UNIVERSITY OF CALIFORNIA
College of Engineering
Department of Electrical Engineering and Computer Sciences
Last modified on Nov. 1st, 2012.

Elad Alon ***FALL 2012 TERM PROJECT PHASE II*** *EECS 141*
Due: Friday, Nov. 9th, 5pm @ Drop Box

1. Design of MUXes and Vertical Logic Block for an FPGA Slice – Description

In phase II of the project, you will be designing the MUX-based logic and the vertical logic which follow the four sets of four 16x1 SRAM columns. The main purpose of this block is that it allows for the slice's outputs to be configured in several different ways which include:

- Eight 5-input functions
- Four 6-input functions
- Two 7-input functions
- One 8-input function
- Four 1-bit full adders
- Other combinations of 5, 6, 7 input functions and adders.

Note: In all diagrams below, assume that two wires are connected **only** if there is an explicit dot.

2. Implementation

Depending on the desired possible configurations, as well as the number of output bits, there are many ways in which we can implement this MUX and vertical logic. For our particular slice implementation, we will have 9 output pins, which may not all be utilized depending on how the slice is configured.

There are two major portions of this block. The first of these is the logic that allows us to combine several N-input LUTs to implement a function of more than N inputs. To demonstrate a concrete example, we will examine how we can use two 5 LUTs to create a single 6 LUT.

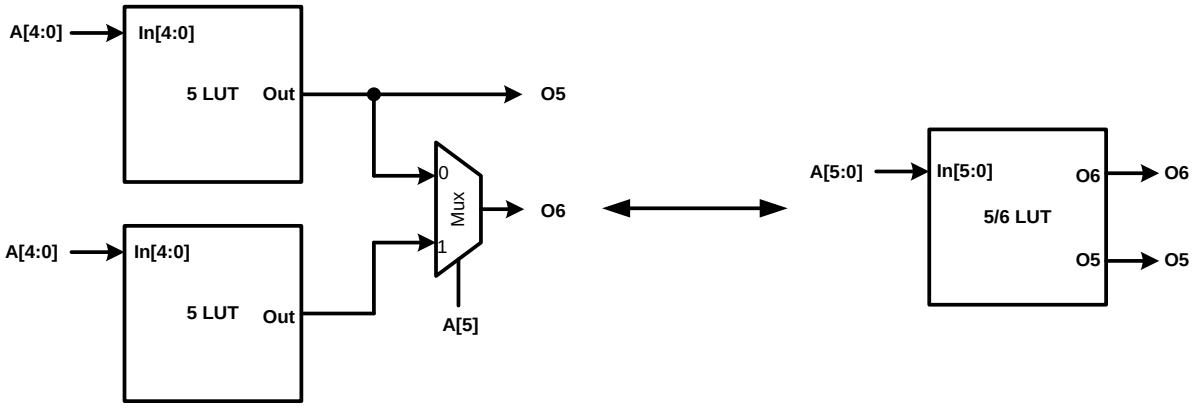


Figure 1. Structure of a 5/6 LUT

The functionality of LUTs is programmable, and this programming is performed well in advance of the functions actually being used. To implement a 6-input function with this circuit, what we would do is to program the outputs of the top 5 LUT for the case in which A<5> =0. Conversely, we would program the bottom 5 LUT for the case in which A<5> = 1. Now that we have computed both cases, we can simply select between the two LUT outputs using a MUX which is controlled by A<5>. One possible complete implementation of this MUXing is shown below. The exact way each output bit will be used is described in a table later in this document. Note that the MUXing portion is the part of the diagram to the right of the dashed line.

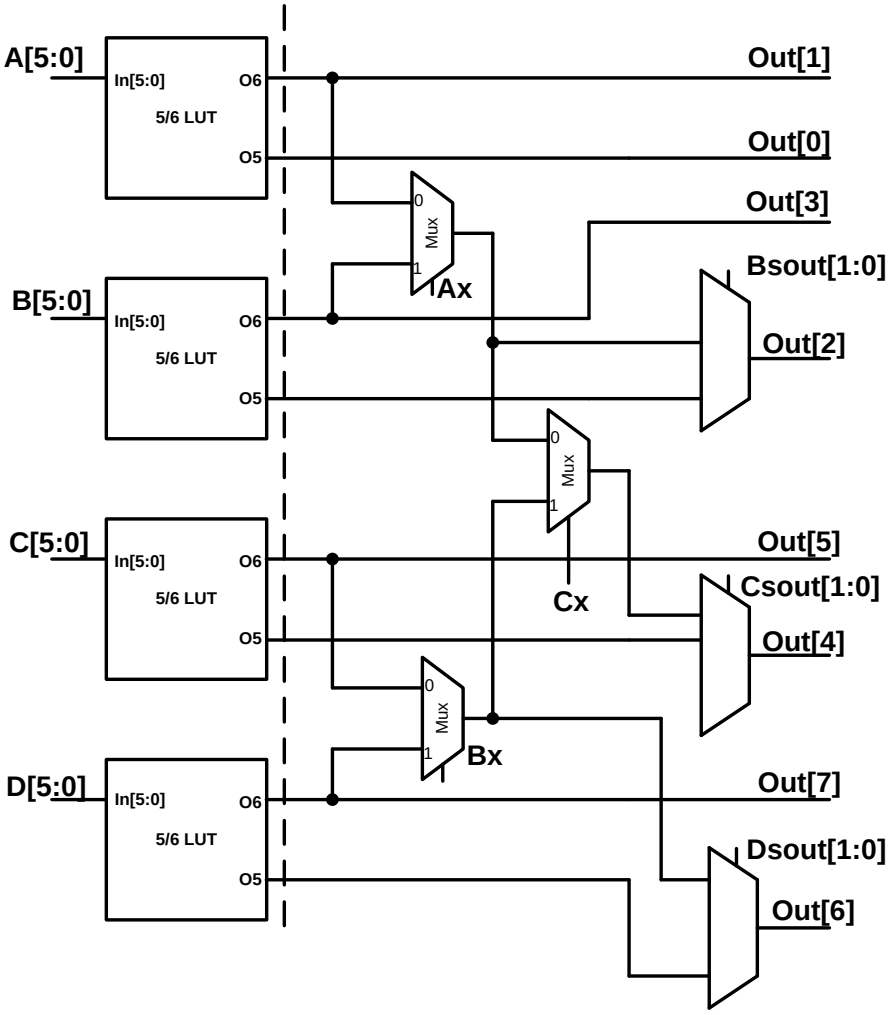


Figure 2. Structure of MUXing of Slice (without Vertical Logic)

This implementation allows for any of the first four configurations described in part 1, as well as other possible combinations, such as one 7-LUT and two 6-LUTs. Note that Bsout, Csout, Dsout (which stand for B select output, for example) would be programmed in advance, and are used to configure how this slice will be used. Note that these control signals are 2-bits each, in preparation for additional possible outputs that will arise from the vertical logic block that we will describe shortly.

For this portion of the block, be sure to implement both the MUXes displayed in Figure 2, as well as the MUXes within each 5 LUT. MUXes within each 5 LUT are necessary because we chose to implement the memories as a 16x2 array, instead of as a 32x1 array.

The second portion of this block is the vertical logic. In particular, this vertical logic is used to speed up addition operations, which tend to be fairly common. The purpose of this logic is to use the carry-in bit to generate the sum at this bit, as well as to generate the carry-in for the next bit. To understand this, we'll take a look at how a full adder works.

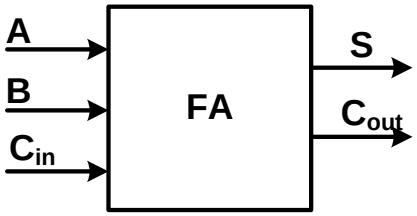


Figure 3. 1-bit Full Adder

$$S = A \oplus B \oplus C_{in}$$
$$C_{out} = AB + BC_{in} + AC_{in}$$

As discussed in class, we can reformulate C_{out} in terms of propagate, generate, and kill, which are defined as below, and give us this new expression for S and C_{out}:

$$P = A \oplus B$$
$$S = P \oplus C_{in}$$
$$G = AB$$
$$C_{out} = PC_{in} + G$$

We can recast the expression for C_{out} as:

$$C_{out} = P\ ?\ C_{in} : G$$

By this, we mean that if P = 1, then $C_{out} = C_{in}$. Otherwise, if P = 0, $C_{out} = G$. It is helpful to write C_{out} this way because it maps directly to a MUX. Figure 4 on the next page shows an example of how we can combine such a MUX with the LUT to implement a 1-bit addition.

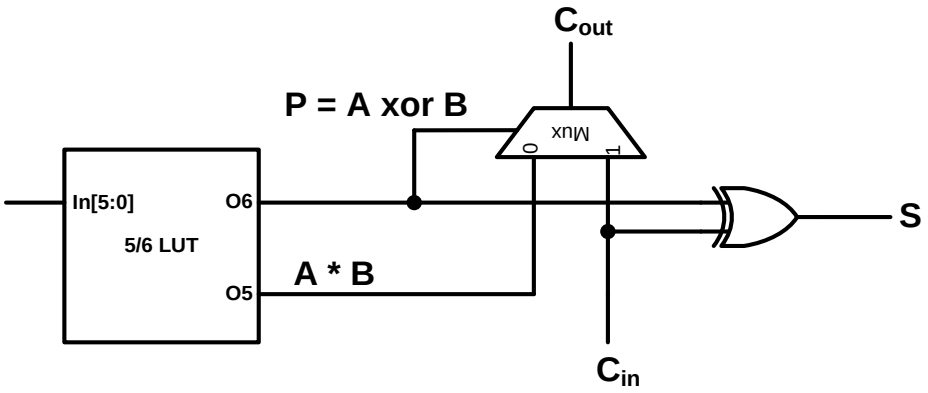


Figure 4. Implementation of 1-bit Full Adder using LUTs and Vertical Logic.

As indicated by Figure 4, the O6 output of the 5/6 LUT is configured to implement $P = A \oplus B$, while the O5 output of the LUT implements $G = AB$. Similarly, an additional XOR gate is used to compute the final SUM. Note that this is just one possible implementation for the vertical logic; your job in this phase is simply to implement any vertical logic that compute both S and C_{out} for each 5/6 LUT configured as an adder. In other words, you should make sure that your vertical logic can add up to 4 pairs of bits.

The previously two previously described portions (MUXing and vertical logic) form one overall block that you will be designing in this phase. This gives us the top level block shown below, which highlights all of the inputs and outputs. In this diagram, input L consists of the 16 outputs from the four sets of four 16x1 SRAM columns.

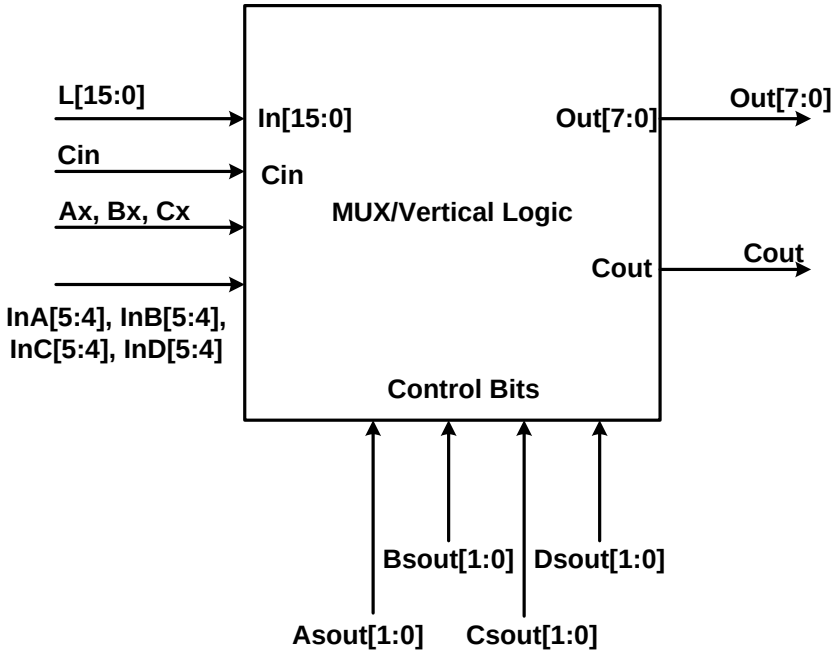
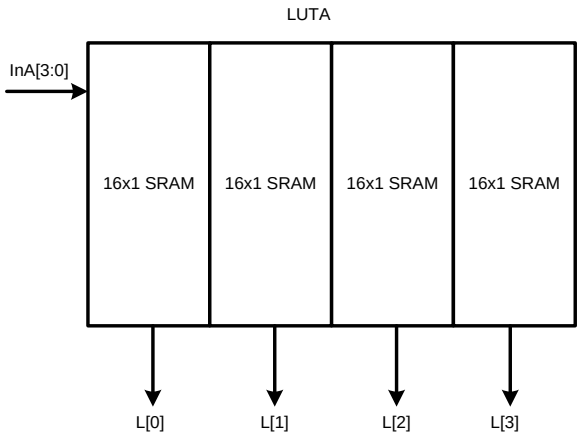


Figure 5. The MUX/Vertical Logic Block and its inputs as well as outputs.

To clarify the various inputs and outputs of the complete block, you should note the following. The Asout, Bsout, Csout, and Dsout inputs are set when the slice is configured, and are used to select what the output bits are. Their usage here is similar to their usage in Figure 2. InA[5:4], InB[5:4], InC[5:4], InD[5:4] are the top two bits of the input into each 5/6 LUT. Note that in the effectively 16x4 SRAM array, we used the bottom input four bits (i.e., InA[3:0]) to decode the appropriate wordline. Finally. Cout is the carry out of the last LUT in the slice.

In order to verify your design, we will be providing you a SPICE deck that will verify the design's functionality. This means however that your design needs to stick to a certain structure in order to be compatible with our desk, and we will next describe that structure. Let's call each set of four 16x1 columns of SRAM different names – LUTA, LUTB, LUTC, LUTD. The outputs of LUTA should look like this:



LUTB, LUTC, and LUTD follow this same output pattern, except they would occupy L[7:4], L[11:8], and L[15:12], respectively. Now that the inputs are specified, we can specify the outputs of each 5/6 LUT as a function of these SRAM outputs. Remember there should be two outputs for the 5/6 LUT, which we will call O5 and O6. This table is for LUTA in particular, but the same pattern applies to the other LUTs. This is described in the table below:

In[5]	In[4]	O5	O6
0	0	L[0]	L[0]
0	1	L[1]	L[1]
1	0	L[0]	L[2]
1	1	L[1]	L[3]

Now we can group LUTA with its MUXes into a 5/6 LUT, which we'll call LUTA6, with outputs $O5_A$, $O5_B$. We will do the same for the other LUTs, so we have LUTB6, LUTC6, and LUTD6 as well. Now, given these O5, O6 outputs, we define the 7 LUT outputs are below, where $O7_A$, $O7_C$ are the two outputs of the two 7 LUT configurations:

$$O7_A = \overline{Ax} \cdot O6_A + Ax \cdot O6_B$$
$$O7_B = \overline{Bx} \cdot O6_C + Bx \cdot O6_D$$

We can also define the single 8 LUT output as:

$$O8 = \overline{Cx} \cdot O7_A + Cx \cdot O7_B$$

Now that we have defined all these intermediate outputs, we can define the output in terms of these intermediate outputs. Also, S_A, C_{outA} are defined as the sum and carry bits associated with LUTA6. S_B, C_{outB}, etc. are defined in a similar manner.

Output Name	Possible Outputs
Out[0]	$O5_A$, S_A, C_{outA}
Out[1]	$O6_A$
Out[2]	$O5_B$, $O7_A$, S_B, C_{outB}
Out[3]	$O6_B$
Out[4]	$O5_C$, O8, S_C, C_{outC}
Out[5]	$O6_C$
Out[6]	$O5_D$, $O7_B$, S_D, C_{outD}
Out[7]	$O6_D$

For the outputs that must be configured to select between one of several possibilities, the output chosen is determined by Asout, Bsout, etc. The behavior should be as shown below:

Select Input	Output
00	O5
01	For Out[2]: $O7_A$ For Out[4]: O8 For Out[6]: O7B
10	S
11	C_{out}

Note that Asout = 01 is unused. You are free to choose how the behavior is defined for this input, given that it is reasonable and does not affect valid inputs.

As previously discussed, you are free to construct the block however you choose, but you must follow the I/O interface from Figure 5 in order to ensure compatibility with our verification deck.

We have provided a library of standard cells that you are free to use in constructing your design. Using the same process outlined in phase I for each of the cells, you can copy the standard cell library from the project directory ~ee141/fall12/project/t2b_blocks/. The library includes many of the building blocks you are likely to need, including an XOR, XNOR, NAND2, NAND3, NOR2, NOR3, INV, and MUX2.

It is up to you to assemble in schematics (there will be no layout in this phase of the project – you will have ample opportunity to do that in phase 3) the standard cells (and any additional cells you decide to design) into the MUX/Vertical Logic.

Note that you are not required to optimize the gate sizes of the MUX/Vertical logic block – i.e., you can simply use the standard cells to implement the required logic functions. However, you should use the cells as efficiently as possible in order to minimize the delay and power of the circuit. If you find that you'd like to have a larger width than that used in the standard cells for one of the gates, you can achieve this by placing two or more of the cells in parallel with each other. If you do decide to explore optimizing this block's sizing (which we would highly discourage at this point since the rest of your design may change substantially, affecting what is optimal vs. not for this block), remember that each output of the slice will be driving a load capacitance of 15 fF.

3. Analysis and Simulation

Your primary goal in any IC design should be to ensure that the circuit you have designed functions as intended. Since the number of inputs to your MUX/Vertical Logic block is relatively small, we have provided to you a SPICE deck (which is at ~ee141/fall12/project/t2b_blocks/check.sp) that you will use to essentially exhaustively check that the output of your block is correct for almost all possible inputs. Please attend one of the discussion sessions for further instructions on how to use this deck to check your design.

As you are assembling your design, you should keep in mind which input pattern will result in the worst case delay. Once you have finalized the assembly of your block and identified which gates are on the critical path, you should hand analyze the delay of this path. You should then simulate your design with this worst case pattern to find the delay. Please remember to load each of the outputs of your block with 15 fF. If there are any major discrepancies between your analysis and the simulation, you should explain the reasons for this in your report.

4. Report

The quality of your report is as important as the quality of your design. Be sure to provide all relevant information and eliminate unnecessary material. **Organization, conciseness, and completeness are of paramount importance.** Do not repeat information we already know. Use the templates provided on the web page. Make sure to fill in the cover page and use the correct units. Turn in the reports for each phase in the homework drop box.

4.1 Report for Phase II

The organization of the report should be based on the following outline:

Cover page: Names, calculated delay, simulated delay.
Page 1: Schematic of the MUX/Vertical Logic Block with details of each of the blocks.
Page 2: Graph from the functionality check SPICE deck.
Page 3: Schematic showing the gates on the critical path and simulated delay.
Page 4: Hand estimate of the MUX/Vertical Logic Block delay, explanation of any discrepancies vs. sim.