WES 237A: Introduction to Embedded System Design (Winter 2026)
Lab 3: Serial and CPU
Due: 2/1/2026 11:59pm

In order to report and reflect on your WES 237A labs, please complete this Post-Lab report by the end of the weekend by submitting the following 2 parts:

- Upload your lab 3 report composed by a single PDF that includes your in-lab answers to the bolded questions in the Google Doc Lab and your Jupyter Notebook code. You could either scan your written copy, or simply type your answer in this Google Doc. **However, please make sure your responses are readable.**
- Answer two short essay-like questions on your Lab experience.

All responses should be submitted to Canvas. Please also be sure to push your code to your git repo as well.

Serial Connection

- Using a micro USB cable, connect your board to your laptop
- Connect to board using the serial connection
    - Linux
        - Open a new terminal
        - Run the command
            - *sudo screen /dev/<port> 115200* #port: ttyUSB0 or ttyUSB1
    - MAC
        - Open a new terminal
        - Run the command and check the PYNQ resources for the port
            - *sudo screen /dev/<port> 115200* #port: check resources
    - Windows
        - Check the resource for how to connect through serial to the PYNQ board
    - Resources:
        - https://pynq.readthedocs.io/en/v2.0/getting_started.html
        - https://www.nengo.ai/nengo-pynq/connect.html
- After connecting
    - Restart the board (*$ sudo reboot*)
    - Interrupt the boot (keyboard interrupt)
    - List current settings (*printenv*)
    - **Put a screenshot of your *$ printenv* output**

```
{soc}-${board}${boardver}.dtb; fi; setenv efi_old_vci ${bootp_vci};setenv efi_old_arch ${bootp_arch};setenv bo
otp_vci PXEClient:Arch:00010:UNDI:003000;setenv bootp_arch 0xa;if dhcp ${kernel_addr_r}; then tftpboot ${fdt_a
ddr_r} dtb/${efi_fdtfile};if fdt addr -q ${fdt_addr_r}; then bootefi ${kernel_addr_r} ${fdt_addr_r}; else boot
efi ${kernel_addr_r} ${fdtcontroladdr};fi;fi;setenv bootp_vci ${efi_old_vci};setenv bootp_arch ${efi_old_arch}
;setenv efi_fdtfile;setenv efi_old_arch;setenv efi_old_vci;
boot; run mmc_boot
bootcmd_nand=nand info && nand read ${scriptaddr} ${script_offset_f} ${script_size_f} && echo NAND: Trying to
boot script at ${scriptaddr} && source ${scriptaddr}; echo NAND: SCRIPT FAILED: continuing...;
bootcmd_nor=cp.b ${script_offset_nor} ${scriptaddr} ${script_size_f} && echo NOR: Trying to boot script at ${s
criptaddr} && source ${scriptaddr}; echo NOR: SCRIPT FAILED: continuing...;
bootcmd_pxe=run boot_net_usb_start; dhcp; if pxe get; then pxe boot; fi
bootcmd_qspi=sf probe 0 0 0 && sf read ${scriptaddr} ${script_offset_f} ${script_size_f} && echo QSPI: Trying
to boot script at ${scriptaddr} && source ${scriptaddr}; echo QSPI: SCRIPT FAILED: continuing...;
bootcmd_usb0=devnum=0; run usb_boot
bootcmd_usb1=devnum=1; run usb_boot
bootcmd_usb_dfu0=setenv dfu_alt_info boot.scr ram $scriptaddr $script_size_f && dfu 0 ram 0 60 && echo DFU0: T
rying to boot script at ${scriptaddr} && sourscript at ${scriptaddr} && source ${scriptaddr}; echo DFU1: SCRIP
T FAILED: continuing...;
bootcmd_usb_thor0=setenv dfu_alt_info boot.scr ram $scriptaddr $script_size_f && thordown 0 ram 0 && echo THOR
0: Trying to boot script at ${scriptaddr} && source ${scriptaddr}; echo THOR0: SCRIPT FAILED: continuing...;
bootcmd_usb_thor1=setenv dfu_alt_info boot.scr ram $scriptaddr $script_size_f && thordown 1 ram 1 && echo THOR
1: Trying to boot script at ${scriptaddr} && source ${scriptaddr}; echo THOR1: SCRIPT FAILED: continuing...;
bootdelay=2
bootm_low=0
bootm_size=20000000
cpu=armv7
distro_bootcmd=for target in ${boot_targets}; do run bootcmd_${target}; done
efi_dtb_prefixes=/ /dtb/ /dtb/current/
fdt_addr_r=0x1f00000
fdtcon; then devtype=mmc; run scan_dev_for_boot_part; fi
modeboot=sdboot
pxefile_addr_r=0x2000000
ramdisk_addr_r=0x3100000
scan_dev_for_boot=echo Scanning ${devtype} ${devnum}:${distro_bootpart}...; for prefix in ${boot_prefixes}; do
 run scan_dev_for_extlinux; run scan_dev_for_scripts; done;run scan_dev_for_efi;
scan_dev_for_boot_part=part list ${devtype} ${devnum} -bootable devplist; env exists devplist || setenv devpli
st 1; for distro_bootpart in ${devplist}; do if fstype ${devtype} ${devnum}:${distro_bootpart} bootfstype; the
n part uuid ${devtype} ${devnum}:${distro_bootpart} distro_bootpart_uuid ; run scan_dev_for_boot; fi; done; se
tenv devplist
scan_dev_for_efi=setenv efi_fdtfile ${fdtfile}; if test -z "${fdtfile}" -a -n "${soc}"; then setenv efi_fdtfil
e ${soc}-${board}${boardver}.dtb; fi; for prefix in ${efi_dtb_prefixes}; do if test -e ${devtype} ${devnum}:${
distro_bootpart} ${prefix}${efi_fdtfile}; then run load_efi_dtb; fi;done;run boot_efi_bootmgr;if test -e ${dev
type} ${devnum}:${distro_bootpart} efi/boot/bootarm.efi; then echo Found EFI removable media binary efi/boot/b
ootarm.efi; run boot_efi_binary; echo EFI LOAD FAILED: continuing...; fi; setenv efi_fdtfile
scan_dev_for_extlinux=if test -e ${devtype} ${devnum}:${distro_bootpart} ${prefix}${boot_syslinux_conf}; then
echo Found ${prefix}${boot_syslinux_conf}; run boot_extlinux; echo EXTLINUX FAILED: continuing...; fi
scan_dev_for_scripts=for script in ${boot_scripts}; do if test -e ${devtype} ${devnum}:${distro_bootpart} ${pr
efix}${script}; then echo Found U-Boot script ${prefix}${script}; run boot_a_script; echo SCRIPT FAILED: conti
nuing...; fi; done
script_offset_f=fc0000
script_offset_nor=0xE2FC0000
script_size_f=0x40000
scriptaddr=3000000
soc=zynq
stderr=serial@e0000000
stdin=serial@e0000000
stdout=serial@e0000000
ubifs_boot=if ubi part ${bootubipart} ${bootubioff} && ubifsmount ubi0:${bootubivol}; then devtype=ubi; devnum
=ubi0; bootfstype=ubifs; distro_bootpart=${bootubivol}; run scan_dev_for_boot; ubifsumount; fi
usb_boot=usb start; if usb dev ${devnum}; then devtype=usb; run scan_dev_for_boot_part; fi
vendor=xilinx

Environment size: 5887/131067 bytes
```

Change Bootargs

- *If you need to return to the default bootargs, you can find them below*

    - https://github.com/Xilinx/PYNQ/blob/master/sdbuild/boot/meta-pynq/recipes-bsp/device-tree/files/pynq_bootargs.dtsi
    - *bootargs = 'root=/dev/mmcblk0p2 rw earlyprintk rootfstype=ext4 rootwait devtmpfs.mount=1 uio_pdrv_genirq.of_id="generic-uio" clk_ignore_unused'*
- To edit bootargs:
    - Interrupt the boot
    - Edit boot arguments:
        - *$ editenv bootargs*
        - Insert arguments included the quotations all in one line:
            - Bootargs (default and more) are at here
        - *$ boot*
- Change bootargs to the following
    - *bootargs = 'console=ttyPS0,115200 root=/dev/mmcblk0p2 rw earlyprintk rootfstype=ext4 rootwait devtmpfs.mount=1 uio_pdrv_genirq.of_id="generic-uio" clk_ignore_unused* ***isolcpus=1*** *&& bootz 0x03000000 - 0x02A00000'*
    - **What does isolcpus=1 do?**

It prevents new threads from moving onto CPU Core 1.

    - **What would isolcpus=0 do?**

It prevents new threads from moving onto CPU Core 0.

Heavy CPU Utilization

- Download *fib.py* from here. This is a recursive implementation for generating Fibonacci sequences. We just do not print the results.
- Jupyter notebook is hosted at: /home/xilinx/jupyter_notebooks
- Make sure your board is booted with custom bootargs above, including *isolcpus=1*

1) Open two terminals (Jupyter):
    - Terminal 1: run *htop* to monitor CPU utilization
    - Terminal 2: run *$ python3 fib.py* and monitor CPU utilization and time spent for running the script (set terms to lower than 40)
    - **Describe the results of *htop*.**

On the core that ran fib.py usage spiked up to 100% until the program was done.

2) Repeat the previous part, but this time use *taskset* to use CPU1:
    - Terminal 2: run *$ taskset -c 1 python3 fib.py* and monitor CPU utilization and time spent for running the script
    - **Describe the results of *htop*. Specifically, what's different from running it in 1)?**

Similar thing happened with the previous step, except this time it was happening on CPU Core 1.

3) Heavy Utilization on CPU0:
    - Open another terminal and run $ *dd if=/dev/zero of=/dev/null*
    - Repeat parts 1 and 2
    - **Describe the results of *htop*.**

This time, both CPU cores were being utilized at 100%.

Jupyter Notebook CPU Monitoring

Download CPU_monitor.ipynb from here. This is an interactive implementation for plotting in a loop. Running this notebook is a computationally heavy task for your CPU, therefore you do not need to run any additional process to utilize your CPU0.

- Create a Jupyter notebook
    - Use the *os* library to create a Python program that accepts a number from user input (0 or 1) and runs *fib.py* on a specific core (0 or 1).
    - *Hint: look at the os.system() call and remember the 'taskset' function we've used previously.*
- You should have two notebooks running: 1) CPU_monitor, 2) CPU_select
    - **Compare your observations between using Jupyter notebook CPU_monitor and linux command *htop* for monitoring CPU utilization.**

The reported usage between CPU_monitor and htop were similar. The main difference was that they were updating at different intervals, so they didn't report the same exact utilization.

ARM Performance Monitoring (C++)

- Download kernel_modules folder
- Read through CPUcntr.c and reference the ARM documentation for the PMU registers here to answer the following question.
    - **According to the ARM docs, what does the following line do? Are they written in assembly code, python, C, or C++?**
        - *asm("MCR p15, 0, 1, c9, c14, 0\n\t");*

"User Enable Register." This is written in C.

    - Compile and insert the kernel module following the instructions from the README file.
- Download clock_example folder
- Read through *include/cycletime.h* and take note of the functions to initialize the counters and get the cyclecount (what datatype do they return, what parameters do they take)
    - **What does the following line do?**
        - *asm volatile ("MRC p15, 0, %0, c9, c13, 0\n\t" : "=r"(value));*

This gets the cyclecount which will be used to calculate the time elapsed.

- Complete the code in *src/main.cpp.* These instructions are for those who have never coded in C++
    - **Declare 2 variables (cpu_before, cpu_after) of the correct datatype**
    - **Initialize the counter**
    - **Get the cyclecount 'before' sleeping**
    - **Get the cyclecount 'after' sleeping**
    - **Print the difference number of counts between starting and stopping the counter**
- After completing the code, open a jupyter terminal and change directory to *clock_examples/*
- Run *$ make* to compile the code
- Run the code with *$ ./lab3 <delay-time-seconds>*
- **Change the delay time and note down the different cpu cycles as well as the different timers.**

Delay time of 1 gave a cycle count of approx. 2000, Timer ~= 9.5e-05; Delay time of 50 gave a cycle count of approx. 1900, Timer ~= 0.00014

In [1]:
```
import time
import pylab as pl
from IPython import display
import psutil
import matplotlib.pyplot as plt
import numpy as np
import os
```

In [11]:
```
# Program to pick CPU core
coreID = int(input("Which core to use? "))
param = int(input("How many terms? ")) # modified fib.py to pass term param

cmd = "taskset -c " + str(coreID) + " python3 fib.py " + str(param)
os.system(cmd)
```

```
Which core to use? 1
How many terms? 36
time spent: 81.00165748596191
```

Out[11]:
```
0
```

In [ ]:
```

```

In [1]:
```
import time
import pylab as pl
from IPython import display
import psutil
import matplotlib.pyplot as plt
import numpy as np
```

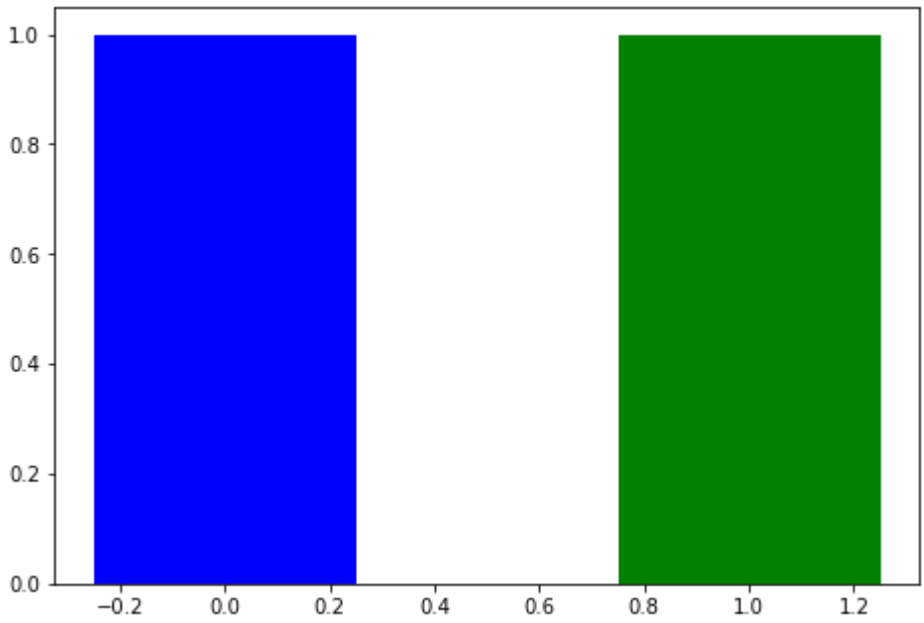
```
Matplotlib is building the font cache; this may take a moment.
```

In [10]:
```
psutil.cpu_percent(percpu=True)
```

Out[10]:
```
[10.1, 10.1]
```

In [13]:
```
%matplotlib inline

X = np.arange(1)
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
for i in range(30):
    data = psutil.cpu_percent(percpu=True)
    ax.cla()
    ax.bar(X + 0.0, data[0]/100, color = 'b', width = 0.5)
    ax.bar(X + 1.0, data[1]/100, color = 'g', width = 0.5)
    display.clear_output(wait=True)
    display.display(plt.gcf())
plt.clf()
```




```
<Figure size 432x288 with 0 Axes>
```

In [ ]: