CS 2230
Assignment 3
Fall 2016
Due Tuesday
Be able to demonstrate it
as well as submitting code in repo.

At this point we hope you can communicate via a terminal program on your laptop to the MSP430. Getting to this point involves different platforms and the proper installation of the toolchain that allows you to flash a program to the processor using the gcc toolchain and the libemb library, each of which is an open source project with possible idiosyncracies. Below I will mention various problems people have had achieving communication.

But assuming you have achieved liftoff …

You are now to write a C program which will be flashed to the MSP430. It will accept characters from the host after providing some suitable prompt to the user. These characters had better be from '0' to '9' because they will represent decimal digits. (For the brave, implementing hex input from user will be rewarded appropriately.)

The characters e.g. '1' '3' '5' followed by <ENTER> will generate characters including those generated by <ENTER> and you must use those characters to build a number (int) that is then printed back to the user. The I/O for all of this uses the so-called libemb library, parts of which are loaded when the –lserial –lconio switches are used when building the executable (whose file name ends with .elf by default, 'elf' standing for 'extensible loader format'). For fun, $file a.out on your laptop will reveal that a.out is also an elf file.

The libemb library has its own functions and syntax, similar but not always the same as the standard library printf() etc functions. We have provided examples of usage, and https://github.com/wendlers/libemb is the github for this open source project. For example, cio_printf() is a printf() like function in the library that accepts format strings just as printf() does.

So basically you will create an int from characters whereas before you created characters from an int. Be clear that ‘0’ is not integer 0. To map the values ‘0’ to ‘9’ to the values 0 … 9 simply subtract ‘0’ which is merely a convenience provided by the compiler to avoid looking up ASCII codes. (Surely you believe that ‘0’ – ‘0’ is numerically 0. Since the ascii codes are ordered, ‘1’ – ‘0’ had best be 1 and so on, eh ?

You must also test the number you received for ‘parity’ , is it odd, or is it even ? If it is odd, you will blink the red light and if it is even you will blink the green light. We have talked about code that does this in class and the program below has code for both serial I/O and for writing to Port 1 pins for both on-board LEDs. NOTE that if you had wires connected to different pins and LEDs in your kit, you could modify the code appropriately. What a good bit of science to attempt ?

The below code is in this repo (a3). Push a single piece of code and be able to demonstrate your program in class on Tuesday. Obviously you should be able to use some terminal program (minicom, screen) to execute this program. Development of the code can be done at first on the laptop, before making sure that your LED lighting is solid. But the logic for character handling can be tested without the MSP. Either scanf() or getc() in the C libraries can be used for testing purposes to supply characters from user.

Errata – a discussion of various complications with MSP430 toolchain.

- One can always reinstall from the original directions for using scripts to make sure the libraries, including libemb are installed. Our biggest mystery has been that the same script when Colin runs it seems to install libemb properly.
- To use minicom or screen one needs to know the ‘device’ name chosen by your Unix when the MSP430 is plugged in. A file is created in the

/dev directory (which is a subdirectory from the 'root' directory of the file system which is '/' . /dev , /home, /usr are all subdirectories of the root, top level directory. FUN FACT: root is the only directory where '.' and '..' are the same directory.

- Be sure to specify 9600 baud if you have declared 9600 in your program
- Be sure your jumpers are turned as in step 5 of Launchpad instructions.
- If you use sudo apt-get msp430gcc instead of our scripts the directories are inconsistent with our intended setup. Please don't.
- When you run msp430-gcc you must supply the mmcu value (NOT MMCU) or the wrong headers are looked for.
- No VMs , No VMs , No VMs
- Some apparent error messages can be ignored. When flashing the board you simply want to see that the program was written to the board; a message that some number of bytes was written to the text section usually is reported. Not being able to find your msp430 device is a different matters. Sometimes unplugging the msp430 and plugging it back in is needed to establish handshaking properly.
- Watch videos. There are 10 or so including one on serial I/O and blinking LEDs.
- Have a good attitude

HERE IS CODE FROM CLASS – BELOW IT IS A MAKEFILE THAT IS IN THE REPO

```
#include <msp430.h>
#include <libemb/serial/serial.h>
#include <libemb/conio/conio.h>

int main(void)
{
   WDTCTL  = WDTPW | WDTHOLD;
   BCSCTL1 = CALBC1_1MHZ;
   DCOCTL  = CALDCO_1MHZ;

   P1DIR |= 0b01000001; // direction output for P1.0 and P1.6
   P1OUT |= 0b01000001; // set P1.0 and P1.6 ON

   serial_init(9600);

   // while(1) {
   for(;;) {
      cio_print((char *) "I'm waiting for a character!\n\r");
      char c = cio_getc();
      cio_printc(c);
      cio_print((char *) "\n\r");

      for(int i = 0; i < 4; i++) {
         P1OUT ^= 0b01000001; // toggle both LEDs
         // P1OUT = P1OUT ^ 0b01000001;
         __delay_cycles(1000000);
      }
   }

   return 0;
}
```

Below is a serious makefile for MSP430 work. Note the use of symbols within make e.g. SOURCE =hworld.c and CFLAGS. All of this is intended to avoid memorizing complicated names and too much (inaccurate) typing. This makefile can be used by invoking

a) make
   to build the elf ,
b) make flash
   to flash to the board,
c) make debug
   to run gdb on the program on the msp430 … THERE IS A GDB VIDEO

Be sure to use the makefile in the repo and don't cut and paste from here.

```
# Project name
SOURCE          = hworld.c
ADDITIONAL      =
# Get base name so we can create .elf file
NAME            = $(basename $(SOURCE))
# MSP430 MCU to compile for
CPU             = msp430g2553
# Optimisation level
OPTIMIZATION    = -O0
# Extra variables
CFLAGS          = -mmcu=$(CPU) $(OPTIMIZATION) -std=c99 -Wall -g
# Libemb library link flags
LIBEMB          = -lshell -lconio -lserial

# Build and link executable
$(NAME).elf: $(SOURCE) $(ADDITIONAL)
ifeq (,$(findstring libemb,$(shell cat $(SOURCE))))
	msp430-gcc $(CFLAGS) -o $@ $(SOURCE) $(ADDITIONAL)
else
	msp430-gcc $(CFLAGS) -o $@ $(SOURCE) $(ADDITIONAL)
$(LIBEMB)
```

```
endif
	msp430-objdump -D $(NAME).elf > $(NAME).hex

# Flash to board with mspdebug
flash: $(NAME).elf
	mspdebug rf2500 "prog $(NAME).elf"

# Erase board
erase:
	mspdebug rf2500 erase

stuff: hworld.c
	echo "I found hworld.c!" && cat hworld.c

# Clean up temporary files
clean:
	rm -f *.elf *.hex

# Remote debug board
debug: $(NAME).elf
	( mspdebug rf2500 "gdb" 1>/dev/null & ); msp430-gdb $(NAME).elf -
ex "target remote :2000"
```