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

EECS150
Fall 2002

J. Wawrzynek
11/1/02

Exam II

Name:_____

ID number:_____

This is a *closed-book, closed-note* exam. No calculators please. You have 2 1/2 hours. Each question is marked with its number of points (one point per expected minute of time).

Put your name and SID on each page. You can work out your answers on the backs of the pages and use the extra blank sheets at the end of the booklet.
Show your work. **Write neatly** and be well organized.
Good luck!

problem	maximum	score
1	15pts	
2	10pts	
3	20pts	
4	5pts	
5	7pts	
6	7pts	
7	10pts	
8	8pts	
total	82pts	

1. [15pt] Consider the design of a 16-bit adder based on two 8-bit ripple adders and a set of 4-LUTs. The 4-LUTs implement a carry look-ahead circuit used to speed up the carry signal to the upper bits of the adder. You are given the two 8-bit ripple adders and are only allowed to use 4-LUTs to implement the remainder of the design. Your carry look-ahead circuit must generate the carry into the top half of the 16-bit adder, C_8, and the final carry out, C_{16}. The delay from *any* full-adder cell input to *any* full-adder cell output is 1ns. The delay through a 4-LUT is 1ns.

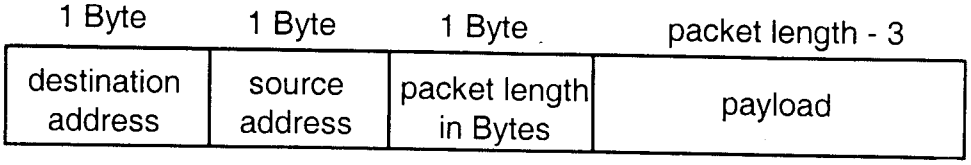
a) What is the minimum number of 4-LUTs needed in this design, ignoring the ripple adders?

b) Calculate the delay from input to the most significant bit of the output (S_{15}).

2. [10pt] Consider the design of an unsigned combinational multiplier (no flip-flops or controller) for multiplying the constant value 1101_2 by the variable X ($x_3x_2x_1x_0$). Using only full-adder cells, draw a circuit that implements the multiplier, minimizing the total amount of hardware and the delay from input to output. *Hint: think about carry-save addition.*

Draw all signal wires without using busses (all wires must have width of 1). Do not show the internal details of your full-adder cells, but label the S and C outputs. Label all inputs to your multiplier and label your outputs as p_i, where the subscript i refers to the bit position.

3. [20pt] For this problem you need to design a circuit for processing packets as they are received from a network. Assume that packets are received from the network continuously one after the other with no pauses, preamble bits or start of frame symbols in between. Packets are received one byte at a time (the bits and nibbles are *not* reversed). The first Byte of the first packet is received on the cycle immediately following reset (i.e., during the cycle that the reset signal goes low, the first packet Byte appears). Unlike the packet in our project, these packets have a variable length payload. The packet format is as follows:



The payload is at least 1 Byte in length.

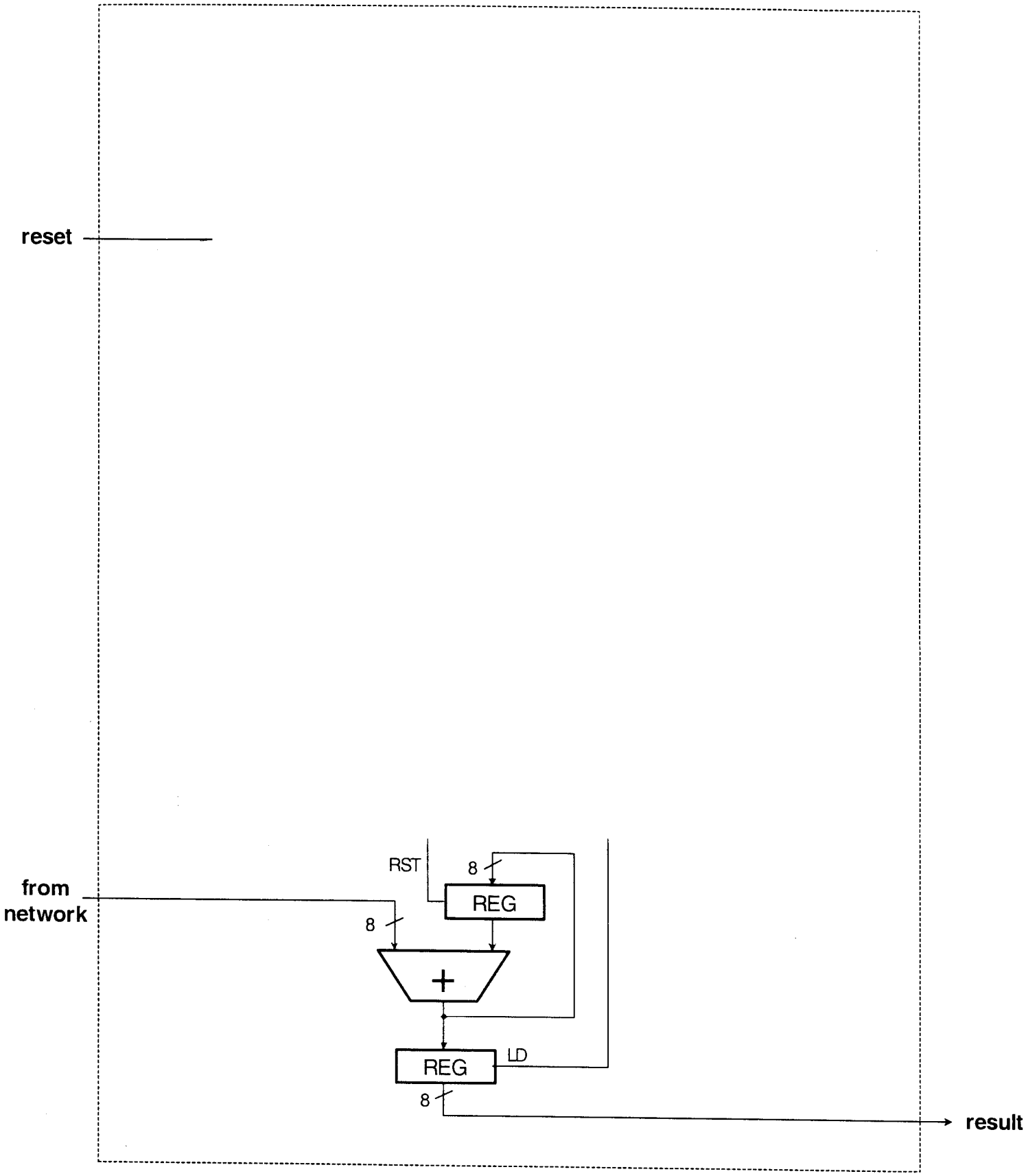
The function of the packet processor is to add up all the bytes of the payload leaving the result in a register. The final sum must remain in the result register during the time the next packet is being received and until a new final sum is ready. A partial design is shown below. Your job is to complete the design, drawing in more circuitry. You may use the following elements:

Counter	Binary counter of any width with optional reset (RST), enable (CE), and terminal count (TC) signals.
=	Equal comparator.
REG	Register with optional load enable (LD) and reset (RST).
Simple logic gates with any number of inputs	
FF	D-flip-flops with optional set (S) and reset (R) inputs.

Avoid the use of logic gates and flip-flops; use the other elements when possible.

Do not modify the partial design, only add to it. Neatly draw in your circuitry below. Label each element with its type (i.e., Counter, =, REG, ...) and the inputs and outputs that you use. Label the width of all data busses (wires with width greater than 1). You do not need to draw the clock signal.

First draw you solution on scratch paper. Once you are satisfied with it, copy it over neatly into the diagram below. It also might help you to draw a timing diagram on your scratch paper to help you understand the design requirements.


reset
from
network
RST
8
REG
8
+
REG
LD
8
result

4. [5pt] Recall that binary-coded-decimal (BCD) is a way to represent the decimal digits using 4 bits; $0_{10} = 0000$, $1_{10} = 0001$, etc. Using a read only memory (ROM) and a register with no reset input, sketch the design of a single digit BCD down counter. Your counter must not have a reset signal and must be self-starting (within a very few numbers of cycles from power-up the counter must start down counting). Draw your circuit in the space provided. As usual, be neat, label all inputs and outputs, and widths of data busses. Show the contents of your ROM in the table provided (you may delete or add table rows).

ROM contents:

ADDRESS	VALUE

Circuit Diagram:

5. [7pt] A 1 Mbit SDRAM has a row size of 32 32-bit words (1K-bits) and 1K rows. Memory addresses are assigned in row-major order so that memory locations 0 through 31 are in the first row, locations 32 through 63 in the second row, etc. A SDRAM data access (read or write) requires 4+L clock cycles, where L is the length of the burst. With this SDRAM you can set the burst length, L, to any value you wish for each read and write operation.

We use the SDRAM in a system to store the contents of a 2-dimensional matrix of 32-bit integers. The matrix is 20 rows by 48 columns. The matrix is stored in consecutive memory locations in row-major order (memory locations 0 through 47 hold the first row of the matrix, memory locations 48 through 95 hold the second row of the matrix, etc.).

A controller writes the matrix into memory by columns; it writes the words of a single column then moves onto the next column until the entire matrix is written. The controller supplies words to the memory at what every rate the SDRAM will accept them. After the entire matrix is written, the controller reads the matrix by rows; it reads all the words of a single row before moving on to the next - again at whatever rate the SDRAM will produce them.

Calculate the minimum total number of cycles needed to completely write then completely read the matrix. Show your work. Your final answer must not be in terms of L.

6. [7pt] Suppose we have implemented a 8-bit wide FIFO of 900 locations using a two-ported memory module. Part of the design is a pair of 10-bit binary up-counters used to keep track of the read and write positions (RP and WP respectively). Your job is to add a new output to the FIFO, called SPACE, that at all times when the FIFO is not full, indicates the number of available locations in the FIFO.

a) Write a Verilog module that generates SPACE as an output, based on RP and WP.

b) Sketch the circuit that generates SPACE, based on RP and WP. Do not show the memory module or other FIFO details.

7. [10pt] Consider the design of a 4-bit synchronous binary counter implemented with combinational logic and four D-flip-flops, each with clock enable (CE), reset (RST), clock (CLK), and data (D) inputs. The counter must have a count enable input (CE), terminal count output (TC), and four data outputs $q_3\ q_2\ q_1\ q_0$.

a) Write the logic equations for each of the flip-flop data inputs $d_3\ d_2\ d_1\ d_0$ and TC. Remember the flip-flops have a CE input.

b) What is the minimum number of 4-LUTs needed to implement the combinational logic part of this counter?

8. [8pt] Short answer.

a) [1pt] An optimally designed carry-select adder (with constant group size) has delay proportional to what function of the input width, n?

b) [1pt] Assuming 1Byte for each R, G, or B value and each Y, Cb, and Cr value of every pixel, what is the relative bandwidth savings (in percentage) by sending images as 4:2:0 YcbCr versus 4:4:4 RGB?

c) [1pt] A "CIF" frame size is 352 x 288 pixels. What is the size in bytes of a CIF frame buffer?

d) [1pt] A 512 x 2-bit memory organized as a square array uses how many address bits for the row-decoder?

e) [1pt] How many address bits for the column-mux control?

f) [1pt] Flash memory is most closely related to (DRAM, SRAM, EPROM)?

g) [1pt] Based on a D-flip-flop, draw a toggle flip-flop (with T and CLK inputs):

h) [1pt] Based on toggle flip-flops, draw a 4-bit "ripple counter":