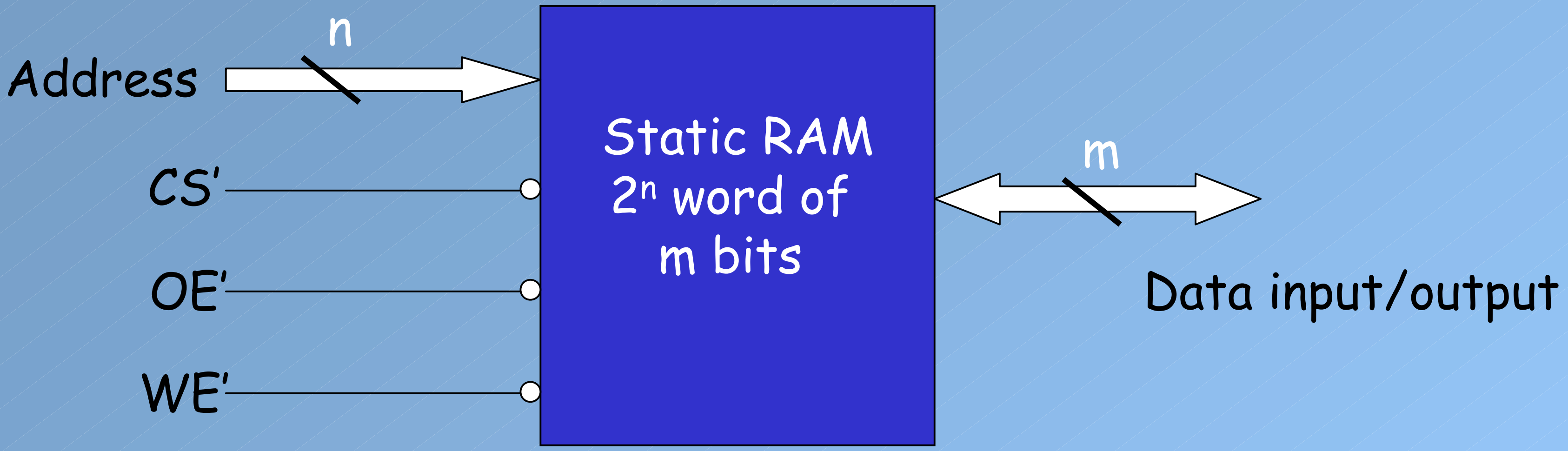
VHDL
Memory Models

EL 310
Erkay Savaş
Sabancı University

ROM

```
library ieee;
use ieee.std_logic_1164.all;
entity rom16x8 is
  port(address: in integer range 0 to 15;
       data: out std_ulogic_vector(7 downto 0));
end entity;
architecture sevenseg of rom16x8 is
  type rom_array is array (0 to 15) of std_ulogic_vector (7
       downto 0);
  constant rom: rom_array := ( “11111011”, “00010010”,
      “10011011”, “10010011”, “01011011”, “00111010”,
      “11111011”, “00010010”, “10100011”, “10011010”,
      “01111011”, “00010010”, “10101001”, “00110110”,
      “11011011”, “01010010”);
begin
  data <= rom(address);
end architecture;
```

Static RAM



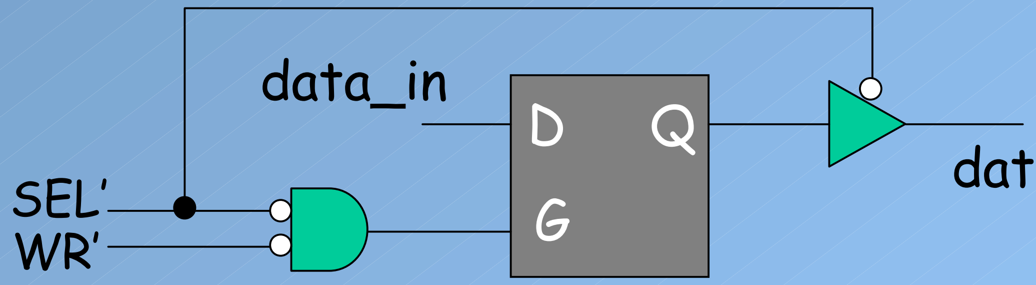
CS' - when asserted low, memory read and write operations are possible.

OE' - when asserted low, memory output is enabled onto an external bus

WE' - when asserted low, memory can be written

A Cell of Static RAM

- The RAM contains address decoders and a memory array.
- A cell of RAM that stores one bit of data



<u>Read mode</u>: SEL' = '0' and WR' = '1', (G='0') and data_out = Q

<u>Write mode</u>: SEL' = '0' and WR' = '0', (G='1') and Q = data_in

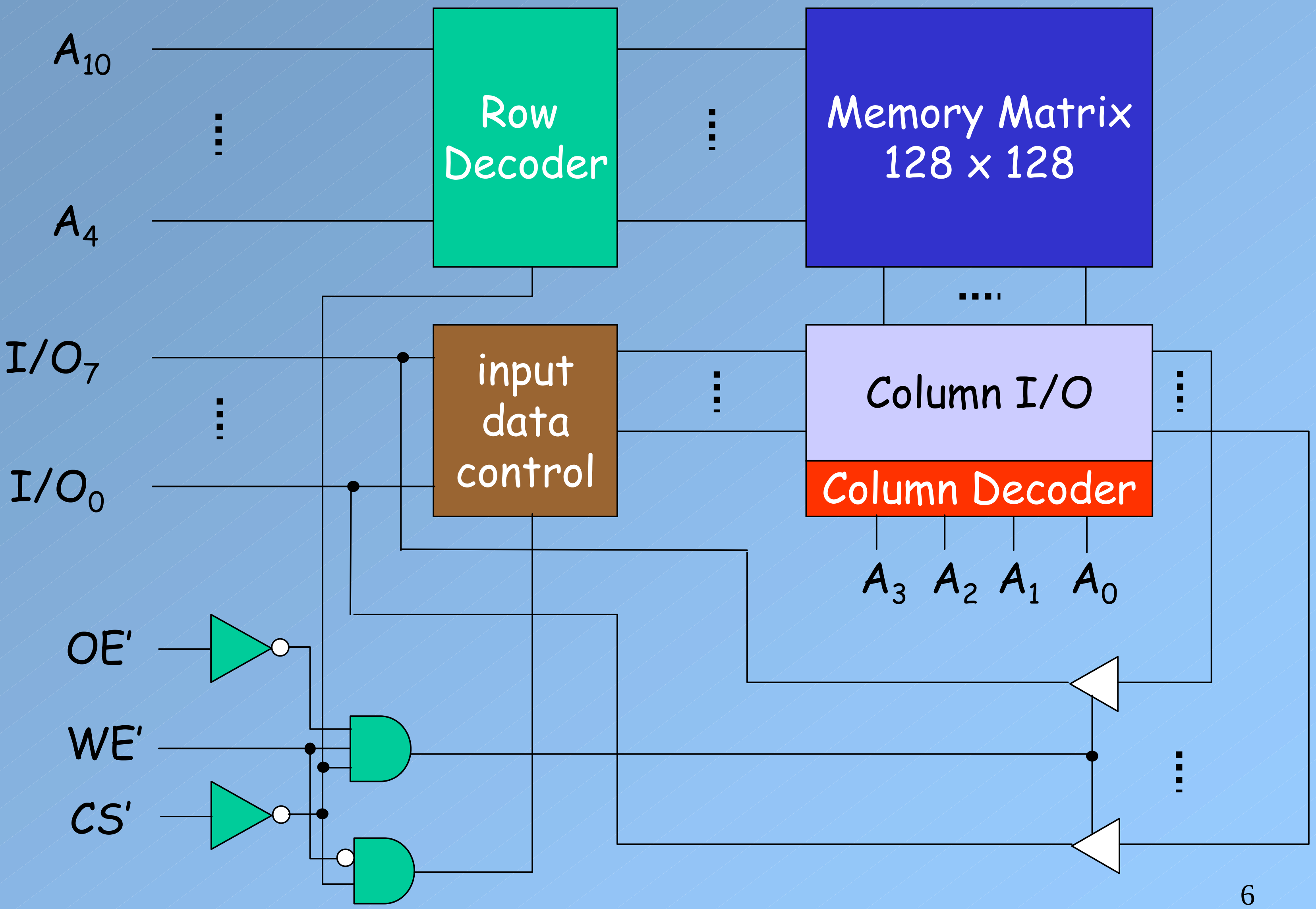
When SEL' = '1' or WR' = '1', the data is stored in the latch

When SEL' = '1', data_out = 'Z'

Truth Table of Static RAM

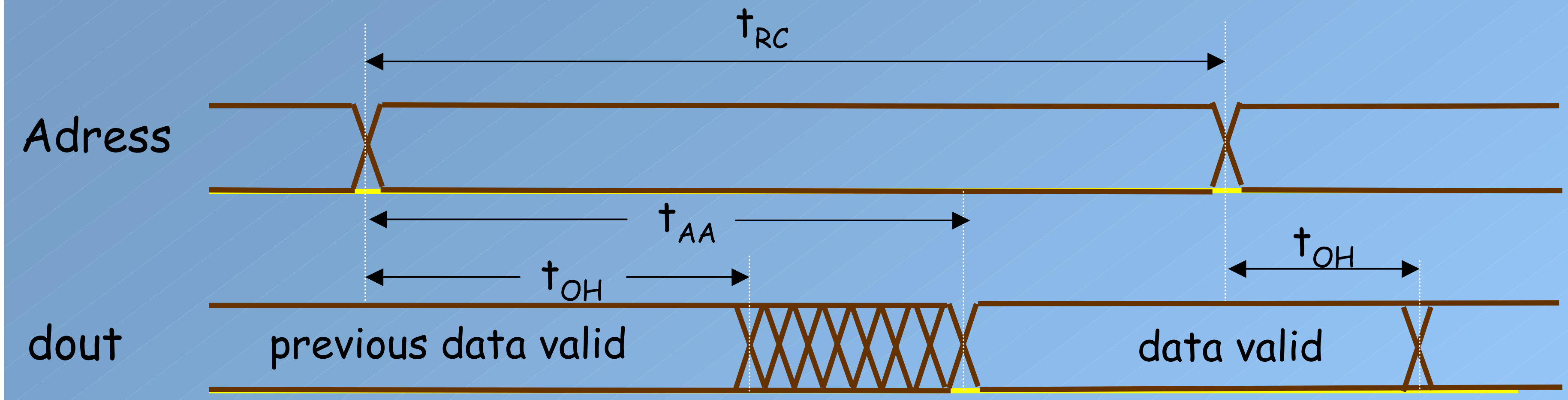
CS'	OE'	WE'	Mode	I/O pins
H	X	X	not selected	high-Z
L	H	H	output disabled	high-Z
L	L	H	read	data_out
L	X	L	write	data_in

6116 Static CMOS RAM



Read Cycle Timing I

CS' = 0, OE' = 0 WE' = 1



The address must be stable for the read cycle time, t_{RC}

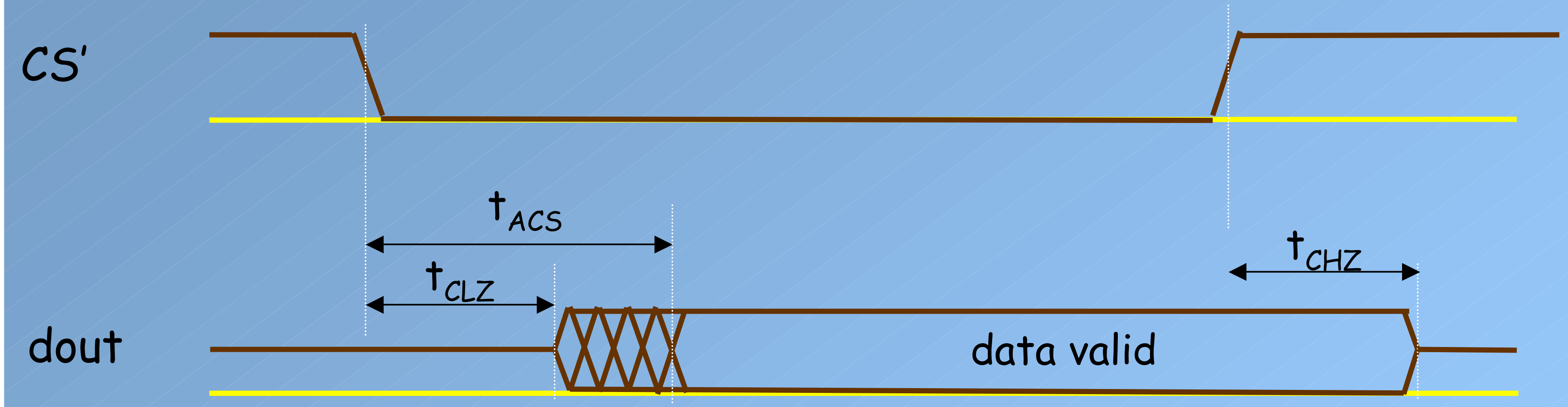
After the address changes, the old data remains at the output for a time t_{OH}

Then there is a transition period during which the data may change (cross-hatching section)

The new data is stable at the memory after the address access time t_{AA}

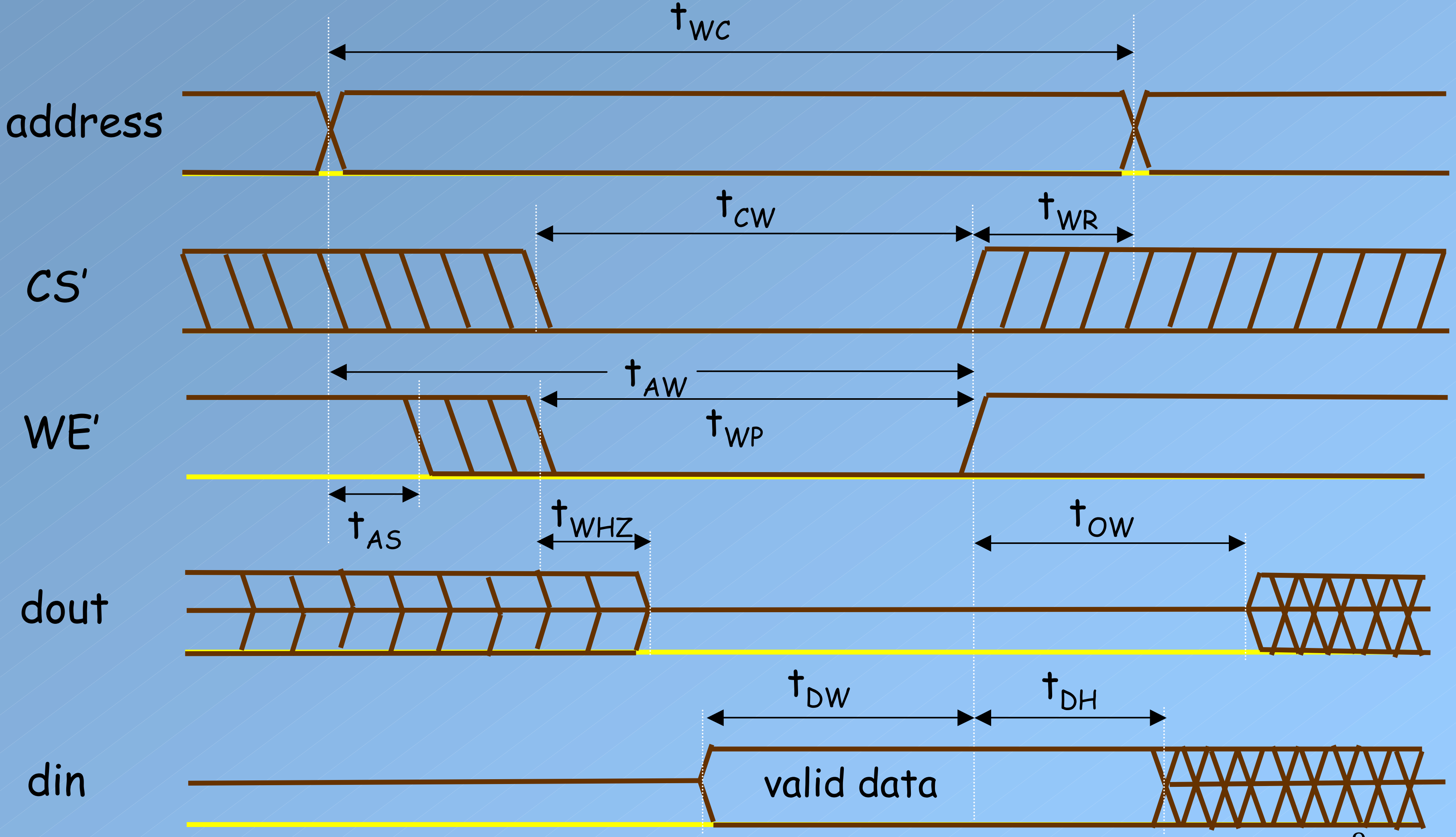
Read Cycle Timing II

Address is stable, OE' = 0, WE' = 1



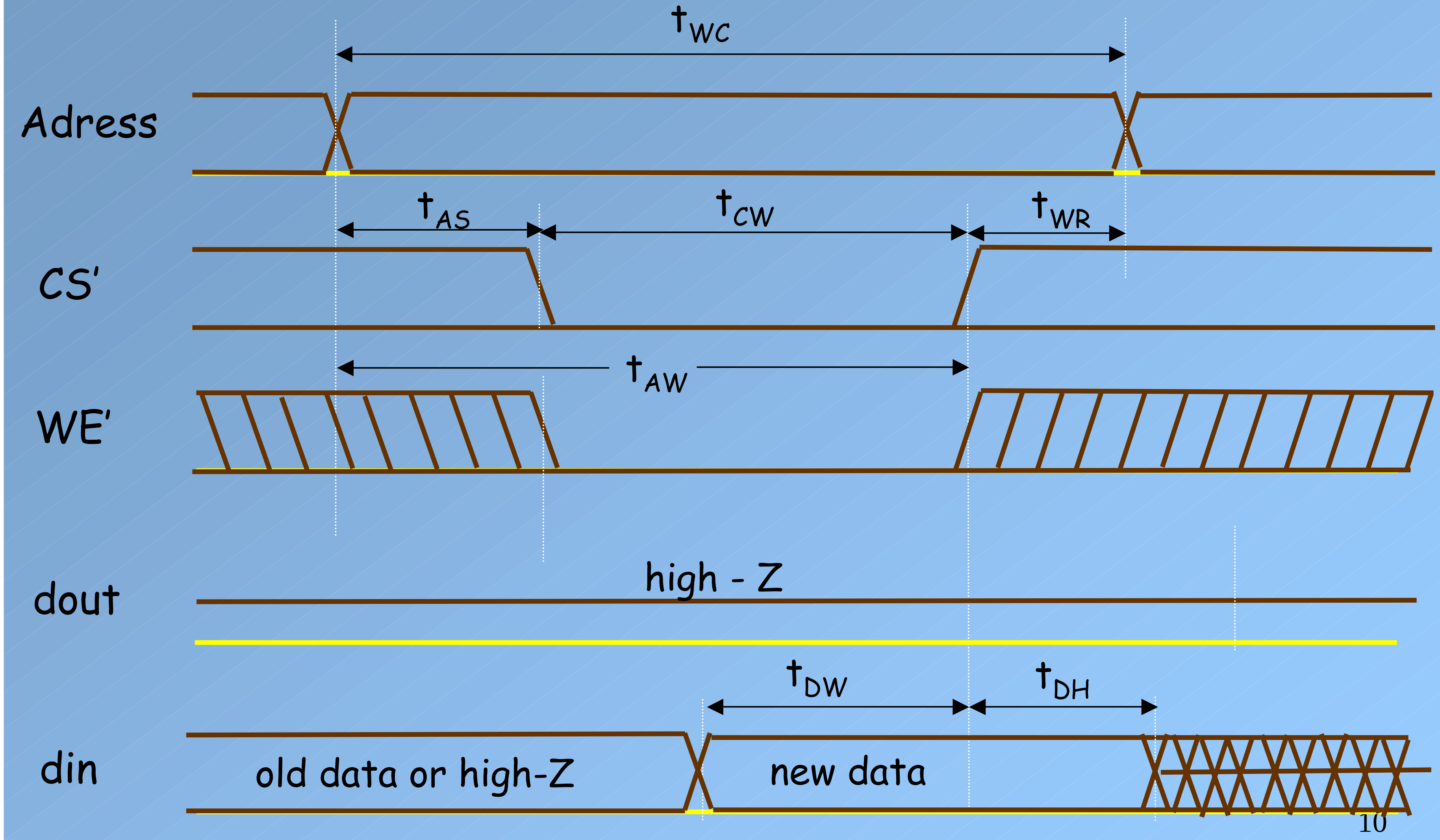
WE’ Controlled Write Cycle Time (OE’ = 0).

- CS’ goes low before or at the same time as WE’ goes low
- WE’ goes high before or at the same time as CS’ goes high



CS' controlled write cycle time (OE' = 0).

- WE' goes low before or at the same time as CS' goes low
- CS' goes high before or at the same time as WE' goes high



Writing to the Memory

- In both CS' and WE' controlled write cycles, writing to memory occurs when both CS' and WE' are low,
- writing is completed when either one of these signals goes high.

Timing Specifications for CMOS SRAM

Parameter	Symbol	61162-2		43258A-25	
		min	max	min	max
Read Cycle Time	t_{RC}	120	-	25	-
Address Access Time	t_{AA}	-	120	-	25
Chip Select Access Time	t_{ACS}	-	120	-	25
Chip selection to output in low-Z	t_{CLZ}	10	-	3	-
Output enable to output valid	t_{OE}	-	80	-	12
Output enable to output in low-Z	t_{OLZ}	10	-	0	-
Chip de-selection to output in high-Z	t_{CHZ}	10	40	3	10
Chip disable to output in high-Z	t_{OHZ}	10	40	3	10
Output hold from address change	t_{OH}	10	-	3	-

Timing Specifications for CMOS SRAM

Parameter	Symbol	61162-2		43258A-25	
		min	max	min	max
Write Cycle Time	t_{WC}	120	-	25	-
Chip selection to end of write	t_{CW}	70	-	15	-
Address valid to end of write	t_{AW}	105	-	15	-
Address setup time	t_{AS}	0	-	0	-
Write pulse width	t_{WP}	70	-	15	-
Write recovery time	t_{WR}	0	-	0	-
Write enable to output in high-Z	t_{WHZ}	10	35	3	10
Data valid to end of write	t_{DW}	35	-	12	-
Data hold from end of write	t_{DH}	0	-	0	-
Output active from end of write	t_{OW}	10	-	0	-

Simple Memory Model

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

entity ram6116 is
  port(address: in unsigned(7 downto 0);
       data: inout std_logic_vector(7 downto 0);
       WE_b, CS_b, OE_b: in std_ulogic);
end entity ram6116;
```

Simple Memory Model

```
architecture simple_ram of ram6116 is
  type ram_type is array (0 to 2**8-1) of
    std_logic_vector(7 downto 0);
  signal ram1: ram_type:= (others => (others => ’0’));
begin
  process
  begin
    data <= (others => ‘Z’);  -- chip is not selected
    if (CS_b = ‘0’) then
      if rising_edge(WE_b) then -- write
         ram1(conv_integer(address’delayed)) <= data;
         wait for 0 ns;
      end if;
      if WE_b = ‘1’ and OE_b = ‘0’ then -- read
        data <= ram1(conv_integer(address));
      else
        data <= (others => ‘Z’);
      end if;
    end if;
   wait on WE_b, CS_b, OE_b, address;
end process; end simple_ram;
```

Synthesizeable Memory Model

```
architecture simple_ram of ram6116 is
  type ram_type is array (0 to 2**8) of
       std_logic_vector(7 downto 0);
  signal ram1: ram_type;
begin
  process (address, CS_b, WE_b, OE_b) is
  begin
    data <= (others => ‘Z’);  -- chip is not selected
    if (CS_b = ‘0’) then
      if WE_b = ‘0’ then -- write
         ram1(conv_integer(address)) <= data;
      end if;
      if WE_b = ‘1’ and OE_b = ‘0’ then -- read
        data <= ram1(conv_integer(address));
      else
        data <= (others => ‘Z’);
      end if;
    end if;
end process;
end simple_ram;
```

Timing Model for SRAM I

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
entity ram6116 is
  generic (constant t_AA: Time := 120 ns;
    constant t_ACS: Time := 120 ns;
    constant t_CLZ: Time := 10 ns;
    constant t_CHZ: Time := 10 ns;
    constant t_OH: Time := 10 ns;
    constant t_WC: Time := 120 ns;
    constant t_AW: Time := 105 ns;
    constant t_WP: Time := 70 ns;
    constant t_WHZ: Time := 35 ns;
    constant t_DW: Time := 35 ns;
    constant t_DH: Time := 0 ns;
    constant t_OW: Time := 10 ns);

  port(address: in unsigned(7 downto 0);
       data: inout std_logic_vector(7 downto 0);
       WE_b, CS_b, OE_b: in std_ulogic);
end entity ram6116;
```

Timing Model for SRAM II

```
architecture SRAM of ram6116 is
  type ram_type is array(0 to 2**8) of std_logic_vector(7 downto 0);
  signal ram1: ram_type := (others => (others => ‘0’));
begin
  ram: process
  begin
    if (rising_edge(WE_b) and CS_b’delayed = ‘0’)
      or (rising_edge(CS_b) and WE_b’delayed = ‘0’) then
      -- write
      ram1(conv_integer(address’delayed)) <= data’delayed;
      -- data’delayed is the value of data just before the falling_edge
      data <= transport data’delayed after t_OW;
    end if;
    if (falling_edge(WE_b) and CS_b = ‘0’) then
      -- enter write mode
      data <= transport “ZZZZZZZZ” after t_WHZ;
    end if;
```

Timing Model for SRAM III

```
architecture SRAM of ram6116 is
  ...
begin
  ram: process
  begin
    ...
    if CS_b’event and OE_b = ‘0’ then
      if CS_b = ‘1’ then   -- RAM is de-selected
        data <= transport “ZZZZZZZZ” after t_CHZ;
      elsif WE_b = ‘1’ then –- read
        data <= “XXXXXXXX” after t_CLZ;
        data <= transport ram1((conv_integer(address)) after t_ACS;
      end if;
    end if;
    if address’event and CS_b = ‘0’ and OE_b = ‘0’ and WE_b = ‘1’
    then
      data <= “XXXXXXXX” after t_OH;
      data <= transport ram1(conv_integer(address)) after t_AA;
    end if;
    wait on CS_b, WE_b, address;
end process RAM;
...
```

Timing Model for SRAM IV

```
architecture SRAM of ram6116 is
  ...
begin
...
  check: process
  begin
    if CS_b’delayed = ‘0’ and NOW /= 0 ns then
      if address’event then
        assert (address’delayed’stable(t_WC)) -- t_RC = t_WC
          report “address cycle is too short”
          severity WARNING;
      end if;
  ...
```

Timing Model for SRAM V

```
architecture SRAM of ram6116 is
  ...
begin
...
  check: process
  begin
    if CS_b'delayed = '0' and NOW /= 0 ns then
      ...
      if rising_edge(WE_b) then
        assert (address'delayed'stable(t_AW))
        report "address not long enough to end of write"
        severity WARNING;
        assert (WE_b'delayed'stable(t_WP))
        report "write pulse is too short" severity WARNING;
        assert (data'delayed'stable(t_DW))
        report "data setup time is too short" severity WARNING;
        wait for t_DH;
        assert (data'last_event >= t_DH)
        report "data hold time is too short" severity WARNING;
      end if;
    end if;
    wait on WE_b, address, CS_b;
  end process check;
end architecture sram;
```

Dynamic RAM

```
library ieee;
use ieee.std_logic_1164.all;
entity dram1024 is
  port(address: in integer range 0 to 2**5-1;
       data: inout std_ulogic_vector(7 downto 0);
       RAS, CAS, WE: in std_ulogic);
end entity;
```

Dynamic RAM

```
architecture beh of dram1024 is
begin
  p0: process(RAS, CAS, WE) is
    type dram_array is array (0 to 2**10-1) of
         std_ulogic_vector (7 downto 0);
    variable row_address: integer range 0 to 2**5-1;
    variable mem_address: integer range 0 to 2**10-1;
    variable mem: dram_array;
  begin
...
end process p0;
end architecture;
```

Dynamic RAM

```
architecture beh of dram1024 is
begin
  p0: process(RAS, CAS, WE) is
  begin
    data <= (others => ‘Z’);
    if falling_edge(RAS) then row_address := address;
    elsif falling_edge(CAS) then
      mem_address := row_address*2**5 + address;
      if RAS = ‘0’ and WE = ‘0’ then
      mem(mem_address) := data;
      end if;
    if CAS = ‘0’ and RAS = ‘0’ and WE = ‘1’ then
      data <= mem(mem_address);
    end if;
  end process p0;
end architecture;
```