4DM4 Lab. #1
A: Introduction to VHDL and FPGAs
B: An Unbuffered Crossbar Switch

(posted Thursday, Sept 19, 2013)

Lab #1: ITB Room 157, Thurs. and Fridays, 2:30-5:20, EOW
Demos to TA: Thurs, Fri, Sept. 26,27, 2013 (tentative)

Part A - Synthesis using ALTERA FPLDs

1. Introduction to the Altera FPLDs (Field Programmable Logic Devices):

Dense FPLDs are usually based upon a multi-level hierarchy of programmable logic modules. Lets assume a dense ALTERA-like device with a 2 level hierarchy. At the bottom level of the programmable logic hierarchy is a simple "**Adaptive Logic Module**" (ALM) or a '**Logic Element**' (LE), depending upon the device family. Each module can realize a simple programmable logic circuit with perhaps 4-8 input bits and 1-2 output bits, and with 1-2 bits of memory. Each module contains a programmable **Lookup-Up Table** (LUT), which holds a truth table for the desired function(s). Each module is programmed to perform a certain function(s) by downloading a truth table.

The Altera STRATIX devices use a 0.40 nanometer CMOS process, and offer the highest performance in the Altera family. In the Stratix devices, the programmable operators at the lowest level of the hierarchy are called '*Adaptive Logic Modules*' (ALMs), as shown below. In the Cyclone family of FPGAs, these programmable modules are called '*Logic Elements*' (LEs).

In the Stratix family, each ALM has 8 input bits, 2 output bits, 2 full-adders, and 2 DFFs. Each ALM has many programmable wires to neighboring ALM. By programming the ALMs and the wires, many logic functions can be realized. In the STRATIX devices, about 34 neighboring ALMs are reachable over a 1-hop programmable wire. About 100 ALMs are reachable over a 2-hop programmable wire. About 200-300 ALMs are reachable over a 3-hop programmable wire.

Figure 1. Stratix IV FPGA ALM

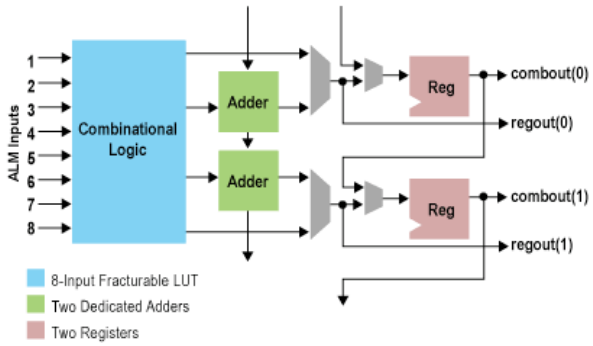
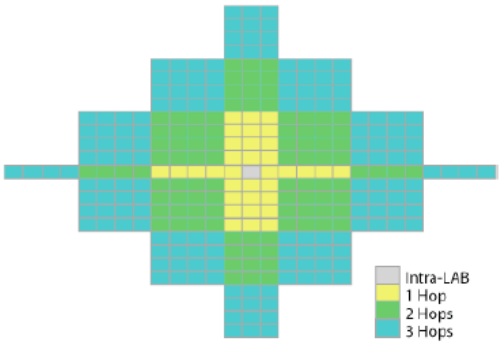


Figure 2. Stratix FPGA Series MultiTrack Interconnect Connectivity



Moving one level up the hierarchy, many ALMs or LEs are typically grouped into a unit call a '**Logic Array Block**'. Each **LAB** typically has enough logic to implement common 8-bit wide or 16-bit wide function, such as a latch, counter, multiplexer or demultiplexer, etc.

2. Logic Array Blocks – "Programmable Byte Operators"

A *Logic-Array-Block* (LAB) is essentially a "*Programmable Word Operator*" - this module performs relatively simple programmable logic operations on a word data, and it can also latch a word of data. The word is typically 8 bits or 16 bits. In Stratix devices, an LAB consists of 8 ALMs. In the Cyclone family, an LAB consists of 16 LEs. You can program one LAB to act as an adder, subtractor, counter, latch, or many other circuits. A LAB can also act as an adder or subtractor and a latch at once (this is near the upper limit of what one LAB can do).

3. Logic Elements or Logic Cells - "Programmable Bit Operators"

Each *ALM or LE* is essentially a "*Programmable Bit Operator*". Each module has a user-programmable 4-8 input Look-Up-Table (LUT), which can perform any digital function of up to 4-8 input bits. The LUT is a Truth Table with 2^k rows representing each possible combination of k input variables, and it stores the output function(s) (i.e., bit values) for each combination of input bits. Whenever an input bit changes, the LUT is consulted and the proper output bit(s) is retrieved. Each logic cell also has 1-2 programmable D Flip-Flops, which can store 1-2 bits.

The LUT is fairly flexible. A large 8-input LUT can be partitioned into one 7-input LUT, or two 6-input LUTs, etc (see the documentation). Therefore, an ALM or LE can also implement many programmable logic functions of fewer input bits each. Each *ALM or LE* has additional logic so it can implement a 1-2 bits of a full adder.

4. Embedded Memory – Programmable Memory

The ALTERA devices also have several *Embedded-Memory* (EM) blocks, called MXK blocks, where X is the number of Kbits in each block. For the Stratix devices, there are M4K and M9K memory blocks. The RAM can be used to implement memory blocks in your designs. By default, Altera Quartus synthesizes all D-Flip-flops and all memory in your VHDL design using the user-programmable D-Flip-flops in the *ALMs or LEs*. You can force Quartus to try to use the EMs for memory, by using the LPM library which contains pre-defined high speed memory components, or using the Altera *MegaWizard* tool, that interacts with Quartus. It is a easy-to-use user interface that will generate the VHDL that calls the appropriate EM block from the LPM library.

The Altera documentation provides detailed logic diagrams of the internal structure of each LAB, LE and EM. It also defines all the possible logic delays within each module; the delays to travel across different wire segments within an LE or LAB or EM. It also defines the delays incurred over the programmable global interconnects. However, Altera provides one with so

many detailed delays, that it is pretty hard to get a simple "overview", or a simple technique to estimate the performance of your VHDL hardware design.

5. Estimating the Resource Usage of your Logic Design

Each FPGA device generally has a fixed number of LABs, or equivalently a fixed number of *ALMs or LEs*. The number these operators determines how much digital logic you can program onto the chip. (The terminology depends upon the device family your are using.)

When you compile a design using Quartus, Quartus reports the utilization in ALMs for the Stratix family, or LEs for the Cyclone family.

You can get a rough estimate how many LABs your VHDL design will require, simply by counting the number of "simple" byte-operations in your design (i.e., byte-wide multiplexers, adders, latches, etc), and estimating that each byte-operation will use one LAB. The byte-operations must be relatively simple. For example, a single LAB cannot multiply 2 bytes. The multiply operation must be broken down into simpler byte addition operators, as described earlier.

You can often get a rough estimate for the delay of your design, by determining the "critical path" in your design. The critical path is the longest chain of unbuffered LABs the data must pass through, from a source latch to a destination latch, in one clock tick. You can estimate the logic delay of your VHDL design by estimating the delay for every LAB operator in the critical datapath, and summing these up. You can use some generic estimate for the delay of any one LAB, and assume all LABs have the same delay. You can also estimate the delay over every row or every column of wires (or programmable interconnect) with some generic fixed delay in nanoseconds.

6. Logic Synthesis Options

"*Synthesis*" is the process performed by the compiler, when a high-level VHDL design is mapped into individual logic resources, such as ALMs or LEs. The Quartus compiler has several options for its global logic synthesis.

There are three general optimization criteria the Quartus synthesizer can use: (1) minimization of hardware resources, and (2) minimization of delay or equivalently maximization of the clock rate, and (3) The 'balanced' mode, where each criterion (area and delay) is important.

When you compile a design in Quartus, you can specify which optimization criterion the synthesizer should use.

If you specify that the optimization criterion is delay minimization, then the compiler will implement arithmetic adders and subtractors using specialized built-in fast hardware ripple-carry chains. These specialized ripple-carry chains are built into every LAB, but they are only used

when the optimizer is instructed to minimize delay. Therefore, if you want a fast hardware circuit, set the optimization criterion appropriately.

In the Cyclone FPGAs, a fast 16-bit adder can be synthesized using only one LAB (with 16 LEs) when the optimizer is set for delay minimization. Otherwise, the synthesizer might build its own ripple-carry logic using *LEs*, instead of using the specialized built-in hardware, which generally results in slower adders and the use of more than 16 LEs.

In the Stratix FPGAs, a fast 8-bit adder can be synthesized using only one LAB (with 8 ALMs) when the optimizer is set for delay minimization. Otherwise, the synthesizer might build its own ripple-carry logic using *ALMs*, instead of using the specialized built-in hardware, which generally results in slower adders and the use of more than 8 ALMs.

To create fast adders with greater than 8 bits, ALTERA extends the specialized ripple-carry chains to longer chains of LABs within one row. Therefore, to create a large 32 bit adder, four LABs may be daisy-chained together using the specialized ripple-carry chain.

Lab #1 will have 2 parts. Part A deals with estimating how many logic resources are used to realize simple adders and multipliers. Part A also deals with estimating the generic delay of a LAB. Part A will be useful to you in the later labs when you try to estimate how much hardware your designs will take, and the clock frequency they can operate at.

Exercise Part (A):

(A1) Write a small VHDL program for a combinational circuit to add 2 n-bit numbers (ignore the carry-out bit). Compile this program, first setting the datapath width to be n= 32, 64, or 128. Use the most recent STRATIX FPGA devices, which use the smallest and fastest CMOS technology. (Stratix V FPGAs built with 22 nanometer CMOS are available, but the library may not be available on our web-edition version of Quartus. Please try the Stratix IV devices if available.)

Experiment with the Quartus control panels, to see if you can influence the synthesizer to follow an optimization strategy: (1) to optimize the resource usage, i.e., use less programmable logic at the expense of a larger delay, or (2) to optimize the delay, i.e., use potentially more programmable logic (if necessary) to get a smaller delay, ort (3) Balanced mode.

Plot the delay versus adder size n (x-axis = number of bits "n", y-axis = delay in nanoseconds). Explain the results as best as you can. Is there a formula to estimate the delay of an arbitrary n-bit adder ? If you managed to influence the optimizer, make sure you specify which optimization criteria you used. You could also report the results for different optimization criteria, for a more thorough report.

Plot the number of ALMs used versus adder size n (x-axis = number of bits "n", y-axis = delay in nanoseconds). Explain the results as best as you can. Is there a formula to estimate the delay of an arbitrary n-bit adder ? If you managed to influence the optimizer, make sure you specify

which optimization criteria you used. You could also report the results for different optimization criteria, for a more thorough report.

Experiment with the Quartus control panels, to see if you can see a 'floorplan' of your logic design, i.e., how the compiler distributed the ALMs on the device. If you can find a floorplan, capture an image of the floorplan for an n=32 adder. Explain the results as best as you can.

(A2) Repeat (A1), this time plotting the resource usage and delay for an 8-to-1 multiplexer, 16-to-1 multiplexer, and 32-to-1 multiplexer, each working on 8-bit numbers.

Part A is very simple, and you can do it at home using the student edition of the Altera software, as preparation. For the lab report for part A, submit a brief report of all these points.

Laboratory 1 – Part B
A Simple Crossbar Switch

For Part B, we will design a very simple crossbar switch, for a 'Network on Chip' (NoC). We explored an unbuffered crossbar switch design called a 'Broadcast-and-Select' switch in class. See FIG 1.

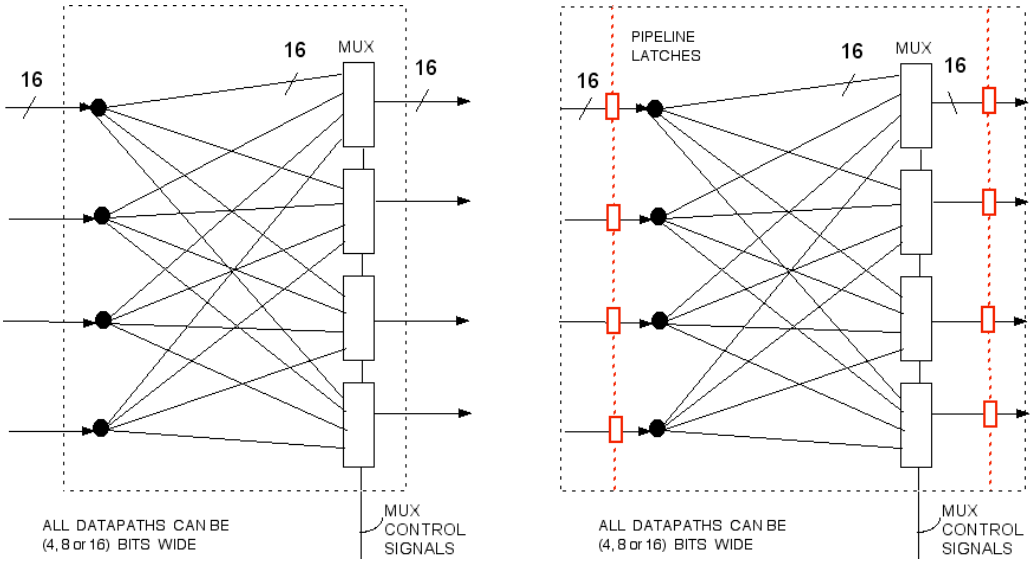


FIG 1. Simple 4x4 Broadcast-and-Select (B&S) Crossbar switches.
(a) Unpipelined, (b) Pipelined.

The switch has 4 data-input ports and 4 data-output ports. Let the datapath width W = 16 bits in this lab. The switch also has an input port for the multiplexer control signals. The pipelined switch needs 2 more inputs, the clock and reset signals (not shown above).

In this lab, we will create a pipelined B&S switch design, by adding pipeline latches at the input ports and the output ports. We will gather data from the Quartus synthesizer report, on the resource usage, and maximum clock rate. We will test the design using a testbench.

WHAT TO DEMONSTRATE TO THE TA and WRITEUP:
Repeat these questions for 2 datapath widths (W = 16 bits, and W = 64 bits).

(R1) Create professionally documented HDL code (VHDL, or Verilog, or SystemsVerilog), for the pipelined crossbar switch design. (The TA can provide support only for VHDL designs.) Show the code to the TA. (Niote; You can probably write the code using a behavioural HDL description. I believe the Quartus synthesizer can synthesize this hardware, with about 12 processes. However, Quartus might object to this many processes, in which case a structural HDL description will be necessary.)

(R2) Test for circular shift permutations: Write a testbench, and show that you can control the switch to implement any 'circular shift' permutation, where input port(i) maps to output port (i+1) mod N. Demonstrate the correct operation of the switch.

(R3) Describe the reported hardware resource usage of your design, and the reported maximum clock rate, based upon the Quartus synthesis report. Report the target FPGA device and the synthesis options.

(R4) Describe the estimated hardware resource usage of your design, and the estimated clock rate, based upon your knowledge of synthesis from Lab 1- Part A.

(R5) Compare these estimates in (R4) with the experimental results of synthesized hardware in (R3). Try to explain any differences or similarities between the hardware you expected to generate, and what Quartus actually generated.

(R6) Take a screen-snap-shot to show the placement of the switch on the FPGA chip.

Submit a brief report for Lab #1, describing all of the above. Mention the day that our TA Maryam marked your demonstration, in your lab report.

You should be able to complete lab #1 entirely on your own laptop, using the student edition of ALTERA Quartus, which you can download from the Altera website. However, you will need to demonstrate your lab. to o ur TA during one of the lab times.

Email your lab report to the prof, by the due date.
The format of the email will be posted, and the due date will be posted.