Hoàng Như Vĩnh (VinhHN7@fe.edu.vn)

VinhHN7@fe.edu.vn

báo cáo Lab 2 : Mạch logic dãy và ứng dụng

Contents

[Phần 1: Các loại flip-flop 2](#_Toc165320185)

[1. Hoàn thành kịch bản kiểm tra các flip-flop 2](#_Toc165320186)

[2. Nhận xét sự khác nhau trong tín hiệu đầu ra theo mối quan hệ đầu vào 5](#_Toc165320187)

[3. Trả lời các câu hỏi 6](#_Toc165320188)

[Phần 2: Mô hình hóa và mô phỏng bộ cộng tích lũy (Accumulator) 6](#_Toc165320189)

[1. Mô hình hóa hành vi của bộ cộng tích lũy được miêu tả ở trên dùng ngôn ngữ VHDL 6](#_Toc165320190)

[2. Đánh giá tín hiệu rst\_n và set 12](#_Toc165320191)

[3. Phần 3: Ứng dụng bộ cộng tích lũy và bộ đếm để tính tích chập MAC 13](#_Toc165320192)

[1. Mô hình hóa hành vi của bộ MAC dùng ngôn ngữ VHDL. Viết kịch bản kiểm tra cho bộ MAC. 13](#_Toc165320193)

# Phần 1: Các loại flip-flop

## Hoàn thành kịch bản kiểm tra các flip-flop

Để kiểm tra hoạt động của flip-flop, các test cases sau sẽ được viết trong file flipflops\_tb.vhd với các cases như sau

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case** | **Description** | **Input Value** | **Clock Edge** | **Expected Output** |
| FFA-1 | QA is 0 when d is 0 | D = '0' | n/a | qa = '0' |
| FFA-2 | QA is 1 when d is 1 | D = '1' | n/a | qa = '1' |
| FFB-1 | QB is 0 when d is 0 | D = '0' | Rising edge | qb = '0' |
| FFB-2 | QB is 1 when d is 1 | D = '1' | Rising edge | qb = '1' |
| FFC-1 | QC is 0 when d is 0 | D = '0' | Falling edge | qc = '0' |
| FFC-2 | QC is 1 when d is 1 | D = '1' | Falling edge | qc = '1' |

Code cho testbench được viết như sau:

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

-------------------------------------------------------------------------------

ENTITY *flipflops\_tb* IS

END ENTITY flipflops\_tb;

-------------------------------------------------------------------------------

ARCHITECTURE *test* OF *flipflops\_tb* IS

  COMPONENT *flipflops*

    PORT (

      clk : IN *STD\_LOGIC*;

      d : IN *STD\_LOGIC*;

      qa : OUT *STD\_LOGIC*;

      qb : OUT *STD\_LOGIC*;

      qc : OUT *STD\_LOGIC*

    );

  END COMPONENT;

  -- component ports

  SIGNAL clk : *STD\_LOGIC* := '0';

  SIGNAL d : *STD\_LOGIC*;

  SIGNAL qa : *STD\_LOGIC*;

  SIGNAL qb : *STD\_LOGIC*;

  SIGNAL qc : *STD\_LOGIC*;

  CONSTANT clk\_cycle : *TIME* := 20 NS;

BEGIN -- ARCHITECTURE test

  -- component instantiation

  DUT : flipflops

  PORT MAP(

*clk* => clk,

*d* => d,

*qa* => qa,

*qb* => qb,

*qc* => qc

  );

  -- Define clock cycle constant

  -- clock generation

  clk\_process : PROCESS

  BEGIN

    WHILE now < 1000 ns LOOP

      clk <= '0';

      WAIT FOR clk\_cycle / 2;

      clk <= '1';

      WAIT FOR clk\_cycle / 2;

    END LOOP;

    WAIT;

  END PROCESS clk\_process;

  test\_cases : PROCESS

  BEGIN

    -- test cases for ffa

    d <= '0';

    WAIT FOR 5 ns;

    IF (clk = '1') THEN

      ASSERT (qa = '0') REPORT "FFA: qa = 0" SEVERITY ERROR;

    END IF;

    d <= '1';

    WAIT FOR 5 ns;

    IF (clk = '1') THEN

      ASSERT (qa = d) REPORT "FFA: qa = 1" SEVERITY ERROR;

    END IF;

    -- test cases for ffb

    d <= '0';

    WAIT UNTIL rising\_edge(clk);

    WAIT FOR 5 ns;

    ASSERT (qb = d) REPORT "FFB: qb = 0" SEVERITY ERROR;

    d <= '1';

    WAIT UNTIL rising\_edge(clk);

    WAIT FOR 5 ns;

    ASSERT (qb = d) REPORT "FFB: qb = 1" SEVERITY ERROR;

    -- test cases for ffc

    d <= '0';

    WAIT UNTIL falling\_edge(clk);

    WAIT FOR 5 ns;

    ASSERT (qc = d) REPORT "FFC: qc = 0" SEVERITY ERROR;

    d <= '1';

    WAIT UNTIL falling\_edge(clk);

    WAIT FOR 5 ns;

    ASSERT (qc = '1') REPORT "FFC: qc = 1" SEVERITY ERROR;

    REPORT "END TEST CASES" SEVERITY NOTE;

    WAIT;

  END PROCESS test\_cases;

END ARCHITECTURE test;

-------------------------------------------------------------------------------

CONFIGURATION flipflops\_tb\_test\_cfg OF flipflops\_tb IS

  FOR test

  END FOR;

END flipflops\_tb\_test\_cfg;

-------------------------------------------------------------------------------

Các waveform của các ffa, ffb, ffc được thấy như trong các hình sau

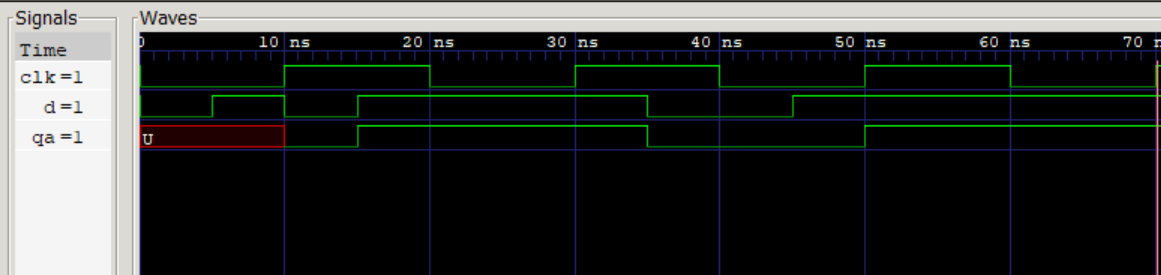


Figure 1 - Waveform của ffa

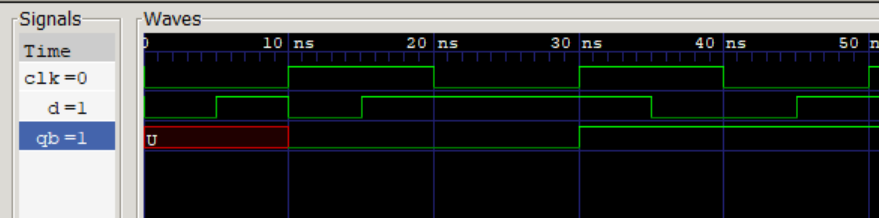


Figure 2 - Waveform của ffb

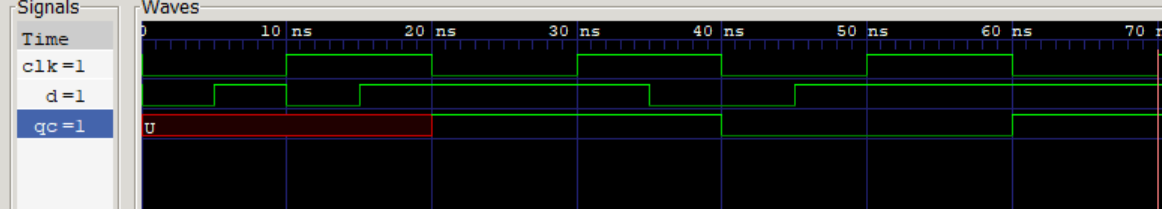


Figure 3 - Waveform của ffc

## Nhận xét sự khác nhau trong tín hiệu đầu ra theo mối quan hệ đầu vào

Đối với ffa, qa sẽ thay đổi theo d không đồng bộ theo clk. ffa còn được gọi là asynchronous d flip flop

Đối với ffb, qb sẽ thay đổi theo d đồng bộ theo sườn lên của clk. ffb còn được gọi là Rising edge d flip flop

Đối với ffc, qc sẽ thay đổi theo d đồng bộ theo sườn xuống của clk. ffc còn được gọi là falling edge d flipflop

## Trả lời các câu hỏi

1. Sensitivity list bao gồm clk và d

## Phần 2: Mô hình hóa và mô phỏng bộ cộng tích lũy (Accumulator)

## Mô hình hóa hành vi của bộ cộng tích lũy được miêu tả ở trên dùng ngôn ngữ VHDL

Mã nguồn của accumulator được viết như sau

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

USE ieee.numeric\_std.ALL;

ENTITY *accumulator* IS

  PORT (

    clk : IN *STD\_LOGIC*;

    rst\_n : IN *STD\_LOGIC*;

    set : IN *STD\_LOGIC*;

    data\_in : IN *SIGNED*(7 DOWNTO 0);

    enable : IN *STD\_LOGIC*;

    accumulator\_out : OUT *SIGNED*(11 DOWNTO 0));

END ENTITY accumulator;

ARCHITECTURE *beh* OF *accumulator* IS

  SIGNAL accumulator\_reg : *SIGNED*(11 DOWNTO 0);

BEGIN -- ARCHITECTURE beh

  accumulator\_proc : PROCESS (clk, rst\_n) IS

  BEGIN -- PROCESS accumulator\_proc

    IF rst\_n = '0' THEN -- asynchronous reset (active low)

      accumulator\_reg <= (OTHERS => '0');

    ELSIF rising\_edge(clk) THEN -- rising clock edge

      IF set = '1' THEN

        accumulator\_reg <= resize(data\_in, 12);

      ELSIF enable = '1' THEN

        IF accumulator\_reg + data\_in <= to\_signed(2047, 12) THEN

          accumulator\_reg <= accumulator\_reg + data\_in;

        ELSE

          accumulator\_reg <= to\_signed(2047, 12);

        END IF;

      END IF;

    END IF;

  END PROCESS accumulator\_proc;

  accumulator\_out <= accumulator\_reg;

END ARCHITECTURE beh;

Để viết testbench, các testcases sẽ được viết dựa theo bảng sau

|  |  |  |
| --- | --- | --- |
| **Test Case** | **Description** | **Expected Output** |
| 1 | **rst\_n = 0** | **accumulator\_out = 0** |
| 2 | **enable = 0** | **accumulator\_out = 0** |
| 3 | **data\_in = 0** | **accumulator\_out = 0** |
| 4 | **set = 1**, **data\_in = 30** | **accumulator\_out = 30** |
| 5 | Incremental accumulation with **data\_in = 30** in **2 cycles** | **accumulator\_out = 60** |
| 6 | Negative input (**data\_in = -30**) | **accumulator\_out = -30** |
| 7 | Maximum positive input (**data\_in = 127**) | **accumulator\_out = 127** |
| 8 | Maximum negative input (**data\_in = -128**) | **accumulator\_out = -128** |

Code của testbench được thể hiển như sau

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

USE ieee.numeric\_std.ALL;

ENTITY *accumulator\_tb* IS

END ENTITY accumulator\_tb;

ARCHITECTURE *behavior* OF *accumulator\_tb* IS

  -- Component declaration for the Unit Under Test (UUT)

  COMPONENT *accumulator*

    PORT (

      clk : IN *STD\_LOGIC*;

      rst\_n : IN *STD\_LOGIC*;

      set : IN *STD\_LOGIC*;

      data\_in : IN *SIGNED*(7 DOWNTO 0);

      enable : IN *STD\_LOGIC*;

      accumulator\_out : OUT *SIGNED*(11 DOWNTO 0)

    );

  END COMPONENT;

  -- Inputs

  SIGNAL clk : *STD\_LOGIC* := '0';

  SIGNAL rst\_n : *STD\_LOGIC* := '1';

  SIGNAL set : *STD\_LOGIC* := '0';

  SIGNAL data\_in : *SIGNED*(7 DOWNTO 0) := (OTHERS => '0');

  SIGNAL enable : *STD\_LOGIC* := '0';

  -- Outputs

  SIGNAL accumulator\_out : *SIGNED*(11 DOWNTO 0);

  -- Clock period definitions

  CONSTANT clk\_period : *TIME* := 10 ns;

BEGIN

  -- Instantiate the Unit Under Test (UUT)

  uut : accumulator PORT MAP(

*clk* => clk,

*rst\_n* => rst\_n,

*set* => set,

*data\_in* => data\_in,

*enable* => enable,

*accumulator\_out* => accumulator\_out

  );

  -- Clock process definitions

  clk\_process : PROCESS

  BEGIN

    WHILE now < 1000 ns LOOP

      clk <= '0';

      WAIT FOR clk\_period / 2;

      clk <= '1';

      WAIT FOR clk\_period / 2;

    END LOOP;

    WAIT;

  END PROCESS;

  -- Stimulus process

  stimulus\_proc : PROCESS

  BEGIN

    -- Test case 1: reset\_n = 0

    rst\_n <= '0';

    enable <= '1';

    data\_in <= to\_signed(30, 8);

    WAIT FOR 20 ns;

    ASSERT accumulator\_out = to\_signed(0, 12)

    REPORT "Test case 1 failed: reset\_n = 0"

      SEVERITY error;

    -- Test case 2: enable = 0

    rst\_n <= '1';

    enable <= '0';

    data\_in <= to\_signed(30, 8);

    WAIT FOR 20 ns;

    ASSERT accumulator\_out = to\_signed(0, 12)

    REPORT "Test case 2 failed: enable = 0"

      SEVERITY error;

    -- Test case 3: data\_in = 0

    rst\_n <= '1';

    enable <= '1';

    data\_in <= to\_signed(0, 8);

    WAIT FOR 20 ns;

    ASSERT accumulator\_out = to\_signed(0, 12)

    REPORT "Test case 3 failed: data\_in = 0"

      SEVERITY error;

    -- Test case 4: set = 1 to test data\_in = accumulator = 30

    rst\_n <= '1';

    enable <= '1';

    set <= '1';

    data\_in <= to\_signed(30, 8);

    WAIT UNTIL clk = '1';

    WAIT FOR 2 ns;

    ASSERT accumulator\_out = to\_signed(30, 12)

    REPORT "Test case 4 failed: set = 1"

      SEVERITY error;

    -- Reset SIGNAL

    rst\_n <= '1';

    enable <= '1';

    set <= '1';

    data\_in <= to\_signed(0, 8);

    WAIT UNTIL clk = '1';

    -- Test case 5: Test accumulator for each clock cycle

    rst\_n <= '1';

    enable <= '1';

    set <= '0';

    data\_in <= to\_signed(30, 8);

    --  Wait for 2 clock cycles

    WAIT UNTIL clk = '1' AND clk'EVENT;

    WAIT FOR 5 ns;

    WAIT UNTIL clk = '1' AND clk'EVENT;

    WAIT FOR 5 ns;

    ASSERT accumulator\_out = to\_signed(60, 12)

    REPORT "Test case 5 failed: Test accumulator for each clock cycle"

      SEVERITY error;

    -- Reset SIGNAL

    rst\_n <= '1';

    enable <= '1';

    set <= '1';

    data\_in <= to\_signed(0, 8);

    WAIT UNTIL clk = '1';

    -- Test case 6: Test accumulator with negative input

    rst\_n <= '1';

    enable <= '1';

    set <= '0';

    data\_in <= to\_signed(-30, 8);

    WAIT UNTIL clk = '1' AND clk'EVENT;

    WAIT FOR 5 ns;

    ASSERT accumulator\_out = to\_signed(-30, 8)

    REPORT "Test case 6 failed: Test accumulator with negative input"

      SEVERITY error;

    -- Reset SIGNAL

    rst\_n <= '1';

    enable <= '1';

    set <= '1';

    data\_in <= to\_signed(0, 8);

    WAIT UNTIL clk = '1';

    -- Test case 7: Test accumulator with maximum positive input

    rst\_n <= '1';

    enable <= '1';

    set <= '0';

    data\_in <= to\_signed(127, 8);

    WAIT UNTIL clk = '1' AND clk'EVENT;

    WAIT FOR 5 ns;

    ASSERT accumulator\_out = to\_signed(127, 8)

    REPORT "Test case 7 failed: Test accumulator with maximum positive input"

      SEVERITY error;

    -- Reset SIGNAL

    rst\_n <= '1';

    enable <= '1';

    set <= '1';

    data\_in <= to\_signed(0, 8);

    WAIT UNTIL clk = '1';

    -- Test case 8: Test accumulator with maximum negative input

    rst\_n <= '1';

    enable <= '1';

    set <= '0';

    data\_in <= to\_signed(-128, 8);

    WAIT UNTIL clk = '1' AND clk'EVENT;

    WAIT FOR 5 ns;

    ASSERT accumulator\_out = to\_signed(-128, 8)

    REPORT "Test case 8 failed: Test accumulator with maximum negative input"

      SEVERITY error;

    REPORT "END ACCUMULATOR\_TB PROCESS";

    WAIT;

  END PROCESS;

END;

Waveform của các testbenches được thể hiện như sau

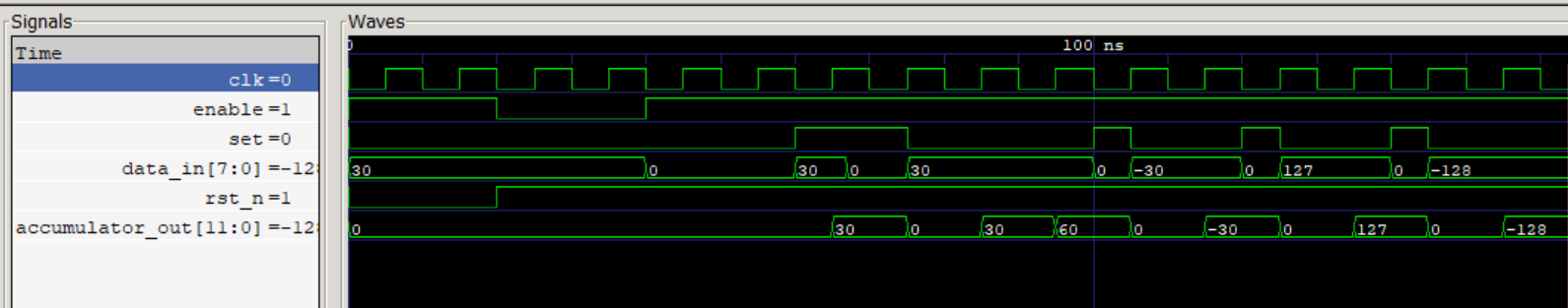


Figure 4 - Waveform của accumulator\_tb

## Đánh giá tín hiệu rst\_n và set

Trong các tín hiệu điều khiển, rst\_n là bất đồng bộ vì giá trị không được đánh giá trong rising\_edge (clk). Trong khi set là đồng bộ vì giá trị được xét trong rising\_edge (clk).

Code để viết lại cho accumulator với rst\_n đồng bộ và set bất đồng bộ như sau:

-- inputs: clk, rst\_n, set, enable, data\_in

-- outputs: accumulator\_reg

accumulator\_proc : PROCESS (clk, rst\_n, set) IS

BEGIN -- PROCESS accumulator\_proc

  IF set = '1' THEN -- asynchronous set

    accumulator\_reg <= resize(data\_in, 12);

  ELSIF rst\_n = '0' THEN -- synchronous reset (active low)

    IF rising\_edge(clk) THEN

      accumulator\_reg <= (OTHERS => '0');

    END IF;

  ELSIF rising\_edge(clk) THEN -- rising clock edge

    IF enable = '1' THEN

      IF accumulator\_reg + data\_in <= to\_signed(2047, 12) THEN

        accumulator\_reg <= accumulator\_reg + data\_in;

      END IF;

    END IF;

  END IF;

END PROCESS accumulator\_proc;

# Phần 3: Ứng dụng bộ cộng tích lũy và bộ đếm để tính tích chập MAC

## Mô hình hóa hành vi của bộ MAC dùng ngôn ngữ VHDL. Viết kịch bản kiểm tra cho bộ MAC.

Mã nguồn của bộ mac được viết như sau

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

USE ieee.numeric\_std.ALL;

LIBRARY work;

ENTITY *mac* IS

  PORT (

    clk : IN *STD\_LOGIC*;

    rst\_n : IN *STD\_LOGIC*;

    ai\_in : IN *SIGNED*(7 DOWNTO 0);

    bi\_in : IN *SIGNED*(7 DOWNTO 0);

    valid\_in : IN *STD\_LOGIC*;

    mac\_out : OUT *SIGNED*(18 DOWNTO 0);

    valid\_out : OUT *STD\_LOGIC*

  );

END ENTITY mac;

ARCHITECTURE *beh* OF *mac* IS

  SIGNAL count\_reg : *UNSIGNED*(2 DOWNTO 0) := (OTHERS => '0');

  SIGNAL accumulate\_reg : *SIGNED*(18 DOWNTO 0);

BEGIN -- ARCHITECTURE beh

  mac\_proc : PROCESS (clk, rst\_n) IS

  BEGIN -- PROCESS mac\_proc

    IF rst\_n = '0' THEN -- asynchronous reset (active low)

      accumulate\_reg <= (OTHERS => '0');

    ELSIF rising\_edge(clk) THEN -- rising clock edge

      IF valid\_in = '1' THEN

        accumulate\_reg <= accumulate\_reg + ai\_in \* bi\_in;

      END IF;

    END IF;

  END PROCESS mac\_proc;

  valid\_proc : PROCESS (clk, rst\_n) IS

  BEGIN -- PROCESS valid\_proc

    IF rst\_n = '0' THEN -- asynchronous reset (active low)

      count\_reg <= (OTHERS => '0');

    ELSIF rising\_edge(clk) THEN -- rising clock edge

      IF valid\_in = '1' THEN

        IF count\_reg = "100" THEN

          count\_reg <= "001";

        ELSE

          count\_reg <= count\_reg + 1;

        END IF;

      END IF;

    END IF;

  END PROCESS valid\_proc;

  mac\_out <= accumulate\_reg;

  -- valid\_out depends on conut\_reg, you can write your logics here

  valid\_out <= '1' WHEN count\_reg = "100" ELSE

    '0';

END ARCHITECTURE beh;

Các testcases trong testbench sẽ được viết dựa theo bảng sau:

|  |  |  |
| --- | --- | --- |
| **Test Case** | **Description** | **Expected Output** |
| 1 | ai\_in = 2, bi\_in = 3, valid\_in = '1' | mac\_out steps: 6, 12, 18, 24 after each cycle  valid\_out = '1' at the 4th cycle |
| 2 | Keep the same config of test case 1 for 4 more cycles | Mac\_out = 48  Valid\_out = ‘1’ at the 4th cycle |
| 3 | valid\_in = '0' | mac\_out maintains 48  valid\_out = '1' |
| 4 | reset\_n = 0 | mac\_out = 0, valid\_out = '0' |

Testbench được viết như sau:

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

USE ieee.numeric\_std.ALL;

-------------------------------------------------------------------------------

ENTITY *mac\_tb* IS

END ENTITY mac\_tb;

-------------------------------------------------------------------------------

ARCHITECTURE *beh* OF *mac\_tb* IS

  COMPONENT *mac*

    PORT (

      clk : IN *STD\_LOGIC*;

      rst\_n : IN *STD\_LOGIC*;

      ai\_in : IN *SIGNED*(7 DOWNTO 0);

      bi\_in : IN *SIGNED*(7 DOWNTO 0);

      valid\_in : IN *STD\_LOGIC*;

      mac\_out : OUT *SIGNED*(18 DOWNTO 0);

      valid\_out : OUT *STD\_LOGIC*

    );

  END COMPONENT;

  CONSTANT PERIOD : *TIME* := 10 NS;

  -- component ports

  SIGNAL clk : *STD\_LOGIC* := '1';

  SIGNAL rst\_n : *STD\_LOGIC* := '0';

  SIGNAL ai\_in : *SIGNED*(7 DOWNTO 0);

  SIGNAL bi\_in : *SIGNED*(7 DOWNTO 0);

  SIGNAL valid\_in : *STD\_LOGIC*;

  SIGNAL mac\_out : *SIGNED*(18 DOWNTO 0);

  SIGNAL valid\_out : *STD\_LOGIC*;

BEGIN -- ARCHITECTURE beh

  -- component instantiation

  DUT : mac

  PORT MAP(

*clk* => clk,

*rst\_n* => rst\_n,

*ai\_in* => ai\_in,

*bi\_in* => bi\_in,

*valid\_in* => valid\_in,

*mac\_out* => mac\_out,

*valid\_out* => valid\_out);

  -- clock generation

  Clk <= NOT Clk AFTER PERIOD/2;

  -- rst\_n <= '1' AFTER 5 \* PERIOD + PERIOD/3;

  -- waveform generation

  WaveGen\_Proc : PROCESS

  BEGIN

    -- set default input signals at reset

    ai\_in <= to\_signed(0, ai\_in'LENGTH);

    bi\_in <= to\_signed(0, bi\_in'LENGTH);

    valid\_in <= '0';

    WAIT FOR 15 ns;

    -- Test case 1: ai\_in = 2, bi\_in = 3, valid\_in = '1'

    ai\_in <= to\_signed(2, 8);

    bi\_in <= to\_signed(3, 8);

    valid\_in <= '1';

    rst\_n <= '1';

    WAIT UNTIL rising\_edge(clk);

    WAIT FOR 2 ns;

    -- 1 cycle, it should be 6

    ASSERT mac\_out = to\_signed(6, 19) REPORT "Test case 1 failed" SEVERITY error;

    -- Another cycle, it should be 12

    WAIT UNTIL rising\_edge(clk);

    WAIT FOR 2 ns;

    ASSERT mac\_out = to\_signed(12, 19) REPORT "Test case 1 failed" SEVERITY error;

    -- Another cycle, it should be 18

    WAIT UNTIL rising\_edge(clk);

    WAIT FOR 2 ns;

    ASSERT mac\_out = to\_signed(18, 19) REPORT "Test case 1 failed" SEVERITY error;

    -- Another cycle, it should be 24

    -- valid\_out = 1

    WAIT UNTIL rising\_edge(clk);

    WAIT FOR 2 ns;

    ASSERT mac\_out = to\_signed(24, 19) REPORT "Test case 1 failed" SEVERITY error;

    ASSERT valid\_out = '1' REPORT "Test case 1 failed" SEVERITY error;

    -- Waiting for 4 cycles, mac\_out = 48

    -- valid\_out = 1

    WAIT FOR 4 \* PERIOD;

    ASSERT mac\_out = to\_signed(48, 19) REPORT "Test case 1 failed" SEVERITY error;

    ASSERT valid\_out = '1' REPORT "Test case 1 failed" SEVERITY error;

    -- Test case 2: valid\_in = '0', keep mac\_out = 48, valid\_out = 1

    valid\_in <= '0';

    WAIT UNTIL rising\_edge(clk);

    WAIT FOR 2 ns;

    ASSERT mac\_out = to\_signed(48, 19) REPORT "Test case 2 failed" SEVERITY error;

    ASSERT valid\_out = '1' REPORT "Test case 2 failed" SEVERITY error;

    -- Test case 3: reset\_n = 0, mac\_out = 0, valid\_out = 0

    rst\_n <= '0';

    WAIT FOR 2 ns;

    ASSERT mac\_out = to\_signed(0, 19) REPORT "Test case 3 failed" SEVERITY error;

    ASSERT valid\_out = '0' REPORT "Test case 3 failed" SEVERITY error;

    REPORT "END OF SIMULATION MAC TB" SEVERITY NOTE;

    WAIT;

  END PROCESS WaveGen\_Proc;

END ARCHITECTURE beh;

-------------------------------------------------------------------------------

CONFIGURATION mac\_tb\_beh\_cfg OF mac\_tb IS

  FOR beh

  END FOR;

END mac\_tb\_beh\_cfg;

-------------------------------------------------------------------------------

Waveform của mac được thể hiện như sau:



Figure 5 - Waveform của mac\_tb

Tín hiệu rst\_n có vai trò reset valid\_out và mac\_out trở về 0