The Design Document for CSEE 4840 Embedded System Design

Invisibility-Curtain

Srivatsan Raveendran (sr3859)

Abhijeet Nayak (an3075)

Guide: Prof. Stephen A. Edwards

Spring 2022

Contents

1 Introduction

The invisibility cloak is a typical chromakey example of detecting a specific color and masking it in a video stream. This project aims to use the DE-1 SoC to perform edge image processing to execute the graphic effect of an invisibility cloak/curtain. The Invisibility cloak is such that if a piece of cloth of a specific hue is held before the camera, the regions of the cloak in the image disappear to reveal the original background behind it. This gives an invisibility effect to the person covered in the veil. The project aims to involve design of hardware system, interfaces, user mode programs and kernel mode drivers to obtain an end-to-end system that is capable of performing real-time camera image processing to achieve the task stated.

Our design aims to perform camera interface, image frame acquisition and its respective format conversion (A2D and YCbCr to RGB) in the FPGA implemented though Altera IP cores. The chroma key effect of replacing foreground color with the background frame is done in software in the HPS. HPS sends back the modified frame to the FPGA to display in the monitor.

The introduction briefly discusses some terminologies that are precursors to understanding the video acquisition pipeline. Later in the Systems Diagram section, the detailed discussion of design decisions will be presented.

1.1 YCrCb Color Space

The Luminance-Chrominance (YCrCb) color space contains information about the brightness (luminance or luma) and color (chrominance or chroma). The color is represented as two components, namely chrominance-red (Cr) and chrominance-blue (Cb). The Altera IP allows 8 bits for each of Y, Cr and Cb. There are two possible format modes of interest for our use-case:

1. YCrCb 4:4:4 -- This format is the normal YCrCb with all components as shown in Figure 1. This mode is defined as 8 bits per color and three color planes.

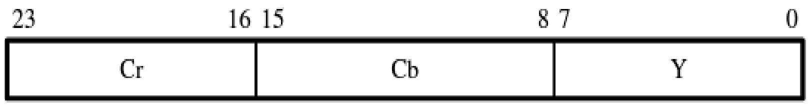


Fig.1 16-bit YCrCb 4:4:4 Color Space

2. YCrCb4:2:2—This format has half of the Cr and Cb entities onlyas shown in Figure 2.. Each consecutive pixel has alternating Cr or Cb components, with the first pixel in the frame starting with the Y and Cb pixel. This mode is defined as 8 bits per color and two color planes.

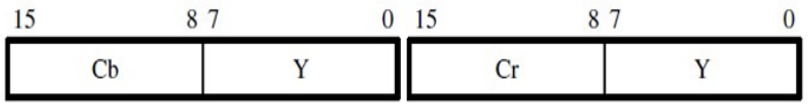


Fig.2 16-bit YCrCb 4:2:2 Color Space

The DE-1 SoC uses a 4:4:4 setting for YCbCr.

1.2 RGB Color Space

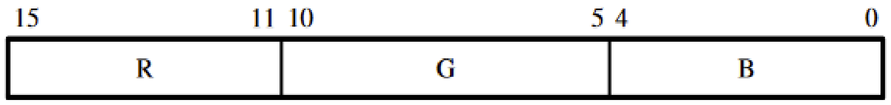


Fig.3 16-bit RGB color space

This format uses 5 bits for red, 6 bits for green, and 5 bits for blue as shown in Figure 3..

If R and B are 5-bit integers and G is a 6-bit integer then `color = B+(G<<5)+(R<<11);`

1.3 NTSC

The NTSC Standards and Video Capturing: NTSC refers to the National Television Standards Committee. Their standards for interfacing video cameras are as follows.

a) The clock frequency is 27MHz.

b) The cycle frequency is 60 Hz.

c) Video is sent interlaced. Implying that two frame cycles are needed to capture a full video display, where the first frame is all the odd horizontal lines and the second frame is all the even horizontal lines.

The graph below shows the I2C timing diagram (SDA- Serial Data; SCL – Serial Clock) for communication between the NTSC peripheral and the DE-1 FPGA's ADV7180 video chip.

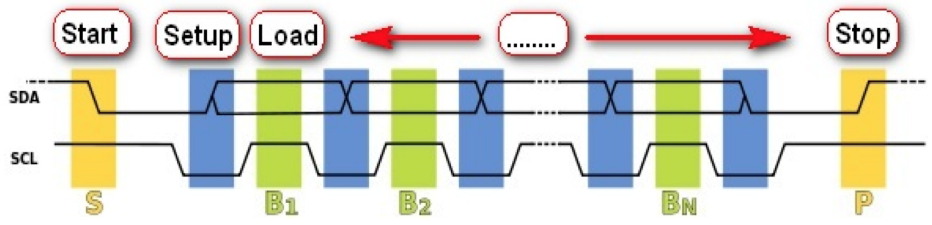


Fig 4. I2C timing diagram

From Page 2 of the datasheet for ADV 7180 we know that the SCL supports a maximum of 400KHz clock. We use this information to build a clock divider RTL that steps down the 27MHz clock into 40KHz. On the DE-1, the switch 0 must be pushed up to enable the 27MHz clock generation. This ensures that the TD_RESET Pin is asserted with logic level high that in-turn triggers the 27MHz XTAL .

Our next step is to create a place in memory to store the 8 bit command, 8 bit address and 8 bit data that we receive from the I2C camera peripheral. These regions are separate registers.

Now, the Video decoder must receive the I2C data and SCL. For this serial shifting of the clock and the data, we generate an I2C SCL and generate 8 bit values for command address and data. Using the I2C protocol we await an acknowledgement from the decoder and then stop sending this information to the decoder. The substitution technique is used to receive the frames from the decoder into the SRAM on one end and read the same frame from memory and display it to the monitor through the VGA Raster RTL.

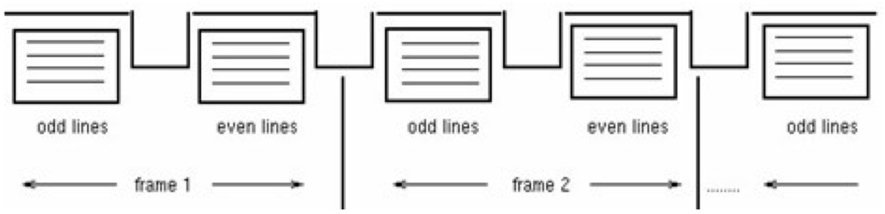


Fig 5. NTSC Interlacing Frame pattern

2 System Block Diagram

The hardware components of this system includes data acquisition from the NTSC camera, logic to display the image data stored in memory via a VGA raster RTL, RTL to synchronize the rates of image acquisition and image display and 2 port SRAM block to buffer the YCbCr image pixels.

The Video decoder receives the I2C data and SCL. For this serial shifting of the clock and the data, we generate an I2C SCL and generate 8 bit values for command address and data. Using the I2C protocol we await an acknowledgement from the decoder and then stop sending this information to the decoder.

Upon clicking a key (Key 0), the hardware starts streaming frames. The first frame that is buffered in memory gets read by the software in the ARM HPS. This is saved as a background image i.e., that which needs to be displayed over the colored curtain. As the clock moves forward, the camera input is updated with newer frames that rewrite the background frame in memory. These form the frames which we operate upon using the core chromakey logic in software. The C program converts, performs edge detection on the thresholded color of interest and generates a mask which is bitwise anded with the stored background frame. The data in the memory follows the substitution fashion of access. New video frame goes to the lower half of memory and moves to the upper half for VGA buffering.

We design our system in such a way that the software component of this system interacts with hardware through the Linux Kernel Mode Driver (KMD) stack. The driver can access the entire SRAM, read a background frame and subsequent frames, operate on them and finally write to the SRAM.

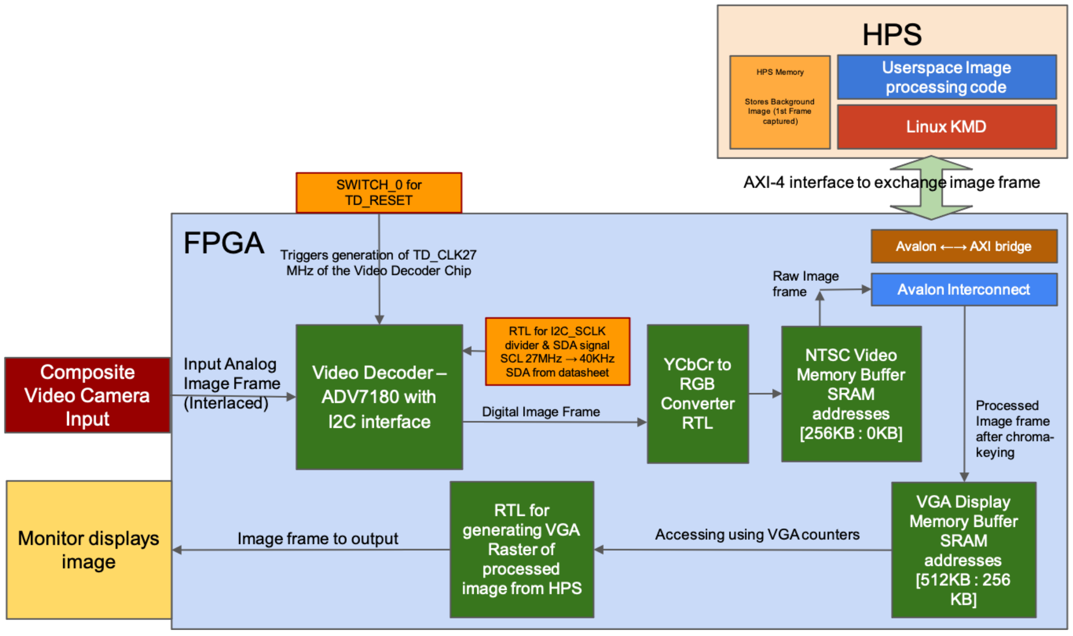

HPS
HPS Memory
Stores Background Image (1st Frame captured)
Userspace Image processing code
Linux KMD
AXI-4 interface to exchange image frame
SWITCH_0 for TD_RESET
FPGA
Triggers generation of TD_CLK27 MHz of the Video Decoder Chip
Avalon ←→ AXI bridge
Raw Image frame
Avalon Interconnect
RTL for I2C_SCLK divider & SDA signal SCL 27MHz → 40KHz SDA from datasheet
Composite Video Camera Input
Input Analog Image Frame (Interlaced)
Video Decoder – ADV7180 with I2C interface
Digital Image Frame
YCbCr to RGB Converter RTL
NTSC Video Memory Buffer SRAM addresses [256KB : 0KB]
Processed Image frame after chroma-keying
Monitor displays image
Image frame to output
RTL for generating VGA Raster of processed image from HPS
Accessing using VGA counters
VGA Display Memory Buffer SRAM addresses [512KB : 256 KB]

Fig 6. Systems Diagram of the Invisibility-Cloak System

3 Algorithms

In order to arrive at the correct steps to implement the system, we first simulate the system through a python program (refer Appendix).

The following are the stages of the simulated program:

1. General Algorithm Flow

    i) Acquiring RGB background image using OpenCV and storing it in an array

    ii) Flip the image frame

    iii) Conversion of RGB to HSV

    iv) Generating a mask that can be applied on the incoming frame

    a. Generate two different ranges of HSV values for the color of interest and apply the ranges to threshold the image. This leaves us with two masks.

    b. Perform addition of these two masks to incorporate the lower and upper bound of the range of HSV values

    c. Apply Morphological opening operation on the mask with a 3,3 filter for 2 iterations

    d. Apply Morphological image dilation operation on the mask using 3,3 filter for a single iteration.

    e. Bitwise invert the mask and call it mask_2

    f. Apply bitwise and on the masked background and the original background using mask_1

    g. Perform bitwise and of the incoming image with the masked version of the image frame using mask_2

    h. Combine the results from steps 'f' and 'g' using weighted addition

    i. Display the result of the addition as the operated final image

Morphological Image Opening

Opening refers to the morphological dilation of the erosion of a set with a structuring element. The effect of the operator is to preserve foreground regions that have a similar shape to this structuring element, or that can completely contain the structuring element, while eliminating all other regions of foreground pixels. It is used to preserve intensity patterns in the image. (Fig 6.)

The equation of opening is: ($A \circ B = (A \ominus B) \oplus B$)

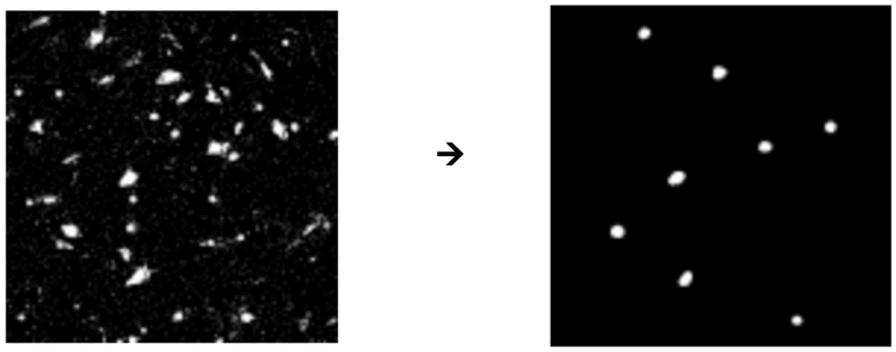


Fig 7. Illustrates the functioning of image opening on a thresholded image

Morphological Image Dilation

The basic effect of the operator on a binary image is to gradually enlarge the boundaries of regions of foreground pixels (i.e. white pixels, typically). Thus areas of foreground pixels grow in size while holes within those regions become smaller. For each background pixel (which we will call the input pixel) we superimpose the structuring element on top of the input image so that the origin of the structuring element coincides with the input pixel position. If at least one pixel in the structuring element coincides with a foreground pixel in the image underneath, then the input pixel is set to the foreground value. If all the corresponding pixels in the image are background, however, the input pixel is left at the background value.

Fig 8. below illustrates the working of image dilation on a sample binary image.

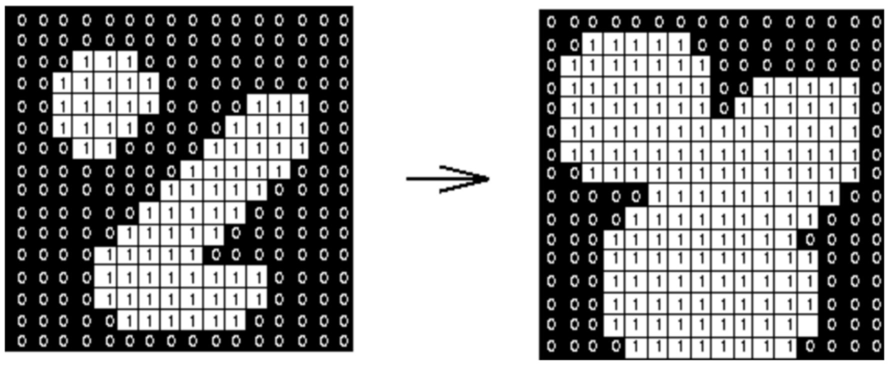


Fig 8. Dilation on sample image

2. Display Logic - VGA Raster algorithm

The following signals and logic must be generated to perform a VGA display scan.

1. vsync, hsync and blank signals.

2. First using the 50 Mhz system clock we will divide it to generate the needed 25 MHz monitor clock

3. Using the 25 MHz clock creates a counter and generates the needed sync pulses.

We follow the steps below to configure the RTL for VGA counters and pixel generation.

1. The memory configuration is 16 address lines and 32 data bits.
2. We were using a substitution method for video capture and display.
3. Lower 16 bits [0 - 15] were for odd video lines.
4. Upper 16 bits [16 - 31] were for even video lines.
5. Two frames were required to create a full video frame.
6. Vid_udl and Vid_udh are used to store all the even lines of video data.
7. Vid_ldl and Vid_ldh are used to store all the odd lines of video data.
8. There are a total of 210 vertical lines. Also note we started after 30 vertical lines and ended at 240 vertical lines (240-30= 210). So the full frames is 210 x 2 = 420
9. There were a total of 624 horizontal video pixels (8 bits per pixel). Again note we
started the counter at 150 pixels in and stopped at 774 pixels (774-150 = 624)

In step 4 of the above algorithm, steps a through h are performed in software and the result image is sent back to the hardware for display to VGA.

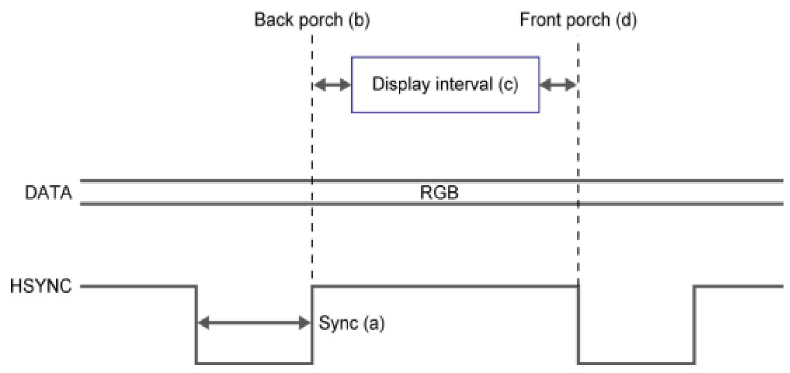


Fig 9. VGA Horizontal timing diagram

4 Resource Budgets

1. How much RAM is needed

a) Video Buffer

To compute the approximate memory requirement for storing a frame of data, we consider 640 pixels and there are 480 vertical rows. This means the frame size is 640X480 = 307200. In this case we would need ~ 307k x8 or 307K bytes. Where 8 is the 8 bits of colour data per pixel.

We consider a smaller frame resolution to allow budgeting memory for incoming video stream as well as the outgoing VGA display stream to be buffered. In a 624 x 480 resolution, since the camera NTSC standard acquisition happens in an interleaved fashion, we are required to acquire 2 frames of size **2 x (624x210) x 8 bits.**

1. At the edge of each 27 MHz clock pulse an **8 bit** video pixel is to be stored in SRAM.

2. Horizontal Line memory: Hsync corresponds to a line of video data. One line occupies **624 x 8** = **4992 bits**.

3. Vertical Line consideration: Vsync allows to capture a number of lines (rows) for each frame. For 210 rows per frame **4992 x 210 = 1,048,320 (~1 Mbit).**

4. Since we do this twice to account for the interleaving, a full frame demands **2 Mbits ~ 256 Kbytes**. This is within the memory capacity of the on chip memory SRAM on the DE1-SOC board.

To write a system Verilog SRAM module, we consider the configuration - **16 address lines x 32 bits of data** = 2^{16} x 32 = 2 Mbits of memory.

b) VGA Buffer
In order to optimize the memory requirement, the access pattern that we design is such that the decoded frame written to the from the camera gets accessed by the VGA controller RTL. The video format we use is the 4:2:2 format where there are 4 red bits, 2 green bits, and 2 blue bits summing to a total of 8 bits.

The common buffer for Display out and Video-in will be synchronized by a state machine rtl. The memory will be generated using Quartus 2 port RAM generating wizard. In order to display the frame, the 2MBit of frame data will be moved to the VGA Buffer which will form the lower half of addresses in the SRAM.

Thus the total of **256*2 = 512Kbytes** will be utilized.

2. Timing (latency)

Synchronization between the display and video feed in will introduce delays due to the different clock rates at which each stores information in the SRAM. Hence, the data available in the RAM is not always the latest data. Hence, it must be ensured that the VGA reads only when the RAM is fully populated with the video feed data. This hopes to tackle synchronization related latency overhead

5 The Hardware/Software Interface

The FPGA fabric and the ARM core are connected through two Advanced Microcontroller Bus Architecture (AMBA) Advanced eXtensible Interface (AXI) bridges. Although these two components can function entirely independently, communication between the ARM core and FPGA fabric can be a bottleneck for the overall system. That is why they are connected with two high-speed 128bit AMBA AXI bus bridges called HPS to FPGA and FPGA to HPS. The data path width for both bridges is not fixed to 128 bits; it can be configured via QSYS to 32, 64, and 128. By having this variable data width, the bridge can be tuned for maximum performance when communication between the FPGA fabric and the HPS L3 occurs.

The different types of interfaces in the DE-1 FPGA region are so following:

1. Composite Video Input – interfaces with NTSC / camcorder
2. VGA Video Output – connects to a VGA monitor for display out
3. PS2 Interface – Interfaces with Keyboards
4. Audio Interface – mic input, line input, speaker output
5. 40 pin general purpose ports – GPIO-0 (JP1), GPIO-1(JP2)

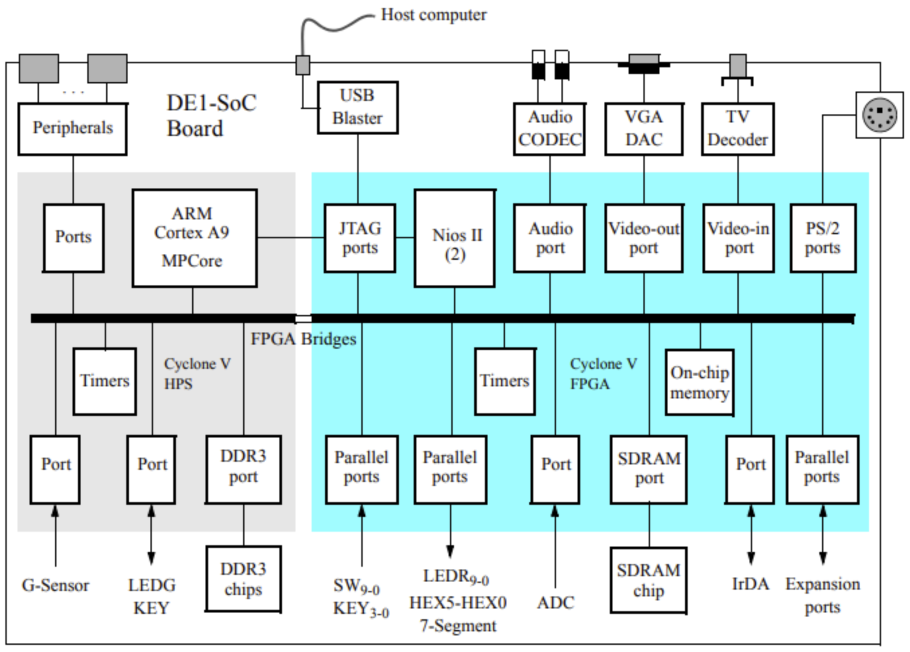


Fig 10. Block diagram of De1-SoC

Memory:
The DE1-SoC Computer has an SDRAM port, as well as two memory modules implemented using the on-chip memory inside the FPGA. These memories are described below.

SDRAM
An SDRAM Controller in the FPGA provides an interface to the 64 MB synchronous dynamic RAM (SDRAM) on the DE1-SoC board, which is organized as 32M x 16 bits. It is accessible by the A9 processor using word (32-bit), halfword (16-bit), or byte operations, and is mapped to the address space 0xC0000000 to 0xC3FFFFFF. Connections between the FPGA and SDRAM are shown below:

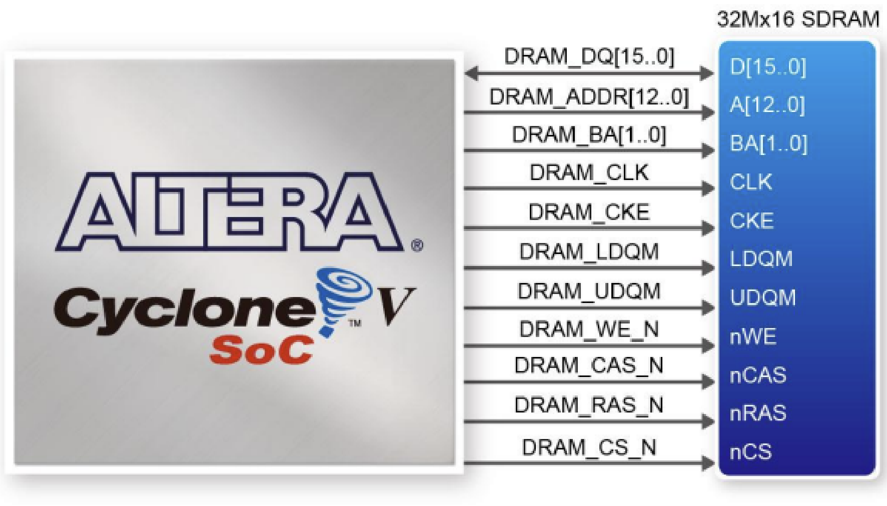


Fig 11: FPGA-SDRAM interface

On-Chip Memory
The DE1-SoC Computer includes a 256 KB memory that is implemented inside the FPGA. This memory is organized as 64K x 32 bits, and spans addresses in the range 0xC8000000 to 0xC803FFFF. The memory is used as a pixel buffer for the video-out and video-in ports.

Dual-clock FIFO:
The Dual-Clock FIFO buffers video data entering into the video decoder from the video source and help transfer a stream between two clock domains. Video streams into the core at the input clock frequency. The data is buffered in a FIFO memory. Then, the data is read out of the FIFO at the output clock frequency and streamed out of the core. The block diagram of the core is represented below:

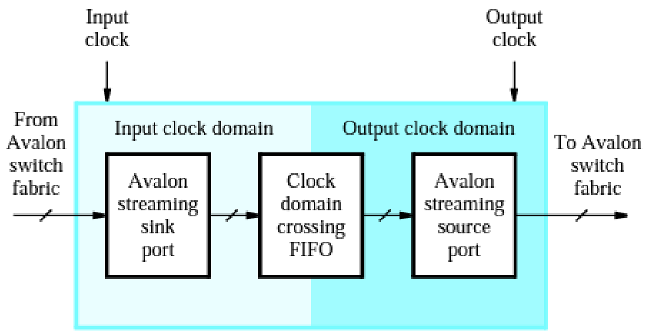

Input clock
Output clock
From Avalon switch fabric
Input clock domain
Output clock domain
To Avalon switch fabric
Avalon streaming sink port
Clock domain crossing FIFO
Avalon streaming source port

Fig 12. Dual-clock FIFO interface interface

DMA Controller:
The DMA Controller IP core stores and retrieves video frames to and from memory. The DMA controller has two modes of operation: "from stream to memory" and "from memory to stream". When in the "from stream to memory" mode, the core stores frames from an incoming stream to an external memory. The core uses its Avalon memory-mapped master interface to send the data to the memory. When in the "from memory to stream" mode, the DMAcontroller uses its Avalon memory-mapped master interface to read video frames from an external memory. Then, it sends those video frames out via its Avalon streaming interface. The block diagram of the core is represented below:

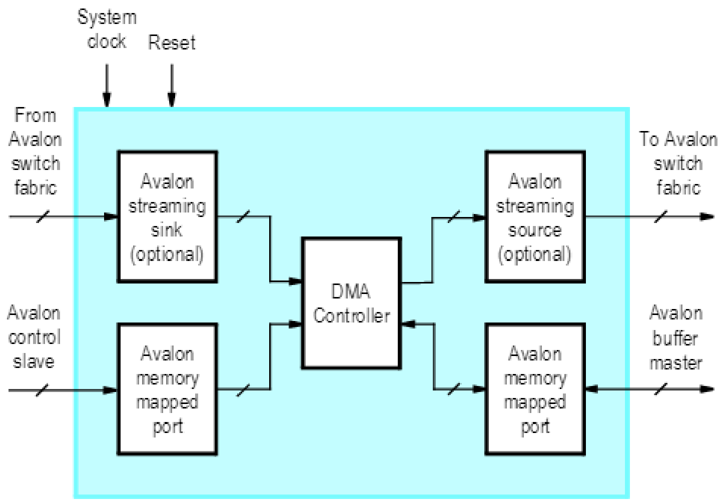

System clock
Reset
From Avalon switch fabric
Avalon streaming sink (optional)
DMA Controller
Avalon streaming source (optional)
To Avalon switch fabric
Avalon control slave
Avalon memory mapped port
Avalon memory mapped port
Avalon buffer master

Fig 13. DMA controller interface

DMA controller register Map:
The DMAcontroller's Avalon memory-mapped slave interface, named avalon_dma_control_slave, is used to communicate with the controller's 4 internal registers.

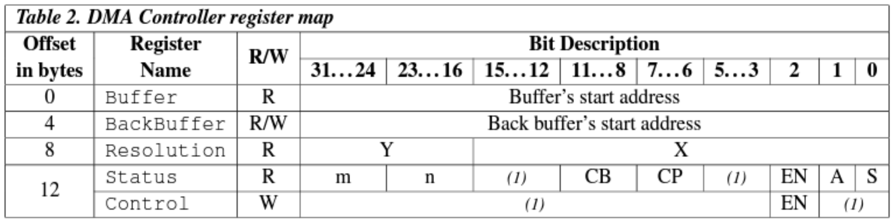


Table 2. DMA Controller register map

Offset in bytes	Register Name	R/W	Bit Description 31…24	23…16	15…12	11…8	7…6	5…3	2	1	0
0	`Buffer`	R	Buffer's start address								
4	`BackBuffer`	R/W	Back buffer's start address								
8	`Resolution`	R	Y		X						
12	`Status`	R	m	n	*(1)*	CB	CP	*(1)*	EN	A	S
	`Control`	W	*(1)*						EN	*(1)*	

Fig 14. DMA Controller Register Map

TheBuffer register holds the 32-bit address of the start of the memory buffer. This register is read-only, and shows the address of the first pixel of the frame currently being output. The BackBuffer register allows the start address of the frame to be changed under program control. To change the frame being displayed, the desired frame's start address is first written into the BackBuffer register. Then, a second write operation is performed on the Buffer register. The value of the data provided in this second write operation is not used by the controller. Instead, it interprets a write to the Buffer register as a request to swap the contents of the Buffer and BackBuffer registers. The swap does not occur immediately. Instead, the swap is done after the DMA controller reaches the last pixel associated with the frame currently being output. While the controller is not yet finished outputting the current frame, bit S of the Status register will be set to 1. After the current screen is finished, the swap is performed and bit S is set to 0. The Resolution register provides the X resolution of the screen in bits 15-0, and the Y resolution in bits 31-16. Finally, the Status/Control register provides information of the DMA controller.

Avalon bus:
Avalon-MM interface that supports read and write transfers with agent-controlled wait requests. The agent can stall the interconnect for as many cycles as required by asserting the waitrequest signal. If an agent uses waitrequest for either read or write transfers, the agent must use waitrequest for both. An agent typically receives address, byteenable, read or write, and writedata after the rising edge of the clock. An agent asserts waitrequest before the rising clock edge to hold off transfers. When the agent asserts waitrequest, the transfer is delayed. While waitrequest is asserted, the address and other control signals are held constant. Transfers complete on the rising edge of the first clk after the agent interface deasserts waitrequest. There is no limit on how long an agent interface can stall. The following figure shows read and write transfers using waitrequest.

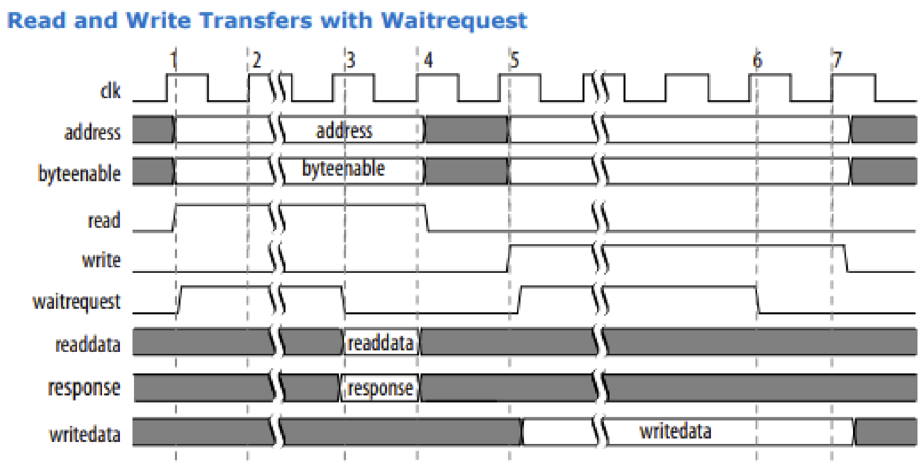


Fig 15. Avalon Bus timing diagram

The numbers in this timing diagram mark the following transitions:
1. address, byteenable, and read are asserted after the rising edge of clk. The agent asserts waitrequest, stalling the transfer.
2. waitrequest is sampled. Because waitrequest is asserted, the cycle becomes a wait-state. address, read, write, and byteenable remain constant.
3. The agent deasserts waitrequest after the rising edge of clk. The agent asserts readdata and response.
4. The host samples readdata, response and deasserted waitrequest completing the transfer.
5. address, writedata, byteenable, and write signals are asserted after the rising edge of clk. The agent asserts waitrequest stalling the transfer.
6. The agent deasserts waitrequest after the rising edge of clk.
7. The agent captures write data ending the transfer.

VGA Port:

The DE1-SoC board has a 15-pin D-SUB connector populated for VGA output. The VGA synchronization signals are generated directly from the Cyclone V SoC FPGA, and the Analog Devices ADV7123 triple 10-bit high-speed video DAC (only the higher 8-bits are used) transforms signals from digital to analog to represent three fundamental colors (red, green, and blue). It can support up to SXGA standard (1280*1024) with signals transmitted at 100MHz.

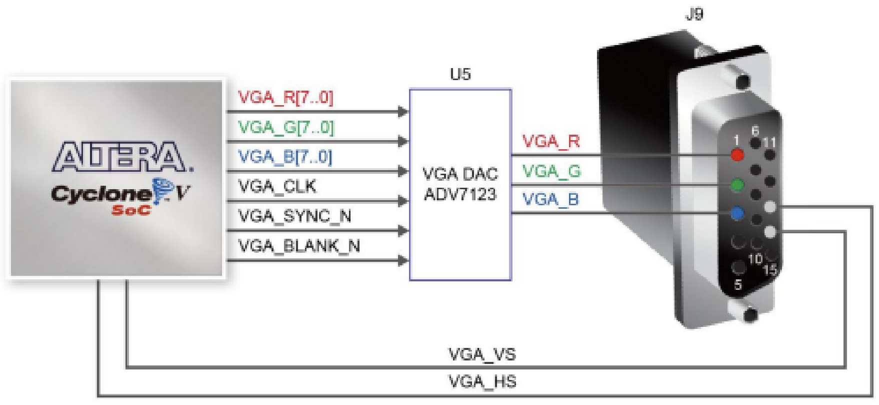


Fig 16. VGA Port Interface

Video-in decoder:

The chip used on the DE1-SoC board is an Analog Devices ADV7180. The ADV7180 is an integrated video decoder which automatically detects and converts a standard analog baseband television signals (NTSC, PAL, and SECAM) into 4:2:2 component video data, which is compatible with the 8-bit ITU-R BT.656 interface standard. Video is collected from a composite video source, such as a camcorder, with a composite video RCA jack output. The VGA controller supports a screen resolution of 640 × 480. The registers in the TV decoder can be accessed and set through serial I2C bus by the Cyclone V SoC FPGA or HPS. The video-in controller interface is illustrated in figure below

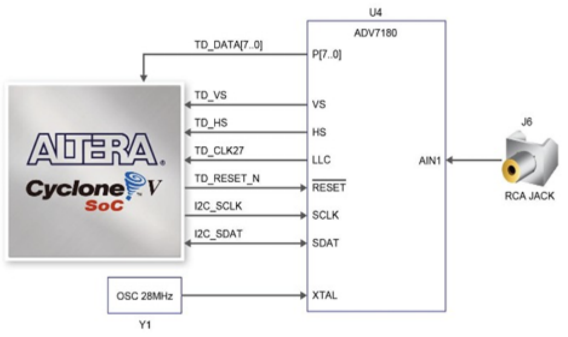


Fig 17. ADV7180 Interface

Signal Name	FPGA Pin No.	Description
TD_Data[0]	Pin_D2	TV Decoder Data[0]
TD_Data[1]	Pin_B1	TV Decoder Data[1]
TD_Data[2]	Pin_E2	TV Decoder Data[2]
TD_Data[3]	Pin_B2	TV Decoder Data[3]
TD_Data[4]	Pin_P1	TV Decoder Data[4]
TD_Data[5]	Pin_E1	TV Decoder Data[5]
TD_Data[6]	Pin_C2	TV Decoder Data[6]
TD_Data[7]	Pin_B3	TV Decoder Data[7]
TD_HS	Pin_A5	TV Decoder H_SYNC
TD_VS	Pin_A3	TV Decoder V_SYNC
TD_CLK27	Pin_H15	TV Decoder Clock Input
TD_RESET	Pin_F6	TV Decoder Reset
TD_SClk	Pin_J12	I2C Clock
TD_SDAT	Pin K12	I2C Data

Table 1- Pin assignments on DE1-SoC FPGA

[TV Decoder Data (7:0)]- 8 bits of video data are connected from the video chip to the FPGA. Pins are assigned according to Table 1.

[TV Decoder H_SYNC] -Horizontal sync pulse generated by the ADV7180 video decoder chip. Pin is assigned according to Table 1.

[TV Decoder V_SYNC]- Vertical sync pulse generated by the ADV7180 video decoder chip. Pin is assigned according to Table 1.

[TV Decoder Clock input]- This is a 27 MHZ clock generated by the ADV7180 video chip. In order to enable the clock, the TD_RESET pin must be asserted to an active high logic level.

I2C Data- This is a bi-directional serial data bus pin, use to program the internal serial register of the ADV7180 video decoder.

I2C Clock- This is the serial clock pin used to clock the serial data. The frequency that must be generated is typically below 400 KHz.

Video processing from Video-in decoder to VGA controller:

There are two major blocks in the system called I2C_AV_Config and TV_to_VGA. The TV_to_VGA block consists of the ITU-R 656 Decoder, SDRAM Frame Buffer, YUV422 to YUV444, YCbCr to RGB, and VGA Controller. The register values of the TV decoder are used to configure the TV decoder via the I2C_AV_Config block, which uses the I2C protocol to communicate with the TV decoder. The ITU-R 656 Decoder block extracts YcrCb 4:2:2 (YUV 4:2:2) video signals from the ITU-R 656 data stream sent from the TV decoder. It also generates a data valid control signal, which indicates the valid period of data output. De-interlacing needs to be performed on the data source because the video signal for the TV decoder is interlaced. The SDRAM Frame Buffer and a field selection multiplexer (MUX), which is controlled by the VGA Controller, are used to perform the de-interlacing operation. The VGA Controller also generates data request and odd/even selection signals to the SDRAM Frame Buffer and filed selection multiplexer (MUX). The YUV422 to YUV444 block converts the selected YcrCb 4:2:2 (YUV 4:2:2) video data to the YcbCr 4:4:4 (YUV 4:4:4) video data format. Finally, the YcrCb_to_RGB block converts the YcbCr data into RGB data output. The VGA Controller block generates standard VGA synchronous signals VGA_HS and VGA_VS to enable the display on a VGA monitor. The block diagram of the system is demonstrated below:

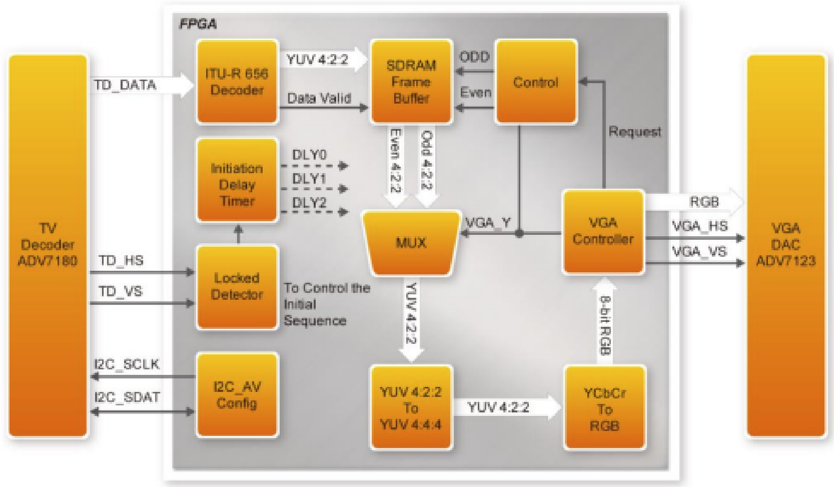


Fig 18. Block diagram for video processing

6 Milestones

Tentative Date	**Action Item**	**Comments**
03/05	Understanding IP components, VGA adapters, Avalon interconnects, HPS, algorithmic complexity	
03/12	Integrating hardware IPs, adapters, avalon interconnect etc. Interfacing Camera with DE-1.	Design documentation of successful IPs integrated
03/24	Writing Test Benches for bare metal hardware	
03/29	Writing device drivers for hardware integrated (i.e., for the video processing RTL and the)	deeper understanding of video interface and processing real time with FPGA
04/01	Mid Review	
04/15	Testing the integration with linux	
04/25	Writing user mode program to compile the kernel for the FPGA	
05/06	Testing System and debugging backgrounds	Drafting final report with debug results
05/16	Final Presentation	

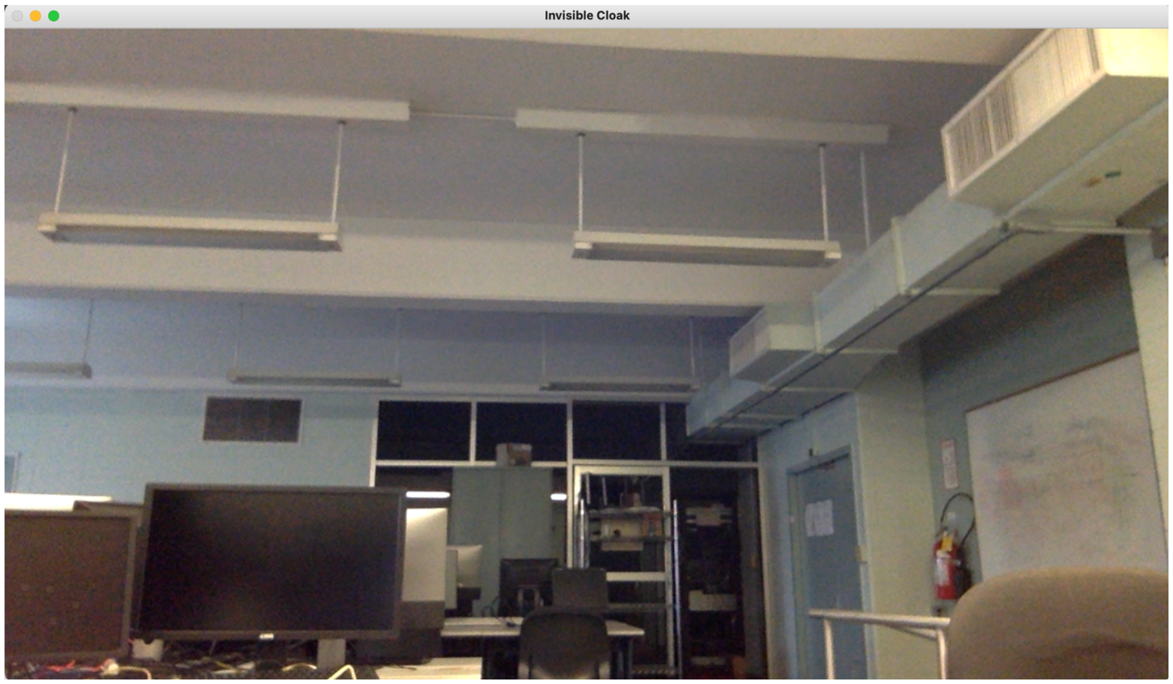
7 Appendix

Python program:

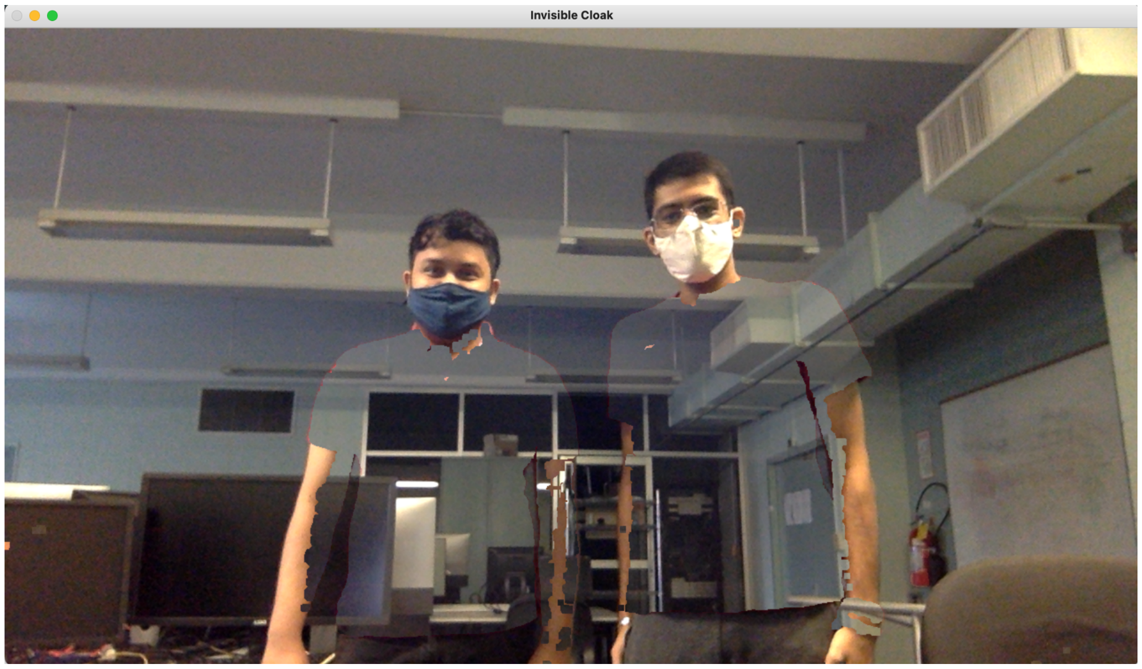
```
# Import Libraries

import numpy as np

import cv2

import time
# camCount=0
#To use webcam  enter 0 and to enter the video path in double quotes
cap = cv2.VideoCapture(0)

time.sleep(3)       # parenthesis has two because the camera needs time to
adjust it self i according to the environment

background = 0
# Capturing the background
# for i in range(60):
ret, background = cap.read()
#capturing image
background = np.flip(background,axis=1)
while(cap.isOpened()):  # Condition for this is when only the webcam is opened
it will only run the code else the code will not run in the background without
the webcam
    ret, img = cap.read() # FPGA
    if not ret:
        break
    # Software _BEGIN_
    img = np.flip(img,axis=1)
    hsv = cv2.cvtColor(img,cv2.COLOR_BGR2HSV) # FPGA - RGB
    #HSV values
    #setting the values for the cloak
    lower_red = np.array([0,120,70])
    upper_red = np.array([10,255,255])

    mask1 = cv2.inRange(hsv, lower_red,upper_red)

    lower_red = np.array([170,120,70])
    upper_red =  np.array([180,255,255])
    mask2 = cv2.inRange(hsv,lower_red,upper_red)
    mask1 = mask1 + mask2
    mask1 = cv2.morphologyEx(mask1,cv2.MORPH_OPEN,np.ones((3,3),np.uint8),
iterations = 2)
    mask1 = cv2.morphologyEx(mask1, cv2.MORPH_DILATE,np.ones((3,3),np.uint8),
iterations = 1)
    mask2 =cv2.bitwise_not(mask1)
    res1 = cv2.bitwise_and(background, background, mask=mask1)
    res2 = cv2.bitwise_and(img, img, mask=mask2)
     # Software _END_
    final_output = cv2.addWeighted(res1,1,res2,1,0)
    cv2.imshow('Invisible Cloak',final_output)
    k = cv2.waitKey(10)
    if k==27:
        break
cap.release()
Gcv2.destroyAllWindows()
```

Test Case:

Background: Image stream from camera capture - 3 channel (RGB) 720x480 matrix



Output: Processed image after applying the algorithm stated in section 3: RGB 3 channel 720x480 image matrix



References

1. Invisible Cloak using OpenCV | Python Project, https://www.geeksforgeeks.org/invisible-cloak-using-opencv-python-project/
2. DE-1 SoC Manual and Datasheet: https://www.intel.com/content/dam/altera-www/global/en_US/portal/dsn/42/doc-us-dsnbk-42-1004282204-de1-soc-user-manual.pdf
3. Video capture using DE1-SoC: https://hackaday.io/project/19945-video-capture-using-de1-soc-hps
4. ADV 7180 Video Decoder Datasheet - https://www.analog.com/media/en/technical-documentation/data-sheets/ADV7180.pdf
5. I2C Protocol Working - https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/
6. Display VGA Video Raster Scan Algorithm - http://www.cs.columbia.edu/~sedwards/classes/2022/4840-spring/video.pdf
7. Altera Memory System Design - https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/nios2/edh_ed51008.pdf
8. FPGA NTSC Video Feed and Processing - https://inst.eecs.berkeley.edu/~cs150/Documents/VideoNutshell.pdf
9. Altera Avalon Bus Specification sheet - https://www.ee.ryerson.ca/~courses/coe718/Data-Sheets/sopc/mnl_avalon_bus.pdf