



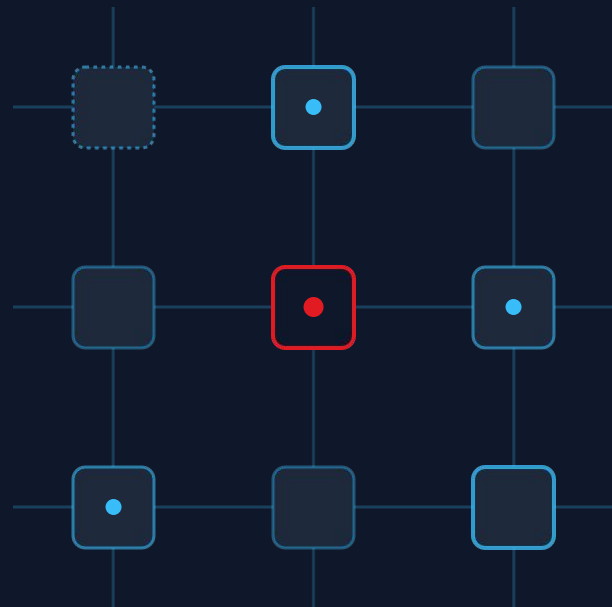
6.2050 Tutorial Session 1

Intro to FPGA Architecture

Primitives and HDL → HW

Subhi Abordan

Fall 2026



Why These Extra Tutorial Sessions?

01 Understanding Over Coding

With AI handling more basic coding, the real value lies in understanding how designs are synthesized into physical circuits, not just writing lines of code.

02 Performance & Modeling

Learn to approach complex hardware engineering from a rigorous performance and modeling perspective.

03 Hardware-Level Intuition

Understanding physical hardware makes you a better designer and coder, showing exactly what code looks like when synthesised.

04 Project & Industry Skills

Cover practical tricks, and core concepts you will directly use in your project and later in internships.

05 Demystify "Magic" Tools

Understand exactly what synthesis and compilation tools (Vivado) do to transform your code and map it onto the hardware.

Topics we will cover

01 FPGA Architecture, Primitives, HDL to HW

How different logic gets placed on the FPGA, trade-offs. CLBs, LUTs, Carry chains, FF, shift registers, ...

02 Vivado, Clock Trees, and Registers

How Vivado works, basic clock distribution, DSP blocks, and multipliers.

03 Timing Situation Solutions

Techniques and solutions for fixing setup/hold timing violations in designs.

04 CDC & Clock Measurement

Clock Domain Crossing (CDC) & design questions for measuring clock frequencies.

05 Performance Modeling Project

Memory on FPGA, and exploring memory-compute tradeoffs, pipelining, latency, and hardware modeling.

06 Flow Control & FIFOs Project

Managing backpressure, understanding FIFOs, and implementing a Skid Buffer.

07 Networking & Ethernet

Core networking concepts and communication protocols, focusing on Ethernet.



Today's Schedule and Outlines

01 Review: Mux is a Universal Gate

Reviewing how multiplexers provide universal computation and logic functions.

02 FPGA Architecture

Overview of overall FPGA architecture, primitives, and hardware mapping.

03 Configurable Logic Blocks (CLBs)

LUTs: basic gates, muxes, reduction trees

Carry Chain: optimized OR reduction, Full Adder

Muxes: F7, F8, F9, wider-input LUTs

Distributed RAM: LUT-based memory elements

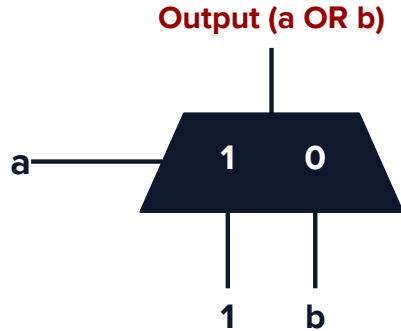
Shift Registers: counter implementation

Review: Mux is a Universal Gate

01 OR Gate

Output = $a + b$

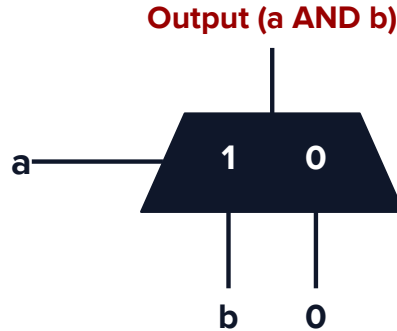
Sel=a $\rightarrow$ In1=1, In0=b



02 AND Gate

Output = $a \cdot b$

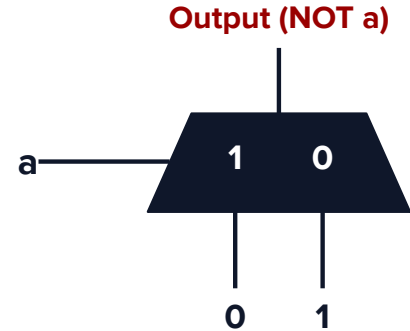
Sel=a $\rightarrow$ In1=b, In0=0

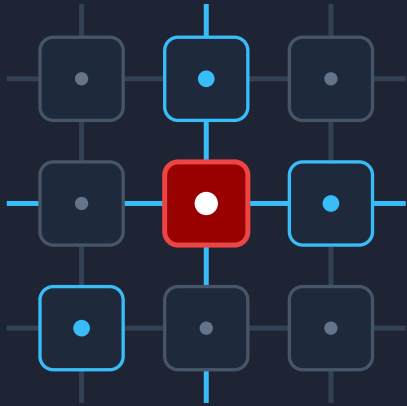


03 NOT Gate

Output = NOT a

Sel=a $\rightarrow$ In1=0, In0=1





FPGA Architecture Overview

FPGA Architecture Overview

Configurable Logic Blocks (CLBs): General-purpose logic

- **Lookup Tables (LUTs)**
- **Flip-flops**
- **Dedicated carry chains** accelerate arithmetic
- **Multiplexers**
- **Distributed RAM and Shift Registers**

Digital Signal Processing (DSP) Slices

- Dedicated multipliers, adders, accumulators
- Used for signal processing, ML, filters, etc.

Memory — On-chip storage

- **Distributed RAM** LUTs can become small RAM
- **Block RAM (BRAM)** provides larger dedicated memories
- **Ultra RAM (URAM)**



FPGA Architecture Overview

Clocking & I/O

- Dedicated clock networks distribute clocks across the chip
- Clock Generation: (PLL, MMCM)
- I/O blocks connect the internal fabric to physical pins

Routing — Connects everything

- Programmable wires and switches connect CLBs, DSPs, memories, and I/O

Key Takeaway: HDL does not “run” on an FPGA — synthesis maps HDL into these physical resources as circuits and the routing between them



FPGA Architecture Overview

Vivado Design Flow — From RTL to a Running FPGA Design

1. Design Input

Describe design & constraints

HDL & Constraints

top_level.**SV** / constraints.**xdc**

- HDL describes functional logic structure
- XDC specifies pin assignments, I/O standards & timing constraints

2. Synthesis

RTL to gate-level netlist

synth_design

TCL Command

- Maps RTL code into FPGA primitives (LUTs, FFs, DSPs, BRAMs)
- Generates **technology-specific unplaced** netlist

3. Implementation

Place & route on device

place_design

Logic Placement

route_design

Interconnect

- Places logic cells onto specific physical fabric sites
- Routes interconnections via programmable wires to satisfy timing

4. Timing Analysis

Verify timing requirements

report_timing_summary

TCL Command

- Validates setup/hold timing constraints and worst negative slack
- Analyzes clock domain crossings and critical paths

5. Bitstream Generation

Create configuration file

write_bitstream

TCL Command

- Compiles binary programming stream (.bit / .bin file)
- Encodes SRAM cell configurations for logic & routing switches

6. Program FPGA

Load bitstream onto device

Hardware Manager

Program via JTAG / Flash

- Downloads **bitstream** directly over JTAG interface or quad-SPI
- Hardware circuit becomes active and runs on physical FPGA silicon



FPGA Architecture Overview

2010

7-Series (28nm)



Introduced unified 28nm architecture (Artix/Kintex/Virtex) & Zynq-7000 SoC. FPGA Fabric = Programmable Logic (PL).

2016

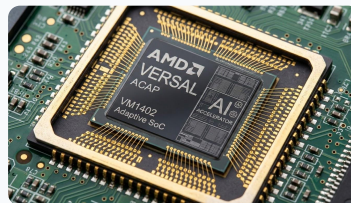
UltraScale+ (16nm)



16nm FinFET MPSoC integrating Programmable Logic alongside ARM Cortex CPUs and GPU.

2020

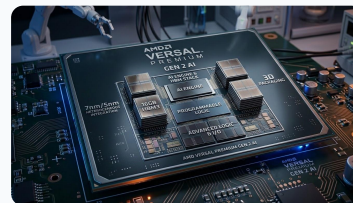
Versal ACAP (7nm)



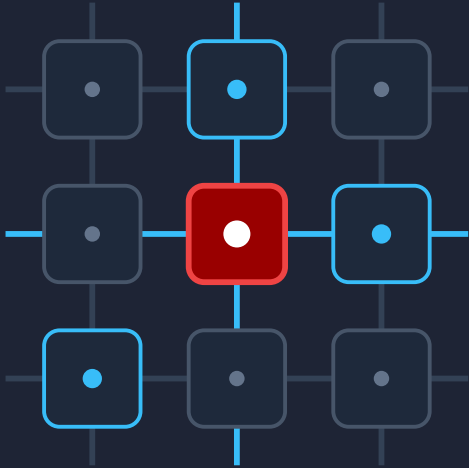
First Adaptive Compute Acceleration Platform (ACAP) incorporating dedicated AI Engines (AIE).

2026

Versal Gen 2



Next-gen Adaptive SoC integrating ARM Cortex-A78AE CPUs, AIE-ML v2, PCIe Gen6, and CXL 3.1.



Configurable Logic Block (CLB)

UltraScale+ Architecture

Configurable Logic Block (CLB)

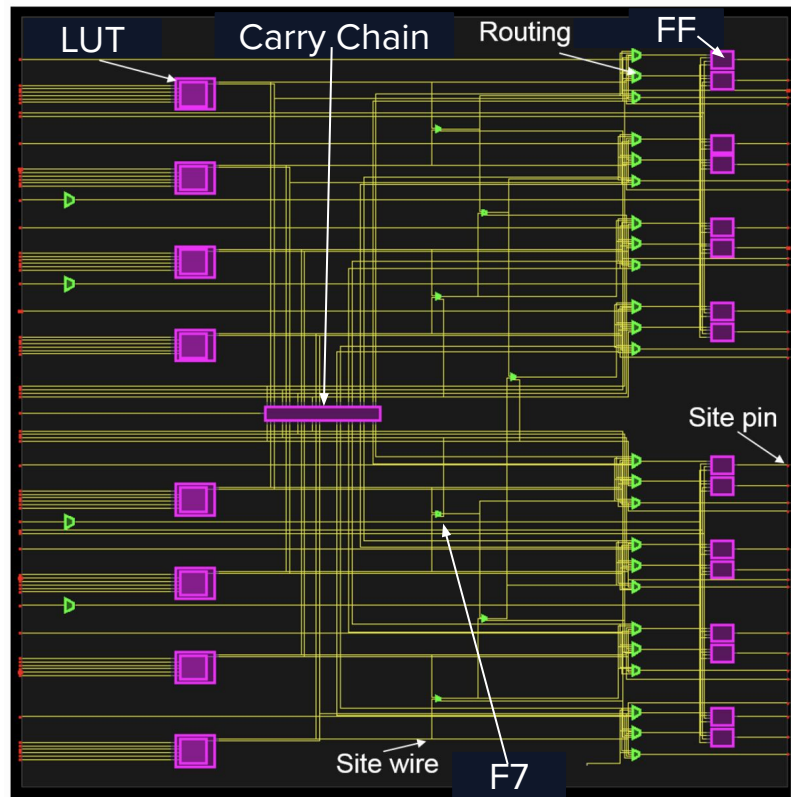
CLB Overview (UltraScale+)

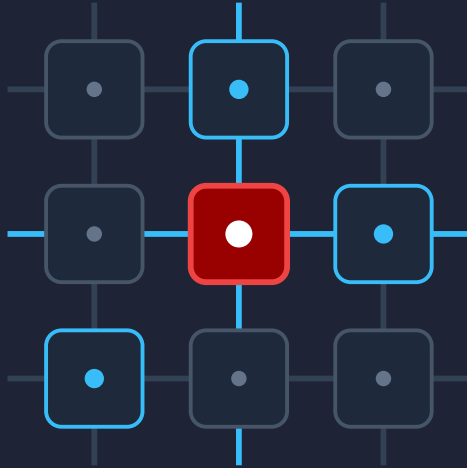
- **Main Resource:** Implementing general-purpose combinatorial and sequential circuits on the FPGA..
- **Logic Slice(s): SLICEL, SLICEM.**
 - L for logic, M for memory
 - They are the same, the only difference is SLICEM LUTs can be configured to be Distributed RAM, Shift Registers

Table 1: Logic Resources in One CLB Slice

CLB Slice	LUTs	Flip-Flops	Arithmetic and Carry Chains	Wide Multiplexers	Distributed RAM	Shift Registers
SLICEL	8	16	1	F7, F8, F9	N/A	N/A

CLB Slice	LUTs	Flip-Flops	Arithmetic and Carry Chains	Wide Multiplexers	Distributed RAM	Shift Registers
SLICEM	8	16	1	F7, F8, F9	512 bits	256 bits





I. Look-Up Tables (LUT)

UltraScale+ Architecture

Look-Up Tables (LUTs)

4-input LUT Conceptually

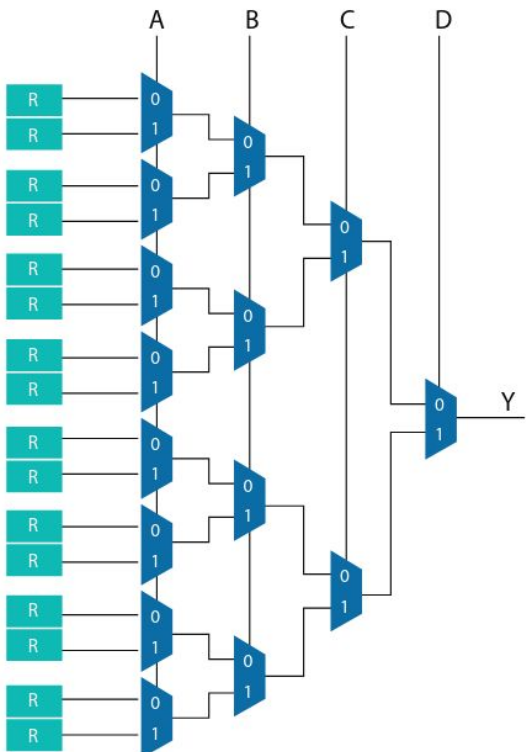
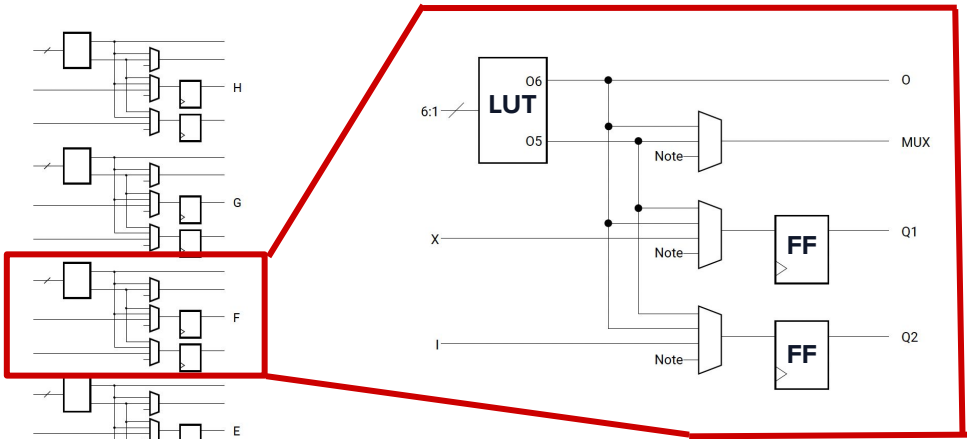
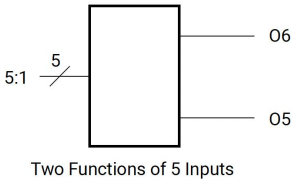
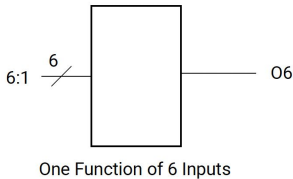


Figure 3: LUTs and Storage Elements in One Slice



Two possible configurations

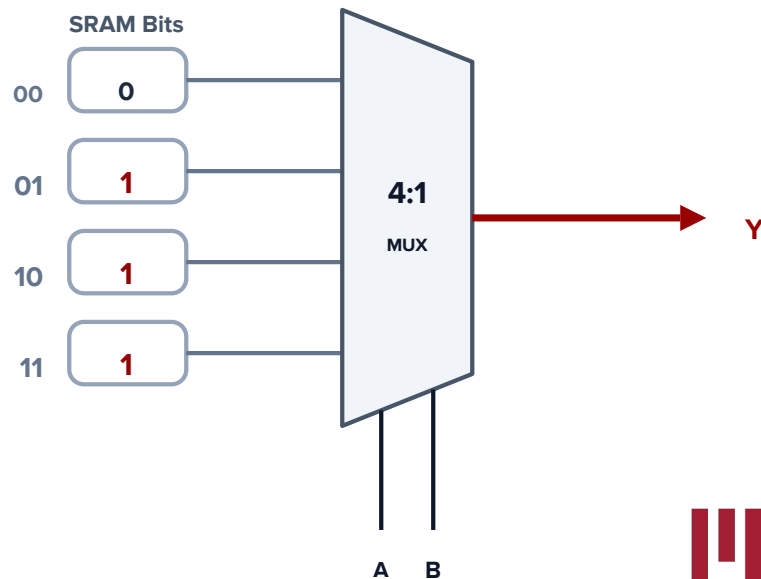


Example: 2-input OR Gate using 2-input LUT

- **Address Mapping:** Use the input as addresses to the SRAM memory in the LUT
- **Configuration Bitstream:** When we write code to implement logic gates, the synthesis tools generate the SRAM content that implements the logic (this defines the **bitstream**).
- **Scaling:** A 6-input LUT can generate any function with 6 input bits and 1 output bit

OR Gate Truth Table

Input A	Input B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

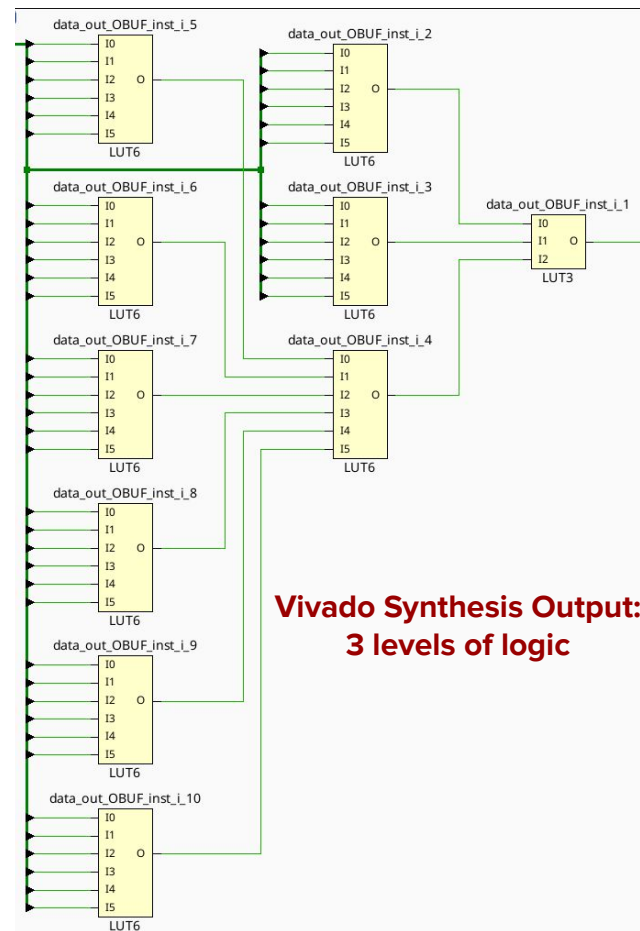
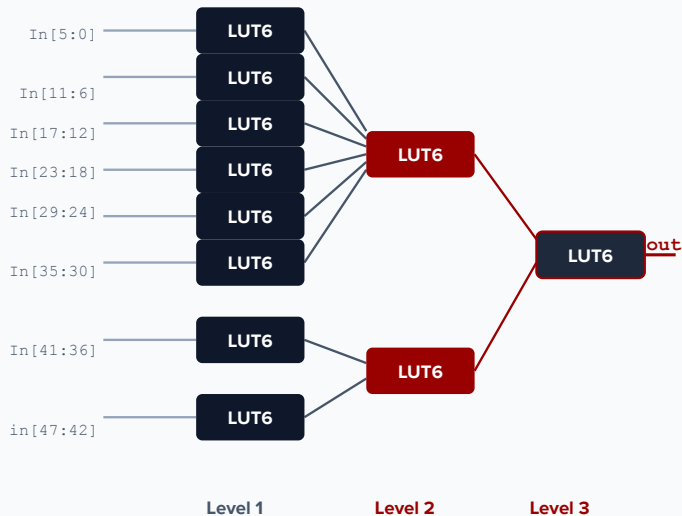


Example: OR Reduction

```
// 48-bit OR Reduction (3 levels of logic)
```

```
module or_reduce_48 (  
    input wire [47:0] data_in,  
    output logic data_out  
);  
    assign data_out = |data_in;  
endmodule
```

48-Bit Reduction Tree (3 Logic Levels)

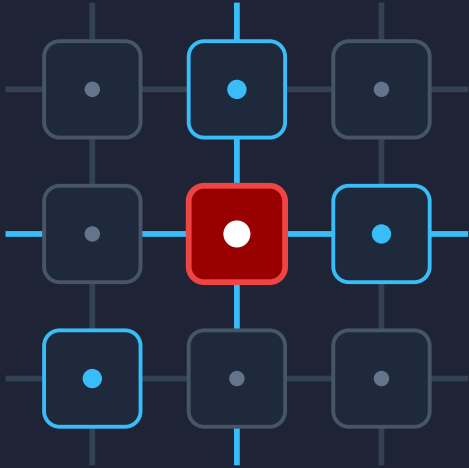


Vivado Synthesis Output:
3 levels of logic

Example: OR Reduction

Vivado Timing Report

Data Path						
Delay Type	Incr (ns)	Path ...	Location	Cell Pin	Cell	Netlist Resources
	(r) 0.000	0.000	Site: E16	data_in[15]		data_in[15]
net (fo=0)	0.000	0.000				data_in_IBUF[15]_inst/I
			Site: E16	PAD	IBUF_INST (IBUF)	data_in_IBUF[15]_inst/IBUF_INST/PAD
IBUF(Prop_IBUF_HPIOB_S_PAD_O)	(r) 0.548	0.548	Site: E16	O	IBUF_INST (IBUF)	data_in_IBUF[15]_inst/IBUF_INST/O
net (fo=1, routed)	0.000	0.548				data_in_IBUF[15]_inst/OUT
			Site: E16	I	IBUFCTRL_INST (IBUFCTRL)	data_in_IBUF[15]_inst/IBUFCTRL_INST/I
IBUFCTRL(Prop_IB...CTRL_HPIOB_S_I_O)	(r) 0.000	0.548	Site: E16	O	IBUFCTRL_INST (IBUFCTRL)	data_in_IBUF[15]_inst/IBUFCTRL_INST/O
net (fo=1, routed)	0.957	1.505				linear_or/data_in_IBUF[15]
			Site: SLI..._X81Y340	I3	data_out_OBUF_inst_i_10 (LUT6)	linear_or/data_out_OBUF_inst_i_10/I3
LUT6(Prop_H6LUT_SLICEL_I3_O)	(r) 0.150	1.655	Site: SLI..._X81Y340	O	data_out_OBUF_inst_i_10 (LUT6)	linear_or/data_out_OBUF_inst_i_10/O
net (fo=1, routed)	0.327	1.982				linear_or/data_out_OBUF_inst_i_10_n_0
			Site: SLI..._X81Y325	I5	data_out_OBUF_inst_i_4 (LUT6)	linear_or/data_out_OBUF_inst_i_4/I5
LUT6(Prop_H6LUT_SLICEL_I5_O)	(r) 0.150	2.132	Site: SLI..._X81Y325	O	data_out_OBUF_inst_i_4 (LUT6)	linear_or/data_out_OBUF_inst_i_4/O
net (fo=1, routed)	0.384	2.516				linear_or/data_out_OBUF_inst_i_4_n_0
			Site: SLI..._X81Y346	I2	data_out_OBUF_inst_i_1 (LUT3)	linear_or/data_out_OBUF_inst_i_1/I2
LUT3(Prop_H6LUT_SLICEL_I2_O)	(r) 0.150	2.666	Site: SLI..._X81Y346	O	data_out_OBUF_inst_i_1 (LUT3)	linear_or/data_out_OBUF_inst_i_1/O
net (fo=1, routed)	1.777	4.443				data_out_OBUF
			Site: A10	I	data_out_OBUF_inst (OBUF)	data_out_OBUF_inst/I
OBUF(Prop_OUTBUF_HDIOB_M_I_O)	(r) 2.381	6.824	Site: A10	O	data_out_OBUF_inst (OBUF)	data_out_OBUF_inst/O
net (fo=0)	0.000	6.824				data_out
			Site: A10	data_out		data_out



II. Carry chain

- Optimised Or-Reduction
- Ripple-Carry Adder

UltraScale+ Architecture

Carry Chain

Figure 8: Fast Carry Logic Path and Associated Elements

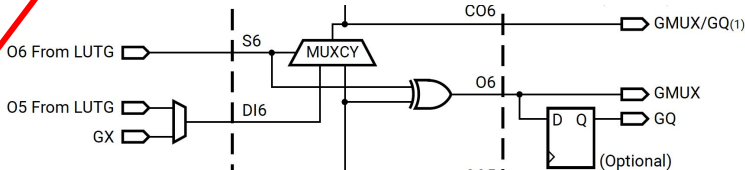
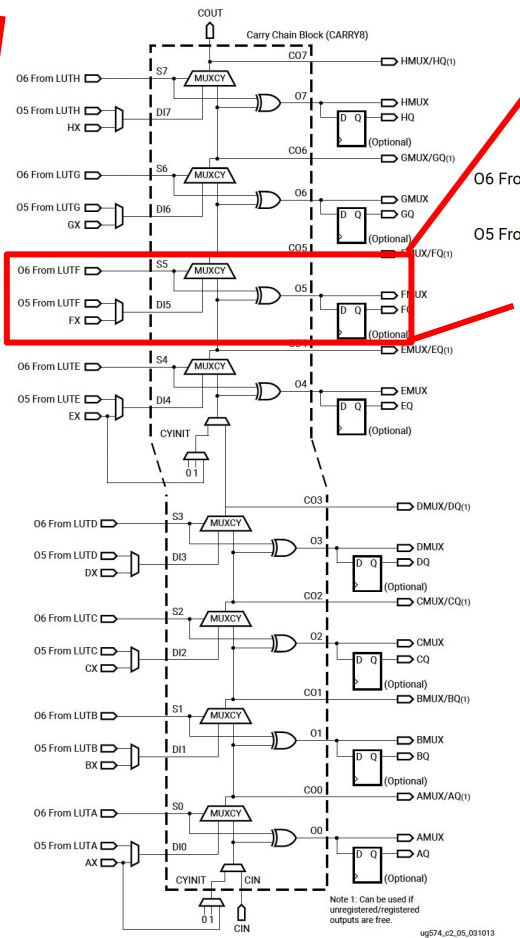
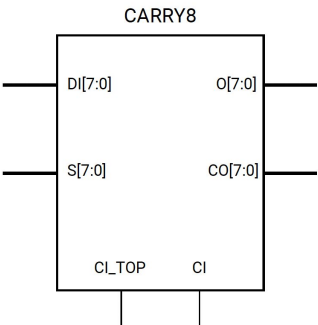


Figure 20: CARRY8 Primitive



UG574.c3.02_100813

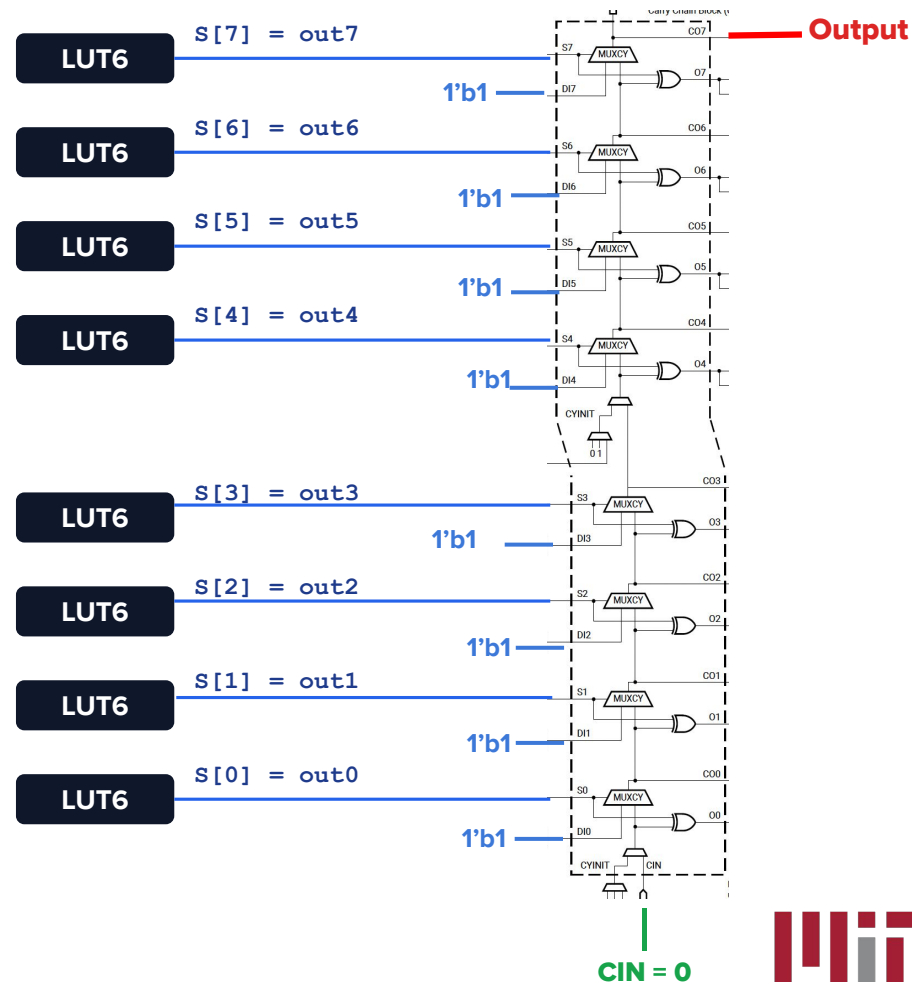


48-Bit OR Reduction Optimization

```
// 48-bit OR Reduction using CARRY8
module wide_or (
    input wire [47:0] din,
    output logic dout
);

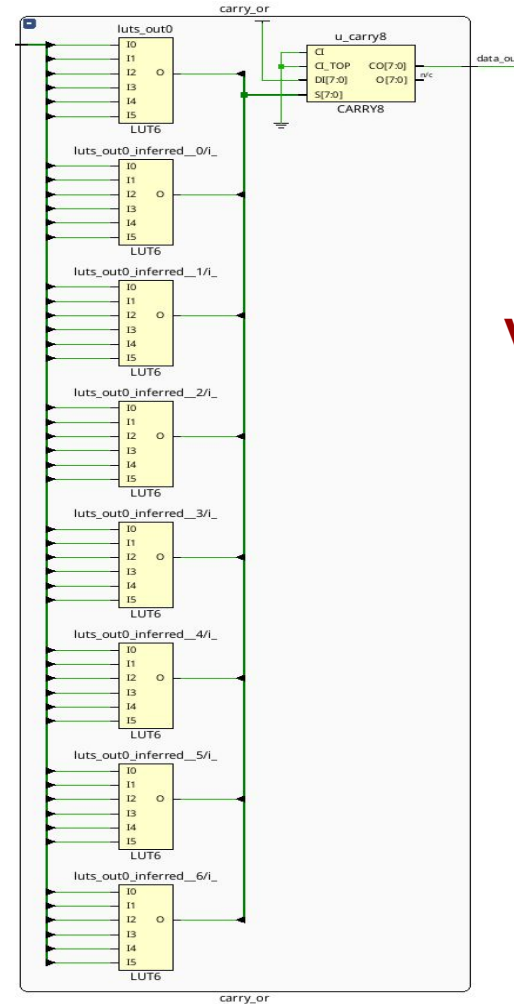
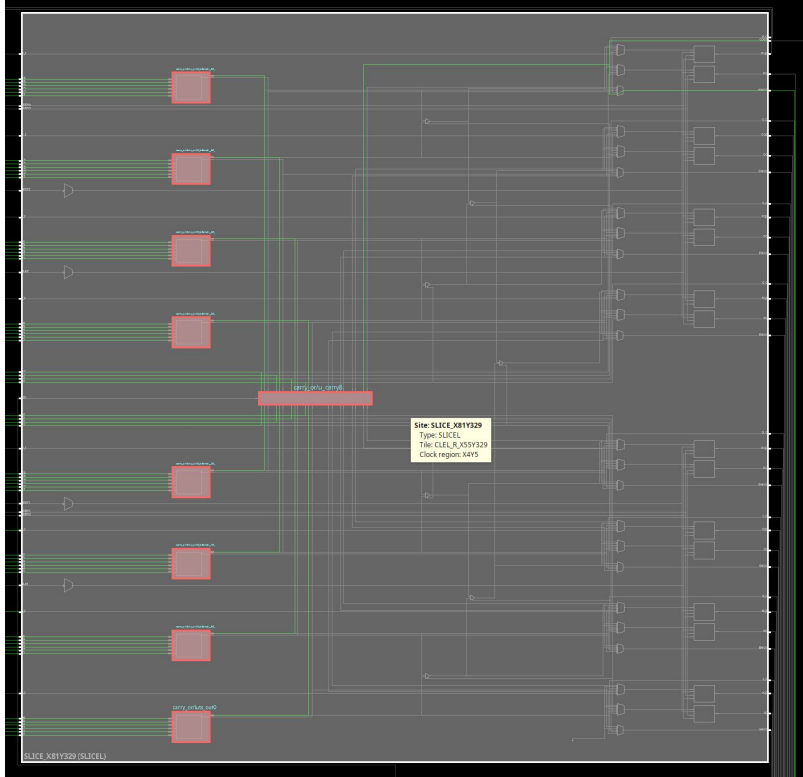
    assign dout0 = |(din[5:0]);
    assign dout1 = |(din[11:6]);
    // .....
    assign dout7 = |(din[47:42]);

    logic [7:0] output_carry;
    CARRY8 u_carry8 (
        .CO ( output_carry ), // out
        .O ( ),               // out
        .CI ( 0 ),            // in
        .CI_TOP ( 0 ),        // in
        .DI ( 8'b1111111 ),  // in
        .S ( {dout7,dout6,dout5,dout4,dout3,dout2,dout1,dout0} )
    );
    assign dout = output_carry[7];
endmodule
```



48-Bit OR Reduction Optimization

Vivado Placement



**Vivado Synthesis
Schematics:
2 levels of logic**

48-Bit OR Reduction Optimization

Vivado Timing Path

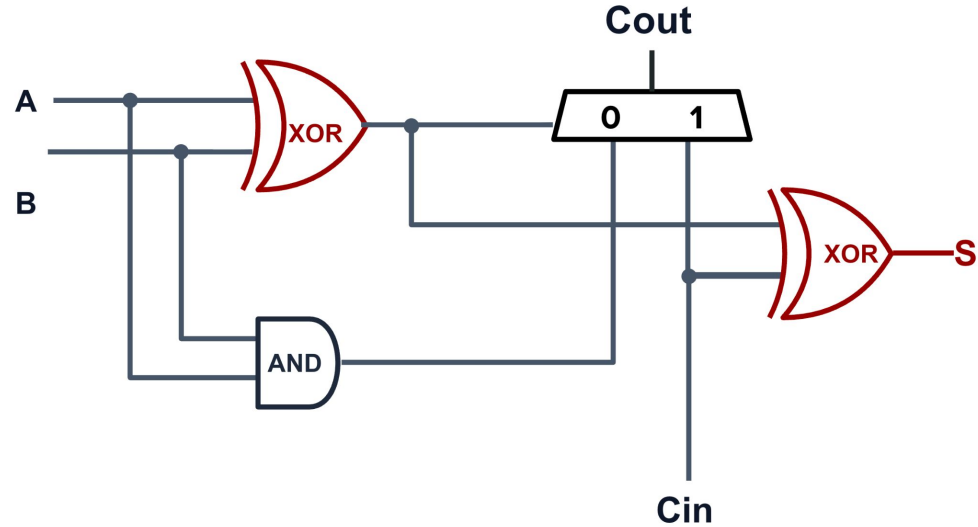
Data Path						
Delay Type	Incr (ns)	Path (ns)	Location	Cell Pin	Cell	Netlist Resources
	(r) 0.000	0.000	Site: N13	data_in[42]		data_in[42]
net (fo=0)	0.000	0.000				data_in_IBUF[42]_inst/I
			Site: N13	PAD	IBUF_INST (IBUF)	data_in_IBUF[42]_inst/IBUF_INST/PAD
IBUF (Prop IBUF HPIOB M PAD O)	(r) 0.543	0.543	Site: N13	O	IBUF_INST (IBUF)	data_in_IBUF[42]_inst/IBUF_INST/O
net (fo=1, routed)	0.000	0.543				data_in_IBUF[42]_inst/OUT
			Site: N13	I	IBUFCTRL_INST (IBUFCTRL)	data_in_IBUF[42]_inst/IBUFCTRL_INST/I
IBUFCTRL (Prop IBUFCTRL HPIOB M I O)	(r) 0.000	0.543	Site: N13	O	IBUFCTRL_INST (IBUFCTRL)	data_in_IBUF[42]_inst/IBUFCTRL_INST/O
net (fo=1, routed)	1.838	2.381				carry_or/data_in_IBUF[42]
			Site: SLI...X81Y329	I1	luts_out0_inferred_6/i_ (LUT6)	carry_or/luts_out0_inferred_6/i_ I1
LUT6 (Prop H6LUT SLICEL I1 O)	(r) 0.090	2.471	Site: SLI...X81Y329	O	luts_out0_inferred_6/i_ (LUT6)	carry_or/luts_out0_inferred_6/i_ /O
net (fo=1, routed)	0.010	2.481				carry_or/luts_out[7]
			Site: SLI...X81Y329	S[7]	u_carry8 (CARRY8)	carry_or/u_carry8/S[7]
CARRY8 (Prop CARRY8 SLICEL S[7] CO[7])	(r) 0.122	2.603	Site: SLI...X81Y329	CO[7]	u_carry8 (CARRY8)	carry_or/u_carry8/CO[7]
net (fo=1, routed)	1.718	4.321				data_out_OBUF
			Site: A10	I	data_out_OBUF_inst (OBUF)	data_out_OBUF_inst/I
OBUF (Prop OBUF HDIOB M I O)	(r) 2.381	6.702	Site: A10	O	data_out_OBUF_inst (OBUF)	data_out_OBUF_inst/O
net (fo=0)	0.000	6.702				data_out
			Site: A10	data_out		data_out

Ripple Carry Adder

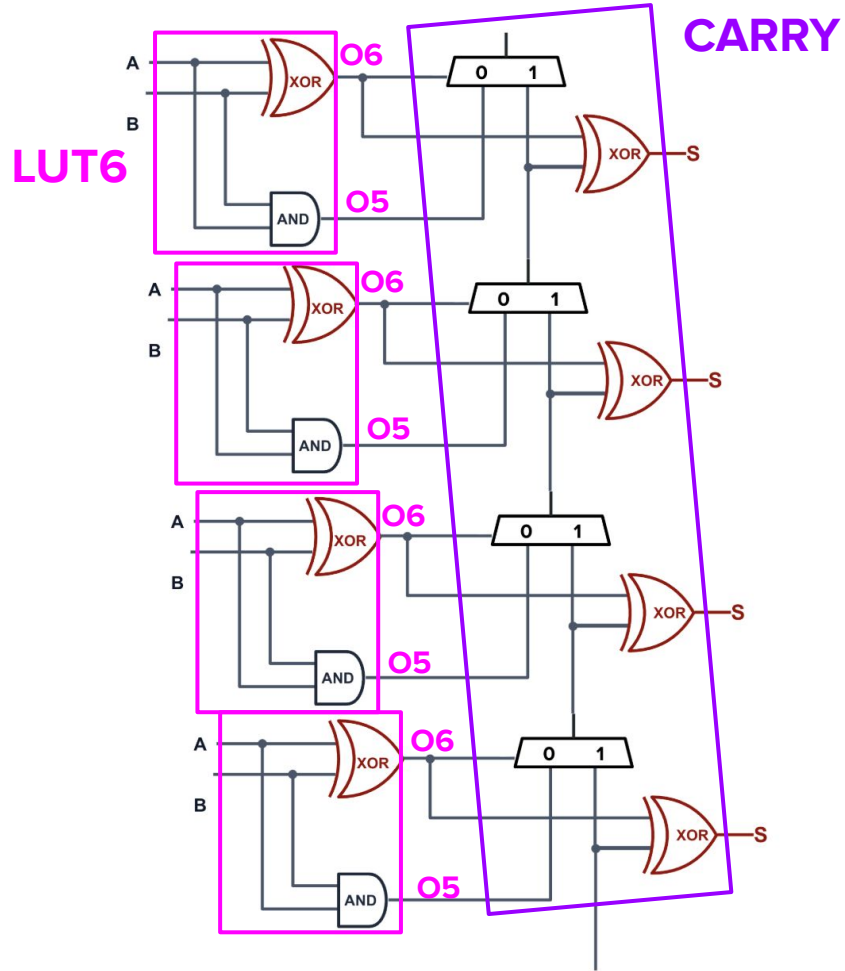


$$S = A \oplus B \oplus C_{in}$$

$$C_{out} = (A \cdot B) + (A \cdot C_{in}) + (B \cdot C_{in})$$
$$= A \cdot B + C_{in} \cdot (A \oplus B)$$



Ripple Carry Adder



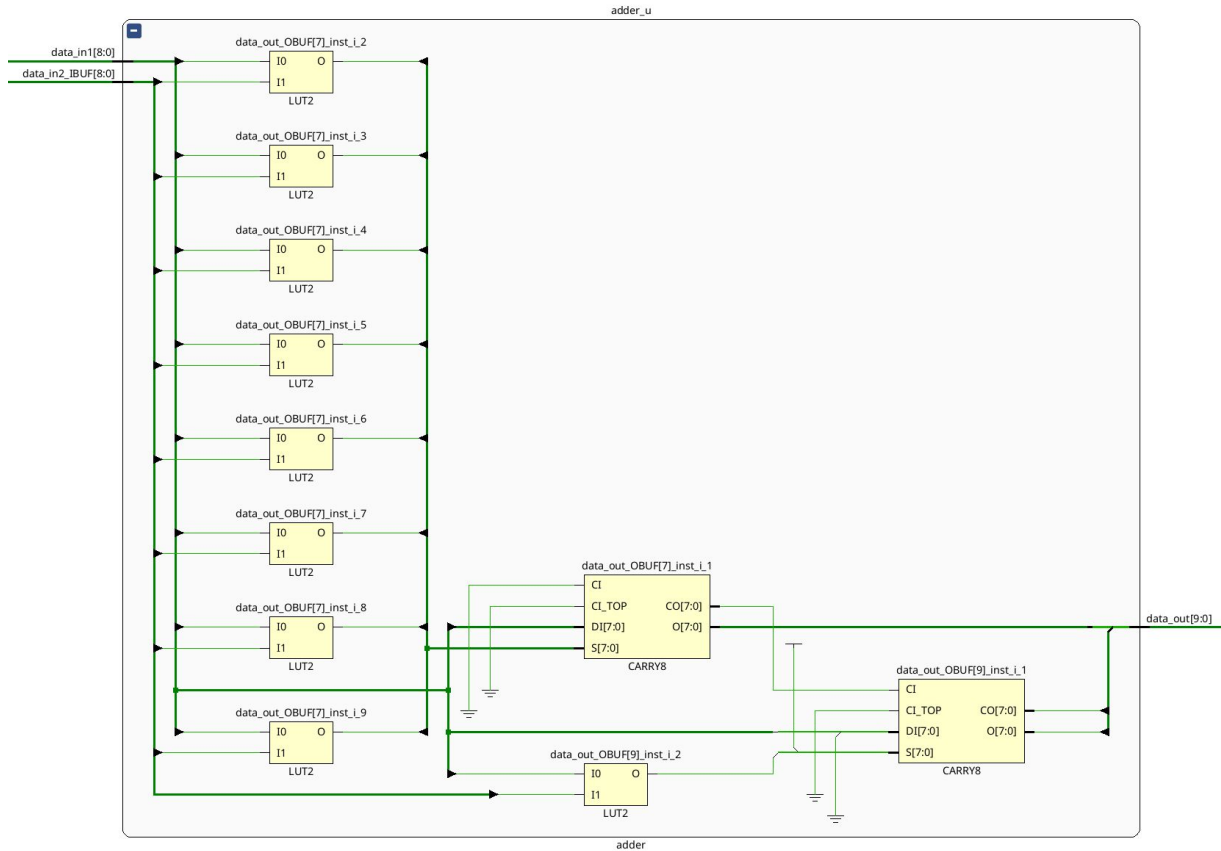
Ripple Carry Adder

SystemVerilog

```
module adder (
    input wire [8:0] data_in1,
    input wire [8:0] data_in2,
    output logic [9:0] data_out
);

assign data_out = data_in1 + data_in2;

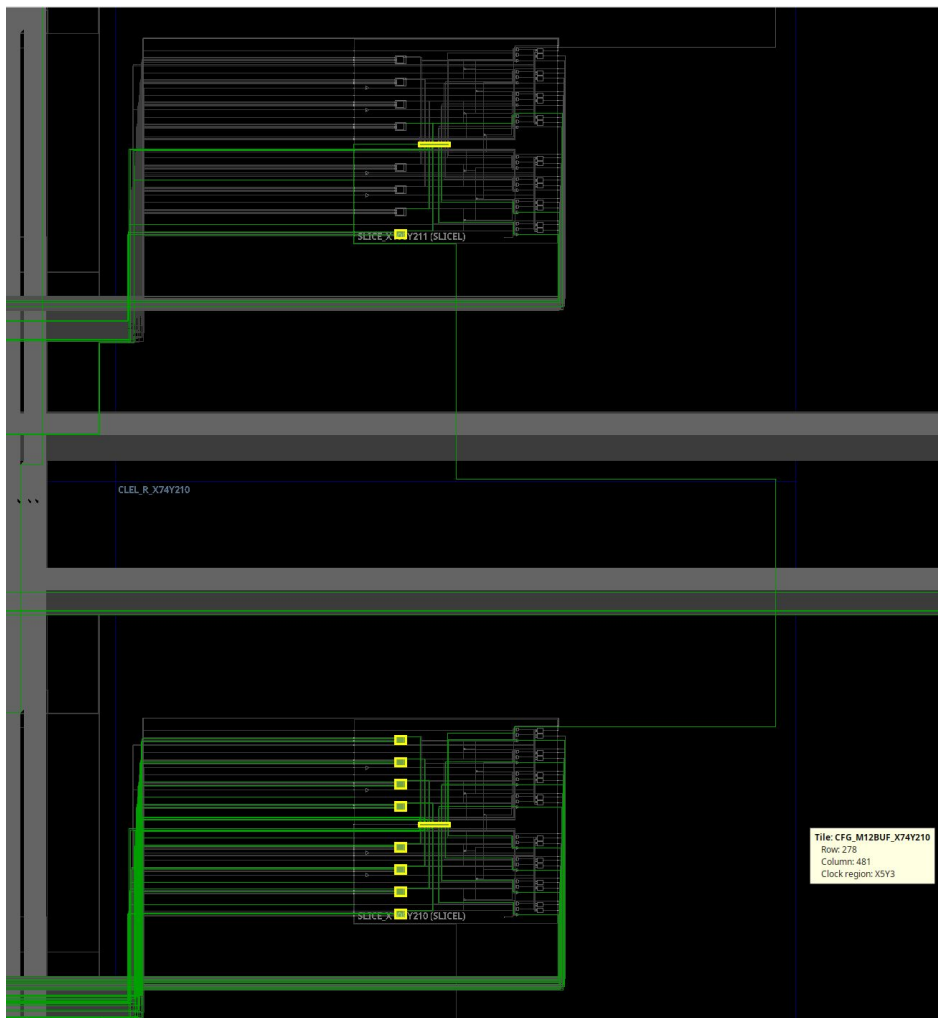
endmodule
```

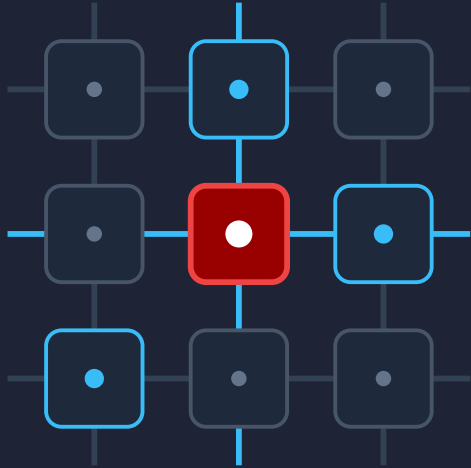


Ripple Carry Adder

SystemVerilog

```
module adder (  
    input wire [8:0] data_in1,  
    input wire [8:0] data_in2,  
    output logic [9:0] data_out  
);  
  
assign data_out = data_in1 + data_in2;  
  
endmodule
```





III. Wide Multiplexers (F7, F8, F9)

UltraScale+ Architecture

Muxes (F7, F8, F9)

Dedicated MUX Resources

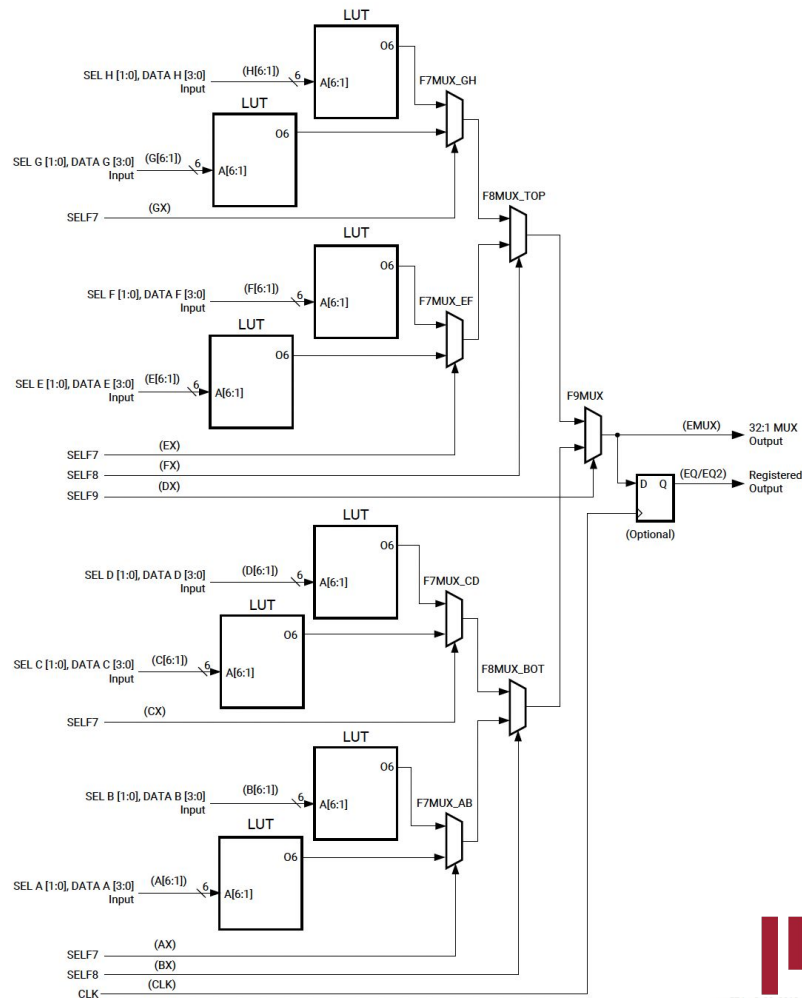
Dedicated F7, F8, and F9 multiplexers combine 6-LUT outputs within a single CLB slice to construct wide functions.

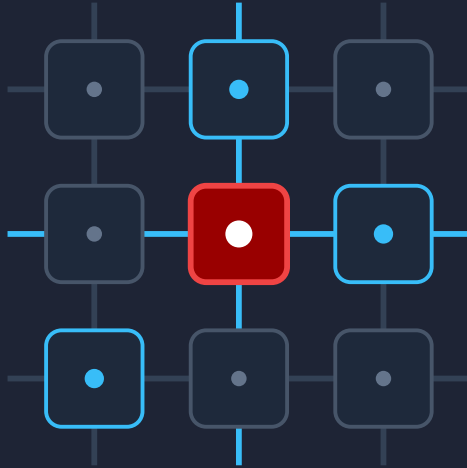
32:1 MUX Hierarchy

Cascades F7MUX (combines two 6-LUTs), F8MUX (combines two F7s), and F9MUX to form up to a 32:1 multiplexer.

⚡ High Performance

Internal dedicated multiplexing avoids general interconnect routing delays, delivering optimal speed for wider logic.





V. (SLICEM)

- **Distributed RAM**
- **Shift Registers Logic (SRL)**

UltraScale+ Architecture

Distributed RAM

Synchronous RAM Resource

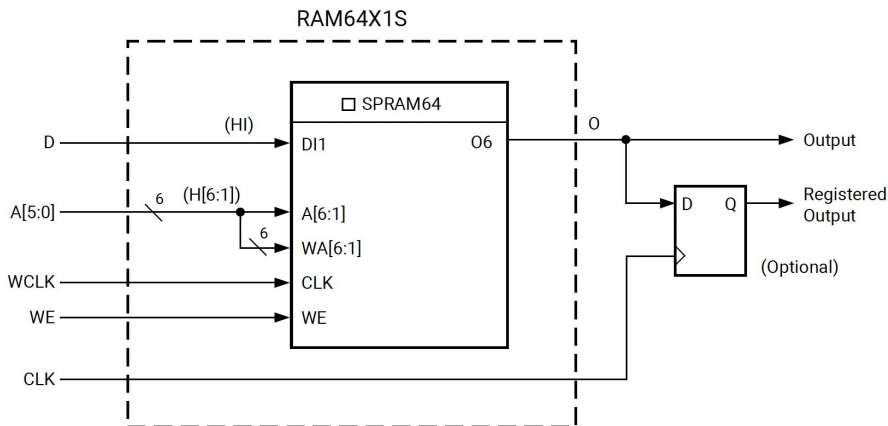
LUTs in SLICEM can be implemented directly as distributed synchronous RAM.

High Storage Density

Multiple LUTs in a SLICEM combine flexible configurations to store up to 512 bits per SLICEM.



Note: We will explore this architecture further in Session 4 following Joe talking about memory concepts.



ug574_c2_06_110512

UltraScale Architecture, Configurable Logic Block User Guide; UG574 (v1.6)



Shift Register (SRL)

32-Bit Shift Register (SRL32E)

Configures SLICEM LUTs directly as shift registers without utilizing flip-flops.

Dynamic Tap Selection

Address inputs A[4:0] dynamically select output depth (1 to 32 bits), enabling programmable delay lines.

Pipeline Delays

Efficiently balances multi-stage pipeline timing and data path alignment without consuming flip-flops.

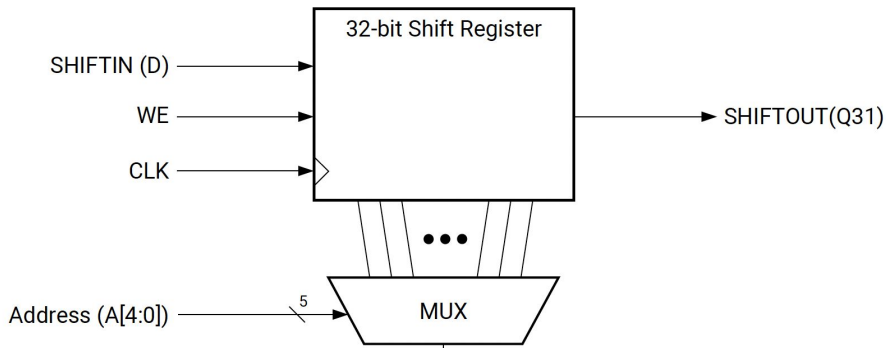
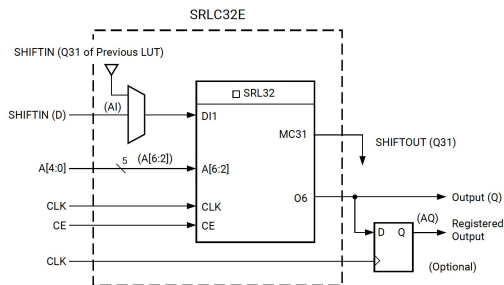


Figure 14: 32-bit Shift Register Configuration

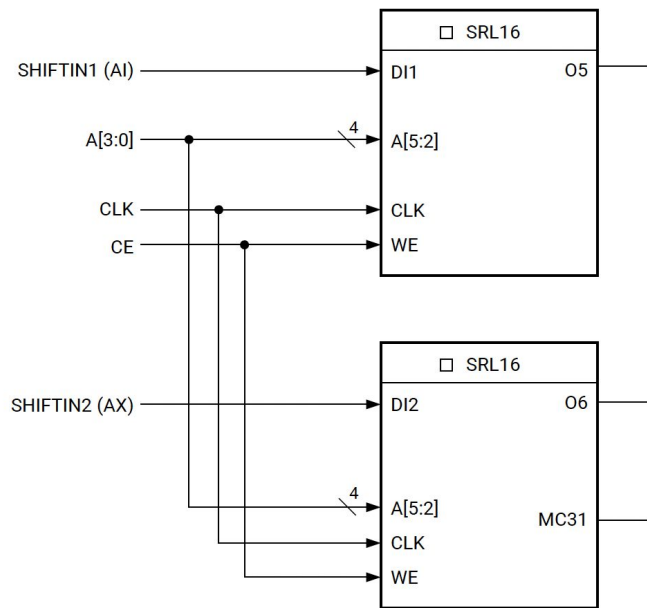


ug174_v2_07_102913



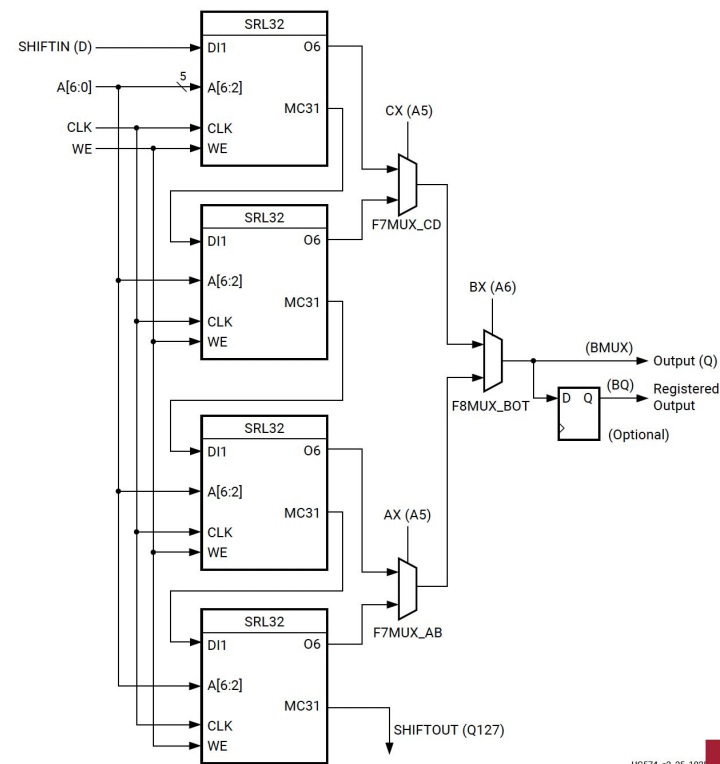
Shift Register (SRL)

Figure 16: Dual 16-bit Shift Register Configuration



ug574_c2_09_110512

Figure 17: 128-bit Shift Register Configuration



UG574_c2_25_1025



Summary

01

RTL describes hardware

Your HDL is synthesized into **physical FPGA resources and connections** — it is not software executing line-by-line.

02

LUTs implement general combinational logic

A LUT can implement **any Boolean function of up to 6 inputs**. Larger functions become networks of LUTs and wide muxes.

03

Use dedicated hardware when it exists

Carry chains and dedicated muxes provide faster paths for structures such as **arithmetic, reductions, and wide muxes** than building everything from general routing.

04

LUTs can be more than logic

In **SLICEM**, LUT resources can also implement **distributed RAM and shift registers (SRLs)** instead of combinational functions.

05

Write RTL with the hardware in mind

Different RTL can describe the same functionality but lead to different **area, timing, routing, and resource usage**. Use Vivado to inspect what synthesis actually produced.

Next Sessions

01 FFs and Clocking Network and Resources

02 Timing Constraints on the FPGA and what can go wrong

03 Solving timing violations

- Logic problems
- Route problems
- Fanout Problems
- fannouts for CE on registers

04

Clock Domain Crossing (CDC):

- All scenarios for cdc: pulse, multibit, gray encoding, fifo, with coding them in practice

Hands-on Design Project:

- example design project (measure the frequency of a clock using a reference clock)



Questions?