

Optimizing single-purpose processors

- Optimization is the task of making design metric values the best possible
 - Remember “design metrics?” speed, power, size...
- Optimization opportunities
 - Original program
 - FSM
 - Datapath

Optimizing the original program

- Analyze program attributes and look for areas of possible improvement
 - number of computations
 - size of variable
 - time and space complexity
 - operations used
 - multiplication and division very expensive

Optimizing the original program (cont')

```
original program
0: int x, y;
1: while (1) {
2:   while (tgo_1);
3:   x = x_1;
4:   y = y_1;
5:   while (x != y) {
6:     if (x < y)
7:       y = y - x;
8:     else
9:       x = x - y;
10:  }
11:  d_o = x;
12: }
```

replace the subtraction operation(s) with modulo operation in order to speed up program

```
optimized program
0: int x, y, r;
1: while (1) {
2:   while (tgo_1);
3:   // x must be the larger number
4:   if (x_1 >= y_1) {
5:     x=x_1;
6:   }
7:   else {
8:     y=y_1;
9:   }
10:  while (y != 0) {
11:    r = x % y;
12:    x = y;
13:    y = r;
14:  }
15:  d_o = x;
16: }
```

GCD(42, 8) - 9 iterations to complete the loop
x and y values evaluated as follows : (42, 8), (43, 8), (26, 8), (18, 8), (10, 8), (2, 8), (2, 6), (2, 4), (2, 2).

GCD(42, 8) - 3 iterations to complete the loop
x and y values evaluated as follows: (42, 8), (8, 2), (2, 0)

Optimizing the original program (cont')

```
original program
0: int x, y;
1: while (1) {
2:   while (tgo_1);
3:   x = x_1;
4:   y = y_1;
5:   while (x != y) {
6:     if (x < y)
7:       y = y - x;
8:     else
9:       x = x - y;
10:  }
11:  d_o = x;
12: }
```

```
optimized program
0: int x, y, r;
1: while (1) {
2:   while (tgo_1);
3:   // x must be the larger number
4:   if (x_1 >= y_1) {
5:     x=x_1;
6:   }
7:   else {
8:     y=y_1;
9:   }
10:  while (y != 0) {
11:    r = x % y;
12:    x = y;
13:    y = r;
14:  }
15:  d_o = x;
16: }
```

What did we optimize for? Speed. This will now run faster, but we will pay the price with a larger silicon implementation (it will be more expensive).

Optimizing the original program (cont')

Optimizing the original program (the algorithm) is the one optimization that is very difficult, if not impossible, for a computer to do for you.

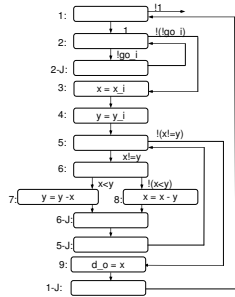
There are no steps, or instructions to optimizing an algorithm, it takes creativity and ingenuity.

Optimizing the FSM

- Areas of possible improvements
 - merge states (to optimize speed)
 - states with constants on transitions can be eliminated, transition taken is already known
 - states with independent operations can be merged
 - separate states (to optimize area)
 - states which require complex operations ($a*b*c*d$) can be broken into smaller states to reduce hardware size
 - scheduling (to optimize speed)
 - beyond the scope of the processors we have made up to this point, but has to do with loading or calculating things early, knowing that they will be used in the future

Optimizations...

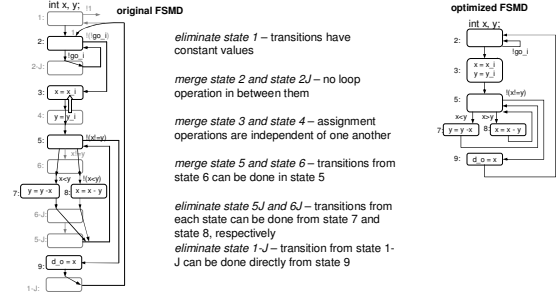
See any room for optimizations here?



Embedded Systems Design: A Unified Hardware/Software Introduction, (c) 2000 Vahid/Givargis

7

Optimizing the FSMD (cont.)



Embedded Systems Design: A Unified Hardware/Software Introduction, (c) 2000 Vahid/Givargis

8

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
 - ALUs support a variety of operations, it can be shared among operations occurring in different states

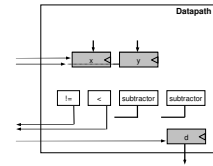
Embedded Systems Design: A Unified Hardware/Software Introduction, (c) 2000 Vahid/Givargis

9

Creating the datapath

We could change this...

```
0: int x, y;
1: while (!go_i) {
2:   x = x - 1;
3:   y = y - 1;
4:   while (x != y) {
5:     if (x < y) {
6:       y = y - x;
7:     } else {
8:       x = x - y;
9:     }
10:   }
11:   d_o = x;
12: }
```



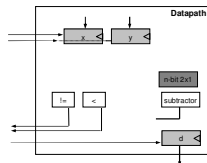
Embedded Systems Design: A Unified Hardware/Software Introduction, (c) 2000 Vahid/Givargis

10

Creating the datapath

to this...

- But, it wouldn't save us a whole lot in this case because we are only trading a subtractor for a multiplexor. It is a lot more savings if you are eliminating a multiplier or something like that.

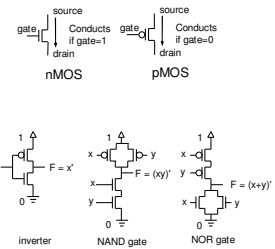


Embedded Systems Design: A Unified Hardware/Software Introduction, (c) 2000 Vahid/Givargis

11

Review of Chapter 2...

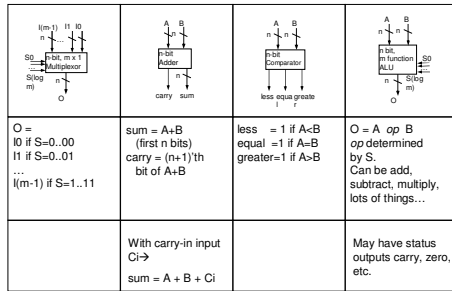
- Complementary Metal Oxide Semiconductor
- We refer to logic levels
 - Typically 0 is 0V, 1 is 5V
- Two basic CMOS types
 - nMOS conducts if gate=1
 - pMOS conducts if gate=0
 - Hence "complementary"
- Basic gates
 - Inverter, NAND, NOR



Embedded Systems Design: A Unified Hardware/Software Introduction, (c) 2000 Vahid/Givargis

12

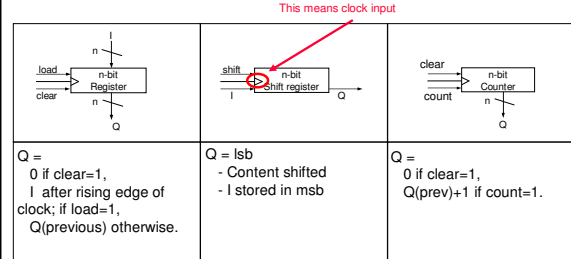
Common Combinational Components



Embedded Systems Design: A Unified
Hardware/Software Introduction, (c) 2000
Vahid/Givargis

13

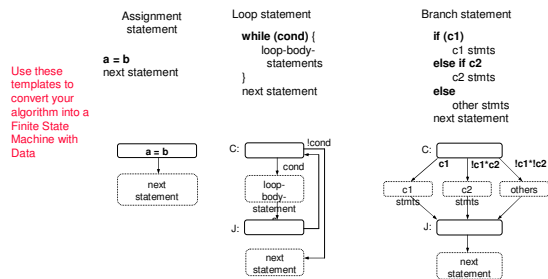
Sequential components



Embedded Systems Design: A Unified
Hardware/Software Introduction, (c) 2000
Vahid/Givargis

14

State diagram templates



Embedded Systems Design: A Unified
Hardware/Software Introduction, (c) 2000
Vahid/Givargis

15

Creating the datapath

1. Create a register for any variable
2. Create a functional unit for each arithmetic operation
3. Connect the ports, registers and functional units
 1. Based on reads and writes
 2. Use multiplexors for multiple sources
4. Create unique identifier
 - for each datapath component control input and output

And that's all of chapter 2!

Embedded Systems Design: A Unified
Hardware/Software Introduction, (c) 2000
Vahid/Givargis

16

To Do before next Thursday...

- Prepare for quiz
 - Know how to create an FSM + datapath from an algorithm
 - Know how to evaluate N-MOS and P-MOS circuits
- Check your e-mail for Homework 1

Embedded Systems Design: A Unified
Hardware/Software Introduction, (c) 2000
Vahid/Givargis

17