Slide Set 10

Implementation Methods

Steve Wilton
Dept. of ECE
University of British Columbia
stevew@ece.ubc.ca

Based on Slides by Res Saleh, 2000
which were based on Slides by Mark Horowitz, Sanford U.

Overview

Now, we can get to the fun stuff. How do you implement a BIG chip? This will go hand-in-hand with the project (although you won't implement a BIG chip in the project, it will still be a chip, and many of the same issues will arise).

We will start by taking a step back, and looking at the different options available to a designer within VLSI.

Implementation Options

There are many constraints that might be placed on the cell design to make the fabrication or CAD tool problem simpler

Implementation fabrics range from

Field Programmable Gate Arrays (FPGA)

Chip are prefabricated, program fuses/antifuses to get logic.

Structured ASICs

Gates are prefabricated, customize metal to generate cell

Standard Cells (often called ASICs)

All cells have fixed height, (wiring maybe restricted to channels)

Standard Cells with Macros

Macros can be datapath and/or memory

Typically, datapath and memory blocks are generated automatically using datapath and memory compilers

True SOC Design:

Use macros (“cores”) from third parties, and put them together.

FPGA’s (what we used in EECE 353/379)

Advantages of FPGAs:

1. "Instant Manufacturability": reduces time to market
2. Cheaper for small volumes because you don’t need to pay for fabrication
   - means you don’t need to be a big company to make a chip
3. Relaxes Designers -> relaxed designers live longer!

Disadvantages of FPGAs:

1. Slower than gate arrays or custom chips
2. Can not get as much circuitry on a single chip

   Today: ~ 1 Million gates is the best you can do

   ~ 200 MHz is about as fast as you can get
3. For large volumes, it can be more expensive than gate arrays and custom chips

Gate Arrays / Structured ASICs

For an FPGA, the transistors and routing wires are prefabricated

- Customization done by setting configuration bits

Gate Array/Structured ASICs: Transistors are prefabricated, and possibly the first few metal layers

- After circuit design is done, need to fabricate one or more metal layers

Cheaper (perhaps) and faster to manufacture than standard cells since you need to customize fewer layers. But since you can't resize transistors or place them where you want, your circuit won't go as fast.

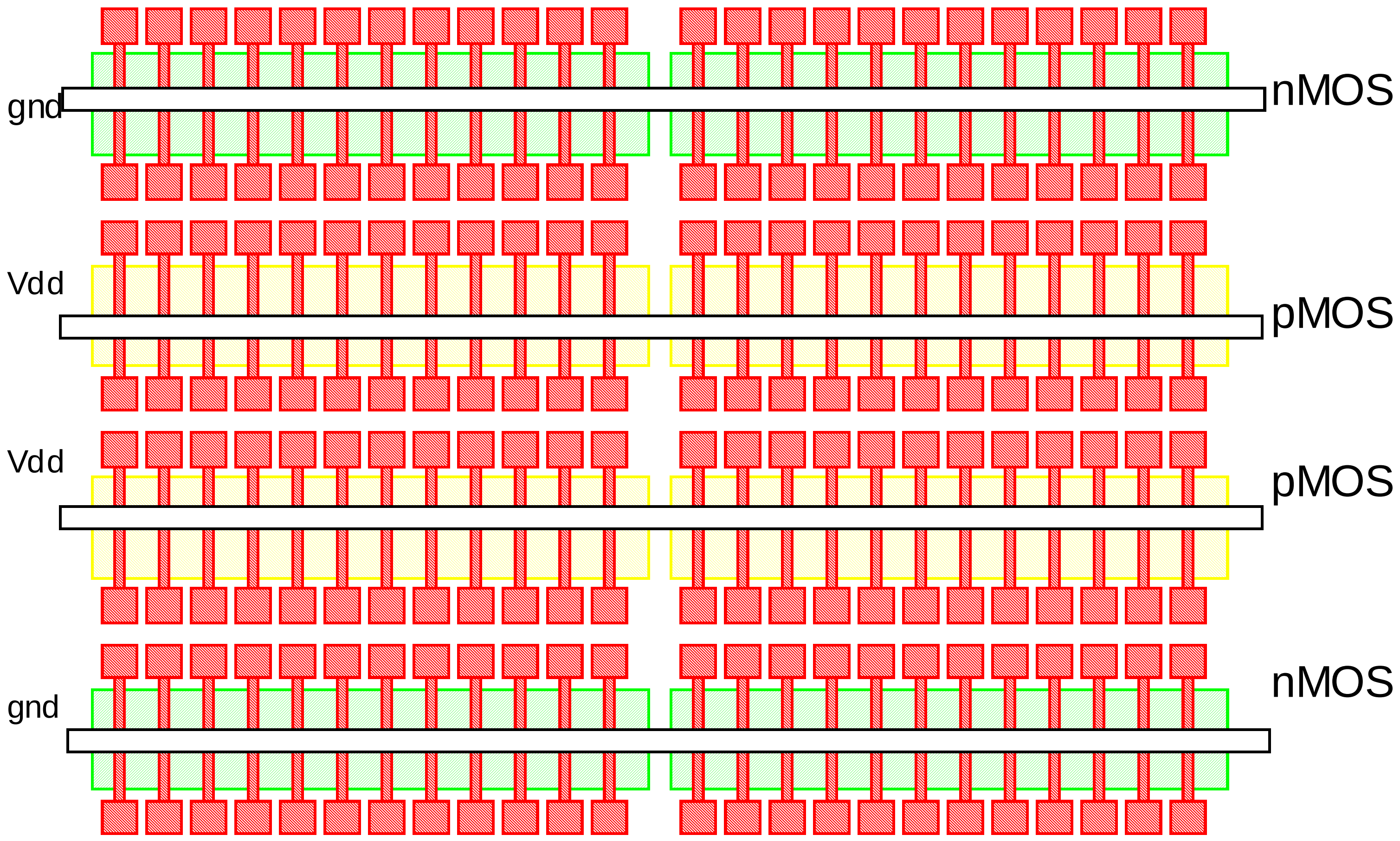
DEAD!

~~Gate Arrays: Base cell a transistor~~

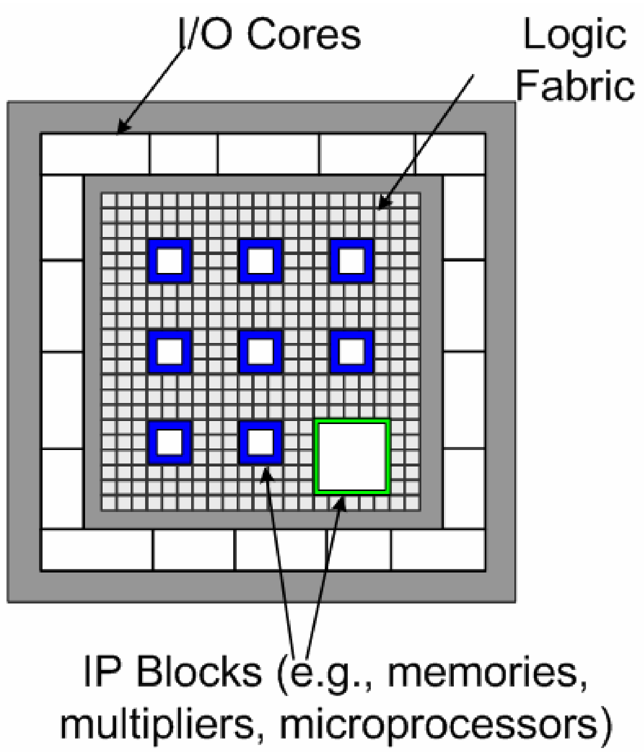
Structured ASICs: Base cell something larger

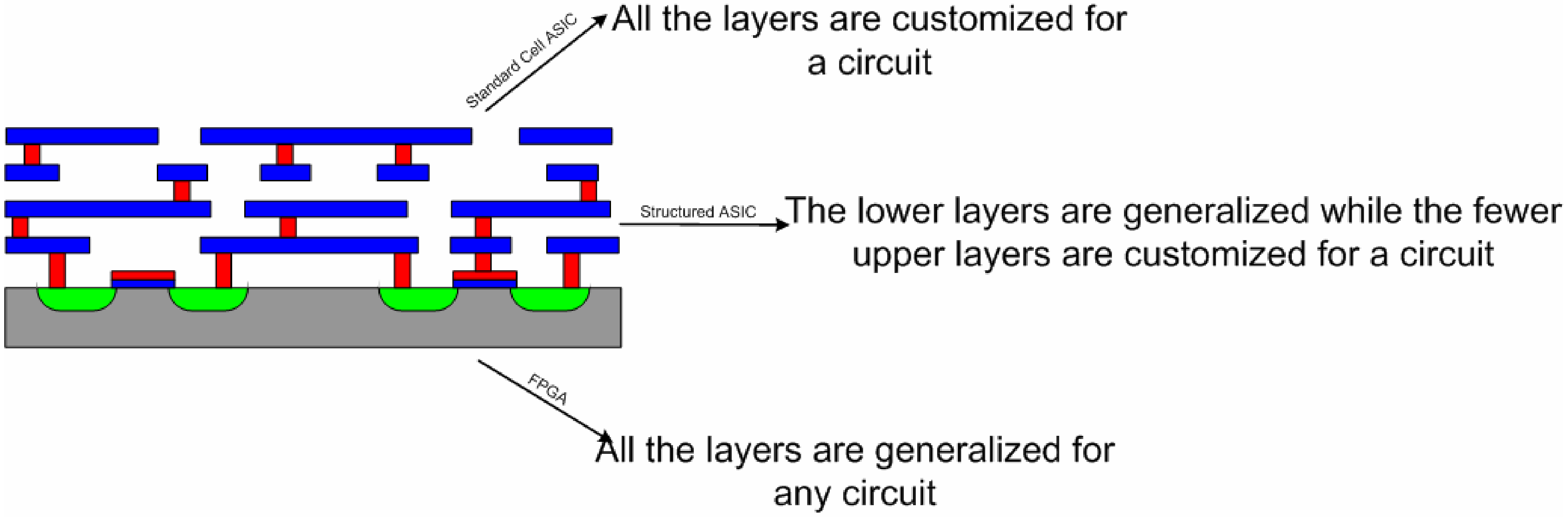
Very much alive!

Typical Gate Array Layout



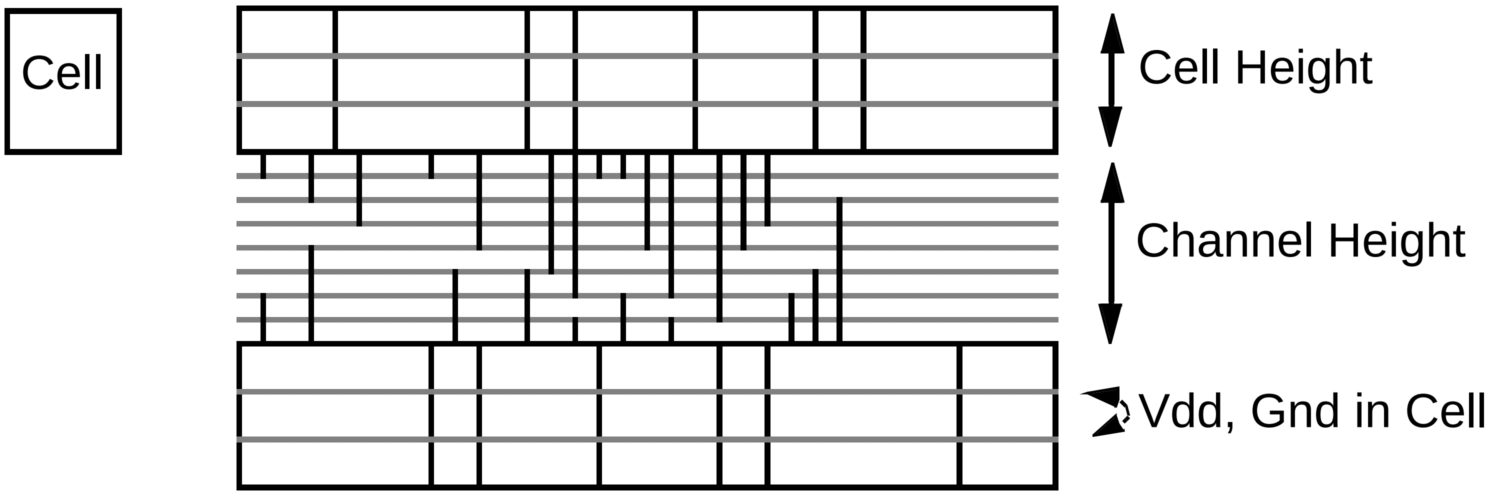
Structured ASICs

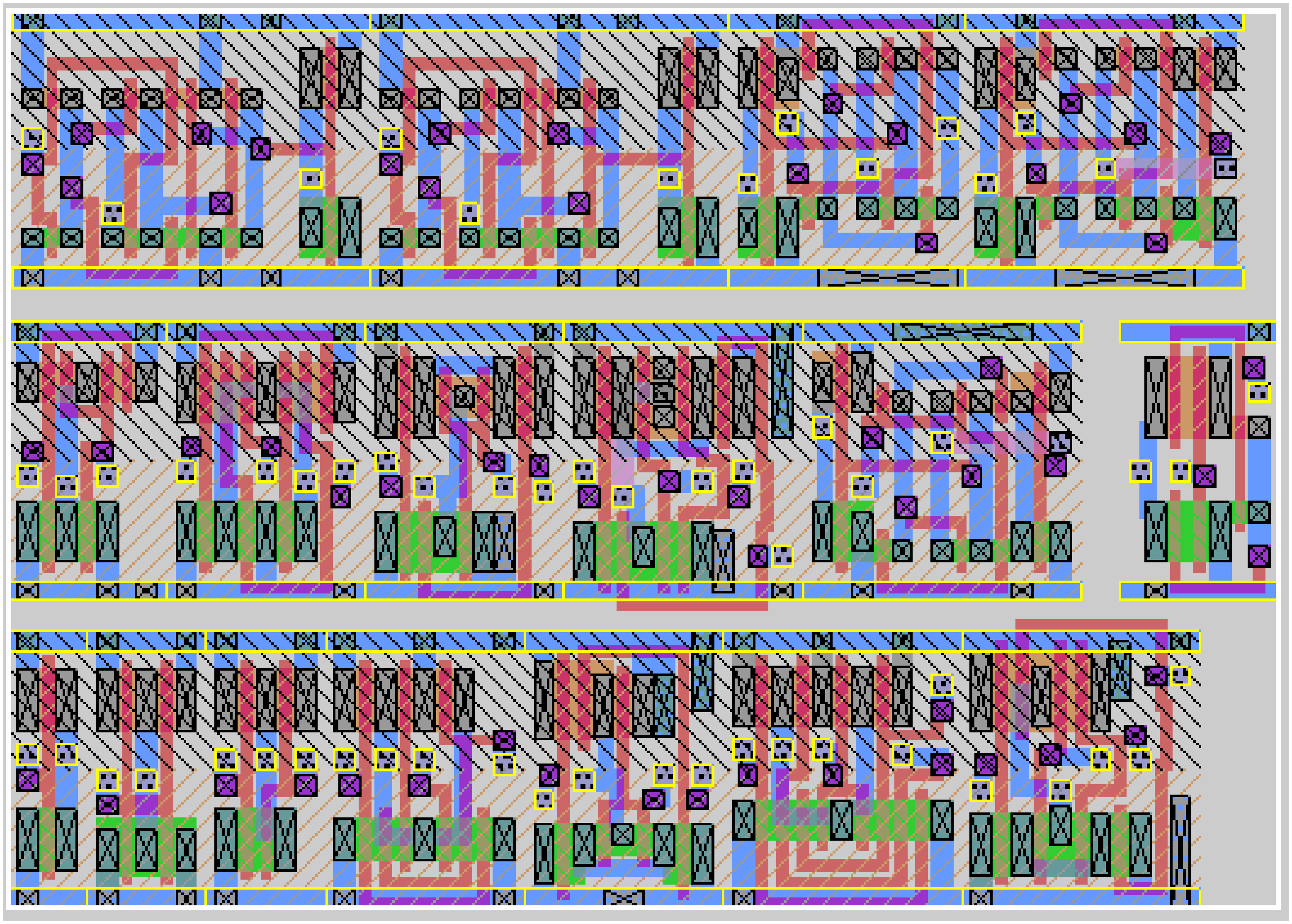


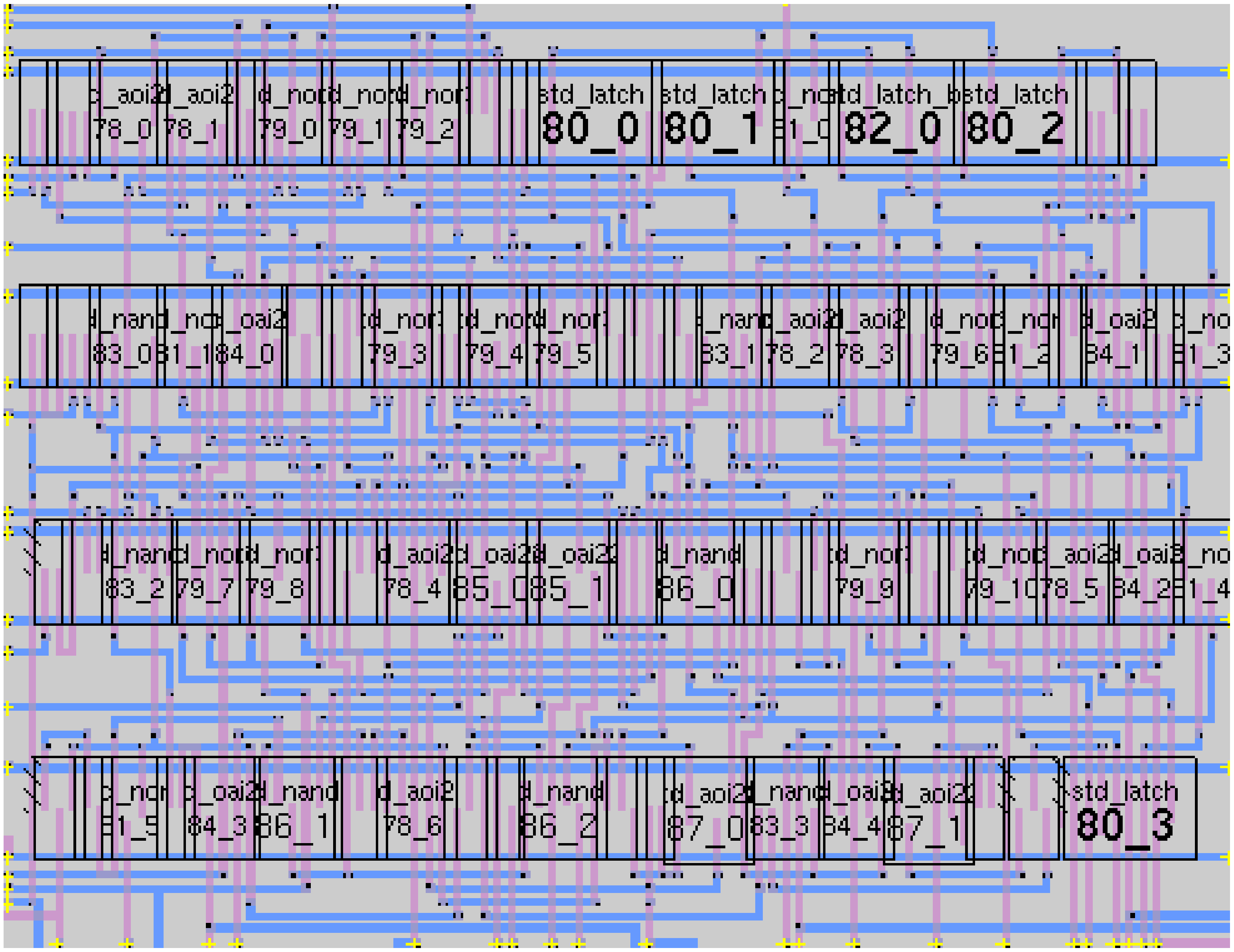

Standard Cell ASIC
All the layers are customized for a circuit
Structured ASIC
The lower layers are generalized while the fewer upper layers are customized for a circuit
FPGA
All the layers are generalized for any circuit

Standard Cells with Channel Routing

- Appropriate for all or part of a custom chip (defining all mask layers)
- All cells are the same height with abutting power and gnd connections
- Cells tiled into rows
- Rows of cells separated by routing. If M3 exists, maybe over-cell routing too.
- Channel height can be set after the routing, so the wires always fit

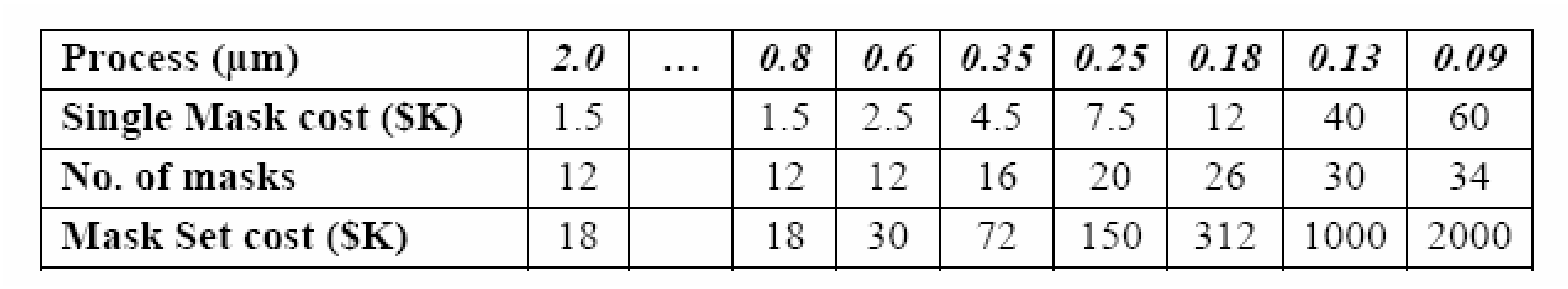






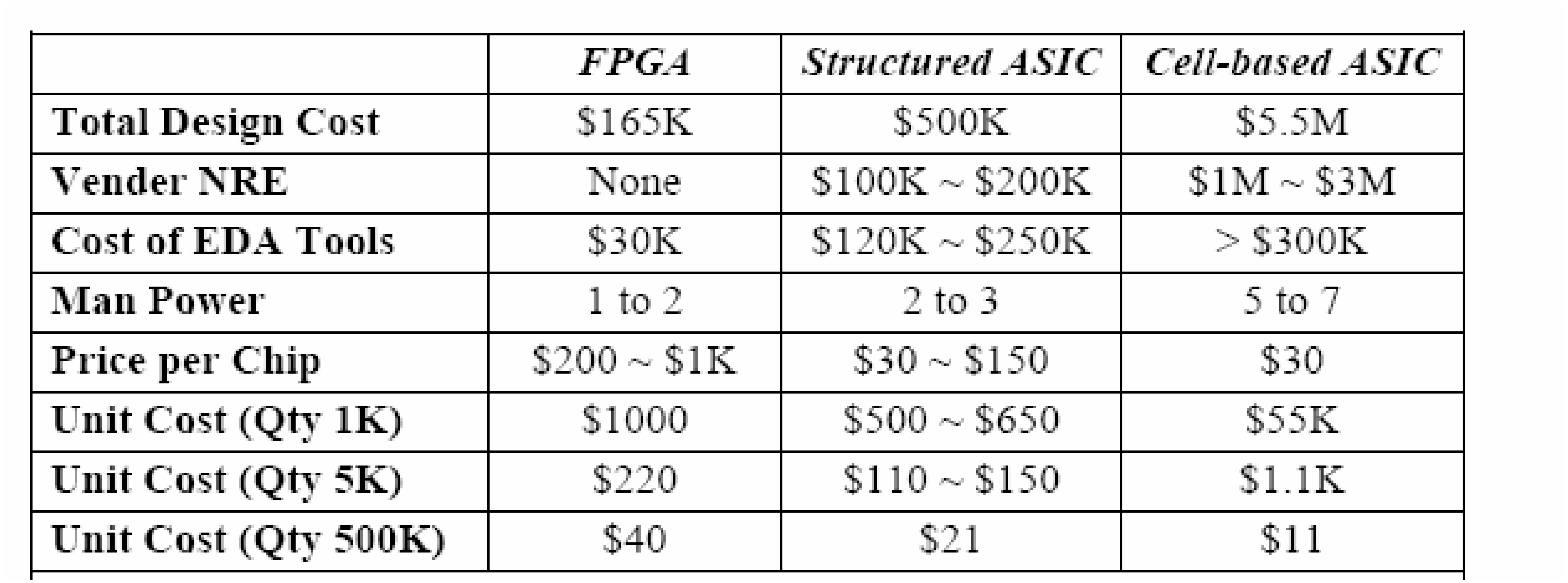
std_latch 80_0
std_latch 80_1
std_latch_b 82_0
std_latch 80_2
std_latch 80_3
85_0
85_1
86_0
86_1
86_2
87_0
87_1

Masks are expensive:



Process (μm)	*2.0*	...	*0.8*	*0.6*	*0.35*	*0.25*	*0.18*	*0.13*	*0.09*
Single Mask cost ($K)	1.5		1.5	2.5	4.5	7.5	12	40	60
No. of masks	12		12	12	16	20	26	30	34
Mask Set cost ($K)	18		18	30	72	150	312	1000	2000

Cost Comparisons:



	FPGA	*Structured ASIC*	*Cell-based ASIC*
Total Design Cost	$165K	$500K	$5.5M
Vender NRE	None	$100K ~ $200K	$1M ~ $3M
Cost of EDA Tools	$30K	$120K ~ $250K	> $300K
Man Power	1 to 2	2 to 3	5 to 7
Price per Chip	$200 ~ $1K	$30 ~ $150	$30
Unit Cost (Qty 1K)	$1000	$500 ~ $650	$55K
Unit Cost (Qty 5K)	$220	$110 ~ $150	$1.1K
Unit Cost (Qty 500K)	$40	$21	$11

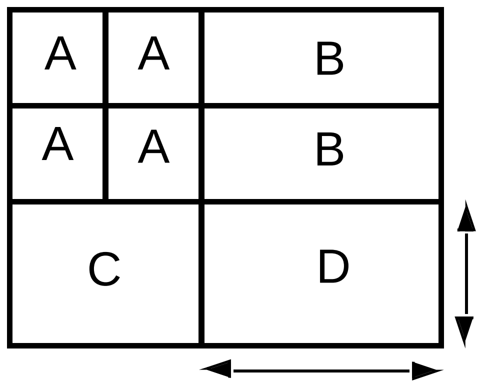
Standard Cells vs. Macros

Standard Cells: The logic is done in fixed height cells. The cells are assembled into rows, and the wires that connect the logic together is done in wiring channels that are outside the cells. The routing is usually done using CAD tools. In today's tech, with so many metal layers, often the routing is *on top* of the cells

'Snap-Together' Cells: Rather than having external channels for the wires, in this design style, the wires are contained in the cells. The wiring pattern is regular enough that the connection between cells is done by simply abutting the cells. This layout style is more restrictive that the standard cell style, since it supports a more limited wiring topology. Its advantage is that if the wiring can be made to fit this layout style, both the area and wirelength (capacitance) can be reduced.

‘Snap-Together’ Cells

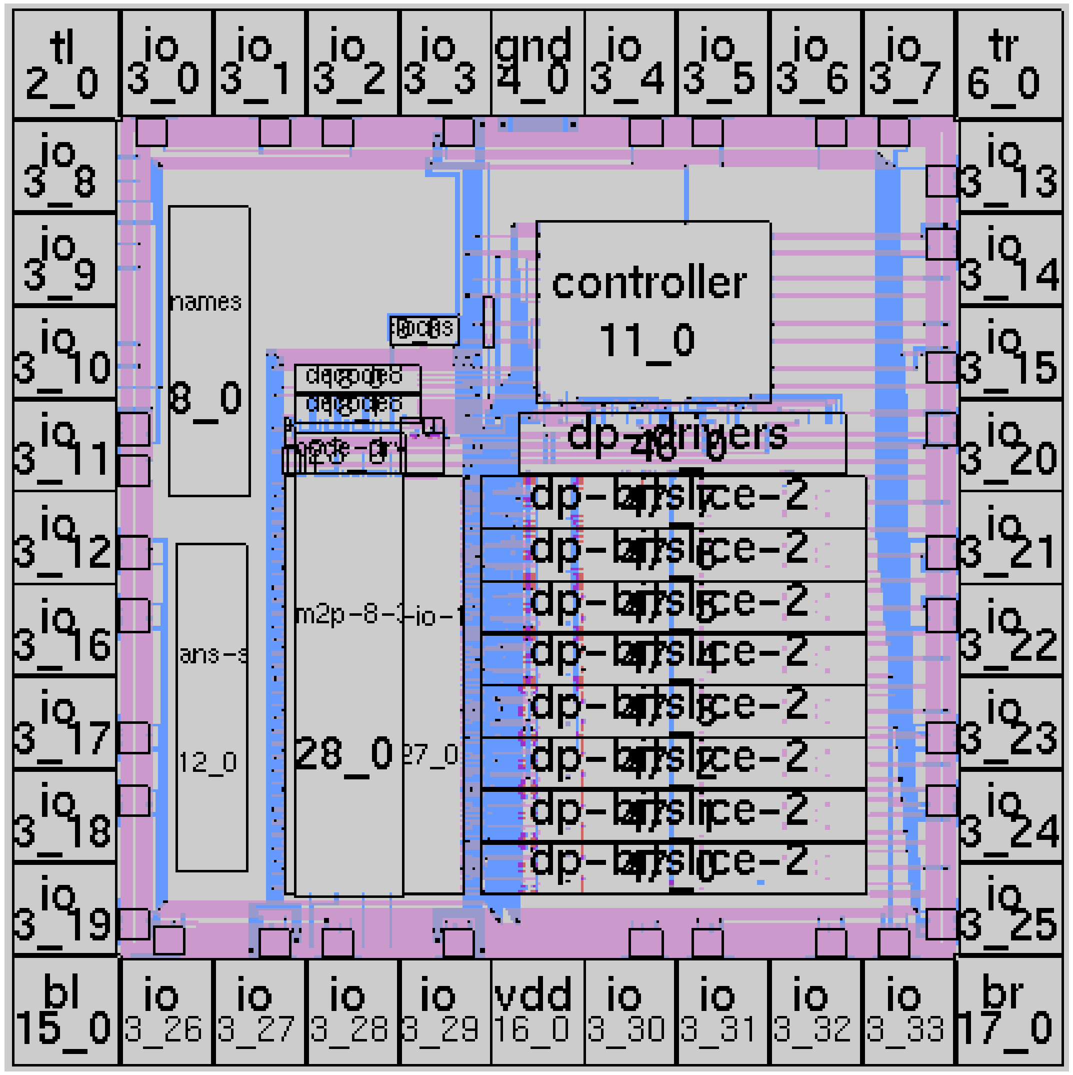
For high-density design blocks, we tend to compose them using ‘snap-together’ cells. The critical issue is ‘pitch-matching’ the cells (like std-cells, but requires matching in both dimensions, not just the height). Since we want them all to fit together, not only do we need to have the wire connect at the cell boundaries, but we also want the cells to be able to ‘tile’ the surface. That is, cells that connect together should have the size in the connecting edge. In this design style, smaller is not always better. You need all connecting cells to share height and width on edges.



Making D shorter would not help

Making D narrower would not help

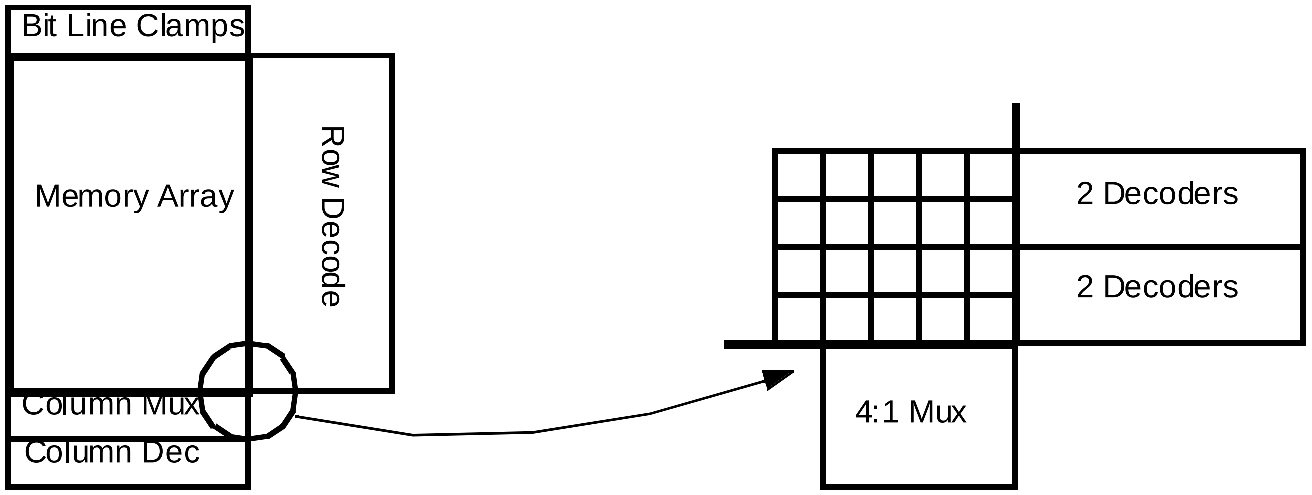
Chip Layout


tl 2_0
io 3_0
io 3_1
io 3_2
io 3_3
gnd 4_0
io 3_4
io 3_5
io 3_6
io 3_7
tr 6_0
io 3_8
io 3_9
io 3_10
io 3_11
io 3_12
io 3_16
io 3_17
io 3_18
io 3_19
io 3_13
io 3_14
io 3_15
io 3_20
io 3_21
io 3_22
io 3_23
io 3_24
io 3_25
names 8_0
ans-s 12_0
controller 11_0
dp-drivers 46_0
dp-bitslice-2
m2p-8-1
28_0
io-1
27_0
bl 15_0
io 3_26
io 3_27
io 3_28
io 3_29
vdd 16_0
io 3_30
io 3_31
io 3_32
io 3_33
br 17_0

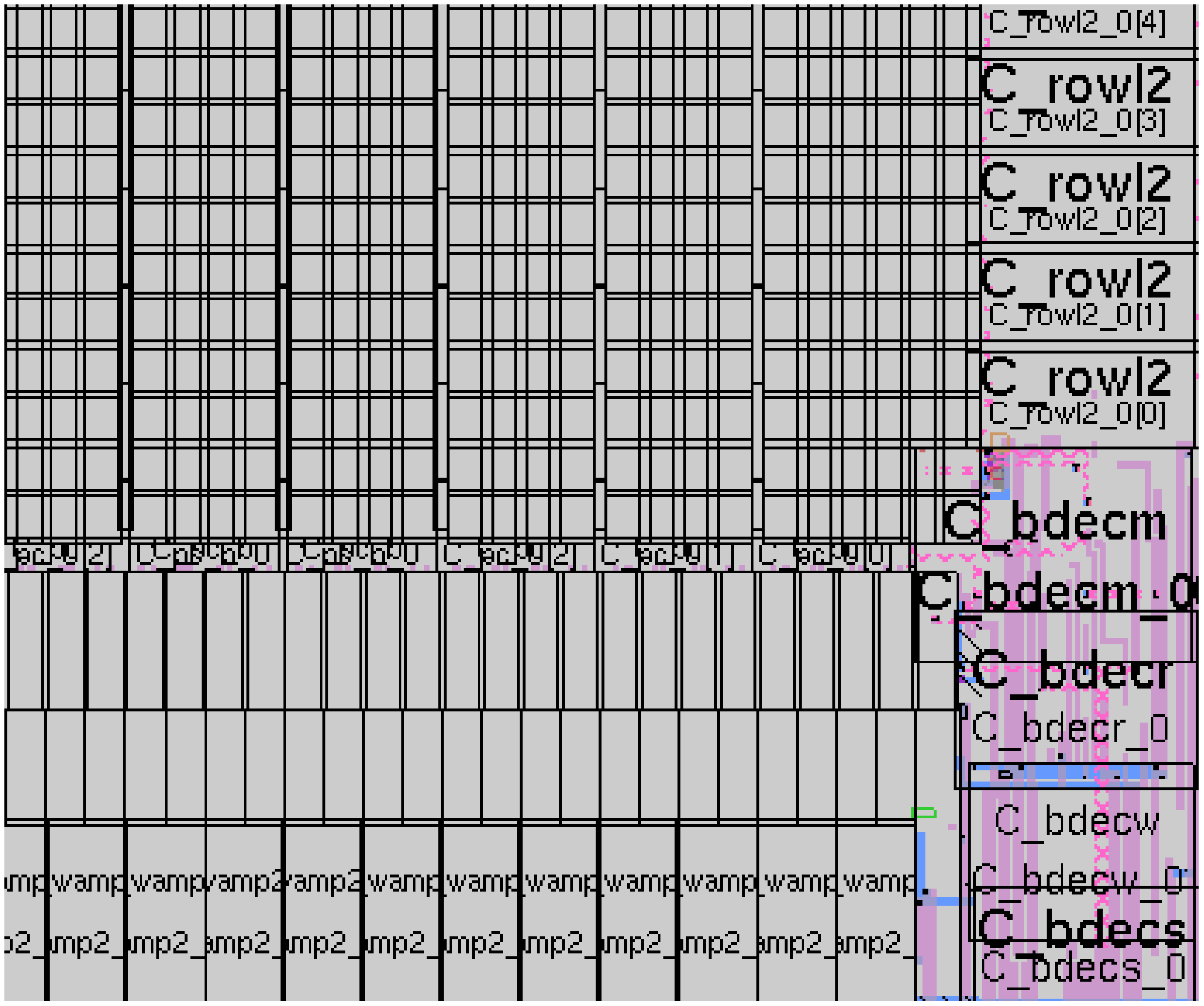
Abutting Cells

Two common examples of blocks that use these cells are regular arrays (usually for memory) and datapath (for dataflow design).

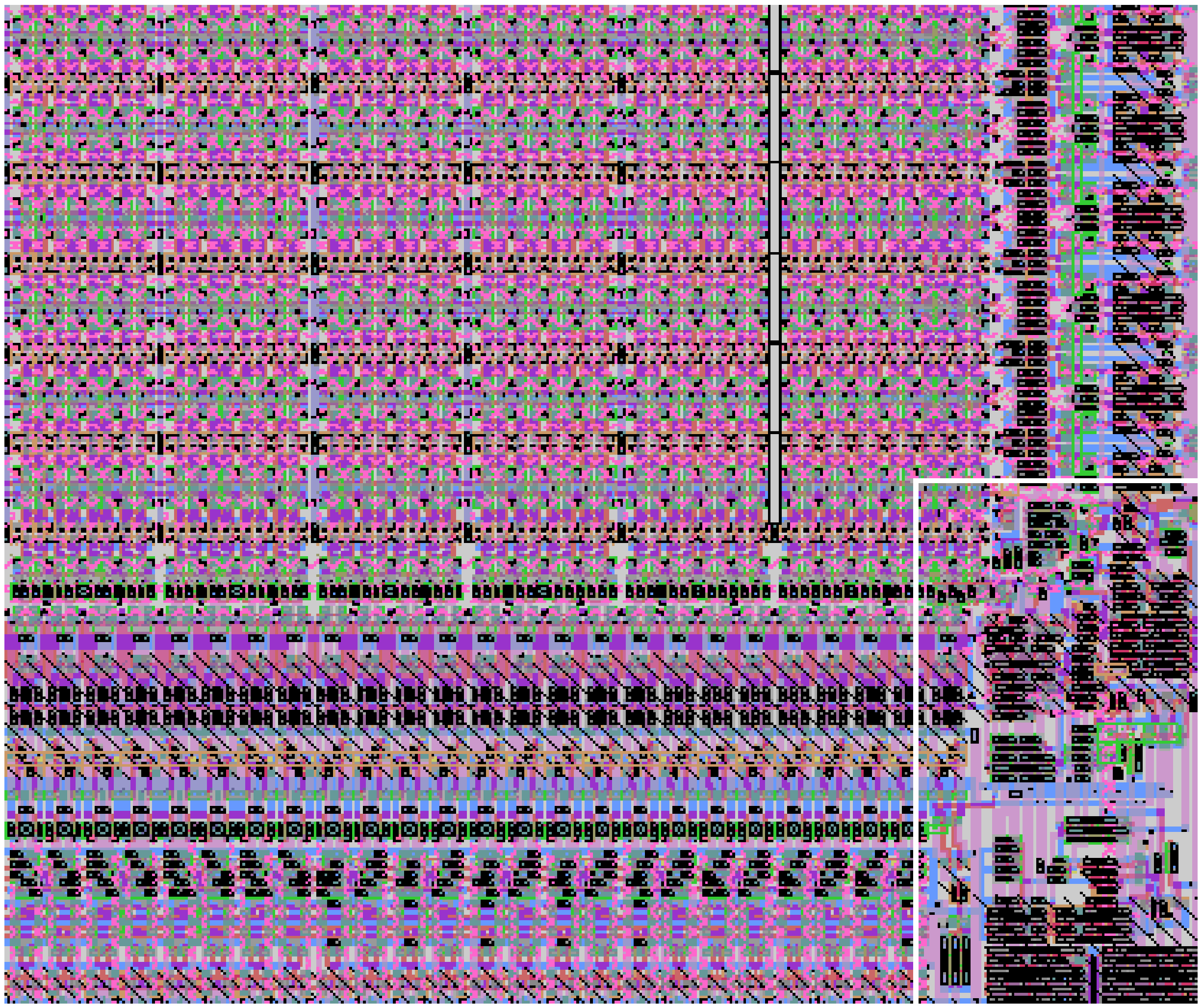
Example: In memory design, the core of the cells is a two dimensional array of bits. Since the communication between the cells is fixed, it is easy to embed the needed wires in the cells. Key here is to get all the edge decoder and mux cells to pitch-match to the small memory cells:



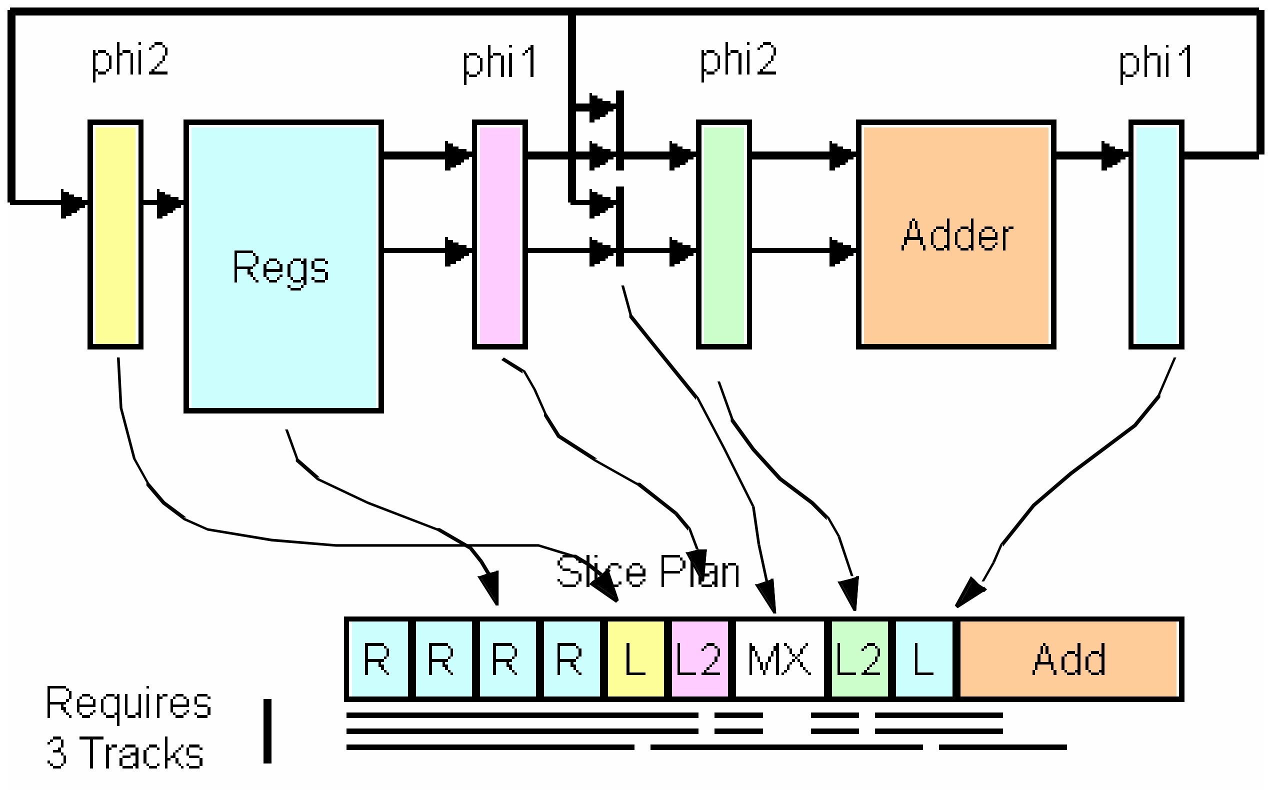
Memory Layout Example



Memory Layout Example



Datapaths

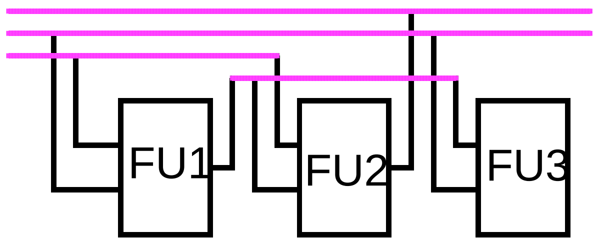


For Datapaths, we *could* use standard cells, but if we use a few specific datapath techniques, our implementation will be much more dense and fast

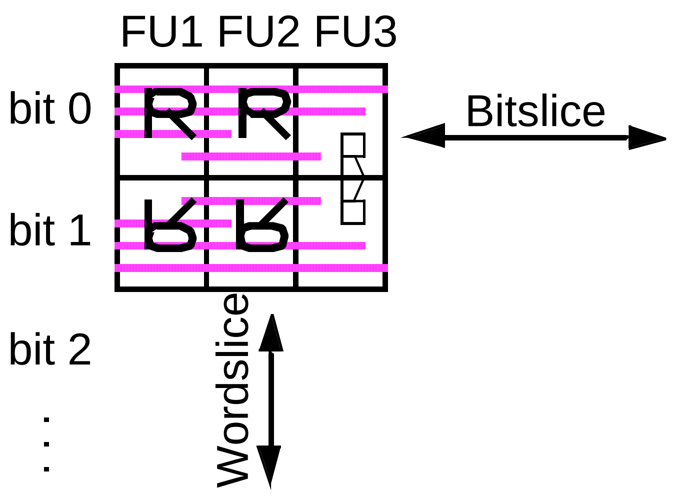
Datapath: Wires are in the Cell

Datapaths operate on multiple bit data. Most of the communication is between functional unit: bit0 of FU1going to bit0 of FU2. At each functional unit, all the bits are operated on roughly the same way. This means that each FU can be constructed from an array of identical cells, and the wires between the FU can be incorporated in the cells, so the whole structure can connect by abutment.

Think:

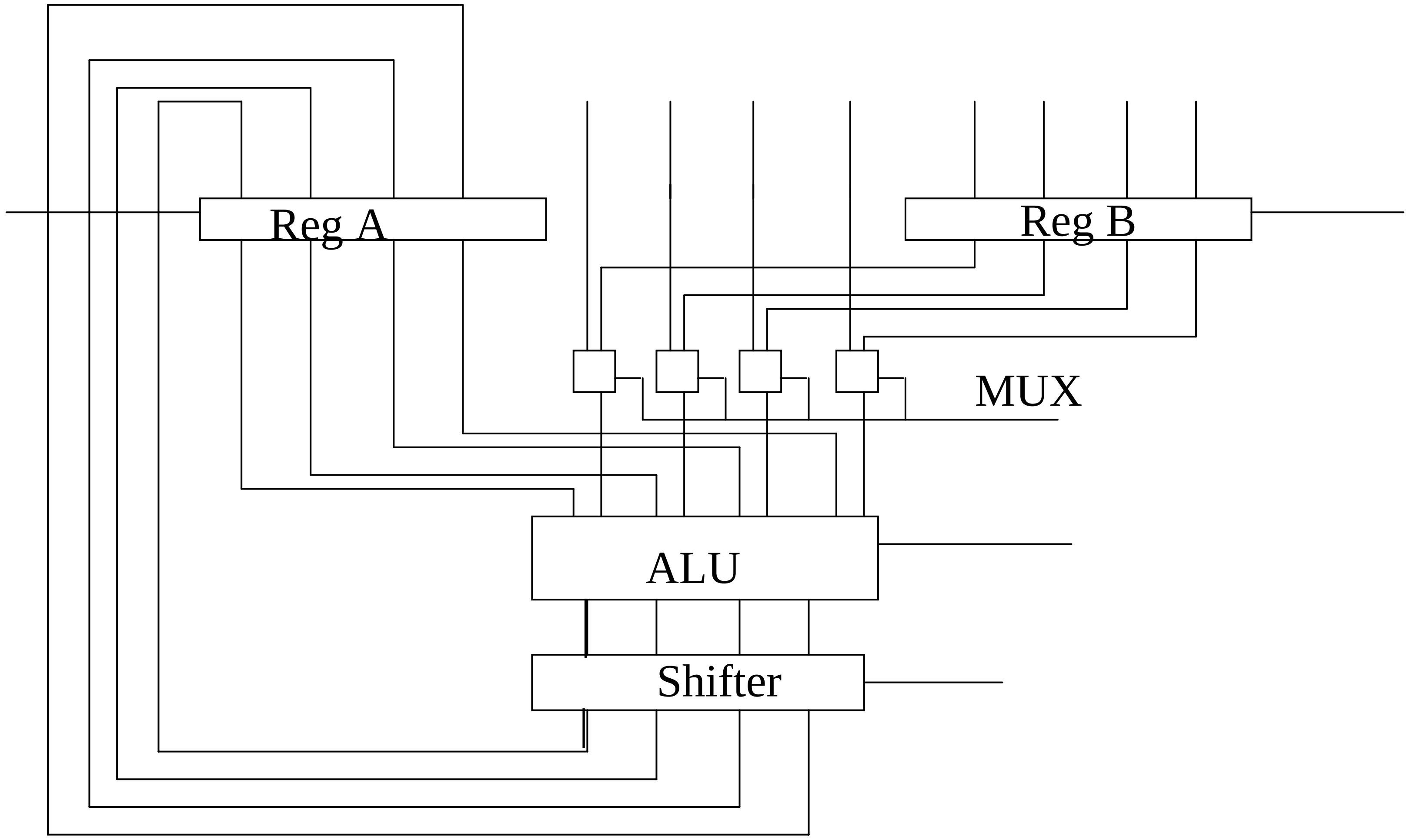


Build:

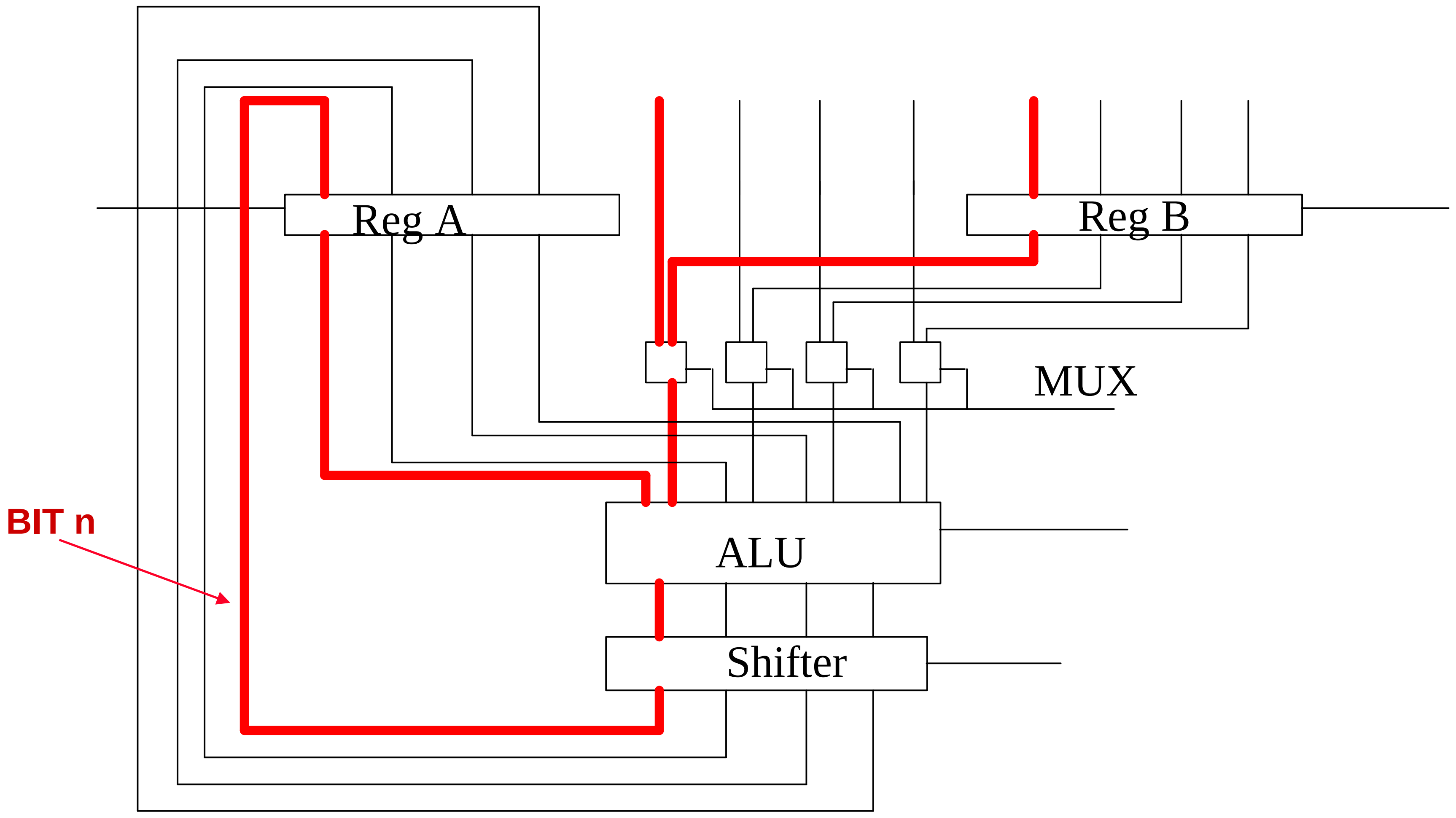


Often cells are mirrored every other row, so the cells share Vdd, Gnd rails.

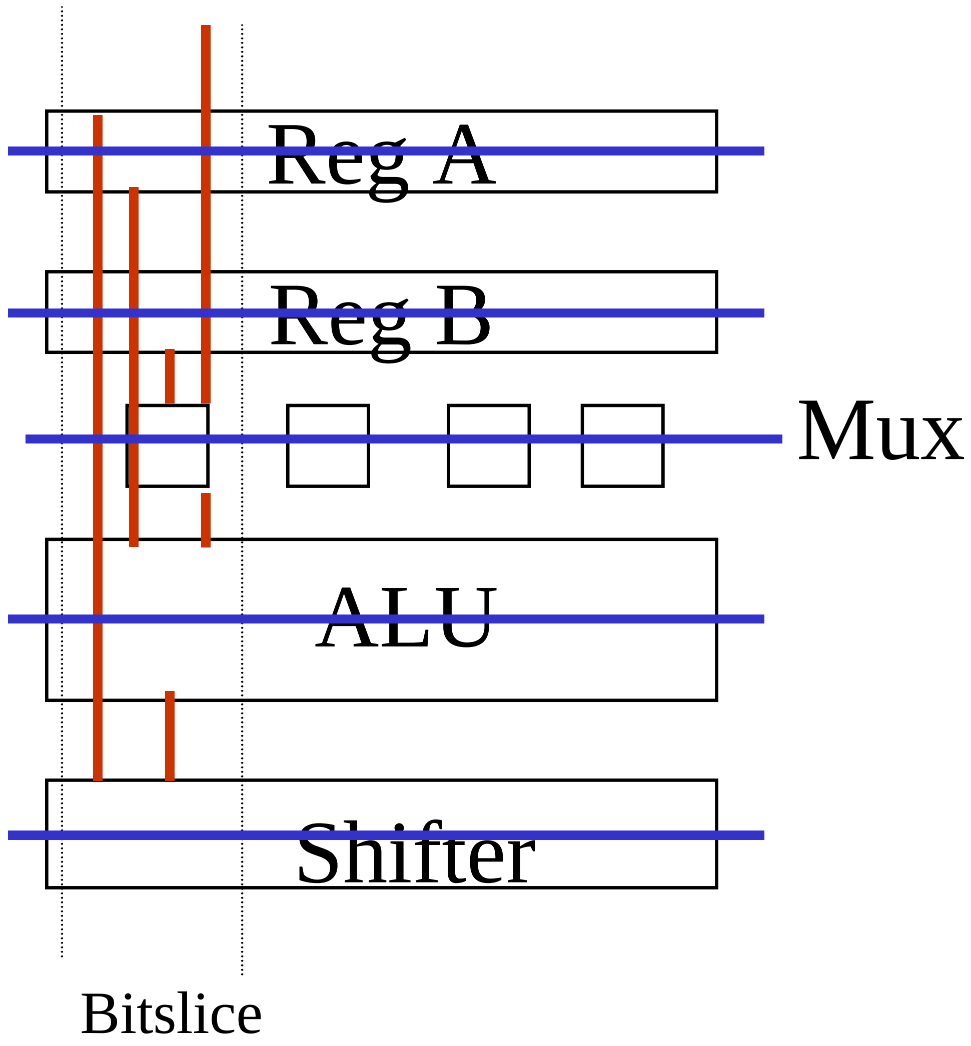
Example Datapath



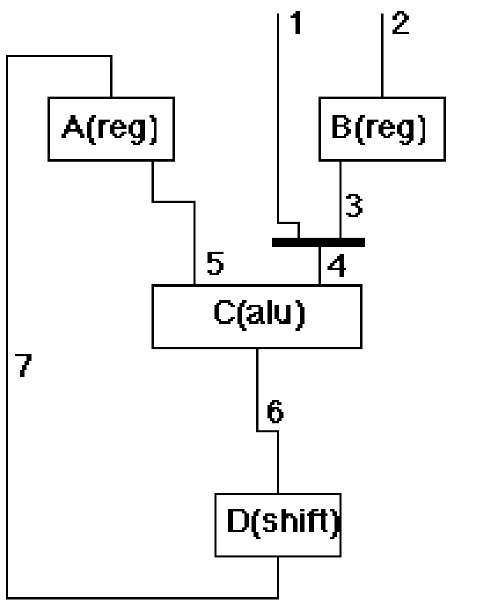
Example Datapath



Align Layout of Each Unit

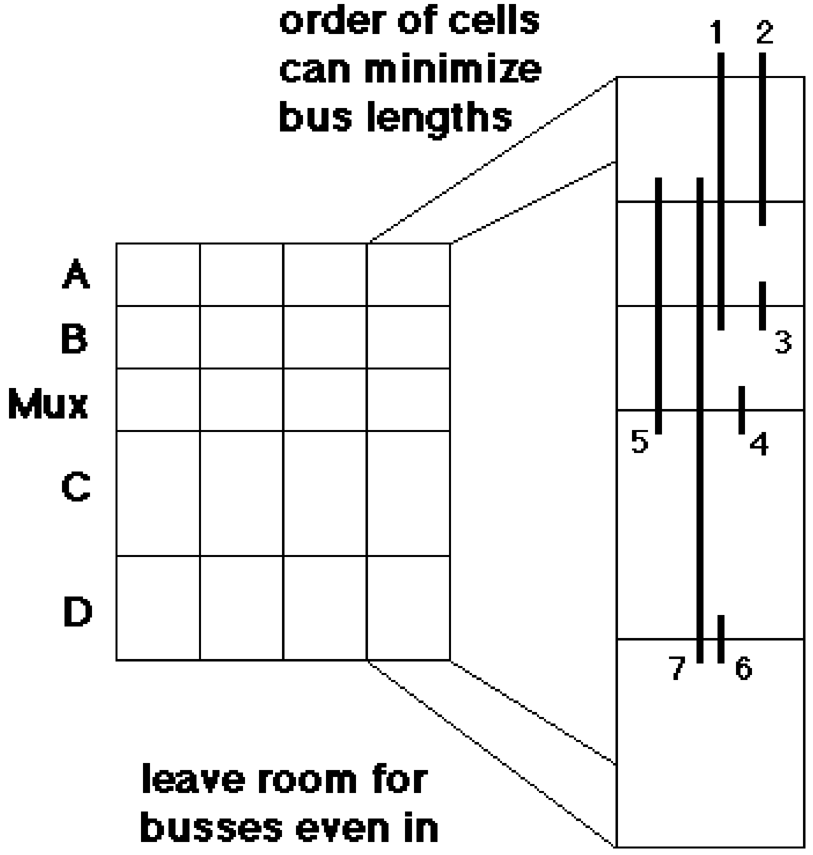


Summary of Datapath Example



if all busses are 32 bits this is a nightmare to interconnect (note: ALU inputs need to be interleaved)

order of cells can minimize bus lengths



leave room for busses even in cells that don't need them

Datapath Cells

Fixed-height cells, the height is called the bit pitch

- Set to height of tallest cell
- Set to allow required number of wires routing over the cell
- 100λ works well (typical range 80-100λ)

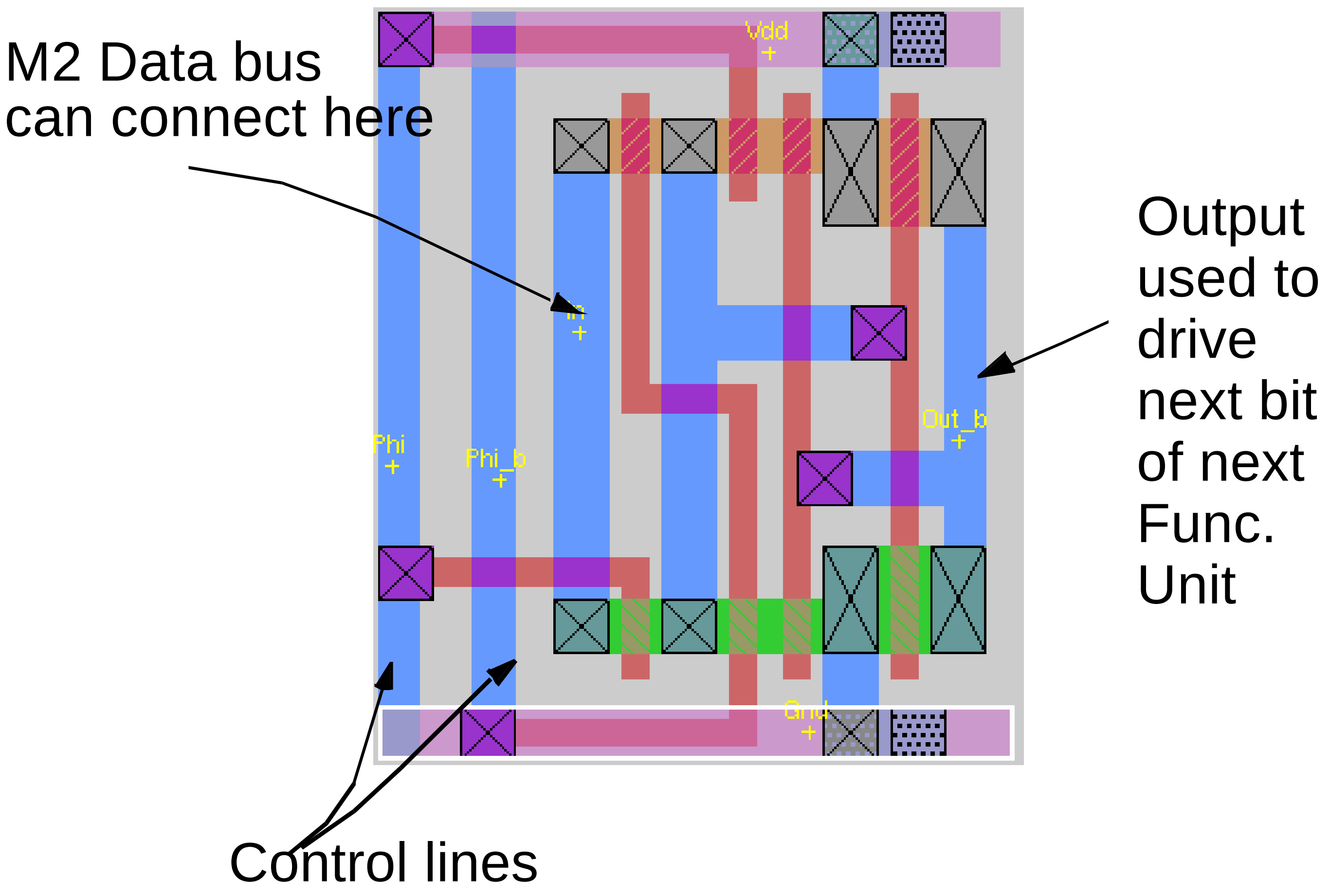
Variable width

- Width is determined by function

Wires over cells

- Vertical wires carry data between function units, busses in the design (often drawn as horizontal)
  - To nearest neighbor, distant neighbor, or pass through
- Orthogonal wires are the control lines for this function
  - Sets up the function to be performed (e.g. Mux select)
  - Often clock lines for latches

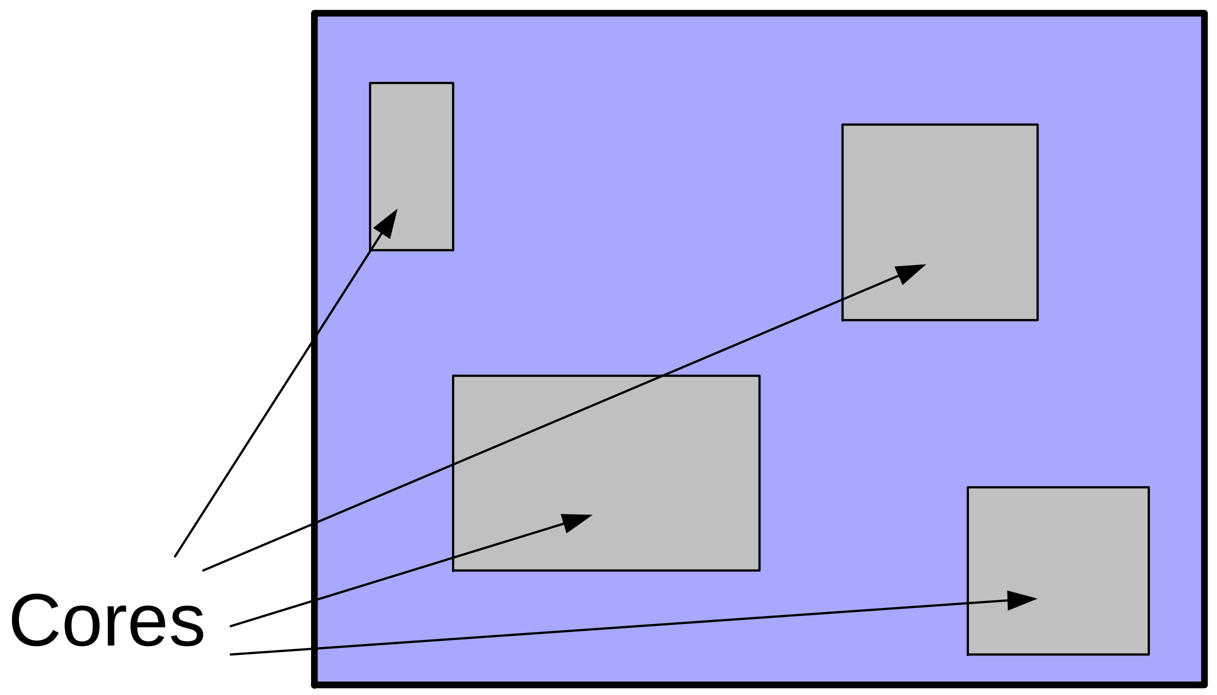
Example Datapath Cell


M2 Data bus can connect here
Vdd
In
Phi
Phi_b
Out_b
Gnd
Output used to drive next bit of next Func. Unit
Control lines

System-on-a-Chip Design (SOC Design)

Buy fixed-function cores from IP (Intellectual Property) vendors:

- combine these on a chip, product differentiation is due to novel ways of combining cores, and any special glue logic used to combine them



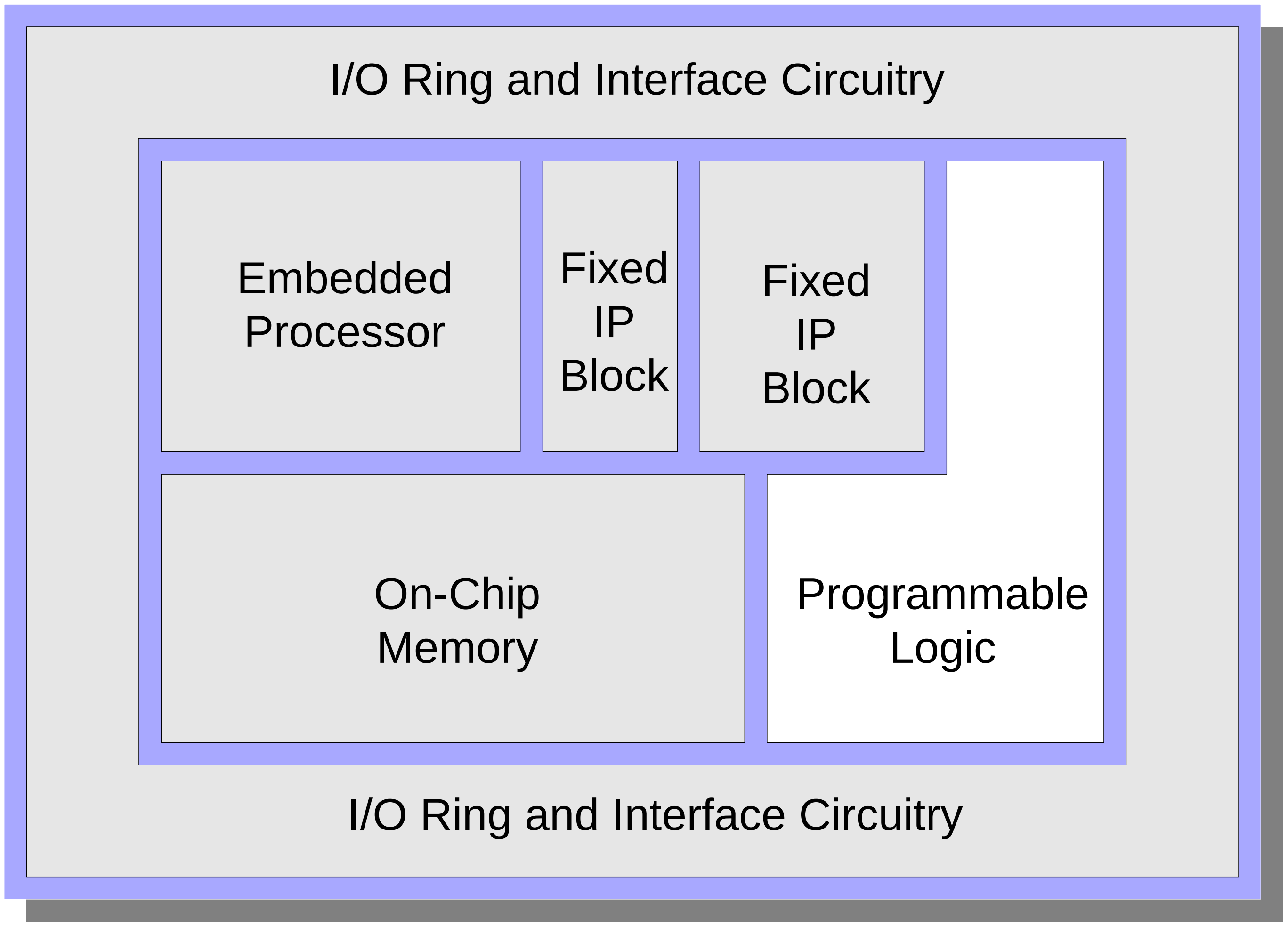
System-on-a-Chip Design (SOC Design)

Some questions:

- What information does the IP Vendor need to tell us when we buy their cores?
- How do we integrate this into a CAD flow?
- Hard cores vs. Soft Cores?
- What about the business model?

At UBC, we have a major project in this area.

Embedded Programmable IP Blocks:



Platform-Based Design

A vendor (such as Cadence, for example) might provide a chip, with many macrofunctions (cores) already pre-placed

- Each of these chips is called a platform, and is aimed at a specific application domain
- Example: a network processor platform will have many of the blocks common to all network processors
- Only a small amount of the chip can actually be customized.
    - Maybe via programmable logic
    - Maybe via embedded software