To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

Application Note

V850/SA1

32-/16-BIT SINGLE-CHIP MICROCONTROLLER

LCD Controller Emulation

μPD703014A
μPD703014AY
μPD703015A
μPD703015AY
μPD703017A
μPD703017AY
μPD70F3017A
μPD70F3017AY

Document No. U14737EE1V0AN00 (1st edition)
Date Published February 2000

NOTES FOR CMOS DEVICES

① PRECAUTION AGAINST ESD FOR SEMICONDUCTORS

Note:

Strong electric field, when exposed to a MOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop generation of static electricity as much as possible, and quickly dissipate it once, when it has occurred. Environmental control must be adequate. When it is dry, humidifier should be used. It is recommended to avoid using insulators that easily build static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work bench and floor should be grounded. The operator should be grounded using wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions need to be taken for PW boards with semiconductor devices on it.

② HANDLING OF UNUSED INPUT PINS FOR CMOS

Note:

No connection for CMOS device inputs can be cause of malfunction. If no connection is provided to the input pins, it is possible that an internal input level may be generated due to noise, etc., hence causing malfunction. CMOS devices behave differently than Bipolar or NMOS devices. Input levels of CMOS devices must be fixed high or low by using a pull-up or pull-down circuitry. Each unused pin should be connected to V_{DD} or GND with a resistor, if it is considered to have a possibility of being an output pin. All handling related to the unused pins must be judged device by device and related specifications governing the devices.

③ STATUS BEFORE INITIALIZATION OF MOS DEVICES

Note:

Power-on does not necessarily define initial status of MOS device. Production process of MOS does not define the initial operation status of the device. Immediately after the power source is turned ON, the devices with reset function have not yet been initialized. Hence, power-on does not guarantee out-pin levels, I/O settings or contents of registers. Device is not initialized until the reset signal is received. Reset operation must be executed immediately after power-on for devices having reset function.

Purchase of NEC I^2C components conveys a license under the Philips I^2C Patent Rights to use these components in an I^2C system, provided that the system conforms to the I^2C Standard Specification as defined by Philips.

The export of these products from Japan is regulated by the Japanese government. The export of some or all of these products may be prohibited without governmental license. To export or re-export some or all of these products from a country other than Japan may also be prohibited without a license from that country. Please call an NEC sales representative.

License not needed: μPD70F3017A, 70F3017AY

The customer must judge the need for license:
μPD703014A, 703014AY, 703015A, 703015AY,μPD703017A, 703017AY

V850 Family, V850/SA1, and EEPROM are trademarks of NEC Corporation.

.

PREFACE

Readers This manual is intended for users who understand the V850/SA1 (μPD703014A, μPD703014AY, μPD703015A, μPD703015AY, μPD703017A, μPD703017AY, μPD70F3017A, μPD70F3017AY) functions and who design application systems using these products.

Purpose The purpose of this application note is to give the users an idea of how to emulate a LCD controller function by using the V850/SA1.

Organization This application note is divided into the following sections

- Introduction
- LCD Driving and Control
- Application Example

How to Read This Manual

In these application notes, it is assumed that the reader has general knowledge of electrical engineering, logic circuits, and microcontrollers. The program and hardware configurations published here are just examples and not intended for mass production.

For details of V850/SA1 hardware functions
→ Refer to the **V850/SA1 Hardware User's Manua**l.

For details of V850/SA1 instruction functions
→ Refer to the **V850 Family Architecture User's Manua**l.

Legend Symbols and notation are used as follows:

Weight in data notation	:	Left is high-order column, right is low order column
Active row notation	:	$\overline{\text{xxx}}$ (pin or signal name is over-scored) or /xxx (slash before signal name)
Memory map address:	:	High order at high stage and low order at low stage
Note	:	Explanation of (Note) in the text
Caution	:	Item deserving extra attention
Remark	:	Supplementary explanation to the text
Numeric notation	:	Binary . . . xxxx or xxxB Decimal . . . xxxx Hexadecimal . . . xxxxH or 0x xxxx

Prefixes representing powers of 2 (address space, memory capacity)

k (kilo) : 2^{10} = 1024

M (mega) : 2^{20} = 1024^2 = 1.048.576

G (giga) : 2^{30} = 1024^3 = 1.073.741.824

Author(s) T. Höveken
NEC Electronics (Europe) GmbH, Düsseldorf (Germany)
Technical Product Support

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION

1.1 General

Whenever a display unit in a low power consumption application is required, a liquid crystal display (LCD) is the adequate solution. LCD's can be found in many low power consumption applications like watches, thermometers, radio displays, measurement equipment and so on.
This application note shall give an outlook of LCD controlling and driving by NEC's low-power consumption 32-bit microcontroller V850/SA1 without any special on-chip LCD controller, but by port emulation.
The LCD's which are applicable for this intention are standard TN LCD's (Twisted Nematic) with an operation voltage of 3 V.

1.2 Basic Operation of an LCD

The principle structure of an LCD (TN-type) is a liquid crystal mixture between two substrates, like a sandwich. The inside surfaces of this cell are coated with a polymer that is grooved to align the molecules of the liquid crystal. The alignment of the molecules on the surface follows the direction of grooving. For such an LCD, the two surfaces are grooved orthogonal to one another and thus forming a 90 degree twist of the liquid crystals from one surface to the other.

A linear polarizing filter is applied to the front and the back of the cell. The polarizing filter on the back is also called analyzer. When this two polarizing filters are arranged along perpendicular polarizing axes, light entering from the front is re-directed 90 degrees along the helical structure of liquid crystal molecules so that it pass through the analyzer. In this case the light will be reflected back through the cell and the observer will see the background of the display, usually the silver gray of the reflector.

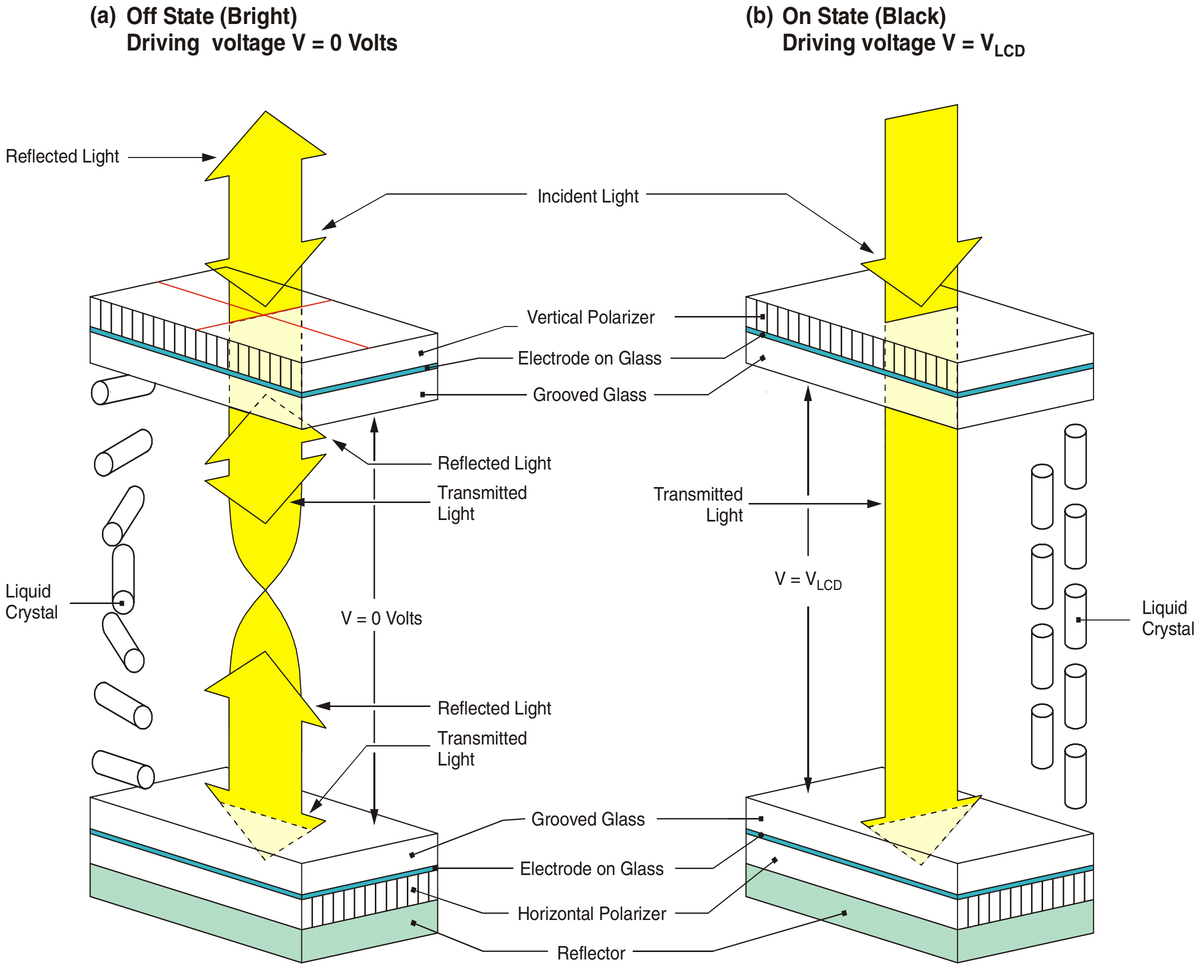
In order to change the alignment of the liquid crystal molecules, an appropriated electric field has to set up across the cell. This can be achieved by applying a driving voltage to the cell electrodes, which are situated on the LCD glass between the alignment layer and polarizing filters of the front and the back substrates respectively. These electrodes are called segment planes on the front substrate and common planes (or backplanes) on the back substrate.

The liquid crystal molecules will straighten out of their helix pattern in the direction of the electric filed and stop redirecting the angle of the light. Thus the light entering the polarizing filter of the front passes through the cell unaffected and is absorbed by the rear analyzer. The observer will see a black segment on a silver gray background.

After turning off the electric field, the liquid crystal molecules relax back to their 90 degree twist structure.

Cell electrodes on the front substrate are normally constitute the segment lines constitute and the electrodes on the back substrate are summarized to common lines, also called backplanes. Because of the summary of common lines, the number of independent driving lines (for front and backplane) can be reduced drastically. In the following chapters different driving methods will be introduced and compared concerning their applicability to V850/SA1.

Figure 1-1. Principle of LCD's Operation (using Reflected Light)



CHAPTER 2 LCD DRIVING & CONTROL

2.1 LCD Driving Signals

2.1.1 AC voltage driving

Unlike other display technologies that respond to peak or average voltage and current, LCD's are sensitive to the RMS voltage across a certain segment to the common plane. This behavior is important for the following drive schemes.

Of course, DC voltage has also an RMS value. However, a long term DC operation of an LCD may cause irreversible electrochemical reactions inside the display, which will lead into a reduced lifetime. Therefore a DC offset voltage above 50 mV must be avoided.

An RMS voltage without DC component can be achieved by changing the voltage level cyclical. The cycle frequencies of LCD glasses are typically in the range from 30 Hz to 100 Hz. Cycle frequencies below 30 Hz result in a flicker to the observer. Frequencies above 100 Hz are not recommended, since this will increase the power consumption without a remarkable improvement for the observer. Nevertheless the LCD, depending on size and design, can be operated at frequencies above several 100 Hz.

2.1.2 Multiplex control

On the assumption that each segment and common plane has to be driven individually, 2×N lines would be required for a LCD of N segments.
However, this can be reduced immediately since each common plane can be driven with the same signal, and thus the common planes can be unite to one common line. Just the segment planes have to be controlled independently. This mode is called the **static mode** and requires **(N+1)** control lines.
The number of controlled segments can be doubled, or vice versa the number segment lines can be halved, using an additional common line. Because of the two common lines it is defined as **duplex mode** and requires **((N/2)+2)** control lines.
Accordingly the **triplex mode** comprises three common lines with **((N/3)+3)** control lines in total. Eventually the expansion to four common lines results in the **quadruplex mode** with **((N/4)+4)** control lines.
The multiplex order could be extended step by step. But due to the more complex driving signal structure by different bias methods, a higher multiplex level than quadruplex can hardly achieved by port emulation.

Table 2-1. Comparison of Control Modes and Segment/Common Lines

Multiplex Control	Number of		Total Lines Note
	Segment Lines Note	Common Lines	
Static (1/1 duty)	N	1	N + 1
Duplex (1/2 duty)	N/2	2	(N/2) + 2
Triplex (1/3 duty)	N/3	3	(N/3) + 3
Quadruplex (1/4 duty)	N/4	4	(N/4) + 4

Note N is the total number of segments.

2.1.3 Bias voltage

An individual pixel on an LCD panel appears when the potential difference of the corresponding common signal and segment signal reaches or exceeds a given voltage (the LCD drive voltage V_{LCD}).
As seen before, the segment and the common voltage levels have to change cyclical, even if the LCD is static driven.
However, unlike the static mode the segment lines in the multiplex modes (duplex, triplex, quadruplex) are assigned to more than one common line. Because of this the common line control must support an off-state too, where the resulting voltage between segment and common plane is low enough to keep an unselected segment switched off. But even in the off-state a DC component must be avoided. This can be solved by driving a bias voltage (V_{BIAS}) less than V_{LCD}. The result is an AC square wave of $+V_{BIAS}/-V_{BIAS}$ between segment and common line, and if the LCD panel is designed for that bias voltage, such an signal does not switch on the segment.

The applicable biases for certain LCD control modes are shown in Table 2-2.

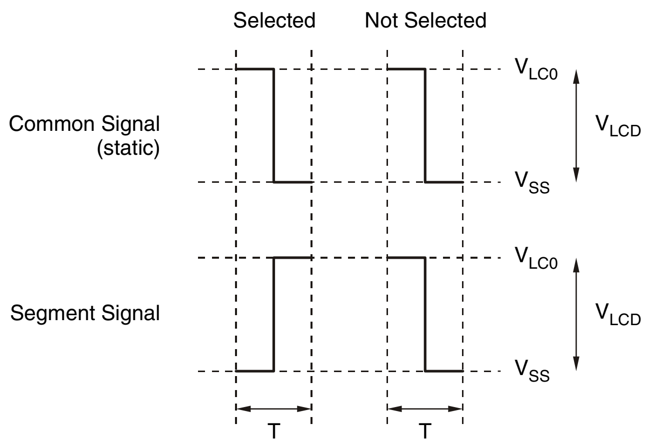
Table 2-2. Control Modes vs. applicable Bias Methods

Multiplex Control	Bias Methods
Static (1/1 duty)	No Bias
Duplex (1/2 duty)	1/2 V_{LCD}
Triplex (1/3 duty)	1/2 V_{LCD} or 1/3 V_{LCD}
Quadruplex (1/4 duty)	1/3 V_{LCD}

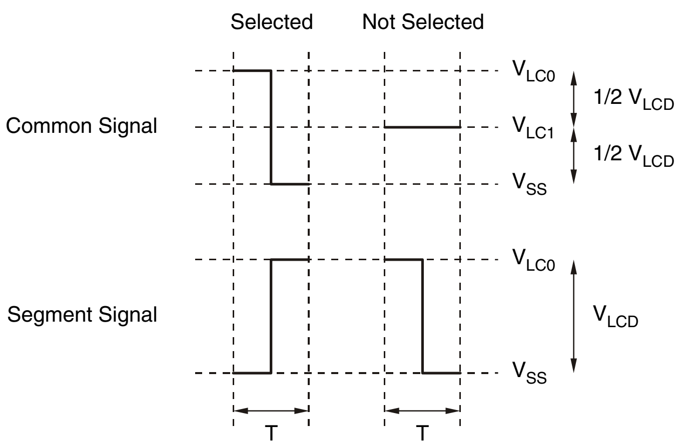
The common and segment signal voltages and phases per single for selected or non-selected level of the different bias methods are shown in the following Figure 2-1

Figure 2-1. Common Signal and Segment Signal Voltages and Phases

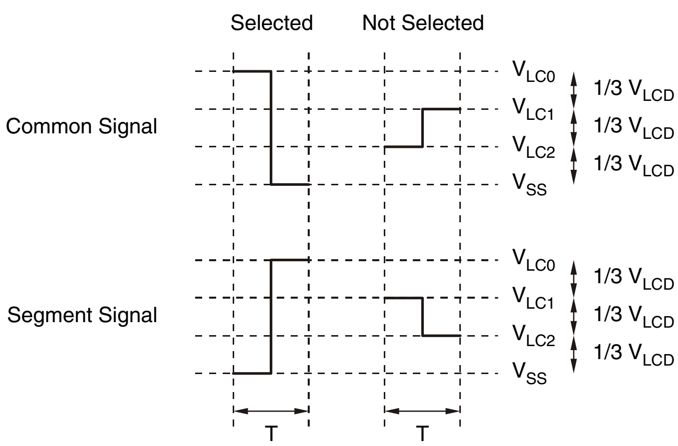
(a) Static display mode



(b) 1/2 Bias method

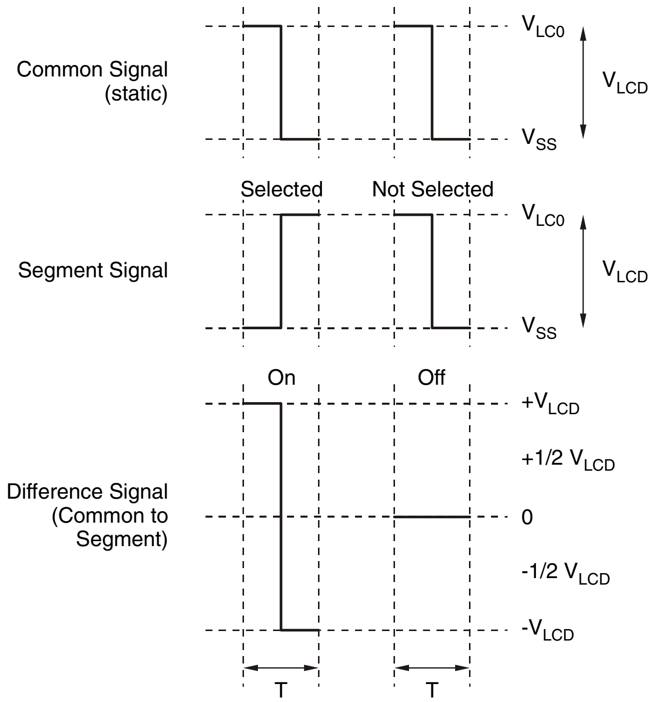


(c) 1/3 Bias method



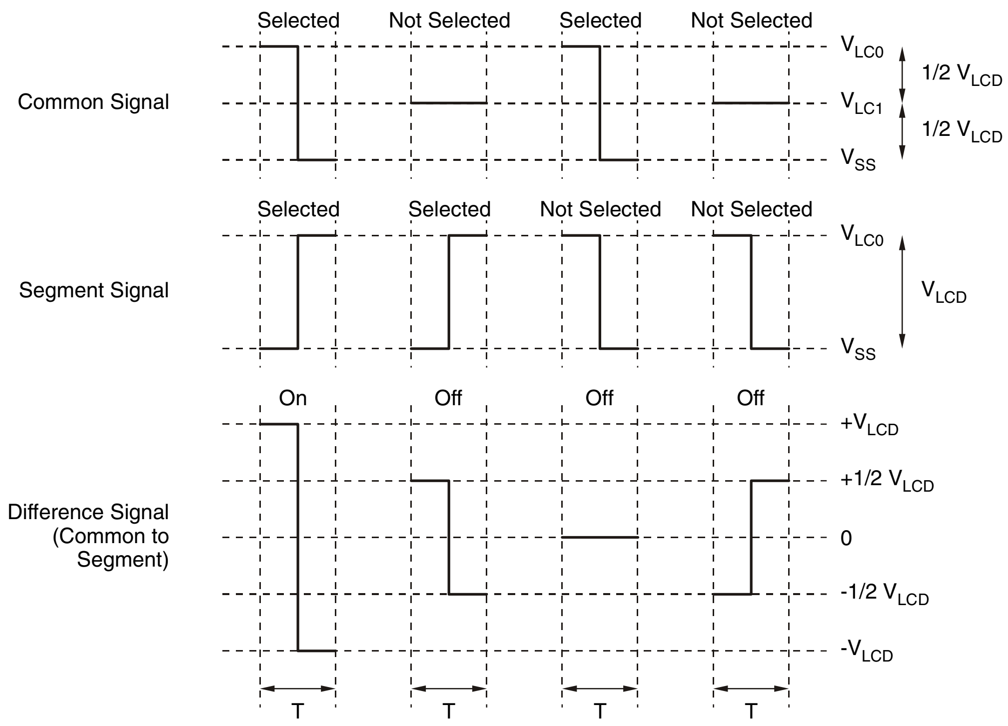
The resulting difference signals of the static display mode are represented in Figure 2-2As expected the static display mode provides the best contrast, due to the highest on-off-voltage ratio. But as seen before, in the static display mode the signals can take on only two independent conditions. The common signal is static after all, and each segment on the LCD has to be driven by its own individually segment line. Therefore, when large LCD's are used, this mode becomes inefficient concerning the required number of segment lines.

Figure 2-2. Difference Signals of Static Display Mode



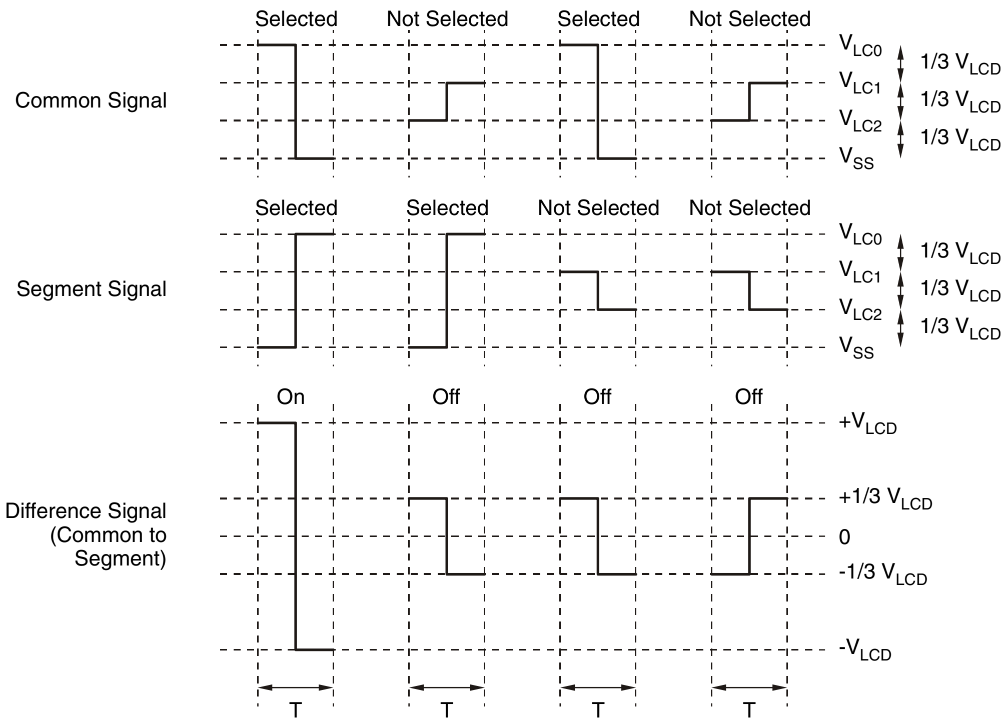
To drive larger LCD's the 1/2 bias method is more suitable, since it provides for both, the common and the segment signals independently selection and non-selection conditions as well. But only the condition is valid for the LCD's segment on-state, where both signals have got the selection-state. Due to this, one segment signal can control two (or three) segments on the LCD by two (or three) common lines. This reduces the required number of control lines dramatically. The resulting difference signals are shown in Figure 2-3
Because only the common signals have to drive the 1/2 bias voltage in case of non-selection condition, the required driver functionality can be kept simple. The drawback of the 1/2 bias method is that the used LCD must support this bias, since the on-off-voltage ratio is no more so high as it was for the static display mode, and thus the contrast is lower.

Figure 2-3. Difference Signals of 1/2 Bias Method



The 1/3 bias method, which provides the four independent states between common and segment signals too, can improve the on-off-voltage ratio.
As you can take from the following Figure 2-4, for this driving method both signals, the common as well as the segment signals must support a 1/3 bias voltage for their non-selection condition. And of course, the appropriated driver control might get more complex as for the 1/2 bias method.
Unnecessary to mention that the used LCD must be design for 1/3 bias voltage.

Figure 2-4. Difference Signals of 1/3 Bias Method



2.2 Emulation of Bias Voltage

A dedicated LCD controller generates the necessary biases by a resistor ladder. Over analog switches controlled by a multiplexing unit the corresponding segment and common lines are driven with appropriated biases.
The V850/SA1, as well as other micro-controllers, does not incorporate an integrated LCD controller. Hence not only the multiplexing, but also the bias control has to be done by software and digital I/O ports.

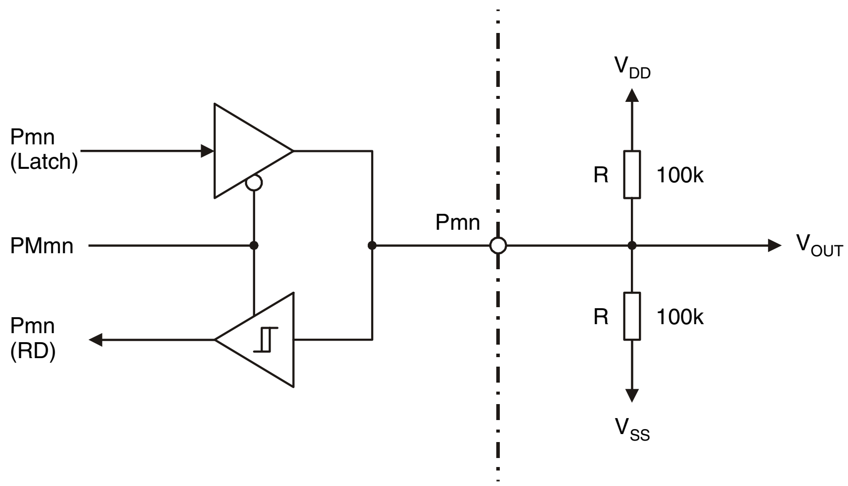
To keep the interface simple, it is supposed that the LCD drive voltage level V_{LCD} has to be equal to the power supply voltage V_{DD}.

A bias generation, unlike than static mode, is not quite easy to achieve with standard digital I/O ports. Therefore some additional external circuitry is necessary.

2.2.1 1/2 Bias voltage by single I/O port

Although a bias of 1/2 V_{DD} means three output states (V_{DD}, 1/2 V_{DD}, V_{SS}) this can be generated by one digital I/O only as shown in figure Figure 2-5

Figure 2-5. I/O Circuitry for 1/2 Bias Method



With the circuitry above and the settings in the following Table 2-3, the required voltages for 1/2 bias method can be achieved. The very low power consumption of LCD's has the favorable effect of low dependency of output voltage at the port pin, when input mode is selected and just the pull-up/pull-down resistors are effective. Thus the pull-up/pull-down resistors can be chosen relatively high, and this limits the total power consumption.

Table 2-3. Port Settings for 1/2 Bias Method

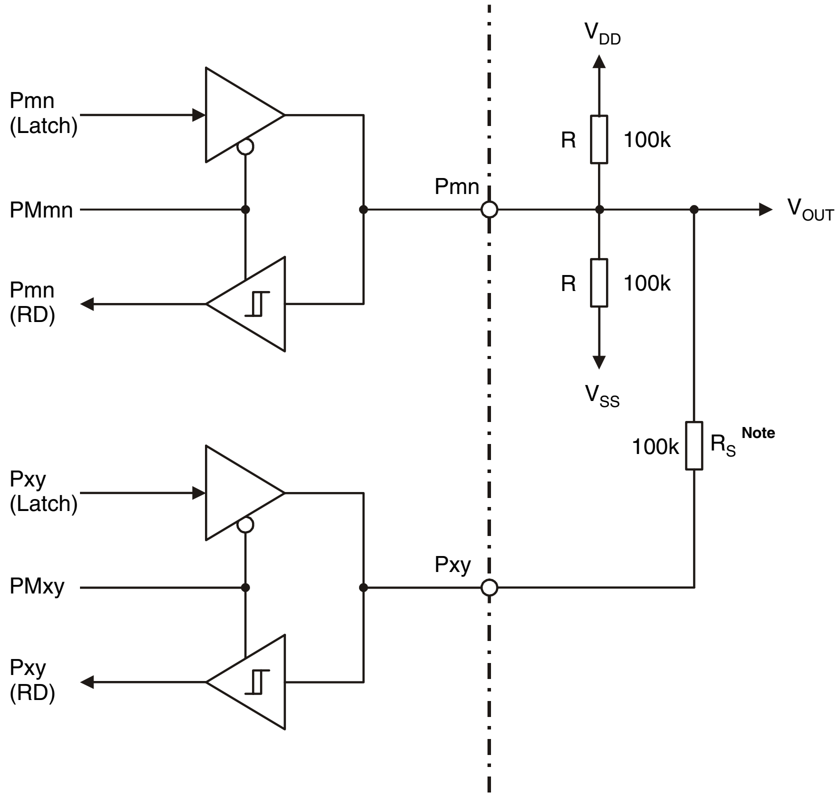
Port Mode PMmn	Port Output Latch Pmn	V_{OUT}
0 (Output)	0	V_{SS}
	1	V_{DD}
1 (Input)	x	1/2 V_{DD}

Caution **When CMOS input ports are operating in the linear range, the leakage current of these ports increase dramatically and thus the power consumption. An input voltage of 1/2 V_{DD} is certainly in the linear range, since this is given by the maximum low input voltage $V_{IL(max)}$ and the minimum high-level input voltage $V_{IH(min)}$.**
To avoid this behavior it is recommended to use only input ports with threshold functionality for the 1/2 bias method.

2.2.2 1/3 Bias voltage by two I/O ports

By expanding the I/O circuitry in Figure 2-5additional output states are possible. With an additional digital I/O port and a serial resistor an I/O circuitry for the 1/3 bias method can be made. This is shown in Figure 2-6

Figure 2-6. I/O Circuitry for 1/3 Bias Method



Note The indicated value of the serial resistor R_S at port Pxy is only valid, if high-level output voltage $V_{QH} = V_{DD}$, and low-level output voltage $V_{QL} = V_{SS}$.
However, the resistor can be adapted easily with the following formula:

$$R_S = \frac{1}{1-(3 \cdot \Delta_Q)} \cdot R$$

provided that the deviation from the nominal output voltage is equal for V_{QH} and V_{QL}

$$V_{QH} = V_{DD} - \Delta_Q \cdot V_{DD}$$

$$V_{QL} = V_{SS} + \Delta_Q \cdot V_{DD}$$

With the circuitry above, note the restriction of output voltage at port Pxy, and the settings in the following Table 2-4, the required output voltages V_{OUT} for 1/3 bias method can be generated.

Table 2-4. Port Settings for 1/3 Bias Method

Port Mode PMmn	Port Output Latch Pmn	Port Mode PMxy	Port Output Latch Pxy	V_{OUT}
0 (Output)	0	1 (Input)	x	V_{SS}
	1		x	V_{DD}
1 (Input)	x	0 (Output)	0	1/3 V_{DD}
	x		1	2/3 V_{DD}

Caution **Due to same reasons as before the digital I/O port Pmn has to be a port with threshold input.**

Remark Unlike port Pmn, the port Pxy doesn't need to be a port with threshold input, since this port is not operated in the linear range when input.
Just if both ports would become inputs, both ports must have an input hysteresis. But this state is not required in case of 1/3 bias method.

2.2.3 Considerations on bias emulation by V850/SA1

The major restriction of connecting a LCD glass panel to the V850/SA1 is the identical power supply for the micro-controller and the LCD, this means $V_{LCD} = V_{DD}$ = 3.0 to 3.6 V.

Further the expenditure of signal generation for the 1/3 bias method is obviously more extensive than for the 1/2 bias method. Moreover each control line (means common lines as well as segment lines) has to be driven with 1/3 bias voltage. Since V850/SA1 incorporates just 24 ports with input hysteresis capability, only a 1/3 bias LCD glass panel with 21 segment lines and 3 common lines could be connected. However, the most 1/3 bias LCD glass panels have more than 21 segment lines.
That's why the following example treat of the 1/2 bias method only.

On the other hand the static display mode is quite easy to implement with any of the I/O ports, and thus the example will not deal with it either.

In the following Table 2-5 for all V850/SA1 ports the operability as segment and common line for 1/2 bias method is shown. For use as segment line the port must have output capability. Additionally an input hysteresis (Schmitt-trigger functionality) is necessary for the I/O port used as common line. Ports with input-only functionality cannot be used for LCD control emulation.

Table 2-5. Segment and Common Line Operation of V850/SA1 Ports for 1/2 Bias Method

<table>
<tr><th rowspan="2">Port</th><th rowspan="2">Pin</th><th colspan="2">Operating as</th></tr>
<tr><th>Segment Line</th><th>Common Line</th></tr>
<tr><td>Port 0</td><td>P00
to
P07</td><td>yes</td><td>yes[Note]</td></tr>
<tr><td rowspan="6">Port 1</td><td>P10</td><td>yes</td><td>yes[Note]</td></tr>
<tr><td>P11</td><td>yes</td><td></td></tr>
<tr><td>P12</td><td rowspan="2">yes</td><td rowspan="2">yes[Note]</td></tr>
<tr><td>P13</td></tr>
<tr><td>P14</td><td>yes</td><td>no</td></tr>
<tr><td>P15</td><td>yes</td><td>yes[Note]</td></tr>
<tr><td rowspan="8">Port 2</td><td>P20</td><td>yes</td><td>yes[Note]</td></tr>
<tr><td>P21</td><td>yes</td><td>no</td></tr>
<tr><td>P22</td><td rowspan="2">yes</td><td rowspan="2">yes[Note]</td></tr>
<tr><td>P23</td></tr>
<tr><td>P24</td><td>yes</td><td>no</td></tr>
<tr><td>P25</td><td rowspan="3">yes</td><td rowspan="3">yes[Note]</td></tr>
<tr><td>P26</td></tr>
<tr><td>P27</td></tr>
<tr><td rowspan="5">Port 3</td><td>P30
to
P33</td><td>yes</td><td>yes[Note]</td></tr>
<tr><td>P34</td><td rowspan="2">yes</td><td rowspan="2">no</td></tr>
<tr><td>P35</td></tr>
<tr><td>P36</td><td rowspan="2">yes</td><td rowspan="2">yes[Note]</td></tr>
<tr><td>P37</td></tr>
<tr><td>Port 4</td><td>P40
to
P47</td><td>yes</td><td>no</td></tr>
<tr><td>Port 5</td><td>P50
to
P57</td><td>yes</td><td>no</td></tr>
<tr><td>P6</td><td>P60
to
P65</td><td>yes</td><td>no</td></tr>
<tr><td>P7</td><td>P70
to
P77</td><td>no</td><td>no</td></tr>
<tr><td>P8</td><td>P80
to
P83</td><td>no</td><td>no</td></tr>
</table>

Table 2-5. Segment and Common Line Operation of V850/SA1 Ports for 1/2 Bias Method

Port	Pin	Operating as	
		Segment Line	Common Line
P9	P90 to P96	yes	no
P10	P100 to P107	yes	no
P11	P110 to P113	yes	no
	P114	no	no
P12	P120	yes	no

Note Because of the tolerance width of internal pull-up resistors do not use them for 1/2 bias voltage generation.

CHAPTER 3   APPLICATION EXAMPLE

3.1 Control of a 10-digits LCD Glass Panel

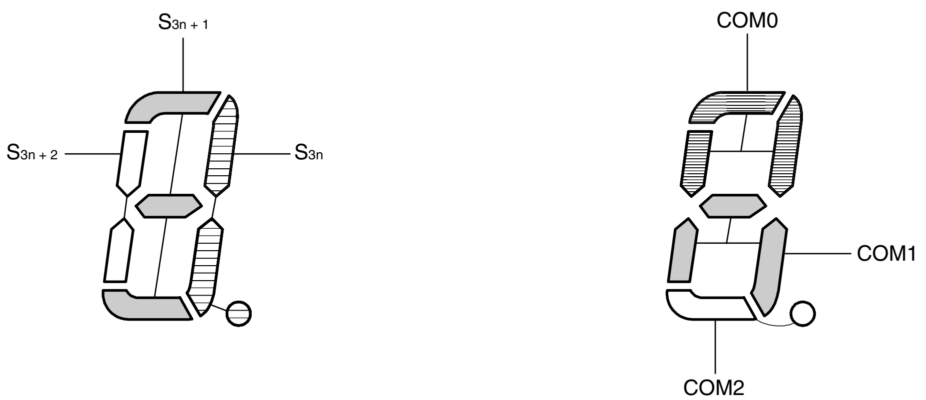
In this example a 10-digits LCD glass panel is connected to the V850/SA1. As mentioned before the LCD glass panel must fulfil the condition $V_{LCD} = V_{DD} = 3V$, where $V_{SS} = 0V$.

Since a 10-digits LCD with 7-segments plus a dot for each digit results in 80 segments, a static driven display would be quite inefficient. Therefore such panels are mostly offered for triplex mode (1/3 duty). However, since 8 cannot be divided by 3 with an integer result, 9 segments are reserved for each digit. Thus for a 10-digits LCD glass panel three common lines and 30 segment lines are required (see 2.1.2 Multiplex control).

Furthermore The used LCD glass panel has to be applicable for 1/2 bias control (see 2.2.3 Considerations on bias emulation by V850/SA1). In the specific case of a 10-digits LCD glass panel 33 port pins with input hysteresis and additionally 33 standard port pins would be necessary for 1/3 bias control - too many for the V850/SA1, where maximum 24 I/O ports with input hysteresis are available.

The configuration of electrodes (segment and common lines) of each digit is shown in Figure 3-1below.

Figure 3-1. LCD Display Pattern and Electrode Connections for Triplex Mode



An explanation is given here taking the example of a digit output of <6.>. In accordance with the display pattern in Figure 3-1, selection and non-selection voltages must be output to pins S_n to S_{n+2} as shown in Table 3-1 at the COM0 to COM2 common signal timings.

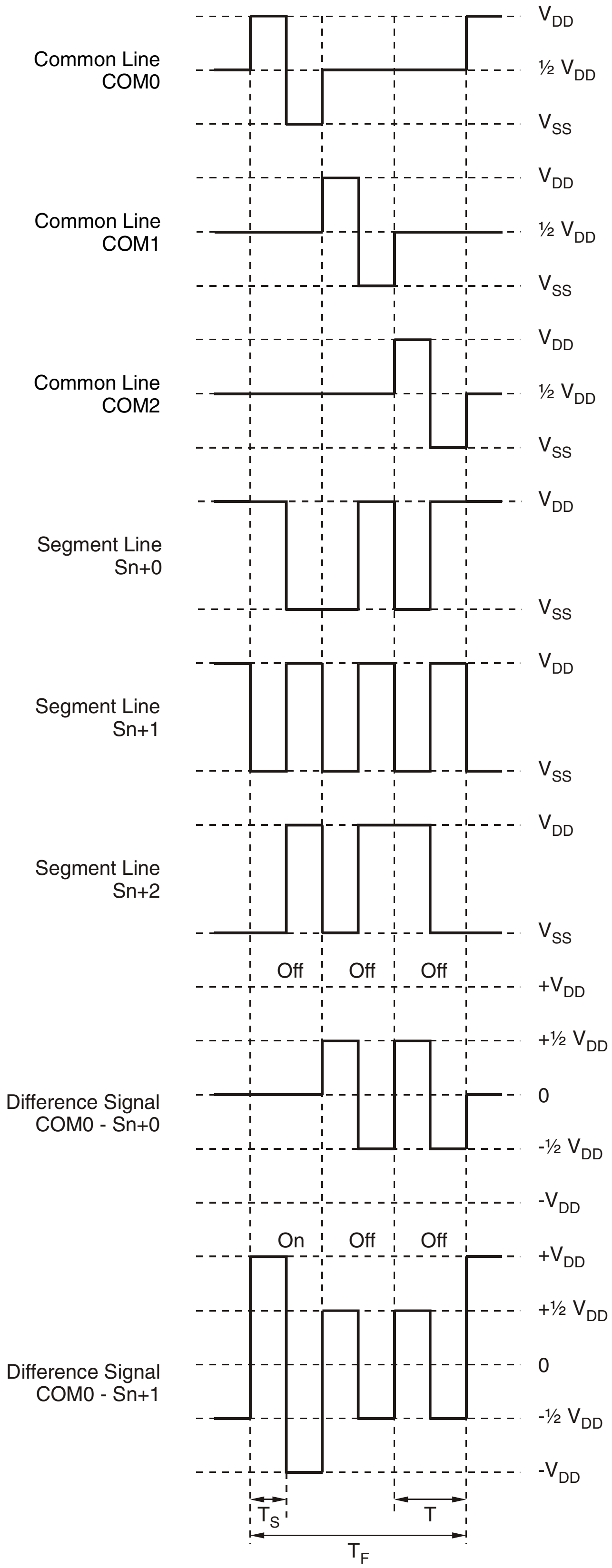
Table 3-1. Selection and Non-Selection Voltages (COM0 to COM2)

Common	Segment		
	Sn	Sn+1	Sn+2
COM0	NS	S	S
COM1	S	S	S
COM2	S	S	—

S: Selection, NS: Non-Selection

In the following timing diagram (Figure 3-2) the difference signals of (COM0 - Sn) and (COM1 - Sn+1) are represented according to the given example in Table 3-1.

Figure 3-2. Segment and Common Lines Drive Waveform Example



T : LCD clock cycle

T_S : Frame state cycle

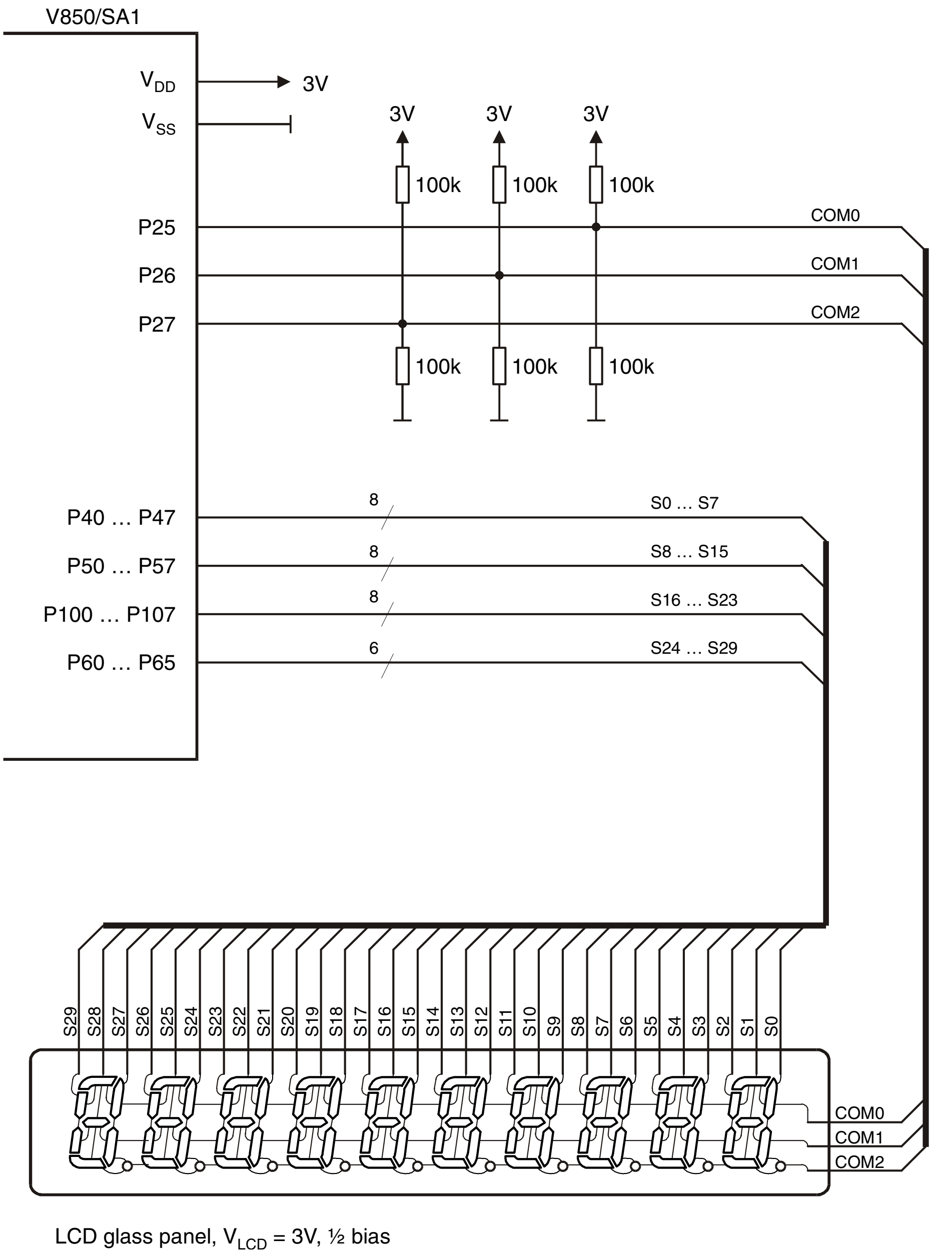
T_F : Frame period

For 1/2 bias control only the common lines have to drive a 1/2 bias voltage (see Figure 2-3). In the example the three common lines are realized by port pins P25 to P27, which are provided with input hysteresis capability.

The segment lines, in case of 1/2 bias mode, have to drive just a low- or high-level output voltage. This can be done with any port, whose mode can be set to output. In the example the port pins P40 to P47, P50 to P57, P100 to P107, and P60 to P65 are selected for segments control. All these ports do not provide an input hysteresis, but can be set to output mode.

The examples diagram is shown in Figure 3-3.

Figure 3-3. 10-digits LCD glass panel (1/3 duty, 1/2 bias) controlled by V850/SA1



3.2 Software Realization

Required resources for the example:

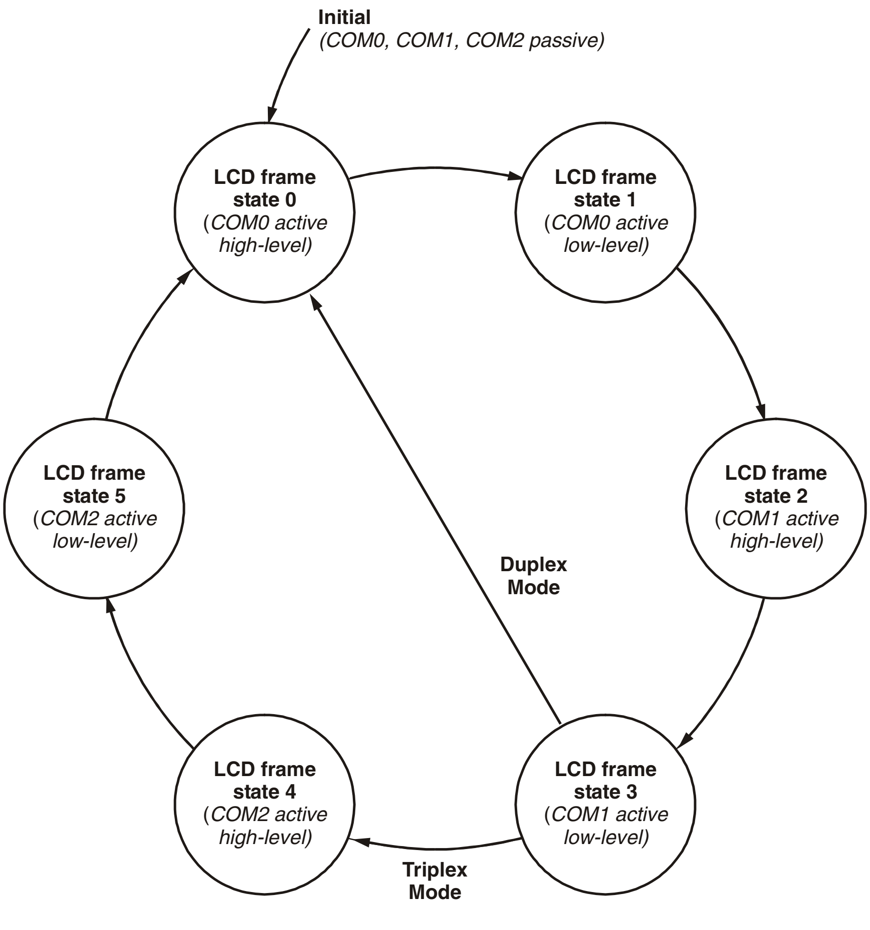
- P4, P5, P10, P6 — 4 ports with 30 port pins for the segment lines
- P25 to P27 — 3 port pins with input hysteresis for the common lines
- TM2 — 8-bit timer which operates as interval timer
- INTTM2 — Interrupt entry for the interval timer interrupt

The example code provides a few defines with which an easy adaptation for duplex mode as well as the number of 7-segment LCD digits can be made (LCD_DUTY_CYCLES, LCD_SEGMENT_LINES, LCD_DIGIT_NUM).

A quite important adaptation has be done by the defines of the internal system frequency (SYSTEM_FREQ) and the LCD frame frequency (LCD_FRAME_FREQ). Further the required 8-bit timer can be selected (TM_CHANNEL) between 2 and 5 (TM2 to TM5).

The kernel of the program is the interrupt service routine of the interval timer where the segment and common lines are switched, called **intr_lcd_timer**. Since two alternating states for each duty cycle are required, the interrupt service routine is realized as a state machine with six states in total for the triplex mode (1/3 duty) of the used LCD glass panel. As mentioned above, a duplex mode control (1/2 duty) is also possible by adaptation (LCD_DUTY_CYCLES). The state transition diagram of the LCD timer interrupt service routine is shown in Figure 3-4.

Figure 3-4. State Transition Diagram of the LCD Timer Interrupt Service Routine



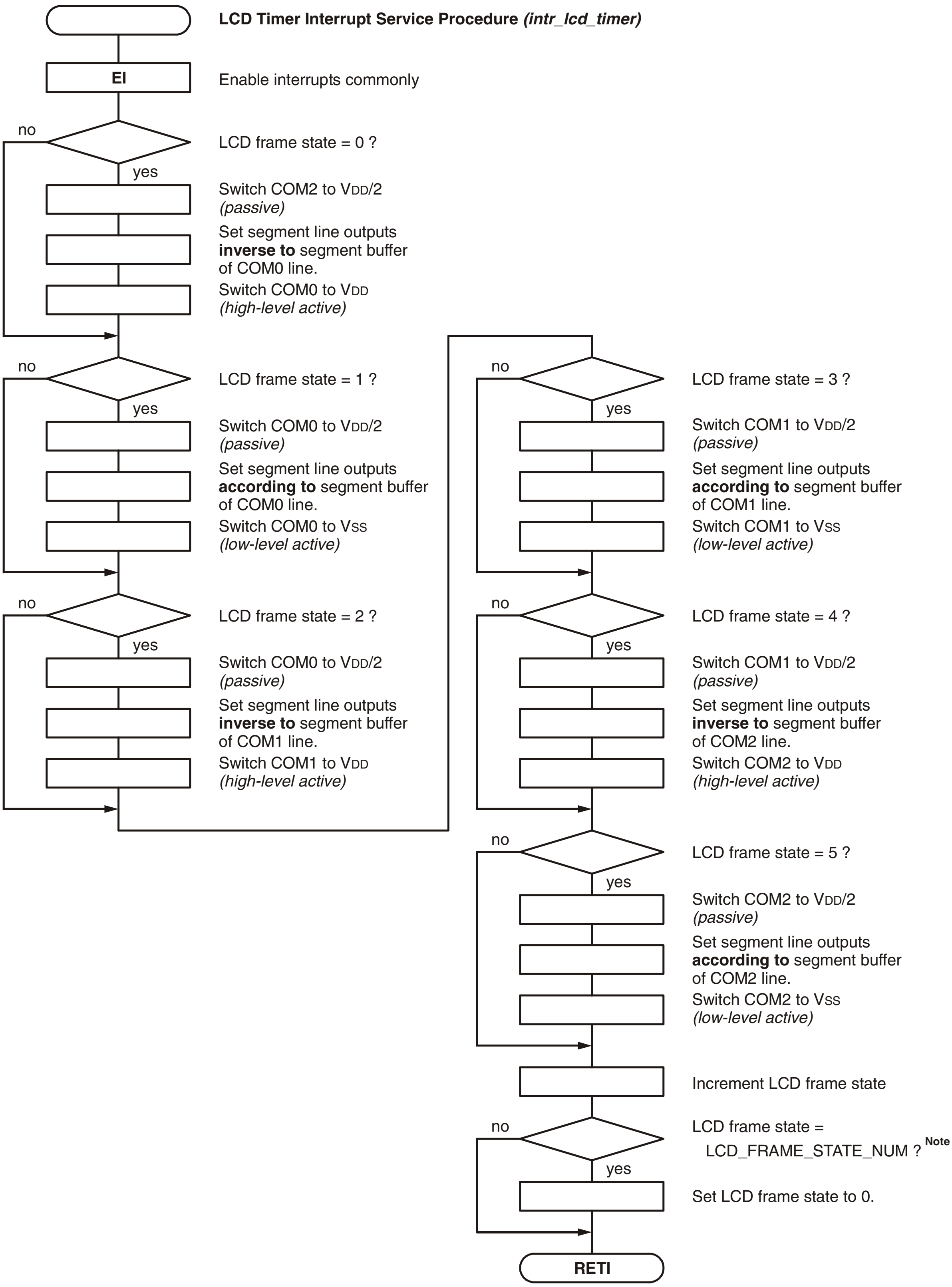
Remark Please note that for each LCD frame state only the mentioned common line is active. All other common lines are set to the passive state ($V_{DD}/2$).

For the segment lines the patterns are buffered in a segment port buffer, called **segment_port**, which contains the present segment port settings for each duty cycle (COM0 to COM2). Moreover, it exists a inverse segment port buffer, called **segment_port_inv**, which contains exactly the inverse segment port settings for the alternate states. With this the interrupt load will decrease by reducing the operating time within the interrupt service routine. Of course, the alternate states of segment port setting could be get from **segment_port** buffer too. The flow chart of the LCD timer interrupt service routine is shown in Figure 3-5.

Additionally the program module contains a procedure to output a number string onto the LCD. In this procedure the segment port buffers are initialized accordingly to the contents of the number string parameter, which may contain any of the alphanumeric characters <0> to <9>, <a> to <f>, <A> to <F> as well as <-> (minus) and <.> (decimal point). The decimal point however will not use a separate character space on the LCD, but the special decimal point segment of a 7-segment LCD digit.

Within the main procedure of the example source code the LCD (as well as the timer interrupt) is initialized and a numeric string is displayed, and continuously rotating left on the LCD.

Figure 3-5. Flow Chart of LCD Timer Interrupt Service Routine



Note The value of LCD_FRAME_STATE_NUM depends on the define of LCD_DUTY_CYCLES. In case of triplex mode this means 6, in case of duplex mode 4 frame states.

3.3 Source Code

In the following the example C source code is represented. It consists of the modules:

- LCD_SA1.C — C module with the functions main (for test purpose), init_lcd, intr_lcd_timer, and lcd_outp
- STARTUP.850 — Assembler module for the controller start-up

Besides the header file V850_SA1.H with the I/O definitions is used.

The sources have been compiled and tested with Green Hills Multi version 1.8.9. and are obtainable at NEC Electronics (Europe) GmbH, Technical Product Support.

(1) C module LCD_SA1.C

```
/*******************************************************************
 *               LCD driver emulation with V850/SA1                *
 *                 for 10-digits LCD glass panel                   *
 *-----------------------------------------------------------------*
 *  LCD Type:    3V LCD (fixed)                                    *
 *  Bias:        1/2 bias voltage (fixed)                          *
 *  Mutiplex:    1/2 duty (duplex) or 1/3 (triplex/default)        *
 *  Frame Freq.: 100 Hz (variable)                                 *
 *                                                                 *
 *            (C) 2000, NEC Electronics (Europe) GmbH              *
 *-----------------------------------------------------------------*
 * 000131TH - release                                              *
 *******************************************************************/

#include "v850sa1.h"  // I/O definitions of V850/SA1

//--------------------------------------------------------------------
// Defines
//--------------------------------------------------------------------

// These entries may be changed by the user for adaption purpose.
//
#define SYSTEM_FREQ       17000000 // Main system clock frequency [Hz]
                                   // fcpu = 17MHz is the max. system
                                   // freq. of V850/SA1
#define LCD_FRAME_FREQ    100      // The frame freq. is repetition
                                   // rate of segments signal control.
#define LCD_DUTY_CYCLES   3        // duty cycles = 2 for duplex
                                   //             = 3 for triplex
#define LCD_SEGMENT_LINES 30       // Number of segment lines

#define LCD_DIGIT_NUM     10       // Number of max. displayed digits

#define TM_CHANNEL        2        // Used timer channel (2/3/4/5)


// Port definitions for common and segment lines.
// It is assumed that the segment ports consist of 8-bit ports each,
// apart from the last, which may have less than 8 bits.
#define SEG_PORT_NUM    ((LCD_SEGMENT_LINES+7)/8)

#define COM0_PORT  _P25
#define COM1_PORT  _P26
#define COM2_PORT  _P27
```

```
#define COM0_PMODE _PM25  // common line 0
#define COM1_PMODE _PM26  // common line 1
#define COM2_PMODE _PM27  // common line 2

#define SEG_PORT0  P4    // 8 segment lines S0-S7
#define SEG_PORT1  P5    // 8 "             S8-S15
#define SEG_PORT2  P10   // 8 "             S16-S23
#define SEG_PORT3  P6    // 6 "             S24-S29

#define SEG_PMODE0 PM4
#define SEG_PMODE1 PM5
#define SEG_PMODE2 PM10
#define SEG_PMODE3 PM6


// The following entries are calculated and checked during compilation
// time and should not be changed by the user!
//
#define LCD_FRAME_STATE_NUM     (2*LCD_DUTY_CYCLES)
#define TM_BASE_FREQ\
        (SYSTEM_FREQ/(LCD_FRAME_FREQ*LCD_FRAME_STATE_NUM))

#if (TM_BASE_FREQ/4 <= 256)
#define TM_CLOCK_DIV     4
#define TM_CL_INIT       0x02
#elif (TM_BASE_FREQ/8 <= 256)
#define TM_CLOCK_DIV     8
#define TM_CL_INIT       0x03
#elif (TM_BASE_FREQ/16 <= 256)
#define TM_CLOCK_DIV     16
#define TM_CL_INIT       0x04
#elif (TM_BASE_FREQ/32 <= 256)
#define TM_CLOCK_DIV     32
#define TM_CL_INIT       0x05
#elif (TM_BASE_FREQ/128 <= 256)
#define TM_CLOCK_DIV     128
#define TM_CL_INIT       0x06
#elif ((TM_BASE_FREQ/256 <= 256) && (TM_CHANNEL <= 3))
#define TM_CLOCK_DIV     256      // only possible for TM2 and TM3
#define TM_CL_INIT       0x07
#else
#error "Increase LCD_FRAME_FREQ or decrease SYSTEM_FREQ."
#endif
#define TM_CR_INIT       (TM_BASE_FREQ/TM_CLOCK_DIV)


//-------------------------------------------------------------------
// Global variables
//-------------------------------------------------------------------
// Variable, which holds the current LCD frame state.
static unsigned short  lcd_frame_state;

// Segment port buffer
static unsigned char  segment_port [LCD_DUTY_CYCLES][SEG_PORT_NUM];

// Inverse segment port buffer
// Remark: the inverse seg. port buffer reduces the operating time
// within the interrupt procedure.
static unsigned char  segment_port_inv [LCD_DUTY_CYCLES][SEG_PORT_NUM];


//-------------------------------------------------------------------
// Set common line x to input
```

```
//-------------------------------------------------------------------
#define set_com_input(x) \
  { COM##x##_PMODE = 1;\
  } // end of set_com_input


//-------------------------------------------------------------------
// Set common line x to output
//-------------------------------------------------------------------
#define set_com_output(x,level) \
  { COM##x##_PORT  = level;\
    COM##x##_PMODE = 0;\
  } // end of set_com_output

//-------------------------------------------------------------------
// Macros for setting timer sfr's/bit's
// "ch" must be in the range of 2 to 5.
//-------------------------------------------------------------------
#define write_TMIC(ch,val)  *((&TMIC2)+0x10*(ch-2)) = val;
#define write_TCL(ch,val)   *((&TCL2) +0x10*(ch-2)) = val;
#define write_TMC(ch,val)   *((&TMC2)+0x10*(ch-2)) = val;
#define write_CR(ch,val)    *((&CR20) +0x10*(ch-2)) = val;
#define start_timer(ch)     *((&TMC2)+0x10*(ch-2)) |= 0x80;

//-------------------------------------------------------------------
// Disable LCD timer interrupt.
//-------------------------------------------------------------------
#define intr_lcd_timer_disabled()\
  write_TMIC(TM_CHANNEL,0x47);


//-------------------------------------------------------------------
// Enable LCD timer interrupt.
//-------------------------------------------------------------------
#define intr_lcd_timer_enabled()\
  write_TMIC(TM_CHANNEL,0x07);


//-------------------------------------------------------------------
// FUNCTION:    init_lcd
// PURPOSE:
//      Initialize LCD controller emulation interface.
// PARAMETER:   none
// RETURNS:     none
//-------------------------------------------------------------------
void  init_lcd (void)
{
   unsigned short i, j;

   lcd_frame_state = 0;

   // Init common line ports:
   // all COM lines will set to VDD/2
   //
   set_com_input(0);
   set_com_input(1);
   set_com_input(2);

   // init segment line ports
   //
   SEG_PORT0 = 0;
   SEG_PORT1 = 0;
   SEG_PORT2 = 0;
```

```
   SEG_PORT3 = 0;

   SEG_PMODE0 = 0;
   SEG_PMODE1 = 0;
   SEG_PMODE2 = 0;
   SEG_PMODE3 = 0;

   // init lcd segment buffers:
   // a segment is off, when the corresponding bit is cleared
   // in the segment_port array and set in the segment_port_inv array
   // (inversion of segment_port array).
   for (i=0; i<3; i++)
     {
     for (j=0; j<5; j++)
       {
       segment_port [i][j] = 0x00;
       segment_port_inv [i][j] = ~segment_port[i][j];
       }
     }

  // LCD timer initialization for frame timing
  write_TMIC(TM_CHANNEL, 0x07);      // timer interrupt enabled,
                                     // low prio
  write_TCL(TM_CHANNEL, TM_CL_INIT); // set prescaler\
  write_TMC(TM_CHANNEL, 0x00);       // interval mode\
  write_CR(TM_CHANNEL, TM_CR_INIT);  // init compare register\
  start_timer(TM_CHANNEL);

} // end of init_lcd


//---------------------------------------------------------------------
// FUNCTION:    intr_lcd_timer
// PURPOSE:
//      Serves the LCD timer interrupt where the different LCD states
//      are perforemd by a finite state machine.
// PARAMETER:   none
// RETURNS:     none
//---------------------------------------------------------------------
void intr_lcd_timer (void)
{
  #pragma ghs interrupt
  EI();
  if( lcd_frame_state == 0 ) {
    set_com_input(2);           // switch COM2 to VDD/2
    SEG_PORT0 = segment_port_inv[0][0];
    SEG_PORT1 = segment_port_inv[0][1];
    SEG_PORT2 = segment_port_inv[0][2];
    SEG_PORT3 = segment_port_inv[0][3];
    set_com_output(0,1);        // switch COM0 to VDD
  }
  else if( lcd_frame_state == 1 ) {
    set_com_input(0);           // switch COM0 to VDD/2
    SEG_PORT0 = segment_port[0][0];
    SEG_PORT1 = segment_port[0][1];
    SEG_PORT2 = segment_port[0][2];
    SEG_PORT3 = segment_port[0][3];
    set_com_output(0,0);        // switch COM0 to VSS
  }
  else if( lcd_frame_state == 2 ) {
    set_com_input(0);           // switch COM0 to VDD/2
    SEG_PORT0 = segment_port_inv[1][0];
    SEG_PORT1 = segment_port_inv[1][1];
```

```
    SEG_PORT2 = segment_port_inv[1][2];
    SEG_PORT3 = segment_port_inv[1][3];
    set_com_output(1,1);        // switch COM1 to VDD
  }
  else if( lcd_frame_state == 3 ) {
    set_com_input(1);           // switch COM1 to VDD/2
    SEG_PORT0 = segment_port[1][0];
    SEG_PORT1 = segment_port[1][1];
    SEG_PORT2 = segment_port[1][2];
    SEG_PORT3 = segment_port[1][3];
    set_com_output(1,0);        // switch COM1 to VSS
  }
  else if( lcd_frame_state == 4 ) {
    set_com_input(1);           // switch COM1 to VDD/2
    SEG_PORT0 = segment_port_inv[2][0];
    SEG_PORT1 = segment_port_inv[2][1];
    SEG_PORT2 = segment_port_inv[2][2];
    SEG_PORT3 = segment_port_inv[2][3];
    set_com_output(2,1);        // switch COM2 to VDD
  }
  else if( lcd_frame_state == 5 ) {
    set_com_input(2);           // switch COM2 to VDD/2
    SEG_PORT0 = segment_port[2][0];
    SEG_PORT1 = segment_port[2][1];
    SEG_PORT2 = segment_port[2][2];
    SEG_PORT3 = segment_port[2][3];
    set_com_output(2,0);        // switch COM2 to VSS
  }

  lcd_frame_state++;
  if( lcd_frame_state == LCD_FRAME_STATE_NUM ) {
    lcd_frame_state = 0;
  }

} // end of intr_lcd_timer


//---------------------------------------------------------------------
// FUNCTION:    lcd_outp
// DESCRIPTION:
//      Outputs an alphanumeric string on the LCD.
//      The alphanumeric characters '0' to '9', 'a' to 'f', 'A to 'F',
//      and '-' as well as '.' can be output. The decimal point '.'
//      counts not as an separate character, so that it doesn't affect
//      the maximum number of digits.
//      Remark: The output result of this function depends on the
//              assignment of LCD segments to the segment and common
//              lines and will commonly differ when using different
//              LCD's.
//      Hint:   The string can be pepared by the standard C function
//              'sprintf', where the decimal outputs for integer as
//              well as for floating point variables are possible.
// PARAMETER:
//      char    *s
//              A null-terminated string with displayable charcters.
//              Non-displayable characters are ignored and a <blank>
//              is output instead. The maximum number of characters,
//              which will be displayed is defined by LCD_DIGIT_NUM.
//              If the string contains more characters (decimal points
//              don't count) the rest of the string is ignored.
// RETURNS:     none
//---------------------------------------------------------------------
void  lcd_outp ( char *s )
```

```
{
  #if (LCD_DUTY_CYCLES == 2)
  static unsigned char seg7_lines = 4;
  static unsigned char seg7_mask  = 0x0f;
  static unsigned char seg7_dp[LCD_DUTY_CYCLES] =
      {0x00, 0x01}; // '.'
  static unsigned char seg7_minus[LCD_DUTY_CYCLES] =
      {0x00, 0x02}; // '-'
  static unsigned char seg7_digit[10][LCD_DUTY_CYCLES] =
    { {0x0f, 0x0c}, // '0'
      {0x03, 0x00}, // '1'
      {0x0a, 0x0e}, // '2'
      {0x03, 0x0e}, // '3'
      {0x07, 0x04}, // '4'
      {0x05, 0x0e}, // '5'
      {0x0d, 0x0e}, // '6'
      {0x03, 0x04}, // '7'
      {0x0f, 0x0e}, // '8'
      {0x07, 0x0e}, // '9'
    };
  static unsigned char seg7_char[6][LCD_DUTY_CYCLES] =
    { {0x0f, 0x06}, // 'A'
      {0x0d, 0x0a}, // 'b'
      {0x08, 0x0a}, // 'c'
      {0x0b, 0x0a}, // 'd'
      {0x0c, 0x0e}, // 'E'
      {0x0c, 0x06}, // 'F'
    };
  #elif (LCD_DUTY_CYCLES == 3)
  static unsigned char seg7_lines = 3;
  static unsigned char seg7_mask  = 0x07;
  static unsigned char seg7_dp[LCD_DUTY_CYCLES] =
      {0x00, 0x00, 0x01}; // '.'
  static unsigned char seg7_minus[LCD_DUTY_CYCLES] =
      {0x00, 0x00, 0x01}; // '-'
  static unsigned char seg7_digit[10][LCD_DUTY_CYCLES] =
    { {0x07, 0x05, 0x02}, // '0'
      {0x01, 0x01, 0x00}, // '1'
      {0x03, 0x06, 0x02}, // '2'
      {0x03, 0x03, 0x02}, // '3'
      {0x05, 0x03, 0x00}, // '4'
      {0x06, 0x03, 0x02}, // '5'
      {0x06, 0x07, 0x02}, // '6'
      {0x03, 0x01, 0x00}, // '7'
      {0x07, 0x07, 0x02}, // '8'
      {0x07, 0x03, 0x02}, // '9'
    };
  static unsigned char seg7_char[6][LCD_DUTY_CYCLES] =
    { {0x07, 0x07, 0x00}, // 'A'
      {0x04, 0x07, 0x02}, // 'b'
      {0x00, 0x06, 0x02}, // 'c'
      {0x01, 0x07, 0x02}, // 'd'
      {0x06, 0x06, 0x02}, // 'E'
      {0x06, 0x06, 0x00}, // 'F'
    };
  #else
  #error "Mode not implemented!"
  #endif

  unsigned char  com;
  unsigned char  port_idx, port_bit, port_mask;
  unsigned char  digit;
  unsigned char  digit_pos;
```

```
unsigned char  segment_idx;
unsigned char  seg7_pattern[LCD_DUTY_CYCLES];

// Initialize the digit position counter and the segment index.
digit_pos = strlen(s);
segment_idx = 0;

// Set LCD patterns for all digit characters of the string parameter
// but consider the maximum number of segment lines.
while( (digit_pos > 0)
     &&(segment_idx < LCD_DIGIT_NUM*seg7_lines)) {

  // Get the next digit character (start with LSD).
  digit_pos--;
  digit = s[digit_pos];

  // Initialize the segements pattern.
  for( com=0; com<LCD_DUTY_CYCLES; com++ ) {
    if( digit == '.' ) {
      seg7_pattern[com] = seg7_dp[com];
    }
    // ... otherwise blank the segment patterns.
    else {
      seg7_pattern[com] = 0x00;
    }
  }

  // If the digit character is a decimal point get
  // the next digit character ...
  if( digit == '.' ) {
    if( digit_pos > 0 ) {
      digit_pos--;
      digit = s[digit_pos];
    }
  }

  // Add the segments pattern for displayable characters ...
  for( com=0; com<LCD_DUTY_CYCLES; com++ ) {
    // when digits ...
    if((digit >= '0') && (digit <= '9')) {
      seg7_pattern[com] |= seg7_digit[digit-0x30][com];
    }
    // when hex characters ...
    else if ((digit >= 'a') && (digit <= 'f')) {
      seg7_pattern[com] |= seg7_char[digit-'a'][com];
    }
    else if ((digit >= 'A') && (digit <= 'F')) {
      seg7_pattern[com] |= seg7_char[digit-'A'][com];
    }
    // when minus character ...
    else if( digit == '-' ) {
      seg7_pattern[com] |= seg7_minus[com];
    }
    // ... otherwise no digit character is output (blank or dp).
  }

  // Write the digit pattern into the segment port buffer.
  port_idx = segment_idx >> 3;   // eq. to segment_idx/8
  port_bit = segment_idx & 0x07; // eq. to segment_idx%8
  port_mask = ~(seg7_mask<<(port_bit));
  for( com=0; com<LCD_DUTY_CYCLES; com++ ) {
    intr_lcd_timer_disabled();
    segment_port[com][port_idx] &= port_mask;
```

```
      segment_port[com][port_idx] |= seg7_pattern[com]<<(port_bit);
      segment_port_inv [com][port_idx] = ~segment_port[com][port_idx];
      intr_lcd_timer_enabled();
    }

    // If one or more pattern bits cannot be written completely into
    // the previous segment port buffer, the remaining pattern
    // will be written into the following.
    port_bit = 7 - port_bit;
    port_mask = ~(seg7_mask>>(port_bit));
    if( port_bit < seg7_lines ) {
      for( com=0; com<LCD_DUTY_CYCLES; com++ ) {
        intr_lcd_timer_disabled();
        segment_port[com][port_idx+1] &= port_mask;
        segment_port[com][port_idx+1] |= seg7_pattern[com]>>(port_bit);
        segment_port_inv [com][port_idx+1] = ~segment_port[com][port_idx+1];
        intr_lcd_timer_enabled();
      }
    }

    // Increment the segment index
    segment_idx += seg7_lines;
  }

} // end of lcd_outp


//----------------------------------------------------------------------
// M A I N
//----------------------------------------------------------------------
void  main (void)
{
  int   count;
  short num;
  char  numchar;
  char  numstr[] = "01234.56789";

  // Hardware initilazation
  init_lcd();
  EI();

  // Output a string on the LCD.
  lcd_outp(numstr);

  while( 1 ) {    // main loop ...
    // Following code is just for test purpose -
    // might be replaced by any useful application
    count++ ;
    if( count == 100000 ) {
      count = 0;
      // Rotate the digit characters left.
      numchar = numstr[0];
      for( num=0; num<strlen(numstr); num++ ) {
        numstr[num] = numstr[num+1];
      }
      numstr[num] = numchar;

      // Output the new string on the LCD.
      lcd_outp(numstr);
    }
  } // end of main loop
} // end of main
```

(2) Assembler module STARTUP.850

```
--**********************************************************************
--
-- STARTUP module for the LCD driver emulation with V850/SA1
--
-- (C) 2000, NEC Electronics (Europe) GmbH
-- 000131TH - release
--**********************************************************************

------------------------------------------------------------------------
-- Definition of control conditions
------------------------------------------------------------------------
-- Timer channel has to be set here addtionally to the define in LCD_SA1.C
TM_CHANNEL  =   2

------------------------------------------------------------------------
-- SFR defintions
------------------------------------------------------------------------
MM      = 0xfffff04c
DWC     = 0xfffff060
BCC     = 0xfffff062

------------------------------------------------------------------------
-- Declaration of external functions
------------------------------------------------------------------------
.extern _intr_lcd_timer

------------------------------------------------------------------------
-- Initializing of interrupt vectors
------------------------------------------------------------------------
.section    ".initvec",.text

.globl _BasicInitOfController

.org 0x0
    jr _BasicInitOfController

.if (TM_CHANNEL == 2)
.org 0x150
.elseif (TM_CHANNEL == 3)
.org 0x160
.elseif (TM_CHANNEL == 4)
.org 0x170
.elseif (TM_CHANNEL == 5)
.org 0x180
.endif
    jr _intr_lcd_timer

------------------------------------------------------------------------
-- Basic initialisation of the controller
------------------------------------------------------------------------
_BasicInitOfController:
     -- Initialisation of the global pointer
     movhi   hi(___ghsbegin_sdabase),zero,gp
     movea   lo(___ghsbegin_sdabase),gp,gp

     -- Initialisation of the stack pointer
     movhi   hi(___ghsend_stack-4),zero,sp
     movea   lo(___ghsend_stack-4),sp,sp

     -- Initialisation of the MM register
     mov     0x07, r10
     st.b    r10, MM[r0]
```

```
    -- Initialisation of the BCC register
    mov     0x0440, r10
    st.h    r10, BCC[r0]

    -- Initialisation of the DWC register
    mov     0x0440, r10
    st.h    r10, DWC[r0]

    -- Jump to the Initialisation functions of the library,
    -- from there to main()
    jr __start
---------------------------------------------------------------------------
```