CSE140: Components and Design Techniques for Digital Systems

Register Transfer Level (RTL) Design

Tajana Simunic Rosing
Where we are now...

- What we have covered last time:
  - Register Transfer Level (RTL) design
- What we are covering today:
  - Continuing RTL design, critical path analysis in RTL, c-code to gates
- Deadlines:
  - ZyBook chap 6 due
  - Vahid textbook chap 5 available on class schedule page
  - Quiz 6 – the last quiz of the quarter!!!!
  - HW#6 due next Tuesday!
  - Exam#3 last day of class
    - 80 minutes long, comprehensive
    - Bring one 8 ½ x 11” paper with handwritten notes, but nothing else

Sources: TSR, Katz, Boriello, Vahid, Perkowski
## RTL Design Process

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
</table>
| **Step 1: Capture behavior** | **Capture a high-level state machine**  
Describe the system’s desired behavior as a high-level state machine. The state machine consists of states and transitions. The state machine is “high-level” because the transition conditions and the state actions are more than just Boolean operations on single-bit inputs and outputs. |
| **2A**                  | **Create a datapath**  
Create a datapath to carry out the data operations of the high-level state machine. |
| **Step 2: Convert to circuit** | **Connect the datapath to a controller**  
Connect the datapath to a controller block. Connect external control inputs and outputs to the controller block. |
| **2B**                  | **Derive the controller’s FSM**  
Convert the high-level state machine to a finite-state machine (FSM) for the controller, by replacing data operations with setting and reading of control signals to and from the datapath. |

Sources: TSR, Katz, Boriello, Vahid, Perkowski
Remember the Soda Dispenser Design?

Inputs: c (bit), a (8 bits), s (8 bits)
Outputs: d (bit)
Local registers: tot (8 bits)
RTL Delay: A Circuit May Have Numerous Paths

- Frequency is limited by **longest register-to-register delay**
  - **critical path**
- Paths can exist
  - In the datapath
  - In the controller
  - Between the controller and datapath
  - May be hundreds or thousands of paths

\[ t_{ccq} + t_{cd} > t_{hold} + t_{skew} \]

\[ T_c \geq t_{pcq} + t_{pd} + t_{setup} + t_{skew} \]
**RTL Design Process: Datapath components**

- **Adder (unsigned)**
  - $S = A + B$
  - $S = A + B$
  - $Q = 0$ if $clk^\wedge$ and $clr = 1$
  - $Q = I$ if $clk^\wedge$ and $ld = 1$
  - Else $Q$ stays the same

- **Comparator**
  - (unsigned)
  - $A < B$: $lt = 1$
  - $A = B$: $eq = 1$
  - $A > B$: $gt = 1$

- **Shift Register**
  - $s0 = 0$: $Q = I0$
  - $s0 = 1$: $Q = I1$

- **Subtractor (signed)**
  - $S = A - B$
  - $P = A * B$
  - $Q = |A|$

- **Multiplier (unsigned)**
  - $P = A * B$
  - $Q = |A|$

- **Absolute Value**
  - $Q = |A|$

- **Register File**
  - $W_d$
  - $W_e$
  - $RF$
  - $W_a$
  - $R_a$

  - $clk^\wedge$ and $W_e = 1$: $RF[W_a] = W_d$
  - $R_e = 1$: $R_d = RF[R_a]$

**Sources:** TSR, Katz, Boriello, Vahid, Perkowski
More Datapath Examples

(a) $\text{Preg} = X + Y + Z$

(b) $\text{Preg} = \text{Preg} + X$

(c) $\text{Preg} = X + Y; \quad \text{regQ} = Y + Z$

(d) $k=0: \text{Preg} = Y + Z$
   $k=1: \text{Preg} = X + Y$

Sources: TSR, Katz, Boriello, Vahid, Perkowski
HLSM Array Example

(a) ArrayEx

Inputs: (none)

Outputs: P (11 bits)

Local storage: A[4](11 bits)

Preg (11 bits)

Init1

Preg := 0
A[0] := 9

(A[0] == 8)'

Init2


A[0] == 8

Out1


(b) ArrayEx

Inputs: A_eq_8

Outputs: A_s, A_Wa0, ...

Preg_clr = 1
A_s = 0
A_Wa1=0, A_Wa1=0
A_We = 1

(A_eq_8)'

A_s = 1
A_Wa1=0, A_Wa0=1
A_We = 1
A_Ra1=0, A_Ra0=0
A_Re = 1

Out1

Preg_ld = 1

Controller

31

Sources: TSR, Katz, Boriello, Vahid, Perkowski
RTL Example: Video Compression

Video is a series of frames (e.g., 30 per second)
Most frames similar to previous frame
  Compression idea: just send difference from previous frame

(a) Digitized frame 1: 1 Mbyte
   Digitized frame 2: 1 Mbyte

(b) Difference of 2 from 1: 0.01 Mbyte
   Just send difference

Sources: TSR, Katz, Boriello, Vahid, Perkowski
Video Compression – Sum of Absolute Differences

If two frames are similar just send a difference instead

- Compare corresponding 16x16 “blocks”
  - Treat 16x16 block as 256-byte array
- Compute the absolute value of the difference of each array item
- Sum the differences
  - If above a threshold, send a complete frame for second frame
  - Else send the difference

Each is a pixel, assume represented as 1 byte (actually, a color picture might have 3 bytes per pixel, for intensity of red, green, and blue components of pixel)
Sum-of-Absolute Differences: High-level FSM

Inputs: A, B (256 byte memory); go (bit)
Outputs: sad (32 bits)
Local registers: sum, sad_reg (32 bits); i (9 bits)

- **S0**: wait for *go*
- **S1**: initialize *sum* and *index*
- **S2**: check if done (*i* >= 256)
- **S3**: add difference to *sum*, increment index
- **S4**: done, write to output *sad_reg*

Sources: TSR, Katz, Boriello, Vahid, Perkowski
Sum-of-Absolute Differences: Datapath

Inputs: A, B (256 byte memory); go (bit)
Outputs: sad (32 bits)
Local registers: sum, sad_reg (32 bits); i (9 bits)

Sources: TSR, Katz, Boriello, Vahid, Perkowski
Sum-of-Absolute Differences: Connect datapath and controller & specify the controller FSM

```
S0
  go
  go

S1
  sum=0 sum_clr=1
  i=0 i_clr=1

S2
  i<256 i_lt_256
  sum=sum+abs(A[i]+B[i])
  sum LD=1; AB_rd=1
  i=i+1 i_inc=1

S3

S4
  sad_reg=sum
  sad_reg LD=1

Controller

AB_addr  A_data  B_data
<256
9       8 8

i  i inc
\> i clr
sum LD
sum CLR
sad_reg LD

abs

+ 8

\>

32 32 32

\>

sad

Sources: TSR, Katz, Boriello, Vahid, Perkowski
```
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

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

Inputs: A, B [256](8 bits); go (bit)
Outputs: sad (32 bits)
Local storage: sum, sadreg (32 bits); i (9 bits)
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

```
target = expression;

if (cond) {
    // then stmts
}
```

Notes:
- Sources: TSR, Katz, Boriello, Vahid, Perkowski
Converting from C to High-Level State Machine

**If-then-else**
- Becomes state with condition check, transitioning to “then” statements if condition true, or to “else” statements if condition false

```c
if (cond) {
    // then stmts
} else {
    // else stmts
}
```

**While loop statement**
- Becomes state with condition check, transitioning to while loop’s statements if true, then transitioning back to condition check

```c
while (cond) {
    // while stmts
}
```
Converting from C to HLSM: Example

- Simple example: computing the maximum of two numbers
  - Convert if-then-else statement to states (b)
  - Then convert assignment statements to states (c)

Inputs: uint X, Y
Outputs: uint Max

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

Sources: TSR, Katz, Boriello, Vahid, Perkowski
Example: SAD C code to HLSM

- Convert each construct to states
  - Simplify states
- Use RTL design process to convert to circuit
- However, only a subset of C can be easily converted
  - Can use language other than C

Inputs: byte A[256], B[256]
Output: int sad

main()
{
  uint sum; short uint i;
  while (1) {
    while (!go);
    sum = 0;
    i = 0;
    while (i < 256) {
      sum = sum + abs(A[i] - B[i]);
      i = i + 1;
    }
    sad = sum;
  }
}

Sources: TSR, Katz, Boriello, Vahid, Perkowski
Circuit vs. Microprocessor

- **Circuit:**
  - Two states (S2 & S3) for each \(i\), 256 \(i\)'s \(\rightarrow\) 512 clock cycles
- **Microprocessor:**
  - Loop (for \(i = 1\) to 256), but for each \(i\), must move memory to local registers, subtract, compute absolute value, add to sum, increment \(i\)
  - Each loop iteration is approx 6 cycles per array item \(\rightarrow\) 256*6 = 1536 cycles
- **Circuit is \(~3\) times (300%) faster**
- It is possible to build a circuit that is much faster than this
  - think about how to leverage parallelism in HW!

\[
\begin{align*}
&i < 256 \quad \text{(i<256)}^* \\
\text{S2} & \quad \text{S3} \\
\text{i<256} & \quad \text{sum:=sum+abs(A[i]-B[i])} \\
\text{i:=i+1} & \quad \text{i<256} \\
\end{align*}
\]