FPGA Cell to Silicon Bridge

1,395 words · ~6 min read

VCA-CSA-101 cross-chapter handout. Audience: students who have completed Ch 5 and are curious about what their synthesized design looks like at the physical layer.

Purpose: to connect the LUT4 cells, flip-flops, and block RAM primitives your Tang Primer 25K bitstream configures to the physical silicon structures that implement them. The central idea: your FPGA bitstream is reprogrammable silicon-equivalent. The configuration changes; the transistors do not.

The three cell types in your synthesized design

When you synthesize a Verilog design for the Tang Primer 25K, the synthesis + place-and-route tools map your logic into three kinds of silicon resources:

Verilog construct	FPGA primitive	Physical structure
Combinational logic (`assign`, `always @(*)`)	LUT4	16-bit SRAM + 4:1 mux tree
Sequential logic (`always @(posedge clk)`)	DFF (D flip-flop)	CMOS transmission-gate latch chain
Large memory arrays (`reg [N:0] mem [0:M]`)	BRAM (block RAM)	Dedicated SRAM macro on die

Each of these is a fixed silicon structure on the GW5AT-138 die. You do not change the transistors. You change the SRAM configuration bits that determine how those transistors behave.

LUT4: the programmable gate

A 4-input lookup table (LUT4) implements any boolean function of 4 inputs. Internally it is:

- A 16-bit SRAM array (one bit for each of the 16 possible input combinations of 4 variables)
- A 4-to-1 multiplexer tree that uses the 4 input signals to select one SRAM bit as the output

When synthesis maps `assign out = (a & b) | (c ^ d)` to a LUT4, it:

1. Evaluates the boolean expression for all 16 combinations of `a, b, c, d`
2. Writes the 16 results into the LUT4's SRAM array via the bitstream

At runtime, the four input signals select one of the 16 SRAM cells; that cell's programmed value drives the output. The transistors in the LUT4 are always the same transistors. Only the 16 SRAM bits change between bitstreams.

Why this matters for your CPU: every AND gate, OR gate, and multiplexer in your ALU and control logic is implemented as one or more LUT4 cells. The tools may pack two related 3-input functions into a single LUT4 using a technique called function sharing, but the physical mechanism is the same.

Approximate transistor budget per LUT4: roughly 50-80 transistors at 55nm (16 SRAM cells + MUX tree + output buffer). The exact count depends on the GOWIN cell library implementation.

DFF: the programmable flip-flop

A D flip-flop in FPGA fabric is a hardwired CMOS flip-flop next to each LUT4. It captures the LUT4 output (or a direct signal) on the rising edge of the clock.

In CMOS, a D flip-flop is typically built from two cross-coupled transmission-gate latches (a master-slave pair) clocked in opposite phases:

- **Master latch:** transparent when clock is low; latches when clock goes high
- **Slave latch:** transparent when clock is high; latches when clock goes low

The result: the flip-flop's output changes once per clock cycle, on the rising edge. The slave latch's output is stable during the entire high phase of the clock, giving downstream combinational logic a full half-cycle to settle.

On the Tang Primer 25K, each DFF cell also has:

- **Clock enable (CE):** the DFF only updates if CE is high, useful for power gating idle registers
- **Set (S) and Reset (R):** forces the output to 1 or 0 regardless of the data input
- **Local set/reset (LSR):** a synchronized version of reset, driven by your design logic

When your Verilog says `always @(posedge clk) if (rst) reg_x <= 0; else reg_x <= next_val;`, synthesis maps this to a DFF cell with LSR driven by `rst` and D driven by `next_val`.

Compared to 1970s silicon: the Z80's register bits are NMOS static latch cells (cross-coupled inverters, always transparent to the appropriate control signal). Your FPGA DFFs are CMOS master-slave latches with an explicit clock edge. Both store one bit. The NMOS latch requires fewer transistors; the CMOS DFF is faster, lower-power, and more predictable in timing.

BRAM: the hard-macro memory block

Block RAM (BRAM) cells are dedicated SRAM macros on the FPGA die. Unlike LUT4 cells, which are general-purpose and can implement either logic or tiny memories, BRAM cells are specifically optimized for memory access patterns: synchronous read and write ports, optional output registers, configurable width/depth.

On the Tang Primer 25K (GW5AT-138):

- The chip has dedicated BRAM blocks organized in columns across the die
- Each BRAM block can be configured as various depth/width combinations (e.g., 16K x 1, 8K x 2, 2K x 8, 512 x 32)
- BRAM has two independent ports (some configured as one read port + one write port, some as two read-write ports)

When your Verilog infers a RAM larger than about 32 bits (the exact threshold depends on synthesis heuristics), the tools automatically place it in BRAM rather than in LUT4 cells. You can force or suppress this inference with synthesis attributes.

Your CPU's instruction memory is the most obvious BRAM candidate in the CSA-101 design: the `rom` or `imem` array that holds the 32-bit instruction words. The synthesis tool places this in one or more BRAM blocks; the bitstream initializes those blocks with your program image.

Compared to 1970s silicon: the 6502 and Z80 accessed external DRAM or SRAM chips for memory -- there was no on-chip RAM. Modern FPGAs have moved the memory on-die as a hard macro. The tradeoff is the same: on-chip is faster but limited in size; off-chip is slower but can be gigabytes.

Programmable silicon vs fixed-function ASIC

An FPGA is often described as "reconfigurable" or "programmable." What that means at the physical level:

Fixed-function ASIC (like the 6502 or Z80):

- Custom transistor layout designed once at fabrication time
- Each gate is permanent; the logic function never changes
- Very fast, very low power, very low unit cost at volume
- Changing the design requires a new silicon mask set (expensive)

FPGA:

- Standard array of LUT4 + DFF + BRAM cells on the die
- SRAM configuration bits (loaded by the bitstream) determine the function of each cell
- The same physical chip can implement a Z80-compatible CPU today and an RV32I-Lite CPU tomorrow -- you just load a different bitstream
- Slower and higher power than equivalent ASIC; higher unit cost at volume
- Zero non-recurring engineering cost to change the design

Your Tang Primer 25K bitstream is approximately 2-5 MB of configuration data that sets the state of the SRAM bits inside every LUT4 and every routing switch on the die. Loading a different bitstream is the FPGA equivalent of fabricating a different ASIC.

The pedagogical summary: when you synthesize your RV32I-Lite CPU and load the bitstream, you are not doing anything fundamentally different from what TSMC does when it fabricates a fixed-function CPU die. You are configuring a silicon structure to implement a specific logic function. The mechanism is different (SRAM programming vs photolithographic mask); the outcome is the same: silicon that executes instructions.

Routing fabric: the invisible third resource

Between the LUT4 cells and the BRAM blocks, the FPGA die contains a large routing fabric: a grid of programmable switches and wire segments that connect cells to each other. Every `wire` in your Verilog becomes a route through this fabric.

The routing fabric consumes a significant fraction of the die area. On a typical FPGA, 60-70% of the die area is routing, not logic. This is why FPGAs are larger (in die area) than equivalent ASICs for the same function: the ASIC's routing is custom-designed to be minimal; the FPGA's routing is a general-purpose grid that must support any possible connection.

When you run place-and-route (the step after synthesis), the tools are finding a path through this grid for every wire in your design. The routing delay (the time a signal takes to traverse the grid) is a major component of your design's critical path timing.

Cross-references

Backward-looking

- **CSA-101 Ch 3** (*Memory*): DFF structure; the FPGA DFF is the CMOS implementation of the D flip-flop you studied.
- **CSA-101 Ch 5** (*Computer Architecture*): synthesis and the Tang Primer 25K FPGA fabric; this handout extends the §5.8 synthesis discussion.
- **`cross-chapter-silicon-level-reading-guide.md`**: the companion handout with die-shot walkthroughs of the 6502 and Z80. Read that handout first for historical context.

Forward-looking

- **CSA-201** (*Computer Systems Architecture II*): timing closure, critical path analysis, pipeline depth vs clock frequency tradeoffs. The routing fabric delay discussed above is the key constraint.
- **RE-201** (*Hardware Reverse Engineering*): FPGA reverse engineering (bitstream analysis, LUT function extraction) is an active research area. Understanding the LUT4/DFF/routing structure is the prerequisite.
- **Virtus Peripheral IP Pack**: the VCP HDL lives on the same FPGA fabric as your CPU. Understanding cell types helps you reason about resource utilization when CPU + VCP share the GW5AT-138 die.