SMART ARM-based Microcontrollers

AT12863: Interfacing LCD Controllers for SAM S70/E70/V70

APPLICATION NOTE

Preface

The Atmel® | SMART ARM® Cortex®-M7 based MCUs deliver the highest performing Cortex-M7 based MCUs to the market with exceptional memory and connectivity options for design flexibility making them ideal for the automotive, IoT, and industrial connectivity markets. The Atmel | SMART ARM Cortex-M7 architecture, while enhancing performance, keeps cost, and power consumption in check.

Liquid Crystal Displays (LCDs) offer several advantages over traditional cathode-ray tube displays that make them ideal for several applications.

This application note provides how Atmel SAM S70/E70/V70 ARM Cortex - M7 based microcontroller can be used to interface with external LCD controllers using EBI or SPI peripheral.

Table of Contents

1. Abbreviations

DMA	Direct Memory Access
EBI	External Bus Interface
FFC	Flexible Flat cable
GPIO	General Purpose Input Output
I2C	Inter Integrated Circuit
IM	Interface Mode
IoT	Internet of Things
LCD	Liquid Crystal Display
MCU	Micro Controller Unit
PWM	Pulse Width Modulation
RAM	Random Access Memory
SMC	Static Memory Controller
SPI	Serial Peripheral Interface
USB	Universal Serial Bus

2. Introduction

This application note explains interfaces with external LCD controllers on SAM S70/E70/V70 family devices. It uses External Bus Interface (EBI), Static Memory Controller (SMC), and Serial Peripheral Interface (SPI) features. The software examples mentioned in this document are provided in Atmel's Software Package.

For more details on EBI, SMC, and SPI peripherals, refer to SAM S70/E70/V70 device complete datasheet.

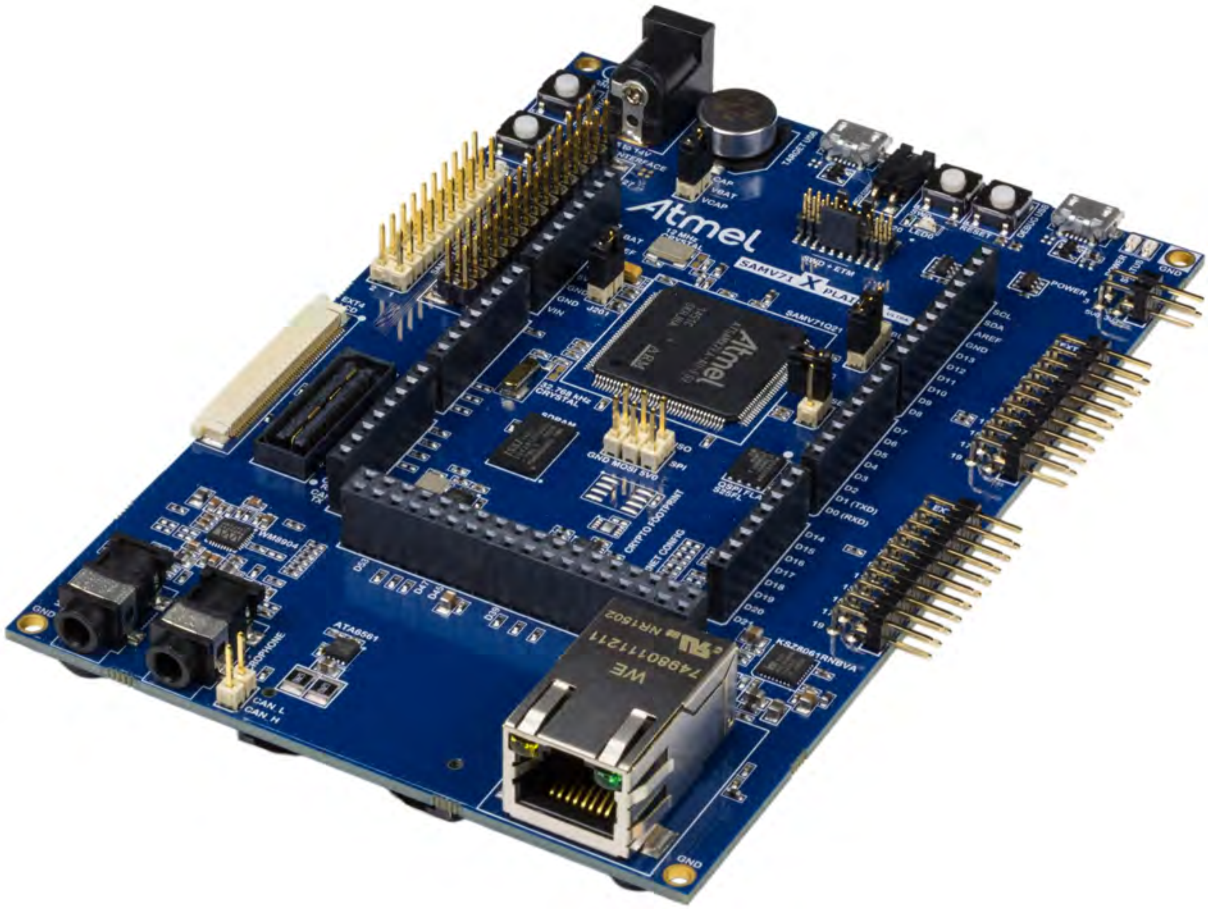
3. Prerequisites

- **Hardware Prerequisites**
  - Atmel | SMART SAM V71 Xplained ULTRA Kit
  - Atmel maXTouch Xplained Pro Kit
  - Interfacing Cables
    - One Micro USB B cable
    - One 50-way Flexible Flat Cable (FFC)
    - One 20-way Ribbon Cable
- **Software Prerequisites**
  - Atmel Studio 6.2 or later
  - Software Package 1.4 or later

Atmel | SMART SAM V71 Xplained ULTRA

The Atmel | SMART SAM V71 Xplained Ultra evaluation kit is a hardware platform to evaluate the ATSAMV71Q21 and other Atmel ARM Cortex-M7-based micro controllers in the SAM V70, SAM S70, and SAM E70 series. Supported by the Atmel Studio integrated development platform, the kit provides easy access to the features of the Atmel ATSAMV71Q21 and explains how to integrate the device in a custom design. The Xplained Ultra series evaluation kits include an on-board Embedded Debugger, and no external tools are necessary to program or debug the ATSAMV71Q21. The Xplained Pro extension kits offers additional peripherals to extend the features of the board and ease the development of custom designs.

Figure 3-1. SAM V71 Xplained ULTRA Board

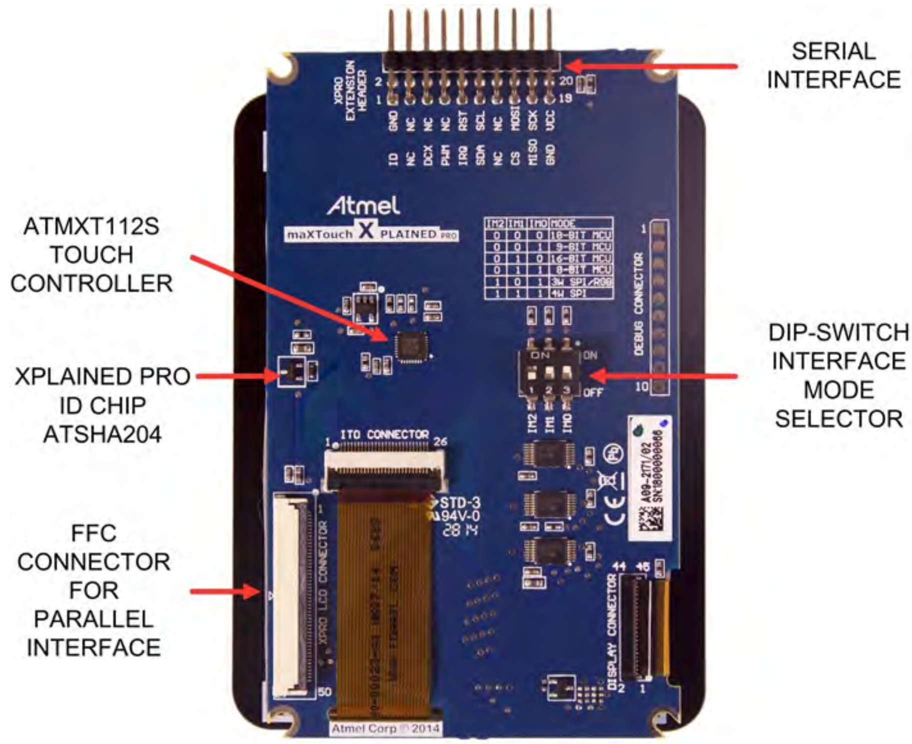


Atmel | maXTouch Xplained Pro

Atmel maXTouch Xplained Pro is an extension board for the Xplained Pro platform with a 320x480 RGB LCD and a capacitive touch sensor with a maXTouch controller. The LCD can be controlled via different interfaces, including 3- and 4-wire SPI, Parallel and RGB Parallel interface mode using the DIP-switch to select the interface. The maXTouch Xplained Pro kit connects to any Xplained Pro standard extension header on any Xplained Pro MCU board using the 20-pin header, but is limited to 3- and 4-wire SPI mode. Atmel maXTouch Xplained Pro also features a standard Xplained Pro LCD connector (FFC), which enables use of the parallel interfaces. Both connections features SPI interface for the LCD and I2C for the maXTouch device.

The ILI9488 controller is used to drive LCD on this board.

Figure 3-2. maXTouch Xplained Pro



4. Application Demonstration

This section demonstrates interfacing LCD controllers with SAM V71 using EBI (Extended Bus Interface) and SPI (Serial Peripheral Interface). EBI is demonstrated using `lcd_ebi` and SPI is demonstrated using `lcd` example projects. Both these projects are developed for ILI9488 controller.

This section is divided into EBI and SPI to describe about the interface details and software modules.

4.1. EBI – Extended Bus Interface

4.1.1. Hardware Setup

The External Bus Interface (EBI) is designed to ensure the successful data transfer between several external devices and the embedded Memory Controller of an ARM-based device. Static Memory Controller (SMC) is part of EBI. This can handle several types of external memory and peripheral devices. The SMC generates signals that control access to these devices. It has 4 chip selects, a 24-bit address bus, a configurable 8 or 16-bit data bus and separate read and write control signals.

Along with SMC, 3 GPIOs are used to control Reset, Command/Data Signal, and Back light.

Test Setup

The application demonstration needs following setup:

1. Atmel maXTouch Xplained Pro features a 3-way DIP-switch that is used for configuring the display Interface Mode (IM). Setting the switch positions to ON, will result in a high level (1) for the IMx line. To enable 16-bit parallel bus interface with maXTouch, set IM2=OFF, IM1=ON, IM0=OFF, refer Figure 4-2 IM2, IM1, IM0 Settings for EBI on page 9.
2. Connect SAM V71 Xplained board to maXTouch Xplained board on “EXT4 LCD” connector.
3. Connect SAM V71 Xplained DEBUG USB to PC. This provides power to target as well as interface to debug/download application image.

Following graphic shows connection setup.

Figure 4-1. Xplained ULTRA and maXTouch Interface on EBI

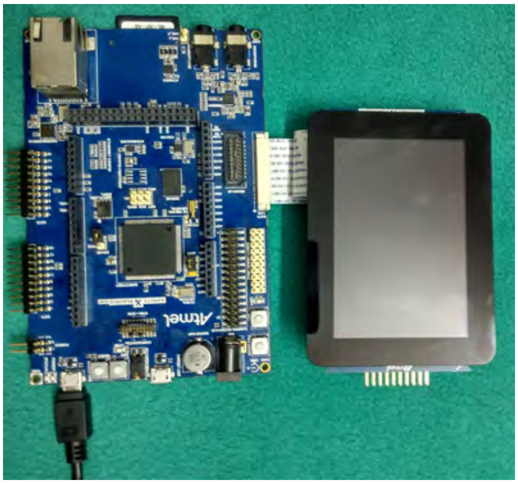
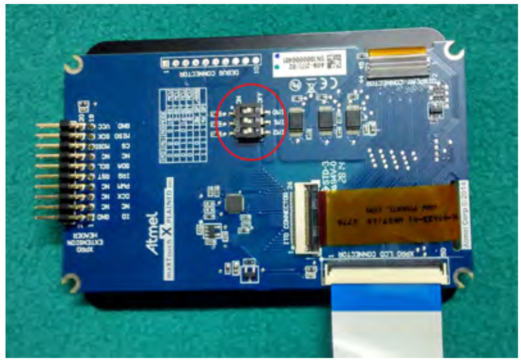
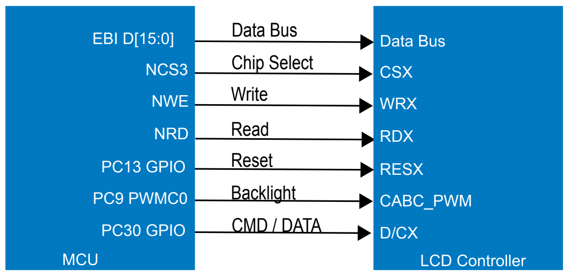


Figure 4-2. IM2, IM1, IM0 Settings for EBI



Interface Details

Figure 4-3. EBI Connections between MCU and LCD Controller

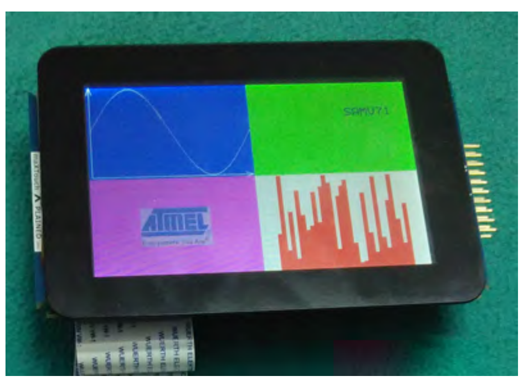


4.1.2. Software Setup

The `lcd_ebi` project is developed using Software Package to demonstrate EBI interfacing with LCD controller. This section describes software modules and their interfaces in this project. Intention of this project is to show how LCD controllers can be interfaced with SAM V71 using EBI. The complete project solution can be found at `Software Package Install Directory\Atmel \samv71_Xplained_Ultra\examples\lcd_ebi\build\studio\lcd_ebi.atsln`.

On executing `lcd_ebi` project, LCD should look as shown in following graphic.

Figure 4-4. LCD Screen on Running lcd_ebi Project



This application splits LCD into 4 equivalent regions to draw a Sine Wave, a “SAMV71” string, an Image (Atmel logo) and a histogram graph (Histogram is a graphical representation of the distribution of numerical data) on individual regions.

This application contains:

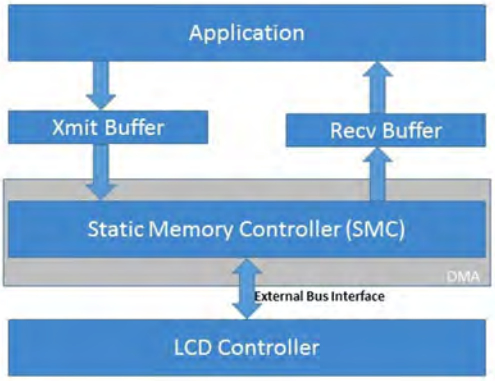
- Hardware Init
  - LCD Initialization

- System Tick Timer Initialization

- Screens Update
  - Screens Initialization
  - Screens Refresh

Following figure depicts how application is interfaced with LCD controller using SMC modules. Whenever application requires to send information to LCD controller, it first updates to Xmit buffer (referred as `canvas_region_buffer` in the code) and then triggers DMA. DMA transfers this data to SMC controller which in turn communicates with LCD controller via EBI.

Figure 4-5. Application Architecture - EBI



4.1.2.1. Hardware Init

LCD Initialization

The `LCDD_Initialize` function initializes EBI hardware pins, SMC configuration, DMA for Memory to Memory transfers among SMC, transmit buffer and receive buffer and register configurations on LCD controller (ILI9488). Following table explains various function calls associated with this function and their description.

Table 4-1. LCDD_Initialize

Function	Description
`ILI9488_EbiInitialize`	Initializes ILI9488 interface in SMC mode
`_ILI9488_EbiHW_Initialize`	Pin configurations for ILI9488 hardware, includes 16-bit data bus, Chip select, Write, Read, Reset, Command/Data, and Back light signals (PWM)
`ILI9488_EbiInitializeWithDma`	Initializes ILI9488 driver with DMA support
`_ILI9488_ConfigureSmc`	Configures SMC to assign NCS3 for ILI9488 interface along with Setup, Pulse, and Cycle timings for static memory. This also enables 16-bit width data bus, Read and write modes, extended wait disabled and data float timings
`_ILI9488_EbiDmaInitialize`	Initializes ILI9488 DMA structure and corresponding DMA driver
`_ILI9488_EbiDmaConfigChannels`	Initializes DMA TX & RX channels for Memory to Memory transfer and call back routines for DMA driver

Function	Description
_ILI9488_EbiDmaConfigureRxTx	Configures DMA TX channel with • Destination address as SMC base address • Transfer type as Memory to Memory • Single Micro block with chunk size as 1 • Data width as 16-bits • Incremented start address and Fixed destination address Same way RX channel with • Source address as SMC based address • Transfer type as Memory to Memory • Single Micro block with chunk size as 1 • Data width as 16-bits • Fixed start address and Incremented destination address TX channel source address and RX channel destination address will be updated in run time
ILI9488_EbiSendCommand	This function will be called multiple times based on required register settings on LCD controller

System Tick Timer Initialization

The `TimeTick_Configure` function initializes System tick timer to perform various timings related activities like individual region refresh rates and wait times based on need. A timer event is created for each of the 4 regions.

4.1.2.2. Screens Update

Screens Initialization

Screens initialization includes defining coordinates for all 4 regions, updating COLUMN ADDRESS SET and PAGE ADDRESS SET registers of LCD controller and updating individual regions data to LCD controller.

Table 4-2. Screens Initialization

Function	Description
init_canvas_region	Initializes coordinates for individual regions and LCD size parameters like Coordinates X, Y and Height, Width of LCD.
LCDD_SetUpdateWindowSize	Updates LCD size parameters to LCD controller registers.

Updating regions data depends on information to display and controller. In this case, it requires to set region size, update regions display information and issue `CMD_MEMORY_WRITE` to LCD controller along with data.

Table 4-3. Sine Wave – 1st Region

Step	Function	Description
Set regions size	LCDD_SetUpdateWindowSize	Updates COLUMN ADDRESS and PAGE ADDRESS registers of LCD controller with this region coordinates.
Update regions information	LCDD_DrawRectangleWithFill	Updates this region's canvas buffer as Rectangle and fills with chosen color (COLOR_BLUE).
	draw_coordinate_axis	Updates this region's canvas buffer to draw X and Y axis's with chosen color (COLOR_WHITE).
	draw_sin_wave	Updates this region's canvas buffer for Sine wave by using predefined (sin_xy array) pixel coordinates.
Send regions information to LCD controller	LCDD_UpdatePartialWindow	Triggers CMD_MEMORY_WRITE to LCD controller with this region data.

Table 4-4. Custom String – 2nd Region

Step	Function	Description
Set regions size	LCDD_SetUpdateWindowSize	Updates COLUMN ADDRESS and PAGE ADDRESS registers of LCD controller with this region coordinates
Update regions information	LCDD_DrawRectangleWithFill	Updates this region's canvas buffer as Rectangle and fills with chosen color (COLOR_GREEN)
	LCD_DrawString	Updates this region's canvas buffer with string to be displayed with chosen font (pCharset10x14 table) and color (COLOR_BLACK)
Send regions information to LCD controller	LCDD_UpdatePartialWindow	Triggers CMD_MEMORY_WRITE to LCD controller with this region data

Table 4-5. Image – 3rd Region

Step	Function	Description
Set regions size	`LCDD_SetUpdateWindowSize`	Updates `COLUMN ADDRESS` and `PAGE ADDRESS` registers of LCD controller with this region coordinates.
Update regions information	`LCDD_DrawRectangleWithFill`	Updates region's canvas buffer as Rectangle and fills with chosen color (`COLOR_MAGENTA`).
	`LCDD_BitBltAlphaBlend`	Updates this region's canvas buffer with source image data (`gImageBuffer`).
Send regions information to LCD controller	`LCDD_UpdatePartialWindow`	Triggers `CMD_MEMORY_WRITE` to LCD controller with this region data.

Table 4-6. Random Histogram – 4th Region

Step	Function	Description
Set regions size	`LCDD_SetUpdateWindowSize`	Updates `COLUMN ADDRESS` and `PAGE ADDRESS` registers of LCD controller with this region coordinates.
Update regions information	`LCDD_DrawRectangleWithFill`	Updates this region's canvas buffer as Rectangle and fills with chosen color (WHITE).
	`draw_random_histogram`	Updates this region's canvas buffer with randomly generated rectangle sizes and filled with chosen color (RED).
Send regions information to LCD controller	`LCDD_UpdatePartialWindow`	Triggers `CMD_MEMORY_WRITE` to LCD controller with this region data.

Screens Refresh

When application contains information that needs update at regular interval, it is required to send such information periodically to LCD controller. Screen Refresh depends on display information and refresh rate.

Sine Wave – 1st Region

There is pointer moving along Sine Wave path. It is configured to update pointer location on every system tick by treating it as a small rectangle filled with YELLOW. Following snippet in `update_region1(uint32_t pos)` helps in displaying the moving pointer.

```
/* pos offset is incremented or decremented before calling update_region1 */
LCDD_DrawRectangleWithFill(cavas_region_buf, 4 + pos - 2, 155 - sin_xy[pos] - 2, 3,
3, COLOR_CONVERT(COLOR_YELLOW));
```

Custom String – 2nd Region

Custom string location and color is set to update on every 500 ticks. Random generator is used to generate position and color code. Following snippet in `while(1)` helps in updating string location and color.

```
case 2:
    /* Enabling Random generator module and wait for number generation */
    TRNG->TRNG_CR = TRNG_CR_KEY_PASSWD | TRNG_CR_ENABLE;
    while(!(TRNG->TRNG_ISR & TRNG_ISR_DATRDY));
    i = TRNG->TRNG_ODATA; /* Use random number for position coordinates */
    j = (i >> 16) & 0x1F; /* Use random number for color choice */
    /* Update coordinates and color code using random number and send information
to LCD */
    region2_x =i & 0xFF;
    region2_y =(i >> 8)&0xFF;
    region2_x = region2_x * (canvas_region[1].width - 80) / 0xFF;
    region2_y = region2_y * (canvas_region[1].height - 20) / 0xFF;
    update_region2(region2_x,region2_y,COLOR_CONVERT(gColorArray[j]));
```

Image – 3rd Region

Custom image (Atmel logo) location is set to update on every 20 ticks. It is configured to create an offset of 4 steps. Following snippet in `while(1)` helps in updating image location.

```
case 3:
    /* Update region with preset offset */
    update_region3(alpha);
    /* adjust offset based on its current position */
    alpha += alpha_direction;
    if(alpha >= 255)
        alpha_direction = -4;
    else if(alpha <= 0)
        alpha_direction = 4;
```

Random Histogram – 4th Region

Random histogram is set to update on every 300 ticks. It is configured to create histogram with varying size rectangles. Random number generator is used to create varying size rectangles. Following snippet in `draw_random_histogram` helps in generating histogram.

```
/* Use random number to create offset */
    for(i = 0; i < 8; i++) {
        TRNG->TRNG_CR = TRNG_CR_KEY_PASSWD | TRNG_CR_ENABLE;
        while(!(TRNG->TRNG_ISR & TRNG_ISR_DATRDY));
        p_val[i] = TRNG->TRNG_ODATA;
    }
    /* Use random numbers to create rectangle offsets */
    for(i = 0; i< 32; i++) {
        rand_val[i] = offset_y - rand_val[i] * offset_y / 255;
    }
    /* Update rectangle coordinates and color code in to region buffer */
    for(i =0; i< 32; i++) {
        LCDD_DrawRectangleWithFill(cavas_region_buf, offset_x+i*w,rand_val[i] , 6,
```

```
offset_y-rand_val[i], color);
    }
```

Refer to `lcd_ebi` solution for complete details.

4.2. SPI – Serial Peripheral Interface

4.2.1. Hardware Setup

The Serial Peripheral Interface (SPI) circuit is a synchronous serial data link that provides communication with external devices in Master or Slave mode. The SPI system consists of two data lines and two control lines:

- **Master Out Slave In (MOSI)** - This data line supplies the output data from the master to into slave(s).
- **Master In Slave Out (MISO)** - This data line supplies the output data from a slave to master.
- **Serial Clock (SPCK)** - This control line is driven by the master and regulates the flow of the data bits.
- **Slave Select (NSS)** - This control line allows slaves to be turned ON and OFF by hardware.

Along with SPI in Master mode, 2 GPIOs and 1 PWM is used to control Reset, Command/Data Signal, and Back light.

Test Setup

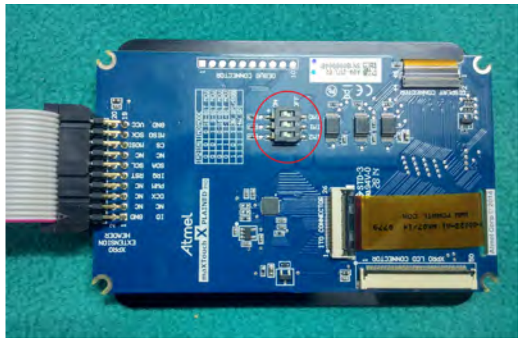
The application demonstration needs following setup:

1. maXTouch Xplained Pro features a 3-way DIP-switch that is used for configuring the display Interface Mode(IM). Setting the switch positions to ON, will result in a high level (1) for the IMx line. To enable 4-wire SPI interface with maXTouch, set IM2=ON, IM1=ON, IM0=ON, refer Figure 4-7 IM2, IM1, and IM0 Settings for SPI on page 16 .
2. Connect SAM V71 Xplained board to maXTouch Xplained board on EXT2 connector.
3. Connect SAM V71 Xplained debug USB to PC. This provides power to target as well as interface to download application executable.

Following graphic shows connection setup.

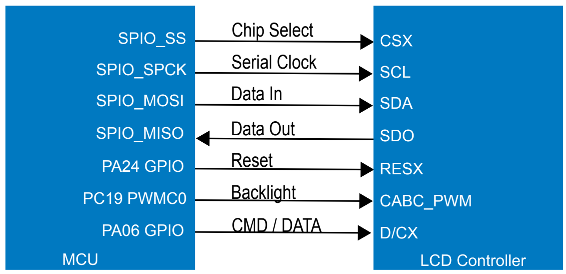
Figure 4-6. Xplained ULTRA and maXTouch Interface on SPI

Figure 4-7. IM2, IM1, and IM0 Settings for SPI



Interface Details

Figure 4-8. SPI Connections between MCU and LCD Controller

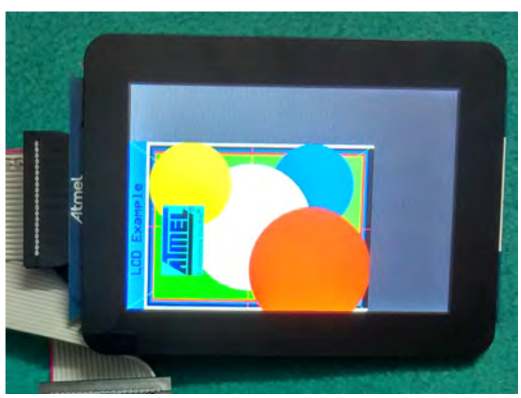


4.2.2. Software Setup

The `lcd` project is developed using Software Package to demonstrate SPI interfacing with LCD controller. This section describes software modules and their interfaces in this project. Intention of this project is to show how LCD controllers can be interfaced with SAM V71 using SPI. The complete project solution can be found at `Software Package Install Directory\Atmel\samv71_Xplained_Ultra \examples\lcd\build\studio\lcd.atsln`.

On executing lcd project, LCD should look as shown in following graphic.

Figure 4-9. LCD Screens on Running lcd Project



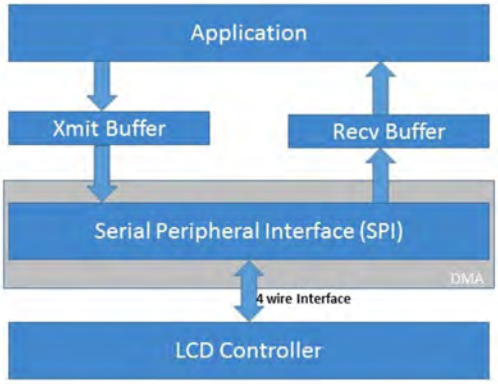
This application uses ¾ of LCD available on maXTouch Xplained Pro. It demonstrates LCD functionality with a single region buffer in RAM. Based on available RAM this limits total region to ¾ of LCD size. This application draws various patterns sequentially and in the end displays Atmel logo at various positions.

It contains the following:

- Hardware Init
  - LCD Initialization
  - System Tick Timer Initialization
- Screens Update
- Screens Refresh

Following diagram explains how application is interfaced with LCD controller using SPI modules. Whenever application requires to send information to LCD controller, it first updates to Xmit buffer (referred as `gLcdCavas` in the code) and then triggers DMA. DMA transfers this data to LCD controller via SPI.

Figure 4-10. Application Architecture – SPI



4.2.2.1. Hardware Init

LCD Initialization

The `LCDD_Initialize` function initializes IO pins, SPI configuration, DMA for Memory to Peripheral transfers among SPI, transmit buffer and receive buffer and performs register configurations on LCD controller (ILI9488). Following table explains various function calls associated with this function.

Table 4-7. LCDD_Initialize

Function	Description
`ILI9488_EbiInitialize`	Initializes ILI9488 interface in SPI mode
`_ILI9488_Spi_HW_Initialize`	Pin configurations for ILI9488 hardware, includes SPI CS, MISO, MOSI, SCLK, Reset, Command/ Data and Back light signals (PWM)
`ILI9488_SpiInitializeWithDma`	Initializes ILI9488 driver with DMA support
`_ILI9488_SpiDmaInitialize`	Initializes ILI9488 DMA structure and corresponding DMA driver
`_ILI9488_SpiDmaConfigChannels`	Initializes DMA TX & RX channels for Peripheral transfers and call back routines to Process DMA driver
`_ILI9488_SpiDmaConfigureRxTx`	Configures DMA TX channel with • Destination address as SPI TDR address • Transfer type as Peripheral transfer • Single Micro block with chunk size as 1 • Data width as 16-bits • Incremented start address and Fixed destination address Same way RX channel with • Source address as SPI RDR address • Transfer type as Peripheral transfer • Single Micro block with chunk size as 1 • Data width as 16-bits • Fixed start address and Incremented destination address TX channel source address and RX channel destination address will be updated in run time.
`ILI9488_SpiSendCommand`	This module will be called multiple times based on required register settings on LCD controller.

System Tick Timer Initialization

The `TimeTick_Configure` function initializes System tick timer to perform various timings related activities and wait times based on need.

4.2.2.2. Screens Update

Screen initialization includes defining canvas region buffer, buffer size, updating COLUMN ADDRESS SET and PAGE ADDRESS SET registers of LCD controller.

```
LCDD_SetCavasBuffer         /* Initializes canvas region buffer and its size */
LCDD_SetUpdateWindowSize    /* Updates LCD size parameters to LCD controller
registers */
```

On completing initialization, application continues displaying patterns on the LCD.

```
#define CANVAS_LCD_WIDTH    240   /* 3/4 of LCD Width i.e. 320 * (3/4) = 240 */
#define CANVAS_LCD_HEIGHT   360   /* 3/4 of LCD Height i.e. 480 * (3/4) = 360 */
```

Snippet	Description
`LCDD_DrawRectangleWithFill(0, 0, 0, ANVAS_LCD_WIDTH - 1, CANVAS_LCD_HEIGHT - 1, COLOR_CONVERT(COLOR_WHITE));LCDD_UpdateWindow();`	Draws a rectangle on full LCD and fills with chosen color
`LCDD_DrawRectangleWithFill(0, 0, 0, CANVAS_LCD_WIDTH - 1, CANVAS_LCD_HEIGHT - 1, COLOR_CONVERT(COLOR_BLUE)); LCDD_UpdateWindow();`	Redraws a rectangle on full LCD and fills with different color
`LCD_DrawString(0, 50, 5, String, RGB_24_TO_18BIT(COLOR_BLACK)); LCDD_UpdateWindow();`	Draws a string “LCD Example” starting from mentioned coordinates with chosen color
`LCDD_DrawRectangleWithFill(0, 0, HEADLINE_OFFSET, CANVAS_LCD_WIDTH - 1, CANVAS_LCD_HEIGHT - 1 HEADLINE_OFFSET, COLOR_CONVERT(COLOR_WHITE)); LCDD_UpdateWindow();`	Draws a rectangle below Header string with chosen color

Snippet	Description
`LCDD_DrawRectangleWithFill(0, 4, 4 + HEADLINE_OFFSET, CANVAS_LCD_WIDTH - 5 - 4, CANVAS_LCD_HEIGHT - 5 - HEADLINE_OFFSET, COLOR_CONVERT(COLOR_BLACK)); LCDD_UpdateWindow();`	Draws a rectangle starting from mentioned coordinates with chosen color
`LCDD_DrawRectangleWithFill(0, 8, 8 + HEADLINE_OFFSET, CANVAS_LCD_WIDTH - 9 - 8, CANVAS_LCD_HEIGHT- 9 - 8 - HEADLINE_OFFSET, COLOR_CONVERT(COLOR_BLUE)); LCDD_UpdateWindow();`	
`LCDD_DrawRectangleWithFill(0, 12, 12 + HEADLINE_OFFSET, CANVAS_LCD_WIDTH - 13 - 12,CANVAS_LCD_HEIGHT - 13 - 12 - HEADLINE_OFFSET, COLOR_CONVERT(COLOR_RED)); LCDD_UpdateWindow();`	
`LCDD_DrawRectangleWithFill(0, 16, 14 + HEADLINE_OFFSET, CANVAS_LCD_WIDTH - 17 - 16,CANVAS_LCD_HEIGHT - 17 - 14 - HEADLINE_OFFSET, COLOR_CONVERT(COLOR_GREEN)); LCDD_UpdateWindow();`	
`LCDD_DrawLine(0, 0, CANVAS_LCD_HEIGHT / 2, CANVAS_LCD_WIDTH -1, CANVAS_LCD_HEIGHT / 2, COLOR_CONVERT(COLOR_RED)); LCDD_UpdateWindow();`	Draws a horizontal line between mentioned coordinates with chosen color
`LCDD_DrawLine(0, CANVAS_LCD_WIDTH / 2, HEADLINE_OFFSET, CANVAS_LCD_WIDTH / 2, CANVAS_LCD_HEIGHT-1, COLOR_CONVERT(COLOR_RED)); LCDD_UpdateWindow();`	
`LCDD_DrawLine(0, 0, 0 , CANVAS_LCD_WIDTH -1, CANVAS_LCD_HEIGHT - 1, RGB_24_TO_RGB565(COLOR_RED)); LCDD_UpdateWindow();`	
`LCDD_DrawLine(0, 0, CANVAS_LCD_HEIGHT - 1, CANVAS_LCD_WIDTH - 1, 0, RGB_24_TO_RGB565(COLOR_RED)); LCDD_UpdateWindow();`	

Snippet	Description

`LCDD_DrawRectangle(0,`
`CANVAS_LCD_WIDTH / 4,`
`CANVAS_LCD_HEIGHT / 4,`
`CANVAS_LCD_WIDTH * 3 / 4 -`
`CANVAS_LCD_WIDTH / 4,`
`CANVAS_LCD_HEIGHT * 3 / 4 -`
`CANVAS_LCD_HEIGHT / 4,`
`COLOR_CONVERT(COLOR_RED));`
`LCDD_UpdateWindow();`
	Draws a rectangle starting from mentioned coordinates with chosen color

`LCDD_DrawRectangle(0,`
`CANVAS_LCD_WIDTH / 3,`
`CANVAS_LCD_HEIGHT / 3,`
`CANVAS_LCD_WIDTH * 2 / 3 -`
`CANVAS_LCD_WIDTH / 3,`
`CANVAS_LCD_HEIGHT * 2 / 3 -`
`CANVAS_LCD_HEIGHT / 3,`
`COLOR_CONVERT(COLOR_RED));`
`LCDD_UpdateWindow();`	

`LCD_DrawFilledCircle(0,`
`CANVAS_LCD_WIDTH * 3 / 4,`
`CANVAS_LCD_HEIGHT * 3 / 4,`
`CANVAS_LCD_WIDTH / 4,`
`COLOR_CONVERT(COLOR_BLUE));`
`LCDD_UpdateWindow();`
	Draws a circle with radius `CANVAS_LCD_WIDTH / 4` of and fills with chosen color

`LCD_DrawFilledCircle(0,`
`CANVAS_LCD_WIDTH / 2,`
`CANVAS_LCD_HEIGHT / 2,`
`CANVAS_LCD_HEIGHT / 4,`
`COLOR_CONVERT(COLOR_WHITE));`
`LCDD_UpdateWindow();`	

`LCD_DrawFilledCircle(0,`
`CANVAS_LCD_WIDTH / 4,`
`CANVAS_LCD_HEIGHT * 3 / 4,`
`CANVAS_LCD_HEIGHT / 4,`
`COLOR_CONVERT(COLOR_RED));`
`LCDD_UpdateWindow();`	

`LCD_DrawFilledCircle(0,`
`CANVAS_LCD_WIDTH * 3 / 4,`
`CANVAS_LCD_HEIGHT / 4,`
`CANVAS_LCD_WIDTH / 4,`
`COLOR_CONVERT(COLOR_YELLOW));`
`LCDD_UpdateWindow();`	

4.2.2.3. Screens Refresh

At the end, application continuously moves image across the screen.

```
/* Draws a rectangle and fills with chosen color */
LCDD_DrawRectangleWithFill(0, 0, 0, CANVAS_LCD_WIDTH - 1, CANVAS_LCD_HEIGHT - 1,
COLOR_CONVERT(COLOR_BLACK));

/* Draws a pre-loaded image (Atmel logo) at varying offsets */
LCDD_DrawImage(0, i, j, (LcdColor_t *)gImageBuffer, (i + DEMO_IMAGE_WIDTH), (j +
DEMO_IMAGE_HEIGHT));
LCDD_UpdateWindow();
```

Refer to `lcd` solution for complete details.

4.3. Memory Footprint

This sections provides information about memory utilization in both EBI and SPI projects.

Note:

- This is just an information and not for comparison. Because these two projects are implemented in different way to illustrate functionality.
- Optimization is not enabled for both projects.
- ARM/GNU C Compiler version : 4.9.3

Table 4-8. Footprints for EBI and SPI Interface Modules

Interface Modules	Program Memory Usage	Data Memory Usage
EBI	85784 bytes (4.1 % Full)	92704 bytes (23.6 % Full - LCD Split into 4 blocks of 160x280 each)
SPI	83824 bytes (4.0 % Full)	274568 bytes (69.8 % Full (Reduced to 3/4th i.e. 240x360)

5. References

SAMV71 Xplained Ultra User Guide - http://www.atmel.com/images/atmel-42408-samv71-xplained-ultra_user-guide.pdf

maXTouch Xplained Pro User Guide - http://www.atmel.com/Images/Atmel-42350-maXTouch-Xplained-Pro_User-Guide.pdf

6. Revision History

Doc. Rev.	Date	Comments
42646A	01/2016	Initial document release.

Atmel | Enabling Unlimited Possibilities®

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