Connecting FSMs

Flow chart

↓

High-level states

↓

Control of logic components

Overview of a modern system's

Storage elements

Happy Halloween!
Let's say that we want to find the minimum number in a set of 10 integers.

In C, our code looks something like...

```c
int array[10];  /* 10 integers - filled in by other code */
int idx;
int min = array[0];
for (idx = 1; 10 > idx; idx++)
    if (min > array[idx])
        min = array[idx];
```

/* minimum value is now in min */

Let's translate to a flow chart...
Now let's think about building an FSM to implement our flow chart.

Variables will become registers / counters. (These are design choices!)
The array will become a memory.
We'll need a comparator - let's use our serial design. [Illustrates sub-state management again]

We need shift registers to 'feed' the comparator one bit per cycle, and a counter to keep track of its progress.

How do we 'identify states?'
- Some dependence on what components we use. (That's why we started with some thoughts on that part of the question)
- Need to be able to implement each 'state' (high-level) in a fixed number of cycles (or at least controllable number of cycles).
- Not every box in flow chart becomes a state

For example (our flow chart)
- Can initialize min, idx in one cycle (state)
- First comparison answer is known (10x1) - skip it
  => Create one INIT state for this purpose

Can predicate execution with logic, especially for short sequences.

For example, when comparator is done, we can use its output to determine whether min copies array Eidx and increment idx in some cycle.
we create a COPY state for this purpose
- We want to be able to control this FSM with other logic
  - fill memory with values
  - execute FSM 'code'
  - check the answer
- create a WAIT state for when FSM is not in use (add external signal START so we can wait on something)

- Sometimes we may need many states to implement what seems a simple step in flow chart...

For example
  Comparing min to array [idx]

Need to prepare!
  Copy min to shift reg A
  Copy array[ idx] to shift reg B
  Reset counter

PREP state
  First bit into comparator
    0

COMPARE state
  (uses 5-bit counter)
  Ready to move to
  COPY state
  [go back and draw state circles on flow chart...]
current state diagram

WAIT \( \xrightarrow{\text{START signal}} \) INIT \( \xrightarrow{\text{always}} \) PREP
\( \xrightarrow{\text{not end of loop}} \) always

\text{COPY} \leftarrow \text{counter back to 0}

In reality (as I did, but not as I'm showing you),
you go back and forth thinking about states,
components, and flow from state to state.

The layout...
Component Outputs we'll need (maybe)

DONE (IDX = 9)
LAST (complete counter #: 31)
Z1, Z0 (comparison result)

Component Inputs we need to Set:

IDX, RST
IDX, CNT
MIN, LD
A, LD
B, LD
CNT, RST

Review states and what they do...

WAIT set IDX to 0 (so we can read ARRAY[0] in INIT)
wait for START signal

ON START

INIT MIN = ARRAY[0]
IDX = 1

always

PREP A = MIN
B = ARRAY[IDX]

always reset CNT

COMPARE run the serial comparator, move to next state
when CNT = 31 (LAST)

DONE

NOT DONE (IDX+1

COPY if we found A>B, then MIN=ARRAY[IDX]
old value

in same cycle, IDX = IDX+1
5 states $\rightarrow$ 3 flip-flops?

Let's just use a one-hot encoding: 5 flip-flops, with exactly one 1 (always).

<table>
<thead>
<tr>
<th>States</th>
<th>IDX.RST</th>
<th>IDX.CNT</th>
<th>MIN.LD</th>
<th>A.LD</th>
<th>B.LD</th>
<th>CNT_RST</th>
</tr>
</thead>
<tbody>
<tr>
<td>WAIT</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>INIT</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PREP</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>COMPARE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>COPY</td>
<td>0</td>
<td>1</td>
<td>?</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Let students fill in.

IDX.RST $= S_4$
IDX.CNT $= S_3 + S_0$
MIN.LD $= S_3 + S_0 Z_1$ (see below)
A.LD $= S_2$
B.LD $= S_2$
CNT_RST $= S_4 + S_2$

\( S_4^+ = S_4 \cdot \overline{\text{START}} + S_0 \cdot \overline{\text{DONE}} \)

\( S_3^+ = S_4 \cdot \text{START} \)

\( S_2^+ = S_3 + S_0 \cdot \overline{\text{DONE}} \)

\( S_1^+ = S_2 + S_1 \cdot \overline{\text{LAST}} \)

in COPY, we want to load MIN
if MIN > ARRAY[IDX]
A $> B$
which is given by

Z_1 output of comparator
Focus here is on storage, so I'll omit interconnect details.