

Administrative Stuff

- Homework2 posted on the www
 - Due Thursday 10/10
 - Problem 10 modified

10. (10 points) A watchdog timer that uses a cascaded 4-bit up-counter is connected to a 18 MHz oscillator. A timeout should occur if the timer watchdog timer is not reset within 8.05 milliseconds. What value should be loaded into the up-counter prior when the function is called?

- Quiz 1 10/15
 - Cover ESD chapter 1,2,3,4,5

CS122A: Embedded System Design, Fall02 1

Standard Single Purpose Processors

Peripherals

2

Outline

- Timers, Counters, Watchdog Timers
- UART
- Pulse Width Modulators
- LCD Controllers
- Keypad Controllers
- Stepper Motor Converters
- Analog-to-Digital Converters
- Real-Time Clocks

CS122A: Embedded System Design, Fall02 3

Single-purpose processors

- Performs specific computation task
 - Faster, smaller, less power
 - Free up GPP to do other stuff
- Custom single-purpose processors
 - Designed by embedded system designer for a unique task
- Standard single-purpose processors
 - Design by IC vendor for sale to a large number of system designers
 - "Off-the-shelf" - pre-designed for a common task
 - Low cost, due to large volume for a variety of product
 - a.k.a. peripherals
 - Often exist on-chip for ASIPs i.e. "On-chip peripherals"
 - Examples
 - UART, PWM
 - Controller/driver (LCD, Keypad, Stepper...)
 - Analog/digital converters
 - Timer, counter, clocks

CS122A: Embedded System Design, Fall02 4

Timers

- Measures time intervals to either
 - Generate timed output events
 - E.g., hold traffic light green for 10 s
 - Measure duration between input events
 - E.g., measure a car's speed using two separate sensors

CS122A: Embedded System Design, Fall02 5

Counting clock pulses

- Given 10 ns period, timer "counts" 20,000 pulses
 - 200 microseconds have passed
- Resolution: min time interval timer can measure
 - Given clock frequency of 100 MHz
 - Resolution = clock period = 1/(100 MHz) = 10 ns
- Range: max time interval timer can measure
 - E.g., 16-bit timer counts up to 65535
- Given a 10 ns period and a 16-bit timer
 - 65,535 * 10 ns = 655.35 microseconds

CS122A: Embedded System Design, Fall02 6

Timers/Counters

- Count pulses on a general input signal
 - E.g., count cars passing over a single sensor
- Can often configure device as either a timer or counter
 - E.g., Mode selects input
 - Timer: input = Clk
 - Counter: input = Cnt_in
- Top: indicates top count reached, wrap-around
 - Increase range
- Timer + counter = rate
 - E.g. to determine speed
 - Count wheel pulses in a second

7

CS122A: Embedded System Design, Fall02

Interval timer

- Indicates when desired time interval has passed
- Terminal count set to desired interval
 - $Terminal\ count = Desired\ time\ interval / Clock\ period$
- Top asserted when terminal count reached
 - Inform user (usually connected to interrupt)
 - Reset counter
- Alternative, load terminal count and use a down counter
 - NOR gate instead of comparator
 - Nor all bits and check for output 1

Timer with a terminal count

8

CS122A: Embedded System Design, Fall02

Cascaded counters and prescaler

- Cascaded counters
 - Input top of first counter to the second counter
 - e.g., two 16-bit counters
 - can be configured as a 16-bit or a 32-bit timer
- Prescaler
 - Divides clock frequency
 - Increases range by decreasing resolution
 - e.g., 16-bit timer with 100 MHz clock, resolution of 10 ns
 - range = $65,535 * 10\ ns = 655.35\ microseconds$
 - with prescaler configured to divide by 8
 - resolution = 80 ns
 - range = $65,535 * 80\ ns = 5.24\ milliseconds$

Timer with prescaler

9

CS122A: Embedded System Design, Fall02

Implementing Timer/Counter with GPP

- Timer
 - Pick a few very predictable instruction
 - NOP
 - Form a loop to count to desired time
- Counter
 - Use interrupt pin
 - Write ISR to increment counts
- Inefficient and probably costly

10

CS122A: Embedded System Design, Fall02

Example: Reaction Timer

- Measure time between turning light on and user pushing button
 - Requirement:
 - Display in ms precision
 - Reaction may take seconds
 - Use a 16-bit timer with a microcontroller
 - Increment timer every microcontroller instruction cycle, i.e. 6 clock cycles
 - Clock frequency is 12 Mhz, clock period is 83.33ns
 - Resolution = 1 instruction cycle = 6 clock cycles = $6 * 83.33 = 0.5\ us$
 - Range = $65535 * 0.5\ us = 32.77\ ms$

11

CS122A: Embedded System Design, Fall02

Example: Reaction Timer

- Resolution ~ 0.5 us
 - Only need 1ms
- Range = 32.77 ms
 - Needs to be in seconds
- Write program to add up the ms
- Use the peripheral to count to 1ms
 - $1ms / (0.5us\ resolution) = 2000$
 - Initialize counter to
 - $65535 - 2000 = 63535$
- Add the overflow pulses in microcontroller's register
 - i.e. count the number of ms
 - Can put this function in ISR
- ...could have used a prescaler also

12

CS122A: Embedded System Design, Fall02

Example: Reaction Timer

```

/* 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);
    }
    }
    
```

Possible source of inaccuracy

CS122A: Embedded System Design, Fall02 13

Watchdog timer

- Must reset timer every X time unit
 - Else timer generates a signal
- Common use:
 - Detect failure
 - Undesired infinite loop, waiting for input never occurs...etc
 - Reset watchdog timer sufficiently frequently during normal operation
 - Self-reset
 - Connect fail signal to microcontroller reset pin
 - Connect fail signal to interrupt
 - Or both: Use ISR to diagnose the problem before reset
 - Embedded system must self-recover
 - Timeouts

CS122A: Embedded System Design, Fall02 14

An example: automatic teller machine

- Close session if no keypad is pressed for 2 mins
- osc = 12MHz
- prescaler divide frequency by 12 to get clk = 1MHz
- 11 bit scalereg, overflow at a rate of..
 - $1\text{MHz}/2^{11} = 488.28125\text{Hz} = 2.048\text{ us (period)} \sim 2\text{us}$
- 16-bit timer, 2 us resolution
- If timer is not reset within X ms, watchdog barks
 - Timerreg value = $2*(2^{16}-1)-X = 131070-X$
- For 2 min., X = 120,000, Timerreg value = 11070

CS122A: Embedded System Design, Fall02 15

Automatic teller machine

```

/* 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 timerreg. Zero is loaded into
    scalereg and 11070 is loaded into
    timerreg */
    checkreg = 1
    scalereg = 0
    timerreg = 11070
}

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

CS122A: Embedded System Design, Fall02 16

Serial Transmission Using UARTs

- UART: Universal Asynchronous Receiver/ Transmitter
 - Takes parallel data and transmits serially
 - Receives serial data and converts to parallel
 - Usually one byte of data at a time
- Serial communication used when:
 - Long distance
 - Few available I/O ports

CS122A: Embedded System Design, Fall02 17

Transmission protocol

- Connected UARTs must agree on:
 - Baud rate (2400, 19.2K, 56.6K...)
 - Synchronizes speed of data exchange
 - Signal changes per second
 - Bit rate usually higher
- Parity
 - Extra bit for simple error checking
 - Even, odd, or none
- Start bit, stop bit

CS122A: Embedded System Design, Fall02 18

Implementing UART with GPP

- **Transmission**
 - Create a routine to send data serially over an I/O port
 - Utilizing a timer to control the rate
- **Reception**
 - Use an ISR
 - Capture serial data from another I/O port
 - Whenever such data arrives
- **Inefficient and probably costly**

19

CS122A: Embedded System Design, Fall02

Pulse width modulator

- Generates pulses with specific high/low times
- **Duty cycle: % time high**
 - Square wave: 50% duty cycle
- **Typical usage**
 - Control average voltage to electric device
 - Simpler than DC-DC converter or digital-analog converter
 - DC motor speed, dimmer lights
 - Encode commands
 - Receiver uses timer to decode
 - E.g. remote control car turn right on 1ms pulse, left on 4ms...

20

CS122A: Embedded System Design, Fall02

Pulse width modulator

5V high, 0V low

21

CS122A: Embedded System Design, Fall02

Example: Controlling DC motor with PWM

| Input Voltage | % of Maximum Voltage Applied | RPM of DC Motor | Counter |
|---------------|------------------------------|-----------------|---------|
| 0 | 0 | 0 | 0 |
| 2.5 | 50 | 4600 | 127 |
| 3.75 | 75 | 6900 | 191 |
| 5.0 | 100 | 9200 | 254 |

Relationship between applied voltage and speed of the DC Motor

Internal Structure of PWM

22

CS122A: Embedded System Design, Fall02

Controlling a DC motor with a PWM

```

void main(void){
    /* controls period */
    PWMP = 0xff;
    /* controls duty cycle */
    PWM1 = 0x7f; /* 4600rpm */
    while(1){};
}
    
```

From processor

5V

DC MOTOR

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

23

CS122A: Embedded System Design, Fall02

Liquid Crystal Display

- **It's everywhere**
 - Watches, fax, copier, calculators...
- **Reflective LCD**
 - Polarized light pass through LCM and reflected back
- **Absorption LCD**
 - Polarized light pass through LCM and get absorbed
- **Common LCD**
 - 7-segment LCD
 - Dot Matrix LCD
- **Can display in normal or inverted form**

24

CS122A: Embedded System Design, Fall02

Liquid crystal display controller

- Controller provides simple interface
 - Control words:
 - Enable, clear display, brighten, display cursor, etc.
 - Data words:
 - ASCII character

25

CS122A: Embedded System Design, Fall02

Example: LCD Initialization Sequence

- RS set low indicates control word being sent
- Controller decodes control words and performs actions
- RS set high when data word sent

| RS | RW | DB7 | DB6 | DB5 | DB4 | DB3 | DB2 | DB1 | DB0 | 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 | ID | 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 | SC | RL | * | * | Move cursor and shifts display |
| 0 | 0 | 0 | 0 | 1 | IL | N | F | * | * | Sets interface data length, number of display lines, and character font |
| 1 | 0 | WRITE DATA | | | | | | | | Writes Data |

26

CS122A: Embedded System Design, Fall02

LCD controller

| CODES | |
|---------------------------|-----------------|
| ID = 1 cursor moves left | DL = 1 8-bit |
| ID = 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 | |

Controller pseudocode to write a character to an LCD

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

27

CS122A: Embedded System Design, Fall02

Keypad controller

- Decodes rather than controls
- Simple example
 - Scans row and column outputs to detect a button press
 - Can be polled at a particular time interval
 - Or can generate interrupt
 - 0.1s sampling rate

28

CS122A: Embedded System Design, Fall02

Stepper motor controller

- Exists in disk drive, printer, photocopy, fax machine, robots, camcorder VCR
- 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 | + | + | - | - |

29

CS122A: Embedded System Design, Fall02

Stepper motor with controller (driver)

- Need only set the direction and pulse the clock

```
/* 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;
}
```

30

CS122A: Embedded System Design, Fall02

Current amplifier

- Output pins of MC3749P do not provide enough current to drive stepper motor
- Current amplifying buffer
 - Q1: MJE3055T NPN transistor
 - Q2: MJE2955T PNP transistor

31

CS122A: Embedded System Design, Fall02

Stepper motor without controller (driver)

- Direction must be controlled manually
- 8051 must send current through coils in correct sequence to get desired movement

32

CS122A: Embedded System Design, Fall02

Stepper motor without controller (driver)

```

/*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 */ /* counter clockwise */
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(); } } }
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, /* move forward, 15 degrees
0, 1, 1, 0, /* (2 steps) */
0, 0, 1, 1, /* move backwards,
1, 0, 0, 1, /* 7.5 degrees (1step) */
1, 1, 0, 0 };
while(1){
/*move forward, 15 degrees
(2 steps) */
move(1, 2);
/* move backwards,
7.5 degrees (1step) */
move(0, 1); } }
    
```

33

CS122A: Embedded System Design, Fall02

Analog-to-digital converters

- Analog
 - Continuously valued signal
 - Infinite possible values in between
 - Temperature, speed, etc.
- Digital
 - Discretely valued signal
 - Encoded in binary for computing systems
- Very common for embedded system
 - Environment typically involves many analog signals

34

CS122A: Embedded System Design, Fall02

Conversion

| | |
|------|------|
| 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 to digital

proportionality

35

CS122A: Embedded System Design, Fall02

Conversion

| | |
|------|------|
| 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 |

digital to analog

proportionality

36

CS122A: Embedded System Design, Fall02

Conversion

| | | |
|--|--|---|
| $V_{min} = 7.5V$ 7.0V 6.5V 6.0V 5.5V 5.0V 4.5V 4.0V 3.5V 3.0V 2.5V 2.0V 1.5V 1.0V 0.5V 0V | 1111 1110 1101 1100 1011 1010 1001 1000 0111 0110 0101 0100 0011 0010 0001 0000 | <ul style="list-style-type: none"> • Ratio: $e / V_{max} = d / (2^n - 1)$ <ul style="list-style-type: none"> - e = present analog voltage - V_{max} = max analog voltage value - d = present digital encoding - n = number of bits for encoding • E.g. e=3, n=4 <ul style="list-style-type: none"> - $3/7.5 = d/(2^4-1)$ - d=6, or d=0110 • Resolution = $V_{max}/(2^n-1)$ <ul style="list-style-type: none"> - $7.5/(2^4-1) = 0.5$ |
|--|--|---|

proportionality

37

CS122A: Embedded System Design, Fall02

Converter design

- Digital-to-analog (D2A): fairly simple design
 - One input for each bit (n) in digital encoding (d)
 - One input for V_{max}
 - One output for analog signal (e)
 - Straightforward circuit involving resistors and op-amp
- Analog-to-digital (A2D)
 - How to go from volts to binary encoding?
 - Must 'guess' digital encoding (d)
 - Put 'guess' through a D2A and compare with original analog signal (e)
 - Use analog voltage comparator
 - E.g. Start with 0001, D2A return 0.5V, is 0.5V>input voltage?
 - No, try 0010, D2A returns 1V, is 1V>input voltage?
 - No, try 0011, D2A returns 1V, is 1V>input voltage?
 - No, try 0100, D2A returns 1V, is 1V>input voltage?
 - No, try 0101, D2A returns 1V, is 1V>input voltage?

38

CS122A: Embedded System Design, Fall02

Analog-to-digital conversion using successive approximation

- In another word:
 - Given an analog signal ranges from 0 to 15 volts
 - Given an 8-bit digital encoding
 - If the analog signal is at 5 volt
 - $5/15 = d/2^8 - 1$
 - d=85
 - Encoding: 01010101
- What if we don't know it is 5 volts, but still want an encoding?
 - Use "naïve" approximation
 - Use successive approximation

39

CS122A: Embedded System Design, Fall02

Converter design

- Successive approximation (conceptually)
 1. Start with $V_{min} = 0$ and V_{max} = maximum voltage and Current_bit=n
 2. Let $V_{candidate} = \frac{1}{2}(V_{min} + V_{max})$
 3. If $V_{candidate} > V_{original}$
 - $V_{max} = V_{candidate}$
 - Current_bit of the final encoding is 0, Current_bit--
 - If $V_{candidate} < V_{original}$
 - $V_{min} = V_{candidate}$
 - Current_bit of the final encoding is 1, Current_bit--
 - If $V_{candidate} = V_{original}$ or Current_bit == -1, DONE
 - else go back to 2.
- Actually, V_{min} and V_{max} are represented by encoding
- Add encoding together, and shift to right by 1
- Feed result through voltage comparator

40

CS122A: Embedded System Design, Fall02

Analog-to-digital conversion using successive approximation

$V_{min} = 15$ volts, $V_{max} = 0$ volt

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

7.5 > 5

$V_{min} = 7.5$ volts

0

```

11111111+00000000=11111111
11111111>>1=01111111
01111111 DAC= 7.5volts
7.5v>5v
V_min_encoding=01111111
    
```

41

CS122A: Embedded System Design, Fall02

Analog-to-digital conversion using successive approximation

$V_{min} = 15$ volts, $V_{max} = 0$ volt

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

7.5 > 5

$V_{min} = 7.5$ volts

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

3.75 < 5

$V_{min} = 3.75$ volts.

0

1

```

01111111+00000000=01111111
01111111>>1=00111111
00111111 DAC= 3.75volts
3.75<5
V_min_encoding=00111111
    
```

42

CS122A: Embedded System Design, Fall02

Analog-to-digital conversion using successive approximation

$V_{max}=15$ volts, $V_{min}=0$ volt

$\frac{1}{2}(V_{max} + V_{min}) = 7.5$ volts
 $7.5 > 5$
 $V_{min} = 7.5$ volts

$\frac{1}{2}(7.5 + 0) = 3.75$ volts
 $3.75 < 5$
 $V_{min} = 3.75$ volts.

$\frac{1}{2}(7.5 + 3.75) = 5.63$ volts
 $5.63 > 5$
 $V_{min} = 5.63$ volts

01111111+00111111=10111110
 10111110>=1=01011111
 01011111 DAC= 5.63 volts
 5.63 volts <5
 V_{min} encoding=01011111

43

CS122A: Embedded System Design, Fall02

Analog-to-digital conversion using successive approximation

$V_{max}=15$ volts, $V_{min}=0$ volt

$\frac{1}{2}(V_{max} + V_{min}) = 7.5$ volts
 $7.5 > 5$
 $V_{min} = 7.5$ volts

$\frac{1}{2}(7.5 + 0) = 3.75$ volts
 $3.75 < 5$
 $V_{min} = 3.75$ volts.

$\frac{1}{2}(7.5 + 3.75) = 5.63$ volts
 $5.63 > 5$
 $V_{min} = 5.63$ volts

$\frac{1}{2}(5.63 + 3.75) = 4.69$ volts
 $4.69 < 5$
 $V_{min} = 4.69$ volts

44

CS122A: Embedded System Design, Fall02

Analog-to-digital conversion using successive approximation

$V_{max}=5.63$ volts, $V_{min}=4.69$ volt

$\frac{1}{2}(5.63 + 4.69) = 5.16$ volts
 $5.16 > 5$
 $V_{min} = 5.16$ volts.

$\frac{1}{2}(5.16 + 4.69) = 4.93$ volts
 $4.93 < 5$
 $V_{min} = 4.93$ volts.

$\frac{1}{2}(5.16 + 4.93) = 5.05$ volts
 $5.05 > 5$
 $V_{min} = 5.05$ volts.

$\frac{1}{2}(5.05 + 4.93) = 4.99$ volts
 $4.99 < 5$

45

CS122A: Embedded System Design, Fall02

Real-time clocks

- Keeps time and date in an embedded system
- Typically composed of:
 - crystal-controlled oscillator
 - generates very consistent high-frequency digital pulse
 - numerous cascaded counters
 - first counts up to oscillator frequency to output one second
 - next counter counts to 59 to output one minute
 - and so on for hour, date, month, and year
 - adjusted for leap years
 - battery backup
 - keeps it running when while power is off

46

CS122A: Embedded System Design, Fall02