Unit-III
Standard Single-Purpose Processors

By

M. Sambasiva Reddy

Asst. Professor, ECE

Contents

- Introduction
- Timers
- Counters
- Watchdog Timers
- UART
- LCD Controllers
- Stepper Motor Controllers
- Analog-to-Digital Converters
- Real-Time Clocks
- Common memory types
- Memory hierarchy and Cache
- Advanced RAM

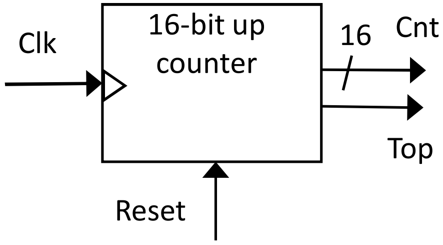
Introduction

- Single-purpose processors
  - – Performs specific computation task
  - – Custom single-purpose processors
    - Designed by us for a unique task
  - – *Standard* single-purpose processors
    - "Off-the-shelf" -- pre-designed for a common task
    - a.k.a., peripherals
    - serial transmission
    - analog/digital conversions

Timers, counters, watchdog timers

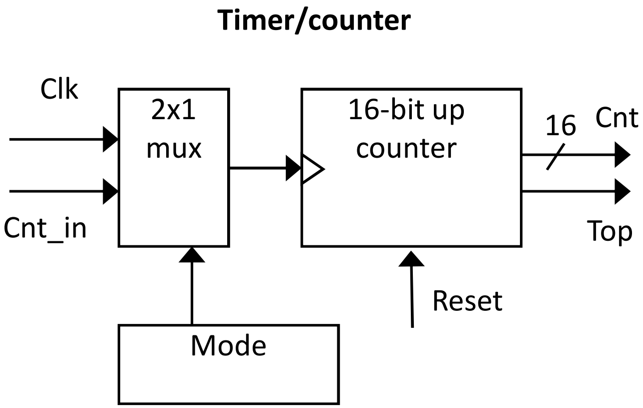
- Timer: measures time intervals
  - To generate timed output events
    - e.g., hold traffic light green for 10 s
  - To measure input events
    - e.g., measure a car’s speed
- Based on counting clock pulses
    - E.g., let Clk period be 10 ns
    - And we count 20,000 Clk pulses
    - Then 200 microseconds have passed
    - 16-bit counter would count up to 65,535*10 ns = 655.35 microsec., resolution = 10 ns
    - Top: indicates top count reached, wrap-around

Basic timer



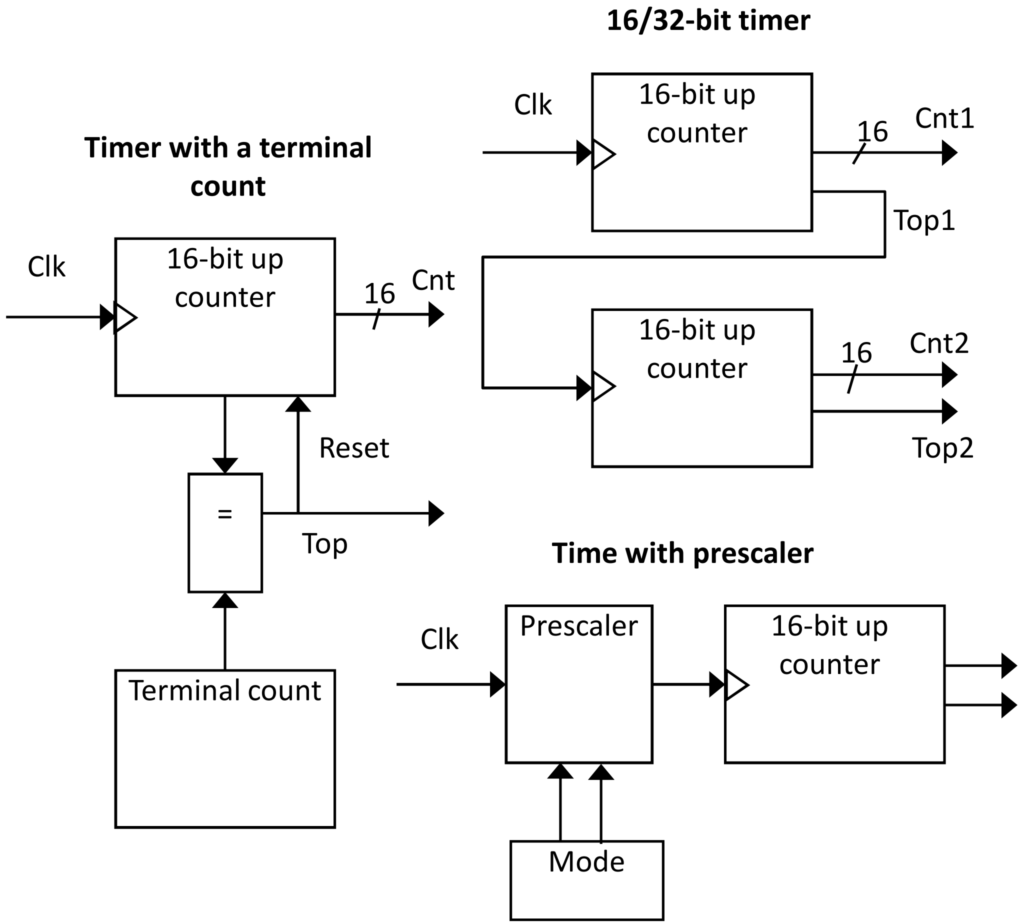
Counters

- Counter: like a timer, but counts pulses on a general input signal rather than clock
  - e.g., count cars passing over a sensor
  - Can often configure device as either a timer or counter



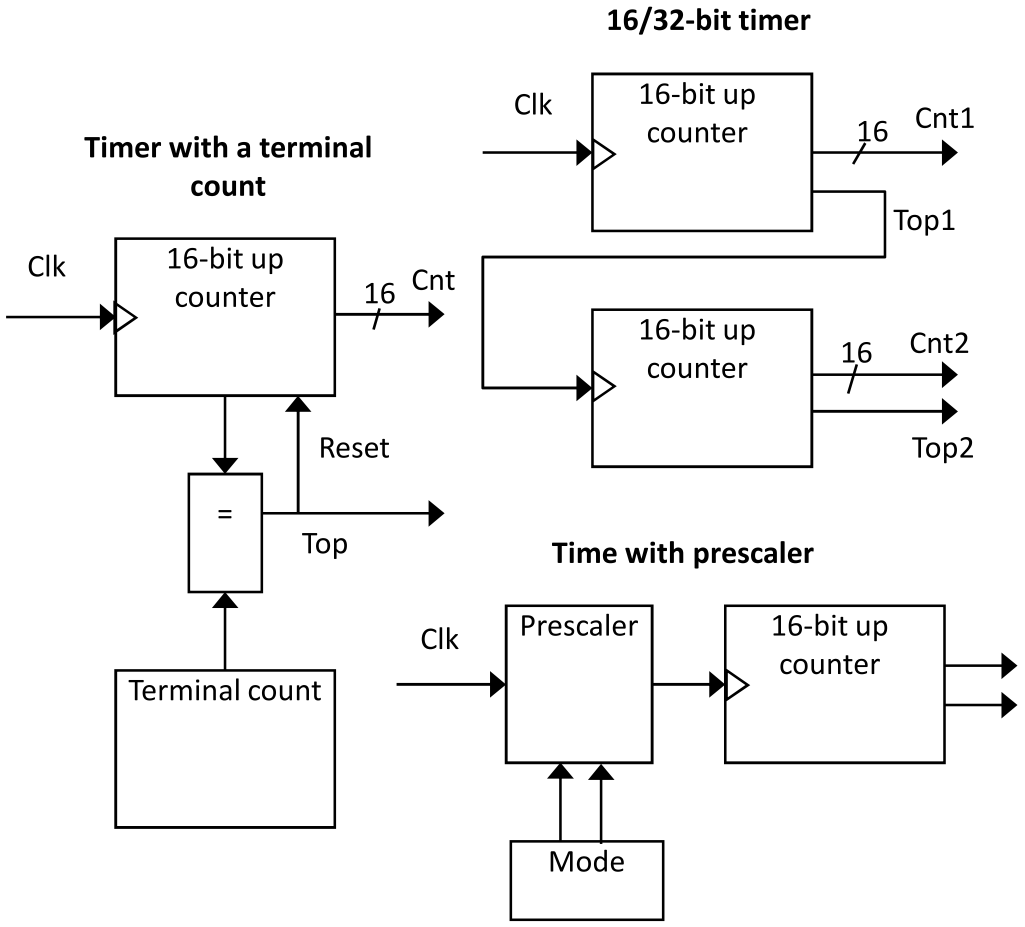
Other timer structures

- Interval timer
  - Indicates when desired time interval has passed
  - We set terminal count to desired interval
    - *Number of clock cycles = Desired time interval / Clock period*
- Cascaded counters
- Prescaler
  - Divides clock
  - Increases range, decreases resolution



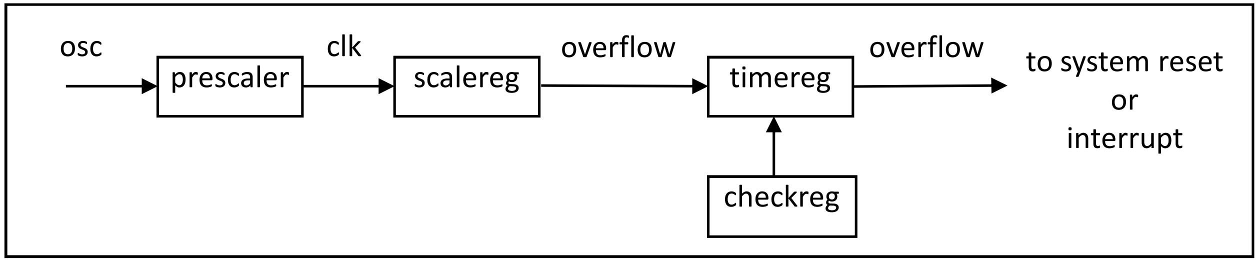
Other timer structures

- Interval timer
  - Indicates when desired time interval has passed
  - We set terminal count to desired interval
    - *Number of clock cycles = Desired time interval / Clock period*
- Cascaded counters
- Prescaler
  - Divides clock
  - Increases range, decreases resolution



Watchdog timer

- Must reset timer every X time unit, else timer generates a signal
- Common use: detect failure, self-reset
- Another use: timeouts
  - e.g., ATM machine
  - 16-bit timer, 2 microsec. resolution
  - *timereg* value = $2*(2^{16}-1)–X = 131070–X$
  - For 2 min., X = 120,000 microsec.

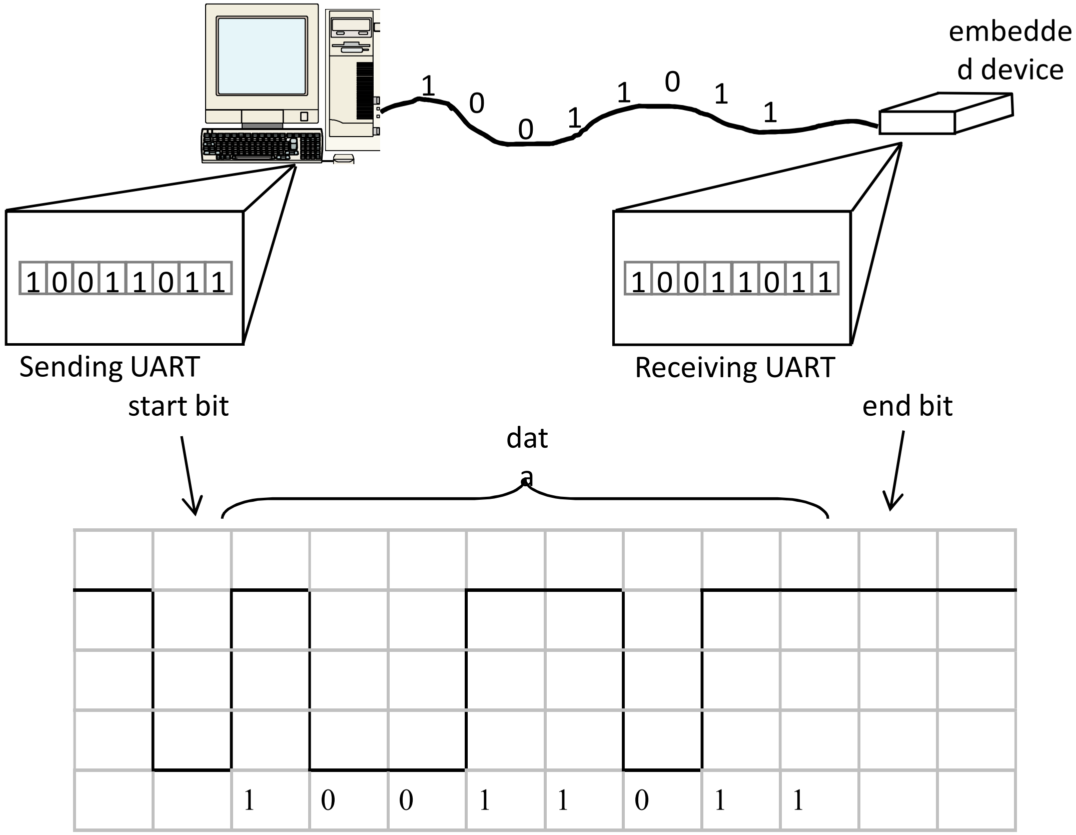


```
/* main.c */

main(){
  wait until card inserted
  call watchdog_reset_routine

  while(transaction in progress){
    if(button pressed){
      perform corresponding action
      call watchdog_reset_routine
    }

/* if watchdog_reset_routine not called
every < 2 minutes, interrupt_service_routine
is called */
}
```

```
watchdog_reset_routine(){
/* checkreg is set so we can load value into
timereg.  Zero is loaded into scalereg and
11070 is loaded into timereg */

  checkreg = 1
  scalereg = 0
  timereg  = 11070
}

void interrupt_service_routine(){
  eject card
  reset screen
}
```

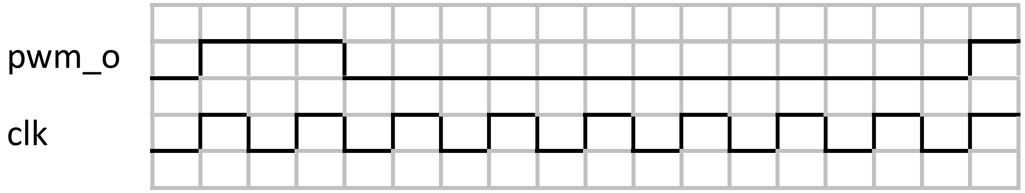
Serial Transmission Using UARTs

- UART: Universal Asynchronous Receiver Transmitter
  - Takes parallel data and transmits serially
  - Receives serial data and converts to parallel
- Parity: extra bit for simple error checking
- Start bit, stop bit
- Baud rate
  - signal changes per second
  - bit rate usually higher

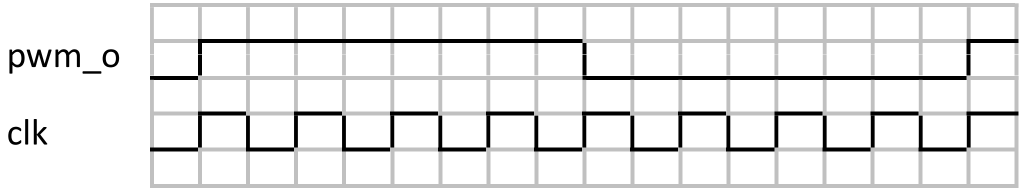


Pulse width modulator

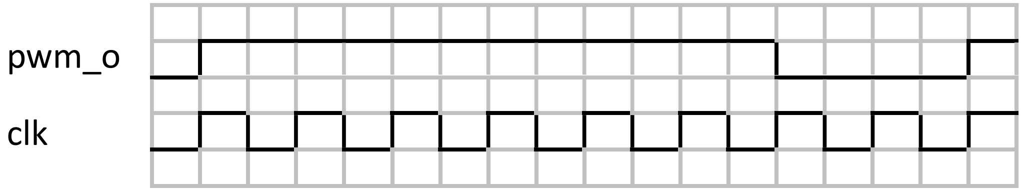
- Generates pulses with specific high/low times
- Duty cycle: % time high
  - Square wave: 50% duty cycle
- Common use: control average voltage to electric device
  - Simpler than DC-DC converter or digital-analog converter
  - DC motor speed, dimmer lights
- Another use: encode commands, receiver uses timer to decode



25% duty cycle – average pwm_o is 1.25V

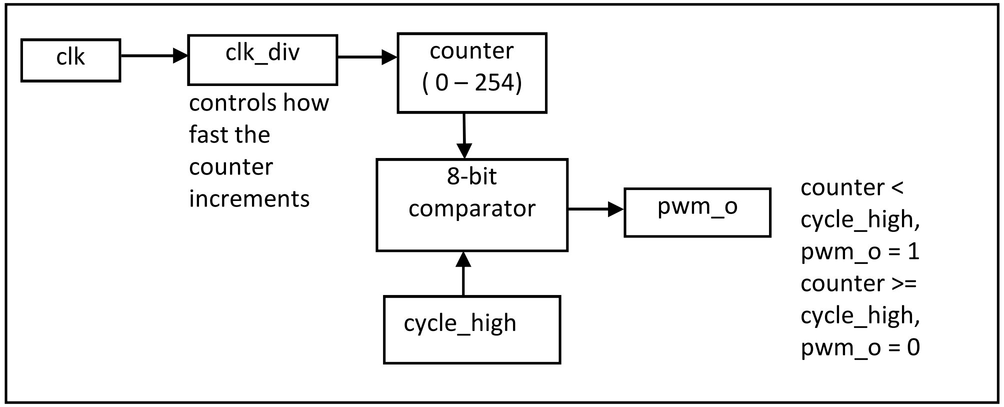


50% duty cycle – average pwm_o is 2.5V.



75% duty cycle – average pwm_o is 3.75V.

Controlling a DC motor with a PWM



clk

clk_div

controls how fast the counter increments

counter (0 – 254)

8-bit comparator

pwm_o

cycle_high

counter < cycle_high, pwm_o = 1 counter >= cycle_high, pwm_o = 0

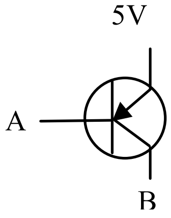
Internal Structure of PWM

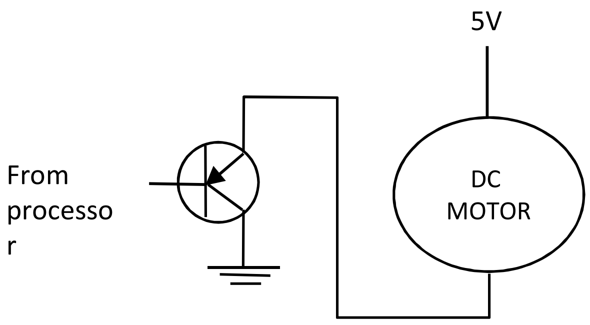
Input Voltage	% of Maximum Voltage Applied	RPM of DC Motor
0	0	0
2.5	50	1840
3.75	75	6900
5.0	100	9200

Relationship between applied voltage and speed of the DC Motor

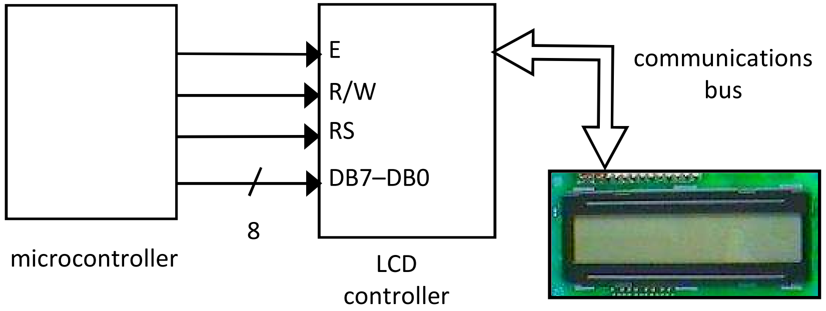
```
void main(void){

   /* controls period */
   PWMP = 0xff;
   /* controls duty cycle */
   PWM1 = 0x7f;

   while(1){};
 }
```

The PWM alone cannot drive the DC motor, a possible way to implement a driver is shown below using an MJE3055T NPN transistor.





LCD controller

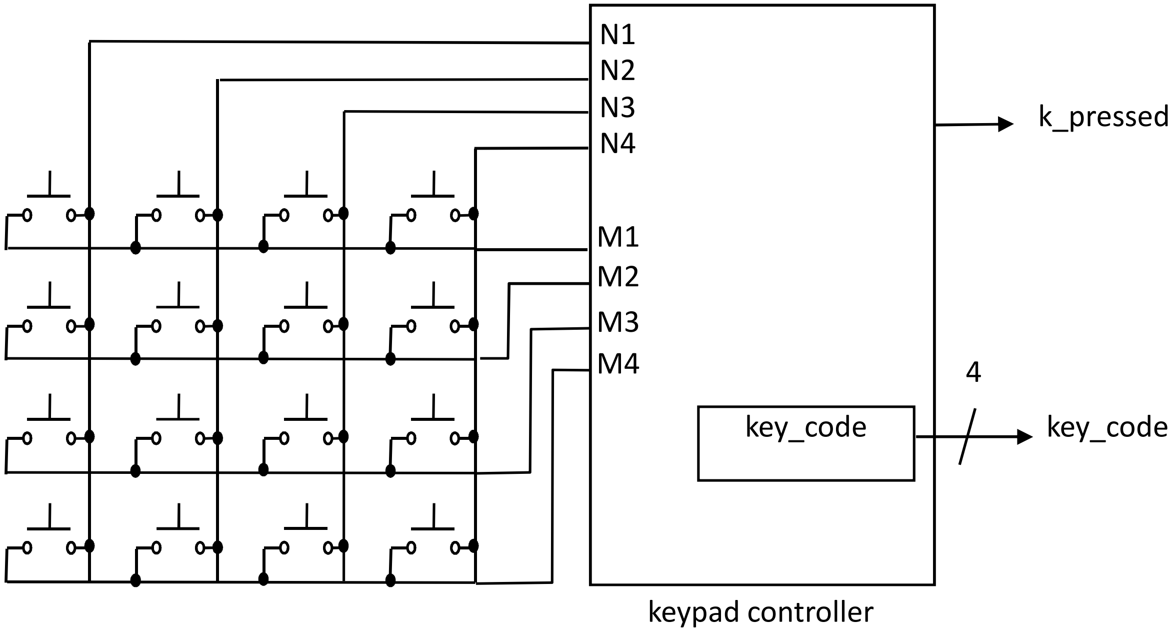


```
void WriteChar(char c){

  RS = 1;                    /* indicate data being sent */
  DATA_BUS = c;             /* send data to LCD */
  EnableLCD(45);             /* toggle the LCD with appropriate delay */
}
```

CODES	
I/D = 1 cursor moves left	DL = 1 8-bit
I/D = 0 cursor moves right	DL = 0 4-bit
S = 1 with display shift	N = 1 2 rows
S/C =1 display shift	N = 0 1 row
S/C = 0 cursor movement	F = 1 5x10 dots
R/L = 1 shift to right	F = 0 5x7 dots
R/L = 0 shift to left	

RS	R/W	DB_7	DB_6	DB_5	DB_4	DB_3	DB_2	DB_1	DB_0	Description
0	0	0	0	0	0	0	0	0	1	Clears all display, return cursor home
0	0	0	0	0	0	0	0	1	*	Returns cursor home
0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and/or specifies not to shift display
0	0	0	0	0	0	1	D	C	B	ON/OFF of all display(D), cursor ON/OFF (C), and blink position (B)
0	0	0	0	0	1	S/C	R/L	*	*	Move cursor and shifts display
0	0	0	0	1	DL	N	F	*	*	Sets interface data length, number of display lines, and character font
1	0	WRITE DATA								Writes Data

Keypad controller

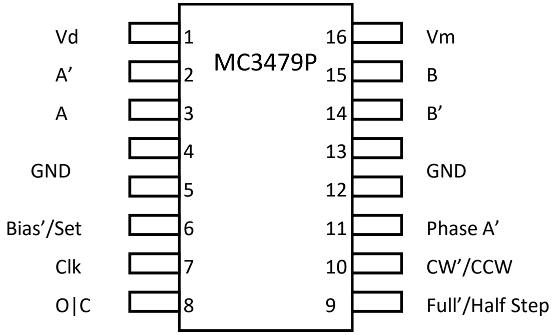


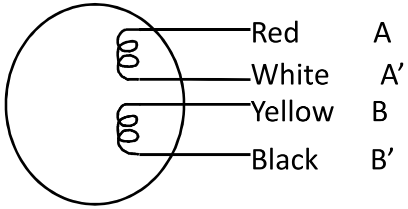
N=4, M=4

Stepper motor controller

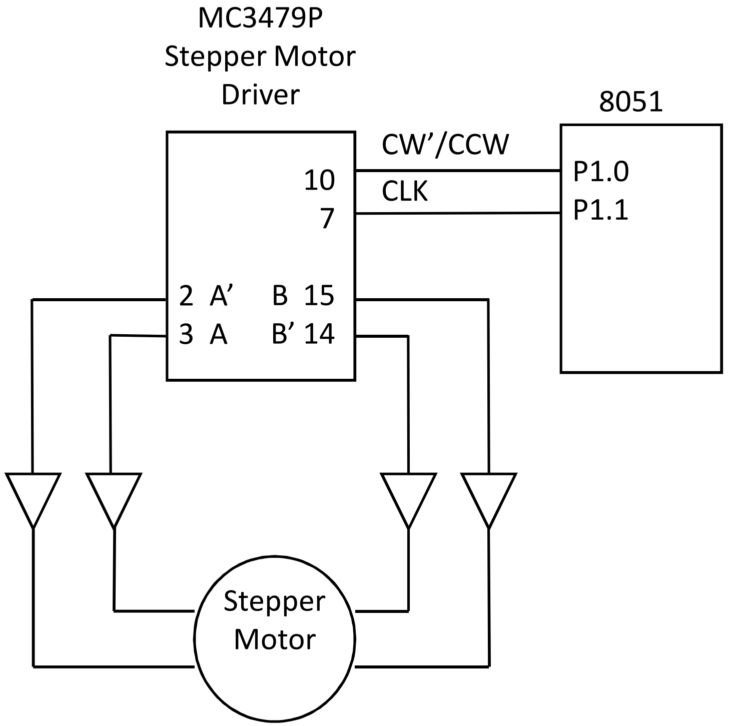
- Stepper motor: rotates fixed number of degrees when given a “step” signal
  - In contrast, DC motor just rotates when power applied, coasts to stop
- Rotation achieved by applying specific voltage sequence to coils
- Controller greatly simplifies this

Sequence	A	B	A’	B’
1	+	+	-	-
2	-	+	+	-
3	-	-	+	+
4	+	-	-	+
5	+	+	-	-





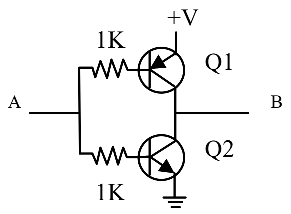
Stepper motor with controller (driver)



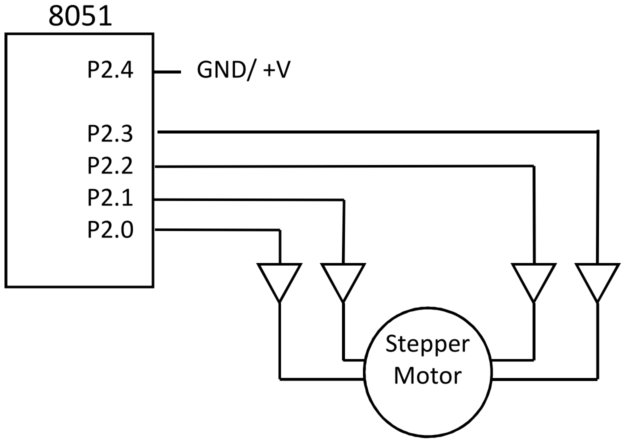
```
/* main.c */

sbit clk=P1^1;
sbit cw=P1^0;

void delay(void){
  int i, j;
  for (i=0; i<1000; i++)
    for ( j=0; j<50; j++)
      i = i + 0;
}
```

```
void main(void){

  */turn the motor forward */
  cw=0;          /* set direction */
  clk=0;         /* pulse clock */
  delay();
  clk=1;

  /*turn the motor backwards */
  cw=1;          /* set direction */
  clk=0;         /* pulse clock */
  delay();
  clk=1;

}
```

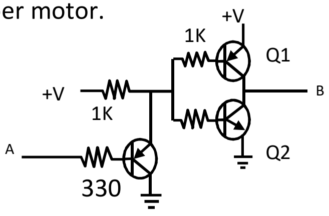
The output pins on the stepper motor driver do not provide enough current to drive the stepper motor. To amplify the current, a buffer is needed. One possible implementation of the buffers is pictured to the left. Q1 is an MJE3055T NPN transistor and Q2 is an MJE2955T PNP transistor. A is connected to the 8051 microcontroller and B is connected to the stepper motor.



Stepper motor without controller (driver)



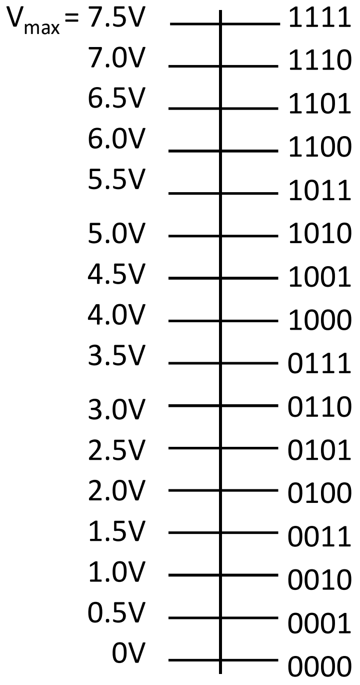
A possible way to implement the buffers is located below. The 8051 alone cannot drive the stepper motor, so several transistors were added to increase the current going to the stepper motor. Q1 are MJE3055T NPN transistors and Q3 is an MJE2955T PNP transistor. A is connected to the 8051 microcontroller and B is connected to the stepper motor.



```
/*main.c*/
sbit notA=P2^0;
sbit isA=P2^1;
sbit notB=P2^2;
sbit isB=P2^3;
sbit dir=P2^4;

void delay(){
  int a, b;
  for(a=0; a<5000; a++)
    for(b=0; b<10000; b++)
      a=a+0;
}

void move(int dir, int steps) {
int y, z;
  /* clockwise movement */
  if(dir == 1){
    for(y=0; y<=steps; y++){
      for(z=0; z<=19; z+4){
        isA=lookup[z];
        isB=lookup[z+1];
        notA=lookup[z+2];
        notB=lookup[z+3];
        delay();
      }
    }
  }
```

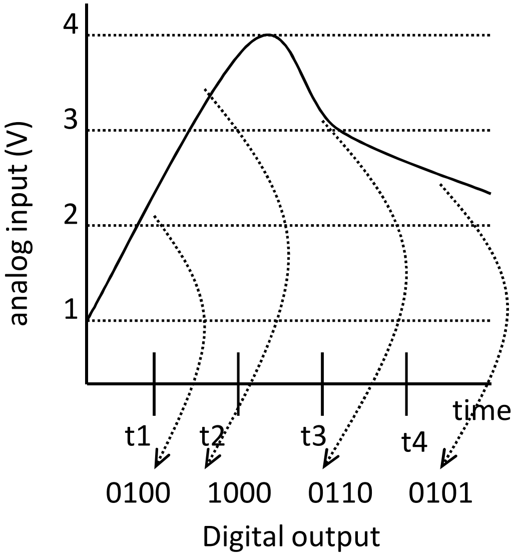
```
/* counter clockwise movement */
 if(dir==0){
    for(y=0;  y<=step; y++){
      for(z=19; z>=0; z - 4){
        isA=lookup[z];
        isB=lookup[z-1];
        notA=lookup[z -2];
        notB=lookup[z-3];
       delay( );
      }
    }
  }
}
void main( ){
  int z;
  int lookup[20] = {
    1,  1,  0,  0,
    0,  1,  1,  0,
    0,  0,  1,  1,
    1,  0,  0,  1,
    1,  1,  0,  0  };
  while(1){
     /*move forward, 15 degrees (2 steps) */
    move(1, 2);
    /* move backwards,  7.5 degrees
(1step)*/
    move(0, 1);
  }
}
```

Analog-to-digital converters

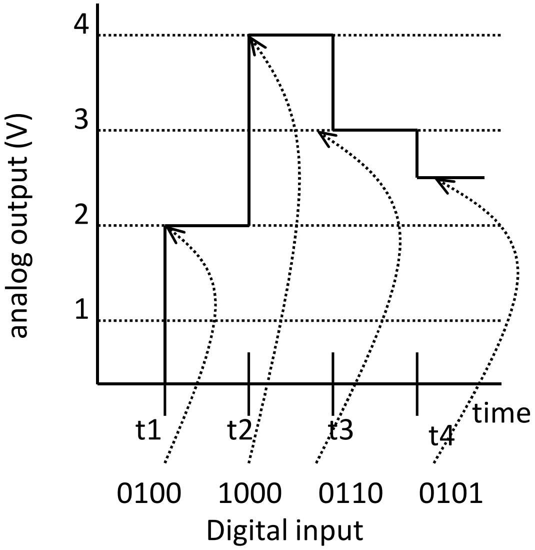


V_{max} = 7.5V	1111
7.0V	1110
6.5V	1101
6.0V	1100
5.5V	1011
5.0V	1010
4.5V	1001
4.0V	1000
3.5V	0111
3.0V	0110
2.5V	0101
2.0V	0100
1.5V	0011
1.0V	0010
0.5V	0001
0V	0000

proportionality



analog to digital



digital to analog

Digital-to-analog conversion using successive approximation

Given an analog input signal whose voltage should range from 0 to 15 volts, and an 8-bit digital encoding, calculate the correct encoding for 5 volts. Then trace the successive-approximation approach to find the correct encoding.

5/15 = d/(28-1)
d= 85

Encoding: 01010101

Successive-approximation method

½(V_{max} – V_{min}) = 7.5 volts
V_{max} = 7.5 volts.

0	0	0	0	0	0	0	0

½(7.5 + 0) = 3.75 volts
V_{min} = 3.75 volts.

0	1	0	0	0	0	0	0

½(7.5 + 3.75) = 5.63 volts
V_{max} = 5.63 volts

0	1	0	0	0	0	0	0

½(5.63 + 3.75) = 4.69 volts
V_{min} = 4.69 volts.

0	1	0	1	0	0	0	0

½(5.63 + 4.69) = 5.16 volts
V_{max} = 5.16 volts.

0	1	0	1	0	0	0	0

½(5.16 + 4.69) = 4.93 volts
V_{min} = 4.93 volts.

0	1	0	1	0	1	0	0

½(5.16 + 4.93) = 5.05 volts
V_{max} = 5.05 volts.

0	1	0	1	0	1	0	0

½(5.05 + 4.93) = 4.99 volts

0	1	0	1	0	1	0	1

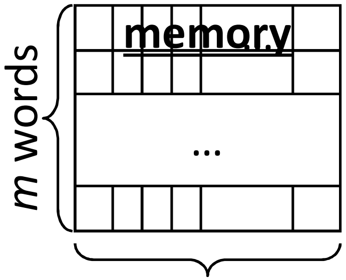
Introduction

- Embedded system’s functionality aspects
  - Processing
    - processors
    - transformation of data
  - Storage
    - memory
    - retention of data
  - Communication
    - buses
    - transfer of data

Memory: basic concepts

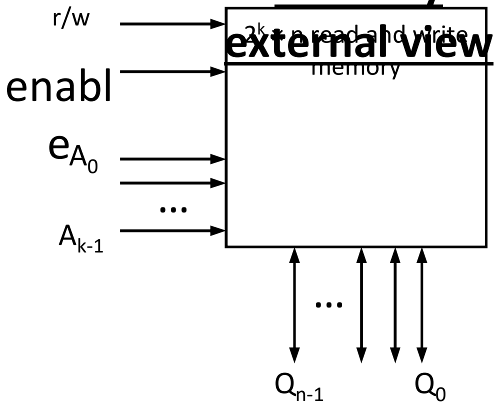
- Stores large number of bits
  - *m* x *n*: *m* words of *n* bits each
  - k = Log2(*m*) address input signals
  - or *m* = 2^k words
  - e.g., 4,096 x 8 memory:
    - 32,768 bits
    - 12 address input signals
    - 8 input/output data signals
- Memory access
  - r/w: selects read or write
  - enable: read or write only when asserted
  - multiport: multiple accesses to different locations simultaneously

m × n memory



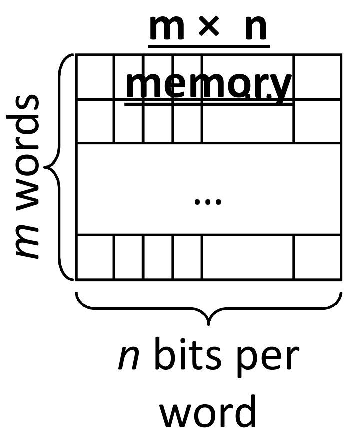
n bits per word

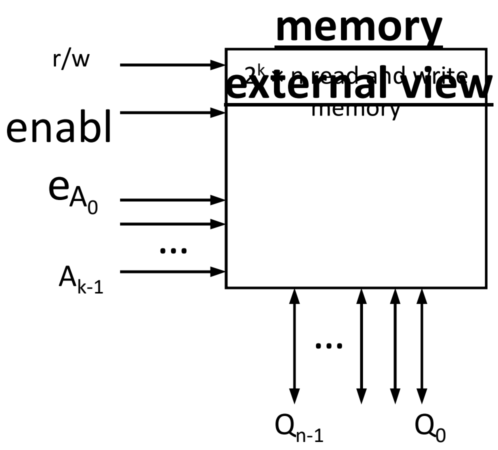
memory external view



Memory: basic concepts

- Stores large number of bits
  - *m* x *n*: *m* words of *n* bits each
  - k = Log$_2$(*m*) address input signals
  - or *m* = 2^k words
  - e.g., 4,096 x 8 memory:
    - 32,768 bits
    - 12 address input signals
    - 8 input/output data signals
- Memory access
  - r/w: selects read or write
  - enable: read or write only when asserted
  - multiport: multiple accesses to different locations simultaneously





Write ability

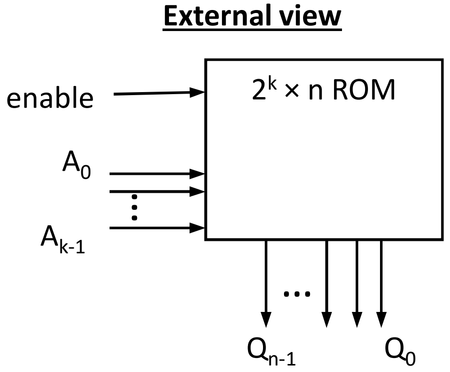
- Ranges of write ability
  - High end
    - processor writes to memory simply and quickly
    - e.g., RAM
  - Middle range
    - processor writes to memory, but slower
    - e.g., FLASH, EEPROM
  - Lower range
    - special equipment, “programmer”, must be used to write to memory
    - e.g., EPROM, OTP ROM
  - Low end
    - bits stored only during fabrication
    - e.g., Mask-programmed ROM
- In-system programmable memory
  - Can be written to by a processor in the embedded system using the memory
  - Memories in high end and middle range of write ability

Storage permanence

- Range of storage permanence
  - High end
    - essentially never loses bits
    - e.g., mask-programmed ROM
  - Middle range
    - holds bits days, months, or years after memory’s power source turned off
    - e.g., NVRAM
  - Lower range
    - holds bits as long as power supplied to memory
    - e.g., SRAM
  - Low end
    - begins to lose bits almost immediately after written
    - e.g., DRAM
- Nonvolatile memory
  - Holds bits after power is no longer supplied
  - High end and middle range of storage permanence

ROM: “Read-Only” Memory

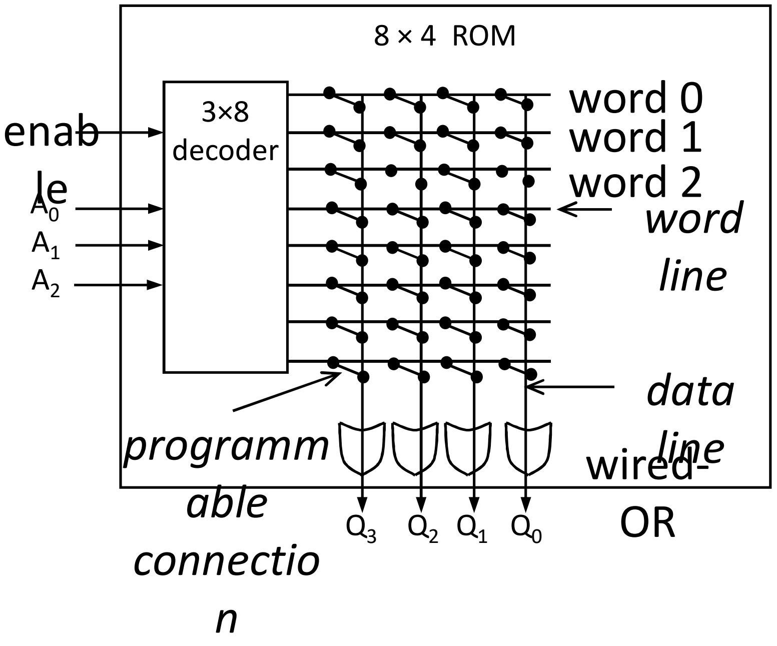
- Nonvolatile memory
- Can be read from but not written to, by a processor in an embedded system
- Traditionally written to, “programmed”, before inserting to embedded system
- Uses
  - Store software program for general-purpose processor
    - program instructions can be one or more ROM words
  - Store constant data needed by system
  - Implement combinational circuit



Example: 8 x 4 ROM

- Horizontal lines = words
- Vertical lines = data
- Lines connected only at circles
- Decoder sets word 2’s line to 1 if address input is 010
- Data lines Q_3 and Q_1 are set to 1 because there is a “programmed” connection with word 2’s line
- Word 2 is not connected with data lines Q_2 and Q_0
- Output is 1010

Internal view



Implementing combinational function

- Any combinational circuit of *n* functions of same *k* variables can be done with 2^k x *n* ROM

Mask-programmed ROM

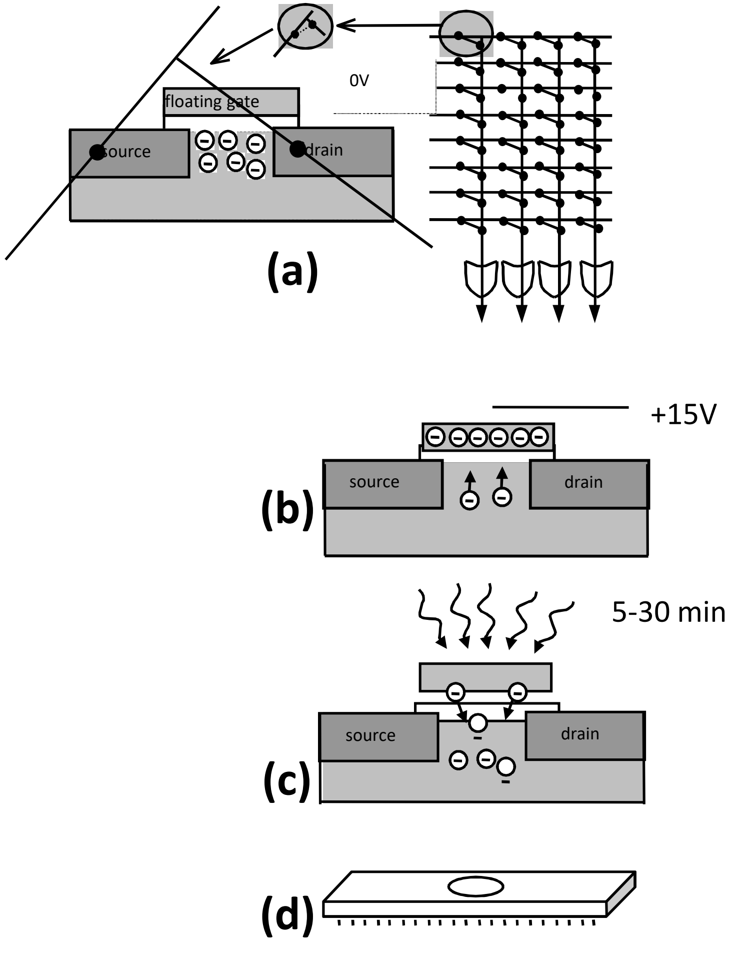
- Connections “programmed” at fabrication
  - set of masks
- Lowest write ability
  - only once
- Highest storage permanence
  - bits never change unless damaged
- Typically used for final design of high-volume systems
  - spread out NRE cost for a low unit cost

OTP ROM: One-time programmable ROM

- Connections “programmed” after manufacture by user
  - user provides file of desired contents of ROM
  - file input to machine called ROM programmer
  - each programmable connection is a fuse
  - ROM programmer blows fuses where connections should not exist
- Very low write ability
  - typically written only once and requires ROM programmer device
- Very high storage permanence
  - bits don’t change unless reconnected to programmer and more fuses blown
- Commonly used in final products
  - cheaper, harder to inadvertently modify

EPROM: Erasable programmable ROM

- **Programmable component is a MOS transistor**
  - Transistor has “floating” gate surrounded by an insulator
  - **(a)** Negative charges form a channel between source and drain storing a logic 1
  - **(b)** Large positive voltage at gate causes negative charges to move out of channel and get trapped in floating gate storing a logic 0
  - **(c)** (Erase) Shining UV rays on surface of floating-gate causes negative charges to return to channel from floating gate restoring the logic 1
  - **(d)** An EPROM package showing quartz window through which UV light can pass
- **Better write ability**
  - can be erased and reprogrammed thousands of times
- **Reduced storage permanence**
  - program lasts about 10 years but is susceptible to radiation and electric noise
- **Typically used during design development**



EEPROM: Electrically erasable programmable ROM

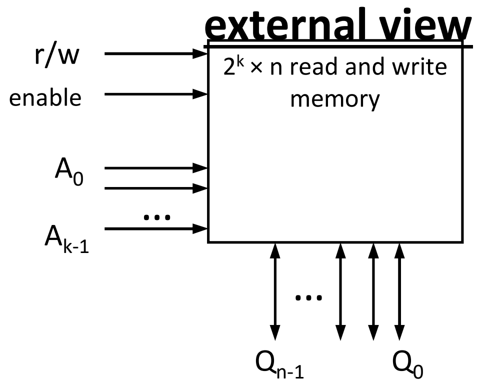
- Programmed and erased electronically
  - typically by using higher than normal voltage
  - can program and erase individual words
- Better write ability
  - can be in-system programmable with built-in circuit to provide higher than normal voltage
    - built-in memory controller commonly used to hide details from memory user
  - writes very slow due to erasing and programming
    - “busy” pin indicates to processor EEPROM still writing
  - can be erased and programmed tens of thousands of times
- Similar storage permanence to EPROM (about 10 years)
- Far more convenient than EPROMs, but more expensive

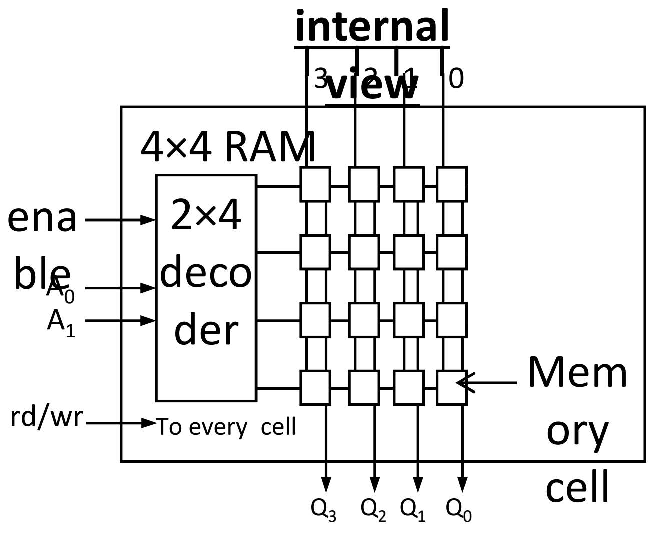
Flash Memory

- Extension of EEPROM
  - Same floating gate principle
  - Same write ability and storage permanence
- Fast erase
  - Large blocks of memory erased at once, rather than one word at a time
  - Blocks typically several thousand bytes large
- Writes to single words may be slower
  - Entire block must be read, word updated, then entire block written back
- Used with embedded systems storing large data items in nonvolatile memory
  - e.g., digital cameras, TV set-top boxes, cell phones

RAM: “Random-access” memory

- **Typically volatile memory**
  - bits are not held without power supply
- **Read and written to easily by embedded system during execution**
- **Internal structure more complex than ROM**
  - a word consists of several memory cells, each storing 1 bit
  - each input and output data line connects to each cell in its column
  - rd/wr connected to every cell
  - when row is enabled by decoder, each cell has logic that stores input data bit when rd/wr indicates write or outputs stored bit when rd/wr indicates read



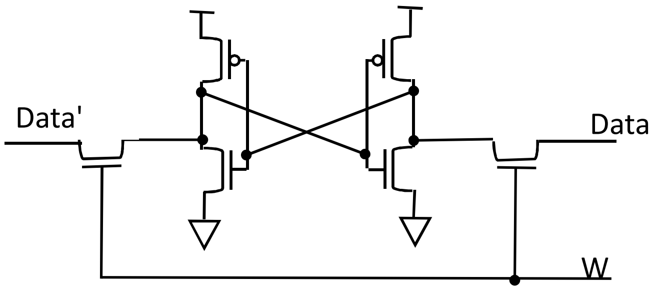


Basic types of RAM

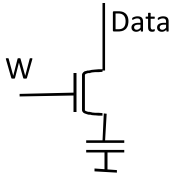
- SRAM: Static RAM
  - Memory cell uses flip-flop to store bit
  - Requires 6 transistors
  - Holds data as long as power supplied
- DRAM: Dynamic RAM
  - Memory cell uses MOS transistor and capacitor to store bit
  - More compact than SRAM
  - “Refresh” required due to capacitor leak
    - word’s cells refreshed when read
  - Typical refresh rate 15.625 microsec.
  - Slower to access than SRAM

memory cell internals

SRAM



DRAM



Ram variations

- PSRAM: Pseudo-static RAM
  - DRAM with built-in memory refresh controller
  - Popular low-cost high-density alternative to SRAM
- NVRAM: Nonvolatile RAM
  - Holds data after external power removed
  - Battery-backed RAM
    - SRAM with own permanently connected battery
    - writes as fast as reads
    - no limit on number of writes unlike nonvolatile ROM-based memory
  - SRAM with EEPROM or flash
    - stores complete RAM contents on EEPROM or flash before power turned off

Example: HM6264 & 27C256 RAM/ROM devices

- Low-cost low-capacity memory devices
- Commonly used in 8-bit microcontroller-based embedded systems
- First two numeric digits indicate device type
  - RAM: 62
  - ROM: 27
- Subsequent digits indicate capacity in kilobits

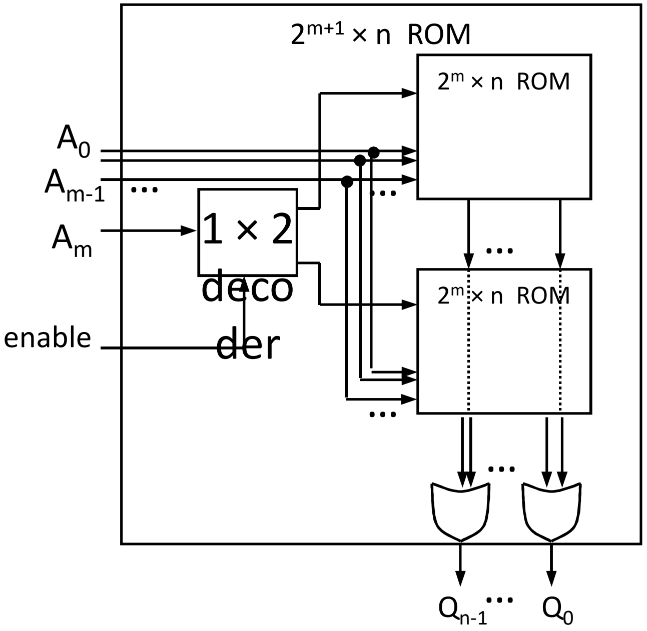
Example:
TC55V2325FF-100 memory device

- 2-megabit synchronous pipelined burst SRAM memory device
- Designed to be interfaced with 32-bit processors
- Capable of fast sequential reads and writes as well as single byte I/O

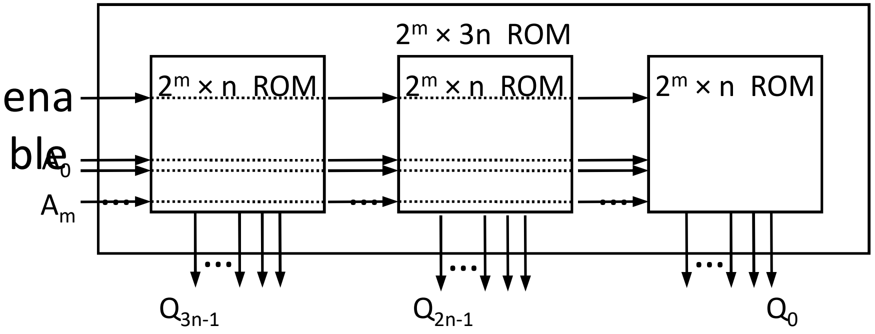
Composing memory

- Memory size needed often differs from size of readily available memories
- When available memory is larger, simply ignore unneeded high-order address bits and higher data lines
- When available memory is smaller, compose several smaller memories into one larger memory
  - Connect side-by-side to increase width of words
  - Connect top to bottom to increase number of words
    - added high-order address line selects smaller memory containing desired word using a decoder
  - Combine techniques to increase number and width of words

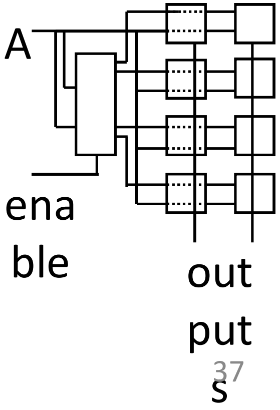
Increase number of words



Increase width of words

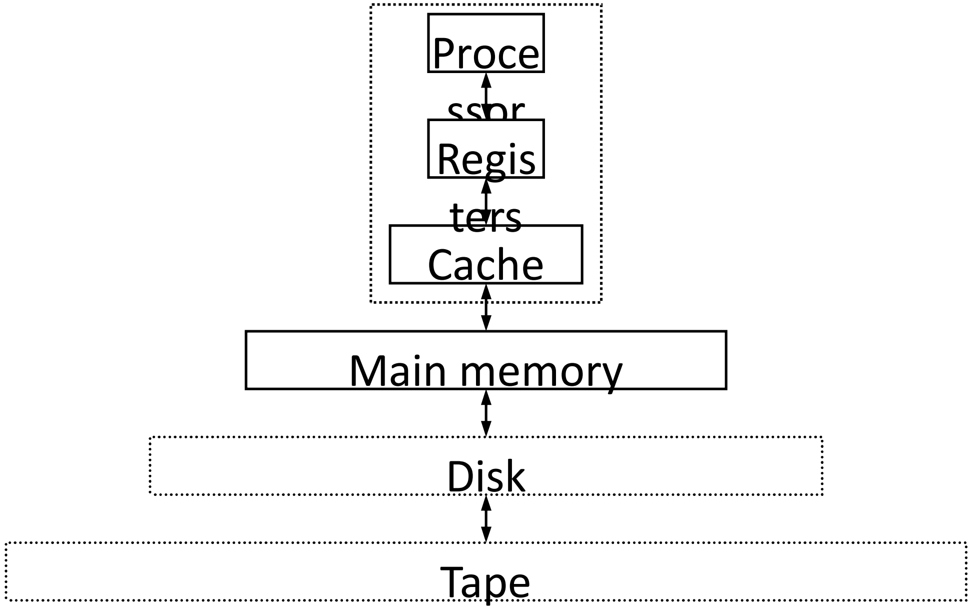


Increase number and width of words



Memory hierarchy

- Want inexpensive, fast memory
- Main memory
  - Large, inexpensive, slow memory stores entire program and data
- Cache
  - Small, expensive, fast memory stores copy of likely accessed parts of larger memory
  - Can be multiple levels of cache



Cache

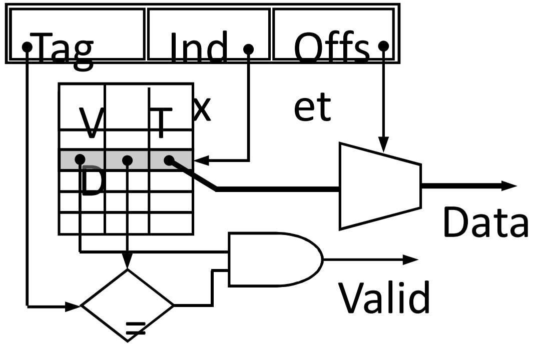
- **Usually designed with SRAM**
  - faster but more expensive than DRAM
- **Usually on same chip as processor**
  - space limited, so much smaller than off-chip main memory
  - faster access (1 cycle vs. several cycles for main memory)
- **Cache operation:**
  - Request for main memory access (read or write)
  - First, check cache for copy
    - cache hit
      - copy is in cache, quick access
    - cache miss
      - copy not in cache, read address and possibly its neighbors into cache
- **Several cache design choices**
  - cache mapping, replacement policies, and write techniques

Cache mapping

- Far fewer number of available cache addresses
- Are address’ contents in cache?
- Cache mapping used to assign main memory address to cache address and determine hit or miss
- Three basic techniques:
  - Direct mapping
  - Fully associative mapping
  - Set-associative mapping
- Caches partitioned into indivisible blocks or lines of adjacent memory addresses
  - usually 4 or 8 addresses per line

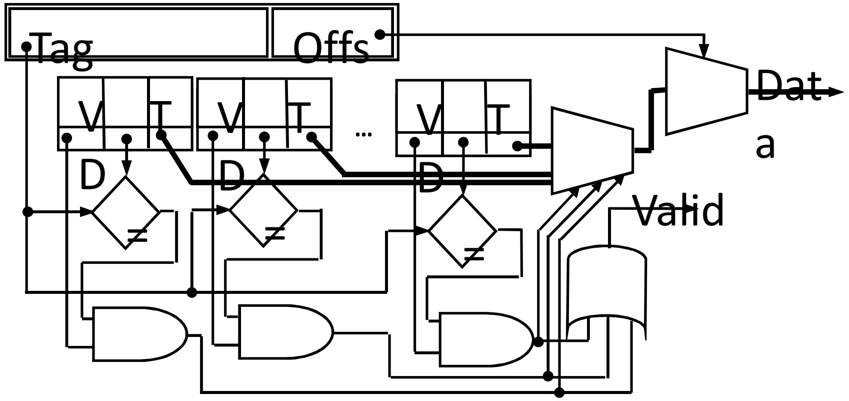
Direct mapping

- Main memory address divided into 2 fields
  - Index
    - cache address
    - number of bits determined by cache size
  - Tag
    - compared with tag stored in cache at address indicated by index
    - if tags match, check valid bit
- Valid bit
  - indicates whether data in slot has been loaded from memory
- Offset
  - used to find particular word in cache line



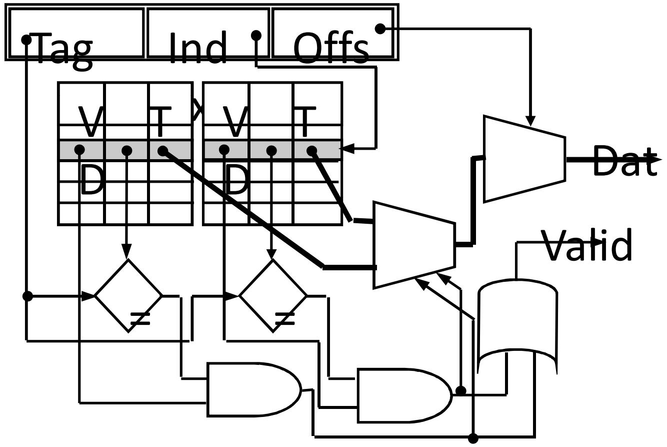
Fully associative mapping

- Complete main memory address stored in each cache address
- All addresses stored in cache simultaneously compared with desired address
- Valid bit and offset same as direct mapping



Set-associative mapping

- Compromise between direct mapping and fully associative mapping
- Index same as in direct mapping
- But, each cache address contains content and tags of 2 or more memory address locations
- Tags of that **set** simultaneously compared as in fully associative mapping
- Cache with set size N called N-way set-associative
  - 2-way, 4-way, 8-way are common



Cache-replacement policy

- Technique for choosing which block to replace
  - when fully associative cache is full
  - when set-associative cache's line is full
- Direct mapped cache has no choice
- Random
  - replace block chosen at random
- LRU: least-recently used
  - replace block not accessed for longest time
- FIFO: first-in-first-out
  - push block onto queue when accessed
  - choose block to replace by popping queue

Cache write techniques

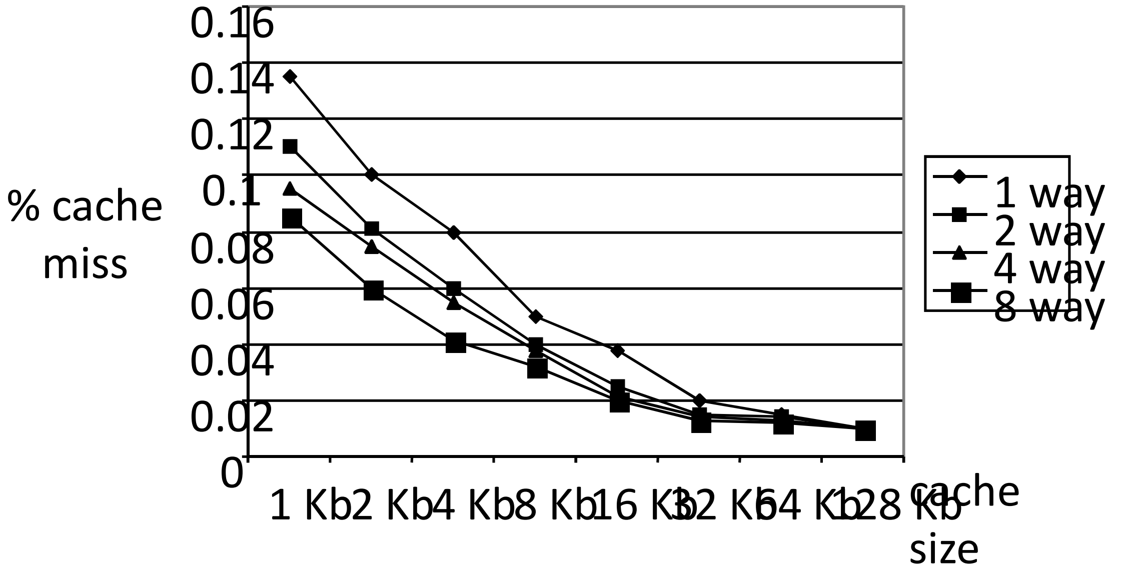
- When written, data cache must update main memory
- Write-through
  - write to main memory whenever cache is written to
  - easiest to implement
  - processor must wait for slower main memory write
  - potential for unnecessary writes
- Write-back
  - main memory only written when “dirty” block replaced
  - extra dirty bit for each block set when cache block written to
  - reduces number of slow main memory writes

Cache impact on system performance

- Most important parameters in terms of performance:
  - Total size of cache
    - total number of data bytes cache can hold
    - tag, valid and other house keeping bits not included in total
  - Degree of associativity
  - Data block size
- Larger caches achieve lower miss rates but higher access cost
  - e.g.,
    - 2 Kbyte cache: miss rate = 15%, hit cost = 2 cycles, miss cost = 20 cycles
      - avg. cost of memory access = (0.85 * 2) + (0.15 * 20) = 4.7 cycles
    - 4 Kbyte cache: miss rate = 6.5%, hit cost = 3 cycles, miss cost will not change
      - avg. cost of memory access = (0.935 * 3) + (0.065 * 20) = 4.105 cycles **(improvement)**
    - 8 Kbyte cache: miss rate = 5.565%, hit cost = 4 cycles, miss cost will not change
      - avg. cost of memory access = (0.94435 * 4) + (0.05565 * 20) = 4.8904 cycles **(worse)**

Cache performance trade-offs

- Improving cache hit rate without increasing size
  - Increase line size
  - Change set-associativity

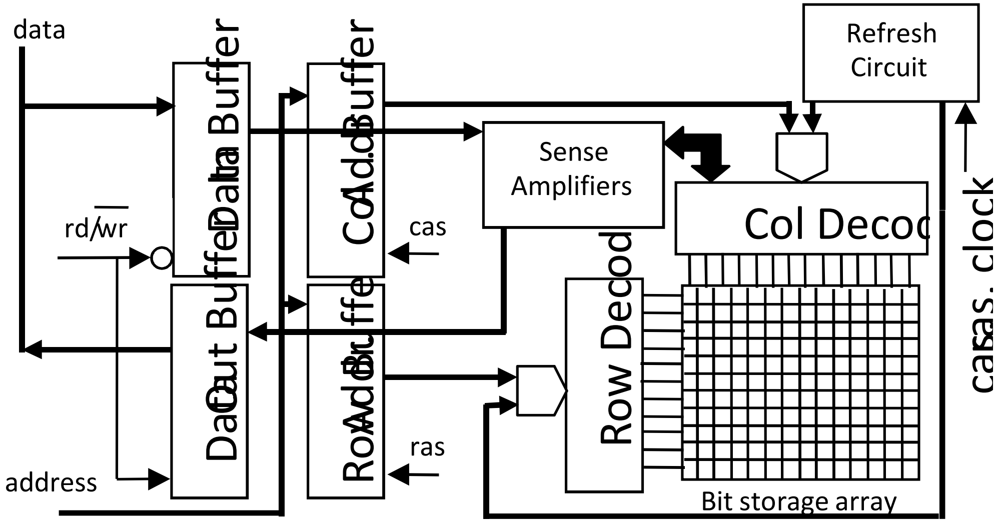


Advanced RAM

- DRAMs commonly used as main memory in processor based embedded systems
  - high capacity, low cost
- Many variations of DRAMs proposed
  - need to keep pace with processor speeds
  - FPM DRAM: fast page mode DRAM
  - EDO DRAM: extended data out DRAM
  - SDRAM/ESDRAM: synchronous and enhanced synchronous DRAM
  - RDRAM: rambus DRAM

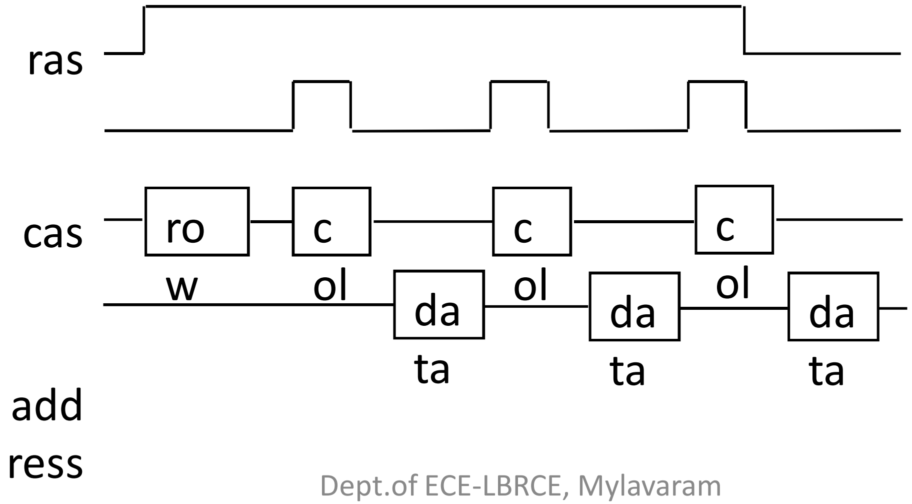
Basic DRAM

- Address bus multiplexed between row and column components
- Row and column addresses are latched in, sequentially, by strobing *ras* and *cas* signals, respectively
- Refresh circuitry can be external or internal to DRAM device
  - strobes consecutive memory address periodically causing memory content to be refreshed
  - Refresh circuitry disabled during read or write operation



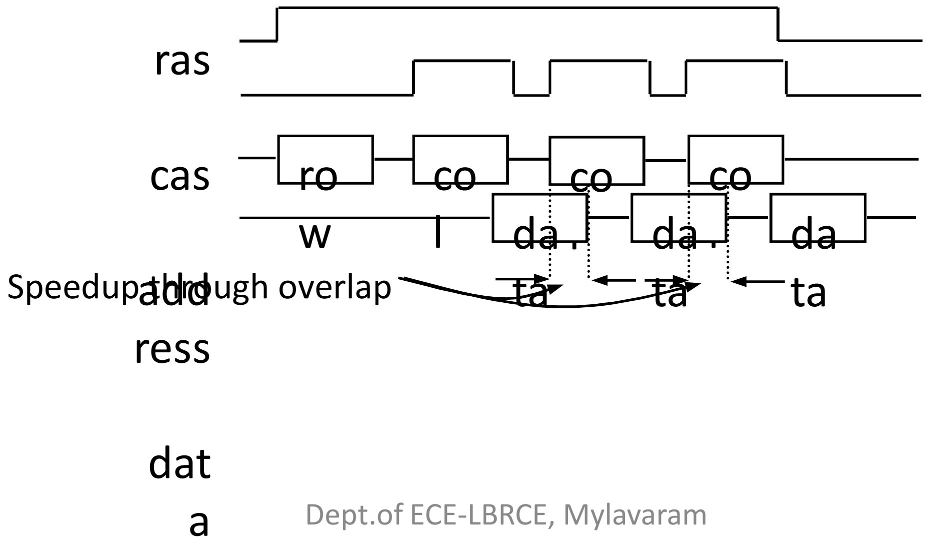
Fast Page Mode DRAM (FPM DRAM)

- Each row of memory bit array is viewed as a page
- Page contains multiple words
- Individual words addressed by column address
- Timing diagram:
  - row (page) address sent
  - 3 words read consecutively by sending column address for each
- Extra cycle eliminated on each read/write of words from same page



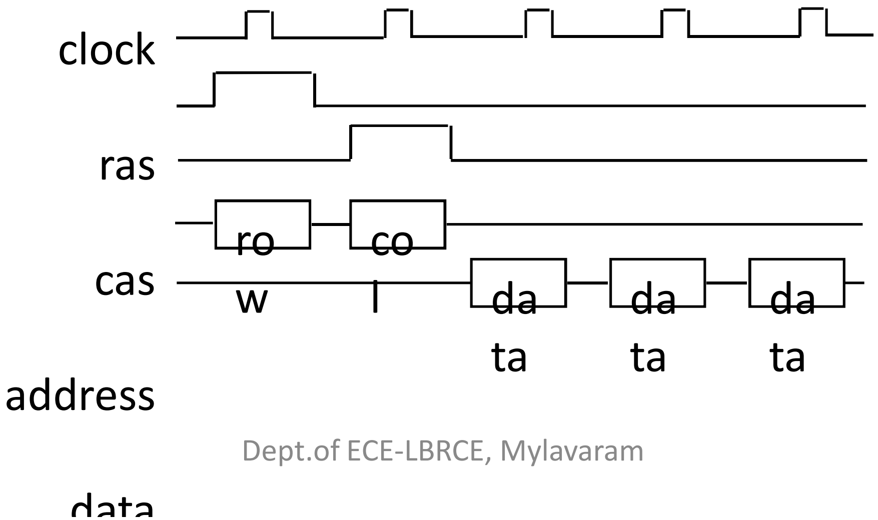
Extended data out DRAM (EDO DRAM)

- Improvement of FPM DRAM
- Extra latch before output buffer
  - allows strobing of *cas* before data read operation completed
- Reduces read/write latency by additional cycle



Synchronous and Enhanced Synchronous (ES) DRAM

- SDRAM latches data on active edge of clock
- Eliminates time to detect *ras/cas* and *rd/wr* signals
- A counter is initialized to column address then incremented on active edge of clock to access consecutive memory locations
- ESDRAM improves SDRAM
  - added buffers enable overlapping of column addressing
  - faster clocking and lower read/write latency possible



Rambus DRAM (RDRAM)

- More of a bus interface architecture than DRAM architecture
- Data is latched on both rising and falling edge of clock
- Broken into 4 banks each with own row decoder
  - can have 4 pages open at a time
- Capable of very high throughput

DRAM integration problem

- SRAM easily integrated on same chip as processor
- DRAM more difficult
  - Different chip making process between DRAM and conventional logic
  - Goal of conventional logic (IC) designers:
    - minimize parasitic capacitance to reduce signal propagation delays and power consumption
  - Goal of DRAM designers:
    - create capacitor cells to retain stored information
  - Integration processes beginning to appear

Memory Management Unit (MMU)

- Duties of MMU
  - Handles DRAM refresh, bus interface and arbitration
  - Takes care of memory sharing among multiple processors
  - Translates logic memory addresses from processor to physical memory addresses of DRAM
- Modern CPUs often come with MMU built-in
- Single-purpose processors can be used

Thank You