AN1447
APPLICATION NOTE

SOFTWARE DRIVER FOR 4-MULTIPLEXED LCD WITH A STANDARD ST62

by Microcontroller Division Applications

DESCRIPTION

This note describes a technique for driving a 4-multiplexed Liquid Crystal Display (LCD) with a standard ST62 microcontroller (MCU), without any dedicated LCD driver peripheral. This technique offers a display capability for applications which require a small display at a low cost together with the versatile capabilities of the standard ST62xx MCU.

Higher display requirements are easily handled by dedicated members of the ST62 MCU family, for example the ST6240. Solutions on how to use a standard ST6 to drive an LCD with a multiplexing ratio of 2 (duplex) can be found in Application Note AN594.

The first section of this note describes the typical waveforms required to drive an LCD, first without multiplexing (“direct” drive), then with a multiplexing rate of 4. The second section explains how to use a software library written in assembly language (MAST6 syntax) implementing a solution based on a standard ST62 MCU driving directly the LCD.

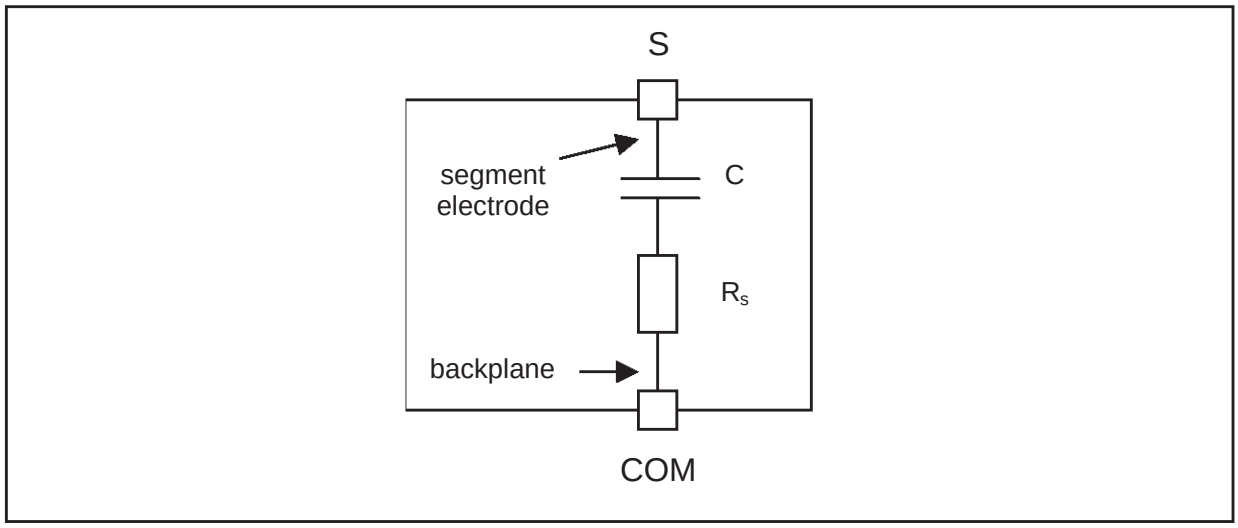
The program size and the CPU time occupation due to the LCD drive are minimized. Consequently many additional tasks can be added to the MCU program. Only few cheap additional components are required.

1 LCD DRIVING PRINCIPLES

1.1 LCD REQUIREMENTS

An LCD segment can either be transparent (“off”) or opaque (“on”), depending on the voltage applied to it. On Figure 1, this voltage is the difference between *COM* and *S* voltages. On most LCDs (reflective positive displays) an opaque segment is seen dark and a transparent segment is seen clear (same colour as the background).

Figure 1. Equivalent Electrical Schematic of an LCD Segment



If no voltage is applied to it, a segment is transparent. To make it opaque, the LCD driver must apply an AC voltage which Root Mean Square (RMS) value is above a certain threshold. This voltage threshold depends on the LCD characteristics.

Segment voltage must also comply with the following conditions:

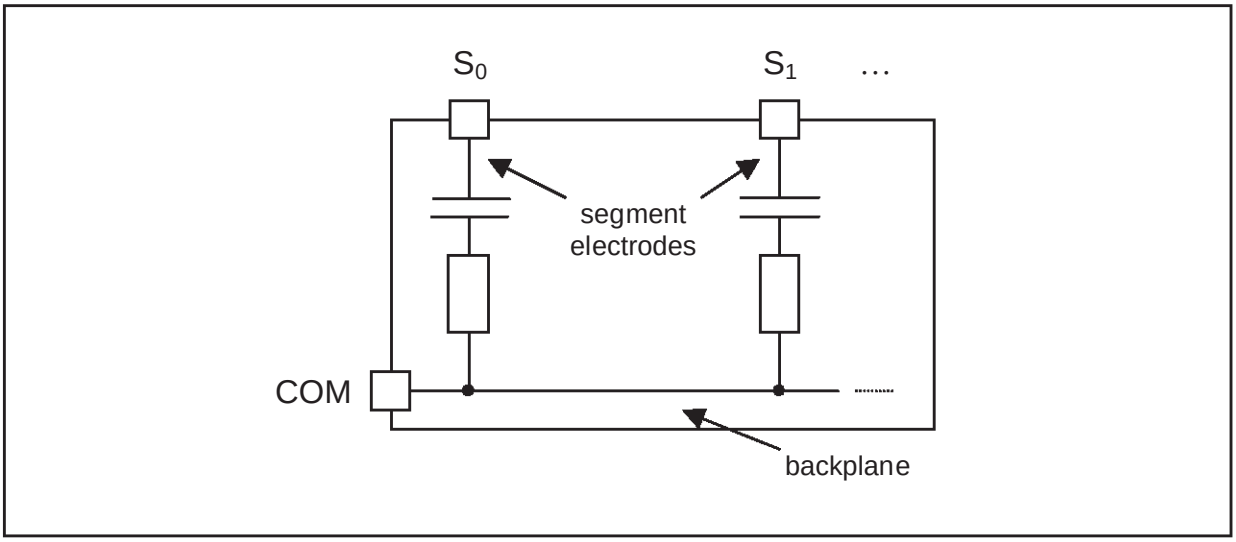
– Its absolute DC (mean) value must be very low (under 100 mV typically). Otherwise, the life time of the LCD can be shortened.

– Its frequency must be in the range 30 – 2000 Hz typically. If too low, the display flickers. If too high, driving generates more power consumption.

1.2 DIRECT LCD DRIVE

Each LCD segment is located between a segment electrode and a backplane common to all the segments (see Figure 2). Therefore, a display using *N* segments contains (*N*+1) external connections: *N* “segment electrode” pins (S_0, S_1,…) and 1 “common” pin (COM).

Figure 2. Connections inside a direct drive LCD

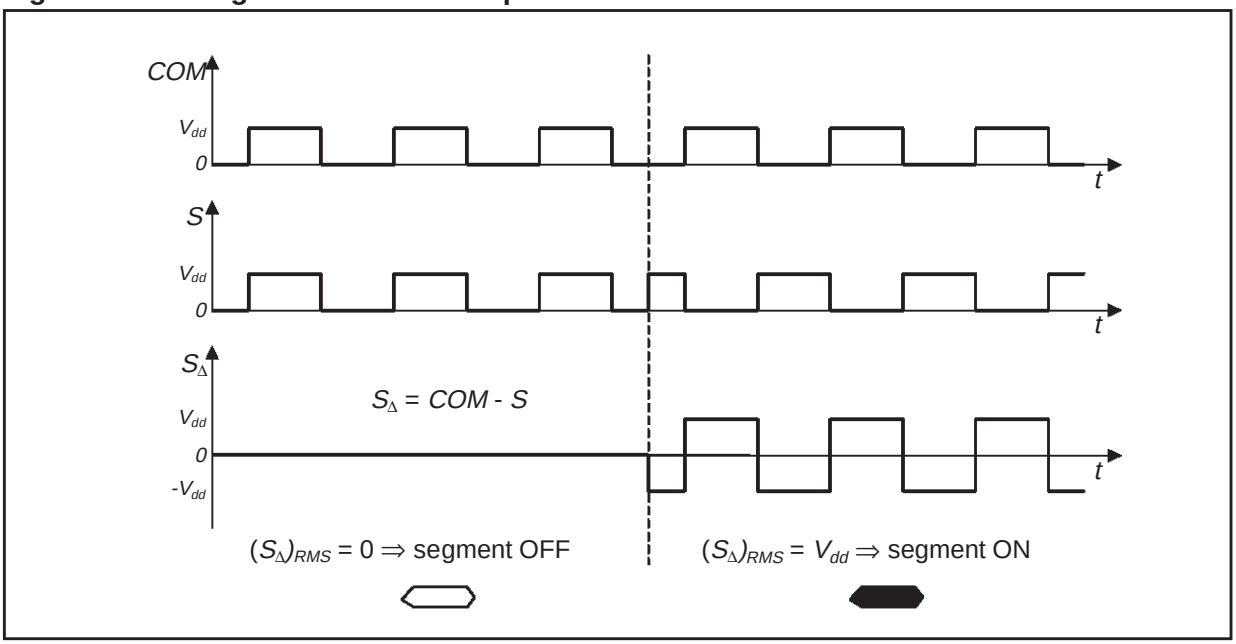


All these pins are connected to MCU I/O pins operating in output mode, either at logic level 0 or at logic level 1.

The backplane is driven with a signal *COM* controlled between 0 and V_{DD} with a duty cycle of 50%.

When selecting a segment ON, a signal with opposite polarity to *COM* is sent to the corresponding segment electrode pin. When the non-inverted signal *COM* is sent to the segment electrode pin, the segment is OFF.

Figure 3. LCD signals for non-multiplexed drive

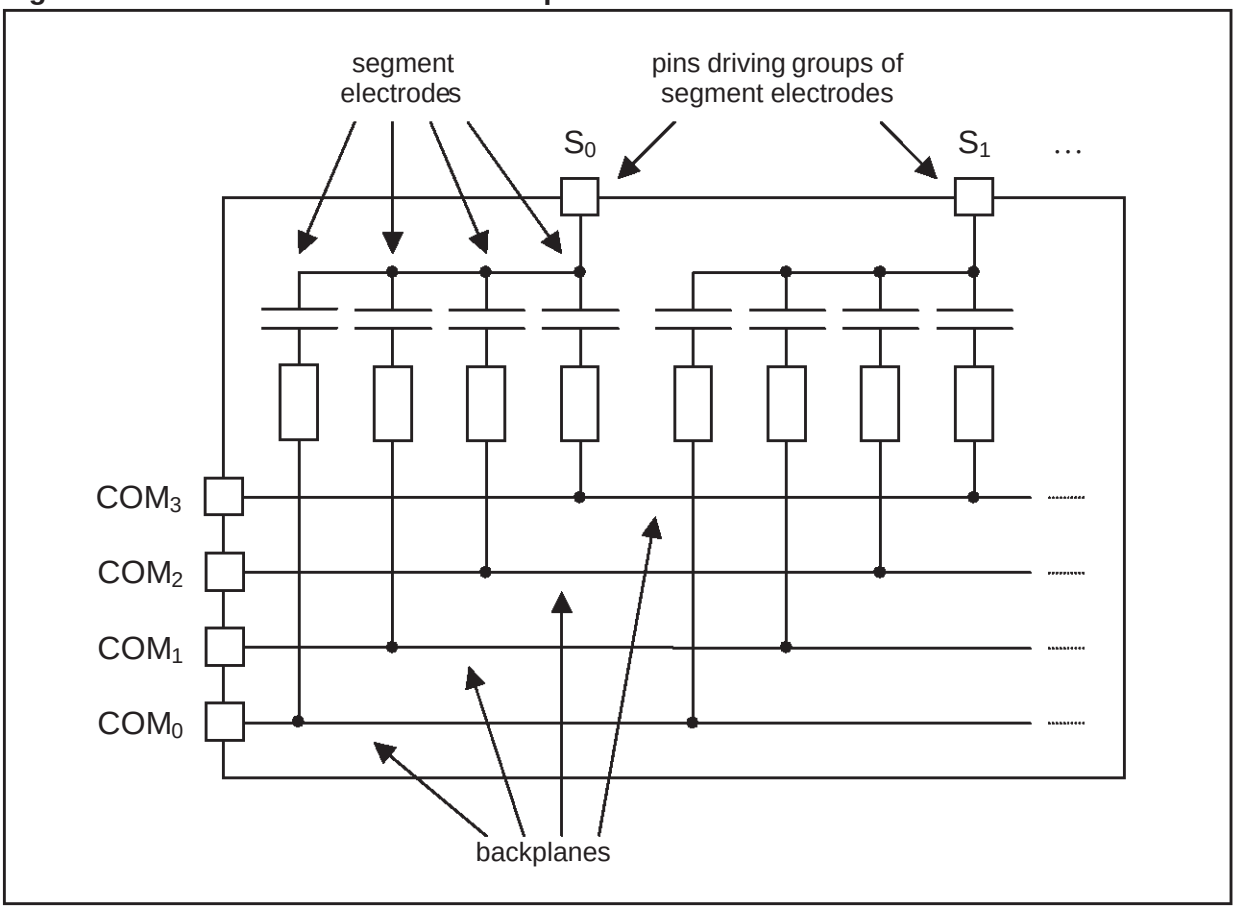


Note: on Figure 3, *S* signal is the “segment electrode voltage” and S_Δ the “segment voltage”.

1.3 4-MULTIPLEXED LCD DRIVE

For 4-multiplexed drive, four backplanes are used instead of one. The LCD segments are equally distributed between the four backplanes. They form groups of 4 segments, where each segment is allocated to a different backplane. All the segment electrodes (or frontplane electrodes) belonging to the same group are connected to a single external pin. Thus, a display using *N* segments contains (*N*/4+4) external connections: *N/4* pins driving groups of segment electrodes (S_0, S_1,...) and 4 "common" pins (COM_0, COM_1, COM_2 and COM_3). On the rest of this document, the pins driving groups of segment electrodes are called "frontplane pins".

Figure 4. Connections inside a 4-multiplexed LCD



Three different voltage levels have to be generated on the backplanes: 0, $V_{DD}/2$ and V_{DD}. The frontplane voltage levels are 0 and V_{DD} only. Figure 5 shows typical backplane, frontplane and segment voltage waveforms.

Each period is divided into 8 phases φ_0 to φ_7. Like in direct drive, *COM* waveforms are applied continuously, whereas *S* waveforms depend on the required display. The logic level applied on S during phase φ_4 is the negation of the one applied during phase φ_0, and so on for φ_5 and

φ_1, φ_6 and φ_5, φ_7 and φ_3. Changing the levels applied during phases φ_0 and φ_4 does not change the DC value nor the RMS of $S_{\Delta 1}$, $S_{\Delta 2}$ and $S_{\Delta 3}$ voltages. It does not change the DC value of $S_{\Delta 0}$ voltage, but affects its RMS, as explained in Table 1.

Table 1. How to switch one segment on and off in 4-multiplexed drive

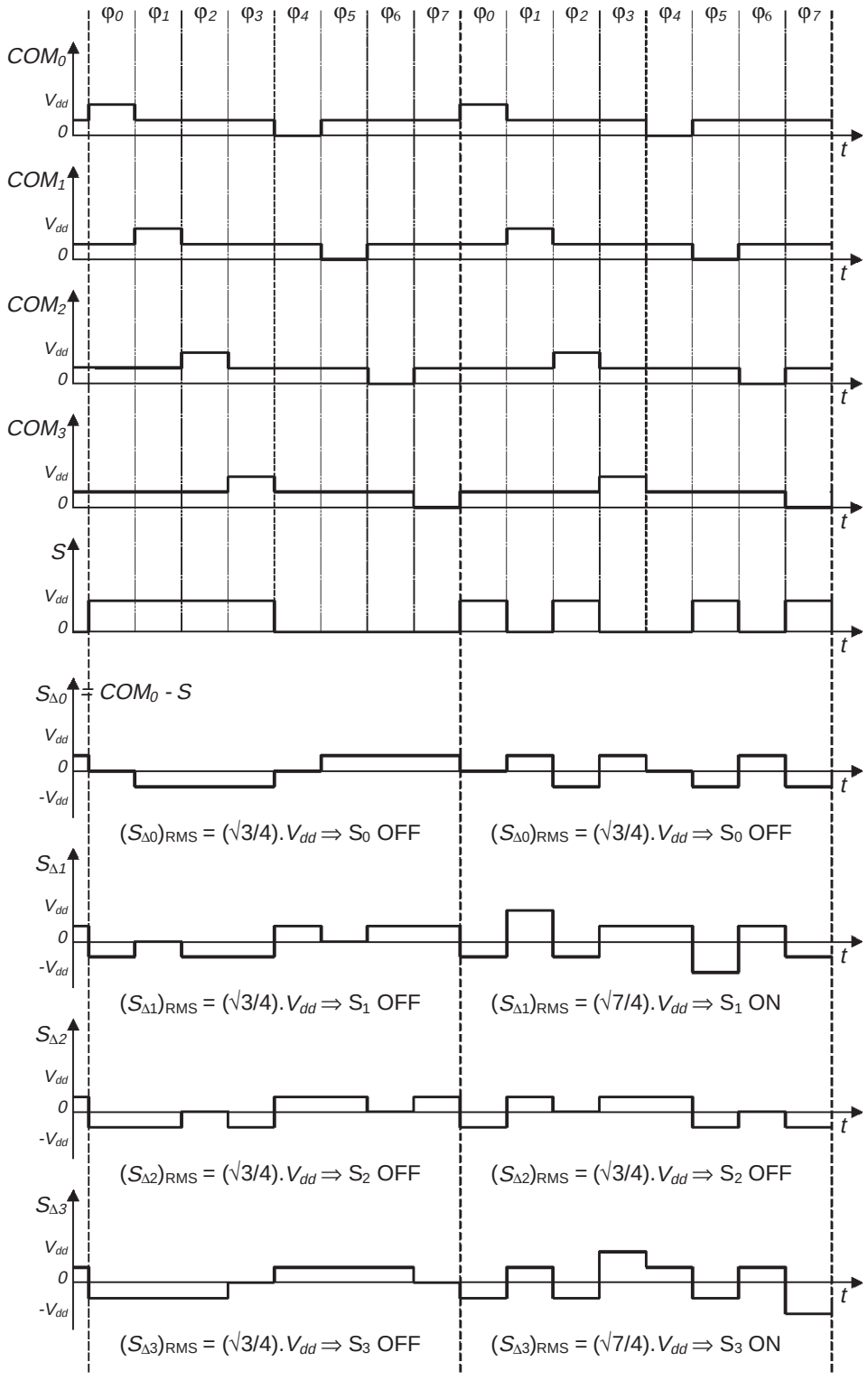
***S* waveform**	**Segment S_0**			**Other segments**		
	DC	RMS	state	DC	RMS	state
H during φ_0, L during φ_4	0	$(\sqrt{3}/4).V_{DD}$	OFF	0	only depend on the rest of *S* waveform	
L during φ_0, H during φ_4	0	$(\sqrt{7}/4).V_{DD}$	ON	0		

Note that even if a segment is OFF, its RMS voltage is not zero. As a result, contrast is not as good as in direct drive. In addition, there is a risk of cross-talk (or ghosting): if segment voltage ($S_{\Delta i}$) frequency is too high, a segment can become opaque even though the RMS voltage is below the threshold. So make sure that the driving frequency (considering the whole cycle, i.e. the 8 phases) is under 2000 Hz typically.

The intermediate voltage $V_{DD}/2$ is only required for the backplane voltages. The ST62 I/O pins connected to the backplanes are configured by software to output mode for 0 or V_{dd} levels or to high impedance input mode for $V_{DD}/2$. This intermediate voltage is defined by two equal-valued resistors externally connected to the I/O pin.

By using an MCU with flexible I/O pin configuration such as a standard ST62, 4-multiplexed LCD drive can be made with only 8 additional resistors.

Figure 5. LCD signals for 4-multiplexed mode (used in library)

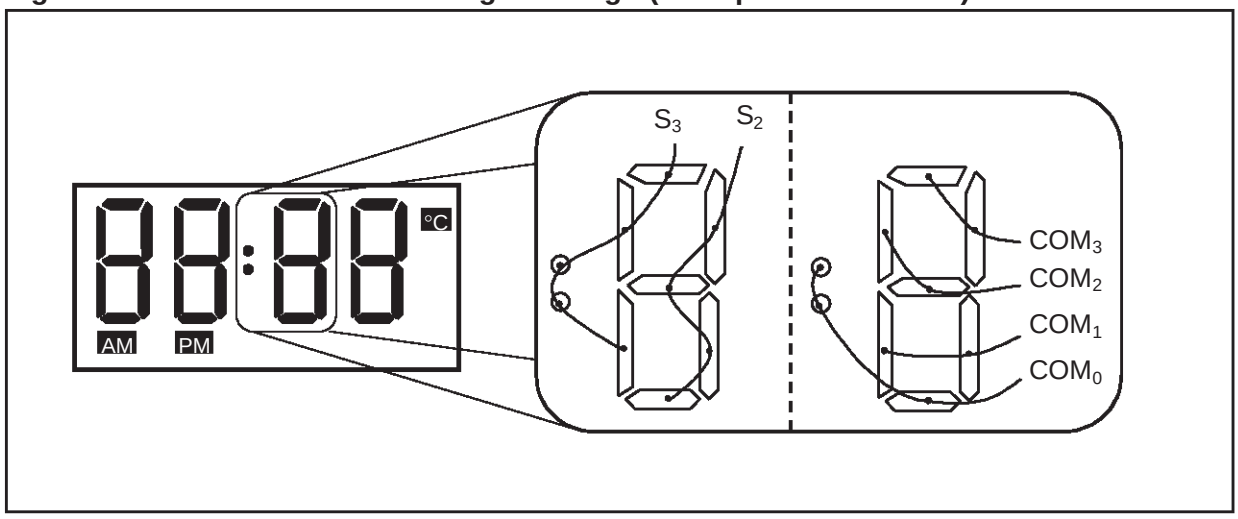


2 LCD DRIVING SOFTWARE LIBRARY

This library consists in one MAST6 source file, `LCD_drv.st6`, and its associated include file, `LCD_drv.inc`. It is targeted to a certain kind of LCD structure. Source code is provided to facilitate customisation to a particular LCD and application. The following section presents some guidelines on how to use and customize the library.

The targeted LCD is organized into four classical 7-segment digits, plus four icons (e.g. a colon at the middle), creating four “8-segment digits”. Each digit uses the four backplane pins and two frontplane pins.

Figure 6. Connections for an 8-segment digit (example with a colon)



Typically, this kind of LCD is suited to 24-hour clock display. Therefore, the digits are called, from left to right: “hours digit 1”, “hours digit 0”, “minutes digit 1”, “minutes digit 0”.

The first part of this section explains how to use the library provided the LCD is wired exactly like the target is, and provided the MCU pin allocations are compatible with the rest of the application. The second part gives more details on the data operations performed internally by the driver, to be able to customise it if necessary. Finally, the third part gives an example of how to manage timing resources to combine LCD requirements with the main tasks of the application.

2.1 NON-CUSTOMISED USAGE

2.1.1 Allocation of I/O resources

All the MCU output pins generating the *S* signals are located in the same I/O port, called “segments port”. A different I/O port, called “commons port” is used for the pins generating the *COM* signals. The software driver has no effect on the other I/O pins, even if they are located in one of those ports.

The code uses DR_seg, DDR_seg and OR_seg labels to access the configuration registers of the segments port. DR_com, DDR_com and OR_com labels are used for the commons port. DR_seg_2 and DR_com_2 are labels referring to RAM variables used as shadow I/O port Data Registers. These 8 labels are declared as external at the beginning of LCD_drv.st6. Therefore, to make the library work, you must define them as synonyms of actual configuration registers, like in the following example:

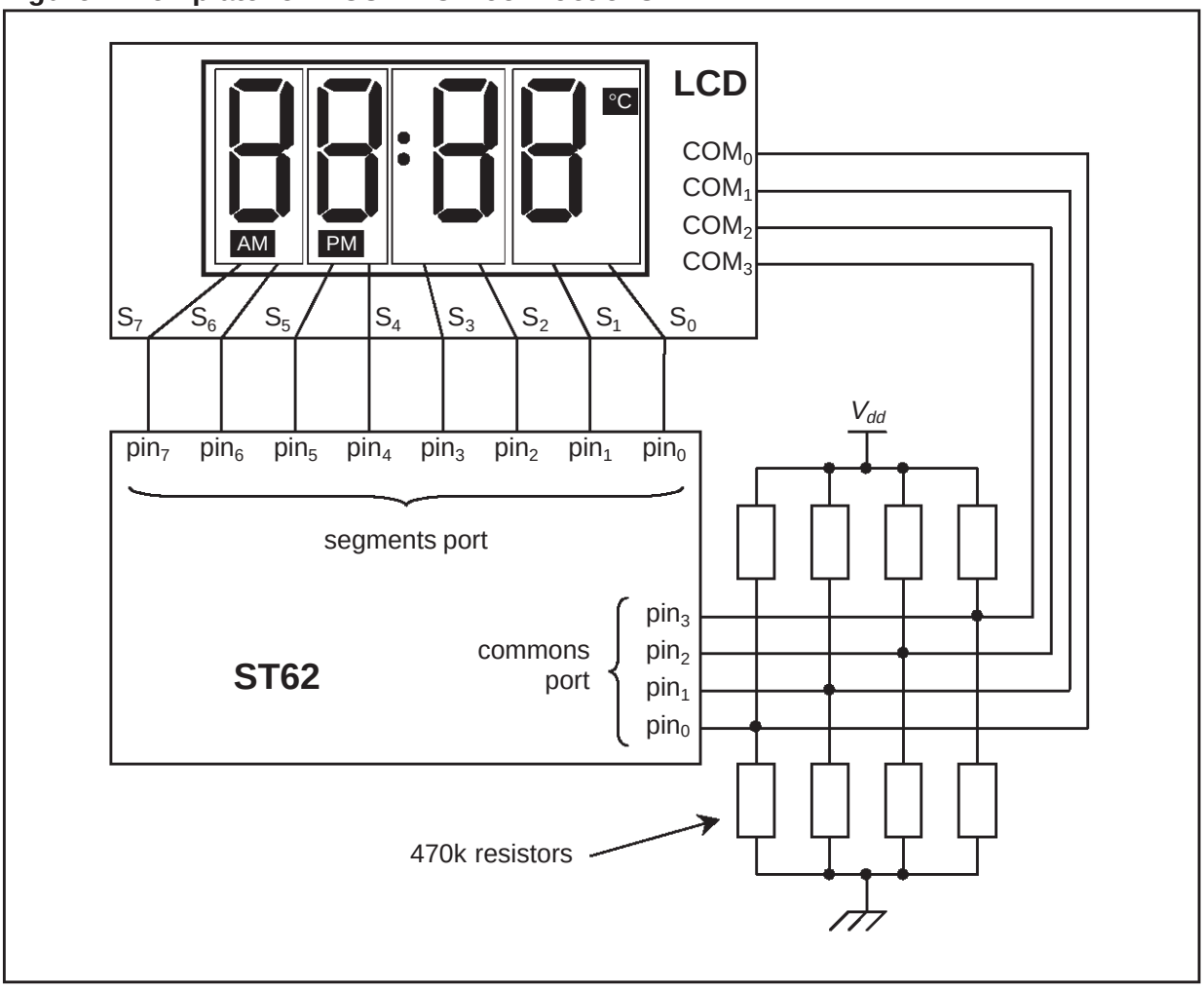
```
DDR_com                 DATA  DDRB
DR_com                  DATA  DRB
OR_com                  DATA  ORB

DDR_seg                 DATA  DDRA
DR_seg                  DATA  DRA
OR_seg                  DATA  ORA

DR_com_2                DATA  DRB_2
DR_seg_2                DATA  DRA_2
```

These definitions must be performed in another source file which is to be linked with LCD_drv.st6.

Note: in this example, DRA_2 and DRB_2 definitions must be in the same source file as DR_seg_2 and DR_com_2 definitions, otherwise the DATA directive does not work.

Once the segments and commons ports defined, the MCU must be wired according to Figure 6. Note that a pair of S pins is assigned to each “8-segment digit”. To understand the roles of each of the two pins, refer to Figure 6.

Figure 7. Template for MCU – LCD connections



2.1.2 Driver subroutines and variables

The main application communicates with the LCD driver through six 8-bit variables and 3 subroutines, all declared in `LCD_drv.inc`.

The six variables are written by the main application and read by the software LCD driver. They describe the information that should be displayed:

– `hr_dig1`, `hr_dig0`, `min_dig1` and `min_dig0` contain the code of the character to be displayed on each 7-segment digit;

– `icons` is a byte of flags indicating, for each icon, if it must be ON or OFF;

– `flashing` is a byte of flags indicating, for each 7-segment digit and each icon, if it must be flashing or not.

The driver supports 16 different characters to be displayed on a digit: the 10 numeric digits, some letters or symbols, or the blank digit. The character coding is included in the library through a look-up table.

Table 2. Character coding for 7-segment digits

Code	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
Display																

Table 3. Bit definitions for `icons` variable

7							0
-	-	-	-	ICO3	ICO2	ICO1	ICO0

Bit 3:0 = **ICO[3..0]** *Icon on/off*
These bits indicate, for each icon segment, if it must be on or off.
0: Icon segment off
1: Icon segment on

Table 4. Bit definitions for `flashing` variable

7							0
HR0	HR1	MIN0	MIN1	ICO3	ICO2	ICO1	ICO0

Bit 3:0 = **ICO[3..0]** *Icon flashing on/off*
These bits indicate, for each icon segment, if it must be flashing or not.
0: Icon segment not flashing
1: Icon segment flashing

Bit 5:4 = **MIN[1..0]** *Minute digit flashing on/off*
These bits indicate, for each minute 7-segment digit, if it must be flashing or not.
0: Digit not flashing
1: Digit flashing

Bit 7:6 = **HR[3..0]** *Hour digit flashing on/off*
These bits indicate, for each hour 7-segment digit, if it must be flashing or not.
0: Digit not flashing
1: Digit flashing

To use the LCD driver, proceed as follows:

– Before calling any LCD driver subroutine, the main routine must initialise the six display variables.

– Then, it must call the `LCD_Init` subroutine to initialise LCD driver internal variables and to configure segment port pins as output push-pull.

– Once the `LCD_Init` subroutine called, the main software must frequently call the `LCD_Do` subroutine. This subroutine updates the I/O ports so as to generate the required waveforms. The delay between two consecutive calls to `LCD_Do` represents the duration of 1 LCD phase, which is $1/8^{th}$ of the total LCD cycle (cf. Section 1.3).

– The six display variables can be modified at any time by the main software. Each time the `LCD_Do` subroutine is executed, it reports the changes on the waveforms, i.e. on the LCD.

– In parallel to `LCD_Do` calls, the main software must call the `LCD_Flash` subroutine. The delay between two consecutive calls to `LCD_Flash` represents half of the flashing period.

– It is the main software that is in charge of generating a time base (generally using a timer peripheral). This way, a single time base can be used at the same time for LCD driving, flashing frequency and other application tasks.

Important notice: if delays between calls to `LCD_Do` are too irregular, LCD segment absolute DC voltage can become too high, with a risk of damaging the LCD.

2.2 ADDITIONAL INFORMATION FOR CUSTOMISATION

The current LCD phase (φ_0 to φ_7) is stored into `LCD_Ph`, an 8-bit variable internal to the LCD driver. This variable is initialised by `LCD_Init` and incremented by `LCD_Do`, from 0 to 7 and back to 0. To update the configuration register of the segments and commons ports, `LCD_Do` uses `LCD_Ph` as an index to scan look-up tables.

Note: because computing the new register values takes time, `LCD_Do` stores the new values in RAM buffers, and updates all the real registers at the same time. This way, transitions on backplane and frontplane waveforms can be synchronised. This synchronisation helps keeping a low DC voltage on LCD segments. For Data Registers, the RAM buffer used is the shadow register.

The algorithms described in this section are designed to generate the proper backplane and frontplane signals as described in Section 1.3.

2.2.1 Generation of backplane signals

Each time `LCD_Do` is executed, it updates the three configuration registers (DDR, DR and OR) of the commons port, in order to output either 0, $V_{DD}/2$ or V_{DD} on the backplane pins.

`LCD_drv.st6` defines three constant tables giving the values of each configuration register depending on the current LCD phase:

```
wave_ddr:               DB    001h,002h,004h,008h,001h,002h,004h,008h
wave_dr:                DB    00fh,00fh,00fh,00fh,00eh,00dh,00bh,007h
wave_or:                DB    001h,002h,004h,008h,001h,002h,004h,008h
```

When applying these values to the port registers, LCD_Do uses a mask (COM_MASK equate) to modify only the appropriate pins. To modify the Data Register, it uses DR_com_2 shadow register.

How to customise: it is easy to modify the pins allocation for backplane pins, as long as they all belong to the same I/O port. You only need to change COM_MASK and the three constant tables.

2.2.2 Generation of frontplane signals

Here, LCD_Do only has to update the Data Register. But operations are more complex because of character coding.

The procedure contains four steps:

Step 1: Writing into variables that are images of min_dig0, min_dig1, hr_dig0, hr_dig1 and icons, taking flashing into account : the image variable contains either a blank value or the content of the original variable. This image variable represents what is really to be displayed on the LCD. For example, if a digit displays a flashing “9”, its image variable contains alternately ’9’ or ’f’ (code for blank digit).

Step 2: Updating the Data shadow Register only considering the 7-segment characters. This requires a constant table to store character coding.

Step 3: Updating the Data shadow Register taking the icons into account.

Step 4: Copying the Data shadow Register into the Data Register.

Steps 1 and 4 are independent from I/O pins allocation, so they will not be described here.

Step 2 starts by clearing the Data shadow Register for all the frontplane pins. Then, for each 7-segment digit, the cycle of operations described by Figure 8 modifies this shadow register. Due to the mask mechanism, the ADD operation is equivalent to an OR (the ST6 instruction set does not supply a direct OR operation).

Because all icon segments are located on the first backplane (COM_0), ***Step 3*** is performed only in phases φ_0 and φ_4. After Step 2, the Data shadow Register is configured in a way that all icon segments are OFF. Step 3 consists in correcting the shadow register to make sure that all required icons are ON. To do this, it performs bit manipulation instructions with the following bit definition equates:

```
ICON0_SEG               EQU   1
ICON1_SEG               EQU   3
ICON2_SEG               EQU   5
ICON3_SEG               EQU   7
```

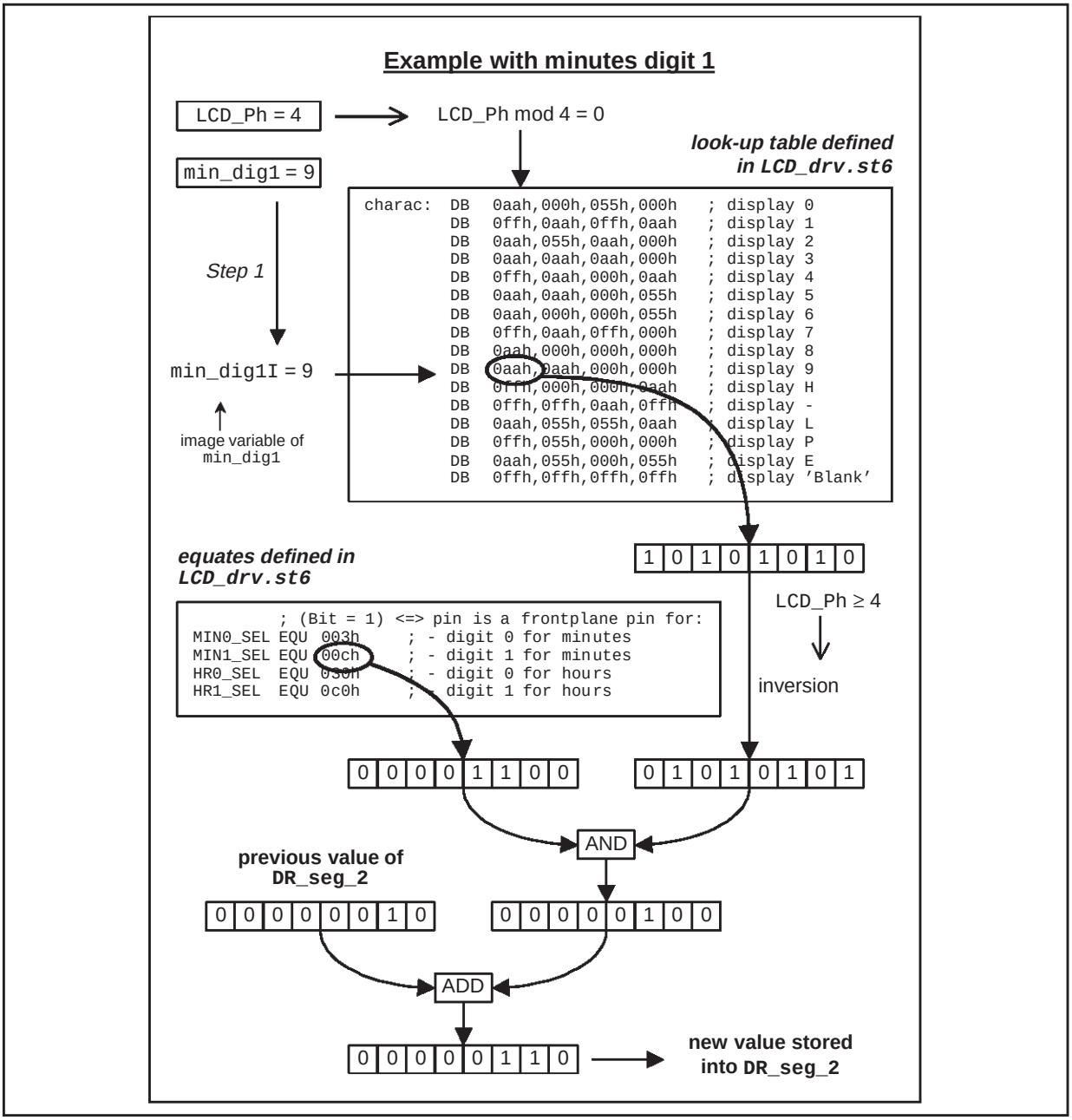
How to customise: as long as all frontplane pins belong to the same I/O port, it is possible to change I/O pin allocations or character coding just by modifying the equates and constant table definitions located at the beginning of `LCD_drv.st6`:

– the mask equates selecting all or a subset of the frontplane pins,

– the look-up table for character coding,

– the bit definition equates selecting frontplane pins related to icon segments.

Figure 8. Operations to convert a character code into a frontplane waveform



2.3 EXAMPLE OF APPLICATION TIMING – A SIMPLE CLOCK

As explained in Section 2.1, the main application is in charge of providing the time base to the LCD driver. Because this time base must be relatively precise, it is usually generated by a specific sub-system of the application, either internal to the MCU (timer peripheral) or external (e.g. external clock source, RC network, etc.). In both cases, MCU resources are dedicated to it (peripheral if internal, pins if external). This part describes a solution to share a single time base between the LCD driver and the main application. Sharing is usually necessary if the ST6 device has few resources.

The main application is a real time 24-hour clock to be displayed on the LCD. In order to use as few pins as possible, the time base is generated by a standard 8-bit timer clocked by the MCU internal clock.

The real time clock requires frequencies far lower than the LCD driver. Consequently, the time base runs at the frequency required by the LCD driver. Then, this frequency is divided by software counters to reach a period of 125 ms, then divided again to reach periods of 0.5 second, 1 second, 1 minute and 1 hour. The half-second period is involved in making some LCD segments flash at 1 Hz. For example, when the clock is running, the colon flashes at that frequency.

The standard timer is configured in output mode with interrupts enabled, so that the timer Interrupt Service Routine (ISR) is called every 1.5 ms. This routine calls the `LCD_Do` routine. As a result, an LCD cycle (8 phases) lasts 1.5*8 = 12 ms, so LCD voltage frequency is 83 Hz, which is in the required range.

Reaching a period of 125 ms requires dividing the timer interrupt frequency by 250/3. To do so, the timer ISR decrements a counter (RAM variable) three times. After each decrement, it checks if the counter has reached 0 or not. If it has, the ISR calls an RTC subroutine. This subroutine reloads the counter with 250.

The RTC subroutine, called once every 1/8 s, performs several frequency divisions to update the second, minute and hour counters. Also, it calls `LCD_Flash` once every half second.

3 APPENDIX: SOURCE CODE

LCD_drv.inc

```
;                   ************************************************** ***
;                   *  (C) 2001 STMicroelectronics                     *
;                   * Project    : 8*4 segment LCD software driver      *
;                   * Toolchain  : ST6 toolchain for RIDE V 1.0.C       *
;                   * Target     : in theory, any ST62                  *
;                   * Module     : LCD_drv.inc                          *
;                   * Version    : V 0.1.1 - May 2001                   *
;                   * Author     : T.B. Hong Kong Application           *
;                   ************************************************** ***
;                   * Software LCD driver library:                      *
;                   *      - definitions of the variables               *
;                   *      - constant tables                            *
;                   *      - driver subroutines                         *
;                   ************************************************** ***

;*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-* -*-*-*
; *-*-*-*-*- Variables and Equates *-*-*-*-*-*-*-
;*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-* -*-*-*

; Character codes
EXTERN DATA (min_dig0)                ; Digit 0 for minutes
EXTERN DATA (min_dig1)                ; Digit 1 for minutes
EXTERN DATA (hr_dig0)                 ; Digit 0 for hours
EXTERN DATA (hr_dig1)                 ; Digit 1 for hours

; Code              : 0 1 2 3 4 5 6 7 8 9 a b c d e f
; Display           : 0 1 2 3 4 5 6 7 8 9 H - L P E
BLANK               EQU   0fh
DASH                EQU   0bh

EXTERN DATA (icons)                   ; Byte of flags for icons on/off state
                                      ; (0 = off, 1 = on)
EXTERN DATA (flashing)                ; Byte of flags for digits and icons
                                      ; flashing state
                                      ; (0 = not flashing, 1 = flashing)
; Bit definitions for 'icons' and 'flashing' variables
ICO0                EQU   0           ; bit 0: icon 0
ICO1                EQU   1           ; bit 1: icon 1
ICO2                EQU   2           ; bit 2: icon 2
ICO3                EQU   3           ; bit 3: icon 3
MIN0                EQU   4           ; bit 4: digit 0 for minutes
MIN1                EQU   5           ; bit 5: digit 1 for minutes
HR0                 EQU   6           ; bit 6: digit 0 for hours
HR1                 EQU   7           ; bit 7: digit 1 for hours
```

```
;*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-* -*-*-*
; *-*-*-*-*-*-*-* Subroutines *-*-*-*-*-*-*-*-*-*
;*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-* -*-*-*

;**************************************** **********************
EXTERN CODE (LCD_Init)
                    ; Initialises the ressources used by the library.
                    ; Prereq.    : none
                    ; Inputs     : none
                    ; Outputs    : internal variables initialised,
                    ;              segment pins configured as output push-pull
;**************************************** **********************

;**************************************** **********************
EXTERN CODE (LCD_Flash)
                    ; Switches the flash strobe if necessary
                    ; Prereq.    : none
                    ; Inputs     : flashing
                    ; Outputs    : none
;**************************************** **********************

;**************************************** **********************
EXTERN CODE (LCD_Do)
                    ; Updates the LCD outputs (commons & segments) to
                    ; display the requested digits.
                    ; Prereq.    : 'LCD_Init' called before
                    ; Inputs     : 'min_dig0', 'min_dig1', 'hr_dig0', 'hr_dig1'
                    ;              + 'icons'
                    ; Outputs    : 'DDR_com', 'DR_com', 'DR_com_2', 'OR_com',
                    ;              'DR_seg' and 'DR_seg_2' refreshed
                    ; IMPORTANT: once the LCD I/Os are initialised, this
                    ; subroutine must be called frequently to prevent
                    ; damaging the LCD.
;**************************************** **********************

;****************** (C) 2001 STMicroelectronics ****************
```

LCD_drv.st6

```
;                   ***************************************************** ***
;                   *  (C) 2001 STMicroelectronics                       *
;                   * Project    : 8*4 segment LCD software driver        *
;                   * Toolchain : ST6 toolchain for RIDE V 1.0.C         *
;                   * Target     : in theory, any ST62                    *
;                   * Module     : LCD_drv.st6                            *
;                   * Version    : V 0.1.2 - June 2001                    *
;                   * Author     : T.B. Hong Kong Application             *
;                   ***************************************************** ***
;                   * Software LCD driver library:                       *
;                   *  - public and local variables                      *
;                   *  - constant tables                                 *
;                   *  - driver subroutines                              *
;                   ***************************************************** ***

$INCLUDE (LCD_drv.inc)                ; Software LCD driver

;*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-* -*-*-*
; *-*-*-*-*- I/O configuration equates *-*-*-*-
;*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-* -*-*-*

; --- Port allocation ---

EXTERN DATA (DDR_com, DR_com, OR_com)         ; LCD commons
EXTERN DATA (DDR_seg, DR_seg, OR_seg)         ; LCD segments
EXTERN DATA (DR_com_2, DR_seg_2)              ; LCD shadow registers

; --- Pin allocation ---

COM_MASK            EQU   0f0h

SEG_MASK            EQU   000h
SEG_SEL             EQU   0ffh
MIN0_SEL            EQU   003h
MIN1_SEL            EQU   00ch
HR0_SEL             EQU   030h
HR1_SEL             EQU   0c0h

ICON0_SEG           EQU   1
ICON1_SEG           EQU   3
ICON2_SEG           EQU   5
ICON3_SEG           EQU   7
```

```
;*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-* -*-*-*
; *-*-*-*-*-*-*-*- Variables *-*-*-*-*-*-*-*-*-
;*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-* -*-*-*

LCD_vars            SEGMENT DATA
RSEG LCD_vars

;--- Public variables ---

PUBLIC min_dig0, min_dig1, hr_dig0, hr_dig1
PUBLIC icons, flashing

; Character codes
min_dig0:           DS    1     ; Digit 0 for minutes
min_dig1:           DS    1     ; Digit 1 for minutes
hr_dig0:            DS    1     ; Digit 0 for hours
hr_dig1:            DS    1     ; Digit 1 for hours

; Code              : 0 1 2 3 4 5 6 7 8 9 a b c d e f
; Display           : 0 1 2 3 4 5 6 7 8 9 H - L P E

icons:              DS    1     ; Byte of flags for icons on/off state
                                ; (0 = off, 1 = on)
flashing:           DS    1     ; Byte of flags for digits and icons
                                ; flashing state
                                ; (0 = not flashing, 1 = flashing)

;--- Local variables ---

LCD_Ph:             DS    1     ; LCD phase (0 to 7)

DDR_com2:           DS    1
OR_com2:            DS    1

strobe:             DS    1     ; Flash strobe (000h <--> 0ffh)
mindig0I:           DS    1     ; Digit 0 for minutes - IMAGE
mindig1I:           DS    1     ; Digit 1 for minutes - IMAGE
hrdig0I:            DS    1     ; Digit 0 for hours - IMAGE
hrdig1I:            DS    1     ; Digit 1 for hours - IMAGE
iconsI:             DS    1     ; Icons - IMAGE
```

```
;*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-* -*-*-*
; *-*-*-*-*-*-* Constant tables *-*-*-*-*-*-*-*-*
;*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-* -*-*-*

; Table of common I/Os configuration for commons waveform

Com_table           SEGMENT CODE INWINDOW
RSEG Com_table

; phase |             0     1      2      3      4      5      6      7
;--------------------------------------- ----------------------------
; COM0 |             Vdd  Vdd/2  Vdd/2 Vdd/2   0    Vdd/2  Vdd/2 Vdd/2
; COM1 |            Vdd/2  Vdd   Vdd/2 Vdd/2  Vdd/2   0    Vdd/2 Vdd/2
; COM2 |            Vdd/2 Vdd/2   Vdd  Vdd/2  Vdd/2  Vdd/2   0    Vdd/2
; COM3 |            Vdd/2 Vdd/2  Vdd/2  Vdd   Vdd/2  Vdd/2  Vdd/2   0


wave_ddr:           DB    001h,002h,004h,008h,001h,002h,004h,008h
wave_dr:            DB    00fh,00fh,00fh,00fh,00eh,00dh,00bh,007h
wave_or:            DB    001h,002h,004h,008h,001h,002h,004h,008h

; Table of segment outputs to display a specific character

Seg_table           SEGMENT CODE INWINDOW
RSEG Seg_table

charac:             DB    0aah,000h,055h,000h          ; display 0
                    DB    0ffh,0aah,0ffh,0aah          ; display 1
                    DB    0aah,055h,0aah,000h          ; display 2
                    DB    0aah,0aah,0aah,000h          ; display 3
                    DB    0ffh,0aah,000h,0aah          ; display 4
                    DB    0aah,0aah,000h,055h          ; display 5
                    DB    0aah,000h,000h,055h          ; display 6
                    DB    0ffh,0aah,0ffh,000h          ; display 7
                    DB    0aah,000h,000h,000h          ; display 8
                    DB    0aah,0aah,000h,000h          ; display 9
                    DB    0ffh,000h,000h,0aah          ; display H
                    DB    0ffh,0ffh,0aah,0ffh          ; display -
                    DB    0aah,055h,055h,0aah          ; display L
                    DB    0ffh,055h,000h,000h          ; display P
                    DB    0aah,055h,000h,055h          ; display E
                    DB    0ffh,0ffh,0ffh,0ffh          ; display 'Blank'
```

```
;*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-* -*-*-*
; *-*-*-*-*-*-*-* Subroutines *-*-*-*-*-*-*-*-*-*
;*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-* -*-*-*

PUBLIC LCD_Init, LCD_Flash, LCD_Do

LCD_subs            SEGMENT CODE
RSEG LCD_subs

;************************************** ************************
LCD_Init:           ; Initialises the ressources used by the library.
                    ; Prereq.    : display variables must have been initialised
                    ;              by the main routine
                    ; Inputs     : none
                    ; Outputs    : internal variables initialised,
                    ;              segment pins configured as output push-pull
;************************************** ************************
                    clr   LCD_Ph
                    clr   strobe

                    ldi   a,SEG_SEL
                    add   a,DDR_seg
                    ld    DDR_seg,a

                    ldi   a,SEG_SEL
                    add   a,OR_seg
                    ld    OR_seg,a

                    ; Output internal to the driver:
                    ;  'LCD_Ph' and 'strobe' initialised'
                    ret

;************************************** ************************
LCD_Flash:          ; Switches the flash strobe if necessary
                    ; Prereq.    : none
                    ; Inputs     : 'flashing'
                    ; Outputs    : none
;************************************** ************************
                    ld    a,strobe
                    jrnz  strobe1

strobe0:            ld    a,flashing
                    jrz   exit_sub
                    ldi   strobe,0ffh
exit_sub:           ret

strobe1:            clr   strobe
                    ret
```

```
;************************************** ***********************
LCD_Do:             ; Updates the LCD outputs (commons & segments) to
                    ; display the requested digits.
                    ; Prereq.    : 'LCD_Init' called before
                    ; Inputs     : 'min_dig0', 'min_dig1', 'hr_dig0', 'hr_dig1'
                    ;             + 'icons'
                    ; Outputs    : 'DDR_com', 'DR_com', 'DR_com_2', 'OR_com',
                    ;              'DR_seg' and 'DR_seg_2' refreshed
                    ; IMPORTANT: once the LCD I/Os are initialised, this
                    ; subroutine must be called frequently to prevent
                    ; damaging the LCD.
;************************************** ***********************

;--- Depending on flash strobe, update segment images ---
                    ld    a,min_dig0
                    jrr   0,strobe,min0
                    jrr   MIN0,flashing,min0
                    ldi   a,BLANK
min0:               ld    mindig0I,a

                    ld    a,min_dig1
                    jrr   0,strobe,min1
                    jrr   MIN1,flashing,min1
                    ldi   a,BLANK
min1:               ld    mindig1I,a

                    ld    a,hr_dig0
                    jrr   0,strobe,hr0
                    jrr   HR0,flashing,hr0
                    ldi   a,BLANK
hr0:                ld    hrdig0I,a

                    ld    a,hr_dig1
                    jrr   0,strobe,hr1
                    jrr   HR1,flashing,hr1
                    ldi   a,BLANK
hr1:                ld    hrdig1I,a

                    clr   a
                    jrr   0,strobe,iconsegs
                    ld    a,flashing
iconsegs:           com   a
                    and   a,icons
                    ld    iconsI,a
```

```
;--- Update segments outputs (shadow register) ---

seg_upd:            ld    a,DR_seg_2
                    andi  a,SEG_MASK
                    ld    DR_seg_2,a

                    ld    a,LCD_Ph
                    andi  a,03h
                    ld    V,a               ; v = LCD_Ph mod 4

                    ; -- Numeric digits --

                    ; Digit 0 for minutes
                    ld  a,mindig0I
                    ldi W,MIN0_SEL
                     call find_seg

                    ; Digit 1 for minutes
                     ld  a,mindig1I
                     ldi W,MIN1_SEL
                     call find_seg

                    ; Digit 0 for hours
                    ld  a,hrdig0I
                    ldi W,HR0_SEL
                    call find_seg

                    ; Digit 1 for hours
                    ld  a,hrdig1I
                    ldi W,HR1_SEL
                    call find_seg

                    ; -- Icons --

                    ld    a,V
                    jrz   do_icon0          ; All icon segments are on COM0
                    jp    com_upd

                    ; icon 0 segment
do_icon0:           jrr   ICO0,iconsI,do_icon1
                    jrs   2,LCD_Ph,icon0_1
icon0_0:            res   ICON0_SEG,DR_seg_2
                    jp    do_icon1
icon0_1:            set   ICON0_SEG,DR_seg_2
```

```
                    ; icon 1 segment
do_icon1:           jrr   ICO1,iconsI,do_icon2
                    jrs   2,LCD_Ph,icon1_1
icon1_0:            res   ICON1_SEG,DR_seg_2
                    jp    do_icon2
icon1_1:            set   ICON1_SEG,DR_seg_2

                    ; icon 2 segment
do_icon2:           jrr   ICO2,iconsI,do_icon3
                    jrs   2,LCD_Ph,icon2_1
icon2_0:            res   ICON2_SEG,DR_seg_2
                    jp    do_icon3
icon2_1:            set   ICON2_SEG,DR_seg_2

                    ; icon 3 segment
do_icon3:           jrr   ICO3,iconsI,com_upd
                    jrs   2,LCD_Ph,icon3_1
icon3_0:            res   ICON3_SEG,DR_seg_2
                    jp    com_upd
icon3_1:            set   ICON3_SEG,DR_seg_2

;--- Update commons I/Os (shadow registers) ---

com_upd:            ldi   DWR,#WINDOW(wave_ddr)

                    ; Update DDR
                    ldi   a,#WINOFFSET(wave_ddr)
                    add   a,LCD_Ph
                    ld    x,a
                    ld    a,DDR_com
                    andi  a,COM_MASK
                    add   a,(x)
                    ld    DDR_com2,a

                    ; Update DR and DR_2
                    ldi   a,#WINOFFSET(wave_dr)
                    add   a,LCD_Ph
                    ld    x,a
                    ld    a,DR_com_2
                    andi  a,COM_MASK
                    add   a,(x)
                    ld    DR_com_2,a

                    ; Update OR
                    ldi   a,#WINOFFSET(wave_or)
```

```
                    add   a,LCD_Ph
                    ld    x,a
                    ld    a,OR_com
                    andi  a,COM_MASK
                    add   a,(x)
                    ld    OR_com2,a

;--- Perform the changes on the real ports ---

                    ; Segments port
                    ld    a,DR_seg_2
                    ld    DR_seg,a

                    ; Commons port
                    ld    a,OR_com
                    andi  a,COM_MASK
                    ld    OR_com,a
                    ld    a,DDR_com2
                    ld    DDR_com,a
                    ld    a,DR_com_2
                    ld    DR_com,a
                    ld    a,OR_com2
                    ld    OR_com,a

;--- Increment phase counter ---

                    inc   LCD_Ph
                    ld    a,LCD_Ph
                    andi  a,07h
                    ld    LCD_Ph,a                ; LCD_Ph = (LCD_Ph + 1) mod 8

                     ; Output internal to the driver:
                     ; 'LCD_Ph' updated
                     ret

;************************************* ***********************
find_seg:           ; Finds the segment outputs for a given digit, a given
                    ; character and a given LCD phase.
                    ; Prereq.    : none
                    ; Inputs     : 'LCD_Ph', V = LCD_Ph mod 4,
                    ;             a = code of the character (0 to 15),
                    ;             W = segment I/O port mask for the digit
                    ; Outputs    : 'DR_seg_2' refreshed
                    ; Modifies   : X
;************************************* ***********************
```

```
                    sla   a
                    sla   a                       ; * 4
                    ldi DWR,#WINDOW(charac)
                    addi  a,#WINOFFSET(charac)
                    add   a,V
                    ld    x,a
                    ld    a,(x)

                    jrr   2,LCD_Ph, dr_chg        ; If LCD_Ph >= 4,
                    com   a                       ; invert outputs

dr_chg:             and   a,W
                    add   a,DR_seg_2
                    ld    DR_seg_2,a
                    ret

END

;****************** (C) 2001 STMicroelectronics *****************
```

”THE PRESENT NOTE WHICH IS FOR GUIDANCE ONLY AIMS AT PROVIDING CUSTOMERS WITH INFORMATION REGARDING THEIR PRODUCTS IN ORDER FOR THEM TO SAVE TIME. AS A RESULT, STMICROELECTRONICS SHALL NOT BE HELD LIABLE FOR ANY DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES WITH RESPECT TO ANY CLAIMS ARISING FROM THE CONTENT OF SUCH A NOTE AND/OR THE USE MADE BY CUSTOMERS OF THE INFORMATION CONTAINED HEREIN IN CONNEXION WITH THEIR PRODUCTS.”