Evolution of Implementation Technologies

- Discrete devices: relays, transistors (1940s-50s)
- Discrete logic gates (1950s-60s)
- Integrated circuits (1960s-70s)
  - e.g. TTL packages: data book for 300s of different parts
  - Map your circuit to the Data Book parts
- Gate Arrays (IBM, 1970s)
  - “Custom” integrated circuit chips
  - Design using a library (like TTL)
  - Transistors are already on the chip
  - Place and route software puts the chip together automatically
  - Large circuits on a chip
  - Automatic design tools (no tedious custom layout)
  - Only good if you want 300s of parts

Gate Array Technology (IBM - 1970s)

- Simple logic gates
  - Use transistors to implement combinational and sequential logic
- Interconnect
  - Wires to connect inputs and outputs to logic blocks
- I/O blocks
  - Special blocks or peripheral for external connections
- Add wires to make connections
  - Done when chip is fabricated
  - “mask-programmed”
  - Construct any circuit

Programmable Logic

- Disadvantages of the Data Book method
  - Constrained to parts in the Data Book
  - Parts are necessary small and standard
  - Need to stock many different parts
- Programmable logic
  - Use a single chip (for a small number of chips)
  - Program it for the circuit you want
  - No reason for the circuit to be small

Programmable Logic Technologies

- Fuse and anti-fuse
  - Fuse makes or breaks interconnections between wires
  - Typical connection is 50-300 ohm
  - One-time programmable (testing before programming)
  - Very high density
- EPROM and EEPROM
  - Highpower consumption
  - Typical connections are 2K-4K ohm
  - Fairly high density
  - RAM-based
    - Memory bit controls a switch that connects/disconnects two wires
    - Typical connections are 5K-3K ohm
    - Can be programmed and re-programmed in the circuit
    - Low density

Programmable Logic

- Program a connection
  - Connect two wires
  - Set bit to 0 or 1
- Regular structures for two-level logic (1960s-70s)
  - All rely on two-level logic minimization
  - PROM connections - permanent
  - EPROM connections - erase with UV light
  - EEPROM connections - erase electrically
  - PROMs
    - Program connected in the upper plane
  - PALs
    - Program the connect in the upper plane
  - PAL2
    - Program the connect in the upper plane

Making Large Programmable Logic Circuits

- Alternative 1: “CLD”
  - Put a lot of PLDs on a chip
  - Add wires between them whose connections can be programmed
  - Use fuse/EPROM technology
- Alternative 2: “FPGA”
  - Emulate gate array technology
  - Hence: Field Programmable Gate Array
  - You need:
    - A way to implement logic gates
    - A way to connect them together
Field-Programmable Gate Arrays

- PALs, PLAs = 10 - 1000 Gate Equivalents
- Field Programmable Gate Arrays = FPGAs
- Altera MAX Family
- Actel Programmable Gate Array
- Xilinx Logical Cell Array
- 100 - 1000(s) of Gate Equivalents!

Tradeoffs in FPGAs

- Logic block - how are functions implemented; fixed function, manipulated inputs or programmable?
- Support complex functions, need fewer blocks, but are bigger so less of them on chip
- Support simple functions, need more blocks, but they are smaller so more of them on chip

Interconnect
- How are logic blocks arranged?
- How many wires will be needed between them?
- Are wires evenly distributed across chip?
- Programmability – how wires drawn - are some wires specialized for long distances?
- How many inputs, outputs must be routed to/from each logic block?
- What utilization are we willing to accept? 50% 60% 90%?

Altera EPLD (Erasable Programmable Logic Devices)

- Historical Perspective
  - PALs: same technology as a programmed array, only 128 bit ROM
  - EPLDs: PROMs replace EPROMs, EPROMs are irradiated by UV light to make them Programmable
  - Altera building block = MACROCELL

Altera EPLD

- EPLDs contain 8 to 48 independently programmed macrocells
- Addressable by EPROM
- Synchronous Mode

- Rising edge triggered
- Controls a clock signal

- Asynchronous Mode

- Falling edge triggered
- Controls a clock signal

- Support for edge triggered and clock synchronous mode

- SSTL2 to SSTL1

Altera Multiple Array Matrix (MAX)

- AND-OR structures are user defined and limited
  - Field-programmable Interconnect
  - 32 EPLDs/LAB
  - 64 EPLDs/LAB
  - 32 Routing/LAB
  - 32 EPLDs/EXLAB

- Global Routing: Programmable Interconnect Array

- MAX fields: 32 fixed inputs, 32 I/O pins, 32 LABs, 64 MAX macros/4 LABs
Xilinx Programmable Gate Arrays

- CLB - Configurable Logic Block
  - 5-input, 1-output function
  - or 2-4-input, 1-output functions
  - optional register on outputs
- Built-in fast carry logic
- Can be used as memory
- Three types of routing
  - direct
  - general-purpose
  - long lines of various lengths
- RAM-programmable
  - can be reconfigured

The Xilinx 4000 CLB

Two 4-input functions, registered output

5-input function, combinational output

CLB Used as RAM
Xilinx FPGA Combinational Logic Examples

- Key: General functions are limited to 5 inputs
  - Even better - 1/2 CLB
  - No limitation on function complexity
- Example:
  - 2-bit comparator:
    A = B = C = D and A + B + C \text{D} implemented with 1 CLB
    \[ (\text{EQ}) \quad G = A'B'C'D' + A'B C'D + A B'C D' + A B C D \]
- Can implement some functions of \( > 5 \) inputs

Xilinx FPGA Adder Example

- Example:
  - 2-bit binary adder - inputs: A1, AO, B1, B0, CIN
    \[ \text{output: } \text{S0, S1, Cout} \]
  - Full Adder: 4 CLB delay to find carry out
  - 2 x Two-bit Adders (3 CLB each) yields 2 CLB to find carry out

Computer-Aided Design

- Can't design FPGAs by hand
  - Too much logic to manage, hard to make changes
- Hardware description languages
  - Specify functionality of logic at a high level
- Validation: high-level simulation to catch specification errors
  - Verify pin-outs and connections to other system components
- Low-level to verify mapping and check performance
- Logic synthesis
  - Process of compiling HDL programs into logic gates and flip-flops
- Technology mapping
  - Map the logic into elements available in the implementation technology

Xilinx CAD Tools

- Verilog (or VHDL) use to specify logic at a high-level
  - Combine with schematics, library components
- Synthesis
  - Compiles Verilog to logic
  - Maps logic to the FPGA cells
  - Optimizes logic
- Xilinx APR - automatic place and route (simulated annealing)
  - Provides controllability through constraints
  - Handles global signals
- Xilinx Xst - measure delay properties of mapping and add in iteration
- Xilinx XACT - design editor to view final mapping results

Xilinx FPGA Combinational Logic

- Examples:
  - 5-input majority function: 1 whenever \( n/2 \) or more inputs are 1
  - 5-input parity functions: 5 inputs/CLB; 2 levels yield 25 inputs

CAD Tool Path (cont'd)

- Placement and routing
  - Assign logic blocks to functions
  - Make wiring connections
- Timing analysis - verify paths
  - Determine delays as routed
  - Look at critical paths and ways to improve
- Partitioning and constraining
  - If design does not fit or is unrouteable as placed split into multiple chips
  - If design too slow prioritize critical paths, fix placement of cells, etc.
  - Few tools to help with these tasks exist today
  - Generate programming files - bits to be loaded into chip for configuration
Applications of FPGAs

1. Implementation of random logic
   - Easier changes at system level (one device is modified)
   - Can eliminate need for full-custom chips
2. Prototyping
   - Whole system of gate arrays used to simulate a circuit to be manufactured
   - Get more/less/better debugging done than with simulation
3. Configurable hardware
   - One hardware block used to implement more than one function
   - Functions must be mutually-exclusive in time
   - Can greatly reduce cost while enhancing flexibility
   - RAM-based only option
4. Special-purpose computation engines
   - Hardware dedicated to solving one problem (or class of problems)
   - Accelerators attached to general-purpose computers

Implementation Strategies

- ROM-based Design
- Example: BCD to Excess 3 Series Converter

<table>
<thead>
<tr>
<th>BCD</th>
<th>Excess 3 Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>0001 0100</td>
</tr>
<tr>
<td>0011</td>
<td>0111 0110</td>
</tr>
<tr>
<td>0101</td>
<td>0101 1000</td>
</tr>
<tr>
<td>0111</td>
<td>1011 1000</td>
</tr>
<tr>
<td>1001</td>
<td>1101 1000</td>
</tr>
<tr>
<td>1011</td>
<td>1111 1000</td>
</tr>
<tr>
<td>1101</td>
<td>1111 1000</td>
</tr>
</tbody>
</table>

Implementation Strategies

- ROM-based Implementation
- Circuit Level Resolution 74175 + 4 positive edge-triggered FFs
- Truth Table/ROM/IOs
- In ROM-based design, no need to consider state assignment

Implementation Strategies

- PLA-based Design
- State Assignment with NOVA

Timing behavior for input string 0000(0) and 1110(7)

<table>
<thead>
<tr>
<th>LSB</th>
<th>MSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>1110</td>
</tr>
<tr>
<td>1100</td>
<td>0010</td>
</tr>
</tbody>
</table>

Implementation Strategies

- PLA-based Design
- State Assignment with NOVA

NOVA derived state assignment

NOVA input file

9 product term implementation
Implementation Strategies

Specifying PALs with ABEL

Specifying PALs with ABEL

FSM Design with Counters

FSM Design with Counters
FSM Design with Counters

Excess 3 Converter

<table>
<thead>
<tr>
<th>Input</th>
<th>Counted</th>
<th>Flip-Flop</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>+1</td>
<td>x</td>
</tr>
<tr>
<td>y</td>
<td>+1</td>
<td>y</td>
</tr>
<tr>
<td>z</td>
<td>+1</td>
<td>z</td>
</tr>
<tr>
<td>x'</td>
<td>+1</td>
<td>x'</td>
</tr>
<tr>
<td>y'</td>
<td>+1</td>
<td>y'</td>
</tr>
<tr>
<td>z'</td>
<td>+1</td>
<td>z'</td>
</tr>
<tr>
<td>x''</td>
<td>+1</td>
<td>x''</td>
</tr>
<tr>
<td>y''</td>
<td>+1</td>
<td>y''</td>
</tr>
<tr>
<td>z''</td>
<td>+1</td>
<td>z''</td>
</tr>
</tbody>
</table>

CLR signal determines LD which determines Count

Implementing FSMs with Counters

<table>
<thead>
<tr>
<th>Excess 3 Converter</th>
<th>Excess 3 Converter</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1</td>
<td>+1</td>
</tr>
<tr>
<td>+1</td>
<td>+1</td>
</tr>
<tr>
<td>+1</td>
<td>+1</td>
</tr>
<tr>
<td>+1</td>
<td>+1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Excess 3 Converter</th>
<th>Excess 3 Converter</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1</td>
<td>+1</td>
</tr>
<tr>
<td>+1</td>
<td>+1</td>
</tr>
<tr>
<td>+1</td>
<td>+1</td>
</tr>
<tr>
<td>+1</td>
<td>+1</td>
</tr>
</tbody>
</table>

Implementation Strategies

Xilinx LCA Architecture

Implementing the BCD to Excess 3 FSM

Q0 = + Q1 + Q2 + Q3
Q2 = + Q0 + Q2 + Q3
Q3 = + Q2 + Q3
Z = + Z1 + Q1

No function more complex than 4 variables
4 FSM implies 2 CLBs

Synchronous Mealy Machine

Global BIST to be used

Place Q0, Q2 in upper CLB
Q1, Q3 in lower CLB

maximum use of direct general purpose interconnections

Design Case Study

Traffic Light Controller

Description into primitive subsystems

- Controller FSM
- next state/output function
- timing buffer
- short time long time
- timer counter
- car sensor
- output decoders and traffic lights
Design Case Study
Traffic Light Controller
Block Diagram

Design Case Study
State Assignment: H0 = 0Q, H1 = 1Q, T0 = 0Q, T1 = 1Q
F0 = T0 Q0 + T0 Q0 + T0 Q0 + T0 Q0
F1 = T1 Q1 + T1 Q1 + T1 Q1 + T1 Q1
ST = C.TQ + ZQ Q0 + T5 Q1 Q0
HL(I) = T1 Q1 Q0 + T5 Q1 Q0
HL(II) = T5 Q0 Q0 + T5 Q0 Q0
R(I) = TQ
R(II) = T5 Q0 Q0 + T5 Q0 Q0
PAL/PAL Implementation:
5 inputs, 2 outputs, 8 product term
PAL, 250 I/O = 1, 10 prog.
ROM Implementation:
32 word by 8 bit ROM (256 bits)

Design Case Study
Counter-based Implementation

Design Case Study
LCA-Based Implementation
Discrete Gate Method:
None of the functions exceed 5 variables
F0, ST are 5 variable (1 CLB each)
R0, HL0, RL0 are 3 variable (1/2 CLB each)
F1 is 1 variable (1/2 CLB)
4 (1/2) CLB total

Design Case Study
Subsystem Logic

Design Case Study
Light Decoders
Car Detector
Interval Timer

Design Case Study
Next State Logic

Design Case Study
Light Controller is Synchronous Output
Light Controllers are Synchronous Outputs
Design Case Study

LCA-Based Implementation

Placement of function selected to minimize the use of direct connections.

Design Case Study

LCA-Based Implementation

Counter/Multiplexer Method:

4: MUX, 2 bit Upcounter

MUX of 4 variables (4 data, 2 control)

But this is reduced to 6 variables function that can be implemented in 1 CLB

2nd CLB to implement TL, C and TL + C

But note that ST/ST is really a function of TL, C, TS, Q0, QD

In CLB to implement the function of 5 variables

2 bit Counter: 2 functions of 3 variables (2 bit state + count)

Also implemented in one CLB

Top flip: High decoder: function of 2 variables (Q2, Q3)

2 per CLB = 3 CLB for the 6 lights

Total count = 5 CLBs