# **ATMEL Studio 7 (main.asm)**

**Reference**: <http://darcy.rsgc.on.ca/ACES/TEI4M/Assembly/images/BCDontheADCEndianness.png>

Again, I’ve organized this code presentation into **4+** sections, hopefully making it clearer:
Page 1&2. **Directives**, Page 3. **Digit Separation**, Page 4&5. **PoV Display**,
Page 6. **Shiftout** followed by the delay code from the [Javascript utility](http://darcy.rsgc.on.ca/ACES/TEI4M/AVRdelay.html).

;PROJECT :BCDontheADCShield

;PURPOSE :4-digit BCD display on ADC Shield

;AUTHOR :C. D'Arcy

;DATE :2020 05 05

;DEVICE :Dolgin Development Platform

;MCU :ATtiny84

;COURSE :ICS4U

;STATUS :Working

.def util = r16 ;readability is enhanced through aliases

.def value = r17 ;holds the working value to be displayed

.def mask = r18 ;byte with one set bit to act as a mask

;.def dir = r19 ;holds shift direction: LSBFIRST:0, MSBFIRST:1

.def base = r19 ;

.def n = r20 ;holds the number being shifted out

.equ DDR = DDRA ;typically, we'll need the use of PortA

.equ PORT = PORTA ;both its DDR and output register and, eventually,

.equ thous = PA1 ;these are the port pins...

.equ hunds = PA2 ; connected to the base pins of

.equ tens = PA3 ; each of the transistors that

.equ units = PA4 ; ground the respective displays

.equ DATA = PA5 ;595 data pin

.equ LATCH = PA6 ;595 latch pin

.equ CLOCK = PA7 ;595 clock pin

;.equ LSBFIRST= 0 ;same familiar constants from Arduino days

.equ MSBFIRST= 1 ;ditto

.dseg

 digits: .byte 4 ;reserve SRAM space for the 4 individual digits

.cseg ;locate for Code Segment (FLASH)

; \*\*\*\*\* INTERRUPT VECTOR TABLE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.org 0x0000 ;start of Interrupt Vector Table (IVT) aka. Jump Table

 rjmp reset ;lowest interrupt address == highest priority!

.org 0x0020 ;position segment LUT beyond the IVT

segStart: ;MSBFIRST: ABCDEFGx

.db 0b11111100,0b01100000,0b11011010,0b11110010 ;0-3

.db 0b01100110,0b10110110,0b10111110,0b11100000 ;4-7

.db 0b11111110,0b11110110,0b11101110,0b00111110 ;8-b

.db 0b10011100,0b01111010,0b10011110,0b10110110 ;C-F (S)

segEnd:

.org 0x0030

 ;1234(BCD) defined MANY ways (or so you may think!) Wait, what?

;define as Bytes..

number:

.db 0b00010010, 0b00110100 ;Base 2 (binary)

.db 0b0001\_0010, 0b0011\_0100 ;Base 2 (binary) (clearer)

.db 022,064 ;Base 8 (octal)

.db 18,52 ;Base 10 (decimal)

.db 0x12,0x34 ;Base 16 (hexadecimal)

.db 0xAC,0xEF ;Base 16 (hexadecimal)

;define as Words (2 bytes)

.dw 0b0001001000110100 ;Base 2 (binary)

.dw 0b0001\_0010\_0011\_0100 ;Base 2 (binary) (clearer)

.dw 011064 ;Base 8 (octal)

.dw 4660 ;Base 10 (decimal)

.dw 0x1234 ;Base 16 (hexadecimal)

; \*\*\*\*\* START OF CODE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.org 0x0100 ;safe standard address for start of executable code

reset: ;PC jumps to here (start of code) on reset interrupt...

 ldi XL,low(digits) ;point to the 4 reserved bytes in SRAM

 ldi XH,high(digits) ;

 ldi ZL,low(number<<1) ;point to the number in Flash

 ldi ZH,high(number<<1) ;

 ldi n,2 ;loop twice; 2 nibbles separated in each iteration

repeat:

 lpm util,Z+ ;get the 'first' byte (2 nibbles) of the number

 swap util ;swap the two nibbles in the first byte

 mov value,util ;obtain a copy as code is 'destructive'

 andi value,0x0F ;mask off the high nibble

 st X+,value ;store the digit in SRAM

 swap util ;restore original number

 mov value,util ;again, obtain a copy as code is 'destructive'

 andi value,0x0F ;mask off the high nibble

 st X+,value ;store the digit in SRAM

 dec n ;reduce the loop control variable by 1

 brne repeat ;if we're not done, do one more time

;OK, the four digits have been separated and are sitting in SRAM

;Now, we need get each digit from SRAM and present them in cyclical order,

; adding each digit to the base address of the segment LUT so it points

; to the correct segment map to be shifted out.

;We also need to synchronize the clearing and setting of the respective

; base pins to see the digits avoiding ghosting

 ser util ;might as well set all port bits for output

 out DDR,util ;do it

 ldi YL,low(segStart<<1) ;Y is set to (permanently) point to the starting

 ldi YH,high(segStart<<1) ; address of our segment LUT in Flash

PoV: ;loop continuously…

 ldi XL,low(digits) ;X is set to point to the starting address of

 ldi XH,high(digits) ;the 4 previously separated digits in SRAM

 ldi base,1<<units ;start loop correctly: assume units have been done

 sbi PORT,units ;this requires setting the units' base pin in the PORT

;We're going to display the digits in the order thousands>hundreds>tens>units, so...

next:

 movw Z,Y ;Assign Y to Z; the stsrt of the segment LUT

 ld util,X+ ;get the digit to be displayed from SRAM

 add ZL,util ;add util to Z to obtain the index into the segment LUT

 lpm n,Z ;get segment map corresponding to digit to be shifted

;------New way to toggle a bit (OFF in this case)in a PORT------------------------------

 in util,PORT ;read the PORT

 eor util,base ;toggle previous base pin off

 out PORT,util ;write the PORT

;------End of new way to toggle a bit in a PORT ----------------------------------------

 rcall shiftout ;shift out the segment map (it's in n!)

 sbrc base,units ;have we already reached the units digit?

 ldi base,1<<PA0 ;if so, we need to prepare for the thous digit (PA1)

 lsl base ;shift the base pin to turn on the next digits

;------New way to toggle a bit (ON in this case)in a PORT-------------------------------

 in util,PORT ;read the PORT

 eor util,base ;toggle previous base pin off

 out PORT,util ;write the PORT

;-------End of new way to toggle a bit in a PORT ---------------------------------------

 rcall delay5ms ;reduce timeout to reduce flicker

 cpi base,1<<units ;have we displayed all 4 digits?

 brne next ; No? OK, go to the next one

 rjmp PoV ;Yes, start the cycle of 4 over again

;\*\*\*\*\*\*\*\*\*\*\*\*SHIFTOUT \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

shiftout: ;shifts constant n into the 595

 ldi mask,0x80 ;assume order is MSBFIRST

; sbrs dir,0 ;if bit 0 is set, it's MSBFIRST

; ldi mask,0x01 ;OK, it's LSBFIRST so redefine the mask

 cbi PORT,LATCH ;pull LATCH pin LOW

again:

 cbi PORT,CLOCK ;pull CLOCK pin LOW

 mov value,n ;reload the value to be presented

 and value,mask ;mask off the target bit

 breq lo ;was it 0?

 sbi PORT,DATA ;no, so pull DATA pin HIGH

 rjmp clockit ;ready to clock the 1

lo:

 cbi PORT,DATA ;else, it was a 0, so pull DATA pin LOW

clockit:

 sbi PORT,CLOCK ;pull CLOCK pin HIGH

; hmmm, must decide what direction to shift the mask...

; sbrs dir,0 ;if bit 0 is set, it's MSBFIRST

; rjmp shiftLeft ;OK, it's LSBFIRST

 lsr mask ;MSBFIRST so shift the mask right

 brne again ;repeat if there are still more bits to stuff in

 rjmp done ;we're done, so only one more thing to do

shiftLeft:

 lsl mask ;LSBFIRST, so shift the mask left

 brne again ;repeat if there are still more bits to stuff in

done:

 sbi PORT,LATCH ;pull LATCH pin HIGH to present 595's internal latches on output pins

 ret ;finished, return.

delay5ms:

; Assembly code auto-generated

; by utility from Bret Mulvey

; Delay 40 000 cycles

; 5ms at 8.0 MHz

 ldi r21, 52

 ldi r22, 242

L1: dec r22

 brne L1

 dec r21

 brne L1

 nop

 ret