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

E. Alon

Friday, Apr 7th, 2017
5:00-6:30pm

EECS 151/251A: SRPING 2017—MIDTERM 2

NAME	Last Solutions First

GRAD/UNDERGRAD	

Problem 1: _____/ 14

Problem 2: _____/ 16

Problem 3: _____/ 20

Problem 4: _____/ 20

Total: _____/ 70

PROBLEM 1. (14 pts) CMOS Power and Energy

a) **(6 pts)** Assuming that our design consists entirely of LUTs, in this problem we will examine how many such LUTs we can include in our design while staying within a given power budget. For each LUT, you can assume that the total capacitance associated with that LUT is 5fF, the activity factor associated with that capacitance ($\alpha_{0\to1}$) is 0.1, and the average leakage current is 100nA. If the design operates at a supply voltage of V_{dd} = 0.8V and the clock frequency is 1GHz, how many LUTs can the design include if we want to keep the total power under 1W?

$$P_{LUT} = \alpha_{0\to1} \cdot C_{LUT} \cdot V_{DD}^2 \cdot f_{clk} + V_{DD} \cdot I_{leak}$$

$$= 0.1 \cdot 5fF \cdot (0.8V)^2 \cdot 1GHz + 0.8V \cdot 100nA$$

$$= 400\ nW$$

$$N_{LUT} = \frac{P_{tot}}{P_{LUT}}$$

$$= \boxed{2.5 \text{ million LUTs}}$$

b) **(8 pts)** Now assuming that the leakage current of each LUT is set by $I_0 e^{\left(\frac{-V_T}{1.5\times 25mV}\right)}$, where V_T is the threshold voltage, and that increasing V_T by 100mV forces you to increase V_{dd} by 100mV (i.e., to V_{dd} = 0.9V) to maintain the same clock frequency, how many LUTs can you now include on the chip?

$$I_{leak,new} = I_{leak,old} \cdot \frac{e^{-(V_T+100mV)/(1.5\cdot 25mV)}}{e^{-V_T/(1.5\cdot 25mV)}}$$

$$= I_{leak,old} \cdot e^{-100mV/(1.5\cdot 25mV)}$$

$$\approx 100nA \cdot 0.0695 = 6.95nA$$

$$P_{new} = \alpha_{tool} C_{LUT} V_{DD,new} \cdot f_{clk} + V_{DD,new} \cdot I_{leak,new}$$

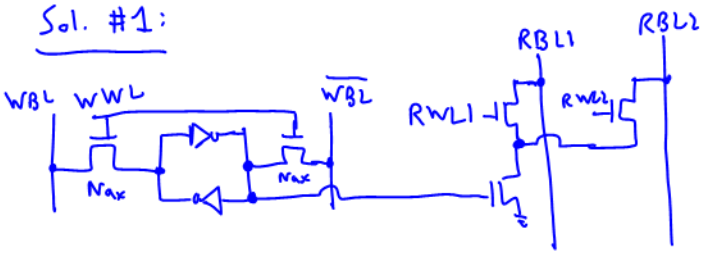
$$= 0.1 \cdot 5fF \cdot (0.9V)^2 \cdot 1GHz + 0.9V \cdot 6.95nA$$

$$\approx \boxed{2.43 \text{ million LUTs}}$$

PROBLEM 2. (16 pts) SRAMs and Decoders

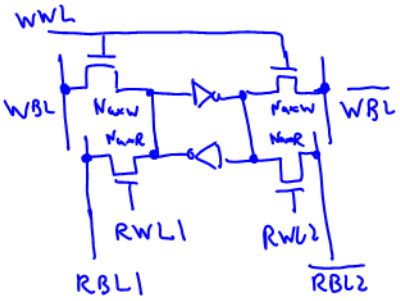
a) (**8 pts**) Using no more than 9 transistors in total, sketch a design for an SRAM cell that would support 2 independent read ports and 1 independent write port. Qualitatively explain how the transistors in your cell should be sized to ensure sufficient stability under the worst-case conditions for reading to/writing from your cell.

Sol. #1:



* For this solution, just need to make sure Nax transistor can overpower PMOS within the inverters to guarantee write stability. (Reads are totally decoupled)

Sol #2:



* For this solution, NaxW needs to overpower PMOS's in the inverters (like sol. #1).

NaxR needs to be sized so that it can never overpower the NMOS in the inverters (just like normal 6T cell).

b) **(8 pts)** Now let's look at the delay of a decoder for reading from one of the ports from an array made out of the multi-ported SRAM cell from part (a). Assuming that the array is 128 x 64 (i.e., each wordline drives 64 cells, and each bitline has 128 cells on it), that the total capacitance on one of the read wordlines from each cell is 0.25fF, that the total capacitance on each address input (you are given both address and address_b as inputs) must be less than 1fF, and that the decoder is built using only NAND2s and inverters, what is the minimum delay of the decoder for this array? You should provide your answer in terms of t_{inv}; note that you can assume that the parasitic delay of the NAND2 gates is γt_{inv} (i.e., the same as an inverter), and you can ignore the fact that you can't actually build a fractional number of stages.

128 cells / bitline → 7 bit address decode, so critical path will need 3 NAND2 gates (2 in predecode, one in final decode).

$C_{WL} = 64 \cdot C_{cell} = 16fF$ $\quad \Pi LE = (4/3)^3$

$\Pi B = 64$

$PE = \Pi LE \cdot \Pi B \cdot \frac{C_{WL}}{C_{in}} = (4/3)^3 \cdot 64 \cdot \frac{16fF}{1fF} \approx 2427.26$

$N_{stages,opt} = \log_4(PE) \approx 5.62$

$t_{p,opt} = N_{stages} \cdot t_{inv} \cdot (\gamma + EF/stage) = \boxed{5.62 \cdot t_{inv}(\gamma + 4)}$

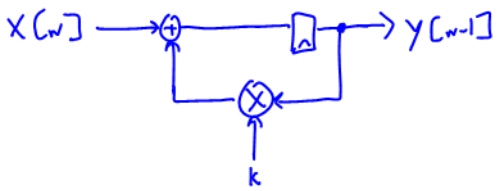
PROBLEM 3. (20 pts) Architectural Optimization

In this problem we'll consider implementing an Infinite Impulse Response (IIR) filter, whose output *y* at a given time index *n* (*y[n]*) is given by:

$y[n] = x[n] + k*y[n-1]$

where *k* is some fixed constant and *x[n]* is the input at time *n*. Throughput this problem, you are allowed to use only adders, multipliers, and registers, whose delays (or clk-q/setup for the registers) are t_{add}, t_{mult}, $t_{clk\text{-}q}$, and t_{setup}. Note also that you can assume that the input *x* to this block is produced directly by a register.

a) **(6 pts)** Sketch a simple hardware implementation of this IIR filter and provide an expression for the maximum frequency of this design.



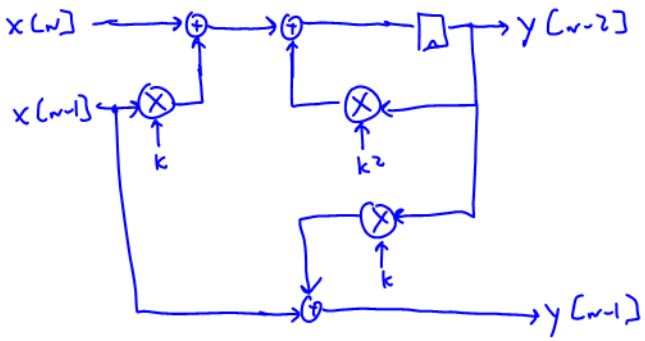
$T_{cyc,min} = t_{clk\text{-}q} + t_{mult} + t_{add} + t_{setup}$

$f_{max} = \frac{1}{T_{cyc,min}}$

b) **(8 pts)** Now sketch a new design the makes use of unrolling to compute both *y[n]* and *y[n+1]* in each cycle. Note that you can assume that both *x[n]* and *x[n-1]* are available to your block as inputs. What is the maximum clock frequency for this new design? Is the maximum achievable throughput of the unrolled design higher or lower than that of the original design from part (a)?

$y[n] = x[n] + ky[n-1]$ $\qquad$ $y[n-1] = x[n-1] + ky[n-2]$

$= x[n] + kx[n-1] + k^2 y[n-2]$



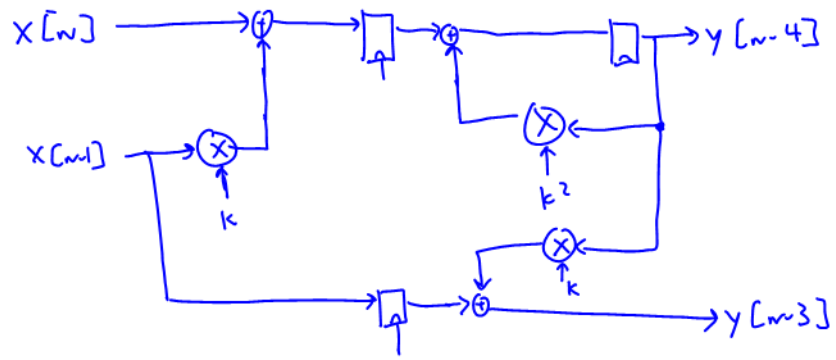
Critical path from $x[n-1]$ to register producing $y[n-2]$:

$T_{cyc,min} = t_{clk-q} + t_{mult} + 2t_{add} + t_{setup}$ $\qquad$ $f_{max} = \frac{1}{T_{cyc,min}}$

$$\frac{\text{Throughput, new}}{\text{Throughput, old}} = \frac{2 / (t_{clk-q} + t_{mult} + 2t_{add} + t_{setup})}{1 / (t_{clk-q} + t_{mult} + t_{add} + t_{setup})} = \frac{2(t_{reg+mult} + t_{add})}{t_{reg+mult} + 2t_{add}} > 1$$

Unrolled design has higher throughput

c) **(6 pts)** Modify your design from part (b) to include an additional stage of pipelining. Sketch the new design and compute the maximum clock frequency of this new (pipelined and unrolled) design.

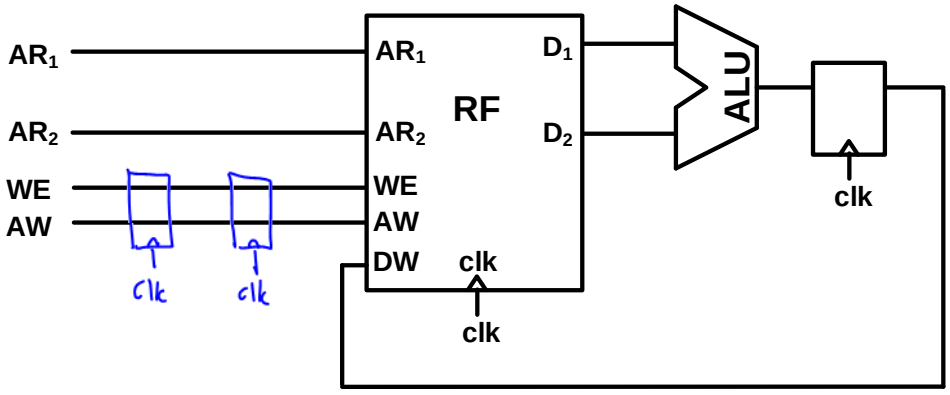


$$T_{cyc,min} = t_{clk-q} + t_{mult} + t_{add} + t_{setup}$$

$$f_{max} = \frac{1}{T_{cyc,min}}$$

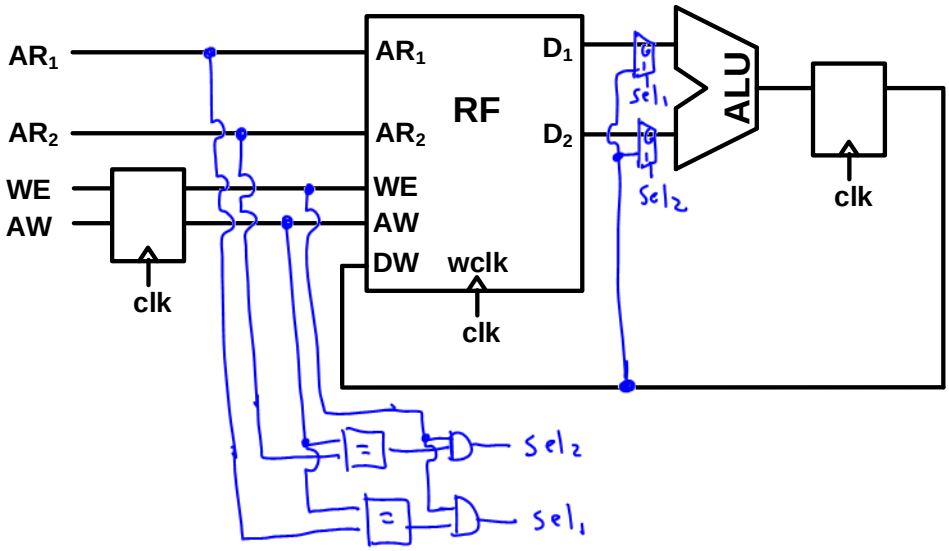
PROBLEM 4. (20 points) Processors

a) **(6 pts)** Shown below is a portion of a prototype design for a pipelined CPU's datapath that uses a register file with **synchronous reads and writes**. However, even ignoring any potential data hazards, this design does not function correctly – in particular, register type instructions (where the ALU operates on data from two registers – specified by AR_1/AR_2 – and writes back to a third – specified by AW) do not seem to execute correctly. Explain what the error is caused by and add any extra components necessary to correct the design.



* Problem is that it takes two more cycles for AR_1/AR_2 to propagate through the regfile and ALU (then back to the regfile) than it takes WE/AW to be applied to the regfile. In other words, we were originally writing the wrong data back in to the reg file. Solution is as shown above – add two registers on to the WE/AW signals to make them line up correctly with the data coming back from the ALU register.

b) **(8 pts)** Now let's examine a slightly different design shown below, where the register file has synchronous writes but **asynchronous reads**. Add appropriate forwarding hardware to eliminate all data hazards from this pipeline; you should show both the forwarding path(s) as well as the hardware you would need to detect the data hazard(s). You are only allowed to use MUXes, adders, comparators (i.e., a block that checks whether two inputs are equal to each other), and any basic logic gates (AND, OR, etc.) in the forwarding hardware you add.



c) **(6 pts)** In order to accelerate certain applications, your company has decided to augment its processor with specialized instructions and hence hardware units within the processor. After adding the new hardware unit to the processor (the unit operates on the same inputs as the ALU, and when the special instruction is called, the unit's output would replace the ALU's output) you find out that the processor's clock frequency is now substantially reduced, and is totally set by the delay of that unit. In order to try and bring the frequency back up, you decide to add two pipeline registers inside of the new unit.

Assuming that the new instruction is called MLinst, and that MLinst(R1, R2, R3) implies using the data stored in registers R1/R2 (whose addresses are specified in AR1/AR2) for D1/D2 and storing the result in R3 (whose address is specified in AW), consider the following snippet of code:

MLinst(r1, r2, r3)
MLinst(r3, r4, r5)

Is it possible to execute this code snippet without introducing any stalls in to the pipeline? If so, describe and/or sketch what the additional forwarding hardware would need to be. If not, explain why not.

Not possible – having added two registers in to the unit means that it takes at least two cycles for a new result to be available at the output of that unit. So, no matter what forwarding you add, you can't produce the new value of r3 fast enough to enable the new unit to start a new computation based off of that value one cycle later. In other words, in this case we have to introduce at least one stall.

EXTRA PAGE FOR ADDITIONAL/SCRATCH WORK