Architectural level design of an FPGA

Sumit Kumar Yadav
15117068

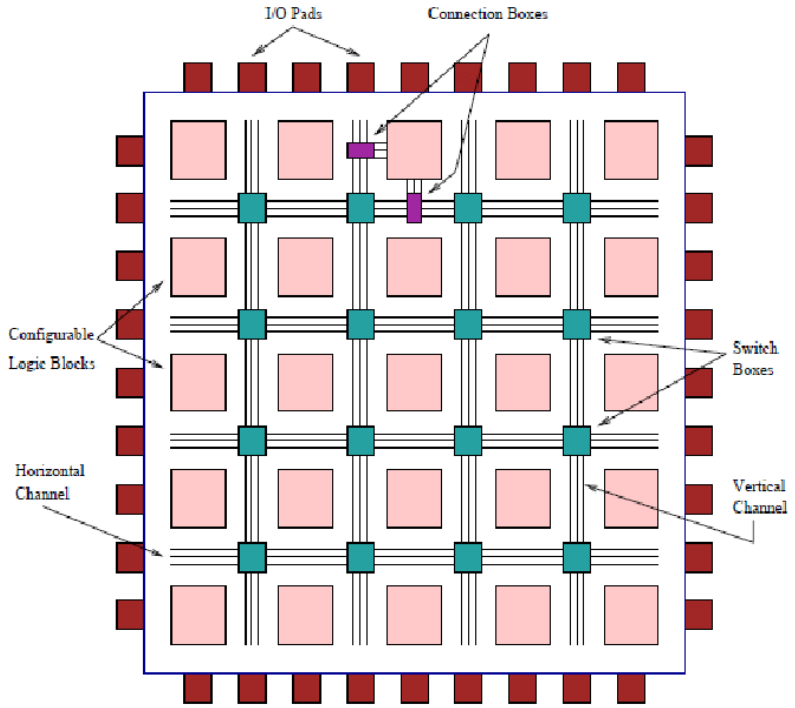
Under the guidance of
Dr. Anand Bulusu
(Associate Professor, ECE Dept., IIT Roorkee)

Contents

- Objective
- Background
- Designing the CLB
- Designing the Switch Matrix
- Designing the I/O Block
- Programmability of the FPGA
- Results and Discussion

Objective

To architect a minimal Island-Style FPGA in SCL-180 nm technology node and test its functionality at schematic level using Cadence Virtuoso.

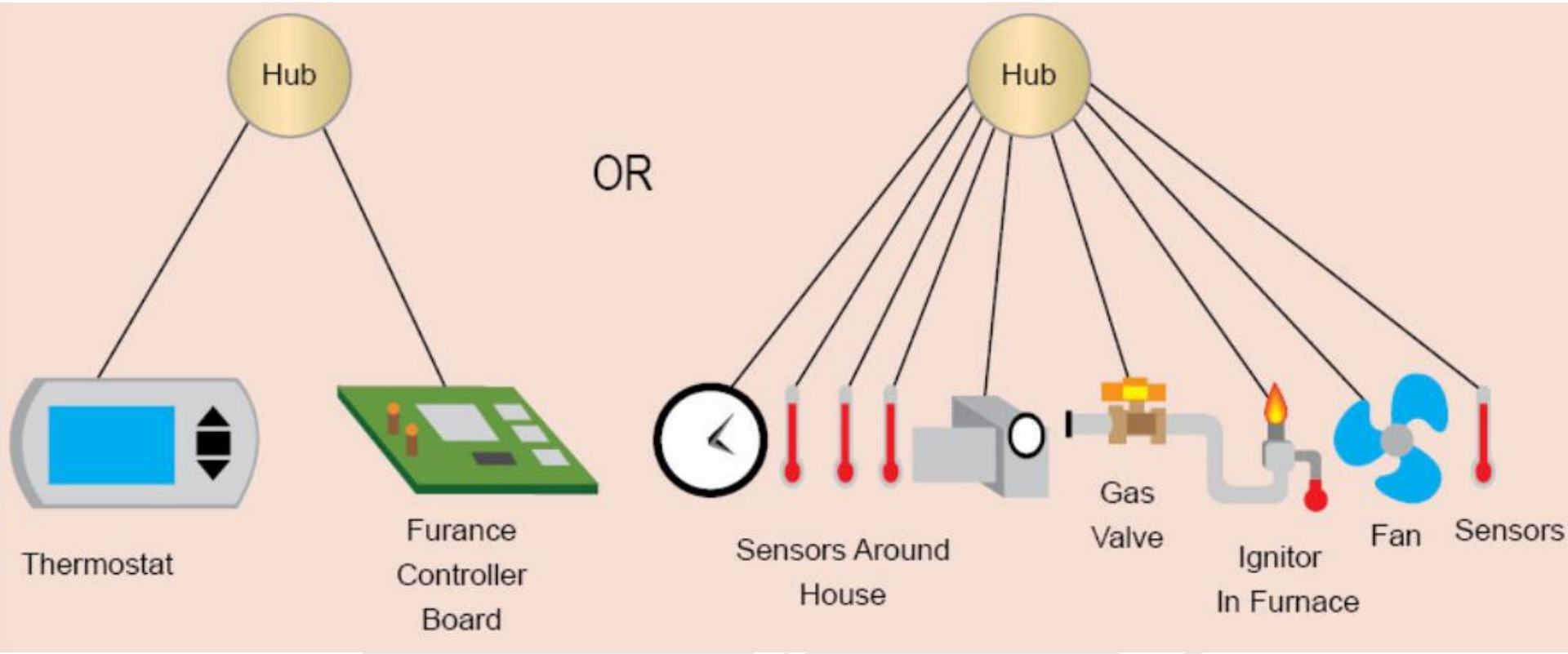


Background: FPGAs

1. Why FPGAs ?
   - Digital ASIC prototyping
   - Acceleration
   - Minimal interfacing logic
   - Minimal smart home automation hubs

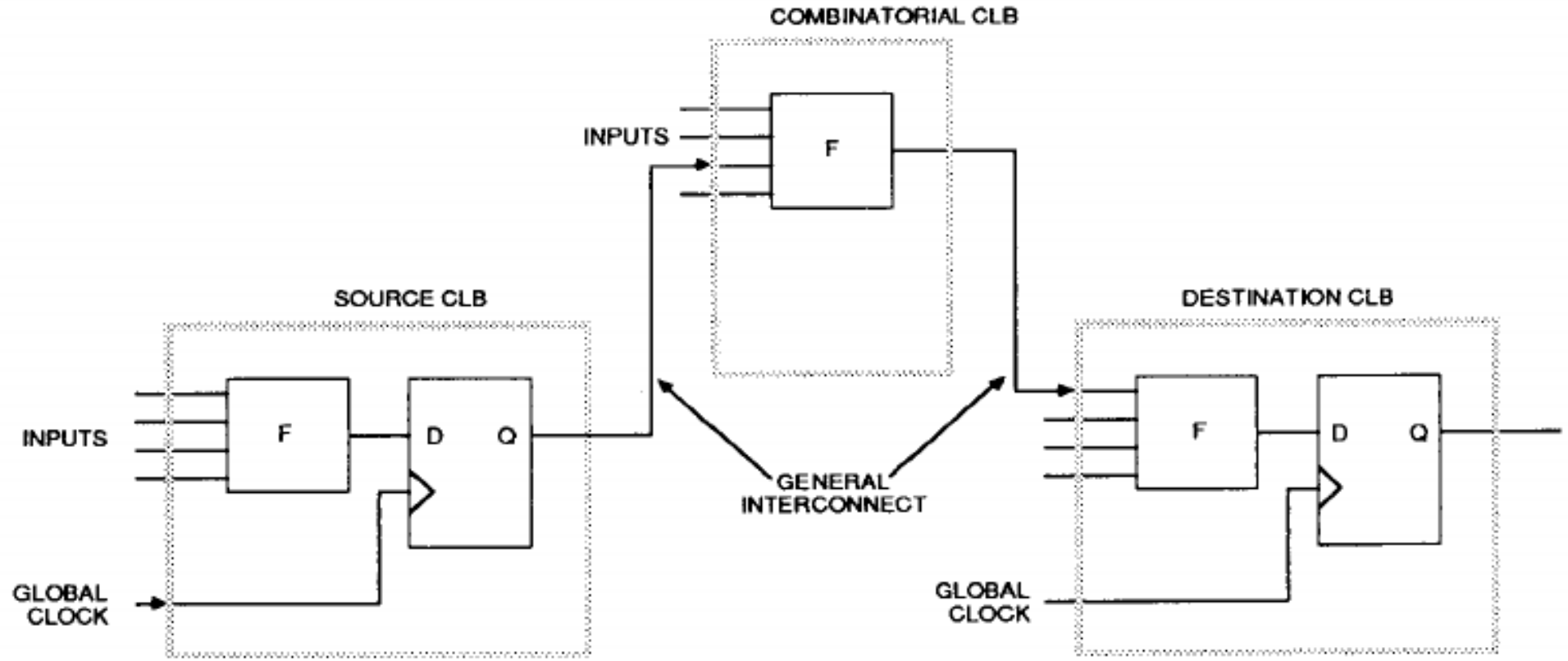
2. Working Basics
   - Any Boolean function with an n-bit input can be completely characterized by a 2^n - bit memory containing the function's truth table.

Background: FPGAs



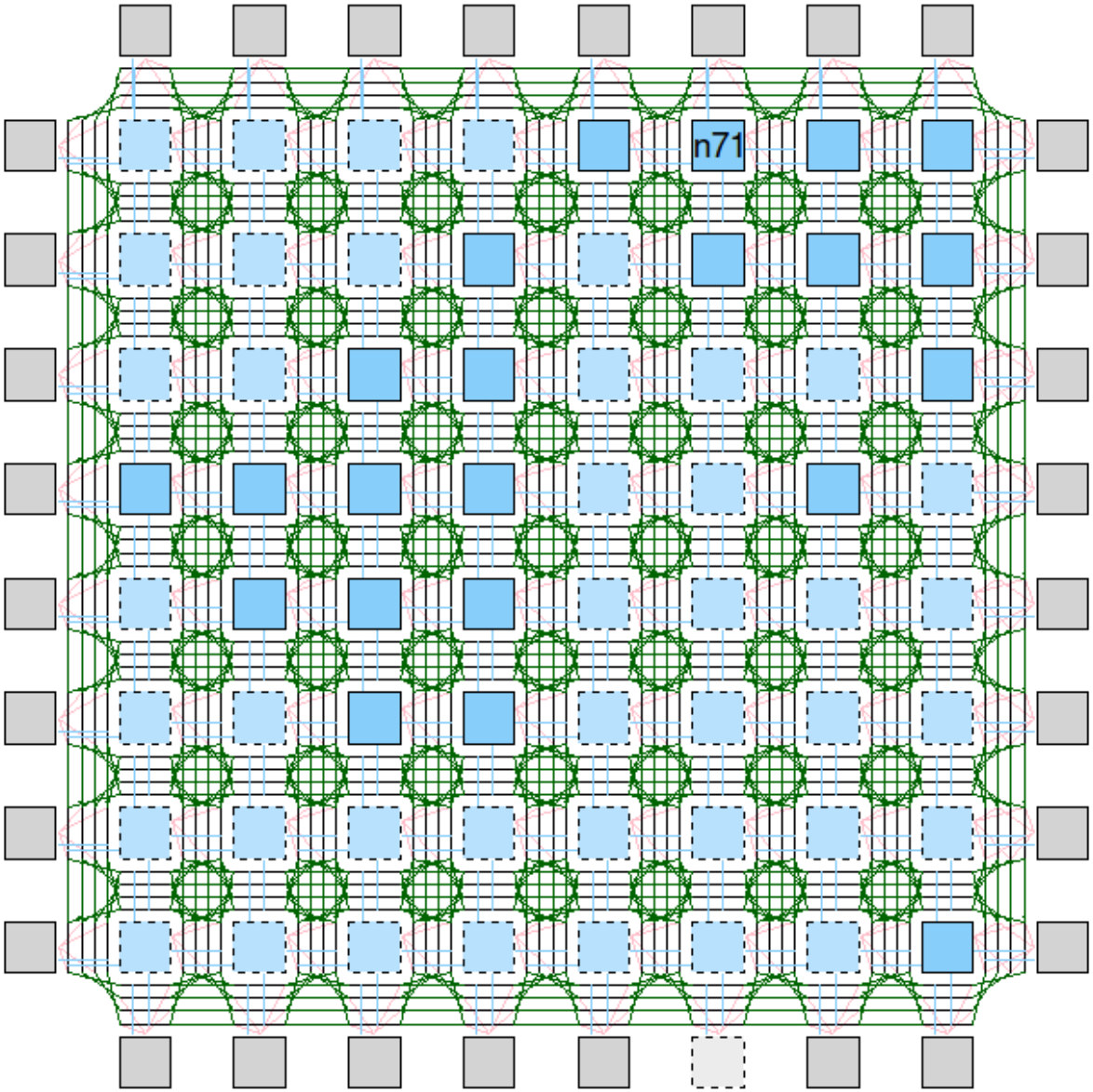
Minimal automation hubs

Background: FPGAs

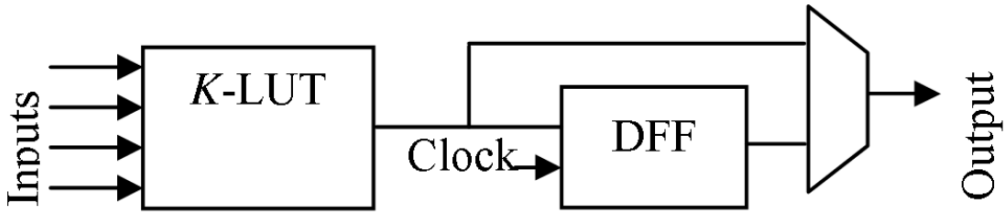


Typical Logic Path

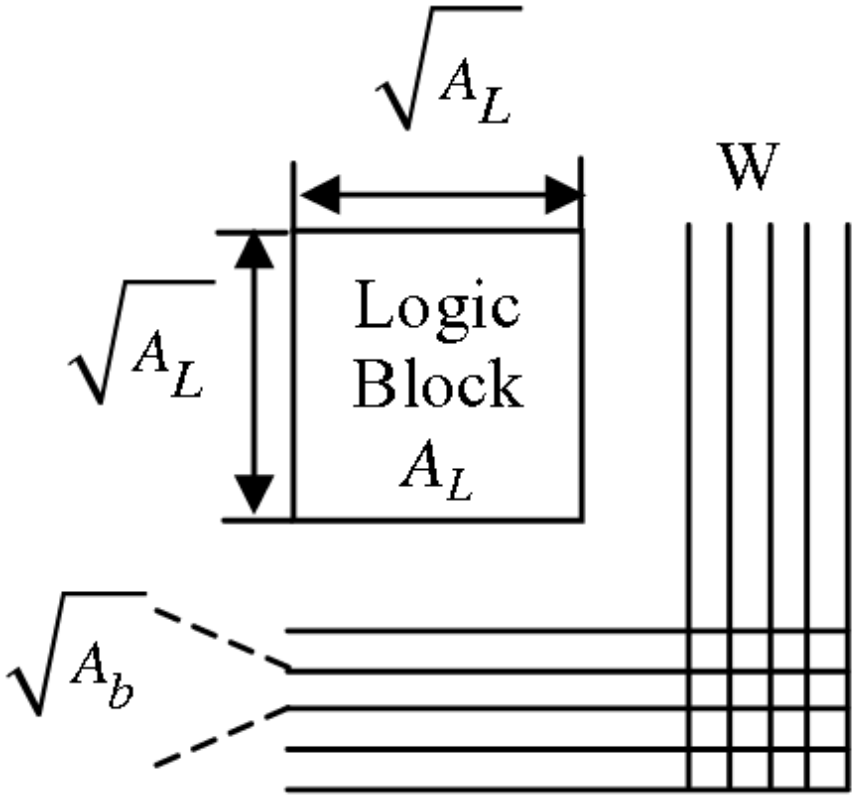
Background: Island Style FPGAs



Designing the CLB: Area model



- Basic CLB wireframe
  - K-LUT
  - D Flip Flop
  - Crossbars!



- Repetitive tile

$$A_{Tile}^{K} = A_{L}^{K} + A_{R}^{K}$$

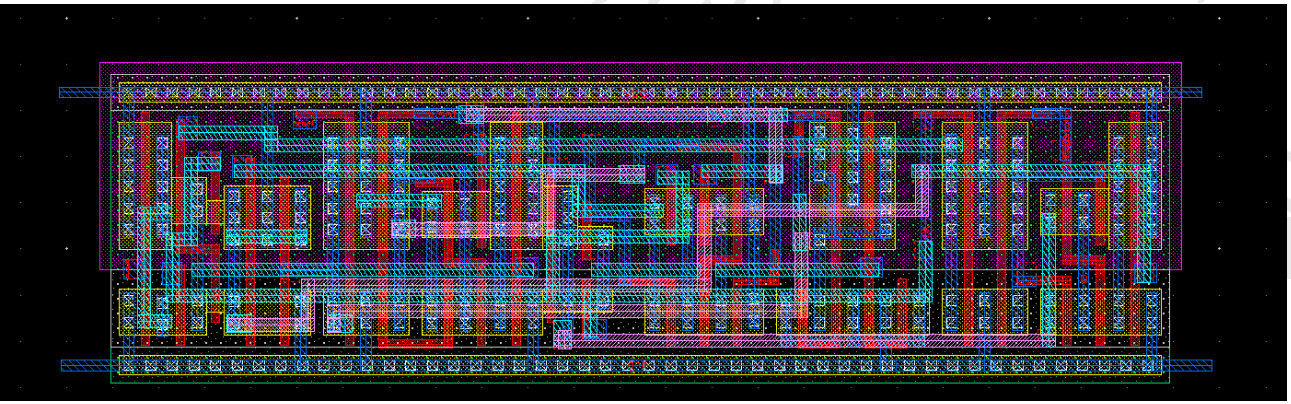
Designing the CLB: Area model

$$A_L^K = A_{bit} \cdot 2^K + A_{FF} + \sqrt{A_{bit}} \cdot (\sqrt{A_{bit} \cdot 2^K} \cdot \log_2 K + \sqrt{A_{bit} \cdot 2^K} \cdot \log_2 W_K)$$

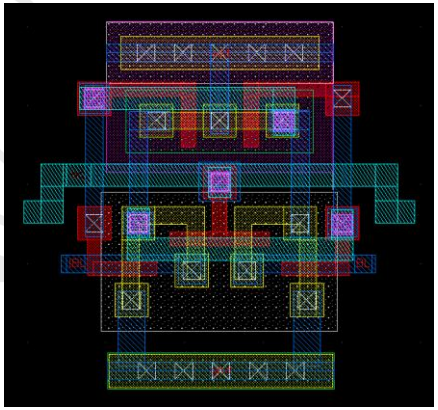
$$A_R^K = A_{bit} \cdot W_K^2 + 2 \cdot \sqrt{A_{bit}} \cdot W_K \cdot \sqrt{A_L^K}$$

$$A_{Tile} = (\sqrt{A_{bit}} \cdot W_K + \sqrt{A_L^K})^2 + A_{bit} \cdot 2^{K/2} \cdot \log_2 (K W_K)$$

$$A_{Total} = A_{Tile} \cdot N_k + 2 \cdot \sqrt{N_k} \cdot (A_{bit} \cdot W_K^2 + \sqrt{A_{bit} \cdot A_L^K} \cdot W_K)$$



A_{FF} = 168.27 um^2



A_{bit} = 16 um^2

Designing the CLB: Area model

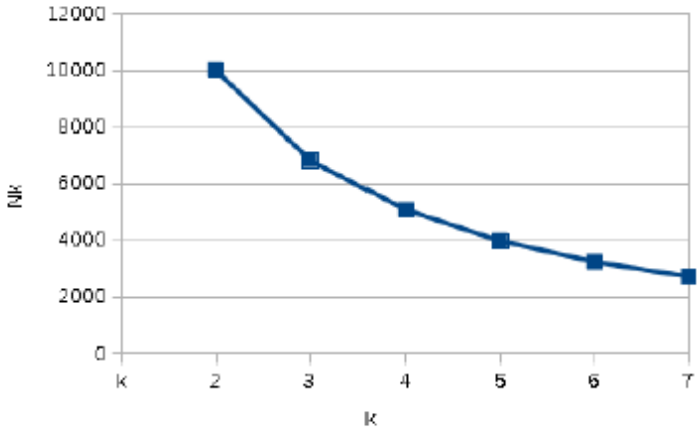
- Estimates for N_k and W_k

$$\frac{N_2}{N_k} = \left(\frac{K+1}{3}\right)^{\frac{1}{p}}$$

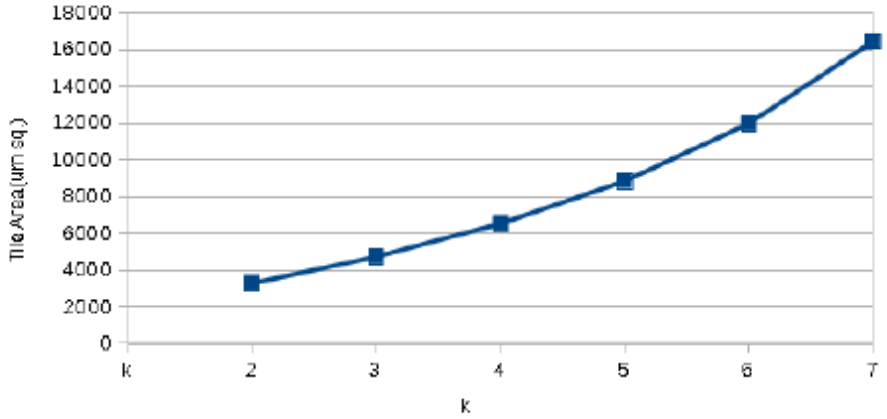
$$W_K = \frac{K+1}{2} \cdot \frac{2}{3} \cdot \frac{1-p}{p-0.5} \cdot \left((N_k)^{p-0.5} - 1\right)$$

  - N_2 = 10000 (something large [7])
  - p = 0.75 (empirical value for FPGAs [3])

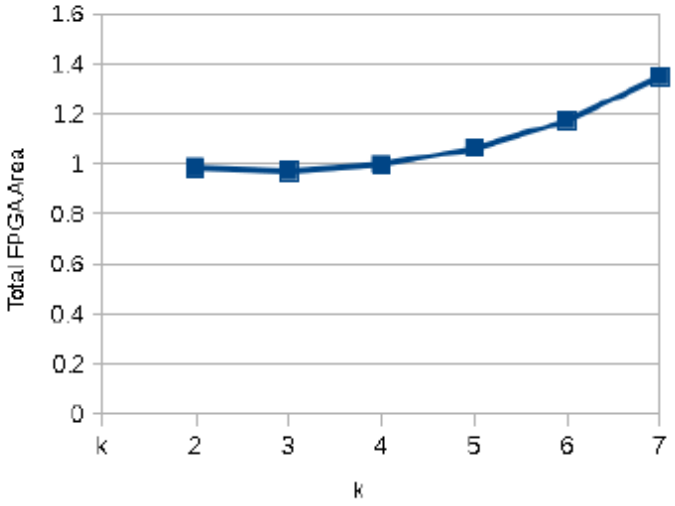
Designing the CLB: Area model



(a) Nk vs k



(b) Tile Area vs k



(c) Total Area vs k

k = 3 v/s k = 4
Really close !

Designing the CLB: Delay model

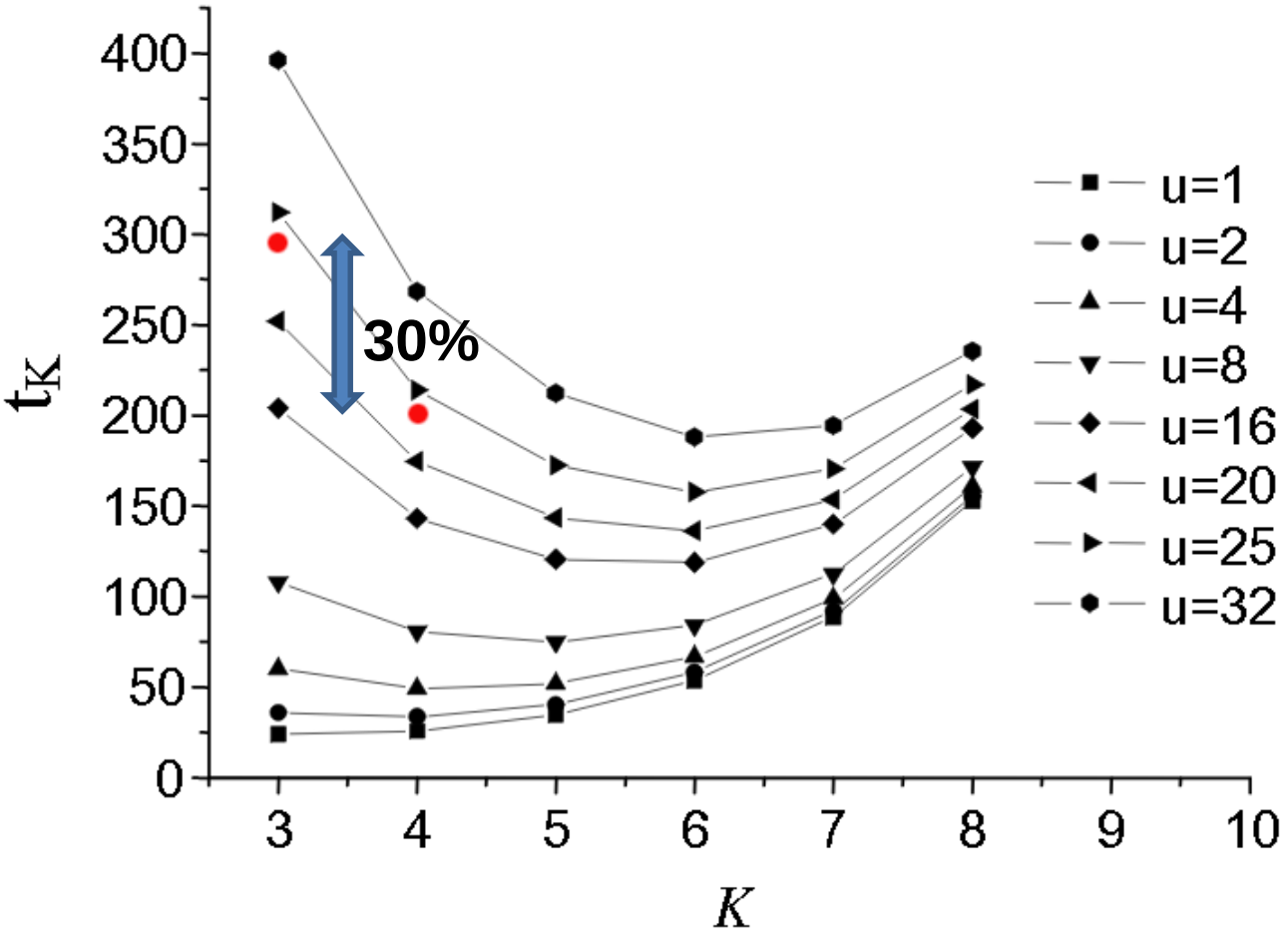
Implementing N-input logic using K-input LUTs

- Tree structure

$$M_K = 2^{N-K} \cdot (1 + \frac{1}{K} + \frac{1}{K^2} + \ldots) \approx 2^{N-K} \cdot \frac{1}{1 - 1/K}$$ (How ?)

$$D_K = \frac{N-K}{x(K)} + 1$$ where: $K = x + 2^{x-1}$

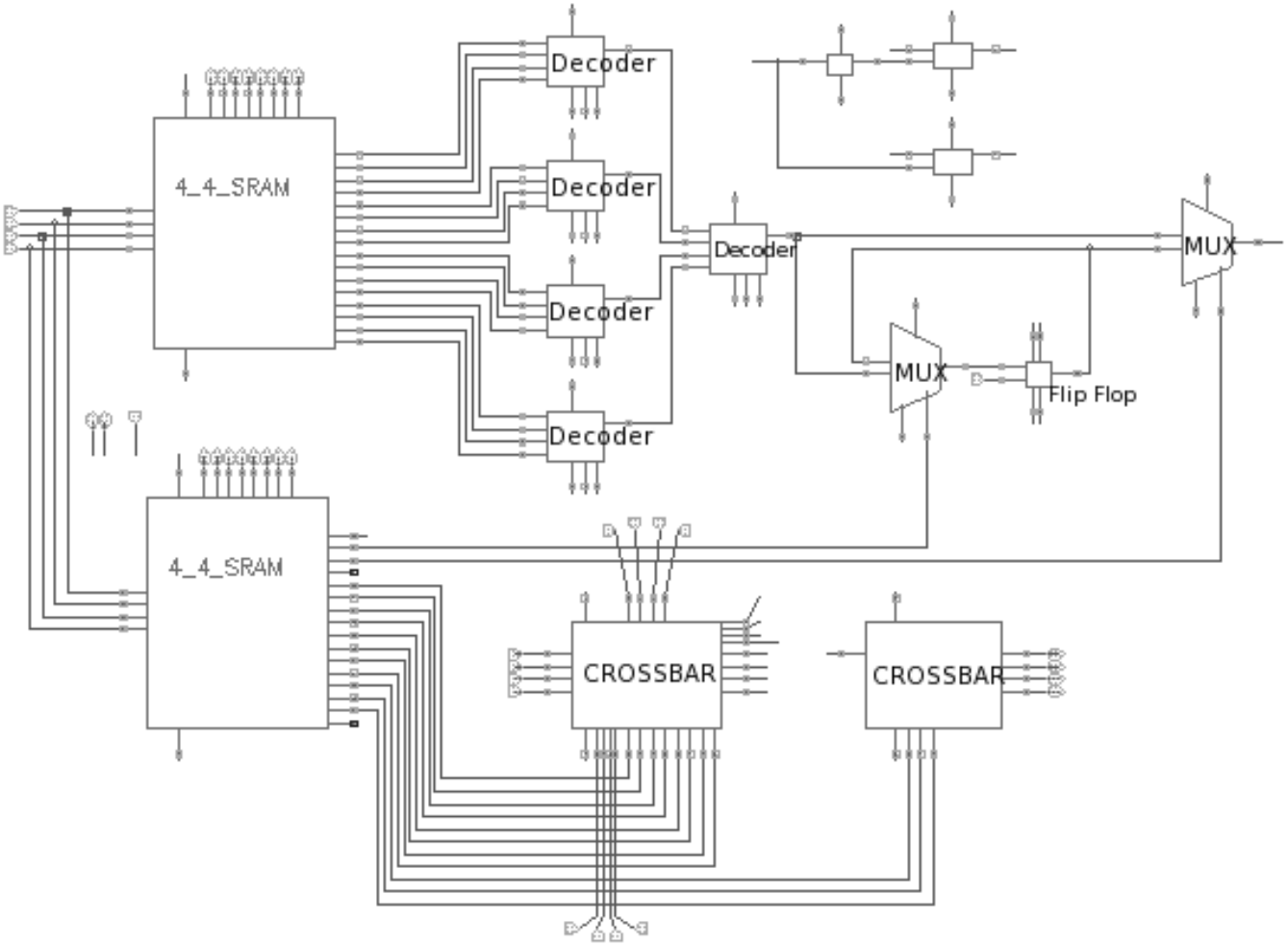
Designing the CLB: Delay model



Here, $\mu = C_{R3}/C_{L3}$ increases with better technology nodes as routing capacitance starts becoming dominant compared to logic capacitance[3]

μ = 20.5/8.25 = 2.5 (using *captab* in ADE L)

Designing the CLB: Schematic

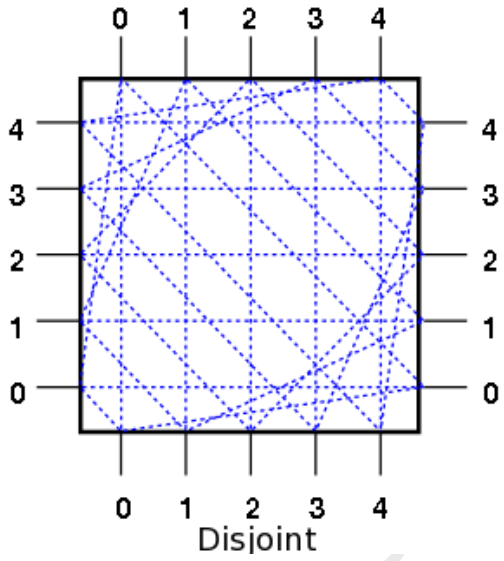


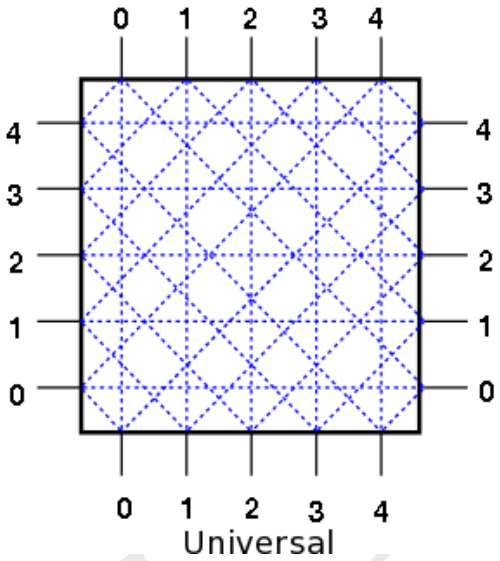
Schematic (Cadence Schematic L)

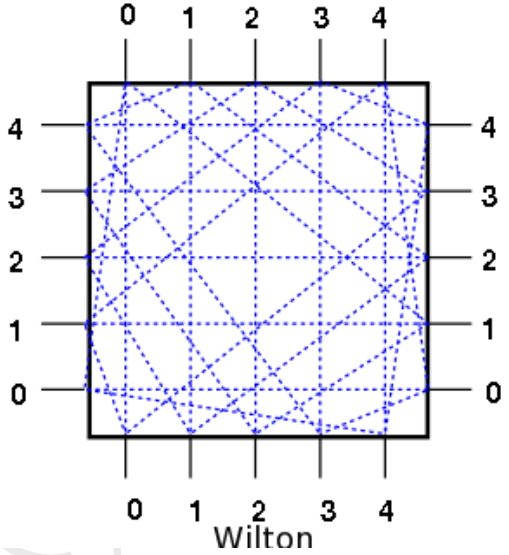
Using k=4 as it is optimal in an Area-Delay Product sense

Designing the Switch Matrix

- Different Types ([8],[9])



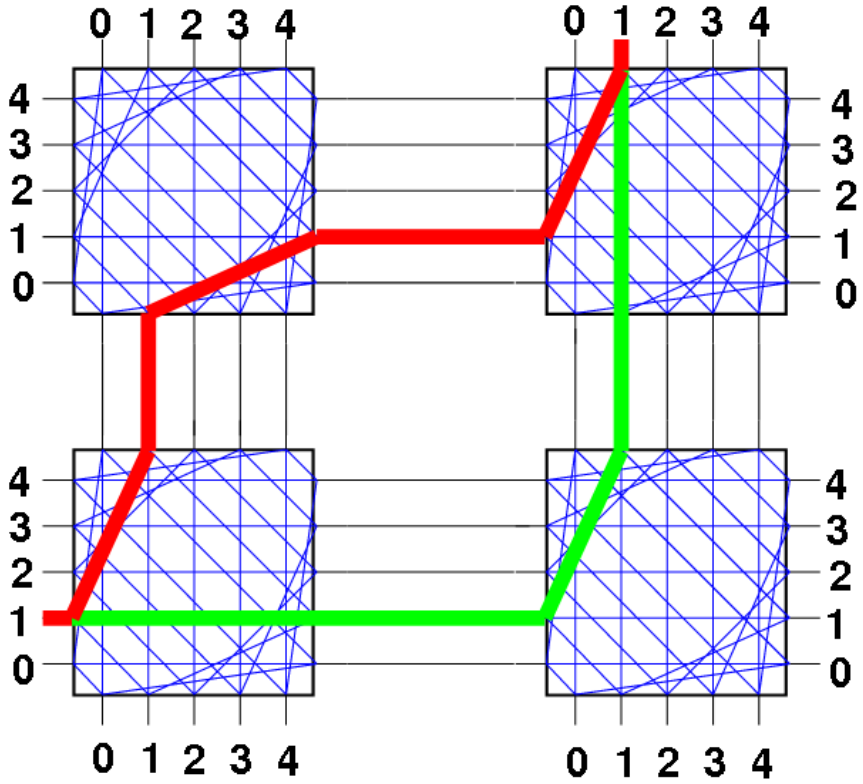




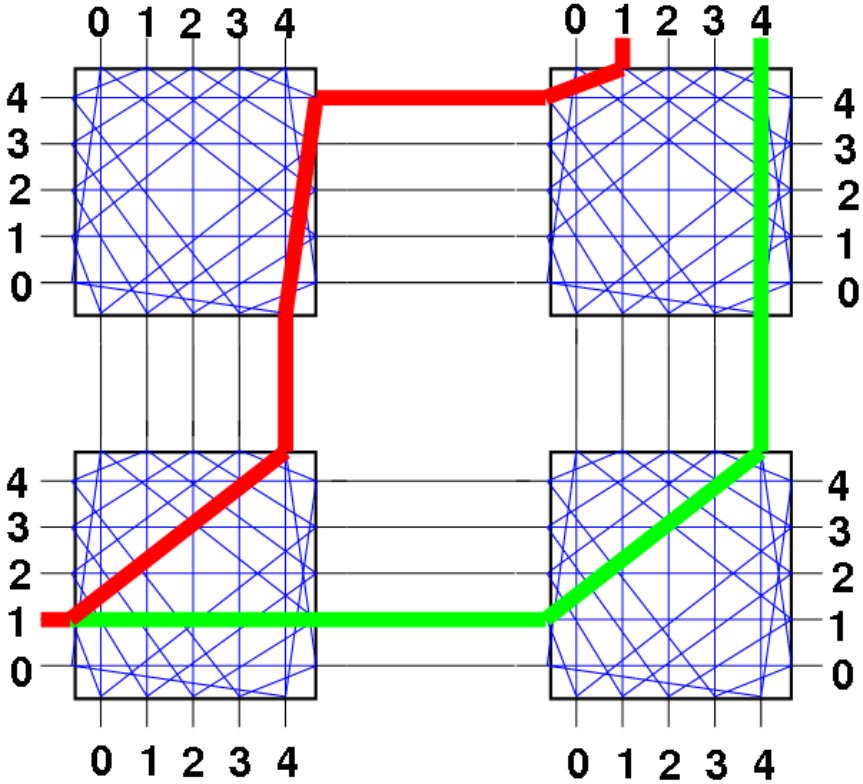
- Why not allowing all permutations?
  - Overkill + Area and delay overheads

Designing the Switch Matrix (type)

Disjoint Switch Matrix



Wilton Switch Matrix

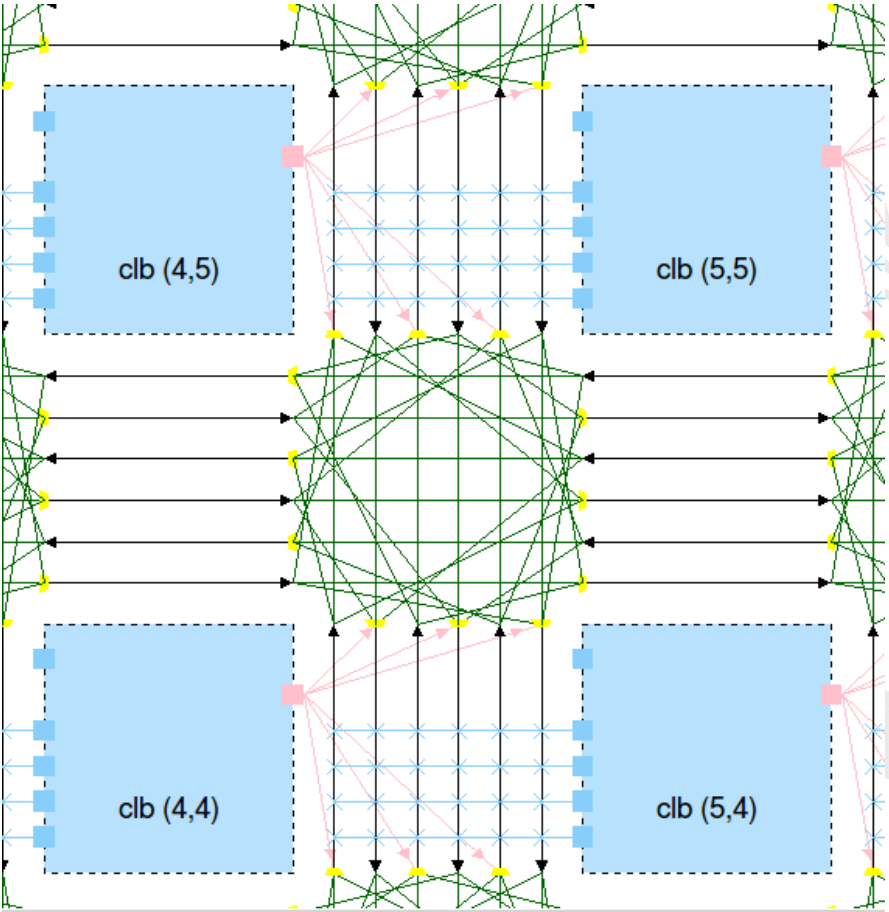


Therefore, we choose unidirectional Wilton Switch Matrix design

- Why Unidirectional??

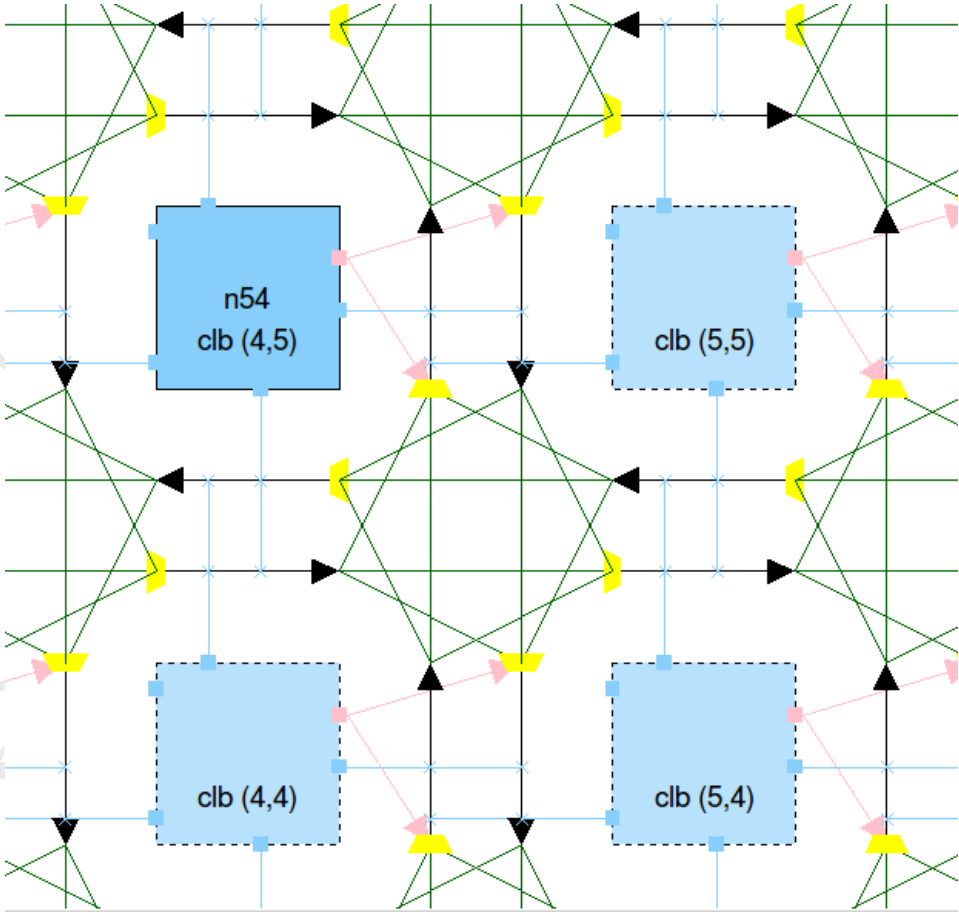
Designing the Switch Matrix (pinout)

All inputs from left



Routing succeeded with a channel width factor of 6.

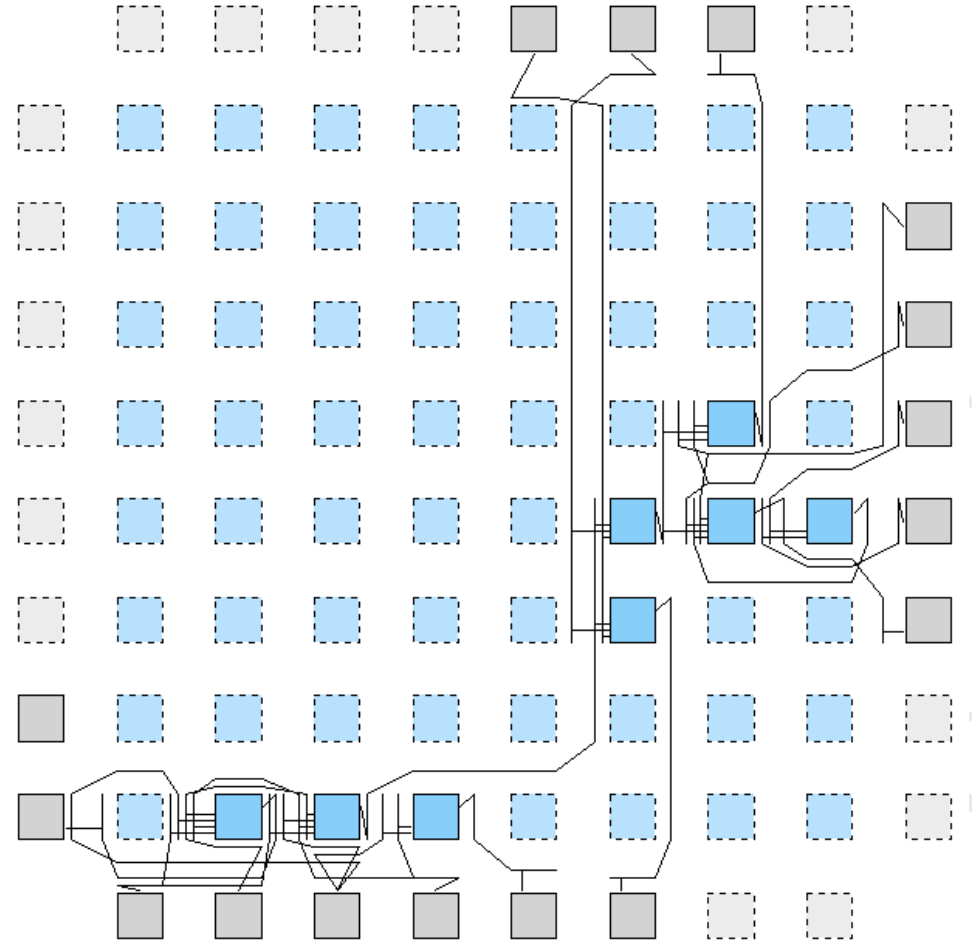
One input each side



Routing succeeded with a channel width factor of 2.

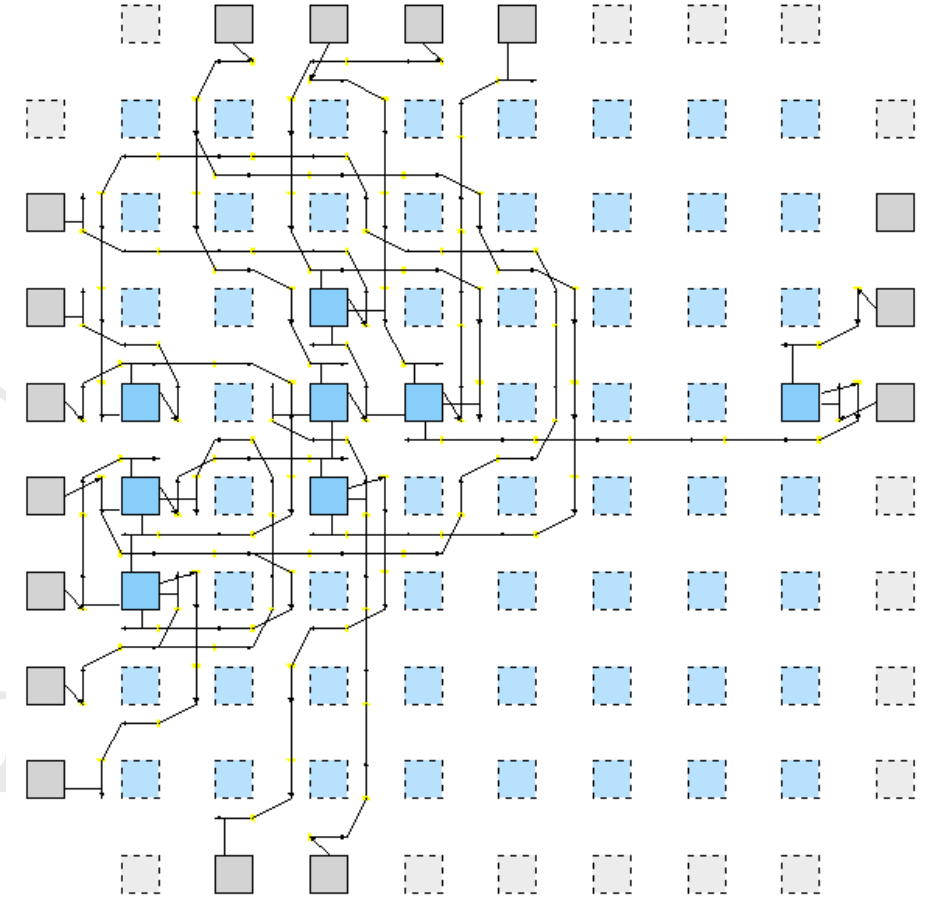
Designing the Switch Matrix (pinout)

All inputs from left



Routing succeeded with a channel width factor of 6.

One input each side

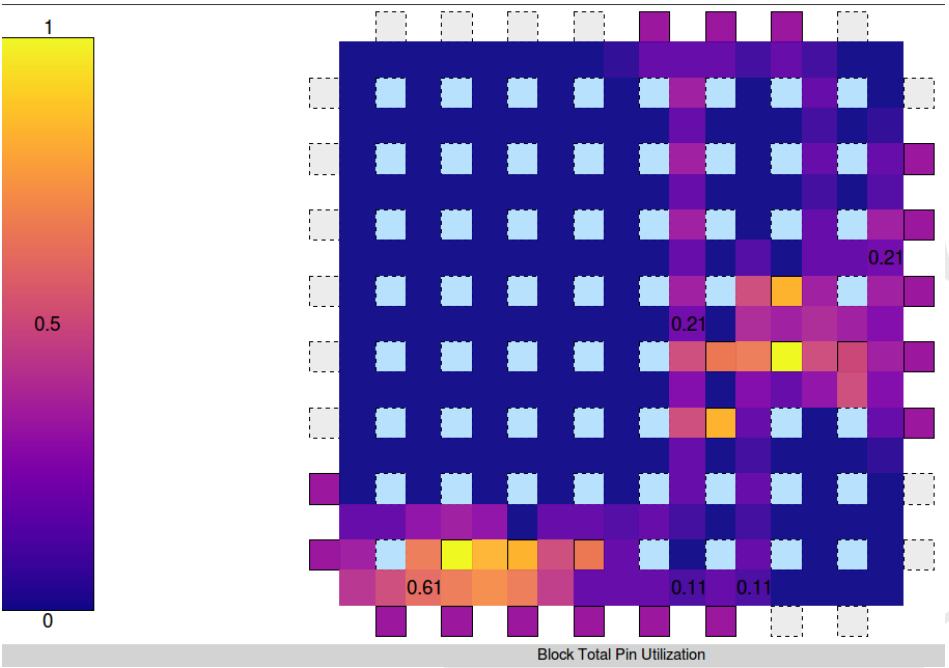


Routing succeeded with a channel width factor of 2.

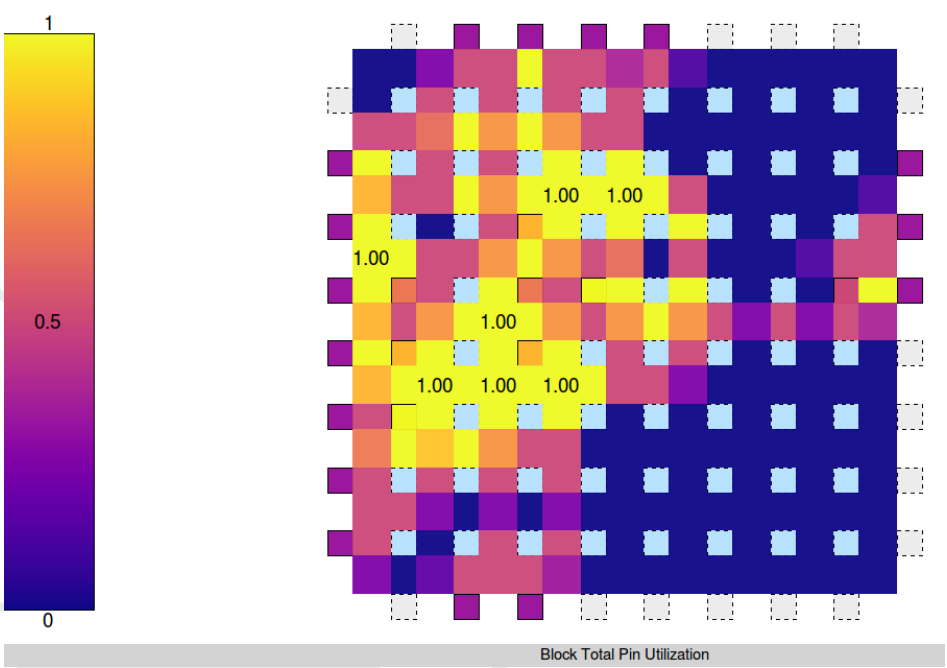
W=6 ~~ 5-bit adder implementation ~~ W=2

Designing the Switch Matrix (pinout)

All inputs from left



One input each side



Routing congestion

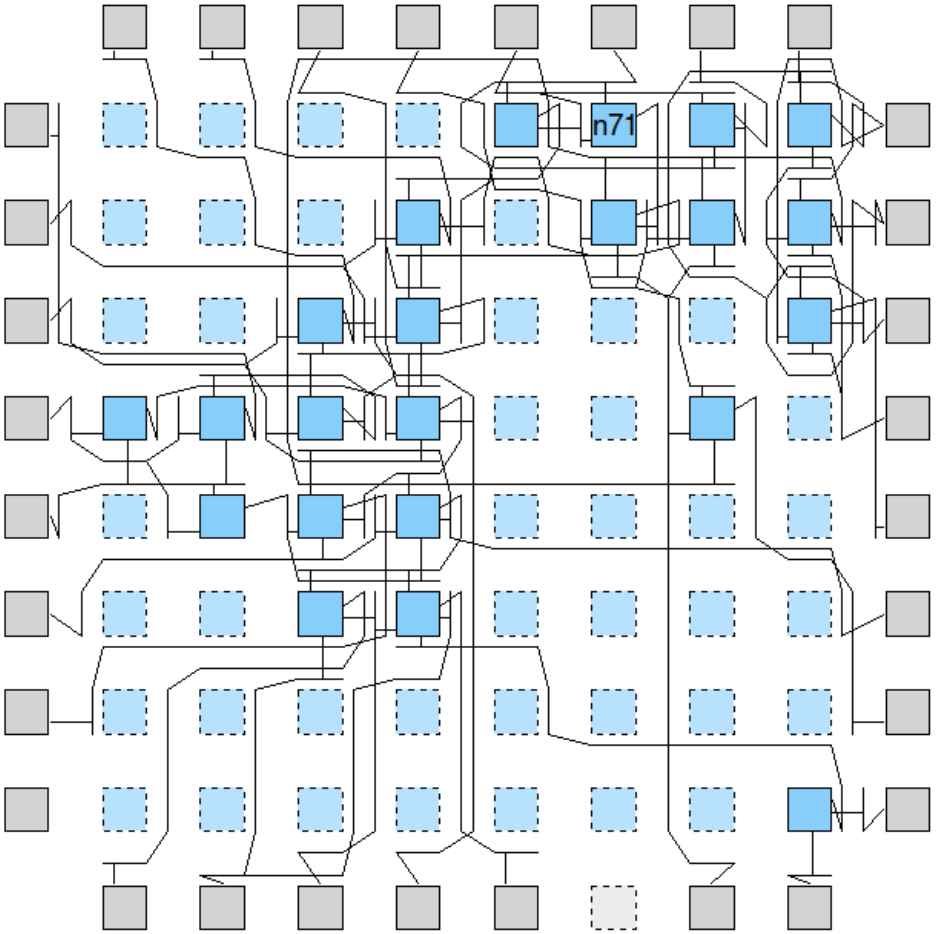
Designing the Switch Matrix (pinout)

The area overheads in making an all sided input crossbar are easily shadowed by the area and delay benefits gained from reduction in routing width(W).

Therefore, we go ahead with one input from each side

Designing the Switch Matrix (width)

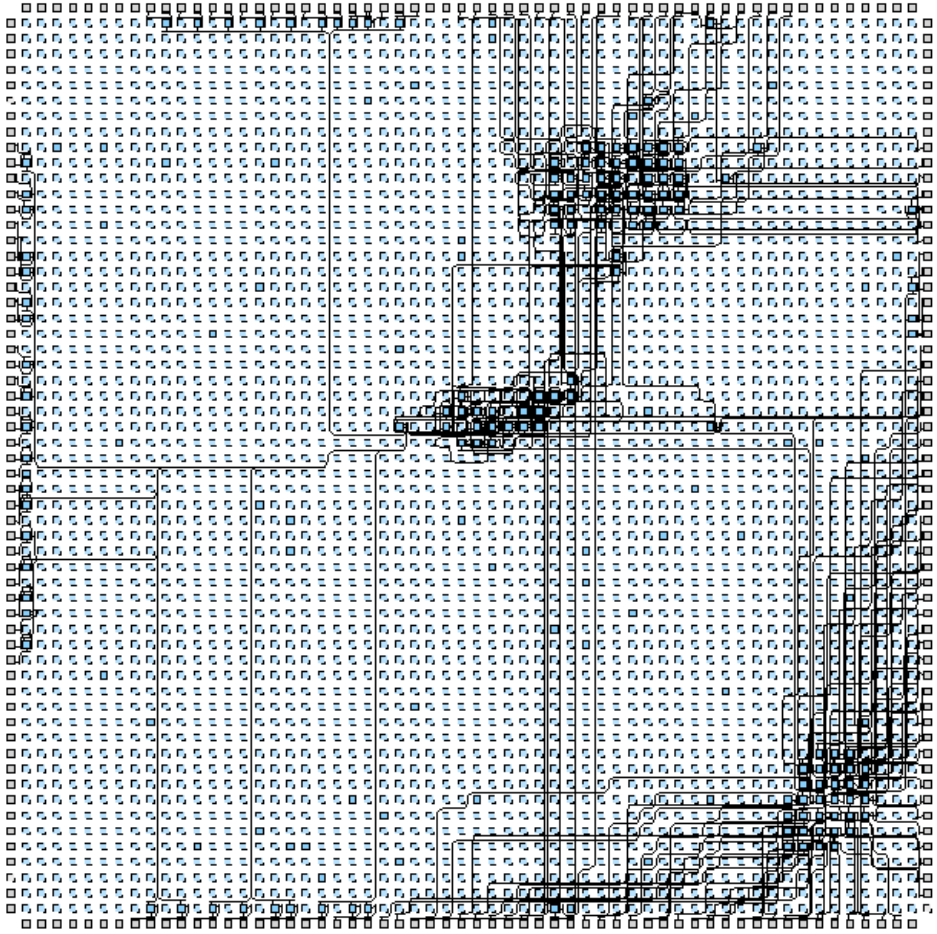
10-bit Adder



Routing succeeded with a channel width factor of 4.

W=4 ; 8X8

Complex memory controller

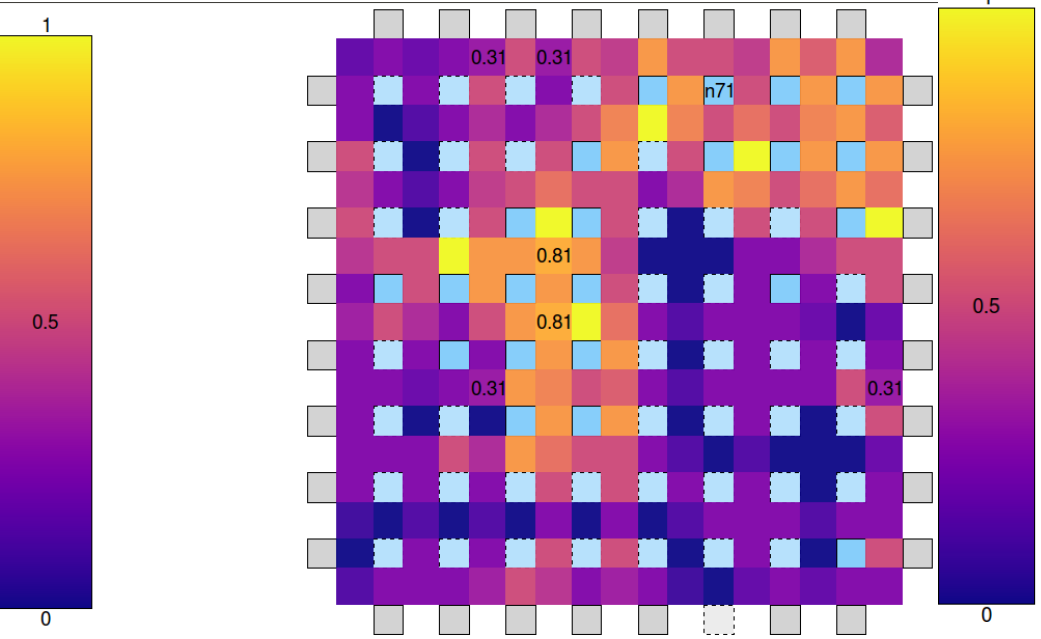


Routing succeeded with a channel width factor of 4.

W=4 ; 60X60

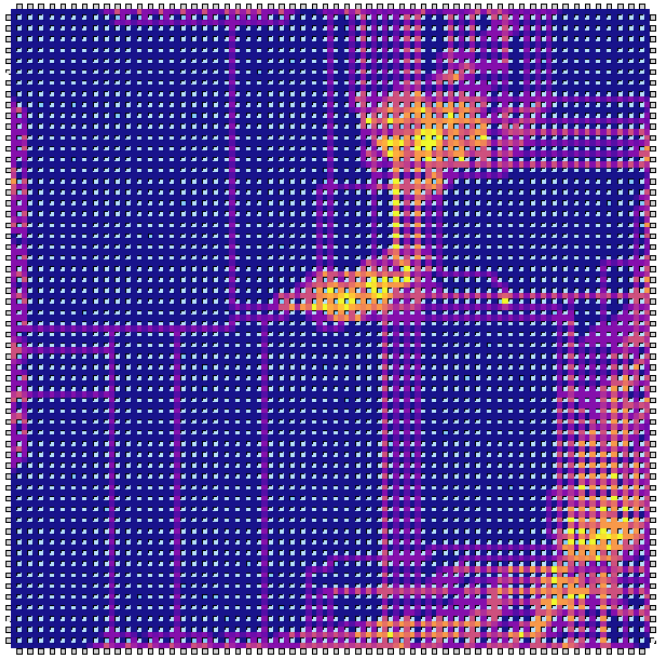
Designing the Switch Matrix (width)

10-bit Adder



Routing succeeded with a channel width factor of 4.

Complex memory controller



Routing succeeded with a channel width factor of 4.

Routing congestion

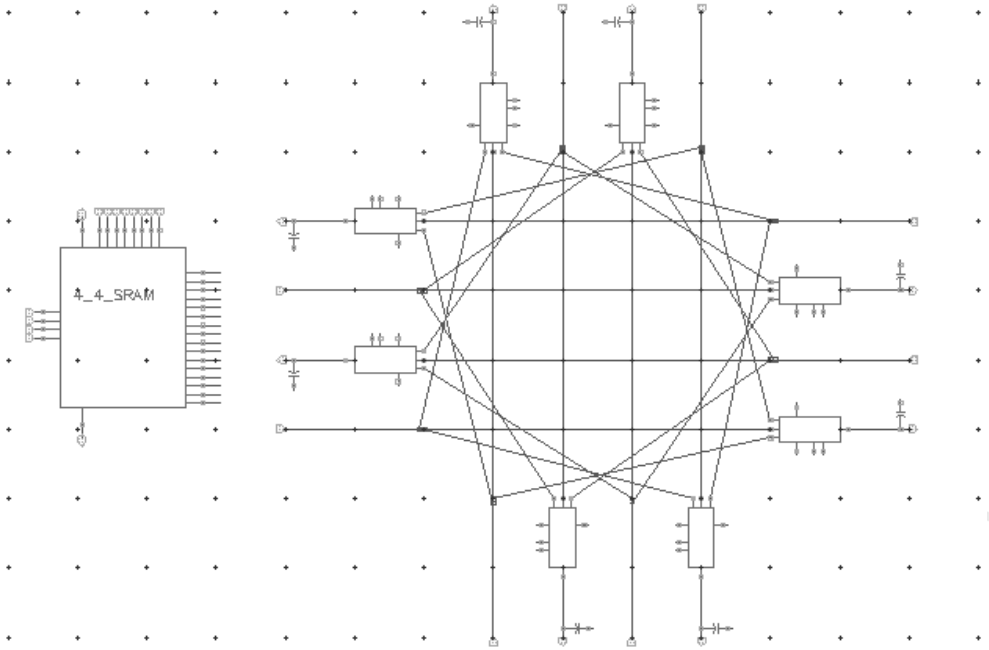
Designing the Switch Matrix (width)

Verilog Design	Grid size		Routing width	
	All left I/O CLB	Spread I/O CLB	All left I/O CLB	Spread I/O CLB
adder.v	6x6	6x6	4	4
multiplier.v	10x10	10x10	8	6
and_latch.v	3x3	3x3	2	2
ch_intrinsics.v	60x60	60x60	6	4
multiclock_separate_and_latch.v	4x4	4x4	4	2
multiclock_output_and_latch.v	4x4	4x4	4	2
multiclock_reader_writer.v	6x6	6x6	6	4

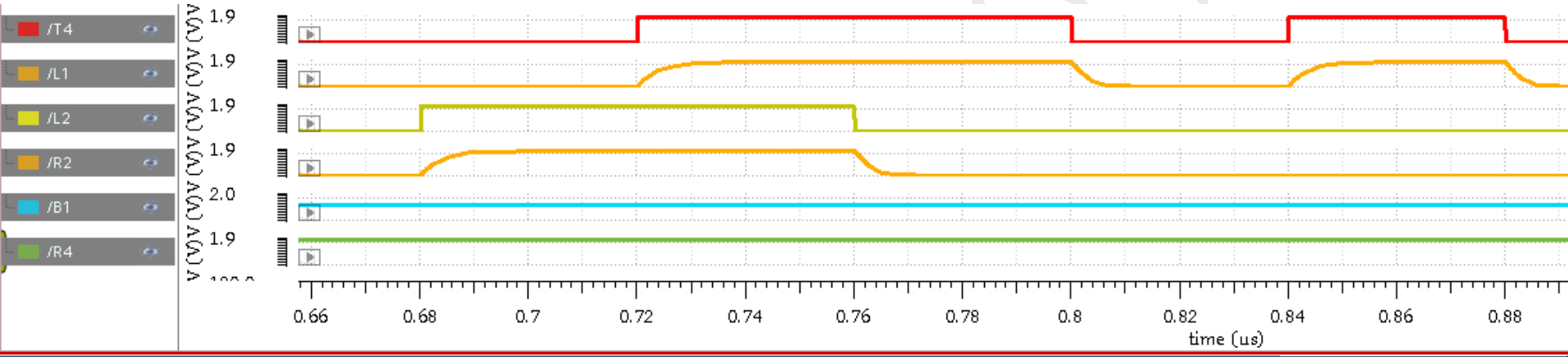
We see that number of CLBs scales up with bigger designs while routing width requirements remain constant at around 4.

- Therefore, we go ahead with Routing Width, W = 4

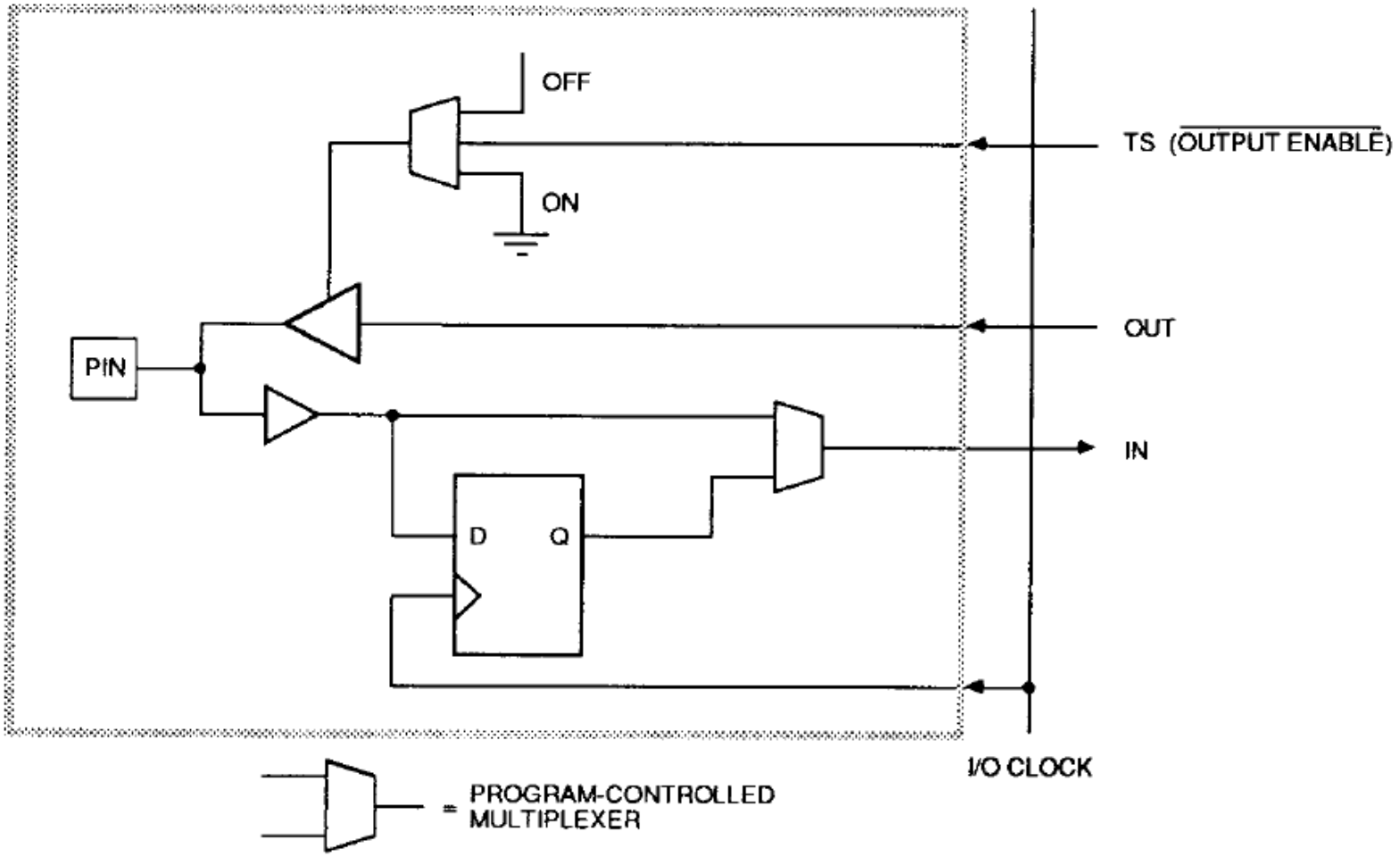
Designing the Switch Matrix (schematic)



- Schematic and simulations
- Symmetric delays



Designing the I/O Block



- From Xilinx 2064[10]

Designing the I/O Block (VTR XML)

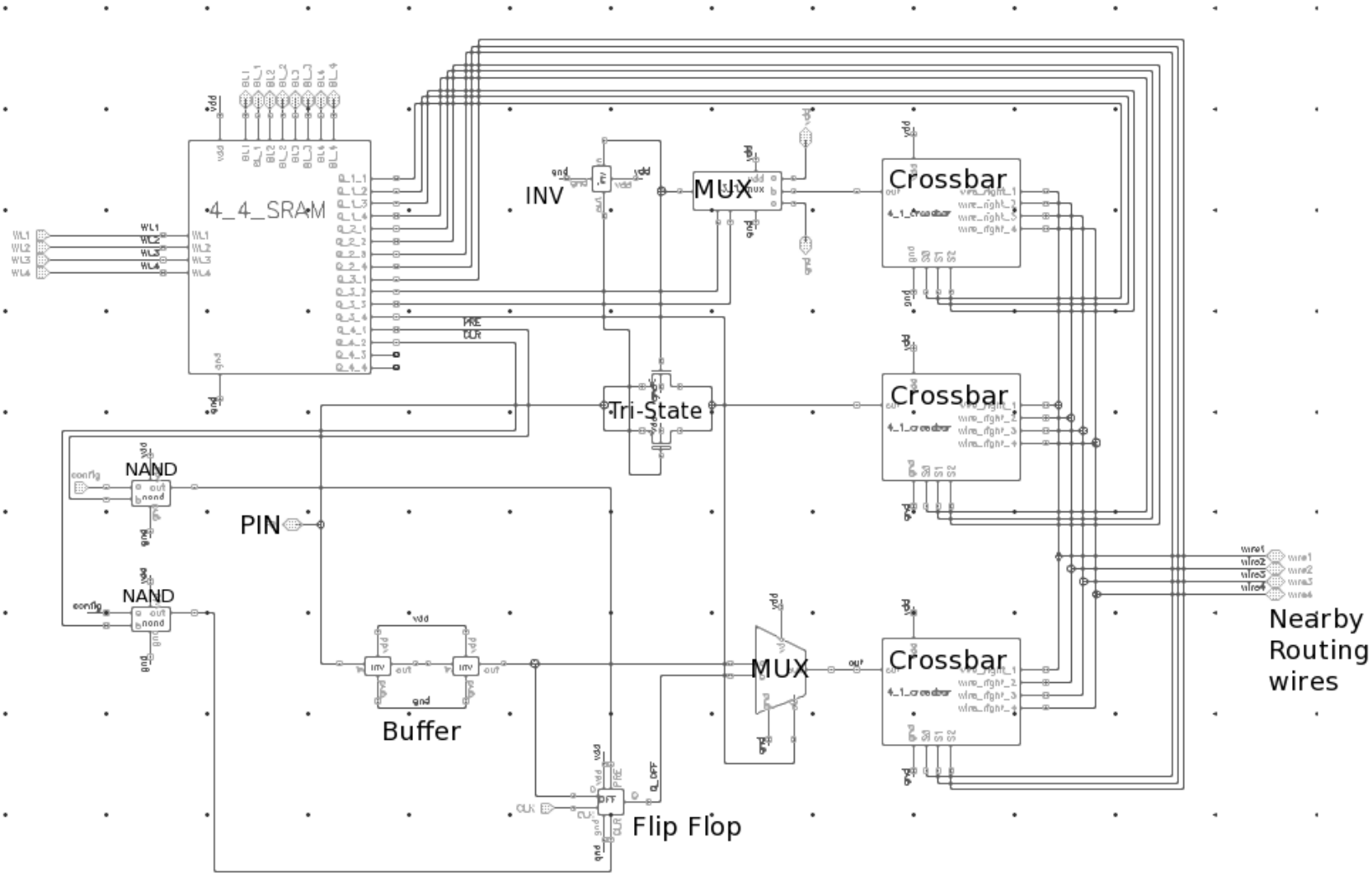
```
  <!-- Every input pin is driven by 100% of the tracks in a channel, every output pin is driven by 100% of the tracks in a channel -->
  <fc in_type="frac" in_val="1" out_type="frac" out_val="1"/>

  <!-- IOs go on the periphery of the FPGA, for consistency,
      make it physically equivalent on all sides so that only one definition of I/Os is needed.
      If I do not make a physically equivalent definition, then I need to define 4 different I/Os, one for each side of the FPGA
    -->
  <pinlocations pattern="custom">
    <loc side="left">io.outpad io.inpad io.clock</loc>
    <loc side="top">io.outpad io.inpad io.clock</loc>
    <loc side="right">io.outpad io.inpad io.clock</loc>
    <loc side="bottom">io.outpad io.inpad io.clock</loc>
  </pinlocations>

  <!-- Place I/Os on the sides of the FPGA -->
  <power method="ignore"/>
</pb_type>
<!-- Define I/O pads ends -->
```

- Describing I/O Block in XML format for importing it to VTR: Verilog-To-Routing

Designing the I/O Block (schematic)



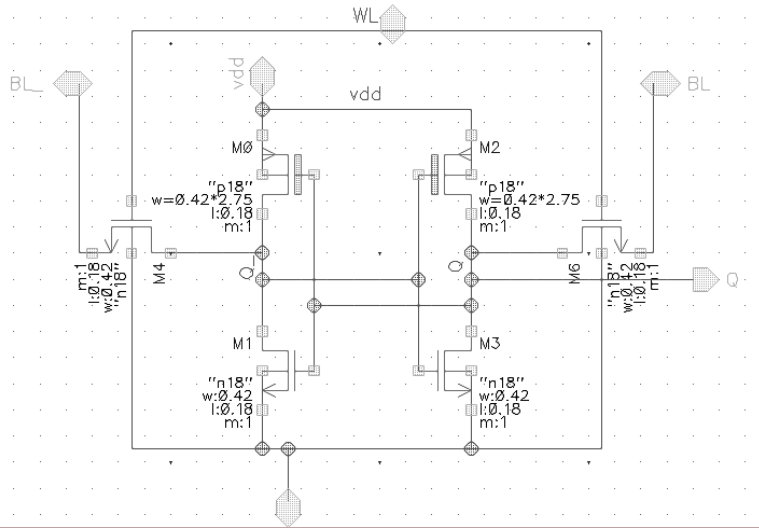
- Schematic in Cadence Schematic L

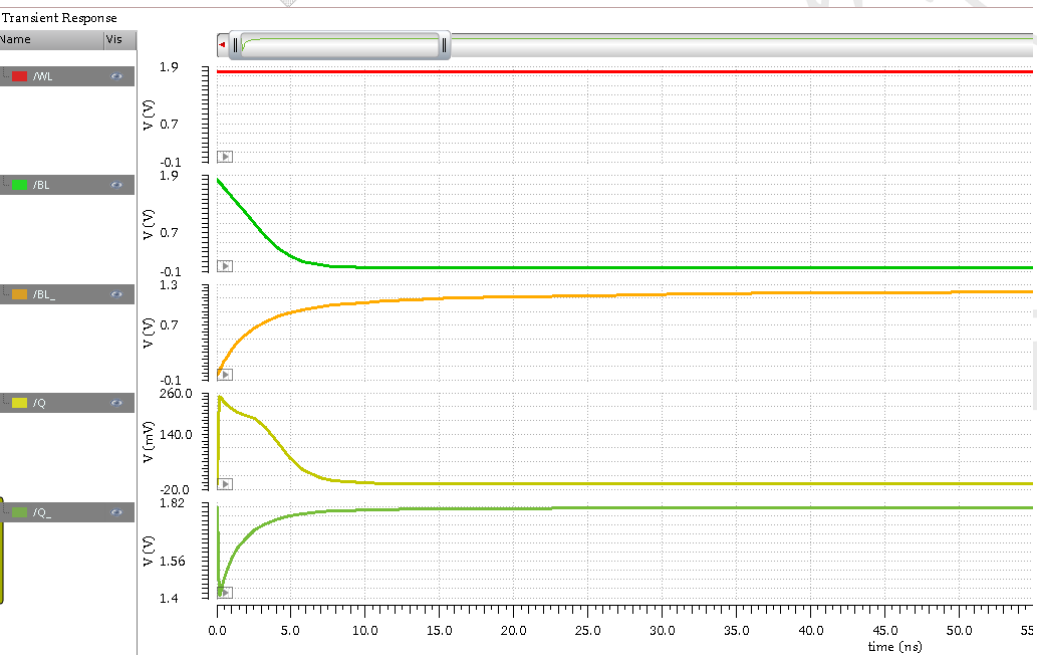
Programmability of the FPGA

1. CLB
   - 4x4 SRAM : LUT memory
   - 4x4 SRAM : configuration memory
2. Switch Matrix*
   - 4x4 SRAM : configuration memory
3. I/O Block
   - 4x4 SRAM : configuration memory

- Shared Word Lines and Bit Lines
- Program 4 bits at a time

Programmability of the FPGA (6T-cell)



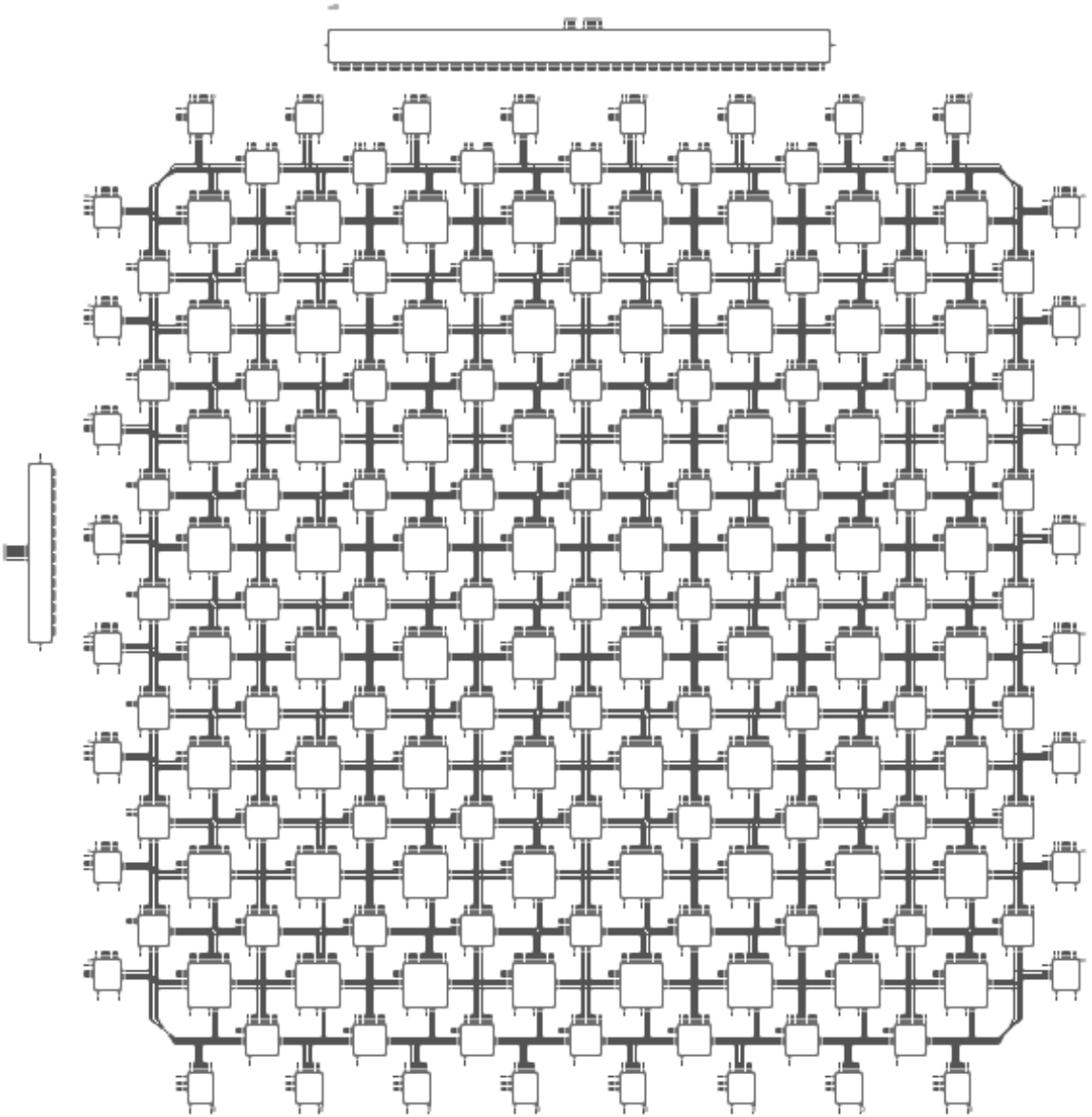


- Schematic and simulations
- Notice how a cell recovers from stray charges on the Bit Line
- $C_{BL} = 5 * C_{captab}$

Programmability of the FPGA (decoders)

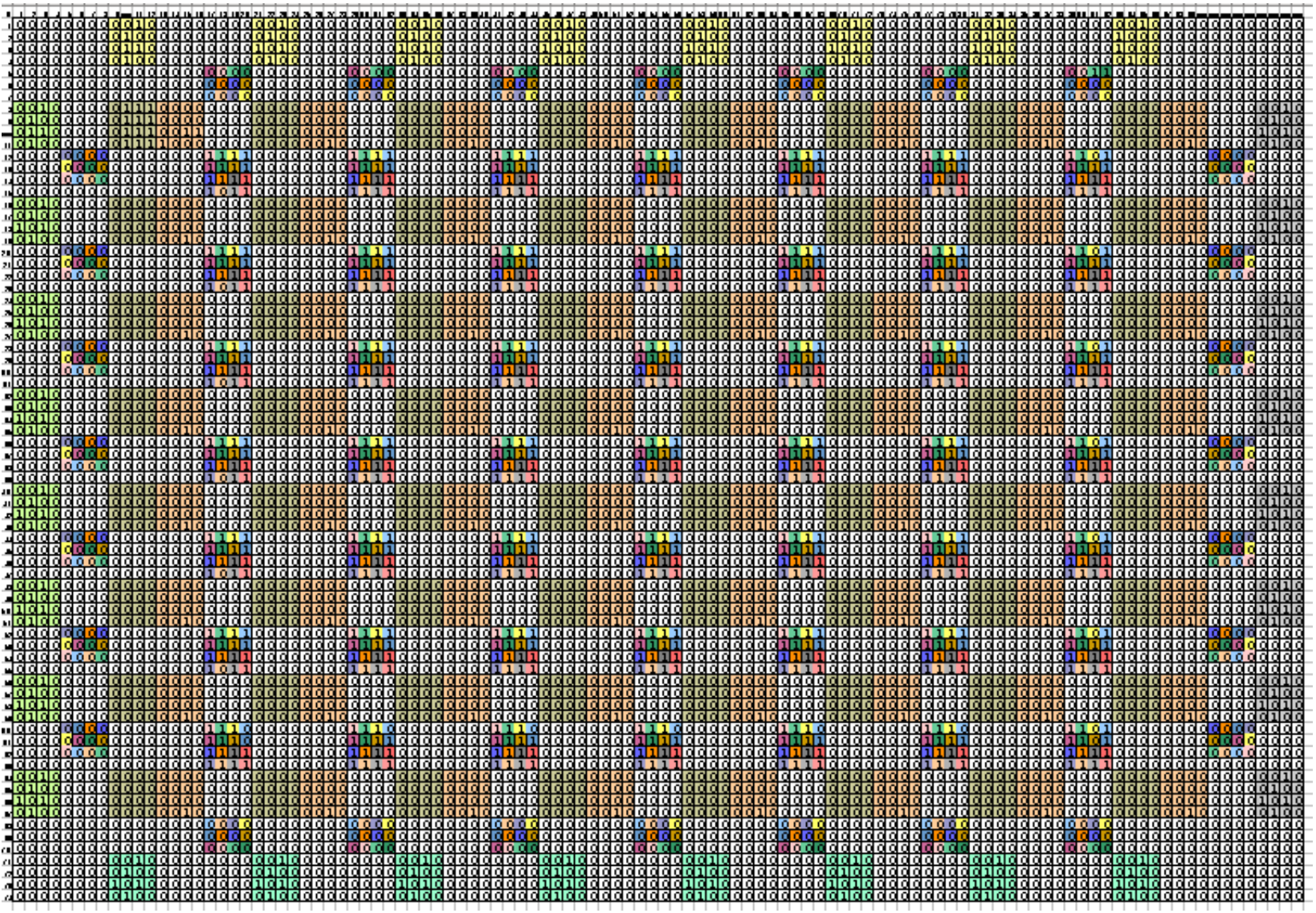
- Schematic for row and column decoders
- Can’t make any sense?
- Generated using Cadence SKILL descriptions!
- Hot-swapping / Partial Reconfiguration

Results and Discussion (schematic)



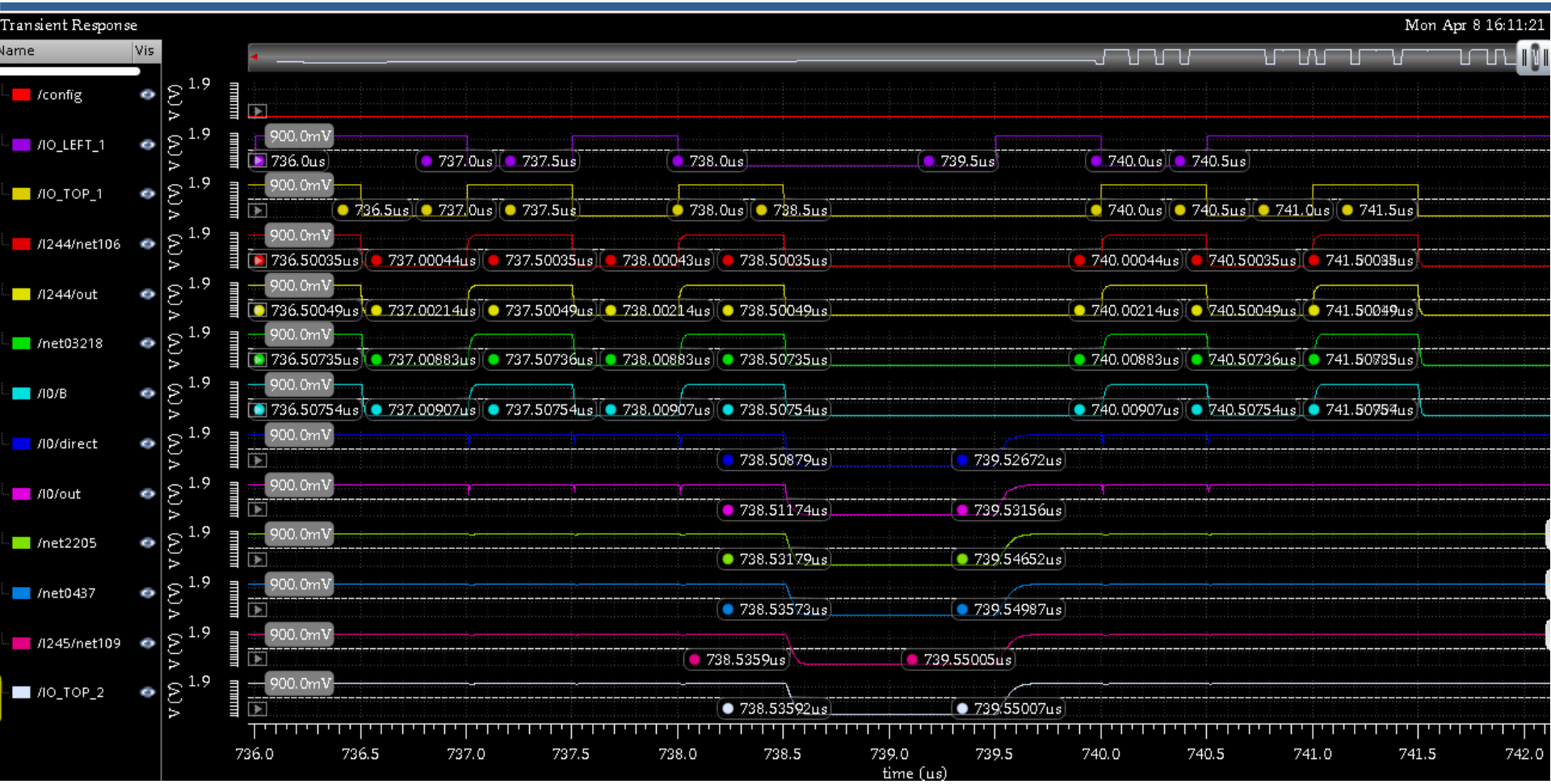
- Schematic for an 8x8 FPGA

Results and Discussion (bitstream)



Bitstream for 2-input OR

Results and Discussion (simulations)



Spectre simulation for 2-input OR

Results and Discussion (delays)

- I/O Block Top 1 : 7.43 ns
  - Input buffer : 0.35 ns
  - 2X1 MUX : 0.14 ns
  - 4X1 Crossbar : 6.94 ns
- CLB 1 : 24.44 ns
  - 4X4 Crossbar : 0.19 ns
  - 2X1 MUX : 1.25 ns
  - 2X1 MUX : 2.95 ns
  - 4X1 Crossbar : 20.05 ns
- Switch Matrix : 3.94 ns
- I/O Block Top 2 : 0.19 ns
  - 4X1 Crossbar : 0.17 ns
  - Tx-Gate : 0.02 ns

Output fall time = 36 ns

Similarly,

Output rise time = 50 ns

Frequency ?

Scope for future work

- Developing the layout of CLB, Switch Matrix and I/O Block
- Laying out the FPGA from its constituents using SKILL and performing post-layout simulations.
- Designing a framework to automate the layout for NxN FPGA
- Delving into the digital flow for optimizing the sizing of transistors at a finer granularity and then developing models to be used for a generic NxN FPGA

Thank You