

Report

Laboratory 1



September 13, 2017

Author:

Caroline Nilsson (cn222nd)

Daniel Alm Grundström (dg222dw)

Term: HT 2017

Course: 1DT301 - Computer

Technology I

Contents

1	Introduction		1	
2		gnment 1 - Light LED2	2	
	2.1	Pseudo code	2	
	2.2	Flowchart	2	
	2.3	Method	2	
	2.4	Assembly Program	3	
3	Assignment 2 - Switch light corresponding LED			
	3.1	Pseudo code	4	
	3.2	Flowchart	4	
	3.3	Method	4	
	3.4	Assembly Program	5	
4	Assignment 3 - Switch 5 lights LED0			
	4.1	Pseudo code	6	
	4.2	Flowchart	6	
	4.3	Method	7	
	4.4	Assembly Program	8	
5	Assignment 4 - Using the AVR simulator			
	5.1	Method	9	
6	Assignment 5 - Waterfall			
	6.1	Assignment description	10	
	6.2	Pseudo code	10	
	6.3		10	
	6.4		11	
	6.5		12	
7	Assignment 6 - Johnson counter			
	7.1	8	13	
	7.2		14	
	7.3		14	
	7.4		15	

1 Introduction

In the process of working with the laboratory assignments we started by doing research about the AVR Assembly language and the STK600 in order to better understand how to solve the different assignments. In each assignment we first created a pseudocode solution which we converted to flowchart diagrams, then it was rather simple to convert this into Assembly code. Common for all assignments is also that we have been using the simulations to confirm that the program is working and completing the correct tasks.

When we tested the Assembly programs on hardware we had trouble with the LEDs outputting inversed values. For example the program for assignment 1, which is supposed light LED2, lit all LEDs except LED2. If we understood this correctly, this was due to the pull-up resistor being activated on PORTB which made the LEDs light when the corresponding bit on PORTB was 0 (as opposed to 1). We fixed this by changing the bit string values we wrote to the LEDs in each assignment.

Another small issue we encountered was that we had assumed that PORTD would be the port connected to the switches on the STK600, as this is the configuration described in the STK600 User Guide. In the hardware we tested on though, the switches were connected to PORTC. To fix this we simply replaced all references of PORTD, PIND and DDRD to PORTC, PINC and DDRC in our Assembly code.

2 Assignment 1 - Light LED2

In the first assignment we were to write an Assembly program that lights up LED2 (which is the third light counting from the right).

2.1 Pseudo code

Algorithm 1 Light LED2 procedure PSEUDOCODE PortB = output $Led2 \ bitstring \rightarrow PortB$

2.2 Flowchart

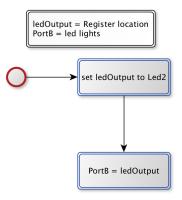


Figure 1: Basic flow in order to light LED2

2.3 Method

The pseudocode (see algorithm 1) and the flowchart (see figure 1) shows that we first set PORTB as an output port. To light up LED2 we then only need to write a 0 to the bit on PORTB that corresponds to LED2.

The minimal number of lines required to write this program we think are four. Two lines are required to set the LED port as output: one to write a value to a register and the other to write that value to the data direction register. The final two lines are for turning on the LED: one to write the LED state to a register and the other to write the LED state to the output port. One could try to write the program in three lines, by reusing the value written to the data direction register when writing to the output port. But in this case the LED will not turn on because of the pull-up resistor which will require that a zero is written to the bit corresponding to the LED that we want to light.

```
Caroline Nilsson
Daniel Alm Grundström
                                                                            (cn222nd)
                                                                          (dg222dw)
           Lab number:
                                      How to use the PORTs. Digital input /output. Subroutine call.
            Title:
STK600, CPU ATmega2560
           Hardware:
            Function:
                                      Lights LED2 on PORTB
           Input ports:
                                      N/A
           Output ports:
                                      PIN2 on PORTB
                                      N/A
m2560def.inc
            Included files:
           Other information: LEDs are configured to light when PINs on PORTB are set to 0. The default state, when no LED is lit must therefore be set to 0b1111_1111.
           Changes in program:
                                      2017-09-01:
Implemented flowchart design.
                                      2017-09-02:
                                      Added comments and .def for r16
                                      Adjusts code to handle pull-up resistors on PORTB.
Removes unnecessary loop that prevented program from exiting after LED2 had been turned on.
      ;<<<<<<<><
      .include "m2560def.inc"
.def ledOutput = r16
      ; Set PORTB to output ldi ledOutput, 0xFF out DDRB, ledOutput
      ; Turn on LED2 on PORTB

| di | ledOutput , 0b1111_1011
| out | PORTB , ledOutput
```

3 Assignment 2 - Switch light corresponding LED

In the second assignment we were to write a program that waits for a switch to be pressed and then lights up the corresponding LED. For example if switch 3 is pressed LED 3 should light up. The way we interpreted the assignment was that the LED should stay on for as long as the switch is pressed down and turn off when the switch is released.

3.1 Pseudo code

```
Algorithm 2 Switches pressed lights corresponding LEDprocedure PSEUDOCODEPortB = outputPortC = inputrepeatPortC value \rightarrow switchState\triangleright switchState = register locationswitchState \rightarrow PortBuntil \infty
```

3.2 Flowchart

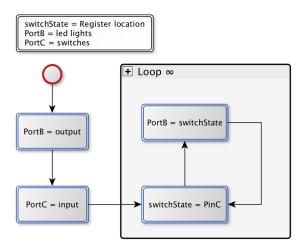


Figure 2: Basic flow in order to read switches and light corresponding LED

3.3 Method

We figured the simplest way to solve this problem was to redirect the input from the switches to the LEDs repeateadly in a loop. We start by setting PORTB as output and PORTC as input. Then, in an infinite loop, we read the value of the pins on PORTC to a register and output the value of that register to PORTB.

```
Author:
                                                                                                                                                       Caroline Nilsson
Daniel Alm Grundström
                                                                                                                                                                                                                                                                                                           (cn222nd)
                                                                                                                                                                                                                                                                                                      (dg222dw)
                                             Lab number:
                                                                                                                                                      How to use the PORTs. Digital input /output. Subroutine call.
                                              Title:
  10
STK600, CPU ATmega2560
                                             Hardware:
                                                                                                                                                      Reads input from the switches SW0..SW7 and lights the corresponding LED when a switch is pressed. (SW0 lights LED0, SW1 lights LED1 and so on)  \frac{1}{2} \frac{1}
                                              Function:
                                            Input ports:
                                             Output ports:
                                                                                                                                                      PORTB
                                                                                                                                                      m2560def.inc
                                             Included files:
                                              Other information: N/A
                                             Changes in program:
                                                                                                                                                      2017-09-01:
Implemented flowchart design.
                                                                                                                                                       2017-09-02:
                                                                                                                                                       Adds header and comments.
                                                                                                                                                       Adjusts code to handle pull up resistor on PORTB. Changes switch port to PORTC.
                      ; Set PORTB (LEDs) as output
                       ldi dataDir, 0xFF
out DDRB, dataDir
                       ; Set PORTC (switches) as input ldi dataDir, 0x00 out DDRC, dataDir
                       loop:
    in switchInput, PINC
    out PORTB, switchInput
                                                                                                                                                                                               ; Read input from switches
; Output switch input to LEDs
                                              rjmp loop
```

4 Assignment 3 - Switch 5 lights LED0

In the third assignment we were to write an Assembly program that turns on LED0 when the switch SW5 is pressed. Nothing should happen when the other switches are pressed. We assumed that the way the program should work is that the LED would stay lit for as long as SW5 was pressed down and turn off when the switch was released.

4.1 Pseudo code

4.2 Flowchart

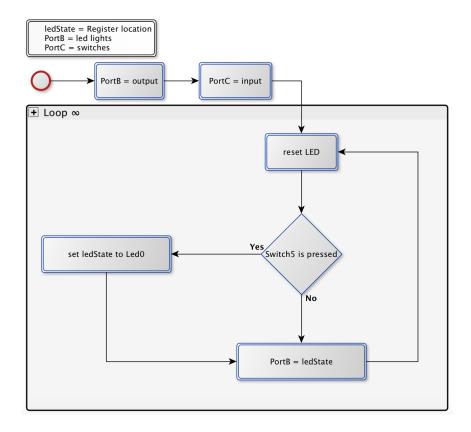


Figure 3: Basic flow in order to light LED 0 when switch 5 is pressed

4.3 Method

As with the previous assignment, we figured the first thing that needed to be done was to set the LED port, PORTB, as output and the switch port, PORTC, as input. In a loop, we then reset the value to write to the leds before checking if SW5 is pressed down. If the switch is pressed down we clear the least significant bit in the LED output value to indicate that we want LED0 to turn on before writing the value out to the LEDs.

```
;>>>>>
          1DT301, Computer Technology I
Date: 2017-09-07
Author:
                                  Caroline Nilsson
Daniel Alm Grundström
                                                                   (cn222nd)
                                                              (dg222dw)
          Lab number:
                                 How to use the PORTs. Digital input /output. Subroutine call.
          Title:
10
STK600, CPU ATmega2560
          Hardware:
          Function:
                                  Turns on LEDO when SW5 is held down.
                                 PORTC
          Input ports:
          Output ports:
                                 PORTB
                                 N/A
m2560def.inc
          Included files:
          Other information: N/A
          Changes in program:
                                  2017-09-01:
                                  Implemented flowchart design.
                                  2017 - 09 - 04:
                                  Minor refactoring. Adds header and comments.
                                  Adjusts code to handle pull up resistor on PORTB. Changes switch port to PORTC.
     , ; <<<<<<<<<<<<<<<i>include "m2560def.inc"
     . def dataDir = r16
. def ledState = r17
     ; Set PortB as output ldi dataDir, 0xFF out DDRB, dataDir
      ; Set PortC as input
     ldi dataDir, 0x00
out DDRC, dataDir
     loop:
ser ledState
                                               ; Set bits in LED state so LEDs are turned ; off when button is released
                                               ; If SW5 is pressed down (PINC5 bit is zero) ; then set LED0 state to turned on
          sbis PINC, PINC5
              ldi ledState, 0xFE
         out PORTB, ledState rjmp loop
                                               ; write state to LEDs
```

5 Assignment 4 - Using the AVR simulator

For this assignment, we were to take the program we wrote for the previous assignment, that lights LEDO when SW5 is pressed down, and run it in the AVR simulator that is included in Atmel Studio.

5.1 Method

First, we needed to setup the debugger to run the program on the simulator which is done by selecting "Simulator" in $Project \rightarrow Properties \rightarrow Tool$. To start stepping through the program we then clicked $Debug \rightarrow Start\ debugging\ and\ break$. To be able to see what happens in the simulator as the program runs, we opened the $Processor\ Status$ and I/O windows by clicking on their respective icons. Finally, because values from the pins on the switches have a default value of 1 we needed to set all PINC bits, which we did by selecting $I/O\ Port\ (PORTC)$ in the $I/O\ window$ and filling in the bits to the right of PINC by clicking on them.

We then started stepping through the program. The first few lines sets PORTB as output (line 42-43) and PORTC to input (line 46-47) and this can be seen in the simulator by keeping an eye on $PORTB \rightarrow DDRB$, which bits are set to 1, and $PORTC \rightarrow DDRC$, which are set to 0, as the debugger executes the instructions. When the debugger executes line 50, which sets all bits in register 17, we can see this in the Processor Status window in that r17 gets set to $0 \times FF$. We tested that the instruction on line 53, which checks if SW5 is pressed down, works by manually clearing bit 5 on PINC through the I/O window. The instruction that gets executed if this is true, can be seen in the simulator in that register 17 changes value from $0 \times FF$ to $0 \times FE$. The result of line 56, which writes the value of register 17 to PORTB, can be seen by clicking on PORTB in the I/O window where the bits next to PORTB and PINB are updated accordingly. We could also see that the *Program Counter* value in the Processor Status window gets updated when the final line real1 loop gets executed.

6 Assignment 5 - Waterfall

6.1 Assignment description

The fifth assignment was to write an Assembly program that outputs a ring counter to the LEDs. Between each value in the counter, there should be a delay of approximately 0.5 seconds. Since the delay should be a subroutine in the program, the stack pointer SP needs to be initialized as instructed in the description of the assignment.

6.2 Pseudo code

```
Algorithm 4 Display Ring counter on LEDsprocedure PSEUDOCODEInitialize stack pointerPortB = output\triangleright ledState = register locationInitialize ledState\triangleright ledState = register locationrepeatledState \rightarrow PortBDelayrotate ledState to leftuntil \infty
```

6.3 Flowchart

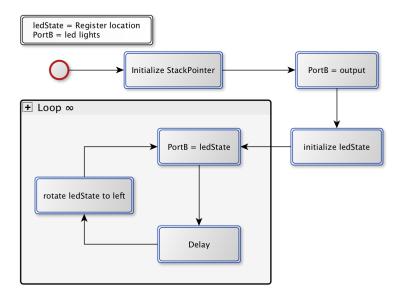


Figure 4: Basic flow in order to display a Ring counter on the LEDs

6.4 Method

The first thing we needed to do, as described above, was to initialize the stack pointer. This is done by setting SP to the end of SRAM (RAMEND). Since SP is a 16-bit register we needed to set both SPL and SPH. SPL is set to the least significant 8 bits of RAMEND and SPH is set to the most significant 8 bits of RAMEND. As always, we also set PORTB as output so we can write values to the LEDs.

The main part of the program consists of a loop where the current LED state is first written to the LEDs. We then delay execution of the program for ~ 0.5 seconds and finally rotate the bits in the LED state to the left using the rol instruction.

The delay functionality is as described in the assignment description implemented in a subroutine which we have calculated using the *AVR Delay Loop Calculator*.¹ We calculated a delay of 500 ms for 1.0 MHz, which is the default clock speed of the AVR ATmega2560.² We modified the subroutine slightly to push the registers that are used in it to the stack at the start of the subroutine and pop them before returning. This is done so we do not accidentally overwrite any values in the registers that the subroutine is using. These additional instructions do of course make the delay slightly longer, particularly since the CPU will need to write to and read from SRAM, but we did not think that mattered that much in this case.

http://www.bretmulvey.com/avrdelay.html

² Atmel ATmega640/V-1280/V-1281/V-2560/V-2561/V DATASHEET, Atmel Corporation, San Jose, CA, 2014, pp. 40

```
·>>>>
           1DT301, Computer Technology I
Date: 2017-09-07
           Author:
                                      Caroline Nilsson
                                                                           (cn222nd)
                                      Daniel Alm Grundström
                                                                         (dg222dw)
 6
7
           Lab number:
           Title:
                                     How to use the PORTs. Digital input /output.
10
                                      Subroutine call.
           Hardware:
                                      STK600, CPU ATmega2560
Repeatedly lights LEDs sequentially right to left.
           Function:
                                      100 0000 -> 0000 0010 -> 0000 0100 -> ... -> 1000 0000 -> 0000 0001 -> 0000 0010 -> ...
           Input ports:
           Output ports:
                                     PORTB
                                     delay - delays execution m2560def.inc
           Included files:
           Other information: Since a subroutine is used, the stack pointer must be initialized so the processor knows where in the code to jump when the subroutine returns.
           Changes in program:
                                      2017-09-01:
                                      Implements flowchart design
                                      2017 - 09 - 04:
                                      Adds header, comments and some minor refactoring
                                      2017-09-07:
Adjusts code to handle pull up resistor on PORTB.
     .include "m2560def.inc"
      . the dataDir = r16
.def dataDir = r16
.def ledState = r17
.equ INITIAL_LED_STATE = 0xFF
; Initialize SP, Stack Pointer Idi r20, HIGH(RAMEND) out SPH, R20 low (RAMEND) out SPL, R20
                                                           ; R20 = high part of RAMEND address
                                                           ; SPH = high part of RAMEND address; R20 = low part of RAMEND address; SPL = low part of RAMEND address
        Set PORTB to output
      ldi dataDir, 0xFF
out DDRB, dataDir
      ldi ledState, INITIAL_LED_STATE
                                                          ; Set initial LED state
      loop:
           out PORTB, ledState
rcall delay
rol ledState
                                                           ; Write state to LEDs
                                                           ; Delay to make changes visible
; Rotate LED state to the left
           rjmp loop
      ; Generated by delay loop calculator
      ; at http://www.bretmulvey.com/avrdelay.html
      ; Delay 500 000 cycles
        500ms at 1 MHz
      delay:
          push r18
push r19
           push r20
     ldi r18, 3
ldi r19, 138
ldi r20, 86
L1: dec r20
           brne L1
           dec r19
brne L1
dec r18
brne L1
           rjmp PC+1
           pop r20
```

7 Assignment 6 - Johnson counter

The final assignment was to write a program that displays a *Johnson counter* to the LEDs in an infinite loop. A Johnson counter is a counter that sets all bits in a bit string one-by-one and, when all bits are set, clears the bits one-by-one. Of course, this program will also need a delay after each value in the Johnson counter has been written to the LEDs. Otherwise the program would run so fast that all LEDs would most likely appear to be lit all the time.

7.1 Pseudo code

Algorithm 5 Johnson counter simulation using LEDs

```
procedure PSEUDOCODE
    PortB = output
                                                    \triangleright complement = register\ location
    Initialize currentValue
                                                  \triangleright currentValue = register\ location
   repeat
                                                                    \triangleright Loop\_1 (count up)
       if LED7 is lit then
            Continue at Loop_2
       else
           currentValue = currentValue \times 2
           Increase currentValue by 1
           complement = complement \ of \ current Value
        complement \rightarrow PortB
        Delay
    until \infty
    repeat
                                                                 \triangleright Loop_2 (count \ down)
       if LED0 is lit then
            Continue at Loop_1
       else
           currentValue = Shift\ right
           complement = complement \ of \ current Value
        complement \rightarrow PortB
        Delay
    until \infty
```

7.2 Flowchart

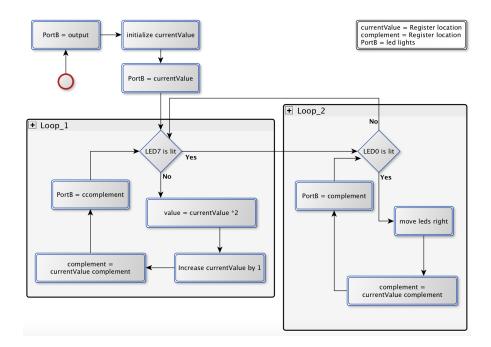


Figure 5: Basic flow in order to display a Johnson counter on LEDs

7.3 Method

We started by trying to figure out the mathematical formula for finding Johnson values. In decimal, the values for an 8-bit Johnson counter are: 0, 1, 3, 7, 15, 31, 63, 127 and 255. We realized that we could get the next value by multiplying by 2 and adding 1, which gives the recurrence relation

$$J_n = J_{n-1} \cdot 2 + 1, \qquad n \in \mathbb{N}_0$$

With this we could get both the next and previous Johnson value. In the case of getting the previous value, we considered that we are dealing with integers, which because of truncating means we only needed to divide the current value by 2 to count down the counter.

As with assignment 5, we start by setting the stack pointer since we are going to use a subroutine for the delay function. We also set PORTB as output. To count the Johnson counter up and down, we created two loops: count_up and count_down. In count_up we multiply the current counter value by 2 by shifting it to the left and then increment it by 1 before outputting the value to the LEDs. We repeat this until all the LEDs are lit, upon which executions jumps to count_down. In this loop, we divide the current counter value by 2 by shifting it to the right and then output the value to the LEDs. This is in turn repeated until all the LEDs are turned off, upon which the execution jumps back to count_up and the process starts again.

```
·>>>>
            1DT301, Computer Technology I
Date: 2017-09-07
            Author:
                                        Caroline Nilsson
                                                                               (cn222nd)
                                        Daniel Alm Grundström (dg222dw)
 6
7
            Lab number:
                                       How to use the PORTs. Digital input /output.
            Title:
10
                                        Subroutine call.
            Hardware:
                                        STK600, CPU ATmega2560
                                        Lights LEDs as a Johnson counter in an infinite loop.
            Function:
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
                                        0000 0001 -> 0000 0011 -> 0000 0111 -> ...
1111 1111 -> 0111 1111 -> 0011 1111 -> ...
           Input ports:
                                       PORTB
            Output ports:
            Subroutines:
                                       delay - delay execution
            Included files:
                                        m2560def.inc
            Other information: N/A
            Changes in program:
                                        2017-09-02
                                        Implements flowchart design
                                        2017-09-04:
                                        Adds header and comments
                                        Adjusts code to handle pull up resistor on PORTB.
Changes code to use shift left instead of multiplying
      ;<<<<<<<>.include "m2560def.inc"
      . def dataDir = r16
. def currentValue = r17
                                                              ; Current value of Johnson counter
      . def complement = r18
46
47
48
49
50
51
52
53
54
55
56
67
68
69
70
71
72
73
74
75
76
77
78
      ; Initialize SP, Stack Pointer
Idi r20, HIGH(RAMEND)
out SPH,R20
Idi R20, low(RAMEND)
out SPL,R20
                                                               ; R20 = high part of RAMEND address
                                                              ; SPH = high part of RAMEND address; R20 = low part of RAMEND address; SPL = low part of RAMEND address
        Set PORTB as output
      ldi dataDir, 0xFF
out DDRB, dataDir
        Set and output initial value
      ldi currentValue, 0x00
      rcall led_out
      count_up:
sbis PORTB, PINB7
                                                              ; If LED7 is lit (i.e. all LEDs lit)
                                                              ; then start counting down
                rjmp count_down
            ; Get next johnson value by multiplying by 2 and adding 1 {\color{red} \textbf{lsl}} currentValue {\color{red} \textbf{inc}} currentValue
            rcall led_out
rcall delay_500ms
                                                               ; Output complement of current value
                                                               ; Delay to make changes visible ; Continue counting up
            rjmp count_up
      count_down:
sbic PORTB, PINB0
rjmp count_up
                                                              ; If LEDO is unlit (i.e. all LEDs unlit) ; then start counting up
            lsr currentValue
                                                               ; Shift to right to get previous ; johnson value
            rcall led_out
                                                              ; Ouput complement of current value
            rcall delay_500ms
rjmp count_down
                                                               ; Delay to make changes visible
; Continue counting down
         Writes the complement of 'currentValue' to PORTB
      led_out:
            mov complement, currentValue
com complement
out PORTB, complement
89
90
91
92
93
94
95
96
97
98
99
      ; Generated by delay loop calculator ; at http://www.bretmulvey.com/avrdelay.html
         Delay 500 000 cycles
         500ms at 1 MHz
       delay:
           push r18
push r19
            push r20
```

```
| 100 | 101 | 1di | r18 | 3 | 102 | 1di | r19 | 138 | 1di | r20 | 86 | 104 | L1 | dec | r20 | brne | L1 | 108 | dec | r18 | 109 | brne | L1 | 110 | rjmp | PC+1 | 111 | 112 | pop | r20 | 113 | pop | r18 | ret | 115 | ret | 115 | ret | 116 | r18 | ret | 117 | r18 | ret | 118 | ret | 118 | r19 | r18 | ret | 119 | r18 | r18 | r19 | r19 | r19 | r18 | r19 |
```