Embedded Systems Lab Experiment 2

Digital Output through 7 segment LED with SRAM memory mapping

Program an ATMega 32 to control a 7-segment display

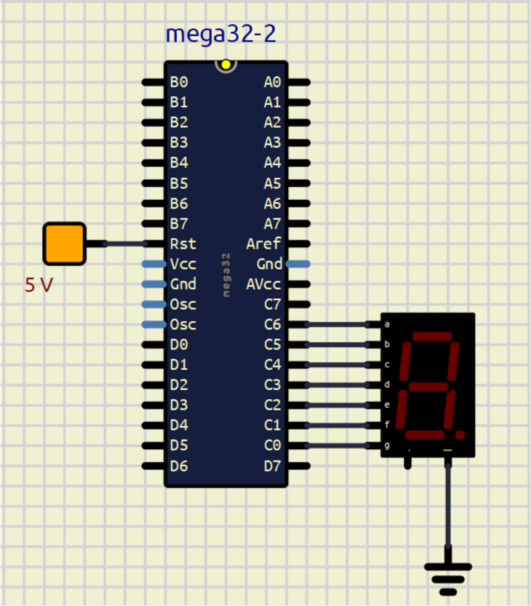
Jitbitan Baroi
1-11-2024

DATE	Exp2: Digital output through 7seg LED to incr, decr with SRAM memory mapping	SHEET NO. 1

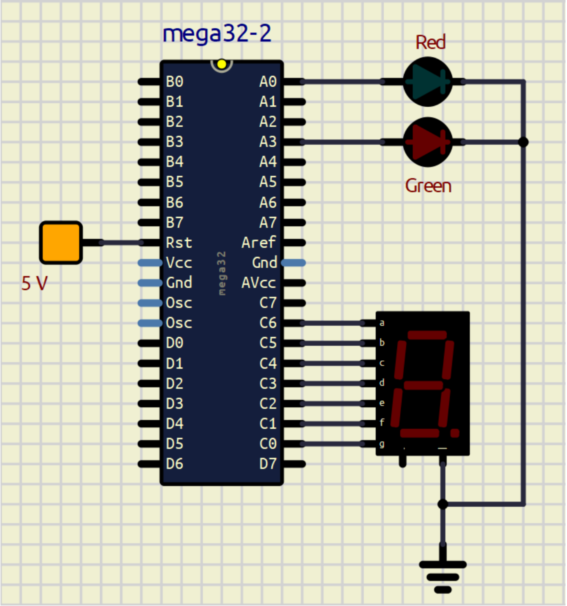
•) Aim: To program ATMega 32 to control a 7segment LED display.

•) Components reqd.:

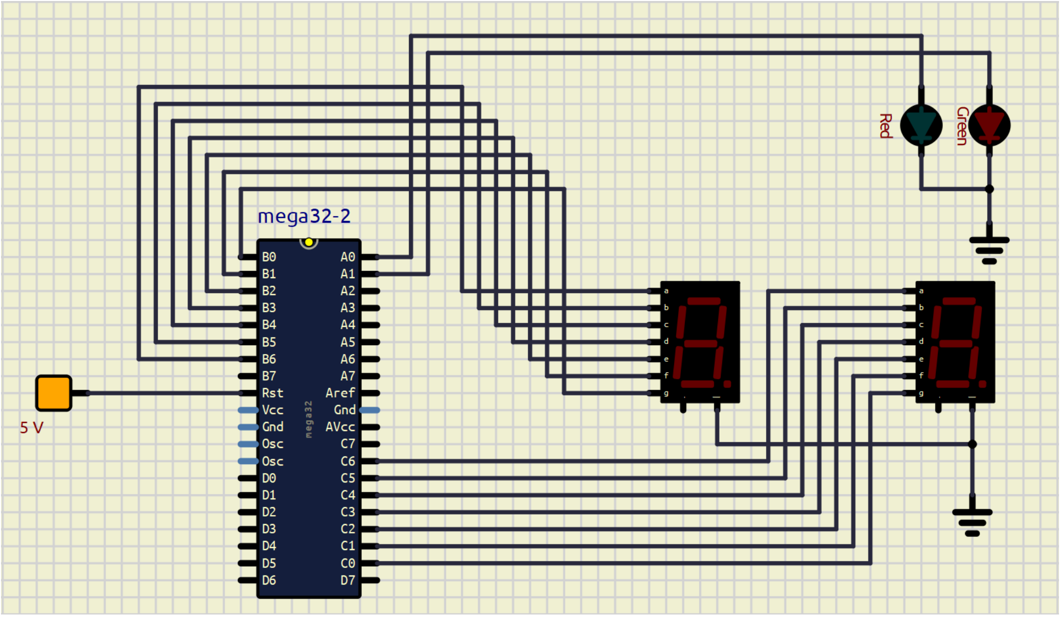
Name.	Specification	Quantity
1. ATmega 32 μC	—	1.
2. AVR dev board.	—	1.
3. USBasp.	—	1.
4. 7-segment display.	common cathode.	2.
5. ~~100~~ Resistors	100Ω.	9.
6. LED.	green, red.	1, 1.
8. Breadboard	—	1.
9. Jumper wire	female-male.	10


mega32-2
B0 B1 B2 B3 B4 B5 B6 B7 Rst Vcc Gnd Osc Osc D0 D1 D2 D3 D4 D5 D6
A0 A1 A2 A3 A4 A5 A6 A7 Aref Gnd AVcc C7 C6 C5 C4 C3 C2 C1 C0 D7
mega32
5 V
a b c d e f g

Schematic for Assignment 1, 2
An up counter $0 \rightarrow 9$


mega32-2
Red
Green
B0 B1 B2 B3 B4 B5 B6 B7 Rst Vcc Gnd Osc Osc D0 D1 D2 D3 D4 D5 D6
A0 A1 A2 A3 A4 A5 A6 A7 Aref Gnd AVcc C7 C6 C5 C4 C3 C2 C1 C0 D7
mega32
5 V
a b c d e f g

Schematic for Assignment 3
Road Traffic Signaling
(a down counter $9 \rightarrow 0$*)*


Red
Green
mega32-2
B0 B1 B2 B3 B4 B5 B6 B7 Rst Vcc Gnd Osc Osc D0 D1 D2 D3 D4 D5 D6
A0 A1 A2 A3 A4 A5 A6 A7 Aref Gnd AVcc C7 C6 C5 C4 C3 C2 C1 C0 D7
mega32
5 V
a b c d e f g
a b c d e f g

Schematic for assignment 4
Road Traffic Signaling, down counter $30 \rightarrow 00$ *using 2 displays*

- **Assignment 1**: (C code for incrementing in display).

```
1.  #include <avr/io.h> ————————————→ including librariesnecessary
2.  #define F_CPU <1000000UL> ——————→ ~~defining~~ synch µC clk.
3.  #include <util/delay.h> ——————→ including delay lib
4   int main (void){ ———————————————→ init main fn.
5     int a[]={0x7E, 0x30, 0x6D, 0x79, ——→ a array consisting of 7
6               0x33, 0x5B, 0x5F, 0x70, 0x7B}; segment codes (a,b,c,d,e,f), two (
7     DDRC = 0xFF; ————————————————→ setting PORTC as output.
8     while (1) { ———————————————————→ an infinite loop.
9        for(int i=0; i<10; i++) ————→ a for loop with increasing i: (
10         PORTC = a[i], _delay_ms(1000); ——→ for every cycle i is indexed.
11    }                                  from a and sent to port C.
12 }
```

- **Assignment 2**: (C code for decrementing in display).

everything will be same as the previous one except

*9. for (int i = 9; i >= 0; ~~i++~~ i--).

needs to be updated.

•) Explanation (1&2):

In the main fn, an array a has been defined in such way that a[i] ~~is the~~ has the binary coded form to display i in 7-segment display. It has two hexadecimal digits - thus 8 binary digits. 7 least significant bits are used to encoded. i in the form. (a, b, c, d, e, f, g). as in a 7-segment display. A "for" loop is implemented such that a dummy variable "i" is incremented by 1 in each step (decremented in ②). and port C is assigned with a[i] (Port C is connected to the display) ~~C0, C1, ——~~ $\{C_6, C_5, \ldots, C_0\} \leftarrow \{a, b, c, d, e, f, g\}$. a delay is added after every updation for our perception

•) Assignment 3: (C code's for traffic system).

```
#include <avr/io.h>                       → including necessary libs.
#define F_cpu 1000000UL                   → synch. μC clk.
#include <util/delay.h>                   → include delay lib
int main (void) {                         → main fn.
   int a[] = {0x7F, 0x30, 0x6D, 0x79,     → an array that contain 7
              0x33, 0x5B, 0x5F, 0x70,     → segment display such that
              0x7F, 0x7B};                → 7segment i = a[i]
DDRC = 0xFF;  ——————→ setting C ports as out.
DDRA = 0xFF;  ——————→ setting A ports as out.
while (1){  ————————→ inf loop.
    PORTA = 0x01;  ——→ ~~se~~ turning red light on.
    for (int i = 9; i >=0; i--)  ——→ ~~for~~ count 9→0 in display
      PORTC = a[i]; _delay_ms(1000); → with 1s delay.
    PORTA = 0x08;  ——→ turning green light on.
    for (int i = 9; i >0; i--)  ——→ count 9→0 in display with
       PORTC = a[i]; _delay_ms(1k); → 1s delay.
  }
}
```

- Explanation 3:

In the inf loop: first we are sending 1 to port C thus C_0 is turned on => red led. display 9→0 in the 7 segment display using a counter explained before. And ~~finally~~ again sending 8→ PORTC. Thus C3 gets high and green led is tomed on. Again, a counter is called (9→0). And this repeats in an inf loop.

•) Assignment 4: (asm code. Double digit display from 30 → 00 and traffic light)

```
. INCLUDE "M32DEF. INC"
. ORG 0.
LDI R16, HIGH (RAMEND).
OUT SPH, R16.
LDI R16, LOW (RAMEND).
OUT SPL, R16.
LDI R16, 0x7E
MOV R0, R16 .
LDI R16, 0x30.
MOV R1, R16.
LDI R16, 0x6D.
MOV R2, R16.
LDI R16, 0x79
MOV R3, R16.
LDI. R16, 0x33
MOV R4, R16.
LDI R16, 0x5B.
MOV R5, R16.
LDI R16, 0x70.
~~MOV~~MOV R7, R16
LDI R16, 0x7F
MOV R8, R16.
LDI R16, 0x7B
MOV R9, R16.
.
LDI R16, 0xFF
OUT DDRC, R16
OUT DDRB, R16.
OUT DDRA, R16.

MAIN:
LDI R16, 0x01
OUT PORTA, R16
CALL MODXY
LDI R16, 0x02.
OUT PORTA, R16
CALL MODXY
```

```
JMP MAIN

MODXY:
LDI YL 0x043
LDI XL, 0x020
N1:
  CALL DISPLAY
CALL DELAY
DEC XL
CPI XL, 0xFF
BRNE N1.

DEC YL.
CPI YL, 0xFF
BRNE RXL.
RET
RXL:
LDI XL, 0x09.
JMP N1.

```

```
DISPLAY:
LD R16, Y
OUT PORTB, R16.
LD R16, X.
OUT PORB, R16.
RET.

DELAY:
LDI R19, 0x0A
LDI R18, 0xFF
LDI R17, 0xFF
L1:
DEC R17
CPI R17, 0xFF
BRNE L1
DEC R18
CPI R18, 0xFF
BRNE L1
DEC R19
CPI R19, 0x0FF
```

78. BRNE L1

79 RET.

• Explanation 4

Line 7 → 24: deals with storing the 7 segment display ~~codes~~ binary codes in respective registers such that R2 will store (a, b, c, d, e, f, g) codes for displaying x on a seven segment display.

Line (26 → 29): sets ~~pro~~ all ports of A, B, C, as output.

In the main program. firstly PORTA ← 1 ⇒ A0 is high ⇒ red light is on. Then a 30 down counter is called. Subsequently PORT ← 0x02 ⇒ A1 is high ⇒ green light is on and another 30 down counter is called.

The counter routine (L40 → 56) loads up R28 ← 3, R26 ← 0. Then r26. is decr every time and checked if it has gen a carry. If no carry continue decr. but once carry is gen ⇒ decr. R28 and check if it has any carry. If no carry. R26 ← 9 and start allover again (L43).
If carry is generated ⇒ MSD is exhausted ⇒ ~~leave~~ exit the counter.

Every time R26, R27 is updated a DISPLAY subroutine is called. (L58 → 63). It loads r16 with ~~[r28]~~ r [r28] and sends to PORTB and in the next clk cycle, loads r16 with r [r28] and sends to PORTB.