Due Date: Sunday November 17, 11:59pm– Submit this editable **Word** document NOT pdf

0 / 12

EEDG/CE 6370
Design and Analysis or Reconfigurable Systems
Homework 8 – Hard Processor System

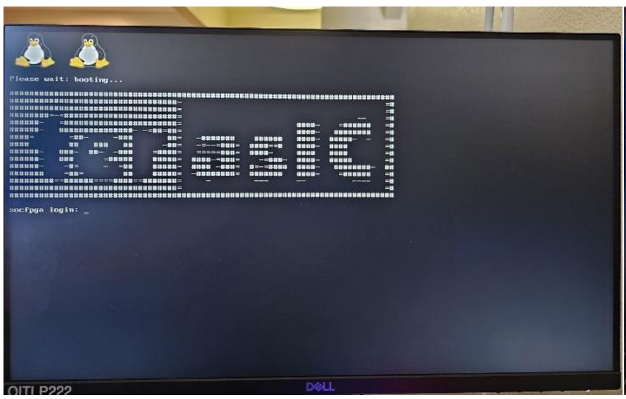
Name: Prathamesh Sanjay Gadad

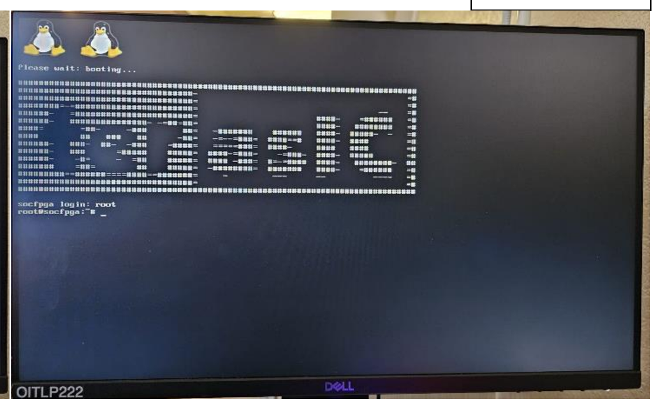
Email: psg220003@utdallas.edu

Part I - Linux

Install any of the Linux versions on the SD card and show that it is working. Include picture.

Marks
2



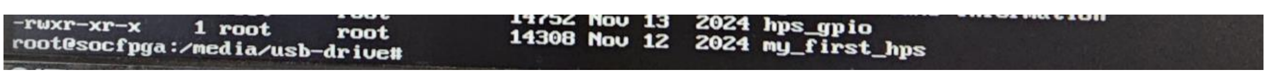


Part II – Simple program

Write a simple "Hello UTD" Program" and compile and run it on the HPS of the FPGA. Report the size of the compiled file. Include a video of the compilation process and working design showing the program runs on the FPGA.

Marks
4

Size of the Compiled file: 14.4KB



https://youtu.be/kloU9zJi1kc

Part III – Controlling Peripherals through HPS

a.) As you can see from the picture below, the HPS in the Terasic board is ‘connected’ to different peripherals:

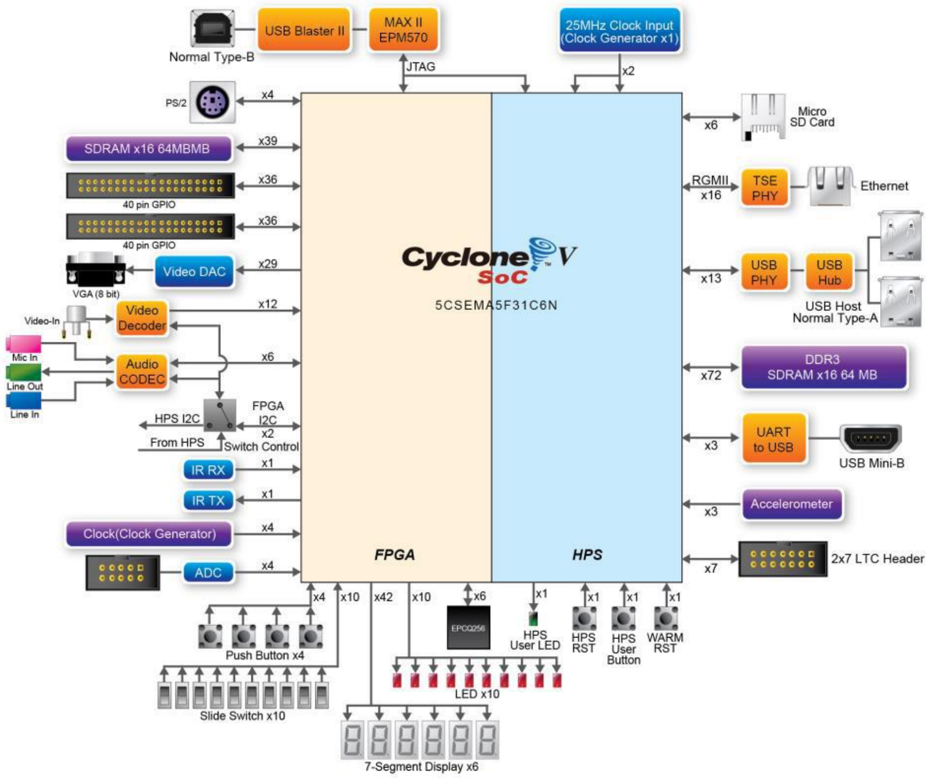


Figure 1 Cyclone V SoC in Terasic DE1-SoC board overview

Some of these peripherals include an HPS user button and a HPS user LED. Compile the hps_gpio/main.c program and annotate next to each line what it does through C/C++ comments. Include the size of the compiled program too. Create a YourTube video showing that it works.

Marks
6

Size of the Compiled file: 14.8KB

https://youtu.be/tn8Whvu2K4M

Annotated version of the code and explanations:

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

#define HW_REGS_BASE ( ALT_STM_OFST )  // Base address for the hardware
registers
#define HW_REGS_SPAN ( 0x04000000 )    // Memory span for hardware registers
#define HW_REGS_MASK ( HW_REGS_SPAN - 1 )  // Mask to access the relevant part
of memory

#define USER_IO_DIR     (0x01000000)    // Direction of I/O for the LED
#define BIT_LED         (0x01000000)    // Bit mask for controlling the LED
#define BUTTON_MASK     (0x02000000)    // Bit mask for the button press

int main(int argc, char **argv) {
    void *virtual_base;
    int fd;
    uint32_t scan_input;
    int i;

    // Open "/dev/mem" to access the physical memory of the system
    // This allows us to directly interact with hardware registers
    if( ( fd = open( "/dev/mem", ( O_RDWR | O_SYNC ) ) ) == -1 ) {
        printf( "ERROR: could not open \"/dev/mem\"...\n" );
        return( 1 );
    }

    // Map the hardware register space into user space so that we can interact
with it
    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 );
    }

    // Initialize the PIO controller
    // Set the direction of the HPS GPIO1 bits attached to the LEDs to output
    alt_setbits_word( ( virtual_base +
( ( uint32_t )( ALT_GPIO1_SWPORTA_DDR_ADDR ) &
( uint32_t )( HW_REGS_MASK ) ) ), USER_IO_DIR );
```

```
    printf("led test\r\n");
    printf("the led flash 2 times\r\n");

    // Flash the LED twice
    for(i=0;i<2;i++) {
        // Set the LED on
        alt_setbits_word( ( virtual_base +
( ( uint32_t )( ALT_GPIO1_SWPORTA_DR_ADDR ) &
( uint32_t )( HW_REGS_MASK ) ) ), BIT_LED );
        usleep(500*1000); // Wait for 500ms
        // Set the LED off
        alt_clrbits_word( ( virtual_base +
( ( uint32_t )( ALT_GPIO1_SWPORTA_DR_ADDR ) &
( uint32_t )( HW_REGS_MASK ) ) ), BIT_LED );
        usleep(500*1000); // Wait for 500ms
    }

    // Test the user button and control the LED with it
    printf("user key test \r\n");
    printf("press key to control led\r\n");

    // Infinite loop to monitor the button press
    while(1) {
        // Read the state of the GPIO input (button)
        scan_input = alt_read_word( ( virtual_base +
( ( uint32_t )(  ALT_GPIO1_EXT_PORTA_ADDR ) &
( uint32_t )( HW_REGS_MASK ) ) ) );

        // Check if the button is pressed (active low)
        if(~scan_input & BUTTON_MASK)
            // If pressed, turn the LED on
            alt_setbits_word( ( virtual_base +
( ( uint32_t )( ALT_GPIO1_SWPORTA_DR_ADDR ) &
( uint32_t )( HW_REGS_MASK ) ) ), BIT_LED );
        else
            // If not pressed, turn the LED off
            alt_clrbits_word( ( virtual_base +
( ( uint32_t )( ALT_GPIO1_SWPORTA_DR_ADDR ) &
( uint32_t )( HW_REGS_MASK ) ) ), BIT_LED );
    }

    // Clean up the memory mapping and close the file descriptor
    if( munmap( virtual_base, HW_REGS_SPAN ) != 0 ) {
        printf( "ERROR: munmap() failed...\n" );
        close( fd );
        return( 1 );
    }

    close( fd );
    return( 0 );
}
```

- To read and write to the peripherals directly, the program opens the file /dev/mem to gain access to physical memory.

- Next the mmap function maps the hardware registers to user space, allowing the access to memory where GPIO control registers are present.
- Then the program sets the direction of GPIO pins connected to the LEDs as output, by writing the memory location.
- To flash the LED, led pin need to be set and cleared with a delay of 500ms between each action.
- The program checks continuously for state of the button connected to the GPIO input. If the button is pressed, it turns on the LED, otherwise it turns off.
- After completion it unmaps the memory and clean the resources.
- The program basically blinks the LED twice and then when we push HPS User Button then the HPS User LEB Blinks.