Camera Controller
Project Report - EDA385

Einar Vading, ael09eva
Alexander Nässlander, ada09ana
Carl Cristian Arlock, ada07car

November 1, 2013

Abstract

This report is on an implementation of camera control hardware with movement detecting capabilities. It describes the process in all steps from design to finished product. Included are thoughts and comments on challenges, solutions and ideas for future improvement. The system is built around a camera bought on ebay and a Nexys-3 platform from Digilent. The end result did not live up to the initial expectations because of unforeseen challenges but a working prototype was delivered.

Contents

1 Introdution

In this project a camera surveillance system with motion detection was developed. The implementation was based around a small camera module using on the OmniVision 7670 CMOS VGA CameraChip^{TM}, hereby referred to as `OV7670`, together with a Xilinx Spartan-6 equipped Nexys-3 board from Digilent. While the initial goal of motion tracking was not achieved, we managed to successfully implement movement detection using a simple hysteresis based algorithm. The reason for not implementing the full motion tracking was mainly time shortage as a result of the camera configuration taking way more time than initially estimated. Both the camera and the VGA had timing constraints that could not be met using software so the entire project was implemented in hardware, synthesized on the FPGA. The initial idea was to do motion tracking in software and then accelerate it using hardware components but as time ran out it was decided that it was simpler to just do detection in hardware rather than adding a Microblaze core to the system. A simple overview of the system is shown in figure 1 and an overview of the implemented hardware components is shown in figure 2. All the components but the final VGA controller were written by us.[1]

2 Implementation

The basic parts of the implementation were identified as

- Camera interface
- VGA interface
- Memory
- Detection system

Each of these parts needs interconnectivity for communication and data transfer. A full system schematic can be seen in figure 3. Hardware utilization on the FPGA can be found in table 1. The list has been trimmed down to fit in the report, omitting all zeroed statistics.

2.1 Camera

2.1.1 Camera Setup

The setup of the cameras many options is done over SCCB (Serial Camera Control Bus), a proprietary bus developed by OmniVision for camera configuration. The bus is similar to I^2C [1] with some subtle differences, the most notable that the data and clock ports are not open drain but regular CMOS logic I/O. Due to difficulties with implementing the bus specification, as can be seen in the challenges section, only write capability was implemented. The registers and their corresponding values that are used for camera initialization are shown in table 3. More details on the specifications in [2].

The purpose of the setup is to make the camera output RGB565 data at a pixel clock rate that is the same as the camera input clock rate supplied

[1] The VGA controller that we ended up using was sourced fromhttp://eewiki.net/x/HgDz

Table 1: Device utilization

Slice Logic Utilization	**Used**	**Available**	**Utilization**
# Slice Registers	239	18,224	1,00%
# used as Flip Flops	239		
# Slice LUTs	535	9,112	5,00%
# used as logic	464	9,112	5,00%
# using O6 output only	261		
# using O5 output only	50		
# using O5 and O6	153		
# used as Memory	64	2,176	2,00%
# used as Single Port RAM	64		
# using O6 output only	64		
# used exclusively as route-thrus	7		
# with same-slice carry load	7		
# occupied Slices	188	2,278	8,00%
# MUXCYs used	296	4,556	6,00%
# LUT Flip Flop pairs used	551		
# with an unused Flip Flop	320	551	58,00%
# with an unused LUT	16	551	2,00%
# fully used LUT-FF pairs	215	551	39,00%
# unique control sets	29		
# slice register sites lost	105	18,224	1,00%
# bonded IOBs	29	232	12,00%
# LOCed IOBs	29	29	100,00%
#IOB Flip Flops	14		
# RAMB16BWERs	30	32	93,00%
# BUFG/BUFGMUXs	2	16	12,00%
# used as BUFGs	2		
# OLOGIC2/OSERDES2s	14	248	5,00%
# used as OLOGIC2s	14		
Average Fanout of Non-Clock Nets	5.42		

Table 3: Camera Setup Settings

Register	Value	Description
CLKRC	0x00	Internal Clock
COM7	0x04	Common Control 7
COM3	0x00	Common Control 3
COM14	0x00	Common Control 14
SCALING_XSC	0x3A	Test Pattern X
SCALING_YSC	0x35	Test Pattern Y
SCALING_DCWCTR	0x11	DCW Control
SCALING_PCLK_DIV	0xF0	Clock Divider
SCALING_PCLK_DELAY	0x02	Clock Delay
COM15	0xD0	Common Control 15

by the camera interface. Also some illegal default values are changed, again see challenges section. The setup is managed by the `cam_setup` IP which in turn uses a `SccbInterf` IP component for communication timing and driving the I/O-pins. The `cam_setup` IP is a simple state machine where the config state consists of a case construct and a counter that counts up. Depending on the counter value a different register is written. When the counter counts past the last register associated value the state machine jumps to a done state from which it never leaves. This works since the register values are only written once, at system startup. The actual communication is handled by a `SccbInterf` subcomponent. In `SccbInterf` everything needed for SCCB write operations is implemented as state machine that shifts out data and clock on the respective pins.

2.1.2 Camera Interface

The camera is capable of outputting a range of different formats. The format chosen for this project is RGB565 color at a 640 by 480 pixels resolution. For the camera module to supply any data at all a main clock, `xclk`, is needed. This clock, 10 Mhz, is supplied by the `cam_interf` IP and is clocking the internal DSP responsible for format conversion and image scaling. The `OV7670` then supplies a `pclk` that clocks outgoing pixel data from the camera on an eight bit parallel bus. A line is signaled by means of the `href` signal and each new frame is indicated by the `vsync` signal. The state, hi/lo, of these signals can be configured during camera setup but for this project the default values were used with `href` and `vsync` active high. It is worth to note that `vsync` signals a new frame and as such an active high `vsync` means that pixels should be read only when `vsync` is low. The `cam_interf` IP samples the `pclk`, `href` and `vsync` signals from the camera and contains a state machine that grabs one byte and waits for the next or writes the RGB values to memory depending on previous input. After failing to use the on chip scaling features the `cam_interf` IP was rewritten to also do scaling of the input image. The scaling works by averaging four by four pixels at a time and using the averaged value as image data. This means that the video ends up at 160 by 120 pixels, (640/4) by (480/4).

2.1.3 Camera Connection

The camera was connected to the Nexys-3 board through the `Pmod` connectors. The mapping can be seen in table 4. A simple connector was built and while it provided basic functionality, it was later discovered that the long cables produced interference. This in turn added some noise to the picture. The connector can be seen in figure 5. The camera itself can be seen in figure 6.

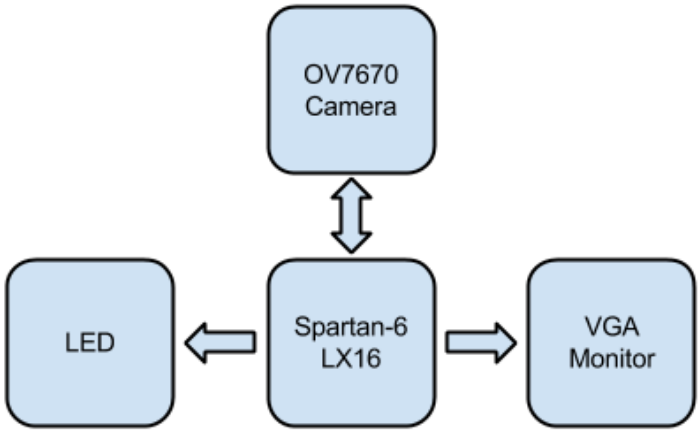

Figure 1: System Overview

2.2 Memory

After a few iterations and tests it was the general consensus that the onboard 16MB of Cellular RAM was too slow to use as video memory. It would work, but as speed was deemed more important the plan was scrapped. Therefore a BRAM controller was implemented in VHDL, named `mem`. This IP contains three memory banks

- Bank 1 - for buffering video from the camera and output to movement detection
- Bank 2 - for storing a single frame for movement comparison
- Bank 3 - video data for output to the VGA monitor

Because of the limited size of the BRAM, the picture was scaled to 160 by 120 pixels, more on that in the camera section. There was a need for three banks because of the limited amount of read and write ports on the BRAM. Limits in the synthesizing software produced unexplained errors until the port issue was resolved. With this solution, there is support to add printouts to the screen where the movement occurs. This is a bonus feature from solving the challenges. However due to time constraints it has not been implemented yet.

The `mem` IP is implemented with three memory banks and three processes. The `cam` process writes bank 1 and 3 continuously while bank 2 is only written when the movement detector demands it, more on that in the movement detector section. The two other processes, `det_rd_mem1` and `vga` outputs memory

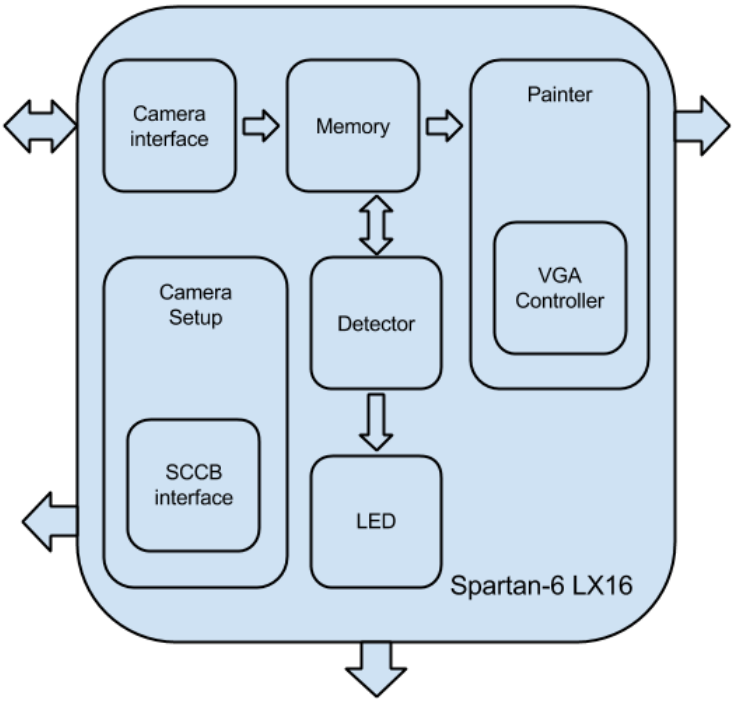

Figure 2: Hardware Overview

contents to the `detector` and `painter` IPs respectively. More on that in those sections. The memory banks requires an address to automatically return the data located there or if written to, overwrites the data. This can be done on two ports simultaneously at a single clock cycle delay.

2.3 VGA Interface

The VGA Interface takes care of the communication between memory bank 3 and the monitor. The interface is divided into two parts, the `painter` IP and the `vga_controller` IP. The `painter` reads pixels from memory bank 3 and outputs them on the VGA connector. Since the frame that is stored in the memory bank 3 has a resolution of 160 by 120 pixels and the monitor is clocked at a resolution of 640 by 480 pixels, the `painter` applies scaling to reach the required output. While the `painter` reads from memory, the `vga_controller` notifies the `painter` when it should output the one byte RGB data. In order to determine when the data should be sent to the monitor the `vga_controller` must do some calculations based on a VGA standard, according to the VGA specifications [3], using a 25 MHz pixel clock. The standard uses two signals in order to tell where a certain pixel value should be applied on the monitor. These two signals are horizontal and vertical sync. When the horizontal sync is high, the monitor picks the pixel values and prints them on the current row. When that row is completely printed it jumps to the next row to repeat the same process. From the first row to the last row, the vertical sync is high. It stays high for one frame and then drives to zero for two cycles according to the

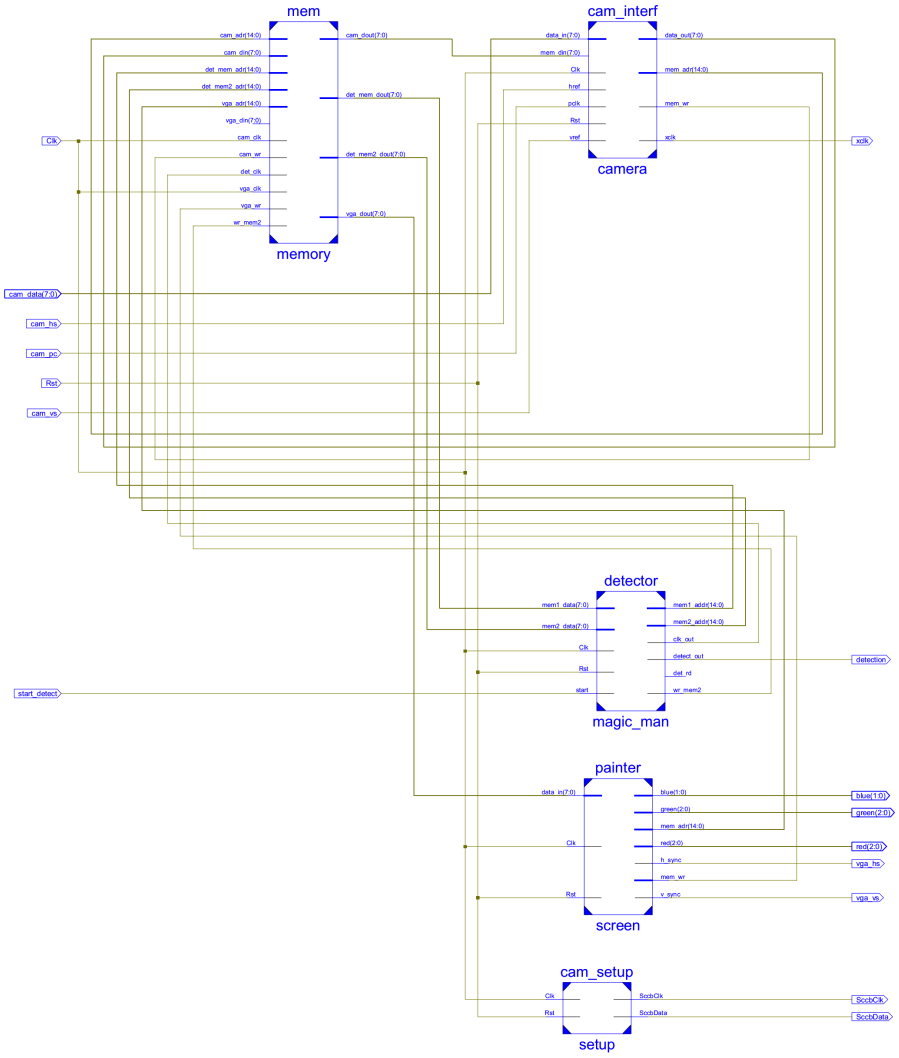

Figure 3: System Schematic

pixel clock. The vertical sync signal defines the refresh frequency of the display, or the frequency at which all information on the display is redrawn. This is shown in figure 7. Both the horizontal sync signal and the vertical sync signal are driven high longer than the display time (HBLANK and VBLANK). The reason is that the standard is suited for old CRT (Cathode ray tube) monitors that needs the time to get into the new position. The `vga_controller` tells the `painter` when the monitor is ready to recieve by sending a `display_en` signal. The `painter` keeps track on where the pixel should be printed and therefore where in the memory the pixel value is located.

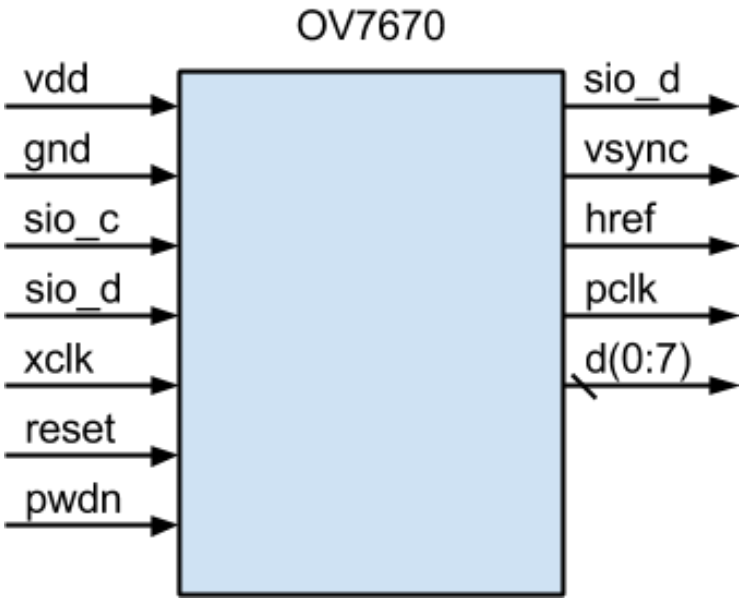

Figure 4: Camera Interface

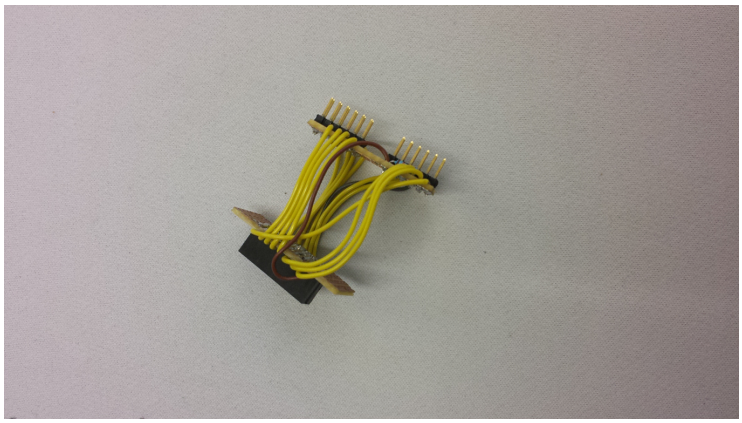

Figure 5: Camera Connector

2.4 Movement Detector

This part of the system, if activated, reads two of the memory banks. Where bank 1 is the continuous stream of input from the camera and bank 2 is a fixed frame, the reference picture. The `detector` IP compares each pixel in the two frames and decides, based on a set sensitivity, if there is a big enough difference between the two. The `detector` then decides if there are enough changed pixels to warrant detection. This sensitivity is also set manually. If the system detects movement it outputs a signal to `LD1` on the Nexys-3 board. The `detector` then resets the reference frame and restarts the process.

The `detector` is implemented as a simple state machine, waiting for input

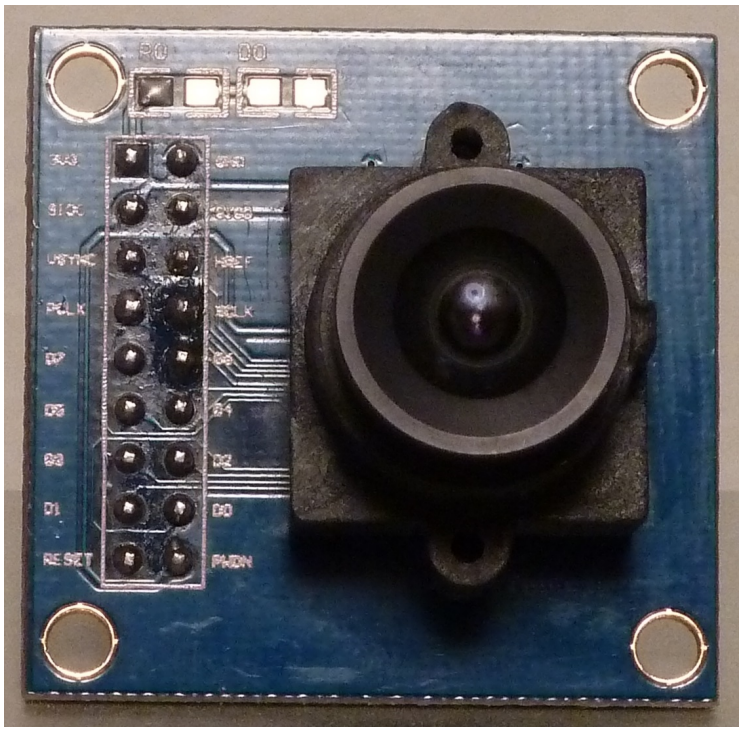

Figure 6: The OV7670 Camera

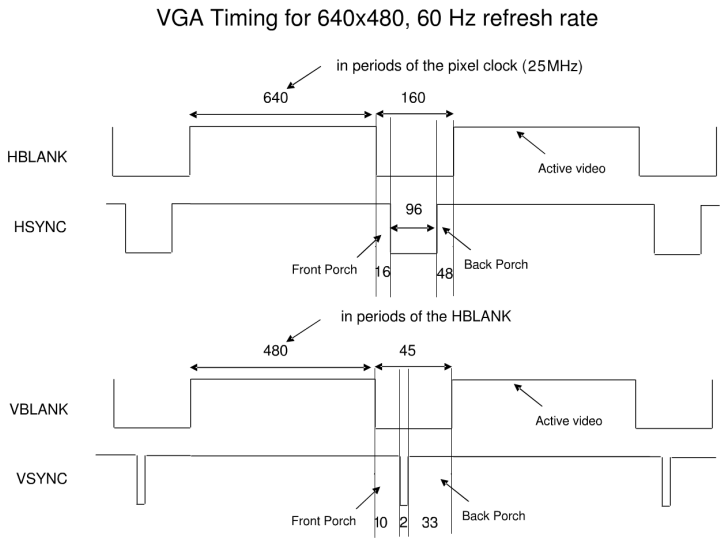

Figure 7: VGA Timing, unmodified original picture taken from http://www.docstoc.com/docs/53768210/VGA-Timing-for-640x480-75-Hz-refresh-rate

from user. When `SW1` is high the state machine is activated. It then awaits the

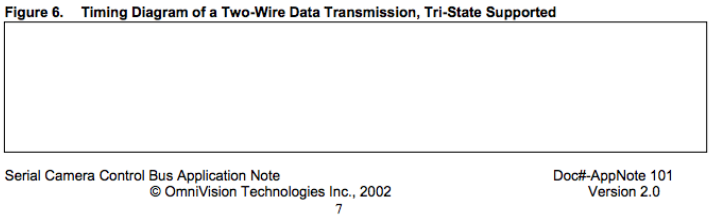

Figure 8: Example of SCCB Specification shortcoming. The only timing diagram that describes tri-state capable two wire communication is empty

complete reference frame before comparing pixels.

The movement detector can with some fine tuning detect if someone walks past the camera. There is a lot of noise in the picture stream from the camera and also the DSP is modifying the picture heavily. That is why the movement detector sometimes indicates false positives or just doesn't detect movement at all. The movement controller also needs to be tuned according to each camera, since they seem to differ a bit. Might be because of external circumstances like blocked lens, different lens focus or the interference between the cables in the connector, as can be seen in figure 5.

3 Challenges

During this project there were some challenges and problems, some were solved or worked around and some remained obstacles.

3.1 Cellular RAM too slow

Initially the Cellular RAM was considered, because of the advantageous size. The main reason why it wasn't possible to use that type of memory is that it would be too slow to use with the desired frame rate. So, in order to maintain the speed of the system the memory type was changed to BRAM instead. The speed of the BRAM suited the system better since one can request an element in the memory and get it within the same clock cycle.

3.2 BRAM too small

When solving the problem with slow memories, another problem emerged. The BRAM is only 576 Kbits large and that is too small to store a full size VGA frame in eight bit color. There was no way to fit the frames desired in the memory. But the system needs to store frames for the detection to work. Instead of storing full sized frames they were scaled to down 25% of their original size before storage.

3.3 SCCB specification

The specification of the SCCB as supplied by OmniVision was a big hurdle. While the specification covers the three wire version at some length, the two-wire version used by the `OV7670` is barely mentioned. Also, only one version of the

specification did refer to a timing diagram but the figure referred to was empty, see figure 8. Another issue with the specification was the nomenclature. One example of this is the fact that the ninth, and last, bit of each serial transmission is called the `don't care bit` initially leading one to believe that it's value was not important. Reading on in the specifications does reveal however that the so called `don't care bit` is thoroughly specified. After some work the write functionality was implemented successfully and as only one camera initialization was performed, at startup, and then never changed it was good enough for this project.

3.4 Illegal default values

The `OV7670` Implementation Guide [5] is a document that aims to aid in development of a camera back end using the `OV7670` as a camera front end. This document contains a small note that states

> Note: All registers shown as reserved have no function or are very sensitive analog circuit references. Use OmniVision reference values (not default values).

The mentioned reference values are unfortunately not available in the datasheet but only through OmniVision after signing an NDA (Non-disclosure agreement) and even then not to individuals. Since there was no way of knowing what values to output, there was a lot of guessing and futile searches on the internet involved. The image output from the camera did not really look anything like what was expected but after a lot of trial and error there was success in finding a set of values that at least gave a recognisable image.

3.5 Pixel clock on non clock port

The Spartan-6 uses clock buffers for clock signals on the I/O pins. These clock buffers are used to provide a clean skew free clock signal but are only available on certain pins. The pinout of the interface cables between the camera and the FPGA did not allow for use of these clock buffers and so there was an error message when trying to synthesise

> 08 - A clock IOB / BUFGMUX clock component pair have been found that are not placed at an optimal clock IOB / BUFGMUX site pair...

This was solved by instead of using `pclk` as a clock source, `pclk` was sampled at each system clock, at ten times the `pclk` frequency, and using a state machine that keeps track of which clock pulse it is currently at. The best solution to this problem would be to re-route the `pclk` signal to a location that could be buffered with a clock buffer but as time ran out it was decided to go with the not so good solution just to have something to show. If in the end there would have been time to spare it could easily have been changed back to the edge driven solution and another pinout for the interconnect could be manufactured.

3.6 DSP causing trouble

The camera chip has a built in DSP that continuously adjusts gain and white balance among other things. This makes it harder to use a simple algorithm that counts pixel changes, as even under near identical conditions the pixels are always varying. According to the datasheet the DSP can be circumvented and the camera is able to output the raw values from the image matrix. Due to the problems with configuring the camera it was never possible to test this but with more time one could choose raw Bayer pattern RGB values and do all processing in the FPGA instead of reading preprocessed RGB565 values.

3.7 Hard to prototype in software on the board

Due to the tight timing constraints for both the camera and the VGA interface it was difficult to find things to prototype in software. Given that an interface was needed for the camera and the VGA in hardware with memory access and all it made sense to make the movement detector in hardware as well. With more luck configuring the camera the plan was to have motion tracking in software running on a Microblaze. Then the bottlenecks could have been identified and hardware accelerators could have been implemented. As things turned out all that was managed to prototype in software was an OpenCV [4] powered background subtraction algorithm on a laptop. In the event of a successful project the plan was to port the OpenCV algorithm to the Microblaze and use that on a scaled down version of the actual image.

4 Lessons Learned

Besides from the obvious like getting more comfortable with VHDL for synthesis and working with the Xilinx tools and the Spartan-6 in general there were some things that will be taken in consideration into future projects, especially embedded systems projects.

4.1 Do calculations beforehand

This might sound obvious, but there was actually a fair amount of time spent in the beginning, implementing memory controllers for the on board Cellular RAM, thinking one could use it as a buffer between the camera and the VGA controller. Looking at the datasheet and calculating the maximum throughput it did look promising. But when it was finally realized that maximum throughput is only important when doing continuous reads and writes, and what's needed for the camera and VGA are single reads and writes at high frequency, the idea of using Cellular RAM was quickly scrapped and instead the image was scaled to fit the BRAM. Had this been thought through from the start and proper calculations had been done on how and when the components connected to the memory needed data it would have saved the trouble of writing and configuring memory controllers for the Cellular RAM.

4.2 Implement in small steps

This is something that was actually practiced from start to finish in this project. The IPs were broken down into subcomponents, that was implemented and thoroughly tested before using them in larger, more complex components. This led to the testing of large components being more about interconnections than about function. This is good since functional testing of a small component with one simple function is unproportionately simpler than doing the same for a complex component with many functions.

4.3 Datasheets does not always have the answer

It was made clear that the datasheets sourced were not always accurate or complete. One example being that of the illegal initial register values, see problem section. This might not be a problem for big corporations buying large quantities of cameras from OmniVision since they could probably get in contact with sales engineers and the like, but one can imagine it being a problem for small companies and startups that does not have the bargaining power of a large company. Even though one can't know if this is true its something worth taking with in the event of starting a business based on some technology.

5 User Manual

Since this project implements movement detection in pure hardware, no software is needed. First of all you need to connect the camera to the Nexys board. The port mapping in table 4 show which ports are connected where. `SW0` is the `reset` switch, with active low. The system should be reset after flashing the Nexys-3 board. `SW1` activates movement detection, with active high. A monitor capable of 25Mhz VGA needs to be connected to the VGA connector on the Nexys-3 board. When the camera is correctly connected to the Nexys-3, connect the FPGA to your computer via the dedicated micro USB jack. Then open the program named Adept. Make sure that Adept has located your device. If so, load the top.bit file to your Nexys-3. When the system is running you must make sure that `SW0` is high. To reset the system, drive it to zero and back again. `SW1` is used for toggling the motion detection, drive it to one to enable. `LD1` will indicate movement detection.

6 Contributions

The three authors worked closely on every part of the project but here are lists on how the work was divided.

6.1 Einar Vading

- Presented the initial idea, got the cameras from ebay
- Decided on the BRAM memory controller after issues with the initial tests
- Part of the final presentation

Table 4: Port Map

Nexys-3 port	OmniVision 7670 port
T12	RESET
V12	SIO_C
N10	VSYNC
P11	PCLK
N9	SIO_D
U11	HREF
V11	XCLK
K2	D[7]
J3	D[6]
K1	D[5]
J1	D[4]
K1	D[3]
K3	D[2]
L3	D[1]
K5	D[0]

- Wrote initial versions of `cam_interf`, `mem` (BRAM controller), `cam_setup`, `SccbInterf` and `painter`.
- Project report sections, Introduction, Camera, Problem section save from Cellular RAM and BRAM subsections, Lessons learned.

6.2 Alexander Nässlander

- Designed initial VGA controller and performed tests.
- Looking up standards for VGA timings and calculations.
- Early state Cellular RAM controller for the VGA controller.
- Part of the final presentation
- Project report sections VGA Interface, part of the problem section, part of the user manual section

6.3 Carl Cristian Arlock

- Part of the project proposal presentation
- Considered memory solutions, presenting several memory Cellular RAM controllers
- Implemented `detector` and performed related tests
- Modified `mem` to fit the needs of the extended system
- Tested the camera setup and interface module

- Project report sections Memory, Movement Detection, part of the user manual section
- LaTeXenthusiast and layout artist
- Spelling and grammar checker

References

[1] I^{2}C on Wikipedia, http://en.wikipedia.org/wiki/I%C2%B2C 20131027

[2] OmniVision Advanced Information Preliminary Datasheet, http://www.voti.nl/docs/OV7670.pdf 20131027

[3] VGA on Wikipedia, http://en.wikipedia.org/wiki/Video_Graphics_Array 20131027

[4] Open Source Computer Vision, http://opencv.org/ 20131027

[5] OmniVision Implementation Guide, http://www.robokits.co.in/datasheets/OV7670.pdf 20131027