Electrical and Computer Engineering
PO Box 23874 | Doha, Qatar
341G Texas A&M Engineering Building | Education City
(Office) +974.4423.0194| (Fax) +974.4423.0064 | GMT+3

Reference Manual
ZedboardOLED Display Controller IP v1.0

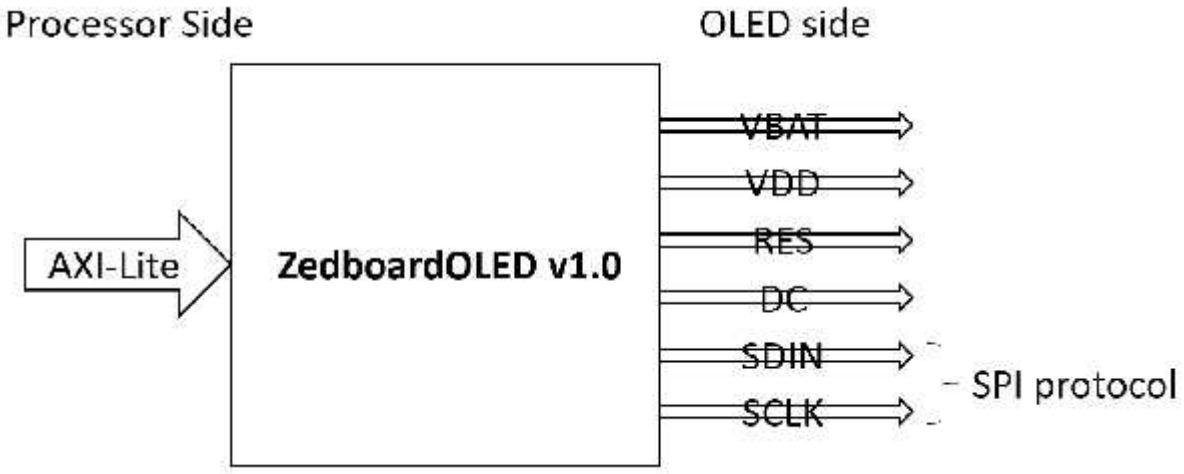
Overview

The ZedboardOLED controller is responsible for initializing the OLED display panel according to the manufacturer's specifications and instructions. The initialization is archived by sending bursts of bytes (commands) separated by measured time intervals through a SPI interface.

After the initialization is complete, the controller provides the processor system access to the OLED display buffer through memory-mapped registers.

The controller is a slave AXI-peripheral, with seventeen software-accessed register of 32-bits each, sixteen of which are data registers, while the seventeenth one is for control. The AXI interface is used to connect the controller to any AXI-crossbar compliant processor system.

On the other hand, the ZedboardOLED communicates with the OLED display panel through SPI interface.

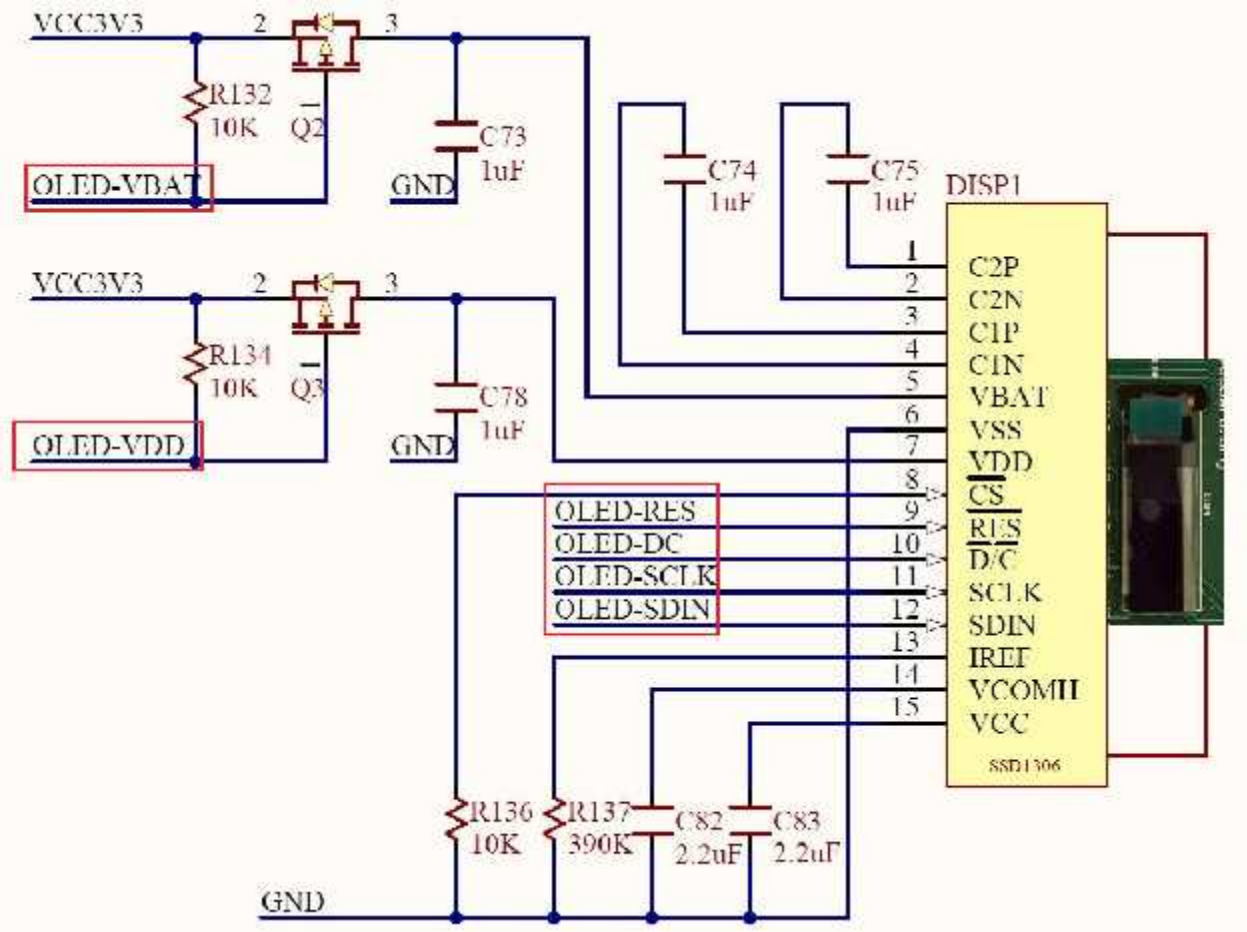


Electrical and Computer Engineering
PO Box 23874 | Doha, Qatar
341G Texas A&M Engineering Building | Education City
(Office) +974.4423.0194| (Fax) +974.4423.0064 | GMT+3

Background

- **OLED display panel**

The OLED display panel hosted on the Zedboard, has 32x128 pixels, the electrical interface of the panel is shown in the figure below:



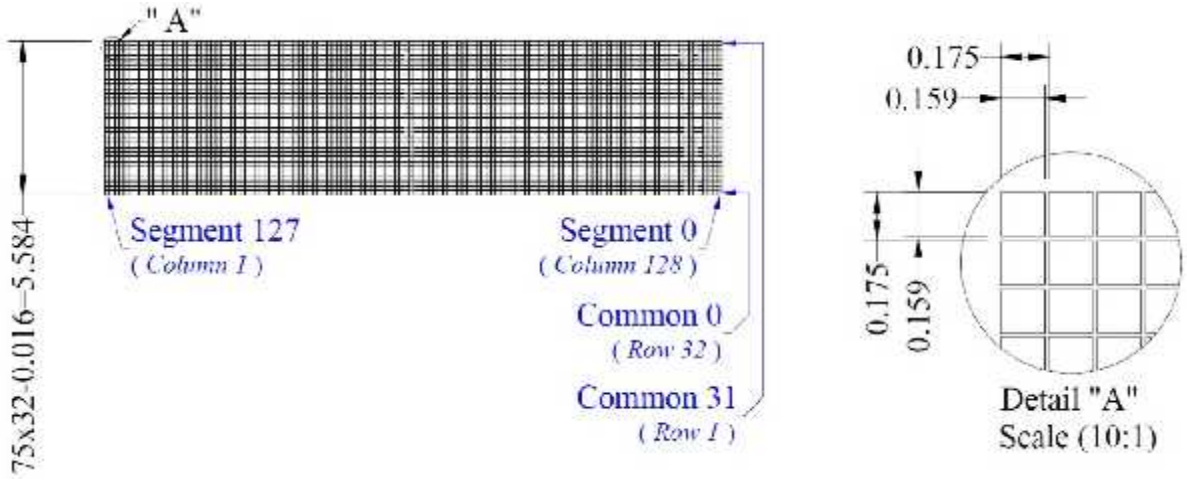
As can be observed from the figure, The OLED panel itself, uses the SSD1306 Segment/Common driver chip(Datasheet included), which deals with the pixels of the panel directly. It has among other things, a display buffer that holds the bitmaps to be displayed, as well as, a command register to hold the command being sent over the SPI interface. The SSD1306 interface signals are summarized in the following table:

Electrical and Computer Engineering
PO Box 23874 | Doha, Qatar
341G Texas A&M Engineering Building | Education City
(Office) +974.4423.0194| (Fax) +974.4423.0064 | GMT+3

VBAT	Display voltage control
VDD	Logic Voltage Control
DC	Data/Command Control
RES	Power Reset
SDIN	SPI Data In (MOSI)
SCLK	SPI Clock

The panel has a ***D/C*** pin (display or command select) that determines whether bytes sent to the display are interpreted as commands or as display data. The **D/C** pin is set high for display buffer access and low for command register access. The **RES** pin is used to reset the SSD1306 display chip. The **RES** pin is driven low for reset and driven high for normal operation. The low-going reset pulse must be a minimum of 3microseconds in duration for the chip to reset correctly.

As we stated previously, the screen is of 32x128 pixels with the physical dimensions illustrated below:



Different shapes can be displayed on the screen by storing different bitmaps in the display buffer (which is named as Graphic Display Data RAM (GDDRAM) in the datasheet of the SSD1306)

The panel is a serial device that is accessed using Serial Peripheral Interface protocol (SPI).

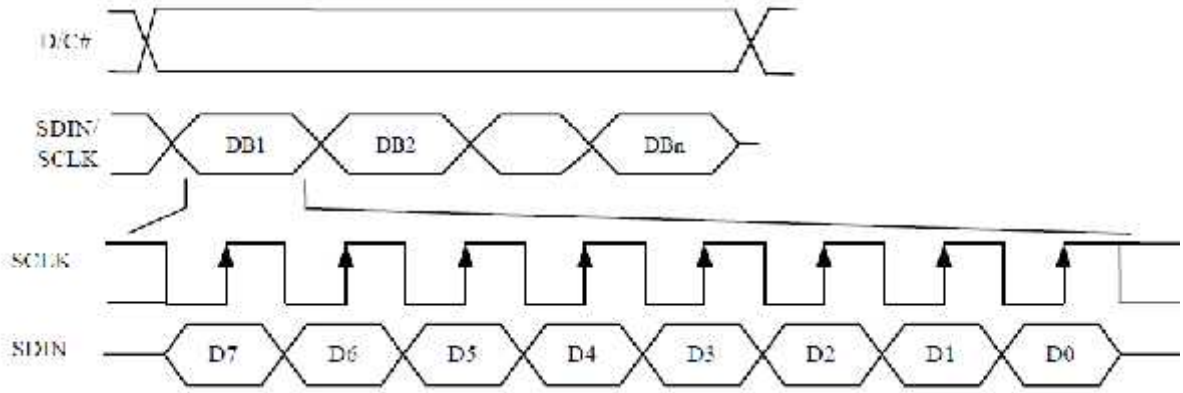
- **SPI protocol**

Data or commands are transferred as bursts of bytes (8-bits) to the panel using the SPI protocol, SPI is commonly used as a method to exchange data between chips on a printed circuit board, and it is usually adopted for short distance, low bandwidth, and single master communication.

The master (in our case is the ZedboardOLED_V1.0) generated a clock (SCLK) and shifts a bit of a data or command on port **SDIN** on each rising edge of **SCLK**.

Electrical and Computer Engineering
PO Box 23874 | Doha, Qatar
341G Texas A&M Engineering Building | Education City
(Office) +974.4423.0194| (Fax) +974.4423.0064 | GMT+3

On the other hand, the slave (the panel –SSD1306) samples every eighth clock and the data byte in the shift register is written to the display buffer or command register in the same clock.



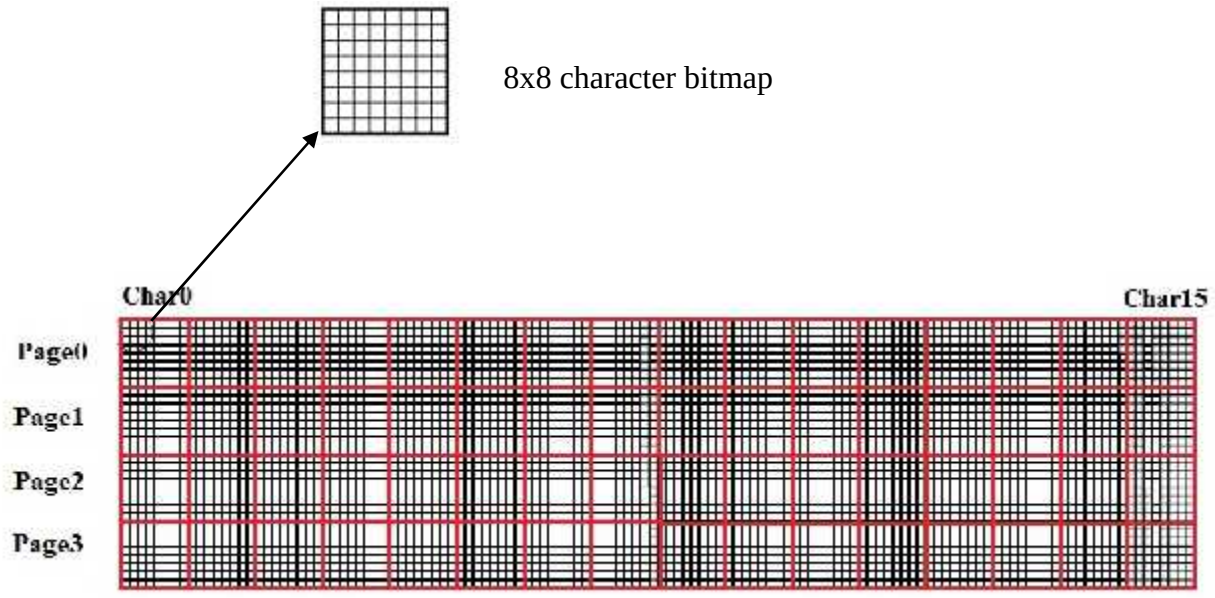
The ZedboardOLED controller

- **The hardware**

The responsibilities of the controller can be summarized by the following points:

1- Implement the AXI-interface (type: Lite-slave).
2- Implement the SPI protocol. (type: master).
3- Initialize and power-up the screen by following the manufacturer instructions.
4- Logically divide the screen into 4 pages, each page can be used to display 16 character, where each character is represented by 8x8 bitmap.

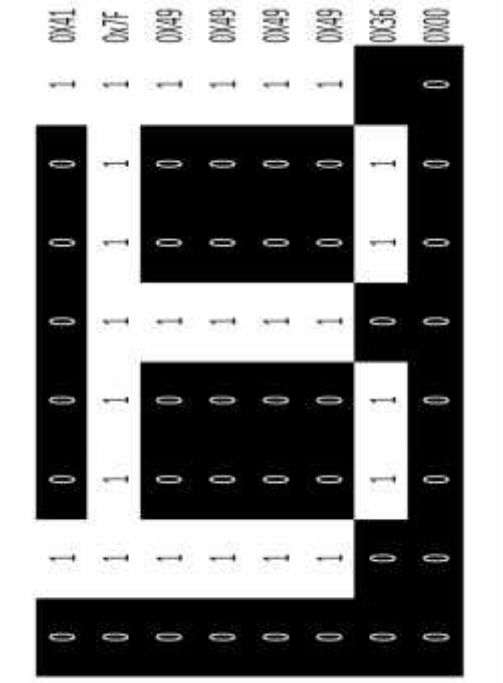
Electrical and Computer Engineering
PO Box 23874 | Doha, Qatar
341G Texas A&M Engineering Building | Education City
(Office) +974.4423.0194| (Fax) +974.4423.0064 | GMT+3



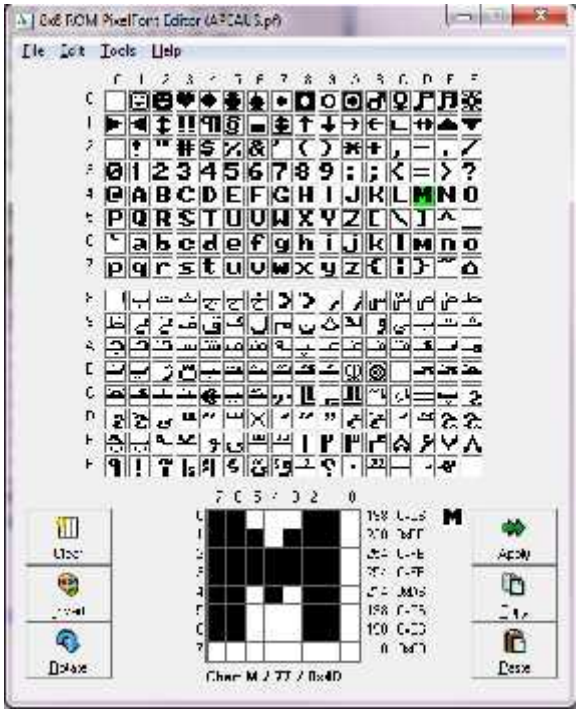
Note:

It is important to realize that the controller needs to sends 8 bytes of information to the screen every time it wants to display a character , the sequence followed to send the information, is set by the controller to by column by column, for instance if the controller wants to display the letter “B” on the screen , it needs to send the following bytes in this order: 0x41,0x7f,0x49,0x49,0x49,0x49,0x36,0x00.

Electrical and Computer Engineering
PO Box 23874 | Doha, Qatar
341G Texas A&M Engineering Building | Education City
(Office) +974.4423.0194| (Fax) +974.4423.0064 | GMT+3



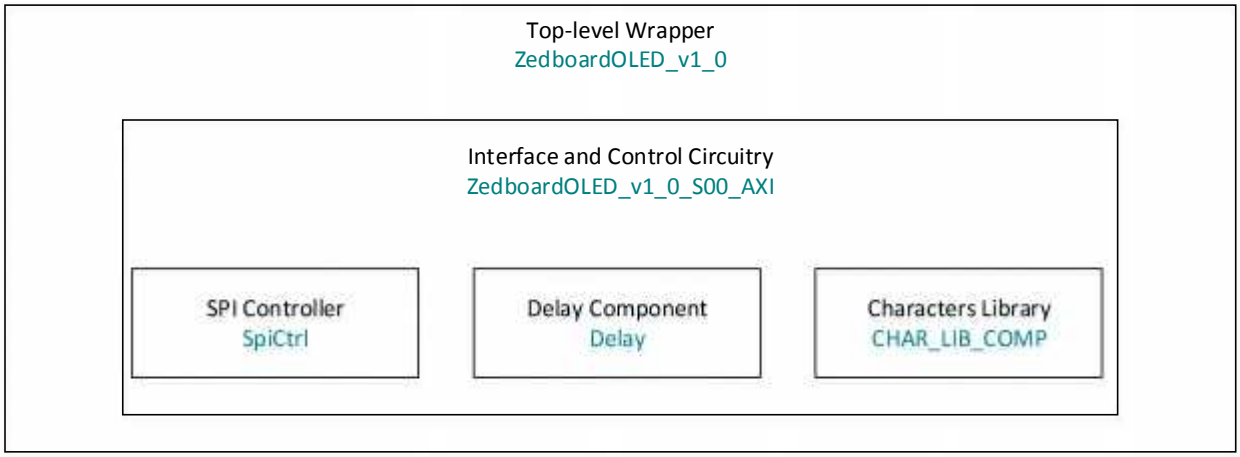
Characters and symbols in different fonts for different languages, have different bitmaps, check **application 8x8 Pixel ROM Font Editor**.



Electrical and Computer Engineering
PO Box 23874 | Doha, Qatar
341G Texas A&M Engineering Building | Education City
(Office) +974.4423.0194| (Fax) +974.4423.0064 | GMT+3

5- Stores the bitmaps of the English alphanumeric.
6- Respond to requests made by the processor, these requests could be to display something on the screen or to clear it up.
7- Provide seventeen 32-bits software-accessed registers, sixteen of which are data registers, while the seventeenth one is for control. These registers are memory-mapped using the AXI-interface.

The controller is implemented using Verilog, it is a hieratical design that delegates different functionalities to different blocks, an abstract hieratical view of the controller is shown be below



Generally speaking, these modules are implemented as finite state machines, a brief description of each module is provided here:

The ***SpiCtrl*** module uses the 100 MHz clock supplied by the PS, to generate a 3.125 MHz SCLK (Serial Clock). This module implements the SPI protocol, every time the ZedboardOLED controller needs to perform a SPI transaction, this module will be involved. This module waits until *SPI_EN* is asserted, active high, and then transitions into the *Send state*. The module starts shifting out the byte currently held in *SPI_DATA* to *SDO* (Serial Data Out) on the rising edge of *SCLK*. The module then transitions into the *Done* state at which the *SPI_FIN* signal is pulled high. The module waits for the *SPI_EN* signal to be de-asserted at which time the module transitions into the *Idle* state and *SPI_FIN* is pulled low.

The ***Delay*** module uses the 100 MHz clock supplied by the PS, to generate a 1KHz counter which is used to count milliseconds, this module provides precise timing capabilities for the controller.

Electrical and Computer Engineering
PO Box 23874 | Doha, Qatar
341G Texas A&M Engineering Building | Education City
(Office) +974.4423.0194| (Fax) +974.4423.0064 | GMT+3

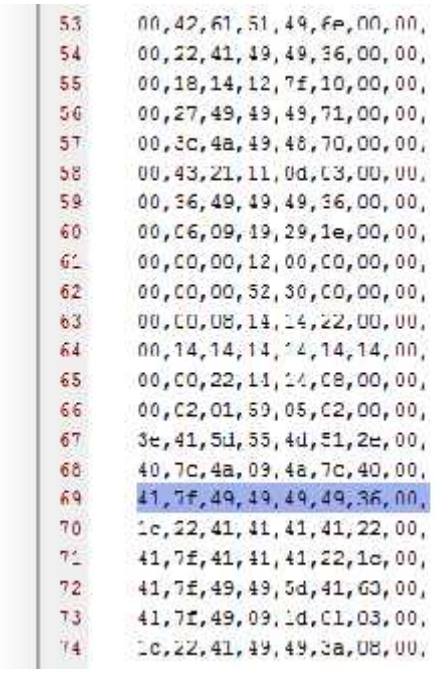
This module waits until *DELAY_EN* is asserted, active high, and then transitions into the *Hold* state. The 1KHz counter then starts counting and when the counter is equivalent to the *DELAY_MS* vector. The controller then transitions into the *Done* state at which the *DELAY_FIN* signal is pulled high. The controller waits for the *DELAY_EN* signal to be de-asserted.

The ***CharLib*** is the block memory that contains the bitmaps for the English alphanumeric . The characters are contained in 8 byte parts(as we stated before each character is an 8x8 bitmap). The addressing is 11 bits and is formatted as followed: ("ASCII Value & "XXX") where the three bits at the end represent the 8 parts of the character. For example the address "01000001001" is the second byte of the character „A".

An important design file associated with the block memory is the ***CharLib.coe*** , the coefficient file for the block memory, the content of this file will be stored in the block memory inside the controller, here the bitmaps of the characters and numbers is saved in locations related to their ASCII code, take a look at how letter „B" is save in the block memory, this will give you an idea on how to add new characters:

The ASCII value of the letter B is "66" , there is a fixed offset of 3 in the coefficient file, therefore, the letter B is saved at address (66+3) =69

The 41,7f,49,49,49,49,36,00 are the bitmaps for the letter B.



Electrical and Computer Engineering
PO Box 23874 | Doha, Qatar
341G Texas A&M Engineering Building | Education City
(Office) +974.4423.0194| (Fax) +974.4423.0064 | GMT+3

The ***ZedboardOLED_v1_0_S00_AXI*** module implements the main functionality of the controller, including the AXI protocol interface, managing the data registers (*slv_reg0- slv_reg15*) and the control register (*slv_reg16*).

A finite state machine is used to go through the initialization sequence starting from state "*Idle*", going through 25 initializations states, the initialization sequence ends with the "*ResetOff*" state, and then the module enters the *WaitRequest* state which keeps waiting for a triggers on the *slv_reg16* control register to initiate an event, the event could be displaying a character on the screen , a string of characters or clearing the screen.

Lastly, the ***ZedboardOLED_v1_0,*** this is the top level wrapper for the controller, here you find the declarations of the AXI interface ports, as well as the OLED display ports.

- **The driver**

A driver (***ZedboardOLED_v1_0***) was developed to communicate with the controller, and provide a user friendly interface to the 32x128 organic LED display. The implementation of the driver is in *ZedboardOLED.c* file, the prototypes of the functions can be found in the header file *ZedboardOLED.h* , to be able to use the controller you need to include the header file.

Electrical and Computer Engineering
PO Box 23874 | Doha, Qatar
341G Texas A&M Engineering Building | Education City
(Office) +974.4423.0194| (Fax) +974.4423.0064 | GMT+3

The figure below summarizes the three main functions of the driver :

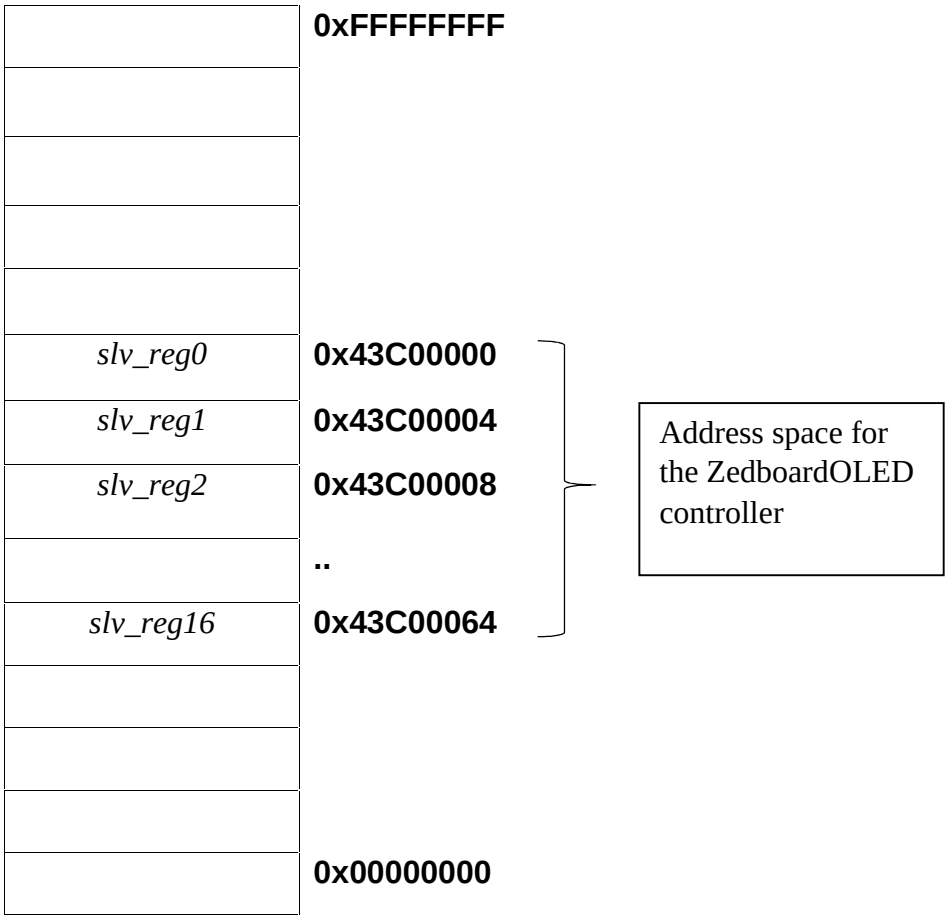
```
/*  driver functions for ZedboardOLED IP core */
/*****************************************************************************/
/**
*
* prints a character on the OLED at the page and the position specified by the second
* and third arguments,example print_char('A',3,0);
*
* @param    char char_seq , the character to be printed.
*
* @param    unsigned int page(0-3) , the OLED is divided into 4 pages , 0 is the upper page
*           3 is the lower page.
* @param    unsigned int position(0-15) , each page can hold 16 characters
*           0 is the leftmost , 15 is the rightmost
*
* @return   int , 1 on success , 0 on failure.
*
******************************************************************************/
int print_char( char char_seq, unsigned int page ,unsigned int position);

/*****************************************************************************/
/**
*
* prints a string of characters on the OLED at the page specified by the second
* argument, maximum string per page =16,example: print_char("Texas A&M Qatar,0);
*
* @param    char *start , the string message to be printed , maximum 16 letters.
*
* @param    unsigned int page(0-3) , the OLED is divided into 4 pages , 0 is the upper page
*           3 is the lower page.
*
* @return   int , 1 on success , 0 on failure.
*
******************************************************************************/
int print_message(char *start , unsigned int page);

/*****************************************************************************/
/**
*
* clears the screen, example: clear();
*
*
* @param    none.
*
* @return   none.
******************************************************************************/
void clear(void);
```

Electrical and Computer Engineering
PO Box 23874 | Doha, Qatar
341G Texas A&M Engineering Building | Education City
(Office) +974.4423.0194| (Fax) +974.4423.0064 | GMT+3

It is crucial to emphasize that the data registers (*slv_reg0- slv_reg15),* and the control register (*slv_reg16*) of the controller are mapped to the memory address space, each register is 32 bits in length (4 bytes), this is why the addresses for these registers are of multiple of 4.



Only two bits of the control register *slv_reg16* was used:

D: Display

1 : Initiate display.

0 : Do nothing.

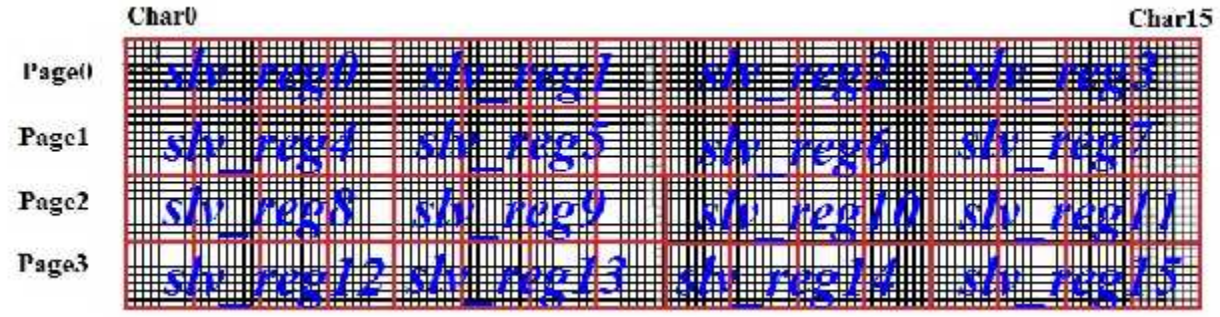
Electrical and Computer Engineering
PO Box 23874 | Doha, Qatar
341G Texas A&M Engineering Building | Education City
(Office) +974.4423.0194| (Fax) +974.4423.0064 | GMT+3

C: Clear

1 : Initiate clear.

0 : Do nothing.

As for the data registers, they ought to be filled with the ASCII codes of the characters to be displayed. Each data register is responsible for four characters on the screen, and as shown in the figure below:



For instance, if the *slv_reg0* register was filled with the following sequence of bits (*01000101 01000011 01000101 01001110)*$_2$ which is equivalent to the decimal 69 67 69 78. The decimal 69 67 69 78 represents the ASCII code for “ECEN”.

0	1	0	0	0	1	0	1	0	1	0	0	0	0	1	1	0	1	0	0	0	1	0	1	0	1	0	0	1	1	1	0
DB0								DB1								DB2								DB3							

Electrical and Computer Engineering
PO Box 23874 | Doha, Qatar
341G Texas A&M Engineering Building | Education City
(Office) +974.4423.0194| (Fax) +974.4423.0064 | GMT+3

Assuming the ***D*** bit of the control register is set high, this will trigger the controller to start bombarding the display buffer of the OLED with the bitmaps of the characters, 256 bits of data will be sent over the SPI interface to the screen, causing the following message to be displayed on the screen.

The task of the driver is to simplify the interface to the controller, instead of going through the tedious process of writing these bits individually, and worrying about moving to the position of the next character every time we wanted to display a characters on the screen, the driver can do it on our behalf and simplify our lives.

The driver backbone function is the `Xil_Out32(Address, Value);` shipped with the Xilinx board support package(BSP) to perform a write operation for a 32-bit memory location by writing the specified *Value* to the specified *Address*.

Electrical and Computer Engineering
PO Box 23874 | Doha, Qatar
341G Texas A&M Engineering Building | Education City
(Office) +974.4423.0194| (Fax) +974.4423.0064 | GMT+3

Take a look at the implementation of the `print_char()` function, implemented by the driver :