DESIGN OF DIGITAL CIRCUITS AND SYSTEMS

Algorithmic State Machines

Instructor: Justin Hsia

Teaching Assistants:

Colton Carroll
Hemil Patel
Rasya Fawwaz
Grace Zhou
Quinlyn Donohue
Rose Maresh

Relevant Course Information

- Homework 2 due Wednesday (4/15)
- Homework 3 released today, due next Friday (4/24)

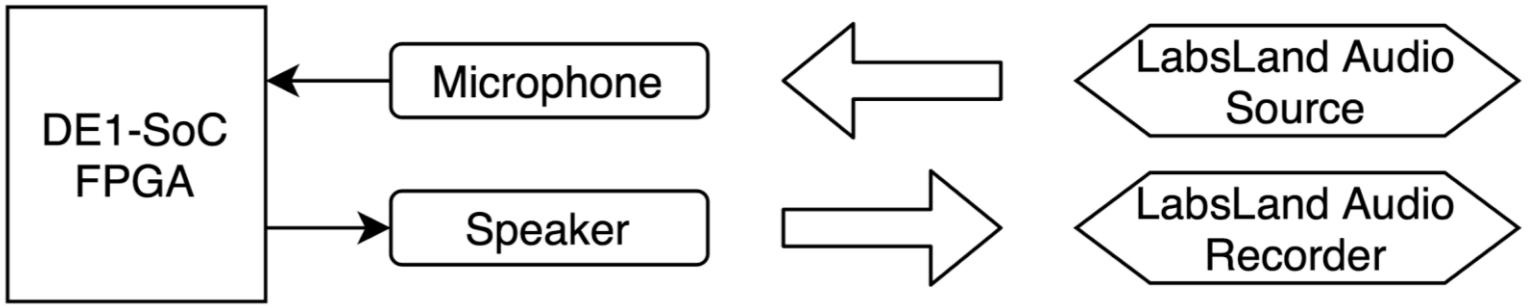
- Lab 2 reports due 4/17, demos 4/20-24
- Lab 3 released today, due in two weeks (5/1)
  - Lab 3 + 4 are really ~1.5 weeks long, so don't wait!

- Quiz 2 not until *next* Thursday (4/23)
  - Spacing between material and quiz will get longer and longer; make sure to give time to review

Lab 3 Notes (1/2)

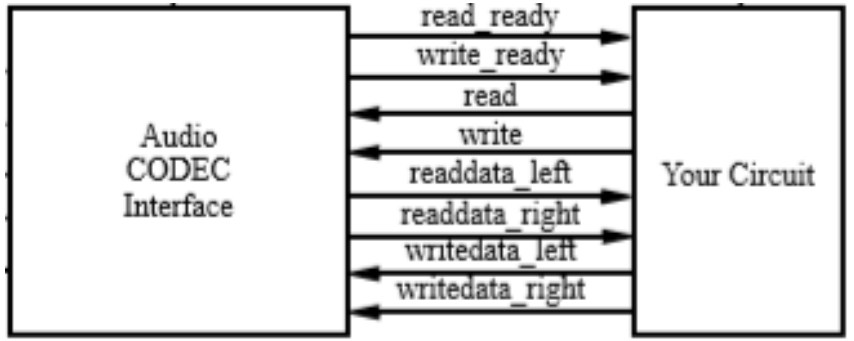
- More practical **applications of memory** on the DE1-SoC using **audio generation and filtering**
  - Task 2: ROM with MIF file to generate audio
  - Task 3: Use a FIFO buffer to implement a noise filter

- See the Audio Guide page for how to use the LabsLand Audio Interface to send audio input and record audio output:



Lab 3 Notes (2/2)

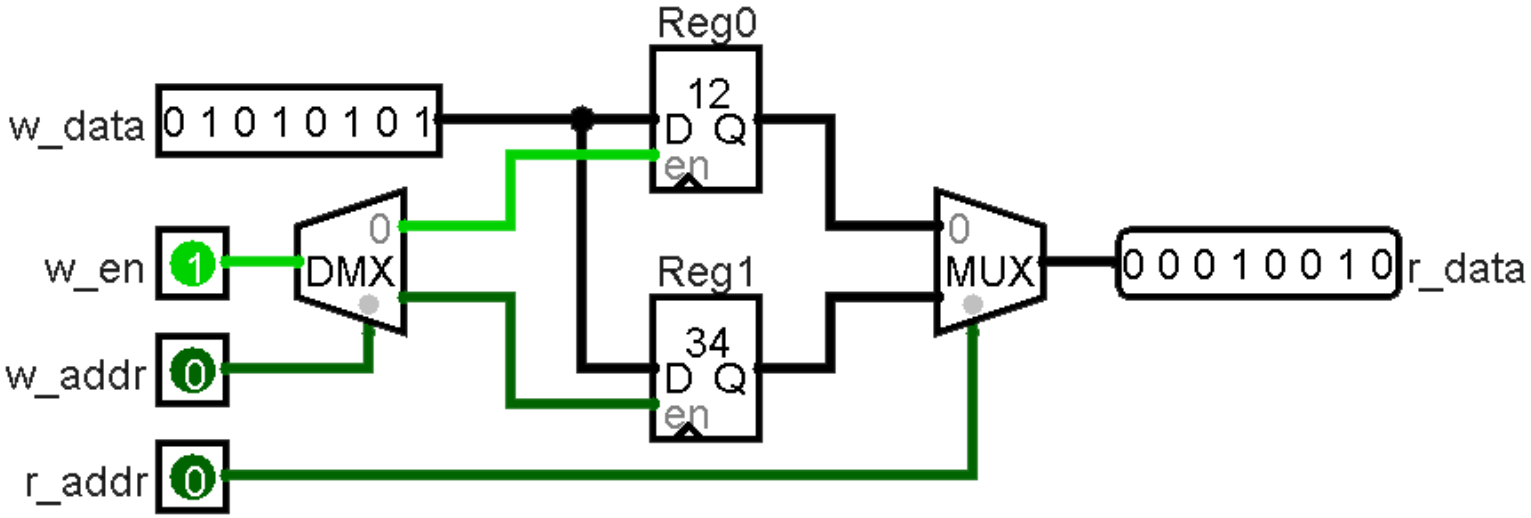
- Example of *communication* as you interface with an audio CODEC (coder/decoder)
  - Inputs: `read`, `write`, `writedata_left`, `writedata_right`
  - Outputs: `read_ready`, `write_ready`, `readdata_left`, `readdata_right`



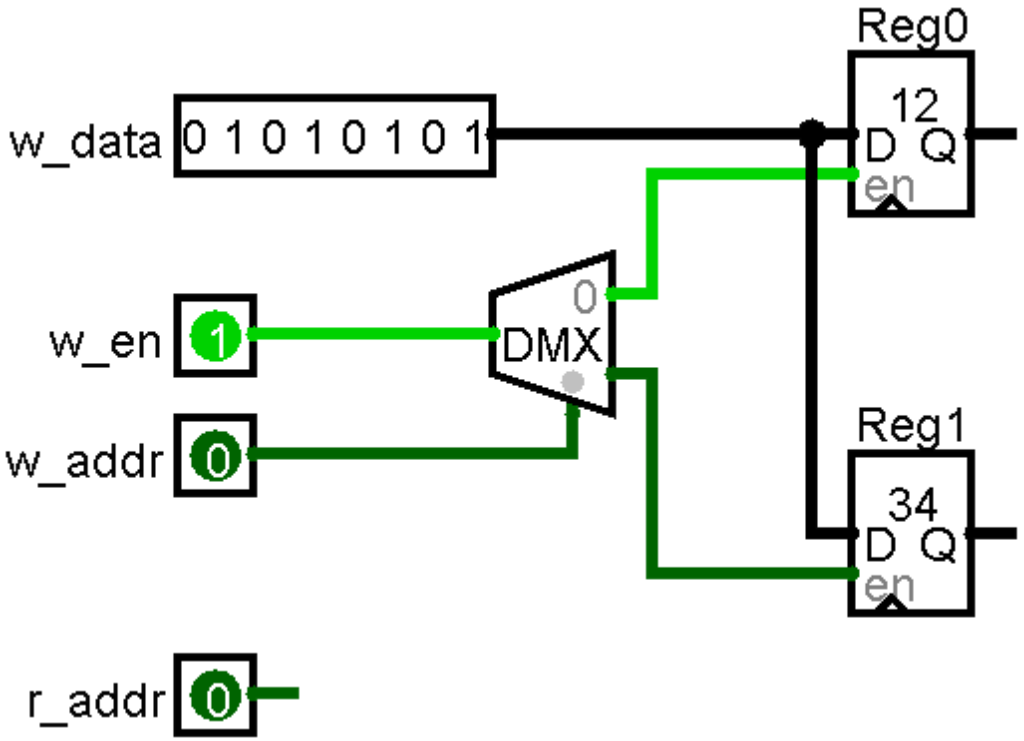
  - Must wait for both sides (CODEC + your circuit) to be ready for data transmission in either direction!
    - Data is ready/generated AND receiver is ready to accept

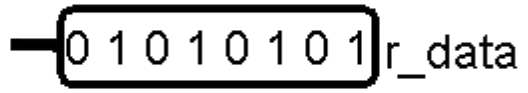
Review Question: Memory Implementation

- Below is an implementation of a 1-port, 2×8 regfile
  - Currently, when we write to the address we are reading from, `r_data` reflects the current/old data in the register (*i.e.*, `r_data` ← `regfile[w_addr]`)
- Modify the implementation so that `r_data` ← `w_data` (the new data) in this edge case instead
  - Hint: It may help to write out the Verilog logic before converting to hardware



Memory Implementation Solution



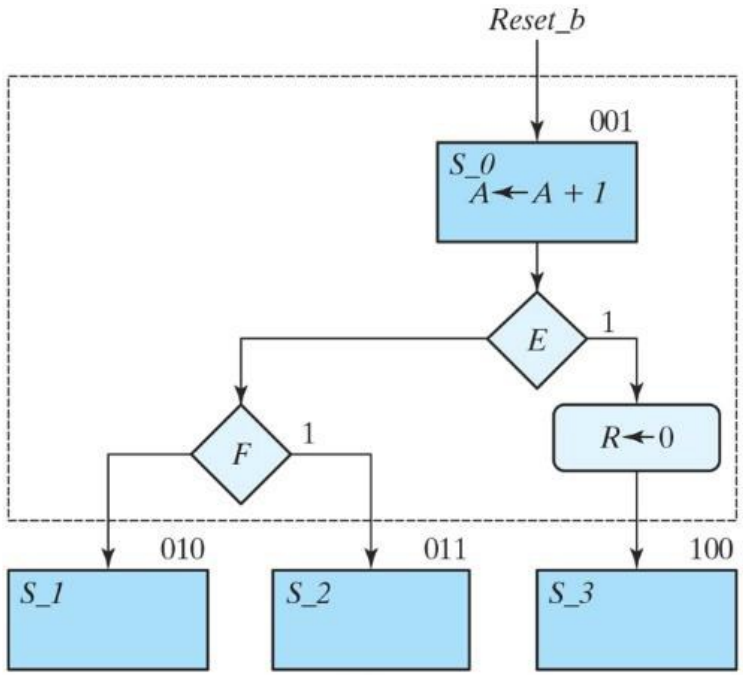


Specifying Synchronous Digital Systems (SDS)

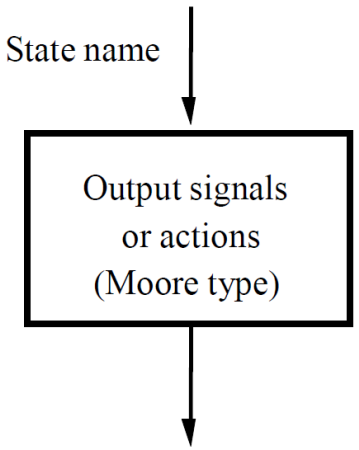
- So far:
  - SystemVerilog
  - Block diagrams
  - Finite State Machines
  - Circuit/gate diagrams
- Issues:
  - SV is a specified language (rigid syntax) and can be very abstract (behavioral)
  - Block diagrams can be vague or unspecified
  - FSMs don’t scale well (# of states + transitions)
  - Gate-level is too detailed and specific

Algorithmic State Machine (ASM)

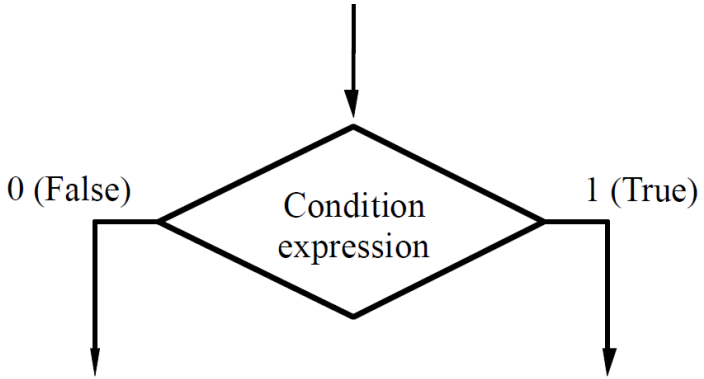
- ASM charts are a method for designing and depicting synchronous digital systems
  - Use more generic syntax (RTL) than SystemVerilog
  - Contain more structured information than FSM state diagrams
  - Can more easily design your system from a *hardware algorithm*



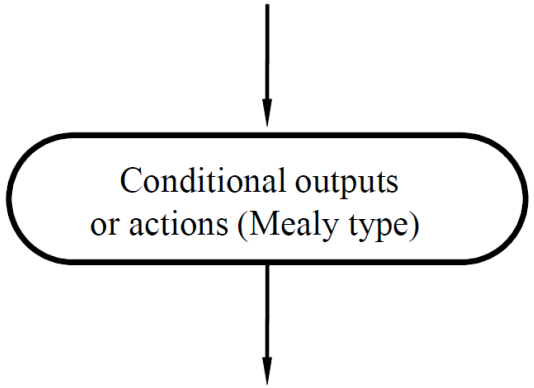
ASM Chart Components



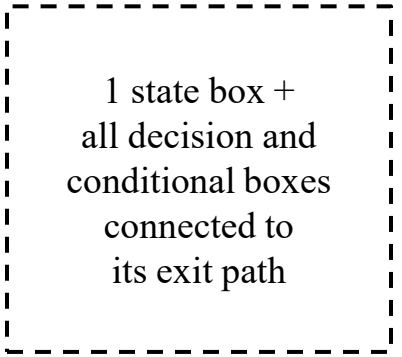
(a) State box



(b) Decision box



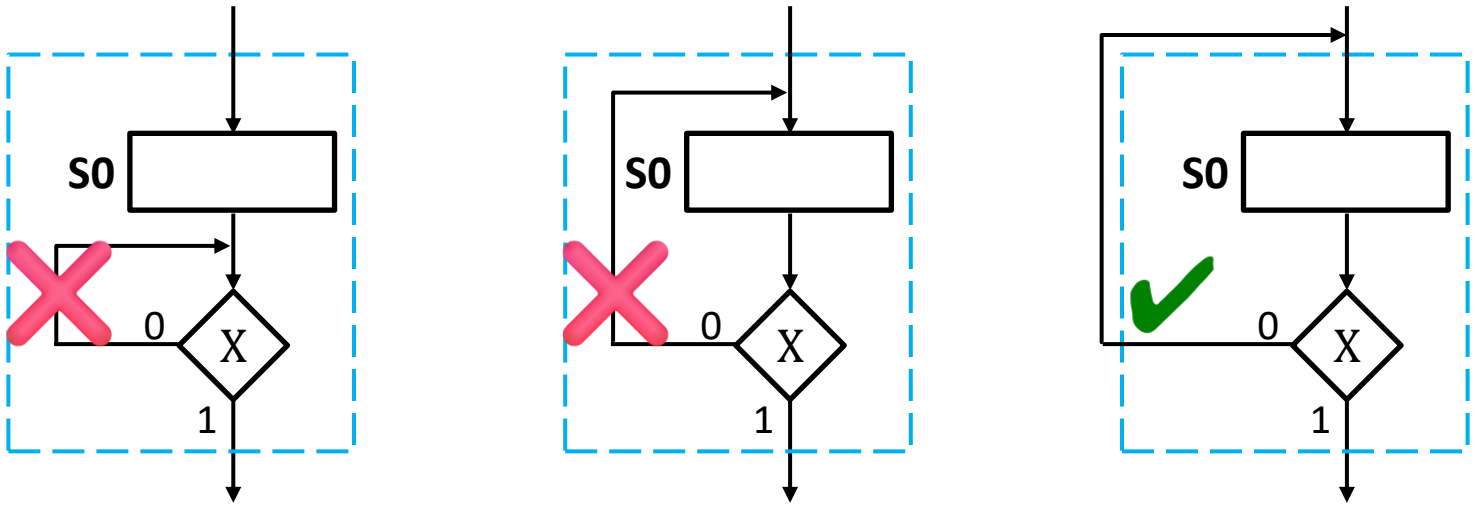
(c) Conditional output box



(d) ASM block

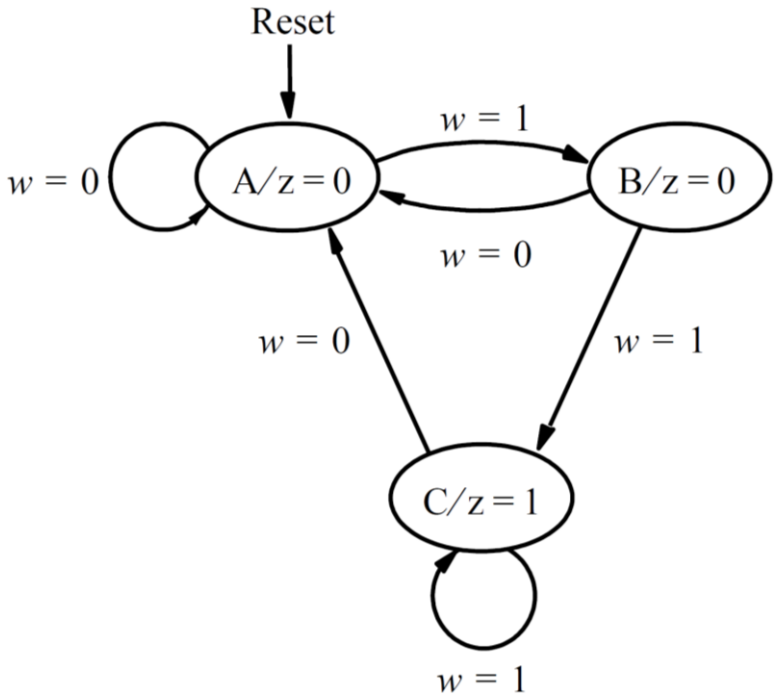
ASM Blocks

- Each block describes the state machine operation in a given state
  - For every valid combination of inputs, there must be **exactly one** exit path
  - There should be **no internal feedback**



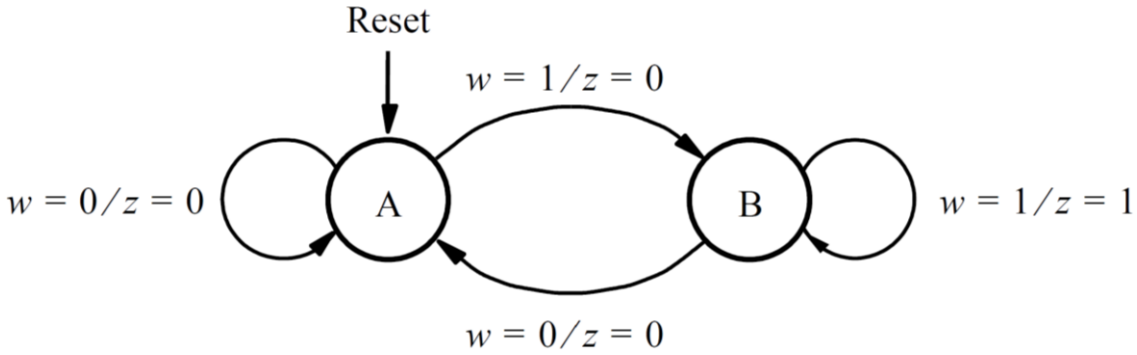
Worked Example #1

- Convert this state machine to an ASM chart:



Worked Example #2

- Convert this state machine to an ASM chart:



TECHNOLOGY

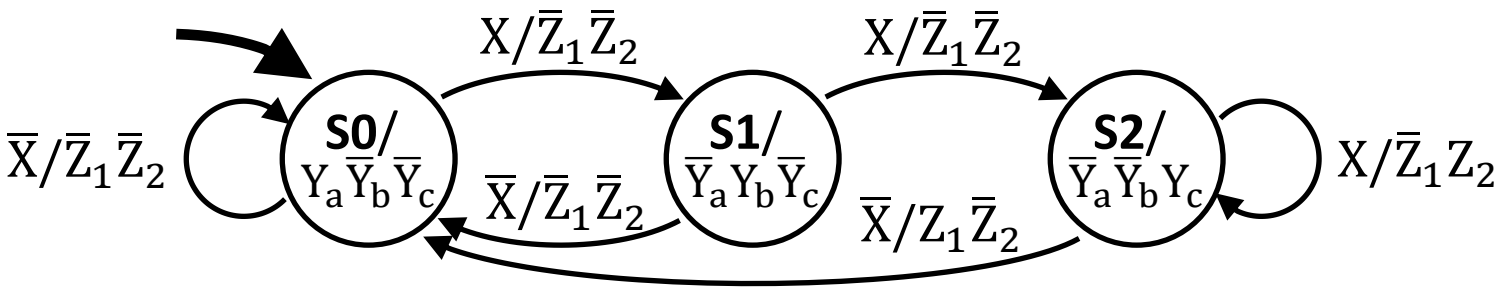
BREAK

Example #3

- Draw an ASM chart for `threeOnes` – asserts `out` iff last 3 values of `in` were all `1`'s.

Example #4

- Convert this state machine to an ASM chart:
  - 1 input: X, 5 outputs: Y_a, Y_b, Y_c (Moore), Z_1, Z_2 (Mealy)

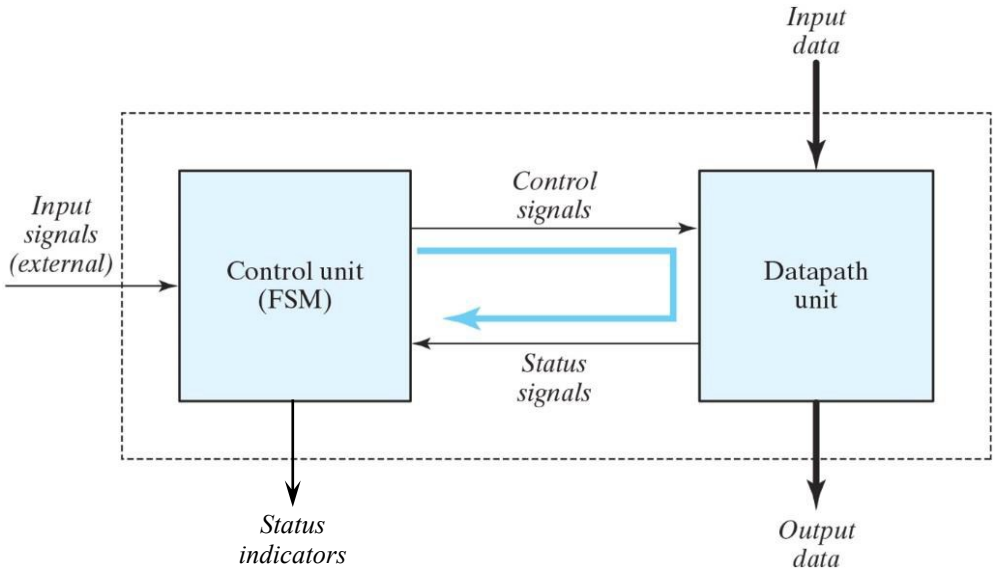


Algorithms for Hardware

- *Sequential* algorithms:
  - Variables used as symbolic memory locations
  - Sequential execution dictates the ordering of operations
- Hardware implementation:
  - Registers store intermediate data (variables)
  - Datapath implements all necessary register operations (computations attached to register inputs)
  - A control path FSM specifies the ordering of register operations
- This design scheme sometimes referred to as **register-transfer level** (RTL) design

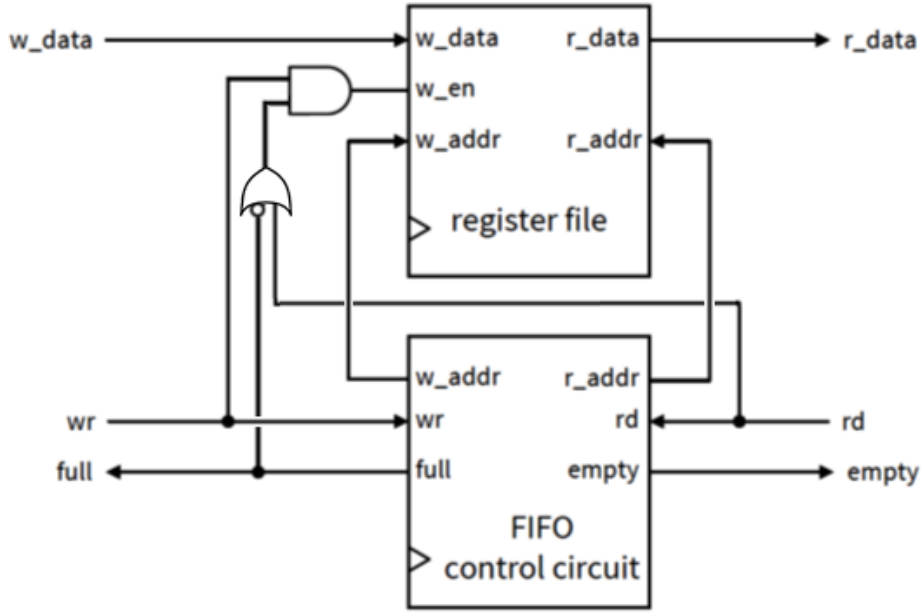
Control and Datapath

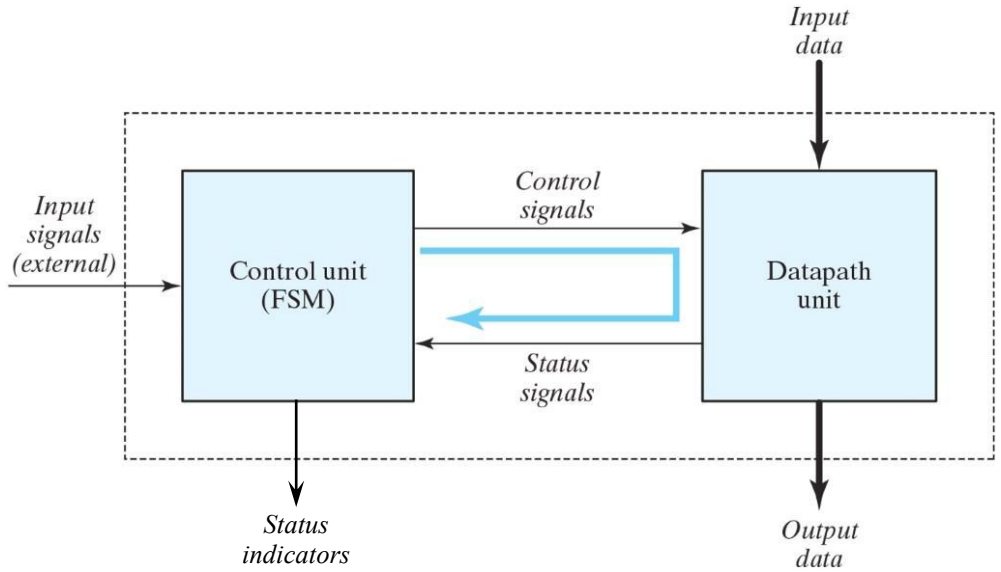
- Signal classification in an SDS:
  - *Data*: Information manipulated/processed by the system
  - *Control*: Signals that coordinate and execute the system operations
- We can logically separate an SDS into two distinct parts/circuits:
  - **Datapath:** Parts needed for data manipulation ("the brawn")
  - **Control:** Logic that tells the datapath what needs to be done ("the brain")



Control and Datapath: FIFO Buffer

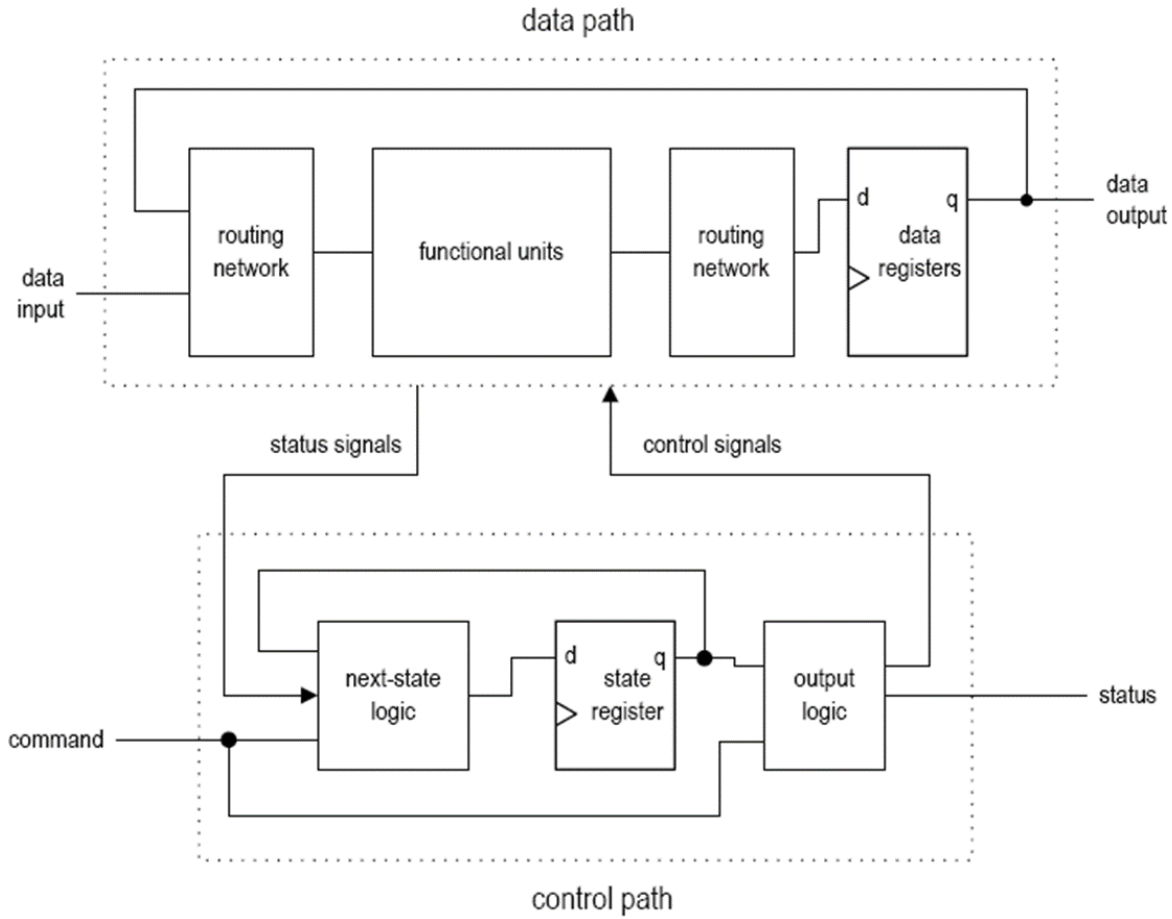
- Circular queue implementation from last lecture:
  - Datapath and control split?





Algorithms for Hardware: Schematic

- The resulting system is called an algorithmic state machine (ASM) or FSM with a datapath (FSMD):



RTL Operations

- Basic form:

$$r_{\text{dest}} \leftarrow f(r_{\text{src1}}, r_{\text{src2}}, \ldots, r_{\text{src}n})$$

  - r_i represent registers and $f()$ represents some combinational function

- Examples:
  - $r_1 \leftarrow 0$
  - $r_2 \leftarrow r_1$
  - $r_2 \leftarrow r_2 \gg 3$
  - $i \leftarrow i + 1$
  - $d \leftarrow s_1 + s_2 + s_3$
  - $y \leftarrow a * a$

RTL Operations: Timing

- Basic form:

$$r_{\text{dest}} \leftarrow f(r_{\text{src1}}, r_{\text{src2}}, \ldots, r_{\text{src}n})$$

  - r_i represent registers and $f()$ represents some combinational function

- Timing Interpretation:
  - After the start of a clock cycle, the outputs of all registers update
  - During the rest of the clock cycle, these outputs propagate through the combinational circuit that performs $f()$
  - At the *next* clock trigger/cycle, the result is stored into r_{dest}

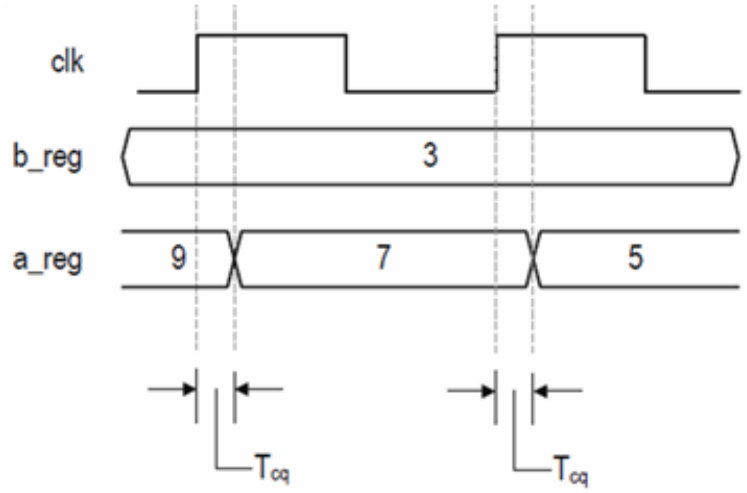
RTL Operations: Example

- Basic form:

$$r_{\text{dest}} \leftarrow f(r_{\text{src1}}, r_{\text{src2}}, \ldots, r_{\text{src}n})$$

  - r_i represent registers and $f()$ represents some combinational function

- Implementation Example: $a \leftarrow a - b + 1$



Worked Example #5 (Preview)

- Convert the ASM chart for a control circuit shown in figure (b) to a state diagram: