Lab1 Sensors/Actuators

Polytech Nice Sophia

You will be asked to answer the following questions on the statement R1, R2, …
You will have to write, test and then show the teacher the following codes C1, C2, …

Open the Nios-II SBT for Eclipse software, create a new "Nios-II application and BSP from template".
Choose :

- SOPC information file name: file with extension **.sopcinfo** provided on moodle
- name: a project name without spaces (the path to the project must not contain spaces either)
- template : Hello_world

Click Finish, cross-compile the project and test the code (see Appendix 1).

1) Management of peripherals

Create a first program to manage the following peripherals :

- 10 red LEDs,
- 7-segment displays from 0 to 5,
- 10 switches.
- The purpose of this first code is to find the display table of the 10 decimal digits on the 7-segment displays. You will write the configuration read on the switches on the red LEDs and on the first 7-segment display (on the right), the others remaining at 0.

To do this, use the two functions of reading and writing to memory mapped devices (include altera_avalon_pio_regs.h) :

- `IOWR_ALTERA_AVALON_PIO_DATA (address, data)`
- `data = IORD_ALTERA_AVALON_PIO_DATA (address)`

The type of data depends on the number of bits of the device used. Device addresses are defined as macros in the system.h file (in the BSP project).

R1. From this system.h file, list all the peripherals of this microcontroller (excluding memories).

Peripheral	Address

C1. Test the values on the switches.
R2. Fill in the following display table (Appendix 2).
The wait between 2 read/write will be done by the usleep(microseconds) function from `<unistd.h>`.

Digit	Binary configuration	Hexa configuration
0		
1		
2		
3		
4		
5		
6		
7		
8		
9		

This table should be used to define a translation table or LUT (Look Up Table) :
int LUT [10] = {config0, config1, config2, ... config9} ;
where each configuration will be written in hexadecimal, for example: 0xff.
C2. Test by sending the value of a counter from 0 to 9999 on the 4 7-segment displays.

2) Programmation of interrupts

Now that the peripherals are properly configured, we will program the processor interrupts to read the push button configuration.
As for any sensor or input device reading, two methods of reading are possible:
- by polling,
- by interruption.

R3. Recall the advantages and disadvantages of each method.

Access method	Pros and cons
Polling	
Interruptions	

For the programming of the interrupts, we will follow the approach seen in course, using the code provided in appendix 3 and replacing the constants ADDRESS_BASE and NUMBER_IRQ by the names defined in system.h (in the BSP project).

R4. What is the role of the following prefixes, often used in embedded systems: volatile on the status variable of the push buttons, static on the interrupt routine?

Prefix	Role
Volatile	
Static	

C3. Write a code that indicates the number of the button pressed on the 7-segment displays. The red LEDs should light up to indicate the switches in the up position. This switch configuration will be used to program the waiting time in the main loop (the more switches the user lifts, the longer the maximum waiting time is).

3) Reflex game

C4. Now that the peripherals are programmable, create a code that meets the following specifications:

- to start the test, the user must press button 1,
- all red LEDs will then light up after a random time,
- during this time the red LEDs blink regularly (even and then odd LEDs...),
- the user should press button 4 (left) as quickly as possible,
- the system measures the user's reaction time between switching on and pressing the button, and displays this value: seconds and milliseconds on the 7 segments. Use the timer device for this purpose (appendix 4).
- Switches are used to provide the maximum waiting time before ignition,
- a new reflex test is started by pressing button 1,
- by pressing button 2, the system displays the average reaction time since the beginning of the tests.

The interrupt routine code in Appendix 3 shall not be modified to keep the interrupt latency as short as possible.

Time (s,ms)	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5
User 1					
User 2					

Annexe 1 – Programming the DE1-SoC / DE10-SoC board

To Configure the FPGA circuit on the target (one time only):

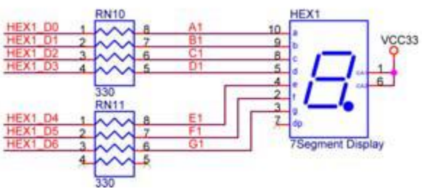
- connect the board to the USB Blaster port
- switch on the board (red button)
- under Eclipse, launch Nios II > Quartus II Programming, and add the following file
    - autodetect
    - modify the second chip with the one of the following file according to your board
    - DE1_Quartus16_1.sof
    - or DE10_Quartus16_1.sof
- Click start to configure the FPGA

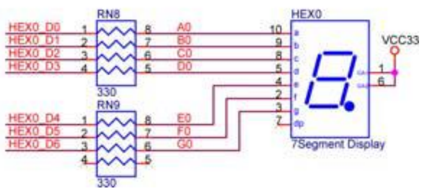
To download the executable (.elf file) into the memory of the microcontroller:

- compile the project
- Start Run > Run Configuration ...
- New NiosII Hardware
- refresh connections
- Run (be sure Switch 9 is up)

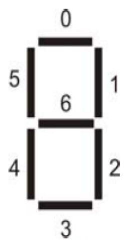
Annexe 2 – 7 segments display

Routing of 7-segment displays on the PCB (example on Hex 0 and Hex 1)





Numbering of display configuration bits :



Annexe 3 - Programming interrupts.

Global variable for button capture :

```
volatile int edge_capture;
```

Interrupt handler:

```
static void handle_button_interrupts(void* context, alt_u32 id)
{
/* Cast context to edge_capture's type. It is important that this
be declared volatile to avoid unwanted compiler optimization. */
volatile int* edge_capture_ptr = (volatile int*) context;
/* Read the edge capture register on the button PIO. Store value. */
*edge_capture_ptr =
```

```
      IORD_ALTERA_AVALON_PIO_EDGE_CAP (ADDRESS_BASE);
/* Write to the edge capture register to reset it. */
IOWR_ALTERA_AVALON_PIO_EDGE_CAP (ADDRESS_BASE, 0);
/* Read the PIO to delay ISR exit. This is done to prevent a
spurious interrupt in systems with high processor -> pio
latency and fast interrupts. */
IORD_ALTERA_AVALON_PIO_EDGE_CAP (ADDRESS_BASE);
}
```

Register the interrupt handler

```
static void init_button_pio()
{
/* Recast the edge_capture pointer to match the alt_irq_register() function
prototype. */
void* edge_capture_ptr = (void*) &edge_capture;
/* Enable all 4 button interrupts. */
IOWR_ALTERA_AVALON_PIO_IRQ_MASK (ADDRESS_BASE, 0xf);
/* Reset the edge capture register. */
IOWR_ALTERA_AVALON_PIO_EDGE_CAP (ADDRESS_BASE, 0x0);
/* Register the ISR. */
alt_irq_register( NUMERO_IRQ,edge_capture_ptr,handle_button_interrupts );
}
```

Annexe 4 – Timer

Configuration of the timer

Under Nios SBT, open the BSP editor and change the configuration to activate the timestamp_timer and deactivate the sys_clk_timer.

First method: **timestamp**

The timestamp access functions are defined in :

```
#include "sys/alt_timestamp.h"
```

alt_timestamp_start(); // intialize the timer
alt_timestamp(); // read the timer
alt_timestamp_freq() // read the frequence of the timer

Second method: **timer**

The timer access functions are defined in :

```
#include "altera_avalon_timer_regs.h"
```

16-bits registers are the following:

- Timer period:

```
IOWR_ALTERA_AVALON_TIMER_PERIODL(address, periodLow) // 16 bits LSB
IOWR_ALTERA_AVALON_TIMER_PERIODH(address, periodHigh)// 16 bits MSB
```

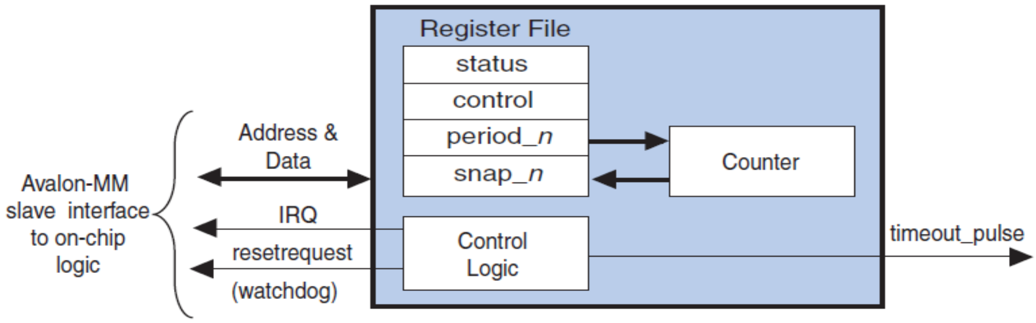
- Timer mode must be set to START :

```
IOWR_ALTERA_AVALON_TIMER_CONTROL(address,ALTERA_AVALON_TIMER_CONTROL_START_MSK)
```

- To read the timer, you must first write it :

```
IOWR_ALTERA_AVALON_TIMER_SNAPL(address, 0x01)
t1 = IORD_ALTERA_AVALON_TIMER_SNAPL (address)
IOWR_ALTERA_AVALON_TIMER_SNAPH(address, 0x01)
t2 = IORD_ALTERA_AVALON_TIMER_SNAPL (address)
```

```
t = t2<<16 | t1
```



Annex 5 - To install drivers on Ubuntu:

1. Download this file https://github.com/MIAOU-Polytech/SI3_PEP_Project/blob/master/doc/51-usbblaster.rules

```
2. $ sudo mv 51-usbblaster.rules /etc/udev/rules.d/
3. $ sudo udevadm control -reload
4. $ sudo udevadm trigger
```