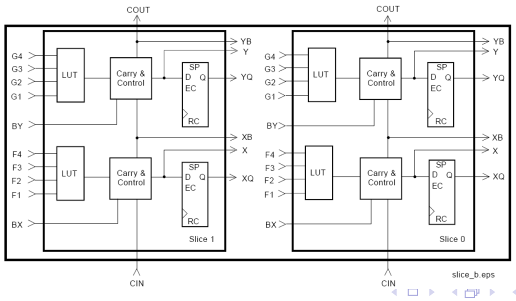
Configurable Logic Block (CLB)

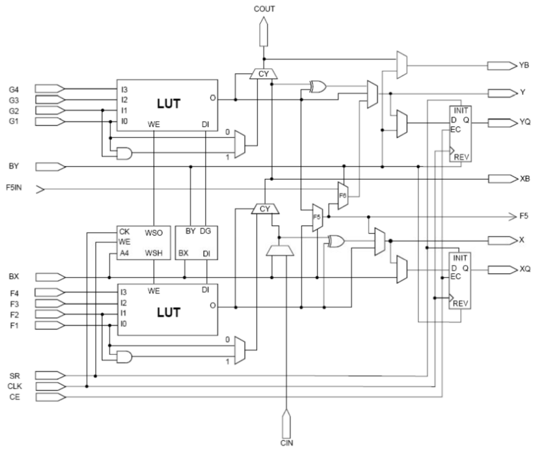
- Combinational logic generated in a lookup table (LUT)
  - Any function of available inputs
  - Function Generator
- LUT output feeds CLB output or D input of flip-flop
  - Combinational output
  - Registered output

Virtex CLB



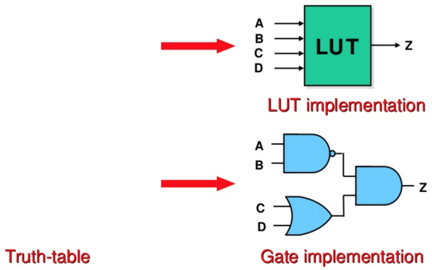
Configurable Logic Block (CLB)

Virtex CLB: Internal structure of a Slice



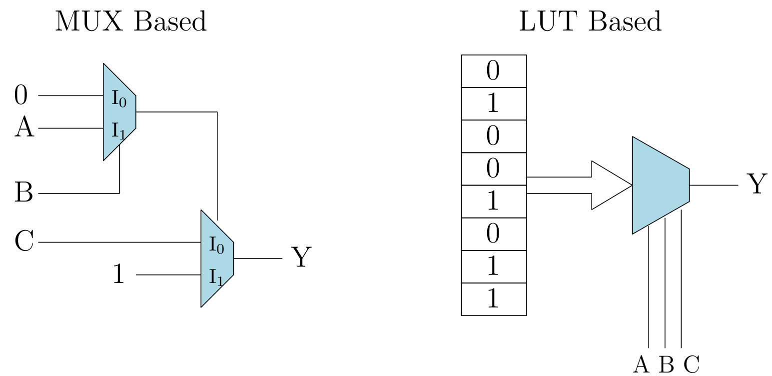
Look-Up Table

- Look-up table with N-inputs can be used to implement any combinatorial function of N inputs
- LUT is programmed with the truth-table
- Address lines as inputs, data lines as output (read mode)
- Truth Table written during configuration (write mode)



Programmable Logic

Y = (A . B) + C



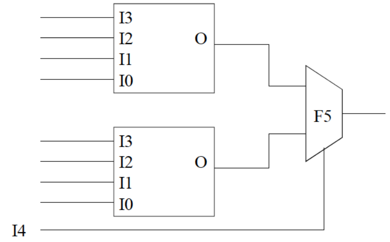
Look-Up Table

- Can be used as LUT and FFs independently

- Can be used as LUT followed with FFs

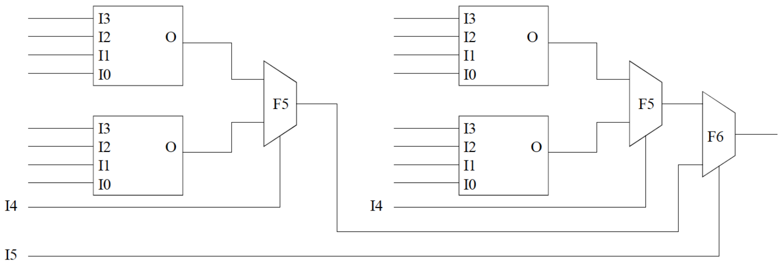
Look-Up Table

- Two 4-input LUTs are MUXed for 5-inputs using F5 MUX
- F5 MUX output may be connected with FF



Look-Up Table

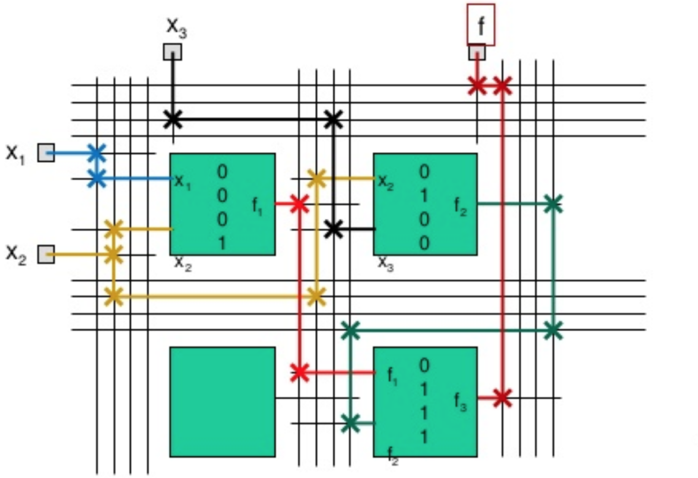
- Two 5 inputs are MUXed using F6 for a 6-input LUTs
- F6 MUX output may be connected with FF.



LUT as Distributed RAM

- LUT is written while configuring FPGA, when used for logic implementation
- Write control signals are available to be connected to routing wires so that it can be used as RAM when it is not used for logic implementation
- In Virtex E, four 16x1 distributed RAM per CLB
- These can be combined to make various memory sizes and data widths
- Since, it is spread across CLBs, it is called Distributed RAM
- Since, it is spread across, access latency can vary

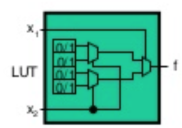
An example of programming an FPGA



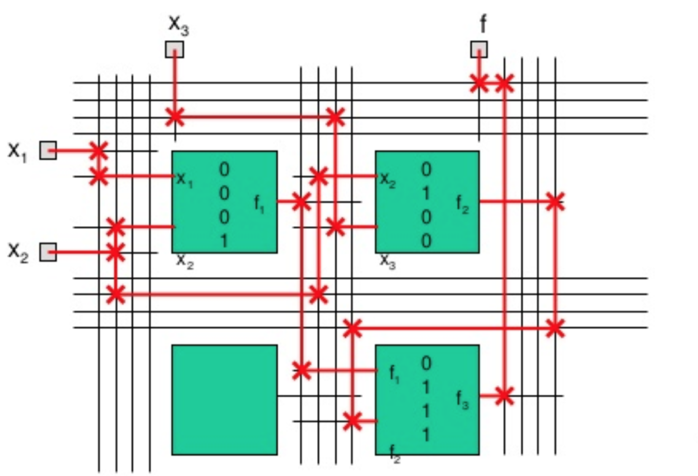
$f_1 = x_1 x_2$

$f_2 = \overline{x_2} x_3$

$f = x_1 x_2 + \overline{x_2} x_3$



An example of programming an FPGA



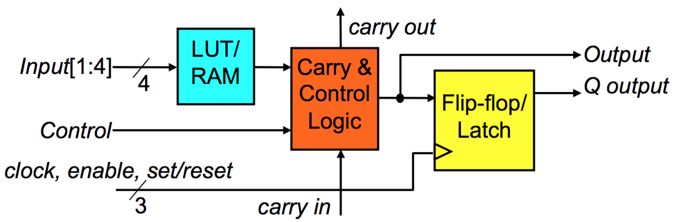
$f_1 = x_1 x_2$

$f_2 = \overline{x_2} x_3$

$f = x_1 x_2 + \overline{x_2} x_3$

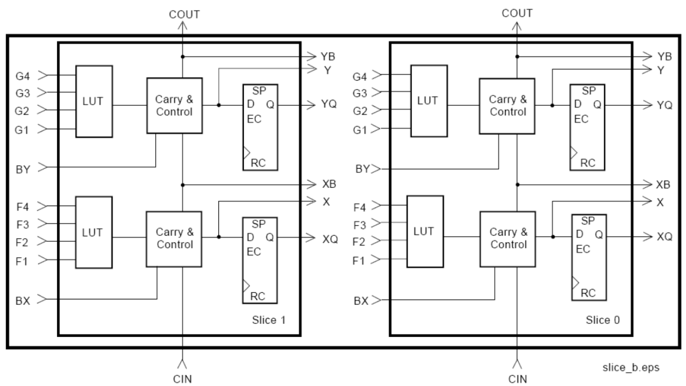
Carry and Control Logic

- Each CLB contains separate logic and routing for the fast generation of sum & carry signals
  - Increases efficiency and performance of adders, subtractors, accumulators, comparators, and counters
- Carry logic is independent of normal logic and routing resources



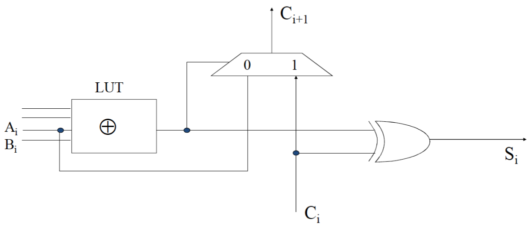
Carry and Control Logic

Virtex CLB



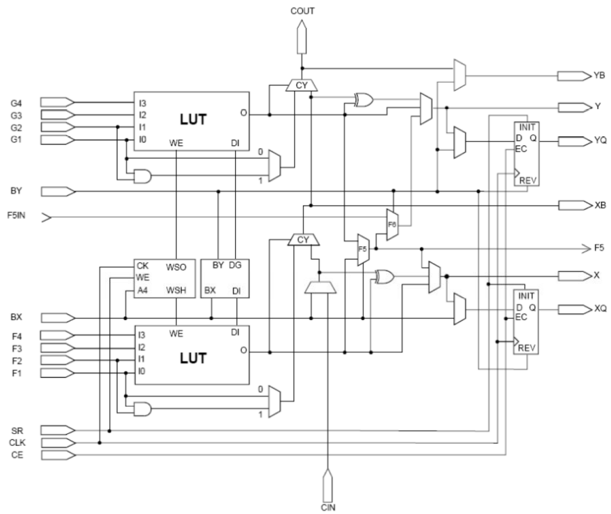
Carry Chain

- Adder
  $$S_i = A_i \oplus B_i \oplus C_i$$
  $$C_{i+1} = A_i.B_i + (A_i \oplus B_i)C_i$$
- Requires two lookup tables (S_i and C_{i+1}) at each stage
- This along with routing makes adder big and slow
- Hence, dedicated carry chain to make adder faster and implementing C_{i+1}



$C_{i+1} = A_i B_i + (A_i \oplus B_i) C_i$

Carry Chain in a Slice



Carry Chain

- For adders use the operator '+' to be able to use carry chains
- For higher level functions like counters etc.; synthesis tool infer and use carry chains
- The AND gates are for partial product generation in multipliers
- In some FPGAs, carry chain has features to cascade (AND/OR) the LUT outputs

FPGA Configuration / Programming

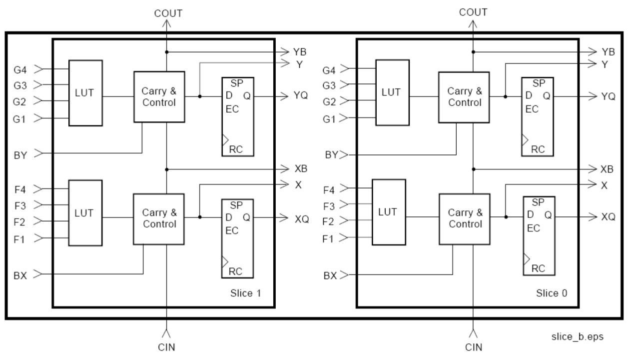
- Writing to Configuration Memory
- Configuring options in Logic Blocks
  - Writing LUTs with Truth Tables
  - Combining LUTs
  - Using LUTs as memory
  - Selecting clocks, Set/Reset for FFs
  - Configuring various MUXes in Slices
- Using special resources (RAM, FIFOs, etc..)
- Programming Switch matrices
- Programming I/O blocks

Logic Block Size

- Coarse Grain:
  - Owing to SRAM interconnection area (6 transistors) the Logic Blocks are made large in SRAM based FPGAs
  - Utilization is made high with configurability within the logic block
- Fine Grain:
  - Few, simple logic elements in a block
  - High utilization of logic block
  - Fine grain FPGA involves more interconnection & programmable switches (overhead)
    - Larger chip area
    - Lower performance

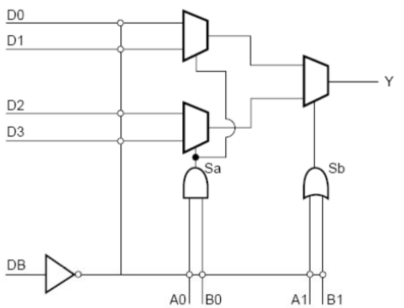
Logic Block Size: Coarse Grain

Structure of Coarse Grain Architecture



Logic Block Size: Fine Grain

Structure of Fine Grain Architecture

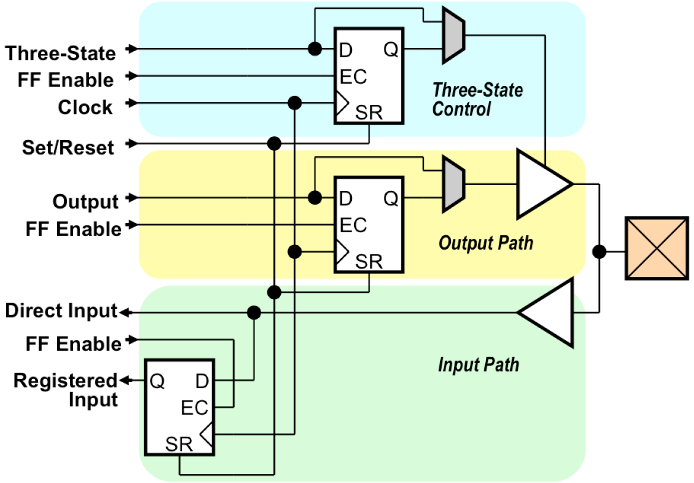


Fine-grained vs. Coarse-grained

- Fine grain is general purpose
  - Slow and area-inefficient, but high parallelism
- Coarse grain is application domain-specific
  - Coarse grain is highly area-efficient
  - Extremely high performance

FPGAs: Basic I/O Block

Structure of Basic I/O Block



FPGAs: IOB Functionality

- IOB provides interface between the package pins and CLBs
- Each IOB can work as uni- or bi-directional I/O
- Outputs can be forced into High Impedance
- Inputs and outputs can be registered
  - Advised for high-performance I/O
- Inputs can be delayed

Computer-Aided Design

- Can't design FPGAs by hand
  - Way too much logic to manage, hard to make changes
- Hardware description languages
  - Specify functionality of logic at a high level
- Validation - high-level simulation to catch specification errors
  - Verify pin-outs and connections to other system components
  - Low-level to verify mapping and check performance
- Logic synthesis
  - Process of compiling HDL program into logic gates and flip-flops
- Technology mapping
  - Map the logic onto elements available in the implementation technology (LUTs for Xilinx FPGAs)

Computer-Aided Design (Contd.)

- Placement and routing
  - Assign logic blocks to functions
  - Make wiring connections
- Timing analysis - verify paths
  - Determine delays as routed
  - Look at critical paths and ways to improve
- Partitioning and constraining
  - if design does not fit or is unroutable as placed split into multiple chips
  - if design it too slow prioritize critical paths, fix placement of cells, etc.
  - few tools to help with these tasks exist today
- Generate programming files - bits to be loaded into chip for configuration

Xilinx CAD Tools

- Verilog (or VHDL) use to specify logic at a high-level
  - Combine with schematics, library components
- Synplicity
  - Compiles Verilog to logic
  - Maps logic to the FPGA cells
  - Optimizes logic
- Xilinx APR - automatic place and route (simulated annealing)
  - Provides controllability through constraints
  - Handles global signals
- Xilinx Xdelay - measure delay properties of mapping and aid in iteration
- Xilinx XACT - design editor to view final mapping results

Applications of FPGAs

- Implementation of random logic
  - Easier changes at system-level (one device is modified)
  - Can eliminate need for full-custom chips
- Prototyping
  - Ensemble of gate arrays used to emulate a circuit to be manufactured
  - Get more/better/faster debugging done than possible with simulation
- Reconfigurable hardware
  - One h/w block is used to implement more than one function
  - Functions must be mutually-exclusive in time
  - Can greatly reduce cost while enhancing flexibility
  - RAM-based only option
- Special-purpose computation engines
  - Hardware dedicated to solving one (or class of) problem(s)
  - Accelerators attached to general-purpose computers

References

1) Christophe Bobda, "Introduction to Reconfigurable Computing - Architectures, Algorithms and Applications", Springer, 2010.
2) Neeraj Goel, "FPGA Architecture, Technologies, and Tools", *Lecture Slide*, IIT Delhi.
3) Kuruvilla Varghese, "Digital System Design", *Lecture Slide*, DESE, Indian Institute of Science.
4) C. Maxfield, "The Design Warrior's Guide to FPGAs", Newnes, 2004.
5) Scott Hauck and Andre Dehon (Eds.), "Reconfigurable Computing - The Theory and Practice of FPGA Based Computation", Elsevier / Morgan Kaufmann, 2008.