

Embedded Systems Lab (TE3060)

Laboratory Exercise 3

Examples for HPS SoC - Controlling user LED and Key

Goal: This is an introductory exercise in using a Linux operating system on Intel's Cyclone V SoC devices. The exercise uses the DE10-Nano / DE10-Standard development and education boards. Linux runs on the ARM processor that is part of the Cyclone V SoC device. The lab shows how to control the users LED and KEY by accessing the register of the GPIO controller through a memory-mapped device driver. Functionally, when the user presses the KEY button, the user LED should turn on.

1. Introduction

Figure 1 shows a block diagram of the system that will be implemented. The users LED and KEY are connected to the GPIO1 controller in the HPS. The behavior of the GPIO controller is controlled by configuration registers in the GPIO controller. The registers can be accessed by application software through the memory-mapped device driver of the LED and KEY. This memory-mapped device driver is built into SoC Linux.

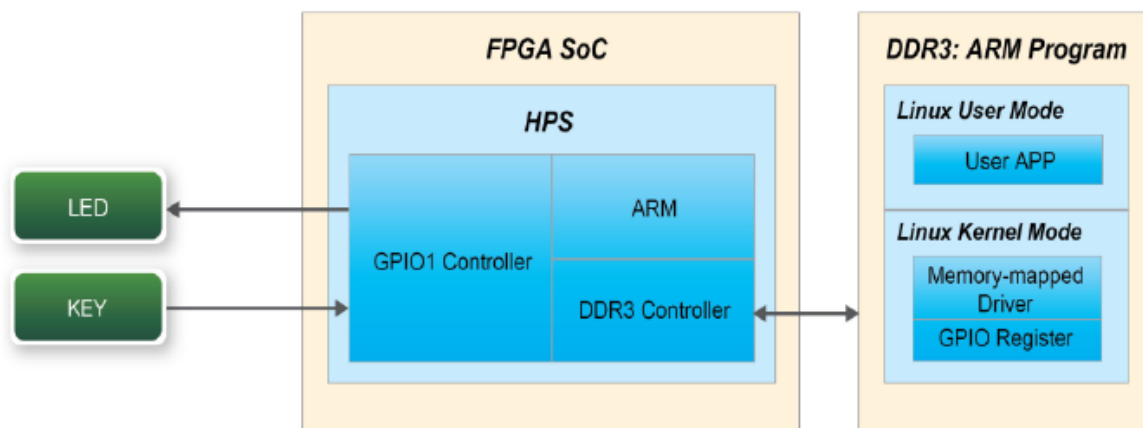


Figure 1. System block diagram.

The HPS provides three general-purpose I/O (GPIO) modules that are used to interface the ARM processor with any hardware that is connected to this interface. Figure 2 shows the block diagram of this GPIO interface. GPIO[28..0] is controlled by GPIO0 controller, GPIO[57..29] is controlled by GPIO1 controller and GPIO[70..58] and input-only GPI[13..0] are controlled by GPIO2 controller.

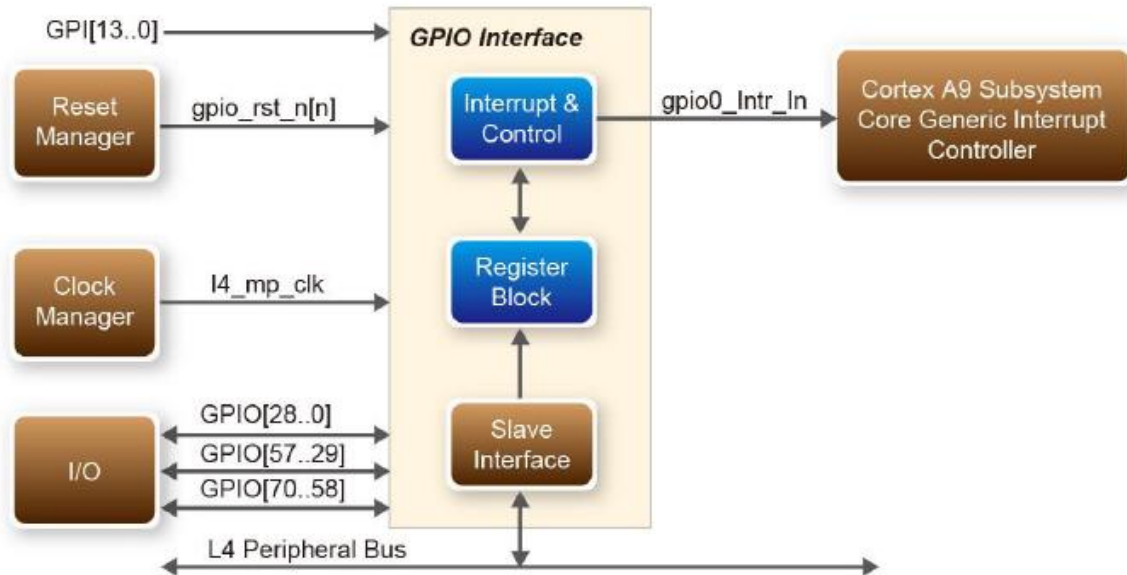


Figure 2. GPIO interface.

The register block in the GPIO interface controls the behavior of the I/O pins. There are three 32-bit registers in the GPIO controller used in this lab exercise. The registers are:

- **gpio_swporta_dr**: write output data to output I/O pin (**data register**)
- **gpio_swporta_dds**: configures the direction of the I/O pin (**data direction register**)
- **gpio_ext_porta**: read input data of I/O input pin

For the LED pin, the **gpio_swporta_dds** register must be configured first so that the pin corresponding to the LED is configured as output. Then, to turn on/off the LED, register **gpio_swporta_dr** is written the corresponding data. The LSB (Least significant bit) of **gpio_swporta_dds** controls the direction of first I/O pin in the associated GPIO controller, the second bit controls the direction of second I/O pin in the associated GPIO controller and so on. A value "1" in the register bit indicates the I/O direction is output, while the value "0" in the register bit indicates the I/O direction is input.

The first bit of **gpio_swporta_dr** register controls the output value of first I/O pin in the associated GPIO controller, the second bit controls the output value of second I/O pin in the associated GPIO controller and so on. The value "1" in the register bit indicates the output value is high, and the value "0" indicates the output value is low.

The status of KEY can be queried by reading the value of **gpio_ext_porta** register. The first bit represents the input status of first I/O pin in the associated GPIO controller, the second bit represents the input status of second I/O pin in the associated GPIO controller and so on. The value "1" in the register bit indicates the input state is high, and the value "0" indicates the input state is low.

HPS peripherals are associated to registers, which are mapped to HPS base address space 0xFC000000 with 64KB size. The GPIO controller are mapped to the following base addresses:

- The registers of GPIO0 controller are mapped to the base address 0xFF708000 with 4KB size
- The registers of GPIO1 controller are mapped to the base address 0xFF709000 with 4KB size
- The registers of the GPIO2 controller are mapped to base address 0xFF70A000 with 4KB size.

2. Software Development

To develop the application, we will use the following APIs to access the register of GPIO controller:

- open: open memory mapped device driver
- mmap: map physical memory to user space
- alt_read_word: read a value from a specified register
- alt_write_word: write a value into a specified register
- munmap: clean up memory mapping
- close: close device driver

In addition, we will use the following MACROS to access the register:

- alt_setbits_word: set specified bit value to one for a specified register
- alt_clrbits_word: set specified bit value to zero for a specified register

In order to use the above APIs and MACROS, the program must include the following header files:

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>
#include "hwlib.h"
#include "socal/socal.h"
#include "socal/alt_gpio.h"
```

Figure 3 and Figure 4 show the LED and KEY pins for the DE10-Nano and DE10-Standard boards, respectively. In both cases, the LED is connected to GPIO53 and KEY to GPIO54. As such, they are controlled by GPIO1 controller. Both tables also show what bits in GPIO1 controller should be used to control the LED and KEY. For more information on this and other peripherals connected to the FPGA or HPS of the cyclone V SoC, refer to the **“DE10-Nano User Manual”** and the **“DE10-Standard User Manual”**.

Table 3-14 gives the pin assignment of all the LEDs, switches, and push-buttons.

Table 3-14 Pin Assignment of LEDs, Switches and Push-buttons

Signal Nam	FPGA Pin No	HPS GPIO	Register/bit	Function
HPS_KEY	PIN_J18	GPIO54	GPIO1[25]	I/O
HPS_LED	PIN_A20	GPIO53	GPIO1[24]	I/O

Figure 3. DE10-Nano HPS LED and KEY pin assignment and GPIO register bit

Table 3-26 gives the pin assignment of all the LEDs, switches, and push-buttons.

Table 3-26 Pin Assignment of LEDs, Switches and Push-buttons

Signal Name	HPS GPIO	Register/bit	Function
HPS_KEY	GPIO54	GPIO1[25]	I/O
HPS_LED	GPIO53	GPIO1[24]	I/O

Figure 4. DE-10 Standard HPD LED and KEY pin assignment, GPIO controller and GPIO register bit

Figure 5 shows the **gpio_swporta_dds** register of the GPIO1 controller. Bit-0 controls the pin direction of HPS_GPIO29. Bit-24 controls the pin direction of HPS_GPIO53, which connects to HPS_LED. Bit-25 controls the pin direction of HPS_GPIO54, which connects to HPS_KEY, and so on. The pin direction of HPS_LED and HPS_KEY are controlled by bit-24 and bit-25 in the **gpio_swporta_dds** register of the GPIO1 controller, respectively. Similarly, the output status of HPS_LED is controlled by the bit-24 in the **gpio_swporta_dr** register of the GPIO1 controller. The status of KEY can be queried by reading the value of the bit-24 in the **gpio_ext_porta** register of the GPIO1 controller.

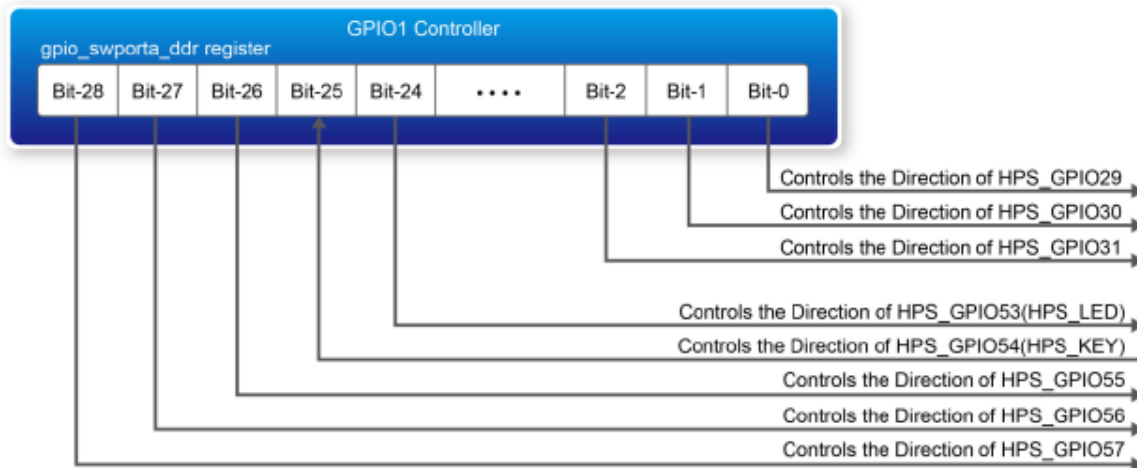


Figure 5. gpio_swporta_dds register for GPIO1 controller

3. Procedure

For this lab exercise, the program has been provided and we will walk through it to understand what is going on:

1. Open the program "main.c" provided with this document
2. Check the header files that have been included and confirm they are complete, i.e., they are the same as indicated in page 4
3. A set of #defines are added

`#define HW_REGS_BASE (ALT_STM_OFST)` **For the Cyclone V SoC, ALT_STM_OFFSET is 0xFC000000 by default, which is the base address of the peripherals that are controlled by the HPS**

`#define HW_REGS_SPAN (0x04000000)` **This define indicates the span of the memory space that will be used. This means that all peripherals that sit within the span address can be accessed in the same way as the LED and KEY addresses that will be shown below.**

`#define HW_REGS_MASK (HW_REGS_SPAN - 1)`

The following define statements are masks that will be used to control the LED and KEY direction (ddr register) and LED's output value

`#define USER_IO_DIR (0x01000000)` **This mask will set bit 25 of the gpio_swporta_ddr register of the GPIO1 controller to zero, meaning that GPIO54 (HPS KEY) is set as input**

`#define BIT_LED (0x01000000)` **This mask will set bit 24 of the gpio_swporta_ddr register of the GPIO1 controller to one, meaning that GPIO53 (HPS LED) is set as output**

`#define BUTTON_MASK(0x02000000)` **This mask will be used to check bit 25 of the gpio_ext_porta register. This bit is set/cleared depending on the input of the KEY button.**

4. In the main function
 - 4.1. After the definition of pointers and other variables, the first instruction is used to map the address space of Linux into user space so that the programmer can interact with the address space devoted to the peripherals that are controlled by the HPS.
 - 4.2. It very important to point out that the software that is ran on Linux (HPS side) does not have access to the physical memory of any peripherals running on the HPS. Hence, in order to work around this restriction, the physical address is mapped in the user space to give the programmer access to the peripherals connected.
 - 4.3. The following instruction is used to create the virtual base address of the peripherals that can be controlled by the HPS. This virtual base address will be used every time we want to access a peripheral. Keep in mind that this is the base address of the memory space related to peripherals. To access a specific peripheral, we must add the required offset to this base address.

```
virtual_base = mmap(NULL, HW_REGS_SPAN, (PROT_READ | PROT_WRITE),
MAP_SHARED, fd, HW_REGS_BASE);
```

```
if( virtual_base == MAP_FAILED ) {
    printf( "ERROR: mmap() failed...\n" );
    close( fd );
    return( 1 );
}
```

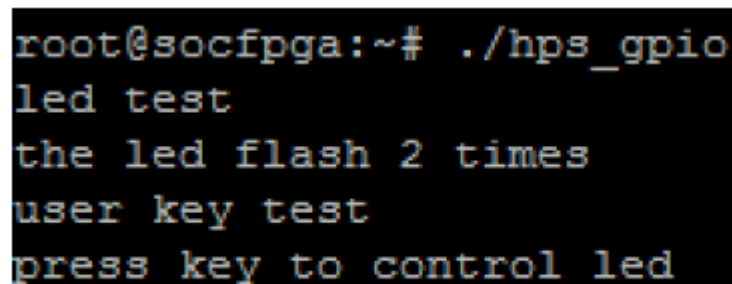
}

4.4. You are assigned the task to understand the rest of the instructions. Hint: take a look at the contents of the header files.

4. Running the Lab

Follow the next steps:

1. Compile the program with the Makefile that is included in this lab and create the executable **hps_gpio**
2. Copy the executable file **hps_gpio** into the microSD card under **/home/root** folder in Linux. Use the ethernet-based procedure we did in lab 2.
3. Connect a USB cable to the USB-to-UART connector on the DE10 Standard/DE10 Nano board and the host PC
4. Launch **PuTTY** and establish connection to the UART port
5. Type **ls** and you should be able to see the executable file **hps_gpio**
6. Type **./hps_gpio** in the UART terminal to start the program. You should see the following in the PuTTY terminal



```
root@socfpga:~# ./hps_gpio
led test
the led flash 2 times
user key test
press key to control led
```

7. HPS_LED will flash twice and then the user can control the LED by pushing the KEY button
8. Press the HPS_KEY button to turn the HPS_LED on
9. If you want to finish the demo, press **"CTRL+C"** to terminate the demo.