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

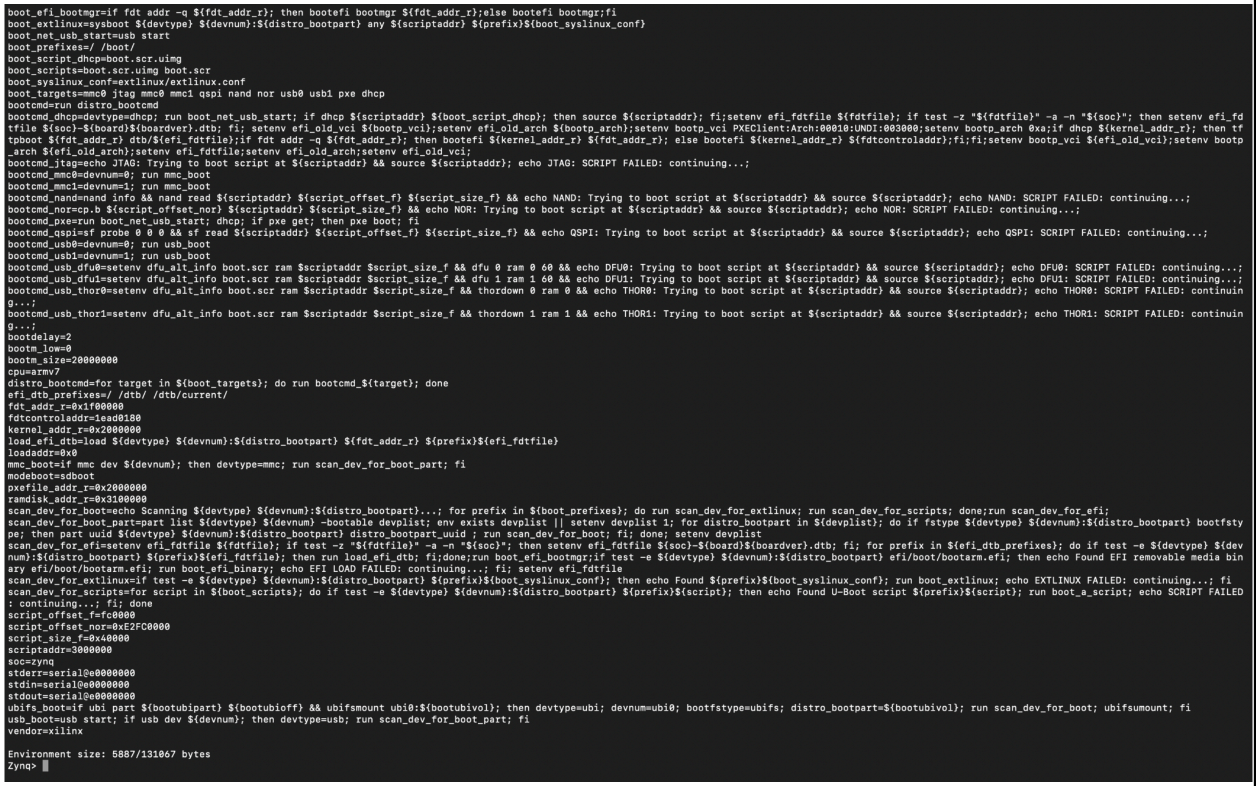
In order to report and reflect on your WES 237A labs, please complete this Post-Lab report by the end of the weekend by submitting the following 2 parts:

- 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**



```
boot_efi_bootmgr=if fdt addr -q ${fdt_addr_r}; then bootefi bootmgr ${fdt_addr_r};else bootefi bootmgr;fi
boot_extlinux=sysboot ${devtype} ${devnum}:${distro_bootpart} any ${scriptaddr} ${prefix}${boot_syslinux_conf}
boot_net_usb_start=usb start
boot_prefixes=/ /boot/
boot_script_dhcp=boot.scr.uimg
boot_scripts=boot.scr.uimg boot.scr
boot_syslinux_conf=extlinux/extlinux.conf
boot_targets=mmc0 jtag mmc0 mmc1 qspi nand nor usb0 usb1 pxe dhcp
bootcmd=run distro_bootcmd
bootcmd_dhcp=devtype=dhcp; run boot_net_usb_start; if dhcp ${scriptaddr} ${boot_script_dhcp}; then source ${scriptaddr}; fi;setenv efi_fdtfile ${fdtfile}; if test -z "${fdtfile}" -a -n "${soc}"; then setenv efi_fd
tfile ${soc}-${board}${boardver}.dtb; fi; setenv efi_old_vci ${bootp_vci};setenv efi_old_arch ${bootp_arch};setenv bootp_vci PXEClient:Arch:00010:UNDI:003000;setenv bootp_arch 0xa;if dhcp ${kernel_addr_r}; then tf
tpboot ${fdt_addr_r} dtb/${efi_fdtfile};if fdt addr -q ${fdt_addr_r}; then bootefi ${kernel_addr_r} ${fdt_addr_r}; else bootefi ${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;
bootcmd_jtag=echo JTAG: Trying to boot script at ${scriptaddr} && source ${scriptaddr}; echo JTAG: SCRIPT FAILED: continuing...;
bootcmd_mmc0=devnum=0; run mmc_boot
bootcmd_mmc1=devnum=1; 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 ${scriptaddr} && 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: Trying to boot script at ${scriptaddr} && source ${scriptaddr}; echo DFU0: SCRIPT FAILED: continuing...;
bootcmd_usb_dfu1=setenv dfu_alt_info boot.scr ram $scriptaddr $script_size_f && dfu 1 ram 1 60 && echo DFU1: Trying to boot script at ${scriptaddr} && source ${scriptaddr}; echo DFU1: SCRIPT FAILED: continuing...;
bootcmd_usb_thor0=setenv dfu_alt_info boot.scr ram $scriptaddr $script_size_f && thordown 0 ram 0 && echo THOR0: Trying to boot script at ${scriptaddr} && source ${scriptaddr}; echo THOR0: SCRIPT FAILED: continuin
g...;
bootcmd_usb_thor1=setenv dfu_alt_info boot.scr ram $scriptaddr $script_size_f && thordown 1 ram 1 && echo THOR1: Trying to boot script at ${scriptaddr} && source ${scriptaddr}; echo THOR1: SCRIPT FAILED: continuin
g...;
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
fdtcontroladdr=1ead0180
kernel_addr_r=0x2000000
load_efi_dtb=load ${devtype} ${devnum}:${distro_bootpart} ${fdt_addr_r} ${prefix}${efi_fdtfile}
loadaddr=0x0
mmc_boot=if mmc dev ${devnum}; 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 devplist 1; for distro_bootpart in ${devplist}; do if fstype ${devtype} ${devnum}:${distro_bootpart} bootfsty
pe; then part uuid ${devtype} ${devnum}:${distro_bootpart} distro_bootpart_uuid ; run scan_dev_for_boot; fi; done; setenv devplist
scan_dev_for_efi=setenv efi_fdtfile ${fdtfile}; if test -z "${fdtfile}" -a -n "${soc}"; then setenv efi_fdtfile ${soc}-${board}${boardver}.dtb; fi; for prefix in ${efi_dtb_prefixes}; do if test -e ${devtype} ${dev
num}:${distro_bootpart} ${prefix}${efi_fdtfile}; then run load_efi_dtb; fi;done;run boot_efi_bootmgr;if test -e ${devtype} ${devnum}:${distro_bootpart} efi/boot/bootarm.efi; then echo Found EFI removable media bin
ary efi/boot/bootarm.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} ${prefix}${script}; then echo Found U-Boot script ${prefix}${script}; run boot_a_script; echo SCRIPT FAILED
: continuing...; 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
Zynq>
```

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 disables CPU1 and only lets CPU0 run. htop shows 0.0% utilization of CPU1 at all times.

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

Both the CPUs are enabled. There is no CPU isolation.

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*.**

With isolcpus=1, htop shows CPU1 at 0% utilization.
CPU0 goes briefly to **high** utilization if the number of terms of Fibonacci sequence to be calculated is **less** than 29.
CPU0 goes briefly to **max** utilization if the number of terms of Fibonacci sequence to be calculated is **more** than 29.

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)?**

htop shows max/high utilization for CPU1 (depending on the input number of terms as described above).
With the -c 1 argument, the process running fib.py processor affinity is set to CPU1.

What's different from running it in 1) is that CPU1 is utilized in this case (despite booting up with isolcpus=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*.**

CPU0 stays at 100% utilization after running the dd command - regardless of whether the fib.py script is run or not .

For 1), the calculation of Fibonacci sequence number takes much longer for the same number of terms compared to earlier (tried with nterms = 30).
For 2), the calculation takes about the same time as earlier.

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.**

htop is far more responsive to instantaneous changes in CPU utilization than CPU_monitor (using psutil).

Due to the for loop implementation, CPU_monitor only runs for about ~16 seconds and hence, can track CPU utilization only for this time period.
The Fibonacci script takes longer if the number of terms is 33 or greater.
htop runs until killed.

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");*

This is inline ARM assembly code, within the C file.
MCR instruction moves a specified value to the coprocessor from an ARM register.

```
MCR{cond} coproc, #opcode1, Rt, CRn, CRm{, #opcode2}
```

`coproc = p15` selects the ARM PMU.
`opcode1 = 0` and `opcode2 = 0`, along with `c9` & `c14` selects the coprocessor register PMUSERENR.
The value written to PMUSERENR is 1.

According to the documentation at https://developer.arm.com/documentation/111107/2025-12/AArch32-Registers/PMUSERENR--Performance-Monitors-User-Enable-Register, this enables or disables EL0 access to the Performance Monitors.

HLOS runs at EL0 (Exception Level 0).

    - 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));*

MRC instruction moves a specified value to an ARM register from the coprocessor.

```
MRC{cond} coproc, #opcode1, Rt, CRn, CRm{, #opcode2}
asm volatile ("MRC p15, 0, %0, c9, c13, 0\n\t" : "=r"(value));
```

`coproc = p15` selects the ARM PMU.
`opcode1 = 0` and `opcode2 = 0`, along with `c9` & `c13` selects the coprocessor register PMCCNTR to be copied to the "value" variable.

Register number	Offset	CRn	Op1	CRm	Op2	Name	Type	Description

31	0x07C	c9	0	c13	0	PMCCNTR	RW	Cycle Count Register, see the *ARM Architecture Reference Manual, ARMv7-A and ARMv7-R edition*

- 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 input arg (usec)	CPU cycles	Delay time
1000	97538	0.00111158
1500	1170487	0.00159306
3000	1393691	0.00308837
500000	15309755	0.500101
1000000	15758519	1.0001

In [11]:
```python
import os

cpu_num = int(input("Enter CPU to run fib.py on (0 or 1): "))
assert(cpu_num == 0 or cpu_num == 1)

nterms = int(input("Enter number of terms (< 40 for sanity): "))

os.system(f"taskset -c {cpu_num} python3 fib_modified.py {nterms}")
```

```
Enter CPU to run fib.py on (0 or 1): 1
Enter number of terms (< 40 for sanity): 33
time spent: 18.80435037612915
```

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

In [2]:
```python
import subprocess
import os, sys

cpu_num = int(input("Enter CPU to run fib.py on (0 or 1): "))
assert(cpu_num == 0 or cpu_num == 1)

os.system(f"taskset -c {cpu_num} python3 fib.py")
```

```
Enter CPU to run fib.py on (0 or 1): 1
How many terms?
Traceback (most recent call last):
  File "/home/xilinx/jupyter_notebooks/Lab3/fib.py", line 6, in <module>
    nterms = int(input("How many terms? "))
EOFError: EOF when reading a line
```

Out[2]:
```
256
```

In [ ]:
```

```

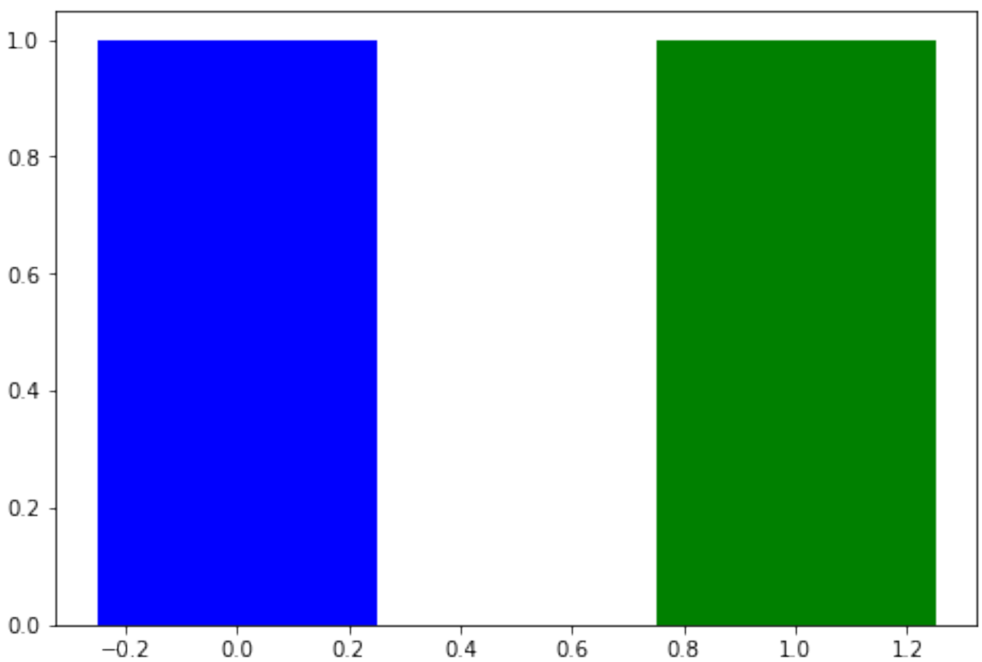
In [1]:
```
import time
import pylab as pl
from IPython import display
import psutil
import matplotlib.pyplot as plt
import numpy as np
```

In [2]:
```
psutil.cpu_percent(percpu=True)
```

Out[2]:
```
[6.9, 0.1]
```

In [8]:
```
%matplotlib inline

X = np.arange(1)
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
tic = time.time()
for i in range(10):
    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()
print(f"Time taken: {time.time() - tic}")
```




```
Time taken: 16.24588680267334
<Figure size 432x288 with 0 Axes>
```

In [ ]:

```
```