Tea Cooker

Nathalie Künzle, Michael Linder

13th February 2022

0.1 Introduction

Our end-of semester project for MICRO-210 consists of a tea-cooker. After a manually entered delay, the machine starts heating water, adds a tea bag, lets it steep for a certain (by the user entered) time and removes the tea bag.

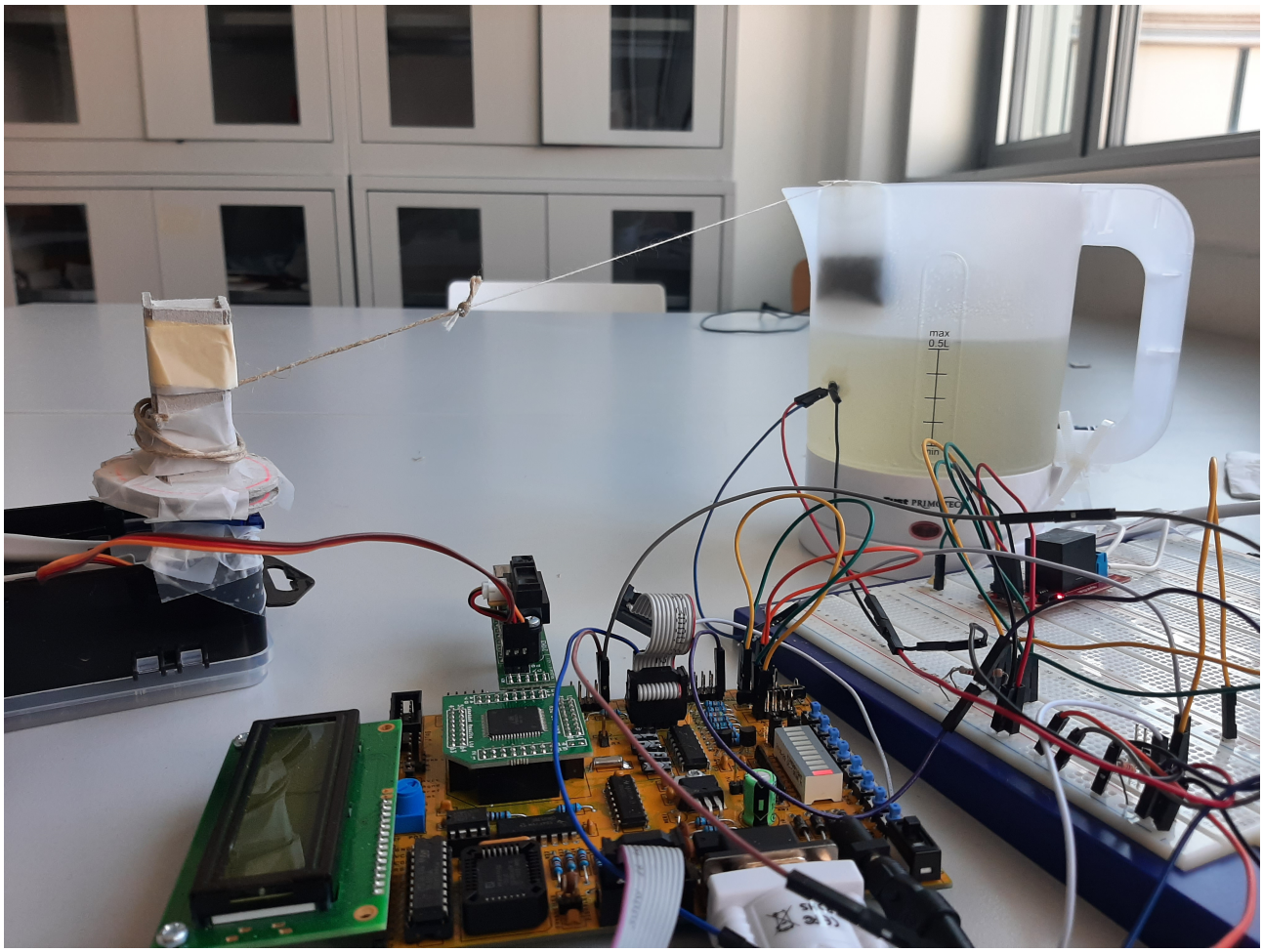


Figure 1: Picture of the TeaCooker in action.

0.2 Short Project description

Our machine is based on a Atmega128L microcontroller on a STK-300 developement board and a standard kettle, at which we replaced the mechanical switch with a relay to control the high voltage currents with the microcontrollers 5V signals. A temperature sensor, which is fixed in a hole in the kettle with water and heat resistant glue, is used to stop heating when the water is boiling before adding the tea bag using the servo motor. After a certain steep time, the motor is used again to remove the tea bag.
The software, which is coded in AVR-assembly, implements a state machine next to multiple interrupt routines. The link between hardware and software is done by multiple drivers controlling the peripherals.

User Manual

At first, fill half a liter of water into the kettle and fix the tea bag to the motor. In order to start the machine, switch on the STK-300 board.

Use a Terminal (i.e. Teraterm) on the computer connected to the board for the UART connection (RS323, 9600 bits/s, 1 stop bit, no parity bit). After starting the machine, the settings dialog appears where you can enter delay time and steep time or leave the default settings (0 min for delay and 5 min for steep time). The delay time is between finishing the settings and begin of the heating.

The time left for the delay and tea steep, aswell as the temperature during heating are displayed on the LCD screen. The machine can be reset any time by pressing button 0.

Once the tea bag is removed, the tea is ready to drink. You can restart the machine by pressing button 1 (which keeps the previously selected settings and skips the setting state) or button 0 (reset). To turn off the machine, turn off the voltage of the STK-300 board.

Technical Description of the Material

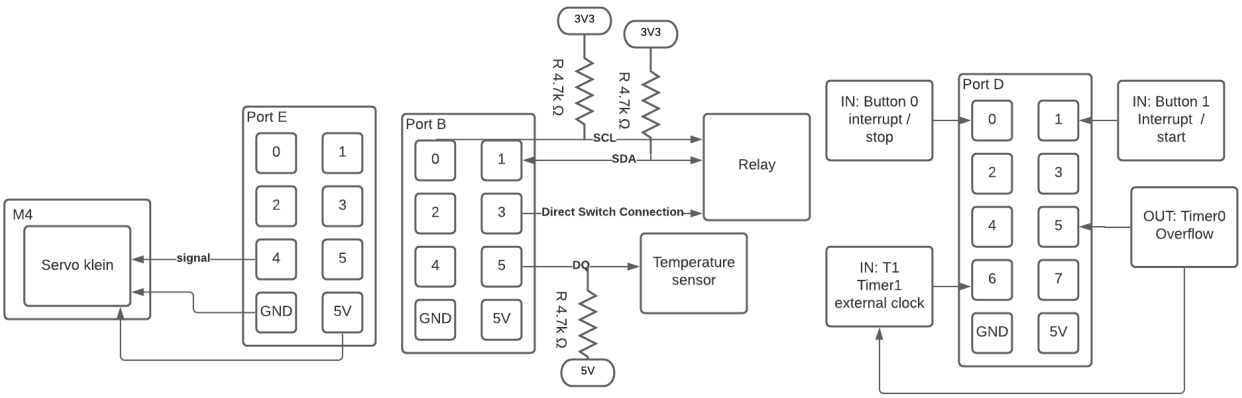
2.1 Bill of Material

- MCU Atmega128L on a STK300 card
- Temperature sensor DS18B20+
- Motor Tower Pro Micro Servo SG90
- USART interface cable
- Hitachi44780U 2x16 LCD Screen
- SparkFun Qwiic Single Relay with a Cubic, Single-pole 10A Power Relay (G5LE-1-ASI)
- Kettle Fust PrimotecQ water heater.

2.2 Material Description

2.2.1 MCU Atmega128L on a STK300 card

The Atmega128L is running at a frequency of 4MHz. The peripherals are connected to the MCU using the ports as follows:



Additionally, the LCD is connected to the designated port on the SDK300. The connection between PortD 5 and 6 can be a jumper.

2.2.2 DS18B20+

The DS18B20+ is a 1-Wire digital thermometer with a 9 to 12 bit Celsius temperature measurement and an alarm function. The sensor is suited to detect boiling water given its maximal operation temperature of 125 °C. However. we observed that $55\mathrm{deg}$ is the shown boiling temperature due to delays in the heat transfer. Even tough powering over the DQ line is possibly, we connect a separate 5V power. A $4.7k\Omega$ pull-up resistor is placed at the DQ line. It's worth noting that there are two identically named versions of this temperature sensor with GND and VDD switched on the market.

2.2.3 Motor Tower Pro Micro Servo SG90

The SG90 servo motor works at an operating voltage of 4.8V, in a temperature range of 0 °C to 55 °C. He therefore needs to be placed at a certain distance to the hot water. The motor provides a torque of 1.8 kg.f.cm which is sufficient for our application (add and remove a tea bag).

The servo is controlled by a PWM signal with a period of about 20 ms. The signal is about 1 to 2 ms at Vcc (4.8V). The exact duration of the high impulse determines the speed of rotation: If the impulse duration is about 1.5ms, the speed is equal to zero. 1.5ms is called the nullpoint. The farther away from the nullpoint the actual impulse duration is, the higher is the speed. If the impulse duration is longer than 1.5ms, the rotation is counterclockwise, if it's shorter the rotation is clockwise. Unlike indicated in the datasheet (appendix A.3), the servo motor can make full rotations of above 360 °.

The servo is branched on P7 of module M4. This corresponds to branching the servo control signal to pin 5 of port E and connecting the servo to ground and VCC.

2.2.4 RS323 interface cable

We use the USB (PC side) to DB-9 (on the STK300 side) cable delivered for the RS-323 interface. On the computer, a terminal set up for serial, baud rate 9600, 1 stop bit, no parity bit is needed.

2.2.5 Hitachi44780U 2x16 LCD Screen

This is an LCD display powered with 2.7V to 5.5V. On both rows, 16 characters can be displayed. The connection the the MCU can be found in the appendix A.6.

2.2.6 SparkFun Qwiic Single Relay with a Cubic G5LE-1-ASI

The G5LE-1-ASI is a single pole power relay controlling currents up to 10 Amps at 230VAC. This is sufficient for our kettle given it's power consumption of 1200W.
The Qwiic relay board provides a I2C interface to control the relay G5LE-1-ASI. We use this interface from the arduino.This is why the *Direct Switch Concection* goes from the STK300 to the arduino pin 13. The arduino is connected to the I2C bus using pin 4 (SCL) and 5 (SDA).

Software Implementation

3.1 Main Program

Our software consists of a state machine as shown in the figure below. The current state is stored in the register r25. We kept the structure of the state machine quite generic: Changing from state A to B requires to go to the *state_change* segment where the jump to state B is done.

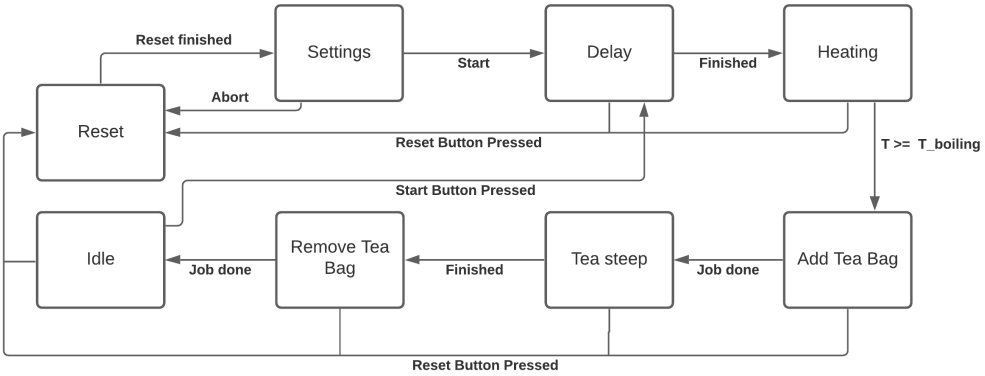


Figure 3.1: The state machine

The states are described in more detail below.

3.1.1 Reset

In this initial state, the initialization of the stack pointer and the peripherals takes place, external interrupts (INT0 and INT1 for the buttons) are enabled and the default values for the timers are stored into the SRAM. Finally, the current state is set to Settings State.

3.1.2 Settings

This state walks the user trough a settings dialogue to set the wait and steep time. We implemented a macro to read and convert four digit numbers using UART. Furthermore, a macro for decisions was implemented such that the user can decide where the program jumps to by entering one, zero or any other character (error handler).

3.1.3 Delay

The Timer/Counter0 and Timer/Counter1 are being initialized and preset to the value defined in the settings. Then, in a loop, the microcontroller is set into sleep mode (idle). He "awakens" at every overflow of timer0, but is reset to sleep and leaves the loop only when overflow of timer1 is reached.

3.1.4 Heating

The relay gets turned on. During heating, the temperature sensor measures the temperature and displays it on the LCD until the boiling temperature ($55\,\mathrm{deg}$) is reached.

3.1.5 Add Tea Bag

In this state, the subroutine *ang_rot_cw* is called to execute a clockwise rotation of the motor. The rotation speed and angle is fixed in the program code.

3.1.6 Tea Steep

The timers are being initialized and preset to the value stored in SRAM for the steep time. Then, as in state Delay, the MCU is set into sleep mode until the overflow interrupt of timer 1 is triggered.

3.1.7 Remove Tea Bag

The subroutine *ang_rot_ccw* is called to execute a counterclockwise rotation of the motor.

3.1.8 Idle

The MCU is set into sleep mode.

3.2 Memory use

We use register r25 (b3) during the whole program to store the current state. Other registers: R8-R9 (c0,c1), R18-R21 (a) and R22-R23 (b0,b1) are used for temporary variables.

We directly access the internal SRAM to store the values for the delay time and tea steep time. Beginning at the start of the SRAM (address 0x100), we reserve in total 5 bytes for the timer preset-values. All strings displayed

on the LCD and UART are stored in the code segment (Flash).

Details about Acces to Peripherals

4.1 Sleep Mode Usage

The microcontroller sleep modes are activated by setting SE of MCUCR at 1. SM is left at 00 (default value), which corresponds to idle state. The MCU is set into idle state by the command *sleep* until the first interrupt is triggered.

4.2 Timers

In Delay State and Tea Steep State, two timers are used: Timer/Counter0 and Timer/Counter1, both as Overflow Interrupt. They are enabled by setting TOIE0 and TOIE1 of TIMSK (Timer/Counter Interrupt Mask Register) at 1.

Timer 0 (8 bit) uses the external 32'768 Hz clock with a prescaler of 1024. TCNT0 is preset at 63 in order to generate an interrupt every 6 seconds. In the corresponding interrupt routine, the output value of PIN 5 of PORT D is changed. The signal in this way generated is used as clock for Timer1 by connecting it to PIN 6 of PORT D.

Timer 1 (16 bit) uses an external clock on PIN 6 of PORT D, at rising edge. For this we set the clock select bits of TCCR1B (Timer/Counter Control Register 1B) at 7. We use no prescaler on Timer1. TCNT1 (2 bytes) is preset to a value corresponding to the value entered in settings (or default value defined in reset).

4.3 Servo Motor

The servo control impulse signal is generated using WAIT macros and value-decrementing loops. The servo output pin (pin 5 of port E) is at 0 as long as a 20ms WAIT macro is executed, then VCC until a certain value stocked in a1:a0 and decremented in a loop arrives at 0. Such an impulse is generated b0 times. Below is a typical impulse sent with pre- and post fluctations that don't affect the movement.

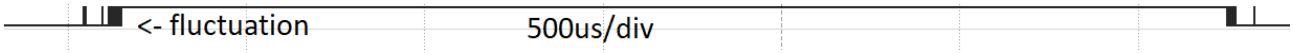


Figure 4.1: Recording of 1 impulse read using a logical analyzer

4.4 Temperature Sensor

We use the *DS18B20+* temperature sensor with a Dallas-one-wire interface. We connect a $4.7k\Omega$ pull-up resistor on the data wire. An important note is that the GND and VCC pins on the datasheet are reversed compared to the actual sensor we're using. The waveform below shows a typical temperature read consisting of resetting, read-scratchpad command (0xBE) and 2-byte read.

Before this, a reset - skipROM - convertTemperature command must be sent.

4.5 Relay

The relay can be turned on (command 0x01) and off (command 0x00) using I2C at address 0x18. We initially implemented the I2C driver on the ATMega128. During testing we noticed stability problems on SCL and SDA as soon as the relay was connected (and not only a logical analyzer). It looked like the relay slightly pulled the I2C lines down. Testing the same wiring with an arduino worked flawlessly. We verified that the waveform produced by the arduino is identical to the one of the ATMega128.
An other way to control the relay is to directly connect a wire to the switch-pin in order to control the current flow by applying *HIGH* or *LOW*. As we hesitated to connect a wire directly to the relay, which is not isolated, we choose to connect this wire (PortB 3) to the arduino and translate the input to the I2C commands. The arduino therefore replaces the pins on PortB 0 and 1.

4.6 Buttons

The buttons are branched as external asynchronous interrupts *INT0* (stop button) and *INT1* (start button). As the buttons are active-low we need to set *ISCn1* and *ISCn0* to zero. The coresponding interrupt routines take care of resetting or starting the machine.

4.7 LCD

The LCD screen is accessed using its controller which can be treated similarly to a sram memory. We have an instruction register (IR) at 0x8000 and a data register (DR) at 0xC000. As suggested by the name, the IR can be used to send instructions to (i.e. 0x01 to clear the display) while the DR is used to store the ASCII characters to display. For example, the display can be cleared by executing the following code (busy flag ignored):

```
1 ldi r16, 0x01
2 sts 0x8000, r16
```

It's worth noting that the DR is larger than the characters that can be displayed which means it's in the responsibility of the software to ensure a correct presentation.

4.8 UART interface

4.8.1 Protocol explanation

UART is a asynchronos serial protocol based on two lines, TX (tranceive = output) and RX (receive = input). We use a frequency of 9600Hz with 8 data-bits, 1 stop-bit and no parity-bits. This means a packet starts with a *LOW* followed by 8 data bits (one ASCII character) and a *HIGH* as stop bit. Between transmissions, RX and TX are pulled up.

4.8.2 Implementation

We use a frequency of 9600Hz with 8 data-bits, 1 stop-bit and no parity-bits. The provided library *uart.asm* relies on the USART module embedded in the ATMega128L.To read numerical values, we added a macro that 1. reads a character 2. converts the ASCII code to its digits value by subracting the code of zero and 3. multiplies the digit by the coresponding power of ten.

Reference

Next to the data sheets in the annex we referred to the following documentation provided on the course server;

- AVR instruction manual
- ATMEGA128 Datasheet

Furthermore, we referred to `https://www.sparkfun.com/products/15093` for informations about the qwiic power relay as there's no official datasheet of this relay-boar. The datasheet of the relay itself exists but probides no important information.

Source of Code

We took the following libraries from the courses TP:

- definitions.asm
- i2cx.asm
- lcd.asm
- macros.asm
- math.asm
- printf.asm
- wire1.asm
- motor1.asm (partly)

Some other code was written by us but inspired by the TPs.

Appendix

A.1 Source Code

```
; file: definitions.asm   target ATmega128L-4MHz-STK300
; purpose library, definition of addresses and constants
; 20171114 A.S.

; === definitions  ===
.nolist         ; do not include in listing
.set    clock   = 4000000

.def    char    = r0    ; character (ASCII)
.def    _sreg   = r1    ; saves the status during interrupts
.def    _u  = r2    ; saves working reg u during interrupt
.def    u   = r3    ; scratch register (macros, routines)

.def    e0  = r4    ; temporary reg for PRINTF
.def    e1  = r5

.equ    c   = 8
.def    c0  = r8    ; 8-byte register c
.def    c1  = r9
.def    c2  = r10
.def    c3  = r11

.equ    d   = 12    ; 4-byte register d (overlapping with c)
.def    d0  = r12
.def    d1  = r13
.def    d2  = r14
.def    d3  = r15

.def    w   = r16   ; working register for macros
.def    _w  = r17   ; working register for interrupts

.equ    a   = 18
.def    a0  = r18   ; 4-byte register a
.def    a1  = r19
.def    a2  = r20
.def    a3  = r21

.equ    b   = 22
.def    b0  = r22   ; 4-byte register b
.def    b1  = r23
.def    b2  = r24
.def    b3  = r25

.equ    px  = 26    ; pointer x
.equ    py  = 28    ; pointer y
.equ    pz  = 30    ; pointer z

; === ASCII codes
.equ    BEL =0x07   ; bell
.equ    HT  =0x09   ; horizontal tab
.equ    TAB =0x09   ; tab
.equ    LF  =0x0a   ; line feed
.equ    VT  =0x0b   ; vertical tab
```

```
.equ    FF  =0x0c   ; form feed
.equ    CR  =0x0d   ; carriage return
.equ    SPACE   =0x20   ; space code
.equ    DEL =0x7f   ; delete
.equ    BS  =0x08   ; back space

; === STK-300 ===
.equ    LED = PORTB ; LEDs on STK-300
.equ    BUTTON  = PIND  ; buttons on the STK-300

; === module M2 (encoder/speaker/IR remote) ===
.equ    SPEAKER = 2 ; piezo speaker
.equ    ENCOD_A = 4 ; angular encoder A
.equ    ENCOD_B = 5 ; angular encoder B
.equ    ENCOD_I = 6 ; angular encoder button
.equ    IR  = 7 ; IR module for PCM remote control system

; === module M5 (I2C/1Wire) ===
.equ    SCL = 0 ; I2C serial clock
.equ    SDA = 1 ; I2C serial data
.equ    DQ  = 5 ; Dallas 1Wire
                ; master transmitter status codes, Table 88
.equ    I2CMT_START = 0x08      ; start
.equ    I2CMT_REPSTART = 0x10   ; repeated start
.equ    I2CMT_SLA_ACK= 0x18     ; slave ack
.equ    I2CMT_SLA_NOACK = 0x20  ; slave no ack
.equ    I2CMT_DATA_ACK = 0x28   ; data write, ack
.equ    I2CMT_DATA_NOACK = 0x30 ; data write, no ack
                ; master receiver status codes, Table 89
.equ    I2CMR_SLA_ACK   = 0x40  ; slave address ack
.equ    I2CMR_SLA_NACK  = 0x48  ; slave address no ack
.equ    I2CMR_DATA_ACK = 0x50   ; master data ack
.equ    I2CMR_DATA_NACK= 0x58   ; master data no ack

; === module M4 (Keyboard/Sharp/Servo) ===
.equ    KB_CLK  = 0 ; PC-AT keyboard clock line
.equ    KB_DAT  = 1 ; PC-AT keyboard data line
.equ    GP2_CLK = 2 ; Sharp GP2D02 distance measuring sensor
.equ    GP2_DAT = 3 ; Sharp GP2D02 distance measuring sensor
.equ    GP2_AVAL = 3; Shart GP2Y0A21 distance measuring sensor
.equ    SERVO1  = 4 ; Futaba position servo

; === module M3 (potentiometer/BNC) ===
.equ    POT = 0 ; potentiometer
.equ    BNC1    = 2 ; BNC input
.equ    BNC2    = 4 ; BNC input
.list
```

```
/*
 * definitions_teaMaker.inc
 *
 *  Created: 08.04.2021 13:56:36
 *
 * Definitions specific to our program
 */

 ; ==============
 ; State machine
 ; ==============
 .equ STATE_SETTINGS            = 0b00000001
 .equ STATE_DELAY               = 0b00000011
 .equ STATE_HEATING             = 0b00000100
 .equ STATE_ADD_TEA_BAG         = 0b00000101
 .equ STATE_TEA_STEEP           = 0b00000110
 .equ STATE_REMOVE_TEA_BAG      = 0b00000111
 .equ STATE_IDLE                = 0b00001000

 ; ==========
 ; Registers
 ; ==========
 .def CURRENT_STATE     = R25


 ; ==============
 ; Motor
 ; ==============
 .equ npt = 1544            ; null point

; ==========
; Temperature
; ==========
 .equ cooking_water_temp = 0x0370 ; corresponds to 55 degree
```

```
/*
Delay_Timer.asm

This file defines macros and sub-routines used for the timers (Timer/Counter 0 and 1) and for timer related

content:
- Macro to calculate timer preset value
- macro/subroutines to preset and initialize timer0 and timer1
- Subroutines for idle mode (des)activation

*/

; converts minutes into preset value for timers
; in: address where timer preset value is stored in SRAM
.macro CALCULATE_TIMER
 lds a0, @0     ; load value (2 bytes) from SRAM
 lds a1, @0+1
 MUL5           ; multiply by 5
 COM2 a0,a1     ; inverse bits
 sts @0, a0     ; store value (2 bytes) to SRAM
 sts @0+1, a1
 .endmacro

; input: address at which timer value is stored
.macro PRESET_TIMER1
    lds w, @0+1 ; load value from SRAM
    out TCNT1H, w
    lds w, @0   ; load value from SRAM
    out TCNT1L, w
.endmacro


preset_timer0:
    lds a0, delay_time_timer0
    ADDI a0, 3
    out TCNT0, a0
    ret

timer_init:
    OUTI    DDRD, 0b00100000    ; configure portD as input for buttons / external interrupts, PD5 as output (overflow0 signal)
    OUTI    ASSR, (1<<AS0)      ; external clock 32'768 for timer 0
    OUTI    TCCR0, 7            ; prescaler 1024 -> overflow period 8s
    OUTI    TCCR1B, 7           ; external clock on T1 (PD6) for timer 1, rising edge
    OUTI    TIMSK, (1<<TOIE0)+(1<<TOIE1)    ; enable interrupt timer0 and timer1
    ret

idle_mode_init:
    in      a0, MCUCR
    sbr     a0, (1<<SE)     ; SE=1 in MCUCR enables sleep modes, SM0=0, SM1=0 -
```

```
      > idle mode
    out     MCUCR, a0
    ret

idle_mode_off:
    in      a0, MCUCR
    andi    a0, ~(1<<SE)        ; disenable sleep modes
    out     MCUCR, a0
    ret

; converts value of Timer1 (stored in TCNT1) into minutes and displays it on
  LCD screen
display_time:
    in      a0,TCNT1L
    in      a1,TCNT1H
    COM2    a0,a1
    ldi     b0, 0x05
    clr     b1
    call    div22       ; calculates a/b (both two register numbers), result
      stored in c
    PRINTF  LCD
    .db "time left=",FDEC2,c," min ",CR,0
    WAIT_MS 100
    ret
```

```
; file  i2cx.asm   target ATmega128L-4MHz-STK300
; purpose extended I2C (400 k bit/s), software emulation

; === definitions ===
.equ    SDA_port= PORTB
.equ    SDA_pin = SDA
.equ    SCL_port= PORTB
.equ    SCL_pin = SCL
/*.equ  SDA_port= PORTD
.equ    SDA_pin = 1
.equ    SCL_port= PORTD
.equ    SCL_pin = 0*/


; === macros ===

; ===============
; self made macros
;================

; call sub-routine with argument in a0
.macro CA_A0
    ldi a0, @1
    rcall @0
    .endmacro

; ===============
; premade macros
;================

; these macros control DDRx to simulate an open collector
; with external pull-up resistors

.macro  SCL0
    sbi SCL_port-1,SCL_pin  ; pull SCL low (output, port=0)
    .endmacro
.macro  SCL1
    cbi SCL_port-1,SCL_pin  ; release SCL (input, hi Z)
    .endmacro
.macro  SDA0
    sbi SDA_port-1,SDA_pin  ; pull SDA low (output, port=0)
    .endmacro
.macro  SDA1
    cbi SDA_port-1,SDA_pin  ; release SDA (input, hi Z)
    .endmacro

.macro  I2C_BIT_OUT ;bit
    sbi SCL_port-1,SCL_pin  ; pull SCL low (output, port=0)
    in  w,SDA_port-1        ; sample the SDA line
    bst a0,@0               ; store a0(bit) to T
    bld w,SDA_pin           ; load w(SDA) with T
    out SDA_port-1,w        ; transfer bit_x to SDA
    cbi SCL_port-1,SCL_pin  ; release SCL (input, hi Z)
```

```
    rjmp    PC+1            ; wait 2 cyles
    .endmacro

.macro  I2C_BIT_IN  ;bit
    sbi SCL_port-1,SCL_pin  ; DDRx=output   SCL=0
    cbi SDA_port-1,SDA_pin  ; release SDA (input, hi Z)
    cbi SCL_port-1,SCL_pin  ; DDRx=input    SCL=1
    nop                 ; wait 1 cycle
    in  w,SDA_port-2        ; PINx=PORTx-2
    bst w,SDA_pin           ; store bit read in T
    bld a0,@0               ; load a0(bit) from T
    .endmacro

; === routines ===
i2c_init:
    cbi SDA_port,  SDA_pin  ; PORTx=0 (for pull-down)
    cbi SCL_port,  SCL_pin  ; PORTx=0 (for pull-down)
    SDA1                    ; release SDA
    SCL1                    ; release SCL
    ret

i2c_rep_start:
; in:   a0 (byte to transmit)
    SCL0
    SDA1
    SCL1
i2c_start:
; in:   a0 (byte to transmit)
    SDA0
i2c_write:
    com a0                  ; invert a0
    I2C_BIT_OUT 7
    I2C_BIT_OUT 6
    I2C_BIT_OUT 5
    I2C_BIT_OUT 4
    I2C_BIT_OUT 3
    I2C_BIT_OUT 2
    I2C_BIT_OUT 1
    I2C_BIT_OUT 0
    com a0                  ; restore a0
i2c_ack_in:
    SCL0
    SDA1                    ; release SDA
    SCL1
    in  w,SDA_port-2        ; PINx=PORTx-2
    bst w,SDA_pin           ; store ACK into T
    ret

i2c_read:
; out:  a0 (byte read)
    I2C_BIT_IN 7
    I2C_BIT_IN 6
    I2C_BIT_IN 5
```

```
    I2C_BIT_IN 4
    I2C_BIT_IN 3
    I2C_BIT_IN 2
    I2C_BIT_IN 1
    I2C_BIT_IN 0
    ret

i2c_ack:
    SCL0
    SDA0
    SCL1
    ret

i2c_no_ack:
    SCL0
    SDA1
    SCL1
    ret

i2c_stop:
    SCL0
    SDA0
    SCL1
    SDA1                ; release again
    ret
```

```
; file  lcd.asm   target ATmega128L-4MHz-STK300
; purpose  LCD HD44780U library
; ATmega 128 and Atmel Studio 7.0 compliant

; === definitions ===
.equ    LCD_IR  = 0x8000    ; address LCD instruction reg
.equ    LCD_DR  = 0xc000    ; address LCD data register

; === subroutines ===
LCD_wr_ir:
; in    w (byte to write to LCD IR)
    lds u, LCD_IR       ; read IR to check busy flag  (bit7)
    JB1 u,7,LCD_wr_ir   ; Jump if Bit=1 (still busy)
    rcall   lcd_4us     ; delay to increment DRAM addr counter
    sts LCD_IR, w       ; store w in IR
    ret

lcd_4us:
    rcall   lcd_2us     ; recursive call
lcd_2us:
    nop                 ; rcall(3) + nop(1) + ret(4) = 8 cycles (2us)
    ret

LCD:
LCD_putc:
    JK  a0,CR,LCD_cr    ; Jump if a0=CR
    JK  a0,LF,LCD_lf    ; Jump if a0=LF
LCD_wr_dr:
; in    a0 (byte to write to LCD DR)
    lds w, LCD_IR       ; read IR to check busy flag  (bit7)
    JB1 w,7,LCD_wr_dr   ; Jump if Bit=1 (still busy)
    rcall   lcd_4us     ; delay to increment DRAM addr counter
    sts LCD_DR, a0      ; store a0 in DR
    ret

LCD_clear:      JW  LCD_wr_ir, 0b00000001       ; clear display
LCD_home:       JW  LCD_wr_ir, 0b00000010       ; return home
LCD_cursor_left:    JW  LCD_wr_ir, 0b00010000   ; move cursor to left
LCD_cursor_right:   JW  LCD_wr_ir, 0b00010100   ; move cursor to right
LCD_display_left:   JW  LCD_wr_ir, 0b00011000   ; shifts display to left
LCD_display_right:  JW  LCD_wr_ir, 0b00011100   ; shifts display to right
LCD_blink_on:       JW  LCD_wr_ir, 0b00001101   ; Display=1,Cursor=0,Blink=1
LCD_blink_off:      JW  LCD_wr_ir, 0b00001100   ; Display=1,Cursor=0,Blink=0
LCD_cursor_on:      JW  LCD_wr_ir, 0b00001110   ; Display=1,Cursor=1,Blink=0
LCD_cursor_off:     JW  LCD_wr_ir, 0b00001100   ; Display=1,Cursor=0,Blink=0

LCD_init:
    in  w,MCUCR                 ; enable access to ext. SRAM
    sbr w,(1<<SRE)+(1<<SRW10)
    out MCUCR,w
    CW  LCD_wr_ir, 0b00000001   ; clear display
    CW  LCD_wr_ir, 0b00000110   ; entry mode set (Inc=1, Shift=0)
    CW  LCD_wr_ir, 0b00001100   ; Display=1,Cursor=0,Blink=0
```

```
    CW  LCD_wr_ir, 0b00111000   ; 8bits=1, 2lines=1, 5x8dots=0
    ret

LCD_pos:
; in    a0 = position (0x00..0x0f first line, 0x40..0x4f second line)
    mov w,a0
    ori w,0b10000000
    rjmp    LCD_wr_ir

LCD_cr:
; moving the cursor to the beginning of the line (carriage return)
    lds w, LCD_IR           ; read IR to check busy flag  (bit7)
    JB1 w,7,LCD_cr          ; Jump if Bit=1 (still busy)
    andi    w,0b01000000    ; keep bit6 (begin of line 1/2)
    ori w,0b10000000        ; write address command
    rcall   lcd_4us         ; delay to increment DRAM addr counter
    sts LCD_IR,w            ; store in IR
    ret

LCD_lf:
; moving the cursor to the beginning of the line 2 (line feed)
    push    a0              ; safeguard a0
    ldi a0,$40              ; load position $40 (begin of line 2)
    rcall   LCD_pos         ; set cursor position
    pop a0                  ; restore a0
    ret
```

```
; file: macros.asm   target ATmega128L-4MHz-STK300
; purpose library, general-purpose macros
; author (c) R.Holzer (adapted MICRO210/EE208 A.Schmid)
; v2019.01 20180820 AxS


; ==============
; self made macros
; ==============
.macro LSLCA ; logical shift left with carry reg a0 and a1
    lsl a0
    brcc PC + 3
    lsl a1
    subi a1, -1
.endmacro

.macro  MUL5    ; works only with a0 and a1. uses a3 as intermediate
    mov a3, a1
    mov w, a0
    LSLCA
    LSLCA
    add a0, w
    adc a1, a3
.endmacro

.macro  MUL10   ; works only with a0 and a1. uses a3 as intermediate
    mov a3, a1
    mov w, a0
    LSLCA
    LSLCA
    LSLCA
    add a0, w
    adc a1, a3
    add a0, w
    adc a1, a3
.endmacro


; ==============
;   pointers
; ==============

; --- loading an immediate into a pointer XYZ,SP ---
.macro  LDIX    ; sram
    ldi xl, low(@0)
    ldi xh,high(@0)
    .endmacro
.macro  LDIY    ; sram
    ldi yl, low(@0)
    ldi yh,high(@0)
    .endmacro
.macro  LDIZ    ; sram
    ldi zl, low(@0)
```

```
    ldi zh,high(@0)

    .endmacro
.macro  LDZD    ; sram, reg ; sram+reg -> Z
    mov zl,@1
    clr zh
    subi    zl, low(-@0)
    sbci    zh,high(-@0)
    .endmacro
.macro  LDSP    ; sram
    ldi r16, low(@0)
    out spl,r16
    ldi r16,high(@0)
    out sph,r16
    .endmacro

; --- load/store SRAM addr into pointer XYZ ---
.macro  LDSX    ; sram
    lds xl,@0
    lds xh,@0+1
    .endmacro
.macro  LDSY    ; sram
    lds yl,@0
    lds yh,@0+1
    .endmacro
.macro  LDSZ    ; sram
    lds zl,@0
    lds zh,@0+1
    .endmacro
.macro  STSX    ; sram
    sts @0,  xl
    sts @0+1,xh
    .endmacro
.macro  STSY    ; sram
    sts @0,  yl
    sts @0+1,yh
    .endmacro
.macro  STSZ    ; sram
    sts @0,  zl
    sts @0+1,zh
    .endmacro

; --- push/pop pointer XYZ ---
.macro  PUSHX           ; push X
    push    xl
    push    xh
    .endmacro
.macro  POPX            ; pop X
    pop xh
    pop xl
    .endmacro

.macro  PUSHY           ; push Y
```

```
    push    yl
    push    yh
    .endmacro
.macro  POPY            ; pop Y
    pop yh
    pop yl
    .endmacro

.macro  PUSHZ           ; push Z
    push    zl
    push    zh
    .endmacro
.macro  POPZ            ; pop Z
    pop zh
    pop zl
    .endmacro

; --- multiply/divide Z ---
.macro  MUL2Z           ; multiply Z by 2
    lsl zl
    rol zh
    .endmacro
.macro  DIV2Z           ; divide Z by 2
    lsr zh
    ror zl
    .endmacro

; --- add register to pointer XYZ ---
.macro  ADDX    ;reg        ; x <- y+reg
    add xl,@0
    brcc    PC+2
    subi    xh,-1       ; add carry
    .endmacro
.macro  ADDY    ;reg        ; y <- y+reg
    add yl,@0
    brcc    PC+2
    subi    yh,-1       ; add carry
    .endmacro
.macro  ADDZ    ;reg        ; z <- z+reg
    add zl,@0
    brcc    PC+2
    subi    zh,-1       ; add carry
    .endmacro

; ====================
;   miscellaneous
; ====================

; --- output/store (regular I/O space) immediate value ---
.macro  OUTI    ; port,k    output immediate value to port
    ldi w,@1
    out @0,w
    .endmacro
```

```
; --- output/store (extended I/O space) immediate value ---
.macro OUTEI    ; port,k    output immediate value to port
    ldi w,@1
    sts @0,w
    .endmacro

; --- add immediate value ---
.macro  ADDI
    subi    @0,-@1
    .endmacro
.macro  ADCI
    sbci    @0,-@1
    .endmacro

; --- inc/dec with range limitation ---
.macro  INC_LIM ; reg,limit
    cpi @0,@1
    brlo    PC+3
    ldi @0,@1
    rjmp    PC+2
    inc @0
    .endmacro

.macro  DEC_LIM ; reg,limit
    cpi @0,@1
    breq    PC+5
    brlo    PC+3
    dec @0
    rjmp    PC+2
    ldi @0,@1
    .endmacro

; --- inc/dec with cyclic range ---
.macro  INC_CYC ; reg,low,high
    cpi @0,@2
    brsh    _low    ; reg>=high then reg=low
    cpi @0,@1
    brlo    _low    ; reg< low  then reg=low
    inc @0
    rjmp    _done
_low:   ldi @0,@1
_done:
    .endmacro

.macro  DEC_CYC ; reg,low,high
    cpi @0,@1
    breq    _high   ; reg=low then reg=high
    brlo    _high   ; reg<low then reg=high
    dec @0
    cpi @0,@2
    brsh    _high   ; reg>=high then high
    rjmp    _done
```

```
_high:  ldi @0,@2
_done:
    .endmacro

.macro  INCDEC  ;port,b1,b2,reg,low,high
    sbic    @0,@1
    rjmp    PC+6

    cpi @3,@5
    brlo    PC+3
    ldi @3,@4
    rjmp    PC+2
    inc @3

    sbic    @0,@2
    rjmp    PC+7

    cpi @3,@4
    breq    PC+5
    brlo    PC+3
    dec @3
    rjmp    PC+2
    ldi @3,@5
    .endmacro

; --- wait loops ---
; wait 10...196608 cycles
.macro  WAIT_C  ; k
    ldi w,  low((@0-7)/3)
    mov u,w         ; u=LSB
    ldi w,high((@0-7)/3)+1  ; w=MSB
    dec u
    brne    PC-1
    dec u
    dec w
    brne    PC-4
    .endmacro

; wait micro-seconds (us)
; us = x*3*1000'000/clock)  ==> x=us*clock/3000'000
.macro  WAIT_US ; k
    ldi w, low((clock/1000*@0/3000)-1)
    mov u,w
    ldi w,high((clock/1000*@0/3000)-1)+1 ; set up: 3 cyles
    dec u
    brne    PC-1        ; inner loop: 3 cycles
    dec u       ; adjustment for outer loop
    dec w
    brne    PC-4
    .endmacro

; wait mili-seconds (ms)
.macro  WAIT_MS ; k
```

```
    ldi w, low(@0)
    mov u,w      ; u = LSB
    ldi w,high(@0)+1    ; w = MSB
wait_ms:
    push    w       ; wait 1000 usec
    push    u
    ldi w, low((clock/3000)-5)
    mov u,w
    ldi w,high((clock/3000)-5)+1
    dec u
    brne    PC-1        ; inner loop: 3 cycles
    dec u       ; adjustment for outer loop
    dec w
    brne    PC-4
    pop u
    pop w

    dec u
    brne    wait_ms
    dec w
    brne    wait_ms
    .endmacro

; --- conditional jumps/calls ---
.macro  JC0         ; jump if carry=0
    brcs    PC+2
    rjmp    @0
    .endmacro
.macro  JC1         ; jump if carry=1
    brcc    PC+2
    rjmp    @0
    .endmacro

.macro  JK  ; reg,k,addr    ; jump if reg=k
    cpi @0,@1
    breq    @2
    .endmacro
.macro  _JK ; reg,k,addr    ; jump if reg=k
    cpi @0,@1
    brne    PC+2
    rjmp    @2
    .endmacro
.macro  JNK ; reg,k,addr    ; jump if not(reg=k)
    cpi @0,@1
    brne    @2
    .endmacro

.macro  CK  ; reg,k,addr    ; call if reg=k
    cpi @0,@1
    brne    PC+2
    rcall   @2
    .endmacro
.macro  CNK ; reg,k,addr    ; call if not(reg=k)
```

```
    cpi @0,@1
    breq    PC+2
    rcall   @2
    .endmacro

.macro  JSK ; sram,k,addr   ; jump if sram=k
    lds w,@0
    cpi w,@1
    breq    @2
    .endmacro
.macro  JSNK    ; sram,k,addr   ; jump if not(sram=k)
    lds w,@0
    cpi w,@1
    brne    @2
    .endmacro

; --- loops ---
.macro  DJNZ    ; reg,addr  ; decr and jump if not zero
    dec @0
    brne    @1
    .endmacro
.macro  DJNK    ; reg,k,addr    ; decr and jump if not k
    dec @0
    cpi @0,@1
    brne    @2
    .endmacro

.macro  IJNZ    ; reg,addr  ; inc and jump if not zero
    inc @0
    brne    @1
    .endmacro
.macro  IJNK    ; reg,k,addr    ; inc and jump if not k
    inc @0
    cpi @0,@1
    brne    @2
    .endmacro
.macro  _IJNK   ; reg,k,addr    ; inc and jump if not k
    inc @0
    ldi w,@1
    cp  @0,w
    brne    @2
    .endmacro

.macro  ISJNK   ; sram,k,addr   ; inc sram and jump if not k
    lds w,@0
    inc w
    sts @0,w
    cpi w,@1
    brne    @2
    .endmacro
.macro  _ISJNK  ; sram,k,addr   ; inc sram and jump if not k
    lds w,@0
    inc w
```

```
    sts @0,w
    cpi w,@1
    breq    PC+2
    rjmp    @2
    .endmacro

.macro  DSJNK   ; sram,k,addr   ; dec sram and jump if not k
    lds w,@0
    dec w
    sts @0,w
    cpi w,@1
    brne    @2
    .endmacro

; --- table lookup ---
.macro  LOOKUP  ;reg, index,tbl
    push    ZL
    push    ZH
    mov zl,@1       ; move index into z
    clr zh
    subi    zl, low(-2*@2)  ; add base address of table
    sbci    zh,high(-2*@2)
    lpm         ; load program memory (into r0)
    mov @0,r0
    pop ZH
    pop ZL
    .endmacro

.macro  LOOKUP2 ;r1,r0, index,tbl
    mov zl,@2       ; move index into z
    clr zh
    lsl zl      ; multiply by 2
    rol zh
    subi    zl, low(-2*@3)  ; add base address of table
    sbci    zh,high(-2*@3)
    lpm         ; get LSB byte
    mov w,r0        ; temporary store LSB in w
    adiw    zl,1        ; increment Z
    lpm         ; get MSB byte
    mov @0,r0       ; mov MSB to res1
    mov @1,w        ; mov LSB to res0
    .endmacro

.macro  LOOKUP4 ;r3,r2,r1,r0, index,tbl
    mov zl,@4       ; move index into z
    clr zh
    lsl zl      ; multiply by 2
    rol zh
    lsl zl      ; multiply by 2
    rol zh
    subi    zl, low(-2*@5)  ; add base address of table
    sbci    zh,high(-2*@5)
    lpm
```

```
    mov @1,r0       ; load high word LSB
    adiw    zl,1
    lpm
    mov @0,r0       ; load high word MSB
    adiw    zl,1
    lpm
    mov @3,r0       ; load low word LSB
    adiw    zl,1
    lpm
    mov @2,r0       ; load low word MSB
    .endmacro

.macro  LOOKDOWN ;reg,index,tbl
    ldi ZL, low(2*@2)   ; load table address
    ldi ZH,high(2*@2)
    clr @1
loop:   lpm
    cp  r0,@0
    breq    found
    inc @1
    adiw    ZL,1
    tst r0
    breq    notfound
    rjmp    loop
notfound:
    ldi @1,-1
found:
    .endmacro

; --- branch table ---
.macro  C_TBL   ; reg,tbl
    ldi ZL, low(2*@1)
    ldi ZH,high(2*@1)
    lsl @0
    add ZL,@0
    brcc    PC+2
    inc ZH
    lpm
    push    r0
    lpm
    mov zh,r0
    pop zl
    icall
    .endmacro
.macro  J_TBL   ; reg,tbl
    ldi ZL, low(2*@1)
    ldi ZH,high(2*@1)
    lsl @0
    add ZL,@0
    brcc    PC+2
    inc ZH
    lpm
    push    r0
```

```
    lpm
    mov zh,r0
    pop zl
    ijmp
    .endmacro

.macro  BRANCH  ; reg       ; branching using the stack
    ldi w, low(tbl)
    add w,@0
    push    w
    ldi w,high(tbl)
    brcc    PC+2
    inc w
    push    w
    ret
tbl:
    .endmacro

; --- multiply/division ---
.macro  DIV2    ; reg
    lsr @0
    .endmacro
.macro  DIV4    ; reg
    lsr @0
    lsr @0
    .endmacro
.macro  DIV8    ; reg
    lsr @0
    lsr @0
    lsr @0
    .endmacro

.macro  MUL2    ; reg
    lsl @0
    .endmacro
.macro  MUL4    ; reg
    lsl @0
    lsl @0
    .endmacro
.macro  MUL8    ; reg
    lsl @0
    lsl @0
    lsl @0
    .endmacro

; ====================================
;   extending existing instructios
; ====================================

; --- immediate ops with r0..r15 ---
.macro  _ADDI
    ldi w,@1
    add @0,w
```

```
    .endmacro
.macro  _ADCI
    ldi w,@1
    adc @0,w
    .endmacro
.macro  _SUBI
    ldi w,@1
    sub @0,w
    .endmacro
.macro  _SBCI
    ldi w,@1
    sbc @0,w
    .endmacro
.macro  _ANDI
    ldi w,@1
    and @0,w
    .endmacro
.macro  _ORI
    ldi w,@1
    or  @0,w
    .endmacro
.macro  _EORI
    ldi w,@1
    eor @0,w
    .endmacro
.macro  _SBR
    ldi w,@1
    or  @0,w
    .endmacro
.macro  _CBR
    ldi w,~@1
    and @0,w
    .endmacro
.macro  _CPI
    ldi w,@1
    cp  @0,w
    .endmacro
.macro  _LDI
    ldi w,@1
    mov @0,w
    .endmacro

; --- bit access for port p32..p63 ---
.macro  _SBI
    in  w,@0
    ori w,1<<@1
    out @0,w
    .endmacro
.macro  _CBI
    in  w,@0
    andi    w,~(1<<@1)
    out @0,w
    .endmacro
```

```
; --- extending branch distance to +/-2k ---
.macro  _BREQ
    brne    PC+2
    rjmp    @0
    .endmacro
.macro  _BRNE
    breq    PC+2
    rjmp    @0
    .endmacro
.macro  _BRCS
    brcc    PC+2
    rjmp    @0
    .endmacro
.macro  _BRCC
    brcs    PC+2
    rjmp    @0
    .endmacro
.macro  _BRSH
    brlo    PC+2
    rjmp    @0
    .endmacro
.macro  _BRLO
    brsh    PC+2
    rjmp    @0
    .endmacro
.macro  _BRMI
    brpl    PC+2
    rjmp    @0
    .endmacro
.macro  _BRPL
    brmi    PC+2
    rjmp    @0
    .endmacro
.macro  _BRGE
    brlt    PC+2
    rjmp    @0
    .endmacro
.macro  _BRLT
    brge    PC+2
    rjmp    @0
    .endmacro
.macro  _BRHS
    brhc    PC+2
    rjmp    @0
    .endmacro
.macro  _BRHC
    brhs    PC+2
    rjmp    @0
    .endmacro
.macro  _BRTS
    brtc    PC+2
    rjmp    @0
```

```
    .endmacro
.macro  _BRTC
    brts    PC+2
    rjmp    @0
    .endmacro
.macro  _BRVS
    brvc    PC+2
    rjmp    @0
    .endmacro
.macro  _BRVC
    brvs    PC+2
    rjmp    @0
    .endmacro
.macro  _BRIE
    brid    PC+2
    rjmp    @0
    .endmacro
.macro  _BRID
    brie    PC+2
    rjmp    @0
    .endmacro

; ====================
;   bit operations
; ====================

; --- moving bits ---
.macro  MOVB    ; reg1,b1, reg2,b2  ; reg1,bit1 <- reg2,bit2
    bst @2,@3
    bld @0,@1
    .endmacro
.macro  OUTB    ; port1,b1, reg2,b2 ; port1,bit1 <- reg2,bit2
    sbrs    @2,@3
    cbi @0,@1
    sbrc    @2,@3
    sbi @0,@1
    .endmacro
.macro  INB ; reg1,b1, port2,b2 ; reg1,bit1 <- port2,bit2
    sbis    @2,@3
    cbr @0,1<<@1
    sbic    @2,@3
    sbr @0,1<<@1
    .endmacro

.macro  Z2C             ; zero to carry
    sec
    breq    PC+2    ; (Z=1)
    clc
    .endmacro
.macro  Z2INVC              ; zero to inverse carry
    sec
    brne    PC+2    ; (Z=0)
    clc
```

```
    .endmacro

.macro  C2Z             ; carry to zero
    sez
    brcs    PC+2    ; (C=1)
    clz
    .endmacro

.macro  B2C ; reg,b         ; bit to carry
    sbrc    @0,@1
    sec
    sbrs    @0,@1
    clc
    .endmacro
.macro  C2B ; reg,b         ; carry to bit
    brcc    PC+2
    sbr @0,(1<<@1)
    brcs    PC+2
    cbr @0,(1<<@1)
    .endmacro
.macro  P2C ; port,b        ; port to carry
    sbic    @0,@1
    sec
    sbis    @0,@1
    clc
    .endmacro
.macro  C2P ; port,b        ; carry to port
    brcc    PC+2
    sbi @0,@1
    brcs    PC+2
    cbi @0,@1
    .endmacro

; --- inverting bits ---
.macro  INVB    ; reg,bit       ; inverse reg,bit
    ldi w,(1<<@1)
    eor @0,w
    .endmacro
.macro  INVP    ; port,bit      ; inverse port,bit
    sbis    @0,@1
    rjmp    PC+3
    cbi @0,@1
    rjmp    PC+2
    sbi @0,@1
    .endmacro
.macro  INVC                ; inverse carry
    brcs    PC+3
    sec
    rjmp    PC+2
    clc
    .endmacro

; --- setting a single bit ---
```

```
.macro  SETBIT  ; reg(0..7)
; in    reg (0..7)
; out   reg with bit (0..7) set to 1.
; 0=00000001
; 1=00000010
; ...
; 7=10000000
    mov w,@0
    clr @0
    inc @0
    andi    w,0b111
    breq    PC+4
    lsl @0
    dec w
    brne    PC-2
    .endmacro

; --- logical operations with masks ---
.macro  MOVMSK  ; reg1,reg2,mask    ; reg1 <- reg2 (mask)
    ldi w,~@2
    and @0,w
    ldi w,@2
    and @1,w
    or  @0,@1
    .endmacro
.macro  ANDMSK  ; reg1,reg2,mask    ; reg1 <- ret 1 AND reg2 (mask)
    mov w,@1
    ori w,~@2
    and @0,w
    .endmacro
.macro  ORMSK   ; reg1,reg2,mask    ; reg1 <- ret 1 AND reg2 (mask)
    mov w,@1
    andi    w,@2
    or  @0,w
    .endmacro

; --- logical operations on bits ---
.macro  ANDB    ; r1,b1, r2,b2, r3,b3   ; reg1,b1 <- reg2,b2 AND reg3,b3
    set
    sbrs    @4,@5
    clt
    sbrs    @2,@3
    clt
    bld @0,@1
    .endmacro
.macro  ORB ; r1,b1, r2,b2, r3,b3   ; reg1.b1 <- reg2.b2 OR reg3.b3
    clt
    sbrc    @4,@5
    set
    sbrc    @2,@3
    set
    bld @0,@1
    .endmacro
```

```
.macro  EORB    ; r1,b1, r2,b2, r3,b3   ; reg1.b1 <- reg2.b2 XOR reg3.b3
    sbrc    @4,@5
    rjmp    f1
f0: bst @2,@3
    rjmp    PC+4
f1: set
    sbrc    @0,@1
    clt
    bld @0,@0
    .endmacro

; --- operations based on register bits ---
.macro  FB0 ; reg,bit       ; bit=0
    cbr @0,1<<@1
    .endmacro
.macro  FB1 ; reg,bit       ; bit=1
    sbr @0,1<<@1
    .endmacro
.macro  _FB0    ; reg,bit       ; bit=0
    ldi w,~(1<<@1)
    and @0,w
    .endmacro
.macro  _FB1    ; reg,bit       ; bit=1
    ldi w,1<<@1
    or  @0,w
    .endmacro
.macro  SB0 ; reg,bit,addr      ; skip if bit=0
    sbrc    @0,@1
    .endmacro
.macro  SB1 ; reg,bit,addr      ; skip if bit=1
    sbrs    @0,@1
    .endmacro
.macro  JB0 ; reg,bit,addr      ; jump if bit=0
    sbrs    @0,@1
    rjmp    @2
    .endmacro
.macro  JB1 ; reg,bit,addr      ; jump if bit=1
    sbrc    @0,@1
    rjmp    @2
    .endmacro
.macro  CB0 ; reg,bit,addr      ; call if bit=0
    sbrs    @0,@1
    rcall   @2
    .endmacro
.macro  CB1 ; reg,bit,addr      ; call if bit=1
    sbrc    @0,@1
    rcall   @2
    .endmacro
.macro  WB0 ; reg,bit       ; wait if bit=0
    sbrs    @0,@1
    rjmp    PC-1
    .endmacro
.macro  WB1 ; reg,bit       ; wait if bit=1
```

```
    sbrc    @0,@1
    rjmp    PC-1
    .endmacro
.macro  RB0 ; reg,bit       ; return if bit=0
    sbrs    @0,@1
    ret
    .endmacro
.macro  RB1 ; reg,bit       ; return if bit=1
    sbrc    @0,@1
    ret
    .endmacro

; wait if bit=0 with timeout
; if timeout (in units of 5 cyc) then jump to addr
.macro  WB0T    ; reg,bit,timeout,addr
    ldi w,@2+1
    dec w   ; 1 cyc
    breq    @3  ; 1 cyc
    sbrs    @0,@1   ; 1 cyc
    rjmp    PC-3    ; 2 cyc = 5 cycles
    .endmacro

; wait if bit=1 with timeout
; if timeout (in units of 5 cyc) then jump to addr
.macro  WB1T    ; reg,bit,timeout,addr
    ldi w,@2+1
    dec w   ; 1 cyc
    breq    @3  ; 1 cyc
    sbrc    @0,@1   ; 1 cyc
    rjmp    PC-3    ; 2 cyc = 5 cycles
    .endmacro

; --- operations based on port bits ---
.macro  P0  ; port,bit      ; port=0
    cbi @0,@1
    .endmacro
.macro  P1  ; port,bit      ; port=1
    sbi @0,@1
    .endmacro
.macro  SP0 ; port,bit      ; skip if port=0
    sbic    @0,@1
    .endmacro
.macro  SP1 ; port,bit      ; skip if port=1
    sbis    @0,@1
    .endmacro
.macro  JP0 ; port,bit,addr     ; jump if port=0
    sbis    @0,@1
    rjmp    @2
    .endmacro
.macro  JP1 ; port,bit,addr     ; jump if port=1
    sbic    @0,@1
    rjmp    @2
    .endmacro
```

```
.macro  CP0 ; port,bit,addr     ; call if port=0
    sbis    @0,@1
    rcall   @2
    .endmacro
.macro  CP1 ; port,bit,addr     ; call if port=1
    sbic    @0,@1
    rcall   @2
    .endmacro
.macro  WP0 ; port,bit      ; wait if port=0
    sbis    @0,@1
    rjmp    PC-1
    .endmacro
.macro  WP1 ; port,bit      ; wait if port=1
    sbic    @0,@1
    rjmp    PC-1
    .endmacro
.macro  RP0 ; port,bit      ; return if port=0
    sbis    @0,@1
    ret
    .endmacro
.macro  RP1 ; port,bit      ; return if port=1
    sbic    @0,@1
    ret
    .endmacro

; wait if port=0 with timeout
; if timeout (in units of 5 cyc) then jump to addr
.macro  WP0T    ; port,bit,timeout,addr
    ldi w,@2+1
    dec w   ; 1 cyc
    breq    @3  ; 1 cyc
    sbis    @0,@1   ; 1 cyc
    rjmp    PC-3    ; 2 cyc = 5 cycles
    .endmacro

; wait if port=1 with timeout
; if timeout (in units of 5 cyc) then jump to addr
.macro  WP1T    ; port,bit,timeout,addr
    ldi w,@2+1
    dec w   ; 1 cyc
    breq    @3  ; 1 cyc
    sbic    @0,@1   ; 1 cyc
    rjmp    PC-3    ; 2 cyc = 5 cycles
    .endmacro

; ============================
;   multi-byte operations
; ============================

.macro  SWAP4           ; swap 2 variables
    mov w ,@0
    mov @0,@4
    mov @4,w
```

```
    mov w ,@1
    mov @1,@5
    mov @5,w
    mov w ,@2
    mov @2,@6
    mov @6,w
    mov w ,@3
    mov @3,@7
    mov @7,w
    .endmacro
.macro  SWAP3
    mov w ,@0
    mov @0,@3
    mov @3,w
    mov w ,@1
    mov @1,@4
    mov @4,w
    mov w ,@2
    mov @2,@5
    mov @5,w
    .endmacro
.macro  SWAP2
    mov w ,@0
    mov @0,@2
    mov @2,w
    mov w ,@1
    mov @1,@3
    mov @3,w
    .endmacro
.macro  SWAP1
    mov w ,@0
    mov @0,@1
    mov @1,w
    .endmacro

.macro  LDX4    ;r..r0      ; load from (x+)
    ld  @3,x+
    ld  @2,x+
    ld  @1,x+
    ld  @0,x+
    .endmacro
.macro  LDX3    ;r..r0
    ld  @2,x+
    ld  @1,x+
    ld  @0,x+
    .endmacro
.macro  LDX2    ;r..r0
    ld  @1,x+
    ld  @0,x+
    .endmacro

.macro  LDY4    ;r..r0      ; load from (y+)
    ld  @3,y+
```

```
    ld  @2,y+
    ld  @1,y+
    ld  @0,y+
    .endmacro
.macro  LDY3    ;r..r0
    ld  @2,y+
    ld  @1,y+
    ld  @0,y+
    .endmacro
.macro  LDY2    ;r..r0
    ld  @1,y+
    ld  @0,y+
    .endmacro

.macro  LDZ4    ;r..r0      ; load from (z+)
    ld  @3,z+
    ld  @2,z+
    ld  @1,z+
    ld  @0,z+
    .endmacro
.macro  LDZ3    ;r..r0
    ld  @2,z+
    ld  @1,z+
    ld  @0,z+
    .endmacro
.macro  LDZ2    ;r..r0
    ld  @1,z+
    ld  @0,z+
    .endmacro

.macro  STX4    ;r..r0      ; store to (x+)
    st  x+,@3
    st  x+,@2
    st  x+,@1
    st  x+,@0
    .endmacro
.macro  STX3    ;r..r0
    st  x+,@2
    st  x+,@1
    st  x+,@0
    .endmacro
.macro  STX2    ;r..r0
    st  x+,@1
    st  x+,@0
    .endmacro

.macro  STY4    ;r..r0      ; store to (y+)
    st  y+,@3
    st  y+,@2
    st  y+,@1
    st  y+,@0
    .endmacro
.macro  STY3    ;r..r0
```

```
    st  y+,@2
    st  y+,@1
    st  y+,@0
    .endmacro
.macro  STY2    ;r..r0
    st  y+,@1
    st  y+,@0
    .endmacro

.macro  STZ4    ;r..r0      ; store to (z+)
    st  z+,@3
    st  z+,@2
    st  z+,@1
    st  z+,@0
    .endmacro
.macro  STZ3    ;r..r0
    st  z+,@2
    st  z+,@1
    st  z+,@0
    .endmacro
.macro  STZ2    ;r..r0
    st  z+,@1
    st  z+,@0
    .endmacro

.macro  STI4    ;addr,k     ; store immediate
    ldi w,  low(@1)
    sts @0+0,w
    ldi w, high(@1)
    sts @0+1,w
    ldi w,byte3(@1)
    sts @0+2,w
    ldi w,byte4(@1)
    sts @0+3,w
    .endmacro
.macro  STI3    ;addr,k
    ldi w,  low(@1)
    sts @0+0,w
    ldi w, high(@1)
    sts @0+1,w
    ldi w,byte3(@1)
    sts @0+2,w
    .endmacro
.macro  STI2    ;addr,k
    ldi w,  low(@1)
    sts @0+0,w
    ldi w, high(@1)
    sts @0+1,w
    .endmacro
.macro  STI ;addr,k
    ldi w,@1
    sts @0,w
    .endmacro
```

```
.macro  INC4            ; increment
    ldi w,0xff
    sub @3,w
    sbc @2,w
    sbc @1,w
    sbc @0,w
    .endmacro
.macro  INC3
    ldi w,0xff
    sub @2,w
    sbc @1,w
    sbc @0,w
    .endmacro
.macro  INC2
    ldi w,0xff
    sub @1,w
    sbc @0,w
    .endmacro

.macro  DEC4            ; decrement
    ldi w,0xff
    add @3,w
    adc @2,w
    adc @1,w
    adc @0,w
    .endmacro
.macro  DEC3
    ldi w,0xff
    add @2,w
    adc @1,w
    adc @0,w
    .endmacro
.macro  DEC2
    ldi w,0xff
    add @1,w
    adc @0,w
    .endmacro

.macro  CLR9            ; clear (also clears the carry)
    sub @0,@0
    clr @1
    clr @2
    clr @3
    clr @4
    clr @5
    clr @6
    clr @7
    clr @8
    .endmacro
.macro  CLR8
    sub @0,@0
    clr @1
```

```
    clr @2
    clr @3
    clr @4
    clr @5
    clr @6
    clr @7
    .endmacro
.macro  CLR7
    sub @0,@0
    clr @1
    clr @2
    clr @3
    clr @4
    clr @5
    clr @6
    .endmacro
.macro  CLR6
    sub @0,@0
    clr @1
    clr @2
    clr @3
    clr @4
    clr @5
    .endmacro
.macro  CLR5
    sub @0,@0
    clr @1
    clr @2
    clr @3
    clr @4
    .endmacro
.macro  CLR4
    sub @0,@0
    clr @1
    clr @2
    clr @3
    .endmacro
.macro  CLR3
    sub @0,@0
    clr @1
    clr @2
    .endmacro
.macro  CLR2
    sub @0,@0
    clr @1
    .endmacro

.macro  COM4            ; one's complement
    com @0
    com @1
    com @2
    com @3
    .endmacro
```

```
.macro  COM3
    com @0
    com @1
    com @2
    .endmacro
.macro  COM2
    com @0
    com @1
    .endmacro

.macro  NEG4            ; negation (two's complement)
    com @0
    com @1
    com @2
    com @3
    ldi w,0xff
    sub @3,w
    sbc @2,w
    sbc @1,w
    sbc @0,w
    .endmacro
.macro  NEG3
    com @0
    com @1
    com @2
    ldi w,0xff
    sub @2,w
    sbc @1,w
    sbc @0,w
    .endmacro
.macro  NEG2
    com @0
    com @1
    ldi w,0xff
    sub @1,w
    sbc @0,w
    .endmacro

.macro  LDI4    ; r..r0, k  ; load immediate
    ldi @3,  low(@4)
    ldi @2, high(@4)
    ldi @1,byte3(@4)
    ldi @0,byte4(@4)
    .endmacro
.macro  LDI3
    ldi @2,  low(@3)
    ldi @1, high(@3)
    ldi @0,byte3(@3)
    .endmacro
.macro  LDI2
    ldi @1,  low(@2)
    ldi @0, high(@2)
    .endmacro
```

```
.macro  LDS4            ; load direct from SRAM
    lds @3,@4
    lds @2,@4+1
    lds @1,@4+2
    lds @0,@4+3
    .endmacro
.macro  LDS3
    lds @2,@3
    lds @1,@3+1
    lds @0,@3+2
    .endmacro
.macro  LDS2
    lds @1,@2
    lds @0,@2+1
    .endmacro

.macro  STS4            ; store direct to SRAM
    sts @0+0,@4
    sts @0+1,@3
    sts @0+2,@2
    sts @0+3,@1
    .endmacro
.macro  STS3
    sts @0+0,@3
    sts @0+1,@2
    sts @0+2,@1
    .endmacro
.macro  STS2
    sts @0+0,@2
    sts @0+1,@1
    .endmacro

.macro  STDZ4   ; d, r3,r2,r1,r0
    std z+@0+0,@4
    std z+@0+1,@3
    std z+@0+2,@2
    std z+@0+3,@1
    .endmacro
.macro  STDZ3   ; d, r2,r1,r0
    std z+@0+0,@3
    std z+@0+1,@2
    std z+@0+2,@1
    .endmacro
.macro  STDZ2   ; d, r1,r0
    std z+@0+0,@2
    std z+@0+1,@1
    .endmacro

.macro  LPM4            ; load program memory
    lpm
    mov @3,r0
    adiw    zl,1
```

```
    lpm
    mov @2,r0
    adiw    zl,1
    lpm
    mov @1,r0
    adiw    zl,1
    lpm
    mov @0,r0
    adiw    zl,1
    .endmacro
.macro  LPM3
    lpm
    mov @2,r0
    adiw    zl,1
    lpm
    mov @1,r0
    adiw    zl,1
    lpm
    mov @0,r0
    adiw    zl,1
    .endmacro
.macro  LPM2
    lpm
    mov @1,r0
    adiw    zl,1
    lpm
    mov @0,r0
    adiw    zl,1
    .endmacro
.macro  LPM1
    lpm
    mov @0,r0
    adiw    zl,1
    .endmacro

.macro  MOV4            ; move between registers
    mov @3,@7
    mov @2,@6
    mov @1,@5
    mov @0,@4
    .endmacro
.macro  MOV3
    mov @2,@5
    mov @1,@4
    mov @0,@3
    .endmacro
.macro  MOV2
    mov @1,@3
    mov @0,@2
    .endmacro

.macro  ADD4            ; add
    add @3,@7
```

```
    adc @2,@6
    adc @1,@5
    adc @0,@4
    .endmacro
.macro  ADD3
    add @2,@5
    adc @1,@4
    adc @0,@3
    .endmacro
.macro  ADD2
    add @1,@3
    adc @0,@2
    .endmacro

.macro  SUB4            ; subtract
    sub @3,@7
    sbc @2,@6
    sbc @1,@5
    sbc @0,@4
    .endmacro
.macro  SUB3
    sub @2,@5
    sbc @1,@4
    sbc @0,@3
    .endmacro
.macro  SUB2
    sub @1,@3
    sbc @0,@2
    .endmacro

.macro  CP4         ; compare
    cp  @3,@7
    cpc @2,@6
    cpc @1,@5
    cpc @0,@4
    .endmacro
.macro  CP3
    cp  @2,@5
    cpc @1,@4
    cpc @0,@3
    .endmacro
.macro  CP2
    cp  @1,@3
    cpc @0,@2
    .endmacro

.macro  TST4            ; test
    clr w
    cp  @3,w
    cpc @2,w
    cpc @1,w
    cpc @0,w
    .endmacro
```

```
.macro  TST3
    clr w
    cp  @2,w
    cpc @1,w
    cpc @0,w
    .endmacro
.macro  TST2
    clr w
    cp  @1,w
    cpc @0,w
    .endmacro

.macro  ADDI4           ; add immediate
    subi    @3,  low(-@4)
    sbci    @2, high(-@4)
    sbci    @1,byte3(-@4)
    sbci    @0,byte4(-@4)
    .endmacro
.macro  ADDI3
    subi    @2,  low(-@3)
    sbci    @1, high(-@3)
    sbci    @0,byte3(-@3)
    .endmacro
.macro  ADDI2
    subi    @1,  low(-@2)
    sbci    @0, high(-@2)
    .endmacro

.macro  SUBI4           ; subtract immediate
    subi    @3,  low(@4)
    sbci    @2, high(@4)
    sbci    @1,byte3(@4)
    sbci    @0,byte4(@4)
    .endmacro
.macro  SUBI3
    subi    @2,  low(@3)
    sbci    @1, high(@3)
    sbci    @0,byte3(@3)
    .endmacro
.macro  SUBI2
    subi    @1,  low(@2)
    sbci    @0, high(@2)
    .endmacro

.macro  LSL5            ; logical shift left
    lsl @4
    rol @3
    rol @2
    rol @1
    rol @0
    .endmacro
.macro  LSL4
    lsl @3
```

```
    rol @2
    rol @1
    rol @0
    .endmacro
.macro  LSL3
    lsl @2
    rol @1
    rol @0
    .endmacro
.macro  LSL2
    lsl @1
    rol @0
    .endmacro

.macro  LSR4            ; logical shift right
    lsr @0
    ror @1
    ror @2
    ror @3
    .endmacro
.macro  LSR3
    lsr @0
    ror @1
    ror @2
    .endmacro
.macro  LSR2
    lsr @0
    ror @1
    .endmacro

.macro  ASR4            ; arithmetic shift right
    asr @0
    ror @1
    ror @2
    ror @3
    .endmacro
.macro  ASR3
    asr @0
    ror @1
    ror @2
    .endmacro
.macro  ASR2
    asr @0
    ror @1
    .endmacro

.macro  ROL8            ; rotate left through carry
    rol @7
    rol @6
    rol @5
    rol @4
    rol @3
    rol @2
```

```
    rol @1
    rol @0
    .endmacro
.macro  ROL7
    rol @6
    rol @5
    rol @4
    rol @3
    rol @2
    rol @1
    rol @0
    .endmacro
.macro  ROL6
    rol @5
    rol @4
    rol @3
    rol @2
    rol @1
    rol @0
    .endmacro
.macro  ROL5
    rol @4
    rol @3
    rol @2
    rol @1
    rol @0
    .endmacro
.macro  ROL4
    rol @3
    rol @2
    rol @1
    rol @0
    .endmacro
.macro  ROL3
    rol @2
    rol @1
    rol @0
    .endmacro
.macro  ROL2
    rol @1
    rol @0
    .endmacro

.macro  ROR8            ; rotate right through carry
    ror @0
    ror @1
    ror @2
    ror @3
    ror @4
    ror @5
    ror @6
    ror @7
    .endmacro
```

```
.macro  ROR7
    ror @0
    ror @1
    ror @2
    ror @3
    ror @4
    ror @5
    ror @6
    .endmacro
.macro  ROR6
    ror @0
    ror @1
    ror @2
    ror @3
    ror @4
    ror @5
    .endmacro
.macro  ROR5
    ror @0
    ror @1
    ror @2
    ror @3
    ror @4
    .endmacro
.macro  ROR4
    ror @0
    ror @1
    ror @2
    ror @3
    .endmacro
.macro  ROR3
    ror @0
    ror @1
    ror @2
    .endmacro
.macro  ROR2
    ror @0
    ror @1
    .endmacro

.macro  PUSH2
    push    @0
    push    @1
    .endmacro
.macro  POP2
    pop @1
    pop @0
    .endmacro

.macro  PUSH3
    push    @0
    push    @1
    push    @2
```

```
    .endmacro
.macro  POP3
    pop @2
    pop @1
    pop @0
    .endmacro

.macro  PUSH4
    push    @0
    push    @1
    push    @2
    push    @3
    .endmacro
.macro  POP4
    pop @3
    pop @2
    pop @1
    pop @0
    .endmacro

.macro  PUSH5
    push    @0
    push    @1
    push    @2
    push    @3
    push    @4
    .endmacro
.macro  POP5
    pop @4
    pop @3
    pop @2
    pop @1
    pop @0
    .endmacro

; --- SRAM operations ---
.macro  INCS4   ; sram      ; increment SRAM 4-byte variable
    lds w,@0
    inc w
    sts @0,w
    brne    end
    lds w,@0+1
    inc w
    sts @0+1,w
    brne    end
    lds w,@0+2
    inc w
    sts @0+2,w
    brne    end
    lds w,@0+3
    inc w
    sts @0+3,w
end:
```

```
    .endmacro

.macro  INCS3   ; sram      ; increment SRAM 3-byte variable
    lds w,@0
    inc w
    sts @0,w
    brne    end
    lds w,@0+1
    inc w
    sts @0+1,w
    brne    end
    lds w,@0+2
    inc w
    sts @0+2,w
end:
    .endmacro

.macro  INCS2   ; sram      ; increment SRAM 2-byte variable
    lds w,@0
    inc w
    sts @0,w
    brne    end
    lds w,@0+1
    inc w
    sts @0+1,w
end:
    .endmacro

.macro  INCS    ; sram      ; increment SRAM 1-byte variable
    lds w,@0
    inc w
    sts @0,w
    .endmacro

.macro  DECS4   ; sram      ; decrement SRAM 4-byte variable
    ldi w,1
    lds u,@0
    sub u,w
    sts @0,u
    clr w
    lds u,@0+1
    sbc u,w
    sts @0+1,u
    lds u,@0+2
    sbc u,w
    sts @0+2,u
    lds u,@0+3
    sbc u,w
    sts @0+3,u
    .endmacro
.macro  DECS3   ; sram      ; decrement SRAM 3-byte variable
    ldi w,1
    lds u,@0
```

```
    sub u,w
    sts @0,u
    clr w
    lds u,@0+1
    sbc u,w
    sts @0+1,u
    lds u,@0+2
    sbc u,w
    sts @0+2,u
    .endmacro
.macro  DECS2   ; sram      ; decrement SRAM 2-byte variable
    ldi w,1
    lds u,@0
    sub u,w
    sts @0,u
    clr w
    lds u,@0+1
    sbc u,w
    sts @0+1,u
    .endmacro
.macro  DECS    ; sram      ; decrement
    lds w,@0
    dec w
    sts @0,w
    .endmacro

.macro  MOVS4   ; addr0,addr1   ; [addr0] <-- [addr1]
    lds w,@1
    sts @0,w
    lds w,@1+1
    sts @0+1,w
    lds w,@1+2
    sts @0+2,w
    lds w,@3+1
    sts @0+3,w
    .endmacro
.macro  MOVS3   ; addr0,addr1   ; [addr0] <-- [addr1]
    lds w,@1
    sts @0,w
    lds w,@1+1
    sts @0+1,w
    lds w,@1+2
    sts @0+2,w
    .endmacro
.macro  MOVS2   ; addr0,addr1   ; [addr0] <-- [addr1]
    lds w,@1
    sts @0,w
    lds w,@1+1
    sts @0+1,w
    .endmacro
.macro  MOVS    ; addr0,addr1   ; [addr0] <-- [addr1]
    lds w,@1
    sts @0,w
```

```
    .endmacro

.macro  SEXT    ; reg1,reg0 ; sign extend
    clr @0
    sbrc    @1,7
    dec @0
    .endmacro

; =====================================
;   Jump/Call with constant arguments
; =====================================

; --- calls with arguments a,b,XYZ ---
.macro  CX  ; subroutine,x
    ldi xl, low(@1)
    ldi xh,high(@1)
    rcall   @0
    .endmacro
.macro  CXY ; subroutine,x,y
    ldi xl, low(@1)
    ldi xh,high(@1)
    ldi yl, low(@2)
    ldi yh,high(@2)
    rcall   @0
    .endmacro
.macro  CXZ ; subroutine,x,z
    ldi xl, low(@1)
    ldi xh,high(@1)
    ldi zl, low(@2)
    ldi zh,high(@2)
    rcall   @0
    .endmacro
.macro  CXYZ    ; subroutine,x,y,z
    ldi xl, low(@1)
    ldi xh,high(@1)
    ldi yl, low(@2)
    ldi yh,high(@2)
    ldi zl, low(@3)
    ldi zh,high(@3)
    rcall   @0
    .endmacro
.macro  CW  ; subroutine,w
    ldi w, @1
    rcall   @0
    .endmacro
.macro  CA  ; subroutine,a
    ldi a0, @1
    rcall   @0
    .endmacro
.macro  CAB ; subroutine,a,b
    ldi a0, @1
    ldi b0, @2
    rcall   @0
```

```
    .endmacro

; --- jump with arguments w,a,b ---
.macro  JW  ; subroutine,w
    ldi w, @1
    rjmp    @0
    .endmacro
.macro  JA  ; subroutine,a
    ldi a0, @1
    rjmp    @0
    .endmacro
.macro  JAB ; subroutine,a,b
    ldi a0, @1
    ldi b0, @2
    rjmp    @0
    .endmacro
.list
```

```
/*
main.asm and entry file for TeaMaker

This file contains:
- interrupt table
- includes
- memory reservation
- interrupt service routines
- reset
*/

; === interrupt table ===
.org 0x0000
    jmp reset
.org    INT0addr
    jmp reset_button_pressed
.org    INT1addr
    jmp start_button_pressed
.org OVF0addr           ; timer overflow 0 interrupt vector
    jmp overflow0
.org OVF1addr           ; timer overflow 1 interrupt vector
    jmp overflow1

.org 0x30

; === includes ===
.include "macros.asm"
.include "math.asm"
.include "definitions.asm"
.include "definitions_teaMaker.inc" ; project specific definitions
.include "relay.asm"
.include "lcd.asm"
.include "printf.asm"
.include "temperature_sensor.asm"
.include "motor1.asm"
.include "Delay_Timer.asm"
.include "uart_settings.asm"
.include "statemachine.asm"

; === memory reservation ===

.dseg
.org 0x100 ; begin sram
delay_time_timer1:   .byte 2
delay_time_timer0 :  .byte 1
steep_time_timer1:   .byte 2
.cseg

; === interrupt service routines ===

reset_button_pressed:
    PRINTF  LCD             ; print formatted
    .db "Reset Button Pressed",CR,0
```

```
    WAIT_MS 1000
    rcall LCD_clear
    rjmp reset

start_button_pressed:
    cpi CURRENT_STATE, STATE_IDLE
    breq start_button_pressed_in_state
    reti
start_button_pressed_in_state:
    PRINTF  LCD
    .db "START",CR,0
    WAIT_MS 1000
    rcall LCD_clear
    ldi CURRENT_STATE, STATE_DELAY
    reti

overflow0:                  ; 8bit counter: 256
    cpi CURRENT_STATE, STATE_DELAY
    breq overflow0_start
    cpi CURRENT_STATE, STATE_TEA_STEEP
    brne overflow0_end
overflow0_start:
    INVP    PORTD,5         ; invert the portD.5 : clock signal for Timer/
      Counter1
    call preset_timer0
overflow0_end:
    reti


overflow1:                  ; 16bit counter: 65536
    cpi CURRENT_STATE, STATE_DELAY
    breq overflow1_delay
    cpi CURRENT_STATE, STATE_TEA_STEEP
    brne overflow1_end
overflow1_steep:
    PRESET_TIMER1 steep_time_timer1
    rjmp PC+2
overflow1_delay:
    PRESET_TIMER1 delay_time_timer1
    ldi b1, 1               ; b1 is 0 before overflow1 is executed for the
      first time
overflow1_end:
    reti




; === reset ===
reset:
    LDSP RAMEND

    RELAY_INIT
    rcall   LCD_init        ; initialize the LCD
```

```
    rcall   wire1_init      ; initialize 1-wire(R) interface
    call init_uart          ; initialize uart interface

    OUTI    EIMSK,0b00000011 ; enable INT0, INT1
    OUTI TIMSK, 0       ; disable interrupt timer 0 (in case of a reset during
      the delay state)
    sei                 ; set global interrupt

    ; default settings
    ; timer0
    ldi a0, 0x3f  ; fixed value for overflow0 every 6 seconds
    sts delay_time_timer0, a0
    ; delay time
    ldi a1, 0xFF    ; no delay time
    ldi a2, 0xFF
    sts delay_time_Timer1, a1
    sts delay_time_Timer1+1, a2
    clr a0
    clr a1
    ; steep time
    ldi a0, 0xE6 ; default value: steep for 5min
    ldi a1, 0xFF
    sts steep_time_timer1, a0
    sts steep_time_timer1+1, a1
    clr a0
    clr a1

    ldi CURRENT_STATE, STATE_SETTINGS ; initial state

    jmp state_change
```

```
; file  math.asm   target ATmega128L-4MHz-STK300
; purpose library, mathematical routines
; copyright R.Holzer

; === unsigned multiplication (c=a*b) ===

mul11:  clr c1          ; clear upper half of result c
    mov c0,b0           ; place b in lower half of c
    lsr c0          ; shift LSB (of b) into carry
    ldi w,8         ; load bit counter
_m11:   brcc    PC+2            ; skip addition if carry=0
    add c1,a0           ; add a to upper half of c
    ROR2    c1,c0           ; shift-right c, LSB (of b) into carry
    DJNZ    w,_m11          ; Decrement and Jump if bit-count Not Zero
    ret

mul21:  CLR2    c2,c1           ; clear upper half of result c
    mov c0,b0           ; place b in lower half of c
    lsr c0          ; shift LSB (of b) into carry
    ldi w,8         ; load bit counter
_m21:   brcc    PC+3            ; skip addition if carry=0
    ADD2    c2,c1, a1,a0        ; add a to upper half of c
    ROR3    c2,c1,c0        ; shift-right c, LSB (of b) into carry
    DJNZ    w,_m21          ; Decrement and Jump if bit-count Not Zero
    ret

mul22:  CLR2    c3,c2           ; clear upper half of result c
    MOV2    c1,c0, b1,b0        ; place b in lower half of c
    LSR2    c1,c0           ; shift LSB (of b) into carry
    ldi w,16            ; load bit counter
_m22:   brcc    PC+3            ; skip addition if carry=0
    ADD2    c3,c2, a1,a0        ; add a to upper half of c
    ROR4    c3,c2,c1,c0     ; shift-right c, LSB (of b) into carry
    DJNZ    w,_m22          ; Decrement and Jump if bit-count Not Zero
    ret

mul31:  CLR3    c3,c2,c1        ; clear upper half of result c
    mov c0,b0           ; place b in lower half of c
    lsr c0          ; shift LSB (of b) into carry
    ldi w,8         ; load bit counter
_m31:   brcc    PC+4            ; skip addition if carry=0
    ADD3    c3,c2,c1, a2,a1,a0  ; add a to upper half of c
    ROR4    c3,c2,c1,c0     ; shift-right c, LSB (of b) into carry
    DJNZ    w,_m31          ; Decrement and Jump if bit-count Not Zero
    ret

mul32:  CLR3    d0,c3,c2        ; clear upper half of result c
    MOV2    c1,c0, b1,b0        ; place b in lower half of c
    LSR2    c1,c0           ; shift LSB (of b) into carry
    ldi w,16            ; load bit counter
_m32:   brcc    PC+4            ; skip addition if carry=0
    ADD3    d0,c3,c2, a2,a1,a0  ; add a to upper half of c
    ROR5    d0,c3,c2,c1,c0      ; shift-right c, LSB (of b) into carry
```

```
    DJNZ    w,_m32          ; Decrement and Jump if bit-count Not Zero
    ret

mul33:  CLR3    d1,d0,c3        ; clear upper half of result c
    MOV3    c2,c1,c0, b2,b1,b0  ; place b in lower half of c
    LSR3    c2,c1,c0        ; shift LSB (of b) into carry
    ldi w,24            ; load bit counter
_m33:   brcc    PC+4            ; skip addition if carry=0
    ADD3    d1,d0,c3, a2,a1,a0  ; add a to upper half of c
    ROR6    d1,d0,c3,c2,c1,c0   ; shift-right c, LSB (of b) into carry
    DJNZ    w,_m33          ; Decrement and Jump if bit-count Not Zero
    ret

mul41:  CLR4    d0,c3,c2,c1     ; clear upper half of result c
    mov c0,b0           ; place b in lower half of c
    lsr c0          ; shift LSB (of b) into carry
    ldi w,8         ; load bit counter
_m41:   brcc    PC+5            ; skip addition if carry=0
    ADD4    d0,c3,c2,c1, a3,a2,a1,a0; add a to upper half of c
    ROR5    d0,c3,c2,c1,c0      ; shift-right c, LSB (of b) into carry
    DJNZ    w,_m41          ; Decrement and Jump if bit-count Not Zero
    ret

mul42:  CLR4    d1,d0,c3,c2     ; clear upper half of result c
    MOV2    c1,c0, b1,b0            ; place b in lower half of c
    LSR2    c1,c0           ; shift LSB (of b) into carry
    ldi w,16            ; load bit counter
_m42:   brcc    PC+5            ; skip addition if carry=0
    ADD4    d1,d0,c3,c2, a3,a2,a1,a0; add a to upper half of c
    ROR6    d1,d0,c3,c2,c1,c0   ; shift-right c, LSB (of b) into carry
    DJNZ    w,_m42          ; Decrement and Jump if bit-count Not Zero
    ret

mul43:  CLR4    d2,d1,d0,c3     ; clear upper half of result c
    MOV3    c2,c1,c0, b2,b1,b0  ; place b in lower half of c
    LSR3    c2,c1,c0        ; shift LSB (of b) into carry
    ldi w,24            ; load bit counter
_m43:   brcc    PC+5            ; skip addition if carry=0
    ADD4    d2,d1,d0,c3, a3,a2,a1,a0; add a to upper half of c
    ROR7    d2,d1,d0,c3,c2,c1,c0    ; shift-right c, LSB (of b) into carry
    DJNZ    w,_m43          ; Decrement and Jump if bit-count Not Zero
    ret

mul44:  CLR4    d3,d2,d1,d0     ; clear upper half of result c
    MOV4    c3,c2,c1,c0, b3,b2,b1,b0; place b in lower half of c
    LSR4    c3,c2,c1,c0     ; shift LSB (of b) into carry
    ldi w,32            ; load bit counter
_m44:   brcc    PC+5            ; skip addition if carry=0
    ADD4    d3,d2,d1,d0, a3,a2,a1,a0; add a to upper half of c
    ROR8    d3,d2,d1,d0,c3,c2,c1,c0 ; shift-right c, LSB (of b) into carry
    DJNZ    w,_m44          ; Decrement and Jump if bit-count Not Zero
    ret
```

```
; === signed multiplication ===
mul11s: rcall   mul11
    sbrc    a0,7
    sub c1,b0
    sbrc    b0,7
    sub c1,a0
    ret

mul22s: rcall   mul22
    sbrs    a1,7
    rjmp    PC+3
    SUB2    c3,c2, b1,b0
    sbrs    b1,7
    rjmp    PC+3
    SUB2    c3,c2, a1,a0
    ret

mul33s: rcall   mul33
    sbrs    a2,7
    rjmp    PC+4
    SUB3    d1,d0,c3, b2,b1,b0
    sbrs    b2,7
    rjmp    PC+4
    SUB3    d1,d0,c3, a2,a1,a0
    ret

mul44s: rcall   mul44
    sbrs    a3,7
    rjmp    PC+5
    SUB4    d3,d2,d1,d0, b3,b2,b1,b0
    sbrs    b3,7
    rjmp    PC+5
    SUB4    d3,d2,d1,d0, a3,a2,a1,a0
    ret

; === unsigned division c=a/b ===
div11:  mov c0,a0           ; c will contain the result
    clr d0          ; d will contain the remainder
    ldi w,8         ; load bit counter
_d11:   ROL2    d0,c0           ; shift carry into result c
    sub d0,b0           ; subtract b from remainder
    brcc    PC+2
    add d0,b0           ; restore if remainder became negative
    DJNZ    w,_d11          ; Decrement and Jump if bit-count Not Zero
    rol c0          ; last shift (C into result c)
    com c0          ; complement result
    ret

div21:  MOV2    c1,c0, a1,a0        ; c will contain the result
    clr d0          ; d will contain the remainder
    ldi w,16            ; load bit counter
_d21:   ROL3    d0,c1,c0        ; shift carry into result c
    sub d0,b0           ; subtract b from remainder
```

```
    brcc    PC+2
    add d0,b0           ; restore if remainder became negative
    DJNZ    w,_d21          ; Decrement and Jump if bit-count Not Zero
    ROL2    c1,c0           ; last shift (carry into result c)
    COM2    c1,c0           ; complement result
    ret

div22:  MOV2    c1,c0, a1,a0        ; c will contain the result
    CLR2    d1,d0           ; d will contain the remainder
    ldi w,16            ; load bit counter
_d22:   ROL4    d1,d0,c1,c0     ; shift carry into result c
    SUB2    d1,d0, b1,b0        ; subtract b from remainder
    brcc    PC+3
    ADD2    d1,d0, b1,b0        ; restore if remainder became negative
    DJNZ    w,_d22          ; Decrement and Jump if bit-count Not Zero
    ROL2    c1,c0           ; last shift (carry into result c)
    COM2    c1,c0           ; complement result
    ret

div31:  MOV3    c2,c1,c0, a2,a1,a0  ; c will contain the result
    clr d0          ; d will contain the remainder
    ldi w,24            ; load bit counter
_d31:   ROL4    d0,c2,c1,c0     ; shift carry into result c
    sub d0, b0          ; subtract b from remainder
    brcc    PC+2
    add d0, b0          ; restore if remainder became negative
    DJNZ    w,_d31          ; Decrement and Jump if bit-count Not Zero
    ROL3    c2,c1,c0        ; last shift (carry into result c)
    COM3    c2,c1,c0        ; complement result
    ret

div32:  MOV3    c2,c1,c0, a2,a1,a0  ; c will contain the result
    CLR2    d1,d0           ; d will contain the remainder
    ldi w,24            ; load bit counter
_d32:   ROL5    d1,d0,c2,c1,c0      ; shift carry into result c
    SUB2    d1,d0, b1,b0        ; subtract b from remainder
    brcc    PC+3
    ADD2    d1,d0, b1,b0        ; restore if remainder became negative
    DJNZ    w,_d32          ; Decrement and Jump if bit-count Not Zero
    ROL3    c2,c1,c0        ; last shift (carry into result c)
    COM3    c2,c1,c0        ; complement result
    ret

div33:  MOV3    c2,c1,c0, a2,a1,a0  ; c will contain the result
    CLR3    d2,d1,d0        ; d will contain the remainder
    ldi w,24            ; load bit counter
_d33:   ROL6    d2,d1,d0,c2,c1,c0   ; shift carry into result c
    SUB3    d2,d1,d0, b2,b1,b0  ; subtract b from remainder
    brcc    PC+4
    ADD3    d2,d1,d0, b2,b1,b0  ; restore if remainder became negative
    DJNZ    w,_d33          ; Decrement and Jump if bit-count Not Zero
    ROL3    c2,c1,c0        ; last shift (carry into result c)
    COM3    c2,c1,c0        ; complement result
```

```
    ret

div41:  MOV4    c3,c2,c1,c0, a3,a2,a1,a0; c will contain the result
    clr d0          ; d will contain the remainder
    ldi w,32            ; load bit counter
_d41:   ROL5    d0,c3,c2,c1,c0      ; shift carry into result c
    sub d0, b0          ; subtract b from remainder
    brcc    PC+2
    add d0, b0          ; restore if remainder became negative
    DJNZ    w,_d41          ; Decrement and Jump if bit-count Not Zero
    ROL4    c3,c2,c1,c0     ; last shift (carry into result c)
    COM4    c3,c2,c1,c0     ; complement result
    ret

div42:  MOV4    c3,c2,c1,c0, a3,a2,a1,a0; c will contain the result
    CLR2    d1,d0           ; d will contain the remainder
    ldi w,32            ; load bit counter
_d42:   ROL6    d1,d0,c3,c2,c1,c0   ; shift carry into result c
    SUB2    d1,d0, b1,b0        ; subtract b from remainder
    brcc    PC+3
    ADD2    d1,d0, b1,b0        ; restore if remainder became negative
    DJNZ    w,_d42          ; Decrement and Jump if bit-count Not Zero
    ROL4    c3,c2,c1,c0     ; last shift (carry into result c)
    COM4    c3,c2,c1,c0     ; complement result
    ret

div43:  MOV4    c3,c2,c1,c0, a3,a2,a1,a0; c will contain the result
    CLR3    d2,d1,d0        ; d will contain the remainder
    ldi w,32            ; load bit counter
_d43:   ROL7    d2,d1,d0,c3,c2,c1,c0    ; shift carry into result c
    SUB3    d2,d1,d0, b2,b1,b0  ; subtract b from remainder
    brcc    PC+4
    ADD3    d2,d1,d0, b2,b1,b0  ; restore if remainder became negative
    DJNZ    w,_d43          ; Decrement and Jump if bit-count Not Zero
    ROL4    c3,c2,c1,c0     ; last shift (carry into result c)
    COM4    c3,c2,c1,c0     ; complement result
    ret

div44:  MOV4    c3,c2,c1,c0, a3,a2,a1,a0; c will contain the result
    CLR4    d3,d2,d1,d0     ; d will contain the remainder
    ldi w,32            ; load bit counter
_d44:   ROL8    d3,d2,d1,d0,c3,c2,c1,c0 ; shift carry into result c
    SUB4    d3,d2,d1,d0, b3,b2,b1,b0; subtract b from remainder
    brcc    PC+5
    ADD4    d3,d2,d1,d0, b3,b2,b1,b0; restore if remainder became negative
    DJNZ    w,_d44          ; Decrement and Jump if bit-count Not Zero
    ROL4    c3,c2,c1,c0     ; last shift (carry into result c)
    COM4    c3,c2,c1,c0     ; complement result
    ret

; === signed division ===
div33s: push    u
    mov u,a2
```

```
    eor u,b2
    sbrs    a2,7
    rjmp    d33a
    NEG3    a2,a1,a0
d33a:   sbrs    b2,7
    rjmp    d33b
    NEG3    b2,b1,b0
d33b:   rcall   div33
    sbrs    u,7
    rjmp    d33c
    NEG3    c2,c1,c0
d33c:   pop u
    ret
```

```
/*
motor1.asm

This file defines macros and sub-routines to control the Towerplus Micro Servo SG90

content:
- Macros related to servo
- Subroutines used for servo
*/

.macro SERVO1_INIT
OUTI    DDRE,0xff           ; configure PortE to output for servos
P0  PORTE,SERVO1            ; pin=0
.endmacro


; purpose: call a subroutine and load values into a0,a1,b0
; in: address of subroutine, speed value, angle value
; changes a1,a0,b0
.macro CA3
    ldi a0, low(@1)     ; speed and rotation direction
    ldi a1, high(@1)    ; speed and rotation direction
    ldi b0, @2          ; angle
    rcall   @0
.endmacro


; purpose: determine direction of rotation
ang_rot_ccw:
CA3     _s360, (npt+100), 0x50  ; address, distance to npt=speed ccw, angle of rotation
ret

ang_rot_cw:
CA3     _s360, (npt-100), 0x50  ; address, distance to npt=speed cw, angle of rotation
ret


; purpose: execute rotation -> generate b0 pulses
_s360:
ls3601:
    rcall   servoreg_pulse  ; generate 1 pulse
    dec     b0              ; decremente b0 until it is 0
    brne    ls3601
    ret


; purpose: generates 1 pulse for servo control impulse signal (at 0 for 20ms, at 1 for duration corresponding to values in a0,a1)
servoreg_pulse:
    WAIT_US 20000           ; servo control impulse signal is at 0 for 20ms
```

```
    MOV2    a3,a2, a1,a0    ; use a3 and a2 as intermediate variables, in order
       not to loose value of a1,a0
    P1  PORTE,SERVO1        ; pin=1: servo control impulse signal is set to Vcc
lpssp01:
    SUBI2   a3,a2,0x1       ; loop: substract 1 until 0 is reached
    brne    lpssp01
    P0  PORTE,SERVO1        ; pin=0: servo control impulse signal is set to 0
    ret
```

```
; file  printf.asm   target ATmega128L-4MHz-STK300
; purpose library, formatted output generation
; author (c) R.Holzer (adapted MICRO210/EE208 A.Schmid)
; v2019.02 20180821 AxS supports SRAM input from 0x0260
;                   through 0x02ff that should be reserved

; === description ===
;
; The program "printf" interprets and prints formatted strings.
; The special formatting characters regognized are:
;
; FDEC  decimal number
; FHEX  hexadecimal number
; FBIN  binary number
; FFRAC fixed fraction number
; FCHAR single ASCII character
; FSTR  zero-terminated ASCII string
;
; The special formatting characters are distinguished from normal
; ASCII characters by having their bit7 set to 1.
;
; Signification of bit fields:
;
; b     bytes       1..4 b bytes        2
; s     sign        0(unsigned), 1(signed)  1
; i     integer digits
; e     base        2,,36           5
; dp    dec. point  0..32           5
; $if   i=integer digits,  0=all digits,  1..15 digits
;       f=fraction digits, 0=no fraction, 1..15 digits
;
; Formatting characters must be followed by an SRAM address (0..ff)
; that determines the origin of variables that must be printed (if any)
; FBIN, sram
; FHEX, sram
; FDEC, sram
; FCHAR,sram
; FSTR, sram
;
; The address 'sram' is a 1-byte constant. It addresses
;    0..1f  registers r0..r31,
;   20..3f  i/o ports, (need to be addressed with an offset of $20)
;   0x0260..0x02ff  SRAM
; Variables can be located into register and I/0s, and can also
; be stored into data SRAM at locations 0x0200 through 0x02ff. Any
; sram address higher than 0x0060 is assumed to be at (0x0260+address)
; from automatic address detection in _printf_formatted: and subsequent
; assignment to xh; xl keeps its value. Consequently, variables that are
; to be stored into SRAM and further printed by fprint must reside at
; 0x0200 up to 0x02ff, and must be addressed using a label. Usage: see
; file string1.asm, for example.

; The FFRAC formatting character must be followed by
```

```
;   ONE sram address and
;   TWO more formatting characters
; FFRAC,sram,dp,$if

; dp    decimal point position, 0=right, 32=left
; $if   format i.f, i=integer digits, f=fraction digits

; The special formatting characters use the following coding
;
; FDEC  11bb'iiis   i=0 all digits, i=1-7 digits
; FBIN  101i'iiis   i=0 8 digits,   i=1-7 digits
; FHEX  1001'iiis   i=0 8 digits,   i=1-7 digits
; FFRAC 1000'1bbs
; FCHAR 1000'0100
; FSTR  1000'0101
; FREP  1000'0110
; FFUNC 1000'0111
;   1000'0010
;   1000'0011
; FESC  1000'0000

; examples
; formatting string         printing
; "a=",FDEC,a,0             1-byte variable a, unsigned decimal
; "a=",FDEC2,a,0            2-byte variable a (a1,a0), unsigend
; "a=",FDEC|FSIGN,a,0       1-byte variable 1, signed decimal
; "n=",FBIN,PIND+$20,0      i/o port, binary, notice offset of $20
; "f=",FFRAC4|FSIGN,a,16,$88,0  4-byte signed fixed-point fraction
;               dec.point at 16, 8 int.digits, 8 frac.digits
; "f=",FFRAC2,a,16,$18,0        2-byte unsigned fixed-point fraction
;               dec.point at 16, 1 int.digits, 8 frac.digits
; "a=",FDEC|FDIG5|FSIGN,a,0 1-byte variable, 5-digit, decimal, signed
; "a=",FDEC|FDIG5,a,0       1-byte variable, 5-digit, decimal, unsigned

; === registers modified ===
; e0,e1 used to transmit address of putc routine
; zh,zl used as pointer to prog-memory

; === constants ============================================

.equ    FDEC    = 0b11000000    ; 1-byte variable
.equ    FDEC2   = 0b11010000    ; 2-byte variable
.equ    FDEC3   = 0b11100000    ; 3-byte variable
.equ    FDEC4   = 0b11110000    ; 4-byte variable

.equ    FBIN    = 0b10100000
.equ    FHEX    = 0b10010100    ; 1-byte variable
.equ    FHEX2   = 0b10011000    ; 2-byte variable
.equ    FHEX3   = 0b10011100    ; 3-byte variable
.equ    FHEX4   = 0b10010000    ; 4-byte variable

.equ    FFRAC   = 0b10001000    ; 1-byte variable
.equ    FFRAC2  = 0b10001010    ; 2-byte variable
```

```
.equ    FFRAC3  = 0b10001100    ; 3-byte variable
.equ    FFRAC4  = 0b10001110    ; 4-byte variable

.equ    FCHAR   = 0b10000100
.equ    FSTR    = 0b10000101

.equ    FSIGN   = 0b00000001

.equ    FDIG1   = 1<<1
.equ    FDIG2   = 2<<1
.equ    FDIG3   = 3<<1
.equ    FDIG4   = 4<<1
.equ    FDIG5   = 5<<1
.equ    FDIG6   = 6<<1
.equ    FDIG7   = 7<<1

; ===macro ================================================

.macro  PRINTF          ; putc function (UART, LCD...)
    ldi w, low(@0)      ; address of "putc" in e1:d0
    mov e0,w
    ldi w,high(@0)
    mov e1,w
    rcall   _printf
    .endmacro

; mod   y,z


; === routines ============================================

_printf:
    POPZ            ; z points to begin of "string"
    MUL2Z           ; multiply Z by two, (word ptr -> byte ptr)
    PUSHX

_printf_read:
    lpm             ; places prog_mem(Z) into r0 (=c)
    adiw    zl,1    ; increment pointer Z
    tst r0          ; test for ZERO (=end of string)
    breq    _printf_end ; char=0 indicates end of ascii string
    brmi    _printf_formatted ; bit7=1 indicates formatting character
    mov w,r0
    rcall   _putw   ; display the character
    rjmp    _printf_read    ; read next character in the string

_printf_end:
    adiw    zl,1    ; point to the next character
    DIV2Z           ; divide by 2 (byte ptr -> word ptr)
    POPX
    ijmp            ; return to instruction after "string"

_printf_formatted:
```

```
; FDEC  11bb'iiis
; FBIN  101i'iiis
; FHEX  1001'iiis
; FFRAC 1000'1bbs
; FCHAR 1000'0100
; FSTR  1000'0101

    bst r0,0        ; store sign in T
    mov w,r0        ; store formatting character in w
    lpm
    mov xl,r0       ; load x-pointer with SRAM address
    cpi xl,0x60
    brlo rio_space
dataram_space:      ; variable originates from SRAM memory
    ldi xh,0x02     ;>addresses are limited to 0x0260 through 0x02ff
    rjmp space_detect_end   ;>that enables automatic detection of the origin
rio_space:          ; variable originates from reg or I/O space
    clr xh          ; clear high-byte, addresses are 0x0000 through 0x003f
      (0x005f)
space_detect_end:
    adiw    zl,1    ; increment pointer Z

;   JB1 w,6,_putdec
;   JB1 w,5,_putbin
;   JB1 w,4,_puthex
;   JB1 w,3,_putfrac
    JK  w,FCHAR,_putchar
    JK  w,FSTR ,_putstr
    rjmp    _putnum

    rjmp    _printf_read

; === putc (put character) ================================
; in    w   character to put
;   e1,e0   address of output routine (UART, LCD putc)
_putw:
    PUSH3   a0,zh,zl
    MOV3    a0,zh,zl, w,e1,e0
    icall           ; indirect call to "putc"
    POP3    a0,zh,zl
    ret

; === putchar (put character) =============================
; in    x   pointer to character to put
_putchar:
    ld  w,x
    rcall   _putw
    rjmp    _printf_read

; === putstr (put string) =================================
; in    x   pointer to ascii string
;   b3,b2   address of output routine (UART, LCD putc)
```

```
_putstr:
    ld  w,x+
    tst w
    brne    PC+2
    rjmp    _printf_read
    rcall   _putw
    rjmp    _putstr

; === putnum (dec/bin/hex/frac) ============================
; in    x   pointer to SRAM variable to print
;   r0  formatting character

_putnum:
    PUSH4   a3,a2,a1,a0 ; safeguard a
    PUSH4   b3,b2,b1,b0 ; safeguard b
    LDX4    a3,a2,a1,a0 ; load operand to print into a

; FDEC  11bb'iiis
; FBIN  101i'iiis
; FHEX  1001'iiis
; FRACT 1000'1bbs

    JB1 w,6,_putdec
    JB1 w,5,_putbin
    JB1 w,4,_puthex
    JB1 w,3,_putfrac

; FDEC  11bb'iiis
_putdec:
    ldi b0,10       ; b0 = base (10)

    mov b1,w
    lsr b1
    andi    b1,0b111
    swap    b1      ; b1 = format 0iii'0000 (integer digits)
    ldi b2,0        ; b2 = dec. point position = 0 (right)

    mov b3,w
    swap    b3
    andi    b3,0b11
    inc b3          ; b3 = number of bytes (1..4)
    rjmp    _getnum ; get number of digits (iii)

; FBIN  101i'iiis   addr
_putbin:
    ldi b0,2        ; b0 = base (2)
    ldi b3,4        ; b3 = number of bytes (4)
    rjmp    _getdig ; get number of digits (iii)

; FHEX  1001'iiis   addr
_puthex:
    ldi b0,16       ; b0 = base (16)
    ldi b3,4        ; b3 = number of bytes (4)
```

```
    rjmp    _getdig

_getdig:
    mov b1,w
    lsr b1
    andi    b1,0b111
    brne    PC+2
    ldi b1,8        ; if b1=0 then 8-digits
    swap    b1      ; b1 = format 0iii'0000 (integer digits)
    ldi b2, 0       ; b2 = dec. point position = 0 (right)
    rjmp    _getnum

; FFRAC 1000'1bbs   addr     00dd'dddd,     iiii'ffff

_putfrac:
    ldi b0,10       ; base=10
    lpm
    mov b2,r0       ; load dec.point position
    adiw    zl,1    ; increment char pointer
    lpm
    mov b1,r0       ; load ii.ff format
    adiw    zl,1    ; increment char pointer

    mov b3,w
    asr b3
    andi    b3,0b11
    inc b3          ; b3 = number of bytes (1..4)

    rjmp    _getnum

_getnum:
; in    a   4-byte variable
;   b3  number of bytes (1..4)
;   T   sign, 0=unsigned, 1=signed

    JK  b3,4,_printf_4b
    JK  b3,3,_printf_3b
    JK  b3,2,_printf_2b

_printf_1b:         ; sign extension
    clr a1
    brtc    PC+3    ; T=1 sign extension
    sbrc    a0,7
    ldi a1,0xff
_printf_2b:
    clr a2
    brtc    PC+3    ; T=1 sign extension
    sbrc    a1,7
    ldi a2,0xff
_printf_3b:
    clr a3
    brtc    PC+3    ; T=1 sign extension
    sbrc    a2,7
```

```
    ldi a3,0xff
_printf_4b:

    rcall   _ftoa       ; float to ascii
    POP4    b3,b2,b1,b0 ; restore b
    POP4    a3,a2,a1,a0 ; restore a

    rjmp    _printf_read

; ================================================
; func  ftoa
; converts a fixed-point fractional number to an ascii string
; author (c) Raphael Holzer
;
; in    a3-a0   variable to print
;   b0  base, 2 to 36, but usually decimal (10)
;   b1  number of digits to print ii.ff
;   b2  position of the decimal point (0=right, 32=left)
;   T   sign (T=0 unsiged, T=1 signed)

_ftoa:
    push    d0
    PUSH4   c3,c2,c1,c0 ; c = fraction part, a = integer part
    CLR4    c3,c2,c1,c0 ; clear fraction part

    brtc    _ftoa_plus  ; if T=0 then unsigned
    clt
    tst a3              ; if MSb(a)=1 then a=-a
    brpl    _ftoa_plus
    set                 ; T=1 (minus)
    tst b1
    breq    PC+2        ; if b1=0 the print ALL digits
    subi    b1,0x10     ; decrease int digits
    NEG4    a3,a2,a1,a0 ; negate a
_ftoa_plus:
    tst b2              ; b0=0 (only integer part)
    breq    _ftoa_int
_ftoa_shift:
    ASR4    a3,a2,a1,a0 ; a = integer part
    ROR4    c3,c2,c1,c0 ; c = fraction part
    DJNZ    b2,_ftoa_shift
_ftoa_int:
    push    b1          ; ii.ff (ii=int digits)
    swap    b1
    andi    b1,0x0f

    ldi w,'.'           ; push decimal point
    push    w
_ftoa_int1:
    rcall   _div41      ; int=int/10
    mov w,d0            ; d=reminder
    rcall   _hex2asc
    push    w           ; push rem(int/10)
```

```
    TST4    a3,a2,a1,a0 ; (int/10)=?
    breq    _ftoa_space ; (int/10)=0 then finished
    tst b1
    breq    _ftoa_int1  ; if b1=0 then print ALL int-digits
    DJNZ    b1,_ftoa_int1
    rjmp    _ftoa_sign
_ftoa_space:
    tst b1              ; if b1=0 then print ALL int-digits
    breq    _ftoa_sign
    dec b1
    breq    _ftoa_sign
    ldi w,' '           ; write spaces
    rcall   _putw
    rjmp    _ftoa_space
_ftoa_sign:
    brtc    PC+3        ; if T=1 then write 'minus'
    ldi w,'-'
    rcall   _putw
_ftoa_int3:
    pop w
    cpi w,'.'
    breq    PC+3
    rcall   _putw
    rjmp    _ftoa_int3

    pop b1              ; ii.ff (ff=frac digits)
    andi    b1,0x0f
    tst b1
    breq    _ftoa_end
_ftoa_point:
    rcall   _putw       ; write decimal point
    MOV4    a3,a2,a1,a0, c3,c2,c1,c0
_ftoa_frac:
    rcall   _mul41      ; d.frac=10*frac
    mov w,d0
    rcall   _hex2asc
    rcall   _putw
    DJNZ    b1,_ftoa_frac
_ftoa_end:
    POP4    c3,c2,c1,c0
    pop d0
    ret

; === hexadecimal to ascii ===
; in    w
_hex2asc:
    cpi w,10
    brsh    PC+3
    addi    w,'0'
    ret
    addi    w,('a'-10)
    ret
```

```
; === multiply 4byte*1byte ===
; funct mul41
; multiplies a3-a0 (4-byte) by b0 (1-byte)
; author (c) Raphael Holzer, EPFL
;
; in    a3..a0  multiplicand (argument to multiply)
;   b0  multiplier
; out   a3..a0  result
;   d0  result MSB (byte 4)
;
_mul41: clr d0          ; clear byte4 of result
    ldi w,32            ; load bit counter
__m41:  clc             ; clear carry
    sbrc    a0,0        ; skip addition if LSB=0
    add d0,b0           ; add b to MSB of a
    ROR5    d0,a3,a2,a1,a0  ; shift-right c, LSB (of b) into carry
    DJNZ    w,__m41     ; Decrement and Jump if bit-count Not Zero
    ret

; === divide 4byte/1byte ===
; func div41
; in    a0..a3  divident (argument to divide)
;   b0  divider
; out   a0..a3  result
;   d0  reminder
;
_div41: clr d0          ; d will contain the remainder
    ldi w,32            ; load bit counter
__d41:  ROL5    d0,a3,a2,a1,a0  ; shift carry into result c
    sub d0, b0          ; subtract b from remainder
    brcc    PC+2
    add d0, b0          ; restore if remainder became negative
    DJNZ    w,__d41     ; Decrement and Jump if bit-count Not Zero
    ROL4    a3,a2,a1,a0 ; last shift (carry into result c)
    COM4    a3,a2,a1,a0 ; complement result
    ret
```

```
/*
 * relay.asm
 *
 * This file defines macros and sub-routines to control the relay
 * SparkFun Qwiic Single Relay - COM-15093
 */

 .include "i2cx.asm"



 ; source: c++ firmware found at https://github.com/sparkfun/Qwiic_Relay/blob/
   master/Firmware/Qwiic_Relay_Firmware/Qwiic_Relay_Firmware.ino
.equ ADDRESS_RELAY              = 0x18
.equ COMMAND_RELAY_OFF          = 0x00
.equ COMMAND_RELAY_ON           = 0x01
.equ COMMAND_CHANGE_ADDRESS     = 0x03
.equ COMMAND_FIRMWARE_VERSION   = 0x04
.equ COMMAND_STATUS             = 0x05
.equ COMMAND_HAS_BEEN_CHECKED   = 0x99

; We had electrical problems with the I2C interface on the sparkfun board side
  (the relay pulled down the lines) -> we directly connect to the relay and use
   simple zero and one to turn it on/off
; As we have only one relay this solution is acceptable.


.macro RELAY_INIT
    OUTI    DDRB,0x0F   ; configure PortB to output for relay/input for
      temperature sensor
    RELAY_OFF
.endmacro

.macro RELAY_ON
    P1 PORTB, 3
.endmacro

.macro RELAY_OFF
    P0 PORTB, 3
.endmacro
```

```
/*
statemachine.asm

This file contains the statemachine of the TeaMaker consisting of
- state change routine
- all states

A state has has the following syntax:

active_state_{statename}:
    {code to be executed when entering the state}
active_state_{statename}_loop:
    {code to be repeated as long as the state doesn't change}
    (if state changes, go to state_change)

States not having a loop do not implement the active_state_{statename}_loop
  address.

The states in this file are
 - idle
 - settings
 - delay
 - heating
 - add_tea_bag
 - remove_tea_bag
 */

 main:
 state_change:
 // Go to the state whose code is currently stored in the CURRENT_STATE
   register
    cpi CURRENT_STATE, STATE_SETTINGS
    brne PC+2
    rjmp active_state_settings
    cpi CURRENT_STATE, STATE_DELAY
    brne PC+2
    rjmp active_state_delay
    cpi CURRENT_STATE, STATE_HEATING
    brne PC+2
    rjmp active_state_heating
    cpi CURRENT_STATE, STATE_ADD_TEA_BAG
    brne PC+2
    rjmp active_state_add_tea_bag
    cpi CURRENT_STATE, STATE_TEA_STEEP
    brne PC+2
    rjmp active_state_tea_steep
    cpi CURRENT_STATE, STATE_REMOVE_TEA_BAG
    brne PC+2
    rjmp active_state_remove_tea_bag
    cpi CURRENT_STATE, STATE_IDLE
    brne PC+2
    rjmp active_state_idle
    jmp reset   ; reset if no state is selected -> error
```

```
 active_state_settings:
    call LCD_clear
    PRINTF  LCD
    .db "SETTINGS",CR,0
    call uart_settings_main ; this chagnes the CURRENT_STATE variable depending
       on the user input.
    call LCD_clear
    rjmp state_change       ; go to the next state


 active_state_delay:
    call LCD_clear
    clr b1      ; b1 is initially 0, is set to 1 as soon as overflow1 happens
    PRESET_TIMER1 delay_time_timer1
    call timer_init     ; activates timer0 and timer1 overflow interrupt
    call idle_mode_init ; activates idle mode of microcontroller
 active_state_delay_loop:
    call display_time   ; displays how much time is left
    sleep   ; starts idle mode
    nop     ; important, without nop, the compare is done before timer1 changed
       the value of b1
    cpi b1, 0x00
    breq state_delay_end ;
    ; if b1=1 (overflow1 happend) stop waiting
    OUTI TIMSK, 0           ; disable interrupt timer 0 and 1
    call idle_mode_off      ; desactivates idle mode
    ldi CURRENT_STATE, STATE_HEATING
state_delay_end:
    cpi CURRENT_STATE, STATE_DELAY  ; check if the state has changed,
    brne PC+2
    rjmp active_state_delay_loop    ; if yes, go to state_change
    rjmp state_change   ; else stay in the state

 active_state_heating:
    rcall LCD_clear
    RELAY_ON ; Turn relay on when entering the state
 active_state_heating_loop:
    ; read and display temperature. if temperature > boiling_temperature go to
      add_teabag state.
    rcall read_temperature
    rcall write_temperature_lcd
    cpi a0,low(cooking_water_temp)  ; Compare low byte of read temperature with
       cooking_water_temp
    ldi a3,high(cooking_water_temp) ; a3 used as temporary storage
    cpc a1, a3                      ; Compare high byte
    brcc temperature_reached        ; go to temperature reached if temperature
      > cooking_water_temp
    rjmp active_state_heating_loop  ; loop otherwise
temperature_reached:
    ldi CURRENT_STATE, STATE_ADD_TEA_BAG
    RELAY_OFF
```

```
    rjmp state_change



 active_state_add_tea_bag:
    call LCD_clear
    SERVO1_INIT
    rcall ang_rot_cw                    ; run the motor to lower the teabag
    ldi CURRENT_STATE, STATE_TEA_STEEP  ; unconditional continuation to the
      tea_steep state
    rjmp state_change


 active_state_tea_steep:
    call LCD_clear
    clr b1                  ; b1 is initially 0, is set to 1 as soon as
      overflow1 happens
    PRESET_TIMER1 steep_time_timer1
    call timer_init         ; activates timer0 and timer1 overflow interrupt
    call idle_mode_init     ; activates idle mode of microcontroller
 active_state_tea_steep_loop:
    call display_time
    sleep                   ; starts idle mode
    nop                     ; important, without nop, the compare is done
      before timer1 changed the value of b1
    cpi b1, 0x00
    ; if b1=1 (overflow1 happend) stop waiting
    breq state_tea_steep_end
    OUTI TIMSK, 0           ; disable interrupt timer 0 and 1
    call idle_mode_off      ; desactivates idle mode
    ldi CURRENT_STATE, STATE_REMOVE_TEA_BAG
state_tea_steep_end:
    cpi CURRENT_STATE, STATE_TEA_STEEP  ; check if the state has changed,
    brne PC+2
    rjmp active_state_tea_steep_loop    ; if yes, go to state_change
    rjmp state_change                   ; else stay in the state


 active_state_remove_tea_bag:
    call LCD_clear
    SERVO1_INIT
    rcall ang_rot_ccw               ; run motor to remove the tea bag

    ldi CURRENT_STATE, STATE_IDLE   ; unconditional continuation to the idle
      state
    rjmp state_change

 active_state_idle:
    call LCD_clear
 active_state_idle_loop:
    PRINTF  LCD
```

```
    .db "TEA READY",CR,0
    call idle_mode_init     ; activates idle mode of microcontroller
    sleep                   ; starts idle mode
    cpi CURRENT_STATE, STATE_IDLE   ; check if the state has changed,
    breq active_state_idle_loop     ; if no, stay in the state
    call idle_mode_off              ; else desactivates idle mode
    rjmp state_change               ; and go to state_change
```

```
/*
 * temperature_sensor.asm
 *
 *  This is a driver for the DS18B20+ temperature sensor
 * content:
 * - read temperature routine
 * - write temperature to lcd routine
 */

  .include "wire1.asm" ; 1 wire software emulation

 read_temperature:
 /*
 * Read the temperature from the sensor. result in a0 and a1, using c0 as
 */
    rcall   wire1_init      ; initialize 1-wire(R) interface
    rcall   wire1_reset         ; send a reset pulse
    CA  wire1_write, skipROM    ; skip ROM identification
    CA  wire1_write, convertT   ; initiate temp conversion
    WAIT_MS 750                 ; wait 750 msec

    rcall   wire1_reset         ; send a reset pulse
    CA  wire1_write, skipROM
    CA  wire1_write, readScratchpad
    rcall   wire1_read          ; read temperature LSB
    mov c0,a0
    rcall wire1_read            ; read temperature MSB
    mov a1,a0
    mov a0,c0

    ret

write_temperature_lcd:
    PRINTF  LCD
    .db "temp=",FFRAC2+FSIGN,a,4,$42," C ",CR,0
    WAIT_MS 1000
    ret
```

```
/*
uart_settings.asm

This file manages the settings dialog over RS323

content:
- Macros related to uart
- sub routines used
-

*/


; === Macros ===

.macro UART_Read_Decimal_1Digit
    rcall   UART0_getc
    subi a0, 48 ; subtract asci code for zero
.endmacro


; asks the user with the message @0 to enter a 4 digit number. displays this
  number afterwards
.macro UART_DEMAND_4DIGIT; @0 string to display
    ldi zl, low(2*@0)
    ldi zh, high(2*@0)
    rcall uart_print_string          ; print message
    call uart_read_decimal_4digit    ; read decimal
    mov a0, c1
    rcall UART0_putc
    WAIT_MS 5
    mov a0, c0
    rcall UART0_putc
.endmacro


.macro SETTINGS_DECISION_DIALOGUE
// print text @0, go to @1 if user enters 0, go to @2 if user enters 1, go to
  @3 otherwise
// One option is to add the address specified at @3 right before the macro such
   that the dialog repeats itself until 0 or 1 was entered.
    ;  print text @0
    ldi zl, low(2*@0)
    ldi zh, high(2*@0)
    rcall uart_print_string
    ; read 0,1 or other and decide
    UART_Read_Decimal_1Digit
    cpi a0, 0x00 ; compare with zero
    breq @1 ; go to @1 if zero was entered
    cpi a0, 0x01 ; compare with one
    breq @2 ; go to @2 if one was entered
    // error string if neighter 0 nor 1 pressed
    ldi zl, low(2*string_error_settings)
```

```
    ldi zh, high(2*string_error_settings)
    rcall uart_print_string
    rjmp @3 ; go to @3 if something else was entered.
.endmacro

; === sub routines ===

uart_print_string: ; input: address of string in program memory stored in z
ldi a0, 0x0D ; new line
rcall   UART0_putc
uart_print_string_loop: ; loop until the end character 0x00 is reached
    lpm
    mov a0,r0
    rcall   UART0_putc
    WAIT_MS 100
    adiw zl,1
    cpi a0, 0x00
    brne uart_print_string_loop
    ret

init_uart:
    rcall   UART0_init  ; initialize UART
    ret

uart_read_decimal_4digit: ; Reads exactly 4 digits and stores result in c0 and c1, uses a0 and a1 to read variables
    UART_Read_Decimal_1Digit ; read first digit
    clr a1 ; as the digit is <10, the most significant byte is 0
    MUL10  ; Multiply by 1000
    MUL10
    Mul10
    mov c0, a0 ; move result to c0 and c1
    mov c1, a1
    UART_Read_Decimal_1Digit ; read 2nd digit
    clr a1 ; as the digit is <10, the most significant byte is 0
    MUL10 ; multiply by 100
    MUL10
    add c0, a0 ; add to the previously read value
    adc c1, a1
    UART_Read_Decimal_1Digit ; repeat for digit 3
    clr a1
    MUL10
    add c0, a0
    adc c1, a1
    UART_Read_Decimal_1Digit ; repeat for last digit
    clr a1
    add c0, a0
    adc c1, a1
    ret

.include "uart.asm" ; driver of the uart module

; === String storage ===
```

```
; all the strings used in this dialogue are stored here
.cseg
string_time_timer: .db "Please enter the wait duration in minutes (4 digits)!",0x0A, 0x00
string_delay_timer: .db "Please enter the steep time in minutes (4 digits)",0x0A, 0x00
string_end_settings: .db "Everything is set up. Press '1' to start the delay, press '0' or the stop button to abort and reset the machine.",0x0A, 0x00
string_error_settings: .db "Error: I didn't understand that.",0x0A, 0x00
string_skip_settings: .db "Welcome to the settings dialogue", 0x0A, "Press '0' to use default settings, press '1' to customize the machine.",0x0A, 0x00
string_default_choosen: .db "You chose the default settings. The program is starting.",0x0A, 0x00


; === uart settings main program ===
uart_settings_main:
    ; asks if the user wants to enter the settings dialogue or use the default settings
    SETTINGS_DECISION_DIALOGUE string_skip_settings, uart_settings_main_default_settings, uart_settings_main_setting_dialogue, uart_settings_main
uart_settings_main_default_settings: ; if the user decided to use default settings
    ldi CURRENT_STATE, STATE_DELAY ; go to the delay state
    ret
uart_settings_main_setting_dialogue: ; if the user decided to enter the settings dialogue
    UART_DEMAND_4DIGIT string_time_timer ; ask for steep time
    sts delay_time_timer1, a0 ; store read value
    sts delay_time_timer1+1, a1
    UART_DEMAND_4DIGIT string_delay_timer ; ask for delay timer
    sts steep_time_timer1, a0 ; store read value
    sts steep_time_timer1+1, a1
    CALCULATE_TIMER delay_time_Timer1 ; transform the input from minutes to 6s (timer interrupt interval)
    CALCULATE_TIMER steep_time_timer1 ; transform the input from minutes to 6s (timer interrupt interval)
    WAIT_MS 1000
uart_settings_end_text:
    ; asks if the user wants to start the machine or abort
    SETTINGS_DECISION_DIALOGUE string_end_settings, uart_settings_main_abort, uart_settings_main_start, uart_settings_end_text
uart_settings_main_abort: ; if the user choose to abort
    jmp reset
uart_settings_main_start: ; if the user choose to start the machine
    ldi CURRENT_STATE, STATE_DELAY
    ret
```

```
; file  uart.asm  target ATmega128L-4MHz-STK300
; purpose setup to two UART units 0 and 1

;.equ   baud0   = 19200
.equ    baud0   = 9600
.equ    _UBRR0  = clock/(16*baud0)-1

UART0_init: ; 19200-8-N-1

    ;ldi        r16, 12
    ;out        UBRR0L,r16
    ldi     r16, 0x00
    sts     UBRR0H,r16
    ldi     r16,0b00000010
    out     DDRE,r16
    clr r16
    ldi r16,(1<<TXEN0)|(1<<RXEN0)
    out UCSR0B,r16
    clr r16
    ldi r16,(1<<UCSZ01)+(1<<UCSZ00)
    sts UCSR0C, r16



    OUTI    UBRR0L, _UBRR0              ; set Baud rate
    ;OUTI   UBRR0H, 0x00
    ;OUTI   DDRE,0b00000010             ; make pin TX0 output
    ;OUTI   UCSR0B,(1<<TXEN0)+(1<<RXEN0)    ; Transmit/Receive Enable
    ;OUTI   UCSR0C, (1<<UCSZ01)+(1<<UCSZ00) ; 8-bit, 1 stop bit, parity
      disabled
    ret

UART0:
UART0_putc:
    sbis    UCSR0A,UDRE0                    ; wait for UART Data Register Empty
    rjmp    PC-1                        ; loop back if not empty
    out     UDR0,a0                     ; output character to UART Data
      Register
    ret

UART0_getc:
    sbis    UCSR0A,RXC0     ; wait for UART Receive Complete
    rjmp    PC-1        ; loop back if not complete
    in      a0,UDR0     ; read character to UART Data Register
    ret
```

```
; file  wire1.asm   target ATmega128L-4MHz-STK300
; purpose Dallas 1-wire(R) interface library

; === definitions ===
.equ    DQ_port = PORTB
.equ    DQ_pin  = DQ

.equ    DS18B20     = 0x28

.equ    readROM     = 0x33
.equ    matchROM    = 0x55
.equ    skipROM     = 0xcc
.equ    searchROM   = 0xf0
.equ    alarmSearch = 0xec

.equ    writeScratchpad = 0x4e
.equ    readScratchpad  = 0xbe
.equ    copyScratchpad  = 0x48
.equ    convertT    = 0x44
.equ    recallE2    = 0xb8
.equ    readPowerSupply = 0xb4

; === routines ===

.macro  WIRE1   ; t0,t1,t2
    sbi DQ_port-1,DQ_pin    ; pull DQ low (DDR=1 output)
    ldi w,(@0+1)/2
    rcall   wire1_wait      ; wait low time (t0)
    cbi DQ_port-1,DQ_pin    ; release DQ (DDR=0 input)
    ldi w,(@1+1)/2
    rcall   wire1_wait      ; wait high time (t1)
    in  w,DQ_port-2         ; sample line (PINx=PORTx-2)
    bst w,DQ_pin            ; store result in T
    ldi w,(@2+1)/2
    rcall   wire1_wait      ; wait separation time (t2)
    ret
    .endmacro

wire1_wait:
    dec w                   ; loop time 2usec
    nop
    nop
    nop
    nop
    nop
    brne    wire1_wait
    ret

wire1_init:
    cbi DQ_port,  DQ_pin    ; PORT=0 low (for pull-down)
    cbi DQ_port-1,DQ_pin    ; DDR=0 (input hi Z)
    ret
```

```
wire1_reset:    WIRE1   480,70,410
wire1_write0:   WIRE1   56,4,1
wire1_write1:   WIRE1   1,59,1
wire1_read1:    WIRE1   1,14,45

wire1_write:
    push    a1
    ldi a1,8
    ror a0

    brcc    PC+3                ; if C=1 then wire1, else wire0
    rcall   wire1_write1
    rjmp    PC+2
    rcall   wire1_write0

    DJNZ    a1,wire1_write+2    ; dec and jump if not zero
    pop a1
    ret

wire1_read:
    push    a1
    ldi a1,8
    ror a0
    rcall   wire1_read1         ; returns result in T
    bld a0,7                    ; move T to MSb
    DJNZ    a1,wire1_read+2     ; dec and jump if not zero
    pop a1
    ret

wire1_crc:
    ldi w,0b00011001
    ldi a2,8
crc1:   ror a0
    brcc    PC+2
    eor a1,w
    bst a1,0
    ror a1
    bld a1,7
    DJNZ    a2,crc1
    ret
```

DS18B20
Programmable Resolution 1-Wire Digital Thermometer

DESCRIPTION

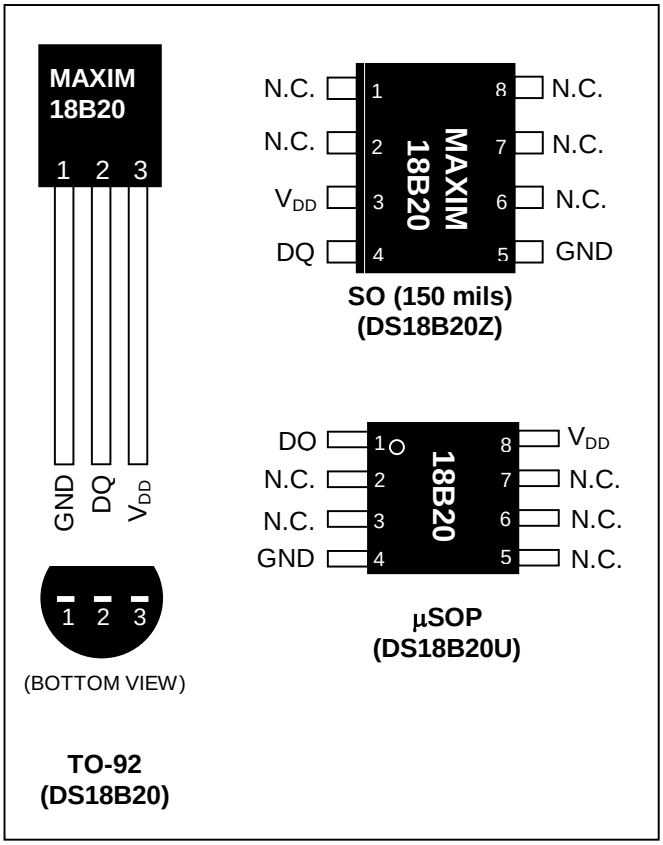
The DS18B20 digital thermometer provides 9-bit to 12-bit Celsius temperature measurements and has an alarm function with nonvolatile user-programmable upper and lower trigger points. The DS18B20 communicates over a 1-Wire bus that by definition requires only one data line (and ground) for communication with a central microprocessor. It has an operating temperature range of -55°C to +125°C and is accurate to ±0.5°C over the range of -10°C to +85°C. In addition, the DS18B20 can derive power directly from the data line ("parasite power"), eliminating the need for an external power supply.

Each DS18B20 has a unique 64-bit serial code, which allows multiple DS18B20s to function on the same 1-Wire bus. Thus, it is simple to use one microprocessor to control many DS18B20s distributed over a large area. Applications that can benefit from this feature include HVAC environmental controls, temperature monitoring systems inside buildings, equipment, or machinery, and process monitoring and control systems.

FEATURES

- Unique 1-Wire® Interface Requires Only One Port Pin for Communication
- Each Device has a Unique 64-Bit Serial Code Stored in an On-Board ROM
- Multidrop Capability Simplifies Distributed Temperature-Sensing Applications
- Requires No External Components
- Can Be Powered from Data Line; Power Supply Range is 3.0V to 5.5V
- Measures Temperatures from -55°C to +125°C (-67°F to +257°F)
- ±0.5°C Accuracy from -10°C to +85°C
- Thermometer Resolution is User Selectable from 9 to 12 Bits
- Converts Temperature to 12-Bit Digital Word in 750ms (Max)
- User-Definable Nonvolatile (NV) Alarm Settings
- Alarm Search Command Identifies and Addresses Devices Whose Temperature is Outside Programmed Limits (Temperature Alarm Condition)
- Available in 8-Pin SO (150 mils), 8-Pin µSOP, and 3-Pin TO-92 Packages
- Software Compatible with the DS1822
- Applications Include Thermostatic Controls, Industrial Systems, Consumer Products, Thermometers, or Any Thermally Sensitive System

PIN CONFIGURATIONS



1-Wire is a registered trademark of Maxim Integrated Products, Inc.

REV: 042208

ORDERING INFORMATION

PART	TEMP RANGE	PIN-PACKAGE	TOP MARK
DS18B20	-55°C to +125°C	3 TO-92	18B20
DS18B20+	-55°C to +125°C	3 TO-92	18B20
DS18B20/T&R	-55°C to +125°C	3 TO-92 (2000 Piece)	18B20
DS18B20+T&R	-55°C to +125°C	3 TO-92 (2000 Piece)	18B20
DS18B20-SL/T&R	-55°C to +125°C	3 TO-92 (2000 Piece)*	18B20
DS18B20-SL+T&R	-55°C to +125°C	3 TO-92 (2000 Piece)*	18B20
DS18B20U	-55°C to +125°C	8 µSOP	18B20
DS18B20U+	-55°C to +125°C	8 µSOP	18B20
DS18B20U/T&R	-55°C to +125°C	8 µSOP (3000 Piece)	18B20
DS18B20U+T&R	-55°C to +125°C	8 µSOP (3000 Piece)	18B20
DS18B20Z	-55°C to +125°C	8 SO	DS18B20
DS18B20Z+	-55°C to +125°C	8 SO	DS18B20
DS18B20Z/T&R	-55°C to +125°C	8 SO (2500 Piece)	DS18B20
DS18B20Z+T&R	-55°C to +125°C	8 SO (2500 Piece)	DS18B20

+Denotes a lead-free package. A "+" will appear on the top mark of lead-free packages.
T&R = Tape and reel.
**TO-92 packages in tape and reel can be ordered with straight or formed leads. Choose "SL" for straight leads. Bulk TO-92 orders are straight leads only.*

PIN DESCRIPTION

PIN			NAME	FUNCTION
SO	µSOP	TO-92		
1, 2, 6, 7, 8	2, 3, 5, 6, 7	—	N.C.	No Connection
3	8	3	V_{DD}	Optional V_{DD}. V_{DD} must be grounded for operation in parasite power mode.
4	1	2	DQ	Data Input/Output. Open-drain 1-Wire interface pin. Also provides power to the device when used in parasite power mode (see the *Powering the DS18B20* section.)
5	4	1	GND	Ground

OVERVIEW

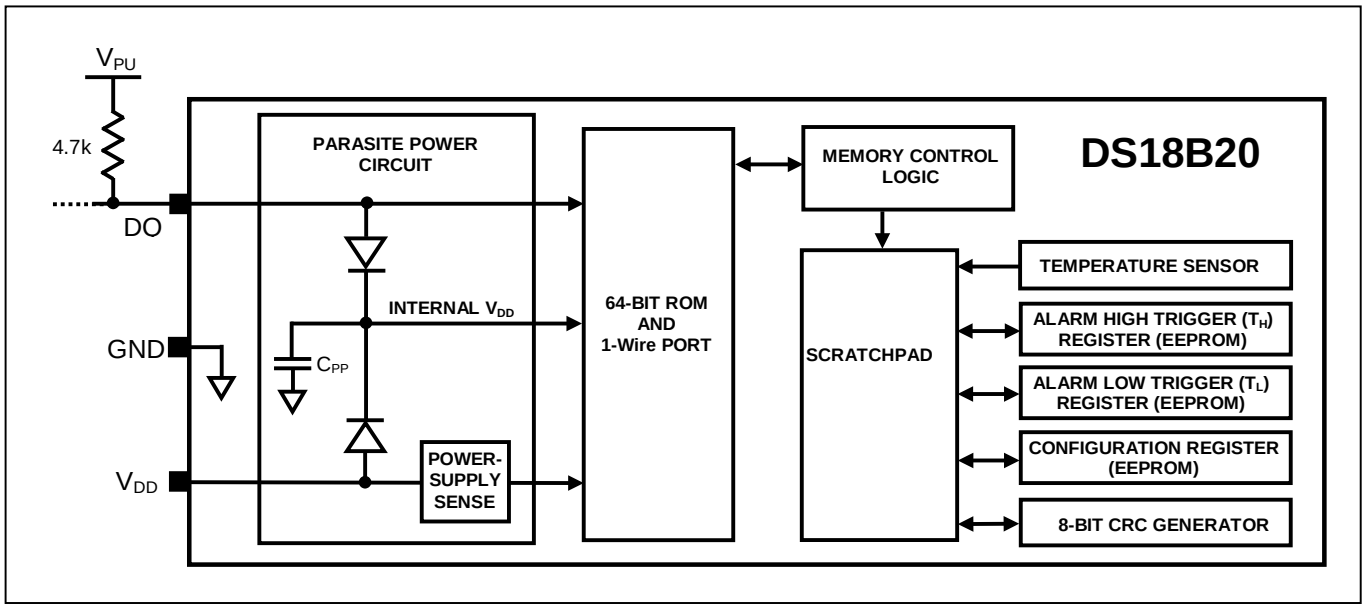
Figure 1 shows a block diagram of the DS18B20, and pin descriptions are given in the *Pin Description* table. The 64-bit ROM stores the device's unique serial code. The scratchpad memory contains the 2-byte temperature register that stores the digital output from the temperature sensor. In addition, the scratchpad provides access to the 1-byte upper and lower alarm trigger registers (T_H and T_L) and the 1-byte configuration register. The configuration register allows the user to set the resolution of the temperature-to-digital conversion to 9, 10, 11, or 12 bits. The T_H, T_L, and configuration registers are nonvolatile (EEPROM), so they will retain data when the device is powered down.

The DS18B20 uses Maxim's exclusive 1-Wire bus protocol that implements bus communication using one control signal. The control line requires a weak pullup resistor since all devices are linked to the bus via a 3-state or open-drain port (the DQ pin in the case of the DS18B20). In this bus system, the microprocessor (the master device) identifies and addresses devices on the bus using each device's unique 64-bit code. Because each device has a unique code, the number of devices that can be addressed on one

bus is virtually unlimited. The 1-Wire bus protocol, including detailed explanations of the commands and "time slots," is covered in the *1-Wire Bus System* section.

Another feature of the DS18B20 is the ability to operate without an external power supply. Power is instead supplied through the 1-Wire pullup resistor via the DQ pin when the bus is high. The high bus signal also charges an internal capacitor (C_{PP}), which then supplies power to the device when the bus is low. This method of deriving power from the 1-Wire bus is referred to as "parasite power." As an alternative, the DS18B20 may also be powered by an external supply on V_{DD}.

Figure 1. DS18B20 Block Diagram



OPERATION—MEASURING TEMPERATURE

The core functionality of the DS18B20 is its direct-to-digital temperature sensor. The resolution of the temperature sensor is user-configurable to 9, 10, 11, or 12 bits, corresponding to increments of 0.5°C, 0.25°C, 0.125°C, and 0.0625°C, respectively. The default resolution at power-up is 12-bit. The DS18B20 powers up in a low-power idle state. To initiate a temperature measurement and A-to-D conversion, the master must issue a Convert T [44h] command. Following the conversion, the resulting thermal data is stored in the 2-byte temperature register in the scratchpad memory and the DS18B20 returns to its idle state. If the DS18B20 is powered by an external supply, the master can issue "read time slots" (see the *1-Wire Bus System* section) after the Convert T command and the DS18B20 will respond by transmitting 0 while the temperature conversion is in progress and 1 when the conversion is done. If the DS18B20 is powered with parasite power, this notification technique cannot be used since the bus must be pulled high by a strong pullup during the entire temperature conversion. The bus requirements for parasite power are explained in detail in the *Powering the DS18B20* section.

The DS18B20 output temperature data is calibrated in degrees Celsius; for Fahrenheit applications, a lookup table or conversion routine must be used. The temperature data is stored as a 16-bit sign-extended two's complement number in the temperature register (see Figure 2). The sign bits (S) indicate if the temperature is positive or negative: for positive numbers S = 0 and for negative numbers S = 1. If the DS18B20 is configured for 12-bit resolution, all bits in the temperature register will contain valid data. For 11-bit resolution, bit 0 is undefined. For 10-bit resolution, bits 1 and 0 are undefined, and for 9-bit resolution bits 2, 1, and 0 are undefined. Table 1 gives examples of digital output data and the corresponding temperature reading for 12-bit resolution conversions.

Figure 2. Temperature Register Format

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
LS BYTE	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
	BIT 15	BIT 14	BIT 13	BIT 12	BIT 11	BIT 10	BIT 9	BIT 8
MS BYTE	S	S	S	S	S	2^6	2^5	2^4

S = SIGN

Table 1. Temperature/Data Relationship

TEMPERATURE (°C)	DIGITAL OUTPUT (BINARY)	DIGITAL OUTPUT (HEX)
+125	0000 0111 1101 0000	07D0h
+85*	0000 0101 0101 0000	0550h
+25.0625	0000 0001 1001 0001	0191h
+10.125	0000 0000 1010 0010	00A2h
+0.5	0000 0000 0000 1000	0008h
0	0000 0000 0000 0000	0000h
-0.5	1111 1111 1111 1000	FFF8h
-10.125	1111 1111 0101 1110	FF5Eh
-25.0625	1111 1110 0110 1111	FE6Fh
-55	1111 1100 1001 0000	FC90h

**The power-on reset value of the temperature register is +85°C.*

OPERATION—ALARM SIGNALING

After the DS18B20 performs a temperature conversion, the temperature value is compared to the user-defined two's complement alarm trigger values stored in the 1-byte T_H and T_L registers (see Figure 3). The sign bit (S) indicates if the value is positive or negative: for positive numbers S = 0 and for negative numbers S = 1. The T_H and T_L registers are nonvolatile (EEPROM) so they will retain data when the device is powered down. T_H and T_L can be accessed through bytes 2 and 3 of the scratchpad as explained in the *Memory* section.

Figure 3. T_H and T_L Register Format

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
S	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Only bits 11 through 4 of the temperature register are used in the T_H and T_L comparison since T_H and T_L are 8-bit registers. If the measured temperature is lower than or equal to T_L or higher than or equal to T_H, an alarm condition exists and an alarm flag is set inside the DS18B20. This flag is updated after every temperature measurement; therefore, if the alarm condition goes away, the flag will be turned off after the next temperature conversion.

The master device can check the alarm flag status of all DS18B20s on the bus by issuing an Alarm Search [ECh] command. Any DS18B20s with a set alarm flag will respond to the command, so the master can determine exactly which DS18B20s have experienced an alarm condition. If an alarm condition exists and the T_H or T_L settings have changed, another temperature conversion should be done to validate the alarm condition.

POWERING THE DS18B20

The DS18B20 can be powered by an external supply on the V_{DD} pin, or it can operate in "parasite power" mode, which allows the DS18B20 to function without a local external supply. Parasite power is very useful for applications that require remote temperature sensing or that are very space constrained. Figure 1 shows the DS18B20's parasite-power control circuitry, which "steals" power from the 1-Wire bus via the DQ pin when the bus is high. The stolen charge powers the DS18B20 while the bus is high, and some of the charge is stored on the parasite power capacitor (C_{PP}) to provide power when the bus is low. When the DS18B20 is used in parasite power mode, the V_{DD} pin must be connected to ground.

In parasite power mode, the 1-Wire bus and C_{PP} can provide sufficient current to the DS18B20 for most operations as long as the specified timing and voltage requirements are met (see the *DC Electrical Characteristics* and *AC Electrical Characteristics*). However, when the DS18B20 is performing temperature conversions or copying data from the scratchpad memory to EEPROM, the operating current can be as high as 1.5mA. This current can cause an unacceptable voltage drop across the weak 1-Wire pullup resistor and is more current than can be supplied by C_{PP}. To assure that the DS18B20 has sufficient supply current, it is necessary to provide a strong pullup on the 1-Wire bus whenever temperature conversions are taking place or data is being copied from the scratchpad to EEPROM. This can be accomplished by using a MOSFET to pull the bus directly to the rail as shown in Figure 4. The 1-Wire bus must be switched to the strong pullup within 10µs (max) after a Convert T [44h] or Copy Scratchpad [48h] command is issued, and the bus must be held high by the pullup for the duration of the conversion (t_{CONV}) or data transfer (t_{WR} = 10ms). No other activity can take place on the 1-Wire bus while the pullup is enabled.

The DS18B20 can also be powered by the conventional method of connecting an external power supply to the V_{DD} pin, as shown in Figure 5. The advantage of this method is that the MOSFET pullup is not required, and the 1-Wire bus is free to carry other traffic during the temperature conversion time.

The use of parasite power is not recommended for temperatures above +100°C since the DS18B20 may not be able to sustain communications due to the higher leakage currents that can exist at these temperatures. For applications in which such temperatures are likely, it is strongly recommended that the DS18B20 be powered by an external power supply.

In some situations the bus master may not know whether the DS18B20s on the bus are parasite powered or powered by external supplies. The master needs this information to determine if the strong bus pullup should be used during temperature conversions. To get this information, the master can issue a Skip ROM [CCh] command followed by a Read Power Supply [B4h] command followed by a "read time slot". During the read time slot, parasite powered DS18B20s will pull the bus low, and externally powered DS18B20s will let the bus remain high. If the bus is pulled low, the master knows that it must supply the strong pullup on the 1-Wire bus during temperature conversions.

Figure 4. Supplying the Parasite-Powered DS18B20 During Temperature Conversions

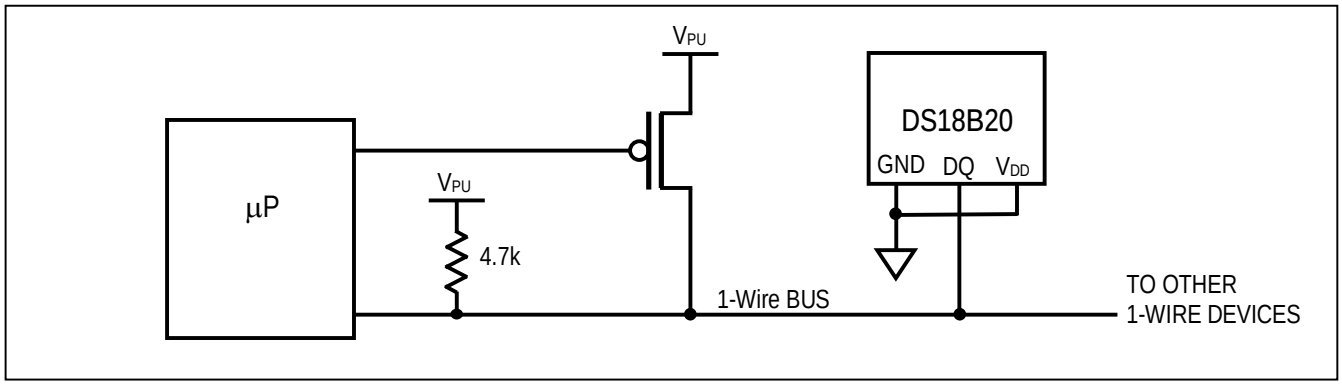
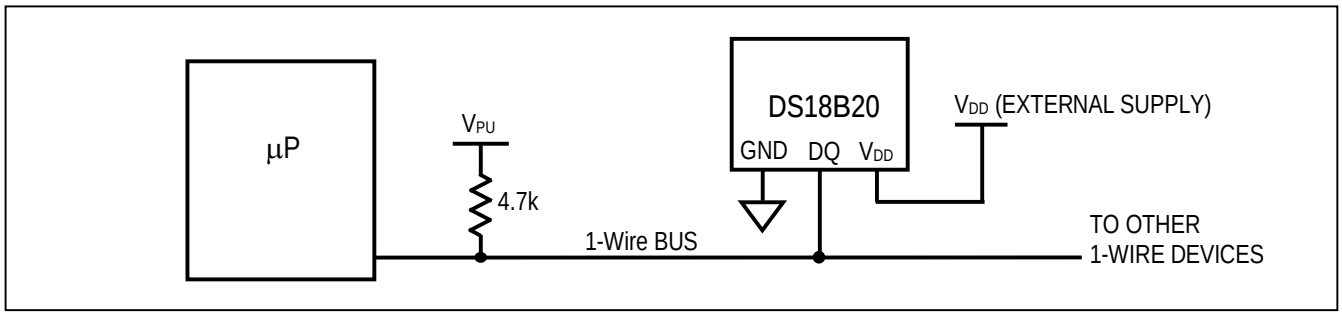


Figure 5. Powering the DS18B20 with an External Supply



64-BIT LASERED ROM CODE

Each DS18B20 contains a unique 64–bit code (see Figure 6) stored in ROM. The least significant 8 bits of the ROM code contain the DS18B20's 1-Wire family code: 28h. The next 48 bits contain a unique serial number. The most significant 8 bits contain a cyclic redundancy check (CRC) byte that is calculated from the first 56 bits of the ROM code. A detailed explanation of the CRC bits is provided in the *CRC Generation* section. The 64-bit ROM code and associated ROM function control logic allow the DS18B20 to operate as a 1-Wire device using the protocol detailed in the *1-Wire Bus System* section.

Figure 6. 64-Bit Lasered ROM Code

8-BIT CRC		48-BIT SERIAL NUMBER		8-BIT FAMILY CODE (28h)	
MSB	LSB	MSB	LSB	MSB	LSB

MEMORY

The DS18B20's memory is organized as shown in Figure 7. The memory consists of an SRAM scratchpad with nonvolatile EEPROM storage for the high and low alarm trigger registers (T_H and T_L) and configuration register. Note that if the DS18B20 alarm function is not used, the T_H and T_L registers can serve as general-purpose memory. All memory commands are described in detail in the *DS18B20 Function Commands* section.

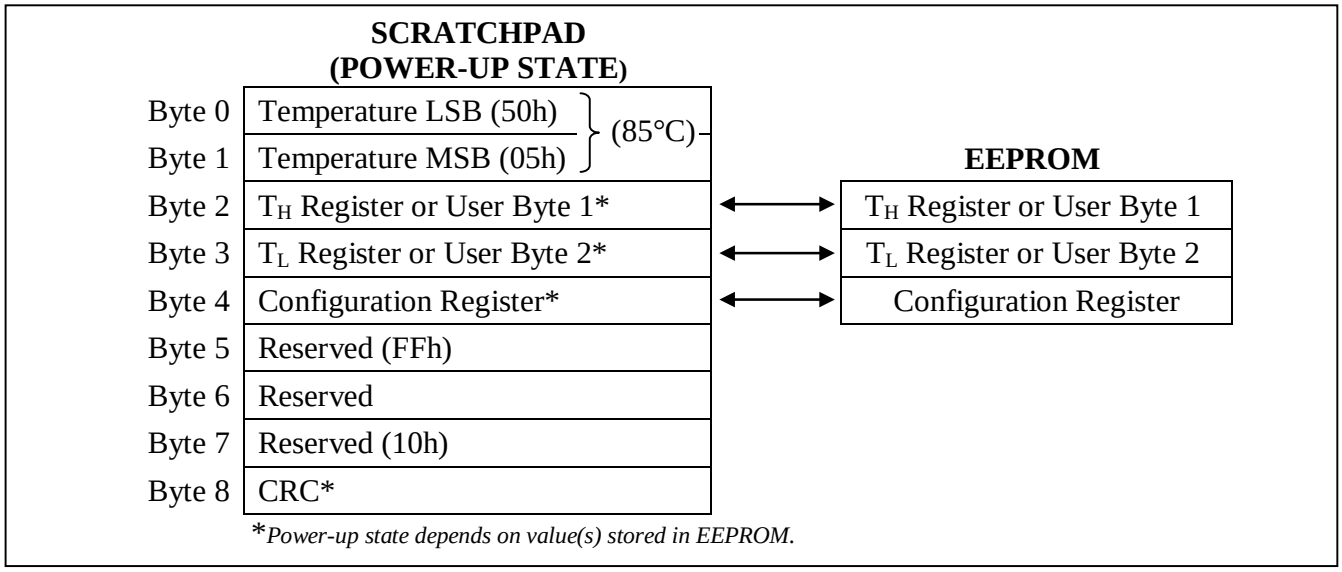
Byte 0 and byte 1 of the scratchpad contain the LSB and the MSB of the temperature register, respectively. These bytes are read-only. Bytes 2 and 3 provide access to T_H and T_L registers. Byte 4 contains the configuration register data, which is explained in detail in the *Configuration Register* section. Bytes 5, 6, and 7 are reserved for internal use by the device and cannot be overwritten.

Byte 8 of the scratchpad is read-only and contains the CRC code for bytes 0 through 7 of the scratchpad. The DS18B20 generates this CRC using the method described in the *CRC Generation* section.

Data is written to bytes 2, 3, and 4 of the scratchpad using the Write Scratchpad [4Eh] command; the data must be transmitted to the DS18B20 starting with the least significant bit of byte 2. To verify data integrity, the scratchpad can be read (using the Read Scratchpad [BEh] command) after the data is written. When reading the scratchpad, data is transferred over the 1-Wire bus starting with the least significant bit of byte 0. To transfer the T_H, T_L and configuration data from the scratchpad to EEPROM, the master must issue the Copy Scratchpad [48h] command.

Data in the EEPROM registers is retained when the device is powered down; at power-up the EEPROM data is reloaded into the corresponding scratchpad locations. Data can also be reloaded from EEPROM to the scratchpad at any time using the Recall E^2 [B8h] command. The master can issue read time slots following the Recall E^2 command and the DS18B20 will indicate the status of the recall by transmitting 0 while the recall is in progress and 1 when the recall is done.

Figure 7. DS18B20 Memory Map



CONFIGURATION REGISTER

Byte 4 of the scratchpad memory contains the configuration register, which is organized as illustrated in Figure 8**.** The user can set the conversion resolution of the DS18B20 using the R0 and R1 bits in this register as shown in Table 2. The power-up default of these bits is R0 = 1 and R1 = 1 (12-bit resolution). Note that there is a direct tradeoff between resolution and conversion time. Bit 7 and bits 0 to 4 in the configuration register are reserved for internal use by the device and cannot be overwritten.

Figure 8. Configuration Register

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
0	R1	R0	1	1	1	1	1

Table 2. Thermometer Resolution Configuration

R1	R0	RESOLUTION (BITS)	MAX CONVERSION TIME	
0	0	9	93.75ms	($t_{CONV}/8$)
0	1	10	187.5ms	($t_{CONV}/4$)
1	0	11	375ms	($t_{CONV}/2$)
1	1	12	750ms	(t_{CONV})

CRC GENERATION

CRC bytes are provided as part of the DS18B20's 64-bit ROM code and in the 9th byte of the scratchpad memory. The ROM code CRC is calculated from the first 56 bits of the ROM code and is contained in the most significant byte of the ROM. The scratchpad CRC is calculated from the data stored in the scratchpad, and therefore it changes when the data in the scratchpad changes. The CRCs provide the bus master with a method of data validation when data is read from the DS18B20. To verify that data has been read correctly, the bus master must re-calculate the CRC from the received data and then compare this value to either the ROM code CRC (for ROM reads) or to the scratchpad CRC (for scratchpad reads). If the calculated CRC matches the read CRC, the data has been received error free. The comparison of CRC values and the decision to continue with an operation are determined entirely by the bus master. There is no circuitry inside the DS18B20 that prevents a command sequence from proceeding if the DS18B20 CRC (ROM or scratchpad) does not match the value generated by the bus master.

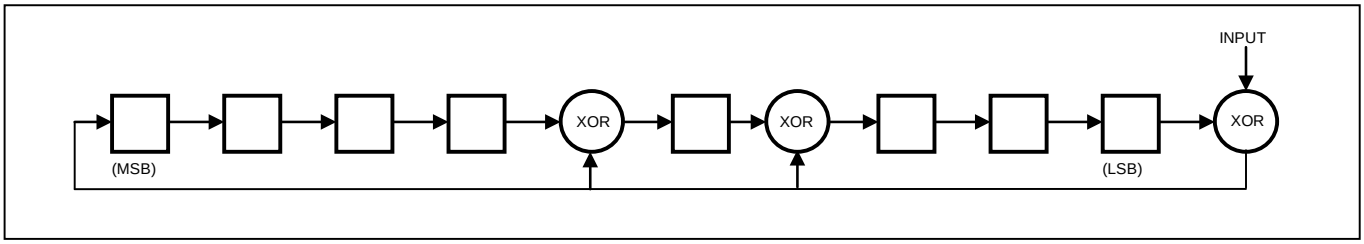
The equivalent polynomial function of the CRC (ROM or scratchpad) is:

$$CRC = X^8 + X^5 + X^4 + 1$$

The bus master can re-calculate the CRC and compare it to the CRC values from the DS18B20 using the polynomial generator shown in Figure 9. This circuit consists of a shift register and XOR gates, and the shift register bits are initialized to 0. Starting with the least significant bit of the ROM code or the least significant bit of byte 0 in the scratchpad, one bit at a time should shifted into the shift register. After shifting in the 56th bit from the ROM or the most significant bit of byte 7 from the scratchpad, the polynomial generator will contain the re-calculated CRC. Next, the 8-bit ROM code or scratchpad CRC from the DS18B20 must be shifted into the circuit. At this point, if the re-calculated CRC was correct, the shift register will contain all 0s. Additional information about the Maxim 1-Wire cyclic redundancy check

is available in *Application Note 27: Understanding and Using Cyclic Redundancy Checks with Maxim iButton Products*.

Figure 9. CRC Generator



1-WIRE BUS SYSTEM

The 1-Wire bus system uses a single bus master to control one or more slave devices. The DS18B20 is always a slave. When there is only one slave on the bus, the system is referred to as a "single-drop" system; the system is "multidrop" if there are multiple slaves on the bus.

All data and commands are transmitted least significant bit first over the 1-Wire bus.

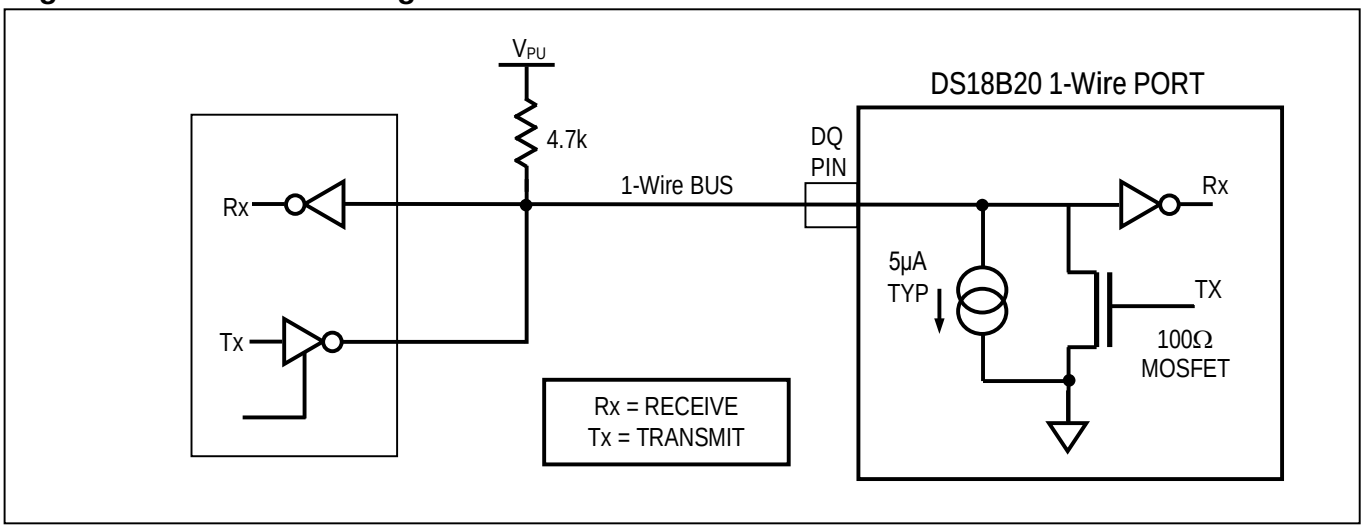
The following discussion of the 1-Wire bus system is broken down into three topics: hardware configuration, transaction sequence, and 1-Wire signaling (signal types and timing).

HARDWARE CONFIGURATION

The 1-Wire bus has by definition only a single data line. Each device (master or slave) interfaces to the data line via an open-drain or 3-state port. This allows each device to "release" the data line when the device is not transmitting data so the bus is available for use by another device. The 1-Wire port of the DS18B20 (the DQ pin) is open drain with an internal circuit equivalent to that shown in Figure 10.

The 1-Wire bus requires an external pullup resistor of approximately 5kΩ; thus, the idle state for the 1-Wire bus is high. If for any reason a transaction needs to be suspended, the bus MUST be left in the idle state if the transaction is to resume. Infinite recovery time can occur between bits so long as the 1-Wire bus is in the inactive (high) state during the recovery period. If the bus is held low for more than 480µs, all components on the bus will be reset.

Figure 10. Hardware Configuration



TRANSACTION SEQUENCE

The transaction sequence for accessing the DS18B20 is as follows:

Step 1. Initialization

Step 2. ROM Command (followed by any required data exchange)

Step 3. DS18B20 Function Command (followed by any required data exchange)

It is very important to follow this sequence every time the DS18B20 is accessed, as the DS18B20 will not respond if any steps in the sequence are missing or out of order. Exceptions to this rule are the Search ROM [F0h] and Alarm Search [ECh] commands. After issuing either of these ROM commands, the master must return to Step 1 in the sequence.

INITIALIZATION

All transactions on the 1-Wire bus begin with an initialization sequence. The initialization sequence consists of a reset pulse transmitted by the bus master followed by presence pulse(s) transmitted by the slave(s). The presence pulse lets the bus master know that slave devices (such as the DS18B20) are on the bus and are ready to operate. Timing for the reset and presence pulses is detailed in the *1-Wire Signaling* section.

ROM COMMANDS

After the bus master has detected a presence pulse, it can issue a ROM command. These commands operate on the unique 64-bit ROM codes of each slave device and allow the master to single out a specific device if many are present on the 1-Wire bus. These commands also allow the master to determine how many and what types of devices are present on the bus or if any device has experienced an alarm condition. There are five ROM commands, and each command is 8 bits long. The master device must issue an appropriate ROM command before issuing a DS18B20 function command. A flowchart for operation of the ROM commands is shown in Figure 11.

SEARCH ROM [F0h]

When a system is initially powered up, the master must identify the ROM codes of all slave devices on the bus, which allows the master to determine the number of slaves and their device types. The master learns the ROM codes through a process of elimination that requires the master to perform a Search ROM cycle (i.e., Search ROM command followed by data exchange) as many times as necessary to identify all of the slave devices. If there is only one slave on the bus, the simpler Read ROM command (see below) can be used in place of the Search ROM process. For a detailed explanation of the Search ROM procedure, refer to the *iButton® Book of Standards* at www.maxim-ic.com/ibuttonbook. After every Search ROM cycle, the bus master must return to Step 1 (Initialization) in the transaction sequence.

READ ROM [33h]

This command can only be used when there is one slave on the bus. It allows the bus master to read the slave's 64-bit ROM code without using the Search ROM procedure. If this command is used when there is more than one slave present on the bus, a data collision will occur when all the slaves attempt to respond at the same time.

MATCH ROM [55h]

The match ROM command followed by a 64-bit ROM code sequence allows the bus master to address a specific slave device on a multidrop or single-drop bus. Only the slave that exactly matches the 64-bit ROM code sequence will respond to the function command issued by the master; all other slaves on the bus will wait for a reset pulse.

iButton is a registered trademark of Maxim Integrated Products, Inc.

SKIP ROM [CCh]

The master can use this command to address all devices on the bus simultaneously without sending out any ROM code information. For example, the master can make all DS18B20s on the bus perform simultaneous temperature conversions by issuing a Skip ROM command followed by a Convert T [44h] command.

Note that the Read Scratchpad [BEh] command can follow the Skip ROM command only if there is a single slave device on the bus. In this case, time is saved by allowing the master to read from the slave without sending the device's 64-bit ROM code. A Skip ROM command followed by a Read Scratchpad command will cause a data collision on the bus if there is more than one slave since multiple devices will attempt to transmit data simultaneously.

ALARM SEARCH [ECh]

The operation of this command is identical to the operation of the Search ROM command except that only slaves with a set alarm flag will respond. This command allows the master device to determine if any DS18B20s experienced an alarm condition during the most recent temperature conversion. After every Alarm Search cycle (i.e., Alarm Search command followed by data exchange), the bus master must return to Step 1 (Initialization) in the transaction sequence. See the *Operation—Alarm Signaling* section for an explanation of alarm flag operation.

DS18B20 FUNCTION COMMANDS

After the bus master has used a ROM command to address the DS18B20 with which it wishes to communicate, the master can issue one of the DS18B20 function commands. These commands allow the master to write to and read from the DS18B20's scratchpad memory, initiate temperature conversions and determine the power supply mode. The DS18B20 function commands, which are described below, are summarized in Table 3 and illustrated by the flowchart in Figure 12.

CONVERT T [44h]

This command initiates a single temperature conversion. Following the conversion, the resulting thermal data is stored in the 2-byte temperature register in the scratchpad memory and the DS18B20 returns to its low-power idle state. If the device is being used in parasite power mode, within 10μs (max) after this command is issued the master must enable a strong pullup on the 1-Wire bus for the duration of the conversion (t_{CONV}) as described in the *Powering the DS18B20* section. If the DS18B20 is powered by an external supply, the master can issue read time slots after the Convert T command and the DS18B20 will respond by transmitting a 0 while the temperature conversion is in progress and a 1 when the conversion is done. In parasite power mode this notification technique cannot be used since the bus is pulled high by the strong pullup during the conversion.

WRITE SCRATCHPAD [4Eh]

This command allows the master to write 3 bytes of data to the DS18B20's scratchpad. The first data byte is written into the T_H register (byte 2 of the scratchpad), the second byte is written into the T_L register (byte 3), and the third byte is written into the configuration register (byte 4). Data must be transmitted least significant bit first. All three bytes MUST be written before the master issues a reset, or the data may be corrupted.

READ SCRATCHPAD [BEh]

This command allows the master to read the contents of the scratchpad. The data transfer starts with the least significant bit of byte 0 and continues through the scratchpad until the 9th byte (byte 8 – CRC) is read. The master may issue a reset to terminate reading at any time if only part of the scratchpad data is needed.

COPY SCRATCHPAD [48h]

This command copies the contents of the scratchpad T_H, T_L and configuration registers (bytes 2, 3 and 4) to EEPROM. If the device is being used in parasite power mode, within 10μs (max) after this command is issued the master must enable a strong pullup on the 1-Wire bus for at least 10ms as described in the *Powering the DS18B20* section.

RECALL E^2 [B8h]

This command recalls the alarm trigger values (T_H and T_L) and configuration data from EEPROM and places the data in bytes 2, 3, and 4, respectively, in the scratchpad memory. The master device can issue read time slots following the Recall E^2 command and the DS18B20 will indicate the status of the recall by transmitting 0 while the recall is in progress and 1 when the recall is done. The recall operation happens automatically at power-up, so valid data is available in the scratchpad as soon as power is applied to the device.

READ POWER SUPPLY [B4h]

The master device issues this command followed by a read time slot to determine if any DS18B20s on the bus are using parasite power. During the read time slot, parasite powered DS18B20s will pull the bus low, and externally powered DS18B20s will let the bus remain high. See the *Powering the DS18B20* section for usage information for this command.

Table 3. DS18B20 Function Command Set

COMMAND	DESCRIPTION	PROTOCOL	1-Wire BUS ACTIVITYAFTER COMMAND IS ISSUED	NOTES
TEMPERATURE CONVERSION COMMANDS				
Convert T	Initiates temperature conversion.	44h	DS18B20 transmits conversion status to master (not applicable for parasite-powered DS18B20s).	1
MEMORY COMMANDS				
Read Scratchpad	Reads the entire scratchpad including the CRC byte.	BEh	DS18B20 transmits up to 9 data bytes to master.	2
Write Scratchpad	Writes data into scratchpad bytes 2, 3, and 4 (T_H, T_L, and configuration registers).	4Eh	Master transmits 3 data bytes to DS18B20.	3
Copy Scratchpad	Copies T_H, T_L, and configuration register data from the scratchpad to EEPROM.	48h	None	1
Recall E^2	Recalls T_H, T_L, and configuration register data from EEPROM to the scratchpad.	B8h	DS18B20 transmits recall status to master.	
Read Power Supply	Signals DS18B20 power supply mode to the master.	B4h	DS18B20 transmits supply status to master.	

Note 1: For parasite-powered DS18B20s, the master must enable a strong pullup on the 1-Wire bus during temperature conversions and copies from the scratchpad to EEPROM. No other bus activity may take place during this time.
Note 2: The master can interrupt the transmission of data at any time by issuing a reset.
Note 3: All three bytes must be written before a reset is issued.

Figure 11. ROM Commands Flowchart

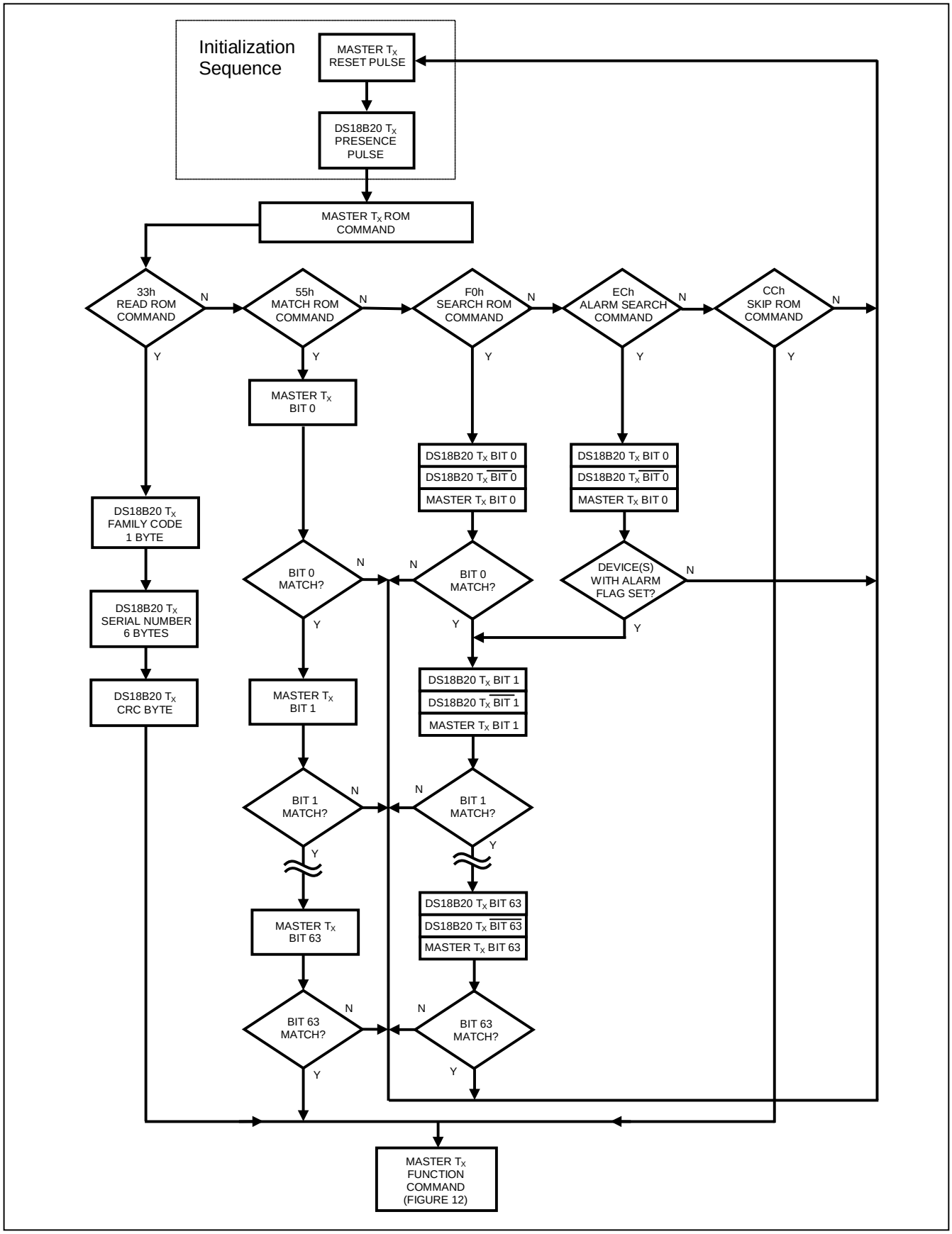
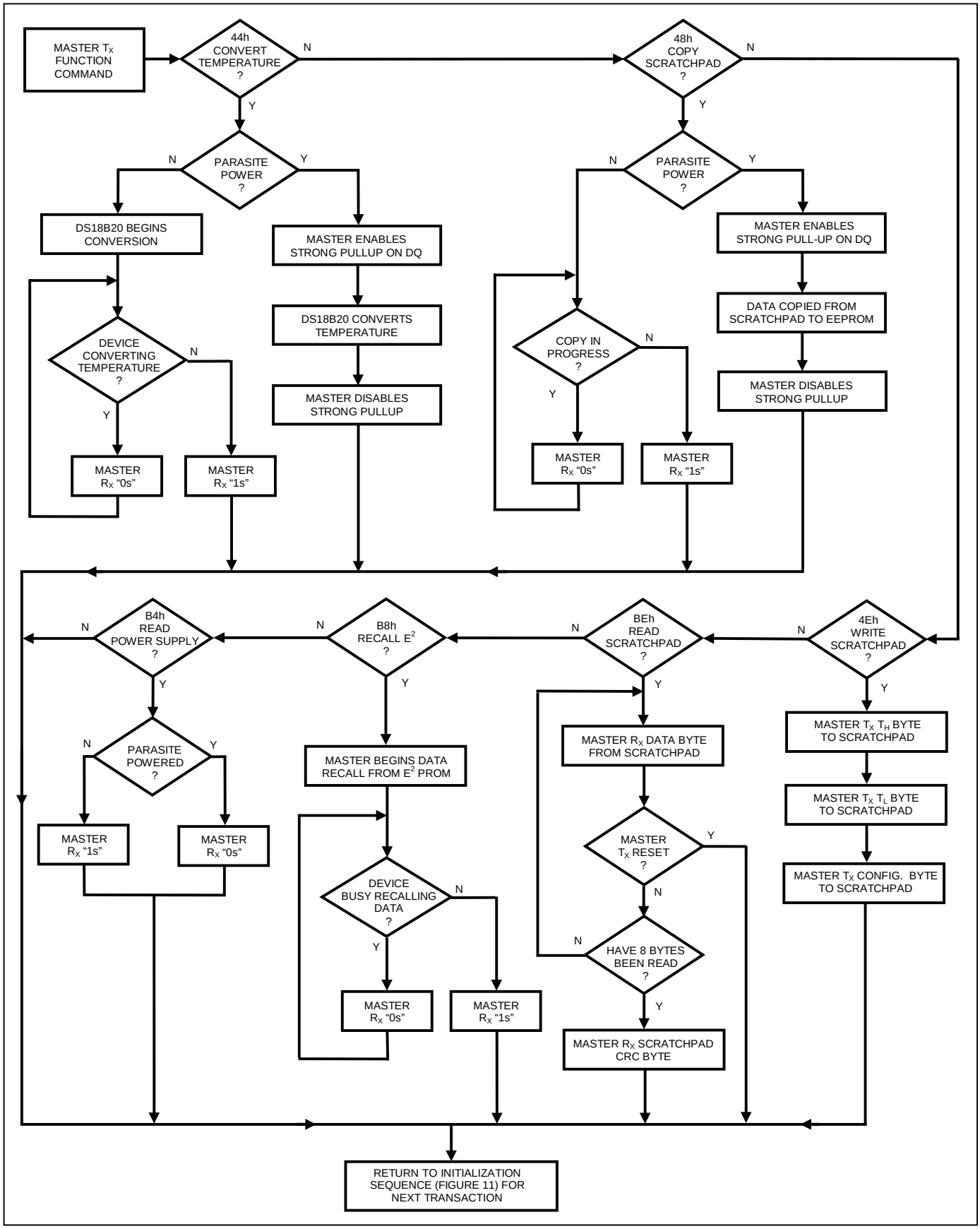


Figure 12. DS18B20 Function Commands Flowchart



1-WIRE SIGNALING

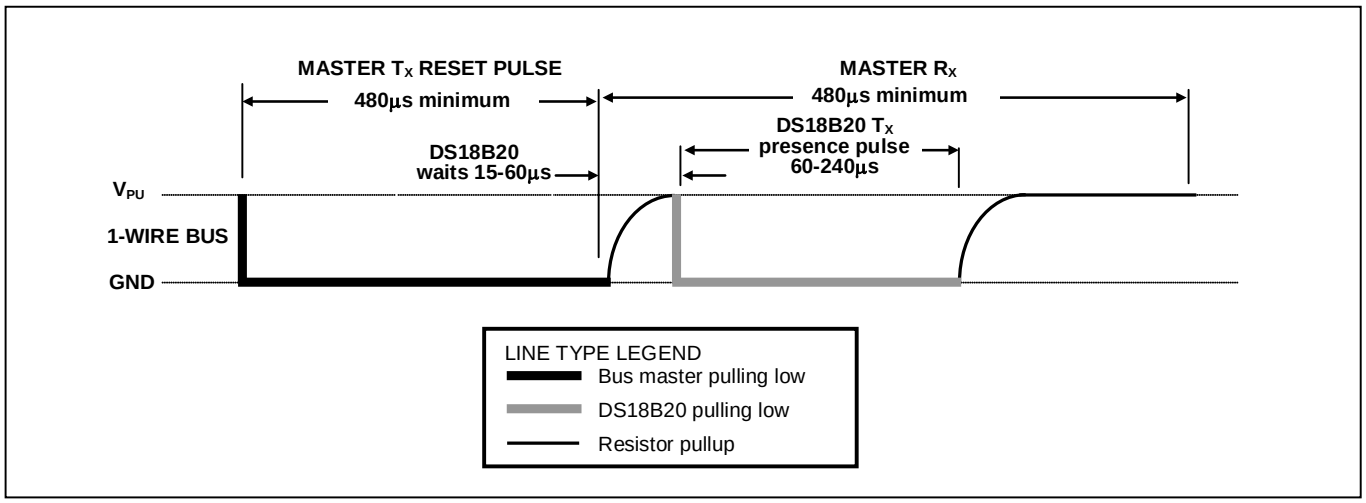
The DS18B20 uses a strict 1-Wire communication protocol to ensure data integrity. Several signal types are defined by this protocol: reset pulse, presence pulse, write 0, write 1, read 0, and read 1. The bus master initiates all these signals, with the exception of the presence pulse.

INITIALIZATION PROCEDURE—RESET AND PRESENCE PULSES

All communication with the DS18B20 begins with an initialization sequence that consists of a reset pulse from the master followed by a presence pulse from the DS18B20. This is illustrated in Figure 13. When the DS18B20 sends the presence pulse in response to the reset, it is indicating to the master that it is on the bus and ready to operate.

During the initialization sequence the bus master transmits (T_X) the reset pulse by pulling the 1-Wire bus low for a minimum of 480µs. The bus master then releases the bus and goes into receive mode (R_X). When the bus is released, the 5kΩ pullup resistor pulls the 1-Wire bus high. When the DS18B20 detects this rising edge, it waits 15µs to 60µs and then transmits a presence pulse by pulling the 1-Wire bus low for 60µs to 240µs.

Figure 13. Initialization Timing



READ/WRITE TIME SLOTS

The bus master writes data to the DS18B20 during write time slots and reads data from the DS18B20 during read time slots. One bit of data is transmitted over the 1-Wire bus per time slot.

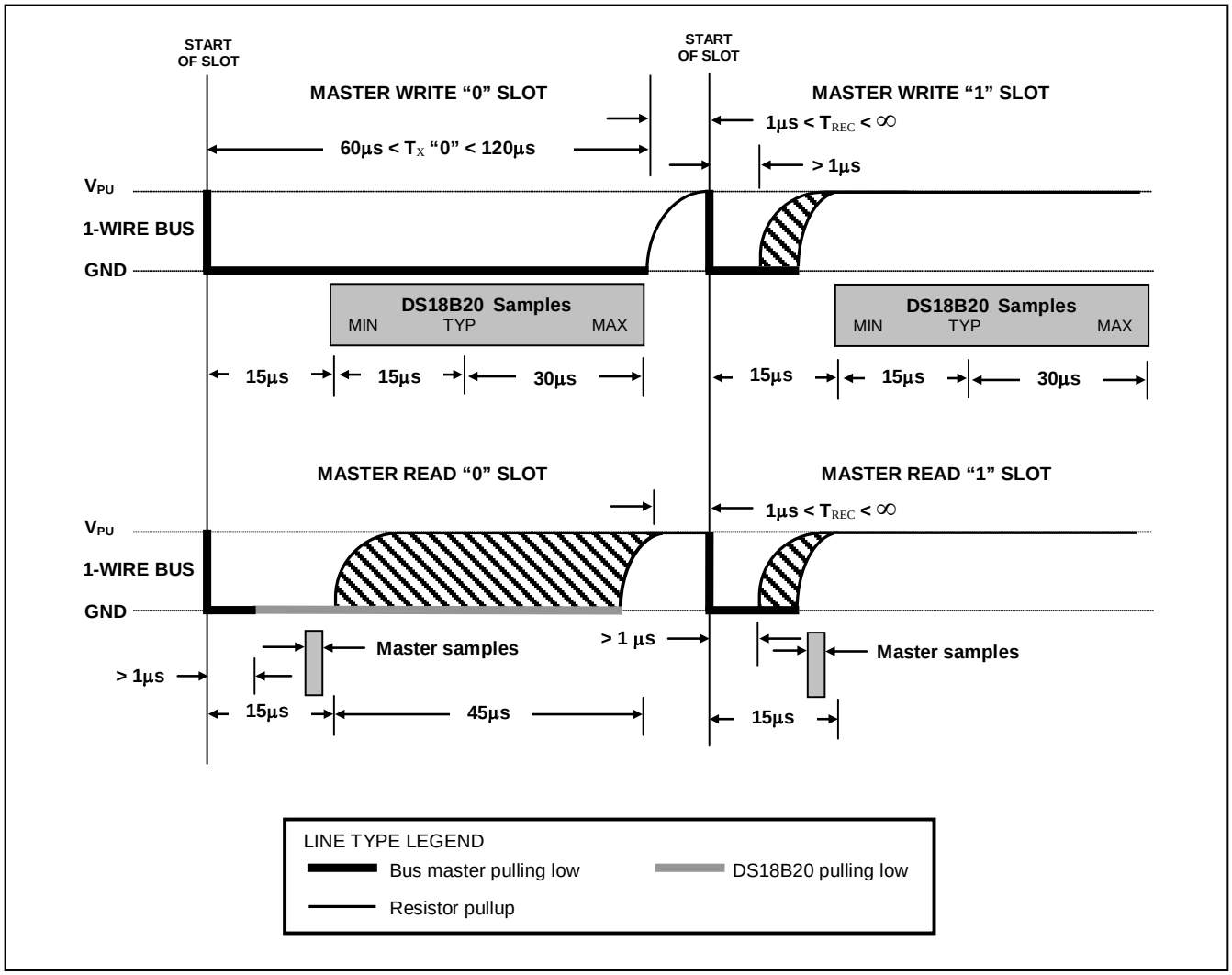
WRITE TIME SLOTS

There are two types of write time slots: “Write 1” time slots and “Write 0” time slots. The bus master uses a Write 1 time slot to write a logic 1 to the DS18B20 and a Write 0 time slot to write a logic 0 to the DS18B20. All write time slots must be a minimum of 60µs in duration with a minimum of a 1µs recovery time between individual write slots. Both types of write time slots are initiated by the master pulling the 1-Wire bus low (see Figure 14).

To generate a Write 1 time slot, after pulling the 1-Wire bus low, the bus master must release the 1-Wire bus within 15µs. When the bus is released, the 5kΩ pullup resistor will pull the bus high. To generate a Write 0 time slot, after pulling the 1-Wire bus low, the bus master must continue to hold the bus low for the duration of the time slot (at least 60µs).

The DS18B20 samples the 1-Wire bus during a window that lasts from 15µs to 60µs after the master initiates the write time slot. If the bus is high during the sampling window, a 1 is written to the DS18B20. If the line is low, a 0 is written to the DS18B20.

Figure 14. Read/Write Time Slot Timing Diagram



READ TIME SLOTS

The DS18B20 can only transmit data to the master when the master issues read time slots. Therefore, the master must generate read time slots immediately after issuing a Read Scratchpad [BEh] or Read Power Supply [B4h] command, so that the DS18B20 can provide the requested data. In addition, the master can generate read time slots after issuing Convert T [44h] or Recall E^2 [B8h] commands to find out the status of the operation as explained in the *DS18B20 Function Commands* section.

All read time slots must be a minimum of 60µs in duration with a minimum of a 1µs recovery time between slots. A read time slot is initiated by the master device pulling the 1-Wire bus low for a minimum of 1µs and then releasing the bus (see Figure 14). After the master initiates the read time slot, the DS18B20 will begin transmitting a 1 or 0 on bus. The DS18B20 transmits a 1 by leaving the bus high and transmits a 0 by pulling the bus low. When transmitting a 0, the DS18B20 will release the bus by the end of the time slot, and the bus will be pulled back to its high idle state by the pullup resister. Output

data from the DS18B20 is valid for 15µs after the falling edge that initiated the read time slot. Therefore, the master must release the bus and then sample the bus state within 15µs from the start of the slot.

Figure 15 illustrates that the sum of T_{INIT}, T_{RC}, and T_{SAMPLE} must be less than 15µs for a read time slot. Figure 16 shows that system timing margin is maximized by keeping T_{INIT} and T_{RC} as short as possible and by locating the master sample time during read time slots towards the end of the 15µs period.

Figure 15. Detailed Master Read 1 Timing

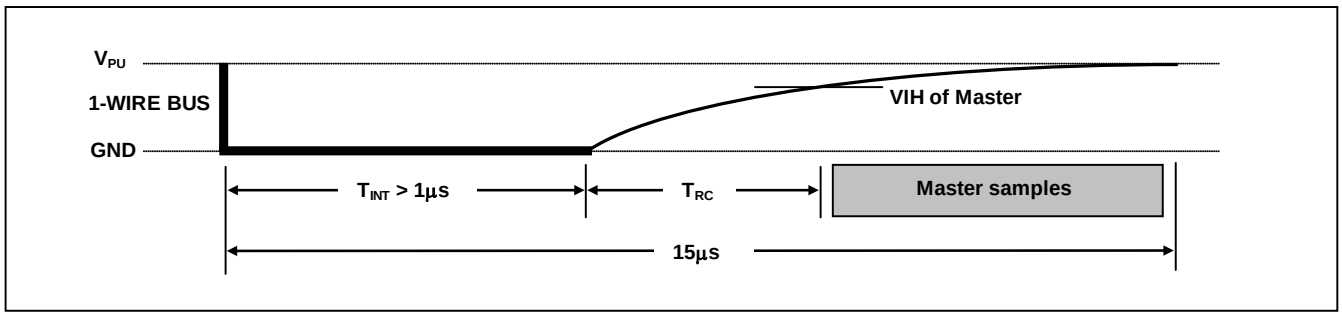
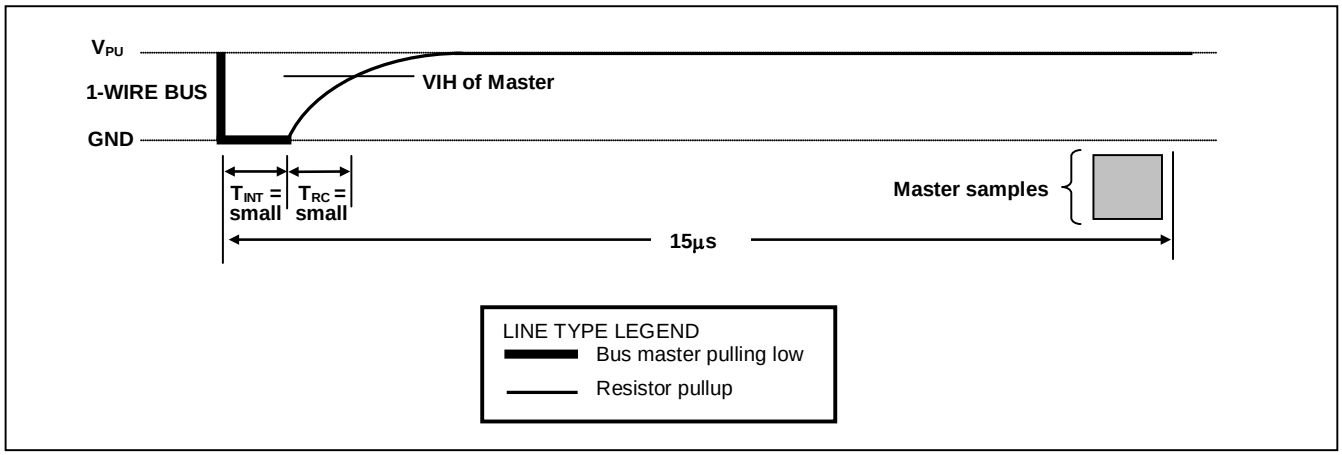


Figure 16. Recommended Master Read 1 Timing



RELATED APPLICATION NOTES

The following application notes can be applied to the DS18B20 and are available on our website at www.maxim-ic.com.

Application Note 27: Understanding and Using Cyclic Redundancy Checks with Maxim iButton Products
Application Note 122: Using Dallas' 1-Wire ICs in 1-Cell Li-Ion Battery Packs with Low-Side N-Channel Safety FETs Master
Application Note 126: 1-Wire Communication Through Software
Application Note 162: Interfacing the DS18x20/DS1822 1-Wire Temperature Sensor in a Microcontroller Environment
Application Note 208: Curve Fitting the Error of a Bandgap-Based Digital Temperature Sensor
Application Note 2420: 1-Wire Communication with a Microchip PICmicro Microcontroller
Application Note 3754: Single-Wire Serial Bus Carries Isolated Power and Data

Sample 1-Wire subroutines that can be used in conjunction with *Application Note 74: Reading and Writing iButtons via Serial Interfaces* can be downloaded from the Maxim website.

DS18B20 OPERATION EXAMPLE 1

In this example there are multiple DS18B20s on the bus and they are using parasite power. The bus master initiates a temperature conversion in a specific DS18B20 and then reads its scratchpad and recalculates the CRC to verify the data.

MASTER MODE	DATA (LSB FIRST)	COMMENTS
Tx	Reset	Master issues reset pulse.
Rx	Presence	DS18B20s respond with presence pulse.
Tx	55h	Master issues Match ROM command.
Tx	64-bit ROM code	Master sends DS18B20 ROM code.
Tx	44h	Master issues Convert T command.
Tx	DQ line held high by strong pullup	Master applies strong pullup to DQ for the duration of the conversion (t_{CONV}).
Tx	Reset	Master issues reset pulse.
Rx	Presence	DS18B20s respond with presence pulse.
Tx	55h	Master issues Match ROM command.
Tx	64-bit ROM code	Master sends DS18B20 ROM code.
Tx	BEh	Master issues Read Scratchpad command.
Rx	9 data bytes	Master reads entire scratchpad including CRC. The master then recalculates the CRC of the first eight data bytes from the scratchpad and compares the calculated CRC with the read CRC (byte 9). If they match, the master continues; if not, the read operation is repeated.

DS18B20 OPERATION EXAMPLE 2

In this example there is only one DS18B20 on the bus and it is using parasite power. The master writes to the T_H, T_L, and configuration registers in the DS18B20 scratchpad and then reads the scratchpad and recalculates the CRC to verify the data. The master then copies the scratchpad contents to EEPROM.

MASTER MODE	DATA (LSB FIRST)	COMMENTS
Tx	Reset	Master issues reset pulse.
Rx	Presence	DS18B20 responds with presence pulse.
Tx	CCh	Master issues Skip ROM command.
Tx	4Eh	Master issues Write Scratchpad command.
Tx	3 data bytes	Master sends three data bytes to scratchpad (T_H, T_L, and config).
Tx	Reset	Master issues reset pulse.
Rx	Presence	DS18B20 responds with presence pulse.
Tx	CCh	Master issues Skip ROM command.
Tx	BEh	Master issues Read Scratchpad command.
Rx	9 data bytes	Master reads entire scratchpad including CRC. The master then recalculates the CRC of the first eight data bytes from the scratchpad and compares the calculated CRC with the read CRC (byte 9). If they match, the master continues; if not, the read operation is repeated.
Tx	Reset	Master issues reset pulse.
Rx	Presence	DS18B20 responds with presence pulse.
Tx	CCh	Master issues Skip ROM command.
Tx	48h	Master issues Copy Scratchpad command.
Tx	DQ line held high by strong pullup	Master applies strong pullup to DQ for at least 10ms while copy operation is in progress.

ABSOLUTE MAXIMUM RATINGS

Voltage Range on Any Pin Relative to Ground .....-0.5V to +6.0V
Operating Temperature Range .....-55°C to +125°C
Storage Temperature Range.....-55°C to +125°C
Solder Temperature .....Refer to the IPC/JEDEC J-STD-020 Specification.

These are stress ratings only and functional operation of the device at these or any other conditions above those indicated in the operation sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods of time may affect reliability.

DC ELECTRICAL CHARACTERISTICS (-55°C to +125°C; V_{DD}=3.0V to 5.5V)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS	NOTES
Supply Voltage	V_{DD}	Local Power	+3.0		+5.5	V	1
Pullup Supply Voltage	V_{PU}	Parasite Power	+3.0		+5.5	V	1,2
		Local Power	+3.0		V_{DD}		
Thermometer Error	t_{ERR}	-10°C to +85°C			±0.5	°C	3
		-55°C to +125°C			±2		
Input Logic-Low	V_{IL}		-0.3		+0.8	V	1,4,5
Input Logic-High	V_{IH}	Local Power	+2.2		The lower of 5.5 or V_{DD} + 0.3	V	1, 6
		Parasite Power	+3.0				
Sink Current	I_L	$V_{I/O}$ = 0.4V	4.0			mA	1
Standby Current	I_{DDS}			750	1000	nA	7,8
Active Current	I_{DD}	V_{DD} = 5V		1	1.5	mA	9
DQ Input Current	I_{DQ}			5		µA	10
Drift				±0.2		°C	11

NOTES:

1) All voltages are referenced to ground.
2) The Pullup Supply Voltage specification assumes that the pullup device is ideal, and therefore the high level of the pullup is equal to V_{PU}. In order to meet the V_{IH} spec of the DS18B20, the actual supply rail for the strong pullup transistor must include margin for the voltage drop across the transistor when it is turned on; thus: $V_{PU\_ACTUAL} = V_{PU\_IDEAL} + V_{TRANSISTOR}$.
3) See typical performance curve in Figure 17.
4) Logic-low voltages are specified at a sink current of 4mA.
5) To guarantee a presence pulse under low voltage parasite power conditions, V_{ILMAX} may have to be reduced to as low as 0.5V.
6) Logic-high voltages are specified at a source current of 1mA.
7) Standby current specified up to +70°C. Standby current typically is 3µA at +125°C.
8) To minimize I_{DDS}, DQ should be within the following ranges: GND ≤ DQ ≤ GND + 0.3V or V_{DD} – 0.3V ≤ DQ ≤ V_{DD}.
9) Active current refers to supply current during active temperature conversions or EEPROM writes.
10) DQ line is high (“high-Z” state).
11) Drift data is based on a 1000-hour stress test at +125°C with V_{DD} = 5.5V.

AC ELECTRICAL CHARACTERISTICS—NV MEMORY

(-55°C to +100°C; V_{DD} = 3.0V to 5.5V)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
NV Write Cycle Time	t_{WR}			2	10	ms
EEPROM Writes	N_{EEWR}	-55°C to +55°C	50k			writes
EEPROM Data Retention	t_{EEDR}	-55°C to +55°C	10			years

AC ELECTRICAL CHARACTERISTICS (-55°C to +125°C; V_{DD} = 3.0V to 5.5V)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS	NOTES
Temperature Conversion Time	t_{CONV}	9-bit resolution			93.75	ms	1
		10-bit resolution			187.5		
		11-bit resolution			375		
		12-bit resolution			750		
Time to Strong Pullup On	t_{SPON}	Start Convert T Command Issued			10	µs	
Time Slot	t_{SLOT}		60		120	µs	1
Recovery Time	t_{REC}		1			µs	1
Write 0 Low Time	t_{LOW0}		60		120	µs	1
Write 1 Low Time	t_{LOW1}		1		15	µs	1
Read Data Valid	t_{RDV}				15	µs	1
Reset Time High	t_{RSTH}		480			µs	1
Reset Time Low	t_{RSTL}		480			µs	1,2
Presence-Detect High	t_{PDHIGH}		15		60	µs	1
Presence-Detect Low	t_{PDLOW}		60		240	µs	1
Capacitance	$C_{IN/OUT}$				25	pF	

NOTES:

1) See the timing diagrams in Figure 18.
2) Under parasite power, if t_{RSTL} > 960µs, a power-on reset may occur.

Figure 17. Typical Performance Curve

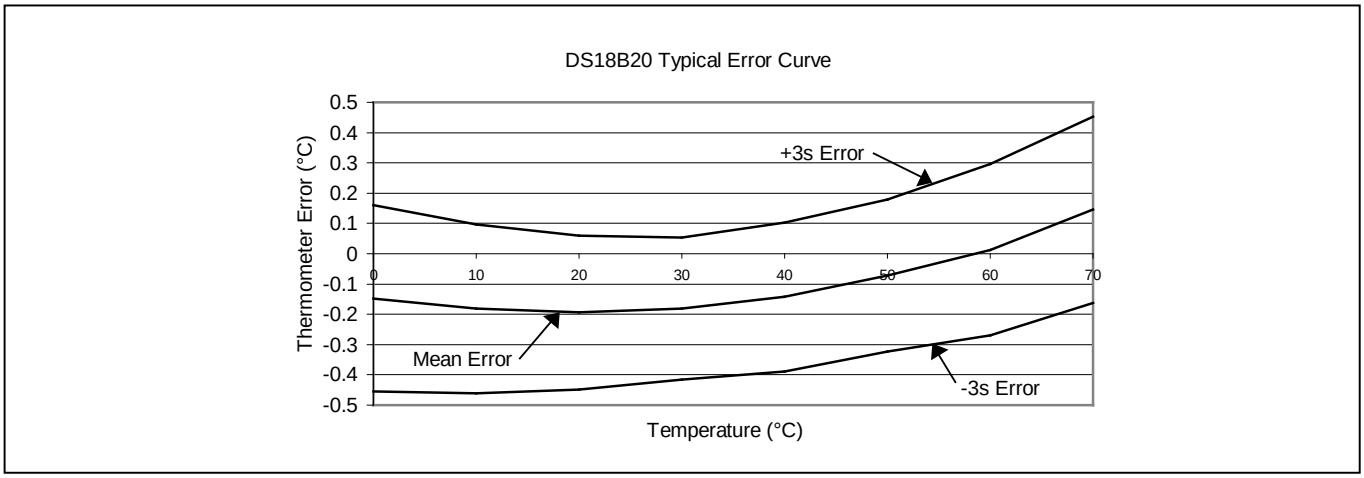
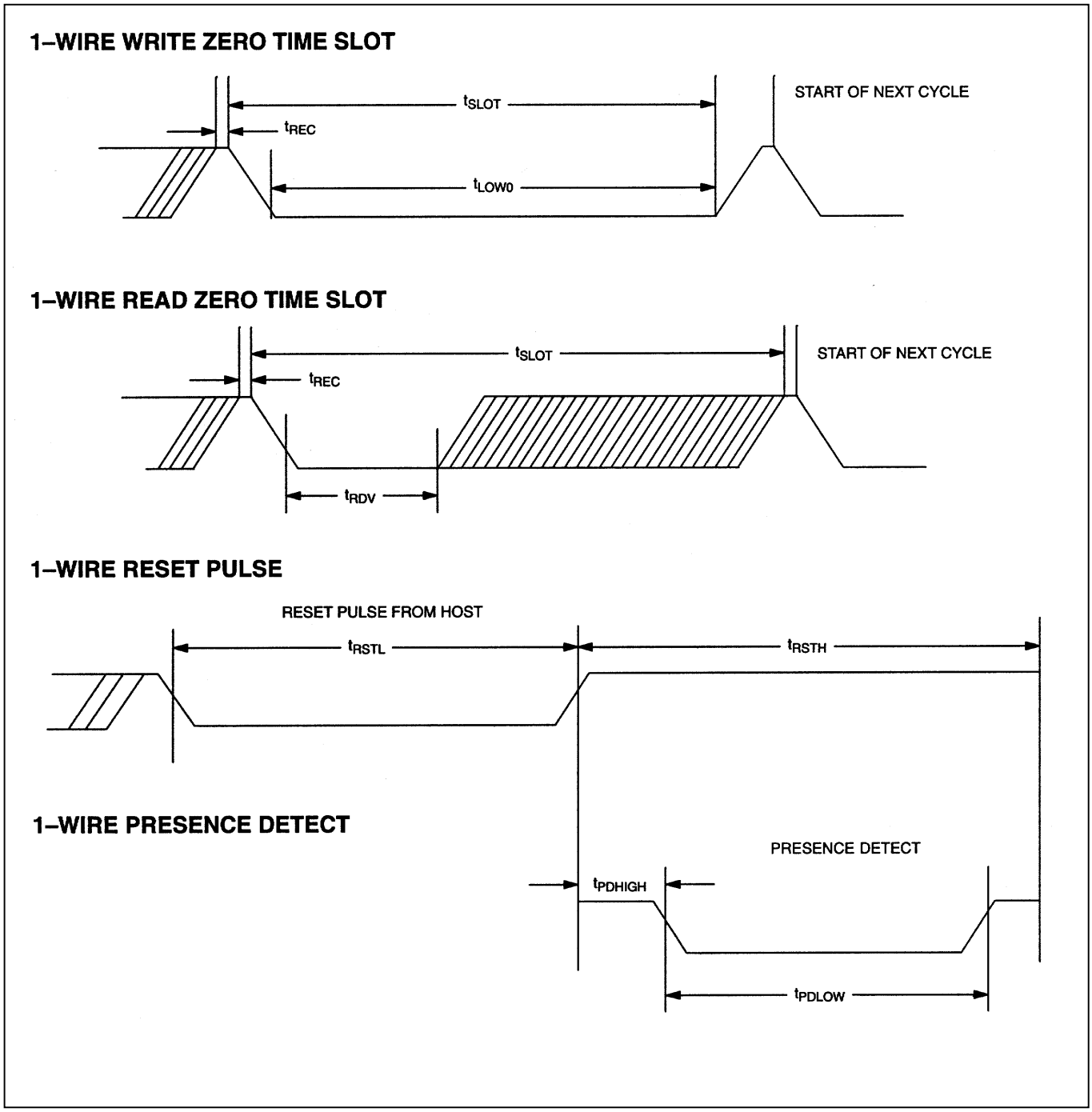


Figure 18. Timing Diagrams

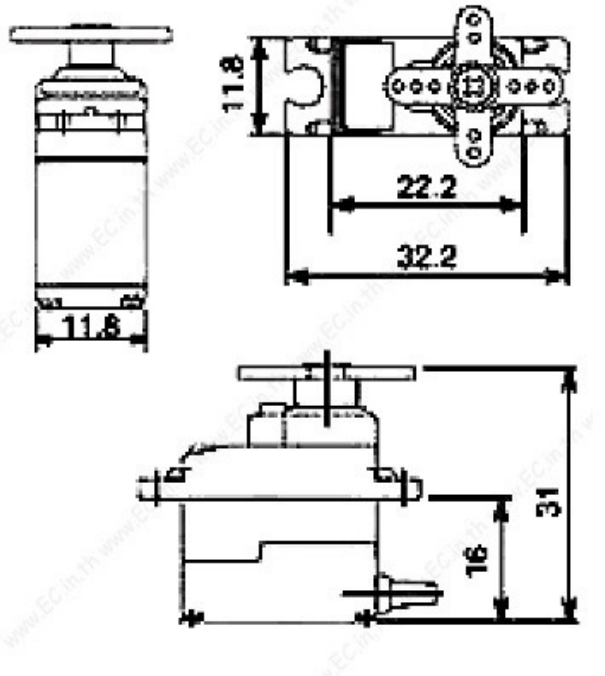


REVISION HISTORY

REVISION DATE	DESCRIPTION	PAGES CHANGED
030107	In the *Absolute Maximum Ratings* section, removed the reflow oven temperature value of +220°C. Reference to JEDEC specification for reflow remains.	19
101207	In the *Operation—Alarm Signaling* section, added “or equal to” in the desciption for a TH alarm condition	5
	In the *Memory* section, removed incorrect text describing memory.	7
	In the *Configuration Register* section, removed incorrect text describing configuration register.	8
042208	In the *Ordering Information* table, added TO-92 straight-lead packages and included a note that the TO-92 package in tape and reel can be ordered with either formed or straight leads.	2

Maxim Integrated Products, 120 San Gabriel Drive, Sunnyvale, CA 94086 408-737-7600

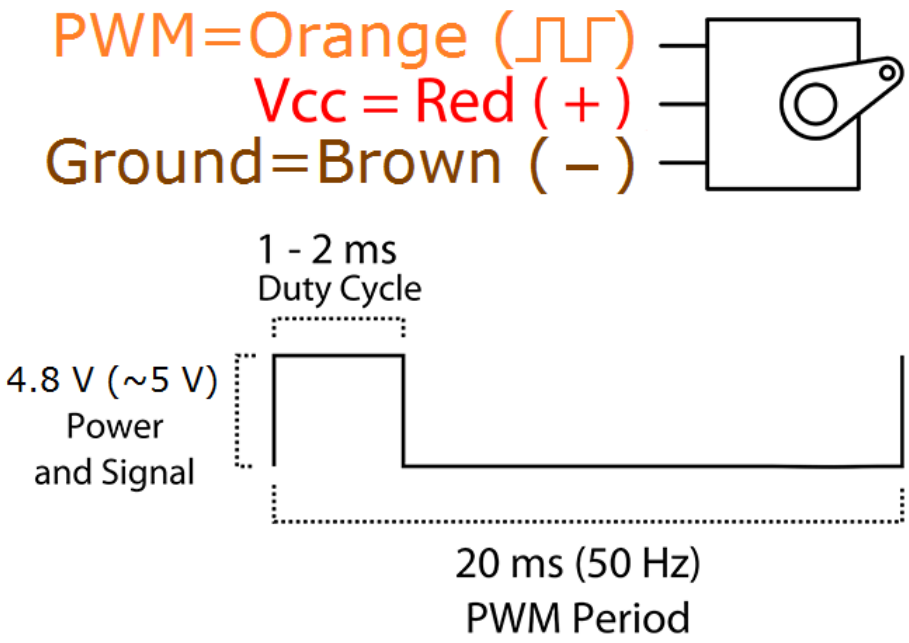
SG90 9 g Micro Servo



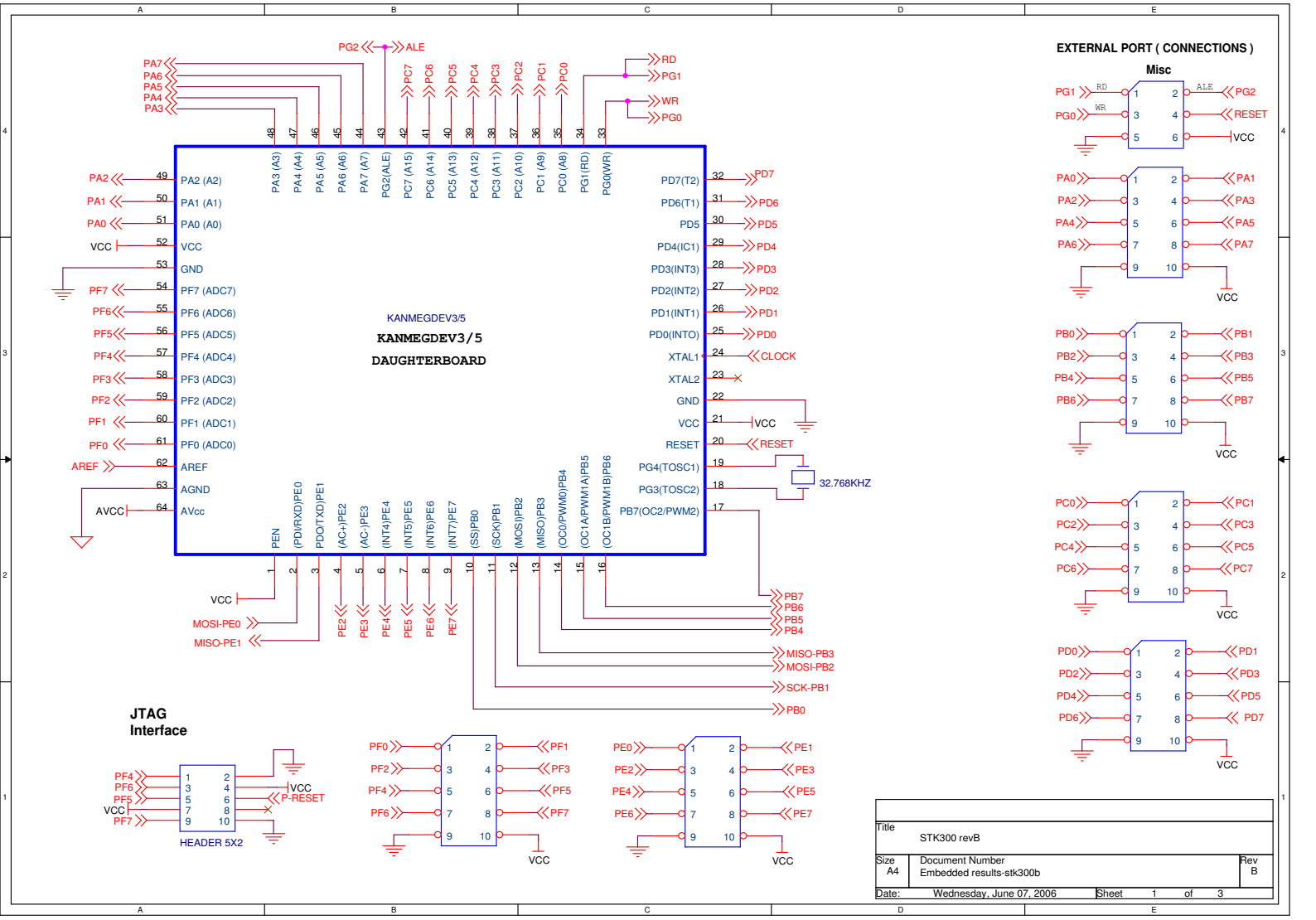
Tiny and lightweight with high output power. Servo can rotate approximately 180 degrees (90 in each direction), and works just like the standard kinds but *smaller*. You can use any servo code, hardware or library to control these servos. Good for beginners who want to make stuff move without building a motor controller with feedback & gear box, especially since it will fit in small places. It comes with a 3 horns (arms) and hardware.

Specifications

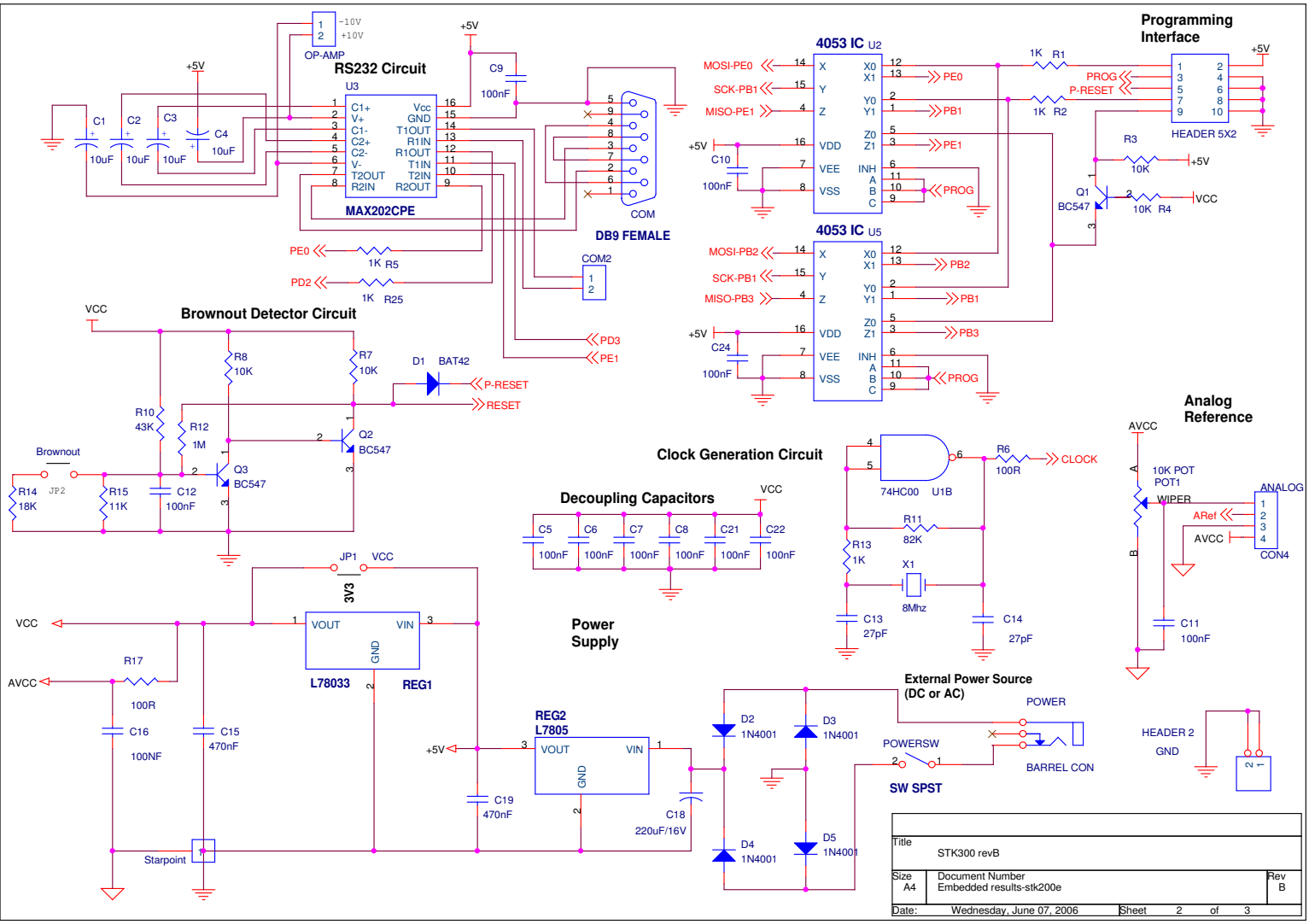
- Weight: 9 g
- Dimension: 22.2 x 11.8 x 31 mm approx.
- Stall torque: 1.8 kgf·cm
- Operating speed: 0.1 s/60 degree
- Operating voltage: 4.8 V (~5V)
- Dead band width: 10 µs
- Temperature range: 0 ºC – 55 ºC

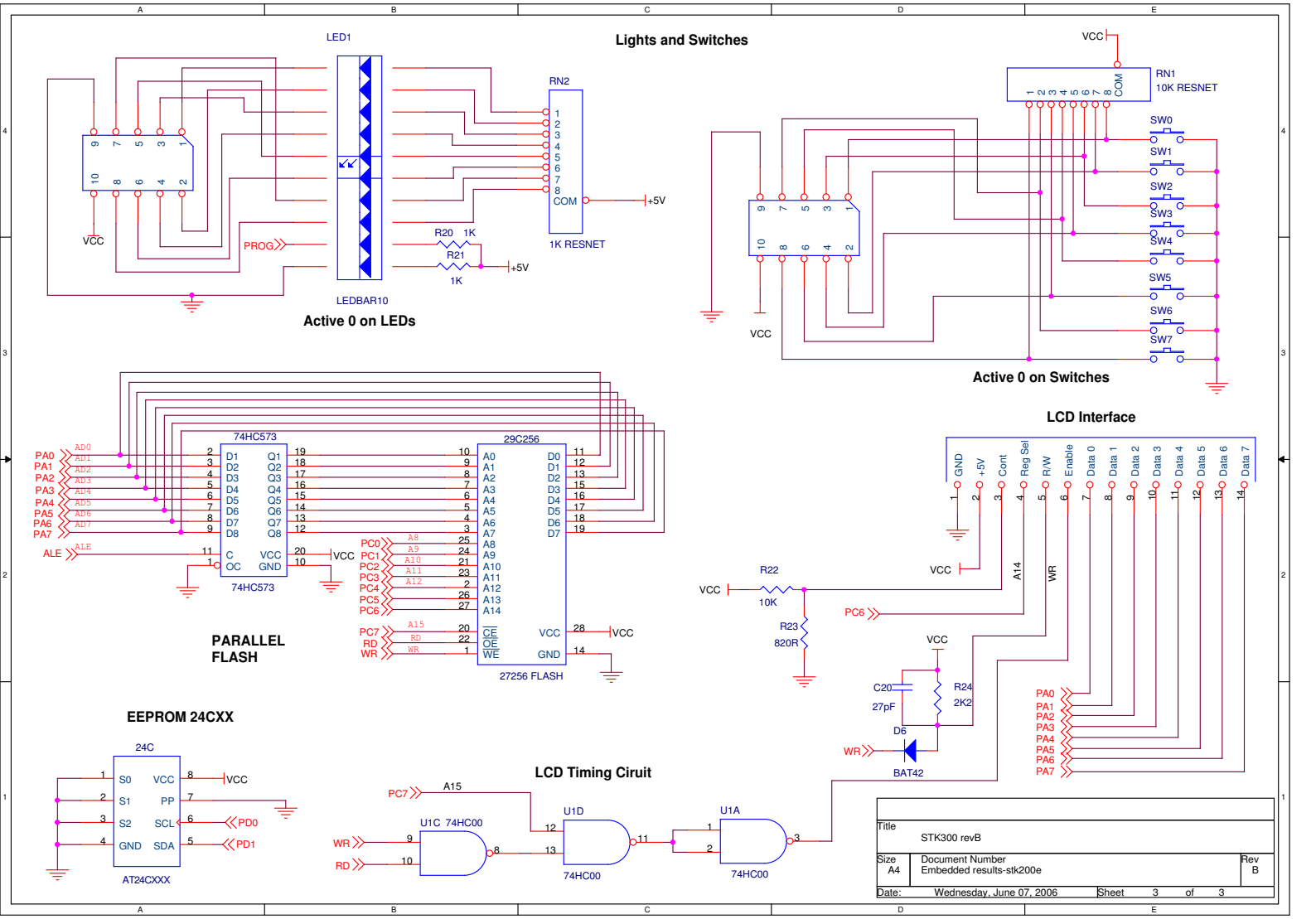


Position "0" (1.5 ms pulse) is middle, "90" (~2 ms pulse) is all the way to the right, "-90" (~1 ms pulse) is all the way to the left.



Title	STK300 revB	
Size A4	Document Number Embedded results-stk300b	Rev B
Date: Wednesday, June 07, 2006	Sheet 1 of 3	


RS232 Circuit
MAX202CPE
OP-AMP
-10V
+10V
+5V
C1
C2
C3
C4
10uF
C9
100nF
U3
C1+
V+
C1-
C2+
C2-
V-
T2OUT
R2IN
Vcc
GND
T1OUT
R1IN
R1OUT
T1IN
T2IN
R2OUT
DB9 FEMALE
COM
COM2
PE0
PD2
1K R5
1K R25
PD3
PE1
4053 IC U2
4053 IC U5
MOSI-PE0
SCK-PB1
MISO-PE1
MOSI-PB2
MISO-PB3
X
Y
Z
X0
X1
Y0
Y1
Z0
Z1
VDD
VEE
VSS
INH
A
B
C
C10
C24
100nF
PE1
PB1
PB2
PB3
PROG
1K R1
1K R2
Programming Interface
PROG
P-RESET
HEADER 5X2
R3
10K
Q1
BC547
10K R4
VCC
Brownout Detector Circuit
R8
10K
R7
10K
D1 BAT42
P-RESET
RESET
R10
43K
R12
1M
Q2
BC547
Q3
BC547
Brownout
JP2
R14
18K
R15
11K
C12
100nF
Clock Generation Circuit
74HC00 U1B
R6
100R
CLOCK
R11
82K
R13
1K
X1
8Mhz
C13
27pF
C14
27pF
Decoupling Capacitors
C5
C6
C7
C8
C21
C22
100nF
Analog Reference
AVCC
10K POT
POT1
WIPER
ANALOG
ARef
CON4
C11
100nF
Power Supply
JP1
3V3
L78033
REG1
VOUT
VIN
GND
R17
100R
C16
100NF
C15
470nF
Starpoint
REG2
L7805
C19
470nF
C18
220uF/16V
D2
1N4001
D3
1N4001
D4
1N4001
D5
1N4001
External Power Source
(DC or AC)
POWER
POWERSW
SW SPST
BARREL CON
HEADER 2
GND
Title
STK300 revB
Size
A4
Document Number
Embedded results-stk200e
Rev
B
Date:
Wednesday, June 07, 2006
Sheet
2
of
3


Lights and Switches
LED1
LEDBAR10
RN2
1K RESNET
R20 1K
R21
1K
+5V
VCC
PROG
Active 0 on LEDs
RN1
10K RESNET
SW0
SW1
SW2
SW3
SW4
SW5
SW6
SW7
Active 0 on Switches
74HC573
29C256
27256 FLASH
PARALLEL
FLASH
LCD Interface
GND
+5V
Cont
Reg Sel
R/W
Enable
Data 0
Data 1
Data 2
Data 3
Data 4
Data 5
Data 6
Data 7
R22
10K
R23
820R
C20
27pF
R24
2K2
D6
BAT42
EEPROM 24CXX
24C
AT24CXXX
LCD Timing Ciruit
U1C 74HC00
U1D
U1A
74HC00
Title
STK300 revB
Size
A4
Document Number
Embedded results-stk200e
Rev
B
Date:
Wednesday, June 07, 2006
Sheet
3
of
3

HD44780U (LCD-II)

(Dot Matrix Liquid Crystal Display Controller/Driver)

HITACHI

ADE-207-272(Z)
'99.9
Rev. 0.0

Description

The HD44780U dot-matrix liquid crystal display controller and driver LSI displays alphanumerics, Japanese kana characters, and symbols. It can be configured to drive a dot-matrix liquid crystal display under the control of a 4- or 8-bit microprocessor. Since all the functions such as display RAM, character generator, and liquid crystal driver, required for driving a dot-matrix liquid crystal display are internally provided on one chip, a minimal system can be interfaced with this controller/driver.

A single HD44780U can display up to one 8-character line or two 8-character lines.

The HD44780U has pin function compatibility with the HD44780S which allows the user to easily replace an LCD-II with an HD44780U. The HD44780U character generator ROM is extended to generate 208 5 × 8 dot character fonts and 32 5 × 10 dot character fonts for a total of 240 different character fonts.

The low power supply (2.7V to 5.5V) of the HD44780U is suitable for any portable battery-driven product requiring low power dissipation.

Features

- 5 × 8 and 5 × 10 dot matrix possible
- Low power operation support:
  - 2.7 to 5.5V
- Wide range of liquid crystal display driver power
  - 3.0 to 11V
- Liquid crystal drive waveform
  - A (One line frequency AC waveform)
- Correspond to high speed MPU bus interface
  - 2 MHz (when V_{CC} = 5V)
- 4-bit or 8-bit MPU interface enabled
- 80 × 8-bit display RAM (80 characters max.)
- 9,920-bit character generator ROM for a total of 240 character fonts
  - 208 character fonts (5 × 8 dot)
  - 32 character fonts (5 × 10 dot)

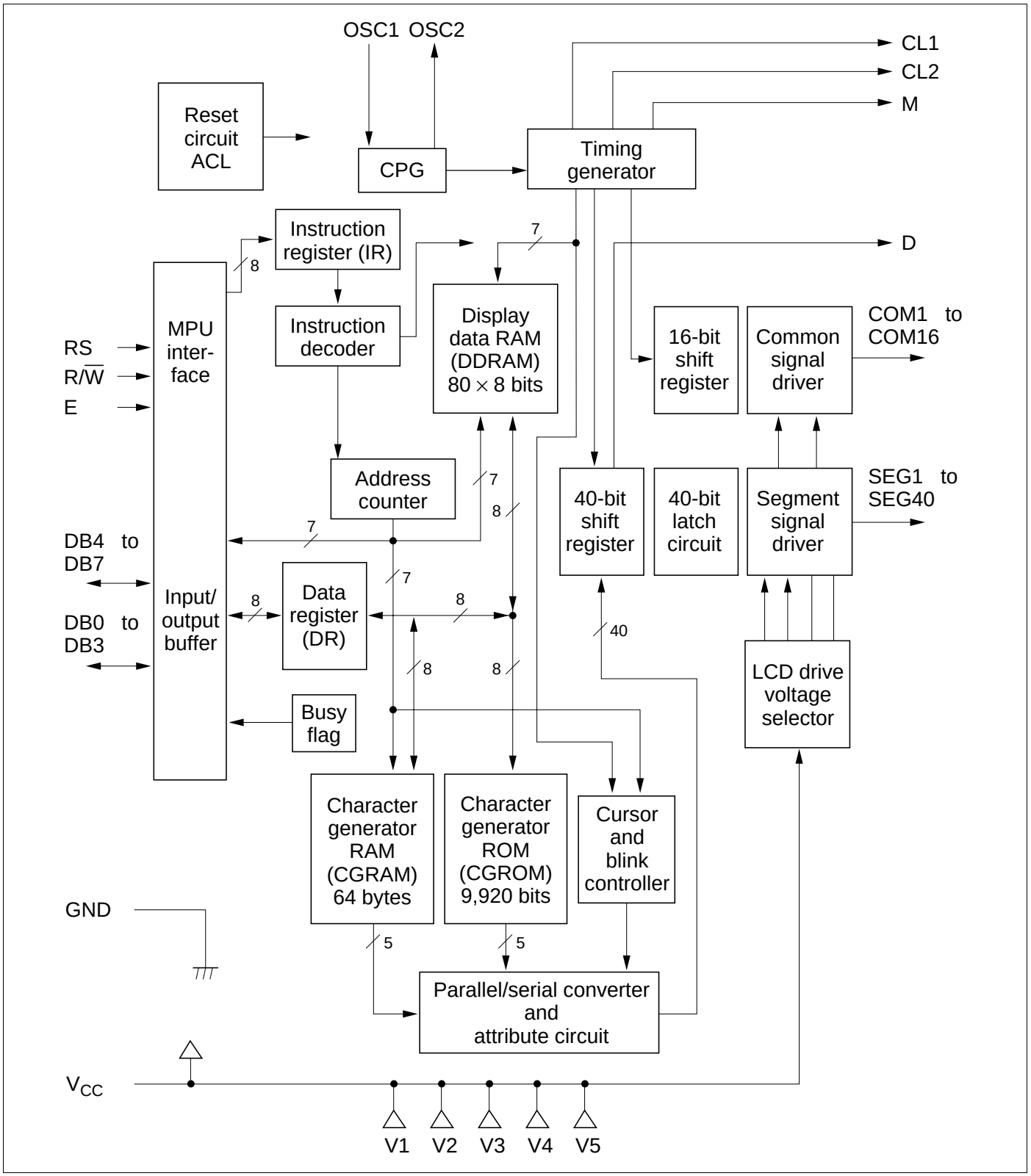
- 64 × 8-bit character generator RAM
  - 8 character fonts (5 × 8 dot)
  - 4 character fonts (5 × 10 dot)
- 16-common × 40-segment liquid crystal display driver
- Programmable duty cycles
  - 1/8 for one line of 5 × 8 dots with cursor
  - 1/11 for one line of 5 × 10 dots with cursor
  - 1/16 for two lines of 5 × 8 dots with cursor
- Wide range of instruction functions:
  - Display clear, cursor home, display on/off, cursor on/off, display character blink, cursor shift, display shift
- Pin function compatibility with HD44780S
- Automatic reset circuit that initializes the controller/driver after power on
- Internal oscillator with external resistors
- Low power consumption

Ordering Information

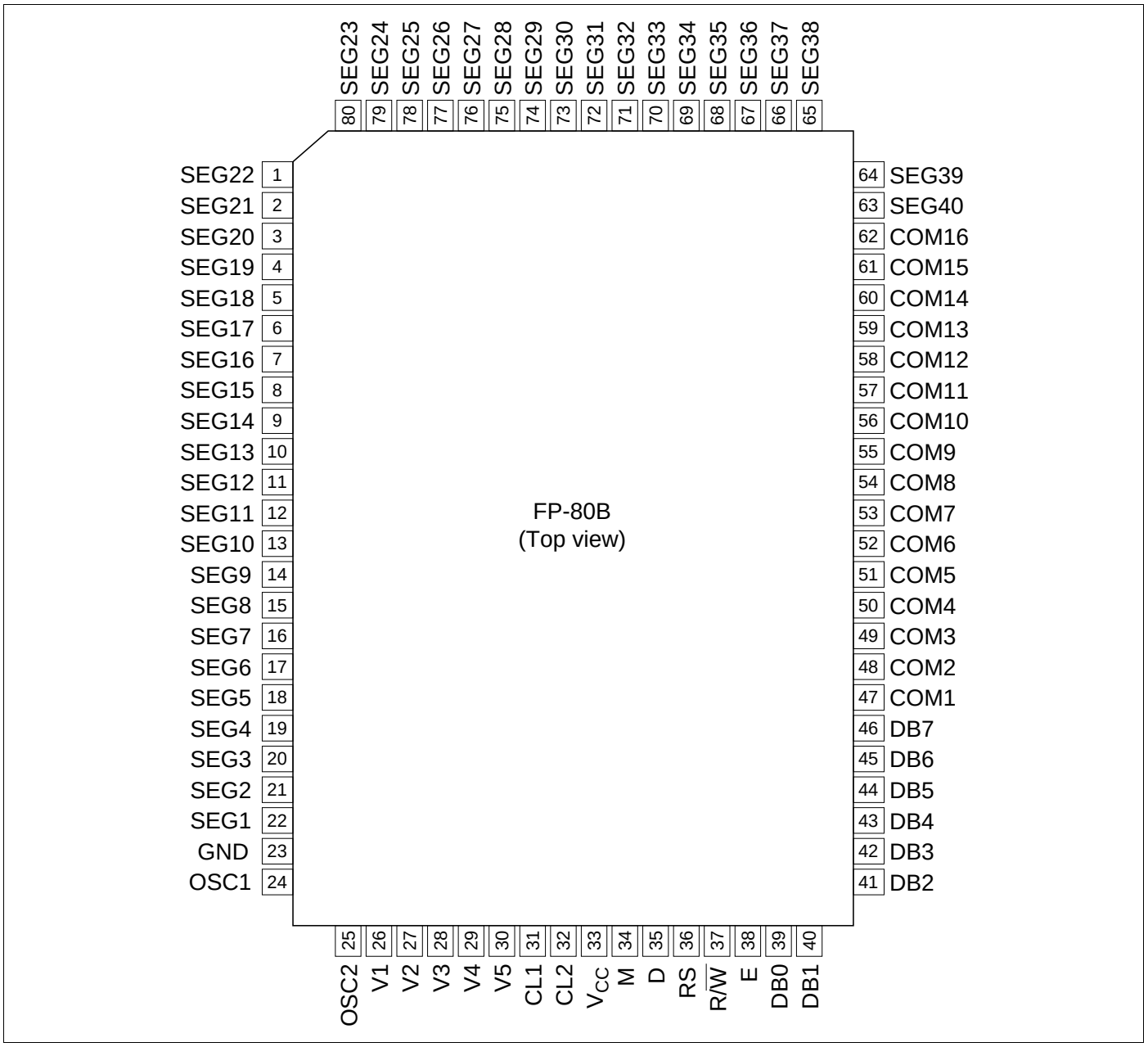
Type No.	Package	CGROM
HD44780UA00FS	FP-80B	Japanese standard font
HCD44780UA00	Chip	
HD44780UA00TF	TFP-80F	
HD44780UA02FS	FP-80B	European standard font
HCD44780UA02	Chip	
HD44780UA02TF	TFP-80F	
HD44780UBxxFS	FP-80B	Custom font
HCD44780UBxx	Chip	
HD44780UBxxTF	TFP-80F	

Note: xx: ROM code No.

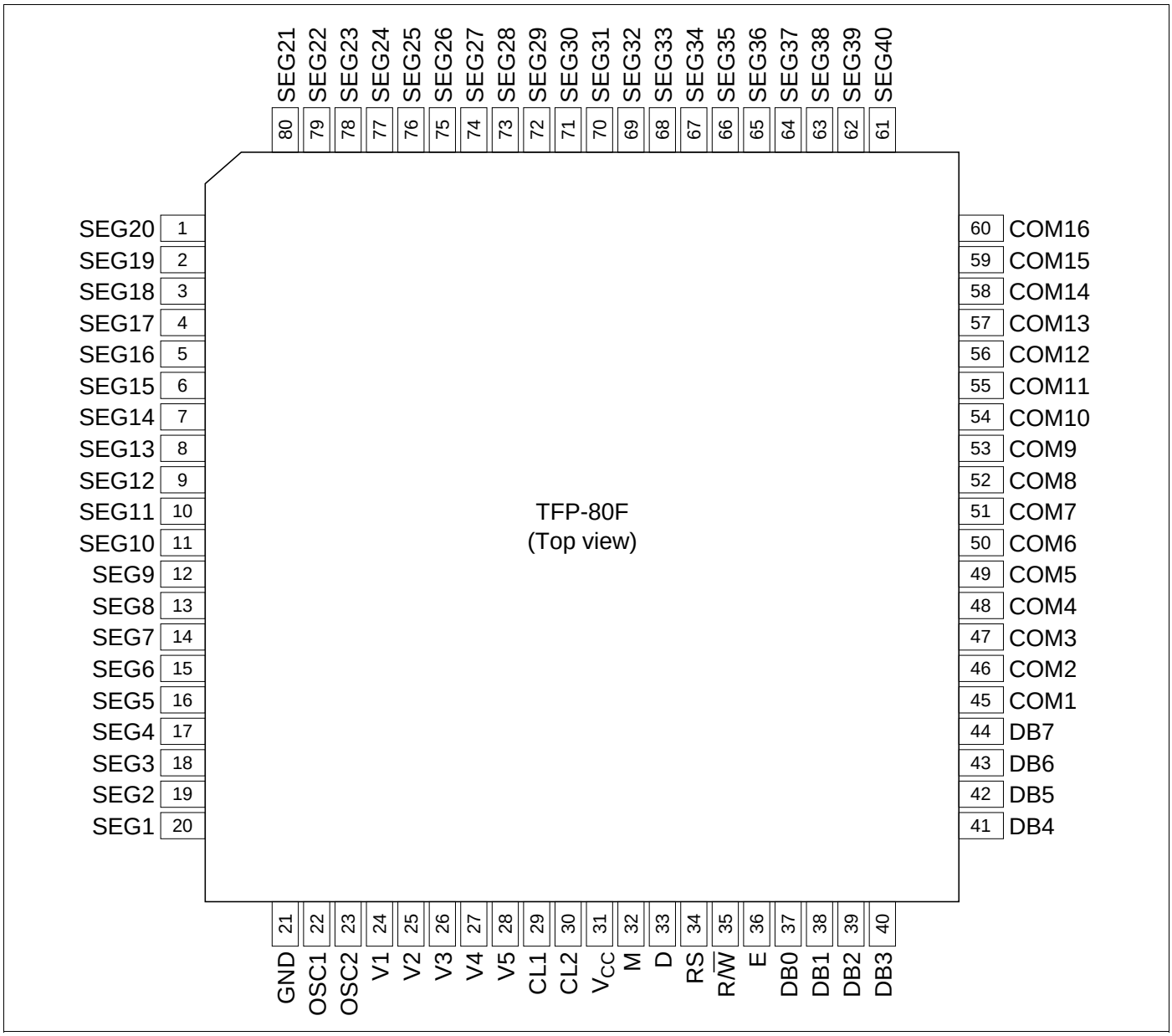
HD44780U Block Diagram



HD44780U Pin Arrangement (FP-80B)

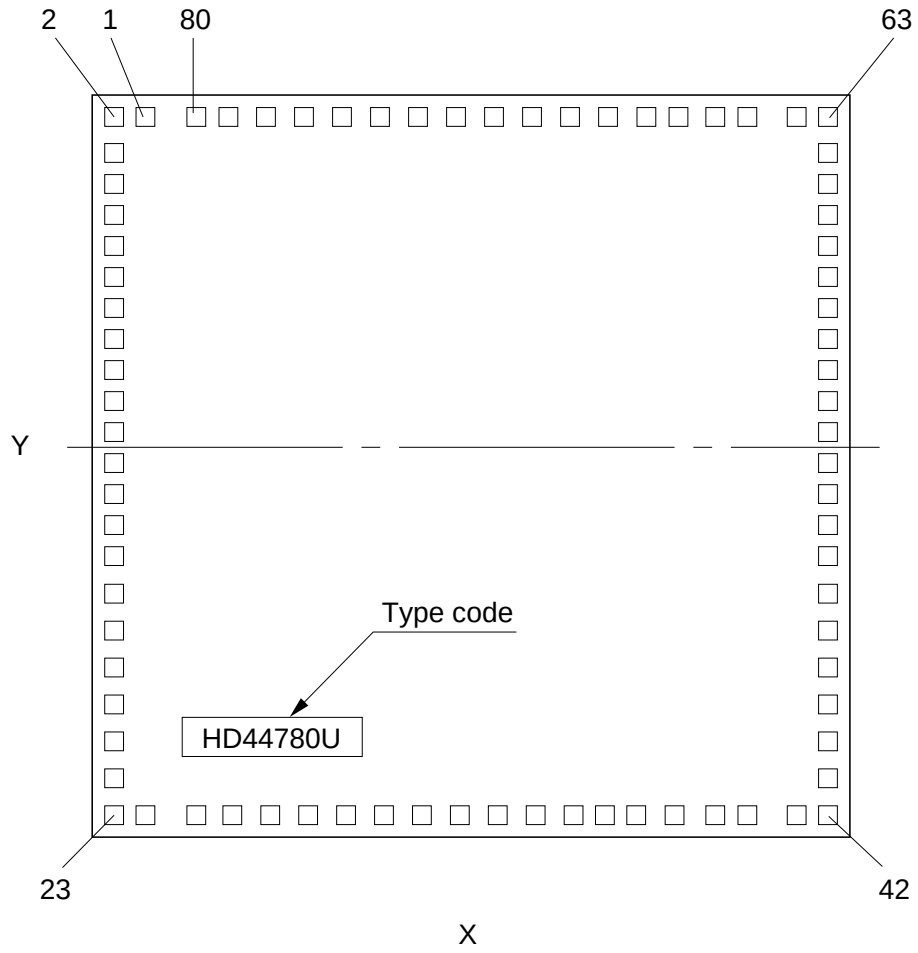


HD44780U Pin Arrangement (TFP-80F)



HD44780U Pad Arrangement

Chip size: 4.90×4.90 mm^2
Coordinate: Pad center (μm)
Origin: Chip center
Pad size: 114×114 μm^2



HCD44780U Pad Location Coordinates

		Coordinate					Coordinate	
Pad No.	Function	X (um)	Y (um)		Pad No.	Function	X (um)	Y (um)
1	SEG22	–2100	2313		41	DB2	2070	–2290
2	SEG21	–2280	2313		42	DB3	2260	–2290
3	SEG20	–2313	2089		43	DB4	2290	–2099
4	SEG19	–2313	1833		44	DB5	2290	–1883
5	SEG18	–2313	1617		45	DB6	2290	–1667
6	SEG17	–2313	1401		46	DB7	2290	–1452
7	SEG16	–2313	1186		47	COM1	2313	–1186
8	SEG15	–2313	970		48	COM2	2313	–970
9	SEG14	–2313	755		49	COM3	2313	–755
10	SEG13	–2313	539		50	COM4	2313	–539
11	SEG12	–2313	323		51	COM5	2313	–323
12	SEG11	–2313	108		52	COM6	2313	–108
13	SEG10	–2313	–108		53	COM7	2313	108
14	SEG9	–2313	–323		54	COM8	2313	323
15	SEG8	–2313	–539		55	COM9	2313	539
16	SEG7	–2313	–755		56	COM10	2313	755
17	SEG6	–2313	–970		57	COM11	2313	970
18	SEG5	–2313	–1186		58	COM12	2313	1186
19	SEG4	–2313	–1401		59	COM13	2313	1401
20	SEG3	–2313	–1617		60	COM14	2313	1617
21	SEG2	–2313	–1833		61	COM15	2313	1833
22	SEG1	–2313	–2073		62	COM16	2313	2095
23	GND	–2280	–2290		63	SEG40	2296	2313
24	OSC1	–2080	–2290		64	SEG39	2100	2313
25	OSC2	–1749	–2290		65	SEG38	1617	2313
26	V1	–1550	–2290		66	SEG37	1401	2313
27	V2	–1268	–2290		67	SEG36	1186	2313
28	V3	–941	–2290		68	SEG35	970	2313
29	V4	–623	–2290		69	SEG34	755	2313
30	V5	–304	–2290		70	SEG33	539	2313
31	CL1	–48	–2290		71	SEG32	323	2313
32	CL2	142	–2290		72	SEG31	108	2313
33	V_{CC}	309	–2290		73	SEG30	–108	2313
34	M	475	–2290		74	SEG29	–323	2313
35	D	665	–2290		75	SEG28	–539	2313
36	RS	832	–2290		76	SEG27	–755	2313
37	R/$\overline{W}$	1022	–2290		77	SEG26	–970	2313
38	E	1204	–2290		78	SEG25	–1186	2313
39	DB0	1454	–2290		79	SEG24	–1401	2313
40	DB1	1684	–2290		80	SEG23	–1617	2313

Pin Functions

Signal	No. of Lines	I/O	Device Interfaced with	Function
RS	1	I	MPU	Selects registers. 0: Instruction register (for write) Busy flag: address counter (for read) 1: Data register (for write and read)
R/$\overline{W}$	1	I	MPU	Selects read or write. 0: Write 1: Read
E	1	I	MPU	Starts data read/write.
DB4 to DB7	4	I/O	MPU	Four high order bidirectional tristate data bus pins. Used for data transfer and receive between the MPU and the HD44780U. DB7 can be used as a busy flag.
DB0 to DB3	4	I/O	MPU	Four low order bidirectional tristate data bus pins. Used for data transfer and receive between the MPU and the HD44780U. These pins are not used during 4-bit operation.
CL1	1	O	Extension driver	Clock to latch serial data D sent to the extension driver
CL2	1	O	Extension driver	Clock to shift serial data D
M	1	O	Extension driver	Switch signal for converting the liquid crystal drive waveform to AC
D	1	O	Extension driver	Character pattern data corresponding to each segment signal
COM1 to COM16	16	O	LCD	Common signals that are not used are changed to non-selection waveforms. COM9 to COM16 are non-selection waveforms at 1/8 duty factor and COM12 to COM16 are non-selection waveforms at 1/11 duty factor.
SEG1 to SEG40	40	O	LCD	Segment signals
V1 to V5	5	—	Power supply	Power supply for LCD drive V_{CC} –V5 = 11 V (max)
V_{CC}, GND	2	—	Power supply	V_{CC}: 2.7V to 5.5V, GND: 0V
OSC1, OSC2	2	—	Oscillation resistor clock	When crystal oscillation is performed, a resistor must be connected externally. When the pin input is an external clock, it must be input to OSC1.

Function Description

Registers

The HD44780U has two 8-bit registers, an instruction register (IR) and a data register (DR).

The IR stores instruction codes, such as display clear and cursor shift, and address information for display data RAM (DDRAM) and character generator RAM (CGRAM). The IR can only be written from the MPU.

The DR temporarily stores data to be written into DDRAM or CGRAM and temporarily stores data to be read from DDRAM or CGRAM. Data written into the DR from the MPU is automatically written into DDRAM or CGRAM by an internal operation. The DR is also used for data storage when reading data from DDRAM or CGRAM. When address information is written into the IR, data is read and then stored into the DR from DDRAM or CGRAM by an internal operation. Data transfer between the MPU is then completed when the MPU reads the DR. After the read, data in DDRAM or CGRAM at the next address is sent to the DR for the next read from the MPU. By the register selector (RS) signal, these two registers can be selected (Table 1).

Busy Flag (BF)

When the busy flag is 1, the HD44780U is in the internal operation mode, and the next instruction will not be accepted. When RS = 0 and R/$\overline{W}$ = 1 (Table 1), the busy flag is output to DB7. The next instruction must be written after ensuring that the busy flag is 0.

Address Counter (AC)

The address counter (AC) assigns addresses to both DDRAM and CGRAM. When an address of an instruction is written into the IR, the address information is sent from the IR to the AC. Selection of either DDRAM or CGRAM is also determined concurrently by the instruction.

After writing into (reading from) DDRAM or CGRAM, the AC is automatically incremented by 1 (decremented by 1). The AC contents are then output to DB0 to DB6 when RS = 0 and R/$\overline{W}$ = 1 (Table 1).

Table 1 Register Selection

RS	R/$\overline{W}$	Operation
0	0	IR write as an internal operation (display clear, etc.)
0	1	Read busy flag (DB7) and address counter (DB0 to DB6)
1	0	DR write as an internal operation (DR to DDRAM or CGRAM)
1	1	DR read as an internal operation (DDRAM or CGRAM to DR)

Display Data RAM (DDRAM)

Display data RAM (DDRAM) stores display data represented in 8-bit character codes. Its extended capacity is 80 × 8 bits, or 80 characters. The area in display data RAM (DDRAM) that is not used for display can be used as general data RAM. See Figure 1 for the relationships between DDRAM addresses and positions on the liquid crystal display.

The DDRAM address (A_{DD}) is set in the address counter (AC) as hexadecimal.

- 1-line display (N = 0) (Figure 2)
  - When there are fewer than 80 display characters, the display begins at the head position. For example, if using only the HD44780, 8 characters are displayed. See Figure 3.

    When the display shift operation is performed, the DDRAM address shifts. See Figure 3.

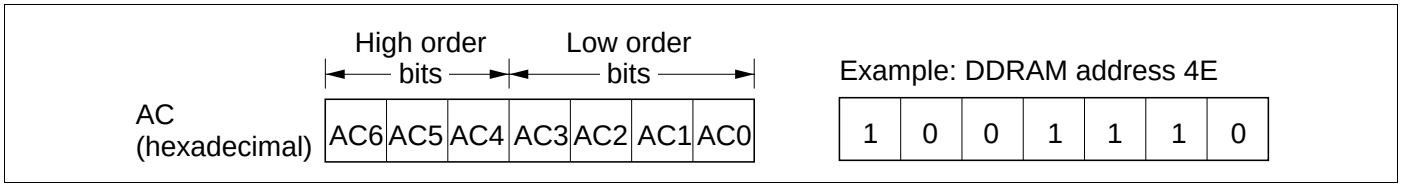


Figure 1 DDRAM Address

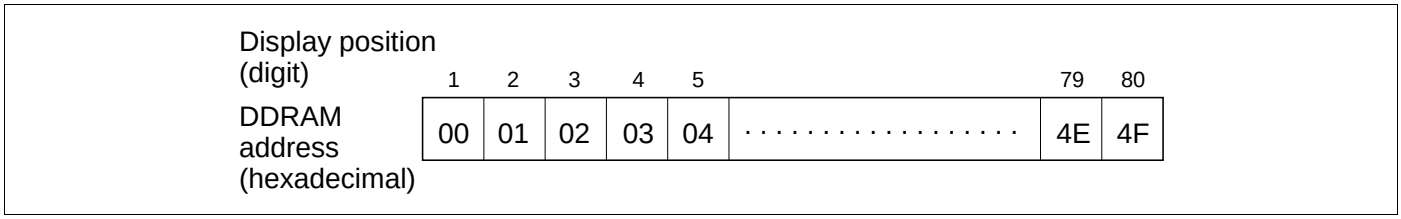


Figure 2 1-Line Display

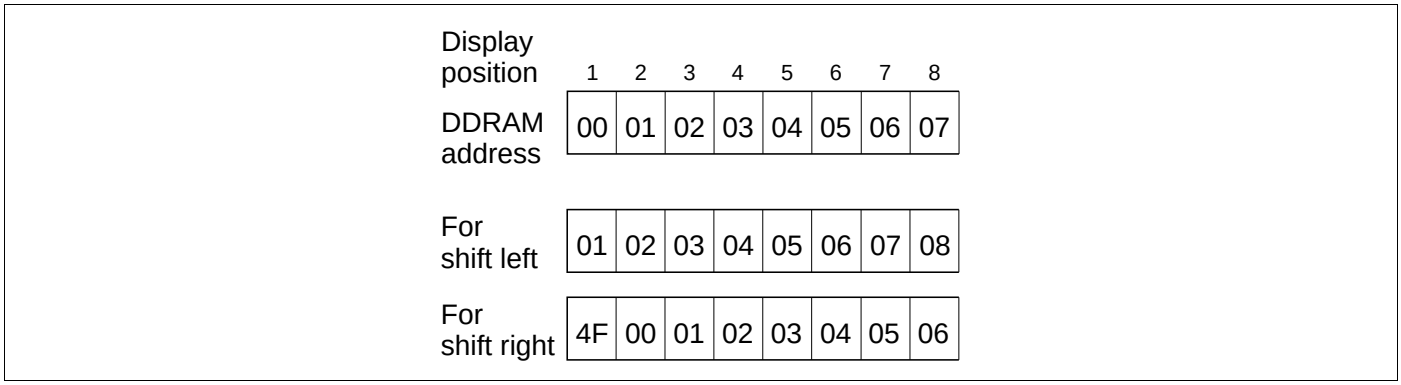


Figure 3 1-Line by 8-Character Display Example

- 2-line display (N = 1) (Figure 4)

  — Case 1: When the number of display characters is less than 40 × 2 lines, the two lines are displayed from the head. Note that the first line end address and the second line start address are not consecutive. For example, when just the HD44780 is used, 8 characters × 2 lines are displayed. See Figure 5.

  When display shift operation is performed, the DDRAM address shifts. See Figure 5.

Display position	1	2	3	4	5	· · · · · · · · · · · · · · · · · ·	39	40
DDRAM address (hexadecimal)	00	01	02	03	04	· · · · · · · · · · · · · · · · · ·	26	27
	40	41	42	43	44	· · · · · · · · · · · · · · · · · ·	66	67

Figure 4 2-Line Display

Display position	1	2	3	4	5	6	7	8
DDRAM address	00	01	02	03	04	05	06	07
	40	41	42	43	44	45	46	47
For shift left	01	02	03	04	05	06	07	08
	41	42	43	44	45	46	47	48
For shift right	27	00	01	02	03	04	05	06
	67	40	41	42	43	44	45	46

Figure 5 2-Line by 8-Character Display Example

— Case 2: For a 16-character × 2-line display, the HD44780 can be extended using one 40-output extension driver. See Figure 6.

When display shift operation is performed, the DDRAM address shifts. See Figure 6.

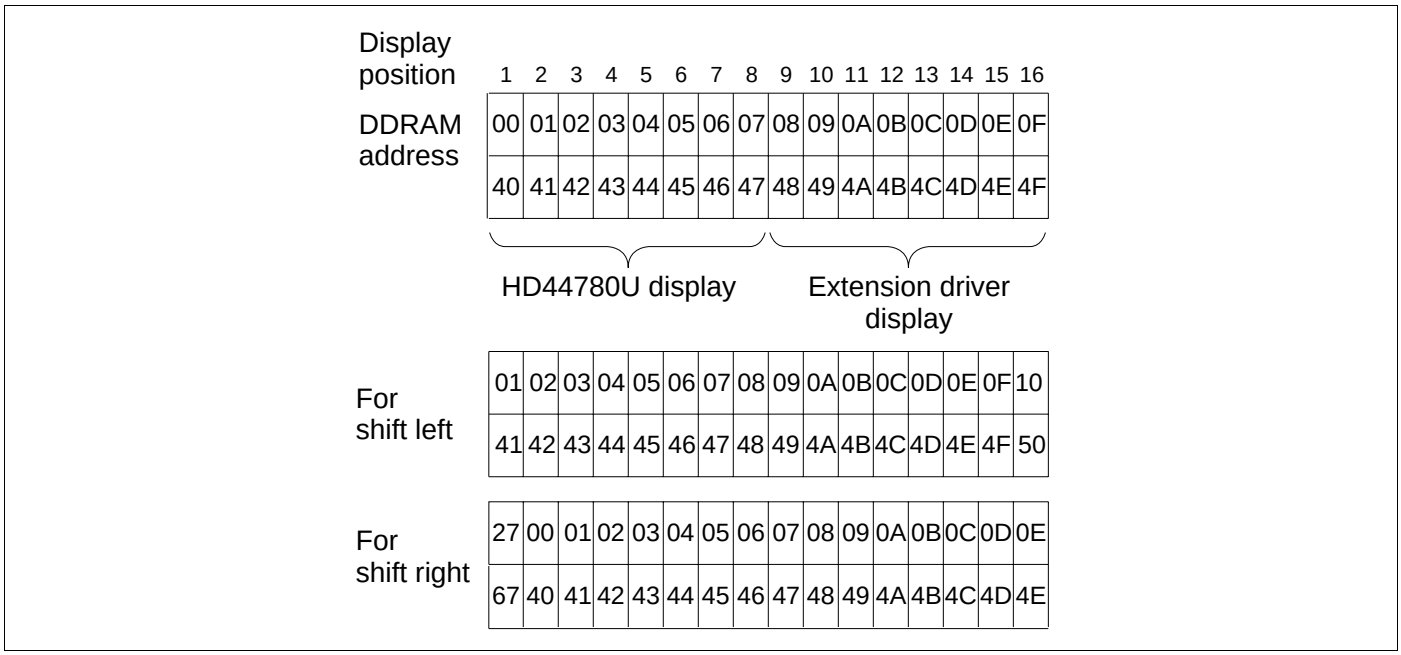


Figure 6 2-Line by 16-Character Display Example

Character Generator ROM (CGROM)

The character generator ROM generates 5 × 8 dot or 5× 10 dot character patterns from 8-bit character codes (Table 4). It can generate 208 5 × 8 dot character patterns and 32 5 × 10 dot character patterns. User-defined character patterns are also available by mask-programmed ROM.

Character Generator RAM (CGRAM)

In the character generator RAM, the user can rewrite character patterns by program. For 5 × 8 dots, eight character patterns can be written, and for 5 × 10 dots, four character patterns can be written.

Write into DDRAM the character codes at the addresses shown as the left column of Table 4 to show the character patterns stored in CGRAM.

See Table 5 for the relationship between CGRAM addresses and data and display patterns.

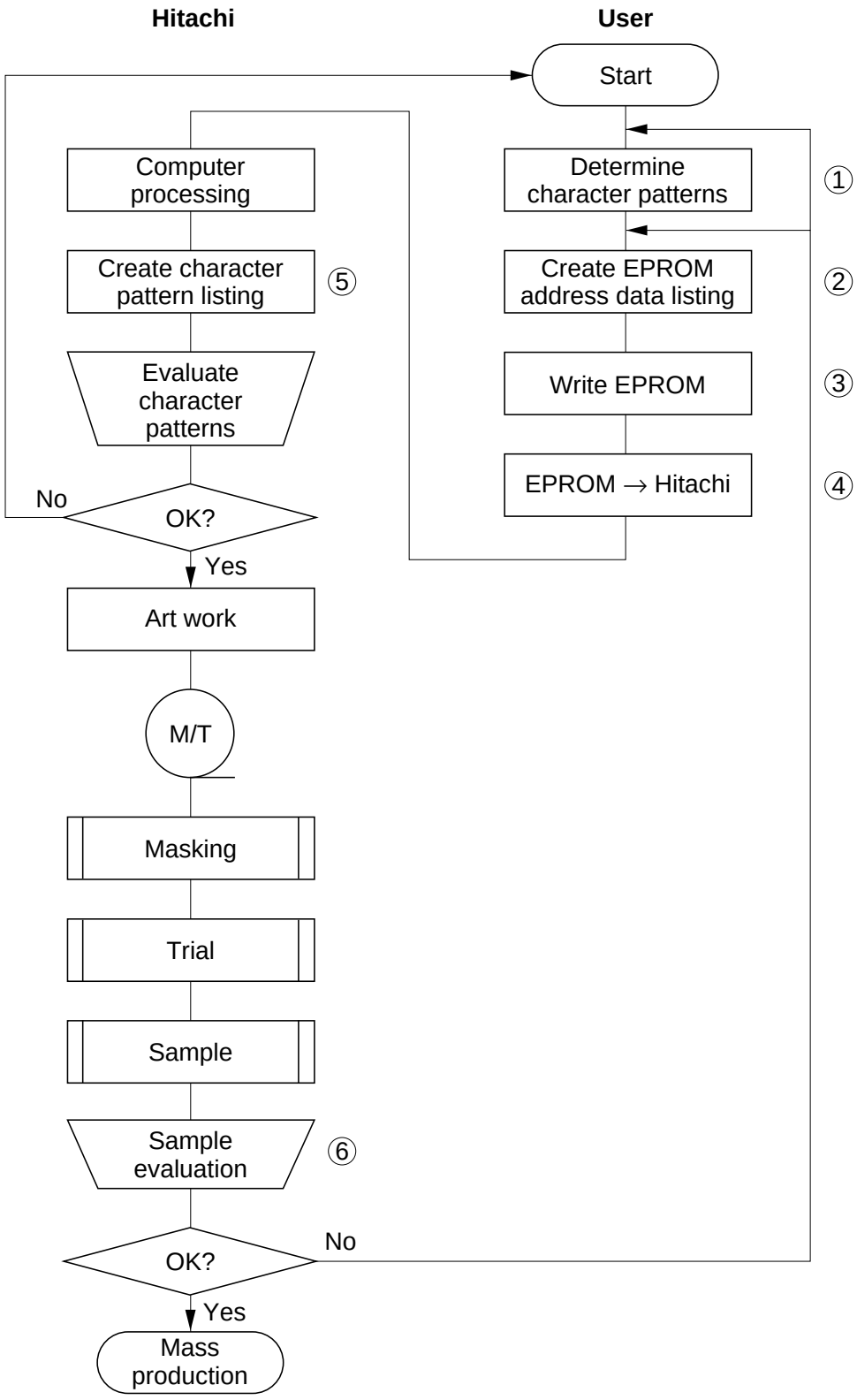
Areas that are not used for display can be used as general data RAM.

Modifying Character Patterns

- Character pattern development procedure

The following operations correspond to the numbers listed in Figure 7:

1. Determine the correspondence between character codes and character patterns.
2. Create a listing indicating the correspondence between EPROM addresses and data.
3. Program the character patterns into the EPROM.
4. Send the EPROM to Hitachi.
5. Computer processing on the EPROM is performed at Hitachi to create a character pattern listing, which is sent to the user.
6. If there are no problems within the character pattern listing, a trial LSI is created at Hitachi and samples are sent to the user for evaluation. When it is confirmed by the user that the character patterns are correctly written, mass production of the LSI proceeds at Hitachi.



Note: For a description of the numbers used in this figure, refer to the preceding page.

Figure 7 Character Pattern Development Procedure

- Programming character patterns

  This section explains the correspondence between addresses and data used to program character patterns in EPROM. The HD44780U character generator ROM can generate 208 5 × 8 dot character patterns and 32 5 × 10 dot character patterns for a total of 240 different character patterns.

  — Character patterns

  EPROM address data and character pattern data correspond with each other to form a 5 × 8 or 5 × 10 dot character pattern (Tables 2 and 3).

Table 2 Example of Correspondence between EPROM Address Data and Character Pattern (5 × 8 Dots)

EPROM Address												Data					
A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	O4	O3	O2	O1	LSB O0	
								0	0	0	0	1	0	0	0	0	
								0	0	0	1	1	0	0	0	0	
								0	0	1	0	1	0	1	1	0	
								0	0	1	1	1	1	0	0	1	
								0	1	0	0	1	0	0	0	1	
								0	1	0	1	1	0	0	0	1	
								0	1	1	0	1	1	1	1	0	
0	1	1	0	0	0	1	0	0	1	1	1	0	0	0	0	0	← Cursor position
								1	0	0	0	0	0	0	0	0	
								1	0	0	1	0	0	0	0	0	
								1	0	1	0	0	0	0	0	0	
								1	0	1	1	0	0	0	0	0	
								1	1	0	0	0	0	0	0	0	
								1	1	0	1	0	0	0	0	0	
								1	1	1	0	0	0	0	0	0	
								1	1	1	1	0	0	0	0	0	
Character code								Line position									

Notes: 1. EPROM addresses A11 to A4 correspond to a character code.
2. EPROM addresses A3 to A0 specify a line position of the character pattern.
3. EPROM data O4 to O0 correspond to character pattern data.
4. EPROM data O5 to O7 must be specified as 0.
5. A lit display position (black) corresponds to a 1.
6. Line 9 and the following lines must be blanked with 0s for a 5 × 8 dot character fonts.

— Handling unused character patterns

1. EPROM data outside the character pattern area: Always input 0s.
2. EPROM data in CGRAM area: Always input 0s. (Input 0s to EPROM addresses 00H to FFH.)
3. EPROM data used when the user does not use any HD44780U character pattern: According to the user application, handled in one of the two ways listed as follows.
   a. When unused character patterns are not programmed: If an unused character code is written into DDRAM, all its dots are lit. By not programing a character pattern, all of its bits become lit. (This is due to the EPROM being filled with 1s after it is erased.)
   b. When unused character patterns are programmed as 0s: Nothing is displayed even if unused character codes are written into DDRAM. (This is equivalent to a space.)

Table 3 Example of Correspondence between EPROM Address Data and Character Pattern (5 × 10 Dots)

EPROM Address												Data					
A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	O4	O3	O2	O1	LSB O0	
								0	0	0	0	0	0	0	0	0	
								0	0	0	1	0	0	0	0	0	
								0	0	1	0	0	1	1	0	1	
								0	0	1	1	1	0	0	1	1	
								0	1	0	0	1	0	0	0	1	
								0	1	0	1	1	0	0	0	1	
								0	1	1	0	0	1	1	1	1	
0	1	0	1	0	0	1	0	0	1	1	1	0	0	0	0	1	
								1	0	0	0	0	0	0	0	1	
								1	0	0	1	0	0	0	0	1	
								1	0	1	0	0	0	0	0	0	← Cursor position
								1	0	1	1	0	0	0	0	0	
								1	1	0	0	0	0	0	0	0	
								1	1	0	1	0	0	0	0	0	
								1	1	1	0	0	0	0	0	0	
								1	1	1	1	0	0	0	0	0	

Character code — Line position

Notes: 1. EPROM addresses A11 to A3 correspond to a character code.
2. EPROM addresses A3 to A0 specify a line position of the character pattern.
3. EPROM data O4 to O0 correspond to character pattern data.
4. EPROM data O5 to O7 must be specified as 0.
5. A lit display position (black) corresponds to a 1.
6. Line 11 and the following lines must be blanked with 0s for a 5 × 10 dot character fonts.

Table 4 Correspondence between Character Codes and Character Patterns (ROM Code: A00)

Upper 4 Bits / Lower 4 Bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000	CG RAM (1)															
xxxx0001	(2)															
xxxx0010	(3)															
xxxx0011	(4)															
xxxx0100	(5)															
xxxx0101	(6)															
xxxx0110	(7)															
xxxx0111	(8)															
xxxx1000	(1)															
xxxx1001	(2)															
xxxx1010	(3)															
xxxx1011	(4)															
xxxx1100	(5)															
xxxx1101	(6)															
xxxx1110	(7)															
xxxx1111	(8)															

Note: The user can specify any pattern for character-generator RAM.

Table 4 Correspondence between Character Codes and Character Patterns (ROM Code: A02)

Upper 4 Bits / Lower 4 Bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000	CG RAM (1)															
xxxx0001	(2)															
xxxx0010	(3)															
xxxx0011	(4)															
xxxx0100	(5)															
xxxx0101	(6)															
xxxx0110	(7)															
xxxx0111	(8)															
xxxx1000	(1)															
xxxx1001	(2)															
xxxx1010	(3)															
xxxx1011	(4)															
xxxx1100	(5)															
xxxx1101	(6)															
xxxx1110	(7)															
xxxx1111	(8)															

Table 5 Relationship between CGRAM Addresses, Character Codes (DDRAM) and Character Patterns (CGRAM Data)

For 5 × 8 dot character patterns

Character Codes (DDRAM data)	CGRAM Address	Character Patterns (CGRAM data)	
7 6 5 4 3 2 1 0 High Low	5 4 3 2 1 0 High Low	7 6 5 4 3 2 1 0 High Low	
0 0 0 0 * 0 0 0	0 0 0	0 0 0 * * * 1 1 1 1 0 0 0 1 * 1 0 0 0 1 0 1 0 * 1 0 0 0 1 0 1 1 * 1 1 1 1 0 1 0 0 * 1 0 1 0 0 1 0 1 * 1 0 0 1 0 1 1 0 * 1 0 0 0 1 1 1 1 * * * 0 0 0 0 0	Character pattern (1) Cursor position
0 0 0 0 * 0 0 1	0 0 1	0 0 0 * * * 1 0 0 0 1 0 0 1 * 0 1 0 1 0 0 1 0 * 1 1 1 1 1 0 1 1 * 0 0 1 0 0 1 0 0 * 1 1 1 1 1 1 0 1 * 0 0 1 0 0 1 1 0 * 0 0 1 0 0 1 1 1 * * * 0 0 0 0 0	Character pattern (2) Cursor position
		0 0 0 * * * 0 0 1	
0 0 0 0 * 1 1 1	1 1 1	1 0 0 1 0 1 1 1 0 1 1 1 * * *	

Notes: 1. Character code bits 0 to 2 correspond to CGRAM address bits 3 to 5 (3 bits: 8 types).

2. CGRAM address bits 0 to 2 designate the character pattern line position. The 8th line is the cursor position and its display is formed by a logical OR with the cursor.

   Maintain the 8th line data, corresponding to the cursor display position, at 0 as the cursor display.

   If the 8th line data is 1, 1 bits will light up the 8th line regardless of the cursor presence.

3. Character pattern row positions correspond to CGRAM data bits 0 to 4 (bit 4 being at the left).

4. As shown Table 5, CGRAM character patterns are selected when character code bits 4 to 7 are all 0. However, since character code bit 3 has no effect, the R display example above can be selected by either character code 00H or 08H.

5. 1 for CGRAM data corresponds to display selection and 0 to non-selection.

* Indicates no effect.

Table 5 Relationship between CGRAM Addresses, Character Codes (DDRAM) and Character Patterns (CGRAM Data) (cont)

For 5 × 10 dot character patterns

Character Codes (DDRAM data)	CGRAM Address	Character Patterns (CGRAM data)	
7 6 5 4 3 2 1 0 High Low	5 4 3 2 1 0 High Low	7 6 5 4 3 2 1 0 High Low	
0 0 0 0 * 0 0 *	0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 1 0 1 0 0 0 1 0 1 0 1 1 0 0 1 1 1 1 0 0 0 1 0 0 1 1 0 1 0	* * * 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 1 1 0 0 1 1 0 0 0 1 1 0 0 0 1 1 1 1 1 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 * * * 0 0 0 0 0	Character pattern Cursor position
	1 0 1 1 1 1 0 0 1 1 0 1 1 1 1 0 1 1 1 1	* * * * * * * * * * * * * * * * * *	
0 0 0 0 * 1 1 *	0 0 0 0 0 0 0 1 1 1 1 0 0 1 1 0 1 0	* * * * * *	
	1 0 1 1 1 1 0 0 1 1 0 1 1 1 1 0 1 1 1 1	* * * * * * * * * * * * * * * * * *	

Notes: 1. Character code bits 1 and 2 correspond to CGRAM address bits 4 and 5 (2 bits: 4 types).

2. CGRAM address bits 0 to 3 designate the character pattern line position. The 11th line is the cursor position and its display is formed by a logical OR with the cursor.
   Maintain the 11th line data corresponding to the cursor display positon at 0 as the cursor display.
   If the 11th line data is “1”, “1” bits will light up the 11th line regardless of the cursor presence.
   Since lines 12 to 16 are not used for display, they can be used for general data RAM.
3. Character pattern row positions are the same as 5 × 8 dot character pattern positions.
4. CGRAM character patterns are selected when character code bits 4 to 7 are all 0.
   However, since character code bits 0 and 3 have no effect, the P display example above can be selected by character codes 00H, 01H, 08H, and 09H.
5. 1 for CGRAM data corresponds to display selection and 0 to non-selection.

* Indicates no effect.

Timing Generation Circuit

The timing generation circuit generates timing signals for the operation of internal circuits such as DDRAM, CGROM and CGRAM. RAM read timing for display and internal operation timing by MPU access are generated separately to avoid interfering with each other. Therefore, when writing data to DDRAM, for example, there will be no undesirable interferences, such as flickering, in areas other than the display area.

Liquid Crystal Display Driver Circuit

The liquid crystal display driver circuit consists of 16 common signal drivers and 40 segment signal drivers. When the character font and number of lines are selected by a program, the required common signal drivers automatically output drive waveforms, while the other common signal drivers continue to output non-selection waveforms.

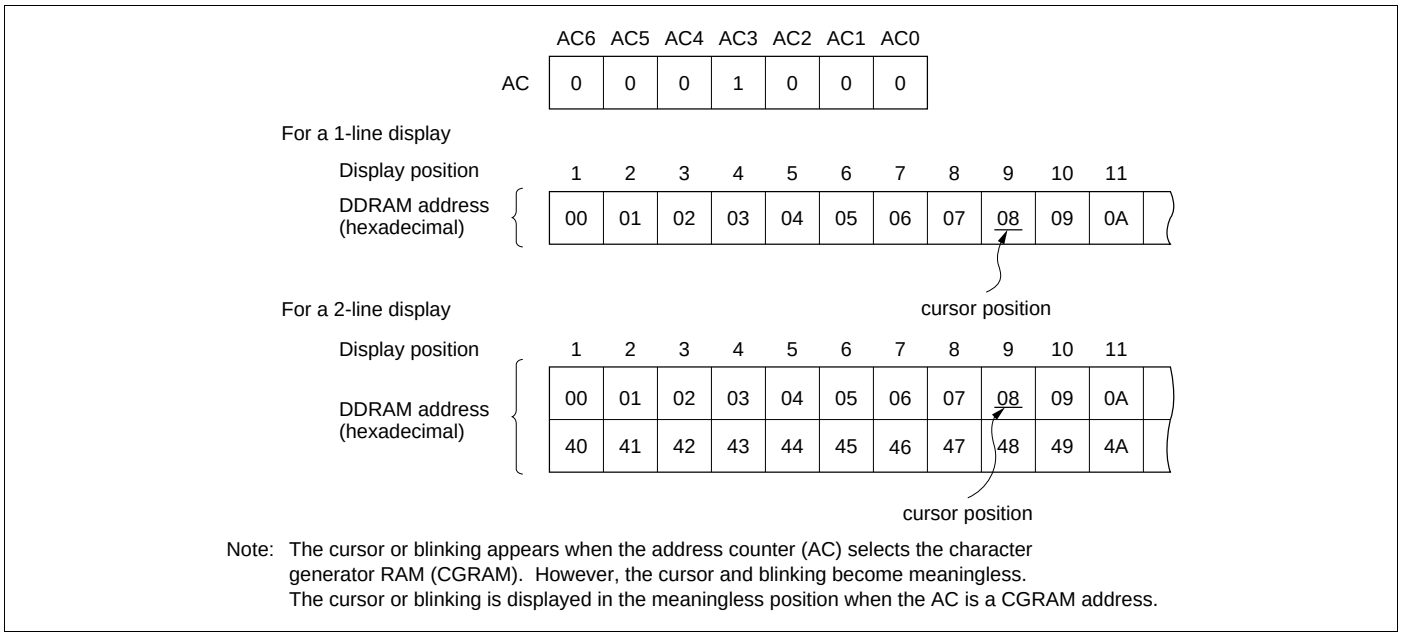
Sending serial data always starts at the display data character pattern corresponding to the last address of the display data RAM (DDRAM).

Since serial data is latched when the display data character pattern corresponding to the starting address enters the internal shift register, the HD44780U drives from the head display.

Cursor/Blink Control Circuit

The cursor/blink control circuit generates the cursor or character blinking. The cursor or the blinking will appear with the digit located at the display data RAM (DDRAM) address set in the address counter (AC).

For example (Figure 8), when the address counter is 08H, the cursor position is displayed at DDRAM address 08H.



Note: The cursor or blinking appears when the address counter (AC) selects the character generator RAM (CGRAM). However, the cursor and blinking become meaningless. The cursor or blinking is displayed in the meaningless position when the AC is a CGRAM address.

Figure 8 Cursor/Blink Display Example

Interfacing to the MPU

The HD44780U can send data in either two 4-bit operations or one 8-bit operation, thus allowing interfacing with 4- or 8-bit MPUs.

- For 4-bit interface data, only four bus lines (DB4 to DB7) are used for transfer. Bus lines DB0 to DB3 are disabled. The data transfer between the HD44780U and the MPU is completed after the 4-bit data has been transferred twice. As for the order of data transfer, the four high order bits (for 8-bit operation, DB4 to DB7) are transferred before the four low order bits (for 8-bit operation, DB0 to DB3).

  The busy flag must be checked (one instruction) after the 4-bit data has been transferred twice. Two more 4-bit operations then transfer the busy flag and address counter data.
- For 8-bit interface data, all eight bus lines (DB0 to DB7) are used.

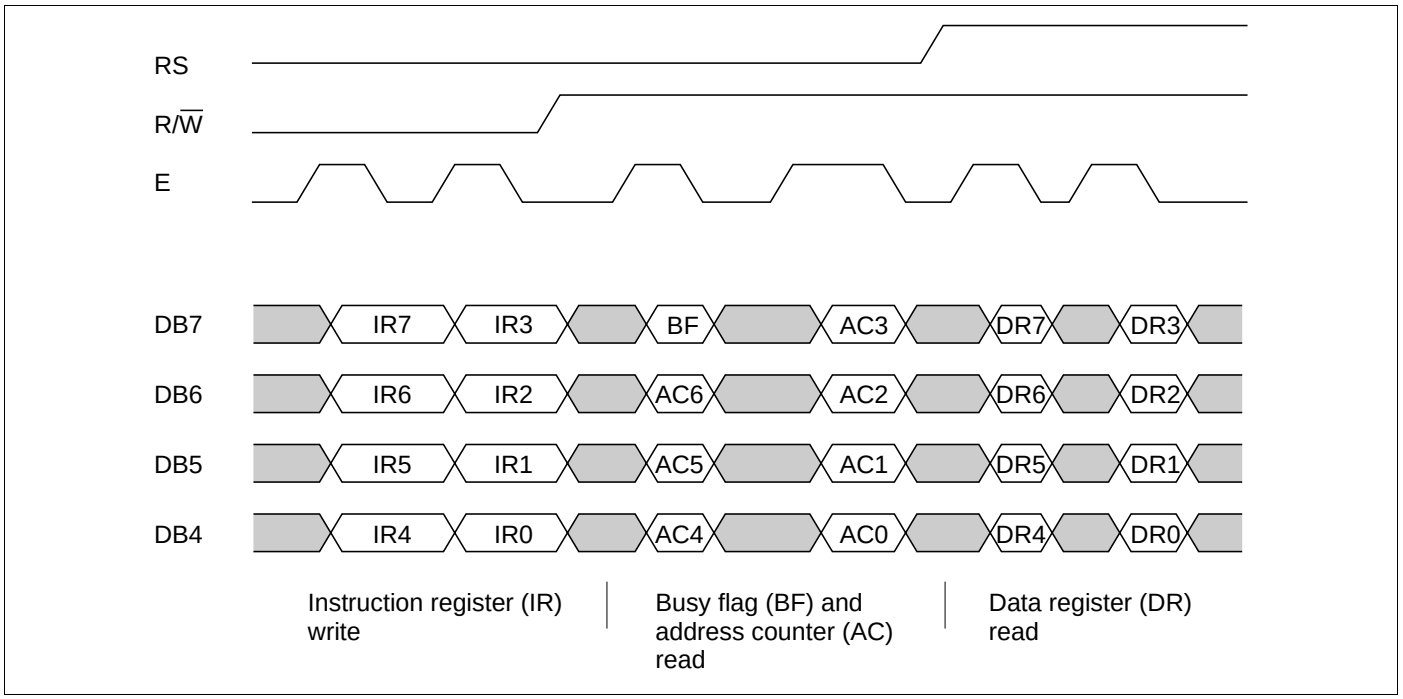


Figure 9 4-Bit Transfer Example

Reset Function

Initializing by Internal Reset Circuit

An internal reset circuit automatically initializes the HD44780U when the power is turned on. The following instructions are executed during the initialization. The busy flag (BF) is kept in the busy state until the initialization ends (BF = 1). The busy state lasts for 10 ms after V_{CC} rises to 4.5 V.

1. Display clear
2. Function set:
   DL = 1; 8-bit interface data
   N = 0; 1-line display
   F = 0; 5 × 8 dot character font
3. Display on/off control:
   D = 0; Display off
   C = 0; Cursor off
   B = 0; Blinking off
4. Entry mode set:
   I/D = 1; Increment by 1
   S = 0; No shift

Note: If the electrical characteristics conditions listed under the table Power Supply Conditions Using Internal Reset Circuit are not met, the internal reset circuit will not operate normally and will fail to initialize the HD44780U. For such a case, initial-ization must be performed by the MPU as explained in the section, Initializing by Instruction.

Instructions

Outline

Only the instruction register (IR) and the data register (DR) of the HD44780U can be controlled by the MPU. Before starting the internal operation of the HD44780U, control information is temporarily stored into these registers to allow interfacing with various MPUs, which operate at different speeds, or various peripheral control devices. The internal operation of the HD44780U is determined by signals sent from the MPU. These signals, which include register selection signal (RS), read/

write signal (R/$\overline{W}$), and the data bus (DB0 to DB7), make up the HD44780U instructions (Table 6). There are four categories of instructions that:

- Designate HD44780U functions, such as display format, data length, etc.
- Set internal RAM addresses
- Perform data transfer with internal RAM
- Perform miscellaneous functions

Normally, instructions that perform data transfer with internal RAM are used the most. However, auto-incrementation by 1 (or auto-decrementation by 1) of internal HD44780U RAM addresses after each data write can lighten the program load of the MPU. Since the display shift instruction (Table 11) can perform concurrently with display data write, the user can minimize system development time with maximum programming efficiency.

When an instruction is being executed for internal operation, no instruction other than the busy flag/address read instruction can be executed.

Because the busy flag is set to 1 while an instruction is being executed, check it to make sure it is 0 before sending another instruction from the MPU.

Note: Be sure the HD44780U is not in the busy state (BF = 0) before sending an instruction from the MPU to the HD44780U. If an instruction is sent without checking the busy flag, the time between the first instruction and next instruction will take much longer than the instruction time itself. Refer to Table 6 for the list of each instruc-tion execution time.

Table 6 Instructions

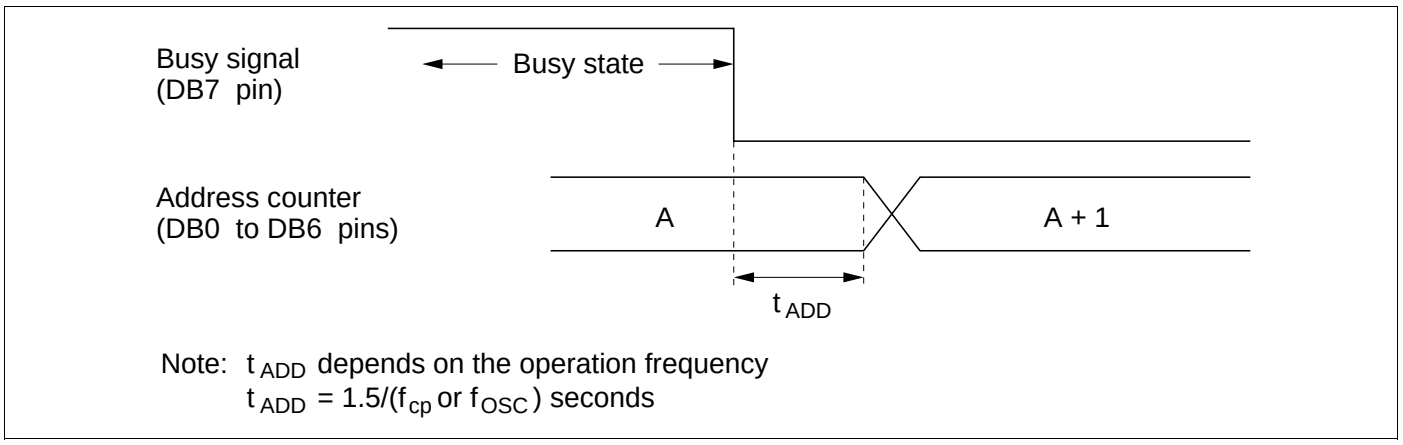
	Code											
Instruction	**RS**	**R/$\overline{W}$**	**DB7**	**DB6**	**DB5**	**DB4**	**DB3**	**DB2**	**DB1**	**DB0**	**Description**	**Execution Time (max) (when f_{cp} or f_{osc} is 270 kHz)**
Clear display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DDRAM address 0 in address counter.	
Return home	0	0	0	0	0	0	0	0	1	—	Sets DDRAM address 0 in address counter. Also returns display from being shifted to original position. DDRAM contents remain unchanged.	1.52 ms
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies display shift. These operations are performed during data write and read.	37 µs
Display on/off control	0	0	0	0	0	0	1	D	C	B	Sets entire display (D) on/off, cursor on/off (C), and blinking of cursor position character (B).	37 µs
Cursor or display shift	0	0	0	0	0	1	S/C	R/L	—	—	Moves cursor and shifts display without changing DDRAM contents.	37 µs
Function set	0	0	0	0	1	DL	N	F	—	—	Sets interface data length (DL), number of display lines (N), and character font (F).	37 µs
Set CGRAM address	0	0	0	1	ACG	ACG	ACG	ACG	ACG	ACG	Sets CGRAM address. CGRAM data is sent and received after this setting.	37 µs
Set DDRAM address	0	0	1	ADD	ADD	ADD	ADD	ADD	ADD	ADD	Sets DDRAM address. DDRAM data is sent and received after this setting.	37 µs
Read busy flag & address	0	1	BF	AC	AC	AC	AC	AC	AC	AC	Reads busy flag (BF) indicating internal operation is being performed and reads address counter contents.	0 µs

Table 6 Instructions (cont)

Instruction	Code RS	R/$\overline{W}$	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description	Execution Time (max) (when f_{cp} or f_{OSC} is 270 kHz)
Write data to CG or DDRAM	1	0	Write data								Writes data into DDRAM or CGRAM.	37 µs t_{ADD} = 4 µs*
Read data from CG or DDRAM	1	1	Read data								Reads data from DDRAM or CGRAM.	37 µs t_{ADD} = 4 µs*
	I/D = 1: Increment I/D = 0: Decrement S = 1: Accompanies display shift S/C = 1: Display shift S/C = 0: Cursor move R/L = 1: Shift to the right R/L = 0: Shift to the left DL = 1: 8 bits, DL = 0: 4 bits N = 1: 2 lines, N = 0: 1 line F = 1: 5 × 10 dots, F = 0: 5 × 8 dots BF = 1: Internally operating BF = 0: Instructions acceptable										DDRAM: Display data RAM CGRAM: Character generator RAM ACG: CGRAM address ADD: DDRAM address (corresponds to cursor address) AC: Address counter used for both DD and CGRAM addresses	Execution time changes when frequency changes Example: When f_{cp} or f_{OSC} is 250 kHz, $37\ \mu s \times \frac{270}{250} = 40\ \mu s$

Note: — indicates no effect.

* After execution of the CGRAM/DDRAM data write or read instruction, the RAM address counter is incremented or decremented by 1. The RAM address counter is updated after the busy flag turns off. In Figure 10, t_{ADD} is the time elapsed after the busy flag turns off until the address counter is updated.



Note: t_{ADD} depends on the operation frequency
t_{ADD} = 1.5/(f_{cp} or f_{OSC}) seconds

Figure 10 Address Counter Update

Instruction Description

Clear Display

Clear display writes space code 20H (character pattern for character code 20H must be a blank pattern) into all DDRAM addresses. It then sets DDRAM address 0 into the address counter, and returns the display to its original status if it was shifted. In other words, the display disappears and the cursor or blinking goes to the left edge of the display (in the first line if 2 lines are displayed). It also sets I/D to 1 (increment mode) in entry mode. S of entry mode does not change.

Return Home

Return home sets DDRAM address 0 into the address counter, and returns the display to its original status if it was shifted. The DDRAM contents do not change.

The cursor or blinking go to the left edge of the display (in the first line if 2 lines are displayed).

Entry Mode Set

I/D: Increments (I/D = 1) or decrements (I/D = 0) the DDRAM address by 1 when a character code is written into or read from DDRAM.

The cursor or blinking moves to the right when incremented by 1 and to the left when decremented by 1. The same applies to writing and reading of CGRAM.

S: Shifts the entire display either to the right (I/D = 0) or to the left (I/D = 1) when S is 1. The display does not shift if S is 0.

If S is 1, it will seem as if the cursor does not move but the display does. The display does not shift when reading from DDRAM. Also, writing into or reading out from CGRAM does not shift the display.

Display On/Off Control

D: The display is on when D is 1 and off when D is 0. When off, the display data remains in DDRAM, but can be displayed instantly by setting D to 1.

C: The cursor is displayed when C is 1 and not displayed when C is 0. Even if the cursor disappears, the function of I/D or other specifications will not change during display data write. The cursor is displayed using 5 dots in the 8th line for 5×8 dot character font selection and in the 11th line for the 5×10 dot character font selection (Figure 13).

B: The character indicated by the cursor blinks when B is 1 (Figure 13). The blinking is displayed as switching between all blank dots and displayed characters at a speed of 409.6-ms intervals when f_{cp} or f_{OSC} is 250 kHz. The cursor and blinking can be set to display simultaneously. (The blinking frequency changes according to f_{OSC} or the reciprocal of f_{cp}. For example, when f_{cp} is 270 kHz, $409.6 \times 250/270 = 379.2$ ms.)

Cursor or Display Shift

Cursor or display shift shifts the cursor position or display to the right or left without writing or reading display data (Table 7). This function is used to correct or search the display. In a 2-line display, the cursor moves to the second line when it passes the 40th digit of the first line. Note that the first and second line displays will shift at the same time.

When the displayed data is shifted repeatedly each line moves only horizontally. The second line display does not shift into the first line position.

The address counter (AC) contents will not change if the only action performed is a display shift.

Function Set

DL: Sets the interface data length. Data is sent or received in 8-bit lengths (DB7 to DB0) when DL is 1, and in 4-bit lengths (DB7 to DB4) when DL is 0.When 4-bit length is selected, data must be sent or received twice.

N: Sets the number of display lines.

F: Sets the character font.

Note: Perform the function at the head of the program before executing any instructions (except for the read busy flag and address instruction). From this point, the function set instruction cannot be executed unless the interface data length is changed.

Set CGRAM Address

Set CGRAM address sets the CGRAM address binary AAAAAA into the address counter.

Data is then written to or read from the MPU for CGRAM.

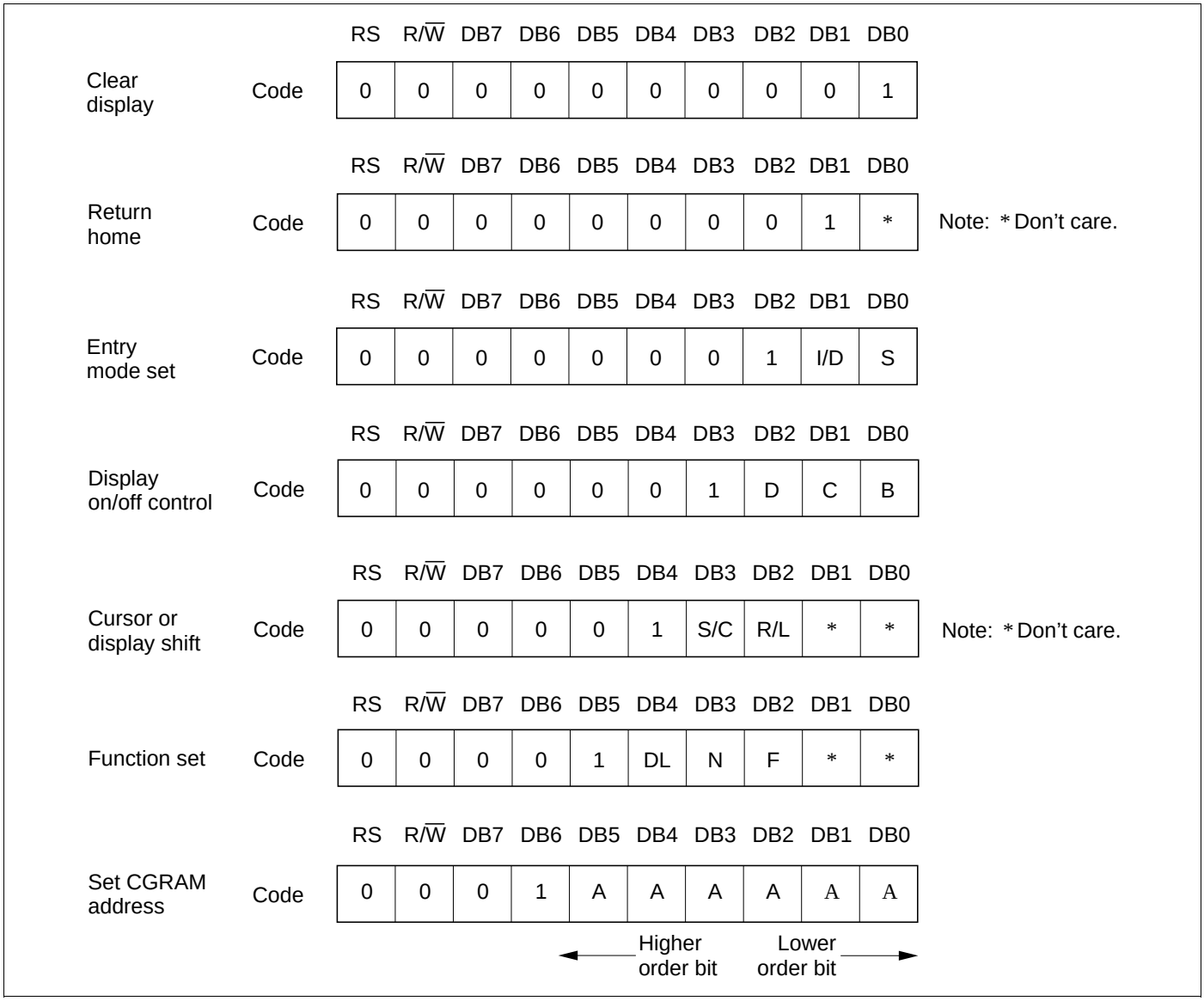


Figure 11 Instruction (1)

Set DDRAM Address

Set DDRAM address sets the DDRAM address binary AAAAAAA into the address counter.

Data is then written to or read from the MPU for DDRAM.

However, when N is 0 (1-line display), AAAAAAA can be 00H to 4FH. When N is 1 (2-line display), AAAAAAA can be 00H to 27H for the first line, and 40H to 67H for the second line.

Read Busy Flag and Address

Read busy flag and address reads the busy flag (BF) indicating that the system is now internally operating on a previously received instruction. If BF is 1, the internal operation is in progress. The next instruction will not be accepted until BF is reset to 0. Check the BF status before the next write operation. At the same time, the value of the address counter in binary AAAAAAA is read out. This address counter is used by both CG and DDRAM addresses, and its value is determined by the previous instruction. The address contents are the same as for instructions set CGRAM address and set DDRAM address.

Table 7 Shift Function

S/C	R/L	
0	0	Shifts the cursor position to the left. (AC is decremented by one.)
0	1	Shifts the cursor position to the right. (AC is incremented by one.)
1	0	Shifts the entire display to the left. The cursor follows the display shift.
1	1	Shifts the entire display to the right. The cursor follows the display shift.

Table 8 Function Set

N	F	No. of Display Lines	Character Font	Duty Factor	Remarks
0	0	1	5 × 8 dots	1/8	
0	1	1	5 × 10 dots	1/11	
1	*	2	5 × 8 dots	1/16	Cannot display two lines for 5 × 10 dot character font

Note: * Indicates don't care.

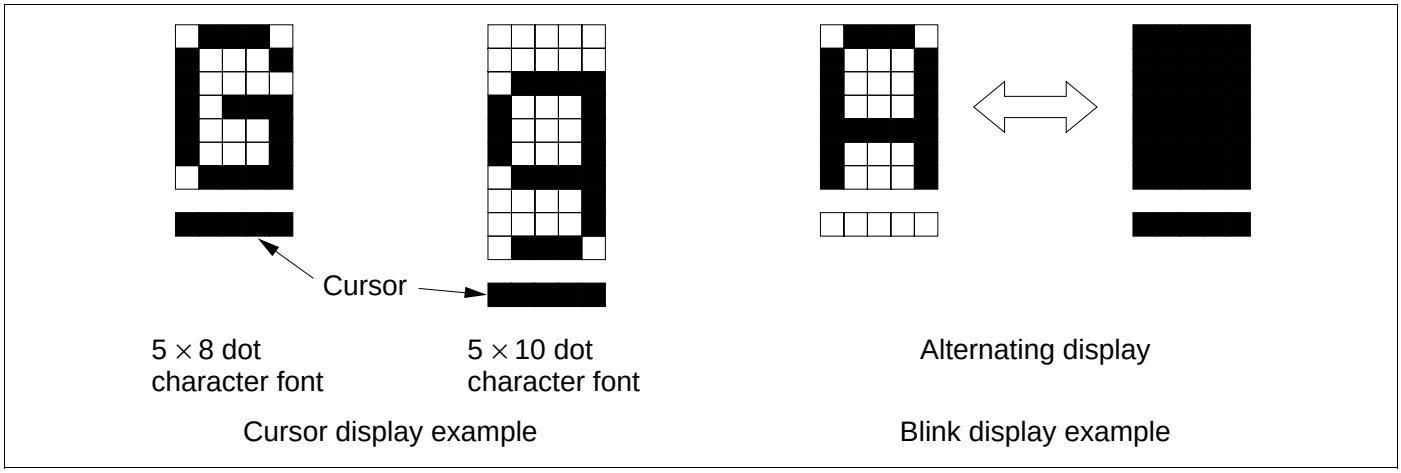


Figure 12 Cursor and Blinking

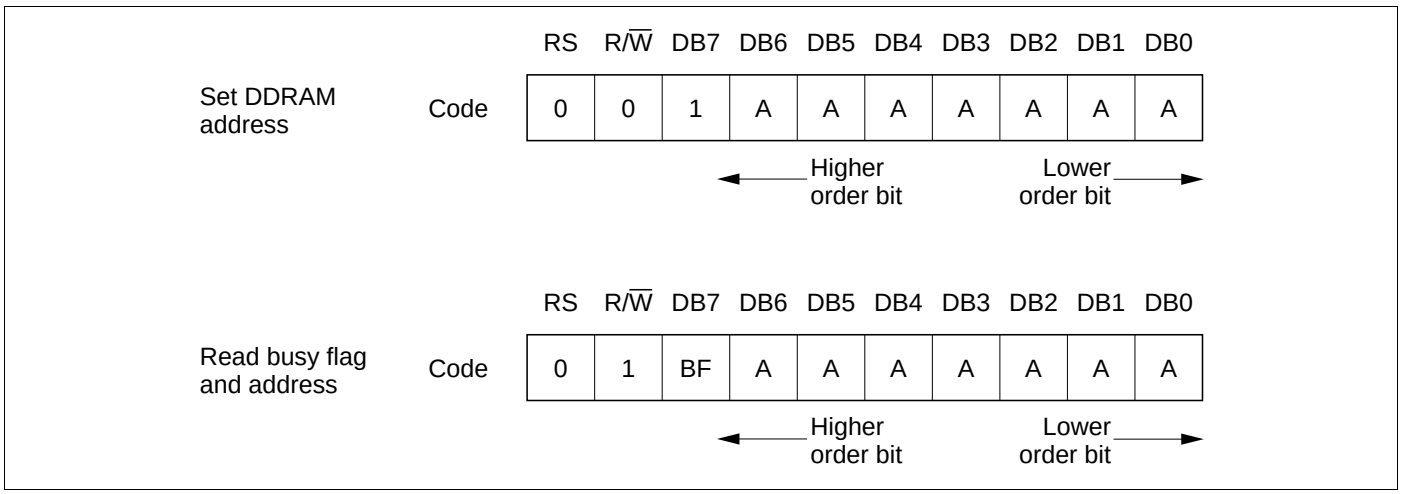


Figure 13 Instruction (2)

Write Data to CG or DDRAM

Write data to CG or DDRAM writes 8-bit binary data DDDDDDDD to CG or DDRAM.

To write into CG or DDRAM is determined by the previous specification of the CGRAM or DDRAM address setting. After a write, the address is automatically incremented or decremented by 1 according to the entry mode. The entry mode also determines the display shift.

Read Data from CG or DDRAM

Read data from CG or DDRAM reads 8-bit binary data DDDDDDDD from CG or DDRAM.

The previous designation determines whether CG or DDRAM is to be read. Before entering this read instruction, either CGRAM or DDRAM address set instruction must be executed. If not executed, the first read data will be invalid. When serially executing read instructions, the next address data is normally read from the second read. The address set instructions need not be executed just before this read instruction when shifting the cursor by the cursor shift instruction (when reading out DDRAM). The operation of the cursor shift instruction is the same as the set DDRAM address instruction.

After a read, the entry mode automatically increases or decreases the address by 1. However, display shift is not executed regardless of the entry mode.

Note: The address counter (AC) is automatically incremented or decremented by 1 after the write instructions to CGRAM or DDRAM are executed. The RAM data selected by the AC cannot be read out at this time even if read instructions are executed. Therefore, to correctly read data, execute either the address set instruction or cursor shift instruction (only with DDRAM), then just before reading the desired data, execute the read instruction from the second time the read instruction is sent.

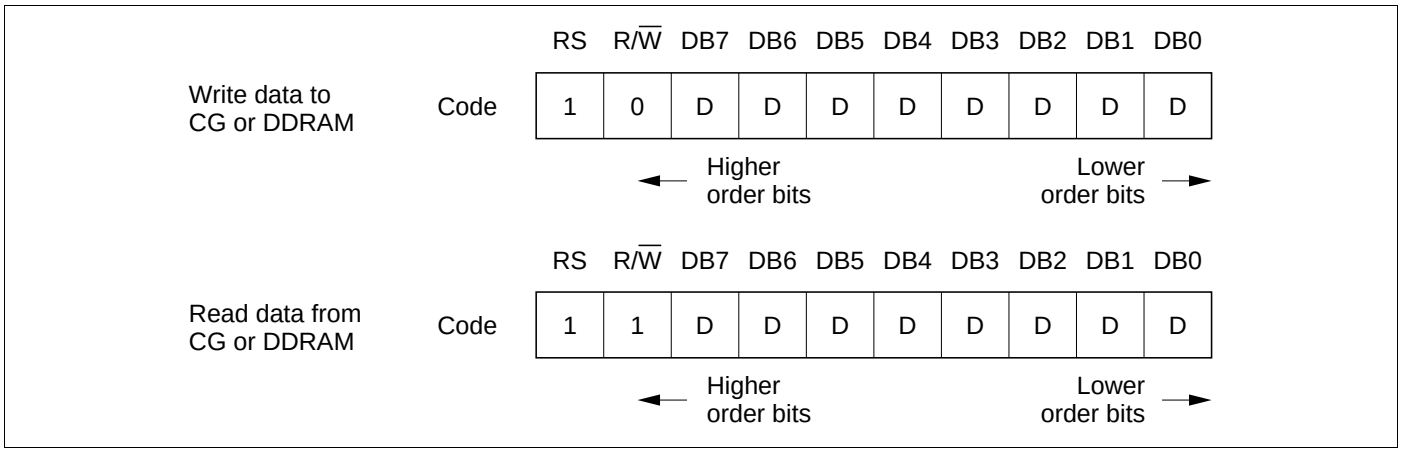


Figure 14 Instruction (3)

Interfacing the HD44780U

Interface to MPUs

- Interfacing to an 8-bit MPU

  See Figure 16 for an example of using a I/O port (for a single-chip microcomputer) as an interface device.

  In this example, P30 to P37 are connected to the data bus DB0 to DB7, and P75 to P77 are connected to E, R/$\overline{W}$, and RS, respectively.

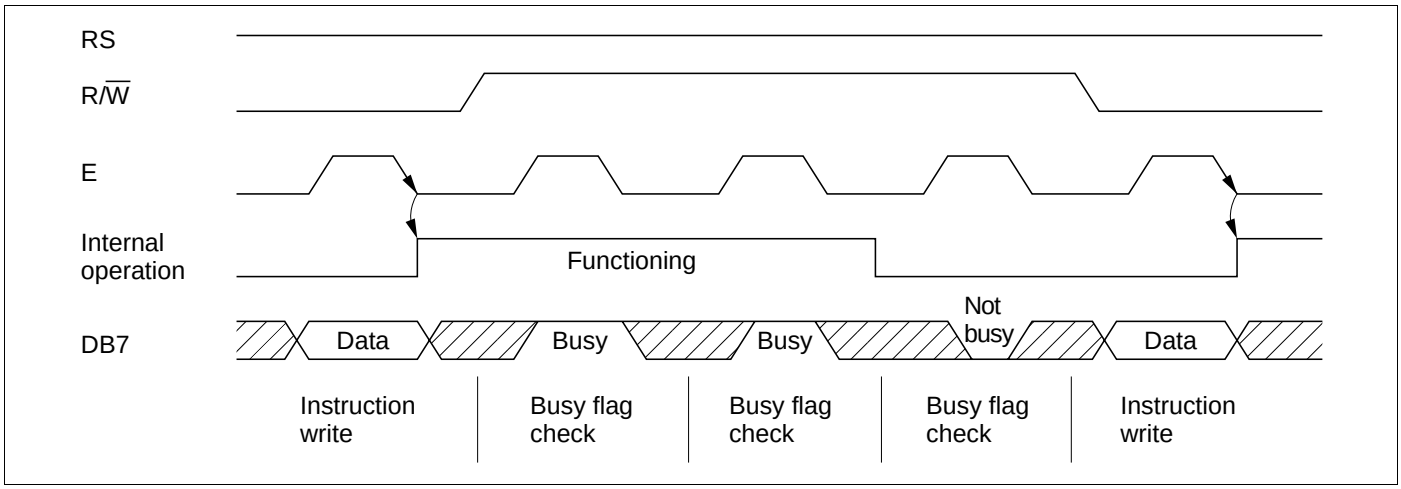


Figure 15 Example of Busy Flag Check Timing Sequence

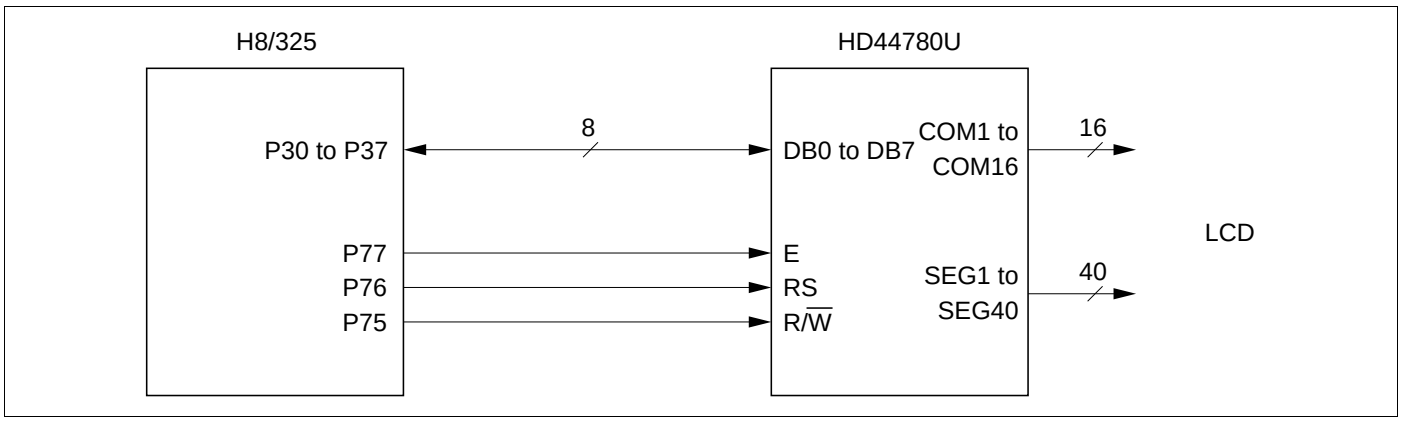


Figure 16 H8/325 Interface (Single-Chip Mode)

- Interfacing to a 4-bit MPU

  The HD44780U can be connected to the I/O port of a 4-bit MPU. If the I/O port has enough bits, 8-bit data can be transferred. Otherwise, one data transfer must be made in two operations for 4-bit data. In this case, the timing sequence becomes somewhat complex. (See Figure 17.)

  See Figure 18 for an interface example to the HMCS4019R.

  Note that two cycles are needed for the busy flag check as well as for the data transfer. The 4-bit operation is selected by the program.

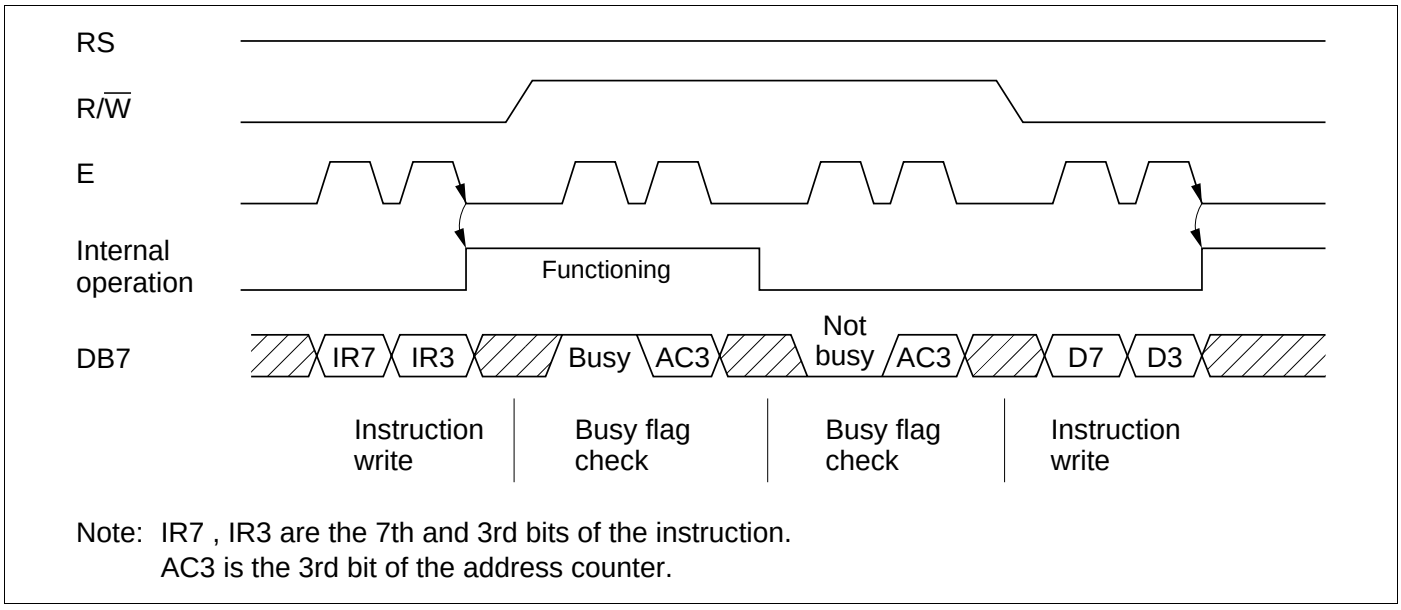


Figure 17 Example of 4-Bit Data Transfer Timing Sequence

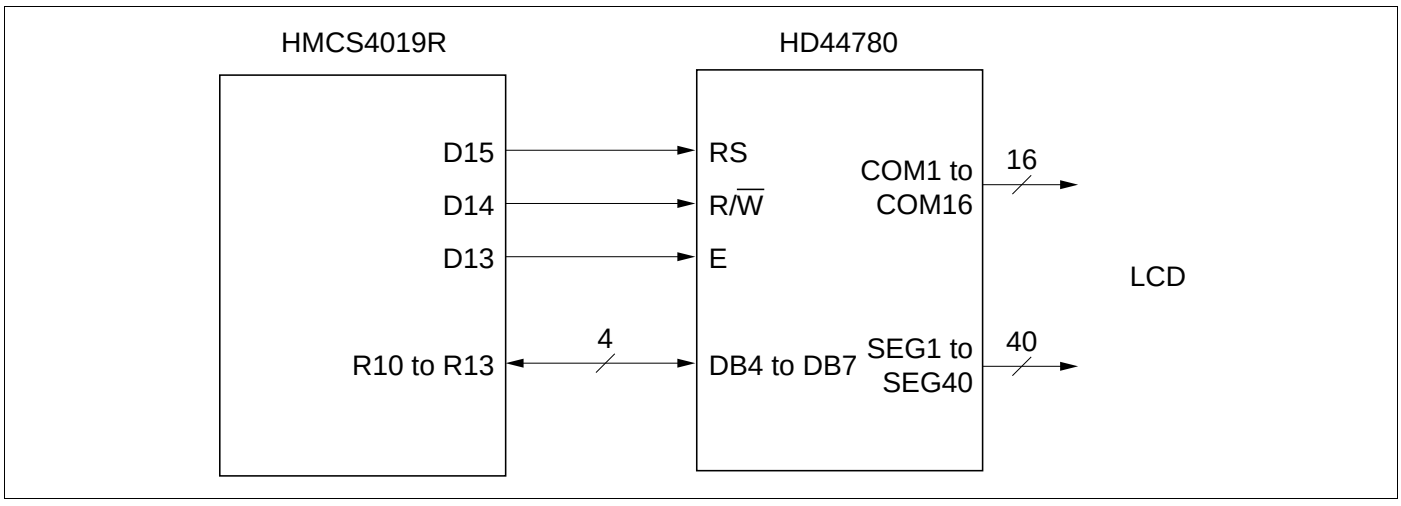


Figure 18 Example of Interface to HMCS4019R

Interface to Liquid Crystal Display

Character Font and Number of Lines: The HD44780U can perform two types of displays, 5 × 8 dot and 5 × 10 dot character fonts, each with a cursor.

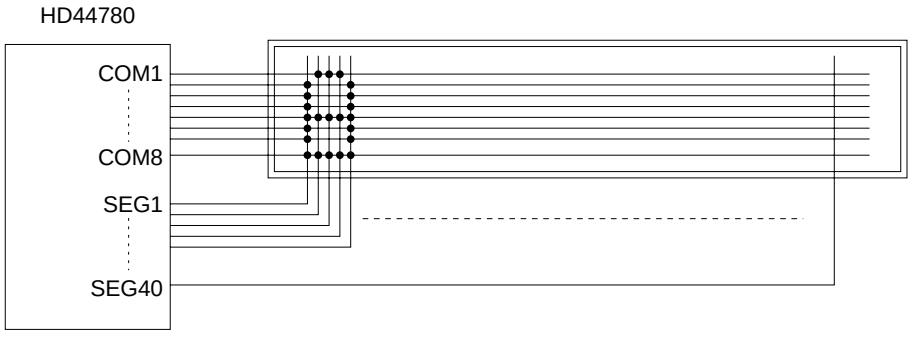
Up to two lines are displayed for 5 × 8 dots and one line for 5 × 10 dots. Therefore, a total of three types of common signals are available (Table 9).

The number of lines and font types can be selected by the program. (See Table 6, Instructions.)

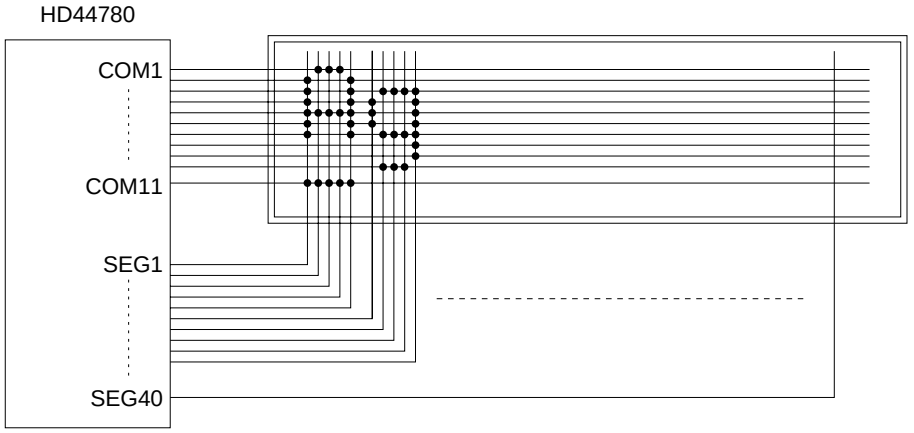
Connection to HD44780 and Liquid Crystal Display: See Figure 19 for the connection examples.

Table 9 Common Signals

Number of Lines	Character Font	Number of Common Signals	Duty Factor
1	5 × 8 dots + cursor	8	1/8
1	5 × 10 dots + cursor	11	1/11
2	5 × 8 dots + cursor	16	1/16



Example of a 5 × 8 dot, 8-character × 1-line display (1/4 bias, 1/8 duty cycle)

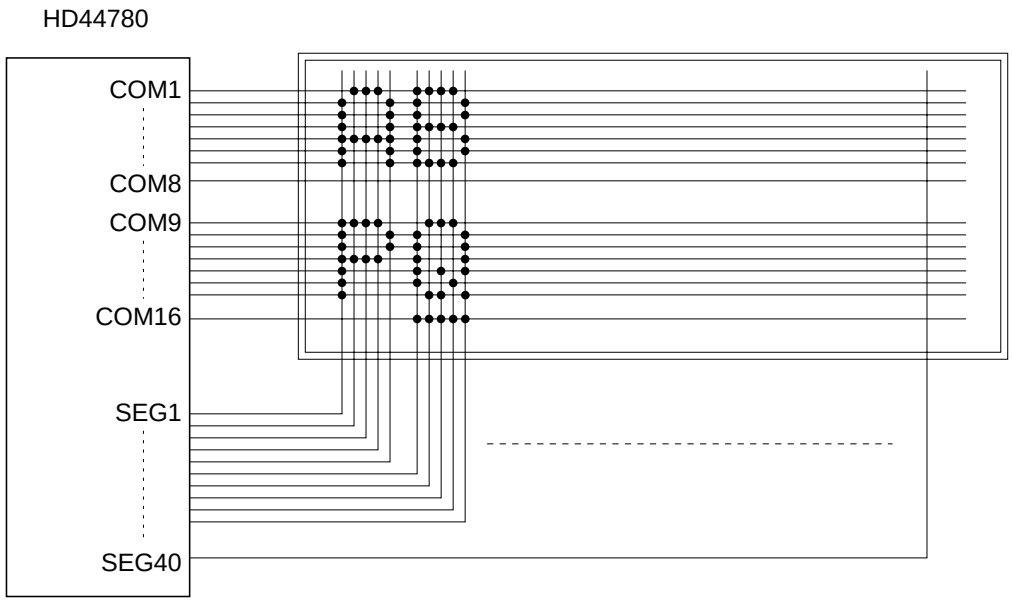


Example of a 5 × 10 dot, 8-character × 1-line display (1/4 bias, 1/11 duty cycle)

Figure 19 Liquid Crystal Display and HD44780 Connections

Since five segment signal lines can display one digit, one HD44780U can display up to 8 digits for a 1-line display and 16 digits for a 2-line display.

The examples in Figure 19 have unused common signal pins, which always output non-selection waveforms. When the liquid crystal display panel has unused extra scanning lines, connect the extra scanning lines to these common signal pins to avoid any undesirable effects due to crosstalk during the floating state.



Example of a 5 × 8 dot, 8-character × 2-line display (1/5 bias, 1/16 duty cycle)

Figure 19 Liquid Crystal Display and HD44780 Connections (cont)

Connection of Changed Matrix Layout: In the preceding examples, the number of lines correspond to the scanning lines. However, the following display examples (Figure 20) are made possible by altering the matrix layout of the liquid crystal display panel. In either case, the only change is the layout. The display characteristics and the number of liquid crystal display characters depend on the number of common signals or on duty factor. Note that the display data RAM (DDRAM) addresses for 4 characters × 2 lines and for 16 characters × 1 line are the same as in Figure 19.

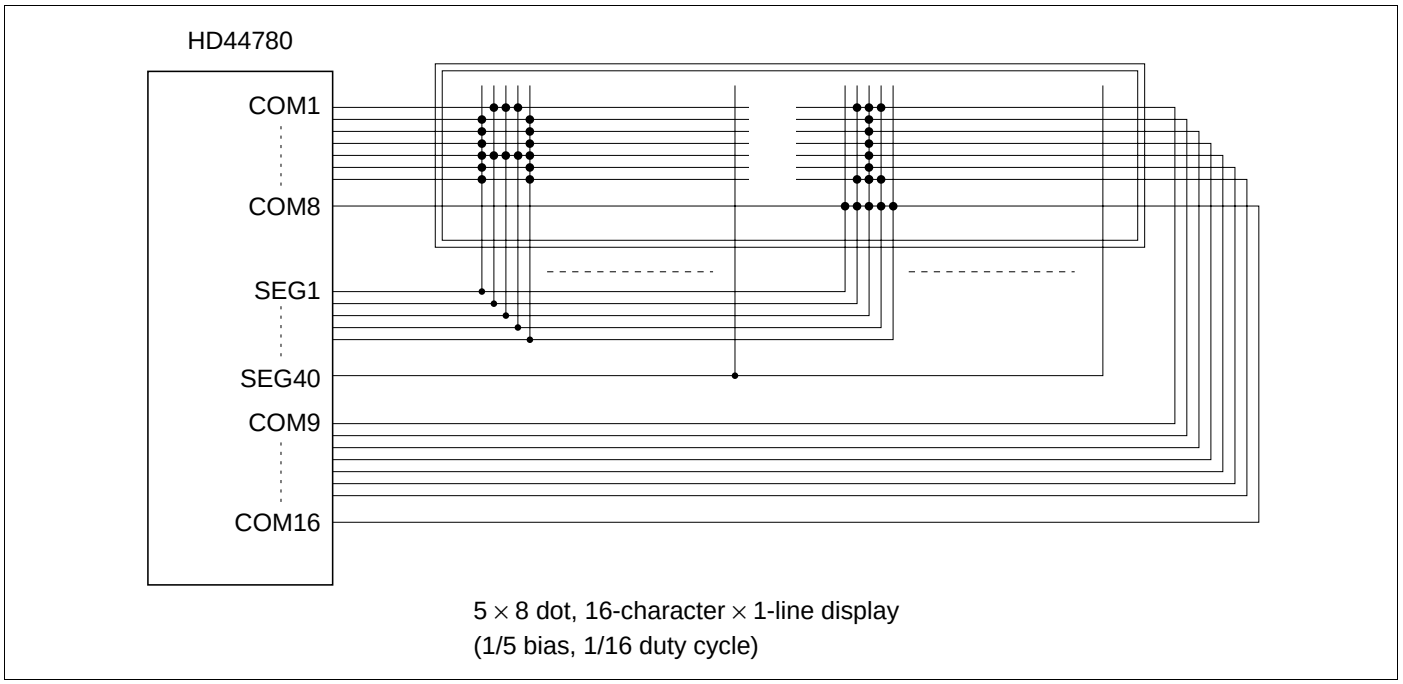


Figure 20 Changed Matrix Layout Displays

Power Supply for Liquid Crystal Display Drive

Various voltage levels must be applied to pins V1 to V5 of the HD44780U to obtain the liquid crystal display drive waveforms. The voltages must be changed according to the duty factor (Table 10).

VLCD is the peak value for the liquid crystal display drive waveforms, and resistance dividing provides voltages V1 to V5 (Figure 21).

Table 10 Duty Factor and Power Supply for Liquid Crystal Display Drive

	Duty Factor	
	1/8, 1/11	1/16
		Bias
Power Supply	1/4	1/5
V1	V_{CC}–1/4 VLCD	V_{CC}–1/5 VLCD
V2	V_{CC}–1/2 VLCD	V_{CC}–2/5 VLCD
V3	V_{CC}–1/2 VLCD	V_{CC}–3/5 VLCD
V4	V_{CC}–3/4 VLCD	V_{CC}–4/5 VLCD
V5	V_{CC}–VLCD	V_{CC}–VLCD

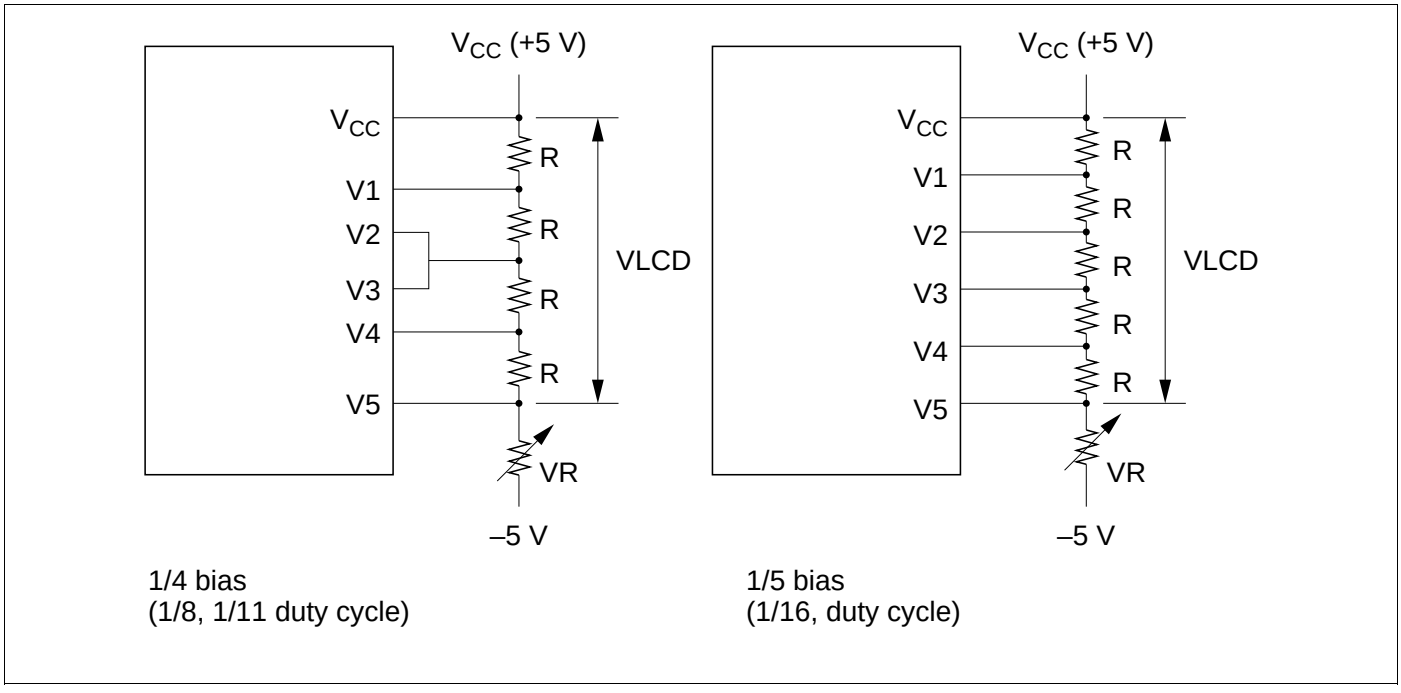


Figure 21 Drive Voltage Supply Example

Relationship between Oscillation Frequency and Liquid Crystal Display Frame Frequency

The liquid crystal display frame frequencies of Figure 22 apply only when the oscillation frequency is 270 kHz (one clock pulse of 3.7 μs).

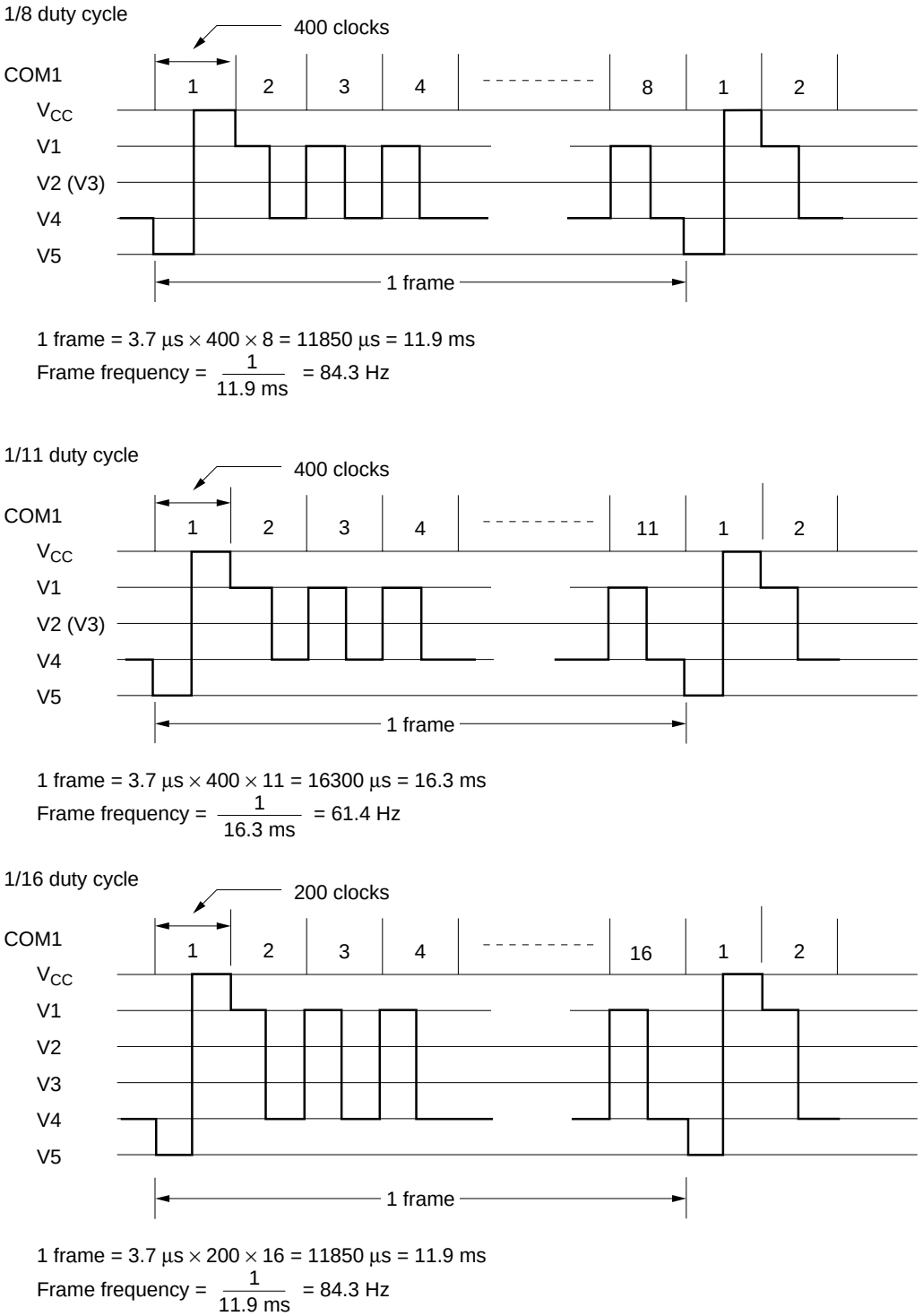


Figure 22 Frame Frequency

Instruction and Display Correspondence

- 8-bit operation, 8-digit × 1-line display with internal reset

  Refer to Table 11 for an example of an 8-digit × 1-line display in 8-bit operation. The HD44780U functions must be set by the function set instruction prior to the display. Since the display data RAM can store data for 80 characters, as explained before, the RAM can be used for displays such as for advertising when combined with the display shift operation.

  Since the display shift operation changes only the display position with DDRAM contents unchanged, the first display data entered into DDRAM can be output when the return home operation is performed.

- 4-bit operation, 8-digit × 1-line display with internal reset

  The program must set all functions prior to the 4-bit operation (Table 12). When the power is turned on, 8-bit operation is automatically selected and the first write is performed as an 8-bit operation. Since DB0 to DB3 are not connected, a rewrite is then required. However, since one operation is completed in two accesses for 4-bit operation, a rewrite is needed to set the functions (see Table 12). Thus, DB4 to DB7 of the function set instruction is written twice.

- 8-bit operation, 8-digit × 2-line display

  For a 2-line display, the cursor automatically moves from the first to the second line after the 40th digit of the first line has been written. Thus, if there are only 8 characters in the first line, the DDRAM address must be again set after the 8th character is completed. (See Table 13.) Note that the display shift operation is performed for the first and second lines. In the example of Table 13, the display shift is performed when the cursor is on the second line. However, if the shift operation is performed when the cursor is on the first line, both the first and second lines move together. If the shift is repeated, the display of the second line will not move to the first line. The same display will only shift within its own line for the number of times the shift is repeated.

Note: When using the internal reset, the electrical characteristics in the Power Supply Conditions Using Internal Reset Circuit table must be satisfied. If not, the HD44780U must be initialized by instructions. See the section, Initializing by Instruction.

Table 11 8-Bit Operation, 8-Digit × 1-Line Display Example with Internal Reset

Step No.	Instruction RS	R/$\overline{W}$	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Display	Operation
1	Power supply on (the HD44780U is initialized by the internal reset circuit)											Initialized. No display.
2	Function set											Sets to 8-bit operation and selects 1-line display and 5 × 8 dot character font. (Number of display lines and character fonts cannot be changed after step #2.)
	0	0	0	0	1	1	0	0	*	*		
3	Display on/off control											Turns on display and cursor. Entire display is in space mode because of initialization.
	0	0	0	0	0	0	1	1	1	0	_	
4	Entry mode set											Sets mode to increment the address by one and to shift the cursor to the right at the time of write to the DD/CGRAM. Display is not shifted.
	0	0	0	0	0	0	0	1	1	0	_	
5	Write data to CGRAM/DDRAM											Writes H. DDRAM has already been selected by initialization when the power was turned on. The cursor is incremented by one and shifted to the right.
	1	0	0	1	0	0	1	0	0	0	H_	
6	Write data to CGRAM/DDRAM											Writes I.
	1	0	0	1	0	0	1	0	0	1	HI_	
7						. . . . .					. . . . .	
8	Write data to CGRAM/DDRAM											Writes I.
	1	0	0	1	0	0	1	0	0	1	HITACHI_	
9	Entry mode set											Sets mode to shift display at the time of write.
	0	0	0	0	0	0	0	1	1	1	HITACHI_	
10	Write data to CGRAM/DDRAM											Writes a space.
	1	0	0	0	1	0	0	0	0	0	ITACHI _	

Table 11 8-Bit Operation, 8-Digit × 1-Line Display Example with Internal Reset (cont)

Step No.	Instruction RS	R/$\overline{W}$	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Display	Operation
11	Write data to CGRAM/DDRAM											
	1	0	0	1	0	0	1	1	0	1	TACHI M_	Writes M.
12						.					.	
						.					.	
						.					.	
						.					.	
						.					.	
13	Write data to CGRAM/DDRAM											
	1	0	0	1	0	0	1	1	1	1	MICROKO_	Writes O.
14	Cursor or display shift											
	0	0	0	0	0	1	0	0	*	*	MICROKO	Shifts only the cursor position to the left.
15	Cursor or display shift											
	0	0	0	0	0	1	0	0	*	*	MICROKO	Shifts only the cursor position to the left.
16	Write data to CGRAM/DDRAM											
	1	0	0	1	0	0	0	0	1	1	ICROCO	Writes C over K. The display moves to the left.
17	Cursor or display shift											
	0	0	0	0	0	1	1	1	*	*	MICROCO	Shifts the display and cursor position to the right.
18	Cursor or display shift											
	0	0	0	0	0	1	0	1	*	*	MICROCO_	Shifts the display and cursor position to the right.
19	Write data to CGRAM/DDRAM											
	1	0	0	1	0	0	1	1	0	1	ICROCOM_	Writes M.
20						.					.	
						.					.	
						.					.	
						.					.	
						.					.	
21	Return home											
	0	0	0	0	0	0	0	0	1	0	HITACHI	Returns both display and cursor to the original position (address 0).

Table 12 4-Bit Operation, 8-Digit × 1-Line Display Example with Internal Reset

Step No.	Instruction RS	R/$\overline{W}$	DB7	DB6	DB5	DB4	Display	Operation
1	Power supply on (the HD44780U is initialized by the internal reset circuit)							Initialized. No display.
2	Function set							Sets to 4-bit operation. In this case, operation is handled as 8 bits by initialization, and only this instruction completes with one write.
	0	0	0	0	1	0		
3	Function set							Sets 4-bit operation and selects 1-line display and 5 × 8 dot character font. 4-bit operation starts from this step and resetting is necessary. (Number of display lines and character fonts cannot be changed after step #3.)
	0	0	0	0	1	0		
	0	0	0	0	*	*		
4	Display on/off control						_	Turns on display and cursor. Entire display is in space mode because of initialization.
	0	0	0	0	0	0		
	0	0	1	1	1	0		
5	Entry mode set						_	Sets mode to increment the address by one and to shift the cursor to the right at the time of write to the DD/CGRAM. Display is not shifted.
	0	0	0	0	0	0		
	0	0	0	1	1	0		
6	Write data to CGRAM/DDRAM						H_	Writes H. The cursor is incremented by one and shifts to the right.
	1	0	0	1	0	0		
	1	0	1	0	0	0		

Note: The control is the same as for 8-bit operation beyond step #6.

Table 13 8-Bit Operation, 8-Digit × 2-Line Display Example with Internal Reset

Step No.	Instruction RS	R/$\overline{W}$	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Display	Operation
1	Power supply on (the HD44780U is initialized by the internal reset circuit)											Initialized. No display.
2	Function set											Sets to 8-bit operation and selects 2-line display and 5 × 8 dot character font.
	0	0	0	0	1	1	1	0	*	*		
3	Display on/off control											Turns on display and cursor. All display is in space mode because of initialization.
	0	0	0	0	0	0	1	1	1	0	_	
4	Entry mode set											Sets mode to increment the address by one and to shift the cursor to the right at the time of write to the DD/CGRAM. Display is not shifted.
	0	0	0	0	0	0	0	1	1	0	_	
5	Write data to CGRAM/DDRAM											Writes H. DDRAM has already been selected by initialization when the power was turned on. The cursor is incremented by one and shifted to the right.
	1	0	0	1	0	0	1	0	0	0	H_	
6	.				.						.	
	.				.						.	
	.				.						.	
	.				.						.	
	.				.						.	
7	Write data to CGRAM/DDRAM											Writes I.
	1	0	0	1	0	0	1	0	0	1	HITACHI_	
8	Set DDRAM address											Sets DDRAM address so that the cursor is positioned at the head of the second line.
	0	0	1	1	0	0	0	0	0	0	HITACHI _	

Table 13 8-Bit Operation, 8-Digit × 2-Line Display Example with Internal Reset (cont)

Step No.	Instruction RS	R/$\overline{W}$	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Display	Operation
9	Write data to CGRAM/DDRAM 1	0	0	1	0	0	1	1	0	1	HITACHI M_	Writes M.
10	.	.	.	.	.	.	.	.	.	.	.	
11	Write data to CGRAM/DDRAM 1	0	0	1	0	0	1	1	1	1	HITACHI MICROCO_	Writes O.
12	Entry mode set 0	0	0	0	0	0	0	1	1	1	HITACHI MICROCO_	Sets mode to shift display at the time of write.
13	Write data to CGRAM/DDRAM 1	0	0	1	0	0	1	1	0	1	ITACHI ICROCOM_	Writes M. Display is shifted to the left. The first and second lines both shift at the same time.
14	.	.	.	.	.	.	.	.	.	.	.	
15	Return home 0	0	0	0	0	0	0	0	1	0	HITACHI MICROCOM	Returns both display and cursor to the original position (address 0).

Initializing by Instruction

If the power supply conditions for correctly operating the internal reset circuit are not met, initialization by instructions becomes necessary.

Refer to Figures 23 and 24 for the procedures on 8-bit and 4-bit initializations, respectively.

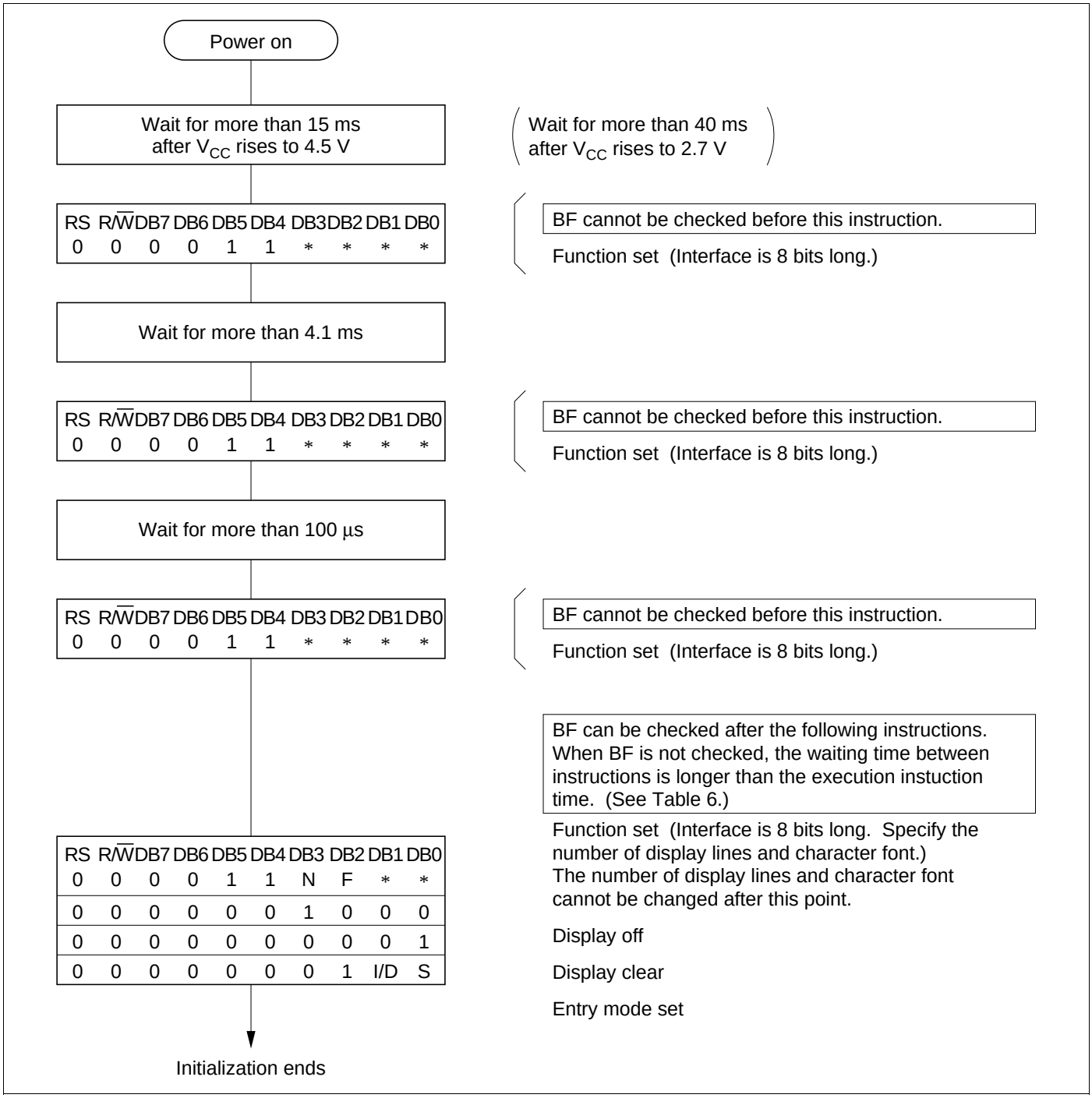


Figure 23 8-Bit Interface

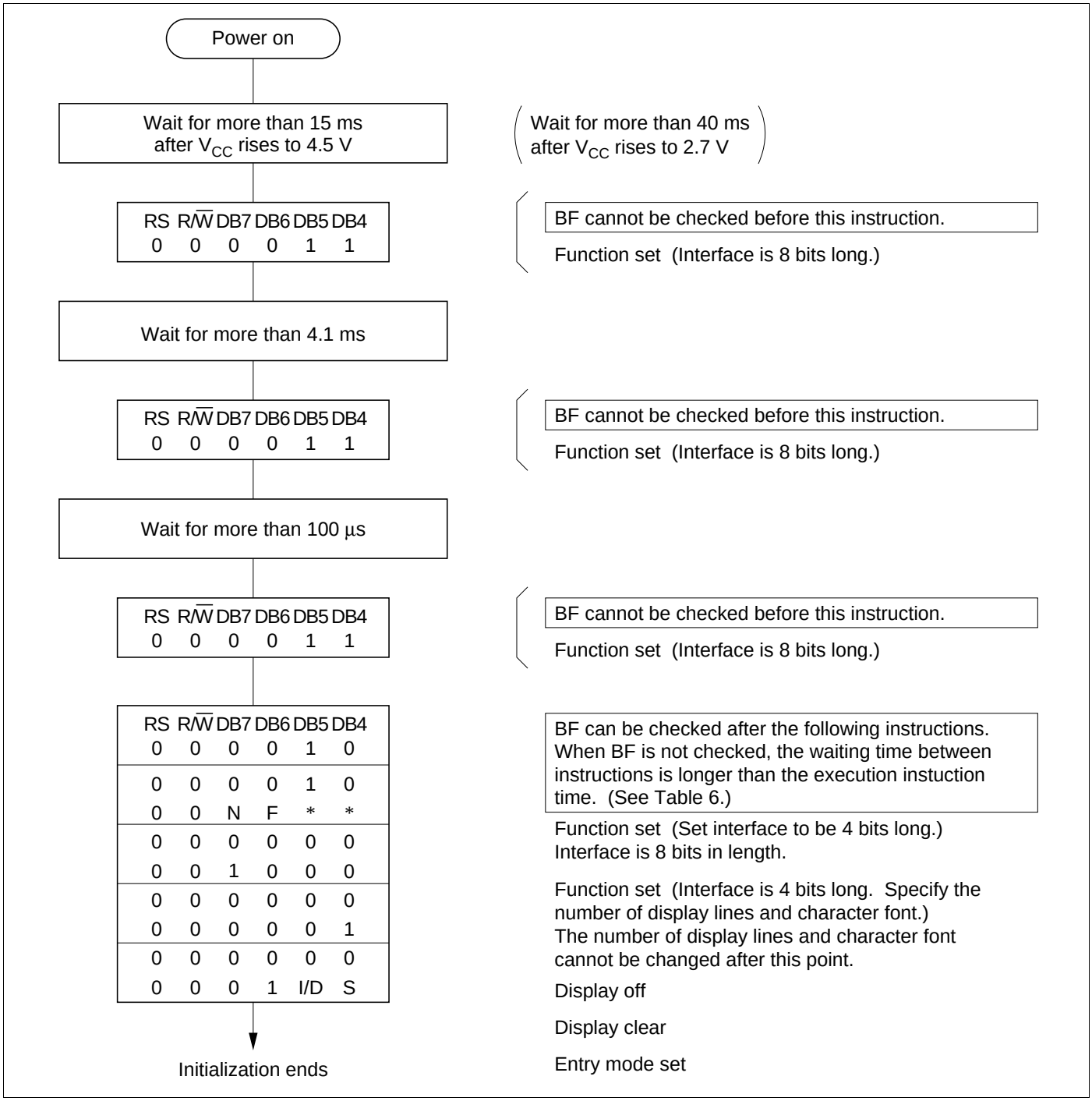


Figure 24 4-Bit Interface

Absolute Maximum Ratings*

Item	Symbol	Value	Unit	Notes
Power supply voltage (1)	V_{CC}–GND	–0.3 to +7.0	V	1
Power supply voltage (2)	V_{CC}–V5	–0.3 to +13.0	V	1, 2
Input voltage	Vt	–0.3 to V_{CC} +0.3	V	1
Operating temperature	T_{opr}	–30 to +75	°C	
Storage temperature	T_{stg}	–55 to +125	°C	4

Note: * If the LSI is used above these absolute maximum ratings, it may become permanently damaged. Using the LSI within the following electrical characteristic limits is strongly recommended for normal operation. If these electrical characteristic conditions are also exceeded, the LSI will malfunction and cause poor reliability.

DC Characteristics (V_{CC} = 2.7 to 4.5 V, T_a = –30 to +75°C*[3])

Item	Symbol	Min	Typ	Max	Unit	Test Condition	Notes*
Input high voltage (1) (except OSC1)	VIH1	$0.7V_{CC}$	—	V_{CC}	V		6
Input low voltage (1) (except OSC1)	VIL1	–0.3	—	0.55	V		6
Input high voltage (2) (OSC1)	VIH2	$0.7V_{CC}$	—	V_{CC}	V		15
Input low voltage (2) (OSC1)	VIL2	—	—	$0.2V_{CC}$	V		15
Output high voltage (1) (DB0–DB7)	VOH1	$0.75V_{CC}$	—	—	V	$–I_{OH}$ = 0.1 mA	7
Output low voltage (1) (DB0–DB7)	VOL1	—	—	$0.2V_{CC}$	V	I_{OL} = 0.1 mA	7
Output high voltage (2) (except DB0–DB7)	VOH2	$0.8V_{CC}$	—	—	V	$–I_{OH}$ = 0.04 mA	8
Output low voltage (2) (except DB0–DB7)	VOL2	—	—	$0.2V_{CC}$	V	I_{OL} = 0.04 mA	8
Driver on resistance (COM)	R_{COM}	—	2	20	kΩ	±Id = 0.05 mA, VLCD = 4 V	13
Driver on resistance (SEG)	R_{SEG}	—	2	30	kΩ	±Id = 0.05 mA, VLCD = 4 V	13
Input leakage current	I_{LI}	–1	—	1	µA	VIN = 0 to V_{CC}	9
Pull-up MOS current (DB0–DB7, RS, R/$\overline{W}$)	$–I_p$	10	50	120	µA	V_{CC} = 3 V	
Power supply current	I_{CC}	—	150	300	µA	R_f oscillation, external clock V_{CC} = 3 V, f_{OSC} = 270 kHz	10, 14
LCD voltage	VLCD1	3.0	—	11.0	V	V_{CC}–V5, 1/5 bias	16
	VLCD2	3.0	—	11.0	V	V_{CC}–V5, 1/4 bias	16

Note: * Refer to the Electrical Characteristics Notes section following these tables.

AC Characteristics (V_{CC} = 2.7 to 4.5 V, T_a = –30 to +75°C*[3])

Clock Characteristics

Item		Symbol	Min	Typ	Max	Unit	Test Condition	Note*
External clock operation	External clock frequency	f_{cp}	125	250	350	kHz		11
	External clock duty	Duty	45	50	55	%		
	External clock rise time	t_{rcp}	—	—	0.2	µs		
	External clock fall time	t_{fcp}	—	—	0.2	µs		
R_f oscillation	Clock oscillation frequency	f_{OSC}	190	270	350	kHz	R_f = 75 kΩ, V_{CC} = 3 V	12

Note: * Refer to the Electrical Characteristics Notes section following these tables.

Bus Timing Characteristics

Write Operation

Item	Symbol	Min	Typ	Max	Unit	Test Condition
Enable cycle time	t_{cycE}	1000	—	—	ns	Figure 25
Enable pulse width (high level)	PW_{EH}	450	—	—		
Enable rise/fall time	t_{Er}, t_{Ef}	—	—	25		
Address set-up time (RS, R/$\overline{W}$ to E)	t_{AS}	60	—	—		
Address hold time	t_{AH}	20	—	—		
Data set-up time	t_{DSW}	195	—	—		
Data hold time	t_H	10	—	—		

Read Operation

Item	Symbol	Min	Typ	Max	Unit	Test Condition
Enable cycle time	t_{cycE}	1000	—	—	ns	Figure 26
Enable pulse width (high level)	PW_{EH}	450	—	—		
Enable rise/fall time	t_{Er}, t_{Ef}	—	—	25		
Address set-up time (RS, R/$\overline{W}$ to E)	t_{AS}	60	—	—		
Address hold time	t_{AH}	20	—	—		
Data delay time	t_{DDR}	—	—	360		
Data hold time	t_{DHR}	5	—	—		

Interface Timing Characteristics with External Driver

Item		Symbol	Min	Typ	Max	Unit	Test Condition
Clock pulse width	High level	t_{CWH}	800	—	—	ns	Figure 27
	Low level	t_{CWL}	800	—	—		
Clock set-up time		t_{CSU}	500	—	—		
Data set-up time		t_{SU}	300	—	—		
Data hold time		t_{DH}	300	—	—		
M delay time		t_{DM}	–1000	—	1000		
Clock rise/fall time		t_{ct}	—	—	200		

Power Supply Conditions Using Internal Reset Circuit

Item	Symbol	Min	Typ	Max	Unit	Test Condition
Power supply rise time	t_{rCC}	0.1	—	10	ms	Figure 28
Power supply off time	t_{OFF}	1	—	—		

DC Characteristics (V_{CC} = 4.5 to 5.5 V, T_a = –30 to +75°C*[3])

Item	Symbol	Min	Typ	Max	Unit	Test Condition	Notes*
Input high voltage (1) (except OSC1)	VIH1	2.2	—	V_{CC}	V		6
Input low voltage (1) (except OSC1)	VIL1	–0.3	—	0.6	V		6
Input high voltage (2) (OSC1)	VIH2	V_{CC}–1.0	—	V_{CC}	V		15
Input low voltage (2) (OSC1)	VIL2	—	—	1.0	V		15
Output high voltage (1) (DB0–DB7)	VOH1	2.4	—	—	V	$–I_{OH}$ = 0.205 mA	7
Output low voltage (1) (DB0–DB7)	VOL1	—	—	0.4	V	I_{OL} = 1.2 mA	7
Output high voltage (2) (except DB0–DB7)	VOH2	0.9 V_{CC}	—	—	V	$–I_{OH}$ = 0.04 mA	8
Output low voltage (2) (except DB0–DB7)	VOL2	—	—	0.1 V_{CC}	V	I_{OL} = 0.04 mA	8
Driver on resistance (COM)	RCOM	—	2	20	kΩ	±Id = 0.05 mA, VLCD = 4 V	13
Driver on resistance (SEG)	RSEG	—	2	30	kΩ	±Id = 0.05 mA, VLCD = 4 V	13
Input leakage current	I_{LI}	–1	—	1	µA	VIN = 0 to V_{CC}	9
Pull-up MOS current (DB0–DB7, RS, R/$\overline{W}$)	$–I_p$	50	125	250	µA	V_{CC} = 5 V	
Power supply current	I_{CC}	—	350	600	µA	R_f oscillation, external clock V_{CC} = 5 V, f_{OSC} = 270 kHz	10, 14
LCD voltage	VLCD1	3.0	—	11.0	V	V_{CC}–V5, 1/5 bias	16
	VLCD2	3.0	—	11.0	V	V_{CC}–V5, 1/4 bias	16

Note: * Refer to the Electrical Characteristics Notes section following these tables.

AC Characteristics (V_{CC} = 4.5 to 5.5 V, T_a = –30 to +75°C*[3])

Clock Characteristics

Item		Symbol	Min	Typ	Max	Unit	Test Condition	Notes*
External clock operation	External clock frequency	f_{cp}	125	250	350	kHz		11
	External clock duty	Duty	45	50	55	%		11
	External clock rise time	t_{rcp}	—	—	0.2	µs		11
	External clock fall time	t_{fcp}	—	—	0.2	µs		11
R_f oscillation	Clock oscillation frequency	f_{OSC}	190	270	350	kHz	R_f = 91 kΩ V_{CC} = 5.0 V	12

Note: * Refer to the Electrical Characteristics Notes section following these tables.

Bus Timing Characteristics

Write Operation

Item	Symbol	Min	Typ	Max	Unit	Test Condition
Enable cycle time	t_{cycE}	500	—	—	ns	Figure 25
Enable pulse width (high level)	PW_{EH}	230	—	—		
Enable rise/fall time	t_{Er}, t_{Ef}	—	—	20		
Address set-up time (RS, R/$\overline{W}$ to E)	t_{AS}	40	—	—		
Address hold time	t_{AH}	10	—	—		
Data set-up time	t_{DSW}	80	—	—		
Data hold time	t_H	10	—	—		

Read Operation

Item	Symbol	Min	Typ	Max	Unit	Test Condition
Enable cycle time	t_{cycE}	500	—	—	ns	Figure 26
Enable pulse width (high level)	PW_{EH}	230	—	—		
Enable rise/fall time	t_{Er}, t_{Ef}	—	—	20		
Address set-up time (RS, R/$\overline{W}$ to E)	t_{AS}	40	—	—		
Address hold time	t_{AH}	10	—	—		
Data delay time	t_{DDR}	—	—	160		
Data hold time	t_{DHR}	5	—	—		

Interface Timing Characteristics with External Driver

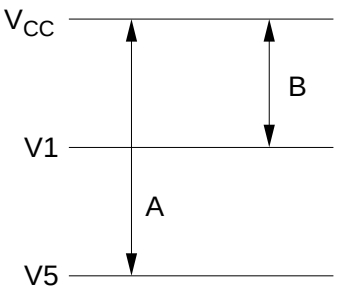
Item		Symbol	Min	Typ	Max	Unit	Test Condition
Clock pulse width	High level	t_{CWH}	800	—	—	ns	Figure 27
	Low level	t_{CWL}	800	—	—		
Clock set-up time		t_{CSU}	500	—	—		
Data set-up time		t_{SU}	300	—	—		
Data hold time		t_{DH}	300	—	—		
M delay time		t_{DM}	–1000	—	1000		
Clock rise/fall time		t_{ct}	—	—	100		

Power Supply Conditions Using Internal Reset Circuit

Item	Symbol	Min	Typ	Max	Unit	Test Condition
Power supply rise time	t_{rCC}	0.1	—	10	ms	Figure 28
Power supply off time	t_{OFF}	1	—	—		

Electrical Characteristics Notes

1. All voltage values are referred to GND = 0 V.

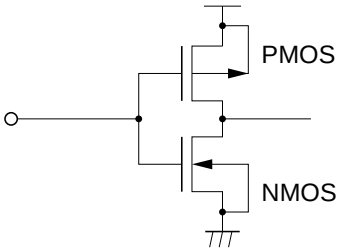


$A = V_{CC} - V5$
$B = V_{CC} - V1$
$A \geq 1.5$ V
$B \leq 0.25 \times A$

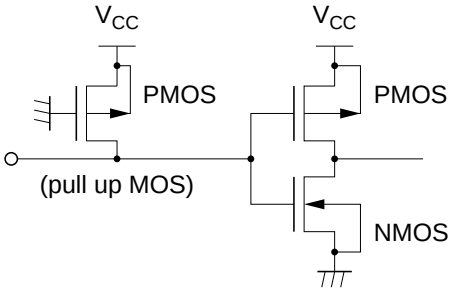
The conditions of V1 and V5 voltages are for proper operation of the LSI and not for the LCD output level. The LCD drive voltage condition for the LCD output level is specified as LCD voltage VLCD.

2. $V_{CC} \geq V1 \geq V2 \geq V3 \geq V4 \geq V5$ must be maintained.
3. For die products, specified at 75°C.
4. For die products, specified by the die shipment specification.
5. The following four circuits are I/O pin configurations except for liquid crystal display output.

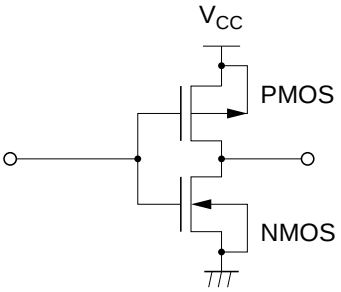
Input pin
Pin: E (MOS without pull-up)



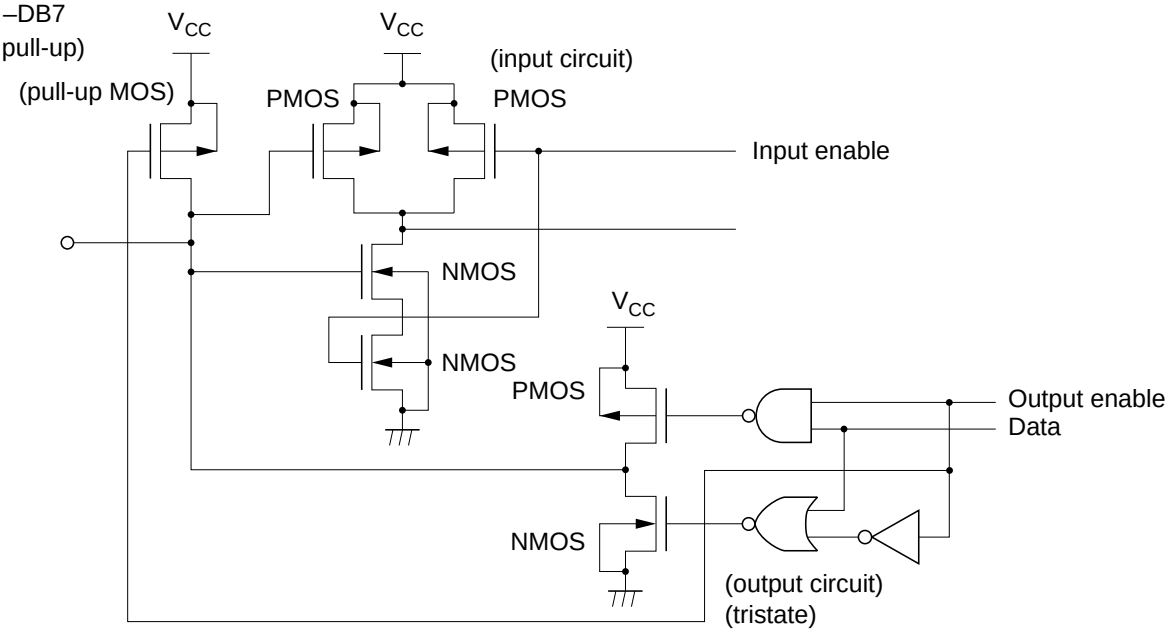
Pins: RS, R/$\overline{W}$ (MOS with pull-up)



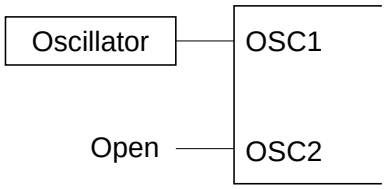
Output pin
Pins: CL1, CL2, M, D

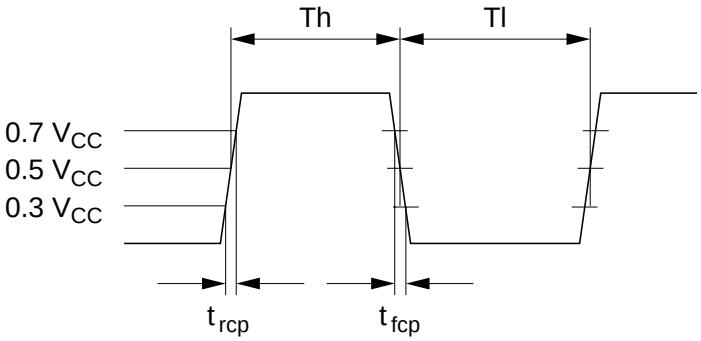


I/O Pin
Pins: DB0 –DB7
(MOS with pull-up)



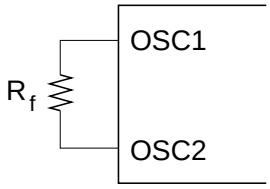
6. Applies to input pins and I/O pins, excluding the OSC1 pin.
7. Applies to I/O pins.
8. Applies to output pins.
9. Current flowing through pull–up MOSs, excluding output drive MOSs.
10. Input/output current is excluded. When input is at an intermediate level with CMOS, the excessive current flows through the input circuit to the power supply. To avoid this from happening, the input level must be fixed high or low.
11. Applies only to external clock operation.



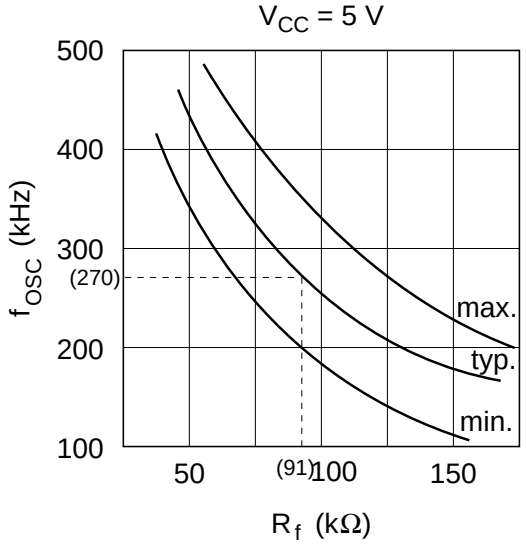


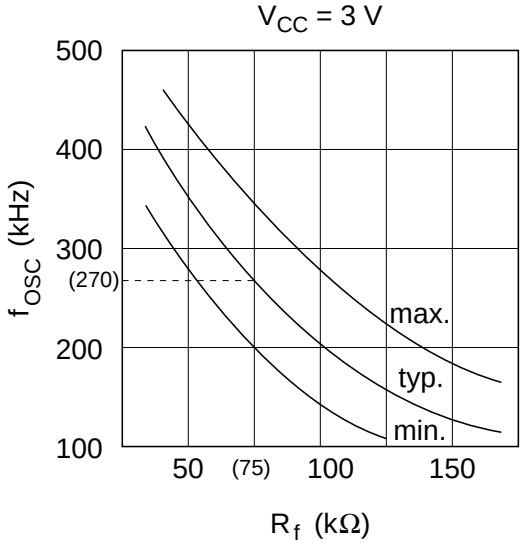
$$\text{Duty} = \frac{Th}{Th + Tl} \times 100\%$$

12. Applies only to the internal oscillator operation using oscillation resistor R_f.



R_f: 75 kΩ ± 2% (when V_{CC} = 3 V)
R_f: 91 kΩ ± 2% (when V_{CC} = 5 V)
Since the oscillation frequency varies depending on the OSC1 and OSC2 pin capacitance, the wiring length to these pins should be minimized.

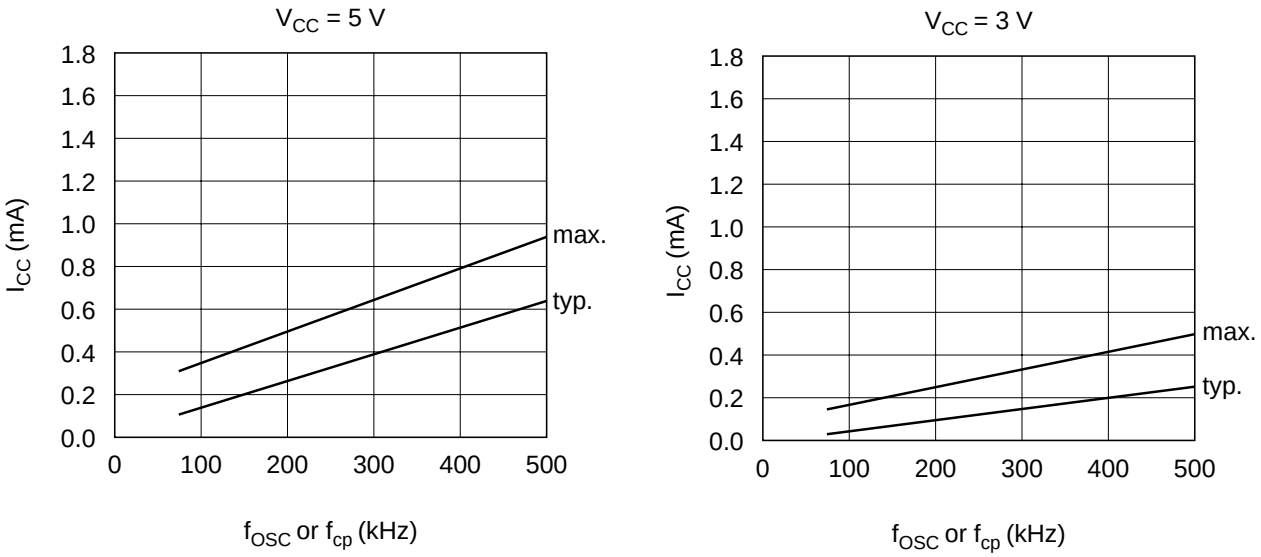




13. RCOM is the resistance between the power supply pins (V_{CC}, V1, V4, V5) and each common signal pin (COM1 to COM16).

    RSEG is the resistance between the power supply pins (V_{CC}, V2, V3, V5) and each segment signal pin (SEG1 to SEG40).

14. The following graphs show the relationship between operation frequency and current consumption.

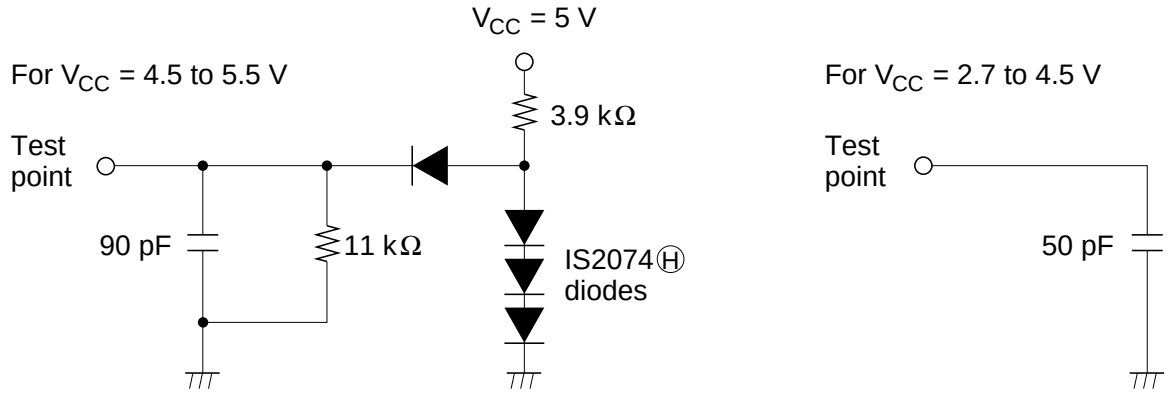


15. Applies to the OSC1 pin.

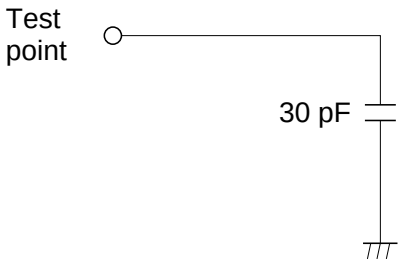
16. Each COM and SEG output voltage is within ±0.15 V of the LCD voltage (V_{CC}, V1, V2, V3, V4, V5) when there is no load.

Load Circuits

Data Bus DB0 to DB7



External Driver Control Signals: CL1, CL2, D, M



Timing Characteristics

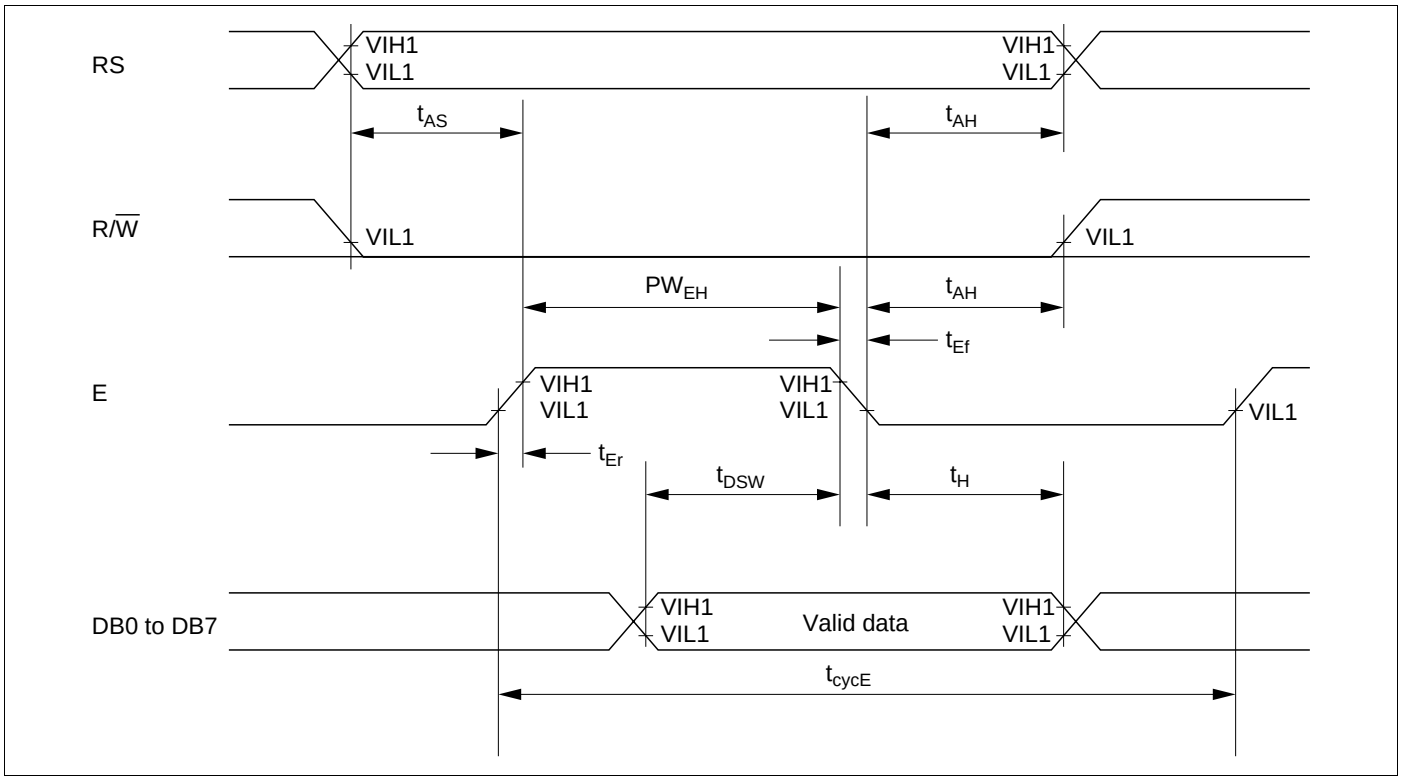
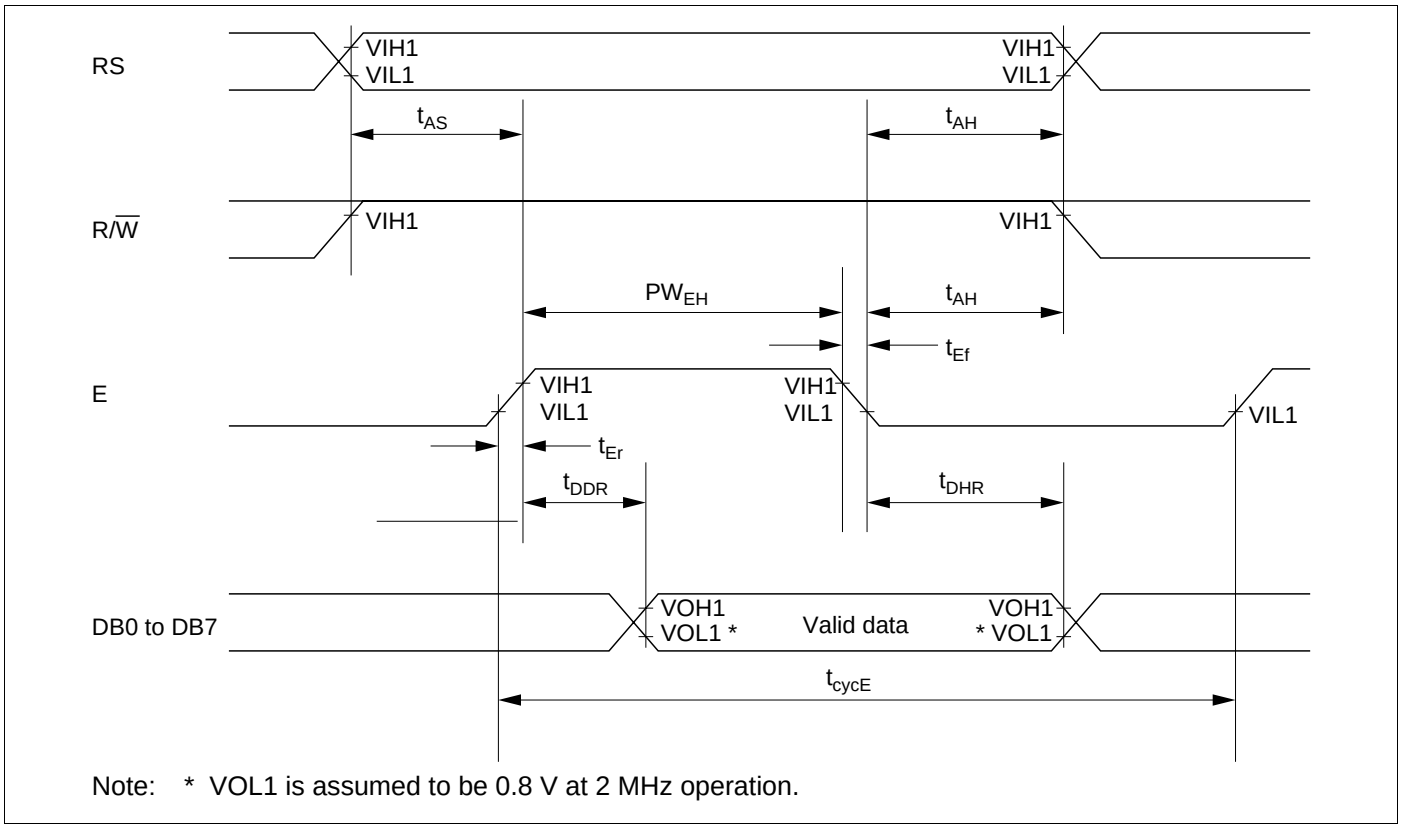


Figure 25 Write Operation



Note: * VOL1 is assumed to be 0.8 V at 2 MHz operation.

Figure 26 Read Operation

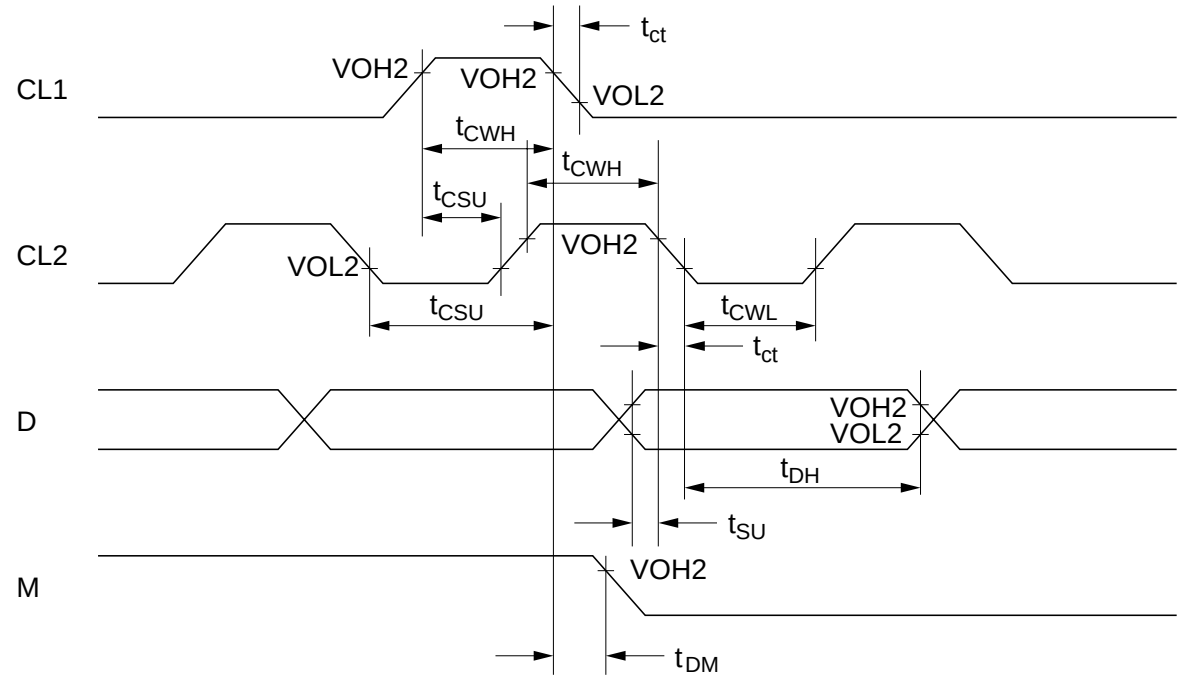
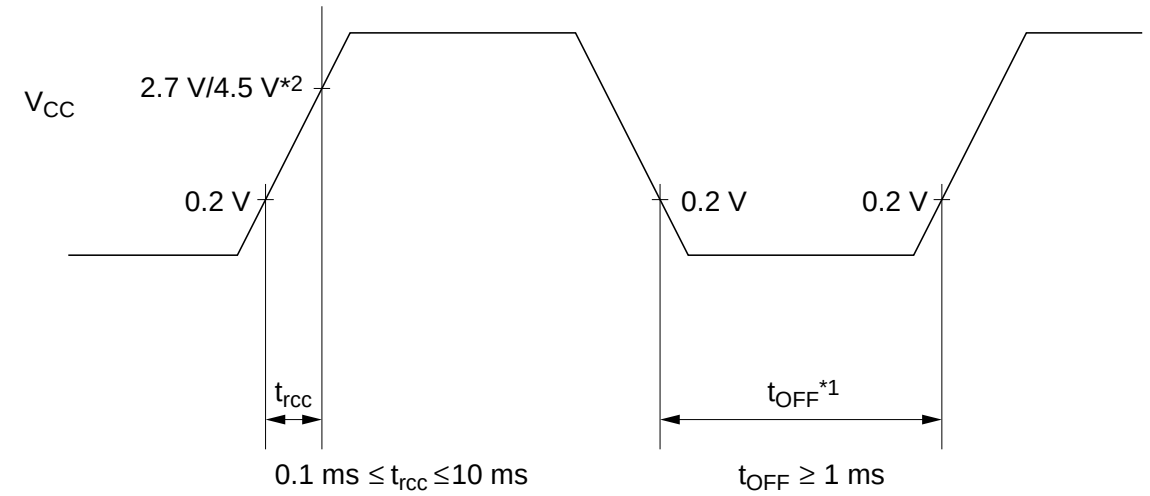


Figure 27 Interface Timing with External Driver



Notes: 1. t_{OFF} compensates for the power oscillation period caused by momentary power supply oscillations.
2. Specified at 4.5 V for 5-V operation, and at 2.7 V for 3-V operation.
3. For if 4.5 V is not reached during 5-V operation, the internal reset circuit will not operate normally.
In this case, the LSI must be initialized by software. (Refer to the Initializing by Instruction section.)

Figure 28 Internal Power Supply Reset

Cautions

1. Hitachi neither warrants nor grants licenses of any rights of Hitachi's or any third party's patent, copyright, trademark, or other intellectual property rights for information contained in this document. Hitachi bears no responsibility for problems that may arise with third party's rights, including intellectual property rights, in connection with use of the information contained in this document.
2. Products and product specifications may be subject to change without notice. Confirm that you have received the latest product standards or specifications before final design, purchase or use.
3. Hitachi makes every attempt to ensure that its products are of high quality and reliability. However, contact Hitachi's sales office before using the product in an application that demands especially high quality and reliability or where its failure or malfunction may directly threaten human life or cause risk of bodily injury, such as aerospace, aeronautics, nuclear power, combustion control, transportation, traffic, safety equipment or medical equipment for life support.
4. Design your application so that the product is used within the ranges guaranteed by Hitachi particularly for maximum rating, operating supply voltage range, heat radiation characteristics, installation conditions and other characteristics. Hitachi bears no responsibility for failure or damage when used beyond the guaranteed ranges. Even within the guaranteed ranges, consider normally foreseeable failure rates or failure modes in semiconductor devices and employ systemic measures such as fail-safes, so that the equipment incorporating Hitachi product does not cause bodily injury, fire or other consequential damage due to operation of the Hitachi product.
5. This product is not designed to be radiation resistant.
6. No one is permitted to reproduce or duplicate, in any form, the whole or part of this document without written approval from Hitachi.
7. Contact Hitachi's sales office for any questions regarding this document or Hitachi semiconductor products.

HITACHI

Hitachi, Ltd.
Semiconductor & Integrated Circuits.
Nippon Bldg., 2-6-2, Ohte-machi, Chiyoda-ku, Tokyo 100-0004, Japan
Tel: Tokyo (03) 3270-2111 Fax: (03) 3270-5109

URL
NorthAmerica : http:semiconductor.hitachi.com/
Europe : http://www.hitachi-eu.com/hel/ecg
Asia (Singapore) : http://www.has.hitachi.com.sg/grp3/sicd/index.htm
Asia (Taiwan) : http://www.hitachi.com.tw/E/Product/SICD_Frame.htm
Asia (HongKong) : http://www.hitachi.com.hk/eng/bo/grp3/index.htm
Japan : http://www.hitachi.co.jp/Sicd/indx.htm

For further information write to:

Hitachi Semiconductor
(America) Inc.
179 East Tasman Drive,
San Jose,CA 95134
Tel: <1> (408) 433-1990
Fax: <1>(408) 433-0223

Hitachi Europe GmbH
Electronic components Group
Dornacher Straße 3
D-85622 Feldkirchen, Munich
Germany
Tel: <49> (89) 9 9180-0
Fax: <49> (89) 9 29 30 00

Hitachi Europe Ltd.
Electronic Components Group.
Whitebrook Park
Lower Cookham Road
Maidenhead
Berkshire SL6 8YA, United Kingdom
Tel: <44> (1628) 585000
Fax: <44> (1628) 778322

Hitachi Asia Pte. Ltd.
16 Collyer Quay #20-00
Hitachi Tower
Singapore 049318
Tel: 535-2100
Fax: 535-1533

Hitachi Asia Ltd.
Taipei Branch Office
3F, Hung Kuo Building. No.167,
Tun-Hwa North Road, Taipei (105)
Tel: <886> (2) 2718-3666
Fax: <886> (2) 2718-8180

Hitachi Asia (Hong Kong) Ltd.
Group III (Electronic Components)
7/F., North Tower, World Finance Centre,
Harbour City, Canton Road, Tsim Sha Tsui,
Kowloon, Hong Kong
Tel: <852> (2) 735 9218
Fax: <852> (2) 730 0281
Telex: 40815 HITEC HX