Agenda

• How to use ESDC FPGA Devices in Linux
• How to develop a Linux Char Device Driver
• How to develop a PCIe device driver
1. ESDC FPGA Drivers Usage Guide

• 1.1 ESDC FPGA devices

• 1.2 Drivers and Usage
1.1 ESDC FPGA Devices

- Intel® Atom™ Processor
- Altera® Arria® II EP2AGX6XX FPGA

- PCIe
- COMe Connector 2
- I/O Bank 7
- I/O Bank 6
- 12-bit ADC 400Ksps
- 8-bit ADC 10Msps
- 10-bit DAC 10Msps
- Analog Video Input
- 8 Leds
- 4 Switches
- 4 Buttons
- Video Decoder
- PHY
- Altera TSE
1.1 ESDC FPGA Devices

- 10-bit DAC
- 12-bit ADC
- 8-bit ADC
- Video Input
- TSE Ethernet
- 4 Buttons
- 4 Switches
- 8 LEDs

Altera® Arria® II
EP2AGXE6XX FPGA
1.1 ESDC FPGA Devices

- 8 LEDs
- 4 Switches
- 4 Buttons
- one 8-bit ADC
- one 12-bit ADC
- one 10-bit DAC
- one Video Interface
- one 10/100/1000M Ethernet
1.2 Drivers and Usage

- 8 LEDs
  - PCB silk: D19~D26
  - PCIe bar0: 0x02210040
  - Driver: zlge6x5c-leds.c
  - Sysfs interface:
    /sys/class/leds/zlge6x5c:red:{0-7}
  - Operation: write \texttt{n} to \texttt{brightness},
    \texttt{ex: $echo \texttt{n} > brightness (1:Lighten, 0:Off)$}
1.2 Drivers and Usage

- 4 SWITCHes
  - PCB silk: S1
  - PCIe bar0: 0x02210060
  - Driver: zlge6x5c-switch.c
  - Sysfs interface:
    /sys/bus/pci/devices/0000:01:00.0/zlge6x5c-switch
  - Operation: read from switch, ex: $ cat switch
1.2 Drivers and Usage

- 4 BUTTONs
  - PCB silk: SW3~SW6
  - PCIe bar0: 0x02210080
  - Driver: zlge6x5c-btn.c
  - System Input Device, Default Key Value: arrow up, down, left, right.
1.2 Drivers and Usage

• 8-bit ADC
  – PCB silk: J4
  – PCIe bar0: 0x02210010
  – Driver: zlge6x5c-ad-da.c
  – Sysfs interface:
    /sys/bus/pci/devices/0000:01:00.0/zlge6x5c-adc8bit
1.2 Drivers and Usage

- **8-bit ADC**
  - Operation: read from `value`, ex: `$ cat value`
  - Equation to calculate voltages:
    \[
    \text{volt} = \frac{\text{value} \times 3.3}{255} \text{ (V)}
    \]
    - Ex: `value = 120 \rightarrow \text{volt} = 1.55\text{V}`
• 12-bit ADC
  – PCB silk: J3
  – PCIe bar0: 0x02210000
  – Driver: zlge6x5c-ad-da.c
  – Sysfs interface:
    /sys/bus/pci/devices/0000:01:00.0/zlge6x5c-adc12bit
1.2 Drivers and Usage

- 12-bit ADC
  - Operation: read from value, ex: \$ cat value
  - Equation to calculate voltages:
    \[ \text{volt} = \text{value} \times \frac{3.3}{4095} \text{ (V)} \]
  - Ex: value = 1500 \( \rightarrow \) volt = 1.209V
1.2 Drivers and Usage

• 10-bit DAC
  – PCB silk: J7
  – PCIe bar0: 0x02210020
  – Driver: zlge6x5c-ad-da.c
  – Sysfs interface:
    /sys/bus/pci/devices/0000:01:00.0/zlge6x5c-dac10bit
1.2 Drivers and Usage

• 10-bit DAC
  – Operation: write \( n \) to \textit{value}, ex: \$ echo \( n \) > \textit{value}
  – Equation for voltage output:
    \[
    n = \frac{((\text{volt} \times 1024) + 1023)}{2}
    \]
  – volt range: \([-1, +1]\)
  – Ex: assume we want \( \text{volt} = 0.5V \) \( \rightarrow \) \( n = 767 \) or 768
1.2 Drivers and Usage

• Video Interface
  – PCB silk: J10
  – PCIe bar0: 0x02214000 ~ 0x0221401F
  – Driver: zlge6x5c-video.c
  – Operation: V4L2 Standard
1.2 Drivers and Usage

- 10/100/1000M Ethernet (Altera TSE)
  - PCB silk: LAN2
  - PCIe bar0: 0x0220A000 ~ 0x0220A3FF
  - Driver: atse.c
  - Operation: eth1, Linux/POSIX Socket API
2. Linux Char Device Drivers

• 2.1 Linux Devices
• 2.2 Linux Device Drivers
• 2.3 How to Write a Char Device Driver?
2.1 Linux Devices

[Diagram showing the structure of Linux devices with categories such as User APP, User space, System Call Interface, Kernel space, Device Control, File System, Networking, Char Drivers, Block Drivers, IF Drivers, Consoles, etc., and Hardware.]
2.1 Linux Devices

• Character(char) Devices
  – The Devices can be accessed as a stream of bytes
  – Most of the devices are char devices, such as ttyS0/console/tty1 etc
2.1 Linux Devices

• Block Devices
  – The block devices are accessed by file system
  – A block device can host a file system
  – A block device can only handle I/O operations that transfer one or more whole blocks, usually 512(or 512*2^n) bytes
  – Such as hard disk, tape, floppy
2.1 Linux Devices

• Network Devices
  – The device that is able to exchange data with other hosts
  – Can be a hardware device (such as eth0) or a pure software device (such as loopback device)
  – No device node in /dev
  – Accessed by socket operation
2.2 Linux Device Drivers
What’s the Linux Driver?

- Driver is a “black box” between hardware and user program (user program cannot directly access the hardware)
- Driver hides all the details of hardware, provides a uniform programming interface
- Driver can also expand kernel function
- Run in kernel space
2.2 Linux Device Drivers

• How to Get a Linux Driver?
  – make modules (choose [M] in kernel configuration)
  – make modules_install (install to local host)
  – Build a kernel module outside the kernel sources tree
  – Kernel module: xxx.ko (2.6 kernel and above)
2.2 Linux Device Drivers

• How to use a Linux Driver?
  – Insert a driver module to a running kernel: `insmod xxx.ko` or `modprobe xxx`
  – Remove a driver module from a running kernel: `rmmod xxx(.ko)`
  – List all the inserted modules: `lsmod`
2.3 How to Write a Linux Char Driver

• Before coding
  – Kernel module and build
  – Understand the hardware
• Important Data Structures
  – struct file_operations
  – See <linux/fs.h>
2.3 How to Write a Linux Char Driver

```c
struct file_operations {
    struct module *owner;
    loff_t (*llseek) (struct file *, loff_t, int);
    ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
    ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
    ssize_t (*aio_read) (struct kiocb *, const struct iovec *, unsigned long, loff_t);
    ssize_t (*aio_write) (struct kiocb *, const struct iovec *, unsigned long, loff_t);
    int (*readdir) (struct file *, void *, filldir_t);
    unsigned int (*poll) (struct file *, struct poll_table_struct *);
    long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
    long (*compat_ioctl) (struct file *, unsigned int, unsigned long);
    int (*mmap) (struct file *, struct vm_area_struct *);
    int (*open) (struct inode *, struct file *);
    int (*flush) (struct file *, fl_owner_t id);
    int (*release) (struct inode *, struct file *);
    .....}
```
2.3 How to Write a Linux Char Driver

- **Member brief introduction**
  - *owner*
    - NOT a operation, point to the “owner”, usually initialized to “THIS_MODULE”
  - *open*
    - Open the device. It is invoked when open the device.
  - *release*
    - Close the device. It is invoked when close the device.
2.3 How to Write a Linux Char Driver

• Member brief introduction
  – *read
    • Invoked by read(), read data from the device
    • Return the number of read bytes to system call
      read() when success, otherwise -EINVAL
  – *write
    • Invoked by write(), send data to device
    • Return the number of written byte to system call
      write() when success, otherwise -EINVAL
2.3 How to Write a Linux Char Driver

- A sample char driver file_operations
  - A driver for LEDs, only implement open/release/ioctl method

```c
static struct file_operations tp_led_fops = {
    .owner   = THIS_MODULE,
    .write   = tp_led_write,
    .open    = tp_led_open,
    .release = tp_led_release,
};
```
2.3 How to Write a Linux Char Driver

• Important Data Struct
  – struct cdev
  – See `<linux/cdev.h>`

```c
struct cdev {
    struct kobject kobj;
    struct module *owner;
    const struct file_operations *ops;  // fops
    struct list_head list;
    dev_t dev;
    unsigned int count;
};
```
2.3 How to Write a Linux Char Driver

- Char device registration
  - `cdev_init()` & `cdev_add()`
    ```c
    void cdev_init(struct cdev *cdev, struct file_operations *fops);
    int cdev_add(struct cdev *dev, dev_t num, unsigned int count);
    ```
  - `cdev_del()`
    ```c
    void cdev_del(struct cdev *dev);
    ```
3. PCIe Device Driver Development

• 3.1 Related command
• 3.2 Accessing PCI Regions
• 3.3 PCI driver framework
• 3.4 MFD Introduction
• 3.5 Example – Driver for LEDs on FPGA
3.1 Related command

- `lspci` -- list all PCI(e) devices

```plaintext
00:1f.0 ISA bridge: Intel Corporation Tunnel Creek LPC Bridge (rev 02)  
01:00.0 Class ff00: Altera Corporation Device e001 (rev 01)  
03:00.0 PCI bridge: Intel Corporation Topcliff PCI Express Port (rev 01)  
04:00.0 Class ff00: Intel Corporation Topcliff Packet Hub (rev 01)
```

- `lspci -x` (more details)

```plaintext
01:00.0 Class ff00: Altera Corporation Device e001 (rev 01)  
  72 11 01 e0 07 04 10 00 01 00 00 ff 10 00 00 00  
  00 00 00 d0 00 00 00 00 00 00 00 00 00 00 00 00 00  
  00 00 00 00 00 00 00 72 11 04 00  
  00 00 00 00 50 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00
```
3.1 Related command

- `/sys/bus/pci/devices/0000:01:00.0/config`

```bash
[root@fedora 0000:01:00.0]# pwd
/sys/bus/pci/devices/0000:01:00.0
[root@fedora 0000:01:00.0]# hexdump config
00000000 1172 e001 0407 0010 0001 ff00 0010 0000
00000010 0000 d000 0000 0000 0000 0000 0000 0000
00000020 0000 0000 0000 0000 0000 0000 0000 0000
00000030 0000 0000 0000 0000 0000 0000 0000 010b 0000
00000040 0000 0000 6160 0200 0000 0000 0000 0000 0000
00000050 7805 0081 300c fee0 0000 0000 4171 0000
00000060 0000 0000 0000 0000 0000 0000 7811 0000 0000
```
3.2 Accessing PCI Regions

- Configuration Space
  - read config
    - `pci_read_config_[byte|word|dword](struct pci_dev *pdev, int offset, int *value)`
  - write config
    - `pci_write_config_[byte|word|dword](struct pci_dev *pdev, int offset, int value)`
3.2 Accessing PCI Regions

- I/O Space
  - I/O read/write
    - \texttt{pci_resource\_\{start\mid len\mid end\mid flags\}}(\texttt{struct pci\_dev \*pdev, int bar})
    - request\_region(start, n, name)
    - inl(addr)/outl(value, addr)
3.2 Accessing PCI Regions

- Memory Space
  - Memory
    - `pci_resource_[start|len|end|flags](struct pci_dev *pdev, int bar)`
    - `request_mem_region(mmio_base, mmio_length, "name")`
    - `pci_iomap()`
    - `ioread32(addr) / iowrite32(value, addr)`
3.3 PCI driver Framework

- struct pci_driver

struct pci_driver {
    struct list_head node;
    const char *name;
    const struct pci_device_id *id_table; /* must be non-NULL for probe to be called */
    int (*probe)(struct pci_dev *dev, const struct pci_device_id *id); /* New device inserted */
    void (*remove)(struct pci_dev *dev); /* Device removed (NULL if not a hot-plug capable driver) */
    int (*suspend)(struct pci_dev *dev, pm_message_t state); /* Device suspended */
    int (*suspend_late)(struct pci_dev *dev, pm_message_t state);
    int (*resume_early)(struct pci_dev *dev);
    int (*resume)(struct pci_dev *dev); /* Device woken up */
    void (*shutdown)(struct pci_dev *dev);
    struct pci_error_handlers *err_handler;
    struct device_driver driver;
    struct pci_dynids dynids;
};
3.3 PCI driver Framework

• pci_device_id

```c
struct pci_device_id {
    __u32 vendor, device;   /* Vendor and device ID or PCI_ANY_ID*/
    __u32 subvendor, subdevice; /* Subsystem ID's or PCI_ANY_ID */
    __u32 class, class_mask; /* (class,subclass,prog-if) triplet */
    kernel_ulong_t driver_data; /* Data private to the driver */
};
```

• create pci_device_id table
  - PCI DEVICE(vend, dev)
  - PCI DEVICE CLASS(dev_class, dev_class_mask)

```c
static struct pci_device_id zlge6x5c_pci_ids[] = {
    { PCI DEVICE(0x1172, 0xe001) },
    { 0, },
};
```
3.3 PCI driver Framework

- PCI probe function
  - enable PCIe device
    - \texttt{pci\_enable\_device(...)}
  - request\_mem\_region \& ioremap
  - enable PCIe msi
    - \texttt{pci\_enable\_msi(...)}
    - request IRQ
static int __devinit zlge6x5c_pci_probe(struct pci_dev *pdev, const struct pci_device_id *id)
{
    int ret = 0;
    resource_size_t mapbase;

    ret = pci_enable_device(pdev);
    if (ret) {......}
    mapbase = pci_resource_start(pdev, 0);
    if (!mapbase) {......}
    ctrl_mapbase = mapbase + CHIPCTRLOFFSET;
    if (!request_mem_region(ctrl_mapbase, CHIPCTRLSIZE, "zlge6x5c-ctrl")) {
        dev_err(&pdev->dev, "CTRL request_mem_region failed\n");
        goto err_ctrl_req_mem;
    }
    ctrl_membase = ioremap(ctrl_mapbase, CHIPCTRLSIZE);
    if (!ctrl_membase) {......}
    ret = pci_enable_msi_block(pdev, 1);
    if (ret) {......}

    ......
3.3 PCI driver Framework

• register/unregister pci driver
  – register
    • pci_register_driver()
  – unregister
    • pci_unregister_driver()
static int __devinit zlge6x5c_pci_probe(struct pci_dev *pdev, 
   const struct pci_device_id *id) 
{
......
} 
static void __devexit zlge6x5c_pci_remove(struct pci_dev *pdev) 
{
......
} 
static struct pci_device_id zlge6x5c_pci_ids[] = {
   { PCI_DEVICE(0x1172, 0xe001) },
   { 0, },
}; 
static struct pci_driver zlge6x5c_pci_driver = {
   .name = "zlge6x5c-mfd",
   .id_table = zlge6x5c_pci_ids,
   .probe = zlge6x5c_pci_probe,
   .remove = zlge6x5c_pci_remove,
}; 
static int __init zlge6x5c_init(void) 
{
   return pci_register_driver(&zlge6x5c_pci_driver);
} 
static void __exit zlge6x5c_exit(void) 
{
   pci_unregister_driver(&zlge6x5c_pci_driver);
} 
module_init(zlge6x5c_init);
module_exit(zlge6x5c_exit);
3.4 MFD Introduction

- 10-bit DAC
- 12-bit ADC
- 8-bit ADC
- Video Input
- TSE Ethernet
- 4 Buttons
- 4 Switches
- 8 LEDs

Altera® Arria® II
EP2AGXE6XX FPGA
3.4 MFD Introduction

- MFD (Multi-function devices)
- Reduce the complexity of the PCI driver
  - PCIE driver only register sub devices and resources
  - Register subdevices drivers
3.4 MFD Introduction

CODE: resources and mfd_cell

```c
static const __devinitconst struct resource led_resources[] = {
    {
        .start = LED_OFFSET,
        .end = LED_END,
        .flags = IORESOURCE_MEM,
    },
};

static __devinitdata struct mfd_cell zlge6x5c_cells_bar0[] = {
    {
        .name = "zlge6x5c-led",
        .num_resources = ARRAY_SIZE(led_resources),
        .resources = led_resources,
    },
    ....
};
```
3.4 MFD Introduction

- MFD – Multifunction device
  - mfd_add_devices(…)

```c
extern int mfd_add_devices(struct device *parent, int id,
                          struct mfd_cell *cells, int n_devs,
                          struct resource *mem_base,
                          int irq_base);
```

- MFD register platform devices
3.4 MFD Introduction

CODE: mfd add devices

```c
ret = mfd_add_devices(&pdev->dev, -1,
                      zlge6x5c_cells_bar0, ARRAY_SIZE(zlge6x5c_cells_bar0),
                      &pdev->resource[0], pdev->irq);

if (ret) {
    dev_err(&pdev->dev, "mfd_add_devices failed: %d\n", ret);
    goto err_mfd;
}
```
3.5 Example – Driver for LEDs on FPGA

CODE: framework

```c
static int zlge6x5c_led_probe(struct platform_device *pdev)
{......}
static int zlge6x5c_led_remove(struct platform_device *pdev)
{......}
static struct platform_driver zlge6x5c_led_driver = {
    .probe = zlge6x5c_led_probe,
    .remove = zlge6x5c_led_remove,
    .driver = {
        .name = "zlge6x5c-led",
    },
};
static int __init zlge6x5c_led_init(void)
{
    return platform_driver_register(&zlge6x5c_led_driver);
}
static void __exit zlge6x5c_led_exit(void)
{
    platform_driver_unregister(&zlge6x5c_led_driver);
}
module_init(zlge6x5c_led_init);
module_exit(zlge6x5c_led_exit);
```
3.5 Example – Driver for LEDs on FPGA

```c
static int zlge6x5c_led_probe(struct platform_device *pdev)
{
    struct resource *res;
    int I, j, ret;
    res = platform_get_resource(pdev, IORESOURCE_MEM, 0);
    if (!res) {......}
    if (!request_mem_region(res->start, resource_size(res), pdev->name)) {
        ......}
    led_membase = ioremap(res->start, resource_size(res));
    if (!led_membase) {......}
    for (i = 0; i < EP2AGX_LEDS; i++) {
        ret = led_classdev_register(&pdev->dev, &zlge6x5c_led[i]);
        if (ret < 0) {
            for (j = i; j >= 0; j--) {
                led_classdev_unregister(&zlge6x5c_led[i]);
            }
            goto led_register_failed;
        }
    }
}
......
```
3.5 Example – Driver for LEDs on FPGA

CODE: LED_set

```c
static void zlge6x5c_led_set (struct led_classdev *led_cdev,
                               enum led_brightness value)
{
    int c;
    u32 val;
    c = led_cdev->name[strlen(led_cdev->name)-1] - 0x30;
    val = ioread32(led_membase);
    val &= ~(1<<c);
    if (value)
        val |= (1<<c);
    iowrite32(val, led_membase);
}
```
References

- Linux Device Drivers (3rd Edition), O’REILLY
- Essential Linux Device Drivers, Prentice Hall PTR
Intel Cup Embedded System Design Contest