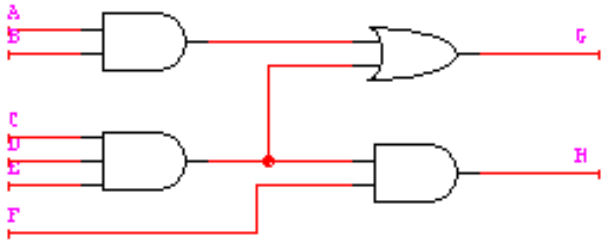
CSE370 Fall '99
Assignment 4
Distributed: 10/18/99
Due: 10/25/99

Reading:
Katz, Chapter 3, sections 3.1.2, 3.2.3, 3.4.5, 3.5 are optional

1. **Technology Mapping (3 pts)**

Functions G and H are implemented as show below. Given 4 2-input NAND gates and 4 2-input NOR gates, and no other hardware, design and draw a schematic for the logic diagram shown below. For this exercise, do not assume that complemented inputs are available. Show your intermediate steps.



2. **Karnaugh Map Analysis (4pts)**

Draw the Karnaugh Map for the function F = Σm(0,2,4,5,6,7,9,11,13,15)

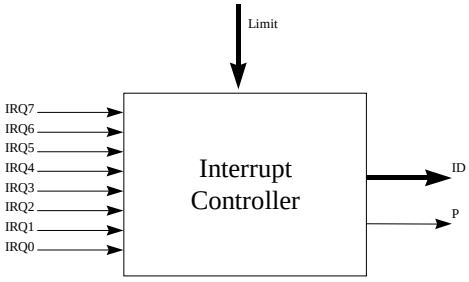
a. How many Implicants are there in this K-map?
b. Identify each Prime Implicant, and write the corresponding product term
c. Identify each Essential Prime Implicant and write the corresponding product term
d. Write every minimum SOP expression for F.
e. Choose one of the above min SoP expressions and identify a single bit static-1 hazard if one exists. (A word on notation: "ABCD: 1111→ 1110" describes a transition on ABCD from 1111 to 1110)
f. Modify the circuit to eliminate all single bit static-1 hazards.

2. **Timing Circuits (3 pts)**

Assuming that all gates have a delay of 10 time units, design a circuit to produce a continuously oscillating output with a period of 100 time units. The output must be high for 50 time units, and low for 50 time units during each cycle. The circuit is controlled by an input Disable. When Disable=1, the output is 0, when Disable = 0 the output is oscillating as specified above.

3. **Combination Logic System Design (10 pts)**

An interrupt is a signal that suspends the normal execution of a program so that a processor can rapidly respond to external events such as keyboard strokes or the arrival of a new packet of streaming media from the network. System programmers like to be able set interrupt priority limits so that high priority tasks can be completed without interruption by lower priority tasks.
The interrupt controller performs these two functions:



- Function 1: Translate eight individual interrupt request lines (IRQ0-IRQ7) into a single 3 bit number that represents the ID of the highest priority interrupt that is currently asserted. Higher priority interrupts have higher IRQ numbers. The output ID = 0 if there are no interrupts or if the current highest priority interrupt request is on IRQ0.
- Function 2: Assert the pending interrupt signal "P" if the current highest priority interrupt is equal to, or higher, than the priority limit set by the programmer (provided as input). P = 0 if there are no pending interrupts or if the highest priority request is lower than the limit.

Please design the interrupt controller shown in the figure by following these steps:

a) Draw a block diagram of the interrupt controller that shows each of the sub-circuits that will be designed independently. **None of your sub-circuits can have more than 4 inputs**. Choose a name for each sub-circuit, and write an English specification for each that defines the inputs, outputs and functionality. The English specification should be complete enough to hand-off to a different designer and may include a truth table if necessary. It is okay if your block diagram requires a few individual gates to tie things together.
b) Design each sub-circuit using any of the techniques presented so far, and create a DW schematic for each. Make sure your design steps are clearly shown. Turn in the DW schematic for each.
c) Turn in the top-level view of your hierarchical schematic in DW.
d) Turn in your DW spreadsheet showing the inputs the verified expected outputs for the following cases:
    - IRQ7-IRQ0 = 00000000, Limit = 0
    - IRQ7-IRQ0 = 00001101, Limit = 2
    - IRQ7-IRQ0 = 00001101, Limit = 3
    - IRQ7-IRQ0 = 00001101, Limit = 4
    - IRQ7-IRQ0 = 10100010, Limit = 6
    - IRQ7-IRQ0 = 10000000, Limit = 7
    - IRQ7-IRQ0 = 00010001, Limit = 0
    - IRQ7-IRQ0 = 00000001, Limit = 1

Justification: limiting the sub-circuits to 4 inputs is actually a realistic design constraint. First, it makes it easy for us to design using K-maps. Second, many programmable logic devices are made of small blocks that can be programmed to implement any function of a small number of inputs. These are called Field Programmable Gate Arrays (FPGAs), which are very popular for rapid prototyping of digital systems. We will cover these briefly in lecture.

Here are some suggestions:

1. Partition the problem so that you can give simple, meaningful names to intermediate signals and sub-circuits. The functionality or interpretation for each intermediate signal or sub-circuit should have simple English descriptions. This will tend to make your blocks easier to specify and require less logic.
2. Write a little psuedo-code to help you visualize how the you have partitioned it, and what each block must do.
3. Look for opportunities to use the same sub-circuit more than once.