EMB3
Advanced Programmable Electronics

Lecture 2

Thursday, February

Jørgen Christian Larsen

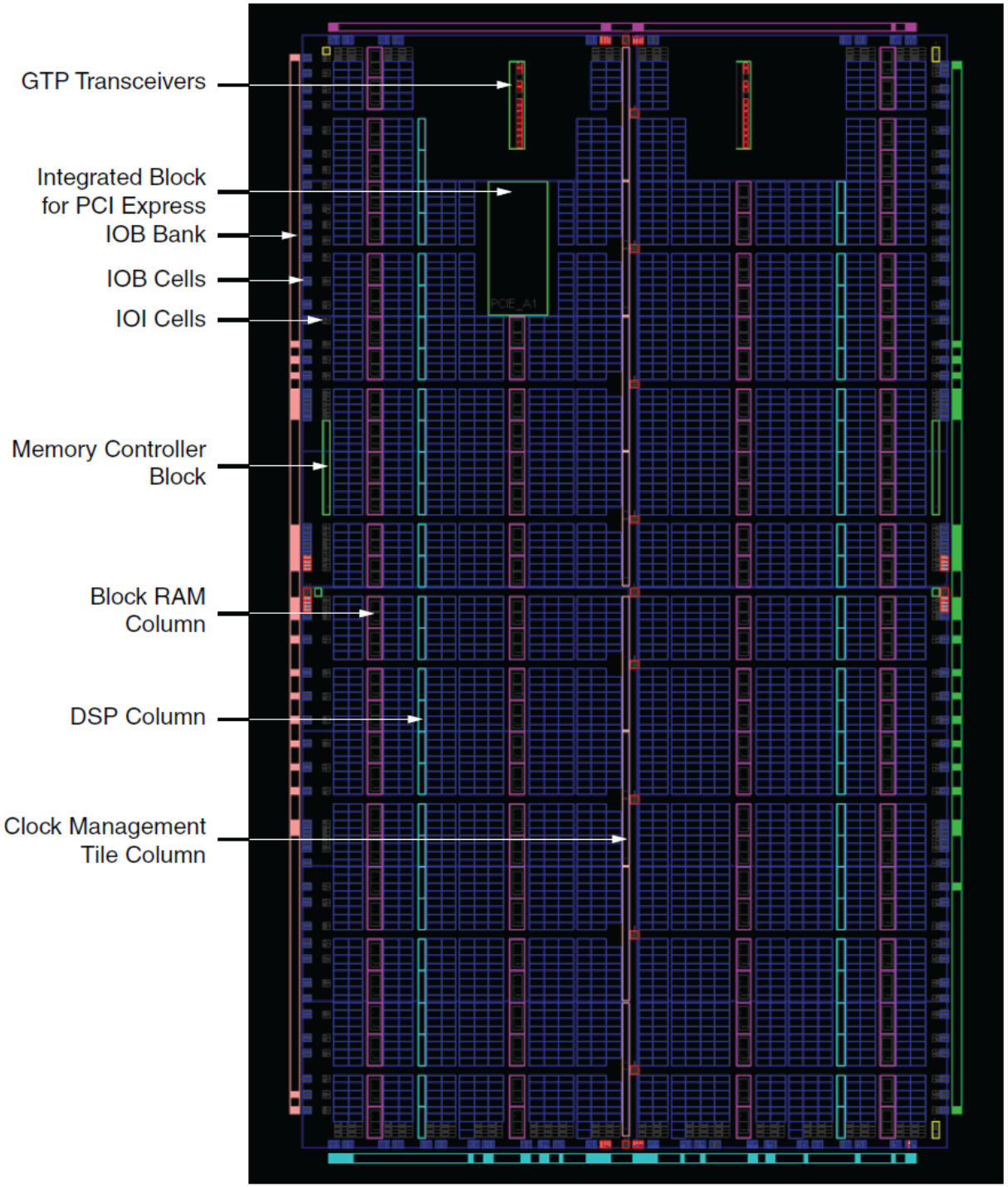
jcla@mmmi.sdu.dk

Lecture Overview

- FPGA internal layout
- Overall design guidelines
- Best practices
- Feedback on assignment
- Exercise

FPGA internal layout

- Spartan-6 family architecture
  - 8 (9) fundamental programmable elements:
    - Configurable Logic Block (CLB)
    - Block RAM (BRAM)
    - Digital Signal Processing slice (DSP slice)
    - Input / Output Tile (IOT)
      - IOB + IOI
    - Clock Management Tile (CMT)
      - DCM + PLL
    - Memory Controller Block (MCB)
    - (LXT only) High Speed GTP serial tranceiver (GTP)
    - (LXT only) Integrated PCI Express Endpoint block (PCIE)

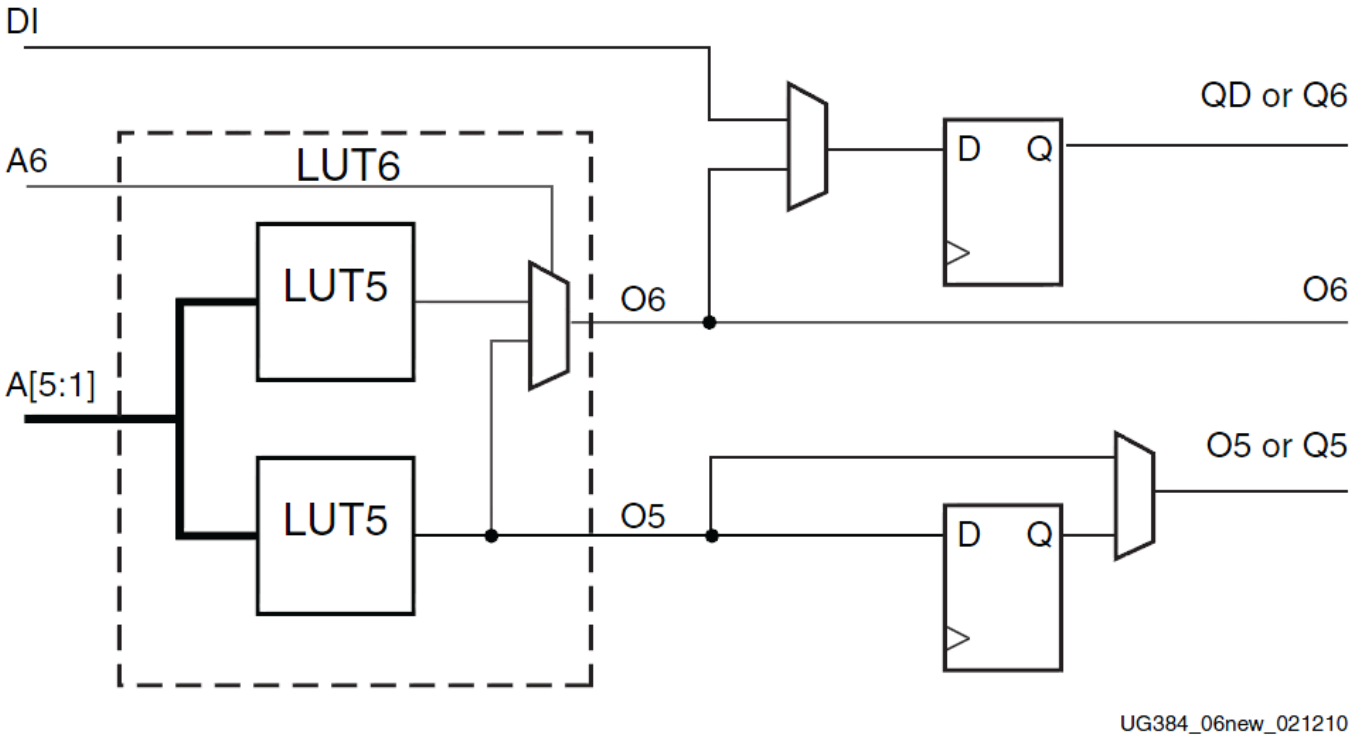

GTP Transceivers
Integrated Block for PCI Express
IOB Bank
IOB Cells
IOI Cells
Memory Controller Block
Block RAM Column
DSP Column
Clock Management Tile Column
PCIE_A1

FPGA internal layout

- Spartan-6 family architecture (continued)
  - Distribution Networks
    - Signal Routing
    - Clock Routing

FPGA internal layout

- Generalized Spartan 6 CLB



FPGA internal layout

Spartan 6 CLB slice's

- Each CLB contains 2 slices
- Each slice contains:
  - 4 6-input LUTS
    - Distributed RAM
    - Shift Registers
  - 8 Flip-Flops
  - Wide Multiplexers
  - Fast Carry Logic (only for slice 0 on Spartan-6)

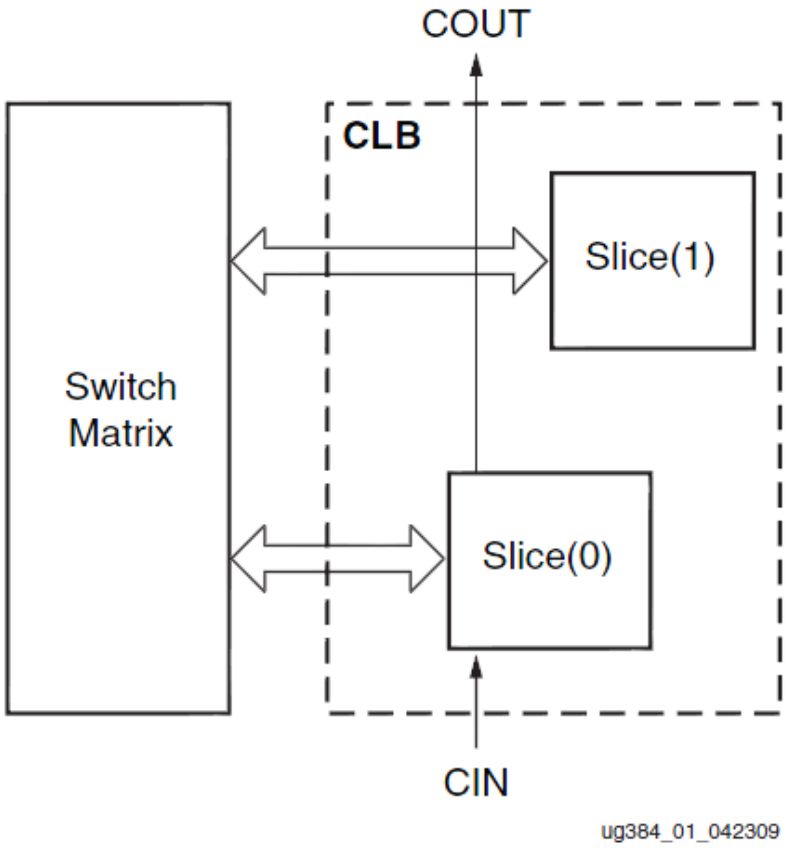


Figure 1: **Arrangement of Slices within the CLB**

FPGA internal layout

- Spartan-6 CLB Storage elements
  - 4 Registers
  - 4 Register/Latch's
  - Reset Type
  - Set, Reset, Init value
  - Clock, CE, Set/Reset

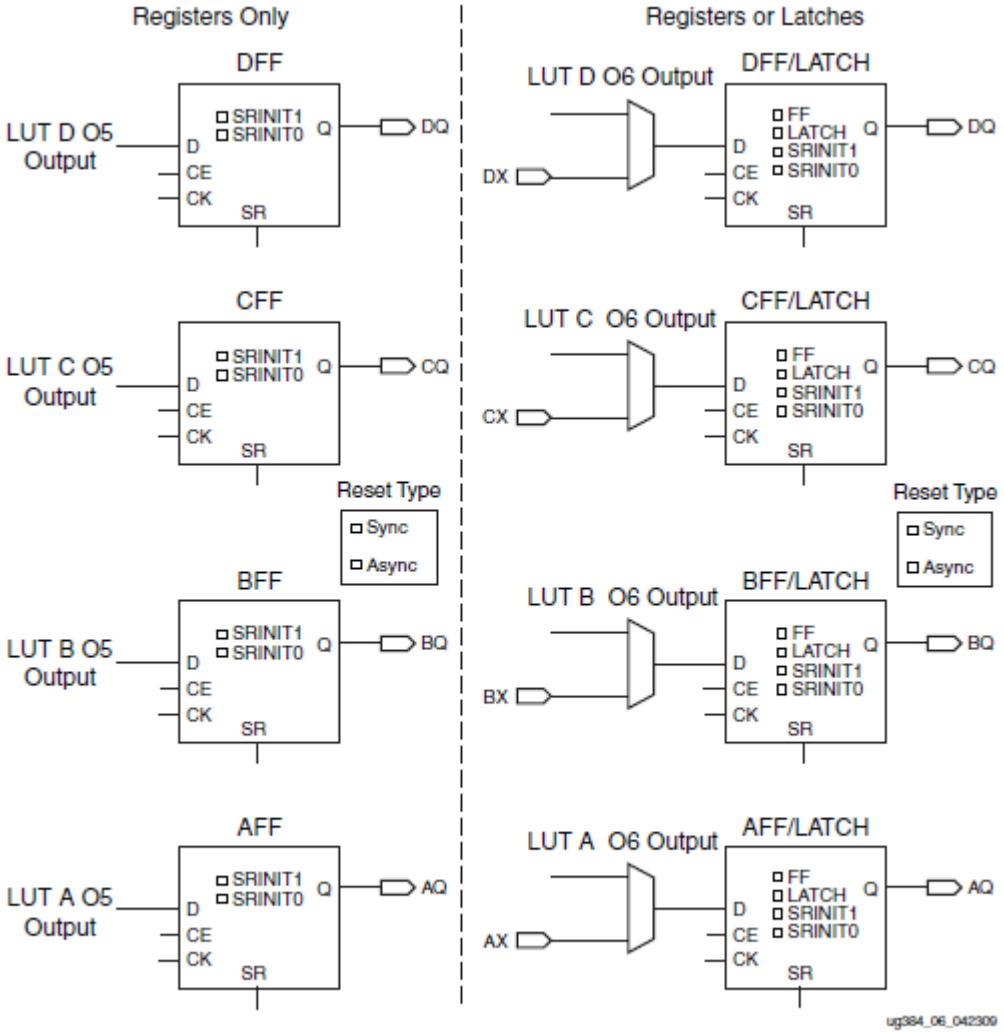


Figure 7: **Configuration in a Slice: 4 Registers Only and 4 Register/Latch**

FPGA internal layout

- Spartan-6 IO Tile
  - 1 pair/ 2 IO-Pads, IOB's, IOI's

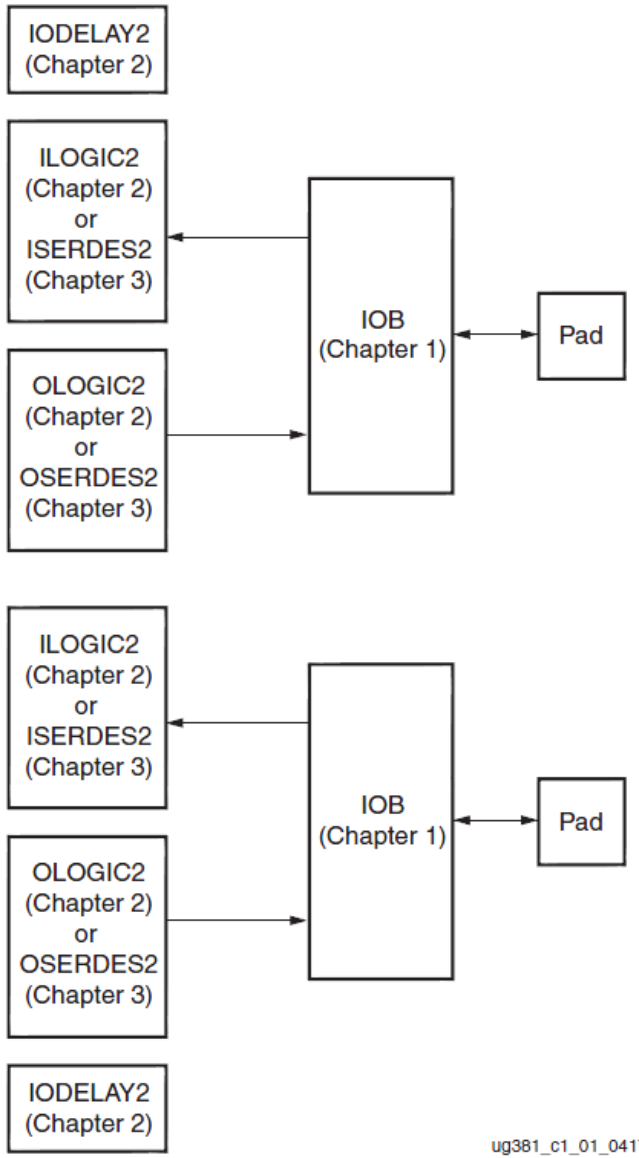


Figure 1-1: **Spartan-6 FPGA I/O Tile**

FPGA internal layout

- IOB
  - IO-Mode: Input, Output, Tri-state
  - Drive Strength
  - Slew Rate
  - Termination
  - IO-Standard

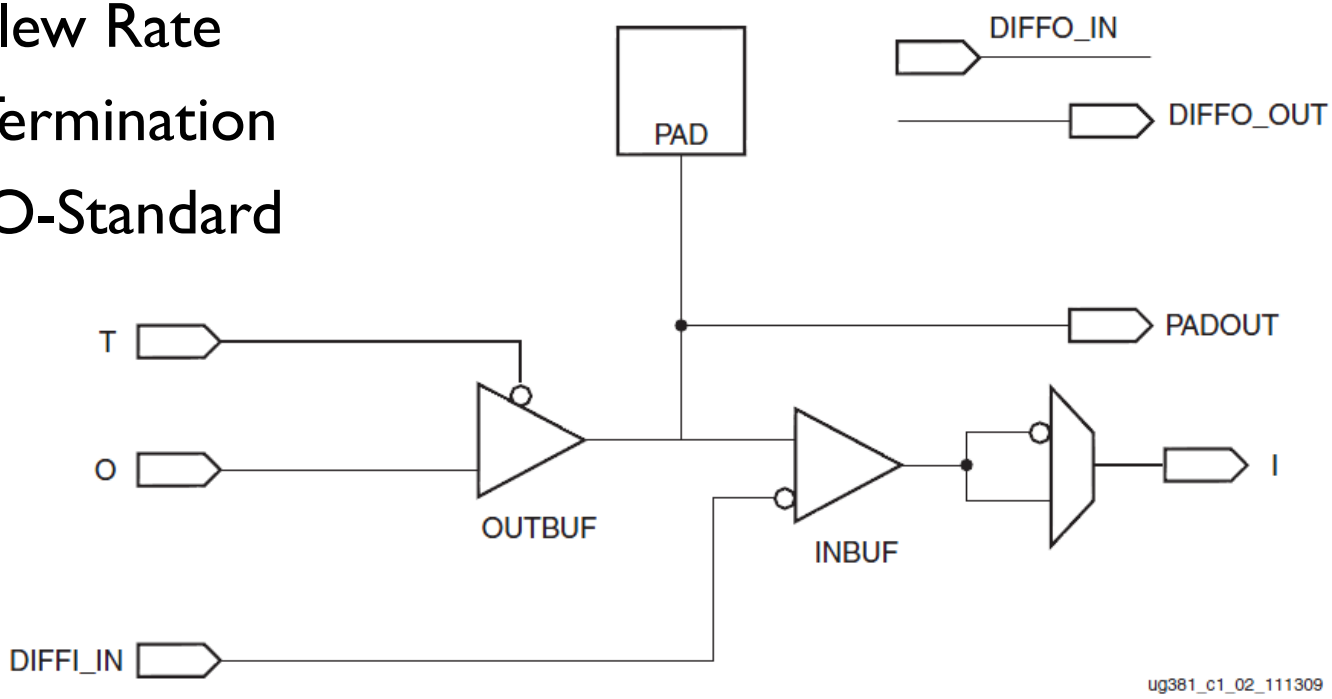


Figure 1-2: **Basic IOB Diagram**

FPGA internal layout

- IO Banks
  - IO pads, IOB's and IOI's are grouped into Banks
  - Each Bank has a common set of Vcco Pins
    - Each IO pin in a given IO-Bank MUST share the same IO voltage Vcoo
    - Allows for easy glueless interfacing to several voltage standards

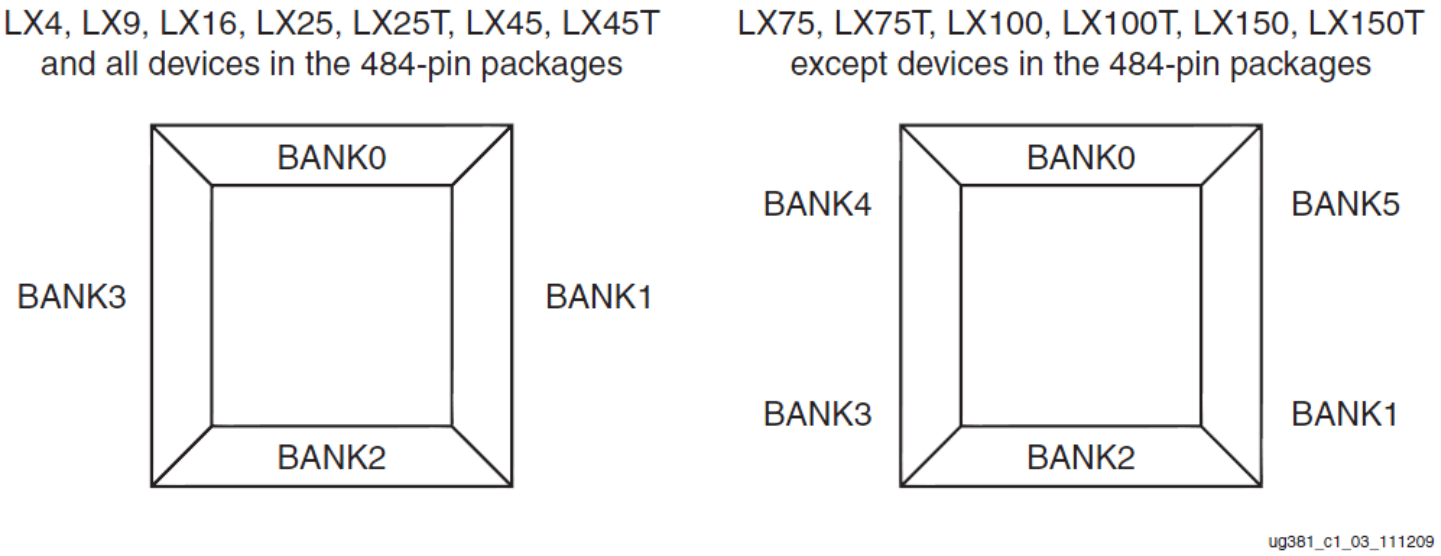


Figure 1-3: **Spartan-6 FPGA I/O Banks**

FPGA internal layout

- IO Interface (IOI) tile
  - Can operate in
    - Parallel
    - Differential mode
  - Adj. IO delay
  - DDR I/O
  - IO Serializer/ De-Serializer
    - Up to 1080Mb/s
    - Single-ended 1:1, 1:2, 1:3, 1:4
    - Differential 1:5, 1:6, 1:7, 1:8

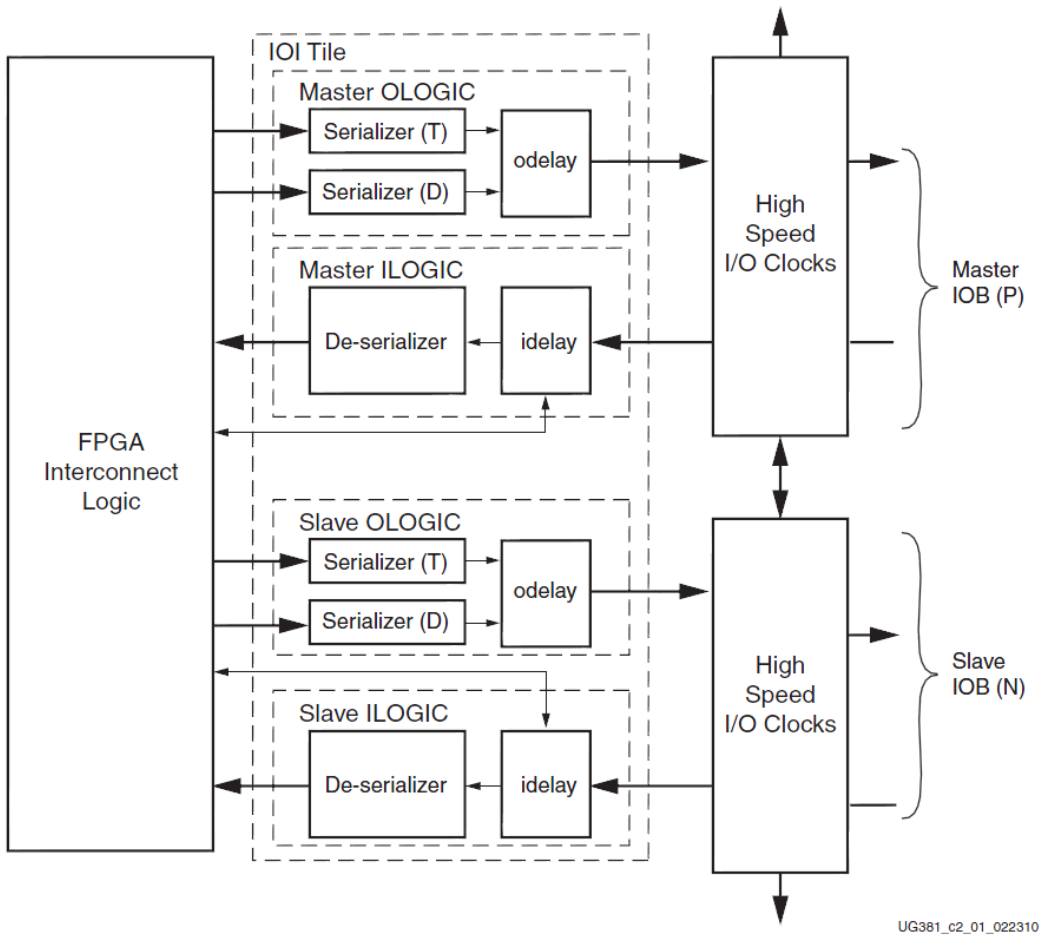


Figure 2-1: **SelectIO Logic Resources within the I/O Input Tile**

FPGA internal layout

- Global Clock Distribution Network

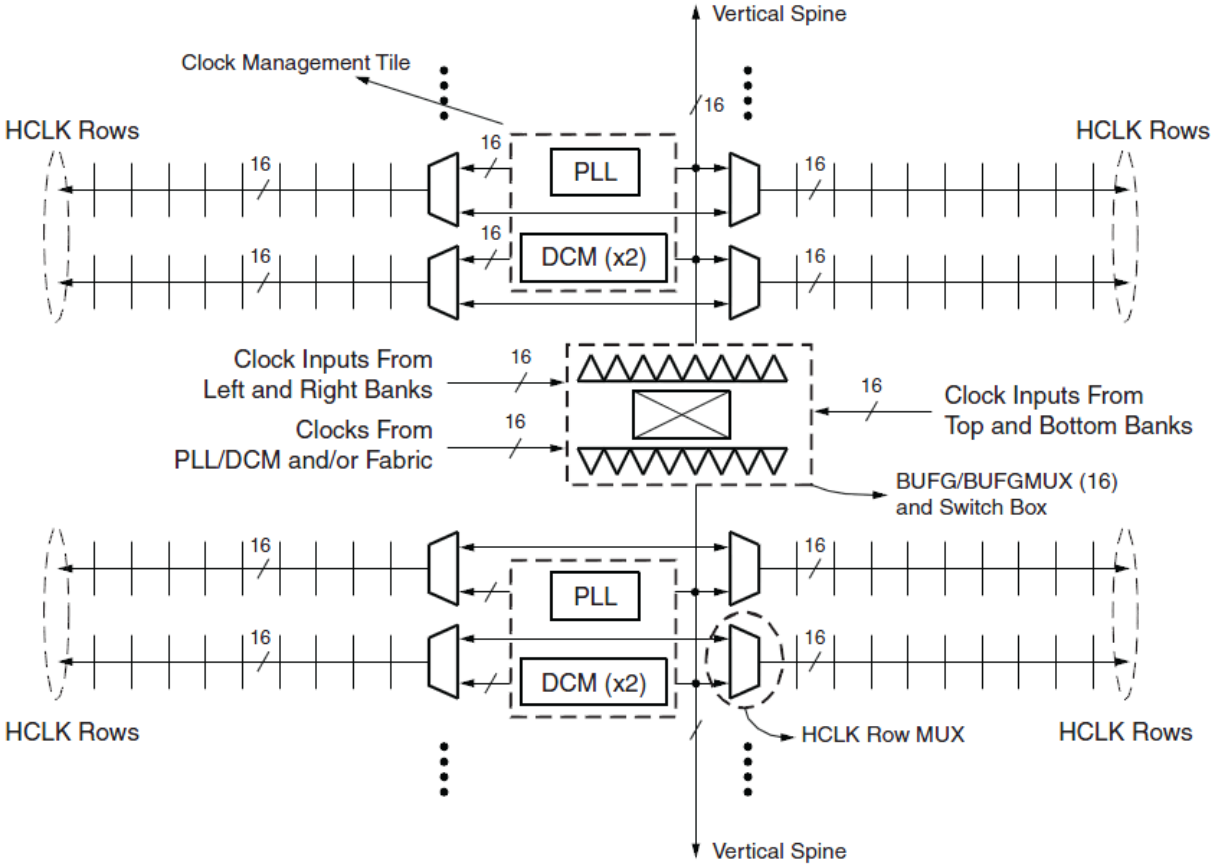


Figure 1-1: **Spartan-6 FPGA Global Clock Structure**

FPGA internal layout

- Global Clock Distribution Network

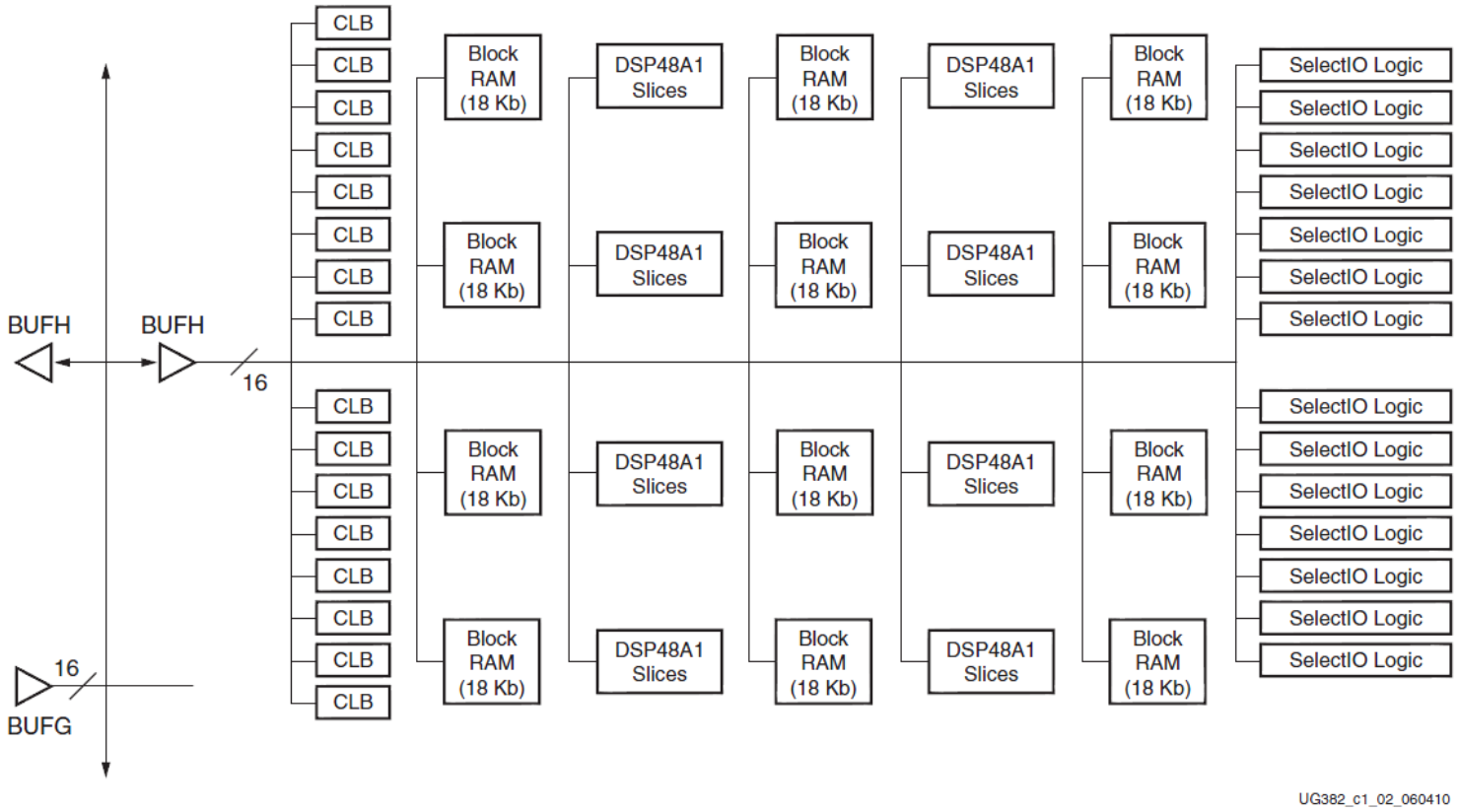


Figure 1-2: **BUFH Routing**

FPGA internal layout

- Interconnect Distribution Network

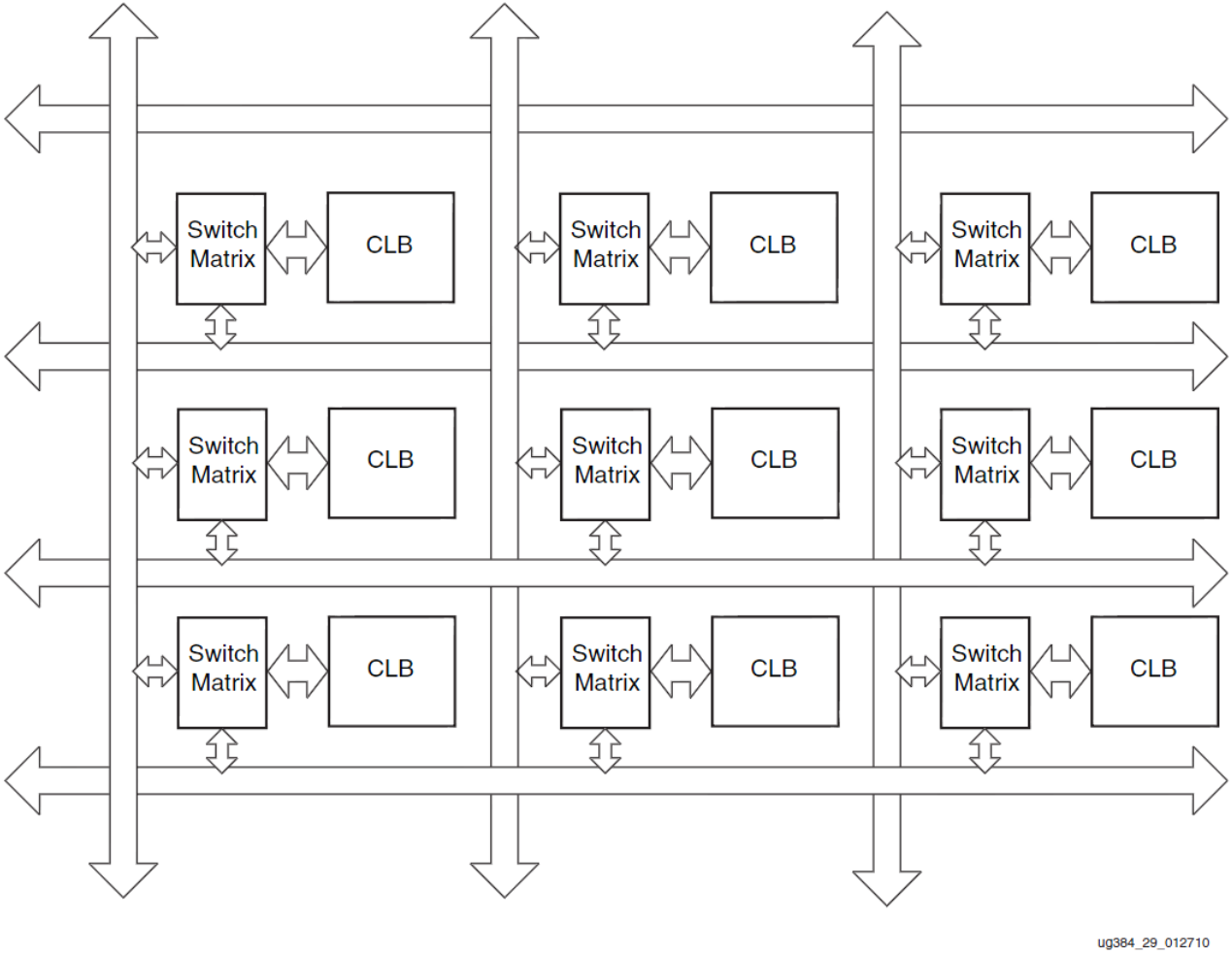


Figure 29: **CLB Array and Interconnect Channels**

FPGA internal layout

- Interconnect Distribution Network
  - Fast
  - Single
  - Double
  - Quad

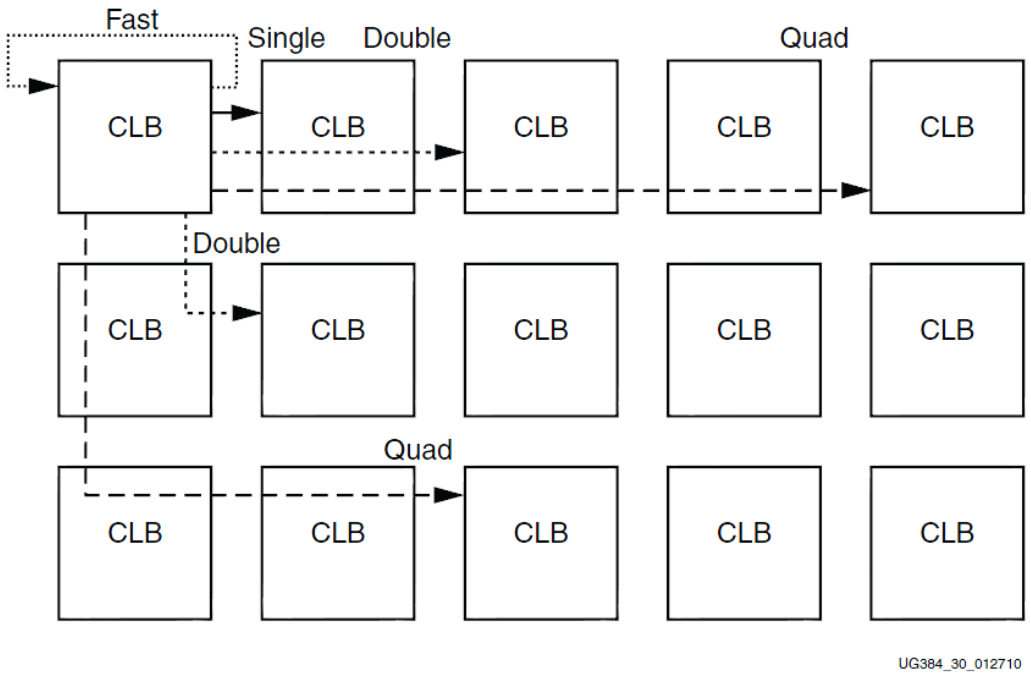
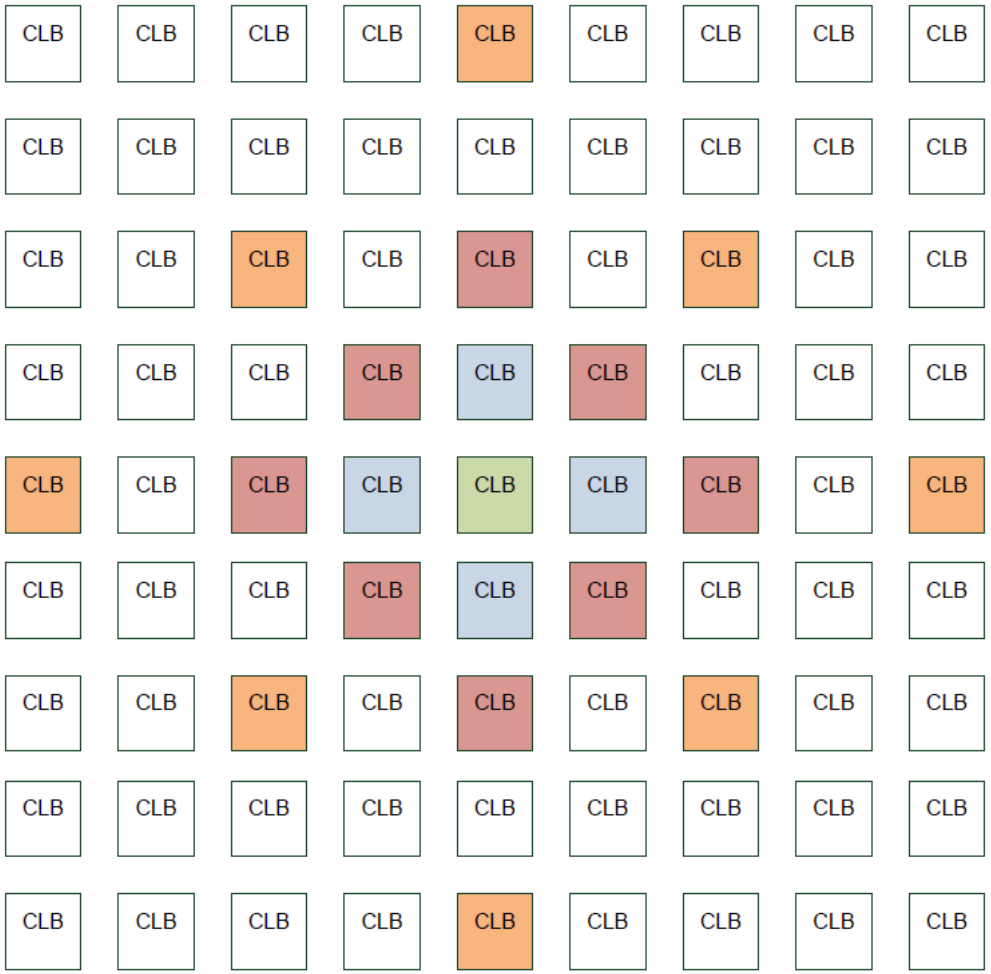


Figure 30: **Examples of Interconnect Types**

FPGA internal layout

- Interconnect Distribution Network
  - Fast
  - Single
  - Double
  - Quad



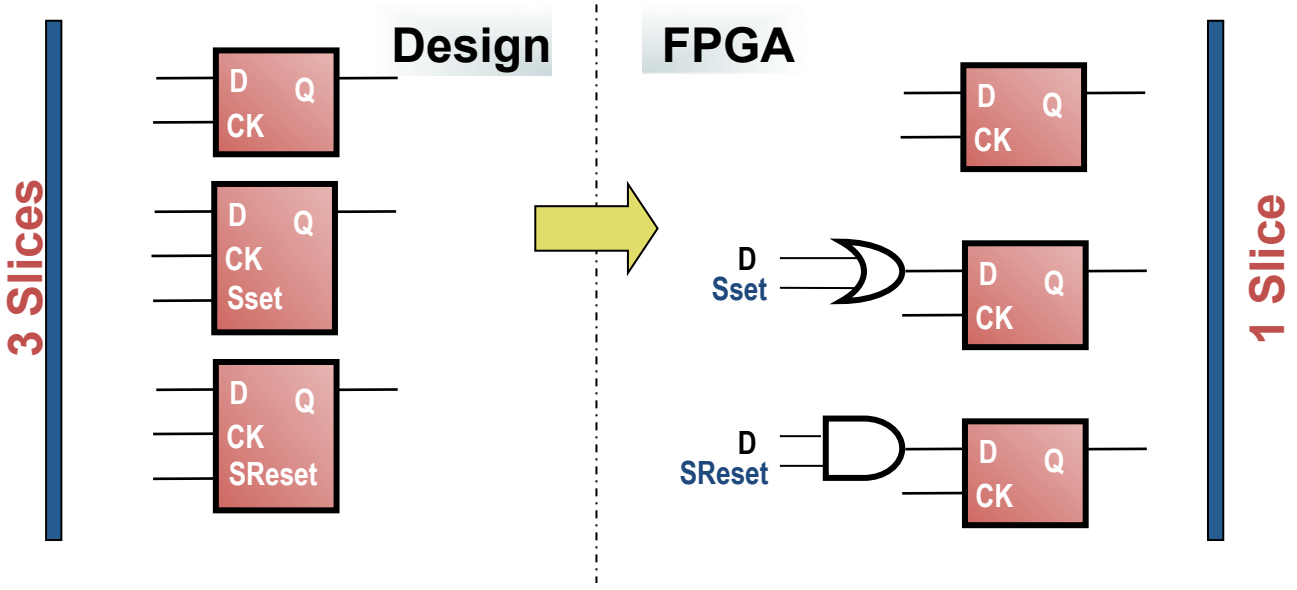
DESIGN GUIDELINES

Overall design guidelines

- General principles and examples
- Throughput vs. Latency
- Timing

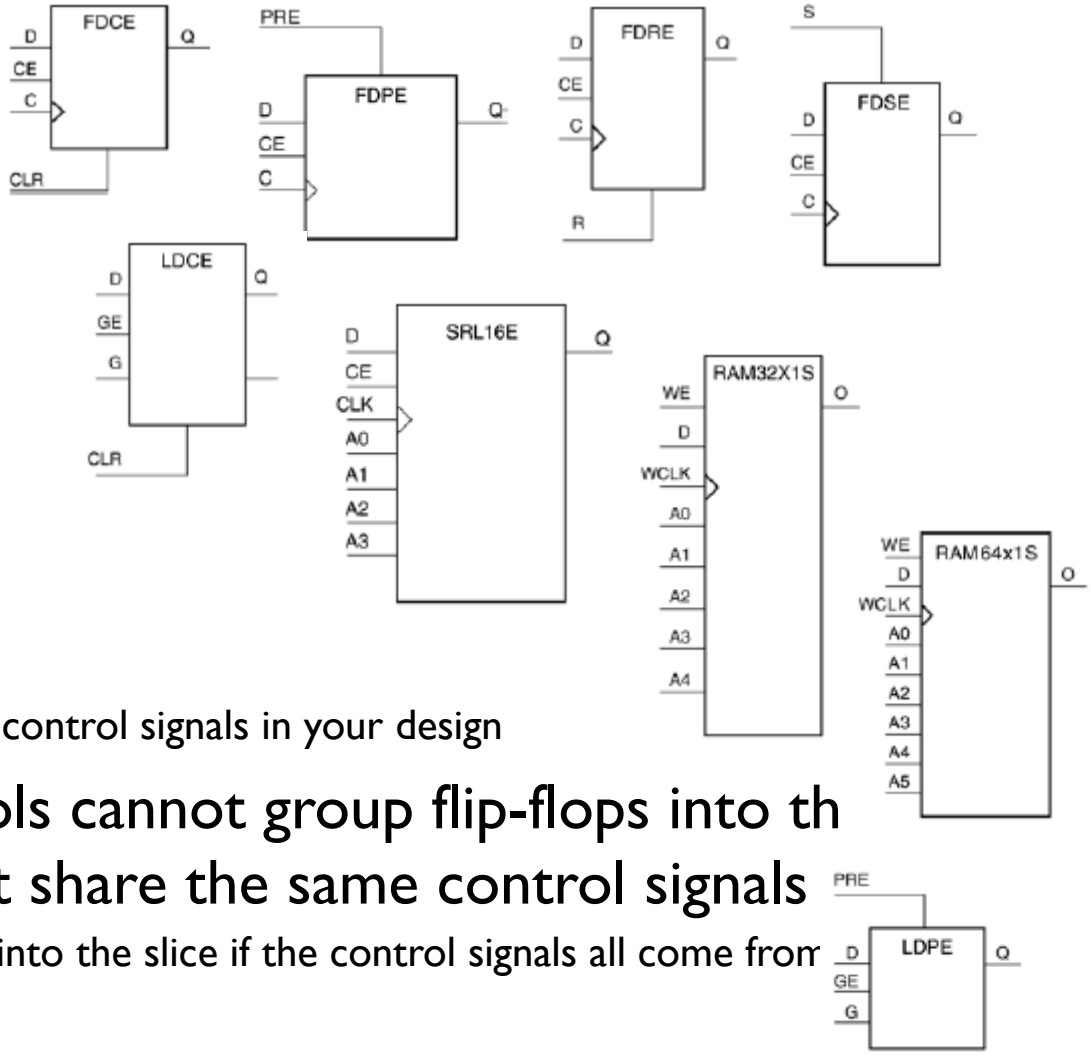
Control Set Reduction

- Flip-flops with different control sets cannot be packed into the same slice
- Software can be instructed to reduce the number of control sets by mapping control logic to LUT resources
  - This results in higher LUT utilization, but a lower overall slice utilization



Introduction to Control Sets

- A control signal is
  - Clock Enable / Gate Enable
  - Write Enable
  - Set / Reset
  - Preset / Clear
  - Clock / Gate
- A control set is
  - A group of enable, set, reset, and clock
- Unique control sets are
  - The number of groups of unique control signals in your design
- The implementation tools cannot group flip-flops into th same slice if they do not share the same control signals
  - However the tools can pack FFs into the slice if the control signals all come from set (next slide)



Question

- Can these FFs be placed into the same slice (are they a part of the same control set)? (Note…all control signals drive the control port of the FF)
  - Case 1
    - FF1: CE, Set, Reset
    - FF2: Set, Reset
    - FF3: Reset
  - Case 2
    - FF1: CE, Set, Reset
    - FF2: Set2, Reset
  - Case 3
    - FF1: CE, Set, Reset
    - FF2: Set, not Reset

Answer

- Case 1…
  - FF1: CE, Set, Reset
  - FF2: Set, Reset
  - FF3: Reset
  - Yes! The tools can pack FFs into the slice if the control signals all come from the same set.
- Case 2…
  - FF1: CE, Set, Reset
  - FF2: Set2, Reset
  - No, two different sets cannot be grouped into the same slice. Same is true for CEs and the other control signals.
- Case 3…
  - FF1: CE, Set, Reset
  - FF2: Set, not Reset
  - Maybe, if the Reset is synchronous then your synthesis tool should be able to drive the reset to a LUT input and invert the reset signal.

- Note…if the control signal can be implemented as a LUT input (synchronous set, synchronous reset, or CE) then the tools have more flexibility to group those FFs into the same slice.

Active-Low Control Signals

- Problem: Active-low control signals can produce sub-optimal results
- Why?
  - Control ports on registers are active-high
  - Hierarchical design and design re-use can propagate bad design practices
- This results in…
  - Poor device utilization
    - More LUTs
    - Less dense slice packing
    - More routing resources necessary
    - This requires additional inverters at all lower leaf levels
  - Longer run times
    - Prohibits hierarchical design flows (like incremental design)
    - More difficult timing
  - Worse timing and power

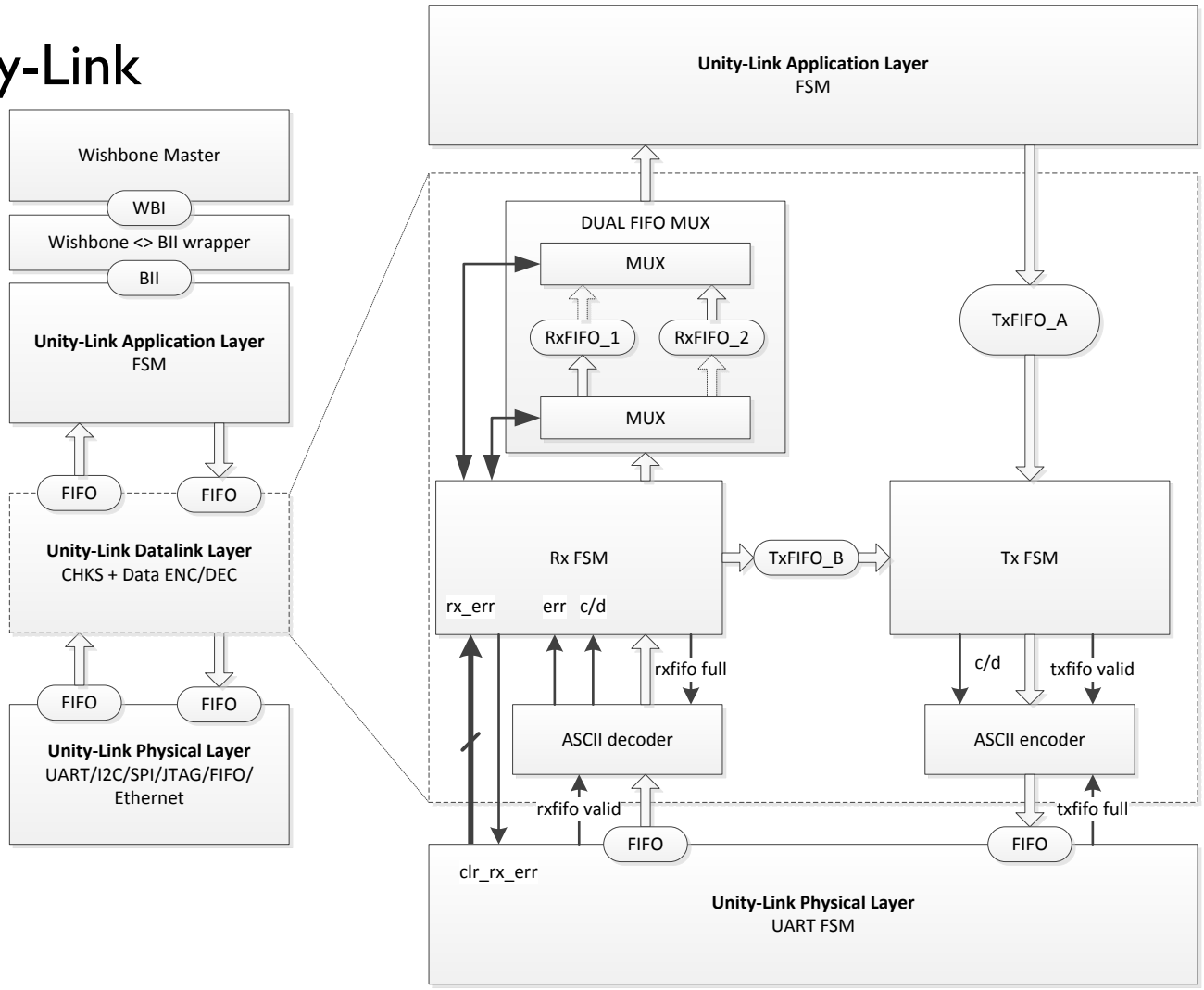
Tip: Use active-high signals for CEs, sets, and resets

Overall design guidelines

- Think hardware
  - Create design block diagrams before coding
  - For all but really small designs implement the logic in small separate functional modules whenever it is possible
    - Easier to understand, implement and test
  - Build the complete design from the small, tested modules
  - I.e. structure your design using hierarchical design techniques
    - Simply instantiate your tested modules, configure and connect them.
    - Basically like doing object oriented SW development.
  - Use the RTL and Technology schematics to verify the design structure and inferred primitives

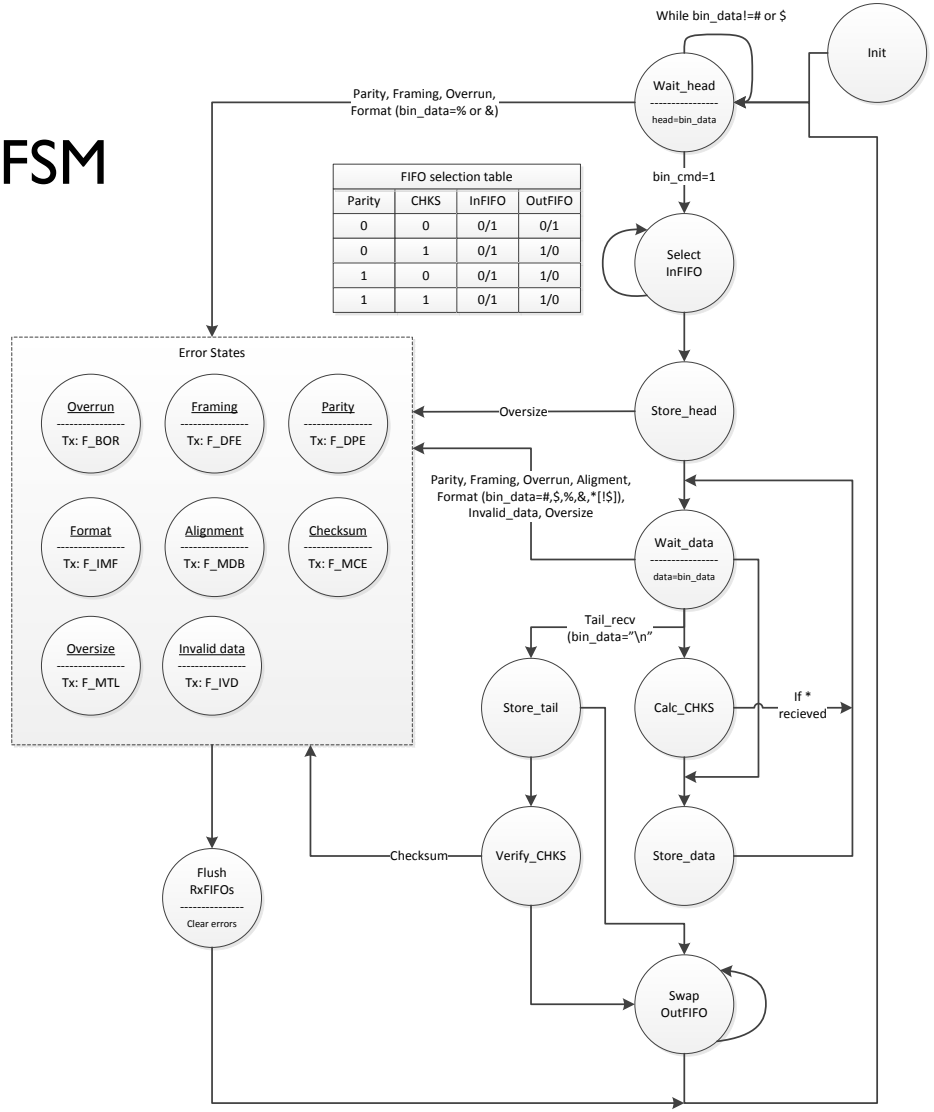
Design Block Diagram Example

- Unity-Link



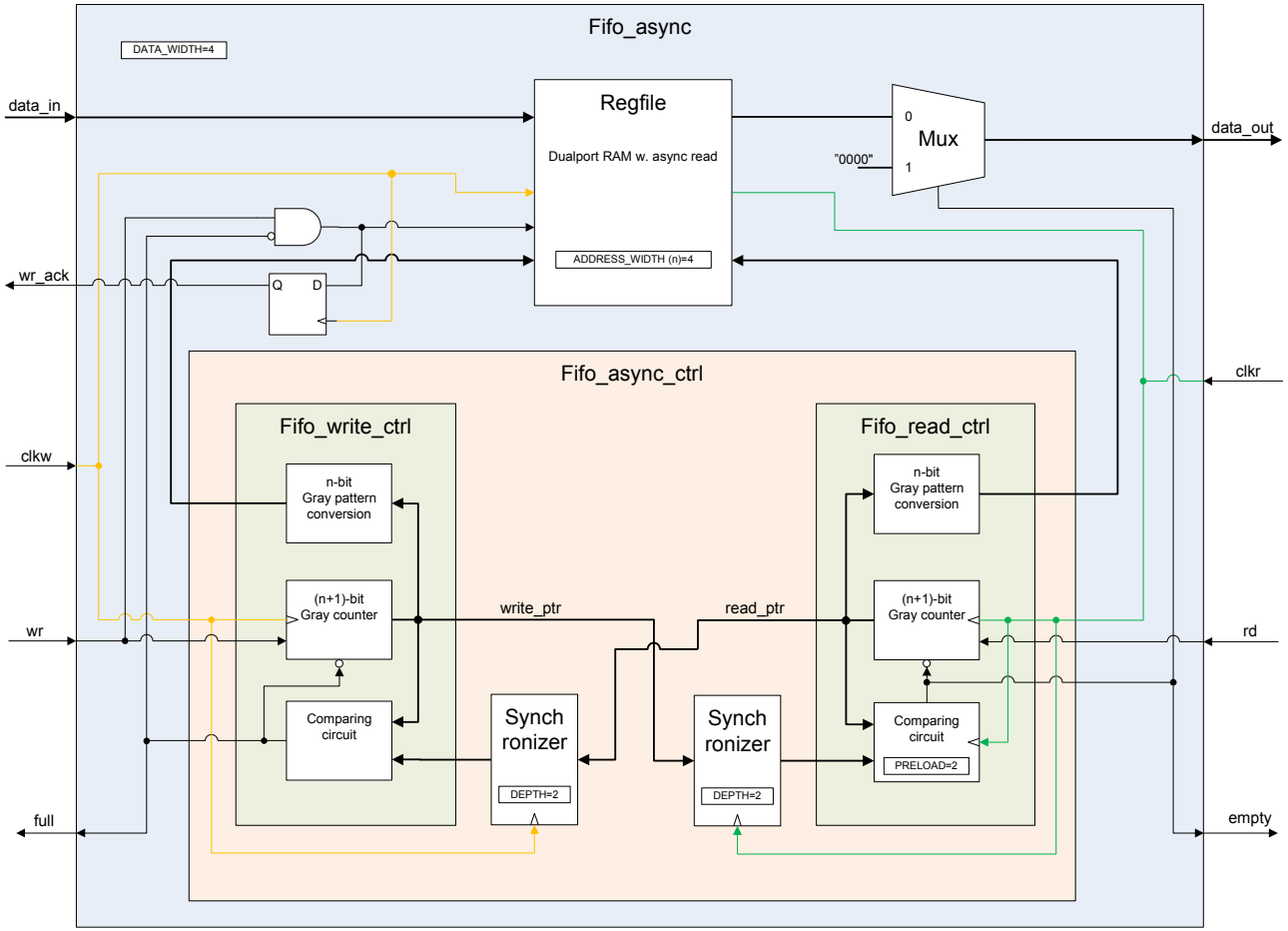
Design Block Diagram Example

- Unity-Link
  Datalink Layer RxFSM



Design Block Diagram example

- Asynchronous FIFO

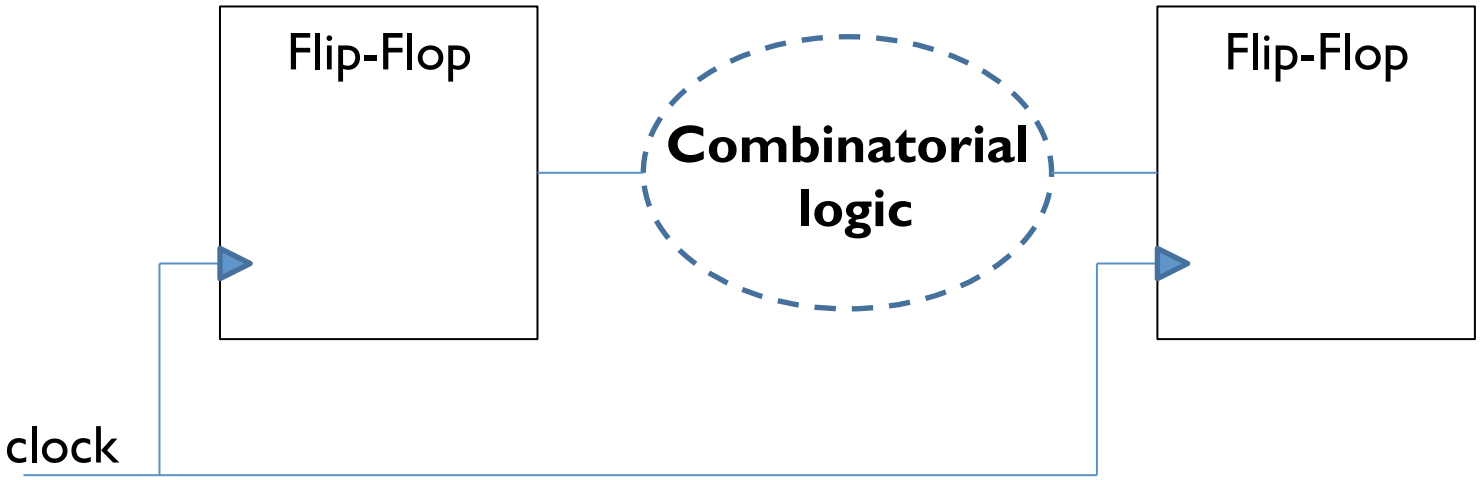


Overall design guidelines

- Throughput vs. Latency
  - Throughput: *Number of bits in-/out-put per time-unit*
  - Latency: *Number of clocks before the output read reflects a given input.*

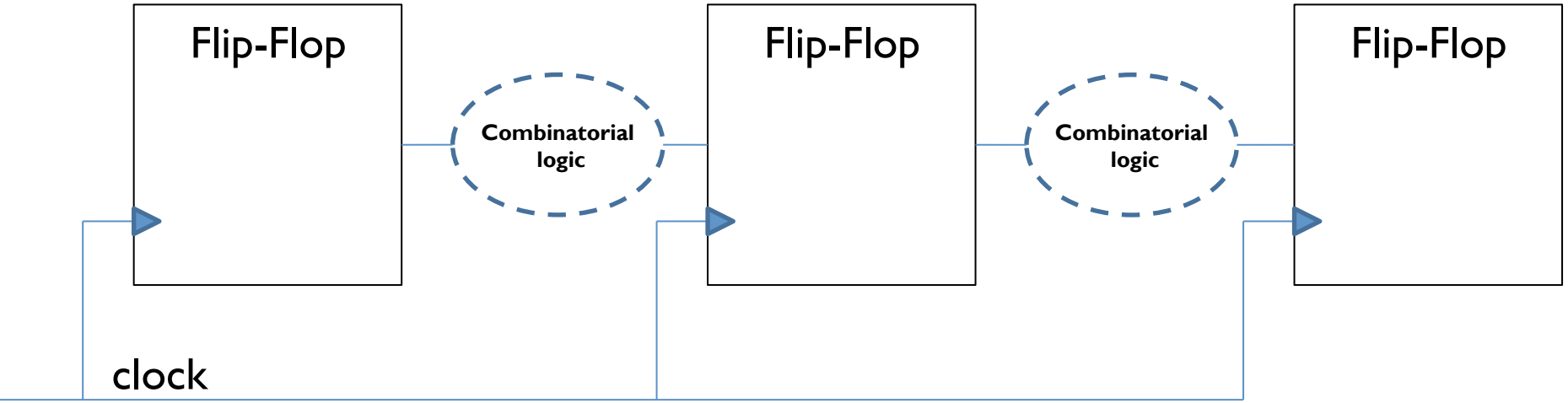
Overall design guidelines

- Timing
  - Maximum frequency
  - Critical path
  - Automatic calculation



Overall design guidelines

- Minimize critical path
  - Add register layers
  - Register balancing
  - Pipelining



Overall design guidelines

- Throughput vs. Latency
- Example: (XC6LX45-3FG484)
  - Iterative (low area: 19 registers, 41 LUTs)
    - Throughput: 4 bits / 4 clocks = 1 bit / clock
    - Latency: 4 clocks
    - Critical path: One multiplier delay (3.325ns) => 288.976MHz
  - Low-Latency (low latency: 23 registers, 46 LUTs)
    - Throughput: 4 bits / clock
    - Latency: 2 clocks
    - Critical path: Two multiplier delays (3.565ns) => 280.493MHz
  - Pipelined (high throughput: 29 registers, 45 LUTs)
    - Throughput: 4 bits / clock
    - Latency: 3 clocks
    - Critical path: One multiplier delay (3.455ns) => 289.457MHz

Synchronous designs

- Always make your design synchronous
  - Recommended for all FPGAs
- Failure to use synchronous design can potentially
  - Waste device resources
    - Not using a synchronous element will not save silicon and it wastes money
  - Waste performance
    - Reduces capability of end products; higher speed grades cost more
  - Lead to difficult design process
    - Difficult timing specifications and tool-effort levels
  - Cause long-term reliability issues
    - Probability, race conditions, temperature, and process effects
- Synchronous designs have
  - Few clocks
  - Synchronous resets
  - No gated clocks; instead, clock enables

Synchronous designs

- RESET
  - Always use synchronous RESET (if at all possible)

```
-- Flip-Flop with Positive Edge Clock
-- and Asynchronous Reset VHDL
-- Coding Example

process (C, CLR)
Begin
      if (CLR = '1') then
            Q <= '0';
      elsif (C'event and C='1') then
            Q <= D;
      end if;
end process;
```

```
-- Flip-Flop with Positive Edge Clock and
-- Synchronous Reset VHDL Coding
-- Example

process (C)
begin
      if (C'event and C='1') then
            if (R='1') then
                  Q <= '0';
            else
                  Q <= D;
            end if;
      end if;
end process;
```

FSM

```
1  ----------------------------------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ----------------------------------------------------------
5  ENTITY <entity_name> IS
6     PORT (clk, rst: IN STD_LOGIC;
7           input: IN <data_type>;
8           output: OUT <data_type>);
9  END <entity_name>;
10 ----------------------------------------------------------
11 ARCHITECTURE <architecture_name> OF <entity_name> IS
12    TYPE state IS (A, B, C, ...);
13    SIGNAL pr_state, nx_state: state;
14    ATTRIBUTE ENUM_ENCODING: STRING; --optional attribute
15    ATTRIBUTE ENUM_ENCODING OF state: TYPE IS "sequential";
16 BEGIN
```

#1 : Page 279-280

FSM

```
17    ------Lower section of FSM:------------
18    PROCESS (clk, rst)
19    BEGIN
20       IF (rst='1') THEN
21          pr_state <= A;
22       ELSIF (clk'EVENT AND clk='1') THEN
23          pr_state <= nx_state;
24       END IF;
25    END PROCESS;
```

#1 : Page 279-280

```
26      ------Upper section of FSM:------------
27      PROCESS (pr_state, input)
28      BEGIN
29         CASE pr_state IS
30            WHEN A =>
31               output <= <value>;
32               IF (input=<value>) THEN
33                  nx_state <= B;
34               ...
35               ELSE
36                  nx_state <= A;
37               END IF;
38            WHEN B =>
39               output <= <value>;
40               IF (input=<value>) THEN
41                  nx_state <= C;
42               ...
43               ELSE
44                  nx_state <= B;
45               END IF;
```

#1 : Page 279-280

```
42              ...
43              ELSE
44                 nx_state <= B;
45              END IF;
46           WHEN ...
47        END CASE;
48     END PROCESS;
49     ------Output section (optional):-------
50     PROCESS (clk, rst)
51     BEGIN
52        IF (rst='1') THEN
53           new_output <= <value>;
54        ELSIF (clk'EVENT AND clk='1') THEN --or clk='0'
55           new_output <= output;
56        END IF;
57     END PROCESS;
58  END <architecture_name>;
59  ------------------------------------------------------
```

#1 : Page 279-280

RESET

- RESET
  - Never use global RESET (if a all possible)

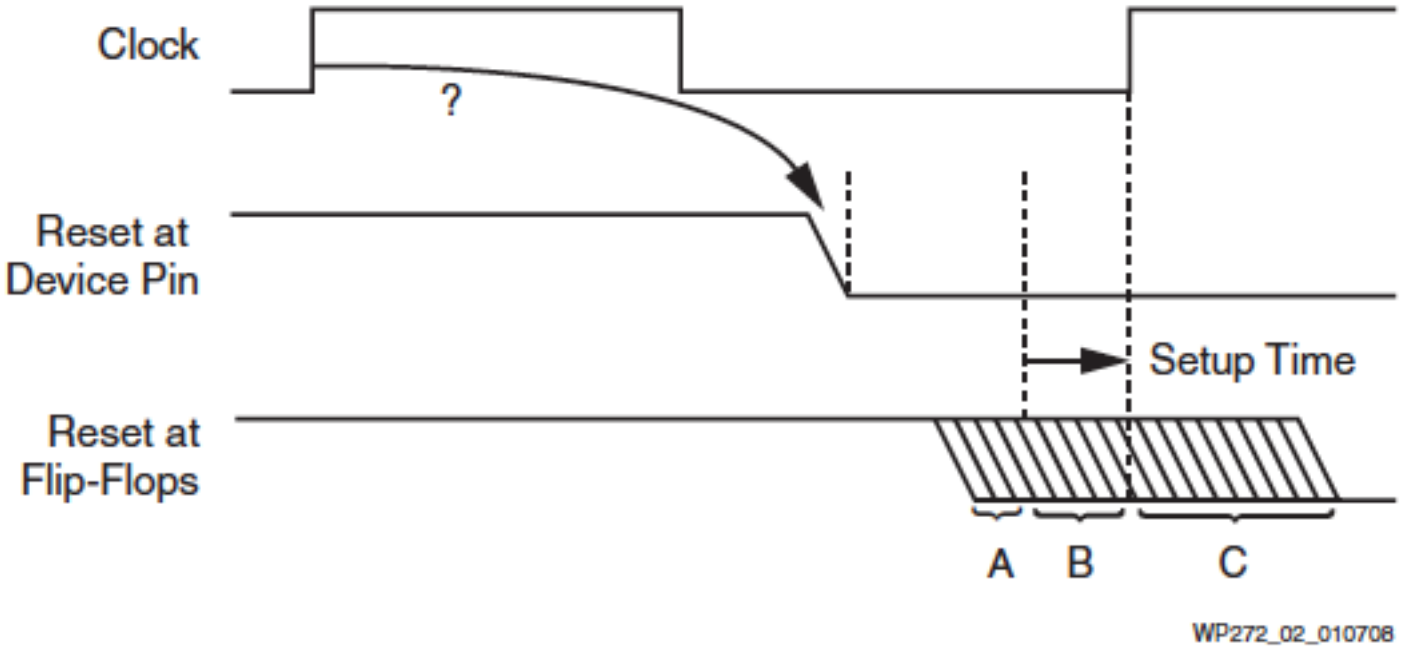


Figure 2: **Reset Timing Diagram - Asserted Asynchronously to the Clock**

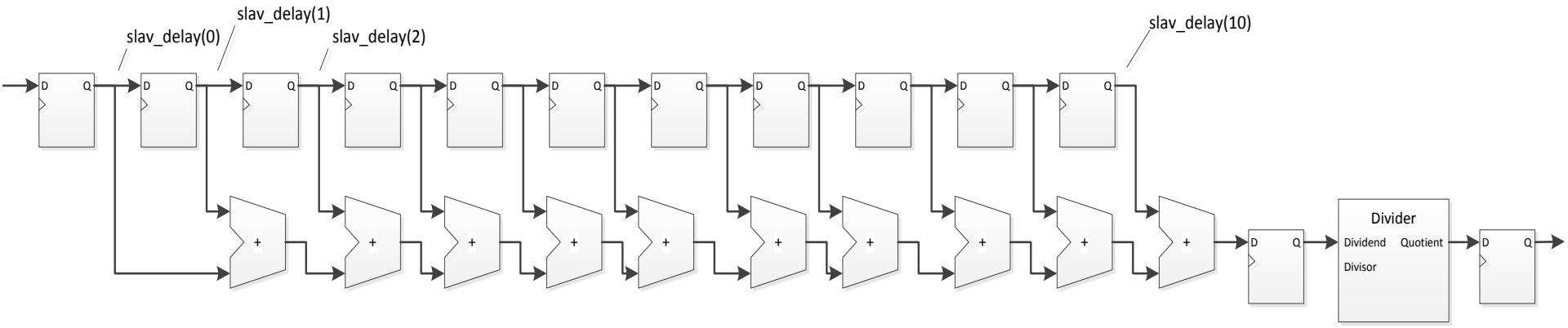
Feedback on exercise from last week

Exercise

- Exercise
  - Sliding average filter
    - Understand what is happening
      - Note the use of a variable! (*sum*)
      - Simulate if necessary

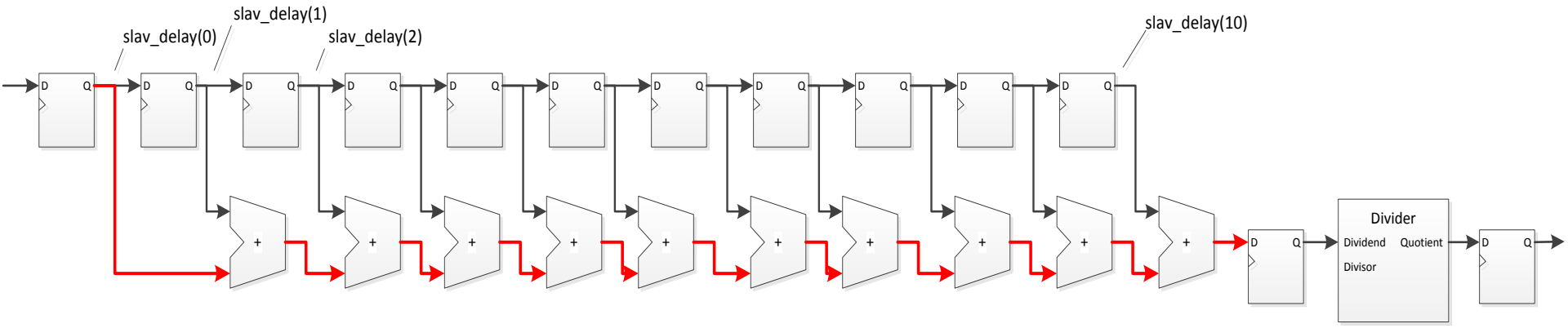
Exercise

- Sliding Average Filter
  - Divisor = 11



Exercise

- Sliding Average Filter
  - Divisor = 11

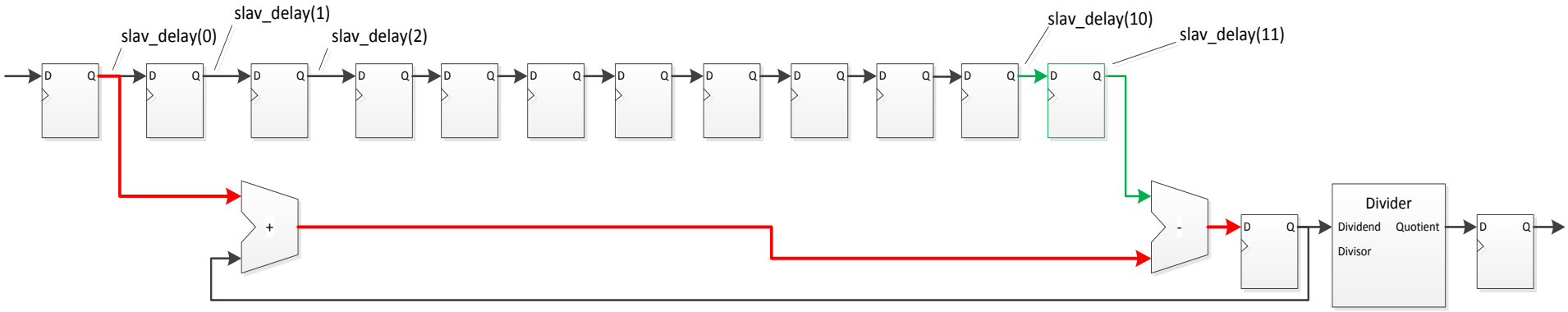


Exercise

- Exercise
  - Sliding average filter
    - Understand what is happening
      - Note the use of a variable! (*sum*)
      - Simulate if necessary
    - Improve the filter! (Target frequency: +400 MHz)
      - Original data:
        - Slice count: 87
        - Slice Register count: 475 (MAP: 404)
        - Slice LUT count: 214 (MAP: 270)
        - Maximum frequency: 147.009 MHz
        - Latency: 19 clock periods
- Save your files for next week!

Exercise

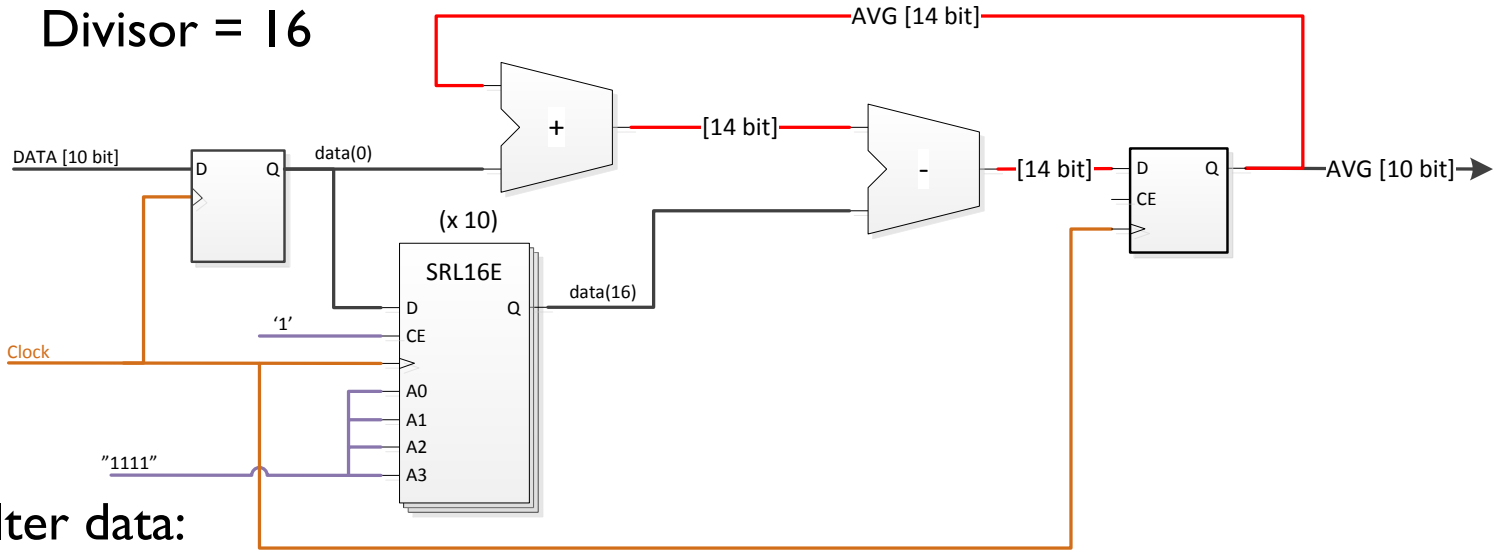
- Improved Sliding Average Filter (Divisor = 11)



- Original Filter data:
  - Slice count: 87
  - Slice Register count: 475 (MAP: 404)
  - Slice LUT count: 214 (MAP: 270)
  - Maximum frequency: 147.009 MHz
  - Latency: 19 clock periods
- Improved Filter Data:
  - Slice count: 75
  - Slice Register count: 424 (MAP: 318)
  - Slice LUT count: 145 (MAP: 206)
  - Maximum frequency: 409.752MHz
  - Latency: 19 clock periods

Exercise

- Alternative Sliding Average Filter using FPGA primitives
  - Divisor = 16



- Filter data:
  - Slice count: 10
  - Slice Register count: 24
  - Slice LUT count: 34 (MAP: 32)
  - Maximum frequency: 251.946MHz
  - Latency: 2 clock periods

Task for next lesson

- Read ALL the mandatory material:
  - See BlackBoard (From Monday)
- Work on / Update your blockdiagram, detailing:
  - All top level-modules/functions and their relationship (connections)
  - Module connections must be detailed signal and bus lines, simple data-flow describtions are not enough.
  - Remember that the ADCs delay the R, G and B color signals relative to the H, and V sync signals…
    - Figure out how much the R, G and B signals are delayed
    - Compensate for this delay, to ensure the H -and V-sync signal are in sync with the color lines.
- Group status presentation (5 minutes pr. group)