

Using the Segment LCD Low Power Features on the SAM4L-EK

AN-4558

Prerequisites

- **Hardware Prerequisites**
 - Atmel® SAM4L-EK Evaluation kit
- **Software Prerequisites**
 - Atmel Studio 6.2
 - Atmel Software Framework 3.17.0 or higher
 - Latest J-Link/SAM-ICE™ Software and Documentation Pack
- **Estimated completion time:** 45min

Introduction

The goal of this hands-on is to:

- Describe the main features of the Segment LCD Controller (LCDCA)
- Understand how to configure and use them to maximize power efficiency

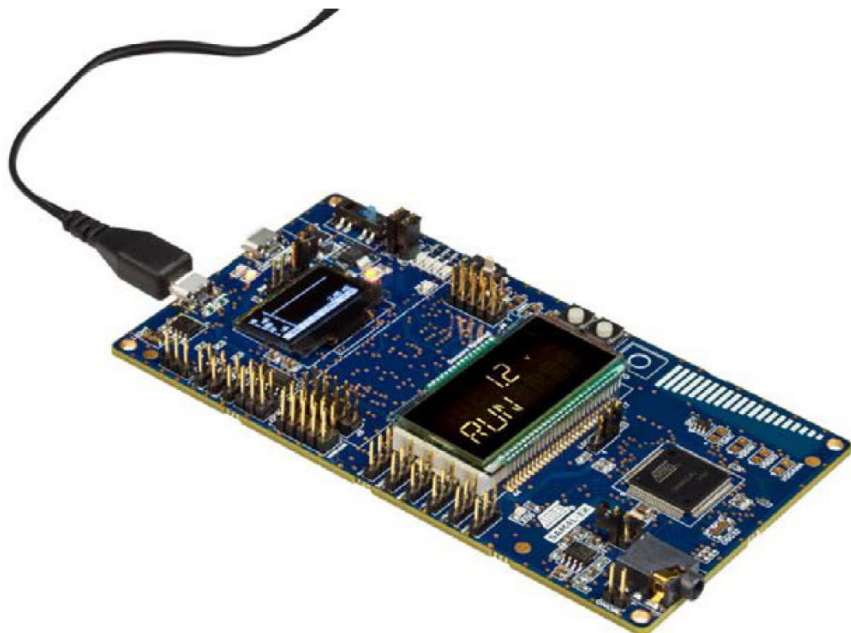








Table of Contents

| | |
|--|----|
| Prerequisites | 1 |
| Introduction | 1 |
| Icon Key Identifiers | 3 |
| 1. Training Module Architecture | 4 |
| 1.1 Atmel Studio Extension (.vsix) | 4 |
| 1.2 Atmel Training Executable (.exe) | 4 |
| 2. Introduction to Segment LCD Controller (LCDCA) Low Power Features | 5 |
| 2.1 LCDCA Clocks | 5 |
| 2.2 LCDCA Power Management | 5 |
| 3. Assignment 1: Create and Configure Your Project | 6 |
| 4. Assignment 2: Segment LCD Controller Initialization | 11 |
| 5. Assignment 3: ASCII Character Mapping | 18 |
| 6. Assignment 4: Frame Frequency Flexibility | 21 |
| 7. Assignment 5: Low Power Waveform | 23 |
| 8. Assignment 6: Configure the LCDCA to Reach the Lowest Power Consumption | 24 |
| 9. Assignment 7: Hardware Blinking Feature | 25 |
| 10. Assignment 8: Software Contrast Adjustment Control | 27 |
| 11. Conclusion | 28 |
| 12. Revision History | 29 |

Icon Key Identifiers

Icons are used to identify different assignment sections and reduce complexity. These icons are:

| | | |
|---|----------------|---|
|  | INFO | Delivers contextual information about a specific topic. |
|  | TIPS | Highlights useful tips and techniques. |
|  | TO DO | Highlights objectives to be completed. |
|  | RESULT | Highlights the expected result of an assignment step. |
|  | WARNING | Indicates important information. |
|  | EXECUTE | Highlights actions to be executed out of the target when necessary. |

1. Training Module Architecture

This training material can be retrieved through different Atmel deliveries:

- As an Atmel Studio Extension (.vsix file), which can be found on the Atmel Gallery web site (<http://gallery.atmel.com/>) or using the Atmel Studio Extension manager
- As an Atmel Training Executable (.exe file) usually provided during Atmel Training sessions

Depending on the delivery type, the different resources needed to complete this training (hands-on documentation, datasheets, application notes, software, and tools) will be found on different locations.

1.1 Atmel Studio Extension (.vsix)

Once the extension has been installed, you can open and create the different projects associated with the training using the “*New Example Project from ASF...*” menu in Atmel Studio.



INFO

The example projects installed through an extension are usually under “*Atmel Training > Atmel Corp. Extension Name*”.

There are different projects which can be available depending on the extension:

- **Hands-on Documentation:** contains the documentation as required resources
- **Hands-on Assignment:** contains the initial project that may be required to start
- **Hands-on Solution:** contains the final application, which is a solution project for this hands-on



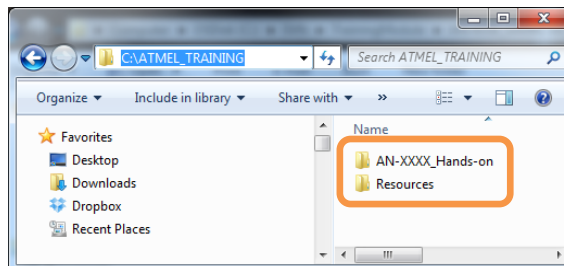
INFO

Each time a reference is made to some resources in the following pages, the user must refer to the **Hands-on Documentation** project folder.

1.2 Atmel Training Executable (.exe)

Depending on where the executable has been installed, you will find the following architecture which is comprised of two main folders:

- **AN-XXXX_Hands-on:** contains the initial project that may be required to start and a solution
- **Resources:** contains required resources (datasheets, software, and tools...)



INFO

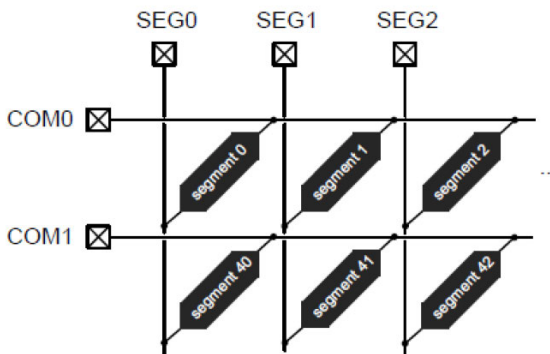
Unless a specific location is specified, each time a reference is made to some resources in the following pages, the user must refer to this **Resources** folder.

2. Introduction to Segment LCD Controller (LCDCA) Low Power Features

A Segment LCD display is made of several segments which can be visible or invisible.

A segment has two electrodes with liquid crystal between them. These electrodes are the common terminal (COM) and the segment terminal (SEG).

When a voltage above a threshold voltage is applied across the liquid crystal, the segment becomes visible.



Our Segment LCD controller (LCDCA) is intended for monochrome passive liquid crystal display (LCD) with up to four Common terminals and up to 40 Segments terminals.



INFO

The SAM4L-EK LCD has four Common and 40 Segment terminals.

2.1 LCDCA Clocks

The LCDCA peripheral has two different clocks:

- CLK_LCDCA: the LCDCA Bus Interface Clock (APB clock), managed by the Power Manager (PM)
- CLK_LCD: the LCD 32kHz clock, managed by the Backup Power Manager (BPM). This clock allows the LCD Controller to run in the different Power Save Modes.

2.2 LCDCA Power Management

The Power Management can control the LCD display while CLK_LCDCA is disabled but stops functioning when CLK_LCD (32kHz) is disabled.

Several features are supported to offload the CPU, reduce interrupts and reduce power consumption.

The power consumption of LCDCA itself can be minimized by:

- Using the lowest acceptable frame rate (refer to the LCD technical characteristics)
- Using the low power waveform mode (default mode)
- Configuring the lowest possible contrast value

In addition to this, other features such as ASCII Character mapping or Segment Blink functionality can be used to reduce interrupts number and offload the CPU. This will allow making the application much more power efficient.



INFO

This Hands-on will describe how to configure and use these different low power features.

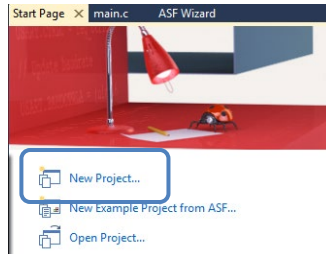
3. Assignment 1: Create and Configure Your Project



TO DO

Create the **Hands-on Assignment** project under Atmel Studio.

- Click on New Project... (or File > New > Project...):



- In New Project Window, select “C/C++ > GCC C ASF Board Project”:



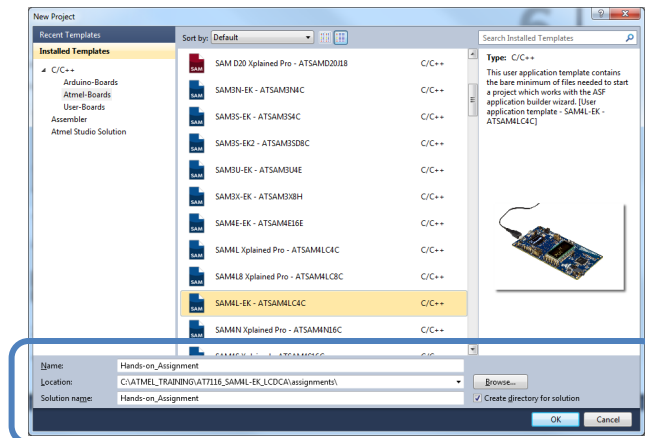
- Fill-in New Project fields according to following use cases:

Atmel Training Executable Case

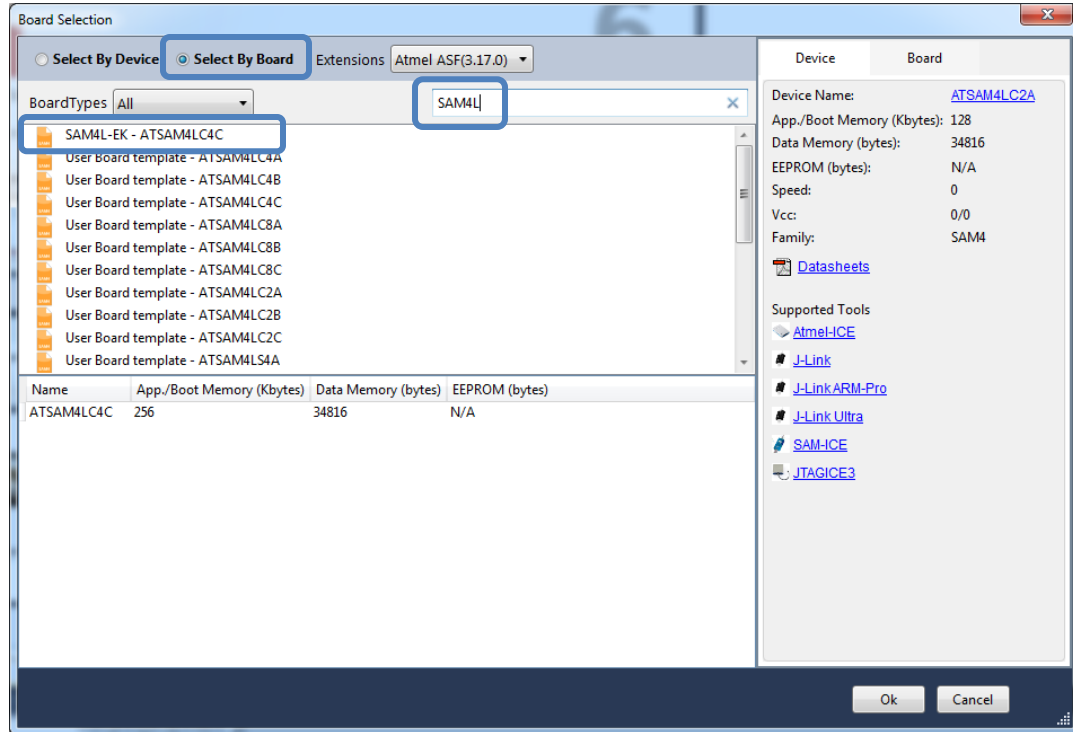
- Name:** Hands-On Assignment
- Location:** “AN-4558_SAM4L-EK_LCDCA\assignments”
(relative path in the ATMEL_TRAINING installation folder)
- Solution name:** Hands-On Assignment
- Click OK

Atmel Extension Case (downloaded from Atmel Gallery or Studio Extension Manager)

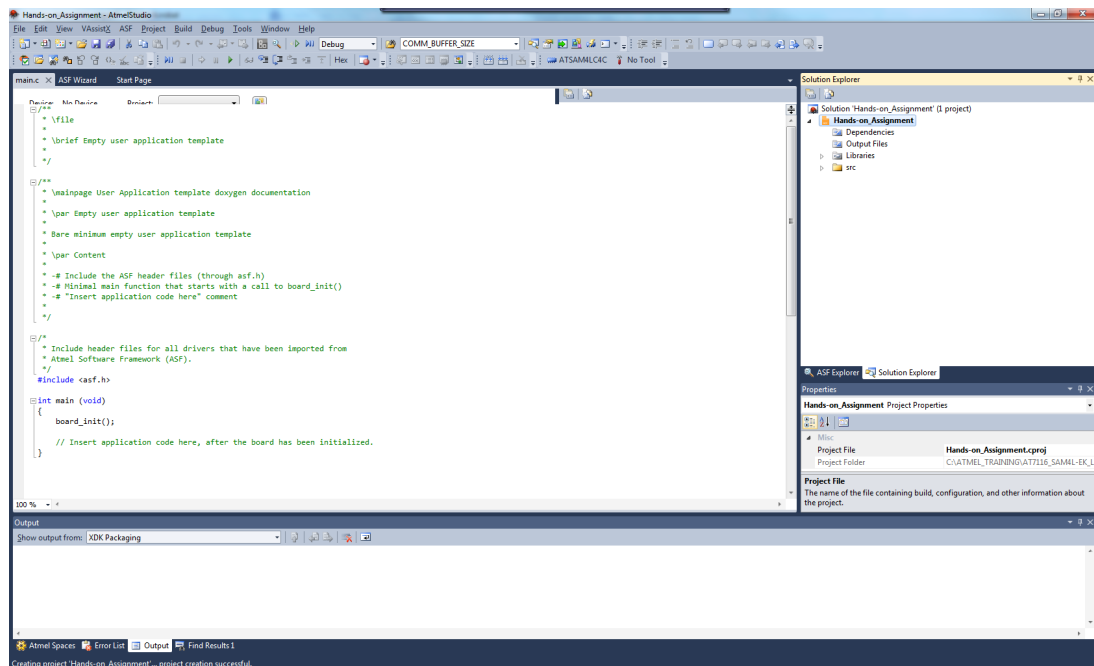
- Name:** Hands-On Assignment
- Location:** existing Hands-on Documentation solution path
- Solution name:** Hands-On Documentation
- Click OK



- In Board selection window, click on Select By Board and type SAM4L in the search field. Then select SAM4L-EK – ATSAM4LC4C board and Click OK:



RESULT “Hands-on Assignment” project is loaded with default settings and source files.





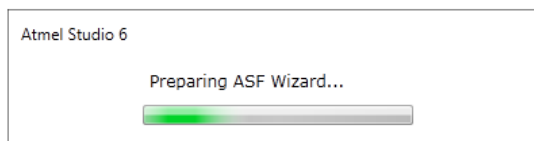
TO DO Add LCD Display Support using ASF Wizard.

- Click on the ASF Wizard icon or right click on the Hands-on Assignment project > ASF Wizard:



INFO ASF Wizard is being loaded.

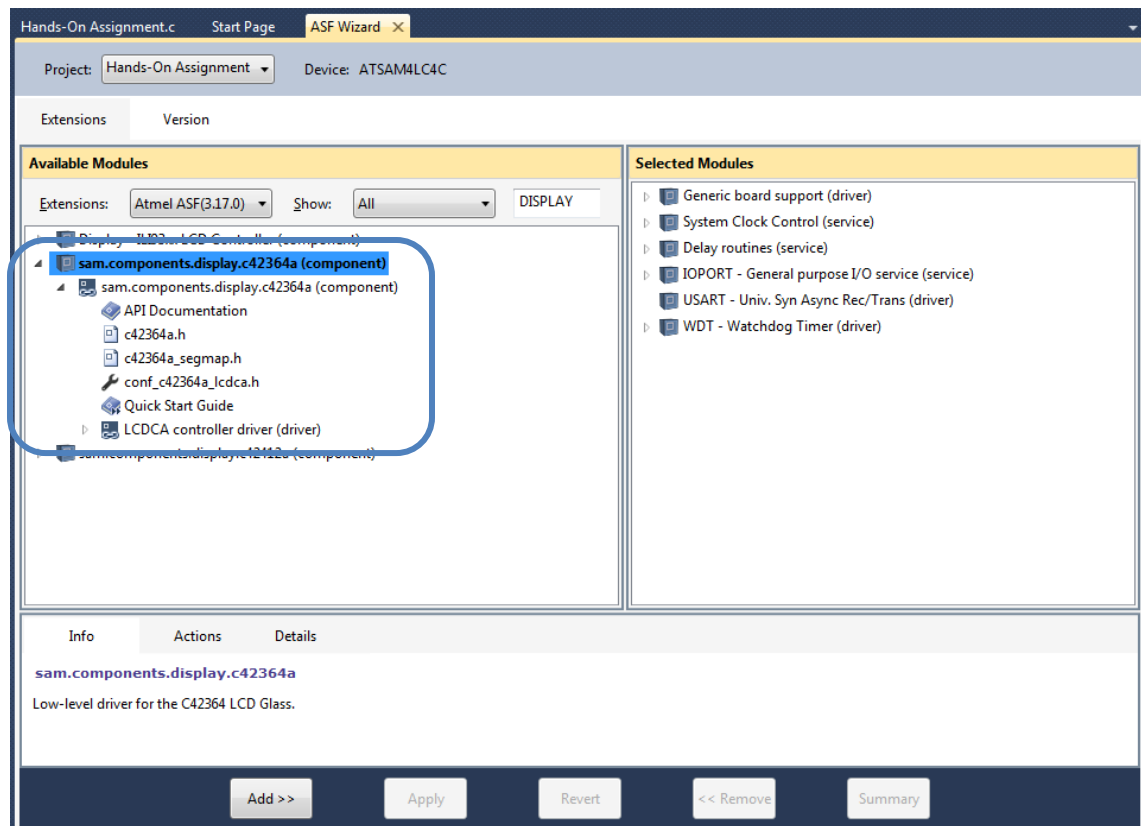
- In Available Modules pane, select as ASF Extensions the latest one available
- Type “Display” in the ASF Wizard Search box



- Select the following components, add them to the Selected Modules (Add >> button):
 - sam.components.display.c42364a (component)



INFO Installing that component will also install the LCDCA Controller driver.

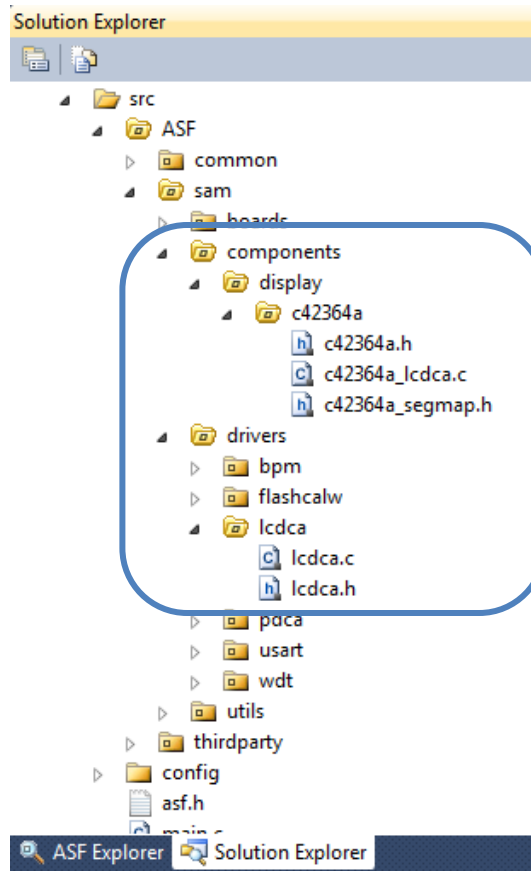


- Click on Apply button to copy all files.



INFO

Once these components have been added, you can check that they have been loaded by looking at components and drivers folders in solution explorer as shown below:



RESULT

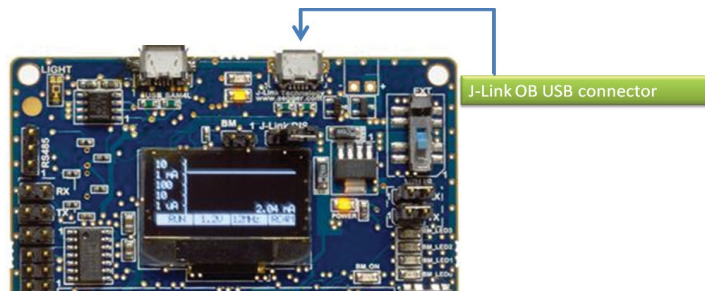
The “Hands-on Assignment” project with required source files and ASF components is now ready to use.



TO DO

Power up the SAM4L-EK.

- Connect the SAM4L-EK board to the computer using the J-Link OB USB connector (J1)





TO DO

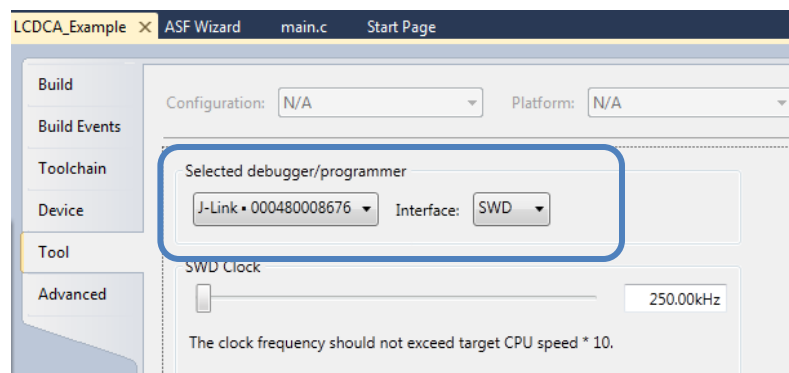
Configure the project to use the Serial Wire Debug Interface (SWD) instead of JTAG.



WARNING

By default, each Studio project is configured to connect to the target using JTAG interface. However, the SAM4L-EK board is designed to use the Serial Wire Debug protocol (SWD) instead of standard JTAG. Thus, the project has to be correctly configured to use the SWD interface in order to be able to program the chip.

- From the solution explorer window, right click on the **Hands-on Assignment** project file and select “Properties”
- Now, select the Tool tab, then select:
 - J-Link as Selected debugger/programmer
 - SWD (Serial Wire Interface) as Interface



- Save the new project properties by clicking on the save button: 



RESULT

The project is now configured to use the SWD interface as debugger/programmer interface.

4. Assignment 2: Segment LCD Controller Initialization

The goal of this assignment is to implement the different functions to correctly initialize the SAM4L Segment LCD Controller.



TO DO SAM4L critical system initializations.

- Open main.c file. In the `main` function, you can see that only `board_init()` function is called

The `board_init()` function initializes the different GPIOs for the board usage.

This function relies on the configuration of `conf_board.h` file located in “src\config”.



INFO There is no `conf_board.h` modifications required for this hands-on.

- Add a call to `sysclk_init()` function at the beginning of project “main” function:

```
#include <asf.h>
int main (void)
{
    sysclk_init();
    board_init();
    // Insert application code here, after the board has been
    initialized
}
```

The `sysclk_init()` function initializes the source clocks (such as OSC0, Fast RC, PLL0...) and switching to the system clock selected by the user.

Then, depending on the new CPU frequency, it configures the best power scaling mode and sets the right number of flash wait states.

This function relies on the configuration of `conf_clock.h` file located in “src\config”.

By calling the `sysclk_init()` function, the following configurations will be performed:

- CPU Clock Frequency (FCPU) = RCSYS = 115kHz (default reset value)
- Peripheral Bridge X (PBx) clock frequency = FCPU = 115kHz (default reset value)
- picoCache Clock is enabled
- Power Scaling Mode 0 (default value at reset) is switched to Power Scaling Mode 1
- Number of Flash Wait State = 0 (default reset value)

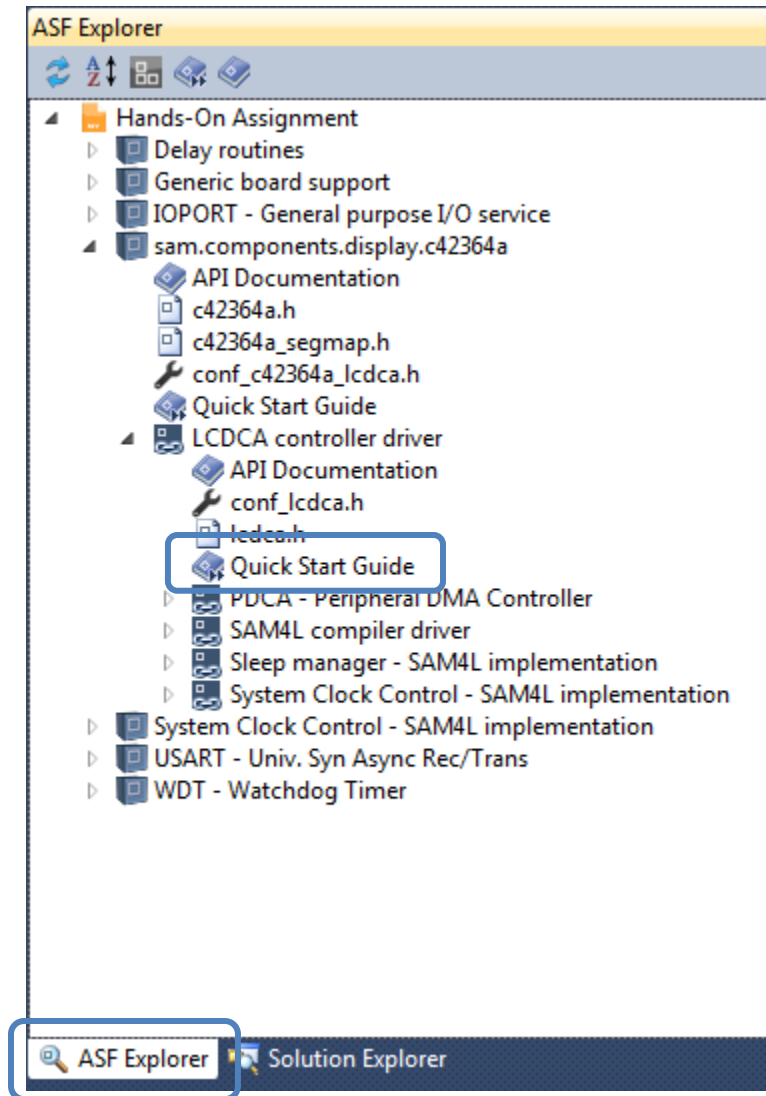


INFO There are no “`conf_clock.h`” modifications required for this hands-on.



RESULT The critical system initializations have now been performed.

The LCDCA Quick Start Guide can be easily accessed by clicking on ASF Explorer and selecting `sam.components.display.c42364a > LCDCA controller driver > Quick Start Guide`:



INFO

The ASF API documentation is exclusively available on the web.

For the LCDCA Quick Start Guide, the direct link for ASF 3.17.0 version is:

http://asf.atmel.com/docs/3.17.0/sam4l/html/sam_drivers_lcdca_quick_start.html

Here is the LCDCA Initialization Workflow extracted from the LCDCA Quick Start Guide:



INFO

The following configuration is given as a reference example but must **NOT** be coded/implemented for now.

1. Initialize LCDCA clock.

```
• lcdca_clk_init();
```

2. Set basic LCDCA configuration.

```
• lcdca_set_config(&lcdca_cfg);
```

Where `lcdca_cfg` is an instance of the structure “`lcdca_config`”, defined below (we will learn later what are their different meaning and use):

- | | | |
|------------------------|-------------------------|---|
| • <code>uint8_t</code> | <code>port_mask</code> | Number of SEG used. |
| • <code>bool</code> | <code>x_bias</code> | External bias (false: internal gen, true: external gen). |
| • <code>bool</code> | <code>lp_wave</code> | Wave mode (false: lowpower waveform, true: standard waveform) |
| • <code>uint8_t</code> | <code>duty_type</code> | Duty type selection. |
| • <code>uint8_t</code> | <code>lcd_clkdiv</code> | Divider of the prescaled clock source. |
| • <code>uint8_t</code> | <code>lcd_pres</code> | prescaler of the clock source. |
| • <code>uint8_t</code> | <code>fc0</code> | Frame Counter 0. |
| • <code>uint8_t</code> | <code>fc1</code> | Frame Counter 1. |
| • <code>uint8_t</code> | <code>fc2</code> | Frame Counter 2. |
| • <code>int8_t</code> | <code>contrast</code> | -32 <= signed contrast value <= 31. |

- **Bias:** to operate correctly, LCD controller requires a reference level

The External BIAS bit (XBIAS) selects the source of VLCD. If XBIAS is zero, VLCD sources voltages from the internal band gap reference. Otherwise, VLCD must be powered externally.



INFO

The internal reference is used by the SAM4L-EK LCD.

- **Duty Type:** this parameter is related to the number of Common Terminals



INFO

The SAM4L-EK LCD has four common terminals and so a ¼ duty is needed.

- **Frame Counters:** for several functions such as blinking modes, a frame counter is used to create a time base. There are three independent frame counters (FC0, FC1 and FC2) which can be associated to any function.
- **Other structure parameters** will be the object of specific assignments. We will come back later on their use.



INFO

LCD datasheet can be found in the Hands-on Documentation project (.vsix delivery) or directly in “`AN-4558_SAM4L-EK_LCDCA\assignments`” folder (.exe delivery).

3. Enable LCDCA module.

- `lcdca_enable();`

4. Enable frame counter timers according to your application.

- `lcdca_enable_timer(LCDCA_TIMER_FC0);`
- `lcdca_enable_timer(LCDCA_TIMER_FC1);`
- `lcdca_enable_timer(LCDCA_TIMER_FC2);`

5. Turn on LCD back light.

- `ioport_set_pin_level(LCD_BL_GPIO, IOPORT_PIN_LEVEL_HIGH);`

We will now create the LCDCA Init function by using pre-implemented code from an existing ASF Example.



TO DO

LCDCA Initialization function implementation.

- In the main.c file, create a new function “static void lcdca_init(void)”:

```
#include <asf.h>
static void lcdca_init(void)
{
}

int main (void)
{
    sysclk_init();
    board_init();
    // Insert application code here, after the board has been
    initialized
}
```

- Create the following global variable above `lcdca_init` function:

```
struct lcdca_config lcdca_cfg;

static void lcdca_init(void)
{
}
```



INFO

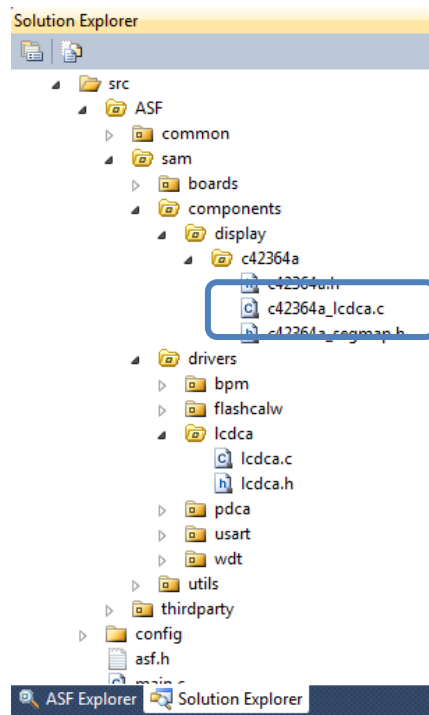
This variable is an instance of the structure called `lcdca_config` defined in `src/ASF/sam/drivers/lcdca/lcdca.h`.

- Include `conf_c42364a_lcdca.h` header file:

```
#include <asf.h>
#include "conf_c42364a_lcdca.h"
```

We will now get a LCDCA Initialization implementation from the existing C42364A LCD Glass module component driver:

- Open the “c42364a_lcdca.c” file from the Solution Explorer:



- **COPY** the following code used for the lcdca initialization which is part of the `c42364a_init()` function (do NOT copy the `struct lcdca_config lcdca_cfg;` line):

```
lcdca_clk_init();  
lcdca_cfg.port_mask = C42364A_PORT_MASK;  
lcdca_cfg.x_bias = CONF_C42364A_X_BIAS;  
lcdca_cfg.lp_wave = true;  
lcdca_cfg.duty_type = C42364A_LCD_DUTY;  
lcdca_cfg.lcd_pres = CONF_C42364A_PREP;  
lcdca_cfg.lcd_clkdiv = CONF_C42364A_CLKDIV;  
lcdca_cfg.fc0 = CONF_C42364A_FC0;  
lcdca_cfg.fc1 = CONF_C42364A_FC1;  
lcdca_cfg.fc2 = CONF_C42364A_FC2;  
lcdca_cfg.contrast = CONF_C42364A_CONTRAST;  
lcdca_set_config(&lcdca_cfg);  
lcdca_enable();  
lcdca_enable_timer(LCDCA_TIMER_FC0);  
lcdca_enable_timer(LCDCA_TIMER_FC1);  
lcdca_enable_timer(LCDCA_TIMER_FC2);
```



INFO

We could directly call the `c42364a_init` function but as we will update the LCDCA initialization during the Hands-on, it's better to create a dedicated function and make a copy of the `c42364a_init` one.

- Paste this code into your “`lcdca_init`” function.

- **Add** the following code in `lcdca_init` function just after the LCDCA initialization you just copied:

```
/* Turn on LCD back light */
ioport_set_pin_level(LCD_BL_GPIO, IOPORT_PIN_LEVEL_HIGH);
```

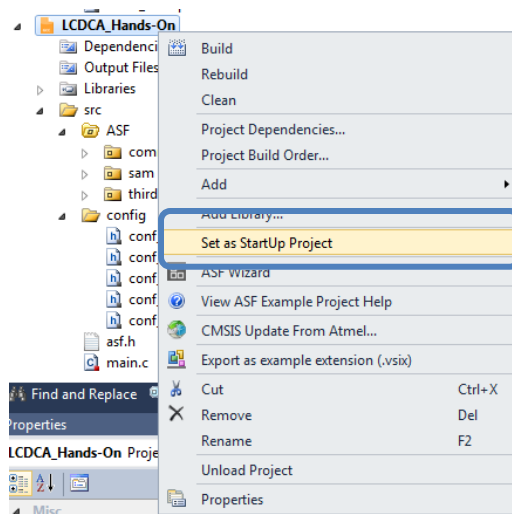
- Call the `lcdca_init` function in your main function:



```
int main (void)
{
    sysclk_init();
    board_init();
    // Insert application code here, after the board has been
    initialized
    lcdca_init();
}
```



TO DO Compile and Debug.

- Check that your working project is the one that will be compiled by right clicking on it from the Solution Explorer view and click on “Set as StartUp Project”:

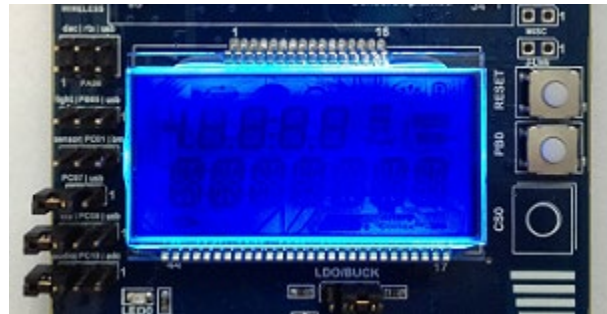


- Click on the “Build” button:  to compile your project
- Then download the program in the internal flash of the SAM4L by clicking on the “Start Without Debugging” or Ctrl+Alt+F5 



RESULT

You should see the LCD switched on (without any characters displayed) which proves you've correctly initialized the SAM4L Segment LCD Controller.



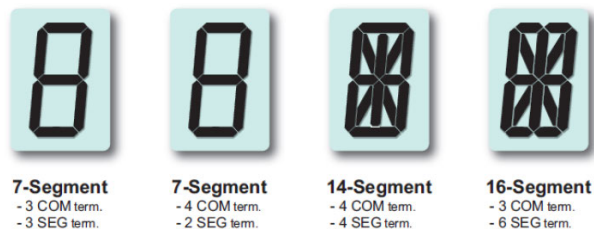
5. Assignment 3: ASCII Character Mapping

LCDCA handles up to four ASCII characters tables, configured in Character Mapping Configuration register (CMCFG).

Instead of handling each segment in display memory for a selected digit, user writes ASCII code in Character Mapping Control Register (CMCR) to display the corresponding character.

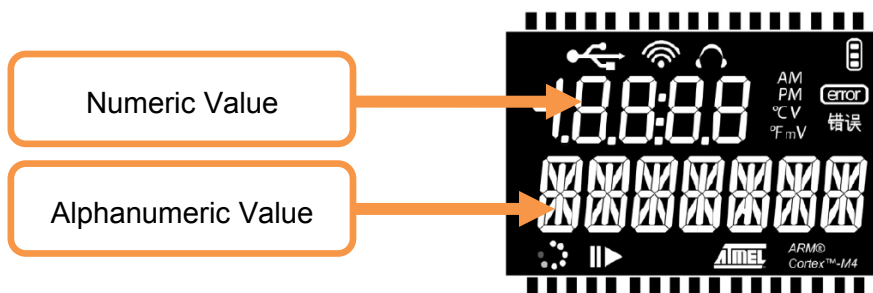
As a consequence, Character mapping will save CPU execution time and allow a fast return to Power Save Mode after display update. This results in an application more power efficient.

Here are the types of Digit supported by our LCD Controller:



INFO

SAM4L-EK Custom Glass Display: this board is equipped with a 4x40 segment LCD which has four 7-Segment with four COM terminals and seven 14-Segment with four COM terminals.



The aim of this assignment is to understand how to use this ASCII Character Mapping.

We will use the `lcdca_write_packet()` function to display ASCII characters using ASCII Character Mapping feature. The function will write the maximum number of byte passed as argument, and will stop writing if a NULL character is found.

```
void lcdca_write_packet(uint8_t lcd_tdg, uint8_t first_seg, const uint8_t
* data, size_t width, uint8_t dir)
```

Where parameters are:

`lcd_tdg` Type of digit decoder.
`first_seg` First SEG where the first data will be written.
`data` Data buffer.
`width` Maximum Number of data.
`dir` Direction (==0: Left->Right, !=0: Left<-Right).

Examples:

```
// Display in numeric field (7-Segment)
lcdca_write_packet(LCDCA_TDG_7SEG4COM, C42364A_FIRST_7SEG_4C, data,
C42364A_WIDTH_7SEG_4C, LCDCA_CMCFG_DREV_LEFT);

// Display in alphanumeric field (14-Segment)
lcdca_write_packet(LCDCA_TDG_14SEG4COM, C42364A_FIRST_14SEG_4C, data,
C42364A_WIDTH_14SEG_4C, LCDCA_CMCFG_DREV_LEFT);
```



TO DO Display ASCII Characters on the LCD using LCDCA controller driver functions.

- Using above examples, reproduce the following display on your SAM4L-EK by implementing the above functions in the main function



TIPS You just need to replace the two “`data`” fields by “`-01-`” and “`HANDSON`” characters.

- Now build the solution and program the target



RESULT You have correctly displayed some ASCII Characters using the ASCII Character Mapping feature of the SAM4L Segment LCD.



TO DO

OPTIONAL Task: Display ASCII Characters on the LCD using Display – C42364A LCD functions.

It's possible to simplify the code by directly using functions from the Display – C42364A LCD component:

Usage Example:

```
// Write string to c42048a lcd numeric field.  
c42364a_write_num_packet (const uint8_t * data);  
    Where data Pointer to the input string (max length is 4)  
  
// Write string to c42048a lcd alphanumeric field.  
c42364a_write_alphanum_packet (const uint8_t * data);  
    Where data Pointer to the input string (max length is 7)
```

- Using Usage Example above, reproduce the same display on your SAM4L-EK
- Now build the solution and program the target



RESULT

You have correctly displayed some ASCII Characters using the ASCII Character Mapping feature of the SAM4L Segment LCD.

6. Assignment 4: Frame Frequency Flexibility

The power consumption of LCDCA can be minimized by using the lowest acceptable frame rate.

A frame rate outside the LCD specification may add flickering so a trade off may be considered based on the display quality of the application.

Minimum and maximum supported frame rates have to be checked in the following two documents:

- The SAM4L Electricals section of the product datasheet to get the minimum and maximum LCD Frame Frequencies supported by our LCDCA Peripheral:

| | | | | | | |
|-------------|---------------------|--------------|-------|--|-----|----|
| f_{frame} | LCD Frame Frequency | F_{CLKLCD} | 31.25 | | 512 | Hz |
|-------------|---------------------|--------------|-------|--|-----|----|

- The LCD datasheet



INFO

The SAM4L-EK LCD is a custom one that has been developed to fit with the minimum and maximum values of the SAM4L LCDCA Peripheral characteristics.

The aim of this assignment is to implement the minimum and maximum frame rate configurations and check the resulting LCD Display.

The LCD Frame Rate is computed using the following formula:

$$FrameRate = \frac{F(CLK_LCD)}{K * N * (1 + CLKDIV) * 2^{1-WMOD}}$$

Where:

- $F(CLK_DIV)$ is the LCD 32kHz clock
- K is factor related to the LCD Duty Cycle (example: $K = 8$ if Duty Cycle is $\frac{1}{4}$)
- N is the LCD Prescaler divider (8 or 16)
- $CLKDIV$ is the Clock Division field (0 to 7 as possible values)
- $WMOD$ is the Waveform Mode (0: low power waveform mode, 1: standard waveform mode)



INFO

From the LCD specification which has four common terminals, a $\frac{1}{4}$ duty is needed, so $K = 8$.

In the current configuration, we have configured the Low Power Waveform Mode. This Waveform mode must not be modified during this assignment.

As a consequence, the Frame Rate Formula becomes:

$$Frame Rate = \frac{32000}{8 * N * (1 + CLKDIV) * 2}$$



WARNING $N = 16$ and $CLKDIV=7$ will give Frame Rate = 15.625Hz which is NOT supported.

The minimum Frame Rate will be obtained with $N = 8$ and $CLKDIV = 7$: *Frame Rate min* = 31.25Hz

The maximum Frame Rate will be obtained with $N = 8$ and $CLKDIV = 0$: *Frame Rate max* = 250Hz



TO DO

Configure the Frame Rates to 31.25Hz and 250Hz and Measure the resulting current consumption.

Frame Rate must be updated using the two following parameters (CLKDIV, N) from `lcda_cfg` structure in `lcdca_init` function:

- `uint8_t` `lcd_clkdiv` Divider of the prescaled clock source.
- `uint8_t` `lcd_pres` prescaler of the clock source.

Where:

`lcdca_cfg lcd_clkdiv`: CLKDIV so from 0 to 7

`lcdca_cfg lcd_pres`: false (N = 8) or true (N = 16)

- Make the appropriate update to fill-in the following table by configuring the different LCD Waveform modes:

| LCD Frame Rate | SAM4L Current consumption [μ A] |
|---|--------------------------------------|
| 31.25Hz (clkdiv = and pres =) | |
| 250Hz (clkdiv = and pres =) | |



TIPS

Such assignments require current consumption measurements. They have to be done using the On-Board Monitor display.



WARNING

To measure the correct current consumption, you have to close the debug session or you will have extra current related to the debug link.



INFO

You will notice that there is no flickering with the two tested frequencies. In an application context where the overall power consumption must be reduced as much as possible, the lowest frequency should be used.



RESULT

You have been able to test the minimum and maximum frame rates and measure the impact of these two configurations on the overall current consumption of the LCDCA.

7. Assignment 5: Low Power Waveform

To reduce toggle activity and hence power consumption, it is possible to configure a specific mode called Low Power Waveform.

The low power waveform period is then twice the standard waveform period.



WARNING The LCD must support this mode. This is the case on our SAM4L-EK.



TO DO Check Low Power Waveform mode influence on power consumption with a Frame Rate equal to 250Hz.

Waveform mode is configured using one parameter of the `lcdca_config` structure:

- `bool lp_wave` Wave mode (true: lowpower waveform, false: standard waveform)

As a consequence, here are the two possible configurations for that parameter:

```
lcdca_cfg.lp_wave = true; // Low Power Waveform Mode  
lcdca_cfg.lp_wave = false; // Standard Waveform Mode
```

- Make the appropriate update to fill-in the following table by configuring the different LCD Waveform modes:

| Waveform Mode | SAM4L Current consumption [μ A] |
|--------------------|--------------------------------------|
| Low Power Waveform | |
| Standard Waveform | |



RESULT You have been able to measure the impact of the Low Power Waveform mode on the LCDCA current consumption for a given Frame Rate.

8. Assignment 6: Configure the LCDCA to Reach the Lowest Power Consumption



TO DO

Based on the different previous assignments, find the best parameter combinations to reach the lowest LCDCA current consumption in the specified running mode (RUN mode with RCSYS running).



TIPS

Use the lowest Frame Rate and the LCD Low Power Waveform Mode.

| lcda_cfg Parameters | SAM4L Current consumption [μ A] |
|--------------------------------------|--------------------------------------|
| <code>lcda_cfg.lp_wave = ?</code> | |
| <code>lcda_cfg.lcd_clkdiv = ?</code> | |
| <code>lcda_cfg.lcd_pres = ?</code> | |

9. Assignment 7: Hardware Blinking Feature

The aim of this optional assignment is to understand how to configure the Segment LCD Hardware Blinking feature and check its impact on the current consumption.



TO DO

Implement the Hardware Blinking Feature.

Here is the LCDCA Hardware Blinking Initialization Workflow from the [online](http://asf.atmel.com/docs/3.17.0/sam4l/html/sam_drivers_lcdca_quick_start.html) LCDCA Quick Start Guide, section “Using Hardware Blinking”:

http://asf.atmel.com/docs/3.17.0/sam4l/html/sam_drivers_lcdca_quick_start.html



INFO

The following configuration is given as a reference example but must **NOT** be coded/implemented for now.

1. Create an instance of the Hardware Blinking structure “`lcdca_blink_config`” defined below:

```
• struct lcdca_blink_config blink_cfg;
```

2. Configure the structure parameters which are:

```
• uint8_t lcd_blink_mode Blink Mode selected (0: blink all SEG, 1: blink selected SEG)
• uint8_t lcd_blink_timer LCD Blink Timer Selected (Frame Counter 0, 1 or 2).
```

For several functions such as blinking modes, a frame counter is used to create a time base.

There are three independent frame counters (FC0, FC1 and FC2) which can be associated to any function.



INFO

FC0, FC1 and FC2 have already been initialized in the `lcdca_init` function using their dedicated parameters (`fc0`, `fc1`, `fc2`) in `lcdca_config` structure.

3. Set LCDCA hardware blinking configuration.

```
• lcdca_blink_set_config(&blink_cfg);
```

4. Enable Blinking Feature.

```
• lcdca_blink_enable();
```

We will now implement the Hardware Blinking mode by blinking all LCD Segments at a specified frequency.

- Create the following global variable `blink_cfg` below `lcdca_cfg` one:

```
#include <asf.h>
struct lcdca_config lcdca_cfg;
struct lcdca_blink_config blink_cfg;
```
- In your **main()** function, configure the `blink_cfg` structure to have ALL Segments blinking and using Frame Counter 1 (FC1):

```
blink_cfg.lcd_blink_timer = LCDCA_TIMER_FC1;
blink_cfg.lcd_blink_mode = LCDCA_BLINK_FULL;
```
- Set the hardware blinking configuration:

```
lcdca_blink_set_config(&blink_cfg);
```
- Finally, enable the Blinking feature:

```
lcdca_blink_enable();
```
- Now build the solution and program the target



INFO

You can simply change the blinking frequency by using another Frame Counter as FC0, FC1 and FC2 have pre-defined different values (`lcdca_init` function).

The following formula is used to compute the Blinking Frequency f_{FCx} :

$$f_{FCx} = \frac{\text{FrameRate}}{(\text{TIM.FCx} \times 8) + 1}$$

Where TIM.FCx is the number of frames before rollover.

The blink frequency is defined by the number of frames (FCx in TIM register) between each state ON/OFF. So after FCx+1 frames, the segment will change state.



RESULT

You have correctly implemented the Hardware Blinking feature.

10. Assignment 8: Software Contrast Adjustment Control

Contrast is defined by the maximum value of VLCD. The higher value is, the higher contrast is.

Fine Contrast value (FCST) in CFG register is a signed value (two's complement) which defines the maximum voltage VLCD on segment and common terminals. New value takes effect at the beginning of next frame.

$$VLCD = 3V + (FCST * 0.016V)$$



TO DO

Play with the Contrast adjustment control feature.

Contrast value is configured using one parameter of the `lcdca_config` structure:

```
• int8_t contrast    -32 <= signed contrast value <= 31.
```

This parameter will be used to program the FCST field which is a 6-bit field.

As a consequence, here are the two minimum/maximum configurations for that parameter:

```
lcdca_cfg.contrast = -32;  
lcdca_cfg.contrast = 31;
```

- Make the appropriate updates to test the limit configurations (in `lcdca_init` function)
 - Minimum Contrast Value Result:



- Maximum Contrast Value Result:



RESULT

You have been able to play with the contrast adjustment control feature.

11. Conclusion

In this Hands-on, you have discovered the main features of the Segment LCD Controller (LCDCA) and how to configure and use them to make the application more power efficient.

Dedicated Low Power Waveform, Contrast Control, ASCII Character Mapping are defined to offload the CPU, reduce interrupts, and reduce power consumption.

12. Revision History

| Doc. Rev. | Date | Comments |
|-----------|---------|--|
| 42222B | 07/2014 | Porting to Atmel Studio 6.2/ASF 3.17.0 |
| 42222A | 12/2013 | Initial document release |



Enabling Unlimited Possibilities®

Atmel Corporation

1600 Technology Drive
San Jose, CA 95110
USA

Tel: (+1)(408) 441-0311

Fax: (+1)(408) 487-2600

www.atmel.com

Atmel Asia Limited

Unit 01-5 & 16, 19F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon

HONG KONG

Tel: (+852) 2245-6100

Fax: (+852) 2722-1369

Atmel Munich GmbH

Business Campus
Parkring 4
D-85748 Garching b. Munich
GERMANY

Tel: (+49) 89-31970-0

Fax: (+49) 89-3194621

Atmel Japan G.K.

16F Shin-Osaki Kangyo Bldg.
1-6-4 Osaki, Shinagawa-ku
Tokyo 141-0032
JAPAN

Tel: (+81)(3) 6417-0300

Fax: (+81)(3) 6417-0370

© 2013 Atmel Corporation. All rights reserved. / Rev.: 42222B-07/2014

Atmel®, Atmel logo and combinations thereof, Enabling Unlimited Possibilities®, and others are registered trademarks or trademarks of Atmel Corporation or its subsidiaries. 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.