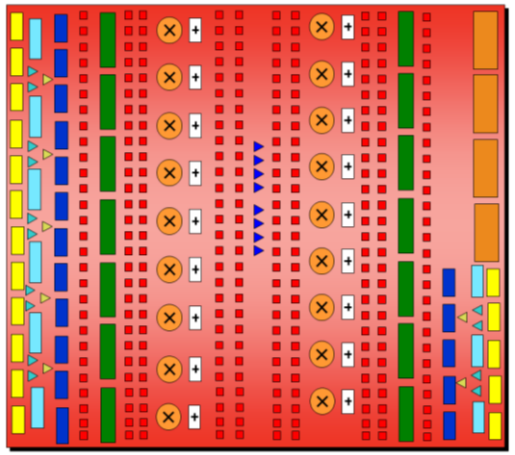
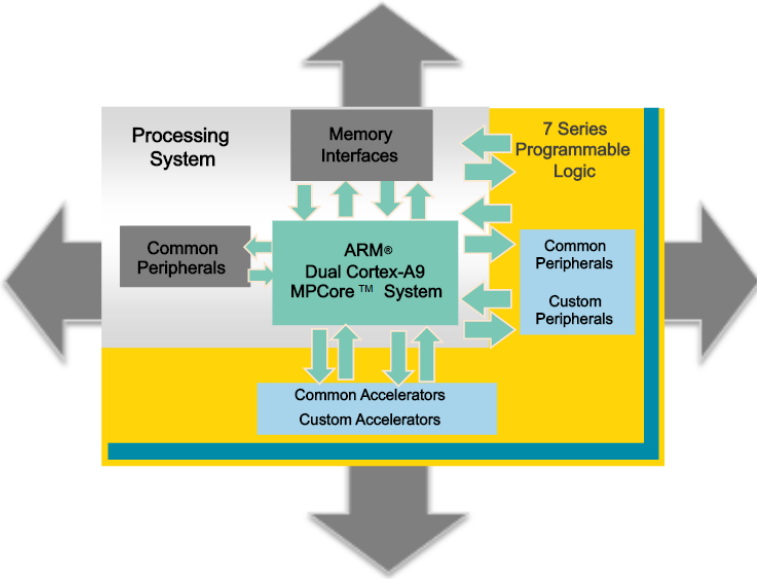
DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

A2A
Algorithms to Architecture

ECE 270: Embedded Logic Design

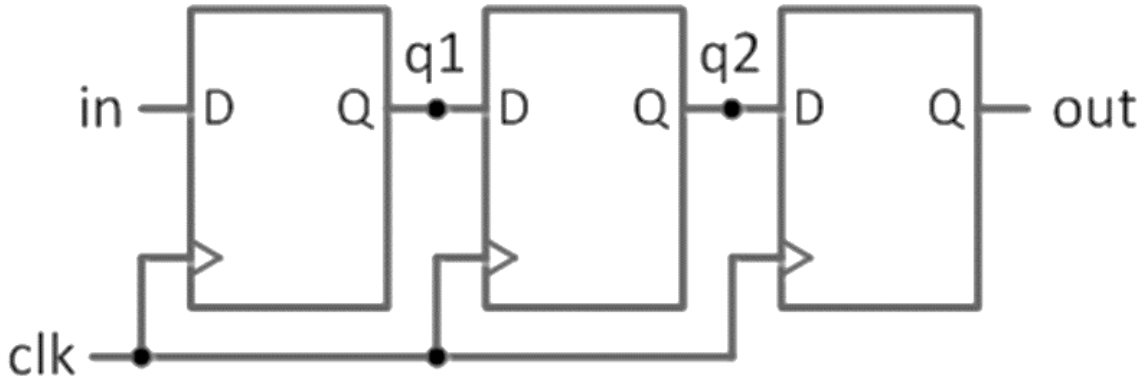




Opine Feedback

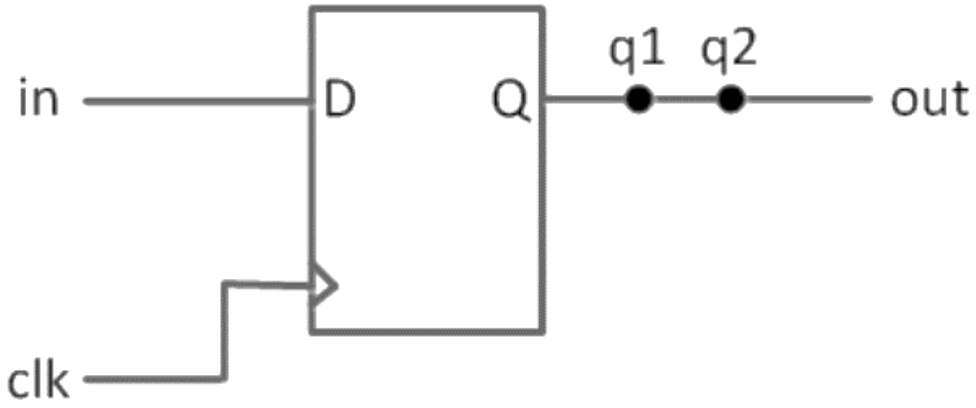
- **No of feedback:** 19% (lost opportunity and can be easily ignored)
- **We all need people who will give us feedback. That's how we improve: Bill Gates**
- **Feedback often tells you more about the person who is giving it than about you: Stephen Covey**
- **Couple of suggestions:** TA office hour slots and Lab homework deadline (Done)
- **Reminder:** Below 30 marks get F grade and 3 plagiarism cases reported so far

Behavioral Modeling

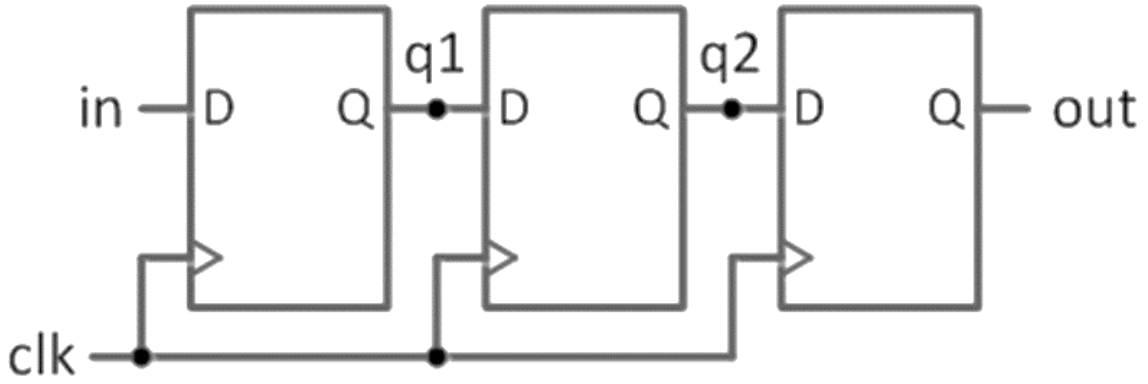



```
module blocking (in,clk,out) ;
   input in, clk ;
   output reg out ;
   reg q1, q2 ;
   always @(posedge clk)
      begin
         q1 = in ;
         q2 = q1 ;
         out = q2 ;
      end
endmodule
```

"at each rising clock edge, q1 = in,
after that, q2 = q1 = in
after that, out = q2 = q1 = in
Therefore, out = in"



Behavioral Modeling

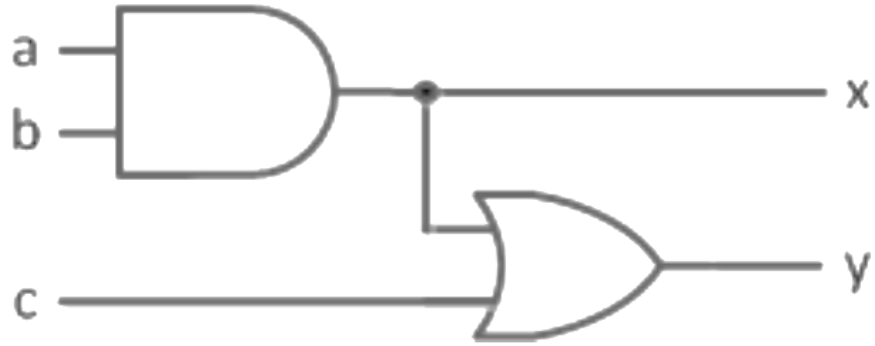



```
module nonblocking (in,clk,out) ;
   input in, clk ;
   output reg out ;
   reg q1, q2;
   always @(posedge clk)
      begin
         q1 <= in ;
         q2 <= q1 ;
         out <= q2 ;
      end
endmodule
```

"at each rising clock edge, q1, q2 and out simultaneously receive the old values of in, q1 and q2. Therefore, out = q2"

- Blocking assignments do not reflect the intrinsic behaviour of multi-stage sequential logic
- Use non-blocking assignments for sequential always blocks

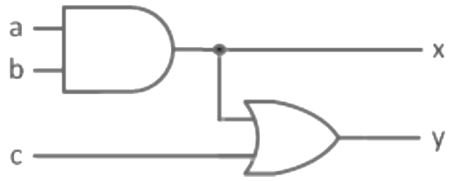
Behavioral Modeling




```
module blocking (a, b, c, x, y) ;
    input a, b, c ;
    output reg x, y ;
    always @ (a or b or c)
      begin
        x = a & b ;
        y = x | c ;
      end
endmodule
```

```
module nonblocking (a, b, c, x, y) ;
    input a, b, c ;
    output reg x, y ;
    always @ (a or b or c)
      begin
        x <= a & b ;
        y <= x | c ;
      end
endmodule
```

Behavioral Modeling




```
module blocking (a, b, c, x, y) ;
   input a, b, c ;
   output reg x, y ;
   always @ (a or b or c)
      begin
         x = a & b ;
         y = x | c ;
      end
endmodule
```

```
module nonblocking (a, b, c, x, y) ;
   input a, b, c ;
   output reg x, y ;
   always @ (a or b or c)
      begin
         x <= a & b ;
         y <= x | c ;
      end
endmodule
```

- Given initial conditions:

  a=1, b=1, c=0, x=1, y=1.
- a changes to 0. always block triggered.
- Blocking behaviour of simulator: 1st calculates x = a & b = 0. Then calculates y = x | c = 0
- Non-Blocking behavior of simulator: Concurrently calculates x(new) = a & b = 0. y = x(old) | c = 1
- Non-blocking assignment *do not* reflect the intrinsic behavior of multi-stage combinational logic
- While non-blocking assignments can be hacked to simulate correctly (expand sensitivity list), its not elegant
- **Guideline**: Use blocking assignments for combinational always blocks

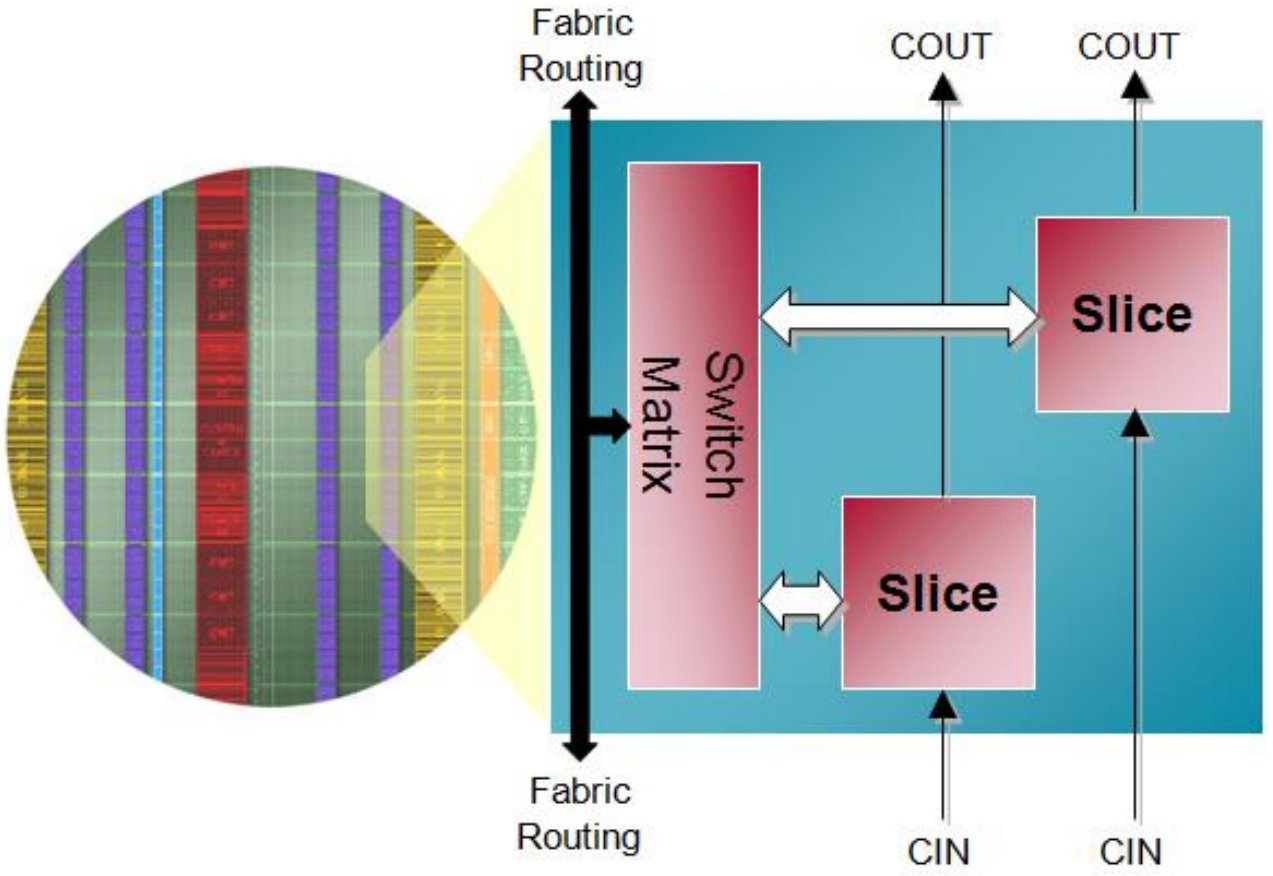
Behavioral Modeling

- When modelling sequential logic, use non-blocking assignments.
- When modelling combinational logic with an always block, use blocking assignments.
- When modelling both sequential and combinational logic within the same always block, use non-blocking assignments.
- Do not mix blocking and non-blocking assignments in the same always block.

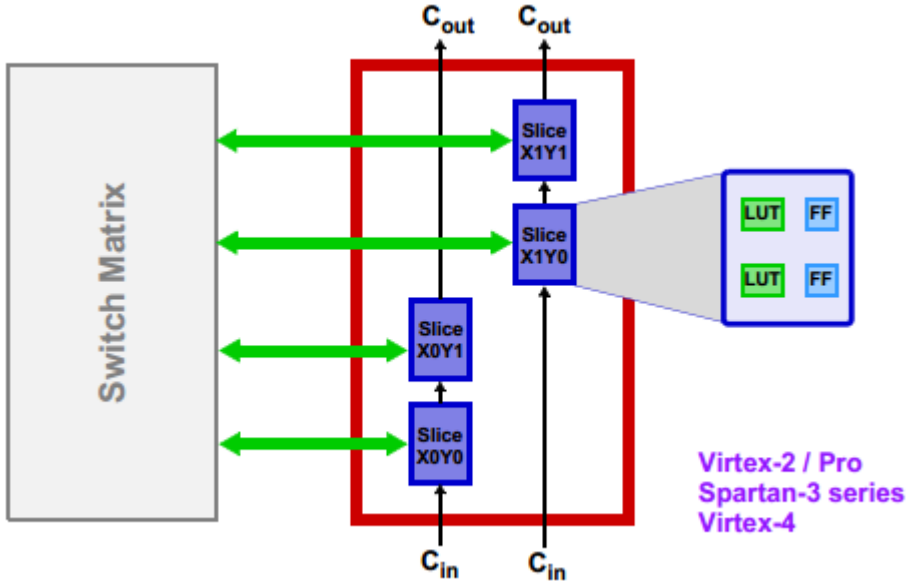
FPGA Architecture

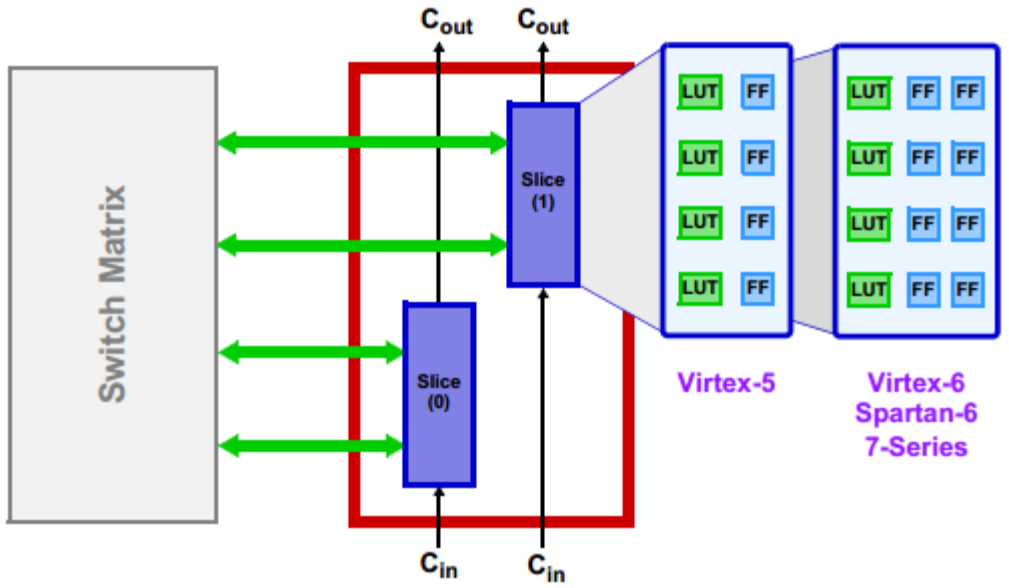
Configurable Logic Block (CLB)

- Primary resource for design in Xilinx FPGAs
- CLB contains more than one slice
- Connected to switch matrix for routing to other FPGA resources
- Carry chain runs vertically in a column from one slice to the one above



Configurable Logic Block (CLB)

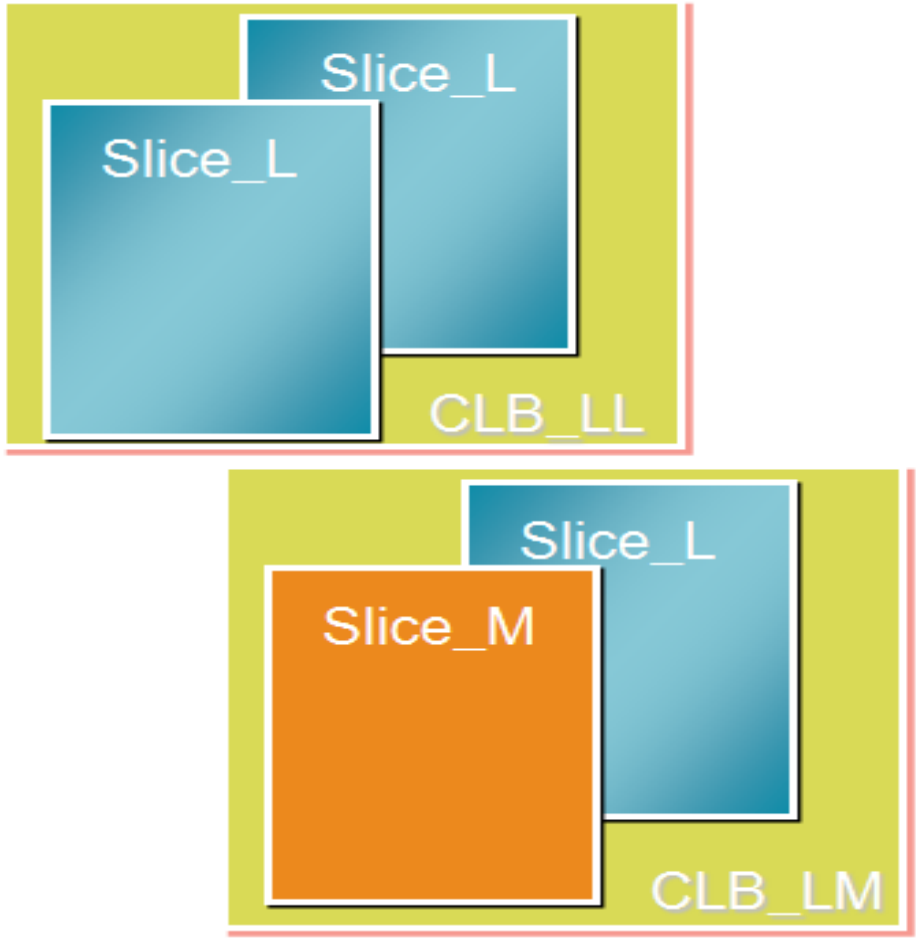




Slices	LUTs	Flip-Flops	Arithmetic and Carry Chains
2	8	16	2

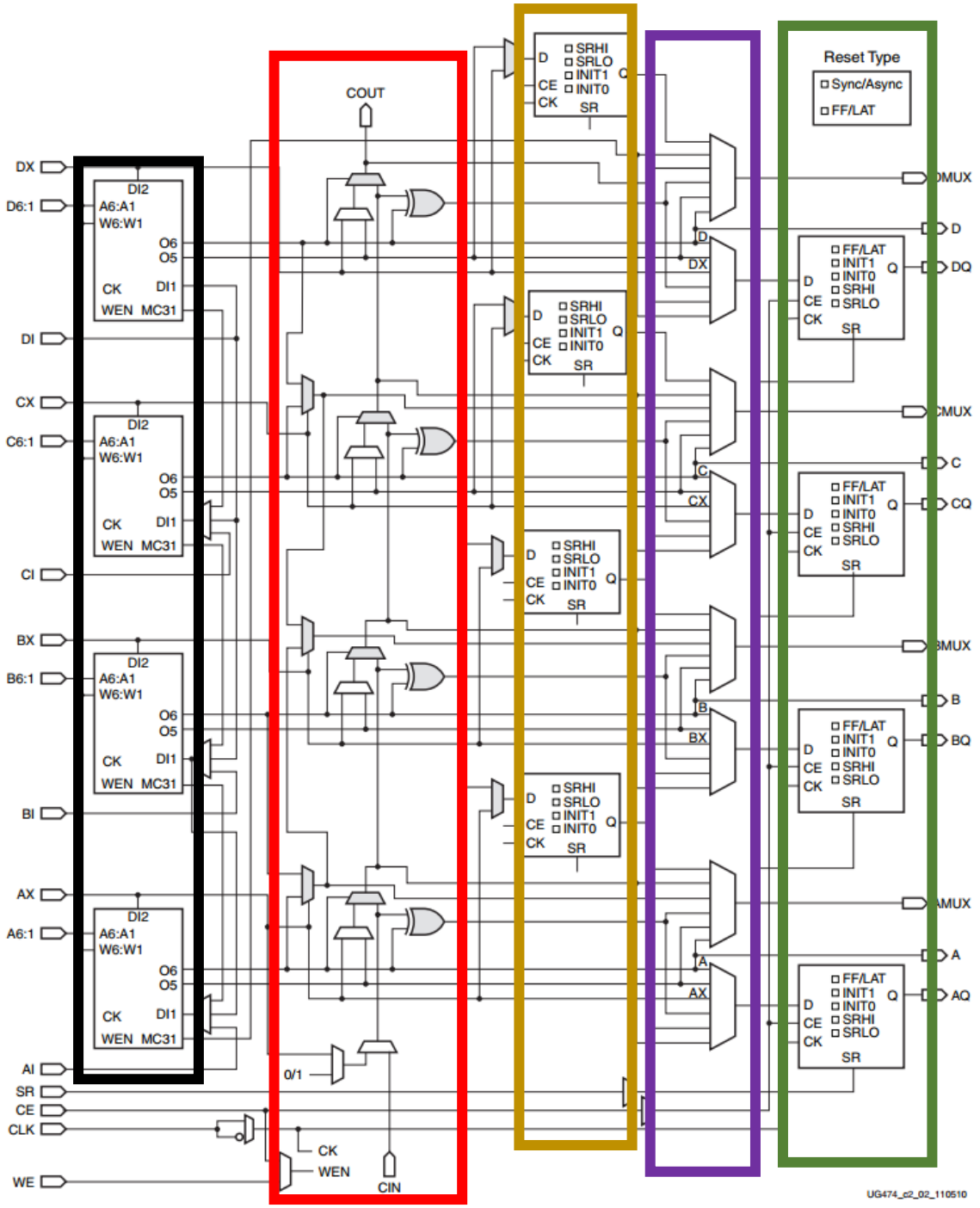
Types of CLB Slices

- **SLICEM: Full slice**
  - LUT can be used for logic and memory/SRL
- **SLICEL: Logic and arithmetic only**
  - LUT can only be used for logic (not memory/SRL)
- Each CLB can contain two SLICEL or a SLICEL and a SLICEM.
- In the 7-series FPGAs, approximately ¼ of slices are SLICEM, the remainder are SLICEL.

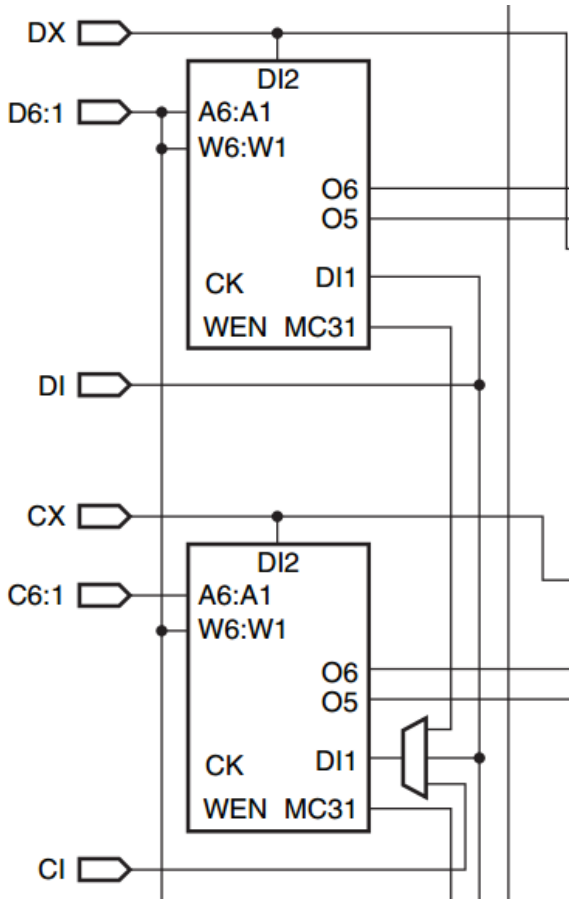


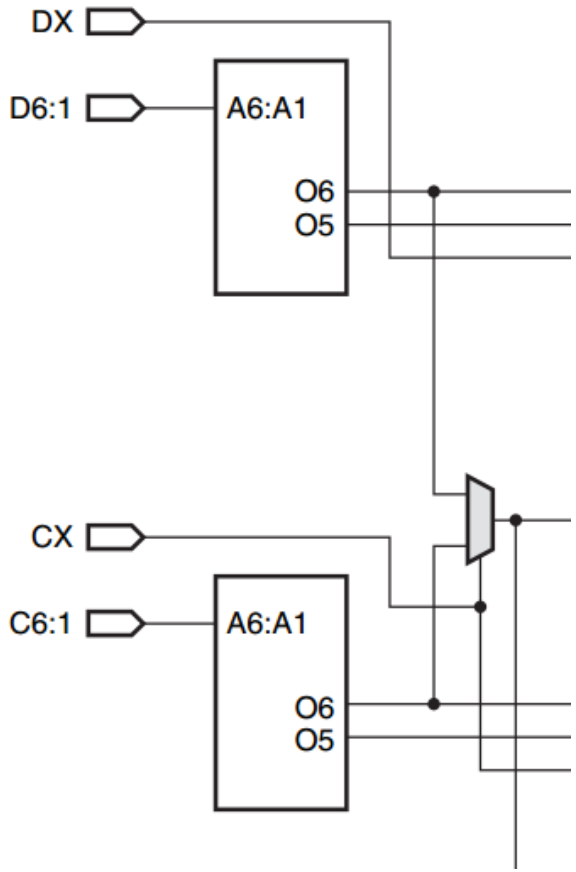
Slice Resource

- Four six-input Look-Up Tables (LUT)
- Multiplexers
- Carry chains
- Four flip-flops/latches
- Four additional flip-flops
- Four 6-input LUTs and their eight flip-flops as well as multiplexers and arithmetic carry logic form a slice, and two slices form a CLB.



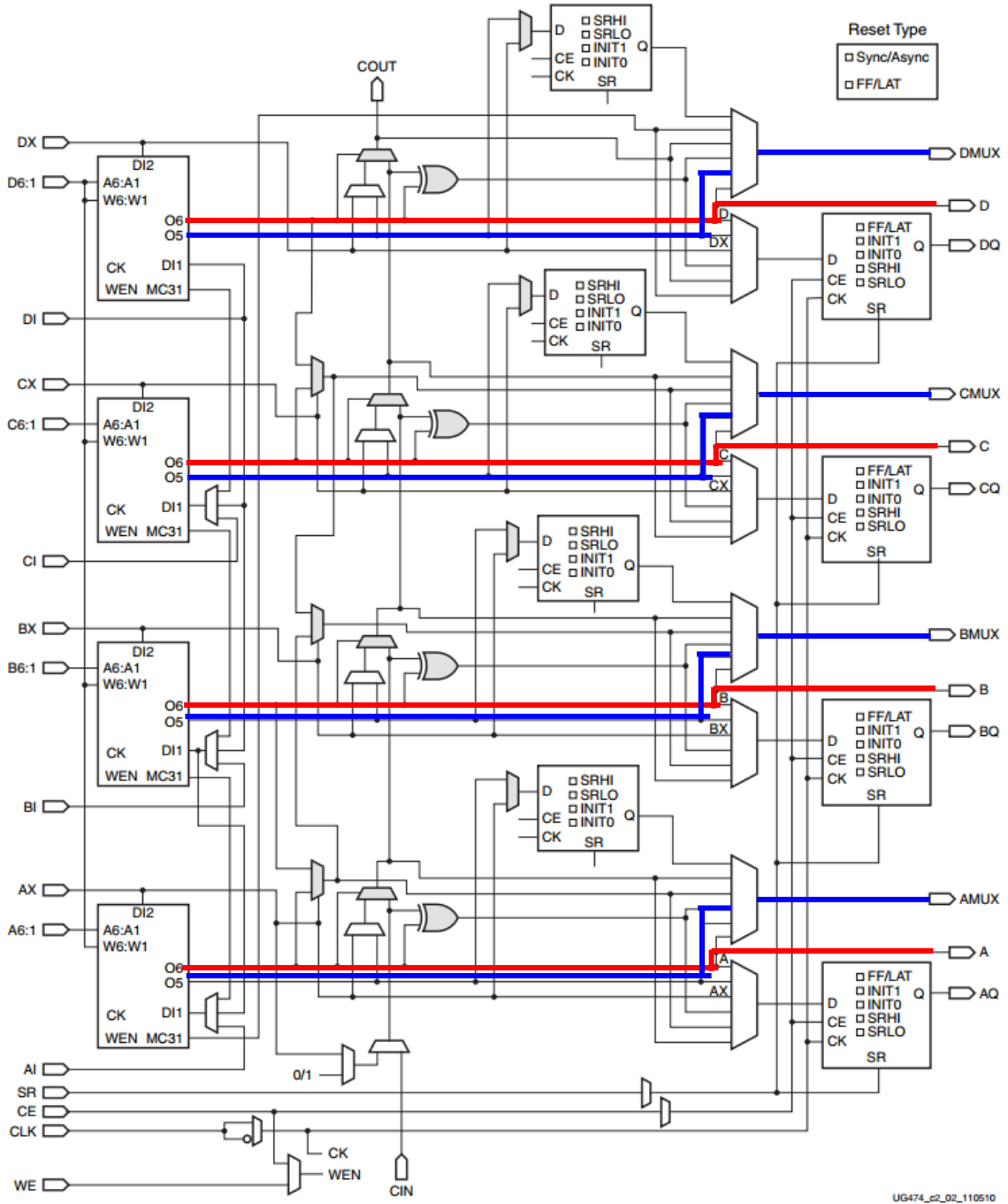
SLICEM Vs SLICEL





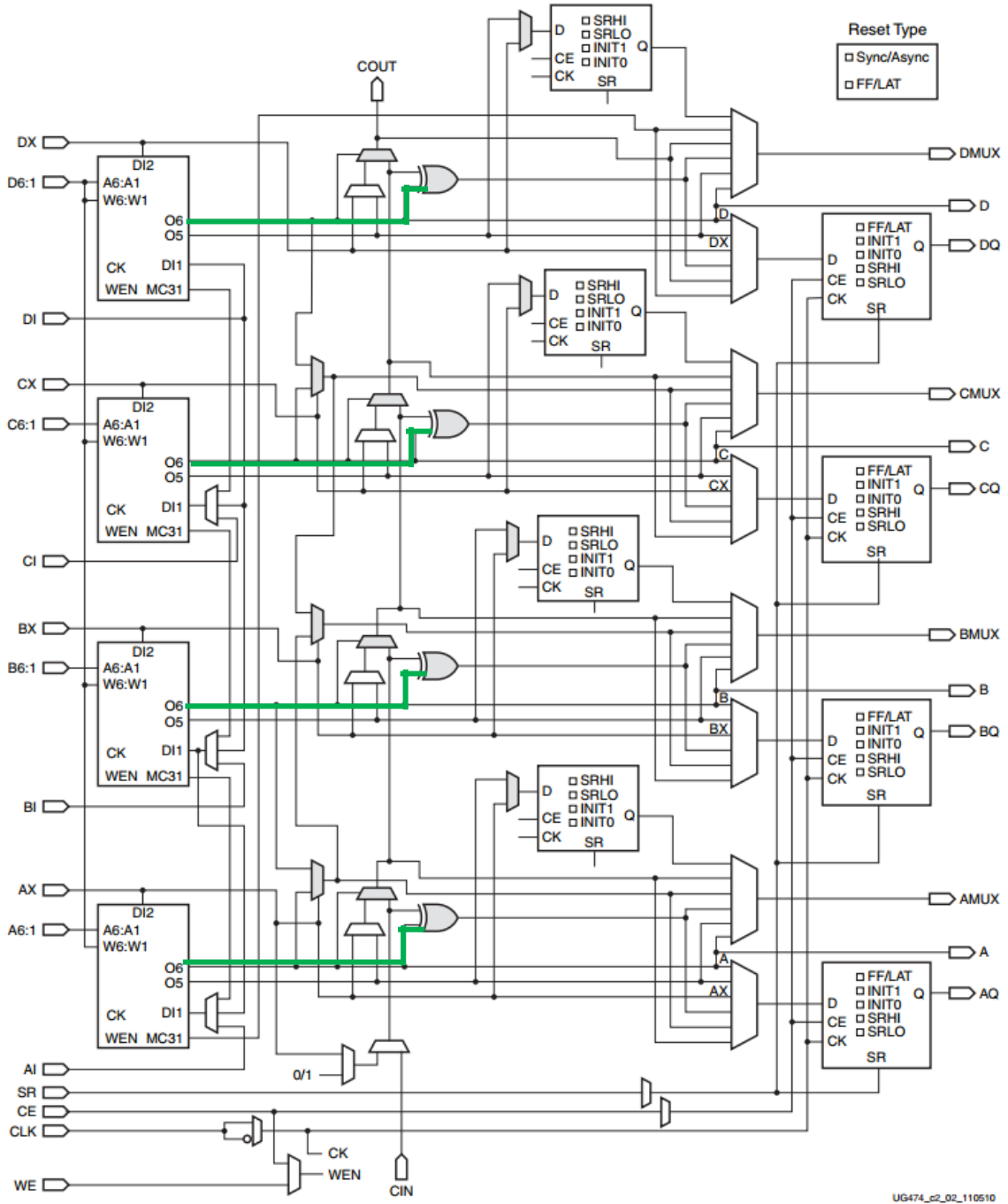
LUT

- Signals from the LUT can:
  - Exit the slice (through A, B, C, D output for O6 or AMUX, BMUX, CMUX, DMUX output for O5)



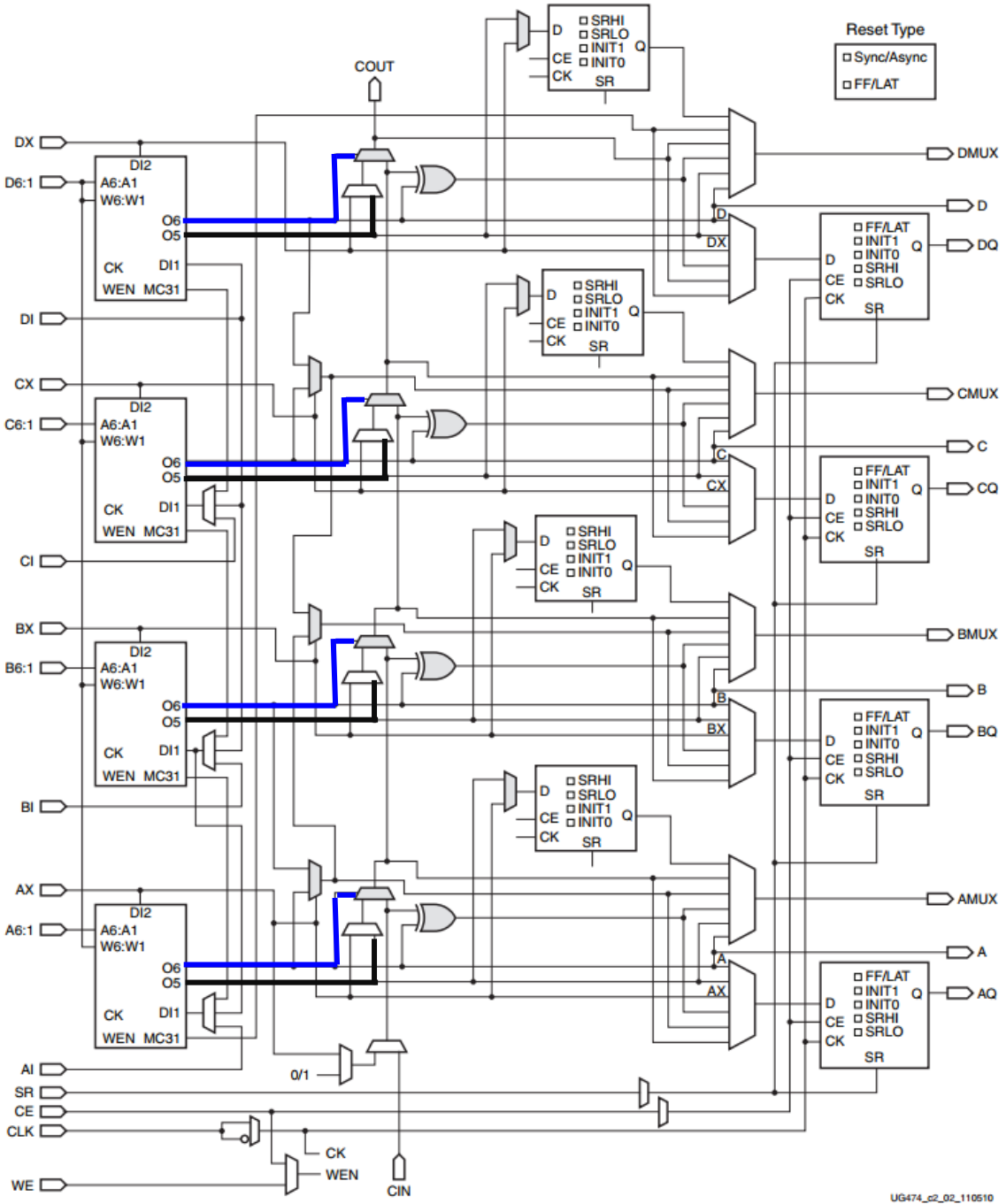
LUT

- Signals from the LUT can:
  - Exit the slice (through A, B, C, D output for O6 or AMUX, BMUX, CMUX, DMUX output for O5)
  - Enter the XOR dedicated gate from an O6 output



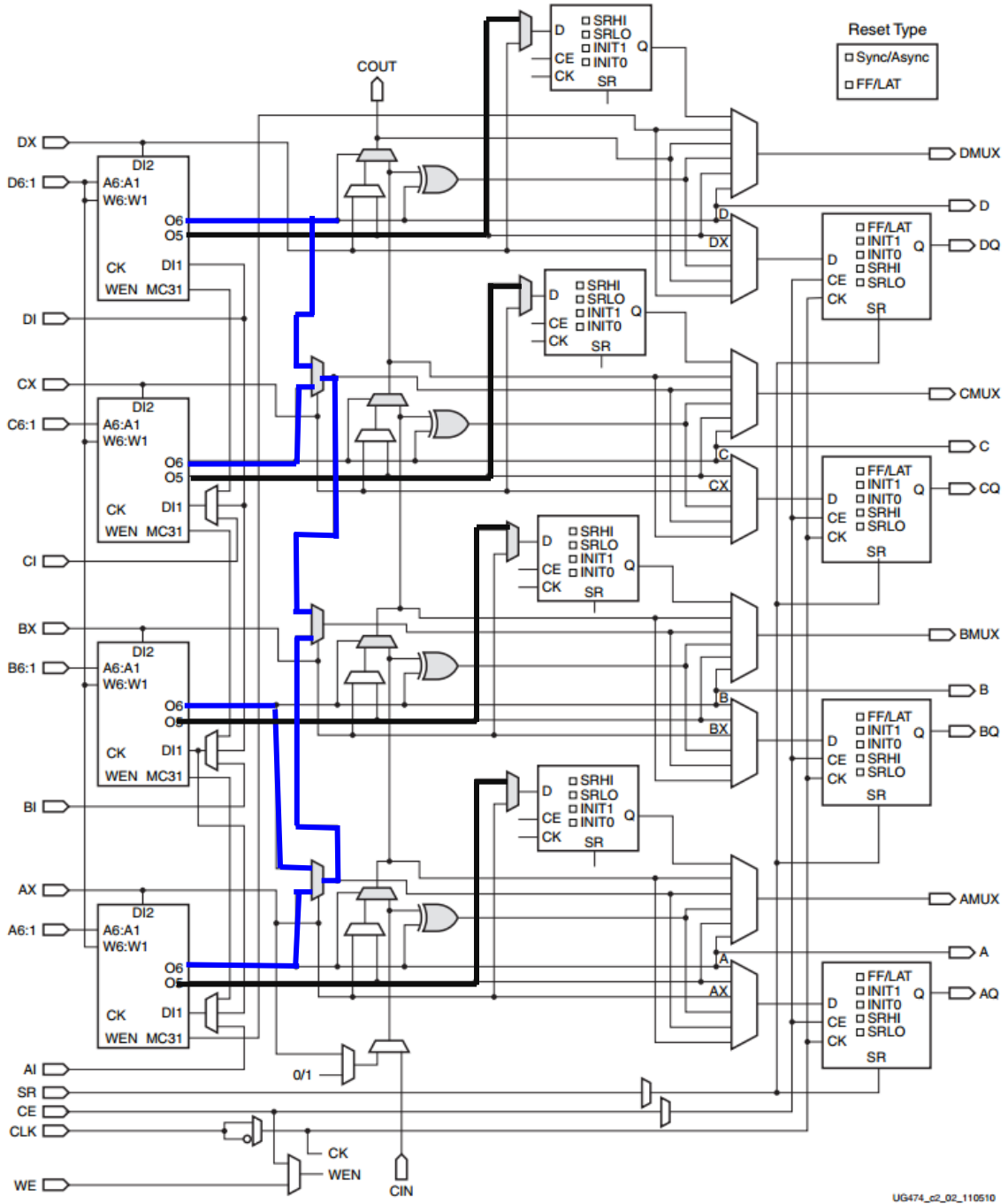
LUT

- Signals from the LUT can:
  - Exit the slice (through A, B, C, D output for O6 or AMUX, BMUX, CMUX, DMUX output for O5)
  - Enter the XOR dedicated gate from an O6 output
- **Enter the carry-logic chain from an O5 output**
- Enter the select line of the carry-logic multiplexer from O6 output



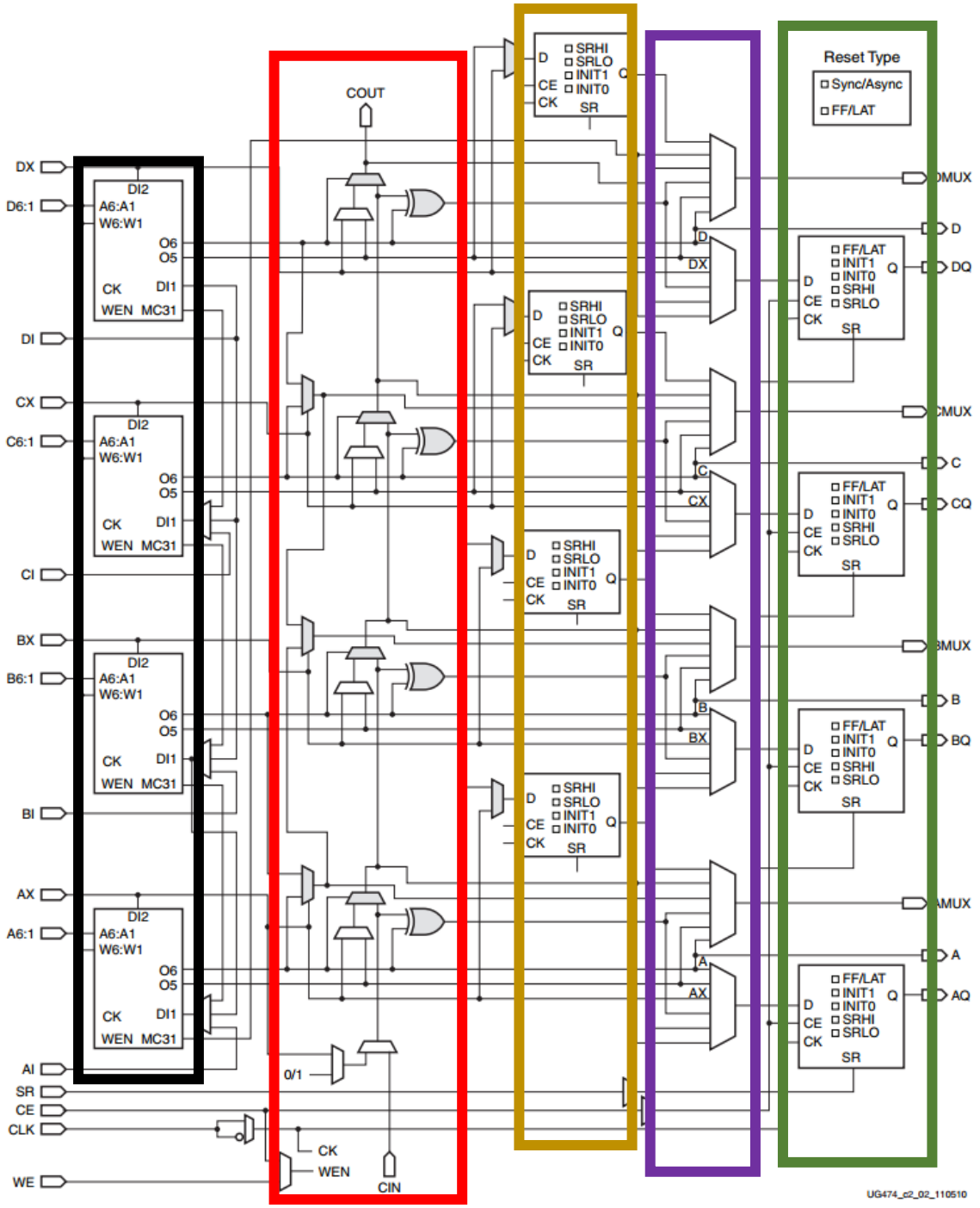
LUT

- Signals from the LUT can:
- Exit the slice (through A, B, C, D output for O6 or AMUX, BMUX, CMUX, DMUX output for O5)
- Enter the XOR dedicated gate from an O6 output
- Enter the carry-logic chain from an O5 output
- Enter the select line of the carry-logic multiplexer from O6 output
- **Feed the D input of the storage element**
- Go to F7AMUX/F7BMUX wide multiplexers from O6 output



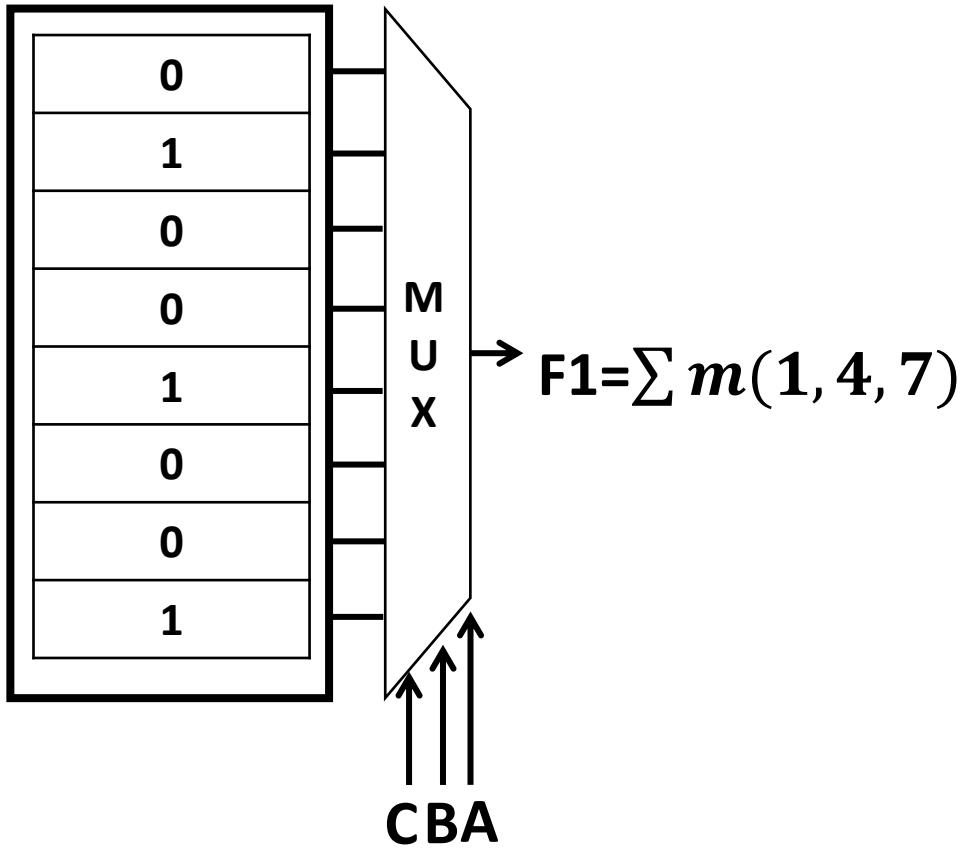
Slice Resource

- Four six-input Look-Up Tables (LUT)
- Multiplexers
- Carry chains
- Four flip-flops/latches
- Four additional flip-flops
- Four 6-input LUTs and their eight flip-flops as well as multiplexers and arithmetic carry logic form a slice, and two slices form a CLB.

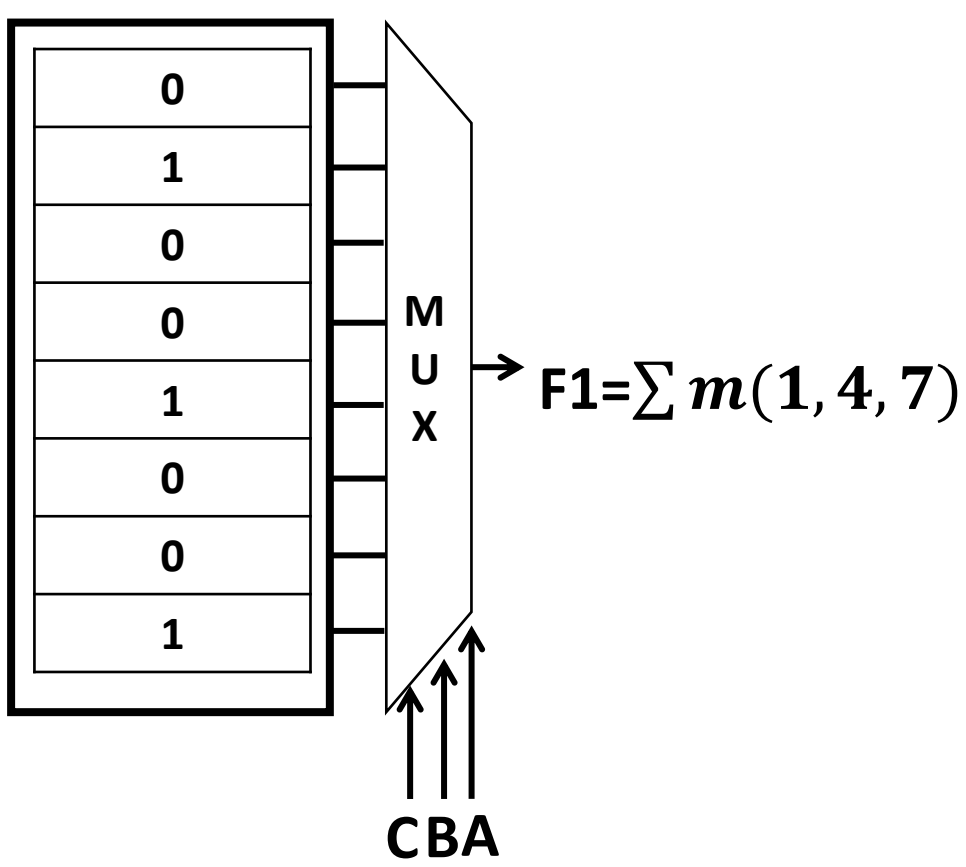


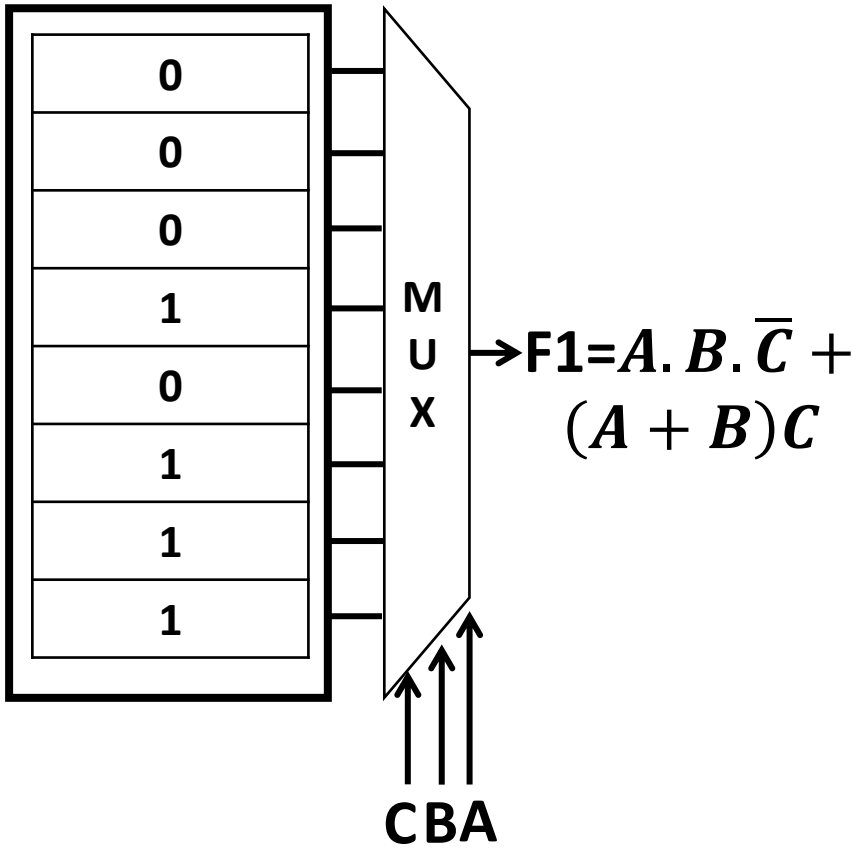
3 Input LUT

C	B	A	F1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



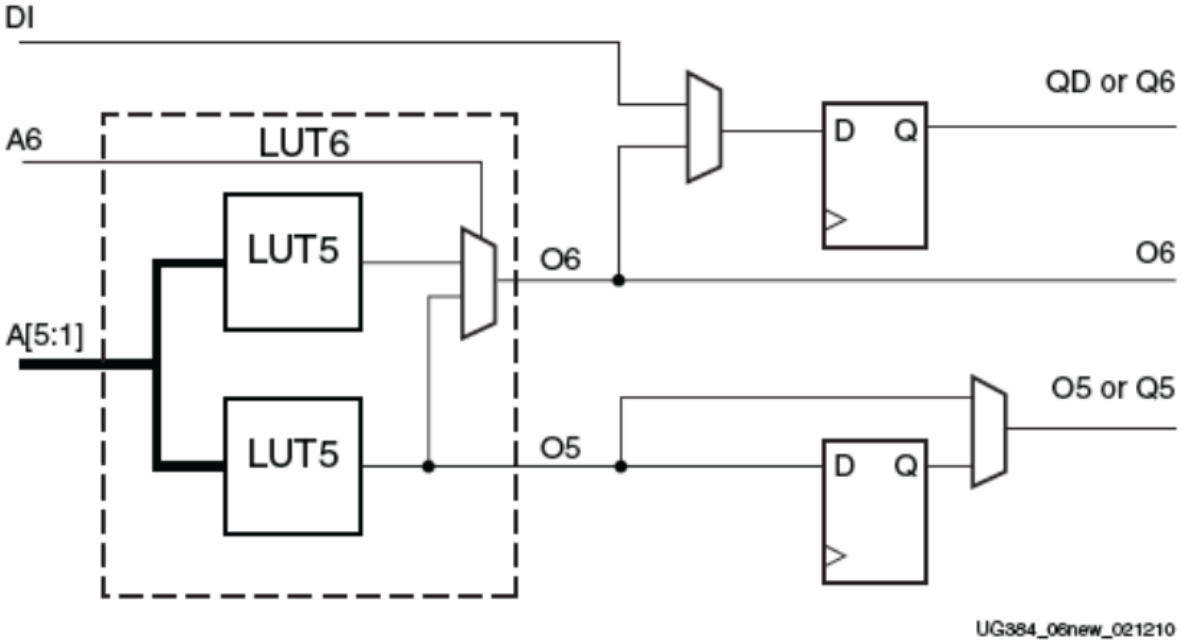
3 Input LUT


0
1
0
0
1
0
0
1
MUX
F1=$\sum m(1,4,7)$
CBA


0
0
0
1
0
1
1
1
MUX
F1=$A.B.\overline{C}+(A+B)C$
CBA

LUT

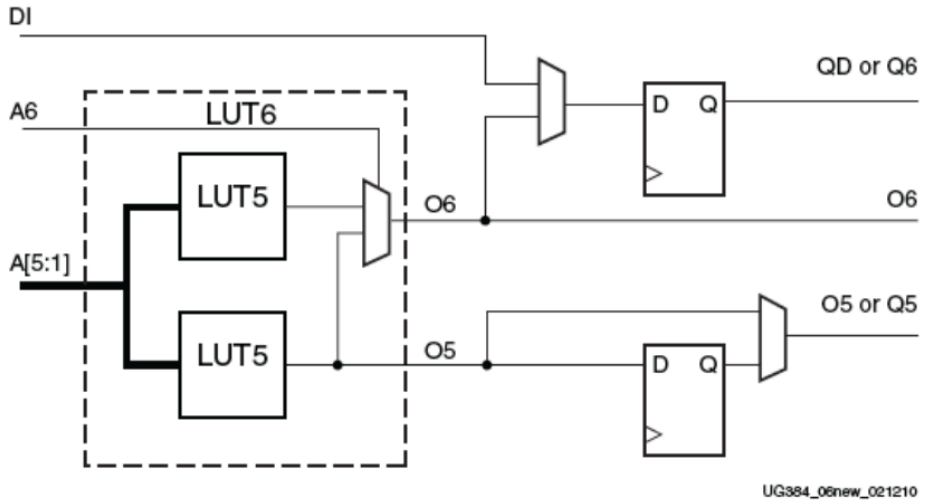
- There are six independent inputs (A inputs - A1 to A6) and two independent outputs (O5 and O6)



3-Input LUT for Logic

- Any single arbitrarily defined 3-input Boolean function
- A 3-input function uses: A1-A3 inputs and O3 output
- Two arbitrarily defined 2-input Boolean functions, as long as these two functions share common inputs
- Two 2-input or less functions use: A1–A2 inputs, A3 driven High, O2 and O3 outputs
- Two arbitrarily defined Boolean functions of 1 input

6-Input LUT for Logic



- Any arbitrarily defined six-input Boolean function
- A six-input function uses: A1-A6 inputs and O6 output
- Two arbitrarily defined five-input Boolean functions, as long as these two functions share common inputs
- Two five-input or less functions use: A1–A5 inputs, A6 driven High, O5 and O6 outputs
- Two arbitrarily defined Boolean functions of 3 and 2 inputs or less