ECE-3226/CSCI-3451 - Lab #7: Timers/Counters and Keypads

Contents:

Overview

Purpose: This lab will introduce you to creating delays using an 8-bit timer/counter. It also introduces the operation of the keypad, and how we can connect the STK500 to external (off-board) peripherals through the ports.

In particular, you will:

- learn how to set up and use Timer/Counter 0, and 8-bit timer/counter
- learn how to control and monitor the timer using the two registers, OCR0 and TCCR0
- learn how to connect a keypad to the STK500
- learn how to poll the rows of the keypad and monitor the column outputs to identify button presses
- gain experience in creating ASCII strings from key presses
- gain further practice using the STK500 hardware, switches, and LEDs

Pre-lab:

First week : Part1a , Part2a

Second week: Part3a

Lab Assignment

- **Part 1: Creating Delays using Timer/Counter 0**

  In Lab #5, you learned how to create delays using Delay Loops (simple loops that iterate a large number of times to generate the desired delay). Unfortunately, delay loops have a couple limitations. First, delay loops are not very accurate in many systems. In the AVR microcontroller they are reasonably accurate since the execution times for instructions is fixed, but in many processors, especially high-performance processors in desktops and laptops, the actual execution times for instructions can vary dramatically, especially for memory loads and stores. Second, when using delay loops, the processor cannot do any other useful work during the delay period.

  As an alternative to delay loops, in the first part of this lab you'll see how to create delays using timers/counters. In particular, we'll focus on Timer/Counter 0, an 8-bit timer/counter that is controlled and monitored using the I/O registers TCNT0 and TCCR0. This is similar to (i.e. an extension of) one of the programs you implemented in Lab #5.

  The TCCR0 register is used to configure when (how frequently) the timer/counter register is incremented, while the TCNT0 register holds the current count. The rate (frequency) at which the TCNT0 counter is incremented can be configured to one of 7 sources. The first 5 sources correspond to various fractions of the processor clock, including:

  - the frequency of the processor clock (CK)
  - one-eighth of the frequency of the processor clock (CK/8)
  - 1/64 of the frequency of the processor clock (CK/64)
  - 1/256 of the frequency of the processor clock (CK/256)
  - 1/1024 of the frequency of the processor clock (CK/1024)

  The other two sources correspond to the rising or falling edge of an externally-driven source.

  When configuring the TCCR0 register, we simply set the bottom 3 bits of TCCR0 to select the desired counter frequency. The other five bits of TCCR0, bits 3 through 7, will always be 0. For example, if the programmer selects a counter frequency of 1/1024 of the processor clock, then the counter will increment once every 1024 cycles. The following pair of instructions perform this selection:

```
ldi   r16, 0b00000101
out   TCCR0, r16
```

  The counting begins <u>immediately</u> once you configure the TCCR0 register. If you ever desire to stop counting, you can write the value 0x00 to TCCR0 (i.e. write

"000" to the bottom 3 bits of TCCR0). You can re-start the count at any time by re-configuring TCCR0 with the desired counter frequency.

The current count of the counter can be read at any time by reading the TCNT0 register using the `in rd, TCNT0` instruction (where *rd* is some register).

Since our delay time will rarely be a full count of the timer/counter (i.e. a full count from 0 to 255), we can create a "mark" to indicate the next value of the counter at which point the delay will be complete. For example, if our timer/counter is configured to count at 1/256 of the frequency of the processor, and the processor's clock frequency is 1 MHz (our default frequency for the ATmega32 on the STK500), our counter will increment once every 256 microseconds

a. **Program:** Use the program from lab #5 (part 2f) that uses all bits of PORTB as outputs and **rotates** (to the left), lighting one LED at a time once every half a second starting with LED0. In other words, LED0 will be lit in the first loop iteration, LED1 in the 2nd iteration, LED2 in the 3rd, and so on. After the last LED is lit, the loop will start over again. Instead of the delay loop use Timer0 to implement the half a second delay.

- **Part 2: Using Keypads**

In the second part of this lab, we'll see how we can connect and use keypads with AVR processor. The keypad you will be using in this lab is a 16-button keypad.

In this lab, you will be using port A to control the keypad, so you need to connect the 8 pins on the keypad to the 8 pins for port A. You may want to have the instructor verify your connection before you begin running the program.

Essentially, we are connecting the keypad such that the first four pins of the keypad connect to pins PA0 - PA3, while the last four pins of the keypad connect to pins PA4 - PA7. We are connecting the pins in this fashion because we will be using the first four pins (PA0 - PA3) as inputs and the last four pins (PA4 - PA7) as outputs. The last four pins on the keypad correspond to the rows of the keypad, while the first four pins correspond to the columns of the keypad. To monitor which key is being pressed at any given point, we need to send a low voltage (logic '0') to one (and only one) of the rows, and then monitor which, if any, of the columns has a low voltage. If none of them do, then no key is being pressed. If one of them has a low voltage, that indicates that the key at the corresponding column position in that row (the row with being sent the low voltage) is being pressed.
Following is a program that contains a procedure for monitoring keypresses: keypress.asm. They procedure returns two values, one value indicating whether a button was pressed, and one value indicating which button was pressed (if one was pressed). The second value indicates which button was pressed by return the

button number (0 to 15) in the keypad. Buttons 1, 2, 3, and A correspond to button numbers 0, 1, 2, and 3. Correspondingly, buttons 4, 5, 6, and B correspond to button numbers 4, 5, 6, and 7, and so on. The main program will display the button number of the last valid button press as a binary number on the LEDs.

Discussion-2-1: In your lab report, describe in your own words how the GetKeyPress program works. (The program given at the end of this handout.)

a. Program: Using the procedure from the original program, create a program that displays on the LEDs the ASCII code for a selected character. In this program, you will use **two** keypresses to select an alphanumeric character. The first keypress will be one of the 12 keys on the left side of the keypad ('0' - '9', '*', or '#'), while the second keypress will be one of the keys 'A' - 'D'. The second keypress will be used to indicate which character to use from the first keypress (when multiple characters are available). For example, if the first keypress was '8', this key also indicates the alphabetic characters of 'T', 'U', and 'V'. The second keypress will indicate which of the four options to use: the 'A' key will select the '8', the 'B' key will select the 'T', and so on. In cases where there is only one option (such as the '1' key), you may select to have either one or two keypresses (i.e. you can forego the second keypress in such cases, if you choose).

You can implement this program any way you like, but the easiest way to implement this program is to use a "look-up table". When using a look-up table, you are effectively performing a conversion between types by using the first type (in this case, the button number (0 to 15) and function number ('A' = 0, 'B' = 1, 'C' = 2, 'D' = 3)) to address into a look-up table that contains the equivalent second type (in this case, the corresponding ASCII code). If you choose to use a look-up table, one possible implementation would a look-up table like:

```
TextArray:
.DB     "11112abc3def", 0, 0, 0, 0
.DB     "4ghi5jkl6mno", 0, 0, 0, 0
.DB     "7prs8tuv9wxy", 0, 0, 0, 0
.DB     "    0000....", 0, 0, 0, 0
```

With this look-up table, you would compute the address as the address of

TextArray + 4 times the button number + the function number.

- **Part 3: Combining Keypads and Timers/Counters**

  This final part will have you create a program that combines your programs for Parts 1 and 2.

  . **a. Program:** Starting from your ASCII program in Part 2, create a new version of the program that will store the ASCII codes for characters in data memory. Then, once the entire message has been entered (the user should press '#' to indicate the end of the message), the full message will be redisplayed on the LEDs in a timed fashion. The user should be able to adjust the delay between successive ASCII codes from 0.5 seconds to 5.0 seconds, in 0.5 increments. Use the Timer subsystem to implement the delay. *(Note: Use INT0 for increasing and INT1 for decreasing the delay)*.

  In addition to enable the user to end the message by pressing the '#', the user should also be able to enter a space, ' ', a comma, ',', or a period, '.', using the '*' key in conjunction with function keys 'A', 'B', and 'C', respsectively (i.e. the two keypress combo of '*' followed by 'B' will generate a comma).

  For example, if the user enters a sequence of characters on the keypad for "dr. fritts", then these 10 characters should be stored in data memory as they are entered. Then, upon pressing the '#' key, the program should begin displaying the ASCII codes for these characters on the LEDs. The delay between subsequent characters should be user-selectable between 0.5 and 5.0 seconds (you select an appropriate default starting delay value).

```
;***********************************************
; written by:           Jason Fritts
; Modified by:          Armineh Khalili
; date:                 7/26/12
; file saved as: Keypad.asm
; for AVR:              STK500 with ATmega32
; clock frequency:      ? MHz
;***********************************************

; Program Function:  Gets keypresses from a keypad

;.device              ATmega32          ; don't need because it's in .inc file below
.nolist
.include        "C:\Program Files\Atmel\AVR
Tools\AvrAssembler2\Appnotes\m32def.inc"
.list

;=================
; Start of Program

        jmp     Init                    ; first line executed

;=========

Init:
        ; initialize Stack Pointer (SP)
        ldi     r16, LOW(RAMEND)
        out     SPL, r16

        ldi     r16, HIGH(RAMEND)
        out     SPH, r16

        ; initialize Port B
        ser     r16                     ; set all PortB pins to output
        out     DDRB,  r16              ;

        ; initialize PortA
        ldi     r16, 0b11110000         ; set PA4-PA7 as outputs, PA0-PA3 as inputs
        out     DDRA, r16

        ldi     r16, 0b11111111         ; enable pull-up registers on PA4-PA7
        out     PORTA, r16
```

```
;================
; Main body of program

Start:
        ser             r25             ; initialize LEDs to off
        out     PORTB , r25

Checkpress:
        call    GetKeyPress             ; check for a keypress

        mov     r24, r0                 ; if key not pressed, check again
        cpi     r24, 0
        breq    Checkpress

        mov     r25, r1                 ; if key pressed, update display
        com     r25

        out     PORTB, r25              ; displays button number (0 to 15)
                                        ;   of last valid keypress
 Release:
        call     GetKeyPress            ; check for  button release

        mov     r24, r0                 ; if key not released, check again
        cpi     r24, 0
        brne    Release

        rjmp    Checkpress              ; repeat indefinitely

End:
        rjmp    End                     ; end of program


;================

; Procedures


;Procedure:  GetKeyPress
;   Function:  checks whether a key has been pressed, and if so, which button
;   Inputs:
;                       none
;       Outputs:
;                       r0  <-  1 if button pressed (0 otherwise)
;                       r1  <-  button number pressed  (0 to 15)
GetKeyPress:
        push    r16                             ; save registers
```

```
        push    r18
        push    r19
        push    r20
        push    r22

        clr     r0
        clr     r1
        ldi     r16, 0b11101111         ; clear output pins PA4-PA7
        clr     r20                     ; initial button number

KeyLoop:
        out     PORTA, r16

        ; short delay while changes to row voltages propagate through keypad
        ldi     r22, 100
Delay:
        dec     r22
        brne    Delay

        ; read status of buttons from keypad
        in      r18, PINA               ; read values from PA0-PA3
        com     r18
        andi    r18, 0x0F               ; check if any buttons pressed
        breq    RowNotLit

        ; if a button was pressed, compute the button number (0 to 15)
        sbrc    r18, 1                  ; if 2nd button in row pressed
        subi    r20, -1                 ; then add 1 to button number for 2nd button in row

        sbrc    r18, 2                  ; if 3rd button in row pressed
        subi    r20, -2                 ; then add 2 to button number for 3rd button in row

        sbrc    r18, 3                  ; if 4th button in row pressed
        subi    r20, -3                 ;  then add 3 to button number for 4th button in row

        ldi     r19, 0x01
        mov     r0, r19                 ; indicate a button was pressed (set r0 to 1)
        mov     r1, r20                 ; copy button number pressed into r1
        rjmp    ExitProc

RowNotLit:
        ; if button not pressed, check next row in keypad
        lsl     r16
        ori     r16, 1
        subi    r20, -4                 ; update button number for next row
                                        ; if all rows checked, exit procedure
```

```
        cpi     r16, 0b11111111
        brne    KeyLoop

ExitProc:
        pop     r22             ; restore registers
        pop     r20
        pop     r19
        pop     r18
        pop     r16
        ret
```

Bottom View