ECE 280 / CSE 280
Digital Design Laboratory
Lecture 5

Memories

Memories

<table>
<tr><th colspan="3">Read-Write Memory</th><th>Read-Only Memory</th></tr>
<tr><th colspan="2">Volatile Memory</th><th>Non-volatile Memory</th><td rowspan="3">Mask-Programmed ROM</td></tr>
<tr><th>Random Access</th><th>Sequential Access</th><td rowspan="2">EPROM
EEPROM
FLASH</td></tr>
<tr><td>DRAM
SRAM</td><td>FIFO
LIFO</td></tr>
</table>

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

VHDL code for ROM

```
library IEEE;
use IEEE.std_logic_1164.all;

ENTITY rom8x4 IS
 PORT (
 addr: in std_logic_vector(2 downto 0);
 q: out std_logic_vector(3 downto 0));
END rom8x4;
```

```
ARCHITECTURE behav OF rom8x4 IS
BEGIN

PROCESS(addr)
BEGIN

  CASE addr IS
    when "000" => q <= "0001";
    when "001" => q <= "0000";
    when "010" => q <= "0111";
    when "011" => q <= "1101";
    when "100" => q <= "1000";
    when "101" => q <= "1100";
    when "110" => q <= "0110";
    when "111" => q <= "1011";
    when others  =>  NULL;
  END case;
 END process;
END behav;
```


John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

ROMS in Xilinx library

- **ROM16x1**
- **ROM32x1**
- **ROM64x1**
- **ROM128x1**
- **ROM256x1**

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

ROMS in Xilinx library

```
component ROM16X1
-- synthesis translate_off
  generic (INIT : bit_vector := X"16");
-- synthesis translate_on
  port (O : out STD_ULOGIC;
        A0 : in STD_ULOGIC;
        A1 : in STD_ULOGIC;
        A2 : in STD_ULOGIC;
        A3 : in STD_ULOGIC);
end component;

architecture beh of rom is
begin
ROM16X1_INSTANCE_NAME : ROM16X1
-- synthesis translate_off
  generic map (INIT => hex_value )
-- synthesis translate_on
  port map (O => user_O, A0 => user_A0 , A1 => user_A1,
            A2 => user_A2, A3 => user_A3 );
end architecture
```


John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

How do we use ROM

- VHDL components in a VHDL design
- Graphical components in a schematic
- Example using VHDL

```
architecture beh of proc is
 signal ABUS : std_logic_vector(3 downto 0);
 signal DBUS : std_logic_vector(1 downto 0);
begin
bit0 : ROM16X1
-- synthesis translate_off
  generic map (INIT => "1010 1110 0001 0001" )
-- synthesis translate_on
  port map (O => DBUS(0), A0 => ABUS(0), A1 => ABUS(1),
            A2 => ABUS(2), A3 => ABUS(3) );
bit1 : ROM16X1
-- synthesis translate_off
  generic map (INIT => X"1101 1010 1111 0101" )
-- synthesis translate_on
  port map (O => DBUS(1), A0 => ABUS(0), A1 => ABUS(1),
            A2 => ABUS(2), A3 => ABUS(3) );
```


John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

How do we use ROM

```
  rom_inc : process(CLK) is
  begin
    if (CLK'event and CLK='1') then
       ABUS <= ABUS + 1;
    end if;
  end process;

 jump : process(DBUS) is
  begin
    if (DBUS="11") then
       ABUS <= "0000";
    end if;
  end;

end architecture
```


John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

Random Access Memories (RAMs)

- Read/Write memory
- Types:
  - Static RAM (SRAM):
    - Once a word is written at a location, it remains stored as long as power is applied to the chip, unless the same location is written again.
    - Fast speed, but their cost per bit higher.
    - Application: Cache memories in Microprocessor
  - Dynamic RAM (DRAM):
    - The data stored at each location must be periodically refreshed by reading it and then writing it back again, or else it disappears.
    - Their density is greater and their cost per bit lower, but the speed is slower.

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

RAMS in Xilinx library

- Static Block RAMs
- Single port
  - RAMB4_S1, RAMB4_S2, RAMB4_S8, RAMB4_S16
- Dual port
  - RAMB4_S1_S1, RAMB4_S1_S2, … RAMB4_S16_S16

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

RAMS in Xilinx library

```
component RAMB4_Sn
-- synthesis translate_off
  generic ( INIT_00 : bit_vector :=

X"0000000000000000000000000000000000000000000000000000000000000000";
            INIT_01 : bit_vector :=

X"0000000000000000000000000000000000000000000000000000000000000000";
          . . .
          );
-- synthesis translate_on
  port (DO : out STD_LOGIC_VECTOR (0 downto 0);
        ADDR : in STD_LOGIC_VECTOR (11 downto 0);
        CLK : in STD_ULOGIC;
        DI : in STD_LOGIC_VECTOR (0 downto 0);
        EN : in STD_ULOGIC;
        RST : in STD_ULOGIC;
        WE : in STD_ULOGIC);
end component;
```


John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

RAMS in Xilinx library

```
RAMB4_Sn_INSTANCE_NAME : RAMB4_Sn
-- synthesis translate_off
  generic map ( INIT_00 => 64bit_hex_value,
                INIT_01 => 64bit_hex_value,
                . . .
                INIT_0F => 64bit_hex_value)
-- synthesis translate_on
  port map (DO => user_DO ,
            ADDR => user_ADDR ,
            CLK => user_CLK ,
            DI => user_DI ,
            EN => user_EN ,
            RST => user_RST ,
            WE => user_WE );
```

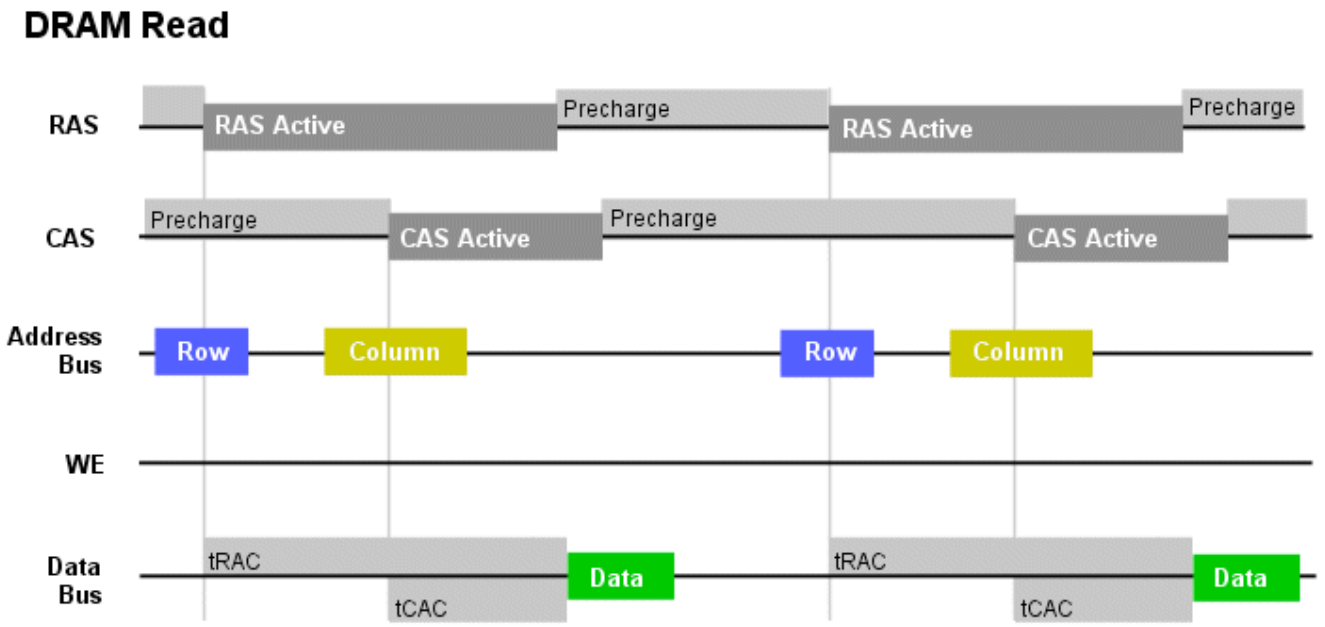

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

Memory Design

- DRAM read cycle
  - Activate RAS, and place row address on bus
  - Row decoders select appropriate row
  - Activate CAS, and place column address on bus
  - Sense amps are activated and data is placed on the data bus

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

Memory Design



from “Ars Technica RAM Guide”, by Jon Stokes, ©Ars Technica LLC

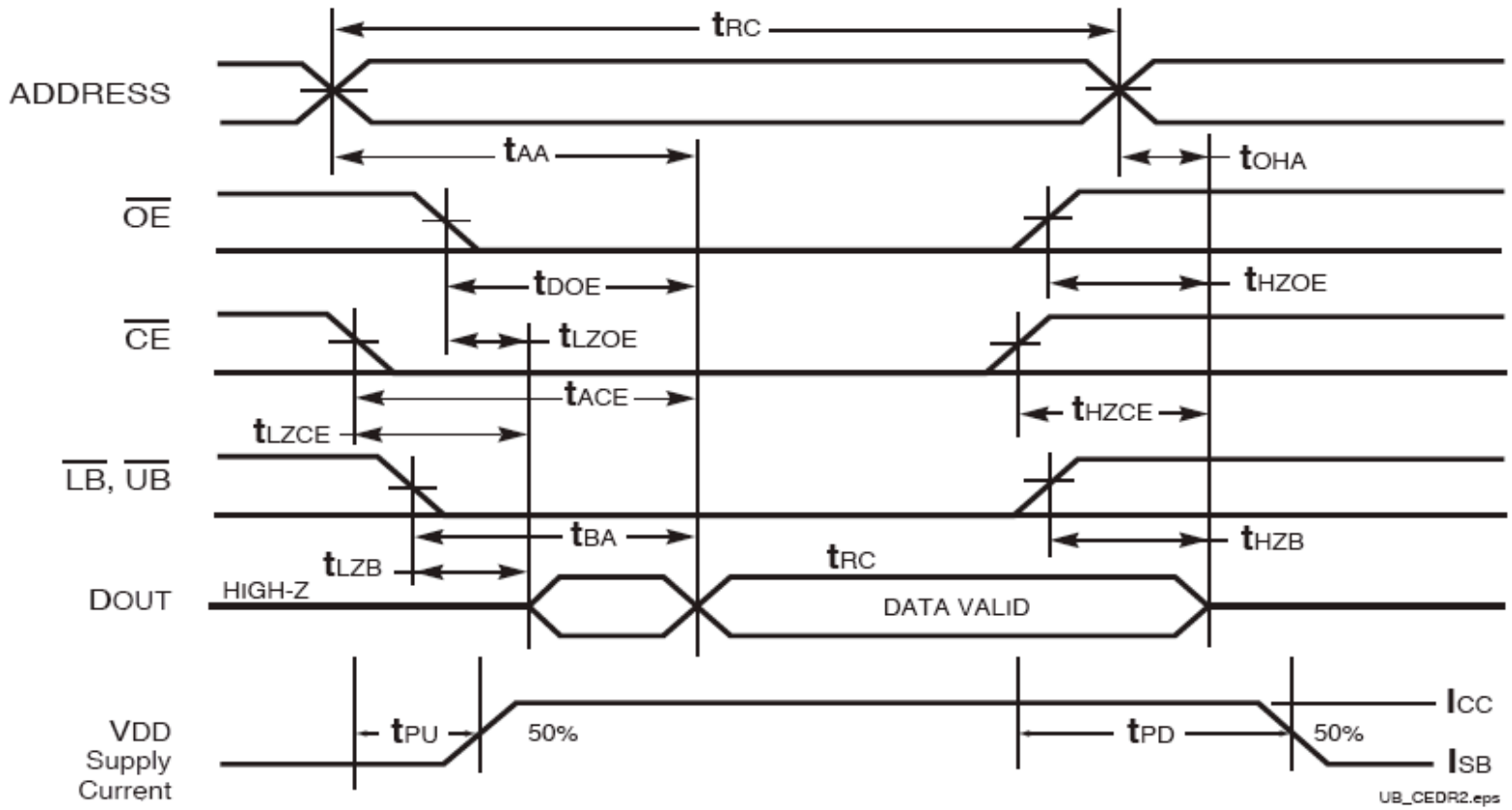
John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

Memory Design

- DRAM extensions
  - FPM
  - EDO
  - SDRAM
  - DDR
  - Rambus XDR

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

SRAM Memory Cycle

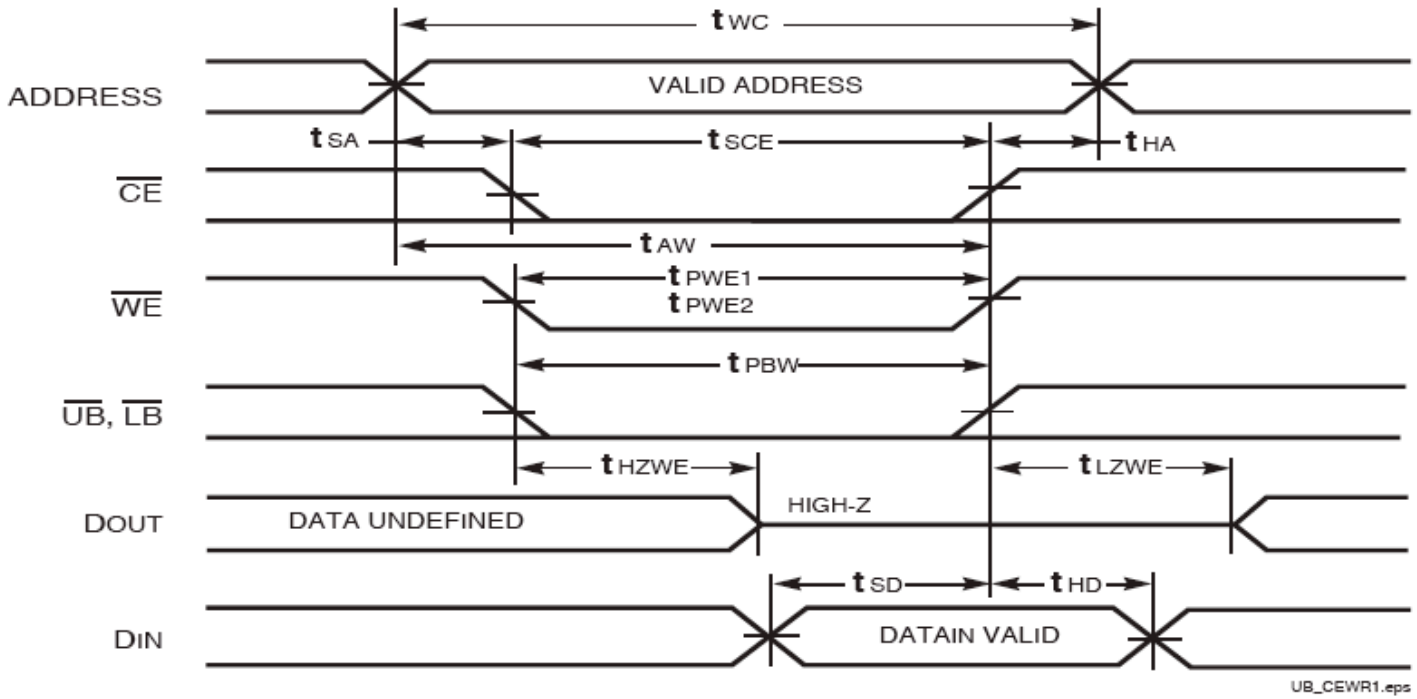


Notes:

1. $\overline{WE}$ is HIGH for a Read Cycle.
2. The device is continuously selected. $\overline{OE}$, $\overline{CE}$, $\overline{UB}$, or $\overline{LB}$ = V_{IL}.
3. Address is valid prior to or coincident with $\overline{CE}$ LOW transition.

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

SRAM Memory Cycle



Notes:

1. WRITE is an internally generated signal asserted during an overlap of the LOW states on the $\overline{CE}$ and $\overline{WE}$ inputs and at least one of the $\overline{LB}$ and $\overline{UB}$ inputs being in the LOW state.
2. WRITE = ($\overline{CE}$) [($\overline{LB}$) = ($\overline{UB}$)] ($\overline{WE}$).

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

Lab 4 Part 2

- 3 modules
  - SRAM interface
  - Reader
  - Writer

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

How to use constraints

Adapted from Xilinx FPGA Design Workshop Tutorial

- Apply timing constraints to a simple synchronous design
- Specify global timing constraints and pin assignments with the Constraints Editor

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

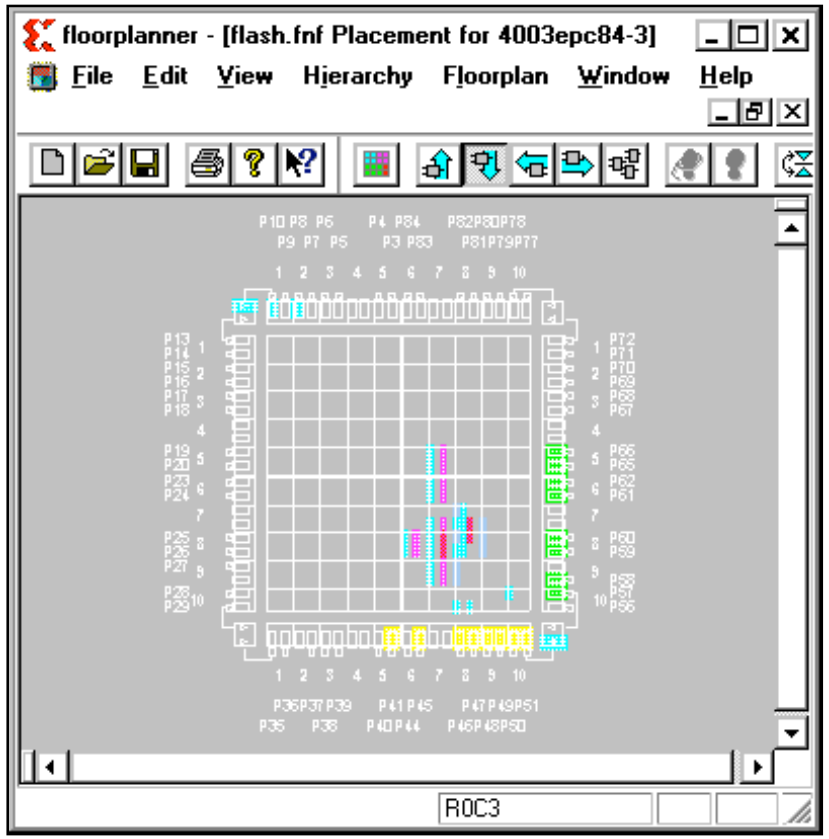
What Effects Do Timing Constraints Have on Your Project?

- The Implementation tools don't try to find the place and route that will obtain the best speed
- Instead, the Implementation tools try to meet your performance expectations
- Performance expectations are communicated with timing constraints
- Timing Constraints improve the design performance by placing logic closer together so shorter routing resources can be used
- Note that when we discuss using the Constraint Editor, we are referring to the Design Manager Constraints Editor

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

Without Timing Constraints

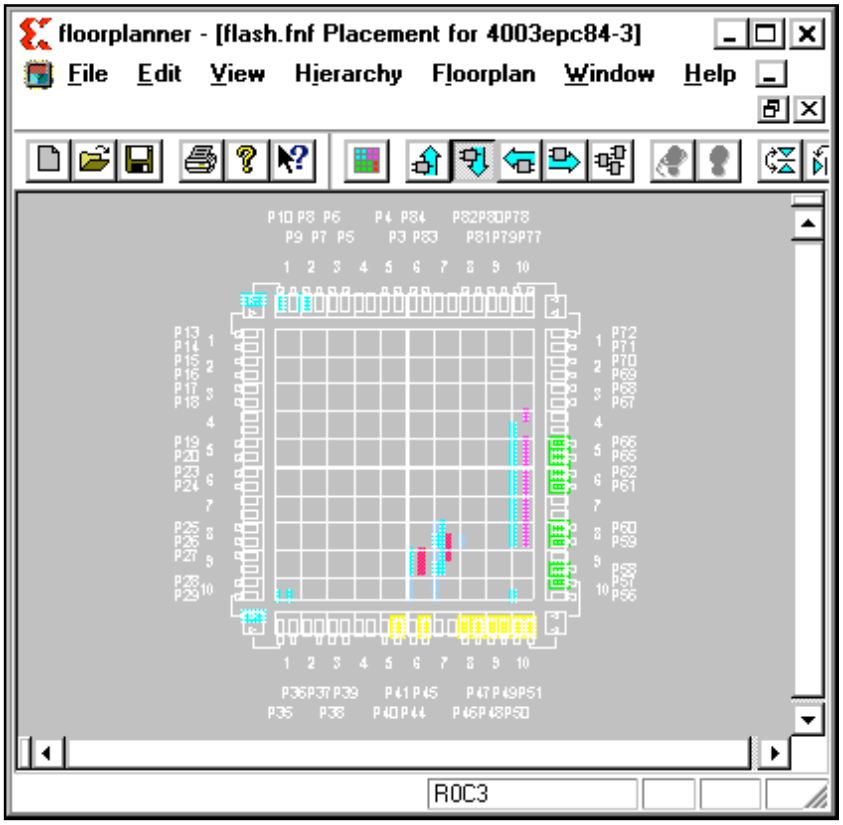
- This design had no timing constraints or pin assignments entered when it was implemented
- Note the logical structure of the placement and pins.
- Xilinx recommends that you compile your design at least once without timing constraints or pin assignments
- This design has a maximum system clock frequency of 50 MHz



John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

With Timing Constraints

- This is the same design with three global timing constraints entered with the Constraints Editor
- It has a maximum system clock frequency of 60 MHz
- Note how most of the logic is placed closer to the edge of the device where the pins have been placed



John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

More About Timing Constraints

- Timing constraints should be used to define your performance objectives
  - Specifying tight timing constraints will increase your compile time
  - Specifying unrealistic constraints will cause the Flow Engine to stop
  - Use the Logic Level Timing Report to determine if your constraints are realistic
    (refer to the Reading Reports module)
- After implementing your design, review the Post Layout Timing Report to determine if your design performance objectives were met
- If your constraints were not met, use the Timing Analyzer to determine the cause

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

Path End Points

- Timing constraints optimize delay paths between path endpoints. Path endpoints can be pads, flip-flops, latches, and RAMs

- Making timing constraints becomes easier when you realize that timing constraints create groups of path endpoints and communicate a timing specification between these groups

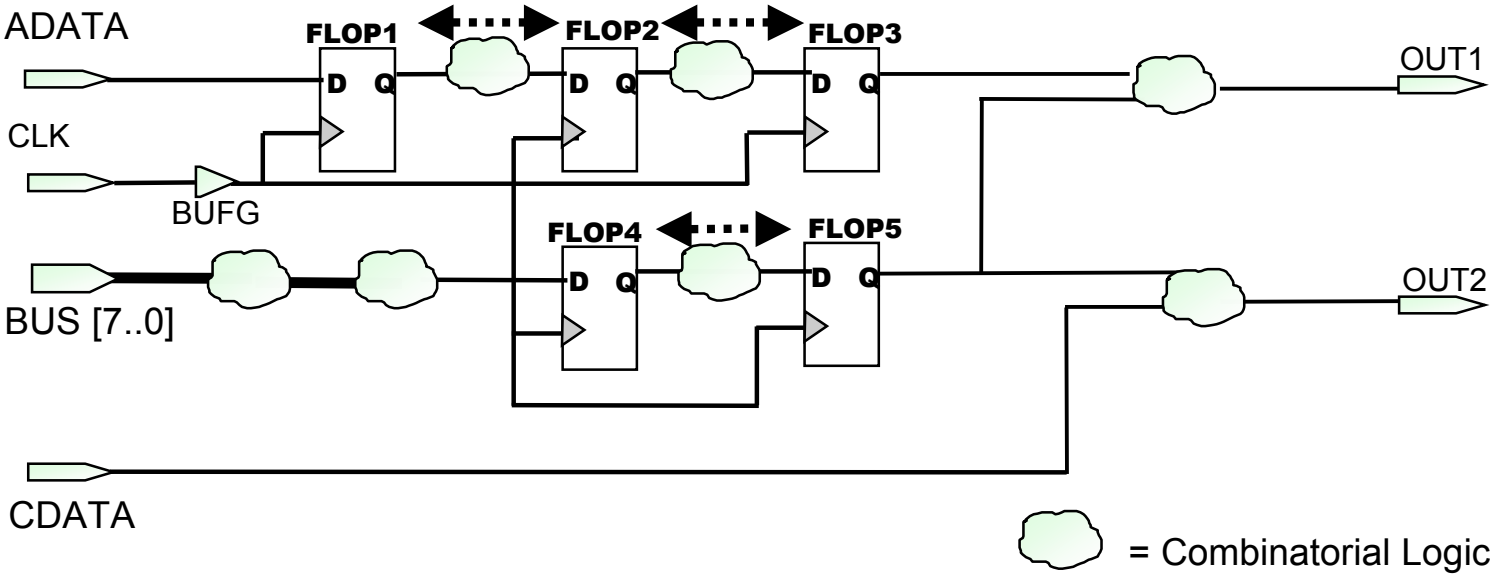
- Since delay paths may require signals to go through multiple function generators in series, optimizing between path endpoints requires the Implementation tools to place logic closer together

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

Period Constraint

- In this example the Period constraint optimizes all delay paths between flip-flops

- The Period constraint does NOT optimize delay paths from input pads to output pads (purely combinatorial), paths from input pads to flip-flops, or paths from flip-flops to output pads



John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

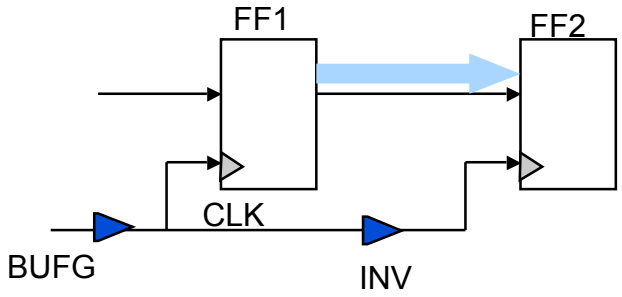
The Period Constraint

- A synchronous element is a flip-flop, latch, or a synchronous RAM
- The Period constraint covers paths…
  - Between synchronous elements which are clocked by the reference net
- Synchronous elements are grouped by the clock signal driving them. This is called *forward propagation* and enables constraining large pieces of logic with a single constraint

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

Some Features of the Period Constraint

- The PERIOD constraint automatically accounts for the extra delay caused by inverters and global clock buffers placed on clock signals
  - This provides the most accurate timing information
- The PERIOD constraint automatically accounts for unequal clock duty cycles
- Assume:
  - 50% duty signal on CLK
  - Period of 20ns
  - Since FF2 will be clocked on the falling edge of CLK, the delay between the two flip-flops will actually be constrained to 20ns - 10ns = 10ns



John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

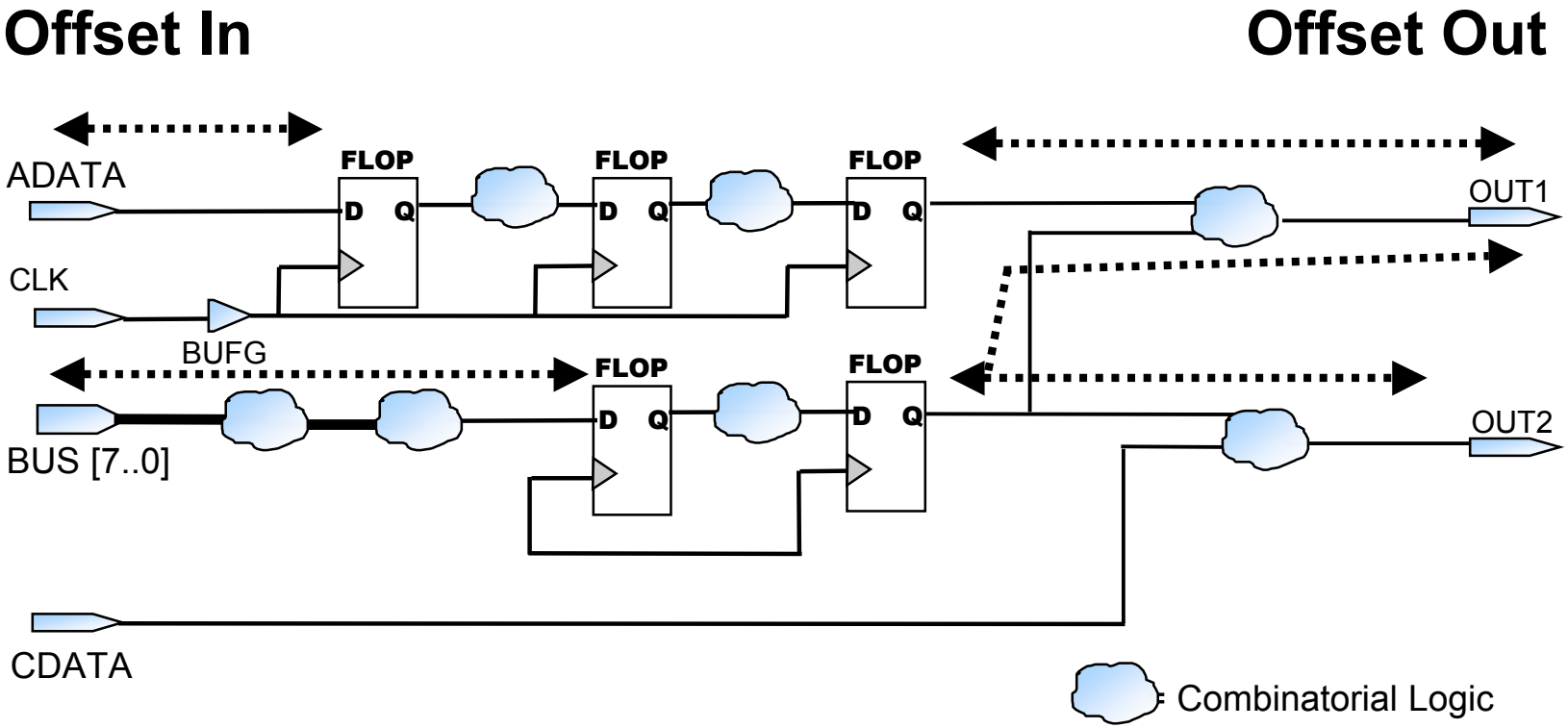
The Pad-to-Pad Constraint

- Purely combinatorial delay paths do not contain any synchronous elements
- Purely combinatorial delay paths start and end at I/O pads and are often left unconstrained by users
- Placing a Pad-to-Pad constraint is essential for completely constraining a design

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

Offset Constraint

- In this example, the Offset constraint optimizes delay paths from input pads to flip-flops and paths from flip-flops to output pads



John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

The Offset Constraint

- The Offset constraint covers paths…
  - From input pads to synchronous elements clocked by the reference net (Offset In)
  - From synchronous elements to output pads clocked by the reference net (Offset Out)

- Note, that this constraint does not cover paths…
  - Between synchronous elements
  - From pads to pads (purely combinatorial paths)

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

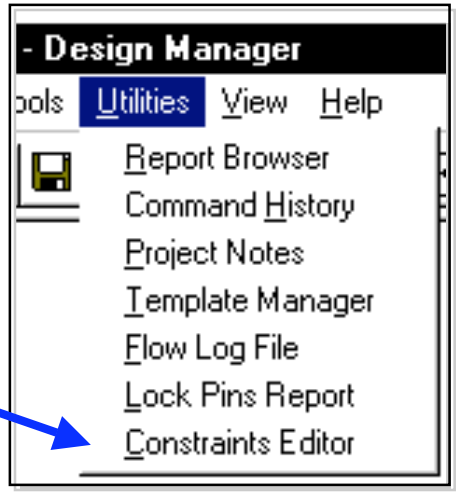
Some Features of the Offset Constraint

- The OFFSET constraint automatically accounts for the extra delay caused by inverters and global clock buffers placed on clock signals
  - This provides the most accurate timing information
  - This increases the amount of time for input signals to arrive at synchronous elements
  - This reduces the amount of time for output signals to arrive at output pins

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

Starting the Constraints Editor

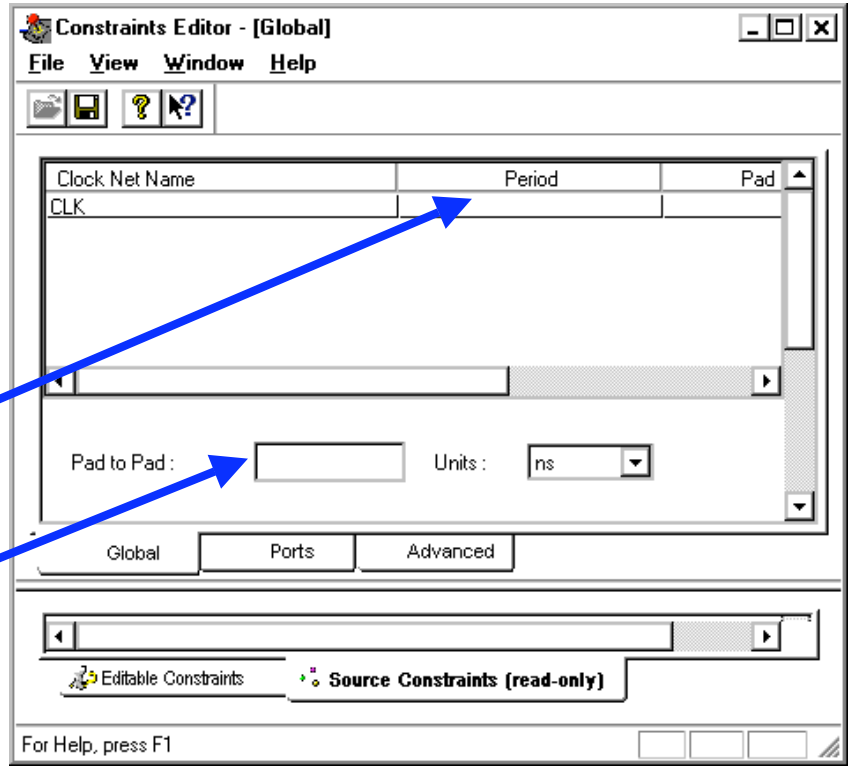
- If you are using the Alliance version, start the Constraints Editor by using the command...
  - Utilities -> Constraints Editor *from the Design Manager*



John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

Making Period and Pad-to-Pad Constraints with the Global Tab

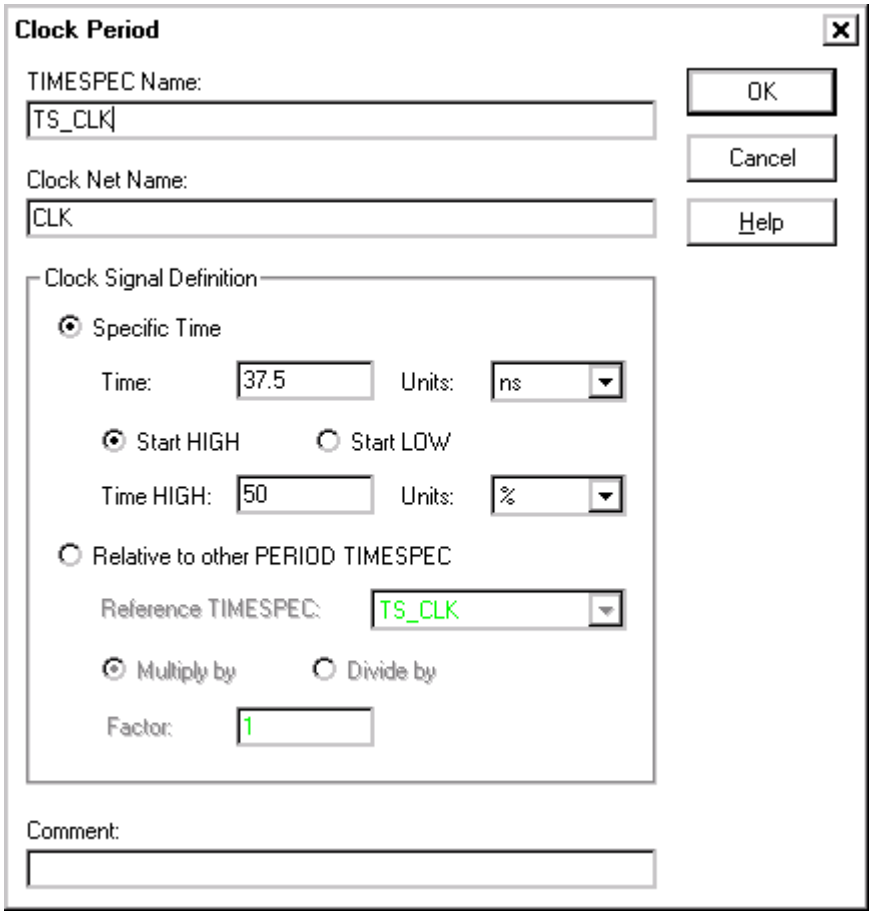
- Clock Periods can be made by clicking on the Global tab and specifying a period length for each clock signal
- Double-click here to make a period constraint
- A global Pad-to-Pad constraint can be entered here



John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

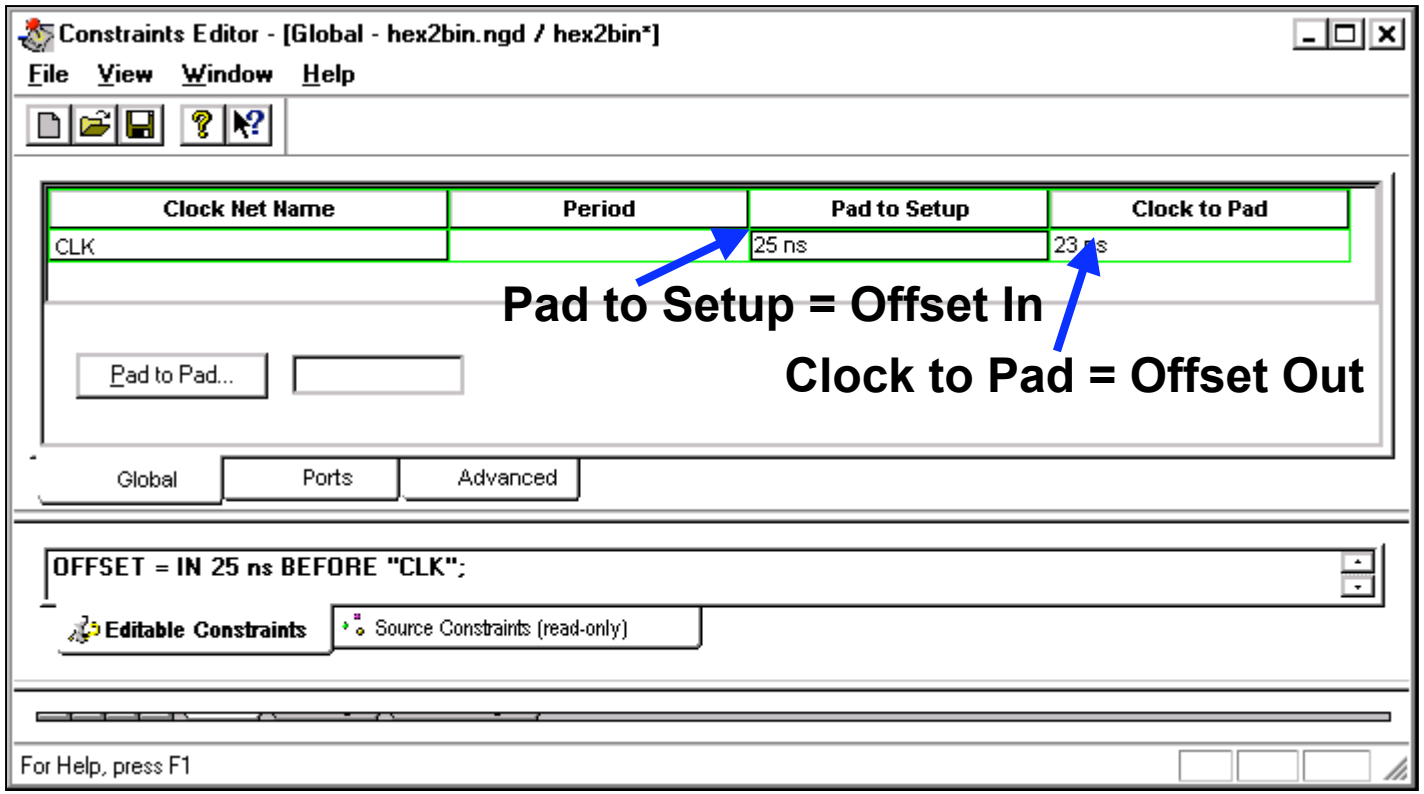
Period Constraint Options

- After double-clicking under the Period heading, the Clock Period dialog box opens
- This allows customizing the constraint to the duty-cycle and rising or falling clock edge
- It is also possible to place timing constraints relative to other timing constraints. This is useful for designs with multiple clock signals and multi-cycle paths



John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

Making Offset Constraints with the Constraints Editor



- Global Offset IN/OUT constraints can be made by clicking on the Global tab

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut