Generating Logical Structure:

top -> down process

starts at top ParseFile structure.

creates LogicalNode, seeded from equivalent seed ParseNode. The constructor is called by a branch, as it adds LogicalNodes to its parentNodes vector.

Conditions for when statements should be states. This would enforce that the condition variables are always valid.

|issue UX

>inst[3:0]

@0

$valid = …;

?valid

@1

$raw\_inst[28:0] = ..;

$mem\_op = $raw\_inst[28:27] == 2’b01;

@4

$sigA = $mem\_op;

?mem\_op

@3

$mem\_addr[50:0] = $op\_a + $raw\_inst[15:0];

@5

$sigB = $mem\_addr && $raw\_inst;

// |issue UX

for (int inst = 0; inst < 4; inst++)

begin

// @0

always\_comb valid\_UX00H\_Inst[inst] = ...;

// ?$valid

// @1

always\_comb raw\_inst\_UX01H\_Inst[28:0][inst] = ..;

always\_comb mem\_op\_UX01H\_Inst[inst] = raw\_inst\_UX01H\_Inst[28:27][inst] == 2’b01;

// @4

always\_comb sigB\_UX04H\_Inst[inst] = mem\_op\_UX04H\_Inst[inst];

// ?$mem\_op

// @3

always\_comb mem\_addr\_UX03H\_Inst[50:0][inst] = op\_a\_UX03H\_Inst[inst] + raw\_inst\_UX03H\_Inst[15:0][inst];

// @5

always\_comb sigA\_UX05H\_Inst[inst] = mem\_addr\_UX05H\_Inst[inst];

end

// Signal Declarations

//Clock gating condition

node valid\_uX00H\_Inst[3:0]; //assigned

node valid\_uX01H\_Inst[3:0];

node valid\_uX02H\_Inst[3:0];

node valid\_uX03H\_Inst[3:0];

`MSFF(valid\_uX01H\_Inst, valid\_uX00H\_Inst, ck\_grid\_u1n44)

`MSFF(valid\_uX02H\_Inst, valid\_uX01H\_Inst, ck\_grid\_u1n44)

`MSFF(valid\_uX03H\_Inst, valid\_uX02H\_Inst, ck\_grid\_u1n44)

//Generating clock gated by $valid

node ck\_valid\_UX00H\_u1n44\_Inst[3:0];

node ck\_valid\_UX01H\_u1n44\_Inst[3:0];

node ck\_valid\_UX02H\_u1n44\_Inst[3:0];

node ck\_valid\_UX03H\_u1n44\_Inst[3:0];

for(int inst = 0; inst < 4; inst++)

begin

`MAKE\_CLK\_LOC(ck\_valid\_UX00H\_u1n44\_Inst[inst], ck\_grid\_u1n22, 1'b1, valid\_uX00H\_Inst[inst], csr\_pwrdn\_ovrd\_unnnH)

`MAKE\_CLK\_LOC(ck\_valid\_UX01H\_u1n44\_Inst[inst], ck\_grid\_u1n22, 1'b1, valid\_uX01H\_Inst[inst], csr\_pwrdn\_ovrd\_unnnH)

`MAKE\_CLK\_LOC(ck\_valid\_UX02H\_u1n44\_Inst[inst], ck\_grid\_u1n22, 1'b1, valid\_uX02H\_Inst[inst], csr\_pwrdn\_ovrd\_unnnH)

`MAKE\_CLK\_LOC(ck\_valid\_UX03H\_u1n44\_Inst[inst], ck\_grid\_u1n22, 1'b1, valid\_uX03H\_Inst[inst], csr\_pwrdn\_ovrd\_unnnH)

end

//Gated pipe signal

node mem\_op\_UX01H\_Inst[3:0]; //assigned

node mem\_op\_UX02H\_Inst[3:0];

node mem\_op\_UX03H\_Inst[3:0];

node mem\_op\_UX04H\_Inst[3:0];

for(int inst= 0; inst < 4; inst++)

begin

`MSFF(mem\_op\_UX02H\_Inst[inst], mem\_op\_UX01H\_Inst[inst], ck\_valid\_UX01H\_u1n44\_Inst[inst])

`MSFF(mem\_op\_UX03H\_Inst[inst], mem\_op\_UX02H\_Inst[inst], ck\_valid\_UX02H\_u1n44\_Inst[inst])

`MSFF(mem\_op\_UX04H\_Inst[inst], mem\_op\_UX03H\_Inst[inst], ck\_valid\_UX04H\_u1n44\_Inst[inst])

end

//Generating clock gated by $mem\_op in addition to $valid

node valid\_AND\_mem\_op\_UX03H\_Inst[3:0]; //assigned

node valid\_AND\_mem\_op\_UX04H\_Inst[3:0];

assign valid\_AND\_mem\_op\_UX03H\_Inst = mem\_op\_UX03H\_Inst && valid\_UX03H\_Inst;

`MSFF(valid\_AND\_mem\_op\_UX04H\_Inst, valid\_AND\_mem\_op\_UX03H\_Inst, ck\_grid\_u1n44)

for(int inst = 0; inst < 4; inst++)

begin

`MAKE\_CLK\_LOC(ck\_valid\_AND\_mem\_op\_UX03H\_u1n44\_Inst[inst], ck\_grid\_u1n22, 1'b1, valid\_AND\_mem\_op\_uX03H\_Inst[inst],

csr\_pwrdn\_ovrd\_unnnH)

`MAKE\_CLK\_LOC(ck\_valid\_AND\_mem\_op\_UX04H\_u1n44\_Inst[inst], ck\_grid\_u1n22, 1'b1, valid\_AND\_mem\_op\_uX04H\_Inst[inst],

csr\_pwrdn\_ovrd\_unnnH)

end

//raw\_inst assigned under $valid gating condition

node raw\_inst\_UX01H\_Inst[28:0][3:0];

node raw\_inst\_UX02H\_Inst[15:0][3:0];

node raw\_inst\_UX03H\_Inst[15:0][3:0];

for(int inst = 0; inst < 4; inst++)

begin

`MSFF(raw\_inst\_UX02H\_Inst[15:0][inst], raw\_inst\_UX01H\_Inst[15:0][inst], ck\_valid\_UX01H\_u1n44\_Inst[inst])

`MSFF(raw\_inst\_UX03H\_Inst[15:0][inst], raw\_inst\_UX02H\_Inst[15:0][inst], ck\_valid\_UX02H\_u1n44\_Inst[inst])

end

//mem\_addr assigned under $mem\_op AND $valid gating conditions

node mem\_addr\_UX03H\_Inst[50:0][3:0];

node mem\_addr\_UX04H\_Inst[50:0][3:0];

node mem\_addr\_UX05H\_Inst[50:0][3:0];

for(int inst = 0; inst < 4; inst++)

begin

`MSFF(mem\_addr\_UX04H\_Inst[50:0][inst], mem\_addr\_UX03H\_Inst[50:0][inst], ck\_valid\_AND\_mem\_op\_UX03H\_u1n44\_Inst[inst])

`MSFF(mem\_addr\_UX05H\_Inst[50:0][inst], mem\_addr\_UX04H\_Inst[50:0][inst], ck\_valid\_AND\_mem\_op\_UX04H\_u1n44\_Inst[inst])

end

//sigB

node sigB\_UX04H\_Inst[3:0];

//sigA

node sigA\_UX05H\_Inst[3:0];

//op\_a

node op\_a\_UX03H\_Inst[3:0];

Rules:

Pipe signals are gated at their assignment.

Flow

1. parseFile() – generate Parse Structure
2. createLogicalStructure() – generate Logical Structure(PipeSignals, Used/Assigned Ranges, gated clocks). Line for line translation from SVX to SV.
3. generateSV() – \_declaration.vs file which will include all node declaration, transition flops/latches, and LCBs.

Scope

Every signal name should be treated as unique within a pipeline.

Signal assigned under “?when” scope, should only be used within that scope, or else you would be risking propagation of invalid transactions.