EIE First Year Project – Final Report

A Virtual Fitting Room implementation with Field Programmable Gate Array (FPGA)

Duo Wang & Raymond Williams

Project outline

The aim of this project was to implement a system that could demonstrate image processing techniques and that would take advantage of the parallelisation offered by the Altera Cyclone II EP2C35F672C6 field-programmable gate array (FPGA). A low-resolution video camera was supplied in order provide a live video stream to process in effective real-time. For user interaction, we were encouraged to use the peripherals and connections present on the Altera DE2 board, upon which the FPGA was housed. These included a VGA output, PS/2 mouse connection, LEDs, switches, keys and seven-segment displays.

For our project, we decided to create a system to allow users to model different items of clothing without having to try them on physically. Almost all clothes shoppers have experienced the exhaustion and boredom caused by having to queue for access to a fitting room to model new clothes. Our project therefore aimed to eradicate this poor shopping experience by providing users with a way to sample items of clothing on a monitor using real-time image processing techniques.

We wanted to be able to offer users a wide range of options in terms of clothing, such as a range of colours, patterns and clothing items. In the end though, we found that a choice of clothing items, such as T-shirts, polo shirts, sweaters and hoodies would be too difficult to implement due to the hardware constraints of the FPGA, including latencies and the clock speed. Our final design was simplified slightly, allowing the user to change the colour or pattern of their T-shirt, as well as giving them the option to overlay an emblem such as a smiley face.

Synthesis process

Conventional development for FPGAs involves describing the structure and design of digital circuits using a hardware description language (HDL) such as Verilog or VHDL. For this project, however, we were strongly encouraged to use high-level synthesis (HLS), which involves using a high-level imperative programming language to describe the hardware.

Programming was undertaken using Catapult C software which permits development in ANSI/ISO C and C++ using Algorithmic C bit accurate data types. There was no need to specify any timing signals in the code as clocks, resets and enables were all added by the synthesis tool and designed to meet the constraints of the specific FPGA.

The compilation of high-level C code to a register transfer level (RTL) hardware description consists of several steps which are carried out in turn by the Catapult C synthesis tool. (Skogstrøm, 2012) Firstly, data dependencies in the code are analysed and a data flow graph is created. This is used to indicate which parts of the code can be executed in parallel, and which steps have to be executed sequentially. Based on the assembled data flow graph, each operation is then mapped onto a hardware resource in a process called resource allocation. Each resource is annotated with timing and area information which is used during the scheduling stage of synthesis. Scheduling involves taking into account the time taken to carry out the operations described in the data flow graph. Registers are added where necessary in order to fit operations around the target clock frequency. To control the scheduled design, a data path finite state machine (DPFSM) is created. (Fingeroff, 2010) A description of this hardware specific design is then written to the RTL file.

For our project, the high-level hardware descriptions were compiled to Verilog code, which was then imported into Altera's Quartus II software for interface with other HLS modules and connection to peripheral pins and controllers. Once the full system design was compiled by Quartus, it was then implemented in hardware through programming to the FPGA.

System architecture

To maximise the time spent developing the functionality of our system, we used hardware controllers provided to us as a basis for our design. These included a PS/2 mouse controller, a Y'CbCr video decoder, and a VGA controller. Our own customised IP blocks were also generated where necessary, such as to create a 7-segment display decoder and a mouse click 'enabler'.

The main part of our design is made up of two modules that were both developed using high-level synthesis. The first of these is used to provide the basic user-interface with a colour palette and mouse control. This user-interface module provides outputs to the image processing module, which provides colour replacement, pattern overlays and the tracked overlay of a smiley face.

User-interface module

Inputs to the user-interface module are RGB video data from the VGA controller, VGA coordinates and mouse coordinates. The video stream uses 30 bits to represent each pixel, with 10 bits for each of red, green and blue. Each 30-bit word is sliced into three parts which are assigned to separate variables. In a similar way, the video and mouse coordinate words are 20 bits long, and these are each sliced into two 10-bit values which provide the X and Y coordinates.

In order to test the imaging parts of the project without having to implement the system in hardware, a C++ test bench was provided. This allowed image processing to be carried out on 24-bit bitmap images which were converted into the 30-bit streaming format. Comparison between the test bench results and the hardware implementation of the system first demonstrated that the output of the test bench did not directly correspond with the area of the frame that was visible on the VGA monitor. Research into this problem showed that it was caused by overscan, a concept inherited from the days when monitors used cathode ray tubes and the horizontal lines used for image display had to extend beyond the visible parts of the screen. (Drawbaugh, 2010)

Whilst the SMPTE 259M-C standard used for NTSC transmission specifies an active image size of 720x486, we decided to crop each of our frames to a 4:3 aspect ratio and size of 640x480 to better

comply with VGA standards. To cater for the overscan regions of the frame, we defined X and Y offsets in a header file which were used as the origin for the 640x480 image.

Another issue we found when we compared the results of the test bench and the hardware implementation was that images in the test bench appeared upside down. This was caused by the fact that by default, the pixel array in a bitmap file starts from the bottom row of pixels and goes up to the top. (Microsoft Corporation, 2012) In the video streams however, the first row of pixels is the top row of the frame. This problem was later fixed with an updated version of the test bench.

In addition to cropping the input to a 640x480 frame, this module also provides a colour palette for the colour replacement function and mouse control via the PS/2 port. The statements to describe the processing of the input data are enclosed within a for loop which ensures that outputs are provided for each pixel of the frame.

The colour palette is displayed by defining regions of the frame where different colours should be shown. If the coordinates of the pixel match one of these regions, the colour required to make up the palette is output, otherwise the input is sent to the output. With a 3x10-bit RGB colour system, allowing values from 0 – 1023, the colours shown below in Table 1 can be selected from the palette.

Colour	Red Value	Green Value	Blue Value
White	1023	1023	1023
Blue	0	0	1023
Red	1023	0	0
Green	0	1023	0
Yellow	1023	1023	0
Magenta	1023	0	1023
Cyan	0	1023	1023

Table 1: Palette colours and their 3x10-bit red, green and blue components.

The positioning of the palette was set around a fixed position, defined in the header file as an offset from the origin of the frame. The size of each coloured square is also defined in the header file as 70x70 pixels.

Movement of the mouse pointer is based around the mouse control provided in the 'vga_mouse' sample project. In the sample project, the mouse pointer was simply a square, but we wanted to provide a better user experience with more relevant icons – a hand for the colour palette to indicate that the colours were clickable, and a pipette for the colour picker to indicate that colours present in the camera feed could also be selected (Figure 1).

Creating a ROM in Quartus was one option for displaying a custom image; however, it would be difficult to do this for images with more than two colours due to the on/off binary nature of the ROM. We also wanted to use HLS as much as possible due to it being one of the focuses of the project. One automatic memory optimisation carried out during high-level synthesis with Catapult C is the mapping of constant arrays to ROMs, and we decided to make use of this feature for the storage of the hand pointers. (Calypto, 2011, pp. 259-260)

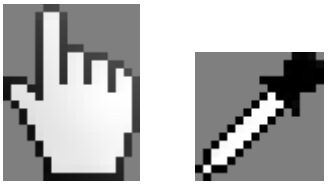

Figure 1: The hand and colour picker icons used as mouse pointers and stored in ROM.

Each pixel of the pointer image needed to have the same 30-bit format as the video frames, so each array element was required to store a 30-bit unsigned integer. Conversion from the bitmap file format to the 30-bit stream format was already being carried out for testing by the test bench using a library called 'bmp_io.h', so we decided to adapt the test bench file so that the stream data was written to a header file which could then be included in the HLS module. The commented C++ code below in Figure 2 shows how output to a header file was carried out. Since the test bench used a bitmap file as its input, colour words were only 8-bits long. The 'bitmap_stream' array was therefore created by shifting values by two bits to the left, resulting in scaling by a factor of $2^2 = 4$.

```
    // declare output file stream
    ofstream bmpstream;

    // output file location
    bmpstream.open("bmp_stream.h", ofstream::out);

    // terminate if output file cannot be opened
    if(!bmpstream.is_open()){
        cout << "Cannot open file.";
        exit(1);
    }

    // declare constant array at head of file
    bmpstream << "#include <ac_fixed.h>" << endl << "const ac_int<30,false>
imgsmile[10000] = {" << endl;

    // write RGB for each pixel as an array element
    int k;
    for(k=0; k<373; k++){
       bmpstream << bitmap_stream[k] << ", " << endl;
    }

    // curly bracket after final element
    bmpstream <<  bitmap_stream[373] << "};" << endl;

    // close header file
    bmpstream.close();
```

Figure 2: Code extract from modified test bench, used to create image header files; in this case, a header file for the hand pointer.

For transparency around the pointer, the surroundings were filled with the 3x8-bit grey colour of 0x7F7F7F, which translates to 533197308 in 3x10-bit decimal format (Figure 1). Display of the pointer is based around a central reference coordinate, with parameters defining the number of pixels to the left, right, top and bottom of this position. A calculation is carried out to map two-dimensional pixel coordinates in the frame to a one-dimensional element in the array, based around the reference coordinate of the image. If a pixel is within the pointer region and the pointer colour is not transparent, the pointer pixels are output in place of the standard video output.

A header file was created for each pointer, and each also required different positioning parameters. For the hand pointer, the reference pixel was at the centre of the image, whereas for the colour picker, the reference pixel was set as being at the bottom left corner so that colour samples are taken from the mouth of the pipette. A conditional statement based on pointer locations specifies that the hand pointer will appear for the colour palette, and the colour picker will appear for sampling colours elsewhere in the image.

With the viewable frame area of 640x480 being smaller than the size of the video input to the module, constraints were set to prevent the mouse pointer appearing beyond the visible region of the screen. This is aimed to help the user by preventing the eventuality of a pointer not being visible and therefore being impossible to locate.

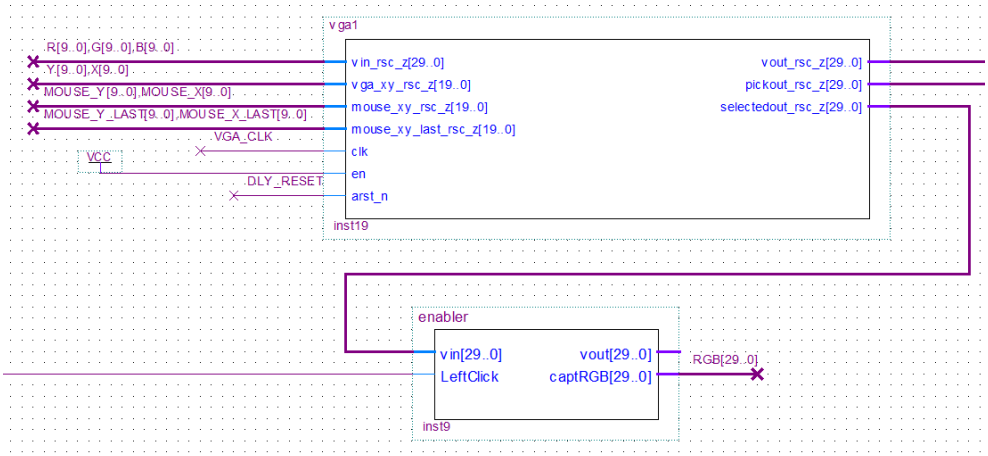

Figure 3: Quartus schematic representation of the user-interface module and mouse 'enabler', showing their inputs and outputs.

The user-interface module provides three video outputs (Figure 3):

1. ‘vout_rsc_z’ provides the full graphical user-interface with the colour palette and the mouse pointers.
2. ‘pickout_rsc_z’ is fed directly to the image processing module without mouse pointers.
3. ‘selectedout_rsc_z’ provides one pixel of colour at the mouse pointer location, which is sent to the mouse click ‘enabler’ for colour picking.

The ‘enabler’ (Figure 4) is a custom block that acts as an interface between the user-interface and image processing modules by providing the colour at the mouse pointer location when a mouse click is made. It works by taking the single pixel video feed from the user-interface module and storing the 30-bit value of this pixel in a register. The user can then left-click the mouse to send this stored colour to the image processing module.

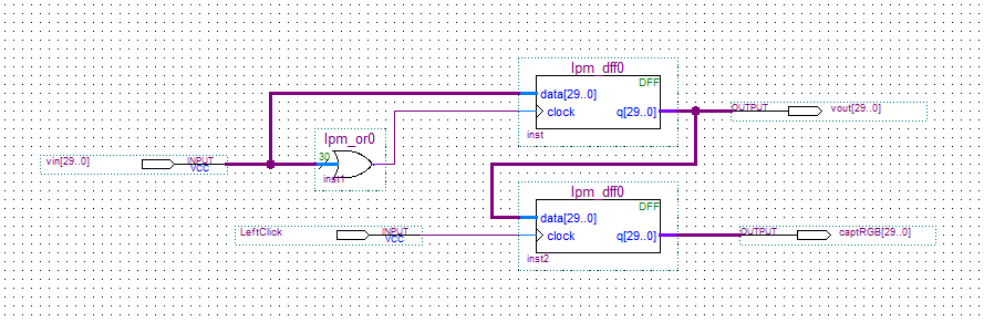

Figure 4: Detailed Quartus schematic view of the mouse 'enabler'.

The video output with only a single pixel of colour is connected to ‘vin[29..0]’. An OR gate is used to determine whether the input is zero. If the input is non-zero, the OR gate produces a pulse sent to D flip-flop ‘inst’ and the input is saved here. If the user left-clicks the mouse, another pulse is sent to the clock input of D flip-flop ‘inst2’ and the value currently saved in ‘inst’ will be sent to this flip-flop and output through the ‘captRGB[29..0]’ bus.

Image processing module

The image processing module provides four modes, namely:

1. To change the colour of the T-shirt
2. To apply a colour mesh to the T-shirt
3. To apply a rainbow effect to the T-shirt
4. To print a smiley face onto the T-shirt

The mode is selected using switches SW16-SW14. In Mode 1, all three switches are down. The switches for Modes 2, 3 and 4 are SW14, SW15 and SW16 respectively. Figure 5 show the schematic representation of the image-processing module with its inputs and output.

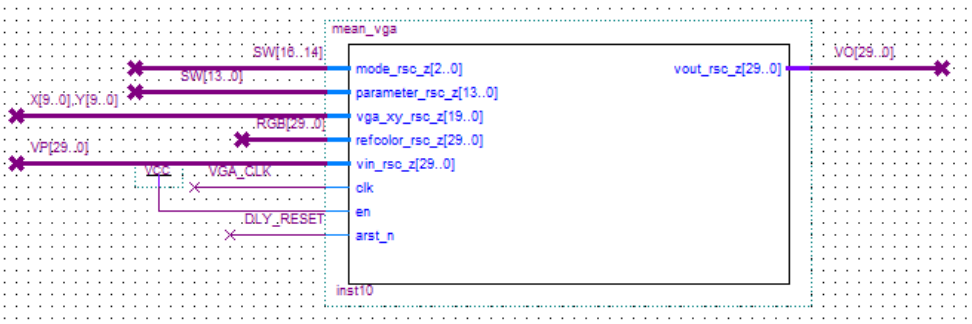

Figure 5: Quartus schematic representation of the image processing module.

Mode 1 – Colour change of the T-shirt

This mode takes the colour input from ‘RGB[29..0]’ as shown in Figure 5 and applies this colour to the T-shirt. The input is stored in ‘*refcolor’ and is then sliced into separate red, green and blue values which are stored in ‘refred’, ‘refgreen’ and ‘refblue’.

In order to retain the shading and texture of the original T-shirt, we applied a grey value offset to each pixel so that the grey values of the changed T-shirt were proportional to the grey values of the original T-shirt. We used ‘refgrey’ to calculate the reference grey value. This is calculated as the

average of 'refred', 'refgreen' and 'refblue'. Another variable, 'grey', is used to store the grey value of the original pixel.

Originally, we wanted to use the following set of equations to scale each pixel's grey value:

$$output\_red = \frac{refred \times grey}{refgrey}$$

$$output\_green = \frac{refgreen \times grey}{refgrey}$$

$$output\_blue = \frac{refblue \times grey}{refgrey}$$

Division by a variable takes three clock cycles, which is too slow for effective real-time image processing though so we therefore decided to use the following set of equations instead:

$$output\_red = refred + grey - refgrey$$

$$output\_green = refgreen + grey - refgrey$$

$$output\_blue = refblue + grey - refgrey$$

These statements efficiently change the colour of the T-shirt while retaining its original texture. Their effect can be seen in Figure 6, which shows the output image generated by the test bench.

Figure 6: a) Unprocessed input T-shirt image (left). b) Processed T-shirt image (right).

Mode 2 – Colour mesh application

This mode applies a 3x3 colour matrix to the T-shirt. The colours are stored in array 'colormesh[9]'. To create the mesh, the screen is divided into grids of a size defined as 'GRID_SIZE' in the header file. The numbering of the grids is shown in Figure 7 and this was calculated with the following line:

```
num=((vga_x/GRID_SIZE)%3)+3*((vga_y/GRID_SIZE)%3);
```

0	1	2	0	1	2
3	4	5	3	4	5
6	7	8	6	7	8
0	1	2	0	1	2
3	4	5	3	4	5
6	7	8	6	7	8

Figure 7: The colour mesh was applied as tiled 3x3 grids.

For each grid, the colour in 'colormesh[num]' is applied, where 'num' is the integer representing the grid value. The scaling of grey values is also used in this mode to keep the original texture and shading of the T-shirt. Figure 8 demonstrates the final test bench result of applying the colour mesh.

Figure 8: The result of applying the colour mesh filter to the test bench bitmap image.

Mode 3 – Rainbow effect application

In this mode a colour spectrum is applied vertically to the T-shirt to provide a 'rainbow effect'. The spectrum starts with red at the top and ends with violet at the bottom of the image. The result of the rainbow effect can be seen in Figure 9, although the spectrum here is inverted due to an older version of the test bench being used to generate the output image.

Figure 9: The output from the test bench when applying the rainbow effect.

The code used to generate the colour spectrum for the rainbow effect is shown below in Figure 10.

```
state=vga_y/DIV;

switch (state)
{
  case 0:{ refred=720; refgreen=vga_y*9; refblue=0; break;}
  case 1:{ refred=(2*DIV-vga_y)*9; refgreen=720; refblue=0; break;}
  case 2:{ refred=0; refgreen=720; refblue=(vga_y-2*DIV)*9; break;}
  case 3:{ refred=0; refgreen=(4*DIV-vga_y)*9; refblue=720; break;}
  case 4:{ refred=(vga_y-4*DIV)*9;refgreen=0;refblue=720; break;}
  case 5:{ refred=720; refgreen=0; refblue=(6*DIV-vga_y)*9; break;}
}
```

Figure 10: High-level synthesis code used to generate the colour spectrum.

'DIV' is a constant equal to 80, which is one sixth of the height of the video frame (480 pixels). The vertical coordinate of the pixel is given by 'vga_y'.

For each of the six horizontal divisions of the image frame (defined as the 'state' value), a set of equations determines the RGB values for each horizontal line in each division. In order for the computation to be fast enough for real-time processing by the FPGA, only linear functions are used. Figure 11 is a graph of red, green and blue values against 'vga_y', shown to illustrate the result of the equations.

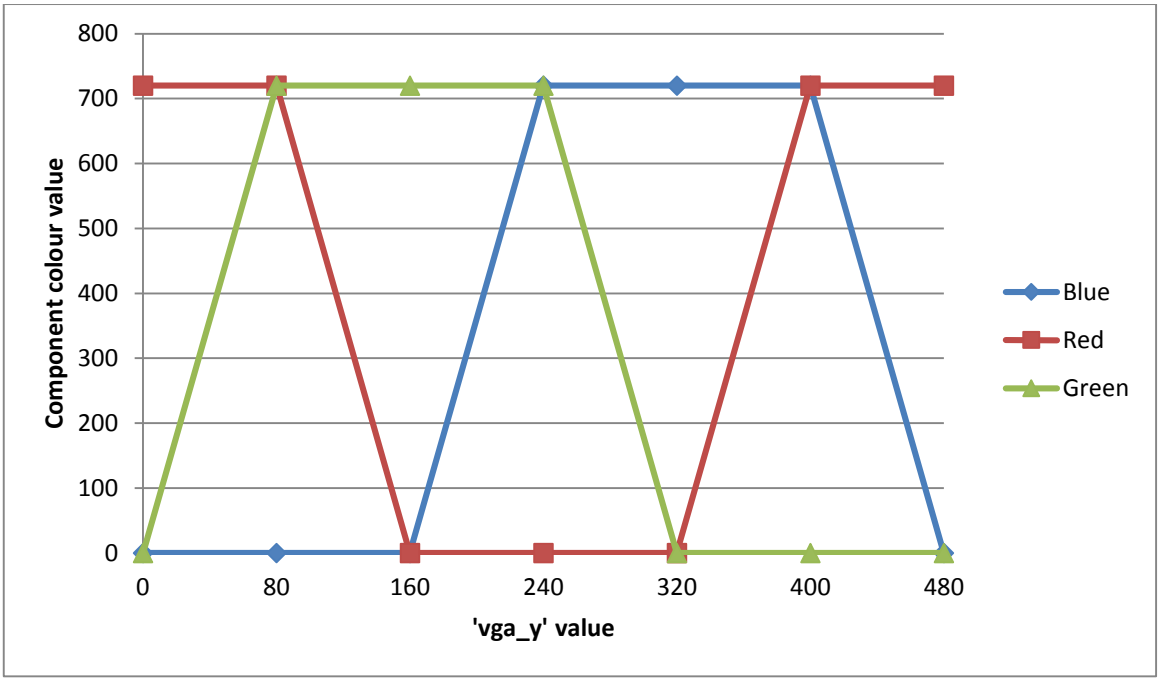

Figure 11: Graph of component colour value against vertical position.

Mode 4 – Printing of a smiley face onto the T-shirt

Selection of this mode results in the printing of a smiley face at the centre of the T-shirt. As with the mouse pointers used in the user-interface module, the smiley face (Figure 12) is read as a bitmap file using the 'bmp_io.h' library provided with the test bench and is then stored in a header file named 'smile_stream.h'. In order to correctly position the face, a red strip is stuck in the top left corner of the T-shirt. The centre of the face is considered as being 100 pixels below and 100 pixels to the right of the top left corner of the red strip. A 48-bit unsigned integer variable, 'x_strip' is used to record if the previous 48 pixels are red, as definied by a threshold value. This variable is considered as a binary array and for every pixel, the 'x_strip' value will be shifted left by 1 bit, leaving '1' at bit 0 if the current pixel is recognised as red or '0' otherwise. If all 48 bits of 'x_strip' are 1, then a red strip has been found and the strip-detection function is disabled until the start of next frame.

Figure 12: The smiley face bitmap image which was converted to the stream format for being overlayed onto the T-shirt.

After determining the centre of the 100x100 sized image (Figure 12), it is printed onto a 100x100 area at the centre of the T-shirt. The grey surroundings of the smiley face are made transparent when the overlaying onto the T-shirt takes place (Figure 13).

Figure 13: The output when this mode is selected is the printing of a smiley face onto the T-shirt.

Use of optimisations and parallelism

One of the key aims of this project was to make use of the parallelism that is possible with a FPGA. Without the use of any optimisations to enable parallelism, we soon found that our system, even at a primitive stage of design, was no longer able to keep up with the timings required for output to a VGA monitor.

Optimising loops is the primary way of improving design timings, but it comes with a trade-off in terms of area as each resource may need to be repeated multiple times (Mentor Graphics Corporation, 2011). The first step we took to optimising our loops was to enable loop pipelining. This allows the next iteration of a loop to start before the current iteration has finished, by creating duplicate hardware and running the two loops in parallel (Fingeroff, 2010). The amount of overlap between parallel loops is defined by the initiation interval which we set to one. This means that a new loop iteration is started every clock cycle.

Another option for parallelism is to enable loop unrolling. This dissolves the loop and replaces it with sequential statements in code which can be implemented in parallel using hardware. Loop unrolling is only possible though if loops are terminated and no dependencies exist between loop iterations. With our pipelined system already meeting timing constraints, we found no need to unroll loops. If our system was more complex however, loop unrolling would be important in ensuring that timing constraints were still met.

How to use the system

To operate the system, the user must be wearing a purple coloured T-shirt, as this is the colour that our thresholding is designed to recognise. For the overlay of the smiley face, the user must also have a red strip attached to the top left corner of their shirt.

System control makes use of the various inputs and outputs provided by the DE2 Board. The inputs used include switches SW0 to SW17, the composite camera feed and the PS/2 mouse input. The

outputs are the VGA connection to the monitor, the 7-segment displays showing a 255-bit representation of the RGB values and an LED that lights up when a mouse click is made.

The user can always switch between the original video stream directly from the camera and the processed video stream using SW17. When SW17 is pushed up, the original video stream is displayed and when it is down, the processed video stream is displayed.

A 7-colour palette is at the bottom of the screen. When SW17 is pushed up the user can use the pointer to select a colour from the palette by clicking on the desired colour. The user can also select a colour from anywhere else on the screen. When not selecting from the palette, the cursor icon will change from a hand to an eyedropper.

When SW17 is pushed down and SW14-16 are all down, the system is in T-shirt colouring mode. The T-shirt will turn into the colour selected by the mouse. The user can use SW13-SW0 to adjust the threshold for T-shirt detection, as described in Table 2Table 1.

Switches	Threshold adjustment
SW13 - SW10	A 4-bit number that sets how much larger the blue value should be than the red value for the pixel to be considered part of the T-shirt.
SW9 - SW6	A 4-bit number that sets how much larger the blue value should be than the green value for the pixel to be considered part of the T-shirt.
SW5 - SW0	A 6-bit number that offsets the grey threshold value. The larger the number, the higher the grey threshold value for colour detection.

Table 2: Key to describe how switches can be used to adjust the T-shirt detection threshold.

When SW14 is pushed up, the colour mesh mode is enabled and when SW15 is pushed up, the rainbow effect mode is selected. When either of these modes are enabled, the user can still use SW13 to SW0 to adjust the detection threshold.

When SW16 is pushed up, the smiley face mode is turned on. In this mode, SW13-SW0 can still be used to adjust the threshold, but SW13-SW10 now sets how much larger the red value should be than the green value and SW9-SW6 sets how much larger the red value should be than the blue value. These changes are needed because in this mode the system detects the red strip at the top left of the T-shirt instead of T-shirt itself.

Key 0, when pressed, initializes the system and Key 1 sets the mouse pointer to a random position on the screen.

Most challenging aspects

The two most challenging tasks we encountered in the project were the detection of the T-shirt colour and making the algorithm for colour detection and replacement simple and efficient enough to run on the limited resources provided by the FPGA.

When we were trying to detect and replace only the colour of the T-shirt, irrespective of the threshold value we set, there were always some background colours that were also picked up and changed. Furthermore, we found that the histogram distribution for the camera feed was very different from the images we saw using our eyes and this posed great difficulty for us with regards to setting the threshold value. In the end, we decided to make the threshold value adjustable with

hardware switches and to use a white curtain as a background to eliminate all other background colours. However, in our future works, we will exploit Machine Learning Algorithms to dynamically adjust the threshold values based on the first few seconds of video streams. We use Machine Learning instead of adjusting the threshold in real-time because the timing-constraint requirement does not allow us to add one more simultaneous processing module. However there are still enough Logic Elements on the FPGA chip for us to add a module that pre-process information.

To detect the shirt and change its colour, we first we tried to use complex algorithms. We soon found though that these algorithms took too many clock cycles to run for a viable FPGA implementation, for example, division by a variable would only be completed in 3 clock cycles. As a result of this, a lot of time was spent redesigning the algorithms to meet the FPGA timing constraints.

Another challenge we encountered was the hardware resources that were provided to us for carrying out the project. Being only allocated with 4GB of disk space, each high-level synthesis module had to be archived and recreated every few days by copying the source files to a new project. Computer performance was also a hindrance, as by the time the project had neared completion, our design was so complex that the compilation process took nearly half an hour to complete.

Conclusion

Our system provides the user with a number of modes which allow them to change the appearance of their T-shirt. They can change its colour using the palette and colour picker, they can apply a mesh or a rainbow effect to it, and they also have the option of printing a smiley face, which tracks the shirt as you move.

Originally one of our planned options was to allow the user to change not only the appearance of their T-shirt, but to infact replace it with other items of clothing. This is one limitation of our design at present. Whilst it may be possible to build our own memory controller to allow us to store images of other items of clothing, we did not feel there was enough time to do this, particularly with the project focus of image processing rather than data interfaces.

The aims of this project were to use high-level synthesis to develop a system that could demonstrate real-time image processing and make use of the parallel processing that is possible with a FPGA. Over the past four weeks, we have learnt a lot about each of these techniques and it has been very rewarding to design and construct our own FPGA-based system. Despite the problems we encountered, such as the poor image quality provided by the cameras, we believe our project has met all of its aims. Our system provides a way of trying different T-shirt colours and styles without having to change clothes or even visit a shop, and we are delighted that it has been a success.

Acknowledgements

We would like to thank the three Graduate Teaching Assistants who have provided us with valuable advice and guidance throughout the course of the project:

- Rui Policarpo Duarte
- Mudhar Bin Rabieah
- Michail Vavouras

We would also like to thank the project organiser:

- Dr Christos-Savvas Bouganis

References

Calypto, 2011. *Catapult C Synthesis User's and Reference Manual - University Version.* Santa Clara, CA: Calypto Design Systems, Inc.

Drawbaugh, B., 2010. *HD 101: Overscan and why all TVs do it.* [Online]
Available at: http://www.engadget.com/2010/05/27/hd-101-overscan-and-why-all-tvs-do-it/
[Accessed 23 May 2013].

Fingeroff, M., 2010. *High-Level Synthesis Blue Book.* Hardcover ed. s.l.:Xlibris Corporation.

Mentor Graphics Corporation, 2011. Architectural Constraints Loops. In: *Catapult University Version (UV) Online Help.* s.l.:s.n.

Microsoft Corporation, 2012. *Top-Down vs. Bottom-Up DIBs.* [Online]
Available at: http://msdn.microsoft.com/en-gb/library/windows/desktop/dd407212%28v=vs.85%29.aspx
[Accessed 23 May 2013].

Skogstrøm, R., 2012. *INF5430: High level synthesis.* [Online]
Available at:
http://www.uio.no/studier/emner/matnat/ifi/INF5430/v12/undervisningsmateriale/roarsk/hls.pdf
[Accessed 22 May 2013].