Materi Lab 7

LCD & Button Interrupt

Silahkan download dan buka File Suplemen Materi Lab 7 dari Scele untuk mempermudah pengerjaan lab minggu ini. Di dalamnya terdapat 3 file, yaitu **“hapsim_config.xml”**, **“test_interrupt.asm”**, dan **“test_lcd.asm”**.

File “**hapsim_config.xml**” adalah konfigurasi sample untuk LED, Button dan LCD hapsim. Kalian dapat menggunakan konfigurasi tersebut lalu memodifikasinya untuk lab ini dan lab seterusnya, namun diharapkan bisa membuat konfigurasi sendiri.

File **“test_interrupt.asm”** merupakan program sample yang menggunakan Button Interrupt untuk menghentikan jalannya loop LED di hapsim.

File “**test_lcd.asm**” merupakan program sample yang outputnya adalah menampilkan sebuah string pada LCD di hapsim.

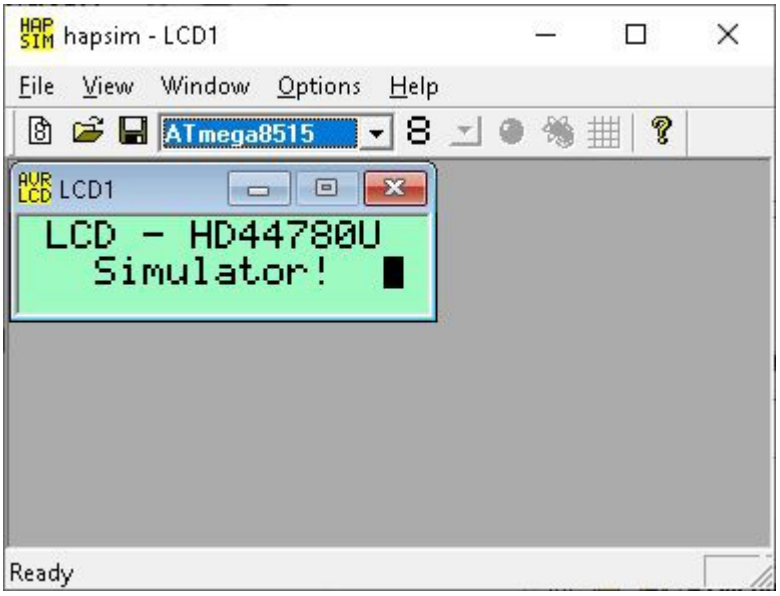
Kedua program sample tersebut dapat digunakan untuk template dan contoh dalam pengerjaan lab minggu ini.

Sebagai contoh, silahkan jalankan kedua program sample tersebut di AVR dan menyalakan hapsim dengan konfigurasi yang diberikan. Lalu pelajari apa yang dilakukan masing-masing program dan outputnya.

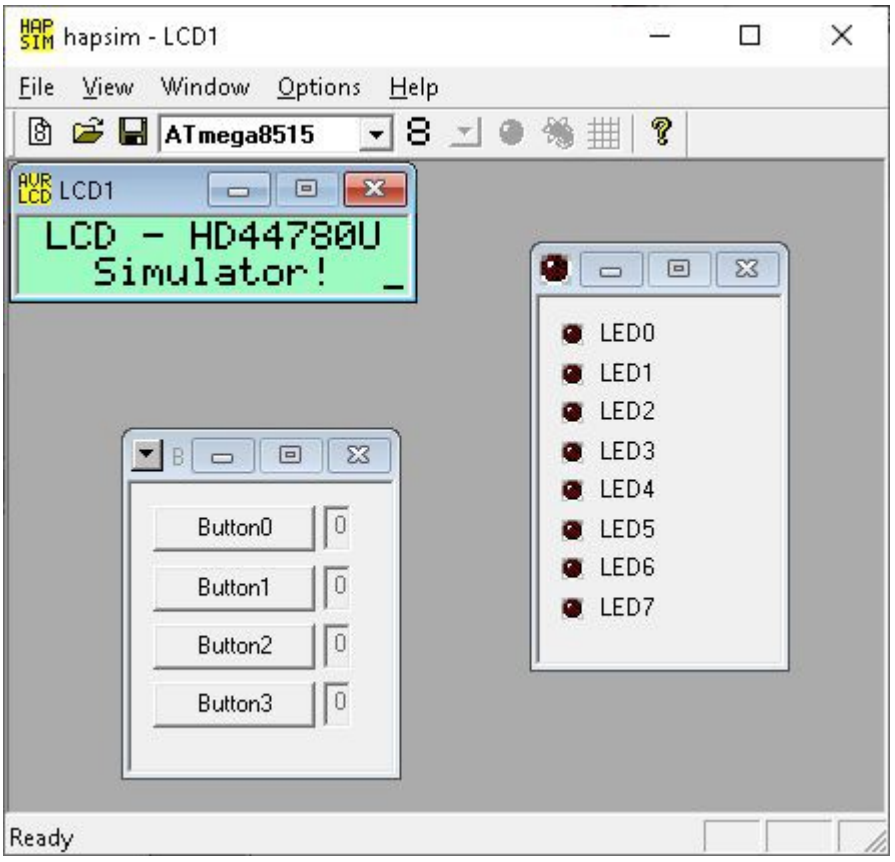
Berikut adalah link LCD Simulator untuk mengetahui apa fungsi setiap bit untuk mengatur fungsional LCD (Dalam contoh ini di-assign pada PORTB):
http://www.dinceraydin.com/djlcdsim/djlcdsim.html

Hapsim

Cara setup dan hook hapsim ke AVR sudah dipelajari pada Materi Lab 6, yaitu dengan Assemble/Build program dulu, tapi tidak di-Run, lalu membuka "hapsim.exe". Lalu ubah prosesor ke ATmega8515.

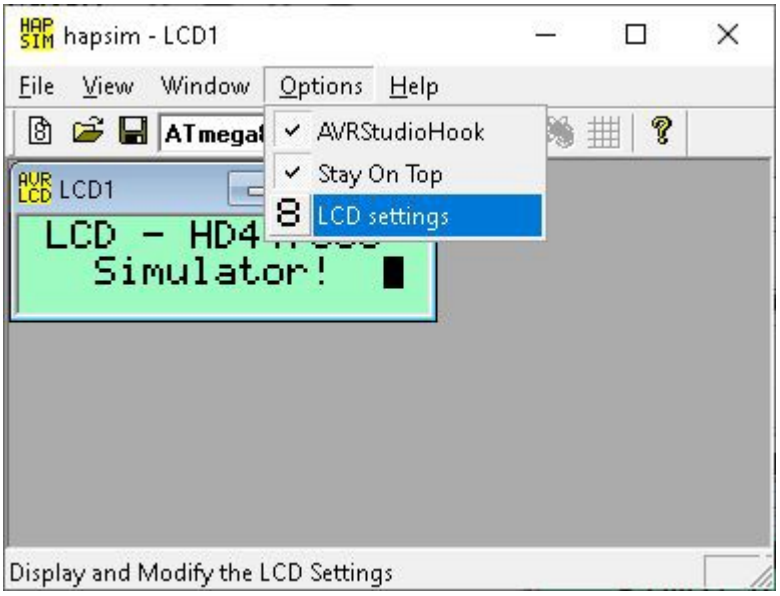


Untuk membuka konfigurasi sample "**hapsim_config.xml**", pada menu bar hapsim klik *File* > *Open Configuration* > *Pilih file*. Berikut tampilan hapsim dengan konfigurasi tersebut:



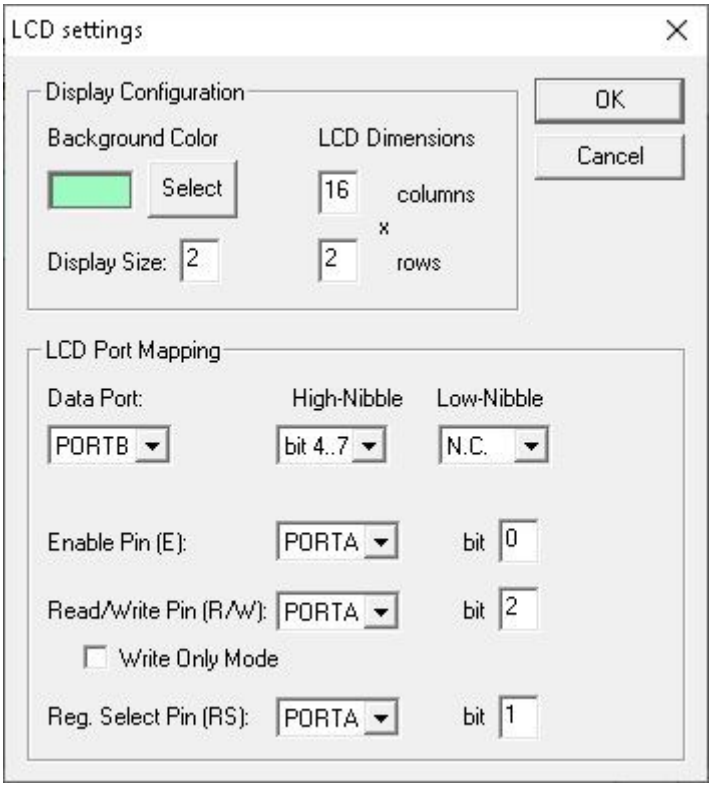
LCD

LCD adalah salah satu bentuk output yang tersedia pada AVR studio. LCD sudah ada secara otomatis pada hapsim. Kita hanya perlu konfigurasikan LCD untuk sesuai dengan port yang kita inginkan. Untuk mengubah settings LCD hanya perlu klik icon “LCD Settings” di toolbar.



Setup port LCD pada lab ini akan mengikuti halaman ke-10 dari Slide **AVR:LCD** di Scele, yaitu seperti berikut:

- PORTA bit 0 = EN
- PORTA bit 1 = RS
- PORTA bit 2 = R/W
- PORTB = DB0 – DB7



- **DB[7..0]** = Data 8-bit
- **EN** = Enable data transfer at falling edge
- **R/W** = Read/Write: 0 for Write/1 for Read
- **RS** = Register Select: 1 for data/0 for instruction

Silahkan buka program sample “**test_lcd.asm**” dari File Suplemen Materi Lab 7 dan perhatikan fungsi-fungsi di dalamnya.

Di bagian paling bawah terdapat kode yang berisi string yang akan ditampilkan di LCD:

```
message:
.db 190, 221, 195, 210, 176, 196, 217, "the peggies", 0
```

Setelah setup stack pointer, hal pertama yang dilakukan program adalah memanggil fungsi INIT_LCD_MAIN untuk setup port LCD:

```
INIT_STACK:
        ldi temp, low(RAMEND)
        ldi temp, high(RAMEND)
        out SPH, temp

rjmp INIT_LCD_MAIN
```

Untuk memberi value ke LCD dari sebuah data, sebuah program perlu melakukan set up untuk port yang akan digunakan LCD tersebut terlebih dahulu.

```
INIT_LCD_MAIN:
        rcall INIT_LCD
        ser temp
        out DDRA,temp       ; Set port A as output
        out DDRB,temp       ; Set port B as output
        rcall INPUT_TEXT
```

SER adalah command yang mengubah semua bit pada suatu register menjadi 1, command ini equivalent dengan “ldi r18, 0b11111111”.

DDRX adalah Data Direction Register dari sebuah port yang berasosiasi dengannya. Ini adalah sebuah I/O register yang menentukan sebuah port ingin digunakan sebagai input atau output. Setiap I/O register memiliki 8 bit.

Jika pada register DDRX kalian memberikan value 0b11111111, maka PORT X akan digunakan untuk output, sementara jika kalian memberikan nilai 0b00000000 maka akan digunakan untuk input. 0b00001111 berarti setengah pertama akan digunakan sebagai input dan setengah kedua akan digunakan sebagai output.

OUT adalah command yang menyimpan data dari register biasa ke I/O register. Karena kita ingin meng-output instruksi dan data ke LCD melalui PORT A dan PORT B, maka kita menset isi dari DDRA dan DDRB menjadi 0b11111111.

Kode di atas akan memanggil fungsi INIT_LCD dan INPUT_TEXT. Fungsi INIT_LCD akan setup display LCD tersebut.

```
INIT_LCD:
        cbi PORTA,1 ; CLR RS
        ldi PB,0x38 ; MOV DATA,0x38 --> 8bit, 2line, 5x7
        out PORTB,PB
        sbi PORTA,0 ; SETB EN
        cbi PORTA,0 ; CLR EN
        rcall DELAY_01

        cbi PORTA,1 ; CLR RS
        ldi PB,$0E ; MOV DATA,0x0E --> disp ON, cursor ON, blink OFF
        out PORTB,PB
        sbi PORTA,0 ; SETB EN
        cbi PORTA,0 ; CLR EN
        rcall DELAY_01

        rcall CLEAR_LCD ; CLEAR LCD

        cbi PORTA,1 ; CLR RS
        ldi PB,$06 ; MOV DATA,0x06 --> increase cursor, display sroll OFF
        out PORTB,PB
        sbi PORTA,0 ; SETB EN
        cbi PORTA,0 ; CLR EN
        rcall DELAY_01
        ret
```

Saat kita setup LCD settings di hapsim, kita gunakan konfigurasi berikut:

- **PORTA bit 0** = EN (Enable data transfer at falling edge)
- **PORTA bit 1** = RS (Register Select: 1 for data/0 for instruction)
- **PORTA bit 2** = R/W (Read/Write: 0 for Write/1 for Read)
- **PORTB** = DB0 – DB7 (Data 8-bit)

PORTX adalah Data Register dari sebuah port. Ia merupakan I/O register yang menyimpang data atau instruksi untuk port yang ingin digunakan. Terdapat 8 bit dalam tiap PORTX.

Karena instruction code untuk setup LCD memerlukan 10 bit (1 bit untuk RS, 1 bit untuk R/W, 8 bit untuk data), kita perlu menggunakan 2 PORTX berbeda, yaitu PORTA dan PORTB. Seluruh 8 bit PORTB digunakan untuk 8 bit data, sementara pada PORTA, yang digunakan hanya bit 0-2.

CBI merupakan command yang me-set value suatu bit di I/O register menjadi 0. Pada kode di atas, “cbi PORTA,1” berarti mengubah value bit 1 dalam PORTA (yaitu bit untuk RS) menjadi 0, sehingga instruction code LCD yang dimasukkan berupa instruksi, bukan data.

SBI adalah kebalikan dari CBI, dan merupakan command yang me-set value suatu bit di I/O register menjadi 1.

Bagian pertama fungsi di atas merupakan instruction code bertipe "***Function Set***" yang menentukan panjang data dan baris yang akan digunakan LCD. Pertama kita akan CBI bit 1 pada PORTA. Lalu Kkita akan load data instruksi 8-bit yang dibutuhkan ke temporary register bernama PB, lalu memasukkannya ke I/O register PORTB. Instruction code lengkap tersebut kemudian akan dijalankan dengan meng-enable lalu disable data transfer EN.

Bagian kedua fungsi di atas merupakan instruction code bertipe "***Display ON/OFF Control***" yang menentukan display dan cursor LCD menyala atau tidak. Sama seperti bagian pertama, CBI bit 1 PORTA, lalu data instruksi 8-bit akan di load ke temporary register PB, lalu dimasukkan ke PORTB, dan akhirnya dijalankan dengan meng-enable dan disable EN.

Fungsi tersebut kemudian akan memanggil fungsi CLEAR_LCD yang akan clear display LCD pada hapsim.

Bagian terakhir fungsi di atas akan merupakan instruction code bertipe "***Entry Mode Set***" yang menentukan arah pergeseran cursor dan display LCD. Sama seperti 2 bagian sebelumnya, CBI bit 1 PORTA, lalu data instruksi 8-bit akan di load ke temporary register PB, lalu dimasukkan ke PORTB, dan dijalankan dengan enable dan disable EN.

Selebihnya tentang berbagai tipe instruction code LCD yang ada, dapat dilihat dari slide perkuliahan tentang **AVR:LCD**.

Fungsi di atas memanggil CLEAR_LCD yang berupa:

```
CLEAR_LCD:
        cbi PORTA,1 ; CLR RS
        ldi PB,$01 ; MOV DATA,0x01
        out PORTB,PB
        sbi PORTA,0 ; SETB EN
        cbi PORTA,0 ; CLR EN
        rcall DELAY_01
        ret
```

Fungsi ini pada akhirnya hanya merupakan instruction code LCD bertipe "***Clear Display***" yang akan mengosongkan isi display LCD. Semua bit dalam instruction code ini memiliki value 0 kecuali bit terakhir, sehingga tampilan bitnya berupa 0b0000000001. Setelah CBI bit 1 pada PORTA, kita akan load data 8-bit ke temporary register PB, lalu dimasukkan ke PORTB, dan dijalankan dengan enable dan disable EN.

Fungsi INIT_LCD_MAIN juga memanggil fungsi INPUT_TEXT berikut:

```
INPUT_TEXT:
        ldi ZH,high(2*message) ; Load high part of byte address into ZH
        ldi ZL,low(2*message) ; Load low part of byte address into ZL
        ret
```

Fungsi di atas menginisiasi pointer ke Program Memory, yang berisi address dari “message” yang berisi data yang akan ditampilkan di LCD. Data dari “message” akan diakses per byte dan di-load ke register Z (ZH dan ZL), untuk nanti ditampilkan dalam LCD.

Setelah semua setup dan load data yang sudah dilakukan, sekarang saatnya akhirnya mulai menulis string pada LCD, melalui fungsi LOADBYTE:

```
LOADBYTE_PHASE1:
        lpm ; Load byte from program memory into r0

        cpi count, 7 ; Check if we've reached the end for the first line
        breq PAUSE ; If so, change line

        mov A, r0 ; Put the character onto Port B
        rcall WRITE_TEXT
        inc count
        adiw ZL,1 ; Increase Z registers
        rjmp LOADBYTE_PHASE1

PAUSE:
        rcall DELAY_02
        rcall CLEAR_LCD
        ldi count, 0

LOADBYTE_PHASE2:
        lpm ; Load byte from program memory into r0

        tst r0 ; Check if we've reached the end of the message
        breq LOOP_LCD ; If so, quit

        mov A, r0 ; Put the character onto Port B
        rcall WRITE_TEXT
        adiw ZL,1 ; Increase Z registers
        rjmp LOADBYTE_PHASE2
```

Fungsi ini merupakan loop yang akan me-load data yang akan ditampilkan per byte lalu memasukkannya ke temporary register bernama A. Fungsi ini lalu akan memanggil fungsi WRITE_TEXT untuk menulis isi register A ke LCD.

LPM adalah command yang akan load data dari Program Memory ke register r0. Pointer pada Z register digunakan untuk mengakses Program Memory yang berisi address dari “message”. Lalu “message” pun berisi string yang akan ditampilkan ke LCD.

CPI adalah command untuk mengecek apakah register tertentu memiliki value sama dengan immediate yang diberikan. Di sini “cpi count 7” akan mengecek apakah program sudah mengiterasi 7 karakter pertama dari “message”.

TST adalah command untuk mengecek apakah register tertentu memiliki value 0 atau tidak. Di sini, “tst r0” akan mengecek apakah pointer sudah mencapai akhir dari “message” atau tidak. Karena jika pointer sudah mencapai akhir, yang di load hanya 0.

BREQ adalah command untuk mengecek apakah TST sebelumnya equal atau tidak. BREQ berarti “Branch if Equal” sehingga program akan melakukan branch ke fungsi lain jika value register r0 = 0. Pada LOADBYTE_PHASE 2, jika pointer sudah mencapai akhir “message”, berarti value register r0 = 0, dan program akan jump ke fungsi END_LOOP dan mengulang dari awal.

Command MOV akan memindahkan value dari register r0 ke temporary register bernama A. Lalu program akan memanggil fungsi WRITE_TEXT yang baru akan menulis string tersebut ke LCD. Setelah itu, pointer pada Z register akan di-increment sekali agar pointer Program Memory menunjuk ke byte address berikutnya.

Dan fungsi ini akan terus berulang sampai tidak ada lagi data untuk di-load, atau dengan kata lain, semua isi “message” sudah ditampilkan di LCD.

Fungsi penting terakhir adalah fungsi WRITE_TEXT yang akhirnya akan benar-benar menampilkan string ke LCD:

```
WRITE_TEXT:
        sbi PORTA,1 ; SETB RS
        out PORTB, A
        sbi PORTA,0 ; SETB EN
        cbi PORTA,0 ; CLR EN
        rcall DELAY_01
        ret
```

Fungsi ini pada dasarnya merupakan instruction code LCD bertipe “***Write Data to RAM***”. Yang pertama dilakukan adalah SBI bit 1 PORTA, berarti mengubah value bit 1 dalam PORTA (yaitu bit untuk RS) menjadi 1, sehingga instruction code LCD yang dimasukkan berupa data untuk ditampilkan, bukan instruksi.

Data yang ingin ditampilkan sudah di-load di dalam temporary register A, kita hanya perlu memasukkannya ke I/O register PORTB dengan command OUT. Akhirnya instruction code dijalankan dan data ditampilkan pada LCD dengan meng-enable lalu disable EN.

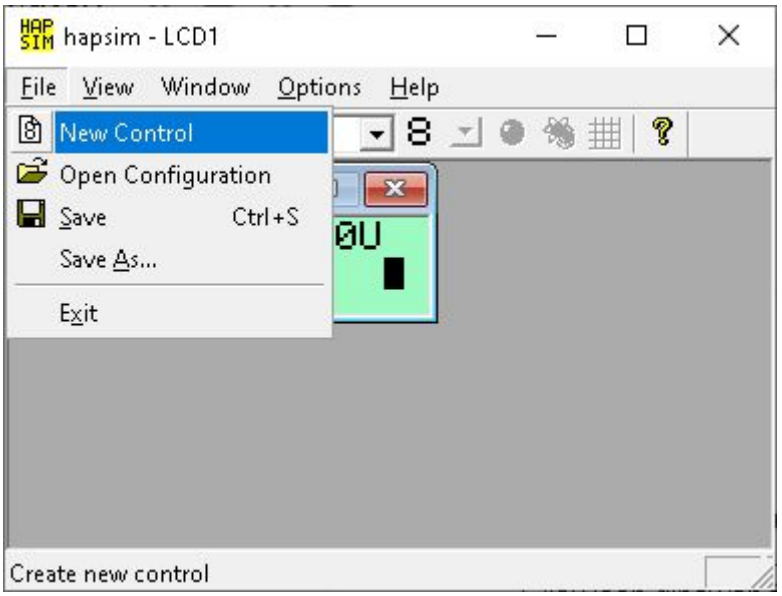
Setelah semua itu, program akan menuju fungsi LOOP_LCD dan melakukan jump ke MAIN. Lalu program akan melakukan ulang instruksi-instruksi dari awal.

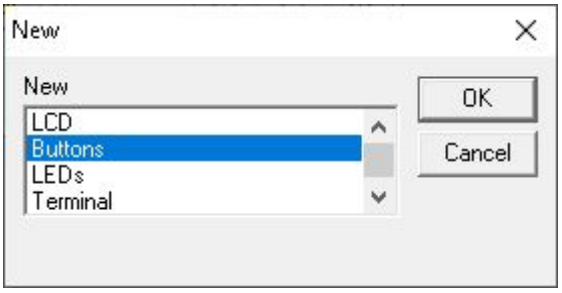
Selain bagian program yang sudah dijelaskan di atas, ada juga section “DELAYS” yang diberikan di bagian bawah kode program. Kalian bisa mengubah delay yang digunakan sesuka hati. Jika ingin LCD berjalan lebih cepat, gunakan DELAY_00, jika ingin LCD berjalan lebih lambat, gunakan DELAY_02.

Sekian penjelasan tentang program “**test_lcd.asm**” yang bisa digunakan sebagai template untuk Lab 7.

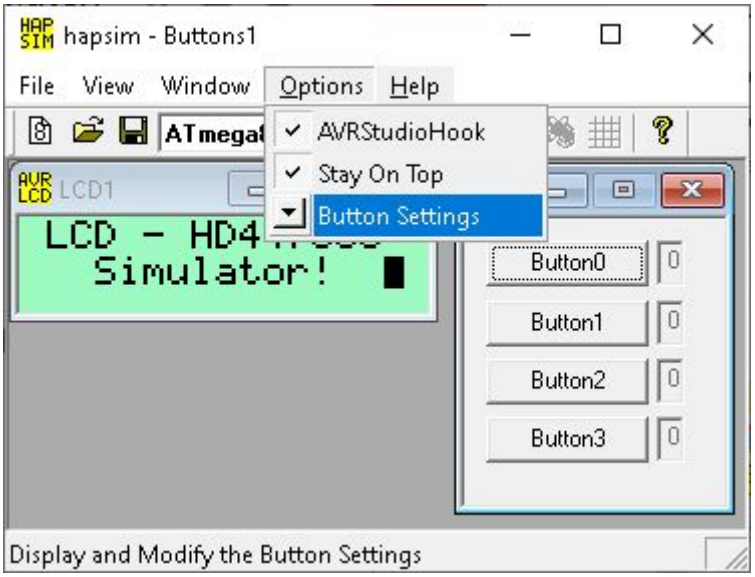
Buttons

Buttons adalah salah satu bentuk input yang tersedia pada AVR studio. Kita perlu menambahkan Buttons lalu konfigurasikan dengan port yang sesuai. Untuk menambah Buttons hanya perlu klik icon “New Control” di toolbar, lalu pilih Buttons dan klik OK.





Untuk mengubah setting Buttons hanya perlu klik icon “Button Settings” di toolbar.



Pada lab 5 ini, port yang dipakai untuk Buttons yang digunakan adalah PORT D dengan bit seperti di bawah. Text yang ditampilkan tiap button juga dapat diubah di sini.