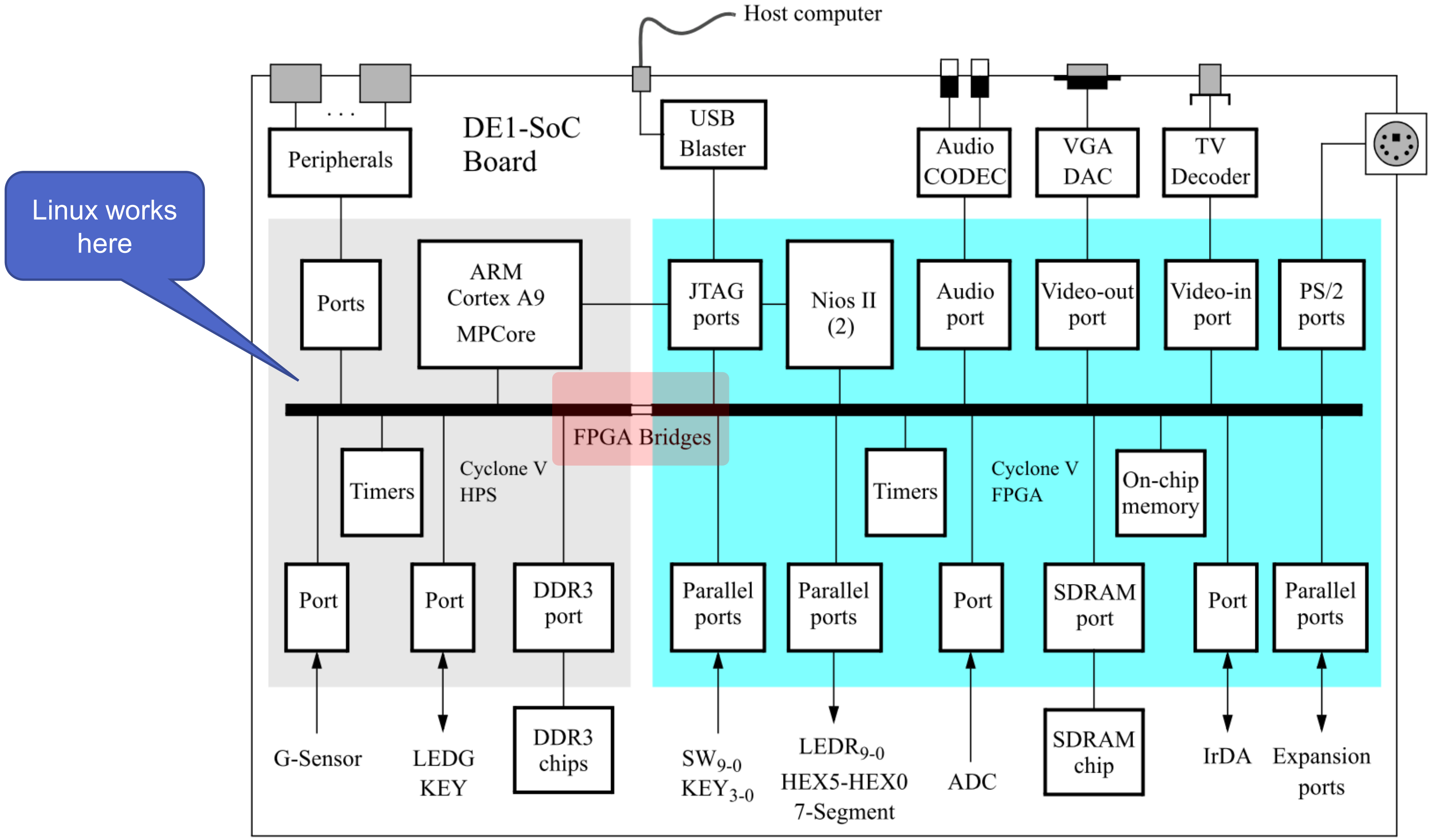
MEMORY-MAPPED IO IN LINUX-BASED SYSTEMS

EMBEDDED SYSTEMS

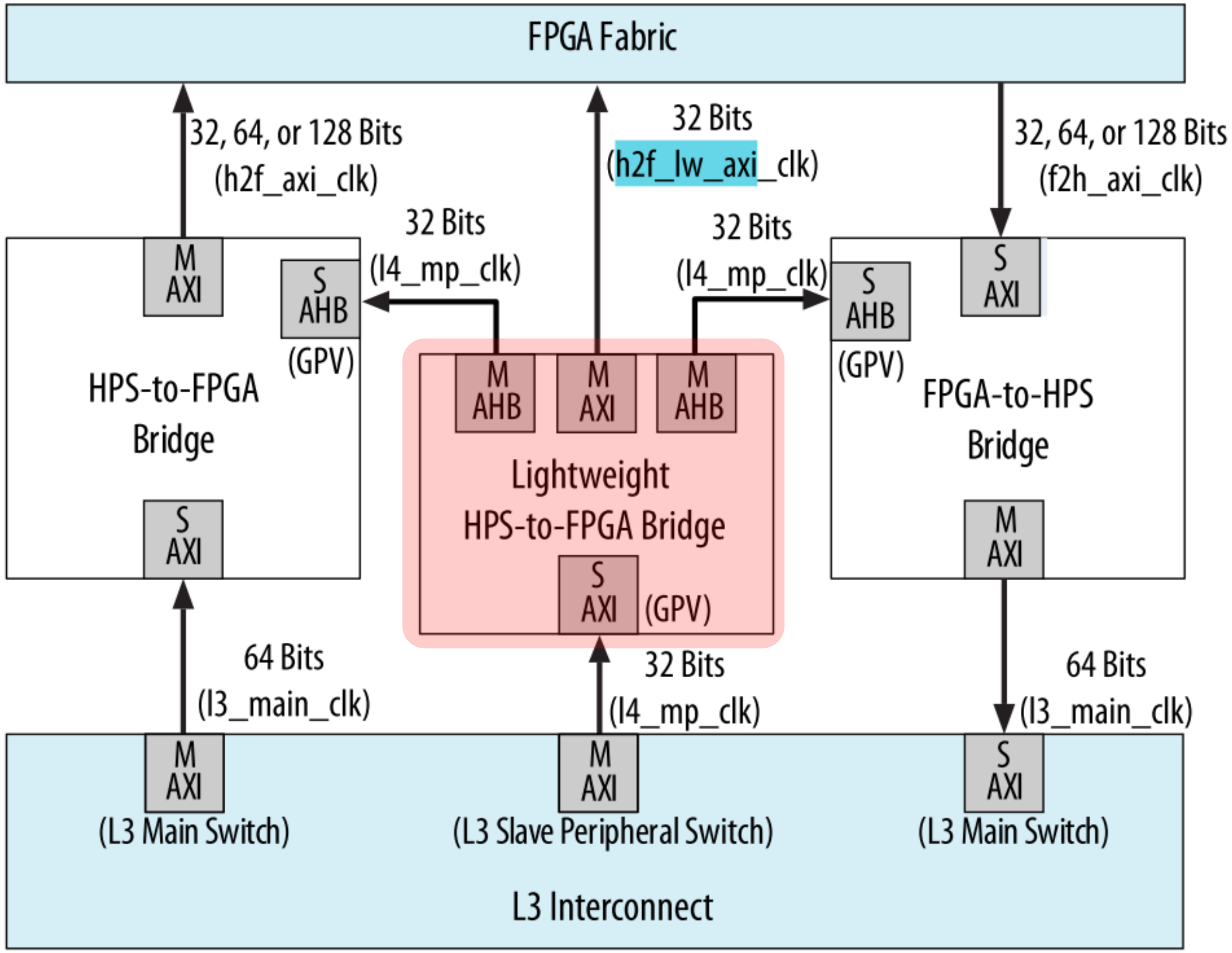
Lab Overview

- The goal of this lab is to implement *Linux* programs to control the devices on the basis of the *memory-mapped IO*.
  - In the memory-mapped IO, communications between the CPU and peripheral devices are performed in the same way as for the memory access, using load/store instructions.
    - Review by yourself the memory-mapped IO in the lecture “CPUs”.

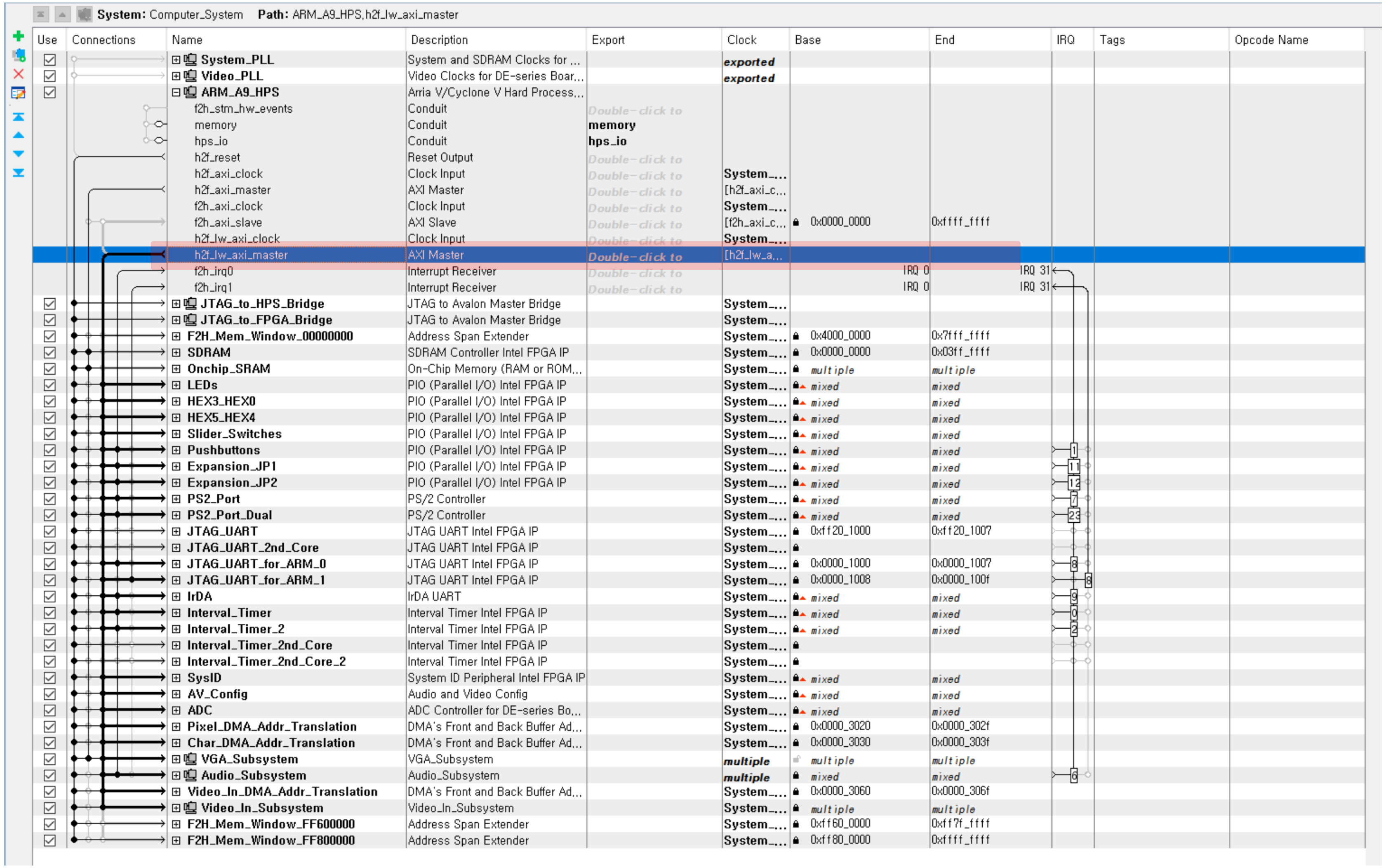
Overall Architecture of Our Platform



Overall Architecture of Our Platform, Cont’d



Overall Architecture of Our Platform, Cont'd



System: Computer_System **Path:** ARM_A9_HPS.h2f_lw_axi_master

Use	Connections	Name	Description	Export	Clock	Base	End	IRQ	Tags	Opcode Name
		System_PLL	System and SDRAM Clocks for ...		exported					
		Video_PLL	Video Clocks for DE-series Boar...		exported					
		ARM_A9_HPS	Arria V/Cyclone V Hard Process...							
		f2h_stm_hw_events	Conduit	Double-click to						
		memory	Conduit	memory						
		hps_io	Conduit	hps_io						
		h2f_reset	Reset Output	Double-click to						
		h2f_axi_clock	Clock Input	Double-click to	System_...					
		h2f_axi_master	AXI Master	Double-click to	[h2f_axi_c...					
		f2h_axi_clock	Clock Input	Double-click to	System_...					
		f2h_axi_slave	AXI Slave	Double-click to	[f2h_axi_c...	0x0000_0000	0xffff_ffff			
		h2f_lw_axi_clock	Clock Input	Double-click to	System_...					
		h2f_lw_axi_master	AXI Master	Double-click to	[h2f_lw_a...					
		f2h_irq0	Interrupt Receiver	Double-click to		IRQ 0	IRQ 31			
		f2h_irq1	Interrupt Receiver	Double-click to		IRQ 0	IRQ 31			
		JTAG_to_HPS_Bridge	JTAG to Avalon Master Bridge		System_...					
		JTAG_to_FPGA_Bridge	JTAG to Avalon Master Bridge		System_...					
		F2H_Mem_Window_00000000	Address Span Extender		System_...	0x4000_0000	0x7fff_ffff			
		SDRAM	SDRAM Controller Intel FPGA IP		System_...	0x0000_0000	0x03ff_ffff			
		Onchip_SRAM	On-Chip Memory (RAM or ROM...		System_...	multiple	multiple			
		LEDs	PIO (Parallel I/O) Intel FPGA IP		System_...	mixed	mixed			
		HEX3_HEX0	PIO (Parallel I/O) Intel FPGA IP		System_...	mixed	mixed			
		HEX5_HEX4	PIO (Parallel I/O) Intel FPGA IP		System_...	mixed	mixed			
		Slider_Switches	PIO (Parallel I/O) Intel FPGA IP		System_...	mixed	mixed			
		Pushbuttons	PIO (Parallel I/O) Intel FPGA IP		System_...	mixed	mixed	1		
		Expansion_JP1	PIO (Parallel I/O) Intel FPGA IP		System_...	mixed	mixed	11		
		Expansion_JP2	PIO (Parallel I/O) Intel FPGA IP		System_...	mixed	mixed	12		
		PS2_Port	PS/2 Controller		System_...	mixed	mixed	7		
		PS2_Port_Dual	PS/2 Controller		System_...	mixed	mixed	23		
		JTAG_UART	JTAG UART Intel FPGA IP		System_...	0xff20_1000	0xff20_1007			
		JTAG_UART_2nd_Core	JTAG UART Intel FPGA IP		System_...					
		JTAG_UART_for_ARM_0	JTAG UART Intel FPGA IP		System_...	0x0000_1000	0x0000_1007	8		
		JTAG_UART_for_ARM_1	JTAG UART Intel FPGA IP		System_...	0x0000_1008	0x0000_100f	8		
		IrDA	IrDA UART		System_...	mixed	mixed	9		
		Interval_Timer	Interval Timer Intel FPGA IP		System_...	mixed	mixed	0		
		Interval_Timer_2	Interval Timer Intel FPGA IP		System_...	mixed	mixed	2		
		Interval_Timer_2nd_Core	Interval Timer Intel FPGA IP		System_...					
		Interval_Timer_2nd_Core_2	Interval Timer Intel FPGA IP		System_...					
		SysID	System ID Peripheral Intel FPGA IP		System_...	mixed	mixed			
		AV_Config	Audio and Video Config		System_...	mixed	mixed			
		ADC	ADC Controller for DE-series Bo...		System_...	mixed	mixed			
		Pixel_DMA_Addr_Translation	DMA's Front and Back Buffer Ad...		System_...	0x0000_3020	0x0000_302f			
		Char_DMA_Addr_Translation	DMA's Front and Back Buffer Ad...		System_...	0x0000_3030	0x0000_303f			
		VGA_Subsystem	VGA_Subsystem		multiple	multiple	multiple			
		Audio_Subsystem	Audio_Subsystem		multiple	mixed	mixed	6		
		Video_In_DMA_Addr_Translation	DMA's Front and Back Buffer Ad...		System_...	0x0000_3060	0x0000_306f			
		Video_In_Subsystem	Video_In_Subsystem		System_...	multiple	multiple			
		F2H_Mem_Window_FF600000	Address Span Extender		System_...	0xff60_0000	0xff7f_ffff			
		F2H_Mem_Window_FF800000	Address Span Extender		System_...	0xff80_0000	0xffff_ffff			

Memory Map of Our Platform

Region Name	Base Address	Size
FPGA slaves	0xC0000000	960 MB
Peripheral	0xFC000000	64 MB
Lightweight FPGA slaves	0xFF200000	2 MB

	ARM_A9_HPS.h2f_axi_master	ARM_A9_HPS.h2f_lw_axi_master
ADC.adc_slave		0x0000_4000 - 0x0000_401f
ARM_A9_HPS.f2h_axi_slave		
AV_Config.avalon_av_config_slave		0x0000_3000 - 0x0000_300f
Audio_Subsystem.audio_slave		0x0000_3040 - 0x0000_304f
Char_DMA_Addr_Translation.slave		0x0000_3030 - 0x0000_303f
Expansion_JP1.s1		0x0000_0060 - 0x0000_006f
Expansion_JP2.s1		0x0000_0070 - 0x0000_007f
F2H_Mem_Window_00000000.win...		
F2H_Mem_Window_FF600000.win...		
F2H_Mem_Window_FF800000.win...		
HEX3_HEX0.s1		0x0000_0020 - 0x0000_002f
HEX5_HEX4.s1		0x0000_0030 - 0x0000_003f
Interval_Timer.s1		0x0000_2000 - 0x0000_201f
Interval_Timer_2.s1		0x0000_2020 - 0x0000_203f
Interval_Timer_2nd_Core.s1		
Interval_Timer_2nd_Core_2.s1		
IrDA.avalon_irda_slave		0x0000_1020 - 0x0000_1027
JTAG_UART.avalon_jtag_slave		
JTAG_UART_2nd_Core.avalon_jt...		
JTAG_UART_for_ARM_0.avalon_jt...		0x0000_1000 - 0x0000_1007
JTAG_UART_for_ARM_1.avalon_jt...		0x0000_1008 - 0x0000_100f
LEDs.s1		0x0000_0000 - 0x0000_000f
Onchip_SRAM.s1	0x0800_0000 - 0x0803_ffff	
Onchip_SRAM.s2		
PS2_Port.avalon_ps2_slave		0x0000_0100 - 0x0000_0107
PS2_Port_Dual.avalon_ps2_slave		0x0000_0108 - 0x0000_010f
Pixel_DMA_Addr_Translation.slave		0x0000_3020 - 0x0000_302f
Pushbuttons.s1		0x0000_0050 - 0x0000_005f
SDRAM.s1	0x0000_0000 - 0x03ff_ffff	
Slider_Switches.s1		0x0000_0040 - 0x0000_004f
SysID.control_slave		0x0000_2040 - 0x0000_2047
VGA_Subsystem.char_buffer_co...		
VGA_Subsystem.char_buffer_slave	0x0900_0000 - 0x0900_1fff	
VGA_Subsystem.pixel_dma_cont...		
VGA_Subsystem.rgb_slave		0x0000_3010 - 0x0000_3013
Video_In_DMA_Addr_Translation....		0x0000_3060 - 0x0000_306f
Video_In_Subsystem.video_in_d...		
Video_In_Subsystem.video_in_e...		0x0000_3070 - 0x0000_307f
ARM_A9_HPS.f2h_axi_slave via F...		
ARM_A9_HPS.f2h_axi_slave via F...		
ARM_A9_HPS.f2h_axi_slave via F...		

Example Program Based on MMIO

```c
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>
#include "address_map_arm.h"

int main(void){
    int fd;
    void *lw_virtual;
    volatile int *ledr;
    int cnt = 0;

    fd = open("/dev/mem", (O_RDWR | O_SYNC));
    lw_virtual = mmap (NULL, LW_BRIDGE_SPAN, (PROT_READ | PROT_WRITE), MAP_SHAR
ED, fd, LW_BRIDGE_BASE);
    ledr = (volatile int *) (lw_virtual + LEDR_BASE);

    *ledr = 0;
    while(cnt<8){
        *ledr = *ledr + 1;
        cnt++;
        usleep(1000000); //1000ms
    }

    munmap(lw_virtual, LW_BRIDGE_BASE);
    close(fd);

    return 0;
}
```

- In Linux, a file can correspond to a general file, a directory, and a device.
- A file descriptor can be considered to a handle to access a file.
- Open the device, /dev/mem, so as to make its file descriptor, in a synchronous read/write mode.

Example Program Based on MMIO, Cont'd

```c
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>
#include "address_map_arm.h"

int main(void){
    int fd;
    void *lw_virtual;
    volatile int *ledr;
    int cnt = 0;

    fd = open("/dev/mem", (O_RDWR | O_SYNC));
    lw_virtual = mmap (NULL, LW_BRIDGE_SPAN, (PROT_READ | PROT_WRITE), MAP_SHAR
ED, fd, LW_BRIDGE_BASE);
    ledr = (volatile int *) (lw_virtual + LEDR_BASE);

    *ledr = 0;
    while(cnt<8){
        *ledr = *ledr + 1;
        cnt++;
        usleep(1000000); //1000ms
    }

    munmap(lw_virtual, LW_BRIDGE_BASE);
    close(fd);

    return 0;
}
```

- Linux in our platform is based on the *virtual memory* system.
- To access a device in a physical memory space, a *mapping* is made *from a virtual address to its physical memory space*.
- Note that the type of ledr is a *volatile* pointer.

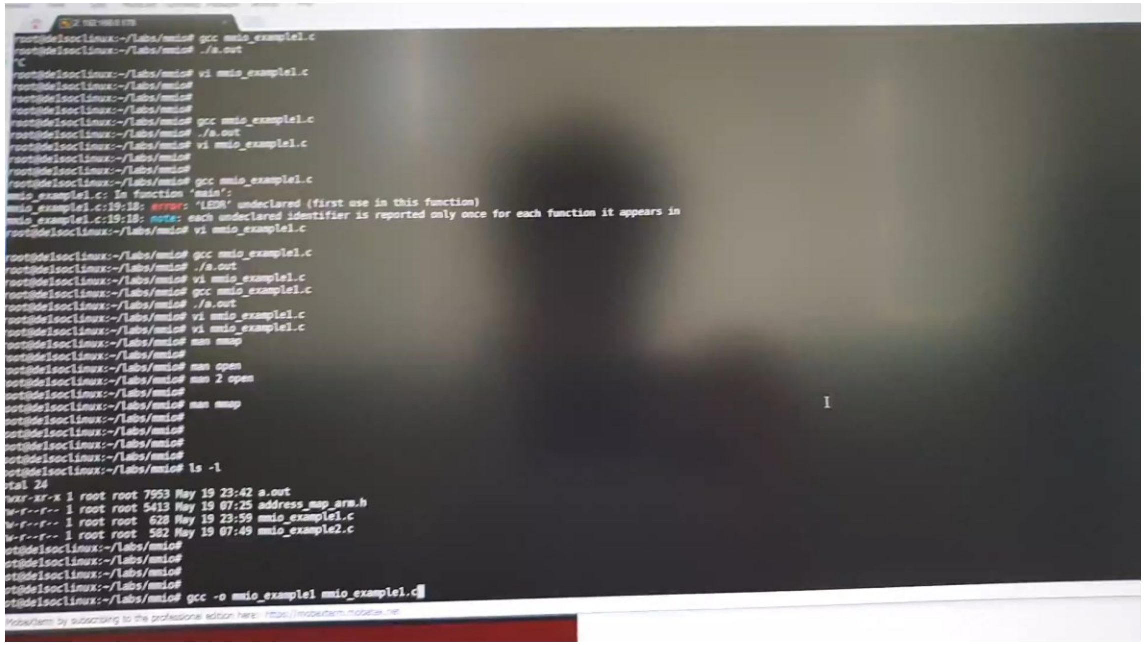
Example Program Based on MMIO, Cont'd

```c
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>
#include "address_map_arm.h"

int main(void){
    int fd;
    void *lw_virtual;
    volatile int *ledr;
    int cnt = 0;

    fd = open("/dev/mem", (O_RDWR | O_SYNC));
    lw_virtual = mmap (NULL, LW_BRIDGE_SPAN, (PROT_READ | PROT_WRITE), MAP_SHAR
ED, fd, LW_BRIDGE_BASE);
    ledr = (volatile int *) (lw_virtual + LEDR_BASE);

    *ledr = 0;
    while(cnt<8){
        *ledr = *ledr + 1;
        cnt++;
        usleep(1000000); //1000ms
    }

    munmap(lw_virtual, LW_BRIDGE_BASE);
    close(fd);

    return 0;
}
```

- lw_virtual is the base pointer to indicate the start address of the LW_BRIDGE.
- The pointer of LEDR is made by adding an offset to it.

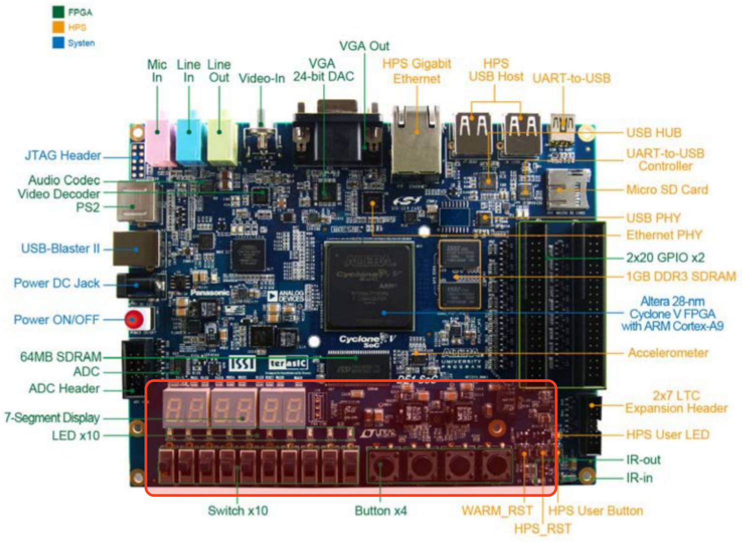
Example Program Based on MMIO, Cont’d

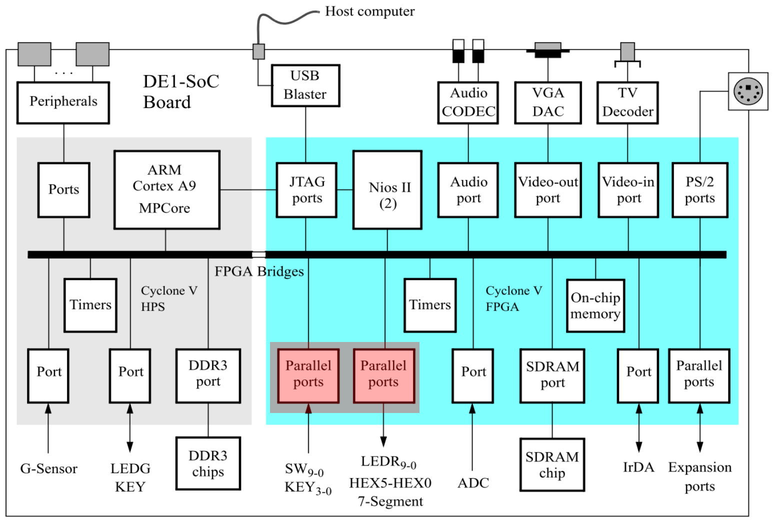
```c
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>
#include "address_map_arm.h"

int main(void){
    int fd;
    void *lw_virtual;
    volatile int *ledr;
    int cnt = 0;

    fd = open("/dev/mem", (O_RDWR | O_SYNC));
    lw_virtual = mmap (NULL, LW_BRIDGE_SPAN, (PROT_READ | PROT_WRITE), MAP_SHAR
ED, fd, LW_BRIDGE_BASE);
    ledr = (volatile int *) (lw_virtual + LEDR_BASE);

    *ledr = 0;
    while(cnt<8){
        *ledr = *ledr + 1;
        cnt++;
        usleep(1000000); //1000ms
    }

    munmap(lw_virtual, LW_BRIDGE_BASE);
    close(fd);

    return 0;
}
```



Parallel Port in FPGA

- LEDRs are in fact controlled by the parallel port.
- The parallel port is used to control push-button KEYs, switches, and 7-segment HEXes, as well as LEDRs.
    - Device types (controlled by the parallel port)
        - Readable, Writable, Readable & Writable





Parallel Port in FPGA, Cont'd

Table 2. Parallel Port register map				
Offset in bytes	**Register name**		**Read/Write**	**Bits** $(n-1)\ldots 0$
0	`data`	Input	R	Data value currently on Parallel Port inputs.
		Output	W	New value to drive on Parallel Port outputs.
4	`direction`		R/W	Individual direction control for each I/O port. A value of 0 sets the direction to input; 1 sets the direction to output.
8	`interruptmask`		R/W	IRQ enable/disable for each input port. Setting a bit to 1 enables interrupts for the corresponding port.
12	`edgecapture`		R/W	Edge detection for each input port.

Notes on Table 2:

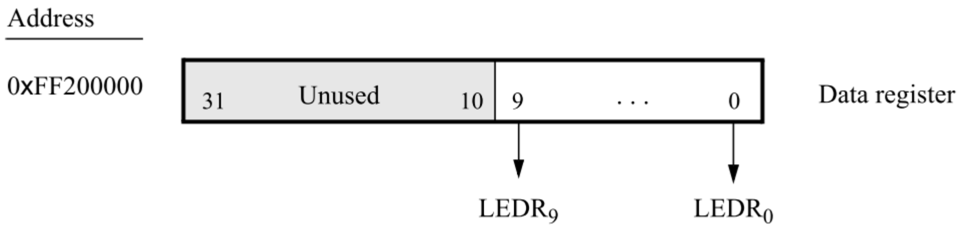
(1) This register may not exist, depending on the hardware configuration. If a register is not present, reading the register returns an undefined value, and writing the register has no effect.

(2) Writing any value to edgecapture clears all bits to 0.

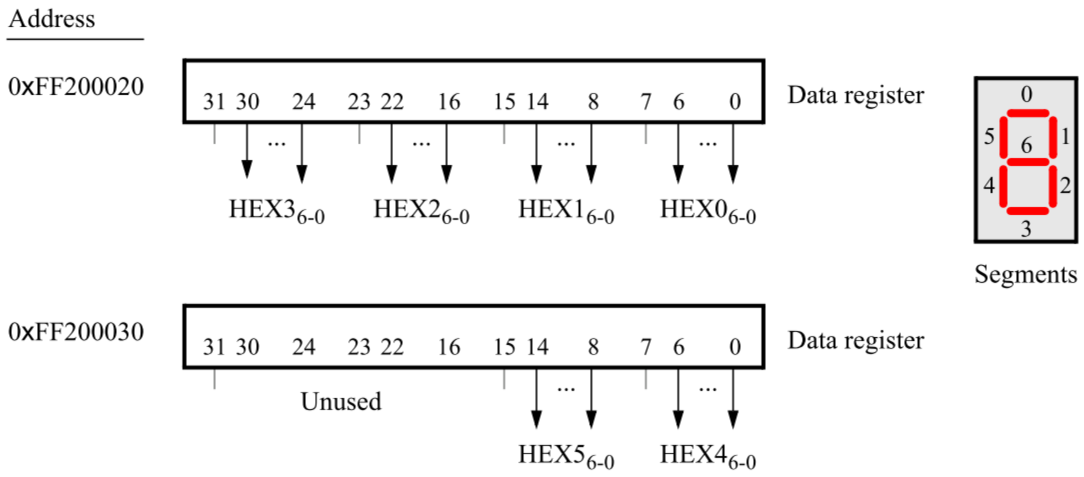
When the parallel port is configured to detect edges, the *edgecapture* register is created to indicate on which bit(s) of the port an edge has occurred. If bit n in the *edgecapture* register is set to 1 whenever an edge is detected on input port n.

Parallel Port in FPGA, Cont'd

- LEDR

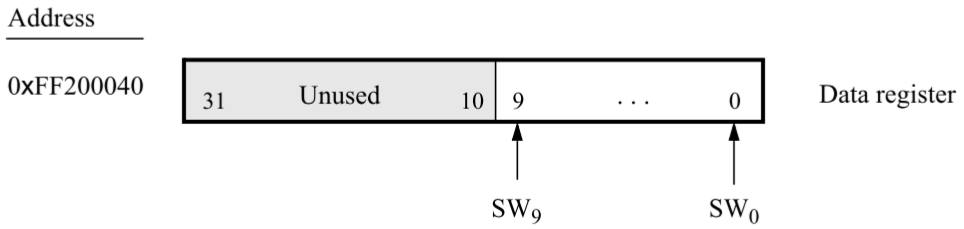


- HEX (7-Segment)

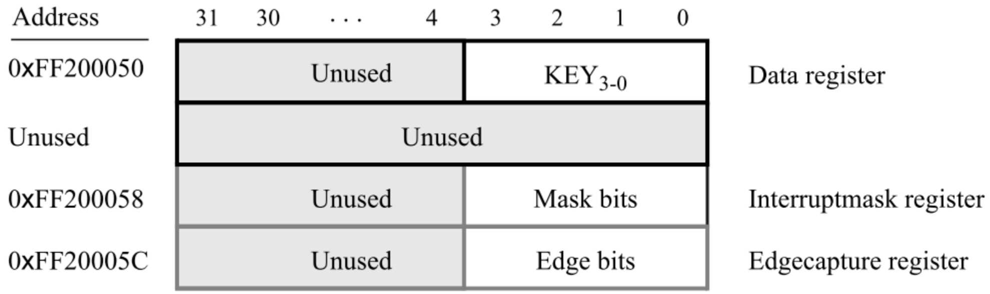


Parallel Port in FPGA, Cont’d

- SW (Slider Switch)

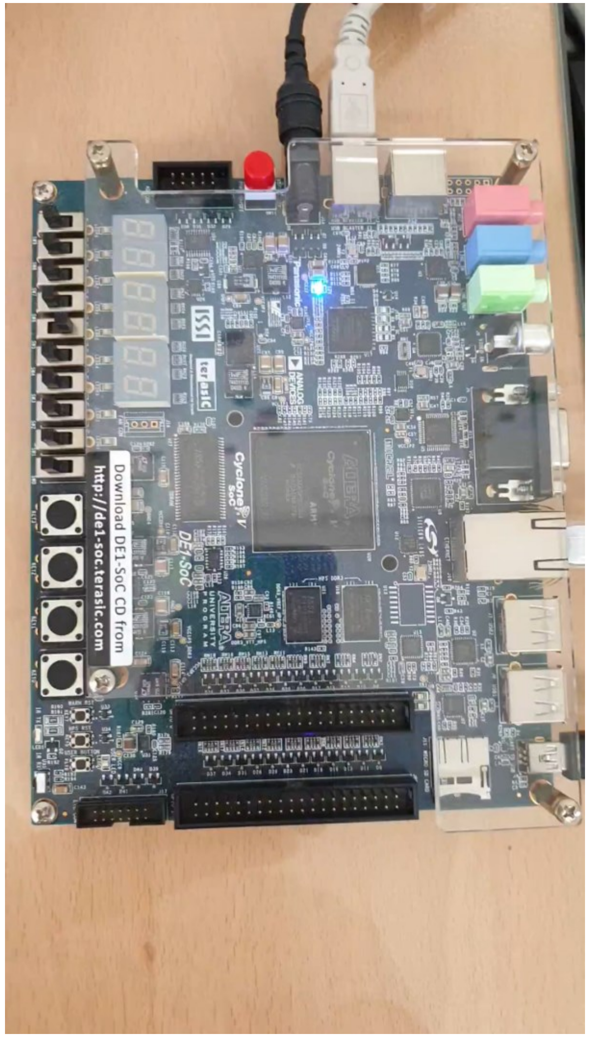


- Key (Pushbutton)



Another Example Program Based on MMIO

```c
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>
#include "address_map_arm.h"

int main(void){
    int fd;
    void *lw_virtual;
    volatile int *ledr;
    volatile int *key;
    int pressed;

    fd = open("/dev/mem", (O_RDWR | O_SYNC));
    lw_virtual = mmap (NULL, LW_BRIDGE_SPAN, (PROT_READ | PROT_WRITE), MAP_SHARED, fd, LW_BRIDGE_BASE);
    ledr = (volatile int *) (lw_virtual + LEDR_BASE);
    key = (volatile int *) (lw_virtual + KEY_BASE);

    *ledr = 0;
    while(1){
        pressed = 0;
        while((*key)&0x1)
                pressed = 1;
        *ledr = *ledr + pressed;
        usleep(5000);
    }

    munmap(lw_virtual, LW_BRIDGE_BASE);
    close(fd);

    return 0;
}
```



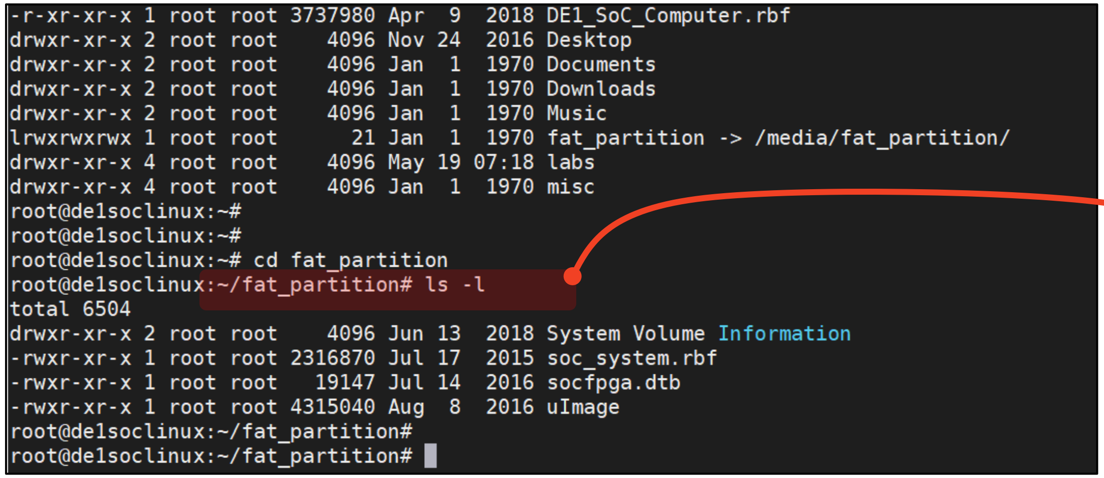
Lab Assignments

- In this lab, you are to implement C programs to access parallel ports, according to the memory-mapped IO. Do not consider employing the interrupt IO.
  1. Implement a program to display all the student IDs in your team through HEXes in a manner like “banner scroll”.

  2. Implement a simple calculator to do the calculations (+, -, x, /) with positive integers of one digit. Use HEXes to display the input / output numbers. Use minimal number of KEYs to input the numbers.

Appendix

- A simple way to move a file between the system in the DE1-SoC board and your host system:
  - ./fat_partition is accessible in your Windows-based host system.
    - You may use this directory as a medium to move the files between the systems.



Linux system in DE1-SoC Board

Windows-based Host System