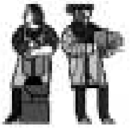
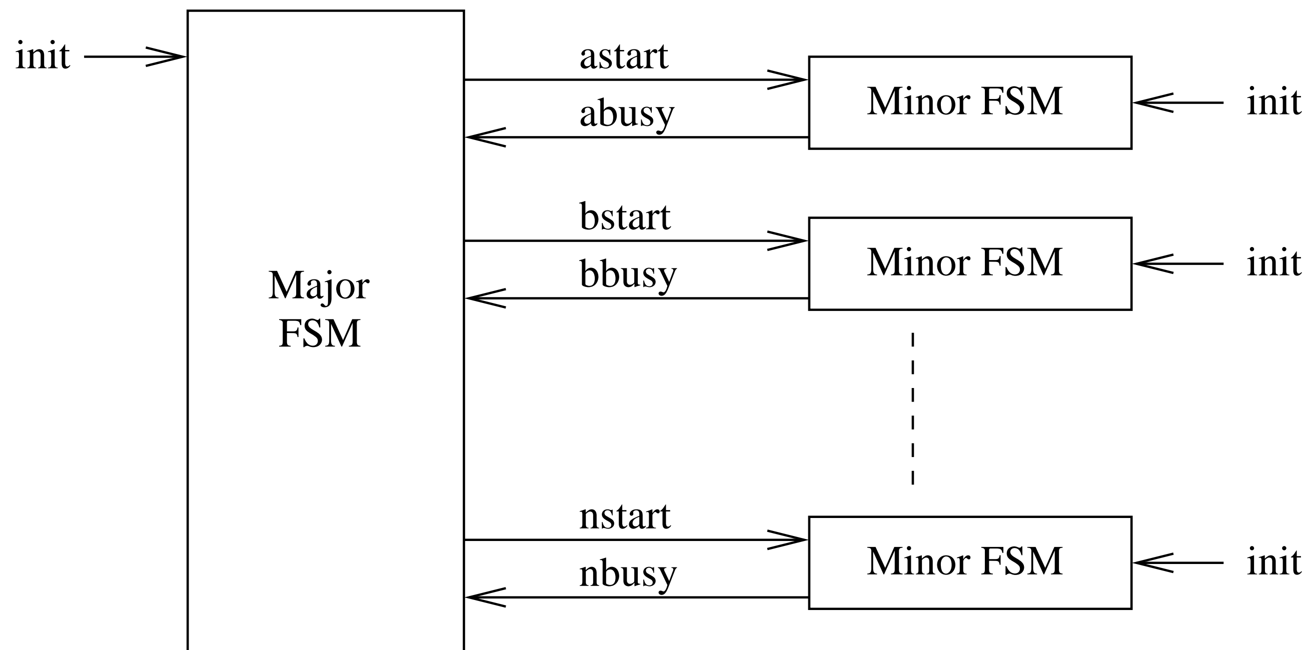


L11: Major/Minor FSMs, Lab 3, and RAM/ROM Instantiation

 Copyright © 2003 by MIT

FSM Hierarchy

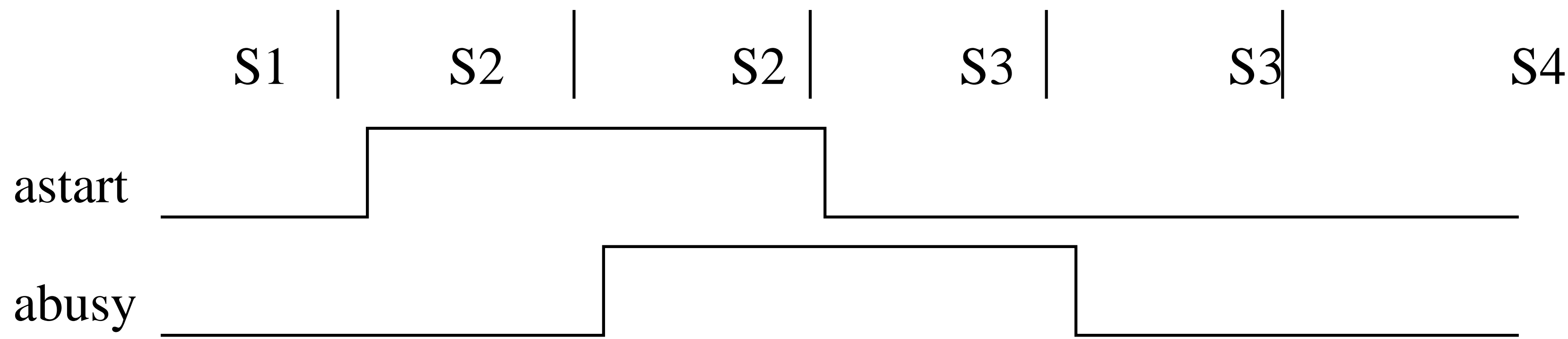
- We want to build a control system using multiple FSMs.
  - Minor FSMs are controlled (supervised) by a major FSM.
  - Minor FSMs may take different (sometimes unknown) numbers of clock cycles.
- All FSMs use the same (rising edge) clock and init signal.
  - The init signal is synchronized to the clock and one clock period long.
- Minor FSMs may be decomposed into multiple FSMs.

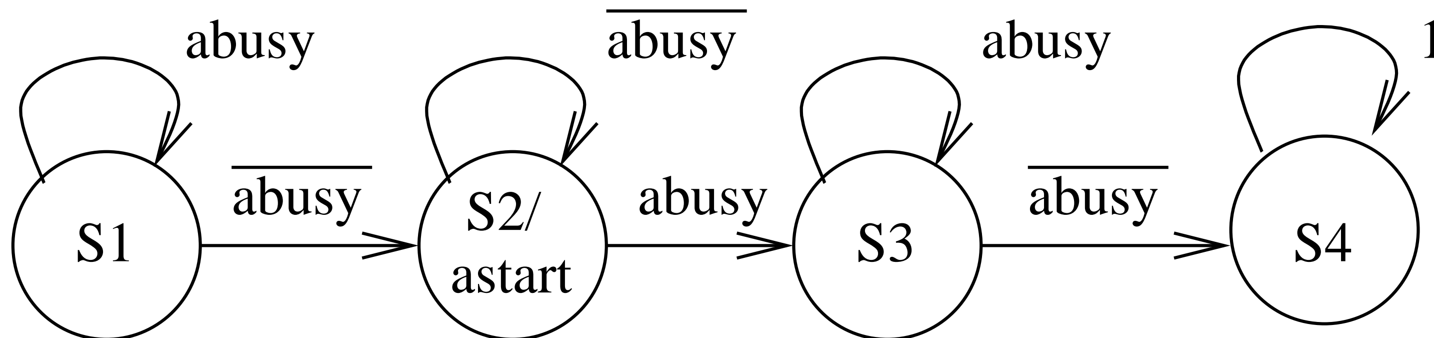


 Copyright © 2003 by MIT

Major FSMs

- A major FSM tests the minor FSM's busy signal to determine when the minor FSM is available.
- Then a control signal (astart) for a minor FSM is generated.
- The busy signal then is tested to determine both when the minor FSM is done and is available for the next computation.



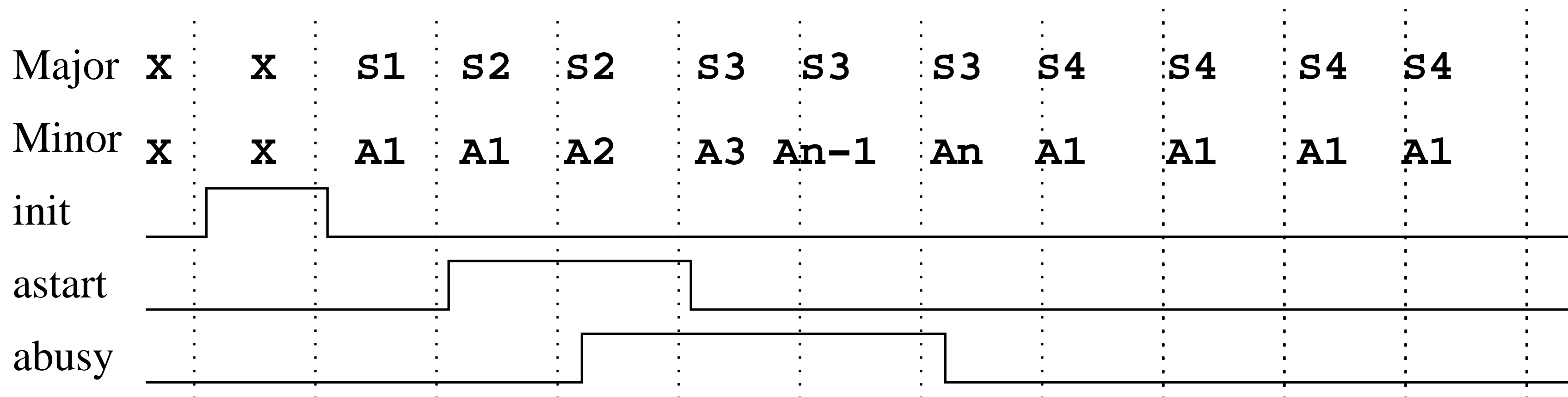


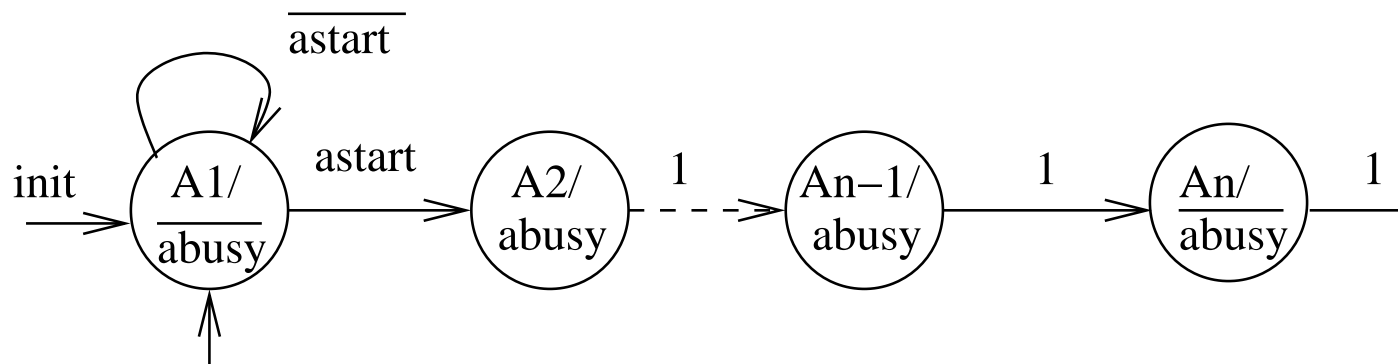
Go on to the next thing.

 Copyright © 2003 by MIT

Minor FSMs

- The minor FSM provides a status to the major FSM.
  - If the minor FSM is NOT busy then it is ready to be started.
- The makeup of the minor FSM is arbitrary.
  - The shortest path back to the starting state (reached by init) must be at least two clock cycles.

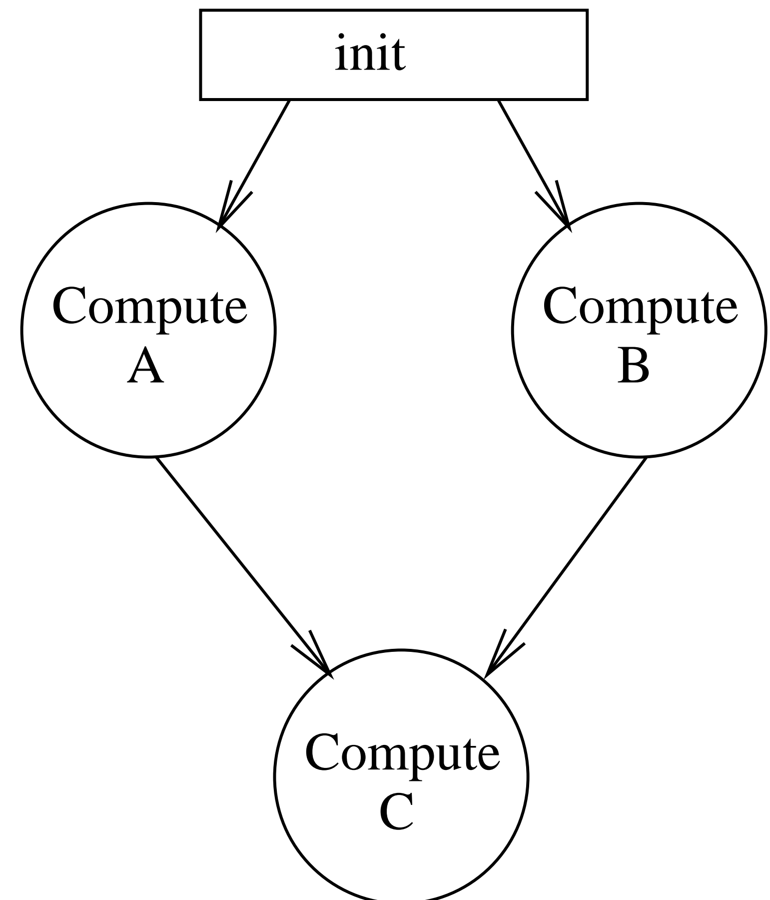




 Copyright © 2003 by MIT

Example Computation

- Suppose one wants to perform computation A in parallel with computation B and then, when both are finished, perform computation C.



- We want to do this whenever a TICK occurs.
  - We assume that TICK is generated by another FSM (namely a counter with appropriate combinational logic).

 Copyright © 2003 by MIT

Example Flow Diagram

- **Wait for TICK to happen.**
- **Check to see if previous computations have finished.**
  - **If not, go to an error state and wait there until reinitialized.**
- **Start both A and B. Wait until they are both done. Then do C. When it is done, wait for the next TICK.**

Example Major FSM

- **Here is a major FSM for the flow diagram.**
  - **Minor FSMs for A, B, and C are similar to minor FSMs previously shown.**
- **Sample waveforms are shown in the next slide.**

abusy+bbusy+cbusy
Error
1
$\overline{abusy}$ + $\overline{bbusy}$
abusy + bbusy
$\overline{tick}$
init
wt
tick
ck
else
sab/
astart,
bstart
else
wab
else
cbusy
$\overline{cbusy}$
$\overline{cbusy}$
wc
cbusy
sc/
startc

 Copyright © 2003 by MIT

Major FSM Waveforms for Example

- bbusy, and cbusy are shown as three clock periods each in order to save room. They need not be the same.
- Their lengths could vary subject to the constraint that they are two clock periods or more.

State X wt ck sab sab wab wab wab sc sc wc wc wc wt ck sab

init

tick

astart

abusy

bstart

bbusy

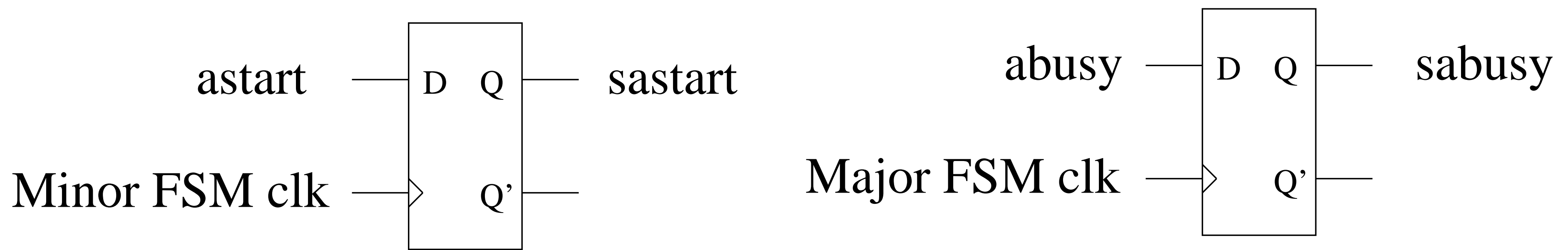
cstart

cbusy

 Copyright © 2003 by MIT

Different Clocks

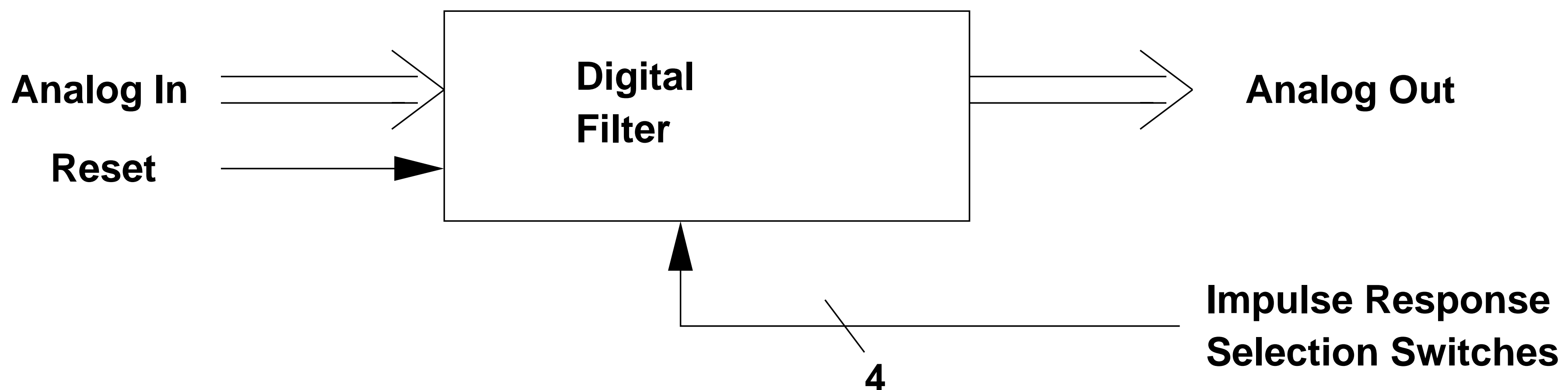
- **Does one or more minor FSMs use different clocks?**
- **Suppose the clocks are even at different frequencies.**
- **The answer is deceptively simple.**
  - **It is easy to implement and extremely difficult to debug.**
- **Simply synchronize the start and busy signals BEFORE they are used to control transitions in the receiving FSM.**
  - **Make sure you synchronize ALL signals that change asynchronously with your clock.**
  - **If in doubt, synchronize the signal.**



 Copyright © 2003 by MIT

Lab 3- Digital Filter

- Handles audio frequencies, such as music or speech.
- Input from waveform generator, microphone, 'boom box'.
- Output to an oscilloscope or speaker.
  - Or to another student's kit to cascade filters.
- We provide a variety of filters. Use four switches on your kit to select which one is used.

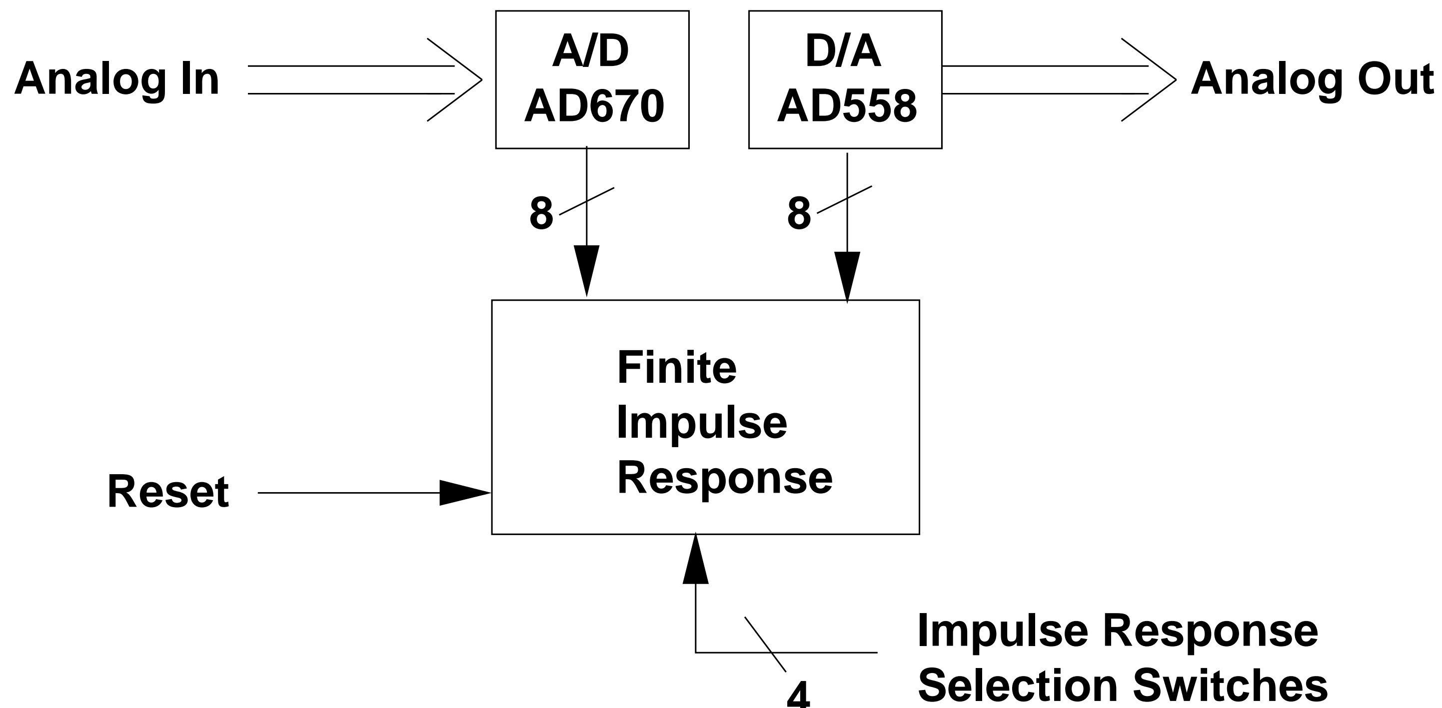


 Copyright © 2003 by MIT

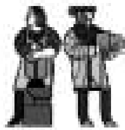
Analog – Digital Separation

- The input is both sampled and converted to a digital number by the A/D converter.
- The output of the (digital) FIR (Finite Impulse Response) is converted to an analog signal by the D/A converter.

More on the A/D and D/A next time.



 Copyright © 2003 by MIT



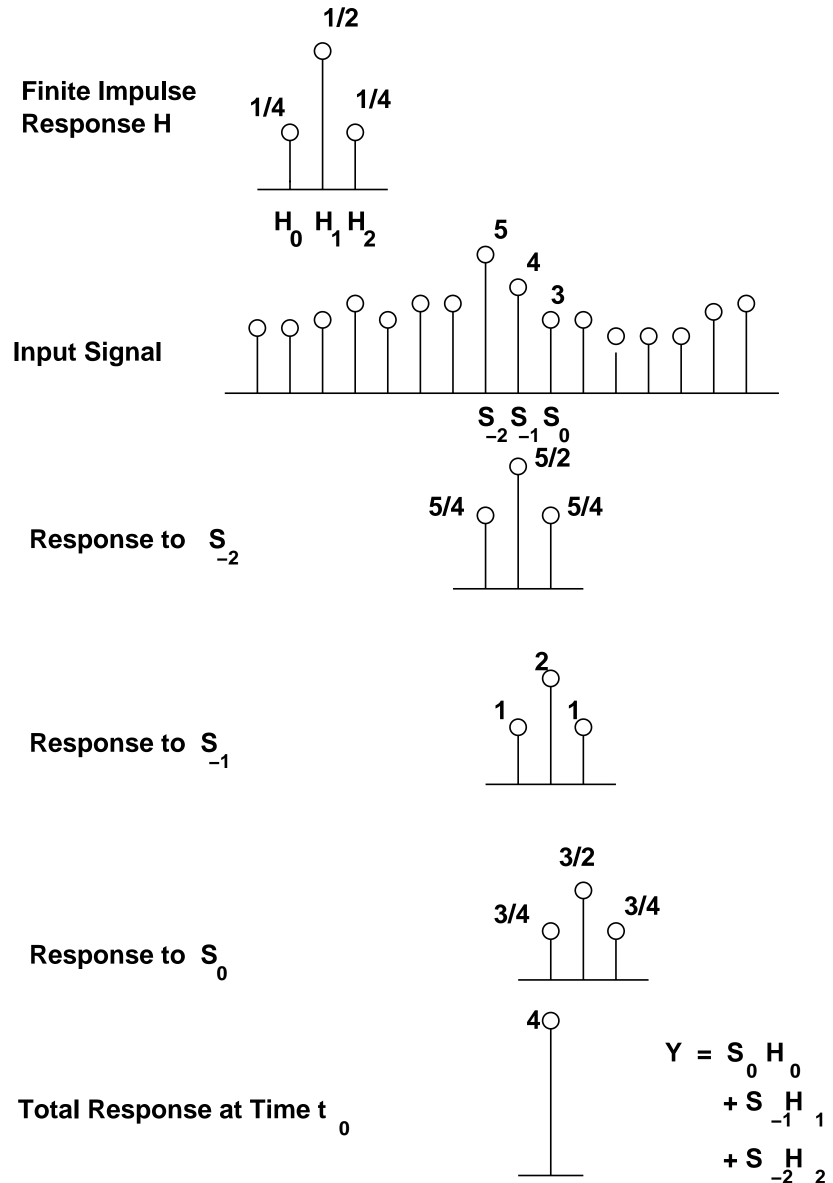
Main Loop

- Wait for a fixed time interval.
- Sample the input signal.
- Output what was computed on the last sample.
- Store the sample just taken.
- Start the next A/D conversion.
- Do the FIR filter algorithm which is a convolution.
- Go back for more...

Convolution

- **Convolution is a weighted sum of the past n samples.**
  - Where n is the number of impulse response coefficients.
  - Think of the impulse response being scaled and shifted by each sample.
  - Then the total response at a particular time is the sum of the scaled, shifted impulse response.

- **An FIR filter can implement almost all filtering functions.**
  - High Pass
  - Low Pass
  - Band Pass
  - All Pass



 Copyright © 2003 by MIT

Physical Block Diagram

- You can implement this FIR filter with three chips.
  - Analog Devices AD670 Analog to Digital Converter
  - Analog Devices AD558 Digital to Analog Converter
  - Altera FLEX 10K FPGA
    - It will fit into an EPF10K10 (the left hand FPGA).
    - If necessary, you may use the EPF10K70 (the right hand FPGA).

 Copyright © 2003 by MIT

Functional Block Diagram

The sample timer figures out when to start each operation.

The A/D converter gets a digital value from the analog signal.

The storage, arithmetic, and impulse response ROM are used to compute the outputs by convolution.

Data paths do not need to be as shown. Likely the A/D and D/A should share an 8 bit bus.

The D/A converts the output to an analog signal.

Clock

Sample Timer

sample

A/D

status

control

8

Storage Unit

status

control

address 4

8

Impulse Response ROM

status

control

address 4

8

Arithmetic Unit

status

control

8

D/A

status

control

8

Data Bus

Finite State Machine(s)

Copyright © 2003 by MIT

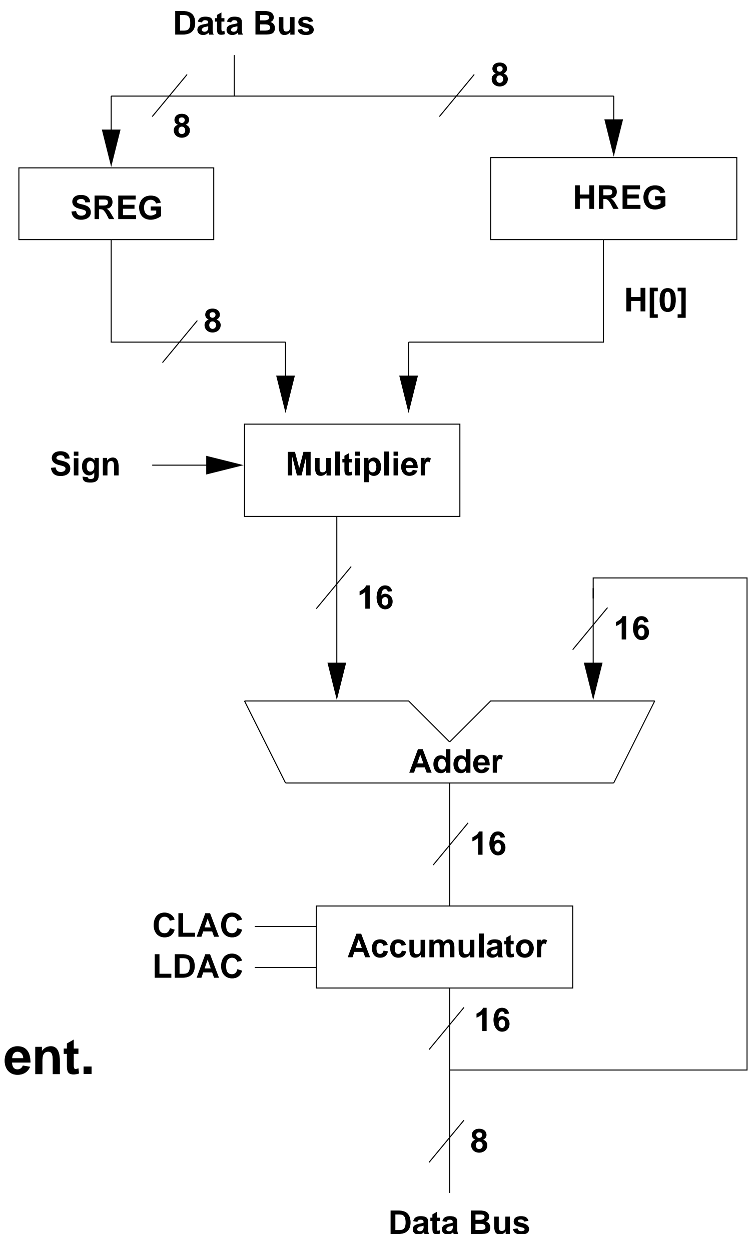
Arithmetic Block Diagram

Convolution is a series of multiplications and summations. These are (fairly) easily done with this type of circuit.

The HREG holds the impulse response coefficient in sign/magnitude form. The low order 7 bits go to the multiplier. What does the multiplier get for the high order bit of the impulse response?

The sign bit is used to invert either the input or the output. Which should you do? Should the inversion be twos complement or ones complement?

The accumulator sums up the product of the signal and the impulse response coefficient.



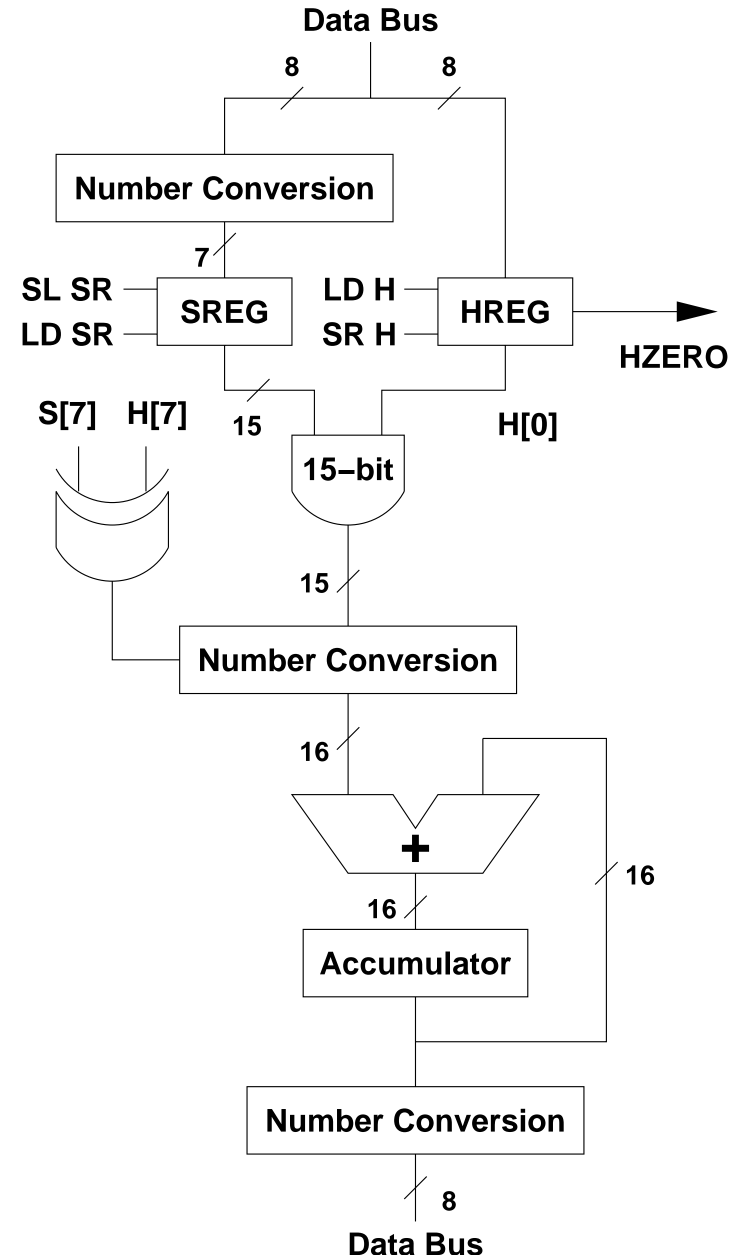
 Copyright © 2003 by MIT

Expanded Arithmetic Block Diagram

Here is one way to do the arithmetic of summing multiplications.
It is easy to multiply an input sample by an impulse response if both are positive. AND the lsb of HREG with SREG to do one step in the multiply. Then shift HREG right and SREG left. Repeat this for 7 cycles.

It is easy to take the magnitude of the impulse response. You need to convert the signal value if it is negative. How much error would you have if you used ones complement instead of twos complement?

At the very end you need to convert the output to offset binary as that is the only format accepted by the D/A converter.



 Copyright © 2003 by MIT

A Search for More Efficiency

An XOR gate can be thought of as a programmable inverter. If the control input is one then the output is inverted; otherwise, the output is the same as the input.

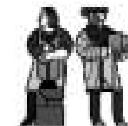
Adding one to form the twos complement can be done by using the carry in to the accumulator.

This scheme can be more efficient.
It also can be more confusing!

DATA BUS
Sign/Magnitude
8
Twos Complement
8
1
H[7]
HREG
LDH
SLH (7 LSB Only)
H[0]
8
SR
LD SR
SL SR
15
16
+
CIN = H[7]
16
CLAC
LDAC
Accumulator
7
8
DATA BUS

 Copyright © 2003 by MIT

Impulse Responses



- The first few impulses are for debugging
  - The first one is a unit sample which should produce an output just like the input.
  - The second is a negative unit sample so the output should be the negative of the input.
  - The third consists of 16 identical values – a boxcar filter.
- Here are the outputs for two different filters.

INPUT SQUARE WAVE

OUTPUT OF "BOX CAR" FILTER

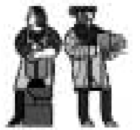
OUTPUT OF "EXPONENTIAL" FILTER

 Copyright © 2003 by MIT

Use LPM to Create ROM/RAM

- Click on File->MegaWizard Plug-In Manager
  - This starts up a series of windows so that you can specify parameters of the LPM module. You can choose
    - ROM
    - RAM
      - dp- Dual Ported
      - dq- Separate Inputs and Outputs
      - io - TriState Inputs and Outputs (like the 6264)
  - In all of these, you can specify the number of address bits, i.e., the width (in powers of two).
  - You can also specify the word size.
  - You can specify a file to set the initial value of the RAM or ROM.
    - This is especially useful for ROM.
  - You can choose registered or unregistered inputs, outputs, and addresses.
    - Be careful in using (or simulating) registered addresses.
      - The actual location specified is the previous address input. This may not be what you meant.
  - Use of registered inputs and outputs also changes the timing.

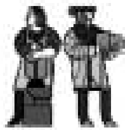
ROM Contents



- **Prepare an xxx.dat file.**
  - You can type this in, write a computer program, get it from another application (speech or graphics, etc.)
  - This has numbers separated by white space.
    - The default base is HEX but you can use binary or decimal if you include the following statement (before the numbers).

    # BASE = BINARY;
  - INSERT   # SET_ADDRESS = 0;
    - This says the data should start at address 0.
- **Run dat2ntl to format your xxx.dat file into Intel HEX.**
  - for details type   man dat2ntl
  - dat2ntl xxx.dat xxx.ntl   or   dat2ntl xxx.dat xxx.hex
  - Or, you can use a file we have already prepared.

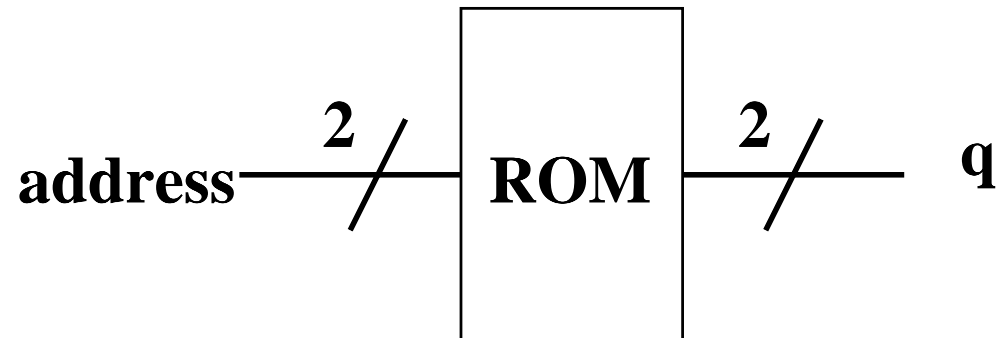
    /mit/6.111/handouts/labs/lab3.f2003/impulses.ntl   or
    /mit/6.111/handouts/labs/lab3.f2003/impulses.hex   the same bits
  - You may have to use impulses.hex to satisfy MAX+plusII



rom2.vhd

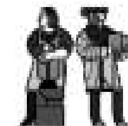
```
-- megafunction wizard: %LPM_ROM%
-- MODULE: lpm_rom
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY rom2 IS PORT (
  address : IN
    STD_LOGIC_VECTOR (2 DOWNTO 0);
  q       : OUT
    STD_LOGIC_VECTOR (7 DOWNTO 0));
END rom2;


ARCHITECTURE SYN OF rom2 IS
  SIGNAL sub_wire0 :
    STD_LOGIC_VECTOR (7 DOWNTO 0);
  COMPONENT lpm_rom
  GENERIC (lpm_width : NATURAL;
          lpm_widthad      : NATURAL;
          lpm_address_control : STRING;
          lpm_outdata              : STRING;
          lpm_file                 : STRING);
   PORT (address : IN
      STD_LOGIC_VECTOR (2 DOWNTO 0);
          q      : OUT
     STD_LOGIC_VECTOR (7 DOWNTO 0));
   END COMPONENT;
```

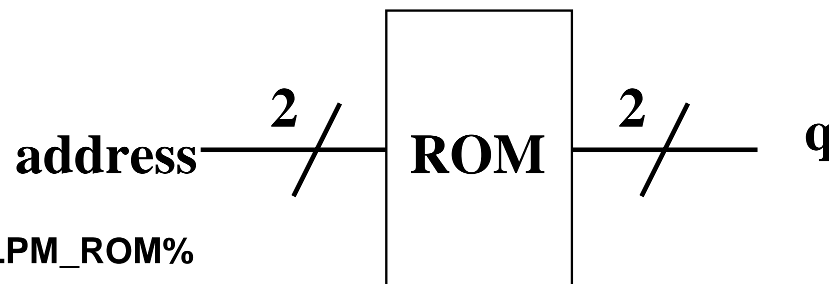



```
BEGIN
  q <= sub_wire0(7 DOWNTO 0);
  lpm_rom_component : lpm_rom
    GENERIC MAP (LPM_WIDTH => 8,
              LPM_WIDTHAD => 3,
              LPM_ADDRESS_CONTROL
                => "UNREGISTERED",
              LPM_OUTDATA
                => "UNREGISTERED",
              LPM_FILE =>
  "C:/Documents and Settings/troxel/
         s2003/13/rom/data.hex")
    PORT MAP (address => address,
            q => sub_wire0);
END SYN;
```

ROM Simulation

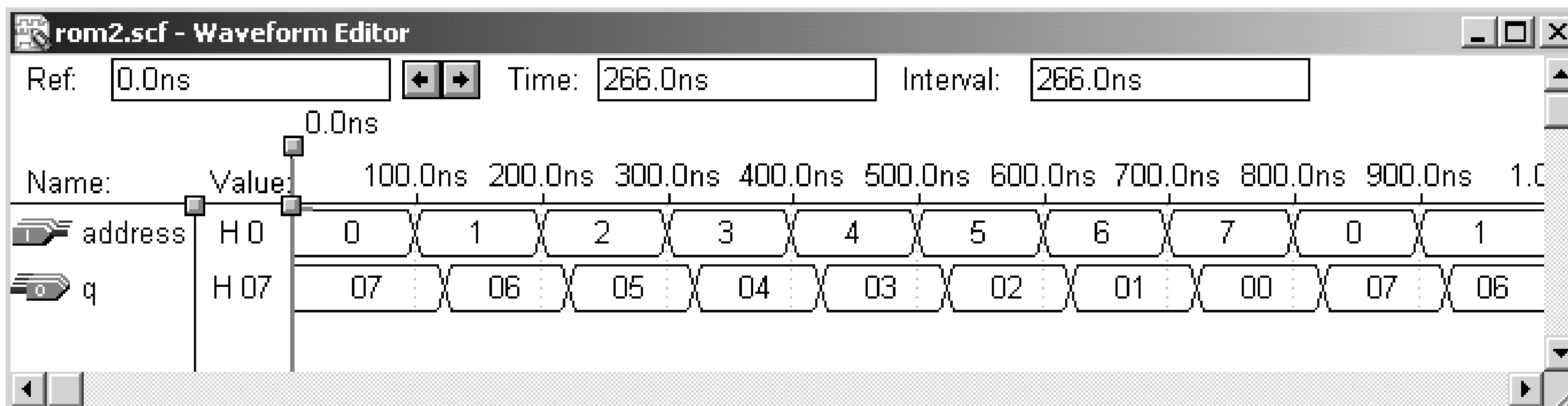


:08000000070605040302010 0DC
:00000001FF

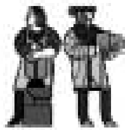


SET_ADDRESS = 0;
7
6
5
4
3
2
1
0

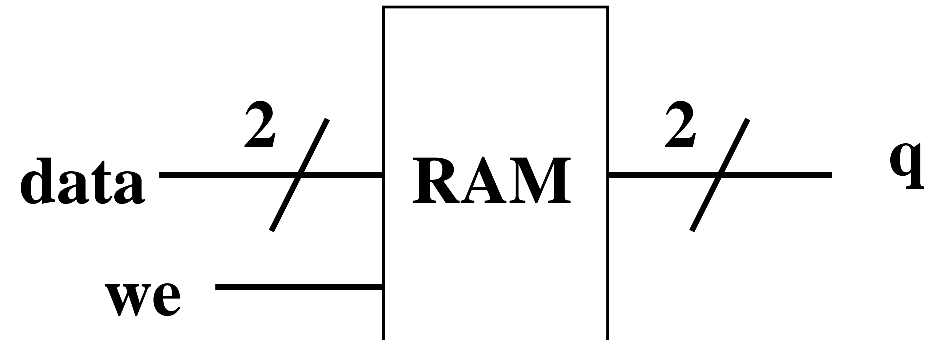
```
-- megafunction wizard: %LPM_ROM%
-- MODULE: lpm_rom
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY rom2 IS PORT (
address : IN STD_LOGIC_VECTOR (2
DOWNTO 0);
q       : OUT STD_LOGIC_VECTOR (7
DOWNTO 0));
END rom2;
```



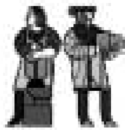
 Copyright © 2003 by MIT



ram2.vhd

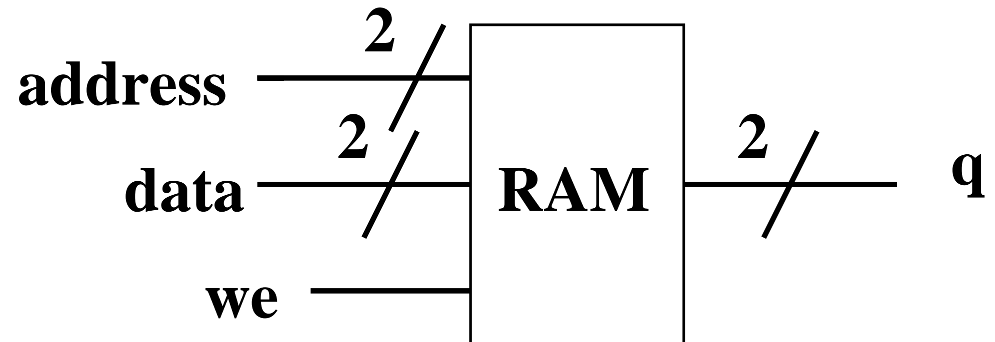


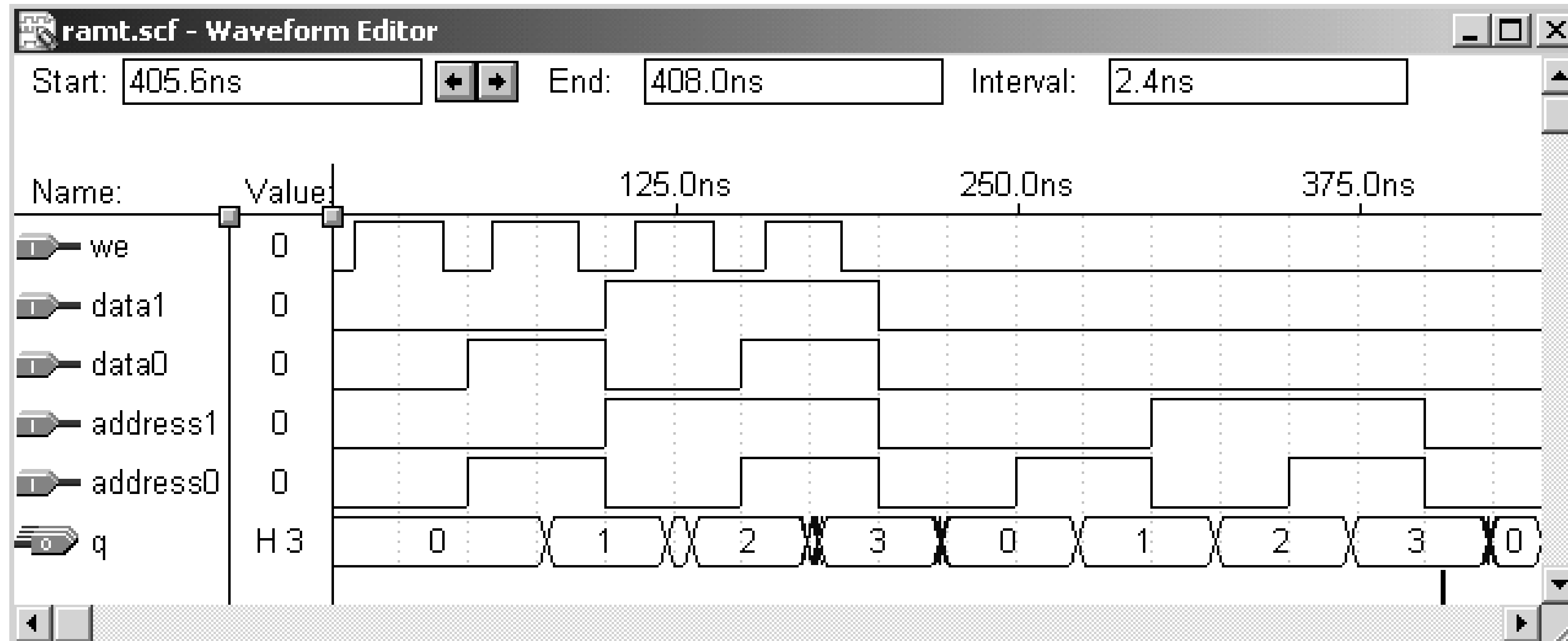

```
-- megafunction wizard: %LPM_RAM_DQ%
-- MODULE: lpm_ram_dq
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY ram2 IS PORT (
  address : IN
    STD_LOGIC_VECTOR (1 DOWNTO 0);
  we : IN STD_LOGIC  := '1';
  data : IN
    STD_LOGIC_VECTOR (1 DOWNTO 0);
  q : OUT
    STD_LOGIC_VECTOR (1 DOWNTO 0));
END ram2;
ARCHITECTURE SYN OF ram2 IS
  SIGNAL sub_wire0:
    STD_LOGIC_VECTOR (1 DOWNTO 0);
  COMPONENT lpm_ram_dq
    GENERIC (lpm_width : NATURAL;
       lpm_widthad     : NATURAL;
       lpm_indata      : STRING;
       lpm_address_control : STRING;
       lpm_outdata         : STRING;
       lpm_hint            : STRING);
    PORT (address : IN
      STD_LOGIC_VECTOR (1 DOWNTO 0);
             q : OUT
      STD_LOGIC_VECTOR (1 DOWNTO 0);
             data : IN
      STD_LOGIC_VECTOR (1 DOWNTO 0);
             we   : IN STD_LOGIC);
  END COMPONENT;
BEGIN
        q <= sub_wire0(1 DOWNTO 0);
        lpm_ram_dq_component : lpm_ram_dq
        GENERIC MAP (LPM_WIDTH => 2,
          LPM_WIDTHAD => 2,
          LPM_INDATA => "UNREGISTERED",
          LPM_ADDRESS_CONTROL =>
             "UNREGISTERED",
          LPM_OUTDATA => "UNREGISTERED",
          LPM_HINT => "USE_EAB=ON"
        )
        PORT MAP (address => address,
                  data => data,
                  we => we,
                  q => sub_wire0);
END SYN;
```



RAM Simulation

```
-- megafunction wizard: %LPM_RAM_DQ%
-- MODULE: lpm_ram_dq
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY ram2 IS PORT (
  address : IN
    STD_LOGIC_VECTOR (1 DOWNTO 0);
  we : IN STD_LOGIC  := '1';
  data : IN
    STD_LOGIC_VECTOR (1 DOWNTO 0);
  q : OUT
    STD_LOGIC_VECTOR (1 DOWNTO 0));
END ram2;
```





 Copyright © 2003 by MIT