ECE381(CAD), Lecture 12b:

Attributes and Constraints

Mehdi Modarressi

Department of Electrical and Computer Engineering,

University of Tehran

Pictures and examples are taken from the slides of "VHDL: Analysis & Modeling of Digital Systems" and also from "VHDL by Example"

Attributes

- Attribute: provide functions for extracting run-time information from objects
- There are two classes of attributes:
  - Predefined as a part of the 1076 standard
  - Introduced outside of the standard:
    - by the designer
    - by the design tool supplier (like Xilinx or Altera)

Attributes

- Attributes can be applied to arrays, types, signals, and entities
- We have already seen some array and signal attributes
  - LENGTH, RANGE, EVENT,…
- Attribute format:

```
array_or_type_or_signal_or_entity_name'ATTRIBUTE_NAME
```

- Example:
  - signal'EVENT
  - Vector'RANGE

Signal attributes

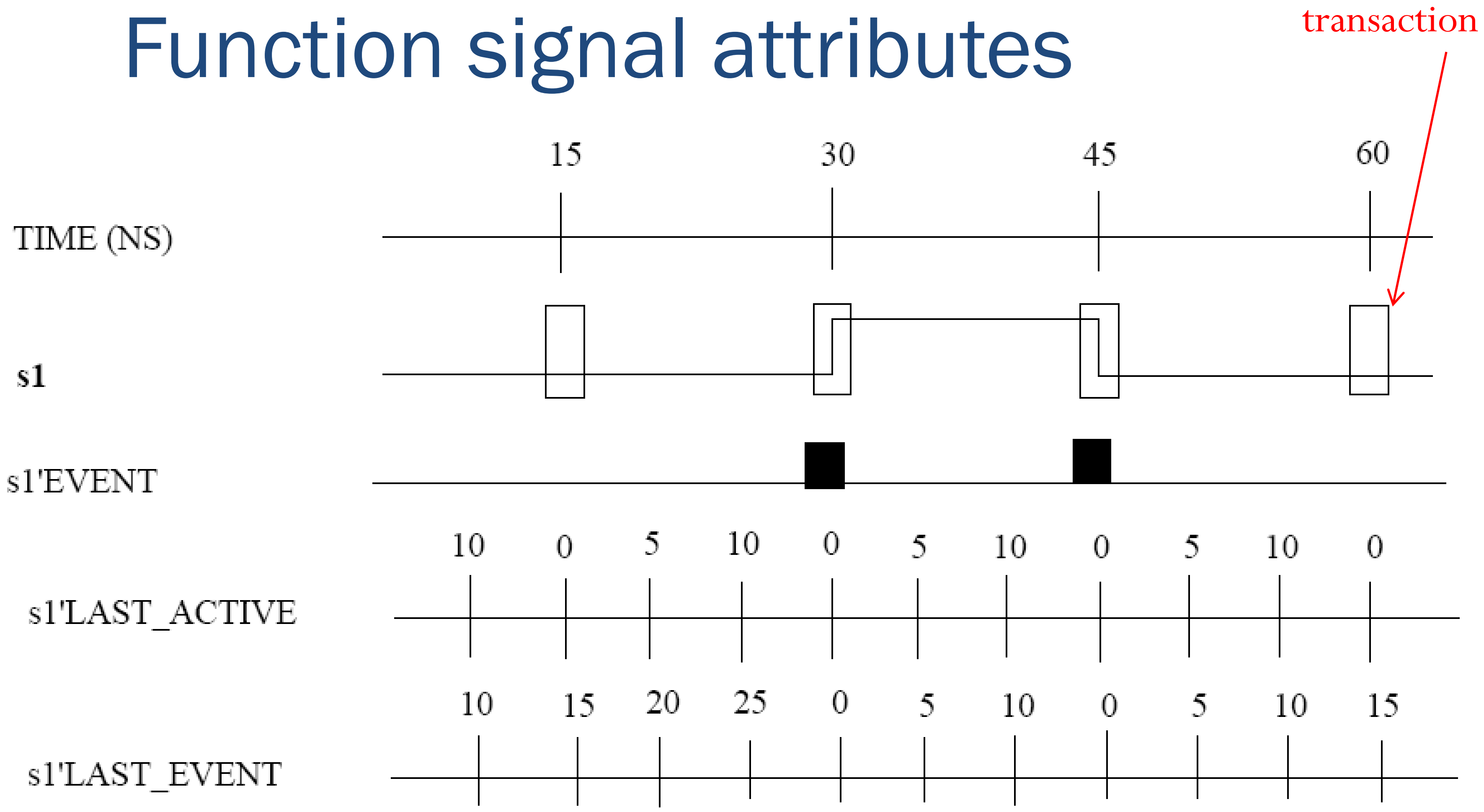
- We just study signal attributes
- For a complete description of attributes, see:
  1. "VHDL: Analysis & Modeling of Digital Systems": Chapter 7.5 to 7.7
  2. "VHDL by Example": Chapter 6
  3. "Designers Guide To VHDL": Chapter 2.4 and 4.1 and 8

- Two kind of signal attributes:
  - Function attributes
    - Return information about the behavior of signals
  - Signal kind attributes
    - Create special signals, based on other signals

Function signal attributes

- Five attributes
- S'EVENT:
  - Returns true if an event occurred during the current time; otherwise, returns false
- S'ACTIVE
  - Returns true if a transaction occurred during the current time; otherwise, returns false
- S'LAST_EVENT
  - Returns time elapsed since the previous event transition of signal
- S'LAST_VALUE
  - Returns previous value of S before the last event
- S'LAST_ACTIVE
  - Returns time elapsed since the previous transaction of signal

Function signal attributes



Function signal attributes

```
IF ( clk = '1' ) AND ( clk'EVENT )
      and ( clk'LAST_VALUE = '0') THEN
 q <= d;
END IF;
```

- Clock falling edge detection
- Just considers 0→1 as falling edge not x→1 and z→1

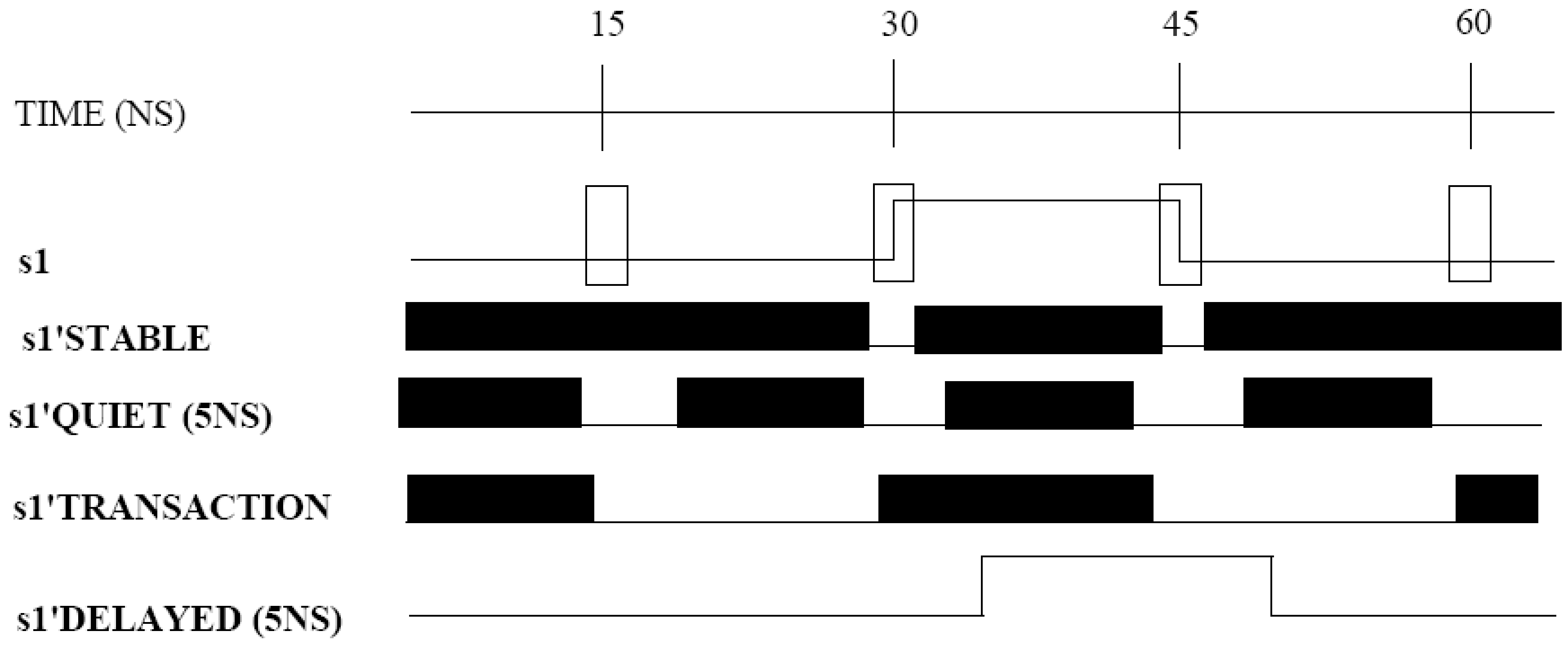
Signal kind attributes

- Four attributes
- s’DELAYED [(time)]
  - Creates a signal of the same type as the reference signal that follows the reference signal, delayed by the time of the optional time expression
- s’STABLE [(time)]
  - Creates a boolean signal that is true whenever the reference signal has had no events for the time specified by the optional time expression
- s’QUIET [(time)]
  - Creates a boolean signal that is true whenever the reference signal has had no transactions or events for the time specified by the optional time expression
- s’TRANSACTION
  - Creates a signal of type BIT that toggles its value for every transaction or event that occurs on s

Signal kind attributes

- Four attributes
- s’DELAYED [(time)]
  - Creates a signal of the same type as the reference signal that follows the reference signal, delayed by the time of the optional time expression
- s’STABLE [(time)]
  - Creates a boolean signal that is true whenever the reference signal has had no events for the time specified by the optional time expression
- s’QUIET [(time)]
  - Creates a boolean signal that is true whenever the reference signal has had no transactions or events for the time specified by the optional time expression
- s’TRANSACTION
  - Creates a signal of type BIT that toggles its value for every transaction or event that occurs on s

Value signal attributes



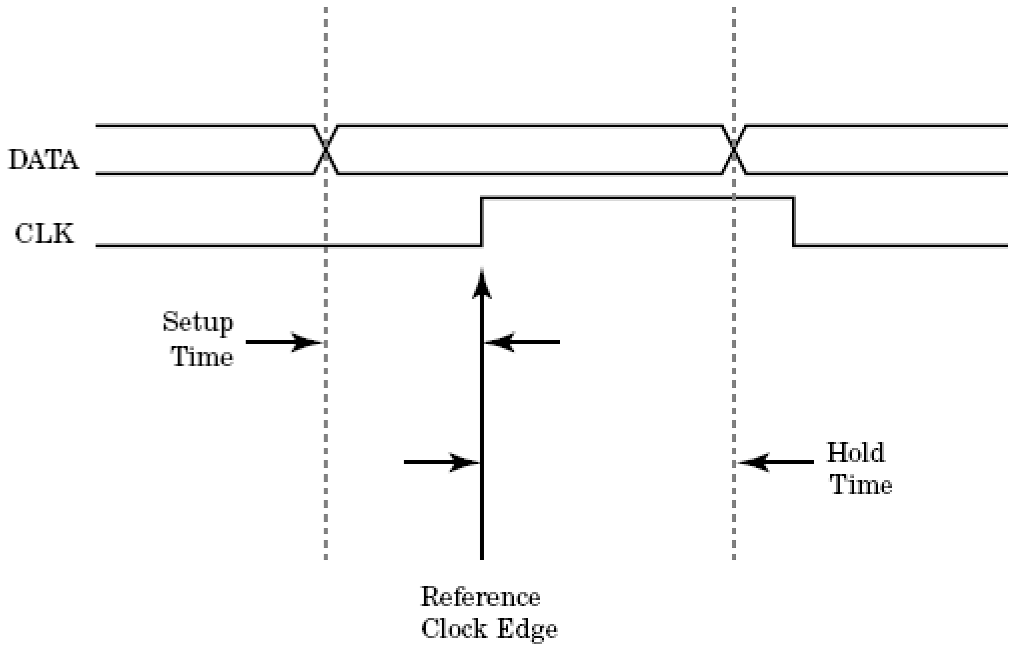
Signal attributes

- Two ways to detect positive edge of the clock

```
IF (( clk'EVENT ) AND ( clk = '1' ) AND
               ( clk'LAST_VALUE = '0' )) THEN
   .
   .   -- DO PROCESSING
   .
END IF;


IF (( NOT( clk'STABLE) ) AND ( clk = '1' ) AND
        (
                       clk'LAST_VALUE = '0' )) THEN
   .
   .   --- DO PROCESSING
   .
END IF;
```

Function signal attributes- setup time



- Setup time: the amount of time during which the data input is not allowed to change before the clock edge

Function signal attributes- setup time

```
 LIBRARY IEEE;
 USE IEEE.std_logic_1164.ALL;
 ENTITY dff IS

 GENERIC ( setup_time, hold_time : TIME );
 PORT( d, clk : IN std_logic;
   q : OUT std_logic);
BEGIN
 setup_check : PROCESS ( clk )
 BEGIN
  IF ( clk = ’1’ ) and ( clk’EVENT ) THEN
   ASSERT ( d’LAST_EVENT >= setup_time )
    REPORT “setup violation”
    SEVERITY ERROR;
  END IF;
 END PROCESS setup_check;
END dff;
```

```
ARCHITECTURE dff_behave OF dff IS
BEGIN
 dff_process : PROCESS ( clk )
 BEGIN
  IF ( clk = ’1’ ) AND ( clk’EVENT ) THEN
   q <= d;
  END IF;
 END PROCESS dff_process;
END dff_behave;
```

- Setup time: the amount of time during which the data input is not allowed to change before the clock edge

```
 LIBRARY IEEE;
 USE IEEE.std_logic_1164.ALL;
 ENTITY dff IS
 GENERIC ( setup_time, hold_time : TIME );
 PORT( d, clk : IN std_logic;
    q : OUT std_logic);
BEGIN
 setup_check : PROCESS ( clk )
 BEGIN
  IF ( clk = '1' ) and ( clk'EVENT ) THEN
   ASSERT ( d'LAST_EVENT >= setup_time )
    REPORT "setup violation"
    SEVERITY ERROR;
  END IF;
 END PROCESS setup_check;
END dff;
```

User-defined attributes

- We can define an attribute for an object in VHDL
- They do not have simulation semantics, so it is up to the user to define them and use them in accordance with the way they are defined
- Attribute are declared using an attribute declaration
- An attribute declaration identifies a name as an attribute with a given type

User-defined attributes

- The first step in defining an attribute is to declare the name and type of an attribute, using an *attribute declaration*
- Example: declaring *sub_dir* as an attribute that can take values of STRING type

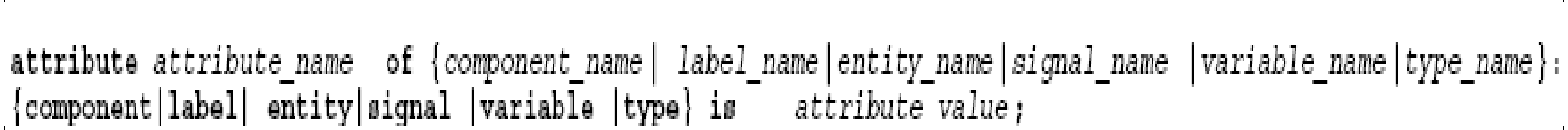
```
ATTRIBUTE sub_dir : STRING;
```

- Then it can be used with objects

```
ATTRIBUTE sub_dir OF multiplexer: ENTITY IS "/user/vhdl";
```

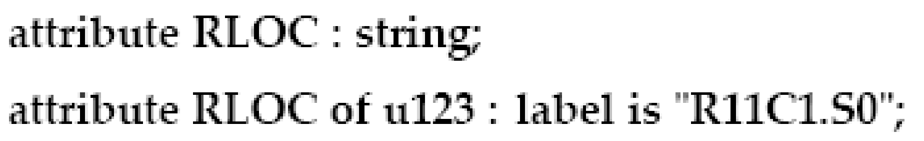
General format

- User-defined attributes can be defined for signals, variables, entities, types, labels, and components

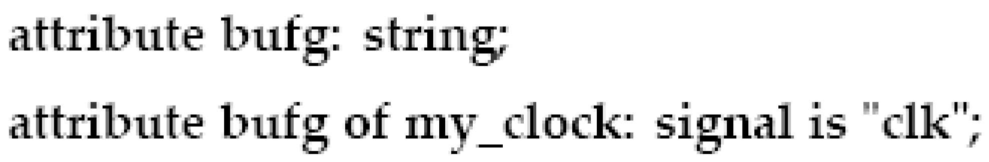


```
attribute attribute_name  of {component_name| label_name|entity_name|signal_name |variable_name|type_name}:
{component|label| entity|signal |variable |type} is    attribute value;
```

- Examples:



```
attribute RLOC : string;
attribute RLOC of u123 : label is "R11C1.S0";
```



```
attribute bufg: string;
attribute bufg of my_clock: signal is "clk";
```

User-defined attributes

- One of the usages of user-defined attributes is in synthesis
- To give special directives to synthesis tool how to synthesize an object
- In Xilinx there is set of reach user defined attributes to manage the synthesis and implementation process
- These guidelines are generally known as Constraints
  - Synthesis constraints
  - Implementation constraints

Synthesis constraints

- Synthesis constraints direct the synthesis tool optimization technique for a particular design or piece of HDL code
- Example:
  - Rules for how to synthesize RAM (**RAM_STYLE**) or Multiplier (**MULT_STYLE**)

Implementation constraints

- Implementation constraints are instructions given to the FPGA implementation tools to direct the mapping, placement, timing,..
- Implementation constraints are generally placed in the UCF (user constraint file) file, but may exist in the VHDL code by attributes
- Examples of implementation constraints are:
  - LOC (placement) constraints
  - PERIOD (timing) constraints

Example: multiplier implementation

- The Multiplier Style (**MULT_STYLE**) constraint controls the way the multipliers are implemented

```
Attribute MULT_STYLE: string;
Attribute MULT_STYLE of  c: signal is block;
.
.
C<=a*b;
```

- Allowed values of the attribute:
- block – dedicated multiplier
- lut - LUT-based multiplier
- pipe_block – pipelined dedicated multiplier
- pipe_lut – pipelined LUT-based multiplier
- auto – automatic choice by the synthesis tool

Example: RAM implementation

- The RAM Style (**RAM_STYLE**) constraint controls the way the inferred RAM is implemented

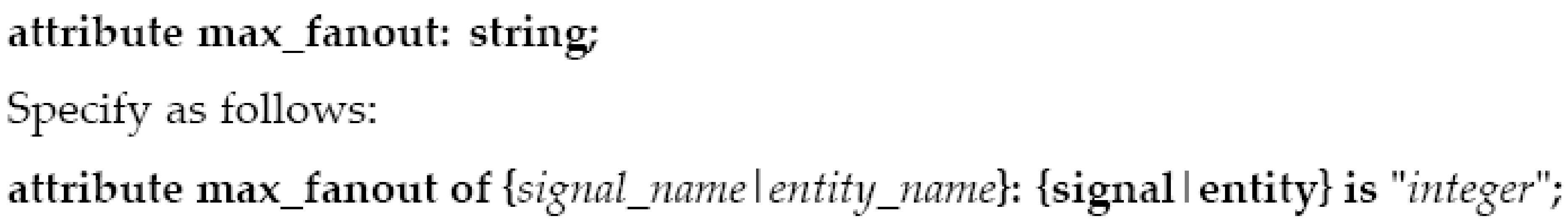
```
attribute ram_style: string;
```

- Used as:

```
attribute ram_style of {signal_name|entity_name} : {signal|entity} is
"{auto|block|distributed|pipe_distributed|block_power1|block_power2}";
```

Example: limiting fan-out

- Limits the maximum fan-out of a signal
- Better speed and routing



attribute max_fanout: string;

Specify as follows:

attribute max_fanout of {*signal_name* | *entity_name*}**: {signal | entity} is** "*integer*";

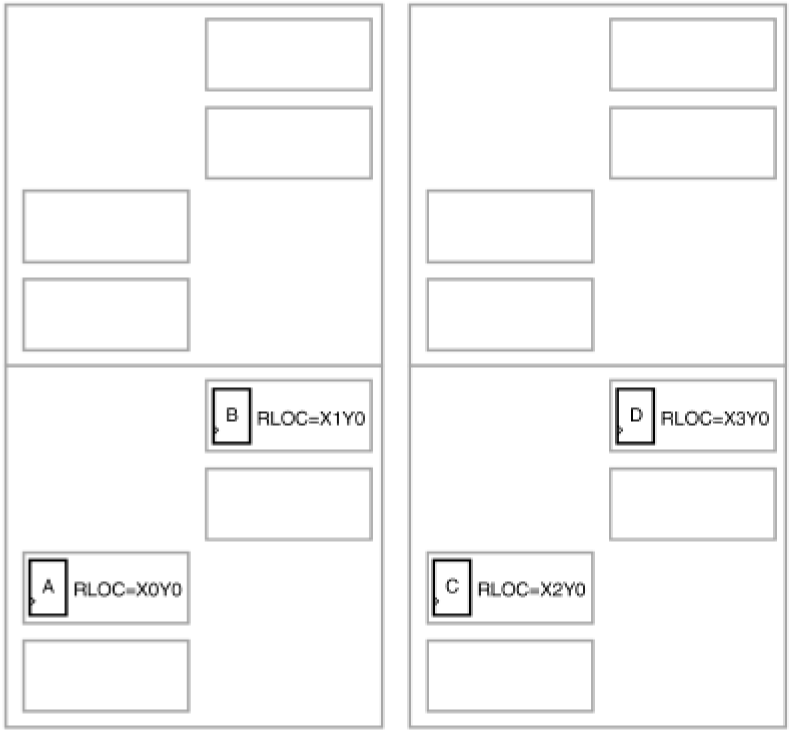
LOC attribute

- LOC defines where a design element can be placed within an FPGA device.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library UNISIM;
use UNISIM.VCOMPONENTS.ALL;

entity top is
  port ( led : out STD_LOGIC; switch : in STD_LOGIC );
end entity;

architecture STRUCTURE of top is
  attribute LOC : string;
  attribute LOC of LUT_INST : label is "SLICE_X0Y0";
begin
  LUT_INST : LUT1
    generic map ( INIT => "01" )
    port map ( O  => led, I0 => switch );
end architecture;
```



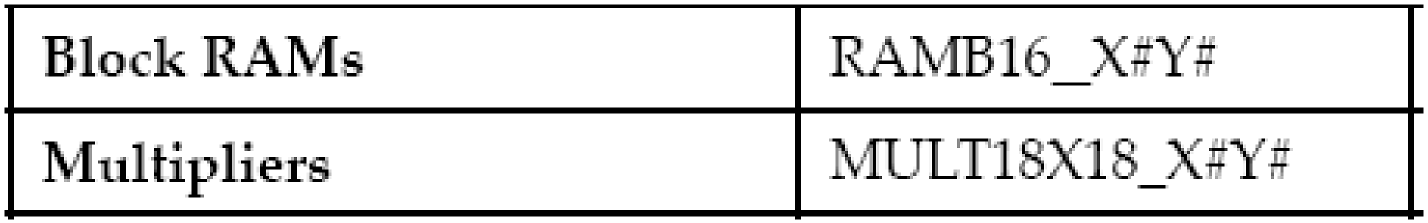
LOC constraint

- FPGA devices use a Cartesian-based XY designator at the slice level
- Location specification uses the form: **SLICE_X***m*Y*n*.
- The XY slice grid starts as X0Y0 in the lower left CLB tile of the chip
  - The X values start at 0 and increase horizontally to the right in the CLB row, with two different X values per CLB
  - The Y values start at 0 and increase vertically up in the CLB column, with two different Y values per CLB.
  - Example:

SLICE_X0Y0	First (bottom) slice of the CLB in the lower left corner of the chip
SLICE_X2Y1	Second slice of the bottom CLB in CLB column 2
SLICE _X50Y125	Slice located 125 slices up from and 50 slices to the right of SLICE_X0Y0

LOC for other components

- FPGA block RAMs and multipliers have their own specification different from the SLICE specifications
- Therefore, the location value must start with **SLICE**, **RAMB**, or **MULT**.
- Format:



Block RAMs	RAMB16_X#Y#
Multipliers	MULT18X18_X#Y#

- Example:

```
Attribute LOC: string;
Attribute LOC of  c: signal is MULT18x18_X0Y0;
C<=a*b;
```

Reference

- See “**Xilinx Constraints Guide.pdf**” for a complete set of constraints