University of Pennsylvania
Department of Electrical and System Engineering
Computer Organization

ESE534, Spring 2010 Final Exercise April 12, 2010

Complete copy — updated 4/23/10;
two component-olibvious to component-specific corrections 5/10/10

Due: Monday, May 10, 5:00PM

1 Scenario

You are working as a design engineer in the CTO Office for Dcorp. Dcorp produces a 16-context architecture at various bit widths (Section 5). Currently, they support a bit-level version, a 4-bit wide version, and a 16-bit wide version.

Looking ahead, CTO Dr. Anne L. Ist is concerned about increasing rates of manufacturing defects. Anne calls a staff meeting to discuss the implication of rising defect rates and how the company should respond. In addition to Anne and yourself, the meeting includes the product manager of each of the three current product lines, Bob Bitwiddler (1-bit wide product), Nancy Nibble (4-bit wide product), and Wally Wide (16-bit product line), and Rob Bust, an architect who previously worked on SRAM architectures.

Rob suggests that defects can be easily handled with sparing just like memories. "After all, most of our area goes into memories," Rob says.

Wally quips, "well, that's likely true of the fine-grained product, but our 16-bit product puts more of its area into compute and interconnect. I doubt that is a complete answer."

Rob acknowledges that defects will likely be a bigger challenge on the 16-bit architecture, but also notes that the sparing idea works for logic blocks and interconnect tracks as well. In fact, he's recently read a paper on doing fine-grained track sparing in FPGAs [2].

Bob baits Wally with, "I don't think you should worry Wally. At high defect rates it will just make more sense to use the bit-level architecture for all designs."

Wally replies, "...and how many of our customers will we keep when your bit-level components burn an order of magnitude more energy than my division's 16-bit architecture!"

Bob replies, "You've been listening to your marketing department too much. Nonetheless, the energy difference will go the other way at high defect rates!"

Realizing the Bob–Wally discussion isn't likely to head in a productive direction, Nancy changes the subject and notes she recently heard a talk from Gojman and Rubin on component-specific mapping. She says, "I see how Xcorp,[1] could use those techniques to accommodate defects inexpensively, perhaps giving them an advantage compared to us." Nancy wonders if there is a component-specific strategy for Dcorp's products that might reduce the costs as the defect rates increase?

Rob complains that the customers would no longer see identical parts with the component-specific approach.

Anne, preferring to see numbers and calculations to fear and speculation, decides it is time to wrap up this meeting and start making progress on answering the questions raised. She says, "Yes, there are many questions for us to sort out here. Rob, can you sketch the sparing-based approach and provide us with a memo by next week?"

Furthermore, Anne realizes that the product managers each have a vested interest in their product lines and may not give her an unbiased assessment of the strengths and weaknesses of their lines. Consequently, she asks you to develop the component-specific solution, compare it to Rob's sparing-based design, and assess the impact on Dcorp's product line, including the viability of wide-word architectures, as defect rates increase.

[1] a single-context FPGA vendor who competes with Dcorp for customers

2 Problem Specification

1. Develop a component-specific defect-tolerance solution for Dcorp's architectures (Section 5).
   (a) Describe your component-specific mapping architecture, parameters, and strategy.
   (b) Develop energy and yield models for your component-specific architecture.
2. Develop an energy model for the base Dcorp architecture.
3. Develop energy and yield models for Rob Bust's proposed sparing-based architecture.
4. Select the energy-minimizing architecture parameters for Rob Bust's sparing-based architecture for defect rates from $P_f = 10^{-19}$ to $P_f = 10^{-2}$ with the goal of achieving 90% component yield of components with 2^{26} bit processing units for the three architecture cases ($w = \{1, 4, 16\}$). For this analysis, $w_{app} = w_{arch}$.
5. Select the energy-minimizing architecture parameters for your component-specific architecture for defect rates from $P_f = 10^{-19}$ to $P_f = 10^{-2}$ with the goal of achieving 90% component yield of components with 2^{26} bit processing units for the three architecture cases ($w = \{1, 4, 16\}$). For this analysis, $w_{app} = w_{arch}$.
6. Plot energy versus defect rate for both the sparing-based and component-specific models on a single graph. Also include the energy of the base Dcorp architecture on the graph for comparison. Provide a separate graph for each width ($w = \{1, 4, 16\}$).
7. Develop a model for comparing mismatched application and architecture widths and analyze whether or not it makes sense to reduce architectural width at high defect rates (i.e., select $w_{arch} < w_{app}$).
   (a) Develop a model for the energy of running a wider task on a narrower architecture.
   (b) Select the 16b-application-energy-minimizing architecture parameters for your component-specific architecture (including w_{arch} and any other granularity parameters you may have added)[2] for defect rates $P_f = 0$ and from $P_f = 10^{-19}$ to $P_f = 10^{-2}$ with the goal of achieving 90% component yield of components with 2^{26} bit processing units.
   (c) Plot energy per 16b operation versus defect rate for your solutions above (7b) and for the component-specific case where $w_{arch} = 16$ (same as $w = 16$ graph from 6).

[2]Please consider all power-of-two w_{arch}'s from 1 to 16, not just 1, 4, and 16.

3 Report Format

Please provide your solution in the form of a memo to CTO Dr. Ist recommending the approach for Dcorp.

- Summary [No longer than 4 pages] including:
  - One paragraph overall recommendation for company strategy, referring to plots and tables in this summary section as necessary.
  - Three plots (Problem 6 plots) showing energy per operation versus defect rate for the sparing based and component-specific, matched-width solutions. Each plot should also include a reference line for the energy of the original Dcorp architecture at that word width.
  - Diagram for component-specific design showing its key parameters.
  - Short highlight summary [0.5 page] of how the component-specific architectures differ from Rob Bust's sparing-based architecture.
  - Plot showing energy on 16b applications versus defect rate for your component-specific design when you allow $w_{arch} \leq w_{app}$ along with the curve above where you demand $w_{arch} = w_{app}$ (Problem 7c plot).
  - Table summarizing parameter selection for each of the 4 architecture cases (three $w_{arch} = w_{app}$ cases and the case where $w_{arch} \leq w_{app}$ for $w_{app} = 16$) for each of the 18 defect rates.
- Consultant log [1 page] summarizing all interactions with other Dcorp employees and consultants. (Date/time, length of session, participants, topics discussed)
  CFO Ty T. Rains will want to know what invoices to expect from consultants and which employee hours are billable to this project.
- Appendix [as long as necessary] showing the derivation and composition of each model and selection of parameters.

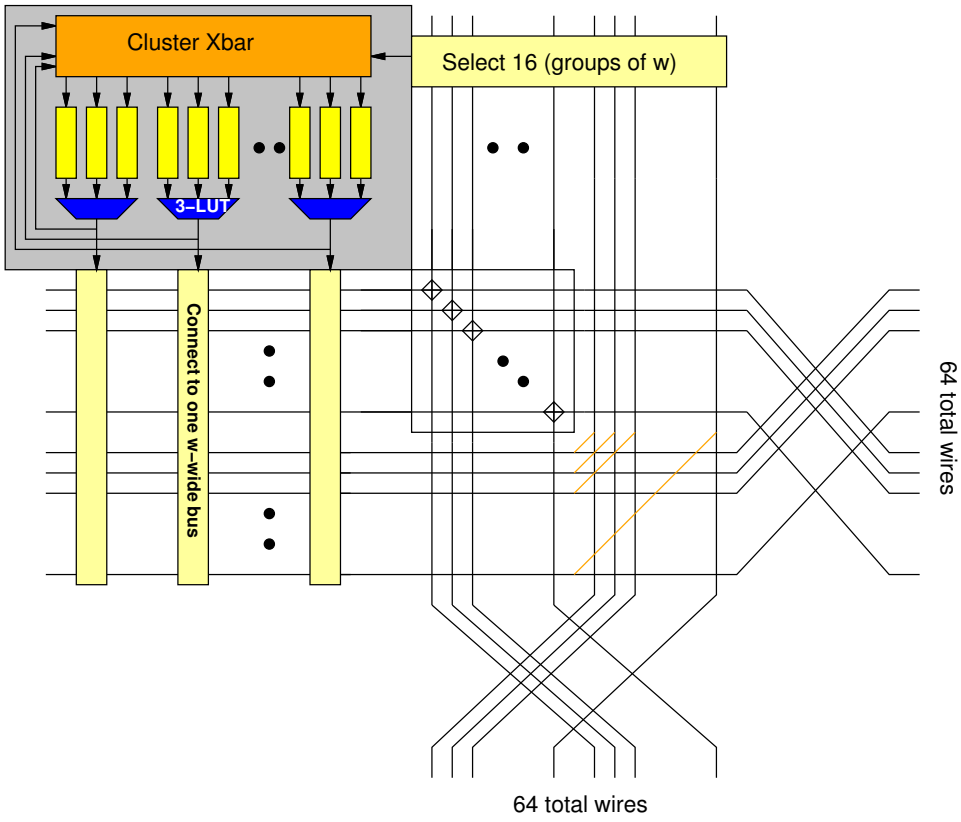
4 Collaboration

Informally: you may discuss strategy and architectural parametrization in groups through April 26th. All detailed analysis and writeup must be done individually, and no collaboration is allowed after April 26th.

More precisely:

- Each person must individually prepare his or her own solution memo.
- After April 26th, you should not discuss the project and solutions with anyone. (You may ask clarifying question of the instructor after April 26th — nonetheless, the instructor will be more generous with answers and discussion before April 26th than after.)
- Before April 26th, you may discuss:
  - provided designs
  - parameters and mapping strategy for the component-specific designs
  - general analysis approach
  - diagrams and equations on a white-board, napkin, and/or pencil and paper
  - component-specific mapping in general with Dcorp consultants Rafi Rubin and/or Benjamin Gojman
- Your writeup must document your discussion groups and sessions.
- You may not:
  - develop final analysis equations with others
  - share analysis code
  - share final diagrams
  - have a joint coding or drawing session for analysis and diagrams
  - show others your computer-drawn diagrams or computer-rendered analysis

5 Dcorp Architecture Before Adding Defect Tolerance



- Parametrized compute datapath width (W_{arch})
- 3-LUT computing elements controlled in groups of W_{arch}
- Compute blocks composed of 16 physical 3-LUTs
- Crossbar interconnect within clusters (operating on W_{arch}-wide busses)
- 16-inputs and 16-outputs from cluster to network (selected as and grouped into W_{arch}-wide busses); treat as fully populated. Input selection can be N-choose-M.
- Mesh interconnect between clusters, channel width $W_{mesh} = 64$ (segment length 2)
  - Interconnect routes busses of W_{arch} wires
  - subset (diamond) linear switchbox population topology
  - bidirectional wires
- $c = 16$ context architecture
- Local data memories:
  - $d = 16$ data memory depth
  - Three W_{arch}-wide memory banks per W_{arch}-wide operator that can read and write one word each cycle
  - Each of the $\frac{3 \times 16}{W_{arch}}$ memory banks is independently addressable (has own read and write address and own decoder).
  - Total memory $3 \times 16 \times 16 = 768$ bits in all cases
- One decoder to select the instruction word for entire tile

For the $W_{arch} = 16$ case:

- Cluster reduces to a single 16b-wide SIMD datapath.
- Cluster crossbar reduces to a 16b-wide, 2-input mux for each of the 3 16b-wide memory banks (one input comes from the channel inputs, the other is the output of the cluster from the previous cycle).
- The 64 wire wide channel is 4 16b busses.
- The input select becomes a 16b-wide, 4-input mux.
- The switchbox and corner turns have the same number of drivers/multiplexers as the other W_{arch} cases, but fewer control bits since groups of 16 switchpoints are controlled together.

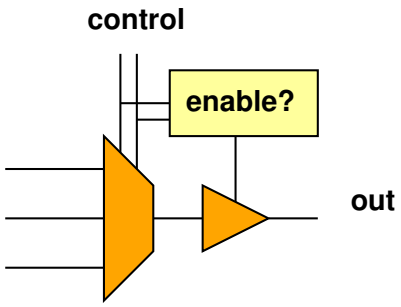
For the $W_{arch} = 4$ case:

- Cluster is 4 4b-wide SIMD datapaths.
- Cluster crossbar takes in 4b input busses (4 from input selection, 4 from outputs of the cluster) and supplies the 12 4b-wide memory bank inputs.
- The 64 wire wide channel is 16 4b busses.
- The input select is a 16-choose-4 selection unit operating on 4b-wide busses.
- Each datapath output bus can be driven to one of the 16 4b busses in the channel.
- The switchbox and corner turns have the same number of drivers/multiplexers as the other W_{arch} cases, but fewer control bits since groups of 4 switchpoints are controlled together.

6 Technology Model

LUT: Model the 3-LUT as an 8:1 mux.

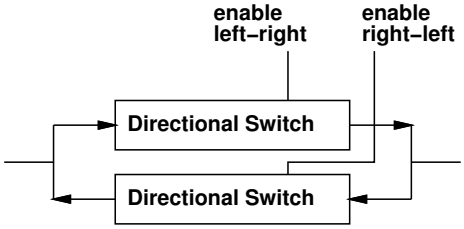
Switchbox selection: Model each switchbox output as a 4:1 mux. These are really 3:1 muxes with the ability to drive or not drive the output (Day 20, slide 61; Figure 4a in [1]). The fourth configuration is for the non-drive case. Since we are modeling a bidirectional drive case, segments can be driven from either end, bidirectional corner-turn switch, or cluster output switch.



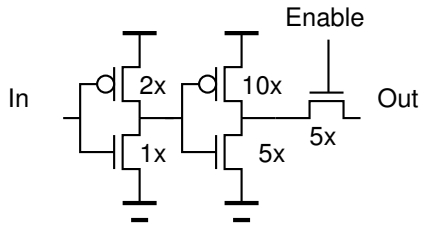
Crossbar/Mux: We will model Crossbars and muxes equivalently based on their inputs, outputs. That is, an N-input, M-output crossbar is M N:1 muxes.

- Each of the $\lceil \log_2(N) \rceil$ control inputs to the mux fails with probability P_f.
- Each of the N data inputs to the mux fails with probability $\frac{P_f}{10}$. If you are looking for a perfect mux, then the mux fails with probability $\left(\lceil \log_2(N) \rceil + \frac{N}{10}\right) \times P_f$; we provide this additional detail in case your component-specific modeling can exploit a partially defective mux. You may assume a failed mux input does not short the input wire connected to it.
- The capacitive load on each data input is 10^{-16}F.
- The capacitive load on the output of the mux is 2×10^{-16}F.
- The capacitive load on each control input is $N \times 10^{-16}$F.
- The capacitive load switched internal to the mux is 10^{-15}F.

Bidirectional Switch: Model a bidirectional switch as two back-to-back directional switches. A corner-turn is a bidirectional switch. For the component-specific case, it may be useful to reason about the independent failure of the two directional switches.



Directional Switch: An output to channel connection is a directional switch.



- Fails with probability P_f. Failure of a directional switch causes the driven wire (out side) to fail.
- The capacitive load on the data input is 10^{-16}F.
- The capacitive load on the output is 2×10^{-16}F.
- The capacitive load on the control input is 2×10^{-16}F. (Single bit control input)
- The capitive load switched internal to the switch is 10^{-15}F.

Memory: Since we are building small (nominally 16-deep) memories, the memory model is specialized around this size. We turn the $\log(d)$ term we've been using during the semester into a constant. We also size the the buffers differently.

- Each nominal 16-deep memory is addressed with 4b. There may be component-specific cases where you use a slightly larger memory that may need 5b of address.
- We will model spare and non-spare rows with the same defect rates and the same capacitive loads.
- Each memory bit fails with probability P_f. A failed memory bit does not short either its row-line or bit-line.
- The decoder for a row fails with probability P_f. A failed decoder makes the row unusable, but does not interfere with addressing of other rows.
- The each output driver from the memory fails with probability P_f.
- Capacitance switched on a data memory operation is:

$$(2dw + 4d + 4w) \times 10^{-16}\text{F}. \tag{1}$$

  d will be $16+r_{spare}$.
- Capacitance switched on an instruction memory operation is:

$$(dw + 2d + 2w) \times 10^{-16}\text{F}. \tag{2}$$

  d will be $16+r_{spare}$. Instruction memories only perform reads, while data memories perform both a read and a write on every cycle.

Wire: Failure and capacitance for wires are lumped into the mux or switch output(s) driving the wire. Wires do not need to be modeled separately. We can do this because we're working with a fixed segment length and channel width, so do not need to account for the effects of wires of significantly different lengths.

The following are assumed not to fail:

- Clock and clock network
- Program Counter
- Distribution of program counter to the Instruction Memory decoders
- Path to load instruction memories (not shown)

We assume these are built out of larger feature size technology. We further assume their energy is second order to the energy for compute, interconnect, and memories.

7 "Rob Bust" Bit-level, Component-Oblivious Architecture for Dcorp

To: CTO Anne L. Ist and staff, Bob Bitwiddler, Nancy Nibble, and Wally Wide
From: Rob Bust, Reliability Architecture
Subject: Sparing-based defect tolerance for Dcorp architectures
Date: April 19, 2010

The goal of this defect-tolerance scheme is to provide a repaired component that is completely compatible with Dcorp's baseline, defect-free design.

The strategy is to use a separate set of configuration controls—perhaps these will be fuses, or perhaps they are also SRAM configuration bits—to replace broken functions and provide a design-independent component.[3] That is, once these configuration controls are applied, the resulting component will support **any** design that would have worked on the baseline Dcorp architecture of the associated bit width.

Basic components of the solution include:

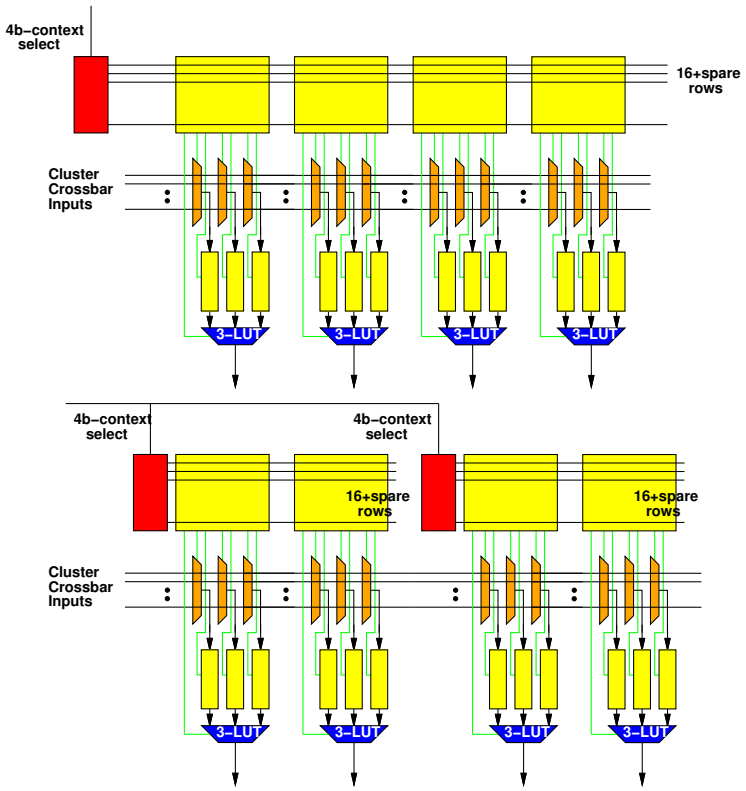
- Segregate instruction decoder within tile (how much segregation is a tuning parameter)—each decoder drives at most i_{decode} memory bits.
- Spare rows in instruction memory (r_{ispare})
- Spare rows in data memory (r_{dspare})
- Spare 3-LUT datapaths in cluster (c_{spare})
- Spare channels in mesh and spare inputs to cluster (t_{spare}) using track shifting architecture around $s \times s$ regions

We tune the parameter set $(i_{decode}, r_{ispare}, r_{dspare}, c_{spare}, t_{spare}, s)$ to best accommodate a target defect rate.

[3]To avoid further complexity, we will assume the area for these configuration controls is small compared to the resources they control. Consequently, you will not need to count these in your models.

Memory Sparing: Since much of the design is memory, we start by using row sparing to repair the memories, providing extra rows in both the data memories and the instruction memories. In the baseline architecture, we treat the instruction memory as a single, tile-wide instruction word. This means we only have to pay for the decoder overhead once for the entire tile. However, this also means that any bit error in the wide instruction memory word renders that word imperfect. Since the instruction memory word is over a hundred bits wide in some architectures, this could make it hard to yield a complete instruction word row for the higher defect rates. Consequently, as the defect rate increases, we may want to decompose the instruction word into multiple, independent memory banks, each with its own decoders, within the tile. [In the memory energy model, the $2d$ term is for the decoder. For example, breaking a w_i-bit memory bank into two $w_i/2$-bit memory banks goes from $C_{imem} = (dw_i + 2d + 2w_i) \times 10^{-16}$F to $C_{imem} = 2 \times (dw_i/2 + 2d + 2w_i/2) \times 10^{-16}$F $= (dw_i + 4d + 2w_i) \times 10^{-16}$F. While we've kept the number of memory bits the same, the need for a second decoder has doubled the energy spent on decode.] As already noted in the base architecture, each of the 3 w-bit wide data memories for each w-bit wide datapath already has its own decoder, so data memories do not have the same difficulty; consequently, we do not consider decomposing the decoders for data memories.

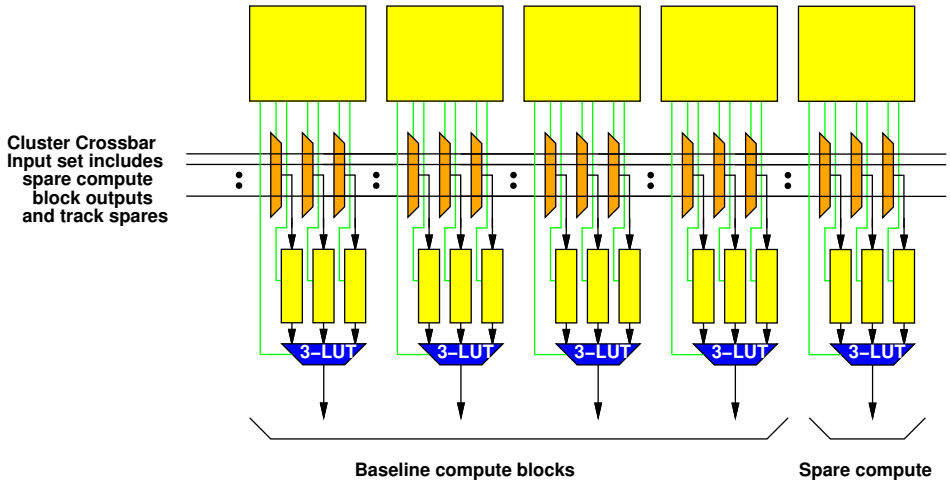
The two figures below illustrate this decoder sharing/decomposition idea. The first case shows a shared decoder across, at least, 4 compute blocks (and their associated data memories and cluster interconnect) within a cluster, while the second shows the decoder shared across only 2 compute blocks.



The complete cluster memory also controls the interconnect in the tile. You do not need to decompose the instruction memory at functional unit boundaries.

Logic Sparing: Even with perfect memory, we could have defects that damage the compute units in the cluster or the cluster interconnect. We will consider the cluster crossbar and the LUTs together. The unit of sparing is a w-bit wide LUT datapath including its output selection from the cluster crossbar. We provide spare w-bit wide LUT datapaths including input selection in the cluster. Since spares come in w-bit wide datapaths, the smallest number of spare LUTs is 1 for the 1-bit case, 4 for the 4-bit case, and 16 for the 16-bit case. This means the 16-bit case demands a duplicate datapath once it needs a spare.

The figure below illustrates sparing. Concretely, you can think of this as the 4 4-bit wide datpaths for the 4b architecture where we've added a 5th datapath as a spare.



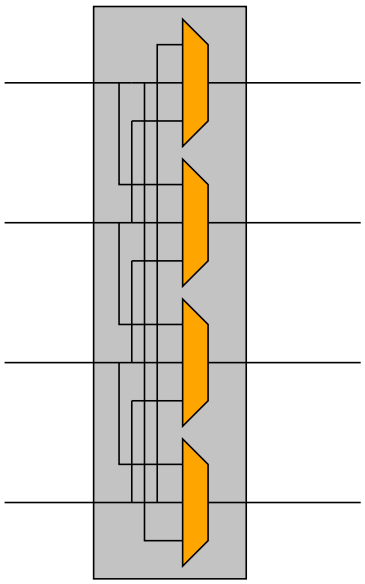
Note that the outputs of **all** of the datapaths (originals and spares) become inputs to the cluster crossbar. The cluster crossbar will also have additional inputs for segment spares as described below.

Track Sparing by Shifting: To tolerate switch and wire defects in the interconnect, we widen the channel by a number of spare busses and add the ability to shift over by a number of busses. To keep this simple, we use a slightly different scheme than [2]. We leave the normal switchboxes alone. However, we divide the array into $s \times s$ regions with track shift blocks. We parametrize both the number of spare tracks and the size of the region.

If we had a small chip, we could just add one or more spare tracks per segment offset.[4] Since we are using a diamond (subset) switch, the tracks are organized into domains. Since we want the component to look like a perfect, defect-free component, any defect in a domain makes the domain imperfect. Consequently, we substitute out defective domains for defect-free domains. However, since we have a large chip, the probability that a domain is defect-free becomes negligibly small. So, instead, we break the chip up into these $s \times s$ regions so that there is a good chance that the domain—in fact, almost all the domains—in the $s \times s$ region are defect-free. We provide spare domains to substitute for defective domains within the region.

Two adjacent regions may not have the same defective domain. So, at the boundary of the domain, we need to match the good domains in the adjacent regions. To do that we use the track shift units. The track shift units allow us to link together domains.

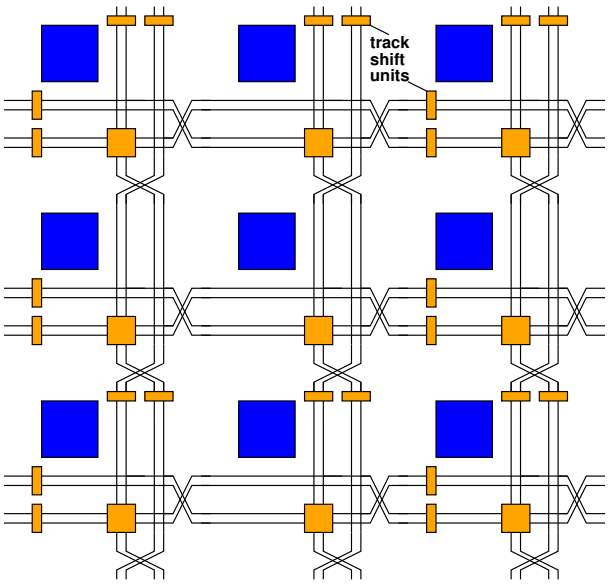
Shown below is a track shift unit in the case where there are 3 base tracks and one spare track. The shifter allows signals to shift one track up or down. The "shift" is actually a rotation so the track that shifts off the top end ends up shifting into the bottom end. In the general case, a region with t_{spare} spare tracks will need $2t_{spare} + 1$ inputs to each mux to shift t_{spare} tracks in each direction.



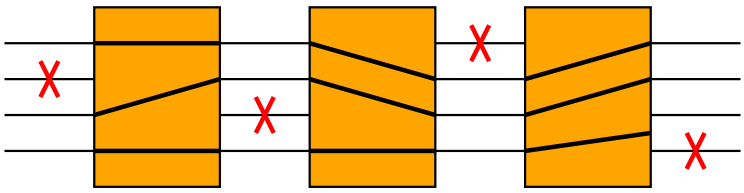
Note that shifters can also be defective. A defective shifter mux renders the domain it drives unusable.

[4] Due to our goal of identical appearance, a spare track can only substitute for a track at the same segment offset.

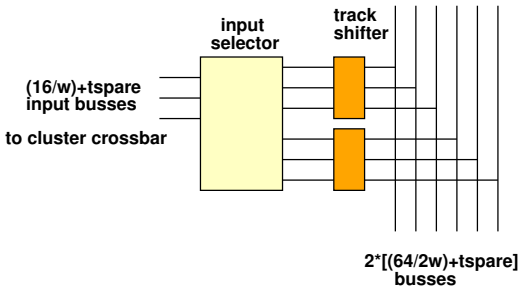
Shown below is a set of track shifters placed around 2×2 tile regions. Note that there are two separate shifters per channel—one for each of the segment offsets.



The example below shows how the shifter functions to connect up domains between regions.



Input Connection-Box Bus Selection: We also bring in t_{spare} tracks into the cluster crossbar and use a pair of track shifter between the channel and the cluster. The pair of track shifters provide one for each segment offset to match the way the wires are shifted and repaired at region boundaries.



References

[1] Guy Lemieux, Edmund Lee, Marvin Tom, and Anthony Yu. Directional and single-driver wires in fpga interconnect. In *Proceedings of the International Conference on Field-Programmable Technology*, pages 41–48, December 2004.

[2] Anthony J. Yu and Guy G. Lemieux. Defect-tolerant FPGA switch block and connection block with fine-grain redundancy for yield enhancement. In *Proceedings of the International Conference on Field-Programmable Logic and Applications*, pages 255–262, 2005.