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

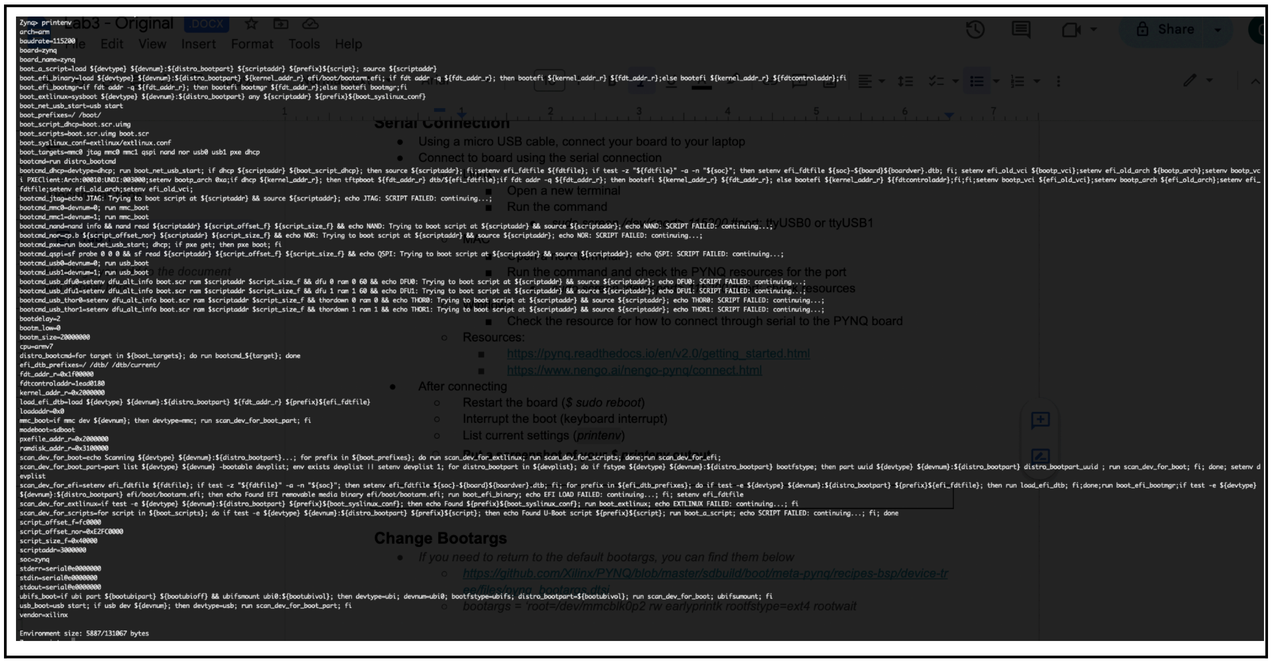
In order to report and reflect on your WES 237A labs, please complete this Post-Lab report by the end of the weekend by submitting the following 2 parts:

- 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**



```
Zynq> printenv
arch=arm
baudrate=115200
board=zynq
board_name=zynq
boot_a_script=load ${devtype} ${devnum}:${distro_bootpart} ${scriptaddr} ${prefix}${script}; source ${scriptaddr}
boot_efi_binary=load ${devtype} ${devnum}:${distro_bootpart} ${kernel_addr_r} efi/boot/bootarm.efi; if fdt addr -q ${fdt_addr_r}; then bootefi ${kernel_addr_r} ${fdt_addr_r};else bootefi ${kernel_addr_r} ${fdtcontroladdr};fi
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_fdtfile ${soc}-${board}${boardver}.dtb; fi; setenv efi_old_vci ${bootp_vci};setenv efi_old_arch ${bootp_arch};setenv bootp_vc
i PXEClient:Arch:00010:UNDI:003000;setenv bootp_arch 0xa;if dhcp ${kernel_addr_r}; then tftpboot ${fdt_addr_r} dtb/${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: continuing...;
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: 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
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} bootfstype; then part uuid ${devtype} ${devnum}:${distro_bootpart} distro_bootpart_uuid ; run scan_dev_for_boot; fi; done; setenv d
evplist
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} ${devnum}:${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 binary 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
```

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 says that the kernel processes scheduler will not interact with CPU1, which is the second CPU (starts at 0)

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

The scheduler will not place tasks on CPU0 unless explicitly told to do so

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*.**

CPU0 usage jumped to 100% while CPU1 usage stayed at 0%

```
root@pynq:/home/xilinx/jupyter_notebooks# python3 fib.py
How many terms? 30
time spent: 4.820886850357056
```

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)?**

CPU1 usage jumped to 100%. CPU0 usage went up to around 16% but stayed relatively low.

```
root@pynq:/home/xilinx/jupyter_notebooks# taskset -c 1 python3 fib.py
How many terms? 30
time spent: 4.477916479110718
```

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*.**

Running on CPU0, CPU0 usage stayed at 100% before, during, after the program. CPU1 stayed at 0%. The time taken to run the code more than doubled as well.

```
root@pynq:/home/xilinx/jupyter_notebooks# python3 fib.py
How many terms? 30
time spent: 9.409242630004883
```

Running on CPU1, CPU0 usage stayed at 100% because of the dd command. CPU1 went up to 100% because the program was running. The time taken to run the program stayed the same as before.

```
root@pynq:/home/xilinx/jupyter_notebooks# taskset -c 1 python3 fib.py
How many terms? 30
time spent: 4.491708993911743
```

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 Jupyter notebook scales the axis so it looks like CPU0 is always at 100% if you don't read the units. Other than that, the htop command updates a little more and gives exact percentage values for the CPU usage. Also, the Jupyter notebook uses CPU0 to run. So no matter which CPU you run fib.py, CPU0 usage will be at 100% during monitoring.

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");*

The code is written in C and executes assembly code.

The code copies the value of a general-purpose register to a System register

It chooses coprocessor 1 (p15) to run on, takes the value from register 1, and puts it into the PMUSERENR register.

reg	898
addr	0xE08
CRn	c9
op1	0
CRm	c14
op2	0
name	PMUSERENR

  - 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));*

The code copies the value of a System register to a general purpose register.

It chooses coprocessor 15 (p15) to run on, takes the value from the PMXEVCNTR register, and puts it into the value register.

reg	PMXEVCNTR
addr	implementation-defined
CRn	c9
op1	0
CRm	c13

```
op2     0
name    PMXEVCNTR
```

- 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.**

```
1
CPU cycles: 99585
Timer: 0.00016024

5
CPU cycles: 101977
Timer: 0.000161705

10
CPU cycles: 97833
Timer: 0.000168942

100
CPU cycles: 319681
Timer: 0.000189551

1000
CPU cycles: 409848
Timer: 0.00108908

10000
CPU cycles: 213402
Timer: 0.010088
```

In [4]:
```
import os
```

In [13]:
```
cpu = int(input("Choose CPU to run fib.py on. (0, 1)"))
nterms = int(input("How many terms? "))

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

```
Choose CPU to run fib.py on. (0, 1)1
How many terms? 32
time spent: 11.616465330123901
```

Out[13]:
```
0
```

In [ ]:

```python
#! /usr/bin/python

import time
import sys, getopt
# Program to calculate the Fibonacci sequence up to n-th term
nterms = int(sys.argv[1])

def recur_fibo(n):
   if n <= 1:
       return n
   else:
       return(recur_fibo(n-1) + recur_fibo(n-2))

tic = time.time()

# check if the number of terms is valid
if nterms <= 0:
   print("Please enter a positive integer")
else:
   recur_fibo(nterms)

tac = time.time()
print('time spent: {}'.format(tac-tic))

```

```cpp

    // 1 argument on command line: delay = arg
    if(argc >= 2)
    {
        delay = atoi(argv[1]);
    }

    //TODO: declare 2 cpu_count variables: 1 for before sleeping, 1 for after sleeping (see cpu_timer)
    unsigned int cpu_before;
    unsigned int cpu_after;
    unsigned int cpu_diff;

    //TODO: initialize the counter
    init_counters(1, 0);

    //TODO: get the cyclecount before sleeping
    cpu_before = get_cyclecount();

    usleep(delay);

    //TODO: get the cyclecount after sleeping
    cpu_after = get_cyclecount();

    //TODO: subtract the before and after cyclecount
    cpu_diff = cpu_after - cpu_before;

    //TODO: print the cycle count (see the print statement for the cpu_timer below)
    cout << "CPU cycles: " << cpu_diff << endl;

    LinuxTimer t;
    usleep(delay);
    t.stop();
    cpu_timer = t.getElapsed();

```