

Digital System Design with PLDs and FPGAs

VHDL

Kuruvilla Varghese
DESE
Indian Institute of Science



Kuruvilla Varghese



Introduction / Features

- Evolution
 - VHDL – VHSIC Hardware Description Language
 - DOD, USA 1970's & 1980's
 - IEEE 1076.3
- Schematic
 - Connectivity Information (Net list)
 - Hierarchy, but bottom-up
 - Portability (Schematic database, Libraries)
 - Controller (FSM) Design external
 - Design Description
- Open standard
- Human readable
- Portability
- Documentation, Simulation, Synthesis
- Hierarchical design (top-down or bottom-up)
- Higher Level constructs
- Supports library based design
- Strict Type checking

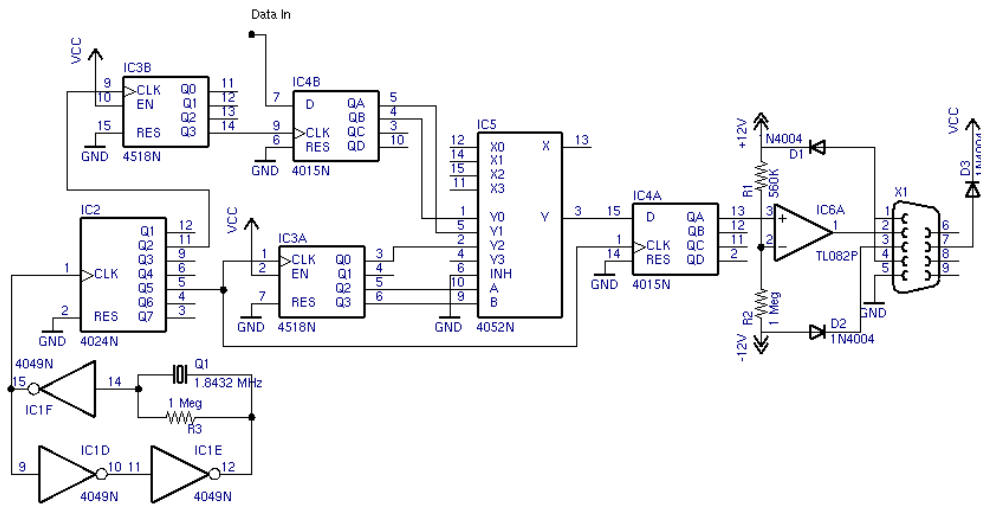


Kuruvilla Varghese



Schematic

3



source: linuxtoys.org



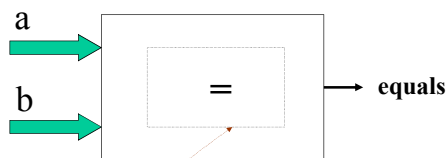
Kuruvilla Varghese



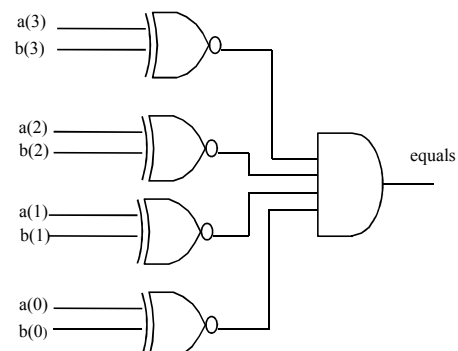
VHDL Features

4

- Entity – Interface Specifications
- Architecture – Functionality
- e.g. 4 bit comparator
- Function



Function



Kuruvilla Varghese



Equality Comparator

5

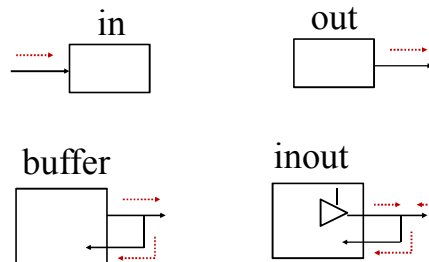
-- 4 bit equality comparator

```
library ieee;
use ieee.std_logic_1164.all;

entity eqcomp is
port (a, b: in std_logic_vector(3 downto 0);
      equals: out std_logic);
end eqcomp;

architecture arch_eqcomp of eqcomp is
begin
  equals <= '1' when (a = b) else '0';
end arch_eqcomp;
```

bit - '0', '1'
std_logic - '0', '1', 'Z', ...



buffer – has restrictions, we use signals (wires) for local feedback



Kuruvilla Varghese



Equality Comparator

6

- Comments start with -- anywhere on the line
- Library → Packages → Components, Functions, Procedures, Data Objects
- Mode
 - in, out, inout, buffer
- Range: downto, to (MSbit, LSbit)
 - Bit order, Byte order (Little Endian, Big Endian)
- Identifiers
 - Alphabetic, Numeric or underscore characters
 - Not case sensitive
 - The first character must be an alphabet
 - The last character cannot be an underscore
 - Two underscores in succession are not allowed



Kuruvilla Varghese



Syntax, Operators

7

- Architecture Body

- Architecture declaration
 - Component declarations
 - Type declarations
 - Constant declarations
 - Signal declarations
 - Function, Procedure definitions
- Architecture statement

- Logical Operators

- and, nand, or, nor, xor, xnor, not

These are defined for data type “bit” and “boolean”

For “std_logic” data type these operators are overloaded in “ieee.std_logic_1164” package



Kuruvilla Varghese



Operators

8

- Arithmetic Operators

- +, -, *, /
- ** (exponentiation)
- mod (modulo division)
- rem (modulo remainder)
- abs (absolute value)

$$A \text{ mod } B = A - B * N$$

$$A \text{ rem } B = A - \lfloor A / B \rfloor * B$$

- These operators are defined for “integer” and “real” data types

- For “std_logic” data type, these operators are overloaded in “ieee.std_logic_unsigned” package



Kuruvilla Varghese



Operators

9

- Relational Operators

=, >, <, <=, >=, /=

These operators are defined for “integer” and “real” data types
For “std_logic” data type, these operators are overloaded in “ieee.std_logic_arith” package

- Shift Operators

sll (shift left logical), srl
sla (shift left arithmetic), sra
rol (rotate left), ror

These operators are defined for “bit” and “boolean” data types
For “std_logic” data type, these operators are overloaded in “ieee.std_logic_arith” package



Kuruvilla Varghese



Operators

10

- Aggregate operator

Applied to elements of same type and size

```
signal a, b, c: std_logic;  
signal tmp: std_logic_vector(2  
    downto 0);  
tmp <= (a, b, c);
```

- Concatenation operator

Concatenate different size arrays of the same element type.

```
type byte is array (7 downto 0)  
    of bit;  
signal count: byte;  
count <= "010" & "00110";
```



Kuruvilla Varghese



Operators - precedence

11

1. Logical operators
2. Relational operators
3. Shift operators
4. Adding operators
5. Multiplying operators
6. Miscellaneous operators

Increasing precedence from 1 to 6
Operators of same category same precedence.
Left to right evaluations.
“not” operator has precedence 6



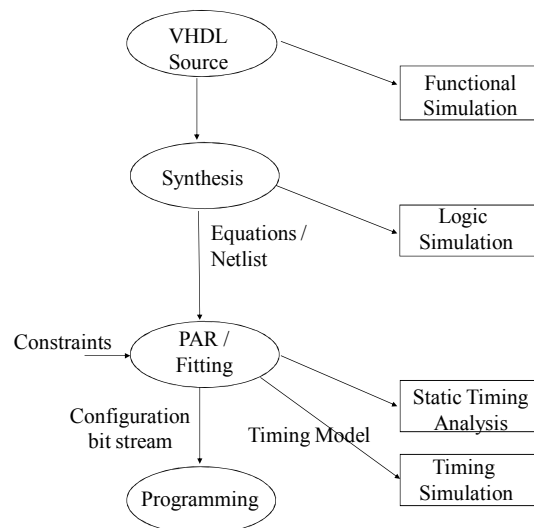
Kuruvilla Varghese



Multiple Architectures / Design Flow

12

- Single Entity, Multiple architectures
- Simulation / Synthesis
- Area / Speed
- FPGA / PLD



Kuruvilla Varghese



Design Flow - Tools

13

- VHDL Editor
- Synthesis Tool
- Constraint Editor
- Place and Route (PAR) / Fitting Tool
- VHDL Simulator
 - Functional/Behavioral simulation
 - Logic Simulation
 - Timing Simulation
- Static Timing Analysis Tool



Kuruvilla Varghese



Data flow Model

14

-- 4 bit equality comparator

```
library ieee;  
use ieee.std_logic_1164.all;  
  
entity eqcomp is  
port (a, b: in std_logic_vector(3  
    downto 0);  
    equals: out std_logic);  
end eqcomp;
```

```
architecture arch_eqcomp of eqcomp  
is  
begin  
    equals <= '1' when (a = b) else '0';  
end arch_eqcomp;
```



Kuruvilla Varghese



Data flow Model

15

-- 4 bit equality comparator

```
library ieee;
use ieee.std_logic_1164.all;

entity eqcomp is
port (a, b: in std_logic_vector(3
    downto 0);
    equals: out std_logic);
end eqcomp;
```

```
architecture arch_eqcomp of
    eqcomp is
begin
    equals <= (a(3) xnor b(3)) and
        (a(2) xnor b(2)) and
        (a(1) xnor b(1)) and
        (a(0) xnor b(0)) ;
end arch_eqcomp;
```

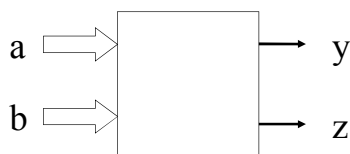


Kuruvilla Varghese



Concurrency

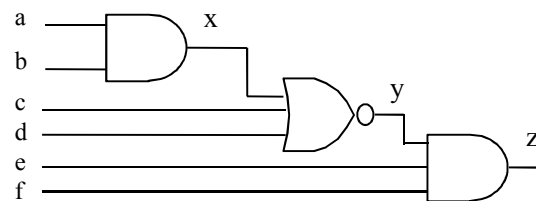
16



$$y = f_1(a, b)$$

$$z = f_2(a, b)$$

Both Y and Z are concurrently active (hardware)



```
z <= e and f and y;
```

```
y <= c nor d nor x;
```

```
x <= a and b;
```

Simulators has to resolve the concurrency from the sequentially written code. Not a problem for Synthesis tools



Kuruvilla Varghese



Behavioral Model

17

```
library ieee;
use ieee.std_logic_1164.all;

entity eqcomp is
port (a, b: in std_logic_vector(3
    downto 0);
    equals: out std_logic);
end eqcomp;
```

```
architecture arch_eqcomp of
    eqcomp is
begin
    eqproc: process (a, b)
begin
    if (a = b) then
        equals <= '1';
    else
        equals <= '0';
    end if;
end process;
end arch_eqcomp;
```



Kuruvilla Varghese



Process

18

- Sequential body
 - The way simulator computes
 - Synthesis is based on the statements
- Process Body
 - Process declarative part
 - Variable, Constant declarations
 - Type declaration
 - Function, Procedure definitions
 - Process statement part
- Higher level constructs
 - if ... then,
 - case ... when
 - for ... loop
 - while ... loop



Kuruvilla Varghese



Process

19

```
eqproc: process (a)
begin
  if (a = b) then
    equals <= '1';
  else
    equals <= '0';
  end if;
end process eqproc;
```

- Sensitivity list
 - Compatibility between simulation model and synthesis model (sensitivity list – RHS of assignments, and Conditions)
 - Focus of synthesis tool is the structure of the circuit, not the real time behavior of the circuit. Simulation tool focuses on latter

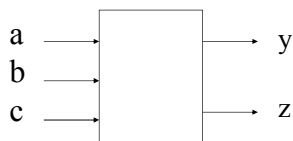


Kuruvilla Varghese



Using Process

20



```
process (a,b,c)
begin
  ....
  y <= f1 (a,b,c)
  ...
  z <= f2 (a,b,c)
end process;
```

* Note: Little more detail is required

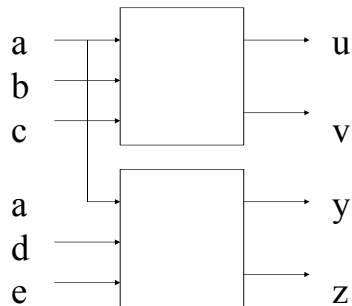


Kuruvilla Varghese



Multiple processes, concurrency

21



```
eqproc: process (a, b, c)  
begin
```

...

```
end process eqproc;
```

```
eqproc: process (a, d, e)  
begin
```

...

```
end process eqproc;
```

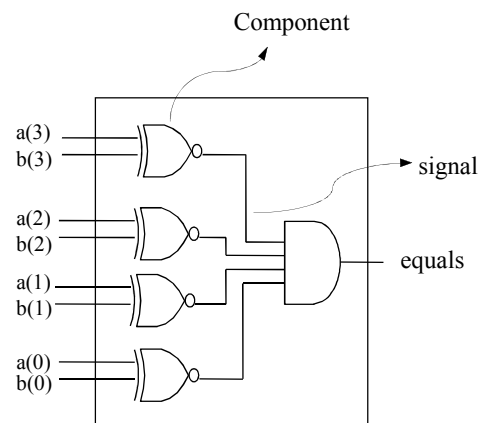
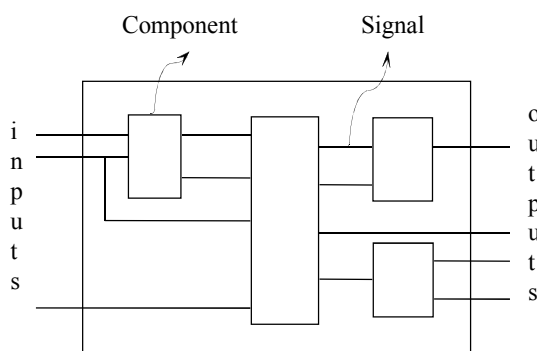


Kuruvilla Varghese



Hierarchy

22



Kuruvilla Varghese



Structural Code

23

```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity eqcomp is
port (a, b: in std_logic_vector(3 downto 0);
      equals: out std_logic);
end eqcomp;
```

```
architecture arch_eqcomp of eqcomp is
```

```
    component xnor2
        port (i1, i2: in std_logic, o1: out
              std_logic);
    end component;
    component and4
        port (i1, i2, i3, i4: in std_logic, o1: out
              std_logic);
    end component;
```

```
    signal int1, int2, int3, int4: std_logic;
begin
```



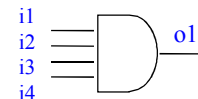
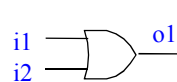
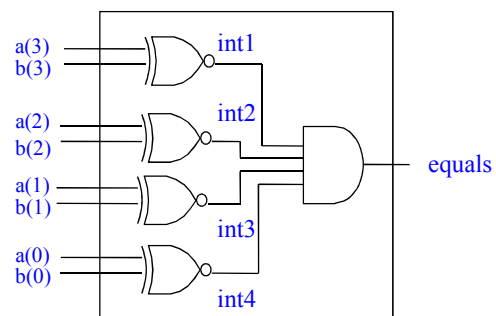
Kuruvilla Varghese



Structural Code

24

```
c1: xnor2 port map (a(3), b(3), int1);
c2: xnor2 port map (a(2), b(2), int2);
c3: xnor2 port map (a(1), b(1), int3);
c4: xnor2 port map (a(0), b(0), int4);
c5: and4 port map (int1, int2, int3, int4,
                  equals);
end arch_eqcomp;
```



Kuruvilla Varghese



Components

25

```
-- Components
library ieee;
use ieee.std_logic_1164.all;

entity xnor2 is
  port (i1, i2: in std_logic, o1: out
        std_logic);
end xnor2;

architecture arch_xnor2 of xnor2 is
begin
  o1 <= i1 xnor i2;
end arch_xnor2;

library ieee;
use ieee.std_logic_1164.all;

entity and4 is
  port (i1, i2, i3, i4: in std_logic, o1:
        out std_logic);
end and4;

architecture arch_and4 of and4 is
begin
  o1 <= i1 and i2 and i3 and i4;
end arch_and4;
```



Kuruvilla Varghese



Component instantiation

26

- Port map
 - Formal to actual mapping
- Positional association
- Named Association

```
xnor2 port map (o1 => int1, i2
=> b(3), i1 => a(3));
```



Kuruvilla Varghese



Naming signals, ports

27

```
signal int1, int2, int3, int4: std_logic;
```

```
↓  
signal int: std_logic_vector(3 downto 0);
```

```
c1: xnor2 port map (a(3), b(3), int1);  
c2: xnor2 port map (a(2), b(2), int2);  
c3: xnor2 port map (a(1), b(1), int3);  
c4: xnor2 port map (a(0), b(0), int4);  
c5: and4 port map (int1, int2, int3, int4,  
    equals);
```

```
↓  
c1: xnor2 port map (a(3), b(3), int(3));  
c2: xnor2 port map (a(2), b(2), int(2));  
c3: xnor2 port map (a(1), b(1), int(1));  
c4: xnor2 port map (a(0), b(0), int(0));  
c5: and4 port map (int(3), int(2), int(1), int(0),  
    equals);
```

```
for i in 0 to 3 generate
```

```
    c: xnor2 port map (a(i), b(i), int(i));  
end generate;
```

```
c4: and4 port map (int(0), int(1), int(2),  
    int(3), equals);
```

- open

when a component is instantiated if any of its output is unused, those can be mapped to 'open'

unused inputs must be tied to appropriate logic levels



Kuruvilla Varghese



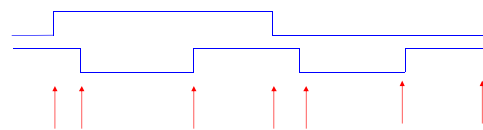
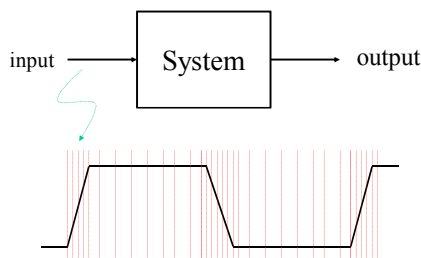
Simulation

28

• Types of Simulation

– Trace Simulation

- Steps
- Analog



• Digital

– Event driven simulation

- Events, trigger computation
- Events on inputs / internal signals

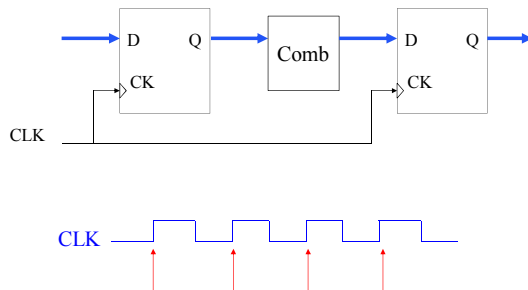


Kuruvilla Varghese



Simulation

29



• Sequential Circuit

- Cycle based simulation
 - Simulated every clock cycle

• Simulation Time

- Event time at which computation happens
- Events are ordered chronologically

• Simulation Cycle

- Resolving concurrency, by sequential computation
- Delta delay

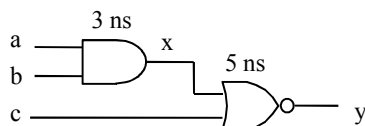


Kuruvilla Varghese



Simulation Cycle - Timing

30



```
architecture dflow of ckt1 is
begin
  y <= c nor x after 5 ns;
  x <= a and b after 3 ns;
end dflow;
```

Time(ns)	a	b	c	x	y
0	1	1	0	1	0
100	0	1	0		
100 + 3				0	0
100 + 3 + 5					1
108	0	1	0	0	1

• Issue 1: Order of statements

- Resolving concurrency – order of concurrent statements may not match the data flow.
- Solution: Event driven computation
- Feedback
- Solution: Keep computing till stable.

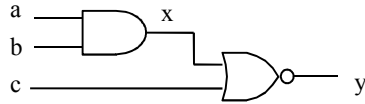


Kuruvilla Varghese



Simulation Cycle – Functional/Logic

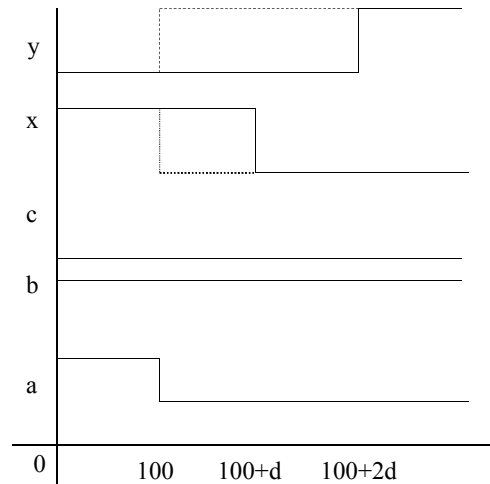
31



```

architecture dflow of ckt1 is
begin
  y <= c nor x;
  x <= a and b;
end dflow;
  
```

Time(ns)	a	b	c	x	y
0	1	1	0	1	0
100	0	1	0		
100 + d				0	0
100 + 2d					1
100	0	1	0	0	1



Kuruvilla Varghese



Logic Simulation

32

- Issue 2: Functional/ Logic simulation
 - Elements have zero delay
 - Solution: delta delay for resolving sequence.
- Events
 - Events (on external signals and one happens on internal signals due to computation) are ordered in simulation time and are handled in the same order.
- How small should be the delta delay (for simulator implementation)?
 - Smaller than the smallest time delay in the circuit,
 - Smaller than smallest change in any input signal

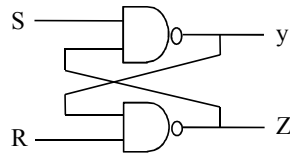


Kuruvilla Varghese



Simulation Cycle - Feedback

33



architecture dflow of bff is

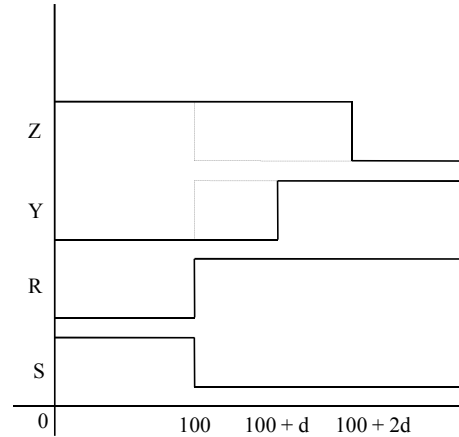
begin

Z <= R nand Y;

Y <= S nand Z;

end dflow;

Time(ns)	S	R	Y	Z
0	1	0	0	1
100	0	1		
100 + d			1	
100 + 2d				0
100	0	1	1	0



Solution: Event driven computation
Keep computing till stable.

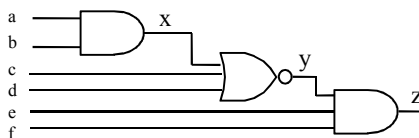


Kuruvilla Varghese



Only inputs in sensitivity list?

34



process (a, b, c, d, e, f)

begin

....

x <= f₁ (a, b);

...

y <= f₂ (c, d, x);

...

z <= f₃ (e, f, y);

end process;

- In response to an event on any of the inputs (in sensitivity list), process computes from top to bottom once.
- Even if the order of assignments match the data flow, each assignment use value of inputs at current simulation time, though the outputs are assigned after a delta cycle.
- i.e. x is assigned after first delta cycle. In the next statement value of x used is that of current simulation time.
- To force a process computation, in response to assignment of values to internal signal, internal signals should be included in the sensitivity list.



Kuruvilla Varghese



Process – Correct usage

35

```
process (a, b, c, d, e, f, x, y)
begin
  ....
  x <= f1 (a, b);
  ...
  y <= f2 (c, d, x);
  ...
  z <= f3 (e, f, y);
end process;
```



Kuruvilla Varghese



Process – Concurrent statements

36

- A concurrent statement is equivalent to a process with signals in the RHS of the concurrent statement, in the process sensitivity list.
- When a combinational block is coded in a single process, all the input signals and all the internal signals should be in the sensitivity list.
- This is required as the assignment happens at $(t + \delta)$ time for an event at 't' and the subsequent statements use the values of internal signals at time 't'.
- In order for these internal signal assignments to trigger the process computation, internal signals should be in the sensitivity list. This is true even if the order of statements are in the order of data flow.
- Concurrent statements are equivalent to process with signals on the RHS of assignments in the sensitivity list.
- Similarly a process could be equivalently written with multiple concurrent statements
- In real cases, multiple concurrent statements and multiple processes works concurrently, responding to various events on inputs or internal signals.



Kuruvilla Varghese



Synthesis

37

- `eql <= '1' when (a = b) else '0';`

a3	a2	a1	a0	b3	b2	b1	b0	eql
0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0
1	1	1	1	1	1	1	1	1

Exponential Complexity

Width 4 $\rightarrow$ 2 exp 8 terms

Width 8 $\rightarrow$ 2 exp 16 terms

Operator Inferencing = ex-or / ex-nor

`y <= a + b;`

`y <= x + 1;`



Kuruvilla Varghese



Data Objects

38

- **Classes**

Constants, Signals, Variables, Files

- **Syntax**

`class object_name: data type;`

- **Signals**

Signals declared in architecture declarative region and used anywhere (architecture, processes, functions, procedures)

`signal carry: std_logic_vector(7 downto 0);`

- **Constants**

– For readability and easy modification of code

`constant width: integer := 8;`

- **Variables**

– Declared in sequential bodies (process, functions, procedures).

– Used in simulation extensively. Indexing, temporary storage etc.. Synthesis is not well defined in non-trivial cases.

`variable count: integer range 0 to 255;`



Kuruvilla Varghese



Data Objects, Types

39

- Files

Used in test benches to store input vectors and to write output vectors.
Screen output

```
type logic_data is file of character;  
file logical_name: logic_data;  
file logical_name: logic_data is  
  "inpvec.dat";  
file logical_name: logic_data open  
  read_mode is "inpvec.dat";
```

- Data Types

- Scalar: enumerated, integer, float
- Composite: array, record

- Enumerated

```
type state_type (init, sample, wait,  
interrupt, out);
```

- Normally takes the values 0,1,2,3,4, ..
- Could be changed using attributes



Kuruvilla Varghese



Data Types - Scalar

40

- Enumerated

```
type boolean is (FALSE, TRUE);  
type bit is ('0', '1');
```

```
type std_ulogic is  
( 'U', -- Un-initialized  
  'X', -- Forcing Unknown  
  '0', -- Forcing 0  
  '1', -- Forcing 1  
  'Z', -- High Impedance  
  'W', -- Weak Unknown  
  'L', -- Weak 0  
  'H', -- Weak 1  
  '-' -- Don't care );
```

```
subtype std_logic is resolved  
std_ulogic;
```

- Synthesis: '0', '1', 'L', 'H', 'Z', '-'
- Simulation: All except '-'



Kuruvilla Varghese



Data Types, Scalar Predefined

41

- Integer

type integer is range -2147483647 to 2147483647;

Range

variable count: integer range 0 to 255;

constant width: integer := 16;

- Floating types

type real is range -1.0E38 to +1.0E38

- Physical Types

type time is range -2147483647 to 2147483647

units

fs;

ps = 1000 fs;

ns = 1000 ps;

us = 1000 ns;

ms = 1000 us;

sec = 1000 ms;

min = 60 sec;

hr = 60 min;

end units;



Kuruvilla Varghese



Subtypes

42

- Subtype

– subtype my_int is integer range 48 to 56;

– subtype “UX01” is resolved std_ulogic range ‘U’ to ‘1’;

- Type and subtype

– subtype my_int is integer range 48 to 56;

– type my_int is range 48 to 56;

- What is the difference between the above two?

– In the first one, all the operators defined for integer works. In the second case various operators need to be overloaded for the new my_int type.



Kuruvilla Varghese



User defined Data Types

43

- User defined

- type MVL is ('U', 'O', '1', 'Z');
- type index is range 0 to 15;
- type word_length is range 31 downto 0;
- type volt_range is 3.3 downto 1.1;
- type current is range 0 to 1E9 units
 - nA;
 - uA = 1000 nA;
 - mA = 1000 uA;
 - amp = 1000 mA;
- end units;
- subtype filter_current is current range 10 uA to 5 mA;

- Array Types

type word is array (15 downto 0) of bit;
signal address: word;

- Unconstrained Array (constrained at the time of object declaration)

type bit_vector is array (natural range <>) of bit;
type std_logic_vector is array (natural range <>) of std_logic;
signal a: std_logic_vector(3 downto 0);



Kuruvilla Varghese



Data Types - Composite

44

type table8x4 is array (0 to 7, 0 to 3) of std_logic;

constant ex_or: table8x4 :=

("000_0", "001_1",
"010_1", "011_0",
"100_1", "101_0",
"110_0", "111_1");

- Record Types

type iocell is record

buffer_inp: std_logic_vector(7 downto 0);

enable: std_logic;

buffer_out: std_logic_vector(7 downto 0);

end record;

signal busa, busb, busc: iocell;

signal vec: std_logic_vector(7 downto 0);



Kuruvilla Varghese



Data Types - Composite

45

```
busa.buffer_inp <= vec;  
busb.buffer_inp <= busa.buffer_inp  
busb.enable <= '1';  
busc <= busb;
```

```
busa.buffer_out <= busa.buffer_inp  
  when (busa.enable = '1') else  
  (others => 'Z');
```

- Alias

```
signal address: std_logic_vector(31  
  downto 0);
```

```
alias top_ad: std_logic_vector(3  
  downto 0) is address(31 downto 28);
```

- Array assignments

```
signal row: std_logic_vector(7 downto  
  0);  
row <= ('1', '0', '1', '1', '1', '1', '1', '1');  
row <= (7 => '1', 6 => '0', others => '1');  
row <= "10111111"  
row <= ('1', '0', others => '1');  
row <= X"BF"  
row <= (others => '0');  
row <= (others => 'Z');
```

Base: Hexadecimal – X,
Octal – O,
Binary – B



Kuruvilla Varghese



Concurrent statements

46

- with-select-when
- when-else

output_signal: output(s),
sel: input
signals in expra, exprb, ..., exprx: inputs

- with-select-when

```
with sel select  
output_signal <= expra when choices,  
  exprb when choices,  
  .....  
  exprx when others;
```

- All values of signal 'sel' must be listed
- Values must be mutually exclusive
- Truth table



Kuruvilla Varghese



with ... select

47

with a select

```
y <= '0' when "00",  
      '0' when "01",  
      '0' when "10",  
      '1' when "11",  
      '0' when others;
```

with a select

```
y <= b when "00",  
      not(b) when "01",  
      c when "10",  
      b when "11",  
      c when others;
```

- Truth Table

a(1)	a(0)	y
0	0	0
0	1	0
1	0	0
1	1	1

$y = a(1) \text{ and } a(0)$



Kuruvilla Varghese



with ... select

48

Truth Table

a(1)	a(0)	b	c	y
0	0	0	x	0
0	0	1	x	1
0	1	0	x	1
0	1	1	x	0
1	0	x	0	0
1	0	x	1	1
1	1	0	x	0
1	1	1	x	1

Equation

$y = a(1) \text{ and } a(0) \text{ and } b \text{ or } a(1) \text{ and } a(0) \text{ and } b \text{ or } a(1) \text{ and } a(0) \text{ and } c \text{ or } a(1) \text{ and } a(0) \text{ and } b$



Kuruvilla Varghese



with - select

49

- For all the mutually exclusive values of an input signal ('select' signal) or signals, output is expressed, sometime as function of other inputs
- In the simplest case, when output values are specified, it is a plain truth table and the choices specify the minterms of input signals
- When output is expressed as a function of other inputs for a choice of 'select' signal, that may expand to multiple rows of truth table.
- It is very easy to work out the equations from the descriptions, each choice forming a minterm or 'OR' of minterms (in case if an expression is used).
- For Simulator, event on any input signal (select signal, signals in expression) would trigger a computation of the output signal
- Synthesis tool may not use the truth table, if the standard operators/structures could be inferred from the code



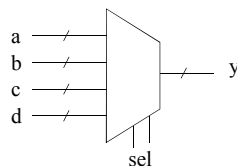
Kuruvilla Varghese



With-select-when

50

4-to-1 Multiplexer



Equations

$$y(i) = a(i) \text{ and sel}(1) / \text{ and sel}(0) / \text{ or} \\ b(i) \text{ and sel}(1) / \text{ and sel}(0) \text{ or} \\ c(i) \text{ and sel}(1) \text{ and sel}(0) / \text{ or} \\ d(i) \text{ and sel}(1) \text{ and sel}(0)$$

```
with sel select
y <= a when "00",
    b when "01",
    c when "10",
    d when "11",
    d when others;
```

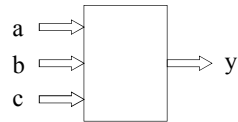


Kuruvilla Varghese



With-select-when

51



with b select

```
y <= func1(a, c) when value_1,  
      func2(a, c) when value_2,  
      .....  
      funci(a, c) when value_i,  
      .....  
      funcn(a, c) when others;
```

- Example - choices

with numb select

```
prime <= '1' when "010" | "011" | "101"  
          | "111",  
          '0' when others;
```

- Combinational circuit with no priority



Kuruvilla Varghese



when - else

52

- Syntax

```
output_signal <= expa when cond1 else  
                expb when cond2 else  
                .....  
                expx when condx else  
                expy;
```

output_signal: output(s),
condi: condition in terms of inputs
expi: expression in terms of inputs

- General conditions

e.g. condition1: (a = b)
condition2: (d > 3)

- Priority

- Truth Table ?
- Yes, much more abstract



Kuruvilla Varghese



when - else

53

```
output_signal <= expa when cond1 else
    expb when cond2 else
    .....
    expx when condx else
    expy;
```

- Equations

```
output_signal =
    expa and cond1 or
    expb and cond2 and not(cond1) or
    expc and cond3 and not(cond2)
    and not(cond1) or
    .....
```



Kuruvilla Varghese



when - else

54

```
y <= a when (p > q) else
    b when (r = 2) else
    c;
```

p(1)	p(0)	q(1)	q(0)	r(1)	r(0)	y
0	0	0	0	0	0	c
0	0	0	0	0	1	c
0	0	0	0	1	0	b
0	1	0	0	x	x	a
0	1	1	0	0	0	c
0	1	1	0	0	1	c
0	1	1	0	1	0	b
1	1	1	1	1	1	c



Kuruvilla Varghese



when - else

55

```
y <= a when (p > q) else
  b when (r = 2) else
  c;
```

- In the code above, first condition translates to all those values of p and q for which (p > q). i.e. it translates to multiple rows of the truth table. In this case, signal 'r' is a don't care
- When it comes to second condition, it translates to all those values of p, q and r for which p <= q and r = 2. Once again, it means multiple rows of the truth table.

- For the simulator, an event on any of the signals in conditions or expressions will trigger the computation of the output signal.
- Synthesis tool may not use the truth table, if the standard operators/structures could be inferred from the code

```
prio <= "00" when (a = '1') else
  "01" when (b = '1') else
  "10" when (c = '1') else
  "11";
```

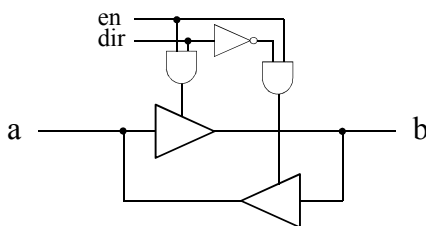


Kuruvilla Varghese



when - else

56



```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity transc is port
(a, b: inout std_logic_vector(7 downto 0);
en, dir: in std_logic);
end transc;
```

architecture dflow of transc is
begin

```
b <= a when (dir = '1' and en = '1') else
  (others => 'Z');
```

```
a <= b when (dir = '0' and en = '1') else
  (others => 'Z');
```

end dflow;



Kuruvilla Varghese



Sequential Statements

57

- if-then-else
 - case-when
 - if-then-else – syntax 1
 - Equation
- a, b, signals of *cond1*: inputs
y: output
- $$y = a \text{ and } cond1 \text{ or } b \text{ and not}(cond1)$$

```
if cond1 then
  y <= a;
else
  y <= b;
end if;
```

Note: *cond1* here means the Boolean equation of the condition.

Note: Sequential statements are used in process, functions and procedures only



Kuruvilla Varghese



if-then-else

58

- General conditions
- Priority
- Syntax 2
- Equations

```
if cond1 then
  y <= a;
elsif cond2 then
  y <= b;
elsif cond3 then
  y <= c;
else
  y <= d;
end if;
```

$$y = a \text{ and } cond1 \text{ or } b \text{ and } cond2 \text{ and not}(cond1) \text{ or } c \text{ and } cond3 \text{ and not}(cond2) \text{ and not}(cond1) \text{ or } d \text{ and not}(cond3) \text{ and not}(cond2) \text{ and not}(cond1)$$

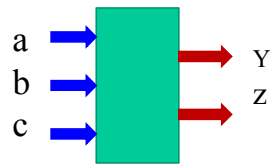

Kuruvilla Varghese



if-then-else

59

- Equivalent to when-else, but
- Multiple outputs
- Nesting



```
if cond1 then
  y <= a; z <= a and b;
elsif cond2 then
  y <= b; z <= c;
elsif cond3 then
  y <= c; z <= a;
else
  y <= d; z <= b;
end if;
```



Kuruvilla Varghese



if-then-else

60

- More complex behaviour/structure can be specified by nesting. E.g. if there are multiple outputs and we may not be able to specify all outputs for same conditions

- Equations

$y = a \text{ and } cond1 \text{ and } cond2 \text{ or } \dots$

```
if cond1 then
  if cond2 then
    y <= a;
  elsif
    .....
  end if;
elsif
  .....
end if;
```



Kuruvilla Varghese



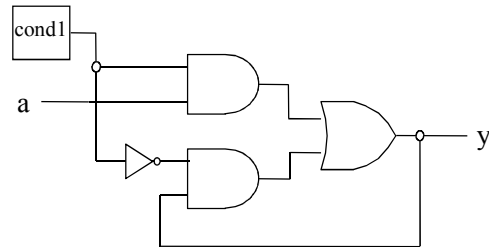
if-then-else

61

```
if cond1 then
  y <= a;
end if;

→

if cond1 then
  y <= a;
else
  y <= y;
end if;
```



- Implied Memory / Inferred latch



Kuruvilla Varghese



Implied Memory / Inferred latch

62

- Concurrent equivalents

```
with en select
  y <= a when '1';
with en select
  y <= a when '1',
    unaffected when others;
y <= a when en = '1';
y <= a when en = '1' else
  unaffected;
```
- Concurrent: unaffected
- Sequential: null
- Implied Memory / Inferred latch is useful in specifying the behaviour of latches and flip-flops or registers
- But, unintentional implied latches can happen
 - e.g. when multiple outputs are specified for each conditions, a missing output can result in implied latch on that output. This is all the more possible when nested loops are not balanced, as it is difficult to detect
- This is one of the common errors that inexperienced designer commit in VHDL coding



Kuruvilla Varghese



Implied Memory / Inferred latch

63

- It is difficult to expose this error in simulation, as just verifying all conditions would not be sufficient.
- Suppose, one output was missing in condition 3, and the previous condition simulated has the same value expected of this output in condition 3, then output will be correct.
- If the designer has inadvertently missed specifying an output, then working out the condition for exposing such an error would be remote.



Kuruvilla Varghese



Case-when

64

- Syntax

```
case sel_signal is
  when value1 =>
    (statements)
  when value2 =>
    (statements)
  .....
  when valuex =>
    (statements)
  when others =>
    (statements)
```
- All mutually exclusive values of sel_signal need to be specified
- No priority, Truth table
- Equivalent to with-select, but
- Multiple Outputs
- Nesting



Kuruvilla Varghese



Case-when

65

```
case sel is
  when val1 =>
    x <= a;
    y <= b;
  when val2 =>
    x <= c;
    y <= d;
  when val3 =>
    .....
end case;
```

- Equations
x = a and (decode of sel = val1) or
c and (decode of sel = val2) or ...
y = b and (decode of sel = val1) or
d and (decode of sel = val2) or ...
- Implied memory, if any output is not specified in all choices of selection signal.



Kuruvilla Varghese



Case-when Nesting

66

- “case ... when ...” can be nested with “if .. then ..” to specify complex structure / behavior.

```
case sel is
  when val1 =>
    if cond2 then
      y <= a;
    elsif
      .....
    end if;
  when val2 =>
    ...
end case;
```

- Equation

y = a and decode of (sel = val1)
and cond2 or ...

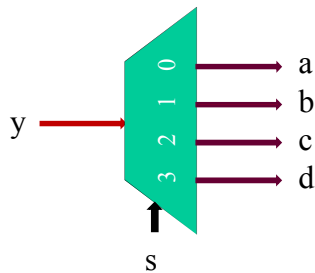


Kuruvilla Varghese



Case-when

67



```
library ieee;
use ieee.std_logic_1164.all;

entity dmux1t4 is
  port (y: in std_logic_vector(3 downto 0);
        s: in std_logic_vector(1 downto 0);
        a, b, c, d: out std_logic_vector(3
          downto 0));
end dmux1t4;
```

```
architecture arch_dmux1t4 of dmux1t4 is
begin
```



Kuruvilla Varghese



Demultiplexer

68

```
process (s, y)
begin
  case s is
    when "00" =>
      a <= y; b <= "0000"; c <= "0000";
      d <= "0000";
    when "01" =>
      b <= y; a <= "0000"; c <= "0000";
      d <= "0000";
    when "10" =>
      c <= y; a <= "0000"; b <= "0000";
      d <= "0000";
    when "11" =>
      d <= y; a <= "0000"; b <= "0000";
      c <= "0000";
    when others =>
      a <= "0000"; b <= "0000"; c <= "0000";
      d <= "0000";
  end case;
end process;
end arch_dmux1t4;
```



Kuruvilla Varghese



Demultiplexer

69

```
process (s, y)
begin
  a <= "0000"; b <= "0000";
  c <= "0000"; d <= "0000";
  case s is
    when "00" =>
      a <= y;
    when "01" =>
      b <= y;
    when "10" =>
      c <= y;
    when "11" =>
      d <= y;
  when others =>
    a <= "0000"; b <= "0000";
    c <= "0000"; d <= "0000";
  end case;
end process;
end arch_dmux1t4;
```



Kuruvilla Varghese



Loops

70

- Concurrent: generate
- Sequential: loop
- Generate
 - Equations
 - Component Instantiations

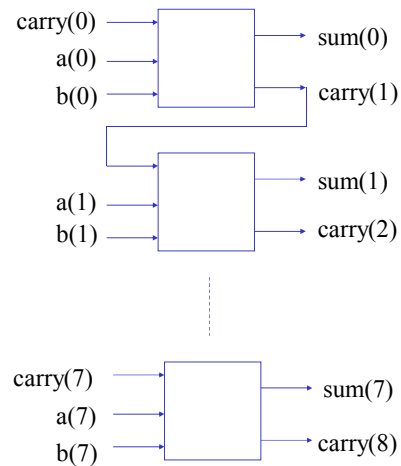
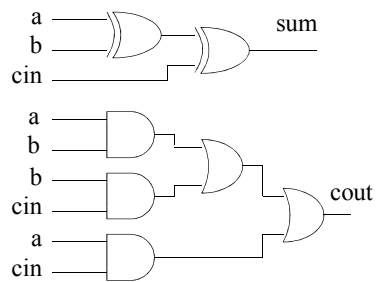


Kuruvilla Varghese



Generate - Example

71

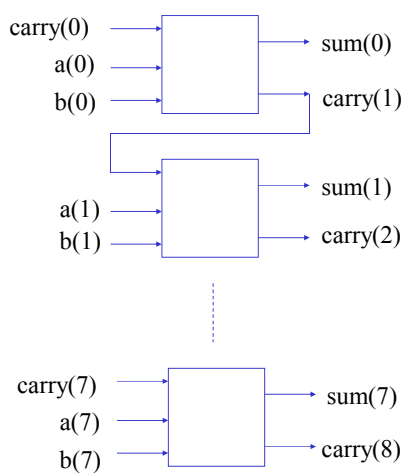


Kuruvilla Varghese



Generate

72



- Equations

for i in 0 to 7 generate

sum(i) <= a(i) xor b(i) xor carry(i);

carry(i+1) <= (a(i) and b(i)) or ((a(i) or b(i)) and carry(i));

end generate;

- Component Instantiations

for i in 0 to 7 generate

u1: fulladd port map (carry(i), a(i), b(i),
sum(i), carry(i+1));

end generate;



Kuruvilla Varghese



Conditional Loops

73

- if ... Generate (Concurrent statement, no else /elsif)

```
loop_label: if (condition / expression)
generate
.....
.....
end generate;
```

- If the condition is true, then the generate is done
- Useful to generate irregular structures, i.e. Conditional on loop index (e.g. $i = 2$) different structures can be generated



Kuruvilla Varghese



Sequential: For ... Loop

74

```
if (rst = '1') then
  for i in 0 to 7 loop
    fifo(i) <= (others => '0');
  end loop;
else
  .....
```

- The code decides whether the loop is synthesizable. Above loop is easily synthesizable.

- Syntax

```
loop_label: while expression loop
.....
.....
end loop;
```

- Example

```
tbloop: while not endfile(vector_file)
loop
.....
.....
```



Kuruvilla Varghese



Loop Control

75

- Exit

```
exit;  
exit [loop_label];  
exit [loop_label] when condition;
```

```
if condition then  
    exit;  
end if;
```

- Next

```
next;  
next [loop_label];  
next [loop_label] when condition;
```

```
if condition then  
    next;  
end if;
```

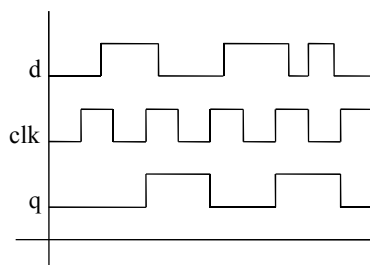
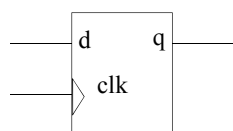


Kuruvilla Varghese



Sequential Circuits: D Flip Flop

76



- Flip Flops behavior is modeled in VHDL, not their equivalent circuit.
- Behavior: Up on the active edge of the clock the input is transferred to the output and is held (Memory) until the next active clock edge.



Kuruvilla Varghese



Flip Flop - Simulator

77

```
process (clk)
begin
  if (clk = '1') then
    q <= d;
  end if;
end process;
```

- 'clk' in the sensitivity list computes the process on both the edges of clocks.
- The clause (clk = '1') selects the positive edge of the clock.
- The Implied memory / Inferred latch takes care of memory



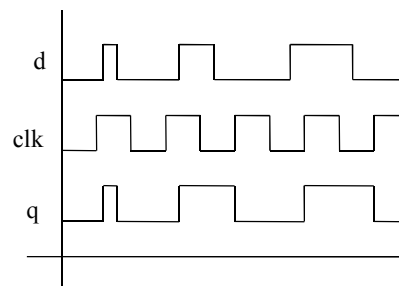
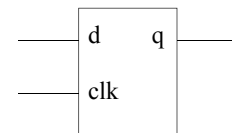
Kuruvilla Varghese



Flip Flop - Simulator

78

- But synthesis tools ignore sensitivity list, hence the above code would mean a (transparent) latch, as the events on 'clk' is not considered. Whenever 'clk' is high 'q' gets 'd' and on the negative edge the last value of 'q' is held. (Memory)



Kuruvilla Varghese



FF – Synthesis, Simulation

79

```
process (clk)
begin
  if (clk'event and clk = '1') then
    q <= d;
  end if;
end process;
```

- `clk'event` is a predefined attribute which is true whenever there is an event on the signal 'clk'. The statement `clk'event and clk = '1'`, would mean the positive edge of clock. Statement `clk'event` would be redundant for simulation.
- The statement `clk'event and clk = '1'`, would also be true when clk transits from 'U' to '1' (`std_logic`), this could occur at the beginning of simulation.



Kuruvilla Varghese



FF – Synthesis, Simulation

80

- To avoid this, functions 'rising_edge(clk)' and 'falling_edge(clk)' in `ieee.std_logic_1164` package could be used
- Equivalent concurrent statement

```
q <= d when clk'event and clk = '1';
```

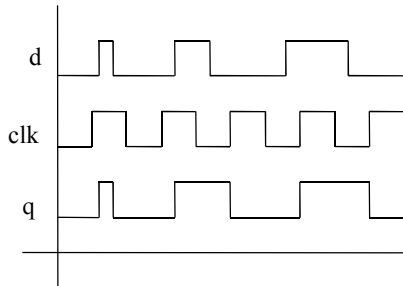


Kuruvilla Varghese



D Latch

81



Synthesis Tool

```
process (clk)
begin
  if (clk = '1') then
    q <= d;
  end if;
end process;
```

```
process (clk, d)
begin
  if (clk = '1') then
    q <= d;
  end if;
end process;
```



Kuruvilla Varghese



D Latch

82

- 'clk' in the sensitivity list invokes process for computation only on clock edges, hence 'd' is added to sensitivity list to respond to changes in the input 'd'
- The statement `clk = '1'` takes care of the transparent latch behavior.
- Implied memory takes care of the latch operation
- Equivalent concurrent statement

```
q <= d when clk = '1';
```

- wait

```
process
begin
  if (clk = '1') then
    q <= d;
  end if;
  wait on clk, d;
end process;
```



Kuruvilla Varghese



Wait

83

- wait on *sensitivity-list*;
- wait until *boolean-expression*;
- wait for *time-expression*;

- Examples

```
wait on clk, d;  
wait until count = 10;  
wait for 10 ns;  
wait on clk for 17 ns;  
wait until sum > 100 for 50 ns;
```

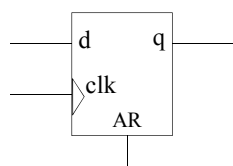


Kuruvilla Varghese



Asynchronous Reset

84



```
process (clk)  
begin  
  if (clk'event and clk = '1') then  
    q <= d;  
  end if;  
end process;
```

```
process (clk, reset)  
begin  
  if (reset = '1') then  
    q <= '0';  
  elsif (clk'event and clk = '1') then  
    q <= d;  
  end if;  
end process;
```

- Concurrent statements

```
q <= '0' when (reset = '1') else  
  d when (clk'event and clk = '1');
```



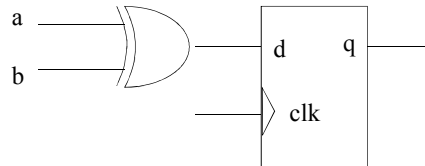
Kuruvilla Varghese



Registers with comb circuit

85

```
process (clk)
begin
  if (clk'event and clk = '1') then
    q <= a xor b;
  end if;
end process;
```



- This means a single process can code registers preceded by any combinational circuit



Kuruvilla Varghese



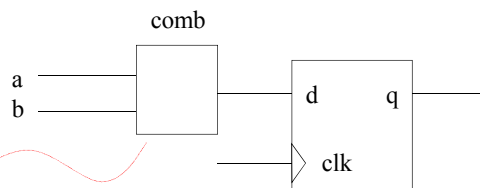
Registers with comb circuit

86

```
process (clk)
begin
  if (clk'event and clk = '1') then

    ** code for combinational
    circuit **

  end if;
end process;
```



- Note: Any assignment you do here will have flip-flop/register



Kuruvilla Varghese



Registers with comb circuit

87

- Combinational circuit at the input 'd' can be coded within the `clk'event and clk = '1'` statement
- This code being in process should be using `if ... then, case ... when, for ... etc.`
- Any signal connected to 'd' is synchronous with clock, hence such code always represent synchronous behavior

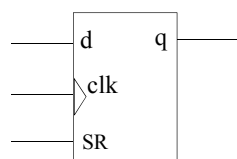


Kuruvilla Varghese



FF Synchronous Reset

88



- Asynchronous Reset

```
process (clk, reset)
begin
  if (reset = '1') then
    q <= '0';
  elsif (clk'event and clk = '1') then
    q <= d;
  end if;
end process;
```

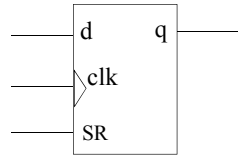


Kuruvilla Varghese



FF Synchronous Reset

89



- if ... then allows nesting and synchronous circuit can be coded
- Nesting isn't possible with concurrent statement, hence doesn't make sense to code synchronous circuit.

```
process (clk)
begin
if (clk'event and clk = '1') then
  if (reset = '1') then
    q <= '0';
  else
    q <= d;
  end if;
end if;
```

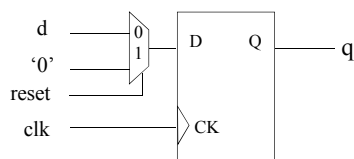


Kuruvilla Varghese



Sync Reset, Synthesis

90



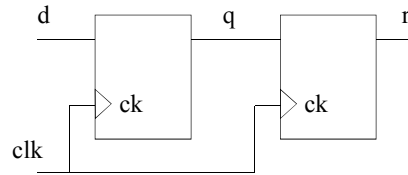
Kuruvilla Varghese



Synchronous circuit - synthesis

91

```
process (clk)
begin
if (clk'event and clk = '1') then
    q <= d;
    r <= q;
end if;
end process;
```



- Above process executes sequentially, but both the assignments happens after ' $t + \text{delta}$ ' time. This along with implied memory makes cascaded flip-flops.

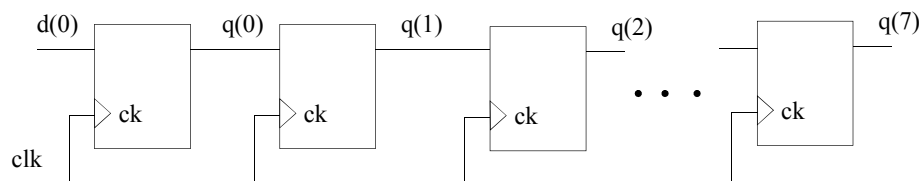


Kuruvilla Varghese



Shift Register

92



Kuruvilla Varghese



Shift Register

93

```
process (clk)
begin
  if (clk'event and clk = '1') then
    q(0) <= d(0);
    for i in 0 to 6 loop
      q(i+1) <= q(i);
    end loop;
  end if;
end process;
```

```
process (clk)
begin
  if (clk'event and clk = '1') then
    q <= q(6 downto 0) & d(0);
  end if;
end process;
```



Kuruvilla Varghese



Counter

94

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity count8 is
  port (clk, reset: in std_logic;
        count: out std_logic_vector(7 downto 0));
end count8;

architecture arch_count8 of count8 is
  signal q: std_logic_vector(7 downto 0);
begin
```

```
  count <= q;

  process (clk, reset)
  begin
    if (reset = '1') then
      q <= (others => '0');
    elsif (clk'event and clk = '1') then
      q <= q + 1;
    end if;
  end process;

end arch_count8;
```

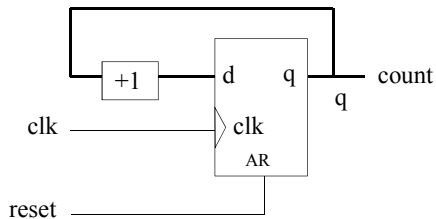


Kuruvilla Varghese



Counter

95



```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
```

```
entity count8 is port
(clk, reset, load: in std_logic;
din: in std_logic_vector(7 downto 0);
count: out std_logic_vector(7 downto 0));
end count8;
```

```
architecture arch_count8 of count8 is
signal q: std_logic_vector(7 downto 0);
begin
```



Kuruvilla Varghese

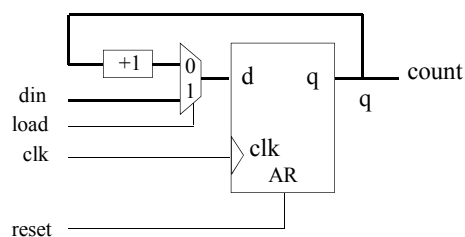


Counter

96

```
count <= q;
```

```
process (clk, reset)
begin
if (reset = '1') then
    q <= (others => '0');
elsif (clk'event and clk = '1') then
    if (load = '1') then q <= din;
    else q <= q + 1;
    end if;
end if;
end process;
end arch_count8;
```



- Synthesis Tools might optimize further to get efficient target specific circuits

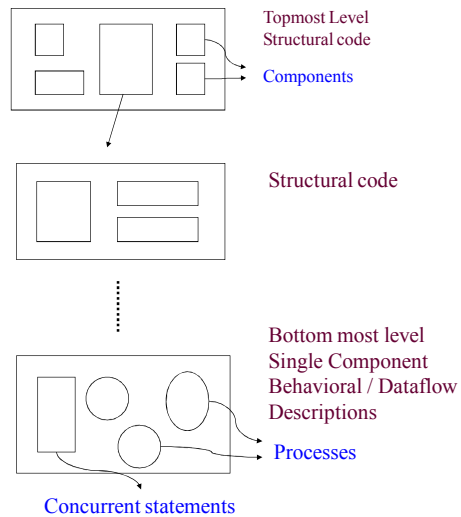


Kuruvilla Varghese



Coding Scenario

97

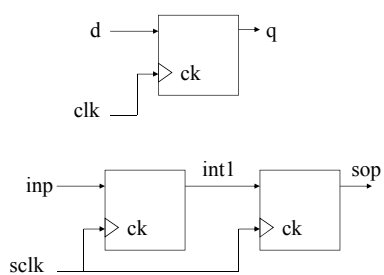


Kuruvilla Varghese



Library, Packages

98



- Component: D Flip-flop
- Tope Level entity: Double Synchronizer

```
library ieee; use ieee.std_logic_1164.all;
```

```
entity dataff is port
  (d, clk: in std_logic; q: out std_logic);
end dataff;
```

```
architecture behave of dataff is
begin
  process (clk)
  begin
    if (clk'event and clk = '1') then q <= d;
    end if;
  end process;
end behave;
```



Kuruvilla Varghese



Library, Packages

99

```
library ieee;  
use ieee.std_logic_1164.all;  
  
entity dsync is  
  port (inp, sclk: in std_logic;  
        sop: out std_logic);  
end dsync;
```

```
architecture struct of dsync is  
  component dataff  
    port (d, clk: in std_logic;  
          q: out std_logic);  
  end component;  
  signal int1: std_logic;  
begin  
  c1: dataff port map (inp, sclk, int1);  
  c2: dataff port map (int1, sclk, sop);  
end struct;
```



Kuruvilla Varghese



Library, Packages

100

- Library → Packages → Components, Functions, Procedures, Data types
- Predefined libraries – STD, WORK
- Predefined packages in STD – standard, textio
- Implicitly declared
library std, work;
use std.standard.all;
- Implicitly not declared
use std.textio.all;



Kuruvilla Varghese



Writing component in Package

101

```
library ieee;  
use ieee.std_logic_1164.all;
```

```
package xy_pkg is  
  component dff  
    port (d, clk: in std_logic;  
          q: out std_logic);  
  end component;  
end xy_pkg;
```

```
library ieee;  
use ieee.std_logic_1164.all;
```

```
entity dff is  
  port (d, clk: in std_logic; q: out std_logic);  
end dff;  
architecture behave of dff is  
begin  
  process (clk)  
begin  
  if (clk'event and clk = '1') then  
    q <= d;  
  end if;  
end process;  
end behave;
```



Kuruvilla Varghese



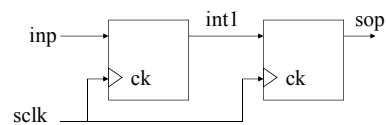
Using Component from a package

102

```
library xylib, ieee;  
use ieee.std_logic_1164.all;  
use xylib.xy_pkg.all;
```

```
entity dsync is  
  port (inp, sclk: in std_logic;  
        sop: out std_logic);  
end dsync;
```

```
architecture struct of dsync is  
  signal int1: std_logic;  
begin  
  c1: dff port map (inp, sclk, int1);  
  c2: dff port map (int1, sclk, sop);  
end struct;
```



Kuruvilla Varghese



Instantiation

103

- Positional association

c1: dff port map (inp, sclk, int1);

- Named association

c1: dff port map (clk => sclk,
 d => inp, q => int1);

- Formal to Actual association
- Signal order doesn't matter
- Need to know only the port names of the components, not the order



Kuruvilla Varghese



Generic

104

- Generic components
- Components that suite various data size, storage sizes etc.
- e.g. Counter with configurable output width
- e.g. FIFO with configurable width, configurable depth

```
library ieee;  
use ieee.std_logic_1164.all;  
  
package xy_pkg is  
  component count  
    generic (size: integer := 4);  
    port (clk, rst: in std_logic;  
          count: out  
          std_logic_vector(size-1 downto 0));  
  end component;  
end xy_pkg;
```



Kuruvilla Varghese



Generic Counter

105

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity count is
generic ( size: integer := 4);
port (clk, rst: in std_logic;
      count: out
      std_logic_vector(size-1 downto 0);
end count;

architecture behave of count is
signal q: std_logic_vector(size-1 downto 0);
begin
    count <= q;

    process (clk, rst)
    begin
        if (rst = '1') then q <= (others => '0');
        elsif (clk'event and clk = '1') then
            q <= q + 1;
        end if;
    end process;
end arch_count;
```



Kuruvilla Varghese



Instantiation

106

- Default value – 4
- Default Usage
c1: count port map (clock, rst, co);
- Generic
c1: count generic map (8) port
map (clock, rst, co);
- Any number of parameters,
any type

```
entity nand2 is
generic (tplh: time := 3 ns; tphl: time := 2
        ns);
port (i1, i2: in std_logic; o1: out std_logic);
end nand2;

.....

o1 <= i1 nand i2 after (tplh + tphl) /2;

.....
```



Kuruvilla Varghese



Generic in Hierarchy

107

Generics of components can be mapped to the generics of the architectures that instantiate those components.

```
c1: count generic map (twidht) port map  
    (clock, co);
```

Here 'twidht' is the generic of the timer which instantiate a counter with generic 'size'. When the timer instantiates counter it uses 'twidht' for 'size'.



Kuruvilla Varghese



Configuration

108

- Configuration Specification
 - Binds the components instantiated in the architecture to entity-architecture pair in any design library.
 - Specified in the architecture declaration region
- Configuration Declaration
 - Binds a top level entity to one of the many architectures it has.
 - Bind the components used at any level of hierarchy to an entity-architecture pair in any design library.
 - Separate design unit.
 - Hierarchical
 - Specified at the end
Library & Packages, Entity,
Architecture 1,
Architecture 2, ..., Configuration

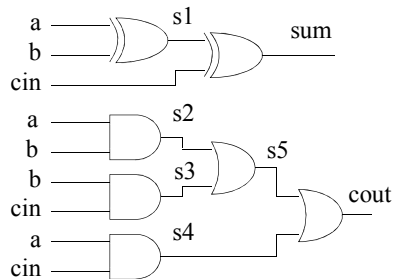


Kuruvilla Varghese



Configuration Specifications

109



```
library ieee, hs_lib, cm_lib;
use ieee.std_logic_1164.all;
```

```
entity full_adder is
  port (a, b, cin: in std_logic;
        sum, cout: out std_logic);
end full_adder;
```

```
architecture fa_str of full_adder is
  component xor2
    port (d1, d2: in std_logic;
          dz: out std_logic);
  end component;
```



Kuruvilla Varghese



Configuration Specifications

110

```
component and2
  port (z: out std_logic;
        a0, a1: in std_logic);
end component;
component or2
  port (n1, n2: in std_logic;
        z: out std_logic);
end component;
signal s1, s2, s3, s4, s5: std_logic;
```

```
-- Configuration specifications
for x1, x2: xor2 use entity
  work.xor2(xorbeh);
for a3: and2 use entity
  hs_lib.and2hs(and2str) port map
    (hs_b=>a1; hs_z=>z; hs_a=>a0);
for all: or2 use entity cmlib.or2cm(or2str);
for others: and2 use entity
  work.agate2(agate_df) port map (a0,
    a1, z);
```



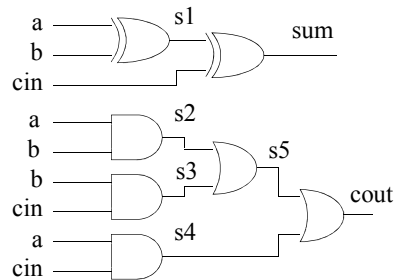
Kuruvilla Varghese



Configuration Specifications

111

```
begin
  x1: xor2 port map (a, b, s1);
  x2: xor2 port map (s1, cin, sum);
  a1: and2 port map (s2, a, b);
  a2: and2 port map (s3, b, cin);
  a3: and2 port map (s4, a, cin);
  o1: or2 port map (s2, s3, s5);
  o2: or2 port map (s4, s5, cout);
end fa_str;
```



Kuruvilla Varghese



Configuration Declaration

112

```
-- binding entity to architecture
and
-- binding components to
architecture

-- binding entity to architecture
configuration fa_con of full_adder is
  for fa_str
    use work.all;
    for a1, a2, a3: and2
      use entity cm_lib.and2(and2df);
    end for;
    for others: or2
    end for;
    for all: xor2
      use configuration work.xor2con;
    end for;
  end for;
end fa_con;
```



Kuruvilla Varghese



Packages: Operators, Functions

113

ieee.std_logic_1164 (ieee)

subtype std_logic is resolved std_ulogic;

type std_ulogic is
('U', -- Un-initialized
'X', -- Forcing Unknown
'0', -- Forcing 0
'1', -- Forcing 1
'Z', -- High Impedance
'W', -- Weak Unknown
'L', -- Weak 0
'H', -- Weak 1
'-' -- Don't care);

type std_ulogic_vector is array (natural
range <>) of std_ulogic;

type std_logic_vector is array (natural
range <>) of std_logic;

ieee.std_logic_1164 (ieee)

- Logical operators
and, nand, or, nor, xor, xnor, not



Kuruvilla Varghese



Packages: Operators, Functions

114

ieee.std_logic_unsigned (ieee)
std_logic, std_logic_vector

- Shift Operators

SHR, SHL

Overloaded operators

- Arithmetic Operators

+, -, *, /

- Relational Operators

<, >, =, /=, <=, >=

- Functions

conv_integer



Kuruvilla Varghese



Packages: Operators, Functions

115

ieee.std_logic_arith (synopsys)

```
type unsigned is array ( natural
    range <> ) of std_logic;
type signed is array ( natural range
    <> ) of std_logic;
```

Overloaded operators

- Arithmetic Operators

+, -, *, /

- Relational Operators

<, >, =, /=, <=, >=

- Shift Operators

SHR, SHL

(unsigned – logical

signed – arithmetic)



Kuruvilla Varghese



Packages: Operators, Functions

116

ieee.std_logic_arith (synopsys)

Conversion Functions

from: std_logic_vector,
unsigned, signed, integer

conv_integer

conv_unsigned

conv_signed

conv_std_logic_vector

- Usage

library ieee ;

use ieee.std_logic_1164.all ;

use ieee.std_logic_arith.all ;

use ieee.std_logic_unsigned.all;

- Recommendations

– Use std_logic_arith for numeric operations

– Use std_logic_unsigned only for counters and testbenches

– Don't use the package
std_logic_signed



Kuruvilla Varghese



Packages: Operators, Functions

117

ieee.numeric_std (ieee)

```
type unsigned is array ( natural range <>
  ) of std_logic;
```

```
type signed is array ( natural range <> )
  of std_logic;
```

Overloaded operators

- Arithmetic Operators
+, -, *, /, abs, rem, mod
- Relational Operators
<, >, =, /=, <=, >=
- Logical operators
and, nand, or, nor, xor, xnor, not



Kuruvilla Varghese



Packages: Operators, Functions

118

ieee.numeric_std (ieee)

• Usage

- Shift operators (signed, unsigned)

shift_left, shift_right
rotate_left, rotate_right
sll, srl, rol, ror

- Conversion Functions

to_integer
to_unsigned
to_signed

```
library ieee ;  
use ieee.std_logic_1164.all ;  
use ieee.numeric_std.all ;
```



Kuruvilla Varghese



Type Conversions

119

- **Automatic**
 - Between base types and subtypes
- **Using Conversion Functions**
 - e.g. `to_integer`, `conv_integer`
- **Type Casting**
 - between signed, unsigned, and `std_logic_vector`

```
sl_vect <= std_logic_vector(usg_vect)
sl_vect <= std_logic_vector(sg_vect)
usg_vect <= unsigned(sl_vect)
sg_vect <= signed(sl_vect)

signed("1101")
```



Kuruvilla Varghese



Type Conversion and synthesis

120

- Type conversion is required when you connect a signal of one data type (e.g. integer) to another (e.g. `std_logic_vector`), as VHDL is a strict type checking language
- Type conversion implies no hardware, Hence directives (user defined attributes) are given to synthesis tool, not to synthesize the code.
- But, in a code, where a `std_logic_vector` address is converted to integer, as an index into a memory array, type conversion implies an address decoder



Kuruvilla Varghese



Arithmetic

121

```
signal a, b, s: unsigned(7 downto 0)
;
signal s9: unsigned(8 downto 0) ;
signal s7: unsigned(6 downto 0) ;
```

-- Simple Addition, no carry out

```
s <= a + b ;
```

-- Carry Out in result

```
s9 <= ('0' & a) + ('0' & b) ;
```

-- For smaller result, slice input
arrays

```
s7 <= a(6 downto 0) + b(6 downto 0)
```

```
signal a, b, s: unsigned(7 downto 0);
signal s9: unsigned(8 downto 0) ;
signal cin: std_logic ;
-- Carry in
s9 <= (a & '1') + (b & cin);
s <= s9(8 downto 1) ;
```



Kuruvilla Varghese



Arithmetic with time

122

- Suppose you want to do some computation with 'time' data type
- Then it is better to remove time unit and cast it to real and do the arithmetic operations like multiplication, division etc.

```
variable period: real;
period := real(time_data / 1 ns)
```



Kuruvilla Varghese



Delay Modeling

123

- Inertial delay,
- Transport delay

- Inertial delay

Models delay through capacitive networks and through gates with threshold values

Pulse rejection value less than the inertial delay with “reject” clause.

- Two parameters:

Minimum pulse width required for the device to recognize the level change (i.e. for the input to cross the threshold)

Propagation delay of the device



Kuruvilla Varghese



Delay Modeling

124

`x <= a after 5 ns;`

`x <= inertial a after 5 ns;`

`y <= reject 3 ns inertial a after 5 ns;`

`u <= '1' after 5 ns, '0' after 8 ns,
 '1' after 12 ns;`

- Transport delay

- Models delay through transmission lines and networks. No pulse rejection.

`z <= transport a after 5 ns;`



Kuruvilla Varghese



Delay Modeling

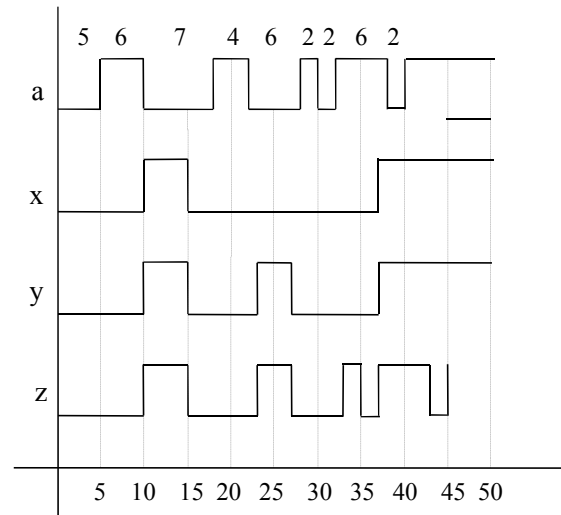
125

```
entity delaym is
end entity;
```

```
architecture arch_d of delaym is
signal a, x, y, z: bit ;
begin
```

```
a <= '1' after 5 ns, '0' after 11 ns,
    '1' after 18 ns, '0' after 22 ns,
    '1' after 28 ns, '0' after 30 ns,
    '1' after 32 ns, '0' after 38 ns,
    '1' after 40 ns;
```

```
x <= a after 5 ns;
y <= reject 3 ns inertial a after 5 ns;
z <= transport a after 5 ns;
end arch_d;
```

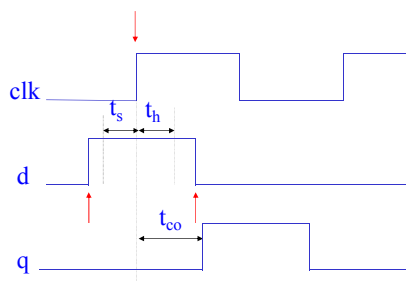
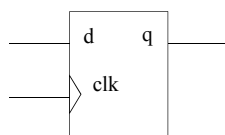


Kuruvilla Varghese



Timing Example

126



```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity dff is
port (d, clk: in std_logic; q: out
std_logic);
end dff;
```

```
architecture tcheck of dff is
constant setup_time: time := 3 ns;
constant hold_time: time := 1 ns;
begin
```



Kuruvilla Varghese



Timing Example

127

```
process (d, clk)
variable d_levent, clk_levent: time;
begin

-- behavioral code of dff
if (clk'event and clk = '1') then
    q <= d after 10 ns;
end if;
if d'event then
    assert now = 0 ns or (now - clk_levent)
        >= hold_time
        report "Hold time violation !"
        severity note;
    d_levent := now;
end if;

if (clk'event and clk = '1') then
    assert now = 0 ns or (now - d_levent)
        >= setup_time
        report "Setup time violation !"
        severity note;
    clk_levent := now;
end if;
end process;
end tcheck;
```



Kuruvilla Varghese



Assert

128

Syntax

```
assert (true or expected condition)
report "any message"
severity level;
```

Levels

Note
Warning
Error
Failure



Kuruvilla Varghese



Example - Demultiplexer

129

```
process (s, y)
begin
case s is
when "00" =>
a <= y; b <= "0000"; c <= "0000";
d <= "0000";
when "01" =>
b <= y; a <= "0000"; c <= "0000";
d <= "0000";
when "10" =>
c <= y; a <= "0000"; b <= "0000";
d <= "0000";
when "11" =>
d <= y; a <= "0000"; b <= "0000";
c <= "0000";
when others =>
a <= "0000"; b <= "0000"; c <= "0000";
d <= "0000";
end case;
end process;
end arch_dmux1t4;
```



Kuruvilla Varghese



Example - Demultiplexer

130

```
library ieee;
use ieee.std_logic_1164.all;

entity dmux1t4 is
port (y: in std_logic_vector(3 downto 0);
s: in std_logic_vector(1 downto 0);
a, b, c, d: out std_logic_vector(3
downto 0));
end dmux1t4;

architecture arch_dmux1t4 of dmux1t4 is
begin
a(0) <= y(0) and not(s(1)) and not(s(0));
a(1) <= y(1) and not(s(1)) and not(s(0));
a(2) <= y(2) and not(s(1)) and not(s(0));
a(3) <= y(3) and not(s(1)) and not(s(0));

b(0) <= y(0) and not(s(1)) and s(0);
b(1) <= y(1) and not(s(1)) and s(0);
b(2) <= y(2) and not(s(1)) and s(0);
b(3) <= y(3) and not(s(1)) and s(0);

c(0) <= y(0) and s(1) and not(s(0));
c(1) <= y(1) and s(1) and not(s(0));
c(2) <= y(2) and s(1) and not(s(0));
c(3) <= y(3) and s(1) and not(s(0));
```



Kuruvilla Varghese



Example - Demultiplexer

131

```
d(0) <= y(0) and s(1) and s(0);  
d(1) <= y(1) and s(1) and s(0);  
d(2) <= y(2) and s(1) and s(0);  
d(3) <= y(3) and s(1) and s(0);  
  
end arch_dmux1t4;
```

```
architecture arch_dmux1t4 of dmux1t4 is  
begin
```

```
    glp: for i in 0 to 3 generate  
        a(i) <= y(i) and not(s(1)) and not(s(0));  
        b(i) <= y(i) and not(s(1)) and s(0);  
        c(i) <= y(i) and s(1) and not(s(0));  
        d(i) <= y(i) and s(1) and s(0);  
    end generate;
```

```
end arch_dmux1t4;
```



Kuruvilla Varghese



With ... select

132

```
architecture arch_dmux1t4 of dmux1t4 is  
begin
```

```
    with s select  
        a <= y when "00",  
            "0000" when others;
```

```
    with s select  
        b <= y when "01",  
            "0000" when others;
```

```
    with s select  
        c <= y when "10",  
            "0000" when others;
```

```
    with s select  
        d <= y when "11",  
            "0000" when others;
```

```
end arch_dmux1t4;
```



Kuruvilla Varghese



When ... else / Case

133

architecture arch_dmux1t4 of dmux1t4 is
begin

```
a <= y when (s = "00") else "0000";  
b <= y when (s = "01") else "0000";  
c <= y when (s = "10") else "0000";  
d <= y when (s = "11") else "0000";
```

end arch_dmux1t4;

architecture arch_dmux1t4 of dmux1t4 is

begin

process (s, y)

begin

```
a <= "0000"; b <= "0000"; c <= "0000";  
d <= "0000";
```

case s is

when "00" => a <= y;

when "01" => b <= y;

when "10" => c <= y;

when "11" => d <= y;

when others => null;

end case;

end process;



Kuruvilla Varghese



If ... then

134

architecture arch_dmux1t4 of dmux1t4 is
begin

process (s, y)

begin

if (s = "00") then

a <= y; b <= "0000"; c <= "0000";

d <= "0000";

elsif (s = "01") then

b <= y; a <= "0000"; c <= "0000";

d <= "0000";

elsif (s = "10") then

c <= y; a <= "0000"; b <= "0000";

d <= "0000";

else

d <= y; a <= "0000"; b <= "0000";

c <= "0000";

end if;

end process;

end arch_dmux1t4;



Kuruvilla Varghese



If ... then

135

```
process (s, y)
begin
  a <= "0000"; b <= "0000"; c <= "0000";
  d <= "0000";
  if (s = "00") then a <= y;
  elsif (s = "01") then b <= y;
  elsif (s = "10") then c <= y;
  else d <= y;
  end if;
end process;
```



Kuruvilla Varghese



If ... then

136

```
process (s, y)
begin
  if (s = "00") then a <= y; else a <=
    "0000"; end if;
  if (s = "01") then b <= y; else b <=
    "0000"; end if;
  if (s = "10") then c <= y; else c <= "0000";
    end if;
  if (s = "11") then d <= y; else d <=
    "0000"; end if;
end process;
```

```
process (s, y)
begin
  a <= "0000"; b <= "0000"; c <= "0000";
  d <= "0000";
  if (s = "00") then a <= y; end if;
  if (s = "01") then b <= y; end if;
  if (s = "10") then c <= y; end if;
  if (s = "11") then d <= y; end if;
end process;
```

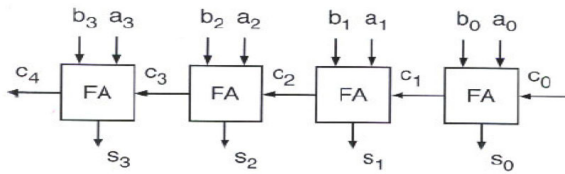


Kuruvilla Varghese



Ripple Adder

137



```
library ieee;
use ieee.std_logic_1164.all;
```

entity add8 is

```
port(a, b: in std_logic_vector(7 downto 0);
      sum: out std_logic_vector(7 downto 0);
      cin: in std_logic; cout: out std_logic);
end add8;
```

architecture arch_add8 of add8 is
 signal carry: std_logic_vector (8 downto 0);
begin

```
carry(0) <= cin; cout <= carry(8);
```

```
glp: for i in 0 to 7 generate
```

```
sum(i) <= a(i) xor b(i) xor carry(i);
```

```
carry(i+1) <= (a(i) and b(i)) or ((a(i) or b(i))
```

```
and carry(i));
```

```
end generate;
```

```
end arch_add8;
```

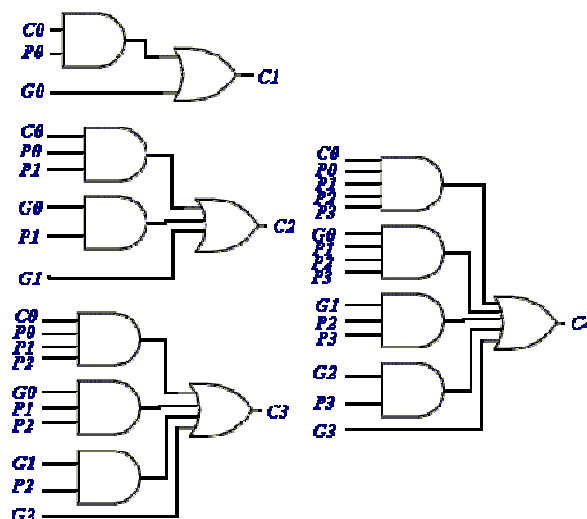


Kuruvilla Varghese



Carry Look Ahead Adder

138



$$s(i) = a(i) \text{ xor } b(i) \text{ xor } c(i)$$

$$c(i+1) = a(i) b(i) \text{ or } (a(i) \text{ or } b(i)) c(i)$$

$$g(i) = a(i) b(i)$$

$$p(i) = a(i) \text{ or } b(i)$$

Image Source: htsvn.getgoo.net



Kuruvilla Varghese



Carry Look Ahead Adder

139

```
architecture arch_add8 of add8 is
    signal carry: std_logic_vector(8 downto 0);
    signal g, p: std_logic_vector(7 downto 0);
begin

    carry(0) <= cin; cout <= carry(8);

    glp: for i in 0 to 7 generate
        g(i) <= a(i) and b(i);
        p(i) <= a(i) or b(i);
        sum(i) <= a(i) xor b(i) xor carry(i);
    end generate;

    carry(1) <= g(0) or (p(0) and carry(0));
    carry(2) <= g(1) or (p(1) and g(0)) or
        (p(1) and p(0) and carry(0));
    carry(3) <= g(2) or (p(2) and g(1)) or
        (p(2) and p(1) and g(0)) or
        (p(2) and p(1) and p(0) and carry(0));
    .....
end arch_add8;
```



Kuruvilla Varghese



Adder: Operator

140

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity add8 is
    port(a, b: in std_logic_vector(7 downto 0);
        sum: out std_logic_vector(7 downto 0));
end add8;

architecture arch_add8 of add8 is
begin
    sum <= a + b;
end arch_add8;
```

- Depends on how operator is implemented
- Mostly ripple adder
- In FPGA's it will result in using in-built adder resource

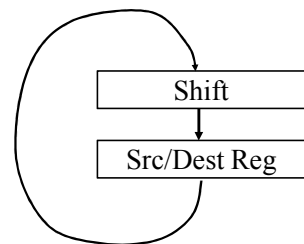
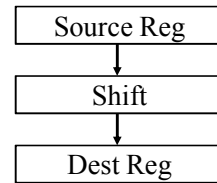
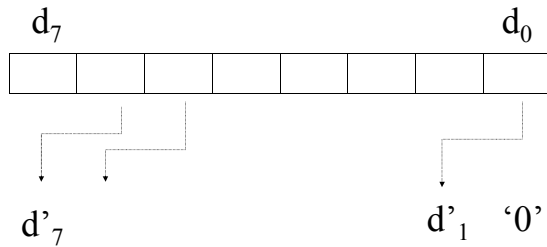


Kuruvilla Varghese



Shift Register

141



- Fixed Shift – wiring
- Destination could be a different register or same register

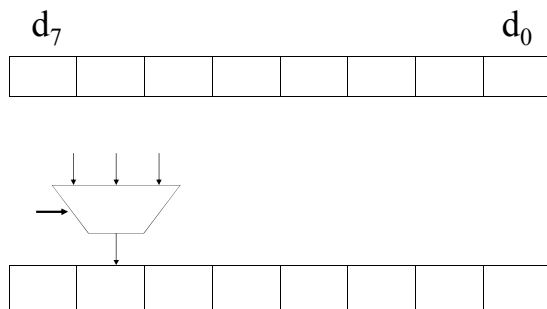


Kuruvilla Varghese



Shift Register

142



- Variable Shift – Multiplexer

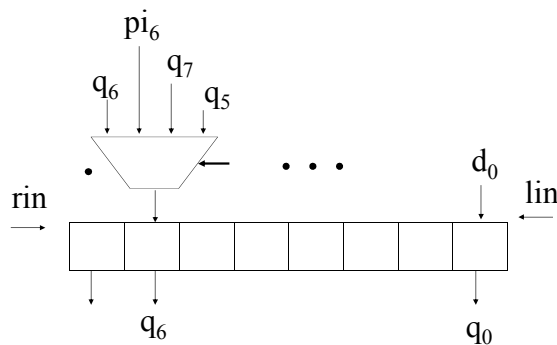


Kuruvilla Varghese



Universal Shift Register

143



```
library ieee;
use ieee.std_logic_1164.all;
```

entity shift8 is port

```
(clk, reset, lin, rin: in std_logic;
 sel: in std_logic_vector(1 downto 0);
 pin: in std_logic_vector(7 downto 0);
 y: out std_logic_vector(7 downto 0));
```

end shift8;

architecture arch_shift8 of shift8 is

```
signal q: out std_logic_vector(7 downto 0);
```

```
begin
```

```
y <= q;
```



Kuruvilla Varghese



Universal Shift Register

144

```
process (reset, clk)
```

```
begin
```

```
if (reset = '1') then q <= (others => '0');
```

```
elsif (clk'event and clk = '1') then
```

```
case sel is
```

```
-- parallel load
```

```
when "00" => q <= pin;
```

```
-- shift left
```

```
when "01" => q(0) <= lin;
```

```
for i in 0 to 6 loop
```

```
q(i+1) <= q(i);
```

```
end loop;
```

```
-- shift right
```

```
when "10" =>
```

```
q(7) <= rin;
```

```
for i in 6 downto 0 loop
```

```
q(i) <= q(i+1);
```

```
end loop;
```

```
-- hold
```

```
when others => q <= q;
```

```
end case;
```

```
end if;
```

```
end process;
```

```
end arch_shift8;
```



Kuruvilla Varghese



Universal Shift Register

145

```

process (reset, clk)
begin
  if (reset = '1') then
    q <= (others => '0');
  elsif (clk'event and clk = '1') then
    case sel is
      -- parallel load
      when "00" => q <= pin;
      -- shift left
      when "01" =>
        q(7 downto 0) <= q(6 downto 0) & lin;

        -- shift right
        when "10" =>
          q(7 downto 0) <= rin & q(7 downto 1);
      -- hold
      when others => q <= q;
    end case;
  end if;
end process;
end arch_shift8;

```

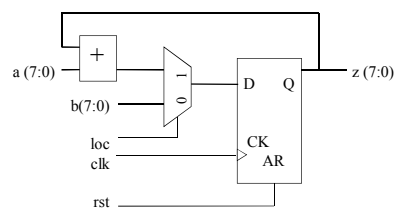
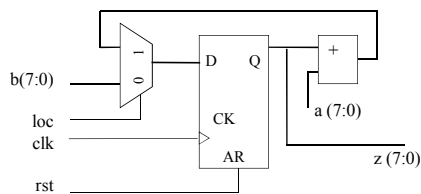


Kuruvilla Varghese



VHDL Code

146



Kuruvilla Varghese



VHDL Code

147

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity dt1q2 is
port (clk, rst, loc: in std_logic;
      a, b: in std_logic_vector(7 downto 0);
      z: out std_logic_vector(7 downto 0));
end entity;

architecture arch_dt1q2 of dt1q2 is
signal q: std_logic_vector(7 downto 0);
begin
z <= q;

process (rst, clk)
begin
if (rst = '1') then q <= (others => '0');
elsif (clk'event and clk = '1') then
if (loc = '1') then
q <= q + a;
else
q <= q + b;
end if;
end if;
end process;
end arch_dt1q2;
```



Kuruvilla Varghese



Draw the block diagram

148

```
library ieee; use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity dt1q3 is
port (a, b, d: in std_logic;
      c: in std_logic_vector(7 downto 0);
      z: out std_logic_vector(7 downto 0));
end entity;

architecture arch_dt1q3 of dt1q3 is
signal y: std_logic_vector(7 downto 0);
begin
z <= y;

process (a, b)
begin
if (a = '1') then y <= (others => '0');
elsif (b'event and b = '1') then
if (y = c) then
for i in 0 to 6 loop
y(i+1) <= y(i);
end loop;
y(0) <= d;
end if;
end if;
end process;
end arch_dt1q3;
```

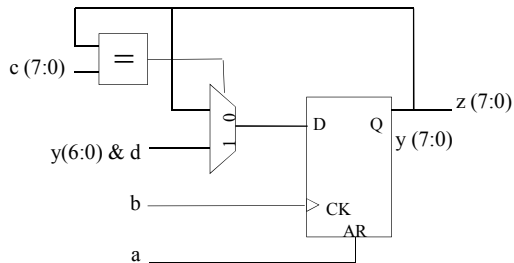


Kuruvilla Varghese



Block Diagram

149



Kuruvilla Varghese



VHDL to Circuit

150

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity dsft1a is
port (u, v: in std_logic_vector(3 downto 0);
      w: out std_logic_vector(7 downto 0));
end dsft1a;
```

```
architecture arch_dsft1a of dsft1a is
type dtype1 is array (3 downto 0) of
    std_logic_vector(3 downto 0);
signal y: dtype1;
type dtype2 is array (3 downto 0) of
    std_logic_vector(7 downto 0);
signal x: dtype2;

constant temp: std_logic_vector(3 downto 0)
    := "0000";

begin
```



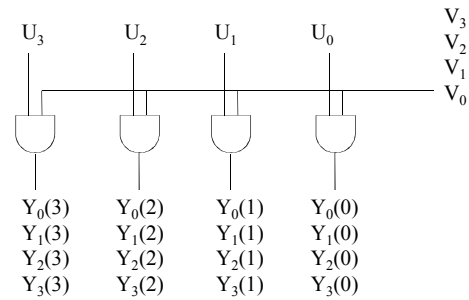
Kuruvilla Varghese



VHDL to Circuit

151

```
gnlp1: for i in 0 to 3 generate
  y(i) <= u and (v(i), v(i), v(i), v(i));
end generate;
```



Kuruvilla Varghese



VHDL to Circuit

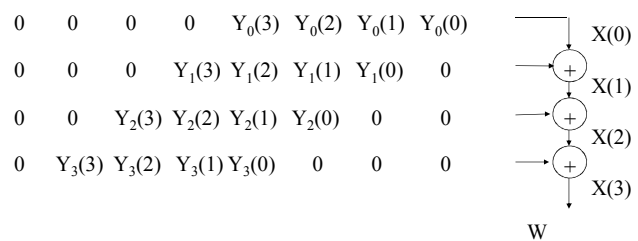
152

```
x(0) <= temp(3 downto 0) & y(0);
```

```
gnlp2: for i in 1 to 3 generate
  x(i) <= temp (3 downto i) & y(i) &
    temp(i-1 downto 0) +
    x(i-1);
end generate;
```

```
w <= x(3);
```

```
end arch_dsft1a;
```



- 4 bit Array Multiplier
- Adders are ripple Adders (+)



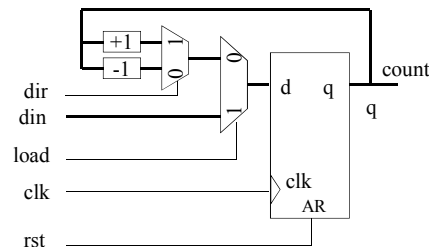
Kuruvilla Varghese



Design

153

- Synchronous 4 bit up/down counter with parallel load feature
- Block schematic
- Behavioral VHDL Code



Kuruvilla Varghese



Design: Counter

154

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity dsft1a is
port (clk, rst, load, dir: in std_logic;
      din: in std_logic_vector(3 downto 0);
      count: out std_logic_vector(3 downto 0));
end dsft1a;

architecture arch_dsft1a of dsft1a is
signal q: std_logic_vector(3 downto 0);
begin

count <= q;

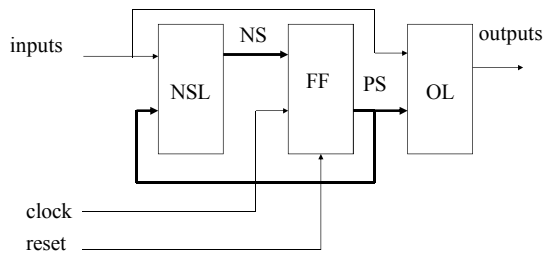
process (clk, rst)
begin
    if (rst = '1') then q <= (others => '0');
    elsif (clk'event and clk = '1') then
        if (load = '1') then q <= din;
        elsif (dir = '1') then q <= q + 1;
        else q <= q - 1;
        end if;
    end if;
end process;

end arch_dsft1a;
    
```

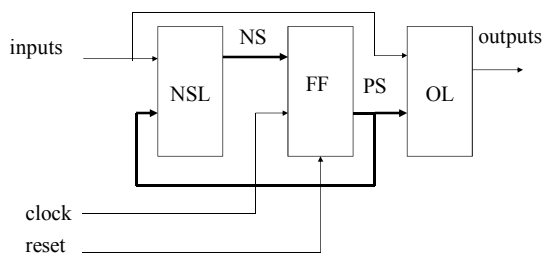


Kuruvilla Varghese





- Inputs, clock, outputs: ports / signals
- Present State, Next State: signals
- Choose the FSM Coding style specified by the Synthesis Tool, as the tool can extract the FSM and optimize it.



- 1 Process for NSL, 1 process for FF, 1 process for outputs
- 1 Process for NSL, 1 process for FF, Concurrent statements for outputs (When number of outputs are less)
- 1 Process for NSL + FF, 1 process for Outputs
- 1 Process for NSL + FF, Concurrent statements for outputs (When number of outputs are less)
- Asynchronous reset in FF process
- Synchronous reset in NSL process



FSM Coding

157

- **NSL: Process,**
“case ... when” (on present state)
with nested “if (on inputs)
...then” for next state
- **Flip Flops: Process,**
“if (rst) ... then”
elsif (clk’event and clk = ‘1’)
then
- **OL: Process**
“case ... when” (present state)
for outputs.
- **OL: Concurrent statements**
with (present state) ... select
- **NSL+FF: Process**
if (rst) ... then
elsif (clk’event and clk = ‘1’) then
case ... when” (on present state)

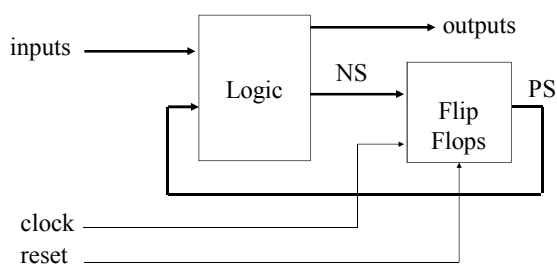


Kuruvilla Varghese



FSM Coding

158



- **Logic: Process,**
“case ... when” (present state)
outputs.
“if (on inputs) ...then” for
next state
- **Flip Flops: Process,**
“if (rst) ... then”
“elsif (clk’event and clk = ‘1’)
then”
- **Implied Latch on outputs when
synchronous reset is used**
- 1 process for Logic (NSL + OL)
- 1 process for FF’s



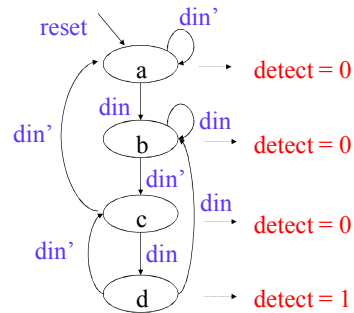
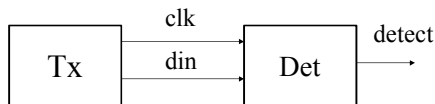
Kuruvilla Varghese



Sequence Detector

159

- A simple sequence Detector. Sequence 101. Data shift clock used as FSM clock. Overlapping detector. Moore Machine



Kuruvilla Varghese



Sequence Detector: VHDL Code

160

```
-- NSL process; FF process; OL
concurrent
library ieee;
use ieee.std_logic_1164.all;

entity sqdet1 is
    port (din, clk, reset: in std_logic;
          detect: out std_logic);
end sqdet1;

architecture sqd1 of sqdet1 is
    type statetype is (a, b, c, d);
    signal pr_state, nx_state: statetype;
begin
```

```
    nsl: process (pr_state, din)
    begin
        case pr_state is
            when a =>
                if din = '1' then nx_state <= b;
                else nx_state <= a;
                end if;
            when b =>
                if din = '0' then nx_state <= c;
                else nx_state <= b;
                end if;
            when c =>
                if din = '1' then nx_state <= d;
                else nx_state <= a;
                end if;
```



Kuruvilla Varghese



Sequence Detector: VHDL Code

161

```
when d =>
  if din = '0' then nx_state <= c;
  else nx_state <= b;
  end if;
when others =>
  nx_state <= a;
end case;
end process nsl;
```

```
fifi: process (clk, reset)
begin
  if (reset = '1') then pr_state <= a;
  elsif (clk'event and clk = '1') then
    pr_state <= nx_state;
  end if;
end process fifi;
```

```
detect <= '1' when (pr_state = d) else '0';
end sqd1;
```

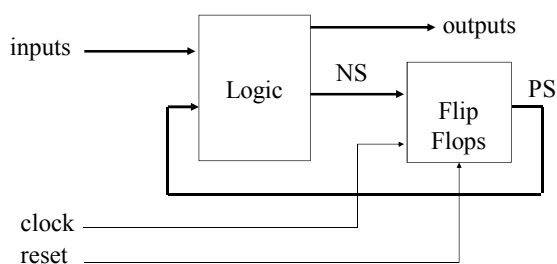


Kuruvilla Varghese



FSM Coding

162



- 1 process for Logic (NSL + OL)
- 1 process for FF's

- Logic: Process,
“case ... when” (present state) outputs.
“if (on inputs) ... then” for next state
- Flip Flops: Process,
“if (rst) ... then”
“elsif (clk'event and clk = '1') then”
- Implied Latch on outputs when synchronous reset is used



Kuruvilla Varghese



Sequence Detector: VHDL Code

163

```
• -- NSL + OL process; FF process;
library ieee;
use ieee.std_logic_1164.all;

entity sqdet1 is
  port (din, clk, reset: in std_logic;
        detect: out std_logic);
end sqdet1;

architecture sqd1 of sqdet1 is
  type statetype is (a, b, c, d);
  signal pr_state, nx_state: statetype;
begin
  comb: process (pr_state, din)
  begin
    case pr_state is
      when a => detect <= '0';
        if din = '1' then nx_state <= b;
        else nx_state <= a;
        end if;
      when b => detect <= '0';
        if din = '0' then nx_state <= c;
        else nx_state <= b;
        end if;
      when c => detect <= '0';
        if din = '1' then nx_state <= d;
        else nx_state <= a;
        end if;
```



Kuruvilla Varghese



Sequence Detector: VHDL Code

164

```
when d => detect <= '1';
  if din = '0' then nx_state <= c;
  else nx_state <= b;
  end if;
when others => detect <= '0';
  nx_state <= a;
end case;
end process comb;

ffl: process (clk, reset)
begin
  if (reset = '1') then pr_state <= a;
  elsif (clk'event and clk = '1') then
    pr_state <= nx_state;
  end if;
end process ffl;
```



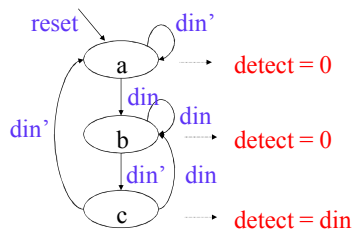
Kuruvilla Varghese



Sequence Detector - Mealy

165

- A simple sequence Detector. Sequence 101. Data shift clock used as FSM clock. Overlapping detector. Mealy machine



-- NSL + OL single process

```

comb: process (pr_state, din)
begin
  case pr_state is
    when a => detect <= '0';
      if din = '1' then nx_state <= b;
      else nx_state <= a; end if;
    when b => detect <= '0';
      if din = '0' then nx_state <= c;
      else nx_state <= b; end if;
    when c => detect <= din;
      if din = '1' then nx_state <= b;
      else nx_state <= a; end if;
    when others => detect <= '0';
      nx_state <= a;
  end case;
end process comb;
    
```



Kuruvilla Varghese



Synchronous Reset

166

-- OL Concurrent statement

```

detect <= '1' when ((pr_state = c) and
  (din = '1')) else '0';
    
```

-- Synchronous reset when NSL + OL is

-- coded in single process

-- Avoid implied latches on outputs

```

comb: process (reset, pr_state, din)
begin
  if (reset = '1') then
    detect <= '-'; nx_state <= a;
  else
    case pr_state is
      -----
    end case;
  end if;
end process comb;
    
```



Kuruvilla Varghese



Synchronous Reset

167

```
-- Synchronous reset when NSL + OL is  
-- coded in single process  
-- Avoid implied latches on outputs  
comb: process (reset, pr_state, din)
```

```
begin
```

```
case pr_state is
```

```
-----
```

```
end case;
```

```
if (reset = '1') then nx_state <= a;
```

```
end if;
```

```
end process comb;
```

- State encoding

sequential, gray, one-hot-one,
one-hot-zero



Kuruvilla Varghese



State encoding

168

- User defined attributes

- attribute state-encoding of
type-name: type is value;
(sequential, gray, one-hot-one,
one-hot-zero)

```
attribute state_encoding of  
statetype: type is gray;
```

- attribute enum_encoding of
type-name: type is "string";
attribute enum_encoding of
statetype: type is "00 01 11
10";

- Explicit declaration of states

```
signal pr_state, nx_state:
```

```
std_logic_vector(3 downto 0);
```

```
constant a: std_logic_vector(3 downto 0)  
:= "0001";
```

```
constant b: std_logic_vector(3 downto 0)  
:= "0010";
```

```
constant c: std_logic_vector(3 downto 0)  
:= "0100";
```

```
constant d: std_logic_vector(3 downto 0)  
:= "1000";
```

- FSM Editors



Kuruvilla Varghese



Test Bench

169

- Interactive simulation
- Design steps verification
- Design Iterations
- Test bench
 - Input test vectors could be stored in a file
 - Output test vectors observed as waveform or could be stored in a file.
 - Output could be checked against the expected response stored.
 - Automation, Documentation
 - Same test bench could be used in different design steps and in design iterations.



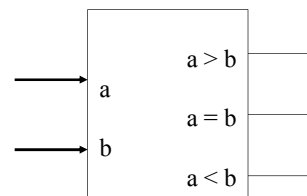
Kuruvilla Varghese



Test Bench

170

- Process
 - Compile the design in to library (normally work library).
 - Create test bench code with empty entity.
 - Declare the top level component of design.
 - Declare the signals of type of the top level component.
 - Instantiate the component in the test bench code.
 - Apply the stimulus to the input ports.
 - Compile the test bench and run simulation.
 - Observe the waveform and verify.
- Component
 - Comparator



Kuruvilla Varghese



Test bench Signal Assignment

171

```
library ieee; use ieee.std_logic_1164.all;    a <= "0000",           -- a = b
entity comp_tb is                             "1111" after 100 ns,  -- a > b
  comp4                                        "1110" after 200 ns,  -- a < b
end comp_tb;                                  "1110" after 300 ns,  -- a > b

architecture arch_comp_tb of comp_tb is       "1010" after 400 ns,  -- a < b
  component comp4                             "1111" after 500 ns; -- a = b
  port (a, b: in std_logic_vector(3 downto 0); b <= "0000",           -- a = b
        agtb, aeqb, altb: out std_logic);     "1110" after 100 ns,  -- a > b
  end component;                             "1111" after 200 ns,  -- a < b
  signal a, b: std_logic_vector(3 downto 0);  "1100" after 300 ns,  -- a > b
  signal agtb, aeqb, altb: std_logic;         "1100" after 400 ns,  -- a < b
  begin                                       "1111" after 500 ns;  -- a = b
  a1: comp4 port map (a, b, agtb, aeqb, altb); end arch_comp_tb;
```



Kuruvilla Varghese



Test bench Signal Assignment

172

-- another way

```
a <= "0000"; b <= "0000";
wait for 100 ns;
a <= "1111"; b <= "1110";
wait for 100 ns;
a <= "1110"; b <= "1111";
wait for 100 ns;

end arch_comp_tb;
```

- Features
 - Stimulus is part of the test bench code
 - Stimulus is distributed in the test bench code
 - Manual verification
- Can we store input test vectors and expected outputs together?
- What data type is best for this?
 - Record



Kuruvilla Varghese



Test bench TV's in Array

173

- Compile the design in to library (normally work library).
- Create test bench code with empty entity.
- Declare the top level component of the design.
- Declare the signals of type of the top level component.
- Declare a record with port signals of top-level component.
- Declare an array of this record.
- Initialize the array with the input test vectors and expected outputs
- Instantiate the component in the test bench code.
- Check Process:
 - Declare a variable of type record
- Loop
 - Read the array element to the record variable
 - Apply the stimulus to the input ports,
 - Verify the actual outputs match the expected output.
- Compile the test bench and run simulation.
- Check the Messages for error; up on error observe debug messages and/or the waveform



Kuruvilla Varghese



Test bench - Component

174

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity count4 is
port (clk, rst: in std_logic;
      q: out std_logic_vector(3 downto 0));
end count4;

architecture arch_count4 of count4 is
signal count: std_logic_vector(3 downto 0);
begin
    q <= count;

    process (clk, rst)
    begin
        if (rst = '1') then count <= (others => '0');
        elsif (clk'event and clk = '1') then
            count <= count + 1;
        end if;
    end process;
end arch_count4;
```



Kuruvilla Varghese



Test bench TV's in Array

175

```
library ieee;
use ieee.std_logic_1164.all;

entity countf_tb is
end countf_tb;

architecture arch_countf_tb of countf_tb is
  component count4
  port (rst, clk: in std_logic;
        q: out std_logic_vector(3 downto 0));
  end component;
  signal clk, rst: std_logic;
  signal q: std_logic_vector(3 downto 0);

  type tblk is record
    rst: std_logic; clk: std_logic;
    q: std_logic_vector(3 downto 0);
  end record;
  type testv is array (natural range <>) of tblk;

  constant tv: testv :=
    (('1', '0', "0000"), ('1', '0', "0000"),
     ('0', '0', "0000"), ('0', '1', "0001"),
     ('0', '0', "0001"), ('0', '1', "0010"),
     ('0', '0', "0010"), ('0', '1', "0011"),
     ('0', '0', "0011"), ('0', '1', "0100"),
     ('0', '0', "0100"), ('0', '1', "0101"),
     ('0', '0', "0101"), ('0', '1', "0110"),
     ('0', '0', "0110"), ('0', '1', "0111"));
begin
```



Kuruvilla Varghese



Test bench TV's in Array

176

```
uut: count4 port map (rst => rst,
                      clk => clk, q => q);

test: process
  variable vtv: tblk;
begin
  for i in tv'range loop
    vtv := tv(i);
    rst <= vtv.rst;
    clk <= vtv.clk;
    wait for 20 ns;

    assert q = vtv.q report "Counter output is not
                           what expected" severity note;
  end loop;
  assert false report "Test over" severity note;
  wait;
end process;
end arch_countf_tb;
```

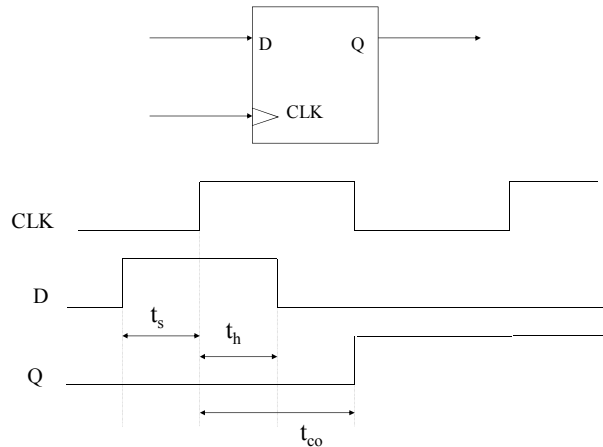


Kuruvilla Varghese



Test bench: Timing Simulation

177



t_s : Setup time: Minimum time input must be valid before the active clock edge

t_h : Hold time: Minimum time input must be valid after the active clock edge

t_{co} : Propagation delay for input to appear at the output from active clock edge



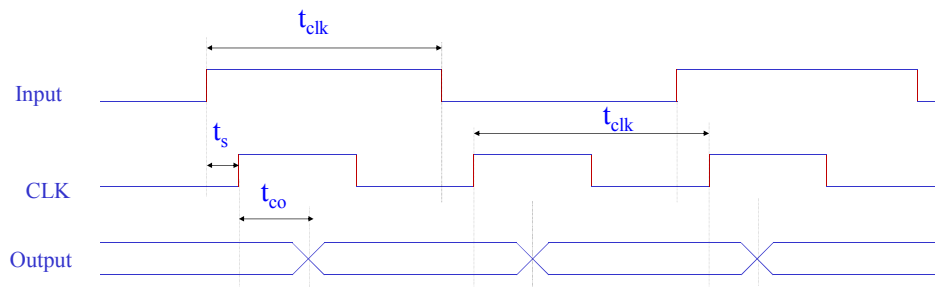
Kuruvilla Varghese



Test bench: Timing Simulation

178

- Setting up input
- Verifying the output



Kuruvilla Varghese



Test bench: Timing Simulation

179

- **Clock Generation**
 - Period, Initial offset, Duty cycle
 - **Asynchronous Reset**
 - Remove t_{RR} time before clock edge
 - Example uses t_s instead of t_{RR}
 - **Applying Inputs**
 - Apply t_s time before active clock edge
 - Detect the clock edge explicitly or implicitly (e.g. add clock period – setup)
 - **Checking outputs**
 - Give a delay for the full run after applying inputs and add $t_s + t_{co}$
 - Or if there is a data valid signal detect it
- Note:** Period, setup time, and t_{co} should be noted from the Static Timing Analysis for test bench, and is with respect to corresponding input/output pin/pad.



Kuruvilla Varghese



Test bench: Timing Simulation

180

- **Clock Generation**
 - Period, Initial offset, Duty cycle
 - **Asynchronous Reset**
 - Remove t_{RR} time before clock edge
 - Example uses t_s instead of t_{RR}
 - **Applying Inputs**
 - Apply t_s time before active clock edge
 - Detect the clock edge explicitly or implicitly (e.g. add clock period – setup)
 - **Checking outputs**
 - Give a delay for the full run after applying inputs and add $t_s + t_{co}$
 - Or if there is a data valid signal detect it
- Note:** Period, setup time, and t_{co} should be noted from the Static Timing Analysis for test bench



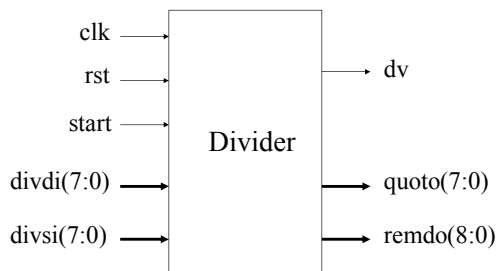
Kuruvilla Varghese



Test bench: Timing Simulation

181

Divider



```
library ieee; use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
```

```
entity lex9_tb is
end lex9_tb;
```

```
architecture arch_lex9_tb of lex9_tb is
component nrdiv8
port (
  clk, rst, start: in std_logic;
  dv: out std_logic;
  divdi, divsi: in std_logic_vector (7 downto 0);
  remdo: out std_logic_vector (8 downto 0);
  quoto: out std_logic_vector (7 downto 0);
end component;
```



Kuruvilla Varghese



Test bench: Timing Simulation

182

```
signal clk: std_logic := '0';
signal rst: std_logic := '1';
signal start: std_logic := '0';
signal dv: std_logic;
signal divdi: std_logic_vector(7 downto 0)
:= "11000000";
signal divsi: std_logic_vector(7 downto 0)
:= "00101101";
signal quoto: std_logic_vector(7 downto 0);
signal remdo: std_logic_vector(8 downto 0);
constant period: time := 200 ns;
constant duty_cycle: real := 0.5;
constant offset: time := 0 ns;
constant setup: time := 15 ns;
constant tco: time := 15 ns;
constant onerun: time := 2200 ns;
```

```
type tblk is record
  divdi: std_logic_vector(7 downto 0);
  divsi: std_logic_vector(7 downto 0);
  quoto: std_logic_vector(7 downto 0);
  remdo: std_logic_vector(8 downto 0);
end record;
type testv is array (natural range <>) of tblk;
constant tv: testv :=
(("11000000", "00101101",
  "00000100", "000001100"),
("00101101", "00000111",
  "00000110", "000000011"),
("11111111", "00000001",
  "11111111", "000000000"));
```

```
begin
```



Kuruvilla Varghese



Test bench: Timing Simulation

183

```
uut: nrdiv8 port map
  (clk => clk, rst => rst, start => start,
   dv => dv, divsi => divsi, divdi => divdi,
   remdo => remdo, quoto => quoto);
clock: process -- clock process for clk
begin
  wait for offset;
clock_loop : loop
  clk <= '0';
  wait for (period - (period * duty_cycle));
  clk <= '1'; wait for (period * duty_cycle);
end loop clock_loop;
end process;
```

```
test: process
variable vtv: tblk;
begin
  wait for (period - (period * duty_cycle) -
    setup);
  rst <= '0';
  wait for period;

  for i in tv'range loop
    vtv := tv(i);
    start <= '1';
    divdi <= vtv.divdi;
    divsi <= vtv.divsi;
    wait for period;
    start <= '0';
    wait for (onerun + setup + tco);
```



Kuruvilla Varghese



Test bench: Timing Simulation

184

```
assert quoto = vtv.quoto
report "Quotient is not what expected"
severity note;
assert remdo = vtv.remdo
report "Remainder is not what expected"
severity note;
wait for (period - (setup + tco));
end loop;
assert false report "Test over" severity
note;
wait;
end process;
end arch_lex9_tb;
```

- Instead of adding (period - tco + ts) at the end of the loop to setup the data correctly at the next iteration, code could wait for active clock edge at the beginning of each iteration

```
wait for (clk'event and clk = '1');
wait for (period - (period * duty_cycle) -
  setup);
```



Kuruvilla Varghese



Test bench: Timing Simulation

185

- You may do this at the end of iteration to check the output

```
wait for (clk'event and clk = '1');  
wait for (tco);
```

OR

```
wait for (dv'event and dv = '1');
```



Kuruvilla Varghese



Test bench: Algorithmic

186

- Instead of explicitly storing test vectors, it could be generated algorithmically in some cases.
- If the input test vectors follow a regular pattern or functions, test vectors could be generated using test bench code

-- Test Bench for 4 bit adder

```
library ieee;  
use ieee.std_logic_1164.all;  
use ieee.std_logic_unsigned.all;  
use ieee.std_logic_arith.all;  
  
entity addtb1 is  
end addtb1;  
  
architecture arch_addtb1 of addtb1 is  
  component adder4  
    port (a, b: in std_logic_vector(3 downto 0);  
          s: out std_logic_vector(4 downto 0));  
  end component;
```



Kuruvilla Varghese



Test bench: Algorithmic

187

```
signal a, b: std_logic_vector(3 downto 0);
signal s: std_logic_vector(4 downto 0);
constant width: integer := 16;
constant delay: time := 12 ns;
begin

-- component Instantiation
 uut: adder4 port map (a => a, b => b,
                      s => s);

test: process
begin
  for i in 0 to width-1 loop
    a <= conv_std_logic_vector(i, 4);
    for j in 0 to width-1 loop
      b <= conv_std_logic_vector(j, 4);
      wait for delay;
      assert s = conv_std_logic_vector(i+j, 5)
        report "Sum is not what is expected"
        severity note;
    end loop;
  end loop;
  assert false report "Test Over"
    severity note;
  wait;
end process;
end arch_addtb1;
```



Kuruvilla Varghese

