-pin header connector
1	P8070SCT-ND	Push button switch

Table 1. Parts required to assemble the STM32F051 microcontroller system.

Before assembling the circuit in the breadboard, the STM32F051 microcontroller has to be soldered to the LQFP32 to DIP32 adapter board. This task can be accomplished using a solder iron as described in this video:

https://www.youtube.com/watch?v=8yyUlABj29o

Once the microcontroller is soldered to the adapter board, solder the two 16-pin header connectors. The assembled bread boarded circuit should look similar to the one in figure 1. Notice that the STM32F051 works with a VDD voltage between 2.0V and 3.6V. For this reason a voltage regulator is required to convert the USB adapter 5V to 3.3V.

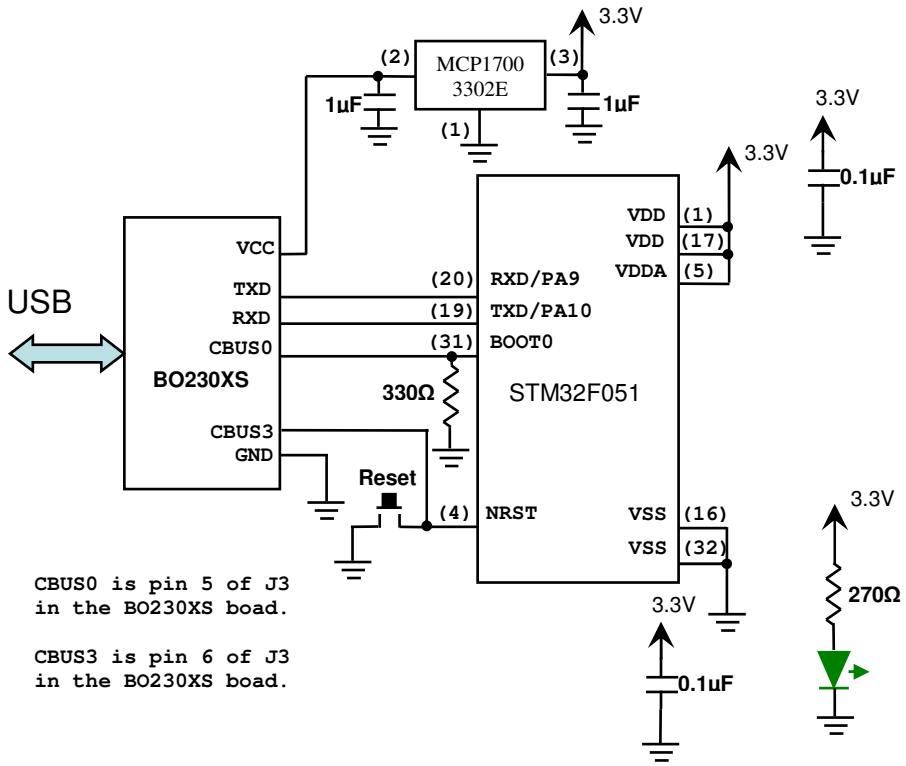


Figure 1. Circuit schematic of the microcontroller system.

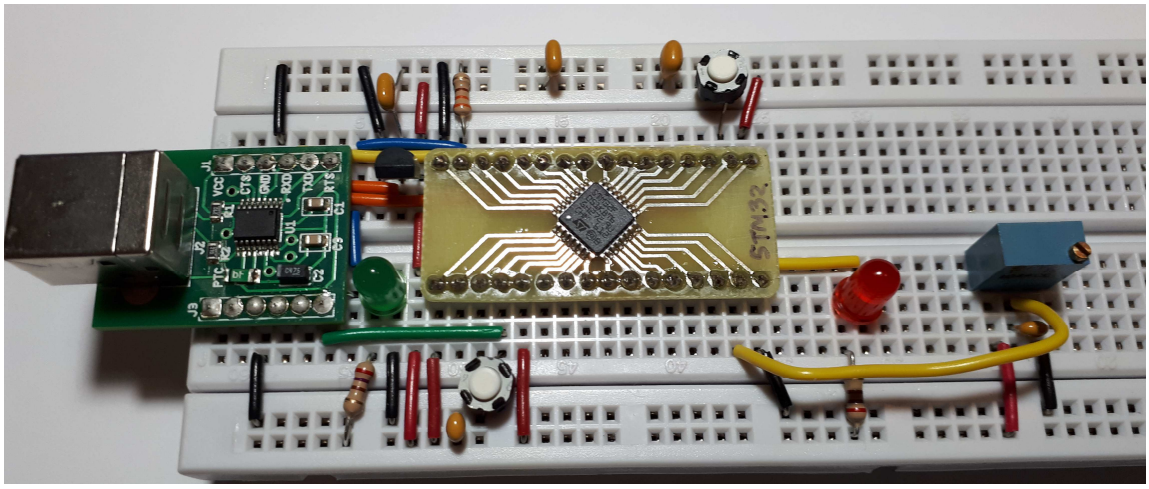


Figure 2. Bread boarded STM32F051 microcontroller system. The red LED is connected to pin 6 (PA0) as needed by the 'blinky' example below. The 1kΩ potentiometer serves as an adjustable voltage divider (3.3V to GND) to provide an analog input to the 'ADC' examples. The middle pin of the potentiometer is connected to pin 15 (PB1). The pushbutton connected between pin 18 (PA8) and ground is used in the 'pushbutton' project example.

Setting up the Development Environment

To establish a workflow for the STM32 we need to install the following three packages:

1. **CrossIDE V2.25 (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 and install the GNU ARM Embedded Toolchain from:

   https://developer.arm.com/open-source/gnu-toolchain/gnu-rm/downloads.

   The "bin" folder of the GNU ARM 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\GNU Tools ARM Embedded", then the folder "C:\Programs\GNU Tools ARM Embedded\5.4 2016q2\bin" must be added at the end of the environment variable "PATH" as described here[1].

   Notice that the folder "5.4 2016q2" was the folder available at the time of writing this document. For newer versions of the GNU ARM Embedded Toolchain this folder name will change to reflect the installed version.

3. **STM32 Flash Loader.**

   Available in the web page for the course is the program "STMFlashLoader"[2]. The "bin" folder of the STM32 Flash loader must be added to the environment variable "PATH" in order for the workflow described bellow to operate properly. For example, if the STM32 Flash loader is installed in the folder "C:\Programs\STMFlashLoader", then the folder "C:\Programs\STMFlashLoader\bin" must be added at the end of the environment variable "PATH" as described here[1].

Workflow.

The workflow for the STM32F051 microcontroller includes the following steps.

1. **Creation and Maintenance of Makefiles.**

---

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

[2] 'FlashLoader' is derived from FLASHER-STM32 available from http://www.st.com/en/development-tools/flasher-stm32.html. The program was modified in order to integrate it with both CrossIDE and GNU Make.

CrossIDE version 2.26 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:

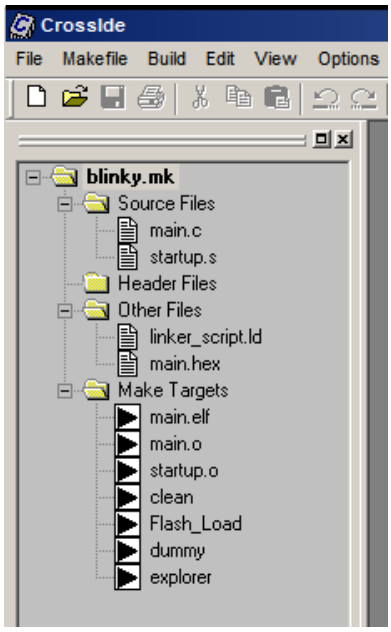
```
# Since we are compiling in windows, select 'cmd' as the default shell.  This
# is important because make will search the path for a linux/unix like shell
# and if it finds it will use it instead.  This is the case when cygwin is
# installed.  That results in commands like 'del' and echo that don't work.
SHELL=cmd
# Specify the compiler to use
CC=arm-none-eabi-gcc
# Specify the assembler to use
AS=arm-none-eabi-as
# Specify the linker to use
LD=arm-none-eabi-ld

# Flags for C compilation
CCFLAGS=-mcpu=cortex-m0 -mthumb -g
# Flags for assembly compilation
ASFLAGS=-mcpu=cortex-m0 -mthumb -g
# Flags for linking
LDFLAGS=-T linker_script.ld --cref -nostartfiles

# List the object files used in this project
OBJS= startup.o main.o

# The default 'target' (output) is main.elf and `depends` on
# the object files listed in the `OBJS` assignment above.
# These object files are linked together to create main.elf.
# The linked file is converted to hex using program objcopy.
main.elf: $(OBJS)
        $(LD) $(OBJS) $(LDFLAGS) -Map main.map -o main.elf
        arm-none-eabi-objcopy -O ihex main.elf main.hex

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

# The object file startup.o depends on startup.s.  startup.c is
# assembled to create startup.o
startup.o: startup.s
        $(AS) $(ASFLAGS) startup.s -asghl=startup.lst -o startup.o

# Target `clean` is used to remove all object files and executables
# associated wit this project
clean:
        del $(OBJS)
        del main.elf main.hex main.map
        del *.lst

# Target `Flash_Load` is used to load the hex file to the microcontroller
# using the flash loader.
Flash_Load:
        STMFlashLoader -ft230-c -i STM32F0_5x_3x_64K -e --all -d --fn main.hex --v

# Dummy targets can be added to show useful files in the file list of
# CrossIDE or execute arbitrary programs:
dummy: linker_script.ld main.hex
        @echo :-)

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.

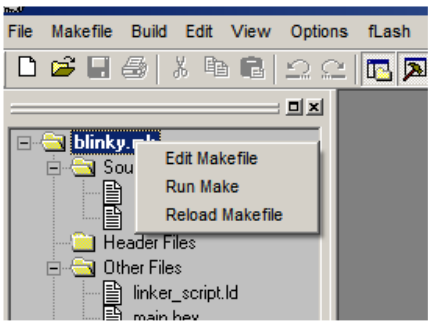
http://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/
https://www.cs.swarthmore.edu/~newhall/unixhelp/howto_makefiles.html
https://en.wikipedia.org/wiki/Makefile

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 -----------------
arm-none-eabi-as -mcpu=cortex-m0 -mthumb -g startup.s -asghl=startup.lst -o startup.o
arm-none-eabi-gcc -c -mcpu=cortex-m0 -mthumb -g main.c -o main.o
arm-none-eabi-ld startup.o main.o -T linker_script.ld -cref -nostartfiles -Map
main.map -o main.elf
objcopy -O ihex main.elf main.hex
```

Loading the Hex File into the Microcontroller's Flash Memory

To load the program into the microcontroller double click the 'Flash_Load' target. This output is then displayed in the report panel of CrossIDE and the program starts running.

```
----------------- CrossIde - Running Make -----------------
STMFlashLoader -ft230 -c -i STM32F0_5x_3x_64K -e --all -d --fn main.hex --v
COM11 <USB Serial Port>
Opening Port                            [OK]
Activating device                               [OK]
Erasing... erasing all pages                            [OK]
main.hex: loaded 316 bytes.
Downloading  * Done.
Verifying    * Done.
Appliying reset to STM32F051... done
```

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 thie example above COM118 is stored in the file. "COMPORT.inc" can be used in the Makefile to create a target that starts a PuTTy serial 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 web page of the course. They are all written for the STM32F051K8T6 ARM microcontroller.

Blinky: 'blinks' an LED connected to pin PA0 (pin 6). This is the same project used in the examples above. The startup code of this project is written in ARM assembly language.

BlinkyNew: Similar to "blinky" above. The main difference is that this project uses a C file for the startup code. Also, this project includes all the requirements to add and use functions such as printf() and scanf().

TimerIRQ: Similar to "blinky" but instead of using a delay loop, it uses a timer interrupt.

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

PrintADC: Reads a channel of the built in ADC (pin 15) and displays the result using printf() and PuTTY. Since this project uses printf() with floating point support, a significant amount of flash memory is used.

PrintADCeff1: Reads a channel of the built in ADC (pin 15) and displays the result using printf() and PuTTY. This project does not use floating point with printf() therefore the hex file is about 1/3 the size of the previous project.

PrintADCeff2: Reads a channel of the built in ADC (pin 15) and displays the result using the serial port and PuTTY. This project does not use printf() therefore the hex file is about 1/2 the size of the previous project.

PrintADC2Inputs: Similar to the ADC examples above, but reads two channels instead.

Pushbutton: Shows how to read a push button connected between PA8 (pin 18) and ground.

Usart2: Shows how to configure and use the second USART in the STM32F051.

Freq_Gen: Shows how to configure timer 1 to generate a square wave at PA0 (pin 6) with a frequency between 400Hz and 200kHz.