NMAM INSTITUTE OF TECHNOLOGY

NITTE-574110

DEPARTMENT

OF

ELECTRONICS AND COMMUNICATION ENGINEERING

MINI PROJECT ON

DEVELOPMENT OF API FOR VGA

Submitted By

MEFIL D'SOUZA	**4NM12EC078**
MOHITH PRABHU M	**4NM12EC080**
N AKSHAY	**4NM12EC082**
PRAJWAL SHARMA K	**4NM12EC101**

Under the guidance of

Mr. SUKESH RAO M,

Associate Professor, E&C DEPT.

Content

Page No.

1. INTRODUCTION

An API is an abbreviation for Application Program Interface. It is analogous to a function in C language. Creation of an API involves the task of creating a header file indicating the class of functions that are to be used in the API creation, creating a .c file where we write the function description along with specifying the input and output parameters and their data types in embedded C. After successfully including these files containing the function definition, in libulk.a, we may use them in our program.

VGA stands for Video Graphic Array that is used display videos by connecting to the processor that sends the necessary signals to it, required for the display. A VGA port connector is a 15 pin connector sending 15 signals, of which 5 are active signals that are of utmost importance. They include the three colour signals namely RGB (red, green, blue) and horizontal and vertical synchronization pulses.

Therefore creating an API for VGA would include five functions corresponding to these five key signals that would have to be invoked in our main program as per the required display of the chosen character(s) on the screen. On the ARM board, these signals are directly connected to the video DAC (THS8135PHP) from OMAP 3530 that gives the corresponding analog value to the VGA pins. The signals that are to be level converted from 1.8V to 5V are also directly given to the level convertor circuit present on the ARM board. Based on this, the task of accomplishing the creation of the API was divided into 4 modules, first being the study of the VGA port, second being the creation of the required header files, third being the creation of .c file and the fourth being the actual code where the built API's were used to get the required display. But due to the lack of information about the port address of VGA on the ARM board, the task couldn't be realized. The other option that was relied on, was the use of GPIO (General Purpose Input Output) pins.

Since the API's for GPIO pins are already built, we would use them to get the required functionality for the five active signals of VGA. The earlier modules that were made, had to be changed as per the new approach taken. The GPIO pins, on configured to be acting as output pins, give a digital output of either a logic '0' or '1' or equivalently 0V or 5V, as per the values decided by the programmer in the code. But certain pins of VGA require analog values. So it is necessary to use an external circuit, namely the digital to analog converter (popularly called the DAC) to achieve task of getting the required analog values.

So, the modules 2,3 and 4 were reframed to the second being study of GPIO pins and the available API's for them, third being design of driver circuit necessary for the interface and the fourth being coding part for the GPIO pins using the already available API's

Again the drawback that had crept in was regarding the generation of the five vital signals that required high frequency which was unable to be matched (since we require a frequency of 25.175MHz and the board is able to give a maximum of only 4MHz) for the required refresh rate of 60Hz for the 640 x 480 standard resolution monitor. So with all these modifications made, the main objective of creation of an API for VGA was accomplished by

using already built in API's of GPIO pins, indirectly used to interface with the VGA port. The detailed explanation regarding the working and design is discussed hereafter.

2. PROJECT OBJECTIVES

The task of designing an Application Programming Interface (API) for Video Graphics Array (VGA) can be divided into few modules based on the various parts included in the design process.

The below mentioned steps are few, identified in the development process and are divided into modules.

Sl.no.	**Name and USN**	**OBJECTIVE**
1.	MEFIL D'SOUZA (4NM12EC078)	WORKING OF THE VGA PORT
2.	N AKSHAY (4NM12EC082)	GENERAL PURPOSE INPUT AND OUTPUT PORT (GPIO)
3.	MOHITH PRABHU M (4NM12EC080)	DRIVER CIRCUIT DESIGN FOR VGA INTERFACE
4.	PRAJWAL SHARMA K (4NM12EC101)	CODE FOR VGA

3. WORKING OF THE VGA PORT

3.1 INTRODUCTION:

VGA Stands for "Video Graphics Array". It is the standard monitor or display interface used in most PCs. The VGA standard was originally developed by IBM in 1987 and allowed for a display resolution of 640x480 pixels. Since then, many revisions of the standard have been introduced. The most common is Super VGA (SVGA), which allows for resolutions greater than 640x480, such as 800x600 or 1024x768. In this module, the interface with a resolution of 640x480 is being discussed. A standard VGA connection has 15 pins and is shaped like a trapezoid as shown in Fig.1a.

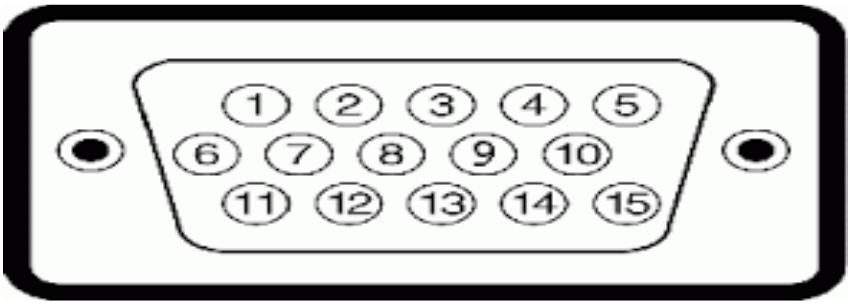

Fig.1a: A standard VGA connector with 15 pins.

3.1.1Pin descriptions:

Pin number	Description
1.	Analog signal (0.7V-1V), used to control red colour.
2.	Analog signal (0.7V-1V), used to control green colour.
3.	Analog signal (0.7V-1V), used to control blue colour.
4.	Not used.
5.	Ground.
6.	Red ground.
7.	Green ground.
8.	Blue ground.
9.	Not connection.
10.	Sync ground.
11.	Ground.
12.	Not used.
13.	HSYNC, a digital signal (0V or 5V), used for synchronization of video.
14.	VSYNC, a digital signal (0V or 5V), used for synchronization of video.
15.	Data clock (not used in our case).

Certain terms to know are:

1. **Screen resolution**: It is the number of picture elements or pixels present in each row x the number of pixels in each column of the display screen. That is, when we say a 640 x 480 resolution monitor, it means that the monitor is characterized by 640 pixels in each row and 480 pixels in each column, as indicated in Fig.1b. Resolution is hence a measure of the highest clarity of the picture that could be displayed on the device.

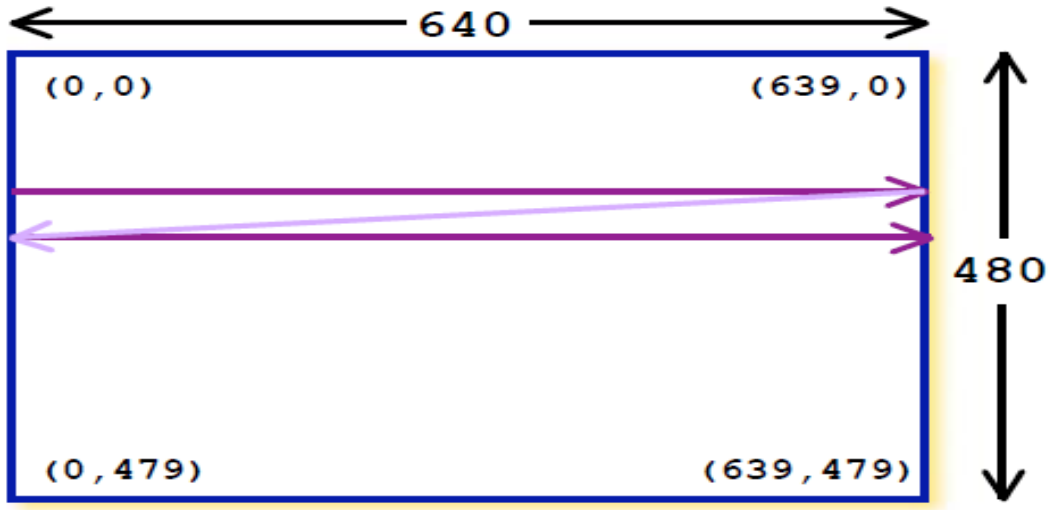

Fig.1b: Diagram indicating a standard monitor with 640 x 480 resolution.

2. **Refresh rate**: This is the rate at which a full frame of pixels (640 x 480) is redrawn per second. Standard monitor refresh rate is 60Hz. By this we mean that the entire frame of pixels is redrawn every $1/60^{th}$ of a second. When many frames are drawn at this rate we get the impression of objects moving in the video.

3.2 METHODOLOGY:

The video signal therefore contains 5 active signals as evident from the pin description table. They include the RGB (red, green, blue) colour signals and the HSYNC and VSYNC signals. Any image can be displayed by varying the analog values of the RGB signals and by appropriately generating the digital signals, HSYNC and VSYNC to ensure that the specified pixel is active at the right time along with the refresh rate being met. Although pixels are made on one at a time, we get the impression that all are on at the same time because the monitor refreshes so quickly. This is why old monitors with low refresh rates flicker.

3.2.1Detailed mechanism of display:

1. The screen refresh process begins and the pixel 1 having the coordinate (0,0) is painted first. The process continues from left to right i.e., a row of pixels gets painted. At the end of the first row, the row increments and the column address is reset to the first column. The process continues till the last pixel having coordinates (639,439) is coloured. Once the entire screen has been painted, the refresh process begins again.

2. The video signal must redraw the entire screen 60 times per second to provide for motion in the image and to reduce flicker: this period is called the refresh rate as discussed earlier. Refresh rate of 60 Hz is used in our case that corresponds to the refresh rate of a normal PC monitor.

3. The vertical sync signal tells the monitor to start displaying a new image or frame, and the monitor starts the display process in the upper left corner with pixel (0,0).

4. The horizontal sync signal tells the monitor to refresh another row of 640 pixels. After 480 rows of pixels are refreshed with 480 horizontal sync signals, a vertical sync signal resets the monitor to the upper left corner and the process continues. This process is illustrated in Fig.1c.

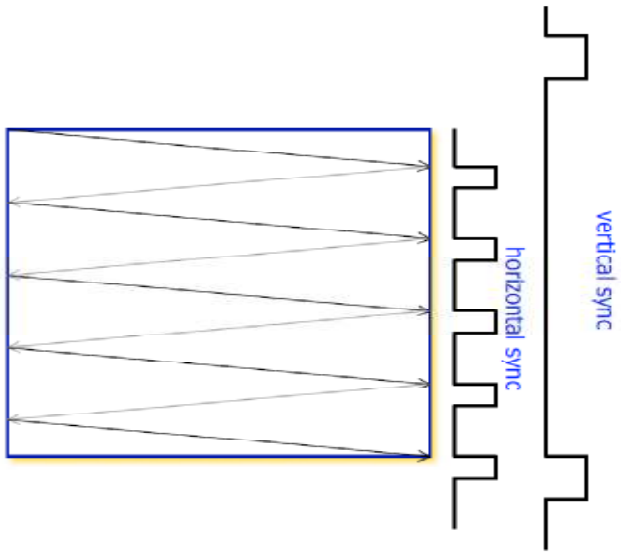

Fig.1c: Illustrating the process of displaying data.

3.2.2 What are the specifications before coding?

Now we know that, the entire functioning of the display module of VGA is based on understanding the timing of the 2 synchronization signals namely HSYNC and VSYNC. By manipulating these 2 signals and the 3 RGB signals, any image can be displayed on the screen. The horizontal and vertical sync timing diagram is shown in Fig.1d. They are being

derived from a 25.175 MHz oscillator clock frequency and the respective number of cycles required for each of these signals is shown.

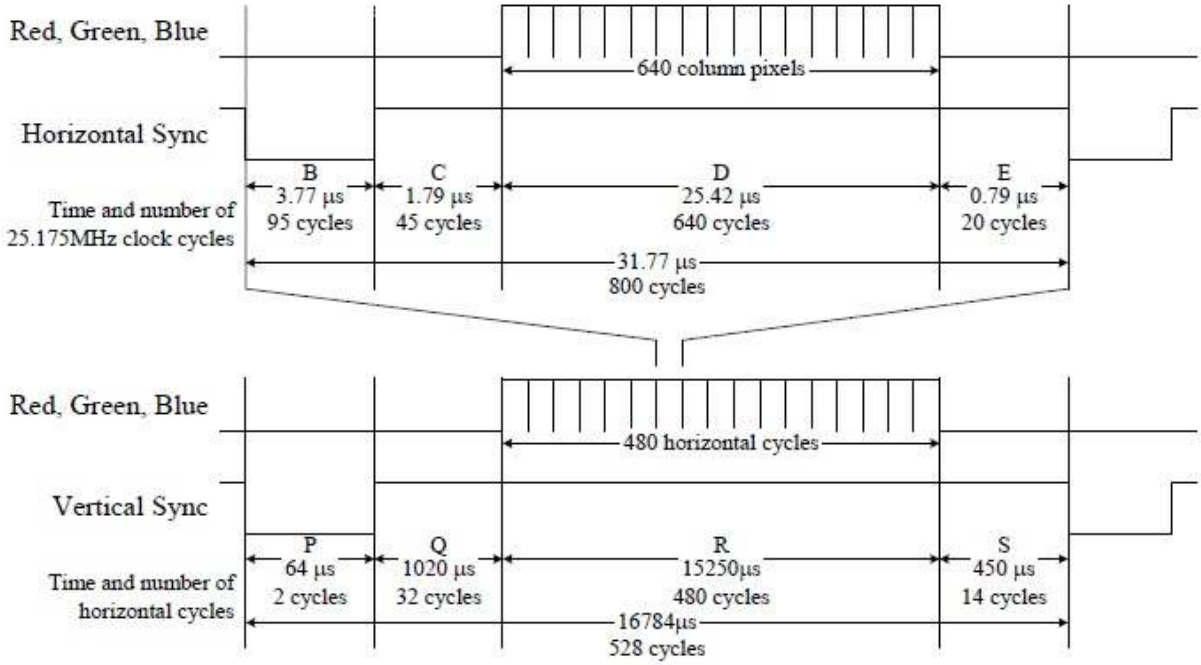

Fig.1d: Timing diagram for horizontal and vertical synchronization pulses, derived from a 25.157 MHz clock.

When inactive both synchronization signals are at logic 1 value (5V). A row scan begins with the HSYNC pulse value going low for 3.77µs (see section B in Figure 4). A 1.79µs high on the signal follows this (see section C in Fig.1d). Next the data for the three colour signals is sent one pixel at a time, for 640 columns for 25.42µs. Finally after the last column pixel, there is another 0.79µs of inactivity on the RGB signal lines for the horizontal retrace before the horizontal sync signal goes low again for the next row scan. The total time required to complete one row scan is 31.77µs.

The timing of the vertical sync signal is analogous to the horizontal signal. The 64µs active low vertical sync signal resets the scan to the top left corner of the screen (see section P in Figure 4). A 1020µs high follows this on the signal. Next there are 480, 31.77µs row scans giving a total of 15,250µs (480 x 31.77) as shown in section R.

Finally, after the last row scan, there are another 450µs before the VSYNC signal goes low again to start another complete screen scan in the top left corner. It takes a total of 16,784µs to complete one full scan. Hence, to get the monitor working properly, we simply have to get the timings of horizontal and vertical sync signals correct and send out the RGB data for each pixel at the right row and column position.

For eg., if we are to turn on a pixel in the array say, at row 50 and column 30 (50,30) then, we have to wait for the scan to reach row 50 and column 30, and then give the highest analog

value of 1V at pin 1, corresponding to intensity of the red signal. This cycle has to repeat every time the screen refreshes ie., 60 times in a second. A different colour for the pixels could be given by varying the intensities of the RGB signals. For instance, we get black colour if all the RGB values are kept at their maximum ie., 1V. If at their minimum ie., 0.7V, we get the pixel turning white. However, the grounds for all these signals are to be given inherently.

3.2.3 Small verification:

Getting the correct timing for the two synchronization signals is easily achieved if correct clock frequency is used. To obtain 640 x 480 screen resolution, we must use a clock frequency of 25.175 MHz. For 25.175 MHz frequency, the period is: $1/ (25.175 \times 10^6)$ which is approximately 0.0397µs per clock cycle. With this as the period of the clock, the number of cycles required for each of the sections is calculated and marked in fig.3. We can verify the correct generation of these pulses by simply calculating the frequency of the VSYNC pulse (VSYNC pulse only because the clock cycles for VSYNC are derived from HSYNC's period. So only if HSYNC generation is correct, then VSYNC could be correct) that must be equal to the screen refresh rate. From the above timing diagram, it is clear that the period of the VSYNC pulse is 16784µs which gives its equivalent frequency as 60Hz, thus matching the screen refresh rate.

3.3 DESIGN AND TESTING:

With all the above specifications regarding the generation of RGB and the HSYNC and VSYNC signals, we may code to get the necessary duty cycles (neglecting voltage levels initially), assuming to be derived from a 25.175 MHz clock source. This design can be achieved in 2 ways.

1. By counting the number of clock cycles required.
   - Get the exact pixel configurations corresponding to the alphabet(s) to be displayed.
   - Derive the HSYNC pulse from the 25.175 MHz clock.
   - Then derive the VSYNC pulse from the sections of HSYNC signal as discussed earlier.
   - Integrate all the 3 parts to the final display.

Since the timing of VSYNC is entirely dependent on HSYNC (because it is derived from it), this design could be called the synchronous design. This is the preferred design because, unless the clock and the HSYNC signals are as per our requirements, VSYNC wouldn't be correct, thereby not matching the standard refresh rate.

2. By defining a standard delay function.
   - Define a standard delay function that would give us a minimal delay of 0.0397µs.
   - Gets the exact pixel configuration corresponding to the alphabet(s) to be displayed.

- Build the HSYNC signal from the standard delay function.
- Similarly build the VSYNC signal from the standard delay function.
- Integrate all the 3 parts to the final display.

Since the two synchronization signals are independently derived, the design could be termed as the asynchronous design. This design is not preferred because, though an error in any signal doesn't affect the other, we may not get the required output, thereby increasing the debugging time. Here for our project we have chosen the 2nd method only due to the absence of an inbuilt clock frequency of 25.175MHz.

3.3.1 The only test condition:

We have a delay function, which gives a delay, only in integral multiples of micro seconds. We may observe that the all the delays that are to be present in between 2 transitions, are in a fraction of micro seconds. Moreover, the major drawback is that the frequency of the ARM board used, is 4MHz. Due to this, the total time the processor takes to execute the smallest delay function (of 1µs) is found to be 24µs. The only solution possible is proportionally increasing all the delay values thereby compensating with the monitor refresh rate. The persistence of vision in our eye is about 1/ 12th of a second, which means that the refresh rate must be at least greater than 12 frames per second to avoid visible flickering. Now if we proportionally increase the delay timings with respect to the minimum delay timing of 24µs, as shown in Fig.1e, it is found that the frequency of the VSYNC pulse comes out to be 7Hz, which means, the new screen refresh rate is 7 frames per second, thereby inevitably introducing visible flicker, leading to a negative test condition.

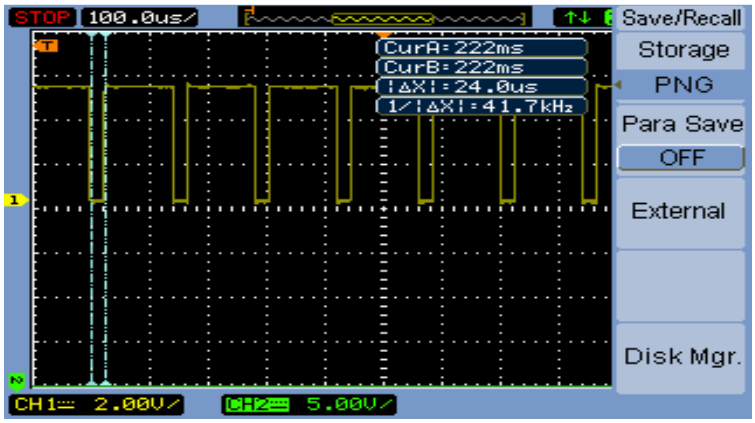

Fig.1e Snapshot of the negative test condition for an ideal delay of 1µs (it inevitably gives 24µs).

4. GENERAL PURPOSE INPUT AND OUTPUT (GPIO)

4.1 INTRODUCTION:

General purpose input and output (GPIO) ports are those ports available in many of the experimental micro controllers/processors kits, which can be configured to act as input or output port based on the requirement of the user. In the ULK trainer kit also, the GPIO ports are available and can be configured as per requirement to send values through these ports or to receive values from this port.

In this project of development of an Application Programming Interface (API) for Video Graphics Array (VGA), we use the available GPIO pins. The necessity of using GPIO comes due to the reason that the VGA port in the trainer cannot be directly accessed.

On a conclusive study on the VGA port available in the ULK trainer kit, we come to know the following highlighting features of it.

- There is no pre-existing API for VGA unlike other functionalities (GLCD, CLCD, I2C etc), forcing us to follow all the steps involved in designing of an API as mentioned below:
  - First creating a library by defining the basic keywords, standard functions.
  - Second step is to create a header file with function definitions as to be used in the main source file. These functions are user defined and are the actual API commands. In future if we want to use the VGA port, calling this particular header file will automatically invoke all functions defined under this file. In general, we can say header files are like database of all necessary functions and are Universal in nature.
  - The final step is to create a source file as per requirement of the user, using API functions defined in the header file. The way and style in which we write source file will give us the desired output. So, source files are flexible in nature.
- If an API is assumed to be created, the next problem is the accessing the pins of VGA port and sending the data. Following observations are made in this regard:
  - The foremost is address lines of VGA and GLCD are same. This implies that, both components use a cascaded version of the address bus from the central OMAP processor.
  - Extensive trials resulted in the preliminary conclusion that, the GLCD is given high priority over VGA port making the address lines available for GLCD in a priority cum authenticated basis. With reference to the hardware reference manual of the ULK trainer kit, no ways were found to make the address lines of VGA active thereby allowing us to access it. This is available to be seen in Fig.4a and Fig.4b.

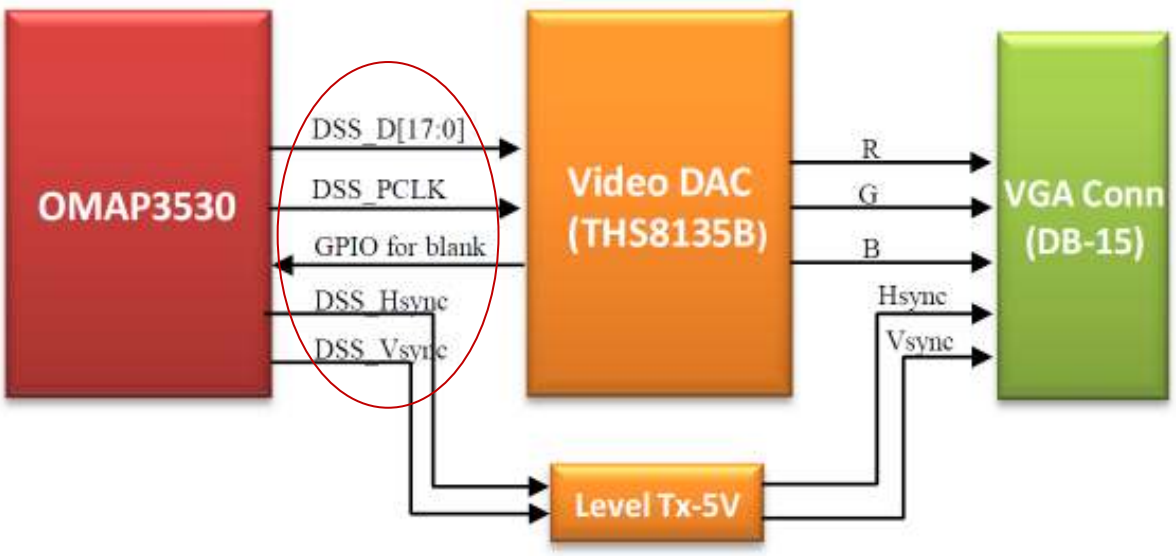

Fig.4a: VGA hardware circuit with OMAP processor.

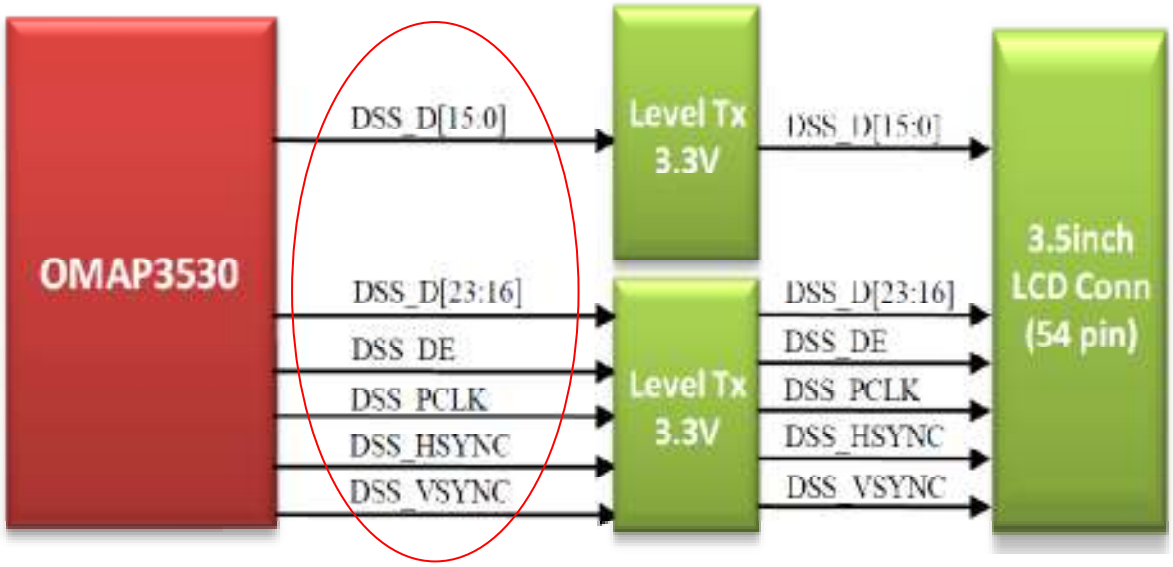

Fig.4b: GLCD hardware circuit with OMAP processor.

- Hence a conclusion was derived that the address lines of VGA are inaccessible and hence if an API is created also, it would be in vain as its functionalities would not work on VGA without this address lines.

With above observations made an alternate way of implementing the project is using the GPIO pins. The following are the contributing factors to select the GPIO method to implement the project:

- GPIO port is having a predefined API with many standard and useful functions declared in its header file. This standard functions can be used to

appropriately model the various GPIO pins as outputs as per requirement. Few standard functions that were found useful are:

- **ulk_proc_gpio_init() –** To initialize all pins of the GPIO port.
- **ulk_proc_gpio_set_dir(x,0) –** To set the pin **'x'** as input or output port.
- **ulk_proc_gpio_set_data_out(x,y) –** To set an output value **'y'** to the already initialized pin **'x'**.
- **ulk_proc_delay() –** A function that allows feasibility to have delay or in simple words a control of timer and timing of the processor.

- The requirement of many pins for R, G, B, HSYNC, VSYNC and GND's which are mostly available in GPIO port only, as there is no other module which has so many user modifiable pins.
- The output from the GPIO pins are standard digital values of 5V high and 0V low, TTL signal making it convenient to modify easily as required.

The above advantages of GPIO makes the creation of an API a possibility for VGA in an alternate way thereby opening up a new interface feasibility in the ULK programming panel.

4.2 METHODOLOGY:

As mentioned we shall use the GPIO pins available in the ULK panel. The following steps describe the sequential way to achieve the required functionality through GPIO using inbuilt API commands of the GPIO.

- **ulk_proc_delay() :**
  - This API function is a part of timer API function set.
  - **Function definition: uint8 ulk_proc_delay(uint32 delay).**
  - Where delay is ULK_USEC, ULK_MSEC, ULK_SEC.
  - Ulk_proc_delay is the API available from timer's header files which is used to generate necessary output time delay.
  - The delay declarations are based on what output time delay is required i.e., for delay in microseconds we use ULK_USEC, for delay in milliseconds we use ULK_MSEC and for delay in seconds we use ULK_SEC.
  - The only drawback of this delay declaration is that the delay that has to be generated should be an integer value i.e., accurate delay by considering floating point values is not possible.
  - There is no API function available for the same and hence a small compromise of the delay generated has to be made.
- **ulk_proc_gpio_init()** :
  - This is an API function available from the GPIO definition set.
  - It forms the basic API to use the GPIO port as this activates the GPIO port to start functioning whenever the complier executes this instruction.

- **Function definition: uint8 ulk_proc_gpio_init(void).**
- The argument is void type. Hence on executing this instruction successfully, this instruction returns a zero value. Any failure in execution of this instruction results in invalid gpio pin error.

- **ulk_proc_gpio_set_dir(x,0) :**
  - This is also an API available from GPIO definition set.
  - Once the GPIO port is activated this pin comes into picture as it decides the fate of the pins of GPIO port i.e., it decides whether the pins must act as output or input.
  - **Function definition: uint8 ulk_proc_gpio_set_dir(uint pin, uint direction).**
  - The integer value of the pin mentioned here gets the direction mentioned by the integer value in the second argument field.
  - Setting direction as '0' will make the pin indicated as output pin and making the direction argument '1' will make it act as input pin. Whereas any other values lead to default input direction of pins mentioned.
  - Successful execution will make the port ready and the this function returns '0' else error is displayed.
- **ulk_proc_gpio_set_data_out(x,y) :**
  - This is also an API available from GPIO definition set.
  - When the pins are assigned with their direction, this API comes to picture when the values have to be sent out.
  - **Function definition: uint8 ulk_proc_gpio_set_data_out(uint8 pin, uint8 value).**
  - The integer value of the pin mentioned here gets the corresponding value in the second argument. This instruction generates a TTL pulse with either a low or high.
  - A low will be assigned to the pin when value takes '0' and a high value will be sent when value is made '1'. Any other values will default low value.

There are few other API functions in the predefined header files to perform few tasks like reset GPIO, data in to GPIO etc. These API's are not used and hence are neglected.

Since we have the knowledge of the necessary API's we shall study the GPIO port further referring to Fig.4c

Fig.4c: GPIO pin map in ULK programming panel.

The pins on the GPIO are to be counted in the fashion shown in the Fig.4c. But in reality this are connected to the OMAP processor in different way as shown in Table.4d.

Pin No.	Signal	Description	Voltage level
1	GPIO140 (UART2_CTS)	Configured as GPIO140	5V
2	GPIO141 (UART2_RTS)	Configured as GPIO141	5V
3	GPIO142 (UART2_TX)	Configured as GPIO142	5V
4	GPIO143 (UART2_RX)	Configured as GPIO143	5V
5	GPT8_PWM_EVT	Configured as PWM signal	5V
6	GPIO158 (McBSP1_DX)	Configured as GPIO158	5V
7	GPIO162 (McBSP1_CLKX)	Configured as GPIO162	5V
8	GND	Ground	
9	GPIO161 (McBSP1_FSX)	Configured as GPIO161	5V
10	GPIO159 (McBSP1_DR)	Configured as GPIO159	5V
11	GPIO156 (McBSP1_CLKR)	Configured as GPIO156	5V
12	GND	Ground	
13	GPIO157 (McBSP1_FSR)	Configured as GPIO157	5V
14	GPIO12 (MMC3_CLK)	Configured as GPIO12	5V
15	GPIO13 (MMC3_CMD)	Configured as GPIO13	5V
16	GPIO14 (MMC3_D4)	Configured as GPIO14	5V
17	GPIO17 (MMC3_D3)	Configured as GPIO17	5V
18	GPIO18 (MMC3_D0)	Configured as GPIO18	5V
19	GPIO19 (MMC3_D1)	Configured as GPIO19	5V
20	GPIO20 (MMC3_D2)	Configured as GPIO20	5V
21	GND	Ground	
22	GPIO22 (MMC3_D6)	Configured as GPIO22	5V
23	GPIO23 (MMC3_D5)	Configured as GPIO23	5V
24	5V	5V power signal	5V
25	5V	5V power signal	5V

Table.4d: GPIO actual pin mapping with OMAP processor.

Once the GPIO user side diagram and GPIO pin mapping is obtained the next procedure would be estimating the total no. Of pins to be needed for the entire design from the GPIO pin set.

It can be done in the following way:

- Since for any display unit especially VGA, control pulses are important like HSYNC, VSYNC, these pulses have to be generated and two GPIO pins are allotted for this purpose.
- Once display is activated by passing the control signals, the next stage would be to lit the screen by passing three basic colours i.e., RED(R), GREEN(G), BLUE(B). Each of this are allotted with two pins thereby summing up the total number of colour pins as 6.
- All the pins require a reference and here it shall be ground. Hence a pin is taken as ground pin for reference.

With the above analysis a total of nine pins of GPIO are identified as necessary. All these pins have to be appropriately controlled in the sequence mentioned previously i.e.,

- The GPIO pins are made active.
- Then the identified nine ports are set as output ports by mentioning appropriate pin numbers with value '0' .
- Once set they have to be assigned with values be it constants defined directly by sending data out or by sending pulses using timer's API.
- When all this steps synchronously work the desired output can be expected on any standard display device.

In this way the GPIO port is found essential in designing of an interface possibility with VGA. On further programming in standard programming procedures the necessary outputs can be seen.

4.3 TESTING:

In this we shall consider few positive and negative test conditions that arise during the designing of GPIO outputs.

4.3.1 POSITIVE TEST CONDITIONS:

1. TEST CASE ID : Pulse_Generation_01

- ENVIRONMENT : output taken from UTLP's GPIO pin (say pin9-userside)
- OUTPUT TO BE RECORDED:
  - If in the pulse generation code, the command is given such that Ton=Toff then a 50% duty cycle square wave with 0-5V has to be generated on the CRO.
- PASS CRITERIA :
  - Based on the values defined in the delay for equal periods of on and off a perfect square wave has to be generated.
  - For e.g. Fig.4e

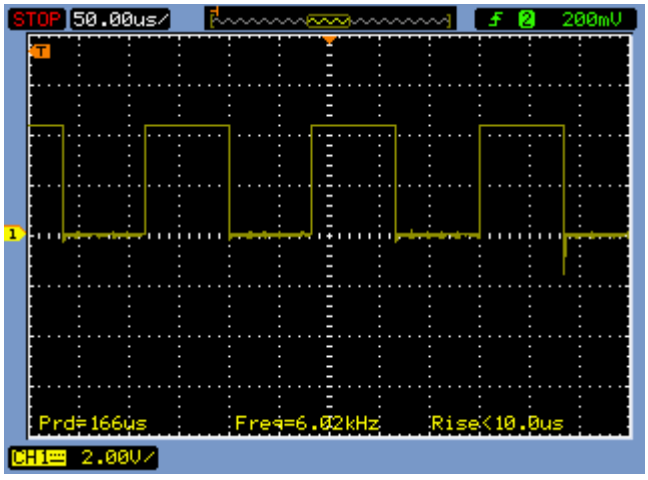

Fig.4e: Symmetrical waveform output for Ton=Toff.

2. TEST CASE ID : Pulse_Generation_02

- ENVIRONMENT : output taken from UTLP's GPIO pin (say pin9-userside)
- OUTPUT TO BE RECORDED:
  - If in the pulse generation code, the command is given such that Ton>Toff then a uneven duty cycle square wave with 0-5V has to be generated on the CRO.
- PASS CRITERIA :
  - Based on the values defined in the delay for different periods of on and off a square wave has to be generated.
  - For e.g. Fig.4f

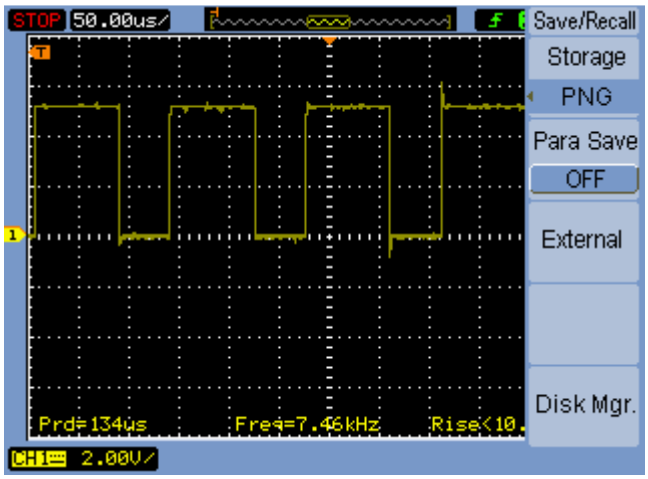

Fig.4f: Non-Symmetrical waveform output for Ton>Toff.

3. TEST CASE ID : Pulse_Generation_03

- ENVIRONMENT : output taken from UTLP's GPIO pin (say pin9-userside)
- OUTPUT TO BE RECORDED:
  - If in the pulse generation code, the command is given such that Ton<Toff then a uneven duty cycle square wave with 0-5V has to be generated on the CRO.
- PASS CRITERIA :
  - Based on the values defined in the delay for different periods of on and off a square wave has to be generated.
  - For e.g. Fig.4g

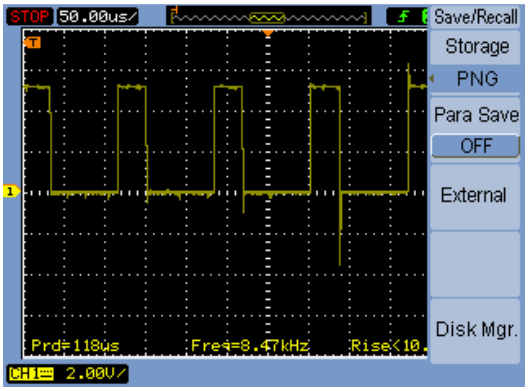

Fig.4g: Non-Symmetrical waveform output for Ton<Toff.

4.3.2 NEGATIVE TEST CONDITION:

4. TEST CASE ID : Pulse_Generation_04

- ENVIRONMENT : output taken from UTLP's GPIO pin (say pin9-userside)
- OUTPUT TO BE RECORDED:
  - If in the pulse generation code, the command is given such that Ton and Toff are specified but not an integer value but a floating point number against the syntax of the API command.
- PASS CRITERIA :
  - Since this is an ambiguous case an erroneous and unexpected functioning might appear. But the test condition is assumed to be passed if such a thing doesn't happen and the signals are generated normally.
  - Any error in signal values are accepted as it is a negative case and on identification it can be corrected.

Hence in this way few possible test conditions are identified and care is taken to avoid negative cases disrupting the output by identifying it appropriately.

5. DRIVER CIRCUIT DESIGN FOR VGA INTERFACE

5.1 INTRODUCTION:

A driver circuit is an electronic circuit used to control an another circuit or component. Driver circuits are usually used to regulate voltage or current flowing through the controlled circuit. In our mini project of developing an API for VGA we have used an external driver circuit which controls the voltage level of analog RGB signals and current rating at the input of VGA.

VGA driver circuit is used to interface standard display devices like monitor, projector etc.VGA requires 3 primary colours red(R), green (G), blue (B) along with HSYNC, VSYNC and respective ground signals. These signals are generated through GPIO port by configuring port pins as output port. The signals generated are digital in nature. These signals cannot be directly fed to VGA port except for HSYNC and VSYNC. It requires an intermediate stage where digital RGB signals are converted to analog form with an approximate match in current rating. These conditions are satisfied using driver circuit.

The ULK ARM board consists of an inbuilt video DAC between the OMAP3530 processor and VGA port which can function as driver circuit. This 10 bit video DAC can convert 24 bit digital parallel RGB signals to analog RGB signals to display on VGA. But video DAC is not used as driver circuit in our mini project. The reason being, unlike graphic LCD, The VGA port of ARM board is not associated with a specific port address. Therefore it is not possible to access VGA port of the ARM board directly. Hence analog RGB values are also not accessible which means that output of video DAC cannot be accessed. The 24 bit digital RGB values are also difficult to access individually. Hence 2 bit digital RGB signals are accessed through GPIO and are converted to analog RGB signals through an external driver circuit.

5.2 METHODOLGY

The VGA driver circuit consists of 4 main blocks

1. Two bit R-2R DAC
2. Voltage divider
3. Adder
4. Current booster

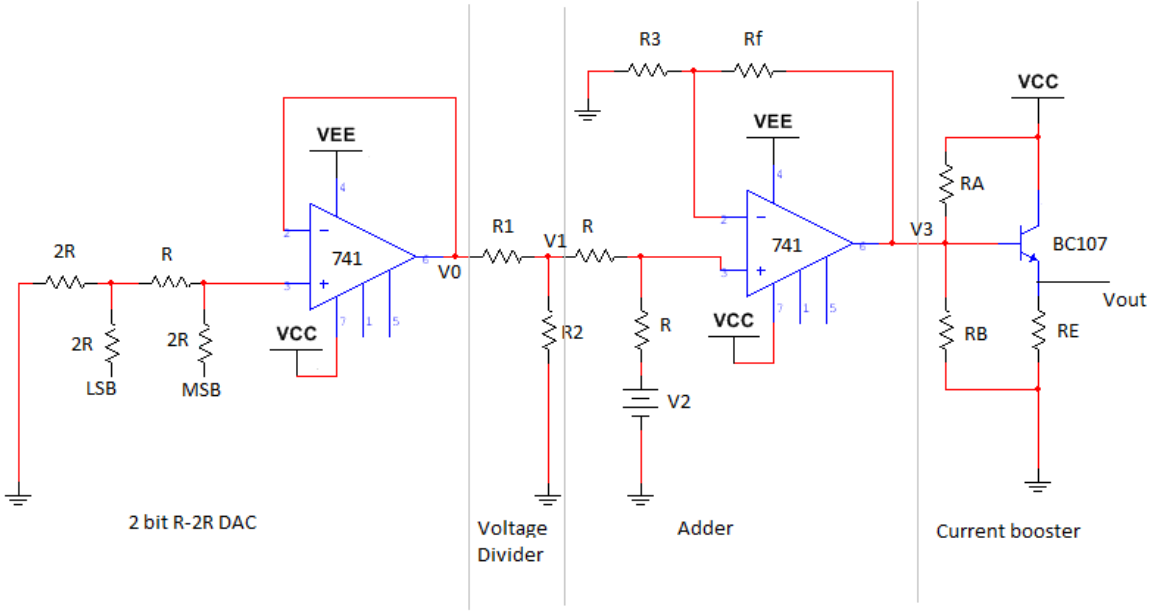

Fig. 5a: Circuit diagram of driver circuit

R =1kΩ, 2R=2.2kΩ, R1=10kΩ, R2=1kΩ, Rf=1kΩ, R3=1kΩ, R_A=10kΩ, R_B=10kΩ, R_E=1kΩ, V2=0.3volts, VCC=12volts, VEE=-12volts

5.2.1. Two bit R-2R DAC:

An R-2R ladder is a simple way to perform digital to analog conversion, using repetitive arrangements of precision resistor networks in ladder like configuration. R-2R DAC is a major part of the VGA driver circuit. A string of equally dimensioned resistors are connected between 2 reference voltages. The resistors act as voltage dividers between referenced voltages. The voltage appearing at the last stage gives the analog equivalent of digital input.

In the driver circuit as in Fig.5a, two bit DAC is used. The reason for using two bit DAC is because of its ease of construction and analysis. Also necessary and sufficient precision is obtained with the same. A simple R-2R ladder itself forms a DAC circuit. But it is followed by a non inverting amplifier which is used for impedance matching and to prevent the load drawing current from the R-2R ladder network.

From Fig.5a, the MSB and LSB of R-2R DAC are switched between logic 0 (0 volts) and logic high (5 volts) (0-5volts is the output voltage range of GPIO).In the circuit 2 bits correspond to total 4 possible analog voltage levels at the output. Depending on which bits are set to 1 and which to 0,the output voltage will be corresponding stepped value between 0 volts and (5 minus value of minimum step).

For a digital value D, of an R-2R DAC of N bits, the output voltage Vo is given by equation,

$$Vo = Vref \times D/2^N$$

In our R-2R DAC, N=2, Vref= 5volts,
If we want to calculate Vo for digital input 11, then D=3

$$\therefore \text{Vo} = 5 \times 3/2^2$$
$$\text{Vo} = 3.75 \text{ volts}$$

which is the maximum analog output voltage possible.
By similar calculations we get,

Digital input	Analog output (in volts)
00	0.00
01	1.25
10	2.50
11	3.75

Table 5b: Output voltage levels of 2 bit DAC

The 4 values of Table 5b represent 4 different intensities of a colour on VGA display. As the number of bits in the digital input increases, intensity level of the colour also increases, giving rise to more clarity in the display. Larger the analog output value then, larger will be the brightness of the colour, smaller the analog output value lesser the brightness of the colour. By varying the intensities of RGB,it is possible to give rise to any of the colour. Maximum intensity of RGB results in pure black. Minimum possible intensity of RGB results in pure white colour.

The gain of the non inverting amplifier connected to DAC is given by,

$$\text{A} = 1 + \text{Rf}/\text{R}'$$

where Rf is the feedback resistance =0Ω
R' = ∞

$$\therefore \text{A} = 1 + 0/\infty$$
$$\text{A} = 1$$

5.2.2. Voltage divider:

The 2 bit R-2R DAC is followed by a voltage divider network in order to limit the output voltage to required value. The voltage level handled by VGA ranges from 0.7volts to 1volt. But the analog output obtained from R-2R DAC ranges from 0volt to 3.75volts.These values

are to be lowered to above specified voltage levels before giving to VGA.This is done by voltage divider network.

The simple voltage divider network consists of 2 resistors R1 and R2 as shown. Ouput of R-2R DAC acts as input for the network at R1.The output is taken across the resistor R2. The voltage divider is designed such that it gives an output voltage range from 0volts to 0.3volts.Then from voltage division formula,

$$V1 = Vo \times R2/(R1 + R2)$$
$$0.3 = 3.75 \times R2/(R1 + R2)$$
$$0.3(R1) + 0.3(R2) = 3.75(R2)$$
$$0.3(R1) = 3.45(R2)$$
$$R1 = 11.5(R2)$$

Let R2=1kΩ then R1=11.5kΩ
i.e R2=1kΩ and R1 ≈10kΩ

5.2.3. Adder:

An adder is used to add 2 or more voltages. It is constructed using IC741 opamp. 2 voltages that are to be added are applied to non inverting terminal of the opamp as shown in Fig.5a. Since the output of voltage divider varies from 0volt to 0.3volts, this range has to be increased to a range 0.7volts to 1volt respectively by adding a constant 0.3volts DC to voltage divider output. 0.3volts is chosen approximately, because a small amount of base-emitter voltage gets added to the input of current booster circuit at the next stage.

Output voltage of the adder is given by,

$$Vout = \frac{\left(1 + \frac{Rf}{R3}\right)(V1 + V2)}{2}$$
$$V3 = \frac{\left(1 + \frac{1}{1}\right)(V1 + V2)}{2}$$
$$V3 = V1 + V2$$

Here V1 varies from 0volt to 0.3 volts and V2 = 0.3volts
Therefore V3 varies from 0.3volts to 0.6 volts

5.2.4. Current booster:

A current booster circuit is the one which amplifies the input current. The current booster circuit is designed using an emitter follower circuit constructed using a BJT (BC107). In the circuit the base terminal of the transistor serves as the input, emitter is the output, and the

collector is common to both power supply and the ground reference. Hence it is also called as common collector amplifier.

From Fig.5a, R_E is the emitter resistance, R_A and R_B are voltage divider bias resistances For emitter follower output voltage Vout ≈ Vin, since base-emitter voltage is very small, it is neglected ∴Vout ≈ V3

5.3 DESIGN AND TESTING :

5.3.1 BOUNDARY TEST CASES:

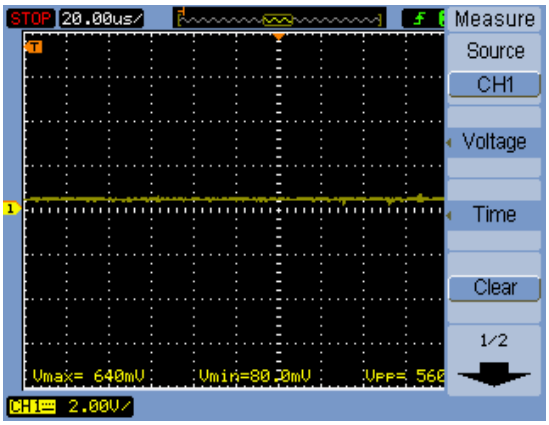

Fig.5c: final output when digital input is 00.

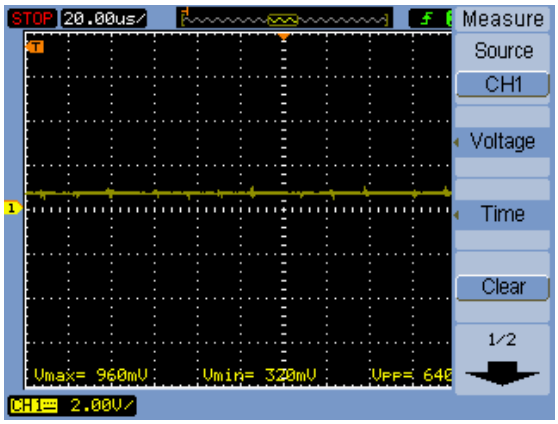

Fig.5d: final output when digital input is 11

1. Digital input to DAC is either 00 or 11 (with reference to Fig.5c and Fig.5d)

- TEST CASE ID : DAC_boundary_01
- ENVIRONMENT : driver circuit connected to GPIO of the UTLP in normal mode
- TEST SEQUENCE :
  - Run the application.
  - Consider the case where digital value either 00 or 11 is the input for DAC.
- OUTPUT TO BE RECORDED:
  - Record the output of the DAC and voltage divider networks
  - Record the output voltage level of the emitter follower circuit

- PASS CRITERIA :
  - Output voltage of driver circuit is 0.7volts corresponding to 00 or the output voltage is 0.925volts corresponding to 11.
  - Output voltage of driver circuit lying between 0.65volts to 0.75volts (with ±0.05volts error from reference 0.7volts) corresponding to 00 or output voltage lying between 0.875volts to 0.975volt (with ±0.05 volts error from reference 0.925volts) corresponding to 11

5.3.2 POSITIVE TEST CASES:

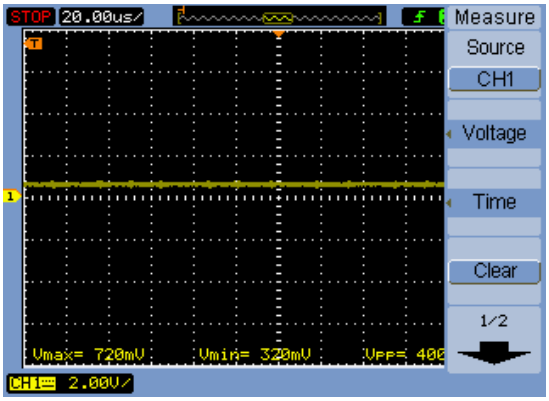

Fig.5e: final output when digital input is 01

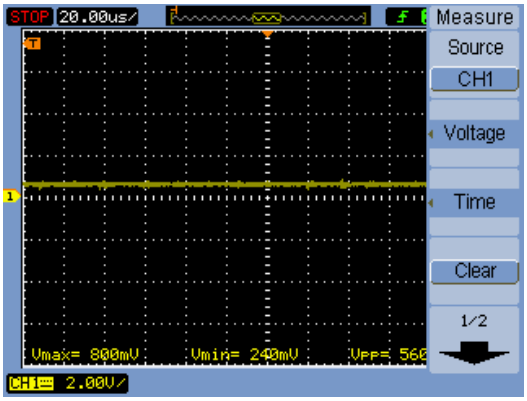

Fig.5f: final output when digital input is10

2. Digital input to DAC is either 01 or 10 (refer Fig.5e and Fig. 5f)

- TEST CASE ID : DAC_positive_01
- ENVIRONMENT : driver circuit connected to GPIO of the UTLP in normal mode
- TEST SEQUENCE :
  - Run the application.
  - Consider the case where digital value either 00 or 11 is the input for DAC.
- OUTPUT TO BE RECORDED:
  - Record the output of the DAC and voltage divider networks
  - Record the output voltage level of the emitter follower circuit
- PASS CRITERIA :
  - Output voltage of driver circuit is 0.775volts corresponding to 01 or the output voltage is 0.85volts corresponding to 10.
  - Output voltage of driver circuit lying between 0.725volts to 0.825volts (with $\pm$0.05volts error from reference 0.775volts) corresponding to 00 or output voltage lying between 0.8volts to 0.9volt (with $\pm$0.05 volts error from reference 0.85volts) corresponding to 11

5.3.3 NEGATIVE TEST CASES

3. Digital input to DAC is negative (less than 0 level)

- TEST CASE ID : DAC_negative_01
- ENVIRONMENT : driver circuit connected to GPIO of the UTLP in normal mode
- TEST SEQUENCE :
  - Run the application.
  - Consider the case where digital value applied is negative.
- OUTPUT TO BE RECORDED:
  - Record the output of the DAC and voltage divider networks
  - Record the output voltage level of the emitter follower circuit

- PASS CRITERIA :
  - Output voltage of emitter follower ranges from 0.4volts to 0.7volts which is not the actual voltage range for working of VGA.

Hence few test conditions are mentioned as above.

6. CODE FOR VGA

6.1. INTRODUCTION:

The module is coding as per the specification of Horizontal Synchronization (HSYNC) signal, Vertical Synchronization (VSYNC) signal and transfer of RGB (Red, Green and Blue) colour to get the output from General Purpose Input output (GPIO) port. The software used is Eclipse Galileo in the Ubuntu platform. The General Purpose Input Output port are used as output ports because the HSYNC, VSYNC and all the colours are output from the Unified Learning Kit to the R-2R Digital to Analog Converter (R-2R DAC).

When the Video Graphics Array (VGA) is idle, i.e. when HSYNC and VSYNC signals are not applied, the screen on which output is displayed will be black in colour. For this RGB colours are initialized as '1' which represents black. When HSYNC and VSYNC pulses are applied the display colour will change according to the output given from the GPIO port.

The monitor screen for standard VGA contains 640 columns by 480 rows of pixels. The VSYNC signal is generated using the delay function. Since 480 rows has to be scanned in one period of VSYNC signal, the HSYNC signal is generated within one period of VSYNC signal using 'for' loop. When the HSYNC signal goes positive RGB colour is sent to the GPIO port as output.

The HSYNC signal and VSYNC signal are synchronized as the generation of HSYNC signal depends on the VSYNC signal. The output are expected as per the specification as the appropriate delay is given. The code is such that the entire screen will be changed to a single colour when HSYNC and VSYNC signals are given. RGB which is of two bit each can take 64 combinations. Hence, 64 colours are possible.

The VGA port of Unified Learning Kit cannot be directly programmed because the port address is not known. This drawback made us use the GPIO pins to generate HSYNC, VSYNC signals and RGB colours. The digital output generated from the GPIO pins are then given to R-2R Digital to Analog Converter to get the analog signals which can then be given to the VGA port if successful in generating the signals with correct timings.

6.2. METHODOLOGY:

The specification for Horizontal Synchronization (HSYNC) signal, Vertical Synchronization (VSYNC) Signal, RGB (Red, Blue and Green) colours are given. Eclipse Galileo software in Ubuntu platform is chosen for coding.

6.2.1. Ubuntu Platform:

Ubuntu is based on Linux Operating System. It is committed to principles of open-source software development.

6.2.2. Eclipse Galileo:

Eclipse is an Integrated Development Environment (IDE) and an extensible plug-in system. It is also a multi-language software development environment tool. Eclipse IDE is used for developing applications based on C language for Unified Technology Learning Platform (UTLP).

6.2.3. Control Panel:

Control Panel is an application on the Ubuntu development PC which provides communication between Ubuntu Host PC and the Unified Technology Learning Platform (UTLP) and also enables downloading and executing UTLP programs in normal mode. The communication takes place over Ethernet with Universal Datagram Protocol (UDP) as the transport between stub host on Ubuntu PC and the stub client on UTLP. The Stub Client responds to the various commands sent from the Stub Host on Ubuntu PC.

6.2.4. Unified Learning Kit:

Unified Learning Kit is based on Texas Instruments OMAP3530 application processor & Spartan-6 FPGA. The OMAP3530 processor supports interfaces such as Mobile DDR, Nand Flash, Audio in & out, TV out, Touch screen LCD, VGA out, Ethernet, Keypad, USB OTG, 2 SD cards & external interface connectors such as Control sensor header, I/O expansion connector, I2C Header for GPS, Bluetooth & Modem Connector, Simple Digital interface connector, IrDA Connector, Camera Connector & LCD connector.

6.2.5. Functions used in the Program:

6.2.5.1. Setting the direction of General Purpose Input Output pins:

The direction of General Purpose Input Output (GPIO) pins are set as output using the Application Program Interface (API)

uint8 ulk_proc_gpio_set_dir (uint8 pin, uint8 direction)

The pin number that has to be initialized along with direction is given as parameter in the function. Here the direction is ‘0’ because all the ports are configured as output ports.

6.2.5.2. Setting the colours:

When the HSYNC and VSYNC signals are not applied to the Video Graphics Array (VGA), i.e., when the VGA is idle black colour is chosen to be displayed on the screen. When the HSYNC and VSYNC pulses are applied the colour which is given as GPIO data out is seen

on the screen. 64 different colours are possible as each red, green and blue are two bit digital data. The API used for this is
uint8 ulk_proc_gpio_set_data_out (uint8 pin, uint8 dataout)
The data is sent to GPIO pin where RGB output is taken.

6.2.5.3. Generation of signals:

The HSYNC and VSYNC signals are generated using the delay function. The API used is
uint8 ulk_proc_delay (uint32 delay)
The delay can be in milliseconds or microseconds. For generation of signals the delay is used in microseconds.

The code is written in Eclipse IDE which runs on Ubuntu. C language is used for coding. The RJ45 Ethernet cable is connected to the processor side in the UTLP. Switch is kept in Normal Mode. The IP address of UTLP is set. This sets both the IPs' of UTLP and host PC. Now, the code is written in the IDE. UTLP is detected in the control panel and the binary file of the project is loaded in to the processor and then executed.

While writing the code simple 'for' loops are used for generation of HSYNC signal and for continuous VSYNC and HSYNC signals generation. General Purpose Input Output (GPIO) ports are initialized and set as output. The output digital values are for Red, Green and Blue colours along with Horizontal Synchronization signal and Vertical Synchronization synchronization are taken from GPIO pins to R-2R Digital to Analog Converter.

6.3. DESIGN AND TESTING:

To understand the requirement and to simplify the effort for coding algorithm is written and tested.

6.3.1. Algorithm:

1. Initialization of GPIO pins.
2. Setting the direction of GPIO pins as output.
3. Sending the data as '1' for screen to appear black.
4. Super loop for generation of HSYNC and VSYNC signals.
5. VSYNC is generated using delay function of microsecond.
    - VSYNC is made '0'for 64 microseconds.
    - Then '1' for 1020 microseconds.
6. 480 rows have to be scanned in one period of VSYNC signal. Hence HSYNC is generated and RGB colour is sent to the output port.
7. Back to main loop.

The code is kept simple by the use of 'for' loop. GPIO is initialized using the command '*ulk_proc_gpio_init (void)'*. The GPIO direction is set using '*ulk_proc_gpio_set_dir (uint8 pin, uint8 direction)'*. The colours are sent using the API '*ulk_proc_gpio_set_data_out (uint8 pin, uint8 dataout)'*. The signals are generated using the command '*ulk_proc_delay (uint32 delay)',* the delay is given in microseconds (USEC).

6.3.2. Testing:

- TEST ID : GPIO_Output.
- ENVIRONMENT : UTLP GPIO pins connected to Cathode Ray Oscilloscope.
- TEST SEQUENCE:
  - Run the Application using control panel.
  - Connections are made from GPIO pins to the CRO.
  - Check for the generation of VSYNC signal and HSYNC signals.
  - Check the time period of these signals and also if RGB signals are sent during the 'HIGH' period of HSYNC.
  - Check the RGB output.
  - Check the period of HSYNC signal both 'HIGH' period (Fig.6.a) and 'LOW' period (Fig6.b).
  - Also check for the period of VSYNC signal (Fig.6.c).

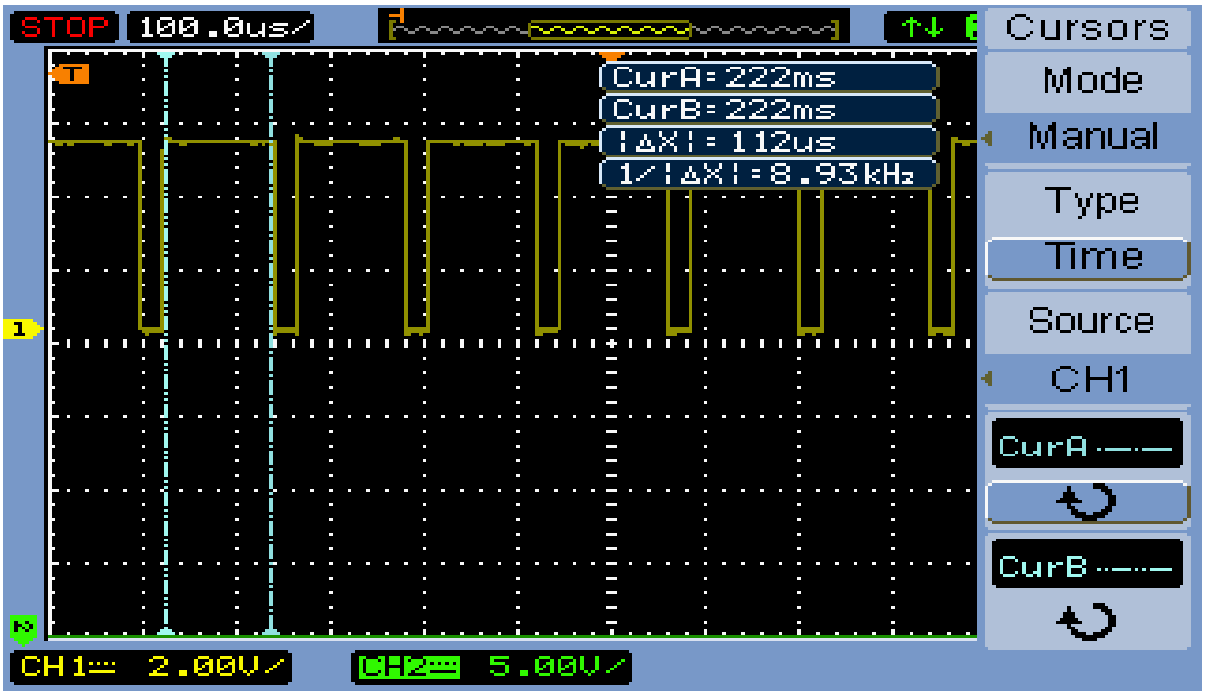

Fig.6.a: HSYNC signal high period output

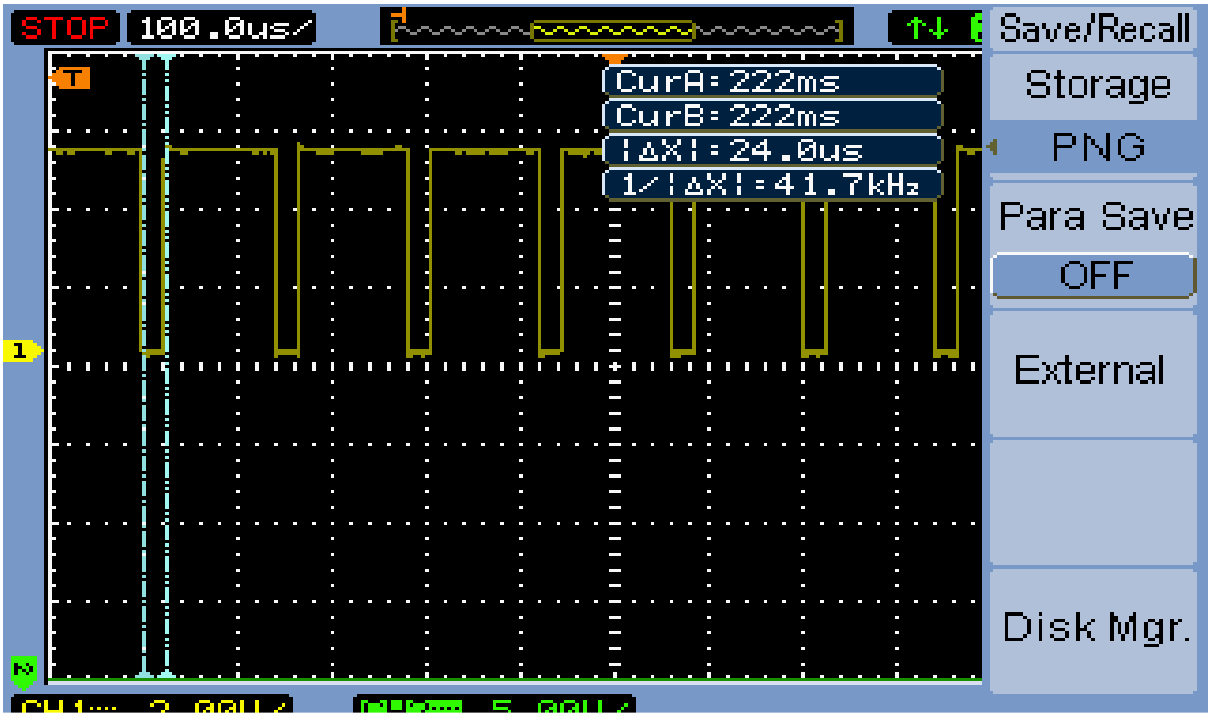

Fig.6.b: HSYNC signal low period output.

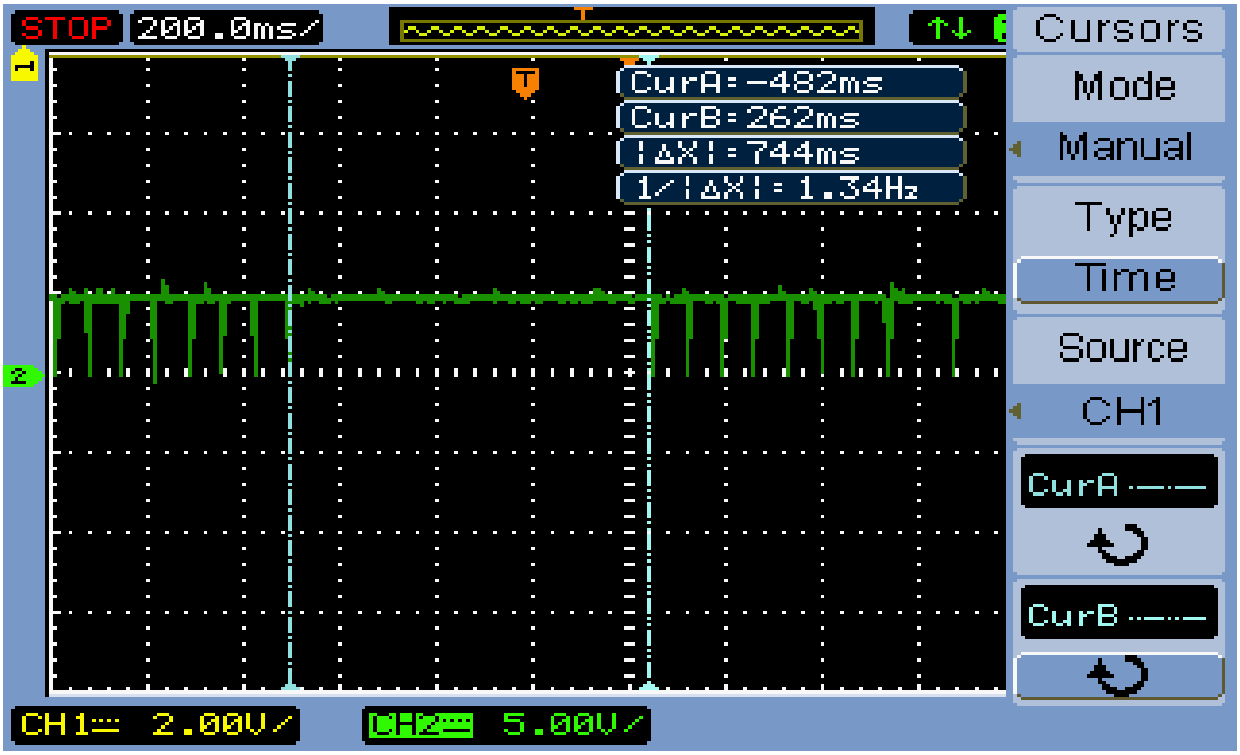

Fig.6.c: VSYNC signal output.

7. INTEGRATION AND TESTING

7.1 INTEGRATION:

Once all the previously described modules are ready, we have to do final integration and testing. It is a very crucial stage in any project as many of the times each individual module works, but the integrated project does not work.

We shall follow the integration in steps mentioned below.

- First the VGA is perfectly understood and hence a compatible VGA wire with a display unit is made ready.
- Secondly, GPIO set up is established using the ULK programming panel and necessary connections like connection of HSYNC, VSYNC are made to the VGA port along with ground.
- Thirdly, the pins configured to give digital voltage for colours RED, GREEN, BLUE are taken and connected to the driver circuit for proper conditioning of this signals by converting into an analog voltage value limited to the range of 0.7V -1V with current boosted sufficiently.
- Next a code is written for the entire working process, then is dumped on to the OMAP processor and testing is done appropriately to show the expected output.

7.2 TESTING:

Once the integration is successful, various tests are performed on the system to see its behavior at various conditions. It is always expected that even in negative test condition the system is in reversible and controllable region of operation.

Follow steps may be performed in this testing process:

- Once the entire integration is done testing of the output to various cases is done and appropriate conclusions are drawn.
- If any of the test condition fails then testing of modules individually is done and that module is identified which is generating the adverse effect on the overall response of the integrated project.
- If in integrated project, no errors are found in final testing then the integrated project is considered as hassle free and final conclusions are drawn from the output.

8. CONCLUSION

The Application Program Interface (API) is built using Ecllipse Galileo in Ubuntu platform. The output is taken from the General Purpose Input Output (GPIO) pins, as the Video Graphics Array (VGA) port cannot be programmed directly. The graphics LCD on the Unified Learning Kit and the Video DAC uses the same pins from the OMAP 3530 processor. The port address for VGA port was to be known to code an API. But, the port address was not found in the Technical Reference Manual for OMAP 3530. Hence, GPIO ports were used to take the digital outputs.

GPIO ports were programmed to generate Horizontal Synchronization (HSYNC) signal and Vertical Synchronization (VSYNC) signal along with Red, Green and Blue (RGB) colours. The output from GPIO is taken to driver circuit. The analog outputs are checked on the CRO for verification.

The output from the driver circuit is given to VGA port of the monitor. The output observed on the monitor was flickering. Since, the refresh rate was according to the specification of the monitor used. This problem arises due to the delay which is occurred in execution of every statement in the code, which was not taken into account.

8.1 Future Enhancements:

The code can be re-designed by taking into consideration the delay caused in execution of every statement so that maximum accuracy is achieved. The port address of VGA port has to be known and this would help in building the API. Building an API and adding it to the library for further use, by other programmers would make programming of the VGA port a lot easier.