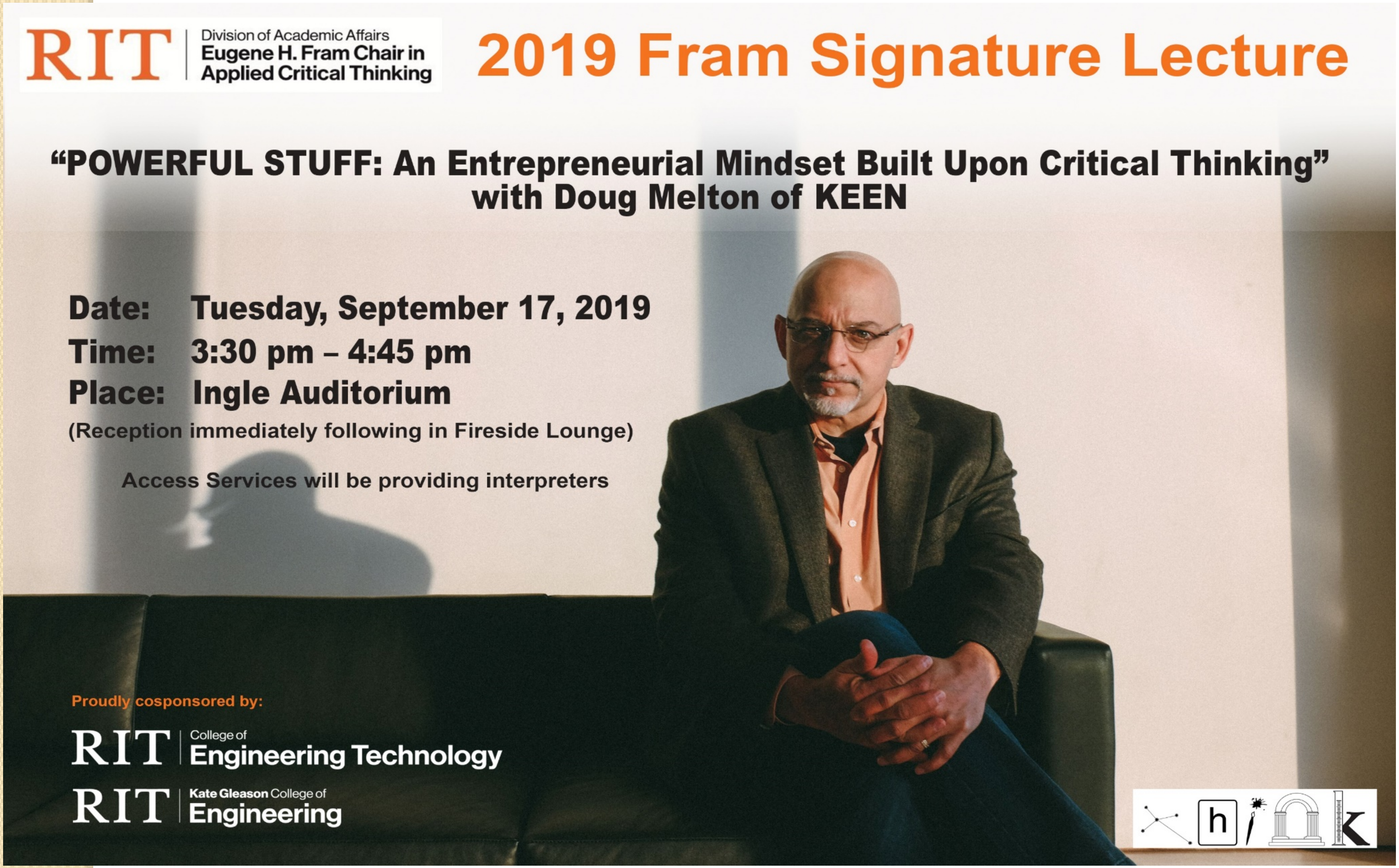
Introduction to Sequential VHDL

Announcements

- Homework #2 due Today
- Homework #3 posted – due next week
- Reading Assignment
  - Ch. 6 sections 3-4, 7-9
- Free homework grade for critical thinking lecture
  - Don’t forget to scan in



RIT | Division of Academic Affairs
Eugene H. Fram Chair in
Applied Critical Thinking

2019 Fram Signature Lecture

"POWERFUL STUFF: An Entrepreneurial Mindset Built Upon Critical Thinking" with Doug Melton of KEEN

Date: Tuesday, September 17, 2019
Time: 3:30 pm – 4:45 pm
Place: Ingle Auditorium
(Reception immediately following in Fireside Lounge)

Access Services will be providing interpreters

Proudly cosponsored by:

RIT | College of Engineering Technology

RIT | Kate Gleason College of Engineering

Before we talk about sequential

- **Constants**
  - function like constants in a programming language
    - Their value cannot change during execution
  - Declared with signals in the architecture's 'declarative section'
    - 'declarative section is after the ARCHITECTURE line, before BEGIN
  - Syntax:
    - constant constant_name : type := value;

Note the syntax here

VHDL Constants

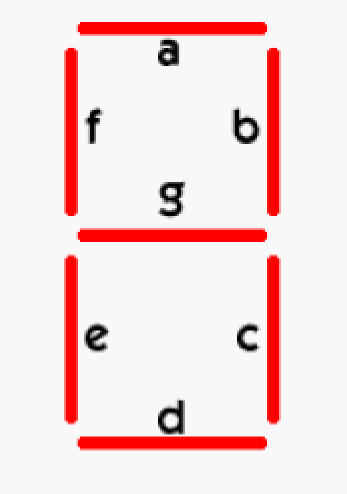
- Example:

```
ARCHITECTURE example OF constants IS
   CONSTANT SEVEN : STD_LOGIC_VECTOR(6 downto 0) := “1111000”;
BEGIN
    HEX0 <= SEVEN;
               :
               :
```

- Write the constant declaration for an 8-bit standard logic vector equivalent to AA$_H$

Constants with 7-segment display

- Seven Segment displays
  - HEX0(0) = a
  - HEX0(1) = b
  - :
  - :
  - HEX0(6) = g



: 1 is OFF

constant ZERO : std_logic_vector(6 downto 0) := "1000000";

segment g is off, all other segments are on

Sequential VHDL code

- Consists of one or more process statements
  - Each process is a concurrent statement
  - All processes execute simultaneously
  - Processes communicate using signals
  - Statements within a process are executed sequentially
  - Most common structures are:
    - If-then-else
    - case

Processes

- A concurrent statement that is comprised of sequential statements
  - Sequential statements are evaluated in sequence and their order is critical
  - *NOTE* all signal assignments are made concurrently at the END of process execution. Signals assigned in the process are not available for reading in the same process

```
Example: Process()
  BEGIN
    a <= ‘1’;
    If (a= ‘1’) then ….
    Else ….
END process;
```

Which branch is taken?

Processes

- Syntax

```
[process_label:]  PROCESS [(sensitivity_list)] [IS]
    {process_declarative_item}
BEGIN
     {sequential_statement}
END PROCESS [process_label];
```

- Sensitivity list
  - List of signals that the process is sensitive to
  - The process will execute when one of the signals in the list has an event (changes value)
  - All signals that are read in the process should be included
  - Do not include signals not used in the process

Sensitivity list (con’t)

- Process execution in simulation is in an endless loop
    - Execution starts when a signal in the sensitivity list has an event
    - Execution suspends when the last statement is completed
- In simulation, all processes are evaluated at the same time - *concurrency*

Case Statements

- Use to select one sequence of statements for execution from a number of alternatives
  - Similar to a C ‘switch’ structure
  - Synthesizes to a MUX
  - Selection is based on the value of a single expression

```
CASE (expression) IS
  WHEN choices => sequence of statements
  {WHEN choices => sequence of statements}
END CASE;
```

  - A list of choices can be associated with one branch
    - Separate choices with |

Case example

```
ENTITY xor_2 IS
   PORT( a, b : IN STD_LOGIC;
          c    : OUT STD_LOGIC);
  END xor_2;

ARCHITECTURE behavioral OF xor_2 IS
    SIGNAL inputs : STD_LOGIC_VECTOR(1 DOWNTO 0);
BEGIN
    inputs <= a & b;
    ex_or:  PROCESS(inputs)
       BEGIN
          CASE  inputs IS
              WHEN “01” => c <= ‘1’;
              WHEN “10” => c <= ‘1’;
              WHEN OTHERS => c <= ‘0’;
          END CASE;
       END PROCESS;
END behavioral;
```

Case Example (shortened)

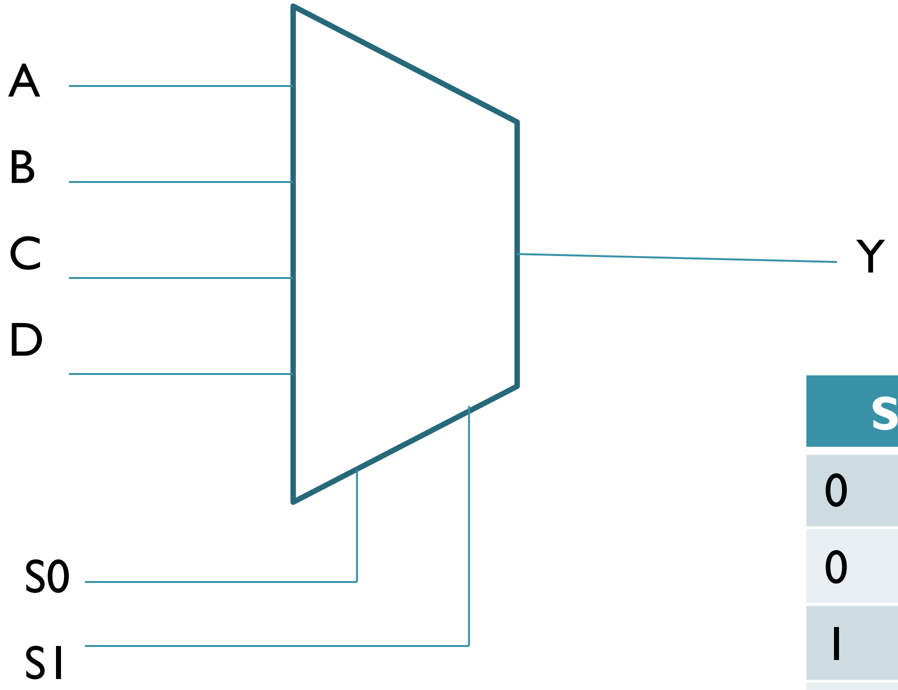
```
ENTITY xor_2 IS
   PORT( a, b : IN STD_LOGIC;
          c    : OUT STD_LOGIC);
  END xor_2;

ARCHITECTURE behavioral OF xor_2 IS
   SIGNAL inputs : STD_LOGIC_VECTOR(1 DOWNTO 0);
BEGIN
    inputs <= a & b;
    ex_or:  PROCESS(inputs)
       BEGIN
          CASE  inputs IS
              WHEN “01”  |“10” => c <= ‘1’;
              WHEN OTHERS => c <= ‘0’;
          END CASE;
       END PROCESS;
END behavioral;
```

Case (con't)

- Others choice
  - Some compilers require WHEN OTHERS => as the last choice
  - Some only require it if the set of choices does not cover every possible value of the case expression
  - CYA and always include it
  - It has to be the last branch

Mux Example

- Consider the following Mux



S1	S0	Y
0	0	A
0	1	B
1	0	C
1	1	D

- Write the architecture
  - Using a case statement

IF Statements

- Selects one or none of its alternative sequence of statements depending on the value of each branch's condition
  - Priority based on order of if – elsif conditions

```
IF condition THEN
  sequence of statements
{ELSIF condition THEN
  sequence of statements}
[ELSE
  sequence of statements]
END IF;
```

If Statements (con't)

- The condition must evaluate to a boolean true or false
  - IF ((a='1') AND (b='0')) THEN
- The ELSE statement is not necessarily optional
  - If a signal is assigned a value in any branch, it must be assigned a value in all branches
  - When a signal is not assigned a value in all branches, a latch is inferred
  - Inferred latches are bad. Don't worry, the synthesis tool will give you a warning

If Statements (con’t)

```
ARCHITECTURE xyz OF xor_2 IS
BEGIN
   PROCESS (a, b)
    BEGIN
       IF a /= b then
         c <= ‘1’;
       ELSE
         c <= ‘0’;
       END IF;
   END PROCESS;
END xyz;
```

```
ARCHITECTURE xyz OF xor_2 IS
BEGIN
   PROCESS (a, b)
    BEGIN
       IF a /= b then
         c <= ‘1’;
       END IF;
   END PROCESS;
END xyz;
```

Signal assignments in processes

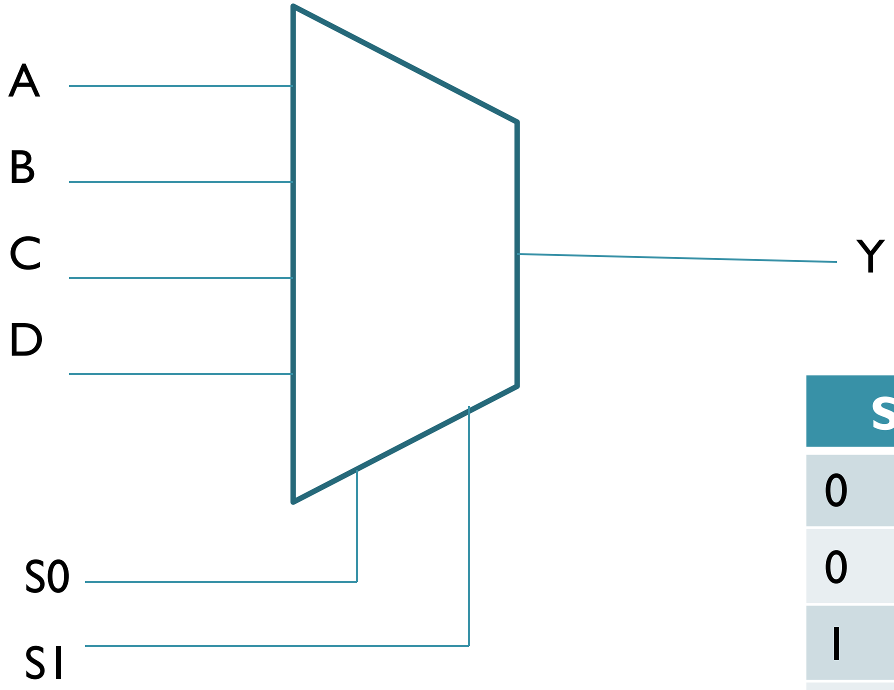
- Signals take the last value they are assigned in a process
- Assignments are made concurrently at the end of process execution
- Default assignments can be used to shorten IF statements

```
ARCHITECTURE xyz OF xor_2 IS
BEGIN
    PROCESS (a, b)
    BEGIN
         c <= ‘0’;   -- this is the default assignment
         IF a /= b then
            c <= ‘1’;  -- c is only re-assigned if a /= b
         END IF;
    END PROCESS;
END xyz;
```

- In this example it is okay to have an IF without and ELSE

Mux Example

- Consider the following Mux



S1	S0	Y
0	0	A
0	1	B
1	0	C
1	1	D

- Write the architecture
  - Using an if statement