Specifications and Simulations Of Digital Systems

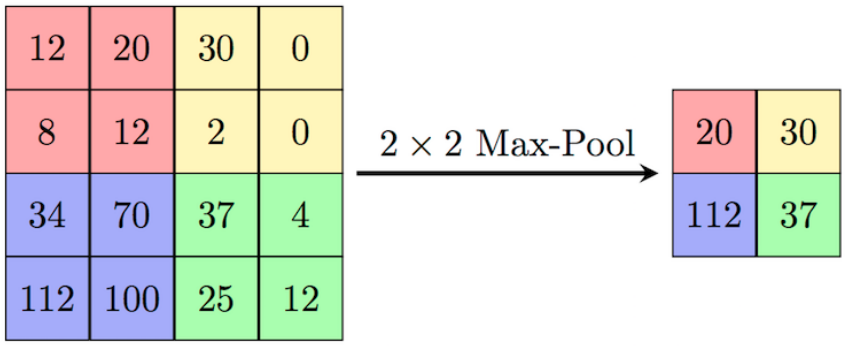
Assignment Report: Dalmasso Luca s281316

INTRODUCTION: Max Pooling algorithm.

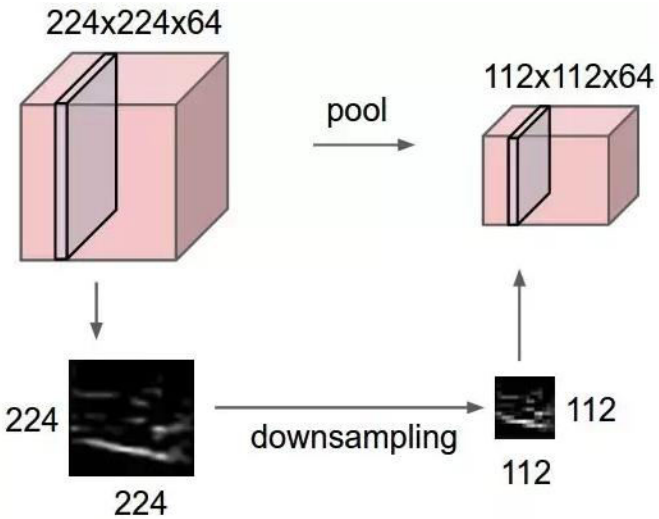
The goal of this assignment was to design a VHDL module capable of doing a Max-Pooling algorithm.

Max pooling is a sample-based discretization process. The objective is to down-sample an input representation (image, hidden-layer output matrix, etc.), reducing its dimensionality and allowing for assumptions to be made about features contained in the sub-regions.

Graphical representation of the algorithm on two dimensions:



Real life example: image



Algorithm in C:

This piece of code is executed with the purpose to generate a memory file used by the testbench in order to test the HLSM output stream.

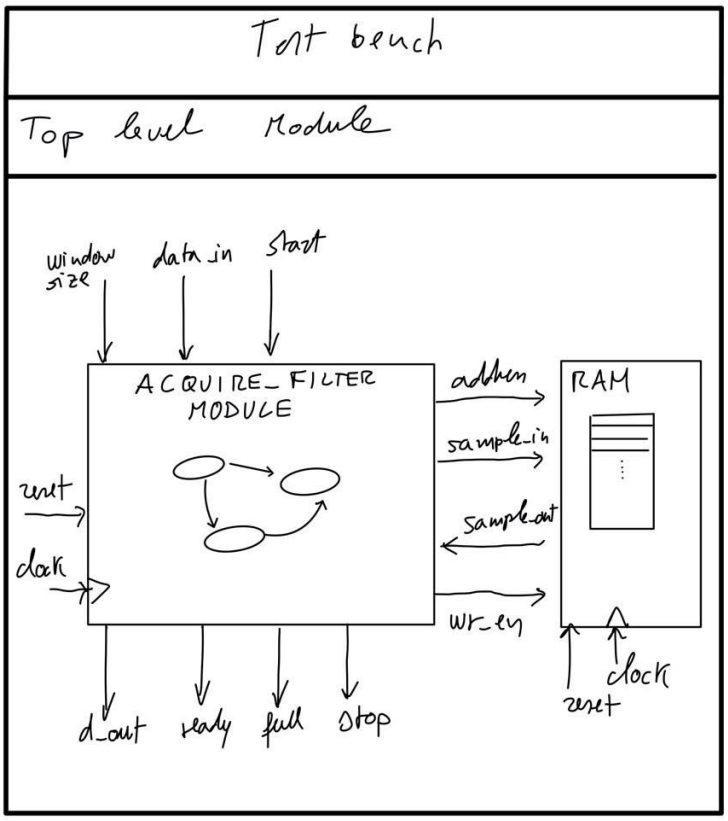
The Max-pooling algorithm is performed by 4 nested loops that basically are looking for the maximum value inside every MxM non overlapped submatrices of the source NxN matrix, where M is the pooling-window's width.

```
void matrix_pooling(int matrix[][N])
{
    int i,j,k,p,max;
    FILE *fp;
    fp=fopen(FILE_DEST,"w");
    if(fp==NULL)
    {
        fprintf(stderr,"no destination file found!\n");
        exit(-1);
    }
    for(i=0;i<N;i=i+M){
        for(j=0;j<N;j=j+M){
            max=matrix[i][j];
            for(k=i;k<i+M;k++){
                for(p=j;p<j+M;p++){
                    if(k==i && p==j)
                        continue;
                    if(matrix[k][p] > max)
                        max=matrix[k][p];
                }
            }
            fprintf(fp,"%d\n",max);
        }
    }
    fclose(fp);
}
```

ARCHITECTURE

I designed 3 main components in order to compute the algorithm.

Acquisition system, Memory, Top-Level module



* signals' names are different from those used in VHDL.

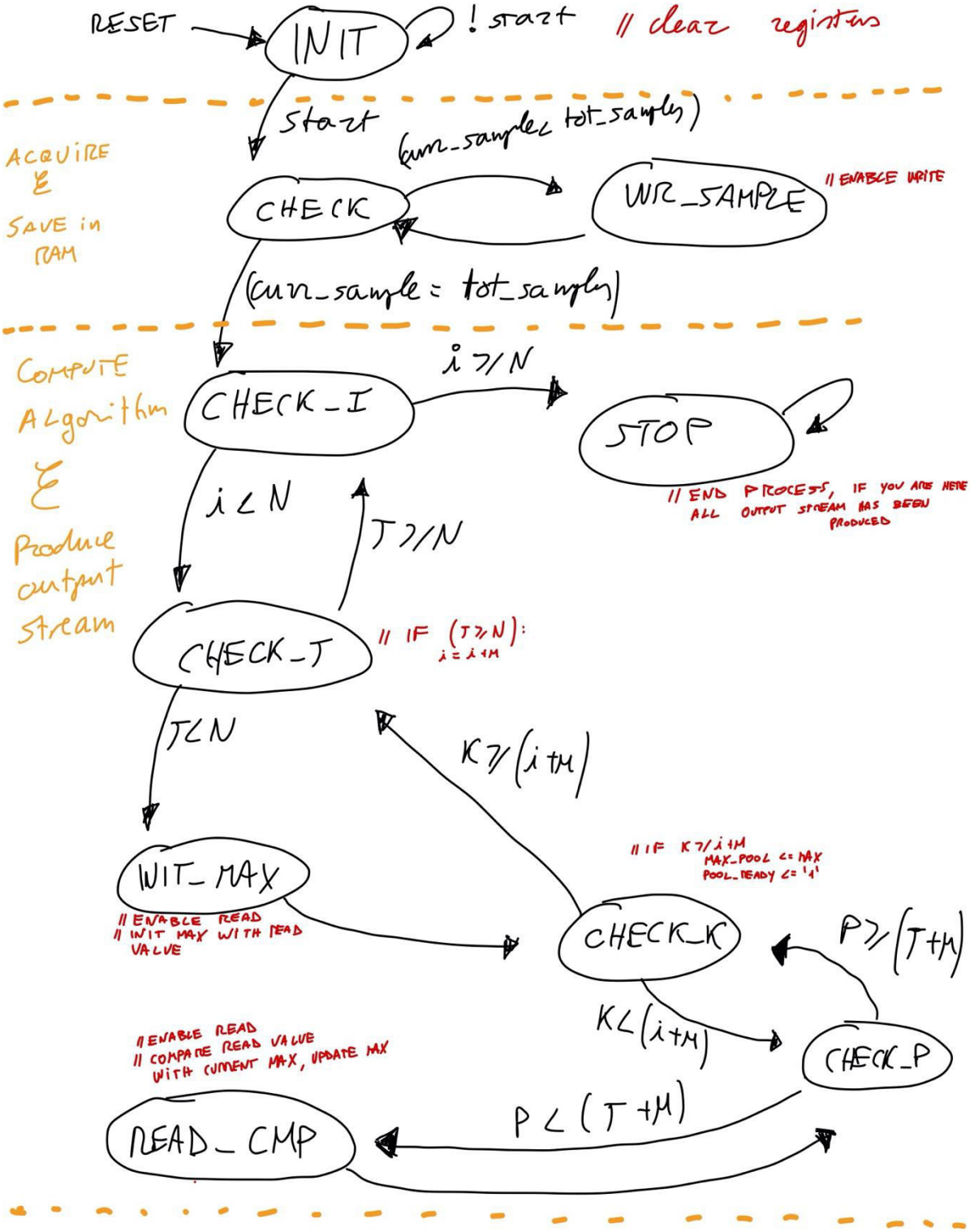
1. **<u>Acquisition system</u>**:

   This module is able to capture a stream of input samples, coming from a Top-Level module, and send them to external memory (RAM type).

   Once all samples have been saved, the Max Pooling algorithm is performed, and a new output data stream, containing sub-region max, is generated.

Once every sub-region max has been computed and streamed, this module goes in an idle state where waits for an external reset;

This module is designed to be a HLSM, the following schema is showing the HLSM states that I designed:



The following is the VHDL code describing the entity acquisition system and the HLSM:

Entity:

```
entity acq_system_HLSM is
    generic(
        --address's bits
        addr_l: natural:=4;
        --data's bits
        data_l: natural:=16;
        --number of samples to be taken (must be <=2^addr_l)
        n_samples: natural:=16;
        --size of sqare matrix containig our samples
        nrows: natural:=40
    );
    port(
        clock:  in std_logic;
        reset:  in std_logic;
        start:  in std_logic;
        data_in:in std_logic_vector(data_l - 1 downto 0); --sample in
        data_from_memory: in std_logic_vector(data_l - 1 downto 0); --sample read from memory
        m_x_m: in std_logic_vector(3 downto 0); --size of max pooling window
        addr_out:out std_logic_vector(addr_l-1 downto 0); --RAM's address where to write
        data_out: out std_logic_vector(data_l-1 downto 0); --sample to be written in RAM
        max_pool_sample: out std_logic_vector(data_l-1 downto 0); --result of current max-pooling
        pool_ready: out std_logic; --if = '1' --> result of current pooling is ready
        end_process: out std_logic; --every pooling on all sub-matrices has been done = end of process
        full: out std_logic; --write operation on RAM ended = all samples in have been captured and saved
        wr_en: out std_logic --write/read signal for RAM
    );
end acq_system_HLSM;
```

HLSM internal registers and signals:

```
--hlsm states register
type statetype is (INIT,CHECK,WR_SAMPLE,CHECK_I,CHECK_J,INIT_MAX,CHECK_K,CHECK_P,READ_SUB_VAL,STOP);
signal next_state,current_state: statetype;

--hlsm index counter
signal next_index,current_index: std_logic_vector(addr_l downto 0);

--i,j,k,p index registers
signal next_i,current_i: std_logic_vector(addr_l-1 downto 0);
signal next_j,current_j: std_logic_vector(addr_l-1 downto 0);
signal next_k,current_k: std_logic_vector(addr_l-1 downto 0);
signal next_p,current_p: std_logic_vector(addr_l-1 downto 0);

--max register
signal next_max,current_max: std_logic_vector(data_l-1 downto 0);
```

Sample and acquire stages:

```
case current_state is
    when INIT=>
        --registers
        next_index<=(others=>'0');
        next_i<=(others=>'0');
        next_j<=(others=>'0');
        next_k<=(others=>'0');
        next_p<=(others=>'0');
        next_max<=(others=>'0');
        full<='0';
        if start='1' then
            next_state<=CHECK;
        else
            next_state<=INIT;
        end if;

   when CHECK=>
        --have all samples been saved?
        full<='0';
        if to_integer(unsigned(current_index)) = (n_samples) then
            next_state<=CHECK_I;
            next_index<=(others=>'0');
        else
            next_state<=WR_SAMPLE;
        end if;

   when WR_SAMPLE=>
        --on next clock cycle the data is written into memory
        next_index<=std_logic_vector(unsigned(current_index) + 1);
        wr_en<='1';
        data_out<=data_in;
        next_state<=CHECK;
        full<='0';
```

Max-Pooling algorithm: subregions, max initialization.

```
when CHECK_I=>
     if to_integer(unsigned(current_i)) = (nrows) then
         next_state<=STOP;
     else
         next_state<=CHECK_J;
     end if;
     next_j<=(others=>'0');

when CHECK_J=>
     if to_integer(unsigned(current_j)) = (nrows) then
         next_state<=CHECK_I;
         next_i<=std_logic_vector(unsigned(current_i)+unsigned(m_x_m));
     else
         next_state<=INIT_MAX;
     end if;

when INIT_MAX=>
     --read RAM for max init, i use an auxiliary variable to traslate
     --a c,c++ like matrix (m[i][j]) into accesses to memory (array)
     --for example in a 2x2 matrix if i want to access element m[1][1] i just need i=1,j=1
     --i and j are translated into a single addres that is: addr=(i*2 + j);
     address_memory:=to_integer(unsigned(current_i)) * nrows;
     addr_out<=std_logic_vector(to_unsigned(address_memory,addr_out'length) + unsigned(current_j));
     wr_en<='0';
     next_k<=current_i;
     next_state<=CHECK_K;
     next_max<=data_from_memory;
```

Max-Pooling algorithm: max research.

```
when CHECK_K=>
     if (unsigned(current_k) = (unsigned(current_i)+unsigned(m_x_m))) then
         max_pool_sample<=current_max;
         next_j<=std_logic_vector(unsigned(current_j)+unsigned(m_x_m));
         next_state<=CHECK_J;
         pool_ready<='1';
     else
         next_state<=CHECK_P;
         next_p<=current_j;
     end if;

when CHECK_P=>
     if (unsigned(current_p) = (unsigned(current_j)+unsigned(m_x_m))) then
         next_k<=std_logic_vector(unsigned(current_k)+1);
         next_state<=CHECK_K;
     else
         next_state<=READ_SUB_VAL;
     end if;

when READ_SUB_VAL=>
     --read a value in submatrix and compare it to the current max value of same submatrix
     address_memory:=to_integer(unsigned(current_k)) * nrows;
     addr_out<=std_logic_vector(to_unsigned(address_memory,addr_out'length) + unsigned(current_p));
     wr_en<='0';
     next_p<=std_logic_vector(unsigned(current_p)+1);
     next_state<=CHECK_P;
     if  to_integer(unsigned(data_from_memory)) > to_integer(unsigned(current_max)) then
         next_max<=data_from_memory;
     end if;

when STOP=>
     --restart everything from scratch only with 'reset' signal
     end_process<='1';
     next_state<=STOP;
```

2. **RAM storage data type**:

   Due to the fact that I have at first a ‘sample’ phase, and then a ‘computation’ stage, I need to memorize all the input stream in a storage.

   Entity:

```
entity RAM is
    generic(
        addr: natural:=4;--address' bits
        word: natural:=16;--word's bits
        size: natural:=16;--size of memory, must be <=(2^addr -1)
        t_acc: time:= 5 ns
    );
    port(
        clock:in std_logic;
        reset:in std_logic;
        data_in:in std_logic_vector(word-1 downto 0);
        addr_in:in std_logic_vector(addr-1 downto 0);
        wr_en:in std_logic;
        data_out:out std_logic_vector(word-1 downto 0)
    );
end RAM;
```

Read & Write processes:

```
--word subtype
subtype WORDT is std_logic_vector(word-1 downto 0);
--storage type
type STORAGE is array(0 to size-1) of WORDT;
--memory
signal MEM: STORAGE;

begin

    --write process
    process(clock)
    begin
        if rising_edge(clock) then
            if reset='1' then
                MEM<=(others=>(others=>'0'));
            else
                if wr_en='1' then
                    MEM(to_integer(unsigned(addr_in)))<=data_in;
                end if;
            end if;
        end if;
    end process;

    --read process
    process(addr_in,wr_en,MEM)
    begin
        if wr_en='0' then
            data_out<=MEM(to_integer(unsigned(addr_in))) after t_acc;
        else
            data_out<=(others=>'Z');
        end if;
    end process;

end Behavioral;
```

3. **Top-Level:**

   VHDL module that includes in a structural way the two previous modules.

   This module is only one accessed externally, in fact is the Unit Under Test in the TestBench.

```
entity top_level is
    port(
        sample_in: in std_logic_vector(15 downto 0);
        clock,reset,start: in std_logic;
        select_window: in std_logic_vector(2 downto 0);
        pool_output: out std_logic_vector(15 downto 0);
        pooling_end: out std_logic;
        pool_is_ready: out std_logic
    );
end top_level;
```

```
system: acq_system_HLSM generic map(4,16,16,4) port map(clock,
                                                        reset,
                                                        start,
                                                        sample_in,
                                                        data_out_mem,
                                                        m,
                                                        addr_out_acq,
                                                        data_out_acq,
                                                        pool_out,
                                                        ready,
                                                        max_pooling_end,
                                                        end_save,
                                                        wr_enable);

memory: RAM generic map(4,16,16,5 ns) port map(clock,
                                               reset,
                                               data_out_acq,
                                               addr_out_acq,
                                               wr_enable,
                                               data_out_mem);

--only predefined pooling windows are allowed

with select_window select m<="0001" when "001",
                              "0010" when "010",
                              "0100" when "100",
                              "1000" when "101",
                              "1010" when others;

pool_output<=pool_out;
pooling_end<=max_pooling_end;
pool_is_ready<=ready;
```

TESTBENCH, test of the proposed solution, waveforms analisys

Unit Under Test: Top-Level module:

```
component top_level is
    port(
        sample_in: in std_logic_vector(15 downto 0);
        clock,reset,start: in std_logic;
        select_window: in std_logic_vector(2 downto 0);
        pool_output: out std_logic_vector(15 downto 0);
        pooling_end: out std_logic;
        pool_is_ready: out std_logic
    );
end component;

signal clk,rst,start: std_logic;
signal data_in: std_logic_vector(15 downto 0);
signal sel: std_logic_vector(2 downto 0);
signal ready: std_logic;
signal pool_output:std_logic_vector(15 downto 0);
signal stop:std_logic ;

signal test: std_logic_vector(15 downto 0);
constant period:time:= 20 ns;

file VECTOR_SAMPLE : text;
file VECTOR_VV: text;

begin

uut:top_level port map(data_in,
                       clk,
                       rst,
                       start,
                       sel,
                       pool_output,
                       stop,ready);
```

Action Performed:

- Generation of the input data stream, selection of the pooling window:

  The input data stream is generated by reading a memory file called ‘samples.m’ , previously created by a C piece of code that randomly generates NxN values and writes them the file.

This is the testbench process that perform this operation:

```
--INPUT STREAM GENERATION PROCESS
process
variable v_ILINE : line;
variable v_TERM: integer;
begin
    file_open(VECTOR_SAMPLE, "samples.mem",  read_mode);
    rst<='1';
    wait for period/2;
    wait for 5 ns;
    rst<='0';
    sel<="010";
    start<='1';
    wait for period;
    wait for period;
    while not endfile(VECTOR_SAMPLE) loop
        readline(VECTOR_SAMPLE, v_ILINE);
        read(v_ILINE, v_TERM);
        data_in<=std_logic_vector(to_unsigned(v_TERM,data_in'length));
        wait for period;
        wait for period;
    end loop;
    file_close(VECTOR_SAMPLE);
    wait;
end process;
```

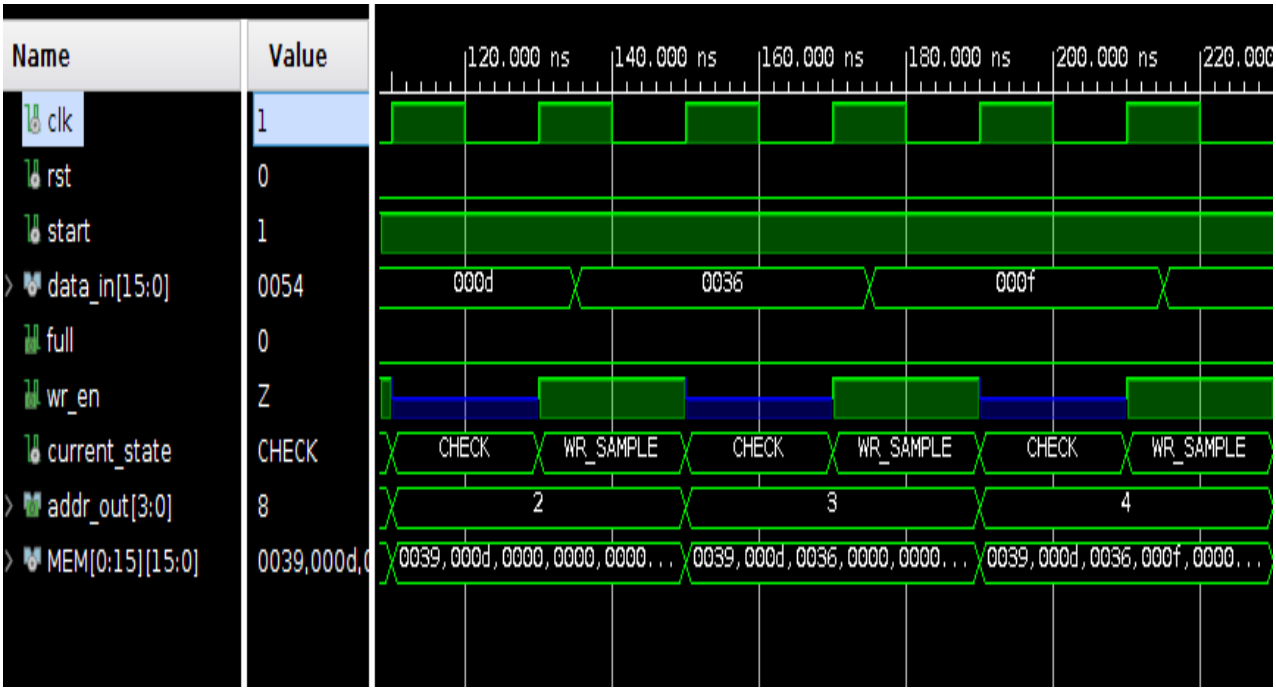
Note:

During the input stage the HLSM is locked in CHECK and WR_SAMPLE stages so we need to wait for 2 clock cycles between an input and another.

Folliwing image shows a waveforms capture of some important signals, such as:

Data_in: input sample coming from testbench

Wr_enable: HLSM output control signal for the RAM



- Test of the results

  Verification and validation of correctness of the results is performed by a process that save output data stream generated by HLSM (every data is the current max of a subregion) and compare it with a correct result saved into another memory file.
  Memory file is written by the C function discussed previously.

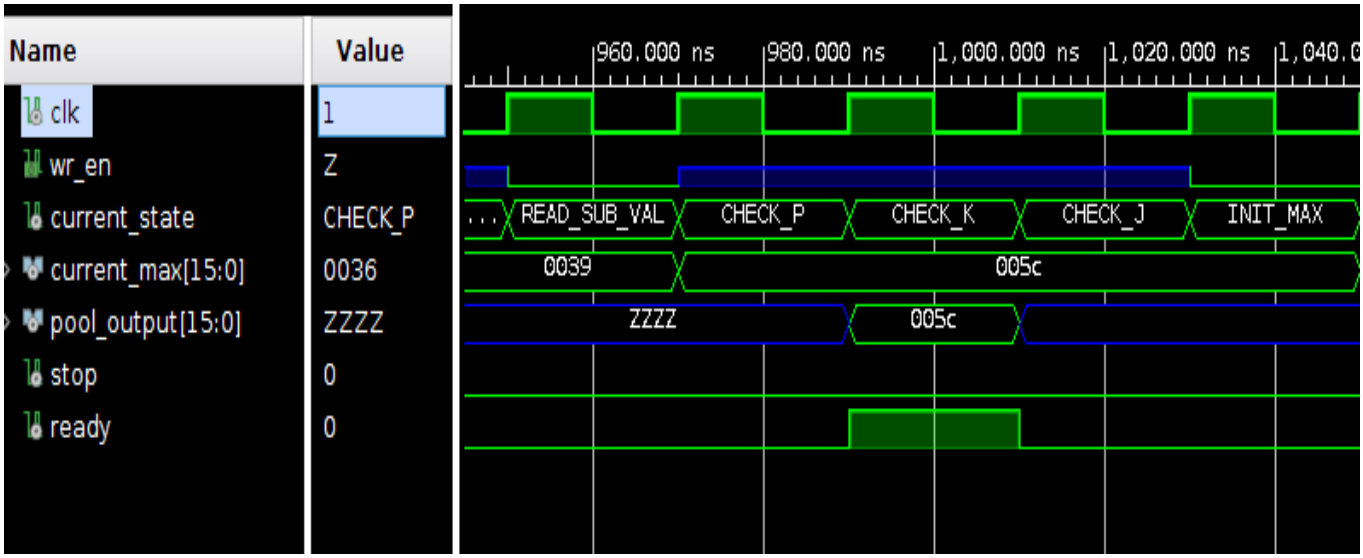
Testbench's test process that perform this operation:

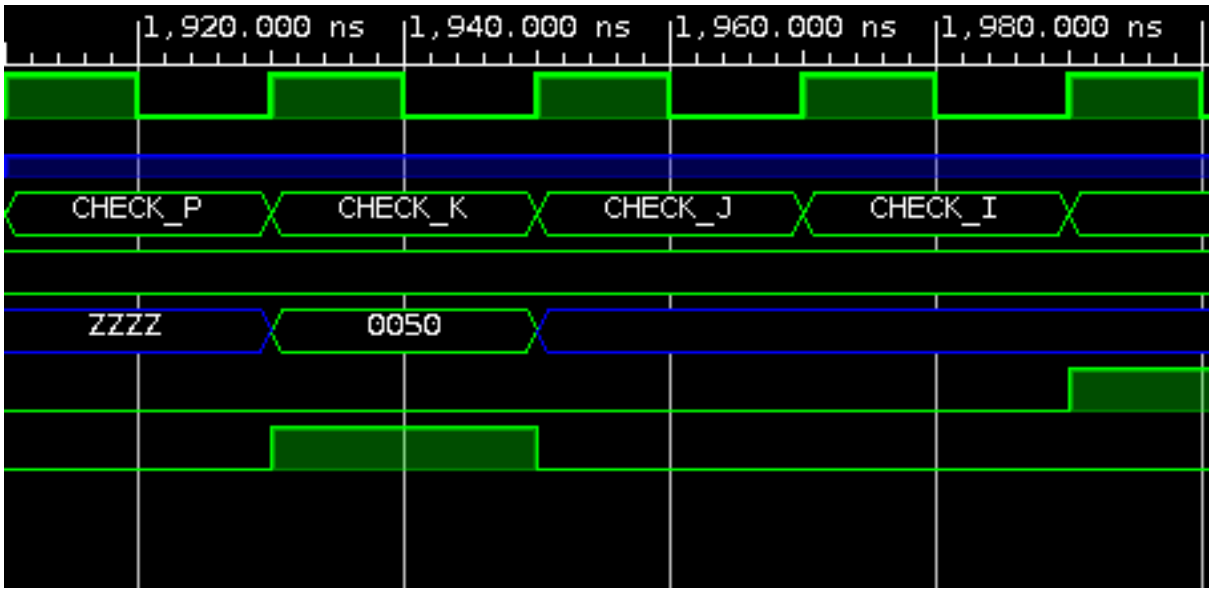
```
--TEST OF CORRECTNESS PROCESS
--read the ouput stream of datas, which are the results of Max-Pooling.
--compare, on the fly, the data captured with a predefined sequence.
process(ready,stop,pool_output)
type array_to_test is  array(0 to 1600) of integer;
variable x: array_to_test;
variable i: integer;
variable term: integer:=0;
variable v_ILINE : line;
variable v_TERM: integer;
begin
    if stop='0' then
        --write process
        if ready='1' then
            x(i):=to_integer(unsigned(pool_output));
            i:=i+1;
        end if;
    else
        --v & v process
        if term=0 then
            i:=0;
            file_open(VECTOR_VV, "pooling.mem",  read_mode);
            while not endfile(VECTOR_VV) loop
                readline(VECTOR_VV, v_ILINE);
                read(v_ILINE, v_TERM);
                assert v_TERM /= x(i) report "System failure!" severity error;
                assert v_TERM  = x(i) report "OK" severity note;
                i:=i+1;
                end loop;
            file_close(VECTOR_SAMPLE);
            term:=1;
        end if;
    end if;
end process;
```

Waveforms capturing pool_output coming from HLSM:
Important signals:

Stop: when = '1', the Max-Pooling on all matrix is computed, the HLSM is idle (goes in STOP state, after going through CHECK_I for the last time) until a reset signal.
The TestBench's test process now compares all results, saved in an array variable, with pooling.mem files containing correct results.

Ready: when='1' the HLSM has the current max-pool of a subregion


Name
Value
clk
1
wr_en
Z
current_state
CHECK_P
current_max[15:0]
0036
pool_output[15:0]
ZZZZ
stop
0
ready
0
960.000 ns
980.000 ns
1,000.000 ns
1,020.000 ns
READ_SUB_VAL
CHECK_P
CHECK_K
CHECK_J
INIT_MAX
0039
005c
ZZZZ
005c


1,920.000 ns
1,940.000 ns
1,960.000 ns
1,980.000 ns
CHECK_P
CHECK_K
CHECK_J
CHECK_I
ZZZZ
0050