Bryan Melanson

How to Not Fail

Digital System Design

While never going to class

Computer Engineering 2020 $\mathcal{BM}$

Contents

1 Implementation Technology

1.1 VLSI

”Very Large-Scale Integration” refers to circuit devices that contain thousands to millions of transistors.
Pros

1. Complete control overall aspects of circuit implementation
2. Can optimize for speed, area, power as needed
3. Allows you to achieve highest device performance
4. Cost effective for large volume designs
5. Facilitates mixed-signal design

Cons

1. Manufacturing is costly
2. Mistakes are costly
3. Verification process is long and thorough
4. Design process is slow - long time to market
5. Device functionality is static - no way to add system features

1.2 ASIC

”Application Specific Integrated Circuit” same manufacturing as VLSI, with reduced design time, made of predefined components. These libraries are accessed with a HDL based design and synthesis process, and then mapped to an implementation in the target technology.
Pros

1. Faster design time than VLSI
2. Some ability to optimize for speed, area, power as needed
3. Allows you to achieve good device performance
4. Can be cost effective for large volume designs

5. Faster testing and verification than VLSI

Cons

1. Manufacturing is costly
2. Mistakes are costly
3. Testing and verification is slow
4. Design and manufacturing is slow - long time to market
5. Once manufactured, device functionality is static - no way to add system features

1.3 Programmable Logic

1.3.1 Simple Programmable Logic Devices

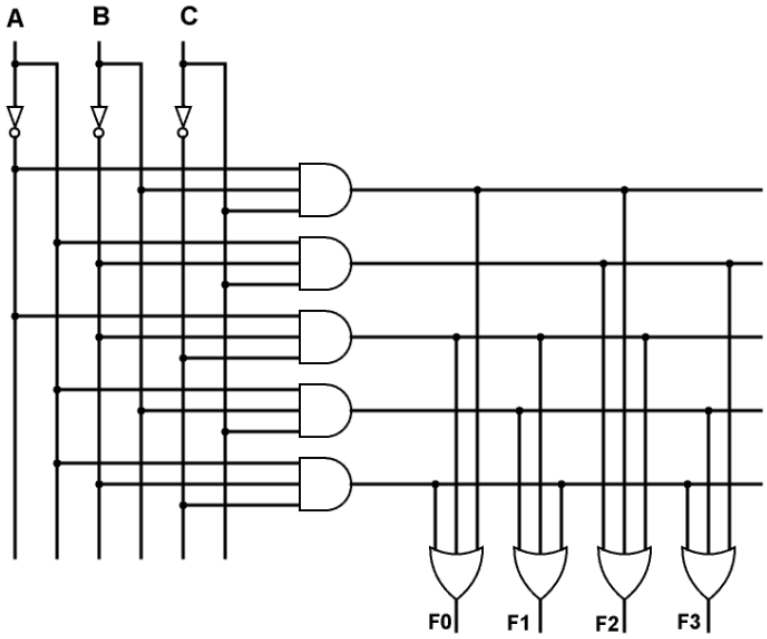
Programmable logic forms were factory programmable such as ROMs and MPGAs (mask programmable gate arrays). The connection between these devices must be designed by the designer, which shortens design time compared to VLSI and ASIC.

1.3.2 ROMs/LUTs

ROMs were intended as memory devices (such as cartridges) but could be used for logic functions. ROMs can store a truth table to implement combinational logic. Each ROM has n address lines and m data outputs.

1.3.3 PLAs/PLDs

Programmable Logic Arrays (PLA) are based on the notion that any logic function is a combination of product terms OR'd together.



1.4 FPGAs

Field Programmable Gate Arrays are flexible programmable logic for larger designs. Complicated to program, but offer large capacities and flexibility.

1. Programmable I/O blocks - Pin configuration
2. Programmable Logic blocks - LUTs, AND-OR, multiplexers, etc.
3. Programmable Routing Resources - Routing between the pins and blocks

Pros

1. Faster design time than VLSI
2. Reduced cost of design errors
3. Flexibility for new requirements
4. Cost effective when low number of devices

Cons

1. Lower logic density (unused elements take up space)

2. Higher power consumption
3. Not applicable to all designs - floating point

2 Intro to VHDL

VHDL ("VHSIC" (Very High Speed Integrated Circuit) Hardwre Description Language) describes digital circuits but can be used to generate circuits automatically (*synthesis*). This facilitates top down design, and provides reusable elements and mature design practices. This can reduce the cost of prototyping complicated systems.

2.1 Timing

```
C <= A and B after 5 ns
```

The **after** will ensure A and B are assigned to C 5 ns after being evaluated.

```
C <= reject 3 ns inertial A and B after 5 ns
```

If a pulse occurs for less than 3 ns, it will be rejected. If longer than the rejection time, it will be propogated after a 5 ns delay. Inertial delay is the default delay assumed by VHDL.

```
C <= transport A after 10 ns
```

Transport will delay the signals without rejecting.

2.2 Data Types

1. bit - 0, or 1
2. bit vector - an array of 0, 1
3. std logic - 0, 1, X, U, etc
4. std logic vector - array of std logic

2.3 VHDL Modules

Entity similar to a function prototype. Describes port inpus and outputs.

Architecture is a description of the internals.

```
entity two_gates is
  port( A, B, D: in bit;
        E: out bit);
end two_gates;

architecture gates of two_gates is
  signal C: bit;
begin
  -- Comments are started with 2 dashes and run to the end
  -- of the line.
  C <= A and B; -- This line is concurrent with
  E <= C or D;  -- this line.
end gates;
```

The devices can be reused in separate files by declaring it as a component and instantiating each element individually, assigning pins.

```
entity Adder4 is
  port(A, B: in bit_vector(3 downto 0); Cin: in bit; -- Inputs
       Sum: out bit_vector(3 downto 0); Cout: out bit); -- Out
end Adder4;

architecture Structure of Adder4 is
  -- Here we declare that we are going to use the
  -- FullAdder component.
  component FullAdder
    port (X, Y, Cin: in bit;
          Cout, Sum: out bit);
  end component;

  -- We also need to declare an internal signal for carries
  -- between the adders; since our diagram used 0, 1, 2...
  signal C: bit_vector(2 downto 0);
begin
  -- Now, instead of writing equations using concurrent
  -- assignments here, we need to instantiate four copies
  -- of the FullAdder component and connect them appropriately
  FA0: FullAdder port map (A(0), B(0), Cin, C(0), Sum(0));
  FA1: FullAdder port map (A(1), B(1), C(0), C(1), Sum(1));
  FA2: FullAdder port map (A(2), B(2), C(1), C(2), Sum(2));
  FA3: FullAdder port map (A(3), B(3), C(2), Cout, Sum(3));
end Structure;
```

2.4 Sequential Statements

The **process** statement will monitor a signal using a sensitivity list and trigger a sequence of lines to be completed in order.

```
process(A, B, C, D)
begin
  C <= A and B;
  E <= C or D;
end process;
```

2.5 Flip Flops

```
process(CLK)
begin
  if CLK'event and CLK = '1' then -- rising edge trigger
    Q <= D;
  end if;
end process;
```

2.6 Conditional Assignments

```
signal_name <= expression1 when condition1
               else expression2 when condition2
               [else expressionN];

sel <= S0 & S1;

process(A, B, C, D, sel)
begin
  case sel is
    when "00" => F <= A;
    when "01" => F <= B;
    when "10" => F <= C;
    when "11" => F <= D;
  end case;
end process;
```

2.7 User Defined Types

```
 -- Define a state_type enumeration.
type state_type is (S0, S1, S2, S3, S4, S5);

-- Declares variable of type state_type and
-- initializes it so S1. (By default, would
-- initialize to leftmost item, S0 in this case.)
signal state: state_type := S1;

-- Assume signals A, B, C, D are declared elsewhere
-- as bit_vector(7 downto 0).
A <= "10010101";
B <= A sll 2; -- shifts A left 2 positions,
              -- inserting zeros: 01010100
C <= A sra 3; -- shift right arithmetic 3 positions,
              -- inserting leftmost bit: 11100101
D <= A ror 5; -- rotates 5 positions right: 10101100
```

3 Advanced Design

3.1 Logic Minimization

A	B	C	$y = f(A,B,C)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$y - A'BC + AB'C + ABC' + ABC$ can be reduced to **minterms**.

$\sum m(3,5,6,7)$

This is known as the Standard Sum of Products form - this can also incorporate Don't-Care:

$\sum m(3,5,6,7) + \sum d(0,3)$

3.2 Quine McCluskey Method

This reduces the SSOP form to a minimum sum of products by eliminating as many literals as possible to produce prime implicants, and then createing a chart of implicants to be OR'd to implement the original function.
subsection
Recall $XY + XY' = X$ - use this to combine minterms if they differ in one minterm. Terms are then represented as binary, for example $AB'CD' = 1010$, so $AB'CD' + AB'CD = AB'C$ or $101-$.

Column I			Column II			Column III	
Group 0	0	0000	Group 4	0,1	000−	Group 7	0,1,8,9 −00−
Group 1	1	0001		0,2	00−0		0,2,8,10 −0−0
	2	0010		0,8	−000		~~0,8,1,9 −00−~~
	8	1000	Group 5	1,5	0−01		~~0,8,2,10 −0−0~~
Group 2	5	0101		1,9	−001	Group 8	2,6,10,14 −−10
	6	0110		2,6	0−10		~~2,10,6,14 −−10~~
	9	1001		2,10	−010		
	10	1010		8,9	100−		
Group 3	7	0111		8,10	10−0		
	14	1110	Group 6	5,7	01−1		
				6,7	011−		
				6,14	−110		
				10,14	1−110		

The numbers which don't reduce to the following column or are redundant are *prime implicants*. In this case, the prime implicants are $0-01$, $01-1$, $011-$, $-00-$, $-0-0$, and $--10$.

midterms →

Prime Implicants		0	1	2	5	6	7	8	9	10	14
0,1,8,9	b'c'	X	X					X	X		
0,2,8,10	b'd'	X		X				X		X	
2,6,10,14	cd'			X		X				X	X
1,5	a'c'd		X		X						
5,7	a'bd				X		X				
6,7	a'bc					X	X				

Prime Implicants		0	1	2	5	6	7	8	9	10	14
0,1,8,9	$b'c'$	X	X					X	X		
0,2,8,10	$b'd'$	X		X				X		X	
2,6,10,14	cd'			X		X				X	X
1,5	$a'c'd$		X		X						
5,7	$a'bd$				X		X				
6,7	$a'bc$					X	X				

Choose the values with a single X in its column, then draw a horizontal line through the row. Any intersections, draw a vertical line. Continue until all X are crossed out, choosing values which will cover off as much as possible.

4 Hazards and Testing

4.1 Hazards in Combinational Circuits

4.1.1 Static 1 Hazard

If a circuit goes 0 when it should have been 1

4.1.2 Static 0 Hazard

A circuit goes 1 when it should be constant 0

4.1.3 Dynamic Hazard

When the circuit should transition from 0 to 1 or 1 to 0, but transitions three or more times

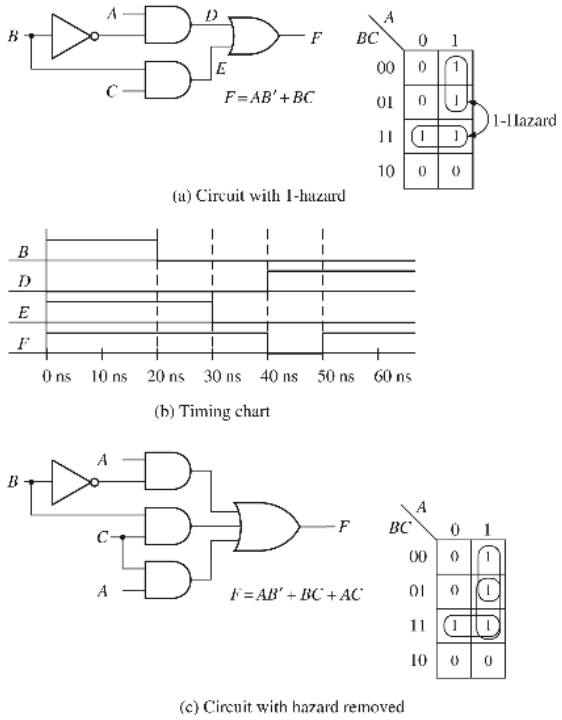


FIGURE 1-10:
Elimination of 1-Hazard

Select inputs to test specific port errors:

a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	Faults Tested
1	1	1	0	X	X	0	X	X	*a*0, *b*0, *c*0, *p*0
0	X	X	1	1	1	0	X	X	*d*0, *e*0, *f*0, *q*0
0	X	X	0	X	X	1	1	1	*g*0, *h*0, *i*0, *r*0
0	1	1	0	1	1	0	1	1	*a*1, *d*1, *g*1, *p*1, *q*1, *r*1
1	0	1	1	0	1	1	0	1	*b*1, *e*1, *h*1, *p*1, *q*1, *r*1
1	1	0	1	1	0	1	1	0	*c*1, *f*1, *i*1, *p*1, *q*1, *r*1

4.2 Scan Testing

5 Memory and Arithmetic

6 More VHDL

7 System Design

7.1 AMBA/AXI

AMBA is a protocol created by ARM for components designed for other companies. This is an open standard on-chip interconnect for connection and management of blocks in a System-On-Chip.

AMBA has four interface protocols, *AXI* (Advanced Extemsible Interface), *AHB* (Advanced High Performance Bus), *APB* (Advanced Peripheral Bus), and *ATB* (Advanced Trace Bus).

AXI4-Lite is a subset of AXI4 used for simpler control register style interfaces. All transactions are burst legnths of 1, data accesses are the same size and width as the data bus, and exclusive access is not supported.

	AXI4	AXI4-Lite	AXI4-Stream
Intended for:	high-performance and memory-mapped systems	register-style interfaces	non-address-based components
Burst size:	up to 256	1	unlimited
Data width:	32 to 1024 bits	32 or 64 bits	any number of bytes
Applications:	embedded, memory	small footprint control logic	DSP, video, communication

Channel is a collection of signals associated to a VALID signal.

Interface is a collection of 1 or more channels that expose an IP's core function. An IP core can have multiple interfaces.

Bus is a multiple bit signal (may be part of an interface or channel, but distinct).

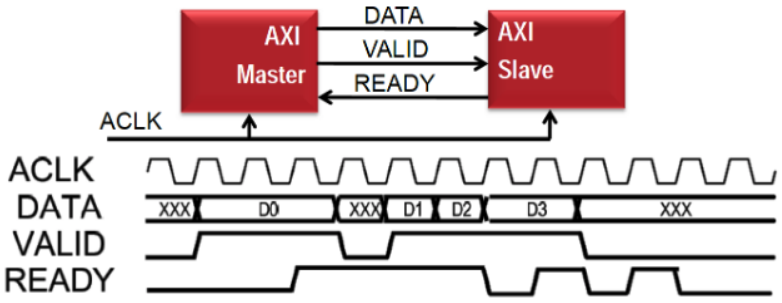
Transfer is a single clock cycle where information is communicated qualified by a VALID handshake.

Transaction is a complete communication operator across a channel composed of one or more transfers.

Burst is a transaction consisting of more than 1 transfer.

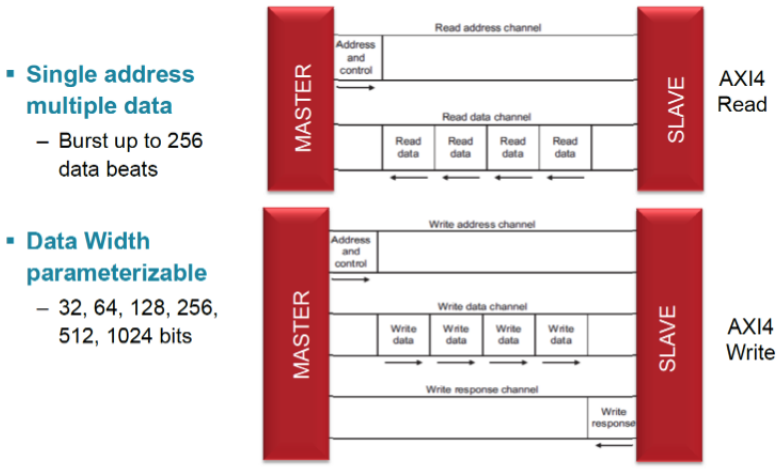
An AXI4 handshake is as follows:

1. Master asserts a VALID when data is available
2. Slave asserts READY if able to accept
3. Date is transferred when VALID and READY
4. If the transaction has more than 1 transfer, next data is sent, otherwise VALID is removed
5. If slave cannot accept, READY is removed to pause

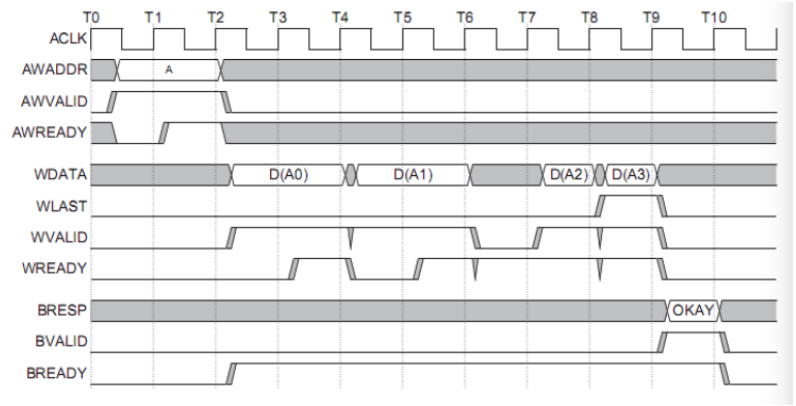


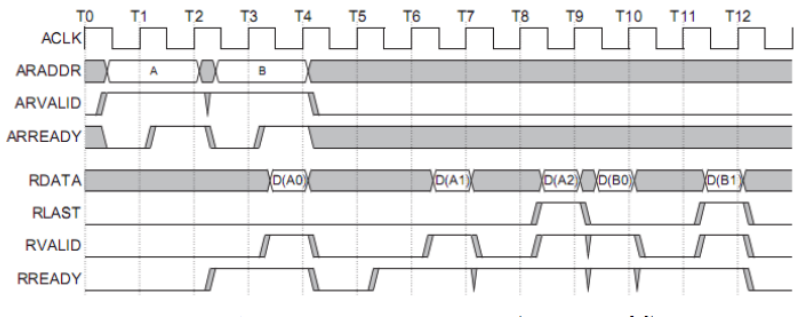
AXI4 and variants all have 5 basic signalling channels:

1. Read address
2. Read data
3. Write address
4. Write data
5. Write response (Slave to Master)



For AXI-Lite, only 1 data tranfer per transaction.





Write Address Channel (AXI4-Lite)

AWVALID - The address write valid signal (master to slave)
AWREADY - The address write ready signal (slave to master)
AWADDR - The target write address
AWPROT - The protection type specifying privilege and security level

Write Data Channel (AXI4-Lite)
WVALID - The data write valid signal (master to slave)
WREADY - The data write ready signal (slave to master)
WDATA - The n-bit write data
WSTRB - The write strobe lines indicate with bytes hold valid data (1 bti for each 8 bit data)

Write Response Channel (AXI4-Lite)
BVALID - The write response valid signal (slave to master)
BREADY - The write reponse ready signal (master to slave)
BRESP - Write reponse indicating state of transaction

Read Address Channel (AXI4-Lite)
ARVALID - The address read valid signal (master to slave)
ARREADY - The address read ready signal (slave to master)
ARADDR - The target read address
ARPROT - The protection type specifying privilege and security level

Read Data Channel (AXI4-Lite)
RVALID - The data read valid signal (master to slave)
RREADY - The data read ready signal (slave to master)
RDATA - The n-bit read data
RRESP - Read response associated with the data, similar to BRESP

7.2 Off-Chip Protocols

7.2.1 Serial Peripheral Interface

Synchronous serial communication used for short distance communication in embedded systems. Uses a master-slave architecture with one or more slaves.

Four signals:

1. SCLK - Clock signal that synchronizes data to clock edges

2. MOSI - Master-out-Slave-in serial data line (master to slave)
3. MISO - Master-in-Slave-Out serial data from slave to master
4. SS - Slave select chooses which slave to communicate with

7.2.2 I^2C

Communication protocol for attaching low-speed peripheral controllers. Short-distance intra-board communication. Multiple masters and slaves with bidirectional lines, Serial Data Line (SDA) and Serial Clock Line (SCL).

Four modes of operation:

1. Master transmit
2. Master receive
3. Slave transmit
4. Slave receive