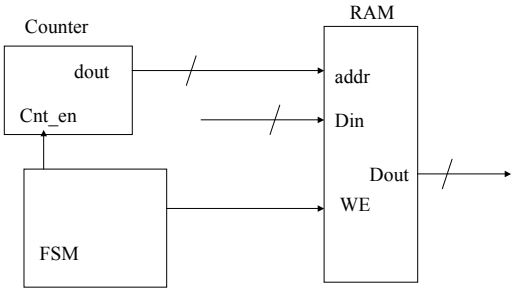
DataPath Elements

- Altera LPM library has many elements useful for building common datapath functions
  - lpm_ram_dq - recommended for either asynchronous or synchronous RAM. Uses EAB in Flex 10K family.
  - lpm_ram_io - recommended for asynchronous RAM. Uses EAB in Flex 10K in family.
  - lpm_compare - comparing two values. Outputs are AEQB, ALB, ALEB, AGB, AGEB
  - lpm_counter - versatile counter function

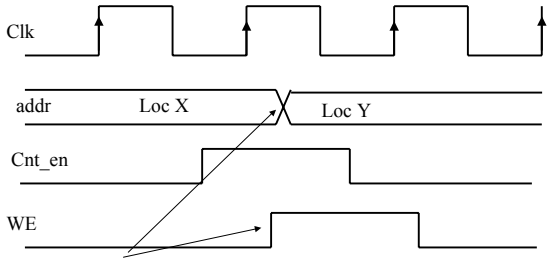
Synchronous vs Asychronous RAM

- Asynchronous RAM looks like a combinational element
  - No Clock
  - Data available after propagation delay from address
  - Address MUST BE stable while WE (write enable) is high so that only ONE location is written too. Data must also be stable during write cycle.
- Synchronous RAM has a clock input and will latch input data and control lines (address, data)

Usually use Counters to Drive address Lines

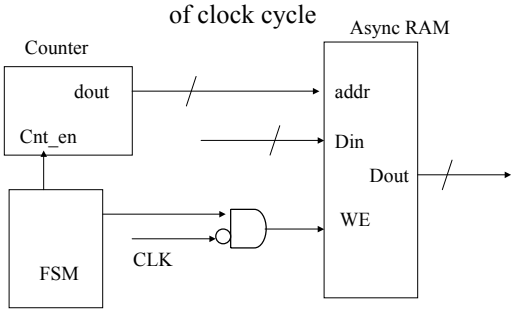


If WE, Address changes same Clock Cycle, may write multiple locations in Async RAM

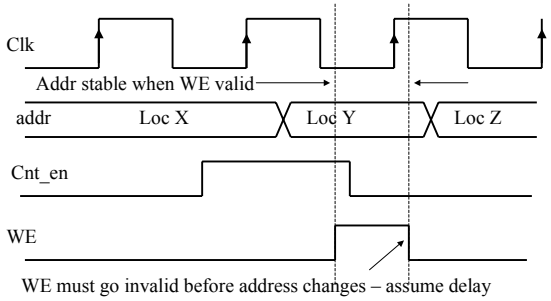


Delays can cause WE to change before or after address. If before, then can write to both Loc X and Loc Y

Make WE valid only during 2nd half of clock cycle

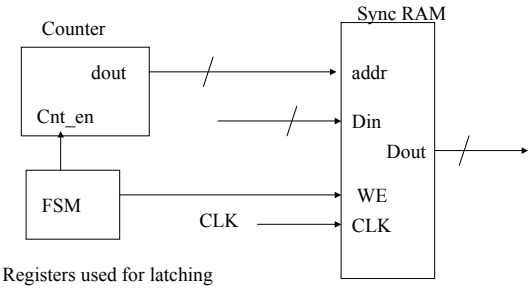


Make WE valid during 2nd half of Clock Cycle



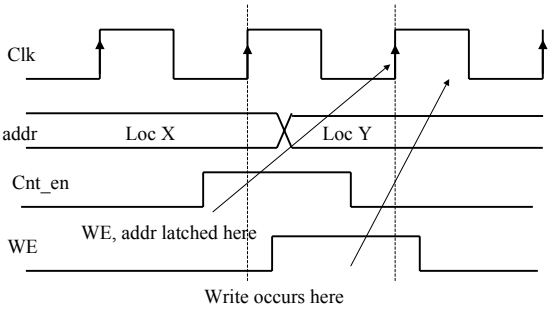
WE must go invalid before address changes – assume delay through AND gate less than register delay of address value.

Sync Ram latches control, addr, input data. Can also latch output data.



Registers used for latching input/output data are internal to RAM.

Sync RAM latches address, WE

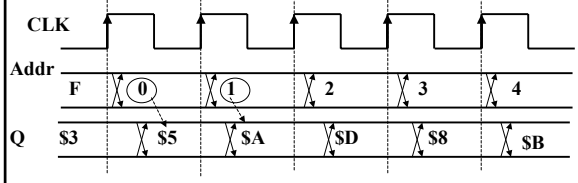


Will usually prefer Sync Ram

- Sync RAM is easier to use from a timing perspective but adds latency to operations
  - If address coming from counter, then have an extra clock cycle of latency from when counter value is updated to when RAM data is available for that address
- In our Lab exercises and class examples, will normally use synchronous RAM
  - Will not latch output data unless specifically needed
  - Options to latch control, input data, output data available on LPM_RAM_DQ

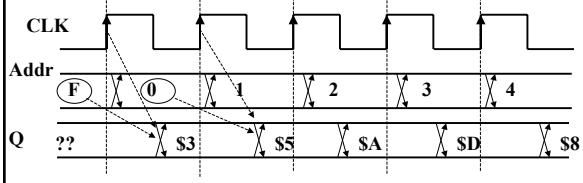
LPM_RAM_DQ Timing

Asynchronous – no latching
LPM_Address_Control= "Unregistered"
LPM_indata = "Unregistered"
LPM_Outdata = "Unregistered"
Assume 16x4 RAM
Contents: loc $F= 3, $0= 5, $1=:$A, $2= $D, $3 = $8, $4=$B



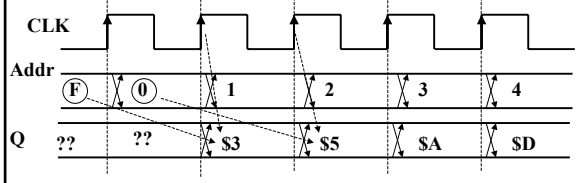
LPM_RAM_DQ Timing

Synchronous, latch input data and control
LPM_Address_Control= "Registered" (on 'inclock')
LPM_indata = "Registered" (on 'inclock')
LPM_Outdata = "Unregistered"
Assume 16x4 RAM
Contents: loc $F= 3, $0= 5, $1=:$A, $2= $D, $3 = $8, $4=$B



LPM_RAM_DQ Timing

Synchronous, latch input data, control, output data
LPM_Address_Control= "Registered" (on 'inclock')
LPM_indata = "Registered" (on 'inclock')
LPM_Outdata = "Registered" (on 'outclock')
Assume 16x4 RAM
Contents: loc $F= 3, $0= 5, $1=:$A, $2= $D, $3 = $8, $4=$B



Asynchronous vs Synchronous Control

- Some LPMS have both synchronous and asynchronous control lines
  - Counter has ‘aload’ (asynchronous load), and ‘sload’ (synchronous load); ‘aclr’ and ‘sclr’ (async and sync clear)
- Should always use a Synchronous control line if possible, especially if connected to a FSM output.
  - Any glitch on an asynchronous control line can trigger it
  - If using a FSM output for an asynchronous control, the output should come directly from a Flip-Flop output, NOT from combinational gating.

Sample Problem

- Create a synchronous RAM block that has a block transfer capability capability
- If ‘xfer’ input asserted, assert BUSY output and transfer WCNT # of word starting at FROM to locatoion TO .
- Counters WCNT, FROM, TO are loaded before transfer operation started.
- RAM size will 64 x 8

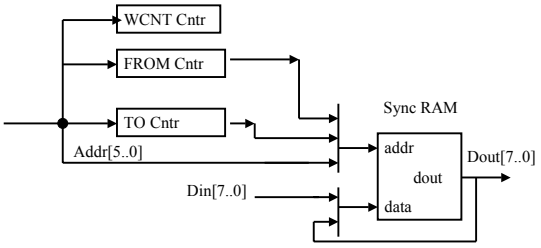
Interface

- Inputs
  - clk, reset
  - we - write enable for RAM
  - cmd_we - asserted when writing WNCT, TO, FROM registers. Register choice determined by Din1,0 bits: “00” →WCNT, “01” → FROM, “10” → TO .
  - DIN[7..0] Data bus to RAM
  - Addr[5..0] Address bus to RAM.
  - xfer - start a zero cycle
- Outputs
  - DOUT[7..0]
  - Busy - when assert, busy zeroing RAM

What Datapath Elements Do We need?

- Counters to hold WCNT, TO, FROM values
  - Use LPM_COUNTER
- Need the RAM (use LPM_RAM_DQ)
- Muxes (LPM_MUX or VHDL)

Datapath Block Diagram



Control lines not shown on datapath diagram

What Control Lines do we need from FSM? (look at each Datapath component)

Counters: Load lines for WCNT, TO, FROM registers driven externally and not under FSM control. Count enables for these counters need to be exercised by FSM. WCTN will be configured to count DOWN, the TO,FROM counters will count UP.

Mux Selects: When doing ‘xfer’ operation, counters will be driving RAM address lines and RAM input data line will be a feedback from the RAM output.

RAM: The WE of the RAM needs to be an OR of the external WE and a WE that is provided by the FSM.

FSM Interface

Inputs:

Clk, Reset

xfer - kicks off transfer operation

cnt_words[5..0] - WCNT counter value – need to check this to see if finished.

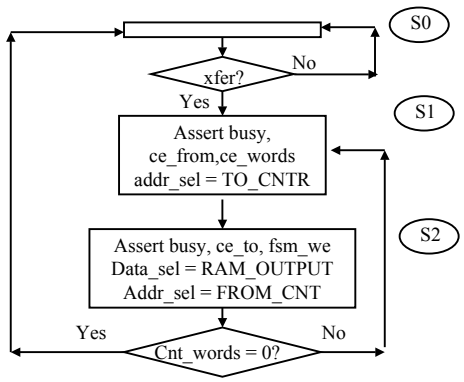
Outputs:

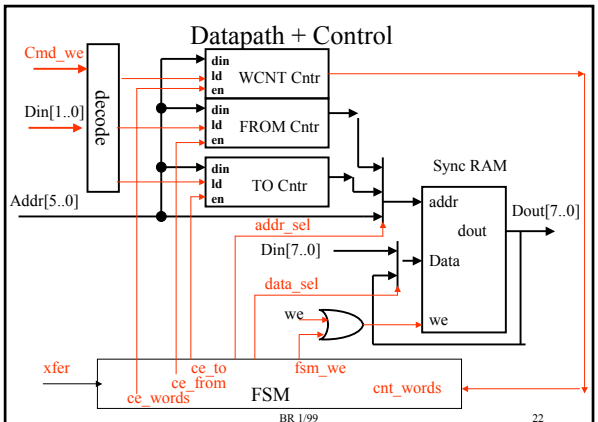
- *busy* – busy output
- *addr_sel[1..0]* -- mux select line for addr muxes
- *ce_from, ce_to, ce_words* -- count enables for FROM, TO, WCNT counters
- fsm_we -- WE to RAM
- data_sel – mux select line for RAM input data mux

What operations do we need for FSM?

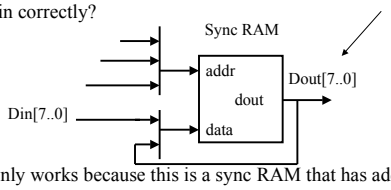
- Wait for XFER command (FSM simply waits for '*xfer*' input to be asserted).
- Read a value from RAM using TO counter address; increment the TO counter and decrement the WCNT counter
- Write data value to RAM via FROM address counter; increment the FROM counter. Loop to read state unless WCNT counter value = 0.

Three DISTINCT operations, need three STATES in FSM. Cannot do both a Read and Write in the same clock cycle.





Comments: Why does the feedback path from Dout to Din correctly?



It only works because this is a sync RAM that has addr, input data, and control registered.

The DOUT value are the memory contents corresponding to the address from the last clock edge (state S1, 'to' address).

The current clock edge (state S2) will latch both this data and current address ('from' counter value) for the write operation.

WOULD NOT WORK IF ASYNC RAM.

Design Implementation

- Once the datapath block diagram and ASM chart is done, the 'design' work is done. What is left is implementation.
- Decide what parts of the datapath will be implemented in VHDL, what parts using LPMs
  - The FSM control will certainly be done in VHDL
  - Misc registers can be easily included in the VHDL FSM code as well instead of using separate datapath components.
- Do the schematic of the datapath FIRST!
  - Sometime just hooking up the datapath elements will expose a flaw in your reasoning.

Design Implementation (cont.)

- After Datapath is finished, do FSM VHDL code
  - ALWAYS bring the FSM state value out as an external output for debugging purposes!!!
  - Should be able to write FSM code directly from ASM chart
- DEBUG - take a systematic approach
  - Your design will NOT WORK the first time - be prepared to debug.
  - Attach external pins to as many internal nets as possible so that you can observe the internal net values
  - Debug your design ONE state at a time. Do not test the next state until the current state works as expected.

Design Implementation (cont.)

- Until you get more confident with VHDL, should use as many LPM components as you can
  - Can easily examine input/outputs to LPMs in waveform viewer so makes it easier to debug
- Always use a VERY LONG clock cycle to start out with so that you do not encounter timing problems
  - To be absolutely safe, make external inputs change on the falling edge if your internal logic is rising edge triggered (this gives you 1/2 clock of setup time).

The remaining slides are from an older discussion about a RAM with zeroing capability.

The implementation is similar to that of the RAM with block transfer capability.

The Zero'ing RAM shows an alternate method for implementing the busy flag.

A separate register is used to hold the LOW value instead of simply using the counter for this storage. The only advantage to this is that the LOW value is preserved after a zero operation and could be reused.

Sample Problem

- Create a synchronous RAM block that has a 'zeroing' capability
- If 'zero' input asserted, assert BUSY output and zero RAM block
- Can load a LOW value, and HIGH value that will set the RANGE of the memory to zero.
- RAM size will 64 x 8

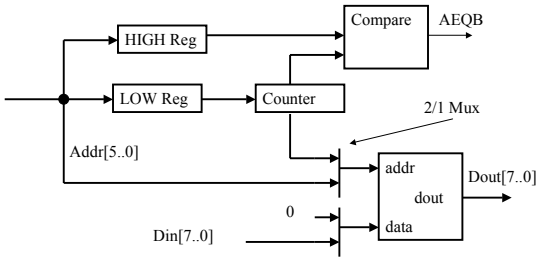
Interface

- Inputs
  - clk, reset
  - low_ld - load LOW value; will be taken from address bus
  - low_high - load HIGH value; will be taken from address bus
  - DIN[7..0] Data bus to RAM
  - Addr[5..0] Address bus to RAM.
  - Zero - start a zero cycle
- Outputs
  - DOUT[7..0]
  - Busy - when assert, busy zeroing RAM

What Datapath Elements Do We need?

- Two registers to hold LOW, HIGH value
  - Use LPM_DFF or write VHDL model (reg6.vhd)
- Need a 6-bit counter to cycle address lines of RAM
  - LPM_COUNTER
  - Counter needs to be loaded with LOW value when we start to zero the RAM
- Need a Comparator to compare Counter value and HIGH value to see if we are finished
- Need the RAM (use LPM_RAM_DQ)
- Muxes (LPM_MUX or VHDL)

Datapath Block Diagram



Control lines not shown on datapath diagram

What Control Lines do we need from FSM? (look at each Datapath component)

Registers: Load lines for LOW, HIGH registers driven externally and NOT under FSM control.

Counter: sload (synchronous load), cnt_en (count enable). Counter will be configured to only count up.

Mux Selects: When doing 'zero' operation, counter will be driving RAM address lines and RAM input data line will be zero. The same select line can drive both muxes.

RAM: The WE of the RAM needs to be an OR of the external WE and a WE that is provided by the FSM.

FSM Interface

Inputs:

Clk, Reset
Zero - kicks off zero operation
Cnt_eq - AEB output from comparator

Outputs:

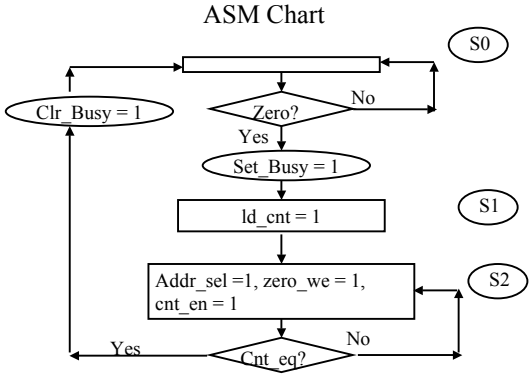
set_busy, clr_busy -- set/clr output for busy flag (JK-FF)
addr_sel -- mux select line for addr, data muxes
cnt_en -- count enable for counter
ld_cnt -- synchronous load line for counter
zero_we -- WE output of FSM, will be OR'ed with external WE

What operations do we need for FSM?

- Wait for ZERO command (FSM simply waits for 'ZERO' input to be asserted).
- Load the counter with the LOW value
- Write '0' data value to RAM via address specified by counter, incrementing counter each clock cycle. Stop writing when HIGH register value equals counter value.

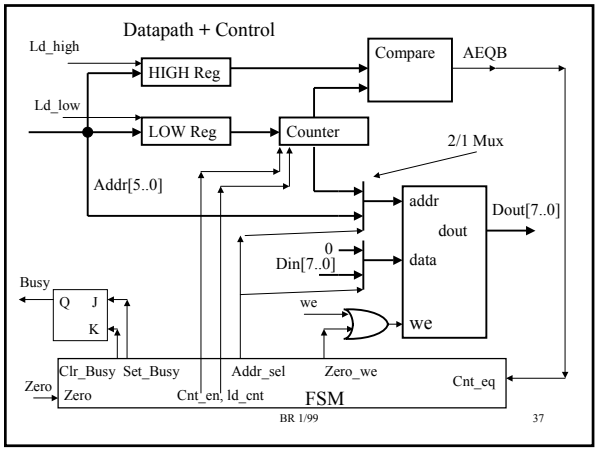
Three DISTINCT operations, need three STATES in FSM.

ASM Chart



ASM States

- Three States
- State S0 waits for Zero operation. In this state the external addr, data, we lines are muxed to RAM. Set busy flag on transition to State S0.
- State S1 loads counter with LOW register value
- State S2 does zero operation. Exit this state with counter value equals to HIGH register value. On state exit, clear the busy flag output (conditional output).
  - We will spend HIGH-LOW+1 clocks in this state (clear LOW to HIGH locations inclusive)


Datapath + Control
Ld_high
HIGH Reg
Compare
AEQB
Ld_low
LOW Reg
Counter
2/1 Mux
Addr[5..0]
addr
dout
Dout[7..0]
0
Din[7..0]
data
Busy
Q
J
K
we
we
Clr_Busy
Set_Busy
Addr_sel
Zero_we
Cnt_eq
Zero
Zero
Cnt_en, ld_cnt
FSM
BR 1/99
37