

Welcome to CS122A: Embedded System Design

Harry Hsieh
Department of Computer Science and Engineering
University of California at Riverside

Administrative Stuff

- Homework1 due Thursday 10/3 at beginning of class
 - No late homework accepted
- Presentation
 - Sign up for article and time slot by NOW
 - 5% of the grade
 - Highly recommended to:
 - Send your presentation material to me no later than 8AM that day ...
 - ...powerpoint files, ps, pdf...
- Make sure you have signed up for Mailing list
 - cs122a@lists.csucr.edu

CS122A: Embedded System Design, Fall02

Undergraduate research opportunity

- Credits available
- INdependent project on compiling (gcc) embedded system benchmarks (EMBCC) to reconfigurable platform (RT-RISC)
- Requirement
 - Independence, self - motivation
 - Relevant courses
 - Compilers (cs152)
 - Algorithms (cs141)
 - Architecture (cs152)
- Send me resume/transcript this week...

CS122A: Embedded System Design, Fall02

The Theory of Evolution

Charles Darwin,
The Original of Species
by Means of Natural Selection
1859

Year (millions)	Cranial Size (cc)
-4	~380
-3	~400
-2	~500
-1	~800
0	~1400

- 4 million BC, "Lucy" has cranial size of 380 cc
- Today, *Homo Sapiens* average 1400 cc
 - Double every 2.08 million years
- Assume no change in brain cell structure
 - Human productivity falls behind!!!

CS122A: Embedded System Design, Fall02

Design productivity exponential increase

Productivity (K) Trans./Staff.- Mo.

- Number of transistor a designer "produce"
 - Exponential increase over the past few decades
 - Improving design technology
 - I.e. brain cells getting better...

CS122A: Embedded System Design, Fall02

The Productivity Gap

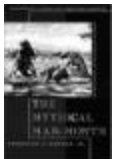
Logic transistors per chip (in millions) vs Productivity (K) Trans./Staff.- Mo.

- Even with advances in design technology
 - Human still fall behind chips!!!
 - More and more designers needed...

CS122A: Embedded System Design, Fall02

The mythical man-month

- The situation is even worse than the productivity gap indicates
- In theory,
 - adding designers to team reduces project completion time
- In reality,
 - Productivity/designer decreases due to complex management & communication
- Known as "the mythical man-month" (Brooks 1975)
- At some point, can actually lengthen project completion time!



CS122A: Embedded System Design, Fall 02 7

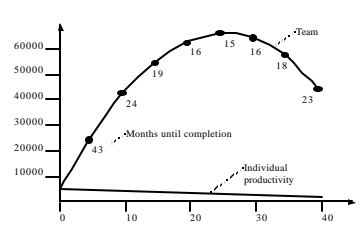
The mythical man-month vs 486

- 1 M. Transistors; 1 designer = 5000 trans/mo; Add. Designer = -100 trans/mo


# of designers	Individual Productivity	Team Productivity
1	5000 trans/mo	5000 trans/mo
2	4900	9800
3	4800	14400
4	4700	18800
5	4600	23000
10	4000	40000
20	3000	60000
30	2000	60000
40	1000	40000

CS122A: Embedded System Design, Fall 02 8

Too many cooks...

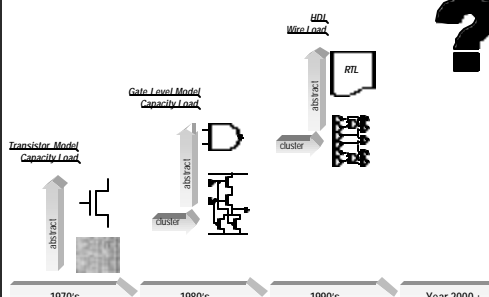


So, How to keep increasing productivity?



CS122A: Embedded System Design, Fall 02 9

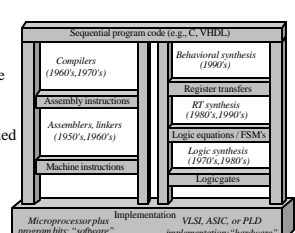
Next Level of Abstraction?



CS122A: Embedded System Design, Fall 02 10

The co-design ladder

- In the past:
 - Hardware and software design technologies were very different
 - Recent maturation of synthesis enables a unified view of hardware and software
- Goal of cs122a



The choice of hardware versus software for a particular function is simply a tradeoff among various design metrics, like performance, power, size, NRE cost, and especially flexibility; there is no fundamental difference between what hardware or software can implement.

CS122A: Embedded System Design, Fall 02 11

Custom Single Purpose Processors

Hardware

CS122A: Embedded System Design, Fall 02 12

What will we cover

- Combinational logic
- Sequential logic
- Custom single-purpose processor design
- Optimizing Custom Single-Purpose Processors

13
CS122A: Embedded System Design, Fall02

Combinational logic

- Output is a function of present input only
 - No memory of past inputs
- Simple circuit designs built with basic logic gates
 - using truth tables, Karnaugh maps
- Larger circuit designs built with RT-level combinational components
 - multiplexor, decoder, adder, comparator, ALU, etc.

14
CS122A: Embedded System Design, Fall02

Combinational logic design - simple circuits

A) Problem description

y is 1 if a is equal to 1, or b and c is equal to 1. z is 1 if b or c is equal to 1, but not both, unless a is 1 also.

B) Truth table

Inputs			Outputs	
a	b	c	y	z
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	1	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

C) Output equations

y = a'bc + ab'c' + ab'c + abc' + abc
z = a'b'c + a'bc' + ab'c + abc' + abc

Possibly 28 gates!!!

15
CS122A: Embedded System Design, Fall02

Combinational logic design - simple circuits

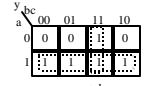
C) Output equations

y = a'bc + ab'c' + ab'c + abc' + abc
z = a'b'c + a'bc' + ab'c + abc' + abc

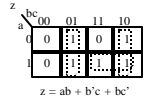
Possibly 28 gates!!!

D) Minimized output equations

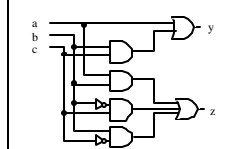
y = a + bc



z = ab + b'c + bc'



E) Logic Gates



16
CS122A: Embedded System Design, Fall02

What will we cover

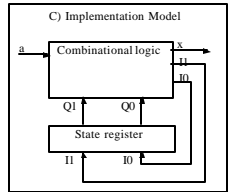
- Combinational logic
- Sequential logic
- Custom single-purpose processor design
- Optimizing Custom Single-Purpose Processors

17
CS122A: Embedded System Design, Fall02

Sequential Logic Design

- Given this implementation model
 - Sequential logic design reduces to combinational logic design
- Separate design and verification of
 - Timing element
 - Latch, Flip-flop...
 - Combinational logic
- Finite State Machines
 - Mealy
 - X depends on "a" and "Q"
 - Moore
 - X depends on "Q" only

C) Implementation Model



18
CS122A: Embedded System Design, Fall02

Sequential logic design

A) Problem Description

You want to construct a clock divider. Slow down your pre-existing clock so that you output a 1 for every four clock cycles

C) Implementation Model

B) State Diagram

Transition at clock boundary

19

Sequential logic design

B) State Diagram

D) State Table (Moore-type)

Inputs			Outputs		
Q1	Q0	a	I1	I0	x
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	1	0	0
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	0	0	0

C) Implementation Model

20

Sequential logic design (cont.)

D) State Table (Moore-type)

Inputs			Outputs		
Q1	Q0	a	I1	I0	x
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	1	0	0
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	0	0	0

E) Minimized Output Equations

$I1 = Q1'Q0a + Q1a' + Q1Q0'$

$I0 = Q0a' + Q0'a$

$x = Q1Q0$

21

Sequential logic design (cont.)

E) Minimized Output Equations

$I1 = Q1'Q0a + Q1a' + Q1Q0'$

$I0 = Q0a' + Q0'a$

$x = Q1Q0$

F) Combinational Logic

22

What will we cover

- Transistors and Gates
- Combinational logic
- Sequential logic
- Custom single-purpose processor design
- RT-level custom single-purpose processor design
- Optimizing Custom Single-Purpose Processors

23

Custom single-purpose processor basic model

controller and datapath

a view inside the controller and datapath

24

Custom design with FSM

- Start with a C-Like description of GCD
- We plan to:
 - Transform it into an FSM through template
 - Construct Data path with sequential component
 - Step 1: Create registers
 - Step 2: Create functional units
 - Step 3: Create wiring and mux/demux
 - Step 4: Create control input and datapath output
 - Construct control logics as an FSM

25

Example: greatest common divisor

- First create algorithm

(a) black-box view

(b) desired functionality

```

0: int x, y;
1: while (!) {
2:   while (!go_i);
3:   x = x_i;
4:   y = y_i;
5:   while (x != y) {
6:     if (x < y)
7:       y = y - x;
8:     else
9:       x = x - y;
9:   d_o = x;
   }
```

26

Example: greatest common divisor

- Convert to Moore FSM
 - Finite-state machine with datapath
 - Statements scheduled into states
 - use templates to convert

(b) desired functionality

```

0: int x, y;
1: while (!) {
2:   while (!go_i);
3:   x = x_i;
4:   y = y_i;
5:   while (x != y) {
6:     if (x < y)
7:       y = y - x;
8:     else
9:       x = x - y;
9:   d_o = x;
   }
```

(c) state diagram

27

State diagram templates

Assignment statement

```

a = b
next statement
```

28

State diagram templates

Loop statement

```

while (cond) {
  loop-body-
  statements
}
next statement
```

29

State diagram templates

Branch statement

```

if (c1)
  c1 stmts
else if c2
  c2 stmts
else
  other stmts
next statement
```

30

Example: greatest common divisor

(c) state diagram

(b) desired functionality

```

0: int x, y;
1: while (1) {
2:   while (!go_i);
3:   x = x_i;
4:   y = y_i;
5:   while (x != y) {
6:     if (x < y)
7:       y = y - x;
8:     else
9:       x = x - y;
9:   }
9:   d_o = x;
}
    
```

CS122A: Embedded System Design, Fall 02

Where were we?

- Start with a C-Like description of GCD
- We plan to:
 - Transform it into an FSM through template
 - Construct Data path with sequential component
 - Step 1: Create registers
 - Step 2: Create functional units
 - Step 3: Create wiring and mux/demux
 - Step 4: Create control input and datapath output
 - Construct control logics as an FSM

CS122A: Embedded System Design, Fall 02

Creating the datapath

- Create a register for any declared variable

(b) desired functionality

```

0: int x, y;
1: while (1) {
2:   while (!go_i);
3:   x = x_i;
4:   y = y_i;
5:   while (x != y) {
6:     if (x < y)
7:       y = y - x;
8:     else
9:       x = x - y;
9:   }
9:   d_o = x;
}
    
```

CS122A: Embedded System Design, Fall 02

Creating the datapath

- Create a functional unit for each arithmetic operation

(b) desired functionality

```

0: int x, y;
1: while (1) {
2:   while (!go_i);
3:   x = x_i;
4:   y = y_i;
5:   while (x != y) {
6:     if (x < y)
7:       y = y - x;
8:     else
9:       x = x - y;
9:   }
9:   d_o = x;
}
    
```

CS122A: Embedded System Design, Fall 02

Creating the datapath

- Connect ports, registers, and functional unit

(b) desired functionality

```

0: int x, y;
1: while (1) {
2:   while (!go_i);
3:   x = x_i;
4:   y = y_i;
5:   while (x != y) {
6:     if (x < y)
7:       y = y - x;
8:     else
9:       x = x - y;
9:   }
9:   d_o = x;
}
    
```

CS122A: Embedded System Design, Fall 02

Creating the datapath

- Create control input and datapath output

< Clock connection

(b) desired functionality

```

0: int x, y;
1: while (1) {
2:   while (!go_i);
3:   x = x_i;
4:   y = y_i;
5:   while (x != y) {
6:     if (x < y)
7:       y = y - x;
8:     else
9:       x = x - y;
9:   }
9:   d_o = x;
}
    
```

CS122A: Embedded System Design, Fall 02

Creating the controller's FSM

- Two steps to convert FSMD to controller FSM
 - 1. Replace write by select and load
 - 2. Replace logical operation by datapath output

37

Creating the controller's FSM

1. Replace write by select and load

38

Creating the controller's FSM

2. Replace the logical ops by DP output

39

Creating the controller's FSM

Binary state encoding

40

State table for the FSM

Inputs				Outputs			
Q3	Q2	Q1	Q0	x_sel	y_sel	x_ld	y_ld
0	0	0	0	*	*	*	*
0	0	0	1	*	*	*	*
0	0	1	0	*	*	*	*
0	0	1	1	*	*	*	*
0	1	0	0	*	*	*	*
0	1	0	1	*	*	*	*
0	1	1	0	*	*	*	*
0	1	1	1	*	*	*	*
1	0	0	0	*	*	*	*
1	0	0	1	*	*	*	*
1	0	1	0	*	*	*	*
1	0	1	1	*	*	*	*
1	1	0	0	*	*	*	*
1	1	0	1	*	*	*	*
1	1	1	0	*	*	*	*
1	1	1	1	*	*	*	*

- * All possible combination of 0 & 1
- X Don't care

41

State table for the FSM

Inputs				Outputs			
Q3	Q2	Q1	Q0	x_sel	y_sel	x_ld	y_ld
0	0	0	0	*	*	*	*
0	0	0	1	*	*	*	*
0	0	1	0	*	*	*	*
0	0	1	1	*	*	*	*
0	1	0	0	*	*	*	*
0	1	0	1	*	*	*	*
0	1	1	0	*	*	*	*
0	1	1	1	*	*	*	*
1	0	0	0	*	*	*	*
1	0	0	1	*	*	*	*
1	0	1	0	*	*	*	*
1	0	1	1	*	*	*	*
1	1	0	0	*	*	*	*
1	1	0	1	*	*	*	*
1	1	1	0	*	*	*	*
1	1	1	1	*	*	*	*

- * All possible combination of 0 & 1
- X Don't care

42

State table for the FSM

Inputs				Outputs						
Q3	Q2	Q1	Q0	x_sel	x_ld	y_sel	y_ld	D	LD	
0	0	0	0	*	*	*	*	*	*	
0	0	0	1	*	*	*	*	*	*	
0	0	1	0	*	*	*	*	*	*	
0	0	1	1	*	*	*	*	*	*	

- * - All possible combination of 0 & 1
- X - Don't care

43

Controller state table for the GCD example

Inputs				Outputs						
Q3	Q2	Q1	Q0	x_sel	x_ld	y_sel	y_ld	D	LD	
0	0	0	0	*	*	*	*	*	*	
0	0	0	1	*	*	*	*	*	*	
0	0	1	0	*	*	*	*	*	*	
0	0	1	1	*	*	*	*	*	*	
0	1	0	0	*	*	*	*	*	*	
0	1	0	1	*	*	*	*	*	*	
0	1	1	0	*	*	*	*	*	*	
0	1	1	1	*	*	*	*	*	*	
1	0	0	0	*	*	*	*	*	*	
1	0	0	1	*	*	*	*	*	*	
1	0	1	0	*	*	*	*	*	*	
1	0	1	1	*	*	*	*	*	*	
1	1	0	0	*	*	*	*	*	*	
1	1	0	1	*	*	*	*	*	*	
1	1	1	0	*	*	*	*	*	*	
1	1	1	1	*	*	*	*	*	*	

- * - All possible combination of 0 & 1
- X - Don't care

44

What have we done?

- Start with a C-Like description of GCD
 - Transform it into an FSMD through template
 - Construct Data path with sequential component
 - Construct control logics as an FSM

(a) Controller

(b) Datapath

45

Completing the GCD custom single-purpose processor design

- We finished the datapath
- We have a state table for the next state and control logic
 - All that's left is combinational logic design
- This is *not* an optimized design, but we see the basic steps

a view inside the controller and datapath

46

A few quirk about FSMD

- Variables are not updated until state exit

foo or bar?

47

What will we cover

- Combinational logic
- Sequential logic
- Custom single-purpose processor design
- Optimizing Custom Single-Purpose Processors

48

Optimizing single-purpose processors

- Optimization is the task of making design metric values the best possible
- Optimization opportunities
 - original program
 - FSMD
 - datapath
 - FSM

49

CS122A: Embedded System Design, Fall02

Optimizing the original program

<p>original program</p> <pre> 0: int x, y; 1: while (1) { 2: while (!go_i); 3: x = x_i; 4: y = y_i; 5: while (x != y) { 6: if (x < y) { 7: y = y - x; 8: } else { 9: x = x - y; 10: } 11: } 12: d_o = x; 13: }</pre> <p>$O(n)$</p>	<p>replace the subtraction operation(s) with modulus operation in order to speed up program</p>	<p>optimized program</p> <pre> 0: int x, y, r; 1: while (1) { 2: while (!go_i); 3: // x must be the larger number 4: if (x_i >= y_i) { 5: x = x_i; 6: y = y_i; 7: } else { 8: x = y_i; 9: y = x_i; 10: } 11: while (y != 0) { 12: r = x % y; 13: x = y; 14: y = r; 15: } 16: d_o = x; 17: }</pre> <p>$O(\log n)$</p>
--	---	--

50

CS122A: Embedded System Design, Fall02

Optimizing the FSMD

- Areas of possible improvements
 - merge states
 - States with constants on transitions can be eliminated, transition taken is already known
 - States with independent operations can be merged
 - separate states
 - states which require complex operations ($a*b*c*d$) can be broken into smaller states to reduce hardware size

51

CS122A: Embedded System Design, Fall02

Optimizing the FSMD (cont')

eliminate state 1 - transitions have constant values

52

CS122A: Embedded System Design, Fall02

Optimizing the FSMD (cont')

eliminate state 1 - transitions have constant values

merge state 2 and state 2J - no loop operation in between them

53

CS122A: Embedded System Design, Fall02

Optimizing the FSMD (cont')

eliminate state 1 - transitions have constant values

merge state 2 and state 2J - no loop operation in between them

merge state 3 and state 4 - assignment operations are independent of one another

54

CS122A: Embedded System Design, Fall02

Optimizing the FSM (cont')

eliminate state 1 – transitions have constant values

merge state 2 and state 2f – no loop operation in between them

merge state 3 and state 4 – assignment operations are independent of one another

merge state 5 and state 6 – transitions from state 6 can be done in state 5

55

CS122A: Embedded System Design, Fall02

Optimizing the FSM (summary)

eliminate state 1 – transitions have constant values

merge state 2 and state 2f – no loop operation in between them

merge state 3 and state 4 – assignment operations are independent of one another

merge state 5 and state 6 – transitions from state 6 can be done in state 5

eliminate state 5f and 6f – transitions from each state can be done from state 7 and state 8, respectively

eliminate state 1 – J – transition from state 1 – J can be done directly from state 9

Cycle by cycle behavior changed!!!

56

CS122A: Embedded System Design, Fall02

Optimizing the datapath

- Sharing of functional units
 - one-to-one mapping, as done previously, is not necessary
 - if same operation occurs in different states, they can share a single functional unit
- Multi-functional units
 - ALU's support a variety of operations, it can be shared among operations occurring in different states

57

CS122A: Embedded System Design, Fall02

Optimizing the FSM

- State encoding
 - task of assigning a unique bit pattern to each state in an FSM
 - size of state register and combinational logic vary for different encodings
 - Given 4 states, 2 bit binary encoding
 - There are 24 possible different encoding!!!
 - Extremely difficult to choose the BEST encoding
- State minimization
 - task of merging equivalent states into a single state
 - two states equivalent if, for all possible input combinations, the states generate the same outputs and transition to the next same state

Not the same as state-merging

58

CS122A: Embedded System Design, Fall02

Optimizing the FSM

- State encoding
 - task of assigning a unique bit pattern to each state in an FSM
 - size of state register and combinational logic vary for different encodings
 - Given 4 states, 2 bit binary encoding
 - There are 24 possible different encoding!!!
 - Extremely difficult to choose the BEST encoding
- State minimization
 - task of merging equivalent states into a single state
 - two states equivalent if, for all possible input combinations, the states generate the same outputs and transition to the next same state

Not the same as state-merging

59

CS122A: Embedded System Design, Fall02

Administrative Stuff

- Homework1 due Thursday 10/3 at beginning of class
 - No late homework accepted
- Presentation
 - Sign up for article and time slot by NOW
 - 5% of the grade
 - Highly recommended to:
 - Send your presentation material to me no later than 8AM that day ...
 - ...powerpoint files, ps, pdf...
- Make sure you have signed up for Mailing list
 - cs122a@lists.cs.ucr.edu

60

CS122A: Embedded System Design, Fall02