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

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
    - \textit{Number of clock cycles} = \textit{Desired time interval} / \textit{Clock period}
- Cascaded counters
- Prescaler
  - Divides clock
  - Increases range, decreases resolution

16/32-bit timer

Example: Reaction Timer

- 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 = \(6 \times 83.33 = 0.5 \) 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
            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.

```
/* 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
  }
```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

Controlling a DC motor with a PWM

```c
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.

<table>
<thead>
<tr>
<th>Input Voltage</th>
<th>% of Maximum Voltage Applied</th>
<th>RPM of DC Motor</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2.5</td>
<td>50</td>
<td>1840</td>
</tr>
<tr>
<td>3.75</td>
<td>75</td>
<td>6900</td>
</tr>
<tr>
<td>5.0</td>
<td>100</td>
<td>9200</td>
</tr>
</tbody>
</table>

Relationship between applied voltage and speed of the DC Motor
**LCD controller**

```c
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

<table>
<thead>
<tr>
<th>RS</th>
<th>E</th>
<th>DB7</th>
<th>DB6</th>
<th>DB5</th>
<th>DB4</th>
<th>DB3</th>
<th>DB2</th>
<th>DB1</th>
<th>DB0</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Clear all display, move cursor home</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Return cursor home</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Set cursor move direction and/or specifies not to shift display</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>TURNS OFF all displays, cursor</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Move cursor and shifts display</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Set interface data length, number of display lines, and character font</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>WRITE DATA</td>
<td>Write Data</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Keypad controller**

```
key_code = k_pressed;
```
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

<table>
<thead>
<tr>
<th>Sequence</th>
<th>A</th>
<th>B</th>
<th>A'</th>
<th>B'</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>5</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Stepper motor with controller (driver)

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.

```c
void main(void){
  /* 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){
    /* main(void)*/
    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){
      /* main(void)*/
      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;
      }
    }
  }
}
```

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.

```c
#include <stdio.h>

#define P20 0
#define P21 1
#define P22 2
#define P23 3
#define P24 4
#define P25 5
#define P26 6
#define P27 7

#define notA P2^0
#define isA P2^1
#define notB P2^2
#define isB P2^3
#define 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, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0};
    while(1){
        /*move forward, 15 degrees (2 steps) */
        move(1, 2);
        /* move backwards, 7.5 degrees (1 step)*/
        move(0, 1);
    }
}
```

Analog-to-digital converters

![Analog-to-digital converters graph](image-url)
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.

\[
\frac{5}{15} = \frac{d}{(2^8-1)}
\]

Encoding: 01010101

**Successive-approximation method**

\[
\frac{1}{2}(V_{\text{max}} - V_{\text{min}}) = 7.5 \text{ volts}
\]

\[
V_{\text{max}} = 7.5 \text{ volts.}
\]

\[
\frac{1}{2}(7.5 + 0) = 3.75 \text{ volts}
\]

\[
V_{\text{max}} = 3.75 \text{ volts.}
\]

\[
\frac{1}{2}(7.5 + 3.75) = 5.63 \text{ volts}
\]

\[
V_{\text{max}} = 5.63 \text{ volts.}
\]

\[
\frac{1}{2}(5.63 + 4.69) = 5.16 \text{ volts}
\]

\[
V_{\text{max}} = 5.16 \text{ volts.}
\]