

Controlling the HT1621 LCD Controller With the HT48 MCU Series

D/N : HA0018E

Introduction

The HT1621 is a 128-bit multi-function LCD controller device with internal RAM memory mapping. The software configuration features allow the device to be used in a wide range of LCD applications including LCD modules and display subsystems. The interface between the main controller and the HT1621 require only 4 to 5 lines for its full implementation. The device contains a full power down function, which will reduce supply current to extremely low levels. This application shows how the HT48R30A-1 microcontroller can be used to control the HT1621 the, and shows how the LCD pixels can be illuminated or turned off.

Operating Description

Before using the HT1621, a flag code should be sent to the device to indicate which type of operation is required. Flag codes are defined as follows:

Operating	Status	Flag Code
Read	Data	110
Write	Data	101
Read-modify-write	Data	101
Control	Command	100

When it is required to illuminate the LCD, two commands must first be sent out: SYSTEM ENABLE and LCD ON. The SYSTEM ENABLE instruction-code is: 1000000001X (X is a don't care bit). The LCD ON instruction-code is: 10000000011X (X is a don't care bit). After the operation has finished, the SYSTEM DISABLE instruction can be used to switch off the LCD.

Because the data is transmitted serially, the data must be presented on the DATA INPUT pin, followed by a read/write enable signal (WR).

The RAM area non-continuous write data flow is as follows:

1	0	1	A5	A4	A3	A2	A1	A0	D0	D1	D2	D3	end	Next flow
---	---	---	----	----	----	----	----	----	----	----	----	----	-----	-----------

First a transmit code of 101 is sent out, indicating that the following data is to be written. Then a corresponding address code A5–A0 is transmitted, the highest 4-bits of which are invalid. Bits D0–D3 are used to indicate the corresponding address. The read process, with the exception of the flag bit which is different, is similar.

For a continuous read-write, send out the start address, after the operation finishes the address will automatically be incremented by 1.

Program Example

This example shows how the entire LCD can be illuminated and extinguished. The program flow is as follows:

Initialise system→Start 1621→Clear all LCD bits→illuminate all the LCD bits→Read some bits and compare→Initialise system

Circuit Drawing: Refer to the HT1621 the specification.

```

;1621driver.asm
; This program is controls HT1621 with the HT48R30A-1
; Control port structure :
;     PB1 -- datum
;     PB2 -- WRB
;     PB3 -- CSB
;     PB4 -- RDB
;     OSC: Ext. Crystal
;     WDT clock source:Disable WDT
;     Input type PA: Schmitt Trigger
;     Pull-high PA: Pull-high PA
;     Pull-high PB: Pull-high PB
;     BZ/BZB :     BZ ENABEL/BZB DISABLE
;     fSYS:      4M
;Note: When writing the program, the order must be correct

```

```

include      ht48r30a-1.inc

csb   equ    pb.3
csbc  equ    pbc.3
wrbc  equ    pbc.2
datum equ    pb.1
datumc equ   pbc.1
rdb   equ    pb.4
rdbc  equ    pbc.4
lig   equ    pc.3

;-----
num_mem equ   [7fh]
;-----
;macro definition
;delay macro instruction, delay 5ms
d_1   macro
      jmp    $+1
      jmp    $+1
      nop
endm

;-----
lcddriver .section 'data'
count     db    ?           ;loop counter
code_datum db    ?           ;command code or memory datum bits
code_datum1 db    ?         ;only used in read_modify_write mode
mem_addr  db    ?           ;memory address for selecting segment

temp_da   db    ?
t_addr_h  db    ?           ;buffer
;-----

lock      .SECTION 'CODE'
org       00h

      jmp    start
org       04h
      reti
org       08h
      reti

```

```
start:
    clr    pb                ;Initialise
    set    csb
    clr    pbc
    set    pbc.0
    clr    pc
    clr    pcc
    clr    intc

    mov    a, 50h
    mov    num_mem, a
    mov    a, 20h
    mov    mp0, a

clr_ram:
                                ;Initialise ram
    clr    r0
    inc    mp0
    sdz    num_mem
    jmp    clr_ram

;-----
ini_status:
    mov    a, 87h
    mov    tmrc, a

show_k:
    set    lig
    mov    a, 01 h          ;SYS ENABLE
    mov    code_datum, a
    call   send_command

LO:
    mov    a, 029h         ;4com;1/3bias
    mov    code_datum, a
    call   send_command

    mov    a, 3            ;LCD On
    mov    code_datum, a
    call   send_command
    call   clr_lcm         ;cls lcd
    jmp    $+1
    jmp    $+1
    call   show_lcm        ;illuminate all pixels
```

```

        clr     code_datum    ;read then write to the same address
        mov     a, 4
        mov     mem_addr, a
        call    read
        mov     a, 07h
        xor     a, code_datum
        snz     z
        jmp     error
        mov     a, 2
        mov     code_datum, a
        call    send_command
        jmp     $+1
        jmp     start        ;repeat
error:
        jmp     $

```

```

;-----
;Purpose:    send command
;Parameter:  code_datum : byte
;Return:     none
;Modified:   acc, status
;-----

```

```

send_command proc
        clr     CSB
        clr     datumC

        set     datum
        clr     WRB        ;COMMAND ID '100'
        d_1
        set     WRB        ;1
        nop
        clr     datum      ;00
        clr     WRB
        d_1
        set     WRB
        nop
        CLR     WRB
        d_1
        set     WRB

        mov     A, 8        ;send code
        mov     count, A

```

```

LOOP1:
    clr    datum
    sz     code_datum.7
    set    datum
    rl     code_datum
    clr    WRB
    d_1
    set    WRB
    sdz    count
    jmp    loop1

    clr    WRB
    d_1
    set    WRB
    nop
    set    CSB          ;close csb signal, device not selected
    ret
send_command  endp

;-----
;Purpose:      write datum to 1621
;Parameter:
;code_datum:   byte
;mem_addr:     byte
;Return:       none
;Modified:     acc, status
;-----
write:
    clr    CSB
    clr    datumc

    set    datum
    clr    WRB          ;WRITE mode ID '101'
    d_1
    set    WRB

    clr    datum
    clr    WRB
    d_1
    set    WRB

    set    datum
    clr    WRB
    d_1
    set    WRB

    mov    a, 6
    mov    count, a

```

```
writeloop1:
    clr    datum
    sz     mem_addr.5    ;send memory address to select segment
    set    datum
    clr    WRB
    d_1
    set    WRB
    rl     mem_addr
    sdz    count
    jmp    writeloop1

    mov    a, 4
    mov    count, a
```

```
writeloop2:
    clr    datum
    sz     code_datum.0 ;send memory content to decide
                                ;comments's state

    set    datum
    clr    WRB
    d_1
    set    WRB
    rr     code_datum
    sdz    count
    jmp    writeloop2
    set    CSB
    ret
```

```
-----
;Purpose:    read datum from 1621
;Parameter:
;mem_addr:   byte
;Return:
;code_datum: byte
;Modified:   acc, status
-----
```

```
read  proc
    clr    CSB
    clr    datumc

    set    datum
    clr    WRB    ;READ mode ID '110'
    d_1
    set    WRB

    clr    WRB
    d_1
    set    WRB
```

```

        clr     datum
        clr     WRB
        d_1
        set     WRB

        mov     a, 6
        mov     count, a

readloop1:
        clr     datum
        sz     mem_addr.5    ;send memory address to select segment
        set     datum
        clr     WRB
        d_1
        set     WRB
        rl     mem_addr
        sdz    count
        jmp    readloop1

        set     datumc
        mov     a, 4
        mov     count, a

readloop2:
        clr     RDB
        d_1
        set     RDB
        rr     code_datum
        clr     code_datum.3
        sz     datum        ;send memory content for decide
                                ;comments's state

        set     code_datum.3
        sdz    count
        jmp    readloop2

        mov     a, 0fh
        andm   a, code_datum
        set     CSB
        ret

read  endp

```

```

;-----
;Purpose:      read datum from 1621, then write a datum in the same
               register
;Parameter:
;mem_addr:    byte
;Return:      none
;Modified:    acc, status
;-----
rm_write      proc
               clr      CSB
               clr      datumc

               set      datum      ;READ-MODIFY-WRITE mode ID '101'
               clr      WRB
               d_1
               set      WRB

               clr      datum
               clr      WRB
               d_1
               set      WRB

               set      datum
               clr      WRB
               d_1
               set      WRB

               mov     a, 6
               mov     count, a

rmwloop1:
               clr      datum
               sz      mem_addr.5  ;send memory address to select segment
               set      datum
               clr      WRB
               d_1
               set      WRB
               rl      mem_addr
               sdz     count
               jmp     rmwloop1

               set      datumc
               mov     a, 4
               mov     count, a

```

```

rmwloop2:
    clr     RDB
    d_1
    set     RDB
    rr      code_datum1
    clr     code_datum1.3
    sz      datum      ;read memory content out
    set     code_datum1.3
    sdz     count
    jmp     rmwloop2

    mov     a, 0fh
    andm   a, code_datum1

    clr     datumc
    mov     a, 4
    mov     count, a
    mov     a, temp_da
    andm   a, code_datum

rmwloop3:
    clr     datum
    sz      code_datum.0 ;send memory content to decide
                                ;comments's state

    set     datum
    clr     WRB
    d_1
    set     WRB
    rr      code_datum
    sdz     count
    jmp     rmwloop3
    set     CSB
    ret

rm_write   endp
;-----
show_lcm:
    mov     a, 00h
    mov     t_addr_h, a
    mov     a, 31h
    mov     num_mem, a
clr_n1:
    mov     a, t_addr_h
    mov     mem_addr, a
    mov     a, 07h
    mov     code_datum, a
    call   write
    inc     t_addr_h
    sdz     num_mem
    jmp     clr_n1
    ret

```

```
-----  
clr_lcm:  
    mov     a, 00h  
    mov     t_addr_h, a  
    mov     a, 31h  
    mov     num_mem, a  
clr_n:  
    mov     a, t_addr_h  
    mov     mem_addr, a  
    mov     a, 0h  
    mov     code_datum, a  
    call    write  
    inc     t_addr_h  
    sdz     num_mem  
    jmp     clr_n  
    ret  
-----
```