

# Application Note

## Static LCD Driver with I2C Interface

### AN-CM-256

#### **Abstract**

*This application note describes how to create a low power static LCD driver using a GreenPAK IC.*

*This application note comes complete with design files which can be found in the References section.*

---

---

---



---

**Static LCD Driver with I2C Interface**

## Contents

<b>Abstract</b> .....	<b>1</b>
<b>Contents</b> .....	<b>2</b>
<b>Figures</b> .....	<b>2</b>
<b>Tables</b> .....	<b>2</b>
<b>1 Terms and Definitions</b> .....	<b>4</b>
<b>2 References</b> .....	<b>4</b>
<b>3 Introduction</b> .....	<b>5</b>
<b>4 Basics of Liquid Crystal Displays</b> .....	<b>5</b>
<b>5 GreenPAK Design Basic Block Diagram</b> .....	<b>7</b>
<b>6 Design Current Consumption</b> .....	<b>8</b>
<b>7 GreenPAK Device Schematic</b> .....	<b>8</b>
7.1 I2C Interface.....	9
7.2 Output Segment Driver .....	10
7.3 4.3 Internal Oscillator and Backplane Clock Source Control .....	10
7.4 Backplane Clock Output or Segment 15 Output Pin Control .....	11
<b>8 LCD System Prototype</b> .....	<b>11</b>
<b>9 I2C Commands for LCD Control</b> .....	<b>13</b>
9.1 I2C Commands for LCD Test.....	16
<b>10 Test Results</b> .....	<b>16</b>
<b>11 Conclusion and Results Discussion</b> .....	<b>18</b>
<b>Appendix A</b> .....	<b>19</b>
A.1 Source Code .....	19
<b>Revision History</b> .....	<b>29</b>

## Figures

Figure 1: Principle Operation of an LCD (source [5]). .....	6
Figure 2: Control Signals for Static LCD Drive .....	6
Figure 3: Basic Block Diagram of GreenPAK Design .....	7
Figure 4: Top View of the GreenPAK Design Schematic.....	9
Figure 5: Close View to I2C Interface Block.....	9
Figure 6: ASM Output Used as Internal Segment Control .....	10
Figure 7: Backplane Clock Source Selection and Oscillator Control .....	10
Figure 8: PIN 20 Operation Control.....	11
Figure 9: Schematic of the System Prototype .....	12
Figure 10: System Prototype Picture. IC1 is on the Left Side and IC2 is on the Right Side .....	13

## Tables

Table 1: Pinout Description for LCD Driver Design.....	8
Table 2: Segments and Device Drivers .....	13

---

---

Static LCD Driver with I2C Interface

Table 3: I2C Command Sequence Description for IC1 ..... 14  
Table 4: Lookup Table to Write Numbers and Letters on LCD Digit 1 and 2..... 15  
Table 5: I2C Command Sequence Description for IC2 ..... 15  
Table 6: Lookup Table to Write Numbers and Letters on LCD Digit 3 and 4..... 15  
Table 7: LCD Prototype Display Sequence ..... 16  
Table 8: Pictures of LCD Display after MCU Commands to GreenPAK Devices ..... 17  
Table 9: Current Measurements for Each GreenPAK Device..... 17

---

---

## Static LCD Driver with I2C Interface

### 1 Terms and Definitions

AC	Alternate current
ASIC	Application specific integrated circuit
ASM	Asynchronous state machine
CPLD	Complex programmable logic device
DC	Direct current
ICs	Integrated Circuits
LCD	Liquid crystal displays

### 2 References

For related documents and software, please visit:

<https://www.dialog-semiconductor.com/products/greenpak>.

Download our free **GreenPAK™** Designer software [1] to open the .gp files [2] and view the proposed circuit design. Use the **GreenPAK** development tools [3] to freeze the design into your own customized IC in a matter of minutes. Renesas Electronics provides a complete library of application notes [4] featuring design examples as well as explanations of features and blocks within the IC.

- [1] [GreenPAK Designer Software](#), Software Download and User Guide, Renesas Electronics
- [2] [AN-CM-256 Static LCD Driver with I2C Interface.gp](#), [GreenPAK Design File](#), Renesas Electronics
- [3] [GreenPAK Development Tools](#), [GreenPAK Development Tools Webpage](#), Renesas Electronics
- [4] [GreenPAK Application Notes](#), [GreenPAK Application Notes Webpage](#), Renesas Electronics
- [5] Application Note AN-001 – Basics of LCD Technology, Hitachi
- [6] Application Note AN-005 – Display Modes, Hitachi
- [7] Application Note AN-1090 Simple I2C IO Controllers with SLG46531V, Renesas Electronics

---

---

## Static LCD Driver with I2C Interface

### 3 Introduction

Liquid Crystal Displays (LCD) are widely used for commercial and industrial applications because of their good visual properties, low cost and, low power consumption. These properties make the LCD the standard solution for battery-operated devices, like portable instruments, calculators, watches, radios, etc.

However, to properly control what the LCD shows, the LCD's electronic driver must generate appropriate voltage waveforms to LCD pins. The waveforms should be AC (alternate current) in nature because DC (direct current) voltages will permanently damage the device. The appropriate driver would source these signals to LCD at a minimum of power consumption.

Two types of LCDs exist, the Static, with only one backplane and one pin for individual segment control and, the Multiplexed, with multiple backplanes and multiple segments connected for each pin.

This application note will present the design of one static LCD driver with SLG46537V [GreenPAK](#) device. The designed LCD driver would drive up to 15 LCD's segments, using a few microamperes of current from the power supply and offer an I2C interface for control.

In the following sections will be shown:

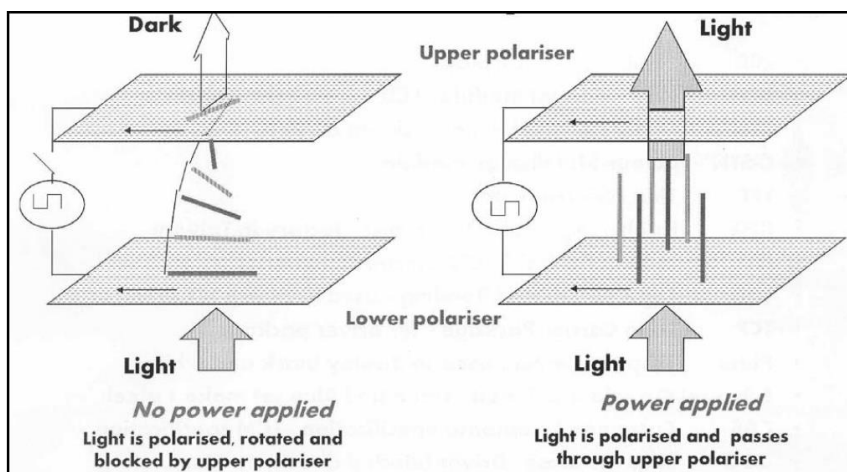
- basic knowledge information about LCDs;
- the SLG46537V [GreenPAK](#) LCD driver design in detail;
- how to drive a seven segment, 4-digit static LCD with two [GreenPAK](#) devices;

### 4 Basics of Liquid Crystal Displays

Liquid Crystal Displays (LCD) is a technology that does not emit light, it only controls how an external light source passes through. This external light source could be the available ambient light, in the reflective display type, or the light from a backlight led or lamp, in transmissive display type.

LCDs are constructed with two plates of glass (upper and bottom), a thin layer of liquid crystal (LC) between them and two light polarizers [5][6]. The polarizer is a light filter for the light electromagnetic field. Only the light components in the right electromagnetic field direction pass through the polarizer, while the other components are blocked. The liquid crystal is an organic material that rotates the electromagnetic field of the light 90 degrees or more. However, when an electrical field is applied to the LC it does not rotate the light anymore. With the addition of transparent electrodes in the upper and bottom display glass, its possible to control when the light passes through, and when not, with an external source of the electrical field. [Figure 1](#) below illustrates this operation control. In [Figure 1](#), the display is dark when there isn't an electrical field. This is because both polarizers filter the light in the same direction. If the polarizers are orthogonal, then the display will be dark when the electrical field is present. This is the most common situation for reflective displays.

## Static LCD Driver with I2C Interface



**Figure 1: Principle Operation of an LCD (source [5]).**

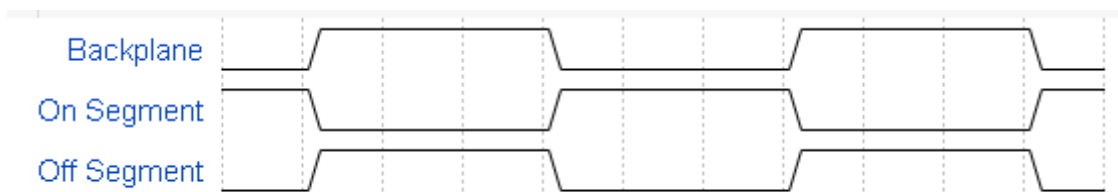
The minimum electrical field, or voltage, to control the LCD is called the ON threshold. The LC is only affected by the voltage, and there is hardly any current in LC material. The electrodes in LCD forms a small capacitance and this is the only load for a driver. This is the reason for an LCD being a low power device to show visual information.

However, its important to note that the LCD can't operate with a direct current (DC) voltage source for too long. The application of a DC voltage will cause chemical reactions in LC material, permanently damaging it [5]. The solution is to apply an alternate voltage (AC) in LCDs electrodes.

In static LCDs, a backplane electrode is built in one glass and individual LCD's segments, or pixels, are put in the other glass. This is one of the simplest LCD types and the one with the best contrast ratio. However, this type of display usually requires too many pins to control each individual segment.

In general, a driver controller sources a square wave clock signal for the backplane and a clock signal for the segments in the front plane together. When the backplane clock is in-phase with the segment clock, the root-mean-square (RMS) voltage between both planes is zero, and the segment is transparent. Otherwise, if the RMS voltage is higher than LCD ON threshold, the segment becomes dark. The waveforms for the backplane, on and off segment are shown in Figure 2. As can be seen in the figure, the ON segment is out-of-phase in relation to the backplane signal. The off segment is in-phase in relation to the backplane signal. The applied voltage could be between 3 and 5 volts for low cost, low power displays.

The clock signal for LCD's backplane and segments usually are in the range of 30 to 100 Hz, the minimum frequency to avoid a visual flicker effect on LCD. Higher frequencies are avoided to reduce the power consumption of the overall system. The system composed of LCD and drivers would consume little current, in the order of microamperes. This makes them perfectly suitable for low power and battery power supply source applications.



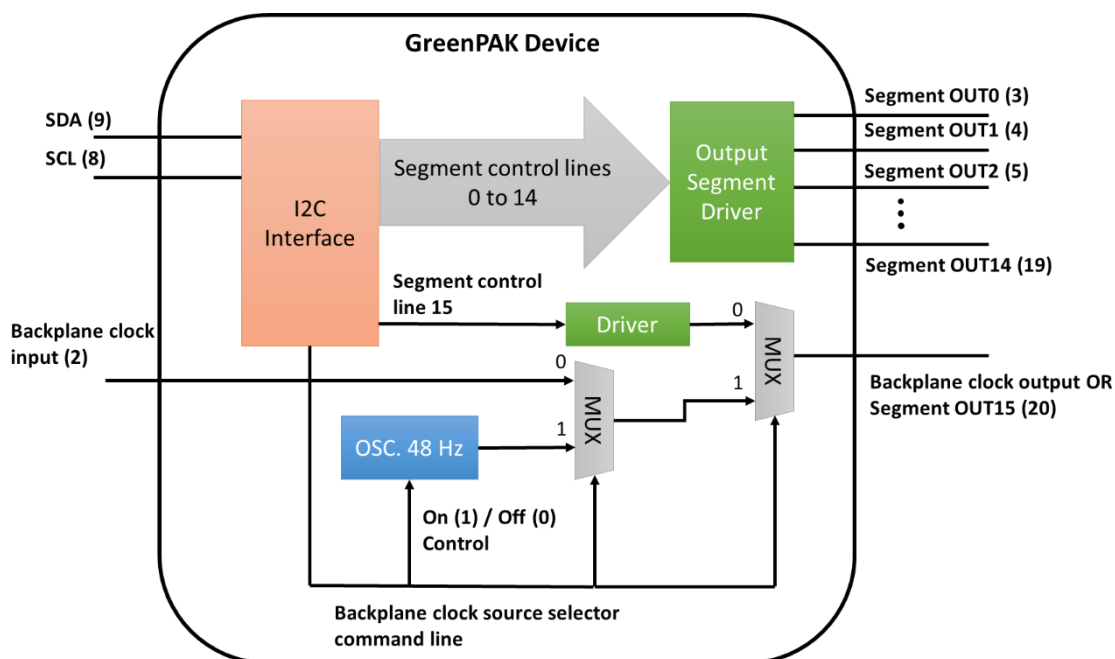
**Figure 2: Control Signals for Static LCD Drive**

## Static LCD Driver with I2C Interface

In the following sections, the design of an LCD static driver with **GreenPAK** device that can generate the backplane clock signal and the individual segment clock signal for a commercial LCD is presented in detail.

### 5 GreenPAK Design Basic Block Diagram

A block diagram that illustrates the **GreenPAK** design is shown in **Figure 3**. The basic blocks of the design are the I2C interface, the output segment driver, the internal oscillator, and the backplane clock source selector.



**Figure 3: Basic Block Diagram of GreenPAK Design**

The I2C interface block controls each individual segment output and the backplane clock source of the LCD. The I2C interface block is the only system input for segment output control.

When the internal segment control line is set (high level) the respective LCD segment is dark opaque. When the internal segment control line is reset (low level) the respective LCD segment is transparent.

Each internal segment control line is connected to an output driver. The output segment driver block will generate an in-phase clock signal with relationship to the backplane clock for transparent segments. For dark segments, this signal is out of phase with relationship to backplane clock.

The backplane clock source is selected with I2C interface too. When the internal backplane clock source is selected, the internal oscillator is turned on. The internal oscillator will generate a clock frequency of 48Hz. This signal will be used by output segment driver block and is addressed to the backplane clock output pin (**GreenPAK** pin 20).

When external backplane clock source is selected, the internal oscillator is turned off. The output segment driver reference is the external backplane clock input (**GreenPAK** pin 2). In this case, the backplane clock output pin could be used as an additional segment control line, the segment OUT15.

More than one **GreenPAK** device could be used on the same I2C line. To do it, each device must be programmed with a different I2C address. In this way is possible to extend the number of LCD segments driven. One device is configured to generate the backplane clock source, driving 14 segments, and the others are configured to use an external backplane clock source. Each additional

## Static LCD Driver with I2C Interface

device could drive more 15 segments in this way. It is possible to connect up to 16 devices on the same I2C line and then is possible to control up to 239 segments of an LCD.

In this application note, this idea is used to control 29 segments of an LCD with 2 GreenPAK devices. The device pinout functionality is summarized in [Table 1](#).

**Table 1: Pinout Description for LCD Driver Design**

PIN	Design Function
1	Power Supply
2	Backplane clock input
3	Segment output 1 (SEG_OUT_1)
4	Segment output 2 (SEG_OUT_2)
5	Segment output 3 (SEG_OUT_3)
6	Segment output 4 (SEG_OUT_4)
7	Segment output 5 (SEG_OUT_5)
8	SCL
9	SDA
10	Segment output 6 (SEG_OUT_6)
11	Ground
12	Segment output 7 (SEG_OUT_7)
13	Segment output 8 (SEG_OUT_8)
14	Segment output 9 (SEG_OUT_9)
15	Segment output 10 (SEG_OUT_10)
16	Segment output 11 (SEG_OUT_11)
17	Segment output 12 (SEG_OUT_12)
18	Segment output 13 (SEG_OUT_13)
19	Segment output 14 (SEG_OUT_14)
20	Backplane clock output OR Segment output 15 (SEG_OUT_15)

## 6 Design Current Consumption

An important concern in this design is the current consumption, that should be as low as possible. The [GreenPAK](#) device estimated quiescent current is 0.75  $\mu\text{A}$  for 3.3 V supply operation and 1.12  $\mu\text{A}$  for 5 V supply operation. The current consumption of the internal oscillator is 7.6  $\mu\text{A}$  and 8.68  $\mu\text{A}$  for 3.3 V and 5 V power supply operation respectively. It is not expected to have a significant increase in current consumption from switching losses, because this design operates at a low clock frequency. The estimated maximum current consumed for this design is lower than 15  $\mu\text{A}$  when the internal oscillator is on, and 10  $\mu\text{A}$  when the internal oscillator is off. The measured current consumed in both situations is shown in Section 1610 - [Test Results](#).

## 7 GreenPAK Device Schematic

The project designed in [GreenPAK](#) software is shown in [Figure 4](#). This schematic will be described using the basic blocks diagrams as the reference.



Static LCD Driver with I2C Interface

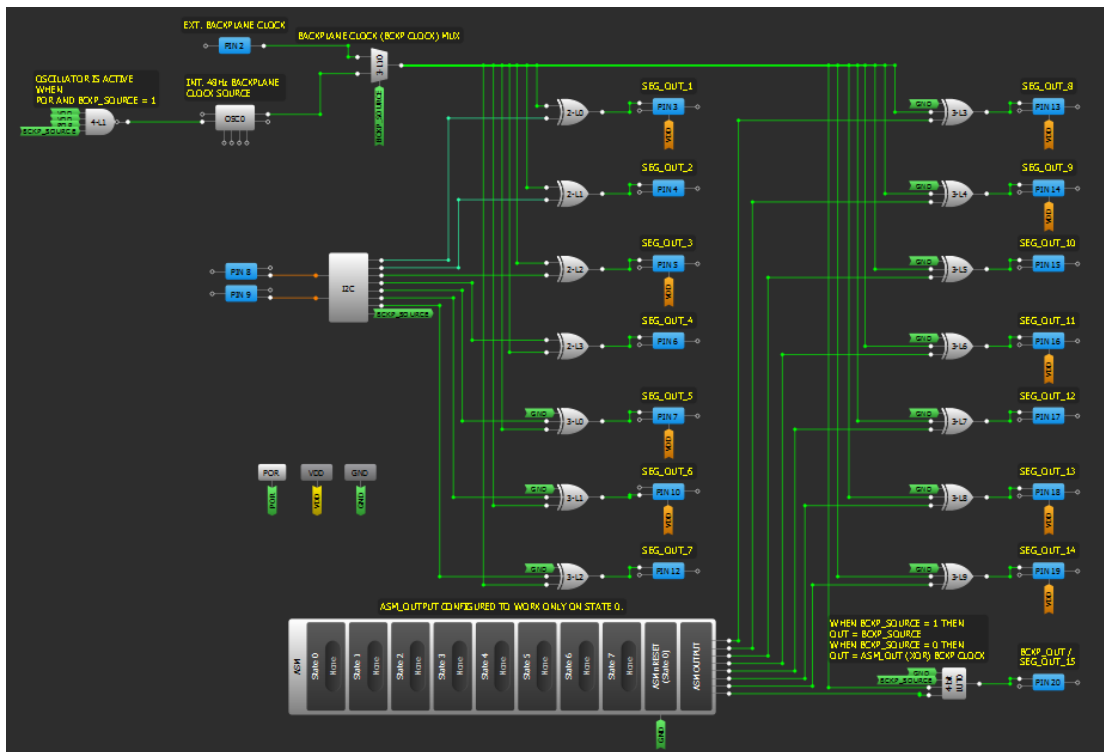


Figure 4: Top View of the GreenPAK Design Schematic

7.1 I2C Interface

I2C interface block is used as the main control block of the device operation control. A close view to the block connections and configured properties are shown in Figure 5.

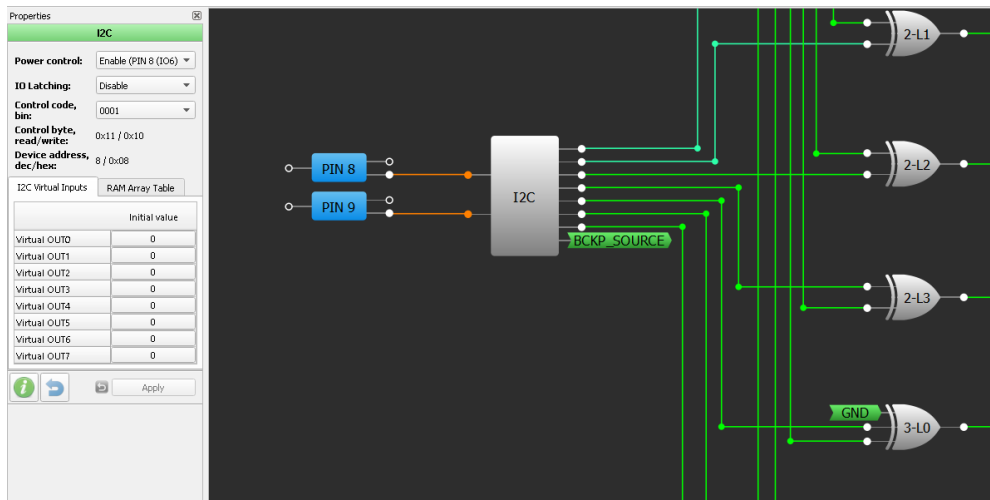


Figure 5: Close View to I2C Interface Block

This block is connected to PIN 8 and PIN 9, that are I2C SCL and SDA pins respectively. Inside the device, the I2C block offers 8 Virtual Inputs. The initial value for each Virtual Input is shown in the properties window (see Figure 5). Virtual inputs from OUT0 until OUT6 are used as segment control lines. These control lines correspond to segment output 1 to segment output 7 and are connected to the segment output driver. Virtual Input OUT7 is used as backplane clock source selector line

## Static LCD Driver with I2C Interface

control, with net name BCKP\_SOURCE. This net will be used by other blocks in the design. The I2C control code is configured with a different value for each IC in the project.

8 more internal segment control lines are available in the Asynchronous State Machine (ASM) output, as shown in Figure 6 below. Segment output line 8 (SEG\_OUT\_8 in properties window) through segment output line 15 (SEG\_OUT\_15) are controlled by ASM output on state 0. There isn't any state transition in ASM block, it is always in state 0. The outputs of ASM are connected to segment output drivers.

The segment output drivers will generate the output signal of the device.

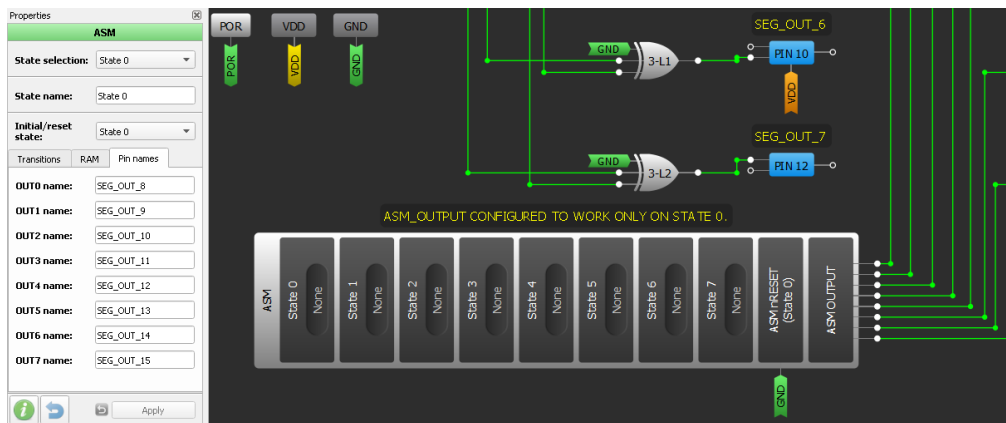


Figure 6: ASM Output Used as Internal Segment Control

### 7.2 Output Segment Driver

The output segment driver is essentially a Lookup table (LUT) configured as an XOR logic port. For each output segment, it must be an XOR port connected to segment control line and to backplane clock (BCKP\_CLOCK). The XOR port is responsible to generate the in-phase and out-of-phase signal to output segment. When the segment control line is at a high level, the XOR port output will invert the backplane clock signal and generate an out-of-phase signal to segment pin. The voltage difference between LCD backplane and LCD segment, in this case, will set the LCD segment as a dark segment. When the segment control line is at a low level, the XOR port output will follow the backplane clock signal and then generate an in-phase signal to segment pin. Because no voltage is applied between the LCD backplane and segment in this case, the segment is transparent to light.

### 7.3 4.3 Internal Oscillator and Backplane Clock Source Control

The internal oscillator is used when the signal BCKP\_CLOCK from I2C interface is set to a high level. A close view of the clock source control diagram is shown in Figure 7 below.

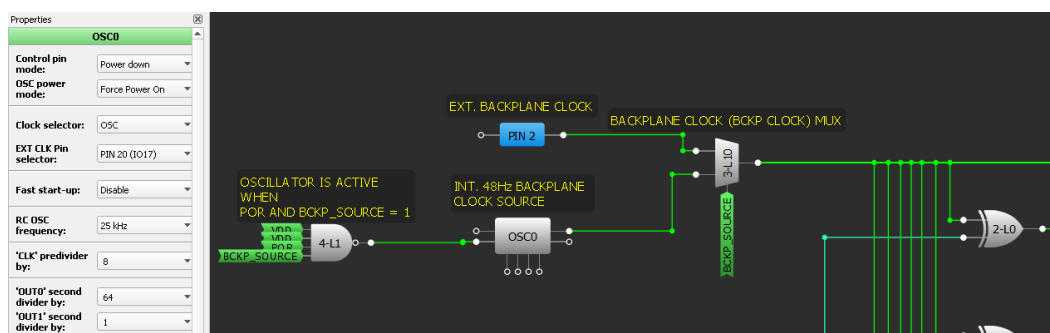


Figure 7: Backplane Clock Source Selection and Oscillator Control

## Static LCD Driver with I2C Interface

The oscillator is configured as 25 kHz RC frequency, with the highest output divisor available at oscillator OUT0 (8/64). The entire configuration is seen in the properties window shown in [Figure 7](#). In this way, the internal oscillator will generate a clock frequency of 48 Hz.

The oscillator is active only when BCKP\_SOURCE signal is at a high level together with POR signal. This control is done by connecting these two signals to the NAND port of the 4-L1 LUT. The output of the NAND is then connected to the input of the oscillator power down control pin.

Signal BCKP\_SOURCE controls the MUX built with 3-L10 LUT. When BCKP\_SOURCE signal is at a low level, the backplane clock source comes from PIN2. When this signal is at a high level the backplane clock source comes from the internal oscillator.

### 7.4 Backplane Clock Output or Segment 15 Output Pin Control

Pin 20 in this design has a double function, which depends on the selected backplane clock source. The operation of this pin is controlled with one 4 input LUT, as shown in [Figure 8](#). With a 4-bit LUT, it is possible to associate the operation of XOR port with an output MUX. When the BCKP\_SOURCE signal is at a high level, the LUT output will follow the internal oscillator clock. Then pin 20 operates as a backplane clock output. When BCKP\_SOURCE signal is at a low level, the LUT output will be the XOR operation between SEG\_OUT\_15, from ASM output, and backplane clock signal. The 4-bit LUT configuration to do this operation is shown in [Figure 8](#).

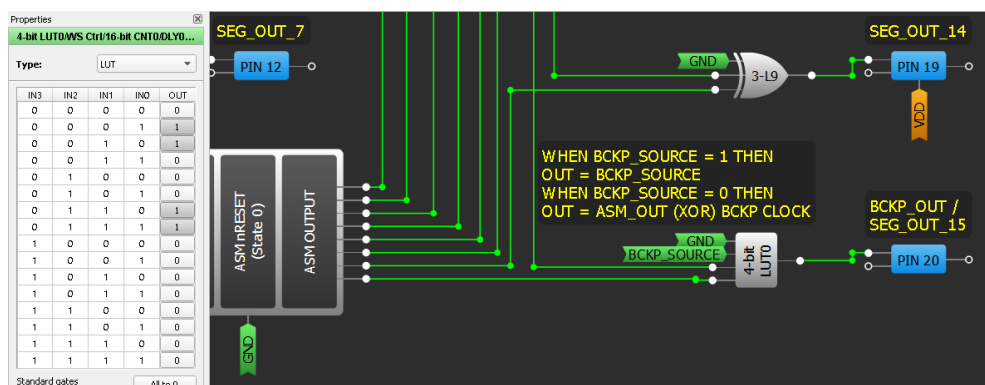


Figure 8: PIN 20 Operation Control

## 8 LCD System Prototype

To demonstrate the use of the [GreenPAK](#) design solution, an LCD system prototype was assembled on a breadboard. For the prototype, a seven segment, 4-digit static LCD is driven by two [GreenPAK](#) devices on DIP board. One device (IC1) uses the internal oscillator to drive LCD backplane, and the other device (IC2) uses this signal as backplane input reference. Both ICs are controlled over I2C interface by an STM32F103C8T6 microcontroller (MCU) in a minimum development board.

[Figure 9](#) shows the schematic of the connections between the two [GreenPAK](#) ICs, the LCD display, and the MCU board. In the schematic, the [GreenPAK](#) device with U1 (IC1) reference drives LCD digit one and two (LCD left side). The [GreenPAK](#) device with U2 (IC2) reference drives LCD digit three and four, plus the COL segment (LCD right side). The power supply for both devices comes from the regulator in the microcontroller development board. Two removable jumpers between the power supply and VDD pins of each [GreenPAK](#) device are added for current measurement with a multimeter.

A picture of the assembled prototype is shown in [Figure 10](#).

Static LCD Driver with I2C Interface

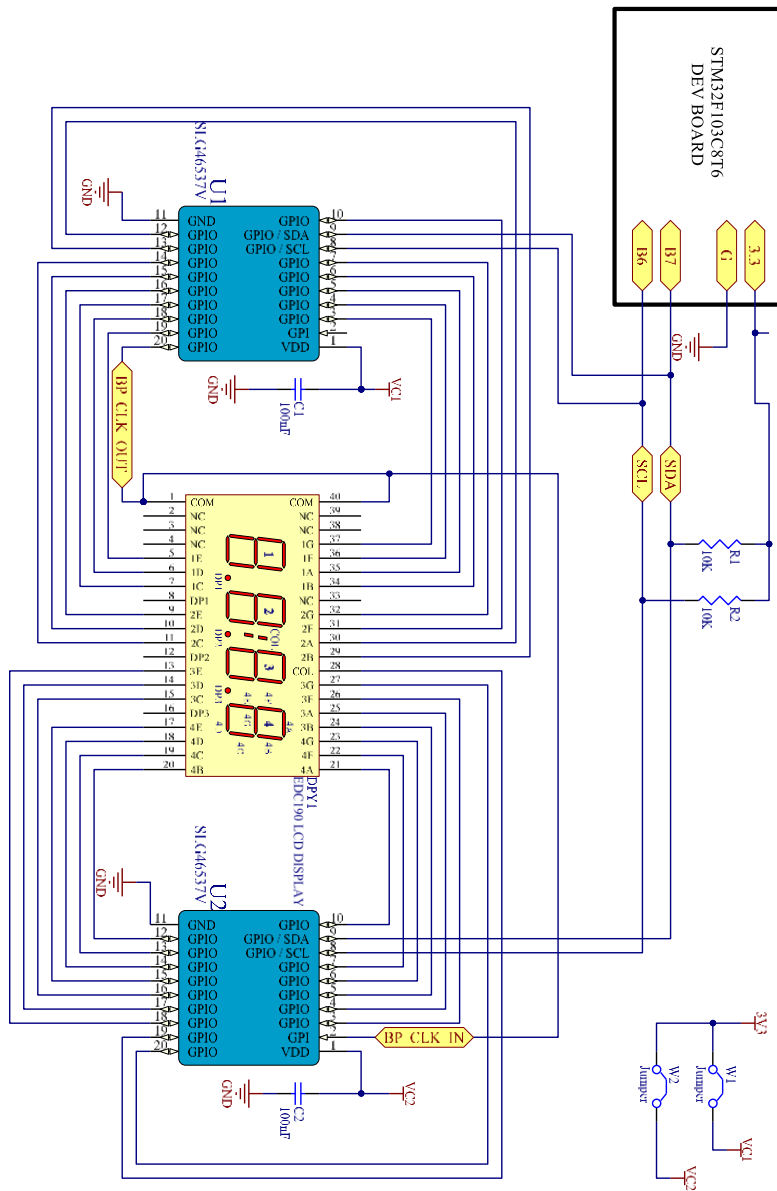


Figure 9: Schematic of the System Prototype

## Static LCD Driver with I2C Interface

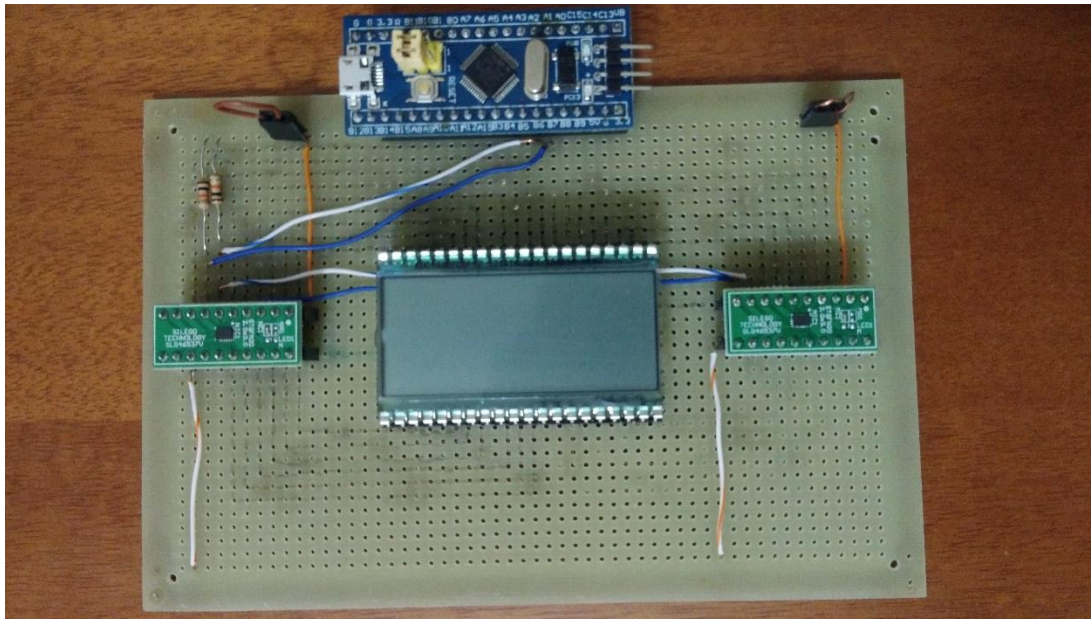


Figure 10: System Prototype Picture. IC1 is on the Left Side and IC2 is on the Right Side

## 9 I2C Commands for LCD Control

The two GreenPAK devices on the breadboard are programmed with the same design, except by the Control Byte value. The control byte of IC1 is 0 (I2C address 0x00), while the I2C control byte is 1 (I2C address 0x10). The connections between display segments and device drivers are summarized in the table below.

Table 2: Segments and Device Drivers

PIN	IO Number	Design Function	(U1) Display Connection	(U2) Display Connection
1	VDD			
2	0	Backplane clock input		COM
3	1	Segment output 1	1G	3F
4	2	Segment output 2	1F	3A
5	3	Segment output 3	1A	3B
6	4	Segment output 4	1B	4G
7	5	Segment output 5	2G	4F
8	6	SCL		
9	7	SDA		
10	8	Segment output 6	2F	4A
11	GND			
12	9	Segment output 7	2A	4B
13	10	Segment output 8	2B	4C
14	11	Segment output 9	2C	4D

## Static LCD Driver with I2C Interface

PIN	IO Number	Design Function	(U1) Display Connection	(U2) Display Connection
15	12	Segment output 10	2D	4E
16	13	Segment output 11	2E	3C
17	14	Segment output 12	1C	3D
18	15	Segment output 13	1D	3E
19	16	Segment output 14	1E	COL
20	17	Backplane clock output OR Segment output 15	COM	3G

The connections were selected in this way to create a clearer schematic and to simplify assembling the breadboard connections.

The control of the segment's output is done by I2C write commands to I2C Virtual Inputs and ASM output registers. As described in the application note AN-1090 Simple I2C IO Controllers with SLG46531V [7], the I2C write command is structured as follows:

- Start;
- Control byte (R/W bit is 0);
- Word address;
- Data;
- Stop.

All the I2C write commands are made to Word Address 0xF4 (I2C Virtual Inputs) and 0xD0 (ASM Output for state 0). The commands to write in IC1 and control LCD digit 1 and 2 are summarized in Table 3. In the command sequence representation, the open bracket “[” denotes the Start signal, and the close bracket “]” denotes the Stop signal.

**Table 3: I2C Command Sequence Description for IC1**

I2C Write command	Description
[0x00, 0xF4, Byte0]	Assign a state value to segments 1 to 7. It configures the backplane oscillator (Most Significant Bit, MSB) too. In IC1 the MSB is always set. In this way, the IC1 is configured to source the backplane clock signal. Byte0 value is described below.
[0x00, 0xD0, Byte1]	Assign a state value to segments 8 to 15. Byte1 value is described below.

The two bytes above control segments of LCD digit 1 and digit 2 together. Here, the approach is to use an individual lookup table (LUT) in software for each digit, considering the segments in both bytes. The byte values from lookup table should be mixed using a bitwise OR operation, and then send to the IC. The Table 4 shown the Byte0 and Byte1 value for each numeric value that should be written in each display digit.

## Static LCD Driver with I2C Interface

**Table 4: Lookup Table to Write Numbers and Letters on LCD Digit 1 and 2**

Digit 1		Digit 2		Number/letter
Byte0	Byte1	Byte0	Byte1	
0x8E	0x70	0xE0	0x0F	0
0x88	0x10	0x80	0x03	1
0x8D	0x60	0xD0	0x0D	2
0x8D	0x30	0xD0	0x07	3
0x8B	0x10	0xB0	0x03	4
0x87	0x30	0xF0	0x06	5
0x87	0x70	0xF0	0x0E	6
0x8C	0x10	0xC0	0x03	7
0x8F	0x70	0xF0	0x0F	8
0x8F	0x10	0xF0	0x03	9
0x8F	0x50	0xF0	0x0B	A
0x83	0x70	0xB0	0x0E	B
0x86	0x60	0xE0	0x0C	C
0x89	0x70	0x90	0x0F	D
0x87	0x60	0xF0	0x0C	E
0x87	0x40	0xF0	0x08	F

For example, to write in the Digit 1 the number 3, and in the Digit 2 the number 4, Byte0 is 0xBD (0x8D bitwise OR with 0xB0) and Byte 1 is 0x33 (0x30 bitwise OR with 0x03).

The command to write in IC2 and controls Digit 3 and 4, are described in [Table 5](#).

**Table 5: I2C Command Sequence Description for IC2**

I2C Write command	Description
[0x10, 0xF4, Byte0]	Assign a state value to segments 1 to 7. It configures the backplane oscillator (Most Significant Bit, MSB). In IC2 the MSB is always reset. In this way, the IC2 is configured to use an external source of backplane clock. This configuration enables an additional segment output to this IC (Segment output 15). Byte0 value is described below.
[0x10, 0xD0, Byte1]	Assign a state value to segments 8 to 15. Byte1 value is described below.

The control logic of digits 3 and 4 are like the control of digits 1 and 2. [Table 6](#) shows the LUT for these two digits.

**Table 6: Lookup Table to Write Numbers and Letters on LCD Digit 3 and 4**

Digit 3		Digit 4		Number/letter
Byte0	Byte1	Byte0	Byte1	
0x07	0x38	0x70	0x07	0
0x04	0x8	0x40	0x01	1
0x06	0xB0	0x68	0x06	2
0x06	0x98	0x68	0x03	3

## Static LCD Driver with I2C Interface

Digit 3		Digit 4		Number/letter
Byte0	Byte1	Byte0	Byte1	
0x05	0x88	0x58	0x01	4
0x03	0x98	0x38	0x03	5
0x03	0xB8	0x38	0x07	6
0x06	0x8	0x60	0x01	7
0x07	0xB8	0x78	0x07	8
0x07	0x88	0x78	0x01	9
0x07	0xA8	0x78	0x05	A
0x01	0xB8	0x18	0x07	B
0x03	0x30	0x30	0x06	C
0x04	0xB8	0x48	0x07	D
0x03	0xB0	0x38	0x06	E
0x03	0xA0	0x38	0x04	F

The difference in IC2 is the COL segment. This segment is controlled by Byte1. To set up this segment dark, a bitwise OR operation between the Byte1 and the value 0x40 should be done.

### 9.1 I2C Commands for LCD Test

For LCD test a firmware was developed in C language for the MCU board. This firmware will send a sequence of commands to both ICs on the breadboard. The source code for this firmware is in the Appendix section. The entire solution was developed using Atollic TrueStudio for STM32 9.0.1 IDE.

The sequence of commands and the respective values shown in the display are summarized in [Table 7](#) below.

**Table 7: LCD Prototype Display Sequence**

Display text (all digits and COL :)	I2C Write command sequence (4 commands).
88:88	[0x00, 0xF4, 0xFF], [0x00, 0xD0, 0x7F], [0x10, 0xF4, 0xFF], [0x10, 0xD0, 0xFF]
0000	[0x00, 0xF4, 0xEE], [0x00, 0xD0, 0x7F], [0x10, 0xF4, 0x77], [0x10, 0xD0, 0x3F]
1234	[0x00, 0xF4, 0xD8], [0x00, 0xD0, 0x1D], [0x10, 0xF4, 0x5E], [0x10, 0xD0, 0x99]
8765	[0x00, 0xF4, 0xCF], [0x00, 0xD0, 0x73], [0x10, 0xF4, 0x3B], [0x10, 0xD0, 0xBB]
EB9D	[0x00, 0xF4, 0xB7], [0x00, 0xD0, 0x6E], [0x10, 0xF4, 0x4F], [0x10, 0xD0, 0x8F]
12:00	[0x00, 0xF4, 0xD8], [0x00, 0xD0, 0x1D], [0x10, 0xF4, 0x77], [0x10, 0xD0, 0x7F]
1200	[0x00, 0xF4, 0xD8], [0x00, 0xD0, 0x1D], [0x10, 0xF4, 0x77], [0x10, 0xD0, 0x3F]

## 10 Test Results





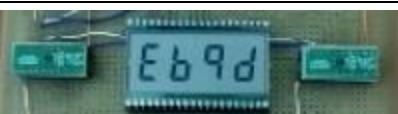


The prototype test consists of verifying the display values after an MCU command and measuring the current sink by each IC during operation.



Static LCD Driver with I2C Interface

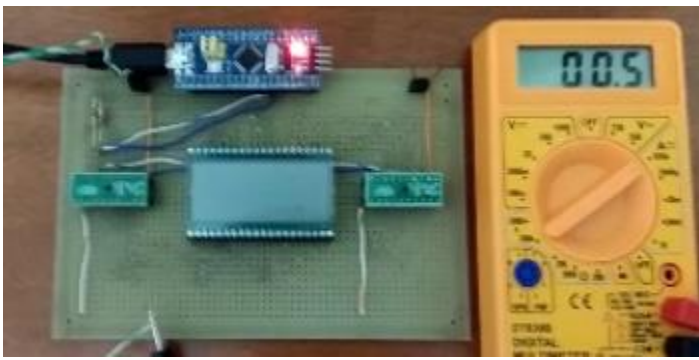
Pictures of the LCD for each command value are shown in [Table 8](#) below.

**Table 8: Pictures of LCD Display after MCU Commands to GreenPAK Devices**

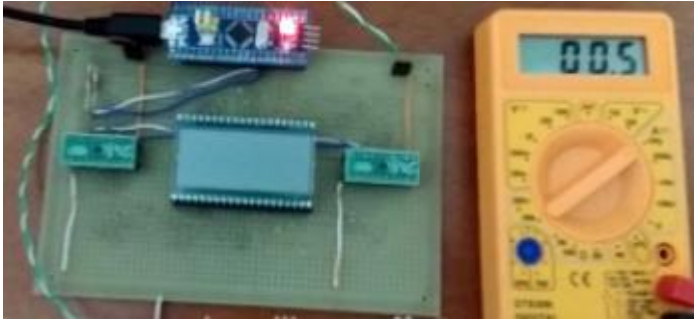

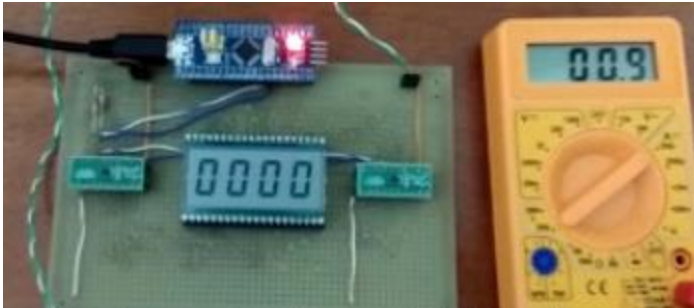
Expected Display text.	Display picture
88:88	
0000	
1234	
8765	
EB9D	
12:00	
1200	

The current sink for each device was measured with a multimeter, in its lowest current range of 200  $\mu$ A. Pictures of the measured current for each device, during start-up and normal operation, are shown in [Table 9](#) below.

**Table 9: Current Measurements for Each GreenPAK Device**

Picture	Measured IC (power supply always at 3,3V)
	IC1 current during start-up (internal oscillator is off and there isn't any external backplane clock source)

## Static LCD Driver with I2C Interface

Picture	Measured IC (power supply always at 3,3V)
	<p>IC1 current in normal operation (internal oscillator is on and sources the backplane signal for the LCD display).</p>
	<p>IC1 current in normal operation (internal oscillator is on and sources the backplane signal for the LCD display).</p>
	<p>IC2 current in normal operation (internal oscillator is off, backplane clock signal comes from IC1).</p>

## 11 Conclusion and Results Discussion

The design of a low power static LCD driver with **GreenPAK** device was presented. This design clearly shows one of the greatest features of the **GreenPAK** devices: their low quiescent current. Because **GreenPAK** devices are a hardware-based solution, it is possible to work at a low-frequency operation, in this case, 48 Hz. An MCU based solution will require a higher operation frequency, even for periodically short periods of time, and then will draw more power. And, comparing **GreenPAK** device with a CPLD (Complex Programmable Logic Device), it is clear to see that usually a CPLD has a quiescent current higher than 20  $\mu$ A.

It is interesting to note that this design could be easily modified for a better fit in the requirements of a specific project. A good example is the segment controls pinout. They could easily be changed to simplify the printed circuit board and the software development at the same time. This is an interesting feature when the device is compared with an off-the-shelf ASIC (Application Specific Integrated Circuit). Usually, ASICs are designed to fit on a broad range of applications, and an initial software routine should be written to properly configure the IC before the operation. A configurable

---



---

## Static LCD Driver with I2C Interface

device could be designed to start ready to use after power up. In this way, it's possible to cut the software development time for IC initial configuration.

## Appendix A

### A.1 Source Code

```

/*
*****
File:   stm32flxx_it.c
Info:   Main Interrupt Service Routines.
*****
*/

uint32_t TickCount = 0;

/**
**=====
**
** Abstract: This function handles SysTick Handler.**
**=====
*/
void SysTick_Handler(void)

```

## Static LCD Driver with I2C Interface

```

{
    TickCount++;
}

/*
*****
File:      main.c
Info:      Demo code for static LCD driver with SLG46537V
*****
*/

/* Includes */
#include <stddef.h>
#include "stm32f10x.h"
#include "stm32f10x_conf.h"
#include "stm32f1xx_it.h"

// Basic definitions
#define ON_BOARD_LED_PIN    GPIO_Pin_13
#define ON_BOARD_LED_PORT  GPIOC
#define ON_BOARD_I2C_PORT  GPIOB
#define SDA_I2C_PIN        GPIO_Pin_7
#define SCL_I2C_PIN        GPIO_Pin_6
#define LED_TURN_ON()      GPIO_ResetBits(ON_BOARD_LED_PORT, ON_BOARD_LED_PIN)
#define LED_TURN_OFF()     GPIO_SetBits(ON_BOARD_LED_PORT, ON_BOARD_LED_PIN)
#define LCD_CONTROL_I2C    I2C1
#define I2C_TIMEOUT_VALUE  0xFF000000
#define TRUE                1
#define FALSE               0

#define BCKP_SOURCE_CTRL_BIT_MASK 0x80 // backplane source control bit mask
#define CONTROL_BYTE_IC1          0x00 // I2C address of device IC 1
#define CONTROL_BYTE_IC2          0x10 // I2C address of device IC 2
#define BYTE_0_ADDRESS             0xF4 // byte 0 config word
address
#define BYTE_1_ADDRESS             0xD0 // byte 1 config word
address

/* Test sequence command list to write on display */
const uint8_t TestDisplayCMDList[][4] =
{
    {0xEE, 0x7F, 0x77, 0x3F}, // write "0000" on display */
    {0xD8, 0x1D, 0x5E, 0x99}, // write "1234" on display */
    {0xCF, 0x73, 0x3B, 0xBB}, // write "8765" on display */
    {0xB7, 0x6E, 0x4F, 0x8F}, // write "EB9D" on display */
    {0xD8, 0x1D, 0x77, 0x7F}, // write "12:00" on display */
    {0xD8, 0x1D, 0x77, 0x3F} // write "1200" on display */
};

/* LUT for LCD's Digit1 control */
const uint8_t Digit1ByteLUT[][2] =
{
    0x8E, 0x70, // 0
    0x88, 0x10, // 1
    0x8D, 0x60, // 2
    0x8D, 0x30, // 3
    0x8B, 0x10, // 4
    0x87, 0x30, // 5

```

---



---

**Static LCD Driver with I2C Interface**

```

        0x87, 0x70,      // 6
        0x8C, 0x10,      // 7
        0x8F, 0x70,      // 8
        0x8F, 0x10,      // 9
        0x8F, 0x50,      // A
        0x83, 0x70,      // B
        0x86, 0x60,      // C
        0x89, 0x70,      // D
        0x87, 0x60,      // E
        0x87, 0x40      // F
};
/* LUT for LCD's Digit2 control */
const uint8_t Digit2ByteLUT[][2] =
{
        0xE0, 0x0F,      // 0
        0x80, 0x03,      // 1
        0xD0, 0x0D,      // 2
        0xD0, 0x07,      // 3
        0xB0, 0x03,      // 4
        0xF0, 0x06,      // 5
        0xF0, 0x0E,      // 6
        0xC0, 0x03,      // 7
        0xF0, 0x0F,      // 8
        0xF0, 0x03,      // 9
        0xF0, 0x0B,      // A
        0xB0, 0x0E,      // B
        0xE0, 0x0C,      // C
        0x90, 0x0F,      // D
        0xF0, 0x0C,      // E
        0xF0, 0x08      // F
};
/* LUT for LCD's Digit3 control */
const uint8_t Digit3ByteLUT[][2] =
{
        0x07, 0x38,      // 0
        0x04, 0x08,      // 1
        0x06, 0xB0,      // 2
        0x06, 0x98,      // 3
        0x05, 0x88,      // 4
        0x03, 0x98,      // 5
        0x03, 0xB8,      // 6
        0x06, 0x08,      // 7
        0x07, 0xB8,      // 8
        0x07, 0x88,      // 9
        0x07, 0xA8,      // A
        0x01, 0xB8,      // B
        0x03, 0x30,      // C
        0x04, 0xB8,      // D
        0x03, 0xB0,      // E
        0x03, 0xA0      // F
};
/* LUT for LCD's Digit4 control */
const uint8_t Digit4ByteLUT[][2] =
{
        0x70, 0x07,      // 0
        0x40, 0x01,      // 1
        0x68, 0x06,      // 2

```

---



---

**Static LCD Driver with I2C Interface**

```

        0x68, 0x03,      // 3
        0x58, 0x01,      // 4
        0x38, 0x03,      // 5
        0x38, 0x07,      // 6
        0x60, 0x01,      // 7
        0x78, 0x07,      // 8
        0x78, 0x01,      // 9
        0x78, 0x05,      // A
        0x18, 0x07,      // B
        0x30, 0x06,      // C
        0x48, 0x07,      // D
        0x38, 0x06,      // E
        0x38, 0x04       // F
};
/**
**=====
**
** Abstract: DelayMs function wait for a specified delay time.
**
**=====
*/
void DelayMs(uint32_t delay_time)
{
    uint32_t ReferenceTick;
    uint32_t ExpectedTickCount;

    ReferenceTick = TickCount;
    ExpectedTickCount = ReferenceTick + delay_time;

    if(ExpectedTickCount < ReferenceTick)
    {
        while(TickCount > ExpectedTickCount)
        {

        }

        return;
    }

    while(TickCount < ExpectedTickCount)
    {

    }
}

/**
**=====
**
** Abstract: BoardConfigInit function initialize dev board
**
**=====
*/
void BoardConfigInit(void)
{
    GPIO_InitTypeDef GPIO_InitStruct;
    I2C_InitTypeDef I2C_InitStruct;

```

## Static LCD Driver with I2C Interface

```

// Clock PORTC and PORTB Enable
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC | RCC_APB2Periph_GPIOB, ENABLE);

// Clock I2C1 Enable
RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C1, ENABLE);

// Configure the GPIO pin for the LED (PC13)
GPIO_InitStruct.GPIO_Pin = ON_BOARD_LED_PIN;
// Configure Led pin
GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_InitStruct.GPIO_Mode = GPIO_Mode_Out_PP;
// Set Output Push-Pull
GPIO_Init(ON_BOARD_LED_PORT, &GPIO_InitStruct);

// Configure the GPIO pins for the I2C interface (PB6 and PB7)
GPIO_InitStruct.GPIO_Pin = SDA_I2C_PIN | SCL_I2C_PIN;
Configure SDA and SCL pins
GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF_OD;
// Set alternate function open drain
GPIO_Init(ON_BOARD_I2C_PORT, &GPIO_InitStruct);

// Configure I2C1 peripheral
I2C_InitStruct.I2C_ClockSpeed = 20000;
// 100 kHz I2C clock speed
I2C_InitStruct.I2C_Ack = I2C_Ack_Disable;
I2C_InitStruct.I2C_AcknowledgedAddress = 0x00;
I2C_InitStruct.I2C_DutyCycle = I2C_DutyCycle_2;
I2C_InitStruct.I2C_Mode = I2C_Mode_I2C;
// Set as I2C interface
I2C_InitStruct.I2C_OwnAddress1 = 0X00;

I2C_Init(LCD_CONTROL_I2C, &I2C_InitStruct);
// set up I2C peripheral
I2C_Cmd(LCD_CONTROL_I2C, ENABLE);
// enable I2C

// ** Configures the SysTick event to fire every 1ms **
SysTick_Config(SystemCoreClock / 1000);
}

/**
**=====
**
** Abstract: I2C_WriteByteCmd is a function to write commands in GreenPAK
** devices. The function receives the ControlByte, Word Adress and the
** data byte.
**=====
*/
uint32_t I2C_WriteByteCmd(uint8_t ControlByte, uint8_t ByteAddress, uint8_t ByteData)
{
    uint32_t TimeoutCount; // variable used to timeout
    routines

    TimeoutCount = I2C_TIMEOUT_VALUE; // set timeout value

```

## Static LCD Driver with I2C Interface

```

// wait until I2C bus is not busy anymore
while(I2C_GetFlagStatus(LCD_CONTROL_I2C, I2C_FLAG_BUSY) == SET)
{
    TimeoutCount--; // decrements timeout
counter
    if(TimeoutCount == 0) // Trigger timeout condition
when
    { // timeout
counter reaches 0, then
        return FALSE; // quit of the function
    }
}

TimeoutCount = I2C_TIMEOUT_VALUE; // set timeout value

// generate start condition
I2C_GenerateSTART(LCD_CONTROL_I2C, ENABLE);

// wait until EV5 is triggered -> start condition correctly
// released on the I2C bus
while(I2C_CheckEvent(LCD_CONTROL_I2C, I2C_EVENT_MASTER_MODE_SELECT) == ERROR)
{
    TimeoutCount--; // decrements timeout
counter
    if(TimeoutCount == 0) // Trigger timeout condition
when
    { // timeout
counter reaches 0, then
        return FALSE; // quit of the function
    }
}

// send the control byte (I2C address)
I2C_Send7bitAddress(LCD_CONTROL_I2C, ControlByte, I2C_Direction_Transmitter);

TimeoutCount = I2C_TIMEOUT_VALUE; // set timeout value

// wait until EV6 is triggered -> Slave device ACK
while(I2C_CheckEvent(LCD_CONTROL_I2C,
I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED) == ERROR)
{
    TimeoutCount--; // decrements timeout
counter
    if(TimeoutCount == 0) // Trigger timeout condition
when
    { // timeout
counter reaches 0, then
        return FALSE; // quit of the function
    }
}

TimeoutCount = I2C_TIMEOUT_VALUE; // set timeout value

// send the word/byte address to CMIC
I2C_SendData(LCD_CONTROL_I2C, ByteAddress);

// wait until EV8_2 is triggered -> Data shifted out on I2C bus

```



## Static LCD Driver with I2C Interface

```

        while(I2C_CheckEvent(LCD_CONTROL_I2C, I2C_EVENT_MASTER_BYTE_TRANSMITTED) ==
ERROR)
    {
        TimeoutCount--; // decrements timeout
counter
        if(TimeoutCount == 0) // Trigger timeout condition
when
        { // timeout
counter reaches 0, then
            return FALSE; // quit of the function
        }
    }

    TimeoutCount = I2C_TIMEOUT_VALUE; // set timeout value

    // send the byte of data to CMIC
    I2C_SendData(LCD_CONTROL_I2C, ByteData);

    // wait until EV8_2 is triggered -> Data shifted out on I2C bus
    while(I2C_CheckEvent(LCD_CONTROL_I2C, I2C_EVENT_MASTER_BYTE_TRANSMITTED) ==
ERROR)
    {
        TimeoutCount--; // decrements timeout
counter
        if(TimeoutCount == 0) // Trigger timeout condition
when
        { // timeout
counter reaches 0, then
            return FALSE; // quit of the function
        }
    }

    TimeoutCount = I2C_TIMEOUT_VALUE; // set timeout value

    // generate stop condition on bus
    I2C_GenerateSTOP(LCD_CONTROL_I2C, ENABLE);

    return TRUE; // send TRUE when command
finish
}

/**
**=====
**
** Abstract: FlashLed -> generate a timed flash on board led
**
**=====
**/
void FlashLed(uint32_t Ton, uint32_t Toff)
{
    LED_TURN_ON();
    DelayMs(Ton);
    LED_TURN_OFF();
    DelayMs(Toff);
}
/**

```

---



---

**Static LCD Driver with I2C Interface**

```

**=====
**
** Abstract: main program
**
**=====
*/
int main(void)
{
    uint8_t Temp;
    uint16_t DisplayValue = 1200;
    uint16_t Result;
    uint8_t Digit1;
    uint8_t Digit2;
    uint8_t Digit3;
    uint8_t Digit4;
    uint8_t TempByte0;
    uint8_t TempByte1;
    BoardConfigInit(); // init board peripherals

    /* flash led signal to advice demo begin */

    FlashLed(3000, 1000); // Hold board led on for 3
seconds

    for(Temp = 0; Temp < 5; Temp++) // short flash led 5 times
    {
        FlashLed(500, 500);
    }

    /*
    * Set all segments in as opaque
    */
    I2C_WriteByteCmd(CONTROL_BYTE_IC1, BYTE_0_ADDRESS, 0xFF);
    I2C_WriteByteCmd(CONTROL_BYTE_IC1, BYTE_1_ADDRESS, 0xFF);
    I2C_WriteByteCmd(CONTROL_BYTE_IC2, BYTE_0_ADDRESS, 0x7F);
    I2C_WriteByteCmd(CONTROL_BYTE_IC2, BYTE_1_ADDRESS, 0xFF);

    /*
    * 3 led flash
    */
    for(Temp = 0; Temp < 3; Temp++)
    {
        FlashLed(250, 500);
    }

    /*
    * Send the list of demo commands to control display behavior
    */
    for(Temp = 0; Temp < (sizeof(TestDisplayCMDList)/sizeof(uint8_t*)); Temp++)
    {
        I2C_WriteByteCmd(CONTROL_BYTE_IC1, BYTE_0_ADDRESS,
TestDisplayCMDList[Temp][0]);
        I2C_WriteByteCmd(CONTROL_BYTE_IC1, BYTE_1_ADDRESS,
TestDisplayCMDList[Temp][1]);
        I2C_WriteByteCmd(CONTROL_BYTE_IC2, BYTE_0_ADDRESS,
TestDisplayCMDList[Temp][2]);
    }
}

```

## Static LCD Driver with I2C Interface

```

        I2C_WriteByteCmd(CONTROL_BYTE_IC2, BYTE_1_ADDRESS,
TestDisplayCMDList[Temp][3]);

        FlashLed(100, 1500);
    }

    /*
    * flashing led fast - finish the code
    */
    while(1)
    {
        Result = DisplayValue / 1000;           // get thousand digit
        Digit1 = (uint8_t) Result;

        Result = DisplayValue % 1000;          // get hundred digit
        Result /= 100;
        Digit2 = (uint8_t) Result;

        // from digit 1 and 2 values mount byte 0 value
        TempByte0 = Digit1ByteLUT[Digit1][0];
        TempByte0 |= Digit2ByteLUT[Digit2][0];
        // from digit 1 and 2 values mount byte 1 value
        TempByte1 = Digit1ByteLUT[Digit1][1];
        TempByte1 |= Digit2ByteLUT[Digit2][1];

        // write new byte values to IC1 - update display
        I2C_WriteByteCmd(CONTROL_BYTE_IC1, BYTE_0_ADDRESS, TempByte0);
        I2C_WriteByteCmd(CONTROL_BYTE_IC1, BYTE_1_ADDRESS, TempByte1);

        Result = DisplayValue % 100;
        Result /= 10;
        Digit3 = (uint8_t) Result;

        Result = DisplayValue % 10;
        Digit4 = (uint8_t) Result;

        // from digit 3 and 3 values mount byte 0 value
        TempByte0 = Digit3ByteLUT[Digit3][0];
        TempByte0 |= Digit4ByteLUT[Digit4][0];

        // from digit 3 and 4 values mount byte 1 value
        TempByte1 = Digit3ByteLUT[Digit3][1];
        TempByte1 |= Digit4ByteLUT[Digit4][1];

        // write new byte values to IC2 - update display
        I2C_WriteByteCmd(CONTROL_BYTE_IC2, BYTE_0_ADDRESS, TempByte0);
        I2C_WriteByteCmd(CONTROL_BYTE_IC2, BYTE_1_ADDRESS, TempByte1);
        // fast led flash
        FlashLed(250, 250);

        TempByte1 |= 0x40;                       // set col segment

        // write command to set col segment
        I2C_WriteByteCmd(CONTROL_BYTE_IC2, BYTE_1_ADDRESS, TempByte1);
        // fast led flash
        FlashLed(250, 250);
        // increment display value

```

---

Static LCD Driver with I2C Interface

```
        DisplayValue++;  
    }  
}
```

**Revision History**

<b>Revision</b>	<b>Date</b>	<b>Description</b>
1.0	16-Oct-2018	Initial Version

## IMPORTANT NOTICE AND DISCLAIMER

RENESAS ELECTRONICS CORPORATION AND ITS SUBSIDIARIES (“RENESAS”) PROVIDES TECHNICAL SPECIFICATIONS AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF THIRD-PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for developers who are designing with Renesas products. You are solely responsible for (1) selecting the appropriate products for your application, (2) designing, validating, and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. Renesas grants you permission to use these resources only to develop an application that uses Renesas products. Other reproduction or use of these resources is strictly prohibited. No license is granted to any other Renesas intellectual property or to any third-party intellectual property. Renesas disclaims responsibility for, and you will fully indemnify Renesas and its representatives against, any claims, damages, costs, losses, or liabilities arising from your use of these resources. Renesas' products are provided only subject to Renesas' Terms and Conditions of Sale or other applicable terms agreed to in writing. No use of any Renesas resources expands or otherwise alters any applicable warranties or warranty disclaimers for these products.

(Disclaimer Rev.1.01)

### Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,  
Koto-ku, Tokyo 135-0061, Japan  
[www.renesas.com](http://www.renesas.com)

### Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

### Contact Information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit [www.renesas.com/contact-us/](http://www.renesas.com/contact-us/).