CS220A Lab#1
Introduction to Spartan-3E
and
Xilinx ISE

Mainak Chaudhuri
Indian Institute of Technology Kanpur

Sketch

- Brief on Xilinx Spartan-3E FPGA
- Xilinx Integrated Synthesis Environment (ISE)
  – Example implementation of a full adder
    - Verilog HDL
    - Schematic
  – Example implementation of a two-bit adder

Field-programmable gate array

- Two-dimensional array of generic logic/storage cells interconnected by programmable switches
  - Each cell can be programmed to carry out simple combinational functions
  - The interconnection switches can be programmed to decide how the result of one function is input to another function or stored in memory
  - Typically the programming bits are downloaded to the FPGA and the circuit is ready
    - Done in the field as opposed to in the fabrication facility; hence the name FPGA

Field-programmable gate array

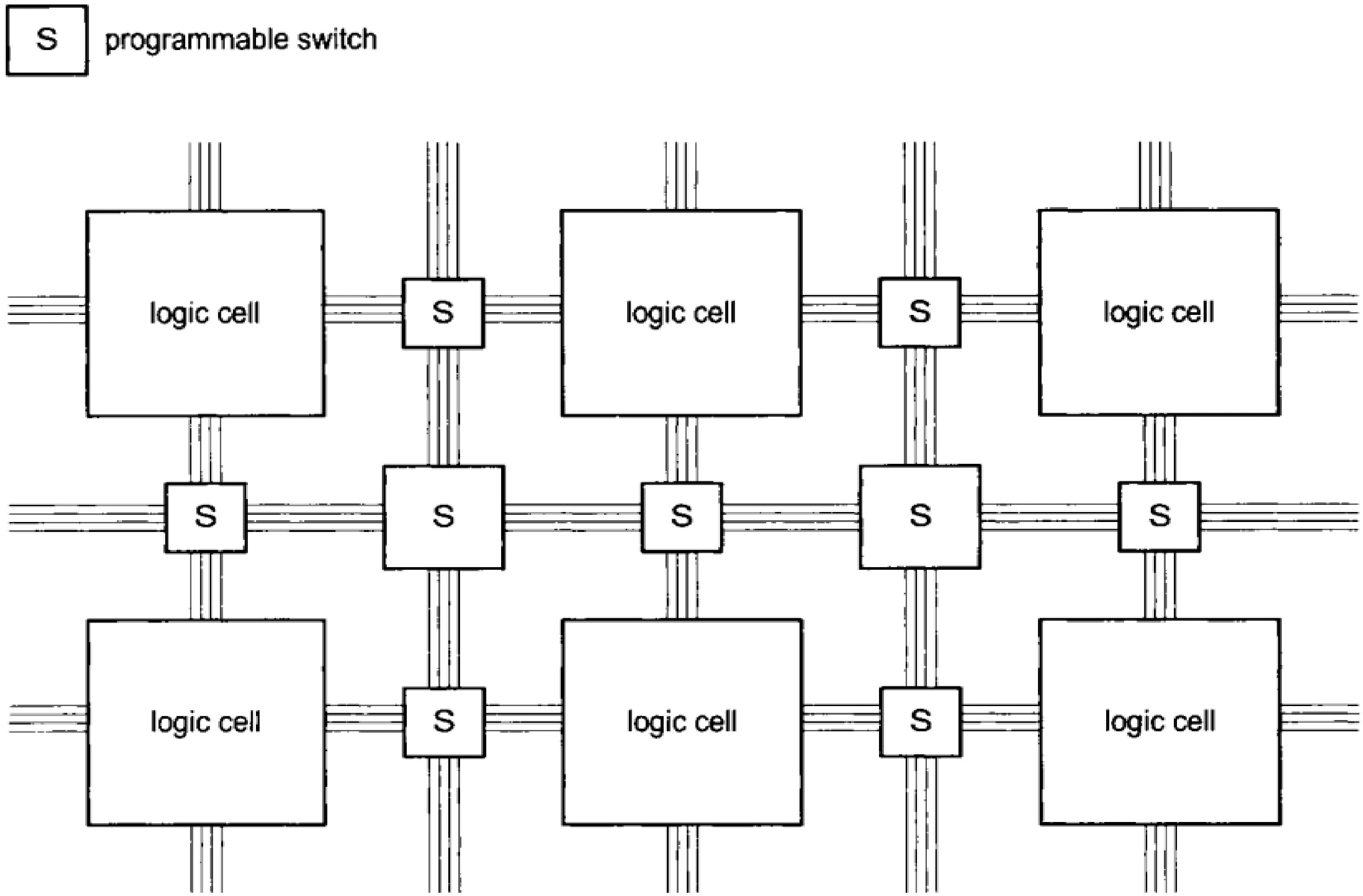


Image source: Chu. FPGA Prototyping by Verilog Examples

Xilinx Spartan 3E FPGA

- A 2D array of configurable logic blocks (CLBs)
  - Each CLB has four logic/memory slices
  - Each of the two logic slices (SLICEL) has
    - Two four-input 16-entry look-up table (LUT) function generators (can store any four-input function)
    - Two registers
    - Two multiplexors
    - Arithmetic logic and carry (two full adders)
  - Each of the two memory slices (SLICEM) has
    - Everything of a logic slice
    - Two 16-bit memory blocks (RAM16)
    - Two 16-bit shift registers (SRL16)

Xilinx Spartan 3E FPGA

- A 2D array of configurable logic blocks (CLBs)
- I/O blocks (IOB)
  - Controls I/O between logic/storage and the user interfaces (slide switches, LEDs, LCD, etc.)
- Block RAM (random access memory)
  - Each block can store 18K bits
- Multiplier blocks
  - Each multiplier operates on two 18-bit inputs
- Digital clock manager (DCM)
  - Routes clock throughout the FPGA

Xilinx Spartan 3E FPGA

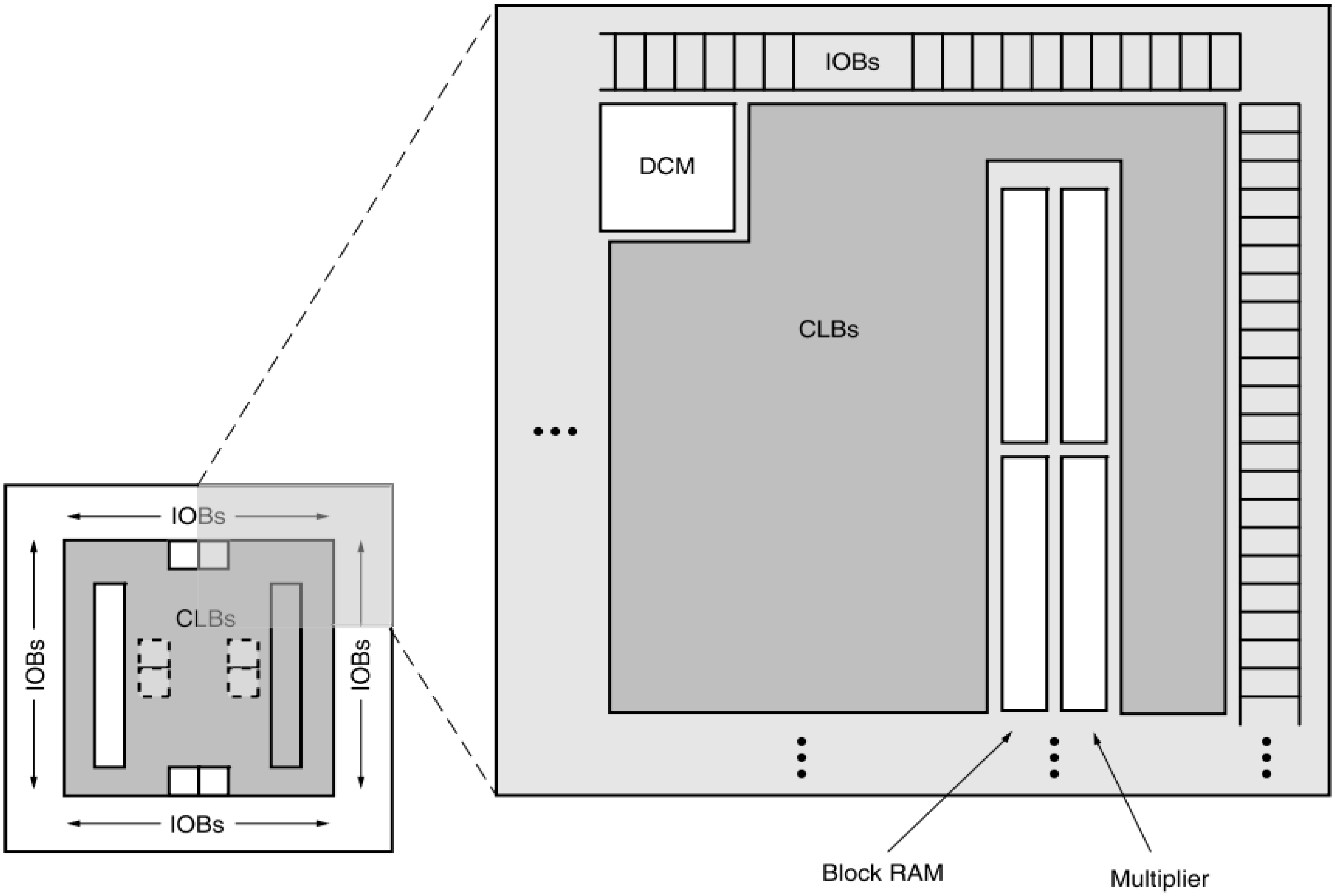


Image source: Xilinx

Xilinx Spartan 3E FPGA

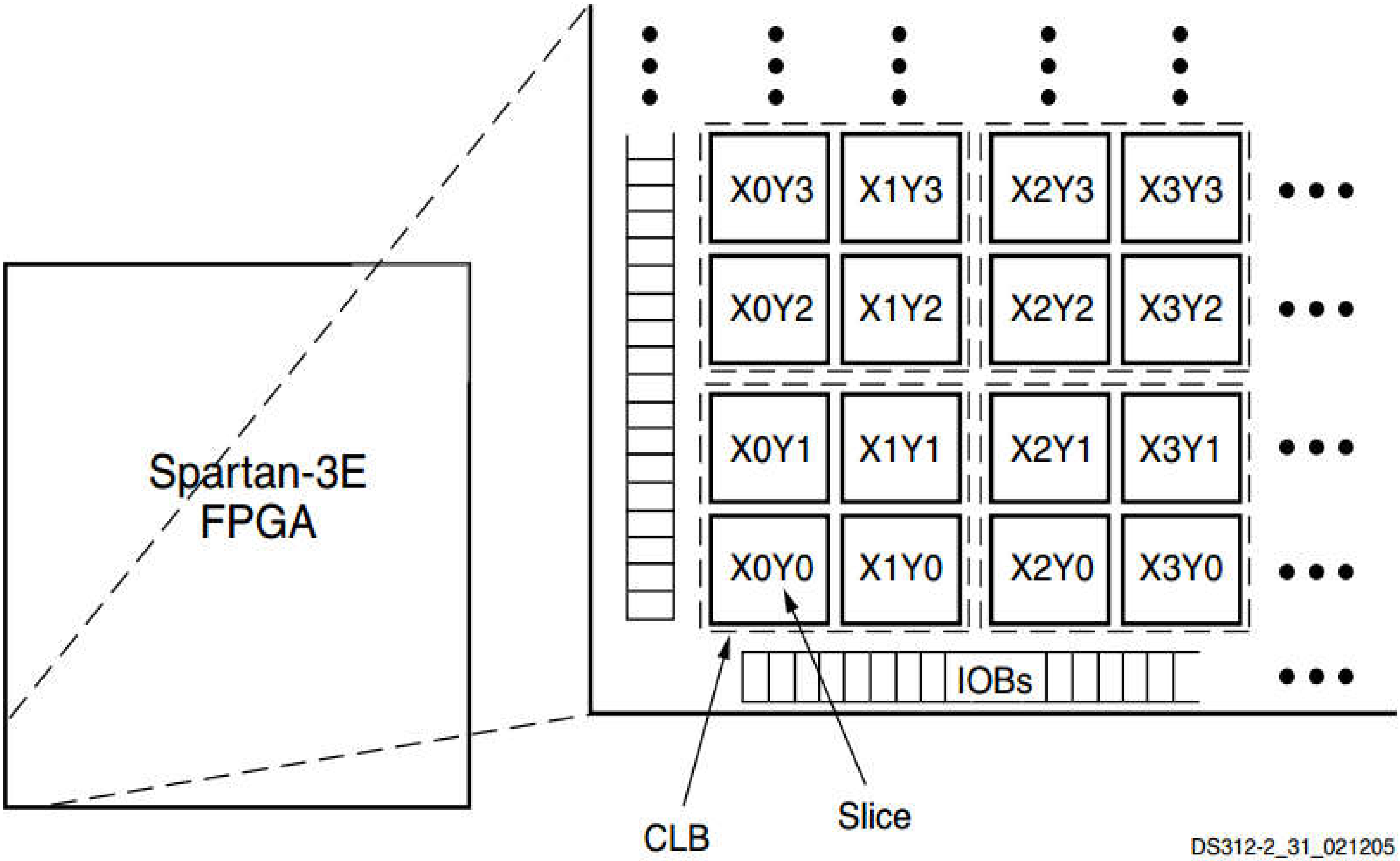


Image source: Xilinx

Xilinx Spartan 3E FPGA

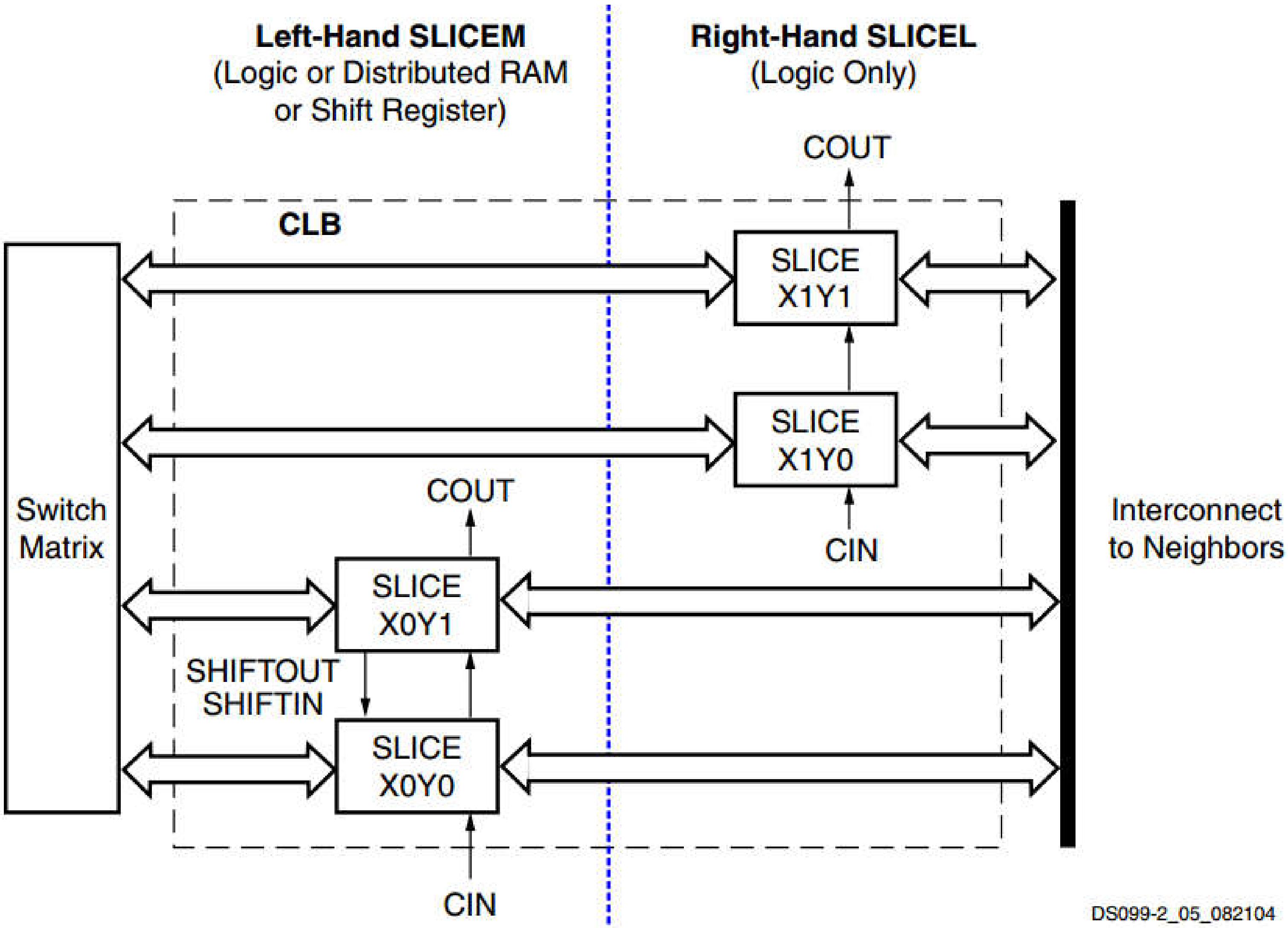


Image source: Xilinx

Xilinx Spartan 3E FPGA

- Five family members
  - 100K, 250K, 500K, 1200K, and 1600K logic gates
  - We will use XC3S500E
    - 500K gates
    - 1164 CLBs (4656 slices); 46 rows, 34 columns
    - Some CLB rows and columns are taken up by block RAM, multiplier blocks, DCM
    - 360K-bit block RAM (20 RAM blocks, each 18K bits)
    - 20 multiplier blocks
    - 4 DCMs

Xilinx ISE

- Rest of the slides go over multiple examples demonstrating how to use Xilinx ISE
  - Can simulate Verilog code
  - Can synthesize hardware on Spartan-3E FPGA
    - Synthesizing a hardware means programming the FPGA so that it models the desired hardware
    - A subset of the CLBs, RAM blocks, multipliers, and the switches will participate in implementing the specified hardware
- In this lab, you will do three syntheses
  - Full adder: Verilog to synthesis
  - Full adder: Mixed Verilog and schematic to synthesis
  - Two-bit adder: Verilog to synthesis

Xilinx ISE

- Insert your USB stick in the USB drive and accept ok when it prompts for opening the medium
- Create a new directory/folder in your USB stick
  - You may call it CS220Labs
  - For today's lab, create three directories/folders under CS220Labs
    - You may name them Lab1_1, Lab1_2, Lab1_3

Xilinx ISE

- Click on the file manager (left bottom corner) and navigate to /opt/Xilinx/14.7/ISE_DS
- Double-click run_ise.sh and click on "Execute"
  - This will launch Xilinx Project Navigator, the primary interface for using the Xilinx ISE
- Click ok on "Tip of the Day" panel
- If you receive a message telling you that a license was not found
  - The license manager will pop up automatically
  - Load the license by browsing to /opt/Xilinx/14.7/ISE_DS/common/licenses/Xilinx.lic

Xilinx ISE

- In the Xilinx Project Navigator
  - Click on File->New Project
  - In Location box, write /media/CS220Labs/Lab1_1
  - In Name box, write full_adder
  - Select HDL in top-level source type
  - Click on Next
  - For Evaluation Development Board, select Spartan-3E Starter Board
  - Leave everything else unchanged and click Next
  - Click Finish

Xilinx ISE

- In the Xilinx Project Navigator
  - Click on Project->New Source
  - Select Verilog Module from left menu
  - Write full_adder in File name box
  - Tick the Add to project box
  - Click Next
  - Leave everything blank in the next page and click on Next
  - Click Finish
  - full_adder.v should automatically open in the right pane of Xilinx Project Navigator
  - You need to fill in the module full_adder

Xilinx ISE

```
module full_adder(a, b, cin, sum, cout
  );

  input a;
  input b;
  input cin;

  output sum;
  wire sum;
  output cout;
  wire cout;

  assign sum = a^b^cin;
  assign cout = (a & b) | (b & cin) | (cin & a);

endmodule
```

Xilinx ISE

- In the Xilinx Project Navigator
  - Save your Verilog module by clicking the save icon in the menu bar above
  - Up to this point the procedure is same for simulation and synthesis
  - We will now explore how to simulate the full adder design using the Xilinx ISim simulator

Xilinx ISE: Simulation

- In the Xilinx Project Navigator
  - We need to build a top-level environment Verilog module
  - Project -> New Source
  - Select Verilog Test Fixture from the left menu
  - Write full_adder_top in File name
  - Tick Add to project box
  - Click Next
  - Click Next
  - Click Finish
  - The code for full_adder_top.v will open automatically for editing

Xilinx ISE: Simulation

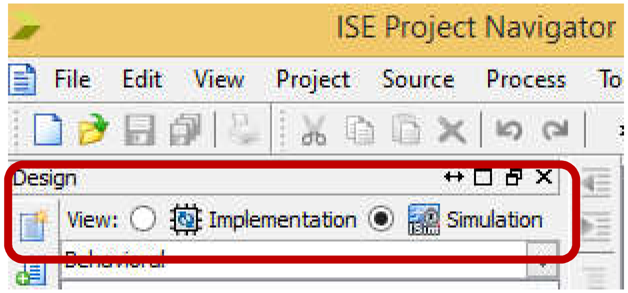
- In the Xilinx Project Navigator
  - Remove the initial block and insert the following code; leave everything else unchanged

```
always @(sum or cout) begin
   $display("time=%d: %b + %b + %b = %b, cout = %b\n", $time, a, b, cin,
     sum, cout);
 end

initial begin
   a = 0; b = 0; cin = 0;
   #5
   a = 0; b = 1; cin = 0;
   #5
   a = 1; b = 0; cin = 1;
   #5
   a = 1; b = 1; cin = 1;
   #5
   $finish;
 end
```

Xilinx ISE: Simulation

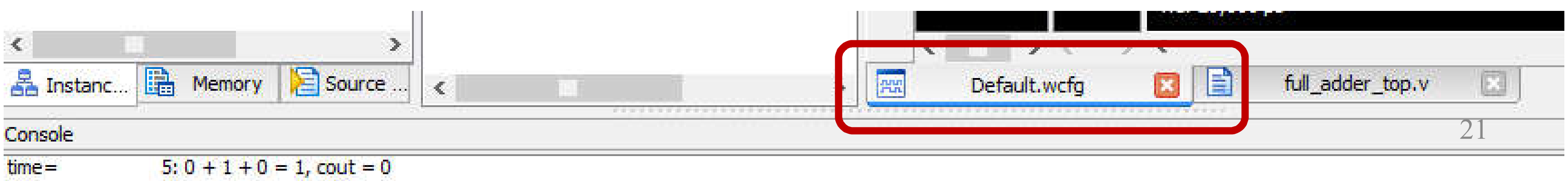
- In the Xilinx Project Navigator
  - Save the top-level module
  - On left top side, change View from Implementation to Simulation



  - Select full_adder_top in the Hierarchy pane just below the View
  - In the lower left pane, the ISim Simulator option should appear; expand that option by clicking on +

Xilinx ISE: Simulation

- In the Xilinx Project Navigator
  - Double-click Behavioral Check Syntax under ISim Simulator
  - If you see a green tick, syntax check passed
    - If not, see the errors in the dialog box at the bottom and fix them; rerun syntax check
  - Double-click Simulate Behavioral Model
  - ISim will open; check the results in the bottom box
  - Click on Default.wcfg to get a visual display of the simulation; zoom to full view



Xilinx ISE: Synthesis

- Synthesis on FPGA does not require a top-level module
  - The FPGA board provides the environment
  - The inputs are provided through switches and buttons
  - The outputs are observed through LEDs
  - The inputs and outputs are specified through a user constraints file (UCF)
- Close ISim and switch to ISE Project Navigator
  - Change View from Simulation to Implementation

Xilinx ISE: Synthesis

- We will map the three inputs on three slide switches in the FPGA board
- We will map the two outputs on two LEDs
- Need to know the FPGA pin numbers connecting to the switches and the LEDs
- Refer to Xilinx Spartan-3E user guide
  - https://www.xilinx.com/support/documentation/boards_and_kits/ug230.pdf
  - Download and save it for future use

Xilinx ISE: Synthesis

- Pages 15 and 16 of Spartan-3E user guide discuss the slide switches
  - The UCF location constraints specify the pin numbers (L13, L14, H18, N17) and IO standards of the switches
  - Note the location of the switches on your FPGA board (should be in one of the corners)

```
NET "SW<0>" LOC = "L13" | IOSTANDARD = LVTTL | PULLUP ;
NET "SW<1>" LOC = "L14" | IOSTANDARD = LVTTL | PULLUP ;
NET "SW<2>" LOC = "H18" | IOSTANDARD = LVTTL | PULLUP ;
NET "SW<3>" LOC = "N17" | IOSTANDARD = LVTTL | PULLUP ;
```

Figure 2-2: **UCF Constraints for Slide Switches**

  - We will tie “a” to L13, “b” to L14, and “cin” to H18 (SW0, SW1, SW2)

Xilinx ISE: Synthesis

- Pages 19 and 20 of Spartan-3E user guide discuss the discrete LEDs

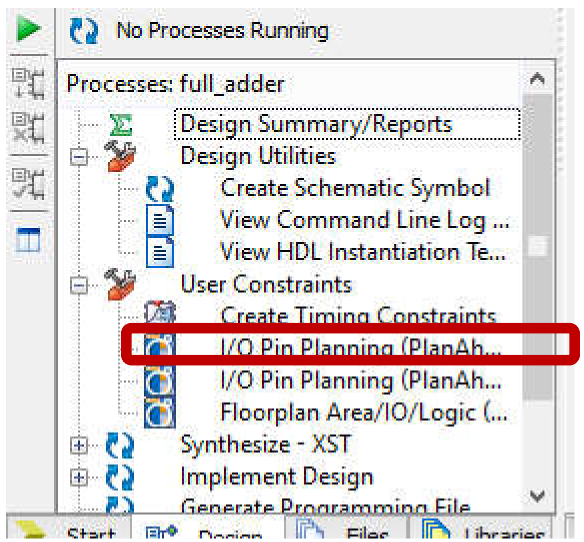
```
NET "LED<7>" LOC = "F9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<6>" LOC = "E9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<5>" LOC = "D11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<4>" LOC = "C11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<3>" LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<2>" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<1>" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<0>" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
```

Figure 2-11: **UCF Constraints for Eight Discrete LEDs**

  - We will map "sum" to F12 and "cout" to F9 (LED0 and LED7)
  - Notice location of the LEDs in your FPGA board
    - Should be above the slide switches

Xilinx ISE: Synthesis

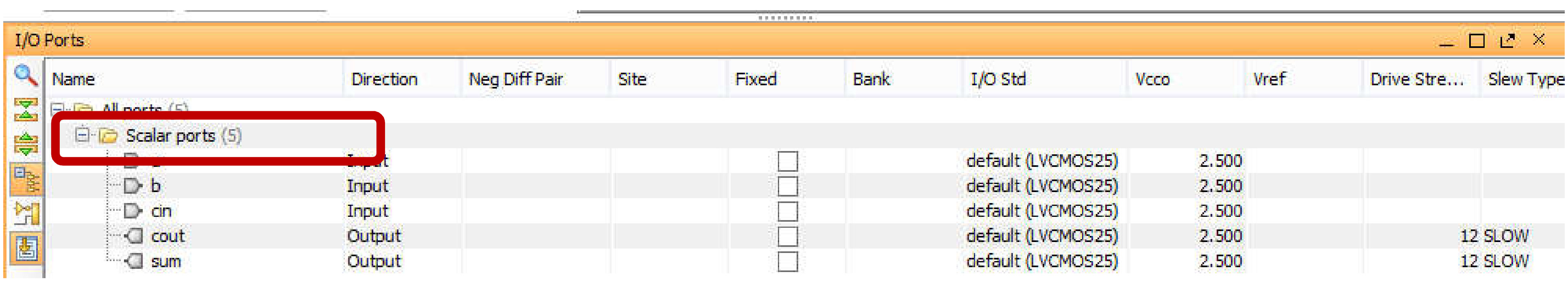
- Let's come back to the Xilinx ISE Project Navigator and prepare the UCF
  - Expand the User Constraints option in the second pane on the left by clicking +
  - Double-click I/O Pin Planning (PlanAhead) – Pre-Synthesis



  - This will launch PlanAhead (click on Yes)

Xilinx ISE: Synthesis

- Close the Welcome to PlanAhead pane by clicking Close
- In I/O Ports pane expand the Scalar ports by clicking +
  - This will list the inputs and outputs of the full_adder module



Xilinx ISE: Synthesis

- PlanAhead pin assignment
  - For each input (a, b, cin), select its row, and set Site (L13, L14, H18), check Fixed, set I/O Std to LVTTL, and Pull Type to PULLUP
  - For each output (sum, cout), select its row, and set Site (F12 and F9), check Fixed, Set I/O Std to LVTTL, and Drive Strength to 8

I/O Ports

Name	Direction	Neg Diff Pair	Site	Fixed	Bank	I/O Std	Vcco	Vref	Drive Stre...	Slew Type	Pull Type
All ports (5)											
Scalar ports (5)											
a	Input		L13	☑	1	LVTTL*	3.300				PULLUP*
b	Input		L14	☑	1	LVTTL*	3.300				PULLUP*
cin	Input		H18	☑	1	LVTTL*	3.300				PULLUP*
cout	Output		F9	☑	0	LVTTL*	3.300		8*	SLOW	NONE
sum	Output		F12	☑	0	LVTTL*	3.300		8*	SLOW	NONE

  - Save by clicking the save icon located at top left corner (below File)
  - File->Exit->Ok (this will close PlanAhead)

Xilinx ISE: Synthesis

- Return back to the ISE Project Navigator
  - Next step is to synthesize and implement the design
  - Double-click on Synthesize – XST option in the second pane on the left (below User Constraints option which you expanded)
  - Once you get a green tick indicating completion of synthesis, you can click on Design Summary (Synthesized) at the bottom of the pane on the right
    - See how many CLB slices and LUTs your design has consumed

Xilinx ISE: Synthesis

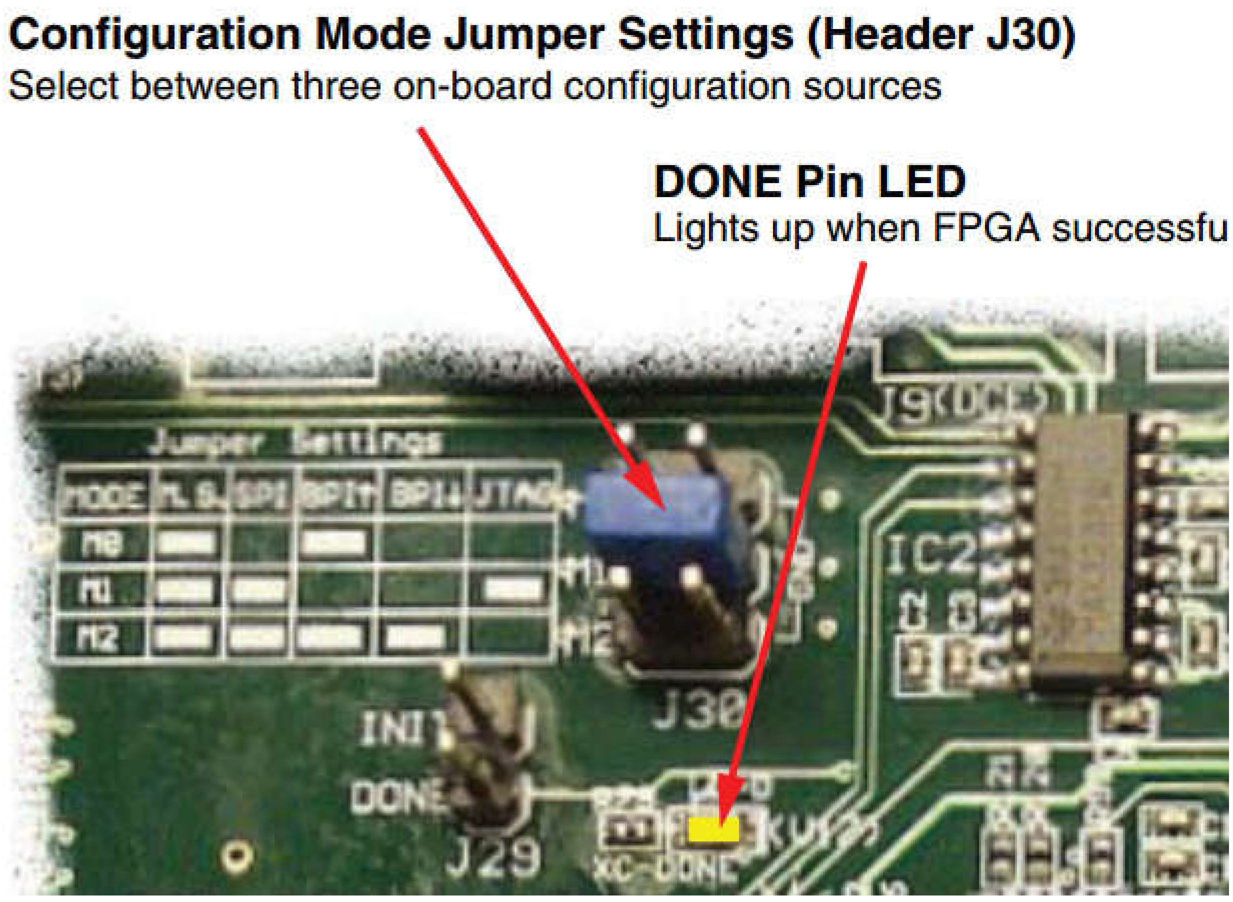
- In the ISE Project Navigator
  - Double-click Implement Design option (just below Synthesize – XST)
  - Once this step completes successfully, you can go back to the Design Summary (Implemented) tab and select Pinout Report from the left menu
    - Check that a, b, cin, sum, cout are mapped to the correct pins
  - Double-click Generate Programming File option (just below Implement Design option on the second left pane)
    - This will generate the bits needed to program the FPGA so that it can implement your design

Xilinx ISE: Synthesis

- In the ISE Project Navigator
  - Double-click Configure Target Device option (just below Generate Programming File)
    - Click ok
    - This will launch iMPACT, the Xilinx tool for programming the FPGA
  - In the FPGA board, the JTAG jumpers need to configured correctly first
    - Page 26 of the user guide shows the correct jumper configuration (you need to keep only the middle jumper connected)

Xilinx ISE: Synthesis

- JTAG jumper configuration



Xilinx ISE: Synthesis

- After doing the jumper setting, power on the FPGA board
  - The red LED beside the power switch should light up
  - Connect the USB cable between the computer and the FPGA board
    - The green LED beside the cable port on the FPGA board should light up indicating a healthy connection
- In iMPACT menu
  - Double-click on Boundary Scan
  - Right-click on the main area and select Initialize Chain; select Yes

Xilinx ISE: Synthesis

- In iMPACT
  - Automatically a file selection will open for programming the FPGA
    - Select /media/CS220Labs/Lab1_1/full_adder/full_adder.bit
    - It will ask you if you want to attach a PROM; say No
    - Select Bypass for the next one
    - Select Bypass for the next one too
  - A Programming Properties selection menu will open
    - Click on Apply and then Ok
  - You will see a chain of three devices
    - The green one is the FPGA; the other two are memory devices which we bypassed

Xilinx ISE: Synthesis

- Right-click on the green FPGA
  - Select Program
  - If the FPGA is programmed correctly it will say Program Succeeded and you will see the orange Done LED light up
- Now the FPGA is running your hardware
  - We need to give inputs and observe the outputs
  - Use the slide switches to provide values for a, b, cin
  - Check if the correct LEDs glow
- This concludes the first assignment of Lab1

Notes for healthy FPGA

- Always power on the FPGA after doing the jumper setting
  - Never change jumper setting with the FPGA powered on
- Always connect the USB cable after powering on the FPGA
- Always power off the FPGA before disconnecting the USB cable
  - Do not pull out the USB cable when the FPGA is powered on
- Keep all jumpers connected to the board when you are not using the board

Xilinx ISE: Synthesis

- Save the iMPACT project
  - /media/CS220Labs/Lab1_1/full_adder/full_adder.ipf
- File->Exit
  - Agree to save when it asks
  - This will close iMPACT
- We will now start the second assignment

Xilinx ISE: Assignment#2

- We will learn how to create schematics of digital design in Xilinx
- Start a new project in Project Navigator
  - File -> New Project
  - Location: /media/CS220Labs/Lab1_2
  - Name: full_adder_schematic
  - Top-level Source Type: Schematic
  - Click Next
  - Select Evaluation Board
  - Click Next
  - Click Finish

Xilinx ISE: Assignment#2

- Idea of the assignment
  - We will create a Verilog module to define a two input xor gate
  - We will use this xor gate to create a schematic of the full-adder
  - Next we will simulate this by writing a Verilog Test Fixture
  - We will also synthesize it on FPGA

Xilinx ISE: Assignment#2

- In Xilinx ISE Project Navigator
  - Project -> New Source -> Verilog Module
    - Name: myxor
    - Next, Next, Finish

```
module myxor(x, y, z);
   input x;
   input y;
   output z;
   wire z;

    assign z = (x & ~y) | (~x & y);
endmodule
```

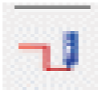
Xilinx ISE: Assignment#2

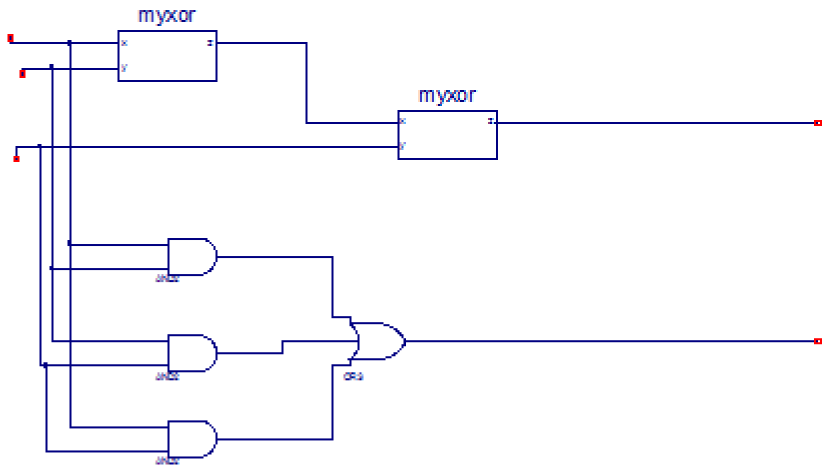
- In Xilinx ISE Project Navigator
  - To be able to use myxor in a schematic, we need to create a symbol for it
  - Select myxor (myxor.v) in left upper pane
  - Expand Design Utilities in left lower pane and double-click Create Schematic Symbol
  - Project -> New Source -> Schematic
    - File name: fuller_adder_sch
    - Click Next, Finish
  - The schematic drawing board will open now
  - You can add components by selecting the Symbols tab at the bottom of the left pane

Xilinx ISE: Assignment#2

- In Xilinx ISE Project Navigator
  - Select General category and title from symbols
    - Place the title box at the left bottom corner of the drawing board
    - Right-click on the title box and select Object properties
    - Put Full adder as the NameFieldText
    - Click Apply and ok
  - Select your project path in category and myxor from symbol
    - Place two myxor symbols on the drawing board
  - Select Logic category
    - From symbol, select three and2 and one or3 gates

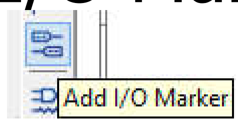
Xilinx ISE: Assignment#2

- In Xilinx ISE Project Navigator
  - Connect the gates to build the full adder
    - Use the wiring icon to draw wires 



  - Use the I/O Marker icon to name the inputs and outputs
  - Right-click on a marker, select Object Properties

Xilinx ISE: Assignment#2

- In Xilinx ISE Project Navigator
  - Use the I/O Marker icon to name the inputs and outputs 

  - Place a I/O Marker on a terminal by selecting the I/O Marker icon and then clicking on the terminal
  - Right-click on a placed marker of a terminal, select Object Properties, change Name under Nets
  - Click Apply and then Ok
  - Name the inputs a, b, cin
  - Name the outputs sum, cout
  - Save the schematic by clicking the save icon

Xilinx ISE: Assignment#2

- In Xilinx ISE Project Navigator
  - Next we will simulate the design and then synthesize
  - This procedure is same as the previous assignment starting from slide 18
  - Follow the steps from there

Xilinx ISE: Assignment#3

- Assignment#3
  - Design a two-bit adder with cin of the least significant bit adder assumed to be zero
    - Total number of inputs is four: x[0], x[1], y[0], y[1]
    - Three outputs: z[0], z[1], and carry from the most significant bit adder
    - z = x + y
    - Use SW0 and SW1 to feed x[0] and x[1]
    - Use SW2 and SW3 to feed y[0] and y[1]
    - Observe z[1] and z[0] in LED1 and LED0
    - Observe carry in LED2
  - Use the folder /media/CS220Labs/Lab1_3

Xilinx ISE: Assignment#3

- Two possible ways to implement a two-bit adder in Xilinx ISE
  - It is your choice which one you select
  - In both cases, first write a Verilog module for a full adder
  - Choice-I: write a Verilog module that instantiates two full adder modules and connects them appropriately to design the two-bit adder
  - Choice-II: draw a schematic for a two-bit adder using a half-adder and a full adder module symbols and wires
  - In both cases, write a top-level Verilog Test Fixture for simulation

Xilinx ISE: Assignment#3

- Verilog module for two-bit adder

```
module two_bit_adder (x, y, z, carry);
   input [1:0] x;
   input [1:0] y;
   output [1:0] z;
   wire [1:0] z;
   output carry;
   wire carry;
   wire carry0;
   full_adder FA0 (x[0], y[0], 1’b0, z[0], carry0);
   full_adder FA1 (x[1], y[1], carry0, z[1], carry);
endmodule
```

Xilinx ISE: Assignment#3

- Schematic requires a half adder and a full adder
  - A half adder takes just two inputs a and b
  - A half adder produces two outputs sum and cout
  - A half adder assumes cin to be zero
  - The cout of the half adder should be connected to cin of the full adder in the schematic