CMSC 313 Lecture 28

- Last Time: Registers & Memory
- DigSim2: 1 day extension
  - Remember to place your username on the paper submission
  - Hard copy submission only
- Final Exam: Tuesday 12/16, 10:30am - 12:30pm
- Semester Review
- Sample Final Exam

DigSim Assignment 2: Finite State Machine Simplifications

Due: Tuesday December 9, 2003

Objective

The objective is to design and simplify a moderately complex a finite state machine.

Assignment

For this project you will design and implement in DigSim a Mealy finite state machine with one input bit x and one output bit z. The machine's output z is 1 whenever the input sequence …0111 or …1000 has been detected. The patterns may overlap. For example, if the machine is given input 0111000, the output would be 0001001. [Adapted from *Contemporary Logic Design*, Randy H. Katz, Benjamin/Cummings Publishing, 1994.]

In order to complete the project, you must accomplish the following tasks. Note that for this project, you will submit both your design work (on paper) and the DigSim implementation of the resulting finite state machine (via submit).

1. Draw a transition diagram for the finite state machine described above. Use the state reduction algorithm to minimize the number of states in your machine. Although, you might start with an FSM with as many as 15 states, at the end of the reduction process you should have 7 states in your machine. Name your states A, B, C, D, E, F and G. You may take shortcuts, but you must show your work.
2. Assign bit patterns to each state of the machine. Use the heuristics for state assignment presented in class (available on the course lecture topics web page). Do this step carefully as it will have a large effect on the complexity of your final circuit. *Hint:* follow the loops in the FSM and try to assign bit patterns such that in each step of the loop only one state bit changes. You might not be able to achieve this for *every* step of the loop, but you can try to have only one state bit change for *most* steps of the loop. Write down your final state assignment in the table given.
3. Fill in the state transition table for the finite state machine given below. In this table, s2, s1 and s0 are the state bits, x is the input bit and z is the output bit. Then s2', s1' and s0' are the next states to be stored in the flip flops.
4. Use the Karnaugh maps provided to simplify the Boolean formulas for a finite state machine to be implemented using D flip-flops. You will need Karnaugh maps for s2', s1', s0' and z.
5. Using the excitation table for J-K flip-flops, fill in the columns j2, k2, j1, k1, j0 and k0 in the truth table below. The columns j2 and k2 represent the settings to the inputs of the J-K flip-flop that will cause it to store the state bit s2'. The columns j1, k1, j0 and k0 are analogous for the J-K flip-flops used to store state bits s1 and s0.
6. Use the Karnaugh maps provided to simplify the Boolean formulas for the J and K inputs to each J-K flip flop. You will need Karnaugh maps for j2, k2, j1, k1, j0 and k0. The formula for the output z will be the same as in Step 4.
7. Decide whether you want to use D flip-flops or J-K flip-flops. Briefly justify your choice.
8. Implement the resulting circuit in DigSim. Hint*:* it is possible to implement this FSM using fewer than 14 gates. If your circuit requires many more gates, redo the simplification steps.

What to submit

In class on Tuesday December 9, turn in your finite state diagram, truth table, Karnaugh maps and the resulting Boolean formulas on paper. Make copies of these, if you still need them to implement your circuit in DigSim.

Save your circuit as you did in DigSim Assignment 1. Submit the circuit file using the Unix submit command as in previous assignments. The submission name for this assignment is: `digsim2`.

CMSC 313 Digsim Exercise 2

Name: _____

Minimized 7-State Transition Diagram (show work)

State Assignment:

A	
B	
C	
D	
E	
F	
G	
unused	

Excitation Table for J-K Flip-Flops

Q	Q'	J	K
0	0	0	*d*
0	1	1	*d*
1	0	*d*	1
1	1	*d*	0

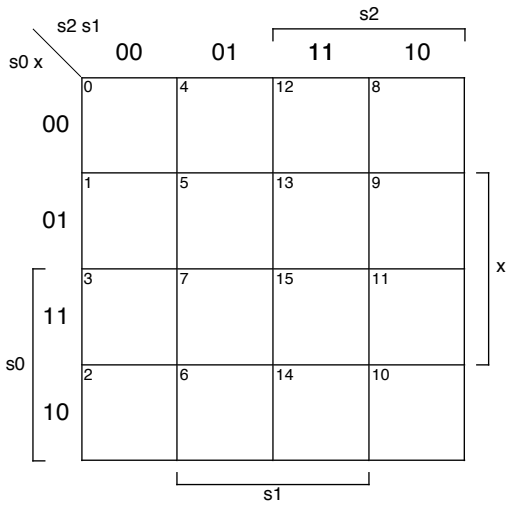
Truth Table:

	s2	s1	s0	x	s2'	s1'	s0'	z	j2	k2	j1	k1	j0	k0
0	0	0	0	0										
1	0	0	0	1										
2	0	0	1	0										
3	0	0	1	1										
4	0	1	0	0										
5	0	1	0	1										
6	0	1	1	0										
7	0	1	1	1										
8	1	0	0	0										
9	1	0	0	1										
10	1	0	1	0										
11	1	0	1	1										
12	1	1	0	0										
13	1	1	0	1										
14	1	1	1	0										
15	1	1	1	1										

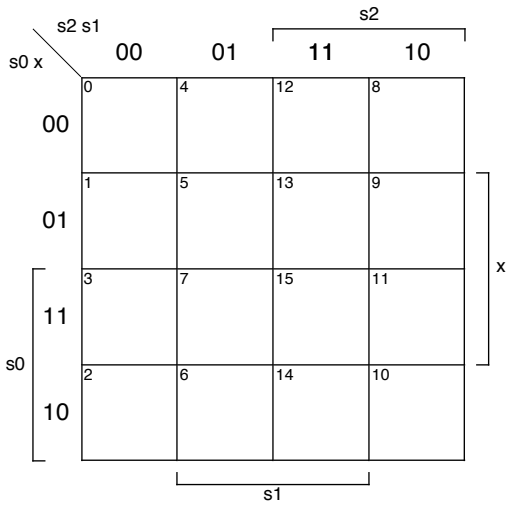
Question:

Should you use D flip-flops or J-K flip-flops to implement this circuit? Why?

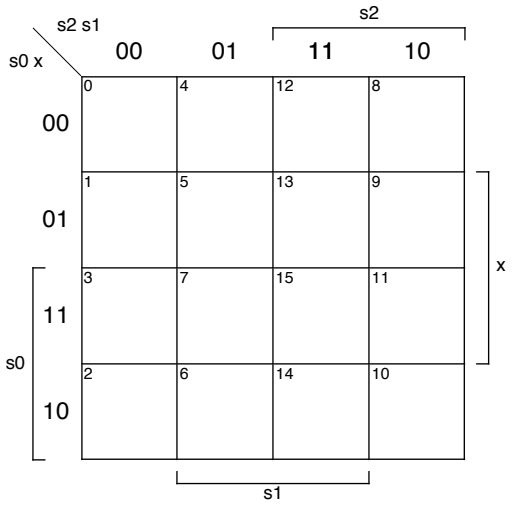
Karnaugh Maps for D Flip-Flops and the output



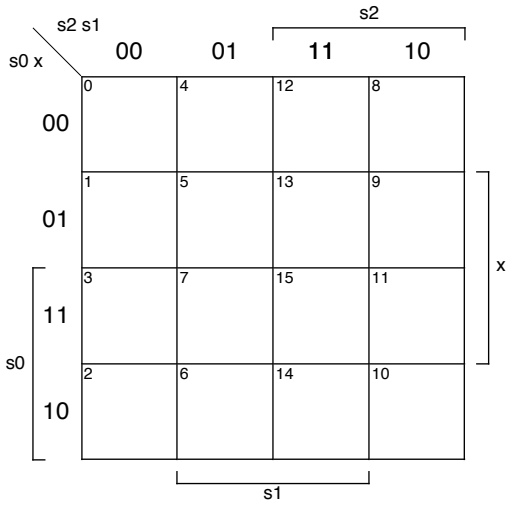
s2' =



s1' =

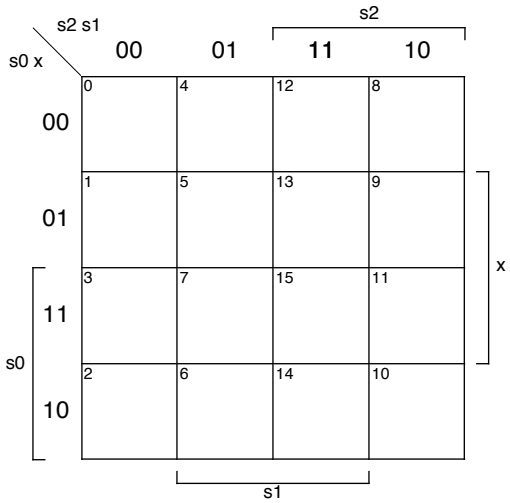


s0' =

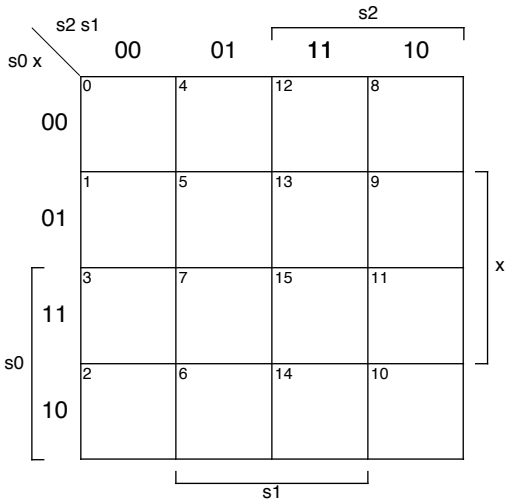


z =

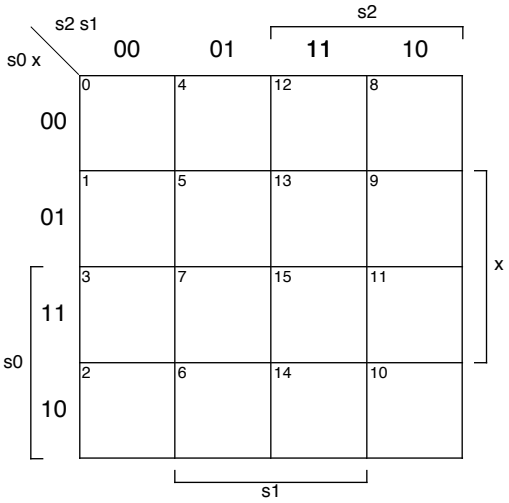
Karnaugh Maps for J-K Flip-Flops



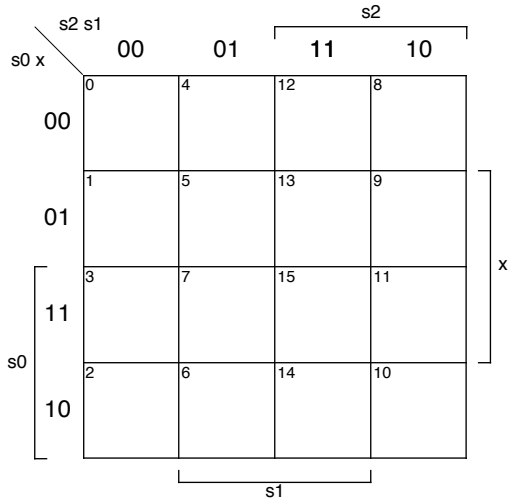
j2 =



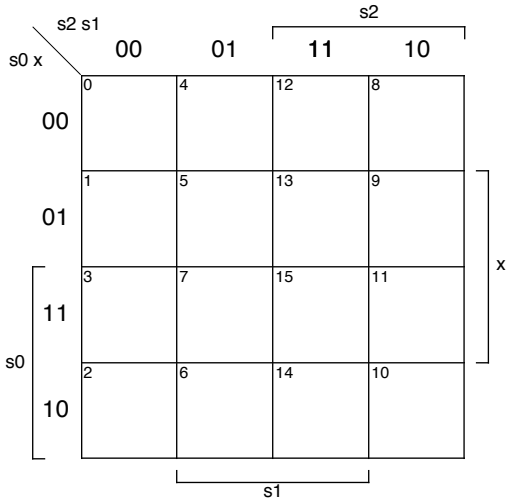
k2 =



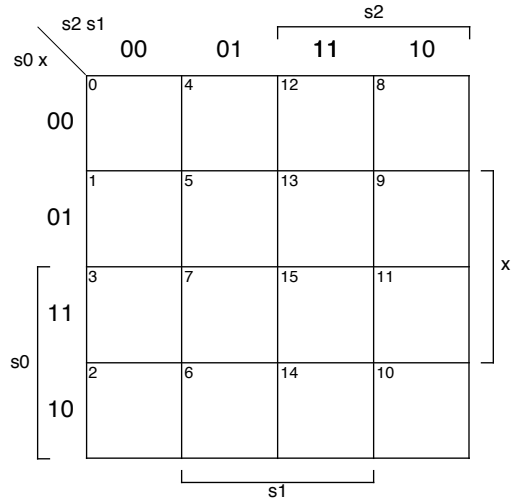
j1 =



k1 =



j0 =



k0 =

Chapters of M&H

1. Introduction [system bus model, levels of machine]
2. Data Representation
3. Arithmetic
4. Instruction Set Architecture [used Intel chip]
5. Languages & The Machine [compiling, assembling, linking & loading]
6. Datapath & Control [skipped, covered in CMSC411]
7. Memory
8. Input & output [skipped]
9. Communication [skipped]
10. Trends in Computer Architecture [easy read]

Appendix A: Digital Logic

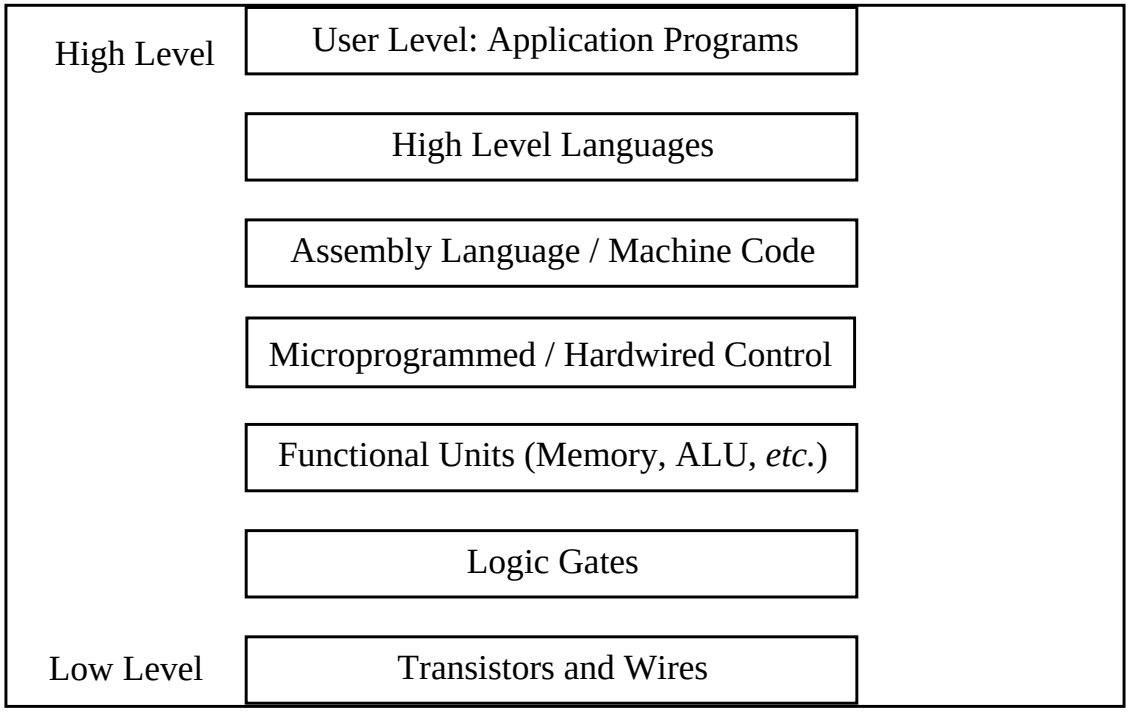
Appendix B: Reduction of Digital Logic

Themes

- Levels of machines
- Modularity
- Models

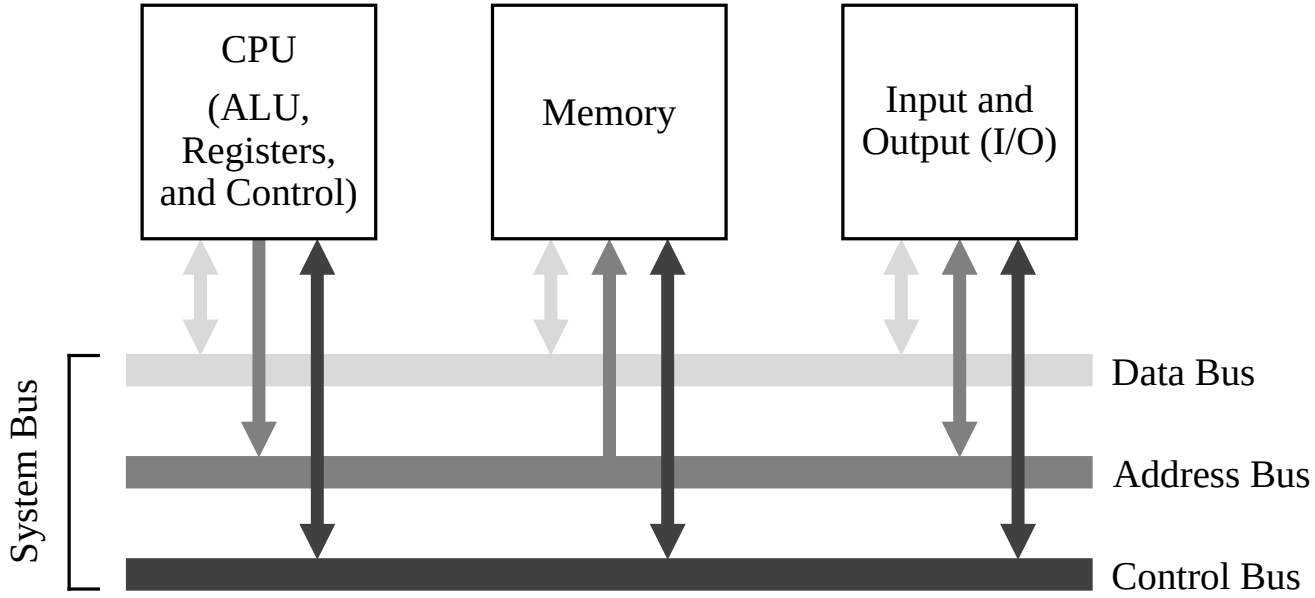
Levels of Machines

- **There are a number of levels in a computer (the exact number is open to debate), from the user level down to the transistor level.**
- **Progressing from the top level downward, the levels become less abstract as more of the internal structure of the computer becomes visible.**



The System Bus Model

- **A refinement of the von Neumann model, the system bus model has a CPU (ALU and control), memory, and an input/output unit.**
- **Communication among components is handled by a shared pathway called the *system bus*, which is made up of the data bus, the address bus, and the control bus. There is also a power bus, and some architectures may also have a separate I/O bus.**



Concentrated On

- **Basic Assembly Language Programming**
  - ◇ **Did RISC-style programming on a CISC chip**
- **Optimizing Digital Logic**

Glossed Over

- Floating Point Instructions
- Input-Output
- Combinational Logic Components
- Programmable Logic Arrays

What's Next

- **CMSC411 Computer Architecture**
  - Design a CPU using VHDL
- **CMSC421 Operating Systems**
  - Includes more on virtual memory and caching
- **CMSC431 Compiler Design**
  - Write a compiler from scratch for a baby programming language
- **CMSC451 Automata Theory**
  - Theorems about finite state machines, context-free grammars, …