SMART ARM-based Microcontrollers

AT09192: SAM L22 Segment Liquid Crystal Display (SLCD) Controller

APPLICATION NOTE

Introduction

This application note briefly describes the following features of the Segment Liquid Crystal Display (SLCD) Controller available on the Atmel® | SMART SAM L22 microcontrollers.

The software example mentioned in this document is available with the latest Atmel Software Framework (ASF).

For more details on SLCD module, refer SAM L22 datasheet.

Features

- Blink mode and frequency configuration (up to 16 segments)
- Regular and low power waveform
- Software contrast adjustment control
- Character mapping
- Automated characters string scrolling
- Automated bit mapping
- DMA support

Table of Contents

1. Glossary

ASF	Atmel Software Framework
ABM	Automated Bit Mapping
ACM	Automated Character Mapping
Atmel Studio	Integrated Development Environment (IDE) for Atmel Microcontrollers
CDC	USB Communication Device Class
CLK	Clock
COM	Common
CTRL	Control
DMAC	DMA (Direct Memory Access) Controller
EDBG	Embedded Debugger
EVSYS	Event System
GPIO	General Purpose Input/Output
IDE	Integrated Development Environment
NVIC	Nested Vectored Interrupt Controller
PM	Power Manager
SERCOM	Serial Communication Interface
SEG	Segment
SLCD	Segmented Liquid Crystal Display
SRAM	Static Random-Access Memory
USART	Universal Synchronous and Asynchronous Serial Receiver and Transmitter

2. Pre-requisites

The solutions discussed in this document require basic familiarity with the following tools:

- Atmel Studio 6.2 SP2 or later versions
- ASF version 3.26.10 or later versions
- SAM L22 Xplained Pro Evaluation Kit with USB cable
- Segment LCD1 Xplained Pro

Note:
Make sure that the SAM L22 Part Pack for Atmel Studio is installed before using the example application.

This application note covers the overview of the following peripherals:

- SERCOM – USART
- DMAC

Refer to the product datasheet for better understanding of each of the peripherals.

3. Setup

The application is developed for SAM L22 Xplained Pro board in Atmel Studio 6.2. This chapter covers hardware and software setup needed to test this application.

3.1. Hardware Setup

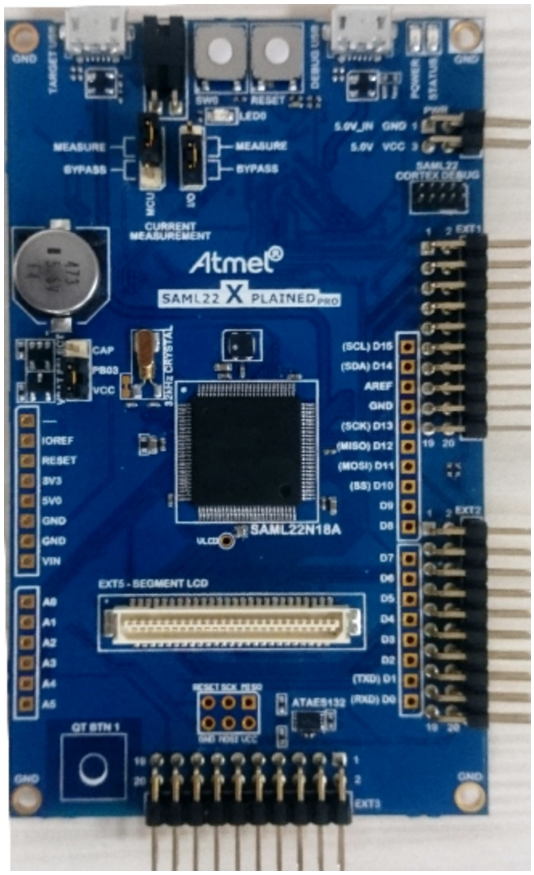
3.1.1. SAM L22 Xplained Pro

The Atmel SAM L22 Xplained Pro evaluation kit is a hardware platform to evaluate the ATSAML22N18A microcontroller. The SAM L22 Xplained Pro kit will be used to run the example application.

This evaluation kit allows connecting multiple external components via a wing connector. A wing is a self-contained board that can be connected to the Xplained Pro using a wing connector. The SAM L22 Xplained Pro has four such wing connectors marked as EXT1, EXT2, EXT3, and EXT4.

To explore more about SAM L22 Xplained Pro hardware, refer the hardware user guide and schematics.

Figure 3-1 SAM L22 Xplained Pro



3.1.2. Segment LCD1 Xplained Pro Board

Atmel Segment LCD1 Xplained Pro extension board is a small circuit board with a custom backlit segment LCD display, which is compatible with Xplained Pro MCU boards with a segment LCD connector. Segment LCD1 Xplained Pro requires four common terminals and segment terminal 0 to 23 to control all the segments.

Figure 3-2 Segment LCD1 Xplained Pro Top Overview

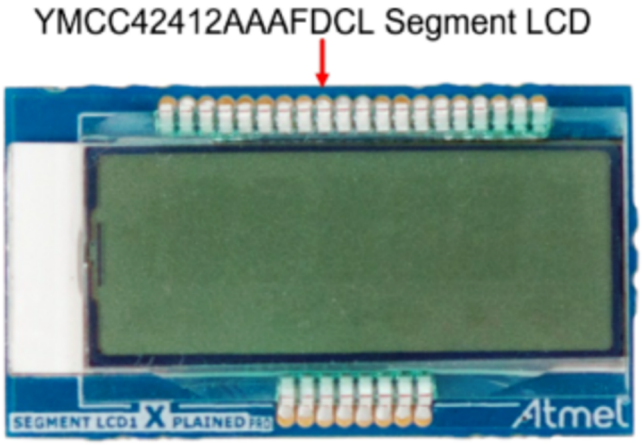
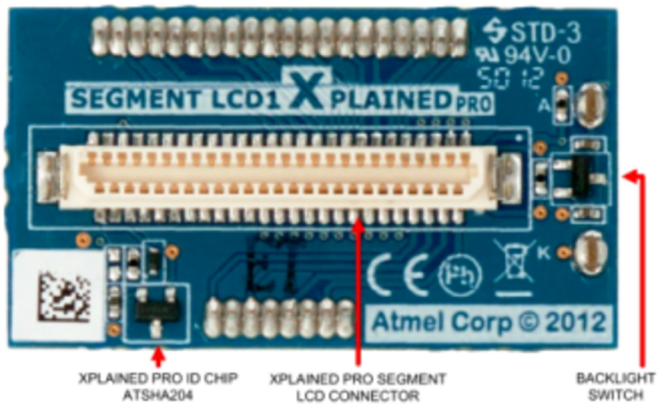


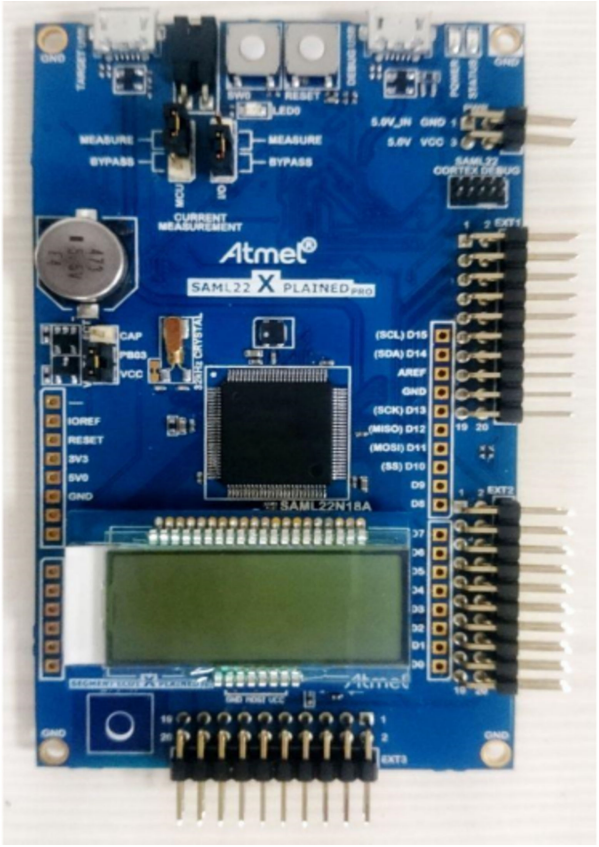
Figure 3-3 Segment LCD1 Xplained Pro Bottom Overview



Segment LCD1 Xplained Pro has been designed to be used with Xplained Pro MCU boards that contains a Segment LCD connector.

In SAM L22 Xplained Pro kit, it is connected to EXT5 header as shown in Figure 3-4 SAM L22 Xplained Pro with Segment LCD1 Xplained Pro Board Connection on page 7. This application utilizes custom backlit segment LCD display on Segment LCD1 Xplained Pro board

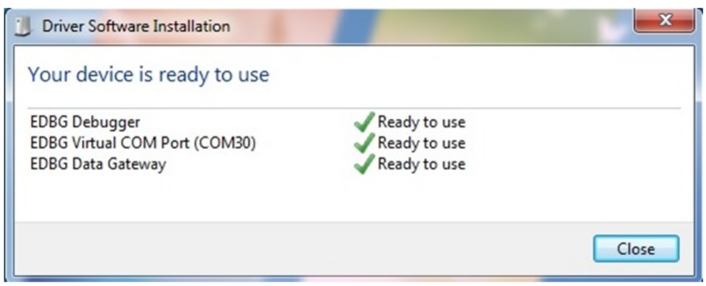
Figure 3-4 SAM L22 Xplained Pro with Segment LCD1 Xplained Pro Board Connection



3.2. Software Setup

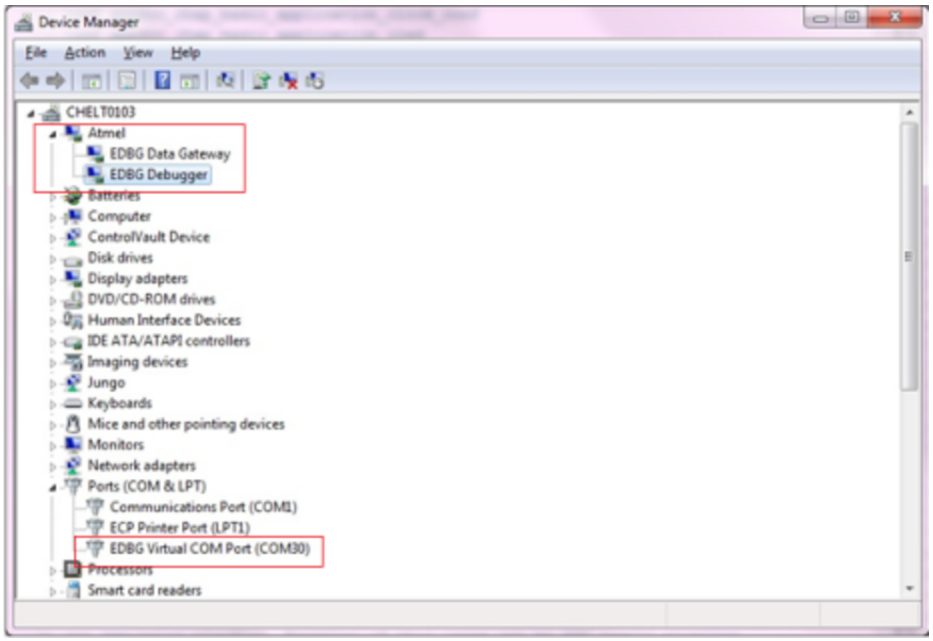
There are two USB ports on the SAM L22 Xplained Pro board; **DEBUG USB** and **TARGET USB**. For debugging using the Embedded debugger (EDBG), the **DEBUG USB** port has to be connected. Once the SAM L22 Xplained Pro kit is connected to the PC, the Windows® task bar will pop up a message as shown in the figure below.

Figure 3-5 SAM L22 Xplained Pro Driver Installation



If the driver installation is proper, EDBG will be listed in the Device Manager as shown in Figure 3-6 Successful EDBG Driver Installation on page 8.

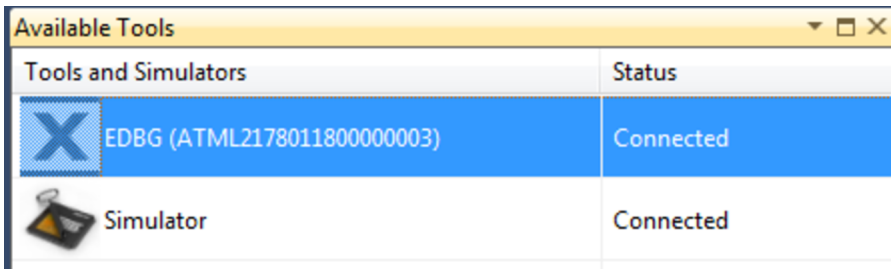
Figure 3-6 Successful EDBG Driver Installation



To ensure that the EDBG tool is getting detected in Atmel Studio:

Open Atmel Studio 6.2, go to '**View**' → '**Available Atmel Tools**'. The EDBG should get listed in the tools as "EDBG" and the tool status should display as "Connected" as shown in the figure below. This indicates that the tool is communicating properly with the Atmel Studio.

Figure 3-7 EDBG under Available Atmel Tools



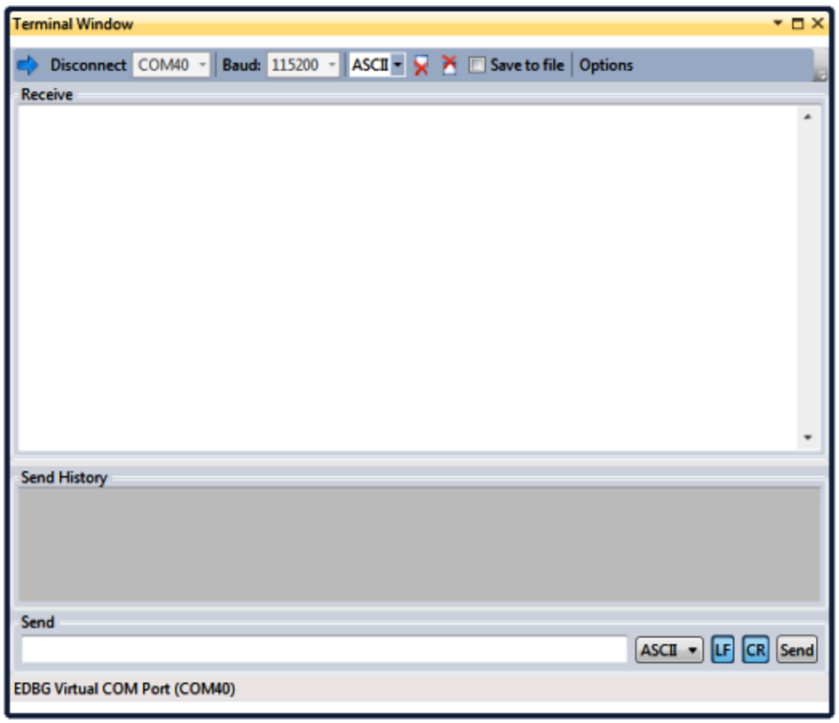
If the tool is not displayed in '**Available Atmel Tools**', disconnect the tool and reconnect again.

Right click on the tool in the '**Available Tools**' list and click on "**Upgrade**". This will check if the firmware in the tool is up to date. Click on "**upgrade**" to upgrade the firmware of the tool to the latest version.

After the software is successfully installed, open the terminal window with the COM port (EDBG Virtual COM port) number detected in Device Manager. The terminal window can be downloaded and installed either from '**Atmel Gallery**' or through '**Tools**' → '**Extension Manager**' in Atmel Studio.

Now the terminal window can be opened from '**View**' → '**Terminal Window**'. The COM port should be opened with a baudrate of 115200 with the display type as 'hex' (see Figure 3-8 Terminal Window in Atmel Studio on page 9). This step ensures that the COM port (EDBG Virtual COM port) enumurates properly, and is used to select the mode of operation. The user input string from the terminal via USART DATA register in this application for the demonstration purpose, which is explained in the following chapters.

Figure 3-8 Terminal Window in Atmel Studio



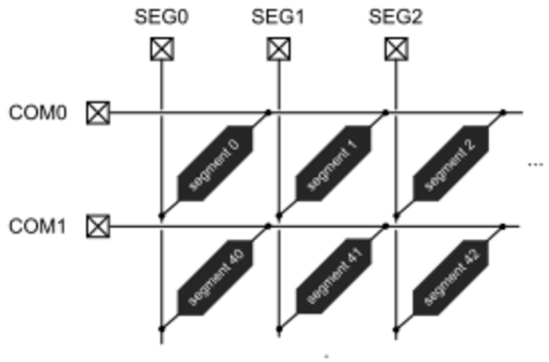
4. Segment Liquid Crystal Display (SLCD) Controller

4.1. Overview

An LCD display is made of several segments such as pixels or complete symbols which can be made visible or invisible. A segment has two electrodes with a liquid crystal between them. These electrodes are the common terminal (COM pin) and the segment terminal (SEG pin). When a voltage above a threshold voltage is applied across the liquid crystal, the segment becomes visible.

The SLCD controller is intended for monochrome passive liquid crystal display (LCD) with up to eight common terminals and up to 44 segment terminals.

Figure 4-1 LCD Panel - Segment/Common Terminals Connections

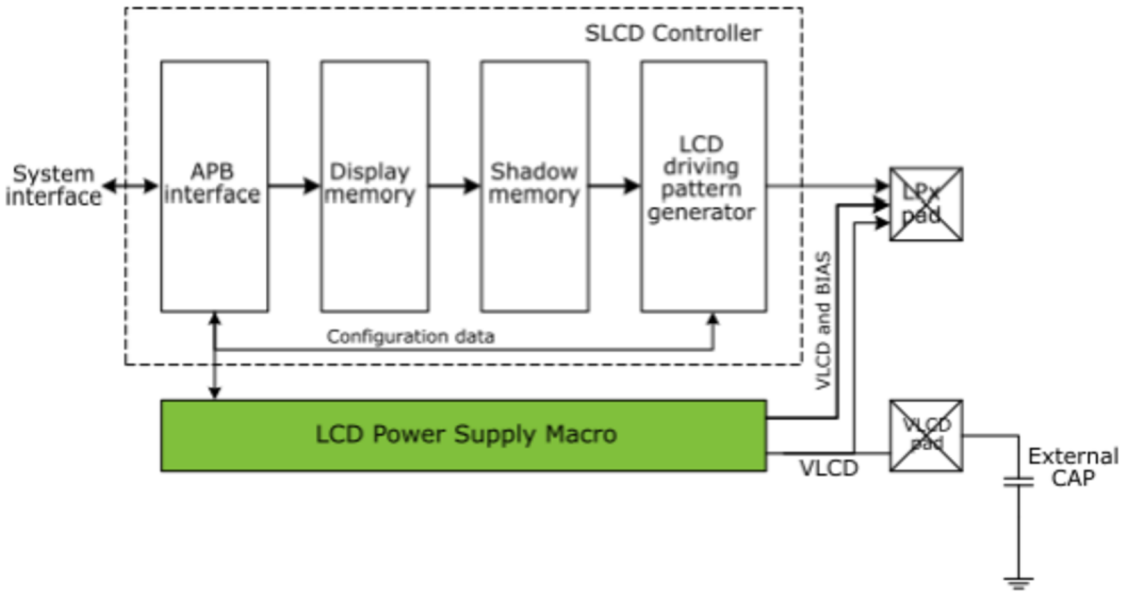


Note:
In order to avoid degradation due to electrophoresis in the liquid crystal, the waveform of the voltage across a segment must not have a DC component.

The Segment LCD1 Xplained Pro has four common and 24 segment terminals.

4.2. Block Diagram

Figure 4-2 SLCD Block Diagram



Note:
LPx - LCD Pin x (COM or SEG terminal) - Analog output

VLCD - LCD Voltage - Analog input or output

4.3. Functional Description

4.3.1. Basic Operation

In the SAM L22 when a bit in the display memory is written to '1', the corresponding segment will be energized (ON / opaque), and de-energized (OFF / transparent) when this bit is written to '0'. The display memory stores the values of all segments to display. It is accessible through APB and should be filled before the next frame starts. A start of a new frame triggers copying the display memory into the shadow display memory. A display memory refresh is thus possible without affecting data already sent to the panel.

Note:
The display memory is not initialized at startup.

Each COM line has identical waveforms but different phases. For each phase of the frame according to the bit value in the shadow display memory the SEG lines are driven to VLCD and GND when the pixel is ON, or to one of the bias voltages when the pixel is OFF.

Note:
The COM and SEG signal waveform depends on the waveform mode being selected either Standard waveform mode or Low power waveform mode

4.3.2. DUTY and BIAS

The configuration of the DUTY bits (DUTY[2:0] in CTRLA) define the number of COM lines used and thus the number of phases. BIAS bits (BIAS[1:0] in CTRLA) define the number of bias voltages. SAM L22 SLCD supports up to eight COM and up to three bias voltages.

For this application, Segment LCD1 Xplained Pro requires four common terminals and segment terminal 0 to 23 to control all segments. So, the Duty and Bias bits are configured as 011 and 10 respectively.

4.3.3. Different Waveform Modes

The LCD controller drives different waveforms according to the different bias configurations and the LCD controller supports two types of waveform drive as follows:

- Bit-inversion (type A, standard) called as Standard waveform mode
- Frame-inversion (type B, low-power) called as Low power waveform mode

The frame-inversion mode has a lower switching frequency than the bit-inversion mode, and thus reduce the power consumption compared to standard waveform. Both waveform modes have the same period and the DC-component is null. For static bias, standard and low-power waveform don't have any differences. However, for 1/2, 1/3, and 1/4 bias, low-power waveform provide less toggle rate compared to standard one.

By default the low-power waveform mode is enabled.

Refer the SAM L22 datasheet for different Standard and Low power waveform templates for the different Duty and Bias configurations.

4.3.4. LCD Frame Frequency

The LCD frame frequency is defined as the number of times the segments are energized per second. The optimal frame frequency should be in range from 30Hz up to 100Hz to avoid flickering and ghosting effect. Frame frequency (or frame rate) depends on the parameters such as SLCD clock source 32KHz oscillator clock CLK_SLCD_OSC, duty setting (CTRLA.DUTY[2:0] bits). prescaler setting (CTRLA.PRESC[1:0] bits), and Clock Divider settings (CTRLA.CKDIV[2:0] bits).

Figure 4-3 Framerate

$$\text{FrameRate} = \frac{F(\text{CLK\_SLCD\_OSC})}{\text{PVAL} \times (\text{CKDIV}+1) \times (\text{DUTY}+1)}$$

For this application the frame rate is calculated as 32Hz with 1/4 DUTY and 32768Hz input clock (CLK 32768 Hz, Prescaler 32 and CKDIV value is 7).

FrameRate = 32768/(32*(7+1)*4) = 32Hz.

Note:
The example values for the various configurations to select the FrameRate is given in the device datasheet.

4.3.5. LCD Pins Selection

Selection of maximum 48 segment/common lines from 52 LCD pins.

There are 52 LCD pins (LPx) from which up to 48 LCD pins can be enabled or disabled individually according to the LCD glass. Each LCD pin can be configured as front plane (SEG) or back plane (COM), offering various configurations. For 1/4 duty settings the maximum number of SEG lines and COM lines are 4 and 44 respectively. For the other duty configuration, this is mentioned in the device datasheet.

Write a '1' to a bit in LPENL or LPENH registers will enable the corresponding LCD pin.

- For LP[31:0], write to LCD Pin Enable Low register bits LPENL[31:0]

- For LP[51:32], write to LCD Pin Enable High register bits LPENH[18:0]

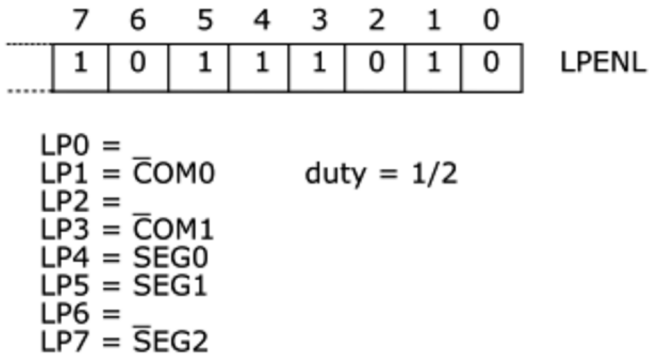
Writing a '0' to a bit in LPENL or LPENH registers will disable the corresponding LCD pin.

Note:

- LCD pins can be enabled individually. LCD pins need not be enabled in a contiguous manner.
- A disabled LCD pin can thus be used as GPIO or any other alternate function
- The number of LCD pins enabled should not be higher than the maximum of COM and SEG lines supported

The assignment of the COM and SEG lines are always in ascending order. According to their duty configuration COM lines are assigned first to the LCD pins enabled. The number of SEG lines enabled is thus the number of LCD pins enabled minus the number of COM lines assigned.

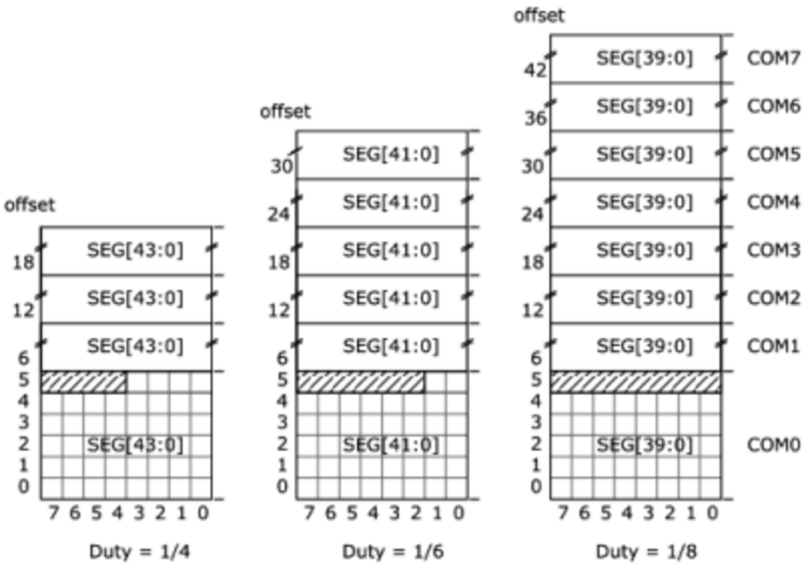
Figure 4-4 LCD Pins Configuration Example



4.3.6. Display Memory Mapping

The display memory size depends on the configured duty ratio. For 1/4 duty ratio the display memory is 44 bits wide per COM line. For 1/6 duty and 1/8 it is 42 bits and 40 bits wide respectively per COM line.

Figure 4-5 Display Memory Mapping



The CPU can access the display memory either through direct access or through indirect access.

With *Direct access*, the CPU can update the display memory by writing to the corresponding Segment's Data Low/ High for COMx Line register (SDATAL/Hx). For example, to update the segment connected to SEG4/COM2, write to bit 4 of the SDATAL2 register.

With *In-direct access*, the CPU can update the display memory by writing to the Indirect Segments Data Access register (ISDATA). This register allows to write up to eight contiguous bits in a single write operation to the display memory:

- SDATA[7:0] - segments data value (see the figure above)
- SDMASK[7:0] - mask for SDATA. When SDMASK[y]=1, SDATA[y] is not written to display memory.
- OFF[5:0] - byte offset in display memory (see the figure above)

4.3.7. Frame Counters

The frame counters are used as time base for the different functions (e.g. blinking or automated modes). There are three independent frame counters FC0, FC1, and FC2, which can be associated with any function.

The frame counter is synchronized to the LCD frame start and generates an internal event each time the counter overflows. The maximum value can be set for the (FCx.OVF) is 0x1F.

The formula used to generate the frequency of the interval event is f_{FCx} = (FrameRate) / ((FCx.OVF x 8) +1).

Note:
The FCx register can only be written when the frame counter x is disabled.

5. Overview of Peripherals Used

Other than SLCD and basic peripherals this chapter covers the overview of the other peripherals used for this applicaiton note. Refer to the respective sections in the product datasheet for more detailed description about their function and configuration.

5.1. DMAC

The Direct Memory Access Controller (DMAC) can transfer data between memories and peripherals, and thus off-load these tasks from the CPU. It enables high data transfer rates (using AHB clock) with minimum CPU intervention and frees up CPU time. This will allow the CPU to sleep for a longer time and thus reduce the power consumption.

A complete DMA read and write operation between memories and/or peripherals is called a DMA transaction. DMA reads data from the source address before writing to the destination address. A new data is read when the previous write operation is completed.

The transaction is initiated by a trigger and uses a DMA channel. The DMA trigger source can be application software, peripheral, or events from Event System (EVSYS).

Each read and write operation are done in blocks. The size of transfer is controlled by the block transfer size and is configured in the software. The size of the block can be from 1 to 64K beats. The beat can be byte, half-word, or word

5.2. SERCOM – USART

The SERCOM serial engine consists of a transmitter and receiver, baud-rate generator, and address matching functionality. The transmitter consists of a single write buffer and a shift register. The receiver consists of a two-levels receive buffer and a shift register. The baud-rate generator is capable of running on the GCLK_SERCOMx_CORE clock or an external clock.

The serial communication interface (SERCOM) can be configured to support a number of modes; I^2C, SPI, and USART. Once configured and enabled, all SERCOM resources are dedicated to the selected mode.

The universal synchronous and asynchronous receiver and transmitter (USART) is one of the available modes in the Serial Communication Interface (SERCOM).

A data transmission is initiated by loading the DATA register with the data to be sent. The data in TxDATA is moved to the shift register when the shift register is empty and ready to send a new frame. When the shift register is loaded with data, one complete frame will be transmitted.

The Transmit Complete interrupt flag in the Interrupt Flag Status and Clear register (INTFLAG.TXC) is set and the optional interrupt is generated, when the entire frame plus stop bit(s) have been shifted out.

The DATA register should only be written when the Data Register Empty flag in the Interrupt Flag Status and Clear register (INTFLAG.DRE) is set, which indicates that the register is empty and ready for new data.

USART can generate DMA request when the transmit buffer (TX DATA) is empty. The request is cleared when DATA is written.

Note:
In this application SAM L22 Xplained pro board the EDGB CDC (SERCOM4) is utilized to select the mode of operation and user input string from the terminal.

6. SLCD Example Implementation in SAM L22 MCUs

This chapter explains the application implementation in detail.

The objective of this application note is to demonstrate the features listed in this document and its configuration.

The example covers the following features:

1. Character Mapping.
2. Automated Character Mapping Scrolling.
3. Automated Bit Mapping.
4. Blink and Blank Feature.

6.1. Main Clock

The SLCD bus clock (`CLK_SLCD_APB`) must be enabled to access the registers and it can be configured in the Main Clock module MCLK,

A 32.768kHz oscillator clock (`CLK_SLCD_OSC`) is required to clock the SLCD. This clock must be configured and enabled in the 32KHz oscillator controller (`OSC32KCTRL`) before using the SLCD.

6.2. Basic Configuration

Basic configuration example covers pin initialization, clock initialization, EDBG USART initialization, and SLCD initialization functions. Below are the function calls for the basic configuration.

- `system_init()`
- `board_init()`
- `configure_console()`
- `xpro_lcd_init()`

The following sections will summarize each function.

6.2.1. System and Board Initialization

`system_init()` is an ASF function used to configure the generic clocks and clock sources as per the settings in the conf_clocks.h file. The main clock will be configured as stated in section Main Clock on page 16. `board_init()` initializes the board hardware of SAM L22 Xplained Pro.

6.2.2. EDBG USART Initialization

In this application SERCOM4 is connected to the EDBG USART lines through which the SAML22 Xplained pro will communicate with the PC terminal application. `configure_console()` function initializes the SERCOM USART module connected to the EDBG USART, configures the corresponding USART pins and the studio serial initializations for the standard library APIs (like scanf, printf, etc,.)

6.2.3. SLCD Initialization

`xpro_lcd_init()` function initializes the basic SLCD configurations such as Software contrast selection and Frame counter initialization. The files conf_slcd.h and conf_xpro_lcd.h has the user configurations macros, which defines the SLCD clock source, duty, bias, frame rate, contrast, VLCD selection, SLCD pin selection, etc.

6.3. Character Mapping

This feature is used to write the display memory for the each character by giving the character start index, segment data, and mask value.

In this mode, multiple segments of the LCD panel can be gathered into digits in order to display as characters. Digits can be of various types (e.g., 7-segment, 16-segment) and mapped in the display memory accordingly. We can find the different memory mapping scenarios in the device datasheet. The SLCD supports displaying up to 24-segment characters on any existing LCD panel.

The user can change the mapping order by configuring the CMCFG.DEC bit to match the display memory mapping to their SLCD segment order.

The CMCFG.NSEG defines the number of segments used to map a single digit. It should be the number of SEG line - 1. In the figure below each digit requires four segment lines. So the value for NSEG = 3.

Figure 6-1 Segment LCD1 Xplained Pro Segments

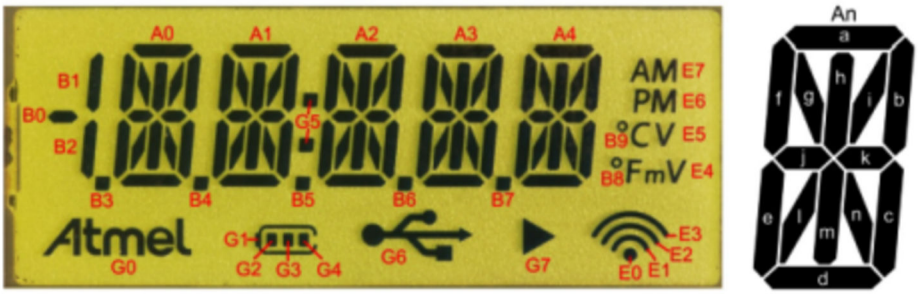


Table 6-1 Segment LCD1 Xplained Pro Segments description

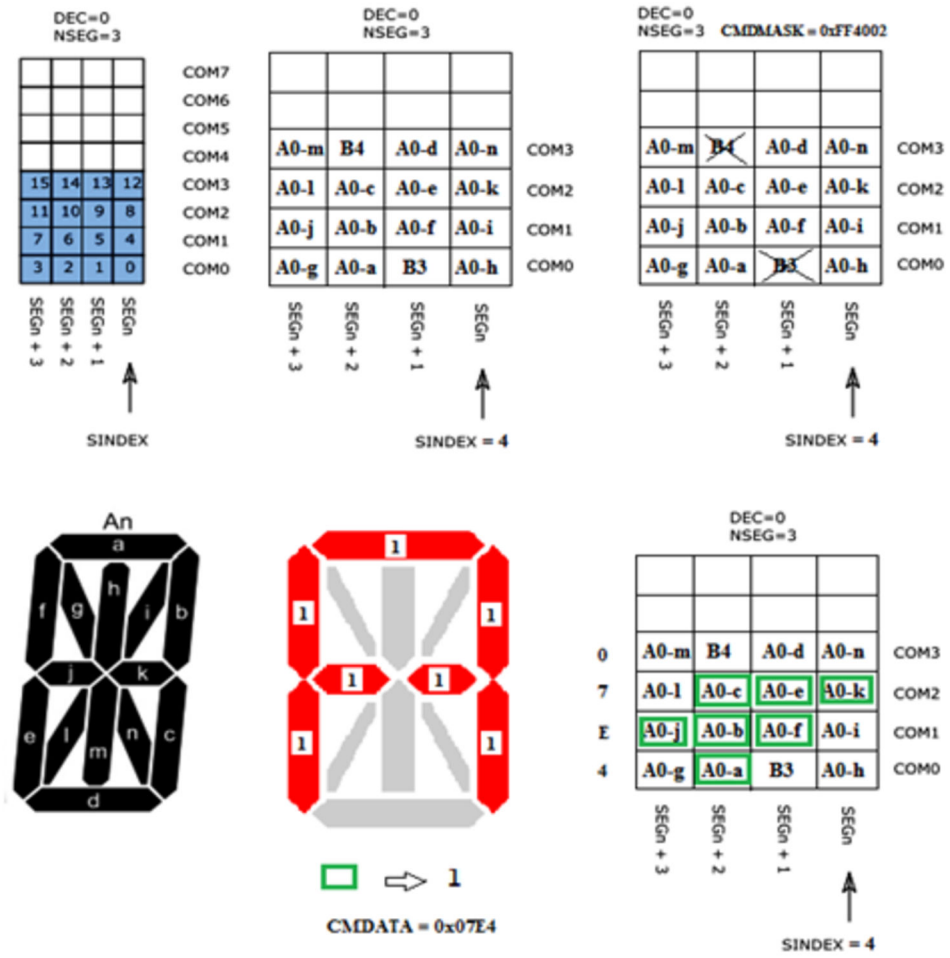
Segments	COM0	COM1	COM2	COM3	Comments
SEG0	G1	G2	G4	G3	Atmel logo, four stage battery-, Dot-point-, usb-, and play indicator
SEG1	G0	G6	G7	G5	
SEG2	E7	E5	E3	E1	Four stage wireless-, AM-, PM- Volt-, and millivolt indicator
SEG3	E6	E4	E2	E0	
SEG4	A0-h	A0-i	A0-k	A0-n	1st 14-segment character
SEG5	B3	A0-f	A0-e	A0-d	
SEG6	A0-a	A0-b	A0-c	B4	
SEG7	A0-g	A0-j	A0-l	A0-m	
SEG8	A1-h	A1-i	A1-k	A1-n	2nd 14-segment character
SEG9	B2	A1-f	A1-e	A1-d	
SEG10	A1-a	A1-b	A1-c	B5	
SEG11	A1-g	A1-j	A1-l	A1-m	

Segments	COM0	COM1	COM2	COM3	Comments
SEG12	A2-h	A2-i	A2-k	A2-n	3rd 14-segment character
SEG13	B1	A2-f	A2-e	A2-d	
SEG14	A2-a	A2-b	A2-c	B6	
SEG15	A2-g	A2-j	A2-l	A2-m	
SEG16	A3-h	A3-i	A3-k	A3-n	4th 14-segment character
SEG17	B0	A3-f	A3-e	A3-d	
SEG18	A3-a	A3-b	A3-c	B7	
SEG19	A3-g	A3-j	A3-l	A3-m	
SEG20	A4-h	A4-i	A4-k	A4-n	5th 14-segment character. Celsius and Fahrenheit indicator.
SEG21	B8	A4-f	A4-e	A4-d	
SEG22	A4-a	A4-b	A4-c	B9	
SEG23	A4-g	A4-j	A4-l	A4-m	

In the figure above the Segment LCD1 Xplained Pro LCD requires four segment lines and four com lines in each digit (14 segment per character). Icons starts from seg0 and alphanumeric digit starts from segment 4/8/12/16/20. When we match the segment mapping with display memory mapping, we can get the mapping order as shown in Figure 6-2 Example Character Mapping for Segment LCD1 Xplained Pro on page 19.

In this mode to write the character value in the segment memory register, the CMDATA register is used. Since the CMDATA register can be up to 24 bits, a mask can be configured to write only selected bits. Based on the user SLCD glass an unwanted bit can be masked by writing a '1' to a bit in the Character Mapping Data Mask register (CMDMASK). Then this bit will not be written in the display memory when CMDATA register is written.

Figure 6-2 Example Character Mapping for Segment LCD1 Xplained Pro



SEG4	A0-h	A0-i	A0-k	A0-n	1st 14-segment character
SEG5	B3	A0-f	A0-e	A0-d	
SEG6	A0-a	A0-b	A0-c	B4	
SEG7	A0-g	A0-j	A0-l	A0-m	

6.3.1. Configuration Steps to Enable and Print the Character in this Mode

- Disable SLCD and disable all the previous running modes as given in `xpro_lcd_clear_all()`
- Configure the NSEG and DEC values in the Character Mapping Configuration register (CMCFG) as shown in Figure 6-2 Example Character Mapping for Segment LCD1 Xplained Pro on page 19 as given in `slcd_character_map_set()`
- Write the coordinate of a character defined by the Character Mapping Index (CMINDEX) register
- Write the mask value in the Character Mapping Data Mask register (CMDMASK)
- Write the CMDATA register with the user character data
- Wait for character mapping write to complete. Then the user can check the status by the Character Mapping Write Busy bit in the Status register (STATUS.CMWRBUSY).

Note:

Any write access to the display memory through CMDATA/SDATA/ISDATA will be ignored when STATUS.CMWRBUSY is high. Therefore, the user must not write to CMDATA while STATUS.CMWRBUSY is asserted.

6.3.2. Example Configuration Used in this Application

The following sequence is a basic example to make the corresponding segments ON for the character 'A' in Segment LCD1 Xplained Pro LCD glass.

- Disable SLCD, disable all the previous running modes, and clear the SLCD screen:

```
xpro_lcd_clear_all();
```

- Write the configuration for the Character mapping mode:

```
/* DEC=0, NSEG=3 */
slcd_character_map_set(0,3);
```

- Write the CMDATA register with the user character data:

```
/* CMINDEX.CINDEX = 0, CMINDEX.SINDEX = 4 */
/* CMDATA = 0x07E4('A'), CMDMASK = 0xFF4002 */
slcd_character_write_data(0,4,0x07E4,0xFF4002);
```

- Wait for character mapping write complete:

```
/* wait for the character mapping write complete */
while (slcd_get_char_writing_status()) { };
```

6.3.3. Example Application Usage

After programming the application code we will get the user menu in the terminal window. Choose option 2 to check the "Character mapping feature".

Figure 6-3 Output on Terminal Window

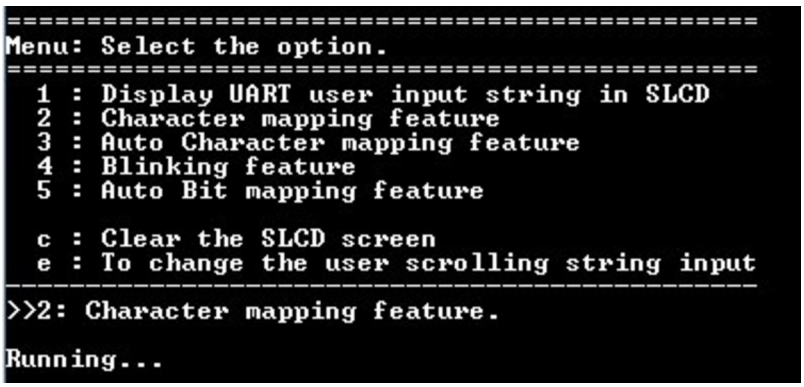
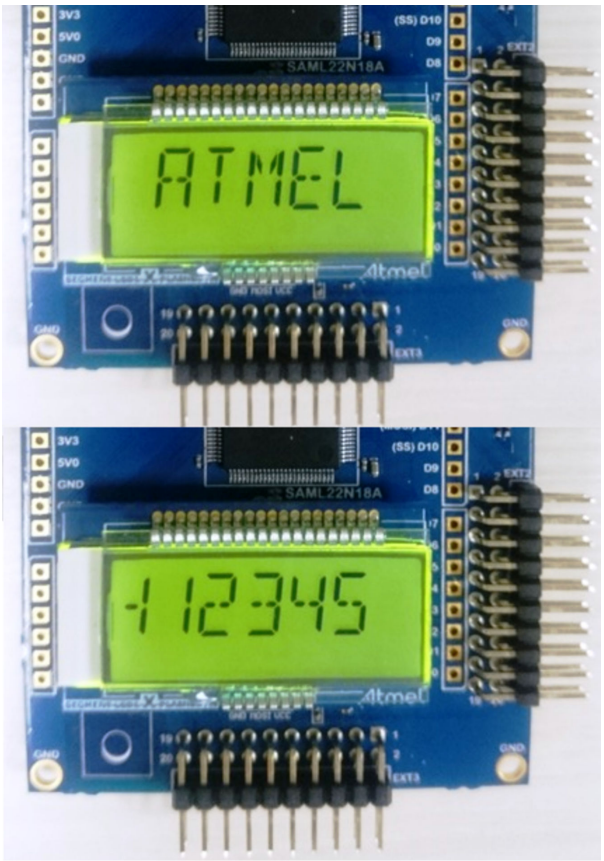


Figure 6-4 Output on SLCD



6.4. Automated Character Mapping

This feature is used to write the display memory with the sequence of segment values for the predefined strings automatically by using the Direct Memory Access (DMA). This Automated Character Mapping (ACM) has the following modes: “Sequential Characters String Display” and “Scrolling of Characters String”. This application briefs about ACM Scrolling of Characters String.

To use this mode we need to configure any of the frame counters among the three independent frame counters FC0, FC1, and FC2 to create a time base.

Note:
In ACM, compared to manual character mapping, there is one restriction; The first character is always mapped to the COM0 line.

6.4.1. Initialization

6.4.1.1. Automated Character Mode Initialization

To enable the automated character mapping the following configurations are necessary.

- Configure the NSEG and DEC values in Character Mapping Configuration register (CMCFG) and the mask value in the CMDMASK register as shown in Figure 6-2 Example Character Mapping for Segment LCD1 Xplained Pro on page 19
- ACMCFG.NDROW defines number of digits per row
- ACMCFG.STSEG defines the index of the first SEG line of the first digit
- ACMCFG.NDIG defines the number of characters in the whole string

Note:
Each row should contain NDROW digits except the last row.

Digits of the next row should be aligned with the digits of the previous row (same SEG lines).

6.4.1.2. ACM Scrolling Mode Initialization

The following configuration parameters in addition to section Automated Character Mode Initialization on page 21 will initialize the ACM scrolling mode.

- Write the number of scrolling steps to the Steps bits (ACMCFG.STEPS). The number of steps is equal to the string length - NDIG + 1.
- Write a '1' to the Mode bit in order to select the ACM mode (ACMCFG.MODE)
- Configure a frame counter to set the display period (period between two steps) and select the frame counter for the ACM scrolling mode by writing the frame counter index to the Frame Counter Selection bits (ACMCFG.FCS)

6.4.1.3. ACM Scrolling Mode DMA Initialization

DMA transfers the segment value for each character present in the String buffer to the CMDATA register upon the DMA peripheral trigger “Automated Character Mapping Data Ready (ACMDRDY)” occurs.

- For the ACM character mapping scrolling mode DMAC is configured to trigger a data transfer to the destination address configured when Automated Character Mapping Data Ready (ACMDRDY) occurs (peripheral trigger source)
- The trigger actions configured to generate a request for a beat transfer
- The destination address configured here is CMDATA register address and the source is DMA transfer buffer (dma_source_buf [ ])
- DMA source address configured as increments for each beat transfer and the destination address is static in this case
- Beat size defines the data transfer size for the each beat (WORD, HWORD, BYTE). In this case it is configured as WORD (32bit).
- Step size defines the source and destination address increment step. In this case the destination address is static and the source address is configured to increment with the step size of beat size * 1.
- Block transfer count defines the number of beats to be transferred for the complete scrolling string, and it is equal to the ACM scrolling string length if the beat size configured as same as the size of the source buffer member
- Block action defines the action made by the DMA after block transfer. In this case it is configured as sets transfer complete interrupt flag after block transfer and channel in normal operation.
- DMA next descriptor defines that the transaction consist of either a single block transfer or several block transfers. When a transaction consists of several block transfers it is called linked descriptors. The Next descriptor configured to 0 if single transaction required.
- Create the call-back function, register the DMA call back, and enable the call-back
- DMA is configured to transfer the same string multiple times to destination address using DMA linked descriptor. However, we have used this transfer complete flag to exit from this continuous transfer by the help of implementing a break logic to exit from this ACM scrolling mode and DMA transaction using a variable dma_break.

Note:
If the application requires the transfer complete, interrupt call back is required.

6.4.2. Configuration Steps to Enable and Scrolling the Characters in this Mode

- Disable SLCD and disable all the previous running modes as given in `xpro_lcd_clear_all()`

- Configure the NSEG and DEC values in the Character Mapping Configuration register (CMCFG) and the mask value in the CMDMASK register, as shown in Figure 6-2 Example Character Mapping for Segment LCD1 Xplained Pro on page 19
- Write the appropriate configuration for the ACM scrolling mode mentioned in the above section (Automated Character Mapping on page 21 and Initialization on page 21), using the function `slcd_automated_char_set_config()`
- Disable the frame counter using the function `slcd_disable_frame_counter(FCx)`
- Write the configurations of the frame counter associated with this mode in FCx.OVF and FCx.PB by using the function `slcd_set_frame_counter(FCx,0,0x1)`, and enable it by using the function `slcd_enable_frame_counter(FCx)`
- Enable the automated character mapping mode by writing the CTRLC. ACMEN bit using the function `slcd_enable_automated_character()`
- Configure the DMA resource parameters and allocate the resource using the function `configure_dma_resource(&example_resource)`
- Configure the DMA descriptor parameters by using the function `setup_transfer_descriptor(&example_resource)`
- Create the call-back function, register call-back, and enable the call-back if the application requires
- Call the function `dma_start_transfer_job(&example_resource)`. When the ACM peripheral trigger occurs, the DMA transfer will be initiated automatically.
- Since DMA transaction has been configured as linked descriptor, the exit from the continuous transaction, disabling the ACM scrolling mode and free the DMA channel used for this mode are implemented in the function `dma_callback()` by using the `dma_break` variable

6.4.3. Example Configuration Used in this Application

The following sequence is a basic example to scroll the string “HELLO WORLD ” by using the ACM scrolling mode in Segment LCD1 Xplained Pro LCD glass.

Buffers used :

```
/* DMA buffer length */
#define BUFFER_LEN 30
/* DMA resource and descriptor */
struct dma_resource example_resource;

COMPILER_ALIGNED(16)
DmacDescriptor example_descriptor;

static char user_scrolling_str[BUFFER_LEN] = {"HELLO WORLD     "};
static volatile uint32_t dma_source_buf[BUFFER_LEN];

static volatile bool transfer_is_done = false;
static volatile uint32_t dma_break = 0, dma_block_count = 0;
```

- Disable SLCD:

```
slcd_disable();
```

- Disable SLCD, disable all the previous running mode, and clear the SLCD screen:

```
xpro_lcd_clear_all();
```

- Load the user input string from `user_scrolling_str[]` to `dma_source_buf[]`
  This step includes the conversion of the ASCII value of each character in the Scrolling string `user_scrolling_str[]` into equivalent segment value from the lookup table array `DIGI_LUT[]` and find the block count. (Refer to Figure 6-2 Example Character Mapping for Segment LCD1 Xplained Pro on page 19 to find the segment values different characters.)

- Create SLCD configuration structure, check whether ACM scrolling mode is selected using the flag `is_scrolling`, get the default configuration of ACM mode, and load the configuration structure with the following configuration parameters for the ACM scrolling mode:

```
/*Structure for the ACM configurations */
struct slcd_automated_char_config automated_char_config;

/** CMCFG.DEC = 0 CMCFG.NSEG = 3 ACMCFG.NDROW = 5 ACMCFG.STSEG = 4
* ACMCFG.NDIG = 5 ACMCFG.STEPS = string length - NDIG + 1
* ACMCFG.MODE = 1 ACMCFG.FCS = 1 */


if(true == is_scrolling ){
    /*configuration for Auto character mapping scrolling mode */
    /* Get default config for the ACM mode */
    slcd_automated_char_get_config_default(&automated_char_config);

    /* Segment mapping order (CMCFG.DEC bit) */
    automated_char_config.order = 0;
    /** select the number of segment used per digit
    * it equal to number of SEG line - 1 (CMCFG.NSEG) */
    automated_char_config.seg_line_num = 3;

    /* Define the number of digit per row. (ACMCFG.NDROW) */
    automated_char_config.row_digit_num = 5;
    /** Define the index of the first segment terminal of the
    * digit to display (ACMCFG.STSEG) */
    automated_char_config.start_seg_line = 4;
    /** Define the number of digit, it must be greater than 1.
    * (ACMCFG.NDIG)*/
    automated_char_config.digit_num = 5;

    /* STEPS = string length - NDIG + 1 (ACMCFG.STEPS) */
    automated_char_config.scrolling_step = dma_block_count - 5 + 1;
    /* Select the ACM mode (ACMCFG.MODE) */
    automated_char_config.mode  = SLCD_AUTOMATED_CHAR_SCROLL;
    /* Select the frame counter for the ACM mode (ACMCFG.FCS) */
    automated_char_config.fc = SLCD_FRAME_COUNTER_1;

    /*Configure the mask value in the CMDMASK register */
    automated_char_config.data_mask = 0x00FF4002;
}
```

- Write the above configuration for the ACM scrolling mode by using the following function:

```
/* write the SLCD ACM configurations in the respective registers */
slcd_automated_char_set_config(&automated_char_config);
```

- Write the above configuration for the ACM scrolling mode by using the following function:

```
/* write the SLCD ACM configurations in the respective registers */
slcd_automated_char_set_config(&automated_char_config);
```

- Write the frame counter configurations:

```
/* Set the frame counter configurations and enable it */
slcd_set_frame_counter(automated_char_config.fc,0,0x1);
slcd_enable_frame_counter(automated_char_config.fc);
```

- Enable SLCD:

```
slcd_enable();
```

- Create the DMA resource configuration structure, get the default configuration of DMA resource configuration, check whether ACM scrolling mode is selected using the flag`is_scrolling`, and load the configuration structure with the following configuration parameters for the ACM scrolling mode:

```
/* Structure for the DMA resource configurations */
struct dma_resource_config config;
/* Get the default configuration */
dma_get_config_defaults(&config);

if( true == is_scrolling ){
    /* Set the peripheral trigger source */
    config.peripheral_trigger = SLCD_DMAC_ID_ACMDRDY;
    /* Set the trigger action */
    config.trigger_action = DMA_TRIGGER_ACTON_BEAT;
}
```

- Write the above DMA resource configuration for the ACM scrolling mode by using the following function:

```
configure_dma_resource(&example_resource);
```

- Create the DMA descriptor configuration structure, check whether ACM scrolling mode is selected using the flag `is_scrolling`, get the default configuration of DMA resource configuration, and load the configuration structure with the following configuration parameters for the ACM scrolling mode:

```
/* Structure for the DMA descriptor configurations */
struct dma_descriptor_config descriptor_config;

if( true == is_scrolling ){
    /* Get the default configuration */
    dma_descriptor_get_config_defaults(&descriptor_config);

    descriptor_config.beat_size = DMA_BEAT_SIZE_WORD;
    descriptor_config.src_increment_enable = true;
    descriptor_config.dst_increment_enable = false;
    descriptor_config.step_size = (DMA_ADDRESS_INCREMENT_STEP_SIZE_1);
    descriptor_config.step_selection = DMA_STEPSEL_SRC;
    descriptor_config.block_action = DMA_BLOCK_ACTION_INT;

    /* Block count manipulated run time from the scrolling string */
    descriptor_config.block_transfer_count = dma_block_count;
    descriptor_config.source_address = (uint32_t)dma_source_buf +
                                (dma_block_count *
sizeof(dma_source_buf[0]));
    descriptor_config.destination_address = (uint32_t)(&SLCD-
>CMDATA.reg);
    descriptor_config.next_descriptor_address =( uint32_t )descriptor;
```

```
    /* Create a DMA descriptor */
    dma_descriptor_create(descriptor, &descriptor_config);
}
```

- Write the above DMA descriptor configuration for the ACM scrolling mode by using the following function:

```
configure_dma_descriptor(&example_resource);
```

- Create the call back function, register call back, and enable:

```
dma_register_callback(&example_resource,
dma_callback,DMA_CALLBACK_TRANSFER_DONE);
dma_enable_callback(&example_resource, DMA_CALLBACK_TRANSFER_DONE);
```

- Start the DMA transfer:

```
/** Start DMA transfer once DMA gets the peripheral
* trigger form the ACM then the data transfer starts */
dma_start_transfer_job(&example_resource);
```

- Enable the ACM mode:

```
/* Enable ACM mode */
slcd_enable_automated_character();
```

6.4.4. Example Application Usage

After programming the application code we will get the user menu in the terminal window. Choose option 3 to check this feature.

Figure 6-5 Output on Terminal Window

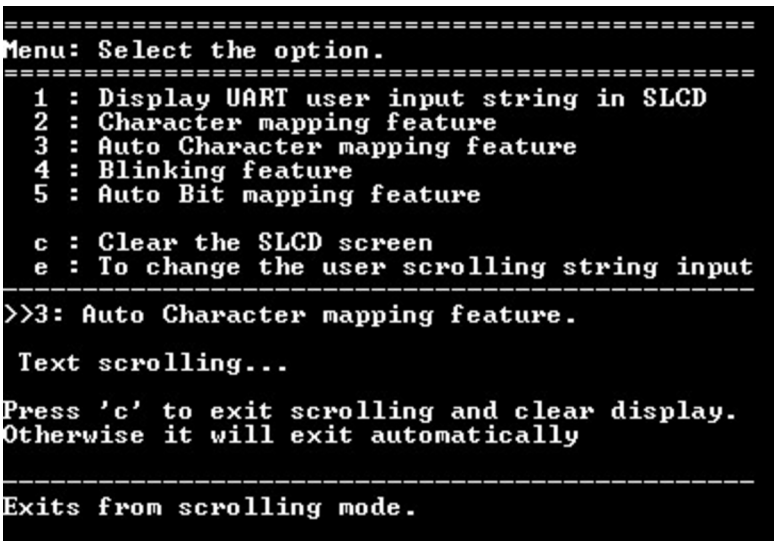
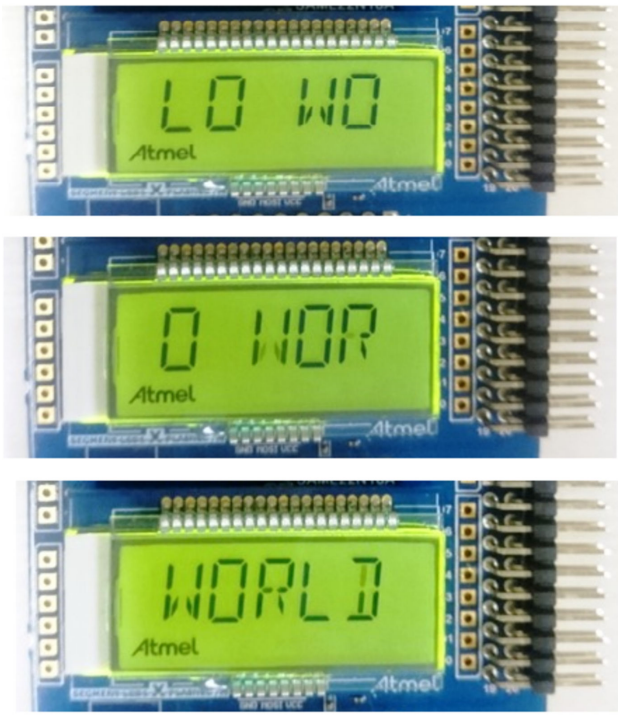


Figure 6-6 Output on SLCD



6.5. Automated Bit Mapping

This feature is used to write the display memory with the sequence of segment values for the predefined symbols automatically by using the Direct Memory Access (DMA). In this Automated Bit Mapping (ABM) mode several segments on the LCD panel can be gathered to make a symbol, which can be animated (i.e., have several states).

Data corresponding to each state of the animation can be stored in the system memory and is transferred periodically to the display memory by using the DMA controller.

To use this mode we need to configure any one of the frame counters among the three independent frame counters FC0, FC1, and FC2 to create a time base.

To make an automated animation of N states with M contiguous segment values in the display memory, the DMA controller must be configured to transfer N x M/8 words (eight contiguous segments are updated per write access).

Here N defines the number of states or frames consist of M contiguous segment values in each state.

M is the number of maximum number segment (24). In a single write the DMA can write eight segments so to write all the segments in the SLCD (for example 24) M/8 DMA writes are required.

In this case for the Segment LCD1 Xplained Pro LCD glass to write all the 24 segments 3 (24 segments/8 segments per write) DMA writes are needed to update all the segment values.

In a single DMA write, the maximum eight bits (for eight segments) can be updated in the display memory by *IN-Direct access* mode using the ISDATA register. The ISDATA register is 32 bit (one word) wide. It has the following fields.

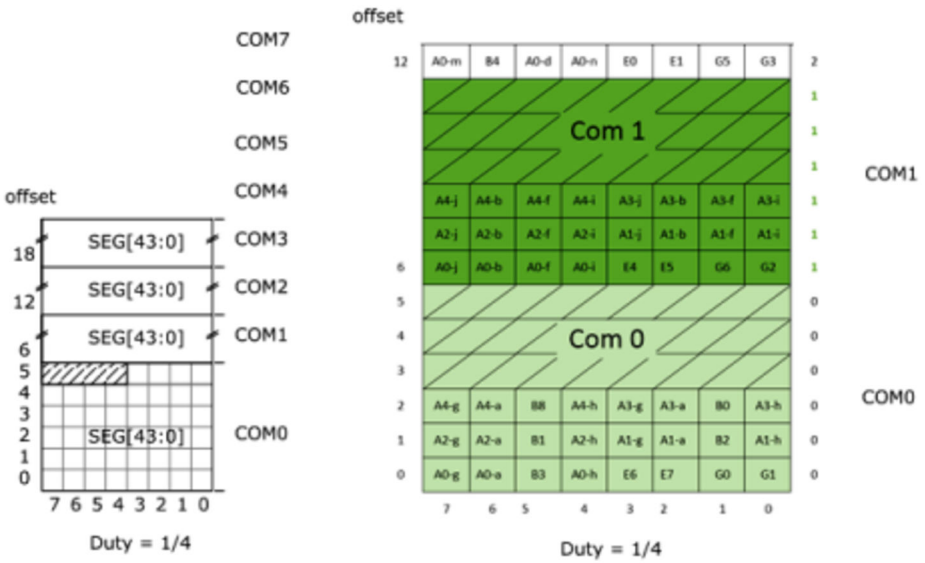
Format: [00, offset, data mask, data]

Format: [0x00, offset:[21-16], data mask:[15-8], data:[7-0]]

Each DMA write needs 32-bit value in the above mentioned format with the 8-bit segment data.

To write more than eight bits the DMA controller must be configured to transfer multiple words. This number of words must be written to the Size bits in the Automated Bit Mapping Configuration register (ABMCFG.SIZE). It defines the number of DMA writes to the display memory to form an animation frame.

Figure 6-7 Example Display Memory Mapping in ABM Mode for Segment LCD1 Xplained Pro



N defines the number of states or frames consist of M contiguous segment values in each state.

Indirect addressing format:

Format: [00, offset, data mask, data]

Format: [0x00, offset:[21-16], data mask:[15-8], data:[7-0]]

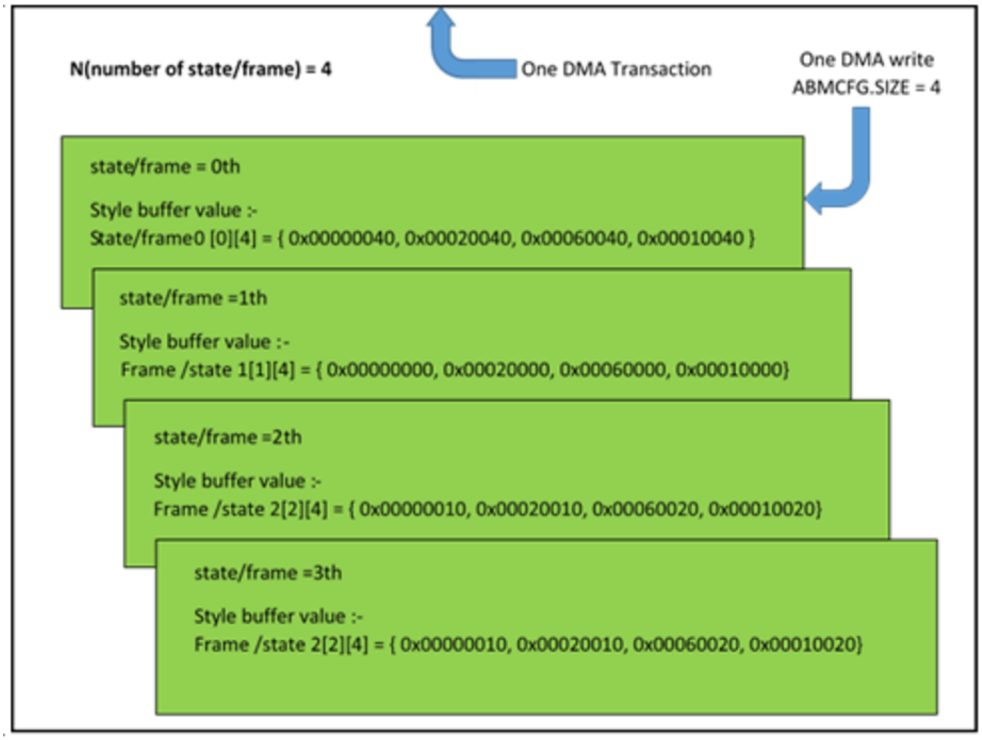
Frame /state 0[0][4] = { 0x00000040, 0x00020040, 0x00060040, 0x00010040}

Frame /state 1[1][4] = { 0x00000000, 0x00020000, 0x00060000, 0x00010000}

Frame /state 2[2][4] = { 0x00000010, 0x00020010, 0x00060020, 0x00010020}

Frame /state 3[3][4] = { 0x00000040, 0x00020040, 0x00060040, 0x00010040}

Figure 6-8 Example DMA Writes and State Formation in ABM Mode for Segment LCD1 Xplained Pro



6.5.1. Initialization

To enable the automated bit mapping the following configurations are necessary.

6.5.1.1. Automated Bit Mapping Mode Initialization

The following configuration parameters will initialize the ABM mode.

- Write the number of words per DMA write as the Size bits in the Automated Bit Mapping Configuration register (ABMCFG.SIZE)

Note: Size is a user defined configuration based on their animation pattern and must be greater than 1.

- Configure a frame counter to set the display period (period between two steps) and select the frame counter for the ACM scrolling mode by writing the frame counter index to the Frame Counter Selection bits (ABMCFG.FCS)

6.5.1.2. ABM Mode DMA Initialization

The DMA transfers the sequence of segment values for each predefined symbol present in the user buffer to the ISDATA register upon the DMA peripheral trigger “Automated Bit Mapping Data Ready (ABMDRDY)” occurs.

- For the ABM mode DMAC is configured to trigger a data transfer to the destination address configured when Automated Bit Mapping Data Ready (ABMDRDY) occurs (peripheral trigger source)
- The trigger actions configured to generate a request for a beat transfer
- The destination address configured here is the ISDATA register address and the source is the DMA transfer buffer (dma_source_buf [ ])
- DMA source address configured as increments for each beat transfer and the destination address is static in this case
- Beat size defines the data transfer size for the each beat (WORD, HWORD, BYTE). In this case it is configured as WORD (32-bit).

- Step size defines the source and destination address increment step. In this case the destination is static. However, the source address is configured to increment with the step size of beat size * 1.
- Block transfer count defines the number of beats to be transferred for all the animation states and it is equal to the ABM animation pattern length if the beat size is configured as the size of the source buffer member
- Block action defines the action done by the DMA after block transfer. In this case it is configured as sets transfer complete interrupt flag after block transfer and channel in normal operation.
- DMA next descriptor defines that the transaction consist of either a single block transfer or several block transfers. When a transaction consist of several block transfers it is called linked descriptors. The Next descriptor configured to 0 if single transaction is required.
- Create the call back function, Register the DMA call back, and enable the callback
- DMA is configured to transfer the same animation pattern in multiple times to destination address using DMA linked descriptor to repeat the animation pattern. However, we have used this transfer complete flag to exit from this continuous transfer by the help of implementing a break logic to exit from this ABM mode and continuous DMA transaction using a variable `dma_break`.

Note:
If the application requires the transfer complete interrupt call back is required.

6.5.2. Configuration Steps to Enable ABM mode

- Disable SLCD and disable all the previous running modes as given in `xpro_lcd_clear_all()`
- Write the appropriate configuration for the ABM mode mentioned in the above section (Automated Bit Mapping Mode Initialization on page 29) using the function `slcd_set_automated_bit()`
- Disable the frame counter by using the function `slcd_disable_frame_counter(FCx)`
- Write the configurations of the frame counter associated with this mode in `FCx.OVF` and `FCx.PB` by using the function `slcd_set_frame_counter(FCx,0,0x1)` and enable it by using the function `slcd_enable_frame_counter(FCx)`
- Enable the automated character mapping mode by writing the CTRLC. ABMEN bit using the function `slcd_enable_automated_bit()`
- Configure DMA resource parameters and allocate resource using the function `configure_dma_resource(&example_resource)`
- Configure the DMA descriptor parameters using the function `setup_transfer_descriptor(&example_resource)`
- Create the call back function, Register call back, and enable the callback if the application requires
- Call the function `dma_start_transfer_job(&example_resource)`. After that upon the ABM peripheral trigger occurs the DMA transfer will be initiated automatically.
- Since DMA transaction has configured as linked descriptor, the exit from the continuous transaction is implemented in the function `dma_callback()` by using `dma_break` variable. This step includes disabling the ABM mode and freeing the DMA channel used for ABM mode.

6.5.3. Example Configuration Used in this Application

The following sequence is a basic example to animate the user style1 using ABM mode in Segment LCD1 Xplained Pro LCD glass.

Buffers used :

```
/* DMA buffer length */
#define BUFFER_LEN 30

/* DMA resource and descriptor */
```

```
struct dma_resource example_resource;

COMPILER_ALIGNED(16)
DmacDescriptor example_descriptor;

static char user_scrolling_str[BUFFER_LEN] = {"HELLO WORLD     "};
static volatile uint32_t dma_source_buf[BUFFER_LEN];

static volatile bool transfer_is_done = false;
static volatile uint32_t dma_break = 0, dma_block_count = 0;

/** prints Atmel in user style1 */
user_style_buf[1][30] = {
0x00000040,0x00070000,0x000D0060,0x00140002, // S
0x000600e0,0x000D0060,0x00130008,0x00020004, // T
0x000C0070,0x00130008,0x00020004,0x0008000a, // Y
0x00120000,0x00010005,0x0008002a,0x000E0003, // L
0x00010085,0x00070070,0x000E0023,0x00140022, // E
0x0000FF00,0x0000FF00,0x0000FF00,0x0000FF00  // 1
}
```

- Disable SLCD:

```
slcd_disable();
```

- Disable SLCD and disable all the previous running mode and clears SLCD screen:

```
xpro_lcd_clear_all();
```

- Clear the previous values present in the dma_source_buf by using the below function:

```
clear_buffer();
```

- Loads the user animation pattern/style (consist of a list of frames) from `user_style_buf[1][30]` to `dma_source_buf[]` and counts the block count.

- ABM mode configuration parameters:

```
/* ABMCFG.SIZE = 3 ABMCFG.FCS = 1 */
config_size = 3;
config_fc_value = 0x0F;
config_fc = SLCD_FRAME_COUNTER_1;
```

- Write the above configuration for the ABM mode by using the following function:

```
/* write the SLCD ABM configurations in the respective registers */
slcd_set_automated_bit(config_size, config_fc);
```

- Write the frame counter configurations:

```
/* Set the frame counter configurations and enable it */
slcd_set_frame_counter(config_fc,0,0x0F);
slcd_enable_frame_counter(config_fc);
```

- Enable SLCD:

```
slcd_enable();
```

- Create DMA resource configuration structure, get the default configuration of DMA resource configuration, check whether ABM mode is selected using the flag `is_bitmapping`, and load the configuration structure with the following configuration parameters for the ABM mode:

```
/* Structure for the DMA resource configurations */
struct dma_resource_config config;
/* Get the default configuration */
dma_get_config_defaults(&config);

if( true == is_bitmapping ){
    /* Set the peripheral trigger source */
    config.peripheral_trigger = SLCD_DMAC_ID_ABMDRDY;
    /* Set the trigger action */
    config.trigger_action = DMA_TRIGGER_ACTON_BEAT;
}
```

- Write the above DMA resource configuration for the ABM mode by using the following function:

```
configure_dma_resource(&example_resource);
```

- Create DMA descriptor configuration structure, check whether ABM mode is selected using the flag `is_bitmapping`, get the default configuration of DMA resource configuration, and load the configuration structure with the following configuration parameters for the ABM mode:

```
/* Structure for the DMA descriptor configurations */
struct dma_descriptor_config descriptor_config;

if( true == is_scrolling ){
    /* Get the default configuration */
    dma_descriptor_get_config_defaults(&descriptor_config);

    descriptor_config.beat_size = DMA_BEAT_SIZE_WORD;
    descriptor_config.src_increment_enable = true;
    descriptor_config.dst_increment_enable = false;
    descriptor_config.step_size = (DMA_ADDRESS_INCREMENT_STEP_SIZE_1);
    descriptor_config.step_selection = DMA_STEPSEL_SRC;
    descriptor_config.block_action = DMA_BLOCK_ACTION_INT;

    /* Block count manipulated run time from the scrolling string */
    descriptor_config.block_transfer_count = dma_block_count;
    descriptor_config.source_address = (uint32_t)dma_source_buf +
                                (dma_block_count *
sizeof(dma_source_buf[0]));
    descriptor_config.destination_address = (uint32_t)(&SLCD-
>ISDATA.reg);
    descriptor_config.next_descriptor_address =( uint32_t )descriptor;

    /* Create a DMA descriptor */
    dma_descriptor_create(descriptor, &descriptor_config);
}
```

- Write the above DMA descriptor configuration for the ABM mode by using the following function:

```
configure_dma_descriptor(&example_resource);
```

- Create the call back function, Register call back and enable:

```
dma_register_callback(&example_resource,
dma_callback,DMA_CALLBACK_TRANSFER_DONE);
dma_enable_callback(&example_resource, DMA_CALLBACK_TRANSFER_DONE);
```

- Start DMA transfer:

```
/** Start DMA transfer once DMA gets the peripheral
* trigger from the ABM then the data transfer starts */
dma_start_transfer_job(&example_resource);
```

- Enable ABM mode:

```
/* Enable ABM mode */
slcd_enable_automated_bit();
```

6.5.4. Example Application Usage

After programming the application code we will get the user menu in the terminal window. Choose option 5 and then choose the options as listed below to check this feature.

Figure 6-9 Output on Terminal Window

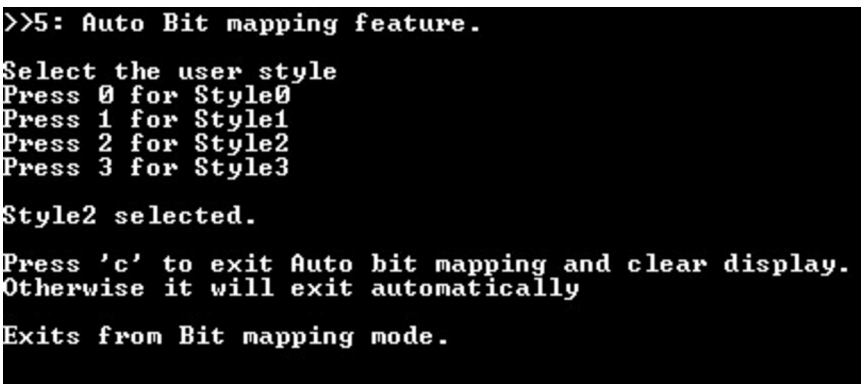
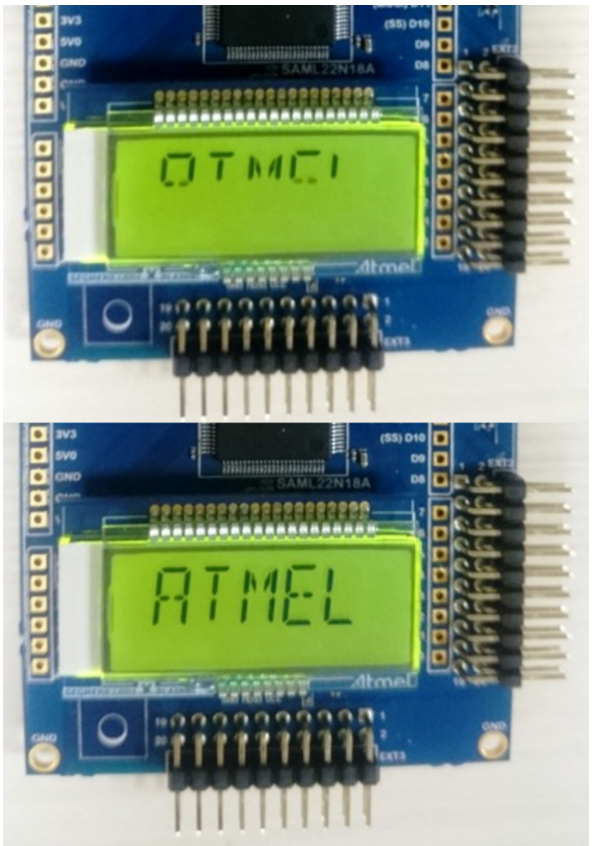


Figure 6-10 Output on SLCD



6.6. Blink Mode and Frequency Configuration

SLCD can be configured to blink all or selected LCD segments. Segments will alternate between ON and OFF state at the frequency given by the selected frame counter.

The blinking feature is configured in the Blink Configuration register (BCFG):

- Blink all the segments
  Writing '0' to the Blinking Mode bit (BCFG.MODE) will blink all the segments.
- In Blink selected segments mode up to sixteen segments can be enabled individually to blink, which are connected to SEG0, SEG1, and COM[0..7]
  Write '1' to the Blinking Mode bit (BCFG.MODE).

  Write '1' to the "Blink Segment Selection 0" bits (BCFG.BSS0) enables to blink the respective segments in segment 0 (SEG0), connected to COM0 up to COM7.

  Write '1' to the "Blink Segment Selection 1" bits (BCFG.BSS1) enables to blink the respective segments in segment 1 (SEG1), connected to COM0 up to COM7.

Note:
A segment will blink only if it is already ON - otherwise it will remain OFF.

In this case for the Segment LCD1 Xplained Pro LCD glass maximum we have four COM lines. Segment 0 and segment 1 are both connected to COM0 to COM3. Maximum eight segments can be configured to blink individually.

- The blink frequency is defined by the frame counter. We can associate any one of the frame counters to the blink feature by writing the corresponding frame counter x index to the Frame Counter Selection bits in the BCFG register (BCFG.FCS).
- Once the desired blink configuration is written to the BCFG register, blinking is enabled by writing a '1' to Blink bit in Control C register (CTRLC.BLINK). Blinking is disabled by writing a '0' to CTRLC.BLINK.

Note:
The BCFG register cannot be written when blink is enabled. The blink frequency can be modified. Before updating the new blink frequency the selected frame counter has been disabled.

6.6.1. Configuration Steps to Enable the Blink Mode

- Disable SLCD and disable all the previous running modes as given in `xpro_lcd_clear_all()`
- First make the segments ON which needs to be blinking by using the `xpro_lcd_blink_icon_start()` or `xpro_lcd_show_text()` function
- Disable the blink mode by writing a '0' to the Blink bit in Control C register (CTRLC.BLINK)
- Write the blink mode by Blinking Mode bit (BCFG.MODE)
- Write the required “Blink Segment Selection x” bits (BCFG.BSSx) if blink selected mode is configured
- Write the frame counter x index to the Frame Counter Selection bits in the BCFG register (BCFG.FCS)
- Write the appropriate configuration for the Blink mode mentioned in the above parameters by using the function `slcd_blink_set_config()` to blink all the segments, or slcd_set_blink_pixel() to blink the selected segments only
- Disable the frame counter using the function `slcd_disable_frame_counter(FCx)`. (Optional, not needed if already configured.)
- Write the configurations of the frame counter associated with this mode in `FCx.OVF` and `FCx.PB` by using the function `slcd_set_frame_counter(FCx,0,0x1)` and enable it by using the function `slcd_enable_frame_counter(FCx)`. (Optional, not needed if already configured.)
- Enable the blink mode by writing a '1' to the Blink bit in Control C register (CTRLC.BLINK)

Note:
The BCFG register cannot be written when blink is enabled. The blink frequency can be modified, before that the selected frame counter has been disabled.

6.6.2. Example Configuration Used in this Application

The following sequence is a basic example to blink the “BLINK” string in blink all segments mode in Segment LCD1 Xplained Pro LCD glass.

- Disable SLCD:

```
slcd_disable();
```

- Disable SLCD and disable all the previous running modes and clears SLCD screen:

```
xpro_lcd_clear_all();
```

- Enable the segments to blink:

```
/* Printing "BLINK" string on SLCD(used Character Mapping mode) */
xpro_lcd_show_text((const uint8_t *)"BLINK");
```

- Blink mode configuration parameters to blink all the segments:

```
/* BCFG.MODE = 0 BCFG.FCS = 1 */
slcd_blink_get_config_defaults(&blink_config);
blink_config.blink_all_seg = true;
blink_config.fc = CONF_XPRO_LCD_BLINK_TIMER;
```

- Write the above configuration for the blink mode by using the below function:

```
/* write the SLCD Blink configurations in the respective registers */
slcd_blink_set_config(&blink_config);
```

- Write the frame counter configurations:

```
/* Set the frame counter configurations and enable it */
slcd_set_frame_counter(blink_config.fc,0,0x1);
slcd_enable_frame_counter(blink_config.fc);
```

- Enable SLCD:

```
slcd_enable();
```

- Enable Blink mode:

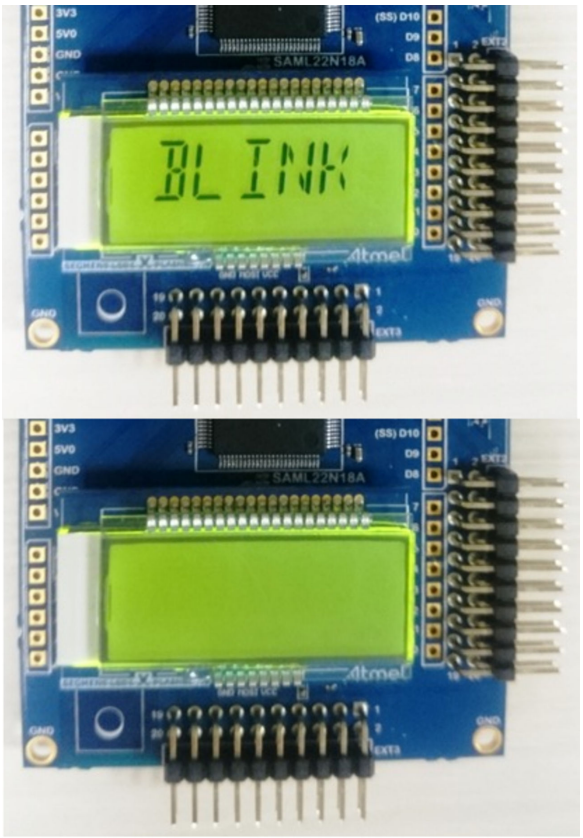
```
/* Enable Blink mode */
slcd_enable_blink();
```

6.6.3. Example Application Usage

After programming the application code we will get the user menu in the terminal window. Choose option 4 to check this feature.

Figure 6-11 Output on Terminal Window

Figure 6-12 Output on SLCD



7. References

SAM L22 Device Datasheet

Web page: http://www.atmel.com/products/microcontrollers/arm/sam-l.aspx?tab=documents

Document: Atmel SAM L22 Datasheet.pdf

SAM L22 Xplained Pro User Guide and Schematics

Web Page: http://www.atmel.com/tools/ATSAML22-XPRO.aspx?tab=documents

Atmel Segment LCD1 Xplained Pro User Guide

Web link: http://www.atmel.com/Images/Atmel-42076-Segment-LCD1-Xplained-Pro_User-Guide.pdf

8. Revision History

Doc Rev.	Date	Comments
42499A	08/2015	Initial document release.

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**

© 2015 Atmel Corporation. / Rev.: Atmel-42499A-SAM-L22-Segment-Liquid-Crystal-Display-SLCD-Controller-AT09192_Application Note-08/2015

Atmel®, Atmel logo and combinations thereof, Enabling Unlimited Possibilities®, and others are registered trademarks or trademarks of Atmel Corporation in U.S. and other countries. ARM®, ARM Connected® logo, and others are the registered trademarks or trademarks of ARM Ltd. Windows® is a the registered trademark of Microsoft Corporation in U.S. and or other countries Other terms and product names may be trademarks of others.

DISCLAIMER: The information in this document is provided in connection with Atmel products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Atmel products. EXCEPT AS SET FORTH IN THE ATMEL TERMS AND CONDITIONS OF SALES LOCATED ON THE ATMEL WEBSITE, ATMEL ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS AND PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF ATMEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Atmel makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and products descriptions at any time without notice. Atmel does not make any commitment to update the information contained herein. Unless specifically provided otherwise, Atmel products are not suitable for, and shall not be used in, automotive applications. Atmel products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.

SAFETY-CRITICAL, MILITARY, AND AUTOMOTIVE APPLICATIONS DISCLAIMER: Atmel products are not designed for and will not be used in connection with any applications where the failure of such products would reasonably be expected to result in significant personal injury or death (“Safety-Critical Applications”) without an Atmel officer's specific written consent. Safety-Critical Applications include, without limitation, life support devices and systems, equipment or systems for the operation of nuclear facilities and weapons systems. Atmel products are not designed nor intended for use in military or aerospace applications or environments unless specifically designated by Atmel as military-grade. Atmel products are not designed nor intended for use in automotive applications unless specifically designated by Atmel as automotive-grade.