Variables and Attributes

Useful in generics, loops and subprograms

Announcements

- Homework #12 due Wednesday
- Quiz Wednesday – loops and generics
- Reminders:
  - Complete the course evaluation on line
  - Complete your teammate evaluations by 12/9

Variable Data Type

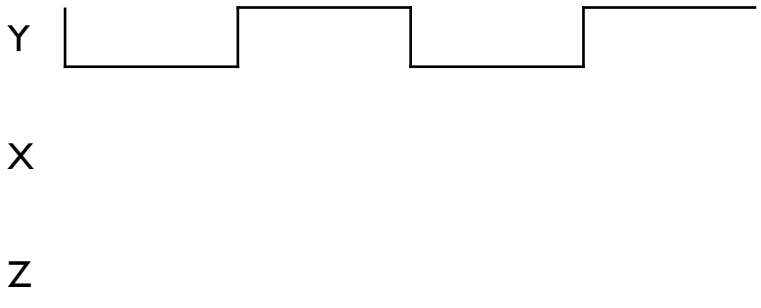
- Used to store intermediate values in a process or subprogram
- Declared like a signal
  - VARIABLE count : std_logic_vector(3 downto 0);
- Assignments made with :=
  - Count := “0000”;

Variables

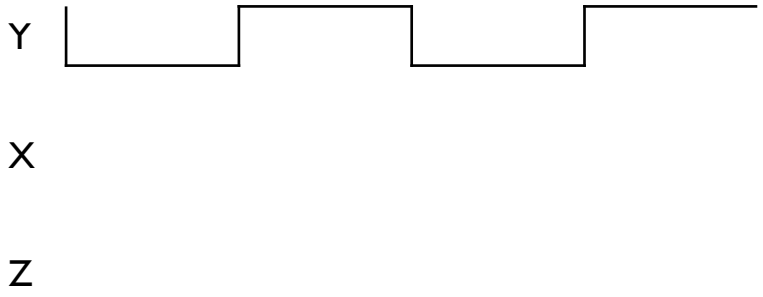
- Declared in the process before the BEGIN statement
- Take the value assigned to them immediately
  - Like a traditional programming language
  - Remember signals don't get assigned until the end of the process
- Order matters because they are updated immediately

Variable Example in Process

```
signal x,y,z : std_logic;
process (y)
    begin
        x<=y;
        z<=not x;
end process;
```




```
signal y : std_logic;
process (y)
    variable x,z : std_logic;
    begin
        x:=y;
        z:=not x;
    end process;
```

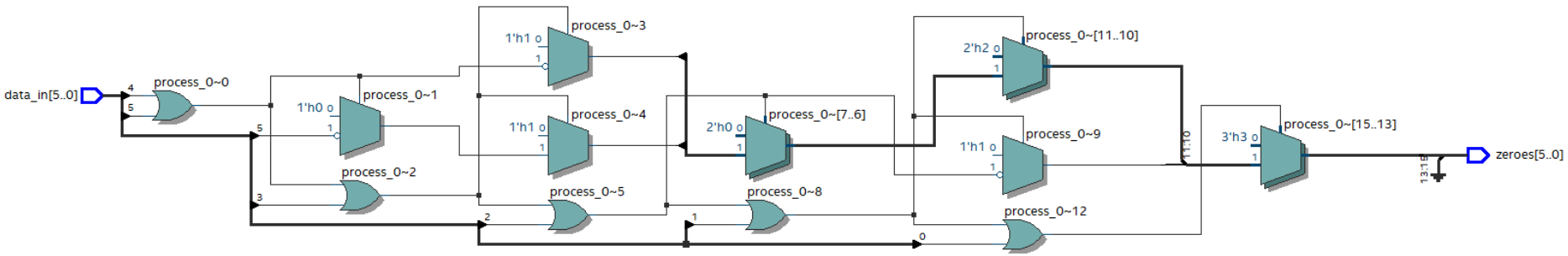


Important Distinction

- A signal represents
  - A wire if it is assigned in a concurrent statement
  - A flip-flop if it is assigned under a clock
- A variable does not represent a hardware component.
  - While code that contains a variable will synthesize ….
  - The variable does not become a wire or register
- Be careful when using variables

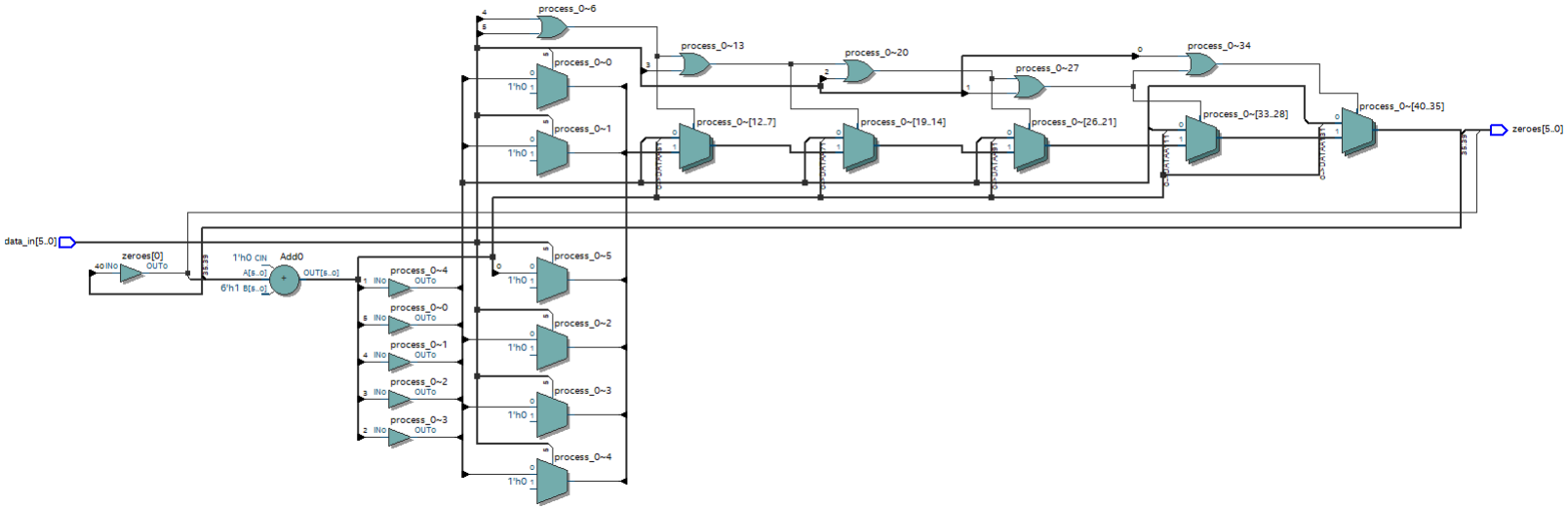
Loop Example - variable

```
--This modules counts the number of zeros in a std_logic_vector
--It loops through the vector and counts the zeros. Once a 1 is found
--    the loop exits

ENTITY leading_zeros IS
   GENERIC (wide : integer := 6);  --input vector length is generic
   PORT (data_in : IN STD_LOGIC_VECTOR(wide-1 DOWNTO 0);
         zeroes  : OUT STD_LOGIC_VECTOR(wide-1 downto 0)); --this is longer than it needs to be.
                                        --in acuality it just needs to be log2(wide) + 1 in length
END leading_zeros;
ARCHITECTURE behavior of leading_zeros IS
BEGIN
   PROCESS(data_in)                             --loop has to go in a process
      variable count: std_logic_vector(wide-1 downto 0);
   BEGIN
      count := (others => '0');
      for i IN data_in'LEFT downto 0 Loop    --use 'LEFT attribute since the length of data_in is generic
                                             --could have also used wide-1 downto 0
         if data_in(i) = '0' then
            count := count + 1; --increment the count with each 0 found
         else
            EXIT;              --exit loop once a 1 is found
          END if;
      END LOOP;
      zeroes <= count;
   END PROCESS;
END behavior;
```



Loop Example - signal

```
--This modules counts the number of zeros in a std_logic_vector
--It loops through the vector and counts the zeros. Once a 1 is found
--    the loop exits

ENTITY leading_zeros IS
   GENERIC (wide : integer := 6);  --input vector length is generic
   PORT (data_in : IN STD_LOGIC_VECTOR(wide-1 DOWNTO 0);
         zeroes  : OUT STD_LOGIC_VECTOR(wide-1 DOWNTO 0)); --this is longer than it needs to be.
                                    --in acuality it just needs to be log2(wide) + 1 in length
END leading_zeros;
ARCHITECTURE behavior of leading_zeros IS
   signal count: std_logic_vector(wide-1 downto 0);
BEGIN
   PROCESS(data_in)                          --loop has to go in a process
     BEGIN
      count <= (others => '0');
      for i IN data_in'LEFT downto 0 LOOP    --use 'LEFT attribute since the length of data_in is generic
                                             --could have also used wide-1 downto 0
         if data_in(i) = '0' then
            count <= count + 1; --increment the count with each 0 found
         else
            EXIT;              --exit loop once a 1 is found
         END if;
      END LOOP;
      zeroes <= count;
   END PROCESS;
END behavior;
```



Attributes

- Used to give information about the values included in the type
- Useful in creating reusable code
- Format:
  - T'attribute
    - T represents any scalar type or subtype
  - A'attribute
    - A represents any array or constrained array type
  - S'attribute
    - S represents any signal
  - E'attribute
    - E represents a named entity

T'attribute

```
T'BASE       is the base type of the type T
T'LEFT       is the leftmost value of type T. (Largest if downto)
T'RIGHT      is the rightmost value of type T. (Smallest if downto)
T'HIGH       is the highest value of type T.
T'LOW        is the lowest value of type T.
T'ASCENDING  is boolean true if range of T defined with to .
T'IMAGE(X)   is a string representation of X that is of type T.
T'VALUE(X)   is a value of type T converted from the string X.
T'POS(X)     is the integer position of X in the discrete type T.
T'VAL(X)     is the value of discrete type T at integer position X.
T'SUCC(X)    is the value of discrete type T that is the successor of X.
T'PRED(X)    is the value of discrete type T that is the predecessor of X.
T'LEFTOF(X)  is the value of discrete type T that is left of X.
T'RIGHTOF(X) is the value of discrete type T that is right of X.
```

A'attribute

```
A'LEFT          is the leftmost subscript of array A or constrained array type.
A'LEFT(N)       is the leftmost subscript of dimension N of array A.
A'RIGHT         is the rightmost subscript of array A or constrained array type.
A'RIGHT(N)      is the rightmost subscript of dimension N of array A.
A'HIGH          is the highest subscript of array A or constrained array type.
A'HIGH(N)       is the highest subscript of dimension N of array A.
A'LOW           is the lowest subscript of array A or constrained array type.
A'LOW(N)        is the lowest subscript of dimension N of array A.
A'RANGE         is the range  A'LEFT to A'RIGHT  or  A'LEFT downto A'RIGHT .
A'RANGE(N)      is the range of dimension N of A.
A'REVERSE_RANGE  is the range of A with to and downto reversed.
A'REVERSE_RANGE(N)  is the REVERSE_RANGE of dimension N of array A.
A'LENGTH        is the integer value of the number of elements in array A.
A'LENGTH(N)     is the number of elements of dimension N of array A.
A'ASCENDING     is boolean true if range of A defined with to .
A'ASCENDING(N)  is boolean true if dimension N of array A defined with to .
```

Array Attributes

- Signal bus : std_logic_vector(7 downto 0);
  - Bus'left =
  - Bus'right =
  - Bus'high =
  - Bus'low =
  - Bus'range =
  - Bus'reverse_range =
  - Bus'length =
  - Bus'ascending (Boolean) =

Other uses for Attributes

```
SIGNAL sel: std_logic_vector(3 DOWNTO 0) := (others => '0');

Tb: PROCESS
        BEGIN
            for i in 0 to ((2**(sel'length))-1) loop
                    count := 1;
                    sel <= std_logic_vector(to_unsigned(i,sel'length);
                    wait for 100*clk_period;
            end loop;
    END PROCESS;
```

*why is this better than hardcoding15 and 4?

S'attribute

```
S'DELAYED(t) is the signal value of S at time now - t .
S'STABLE     is true if no event is occurring on signal S.
S'STABLE(t)  is true if no even has occurred on signal S for t units of time.
S'QUIET      is true if signal S is quiet. (no event this simulation cycle)
S'QUIET(t)   is true if signal S has been quiet for t units of time.
S'TRANSACTION  is a bit signal, the inverse of previous value each cycle S is active.
S'EVENT      is true if signal S has had an event this simulation cycle.
S'ACTIVE     is true if signal S is active during current simulation cycle.
S'LAST_EVENT is the time since the last event on signal S.
S'LAST_ACTIVE  is the time since signal S was last active.
S'LAST_VALUE is the previous value of signal S.
S'DRIVING    is false only if the current driver of S is a null transaction.
S'DRIVING_VALUE  is the current driving value of signal S.
```