Lab#5

– Digital Electronics –

Using interrupts to program a NIOS system

Students:

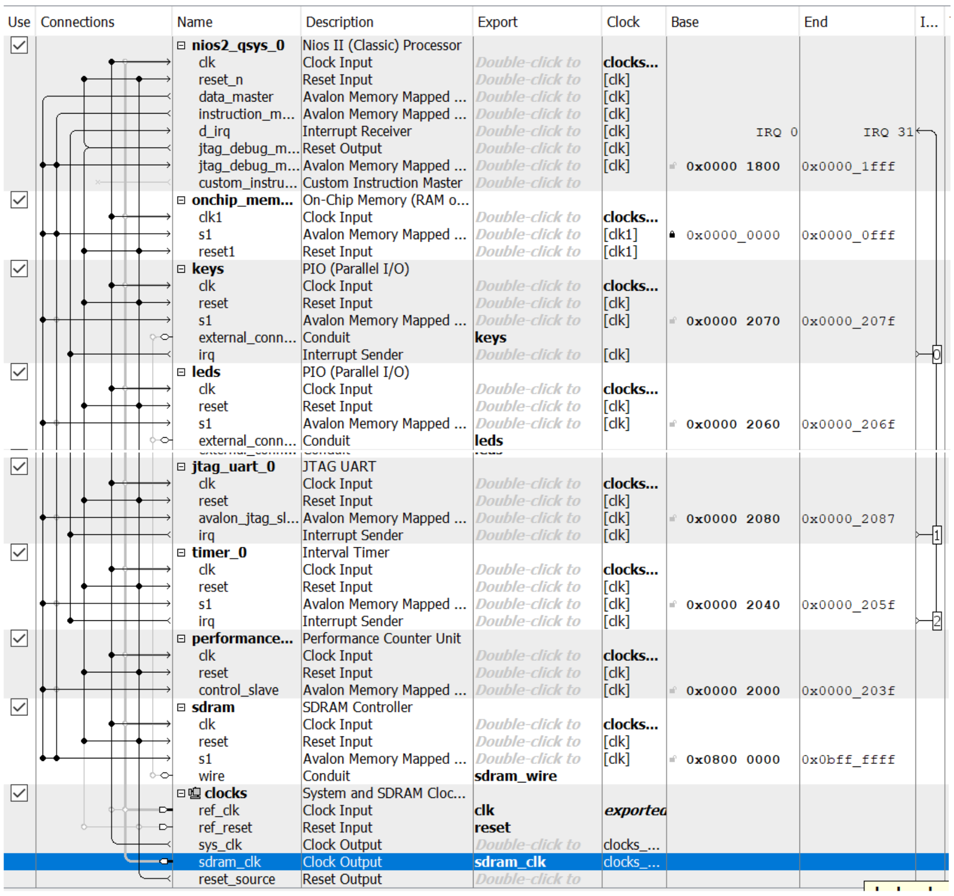
Masson Nicolas s293826

Callegarin Demangeat Martin s294064

Wojciech Paluszkiewicz s293958

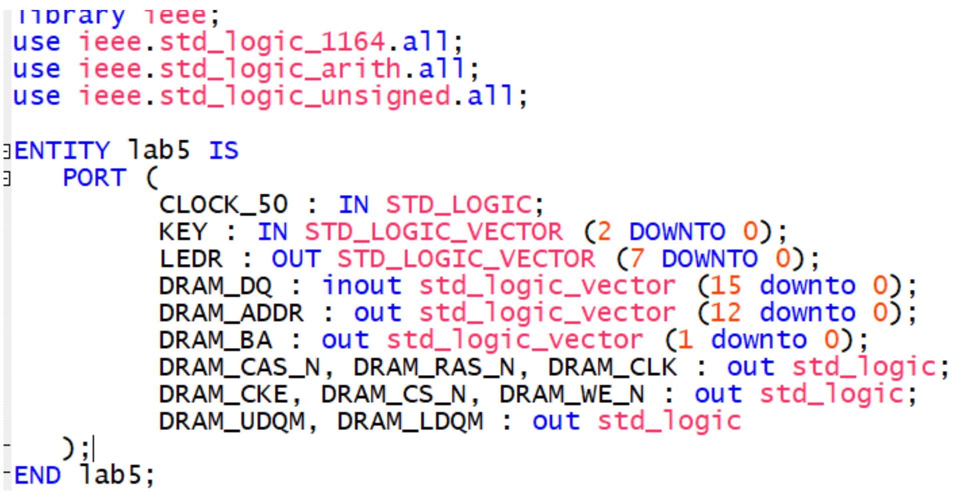
Muhammad Arshad s288059

After manipulating for the first time a NIOS system with the previous lab, we are now going further by introducing and implementing interruption in our projects. An interruption consists in stopping the execution of the main program to perform an interrupt routine. considered as a priority. It is necessary to use interruption in some cases: response to critical events (emergency button, smoke sensor...) or receiving a data frame on the serial port.



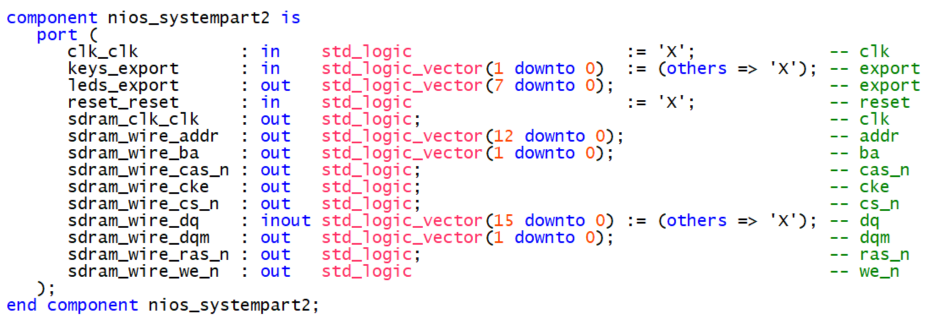
Use	Connections	Name	Description	Export	Clock	Base	End	I...
✓		⊟ **nios2_qsys_0**	Nios II (Classic) Processor					
		clk	Clock Input	*Double-click to*	**clocks...**			
		reset_n	Reset Input	*Double-click to*	[clk]			
		data_master	Avalon Memory Mapped ...	*Double-click to*	[clk]			
		instruction_m...	Avalon Memory Mapped ...	*Double-click to*	[clk]			
		d_irq	Interrupt Receiver	*Double-click to*	[clk]	IRQ 0	IRQ 31	
		jtag_debug_m...	Reset Output	*Double-click to*	[clk]			
		jtag_debug_m...	Avalon Memory Mapped ...	*Double-click to*	[clk]	0x0000 1800	0x0000_1fff	
		custom_instru...	Custom Instruction Master	*Double-click to*				
✓		⊟ **onchip_mem...**	On-Chip Memory (RAM o...					
		clk1	Clock Input	*Double-click to*	**clocks...**			
		s1	Avalon Memory Mapped ...	*Double-click to*	[clk1]	0x0000_0000	0x0000_0fff	
		reset1	Reset Input	*Double-click to*	[clk1]			
✓		⊟ **keys**	PIO (Parallel I/O)					
		clk	Clock Input	*Double-click to*	**clocks...**			
		reset	Reset Input	*Double-click to*	[clk]			
		s1	Avalon Memory Mapped ...	*Double-click to*	[clk]	0x0000 2070	0x0000_207f	
		external_conn...	Conduit	**keys**				
		irq	Interrupt Sender	*Double-click to*	[clk]			0
✓		⊟ **leds**	PIO (Parallel I/O)					
		clk	Clock Input	*Double-click to*	**clocks...**			
		reset	Reset Input	*Double-click to*	[clk]			
		s1	Avalon Memory Mapped ...	*Double-click to*	[clk]	0x0000 2060	0x0000_206f	
		external_conn...	Conduit	**leds**				
✓		⊟ **jtag_uart_0**	JTAG UART					
		clk	Clock Input	*Double-click to*	**clocks...**			
		reset	Reset Input	*Double-click to*	[clk]			
		avalon_jtag_sl...	Avalon Memory Mapped ...	*Double-click to*	[clk]	0x0000 2080	0x0000_2087	
		irq	Interrupt Sender	*Double-click to*	[clk]			1
✓		⊟ **timer_0**	Interval Timer					
		clk	Clock Input	*Double-click to*	**clocks...**			
		reset	Reset Input	*Double-click to*	[clk]			
		s1	Avalon Memory Mapped ...	*Double-click to*	[clk]	0x0000 2040	0x0000_205f	
		irq	Interrupt Sender	*Double-click to*	[clk]			2
✓		⊟ **performance...**	Performance Counter Unit					
		clk	Clock Input	*Double-click to*	**clocks...**			
		reset	Reset Input	*Double-click to*	[clk]			
		control_slave	Avalon Memory Mapped ...	*Double-click to*	[clk]	0x0000 2000	0x0000_203f	
✓		⊟ **sdram**	SDRAM Controller					
		clk	Clock Input	*Double-click to*	**clocks...**			
		reset	Reset Input	*Double-click to*	[clk]			
		s1	Avalon Memory Mapped ...	*Double-click to*	[clk]	0x0800 0000	0x0bff_ffff	
		wire	Conduit	**sdram_wire**				
✓		⊟ **clocks**	System and SDRAM Cloc...					
		ref_clk	Clock Input	**clk**	*exported*			
		ref_reset	Reset Input	**reset**				
		sys_clk	Clock Output	*Double-click to*	clocks_...			
		sdram_clk	Clock Output	**sdram_clk**	clocks_...			
		reset_source	Reset Output	*Double-click to*				

Figure 1 : Nios system of the circuit.

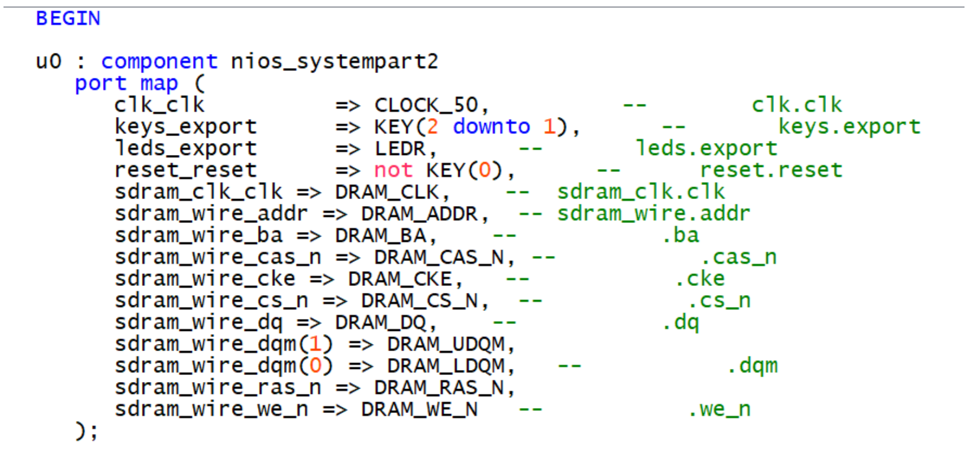


```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

ENTITY lab5 IS
   PORT (
         CLOCK_50 : IN STD_LOGIC;
         KEY : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
         LEDR : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
         DRAM_DQ : inout std_logic_vector (15 downto 0);
         DRAM_ADDR : out std_logic_vector (12 downto 0);
         DRAM_BA : out std_logic_vector (1 downto 0);
         DRAM_CAS_N, DRAM_RAS_N, DRAM_CLK : out std_logic;
         DRAM_CKE, DRAM_CS_N, DRAM_WE_N : out std_logic;
         DRAM_UDQM, DRAM_LDQM : out std_logic
   );
END lab5;
```

Figure 2 : Top entity of the VHDL code.



```vhdl
component nios_systempart2 is
   port (
      clk_clk          : in    std_logic                     := 'X';              -- clk
      keys_export      : in    std_logic_vector(1 downto 0)  := (others => 'X'); -- export
      leds_export      : out   std_logic_vector(7 downto 0);                     -- export
      reset_reset      : in    std_logic                     := 'X';              -- reset
      sdram_clk_clk    : out   std_logic;                                         -- clk
      sdram_wire_addr  : out   std_logic_vector(12 downto 0);                    -- addr
      sdram_wire_ba    : out   std_logic_vector(1 downto 0);                     -- ba
      sdram_wire_cas_n : out   std_logic;                                         -- cas_n
      sdram_wire_cke   : out   std_logic;                                         -- cke
      sdram_wire_cs_n  : out   std_logic;                                         -- cs_n
      sdram_wire_dq    : inout std_logic_vector(15 downto 0) := (others => 'X'); -- dq
      sdram_wire_dqm   : out   std_logic_vector(1 downto 0);                     -- dqm
      sdram_wire_ras_n : out   std_logic;                                         -- ras_n
      sdram_wire_we_n  : out   std_logic                                          -- we_n
   );
end component nios_systempart2;
```

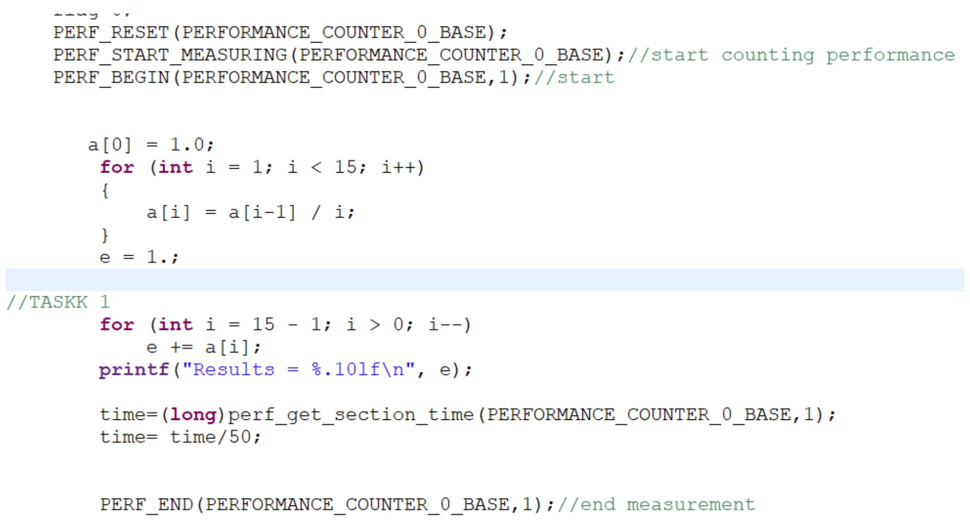


```vhdl
  BEGIN

  u0 : component nios_systempart2
     port map (
        clk_clk          => CLOCK_50,        --         clk.clk
        keys_export      => KEY(2 downto 1),      --         keys.export
        leds_export      => LEDR,      --        leds.export
        reset_reset      => not KEY(0),      --        reset.reset
        sdram_clk_clk => DRAM_CLK,      --  sdram_clk.clk
        sdram_wire_addr => DRAM_ADDR,   -- sdram_wire.addr
        sdram_wire_ba => DRAM_BA,     --           .ba
        sdram_wire_cas_n => DRAM_CAS_N, --             .cas_n
        sdram_wire_cke => DRAM_CKE,    --            .cke
        sdram_wire_cs_n => DRAM_CS_N,   --            .cs_n
        sdram_wire_dq => DRAM_DQ,     --           .dq
        sdram_wire_dqm(1) => DRAM_UDQM,
        sdram_wire_dqm(0) => DRAM_LDQM,   --             .dqm
        sdram_wire_ras_n => DRAM_RAS_N,
        sdram_wire_we_n => DRAM_WE_N   --            .we_n
     );
```

Figure 3 & 4 : Introducing the nios as component and liking to peripherals of the FPGA

Do it yourself :

- Compute and display the first 15 successive approximations of the mathematical constant e (i.e. Euler's Number) calculated as: $e_i = \sum_{n=0}^{i} \frac{1}{n!}$



```c
PERF_RESET(PERFORMANCE_COUNTER_0_BASE);
PERF_START_MEASURING(PERFORMANCE_COUNTER_0_BASE);//start counting performance
PERF_BEGIN(PERFORMANCE_COUNTER_0_BASE,1);//start


   a[0] = 1.0;
    for (int i = 1; i < 15; i++)
    {
        a[i] = a[i-1] / i;
    }
    e = 1.;
//TASKK 1
    for (int i = 15 - 1; i > 0; i--)
        e += a[i];
    printf("Results = %.10lf\n", e);

    time=(long)perf_get_section_time(PERFORMANCE_COUNTER_0_BASE,1);
    time= time/50;


    PERF_END(PERFORMANCE_COUNTER_0_BASE,1);//end measurement
```

Figure 5 : C code to calculate Euler’s number approximation

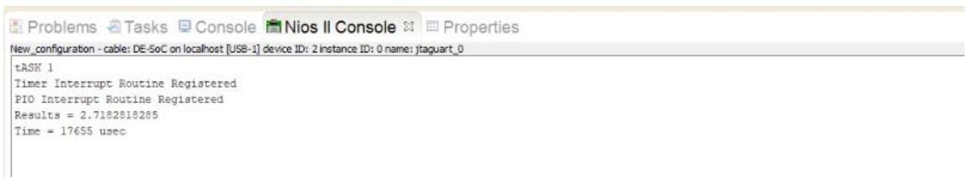
By searching on the internet a way to calculate an approximation of Euler's Number we found the Taylor expansion method described with the formula above. That leads us to calculate 1+1/1+1/(1*2)+1/(1*2*3)+1/(n!) . So we created a *for* loop with n=15.

To measure its execution time we use a performance counter that can simultaneously measure several sections of our program using a 64 bit time-counter which measures the total time spent in the specified code section with a single clock resolution.
perf_get_section_time(base,counter_number) returns the value of the elapsed time as number of clock cycles. It is necessary to add the performance reset, start, begin and end.
Inside the printf we use *%.10* to set the decimal digits of the result to 10.

By writing the report we just realised that we should put the *printf* inside the last *for* loop to display each iteration with 10 digits.

If we had made this change we would have a larger execution time because each line of code take a certain execution time. An order of few microseconds for a single *printf* or a *cout*.

By compiling our code we have the expected result :



```
New_configuration - cable: DE-SoC on localhost [USB-1] device ID: 2 instance ID: 0 name: jtaguart_0
tASK 1
Timer Interrupt Routine Registered
PIO Interrupt Routine Registered
Results = 2.7182818285
Time = 17655 usec
```

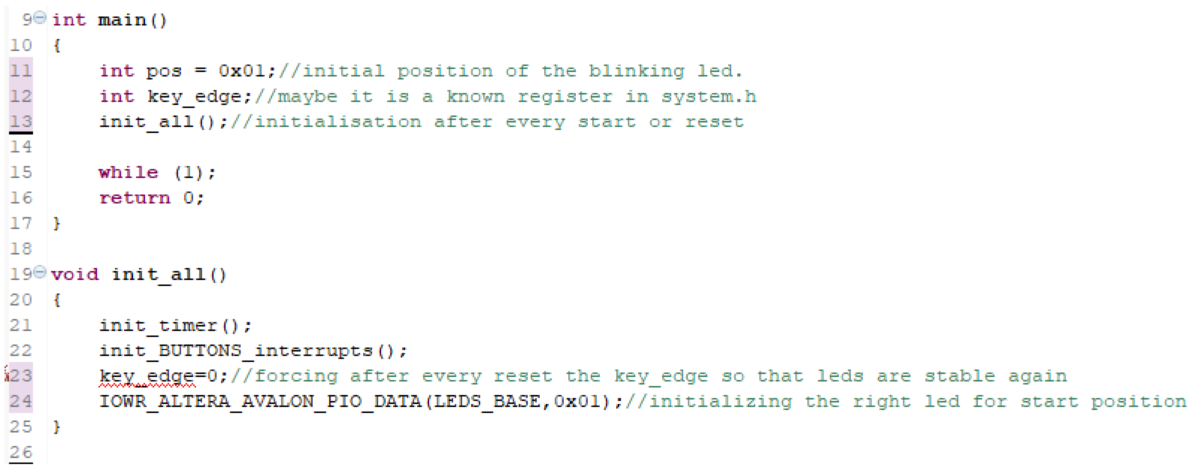
<u>Figure 6 : Euler's number result and execution time</u>

- Display on the 8 red LEDs a light rotating to the left if KEY(2) has been pressed and rotating to the right if KEY(1) has been pressed. If no key has been pressed a single led will be ON but not moving. The moving interval must be ¼ of a second.

For this part we started from the code given in the appendix of lab5 pdf but some changes must have been made.

2nd task : making a led move.

The idea of the code is first to initialize the system and especially the interrupting components. First one is the timer_0 with it's interruption period. Then we also authorize the buttons to interrupte. Then, by coding the ISR of the timer, we can drive the led displaying. For the next pages, we will comment a code we wrote, hoping it would work but without truly knowing since we couldn't send it to the card due to not understanble errors like "make**** something...".

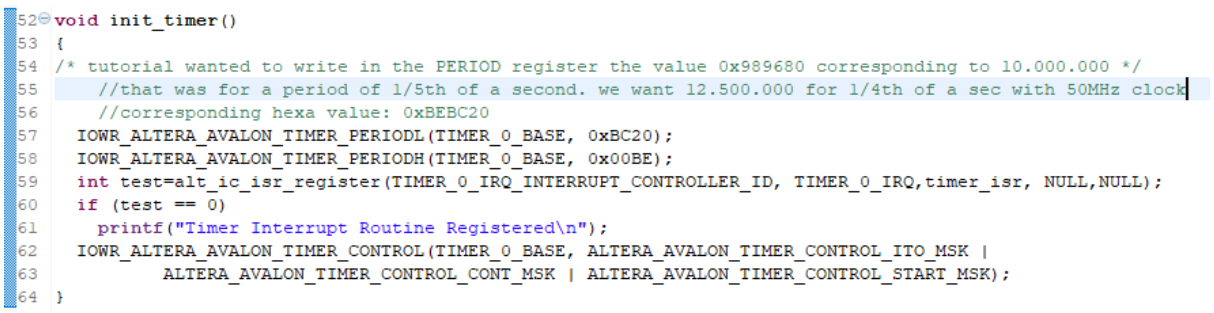


```c
int main()
{
    int pos = 0x01;//initial position of the blinking led.
    int key_edge;//maybe it is a known register in system.h
    init_all();//initialisation after every start or reset

    while (1);
    return 0;
}

void init_all()
{
    init_timer();
    init_BUTTONS_interrupts();
    key_edge=0;//forcing after every reset the key_edge so that leds are stable again
    IOWR_ALTERA_AVALON_PIO_DATA(LEDS_BASE,0x01);//initializing the right led for start position
}
```

Figure 7 : Initialization routine and main

up above is the begining of the code. We won't ever come out of the while(1) loop except due to interruptions from the timer and keys button.

During the main, we call the void init_all(), which will itself call two void of initalization. The timer initialization and the key one.

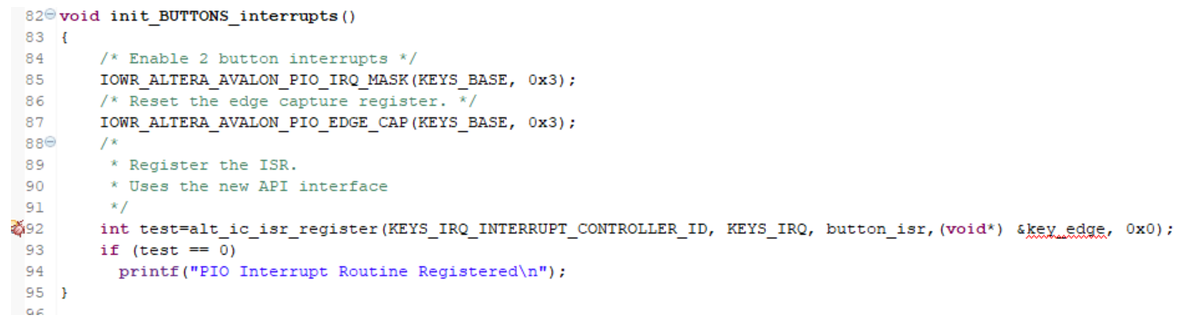


```c
void init_timer()
{
/* tutorial wanted to write in the PERIOD register the value 0x989680 corresponding to 10.000.000 */
    //that was for a period of 1/5th of a second. we want 12.500.000 for 1/4th of a sec with 50MHz clock
    //corresponding hexa value: 0xBEBC20
  IOWR_ALTERA_AVALON_TIMER_PERIODL(TIMER_0_BASE, 0xBC20);
  IOWR_ALTERA_AVALON_TIMER_PERIODH(TIMER_0_BASE, 0x00BE);
  int test=alt_ic_isr_register(TIMER_0_IRQ_INTERRUPT_CONTROLLER_ID, TIMER_0_IRQ,timer_isr, NULL,NULL);
  if (test == 0)
    printf("Timer Interrupt Routine Registered\n");
  IOWR_ALTERA_AVALON_TIMER_CONTROL(TIMER_0_BASE, ALTERA_AVALON_TIMER_CONTROL_ITO_MSK |
          ALTERA_AVALON_TIMER_CONTROL_CONT_MSK | ALTERA_AVALON_TIMER_CONTROL_START_MSK);
}
```

Figure 8 : Timer initialization routine

As we can see, the period register will be driving the interruptions frequency. As we have a lot of usage based on periodic interruptions, we have at disposition the period register divided in two register. One is the MSB and the other the LSB of the said period register.

With a 50MHz clock, if we want the led to move every 1/4th of a second, we must count 12.5Millions clock cycle between each interruptions. Translating 12.500.000 in hexa gives BE BC20.

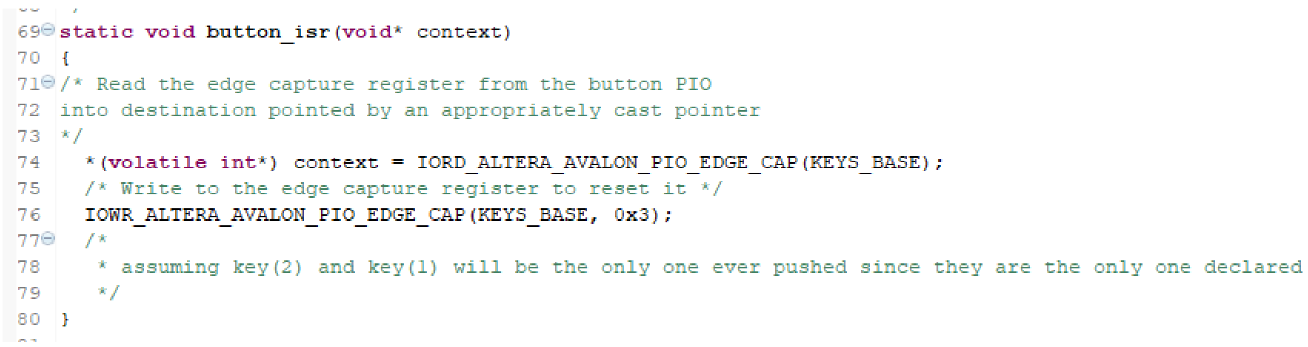


```c
void init_BUTTONS_interrupts()
{
    /* Enable 2 button interrupts */
    IOWR_ALTERA_AVALON_PIO_IRQ_MASK(KEYS_BASE, 0x3);
    /* Reset the edge capture register. */
    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE, 0x3);
    /*
     * Register the ISR.
     * Uses the new API interface
     */
    int test=alt_ic_isr_register(KEYS_IRQ_INTERRUPT_CONTROLLER_ID, KEYS_IRQ, button_isr,(void*) &key_edge, 0x0);
    if (test == 0)
      printf("PIO Interrupt Routine Registered\n");
}
```

Figure 9 : Buttons initialization

There is the need to declare the keys as possible interrupting device even after plugin them on the irq port of the nios. So sending the mask of 0x03 corresponding to the first 2 bits.

Another more interesting line in the initialisation is also found in the button ISR. Let's show the button isr and talk about it:



```
69 static void button_isr(void* context)
70 {
71 /* Read the edge capture register from the button PIO
72 into destination pointed by an appropriately cast pointer
73 */
74   *(volatile int*) context = IORD_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE);
75   /* Write to the edge capture register to reset it */
76   IOWR_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE, 0x3);
77   /*
78    * assuming key(2) and key(1) will be the only one ever pushed since they are the only one declared
79    */
80 }
```

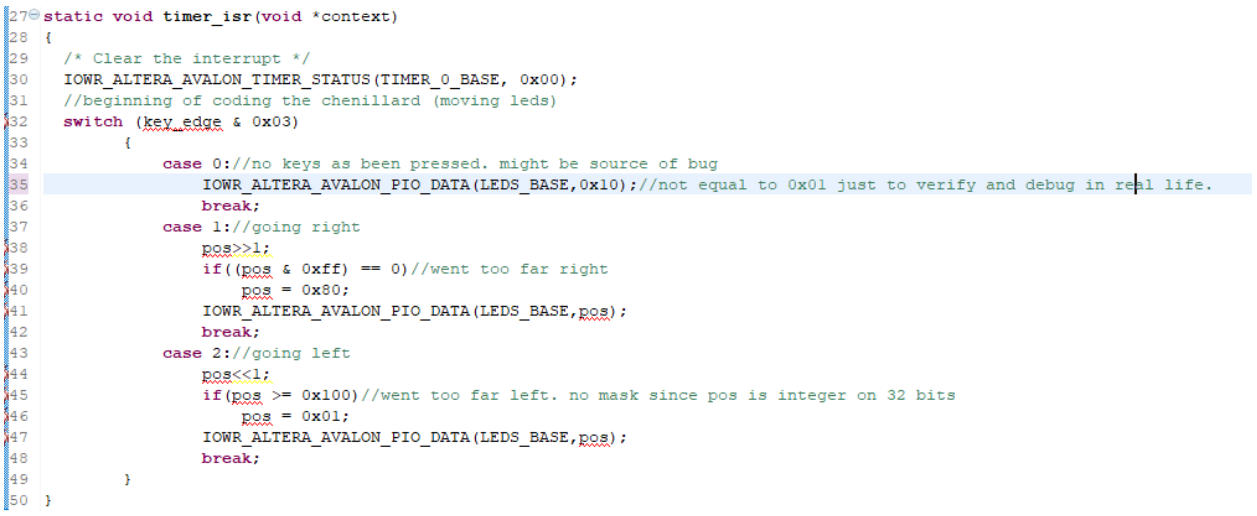
<u>Figure 10 : button_isr</u>

We can see it’s needed to send a positive value on the edge_cap register. This one will produce a signal each time a button is pressed, the isr will start but the edge_cap register would stay at 1 as long as we don’t clear it manually. It’s like a student raising it’s hand to warn of an interruption, we have to give him a high five so he can lower his hand and start paying attention to interruptions once again. We only need to send 0x03 since they are only two bits controling the 2 keys of control.

Key(0) is directly wired to the nios reset.

The line 74 of the button isr is a rather complex one. Context is a pointer to a register that will inform about the key_edge status to others parts of the code. That is why we have to give him the PIO_edge_cap register based on the keys peripherals. and we do this before clearing the edge capturing register otherwise, we won’t send anything to the context pointer.

Next is the most interresting part of the code, the timer ISR.



```
static void timer_isr(void *context)
{
  /* Clear the interrupt */
  IOWR_ALTERA_AVALON_TIMER_STATUS(TIMER_0_BASE, 0x00);
  //beginning of coding the chenillard (moving leds)
  switch (key_edge & 0x03)
        {
            case 0://no keys as been pressed. might be source of bug
                IOWR_ALTERA_AVALON_PIO_DATA(LEDS_BASE,0x10);//not equal to 0x01 just to verify and debug in real life.
                break;
            case 1://going right
                pos>>1;
                if((pos & 0xff) == 0)//went too far right
                    pos = 0x80;
                IOWR_ALTERA_AVALON_PIO_DATA(LEDS_BASE,pos);
                break;
            case 2://going left
                pos<<1;
                if(pos >= 0x100)//went too far left. no mask since pos is integer on 32 bits
                    pos = 0x01;
                IOWR_ALTERA_AVALON_PIO_DATA(LEDS_BASE,pos);
                break;
        }
}
```

<u>Figure 11 : Timer interrupt routine</u>

Or also, what is the system gonna do at every interruptions of the timer. What we want it to do is moving the led according to our wishes.

As for the edge_cap, we must clear the interruption status bit from the timer. This time it’s by sending 0.

The key_edge variable holding the rank of the very last pressed key, it’s gonna be either 00 from the start of after a reset, 01 or 10. Therefore we can make a switch case (with a mask for safety) and then the rest is just using the shifting comand on C which is very usefull in this case.