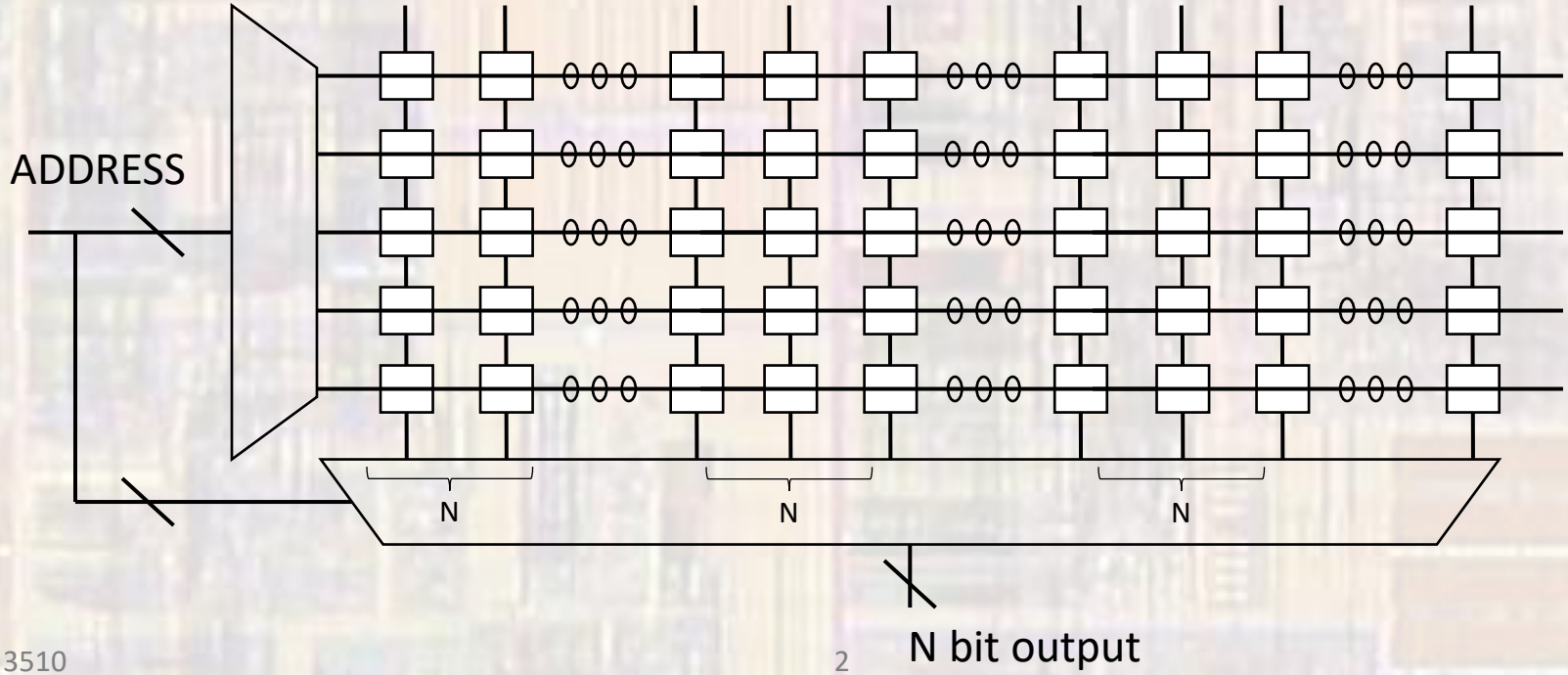
SRAM Controller

Last updated 10/6/26

SRAM Controller

- General Memory Topology
  - Array of n bit cells
  - Row decoder chooses 1 row
  - Column decoder chooses one N bit ‘column’
    - 1,4,8,16,32,64,128,… bits/column



© tj

SRAM Controller

- Memory Descriptive Terminology
  - 16Mb in a x4
    - 16Mb total memory, each memory address provides 4 bits
  - 16Mb x 4
    - 64Mb total memory, each memory address provides 4 bits
    - Could be configured as 4 16MB memories, each providing 1b to the output

© tj

SRAM Controller

- Example – 16Mb in a x4 configuration

x4 means each column is 4 bits

or each address points to 4 bits

16Mb → 16,777,216 bits

16Mb in a x4 configuration → 4 bits / address → 4,194,304 - individual addresses

4,194,304 addresses → 22 address bits

x4 means 4 bit cells for every column

Assuming a square memory array and a square bit cell → 4 times as many rows as columns

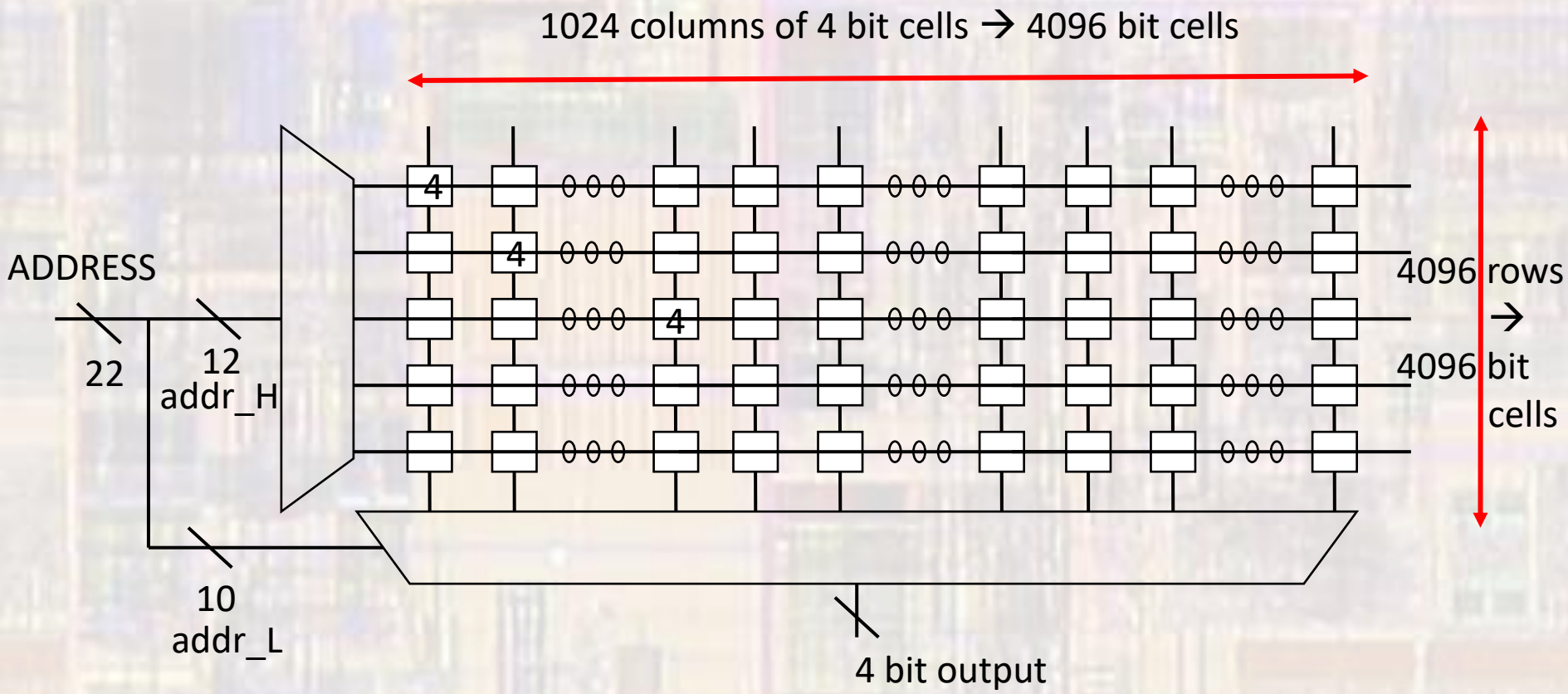
4 times as many rows as columns → 2 more row address bits than column address bits

22 address bits → 12 bits for row addresses and 10 bits for column addresses

© tj

SRAM Controller

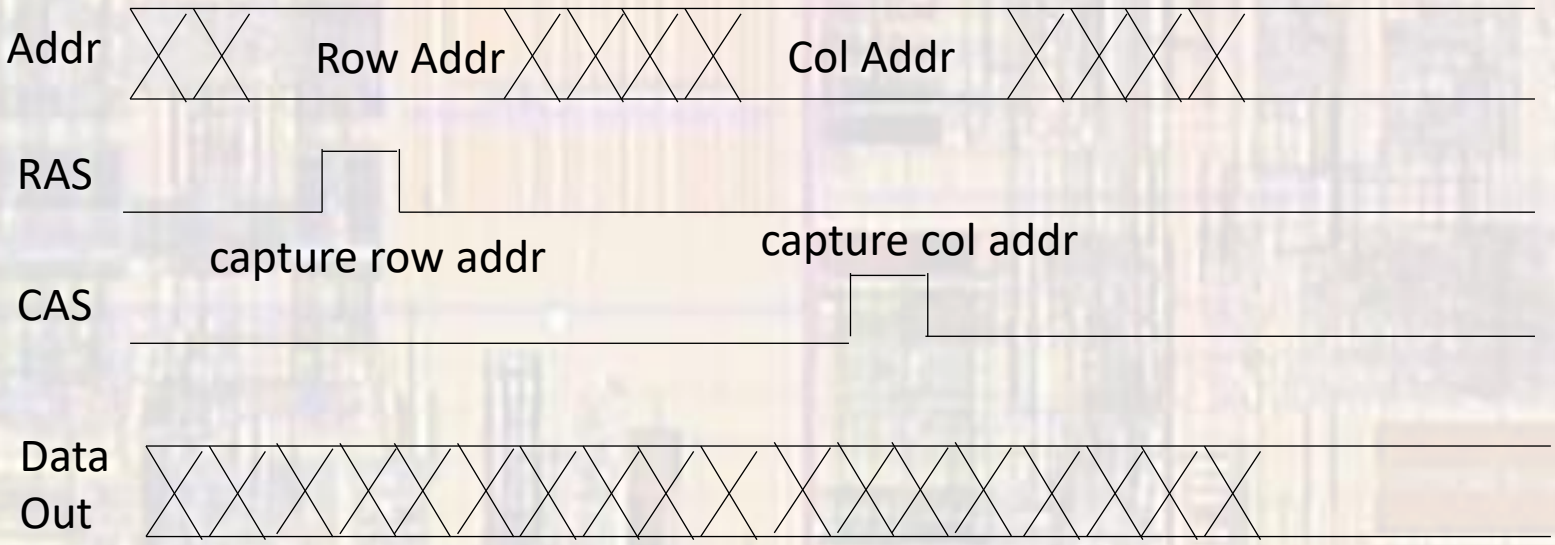
- Example – 16Mb in a x4 configuration



© tj

SRAM Controller

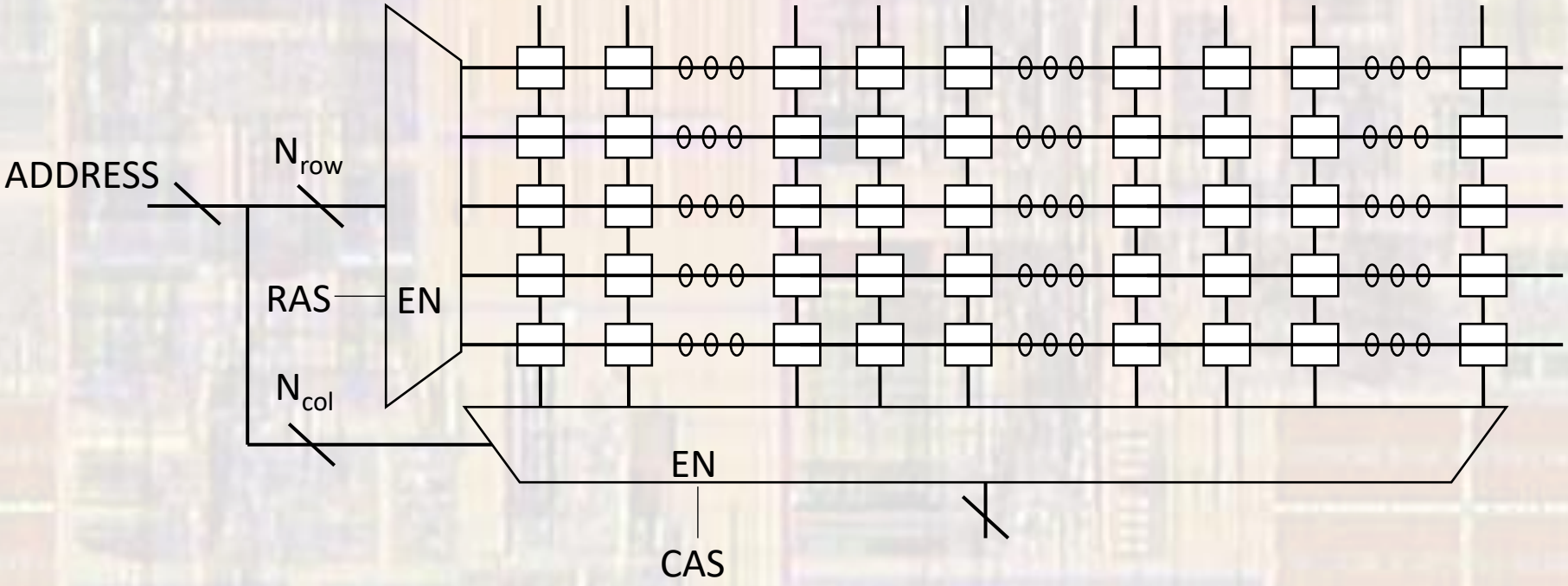
- Address Line Sharing
  - Reduce I/O by sharing the address inputs between Row addresses and Column addresses in time
  - RAS – Row Address Strobe (current address is for row decode)
  - CAS – Column Address Strobe (current address is for column decode)



© tj

SRAM Controller

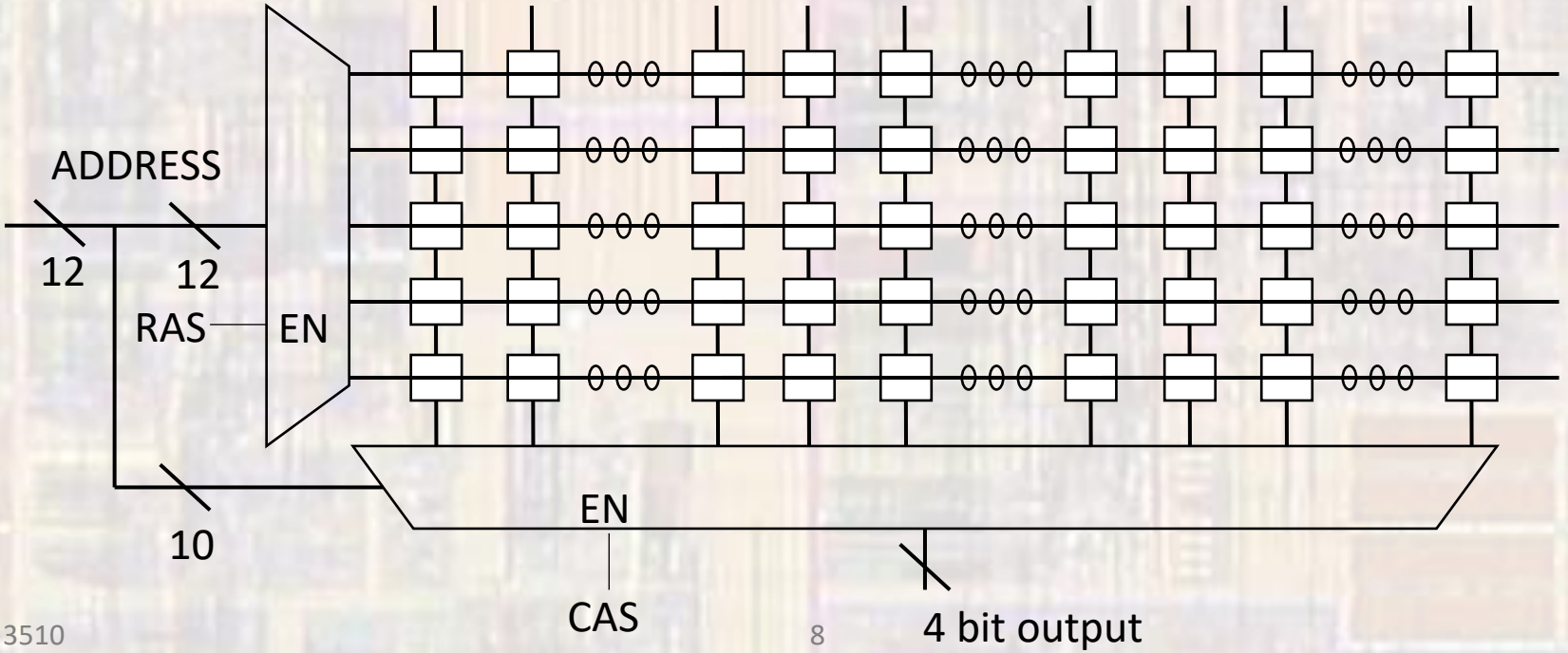
- Address Line Sharing
  - Reduce I/O by sharing the address inputs between Row addresses and Column addresses – in time
  - Number of address bits is the greater of the number of row address bits and column address bits



© tj

SRAM Controller

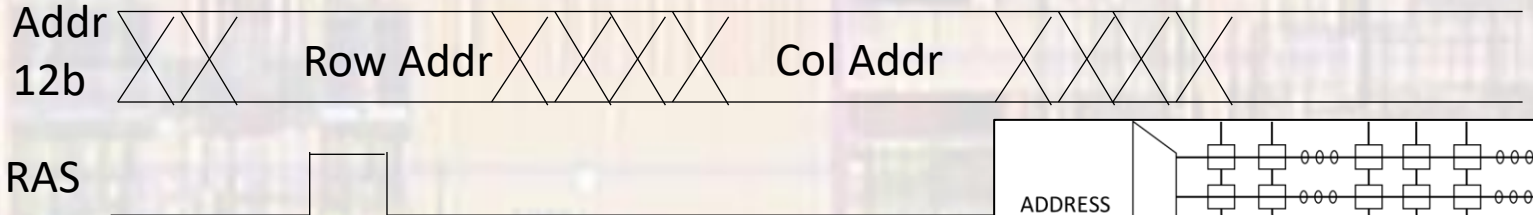
- Address Line Sharing
  - Reduce I/O by sharing the address inputs between Row addresses and Column addresses
  - 22 bit addr → 12 bit addr + RAS and CAS → 14 wires
  - RAS – Row Address Strobe (current address is for row decode)
  - CAS – Column Address Strobe (current address is for column decode)

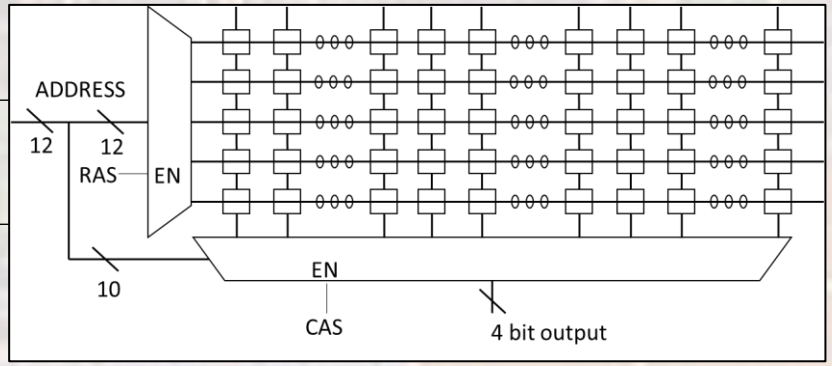


© tj

SRAM Controller

- Address Line Sharing
  - Reduce I/O by sharing the address inputs between Row addresses and Column addresses
  - 22 bit addr → 12 bit addr + RAS and CAS → 14 wires
  - RAS – Row Address Strobe (current address is for row decode)
  - CAS – Column Address Strobe (current address is for column decode)





© tj

SRAM Controller

- Entity

```
------------------------------------------------
--
-- sram_controller.vhdl
--
-- created 10/5/26 by tj
--
------------------------------------------------
-- Simple RAS/CAS memory controller
--
-- Generics: row address width, col address width, data width
--             row address width must be greater than column address width
--
-- inputs: clk, rstb, address, data in, read, write
-- outputs: busy, done, rasN, casN, weN
--          address to sram, data to sram
--
------------------------------------------------


library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
```

```
entity sram_controller is

   generic(
         -- ROW_ADDRESS_WIDTH MUST BE > COL_ADDR_WIDTH
         ROW_ADDR_WIDTH:   positive := 6;
         COL_ADDR_WIDTH:   positive := 4;
         DATA_WIDTH:       positive := 16
         );

   port(
         i_clk:      in std_logic;
         i_rstb:     in std_logic;

         i_address:  in std_logic_vector(((ROW_ADDR_WIDTH + COL_ADDR_WIDTH) - 1) downto 0);
         i_data_in:  in std_logic_vector((DATA_WIDTH - 1) downto 0);
         i_read:     in std_logic;
         i_write:    in std_logic;

         o_busy:     out std_logic;
         o_done:     out std_logic;

         -- SRAM interface
         o_ras_n:    out std_logic;
         o_cas_n:    out std_logic;
         o_we_n:     out std_logic;
         o_address:  out std_logic_vector((ROW_ADDR_WIDTH - 1) downto 0);
         o_data_to_sram:   out std_logic_vector((DATA_WIDTH - 1) downto 0)
      );
end entity;
```


© tj

SRAM Controller

- Internal Signals

```
architecture behavioral of sram_controller is

   type state_type is (Idle, Row, Column, Complete);

   signal state:        state_type;
   signal state_next:   state_type;

   signal address_reg:  std_logic_vector(((ROW_ADDR_WIDTH + COL_ADDR_WIDTH) - 1) downto 0);
   signal data_reg:     std_logic_vector((DATA_WIDTH - 1) downto 0);

   -- reg to track if a write   0 = READ     1 = WRITE
   signal write_reg : std_logic;

begin
```


© tj

SRAM Controller

- Input Registers

```
-- input registers
input_reg: process(i_clk, i_rstb)
begin
   if(i_rstb = '0') then
      address_reg <= (others => '0');
      data_reg    <= (others => '0');
      write_reg   <= '0';
   elsif(rising_edge(i_clk)) then
      if(state = Idle) then
         if(i_write = '1') then
            address_reg <= i_address;
            data_reg    <= i_data_in;
            write_reg   <= '1';
         elsif(i_read = '1') then
            address_reg <= i_address;
            data_reg    <= i_data_in;
            write_reg   <= '0';
         end if;
      end if;
   end if;
end process;
```


© tj

SRAM Controller

- FSM – Next State Logic / Register

```
next_state_logic: process(state, i_read, i_write)
begin
   case state is
      when Idle =>
         if((i_read = '1') or (i_write = '1')) then
            state_next <= Row;
         else
            state_next <= Idle;
         end if;
      when Row =>
         state_next <= Column;
      when Column =>
         state_next <= Complete;
      when Complete =>
         state_next <= Idle;
      when others =>
         state_next <= Idle;
   end case;
end process;
```

```
fsm_register: process(i_clk, i_rstb)
begin
   if (i_rstb = '0') then
      state <= Idle;
   elsif rising_edge(i_clk) then
      state <= state_next;
   end if;
end process;
```


© tj

SRAM Controller

- FSM – Output Logic (to SRAM)

```
   -- output logic to the SRAM
   sram_output_logic: process(state, address_reg, data_reg, write_reg)
   begin
      -- pass through
      o_data_to_sram    <= data_reg;

      case state is
         when Idle =>
            o_ras_n     <= '1';
            o_cas_n     <= '1';
            o_we_n      <= '1';
            o_address   <= (others => '0');
         when Row =>
            o_ras_n <= '0';
            o_cas_n <= '1';
            -- WE does not matter during row selection.
            o_we_n <= '1';
            o_address   <= address_reg(((ROW_ADDR_WIDTH + COL_ADDR_WIDTH) - 1) downto COL_ADDR_WIDTH);
--          o_address   <= address_reg((ROW_ADDR_WIDTH - 1) downto 0);
         when Column =>
            o_ras_n     <= '0';
            o_cas_n     <= '0';
            if(write_reg = '1') then
               o_we_n   <= '0';
            else
               o_we_n   <= '1';
            end if;
            o_address   <= std_logic_vector(resize(unsigned(address_reg((COL_ADDR_WIDTH - 1) downto 0)),ROW_ADDR_WIDTH));
         when Complete =>
            o_ras_n <= '1';
            o_cas_n <= '1';
            o_we_n  <= '1';
            o_address   <= (others => '0');
         when others =>
            o_ras_n <= '1';
            o_cas_n <= '1';
            o_we_n  <= '1';
            o_address   <= (others => '0');
      end case;
   end process;
```


© tj

SRAM Controller

- Output Logic (to Processor)

```
   busy_process: process(state)
   begin
      if(state = Idle) then
         o_busy <= '0';
      else
         o_busy <= '1';
      end if;
   end process;

   done_process: process(state)
   begin
      if(state = Complete) then
         o_done <= '1';
      else
         o_done <= '0';
      end if;
   end process;

end architecture;
```


© tj

SRAM Controller

- RTL

