Determining Clock Frequency

- Designers of digital circuits often want fastest performance
  - Means want high clock frequency
- Frequency limited by longest register-to-register delay
  - Known as critical path
  - If clock is any faster, incorrect data may be stored into register
  - Longest path on right is 2 ns
  - Ignoring wire delays, and register setup and hold times, for simplicity

Critical Path

- Example shows four paths
  - a to c through +: 2 ns
  - a to d through + and *: 7 ns
  - b to d through + and *: 7 ns
  - b to d through *: 5 ns
- Longest path is thus 7 ns
- Fastest frequency
  - \( \frac{1}{7 \text{ ns}} = 142 \text{ MHz} \)

Critical Path Considering Wire Delays

- Real wires have delay too
  - Must include in critical path
- Example shows two paths
  - Each is 0.5 + 2 + 0.5 = 3 ns
- Trend
  - 1980s/1990s: Wire delays were tiny compared to logic delays
  - But wire delays not shrinking as fast as logic delays
  - Wire delays may even be greater than logic delays!
- Must also consider register setup and hold times, also add to path
- Then add some time to the computed path, just to be safe
  - e.g., if path is 3 ns, say 4 ns instead

A Circuit May Have Numerous Paths

- Paths can exist
  - In the datapath
  - In the controller
  - Between the controller and datapath
  - May be hundreds or thousands of paths
- Timing analysis tools that evaluate all possible paths automatically very helpful
Behavioral Level Design: C to Gates

- Earlier sum-of-absolute-differences example
  - Started with high-level state machine
  - C code is an even better starting point — easier to understand

C code

```c
// not quite C syntax
int SAD(byte A[256], byte B[256]) // a
{
  uint sum; int i;
  sum = 0;
  i = 0;
  while (i < 256) {
    sum = sum + abs(A[i] - B[i]);
    i = i + 1;
  }
  return sum;
}
```

Digital Design Copyright © 2006 Frank Vahid

Behavioral-Level Design: Start with C (or Similar Language)

- Replace first step of RTL design method by two steps
  - Capture in C, then convert C to high-level state machine
  - How convert from C to high-level state machine?

Step 1A: Capture in C
Step 1B: Convert to high-level state machine

Converting from C to High-Level State Machine

- Convert each C construct to equivalent states and transitions
  - Assignment statement
    - Becomes one state with assignment
  - If-then statement
    - Becomes state with condition check, transitioning to "then" statements if condition true, otherwise to ending state
    - "then" statements would also be converted to states
  - If-then-else
    - Becomes state with condition check, transitioning to "then" statements if condition true, or to "else" statements if condition false
  - While loop statement
    - Becomes state with condition check, transitioning to while loop's statements if true, then transitioning back to condition check

Example: Computing the maximum of two numbers

Inputs: uint X, Y
Outputs: uint Max

```c
if (X > Y) {
  Max = X;
} else {
  Max = Y;
}
```

Digital Design Copyright © 2006 Frank Vahid

Example: Converting Sum-of-Absolute-Differences C code to High-Level State Machine

- Convert each construct to states
  - Simplify when possible, e.g., merge states
- From high-level state machine, follow RTL design method to create circuit
- Thus, can convert C to gates using straightforward automatable process
- Not all C constructs can be efficiently converted
- Use C subset if intended for circuit
- Can use languages other than C, of course
Design Challenge

- Convert the following C code, which calculates the number of times the value b is not found within an array A consisting of 256 8-bit values, into a high-level state machine.

```
Inputs: byte a[256], byte b, bit go
Outputs: byte freq, bit done
NFREQ:
  while(1) {
    while(!go);
    done = 0;
    i = 0;
    tfreq = 0;
    while( i < 256 ) {
      if( a[i] != b ) {
        tfreq = tfreq + 1;
      }
    }
    freq = tfreq;
    done = 1;
  }

Due Next Lecture (as announced in class)
1 point extra credit (Homework)