Progettazione e sintesi di circuiti digitali

Lecture 2

Fundamentals of Digital Design
Combinational logic

- A combinational logic circuit is an electronic circuit implementing a Boolean function.
- After a finite propagation delay, the function takes a value determined by the current input value.
- The acyclic (i.e., without feedback) composition of combinational circuits is a combinational circuit.
Combinational logic

• A Boolean function is completely defined by:
  – a truth table or ...
  – a boolean expression

• Two canonical forms for boolean expression
  – sum of minterms or sum of products (SoP)
  – product of maxterms or product of sums (PoS)
  – minterm: logical product (AND) of all input variables in direct or complemented form
  – maxterm: logical sum (OR) of all input variables in direct or complemented form
Combinational logic

- A truth table can be conveniently represented by a Karnaugh Map (KM)

### Truth Table

<table>
<thead>
<tr>
<th>CBA</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>0</td>
</tr>
<tr>
<td>001</td>
<td>0</td>
</tr>
<tr>
<td>010</td>
<td>0</td>
</tr>
<tr>
<td>011</td>
<td>1</td>
</tr>
<tr>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>101</td>
<td>1</td>
</tr>
<tr>
<td>110</td>
<td>1</td>
</tr>
<tr>
<td>111</td>
<td>1</td>
</tr>
</tbody>
</table>

### Karnaugh Map

- **Product of maxterms (Sums) - PoS**
  
  \[ Y = (A + B + C) \cdot (\overline{A} + B + C) \cdot (A + \overline{B} + C) \cdot (A + B + \overline{C}) \]

- **Sum of minterms (Products) - SoP**
  
  \[ Y = \overline{C} \cdot B \cdot A + C \cdot \overline{B} \cdot A + C \cdot B \cdot \overline{A} + C \cdot B \cdot A \]
Combinational building blocks

• Logic blocks
  – multiplexers/demultiplexers
  – encoders/decoders
  – arbiters/priority encoders
  – shifters

• Arithmetic blocks
  – comparators
  – adders/subtractors
  – multipliers
Sequential logic

• The output is a function of the current input and of the state of the circuit
• State information depends on history of previous inputs and is stored in the circuit
  – state storage implies memory
• When state changes are synchronized with a clock signal, then the circuit is synchronous
  – Finite State Machine (FSM)
• When the state can change in response to any input variation, the circuit is asynchronous
Sequential logic examples

Asynchronous: the state of the SR latch changes whenever either S or R go high.

Synchronous: the D latch is transparent (Q follows D) when CLK is high; the state is stored when CLK goes low.

Synchronous: the positive-edge-triggered D flip-flop state can change only at the rising edge of CLK.
Sequential building blocks

• Multibit registers
• Shift registers
• Counters

Sequential building blocks for memories:
• Static memory cells
• Dynamic memory cells
Synchronous sequential circuits

- A general model for synchronous sequential circuits is the Finite State Machine (FSM)
  - In a FSM, a register, usually made of D flip-flops, stores the present state (state memory)
  - A block of combinational logic computes the next state and the output from the present state and the input
Synchronous sequential circuits

- FSM equations:

\[ S(k + 1) = NS(k) = F[S(k), X(k)] \]
\[ Y(k + 1) = G[S(k + 1), X(k + 1)] \]
\[ Q(k + 1) = Y(k) = G[S(k), X(k)] \]
Synchronous systems

• A useful model for a synchronous system:

- Control unit (CU)
- Arithmetic/logic unit (ALU)
- Memory
- Input/output management (I/O)
- Bus
- Pad ring
Control unit (CU)

• It is a FSM that generates the control signals driving the operation of the other blocks of the system. Examples:
  – read/write signal for memories
  – ALU operating codes
  – bus access control signals

• The CU inputs are feedback signals from the other system blocks

• A CU with no inputs can be realized as synchronous counter with a state decoder
Arithmetic/logic unit (ALU)

• Mostly combinational block devoted to data processing controlled by signals from the CU.
• Often called datapath.
• Most ALU operations are arithmetic or logic combinational functions.
• ALU often includes registers to store intermediate results.
• High speed datapath are realized with a pipeline structure.
Memory

• Read only (ROM): content may only be read, not changed.
• Random access (RAM): content may be read and changed (read/write); access to memory words through an address (random access).
• First-in, first-out (FIFO): implemented as a queue; when writing, words are queued; when reading, words are pulled out in FIFO order.
• Last-in, first-out (LIFO): implemented as a stack; when writing, words are stacked; when reading, words are pulled out in LIFO order.
Bus

• Parallel interconnections connecting one or more transmitters (TX) to one or more receivers (RX)

• Bus access in transmission must be regulated to prevent conflicts
  – only one TX at a time can access the bus, the other being in high impedance state
  – bus access is regulated by the CU
Basic digital design review

Combinational design example

• Step 0 – Specifications
  – design a circuit to detect prime numbers (including 1) between 0 and 15

• Step 1 – Architecture and interface definition
  – the required operation is combinational
  – 4-bit input, 1-bit output

(d, c, b, a): binary coding of an integer between 0 and 15
  d: most significant bit (MSB)
  a: less significant bit (LSB)
Combinational design example

- Step 2 – Input-output relation definition
  - a combinational circuit is completely defined by its truth table
  - in more complicated case an algorithmic description is required
Combinational design example

• Step 3 – Logic synthesis
  – transform the abstract description of the circuit (truth table) into a network of logic gates (gate netlist)
  – optimize the netlist (i.e., minimize the number of gates, reduce the input-output logic depth, ...)
  – logic optimization is based on the concept of implicant
  – a product of one or more input variables that, if true, implies that the function is also true, is called an implicant of the function
Combinational design example

• Step 3 (continued) – Logic synthesis
  – in a prime implicant, the removal of any literal results in a product term which is no more an implicant
  – if a minterm of a function is included in only one prime implicant, that implicant is said to be an essential implicant
  – a set of implicants is a cover of a function if each minterm of the function is included in at least one implicant of the cover
Combinational design example

• Step 3 (continued) – Logic synthesis

– in this example, all implicants are essential, thus the cover of the function contain the minimal number of logic products

\[ A \cdot B \cdot \overline{C} = D \cdot A \cdot B \cdot \overline{C} + \overline{D} \cdot A \cdot B \cdot \overline{C} \]

\[ B \cdot \overline{C} \cdot \overline{D} \]

\[ A \cdot \overline{D} \]

\[ A \cdot \overline{B} \cdot \overline{C} \]

\[ F = A \cdot B \cdot \overline{C} + B \cdot \overline{C} \cdot \overline{D} + A \cdot \overline{D} + A \cdot \overline{B} \cdot \overline{C} \]
Combinational design example

• Step 3 (continued) – Logic synthesis
  – the cover of the function can be translated directly to a logic circuit using AND, OR and NOT gates or ...
  – it can be mapped to a different set of gates

AND, OR, NOT realization

CMOS realization using only onverting gates
Combinational design example

• Summary of design flow

<table>
<thead>
<tr>
<th>REPRESENTATION DOMAIN</th>
<th>BEHAVIORAL</th>
<th>STRUCTURAL</th>
<th>PHYSICAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACTION LEVEL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYSTEM</td>
<td>Specifications</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REGISTER TRANSFER</td>
<td>Truth table</td>
<td>Block diagram</td>
<td></td>
</tr>
<tr>
<td>LOGICAL</td>
<td>Optimized Boolean equations</td>
<td>Gate netlist</td>
<td>Standard cells</td>
</tr>
</tbody>
</table>
Sequential design example

• Step 0 – Specifications
  – design a circuit to detect the sequences “1101” in a stream of binary values synchronized to a periodic signal CLK (clock)

• Step 1 – Architecture and interface definition
  – FSM with 1-bit input, 1-bit output
Sequential design example

• Step 2 – Input-output relation definition
  – a FSM is completely defined by a state diagram or a state table or by state equations

<table>
<thead>
<tr>
<th>PRESENT STATE</th>
<th>INPUT</th>
<th>NEXT STATE</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1 S0</td>
<td>X</td>
<td>N1 N0</td>
<td>Y</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>B</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>C</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>D</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>C</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>A</td>
<td>1</td>
</tr>
</tbody>
</table>
Sequential design example

• Step 3 – Logic synthesis
  – assign binary codes to states (state assignment):

  • binary encoding: states are ordered and encoded as binary integers
    • A → 00, B → 01, C → 10, D → 11
  • one-hot encoding: code with a number of bits equal to the number of states; only one bit high
    • A → 0001, B → 0010, C → 0100, D → 1000
  • output=state: for Moore FSM’s only; a code is chosen for each state so that it contains the output value

<table>
<thead>
<tr>
<th>PRESENT STATE</th>
<th>INPUT</th>
<th>NEXT STATE</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1 S0</td>
<td>X</td>
<td>N1 N0</td>
<td>Y</td>
</tr>
<tr>
<td>0 0</td>
<td>0</td>
<td>0 0</td>
<td>0</td>
</tr>
<tr>
<td>0 0</td>
<td>1</td>
<td>0 1</td>
<td>0</td>
</tr>
<tr>
<td>0 1</td>
<td>0</td>
<td>0 0</td>
<td>0</td>
</tr>
<tr>
<td>0 1</td>
<td>1</td>
<td>1 0</td>
<td>0</td>
</tr>
<tr>
<td>1 0</td>
<td>0</td>
<td>1 1</td>
<td>0</td>
</tr>
<tr>
<td>1 0</td>
<td>1</td>
<td>1 0</td>
<td>0</td>
</tr>
<tr>
<td>1 1</td>
<td>0</td>
<td>0 0</td>
<td>0</td>
</tr>
<tr>
<td>1 1</td>
<td>1</td>
<td>0 0</td>
<td>1</td>
</tr>
</tbody>
</table>

Example of state assignment using binary encoding in counting order
Sequential design example

- Step 3 (continued) – Logic synthesis – optimize the output and state equations

<table>
<thead>
<tr>
<th>X</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1 S0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0 1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1 1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1 0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ N1 = S1 \cdot S0 + S0 \cdot X \]

<table>
<thead>
<tr>
<th>X</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1 S0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0 1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1 1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1 0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ N0 = \overline{S1} \cdot \overline{S0} \cdot X + S1 \cdot \overline{S0} \cdot \overline{X} \]

<table>
<thead>
<tr>
<th>X</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1 S0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0 1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1 1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1 0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ Y = S1 \cdot S0 \cdot X \]
Sequential design example

• Step 3 (continued) – Logic synthesis
  – map the cover of the function to a library of gates
Sequential design example

- **Summary of design flow**

<table>
<thead>
<tr>
<th>REPRESENTATION DOMAIN</th>
<th>BEHAVIORAL</th>
<th>STRUCTURAL</th>
<th>PHYSICAL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ABSTRACTION LEVEL</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYSTEM</td>
<td>Specifications</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REGISTER TRANSFER</td>
<td>State diagram</td>
<td>Block diagram</td>
<td></td>
</tr>
<tr>
<td>LOGICAL</td>
<td>Optimized Boolean equations</td>
<td>Gate netlist</td>
<td>Standard cells</td>
</tr>
</tbody>
</table>
Basic digital design summary

• The “manual” design flow just reviewed is suitable for circuit of limited complexity
  – design with SSI/MSI/LSI off-the-shelf components like PLA, PAL, PROM, GAL

PLA: programmable logic array
GAL: generic array logic
Off-the-shelf GAL’s