

**Example 6.3. Registered add-and-compare circuit**

|  |  |
| --- | --- |
| 123456789101112131415161718192021222324252627282930313233343536 | --------------------------------------------------------------**library** ieee;**use** ieee.std\_logic\_1164.**all**;**use** ieee.numeric\_std.all;**entity** add\_compare\_registered **is** **generic** ( NUM\_BITS: natural:= 8); **port** ( clk: **in** std\_logic; a, b: **in** std\_logic\_vector(NUM\_BITS-1 **downt**o 0); comp: **out** std\_logic; sum: **out** std\_logic\_vector(NUM\_BITS **downt**o 0));**end entity** add\_compare\_registered;**architecture** rtl **of** add\_compare\_registered **is** **signal** a\_uns, b\_uns: unsigned(NUM\_BITS **downto** 0);**begin** a\_uns <= unsigned('0' & a); b\_uns <= unsigned('0' & b); **process** (clk) **begin** **if** rising\_edge(clk) **then** **if** a\_uns > b\_uns **then**  comp <= '1'; **else** comp <= '0'; **end if**; sum <= std\_logic\_vector(a\_uns + b\_uns); **end if**; **end process**;**end architecture** rtl;-------------------------------------------------------------- |

**Example 10.3. Carry-ripple adder built with full-adder components**

|  |  |
| --- | --- |
| 1234567891011121314151612345678910111213141516171819202122232425 | --------------------------------------------------------------**library** ieee;**use** ieee.std\_logic\_1164.**all**;**entity** full\_adder\_unit **is**  **port** ( in1, in2, cin: **in** std\_logic; sum, cout: **out** std\_logic);**end entity**;**architecture** boolean **of** full\_adder\_unit **is****begin** sum <= in1 **xor** in2 **xor** cin; cout <= (in1 **and** in2) **or** (in1 **and** cin) **or** (in2 **and** cin);**end architecture**;----------------------------------------------------------------------------------------------------------------------------**library** ieee;**use** ieee.std\_logic\_1164.**all**;**entit**y carry\_ripple\_adder **is**  **generic** ( NUM\_BITS: natural := 8); **port** ( a, b: **in** std\_logic\_vector(NUM\_BITS-1 **downto** 0); cin: **in** std\_logic; sum: **out** std\_logic\_vector(NUM\_BITS-1 **downto** 0); cout: **out** std\_logic);**end entity**;**architecture** structural **of** carry\_ripple\_adder **is** **signal** carry: std\_logic\_vector(0 **to** NUM\_BITS);**begin** carry(0) <= cin; gen\_adder: **for** i **in** 0 **to** NUM\_BITS-1 **generate** adder: **entity** work.full\_adder\_unit  **port map** (a(i), b(i), carry(i), sum(i), carry(i+1)); **end generate**; cout <= carry(NUM\_BITS);**end architecture**;-------------------------------------------------------------- |

**Example 10.4. Hamming-weight calculator**

|  |  |
| --- | --- |
| 12345678910111213141516171819202122232425 | ----------------------------------------------------------------------------------**library** ieee;**use** ieee.std\_logic\_1164.**all**;**use** ieee.numeric\_std.**all**;**entity** hamming\_weight\_calculator **is** **generic** ( BITS\_IN: positive := 16; BITS\_OUT: positive := 5); --calculated by user as ceil(log2(BITS\_IN+1)) **port** ( inp\_vector: **in** std\_logic\_vector(BITS\_IN-1 **downto** 0); hamm\_weight: **out** std\_logic\_vector(BITS\_OUT-1 **downto** 0));**end entity**;**architecture** concurrent **of** hamming\_weight\_calculator **is** **type** integer\_array **is** **array** (0 **to** BITS\_IN) **of** integer **range** 0 **to** BITS\_IN;  **signal** internal: integer\_array;**begin** internal(0) <= 0; gen: **for** i **in** 1 **to** BITS\_IN **generate** internal(i) <= internal(i-1) + 1 **when** inp\_vector(i-1) **else** internal(i-1); **end generate**; hamm\_weight <= std\_logic\_vector(to\_unsigned(internal(BITS\_IN), BITS\_OUT));**end architecture**;---------------------------------------------------------------------------------- |

|  |
| --- |
| **use** ieee.math\_real.**all**;... **generic** ( BITS\_IN: positive := 16; BITS\_OUT: positive := integer(ceil(log2(real(BITS\_IN+1))))); --a *dependent* constant **port** (... |

|  |
| --- |
| **entity** hamming\_weight\_calculator **is** **generic** ( BITS\_IN: positive := 16); **port** ( inp\_vector: **in** std\_logic\_vector(BITS\_IN-1 **downto** 0); hamm\_weight: **out** std\_logic\_vector(integer(ceil(log2(real(BITS\_IN+1))))-1 **downto** 0)));**end entity**;**architecture** concurrent **of** hamming\_weight\_calculator **is** **constant** BITS\_OUT: positive := integer(ceil(log2(real(BITS\_IN+1)))); ... |

**Example 10.5. Signed integer adder**

|  |  |
| --- | --- |
| 12345678910111213141516171819202122232425262728293031 | ----------------------------------------------------------------**library** ieee;**use** ieee.std\_logic\_1164.**all**;**use** ieee.numeric\_std.**all**;**entity** adder\_signed **is** **generic** ( NUM\_BITS: integer := 4);  **port** ( a, b: **in** std\_logic\_vector(NUM\_BITS-1 **downto** 0); cin: **in** std\_logic; sum: **out** std\_logic\_vector(NUM\_BITS-1 **downto** 0); sumMSB, cout, oflow: **out** std\_logic);**end entity**;**architecture** suggested **of** adder\_signed **is** **signal** sum\_sig: signed(NUM\_BITS **downto** 0);**begin** --Sign-extension, conversion to signed, and addition: sum\_sig <= signed(a(NUM\_BITS-1) & a) + signed(b) + cin; --sum\_sig <= resize(signed(a), NUM\_BITS+1) + signed(b) + cin;   --Conversion to std\_logic\_vector and final operations: sum <= std\_logic\_vector(sum\_sig(NUM\_BITS-1 **downto** 0)); sumMSB <= sum\_sig(NUM\_BITS); cout <= a(NUM\_BITS-1) **xor** b(NUM\_BITS-1) **xor** sumMSB; oflow <= sum\_sig(NUM\_BITS) **xor** sum\_sig(NUM\_BITS-1);**end architecture**;---------------------------------------------------------------- |

**Example 11.2. Programmable combinational delay line (structural)**

|  |  |
| --- | --- |
| 12345678910111213141516171819202122232425261234567891011121314151617181920212223242526272829 | --The component (delay block)----------------------------------------------------------------**library** ieee;**use** ieee.std\_logic\_1164.**all**;**entity** delay\_block **is** **generic** ( BLOCK\_LENGTH: natural); **port** ( din: **in** std\_logic; sel: **in** std\_logic; dout: **out** std\_logic); **end entity**;**architecture** delay\_block **of** delay\_block **is**  **signal** node\_vector: std\_logic\_vector(0 **to** 2\*BLOCK\_LENGTH); **attribute** keep: boolean; **attribute** keep **of** node\_vector: **signal is** true;**begin** node\_vector(0) <= din; gen: **for** i **in** 1 **to** 2\*BLOCK\_LENGTH **generate** node\_vector(i) <= **not** node\_vector(i-1); **end generate**; dout <= node\_vector(2\*BLOCK\_LENGTH) **when** sel **else** din;**end architecture**;------------------------------------------------------------------Main code (complete delay line)----------------------------------------------------------------**library** ieee;**use** ieee.std\_logic\_1164.**all**;**entity** delay\_line **is**  **generic** ( NUM\_BLOCKS: natural := 4); --this is M in figure 11.2b **port** ( din: **in** std\_logic; --sel: **in** std\_logic\_vector(0 **to** NUM\_BLOCKS-1); sel: **in** std\_logic\_vector(NUM\_BLOCKS-1 downto 0); dout: **out** std\_logic); **end entity**;**architecture** structural **of** delay\_line **is**  **signal** node\_vector: std\_logic\_vector(0 **to** NUM\_BLOCKS); **attribute** keep: boolean; **attribute** keep **of** node\_vector: **signal is** true;**begin** node\_vector(0) <= din; gen: **for** i **in** 1 **to** NUM\_BLOCKS **generate** blocki: **entity** work.delay\_block  **generic map** (2\*\*(i-1))  **port map** (node\_vector(i-1), sel(i-1), node\_vector(i)); **end generate**; dout <= node\_vector(NUM\_BLOCKS);**end architecture**;---------------------------------------------------------------- |

**Example 12.3. Counter with *is\_max* flag and RTL analysis**

|  |  |
| --- | --- |
| 12345678910111213141516171819202122232425262728293031323334353637 | ----------------------------------------------------------------**library** ieee;**use** ieee.std\_logic\_1164.**all**;**use** ieee.numeric\_std.**all**;**entity** counter\_with\_is\_max\_flag **is** **generic** ( MAX: natural := 6; BITS: natural := 3); --not independent (see section 6.7) **port** ( clk: **in** std\_logic; count: **out** std\_logic\_vector(BITS-1 **downto** 0); is\_max: **out** std\_logic);**end entity**; **architecture** rtl **of** counter\_with\_is\_max\_flag **is** **signal** i: natural **range** 0 **to** MAX;**begin** P1: **process** (clk) **begin** **if** rising\_edge(clk) **then** **if** i=MAX-1 **then** is\_max <= '1'; i <= i + 1; **elsif** i=MAX **then** is\_max <= '0'; i <= 0; **else** i <= i + 1; **end if**; **end if**; **end process**; count <= std\_logic\_vector(to\_unsigned(i, BITS));**end architecture**;---------------------------------------------------------------- |

|  |  |
| --- | --- |
| 2021222324252627282930313233343536373839 |  P2: **process** (clk) **begin** **if** rising\_edge(clk) **then**  --Counter: **if** i /= MAX **then** i <= i + 1; **else** i <= 0; **end if**; --Flag generator: **if** i=MAX-1 **then** is\_max <= '1'; **else** is\_max <= '0'; **end if**; **end if**; **end process**; |

# Example 12.4. Counters with signal and variable

|  |  |
| --- | --- |
| 12345678910111213141516171819202122232425262728293031323334 | --------------------------------------------------**entity** dual\_counter **is** **port** ( clk: **in** bit; count1, count2: **out** natural **range** 0 **to** 10);**end entity**;**architecture** rtl **of** dual\_counter **is****begin** counter1: **process** (clk) **begin** **if** clk'event **and** clk='1' **then** count1 <= count1 + 1; **if** count1=10 **then**  count1 <= 0; **end if**; **end if**; **end process** counter1; counter2: **process** (clk) **variable** var: natural **range** 0 **to** 10; **begin** **if** clk'event **and** clk='1' **then** var := var + 1; **if** var=10 **then**  var := 0; **end if**; **end if**; count2 <= var; **end process** counter2;**end architecture**;-------------------------------------------------- |

**Example 12.10. Recommended shift register implementation**

|  |  |
| --- | --- |
| 1234567891011121314151617181920212223242526272829 | ----------------------------------------------------------------------------------------**library** ieee;**use** ieee.std\_logic\_1164.**all**;**entity** shift\_register **is** **generic** ( NUM\_BITS: natural := 8;  NUM\_STAGES: natural := 4); **port** ( clk: **in** std\_logic; din: **in** std\_logic\_vector(NUM\_BITS-1 **downto** 0); dout: **out** std\_logic\_vector(NUM\_BITS-1 **downto** 0));**end entity**;  **architecture** recommended **of** shift\_register **is** **type** slv\_array **is array** (0 **to** NUM\_STAGES-1) **of** std\_logic\_vector(NUM\_BITS-1 **downto** 0); **signal** q: slv\_array;**begin** **process** (clk) **begin** **if** rising\_edge(clk) **then** q <= din & q(0 **to** NUM\_STAGES-2); **end if**; dout <= q(NUM\_STAGES-1); **end process**;**end architecture**;---------------------------------------------------------------------------------------- |

**Example 13.1. Generic tree-type adder array**

|  |  |
| --- | --- |
| 123456712345678910111213141516171819202122232425262728293031323334353637383940414243 | -------------------------------------------------------------**library** ieee;**use** ieee.numeric\_std.**all**;**package** user\_defined\_type\_pkg **is** **type** signed\_vector **is array** (natural **range** <>) **of** signed;**end package**;-----------------------------------------------------------------------------------------------------------------------------------------------------**library** ieee;**use** ieee.numeric\_std.**all**;**use** ieee.math\_real.**all**;**use** work.user\_defined\_type\_pkg.**all**;**entity** adder\_array\_generic\_tree **is** **generic** ( NUM\_INPUTS: natural := 10;  NUM\_BITS: natural := 7); **port** ( x: **in** signed\_vector(0 **to** NUM\_INPUTS-1)(NUM\_BITS-1 **downto** 0); sum: **out** signed(NUM\_BITS+integer(ceil(log2(real(NUM\_INPUTS))))-1 **downto** 0)); **end entity**;**architecture** tree\_type\_generic **of** adder\_array\_generic\_tree **is** **constant** LAYERS: natural := integer(ceil(log2(real(NUM\_INPUTS)))); **constant** PWR\_OF\_TWO: natural := 2\*\*LAYERS; **alias** EXTRA\_BITS: natural **is** LAYERS;**begin** **process** (**all**) **variable** accum: signed\_vector(0 **to** PWR\_OF\_TWO-1)(NUM\_BITS+EXTRA\_BITS-1 **downto** 0); **begin**  --Initialization: loop1: **for** i **in** 0 **to** NUM\_INPUTS-1 **loop** accum(i) := resize(x(i), NUM\_BITS+EXTRA\_BITS); **end loop** loop1; accum(NUM\_INPUTS **to** PWR\_OF\_TWO-1) := (**others** => (**others** => '0'));  --Generic tree-type adder array: loop3: **for** j **in** 1 **to** LAYERS **loop** loop4: **for** i **in** 0 **to** PWR\_OF\_TWO/(2\*\*j)-1 **loop** accum(i) := accum(2\*i) + accum(2\*i+1); **end loop** loop4;  **end loop** loop3; sum <= accum(0); **end process**; **end architecture**;---------------------------------------------------------------------------------------- |

**Example 13.2. Single-switch debouncer**

|  |  |
| --- | --- |
| 123456789101112131415161718192021222324252627282930313233343536373839404142434445 | ----------------------------------------------------------------------------------------**library** ieee;**use** ieee.std\_logic\_1164.**all**;**use** ieee.numeric\_std.**all**;**use** ieee.math\_real.**all**;**entity** debouncer **is** **generic** ( T\_DEB\_MS: natural := 25; --minimum debounce time in ms F\_CLK\_KHZ: natural := 50\_000); --clock frequency in kHz **port** ( x, clk: **in** std\_logic; y: **out** std\_logic);**end entity**;**architecture** single\_switch **of** debouncer **is**  **constant** COUNTER\_BITS: natural := 1 + integer(ceil(log2(real(T\_DEB\_MS\*F\_CLK\_KHZ)))); **signal** x\_reg: std\_logic;**begin** **process** (clk) **variable** count: unsigned(COUNTER\_BITS-1 **downto** 0); **begin** --Timer (with input register): **if** rising\_edge(clk) **then** x\_reg <= x; **if** y=x\_reg **then** count := (**others** => '0'); **else** count := count + 1; **end if**; **end if**; --Output register: **if** falling\_edge(clk) **then** **if** count(COUNTER\_BITS-1) **then** y <= **not** y; **end if**; **end if**; **end process**;**end architecture**;---------------------------------------------------------------------------------------- |

**Example 14.2. Function *ceil\_log2* in a package**

|  |  |
| --- | --- |
| 1234567891011121314151612345678910111213141516 | ----------------------------------------------------------**package** subprograms\_pkg **is**  **function** ceil\_log2 (input: positive) **return** natural;**end package**;**package body** subprograms\_pkg **is** **function** ceil\_log2 (input: positive) **return** natural **is** **variable** result: natural := 0; **begin** **while** 2\*\*result < input **loop** result := result + 1; **end loop**; **return** result; **end function** ceil\_log2;**end package body**;--------------------------------------------------------------------------------------------------------------------**use** work.subprograms\_pkg.**all**;**entity** test\_circuit **is** **generic** ( BITS: natural := 8);  **port** ( inp: **in** positive **range** 1 **to** 2\*\*BITS-1; outp: **out** natural **range** 0 **to** BITS);**end entity**;**architecture** test\_circuit **of** test\_circuit **is****begin** outp <= ceil\_log2(inp);**end architecture**;---------------------------------------------------------- |

**Example 14.4. Function *slv\_to\_integer* in a process**

|  |  |
| --- | --- |
| 1234567891011121314151617181920212223242526272829303132333435 | ---------------------------------------------------------------------------**library** ieee;**use** ieee.std\_logic\_1164.**all**;**entity** test\_circuit **is** **generic** ( **WIDTH**: natural := 8);  **port** ( clk: **in** std\_logic; inp: **in** std\_logic\_vector(**WIDTH**-1 **downto** 0); outp: **out** integer **range** 0 **to** 2\*\***WIDTH**-1);**end entity**;**architecture** test\_circuit **of** test\_circuit **is****begin** **process** **function** slv\_to\_integer (slv: std\_logic\_vector) **return** integer **is** **variable** result: integer **range** 0 **to** 2\*\*slv'length-1 := 0;  **begin** **for** i **in** slv'**range** **loop** result := result\*2; **if** slv(i)='1' **or** slv(i)='H' **then**  result := result + 1; **end if**; **end loop**; **return** result; **end function** slv\_to\_integer; **begin** **wait until** clk; outp <= slv\_to\_integer(inp); **end process**;**end architecture**;--------------------------------------------------------------------------- |

**Example 16.2. Garage door controller**

|  |  |
| --- | --- |
| 12345678910111213141516171819202122232425262728293031323334353637383940414243444546474849505152535455565758596061626364656667686970717273747576777879808182838485868788899091929394959697 | ---------------------------------------------------------------------------**library** ieee;**use** ieee.std\_logic\_1164.**all**;**entity** garage\_door\_controller **is** **port** ( clk, rst: **in** std\_logic; remote, door\_closed, door\_open: **in** std\_logic; motor, direction: **out** std\_logic);**end entity**;**architecture** fsm **of** garage\_door\_controller **is**  **type** state\_type is (ready\_to\_open, opening1, opening2,  opening3, ready\_to\_close, closing1, closing2, closing3);  **signal** state: state\_type;**begin** --Logic + register for state: **process** (clk, rst) **begin** **if** rst **then** state <= ready\_to\_open; **elsif** rising\_edge(clk) **then** **case** state **is** **when** ready\_to\_open => **if** door\_open **then** state <= ready\_to\_close; **elsif** remote **then** state <= opening1; **else** state <= ready\_to\_open; **end if**; **when** opening1 => **if** not remote **then** state <= opening2; **else** state <= opening1; **end if**;  **when** opening2 => **if** door\_open **then** state <= ready\_to\_close; **elsif** remote **then** state <= opening3; **else** state <= opening2; **end if**;  **when** opening3 => ...  **when** ready\_to\_close => ...  **when** closing1 => ...  **when** closing2 => ...  **when** closing3 => ...  **end case**; **end if**; **end process**; --Logic + register for outputs: **process** (clk) **begin** **if** rising\_edge(clk) **then** **case** state is **when** ready\_to\_open => motor <= '0'; direction <= '-'; **when** opening1 => motor <= '1'; direction <= '1'; **when** opening2 => motor <= '1'; direction <= '1'; **when** opening3 => motor <= '0'; direction <= '-'; **when** ready\_to\_close => motor <= '0'; direction <= '-'; **when** closing1 => motor <= '1'; direction <= '0'; **when** closing2 => motor <= '1'; direction <= '0'; **when** closing3 => motor <= '0'; direction <= '-'; **end case**; **end if**; **end process**; --Notice that default values could have been used in the process above **end architecture**;--------------------------------------------------------------------------- |

**Example 16.4. SPI interface for an A/D converter**

|  |  |
| --- | --- |
| 123456789101112131415161718192021222324252627282930313233343536373839404142434445464748495051525354555657585960616263646566676869707172737475767778798081828384858687888990919293949596979899100101102103104105106107108109110111112113114115116117118119120121122123 | --------------------------------------------------------------------------**library** ieee;**use** ieee.std\_logic\_1164.**all**;**entity** SPI\_interface\_for\_MAX1118 **is** **port** ( clk, rst: **in** std\_logic; rd: **in** std\_logic; SSn, SCLK: **out** std\_logic; MISO: **in** std\_logic; received\_data: **out** std\_logic\_vector(7 **downto** 0));**end entity**;**architecture** fsm **of** SPI\_interface\_for\_MAX1118 **is** **signal** clk\_5MHz: std\_logic; **type** state\_type is (idle, convert, read\_data, hold); **signal** pr\_state, nx\_state: state\_type; **signal** t, tmax: natural **range** 0 **to** 37;**begin** --Generate SPI clock (5 MHz): **process** (clk) **variable** count: natural **range** 0 **to** 4; **begin** **if** rising\_edge(clk) **then** **if** count=4 **then** clk\_5MHz <= **not** clk\_5MHz; count := 0; **else** count := count + 1; **end if**; **end if**; **end process**;  --Register for machine state: **process** (clk\_5MHz, rst) **begin** **if** rst **then** pr\_state <= idle; **elsif** rising\_edge(clk\_5MHz) **then** pr\_state <= nx\_state; **end if**; **end process**; --Logic for machine state: **process** (**all**) **begin** **case** pr\_state **is** **when** idle => **if** rd **then** nx\_state <= convert; **else** nx\_state <= idle; **end if**; **when** convert => **if** t=tmax **then** nx\_state <= read\_data; **else** nx\_state <= convert; **end if**; **when** read\_data => **if** t=tmax **then** nx\_state <= hold; **else** nx\_state <= read\_data; **end if**; **when** hold => **if not** rd **then** nx\_state <= idle; **else** nx\_state <= hold; **end if**;  **end case**; **end process**; --Logic for machine outputs: **process** (**all**) **begin** **case** pr\_state **is** **when** idle =>  SSn <= '1'; SCLK <= '1'; **when** convert =>  SSn <= '0'; SCLK <= '1'; **when** read\_data =>  SSn <= '0'; SCLK <= clk\_5MHz; **when** hold =>  SSn <= '1'; SCLK <= '1'; **end case**; **end process**;  --Store data read from ADC: **process** (clk\_5MHz) **begin** **if** rising\_edge(clk\_5MHz) **then** **if** pr\_state = read\_data **then** received\_data(7-t) <= MISO; **end if**; **end if**; **end process**; --Timer: **process** (**all**) **begin** **case** pr\_state **is** **when** convert => tmax <= 37; **when** read\_data => tmax <= 7; **when others** => tmax <= 0; **end case**; **if** rising\_edge(clk\_5MHz) **then** **if** pr\_state /= nx\_state **then** t <= 0; **elsif** t /= tmax **then** t <= t + 1; **end if**; **end if**; **end process**; **end architecture**;-------------------------------------------------------------------------- |

**Example 17.1. SPI Interface for an EEPROM Device (with FSM)**

|  |  |
| --- | --- |
| 123456789101112131415161718192021222324252627282930313233343536373839404142434445464748495051525354555657585960616263646566676869707172737475767778798081828384858687888990919293949596979899100101102103104105106107108109110111112113114115116117118119120121122123124125126127128129130131132133134135136137138139140141142143144145146147148149150151152153154155156157158159160161162163164165166167168169170171172173174175176177178179180181182183184185186187188189190191192193194195196197198199200201202203204205206207208209210211212213214215 | -----------------------------------------------------------------------------**library** ieee;**use** ieee.std\_logic\_1164.**all**;**entity** SPI\_interface\_for\_EEPROM **is** **generic** ( WREN\_OPCODE: std\_logic\_vector(7 **downto** 0) := "00000110"; WRITE\_OPCODE: std\_logic\_vector(7 **downto** 0) := "00000010"; READ\_OPCODE: std\_logic\_vector(7 **downto** 0) := "00000011"; INITIAL\_WRITE\_ADDR: std\_logic\_vector(15 **downto** 0) := (**others** => '0'); INITIAL\_READ\_ADDR: std\_logic\_vector(15 **downto** 0) := (**others** => '0')); **port** ( clk, rst: **in** std\_logic; wr, rd: **in** std\_logic; SSn: **out** std\_logic; SCLK: **out** std\_logic; MOSI: **out** std\_logic; MISO: **in** std\_logic; data\_read\_from\_EEPROM: **out** std\_logic\_vector(7 **downto** 0));**end entity**;**architecture** fsm **of** SPI\_interface\_for\_EEPROM **is** --Reference clock (12.5 MHz): **signal** clk\_12MHz: std\_logic;   --FSM-related declarations:  **type** state\_type **is** (idle, WREN\_OP, deselect1, deselect2, WRITE\_OP,  initial\_wr\_addr, write\_data, wait\_wr\_low, READ\_OP, initial\_rd\_addr, read\_data, wait\_rd\_low);  **signal** pr\_state, nx\_state: state\_type;  --Timer-related declarations: **signal** i, imax: natural **range** 0 **to** 15; **signal** j, jmax: natural **range** 0 **to** 4;  --Test data (to be written to and read from EEPROM): **type** data\_array **is array** (natural **range** <>) **of** std\_logic\_vector; **constant** test\_data\_out: data\_array(0 **to** 4)(7 **downto** 0) := ("00000001", "10000000", "00111100", "11000011", "11111111"); **signal** test\_data\_in: data\_array(0 **to** 4)(7 **downto** 0); **begin** --Generate SPI clock (12.5MHz) from system clock (50MHz): **process** (clk) **variable** count: natural **range** 0 **to** 1; **begin** **if** rising\_edge(clk) **then** **if** count=1 **then** clk\_12MHz <= **not** clk\_12MHz; count := 0; **else** count := count + 1; **end if**; **end if**; **end process**;  --Dual timer (i=primary, j=secondary): **process** (**all**) **begin** --Define imax and jmax values: **case** pr\_state **is** **when** WREN\_OP | WRITE\_OP | write\_data | READ\_OP | read\_data => imax <= 7; **when** initial\_wr\_addr | initial\_rd\_addr => imax <= 15; **when others** => imax <= 0; **end case**; **case** pr\_state **is** **when** write\_data | read\_data => jmax <= 4; **when others** => jmax <= 0; **end case**; --Implement pointers i and j: **if** rst **then** i <= 0; j <= 0; **elsif** falling\_edge(clk\_12MHz) **then** **if** pr\_state /= nx\_state **then** i <= 0; j <= 0; **elsif not** (i=imax **and** j=jmax) **then** **if** i/=imax **then** i <= i + 1; **elsif** j/=jmax **then** i <= 0; j <= j + 1; **end if**; **end if**; **end if**; **end process**;  --State register (block R1 of figure 15.2b): **process** (clk\_12MHz, rst) **begin** **if** rst **then**  pr\_state <= idle; **elsif** falling\_edge(clk\_12MHz) **then** pr\_state <= nx\_state; **end if**; **end process**;  --State logic (block L1 of figure 15.2b): **process** (**all**) **begin** **case** pr\_state **is** **when** idle => **if** wr **and not** rd **then**  nx\_state <= WREN\_OP; **elsif** rd **and not** wr then  nx\_state <= READ\_OP;  **else**  nx\_state <= idle; **end if**; **when** WREN\_OP => **if** i=imax **then** nx\_state <= deselect1; **else** nx\_state <= WREN\_OP; **end if**; **when** deselect1 => nx\_state <= deselect2; **when** deselect2 => nx\_state <= WRITE\_OP; **when** WRITE\_OP => **if** i=imax **then** nx\_state <= initial\_wr\_addr; **else** nx\_state <= WRITE\_OP; **end if**; **when** initial\_wr\_addr => ...  **when** write\_data => ...  **when** wait\_wr\_low => ...  **when** READ\_OP => ...  **when** initial\_rd\_addr => ... **when** read\_data => ...  **when** wait\_rd\_low => ...  **end case**; **end process**; --Output logic (block L2' of figure 15.2b): **process** (**all**) **begin** --Default values: SSn <= '0'; SCLK <= clk\_12MHz; MOSI <= '-'; --Other values: **case** pr\_state **is**  **when** idle => SSn <= '1'; SCLK <= '0'; **when** WREN\_OP => MOSI <= WREN\_OPCODE(7-i); **when** deselect1 => SCLK <= '0';  **when** deselect2 => SSn <= '1'; SCLK <= '0'; **when** WRITE\_OP => MOSI <= WRITE\_OPCODE(7-i); **when** initial\_wr\_addr => MOSI <= INITIAL\_WRITE\_ADDR(15-i); **when** write\_data => MOSI <= test\_data\_out(j)(7-i); **when** wait\_wr\_low => SCLK <= '0';  **when** READ\_OP => MOSI <= READ\_OPCODE(7-i); **when** initial\_rd\_addr => MOSI <= INITIAL\_READ\_ADDR(15-i);  **when** wait\_rd\_low => SCLK <= '0'; **end case**; **end process**;  --Store data read from EEPROM: **process** (clk\_12MHz) **begin**  **if** rising\_edge(clk\_12MHz) **then**  **if** pr\_state=read\_data **then** test\_data\_in(j)(7-i) <= MISO; **end if**; **end if**; **end process**;   --Display data read from EEPROM (test circuit @1Hz): **process** (clk\_12MHz) **variable** count1: natural **range** 0 **to** 12\_500\_000; **variable** count2: natural **range** 0 **to** 4; **begin** **if** falling\_edge(clk\_12MHz) **then** **if** pr\_state=idle **then** **if** count1=12\_500\_000 **then**  count1 := 0; **if** count2 /= 4 **then** count2 := count2 + 1; **else**  count2 := 0; **end if**; **else** count1 := count1 + 1; **end if**; data\_read\_from\_EEPROM <= test\_data\_in(count2); **end if**; **end if**; **end process**; **end architecture**;----------------------------------------------------------------------------- |

**Example 17.3. I2C Interface for an A/D Converter (with Pointer)**

|  |  |
| --- | --- |
| 123456789101112131415161718192021222324252627282930313233343536373839404142434445464748495051525354555657585960616263646566676869707172737475767778798081828384858687888990919293949596979899100101102103104105106107108109110111112113114115116117118119120121122123124125126127128129130131132133134135136137138139140141142143144145146147148149150151 | ---------------------------------------------------------------------------**library** ieee;**use** ieee.std\_logic\_1164.**all**;**entity** I2C\_interface **is** **generic** ( SLAVE\_ADDRESS: std\_logic\_vector(6 **downto** 0) := "0110110"; SETUP\_REGISTER: std\_logic\_vector(7 **downto** 0) := "1101001-"; CONFIG\_REGISTER: std\_logic\_vector(7 **downto** 0) := "011---01"); **port** ( clk, rst: **in** std\_logic; wr, rd: **in** std\_logic; SCL: **out** std\_logic; SDA: **inout** std\_logic; received\_data: **out** std\_logic\_vector(9 **downto** 0));**end entity**;**architecture** fsm **of** I2C\_interface **is** **signal** clk\_400kHz: std\_logic; --reference clock **signal** i: natural **range** 0 **to** 34; --pointer **signal** wr\_enable, rd\_enable: std\_logic;**begin** --Generate 400kHz from 50MHz system clock: **process** (clk) **variable** count: natural **range** 0 **to** 62; **begin** **if** rising\_edge(clk) **then** **if** count=62 **then** clk\_400kHz <= not clk\_400kHz; count := 0; **else** count := count + 1; **end if**; **end if**; **end process**;  --Generate pointer and enable signals: **process** (clk\_400kHz, rst) **begin** **if** rst **then** wr\_enable <= '0'; rd\_enable <= '0'; i <= 0; **elsif** falling\_edge(clk\_400kHz) **then** **if** (wr **and not** rd\_enable) **or** wr\_enable **then** **if** i<30 **then** wr\_enable <= '1'; i <= i + 1; **elsif** **not** wr **then** wr\_enable <= '0'; i <= 0; **end if**; **elsif** (rd **and not** wr\_enable) **or** rd\_enable **then** **if** i<34 **then** rd\_enable <= '1'; i <= i + 1; **elsif not** rd **then** rd\_enable <= '0'; i <= 0; **end if**; **else** wr\_enable <= '0'; rd\_enable <= '0'; i <= 0; **end if**; **end if**; **end process**;  --Generate SCL and SDA signals: **process** (all) **begin**  **if** wr\_enable **then** --Define SCL for writing: **case** i **is** **when** 1 | 30 => SCL <= '1'; **when others** => SCL <= clk\_400kHz; **end case**; --Define SDA for writing: **case** i **is** --Start: **when** 1 => SDA <= '0'; --Slave address to write: **when** 2 to 8 => SDA <= SLAVE\_ADDRESS(8-i); **when** 9 => SDA <= '0'; **when** 10 => SDA <= 'Z'; --Setup register: **when** 11 **to** 18 => SDA <= SETUP\_REGISTER(18-i); **when** 19 => SDA <= 'Z'; --Configuration register: **when** 20 to 27 => SDA <= CONFIG\_REGISTER(27-i); **when** 28 => SDA <= 'Z'; --Stop: **when** 29 => SDA <= '0';  **when others** => SDA <= '1';  **end case**;  **elsif** rd\_enable **then** --Define SCL for reading: **case** i **is** **when** 1 | 34 => SCL <= '1'; **when** 11 **to** 14 => SCL <= '0'; **when others** => SCL <= clk\_400kHz; **end case**; --Define SDA for reading: **case** i **is** --Start: **when** 1 => SDA <= '0'; --Slave address to read: **when** 2 **to** 8 => SDA <= SLAVE\_ADDRESS(8-i); **when** 9 => SDA <= '1'; **when** 10 => SDA <= 'Z'; --Clock stretch: **when** 11 **to** 14 => SDA <= 'Z'; --Read result byte 1: **when** 15 **to** 22 => SDA <= 'Z';  **when** 23 => SDA <= '0'; --Read result byte 0:  **when** 24 **to** 31 => SDA <= 'Z'; **when** 32 => SDA <= '1'; --Stop: **when** 33 => SDA <= '0'; **when others** => SDA <= '1';  **end case**;  **else** SCL <= '1'; SDA <= '1'; **end if**; **end process**;  --Store data read from ADC: **process** (clk\_400kHz) **begin**  **if** rising\_edge(clk\_400kHz) **then**  **if** rd\_enable **then** **if** i=21 **then** received\_data(9) <= SDA; **elsif** i=22 **then** received\_data(8) <= SDA; **elsif** i>23 **and** i<32 **then** received\_data(31-i) <= SDA; **end if**; **end if**; **end if**; **end process**;  **end architecture**;--------------------------------------------------------------------------- |

**Example 17.6. VGA Video Interface for a Hardware-Generated Image**

|  |  |
| --- | --- |
| 123456789101112131415161718192021222324252627282930313233343536373839404142434445464748495051525354555657585960616263646566676869707172737475767778798081828384858687888990919293949596979899100101102103104105106107108109110111112113114115116117118119120121122 | ------------------------------------------------------------**library** ieee;**use** ieee.std\_logic\_1164.**all**;**entity** image\_generator\_plus\_vga\_interface **is**  **generic** ( H\_LOW: natural := 96; HBP: natural := 48; H\_HIGH: natural := 640; HFP: natural := 16; V\_LOW: natural := 2; VBP: natural := 33; V\_HIGH: natural := 480; VFP: natural := 10); **port** ( clk: **in** std\_logic; --50MHz system clock clk\_vga: **out** std\_logic; --25MHz pixel clock  R\_switch, G\_switch, B\_switch: **in** std\_logic; Hsync, Vsync: **out** std\_logic; R, G, B: **out** std\_logic\_vector(9 downto 0); BLANKn, SYNCn : **out** std\_logic);**end entity**;**architecture** rtl **of** image\_generator\_plus\_vga\_interface **is** **signal** Hactive, Vactive, dena: std\_logic;**begin** --CIRCUIT 1: CONTROL GENERATOR --Static signals for DAC: BLANKn <= '1'; --no blanking SYNCn <= '0'; --no sync on green --Create VGA clock (50MHz -> 25MHz): **process** (clk) **begin** **if** rising\_edge(clk) **then** clk\_vga <= not clk\_vga; **end if**; **end process**; --Create horizontal signals: **process** (clk\_vga) **variable** Hcount: natural **range** 0 **to** H\_LOW + HBP + H\_HIGH + HFP; **begin** **if** rising\_edge(clk\_vga) **then** Hcount := Hcount + 1; **if** Hcount = H\_LOW **then** Hsync <= '1'; **elsif** Hcount = H\_LOW + HBP **then** Hactive <= '1'; **elsif** Hcount = H\_LOW + HBP + H\_HIGH **then** Hactive <= '0'; **elsif** Hcount = H\_LOW + HBP + H\_HIGH + HFP **then** Hsync <= '0'; Hcount := 0; **end if**; **end if**; **end process**; --Create vertical signals: **process** (Hsync) **variable** Vcount: natural **range** 0 **to** V\_LOW + VBP + V\_HIGH + VFP; **begin** **if** rising\_edge(Hsync) **then** Vcount := Vcount + 1; **if** Vcount = V\_LOW **the**n Vsync <= '1'; **elsif** Vcount = V\_LOW + VBP **then** Vactive <= '1'; **elsif** Vcount = V\_LOW + VBP + V\_HIGH **then** Vactive <= '0'; **elsif** Vcount = V\_LOW + VBP + V\_HIGH + VFP **then** Vsync <= '0'; Vcount := 0; **end if**; **end if**; **end process**; --Enable diplay: dena <= Hactive **and** Vactive; --CIRCUIT 2: IMAGE GENERATOR **process** (**all**) **variable** line\_count: natural **range** 0 **to** V\_HIGH; **begin** **if** rising\_edge(Hsync) **then** **if** Vactive **then** line\_count := line\_count + 1; **else** line\_count := 0; **end if**; **end if**; **if** dena **then** **case** line\_count **is** **whe**n 0 => R <= (**others** => '1'); G <= (**others** => '0'); B <= (**others** => '0'); **when** 1 | 2 | 479 => R <= (**others** => '0'); G <= (**others** => '1'); B <= (**others** => '0'); **when** 3 **to** 5 => R <= (**others** => '0'); G <= (**others** => '0'); B <= (**others** => '1'); **when others** => R <= (**others** => R\_switch); G <= (**others** => G\_switch); B <= (**others** => B\_switch); **end case**; **else** R <= (**others** => '0'); G <= (**others** => '0'); B <= (**others** => '0');  **end if**; **end process**;**end architecture**;------------------------------------------------------------ |