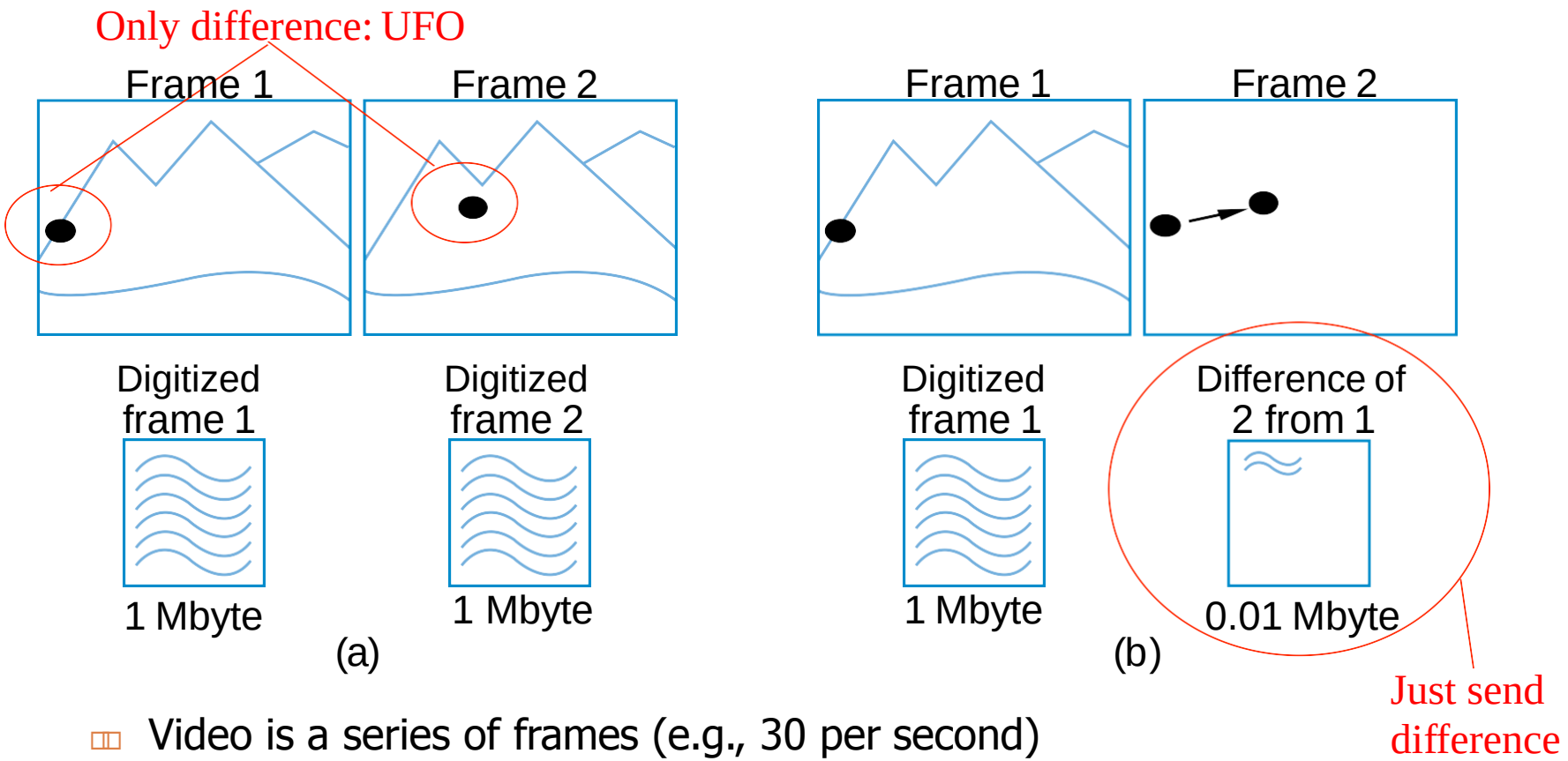
6 – RTL: C to HLSM

ECE 474A/574A
COMPUTER-AIDED LOGIC DESIGN

RTL Design Method

	Step	Description
Step 1:	Capture the high-level FSM	Describe the system's desired behavior as a high-level state machine. The state machine consists of states and transitions. The state machine is "high-level" because the transition conditions and the state actions are more than just Boolean operations on bit inputs and outputs
Step 2:	Create a datapath	Create a datapath to carry out the data operations on the high-level state machine
Step 3:	Connect the datapath to the controller	Connect the datapath to the controller block. Connect external Boolean inputs and output to the controller block
Step 4:	Derive the controller's FSM	Convert the high-level state machine to a finite-state machine (FSM) for the controller, by replacing data operations with setting and reading of control signals to and from the datapath

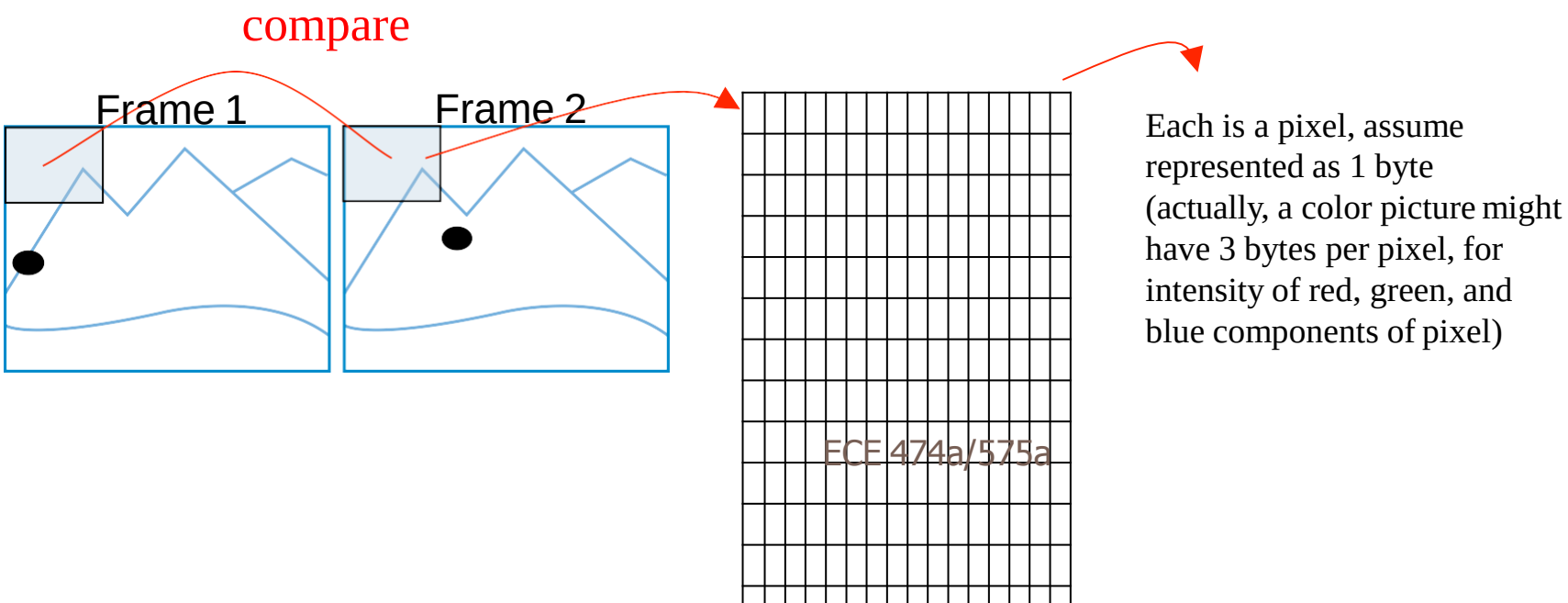
Video Compression – Sum of Absolute Differences



- Video is a series of frames (e.g., 30 per second)
- Most frames similar to previous frame
  - Compression idea: just send difference from previous frame

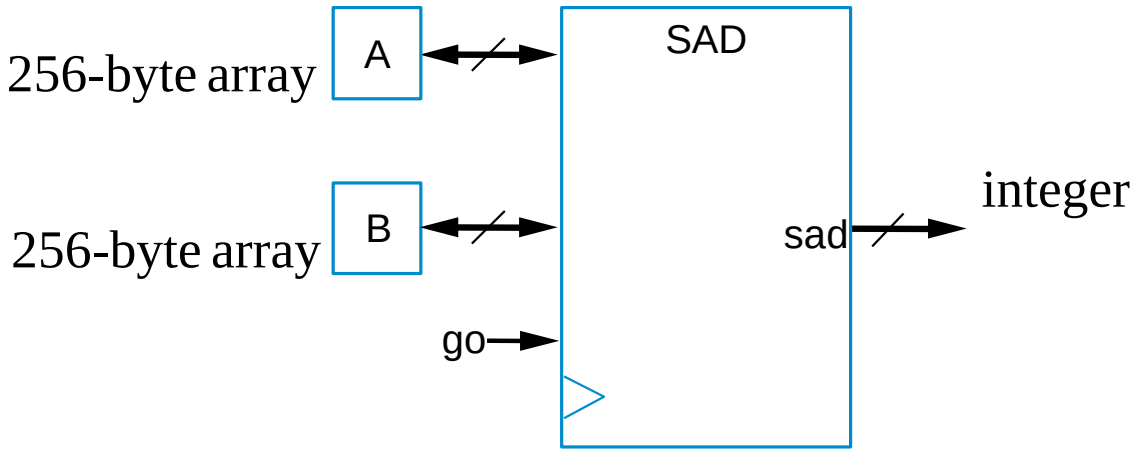
Video Compression – Sum of Absolute Differences

- Need to quickly determine whether two frames are similar enough to just send difference for second frame
  - Compare corresponding 16x16 “blocks”
    - Treat 16x16 block as 256-byte array
  - Compute the absolute value of the difference of each array item
  - Sum those differences – if above a threshold, send complete frame for second frame; if below, can use difference method (using another technique, not described)



Video Compression – Sum of Absolute Differences

- Want fast sum-of-absolute-differences (SAD) component
  - When *go=1*, sums the differences of element pairs in arrays *A* and *B*, outputs that sum



Behavioral Level Design: C to Gates

- Earlier sum-of-absolute-differences example
  - Started with high-level state machine
  - C code is an even better starting point -- easier to understand

C code

```
int SAD (byte A[256], byte B[256]) // not quite C syntax
{
    uint sum; short uint I;
    sum = 0;
    i = 0;
    while (i < 256) {
        sum = sum + abs(A[i] – B[i]);
        i = i + 1;
    }
   return sum;
}
```

Behavioral-Level Design

Start with C (or Similar Language)

- Replace first step of RTL design method by two steps
  - Capture in C, then convert C to high-level state machine
  - How convert from C to high-level state machine?

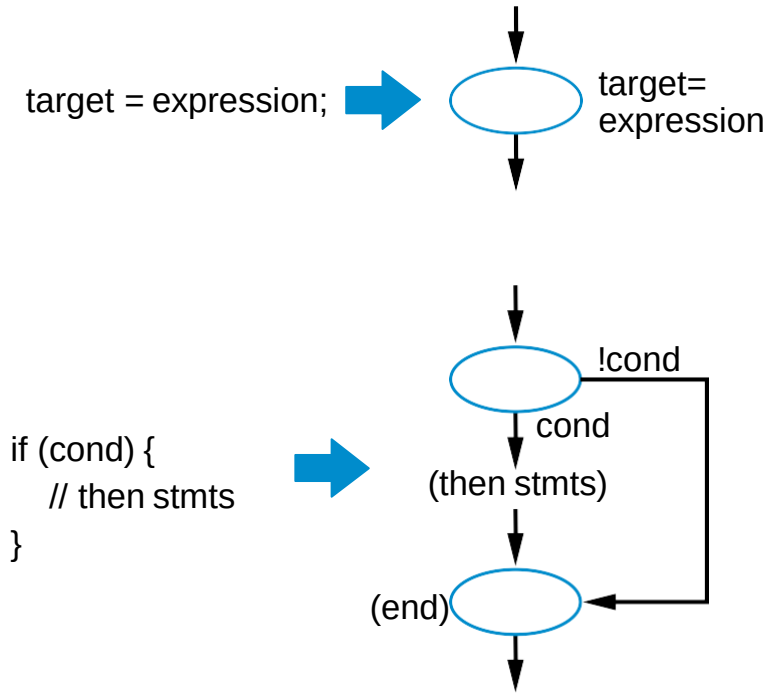
Step 1A: Capture in C

Step 1B: Convert to high-level state machine

	Step	Description
Step 1	*Capture a high-level state machine*	Describe the system's desired behavior as a high-level state machine. The state machine consists of states and transitions. The state machine is "high-level" because the transition conditions and the state actions are more than just Boolean operations on bit inputs and outputs.
Step 2	*Create a datapath*	Create a datapath to carry out the data operations of the high-level state machine.
Step 3	*Connect the datapath to a controller*	Connect the datapath to a controller block. Connect external Boolean inputs and outputs to the controller block.
Step 4	*Derive the controller's FSM*	Convert the high-level state machine to a finite-state machine (FSM) for the controller, by replacing data operations with setting and reading of control signals to and from the datapath.

Converting from C to High-Level State Machine

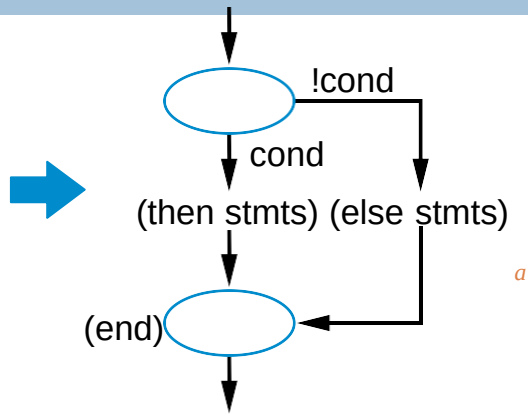
- Convert each C construct to equivalent states and transitions
- *Assignment* statement
  - Becomes one state with assignment
- *If-then* statement
  - Becomes state with condition check, transitioning to "then" statements if condition true, otherwise to ending state
    - "then" statements would also be converted to states



Converting from C to High-Level State Machine

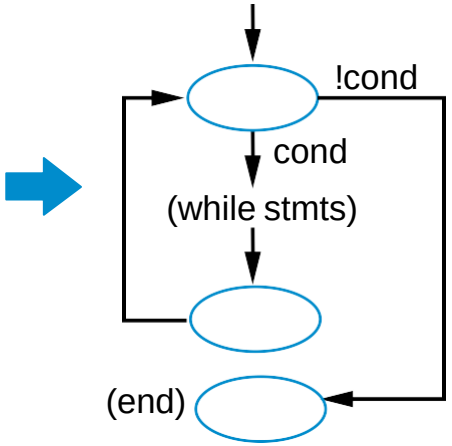
- *If-then-else*
  - Becomes state with condition check, transitioning to "then" statements if condition true, or to "else" statements if condition false

```
if (cond) {
   // then stmts
}
else {
   // else stmts
}
```



- *While loop* statement
  - Becomes state with condition check, transitioning to while loop's statements if true, then transitioning back to condition check

```
while (cond) {
   // while stmts
}
```



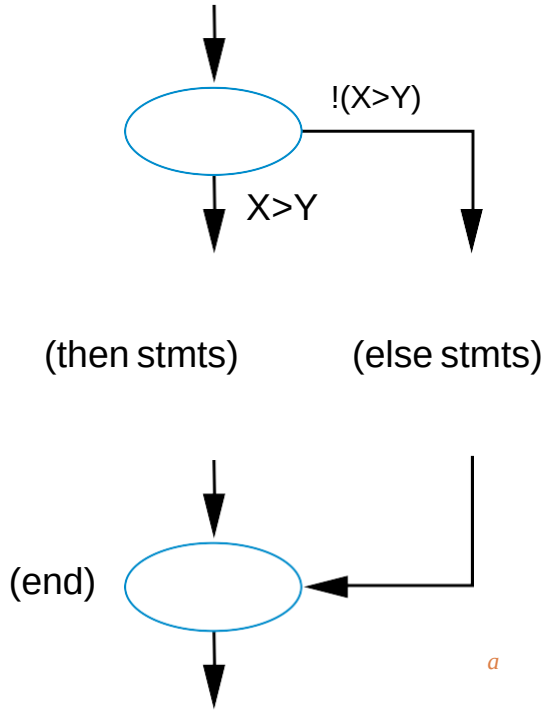
Simple Example of Converting from C to High-Level State Machine

- Simple example: Computing the maximum of two numbers
  - Convert if-then-else statement to states (b)
  - Then convert assignment statements to states (c)

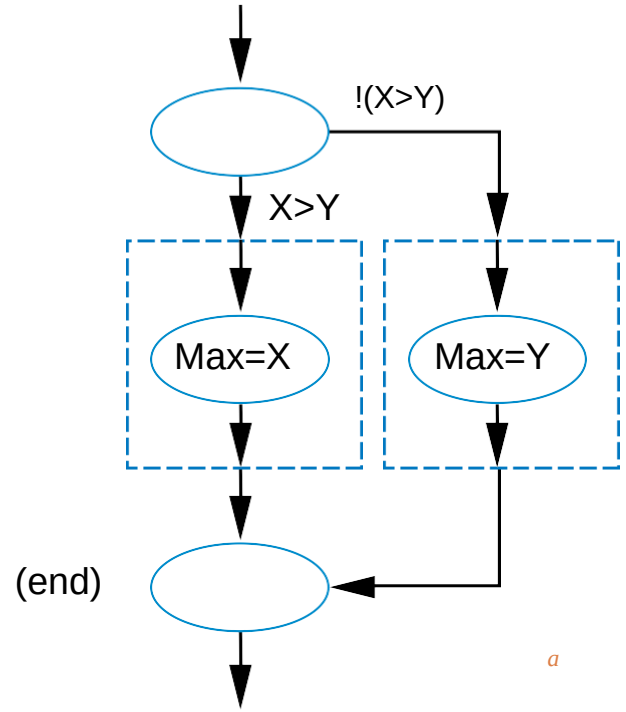
Inputs: uint X, Y
Outputs: uint Max

```
if (X > Y) {
    Max = X;
}
else {
    Max = Y;
}
```

(a)



(b)



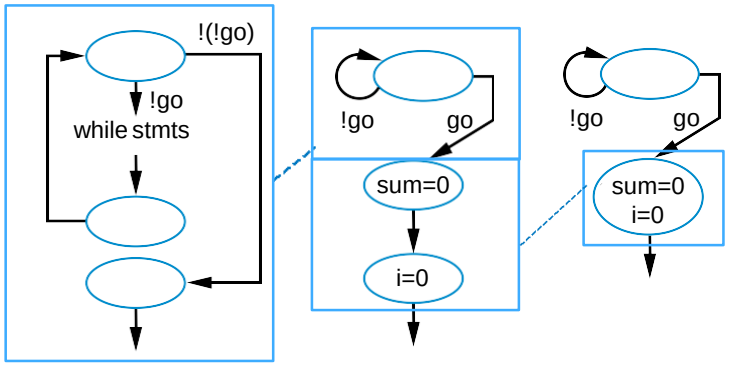
(c)

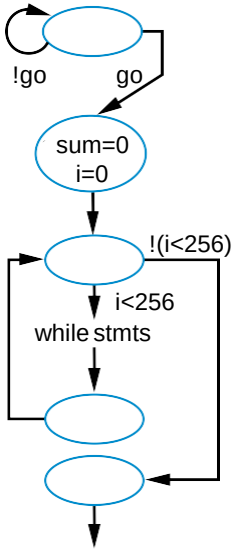
Converting Sum-of-Absolute-Differences

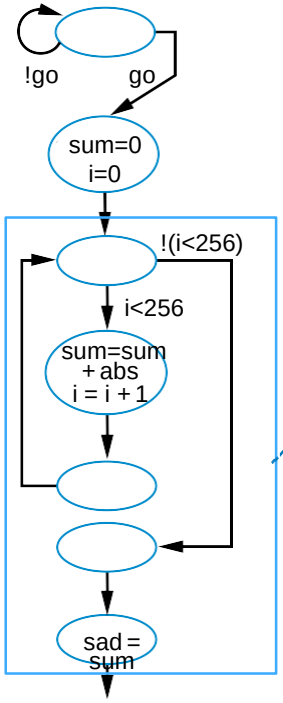
C code to High-Level State Machine

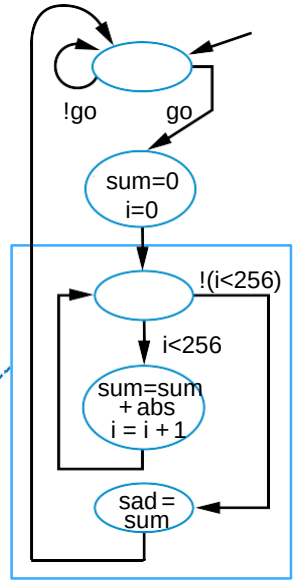
```
Inputs: byte A[256], B[256]
        bit go;
Output: int sad
main()
{
  uint sum; short uint I;
  while (1) {
    while (!go);
    sum = 0;
    i = 0;
    while (i < 256) {
      sum = sum + abs(A[i] - B[i]);
      i = i + 1;
    }
    sad = sum;
  }
}
```

- Convert each construct to states
  - Simplify when possible, e.g., merge states





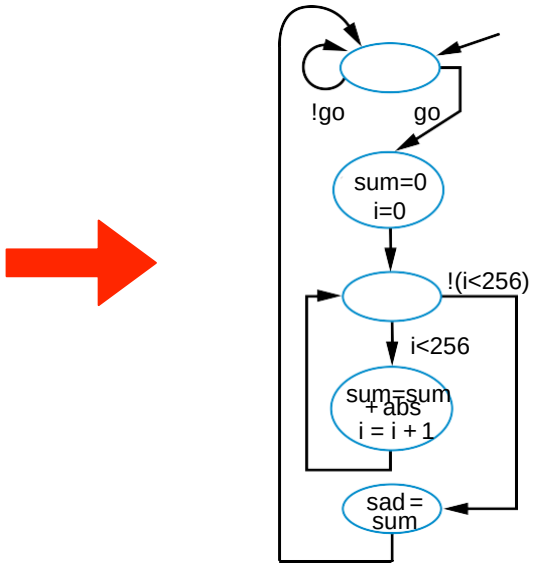




Converting Sum-of-Absolute-Differences
C code to High-Level State Machine

- From high-level state machine, follow RTL design method to create circuit
- Thus, can convert C to gates using straightforward automatable process
  - Not all C constructs can be efficiently converted
  - Use C subset if intended for circuit
  - Can use languages other than C, of course
  - **May not be optimized for resource usage or latency.**

```
Inputs: byte A[256, B[256]
        bit go;
Output: int sad
main()
{
  uint sum; short uint I;
  while (1) {
      while (!go);
      sum = 0;
      i = 0;
      while (i < 256) {
          sum = sum + abs(A[i] - B[i]);
          i = i + 1;
      }
      sad = sum;
  }
}
```



Exercise 1

Convert the following C code, which calculates the number of times the value *b* is found within an array *A* consisting of 256 8-bit values, into a high-level state machine.

Inputs: byte a[256], byte b, bit go

Outputs: byte freq, bit done

```
while(1) {
        while(!go);
        done = 0;
        i = 0;
        freq = 0;
        while( i < 256 ) {
                if( a[i] == b ) {
                        freq = freq + 1;
                }
                i = i + 1;
        }
        done = 1;
}
```

Exercise 2

Convert the following C code to a high-level state machine that interfaces with a memory supporting synchronous reads and writes. Be sure to explicitly list all inputs, outputs, and local registers. Assume **int**'s are 32 bits, **short**'s are 16 bits, and **char**'s are 8 bits.

```
// Inputs/Outputs
int A[2048];
int threshold;
char en;

// Local variables
short I, dcnt;
int now, last;

while(1) {
   while(en != 0) {
      dcnt = 0;
      now = 0;
      for(i = 0; i < 1024; i = i + 1) {
         last = now;
         now = A[i];
         if ( (last - now) > threshold ) {
            dcnt = dcnt + 1;
         }
      }
      A[1050] = dcnt;
   }
}
```

Exercise 3

```
void main()
{
      unsigned int i, start, data, mod4_count, A[128];
      while(1) {
            while(!start);
            i=0;
            mod4_count = 0;
            while(i<128) {
                  data = A[i];
                  if(data%4==0) {
                        mod4_count++;
                  }
                  i++;
            }
      }
}
```

Use the code shown above to create the high level FSM.
Show the datapath for this design.
Draw the interface between the controller and the datapath, show all inputs and outputs.
Provide the controller for the HLSM