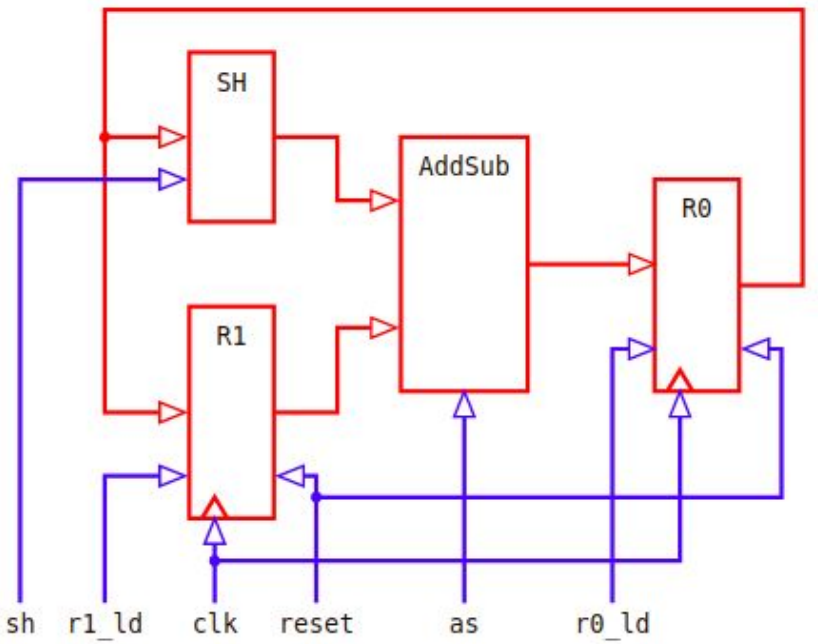
Design Of A Mini Data Path Part 2

Instructor: Dr. Vinicius Prado da Fonseca (vpradodafons@online.mun.ca)

Register Transfer Level (RTL) Design

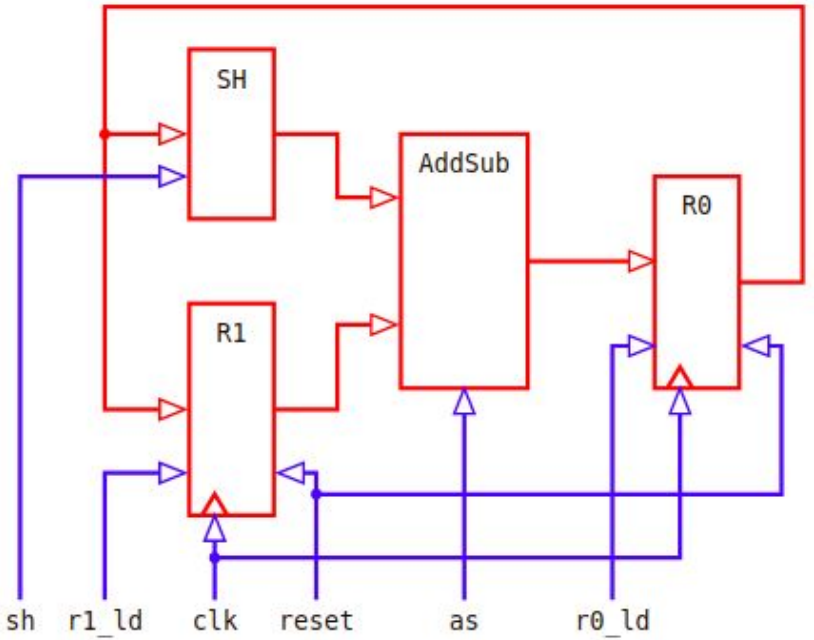
- The mini data path performs an operation
  - setting the control lines
  - and then providing a positive clock edge
- Can be examined at the logic level, to determine the effect of all the control inputs
  - i.e. analysis of control lines for every operation
- but in general it is better if the Register transfer level RTL is used



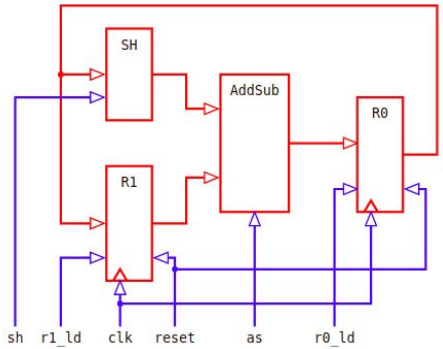
Register Transfer Level (RTL) Design

- RTL views the systems as the effect the control line has on the values of the registers after the clock goes positive
- Below some “Memory operations”

as_ctl	shift_ctl	r0_ld	r1_ld	RTL
AS_B	X	1	0	R0 <= R1
X	X	0	1	R1 <= R0
AS_B	X	1	1	R1 <= R0; R0 <= R1



Register Transfer Level (RTL) Design

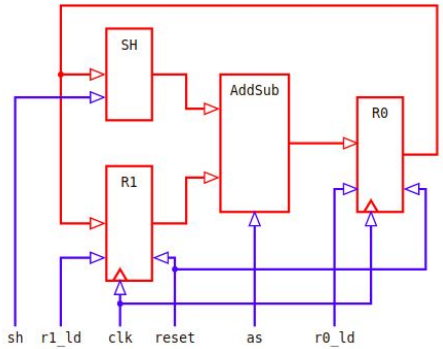


R0 <= R1

- Contents of R1 are copied to R0
- Add/subtract is set to only pass input B. Input A is ignored
- Since the A input is ignored the shifter is controlled is irrelevant
- SH control inputs can be considered as do not cares
- The R0 register is updated
  - r0_ld=1
- R1 is not updated since r1_ld=0.

as_ctl	shift_ctl	r0_ld	r1_ld	RTL
AS_B	X	1	0	R0 <= R1
X	X	0	1	R1 <= R0
AS_B	X	1	1	R1 <= R0; R0 <= R1

Register Transfer Level (RTL) Design

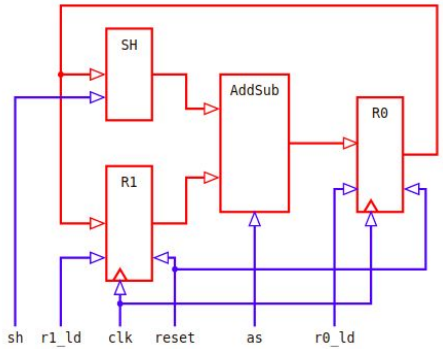


R1 <= R0

- R0 is copied to R1
- r1_ld=1 and r0_ld=0
- R0 is not updated
- Add/subtract control signals are do not cares.

as_ctl	shift_ctl	r0_ld	r1_ld	RTL
AS_B	X	1	0	R0 <= R1
X	X	0	1	R1 <= R0
AS_B	X	1	1	R1 <= R0; R0 <= R1

Register Transfer Level (RTL) Design



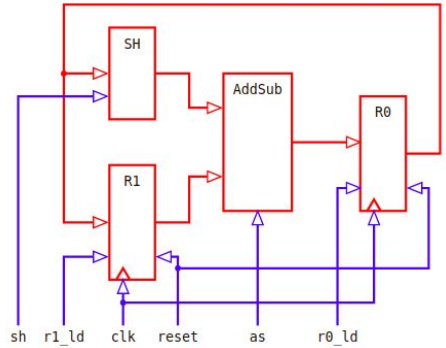
R1 <= R0; R0 <= R1

- r1_ld=1
  - R1 <= R0
- r0_ld=1 and as_ctl=AS_B
  - R0 <= R1
- Since the transfers happen at the same time
- The values in R0 and R1 are exchanged

as_ctl	shift_ctl	r0_ld	r1_ld	RTL
AS_B	X	1	0	R0 <= R1
X	X	0	1	R1 <= R0
AS_B	X	1	1	R1 <= R0; R0 <= R1

Register Transfer Level (RTL) Design

- The transfer of R1 <= R0 is independent of the rest of the data path
- “Data operations” happen only using AS, SH and R0
  - r1_ld is ignored here



as_ctl	shift_ctl	r0_ld	RTL
AS_ADD	SH_PASS	1	R0 <= R0 + R1
AS_ADD	SH_LEFT	1	R0 <= 2*R0 + R1
AS_SUB	SH_PASS	1	R0 <= R0 - R1
AS_SUB	SH_LEFT	1	R0 <= 2*R0 - R1
AS_A	SH_ONE	1	R0 <= 1
AS_A	SH_ONES	1	R0 <= -1
AS_A	SH_LEFT	1	R0 <= 2*R0

RTL Examples

Set R0 and R1 to 1

- R0 <= 1
  ctl?
- R1 <= R0
  ctl?

Two steps (clock cycles) are required, since R0 must contain 1, before it can be transferred to R1

as_ctl	shift_ctl	r0_ld	RTL
AS_ADD	SH_PASS	1	R0 <= R0 + R1
AS_ADD	SH_LEFT	1	R0 <= 2*R0 + R1
AS_SUB	SH_PASS	1	R0 <= R0 - R1
AS_SUB	SH_LEFT	1	R0 <= 2*R0 - R1
AS_A	SH_ONE	1	R0 <= 1
AS_A	SH_ONES	1	R0 <= -1
AS_A	SH_LEFT	1	R0 <= 2*R0

as_ctl	shift_ctl	r0_ld	r1_ld	RTL
AS_B	X	1	0	R0 <= R1
X	X	0	1	R1 <= R0
AS_B	X	1	1	R1 <= R0; R0 <= R1

RTL Examples

Set R0 and R1 to 1

- R0 <= 1
  - as_ctl=AS_A, shift_ctl=SH_ONE, r0_ld=1, r1_ld=0
- R1 <= R0
  - as_ctl=X, shift_ctl=X, r0_ld=0, r1_ld=1

Two steps (clock cycles) are required, since R0 must contain 1, before it can be transferred to R1

as_ctl	shift_ctl	r0_ld	RTL
AS_ADD	SH_PASS	1	R0 <= R0 + R1
AS_ADD	SH_LEFT	1	R0 <= 2*R0 + R1
AS_SUB	SH_PASS	1	R0 <= R0 - R1
AS_SUB	SH_LEFT	1	R0 <= 2*R0 - R1
AS_A	SH_ONE	1	R0 <= 1
AS_A	SH_ONES	1	R0 <= -1
AS_A	SH_LEFT	1	R0 <= 2*R0

as_ctl	shift_ctl	r0_ld	r1_ld	RTL
AS_B	X	1	0	R0 <= R1
X	X	0	1	R1 <= R0
AS_B	X	1	1	R1 <= R0; R0 <= R1

RTL Examples

Set R0 to 5

- R0 <= 1
- R1 <= R0
- R0 <= 2*R0
- R0 <= 2*R0 + R1
- HALT

as_ctl	shift_ctl	r0_ld	RTL
AS_ADD	SH_PASS	1	R0 <= R0 + R1
AS_ADD	SH_LEFT	1	R0 <= 2*R0 + R1
AS_SUB	SH_PASS	1	R0 <= R0 - R1
AS_SUB	SH_LEFT	1	R0 <= 2*R0 - R1
AS_A	SH_ONE	1	R0 <= 1
AS_A	SH_ONES	1	R0 <= -1
AS_A	SH_LEFT	1	R0 <= 2*R0

as_ctl	shift_ctl	r0_ld	r1_ld	RTL
AS_B	X	1	0	R0 <= R1
X	X	0	1	R1 <= R0
AS_B	X	1	1	R1 <= R0; R0 <= R1

RTL Examples

Set R0 to 5

- R0 <= 1
- R1 <= R0
- R0 <= 2*R0
- R0 <= 2*R0 + R1
- HALT

- After the first two steps R1 and R0 are 1
- The next step multiplies R0 by 2 making R0=2
- The last step multiplies R0 by 2 and then
- adds R1 which is 1, this gives 5
- Halt: r1_ld = 0; r0_ld = 0
- Example in the notes

RTL Examples

Set R0 to 11

- R0 <= 1
- R1 <= R0
- R0 <= 2*R0
- R0 <= 2*R0 + R1
- R0 <= 2*R0 + R1
- HALT

as_ctl	shift_ctl	r0_ld	RTL
AS_ADD	SH_PASS	1	R0 <= R0 + R1
AS_ADD	SH_LEFT	1	R0 <= 2*R0 + R1
AS_SUB	SH_PASS	1	R0 <= R0 - R1
AS_SUB	SH_LEFT	1	R0 <= 2*R0 - R1
AS_A	SH_ONE	1	R0 <= 1
AS_A	SH_ONES	1	R0 <= -1
AS_A	SH_LEFT	1	R0 <= 2*R0

as_ctl	shift_ctl	r0_ld	r1_ld	RTL
AS_B	X	1	0	R0 <= R1
X	X	0	1	R1 <= R0
AS_B	X	1	1	R1 <= R0; R0 <= R1

RTL Examples

Set R0 to 11

- R0 <= 1
- R1 <= R0
- R0 <= 2*R0
- R0 <= 2*R0 + R1
- R0 <= 2*R0 + R1
- HALT

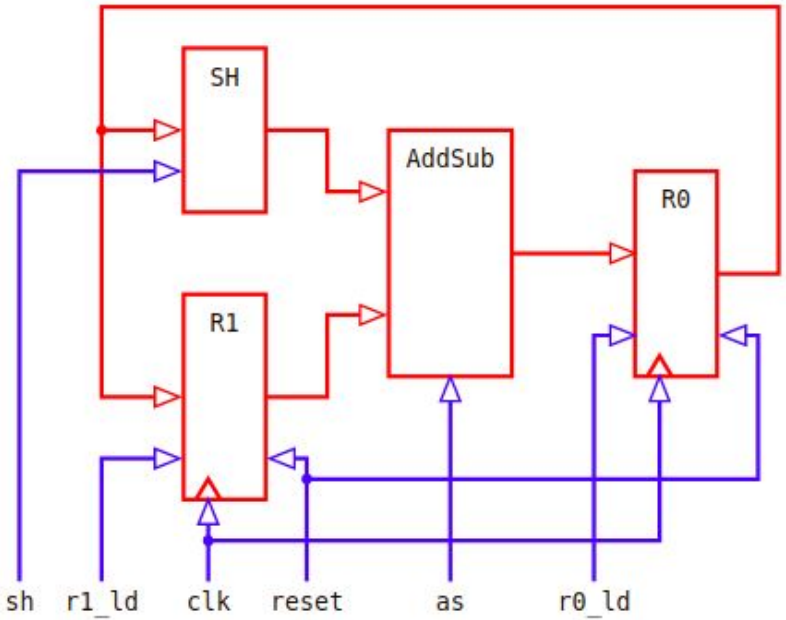
The first 4 steps sets R0 to 5, the last steps multiplies 5 by 2 and adds 1 yield 11. By following similar steps, R0 can be set to any value.

- Example in the notes

RTL Examples

Multiply R0 by 3 (example: start with 7 in R0)

- R1 <= R0; R0 <= 2*R0
- R0 <= R0 + R1



RTL Examples

Multiply R0 by 3 (example: start with 7 in R0)

- R1 <= R0; R0 <= 2*R0
- R0 <= R0 + R1

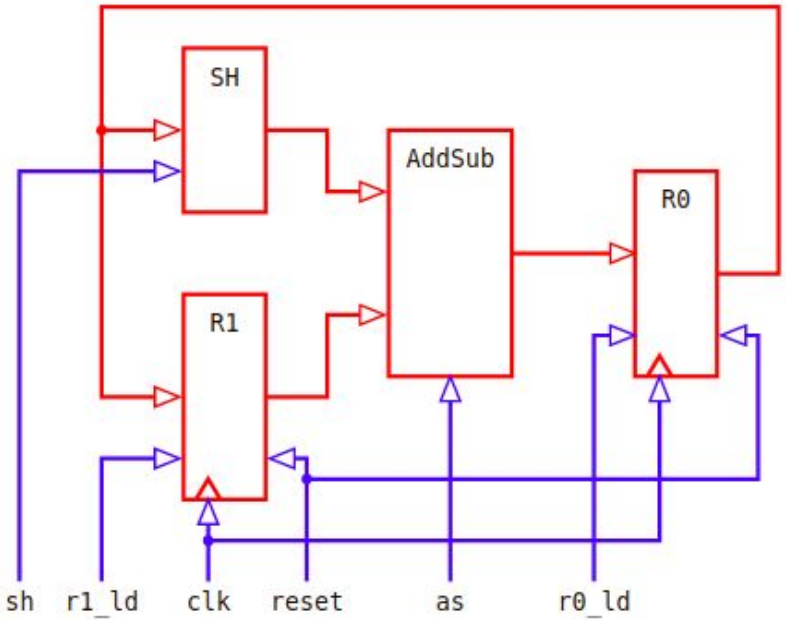
The first step copies R0 (e.g., R0 is saved in R1), and at the same step, R0 is multiplied by 2. The second step adds the original value of R0 to its value from step 1. This is equivalent to R0 = R0 + 2*R0 = 3*R0.

- Example in the notes

RTL Examples

Multiply R0 by 7 (start with 2 in R0)

- R1 <= R0; R0 <= 2*R0
- R0 <= 2*R0
- R0 <= 2*R0
- R0 <= R0 - R1



RTL Examples

Multiply R0 by 7 (start with 2 in R0)

- R1 <= R0; R0 <= 2*R0
- R0 <= 2*R0
- R0 <= 2*R0
- R0 <= R0 - R1

These four steps result in R0 = 8*R0 - R0 = 7*R0.

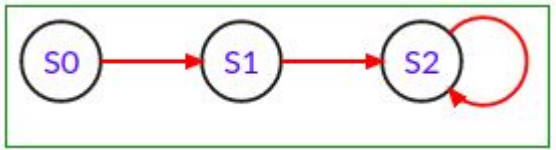
- Example in the notes

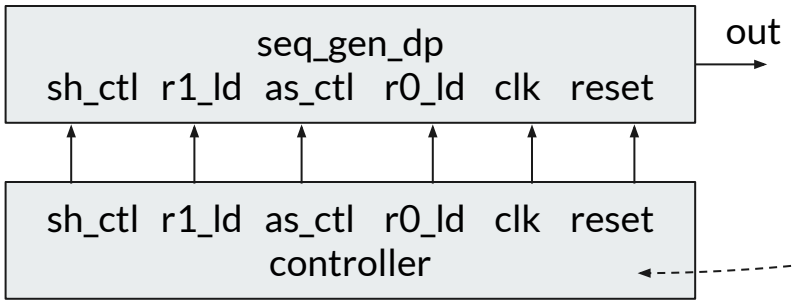
Implementing A FSM To Set R0 and R1 to 1

- Use a FSM to help us perform the operations
- Each FSM state set a logic control signals following RTL
- At the clock, states changes and update the registers based on the signals

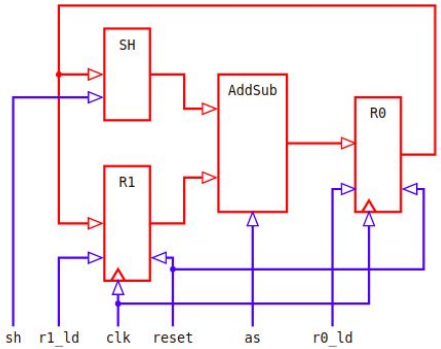
Implementing A FSM To Set R0 and R1 to 1

- The state graph for a FSM that sets R0 and R1 to 1
- S0 is the reset state
- 1 is loaded into R0 during the transition from S0 to S1
- R1 is set to 1 during the transition from S1 to S2
- S2 is the final state, and indicates there is no more work to be done.





Testing The Mini Data Path



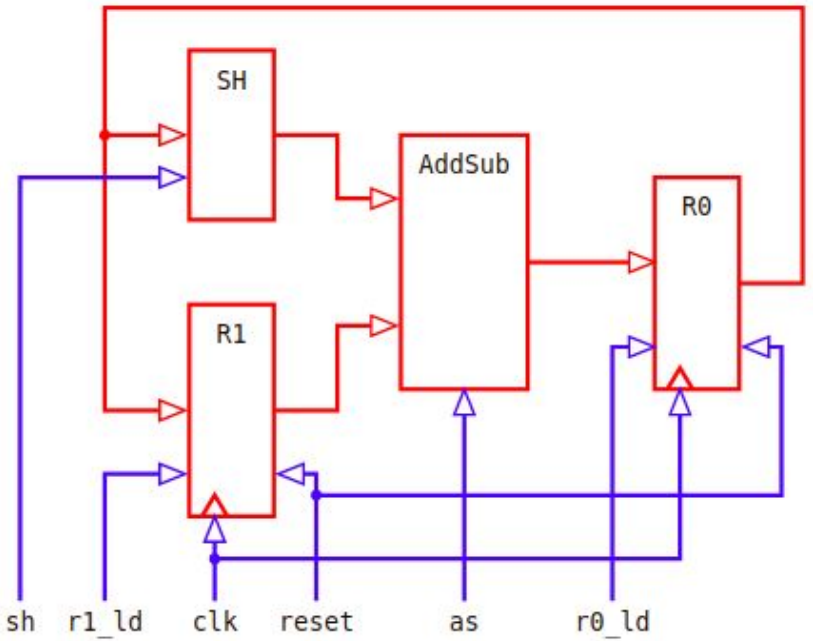
Verilog provides the “include” statement to include one verilog program into another verilog program

- `seq_gen.v`

```
`include "types.v"

`include "ld_reg.v"

`include "as_alu.v"

`include "shifter.v"

`include "seq_gen_dp.v"
```

Register Transfer Level (RTL) Implementation

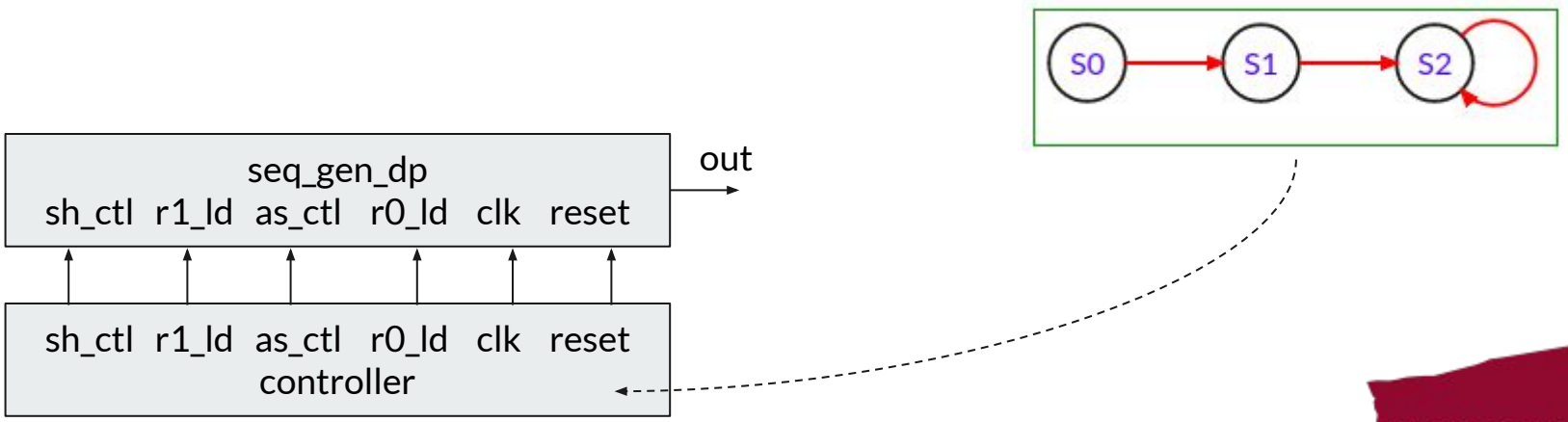
- Datapath files:
  - `include "types.v"
  - `include "ld_reg.v"
  - `include "as_alu.v"
  - `include "shifter.v"
  - `include “seq_gen_dp.v"
- seq_gen.v
- tb_rtl.v (no controller, hard coded rtl control)



Implementing A FSM To Set R0 and R1 to 1

The module sections presented implements the FSM with the states :

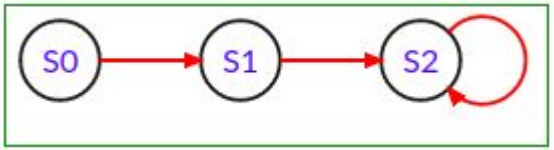
```
enum logic [1:0] { S0, S1, S2 } st;
```



Implementing A FSM To Set R0 and R1 to 1

- States set the control signals. Clock changing the states:

```
always_ff @(posedge clk) begin
    if ( reset ) st <= S0;
    else begin
        case( st )
            S0: st <= S1;
            S1: st <= S2;
            S2: st <= S2;
        endcase
    end
end
```



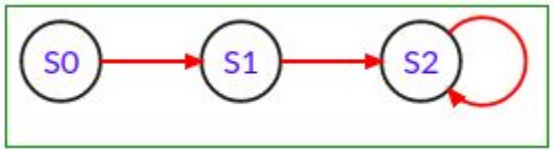
Implementing A FSM To Set R0 and R1 to 1

There is also a “ctrl” task:

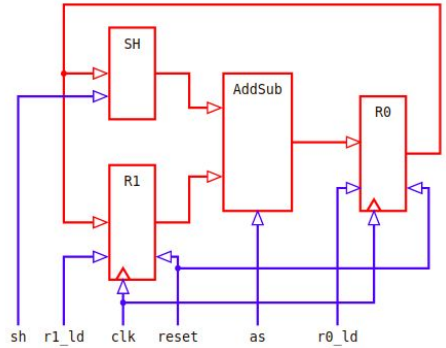
```
task automatic ctrl(as_ctl_t as, shift_ctl_t sh, logic r1, logic r0 );
     as_ctl = as; shift_ctl = sh; r1_ld = r1; r0_ld = r0;
endtask
```

Connects the internal values: as, sh, r1, r0 to the outputs as_ctrl, shift_ctl, r1_ld, r0_ld

The ctrl task is used to compactly write the configuration of the control signal.



Register Transfer Level (RTL) Design



For

```
ctrl( as, sh, r1, r0 );
```

During S0 (`R0 <= 1`) the value of the control signals is specified with:

```
ctrl(AS_A, SH_ONE, 0, 1 )
```

- r0_ld=1, and the shifter is outputs a 1
- the add/sub units passes the 1 to R0.
- tb_rtl.v

as_ctl	shift_ctl	r0_ld	RTL
AS_ADD	SH_PASS	1	R0 <= R0 + R1
AS_ADD	SH_LEFT	1	R0 <= 2*R0 + R1
AS_SUB	SH_PASS	1	R0 <= R0 - R1
AS_SUB	SH_LEFT	1	R0 <= 2*R0 - R1
AS_A	SH_ONE	1	R0 <= 1
AS_A	SH_ONES	1	R0 <= -1
AS_A	SH_LEFT	1	R0 <= 2*R0

Implementing A FSM To Set R0 and R1 to 1

```
always_comb begin

    case( st )

        S0: ctrl(AS_A, SH_ONE, 0, 1 ); // R0 <= 1

        S1: ctrl(AS_A, SH_ONE, 1, 0 ); // R1 <= R0

        S2: ctrl(AS_A, SH_ONE, 0, 0 );

    endcase

end
```

as_ctl	shift_ctl	r0_ld	r1_ld	RTL
AS_B	X	1	0	R0 <= R1
X	X	0	1	R1 <= R0
AS_B	X	1	1	R1 <= R0; R0 <= R1

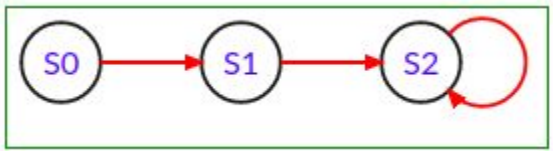
as_ctl	shift_ctl	r0_ld	RTL
AS_ADD	SH_PASS	1	R0 <= R0 + R1
AS_ADD	SH_LEFT	1	R0 <= 2*R0 + R1
AS_SUB	SH_PASS	1	R0 <= R0 - R1
AS_SUB	SH_LEFT	1	R0 <= 2*R0 - R1
AS_A	SH_ONE	1	R0 <= 1
AS_A	SH_ONES	1	R0 <= -1
AS_A	SH_LEFT	1	R0 <= 2*R0

Implementing A FSM To Set R0 and R1 to 1

The controller generates all the control signals needed to operate the mini data path

```
seq_gen_dp DP(seq_out, clk, reset, as_ctl, shift_ctl, r1_ld, r0_ld);

controller CTL( as_ctl, shift_ctl, r1_ld, r0_ld, clk, reset);
```

- tb_fsm_set_r0_r1.v (includes datapath and controller)

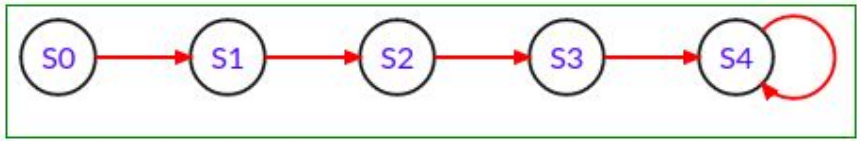


Implementing A FSM To Set R0 to 5

The RTL code to set R0 to 5 is:

- R0 <= 1
- R1 <= R0
- R0 <= 2*R0
- R0 <= 2*R0 + R1

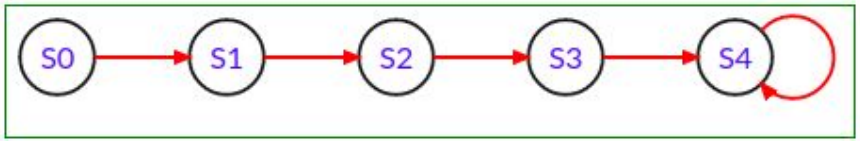
The state graph for a FSM that sets R0 to 5



Implementing A FSM To Set R0 to 5

Define states/ctrls (RTL)

- S0: R0 <= 1
- S1: R1 <= R0
- S2: R0 <= 2*R0
- S3: R0 <= 2*R0 + R1
- S4: Idle state

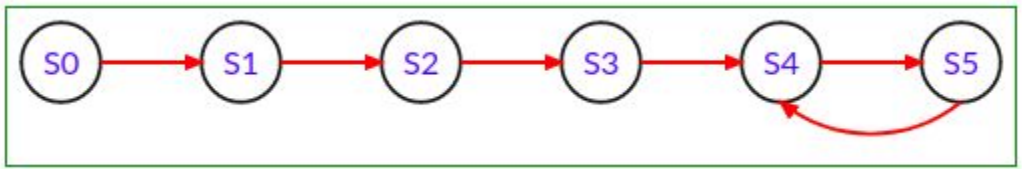


Code example in the notes

Implementing A FSM To Generate 5, 15, 45, 135...

The mini datapath can be used to generated arithmetic sequences

The RTL for a controller that generates the sequence: 5, 15, 135 ... (each value is multiplied by 3) is:



Implementing A FSM To Generate 5, 15, 45, 135...

S0: R0 <= 1

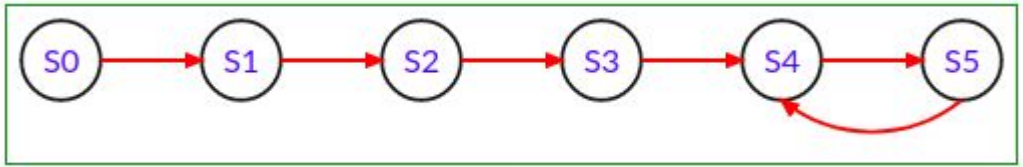
S1: R1 <= R0

S2: R0 <= 2*R0

S3: R0 <= 2*R0 + R1

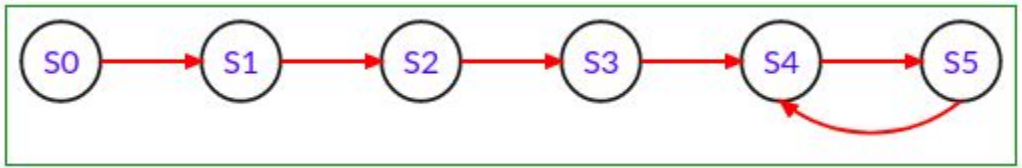
S4: R1 <= R0; R0 <= 2*R0

S5: R0 <= R0 + R1



Implementing A FSM To Generate 5, 15, 45, 135...

- The first four steps set R0 to 5
- Repeating steps 5 and 6 multiples the value in R0 by 3
- The FSM will continue to generate multiples of 3
- After S5 transition to S4
- S4 is the output state
- tb_fsm_seq.v



Next Unit

- Assembly language