MHV FPGA Workshop

Series 1

Welcome

* Stephen Davies, MHV login sjdavies
* Space facilities
  * Bathrooms/Refreshments
  * WiFi
* Free workshop, donations welcome

Series Goals

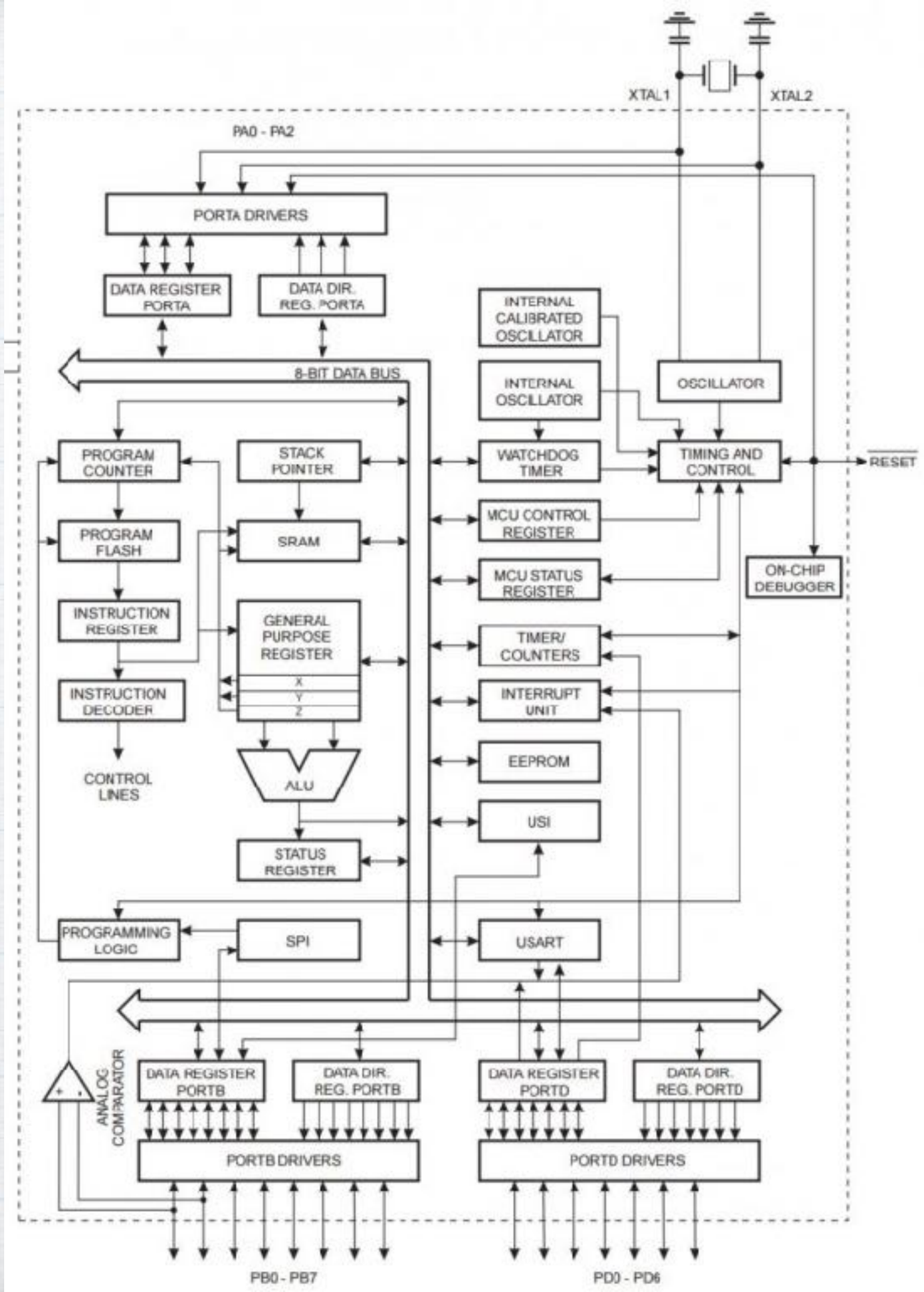
* FPGA programming in Verilog
  * Assume no Verilog skills
    * Some programming experience e.g. Arduino
  * Assume no digital design skills
  * 8 bit processor as challenge

Todays Goals

* Basic Verilog coding and testing
* AVR Architecture basics
* CPU outline
* Implement 3 basic instructions
* Parallel thinking

Atmel AVR

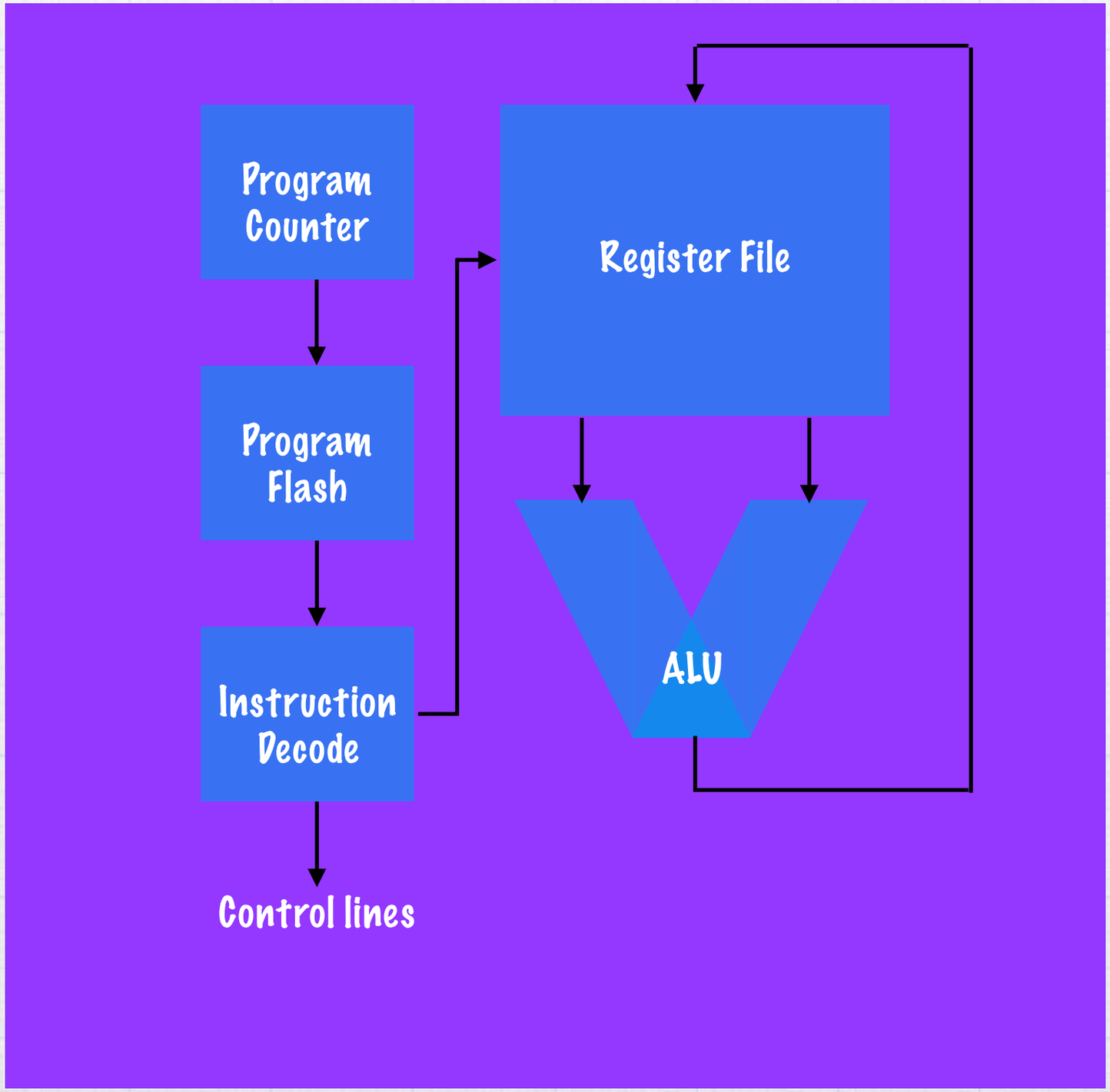
* First released 1996
* Device family
  * 6 pins, attiny10, USD 0.24
  * 64 pins, atmega2561, USD 8.51
* Processor core very consistent over entire family
* Arduino uses atmega328

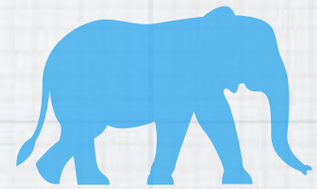

XTAL1
XTAL2
PA0 - PA2
PORTA DRIVERS
DATA REGISTER PORTA
DATA DIR. REG. PORTA
8-BIT DATA BUS
INTERNAL CALIBRATED OSCILLATOR
INTERNAL OSCILLATOR
OSCILLATOR
PROGRAM COUNTER
STACK POINTER
WATCHDOG TIMER
TIMING AND CONTROL
RESET
PROGRAM FLASH
SRAM
MCU CONTROL REGISTER
MCU STATUS REGISTER
ON-CHIP DEBUGGER
INSTRUCTION REGISTER
GENERAL PURPOSE REGISTER
X
Y
Z
TIMER/ COUNTERS
INTERRUPT UNIT
INSTRUCTION DECODER
CONTROL LINES
ALU
EEPROM
USI
STATUS REGISTER
PROGRAMMING LOGIC
SPI
USART
ANALOG COMPARATOR
DATA REGISTER PORTB
DATA DIR. REG. PORTB
DATA REGISTER PORTD
DATA DIR. REG. PORTD
PORTB DRIVERS
PORTD DRIVERS
PB0 - PB7
PD0 - PD6

AVR Architecture

- 8 bit
- Instructions are 16 bits wide
  - 2k flash = 1k instructions
- Register file
  - 32 x 8 bits (in general)
  - 16 x 8 bits (some tiny)

Stage 1 Roadmap





Tasty...

Demo 1

* Workshop repository tour
  * Design notes
  * External resources
  * Exercise structure
  * Sample files

Lab 02 - Project Setup

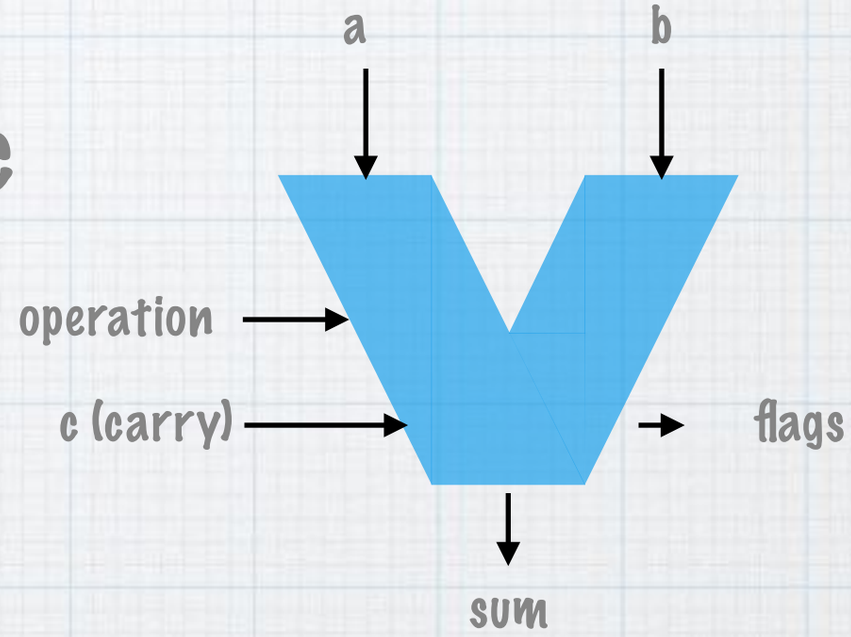
* Create Vivado project
* Copy template files
* Add files to project
* Run behavioural simulation
* Look around some of the .v files

Combinational Logic

* Output value(s) are derived from current inputs
    * Has no 'state' (memory)
* Combination of basic functions
    * Not
    * And
    * Or
* ALU is purely combinational logic

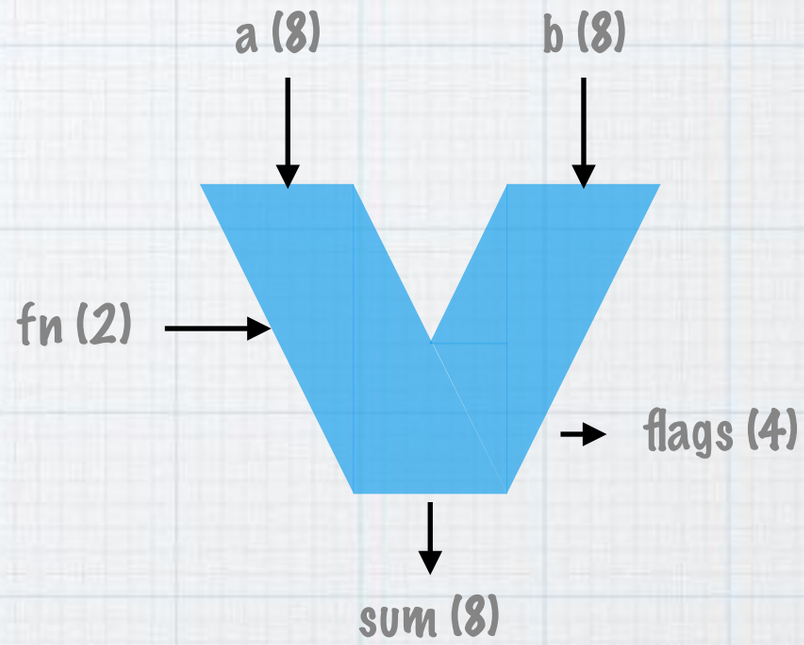
ALU

* Arithmetic Logic Unit
* Output is function of a, b and c
  * +, -, AND, OR, XOR, SHIFT
* Flags - boolean conditions
  * C (carry)
  * Z (zero)

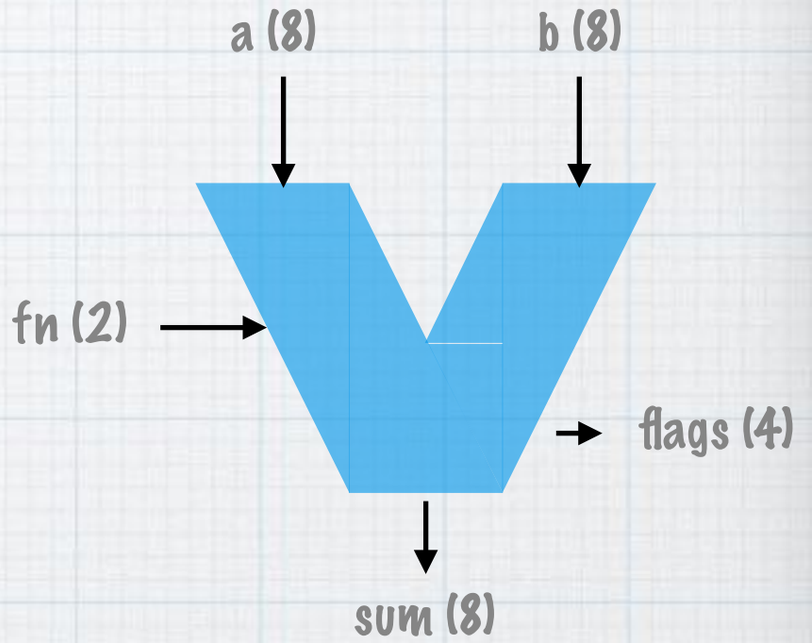


Simplified ALU

fn	name	sum
2'b00	AND	a AND b
2'b01	OR	a OR b
2'b10	XOR	a XOR b
2'b11	PASSB	b



```
module logic (
    input        [1:0]   fn,
    input        [7:0]   a,
    input        [7:0]   b,
    output reg   [7:0]   sum,
    output               s, v, n, z
    );

    always @*
        case (fn)
            default: sum = 8'b0;
        endcase

    // Flag logic, leave this bit alone
    assign s = sum[7];
    assign v = 1'b0;
    assign n = sum[7];
    assign z = ~| sum;

endmodule
```



Verilog

* Verilog 2001 standard
* C like
* Case-sensitive
  * DON'T use case alone to separate signals
* Less verbose than VHDL
* Vivado has free Verilog Simulation & Synthesis

Verilog Logic Values

* Four valued logic system - 0, 1, X and Z
* Subtle Simulation v Synthesis differences

Value	Simulation	Synthesis
0	0	0
1	1	1
X	undefined	n/a
Z	tri-state (out) Z = X (input)	tri-state (out)

Basic Logic Equations

0	1
1	0
X	X

Not

	0	1	X
0	0	0	0
1	0	1	X
X	0	X	X

And

	0	1	X
0	0	1	X
1	1	1	1
X	X	1	X

Or

	0	1	X
0	0	1	X
1	1	0	X
X	X	X	X

Xor

Module

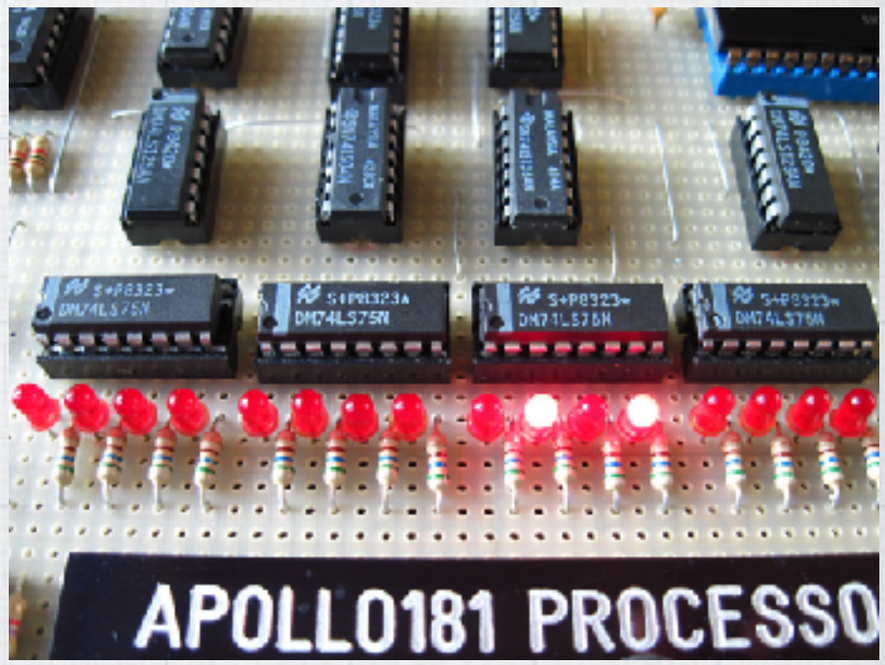
* Describes 'components'
  * Basic gates - NOT, AND, OR
  * System components - Microblaze, DDR3 memory controller

```
module logic (
    input         [1:0]   fn,
    input         [7:0]   a,
    input         [7:0]   b,
    output reg    [7:0]   sum,
    output                s, v, n, z
    );

    // module_items
```

Instantiation

* 'Creates' a module instance
  * 'wires' a component into the circuit
  * Same as Object Oriented program

```
logic u1 (
    .fn(funk),
    .a(a), .b(b), .sum(out1),
    .s(s), .v(v), .n(n), .z(z)
    );


logic u2 (
    .fn(funky),
    .a(a), .b(b), .sum(out2)
    );
```

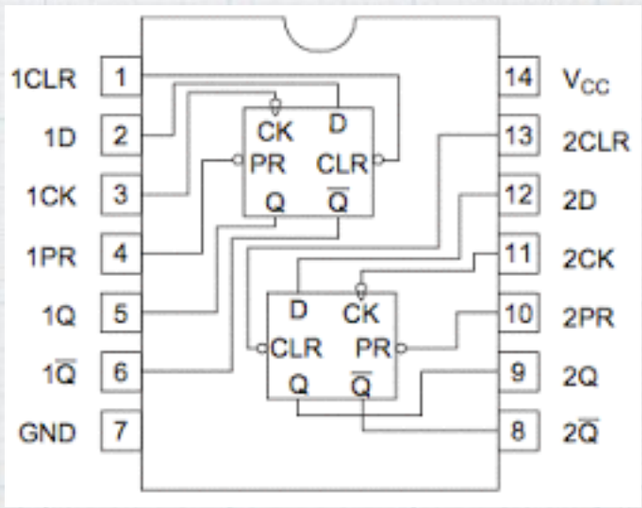


Ports

* Define connections to a module
  * Direction - input, output and inout

* Data type, range and name

```
module logic (
    input          [1:0]    fn,
    input          [7:0]    a,
    input          [7:0]    b,
    output reg     [7:0]    sum,
    output                  s, v, n, z
    );
```



Data Type - Wire

* Network data type
  * Connects modules together
  * Has no 'state'
* The default type
* Multiple outputs can be wired together
  * Resolution function
* Used with assign

Data Type - Reg

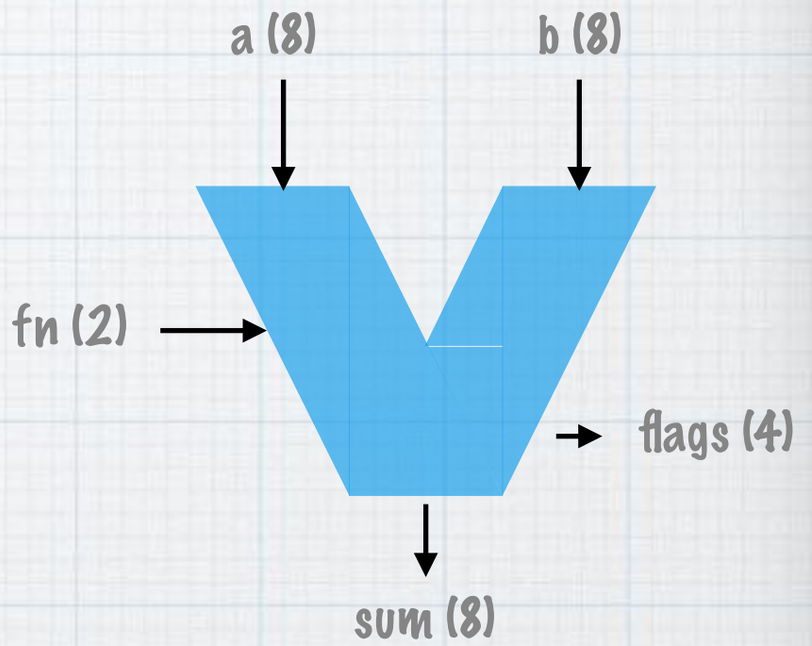
* Variable data type
* Holds state
* Used in procedural blocks
  * initial
  * always

Scalars and Vectors

* Scalar - single bit
* Vector - multiple bits
    * Specified by range [MSB : LSB]
        * MSB (leftmost), LSB (rightmost)

```
wire          a;          // scalar
reg           b, c;

wire [7:0]    x;          // vector
wire [15:8]   y;
reg  [0:7]    z;

       x[7]               // scalar or vector?
```

```
module logic(
    input        [1:0]   fn,
    input        [7:0]   a,
    input        [7:0]   b,
    output reg   [7:0]   sum,
    output               s, v, n, z
    );

    always @*
        case (fn)
            default: sum = 8'b0;
        endcase

    // Flag logic, leave this bit alone
    assign s = sum[7];
    assign v = 1'b0;
    assign n = sum[7];
    assign z = ~| sum;

endmodule
```



Literals

* Hardcoded constant values
  * <value> - unsigned decimal number (32)
  * <size> ' <s> <base> <value>
* Common bases - b/B, d/D, h/H

```
42;              // 32 bits, hex 0000002A
8'b00000000; // 8 bits, binary 00000000
4'D5;            // 4 bits, binary 0101
```

Continuous Assignment

* assign keyword
* LHS data type must be wire
* RHS is expression of wire/reg

```
assign s = sum[7];
assign v = 1'b0;
assign n = sum[7];
assign z = ~| sum;
```

Procedures

* 'Wrappers' for procedural statements
  * =, case, if-then-else
* Two kinds
  * initial
  * always

initial

* Starts at simulation time 0
* Enclosed statements execute once
* Testbenches

```
initial begin
    errors = 0;
    $display($time, " << Starting simulation >>");
    #10;

    for (test = 0; test < TESTS; test = test + 1) begin
```

always

* Never ends
* Normally has a sensitivity list

```
always @( ) begin
    …
end

// Combinational logic
@*                    // Wildcard, use this!
@(a or b)             // Verilog 1995 style
@(a, b)               // Verilog 2001 style

// Sequential logic
@(posedge clk)
```

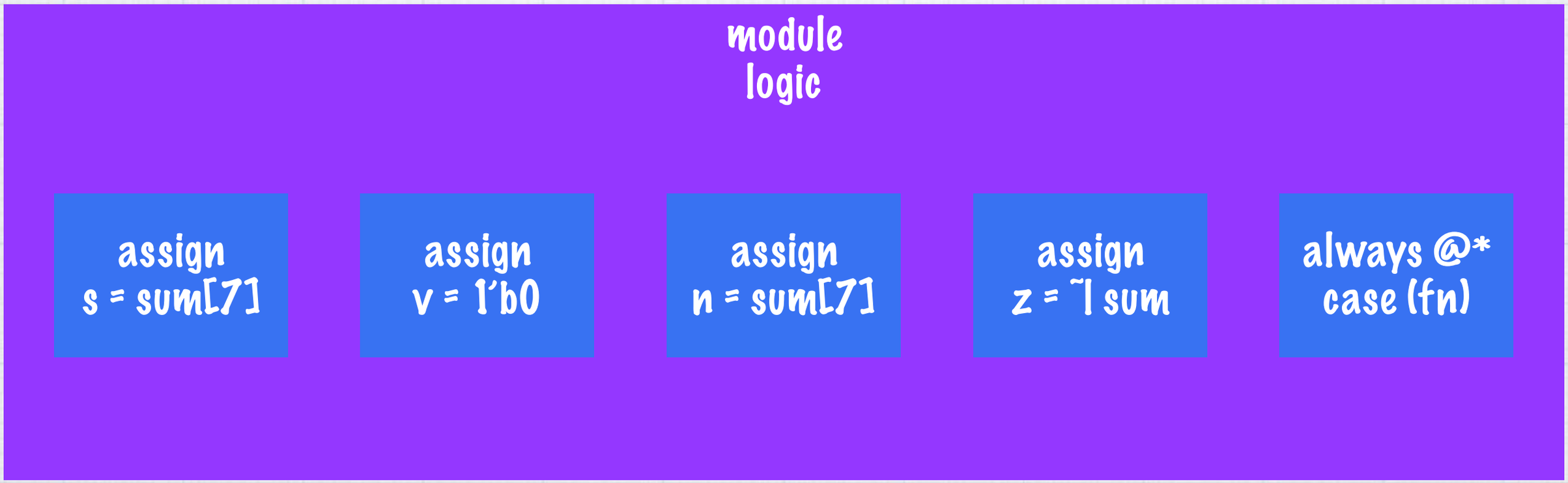
case Statement

```
case ( expression )
  case_item:                  statement or statement_group
  case_item, case_item:       statement or statement_group
  default:                    statement or statement_group
endcase
```

```
wire [7:0] a, [7:0] b;
wire [1:0] fn;
reg  [7:0] sum;
…
case (fn)
  2'b01:   sum = a | b;
  2'b10:   sum = a ^ b;
  default: sum = a & b;
endcase
```

Parallelism

* Procedures and continuous assignments operate in parallel



Bitwise Operators

* Binary - <value> OP <value>

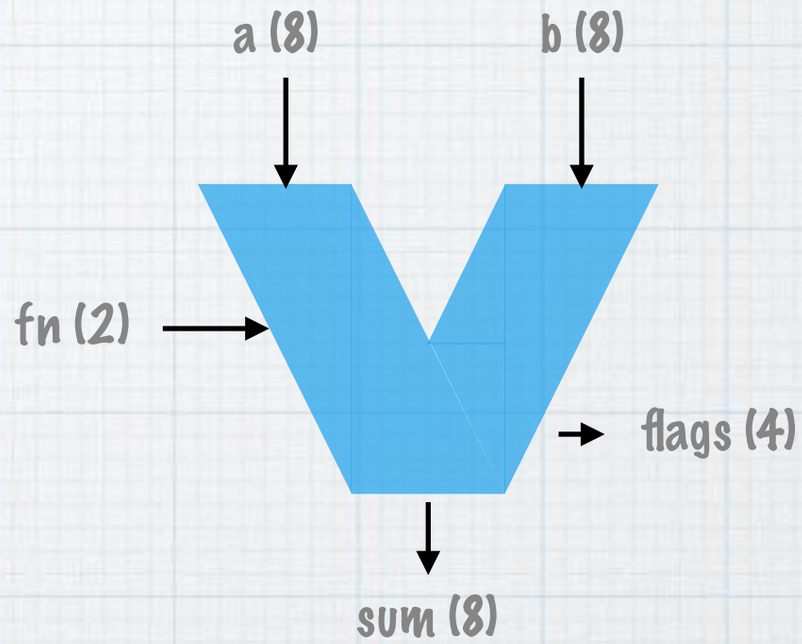
```
a & b               // bitwise AND
a | b               // bitwise OR
a ^ b               // bitwise XOR
```

Lab 03 - ALU Logic

* Use code template, extending CASE for
  * AND
  * OR
  * XOR
  * PASSB
* Run behavioural simulation
  * Make all tests work
* Don't over think it, 4 lines is sufficient!

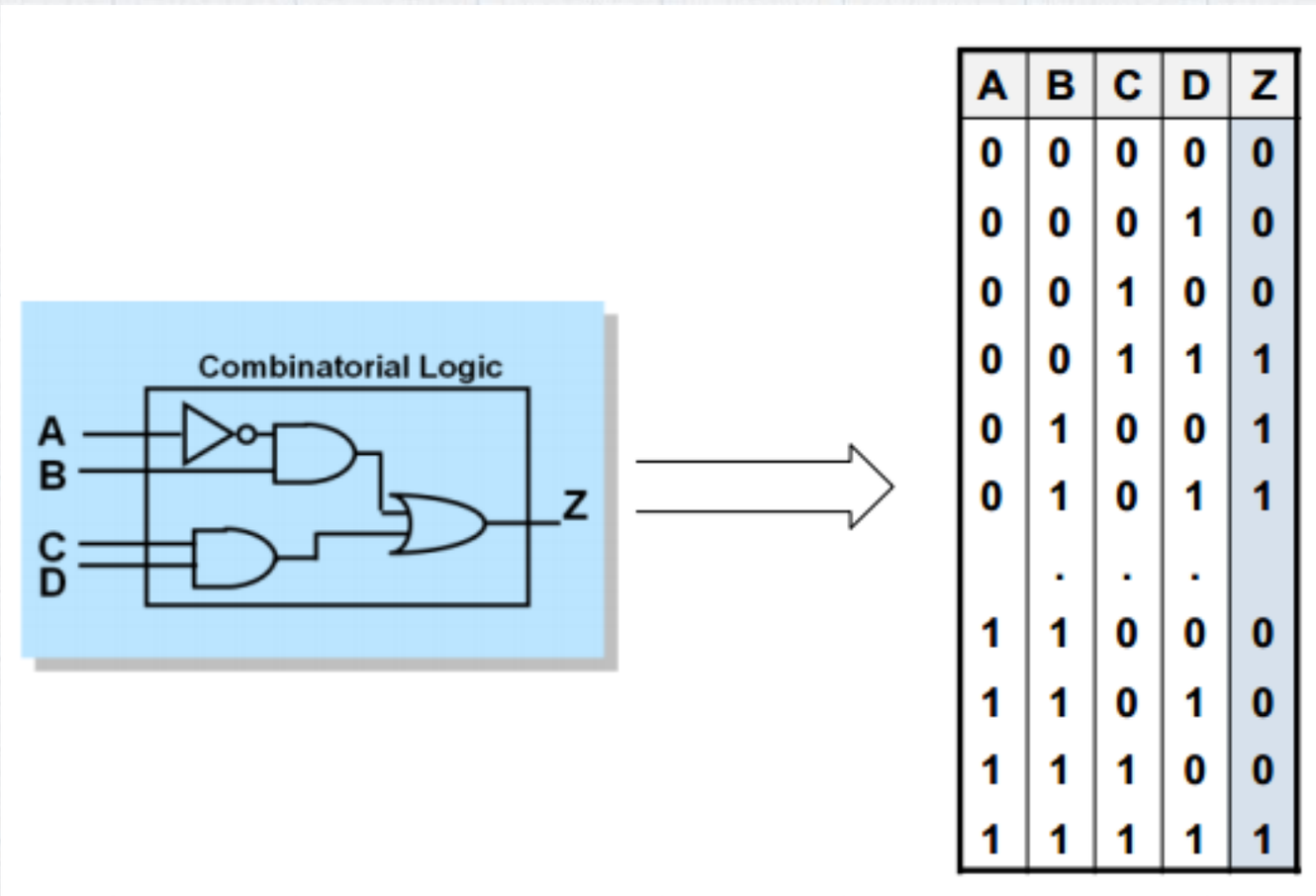
ALU Logic Module

fn	name	sum
2'b00	AND	a & b
2'b01	OR	a | b
2'b10	XOR	a ^ b
2'b11	PASSB	b



Lookup Table (LUT)

* Fundamental to FPGA's
  * Small Static RAM (SRAM) cell
  * Can implement ANY logic function



A	B	C	D	Z
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
	.	.	.	
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Lookup Table

* LUT sizes vary
  * Xilinx Series 7 - LUT-6
  * Others generally LUT-4
* Complicates vendor comparisons
  * Apples v Oranges

Synthesised Design

13	12	11	10	0
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0

13	12	11	10	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

Testbenches

* Verilog code
  * Simulation code style
    * 1 to 1 with module
      * logic.v
      * logic_tb.v

New Syntax

* Arrays
* Concatenation
* Symbolic values
* System functions and tasks
  * $display, $time
* Delays - #1, #9

Verilog Headers

* C preprocessor equivalent
* .vh suffix
* Include globally

```
// logic
`define ALUFN_AND       2'b00
`define ALUFN_OR        2'b01
`define ALUFN_XOR       2'b10
`define ALUFN_PASSB     2'b11
```

Using Headers

* Compiler performs 'cut & paste'

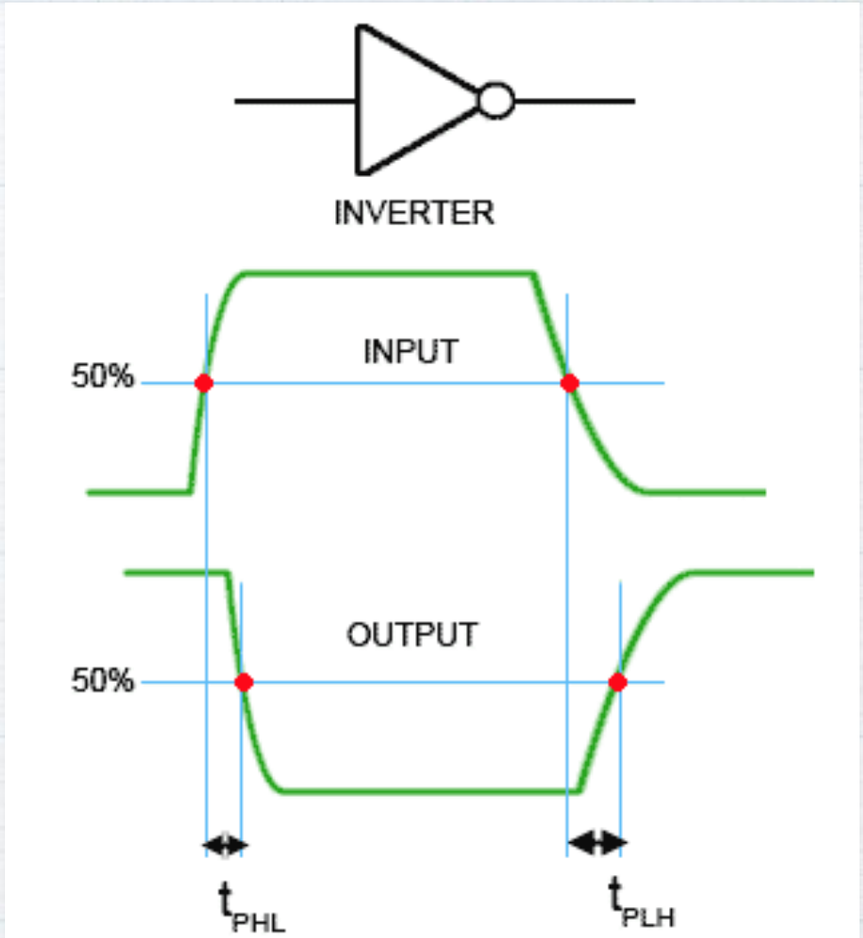
```
`timescale 1 ns / 1 ps
`include "defines.vh"

…

case (fn)
   `ALUFN_AND:   sum = …;
   `ALUFN_OR:    sum = …;
   `ALUFN_XOR:   sum = …;
   default:      sum = …;
endcase
```

Delays

* Simulation code only
* #N - N cycle delay
* Units taken from module
  * `timescale 1 ns / 100 ps
* #1.2 = 1.2 ns delay

Propagation Delay

* All circuitry has delays
  * LUT's
  * Signal routing (wires)



Simulation Types

* Behavioural
  * 'Perfect world', no propagation delays
* Post-Synthesis Timing
  * Nominal delays
* Post-Implementation Timing
  * Most accurate, logic and signal path delays

Lab 04 - Symbolic Values

* Replace binary literals
* Use symbolic names

Newb Erros #1

* Reg vs Wire
* VERY confusing
    * Try to Google answer
* Error messages like:
    * concurrent assignment to a non-net s is not permitted

```
module logic(
    input        [1:0]   fn,
    input        [7:0]   a,
    input        [7:0]   b,
    output reg   [7:0]   sum,
    output               s, v, n, z
    );

    always @*
        case (fn)
            default: sum = 8'b0;
        endcase

    // Flag logic, leave this bit alone
    assign s = sum[7];
    assign v = 1'b0;
    assign n = sum[7];
    assign z = ~| sum;

endmodule
```

Newb Erros #2

* Procedural 'weirdness'
  * if / case statements do strange things
  * Inputs change, outputs do not
* Inferred latches
  * Incorrect sensitivity list
  * Verilog 1995 and older is prone

```
// Verilog 2001 - always combinational
// Compiler auto calculates sensitivity list
always @*
    case (fn)
        `ALUFN_AND:   sum = a & b;
        `ALUFN_OR:    sum = a | b;
        `ALUFN_XOR:   sum = a ^ b;
        default:      sum = b;
    endcase
```

```
// Verilog 2001 - maybe combinational (depends)
// Manual sensitivity list
always @(fn, a, b)
```

```
// Verilog 1995 - maybe combinational (depends)
// Manual sensitivity list
always @(fn or a or b)
```

```
// Verilog 1995 - sequential
// Manual sensitivity list
always @(posedge clock, negedge reset)
```

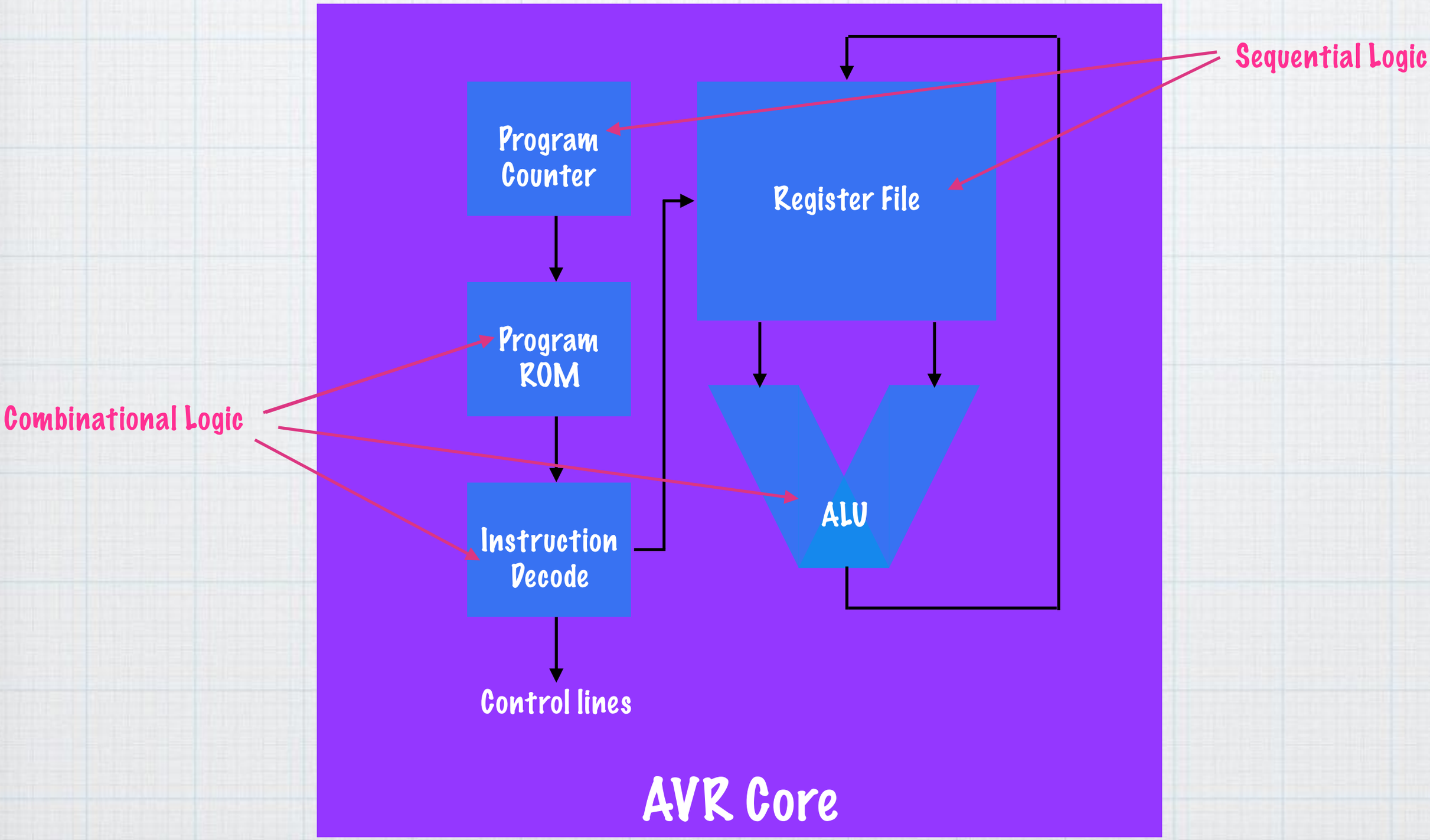
```
// Error!
// sum value only changes when fn value changes

always @(fn)
    case (fn)
        `ALUFN_AND:   sum = a & b;
        `ALUFN_OR:    sum = a | b;
        `ALUFN_XOR:   sum = a ^ b;
        default:      sum = b;
    endcase
```

```
// For combinational Verilog 2001

always @*
```

Lab 05 - Common Errors

* Make deliberate errors
* Observe Vivado behaviour

Stage 1 Roadmap



Program ROM

```
module prog1(
    input        [3:0]   a,       // address
    output reg   [15:0]  dout     // instruction
    );

    always @*
        case (a)
            4'h0:    dout = 16'hea05;  // ldi
            4'h1:    dout = 16'he01f;  // ldi
            4'h2:    dout = 16'h2301;  // and
            default: dout = 16'h0000;  // nop
        endcase

endmodule
```