WES 237A

Lab 3

What is in the SDCard?

- Two partitions
    - Boot (~100MB)
    - Filesystem (5GB<)

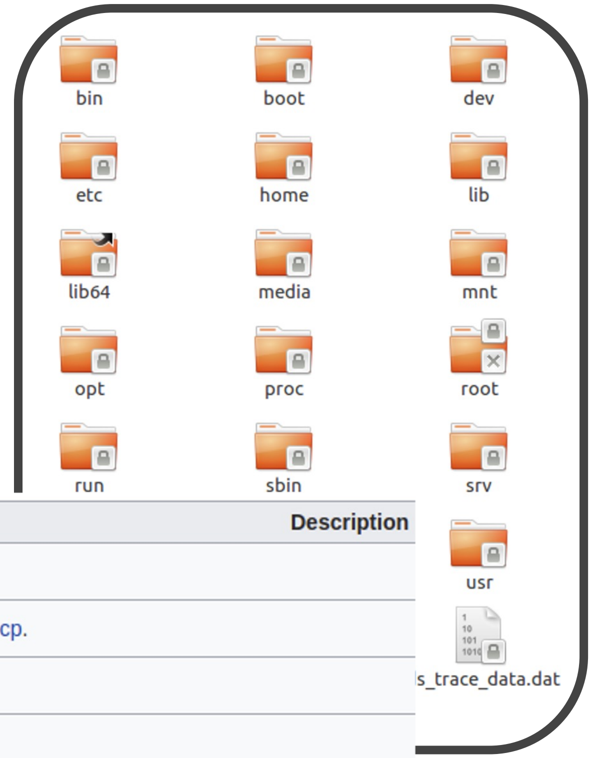
filesystem partition

Boot partition

Filesystem Partition

- (5GB<)
  - Linux standard filesystem hierarchy
  - A good summary: https://en.wikipedia.org/wiki/Filesystem_Hierarchy_Standard

filesystem partition



Directory	Description
`/`	*Primary hierarchy* root and root directory of the entire file system hierarchy.
`/bin`	Essential command binaries that need to be available in single user mode; for all users, *e.g.*, cat, ls, cp.
`/boot`	Boot loader files, *e.g.*, kernels, initrd.
`/dev`	Device files, *e.g.*, `/dev/null`, `/dev/disk0`, `/dev/sda1`, `/dev/tty`, `/dev/random`.

Boot Partition

- (~100MB)
  - BOOT.BIN (657 kB)
  - Image.ub (4.8 MB)
- Boot process
  - ROM
    - Contains code that execute right after reset or power-on
  - Secondary Program Loader
    - Configures the memory controller and other components
  - Tertiary Program Loader
    - Contains full bootloader and allows user interaction
  - Kernel

Boot partition

Source: https://www.xilinx.com/products/design-tools/embedded-software/petalinux-sdk.html

Boot Partition

- (~100MB)
  - BOOT.BIN (657 kB)
    - First-stage boot loader (FSBL) image
    - U-boot boot loader
    - FPGA bitstream
    - Power management unit (PMU) firmware
    - ARM Trusted Firmware image
  - Image.ub (4.8 MB)
    - Linux kernel image
    - Device-tree blob (DTB)
    - Root filesystem image

Boot partition

Source: https://www.xilinx.com/products/design-tools/embedded-software/petalinux-sdk.html
https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842374/U-Boot+Images
https://github.com/Xilinx/u-boot-xlnx

Boot Partition

- (~100MB)
  - BOOT.BIN (657 kB)
    - First-stage boot loader (FSBL) image
    - U-boot boot loader
    - FPGA bitstream
    - Power management unit (PMU) firmware
    - ARM Trusted Firmware image
  - Image.ub (4.8 MB)
    - Linux kernel image
    - Device-tree blob (DTB)
    - Root filesystem image

Boot partition

Source: https://www.xilinx.com/products/design-tools/embedded-software/petalinux-sdk.html

Boot Loader

- Load OS & runtime environment into memory.
- Key Components:
  - BIOS
  - Firmware found in the ROM

Boot Loaders

Name	License	Can reside in				Can boot from								Can boot								
		ESP (UEFI)	MBR	VBR	Floppy	Hard disk	Second Hard disk	Logical partitions	CD-ROM	Floppy	USB	Zip	LAN	MS-DOS	Windows 9x/Me	Windows NT series	Windows Vista/7/8/10	Linux	ReactOS	MenuetOS	*BSD	Mac OS X
Acronis OS Selector	Proprietary	?	?	?	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	?	Yes	Yes	Yes	Yes	Yes	?	?	Yes	Yes
AiR-Boot	GPLv3	?	Yes	No	?	?	?	?	?	?	?	?	?	?	?	Yes	Yes	Yes	?	?	?	?
AKernelLoader	GPLv2	?	Yes	No	Yes	Yes	Yes	Yes	?	Yes	Yes	?	?	?	?	?	?	Yes	?	?	?	?
Barebox	GPLv2	Yes	Yes	No	?	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	?	?	?	?	Yes	?	?	?	?
BootIt Bare Metal (formerly BootIt Next Generation)	Proprietary	?	?	?	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	?	Yes	Yes	Yes	Yes	Yes	?	?	?	?
BootKey	Proprietary	?	No	No	Yes	No	No	No	Yes	Yes	Yes	No	No	Yes	Yes	Yes	Yes	Yes	?	?	?	?
BootManager	MIT	?	Yes	No	No	Yes	?	?	?	?	?	?	No	Yes	Yes	Calls NTLDR	Calls Windows Boot Manager	Calls GRUB or LILO	?	?	?	?
BootX (Apple)	Proprietary	?	?	?	?	Yes	?	?	?	?	?	?	Yes	?	?	?	?	?	?	?	?	Yes
BootX (Linux)	Proprietary	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	Yes	?	?	?	?
Clover (fork of rEFIt)	GPLv2/BSD license	Yes	Yes	Yes	Yes	Yes	Yes	?	No	?	Yes	?	No	?	?	Yes	Yes	Yes	?	?	Yes	Yes
Darwin Boot Loader	APSL 2.0	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	Yes	Yes
Das U-Boot	GPLv2	?	?	?	?	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	?	?	?	?	Yes	?	?	Yes (FreeBSD)	?
GAG	GPLv2+	?	Yes (SafeBoot)	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes	Calls NTLDR	Calls Windows Boot Manager	Calls GRUB or LILO	Calls bootloader	Calls bootloader	Calls bootloader	No
GRUB Legacy	GPLv2+	?	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Calls NTLDR	Calls Windows Boot Manager	Yes	Calls FreeLoader	Yes	Yes	Yes
GNU GRUB	GPLv3	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Calls NTLDR	Calls Windows Boot Manager	Yes	Calls FreeLoader	Yes	Yes	Yes
GRUB4DOS	GPLv2+	?	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Calls NTLDR	Calls Windows Boot Manager	Yes	?	?	?	?
Gujin [1]	GPLv2	?	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Calls NTLDR	Yes	Yes	?	?	?	?
systemd-boot / Gummiboot	LGPL 2.1	Yes	No	No	?	Yes	Yes	Yes	?	?	Yes	?	?	No	No	Windows Server 2013 64bits with UEFI only	Calls Windows Boot Manager[1]	Yes	?	?	UEFI only	Yes[1]
iBoot	Proprietary	Yes	?	?	?	Yes	?	Yes	Yes	?	?	?	?	?	?	?	?	?	?	?	?	Yes
LILO	BSD license	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	?	?	Yes	Calls NTLDR	Yes	Yes	?	?	Calls biosboot (FreeBSD, PC-BSD, ...)	?
loader(8)	BSD license	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	?	Yes	?	?	?	?	?	?	?	Yes (FreeBSD, TrueOS)	?
loadlin	GPLv2+	?	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	No	Yes	No	No	?	?
MasterBooter	Proprietary	?	Yes	?	?	Yes	Yes	Yes	?	Yes	?	?	?	Yes	Yes	Yes	Yes	Yes	?	?	Yes	?
NTLDR	Proprietary	?	No	Yes	Yes	Yes	?	No	No	Yes	Yes	?	?	Yes	Yes	Yes	No	Calls GRUB4DOS	?	?	?	Calls Darwin bootloader[2]
OSL2000 Boot Manager	Proprietary	?	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Calls GRUB or LILO	?	?	?	?
PLoP Bootmanager	Proprietary	?	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Calls GRUB or LILO	?	?	?	?
RedBoot	GPLv2+	?	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Calls NTLDR	Calls Windows Boot Manager	Yes	Calls FreeLoader	Yes	Yes	Yes
rEFInd (fork of rEFIt)	GPLv3/BSD license	Yes	Yes	Yes	Yes	Yes	Yes	?	Yes	?	Yes	?	Yes	?	?	Yes	Yes	Yes	?	?	Yes	Yes
rEFIt (not maintained)	GPLv2/BSD license	Yes	Yes	Yes	Yes	Yes	Yes	?	No	?	Yes	?	No	?	?	Yes	Yes	Yes	?	?	Yes	Yes
Smart Boot Manager	GPLv2+	?	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	?	?	?	Yes	Yes	Yes	?	Yes	?	?	Yes	?
SPFdisk	GPLv2+	?	Yes	Yes	Yes	Yes	Yes	?	?	Yes	?	?	?	Yes	Yes	?	?	?	?	?	?	?
SYSLINUX	GPLv2+	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Calls NTLDR	Calls Windows Boot Manager	Yes	?	Yes	via mboot.c32 module [2]	?
XOSL	GPLv2	?	No	No	Yes	Yes	Yes	Yes	Yes	?	No	?	No	Yes	Yes	Yes	Yes	Yes	?	?	?	?
Windows Boot Manager	Proprietary	Yes	No	Yes	No	Yes	Yes	?	Yes	Yes	Yes	Yes	?	?	Yes	Calls NTLDR	Yes	Calls GRUB or LILO	?	?	?	?
FreeLoader (ReactOS Boot Loader)	GPLv2+	?	No	Yes	Yes	Yes	Yes	?	Yes	Yes	Yes	Yes	?	Yes	Yes	Partial[3]	Calls Windows Boot Manager	Yes	Yes	?	?	?
Name	Software license	Can reside in				Can boot from								Can boot								
		ESP (UEFI)	MBR	VBR	Floppy	Hard disk	Second hard disk	Logical partitions	CD-ROM	Floppy	USB	Zip	LAN	MS-DOS	Windows 9x/Me	Windows NT series	Windows Vista/7/8	Linux	ReactOS	MenuetOS	*BSD	Mac OS X

Source: https://en.wikipedia.org/wiki/Comparison_of_boot_loaders

Das U-Boot (the Universal Boot Loader)

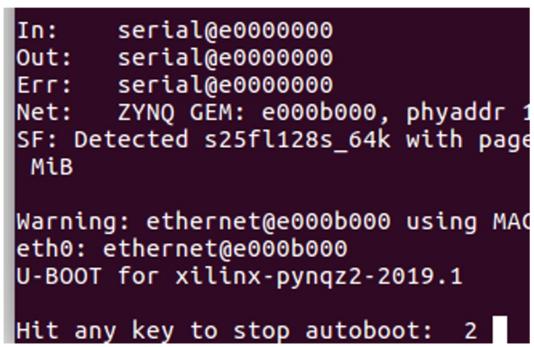
- Primary boot loader used in embedded devices
- Available for a number of architectures: ARM, RISC-V, MicroBlaze, …
- Can work on very limited amount of resources
- Comes with a command line tool for booting a particular kernel, manipulate device trees, download files, work with environment variables, …

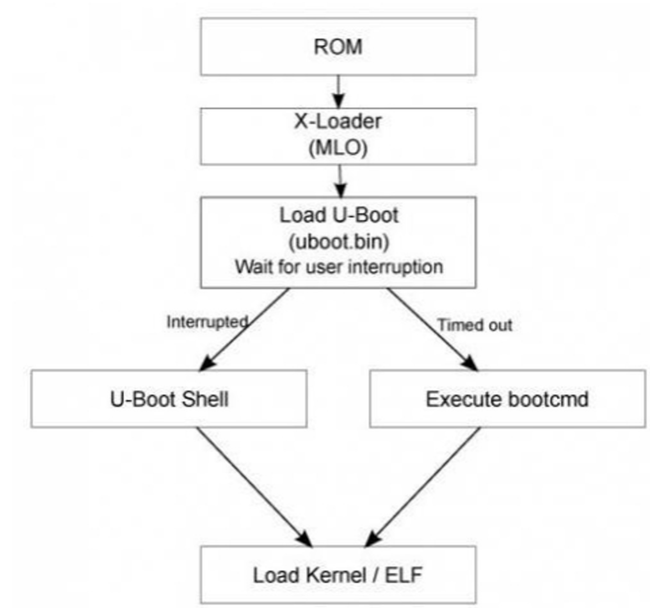
Source: https://www.denx.de/wiki/U-Boot/WebHome

U-Boot

Das U-Boot - Accessing Command Line Tool

- Keyboard interrupt during boot

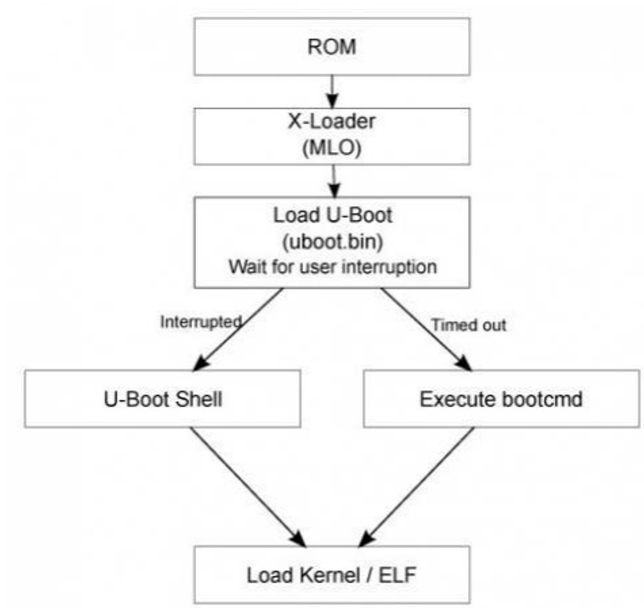




Das U-Boot (the Universal Boot Loader)

- Boot arguments can be changed at U-Boot Shell
- Bootargs examples:
  - root: root directory
  - rootfstype: file system type for root
  - console: serial port console
  - etc.
- Default args (after boot)
  - *$ cat /proc/cmdline*

    root=/dev/mmcblk0p2 rw
    earlyprintk rootfstype=ext4
    rootwait devtmpfs.mount=1
    uio_pdrv_genirq.of_id="generic-uio"
    clk_ignore_unused



A good summary: https://manpages.ubuntu.com/manpages/bionic/en/man7/bootparam.7.html

Lab Work 1 - Serial Connection

- Using a micro USB cable, connect your board to your laptop
- Connect to board using the serial connection
  - Linux: sudo screen /dev/<port> 115200, port: ttyUSB0 or ttyUSB1
  - MAC: sudo screen /dev/<port> 115200, port: check resources
  - Windows: check resources
  - Resources:
    - https://pynq.readthedocs.io/en/v2.0/getting_started.html
    - https://www.nengo.ai/nengo-pynq/connect.html
- After connecting to the board restart the board (*$ sudo reboot*), interrupt the boot (keyboard interrupt), and list current settings (*printenv*)

```
Zynq> printenv
arch=arm
autoload=no
baudrate=115200
board=zynq
board_name=zynq
boot_img=BOOT.BIN
boot_targets=mmc
bootcmd=run default_bootcmd
bootdelay=4
bootenv=uEnv.txt
bootenvsize=0x20000
bootenvstart=0x500000
clobstart=0x10000000
console=console=ttyPS0,115200
cp_kernel2ram=mmcinfo && fatload mmc ${sdbootdev} ${netstart} ${kernel_img}
cpu=armv7
default_bootcmd=run uenvboot; run cp_kernel2ram && bootm ${netstart}
dfu_mmc=run dfu_mmc_info && dfu 0 mmc 0
dfu_mmc_info=set dfu_alt_info ${kernel_image} fat 0 1\\;${devicetree_image} fat 0 1\\;${ramdisk_image} fat 0 1
dfu_ram=run dfu_ram_info && dfu 0 ram 0
dfu_ram_info=set dfu_alt_info ${kernel_image} ram 0x3000000 0x500000\\;${devicetree_image} ram 0x2A00000 0x20000\\;${ramdisk_image} ram 0x2000000 0x600000
dtb_img=system.dtb
dtbnetstart=0x23fff000
eraseenv=sf probe 0 && sf erase ${bootenvstart} ${bootenvsize}
ethaddr=00:05:6b:00:b6:9c
fault=echo ${img} image size is greater than allocated place - partition ${img} is NOT UPDATED
fdtcontroladdr=1f326c00
importbootenv=echo "Importing environment from SD ..."; env import -t ${loadbootenv_addr} $filesize
install_boot=mmcinfo && fatwrite mmc ${sdbootdev} ${clobstart} ${boot_img} ${filesize}
install_jffs2=sf probe 0 && sf erase ${jffs2start} ${jffs2size} && sf write ${clobstart} ${jffs2start} ${filesize}
install_kernel=mmcinfo && fatwrite mmc ${sdbootdev} ${clobstart} ${kernel_img} ${filesize}
jffs2_img=rootfs.jffs2
kernel_img=image.ub
loadaddr=0x10000000
loadbootenv=load mmc $sdbootdev:$partid ${loadbootenv_addr} ${bootenv}
loadbootenv_addr=0x00100000
modeboot=sdboot
netstart=0x10000000
psserial0=setenv stdout ttyPS0;setenv stdin ttyPS0
sd_uEnvtxt_existence_test=test -e mmc $sdbootdev:$partid /uEnv.txt
sd_update_dtb=echo Updating dtb from SD; mmcinfo && fatload mmc ${sdbootdev}:1 ${clobstart} ${dtb_img} && run install_dtb
sd_update_jffs2=echo Updating jffs2 from SD; mmcinfo && fatload mmc ${sdbootdev}:1 ${clobstart} ${jffs2_img} && run install_jffs2
sdbootdev=0
serial=setenv stdout serial;setenv stdin serial
soc=zynq
stderr=serial@e0000000
stdin=serial@e0000000
stdout=serial@e0000000
test_crc=if imi ${clobstart}; then run test_img; else echo ${img} Bad CRC - ${img} is NOT UPDATED; fi
test_img=setenv var "if test ${filesize} -gt ${psize}; then run fault; else run ${installcmd}; fi"; run var; setenv var
thor_mmc=run dfu_mmc_info && thordown 0 mmc 0
thor_ram=run dfu_ram_info && thordown 0 ram 0
uenvboot=if run sd_uEnvtxt_existence_test; then run loadbootenv; echo Loaded environment from ${bootenv}; run importbootenv; fi; if test -n $uenvcmd; then echo Running uenvcmd ...; run uenvcmd; fi
vendor=xilinx

Environment size: 2625/131068 bytes
Zynq>
```

Changing bootargs During Boot - Default args

- Default parameters
  - 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 apply changes:
  - Interrupt the boot
  - Edit boot arguments:
    - *$ editenv bootargs*
    - Insert arguments included the quotations all in one line:
      - Default args at “copypaste.txt”
    - *$ boot*

Scheduler Options - isolcpus

- Remove the specified CPUs, as defined by the *cpu_number* values.
- These are set of CPUs that the kernel process scheduler will not interact.

Lab Work 3 - Changing Bootargs

- Custom for WES237A-Lab3 (lab work)
  - add isolcpus to bootargs
  - 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'
  - After boot, by using *htop* command monitor the CPU utilization
  - To exit from *htop*, press “q” or “Control + C”.
- To apply changes:
  - Interrupt the boot
  - Edit boot arguments:
    - *$ editenv bootargs*
    - Insert arguments included the quotations all in one line:
      - isolcpus args at “copypaste.txt”
    - *$ boot*

Fibonacci Sequence

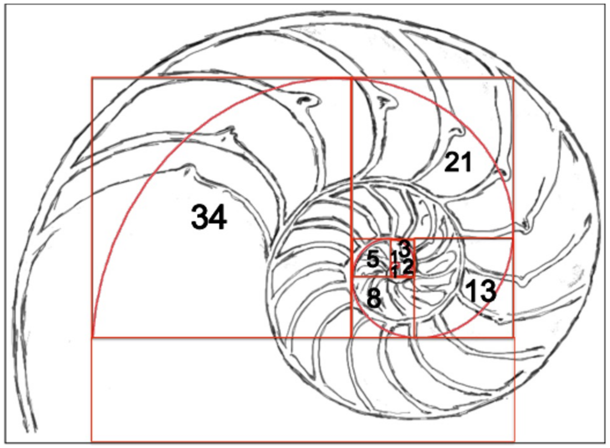
$$F_0 = 0, \quad F_1 = 1,$$

and

$$F_n = F_{n-1} + F_{n-2},$$

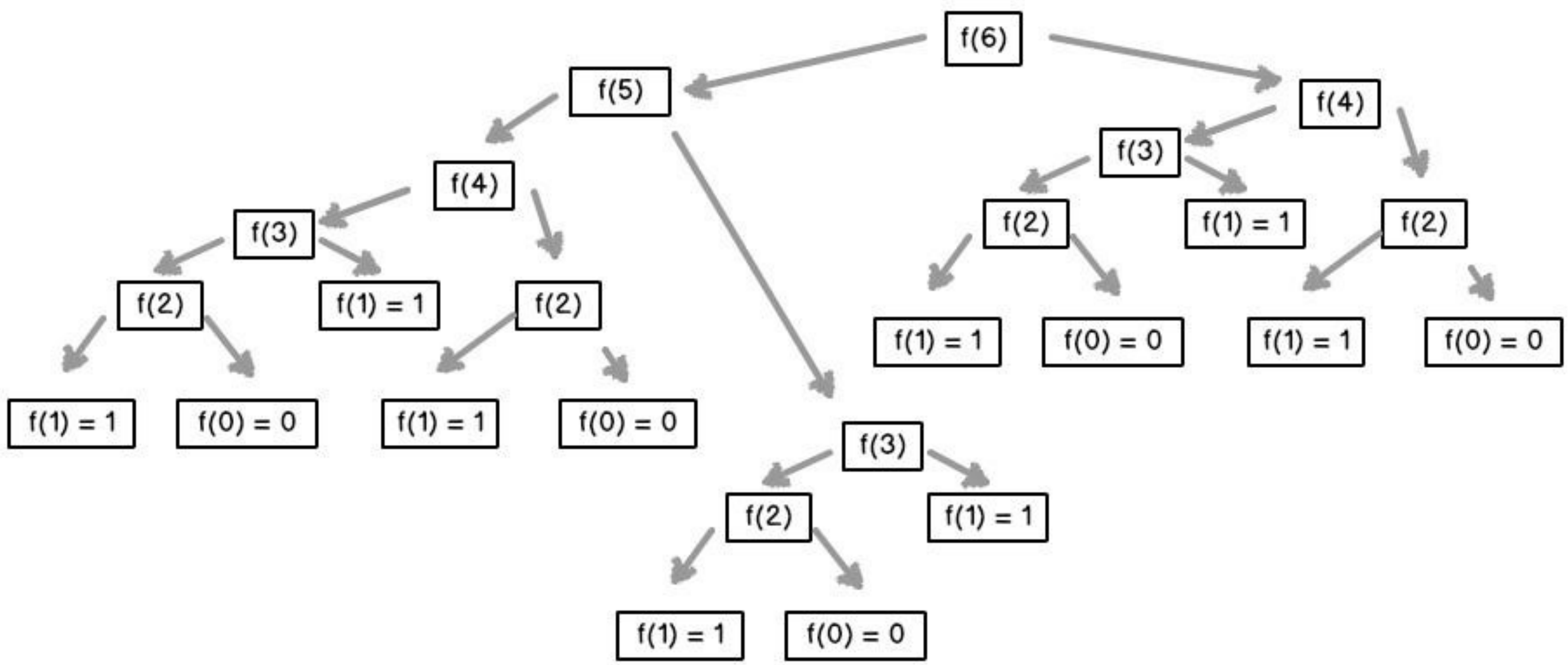
for $n > 1$.

$$0,\ 1,\ 1,\ 2,\ 3,\ 5,\ 8,\ 13,\ 21,\ 34,\ 55,\ 89,\ 144,\ \ldots$$



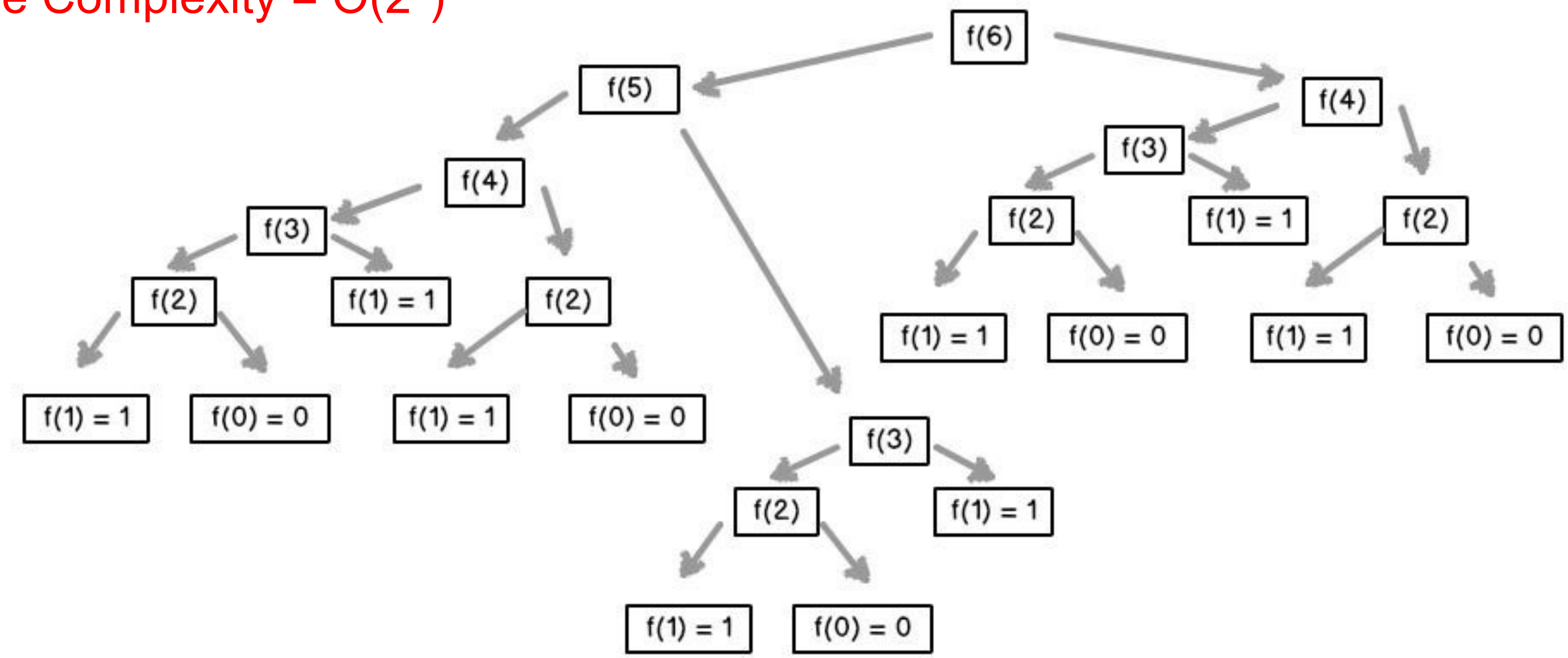

```python
def recur_fibo(n):
   if n <= 1:
       return n
   else:
       return(recur_fibo(n-1) + recur_fibo(n-2))
```

Time is a very computational heavy algorithm



Time is a very computational heavy algorithm

Time Complexity = $O(2^n)$



Lab Work 2 - Heavy CPU Utilization

- Download *fib.py*. This is a recursive implementation for generating Fibonacci sequence. We just do not print the results.
- Make sure your board is booted with custom bootargs 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)
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
3) Heavy Utilization on CPU0:
    - Open another terminal and run $ *dd if=/dev/zero of=/dev/null*
    - Repeat parts 1 and 2

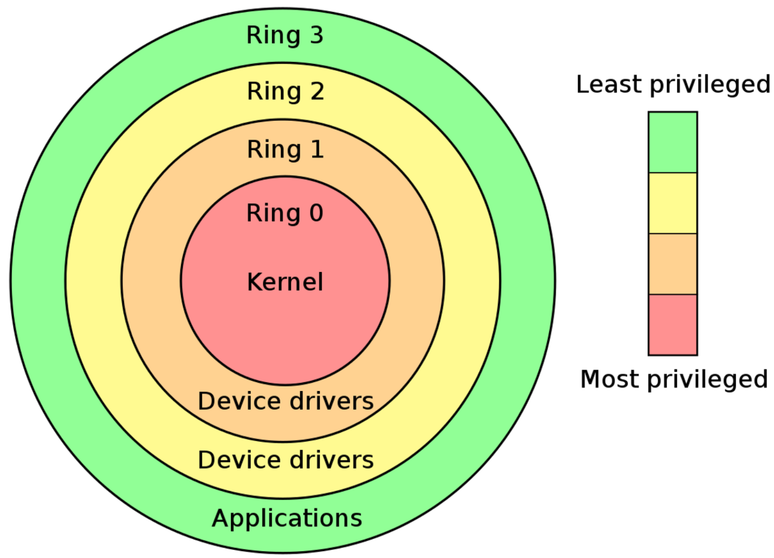
ARM Performance Monitoring Unit (PMU)

- PMU is an event counting hardware which can be used to profile and benchmark code
- Cortex-A9 PMU provides six counters
- Each counter can count any of the 58 events available in the Cortex-A9 processor
- These counters can be accessed through debugging tools or directly through the CP15 registers

Reference: https://developer.arm.com/docs/101392/latest/part-c-debug-descriptions/performance-monitor-unit/pmu-events

Accessing ARM PMU

- By default user space (applications), do not have access to PMU
- Kernel has access to PMU



Providing Access to PMU for Applications

- A kernel object can enable user-mode for PMU
  - kernel_modules/CPUcntr.c

```c
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Alireza Khodamoradi");
MODULE_DESCRIPTION("This module enables users to access performance counter");

static int __init CPUcntr_init(void){
  asm ("MCR p15, 0, %0, c9, c14, 0\n\t" :: "r"(1));
  // disable counter overflow interrupts
  asm ("MCR p15, 0, %0, c9, c14, 2\n\t" :: "r"(0x8000000f));
  printk(KERN_INFO "CPU clock counter is enabled.\n");
  return 0;
}

static void __exit CPUcntr_clean(void){
  // disable user-mode access to the performance counter
  asm ("MCR p15, 0, %0, c9, c14, 0\n\t" :: "r"(0));
  printk(KERN_INFO "CPU clock counter is disabled.\n");
}

module_init(CPUcntr_init);
module_exit(CPUcntr_clean);

~
```

Reference: https://developer.arm.com/docs/100511/latest/performance-monitoring-unit/pmu-register-summary

Move to Coprocessor from ARM Register (MCR)

System control processor registers overview

This section gives details of all the registers in the system control coprocessor. The section presents a summary of the registers and detailed descriptions in register order of CRn, Opcode_1, CRm, Opcode_2.

You can access CP15 registers with MRC and MCR instructions, as described in *Use of the system control coprocessor*.

```
MCR{cond} p15,<Opcode_1>,<Rd>,<CRn>,<CRm>,<Opcode_2>
```

```
MRC{cond} p15,<Opcode_1>,<Rd>,<CRn>,<CRm>,<Opcode_2>
```

← Previous Section

Next Section →

Lab Work 3 - Accessing ARM PMU

- PMU is a coprocess
- Assembly code to transfer between coprocess and ARM register
  - MCR: ARM register to Co-process
    - https://www.keil.com/support/man/docs/armasm/armasm_dom1361289877204.htm
  - MRC: Co-process to ARM register
    - https://www.keil.com/support/man/docs/armasm/armasm_dom1361289880404.htm
- PMU coprocID is p15
- Steps to access PMU
  - Write a kernel object which can call these commands
  - Compile an insert kernel object
  - Include *asm()* commands in a ‘C’ header for a ‘C’ program
    - In the c-code, initialize the counter and then read from the counter to get cycle count

Think .ko as .so, but in the kernel space.

Creating a Kernel Object

- A kernel object can be created from C code (CPUcntr.c)
  - kernel_modules/README

```
*** Make sure `pwd` does not contain any space ***

To compile:
 make -C /lib/modules/$(uname -r)/build M=$(pwd) modules

To clean:
 make -C /lib/modules/$(uname -r)/build M=$(pwd) clean

To insert module:
 insmod CPUcntr.ko

To remove module:
 rmmod CPUcntr.ko

To check module:
 dmesg | tail -1
```

  - Check the module insertion using the $ dmesg command

Reference: https://developer.arm.com/docs/100511/latest/performance-monitoring-unit/pmu-register-summary

Interacting with PMU

- A program written in C can read from PMU
  - clock_example/include/cycletime.h

```c
/*
 Author = "Alireza Khodamoradi"
*/

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>

static inline unsigned int get_cyclecount(void){
  unsigned int value;
  asm volatile ("MRC p15, 0, %0, c9, c13, 0\n\t" : "=r"(value));
  return value;
}

static inline void init_counters(int32_t do_reset, int32_t enable_divider){
  int32_t value = 1;
  if(do_reset)
    value |= 6; // reset all counters to zero.
  if(enable_divider)
    value |= 8;
  value |= 16;
  // Program the performance-counter control-register
  asm volatile ("MCR p15, 0, %0, c9, c12, 0\n\t" :: "r"(value));
  // Enable all counters
  asm volatile ("MCR p15, 0, %0, c9, c12, 1\n\t" :: "r"(0x8000000f));
  // Clear overflow
  asm volatile ("MCR p15, 0, %0, c9, c12, 3\n\t" :: "r"(0x8000000f));
}
```

Volatile, what does this mean?

Reference: https://usermanual.wiki/Document/armv7armanual.23331929/view

Interacting with PMU

- A program written in C can read from PMU
  - clock_example/include/cycletime.h

```c
/*
 Author = "Alireza Khodamoradi"
*/

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>

static inline unsigned int get_cyclecount(void){
  unsigned int value;
  asm volatile ("MRC p15, 0, %0, c9, c13, 0\n\t" : "=r"(value));
  return value;
}

static inline void init_counters(int32_t do_reset, int32_t enable_divider){
  int32_t value = 1;
  if(do_reset)
    value |= 6; // reset all counters to zero.
  if(enable_divider)
    value |= 8;
  value |= 16;
  // Program the performance-counter control-register
  asm volatile ("MCR p15, 0, %0, c9, c12, 0\n\t" :: "r"(value));
  // Enable all counters
  asm volatile ("MCR p15, 0, %0, c9, c12, 1\n\t" :: "r"(0x8000000f));
  // Clear overflow
  asm volatile ("MCR p15, 0, %0, c9, c12, 3\n\t" :: "r"(0x8000000f));
}
```

Volatile, what does this mean?
Compiler should not optimize this. The writes could not be collapsed and the accesses should be done in the same order as how they appeared in the code execution.

Reference: https://usermanual.wiki/Document/armv7armanual.23331929/view

Interacting with PMU

- A program written in C to read from PMU
    - clock_example/src/main.cpp

```cpp
#include "main.h"
#include "cycletime.h"
#include "timer.h"
#include <unistd.h>

using namespace std;

int main(int argc, const char * argv[])
{
    float cpu_timer;
    unsigned int delay = 1;

    cout << "WES237A lab 4" << endl;

    char key=0;

    // 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)
    //TODO: initialize the counter

    //TODO: get the cyclecount before sleeping
    usleep(delay);
    //TODO: get the cyclecount after sleeping

    //TODO: subtract the before and after cyclecount
    //TODO: print the cycle count (see the print statement for the cpu_timer below)

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


    cout << "Timer: " << (double)cpu_timer/1000000000.0 << endl;

    return 0;
}
```

Lab Work 3 (cont) - Example Code

- Navigate to *clock_example* directory. And build the example code by running *$ make*
- By using taskset, run the elf file (*$ taskset -c <0,1> ./lab3*) on both CPUs

A Simple Trick to Enable PMU on Both Cores

- Instructions are at *kernel_modules/README*
- Remove CPUcntr object from kernel
- Insert CPUcntr.ko to CPU0
- Remove CPUcntr.ko from CPU1
- Insert CPUcntr.ko to CPU1
- Now try to run *lab3* on both CPUs

Summary

- Lab Work 1 + 2: Serial Connection and Boot Args
- Lab Work 3: Heavy CPU Utilization and Accessing ARM PMU