Embedded Systems Lab – CSEE 4840

Programmable Video Processor with Predictable Timing

Julio A. Rios	jar2153
Sharmila Gupta	sg2337
Emil Nuñez	ern2101

Thursday, March 31, 2005

Overview

The main goal of this project is to design and implement a simple processor with predictable timing that can be used for hard real-time tasks such as video generation. The implementation will involve enhancing the MIPS [6] architecture, a RISC [8] microprocessor architecture, to allow for the stalling of the execution. A modified version of the MIPS architecture will be implemented on the FPGA. In addition, the standard MIPS assembly language will be enhanced with a new wait instruction.

The system design will be illustrated with diagrams. Namely, modified versions of the MIPS architecture schematic and state transition diagram will account for the new state in the FSM controller. The processor will be implemented as a VHDL circuit. The assembly language will be illustrated by our specific application program, NTSC signal generation of a text display. This language is basically the standard MIPS language with the enhancement of the wait instruction (which will be explained). A simple assembler will be provided, but there will be no high level language compiler. However, the functionality of the assembly code will be described by an equivalent C program.

Background

For hard real-time tasks, predictable timing is essential. For example, in generating NTSC [7] video signals, developers working with simple generic processors manually analyze the timing of their programs and insert NOP instructions to fill in idle time demanded by the standard. This approach yields software that is hardly robust. It is our goal to circumvent this problem by providing direct hardware support for real-time tasks in a simple specialized CPU that is programmable through software. We take the video generation problem as our specialized application and focus our efforts on developing a solution to that specific problem. However, we hope the design is simple enough to generalize to a wide array of real-time applications.

There have been other approaches to this problem. Namely, the problem of wasted cycles (NOP instructions) in video generation has been attacked at the compiler level by software thread integration (STI) in [2]. STI uses threads to provide real-time programming support for video processing on a generic CPU. Our approach is different in that we are trying to implement direct hardware support to make programmability simpler.

Xyron Semiconductor [9] is a commercial manufacturer of processors for real-time embedded systems. They also provide custom systems to their customers. However, we are not able to use their research as support, because it is not publicly available due to its commercial nature.

Another more general work has helped in the design of the architecture and organization of our system. Colnari et al. [1] is a broad reaching summary on implementing a hard-real time system, specifically for the embedded systems domain, from the hardware level to the operating system level to the assembler level.

DESIGN

First, we read the relevant literature to prepare to design a solution to the project. In addition, we studied the VHDL code for Professor Edwards' video controller for the Lab 5 assignment in his Embedded Systems Design [3]; this is the paradigm on which we base our design. Figure 1 shows an abstract view of the operation of the controller.

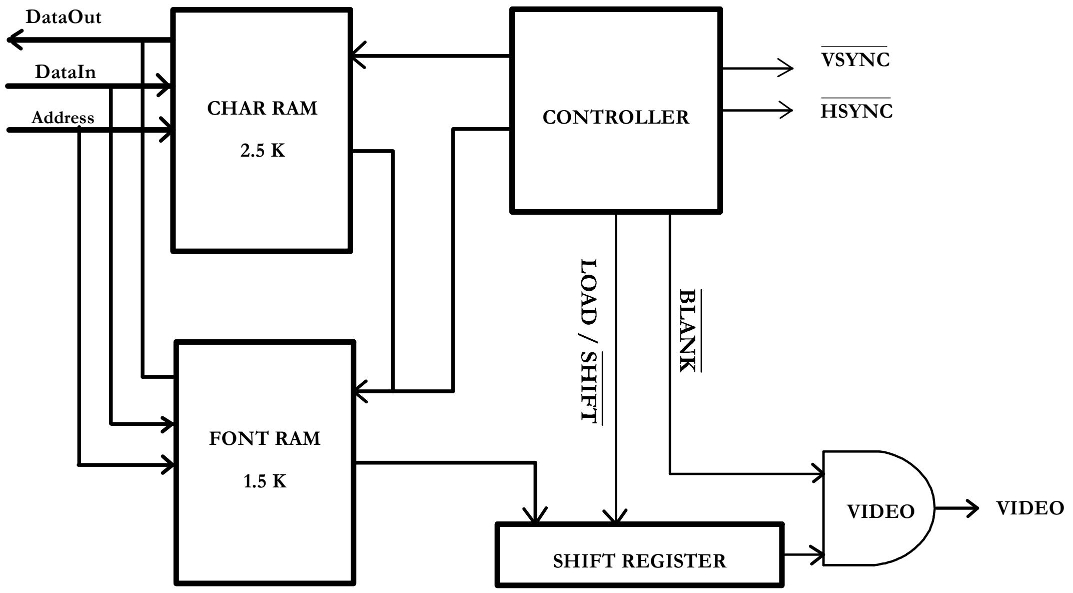
DataOut
DataIn
Address
CHAR RAM
2.5 K
CONTROLLER
$\overline{\text{VSYNC}}$
$\overline{\text{HSYNC}}$
LOAD / $\overline{\text{SHIFT}}$
$\overline{\text{BLANK}}$
FONT RAM
1.5 K
SHIFT REGISTER
VIDEO
VIDEO

Figure 1. NTSC video generation paradigm.

We converted the controller's operation to a C-type program, and subsequently, to simple assembly code. Whereas this video controller was implemented entirely in hardware described by VHDL, our controller will be mostly based in software written in assembly code, with a few hardware components. There will be special hardware support for a new wait instruction as part of the multicycle MIPS architecture. This will allow the programmer to implement the real-time behavior necessary for NTSC signal generation. There will also be a special shift register to feed video signal bytes one bit at a time to the video peripheral. The processor will communicate with the VGA port via an OPB (on-chip peripheral bus) controller. Figure 2 shows how the different components in the NTSC application will communicate with each other.

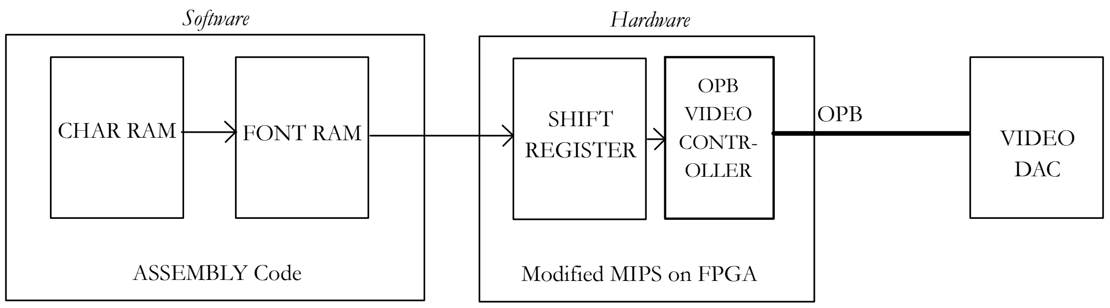

Figure 2. NTSC Application Block Diagram.

The problem is how to provide real-time support on a programmable processor. Our solution involves the addition of a wait instruction to the MIPS assembly language. This new instruction is designed to halt the execution of a program by a specified number of instruction cycles to provide precise real-time capability. To achieve this, a special type of register with a built-in down-counter will need to be added to the MIPS architecture. As a starting point, we shall provide four such registers to allow multiple wait conditions. This will suffice for our specific task of NTSC signal generation, but more may be added later to provide support for more complex real-time applications (to be explored later). The collection of four wait registers we will call the "Wait Register File".

Basically, the wait instruction will have the form:

wait [wait-register], [value]

For example (as in the NTSC case):

wait $w1, 8

First, the wait register file will check to see if the register being addressed is at its default value (zero) by means of checking a ready flag. If it is ready, then it will simply load the counter with the value provided, begin countdown, and continue with the next instruction. If, however, the register is not ready (i.e. has a value other than zero), it will reload the old value of the PC (of the wait instruction) and execute the same instruction again. This strategy may need to be changed for power optimization.

We have developed an abstract design of the computer architecture to implement the new enhanced instruction set. It also includes the special shift register and peripheral controller mentioned above to implement the communication link between the processor and the video converter. The shift register will be loaded with a value by issuing a store word (sw) instruction on a special memory location. Figure 3 shows a modified top level schematic of the data path of the MIPS processor. The control unit and control signals have been omitted for clarity.

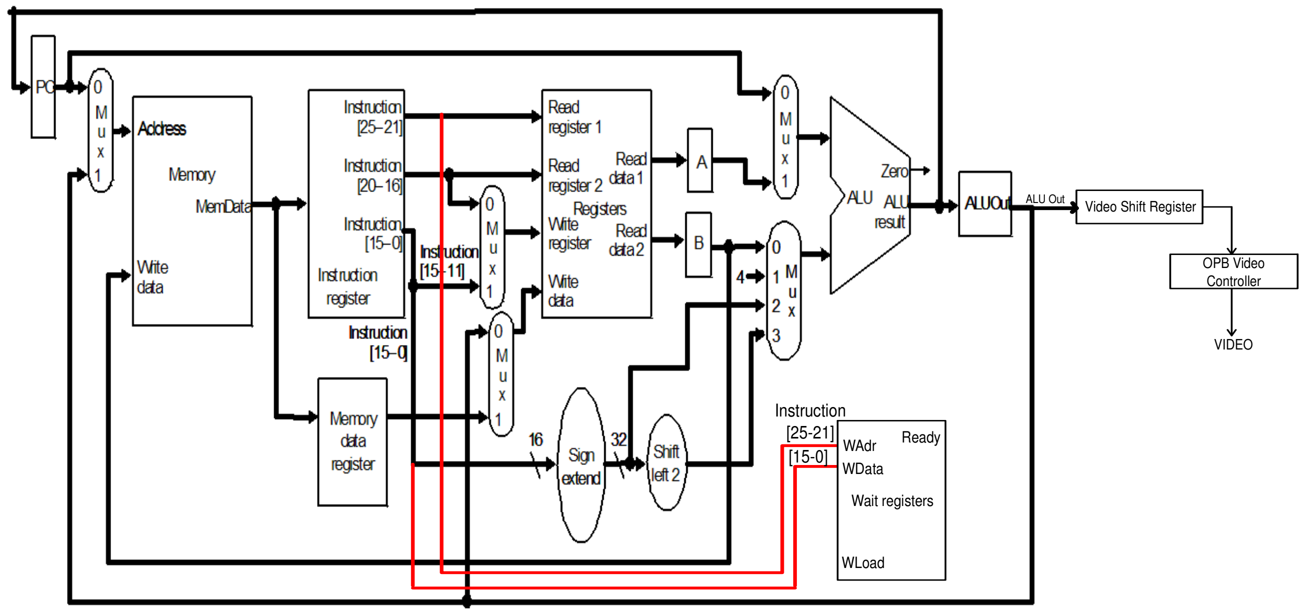

Figure 3. Top Level MIPS architecture schematic of data path.

Figure 4 shows a modified state transition diagram for the FSM controller. The design is still abstract and will likely be modified slightly during the implementation phase.

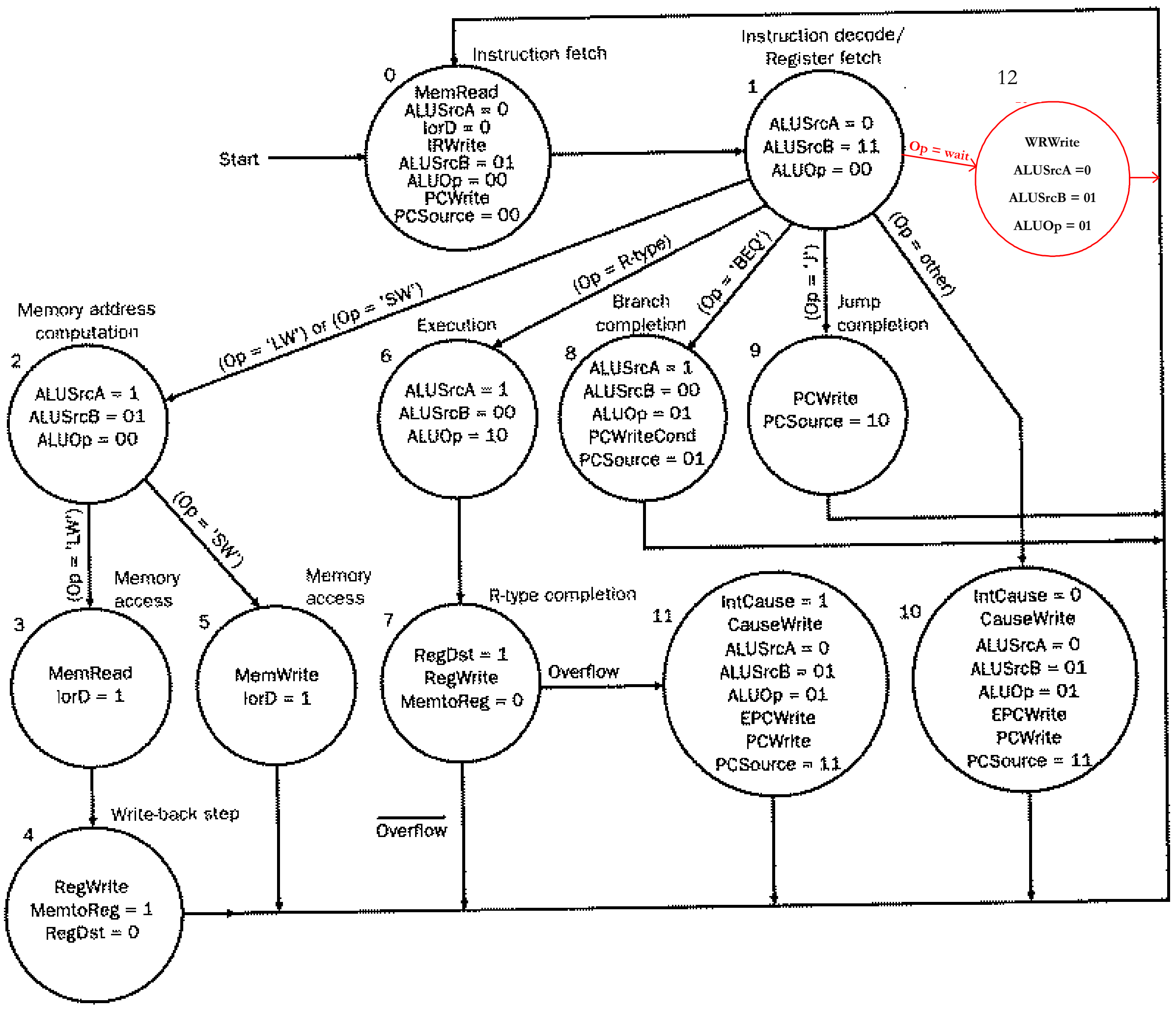

Figure 4. State transition diagram for MIPS finite state machine controller.

IMPLEMENTATION AND TESTING

This project is a collaboration of students of Stephen Edwards' Embedded Systems class and Luca Carloni's Distributed Concurrent Systems course. Namely, Sharmila Gupta and Emil Nunez of Professor Edwards' class will help in the implementation. Development and implementation of the hardware will be geared for the XESS XSB-300E FPGA board and we will use Xilinx software to synthesize and simulate our design, all of which is available in the workstations for the Embedded Systems class which run the latest version of Red Hat Linux. In addition, some auxiliary development will be done on Xilinx's Free ISE WebPACK 7.1i, which runs on MicrosoftWindows XP machines.

The simple MIPS architecture will be used as a starting point for this effort. We will use miniMIPS [5], a functional, open-source 32-bit implementation of MIPS. miniMIPS is written in VHDL and provides an assembler, gasm, written in C. The gasm assembler will, in turn, be modified to account for the new instruction. The VHDL code will be modified to provide enhanced functionality. Specifically, the wait register file and the special shift register will be implemented as VHDL modules. Also, an OPB controller will have to be implemented to serve as the interface between the processor and the video converter. We will use the OPB communication section of the video controller [5] as a starting point. Figure 5 shows the connection diagram for the video digital-to-analog converter. The OPB controller will be implemented accordingly.

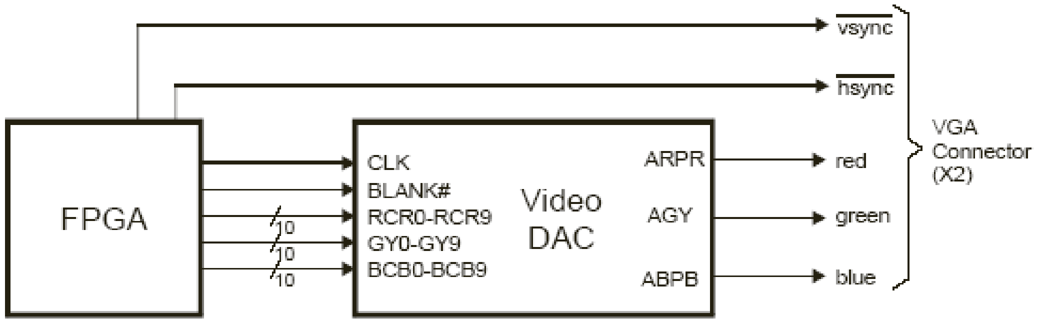

Figure 5. Video DA Converter connection to the FPGA on the XSB-300E board

It is our expectation that we will not need to use off-chip memory, like SRAM, but rather to use on-chip Block RAM. The standalone video controller used BRAMs to store the Character RAM and Font RAM, so we should be able to do the same. The assembly source, once assembled into bit code, should also fit on a few BRAMs.

The finished product should look like a multipurpose MIPS microprocessor programmed to behave like the video controller used in the Lab 5 assignment, hence exhibiting real-time capabilities. To test the implementation, it should be sufficient to write a subroutine to write to the location where the Character RAM is stored, and watch the LCD display change.

REFERENCES

[1] M. Colnari, D. Verber, R. Gumzej, and W. A. Halang. Implementation of hard real-time embedded control systems. Real-Time Syst., 14(3):293–310, 1998.

[2] A. Dean, S. Kanaujia, and B. Welch. Generate video using software thread integration. Circuit Cellar, 161:10–18, dec 2003.

[3] S. Edwards. Text-mode vga controller for the xess-300e. Not publicly available, 2005.

[4] M. Jacomet. RISC Processor Design. Available from http://www.microlab.ch/courses/risc/risc01.pdf, apr 1996

[5] H. Samuel, J. Sebastien, M. Louis-Marie, and S. Olivier. Minimips. http://www.opencores.org/cvsweb.shtml/minimips/.

[6] Wikipedia contributors. MIPS architecture. Wikipedia: The Free Encyclopedia. http://en.wikipedia.org/wiki/MIPS_architecture, Accessed 30 mar 2005.

[7] Wikipedia contributors. NTSC. Wikipedia: The Free Encyclopedia. http://en.wikipedia.org/wiki/NTSC_standard, Accessed 30 mar 2005.

[8] Wikipedia contributors. RISC. Wikipedia: The Free Encyclopedia. http://en.wikipedia.org/wiki/RISC, Accessed 30 mar 2005.

[9] Xyron Semiconductor. Freeing tomorrow from today's past. http://www.xyronsemi.com/.