**Ov7670 Camera Capture Work Report**

**Contents**

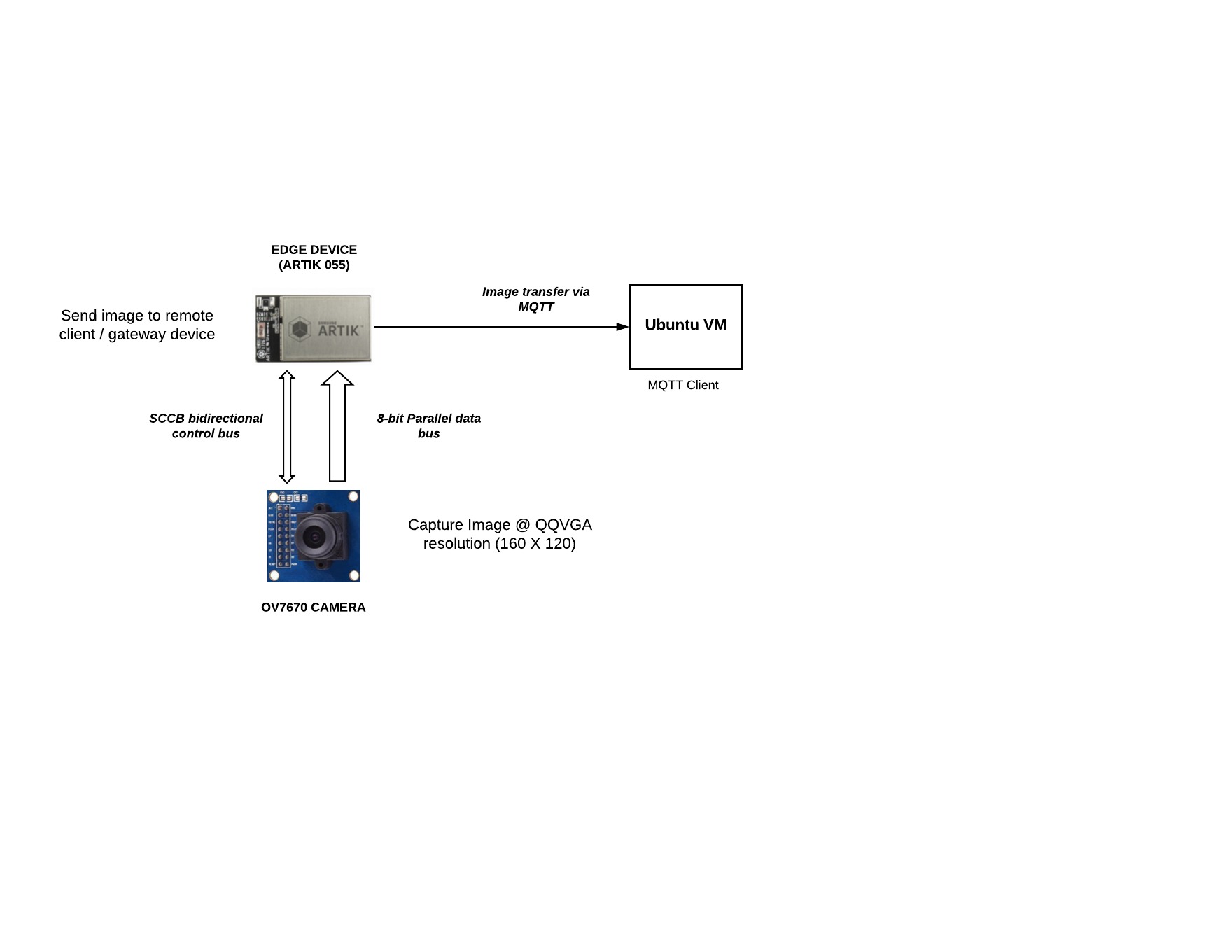
1. Abstract 1
2. Block Diagram 2
3. Pin Connections 3
4. Resolution and Color Formats 3
5. Register Configuration 4
6. Frame Synchronization 5
7. Camera capture 8
8. Code Description 8
9. TizenRT Tash Commands 9
10. Execution sequence 10
11. Testing and Debugging 12
12. Software Installation 12
13. References 13
14. Observations 13
15. **Abstract**

This report includes the details and description of the work done for camera capture project. It also includes changes and additions made in the existing code base provided for the ov7670 camera module and errors that resulted due to incorrect documentation of the module’s datasheet and implementation guide. These errors were fixed after debugging the hardware using the oscilloscope.

OV7670 is a low-cost camera module which can operate in multiple resolution formats - VGA, QVGA, QQVGA, QCIF, CIF, etc. It is capable of capturing images from less than 1fps to 30fps. It has an SCCB interface which works similar to the standard I2C protocol.

**The camera runs at 24 MHz input clock using an external crystal oscillator.**

The main objective was to capture the image at the edge device (ARTIK 055) at QQVGA resolution and send the image buffer to the gateway device for further processing.

1. **Block Diagram**
2. **Pin Connections**

The upper 7 bits of parallel data bus was connected to GPG0 port. The SCCB control bus was connected to GPA0 port.

The SCL and SDA lines are connected to I2C1 and operates at a frequency of 100 kHz.

|  |  |  |
| --- | --- | --- |
| CAMERA PIN | 055 PIN | BOARD PIN |
| HS | GPA0[2] | XEINT2 |
| VS | GPA0[1] | XEINT1 |
| PCLK | GPA0[0] | XEINT0 |
| D[7:1] | GPG0[7:1] | XGPIO[7:1] |
| RESET | - | 3.3V |
| GND | - | GND |
| 3.3V | - | 3.3V |
| SCL | - | XI2C1\_SCL |
| SDA | - | XI2C1\_SDA |

*Note: HS, VS, PCLK, SCL and SDA should be connected to the 055 board using shorter white jumper wires to reduce the effect of noise.*

1. **Resolution and Color Formats**

The camera is configured to capture image at QQVGA resolution in YUV422 color format.

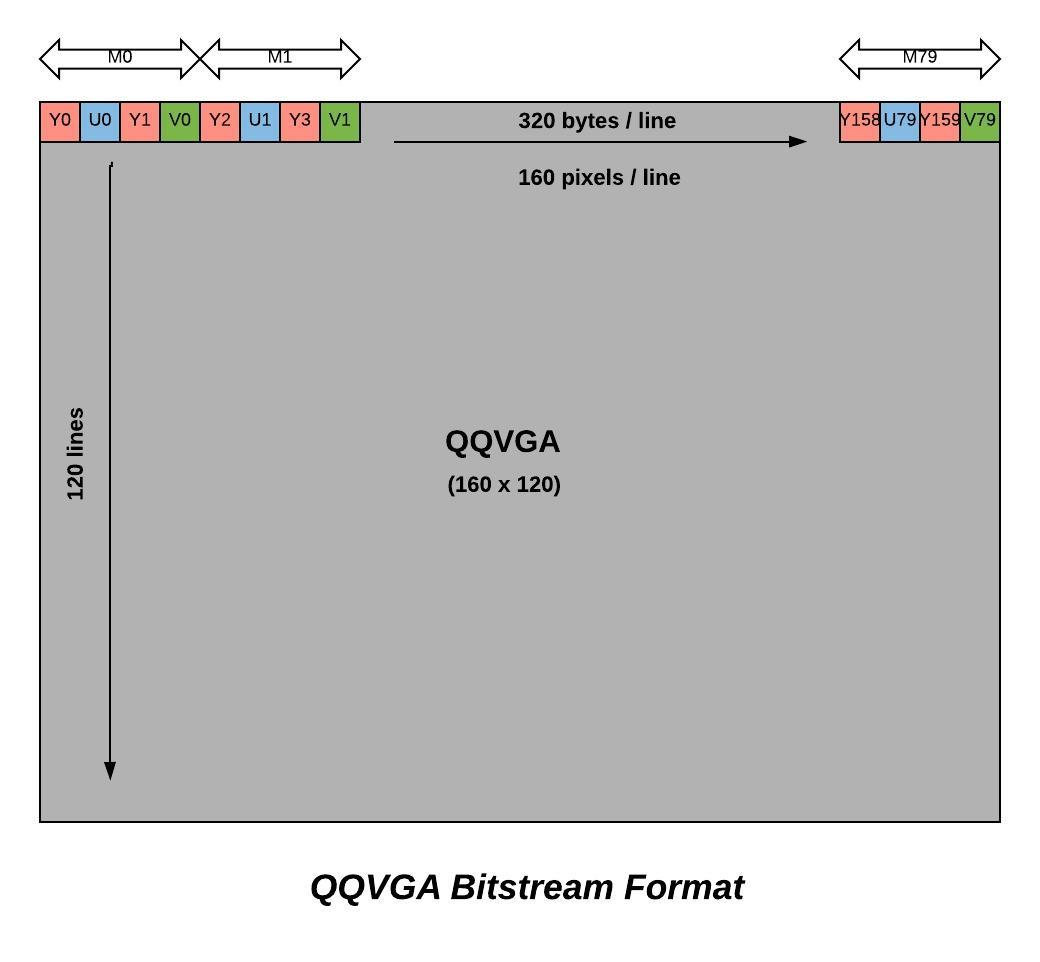
**QQVGA (160 X 120)**

QQVGA resolution corresponds to 160 pixels and 120 lines with aspect ratio of 4:3. So there are 160 pixels in each line. The size of the image is therefore 160 X 120.

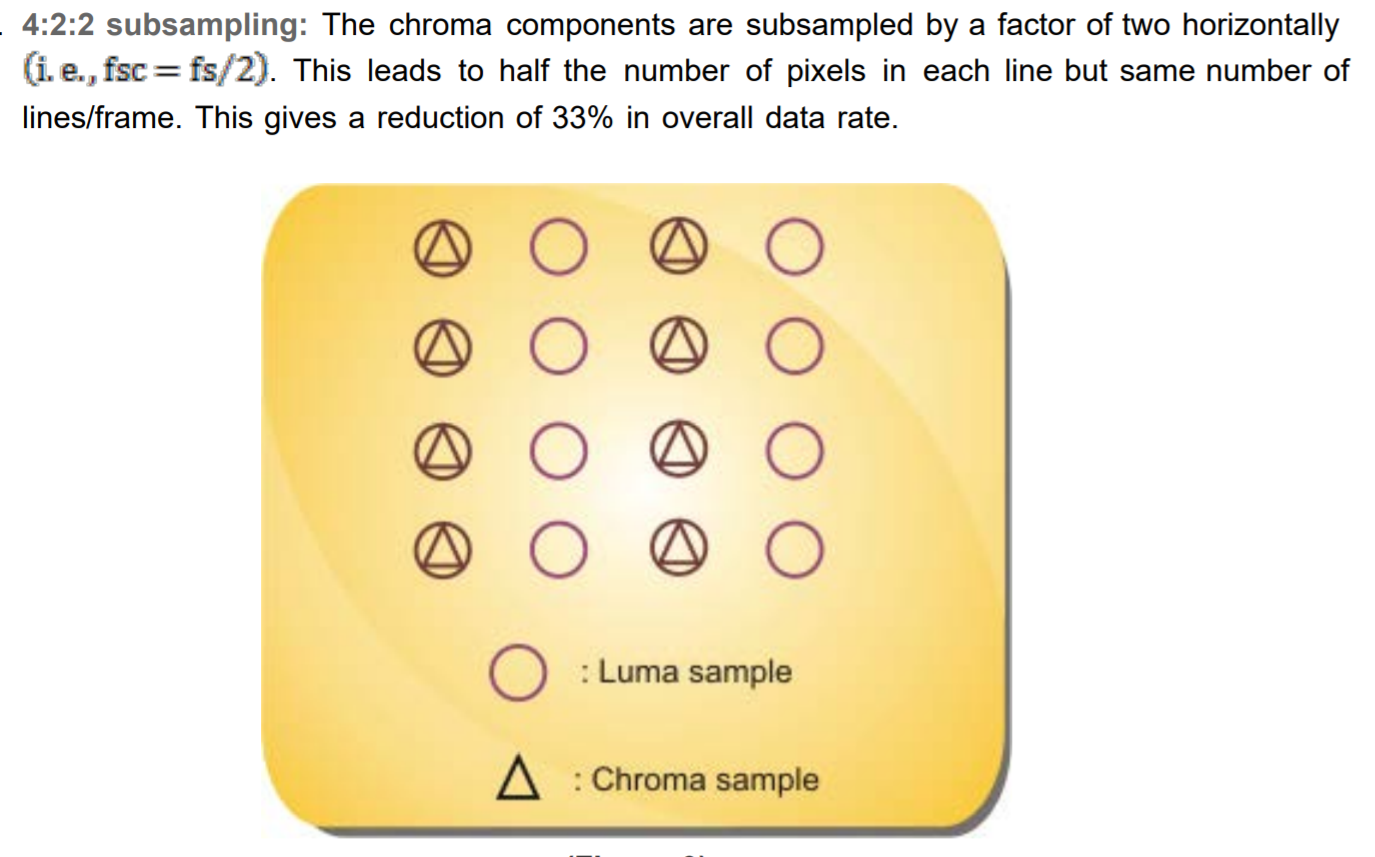
**YUV422**

*YUV is a color encoding system typically used as part of a*[*color image pipeline*](https://www.wikiwand.com/en/Color_image_pipeline)*. It encodes a*[*color image*](https://www.wikiwand.com/en/Color_image)*or video taking*[*human perception*](https://www.wikiwand.com/en/Human_perception)*into account, allowing reduced*[*bandwidth*](https://www.wikiwand.com/en/Bandwidth_(signal_processing))*for*[*chrominance*](https://www.wikiwand.com/en/Chrominance) *components, thereby typically enabling transmission errors or*[*compression artifacts*](https://www.wikiwand.com/en/Compression_artifacts)*to be more efficiently masked by the human perception than using a "direct" RGB-representation*

*Source -* [*https://www.wikiwand.com/en/YUV*](https://www.wikiwand.com/en/YUV)



The above figure shows the bitstream format for a single frame in QQVGA resolution with the **YUYV byte pattern**. The chroma sampling technique used here is **progressive interlaced YUV 4:2:2** (also known as YUV422). In YUV color space, Y is the luminance component, U and V are chrominance component.

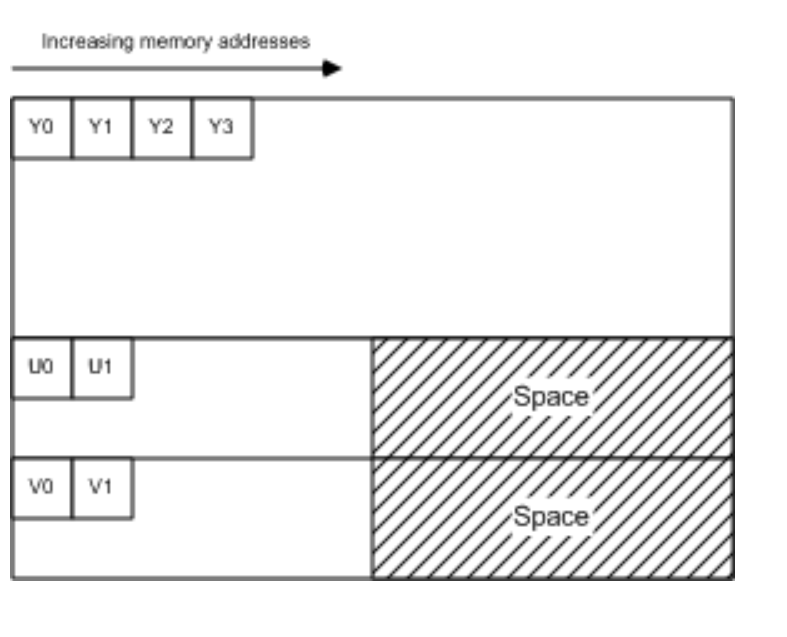


*Source -* [*https://nptel.ac.in/courses/117104020/5*](https://nptel.ac.in/courses/117104020/5)

Originally, the camera captures image in RGB color space but converts to YUV to reduce the bandwidth requirements during image / video transmission. After receiving the image buffer, it is again converted to RGB color space for display.

For this application, the YUV bitstream is generated in YUYV pattern which is consistent with the industry standard memory allocation format - **IMC3**. Although, the image buffer being published is just the raw YUYV bitstream and the Y, U and V channels are processed on the MQTT client side i.e., the gateway device (Ubuntu VM).

**IMC3 : image buffer memory allocation format**



*Image Source -* [*https://docs.microsoft.com/en-us/windows/desktop/medfound/recommended-8-bit-yuv-formats-for-video-rendering#422-formats-16-bits-per-pixel*](https://docs.microsoft.com/en-us/windows/desktop/medfound/recommended-8-bit-yuv-formats-for-video-rendering#422-formats-16-bits-per-pixel)

1. **Register Configuration**

The camera registers are configured by reading and writing hex values at their specified addresses.

Registers were primarily configured for –

1. ***Camera reset***  –

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Register | Bit | Address | Value | Description |
| COM7 | 7 | 0x12 | 0x80 | SCCB register reset |

Struct - ov7670\_reset[]

Type - t\_codec\_init\_script\_entry

1. ***Pixel clock rate / frequency –***

|  |  |  |  |
| --- | --- | --- | --- |
| Register | Address | Value | Description |
| CLKRC | 0x11 | 0x0c | Clock prescaler set to 12 for 233 kHz pixel clock frequency |

Struct - ov7670\_pclk\_rate[]

Type - t\_codec\_init\_script\_entry

1. ***Scaling and offset for Resolution –***

|  |  |  |  |
| --- | --- | --- | --- |
| Register | Address | Value | Description |
| COM3 | 0x0C | 0x04 | DCW and Scaling disabled |
| COM14 | 0x3E | 0x1A | Scaling PCLK enabled, PCLK divider set to 4 |
| SCALING\_PCLK\_DIV | 0x73 | 0xF2 | Clock divider by 4 for QQVGA |
| SCALING\_PCLK\_DLY | 0xA2 | 0x02 | Pixel Clock Delay |
| DBLV | 0x6B | 0xCA | PLL clock x8 |
| SCALING\_XSC | 0x70 | 0x3A | Horizontal scaling factor |
| SCALING\_YSC | 0x71 | 0x35 | Vertical scaling factor |
| SCALING\_DCWCTR | 0x72 | 0x22 | Horizontal and Vertical Downsampling by factor of 4 |
| HSTART | 0x17 | 0x16 |  |
| HSTOP | 0x18 | 0x04 |  |
| HREF | 0x32 | 0xa4 |  |
| VSTART | 0x19 | 0x02 |  |
| VSTOP | 0x1a | 0x7a |  |
| VREF | 0x03 | 0x0a |  |

Struct - ov7670\_QQVGA\_320[]

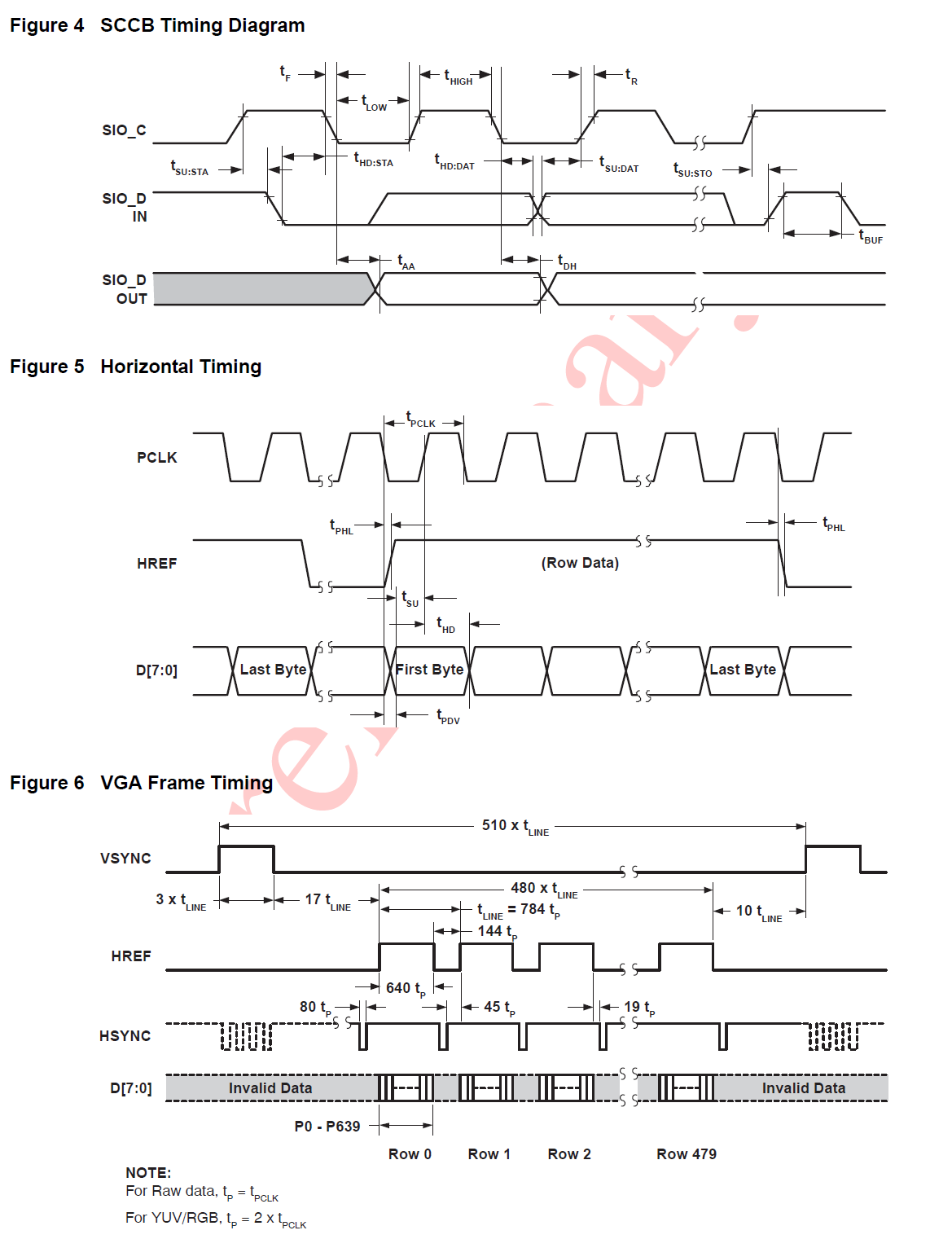
Type - t\_codec\_init\_script\_entry

1. ***Color, Contrast, Brightness, White Balance, Exposure, Gain and other features -***

Struct - ov7670\_color[]

Type - t\_codec\_init\_script\_entry

1. **Frame Synchronization**



1. **Camera capture**
2. **State Machine Implementation**
3. **Image Buffer Padding**
4. **Code Description –**

The ov7670\_test contains the following source files –

1. ***Ov7670\_test\_main.c –*** The main camera capture code exists in this file. Some major functions are –
2. *Ov7670\_state\_machine() :*

* Configures GPA0 pins for HS, VS and PCLK as GPIO input.
* Configures GPG0 pins for the parallel data bus as GPIO input.
* Captures a single frame as described in the state machine implementation
* Stores the number of pixels per line in a line buffer.
* Increments line count at the beginning of each line.
* Calls shift\_line\_counter() to split lines in case the falling edge of HS is not detected and data of 2 lines is stored in a single line.
* Calls pad\_yuv\_buffer() to stitch frame buffer data to the MQTT buffer.

1. Shift\_line\_counter():

* Split lines in case the falling edge of HS is not detected and data of 2 lines is stored in a single line.
* Increments line counter in case of line split.

1. Pad\_yuv\_buffer():

* Dupliactes line if pixels captured in the line are less than the twice the horizontal resolution. (320)

1. init\_ov7670\_communication():
2. atoi\_to\_hex():
3. ov7670\_tash\_modify\_reg():
4. ov7670\_tash\_write \_reg():
5. ov7670\_tash\_read \_reg()
6. ov7670\_modifyreg():
7. ov7670\_writeByte():
8. ov7670\_readByte():
9. **TizenRT Tash Commands**

In the TizenRT, ov7670\_test was added as an example using the kconfig settings. Since the previous configuration was giving incorrect resolution, the registers required to be configured from the tash command line to expedite the debugging. Therefore, functions for reading, writing and modifying the registers individually as well as in the new QQVGA configurations were added.

The following tash commands were added –

|  |  |
| --- | --- |
| Command | Description |
| ov7670\_test in | Initialize the I2C port and channel |
| ov7670\_test st | Run the camera in VGA configuration (new) |
| ov7670\_test pub\_init <ip\_addr> | Initiate an MQTT connection |
| ov7670\_test pub | Publish the MQTT packet with image buffer as payload |
| ov7670\_test r <reg> | Read register |
| ov7670\_test w <reg> <val> | Write a value to register |
| ov7670\_test m <reg> <val> | Modify the value of register |
|  |  |

1. **Execution sequence**
2. **Building and Flashing binaries on ARTIK 055s**
3. Connect the board to the power source via the USB cable. Make sure that the ARTIK 055s chip is inserted properly in the provided slot.
4. Open Ubuntu VM
5. Under “Devices” tab on the upper taskbar, select “USB” and then **select** “FTDI –RS232”.
6. Open wifi.h located in /home/<username>/TizenRT/apps/examples/ov7670\_test/
7. Change the ssid and password macros. For example –

#define SSID "ARTIK"

#define PSK "XXXXXXX"

1. Save the changes made in wifi,h file
2. Open Terminal and run the following commands –
3. Cd TizenRT/os
4. Make
5. Make download os
6. Exit
7. Under “Devices” tab on the upper taskbar, select “USB” and then **de-select** “FTDI –RS232”.
8. **Camera image capture using ARTIK 055s**
9. Connect the board to the power source via the USB cable. Make sure that the ARTIK 055s chip is inserted properly in the provided slot.
10. Open Device Manager and check the COM port allocated to the UART (There are two COM ports – JTAG and UART.).
11. Open PuTTy and select “Serial”. Enter the COM port and baud rate – 115200.
12. Press reset button on the ARTIK 055 board and check the terminal for initialization after reset.
13. Run the commands given below in the prescribed order to capture and publish the image buffer –
14. **Ov7670\_test in** – Initialize only once after reset
15. **Ov7670\_test m 0x9f 0xcf** – Adjust high reference luminance if required
16. **Ov7670\_test m 0xa0 0xaf** – Adjust low reference luminance if required. Should be less than high reference luminance
17. **Ov7670\_test st**
18. **Ov7670\_test pub\_init <ip\_address> -** Establish MQTT connection between publisher and broker.
19. **Ov7670\_test pub –** Publish message to the broker
20. **Exit**
21. Unplug the USB cable to power off the board.
22. **MQTT subscribe on gateway i.e., Ubuntu VM**
23. Open Terminal and run the following commands –
24. Cd paho.mqtt.c/build/output/test
25. ./mqtt\_test <ip\_address>
26. Wait for “mqtt\_recd\_frame.yuv written” to appear on the terminal to ensure that the image buffer is written successfully in mqttt\_rec\_frame.yuv file.
27. Press “q” to terminate the program
28. **Image Display on Ubuntu VM**
29. Open Terminal and run the following commands –
30. Cd opencv\_yuv422
31. ./yuv422\_display
32. Press “ctrl + c” once the image is displayed.
33. **Testing and debugging**
34. **Software Installation**
35. **MQTT paho client library intall –**
36. Open terminal and run the following commands –
37. git clone <https://github.com/eclipse/paho.mqtt.c.git>
38. sudo apt-get intall build-essential gcc make cmake-gui cmake-curses-gui
39. sudo apt-get install fakeroot fakeroot devscripts dh-make lsb-release
40. sudo apt-get install libssl-dev
41. sudo apt-get install dowxygen graphviz
42. cd paho.mqtt.c
43. make
44. make clean
45. make all
46. exit
47. **Opencv Install -** 
    1. Open terminal an d run the following commands –
       * 1. cd OpenCV
         2. bash install-opencv.sh
         3. exit
48. **Observations**
49. **References**