Lab – Adding New Hardware To Android Device

Simulating new hardware

- Update the file external/qemu/hw/goldfish_timer.c

```
static int start=0,stop=10;
static int current=0;
static int intval = 5;
static int intcount=0;
static int ticks=0;
static int membuffer[0x7000];

static struct timer_state fpga_state = {
   .dev = {
      .name = "goldfish_fpga",
      .id = -1,
      .size = 0x8000,
      .irq_count = 1,
   }
};

static void goldfish_timer_tick(void *opaque)
{
   struct timer_state *s = (struct timer_state *)opaque;

   s->armed = 0;
   goldfish_device_set_irq(&s->dev, 0, 1);
   ticks++;
   if(ticks==TPERSEC)
   {
          current++;
          if(current == intval)
          {
                    intcount++;
                    fpga_state.armed = 0;
                    goldfish_device_set_irq(&fpga_state.dev, 0, 1);
          }
          if(current == stop)
                    current = start;
          ticks=0;
   }
}
```

```
static void  goldfish_fpga_save(QEMUFile*  f, void*  opaque)
{
    struct rtc_state*  s = opaque;

    qemu_put_be64(f, 33);
}

static int  goldfish_fpga_load(QEMUFile*  f, void*  opaque, int  version_id)
{
    struct  rtc_state*  s = opaque;

    if (version_id != GOLDFISH_RTC_SAVE_VERSION)
        return -1;

    /* this is an old value that is not correct. but that's ok anyway */
    s->now = qemu_get_be64(f);
    return 0;
}




static uint32_t goldfish_fpga_read(void *opaque, target_phys_addr_t offset)
{
    struct rtc_state *s = (struct rtc_state *)opaque;
    if(offset >= 0x1000 && offset <= 0x8000)
             return membuffer[offset - 0x1000];
    switch(offset) {
        case 0x0:
            return current;
        case 0x4:
            return start;
        case 0x8:
            return stop;
             case 0xc:
                 return intval;
             case 0x10:
                 return intcount;
        default:
            cpu_abort (cpu_single_env, "goldfish_fpga_read: Bad offset %x\n", offset);
            return 0;
    }
}
```

```
static void goldfish_fpga_write(void *opaque, target_phys_addr_t offset, uint32_t value)
{
    struct rtc_state *s = (struct rtc_state *)opaque;
    int64_t alarm;
    if(offset >= 0x1000 && offset <= 0x8000)
             membuffer[offset - 0x1000] = value;
    switch(offset) {
        case 0x4:
            start = value;
                 break;
        case 0x8:
            stop = value;
                 break;
        case 0xc:
             intval = value;
             break;
         case 0x10:
            goldfish_device_set_irq(&s->dev, 0, 0);
            break;
              case 0x14:
                          intcount=0;
                          break;
        default:
            cpu_abort (cpu_single_env, "goldfish_rtc_write: Bad offset %x\n", offset);
    }
}



static CPUReadMemoryFunc *goldfish_fpga_readfn[] = {
    goldfish_fpga_read,
    goldfish_fpga_read,
    goldfish_fpga_read
};

static CPUWriteMemoryFunc *goldfish_fpga_writefn[] = {
    goldfish_fpga_write,
    goldfish_fpga_write,
    goldfish_fpga_write
};
```

```
void goldfish_timer_and_rtc_init(uint32_t timerbase, int timerirq)
{
   timer_state.dev.base = timerbase;
   timer_state.dev.irq = timerirq;
   timer_state.timer = qemu_new_timer_ns(vm_clock, goldfish_timer_tick, &timer_state);
   goldfish_device_add(&timer_state.dev, goldfish_timer_readfn, goldfish_timer_writefn,
&timer_state);
   register_savevm( "goldfish_timer", 0, GOLDFISH_TIMER_SAVE_VERSION,
                     goldfish_timer_save, goldfish_timer_load, &timer_state);

   goldfish_device_add(&rtc_state.dev, goldfish_rtc_readfn, goldfish_rtc_writefn, &rtc_state);
   register_savevm( "goldfish_rtc", 0, GOLDFISH_RTC_SAVE_VERSION,
                     goldfish_rtc_save, goldfish_rtc_load, &rtc_state);


   fpga_state.dev.base = 0xff007000;
   fpga_state.dev.irq=6;
   goldfish_device_add(&fpga_state.dev, goldfish_fpga_readfn, goldfish_fpga_writefn, &fpga_state);
   register_savevm( "goldfish_fpga", 0, GOLDFISH_RTC_SAVE_VERSION,
                     goldfish_fpga_save, goldfish_fpga_load, &fpga_state);


}
```

- Build the rom and check for the new hardware:

```
cat /proc/iomem
```

Add a kernel Driver

- Add a new device driver to drivers/mfd/fpga.c

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/moduleparam.h>
#include <asm/uaccess.h>
#include <linux/fs.h>
#include <linux/gfp.h>
#include <linux/cdev.h>
#include <linux/kdev_t.h>
#include <linux/ioctl.h>
#include <linux/mm.h>
#include <asm/io.h>
#include <linux/device.h>
#include <linux/interrupt.h>

#define PHY_IO_ADD 0xff007000
#define PHY_MAP_ADD 0xff008000
#define GETVAL              10
#define GETSTART            20
#define GETSTOP             30
#define GETINTVAL           40
#define GETINTCOUNT         50
#define SETSTART            60
#define SETSTOP             70
#define SETINTVAL           80
#define CLRINTCOUNT         90
#define WAITFORINT          100

DECLARE_WAIT_QUEUE_HEAD(hq);
static int flag=0;

static int acme_count = 1;
static dev_t acme_dev;
static int *dev_id;
static struct cdev *acme_cdev;
static int int_counter = 0;

static int *regs;
static int device_ioctl(struct file*file,unsigned int num,
     unsigned long param)
{
   unsigned int r;
   int (*fptr)(int);
   int fr;
   switch(num)
   {
       case GETVAL:
                return regs[0];
       case GETSTART:
               return regs[1];
       case GETSTOP:
               return regs[2];
         case GETINTVAL:
               return regs[3];
       case GETINTCOUNT:
               return regs[4];
       case SETSTART:
               regs[1] = param;
               break;
       case SETSTOP:
               regs[2] = param;
               break;
       case SETINTVAL:
               regs[3] = param;
           break;
       case CLRINTCOUNT:
               regs[5] = 1;
               break;
       case WAITFORINT:
               wait_event(hq,flag);
               flag=0;
               break;
   }
   return 0;
}
```

```
static int device_mmap(struct file *file,struct vm_area_struct* vma)
{
   int size;
   size=vma->vm_end - vma->vm_start;
   if(remap_pfn_range(vma,vma->vm_start, PHY_MAP_ADD>>PAGE_SHIFT, size,
          vma->vm_page_prot))
      return -EAGAIN;
   return 0;
}

static struct file_operations acme_fops =
{
   .owner = THIS_MODULE,
   .unlocked_ioctl = device_ioctl,
   .mmap = device_mmap
};


static irqreturn_t *irq_handle(void * dev_id)
{

   printk(KERN_DEBUG "fpga Interrupt\n");
   regs[4]=1;
   flag = 1;
   wake_up(&hq);
   return 0;
}
static int
hello_init (void)
{
   int req_irq = request_irq(6, irq_handle, 0, "myfpga", &dev_id);
   regs = ioremap(PHY_IO_ADD,0x1000);

   acme_dev = MKDEV(237,0);
   register_chrdev_region(acme_dev,1,"myfpga");

   device_create(class_create(THIS_MODULE,"myclass"),NULL,acme_dev,NULL,"myfpga");
   acme_cdev=cdev_alloc();
   if(!acme_cdev)
   {
      printk (KERN_INFO "cdev alloc error.\n");
       return -1;
   }
   acme_cdev->ops = &acme_fops;
   acme_cdev->owner = THIS_MODULE;

   if(cdev_add(acme_cdev,acme_dev,acme_count))
   {
      printk (KERN_INFO "cdev add error.\n");
       return -1;
   }

  return 0;

}

static void
hello_cleanup (void)
{
   cdev_del(acme_cdev);
   unregister_chrdev_region(acme_dev, acme_count);
}

module_init (hello_init);
module_exit (hello_cleanup);
MODULE_LICENSE("GPL");
```

- Configure and build the kernel
- Run emulator –kernel arch/arm/boot/zImage and check interrupts
- Write a simple test application to check the driver
  - Open the file: /dev/myfpga
  - Do some ioctl’s
- Build the image and test your app

Writing a specific hardware library

Base path for all parts: ~/aosp/device/generic/goldfish

- Create a new folder: include
- Add a new file to include directory : libfpga.h

```
#ifndef _FPGA_INTERFACE_H
#define _FPGA_INTERFACE_H

#include <stdint.h>
#include <sys/cdefs.h>
#include <sys/types.h>
#include <hardware/hardware.h>

__BEGIN_DECLS

#define MYFPGA_HARDWARE_MODULE_ID "myfpga"

struct fpga_device_t {
  struct hw_device_t common;

  int (*get_value)(struct fpga_device_t* dev);
  int (*set_start)(struct fpga_device_t* dev,int start);
  int (*set_end)(struct fpga_device_t* dev,int end);
  int (*set_int)(struct fpga_device_t* dev,int intval);
  int (*wait_for_int)(struct fpga_device_t* dev);
  int (*get_int_count)(struct fpga_device_t* dev);
};

__END_DECLS

#endif
```

- Create a new folder: lib
- Add Android.mk file to lib (include $(call all-subdir-makefiles))
- Create a sub-direcory libfpga
- Write the implementation file using the HAL method

```
#include <libfpga.h>
#include <log/log.h>
#include <log/logger.h>
#include <fcntl.h>
#include <poll.h>
#include <errno.h>
#include <sys/ioctl.h>

#define GETVAL          10
#define GETINTCOUNT  50
#define SETSTART                60
#define SETSTOP                 70
#define SETINTVAL               80
#define WAITFORINT     100

static int ioctl_fpga(int request, int param) {
  int logfd = open("/dev/myfpga", O_RDWR);
  if (logfd < 0) {
    return -1;
  } else {
    int ret = ioctl(logfd, request, param);
    close(logfd);
    return ret;
  }
}

static int fpga_get_value(struct fpga_device_t* dev) {
        return ioctl_fpga(GETVAL, 0);
}

static int fpga_set_start(struct fpga_device_t* dev,int start)
{
        return ioctl_fpga(SETSTART, start);
}
static int fpga_set_end(struct fpga_device_t* dev,int end)
{
        return ioctl_fpga(SETSTOP, end);
}
static int fpga_set_intval(struct fpga_device_t* dev,int intval)
{
        return ioctl_fpga(SETINTVAL, intval);
}
```

```
static int fpga_wait_for_int(struct fpga_device_t* dev)
{
        return ioctl_fpga(WAITFORINT, 0);
}
int fpga_get_int_count(struct fpga_device_t* dev)
{
        return ioctl_fpga(GETINTCOUNT, 0);
}
static int close_fpga(struct fpga_device_t* dev) {

  return 0;
}

static int open_fpga(const struct hw_module_t *module, char const *name,
   struct hw_device_t **device) {
    struct fpga_device_t *dev = malloc(sizeof(struct fpga_device_t));
    if (!dev) {
      return -ENOMEM;
    }
    memset(dev, 0, sizeof(*dev));
    dev->common.tag = HARDWARE_DEVICE_TAG;
    dev->common.version = 0;
    dev->common.module = (struct hw_module_t *)module;
    dev->common.close = (int (*)(struct hw_device_t *)) close_fpga;

    dev->get_value = fpga_get_value;
    dev->set_start = fpga_set_start;
    dev->set_end = fpga_set_end;
    dev->set_int = fpga_set_intval;
    dev->wait_for_int = fpga_wait_for_int;
    dev->get_int_count = fpga_get_int_count;
    *device = (struct hw_device_t *)dev;
    return 0;
}

static struct hw_module_methods_t fpga_module_methods = {
  .open = open_fpga,
};

struct hw_module_t HAL_MODULE_INFO_SYM = {
  .tag = HARDWARE_MODULE_TAG,
  .version_major = 1,
  .version_minor = 0,
  .id = MYFPGA_HARDWARE_MODULE_ID,
  .name = "fpga module",
  .author = "Mabel Tech.",
  .methods = &fpga_module_methods,
};
```

- Android.mk

```
LOCAL_PATH:= $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE_TAGS := optional
LOCAL_C_INCLUDES := $(LOCAL_PATH)/../../include/
LOCAL_SRC_FILES := libfpga.c
LOCAL_SHARED_LIBRARIES := libcutils
LOCAL_MODULE := myfpga.default
LOCAL_MODULE_PATH := $(TARGET_OUT_SHARED_LIBRARIES)/hw
LOCAL_CFLAGS += -g -O0
include $(BUILD_SHARED_LIBRARY)
```

Create a simple test application for the lib

```
#include <stdio.h>
#include <string.h>
#include <errno.h>

#include <libfpga.h>
#include <hardware/hardware.h>

int main (int argc, char* argv[]) {
  hw_module_t* module;
  int ret = hw_get_module(MYFPGA_HARDWARE_MODULE_ID, (hw_module_t const**)&module);
  if (ret == 0) {
    struct fpga_device_t *dev;
    module->methods->open(module, 0, (struct hw_device_t **) &dev);
    int val = dev->get_value(dev);
    printf("val=%d\n",val);
    dev->common.close((struct hw_device_t *)dev);
  }
  return ret;
}
```

- Android.mk

```
LOCAL_PATH:= $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE_TAGS := optional
LOCAL_SRC_FILES := fpgatest.c
LOCAL_C_INCLUDES := $(LOCAL_PATH)/../include/
LOCAL_SHARED_LIBRARIES := libhardware
LOCAL_CFLAGS += -g -O0
LOCAL_MODULE := fpgatestapp
include $(BUILD_EXECUTABLE)
```

Writing a system service for our hardware

- Create the following directory structure:

  framework/fpgaservice/com/android/fpgaservice

- Add Android.mk (generic) to framework directory
- Create file IFpgaService.aidl

```
package com.android.fpgaservice;


interface IFpgaService {
  int getval();

// add other methods....
}
```

- Create a FpgaManager class

```
package com.android.fpgaservice;

import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.os.RemoteException;
import android.os.ServiceManager;
import android.util.Log;
import android.util.Slog;
import java.util.HashSet;
import java.util.Set;

public class FpgaManager {
  private static final String REMOTE_SERVICE_NAME = IFpgaService.class.getName();
  private final IFpgaService service;

  public static FpgaManager getInstance() {
    return new FpgaManager();
  }

    public int getval()
    {
          try
          {
                  return service.getval();
          }catch(Exception ec){}
          return 0;
    }

//.... Implement more
  private FpgaManager() {
    this.service = IFpgaService.Stub.asInterface(
      ServiceManager.getService(REMOTE_SERVICE_NAME));
    if (this.service == null) {
      throw new IllegalStateException("Failed to find IFpgaService by name [" +
REMOTE_SERVICE_NAME + "]");
    }
  }

}
```

- Inside fpgaservice directory add the following xml file (com.android.fpgaservice.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<permissions>
  <library name="com.android.fpgaservice"
    file="/system/framework/com.android.fpgaservice.jar"/>
</permissions>
```

- Android.mk

```
LOCAL_PATH := $(call my-dir)

# Build the library
include $(CLEAR_VARS)
LOCAL_MODULE_TAGS := optional
LOCAL_MODULE := com.android.fpgaservice
LOCAL_SRC_FILES := $(call all-java-files-under,.)
LOCAL_SRC_FILES += com/android/fpgaservice/IFpgaService.aidl
include $(BUILD_JAVA_LIBRARY)

# Copy com.android.fpgaservice.xml to /system/etc/permissions/
include $(CLEAR_VARS)
LOCAL_MODULE_TAGS := optional
LOCAL_MODULE := com.android.fpgaservice.xml
LOCAL_MODULE_CLASS := ETC
LOCAL_MODULE_PATH := $(TARGET_OUT_ETC)/permissions
LOCAL_SRC_FILES := $(LOCAL_MODULE)
include $(BUILD_PREBUILT)
```

Create an application to host the service

- Create the following directory structure:
  - app
    - Android.mk (generic)
    - FpgaServiceApp
      - Android.mk (generic)
      - java
        - AndroidManifest.xml
        - Android.mk
        - com
          - android
            - fpgaappservice
              - FpgaServiceApp.java
              - IFpgaServiceImpl.java
      - Jni
        - Android.mk
        - Fpgajni.cpp
  - Create the service implementation (IFpgaServiceImpl.java)

```
package com.android.fpgaappservice;

import android.os.Binder;
import android.os.IBinder;
import android.os.RemoteException;
import com.android.fpgaservice.*;

class IFpgaServiceImpl extends IFpgaService.Stub {
        public static native int getVal();
        // add other native methods

        public int getval() {
                return getVal() ;
        }

        static
        {
                System.loadLibrary("fpga_jni");
        }
}
```

- Create the service application (FpgaServiceApp.java)

```
package com.android.fpgaappservice;

import android.app.Application;
import android.os.ServiceManager;
import android.util.Log;
import com.android.fpgaservice.IFpgaService;

public class FpgaServiceApp extends Application {
  private static final String REMOTE_SERVICE_NAME = IFpgaService.class.getName();
  private IFpgaServiceImpl serviceImpl;

  public void onCreate() {
    super.onCreate();
    this.serviceImpl = new IFpgaServiceImpl();
    ServiceManager.addService(REMOTE_SERVICE_NAME, this.serviceImpl);
  }

  public void onTerminate() {
    super.onTerminate();
  }
}
```

- Create the JNI native code and add all methods

```
#include <jni.h>
#include <libfpga.h>
#include <hardware/hardware.h>
#include "JNIHelp.h"

static const char * class_name = "com/android/fpgaappservice/IFpgaServiceImpl";

static jint getval(JNIEnv *env, jobject object) {
  hw_module_t* module;
  int ret = hw_get_module(MYFPGA_HARDWARE_MODULE_ID, (hw_module_t const**)&module);
  if (ret == 0) {
    struct fpga_device_t *dev;
    module->methods->open(module, 0, (struct hw_device_t **) &dev);
    int val = dev->get_value(dev);
    return val;
  }
return 0;
}
```

```
static JNINativeMethod method_table[] = {
  { "getVal", "()I", (void *) getval},
};


extern "C" jint JNI_OnLoad(JavaVM* vm, void* reserved) {
  JNIEnv* env = NULL;
  if (vm->GetEnv((void**) &env, JNI_VERSION_1_6) == JNI_OK) {
    if (jniRegisterNativeMethods(env, class_name, method_table, NELEM(method_table)) == 0) {
      return JNI_VERSION_1_6;
    }
  }
  return JNI_ERR;
}
```

- Android.mk

```
LOCAL_PATH:= $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE_TAGS := optional
LOCAL_SRC_FILES := fpgajni.cpp
LOCAL_C_INCLUDES += $(JNI_H_INCLUDE) $(LOCAL_PATH)/../../../include/
LOCAL_CFLAGS += -g -O0
LOCAL_SHARED_LIBRARIES := libhardware libnativehelper
LOCAL_MODULE := libfpga_jni
include $(BUILD_SHARED_LIBRARY)
```

- AndroidManifest.xml

```
  <?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.android.fpgaappservice"
  android:sharedUserId="android.uid.system">
  <application android:name=".FpgaServiceApp" android:persistent="true">
    <uses-library android:name="com.android.fpgaservice" />
  </application>
</manifest>
```

- Android.mk for the java code

```
LOCAL_PATH:= $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE_TAGS := optional
LOCAL_SRC_FILES := $(call all-java-files-under,.)
LOCAL_REQUIRED_MODULES := com.android.fpgaservice
LOCAL_JAVA_LIBRARIES := com.android.fpgaservice \
          core \
          framework
LOCAL_PACKAGE_NAME := FpgaServiceApp
LOCAL_SDK_VERSION := current
LOCAL_PROGUARD_ENABLED := disabled
LOCAL_CERTIFICATE := platform
include $(BUILD_PACKAGE)
```

Final steps before build

- Add all the packages to build/target/product/generic/core.mk
- Update the file ueventd.goldfish.rc to set the correct permissions
- Build the image and run the emulator with the modified kernel
- run ps to see if serviceapp is running (with system user)

Build a user application to test the service

- Create a simple activity to test the service
- The library for the client (classes-full-debug.jar) is generated in:
  out/target/common/obj/JAVA_LIBRARIES/com.android.fpgaservice_intermediates/
- Project->properties->java build path->libraries-> add library-> user library -> user libraries -> new -> set name -> add jar
- Add the user library to the AndroidManifest.xml file
  <uses-library android:name=*"com.android.fpgaservice" android:required="true" />*