תוכן

[Work: 2](#_Toc449475770)

[Hdl desginer 3](#_Toc449475771)

[Vhdl-templates 4](#_Toc449475772)

[Simple vhdl entity + architector + library template 4](#_Toc449475773)

[State machine 6](#_Toc449475774)

[Components and instantiate: 7](#_Toc449475775)

[a. 7](#_Toc449475776)

[b. 8](#_Toc449475777)

[Declare of integer variable: 10](#_Toc449475778)

[Test bench : clk 10](#_Toc449475779)

[Psoc 11](#_Toc449475780)

[Rgb 11](#_Toc449475781)

[Mentor 12](#_Toc449475782)

[Quartus 13](#_Toc449475783)

[Max 10 14](#_Toc449475784)

# Work:

# Hdl desginer

# Vhdl-templates

## Simple vhdl entity + architector + library template

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

USE ieee.std\_logic\_arith.all;

ENTITY slave IS

PORT)

clk : IN std\_logic;

led1 : OUT std\_logic;

led2 : OUT std\_logic;

led3 : OUT std\_logic;

led4 : OUT std\_logic;

led5 : OUT std\_logic;

led6 : OUT std\_logic

);

-- Declarations

END slave ;

--

ARCHITECTURE behave OF slave IS

signal counter : integer range 0 to 5000000:=0;

type state\_type is (s0,s1); --type of state machine.

signal current\_s: state\_type:=s0; --current and next state declaration.

BEGIN

blinkLed : process(clk(

begin

if clk'EVENT and clk='1' then

--combinatorial part

CASE current\_s IS

WHEN s0 =>

counter<=counter+1;

if counter=5000000 then

led1 <='0';

led2 <='0';

led3 <='0';

led4 <='0';

led5 <='0';

led6 <='0';

current\_s <= s1;

else

led1 <='0';

led2 <='0';

led3 <='0';

led4 <='0';

led5 <='0';

led6 <='0';

counter<=0;

current\_s <= s0;

end if;

WHEN s1 =>

counter<=counter+1;

if counter=5000000 then

led1 <='1';

led2 <='1';

led3 <='1';

led4 <='1';

led5 <='1';

led6 <='1';

current\_s <= s0;

else

led1 <='1';

led2 <='1';

led3 <='1';

led4 <='1';

led5 <='1';

led6 <='1';

counter<=1;

current\_s <= s1;

end if;

END CASE;

end if;

END PROCESS;

END behave;

## State machine

-- Architecture definition for the SimpleFSM entity

Architecture RTL of SimpleFSM is

TYPE State\_type IS (A, B, C, D); -- Define the states

SIGNAL State : State\_Type; -- Create a signal that uses

BEGIN

PROCESS (clock, reset(

BEGIN

If (reset = ‘1’) THEN -- Upon reset, set the state to A

State <= A;

ELSIF (clock 'EVENT and clock='1' ) THEN -- if there is a rising edge of the

CASE State IS

WHEN A =>

IF P='1' THEN

State <= B;

END IF;

WHEN B =>

IF P='1' THEN

State <= C;

END IF;

WHEN C =>

IF P='1' THEN

State <= D;

END IF;

WHEN D=>

IF P='1' THEN

State <= B;

ELSE

State <= A;

END IF;

WHEN others =>

State <= A;

END CASE;

END IF;

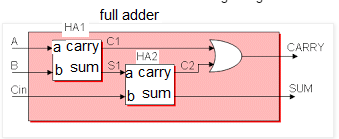
END PROCESS;

## Components and instantiate:

### a.

*-- define entity dataCounter and its architecture, then instantiate it in*  
*-- the architecture of entity FIFO:*  
  
*-- to be instantiated in FIFO later:*  
entity dataCounter is  
  port(  
    cntEn    : in std\_logic;                 *-- count enable*  
    Q        : std\_logic\_vector(7 downto 0); *-- count value*  
    clk, rst : std\_logic);  
end dataCounter;  
  
architecture arch of dataCounter is  
  [...]  
end arch;  
  
entity FIFO is  
  [...]  
end FIFO;  
  
architecture arch of FIFO is  
  [...]  
  *-- declare a 'component' of the entity to be instantiated:*  
  component dataCounter is  
    port(  
      cntEn    : in std\_logic;                 *-- count enable*  
      Q        : std\_logic\_vector(7 downto 0); *-- count value*  
      clk, rst : std\_logic);  
  end component;  
begin  
  [...]  
  *-- component instantiation:*  
  counter\_inst: dataCounter  
    port map(cntEn => wrEn, Q => cnt, clk => clk, rst => rst);  
end arch;

### b.



--top module(full adder) entity declaration  
**entity** fulladder **is**  
    **port** (a : **in** std\_logic;  
            b : **in** std\_logic;  
           cin : **in** std\_logic;  
           sum : **out** std\_logic;  
           carry : **out** std\_logic  
         );  
**end** fulladder;  
--top module architecture declaration.  
**architecture** behavior **of** fulladder **is**  
--sub-module(half adder) is declared as a component before the keyword "begin".  
   **component** halfadder  
    **port**(  
         a : **in** std\_logic;  
         b : **in** std\_logic;  
         sum : **out** std\_logic;  
         carry : **out** std\_logic  
        );  
    **end** **component**;  
--All the signals are declared here,which are not a part of the top module.  
--These are temporary signals like 'wire' in Verilog.  
**signal** s1,c1,c2 : std\_logic:='0';  
  
**begin**  
--instantiate and do port map for the first half adder.  
  HA1 : halfadder **port** **map** (  
          a => a,  
          b => b,  
          sum => s1,  
          carry => c1  
        );  
--instantiate and do port map for the second half adder.  
 HA2 : halfadder **port** **map** (  
          a => s1,  
          b => cin,  
         sum => sum,  
         carry => c2  
        );  
carry <= c1 **or** c2;  --final carry calculation  
  
**end**;

כדי להכריז על קומפוננטה יש להעתיק את הentity בדיוק כמו שהיא אל קובץ VHDL כלשהו לפני ה begin , לאחר שהצהרנו על כל הקומפוננטות בקוד שלנו אחרי הbegin נשתמש במילה portmap כדי לחבר את הקווים אל הקומפוננטה , מה שמצד שמאל אלו ההדקים בדיוק כפי שהגדרו אותם ב entity , מה שמצד ימין אלו הקווים ב top שלנו .

**איך לזכור portmap : ההדקים של הcomponent עצמו הם שמאלייים תמיד ומה שמחברים אליהם ימניים**

**תיקון תהליך עבודה נכון: יוצרים קובץ VHD כלשהו , כאשר עולים לרמה אחת מעליו פשוט מעתיקים את הentity כמו שהוא דוג :**

dataCounter **.vhd**

entity dataCounter is  
  port(  
    cntEn    : in std\_logic;                 *-- count enable*  
    Q        : std\_logic\_vector(7 downto 0); *-- count value*  
    clk, rst : std\_logic);  
end dataCounter;

**ברמה מעליו top.vhd לפני הbegin ואחרי ה architctor מצהירים פשוט עתיקיםמ entity datacounter מדביקים ורושמים במקום entity – component**

architecture behave of top is :

component dataCounter    
  port(  
    cntEn    : in std\_logic;                 *-- count enable*  
    Q        : std\_logic\_vector(7 downto 0); *-- count value*  
    clk, rst : std\_logic);  
end component ;

Begin

End behave;

לאחר מכן מעתיקים את התוכן של הentity בשביל לחבר את החוטים ברמה העליונה ורושמים port map( במקום component איפה שרשום X לחבר את הסיגנל שרוצים ,אפשר גם במפורש '1'

architecture behave of top is :

component dataCounter    
  port(  
**cntEn**    : in std\_logic;                 *-- count enable*  
**Q**        : std\_logic\_vector(7 downto 0); *-- count value*  
**clk, rst** : std\_logic);  
end component ;

Begin

Inst1 : dataCounter

Port map (

**cntEn** =>X,

**Q** =>X,

**Clk**=>X ,

**Rst** =>X

);

End behave;

## Declare of integer variable:

ARCHITECTURE behave OF slave IS

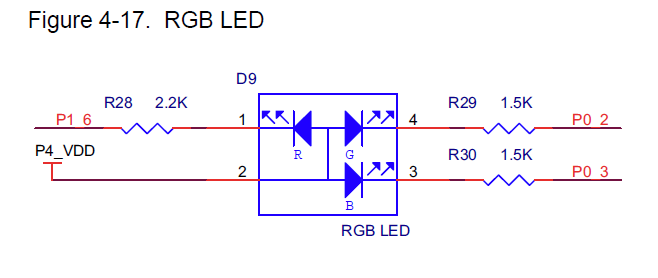
signal counter : integer range 0 to 5000000:=0;

begin

## Test bench : clk

# Psoc

## Rgb

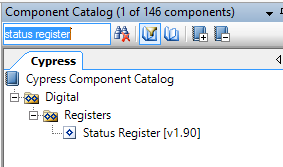
if I want green connect to p0\_2

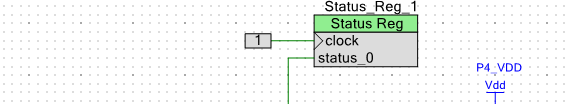
If I want blue connect to p0\_3

If I want red connect to p1\_6

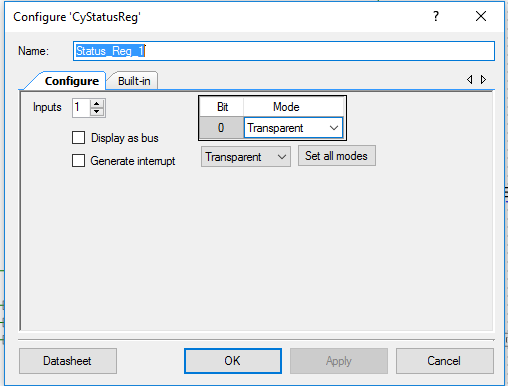
## Status register

Put status\_reg component in schematic





**Put '1' to clock if u use transpernt mode(alwys read in the background) , in stick mode u have to enter a clk and then the status register will only make a read when I manually request it with a read , this way I can save the reading**



Connect something u want to monitor to status\_0 ..

In main c just use: **uint8 READ\_VALUE=Status\_Reg\_1\_read();**

If u want to display it in uart just use : UART\_lampel\_PutChar('0'+READ\_VALUE);

# Mentor

# Quartus

# Max 10

**Diamond**

**machxo**