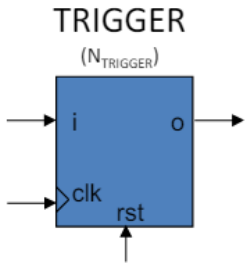
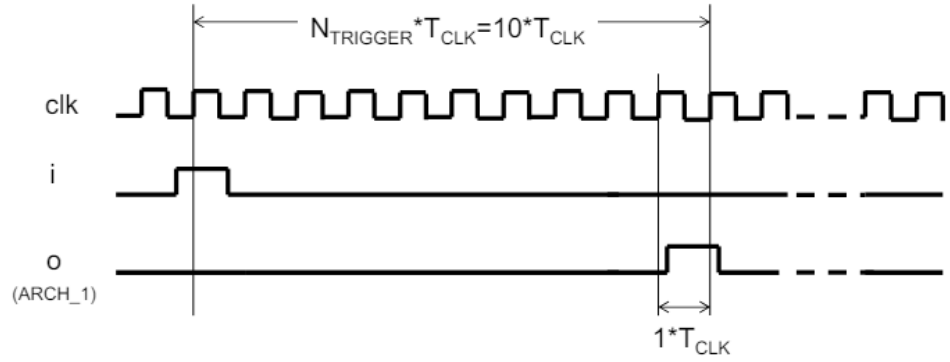
Create a new ISE project (any name and any working directory) for the Spartan-6 of the LX9 Microboard (device: XC6SLX9, package: CSG324, Speed: -3) in order to create and simulate a VHDL design. Copy into the working directory of the project the two VHDL files pack_exam.vhd and trigger.vhd. Then, add both source VHDL files to the project.

The pack_exam.vhd file declares a VHDL package, named PACK_EXAM, and its package body. This package contains the function F_NBITS (returns the number of bits required for a number of counts) which is used in trigger.vhd.

The trigger.vhd file declares the entity TRIGGER and its architecture ARCH_1. This VHDL module contains a synthesizable description of a synchronous circuit which asserts (during a single clock cycle) the output (port o), $N_{TRIGGER}$ clock cycles (T_{CLK}) after the circuit has been triggered,. The circuit is triggered when it detects i='1' in a rising edge of the clock. The $N_{TRIGGER}$ ($N_{TRIGGER} \geq 2$) is a configurable parameter that is configured through a GENERIC. The input ports clk and rst are the clock and reset signals usually used in any synchronous circuit.



The next Figure shows the behaviour of the circuit when $N_{TRIGGER}$=10. The output (o) is depicted with a delta (δ) delay. The circuits asserts, during a single T_{CLK}, the output (o='1') 10*T_{CLK} later after the circuit was triggered by the input (i).

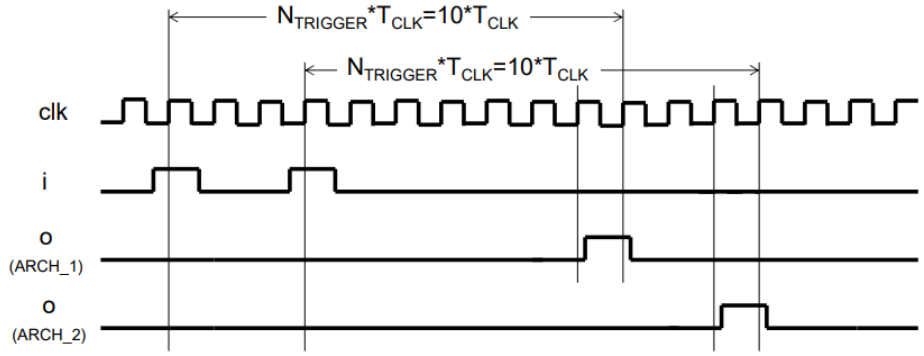


A testbench (entity TB_TRIGGER, architecture TEST, at the bottom of the file trigger.vhd) is also provided to perform the functional simulation, as depicted in the previous figure.

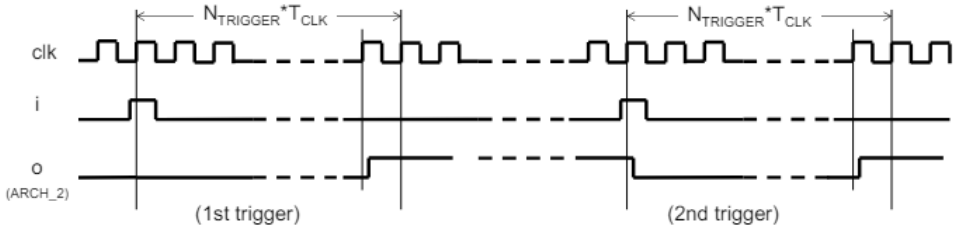
The exercise consists on creating a new architecture (named ARCH_2) which is a modification of the previous one (ARCH_1). The qualification of the exercise is based on the simplicity of the VHDL description and the efficiency (area and speed) of the implementation for the new architecture.

1-Simulate ARCH_1 with $N_{TRIGGER}$=10 to check the circuit behaves as required. Then, simulate when $N_{TRIGGER}$ equals a power of 2 (for instance $N_{TRIGGER}$=8) and check the circuit does not work as expected. The new architecture ARCH_2 must behave as expected also for these cases. Report the reason why the ARCH_1 does not work when $N_{TRIGGER}$ equals a power of 2, and the modifications done in ARCH_2 in order to behave as expected in all cases.

2-Modify the testbench in order to simulate ARCH_1 when it is triggered a second time (or more), before the output is asserted due to the first trigger. The ARCH_1 asserts the output (o) $N_{TRIGGER}*T_{CLK}$ after the first trigger. Modify ARCH_2 to assert the output $N_{TRIGGER}*T_{CLK}$ after the last trigger, as depicted in the next Figure when $N_{TRIGGER}$=10. Report the reason why the ARCH_1 do not take into account the second trigger, and the modifications done in ARCH_2 to assert the output $N_{TRIGGER}*T_{CLK}$ after the last trigger.



3-Modify the testbench in order to simulate ARCH_2 when it is triggered a second time, but after the output is asserted due to the first trigger. The output port (o) in ARCH_2 is asserted during a single clock cycle, as depicted in the previous figures. Modify ARCH_2 in order the output remains asserted after the first trigger but until the next trigger, as showed in the next Figure. Report the modifications done in ARCH_2.



4-Add top.vhd to the project and complete the architecture for the TOP entity in order to implement an instance of ARCH_2 for the TRIGGER in the LX9 MicroBoard. Add to the project the lx9_microboard.ucf UCF file and complete it to use any of the SWITCHES of the FPGA board as the input (i), and any of the LEDs as the output (o).The $N_{TRIGGER}$ must accomplish $N_{TRIGGER}*T_{CLK}$=1 second. Synthetize it and report the number of required IOBs, FFs and LUTs, and justify the results. Also report the top.vhd file and the UCF modifications.

Solution:

1. The error is in the assignation of cnt_end, since it is never asserted to '1' when N_TRIGGER=8. This is because the width of cnt is 3-bit, therefore the range of cnt(2:0) is from "000"=0 to "111"=7. The comparator sentence cnt+1="111"+1="000" will never reach 8="1000"

```
cnt_end<='1' when cnt+1=N_TRIGGER else '0';
```

The easiest solution and more efficient one is the next code. If N_TRIGGER=8, when the cnt="111", then cnt_end='1' since N_TRIGGER-1=7="111"

```
cnt_end<='1' when cnt=N_TRIGGER-1 else '0';
```

2. In the ARCH_1, the start signal will not be asserted to '1' if state=TRIGGERED (the circuit is running). Moreover, the counter cnt continues incrementing since the state=TRIGGERED. Therefore, the ARCH_2 should assert the start signal to '1' always when i='1' (independently from the state), and the counter cnt must be initialized to the number 1 ("00…001") when i='1'. Addtionaly, stop='0' when i='1' and the counter cnt is in the last count (cnt_end='1') to re-start the counter from 1 instead from 0

```
start<=i
stop<=(cnt_end and not i) when state=TRIGGERED else '0';

counter: process
begin
   wait until rising_edge(clk);
   if rst='1' or stop='1' then
         cnt<=(others=>'0');
   elsif start='1' then
         cnt<=(0=>'1', others=>'0');
   elsif state=TRIGGERED then
         cnt<=cnt+1;
   end if;
end process;
```

3. The third modification:

```
reg: process
begin
   wait until rising_edge(clk);
   if rst='1'then
         o<='0';
   else
         if start='1' then o<='0'; end if;
         if stop='1' then o<='1'; end if;
   end if;
end process;
```

4. The top.vhd file:

```
architecture ARCH of TOP is
constant N_TRIGGER: integer:=100e6;    --10ns*100e6=1000e-3(s)=1s
begin
   i0: entity work.TRIGGER(ARCH_2)
         generic map(N_TRIGGER)
         port map(fpga_clk,fpga_rst,switch,led);
end architecture;
```

The lx9_microboard.ucf file:

```
NET fpga_clk TNM_NET = clk_net;
TIMESPEC TS_clk_net = PERIOD clk_net 100 MHz;
NET fpga_clk                    LOC=C10;    #100MHz clock input
NET fpga_rst                    LOC=V4;     #USER PUSH-BUTTON
NET switch                      LOC=B3;     #SWITCH-0
NET led                         LOC=P4;     #LED-0
```

Some of thee synthetized results (of the last architecture with all of the previously reported modifications):

```
Slice Logic Utilization:
 Number of Slice Registers:            29  out of  11440      0%
 Number of Slice LUTs:                 63  out of   5720      1%
    Number used as Logic:              63  out of   5720      1%

IO Utilization:
 Number of IOs:                         4
 Number of bonded IOBs:                 4  out of    200      2%

Specific Feature Utilization:
 Number of BUFG/BUFGCTRLs:              1  out of     16      6%
```

The number of IOs is 4 since there are three 1-bit inputs (fpga_clk,fpga_rst,switch) and one 1-bit output (led)

The number of BUFG is one since it is devoted to drive the internal clock signal (from the fpga_clk input)

The FFs are due to the rising_edge(clk) condition in a process. The number of FFs for the 1-bit output o is 1. The number of FFs for the internal signal state is 1 (since T_STATE has two

possible values, and therefore only one bit is needed to encode the state). The number of FFs for the internal cnt is 27 (N_BIT=27 since it is the minimum number of bits that accomplishes 2**N_BIT >= 100e6=N_TRIGGER). Therefore, the total number of FFs is 1+1+27=29. These conclussions can also be observed during the synthesis

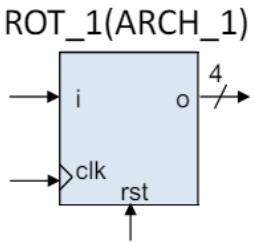
```
Synthesizing Unit <TRIGGER>.
    Related source file is "trigger.vhd".
        N_TRIGGER = 100000000
    Found 1-bit register for signal <state>.
    Found 27-bit register for signal <cnt>.
    Found 1-bit register for signal <o>.
    Found 27-bit adder for signal <cnt[26]_GND_6_o_add_1_OUT> ...
    ...
```

Create a new ISE project (any name and any working directory) for the Spartan-6 of the LX9 Microboard (device: XC6SLX9, package: CSG324, Speed: -3) in order to create and simulate a VHDL design. Copy into the working directory of the project the two VHDL files pack_exam.vhd and rot.vhd. Then, add both source VHDL files to the project.

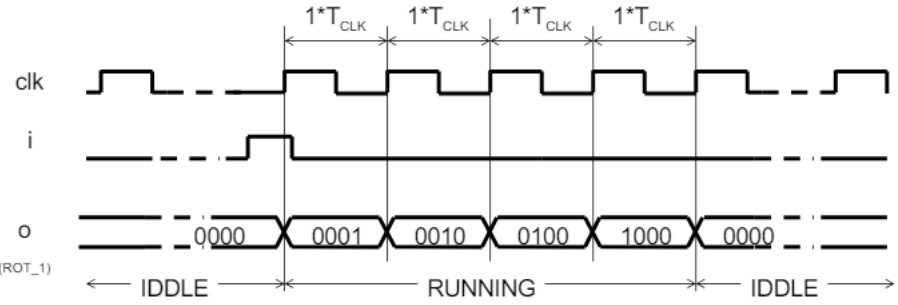
The pack_exam.vhd file declares a VHDL package, named PACK_EXAM, and its package body. This package contains the function F_ROT_LEFT (describes the left rotation of a std_logic_vector) which is used in rot.vhd.

The rot.vhd file declares the entity ROT_1 and its architecture ARCH_1. This VHDL module contains a synthesizable description of a synchronous circuit which implements two states: IDDLE and RUNNING. The input port (i) changes the state from IDDLE to RUNNING, according to:

1. Initially, after a reset (rst='1'), the state goes to IDDLE and o="0000". The state remains IDDLE while the i='0'.
2. The state changes to RUNNING and the output o="0001" when i='1' at a rising edge of the clock (clk).
3. While the state is RUNNING, the output o rotates at each clock cycle (T_{CLK}), following the sequence "0001", "0010", "0100", "1000", independently of the input i
4. In the next clock cycle, the state changes to IDDLE and o="0000", as initially.



The next Figure shows the behaviour of the circuit (after reset)



A testbench (entity TB_ROT, architecture TEST (at the end of the file rot.vhd) is also provided to perform the functional simulation.

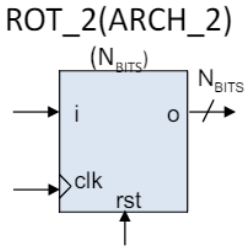
The exercise consists on creating a new entity (named ROT_2) and its architecture (ARCH_2) which is a modification of the previous ones (ROT_1 and ARCH_1). Copy ROT_1 and ARCH_1 and change the entity name to ROT_2 and architecture name to ARCH_2.

The qualification of the exercise is based on the simplicity of the VHDL description and the efficiency (area and speed) of the implementation for the new architecture.

1-Modify ROT_2 and ARCH_2 to parametrize the number of bits (N_{BITS}) of the output using a VHDL generic. For instance, when N_{BITS}=10 the behaviour is the same as before but with a larger number of bits at the output:

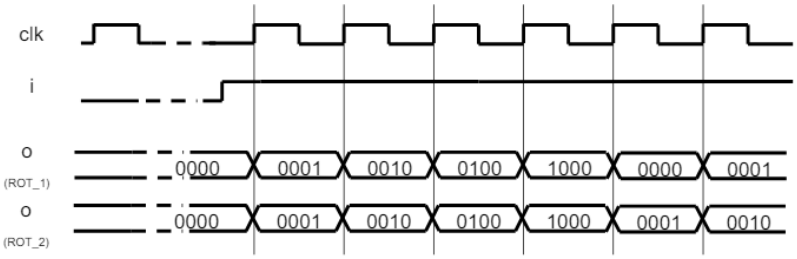
o="0000000000" in IDDLE state

o="0000000001", "0000000010", … , "0100000000", "1000000000" in RUNNING state



2-Modify the TB_ROT to simulate ROT_1(ARCH_1) and ROT_2(ARCH_2) concurrently. The NBITS parameter should be changed in the TB_ROT to N_{BITS}=4 and N_{BITS}=10 to check the behaviour of ROT_2(ARCH_2)

3-Modify the testbench in order to simulate ROT_1(ARCH_1) when i='1' during a large number of T_{CLK}, to check the output (o) goes from "1000" to "0000", and then to "0001". Modify the ROT_2(ARCH_2) in order its output (when N_{BITS}=4) changes from "1000" to "0001" directly (the "0000" is avoided when i='1'), as depicted in the next Figure



4-Add the provided top.vhd and lx9_microboard.ucf files to the project. Complete them in order to implement an instance of ARCH_2 for the ROT_2 and attach it to any of the SWITCHES of the FPGA LX9 MicroBoard board as the input (i), and the fourt LEDs of the FPGA board as the output (o). Synthetize it and report the number of required IOBs, FFs and LUTs, and justify the results. Also report the top.vhd file and the UCF modifications.

Solution:

Solution:

1. For the N_{BITS} parameter, see modification 1 at the ROT_2 and ARCH_2
2. For the simulation of ROT_2(ARCH_2) see Modification 2 at TB_ROT
3. For the modification, see Modification 3 Start to End at ARCH_2

```
entity ROT_2 is
	generic(
		NBITS: integer:=10 ); --any natural number>=1		--Modification 1
	port(
		clk: in std_logic;
		rst: in std_logic;
		i: in std_logic;
		o: out std_logic_vector(NBITS-1 downto 0) );		--Modification 1
end entity;

architecture ARCH_2 of ROT_2 is
signal q: std_logic_vector(o'range);
type T_STATE is (IDLE, RUNNING);
signal state: T_STATE;
begin
	o<=q;

	p: process
	begin
		wait until rising_edge(clk);
		if rst='1' then
			state<=IDLE;
			q<=(others=>'0');				--Modification 1
		else case state is
			when IDLE=>
				if i='1' then
					state<=RUNNING;
					q(0)<='1';
				end if;
			when RUNNING=>
				q<=F_ROT_LEFT(q);
				if q(q'high)='1' then		--Modification 1
				--or q(NBITS-1) or q(q'left)
					state<=IDLE;
					if i='1' then			--Modification 3 Start
						state<=RUNNING;
						q(0)<='1';
					end if;					--Modification 3 End
				end if;
		end case; end if;
	end process;



end architecture;

architecture TEST of TB_ROT is
constant T_CLK: time:=10 ns;
constant NBITS: integer:=10; --constant NBITS: integer:=4;		--Modification 2
signal clk: std_logic:='0';
signal rst,i: std_logic;
signal o_1: std_logic_vector(3 downto 0);
signal o_2: std_logic_vector(NBITS-1 downto 0);				--Modification 2
begin
	clk<=not clk after T_CLK/2;
	rst<='1', '0' after 5*T_CLK;
	--i<='0', '1' after 8*T_CLK, '0' after 9*T_CLK;
	i<='0', '1' after 8*T_CLK, '0' after 30*T_CLK;			--Modification 3
	DUT1: entity work.ROT_1(ARCH_1) port map(clk,rst,i,o_1);
	DUT2: entity work.ROT_2(ARCH_2)
		generic map(NBITS)
		port map(clk,rst,i,o_2);							--Modification 2
end architecture;
--synthesis translate_on
```

4.

TOP.vhd file:

```
architecture ARCH of TOP is
begin
       i0: entity work.ROT_2(ARCH_2)
              generic map(led'length)       --or generic map(4)
              port map(fpga_clk,fpga_rst,switch,led);
end architecture;
```

UCF File:

```
NET fpga_clk TNM_NET = clk_net;
TIMESPEC TS_clk_net = PERIOD clk_net 100 MHz;

NET fpga_clk                  LOC=C10;       #100MHz clock input
NET fpga_rst                  LOC=V4;        #USER PUSH-BUTTON

NET switch                    LOC=B3;        #SWITCH0
NET led<0>                    LOC=P4;        #LED1
NET led<1>                    LOC=L6;        #LED2
NET led<2>                    LOC=F5;        #LED3
NET led<3>                    LOC=C2;        #LED4
```

```
   Found 4-bit register for signal <q>.
   Found 1-bit register for signal <state>
    Summary:
       inferred   5 D-type flip-flop(s).
       inferred   3 Multiplexer(s).

# Registers                                            : 2
 1-bit register                                       : 1
 4-bit register                                       : 1
# Multiplexers                                         : 3
 1-bit 2-to-1 multiplexer                             : 2
 4-bit 2-to-1 multiplexer                             : 1

IO Utilization:
 Number of IOs:                           7
 Number of bonded IOBs:                   7  out of    200     3%

Specific Feature Utilization:
 Number of BUFG/BUFGCTRLs:                1  out of     16
```

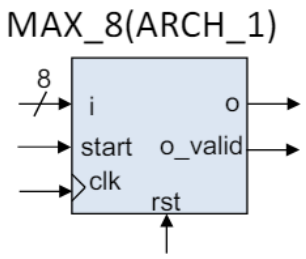
The synthetizer infers 5 FFs. One FF for the state register (a single FF since the state is IDDLE or RUNNING), and a 4-bit register (four FFs) for the q (q=o) (when NBITS=4).

The number of IOB are NBITS+3 = 7 (NBITS due the output:o, 3 due to the inputs: clk, rst and i)

Create a new ISE project (any name and any working directory) for the Spartan-6 of the LX9 Microboard (device: XC6SLX9, package: CSG324, Speed: -3) in order to create and simulate a VHDL design. Copy into the working directory of the project the two VHDL files pack_exam.vhd and max.vhd. Then, add both source VHDL files to the project.

The pack_exam.vhd file declares a VHDL package, named PACK_EXAM, and its package body. This package contains the function F_BIT_MAX which is used in max.vhd.

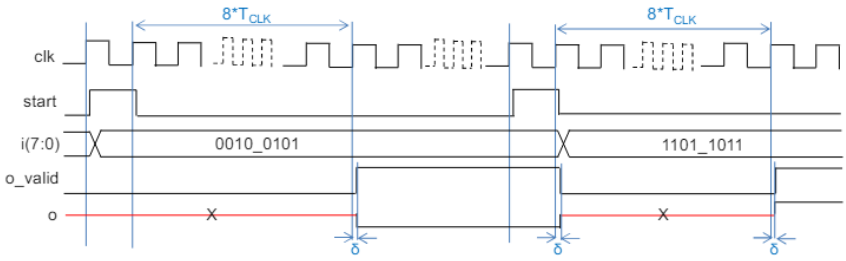
The max.vhd file declares the entity MAX_8 and its architecture ARCH_1. This VHDL module contains a synthesizable description of a synchronous circuit, where the clock and reset ports (clk, rst) are the typical ports used in this kind of circuits. From the 8-bit input port (i), the circuit computes the number of bits that are zeros ('0') and ones ('1') at this input, driving the output port (o) to the bit ('0' or '1') with the highest number. The handshake ports (start, o_valid) are asserted to start the computation and to signal when it is completed, respectively.



For example, when the computation is completed (o_valid='1'):

1. If i="0010_0101", then o='0' since there are 5 zeros ('0') and 3 ones ('1')
2. If i="1101_1011", then o='1' since there are 2 zeros ('0') and 6 ones ('1')
3. If i="1000_1101", then o='X' (any value) since the number of zeros ('0') and ones ('1') are equal

The next Figure shows, with the δ delay, the behaviour of the circuit (after reset) for the first two cases of the previous example:



The process fsm implements a FSM with the state of the circuit, and a counter (idx) which contains the index (from 0 to 7) of the bit at the input to read. The bit of the input (i) at the index

(idx) is assigned to the i_idx signal. The process cnt computes of the number of bits that are zeros ('0') and ones ('1') from the i_idx signal, implemented with two counters (cnt_0 and cnt_1). Finally, when the computation is completed (8 clock cycles after the starting), the circuit asserts the o_valid port and drives the o port from the cnt_0 and cnt_1 through the function F_BIT_MAX.

A testbench (entity TB_MAX, architecture TEST (at the end of the file max.vhd) is also provided to perform the functional simulation.

The exercise consists on creating a new entity (named MAX_N) and its architecture (ARCH_1) which is a modification of the previous ones. Copy MAX_8 and ARCH_1 and change the entity name to MAX_N for the new entity and architecture.

The qualification of the exercise is based on the simplicity of the VHDL description and the efficiency (area and speed) of the implementation for the new architecture.

1-Modify MAX_N and its ARCH_1 to parametrize the number of bits (N_{BITS}) of the output using a VHDL generic. For instance, when N_{BITS}=16 the behaviour is the same as before, but with a larger number of bits at the output:

1. If i="1111_0000_0010_0101", then o='0' since there are 9 zeros ('0') and 7 ones ('1')
2. If i="1111_0000_1101_1011", then o='1' since there are 6 zeros ('0') and 10 ones ('1')
3. If i="1111_0000_1000_1101", then o='X' (any value) since the number of zeros ('0') and ones ('1') are equal

2-Modify the TB_MAX to simulate MAX_N(ARCH_1) for all the 2**N_{BITS} possible combinations in the input (i). The N_{BITS} parameter should be changed (to 4, for instance) in the TB_MAX to check the behaviour of MAX_N(ARCH_1)

3-Create a new architecture ARCH_2 of MAX_N to remove the cnt_1 counter, but the behaviour must remain as in the ARHC_1. Create a new version of the F_BIT_MAX which does not use cnt_1 as input. Modify the TB_MAX to simulate concurrently both architectures at the same time.

4-Add the provided top.vhd and lx9_microboard.ucf files to the project. Complete them in order to implement an instance of ARCH_1 for the ROT_N (parametrized to N_{BITS} =4) and attach it to the FPGA LX9 MicroBoard board. The input (i) is attached to the SWITCHES of the FPGA board, and two of the LEDs as the output (o) and output-valid (o_valid). The start input (start) must be attached to the j4_pin1 (the pin#1 of the J4 PMOD-connector) input port. Synthetize it and report the main synthesis results and justify them. Also report the top.vhd file and the UCF modifications.

Solution:

1. For the N_{BITS} parameter, see modification 1 at the MAX_N and its ARCH_1

```
entity MAX_N is
        generic(
                NBITS: integer:=8 );
        port(
                ...
                i: in std_logic_vector(NBITS-1 downto 0);
                ...
                o_valid: out std_logic );
end entity;

architecture ARCH_1 of MAX_N is
subtype T_CNT is integer range 0 to NBITS-1;
...
begin
        ...

        fsm: process
        begin
                ...
                else case state is
                        when COMPUTING=>
                                if idx=T_CNT'high then
                                        state<=COMPLETED;
                                else
                                        idx<=idx+1;
                                end if;
                ...
        end process;

        ...
end architecture;
```

2. For the simulation of MAX_N(ARCH_1) see modification 2 at TB_MAX

```
architecture TEST of TB_MAX is
constant NBITS: integer:=4;   --It can be changed to any value>=1
...
signal i: std_logic_vector(NBITS-1 downto 0);
...
   DUT1: entity work.MAX_N(ARCH_1) generic map(NBITS)
                                   port map(clk,rst,start,i,o,o_valid);
   process
       begin
               start<='0';
               rst<='X';
               wait for 5*T_CLK;
               rst<='0';

               for j in 0 to 2**NBITS-1 loop
                       wait for 10*T_CLK;
                       start<='1'; i<=std_logic_vector(conv_unsigned(j,NBITS));
                       wait until rising_edge(clk);
                       start<='0';
                       wait until o_valid='1';
               end loop;

               wait;
   end process;
end architecture;
```

3. For the MAX_N(ARCH_2) and its simulation, see Modification 3 at the PACK_EXAM, ARCH_2 of MAX_N and TB_MAX

```
package pack_exam is
        ...
        function F_BIT_MAX_V2(num_bits,num_0: in integer) return std_logic;
end package;

package body pack_exam is
        ...
        function F_BIT_MAX_V2(num_bits,num_0: in integer) return std_logic is
        variable max: std_logic;
        begin
                max:='X';
                if num_0>num_bits/2 then
                        max:='0';
                elsif num_bits/2>num_0 then
                        max:='1';
                end if;
                return max;
        end function;
end package body;

architecture ARCH_2 of MAX_N is
...
signal idx,cnt_0: T_CNT;
...
begin
        o<=F_BIT_MAX_V2(NBITS,cnt_0) when state=COMPLETED else 'X';
        ...
        cnt: process
        begin
                ...
                        when COMPUTING=>
                                case i_idx is
                                        when '0'=> cnt_0<=cnt_0+1;
                                        --when '1'=> cnt_1<=cnt_1+1;
                                        when others=>
                                end case;
                        when others=>
                                if start='1' then
                                        cnt_0<=0;
                                        --cnt_1<=0;
                                end if;
                end case;
        end process;
end architecture;


architecture TEST of TB_MAX_RESULT is
...
signal o2,o2_valid: std_logic;
begin
...

    DUT1: entity work.MAX_N(ARCH_1) generic map(NBITS)
                                    port map(clk,rst,start,i,o,o_valid)
    DUT2: entity work.MAX_N(ARCH_2) generic map(NBITS)
                                    port map(clk,rst,start,i,o2,o2_valid);
...
end architecture;
```

4. Synthesis

TOP.vhd file:

```
architecture ARCH of TOP_RESULT is
...
begin
    ...
    led<=(1=>o, 2=>o_valid, others=>'0');
    start<=j4_pin1;
    i<=switch;
end architecture;
```

UCF File:

```
...
NET j4_pin1 LOC=H12;  #attach to the pin#1 of the j4 pmod-connector
...
```

Synthesis report:

```
Synthesizing Unit <MAX_N>.
    Found 2-bit register for signal <idx>.
    Found 2-bit register for signal <cnt_0>.
    Found 2-bit register for signal <cnt_1>.
    Found 2-bit register for signal <state>.
        ...
    Found 2-bit adder for signal <idx[1]_GND_6_o_add_6_OUT> created at line 39.
    Found 2-bit adder for signal <cnt_0[1]_GND_6_o_add_17_OUT> created at line 55.
    Found 2-bit adder for signal <cnt_1[1]_GND_6_o_add_18_OUT> created at line 56.
    Found 1-bit 4-to-1 multiplexer for signal <i_idx> created at line 27.
    Found 2-bit comparator lessequal for signal <cnt_1[1]_cnt_0[1]_LessThan_2_o> created
at line 15
    Summary:
        inferred   3 Adder/Subtractor(s).
        inferred   6 D-type flip-flop(s).
        inferred   1 Comparator(s).
        inferred   1 Multiplexer(s).
        inferred   1 Finite State Machine(s).
Unit <MAX_N> synthesized.
```

The state of the FSM is stored in a 2-bit register, since there are three possible states

Each of the three counters (idx,cnt_0 and cnt_1) is stored in a 2-bit register, since they can store a number from 0 to 3 (4 counts, therefore 2-bits). The adders of the three counters are also 2-bit wide to perform the addition of the register-q + 1

The multiplexor which implements the i_idx signal is a 4-input multiplexor (1-bit wide), since idx stores from 0 to 3 to select one bit from the 4-bits of the input i

The function F_BIT_MAX is implemented in a 2-bit comparator, since both inputs (cnt_0, cnt_1) are 2-bit wide

```
Slice Logic Utilization:
 Number of Slice Registers:               8  out of  11440     0%
 Number of Slice LUTs:                   11  out of   5720     0%
    Number used as Logic:                11  out of   5720     0%

IO Utilization:
 Number of IOs:                          11
 Number of bonded IOBs:                  11  out of    200     5%

Specific Feature Utilization:
 Number of BUFG/BUFGCTRLs:                1  out of     16     6%
```

The synthesizer infers 8 FFs: 2 FFs for each of the three counters(idx, cnt_0, cnt_1) plus 2 FFs for the FSM (state)

The number of IOBs is 1+1+1+4+4=11 (fpga_clk+fpga_rst+j4_pin1+switch+led)

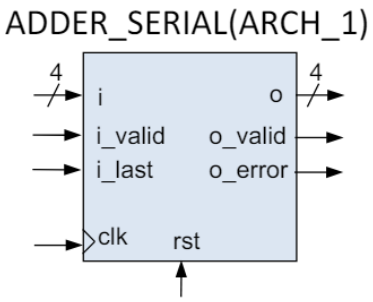
The BUFG is a global buffer dedicated to the clock distribution (clk)

Create a new ISE project (any name and any working directory) for the Spartan-6 of the LX9 Microboard (device: XC6SLX9, package: CSG324, Speed: -3) in order to create and simulate a VHDL design. Copy into the working directory of the project the two VHDL files pack_exam.vhd and adder_serial.vhd. Then, add both source VHDL files to the project.

The pack_exam.vhd file declares a VHDL package, named PACK_EXAM, and its package body. This package contains the functions ADDER_ADD, ADDER_ERROR, which are used in the adder_serial.vhd.

The adder_serial.vhd file declares the entity ADDER_SERIAL and its architecture ARCH_1. This VHDL module contains a synthesizable description of a synchronous circuit, which computes the addition of a set of numbers, serially provided. The input numbers and the result are codified in 4-bit signed data (A2 complement). Therefore, the range of numbers that can be codified is from “1000” to “0111” in binary (-8 to +7 in base 10, respectively)[(1)]. For instance, “0110” codifies the +6 (base 10), and “1010” codifies the -6 (base 10)

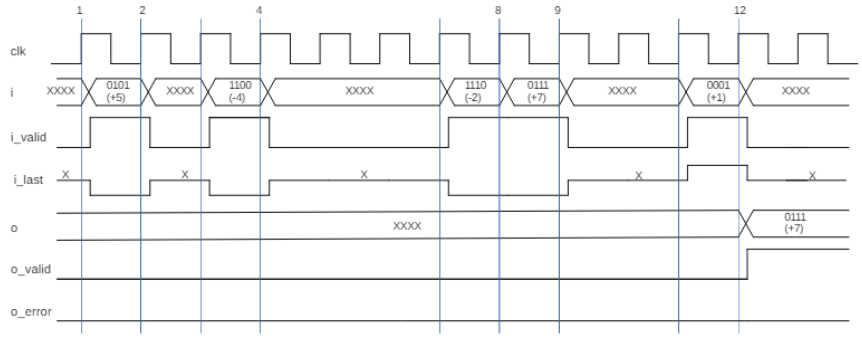
Conversion from negative binary in A2 “1010” to base 10: −(/”1010”+1)=−(“0101”+1)=−(5+1)=−6



The clock and reset ports (clk, rst) are the typical ports used in this kind of synchronous circuits. At the rising edge of the clock, a new number is serially provided in the 4-bit input port (i) when i_valid=’1’. The last number follows the same rule, but also asserting i_last=’1’. Once the last number is provided, the result is available at the 4-bit output port (o) when o_valid=’1’ and o_error=’0’, if the result is correct. However, the circuit drives o_error=’1’ and o_valid=’0’ when the result of any of the additions is out of the valid range[(1)]. The circuit remains stopped when o_valid=’1’ or o_error=’1’.

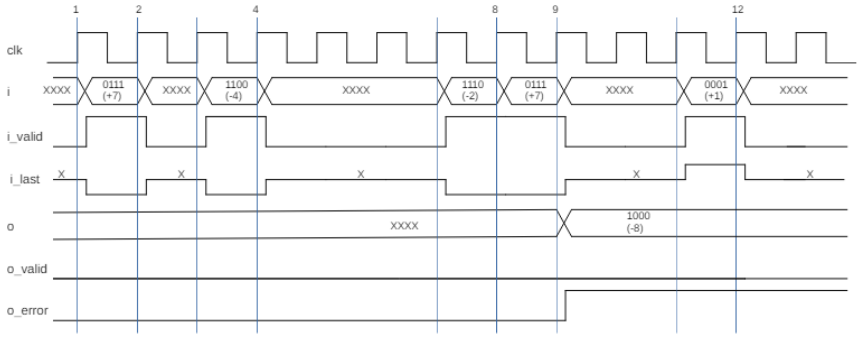
The architecture uses a 5-bit adder (signal adder), which stores the result into the register (signal adder_reg). The additional bit is required to detect if an addition was erroneous. The circuit detects an erroneous addition (signal error=’1’), and stores it in a register (signal error_reg). Finally, the circuit validates the last addition (signal valid=’1’) when it is not erroneous (i_last=’1’ and error=’0’), storing it in another register (signal valid_reg). When the addition is completed, the circuit provides o_valid=’1’ (correct result) or o_error=’1’ (erroneous result), and it remains stopped.

A testbench (at the end of the file adder_serial.vhd) is also provided to perform the functional simulation of the next Figure (δ delays are depicted). After resetting the circuit, the reg_adder="00000" (0 in base 10). The simulation provides five numbers (+5, -4, -2, +7 and +1) when i_valid='1' at rising edges of the clock (2, 4, 8, 9 and 12). The i_last='X' value does not care when i_valid='0', but must be asserted i_last='1' during the last i_valid='1'. The 5-bit reg_adder will store five new values: "00100" (+5), "00001" (+1), "11111" (-1), "00110" (+6), "00111" (+7) from the five additions. Since all the additions are not erroneous, the circuit drives o_valid='1', o_error='0', and o= "0111" (+7) after the last clock edge, and the circuit remains stopped.



The first number provided at the testbench can be easily changed to any other number (constant INIT_VAL). The next Figure repeats the simulation, but starting with "0111" (+7). The 5-bit reg_adder stores four new values: "00111" (+7), "00011" (+3), "00001" (+1) and "01000" (+8), since the last addition is erroneous, the circuit drives o_valid='0', o_error='1' and remains stopped. The last addition is erroneous since only 4-bits from the adder_reg are provided at the output o="1000" ("1000" codifies the negative number -8),

The first number provided at the testbench can be easily changed to any other number (constant INIT_VAL). The next Figure repeats the simulation, but starting with "0111" (+7). The 5-bit reg_adder stores four new values: "00111" (+7), "00011" (+3), "00001" (+1) and "01000" (+8), since the last addition is erroneous, the circuit drives o_valid='0', o_error='1' and remains stopped. The last addition is erroneous since only 4-bits from the adder_reg are provided at the output o="1000" ("1000" codifies the negative number -8),

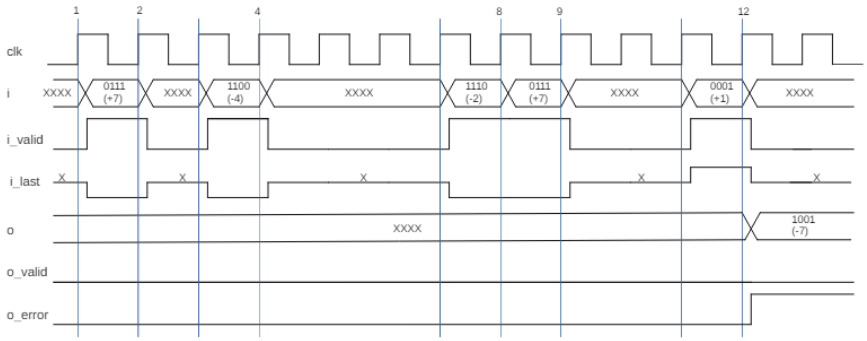


The qualification of the exercise is based on the simplicity of the VHDL description and the efficiency (area and speed) of the implementation for the new architecture.

1-Modify the entity ADDER_SERIAL and its ARCH_1 to parametrize the number of bits (N_{BITS}) of the input (i) and output (o) ports by using a VHDL generic. For instance, when N_{BITS}=6 the valid range of data is from “100000” to “011111” (-32 to +31 in base 10), and any addition out of this range is erroneous.

2-Modify the TB_ADDER_SERIAL to permit the simulation of the new ADDER_SERIAL and ARCH_1 when N_{BITS}=6. One simulation must give a correct result, and the other simulation must give an erroneous result.

3-Create a new architecture ARCH_2, which is very similar to ARCH_1. The difference is the ARCH_2 does not stop when an addition is erroneous. The circuit is stopped when the last addition is computed, asserting either o_valid=’1’ (o contains a valid result) or o_error=’1’ (o contains an erroneous result) at the last addtion. The next Figure depicts a simulation (N_{BITS}=4), which is very similar to the 2nd simulation that was previously explained.



4- Synthesize the ARCH_1 of the ADDER_SERIAL with N_{BITS}=4 with the default synthesis options (-iob=auto), and report the main synthesis results and justify them. Change the synthesis option -iob=yes, and justify the main differences due to this change

Solution:

1. For the N_{BITS} parameter, see modification 1 at the entity ADDER_SERIAL and its ARCH_1

```
entity ADDER_SERIAL is
	generic(
		NBITS: natural:=4 );
	port(
		...
		i: in std_logic_vector(NBITS-1 downto 0);
		...
		o: out std_logic_vector(NBITS-1 downto 0);
		...
end entity;

architecture ARCH_1 of ADDER_SERIAL is
constant INT_NBITS: natural:=NBITS+1;
...
begin
	o<=std_logic_vector(adder_reg(o'range)) when stopped='1' else ...;
	...

	process begin
		wait until rising_edge(clk);
		if rst='1' then
			adder_reg<=(others=>'0');
		...
	end process;
end architecture;
```

2. For the simulation of TB_ADDER_SERIAL with N_{BITS}=6, see modification 2 (it is just a possible solution, the list of serial numbers can be changed).

```
architecture TEST of TB_ADDER_SERIAL is
constant NBITS: integer:=6;
...
begin
        ...
        DUT1: entity work.adder_serial(ARCH_1)
                generic map(NBITS) port map(...);

        process
        ...
        constant INIT_VAL: integer:=+30;
        --Change to +30 to get valid result(+31), -8 for erroneous result (3rd addition)
        begin
                ...
                i<=std_logic_vector(conv_signed(INIT_VAL,NBITS)); ...
                ...
                i<=std_logic_vector(conv_signed(-24,NBITS)); ...
                ...
                i<=std_logic_vector(conv_signed(-12,NBITS)); ...
                ...
                i<=std_logic_vector(conv_signed(+20,NBITS)); ...
                ...
                i<=std_logic_vector(conv_signed(+17,NBITS)); ...
                ...
        end process;
end architecture;
```

In this case, the additions with +30, -24, -12, +20 and +17 will modify the adder_reg to +30, +6, -6, +14, +31. Therefore, all the additions are correct, and the final result is o=+31 ("111111"), o_valid='1' o_error='0'.

Repeating the simulation, but starting with -8 (instead of +30), the adder_reg changes to -8, -32, -44 ("1010100" is out of range). The circuit is stopped after the 3rd addition, with o=+20 ("010100"), o_valid='0', o_error='1'.

3. The ARCH_2 is modified from ARCH_1, according to the modification 3

```
architecture ARCH_2 of ADDER_SERIAL is
...
begin
        ...
        o_valid<=stopped and not error_reg;
        o_error<=stopped and error_reg;

        --stopped<=valid_reg or error_reg;  --removed (replaced as a register)
        i_nxt<=i_valid and not stopped;

        adder<=ADDER_ADD(INT_NBITS, adder_reg,i);
        error<=ADDER_ERROR(INT_NBITS, adder) or error_reg; --propagates the error
        --valid<=i_last and not error;        --removed (unused)

        process begin
                wait until rising_edge(clk);
                if rst='1' then
                        adder_reg<=(others=>'0');
                        error_reg<='0';
                        stopped<='0';
                elsif i_nxt='1' then
                        adder_reg<=adder;
                        error_reg<=error;
                        stopped<=i_last;
                end if;
        end process;
end architecture;
```

4. Synthesis

With default option (-iob=auto)

```
Slice Logic Utilization:

 Number of Slice Registers:                 7  out of  11440     0%

...

IO Utilization:

 Number of IOs:                            14


Specific Feature Utilization:

 Number of BUFG/BUFGCTRLs:                  1  out of     16     6%
```

A single general buffer (BUFG) for the distribution of clk (clock signal) on dedicated routing

The number of IOBs=2 (clk,rst) +6 (i(3:0),i_valid,i_last) +6 (o(3:0),o_valid,o_error) = 14

The numer of FFs (in CLB-Slices)=5(adder_reg(4:0)) +1(valid_reg) +1(error_reg) = 7

All the FFs are placed in CLB-Slices and no FFs from IOBs are used.

Changing the option (-iob=yes)

```
Slice Logic Utilization:

 Number of Slice Registers:                 7  out of  11440     0%

...

IO Utilization:

 Number of IOs:                            14

    IOB Flip Flops/Latches:                 6


Specific Feature Utilization:

 Number of BUFG/BUFGCTRLs:                  1  out of     16     6%
```

The adder_reg(3:0), error_reg, valid_reg are now packed into the 6 FFs from IOBs, since they are output ports. However, the adder_reg(3:0), error_reg, valid_reg are replicated into internal FFs from CLB-Slices, since all of them are internally required (they are read). There is an additional internal FF, which stores the bit adder_reg(4).

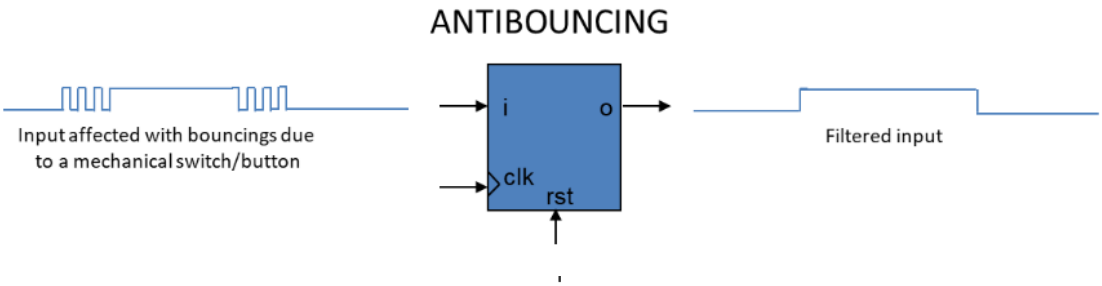
Number of FFs (in IOB)=6 (FFs adder_reg(3:0), error_reg, valid_reg)

Numer of FFs (in CLB-Slices)=7 (the replication of the previous 6 and the adder_reg(4))

Create a new ISE project (any name and any working directory) for the Spartan-6 of the LX9 Microboard (device: XC6SLX9, package: CSG324, Speed: -3) in order to create and simulate a VHDL design. Copy into the working directory of the project two VHDL files: pack_exam.vhd and antibouncing.vhd. Then, add both source VHDL files to the project.

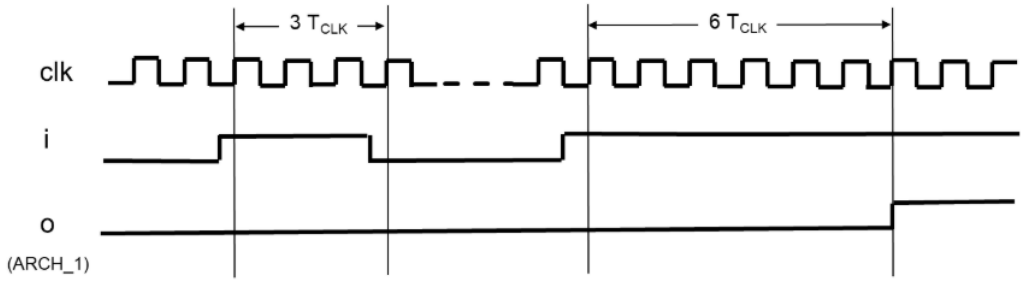
The pack_exam.vhd file declares a VHDL package, named PACK_EXAM, and its package body. It contains the function F_NBITS which returns the number of bits required for a number of counts, which is used in antibouncing.vhd.

The antibouncing.vhd contains a VHDL model of an anti-bouncing circuit, which filters bouncings at the FPGA inputs that are connected to mechanical switches or buttons. The output (o) of the circuit changes to the input (i) value when the input is stable (does not change) during period of time greater than the configured stabilization time. Otherwise, the output (o) does not change.



In the provided circuit, the stabilization time is configured to 6 clock cycles (T_{CLK}) (COUNTS =6="110" in binary). The clock (clk) and reset (rst) input ports are the usually available in any synchronous circuit. The VHDL contains two processes, p1 devoted to edge detections at the input, and p2 to describe a FSM and a counter.

A testbench (entity TB_ANTIBOUNCING, architecture TEST) is provided to perform the functional simulation, at the end of the antibouncing.vhd file. The next Figure shows the behaviour of the circuit. The ouput (o) does not change to the value of the input (i) if the input is not stable at least 6*T_{CLK} (COUNTS=6).



The qualification of the exercise is based on the simplicity of the VHDL description and the efficiency (area and speed) of the implementation for the proposed modifications.

1-The entity is parametrized by including the integer parameter COUNTS as a VHDL generic. Modify the ARCH_1 in order to behaves as expected for any positive value of the COUNTS parameter. Also, modify the testbench to check the simulation for COUNTS 6,10 or 16

```
entity antibouncing is
      generic(
            COUNTS: integer:=6 );
      port(
            clk, rst: in std_logic;
            i: in std_logic;
            o: out std_logic );
end entity;
```

2-Check the cases when COUNTS is a power of two (for instance COUNTS=8 or COUNTS=16). Modify the ARCH_1 if the circuit is not working as expected also for these cases.

3-Modify architecture ARCH_1 to remove signals state and cnt and declare them as variables of the p2 process. The behaviour of the modified architecture must be exactly the same as before.

```
--signal state: FSM;    --remove the state and cnt signals
--signal cnt: COUNTER;  --from the architecture
begin
      ...
      p2: process(clk,rst)
      variable state: FSM;          --state and cnt are variables of
      variable cnt: COUNTER;        --of the p2 process
      begin
            ...
      end process;
```

3-Modify architecture ARCH_1 to remove signals state and cnt and declare them as variables of the p2 process. The behaviour of the modified architecture must be exactly the same as before.

```
--signal state: FSM;    --remove the state and cnt signals
--signal cnt: COUNTER;  --from the architecture
begin
      ...
      p2: process(clk,rst)
      variable state: FSM;          --state and cnt are variables of
      variable cnt: COUNTER;        --of the p2 process
      begin
            ...
      end process;
```

4-Copy and add to the project the files top.vhd and lx9_microboard.ucf, and complete them to connect the 4 (DIP) SWITCHES to the 4 LEDS of the LX9 MicroBoard through ANTIBOUNCING circuits. The parameter ANTIBOUNCING_COUNTS must be configured to set the stabilization time to 10 ms (milliseconds). Report the modifications in the top.vhd and the UCF files. Then, synthesize the TOP circuit to report the number of required IOBs and FFs, and justify these results.

Solution:

1. The modifications of the arch_1

```
architecture arch_1 of antibouncing is
...
subtype COUNTER is unsigned(F_NBITS(COUNTS)-1 downto 0);
signal cnt: COUNTER;
--constant COUNTS: COUNTER:="110";        --removed

p2: process(clk,rst)
begin
    ...
    when IDLE=>
           if edge='1' then
                  state<=RUN; cnt<=conv_unsigned(COUNTS,COUNTER'length);
    when RUN=>
           ...
           elsif edge='1' then
                  cnt<=conv_unsigned(COUNTS,COUNTER'length);
```

The modifications of the testbench

```
architecture test of tb_antibouncing is
constant ANTIBOUNCING_COUNTS: natural:=16;       --change to 6,10,16
...
dut: entity work.antibouncing(arch_1) generic map(ANTIBOUNCING_COUNTS) port map(...)
```

2. The previous architecture will not work for COUNTS=8 (or any power of two value), since cnt is declared as a 3-bit register due to F_NBITS(8)=3. The counter register starts from COUNTS=8 (“1000” does not fit in a 3-bit register), and cnt starts from value 0 (instead of 8)

```
cnt<=conv_unsigned(COUNTS,COUNTER'length)=conv_unsigned(8,3)=”000”=0
```

In order to start at COUNTS=8 (“1000” does not fit in a 3-bit register), the number of bits must be F_NBITS(8+1)=4. The counter is checked with 1 (“0001”) to change the state to IDLE.

```
subtype COUNTER is unsigned(F_NBITS(COUNTS+1)-1 downto 0);
```

An alternative solution is to start from COUNTS-1 and compare cnt=0 to change the state to IDLE, without increasing the number of bits NBITS(8)=3

```
subtype COUNTER is unsigned(F_NBITS(COUNTS)-1 downto 0);

   ...
   when IDLE=>
          if edge='1' then
                 state<=RUN; cnt<=conv_unsigned(COUNTS-1,COUNTER'length);
          end if;
   when RUN=>
          cnt<=cnt-1;
          if cnt=0 then
                 state<=IDLE; o<=i;
          elsif edge='1' then
                 cnt<=conv_unsigned(COUNTS-1,COUNTER'length);
          end if;
```

3. The assignation operator for variables is := and it is instantaneous (no delay). Therefore, cnt is compared with 0 (instead of 1)

```
p2: process(clk,rst)
variable state: FSM;
variable cnt: COUNTER;
begin
        if rst='1' then
                state:=IDLE; o<='0';
        elsif rising_edge(clk) then case state is
                when IDLE=>
                        if edge='1' then
                                state:=RUN; cnt:=conv_unsigned(...);
                        end if;
                when RUN=>
                        cnt:=cnt-1;
                        if cnt=0 then   --or if cnt<=0 then
                                state:=IDLE; o<=i;
                        elsif edge='1' then
                                cnt:=conv_unsigned(...);
                        end if;
        end case; end if;
end process;
```

Another solution is to perform the subtraction of cnt, after comparing it with number 1

```
                ...
                when RUN=>
                        if cnt<=1 then
                                state:=IDLE; o<=i;
                        elsif edge='1' then
                                cnt:=conv_unsigned(...);
                        end if;
                        cnt:=cnt-1;
```

4. Architecture ARCH of TOP:

```
architecture ARCH of TOP is
constant ANTIBOUNCING_COUNTS: integer:=1e6;  --(100*10^6)Hz*(10*10^-3)s=1*10^6
counts
begin

    g0: for j in 0 to 3 generate
            u0: entity work.antibouncing(arch_1a)
                    generic map(ANTIBOUNCING_COUNTS)
                    port map( fpga_clk, fpga_rst, switch(j), led(j) );
    end generate;

end architecture;
```

El fichero UCF:

```
NET switch<0> LOC=B3;
NET switch<1> LOC=A3;
NET switch<2> LOC=B4;
NET switch<3> LOC=A4;

NET led<0> LOC=P4;
NET led<1> LOC=L6;
NET led<2> LOC=F5;
NET led<3> LOC=C2;
```

IOBs=2+4+4 (clk,rst,4 switches, 4 leds)

GBUFF=1 the clk (global buffer)

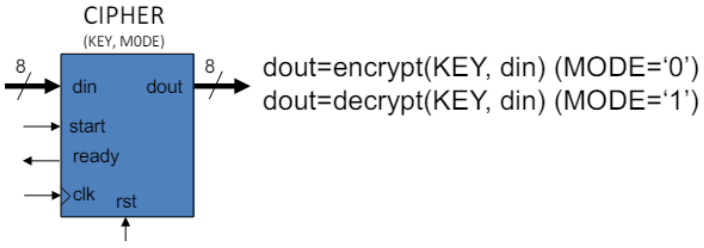
FFs=92=4*23. Each debouncing circuit requires 20-bit register for the cnt (1+10^6 counts), 1-bit for the state FSM (2 states), 1-bit for the i_reg register and 1-bit for the o register.

Create a new ISE project (any name and any working directory) for the Spartan-6 of the LX9 Microboard (device: XC6SLX9, package: CSG324, Speed: -3) in order to create and simulate a VHDL design. Copy into the working directory of the project two VHDL files: pack_exam.vhd and cipher.vhd. Then, add both source VHDL files to the project.

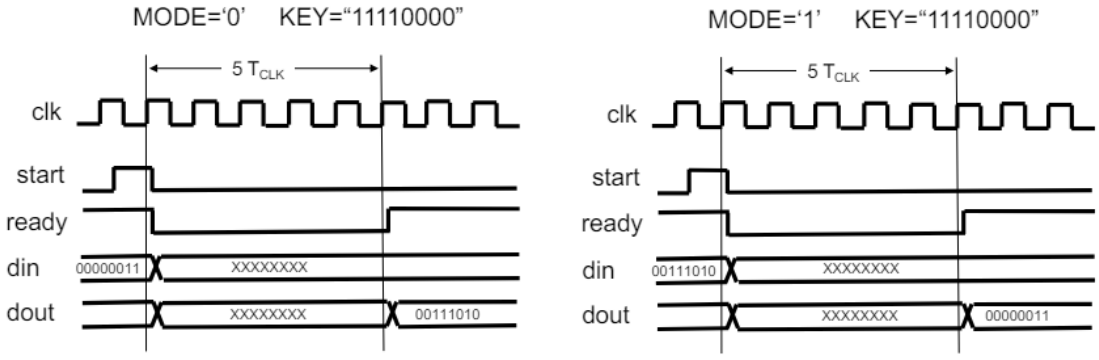
The pack_exam.vhd file declares a VHDL package, named PACK_EXAM, and its package body. It contains:

- The declaration of the constant ROUNDS as 5
- The declaration of the type T_DATA as the 8-bit std_logic_vector
- The encryption F_ENCRYPT and decryption F_DECRYPT functions

The architecture a1 of the cipher can encrypt (MODE='0') or decrypt data (MODE='1') by using a parametrizable KEY. The encryption/decryption starts by asserting start='1'. The circuit starts loading a register (q) with the input data port (din) and driving ready='0'. While encrypting, the register (q) is left-rotated and combined with the KEY (by using XOR gates) at each clock cycle, during several ROUNDS. The decryption is similar but reversing the order. After completing the encryption/decryption, the circuit drives ready='1' and the output data port (dout) contains the result.



A testbench (TB_CIPHER) is provided at the end of the cipher.vhd file to perform the functional simulation of several encryption/decryption of a number. The next Figure shows the behaviour of the circuit for one of simulated encryption-decryption. For the tested KEY="11110000", the encrypted dout="00111010" when din="00000011" (*), and the decrypted dout="0000011" (retrieves the same code *)



The qualification of the exercise is based on the simplicity of the VHDL description and the efficiency (area and speed) of the implementation for the proposed modifications.

1-Modify T_DATA to declare it as a 16-bit, 32-bit or 64-bit std_logic_vector and check encryption/decryption does not work as expected. Modify the pack_exam to properly work for an any arbitrary number of bits for T_DATA.

```
subtype T_DATA is std_logic_vector(31 downto 0); --or any other arbitrary number of bits
```

2-Create a new architure a2 by removing the variable cnt and adding the signal cnt. Modify a2 to behave the same as the original architecture a1.

```
signal cnt: integer;          --remove the variable cnt: integer in the p_fsm process
```

3-The architecture a1 (or a2) works correctly if started (by asserting start='1') while ready='1' (*), but it provides erroneous encryption/decryption if it is started while ready='0' (**). Modify the testbench (see the following code) to check the incorrect behaviour when the condition (**) is accomplished. Then, create a new architecture a3 that ensures the circuit starts the encryption/decryption only with the condition (*), ignoring the starting at the wrong condition (**)

```
--replace these two lines of the testbench
start_e<='1'; din_e<=data; wait until rising_edge(clk);
start_e<='0'; din_e<=(others=>'X'); wait until ready_e='1';

--with the following four lines, to test the condition (*) and (**)
start_e<='1'; din_e<=data; wait until rising_edge(clk);     --Condition (*)
start_e<='0'; wait until rising_edge(clk); wait until rising_edge(clk);
start_e<='1'; wait until rising_edge(clk);                  --Condition (**)
start_e<='0'; din_e<=(others=>'X'); wait until ready_e='1';
```

4-Copy and add to the project the files top.vhd and lx9_microboard.ucf, and complete them to synthetize and implement an 8-bit cipher with the parameters MODE='0', KEY="11010001". Report the modifications done in the top.vhd and the UCF files. Then, synthesize the TOP circuit to report the number of required IOBs and FFs, and justify these results, specially focussing in the cnt counter. Propose a simple solution to reduce the number of resources devoted the this counter, and check by synthesis the reduction of resources.

1. The modifications on PACK_EXAM:

```
subtype T_DATA is std_logic_vector(31 downto 0); --(15 downto 0),(63 downto 0)...

function F_ENCRYPT(key: T_DATA; reg: T_DATA) return T_DATA is
...
begin
        shft:=reg(T_DATA'left-1 downto T_DATA'right) & reg(T_DATA'left);
        ...
end function;

function F_DECRYPT(key: T_DATA; reg: T_DATA) return T_DATA is
...
begin
        ...
        shft:=mix(T_DATA'right) & mix(T_DATA'left downto T_DATA'right+1);
        ...
end function;
```

2. When replacing the variable cnt to be a signal, there are several solutions. For instance:

```
p_fsm: process
begin
        wait until rising_edge(clk);
        if rst='1' then
                cnt<=0;
        elsif start='1' or state=RUNNING then
                cnt<=cnt+1;
                if cnt+1>ROUNDS then  --or cnt>=ROUNDS or cnt>ROUNDS-1
                        cnt<=0;
                end if;
        end if;
end process;

state<=IDLE when cnt=0 else RUNNING;
```

Another possible solution:

```
p_fsm: process
begin
        wait until rising_edge(clk);
        if rst='1' then
                cnt<=0; state<=IDLE;
        elsif start='1' or state=RUNNING then
                cnt<=cnt+1; state<=RUNNING;
                if cnt+1>ROUNDS then  --or cnt>=ROUNDS or cnt>ROUNDS-1
                        cnt<=0; state<=IDLE;
                end if;
        end if;
end process;
```

3. For instance, adding a new signal start_ready='1' when start='1' and state=IDLE (equivalent to ready='1'), and using start_ready='1' to start the encryption/decryption (instead of start='1')

```
signal start_ready: std_logic;
begin
        ...
        start_ready<=start when state=IDLE else '0';

        p_fsm: process
                ...
                elsif start_ready='1' or state=RUNNING then
                        cnt<=cnt+1;
                ...
```

```
        end process;

        p_q: process
                ...
                elsif start_ready='1' then
                        q<=din;
                ...
        end process;
...
```

4. Architecture ARCH of TOP:

```
architecture arch of top is
constant KEY: std_logic_vector(7 downto 0):="11010001";
begin

    u0: entity work.cipher(a1) generic map(KEY,'0')
            port map(fpga_clk,fpga_rst,switch,led,in_8bit,out_8bit);

end architecture;
```

The UCF file:

```
NET switch LOC=B3;
NET led LOC=P4;
```

The synthesis reports:

IOBs=1+1+1+1+8+8=20 (clk+rst+switch+led+in_8bit+out_8bit)

GBUFF=1 the clk (global buffer)

FFs=32+8+1=41. The 32-bit counter (cnt), the 8-bit register (q) and the 1-bit register (state)

Additionally, since cnt is a 32-bit register, it implements by LUTs a 32-bit adder, a 32-bit multiplexor and a 32-bit comparator for the conditional assignment of cnt

The total number of LUTs is 88

In order to reduce the resources devoted to cnt

```
variable cnt: integer range 0 to ROUNDS;
```

The synthesis reports the following main differences:

FFs=3+8+1=12. The cnt is reduced from 32-bit to 3-bit counter (since 3-bits are required to count from 0 to 5)

Additionally the LUTs that implement the 3-bit adder, 3-bit multiplexor and 3-bit comparator are reduced

The total number of LUTs is reduced to13