Spider Robot Starter Kit

Introduction

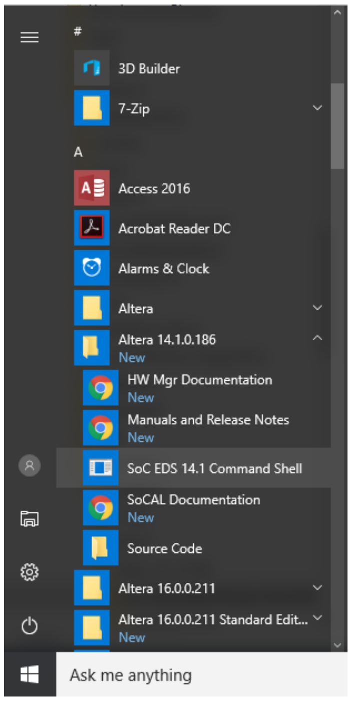
In short, there are modules in the FPGA that control the different parts of the Spider (legs, LEDs, pushbuttons, etc.) that are accessible through memory mapped registers (which we will use the embedded ARM processor in the FPGA.)

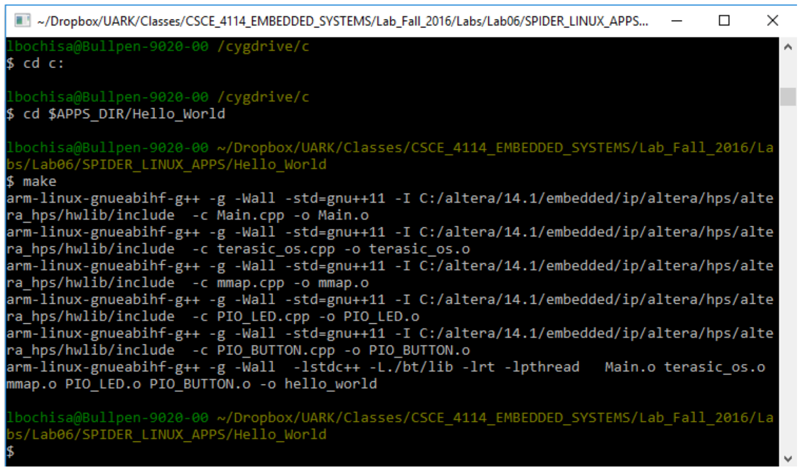
This platform requires the use of the SoC EDS 14.1 (or newer) Command Shell. Please refer to this link if you would like to install this development environment on your personal machine. However, this software is available on the workstations in the Embedded System Lab.

http://dl.altera.com/soceds/17.1/?edition=standard&platform=linux&download_manager=dlm3

Getting Started with C++ Development

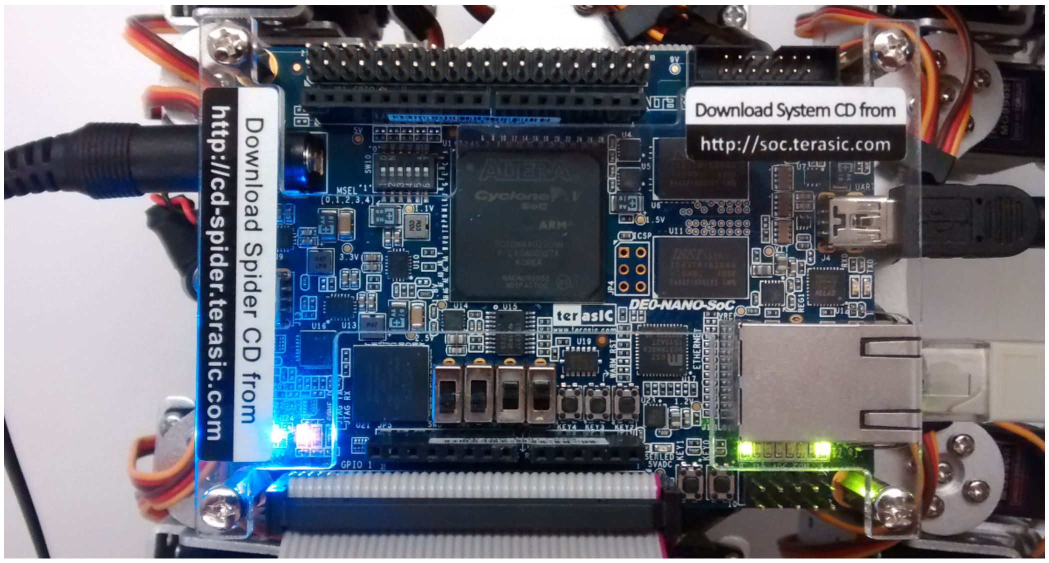
1. Open the application "SoC EDS 14.1 Command Shell".
2. On the command shell, browse to the SpiderRobotStartKit directory and type "make" to build the applications.



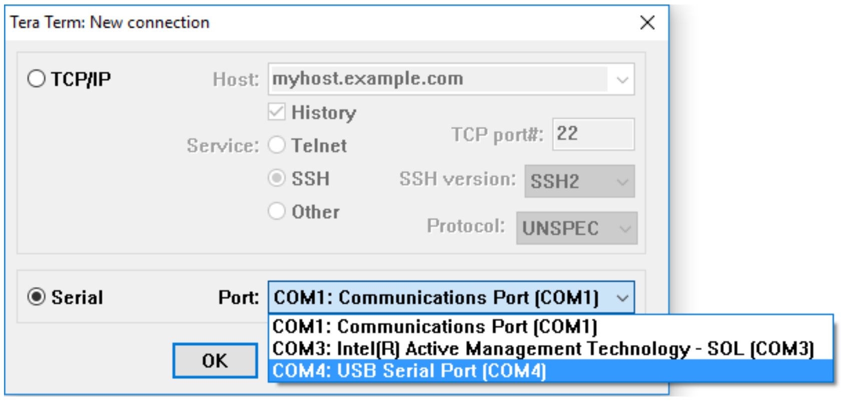

~/Dropbox/UARK/Classes/CSCE_4114_EMBEDDED_SYSTEMS/Lab_Fall_2016/Labs/Lab06/SPIDER_LINUX_APPS...

```
lbochisa@Bullpen-9020-00 /cygdrive/c
$ cd c:

lbochisa@Bullpen-9020-00 /cygdrive/c
$ cd $APPS_DIR/Hello_World

lbochisa@Bullpen-9020-00 ~/Dropbox/UARK/Classes/CSCE_4114_EMBEDDED_SYSTEMS/Lab_Fall_2016/La
bs/Lab06/SPIDER_LINUX_APPS/Hello_World
$ make
arm-linux-gnueabihf-g++ -g -Wall -std=gnu++11 -I C:/altera/14.1/embedded/ip/altera/hps/alte
ra_hps/hwlib/include  -c Main.cpp -o Main.o
arm-linux-gnueabihf-g++ -g -Wall -std=gnu++11 -I C:/altera/14.1/embedded/ip/altera/hps/alte
ra_hps/hwlib/include  -c terasic_os.cpp -o terasic_os.o
arm-linux-gnueabihf-g++ -g -Wall -std=gnu++11 -I C:/altera/14.1/embedded/ip/altera/hps/alte
ra_hps/hwlib/include  -c mmap.cpp -o mmap.o
arm-linux-gnueabihf-g++ -g -Wall -std=gnu++11 -I C:/altera/14.1/embedded/ip/altera/hps/alte
ra_hps/hwlib/include  -c PIO_LED.cpp -o PIO_LED.o
arm-linux-gnueabihf-g++ -g -Wall -std=gnu++11 -I C:/altera/14.1/embedded/ip/altera/hps/alte
ra_hps/hwlib/include  -c PIO_BUTTON.cpp -o PIO_BUTTON.o
arm-linux-gnueabihf-g++ -g -Wall  -lstdc++ -L./bt/lib -lrt -lpthread   Main.o terasic_os.o
mmap.o PIO_LED.o PIO_BUTTON.o -o hello_world

lbochisa@Bullpen-9020-00 ~/Dropbox/UARK/Classes/CSCE_4114_EMBEDDED_SYSTEMS/Lab_Fall_2016/La
bs/Lab06/SPIDER_LINUX_APPS/Hello_World
$
```

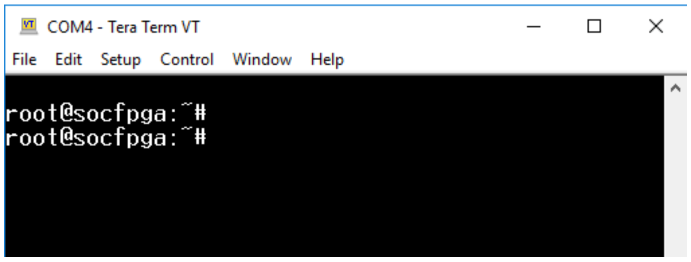
3. Grab a Spider, connect the USB cable in the interface marked "UART", connect the Ethernet cable and use the **small power adapter** to turn on the board. We will be connecting our Ethernet cables to the lab's floor ports. There are only a few open ports, so we will have to share.



4. Open the application "Tera Term". Select "Serial" and "COM4: USB Serial Port". Note that in your system, the COM number may be different.

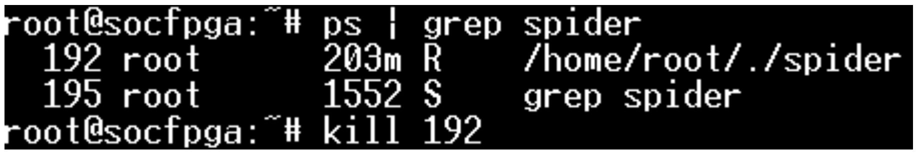

Tera Term: New connection
TCP/IP
Host: myhost.example.com
History
Service: Telnet
SSH
Other
TCP port#: 22
SSH version: SSH2
Protocol: UNSPEC
Serial
Port: COM1: Communications Port (COM1)
COM1: Communications Port (COM1)
COM3: Intel(R) Active Management Technology - SOL (COM3)
COM4: USB Serial Port (COM4)
OK

5. Click Setup->Serial Port... and select 115200 as the baud rate.
6. Restart the Spider by disconnecting and reconnecting the power supply. You should see the boot sequence of the OS on the Tera Term screen.
   You can make the font bigger by clicking on Setup->Font.
7. After the OS has finished booting, press ENTER a few times to see the terminal:



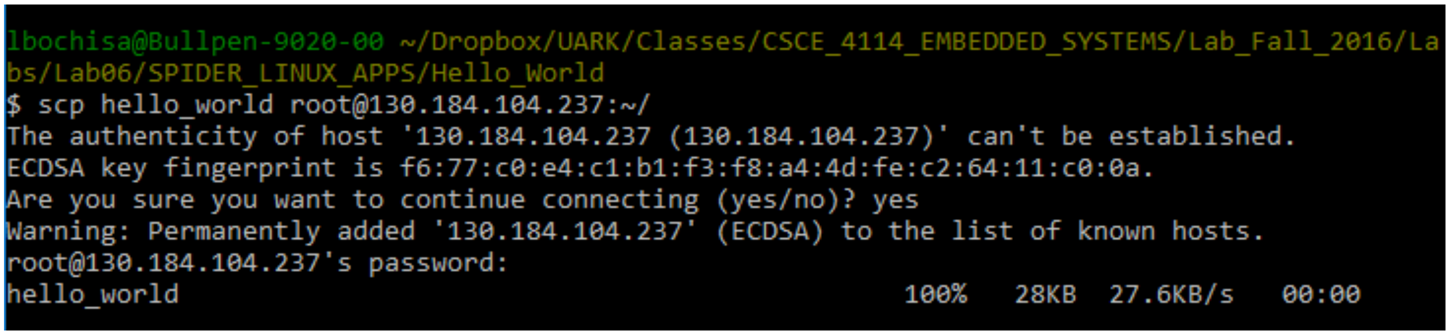
```
root@socfpga:~#
root@socfpga:~#
```

8. Change the permissions of the Spider application to no execution (to avoid interference with our app).



```
root@socfpga:~# ps | grep spider
  192 root       203m R    /home/root/./spider
  195 root       1552 S    grep spider
root@socfpga:~# kill 192
```

9. Acquire an IP address by typing "udhcpc". This step may have been performed upon booting
10. On the SoC EDS 14.1 Command Shell, use the command scp to copy the CustomSpiderProject application to the Spider (use the IP address returned by udhcpc). When prompted for a password, type "terasic".



```
lbochisa@Bullpen-9020-00 ~/Dropbox/UARK/Classes/CSCE_4114_EMBEDDED_SYSTEMS/Lab_Fall_2016/La
bs/Lab06/SPIDER_LINUX_APPS/Hello_World
$ scp hello_world root@130.184.104.237:~/
The authenticity of host '130.184.104.237 (130.184.104.237)' can't be established.
ECDSA key fingerprint is f6:77:c0:e4:c1:b1:f3:f8:a4:4d:fe:c2:64:11:c0:0a.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '130.184.104.237' (ECDSA) to the list of known hosts.
root@130.184.104.237's password:
hello_world                                         100%   28KB  27.6KB/s   00:00
```

11. Use the scp command shown above by navigating to the SpiderLinuxApplication and running with hello_world replaced with your CustomSpiderProject.
12. Run the app. On the Tera Term screen, type: ./CustomSpiderProject
    Does it work? (ignore the last LED, which is always blinking).

Custom Application

The Spider program code is located in the SpiderLinuxApplication directory. As you will be implementing a custom linux application, you will need to create a binary from your CPP code and transfer it to the Spider. Instead of using the 'Main.cpp' file for your code, I have created a 'Project.cpp' file that contains the essentials for starting your application the correct way. I've included all the header files from Main.cpp and added the ADC header file (See below). This will be built into binary file CustomSpiderProject. Its suggested to use Main.cpp as a reference when getting acquainted to this code base.

Utilizing the makefile, you only need to issue the 'make' command to compile your application. The above/below mentioned applications and the original Spider program are also built when issuing 'make'. Use 'make clean' to remove all compilation files.

The spider application is run after the system is booted up. Since the makefile will be making the CustomSpiderProject file for you to run your project, we don't need the spider application to run at boot. Simply change the permission of the spider binary file to not execute or just delete it. THIS IS IMPORTANT TO AVOID THE POWER QUIRK CAUSED BY THE LEGS.

Tips:

- File terasic_os.h contains useful functions to tell time.
    - OS_TicksPerSecond() returns the number of ticks in 1 second, and OS_GetTickCount() returns the current tick count of the processor. A frequency of 8 Hz, for example, has a period of OS_TicksPerSecond()/8 ticks.

- Use only 7 LEDs, since the last one is always blinking to demonstrate that the OS is running.
- The pushbuttons are '1' when **NOT** pressed and '0' when pressed.
- To test if pushbutton 0 was pressed, you can do:
    - If ((BUTTON_PIO.GetBUTTON() & 0x01) == 0)
- To test if pushbutton 1 was pressed, you can do:
    - If ((BUTTON_PIO.GetBUTTON() & 0x02) == 0)
- To test both pushbuttons were pressed, you can do:
    - If ((BUTTON_PIO.GetBUTTON() & 0x03) == 0)

SD Card

The SpiderSDCardFiles directory contains all the updated files for the Spider to utilize to boot its hardware, boot Linux, and then run the spider application. The SD cards supplied in the lab should all be formatted correctly, so just pop in the SD card and copy this directory's contents onto the mounted SD card partition. Typically, there will be only one partition mounted, though there are actually a couple partitions on the SD card, so if your machine doesn't make it obvious, we need to copy our files to the partition with similar files as those in this directory.

If we run out of SD cards or yours does not seem like the structure described above, let me know: Taylor Whitaker <txw043@uark.edu>.

Spider Controller Program

The 'Controller' binary file was created to demonstrate some of the actions you can do with Spider's legs. Primarily, it was created for folding the Spider's leg for easy storage in their boxes. Simply run the binary supplied in the Binaries directory to see the commands.

I have supplied the code for this program. You may use any part you wish, though I would strongly suggest taking advantage of the fold function I implemented in the 'CSpider.cpp' file.

ADC for Sensors

(Probably not needed for Mobile Autonomous Robots class, but its already here)

In order to use the ADC sensors for your projects, I have updated the Spider hardware so that the sensors can be accessed from the Linux OS on the processor. I have also added and ADC class to the Spider application folder. This class allows you to read a particular channel from the ADC pins. There is also an ADCTest program that you can use to understand the readings the sensor will provide.

Quirks

The spider robot is awesome but has its downfalls. Here is a running list of things you might run into.

Spider loses power when stretching legs / Leg stretching is super slow.

This occurs when any application that uses the spider's legs is run more than once. The way they mapped hardware into the software we are using causes weird behavior if any program calls the spider initialization function, which the CustomSpiderProject currently does and should be left that way.

In other words, if you have to re-run your program, you should reboot first.

As mentioned above, this can happen if the spider application is run on boot. If you were to run your program after stopping the spider app, you're going to hit this problem. Simply stop the spider app from executing or delete it all together.