Accessing Custom Computing Cores

- Standalone C Progams — simple!
  - set a pointer (`x`) to the base address from `xparameters.h` file
  - read/write registers with `*x`
- With an OS — a little more complicated

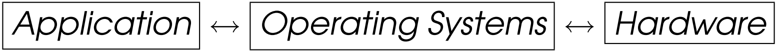
OS Issues

- OS provides *protection* so processes cannot disrupt each other
- OS manages resources (including custom cores)
- OS turns on Memory Management Unit (MMU) so processes have execute in a ***virtual*** *address space* (xparameters.h and custom cores have ***physical*** addresses

OS Issues

- OS provides *protection* so processes cannot disrupt each other
- OS manages resources (including custom cores)
- OS turns on Memory Management Unit (MMU) so processes have execute in a ***virtual*** *address space* (xparameters.h and custom cores have ***physical*** addresses
- *Consequently, we cannot just use physical addresses in our applications.*

Accessing Hardware with an OS

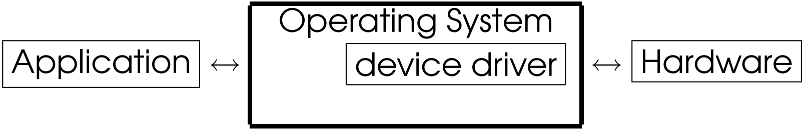


Application ↔ *Operating Systems* ↔ *Hardware*

Accessing Hardware with an OS

Application ↔ *Operating Systems* ↔ *Hardware*

Or (more precisely)



Application ↔ Operating System device driver ↔ Hardware

Our Job

- create custom hardware core
- create base platform system with custom core
- create a software application

Our Job

- create custom hardware core
- create base platform system with custom core
- create a software application
- create root filesystem
- **create device driver**
- compile kernel

Our Job

- create custom hardware core
- create base platform system with custom core
- create a software application
- create root filesystem
- **create device driver**
- compile kernel
- roll it all into an ACE file

Creating a Device Driver for Kernel

- Two Ways of Compiling
  - "in-tree" — starting with a Linux kernel, we add our source code in an appropriate subdirectory (and *update* existing Makefiles)
  - "out-of-tree" — in our own subdirectory, we add our source code and create our own Makefile; an environment variable *points* to the Linux's kernel subdirectory
- Device driver can either be directly compiled in (i.e. the equivalent of 'Y' in menuconfig) or it can be compiled as a module (a 'M' in menuconfig)
- Note: to have your device show up in menuconfig is another step beyond what we describe here

Linux Kernel Modules

- our focus: out-of-tree compilation, always as a module
- whole classes can be taught about single kernel subsystems!
- we are going to cherry-pick for this class
- you *must* read on your own:

  `http://lwn.net/Kernel/LDD3/`

  Chapter 3: Char Devices

Kernel Module Commands

- `lsmod`
- `insmod`
- `rmmod`
- `modprobe`

Device Files

- major, minor numbers
- character versus block
- `mknod`
- /dev
- udev v. MAKEDEV

Communication through Files

- application side — system calls
- kernel side — file operations