

University of California at Berkeley
College of Engineering
Department of Electrical Engineering and Computer Sciences

EECS150
Spring 2000

J. Wawrzynek
E. Caspi

Homework #3 – Solution

1. You are asked to choose a technology (discrete gates, application-specific IC (ASIC), microprocessor, or FPGA) for implementing an automobile's digital engine control system.

- a) NRE (non-recurrent engineering) cost reflects the design cost. In general, software is easiest to design, while custom circuits are hardest. Also, programming someone else's components is easier than fabricating and programming your own. Ranking the technologies with respect to NRE, most expensive to cheapest:

ASIC, discretely, FPGA, microprocessor

The order of discretely and FPGA may be switched, since they can be programmed by similar software (*e.g.* schematic capture, as in our labs).

Manufacturing cost depends on fabrication cost, parts cost, and assembly cost. Fabrication is required only for ASICs, and can be made very cheap when amortized over a large number of dies. Assembly cost depends on the parts used and the number of parts. Ranking the technologies with respect to manufacturing cost, most expensive to cheapest:

Discretely, FPGA, microprocessor, ASIC

The order of FPGA and microprocessor may be switched, since their part cost and packaging (hence assembly cost) are similar.

- b) The two most attractive solutions are ASIC (for a low cost, single-chip solution) or microprocessor (for ease of development).

2. Questions from the Xilinx data book

- a) CLB configuration bits are required for three kinds of elements:

- n -LUTs – each is a lookup table with 2^n single-bit entries
- Multiplexors – a 2^m -to-1 multiplexor requires m control bits
- Flip-flops – a FF may use a configuration bit for its initial value

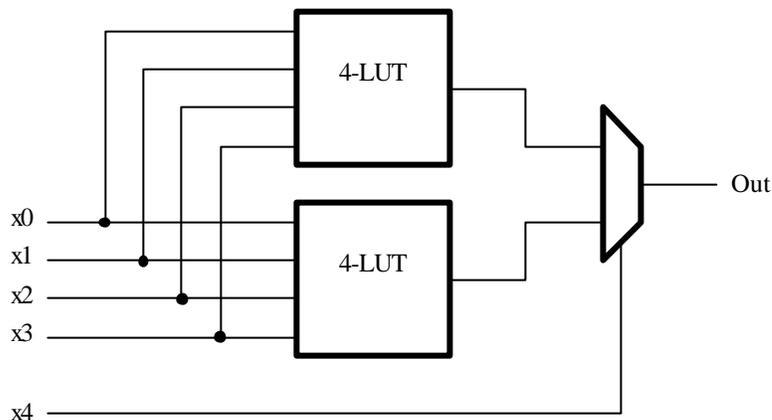
Figure 1 (page 6-10) shows that the Xilinx CLB contains 2 4-LUTs, 1 3-LUT, 2 4-to-1 multiplexers, 10 2-1 multiplexers, and 2 S/R registers (to set the initial value of the D flip-flops). Thus the configuration bits required are:

- 2 4-LUTs: $2 \cdot (2^4) = 32$ bits
- 1 3-LUT: $1 \cdot (2^3) = 8$ bits
- 2 4-to-1 (i.e. 2^2 -to-1) multiplexers: $2 \cdot 2 = 4$ bits
- 10 2-to-1 multiplexers: $10 \cdot 1 = 10$ bits
- 2 S/R registers: $2 \cdot 1 = 2$ bits
- Total: $32+8+4+10+2 = \mathbf{56}$ bits

Note that we have intentionally ignored the 4 multiplexers at the top of the figure. Those muxes belong to the fast carry logic which is detailed in Figures 13 and 14.

Our estimate of 56 configuration bits per CLB is actually much smaller than the real number. We have completely ignored configuration of the programmable interconnect. Specifically, configuration bits are needed for switches connecting each CLB to the general interconnect as well as for switches bridging different segments of interconnect. Figure 27 shows a “tile” containing a CLB and all the switches associated with it – a Xilinx 4K FPGA is essentially composed of adjoining repetitions of this tile. According to Table 20, the Xilinx 4000E series uses around 500 configuration bits per tile.

- b) The “Passive Pull-up / Pull down” circuitry of Figure 15 is used to automatically pull an interconnect wire to a high or low logic level when no CLB is outputting to it. This feature is useful when a wire is used as a bus, with multiple CLB outputting to it through tri-state buffers. If all CLBs disconnect from the bus (i.e. set their tri-states to high-impedance), then the bus voltage may float to a value which is neither logic high nor low. Such a situation may cause glitching and excessive current draw in circuits that take their input from the bus.
3. A 5-LUT can be implemented using a pair of 4-LUTs and a multiplexer. Considering the 5-LUT as a function of 5-inputs $x_0 \dots x_4$, we partition it as follows: a first 4-LUT implements the function with $x_4=0$, and a second 4-LUT implements the function with $x_4=1$. A multiplexer controlled by x_4 then selects which 4-LUT represents the true value of the function.



The multiplexor can be implemented in its own 4-LUT to create a pure 4-LUT implementation.

4. A 4-LUT can implement *any* function of 4 inputs. To implement a given circuit in 4-LUTs, we must partition it into a composition of 4-to-1 subcircuits, each with 1 output and no more than 4 inputs. This can be done graphically by drawing circles encompassing the gates of the circuit, such that each circle has 1 output and no more than 4 inputs. The circles may overlap (this amounts to a duplication of logic gates, but it is essentially free so long as it does not require any new 4-LUTs – it only affects wiring).

