# FPGAs: Verilog Sequence Alignment (maybe)

Chris Rossbach cs378 Fall 2018

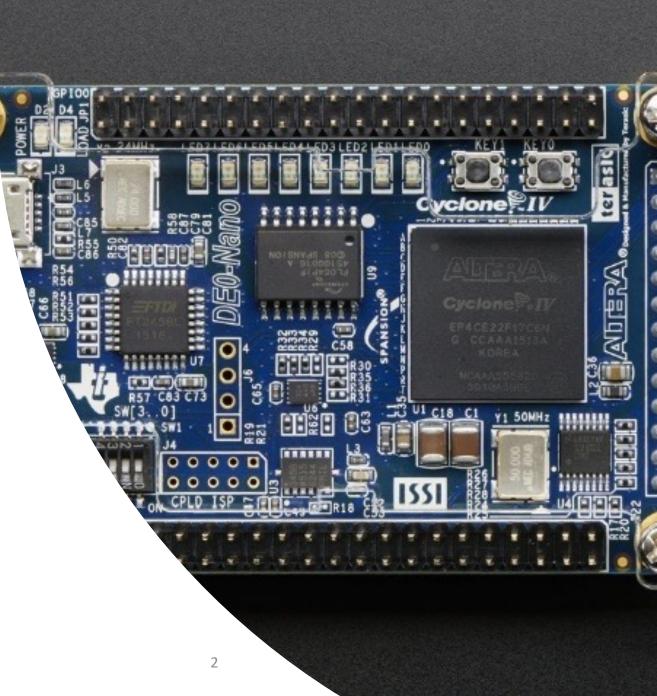
11/5/2018

# Outline for Today

- Questions?
- Administrivia
  - Re: Exams
  - Keep thinking about projects!
  - Website updates
- Agenda
  - FPGAs: POTPOURRI of things you need to know
  - NW

#### Acknowledgements/References:

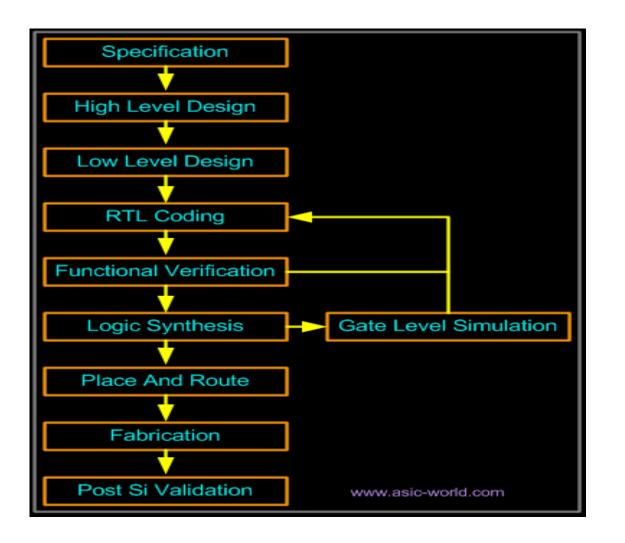
- https://s3-us-west-2.amazonaws.com/cse291personalgenomics/Lectures2017/Lecture12\_AlignmentVariantCalling.pptx
- <a href="https://web.stanford.edu/~jurafsky/slp3/slides/2\_EditDistance.ppt">https://web.stanford.edu/~jurafsky/slp3/slides/2\_EditDistance.ppt</a>
- https://moodle.med.lu.se/pluginfile.php/45044/mod\_resource/content/0/sequence\_alignment\_2015.pptx
- <u>http://www.cbs.dtu.dk/phdcourse/cookbooks/PairwiseAlignmentPhD2.ppt</u>
- <u>http://cwcserv.ucsd.edu/~billin/classes/ECE111/lectures/Lecture1.pptx</u>
- <u>http://www.cs.unc.edu/~montek/teaching/Comp541-Fall16/VerilogPrimer.pptx</u>
- Evita\_verilog Tutorial, <u>www.aldec.com</u>
- http://www.asic-world.com/verilog/



#### Faux Quiz Questions

- Why/when might one prefer an FPGA over an ASIC, CPU, or GPU?
- Define CLB, BRAM, and LUT. What role do these things play in FPGA programming?
- What is the difference between blocking and non-blocking assignment in Verilog?
- What is the difference between structural and behavioral modeling?
- How is synthesizable Verilog different from un-synthesizable? Give an example of each?
- What is discrete event simulation?

#### Review: FPGA Design/Build Cycle



- HW design in Verilog/VHDL
- Behavioral modeling + some structural elements
- Simulate to check functionality
- Synthesis  $\rightarrow$  netlist generated
- Static analysis to check timing

# Verilog

- Originally: modeling language for event-driven digital logic simulator
- Later: specification language for logic synthesis
- Consequence:
  - Combines structural and behavioral modeling styles

# Components of Verilog

- Concurrent, event-triggered processes (behavioral)
  - Initial and Always blocks
  - Imperative code  $\rightarrow$  standard data manipulation (assign, if-then, case)
  - Processes run until triggering event (or #delay expire)
- Structure
  - Verilog program builds from modules with I/O interfaces
  - Modules may contain instances of other modules
  - Modules contain local signals, etc.
  - Module configuration is static and all run concurrently

#### **Discrete-event Simulation**

- Key idea: *only* do work when something changes
- Core data structure: *event queue* 
  - Contains events labeled with the target simulated time
- Algorithmic idea:
  - Execute every event for current simulated time
  - May change system state and may schedule events in the future (or now)
  - No events left at current time  $\rightarrow$  advance simulated time (next event in Q)

# Two Main Data Types

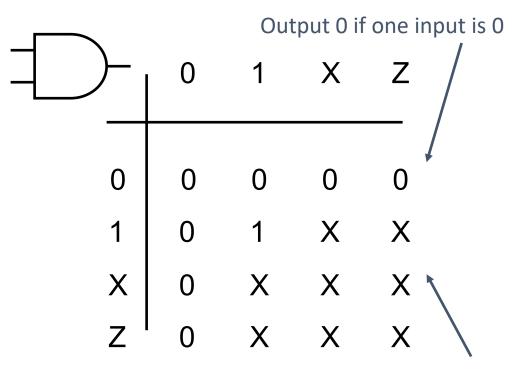
- Nets represent connections between things
  - Do not hold their value
  - Take their value from a driver such as a gate or other module
  - Cannot be assigned in an *initial* or *always* block
- Regs represent data storage
  - Behave exactly like memory in a computer
  - Hold their value until explicitly assigned in an *initial* or *always* block
  - Model latches, flip-flops, etc., but do not correspond exactly
  - Shared variables
    - Similar known shared state issues

# Four-valued Data and Logic

Nets and regs hold four-valued data

- 0, 1 → Umm...
- Z
- Output for undriven tri-state (hi-Z)
- Nothing is setting a wire's value
- X
- Simulator can't decide the value
- Initial state of registers
- Wire driven to 0 and 1 simultaneously
- Output of gate with Z inputs
- Data representation
  - Binary  $\rightarrow$  6'b100101
  - Hex  $\rightarrow$  6'h25

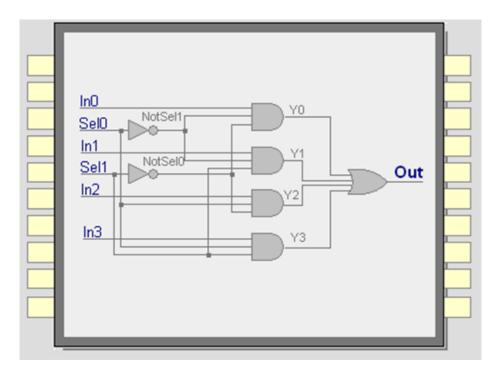
• Logical operators work on threevalued logic

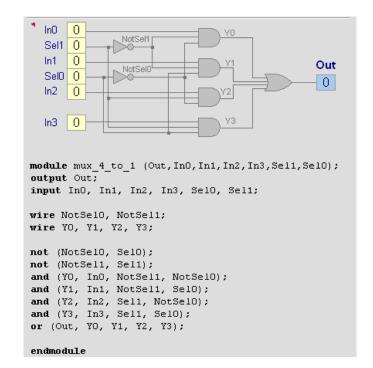


#### Output X if inputs are junk

# Structural Modeling

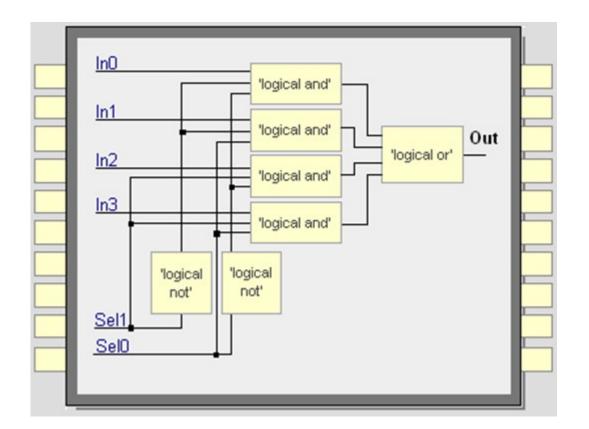
- Specification
  - Netlist: gates and connections
  - Primitives/components (e.g logic gates)
  - Connected by wires
- Easy to translate to physical circuit

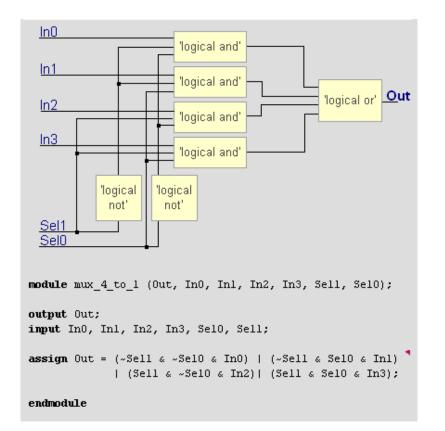




#### Dataflow Modeling

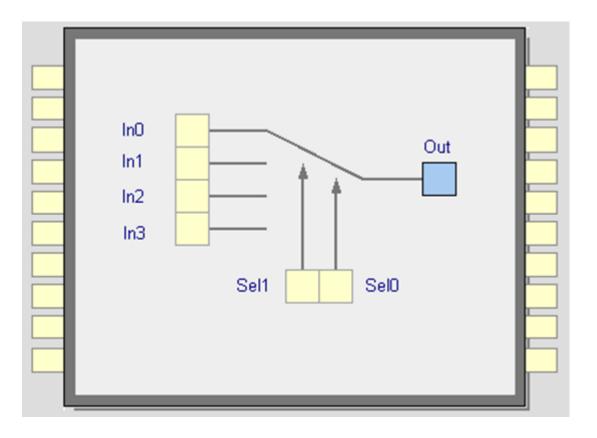
- Specification
  - Components (similar to logical equations)
  - Connected by wires
- Easy to translate to structure, then to physical circuit



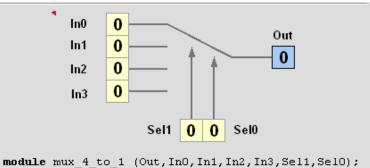


#### **Behavioral Modeling**

- Specification
  - In terms of expected behavior
  - Closest to natural language
- Most difficult to synthesize



- Easier for testbenches
- Easier for abstract models of circuits
  - Simulates faster
- Provides sequencing



module mux\_4\_to\_1 (Out, InO, In1, In2, In3, Sel1, Sel0).
output Out;
input InO, In1, In2, In3, Sel0, Sel1;
reg Out;

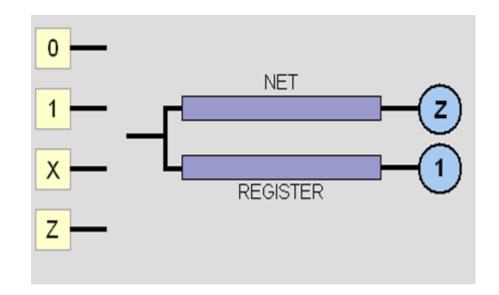
always @(Sel1 or Sel0 or In0 or In1 or In2 or In3)
begin
 case ((Sel1, Sel0))
 2'b00 : Out = In0;
 2'b01 : Out = In1;
 2'b10 : Out = In2;
 2'b11 : Out = In3;
 default : Out = 1'bx;
 endcase

endmodule

end

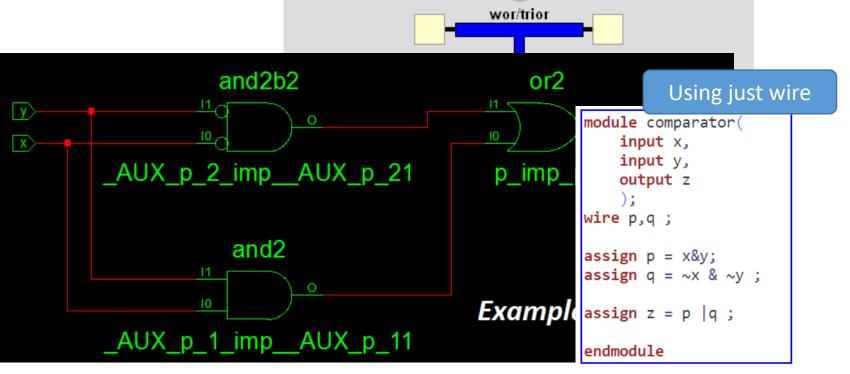
# Signals

- Nets
  - Physical connection between hardware elements
- Registers
  - Store value even if disconnected



#### Nets

- wire/tri
- wand/triand
- wor/trior
- Force synthesis to insert gates
  - (e.g. AND, OR)

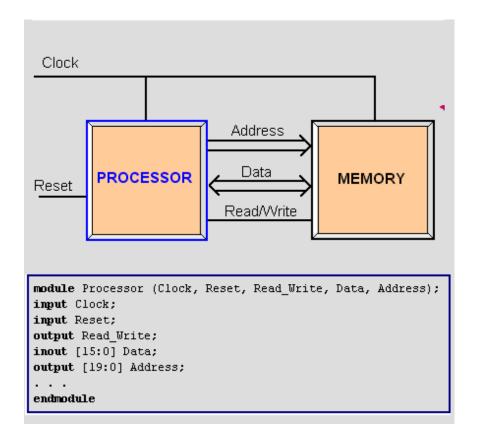


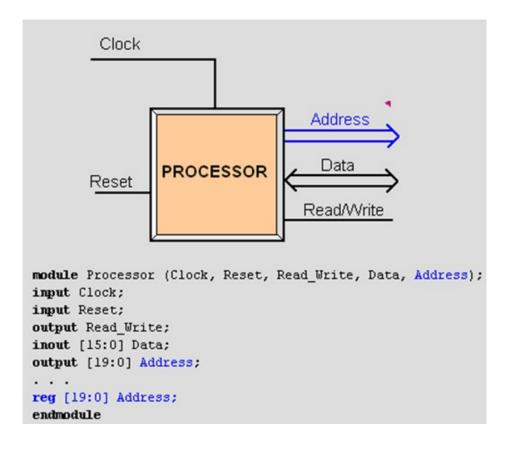
wire/tri

wand/triand

module comparatorwithwor(
 input x,
 input y,
 output z
 );
wor p ;
assign p = x&y;
assign p = ~x & ~y ;
assign z = p ;
endmodule

#### Ports and Registered Output





Output ports can be type register

- Add reg type to declaration
- Output holds state

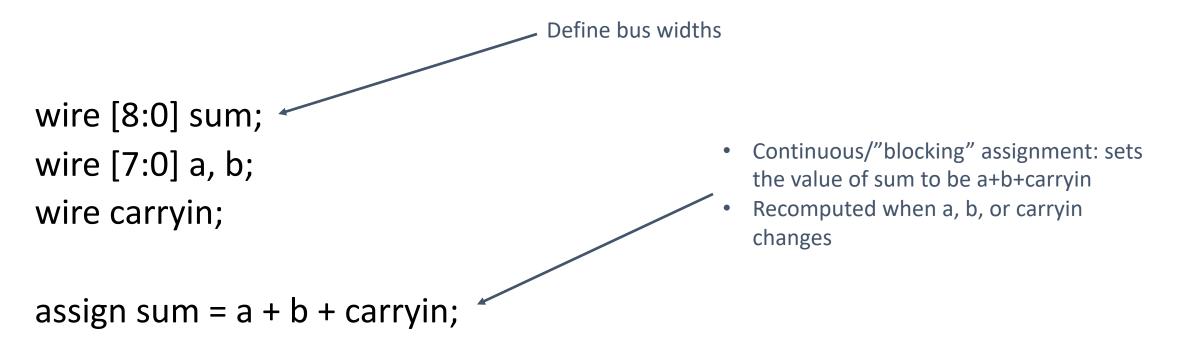
#### Examples of Nets and Registers

Wires and registers can be bits, vectors, and arrays

wire a; // Simple wire
tri [15:0] dbus; // 16-bit tristate bus
tri #(5,4,8) b; // Wire with delay
reg [-1:4] vec; // Six-bit register
trireg (small) q; // Wire stores a small charge
integer imem[0:1023]; // Array of 1024 integers
reg [31:0] dcache[0:63]; // A 32-bit memory

#### Continuous Assignment

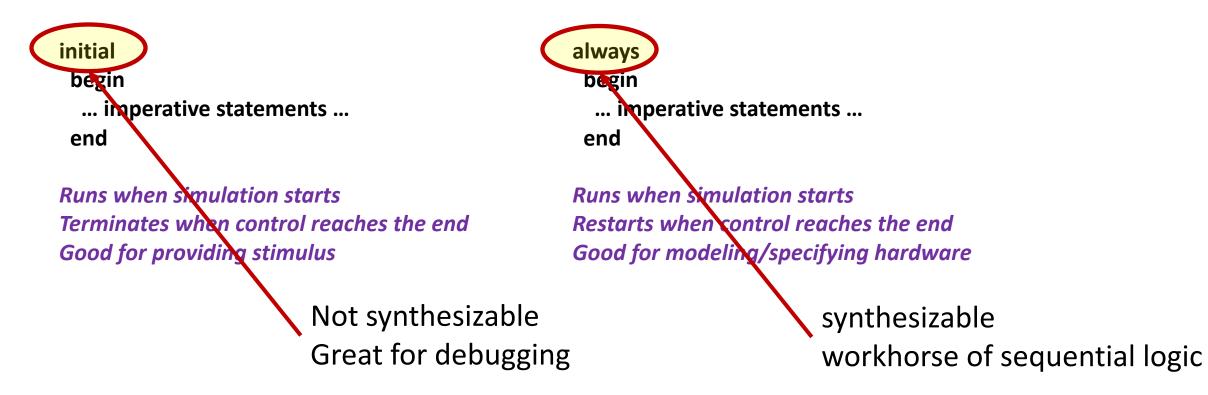
- Another way to describe combinational function
- Convenient for logical or datapath specifications



# **Behavioral Modeling**

#### Initial and Always Blocks

Basic components for behavioral modeling



#### Initial and Always

• Run until they encounter a delay

```
initial begin
#10 a = 1; b = 0;
#10 a = 0; b = 1;
end
```

• or a wait for an event

always @(posedge clk) q = d; always begin wait(i); a = 0; wait(~i); a = 1; end

## Procedural Assignment

• Inside an initial or always block:

sum = a + b + cin;

- Just like in C:
  - RHS evaluated
  - assigned to LHS
  - before next statement executes
- RHS may contain wires and regs
  - Two possible sources for data
- LHS must be a reg
  - Primitives or cont. assignment may set wire values

#### Imperative Statements

case (op)
2'b00: y = a + b;
2'b01: y = a - b;
2'b10: y = a ^ b;
default: y = 'hxxxx;
endcase

#### For and While Loops

Increasing sequence of values on an output

reg [3:0] i, output;

for ( i = 0 ; i <= 15 ; i = i + 1 ) begin
 output = i;
 #10;
end</pre>

reg [3:0] i, output;

i = 0; while (I <= 15) begin output = i; #10 i = i + 1; end

#### A Flip-Flop With Always

Edge-sensitive flip-flop

reg q;

always @(posedge clk) q = d;

- q = d assignment
  - runs when clock rises
  - exactly the behavior you expect

# Blocking vs. Nonblocking

- Verilog has two types of procedural assignment
- Fundamental problem:
  - In a synchronous system, all flip-flops sample simultaneously
  - In Verilog, always @(posedge clk) blocks run in some undefined sequence

#### A Shift Register aka Blocking vs Non-blocking assignment

reg d1, d2, d3, d4;

always @(posedge clk) d2 = d1; "Blocking assignment" always @(posedge clk) d3 = d2; always @(posedge clk) d4 = d3;

- These run in some order, but you don't know which
- So...*might* not work as you'd expect

#### Non-blocking Assignments

reg d1, d2, d3, d4;

always @(posedge clk) d2 <= d1; always @(posedge clk) d3 <= d2; always @(posedge clk) d4 <= d3; Nonblocking rule: RHS evaluated when assignment runs

- Blocking vs. Non-blocking: misnomer
- prefer "continuous" to "blocking"
- Guideline: blocking for combinational
- Guideline: non-blocking for sequential

LHS updated only after all events for the current instant have run

#### Non-blocking Behavior

• A sequence of nonblocking assignments don't communicate

a = 1;	a <= 1;
b = a;	b <= a;
c = b;	c <= b;

Blocking assignment: a = b = c = 1 Nonblocking assignment: a = 1 b = old value of a c = old value of b

# Dirty/tricky question: which assignment type yields a correct shift register?

reg d1, d2, d3, d4;

always @(posedge clk) begin d2 *op* d1; d3 *op* d2; d4 *op* d3; end

Should **op** be = or <= ?

#### Implementation: Building FSMs

- Many ways to do it
- Define the next-state logic combinationally
  - define the state-holding latches explicitly
- Define the behavior in a single always @(posedge clk) block
- Define behavior per signal in different @(posedge clk) blocks
- Variations on these themes

#### FSM with Combinational Logic

```
module FSM(o, a, b, reset);
output o;
reg o;
input a, b, reset;
reg [1:0] state, nextState;
always @(a or b or state)
case (state)
  2'b00: begin
    nextState = a ? 2'b00 : 2'b01;
   o = a \& b;
  end
  2'b01: begin nextState = 2'b10; o = 0; end
endcase
```

Combinational block must be sensitive to any change on any of its inputs (Implies state-holding elements otherwise)

#### FSM with Combinational Logic

```
module FSM(o, a, b, reset);
...
always @(posedge clk or reset)
                                                         Latch implied by sensitivity
                                                        to the clock or reset only
 if (reset)
  state <= 2'b00;
 else
  state <= nextState;</pre>
```

#### FSM from Combinational Logic

```
always @(a or b or state)
case (state)
2'b00: begin
nextState = a ? 2'b00 : 2'b01;
o = a & b;
end
2'b01: begin nextState = 2'b10; o = 0; end
endcase
```

```
always @(posedge clk or reset)
if (reset)
state <= 2'b00;
else
state <= nextState;</pre>
```

#### FSM with a Single Always Block

Outputs are latched module FSM(o, a, b); Inputs only sampled at clock output o; reg o; edges input a, b; reg [1:0] state; Nonblocking assignments always @(posedge clk or reset) used throughout. if (reset) state  $\leq 2'b00$ ; **RHS** refers to values else case (state) calculated in previous clock 2'b00: begin cycle state <= a ? 2'b00 : 2'b01; o <= a & b; end 2'b01: begin state <= 2'b10; o <= 0; end endcase

#### Parameters

localparam keyword

localparam state1 = 4'b0001,
 state2 = 4'b0010,
 state3 = 4'b0100,
 state4 = 4'b1000;

```
localparam A = 2'b00,
G = 2'b01,
C = 2'b10,
T = 4'b11;
```

# Operations for HDL simulation/build

• Compilation/Parsing

#### • Elaboration

- Binding modules to instances
- Build hierarchy
- Compute parameter values
- Resolve hierarchical names
- Establish net connectivity
- ...(simulate, place/route, etc)

# Generate Block

- Dynamically generate Verilog code at *elaboration* time
  - Usage:
    - Parameterize modules when the parameter value determines the module contents
  - Can generate
    - Modules
    - User defined primitives
    - Verilog gate primitives
    - Continuous assignments
    - initial and always blocks

## Generate Loop

```
module bitwise_xor (output [N-1:0] out, input [N-1:0] i0, i1);
parameter N = 32; // 32-bit bus by default
genvar j; // This variable does not exist during simulation
```

```
generate for (j=0; j<N; j=j+1) begin: xor_loop
//Generate the bit-wise Xor with a single loop</pre>
```

```
xor g1 (out[j], i0[j], i1[j]);
```

```
end
```

```
endgenerate //end of the generate block
```

```
/* An alternate style using always blocks:
```

```
reg [N-1:0] out;
```

```
generate for (j=0; j<N; j=j+1) begin: bit</pre>
```

```
always @(i0[j] or i1[j]) out[j] = i0[j] ^ i1[j];
```

end

#### endgenerate

endmodule \*/

Can do this with code but requires different numbers of xor modules depending on N

## Generate Conditional

module multiplier (output [product\_width -1:0] product, input [a0\_width-1:0] a0, input [a1\_width-1:0] a1);

parameter	a0_width = 8;
parameter	a1_width = 8;
localparam	product_width = a0_width + a1_width;
generate	
<b>if</b> (a0_wic	lth <8)    (a1_width < 8)
	cla_multiplier #(a0_width, a1_width) m0 (product, a0, a1);
else	
	tree_multiplier #(a0_width, a1_width) m0 (product, a0, a1);
endgenerate	

endmodule

## Generate Case

module adder(output co, output [N-1:0] sum, input [N-1:0] a0, a1, input ci);

```
parameter N = 4;
```

// Parameter N that can be redefined at instantiation time.

```
generate
```

```
case (N)
```

1:	adder_1bit	adder1(c0, sum, a0, a1, ci);
2:	adder_2bit	adder2(c0, sum, a0, a1, ci);
default:	adder_cla #(N)	adder3(c0, sum, a0, a1, ci);

endcase

endgenerate

endmodule

# Nesting

- Generate blocks can be nested
  - Nested loops cannot use the same genvar variable

```
8 //
9 // Change history: 8/23/18 - Initial revision
10 //
12
   include nwcell.v;
13
14
15
   module nwgrid #(parameter N=8) (clk, reset, enable);
16
17
     input wire clk;
     input wire reset;
18
19
     input wire enable;
20
21
     genvar
               i;
22
               j;
     genvar
     for (i=0; i<N; i=i+1) begin : X</pre>
23
24
        for (j=0; j<N; j=j+1) begin : Y</pre>
25
26
       wire scout v;
       wire [N-1:0] scout;
27
28
       wire [1:0] backpath;
29
30
       if(i==0 && j==0) begin
```

# Logic Synthesis

- Verilog: two use-cases
  - Model for discrete-event simulation
  - Specification for a logic synthesis system
- Logic synthesis: convert subset of Verilog language  $\rightarrow$  netlist

#### Two stages

- 1. Translate source to a netlist
  - Register inference
- 2. Optimize netlist for speed and area
  - Most critical part of the process
  - Awesome algorithms

# What Can/Can't Be Translated

- Structural definitions
  - Everything
- Behavioral blocks
  - When they have reasonable interpretation as combinational logic, edge, or level-sensitive latches
- User-defined primitives
  - Primitives defined with truth tables
  - Some sequential UDPs can't be translated (not latches or flip-flops)

### Initial blocks

- Used to set up initial state or describe finite testbench stimuli
- Don't have obvious hardware component
- Delays
  - May be in the Verilog source, but are simply ignored
- Other obscure language features
  - In general, things dependent on discrete-event simulation semantics
  - Certain "disable" statements
  - Pure events

## FPGAs and Programming in Cascade

#### eric schkufza

November 7, 2018

Confidential | ©2018 VMware, Inc.

## Agenda

## **FPGAs**

What's so good about them? What's so bad about them?

#### Cascade

How we make the good stuff better, and the bad stuff less awful

### Live Demo

Writing a simple program in Cascade

### **Time Permitting**

How Cascade works? Verilog minutiae?

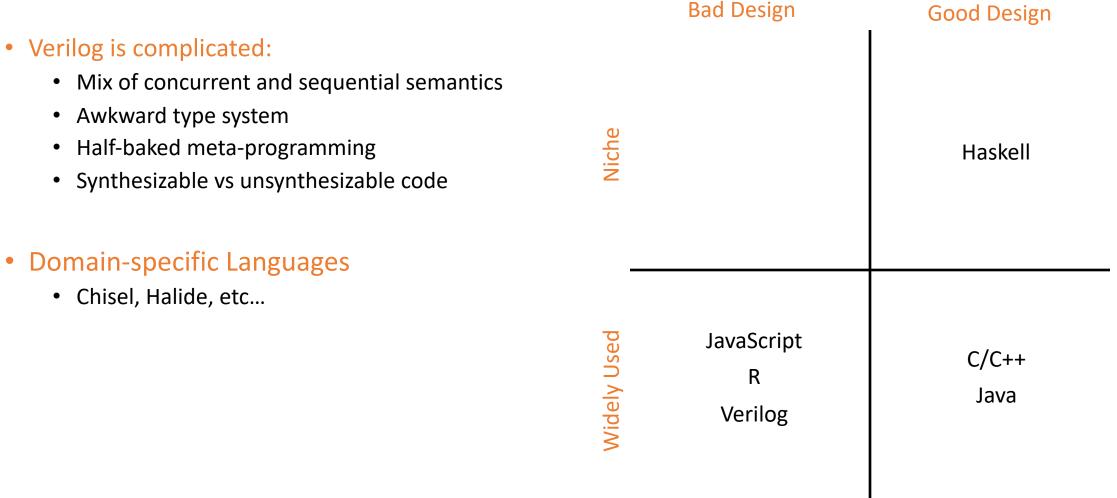
## What are FPGAs good for?



### • FPGAs make sense when:

- A workload is high-performance but also predictable
- Application requirements change relatively frequently

# But programming an FPGA is HARD!

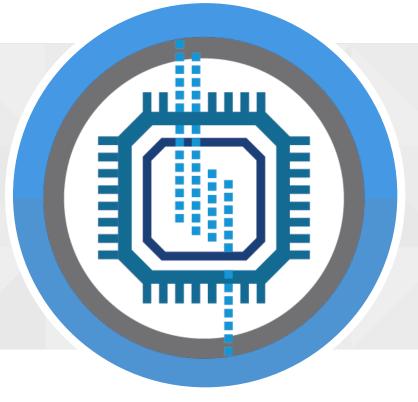


# And compilation takes FOREVER!

- Software Compilers:
  - O(seconds)
  - Reason about programs locally
  - Pre-defined O(n^k) rules
- Software Development:
  - Compile-test-debug cycle
  - Test and deploy in the same environment

- Hardware Compilers:
  - O(minutes to hours)
  - Reason about programs globally
  - NP-hard constraint satisfaction
- Hardware Development:
  - Debug behavior in a simulator
  - Debug timing in hardware
  - Test and deploy in different environments

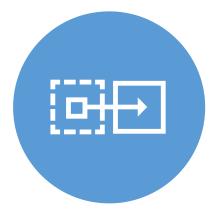
- CASCADE
- Makes programming hardware feel like programming software



# Design Goals









• Interactivity

Modify a running program, I/O side effects visible immediately • Expressiveness

Eliminate synthesizable vs non-synthesizable distinction

#### Portability

Write code once, run on many platforms with little modification

Performance
 Don't pay for
 features you don't

## Interactivity

#### • Just-in-Time Compilation

- Code runs immediately in a simulator
- Compilation takes place in the background
- Control switches when compilation is done
- Code appears to run faster over time

#### • Why can we do this?

- What's the meaning of a Verilog program?
- What's the meaning of any program?

## Interactivity

- Just-in-Time Compilation
  - Code runs immediately in a simulator
  - Compilation takes place in the background
  - Control switches when compilation is done
  - Code appears to run faster over time
- Why can we do this?
  - What's the meaning of a Verilog program?

- 1: **procedure** EVAL(*e*)
- 2: **if** *e* is an update **then**
- 3: perform sequential update
- 4: else
- 5: evaluate combinational logic
- 6: end if
- 7: enqueue new events
- 8: end procedure
- 1: procedure REFERENCESCHEDULER while  $\top$  do 2: if  $\exists$  activated events then 3: EVAL(any activated event) 4: else if  $\exists$  update events then 5: activate all update events 6: else 7: advance time t; schedule recurring events 8: end if 9: end while 10: 11: end procedure

## Expressiveness

- Unsynthesizable Verilog in hardware
  - Display statements
  - Finish statements
  - Longer term: support for the entire unsynthesizable language subset
- Why can we do this?
  - What is Cascade doing differently compared to a traditional compiler?

1:	module Main(
2:	input wire clk,
3:	<pre>input wire [3:0] pad_val,</pre>
4:	<pre>output wire [7:0] led_val,</pre>
5:	<pre>output wire [7:0] r_x,</pre>
6:	<pre>input wire [7:0] r_y</pre>
7:	);
8:	reg cnt [7:0] = 1;
9:	<pre>assign r_x = cnt;</pre>
10:	<b>always</b> @( <b>posedge</b> clk_val)
11:	<pre>if (pad_val == 0)</pre>
12:	cnt <= r_y;
13:	else
14:	<pre>\$display(cnt);</pre>
15:	\$finish;
16:	<pre>assign led_val = cnt;</pre>
17:	endmodule

## Expressiveness

```
1: module Main(
      input wire
 2:
                         CLK,
 3:
      input wire
                         RW,
      input wire [31:0] ADDR,
 4:
      input wire [31:0] IN,
 5:
      output wire [31:0] OUT,
 6:
      output wire
 7:
                         WAIT
 8:);
 9:
      reg [31:0] _vars[3:0];
      reg [31:0] _nvars[3:0];
10:
      reg _umask = 0, _numask = 0;
11:
12:
      reg [ 1:0] _tmask = 0, _ntmask = 0;
13:
      reg [31:0] _oloop = 0, _itrs = 0;
14:
15:
      wire clk val = vars[0];
      wire [3:0] pad_val = __vars[1];
16:
      wire [7:0] led_val;
17:
      wire [7:0] cnt = _vars[2];
18:
19:
20:
      always @ (posedge clk_val)
21:
       if (pad_val == 0)
22:
          _nvars[2] <= pad_val << 1;
23:
          _numask <= ~_umask;
24:
        else
          _nvars[3] <= cnt;
25:
          ntmask <= ~ tmask;</pre>
26:
27:
      assign led_val = cnt;
```

```
wire _updates = _umask ^ _numask;
28:
29:
      wire _latch = <LATCH>
30:
       ( updates & oloop);
     wire _tasks = _tmask ^ _ntmask;
31:
     wire _clear = <CLEAR>;
32:
     wire otick = oloop & ! tasks;
33:
34:
35:
      always @ (posedge CLK)
36:
       _umask <= _latch ? _numask : _umask;</pre>
       _tmask <= _clear ? _ntmask : _tmask;</pre>
37:
38:
       _oloop <= <OLOOP> ? IN :
39:
         _otick ? (_oloop-1) :
         _tasks ? 0 : _oloop;
40:
       _itrs <= <OLOOP> ? 0 :
41:
42:
          _otick ? (_itrs+1) : _itrs;
       _vars[0] <= _otick ? (_vars[0]+1) :
43:
          <SET 0> ? IN : _vars[0];
44:
45:
        _vars[1] <= <SET 1> ? IN : _vars[1];
46:
       _vars[2] <= <SET 2> ? IN :
          _latch ? _nvars[2] : _vars[2];
47:
48:
49:
     assign WAIT = _oloop;
     always @(*)
50:
51:
        case (ADDR)
52:
          0: OUT = clk val;
53:
         // cases omitted ...
54: endmodule
```

# Limitations and Future Work

- Non-Monotonic language features
  - Code deletion
  - Genvar statements
- Timing-sensitive applications
  - A giga-bit ethernet switch?
  - A peripheral which expects inputs on a perfectly periodic clock?

- FPGA Virtualization:
  - Share one FPGA between two instances of Cascade
  - Use Cascade to transparently run one very large program on two separate FPGAs

#### • Speculative Optimization:

- Specialize the implementation of a program to the values that it sees at runtime
- Generate smaller / faster code

# Thank You

Questions on Piazza

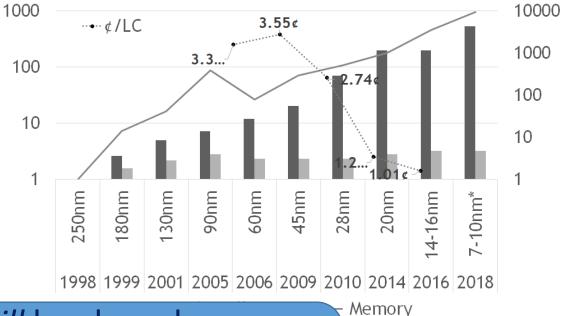
Bug Reports on https://github.com/vmware/cascade

Confidential | ©2018 VMware, Inc.

# AmorphOS Motivation

Bigger, faster FPGAs deployed in the cloud

- Microsoft Catapult/Azure
- Amazon F1
- FPGAs: Reconfigurable Accelerators
  - ASIC Prototyping, Video & Image Proc., DNN, Blockchain
  - Potential solution to accelerator provisioning challenge

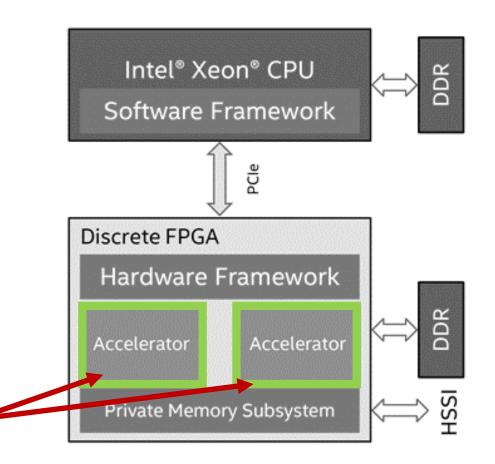


### Our position: FPGAs will be shared

- Sharing requires protection
- Abstraction layers provide compatibility
- Beneficiary: provider  $\rightarrow$  consolidation

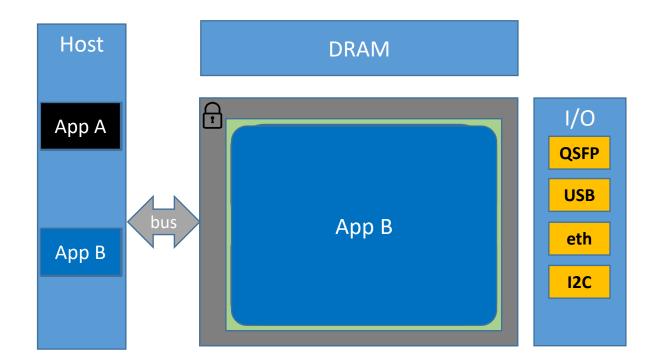
# FPGA Background

- Field Programmable Gate Array (FPGA)
  - Reconfigurable interconnect  $\rightarrow$  custom data paths
  - FPGAs attached as *coprocessors to a CPU*
- FPGA Build Cycle
  - Synthesis: HDL → Netlist (~seconds)
  - Place and Route: Netlist → Bitstream (~min-hours)
  - Reconfiguration/Partial Reconfiguration (PR)
- Production systems: No multi-tenancy
- Emerging/Research Systems use *fixed slots/PR*
  - Fixed-sized slots → fragmentation (50% or more)
  - Elastic resource management needed



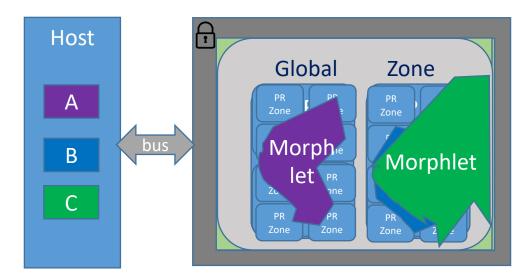
# AmorphOS Goals

- Protected Sharing/Isolation
  - Mutually distrustful applications
- Compatibility / Portability
  - HDL written to AmorphOS interfaces
  - 14 benchmarks run unchanged on Microsoft Catapult and Amazon F1
- Elasticity
  - User logic scales with resource availability
  - Sharing density scales with availability



# AmorphOS Abstractions

- Zone: Allocatable Unit of Fabric
  - 1 Global zone
  - N dynamically sized, sub-dividable PR zones
- Hull: OS/Protection Layer
  - Memory Protection, I/O Mediation
  - Interfaces form a compatibility layer
- Morphlet: Protection Domain
  - Extends Process abstraction
  - Encapsulate user logic on PR or global zone
- Registry: Bitstream Cache
  - Hides latency of place-and-route (PaR)

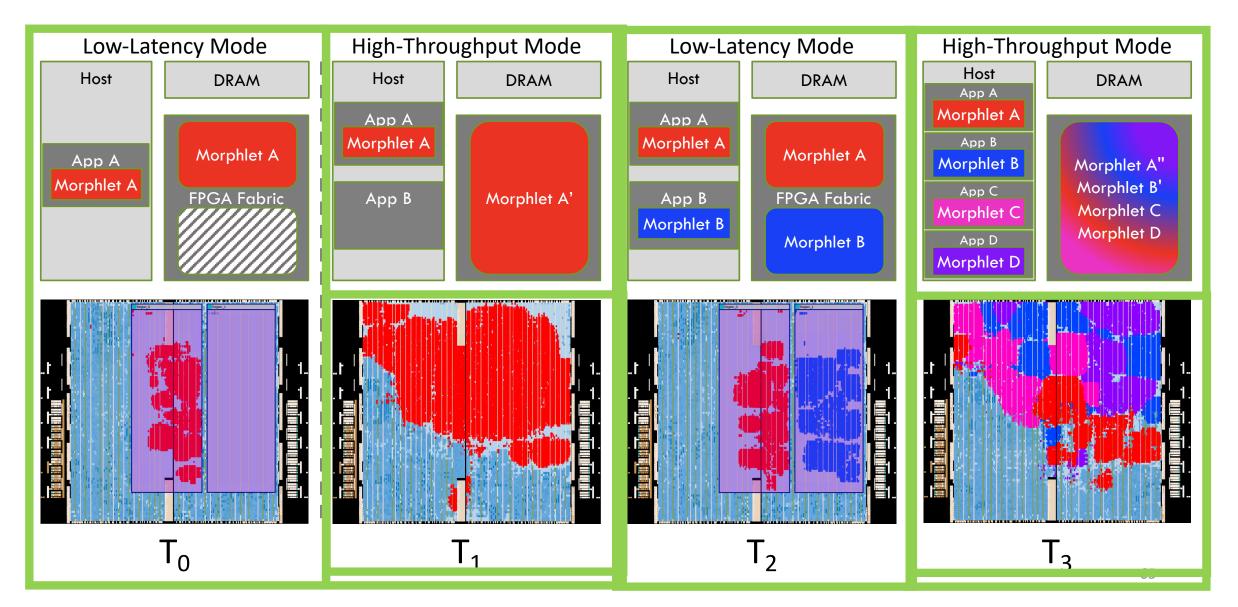


	Morphlets	Bitstream
stry	<a,b></a,b>	0x0a1
Registry	<a,b,c></a,b,c>	0x0fb01
	<b,c></b,c>	0x11ad

## Scheduling Morphlets

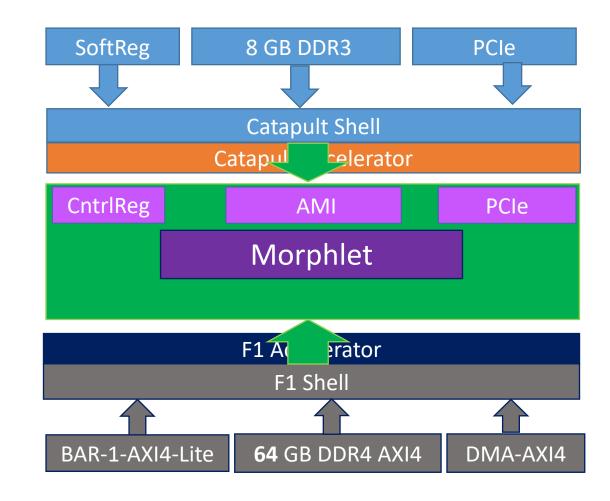
- Tradeoff 🛛 😶
  - Fixed zones + PR  $\rightarrow$  fast, fragmentation
  - Global zone + PaR  $\rightarrow$  eliminates fragmentation, slow
- AmorphOS: best of both worlds
  - Low Latency Mode
    - Fixed zones + PR
    - Default Morphlet bitstream
  - High Throughput Mode
    - Combine multiple Morphlets
    - Co-schedule on a global zone

# Scheduling Case Study



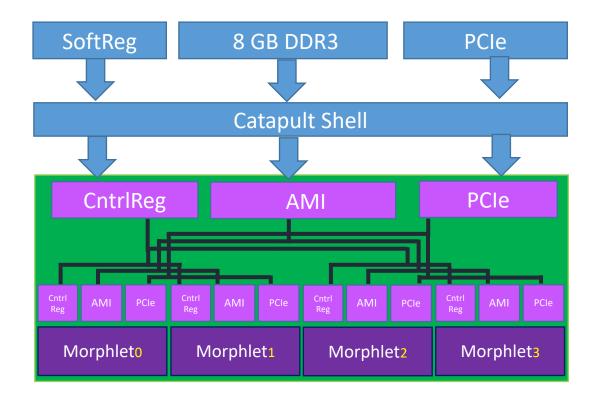
# AmorphOS Hull

- Hardens and extends vendor Shells
  - Microsoft Catapult
  - Amazon F1
- AmorphOS Interfaces
  - Control: CntrlReg
  - Virtual Memory: AMI
  - Bulk Data Transfer: Simple-PCIe



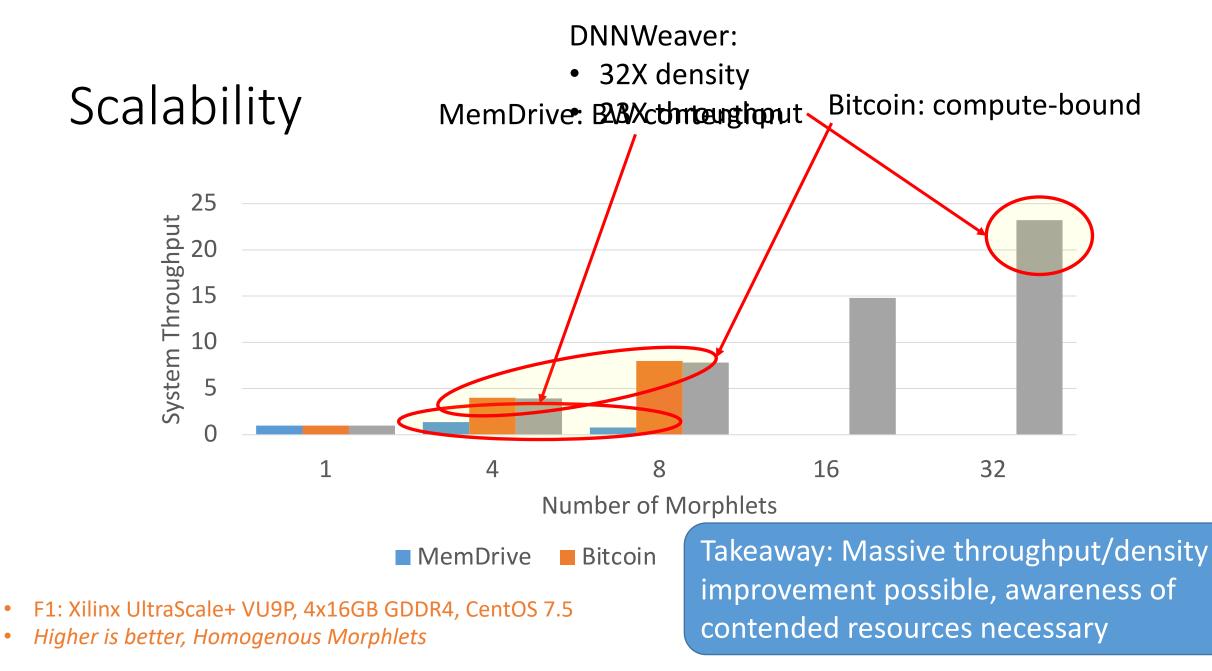
# AmorphOS Hull

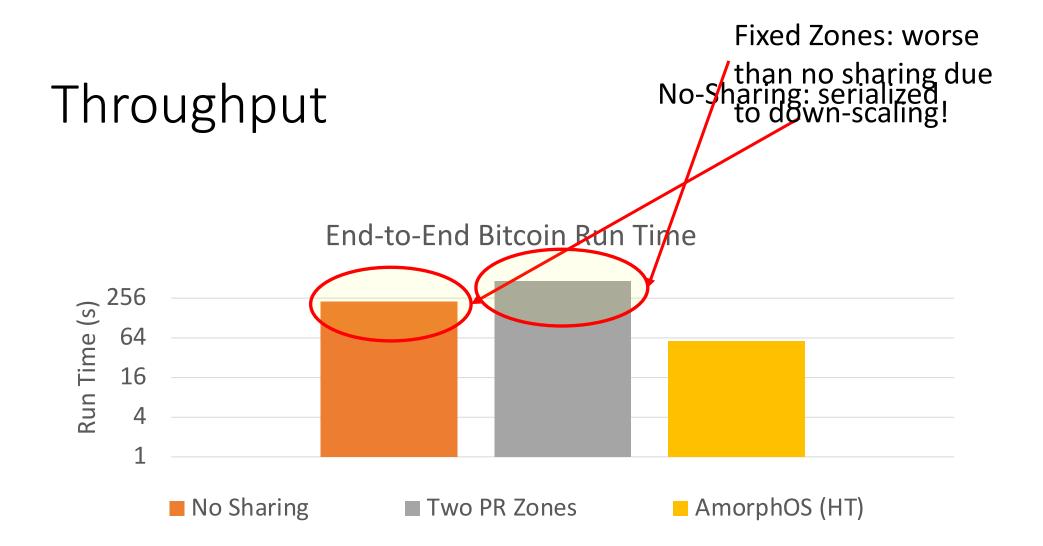
- Hardens and extends vendor Shells
  - Microsoft Catapult  $\rightarrow$  Higher Level
  - Amazon F1  $\rightarrow$  Lower Level
- AmorphOS Interfaces
  - Control: CntrlReg
  - Virtual Memory: AMI
  - Bulk Data Transfer: Simple-PCIe
- Multiplexing of interfaces
  - Isolation/data protection
  - Scalable, 32 accelerators
    - Tree of multiplexers



# Implementation & Methodology

- Catapult Prototype
  - Altera Mt. Granite Stratix V GS 2x4GB DDR3, Windows Server
  - Segment-based protection, partial reconfiguration (PR)
- Amazon F1 Prototype
  - Xilinx UltraScale+ VU9P, 4x16GB GDDR4, CentOS 7.5
  - No PR, but much more fabric than Catapult
- Workloads
  - DNNWeaver DNN inference
  - MemDrive *Memory Bandwidth*
  - Bitcoin *blockchain hashing*
  - CHStone 11 accelerators (e.g. AES, jpeg, etc)





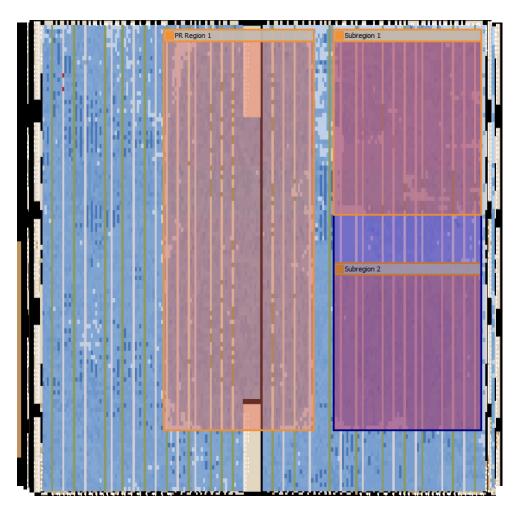
- 8 Bitcoin Morphlets
- Catapult Altera Stratix V GS 2x4GB DDR3, Windows
- Registry pre-populated: ctxt sw. 200ms
- Log Scale, Lower is better

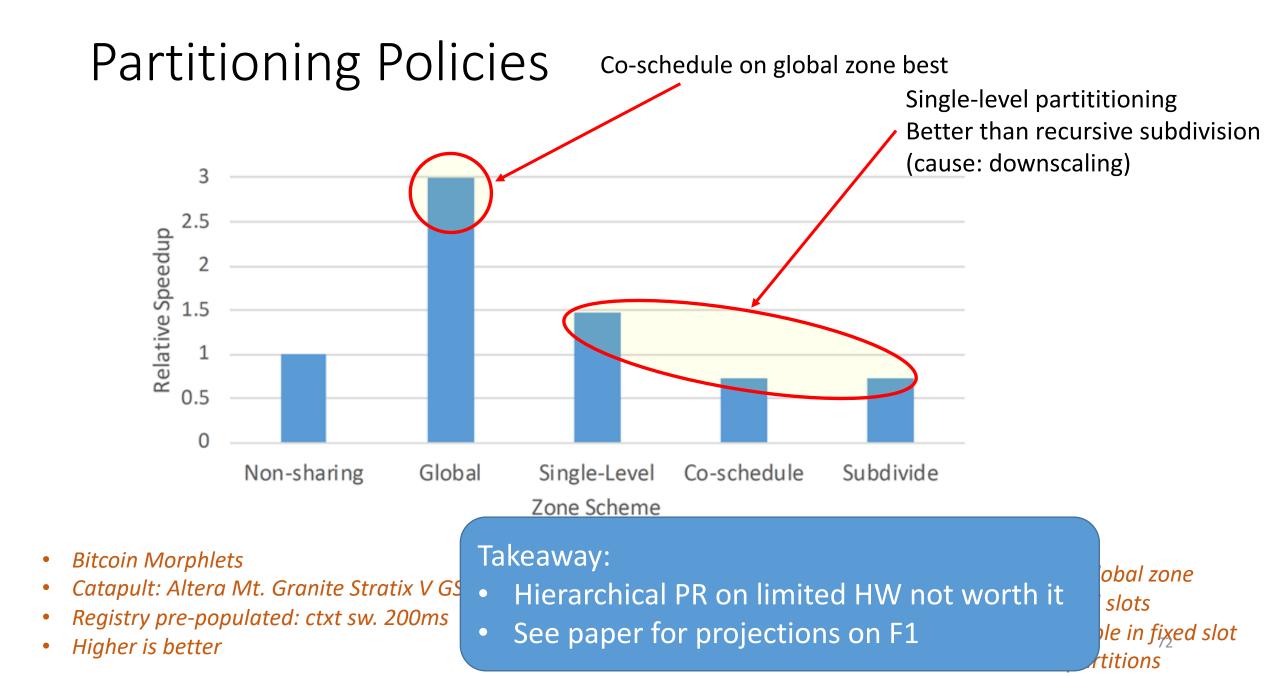
Takeaway: Co-scheduling on a global zone can perform better than fixed-sized slots and PR

# Partitioning Policies

**Gitte Charachige** ne scheme

- Main Antipation Anti
- Single PR zone





# Related Work

### Access to OS-managed resources

- Borph: So [TECS '08, Thesis '07]
- Leap: Adler [FPGA '11]
- CoRAM: Chung [FPGA '11]
- First-class OS support
  - HThreads: Peck [FPL'06], ReconOS: L
    übbers [TECS '09] -- extend threading to FPGA SoCs
  - MURAC: Hamilton [FCCM '14] extend process abstraction to FPGAs

### Single-application Frameworks

- Catapult: Putnam [ISCA '14] / Amazon F1
- Fixed-slot + PR
  - OpenStack support: Chen [CF '14], Byma [FCCM '14]; Fahmy [CLOUDCOM '15];
  - Disaggregated FPGAs: Weerasinghe [UIC-ATC-ScalCom '15]

### **Overl**ays

- Zuma: Brant [FCCM '12],
- Hoplite: Kapre [FPL '15],
- ReconOS+Zuma: [ReConfig '14]

# Conclusions & Future Work

- Compatibility Improved
  - without restricting programming model
  - Comprehensive set of stable interfaces
  - Port AmorphOS per platform not each accelerator per platform
- Scalability achieved within and across accelerators
  - AmorphOS transparently scales morphlets up/down
  - Powerful combination of slots/Partial Reconfiguration and full FPGA bitstreams
- Future work
  - Transparently scale across multiple FPGAs
  - Scale across more than just FPGAs
  - Open source AmorphOS/port to more platforms

# Questions?

### **Example alignment view**

00571 45000581 45000591 45000701 45000711 45000721 TGGCTGATGCAGGCGGGCGTAGTCAGCATAGTGGTCGTCAACCTTTACGGGGGCAAA	WAGCAACATGACGCTCTCGCATGGAAAGATCGCCTCCCCAG	CGGGGACAAGCAACAAGACAAAAGTGCCCAGGT	CAGAGECEGGGEAGGGGG	пестгостетес	TTCAAGTCAACCCTCATCTCTA	STGACAGACCCACG	TTTAACACCTTCACGTTGCAGTCTAAAAGCA	TTTCATATCTGCTGAT	TTCATCGCAATCACCAGGTGAGTGGGTCTT	ATTECCACCCCCACT	TCACAGCTGGCAAACGGAGCTC	STTCAGAGAGGCAGAAT	TESCTTEC
GGC TGATGCAGGCGGCGTAG TCAGCATAG TGG TCG TCAACC TTTACGGGGGGCAAA	MAGEAACATGACGATCTCGCATGGAAAGATCGCCTCCCC MAGCAACATGACGCTCTCGCATGGAAAGATCGCCTCCCCA	GGGGACAAGCAACAAGACAAAAGTGCCCAGGT GGGGACAAGCAACAAGACAAAAGTGCCCAGGT	CAGAGCCCGGGCAGGGGGG	IGG ICC IG IGG	CANGICAACCC ICA IC IC A	STGACAGACOCACG		TTTCATATCTGCTGAT	TICA TOGCAA TOACCAGE IGAG IGGG ICT	A ICCCACCCCAC	TCACAGCTGGC acggagetor	ttcagagaggcagaat	Looc Lloc
gcigalgenggcggcgiagicogcalagiggicgicaecttacgggggcaa ggcigalgenggcggcgiagicogcalagiggicgicaecttacgggggcaa ggcigalgenggcggcgiagicogcalagiggicgicaecttacgggggcaa			CagagCCCgggCagggggg CagagCCCgggCagggggg		ticangicance cateleta ticangicance cateleta	I gacagacccacg	TTT caccilcacgligcagiciaaaagca tita ACCTCACGTGCAGTCAAAACAA titaacacci ACGTGCAGTCAAAACAA	TTICATATC IGC IGAT	T CA TOCCAA TCACCAGG IGAG IGGG IC T TCA TOCCAA TCACCAGG IGAG IGGG IC T TCA TOCCAA TCACCAGG IGAG IGGG IC T	AT CCCACCCCAC		GTTCAGAGAGGCAGAAT GTTCAGAGAGGCAGAAT	
TECAGEGEGEG TAG ICAGEA TAG LEG ICG ICAACETTTAEGEGEGECAM ocagegegegegegegegegegegegegegegegegegege	saocaaca gacoc c coca ggaaaga cocc ccccag	C00003C330C3 C3303C33330 OCCC300	cagageccggggcagggggg	toattectatoa	t caad caacce cate ca	toacagacccacg	ttascaccitca agiciasaspos ttascaccitcac agiciasaspos ttascaccitcacgi CTAMACA	tticatatclocloat	ttcatcocaatcaccaogtgagtgggtct	a teccacececae	Cacaoc toocasacogaoc too	attea GAAT	TGGC TTGC
99 ang copy of land conjust and top for the conjugation of the conjuga	Isagesises I gaege I e legen I ggasaga I egee I ecceage MAGEMICATGACGE I E TEGEA I GGAMAGA TEGEE I ECCEAGE	cggggacaagcaac agacaaaagtgcccaggh CGGGG/CMGCMCMGACA aagtgcccaggh	cagagcccgggcagggggg		Caag Caacce calle te la	I gacagacecacd	tttaacaccttcacgttgcagt ANAAGCA	TTEATATE IGE IGAT	CA CGCAA CACCAGG GAG GGG C	A CCCACCCCAC	TCACAGE TGGCAAACGGAGE TC	GTTCAGAGAG t	1990 1190
SOCTG AGTEMBER AGTEMBER OF CAACETT ACGOGGECAA SOCTGATIGC GCATAGTIGG ICG ICAACETT ACGOGGECAA	MAGCANCA IGACISC IC ICISCA IGGAANGA ICISCO ICCCCAR MAGCAACA IGACISC IC ICISCA IGGAANGA ICISCO ICCCCAR MAGCAACA IGACISCI IC ICISCA IGGAANGA ICISCO ICCCCAR	CGGGGACMGCMCMGACAMA CCCuggt CGGGGACMGCMCMGACAMAGTGCC GGT	CAGAGCCCGGGCAGGGGGG	I GG T I CC I G I GG	TTC///GTC///CCCTCATCTCTA	STGACAGACCCACG	TTTAACACCTTCACGTTGCAGTCTAAAAGCA	TCATATCTGCTGAT	TTCA TCGCAATCACCAGGTGAGTGGGTCTC	T DADDODADDD TA	TCACAGCTGGCAAACGGAGCTCO	GTTC/G/G/G/GGC/G//T	TGGC go
ggctgstgcs AGTGGTCGTCAACCTTTACGGGGGCAAA GGCTGATGCAG gtggtcgtcsscctttscggggggcsss GGCTGATGCAGGCG TGGTCGTCAACCTTTACGGGGGCAAA	MACHAEL GACOC C COCA GOMMON COLC CLCCAN MAGCHICA TGACOC C COCA GOMMON COLC CLCCAN	cggggacaagcaacaagacaaaag igcccaggt	agagcccgggcagggggg		theory career or chercha theory career or chercha theory career or chercha	Lancagacccaca	l lancace i leneg i igeng i e lannapen l lancace i leneg i igeng i e lannapen l lancace i leneg i igeng i e lannapen	ttt TCTGCTGAT	TCA COCAA CACCAGO GAG GOO C		CACAGE IGECAMEGGAGE IC	TTCAGAGAGGCAGAAT	GGC TTGC
ggctgatgcaggcg tggtcgtcaacctttacgggggcaaa	isagcaaca tgacge te tegea tggaaaga tegee teeccaga	cggggacaagcaacaagacaaaagtgcccaggt	c GCCCGGGCAGGGGGG	IGG TOC IG IGG	TTC///GTC///CCCTC//TCTCTA	STGACAGACCCACG	TTAACACC TCACG TGCAG TC TAAAAGCA	TTTCAT CTGAT	TICA TOGCAA TCACCAGG TGAG TGGG TC T	A CCCACCCCAC	TCACAGC TGGCAAACGGAGC TCO	STTCAGAGAGGCAGAAT	TGGC TGC
IGCTGATGCAGGCGGCGTAGTCAGCA GTCGTCAACCTTTACGGGGGGCAAA	INNEGRANCH IGNORE I C'OCCH IGGANAGA I COCCH COCCHO MAGCAACA IGACGC I CICCCA IGGAAAGA I COCCH COCCAG MAGCAACA IGACGC I CICCCA IGGAAAGA I COCCH COCCAG	сбббблембемемблеммбтоссибат сбббблембемемблеммбтоссибат		testiccistes	I congi conceci ca i e le la TICAAGICAACCEICATETETA I congi conceci ca i e le la	tgacagacccacg	ttlaacacctlcacgtlgcagtclaaaagca	tttcatatctgctga	atcocaatcaccaogtoagtoogtott	atteccaccecact	tcacagetggcatacggagetco	ttcagagaggcagaat	toocttoo
GCTGATGCAGGCGGCGTAGTCAGCATAGT casectttacgggggcaaa getgatgcaggcggcgtagtcagcatagtg casectttacgggggcaaa	nagenaen ignegel el egen igganagn i egee i ecceggi nagenaen ignegel el egen igganagn i egee i eccengi	cggggacaagcaacaagacaaaagtgeccaggti	cagageee ggg	TGGTTCCTGTGG	TTCAAGTCAACCC I Calle Le la TTCAAGTCAACCC TCATCTCTA	TGACAGACCCACG	<pre>titaacacclicacgligcagiclassagca trrMCACCTTCACGTT6CAGTCTAAAA6CA</pre>	tttcatatctgctgat	tt tcaccaggigagigggicti TTCAT CACCAGGTGAGTGGGTCT	a l'eccaccocacit ATTOCCACCOCCACT	tcacage1ggcaaacggage1c TCACAGCTGGCAAACGGAGCTC	g til cogogoggcogoo t STTCAGAGAGGCAGAAT	TGGC TTGC
ppctgatgcaggcggcgtagtcagcatagtg cctttacgggggcaaa S6CT6AT6CA66C66C6TA6TCA6CATA6T66 CTTTAC666666CAAA	HINGCANCA GACGE E EGEN GGANAGA EGEE EEEEAG MAGEANEA GAEGE E EGEA GGANAGA EGEE EEEEAG	cggggacaagcaacaagacaaaag (gcccagg) CGGGGACAAGCAACAAGACAAAAG TGCCCAGGT	CAGAGCCCGGGC	GGTTCCTGTGG			It is acgee it eacy it goag is is assages TTTMC/CCTTC/CGTT6C/GTCTAAAAGCA	TTICATATCTGCTGAT	TTCATC gtgagtgggtcti	ATTOCCACCCCCACT attoccacccccact	CACAGCTGGCAAACGGAGCTCC Leacage Lggcaaacggage Lea TCACAGCTGGCAAACGGAGCTCC	GTTCAGAGAGGGCAGAAT gttcagagaggcagaat	igge titge
SGC TGATGCAGGCGGCGTAGTCAGCATAGTGG CHIgacgggggcoss SGC TGATGCAGGCGGCGTAGTCAGCATAGTGG GGGGGCAA SGC TGATGCAGGCGGCGTAGGCAGCATAGTGGTC gggcoss	IN GENERAL GREECE COLOGEN GENERAL COLORIS	CggggliCalgealealagaealang geccagg CggggliCalgealealagaealang geccagg	CAGAGCCCGGGCAGGGGG	ggtteetgtgg tteetgtgg	cang cance cale cale	I gacagacccacg	LLI ANCACC   CACG   I gCag   C   ANAAGCA LLI ANCACC   CACG   I gCag   C   ANAAGCA TTTAACACC   CACG   I GCAG   C   AAAAGCA	tticalatelgelga	tteate TGAGTGGGTCT tteatege GAGTGGGTCT	AT ICCCACCCCCAC	CACAGE IGGCAAACGGAGE ICO TCACAGE IGGCAAACGGAGE ICO TCACAGE IGGCAAACGGAGE ICO	GTTCAGAGAGGCAGAAT GTTCAGAGAGGCAGAAT	GGC TTGC
	INNOCALE AND CONTRACT	CGGGGI CHIGCHICHIGH CHIMIG GCCCAGGT CGGGGI CHIGCHICHIGH CHIMIGTGCCCAGGT	CAGAGCCCGGGCAGGGGGG CAGAGCCCGGGCAGGGGGG	TGGTT tataa	t caagi caacce teateteta	I dacadacceacd	taacacc t caco tocag to aaaaoca	tticatatcloctoat	ticalogoate eti TICALOGOATCACCA T		CICIES GOLINE CONCERNENCE CONCERNE	tt cagagaggcagaat	
ggclgalgcaggcggcglaglcagcalaglgglc GGClGAlGCAGGCGGCGTAGICAGCATAGIGGICGICA GGClGAlGCAGGCGGCGTAGICAGCATAGIGGICGICAA	ASCALCATISADSCTC LOSCATIGAAAGATOSCC LOCCAG CALCATISADSCTC LOSCATIGAAAGATOSCC LOCCAG aucutascoc LC LOSCATIGAAAGATOSCC LOCCAG			IGGTTCCTGTG Lggttcctgtgg		TEICIGICCOCICE	TTANCICCTCICGTECIG: CTANAGON TTANCICCTCICGTECIGICTANAGON TTANCICCTCICGTECIGCIGCIGCIGCIANAGON	TTICATATC IGC IGAT	TTCATCGCAATCACCAGGTG TTCATCGCAATCACCAGGTGA	CCCACCCCCAC	TCACAGC TGGCAAACGGAGC TCA tcacage tggcaaacggage to	GTTC/G/G/GGC/G/AT	GICTIAC
got tast geogogg tagt cages tagt ggt cgt caace	BED IGREGETE COCCIONNAL STREET STR	CGGGGACAAGCAACAAAATTGCCCAGGT	CAG	TGGTTCCTGTGG 99	TTCAAGTC CCTCATCTCTA theory tenecode tene to te	TGICIGICCICG tgactgacccacg	TTMC/CCTTC/CGTTGC/GTCT/////GC/ titlascace.itcacgtigcagictassages	TTTCATATCTGCTGAT	TTCATCGCAATCACCAGGTGAGTGGGTC thcatcgcaatcacc	cccccact	TCACAGE 166CAAAC66AGE 10 I cocoge I ggenooeggoge I e	STTENGNGNGGENGNNT	TGGCTTGC
ggelgalgenggeggeglagiengenlagigglegienaeettt ggelgalgenggeggeglagiengenlagigglegienaeetttae	tetegeatggaaagategeeteeeag	cggggacaagcaacaagacaaaagtgcccaggt cggggacaagcaacaagacaaaagtgcccaggt	Cagageeegggcagggggg	togtteetgtog	tteaagte eatetela tteaagteaa tela	tgacagacccacg	Litancaccit cacgit goagic lasaagoa Litancaccit cacgit goagic lasaagoa TTTA/CACCTCACGTCAGIC TAAAGCA	tttcatatctgctgat	ttcatcgcaatcaccaggtgagtgggtctl		Cacage agenaneggage of teacage agenaneggage of TCACAGE IGGENANEGGAGE OF	gttcagagaggcagaa t	itggettge
ggotiga tigoaggoggog tag tongon tag tigg tog tonacettilae ggotiga tigoaggoggog tag tongon tag tigg tog tonacettilae ggOtigA tigOAGGOGGOGTAG TOAGCA TAG TGG TOA TOAACOTTACGGG	Choges toges and the construction of the const	cggggacaagcaacaagacaaagtgcccaggta cggggACAAGCAACAAGACAAAAGTGCCCAGGT cggggACAAGCAACAAGACAAAAGTGCCCAGGT	Cagagecccgggcagggggg CAGAGECCGGGCAGGGGGG	I GGT I CC I G I GG	TTCALCTCALCOCTC	TELEVENCE	TTAK ACCUTCACCT TO ACCUTCACCE	TTTPATATE TOP TOAT	TTENTERCHATCHEELET		CACAGE IGGCAAACGGAGE IC Leacage Iggcaaacggage I c	GTTC/G/G/G/GGC/G/AT pttcsgsgsggcsgsst	IGGC TIGC
ggetgatgeaggeggegtagteageatagtggtegteaacetttaeggg	aagategeeteeeaga	cggggacaagcaacaagacaaaag tgcccagg t	engngeeegggengggggg engngeeegggenggggggg	logitec igigg	the angle accel cale let	GICIGICCCICC	ACACE CACE GOOD C ANALOCA	TTICATATC IGC IGAT	TTCATCGCAATCACCAGGTGAGTGGGTCTT TTCATCGCAATCACCAGGTGAGTGGGTCTT TTCATCGCAATCACCAGGTGAGTGGGTCTT		tenenge tiggenninegginge Ten TEAEAGETIGGEAAACGGAGETIG AGETIGGEAAACGGAGETIG EAAACGGAGETIG	STTCAGAGAGGCAGAAT STTCAGAGAGGCAGAAT	IGGC TIGC
ggotgatgoaggoggoggaagtoagcatagtggtogtoaacotttacgggg ggotgatgoaggoggoggaagtoagcatagtggtogtoaacotttacgggg GGCTGATGCAGGCGGCGTAGTCAGCATAGTGGTCGTCAACCTTTACGGGGGCAA	GCCTCCCCAG	CGGGGACAAGCAACAAGACAAAAGTGCCCAGGT	CAGAGCCCGGGCAGGGGG				TTAACACCTTCACGTTGCAGTCTAAAAGCA	TTTCATATCTGCTGAT	TTCATCGCAATCACCAGGTGAGTGGGTCT TTCATCGCAATCACCAGGTGAGTGGGTCT	AT CCCACCCCAC	T AAACGGAGCTC	GTTCAGAGAGGCAGAAT GTTCAGAGAGGCAGAAT	IGGCTTGC
GGC I GA I GCAGGCGGGCG I AG I CAGCA I AG I GG I CCIACC I TT ACGGGGGCAA GGC I GA I GCAGGCGGCG I AG I CAGCA I AG I GG I CCIACC I TT ACGGGGGCAAA	001000000	C9999#C899C88C899#C888919CCC8991 C66664C446C44C4464C4444GT6CCC466T1 C66664C446C44C4464C4444GT6CCC466T1	CAGAGCCCGGGCAGGGGGG	TEGTTCCTGTEG		STGACAGAC g	ttlaacacctlcacgtlgcagtclaaaagca	tttcatatctgctgat	ttcatcgcaatcaccaggtgagtgggtcti		tcacag aacggageter	g t Leogagaggeogaa t g t Leogagaggeogaa t	tggettge
ggctgatgcaggcggcgtagtcagcatagtggttgtcaacctttacgggggcaaa	saage AG	cggggscaagcaacaagacaaagtgcccaggt CGGGG/C/AGC/AC/AGIC/AAAGTGCCC/AGT	CHIGHIGCCCGGGGCHGGGGGGGGGGGGGGGGGGGGGGG	TGGTTCCTGTGG	TCMG/CMCCCCATCTCA	TGACTGATCCAC	ttlaacaccalcacgcltcagtclaaattca	ttigatatcigcigat	ttcalcocalcaccagigagigggtct TTCATCGCAATCACCAGGTGAGTGGGTCT ttcat			t t cogogoggcogoo t t t cogogoggcogoo t	tggcttgo
GGC IGA IGCAGGCGGCG IAG ICAGCA IAG IGG ICG ICAACCTT ACGGGGGCAAA ggC iga igcaggcggcg iag icagca iag igg icg icaacctt iacgggggcaaa GGC IGA IGCAGGCGGCG IAG ICAGCA IAG IGG ICG ICAACCTT ACGGGGGCAAA	WAGCAA seegcaacat	gggacaagcaacaagacaaaagtgcccaggt gacaagcaacaagacaaaagtgcccaggt	cagageeegggcagggggg	tggtteetgtgg	tteaagteaacceteateteta	rtgacagacccacg	tttaa TTCACGTGCAGTCTAAAAGCA	TTICATATCTGCTGAT	ti cal cocal caccaggi gagi gggi ci i TTCATCGCAATCACCAGGTGAGTGGGTCT	AT CCCACCCCAC	tcacagetggcasac of TCACAGCTGGCAAACGG	gagaggcagaat gagaggcagaat	toocttoc
gi oʻnga i gonggoggogi agʻonaon agʻiggʻogʻonaoci ti acgggggona gi oʻnga i gonggoggogʻagʻonaon agʻiggʻogʻonaoci ti acgggggona GGC IGA I GOAGGOGGGI AGʻOAGCA I AgʻIGGʻOG CAACCTTI ACGGGGGAAA	wachichterege wagesnesitgsege	Casgessesagecasagtgeceagt casgessesagecasagtgeceagt CAICAIGICAAAGTGCCTAGT	ca8a8ccc888ca888888	99 CC 9 99	ticangleancecleateleta ticangleancecleateleta	I decedencesed	ttiaaca ctassagca ttiaaca TAAAGCA	TTTCATATC TOCTGAT	Callogenal caccargo gargi gargi ci TICA ICGCAA ICACCAGGI GAGI GGGI CI	AT CCCACCCCCAC	TCACAGC TGGCAAACGGAGC TC	STTCAGAGA	
ggeligs i genggeggeg ing lengen ing i ggleg lenseet it inegggggens ggeligs i genggeg leg ing lengen ing i ggleg lenseet it ineggggggens dae lan leenadeade ing lenden ing i ggleg lenseet it ineggggggens	sagcsacatgacgetete	ACAAAAGTGCCCAGGT	CAGAGCCCGGGCAGGGGGG	I GG I I CC I G I GG	TTCANGTCAACCC TCATCTC TA	STGACAGACCCACG	TTTACACCTTCACGTTG tttasceccttcscgttgcagtc	catatetgetgat	Li Calicgcaa i caccaggiigagiigagiigagiic Li Li Calicgcaa i caccaggiigagiigagiigagiic Li TICA ICGCAA ICACCAGGIGAGIIGGGIIC II	atteccaccecet	Cacage agenateggage e	ttcagagaggcagaat ttcagagaggcagaat	tggct gc
gcignigcogocgicgingicopeningingicgicanecitiacgggggenn gcignigcogocgicgingicopeningingicgicanecitiacgggggenn	sagcaaca tgacgc to togca tgga	caggi	CagagCCCgggCagggggg CAGAGCCCGGCAGGGGGG	I GGT ICC I G I GG	Construction of the second sec	TGACAGACCCACG	tttaacaccttcacgttgcagtctaaaag TTTAICACCTTCACGTTGCAGTCTAAAAGCA						c
ggc igni gcaggeggeg i ag i cagca i ag i gg i cg i cance i ti acgggggcan GGC iGA i GCAGGCGGCG i AG i CAGCA i AG i GG i CG i C	International and the second s		agagcccgggcagggggg	rtggttectgtgg G	CANGI CARCCE CALLE I A	TGACAGACCCACG	It is a cace it caegit geagit is as a gea TTANCAECT CAEGITIGEAGIT ANAAGEA	TTTCATATCTGCTGAT	TICATOGCANTCACCA				
GGC TGA TGCAGGCGGCG TAG TCAGCA TAG TGG TCG TCAACC TTTACGGGGGGCAAA	UNIGCANCATGACGCTCTCGCATGGAAAGATCGC			G	AGTCAACCCTCATCTCTA	STGACAGACCCACG	TTAICACCTICACGTIGCAGTCTAAAAGCA	TTTCATATC TGC TGAT	TTCATCGCAATCACCAGGTGA				
ggclgslgcsggcglaglcagcslaglgglcglcaacclltacgggggcaa	saagesaes ligaege ie tegen liggagaga liegee ieeee				aglesacceleateleta tesacceleateleta	gtgacagacccacg	tttaacaccttcacgttgcagtctaaaagca						
					CARCOCTCATCTCTA CARCOCTCATCTCTA assocication	STGACAGACCCACG	TTTAACACCTTCACGTTGCAGTCTAAAAGCA	TTTCATATC TGCTGAT	TTCATCGCAATCACCAGGTGAGTG				
						GICIGICCCICG	TT ANCACC T CACG T GCAG TC TANAAGCA	CATA CIGCIGAT	TICA COCAN CACCAGE GAG GGG C				
							ttianenee kieneg kigeng te lannagen ttianenee kieneg kigeng te lannagen	tttcatatctgctgat	tical cocast caccagg i gag i gog i c ti	atteccaccecet	tereaget		

Aligned reads —

## Sequence alignment: Scoring

Option 1	Option 2	Option 3
- A C - G G C - G	- A C G G - C - G	- A C G - G C - G
T	T A C G G G C A G	T A C G G G C A G

- Scoring matrices are used to assign scores to each comparison of a pair of characters
- Identities and substitutions by similar amino acids are assigned positive scores
- Mismatches, or matches that are unlikely to have been a result of evolution, are given negative scores

А	С	D	Е	F	G	Н	I	К	
А	С	Y	Е	F	G	R	Ι	Κ	
+5	+5	-5	+5	+5	+5	-5	+5	+5	

# Pairwise alignment: the problem

The number of possible pairwise alignments increases explosively with the length of the sequences:

Two protein sequences of length 100 amino acids can be aligned in approximately 10<sup>60</sup> different ways

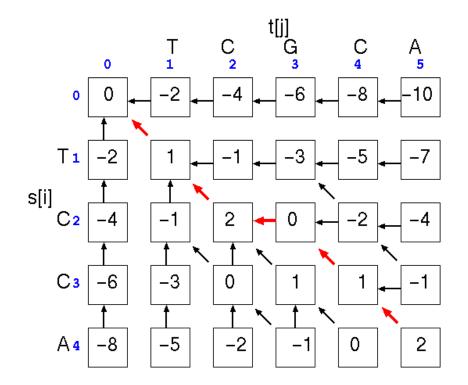


itude as the entire

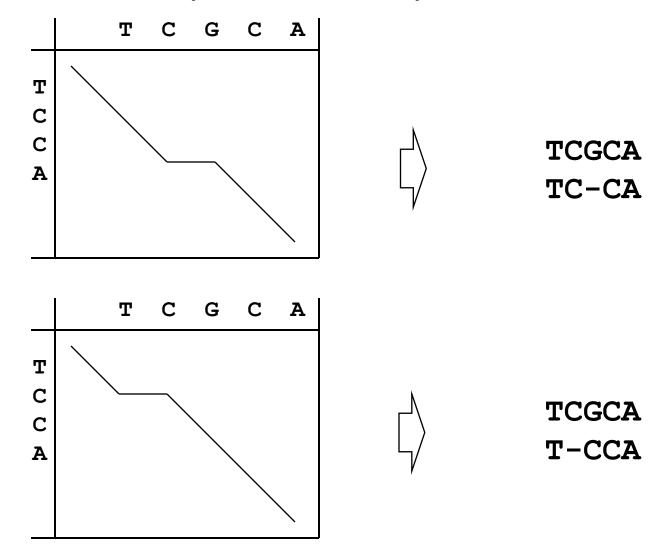
#### Pairwise alignment: the canonical solution

Dynamic programming

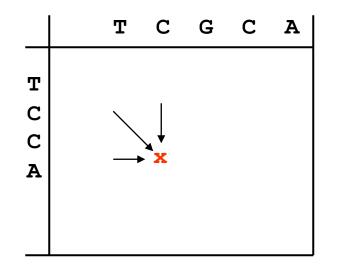
(the Needleman-Wunsch algorithm)



#### Alignment depicted as path in matrix



#### Dynamic programming: computing scores



Any given point in matrix can only be reached from three possible positions (you cannot "align backwards"). => Best scoring alignment ending in any given point in the matrix can be found by choosing the highest scoring of the three possibilities.

# Dynamic programming

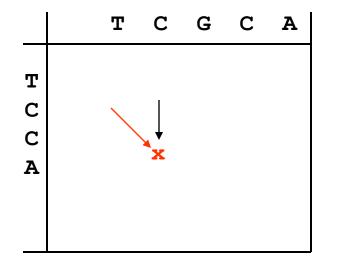
	Т	С	G	С	A
T C C A		↓ x			

Any given point in matrix can only be reached from three possible positions (you cannot "align backwards"). => Best scoring alignment ending in any given point in the matrix can be found by choosing the highest scoring of the three possibilities.

score(x,y-1) - gap-penalty

score(x,y) = max

## Dynamic programming

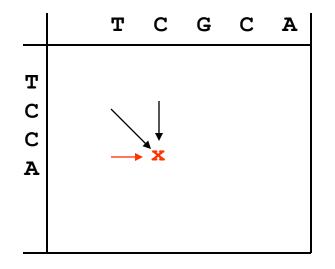


Any given point in matrix can only be reached from three possible positions (you cannot "align backwards"). => Best scoring alignment ending in any given point in the matrix can be found by choosing the highest scoring of the three possibilities.

score(x,y-1) - gap-penalty
max { score(x-1,y-1) + substitution-score(x,y)

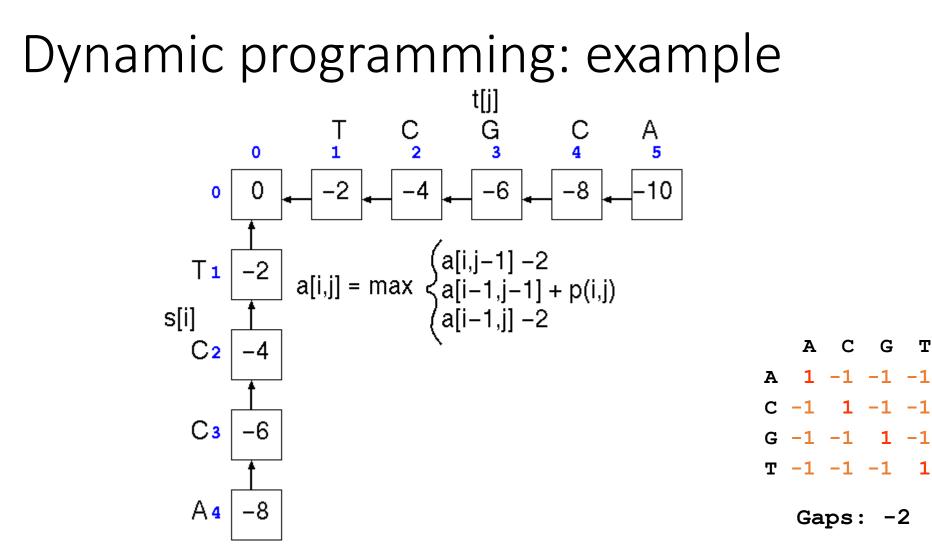
score(x,y) = max

# Dynamic programming



Any given point in matrix can only be reached from three possible positions (you cannot "align backwards"). => Best scoring alignment ending in any given point in the matrix can be found by choosing the highest scoring of the three possibilities.

 $score(x,y) = max \begin{cases} score(x,y-1) - gap-penalty \\ score(x-1,y-1) + substitution-score(x,y) \\ score(x-1,y) - gap-penalty \end{cases}$ 

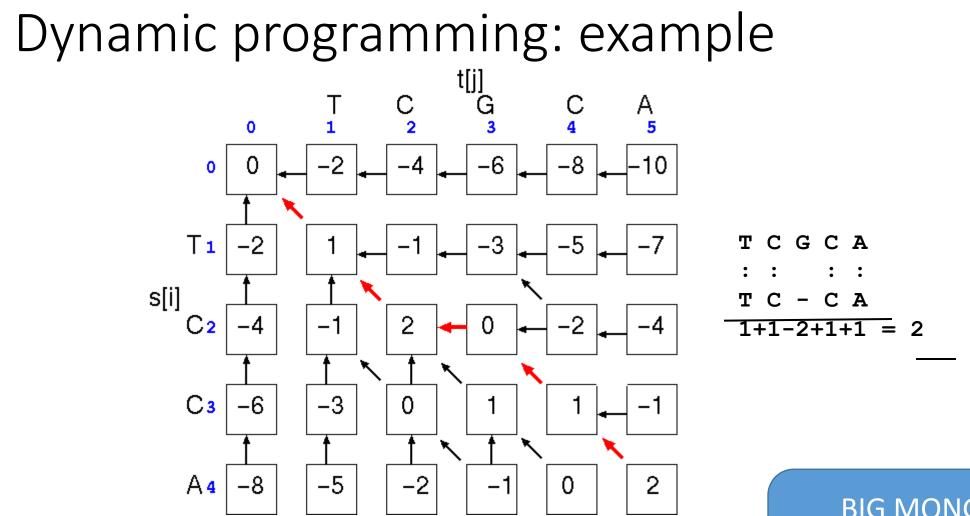


Dynamic programming: example t[j] G 3 C 2 А 5 T 1 C 4 0 -2 -6 -8 -4 10 0 0 Τı -2  $a[i,j] = \max \begin{cases} a[i,j-1] - 2 \\ a[i-1,j-1] + p(i,j) \\ a[i-1,j] - 2 \end{cases}$ s[i] C<sub>2</sub> -4 Сз -6 A4 -8

Dynamic programming: example t[j] G 3 C 2 А 5 T 1 C 4 0 -2 -6 -8 -4 10 0 0 Τı -2  $a[i,j] = \max \begin{cases} a[i,j-1] - 2 \\ a[i-1,j-1] + p(i,j) \\ a[i-1,j] - 2 \end{cases}$ s[i] C<sub>2</sub> -4 Сз -6 A4 -8

Dynamic programming: example t[j] G 3 C 2 Т 1 А 5 C 0 -6 -2 -8 10 -4 0 0 Τı -2 -3 -5 -7 \_ ] **K**−2 -5 s[i] C2 2 -1 0 -4 0  $a[i,j] = \max \begin{cases} a[i,j-1] - 2 \\ a[i-1,j-1] + p(i,j) \\ a[i-1,j] - 2 \end{cases}$ Сз -6 A4 -8

Dynamic programming: example t[j] G 3 C 2 А 5 T 1 C 0 -8 ' -6 -2 10 -4 0 0 Τı -7 -2 -3 -5 \_ s[i] C2 2 -2 0 -4 -1 -4 Сз -3 -6 0 1 -1 1 .2 -3 A4 -8 -5 -2 0 2 -1 .2



BIG MONGO HINT: What if each box is a parallel process?

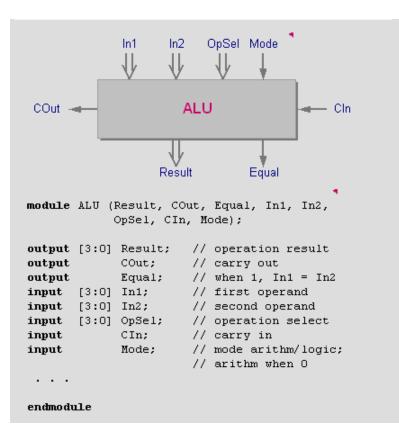
## References:

- Evita\_verilog Tutorial, <u>www.aldec.com</u>
- http://www.asic-world.com/verilog/

# Review: Module definition

- Interface: port and parameter declaration
- Body: Internal part of module
- Add-ons (optional)

module module_name (port_list) ;	
port declarations parameter declarations	interface
'include directives	add-ons
variable declarations assignments lower-level module instantiation <i>initial</i> and <i>always</i> blocks tasks and function	body
endmodule	module definition

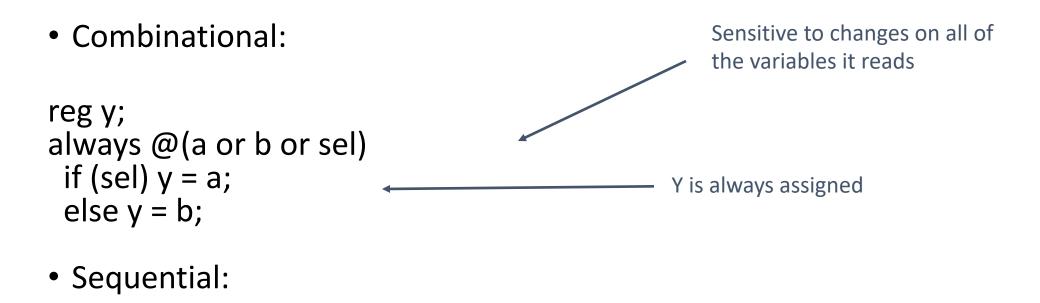


## Delays on Primitive Instances

• Instances of primitives may include delays

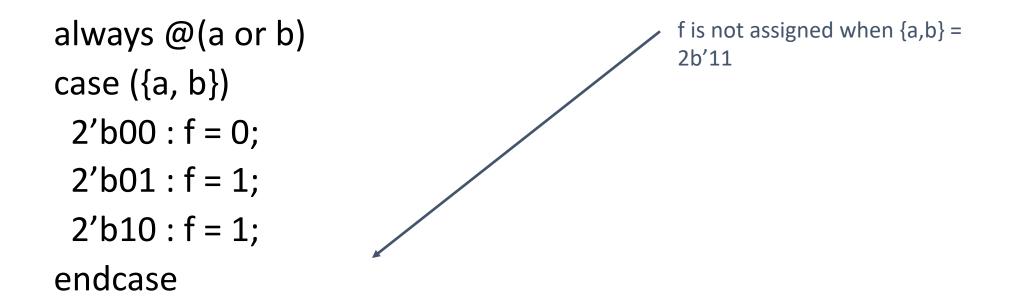
bufb1(a, b);// Zero delaybuf #3b2(c, d);// Delay of 3buf #(4,5)b3(e, f);// Rise=4, fall=5buf #(3:4:5)b4(g, h);// Min-typ-max

- The main trick
- reg does not always equal latch
- Rule: Combinational if outputs always depend exclusively on sensitivity list
- Sequential if outputs may also depend on previous values

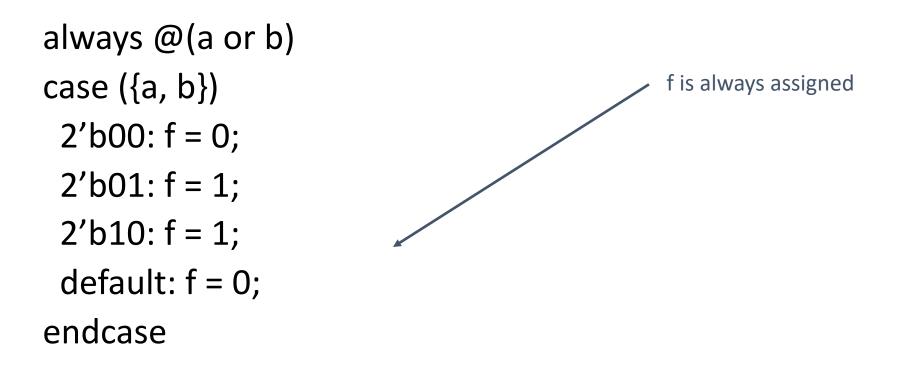


reg q; always @(d or clk) if (clk) q = d;

- A common mistake is not completely specifying a case statement
- This implies a latch:



• The solution is to always have a default case



# Inferring Latches with Reset

- Latches and Flip-flops often have reset inputs
- Can be synchronous or asynchronous
- Asynchronous positive reset:

always @(posedge clk or posedge reset)
if (reset)
q <= 0;
else q <= d;</pre>

# Simulation-synthesis Mismatches

- Many possible sources of conflict
- Synthesis ignores delays (e.g., #10), but simulation behavior can be affected by them
- Simulator models X explicitly, synthesis doesn't
- Behaviors resulting from shared-variable-like behavior of regs is not synthesized
  - always @(posedge clk) a = 1;
  - New value of a may be seen by other @(posedge clk) statements in simulation, never in synthesis

# Compared to VHDL

- Verilog and VHDL are comparable languages
- VHDL has a slightly wider scope
  - System-level modeling
  - Exposes even more discrete-event machinery
- VHDL is better-behaved
  - Fewer sources of nondeterminism (e.g., no shared variables)
- VHDL is harder to simulate quickly
- VHDL has fewer built-in facilities for hardware modeling
- VHDL is a much more verbose language
  - Most examples don't fit on slides