



Fiz**Zim** – an open-source FSM design environment

Paul Zimmer

Zimmer Design Services

(paulzimmer@zimmerdesignservices.com)

Michael Zimmer

Zimmer Design Services

(and University of California, Santa Barbara)

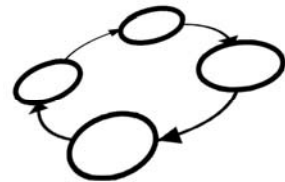
Brian Zimmer

Zimmer Design Services

Zimmertech

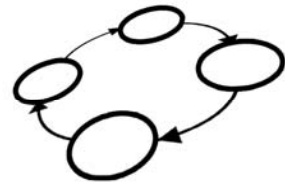
(and University of California, Davis)

- FSMs are a common design task.
- Many designers use “bubbles and arrows” and translate to Verilog.
- It would be nice to have the translation done automatically.
- Various attempts by EDA companies have generally failed.
- Good fit for an open-source project:
FIZZIM !



FIZZIM

The FSM Design Tool



FIZZIM

The FSM Design Tool

[^]
Free!

- Designers:
Mike Zimmer – GUI design and implementation
Brian Zimmer – GUI design and web work
Paul Zimmer – Backend (Verilog generator)
- Gui is in Java, backend is in perl.

1. Intro – attributes, gui, etc
2. Cliff's Classic FSM
3. HEROS output
4. ONEHOT output
5. Mealy outputs
6. Datapath outputs
7. Transition priority
8. Conclusion

To keep gui independent of backend, virtually everything is implemented as “attributes”

Attributes can be set on the FSM itself, on states, and on transitions.

State and transition attributes must be created globally before they can be set on specific state/transitions.

Gui knows about a few special attributes (transition equation, state name) just to make the output prettier.

New backend features can easily be implemented just by creating and setting attributes.

Each attribute has 5 fields:

- Attribute Name – this is the name of the input or output, or the name of the special attribute.
- Default Value – Default value of the attribute. Will be used if no value is assigned in a state/transition.
- Visibility – Turns on/off visibility on the display. “Only non-default” means to only show the attribute if its value doesn’t match “Default Value”.
- Type – Information about the attribute. Inputs currently have no defined type, outputs can be “reg”, “regdp”, or “comb”. Others are attribute-specific.
- Comment – An optional comment that will show up on the diagram, in the Verilog, both, or neither, depending on the attribute.
- Color – Text color.

Encodings:

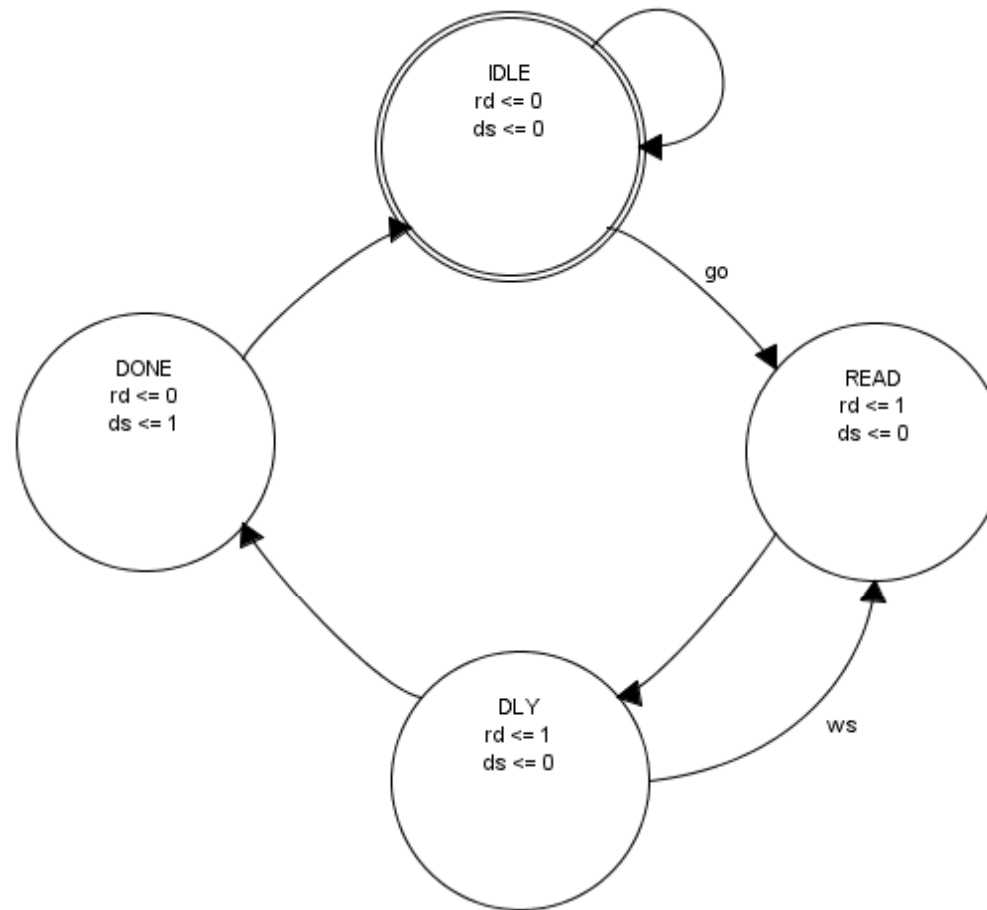
- HEROS - Highly Encoded with Registered Outputs as Statebits

(Verilog format based on Cliff Cumming's Paper "Coding And Scripting Techniques For FSM Designs With Synthesis-Optimized, Glitch-Free Outputs")

- Onehot

(Verilog format based on Steve Golson's paper "State machine design techniques for Verilog and VHDL")

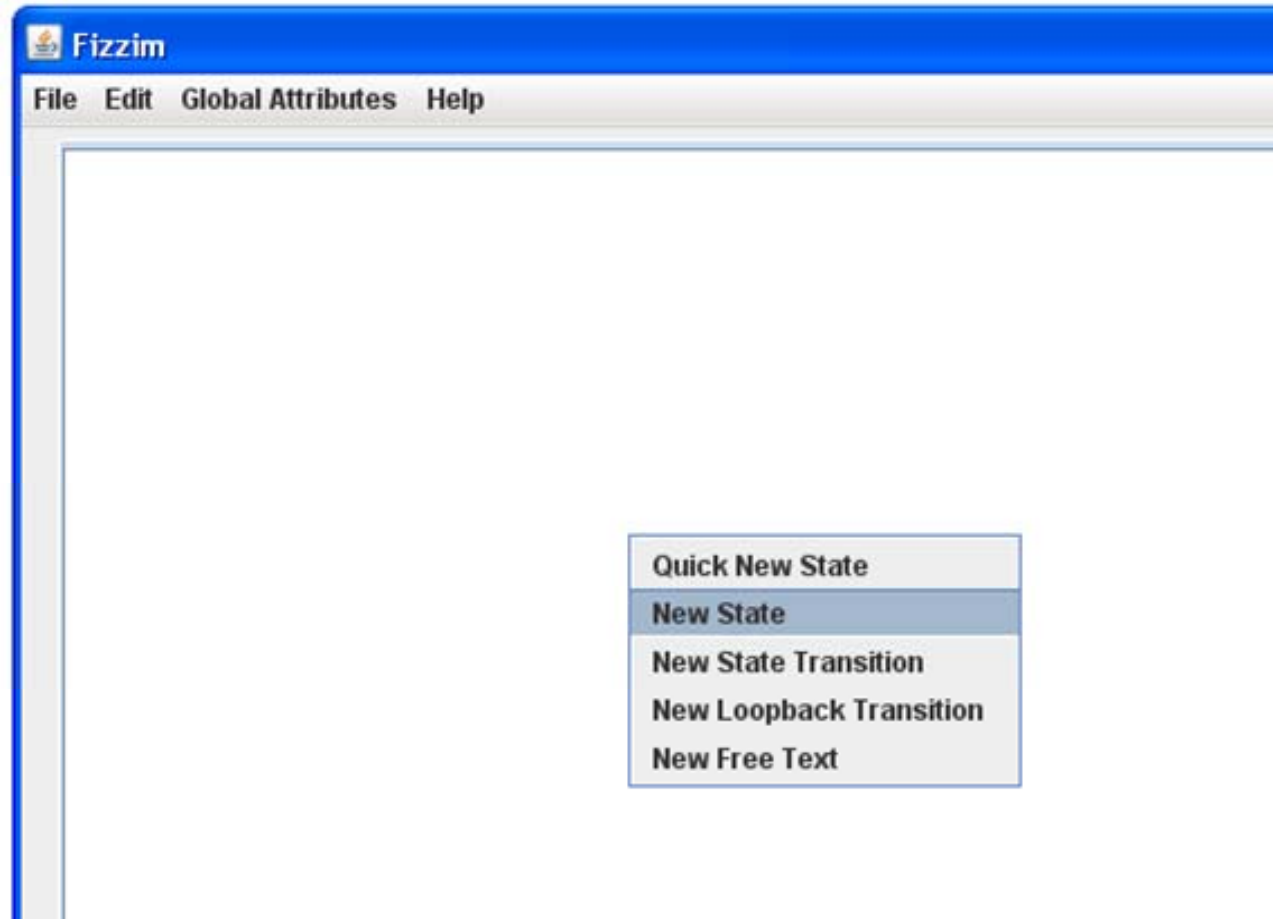
Cliff's Classic



Starting fizzim gui

- Fizzim is written in Java and runs under the Java Runtime Environment.
- Version should be 1.5 (“Java 5”) or greater, but there is an unsupported 1.4 version on the web site if you just want to try it out.
- Java RE is already loaded on most machines.
- If the correct file association is set up (.jar), you can just double-click the icon.
- Otherwise, you can launch from the command line:
> java -jar fizzim.jar

Cliff's Classic



Edit State Properties [X]

Edit the properties of the selected state:

Attribute Name	Value	Visibility	Type	Comment	Color
name	state0	Yes	def_type		

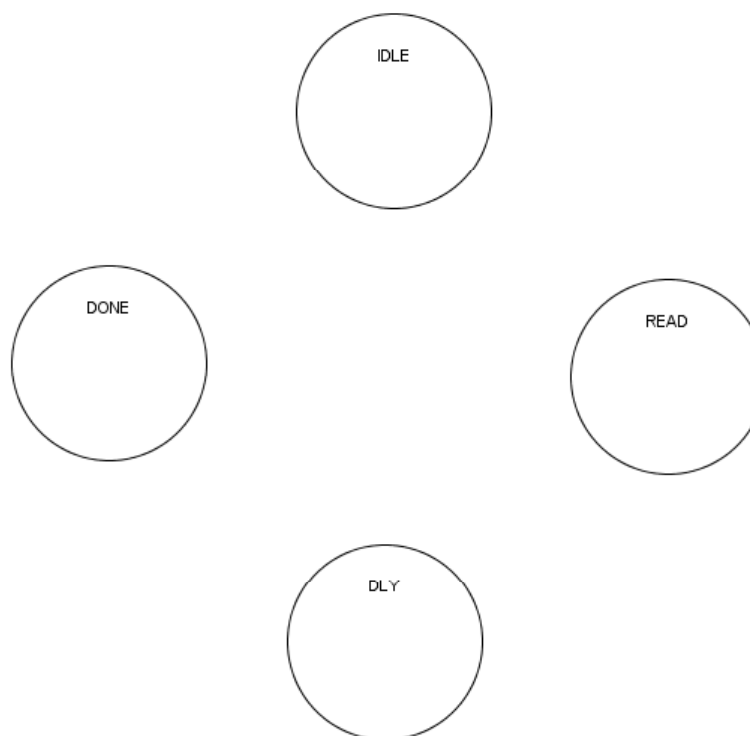
Width:

Height:

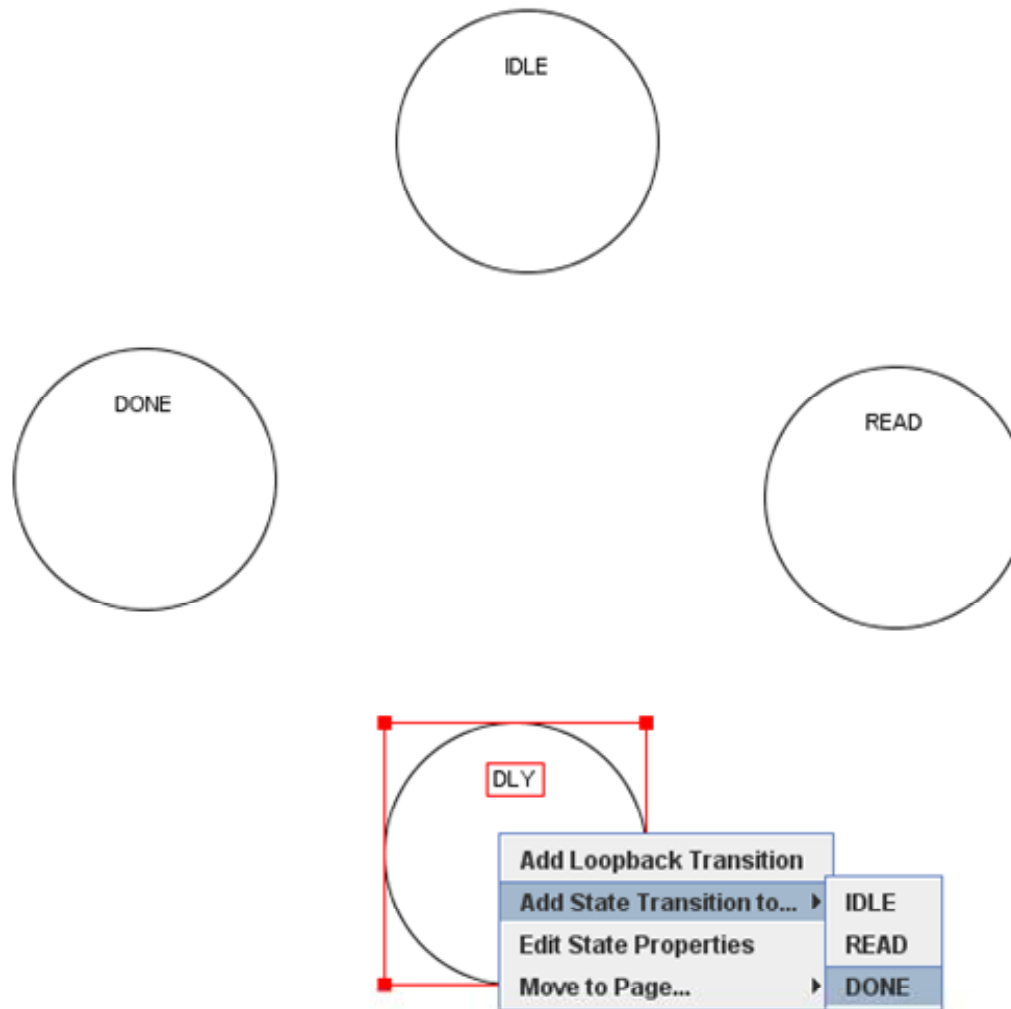
OK Cancel

Cliff's Classic

STATE MACHINE
name def_name
clock clk posedge
TRANSITIONS
equation 1 def_type

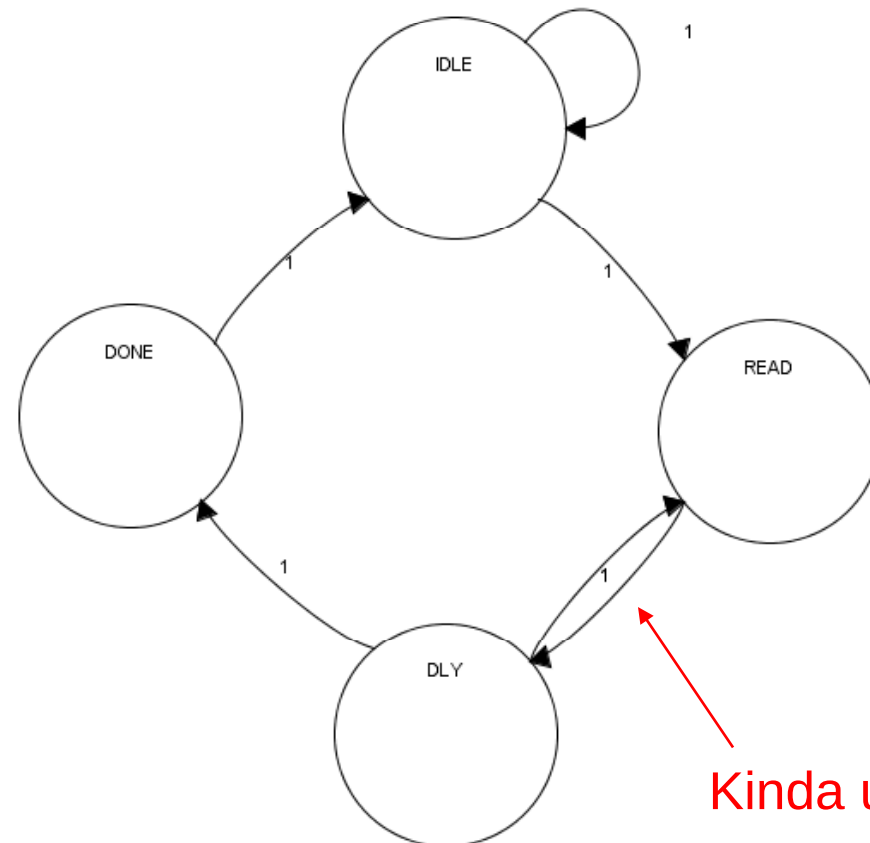


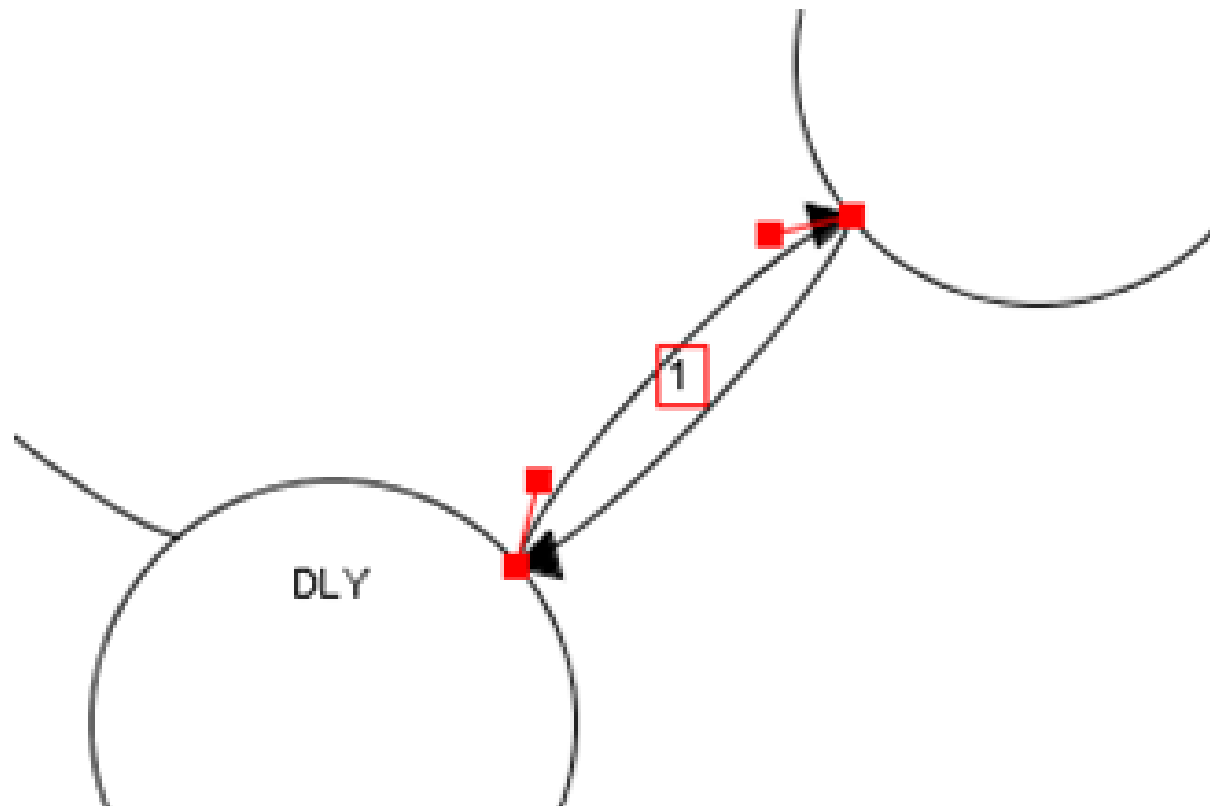
Cliff's Classic

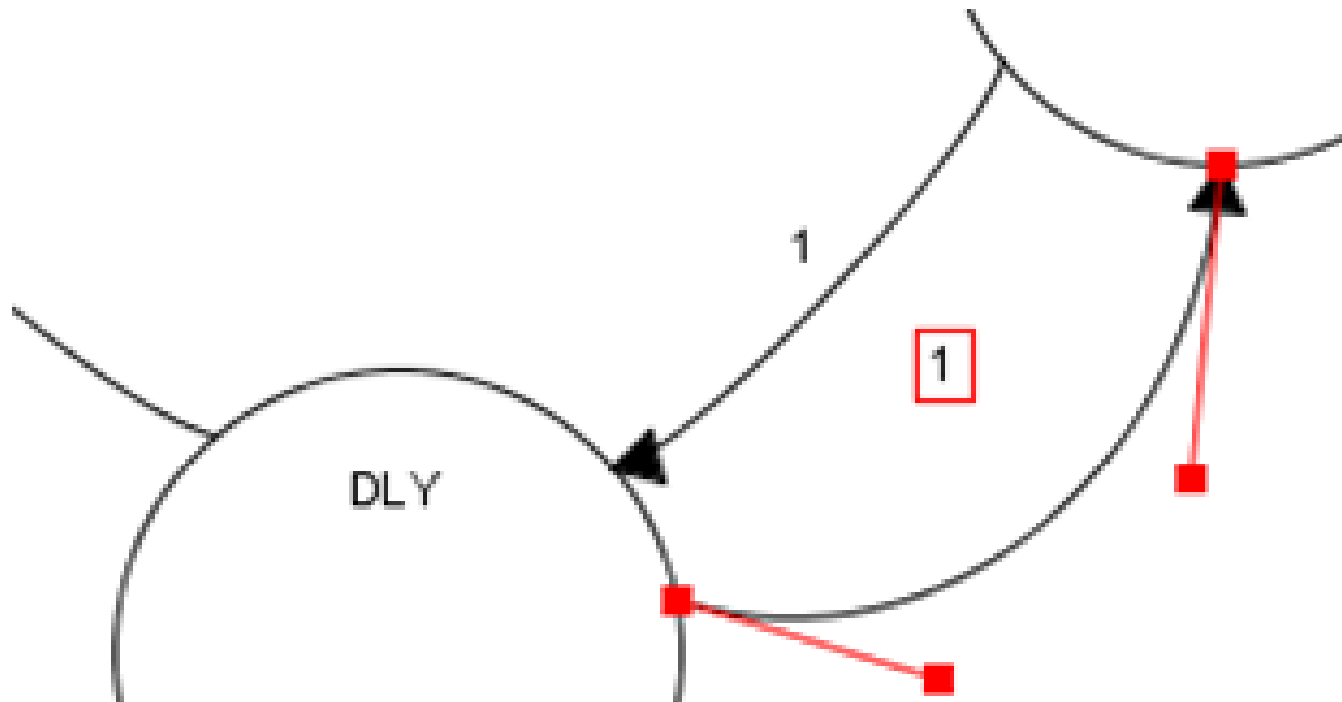


Cliff's Classic

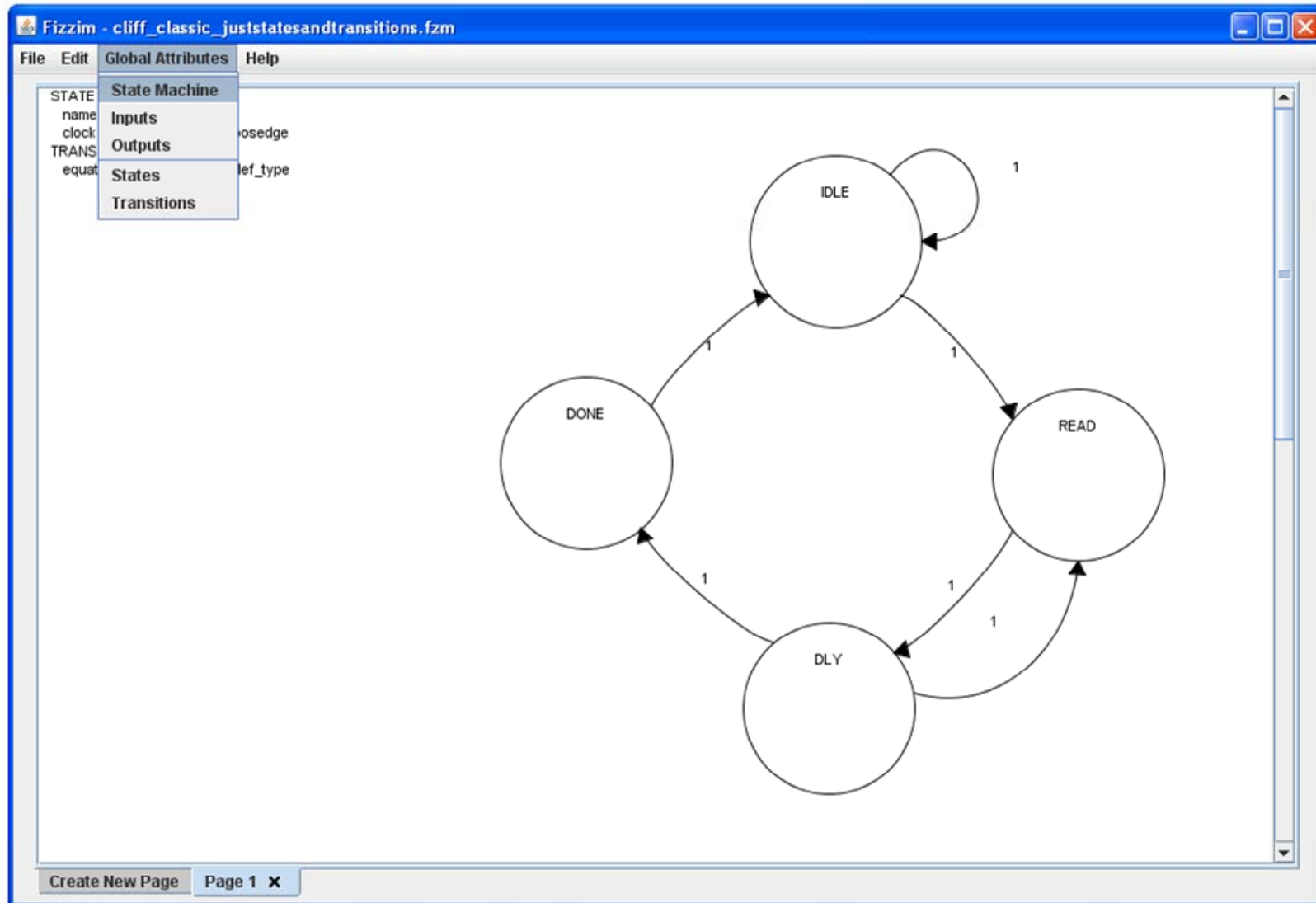
STATE MACHINE
name def_name
clock clk posedge
TRANSITIONS
equation 1 def_type







Cliff's Classic



Edit Global Properties

Here you can change the global attributes of all objects. Once an attribute is added, its default value can be overridden by right clicking on an object and selecting to 'Edit Properties.'

State Machine					
Inputs					
Outputs					
States					
Transitions					
Attribute Name	Default Value	Visibility	Type	Comment	Color
name	def_name	No			
clock	clk	No	posedge		

Edit Global Properties [X]

Here you can change the global attributes of all objects. Once an attribute is added, its default value can be overridden by right clicking on an object and selecting to 'Edit Properties.'

State Machine					
Attribute Name	Default Value	Visibility	Type	Comment	Color
name	cliff	No			
clock	clk	No	posedge		
reset_signal	rst_n	No	negedge		
reset_state	IDLE	No	anyvalue		

Delete User Reset

OK Cancel

×

Edit Global Properties

Here you can change the global attributes of all objects. Once an attribute is added, its default value can be overridden by right clicking on an object and selecting to 'Edit Properties.'

State Machine

Inputs

Outputs

States

Transitions

Attribute Name	Default Value	Visibility	Type	Comment	Color
clk		No			
rst_n		No			
go		No			
ws		No			

Delete

User

Input

Multibit Input

OK

Cancel

×

Edit Global Properties

Here you can change the global attributes of all objects. Once an attribute is added, its default value can be overridden by right clicking on an object and selecting to 'Edit Properties.'

State Machine

Inputs

Outputs

States

Transitions

Attribute Name	Default Value	Visibility	Type	Comment	Color
rd	0	Yes	reg		
ds	0	Yes	reg		

Delete

User

Output

Multibit Output

OK

Cancel

Edit Global Properties

Here you can change the global attributes of all objects. Once an attribute is added, its default value can be overridden by right clicking on an object and selecting to 'Edit Properties.'

State Machine

Inputs

Outputs

States

Transitions

Attribute Name	Default Value	Visibility	Type	Comment	Color
name	def_name	Yes	def_type		
rd	0	Yes	output		
ds	0	Yes	output		

Delete

User

OK

Cancel

×

Edit Global Properties

Here you can change the global attributes of all objects. Once an attribute is added, its default value can be overridden by right clicking on an object and selecting to 'Edit Properties.'

State Machine

Inputs

Outputs

States

Transitions

Attribute Name	Default Value	Visibility	Type	Comment	Color
name	def_name	No	def_type		
equation	1	Only non-def...	def_type		

Delete

User

Graycode

Output

Priority

OK

Cancel

Cliff's Classic

STATE MACHINE

name	cliff	
clock	clk	posedge
reset_signal	rst_n	negedge
reset_state	IDLE	anyvalue

INPUTS

clk	
rst_n	
go	
ws	

OUTPUTS

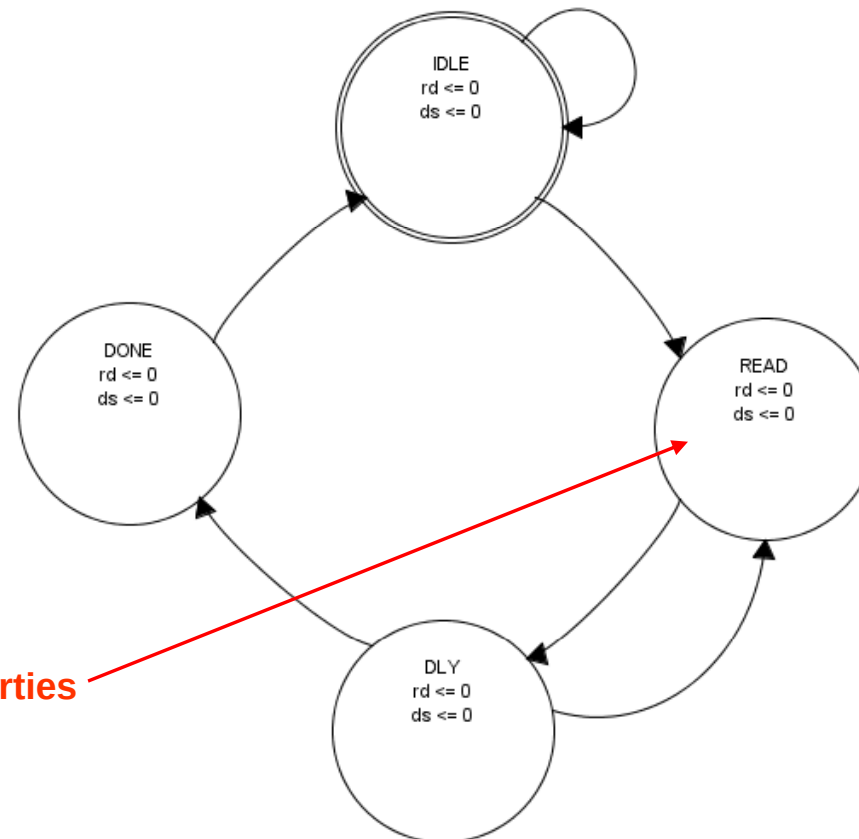
rd	0	reg
ds	0	reg

STATES

rd	0	output
ds	0	output

TRANSITIONS

equation	1	def_type
----------	---	----------



Double-click READ state to edit properties

Edit State Properties [X]

Edit the properties of the selected state:

Attribute Name	Value	Visibility	Type	Comment	Color
name	READ	Yes	def_type		
rd	1	Yes	output		
ds	0	Yes	output		

Width:

Height:

OK Cancel

Edit State Transition Properties [X]

Edit the properties of the selected state transition:

Attribute Name	Value	Visibility	Type	Comment	Color
name	trans1	No	def_type		
equation	go	Only non-default	def_type		

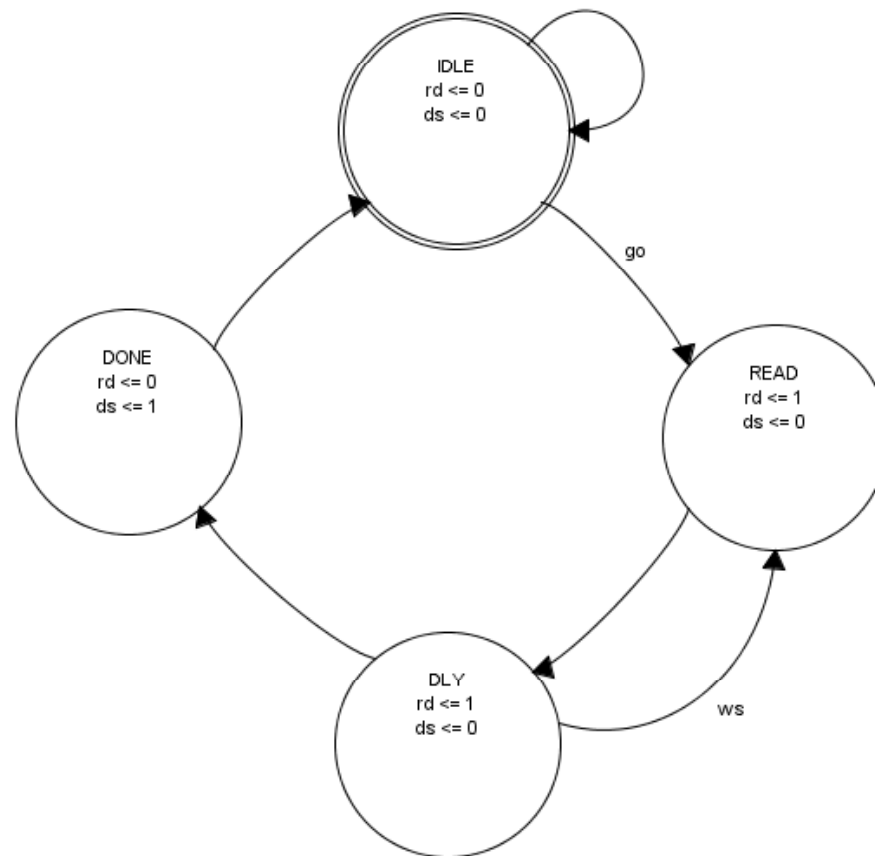
Start State: IDLE ▾ ☐ Stub?

End State: READ ▾

OK Cancel

Put in all the output values and transition equations:

STATE MACHINE
name cliff
clock clk posedge
reset_signal rst_n negedge
reset_state IDLE anyvalue
INPUTS
clk
rst_n
go
ws
OUTPUTS
rd 0 reg
ds 0 reg
STATES
rd 0 output
ds 0 output
TRANSITIONS
equation 1 def_type



Run fizzim.pl in heros mode:

```
fizzim.pl < cliff.fzm > cliff.v
```

```
module cliff (  
    output wire ds,  
    output wire rd,  
    input wire clk,  
    input wire go,  
    input wire rst_n,  
    input wire ws );
```

State bit encoding:

```
// state bits
parameter
IDLE = 3'b000, // extra=0 rd=0 ds=0
DLY  = 3'b010, // extra=0 rd=1 ds=0
DONE = 3'b001, // extra=0 rd=0 ds=1
READ = 3'b110; // extra=1 rd=1 ds=0

reg [2:0] state;
reg [2:0] nextstate;
```

Combinational always block:

```
// comb always block
always @* begin
    // Warning: Neither implied_loopback
    nor default_state_is_x attribute is set on
    state machine - this could result in
    latches being inferred
    case (state)
        IDLE: begin
            if (go) begin
                nextstate = READ;
            end
            else begin
                nextstate = IDLE;
            end
        end
        DLY: begin
            if (ws) begin
                nextstate = READ;
            end
            else begin
                nextstate = DONE;
            end
        end
    end
end
```

```
DONE: begin
    begin
        nextstate = IDLE;
    end
end
READ: begin
    begin
        nextstate = DLY;
    end
end
endcase
end
```

What's this warning?

```
// Warning: Neither implied_loopback nor default_state_is_x attribute is set on  
state machine - this could result in latches being inferred
```

“case” statement is incomplete, will infer latches!

HEROS encoding

Edit Global Properties

Here you can change the global attributes of all objects. Once an attribute is added, its default value can be overridden by right clicking on an object and selecting to 'Edit Properties.'

State Machine

Inputs

Outputs

States

Transitions

Attribute Name	Default Value	Visibility	Type	Comment	Color
name	cliff	No			
clock	clk	No	posedge		
reset_signal	rst_n	No	negedge		
reset_state	IDLE	No	anyvalue		
default_state_i...	1	No			

Delete

User

Reset

OK

Cancel

With default_state_is_x = 1:

```
// comb always block
always @* begin
    nextstate = 3'bx; // default to x because default_state_is_x is set
    case (state)
        IDLE: begin
            . . .
```

With implied_loopback = 1:

```
// comb always block
always @* begin
    nextstate = state; // default to hold value because
implied_loopback is set
    case (state)
        IDLE: begin
            . . .
```

Continuing with the HEROS output:

```
// Assign reg'd outputs to state bits
assign ds = state[0];
assign rd = state[1];

// sequential always block
always @(posedge clk or negedge rst_n) begin
    if (!rst_n)
        state <= IDLE;
    else
        state <= nextstate;
end
```

Run fizzim.pl in onehot mode:

```
fizzim.pl -enc onehot < cliff.fzm > cliff.v
```

```
module cliff (  
    . . .  
  
    // state bits  
    parameter  
    IDLE = 0,  
    DLY  = 2,  
    DONE = 1,  
    READ = 3;  
  
    reg [3:0] state;  
    reg [3:0] nextstate;
```

Combinational always block:

```
// comb always block
always @* begin
    nextstate = 4'b0000;
    case (1'b1) // synopsys parallel_case
full_case
        state[IDLE]: begin
            if (go) begin
                nextstate[READ] = 1'b1;
            end
            else begin
                nextstate[IDLE] = 1'b1;
            end
        end
        state[DLY]: begin
            if (ws) begin
                nextstate[READ] = 1'b1;
            end
            else begin
                nextstate[DONE] = 1'b1;
            end
        end
    end
end
```

```
state[DONE]: begin
    begin
        nextstate[IDLE] = 1'b1;
    end
end
state[READ]: begin
    begin
        nextstate[DLY] = 1'b1;
    end
end
endcase
end
```

Sequential always block:

```
// sequential always block
always @(posedge clk or negedge rst_n) begin
    if (!rst_n)
        state <= 4'b0001 << IDLE;
    else
        state <= nextstate;
end
```


ONEHOT encoding

NEW sequential always block:

```
// datapath sequential always block
always @(posedge clk
or negedge rst_n) begin
    if (!rst_n) begin
        ds <= 0;
        rd <= 0;
    end
    else begin
        case (1'b1)
            nextstate[IDLE]: begin
                ds <= 0;
                rd <= 0;
            end
            nextstate[DLY]: begin
                ds <= 0;
                rd <= 1;
            end
        endcase
    end
end
```

```
nextstate[DONE]: begin
    ds <= 1;
    rd <= 0;
end
nextstate[READ]: begin
    ds <= 0;
    rd <= 1;
end
endcase
end
end
```

Simulation code:

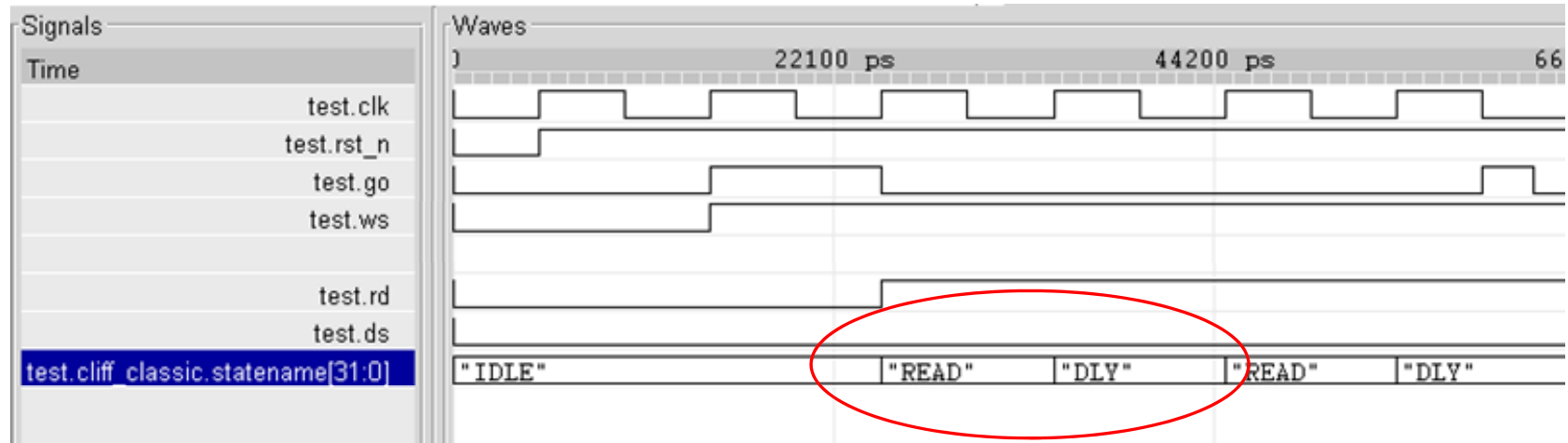
HEROS:

```
// This code allows you to see
state names in simulation
`ifndef SYNTHESIS
reg [31:0] statename;
always @* begin
    case (state)
        IDLE:
            statename = "IDLE";
        DLY:
            statename = "DLY";
        DONE:
            statename = "DONE";
        READ:
            statename = "READ";
        default:
            statename = "XXXX";
    endcase
end
`endif
```

ONEHOT:

```
// This code allows you to see
state names in simulation
`ifndef SYNTHESIS
reg [31:0] statename;
always @* begin
    case (1)
        state[IDLE]:
            statename = "IDLE";
        state[DLY]:
            statename = "DLY";
        state[DONE]:
            statename = "DONE";
        state[READ]:
            statename = "READ";
        default:
            statename = "XXXX";
    endcase
end
`endif
```

Simulation Code



1. Intro – attributes, gui, etc
2. Cliff's Classic FSM
3. HEROS output
4. ONEHOT output
5. Mealy outputs
6. Datapath outputs
7. Transition priority
8. Conclusion

Two ways to describe a mealy output

1. As a function of the inputs on each state
2. As a function of the inputs on each transition

Fizzim supports both!

Mealy on-states

STATE MACHINE

name	cliff_classic	
clock	clk	posedge
reset_signal	rst_n	negedge
reset_state	IDLE	anyvalue
default_state_is_x	1	

INPUTS

clk		
rst_n		
go		
ws		

OUTPUTS

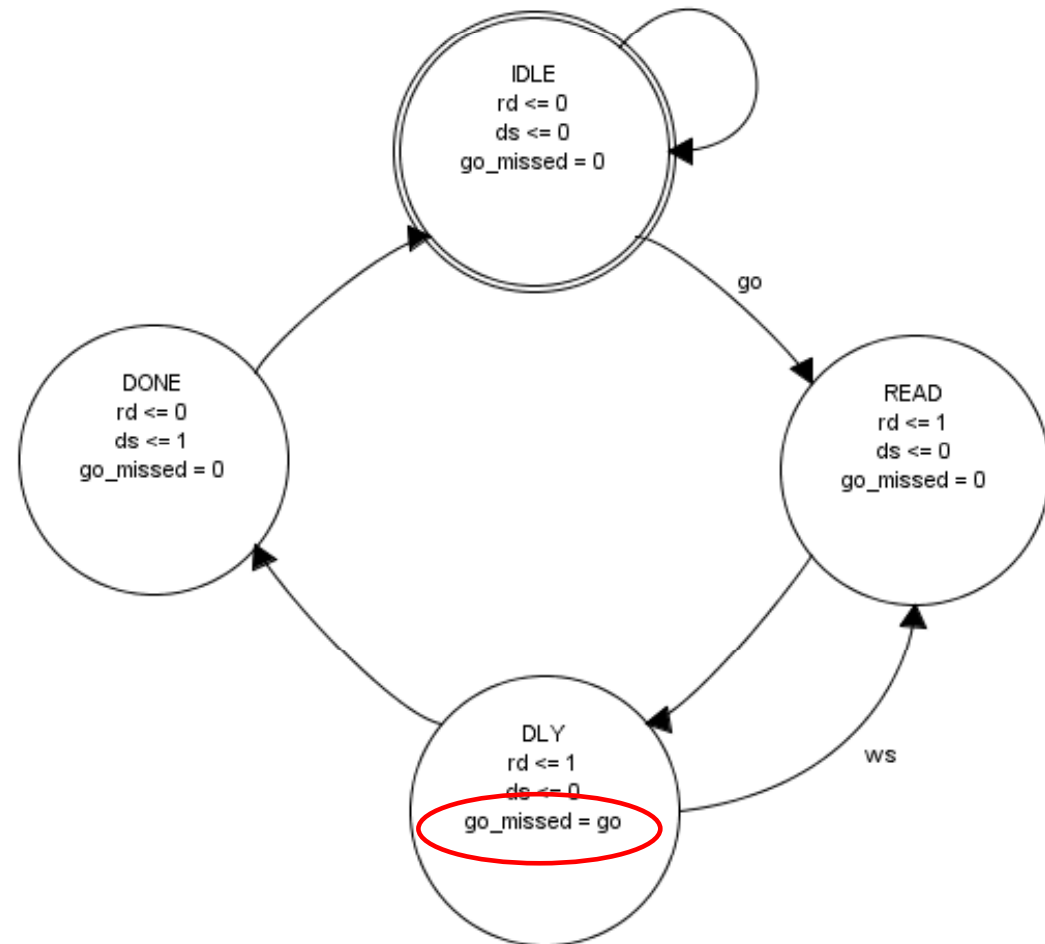
rd	0	reg
ds	0	reg
go_missed	0	comb

STATES

rd	0	output
ds	0	output
go_missed	0	output

TRANSITIONS

equation	1	def_type
----------	---	----------



Mealy on-states

Edit Global Properties

Here you can change the global attributes of all objects. Once an attribute is added, its default value can be overridden by right clicking on an object and selecting to 'Edit Properties.'

State Machine	Inputs	Outputs	States	Transitions	
Attribute Name	Default Value	Visibility	Type	Comment	Color
rd	0	Yes	reg		
ds	0	Yes	reg		
go_missed	0	Yes	comb		

Delete User Output Multibit Output

OK Cancel

Mealy on-states

Edit State Properties

Edit the properties of the selected state:

Attribute Name	Value	Visibility	Type	Comment	Color
name	DLY	Yes	def_type		
rd	1	Yes	output		
ds	0	Yes	output		
go_missed	go	Yes	output		

Width:

Height:

OK Cancel

HEROS output:

```
// comb always block
always @* begin
    nextstate = 3'bxxx; // default to x
    because default_state_is x is set
    go_missed = go_missed; // default to
    hold value to avoid latch inference
    case (state)
        IDLE: begin
            go_missed = 0;
            if (go) begin
                nextstate = READ;
            end
            else begin
                nextstate = IDLE;
            end
        end
        DLY: begin
            go_missed = go;
            if (ws) begin
                nextstate = READ;
            end
            else begin
                nextstate = DONE;
            end
        end
    endcase
end
```

```
DONE: begin
    go_missed = 0;
    begin
        nextstate = IDLE;
    end
end
READ: begin
    go_missed = 0;
    begin
        nextstate = DLY;
    end
end
endcase
end
```

Mealy on-transitions

STATE MACHINE

name	cliff_classic	
clock	clk	posedge
reset_signal	rst_n	negedge
reset_state	IDLE	anyvalue
default_state_is_x	1	

INPUTS

clk		
rst_n		
go		
ws		

OUTPUTS

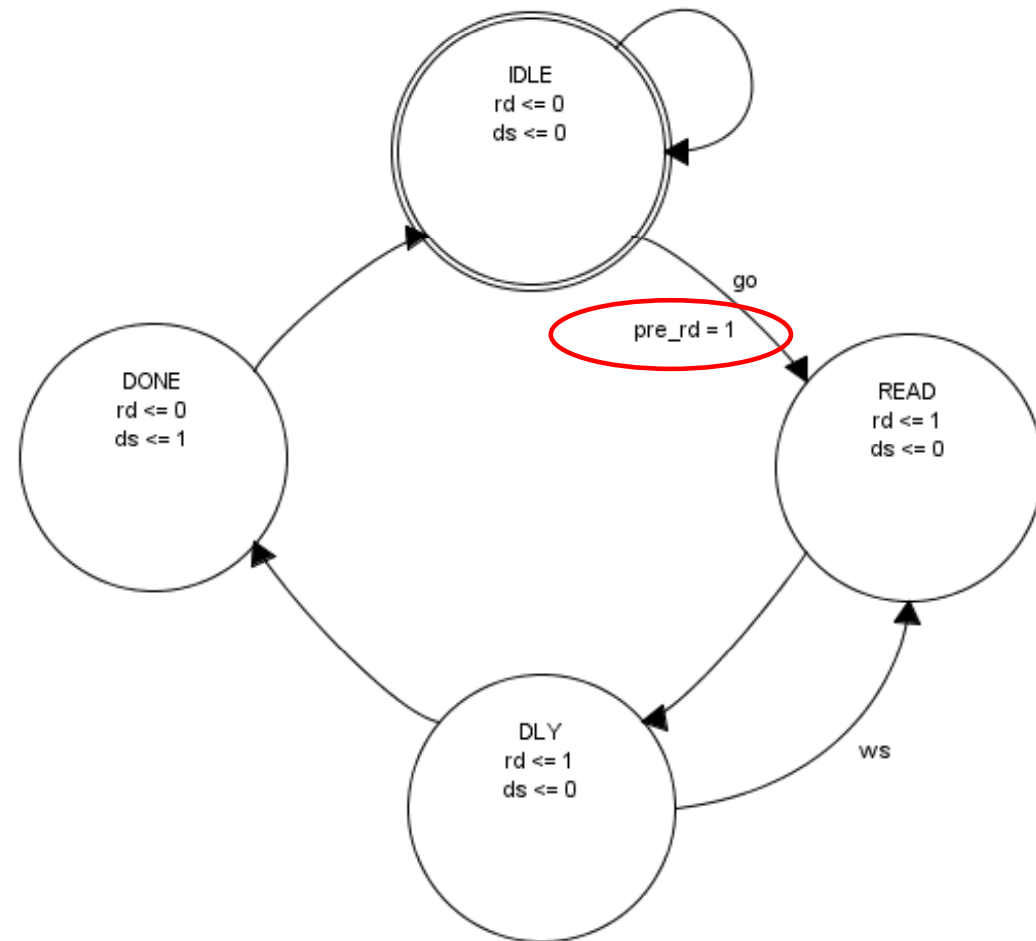
rd	0	reg
ds	0	reg
pre_rd	0	comb

STATES

rd	0	output
ds	0	output
pre_rd	0	output

TRANSITIONS

equation	1	def_type
pre_rd	0	output



Mealy on-transitions

Edit Global Properties

Here you can change the global attributes of all objects. Once an attribute is added, its default value can be overridden by right clicking on an object and selecting to 'Edit Properties.'

State Machine	Inputs	Outputs	States	Transitions	
Attribute Name	Default Value	Visibility	Type	Comment	Color
rd	0	Yes	reg		
ds	0	Yes	reg		
pre_rd	0	Only non-default	comb		

Mealy on-transitions

Edit Global Properties

Here you can change the global attributes of all objects. Once an attribute is added, its default value can be overridden by right clicking on an object and selecting to 'Edit Properties.'

State Machine Inputs Outputs States **Transitions**

Attribute Name	Default Value	Visibility	Type	Comment	Color
name	def_name	No	def_type		
equation	1	Only non-def...	def_type		
pre_rd	0	Only non-def...	output		

Delete User Graycode **Output** Priority

OK Cancel

Mealy on-transitions

HEROS output:

```
// comb always block
always @* begin
    nextstate = 3'bxxx; // default to x
    because default_state_is_x is set
    pre_rd = pre_rd; // default to hold
    value to avoid latch inference
    case (state)
        IDLE: begin
            if (go) begin
                nextstate = READ;
                pre_rd = 1;
            end
            else begin
                nextstate = IDLE;
                pre_rd = 0;
            end
        end
        DLY: begin
            if (ws) begin
                nextstate = READ;
                pre_rd = 0;
            end
            else begin
                nextstate = DONE;
                pre_rd = 0;
            end
        end
    end
end
```

```
DONE: begin
    begin
        nextstate = IDLE;
        pre_rd = 0;
    end
end
READ: begin
    begin
        nextstate = DLY;
        pre_rd = 0;
    end
end
endcase
end
```

You *can* mix the styles. Order of priority:

1. Non-default value on a transition.
2. Non-default value on a state.
3. Default value on a transition.

See the tutorial for more details.

1. Intro – attributes, gui, etc
2. Cliff's Classic FSM
3. HEROS output
4. ONEHOT output
5. Mealy outputs
6. Datapath outputs
7. Transition priority
8. Conclusion

Fizzim allows registered outputs that are *not* part of the state vector.

All “reg” or “regdp” in onehot.

Only “regdp” in heros.

Datapath outputs

Edit Global Properties

Here you can change the global attributes of all objects. Once an attribute is added, its default value can be overridden by right clicking on an object and selecting to 'Edit Properties.'

State Machine Inputs **Outputs** States Transitions

Attribute Name	Default Value	Visibility	Type	Comment	Color
rd		Yes	regdp		
ds		Yes	reg		

Delete User Output Multibit Output

OK Cancel

State bit assignment now looks like this (heros):

```
// state bits
parameter
IDLE = 3'b000, // extra=00 ds=0
DLY  = 3'b010, // extra=10 ds=0
DONE = 3'b001, // extra=01 ds=1
READ = 3'b100; // extra=00 ds=0
```

New datapath sequential always block added (like onehot):

```
// datapath sequential always block
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        rd <= 0;
    end
    else begin
        case (nextstate)
            IDLE: begin
                rd <= 0;
            end
            DLY: begin
                rd <= 1;
            end
            DONE: begin
                rd <= 0;
            end
            READ: begin
                rd <= 1;
            end
        endcase
    end
end
```

Datapath outputs

STATE MACHINE

name	cliff_classic	
clock	clk	posedge
reset_signal	rst_n	negedge
reset_state	IDLE	anyvalue
default_state_is_x	1	

INPUTS

clk
rst_n
go
ws
load[8:0]

OUTPUTS

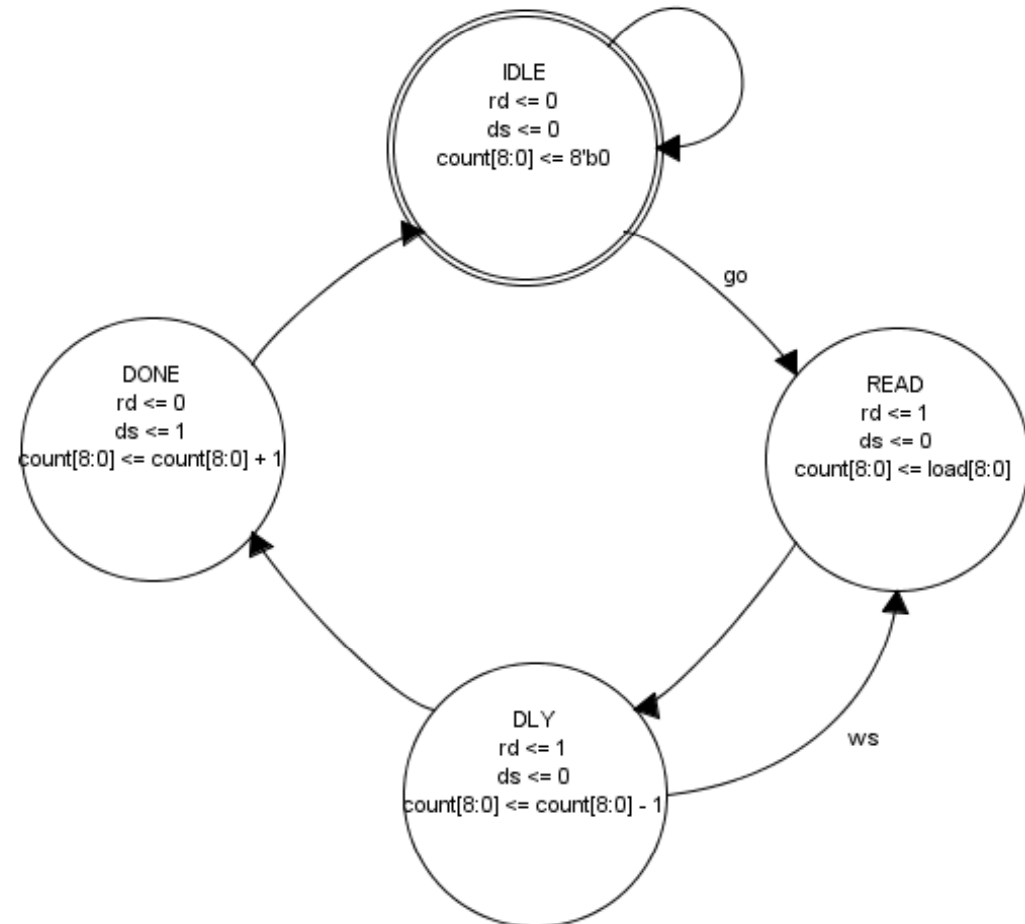
rd	0	reg
ds	0	reg
count[8:0]	8'b00000000	regdp

STATES

rd	0	output
ds	0	output
count[8:0]	8'b00000000	output

TRANSITIONS

equation	1	def_type
----------	---	----------



Datapath outputs

Edit Global Properties

Here you can change the global attributes of all objects. Once an attribute is added, its default value can be overridden by right clicking on an object and selecting to 'Edit Properties.'

State Machine	Inputs	Outputs	States	Transitions	
Attribute Name	Default Value	Visibility	Type	Comment	Color
rd	0	Yes	reg		
ds	0	Yes	reg		
count[8:0]		Yes	regdp		

Delete User Output Multibit Output

OK Cancel

Datapath outputs

Edit Global Properties

Here you can change the global attributes of all objects. Once an attribute is added, its default value can be overridden by right clicking on an object and selecting to 'Edit Properties.'

State Machine **Inputs** Outputs States Transitions

Attribute Name	Default Value	Visibility	Type	Comment	Color
clk		No			
rst_n		No			
go		No			
ws		No			
load[8:0]		No			

Delete User Input Multibit Input

OK Cancel

Datapath sequential always block:

```
// datapath sequential always block
always @(posedge clk or negedge rst_n) begin
  if (!rst_n) begin
    count[8:0] <= 8'b0;
  end
  else begin
    case (nextstate)
      IDLE: begin
        count[8:0] <= 8'b0;
      end
      DLY : begin
        count[8:0] <= count[8:0] - 1;
      end
      DONE: begin
        count[8:0] <= count[8:0] + 1;
      end
      READ: begin
        count[8:0] <= load[8:0];
      end
    endcase
  end
end
```

1. Intro – attributes, gui, etc
2. Cliff's Classic FSM
3. HEROS output
4. ONEHOT output
5. Mealy outputs
6. Datapath outputs
7. Transition priority
8. Conclusion

Transition Priority

STATE MACHINE

name	cliff_classic	
clock	clk	posedge
reset_signal	rst_n	negedge
reset_state	IDLE	anyvalue
default_state_is_x	1	

INPUTS

clk
rst_n
go
ws
test

OUTPUTS

rd
ds

reg
reg

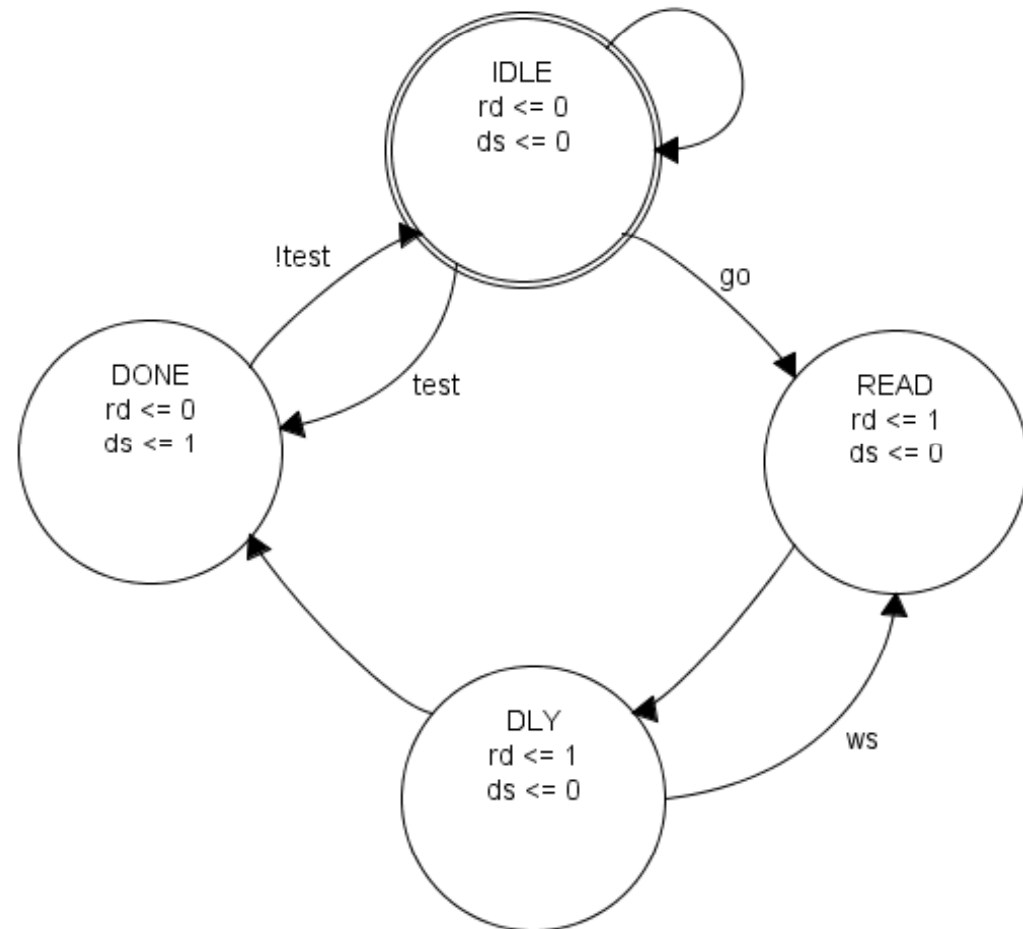
STATES

rd
ds

output
output

TRANSITIONS

equation	1	def_type
----------	---	----------



Make the appropriate edits, and run fizzim.pl:

```
IDLE: begin
// Warning P3: State IDLE has multiple exit transitions, and
transition trans0 has no defined priority
// Warning P3: State IDLE has multiple exit transitions, and
transition trans6 has no defined priority
```

What's that all about?

What is supposed to happen when both test and go are true?

Assume test has priority.

Could make the go trans equation “!test && go” ,
but this gets really messy really quickly.

If you were coding by hand, you would do this:

```
if (test) begin
    nextstate = DONE;
end
else if (go) begin
    nextstate = READ;
end
else begin
    nextstate = IDLE;
end
```

Transition priority allows you to emulate this!

Transition Priority

Edit Global Properties

Here you can change the global attributes of all objects. Once an attribute is added, its default value can be overridden by right clicking on an object and selecting to 'Edit Properties.'

State Machine Inputs Outputs States **Transitions**

Attribute Name	Default Value	Visibility	Type	Comment	Color
name	def_name	No	def_type		
equation	1	Only non-def...	def_type		
priority	1000	Only non-def...			

Delete User Graycode Output **Priority**

OK Cancel

Transition Priority

STATE MACHINE

name	cliff_classic	
clock	clk	posedge
reset_signal	rst_n	negedge
reset_state	IDLE	anyvalue
default_state_is_x	1	

INPUTS

clk		
rst_n		
go		
ws		
test		

OUTPUTS

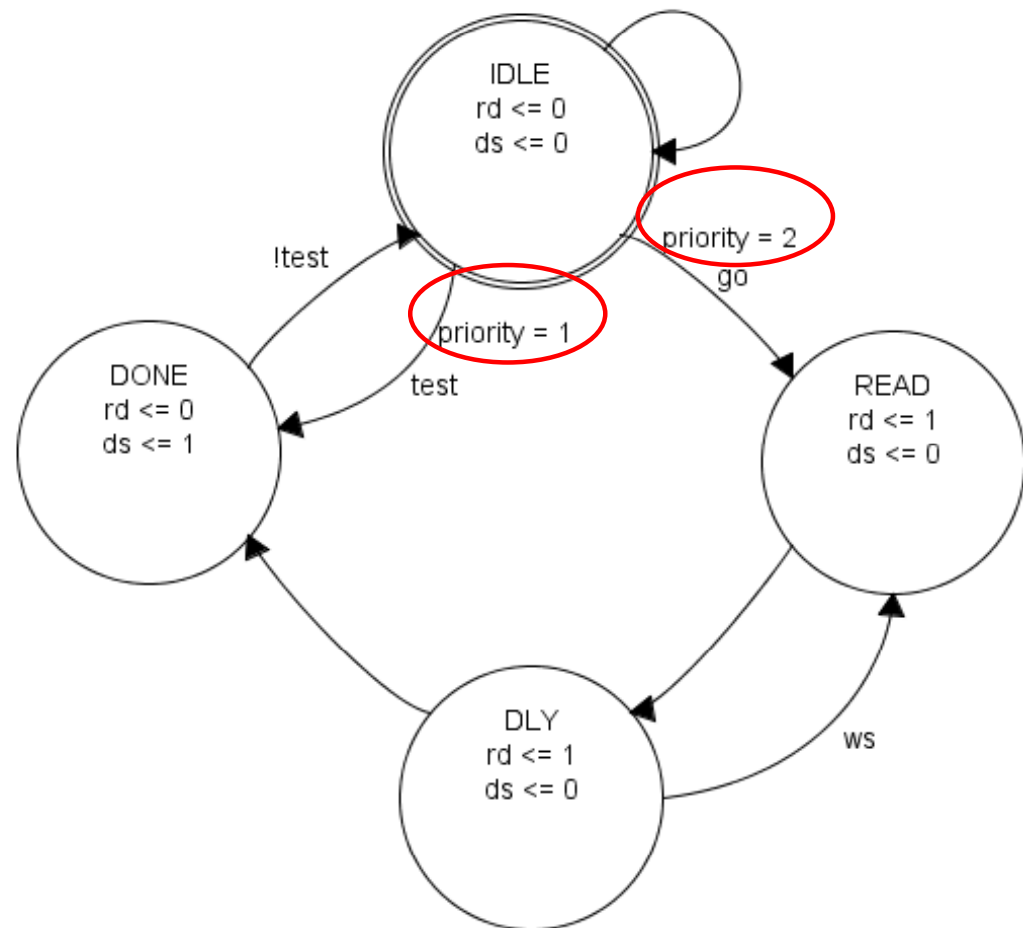
rd	0	reg
ds	0	reg

STATES

rd	0	output
ds	0	output

TRANSITIONS

equation	1	def_type
priority	1	



Run fizzim.pl and the IDLE transition block looks like this:

```
IDLE: begin
  if (test) begin
    nextstate = DONE;
  end
  else if (go) begin
    nextstate = READ;
  end
  else begin
    nextstate = IDLE;
  end
end
```

Just like you had coded it by hand!

Hey, wait a minute – how did this EVER work??

Before adding priorities, how did fizzim know that the loopback on IDLE was lowest priority?

How does it know that the DLY->DONE transition is lower priority than DLY->READ?

The special case of equation “1”

STATE MACHINE

name	cliff_classic	
clock	clk	posedge
reset_signal	rst_n	negedge
reset_state	IDLE	anyvalue
default_state_is_x	1	

INPUTS

clk		
rst_n		
go		
ws		

OUTPUTS

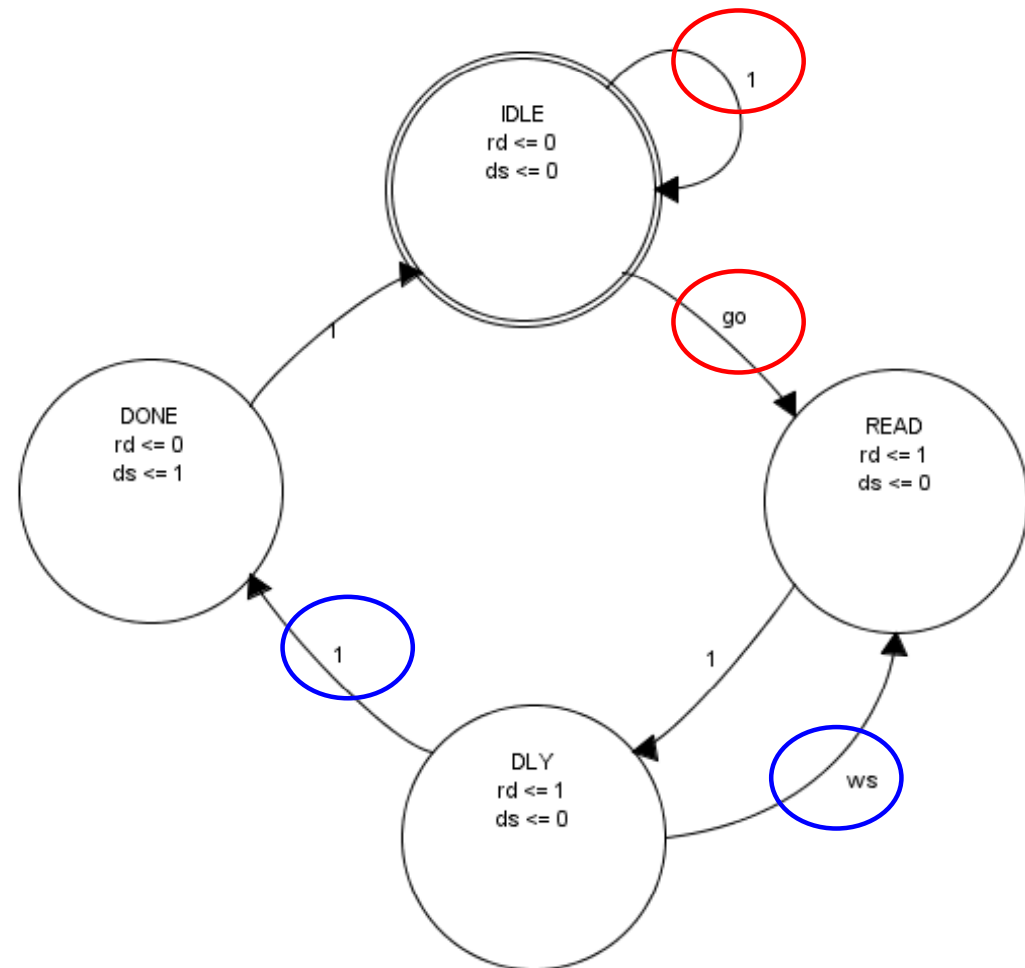
rd	0	reg
ds	0	reg

STATES

rd	0	output
ds	0	output

TRANSITIONS

equation	1	def_type
----------	---	----------



The special case of equation “1”



Fizzim.pl has special rules for equation equal to “1”:

1. If two exit transitions have the same (or no) priority set, the one with the always-true equation (“1”) is assumed to have lower priority, and no warning is issued.
2. If there are only two exit conditions and the always-true one is the lower priority (either due the rule above or because it has explicitly been set), no warning is issued.

The special case of equation “1”



Don't like priorities? You don't have to use them!

- Make sure all of your transitions have (non-1) equations.
- Suppress the P3 warnings by doing:
fizzim.pl –nowarn P3

1. Intro – attributes, gui, etc
2. Cliff's Classic FSM
3. HEROS output
4. ONEHOT output
5. Mealy outputs
6. Datapath outputs
7. Transition priority
8. Conclusion

Features not covered

- Gray codes (fizzim can do this automatically!)
- Stubs (short-hand transition arcs)
- Renaming/outputting internal signals (state, nextstate)
- Inserting code at strategic places using attributes (copyright statement, `include, etc)
- Multiple pages
- Comments (on the diagram and in the Verilog)

Features not covered (cont)



- Forcing the state vector
- Controlling and suppressing warning messages (individually or by group)
- Printing and exporting the state diagram.
- Group select and move.
- Specifying the backend command and options

Features **not implemented** (yet)



-
- Multi-page print
 - Better support for page sizes other than letter
 - (limited) parsing of ``include` files...
 - -terse option to output code with the “good Cliff-keeping seal of approval”

Thank you !