NICTA

CAmkES Tutorial

25 July 2014

Ihor Kuz

Australian Government
Department of Broadband, Communications and the Digital Economy
Australian Research Council

NICTA Funding and Supporting Members and Partners

Overview

- Installing and runninng on qemu
- Setting Up Hardware
- Running on Hardware
- Writing a CAmkES System
- Device Drivers and Device Access
- The Daughterboard
  - the DARPA example app
- Misc:
  - Priorities
  - Mutexes
  - One-way RPC

Prerequisites

- Linux (Ubuntu)
  - sudo apt-get install lib32z1 lib32ncurses5 lib32bz2-1.0
- Compiler

  -
    ```
    wget https://sourcery.mentor.com/public/gnu_toolchain/arm-none-eabi/arm-2013.11-24-arm-none-eabi-i686-pc-linux-gnu.tar.bz2
    ```
  - unpack into /opt/local

  -
    ```
    echo "export PATH=/opt/local/arm-2013.11/bin:\$PATH" >> ~/.bashrc
    ```
- Python

  -
    ```
    sudo apt-get instal python-pip python-tempita
    ```

  -
    ```
    sudo pip install --upgrade pip
    ```

  -
    ```
    sudo pip install jinja2 ply pyelftools
    ```
- Haskell

  -
    ```
    sudo apt-get install cabal-install
    ```

  -
    ```
    cabal update; cabal install MissingH data-ordlist split
    ```
- Qemu

  -
    ```
    sudo apt-get install qemu
    ```
- Misc

  -
    ```
    sudo apt-get install realpath libxml2-utils
    ```
- CAmkES

  -
    ```
    wget https://www.dropbox.com/s/8sbfvmv9c1a26b4/camkes-project-archive.tgz.gpg
    ```

Building

- Config
  - `ls configs/`
  - `make arm_simple_defconfig`
  - `make silentoldconfig`
- Manual config
  - `make menuconfig`
- Build it!
  - `make`
  - `make V=1`
  - results in
    - stage/
    - build/
    - images/
- Clean up
  - `make clean`
  - `make clobber`
  - `make mrproper`

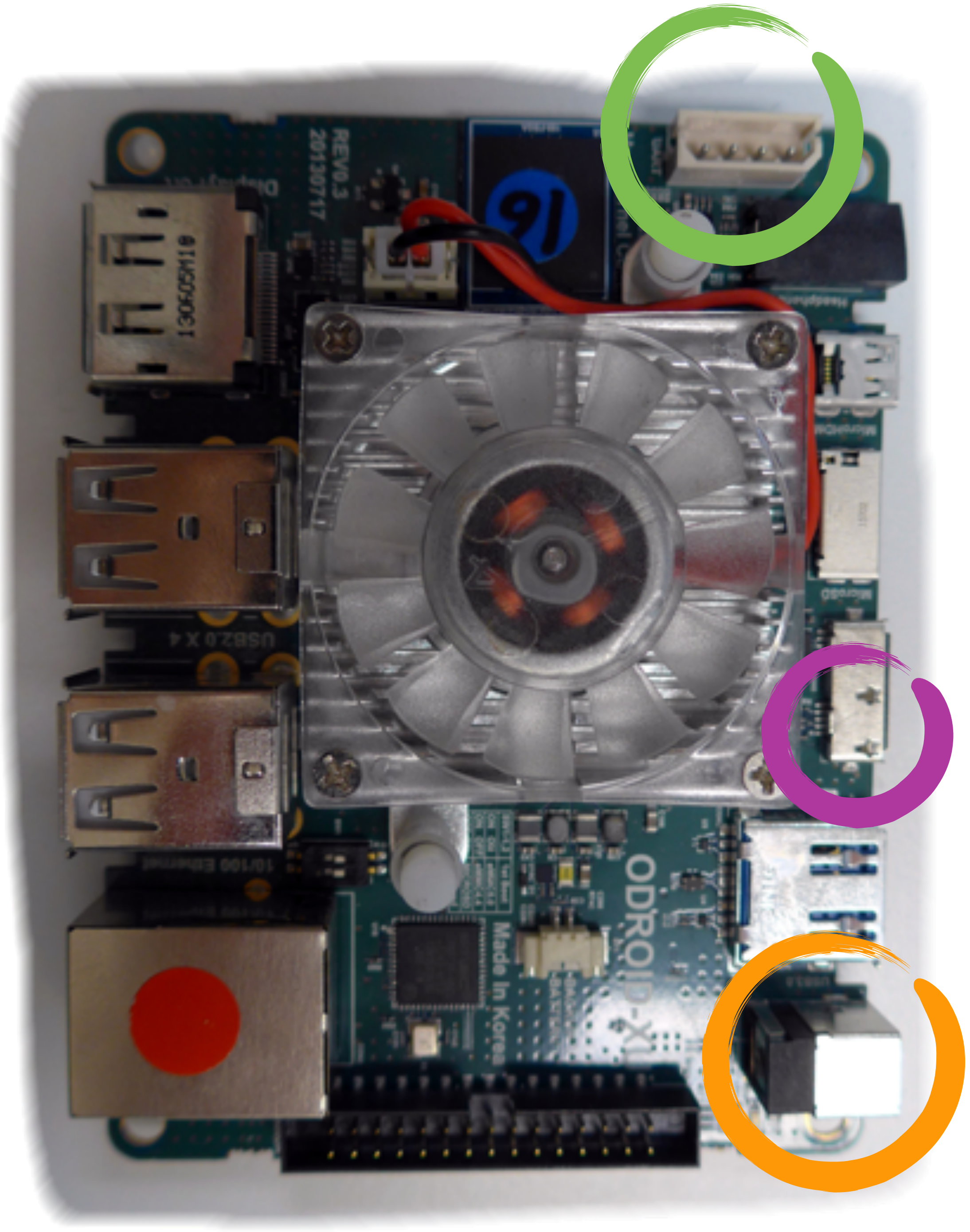
Running in Qemu

- Run in Qemu
  - `qemu-system-arm -M kzm -nographic -kernel images/capdl-loader-experimental-image-arm-imx31`
- Results

```
Starting the client
-------------------
echo_int: 42 -> 42
echo_float: 273421.437500 -> 273421.437500
echo_double: 273421.427400 -> 273421.427400
echo_mix: 273421.427400 -> 273421
echo_string: "hello world" -> "hello world"
echo_string: "a longer string that will overflow the message
registers on ARM" -> "a longer string that will overflow the
message registers on ARM"
echo_parameter: 123 -> 123 (returned = 123)
increment_parameter: 100 -> 101
After the client
```

Hardware Setup

- Odroid-XU
  - Power
  - USB
  - Serial



Hardware Setup

- Prerequisites
  - `sudo apt-get install minicom android-tools-fastboot u-boot-tools`
  - configure minicom
    - update `/etc/group`: add self to dialup
    - 115200 8N1 HW/SW flow control off, save as `odroid`
  - configure fastboot
    - `echo SUBSYSTEM=="usb", ATTR{idVendor}=="18d1", MODE=="0666", GROUP=="users" | sudo tee /etc/udev/rules.d/40-odroidxu-fastboot.rules`
- Connect the cables
  - UART-USB to UART
  - micro USB to micro USB slot
- Start minicom
  - new window
  - `minicom odroid`
- Start the Odroid-XU

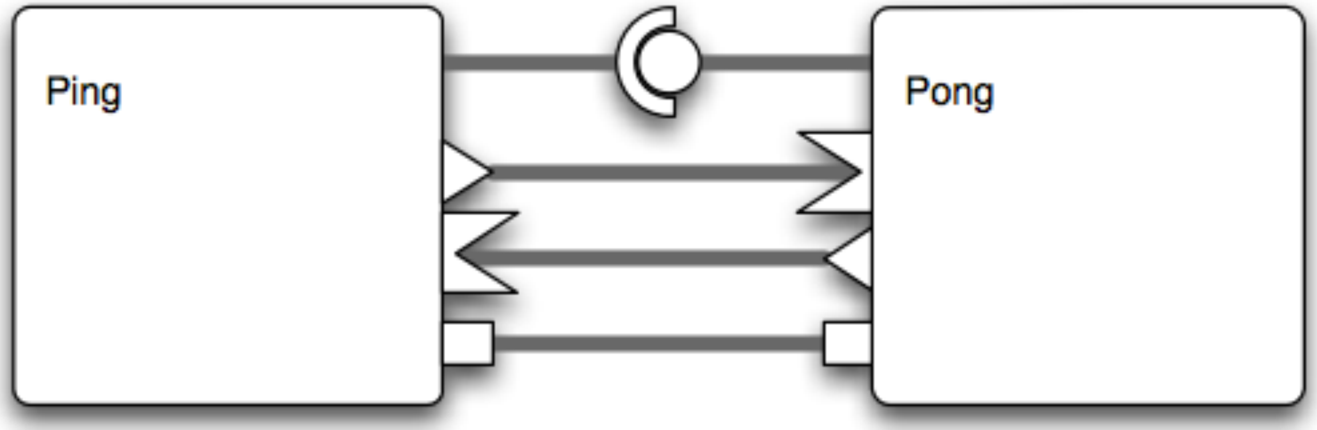
Flashing U-Boot

- Prerequisites
  - minicom, fastboot, see previous slide
  - bl2: `odroid/smdk5410-spl.bin.signed`
  - u-boot: `odroid/u-boot.bin`
- Turn it on
  - in minicom window, make sure Odroid goes into fastboot
- Flash away
  - `sudo fastboot flash bl2 smdk5410-spl.bin.signed`
  - `sudo fastboot flash bootloader u-boot.bin`
  - `sudo fastboot reboot`
- Set fastboot as boot command
  - `setenv bootcmd fastboot`
  - `saveenv`

Building, Loading, Running

- Build it
  - `make clean` !!
  - `make arm_exynos5_simple_defconfig`
  - `make silentoldconfig`
  - `make`
  - `cd images`
  - `mkimage -a 0x48000000 -e 0x48000000 -C none -A arm -T kernel -O qnx -d capdl-loader-experimental-image-arm-exynos5 odroid-image`
- Load and Run
  - `sudo fastboot boot odroid-image`
  - output is in minicom
- Restart

My First System



- Prepare Directory Structure
  - apps/pingpong
    - pingpong.camkes
    - Kconfig, Kbuild, Makefile
    - components
      - Ping, Pong
        - » Ping.camkes, Pong.camkes
        - » src
    - interfaces
      - PingPong.idl4

Write the Code

- CAmkES ADL
  - `pingpong.camkes`
  - `Ping.camkes, Pong.camkes`
  - `PingPong.idl4`
- Components
  - `Ping/src/ping.c`
  - `Pong/src/pong.c`
- Configure
  - Kconfig, Kbuild, Makefile, main Kconfig
  - `make menuconfig`
- Build
  - `make`
- Run
  - `qemu-system-arm -M kzm -nographic -kernel images/capdl-loader-experimental-image-arm-imx31`

Debugging

- Let’s make a mistake
  - Null dereference in `ping.c`
  - `*(int *)0 = 1;`
- Build, run, crash
  - where did it crash?
    - `vm fault on data at address 0x0 with status 0x805`
    - `in thread 0xf2257b00 at address 0x100054`
- objdump to the rescue
  - find the component binary
    - `build/arm/imx31/pingpong/ping.instance.bin`
  - dump it
    - `arm-none-eabi-objdump -dS build/..../ping.instance.bin`
  - find the instruction, and corresponding C statement
- But which component was it?
  - good question...

Device Access

- Hardware components
  - memory-mapped IO
  - Interrupts
  - IOports (x86)
- Example (epit)
  - epit.camkes

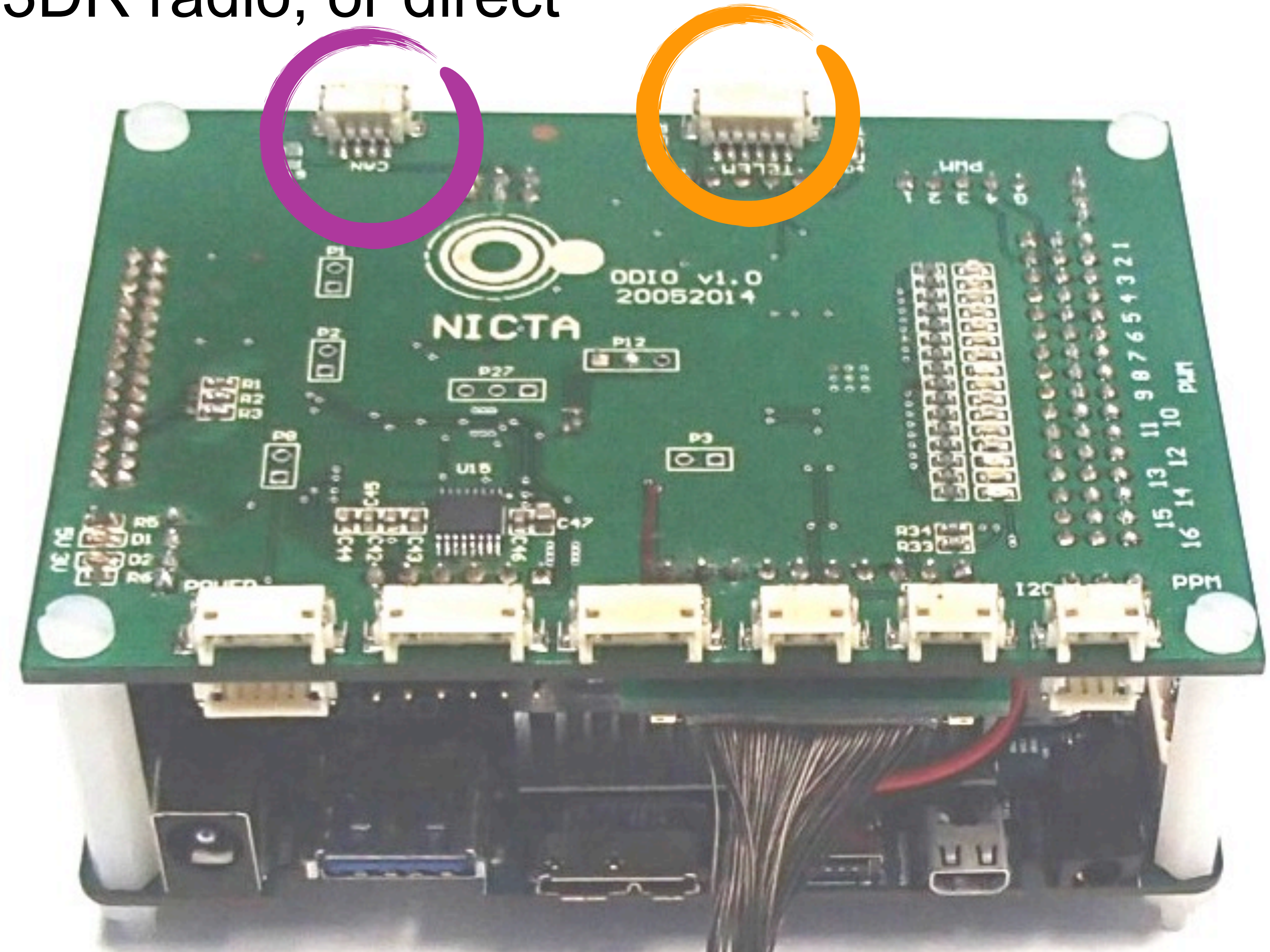
```
component EPIT {
    hardware;
    dataport Buf mem;
    emits DataAvailable irq;
}
```

```
component Driver {
        control;
        dataport Buf mem;
        consumes DataAvailable irq;
}
```

```
connection seL4HardwareMMIO epit_mem(from drv.mem, to epit.mem);
connection seL4HardwareInterrupt irq(from epit.irq, to drv.irq);
configuration {
        epit.mem_attributes = "0x53F98000:0x1000";
        epit.irq_attributes = 27;
}
```

Daughterboard setup

- What to hookup
  - Odroid-XU connections
  - CAN
  - UART - 3DR radio, or direct



The “DARPA” app

- Config, Build, Load, Run

```
make arm_exynos5_DARPA_config
make silentoldconfig
make
cd images
mkimage -a 0x48000000 -e 0x48000000 -C none -A arm -T kernel -O qnx -d capdl-loader-experimental-image-arm-exynos5 odroid-image
sudo fastboot boot odroid-image
```

- What does it do?
  - UART: echo in 10 character blocks
  - CAN: send and receive simple messages, reset CAN on errors.

Priorities

- Control thread priority
  - <instance>._control_priority = <priority>
- Interface thread priority
  - <instance>.<interface>_priority = <priority>

```
configuration {
   ping._control_priority = 100;
   pong._control_priority = 200;
   pong.ping_priority = 250;
}
```

Mutex

- Example app
  - `make arm_mutex_defconfig`
    - mutexes as connectors
  - `make arm_socket_defconfig`
    - use spinlocks
  - libs/libsel4sync
    - mutex
    - spinlock
    - semaphore
    - atomic ops

One Way RPC

- RPC connector
  - uses seL4 Call/ReplyWait
  - caller blocks until callee finishes and returns
  - can result in stalled call chains
- One Way RPC
  - calls that don’t need results
  - transfer call and return immediately
- Sel4RPCAsync connector
  - not builtin (yet)
  - user defined connector
    - easy modification of existing seL4RPCCall connector