The Terasic/Intel DE2i-150
Hello PClex Design Example

Patrick Schaumont

Bradley Department of Electrical and Computer Engineering
Virginia Tech
Blacksburg, VA

26 June 2013
DE2i-150 Hardware and Software
- DE2i-150 Hardware
- DE2i-150 Software

Development Environment
- Hardware Setup
- HW/SW Codesign with Embedded OS

PClex Basics

HelloPCI Design
- HelloPCI FPGA Design
- HelloPCI Device Driver
- HelloPCI User Application
The DE2i-150 Development Kit
The Hardware

Block diagram from DE2i-150 System Manual
Atom-N2600 subsystem

- Atom N2600 (cedartrail) dual-core dual-thread 1.6GHz
  - Atom SSE2, SSE3, SSSE3
  - on-chip 400MHz Graphics Processing Unit
  - 3.5W Thermal Design Power
- NM-10 chipset
  - USB, SATA, LAN, Audio, miniPCI, 2 PCIe to FPGA
- 2GB DDR3 (SO-DIMM) system memory
- 64GB mSATA SSD
- Ports: Wifi, Ethernet, 4xUSB, HDMI, VGA
Altera Cyclone IV GX-150 subsystem

- Altera Cyclone IV EP4CGX150DF31 FPGA
  - 150K LE (eqv 82 Nios/f cores)
  - 6480Kbit on-chip RAM, 360 18x18 Mult, 4 PLL
  - 1 PClex hard IP

- 2MB SRAM, 64 MB SDRAM, 64 MB Flash

- Buttons, Leds, LCD, IR

- TV, VGA, User IO, Ethernet

- 2 PClex to NM10
Atom-N2600 software

Yocto ([www.yoctoproject.org](http://www.yoctoproject.org)) provides complete software development environment

- Source download and patching
- Board Support Package(s)
- Configuration and Build System: Kernel, Filesystem, Applications
- Software Cross-development Tools
- Simulation (through QEMU)
- Performance profiling and tracing
Development setup

- Ethernet connection for quick transfer of software images
- USB connection for FPGA configuration
- Optional: Keyboard/Monitor for DE2i-150
Configuration Steps

Step-by-Step Guide on rijndael.ece.vt.edu/de2i150 (see Design Documentation for hellopci)

Steps

1. Configure network and forwarding
2. Install and configure TFTP
3. Install FPGA Design Environment (Quartus 12.1)
4. Install Yocto
   - Install Yocto Design Environment (Yocto Danny 8.0.2)
   - Install cedartrail Board Support Package
   - Configure and compile kernel
   - Install kernel, networking on DE2i-150
   - Configure and compile cross-development environment
   - Install cross-development
Bare metal C vs Embedded OS

Bare Metal Application

Application (C)

Hardware Abstraction

Memory Map

IO Map

Interrupt Vectors

Hardware

Application on Embedded OS

Application

Kernel

File Systems

Tasks

Memory Management

Hardware Abstraction

Memory Map

IO Map

Interrupt Vectors

Hardware
Bare metal C vs Embedded OS

- **Bare Metal C**
  - Hardware Abstraction (memory-map, interrupts, custom instruction, ..) is directly visible to the application in C
  - Concurrency is handled by the application
  - No privileges (no protection)

- **Embedded OS**
  - Hardware Abstraction is handled by the kernel
  - Application only sees kernel abstractions (files, threads, devices, ..)
  - Concurrency is handled by the kernel
  - Distinguishes software in kernel space from software in user space
Platform Customization

Bare Metal Application

1. Application (C)
   - New Hardware Abstraction
   - Hardware Abstraction
     - Memory Map
     - IO Map
     - Interrupt Vectors
   - Hardware

Application on Embedded OS

1. Application
   - New Kernel Abstraction
   - Kernel
     - File Systems
     - Tasks
     - Memory Management
     - Hardware Abstraction
       - Memory Map
       - IO Map
       - Interrupt Vectors
   - Hardware
PCI/PCIex basics

- **PCIex = PCI Express**
- Communication between FPGA and Atom is handled by two PCIex 1x links
  - ’1x’ means a double differential pair for tx, rx
  - Speed equals 2.5Gbps in each direction tx, rx
  - PCI is a computer bus with auto-configuration, on-device bios, Qos
  - PCIex offers error handling, QoS, power management, hot-plug
  - PCI is bus-based (master/slave/bridge), works with bus cycles
  - PCIex is point-to-point (root/endpoint/switch), works with packets
- As seen from software, PCIex is compatible with PCI
PClex basics

- PClex devices define three ‘spaces’ for communication between processor and hardware
  - One or more memory regions (accessible with memory load/store instructions)
  - One or more IO regions (accessible with input/output instructions - X86 specific)
  - A configuration regions for device management
- PClex communications are formatted as packets from CPU to hardware and vice versa
  - Packets hold address, data, control information
  - Posted transactions require a single packet; non-posted transactions require a request packet and a completion packet.
  - Memory Read, IO R/W, Configuration R/W are non-posted
  - Memory Write and Messages are posted
Inspecting PCI Devices in Linux

```
BOARD$ lspci
00:00.0 Host bridge: Intel Corporation Atom Processor
  D2xxx/N2xxx DRAM Controller (rev 04)
00:1b.0 Audio device: Intel Corporation N10/ICH 7 Family
  High Definition Audio Controller (rev 02)
..
01:00.0 Non-VGA unclassified device: Altera Corporation
  Device 0004 (rev 01)
```

- Devices are listed as `bus:slot:function`
- Same terminology applies for PCI and PClex
- Use `lspci -v` and `lspci -vv` for increased verbosity
Design Specs

HelloPCI

while (1) {
    read (16 switches);
    write (value to 4 hex display);
}
HelloPCI Architecture
HelloPCI Design Steps

1. Design FPGA configuration
   - PCI Interface as Avalon Bus Master
   - PIO ports as Avalon Bus Slaves

2. Design Device driver
   - PCI Driver to map PIO ports into kernel address space
   - Character Device Driver to connect user application and kernel

3. Design Application
   - Top-level application in C
**FPGA Design**

![FPGA Design Interface](image)

- **Qsys - pchellocore.qsys**
  - Components: `Project`, `Library`, `System`
  - Connections: `hexport`, `reset`, `clk`, `s1`, `external_connection`
  - `pcie_hard_ip_0`: IP Compiler for PCI Express

### System Contents

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Export</th>
</tr>
</thead>
<tbody>
<tr>
<td>hexport</td>
<td>PO (Parallel I/O)</td>
<td>Double-click to export</td>
</tr>
<tr>
<td>clk</td>
<td>Clock Input</td>
<td>Double-click to export</td>
</tr>
<tr>
<td>reset</td>
<td>Reset Input</td>
<td>Double-click to export</td>
</tr>
<tr>
<td>s1</td>
<td>Avalon Memory Mapped Slave</td>
<td>Double-click to export</td>
</tr>
<tr>
<td>external_connection</td>
<td>Conduit Endpoint</td>
<td>Double-click to export</td>
</tr>
<tr>
<td>import</td>
<td>PO (Parallel I/O)</td>
<td>Double-click to export</td>
</tr>
<tr>
<td>clk</td>
<td>Clock Input</td>
<td>Double-click to export</td>
</tr>
<tr>
<td>reset</td>
<td>Reset Input</td>
<td>Double-click to export</td>
</tr>
<tr>
<td>s1</td>
<td>Avalon Memory Mapped Slave</td>
<td>Double-click to export</td>
</tr>
<tr>
<td>external_connection</td>
<td>Conduit Endpoint</td>
<td>Double-click to export</td>
</tr>
</tbody>
</table>

**Messages**

- **29 Warnings**
  - Module dependency loop involving: "pcie_hard_ip_0" (altera_pcie_hard_ip_12.1), "hexport" (altera_a...
  - System.pchellocore

**Desired Functionality**

- **DE2i-150 Hardware and Software Development Environment PCIex Basics**
- **HelloPCI Design**
- **FPGA Design**
PCI configured with single, 32K, non-prefetchable 32-bit memory space
User space includes switch/LED PIOs, PCI control
PCI Device Driver Data Structure

static struct pci_device_id pci_ids[] = {
    { PCI_DEVICE(0x1172, 0x0004), },
    { 0, } };
MODULE_DEVICE_TABLE(pci, pci_ids);

static int pci_probe(struct pci_dev *dev,
        const struct pci_device_id *id);
static void pci_remove(struct pci_dev *dev);

static struct pci_driver pci_driver = {
    .name = "alterahello",
    .id_table = pci_ids,
    .probe = pci_probe,
    .remove = pci_remove,
};

- Kernel expects standard interface \(\text{pci\_driver}\) to load/remove device, and to describe PCI device \(\text{pci\_device\_id}\)
- Altera’s vendor ID is 0x1172
- This FPGA is given device id 0x4
static int pci_probe(struct pci_dev *dev, const struct pci_device_id *id) {
    int vendor;
    int retval;
    unsigned long resource;
    retval = pci_enable_device(dev);
    if (pci_get_revision(dev) != 0x01) {
        printk(KERN_ALERT "altera_driver: cannot find pci device\n");
        return -ENODEV;
    }
    pci_read_config_dword(dev, 0, &vendor);
    printk(KERN_ALERT "altera_driver: Found Vendor id: \%x\n", vendor);
    resource = pci_resource_start(dev, 0);
    printk(KERN_ALERT "altera_driver: Resource start at bar 0: \%lx\n", resource);
    hexport = ioremap_nocache(resource + 0XC000, 0x20);
    inport = ioremap_nocache(resource + 0XC020, 0x20);
    return 0;
}
Character Device Driver Data Structure

```c
static int char_device_open ( struct inode * , struct file * );
static int char_device_release ( struct inode * , struct file * );
static ssize_t char_device_read ( struct file * , char * , size_t , loff_t * );
static ssize_t char_device_write ( struct file * , const char * , size_t , loff_t * );

static struct file_operations file_opts = {
    .read = char_device_read,
    .open = char_device_open,
    .write = char_device_write,
    .release = char_device_release
};
```

- Kernel expects standard interface (`file_opts`) to open/close device, write data to/read data from device
- User needs to provide each function
static ssize_t char_device_read(struct file *filep, char *buf, size_t len, loff_t *off) {

    short switches;
    size_t count = len;
    while (len > 0) {
        switches = ioread16(inport);
        put_user(switches & 0xFF, buf++);
        put_user(((switches >> 8) & 0xFF, buf++);
        len -= 2;
    }
    return count;
}
static ssize_t char_device_write(struct file *filep,
        const char *buf,
        size_t len,
        loff_t *off) {

    char *ptr = (char *) buf;
    size_t count = len;
    short b = 0;
    while (b < len) {
        unsigned k = *((int *) ptr);
        ptr += 4;
        b += 4;
        iowrite32(k, hexport);
    }

    return count;
}
printk: **Kernel-level printf**

- Debug messages, status, .. are logged in `/var/log/messages`
- **Display with dmesg:**
  ```bash
  BOARD$ dmesg
  ...
  altera_driver: Found Vendor id: 41172
  altera_driver: Resource start at bar 0: 80200000
  altera_driver: char+pci drivers registered.
  altera_driver: opened 1 time(s)
  ```
Device Driver as Kernel Module

Kernel Modules are runtime-loadable elements of code

Can be inserted/removed with `insmod`/`rmmod`

Can be inspected with `lsmod`

```bash
BOARD$ lsmod
Module Size Used by
altera_driver 2158 0
iptable_nat 3685 0
...
```

Need to be compiled/linked against a specific kernel

```bash
BOARD$ uname -rs
Linux 3.0.32-yocto-standard
```
```c
#include <unistd.h>
#include <fcntl.h>

unsigned char hexdigit[] = {
    0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07,
    0x7F, 0x6F, 0x77, 0x7C, 0x39, 0x5E, 0x79, 0x71};

int main() {
    int i, j, k;

    int dev = open("/dev/de2i150_altera", O_RDWR);

    for (i=0; i>-1; i++) {
        read(dev, &j, 4);
        k = hexdigit[j & 0xF]
            | (hexdigit[(j >> 4) & 0xF] << 8)
            | (hexdigit[(j >> 8) & 0xF] << 16)
            | (hexdigit[(j >> 12) & 0xF] << 24);
        k = ~k;
        write(dev, &k, 4);
    }

    close(dev);
    return 0;
}
```
Useful References

- Altera Cyclone IV Handbook

- Altera PCI IP Compiler User Guide

- PCI Express System Architecture (Safari Books Online)

- Linux Device Drivers 3th Edition
  - http://lwn.net/Kernel/LDD3/

- Yocto Quick Start and Manual
  - https://www.yoctoproject.org/documentation/current
  - https://www.yoctoproject.org/tools-resources/presentations