Clocking and Pipelining

Computer Design course Practical 3

October 2011

1 Objectives

This practical aims to give further experience with clocked processes and the use of pipelining to decrease the clock period, which is an important property in a digital design. Example of the utility of a digital design is also provided, as the design given interfaces other real hardware. Further experience with the Verilog language, as well as with digital design and constraints, is also achieved.

This practical is intended to be performed in groups of two. This practical requires that you write a report of your activities and hand it in with your labsheet. The report is individual, and your attention is drawn to the guidelines on plagiarism. Developing and making the design work is a group task, but writing the report and framing the explanation on how it works is an individual task.

2 Introduction

For this practical a ISE project is already created that you can use. This project already has the correct top-level Verilog and UCF definitions, so you should not need to worry about those details. The project file can be found on the course website as a zipped folder. Unzip this in a suitable location and open the project in ISE.

You will find a large design with multiple modules already defined for you. As it stands, the entire project is already complete and can be synthesized, you should do so to convince yourself it operates correctly. Additionally, you may wish to do so to see the base functionality of the system that you are expected to preserve through your modifications to the design.

3 Design Description

The design is of a simple VGA pipeline, designed to drive a LCD or CRT monitor at 1280x1024 resolution, and a simple RS-232 receiver, accepting byte data at 115200 baud. Any data sent over the RS-232 connection is displayed on the VGA monitor attached; the design therefore works as an output-only serial terminal. A detailed breakdown of the pipeline is shown in figure 1.

Note that the escape character (0x1B), if sent, is treated specially, and is used to modify the internal number of text lines and columns in the design. You may wish to use this to see what the effect is.

The VGA pipeline consists of five stages, all operating at 108MHz, the pixel clock. The pipeline has to output the data for 1 pixel every clock cycle. This is a requirement of the particular VGA resolution chosen, and cannot be changed. The stages of the pixel pipeline is:

1. Clock counter stage (dotclock), which generates the sync signals.

2. Address Generator (dotaddr), which generates the RAM address of the character that we need a pixel from.
3. Fetch Stage (fetch), which fetches the character value (8-bit ASCII) from the character buffer.
4. Character Map Stage, which looks up the pixel value in another RAM, given the character value and the exact pixel location in the character.
5. Output Stage, which puts the pixel on the wires that leads to the monitor.

The RS-232 receiver runs at 50MHz, and pushes the received data into the RAM accessed by the fetch stage. You will note that the RS-232 and VGA pipeline run at different frequencies, and you might also know that this is in general a tricky thing to do. For the design present, however, the issues have been overcome and you need not concern yourself with that detail.

The reset, which resets the entire design, is tied to the ”BTN West (D18)” push button.

Your board should be connected to a DICE monitor in the labs; you should be able to switch between the board output and the DICE desktop with the input select on the monitor itself. Your board should also already be connected to the serial port of a DICE machine, though you may need to figure out which DICE machine. The serial port is accessible as a file, `/dev/ttyS0`, on the DICE machine; you may need to set the serial speed of it correctly using the command ’`stty -F /dev/ttyS0 115200`’. Writing text to the board from a file on the DICE machine should then be as simple as using ’`cat filename > /dev/ttyS0`’ or ’`echo "text" > /dev/ttyS0`’ in a terminal.

The design uses the multiplication module, `simplemult`, in two places; both of them immediately before accessing the character buffer, once in the 50MHz domain and once in 108 MHz domain. This should be clear from the pipeline diagram in Figure 1.

This is the module which you will need to modify - as given to you, the module uses the Verilog + and $\star$ operators on 8-bit fields. This module gets automatically implemented as a hard block (also referred to as hard macro) for the multiplication, and optimized logic for the addition. Unfortunately, and as evidenced if you synthesize the base design, the path through this block is already one of the longest in the entire design, and needs work. In addition, we cannot use the + and $\star$ operators in an ASIC implementation but have to supply our own logic.

The task, then, is to replace these operators with other logic that is more suitable.

4 Replacing the Addition

You should first replace the + in `simplemult` with an implementation that does not use any arithmetic verilog operator at all, in `simplemult` or in any child modules you choose to create.

You can use the half and full adders already developed if you wish. Note that the return from `simplemult` is 17 bits wide, and you are adding 8 bits to 16 bits from the multiplication. You will need to rewrite the existing code to keep the $\star$ statement while replacing the +.

Your adder should not present a significant timing problem, even at 108MHz. Note that the original `simplemult` module contains a single register bank holding the result of the operations. It is assumed, by the rest of the design, that the `simplemult` module has an inherent single clock cycle delay. This should be visible in Figure 1 as well as in the code itself.

Once you have a working adder in the FPGA, that meets timing, show it to the lab demonstrator to get it signed off.

5 Replacing the Multiplication

You need to build a multiplier that multiplies 8 bits with 8 bits, generating a 16 bit answer. You should not use any arithmetic verilog operator, or direct instantiuation of hard multipliers, in your solution. Other approaches to building a multiplier that generates correct results and fit in the FPGA and clock cycle are considered legitimate successful designs.

You may want to generate a test verilog file or test waveform to verify that your multiplier does generate the correct answers in all cases. You might observe that the B input to `simplemult` very rarely changes, and assume that this is fixed to `0xA0` (160) and thus simplify your multiplier. You should not assume this, as we might adjust this parameter through the escape character. Adjusting the number of internal character lines would also be neccesary if we wanted to extend the pipeline to also handle a different output resolution.

Multipliers are generally more complex than adders, and it may be that you will not be able to do the full multiplication in a single clock cycle. You can find out if your design meets the timing requirement by using the post-place-and-route timing report. In this design, relaxing the clock so the multiplication does fit in one clock cycle is not possible, due to the operation frequency being a system constraint - we can't change the VGA spec because your multiplier is too large. If this happens, you must come up with a way to pipeline your multiplier, as well as extend the pipelining of the VGA and RS-232 logic to match.

Remember to show your working design to the lab demonstrator to get a signature.

6 Report

This practical requires that you hand in a report of your activities as well as the lab sheet. The report should outline the problem(s) you encountered and present your solution to them in detail. Make sure you address at least the following questions in the report:

- How does your multiplier and adder work in detail? Include logic/pipeline diagrams and a discussion on the merits of your chosen design(s) over other possible designs.
- Why is multiplication much 'harder' than addition? What about division and modulo operations - how long may they take?
- How did your design fare in terms of resource use in the FPGA, compared to the original design? How much extra logic did you use?
- What other blocks in this design would need to be replaced to make the design suitable for an ASIC implementation?

Obviously, the report should be in the normal format with an introduction, background, conclusion etc. You should also include a full report of the device usage as presented by the tools, which can help with the third question above. Hand the report in together with the lab sheet.

Pipeline Diagram

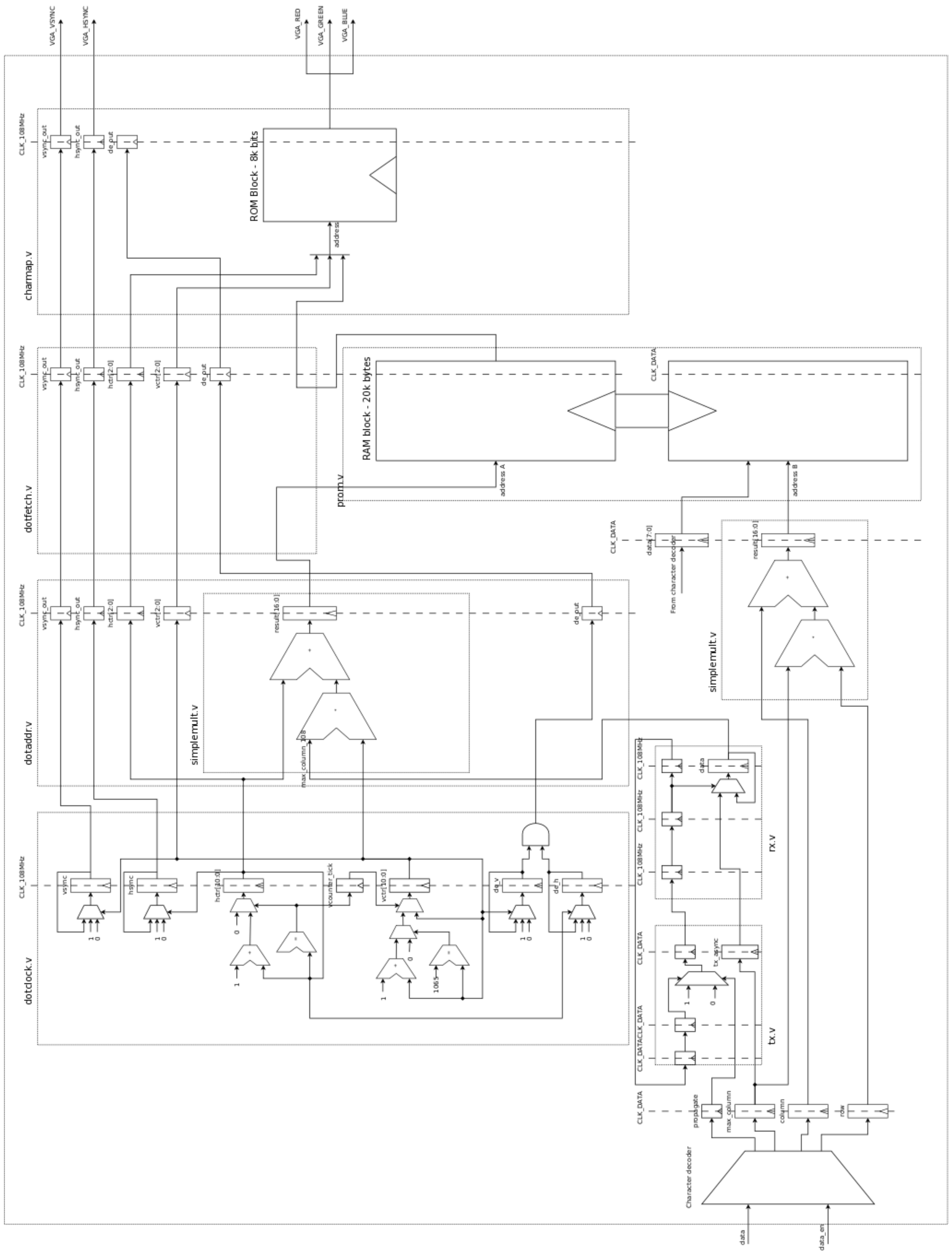

Figure 1: Logic element diagram of the display pipeline