
EE108A

Lecture 10: Microcode

Announcements

- Problem set 3 due
- Lab 4 hints
 - **Do a good *clean* design**
 - Lab 5 and the final project will reuse most of this lab
 - You'll regret any mistakes here for the rest of the quarter
 - We've detailed the interfaces to try and make this easier
 - **Understand how the lab works**, not just how to build it
 - How sine waves are generated (handout)
 - How to make FSM Timing Diagrams to deal with memories and data paths (last section of lab handout)
 - How all the modules interact (**a good block diagram really helps**)
 - **Split up module design and testing**
 - Decide on how a module should work as a group
 - Have one person design it and the other person write the test bench
 - Finds errors in design and description
 - **Watch your timing**
 - Check your synthesis reports to make sure you are meeting timing (If your critical path is too long your design **invalid**.)
 - Run the timing analyzer and find the paths that are too slow
 - Find those paths on your block diagram and figure out where to insert pipeline registers to split up the long paths without losing synchronization. (Hint: before and after multipliers, but watch out for control paths!)

Microcode – an FSM realized with a memory array

- Original concept by Wilkes (1951)
 - Put state table in a memory (ROM or RAM)
 - ROM Address = {current state, inputs}
 - ROM Output = {next state, outputs}

Effectively replace the “combinational next state logic” block with the full truth table.

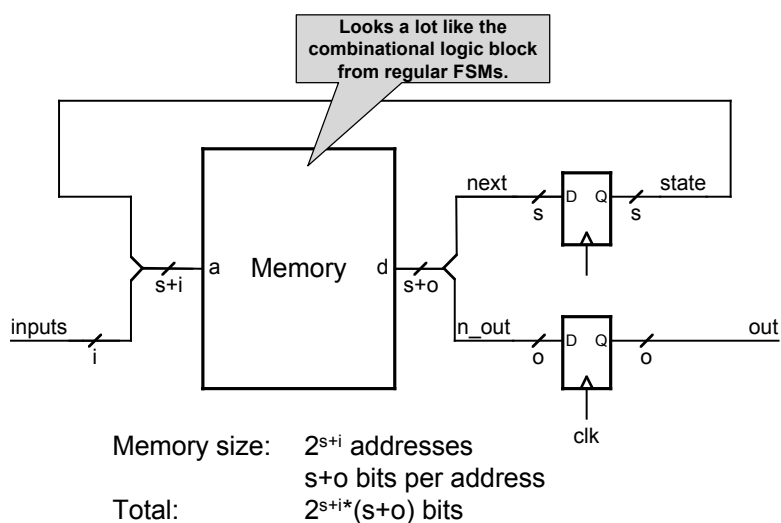
Why is this good? Why might this be bad?

10/24/07

EE 108A Lecture 9 (c) 2006 W. J. Dally and D. Black-Schaffer

3

Microcode – the picture

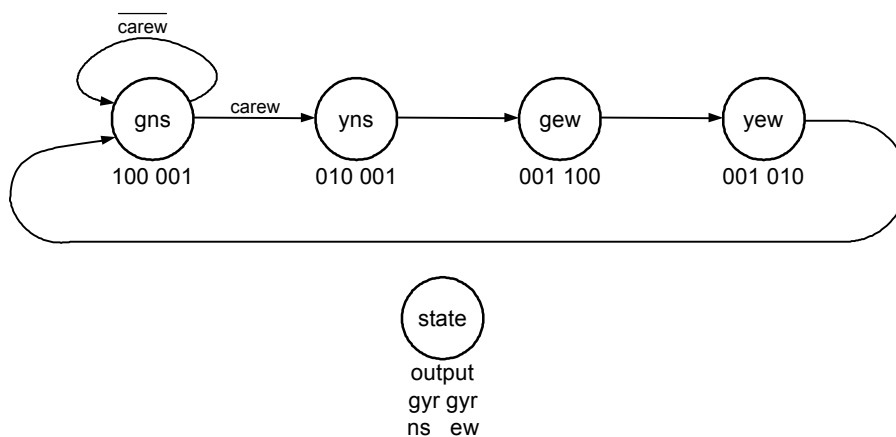


10/24/07

EE 108A Lecture 9 (c) 2006 W. J. Dally and D. Black-Schaffer

4

Example: Simple Light-Traffic Controller



10/24/07

EE 108A Lecture 9 (c) 2006 W. J. Dally and D. Black-Schaffer

5

Re-writing State Table Of Example

ROM Address: {**state**, inputs}

ROM Output: {**next state**, outputs}

	State	Next State		Output
		!carew	carew	
gns	00	00	01	100001
yns	01	11	11	010001
gew	11	10	10	001100
yew	10	00	00	001010

	State	carew	address	data
gns	00	0	000	00100001
	00	1	001	01100001
yns	01	0	010	11010001
	01	1	011	11010001
gew	11	0	110	10001100
	11	1	111	10001100
yew	10	0	100	00001010
	10	1	101	00001010

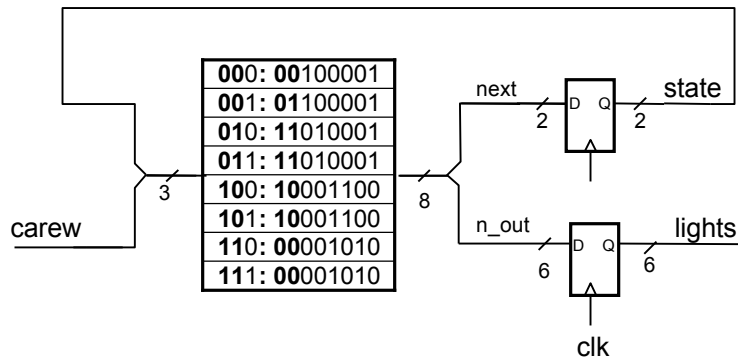
Note: We've enumerated every possible combination of inputs and state. Is this efficient?

10/24/07

EE 108A Lecture 9 (c) 2006 W. J. Dally and D. Black-Schaffer

6

Microcode Of Light-Traffic Controller



10/24/07

EE 108A Lecture 9 (c) 2006 W. J. Dally and D. Black-Schaffer

7

Microcoded Traffic Light Controller – in Verilog

```

module ucodeTLC(clk,rst,in,out) ;
  parameter n = 1 ; // input width
  parameter m = 6 ; // output width
  parameter k = 2 ; // bits of state

  input  clk, rst ;
  input  [n-1:0] in ;
  output [m-1:0] out ;

  wire  [k-1:0] next, state ;
  wire  [k+m-1:0] uinst ;

  DFF #(k) state_reg(clk, next, state) ; // state register
  DFF #(m) out_reg(clk, uinst[m-1:0], out) ; // output register
  ROM #(n+k,m+k) uc({state, in}, uinst) ; // microcode store
  assign next = rst ? {k{1'b0}} : uinst[m+k-1:m] ; // reset state
endmodule

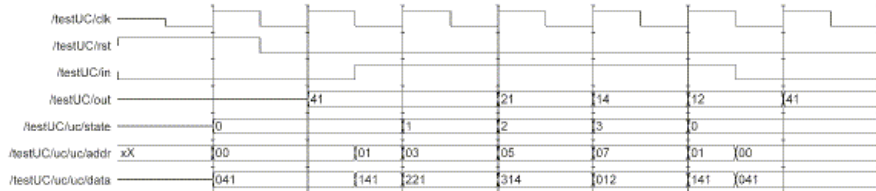
```

10/24/07

EE 108A Lecture 9 (c) 2006 W. J. Dally and D. Black-Schaffer

8

Waveforms Of Light-Traffic Controller Microcode



```

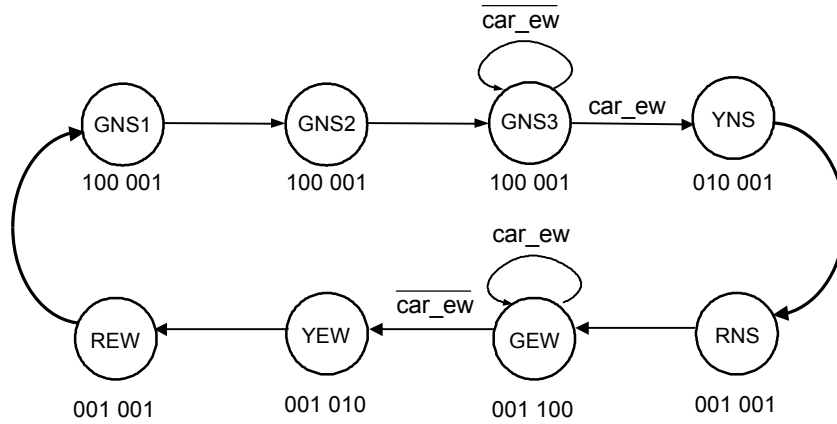
00100001
01100001
10010001
10010001
11001100
11001100
00001010
00001010
    
```

A New-and-Improved Light-Traffic Controller

Additional functions to original Light-Traffic Controller:

- Light stays green in east-west direction as long as car_ew is true.
- Light stays green in north-south direction for a minimum of 3 states (GNS1, GNS2, and GNS3).
- After a yellow light, lights should go red in both directions for 1 cycle before turning new light green.

Light-Traffic Control State Diagram



10/24/07

EE 108A Lecture 9 (c) 2006 W. J. Dally and D. Black-Schaffer

11

Light-Traffic Controller State Microcode

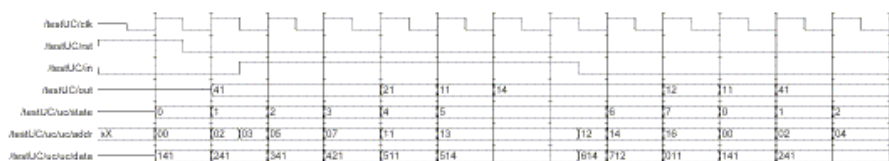
Address	State	car_ew	Next State	Output	Data
0000	GNS1(000)	0	GNS2(001)	100001	001100001
0001	GNS1(000)	1	GNS2(001)	100001	001100001
0010	GNS2(001)	0	GNS3(010)	100001	010100001
0011	GNS2(001)	1	GNS3(010)	100001	010100001
0100	GNS3(010)	0	GNS3(010)	100001	010100001
0101	GNS3(010)	1	YNS (011)	100001	011100001
0110	YNS (011)	0	RNS (100)	010001	100010001
0111	YNS (011)	1	RNS (100)	010001	100010001
1000	RNS (100)	0	GEW (101)	001001	101001001
1001	RNS (100)	1	GEW (101)	001001	101001001
1010	GEW (101)	0	YEW (110)	001100	110001100
1011	GEW (101)	1	GEW (101)	001100	101001100
1100	YEW (110)	0	REW (111)	001010	111001010
1101	YEW (110)	1	REW (111)	001010	111001010
1110	REW (111)	0	GNS (000)	001001	000001001
1111	REW (111)	1	GNS (000)	001001	000001001

10/24/07

EE 108A Lecture 9 (c) 2006 W. J. Dally and D. Black-Schaffer

12

Waveforms Of Light-Traffic Controller Microcode



```

001100001    101001001
001100001    101001001
010100001    110001100
010100001    101001100
010100001    111001010
011100001    111001010
100010001    000001001
100010001    000001001

```

10/24/07

EE 108A Lecture 9 (c) 2006 W. J. Dally and D. Black-Schaffer

13

Instruction Sequencing

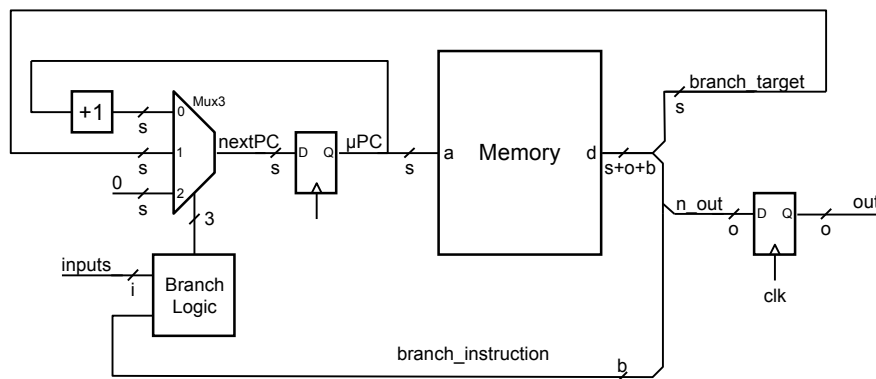
- With lot of inputs, size of memory increases rapidly (exponentially).
- Can reduce memory size by observing:
 - Most of the time we move to the next state.
 - We usually only need to *branch* to one other state based one (or a few) inputs.
- Add a microprogram counter (μ PC) register to simplify state sequencing. (Think of the PC as the “current state” of the FSM.)
 - Next state is now μ PC+1 unless we are branching

10/24/07

EE 108A Lecture 9 (c) 2006 W. J. Dally and D. Black-Schaffer

14

Instruction Sequencing – the picture



10/24/07

EE 108A Lecture 9 (c) 2006 W. J. Dally and D. Black-Schaffer

15

Branching Logic

- Branch logic selects between $\mu\text{PC}+1$ and branch_target , depending on input and $\text{branch_instruction}$.
- Instructions are of the form *branch if $f(\text{inputs})$*
 - For example *branch if car_ew* or *branch if not car_ew*

NOTE: The example has changed here.

Inputs are now car East/West and Left Turn for North/South.

Outputs are now: NS{gyr}LT{gyr}EW{gyr}

Logic: stay green East/West until a Left Turn input comes in.

10/24/07

EE 108A Lecture 9 (c) 2006 W. J. Dally and D. Black-Schaffer

16

Branch Microinstructions

Opcode	Encoding	Description
NOP	000	No branch – always go to next instruction
br	100	Always branch
brlt	001	Branch when left-turn car is detected
brnlt	101	Branch if no left-turn car is detected
brew	010	Branch when east-west car is detected
brnew	110	Branch if no east-west car is detected

10/24/07

EE 108A Lecture 9 (c) 2006 W. J. Dally and D. Black-Schaffer

17

Microcode Of Traffic-Light Controller With Branches

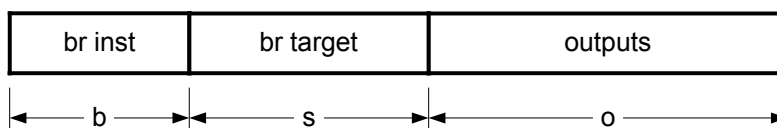
State	Addr	Inst	Target	Output	
nsg1	0000	brlt	lt1	100001001	green ns
nsg2	0001	brnew	nsg1	100001001	green ns
ew1	0010	nop		010001001	yellow ns
ew2	0011	brew	ew2	001001100	green ew
ew3	0100	br	nsg1	001001010	yellow ew
lt1	0101	nop		010001001	yellow ns
lt2	0110	brlt	lt2	001100001	green lt
lt3	0111	br	nsg1	001001010	yellow lt

10/24/07

EE 108A Lecture 9 (c) 2006 W. J. Dally and D. Black-Schaffer

18

Microinstruction Format



For our example:

$b = 3$

$s = 4$

$o = 9$

10/24/07

EE 108A Lecture 9 (c) 2006 W. J. Dally and D. Black-Schaffer

19

Implementing Microcode Of Light-Traffic Controller With Branches Using Verilog

```

module ucodeIS(clk,rst,in,out) ;
  parameter n = 2 ; // input width
  parameter m = 9 ; // output width
  parameter k = 4 ; // bits of state
  parameter j = 3 ; // bits of instruction

  input  clk, rst ;
  input  [n-1:0] in ;
  output [m-1:0] out ;
  wire  [k-1:0] nupc, upc ; // microprogram counter
  wire  [j+k+m-1:0] uinst ; // microinstruction word
  // split off fields of microinstruction
  wire  [m-1:0] nxt_out ; // = uinst[m-1:0] ;
  wire  [k-1:0] br_upc ; // = uinst[m+k-1:m] ;
  wire  [j-1:0] brinst ; // = uinst[m+j+k-1:m+k] ;
  assign {brinst, br_upc, nxt_out} = uinst ;

  DFF #(k) upc_reg(clk, nupc, upc) ; // microprogram counter
  DFF #(m) out_reg(clk, nxt_out, out) ; // output register
  ROM #(k,m+k+j) uc(upc, uinst) ; // microcode store

  // branch instruction decode
  wire branch = (brinst[0] & in[0] | brinst[1] & in[1]) ^ brinst[2] ;
  // sequencer
  assign nupc = rst ? {k{1'b0}} : branch ? br_upc : upc + 1'b1 ;
endmodule

```

10/24/07

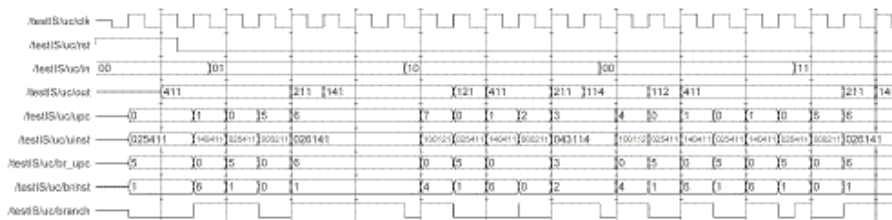
EE 108A Lecture 9 (c) 2006 W. J. Dally and D. Black-Schaffer

20

Microcode Of Light-Traffic Controller With Left Turn

Address	State	Br Inst	Target	NS LT EW	Data
0000	NS1	BLT (001)	LT1(0101)	100001001	0010101100001001
0001	NS2	BNEW (110)	NS1(0000)	100001001	1100000100001001
0010	EW1	NOP (000)		010001001	0000000010001001
0011	EW2	BEW (010)	EW2(0011)	001001100	0100011001001100
0100	EW3	BR (100)	NS1(0000)	001001010	1000000001001010
0101	LT1	NOP (000)		010001001	0000000010001001
0110	LT2	BLT (001)	LT2(0110)	001100001	0010110001100001
0111	LT3	BR (100)	NS1(0000)	001010001	1000000001010001

Waveforms Of Light-Traffic Controller With Left Turn Microcode



```

0010101100001001
1100000100001001
0000000010001001
0100011001001100
1000000001001010
0000000010001001
0010110001100001
1000000001010001
    
```

Alternate Microcode For Light-Traffic Controller With Left Turn

Address	State	Br Inst	Target	NS LT EW	Data
0000	NS1	BNA (111)	NS1(0000)	100001001	1110000100001001
0001	NS2	BLT (001)	LT1(0100)	010001001	0010100010001001
0010	EW1	BEW (010)	EW1(0010)	001001100	0100010001001100
0011	EW2	BR (100)	NS1(0000)	001001010	1000000001001010
0100	LT1	BLT (001)	LT1(0100)	001100001	0010100001100001
0101	LT2	BR (100)	NS1(0000)	001010001	1000000001010001

Previous microcode has:

- 2 states with GNS light (NS1 & NS2)
- 2 states with YNS light (EW1 & LT1)

By adding a new branch instruction – BNA:

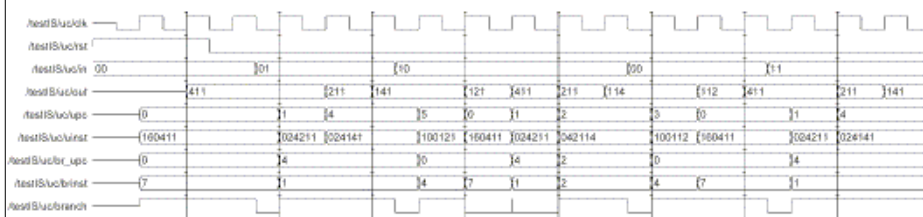
- 1 state with GNS light (NS1)
- 1 state with YNS light (NS2)

10/24/07

EE 108A Lecture 9 (c) 2006 W. J. Dally and D. Black-Schaffer

23

Waveforms Of Alternate Microcode



1110000100001001

0010100010001001

0100010001001100

1000000001001010

0010100001100001

1000000001010001

10/24/07

EE 108A Lecture 9 (c) 2006 W. J. Dally and D. Black-Schaffer

24

Multiple Instruction Types

- For some FSMs the micro-instruction word can start getting a bit long.
- To shorten it, we observe:
 - Not every state needs a branch
 - Not every output changes on every state
- Define instruction that does just a branch or a load of one register:
 - brx – 1yyyvvvv - branch to value vvvv on condition yyy
 - ldx - 0yyyvvvv - load register yyy with value vvvv
- Note the tradeoff: **more instructions** (can only branch or change outputs on each instruction) but **each instruction is much smaller**.

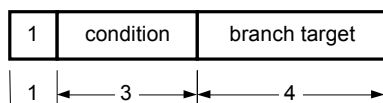
10/24/07

EE 108A Lecture 9 (c) 2006 W. J. Dally and D. Black-Schaffer

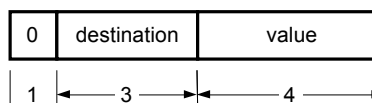
25

Instruction Format Of Microcode With 2 Instruction Types

Branch Instruction



Store Instruction



- Branch Instructions:
 - Operates branch mux as before
 - No write to output registers
- Load Instructions:
 - Sets branch mux to +1
 - Update selected output register:
 - Can include other datapath components
 - e.g., timer in place of an output register

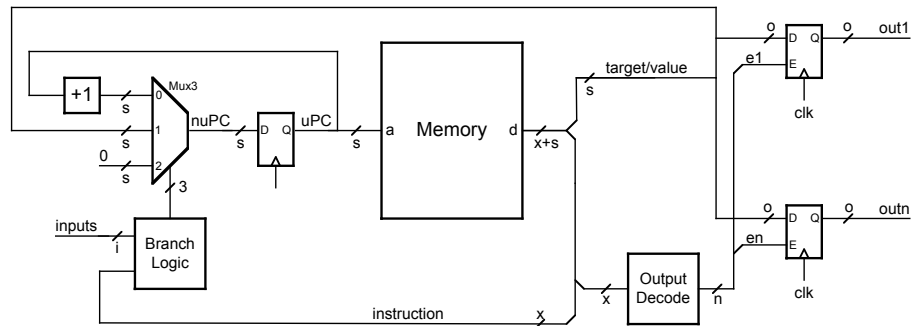
10/24/07

EE 108A Lecture 9 (c) 2006 W. J. Dally and D. Black-Schaffer

26

Block Diagram Of Microcode With Output Instructions

Multiple output registers:
E.g., NS/EW/LT



10/24/07

EE 108A Lecture 9 (c) 2006 W. J. Dally and D. Black-Schaffer

27

Microcode For Traffic-Light Controller With brx & ldx Instructions

Registers to load:

- ldns – load north/south light with value
- ldlt – load left-turn light with value
- ldew – load east/west light with value
- ltim1 – load timer 1 with value – starts timer

10/24/07

EE 108A Lecture 9 (c) 2006 W. J. Dally and D. Black-Schaffer

28

Microcode For Traffic-Light Controller With brx & Idx Instructions (Cont)

	State	Addr	Inst	Value		
NS Green	rst1	00000	ldlt	RED	EW to Red	
	rst2	00001	ldew	RED		
	ns1	00010	ldns	GREEN		
	ns3	00011	ltim	T_GREEN		
	ns4	00100	bntz	ns4		
Until input	ns5	00101	brnle	ns5		Back to NS
NS Yellow to Red	ns6	00110	ldns	YELLOW		
	ns7	00111	ltim	T_YELLOW		
	ns8	01000	bntz	ns8		
	ns9	01001	ldns	RED		
	EW or LT?	ns10	01010	blt	lt1	
Wait for NS Red, make EW Green	ew1	01011	ltim	T_RED	LT to Red	
	ew2	01100	bntz	ew2		
	ew3	01101	ldew	GREEN		
	ew4	01110	ltim	T_GREEN		
	ew5	01111	bntz	ew5		
EW to Red	ew6	10000	ldew	YELLOW		Back to NS
	ew7	10001	ltim	T_YELLOW		
	ew8	10010	bntz	ew8		
	ew9	10011	ldew	RED		
	ew10	10100	br	ns1		
Wait for NS Red, make LT Green	lt1	10101	ltim	T_RED	Back to NS	
	lt2	10110	bntz	lt2		
	lt3	10111	ldlt	GREEN		
	lt4	11000	ltim	T_GREEN		
	lt5	11001	bntz	lt5		
LT to Red	lt6	11010	ldlt	YELLOW		Back to NS
	lt7	11011	ltim	T_YELLOW		
	lt8	11100	bntz	lt8		
	lt9	11101	ldlt	RED		
	lt10	11110	br	ns1		

10/24/07

EE 108A Lecture 9 (c) 2006 W. J. Dally and D. Black-Schaffer

29

Implementing Traffic-Light Controller Microcode With brx & Idx Instructions In Verilog

```
//-----
module ucodeMI(clk,rst,in,out) ;
  parameter n = 2 ; // input width
  parameter m = 9 ; // output width
  parameter o = 3 ; // output sub-width
  parameter k = 5 ; // bits of state
  parameter j = 4 ; // bits of instruction

  input  clk, rst ;
  input  [n-1:0] in ;
  output [m-1:0] out ;

  wire  [k-1:0] nupc, upc ; // microprogram counter
  wire  [j+k-1:0] uinst ; // microinstruction word
  wire done ; // timer done signal

  // split off fields of microinstruction
  wire opcode ; // opcode bit
  wire [j-2:0] inst ; // condition for branch, dest for store
  wire [k-1:0] value ; // target for branch, value for store
  assign {opcode, inst, value} = uinst ;
```

To be continued on next page...

10/24/07

EE 108A Lecture 9 (c) 2006 W. J. Dally and D. Black-Schaffer

30

Implementing Traffic-Light Controller Microcode With brx & Idx Instructions In Verilog (Cont)

```

DFF #(k) upc_reg(clk, nupc, upc) ; // microprogram counter
ROM #(k,k+j) uc(upc, uinst) ; // microcode store

// output registers and timer
DFFE #(o) or0(clk, e[0], value[o-1:0], out[o-1:0]) ; // NS
DFFE #(o) or1(clk, e[1], value[o-1:0], out[2*o-1:o]) ; // EW
DFFE #(o) or2(clk, e[2], value[o-1:0], out[3*o-1:2*o]) ; // LT
Timer #(k) tim(clk, rst, e[3], value, done) ; // timer

// enable for output registers and timer
wire [3:0] e = opcode ? 4'b0 : 1<<inst ;

// branch instruction decode
wire branch = opcode ? (inst[2] ^ ((inst[1:0] == 0) & in[0]) | // BLT
                        ((inst[1:0] == 1) & in[1]) | // BEW
                        ((inst[1:0] == 2) & (in[0]|in[1])) | //BLE
                        ((inst[1:0] == 3) & done)) // BTD
                        : 1'b0 ; // for a store opcode

// microprogram counter
assign nupc = rst ? {k{1'b0}} : branch ? value : upc + 1'b1 ;
endmodule

```

10/24/07

EE 108A Lecture 9 (c) 2006 W. J. Dally and D. Black-Schaffer

31

Summary

- Microcode is just FSM implemented with a ROM or RAM
 - One address for each state x input combination
 - Address contains next state and output
- Adding a *sequencer* reduces size of ROM/RAM
 - One entry per state rather than 2^{inputs}
 - μ PC, incrementer, branch address, and branch control
- Adding instruction types reduces width of ROM/RAM
 - Branch or output in each instruction – rather than both
 - Type field specifies which one
 - Need more instructions, but each is smaller
- One step away from a full processor
 - Just add more instructions (defer until EE108B)

10/24/07

EE 108A Lecture 9 (c) 2006 W. J. Dally and D. Black-Schaffer

32