OUTLINE

- Prepare the Hardware
- Set up Environment
- Write a Camera Driver
- Solve Remaining Problems
PREPARE THE HARDWARE

- TI OMAP4 Pandaboard
- Aptina AR0832 Image Sensor
- Ominivision OV5650 Image Sensor
- Adapters and Connectors
- Datasheets
PREPARE THE HARDWARE

System Setup
OUTLINE

- Prepare the Hardware
- Set up Environment
- Write a Camera Driver
- Solve Remaining Problems
SET UP ENVIRONMENT

- Install Ubuntu 12.04 on Pandaboard

- Powerful and fast server for cross compilation
  - 32-core
  - 5 min on server vs. 5 hours on Pandaboard

- Understand Data Path and Interfaces
DATA PATH

Camera Serial Interface (CSI-2)

Image Sub-System (ISS)  Image Sensor
INTERFACE: CSI-2

- N data lanes and one data clock lane
  - Each lane is a pair of +/- lines.

- I2C standard control interface (CCI)
  - SCL
  - SDA
INTERFACE: Hardware Level

- Pandaboard Camera Expansion Connector (J-17)
  - 5 CSI-2 lanes (including one CLK lane)
  - I2C control lines (SCL, SDA)
  - Other lines for battery power, camera clock, GPIO…

- Image Sensor
  - 2 CSI-2 data lanes + 1 CSI-2 clock lane
  - I2C control lines (as slave)
  - Powered from on board battery
OUTLINE

- Prepare the Hardware
- Set up Environment
- Write a Camera Driver
- Solve Remaining Problems
WRITE A CAMERA DRIVER

- Describe the camera to the board

- Write the driver file
  - a. Voltage Regulator and Clock
  - b. I2C Bus
  - c. V4L2
  - d. Driver Framework
  - e. Complete Template

- Implement the driver to Linux Kernel
DESCRIBE THE CAMERA TO THE BOARD

- Create `ar0832.h` to describe camera’s platform information
  - `struct ar0832_platform_data` containing fields about voltage regulator, clock, power on/off methods

- Edit `board-omap4panda-camer.c`
  - Assign values or functions to `ar0832_platform_data`’s fields.
  - Tells the board about camera’s name, slave address and assign one I2C bus to it.
  - Describe CSI-2 interface connection.
DESCRIBE THE CAMERA TO THE BARD

- Sample Code 1:

```c
static struct ar0832_platform_data ar0832_platform_data = {
    .reg_vaa = NULL,
    .reg_vdd = NULL,
    .clk_xvclk = "auxclk1_ck",
    .pre_poweron = NULL,
    .post_poweroff = NULL,
};
```

Assign Value for Platform Data
DESCRIBE THE CAMERA TO THE BARD

Sample Code II:

```c
// Add for AR0832 Image Sensor
#define AR0832_I2C_ADDRESS (0x54)

static struct i2c_board_info ar0832_camera_i2c_device = {
    I2C_BOARD_INFO("ar0832", AR0832_I2C_ADDRESS),
    .platform_data = &ar0832_platform_data,
};

static struct iss_subdev_i2c_board_info ar0832_camera_subdevs[] = {
    {
        .board_info = &ar0832_camera_i2c_device,
        .i2c_adapter_id = 3,
    },
    { NULL, 0, },
};
```
DESCRIBE THE CAMERA TO THE BARD

- Sample Code III:

```cpp
// Add for AR0832 Image Sensor
{
    .subdevs = ar0832_camera_subdevs,
    .interface = ISS_INTERFACECSI2A_PHY1,
    .bus = { .csi2 = {
        .lanecfg = {
            .clk = {
                .pol = 0,
                .pos = 1,
            },
            .data[0] = {
                .pol = 0,
                .pos = 2,
            },
            .data[1] = {
                .pol = 0,
                .pos = 3,
            },
        } },
    } },
}
```
WRITE A CAMERA DRIVER

- Describe the camera to the bard

- Write the driver file
  - a. Voltage Regulator and Clock
  - b. I2C Bus
  - c. V4L2
  - d. Driver Framework
  - e. Complete Template

- Implement the driver to Linux Kernel
A. VOLTAGE REGULATOR AND CLOCK

- Use pre-defined regulator and clock symbols from the board.

- Regulator: regulator_get, regulator_put
  regulator_enable, regulator_disable

- Clock: clk_get, clk_put
  clk_enable, clk_disable

- Both regulator and clock are hardwired in our case.
B. I2C BUS

- I2C is used to read/write values from/to camera’s registers to achieve various controls.

- Use kernel API `i2c_transfer` for communication.

- Based on datasheet to determine the bit sequence to be send.
B. I2C BUS

```c
static int ar8032_reg_read_16(struct i2c_client *client, u16 reg, u16 __val)
{
    int ret;
    u8 data[4] = {0};
    struct i2c_msg msg = {
        .addr = client->addr,
        .flags = 0,
        .len = 2,
        .buf = data,
    };

    data[0] = (u8)(reg >> 8);
    data[1] = (u8)(reg & 0xff);

    ret = i2c_transfer(client->adapter, &msg, 1);
    if (ret < 0)
        goto err;

    msg.flags = I2C_M_RD;
    msg.buf = data + 2;
    ret = i2c_transfer(client->adapter, &msg, 1);
    if (ret < 0)
        goto err;

    __val = ((data[2] << 8) | data[3]);
    printk("%s\004xh\004xh = \004x\004xh\n", __FUNCTION__, reg, __val);
    return __val;
}
```

![I2C Read Diagram]
B. I2C BUS

```c
static int ar0032_reg_write_16(struct i2c_client *client, u16 reg, u16 val)
{
    int ret;
    unsigned char data[4] = { (u8)(reg >> 8), (u8)(reg & 0xff), (u8)(val >> 8), (u8)(val & 0xff) };
    struct i2c_msg msg = {
        .addr = client->addr,
        .flags = 0,
        .len = 4,
        .buf = data,
    };

    ret = i2c_transfer(client->adapter, &msg, 1);
    if (ret < 0) {
        dev_err(client->dev, "Failed writing register 0x%02x!\n", reg);
        return ret;
    }

    printk("%s(0x%04x) = 0x%04x\n", __FUNCTION__, reg, val);
    usleep_range(5000, 7500);
    return 0;
}
```

I2C Write
C. V4L2

- V4L2, the second version of Video for Linux, is a video capture application programming interface for Linux.

- Closely integrated with the Linux kernel; a bridge between kernel space and user space. It takes care of device register and file handle in the kernel space.

- Regard ISS as v4l2 device and image sensor as a v4l2 sub-device.
C. V4L2

1. V4L2 Sub-device Internal Operations
   - .registered: Called when camera is registered.
     Read some registers’ values to check connection.
   - .open: Called when the camera node is opened by an application.
     Initialize the format information (frame size, code…)
   - .closed: Called when the camera node is closed.
C. V4L2

```c
static int ar0832_open(struct v4l2_subdev *subdev, struct v4l2_subdev_fh *fh)
{
    struct v4l2_mbus_framefmt *format;

    format = v4l2_subdev_get_try_format(fh, NULL);
    format.code = V4L2_MBUS_FMT_SGRBG10_1X10;
    format.width = ar0832_frame_sizes[MODE_3264X2448].width;
    format.height = ar0832_frame_sizes[MODE_3264X2448].height;
    format.field = V4L2_FIELD_NONE;
    format.colorspace = V4L2_COLORSPACE_SRGB;

    return 0;
}
```
2. V4L2 Sub-device Operations

- `v4l2_subdev_ops`
  - `v4l2_subdev_pad_ops`
    - `enum_mbus_code`
    - `enum_frames_code`
    - `get_fmt`
    - `set_fmt`
  - `v4l2_subdev_video_ops`
    - `s_stream`
  - `v4l2_subdev_core_ops`
    - `s_power`
C. V4L2

- **v4l2_subdev_pad_ops**
  - For cameras that support multiple frame size and image sample format, these operations allow users to choose from the options.

- **v4l2_subdev_video_ops**
  - `.s_stream` is called when streaming starts. It writes different configuration values to camera’s registers based on the chosen frame size and format.

- **v4l2_subdev_core_ops**
  - `.s_power` puts camera in power saving mode (on==0) or normal operation mode (on==1).
C. V4L2

3. V4L2 Control

Implementing the controls that are compatible with the image sensor and exporting those control symbols to user space applications.

Consists of two main objects:

- v4l2_ctrl: Describes the control properties and keeps track of the control’s value.
- v4l2_ctrl_handler: Keeps track on controls.
C. V4L2

3.1: Initializing v4l2 controls:

```c
v4l2_ctrl_handler_init(&foo->ctrl_handler, nr_of_controls);
v4l2_ctrl_new_std(&foo->ctrl_handler, &foo_ctrl_ops,
    V4L2_CID_BRIGHTNESS, 0, 255, 1, 128);
v4l2_ctrl_new_std(&foo->ctrl_handler, &foo_ctrl_ops,
    V4L2_CID_CONTRAST, 0, 255, 1, 128);
v4l2_ctrl_new_std_menu(&foo->ctrl_handler, &foo_ctrl_ops,
    V4L2_CID_POWER_LINE_FREQUENCY,
    V4L2_CID_POWER_LINE_FREQUENCY_60HZ, 0,
    V4L2_CID_POWER_LINE_FREQUENCY_DISABLED);
```
3.2: Hook v4l2 controls to the driver

```c
struct foo_dev {
    ...
    struct v4l2_subdev sd;
    ...
    struct v4l2_ctrl_handler ctrl_handler;
    ...
};
foo->sd.ctrl_handler = &foo->ctrl_handler;
```
C.V4L2

3.3: Implementing v4l2 controls:

```c
static const struct v4l2_ctrl_ops foo_ctrl_ops = {
    .s_ctrl = foo_s_ctrl,
};

static int foo_s_ctrl(struct v4l2_ctx *ctx)
{
    struct foo *state = container_of(struct->handler, struct foo, ctx_handler);
    switch (ctrl->id) {
    case V4L2_CID_BRIGHTNESS:
        write_reg(0x123, ctrl->val);
        break;
    case V4L2_CID_CONTRAST:
        write_reg(0x456, ctrl->val);
        break;
    }
    return 0;
}
```
D. DRIVER FRAMEWORK

- **Standard I2C Device Driver Code**
  - **loadable module**: module_init/module_exit
  - **I2C Device**: i2c_add_driver/i2c_del_driver
  - **struct i2c_driver**
    - **.id_table**: Used to identify the supported chip based on name and I2C slave address; Must be same as described in board-omap4panda-camera.c file.
    - **.probe**: Function called when supported chip is found and the driver is added.
    - **.remove**: Function called before driver is deleted.
D. DRIVER FRAMEWORK

```c
static const struct i2c_device_id ar0832_id[] = {
    { "ar0832", 0 },
};
MODULE_DEVICE_TABLE(i2c, ar0832_id);

static struct i2c_driver ar0832_i2c_driver = {
    .driver = {
        .name = "ar0832",
        .probe = ar0832_probe,
        .remove = ar0832_remove,
        .id_table = ar0832_id,
    },
};

static int __init ar0832_mod_init(void)
{
    return i2c_add_driver(&ar0832_i2c_driver);
}

static void __exit ar0832_mod_exit(void)
{
    i2c_del_driver(&ar0832_i2c_driver);
}

module_init(ar0832_mod_init);
module_exit(ar0832_mod_exit);
MODULE_DESCRIPTION("Aptina AR0832 Camera driver");
MODULE_AUTHOR("Kyle Xu <sx3@rice.edu>");
MODULE_LICENSE("GPL");
```
D. DRIVER FRAMEWORK

- Probe and Remove Methods
  - Probe: Initialize format image information that contained as fields in device specific structure.
    Enable v4l2 interface: v4l2_i2c_subdev_init and sets v4l2 internal operations.
  - Remove: Disable v4l2 interface:
    v4l2_device_unregister_subdev
    v4l2_ctrl_handler_free
D. DRIVER FRAMEWORK

```c
ar0832->format.code = V4L2_MBUS_FMT_SGRBG10_1X10;
ar0832->format.width = ar0832_frame_sizes[MODE_3264X2448].width;
ar0832->format.height = ar0832_frame_sizes[MODE_3264X2448].height;
ar0832->format.field = V4L2_FIELD_NONE;
ar0832->format.colorspace = V4L2_COLORSPACE_SRGB;
// More initialization here
v4l2_i2c_subdev_init(&ar0832->subdev, client, &ar0832_subdev_ops);
ar0832->subdev.internal_ops = &ar0832_subdev_internal_ops;
ar0832->subdev.flags |= V4L2_SUBDEV_FL_HAS_DEVNODE;
// ar0832->subdev.entity.type = MEDIA_ENT_T_V4L2_SUBDEV_SENSOR;
```

```c
static int ar0832_remove(struct i2c_client *client)
{
    struct v4l2_subdev *subdev = i2c_get_clientdata(client);
    struct ar0832 *ar0832 = to_ar0832(subdev);

    v4l2_ctrl_handler_free(&ar0832->ctrl_handler);
ar0832_put_resources(ar0832);
    media_entity_cleanup(&subdev->entity);
v4l2_device_unregister_subdev(subdev);
kfree(ar0832);
return 0;
}
```
WRITE A CAMERA DRIVER

- Describe the camera to the bard

- Write the driver file
  - a. Voltage Regulator and Clock
  - b. I2C Bus
  - c. V4L2
  - d. Driver Framework
  - e. Complete Template

- Implement the driver to Linux Kernel
IMPLEMENT THE DRIVER TO LINUX KERNEL

- Edit Kconfig files:
  - kernel_source/drivers/media/Kconfig
  - kernel_source/arch/arm/mach-omap2/Kconfig
IMPLEMENT THE DRIVER TO LINUX KERNEL

- Edit Makefile.
  - kernel_source/drivers/media/Makefile

  ```
  obj-$(CONFIG_VIDEO_OV5640) += ov5640.o
  obj-$(CONFIG_VIDEO_OV5650) += ov5650.o
  obj-$(CONFIG_VIDEO_AR0832) += ar0832.o
  ```

  Driver Makefile

- Cross-compile on server.

- Install and swap to the new kernel!
OUTLINE

- Prepare the Hardware
- Set up Environment
- Write a Camera Driver
- Solve Remaining Problems
SOLVE REMAINING PROBLEMS

- Compile and install the driver only without re-compile the entire kernel.
- Weird i2c slave address
- AR0832’s full register table
REFERENCES

- http://www.mipi.org/specifications/camera-interface#CSI2
- http://lxr.free-electrons.com/source/include/media/v4l2-subdev.h?v=3.4;a=arm
- Aptina AR0832 Data Sheet