Embedded Systems Design: A Unified Hardware/Software Introduction

Chapter 4 Standard Single Purpose Processors: Peripherals

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

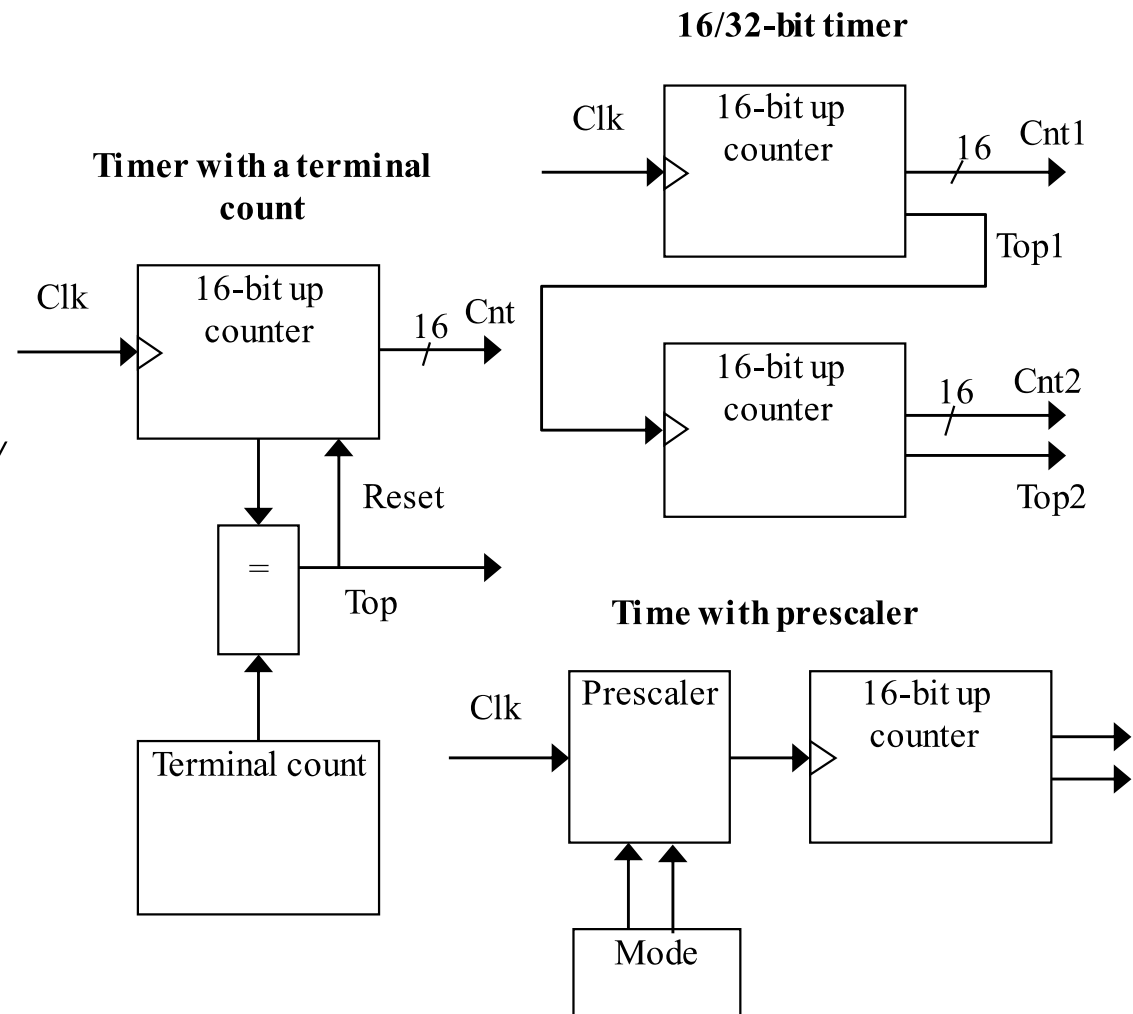Clk → [16-bit up counter] → 16 / Cnt
Top
Reset
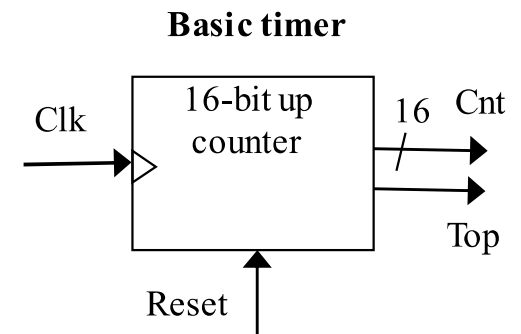
Counters

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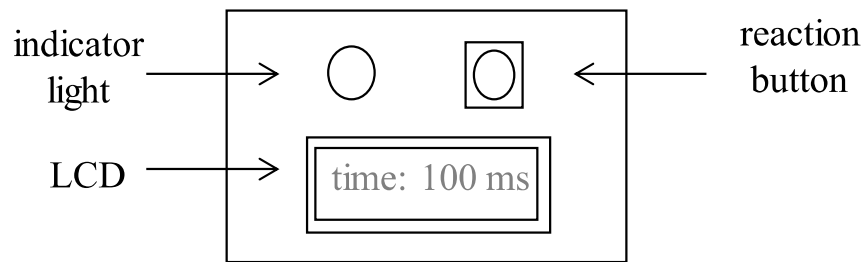   - Can often configure device as either a timer or counter

Timer/counter

Clk

| 2x1 mux |

Cnt_in

| 16-bit up counter |

16 Cnt

Top

Mode

Reset
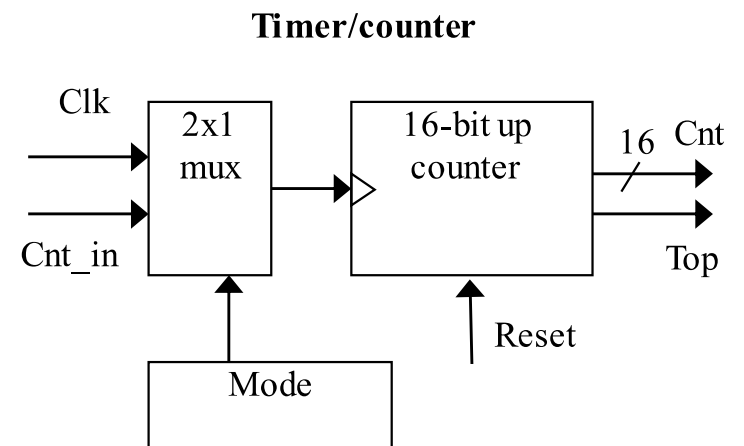
Other timer structures

- ## 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

Timer with a terminal count

16/32-bit timer

Time with prescaler

Example: Reaction Timer

indicator light →

reaction button

LCD → time: 100 ms

- **Measure time between turning light on and user pushing button**
  - 16-bit timer, clk period is 83.33 ns, counter increments every 6 cycles
  - Resolution = 6*83.33=0.5 microsec.
  - Range = 65535*0.5 microseconds = 32.77 milliseconds
  - Want program to count millisec., so initialize counter to 65535 – 1000/0.5 = 63535

```
/* main.c */

#define MS_INIT      63535
void main(void){
   int count_milliseconds = 0;

   configure timer mode
   set Cnt to MS_INIT

   wait a random amount of time
   turn on indicator light
   start timer

while (user has not pushed reaction button){
   if(Top) {
      stop timer
      set Cnt to MS_INIT
      start timer
      reset Top
      count_milliseconds++;
   }
}
   turn light off
   printf("time:  %i ms", count_milliseconds);
}
```
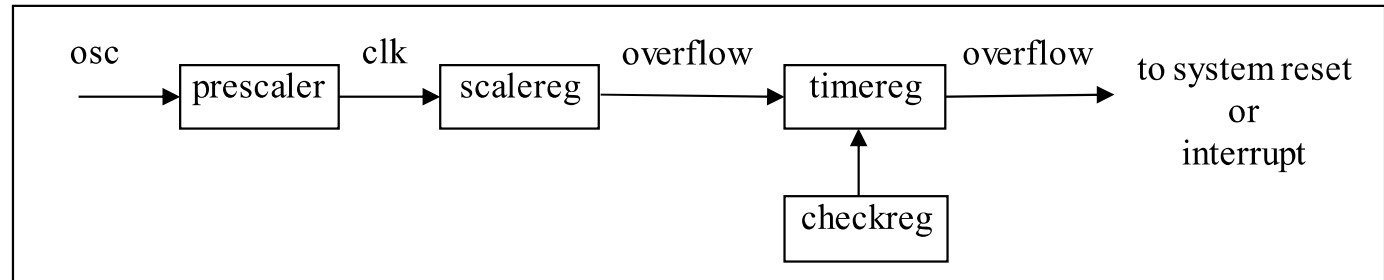
Watchdog timer

- **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.

```
osc          clk          overflow          overflow
  ──▶ prescaler ──▶ scalereg ──────────▶ timereg ──────────▶ to system reset
                                           ▲                      or
                                           │                    interrupt
                                        checkreg
```

```
/* 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

embedded device

Sending UART
Receiving UART

start bit
data
end bit

1 0 0 1 1 0 1 1
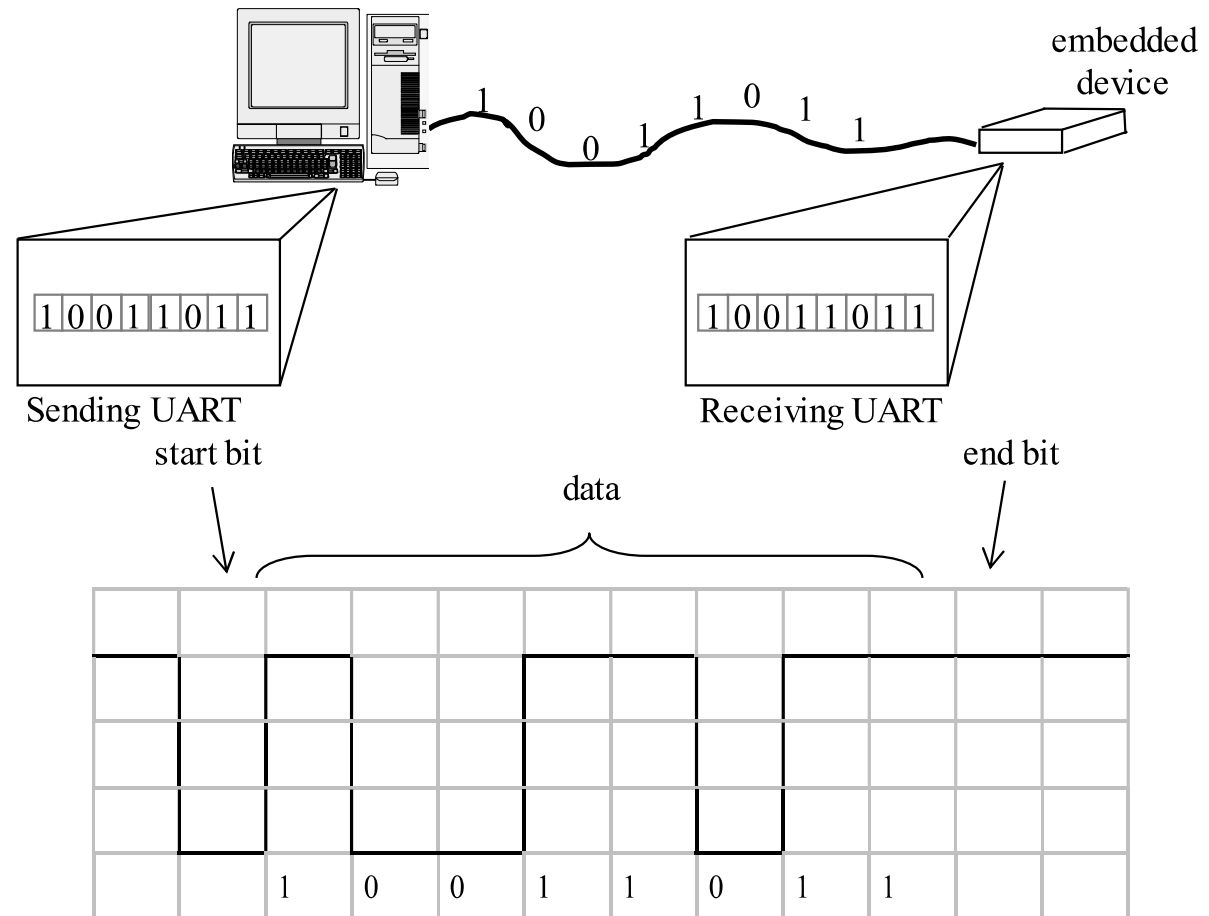
Pulse width modulator

- 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

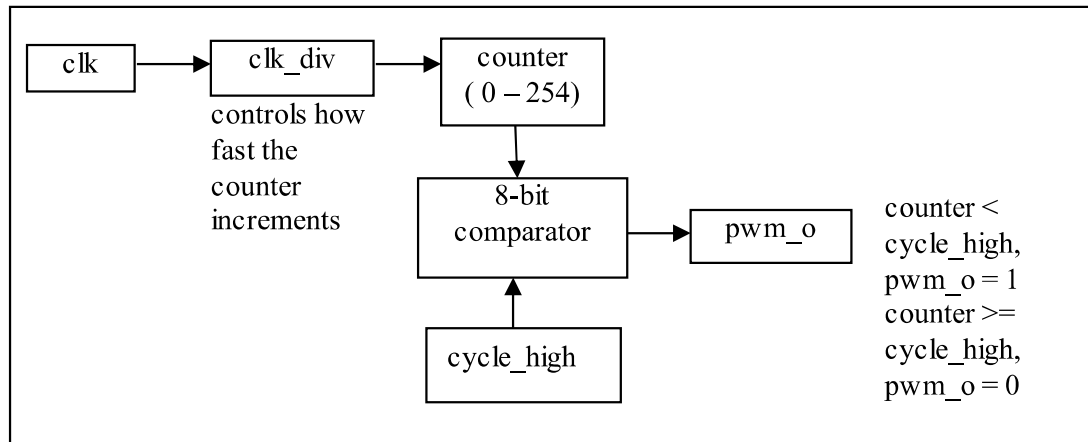pwm_o

clk
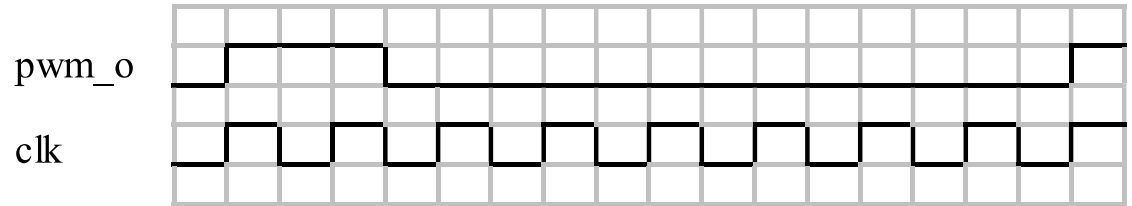
25% duty cycle – average pwm_o is 1.25V

pwm_o

clk
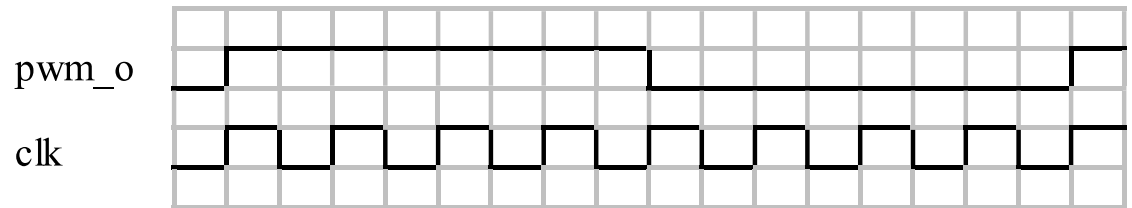
50% duty cycle – average pwm_o is 2.5V.

pwm_o

clk
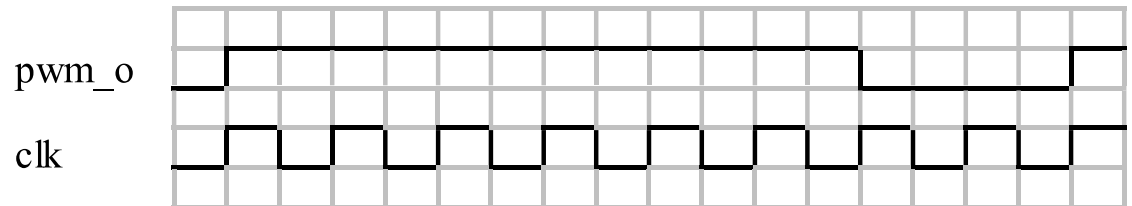
75% duty cycle – average pwm_o is 3.75V.

Controlling a DC motor with a PWM

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

In the internal structure diagram:

clk → clk_div → counter (0 – 254)

controls how fast the counter increments

8-bit comparator → pwm_o

cycle_high

counter < cycle_high, pwm_o = 1
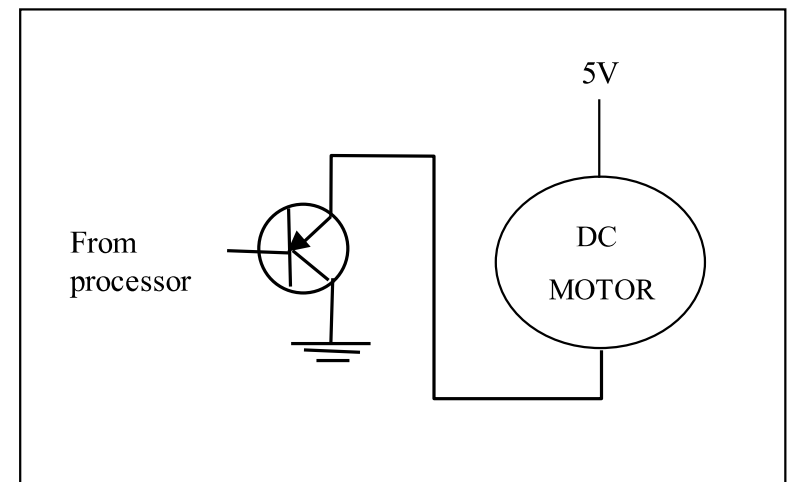counter >= cycle_high, pwm_o = 0
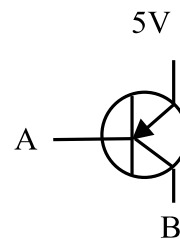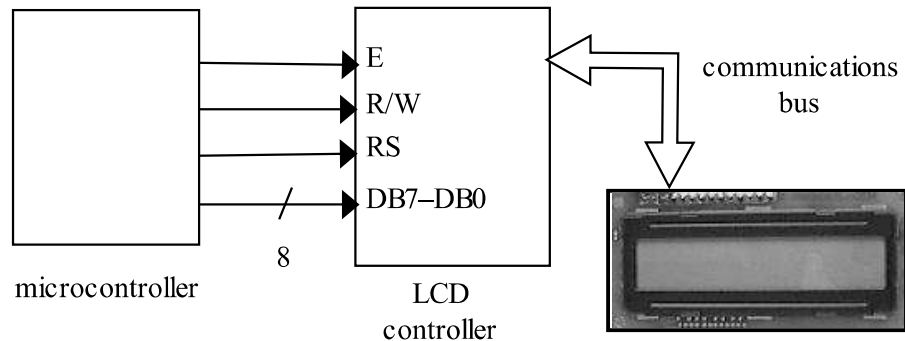
```
void main(void){

    /* 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 */
}
```

microcontroller   LCD controller   communications bus

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

N1
N2
N3
N4

M1
M2
M3
M4

k_pressed

4

key_code

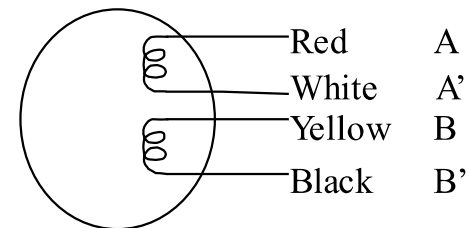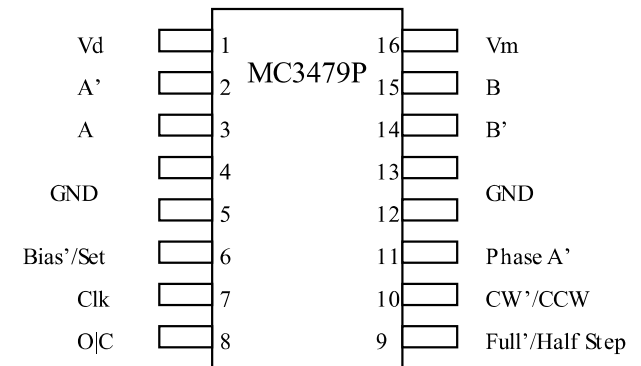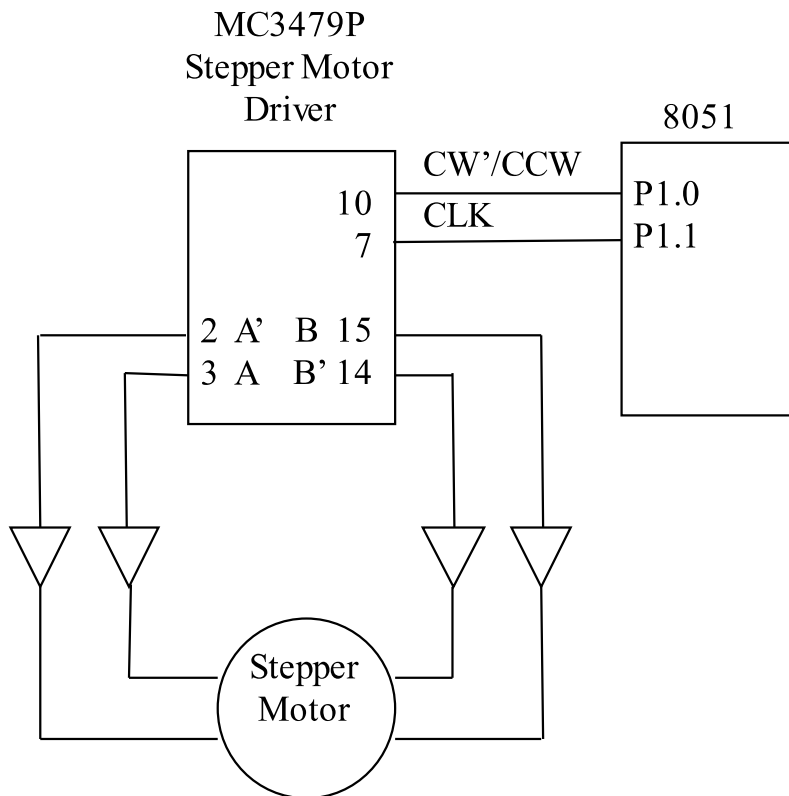key_code

keypad controller
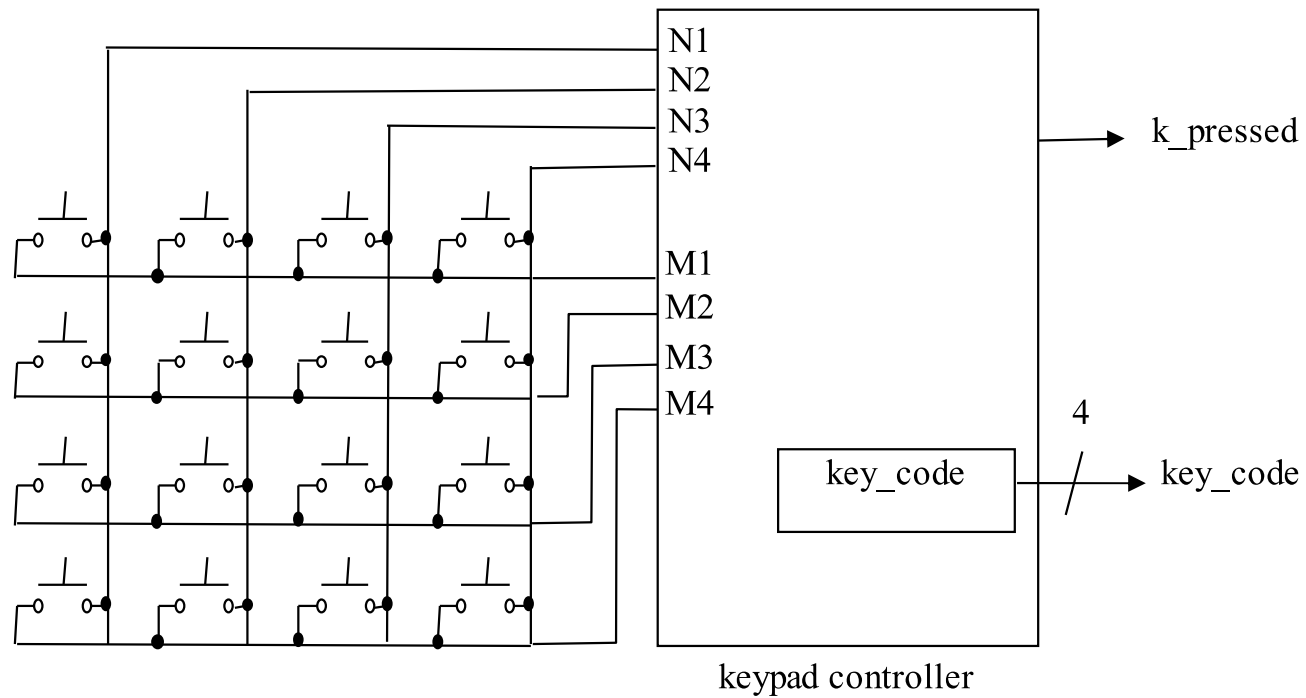
N=4, M=4

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	+	+	-	-

```
Vd        ┌─ 1   MC3479P   16 ─┐  Vm
A'        ├─ 2             15 ─┤  B
A         ├─ 3             14 ─┤  B'
          ├─ 4             13 ─┤
GND       ├─ 5             12 ─┤  GND
Bias'/Set ├─ 6             11 ─┤  Phase A'
Clk       ├─ 7             10 ─┤  CW'/CCW
O|C       └─ 8              9 ─┘  Full'/Half Step
```

Red   A
White A'
Yellow B
Black B'

Stepper motor with controller (driver)

MC3479P
Stepper Motor
Driver

8051

CW'/CCW

10

CLK

7

P1.0
P1.1

2 A'   B 15
3 A   B' 14

Stepper
Motor

```
/* 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.

+V

1K

Q1

A

B

Q2

1K
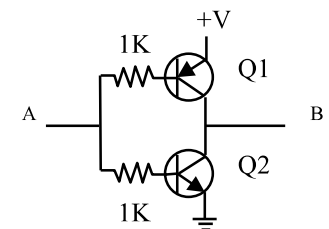
Stepper motor without controller (driver)

8051

```
P2.4 ── GND/ +V

P2.3
P2.2
P2.1
P2.0
```
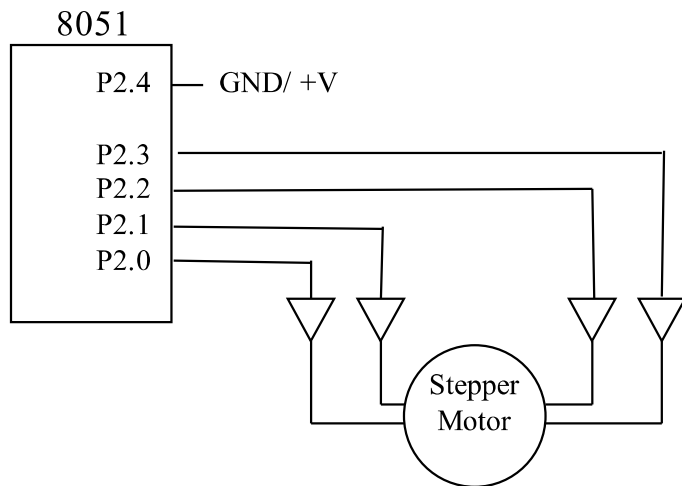
Stepper Motor

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.

```
              +V
        1K
  +V ──WW──         Q1
        1K
  A ──WW──          Q2
       330
```
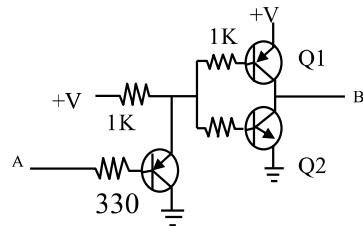
```
/*main.c*/
sbit notA=P2^0;
sbit isA=P2^1;
sbit notB=P2^2;
sbit isB=P2^3;
sbit dir=P2^4;

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);
  }
}
```
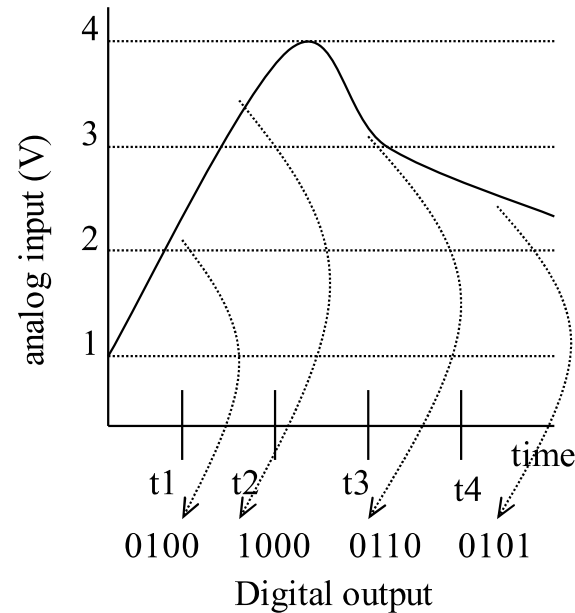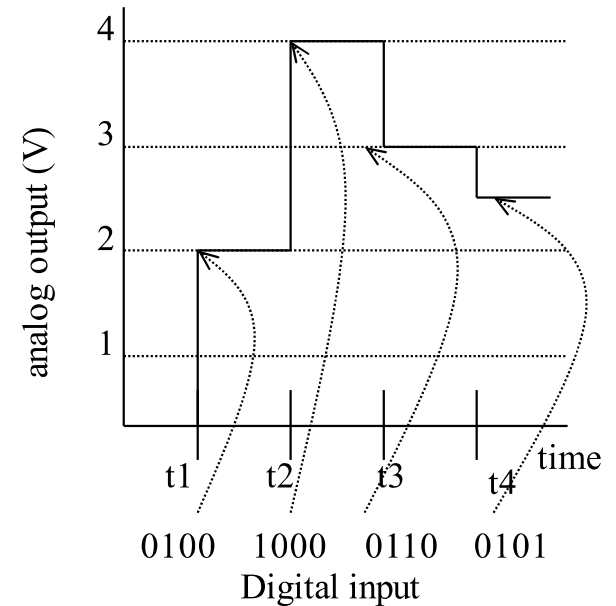
Analog-to-digital converters

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

analog input (V)

4

3

2

1

time

t1   t2   t3   t4

0100   1000   0110   0101

Digital output

analog output (V)

4

3

2

1

time

t1   t2   t3   t4

0100   1000   0110   0101

Digital input

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.

$5/15 = d/(2^8-1)$

$d= 85$

Encoding: 01010101

Successive-approximation method

$\frac{1}{2}(V_{max} - V_{min}) = 7.5$ volts

$V_{max} = 7.5$ volts.

0	0	0	0	0	0	0	0

$\frac{1}{2}(5.63 + 4.69) = 5.16$ volts

$V_{max} = 5.16$ volts.

0	1	0	1	0	0	0	0

$\frac{1}{2}(7.5 + 0) = 3.75$ volts

$V_{min} = 3.75$ volts.

0	1	0	0	0	0	0	0

$\frac{1}{2}(5.16 + 4.69) = 4.93$ volts

$V_{min} = 4.93$ volts.

0	1	0	1	0	1	0	0

$\frac{1}{2}(7.5 + 3.75) = 5.63$ volts

$V_{max} = 5.63$ volts

0	1	0	0	0	0	0	0

$\frac{1}{2}(5.16 + 4.93) = 5.05$ volts

$V_{max} = 5.05$ volts.

0	1	0	1	0	1	0	0

$\frac{1}{2}(5.63 + 3.75) = 4.69$ volts

$V_{min} = 4.69$ volts.

0	1	0	1	0	0	0	0

$\frac{1}{2}(5.05 + 4.93) = 4.99$ volts

0	1	0	1	0	1	0	1