Project: Bit Slice

Matthew Taylor, Fall 2023

1 Overview

Modern digital integrated circuits can contain a stagger number of transistors. NVIDIA's current flagship GPU is based on their GH100 die, which has over 80 billion transistors. The geometry and position of these transistors determine key factors like power usage and maximum speed.

It would be virtually impossible to design a chip of that scale by hand. Luckily, these massive chips have many redundant structures. The bit slice is an integral piece that contains the circuitry for processing a single bit of data. This is then tiled for the width of the CPU; 8 times for an 8-bit CPU, 64 for 64-bit, etc.

This project is meant to demonstrate this principle, while introducing us to the tooling and workflow. The objective is to design a processor data path that can be fabricated on the SkyWater SKY130 process, following a design flow like this:

1. Design a circuit schematic in `sue`, and generate an IRSIM netlist.
2. Write an IRSIM command file to test and validate the design.
3. Recreate the schematic in `xschem` to create a SPICE netlist.
4. Draw a stick diagram to plan the physical layout.
5. Draw a physical layout in `magic` following the stick diagram, and extract a SPICE netlist for the cell.
6. Use `netgen` to perform LVS (layout-versus-schematic), comparing the SPICE netlist from the schematic and from the cell.

The data path is designed as a hierarchy of circuits:

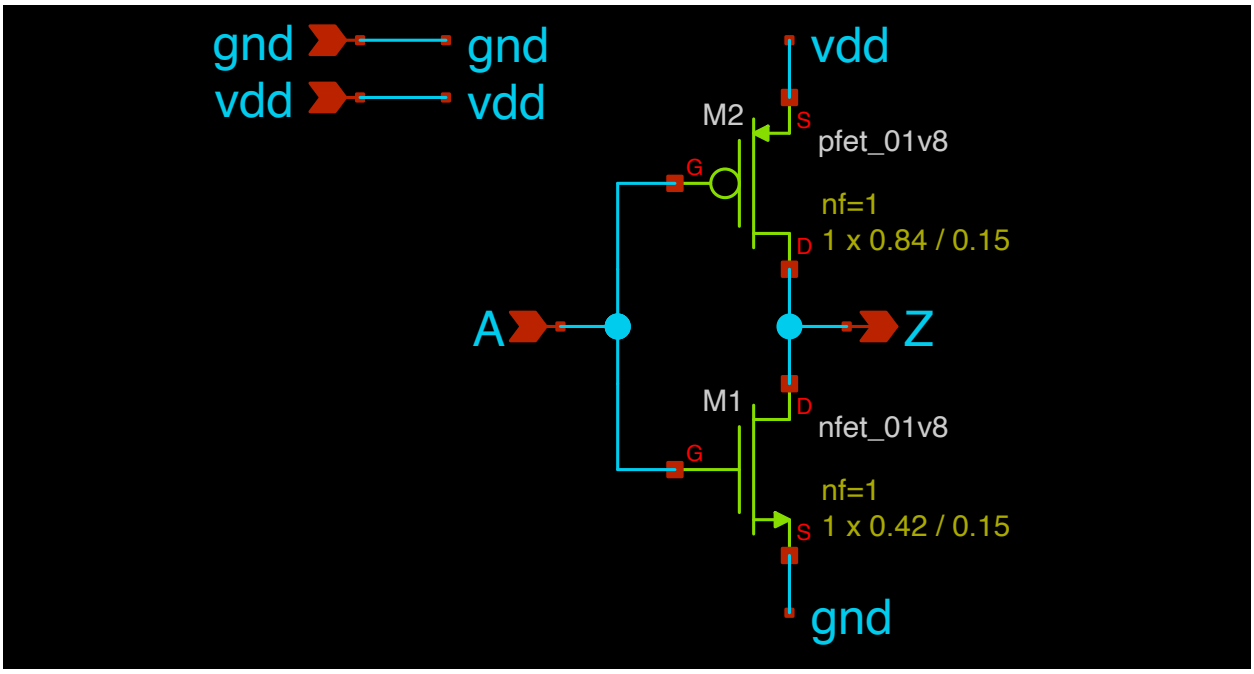
- 1-bit full adder
- 1-bit D flip-flop
- 2-1 multiplexer
- 2-input AND gate
- 1-bit register
- Bit slice
- 8-bit data path

2 Schematics

The bit slice is built up from hierarchical and leaf sub-circuits. Hierarchical circuits made up of leaf circuits, and represent high-level constructs. Leaf circuits are made up of transistors directly, so their physical sizes can be directly controlled.

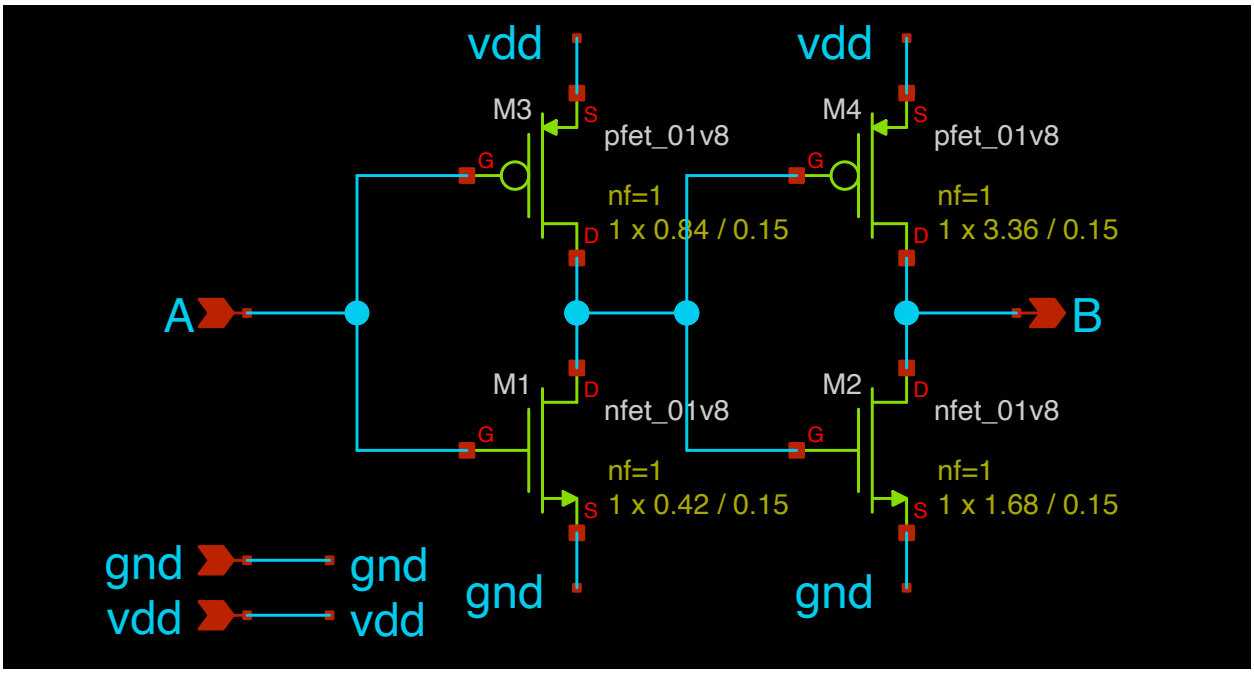
2.1 Leaf circuits

Inverter (`inv.sch`)



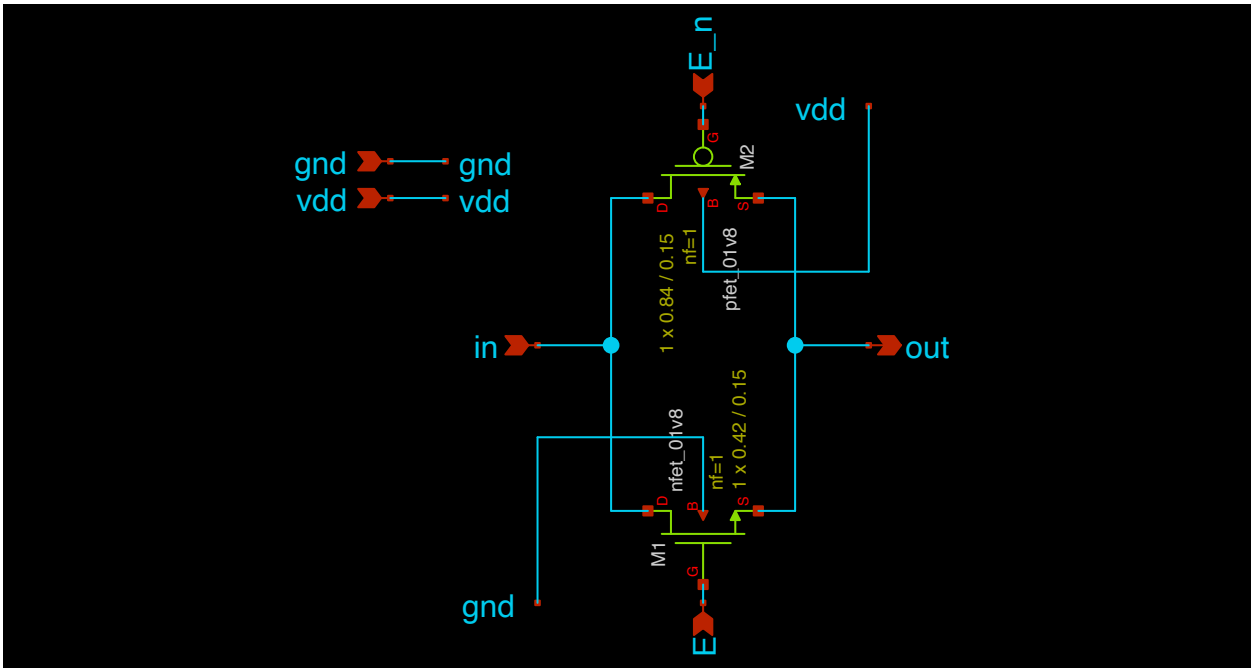
The inverter outputs the opposite value of the input.

Buffer (`buffer_fo4.sch`)



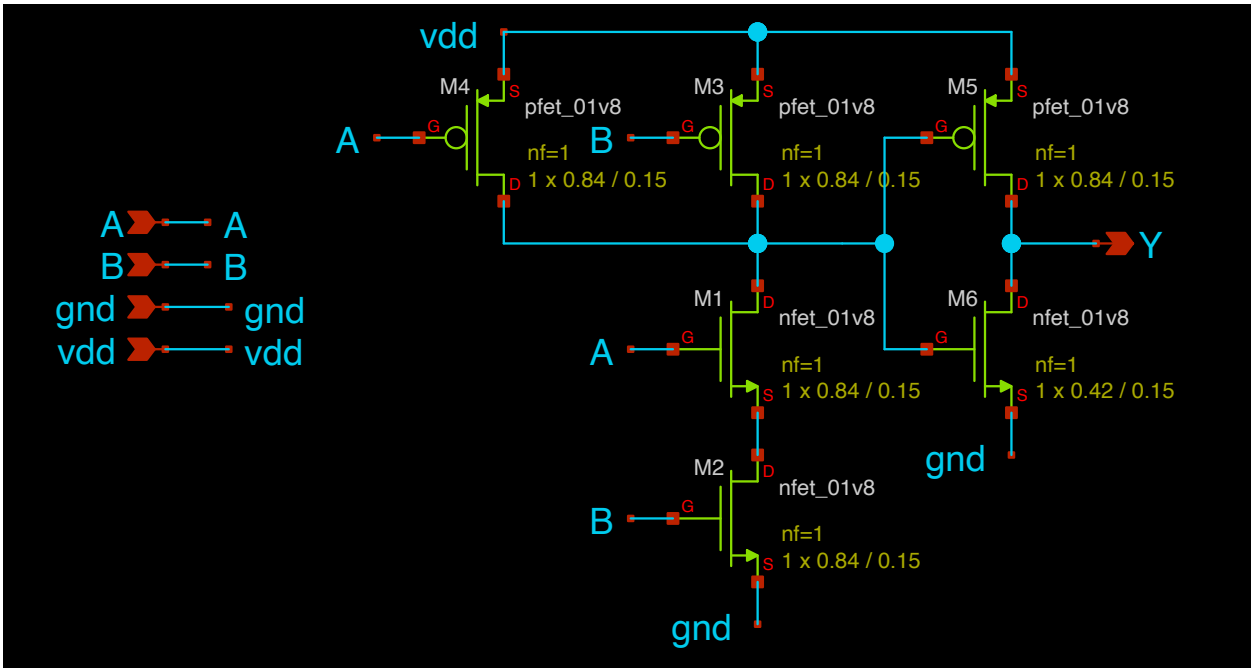
The buffer uses two sequential inverters to output the same value as the input, but electrically isolate the input and output. In this design, the first inverter is quite small, and the second is four times larger. This creates a delay, which is useful for ensuring correct sequencing, at the cost of area and speed.

Transmission Gate (t_gate.sch)



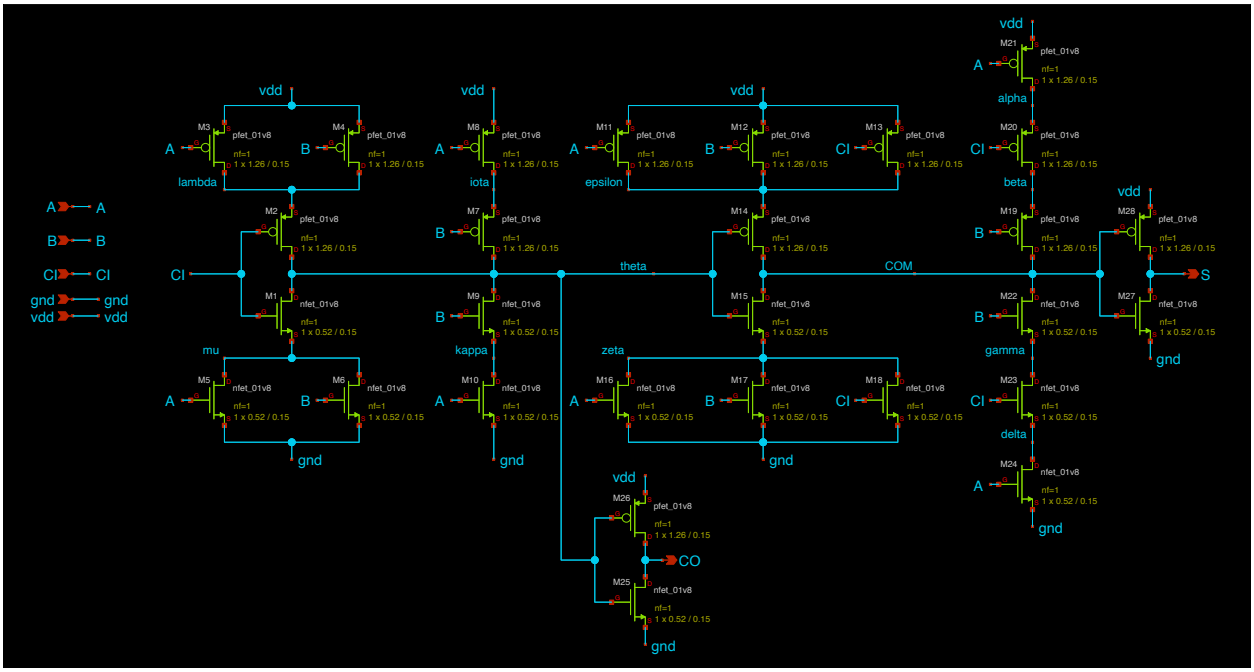
The transmission gate uses two transistors to connect two nodes together.

AND Gate (and2.sch)



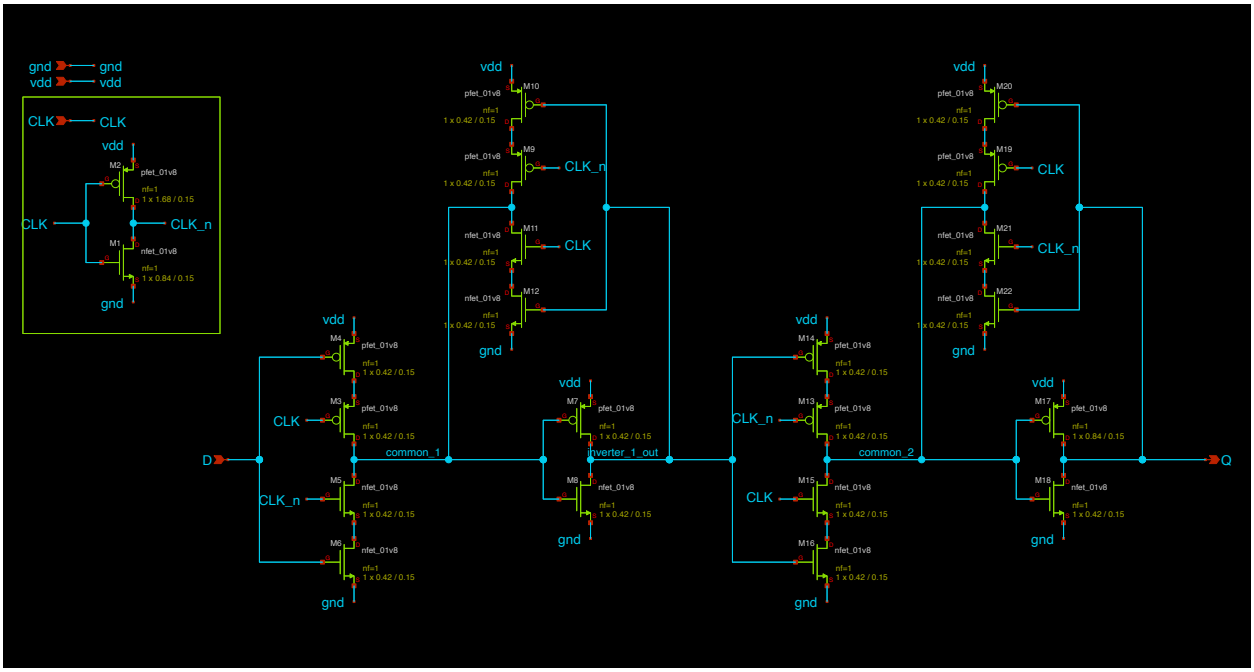
The AND gate outputs a logical high only if both inputs are high.

Full Adder (addf.sch)



This circuit was provided by Dr. Stine, and redrawn in xschem. The first half of the circuit calculates the carry out, and the second half calculates the sum.

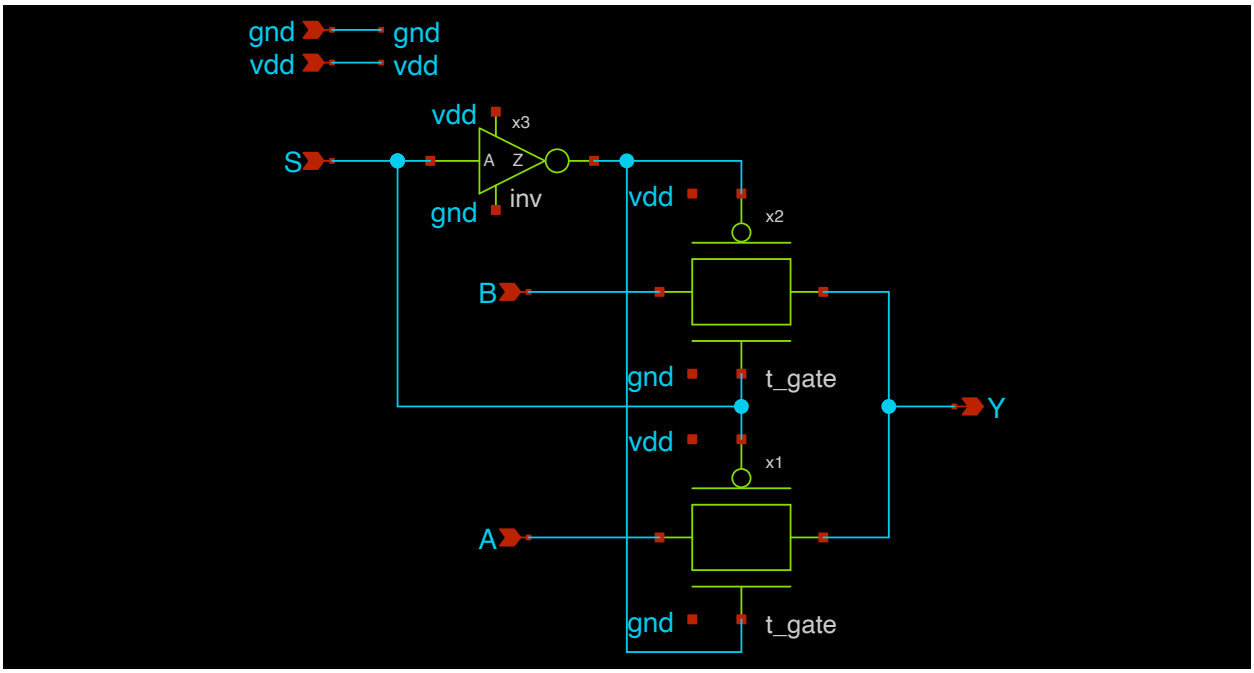
D Flip-Flop (d_flip_flop.sch)



The D flip-flop uses four transmission gates to hold a stable value, only taking a new value on the rising edge of the clock (CLK).

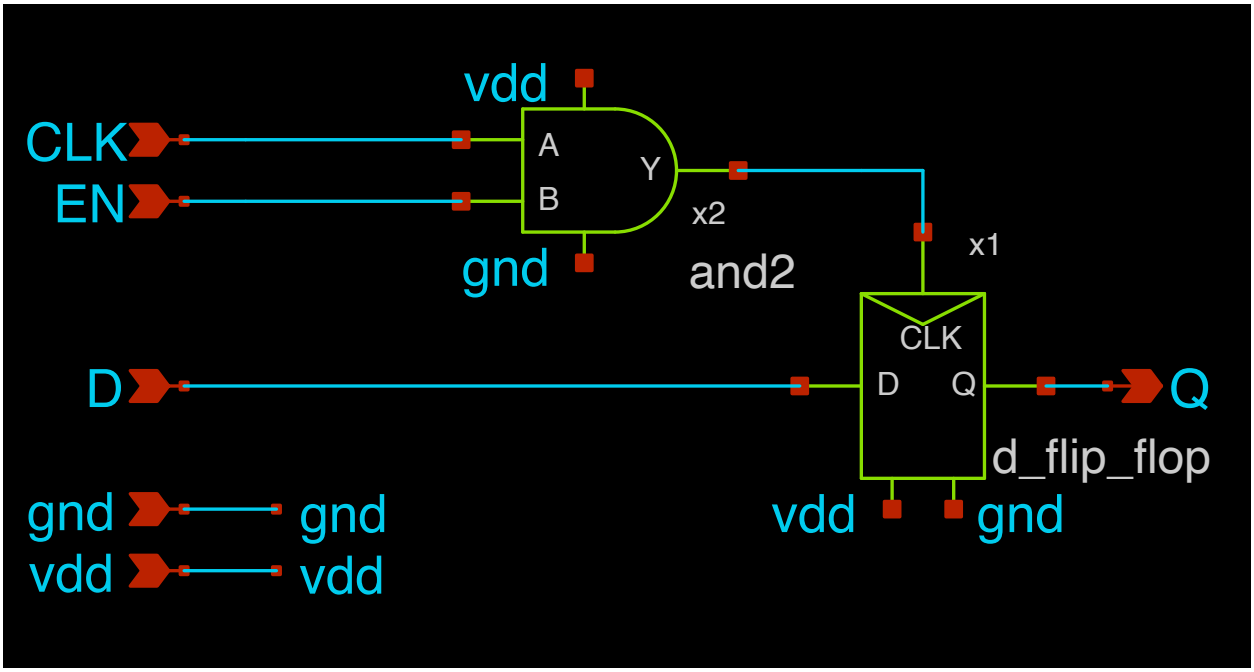
2.2 Hierarchical circuits

Multiplexer (mux21.sch)



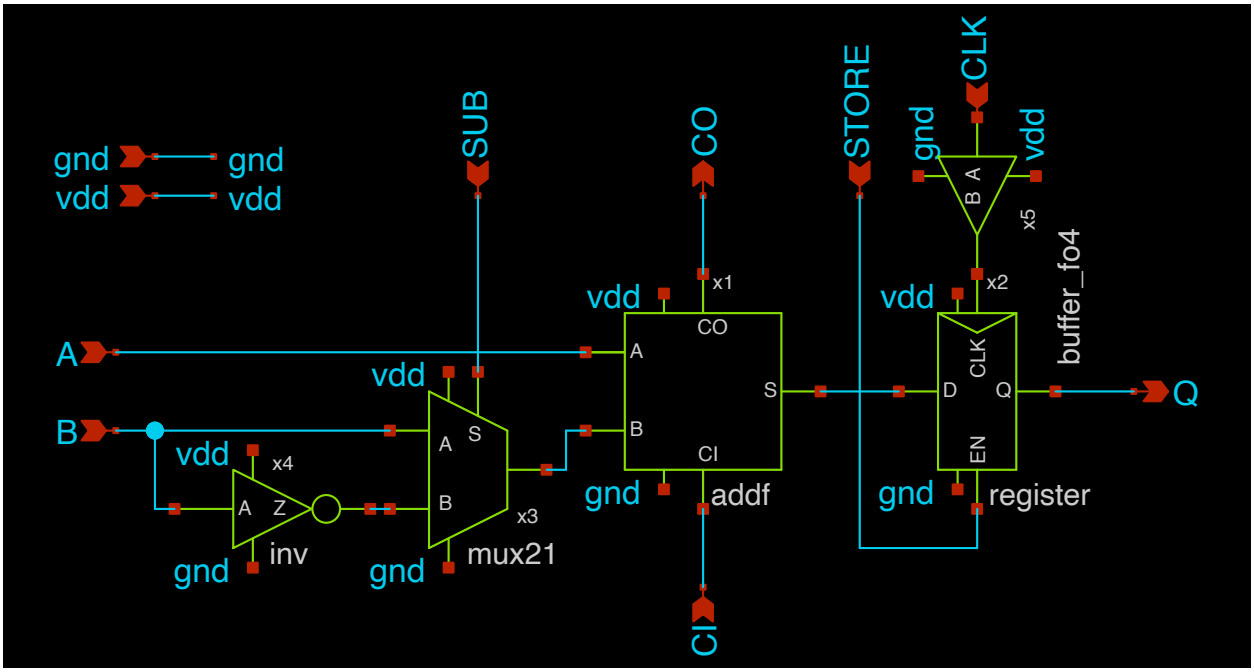
The multiplexer uses two transmission gates to permit either A or B to be connected to Y. This is controlled by the select input (S).

Register (register.sch)



The register extends the D flip-flop with a AND gate. The AND gate blocks the clock if enable (EN) is not asserted, which makes the D flip-flop hold its current value.

Bit slice (`bitslice.sch`)



The bit slice takes three data inputs (A, B, C_{in}) and has two data outputs (Q, C_{out}).

The function is based on the two control inputs (SUB, STORE) and a clock input. It can add or subtract A and B and compute carry/borrow.

The result is stored at the rising edge of the clock if STORE is asserted. Otherwise, the register keeps the previous value of Q.

3 Circuit Simulations

It is paramount to verify that schematics accurately describe the intended circuits, and that the circuits perform their intended function. To this end, the schematics were checked in two main ways:

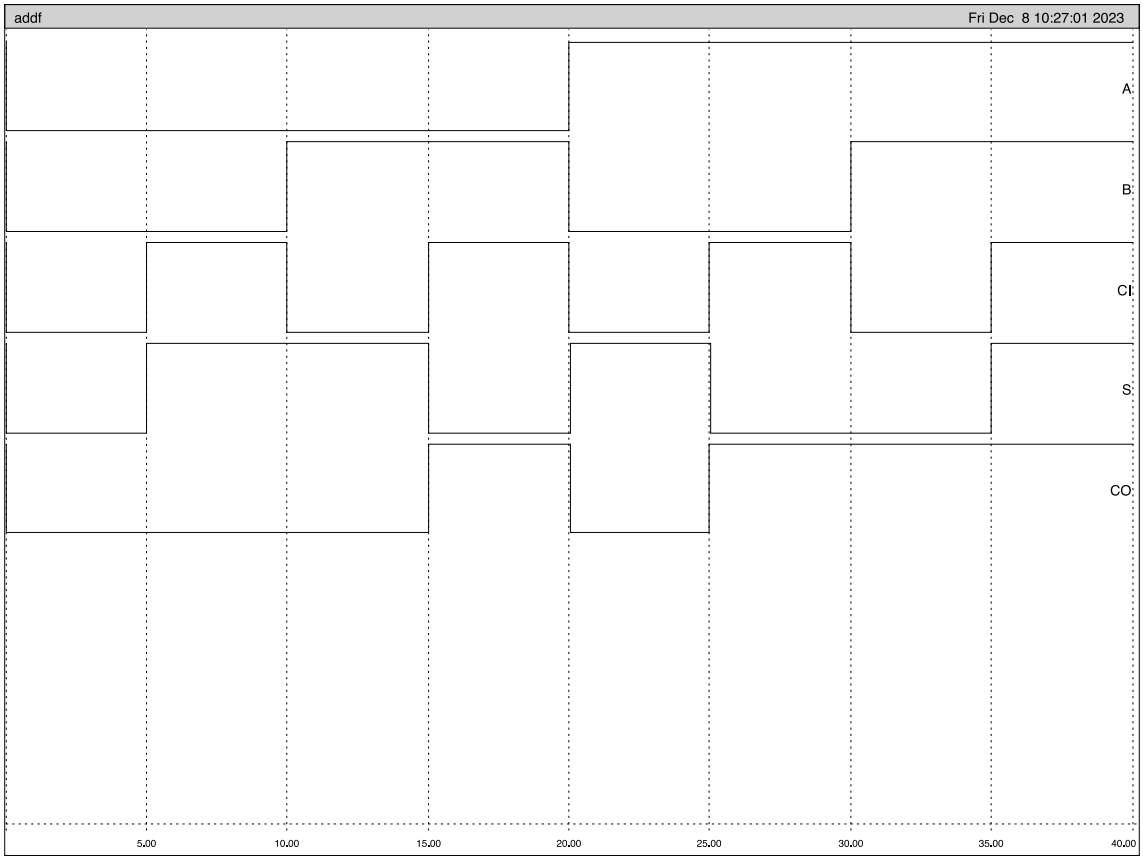
- Switch-level simulation with `IRSIM`
- Electrical modeling with `ngspice`

3.1 IRSIM

IRSIM performs switch-level simulations of circuits. It models transistors as switches, with capacitance to estimate critical path delay. It uses command files that provide digital inputs and computes the digital output value.

For brevity, here are the key simulations. More complete explanations can be found in the comments of the simulation command files.

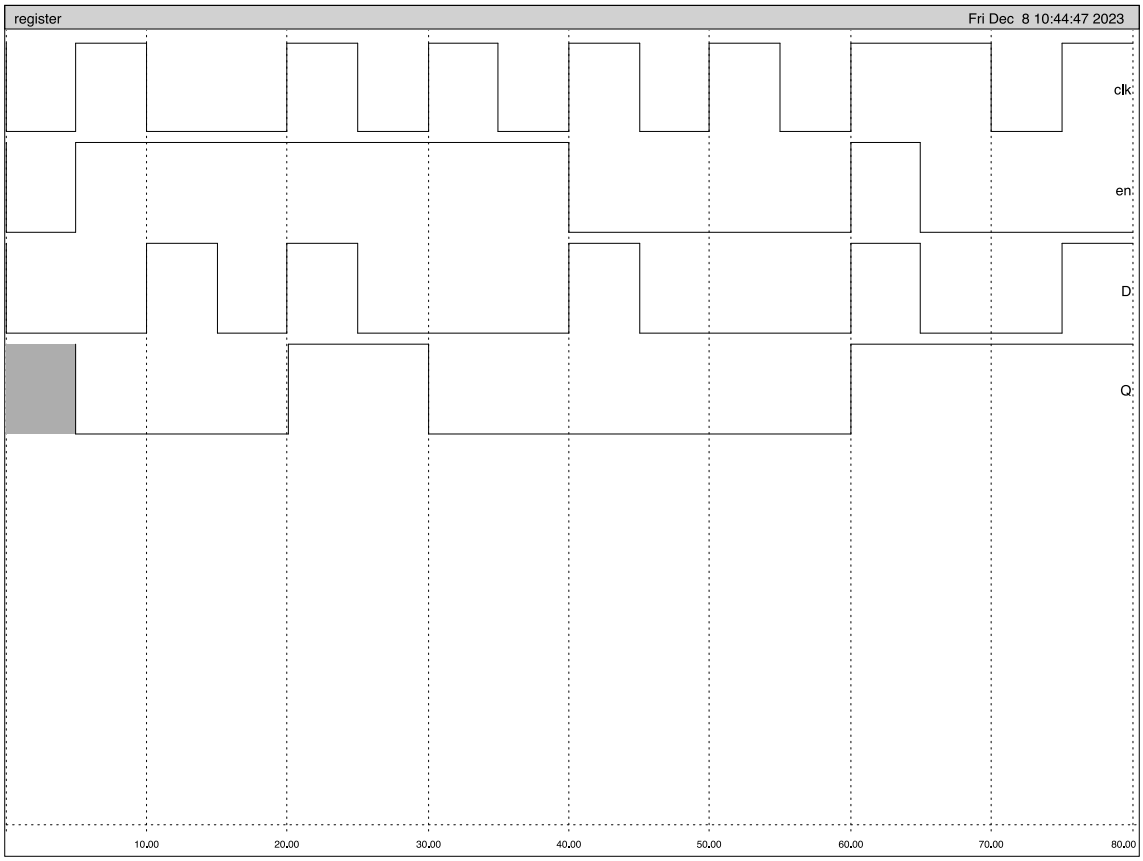
Full Adder



The full adder simulation tests all combinations of inputs and asserts the correct outputs.

S is asserted if an odd number of the inputs are high, and C_{out} is asserted if two or more inputs are high.

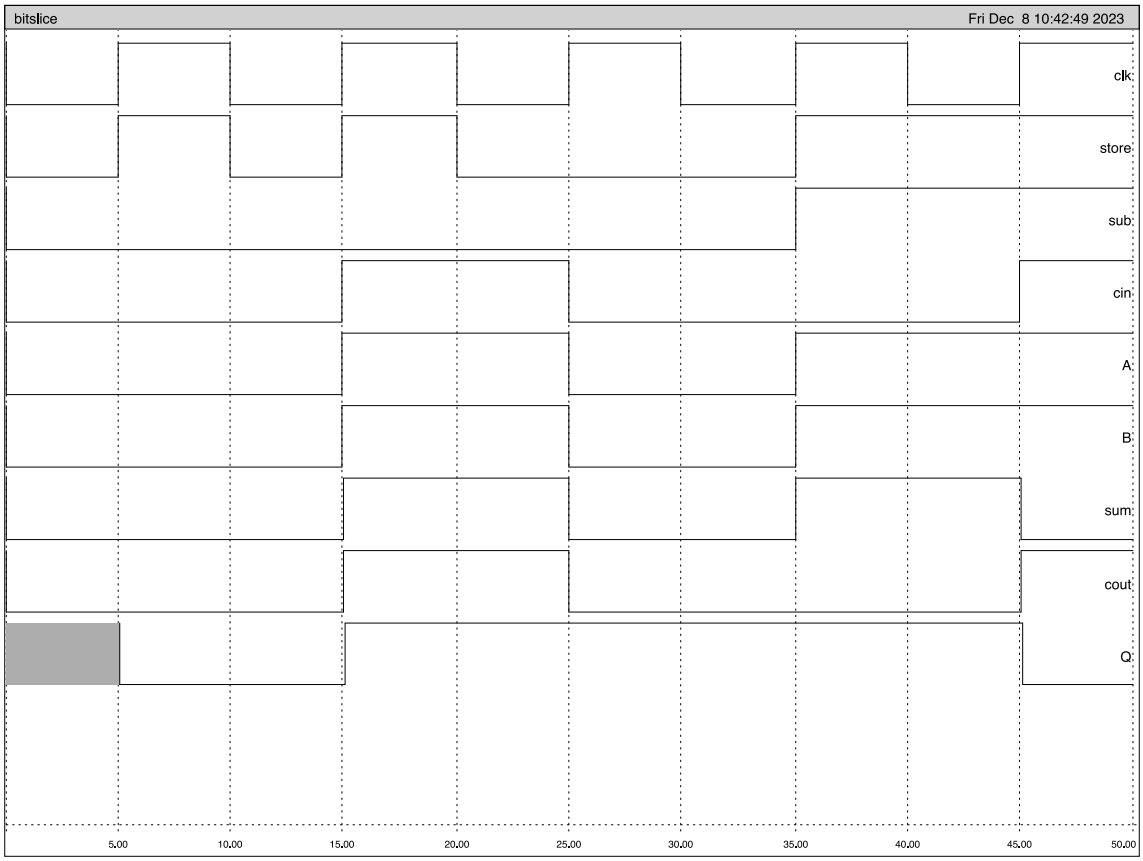
Register



The register simulation demonstrates multiple behaviors:

- The value of Q is invalid until the first rising edge of the clock.
- The value of D is only captured at the rising edge of CLK or EN, and only if the other is already asserted.
- Otherwise, the value of Q is kept stable indefinitely.

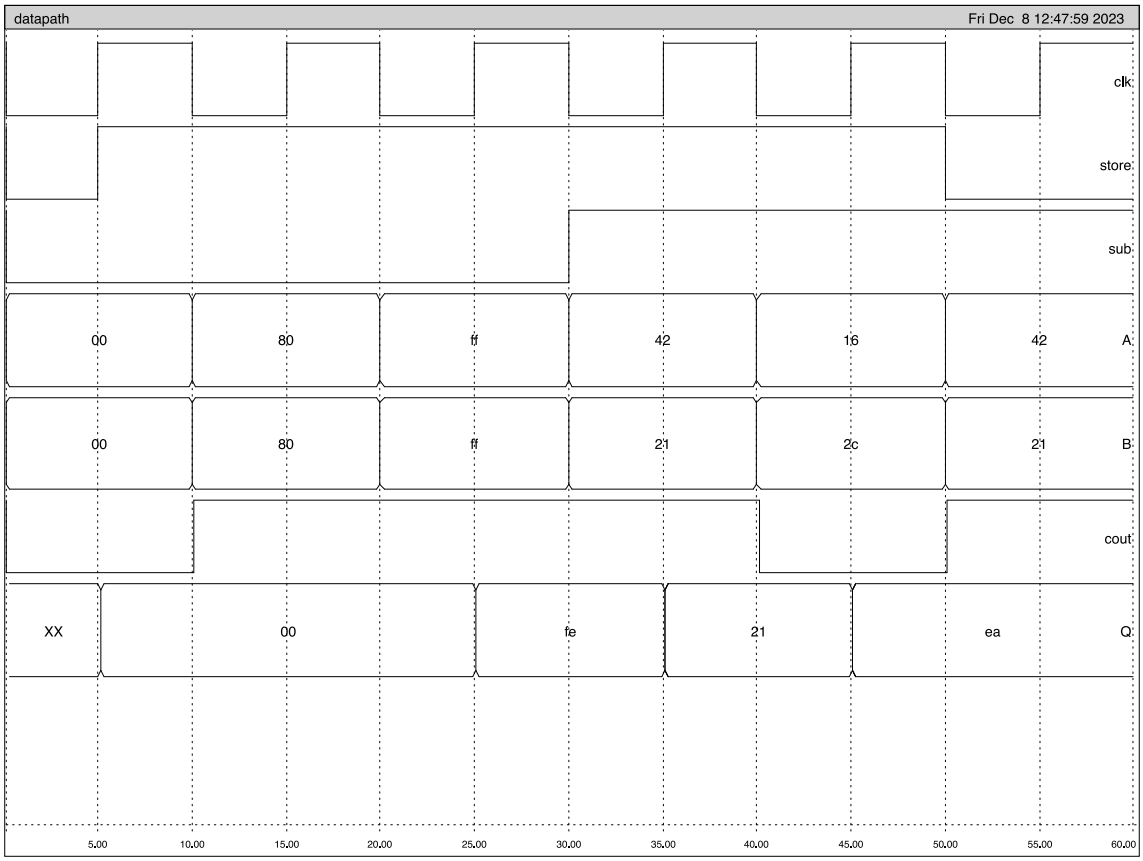
Bit Slice



The bit slice simulation tests these things:

- The full adder acts as expected.
- The register will not take a new value if STORE is not asserted, but C_{out} will update.
- Subtraction works, and the value C_{out} will be the complement of borrow.

Data Path



The data path simulation extends the bit slice simulation to a full eight bits.

It tests these operations:

- 128 + 128 = 0 carry 1
- 255 + 255 = 255 carry 1
- 66 - 33 = 33 borrow 0 (carry 1)
- 22 - 44 = (-128 + 106) borrow 1 (carry 0)

3.2 ngspice

ngspice is a mixed-level circuit simulator. It simulates the design as an analog circuit, so it can be used to estimate speed and power consumption. It works by reading a

SPICE deck, a text file containing directives and a netlist, and performing the simulation specified.

To test circuits, a test bench SPICE deck is written. This deck connects the circuit to specially created voltage sources to apply specific values, simulate the circuit, and evaluate how the outputs change.

The two key factors of a CMOS circuit are the critical path delay and voltage-transfer characteristics.

The critical path delay measures the worst-case response time of the circuit, which constrains the maximum speed of the device.

The voltage-transfer characteristics determine the voltages levels at which the values switch, and the voltage the device will output for a given input level. This is important for integrating the circuit within a larger design, and calculating the noise margins that design can withstand.

ngspice was used early in this project to test some early gates, but otherwise wasn't used until the final timing analysis.

4. Layout

The layout is the meat of the design. Here, we take our circuit designs and create physical geometry that implements it.

The process is as follows:

1. Create a floor plan, which shows where the gates will be placed.
2. Draw stick diagrams for the gates that will be used.
3. Implement the stick diagrams as cells in `magic`.
4. Perform LVS, verifying the gates match the schematic they're based on.
5. Integrate the gate cells into a hierarchical structure, and adding material to connect the circuit together, and designating areas as ports.
6. Perform LVS on the integrated design.

4.1 Floor Plan

The first step is the floor plan, determining the position of gates.

This can be an incredibly challenging process, but the requirements of the project impose constraints that simplify it. The objective is to create a data path by tiling bit slices, which naturally suggests a stack of long circuits.

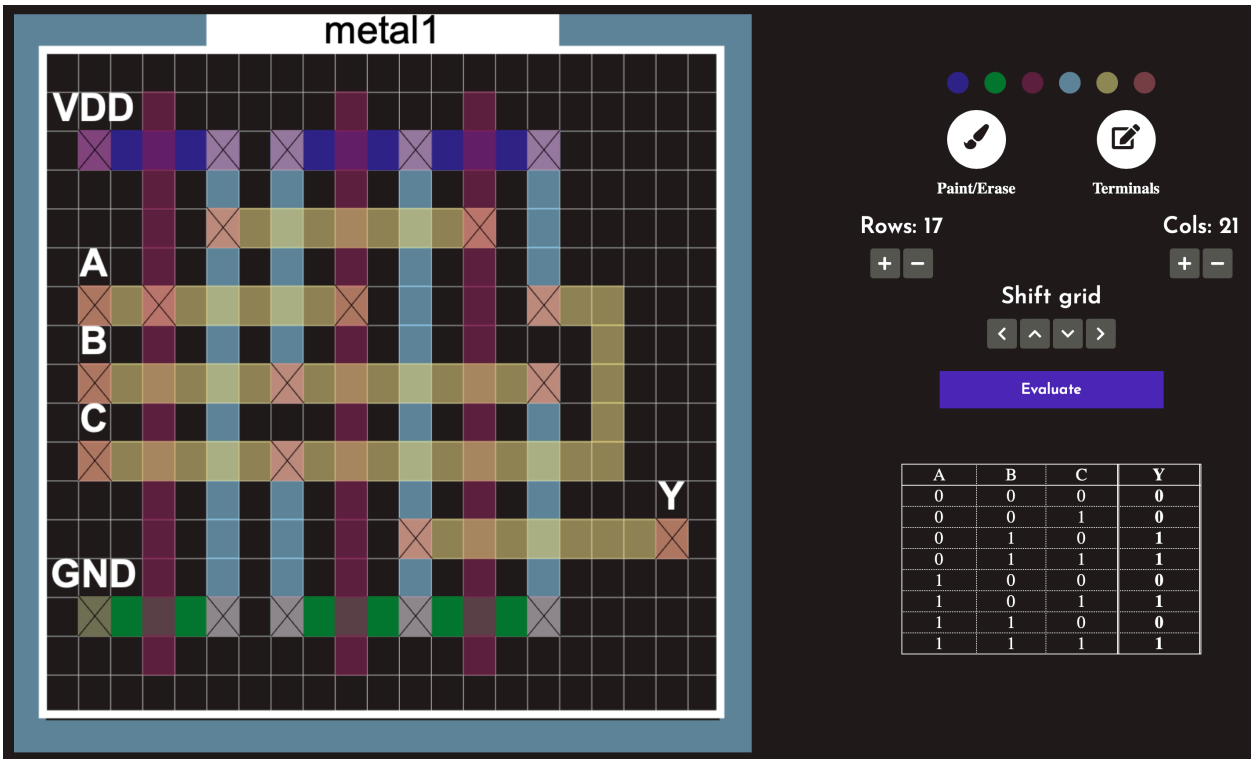
I figured the logical way to design it was to lay the gates out horizontally, with data inputs on the left and data outputs on the right. The control signals would run vertically across the stack, and the carry in and out would be placed at the bottom and top.

4.2 Stick Diagrams

The second step of creating a layout is to plan the actual gates. Stick diagrams ignore sizing, allowing us to focus on creating an efficient layout.

For this project, I used the stick diagramming tool at https://stixu.io/. This tool allowed me to draw stick diagrams and then simulate them. This greatly accelerated this project, as I was to verify that my planned layout would work correctly.

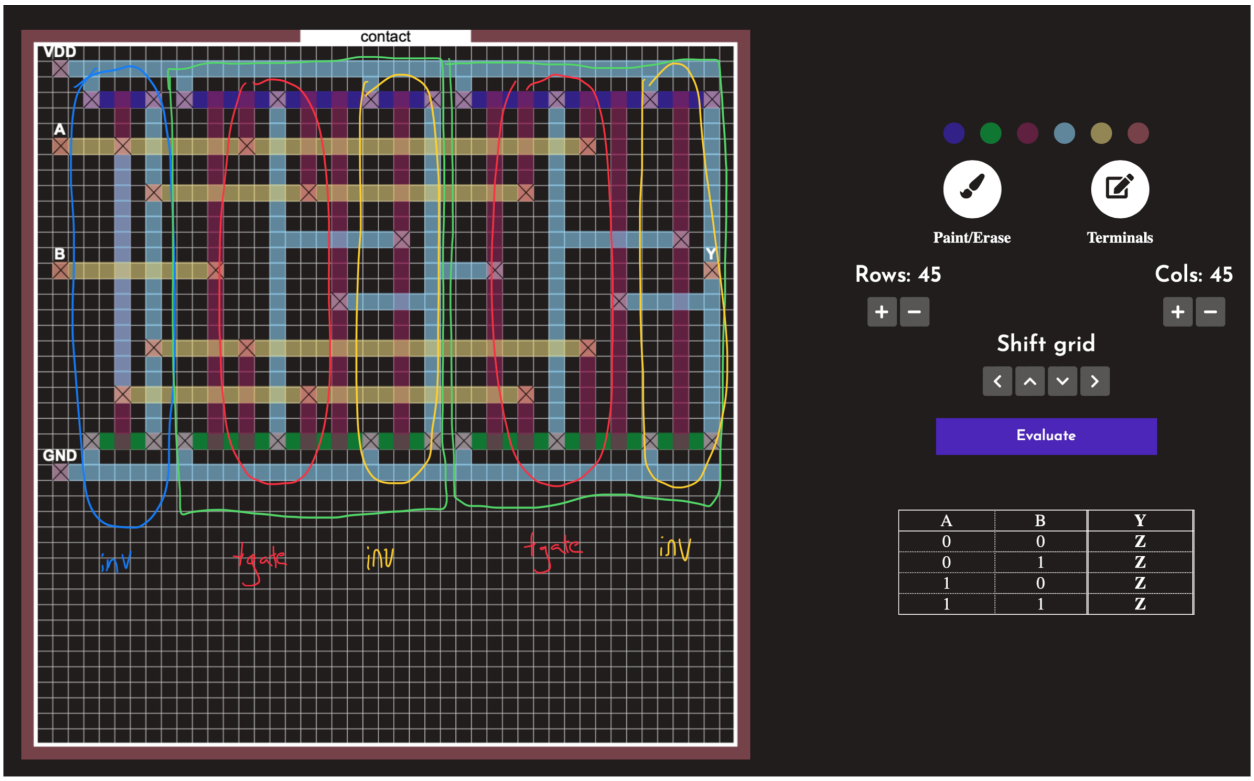
Multiplexer



A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

The first stick diagrams shows the multiplexer. There are two separate n and p-diffusions. The first forms an inverter, which calculates the complement of select (A on the diagram). Second is two transmission gates, with the enable and complement flipped for A, so that A is enabled when select is not asserted.

D Flip-Flop



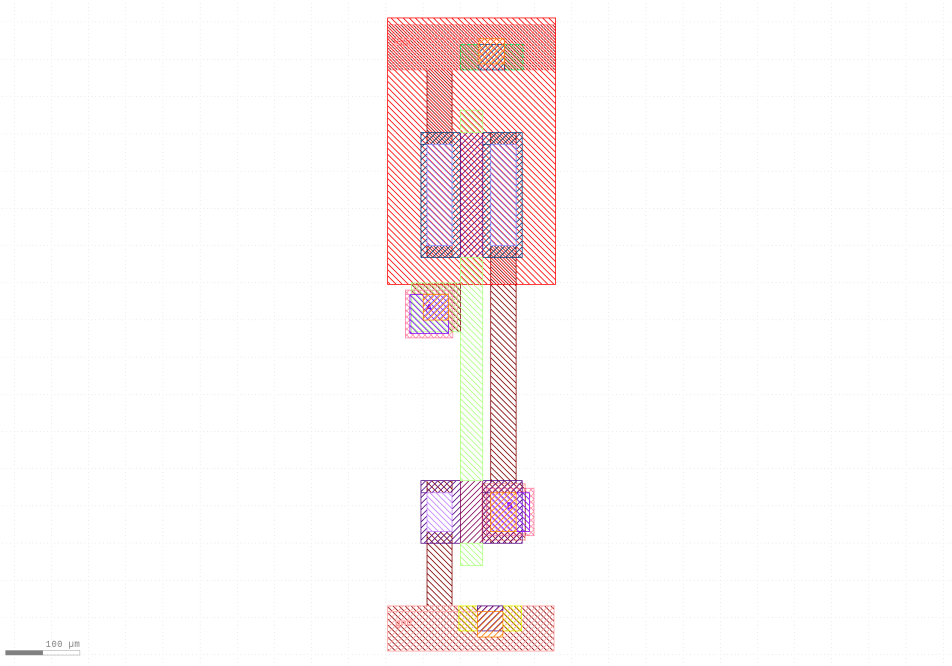
The second stick diagram has three diffusion pairs. First is an inverter, which gets the complement of the clock (A on the diagram). Second and third are transmission gates controlled by the clock and complement.

The simulation shows the output to be floating (Z) in all cases. This is expected, as the D flip-flop takes a cycle to reset and stabilize. The stick diagram tool can only simulate a single cycle.

4.3 Magic Cells

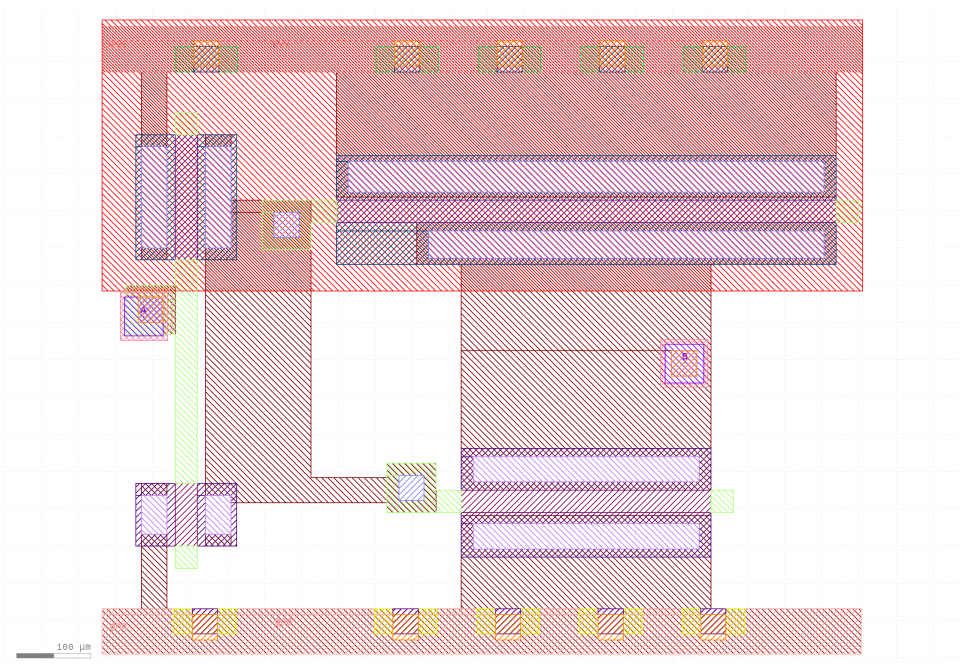
Note: the cells were created in `magic`, but rendered with KLayout.

Inverter



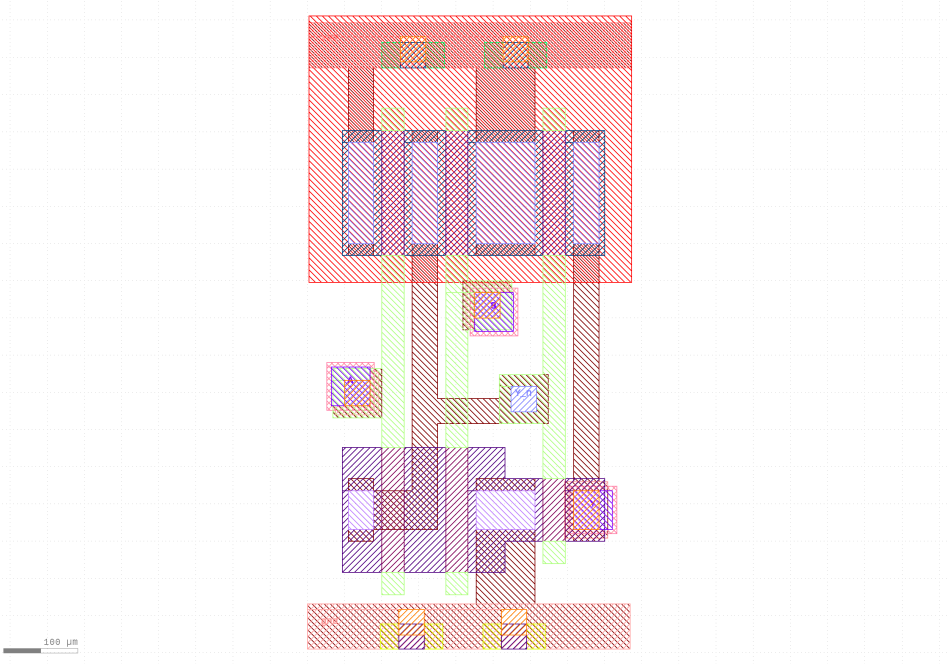
Dimensions: 1.13 µm × 4.26 µm
Area: 4.808 µm^2

Buffer



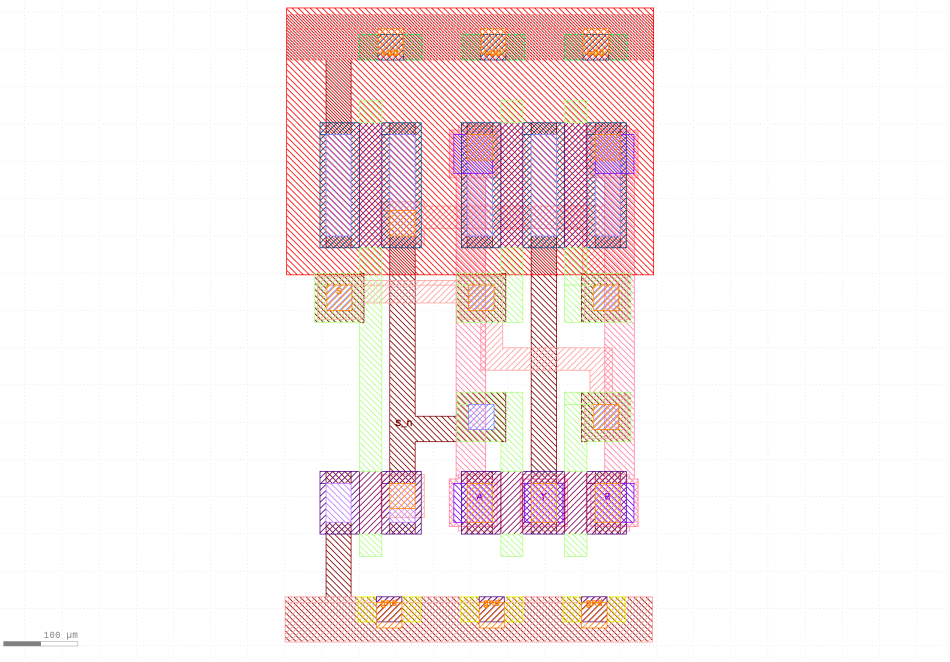
Dimensions: 5.12 µm × 4.26 µm
Area: 21.76 µm^2

AND Gate



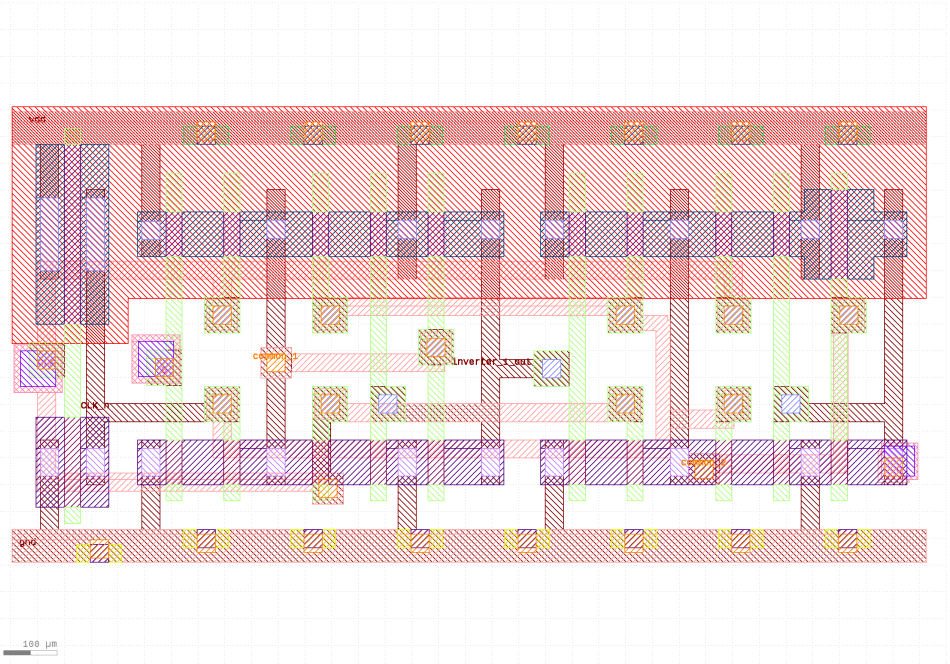
Dimensions: 2.18 µm × 4.26 µm
Area: 9.276 µm^2

Multiplexer



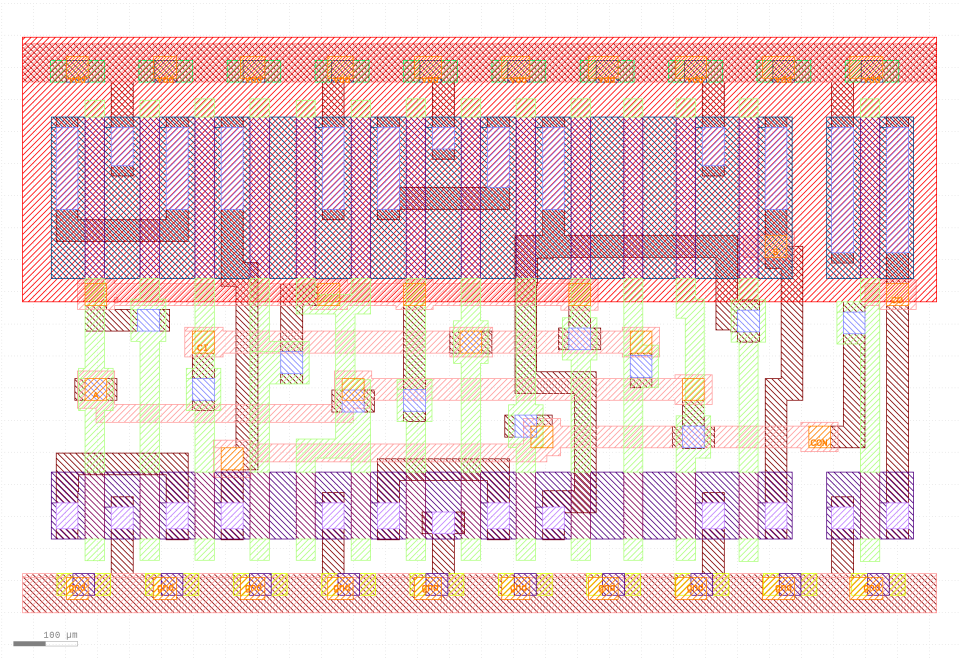
Dimensions: 2.48 µm × 4.26 µm
Area: 10.55 µm^2

D Flip-Flop



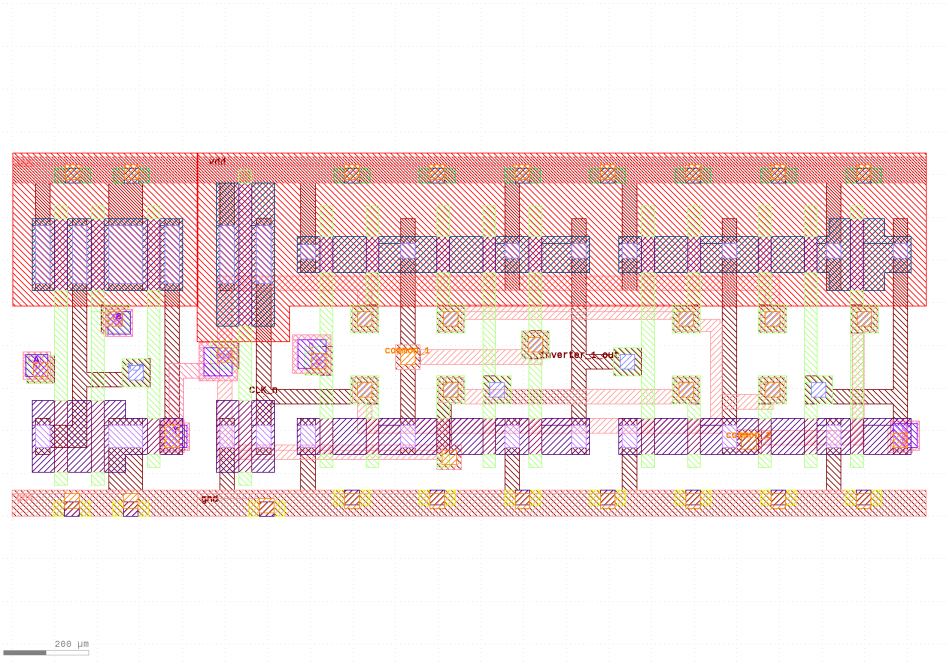
Dimensions: 8.55 µm × 4.26 µm
Area: 36.38 µm^2

Full Adder



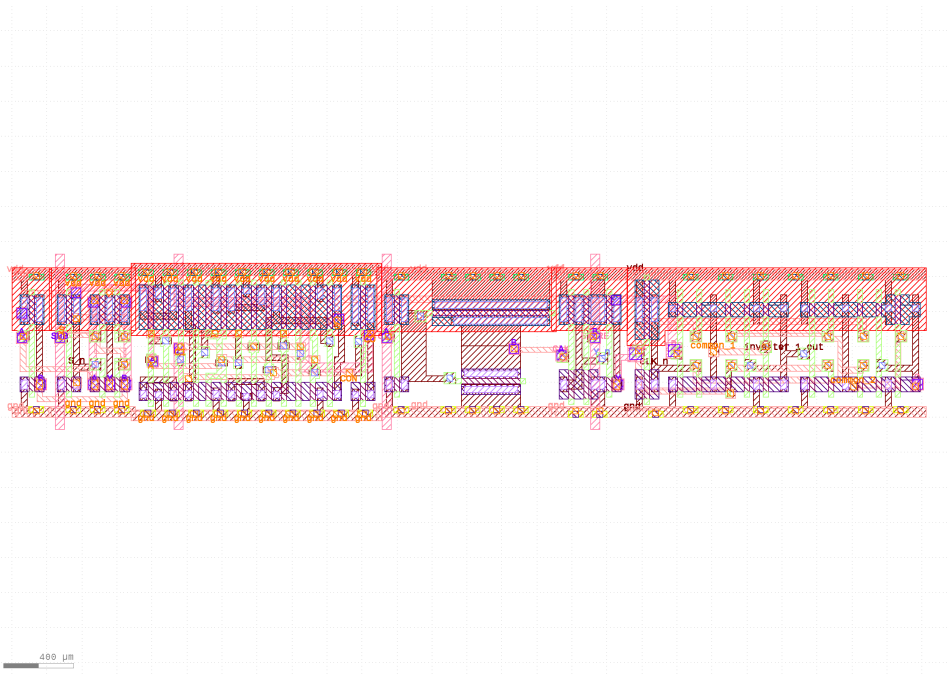
Dimensions: 7.15 µm × 4.49 µm
Area: 32.05 µm^2

Register



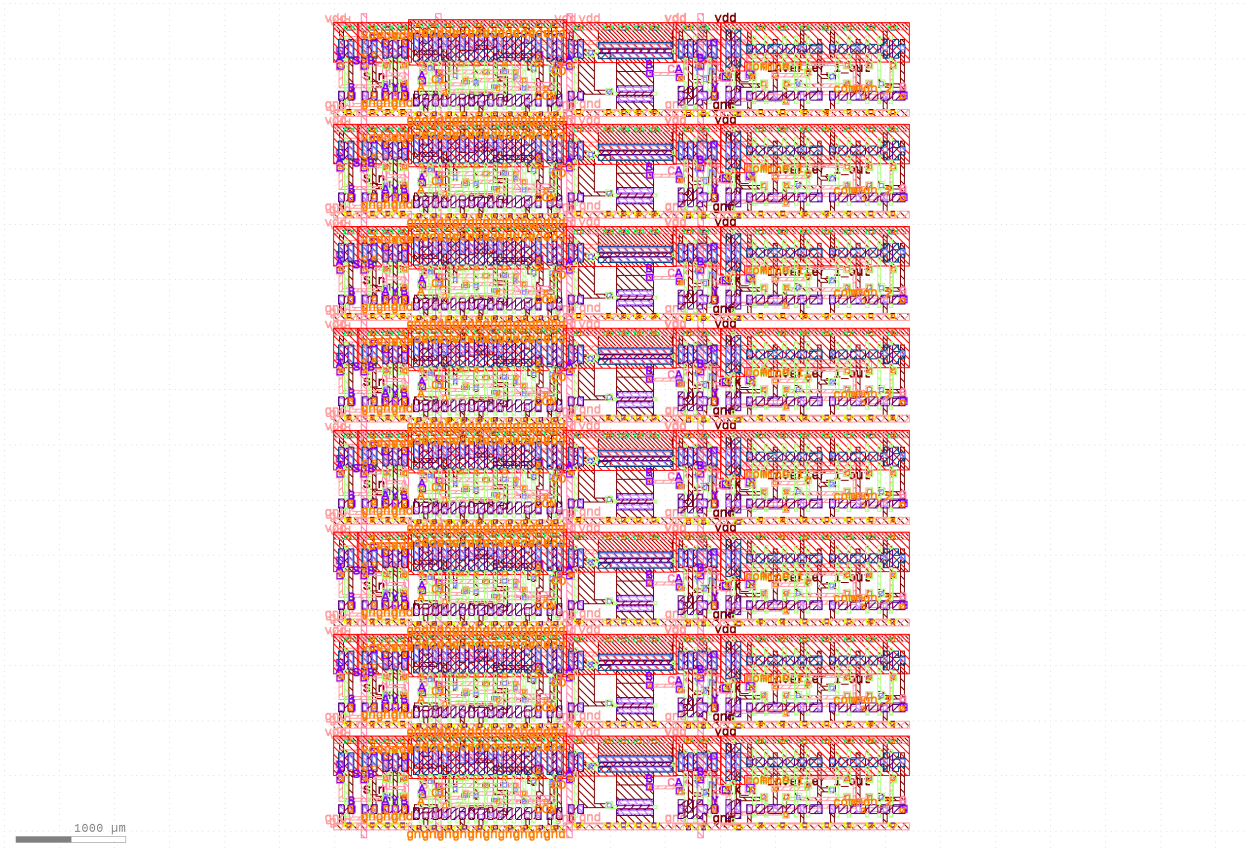
Dimensions: 10.72 µm × 4.26 µm
Area: 45.61 µm^2

Bit slice



Dimensions: 26.12 µm × 5.00 µm
Area: 130.6 µm^2

Data path



Dimensions: 26.12 µm × 37.34 µm
Area: 975.3 µm^2

4.4 LVS

All cells passed their LVS checks, meaning the layout-extracted netlist and schematic were functionally equivalent. The outputs are in the `netgen/` directory.

I had a major issue where leaf circuits, which didn't integrate any other circuits in their schematic, always failed to pass LVS. Dr. Stine pointed out that I was using the VDD and GND symbols from `xschem`, which were placed in the netlist as global nets. `netgen` couldn't determine that they were equivalent to the VDD and GND net created by `magic`, and so couldn't confidently determine if the circuits were the same. This was solved by switching to using lab pins in the schematics for VDD and GND.

5 Design Analysis

5.1 Size and Area

The bit slice is 26.12 µm × 5.00 µm, using 130.6 µm^2 of space. This could be significantly reduced by removing the buffer. The buffer is used to delay the clock reaching the D flip-flop, because the `irsim` simulation showed that the register would capture the output of the adder before it had settled.

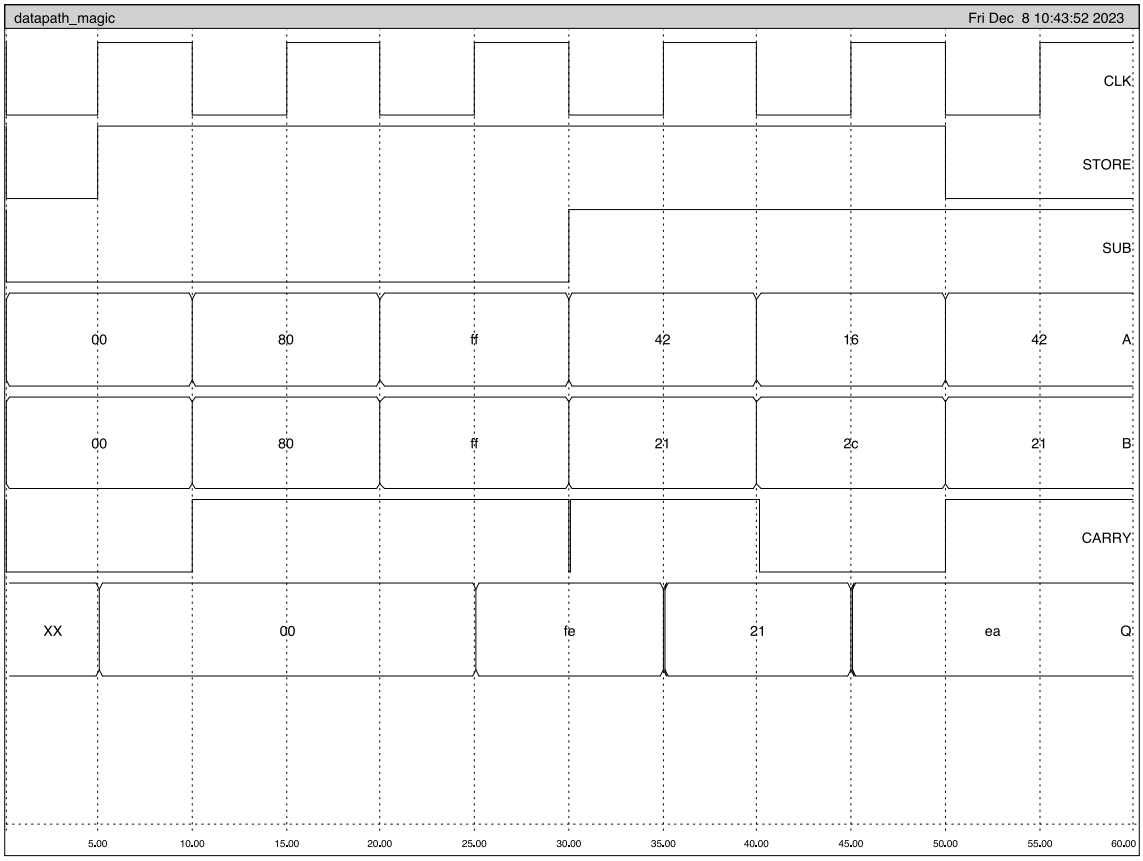
In a proper design, this would be handled by determining how long the input data must be valid before the rising edge of the clock, called the *setup time*.

The bitslice was meant to be stacked, and is designed to overlap slightly. This allows the vertical-running control signals to be automatically joined. Thus, an 8-bit data path occupies 26.12 µm × 37.34 µm, and uses 975.3 µm^2 of space.

5.2 Functionality

The `ext4mag` tool extracts outputs from a `magic` cell, including a sim netlist for `IRSIM`. I used this to reuse the command file to validate the data path for the final layout.

The extracted sim netlist behaved identically to the schematic drawn in `sue`.



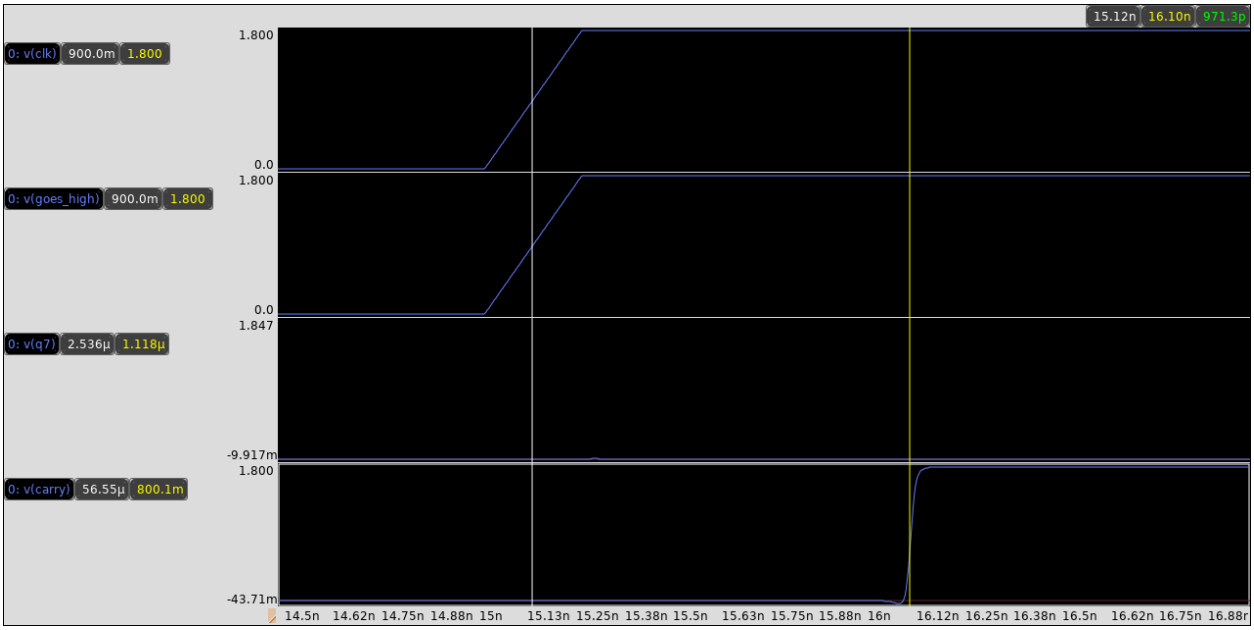
The command file used is a modified version of the one used for `sue`, because the port names are different between the two files. The functions tested are the same.

5.3 Critical Path

To test the critical path, I used `xschem` to create a testbench. The longest path through the circuit is subtraction, where every step outputs the carry signal (which is the complement of borrow).

To accomplish this, the testbench keeps all inputs low for some time. Then, it applies a value to the inputs (A = 1, B = 1) and asserts all control signals; SUB, STORE, and CLK. Then we measure how long it takes for the outputs Q7 and CARRY to rise.

The testbench waits 5ns with all inputs low. It pulses CLK high for 5 ns, and then low for 5 ns. Finally, it pulses all inputs (sans B_7) high. The output of Q7 and CARRY are plotted.



The midpoint of the inputs transition high takes place at 15.12 ns.

The midpoint of the CARRY transition high takes place at 16.10 ns.

The measured critical path delay is 971.3 ps. The rise time from 20% to 80% (0.36 V to 1.44 V) was 16.19 ps. This suggests this data path would function in excess of 100 MHz.

6 Artifacts

All files are hosted on GitHub in a public repository: https://github.com/mtayl14/Bitslice

6.1 Deliverables

Schematics (xschem)

- xschem/datapath.sch
  - xschem/bitslice.sch
    - xschem/inv.sch
    - xschem/mux21.sch
      - xschem/t_gate.sch
    - xschem/addf.sch
    - xschem/register.sch
      - xschem/and2.sch
      - xschem/buffer_fo4.sch
      - xschem/d_flip_flop.sch

Schematics (sue)

- irsim/datapath.sue
  - irsim/bitslice.sue
    - irsim/mux21.sue
      - irsim/t_gate.sue
    - irsim/addf.sue
    - irsim/register.sue
      - irsim/d_flip_flop.sue

Simulation Commands (irsim)

- irsim/addf.cmd
- irsim/bitslice.cmd
- irsim/d_flip_flop.cmd
- irsim/datapath.cmd
- irsim/datapath_magic.cmd
- irsim/mux21.cmd
- irsim/register.cmd
- irsim/t_gate.cmd

Cells (magic)

- magic/datapath.mag
  - magic/bitslice.mag
    - magic/inverter.mag
    - magic/mux21.mag
    - magic/addf.mag (provided)
    - magic/buffer_fo4.mag
    - magic/register.mag
      - magic/and2.mag
      - magic/d_flip_flop.mag

LVS Results (netgen)

- irsim/addf.out
- irsim/bitslice.out
- irsim/d_flip_flop.out
- irsim/datapath.out
- irsim/datapath_magic.out
- irsim/mux21.out
- irsim/register.out
- irsim/t_gate.out

6.2 Ancillary Scripts

LVS Runner (`netgen/do_lvs.sh`)

This script invokes `netgen` to perform LVS.

It checks that the necessary files exist, and will automatically invoke `ext4mag` if necessary. It is filled out with helpful info and error messages.

Here's an example invocation:

```
[mtayl14@angmar netgen]$ ./do_lvs.sh datapath
INFO:     Using default circuit name 'datapath'
ERROR:    Couldn't find 'datapath.spice' in the magic directory!
INFO:     Found 'datapath.mag' in the magic directory!
…
Final result:
Circuits match uniquely.
.
Logging to file "datapath.out" disabled
LVS Done.
```

IRSIM Runner (`irsim/run_irsim.sh`)

This script invokes `irsim` to run a simulation.

It checks that the necessary files exist, and is filled out with helpful info and error messages. It pulls the parameter file from `$PDK_ROOT`, so it doesn't need to be kept in the `irsim/` directory.

Here's an example invocation:

```
[mtayl14@angmar irsim]$ ./run_irsim.sh d_flip_flpo
ERROR:    Couldn't find IRSIM command file 'd_flip_flpo.cmd' in
this directory!
[mtayl14@angmar irsim]$ ./run_irsim.sh d_flip_flop
INFO:     Simulating circuit 'd_flip_flop' with 'irsim'...
```