


ECE 274 Digital Logic

RTL Design – Memories and Hierarchy

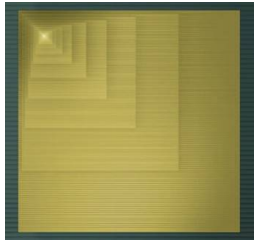
Digital Design 5.6 – 5.8



THE UNIVERSITY OF
ARIZONA
TUCSON ARIZONA

Digital Design

Chapter 5: RTL Design



Slides to accompany the textbook *Digital Design*, First Edition,
by Frank Vahid, John Wiley and Sons Publishers, 2007.
<http://www.ddvahid.com>

Copyright © 2007 Frank Vahid

Instructors of courses requiring Vahid's *Digital Design* textbook (published by John Wiley and Sons) have permission to modify and use these slides for customary course-related activities, subject to keeping this copyright notice in place and unmodified. These slides may be posted as ~~unmodified~~ pdf versions on publicly-accessible course websites. PowerPoint source (or pdf with animations) may ~~not~~ be posted to publicly-accessible websites, but may be posted for students on internal protected sites or distributed directly to students by other electronic means. Instructors may make printouts of the slides available to students for a reasonable photocopying charge, without incurring royalties. Any other use requires explicit permission. Instructors may obtain PowerPoint source or obtain special use permissions from Wiley – see <http://www.ddvahid.com> for information.

RTL Design

RTL Design Method

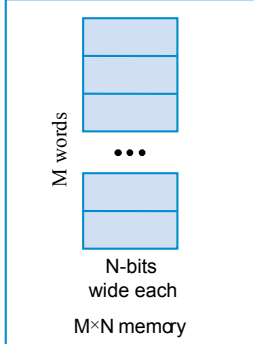
Step	Description
Step 1 <i>Capture a high-level state machine</i>	Describe the system's desired behavior as a high-level state machine. The state machine consists of states and transitions. The state machine is "high-level" because the transition conditions and the state actions are more than just Boolean operations on bit inputs and outputs.
Step 2 <i>Create a datapath</i>	Create a datapath to carry out the data operations of the high-level state machine.
Step 3 <i>Connect the datapath to a controller</i>	Connect the datapath to a controller block. Connect external Boolean inputs and outputs to the controller block.
Step 4 <i>Derive the controller's FSM</i>	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.

3

RTL Design

Memory Components

- Register-transfer level design instantiates datapath components to create datapath, controlled by a controller
 - A few more components are often used outside the controller and datapath
- *MxN memory*
 - M words, N bits wide each
- Several varieties of memory, which we now introduce

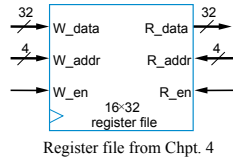


4

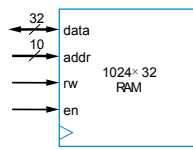
RTL Design

Random Access Memory (RAM)

- RAM – Readable and writable memory
 - “Random access memory”
 - Strange name – Created several decades ago to contrast with sequentially-accessed storage like tape drives
 - Logically same as register file – Memory with address inputs, data inputs/outputs, and control
 - RAM usually just one port; register file usually two or more
 - RAM vs. register file
 - RAM typically larger than *roughly* 512 or 1024 words
 - RAM typically stores bits using a bit storage approach that is more efficient than a flip flop
 - RAM typically implemented on a chip in a square rather than rectangular shape – keeps longest wires (hence delay) short



Register file from Chpt. 4

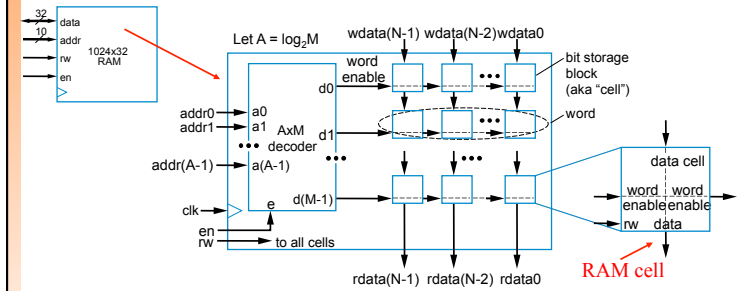


RAM block symbol

5

RTL Design

RAM Internal Structure

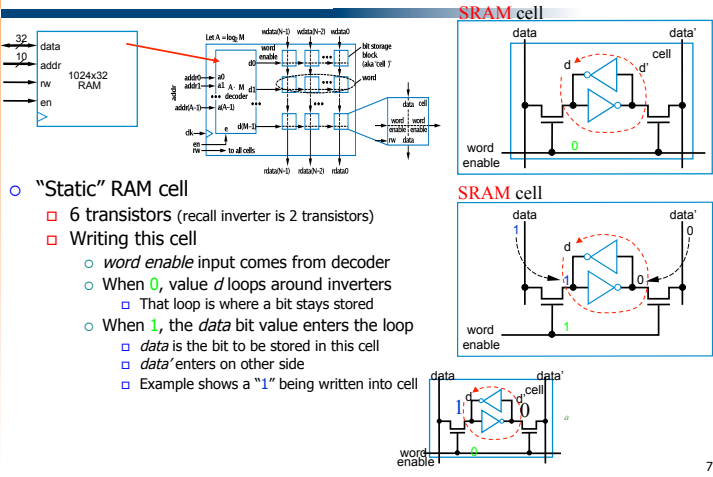


- Similar internal structure as register file
 - Decoder enables appropriate word based on address inputs
 - rw controls whether cell is written or read
 - Let's see what's inside each RAM cell

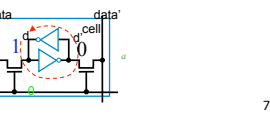
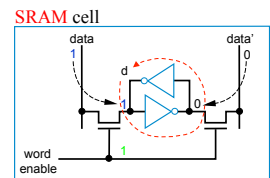
6

RTL Design

Static RAM (SRAM)



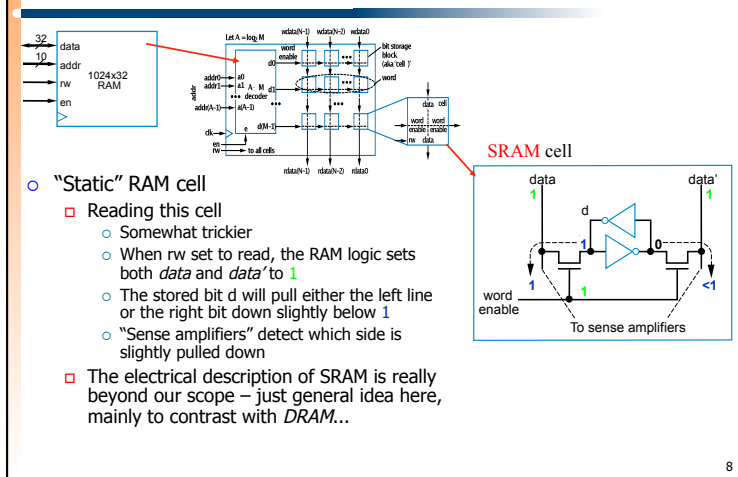
- “Static” RAM cell
 - 6 transistors (recall inverter is 2 transistors)
 - Writing this cell
 - word enable input comes from decoder
 - When 0, value *d* loops around inverters
 - That loop is where a bit stays stored
 - When 1, the *data* bit value enters the loop
 - data* is the bit to be stored in this cell
 - data'* enters on other side



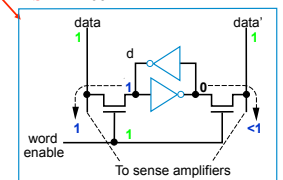
7

RTL Design

Static RAM (SRAM)



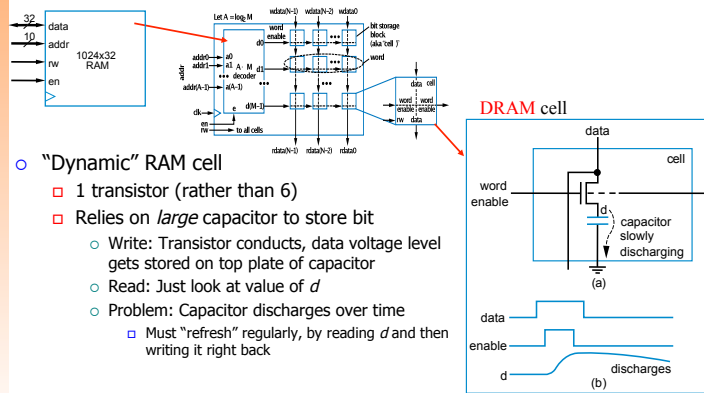
- “Static” RAM cell
 - Reading this cell
 - Somewhat trickier
 - When rw set to read, the RAM logic sets both *data* and *data'* to 1
 - The stored bit *d* will pull either the left line or the right bit down slightly below 1
 - “Sense amplifiers” detect which side is slightly pulled down
 - The electrical description of SRAM is really beyond our scope – just general idea here, mainly to contrast with *DRAM*...



8

RTL Design

Dynamic RAM (DRAM)



“Dynamic” RAM cell

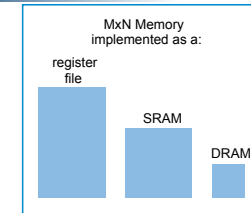
- 1 transistor (rather than 6)
- Relies on *large* capacitor to store bit
 - Write: Transistor conducts, data voltage level gets stored on top plate of capacitor
 - Read: Just look at value of *d*
 - Problem: Capacitor discharges over time
 - Must “refresh” regularly, by reading *d* and then writing it right back

9

RTL Design

Comparing Memory Types

- Register file
 - Fastest
 - But biggest size
- SRAM
 - Fast
 - More compact than register file
- DRAM
 - Slowest
 - And refreshing takes time
 - But very compact
- Use register file for small items, SRAM for large items, and DRAM for huge items
 - Note: DRAM’s big capacitor requires a special chip design process, so DRAM is often a separate chip

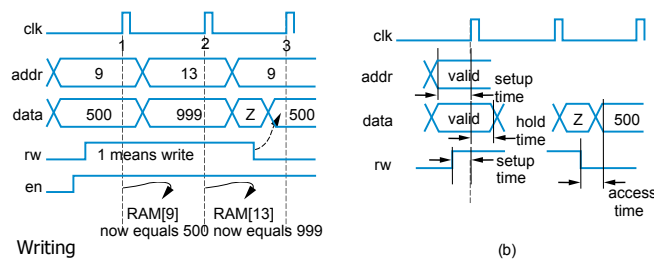


Size comparison for same number of bits (not to scale)

10

RTL Design

Reading and Writing a RAM

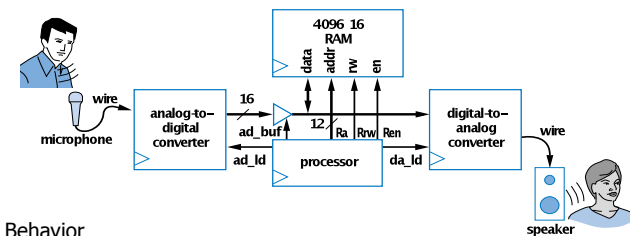


- Writing
 - Put address on *addr* lines, data on *data* lines, set *rw=1*, *en=1*
- Reading
 - Set *addr* and *en* lines, but put nothing (Z) on *data* lines, set *rw=0*
 - Data will appear on *data* lines
- Don’t forget to obey setup and hold times
 - In short – keep inputs stable before and after a clock edge

11

RTL Design

RAM Example: Digital Sound Recorder



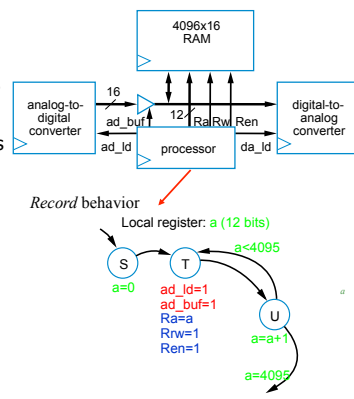
- Behavior
 - Record: Digitize sound, store as series of 4096 12-bit digital values in RAM
 - We’ll use a 4096x16 RAM (12-bit wide RAM not common)
 - Play back later
 - Common behavior in telephone answering machine, toys, voice recorders
- To record, processor should read a-to-d, store read values into successive RAM words
 - To play, processor should read successive RAM words and enable d-to-a

12

RTL Design

RAM Example: Digital Sound Recorder

- RTL design of processor
 - Create high-level state machine
 - Begin with the *record* behavior
 - Keep local register *a*
 - Stores current address, ranges from 0 to 4095 (thus need 12 bits)
 - Create state machine that counts from 0 to 4095 using *a*
 - For each *a*
 - Read analog-to-digital conv.
 - $ad_ld=1, ad_buf=1$
 - Write to RAM at address *a*
 - $Ra=a, Rrw=1, Ren=1$

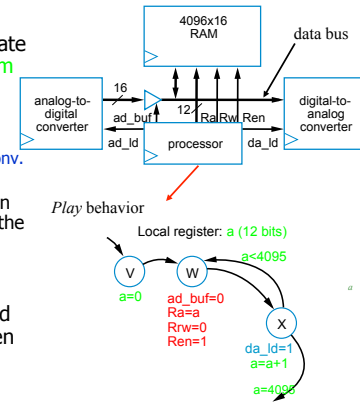


13

RTL Design

RAM Example: Digital Sound Recorder

- Now create *play* behavior
- Use local register *a* again, create state machine that counts from 0 to 4095 again
 - For each *a*
 - Read RAM
 - $Ra=a, Rrw=0, Ren=1$
 - Write to digital-to-analog conv.
 - $da_ld=1, a=a+1$
 - Note: Must write d-to-a one cycle *after* reading RAM, when the read data is available on the data bus
- The record and play state machines would be parts of a larger state machine controlled by signals that determine when to record or play

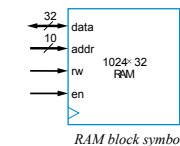


14

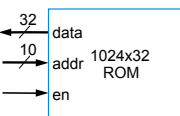
RTL Design

Read-Only Memory – ROM

- Memory that can only be read from, not written to
 - Data lines are output only
 - No need for *rw* input
- Advantages over RAM
 - Compact: May be smaller
 - **Nonvolatile**: Saves bits even if power supply is turned off
 - Speed: May be faster (especially than DRAM)
 - Low power: Doesn't need power supply to save bits, so can extend battery life
- Choose ROM over RAM if stored data won't change (or won't change often)
 - For example, a table of Celsius to Fahrenheit conversions in a digital thermometer



RAM block symbol

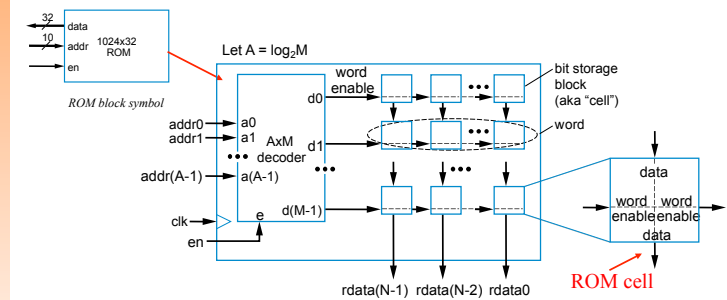


ROM block symbol

15

RTL Design

Read-Only Memory – ROM



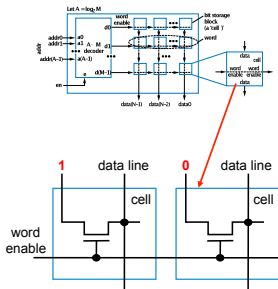
- Internal logical structure similar to RAM, without the data input lines

16

RTL Design

ROM Types

- If a ROM can only be read, how are the stored bits stored in the first place?
 - Storing bits in a ROM known as *programming*
 - Several methods
- **Mask-programmed ROM**
 - Bits are hardwired as 0s or 1s during chip manufacturing
 - 2-bit word on right stores "10"
 - word enable (from decoder) simply passes the hardwired value through transistor
 - Notice how compact, and fast, this memory would be

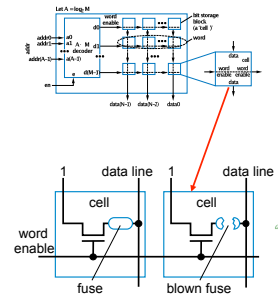


17

RTL Design

ROM Types

- **Fuse-Based Programmable ROM**
 - Each cell has a fuse
 - A special device, known as a programmer, blows certain fuses (using higher-than-normal voltage)
 - Those cells will be read as 0s (involving some special electronics)
 - Cells with unblown fuses will be read as 1s
 - 2-bit word on right stores "10"
 - Also known as **One-Time Programmable (OTP) ROM**

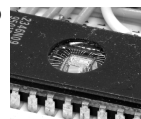
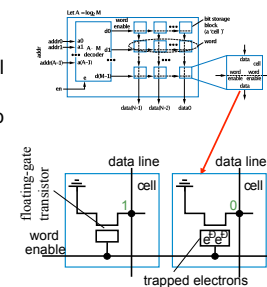


18

RTL Design

ROM Types

- **Erasable Programmable ROM (EPROM)**
 - Uses "floating-gate transistor" in each cell
 - Special programmer device uses higher-than-normal voltage to cause electrons to *tunnel* into the gate
 - Electrons become trapped in the gate
 - Only done for cells that should store 0
 - Other cells (without electrons trapped in gate) will be 1
 - 2-bit word on right stores "10"
 - Details beyond our scope – just general idea is necessary here
 - To erase, shine ultraviolet light onto chip
 - Gives trapped electrons energy to escape
 - Requires chip package to have window

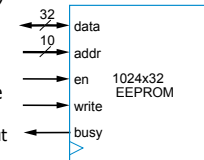


19

RTL Design

ROM Types

- **Electrically-Erasable Programmable ROM (EEPROM)**
 - Similar to EPROM
 - Uses floating-gate transistor, electronic programming to trap electrons in certain cells
 - But erasing done *electronically*, not using UV light
 - Erasing done one word at a time
- **Flash memory**
 - Like EEPROM, but all words (or large blocks of words) can be erased *simultaneously*
 - Become common relatively recently (late 1990s)
- Both types are in-system programmable
 - Can be programmed with new stored bits while in the system in which the ROM operates
 - Requires bi-directional data lines, and write control input
 - Also need busy output to indicate that erasing is in progress – erasing takes some time

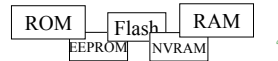


20

RTL Design

Blurring of Distinction Between ROM and RAM

- We said that
 - RAM is readable and writable
 - ROM is read-only
- But some ROMs act almost like RAMs
 - EEPROM and Flash are in-system programmable
 - Essentially means that writes are slow
 - Also, number of writes may be limited (perhaps a few million times)
- And, some RAMs act almost like ROMs
 - Non-volatile RAMs: Can save their data without the power supply
 - One type: Built-in battery, may work for up to 10 years
 - Another type: Includes ROM backup for RAM – controller writes RAM contents to ROM before turning off
- New memory technologies evolving that merge RAM and ROM benefits
 - e.g., MRAM
- Bottom line
 - Lot of choices available to designer, must find best fit with design goals

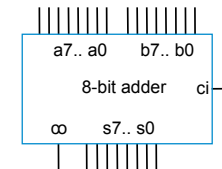


21

RTL Design

Hierarchy and Abstraction

- Hierarchy helps us **manage complexity**
 - To go from transistors to gates, muxes, decoders, registers, ALUs, controllers, datapaths, memories, queues, etc.
 - Imagine trying to comprehend a controller and datapath at the level of gates
- Abstraction
 - Hierarchy often involves not just grouping items into a new item, but also associating higher-level behavior with the new item, known as **abstraction**
 - e.g., an 8-bit adder has an understandable high-level behavior – it adds two 8-bit binary numbers
 - Frees designer from having to remember, or even from having to understand, the lower-level details

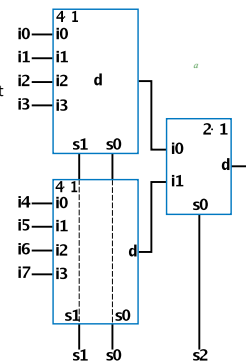
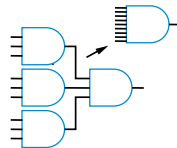


22

RTL Design

Hierarchy and Composing Larger Components from Smaller Versions

- A common task is to compose smaller components into a larger one
 - Gates: Suppose you have plenty of 3-input AND gates, but need a 9-input AND gate
 - Can simple compose the 9-input gate from several 3-input gates
 - Muxes: Suppose you have 4x1 and 2x1 muxes, but need an 8x1 mux
 - s2 selects either top or bottom 4x1
 - s1s0 select particular 4x1 input
 - Implements 8x1 mux – 8 data inputs, 3 selects, one output

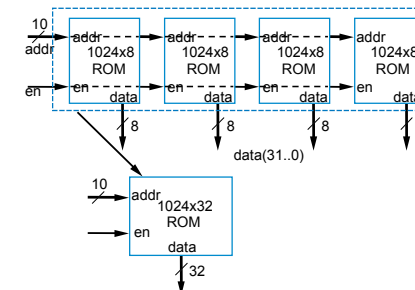


23

RTL Design

Hierarchy and Composing Larger Components from Smaller Versions

- Composing memory very common
- Making memory words wider
 - Easy – just place memories side-by-side until desired width obtained
 - Share address/control lines, concatenate data lines
 - Example: Compose 1024x8 ROMs into 1024x32 ROM



24

