Design of a Generic Interface for Visual Output with CλasH

Vincent Smit
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
v.j.smit@student.utwente.nl

ABSTRACT

This paper researches the possibilities for generalization of visual output from an FPGA by looking into and implementing two different screens and comparing the results. The goal is to extract the similarities from these results and implement a generic visual interface. This generic interface will account for transferring the visual information to the screen that is selected, without knowing the specific details of that screen. For this research the Altera DE1 FPGA board is used. The research has focused on the VGA screen and a the LT24 Terasic screen that is driven via the GPIO pins on the board. The problems encountered and discussed are the need for memory and the abstraction of screen resolution and the colour settings for each screen. This discussion results in an implemented example for the individual interfaces. A generic interface has been implemented as well. It is probably not yet the best solution, and possible improvements for the future have been identified.

Keywords

CλasH, VGA, LT24, FPGA, generic interface

1. INTRODUCTION

A Hardware Description Language (HDL) is a language developed for the purpose of the design of digital logic or electronic circuits. The process of designing hardware with such a language however, is a tedious one. More complex designs are often not simply written in a HDL, but the translation from specification to hardware design includes multiple languages that mismatch the HDL and is therefore errorprone. Besides this translation, the HDL itself is often missing out on some higher order properties like parameterization and abstraction.

In order to improve the process of hardware design, the CAES group at the University of Twente has developed a language called CλasH. This is a functional HDL based on the functional programming language Haskell.

Compiling a hardware design with CλasH yields synthesizable VHDL, that can run on an FPGA. An FPGA often runs on a board that contains more hardware components that are controlled by the FPGA. The components that this research focuses on, are those considering visual output. Specifically an interface will be developed for the LT24 Terasic display module and compared with an existing VGA module. These will then be compared to possibly yield a generic interface for visual output on the Altera DE1 FPGA development board.

The comparison focuses on several different issues. For each of these issues the need and purpose of generalization will be discussed. First of all there is the use of memory for storing the next output data. Second there is screen resolution and last is the colour coding. The resolution is the amount of pixels displayed on the screen and the colour coding is the encoding of colour information per pixel.

The goal of a generic interface is reducing the complexity of controlling visual output. In other words, the issues mentioned above are covered by one interface, which is applicable to different screens. Besides these possible similarities, both VGA and the LT24 are driven by different protocols. The generic interface should therefore also contain some screen specific logic to drive each individual interface. Concluding, simply knowing how to work with one interface is thus sufficient to use a set of different screens, given that screen specific logic of these screens is included in the generic interface.

2. PROBLEM STATEMENT

To reduce the complexity of the research, it will focus on two displays, being VGA and the LT24 Terasic display module. Despite this simplification of the task, a few challenges had to be met.

First of all the display module must be driven by an interface written in CλasH. Although a number of functional HDLs have been developed over the years, hardly any design specification can be found. Most of the support has come from the developers at the University of Twente.

Second, some thought must be put into the management of the scarce available memory. The Altera DE1 used for this research has limited memory, both on and off-chip. In order to make a screen operate fluently however, memory is required for frame buffering. This dependency is derived from the choice between two strategies. Either the outputdata is delivered in a just-in-time manner, or the screen updates with an entire frame each period. The first of these strategies requires a small amount of memory, but rather accurate timing. The latter on the other hand requires a significant amount of memory, since it needs to store the information for all pixels of a frame, instead of a few. Since it is more complex to get the timing right, the first goal of the research is to make the design work with entire frames.
Finally, the developed interface for the LT24 Terasic will be compared with the ported VGA interface. The features on which the comparison is based are the issues introduced in the previous section, being the use of memory, the resolution and the colour coding.

2.1 Research Questions
The following research questions have been derived from the problem statement above.

1. Considering the low amount of available memory on the FPGA, what techniques can be used to keep the memory cost of a frame buffer within the hardware limitations?

2. Do the VGA and LT24 Terasic visual interfaces yield sufficient similarities to design a generic interface using CλasH?

3. RELATED WORK
Most of the work related to this project has been performed by the University of Twente itself. The CAES group is responsible for developing CλasH and all of its documentation [2][3]. Further information on hardware design with functional languages comes from G. Chen [4].

A lot of work with VGA has been done as well. MS Khuzhalí[5] has designed a system that utilizes the VGA component on an FPGA. As for the colour coding, LW MacDonald[6] has been researching computer graphics.

4. APPROACH
In order to get a proper insight into the design specifications of a generic interface, the specifications for the individual interfaces need to be clear. Therefore, both interfaces will be discussed elaborately. First of all the memory issues concerning the frame buffering technique are described. Thereafter, the VGA technique is discussed, followed by the LT24 Terasic. The last subsection is reserved for the comparison of both interfaces.

4.1 Memory
Frame buffering is the process that loads one or more frames into memory before they are loaded to the actual display. It is one of the techniques available to assure that screen data is always present at the time that it is needed. Naturally, such a frame needs a place to be stored before it is used. Here, memory comes into play.

Looking at the available memory on the Altera DE1 board, three components are available. First an 8MByte SDRAM, second a 4MByte flash memory and last a 512KByte SRAM. To make an appropriate choice the data handled and the frequency of use of this data is important. Considering a screen that refreshes sixty times each second, which is not unreasonable, the frequency of writing and loading a frame is high. Therefore the flash memory seems unfit. Flash memory is more suitable for long term storage. Also, a great deal of timing is involved and a fast responsive memory would be ideal. The SRAM is rather small, but controlling it is quite simple. Writing and reading of data can be done in an asynchronous manner and writing one pixel at a time is decently possible. The SDRAM has as an advantage that it is larger than the SRAM. However, it is more complex to control it. It is not preferable or maybe not even possible to write one pixel at a time to an SDRAM because of the latency and bursts. The latency is introduced because the SDRAM is synchronized with the memory bus and therefore has to wait for the bus to be ready. Once it is ready, you can write a larger amount of data, called a burst, within a small period of time. However, this introduces so much overhead that the SRAM is favored in this research.

4.2 VGA Interface
Considering the output possibilities for the Altera DE1, programming an interface for VGA output is a sensible choice. The challenges that come along with this choice, are explained by elaborating on the VGA standard. In order to display anything on a screen, the screen is represented as a grid. This grid or frame is made up of horizontal lines, which in turn are build out of the individual pixels. Thus the VGA standard resolution of 640×480 actually means 480 lines of 640 pixels. The choice of resolution affects both the memory and timing problems of the design. Both are explained in this section.

4.2.1 Resolution and colour coding
The VGA standard supports many resolutions, varying from 640 by 400 pixels up to 2048 by 1536 pixels. Choosing a large resolution for the screen will require a large amount of memory when using a frame-by-frame strategy. Since the choice of memory for storing the next frame is the 512Kb SRAM module, the size of the SRAM is the upper limit for the size of a frame.

When considering a one frame at a time approach, an entire image will be saved in memory before it is written to the screen. This means the information for every pixel needs to be stored. As an example we take a 640 by 480 pixels screen with 265K (18 bit) colours. This requires 640×480×18 = 675,600 bytes of memory. Since there is only 512Kb available, a smaller frame size is required. The resolution is already rather small. Therefore reducing the amount of displayable colours is the better option.

Proceeding with the conclusion above and considering it is an industry standard, the reasonable choice for a the resolution is 640 by 480 pixels. This leaves the choice for the number colours displayable per pixel. Considering the 256,000 16 bit address, it is impossible to assign each of the 640×480 = 307,200 pixels its own memory address. To still be able to store an entire frame in memory it is necessary to store information for more than one pixel in one memory location. Two different approaches are considered for solving this problem. The first approach takes eight bits, or one byte, as colour information. Within this byte, three bits are reserved for both red and green. The remaining two bits encode the blue value. This results in a 640 by 480 pixels screen capable of showing 256 colours. Since two bytes can be stored in each memory address instead of one, half the memory is required. The second approach takes twelve bits per pixel, and stores four pixels in three memory addresses. This would result in a screen that can show 4096 colours, but requires shift register for dealing with the memory address sharing. Although this results in a better colour palette, it is more complex than the one byte solution. Therefore, the one byte solution is the preferable one.

4.2.2 Timing
Driving a VGA display, besides resolution and colours, also entails signal timing. The process of signal timing ensures that each pixel shows the colour it is supposed to. Programming a VGA screen driver thus requires a certain understanding of this process. There are three analog signals that matter for the purpose of this research. First there is the main video signal. This signal contains the red, blue and green values for one pixel. Addition-
ally, there are the horizontal synchronize (HSYNC) and vertical synchronize (VSYNC) signals. Their purpose is a consequence of the mechanics of the CRT monitor. These monitors have an electron beam that moves across the screen pixel by pixel, line by line and frame by frame. In order to tell this beam when the end of a line or frame is reached, the HSYNC and VSYNC signals come into play. Although modern LCD screens do not operate with an electron beam, the principle is similar.

As can be read from Figure 1, each time the beam reaches the end of a line, the low active HSYNC has to be triggered. In terms of horizontal and vertical indices, when the HSYNC is triggered the $x$ value is reset to zero and the $y$ value is incremented by one. Reaching the end of the last line requires triggering the low active VSYNC. This resets both the $x$ and $y$ value to zero. The frontporch and backporch are time intervals before respectively after the actual SYNC signals. In these intervals, no visible data is written to the screen and are therefore named blank periods. The entire process of writing a frame including the blank periods is called a screen refresh. The number of screen refreshes per second is called the refresh rate, where refresh rates of 60Hz or 72Hz are most common for VGA screens.

These signals need to convey all the colours in a screen. Therefore the VGA signals run on a so called pixel clock.\(^5\) By sampling the main line on the period of this clock, it is possible to extract pixel information one pixel at a time. Knowing the resolution and refresh rate of a screen, the frequency of this pixel clock can be calculated. The values in Table 1 are relevant for a screen with a resolution of 640 by 480 and a refresh rate of 60Hz. The time required for displaying a horizontal line is measured in pixels. The vertical equivalent is measured in lines.

The frequency of the pixel clock can easily be derived from these values. First, we calculate the vertical sync frequency, where 60 times a second, 525 lines need to be written.

vertical sync frequency = $60 \times 525 = 31,5\text{KHz}$ \hspace{1cm} (1)

\[
\text{vertical sync frequency} = 60 \times 525 = 31,5\text{KHz}
\]

And within every line, 800 pixels are written.

\[
\text{pixel write frequency} = 31,500 \times 800 = 25,2\text{MHz} \hspace{1cm} (2)
\]

From Equations (1) and (2) can be derived that the pixel clock in this example runs with a frequency of 25,2MHz.

However, since the blank periods are included in the calculation, over 25% of the time is used for the SYNC signals. These blank periods are interesting, because in these time intervals the memory is unused. Therefore these intervals can be utilized for writing the next frame into memory. The time this process requires, knowing that a memory read or write action takes 10-15 nanoseconds, is shown in Equations (3) and (4). The number of memory actions can be cut in half, because two pieces of pixel information fit in one memory location.

\[
\text{blank time} = \frac{480 \times 140 + 45 \times 800}{25,2 \times 10^6} = 4,1\text{ms} \hspace{1cm} (3)
\]

\[
\text{max write time} = \frac{(640 \times 480)}{15} = 2,3\text{ms} \hspace{1cm} (4)
\]

Thus, writing framedata in the blank periods of the VGA screen is a decent strategy.

Now that all challenges considering VGA have been elaborated on, the implementation is rather straightforward. The next step entails the specification of the LT24 Terasic screen, before moving on the generic interface design.
4.3 LT24 Interface

The second interface implemented is the LT24 Terasic LCD and touch screen. For this research the touch component is ignored, as the goal is visual output rather than touch input. This component contains a 320 by 240 screen with 16 bit colours, but unlike VGA it offers no choice in either of the features. Even the refresh rate is set at 50Hz. However, the internal architecture of the LT24 implements a part the VGA logic. The device itself handles the synchronization signals.

In contrary to the analog VGA signal, the LT24 is controlled with digital signals. The signals are transferred to the screen via fourty general-purpose input/output (GPIO) pins on the FPGA board. In order to drive the LCD display, the ILI9341 LCD Driver is used. The communication between the FPGA and the display is handled by a few signals.

1. CSX - a low active chip select signal
2. D/CX - Data or Command selection signal
3. WRX - a low active write signal
4. RDX - a low active read signal
5. 16 data pins, utilized for both reading and writing

The CSX signal enables or disables the ILI9341 chip. The WRX signal enables writing data from the host to the LT24, the RDX signal enables reading data from the device. Writing data to the display is timed on the falling edge of the WRX signal. All sixteen data pins are used when the D/CX signal is high, or in data mode. When the signal is low, thus in command mode, only the lower eight bits are used. In order to write a full frame to the screen, the write action has to be triggered 320 * 240 = 76,800 times. This takes approximately 1.3ms, equivalent to Equation (4). Together with the known refresh rate, this results into a 3.84MHz pixel clock with artificial blanks to prepare the next frame.

4.4 Generic Interface

Both the VGA and the LT24 screens have been discussed and some similarities and differences can be determined. The major similarity is the use of memory for storing the next frame. The colour coding of the screens is a major difference, just like the communication of the programmed interface with the actual hardware. In the same order as mentioned here, all issues will be elaborated on. The last section looks into the dataflow between all the separate components.

4.4.1 Memory

As introduced, both screens require memory to store the next frame that is to be displayed. The communication with the SRAM memory is the same for both screens and is therefore easily extracted and implemented in the generic interface. The main purpose of the generic interface is to handle the communication with the SRAM module that stores the frames.

4.4.2 Colour coding

Due to the scarce memory available on the FPGA used, the VGA screen can’t show more than 256 colours. This is equivalent to one byte of data. Of the eight bits, three are reserved for the red component, three bits are also reserved for green colouring and the remaining two encode the blue colour. On the other hand, the LT24 can choose from a palette of over 65,000 colours encoded in 16 bits. Five bits encode both red and blue. The remaining six represent the green value. Dealing with this difference can be done with two different approaches.

1. Maintaining image quality for every individual screen.
2. The individual specifications for colours are ignored and an input colour coding for all the screens is introduced.

The first approach honours all screen colour settings. The second approach considers generics as the most important feature. The advantage of the first approach is the conservation of screen quality, since there is no loss in colour. The disadvantage of colour coding this way raises the advantage of the latter method. That is, knowledge of an individual screen is required from the user. This is exactly the case that a generic interface intends to prevent and therefore the generic approach is favored.

A sensible choice for the colour coding for this interface would be a meet-in-the-middle between the VGA and LT24 screen. This comes down to twelve bits of colour information per pixel. Each of the three colours is represented with four bits. Depending on the screen chosen, the generic interface will be handling the mapping from twelve input bits to either the eight or sixteen output bits. To map the colour bits the following mapping functions have been defined.

1. 4bit to 2bit: right shift by two
2. 4bit to 3bit: right shift by one
3. 4bit to 5bit: left shift by one, binary OR least significant bit of 5 bit result with most significant bit of the 4 bit input.
4. 4 bit to 6bit: left shift by two, binary OR two least significant bits of 6 bit result with two most significant bits of the 4 bit input.

The first two mappings are for the VGA 8bit colour and the procedure is straightforward. The second two procedures for the LT24 require some more explanation. In order to map to a larger number of bits, the binary left shift operation is obvious. The OR operation is performed to cover the complete spectrum of colours. This means that 00 00 is converted to 000000 and 1111 to 111111. Without the binary OR, the latter example would result into 111100, despite having maximum intensity in the 4bit colour setting. The code of this mapping can be found in the Appendix.

4.4.3 Dataflow

For the dataflow of the design, some graphical support is supplied by Figure 2. The arrows indicate the flow of a frame through the different components. A frame enters the generic part and is stored in memory. When a full frame is stored into memory, the frame will be loaded from memory and transferred to the right screen interface. This interface will handle timing communication with the hardware components. The design of a frame is rather simple. It is simply a list of pixels that is stored in to memory one after another. The list is as long as the number of pixels a screen, which depends on the resolution. In contrary to the colour coding, the resolution is not generalized in the design. The reason for this decision is that the LT24 runs on only one possible resolution and this resolution is not a standard for VGA.
The only complexity left for the user, is to implement the stream that loads the frame into the generic interface. This needs to be done carefully, because the interface can only accept new information is the SRAM is not in use. Therefore, the user should write a pixel to interface as long as it indicates that it is free. The return signal of the interface takes care of this. If the interface is occupied, it will not return a signal until it is free again.

5. RESULTS

Following the outcomes of the considerations in the previous sections, the following interfaces have been developed using the CλasH language.

1. A VGA interface that runs a 640 by 480 screen with 256 colours and a 60Hz refresh rate.
2. An LT24 interface that runs the LT24 display module.
3. A generic interface with a 12bit colour interface, that controls the communication with memory and supports the first two interfaces.

6. CONCLUSION

Reviewing the produced result, the conclusion can be drawn that designing a generic interface for visual output is indeed possible. This in itself is not very surprising. It is more important to evaluate whether the implemented design reaches the goal of easing the use of screens and to what extend.

Firstly a review of the memory issue. Because of the timing constraint the smallest of the available memory components has been chosen. In order to load an entire frame into memory, frames have to be rather small. This problem has been solved by choosing a small resolution and a small number of displayable colours. This might not be an optimal solution, but it is functional. A better solution would have both a larger resolution and a larger amount of displayable colours. With both of these properties larger, the quality of the visual output improves. However, this is only possible if the interface does not load an entire frame.

Second is the issue of similarities. Since a design has been presented, the straightforward answer to the question is yes. However, lets take a look at the design and it functionality. The process extracted from the individual components is the interaction with the memory module. Besides, a 12 Bit colour encoding has been introduced to provide generics to colours as well. So, when considering to add another type of screen to the generic interface one can also choose to implement to rather simple interaction with memory also. This comes with the added bonus of conservation of the colour coding.

Since this research has focused on delivering a functional design in a small working example, there exists room for critical notes. For example, when one finds colours and high resolution important the choice for a small memory may be considered wrong. To improve on these features, the SDRAM is required to be able to store more data.

Also the introduced 12-bit colour in the generic interface might not be the best choice. Running every individual screen with its own colour coding is a reasonable alternative. However, the research focuses on the generics and the 12-bit solution is more conforming to the research. Concluding, a generic interface makes life a little easier but it does come at a price. Question is if it is worth paying.

7. FUTURE WORK

Since this research has focused on a simple start for a generic interface, quite a number of tweaks and improvements can be thought of. A major improvement would be to add a just-in-time like algorithm to the design. This would not only result in an alternative for the entire frame. It would also give opportunities for displaying more colours and resolutions, since the memory limitation is mostly gone. Instead of the entire frame, only a few pixels need to be loaded into memory.

An alternative for the memory limitation is using the SDRAM instead of the SRAM for storing frames. Using this memory requires a more complex controller, because of the burst of data and handling the latency. But as stated, it does yield more possibilities for the amount of displayable colours and resolutions.

8. REFERENCES

[1] C. Baaij, CλasH- from haskell to hardware. found online on http://www.clash-lang.org
[8] Terasic. Lt24 card. found online on
APPENDIX

A. CODE FRAGMENTS

Colourdecoders

1 — Colour decoders
2 — Most significant bit is right.
3 decodeLT24 :: (Vec 4 Bit, Vec 4 Bit, Vec 4 Bit) -> Vec 16 Bit
4 decodeLT24 (r, g, b) = r' ++ g' ++ b'
5     where
6       r' = last r ++ r
7       g' = drop d2 g ++ g
8       b' = last b ++ b

9 decodeVGA :: (Vec 4 Bit, Vec 4 Bit, Vec 4 Bit) -> Vec 8 Bit
10 decodeVGA (r, g, b) = r' ++ g' ++ b'
11     where
12       r' = tail r
13       g' = tail g
14       b' = drop d2 b