CSE140L: Components and Design Techniques for Digital Systems Lab

Timing, Mux, Demux, Adders

Tajana Simunic Rosing
Outline

• Non-ideal gate behavior (3.5)
  – Rise/fall time
  – Delay
  – Pulse width
• Pass gates (Appendix B)
• Muxes & Demuxes (chap 4.2 pp. 171-183)
• Adders (chap 5.6)
Charge/discharge in CMOS

- Calculate on resistance
- Calculate capacitance of the gates circuit is driving
- Get RC delay & use it as an estimate of circuit delay
  - \( V_{\text{out}} = V_{\text{dd}} \left( 1 - e^{-t/RpC} \right) \)

Source: Prof. Subhashish Mitra
Which gate is faster?
Time behavior of combinational networks

- Waveforms
  - visualization of values carried on signal wires over time
  - useful in explaining sequences of events (changes in value)
- Simulation tools are used to create these waveforms
  - input to the simulator includes gates and their connections
  - input stimulus, that is, input signal waveforms
- Some terms
  - gate delay — time for change at input to cause change at output
    - min delay – typical/nominal delay – max delay
    - careful designers design for the worst case
  - rise time — time for output to transition from low to high voltage
  - fall time — time for output to transition from high to low voltage
  - pulse width — time that an output stays high or stays low between changes
Waveform view of logic functions

- Just a sideways truth table
  - but note how edges don’t line up exactly
  - it takes time for a gate to switch its output!

change in Y takes time to "propagate" through gates
When is non-ideal gate behavior a good thing?

- Can be useful — pulse shaping circuits
- Can be a problem — incorrect circuit operation (glitches/hazards)
- Example: pulse shaping circuit
  - $A' \cdot A = 0$
  - delays matter

![Diagram of pulse shaping circuit]

- $D$ remains high for three gate delays after $A$ changes from low to high
- $F$ is not always 0, pulse 3 gate-delays wide
Oscillatory behavior

- Another pulse shaping circuit
Muxes and demuxes

Tajana Simunic Rosing
Pass transistor – Mux building block

- Connects $X$ & $Y$ when $A=1$, else $X$ & $Y$ disconnected
  - $A_{\text{b}} = \text{not}(A)$
**Multiplexor (Mux)**

- Mux routes one of its $N$ data inputs to its one output, based on binary value of select inputs
  - 4 input mux $\rightarrow$ needs 2 select inputs to indicate which input to route through
  - 8 input mux $\rightarrow$ 3 select inputs
  - $N$ inputs $\rightarrow \log_2(N)$ selects
Mux Internal Design

- Selects input to connect to Y
  - selA == 1: connects A to Y
  - selB == 1: connects B to Y

Fig source: Prof. Subhashish Mitra
Multiplexers/selectors

- 2:1 mux: \( Z = A'I_0 + AI_1 \)
- 4:1 mux: \( Z = A'B'I_0 + A'BI_1 + AB'I_2 + ABI_3 \)
- 8:1 mux: \( Z = A'B'C'I_0 + A'B'CI_1 + A'BC'I_2 + A'BCI_3 + AB'C'I_4 + AB'CI_5 + ABC'I_6 + ABCI_7 \)

- In general: \( Z = \sum_{k=0}^{2^n-1} (m_k I_k) \)
  - in minterm shorthand form for a 2\(^n\):1 Mux
N-bit Mux Example

Four possible display items
- Temperature (T), Average miles-per-gallon (A), Instantaneous mpg (I), and Miles remaining (M) -- each is 8-bits wide
- Choose which to display using two inputs x and y
- Use 8-bit 4x1 mux
Multiplexers as general-purpose logic

- A $2^{n-1}$:1 multiplexer can implement any function of $n$ variables
  - with $n$-1 variables used as control inputs and
  - the data inputs tied to the last variable or its complement
- Example: $F(A,B,C) = m_0 + m_2 + m_6 + m_7$
Demultiplexers/decoders

- Decoders/demultiplexers: general concept
  - single data input, n control inputs, 2^n outputs
  - control inputs (called “selects” (S)) represent binary index of output to which the input is connected
  - data input usually called “enable” (G)

1:2 Decoder:
O0 = G \cdot S'
O1 = G \cdot S

2:4 Decoder:
O0 = G \cdot S1' \cdot S0'
O1 = G \cdot S1 \cdot S0
O2 = G \cdot S1 \cdot S0'
O3 = G \cdot S1' \cdot S0

3:8 Decoder:
O0 = G \cdot S2' \cdot S1' \cdot S0'
O1 = G \cdot S2' \cdot S1 \cdot S0
O2 = G \cdot S2 \cdot S1 \cdot S0'
O3 = G \cdot S2 \cdot S1 \cdot S0
O4 = G \cdot S2 \cdot S1' \cdot S0'
O5 = G \cdot S2 \cdot S1' \cdot S0
O6 = G \cdot S2' \cdot S1 \cdot S0'
O7 = G \cdot S2' \cdot S1 \cdot S0
Gate level implementation of demultiplexers

- 1:2 decoders
- 2:4 decoders
Demultiplexers as general-purpose logic (cont’d)

- $F_1 = A'BC'D + A'B'CD + ABCD$
- $F_2 = ABC'D' + ABC$
- $F_3 = (A' + B' + C' + D')$

![Diagram showing a 4:16 decoder with inputs A, B, C, D and outputs $A'B'C'D'$ to $ABCD$]
Mux and demux combination

- Uses in multi-point connections

![Diagram of Mux and Demux combination]

- Multiple input sources: A0, A1, B0, B1
- Multiple output destinations: S0, S1
- Sum: combined output

Diagram showing the use of MUX and DEMUX in a multi-point connection scenario.
Mux example: Logical function unit

- Multi-purpose function block
  - 3 control inputs to specify operation to perform on operands
  - 2 data inputs for operands
  - 1 output of the same bit-width as operands

<table>
<thead>
<tr>
<th>C0</th>
<th>C1</th>
<th>C2</th>
<th>Function</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>always 1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>A + B</td>
<td>logical OR</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>(A • B)’</td>
<td>logical NAND</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>A xor B</td>
<td>logical xor</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>A xnor B</td>
<td>logical xnor</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>A • B</td>
<td>logical AND</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>(A + B)’</td>
<td>logical NOR</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>always 0</td>
</tr>
</tbody>
</table>
CSE140: Components and Design Techniques for Digital Systems

Arithmetic circuits

Tajana Simunic Rosing
Example: 4-bit binary adder

- Inputs: A, B, Carry-in
- Outputs: Sum, Carry-out

\[ \begin{align*}
A & \quad B & \quad \text{Cin} \\
\rightarrow & & \rightarrow \\
\quad & & \quad S \\
& & \quad \text{Cout}
\end{align*} \]
Ripple-carry adder critical delay path

@0 \quad A \quad @1 \quad @N+1
@0 \quad B
@N \quad Cin
@0 \quad A \quad @1
@0 \quad B

late arriving signal
two gate delays to compute Cout

S0, C1 Valid
S1, C2 Valid
S2, C3 Valid
S3, C4 Valid
100

T0 \quad T2 \quad T4 \quad T6 \quad T8

4 stage adder
A0 \quad \rightarrow \quad S0 @2
B0 \quad \rightarrow \quad C1 @2
A1 \quad \rightarrow \quad S1 @3
B1 \quad \rightarrow \quad C2 @4
A2 \quad \rightarrow \quad S2 @5
B2 \quad \rightarrow \quad C3 @6
A3 \quad \rightarrow \quad S3 @7
B3 \quad \rightarrow \quad Cout @8
• Evaluate Sum and Ci+1
  – Sum = \( A_i \oplus B_i \oplus C_i \)
  – \( C_{i+1} = A_i B_i + A_i C_i + B_i C_i \)
    = \( A_i B_i + C_i (A_i \oplus B_i) \)
Carry-lookahead implementation

- Adder with propagate and generate outputs
Carry-lookahead implementation (cont’d)

- Carry-lookahead logic generates individual carries
  - sums computed much more quickly in parallel
  - however, cost of carry logic increases with more stages

```
0
A0 → S0 @2
B0 – C1 @2

A1 → S1 @3
B1 – C2 @4

A2 → S2 @5
B2 – C3 @6

A3 → S3 @7
B3 – Cout @8
```

```
0
A0 → S0 @2
B0 →

C1 @3
A1 → S1 @4
B1 →

C2 @3
A2 → S2 @4
B2 →

C3 @3
A3 → S3 @4
B3 →

C4 @3
```
Carry-lookahead adder
with cascaded carry-lookahead logic

- Carry-lookahead adder
  - 4 four-bit adders with internal carry lookahead
  - second level carry lookahead unit extends lookahead to 16 bits
Carry-select adder

- Redundant hardware to make carry calculation go faster
  - compute two high-order sums in parallel while waiting for carry-in
  - one assuming carry-in is 0 and another assuming carry-in is 1
  - select correct result once carry-in is finally computed
What we’ve covered thus far

- Xilinx Virtex II Pro board and tools
- Transistor design
- Delay estimates
- Pass transistors
- Muxes
- Demuxes
- Adders