
# Atari ST System-on-Chip in VHDL

Individual Project

UFEEJ4-40-3

Author: Lyndon Amsdon

Student Number: 05500164

Word Count

15116

# Table of Contents

# List of Illustrations

# List of Tables

# Chapter 1

# Preface

**1.1**

Within this personal project is a complete guide to the research, development, implementation and conclusions to creating a System on Chip, based on the Atari ST series of home microcomputer which spanned a production date of 1985-1993 [1]. The project is designed to be left open to continual work and extensions beyond the original Atari ST design.

This document assumes prior knowledge of Microsystems design and the Motorola MC68000 CPU, and going in depth into these topics is beyond the scope of this document.

# Introduction and Overview

**1.2**

This project is inspired by other enthusiast's attempts at creating systems on chip that faithfully reproduce early home microcomputers and arcade machines. The development in the last few years in programmable logic devices, with increased logic elements and low development costs, has meant it is possible to fit entire computers into one semiconductor.

**1.2.1**

**TV Boy**

One of the first commercially available products has to be the 'TV Boy', which was an unlicensed reverse engineered copy of the Atari 2600. It first went on sale around the mid 1990s. The original Atari 2600 used 4Kbyte game cartridges whereas the TV Boy used a


*Figure 1 - TV Boy*

512Kbyte ROM as storage for the 127 internal games [2]. A spare 4Kbyte slot was used as the game menu selection. As it was unlicensed by Atari the games had different names and some had very small changes to the graphical details. All the digital electronics were designed into a single ASIC, intended for mass production and low cost.

### 1.2.2

### Flashback 2

There have been some official licensed Atari consoles recreated in modern silicon. The Flashback 2 is another Atari 2600 with 40 games included. The design of the case is reminiscent of the original Atari 2600, but being somewhat smaller and lighter. The Flashback 2 was designed by Curt Vendel and Legacy Engineering, and in an interview Curt Vendel remarked that the "Flashback 2 did exceptionally well with 860,000 sold in the U.S./domestic" [3].

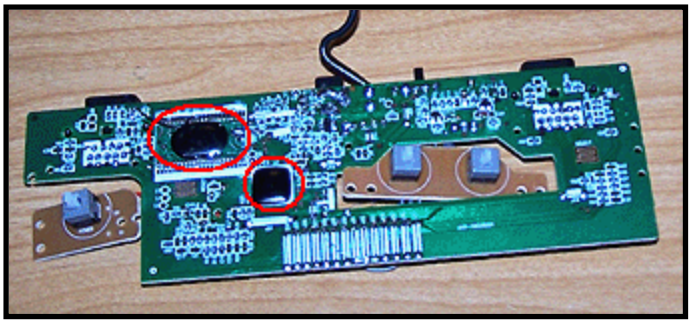


*Figure 2 - Flashback 2 Main PCB*

### 1.2.3

### NOAC

Another unlicensed reversed engineered copy of a console exists, based on Nintendo's NES (Nintendo Entertainment System). These are known as NOAC (Nintendo On A Chip) and originate from a variety of manufactures in China and are inaccurate in many ways to an original NES [4]. The Integrated Circuit is supplied without a real physical package, instead being covered with an epoxy glue material.

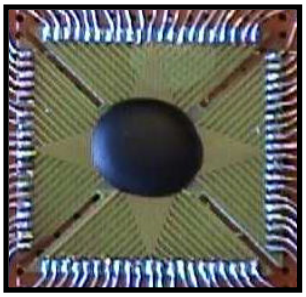


*Figure 3 - NOAC SOC*

A brief look on the internet at current and past projects in this particular field has shown the following popular home computers being implemented into FPGAs.

- Msx Bazix – MSX (Japanese home computer)
- Minimig – Amiga A500
- Suska – Atari ST/STE
- C-One – Reconfigurable Commodore 64 & Commodore VIC-20

### 1.2.4
### MSX Bazix

The MSX Bazix [5] was a project led to create primarily a clone of the technically advanced MSX home computer, which was very popular in Japan in the 1980s. The MSX Bazix was also designed to pave the way for other developers to create projects on, with the design of the hardware being open source and an array of I/O ports to cater for most needs. It's future and success is unknown with no news on their website for over 2 years.

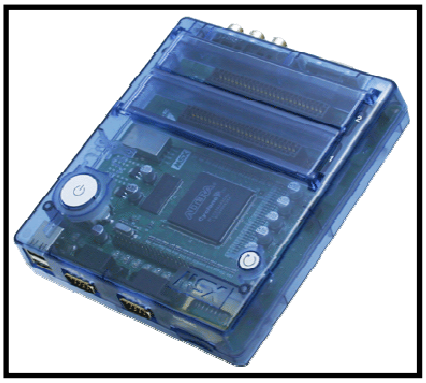


*Figure 4 - MSX Bazix Unit*

### 1.2.5
### Minimig

The Minimig (short for Mini Amiga) is based around a Xilinx FPGA and MC68SEC000 CPU. It has some key changes from the original Amiga 500, including support for a PS/2 mouse and keyboard and games that load from a removable MMC Flash memory device [6]. The source code for both the FPGA and PIC microcontroller became available to download on 24/07/2007 and the hardware is available to buy through online resellers.

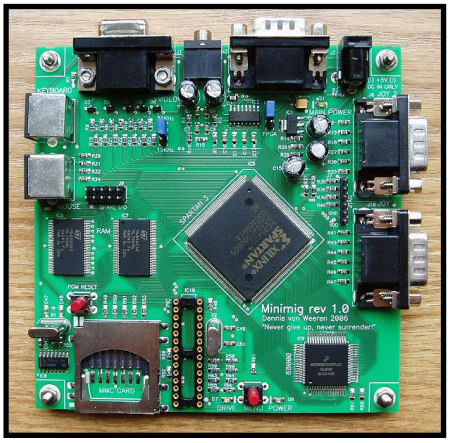



*Figure 5 - Minimig PCB*

### 1.2.6

### C-One/C64DTV

Many other exist, at various stages of completion. What are more interesting are the results of some of these projects. The best example of this is the C-One. The C-one was designed by Jeri Ellsworth in 2002, to replicate a Commodore 64 using an Altera FPGA [7]. By 2004 a marketing company had approached Jeri Ellsworth to use the design in a low cost hand held console to plug directly into a TV, the result being the C64DTV**.**

The C64DTV hardware is all based on an ASIC, or Application Specific Integrated Circuit, which is like a fixed design FPGA. These are commonly used in mass produced products. The software comprised of 30 games, originally produced for the Commodore 64 in the mid to late eighties and licensed to be used. The C64DTV was very successful on release, selling 70,000 units in a single day via a TV shopping channel priced around £20[8].


*Figure 6 - C64DTV*

# Chapter 2

# In Depth Introduction

**2.1**

Atari was founded in 1972 by Nolan Bushnell and Ted Dabney firstly creating arcade games, and then moving onto home computers and home video game consoles. Atari at the time had created groundbreaking games like Pong, and also designed an affordable 8 bit home video game console, called the 2600 based around the Motorola 6502 CPU.

By 1976 Atari was sold to Time-Warner and work had started on a replacement for the 2600 video game console. A shift towards people wanting to do more than play games meant the next computers, the Atari 800 & 400, had keyboards and the term 'home computers' arrived.

In the early eights there was the crash of the US Video Games industry, where many companies producing video game consoles and home computers in North America either went bankrupt or lost a lot of money. Some of the reasons for this were too much competition, a flood of poor software titles and not enough compatibility between consoles, even ones made by the same manufactures.

It was then in 1984 Atari was sold by Time Warner to Jack Tramiel, who was the founder of Commodore. Atari was restructured selling off old stock at reduced prices to fund a new home computer, which would be called the Atari ST and was released in 1985 [9],[10].

**2.1.1**

**Atari ST Models**

The Atari ST was particularly strong in the music industry, with MIDI (Musical Instrument Digital Interface) ports being built in. One of the video modes, being monochrome high resolution (for the time) also meant the Atari ST found its way in DTP (Desktop Publishing) and CAD (Computer Aided Design).

The Atari ST stands for Sixteen/Thirty-two, as it was based around the powerful Motorola MC68000 which had a 16bit external data bus, but internal 32bit registers.

The Atari ST range came in quite a few different flavours [10].

| | |
|---|---|
| ST | Original |
| STM | RF modulator for TV output |
| STF | Internal floppy drive |
| STFM | RF Modulator for TV output, internal floppy drive |
| STE | DMA Sound, Blitter chip, enhanced graphics, RF Modulator, internal floppy drive |
| Mega ST | Detachable keyboard, Blitter Chip, internal floppy drive, internal expansion bus |
| Mega STE | Detachable keyboard, Blitter Chip, internal floppy drive, internal VME expansion bus, optional FPU, 16 MHz CPU with L2 Cache |
| Stacy | Portable Laptop version, internal floppy drive, monochrome 9” LCD screen |

*Table 1. List of different models Atari produced based around the original ST hardware*

## Atari ST Hardware

**2.2**

The original prototype of the Atari ST was built by hand using discrete TTL logic devices using wire wrapping and prototyping printed circuit boards. These were then integrated into four custom ASICs on the production models [11].

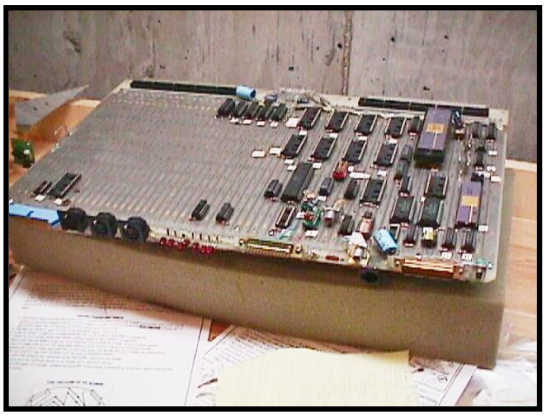


*Figure 7 - Atari ST CPU Board Prototype*

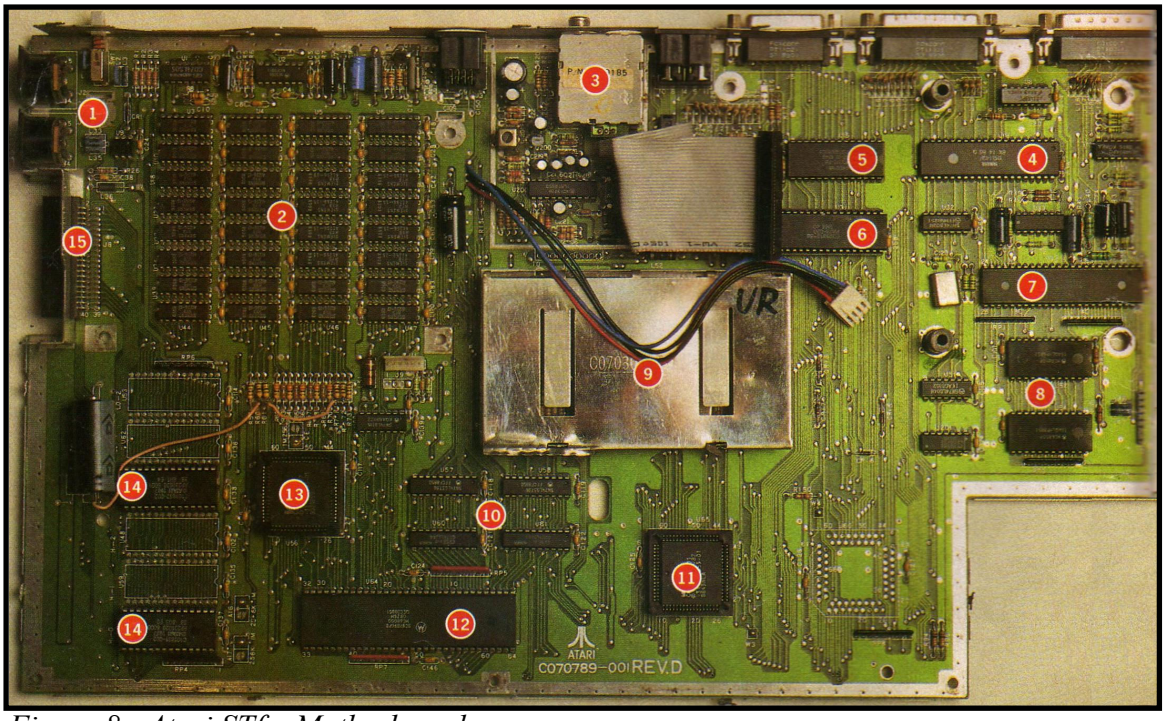


*Figure 8 - Atari STfm Motherboard*

1. Reset circuitry consisting of NE555 monostable
2. FPM DRAM, consisting of two banks of 512kbytes
3. RF Modulator to convert composite video and audio to RF
4. Custom DMA chip
5. Western Digital WD1772 Floppy Disk Controller
6. Yamaha YM2149 Sound Chip
7. Motorola MC68901 MFP
8. Motorola MC6850 ACIA one for keyboard interface and another for MIDI
9. Custom SHIFTER Video chip inside shielded enclosure
10. Bus transceivers to bridge between Data Bus and RAM Data Bus
11. Custom GLUE chip
12. Motorola MC68000 CPU
13. Custom MMU chip
14. EPROM's containing TOS (The Operating System)
15. Cartridge Port for additional EPROM's

### 2.2.1
### MC68000 CPU

The Atari ST was built around the Motorola 68000 CPU. Some of the MC68000 features are listed below.

- 16 bit data bus
- 24 bit address bus
- Asynchronous bus cycles (to allow for wait states)
- Synchronous bus cycles to interface to older 8 bit 6800 peripherals
- 32 bit internal registers
- 7 Interrupt levels
- Byte, Word and Long data transfers

Listed below are the four custom integrated circuits, which are all closely linked together, and to operate rely on each other.

### 2.2.2
### GLUE

As the name suggests, this IC glues the system together. It is responsible for address decoding and providing chip select lines. It also handles the control of interrupt lines to the CPU, and bus arbitration between CPU and DMA. It also creates the video timing signals.

### 2.2.3
### MMU

This integrated circuit controls the Dynamic RAM signals. It is not as powerful as the name suggests, it doesn't do any memory protection, translation from virtual to physical address or paging. This would be better called a Memory Controller Unit. It multiplexes the CPU address lines to Column and Rows. It also contains a counter for sending video data from RAM to the SHIFTER and also a counter for DMA transfers.

### 2.2.4

### SHIFTER

This integrated circuit takes the data supplied by the MMU and uses a lookup table to display the colour from a palette. All the Atari ST video modes are based on bit planes. There are 3 video modes, 320x200 16 colours (4 bit planes), 320x400 4 colours (2 bit planes) and 640x400 (1 bit plane). The reason for using this method was because the memory bandwidth is not enough to support "chunky" graphic modes where each byte represents a pixel on the screen.

### 2.2.5

### DMA

The DMA (Direct Memory Access) controller is responsible for transferring chunks of data between the RAM and DMA port, which is used for connection of hard drives. It also resizes the 16 bit data bus to the external 8 bit bus featured on the DMA port. It is also used to carry out DMA transfers to and from the Western Digital WD1772 FDC (Floppy disk controller).

### 2.2.6

### MFP

The MFP is a MC68901 manufactured by Motorola and is an abbreviation for Multi Function Peripheral Chip. In the Atari ST it is used to provide a RS232 serial port. It also serves as an interrupt controller, allowing more interrupt sources than the Motorola MC68000 CPU provides. It also contains four universal timers.

### 2.2.7

### YM2149

The YM2149 is manufactured by Yamaha and is primarily the sound generator. It contains 3 independent tone generators. It also has two general purpose 8 bit data ports. In the Atari ST these are used for the Centronics printer interface and the other is used to help control the floppy disk and RS232 hardware flow control.

### 2.2.8

**ACIA**

The ACIA is an abbreviation for Asynchronous Communications Interface Adapter. The Atari ST contains two MC6850. Their task is to serialize data to communicate with the Keyboard and MIDI devices. They were designed as a peripheral chip to the MC6800 processor, and so they only feature an 8 bit wide bus and use the legacy synchronous bus that the MC68000 CPU can offer.

**2.2.9**

**FDC**

The FDC is an abbreviation for Floppy Disk Controller. It is a WD1772 made by Western Digital. It is connected to the DMA chip so that all transfers are via DMA relieving the CPU from disk transfers. It contains the logic for precise timing of the floppy disk drive heads and motors and sterilization of data.

## Atari ST Operating System

**2.3**

After the hardware came close to being completed an operating system was needed. Atari decided to use a new operating system with a GUI (Graphical User Interface) from Digital Research, providing a WIMP (Windows, Icons, Menu, Pointing Device) environment, much like the Apple Macintosh. This was essentially a port from the Intel 8088 version they had developed for the IBM compatible machines. The operating system was called TOS (The Operating System), and provided the programmer with many system calls by using the TRAP software exception calls. In TOS there are three layers, called the BIOS, XBIOS and GEMDOS. The BIOS and XBIOS are hardware dependant, while the GEMDOS layer is hardware independent. The operating system on very early Atari ST models came on floppy disk, but the more common later versions placed this operating system on PROM memory devices [12].

With this project being mainly hardware based only the low level parts of the operating system, and particularly how the operating system starts up and boot straps.

### 2.3.1

### Atari ST Boot Up Operation

The Motorola 68000 on boot up requires initial values to load into its supervisor stack pointer and reset vector address. These come in the form of two long words at address 0x000000 to 0x000007.

Figure 9 shows the path the operating system takes on boot up. It was drawn from the reverse engineered commented source code [13],[14].

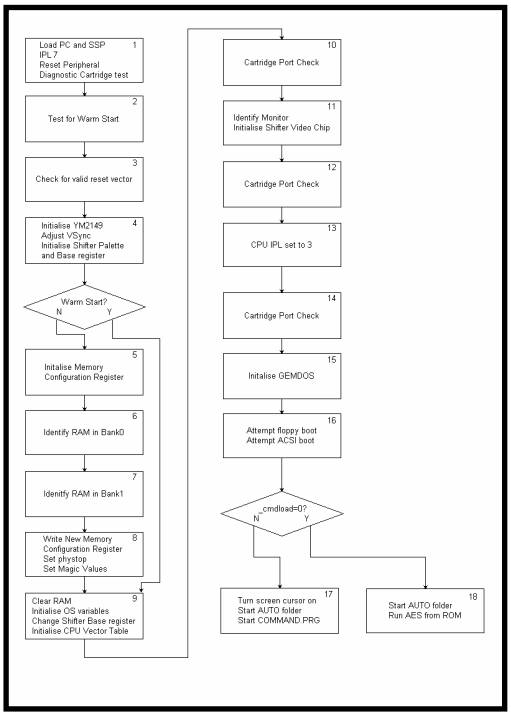



*Figure 9 - Atari ST boot up sequence*

Boot up sequence

(1)

- Load SSP with long word value from 0xFC0000.
- Load PC with long word value from 0xFC0004 (Garbage value, memory not yet sized).
- CPU Supervisor Mode Interrupts disabled (IPL=7).
- RESET instruction to reset all peripheral chips.
- Check for magic number 0xFA52235F on cartridge port, if present jump to diagnostic cartridge.

(2).

- Test for warm start, if memvalid (0x000420) and memval2 (0x00043A) contain the Magic numbers 0x7520191F3 and 0x237698AA respectively, then load the memconf (0xFF8001) contents with data from memctrl (0x000424).

(3)

- If the resvalid (0x000426) contains the Magic number 0x31415926, jump to reset vector taken from Resvector (0x00042A).

(4)

- YM2149 sound chip initialized (Floppy deselected).
- The vertical synchronization frequency in syncmode (0xFF820A) is adjusted to 50Hz or 60Hz depending on region.
- Shifter palette initialized.
- Shifter Base register (0xFF8201 and 0xFF8203) are initialized to 0x010000.
- The following steps 5 to 8 are only done on a coldstart to initialize memory.

(5)

- Write 0x000a (2 Mbyte & 2 Mbyte) to the MMU Memory Configuration Register 0xff8001).

(6)

- Write Pattern to 0x000000 - 0x000lff.
- Read Pattern from 0x000200 - 0x0003ff.
- If Match then Bank0 contains 128 Kbyte; goto step 7.
- Read Pattern from 0x000400 - 0x0005ff.
- If Match then Bank0 contains 512 Kbyte; goto step 7.
- Read Pattern from 0x000000 - 0x0001ff.
- If Match then Bank0 contains 2 Mbyte; goto step 7.
- panic: RAM error in Bank0.

(7)

- Write Pattern to 0x200000 - 0x200lff.
- Read Pattern from 0x200200 - 0x2003ff.
- If Match then Bank1 contains 128 Kbyte; goto step 8.
- Read Pattern from 0x200400 - 0x2005ff.
- If Match then Bank1 contains 512 Kbyte; goto step 8.
- Read Pattern from 0x200000 - 0x2001ff.
- If Match then Bank1 contains 2 Mbyte; goto step 8.
- note: Bank1 not fitted.

(8)

- Write Configuration to MMU Memory Configuration Register (0xff8001).
- Note Total Memory Size (Top of RAM) for future reference in phystop (0x00042E).
- Set magic values in memvalid (0x000420) and memval2 (0x00043A).

(9)

- Clear the first 64 Kbytes of RAM from top of operating system variables (0x00093A) to Shifter base address (0x010000).
- Initialize operating system variables.
- Change and locate Shifter Base register to 32768 bytes from top of physical ram.
- Initialize interrupt CPU vector table.

- Initialize BIOS.
- Initialize MFP.

(10)

- Cartridge port checked, if software with bit 2 set in CA_INIT then start.

(11)

- Identify type of monitor attached for mode of operation for the Shifter video chip and initialize.

(12)

- Cartridge port checked, if software with CA_INIT clear (execute prior to display memory and interrupt vector initialization) then start.

(13)

- CPU Interrupt level (IPL) lowered to 3 (HBlank interrupts remain masked).

(14)

- Cartridge port checked, if software with bit 1 set in CA_INIT (Execute prior to GEMDOS initialization) then start.

(15)

- The GEMDOS Initialization routines are completed.

(16)

- Attempt boot from floppy disk if operating system variable _bootdev (0x000446) smaller than 2 (for floppy disks) is. Before a boot attempt is made bit 3 in CA_INIT (Execute prior to boot disk) checked, if set, start cartridge.
- The ACSI Bus is examined for devices, if successful search and load boot sector.

- If system variable _cmdload (0x000482) is 0x0000, skip step 17.

(17)

- Turn screen cursor on
- Start any program in AUTO folder of boot device
- Start COMMAND.PRG for a shell

(18)

- Start any program in AUTO folder of boot device
- AES (in the ROM) starts.

# Chapter 3

# Research

### 3.1

### FPGA

An FPGA is a programmable logic device, with the configuration being volatile. The FPGA contains many complex logic blocks that have interconnects running between them in a grid like fashion. There are also dedicated interconnects like global clock lines. The configuration is often programmed in a high level HDL (Hardware descriptive Language) like Verilog or VHDL, or sometimes as a schematic. The majority of modern FPGAs contain embedded functions, such as adders, multipliers, memory, digital PLLs and even DSP cores. There has been a recent trend in pushing soft core processors into designs for FPGA creating complete systems on chip that can be fine tuned for specific tasks [15].

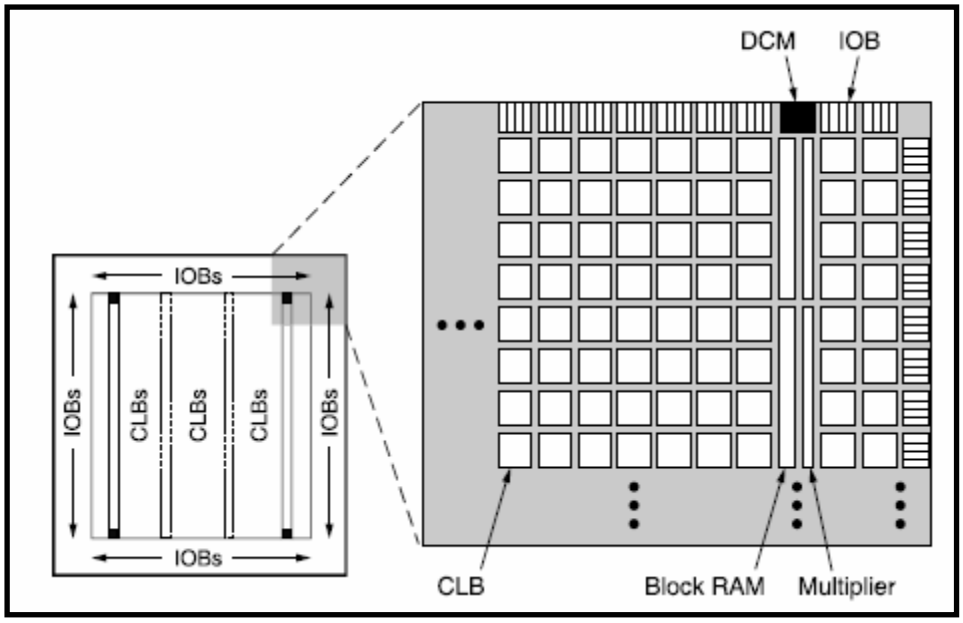



*Figure 10 –Xilinx FPGA block layout*

**3.2**

**Base Hardware**

The design is to be based around a Xilinx FPGAs, as there are special free versions of the IDE (Integrated Development Environment) which are only slightly limited from the commercial versions. The type of Xilinx fitted to the board needs to be large enough (in terms of logic elements) to fit the whole project, which is not something that can be estimated easily. The Atari ST is based on a 5 volt logic platform, and so having some 5 volt capabilities on the chosen development board will be a real bonus.

| **Board** | **FPGA** | **I/O's** | **Notes** | **Price** |
|---|---|---|---|---|
| Xilinx Spartan-3 Starter Kit | XC3S200 | 100 | Programming Cable, PS/2 ports, VGA | £80 plus shipping and customs |
| Enterpoint Raggedstone | XC3S1500 | 120 plus 50 5v tolerant on PCI header | Programming Cable, 7 segment display | £120 inc shipping (special student price) |
| Inrevium TB-3S-1400A-IMG | XC3S1400A | 128 | 4Mbyte DDR SDRAM, RS232 | £650 plus shipping and customs |
| Philips PXPDKSP3 | XC3S1000 | 80 | PCI Express Bridge, Prototype Area. | £700 plus shipping and customs |
| Digilent Inc. Nexys-2 | XC3S500E | 59 | Programming Cable, ps2 ports, vga, SDRAM | £50 plus shipping and customs |

*Table 2 - Comparison of some of the available Xilinx development boards*

### 3.3

### IP Cores

IP Core stands for Intellectual Property Core. They are a block of logic as an element to design reuse, a trend towards repeated use of previously designed components. IP cores may be licensed to another party or can also be owned and used by a single party alone. Some cores are only offered as netlists, to protect the vendor against reverse-engineering. Others are offered as synthesizable cores in hardware descriptive languages like Verilog or VHDL [16].

There are already a couple of projects for putting an Atari ST inside an FPGA, thankfully both in the VHDL language. There is MikeJ's project, although only the source to his YM2149 Sound Chip is available. There is also Wolfgang Forester's project, which includes an IP Core of every Atari ST semiconductor.

### 3.4

### Software Suite

The Xilinx IDE comes in two flavours, ISE Foundation and ISE Webpack where the latter is a free version. The free ISE WebPack is only restricted in the devices it supports, and that is generally the newest or largest devices like the Virtex 5 SXT family [17]. There are a range of tools included like Simulators, Timing Analysers and Power Analysis. There are additional options that can be bought for some of the more advanced features like ChipScope (FPGA probe) and Modelsim (Powerful Simulator).

### 3.5

### Processor

The processor can either be an IP Core or real genuine Motorola (now Freescale) 68000. At the time of writing no free 68000 IP Core is available that has been tested and verified. There are a few different incarnations of the 68000 to help keep it up to date as production has spanned almost 30 years now [18].

| Model | Technology | Voltage | Details | Manufactured |
|---|---|---|---|---|
| 68000 | NMOS | 5v | Original | No |
| 68HC000 | CMOS | 5v | Low Power | Yes |
| 68HC001 | CMOS | 5v | Low Power, 8/16bit data bus | Yes |
| 68EC000 | CMOS | 5v | Embedded version, 8/16bit data bus | No |
| 68SEC000 | CMOS | 3.3v | Embedded version, 8/16bit data bus, static clock | Yes |
| 68008 | NMOS | 5v | 8bit data bus, 20/22bit address bus | No |
| 68010 | NMOS | 5v | Virtual machine & virtual memory instructions | No |

*Table 3 - Comparison of 68000 family processors*

**3.6**

**Books and literature**


- Atari ST Internals ISBN : 0-916439-46-1
- Atari ST Profibuch ISBN : 3-88745-563-0
- 68000 Microsystems Design ISBN : 0-534-94822-7

- MC68000 Hardware Datasheet
- MC68000 Programmers Reference Manual

## 3.7
## Debugging

There are many ways to debug and fault find hardware. These range from the very basic up to monitoring registers in a CPU and data flow.

A set of LEDs can be used to check that an FPGA has been programmed correctly. By using a clock signal and diving it down to a signal of a one or two Hz this can be used to drive an LED and make it flash. Another use of LEDs is to show the status of signals, like a reset line or processor state. They are very often the first thing to get working when starting on a new development board.

A 7 segment display can be used much in the same way as a single LED but allowing display of whole bytes, words of even long words if enough segments are available. A bit more functionality is need in an FPGA to achieve this as quite often 7 segment displays need to be scanned one segment at a time at a fast enough rate for the eye not to see any flicker.

Single Stepping is a way of stepping through the boot up code of a board, one instruction at a time. It will usually be used in combination with a method to display bus signals to verify or diagnose a problem with the board. Using this method needs hardware than can support halting the system.

Xilinx Chipscope and Altera SignalTap are pieces of software to view any internal signal of an FPGA. They manage this by using the JTAG interface and modifying the FPGA bitstream with some additional logic.

A Monitor program is a utility that is loaded from ROM into an available processor. It is designed to use little or no resources so it can run when some hardware isn't fully functioning. It usually communicates over a simple RS232 implementation and allows

the user to write small assembling programs. These can be used to test various parts of a system.

### 3.8

### Operating System Versions and storage

The operating system for the Atari ST went through various versions from TOS 1.00 to TOS 2.06 [19].

| Version | Date | Computer | Details |
|---|---|---|---|
| 1.00 | 20th November 1985 | ST | Original Version |
| 1.02 | 22nd April 1987 | ST, Mega ST | Mega ST Blitter & RTC support |
| 1.04 | 6th April 1989 | ST, Mega ST, Stacy | Bug Fixes, faster disk I/O |
| 1.60 | Unknown | STE | Support for STE hardware |
| 1.62 | 1st January 1990 | STE | Bug Fixes |
| 2.05 | Unknown | Mega STE | Support for Mega STE hardware |
| 2.06 | 14th November 1991 | Mega STE | Features added to GUI, support for all ST range |

*Table 4 - Comparison of different TOS versions*

The Operating System on the Atari ST is stored in PROMs which are rather out dated these days and not ideal for early stages of design. Non Volatile Flash memory is now the norm and is being used as a replacement to PROM, many of the FPGA development boards contain some Flash memory.

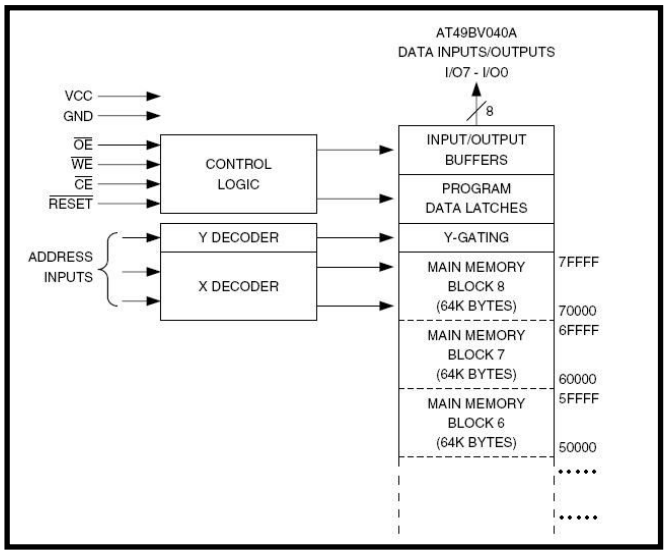



*Figure 11 – Flash memory organisation*

Reading from Flash is the same as a PROM, but writing to Flash takes a little more work. Before writing to Flash memory command sequences need to be issued. Also the data in the flash is organised into Blocks as shown in figure x.

From the Atmel data sheet for the AT49BV040A 4-megabit flash memory chip it's also worth noting that it is not possible to write bits that are currently 0s back to 1s, only erase commands can do that. Below is the list of commands that the Atmel Flash memory uses.

- Read
- Full Chip erase
- Sector erase (block erase)
- Byte program (Write byte)
- Boot block lockout
- Product ID entry
- Product ID Exit

**3.9**

**System memory**

There are many types of RAM available, but they can be split into two types depending on the technology used to store the data. Dynamic RAM uses capacitance to store a charge representing a bit of data, therefore it needs to be refreshed periodically. Static RAM uses flip flops, and thus need more logic per data bit of storage [20].

Fast Page Mode (FPM) DRAM is the type of memory fitted to the Atari ST. A row address only needs to be sent once, for many accesses to adjacent memory locations. They are only commonly available in 5v and usually come in a package called a SIMM with either 30 or 72 pins providing 8 bits or 32 bits respectively.

EDO DRAM is essentially the same as FPM, except that the timing has altered slightly for a small access time improvement. They are available in 5v and 3.3v and usually come packaged in a 32 bit wide 72 pin SIMM.

Synchronous Dynamic RAM (SDRAM) was the first type of synchronous ram, spawning many newer types like Double Data Rate (DDR) SDRAM. Data transfers are synchronised to the system clock. To access the SDRAM commands are issued to be executed. Due to their command structure and high clock speed (66Mhz and above) they are inherently more difficult to interface to. They are available in 5v and more commonly 3.3v and usually come packaged in a 64 bit wide 168 pin DIMM.

Static RAM (SRAM) is quite different from dynamic memory. Rather than using capacitors to hold a charge to represent a state of a bit it uses flip flops. This also means that it does not need the usual periodic refresh that dynamic RAM needs. It is also addressed by its full address width in one transaction, the column and row decoding is done internally. The disadvantage of static RAM is the cost. It is generally faster than dynamic RAM and so is often used for cache memory.

**3.10**

**Serial Port**

An RS232 serial port can have many uses from debugging, transferring data from a host computer and communications. In the Atari ST it was primarily used for communications with Modem's. Interfacing a serial port to an FPGA is quite simple, the voltage levels of RS232 swing from -12 to +12v so a voltage level translator is needed like a Maxim MAX232. The software overheads are very small which is why serial is still favoured over USB and other communication buses. Serial Ports can be implemented by only three wires, ground, transmit and receive.

**3.11**

**Video Output**

Conversions for video from digital to analogue are usually done by one of two methods, a specialist Video DAC or an inexpensive resistor ladder. An example of the resistor ladder was found in the schematics of the Xilinx Spartan 3A start kit.

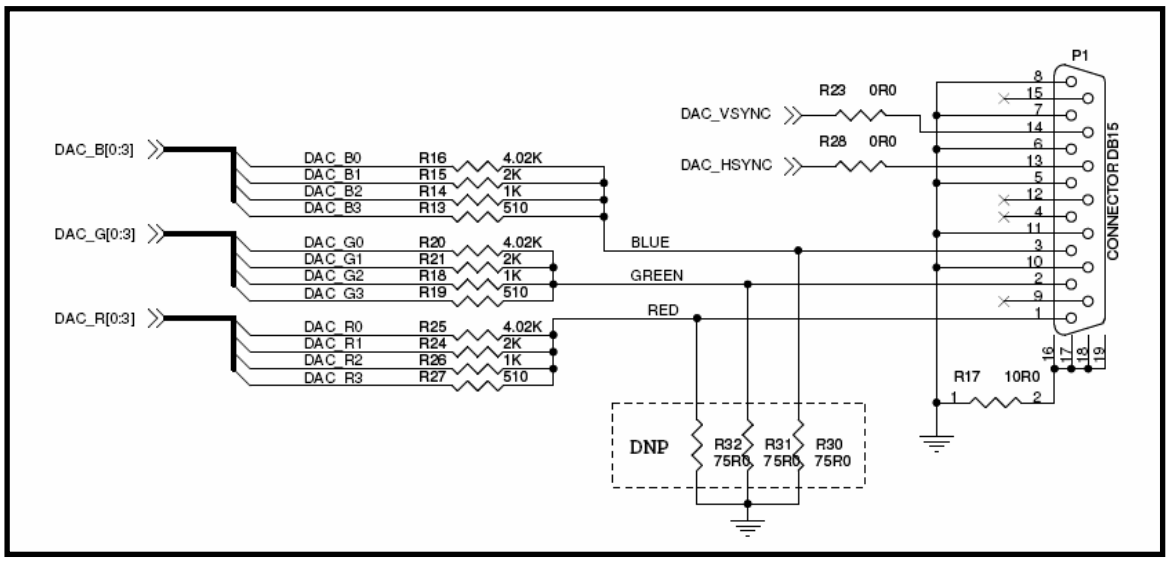



*Figure 12 – Digital to Analogue using resistor ladders*

The resistor ladder is easy to implement and is inexpensive but suffers from bad picture quality especially when used at higher resolutions, requiring higher video bandwidth. The resistors used in the schematic above are non standard values that appear in the E48 and onwards range of resistor values. They need to be of good quality and high degree of tolerance, but are still susceptible to drifting in value with temperature. The video intensity will also change depending on the load that the resistor ladders are driving into.

An example of a Video DAC was found in the schematics of the Xilinx Spartan-3 PCI Express Starter Kit. It uses a Philips TDA8777 Video DAC and although requires little external circuitry, it does cost more than the resistor ladder. It has a maximum conversion frequency of 330 MHz. It also helps to protect the FPGA from possible electrical damage, as it is bad practice to use non buffered FPGA signals onto external ports or connectors.

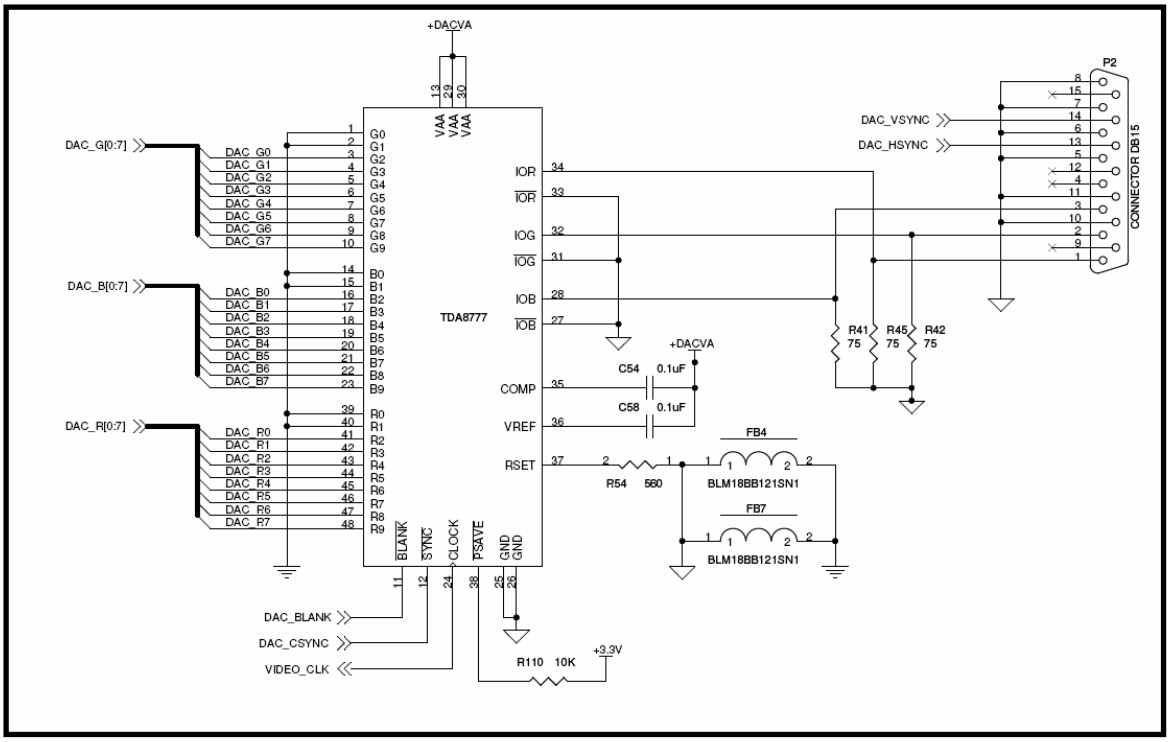



*Figure 13 – Digital to Analogue using specialised DAC*

# Chapter 4

# Design

### 4.1

### Components

This section describes the components chosen and how they will interconnect with each other. The components chosen have been based on the previous research and on availability.

### 4.1.1

### Base Hardware

It has been decided to use the Enterpoint RaggedStone development board. Their reduced student price, large FPGA, and plentiful I/O including optional 5v I/O header will be ideal. The RaggedStone was also designed to accept plug in daughterboard modules, one on each end of the board.

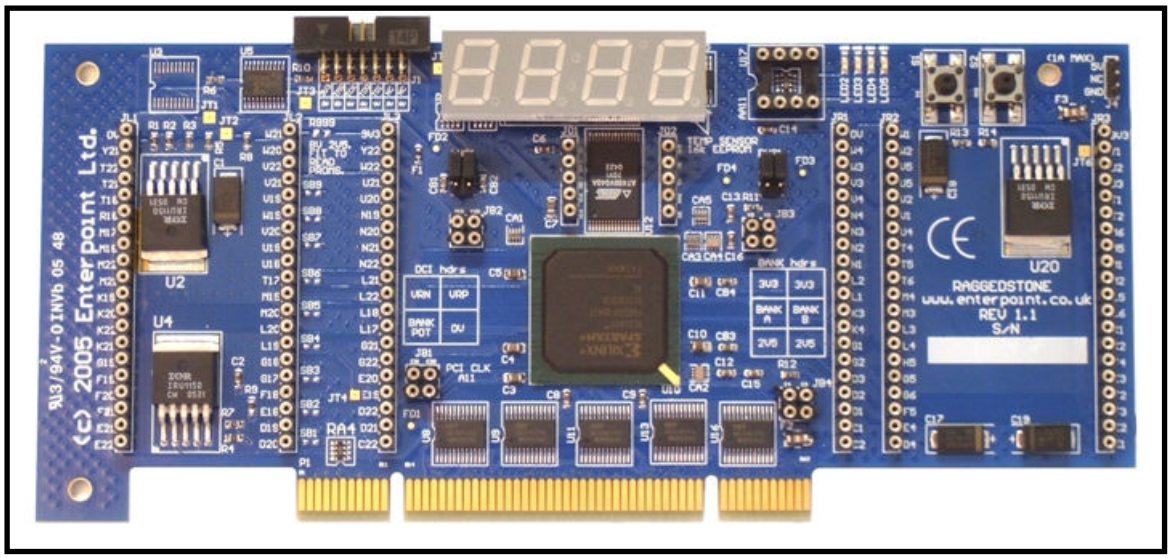



*Figure 14 – Enterpoint RaggedStone FPGA development board*

The board has the following features:

- 4 Digit 7 Segment Display
- 4Mbit Flash memory (524288 x 8) Atmel AT49BV040A
- 16k Serial EEPROM
- Temperature Sensor
- Oscillator Socket
- 4 LED's and 2 momentary push switches
- Voltage selectors for modules and associated FPGA bank (3.3v and 2.5v)
- Self resetting Poly fuses

**4.1.2**

**Processor**

The CPU chosen is the MC68SEC000, purely because it is the only version that is 3.3v. It is object-code compatible with the MC68000 but not entirely hardware compatible.

Bus Arbitration (a method for allowing other devices on the system bus to take control) is handled with a 2 wire protocol, instead of the original overly complex 3 wire protocol. The differences are covered in depth in the MC68000 datasheet.

The $\overline{\text{MODE}}$ pin selects 8 or 16 bit data bus operation, and is sampled at reset.

Support for legacy MC6800 synchronous peripherals has been completely removed. The missing signals are the E Clock, $\overline{\text{VPA}}$ (Valid Peripheral Address) and $\overline{\text{VMA}}$ (Valid Memory Address). A VHDL component will replicate these signals, creating a synchronous bus from the more commonly used asynchronous MC68000 bus [21].

The processor will fit onto a daughterboard installed on the RHS (Right Hand Side) I/O pins. One of the momentary push switches will act as the system reset.

### 4.1.3

**System memory**

The memory chosen is SRAM (Static RAM), because it simplifies a design and is available in a variety of voltages. The ability to use it without refreshing means it is great for prototypes or in debugging situations, as the whole design can be halted without loosing the contents of the memory.

The memory will fit onto a daughterboard installed on the LHS (Left Hand Side) I/O pins.

Using SRAM for the memory will be transparent to the user and all software, and will not create any problems.

### 4.1.4

**Debugging**

The following features will aid in debugging the system

A 7 segment display will show the current status of the CPU data and address bus. As the display can only show a maximum of 4 hexadecimal characters, the display will scroll.

One of the onboard switches will be used to step through the operating system. This will be achieved by intercepting $\overline{\text{DTACK}}$ and $\overline{\text{BERR}}$ bus cycle termination signals. This is an interpretation of the design from Microprocessor Systems Design by Alan Clements.

The four onboard LED's will be used to show the status of the CPU or other parts of the design. One useful signal is $\overline{\text{HALT}}$ which the CPU drives when it has encountered a situation from which it can't recover. In this state it drives all its pins to high impedance.

A set of 5 header pins will be dedicated as points to connect a dual channel oscilloscope. This will assist in finding timings errors, phase and cycle time of clock signals and general verification.

### 4.1.5
### Operating System Versions and storage

TOS 1.00 has been chosen as the initial Operating System to use. Although is suffers from many bugs, the BIOS has been listed and fully commented in the book "Atari ST Internals" by Data Becker. Combined with the ability to single step through each instruction will undoubtedly help finding any problems in the design.

The operating system will be stored in the Flash memory that is part of the RaggedStone development board. The Flash memory data bus is only 8 bits wide, therefore it will be necessary to design a VHDL component to wrap around the Flash memory and resize the data bus to 16 bit that the Atari ST uses.

There also needs to be a way to load the Operating System into the Flash memory. As the Flash memory is non-volatile once this has been programmed, the contents remain even after power is removed. Xilinx have created for their own Spartan development board an FPGA design that uses the RS232 serial port to receive data from a host computer and load into the Flash memory. Their design is based around the PicoBlaze system on chip and using the ST Microelectronics M29DW323DT Flash memory chip [22].

**4.1.6**

**5v PCI I/O**

The 5v tolerant I/O will be attached to a custom PCB at the rear of the enclosure for the following use:

- VGA interface port
- Floppy Disk Drive port
- RS232 Serial port
- Keyboard/Mouse interface

**4.1.7**

**VGA**

VGA is a mix of analogue and digital. The colour intensity is carried over three analogue signals for Red Green and Blue. The horizontal and vertical synchronisation signals are digital 5v.

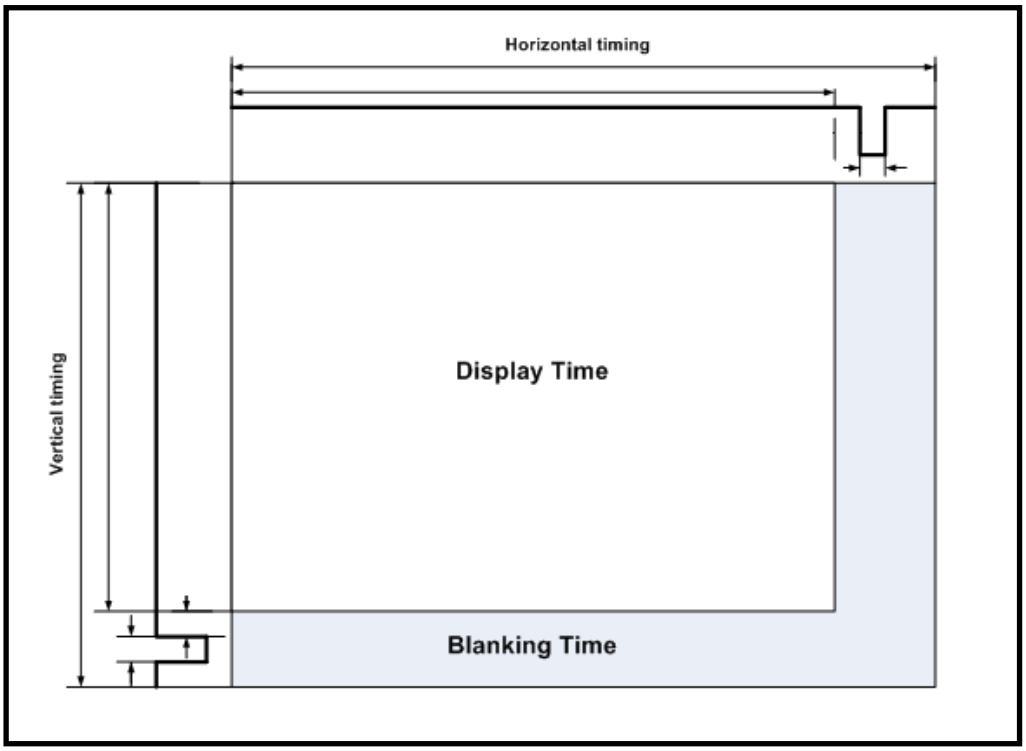



*Figure 15 – VGA video timing*

The Atari ST can display colours from a palette of 4096 different colours. Therefore each colour component can have 2^4 levels of intensity. To convert from the digital output from an FPGA to the analogue input of a VGA monitor, a DAC (Digital to Analogue) converter will be used.

### 4.1.8

### Floppy Disk Drive

The floppy drive requires a minimum of 11 signals to function. In the Atari ST these were the following, with reference to the pin number on the 34 pin 0.1" pitch IDC header [23].

To Floppy Drive

- 10: Drive Select
- 16: Motor On
- 18: Step Direction When you step the head, this line must tell the drive whether to step in or out.
- 20: Step. This line is briefly signaled to step the drive one track in the direction step direction specifies (in or out).
- 22: Write Data. This is a bit stream data for the disk track at around 100,000 baud.
- 24: Read/Write. When +5, the drive is reading. When +0, the drive is writing.
- 32: Side Select. Pull to +0 volts to write to back side of diskette.

From Floppy Drive

- 8: Index Pulse. Goes to ground briefly each rotation, five times per second (300 RPM). Otherwise +5.
- 26: Track 0. +5 unless drive is at track 0, when this pin goes to +0 volts. This is how the drive tells the FDC to stop stepping it towards track 0.
- 28: Write Protect. +0 volts if the write protect tab is set on the diskette; +5 volts if it is okay to write to the diskette.

- 30: Read Data: This is the bit stream data from the track, at 100,000 baud, complete with wow and flutter.

The floppy drive used in the Atari ST was in fact the same as used in many IBM Clone PCs. The Atari ST from TOS v1.02 onwards even uses the FAT12 filing system, compatible with a PC formatted disk.

### 4.1.9

### RS232 Serial

The RS232 will serve two purposes. Firstly it is used as an interface to a host computer to transfer the operating system into the Flash Memory. Secondly, it will be used for the Atari ST serial port. The more common 9 pin female D-Type connector will be used, instead of the Atari ST 25pin D-Type connector. Flow control and specialist MODEM only signals will be omitted to reduce the number of FPGA I/Os needed from 9 down to only 2.

| | 9-Pin | 25-Pin |
|---|---|---|
| Carrier Detect | 1 | 8 |
| Receive Data | 2 | 3 |
| Transmit Data | 3 | 2 |
| Data Terminal Ready | 4 | 20 |
| System Ground | 5 | 7 |
| Data Set Ready | 6 | 6 |
| Request to Send | 7 | 4 |
| Clear to Send | 8 | 5 |
| Ring Indicator | 9 | 22 |

*Table 5 – Serial port pin description*

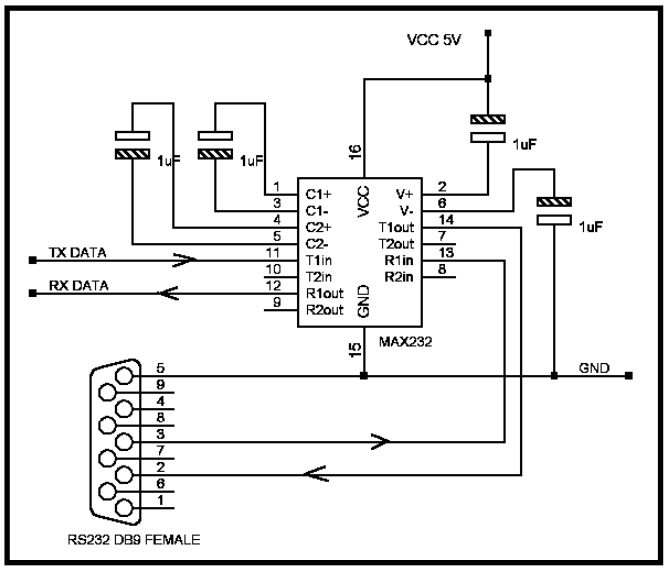



*Figure 16 – RS232 to TTL level translator*

### 4.1.10
### Keyboard and Mouse

The Atari ST communicates to the keyboard via a simple serial interface. The original keyboard contains a small Hitachi microcontroller that scans the keyboard matrix and the status of the mouse and joysticks. It then creates packets of data to be sent over the serial connection to the MC6850 UART on the Atari ST motherboard. The Atari keyboard and mouse contain mechanical components, and are one of the first parts to break or become faulty. For this reason it has been decided to implement a conversion from PC PS2 keyboard and mouse protocol. A project called 'Eiffel' by Laurent Favard, and later Didier Méquignon does just that, using an inexpensive PIC Microcontroller [24].

## 4.2
## Other differences between the original Atari ST and the design

The Atari ST as mentioned previously has 3 different resolutions. The monochrome high resolution has vertical and horizontal timings that are close to the VGA specification. The Low and Medium colour resolutions, which were designed to be displayed on a television, do not meet the VGA timing specification. The problem arises from the slow pixel clock, resulting in a horizontal synchronisation frequency of 15 KHz, which is half of VGA timing. To use the Atari ST in all three resolutions it meant you either need to have both a television and high resolution monitor, or a very expensive 'Multisync' monitor. To overcome this, a device known as a 'scan doubler' will be designed to buffer the RGB data and resynchronise it to a higher pixel clock.

The Atari ST didn't have the facility for an internal hard drive, to keep the costs low. However, a DMA port was available for connecting to external hard drives. The Atari ST was designed just before the SCSI (Small Computer System Interface) command protocol was finalised, and thus Atari used the ACSI (Atari Computer System Interface) command protocol.

As the IDE (Integrated Disk Electronics) hard drives became popular on IBM PC Clones, their price dropped compared to the SCSI equivalent. Atari realized that and on their last home computers, the Falcon and Stacy, they added IDE support.

The design will use a simple IDE interface, to use a Compact Flash card in IDE mode as they are 3.3v tolerant which enables direct linking to the Xilinx FPGA. As an IDE interface is to be designed, the original ACSI port will be left out of the design [25].

The Atari ST featured a port for a cartridge, sometimes known as the ROM port. As the name suggests this is a read only direct connection to the CPU data and address bus. The design will not include this, as although easy to implement in logic, the required 2mm pitch edge connector is not available. It is also not needed for the Atari ST to function, it rarely used for software protection dongles and most recently Ethernet and USB interfaces.

The MIDI ports are only use by music sequencing software and are not an essential part of the Atari ST design. They are based around the same UART that the keyboard uses. MIDI uses a current loop, where the current, not the voltage level defines the logic state. Therefore, MIDI ports require quite a bit of additional external circuitry.

The external 2nd floppy drive port will not be implemented as no software requires two drives. The only case where one may be useful is duplicating floppy disks, but the majority of floppy disk copying software use the system RAM as temporary storage of files.

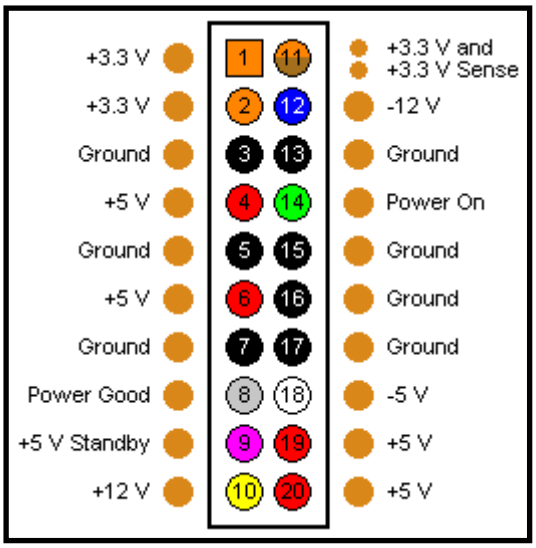


*Figure 17 – ATX pin description*

The design will be built into a standard Micro ATX enclosure. This will provide physical protection for the delicate electronics and

allow use of an ATX power supply. The ATX power supply provides a wide range of voltages. One in particular, the +5v Standby can be used to provide power to additional circuitry on a motherboard. This is used to support soft-off or standby and can be used for remote wake up through Wake-on-Ring or Wake-on-LAN. It has been chosen that the 'Eiffel' Keyboard and Mouse microcontroller will be powered from this +5v standby voltage and use modified firmware to control the ATX power supply. This allows the Power On key on many extended PS2 keyboards to turn on the computer.

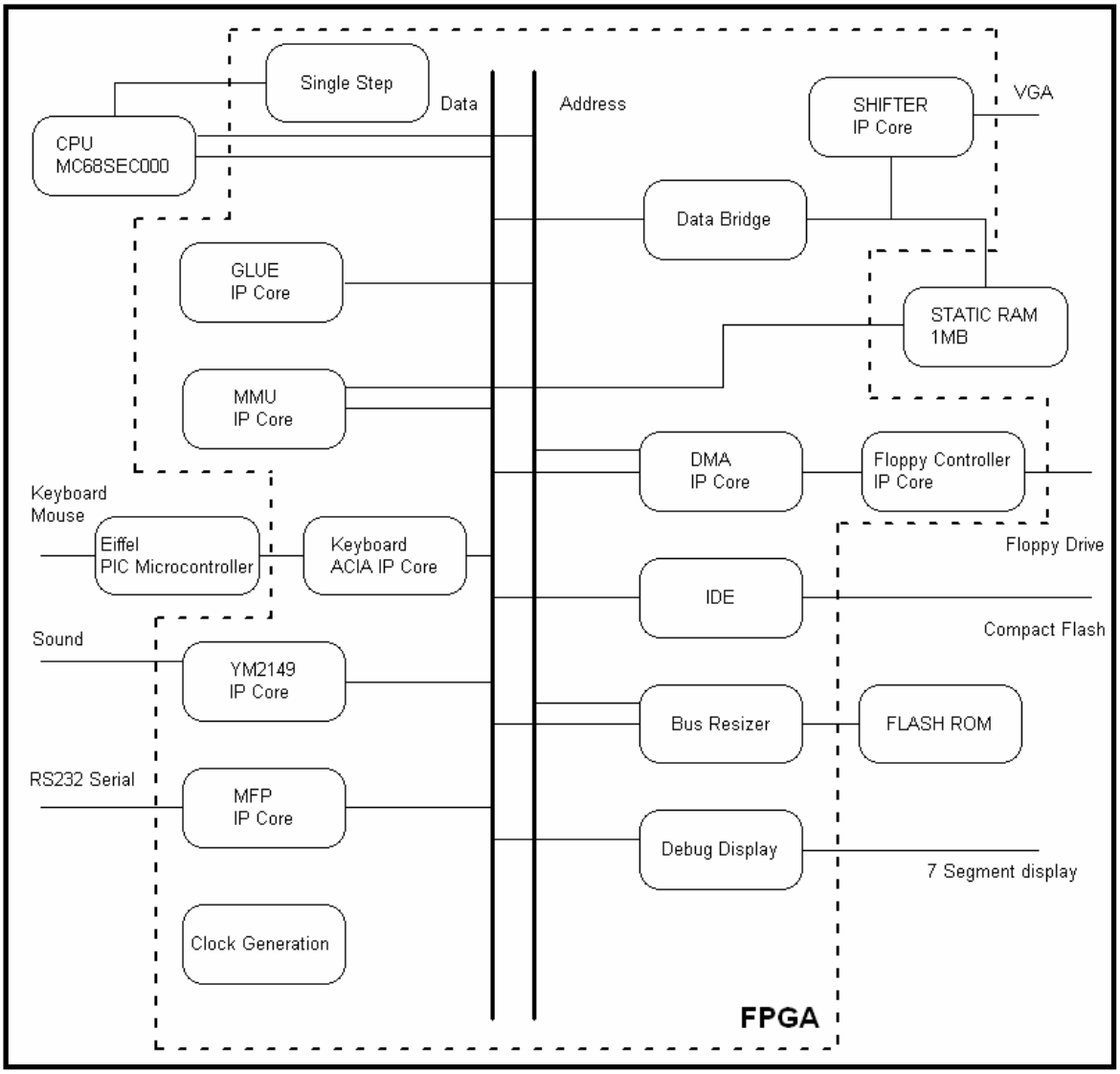



*Figure 18 – Design Block diagram*

To summarise, the design will follow the original Atari ST, but make use of more common and readily available components from IBM PC Clones.

## 4.3
## Process of Implementation

The implementation will be created in stages, logically from a small system with minimal IP Cores to the final version. Below is a brief proposal of stages involved. The order may change during implementation due to certain stages requiring later parts of the design.

- A simple LED Flash
- Verify Video Digital to Analogue works
- A project to transfer Operating System into Flash memory
- Test reading from Flash memory
- A new project with support logic, reset and clock generation
- A debug control and display
- Add Glue IP Core and verify
- Add MMU IP Core, SRAM memory and verify
- Add Shifter IP Core and verify
- Add MFP IP Core and verify
- Add Yamaha IP Core and verify
- Add Keyboard/Mouse ACIA IP Core
- Add Eiffel interface and verify
- Add DMA IP Core and verify
- Add FDC IP Core and verify

# Chapter 5

# Implementation

### 5.1

### Flashing LED

The first task was to make sure that the oscillator clock works and that the JTAG programming works. To do this a simple LED flash routine was written. However, before this was done, constraints for the I/O pin mapping and a top level component needed to be written. This was done by looking at the schematics of the RaggedStone development board and laboriously assigning names for each I/O pin. Appendix A lists the constraints file and component file.

Shown right was the VHDL architecture for flashing an LED at approximately 1Hz using a 32MHz oscillator clock.

```
architecture Behavioral of main was
signal counter : std_logic_vector(24 downto 0);
begin

       process(clock) was
              begin
              if rising_edge(clock) then
                     counter <= counter + '1';
              end if;
       end process;

LED2 <= counter(24);

end Behavioral;
```

*Figure 19 – Architecture for LED Flash*

### 5.2

### VGA Colour pattern

The next step was to implement a Digital to Analogue Converter (DAC) for the Red, Green and Blue signals to drive the VGA port. The converter being used was an ST Microelectronics STV8438, which is capable of 3 x 8bit colour. As the Atari ST can only produce 3 x 4bit colour, the MSB (Most Significant Bits) are used and the rest are tied low. Appendix B shows the schematic.

To drive the VGA monitor a colour pattern generator was discovered written by MikeJ [26]. The colour pattern generator was designed for the Xilinx Spartan-3E Starter Kit

development board, so using the project on the Raggedstone will not work as the pin constraints are wrong. To rectify this, the top level of the project was instantiated as a component using the same constraints from the previous LED Flasher stage.

Another change that needed to be made was to exchange the 32 MHz oscillator for a 50 MHz oscillator as mentioned in the comments in the colour pattern generator project. If this isn't changed the signals will not adhere to the VGA specification and a monitor will unlikely be able to 'sync' to the reduced frequencies.

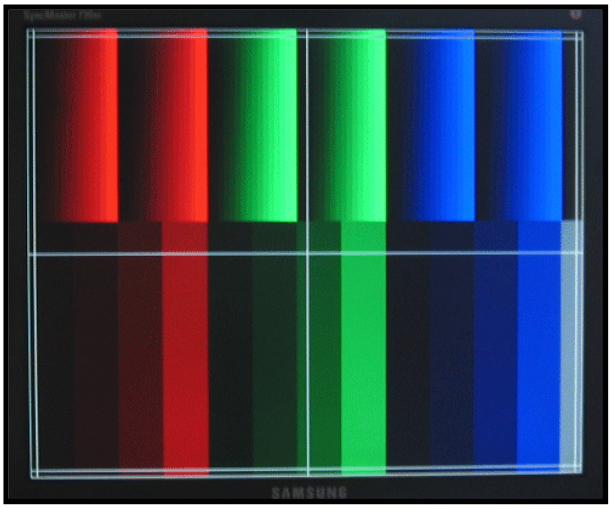


*Figure 20 – Colour test display*

Once the project was built a colourful test pattern was displayed on an attached VGA monitor. This verifies that the video DAC was functioning correctly and the conversion from 3.3v to 5v works well, even for high speed digital signals.

**5.3**

**Writing bytes to Flash Memory**

Next on the list was to load data into the RaggedStone onboard Flash memory. A design was found on the Xilinx website for the Spartan-3A/3AN Development Starter Kit. This design was intended to be used with the ST Microelectronics M29DW323DT Flash memory that was featured on the Xilinx Spartan-3A/3AN development board. It uses the Xilinx Picoblaze embedded

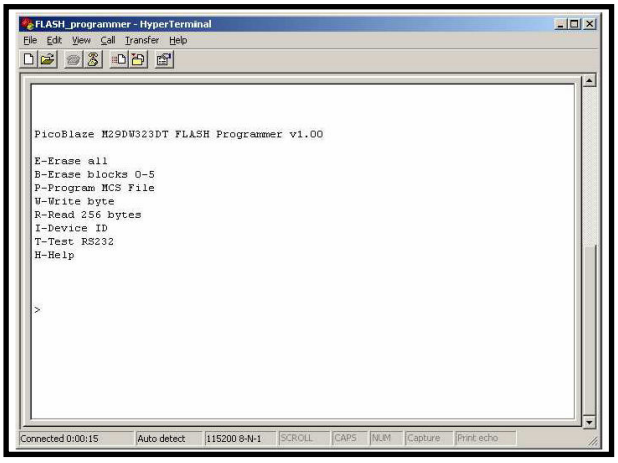



*Figure 20 –Flash Programmer menu*

microcontroller, and by using a simple terminal program over an RS232 serial connection you can manually program individual bytes, download complete files, erase the flash, read the memory to verify contents, and display the Flash memory device identifier and 64-bit unique device numbers.

An RS232 serial port was added as mentioned in the Design. (Paragraph 4.1.9)

After building the project for the RaggedStone board, it became apparent that it didn't work. The menu choices were available proving the serial connection worked fine but programming a single byte didn't work, let alone the entire Flash memory space. The only real difference between the Xilinx Spartan-3A/3AN development board and the RaggedStone was the type of Flash memory device. The RaggedStone uses an Atmel AT49BV040A and the Xilinx board uses an ST Microelectronics M29DW323DT, both configured as 8 bit wide data bus. The difference becomes quite clear when reading the data sheets provided by the manufacturers.

Atmel AT49BV040A

| Command Sequence | Bus Cycles | 1st Bus Cycle | | 2nd Bus Cycle | | 3rd Bus Cycle | | 4th Bus Cycle | | 5th Bus Cycle | | 6th Bus Cycle | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Addr | Data | Addr | Data | Addr | Data | Addr | Data | Addr | Data | Addr | Data |
| Read | 1 | Addr | $D_{OUT}$ | | | | | | | | | | |
| Chip Erase | 6 | 555 | AA | AAA[2] | 55 | 555 | 80 | 555 | AA | AAA | 55 | 555 | 10 |
| Sector Erase | 6 | 555 | AA | AAA | 55 | 555 | 80 | 555 | AA | AAA | 55 | SA[5] | 30 |
| Byte Program | 4 | 555 | AA | AAA | 55 | 555 | A0 | Addr | $D_{IN}$ | | | | |
| Boot Block Lockout[3] | 6 | 555 | AA | AAA | 55 | 555 | 80 | 555 | AA | AAA | 55 | 555 | 40 |
| Product ID Entry | 3 | 555 | AA | AAA | 55 | 555 | 90 | | | | | | |
| Product ID Exit[4] | 3 | 555 | AA | AAA | 55 | 555 | F0 | | | | | | |
| Product ID Exit[4] | 1 | XXX | F0 | | | | | | | | | | |

ST Microelectronics M29DW323DT

| Command | Length | Bus Write Operations | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st | | 2nd | | 3rd | | 4th | | 5th | | 6th | |
| | | Add | Data | Add | Data | Add | Data | Add | Data | Add | Data | Add | Data |
| Read/Reset | 1 | X | F0 | | | | | | | | | | |
| | 3 | AAA | AA | 555 | 55 | X | F0 | | | | | | |
| Auto Select | 3 | AAA | AA | 555 | 55 | (BKA) AAA | 90 | | | | | | |
| Program | 4 | AAA | AA | 555 | 55 | AAA | A0 | PA | PD | | | | |
| Quadruple Byte Program | 5 | AAA | 55 | PA0 | PD0 | PA1 | PD1 | PA2 | PD2 | PA3 | PD3 | | |
| Unlock Bypass | 3 | AAA | AA | 555 | 55 | AAA | 20 | | | | | | |
| Unlock Bypass Program | 2 | X | A0 | PA | PD | | | | | | | | |
| Unlock Bypass Reset | 2 | X | 90 | X | 00 | | | | | | | | |
| Chip Erase | 6 | AAA | AA | 555 | 55 | AAA | 80 | AAA | AA | 555 | 55 | AAA | 10 |
| Block Erase | 6+ | AAA | AA | 555 | 55 | AAA | 80 | AAA | AA | 555 | 55 | BA | 30 |
| Erase Suspend | 1 | BKA | B0 | | | | | | | | | | |
| Erase Resume | 1 | BKA | 30 | | | | | | | | | | |
| Read CFI Query | 1 | AA | 98 | | | | | | | | | | |
| Enter Extended Block | 3 | AAA | AA | 555 | 55 | AAA | 88 | | | | | | |
| Exit Extended Block | 4 | AAA | AA | 555 | 55 | AAA | 90 | X | 00 | | | | |

*Table 6 – Comparison of different Flash memory commands*

Flash Memory works with commands that are passed on the address bus, and it can be seen that the commands vary from different manufacturers. However, as the project uses a PicoBlaze microcontroller it was quite easy to change the software that it runs to use different commands. Luckily, the assembler source code for the PicoBlaze was provided and was commented and structured cleanly. The commands with 'AAA' are changed to '555' and the commands with '555' are changed to 'AAA'. The assembler source was then assembled with the PicoBlaze assembler which generates a VHDL ROM file.

After these changes were made the individual bytes of the Flash memory could be programmed and read back. However, the Atari ST operating system was 192K bytes, so there needs to be a method of programming an entire file to the Flash memory.

## 5.4
## Writing file to Flash Memory

The programmer menu does accept entire files, but of the MCS type. MCS was a file format by Xilinx for storing the FPGA configuration inside a PROM. It's formatted as an ASCII file with each line following the format below.

```
: [ Address ] [ Data ] [ CR ] [ LF ]
```

*Figure 22 – MCS file format*

The PicoBlaze project expects this file format, so the project was changed with a new choice in the menu to read raw bytes from the serial port and program the Flash memory, incrementing the address on each byte. A test was then done after each byte programmed to see if the address had reached 196608. This way a raw binary file can be transferred and programmed. Appendix C shows the assembler source code for this part of the program.

It's important to use a terminal program that is capable of sending raw binary data. It was found Microsoft's HyperTerminal interprets some of the raw data as terminal control codes and these won't get sent out over the serial port. A rather good freeware program called Realterm which has a vast array of options was used instead of HyperTerminal.

After these changes were made the Flash memory was successfully programmed with the Atari ST operating system, version 1.00.

**5.5**

**CPU**

Now it was time for the CPU to be connected to the FPGA. A daughter board was created to be used on the RaggedStones right hand I/O bank. The MC68EC000 and MC68SEC000 feature a $\overline{\text{MODE}}$ pin which selects the data bus, and as the Atari ST uses a 16 bit data bus this was tied to VCC. Any bidirectional signals, like the CPU data bus are terminated with Xilinx internal pull-ups that were added as constraints into the design.

**5.6**

**Reset**

Next was to provide the new CPU with a clock and reset. The Atari ST uses an NE555 timer chip to produce the reset signal. This is activated on power up and whenever the reset button is pressed. A VHDL component was created with a couple of counters, one for a power up reset signal and the other to produce a reset signal when the reset button is pressed.

It is important to have these two different reset signals, as some parts of the design only need to be reset on power up to known states. One of these components was the clock signal component. It was important for the CPU that the clock was running while a reset is issued, and that the reset was active for at least 132 clock cycles [27].
Appendix D lists the VHDL reset component with the RaggedStone switch S1 used as the reset button.

**5.7**

**Clock**

The clock component was necessary for generation of clock signals from the master clock, which in the Atari ST was 32 MHz. Below, was the clock frequencies that each component of the Atari ST needs.

- CPU – 8 MHz
- GLUE – 8 MHz

- MMU – 16 MHz
- SHIFTER – 32MHz
- MFP – 4 MHz and 2.4576 MHz
- YM2149 – 2 MHz
- ACIA – E Clock and 0.5 MHz
- FDC – 8 MHz
- DMA – 8 MHz

It was found that it's very important to use the dedicated DCM (Digital Clock Management) PLLs (Phase Locked Loops) that are provided inside the Xilinx Spartan. Using these reduces clock skew and jitter, and also use dedicated global clock routes inside the FPGA. This was to help prevent the clock edges arriving at different times to various components in the FPGA. The DCM can divide a clock from the master frequency and/or multiply it. Without using DCMs the Xilinx ISE software was producing warnings about non dedicated clock routing, and building the project with only small changes was resulting in very significant changes in system stability. As a result of using DCM and dedicated clock routing there was a twofold increase in maximum clock frequency [28], [29], [30].

The MFP in the Atari ST used a dedicated crystal to achieve the 2.4576 MHz frequency. This was used by the MFP for the serial port baud rate. In the FPGA it was possible to use a DCM to create this frequency. The most accurate was to synthesize a 27 MHz clock from the 32 MHz master clock and then divide by 11 to get 2.4545 MHz.

**5.8**

**Synchronous Bus interface**

The ACIA uses the E Clock, which unfortunately the MC68SEC000 CPU doesn't provide. The E Clock was at one tenth of the CPU frequency with a 60/40 duty cycle. The 68SEC000 also doesn't have connections for the $\overline{\text{VPA}}$ or $\overline{\text{VMA}}$ signals.

The E Clock was created by a counter that counts from 0 to 9 and then rolls over. If the value of the counter was 0 to 5 then the E clock was 0, otherwise it will be 1. The Glue component of the Atari ST then asserts the $\overline{\text{VPA}}$ signal to tell the CPU an access to a 6800 synchronous device has been made, which in the Atari ST was an access to the Keyboard or Midi ACIA MC6850. The $\overline{\text{VPA}}$ signal was checked when the E Clock counter was 2, and if it was active then $\overline{\text{VMA}}$ was asserted. $\overline{\text{DTACK}}$ was then asserted later when the E Clock counter was 8 or 9 to end the bus cycle. By asserting $\overline{\text{DTACK}}$ late, the CPU automatically inserts wait states. Appendix E shows the Clock VHDL component.

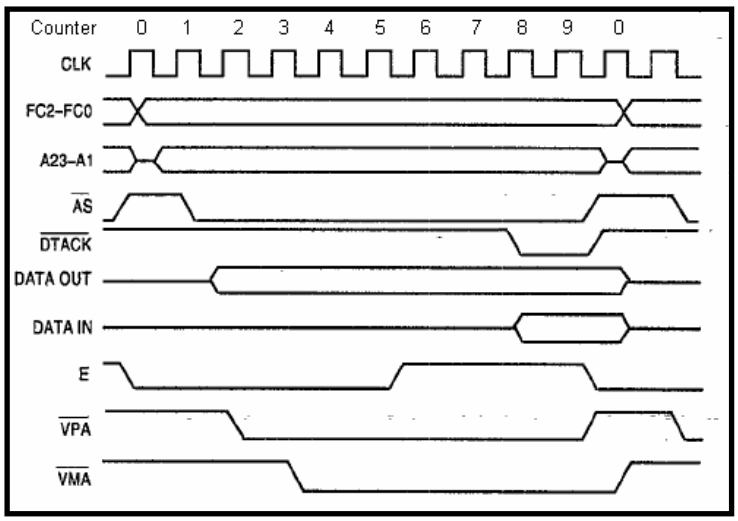


*Figure 23 – Synchronous bus interface*

**5.9**

**Flash data bus resizing**

Now that the clock and reset was provided to the CPU, next was to make the connection between CPU and Flash memory where the operating system was located. As previously mentioned, the Atari ST had 16 bit wide ROM, but the Raggedstone Flash memory was only 8 bit wide.

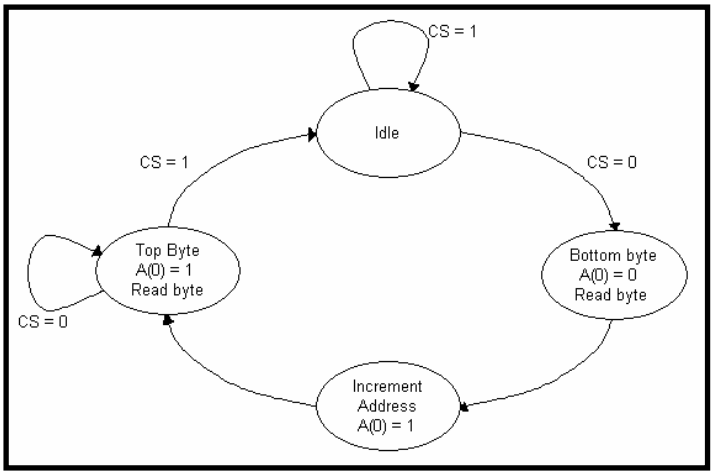


*Figure 24 – Flash State machine*

As the MC68000 completes a bus cycle in 4 cycles (500ns at 8 MHz), and the Atmel AT49BV040A has an access time of 70ns it's quite possible to fit two 8 bit accesses to the Flash to make it appear 16 bit wide to the CPU. To achieve this, a wrapper VHDL component was created with a FSM (Finite State Machine) controlling latching of data and the LSB of the address. Appendix F shows the VHDL component.

**5.10**

**7 Segment Display**

As part of debugging, a VHDL component was created to use the RaggedStone onboard 7 segment display. The RaggedStone has four of these 7 segment displays, enabling 4 hex characters (or 16 bits) to be displayed. This was perfect for displaying the 16 bit data bus, but not the 24 bit address bus. To overcome this limitation, the display will sequence through the upper portion of the address bus, then the lower portion of the address bus and lastly the data bus.

The VHDL component will also be responsible for changing the 4 bit hex value into a value to drive the 7 segment display. The 7 segment displays on the RaggedStone, are just a set of LEDs with no intelligence. Another part it will cater for was scanning the digits of the 7 segment display. Only one digit can be displayed at one time, and thus it needs to scan through the digits quick enough for the human eye not to see any flicker. Appendix G shows this VHDL component. One problem encountered was that the mapping for the digits to FPGA pins listed in the RaggedStone user manual appears to be wrong. The table below shows the correct pin mapping.

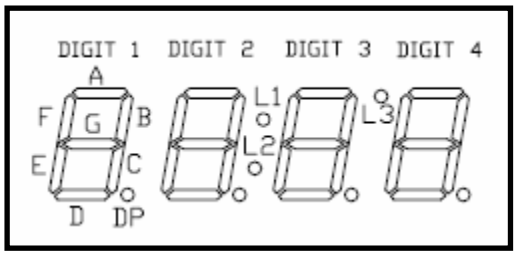


*Figure 25 – 7 segment display*

| Digit 1 | Digit 2 | Digit 3 | Digit 5 |
|---|---|---|---|
| FPGA U14 | FPGA AA17 | FPGA U17 | FPGA U16 |

*Table 7 – FPGA connections for 7 segment display*

**5.11**

**Single Step**

Having the address and data bus displayed was only good if there was a way to slow the system down, or even single step through each instruction. A solution was available, with the idea taken from 'Microprocessors System Design' by Alan Clements. Alan Clements design was for a single board computer based on the MC68000, where one can pause the CPU and by pressing a push button, single step through instructions. His design was based on four TTL logic flip flops. Appendix H shows the schematic design.

Quite simply there was a switch to bypass and let the system run normally. In single step mode the $\overline{\text{DTACK}}$ signal was intercepted. When the MC68000 starts a bus cycle it will insert wait states until it receives the $\overline{\text{DTACK}}$ signal. A momentary push switch controls the assertion of $\overline{\text{DTACK}}$, with flip flops used to de-bounce the push button and insure only one bus cycle was executed no matter how long or short the button was held down for.

The VHDL component in Appendix I for single step works on the same principles. However, the $\overline{\text{BERR}}$ signal was also intercepted as this signal was also used to terminate a bus cycle in the event of a bus cycle error, e.g. no device at address specified. One of the RaggedStone buttons, S2 was used as the single step button. The Run/Stop was implemented as a 'jumper' across two spare I/O pins on the RaggedStone.

**5.12**

**Glue IP core**

Before trying the system the Glue VHDL IP Core had to be added. It was at this point that I noticed the IP Cores at the top level used bit and bit_vector signal types and all the other components that I had written used std_logic and std_logic_vector. It is possible to convert between the two types, but this can become untidy because it is no possible to do the conversion within the component port maps. A decision was made to alter the

previous components to use bit and bit_vector which creates a cleaner implementation. The only parts to use std_logic and std_logic_vector are when the signals leave the FPGA and need to be bi-directional or tri-state.

The Glue is needed for address decoding, and it is responsible for generating the chip select for the Atari ST ROM space. An excerpt from the Glue component wf25915ip_adrdec.vhd is shown in figure 24.

```
ROM_2n <= '0' when ST_RD = '1' and ADR_HI >= x"FC" and ADR_HI < x"FD" else
       -- ST TOS ROM LOW.
          '0' when READx = '1' and ADR_INT < x"000008" else '1';
       -- TOS mirroring.

ROM_1n <= '0' when ST_RD = '1' and ADR_HI >= x"FD" and ADR_HI < x"FE" else
       -- ST TOS ROM MID.

ROM_0n <= '0' when ST_RD = '1' and ADR_HI >= x"FE" and ADR_HI < x"FF" else
          '1';  -- ST TOS ROM HI.
```

*Figure 26 – Address decoding in Glue*

The Atari ST used 6 small 8 bit wide 32KB PROMs to make up the 192KB size of the operating system. With the ROM being accessed as 16 bit, this requires the three chip select lines shown in the above VHDL code. The ROM space was located from 0xFC0000 to 0xFEFFFF. A special mirror, or sometimes known as shadow was created for the first 8 bytes of the operating system, located at address 0x000000 to 0x000007. The purpose of this was explained in the 9th Edition of the MC68000 User Manual

When $\overline{\text{RESET}}$ and $\overline{\text{HALT}}$ are driven by an external device, the entire system, including the processor, is reset. Resetting the processor initializes the internal state. The processor reads the reset vector table entry (address $00000) and loads the contents into the supervisor stack pointer (SSP). Next, the processor loads the contents of address $00004 (vector table entry 1) into the program counter. Then the processor initializes the interrupt level in the status register to a value of seven.

*Figure 27 – MC68000 Start up sequence*

Inspecting the start of the Operating System with a hexadecimal editor shows the values that get loaded into the SSP (Supervisor Stack Pointer) and PC (Program Counter). After

the CPU has fetched these two long words, it continues execution from the address in the PC. The address was 0xFC0020 which was a jump into the ROM space. It can be seen that absolute addresses are 32 bit, even though the external address bus was only 24 bit.

The RaggedStone has four onboard LEDs. These are assigned to the following signals, $\overline{\text{BERR}}$, $\overline{\text{DTACK}}$, $\overline{\text{RESET}}$ and $\overline{\text{HALT}}$. The system was then powered up and the start up sequence of the $\overline{\text{RESET}}$ and $\overline{\text{HALT}}$ remaining active for 1 second was observed. The 7 segment display then shows the address as 0x000000 and data as 0x601E. Pressing the single step button showed the address change to 0x000002 and the data as 0x0100.

| OFFSET | 00 | 01 | 02 | 03 |
|---|---|---|---|---|
| 0000:0000 | 60 | 1E | 01 | 00 |
| 0000:0004 | 00 | FC | 00 | 20 |

*Table 8 – Start of Operating System*

The components verified as working are the CPU, GLUE address decoding, 7 segment debug display, Flash memory and the single step component. At this stage other sub components of the GLUE, like the interrupt controller and video timing generation are left unconnected, or in VHDL terms 'open' in the port map.

When building the project with the Glue added to the project, the Xilinx ISE software would crash with an exception error and consequently exit. After a lot of trial and error it was found that upgrading from Xilinx ISE v8.2 to Xilinx ISE v9.2 solved this problem!

The system can now be stepped through, up until the address 0xFC05DA. At this point a $\overline{\text{BERR}}$ (bus error) is signalled for the bus cycle. Looking through the commented assembler source code of the Operating System it can be seen that a read was attempted from RAM. Without the MMU implemented to generate a $\overline{\text{DTACK}}$ response, the GLUE time out counter issues a $\overline{\text{BERR}}$ response.

## 5.13

## MMU IP core

Next to implement was the MMU to enable the system to run further through the operating system. The SRAM main memory chosen was the BS616LV8017 512K by 16 bit SRAM used in conjunction with a Roth Elektronik TSOP to DIL adapter. This was mounted onto the RaggedStones left hand I/O bank.

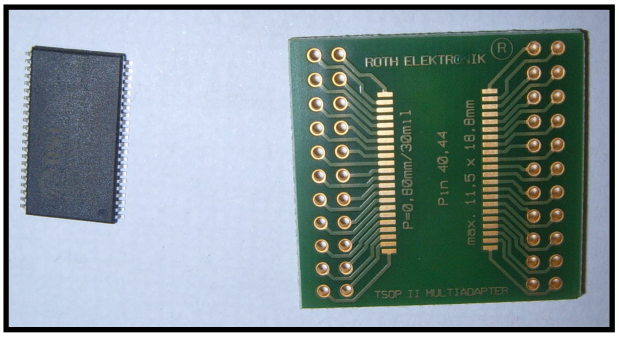



*Figure 28 – SRAM and adapter PCB*

One of the MMUs tasks was to keep the Atari ST DRAM memory refreshed. As the design was using SRAM this refreshing must be disabled. This also makes debugging far easier as the only accesses to RAM will be memory accesses, not refresh cycles as well. To remove the refreshing the refresh address counter was removed from the multiplexer in wf25912ip_ram_adrmux.vhd.

```
M_ADR <=    ADR when MCU_PHASE = RAM and DMAn = '1' else
            DMA_ADR when MCU_PHASE = RAM and DMAn = '0' else
            VIDEO_ADR when MCU_PHASE = VIDEO else
            SOUND_ADR when MCU_PHASE = SOUND else
            -- Lyndon Amsdon Removed refresh
            --"00000000000" & REF_ADR; -- Refresh cycles.
            (others => '0');
```

*Figure 29 – Removed refresh address generation*

Also, when in the MCU_PHASE = REFRESH state the assertion of RAS was removed in wf25912ip_ctrl.vhd.

```
--RAS0n <='0'when(RAS_Pn = '0' and RAS_Nn = '0' and BANK_SWITCH = BANK0)else
--'0' when (MCU_PHASE_I = REFRESH and RAS_Pn = '0' and RAS_Nn = '0') else '1';

--RAS1n <='0'when(RAS_Pn = '0' and RAS_Nn = '0' and BANK_SWITCH = BANK1) else
--'0' when (MCU_PHASE_I = REFRESH and RAS_Pn = '0' and RAS_Nn = '0') else '1';

RAS0n <= '0' when (RAS_Pn = '0' and RAS_Nn = '0' and BANK_SWITCH = BANK0) else
         '1';

RAS1n <= '0' when (RAS_Pn = '0' and RAS_Nn = '0' and BANK_SWITCH = BANK1) else
         '1';
```

*Figure 30 - Removed refresh RAS generation*

As the Atari ST used DRAM memory with multiplexed address into rows and columns it was necessary to take the address from the MMU prior to this multiplexing. The CAS lines are now used as byte selection for the SRAM. The $\overline{WE}$ (Write Enable) was used to control the SRAM Write Enable. The address for the SRAM was constructed from the non multiplexed address and the top address bit was taken from the bank selection inside the MMU.

```
--MMU section
SRAM_OEn <= '0';
SRAM_CEn <= '0';
SRAM_UBn <= dram_cas0hn AND dram_cas1hn;
SRAM_LBn <= dram_cas0ln AND dram_cas1ln;
SRAM_ADDR <= bank_bit & dram_madh(8 downto 0) & dram_madl(8 downto
0);
```

*Figure 31 – 1MB SRAM signal creation*

Now the MMU and SRAM was implemented another run of the system was made. This time when the MPU makes an access to the RAM, the MMU generates a $\overline{DTACK}$ response. This allows the system to progress further into booting the Operating System, by doing an initial RAM test to figure out the size of RAM installed in each bank. It achieves this by a loop of writing data to RAM and reading back the data from RAM with a comparison. At the end of the test it programs the MMU with the RAM

configuration. It does this many times; therefore single stepping through this section would be laborious.

A problem was found in the RAM test however which took a great deal of time to figure out. The CPU has its Stack Pointer loaded right at the start upon reset, and with TOS 1.00 that is 0x601E0100 (paragraph 5.12). In TOS 1.00 there is a complex RAM test that follows from 0xFC014A onwards. It first modifies the bus error vector to point to a hander routine (0xFC0188). It starts at 128k and increments by 128k reading and writing checking if the RAM is there. When it reaches 1MB or more it goes out of the RAM range and a $\overline{\text{BERR}}$ is signalled to the CPU from the MMU.

The CPU then goes to save the program counter and copy the status register onto the stack, before it fetches the vector address from address 0x000008. The problem is that the stack pointer is still 0x601E0100 and that creates another bus error. Two bus errors in close succession create what Motorola call a double bus fault happens, and the CPU signal it has halted [27].

To mitigate this issue the MMU was fixed to generate a DTACK for a small address region from 0x1E0100 to 0x1E00F0.

At this stage it was also evident that the SRAM memory can hold its contents for a few seconds after power has been removed. On the original Atari ST a ‘cold reset’ can be performed by powering on and off the power supply, and with the refreshing of the DRAM not being performed, the contents are lost. With the SRAM memory and the fact that the design takes less power than the original Atari ST powering on and off the power supply may not be enough to force a cold reset.

## 5.14

### Hardware Breakpoint

A change was made to the single step unit, adding in address ranges to qualify for the single step mode. In other words, some portions of the Operating System can run at normal speed and when a certain address was encountered a break was made into Single Step mode. To enable this, the following code was entered into the top level of the project.

```
CPU_DTACK <=
      '0' when system_dtackn = '0' and (CPU_ADDR & '0' < x"FC01AA"
                                        and CPU_ADDR & '0' >= x"FC0200")
      else
      '0' when ss_dtack = '0'
      else
      '1';
```

*Figure 32 – Single Step DTACK generation*

With this modification in place the system was once again tested. After a brief flicker on the 7 segment display the breakpoint was hit at address 0xFC01AA after the memory test, but before the Operating System programs the MMU with the RAM configuration. It's now possible to single step and view the value written to the MMU register at address 0xFF8001. The value was 0x05 which according to the memory map from 'ST Internals' is a total of 1MB, arranged as two banks of 512KB.

| $FF8001 | Bit | Memory configuration | |
|---|---|---|---|
| | 3-0 | Bank 0 | Bank 1 |
| | 0000 | 128K | 128K |
| | 0001 | 128K | 512K |
| | 0010 | 128K | 2 M |
| | 0011 | reserved | |
| | 0100 | 512K | 128K |
| | 0101 | 512K | 512K |
| | 0100 | 512K | 2 M,normally reserved |
| | 0100 | reserved | |
| | 1000 | 2M | 128K |
| | 1001 | 2M | 512K |
| | 1010 | 2M | 2M |
| | 1011 | reserved | |
| | 11XX | reserved | |

*Table 9 – Memory Configuration register*

Single stepping further through the Operating System shows the system variables MEMTOP and PHYSTOP being correctly set to 0x0F8000 and 0x100000 respectively. The 32KB gap between the top of available RAM and the physical top of RAM is due to reserved space for the graphics screen buffer.

A bug was discovered in the 7 segment display at this point, where it doesn't display the odd addresses. This is because the MC68000 CPU doesn't have A0, but instead two data strobes ( $\overline{\text{UDS}}$ and $\overline{\text{LDS}}$ ). The Upper Data Strobe ( $\overline{\text{UDS}}$ ) can be used to identify an odd address bus cycle.

**5.15**

**Shifter IP Core**

Now the RAM test was passed it was time to implement the Shifter graphics chip. The Shifter takes the data directly from RAM bypassing the data bridge transceiver. It was important at this stage to recognise that the FPGA can not have internal tri state logic states. All the data buses are driven by multiple internal components and external devices, and for this a large multiplexer was used. Tri-state buffers are used when the data bus leaves the FPGA to become an external data bus for the CPU and SRAM.

Testing the Shifter at this stage was not possible, as the Operating System sets up the screen late in the boot up sequence. The next point where the Operating System fails was at address 0xFC21B4 where it attempts to initialise the currently unimplemented MFP.

**5.16**

**MFP IP Core**

Implementing the MFP allowed the Operating System to boot further, but a continuous loop was occurring at address 0xFC0CE4 to 0xFC0D1E. Inspecting the commented source code shows the MFP internal Timer B was loaded with the value of 240 and is polled to ensure the value has changed. The Timer was not running due to an error in the IP Core. It was found the strobe signal that was responsible for decrementing the timer was not running. This was verified by taking the strobe signal out to an external FPGA

I/O pin and using an oscilloscope to view the signal, which clearly identifies the signal was always at logic level ‘0’.

The strobe goes to ‘1’ when the counter value was “00”. However, at the point of being less than “01”, it was immediately loaded with the prescale value. Therefore the MFP implementation requires a small change to the IP Core as listed in Appendix J. The same change was made to all four instances of the Timer strobe signal generation. The result was verified with an oscilloscope in relation to the master clock.

Now the system was restarted and the MFP test passes fine. Removing the breakpoint for debugging the MFP allows the Operating System to start and load the AES (Application Environment System). The AES was the graphical environment that the Atari ST uses. At this point it was worth noting that the design has managed to boot up to a desktop environment without the following components.

- Keyboard/Mouse ACIA
- Yamaha YM2149 sound chip
- Midi ACIA
- DMA
- WD1772 floppy controller
- Interrupts still disabled


*Figure 33 – Photo of Desktop*

In the middle of implementing the MFP, the Xilinx ISE software began to report an error.

```
FATAL_ERROR:Portability:PortDynamicLib.c:358:1.27 - dll open of library
   <C:/Xilinx92i\bin\nt\libGenParTask.dll> failed due to an unknown
reason.
   Process will terminate. For more information on this error, please
consult
   the Answers Database or open a WebCase with this project attached at
   http://www.xilinx.com/support.
```

*Figure 34 – Xilinx ISE DLL error*

Even reverting back to a previous build version this error kept on appearing when trying to build the project. The Xilinx ISE software was reinstalled, but this didn't fix the problem. It was found that the AVG antivirus software had miss detected one of the Xilinx DLLs as a Trojan horse virus and stored it in the AVG virus vault. Marking the DLL as safe and restoring it to its previous location prevented the error.

**5.17**

**ACIA IP Core**

At this stage the keyboard ACIA was added to the project. Interrupts are required for this component to work, so the relevant connections between the Glue (which contains a simple interrupt priority encoder) and the CPU IPL(2..0) signals are made. Only three of the CPU interrupt levels are used in the Atari ST as shown in the table below. The MFP acts as an additional cascaded interrupt controller.

| Level | Source | CPU IPL(2..0) |
|---|---|---|
| 2 | Horizontal Blank | 101 |
| 4 | Vertical Blank | 011 |
| 6 | MFP | 001 |

*Table 10 – IPL encoding*

Now the ACIA was built into the project and an original Atari ST keyboard was connected, the keyboard and mouse worked but screen redraws were not occurring properly. Screen redraws are part of the VBL ISR (Interrupt Service Routine) and an error was found in the Glue component wf25915ip_interrupts.vhd. The error was that the HBL and VBL encoding was the wrong way around. Appendix K shows the fix for the Glue subcomponent.

**5.18**

**YM2149 IP Core**

Next to implement was the Yamaha YM2149 sound generator. This IP core differs in the fact that the original semiconductor has an analogue output stage, but in a Xilinx Spartan FPGA (and the vast majority of other FPGAs) are not mixed signal. Instead the IP core uses a fast PWM (Pulse Width Modulation) and an external low pass discrete filter to create the 'shape'. In the implementation the 3 channels of sound are externally mixed with resistors and then into a simple RC low pass filter.

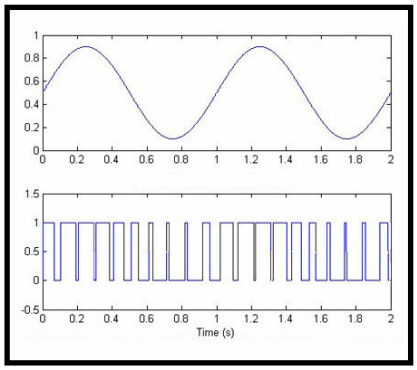



*Figure 35 – PWM sound*

The first test of the sound chip didn't work very well with a lot of background noise. It later turned out that the Glue address decoded chip select had not been connected to the YM2149 chip select, so the YM2149 was enabled all the time and acting on all the random data bus signals.

After fixing the previous issue, and powering up the system, each time a key was pressed a 'bell' sound was clearly heard as per the original Atari ST but this wasn't testing all the different envelope shapes. Further testing can only be achieved by using a program, and without a floppy drive implemented to load a program in, further testing had to wait.

While adding the YM2149 IP Core more problems crept up with the internal FPGA clock routing. Small insignificant changes were making the system refuse to boot up. While looking at differences in a previous working version, and a non working new version it became apparent that the placement tool was moving a lot of the global clock routes and DCM usage around the FPGA. A test was made by reverting back to a previous version and locking the usage of DCMs and global clock routes as a constraint to the project. This fixed the issues, but only warns of the importance of fixed and dedicated clock

signals within the FPGA and how difficult it could be in very high speed FPGA designs [31].

**5.19**

**DMA IP Core**

At this point the DMA IP Core was added. The requirement for the DMA to take over and master the system bus required quite a large change to the way the components were connected together at the top level. Instead of just the CPU driving the control signals for bus cycles ( $\overline{AS}$ , R/$\overline{W}$ , $\overline{UDS}$ and $\overline{LDS}$ ) the Glue also needs to be able to drive these signals. Even on a DMA bus cycle, it is the Glue that drives the control signals and does the bus arbitration with the CPU. The MMU also helps out by providing the address and DMA counter. It's at this point that you realise how closely linked all the custom semiconductors in the Atari ST are. The control signals from the CPU and Glue are now fed through a multiplexer.

The DMA IP core is implemented and the signals for bus arbitration between the CPU and Glue are joined, with a test made to make sure the system still functions as it did before. Without anything currently connected to the ACSI DMA bus, no further tests can be done.

Just after adding this component IP core the RaggedStone board developed a fault. While trying to identify devices on the JTAG boundary scan, the RaggedStone kept on reporting an infinite number of 'unknown devices'. The three devices in the chain (Xilinx Spartan, XCF02 configuration PROM and XCF04 configuration PROM) were separated and scanned individually. The fault was identified as the XCF02 configuration PROM and luckily it was not the FPGA. The faulty XCF02 was simply removed and a new one soldered in place.

**5.20**

**FDC IP Core**

The Floppy Drive Controller (FDC) IP Core was added to the project, along with the physical port for the floppy disk drive to attach to. In the project the FDC connects directly to the ACSI DMA bus as in the design there is no external ACSI bus, unlike the original Atari ST.

After adding the FDC IP Core to the project the system now failed to boot to a desktop environment. The interrupts to the CPU had to be disabled, and breakpoints set up in hardware to start debugging the operating system at the point it initialises the floppy disk drive controller. After a lot of time tracing through pages of the operating system assembly code it appeared the CPU could not read from the FDC status register. The read is attempted at address 0xFC1CA6 as shown in figure 36.

```
*****************************************************   rdiskctl
FC1CA4 6106                bsr     $FC1CAC             Delay loop for disk controller
FC1CA6 3039FFFF8604        move.w  $FFFF8604,D0        Disk controller status to D0
FC1CAC 40E7                move.w  SR, -(A7)           Save status
FC1CAE 3F07                move.w  D7,-(A7)            Save D7
FC1CB0 3E3C0020            move.w  #$20,D7             Counter
FC1CB4 51CFFFFE            dbra    D7,$FC1CB4          Delay loop
FC1CB8 3E1F                move.w  (A7)+,D7            D7 back
FC1CBA 46DF                move.w  (A7)+,SR            Status back
FC1CBC 4E75                rts
```

*Figure 36 – Excerpt from OS*

With the help of an attached oscilloscope it was found the timing of the chip select (FDCSn) in relation to the window of valid data was wrong and this was changed as shown below. The CTRL_MASK signal is essentially a counter to synthesise signal timings for the DMA ACSI bus. The alteration makes the FDCSn signal active for longer.

```
-- with CTRL_MASK select  FDCSn <= FDCSn_I when "110" | "101",
-- Phases 6, 5.
-- '1' when others;

with CTRL_MASK select  FDCSn <= FDCSn_I when "111" | "110" | "101" | "100",
-- Phases 7,6,5,4.
     '1' when others;
```

*Figure 37 – FDCSn from wf25913ip_ctrl.vhd*

Upon powering up the system the floppy drive was detected and the desktop appeared with the floppy drive icons. Trying to read a floppy disk still failed however, with the floppy drive light remaining on and the system locking up by not responding to mouse of keyboard actions.

The FDC IP Core was examined and the main control is implemented as a very large state machine with 73 possible states in a file called wf1772ip_control.vhd. A decision was made to see if it was this state machine that was locking up. The four LEDs on the RaggedStone were used to show the current state of the state machine, but that only provide a maximum of 16 different states. The 73 states were split into groups of 16 states and the project re-built each time to test the next group. The offending state was T1_VERIFY_CRC and the only action that controls the exit from this state are DELAY = True as shown in figure 38.

```
when T1_VERIFY_CRC =>
-- The CRC logic starts during T1_SPINDOWN (missing clock transitions).
       if DELAY = true then
              if CRC_ERR = '1' then
                     NEXT_CMD_STATE <= T1_SPINDOWN; -- CRC error.
              else
                     NEXT_CMD_STATE <= IDLE; -- Operation finished.
              end if;
       else
       -- Wait until CRC logic is ready.
              NEXT_CMD_STATE <= T1_VERIFY_CRC;
       end if;
```

*Figure 38 – T1_VERIFY_CRC state*

To rectify this, in the section for generation of the delay signal (line 747 to 860) the T1_VERIFY_CRC state was added.

After this the floppy drive works for small files at the start of the disk, nearest track 00. Anything larger and it reports the disk as unreadable. Two fixes were found that appeared to help. The operating system in the Flash memory was upgraded to TOS v1.04 that has some important fixes to hard drive and floppy disk drive handling. The other change made was in the FDC IP Core. In the IP Core there is a settling delay after each disk drive head step command of 30ms to allow the physical mechanism to move and

settle in the correct place. After reading the WD1772 data sheet it became clear the 30ms settling delay is only true for the WD1770 device, for the WD1772 it is 15ms [23],[32],[33],[34].

```
when DELAY_30ms | T1_VERIFY_DELAY =>
      case DELCNT is
            -- when x"75300" => DELAY <= true; -- 30ms
            when x"3a980" => DELAY <= true; -- 15ms
            -- when x"1d4c0" => DELAY <= true; -- 7.5ms
            when others => DELAY <= false;
      end case;
```

*Figure 39 – WD1772 delay state*

Both these fixes made the floppy drive much better in operation and now programs can be loaded to further the testing of the system.

**5.21**

**Eiffel PS/2 conversion**

A Microchip PIC Microcontroller was finally added to provide an interface for a PS/2 Keyboard and Mouse. The firmware for the PIC, called 'Eiffel' which is a GPL project was also modified to control the ATX power supply. The original project supports a temperature sensor and control of a CPU Fan, but this was removed to create some spare I/O pins [24]. Bit 0 of Port A was used for the power switch, and Bit 5 of Port C was used to drive the ATX power supply on signal. See appendix L for the modified schematic.

The PIC firmware was altered so that either a transition on the power switch or the keyboard scan code for the Enter key (0x73) will enable the power supply [35]. Another change was made to the firmware, which deviates from the originally planned Design. It was decided that using Sony Playstation controllers would be better than the original Atari style joysticks. The reasoning behind this is that the original Atari style joysticks are getting increasingly harder to find, and you certainly can't buy new ones. The Atari joysticks are simply a set of five push switches, one for fire and the other four for direction. The Sony Playstation controllers use a five wire serial communication bus to

send commands to the Playstation controller and receive status data of the controller. The ATT stands for attention, and this is a signal to define the start of the sequence. The ACK stands for Acknowledge, and is a confirmation that the controller received the command byte. The data and commands are sent LSB (Least Significant Bit) first.

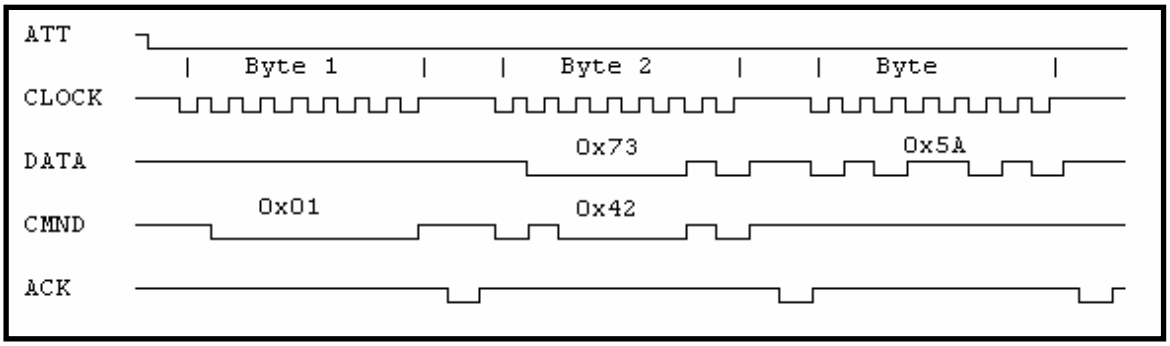



*Figure 40 – Playstaion controller protocol*

The first three bytes of the transmission are used for a handshake protocol. The next two bytes of the transmission are used to transmit the data representing the button presses. An extension was made to the protocol when Sony released the dual analogue version of their controller, and this uses the last four bytes with each byte representing the position [36].

| BYTE | CMND | DATA | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 01 | 0x01 | | | | | | | | |
| 02 | 0x42 | 0x73 | | | | | | | |
| 03 | | 0x5A | Bit0 | Bit1 | Bit2 | Bit3 | Bit4 | Bit5 | Bit6 | Bit7 |
| 04 | | data | SLCT | JOYR | JOYL | STRT | UP | RGHT | DOWN | LEFT |
| 05 | | data | L2 | R2 | L1 | R1 | △ | O | X | □ |
| 06 | | data | Right Joy 0x00 = Left | | | | | 0xFF = Right | | |
| 07 | | data | Right Joy 0x00 = Up | | | | | 0xFF = Down | | |
| 08 | | data | Left Joy 0x00 = Left | | | | | 0xFF = Right | | |
| 09 | | data | Left Joy 0x00 = Up | | | | | 0xFF = Down | | |

*Table 11 – Playstation controller packet*

The digital directional pad was used to emulate the original Atari joystick direction, and the X button for the original Atari fire button.

## 5.22
## IDE Compact Flash

This was not implemented as there was not an easy way to implement the 40 way header anywhere on the RaggedStone board. There are just enough I/O pins, but these are separated across the board and would require a lot of trailing wires running to these various points. The IDE bus is simply memory mapped into the address space from 0xF00000 to 0xF00039 and used in PIO (Programmed Input Output) Mode [37],[38]. The interrupt request from IDE is made with a logical OR with the original Atari ACSI hard disk interface.

# Chapter 6

# Verification and Testing

**6.1**

**Benchmarking**

Benchmarking software are useful tools to identify the performance of a computer. By running benchmark software on the system, it should be possible to identify any errors either in performance or functionality. The program used is called Gembench written by Ofir Gal in 1995, along with another program called SysInfo by Thorsten Bergner in 1997. Gembench benchmarks the AES (opening dialog boxes, scrolling text etc), CPU speed (maths routines) and memory bandwidth. SysInfo reports information on system variables, memory configuration and size. Appendix M shows the results of these two tests. The Gembench scored 99% of a real Atari ST, probably due to the real PAL Atari ST having a 32.08 MHz master clock. The 102% score for VDI Scroll is probably due to using TOS v1.04 where certain areas of the operating system were optimised slightly. The SysInfo results are exactly the same as a 1MB Atari ST, and this verifies it has detected the size of SRAM memory correctly and set up all the associated system variables and configuration registers.

**6.2**

**Colour Palette**

In the colour resolution modes the colours were incorrect compared to an Atari ST. At first the connection between the FPGA and Video DAC were checked and these were correct. The connections between the Video DAC and VGA connector were also correct. With the floppy drive now operational, it was possible to load

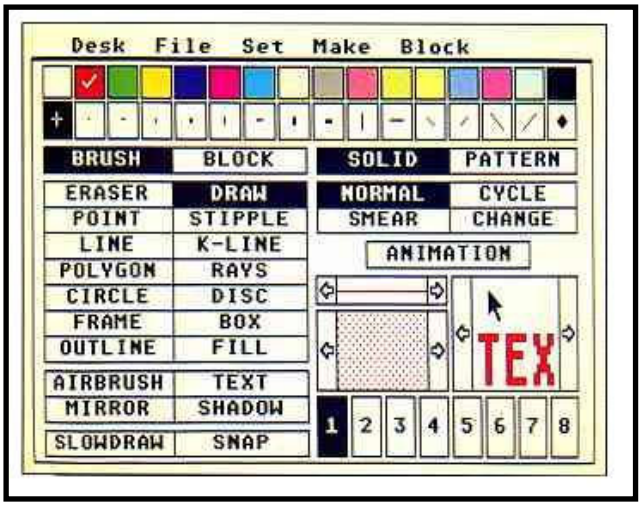



*Figure 41 – Degas Elite*

programs in. An art drawing program called “Degas Elite” written by Tom Hudson in 1987 was loaded from the floppy drive. With this program it is possible to change the shade of colour in the palette registers of the Shifter graphics IP Core. It was found that adjusting certain palettes changed the wrong colour on the screen. A table was made to figure out what was happening.

| **Palette changed** | **Bit pattern (3..0)** | **Actual change** | **Bit pattern (3..0)** | **Correct?** |
|---|---|---|---|---|
| 0 | 0000 | 0 | 0000 | YES |
| 1 | 0001 | 4 | 0100 | NO |
| 2 | 0010 | 2 | 0010 | YES |
| 3 | 0011 | 6 | 0110 | NO |
| 4 | 0100 | 1 | 0001 | NO |
| 5 | 0101 | 5 | 0101 | YES |
| 6 | 0110 | 3 | 0011 | NO |
| 7 | 0111 | 7 | 0111 | YES |
| 8 | 1000 | 8 | 1000 | YES |
| 9 | 1001 | 12 | 1100 | NO |
| 10 | 1010 | 10 | 1010 | YES |
| 11 | 1011 | 14 | 1110 | NO |
| 12 | 1100 | 9 | 1001 | NO |
| 13 | 1101 | 13 | 1101 | YES |
| 14 | 1110 | 11 | 1011 | NO |
| 15 | 1111 | 15 | 1111 | YES |

*Table 12 – Colour palette error*

By looking at the bit patterns or bit planes, it can be seen bits 0 and 2 are around the wrong way. The Shifter IP Core has a file

```
if SH_MOD = "00" then -- Low resolution.
--     SR(3) <= YINT_D(15 - H_SHIFT);
--     SR(2) <= YINT_B(27 - H_SHIFT);
--     SR(1) <= YINT_C(23 - H_SHIFT);
--     SR(0) <= YINT_A(19 - H_SHIFT);
       SR(3) <= YINT_D(15 - H_SHIFT);
       SR(2) <= YINT_A(19 - H_SHIFT);
       SR(1) <= YINT_C(23 - H_SHIFT);
       SR(0) <= YINT_B(27 - H_SHIFT);
```

*Figure 42 – Change to Shifter*

called wf25914ip_cr_shift_reg.vhd and within that is a process called shift_out where the colour creation is made. This was modified based on the previous findings.

### 6.3

### Sound Techniques

The sound from the YM2149 IP Core is next tested. The sound generator is very simple, just 3 channels of square wave, which can be mixed with white noise and fed into envelope filters. Many programmers developed new ways of using the YM2149 to produce better sounds by carefully timed writes to the YM2149 registers.


*Figure 43 – SND Player*

The program used to test it is a freeware program called 'SND Player' written by Odd Skancke and Anders Eriksson in 2006 and comes with a few demo songs to try. After playing a few of the demo songs and comparing to a real Atari ST, it was apparent some had problems with some channels not having the right sound. One of these songs was 'Chu Chu Rocket' by Malcolm Grant which was chosen as the problem is evident while only using one sound channel. The envelope register was routed to be displayed on the four LEDs on the RaggedStone board. It was found the sound was incorrect only on some envelope shapes, namely '1010' which is a repeating sawtooth.

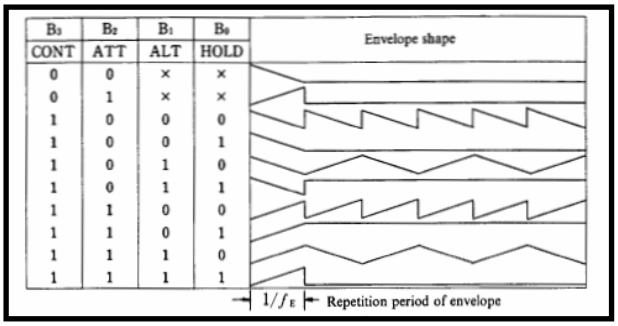



*Figure 44 – Envelope Shapes*

From looking at the process which generates the envelope shape (in the file wf2149_wave.vhd) it became apparent what the error was. On the rising slope of the envelope when it reaches the highest peak (VOL_ENV = "11111") it should start falling

back down again. The way the VHDL is structured, the signal to control the volume decrement occurs too late and the VOL_ENV rolls over. The effect is a very fast repeating square wave, rather than the desired sawtooth wave. The changes made are shown in figure 46.

```
when "1110" | "1010" =>
       if ENV_UP_DNn = '0' then
              VOL_ENV <= VOL_ENV - '1';
       else
              VOL_ENV <= VOL_ENV + '1';
       end if;

       if VOL_ENV = "00000" then
              ENV_UP_DNn := '1';
       elsif VOL_ENV = "11111" then
              ENV_UP_DNn := '0';
       end if;
```

*Figure 45 – Old envelope generator*

Some of the special effect techniques used to achieve better sounds have been given names like Sync-Buzzer, Digidrum and Sid Voice [39].

```
when "1110" | "1010" =>
       if ENV_UP_DNn = '0' then
              VOL_ENV <= VOL_ENV - '1';
       else
              VOL_ENV <= VOL_ENV + '1';
       end if;

       if VOL_ENV = "00001" then
              ENV_UP_DNn := '1';
       elsif VOL_ENV = "11110" then
              ENV_UP_DNn := '0';
       end if;
```

*Figure 46 – New envelope generator*

## 6.4

## Software Over Scan

Although not deemed as necessary, using software that attempts to use the hardware in ways that were not specified by Atari is a good test of compatibility. Just like the special effects for the sound chip, as mentioned in paragraph 6.2 there were also techniques to gain special graphics. One of these is software over scanning. The Atari ST low resolution mode is 320 x 200 pixels, but in fact displays a lot more in the form of borders around the working screen area as shown in figure 47.

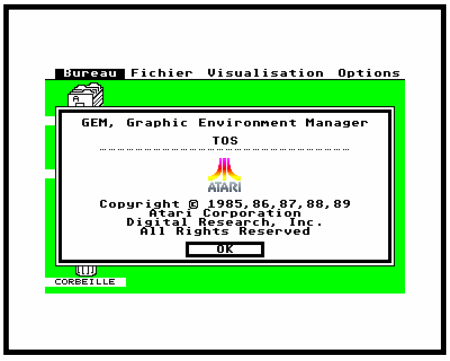



*Figure 47 – Screen borders*

It was discovered in 1988 by a team of people called ‘TNT’ that by switching graphics modes (ST Low/Medium/High or 50/60Hz) on certain line numbers the graphics sub system can be fooled into displaying graphics within the borders.

A good piece of test software was found called ‘Hallucinations’ released by RG in 2003 (although the over scan technique is much older). The current implementation of the Atari ST failed trying to run this software, with the screen borders remaining intact. From looking at the hardware the DE (Display Enable) signal generated by the Glue informs the Shifter and MMU when to display the graphics, otherwise they will display the border. Inside the Glue IP Core (file wf25915ip_video_timing.vhd), there are two main counters, one for the horizontal position and the other for the vertical line number. The DE signal is controlled by relational operators (less than and greater than) as shown in Appendix N and this precisely where the problem stems from. In a real Atari ST, if the software switches to a different screen mode at precisely the right time, the Glue then misses the qualifier to end the DE signal, and it remains active for longer displaying more pixels from RAM. Because the IP Core of the Glue uses less than and greater than, this doesn’t work. After changing all these to simple equality relation operators, and basing all the timings on a document by Dr Sengan Baring-Gould published in the French ‘ST Magazine’ in 1991 the system now correctly displayed the over scan technique [40]! Another point that backs up the use of equality operators is that they require a lot less transistors, and this would be how the original Atari semiconductors would have been constructed. As well as changing the generation of the DE signal, the V Sync, H Sync and video Blanking were changed.

# Future Additions and Possibilities

**7.1**

**Floppy Drive Emulation**

Nowadays floppy drives are being phased out. Many PC manufactures don't supply floppy drives, and Apple Inc scrapped the floppy drive in 1998 with introduction of iMac. It is quite easy to see why after implementing the floppy drive and reading the Western Digital WD1772 data sheet. Many commands given to the floppy drive have to be followed by delays. It is advised to wait for five Index Pulses after the Spin Up command before reading data. At a disk revolution of 300 rpm, five index pulses are equal to one second. More delays are needed when stepping the floppy drive head to a different track, 15ms for the WD1772. 15ms in computer terms is a long time, the MC68000 can execute up to 30,000 instructions in that time.

There exists a project called hXc (that replaces the physical floppy drive and allows a host computer to mimic a floppy drive [41]. Going one step further it would be possible to remove the WD1772 floppy controller altogether, with an image of a floppy disk stored on a removable media like an SD (Secure Digital) Card communicating directly to the DMA bus. The delays could be added for possible compatibility problems, or allowed to run at full speed.

**7.2**

**MIDI**

Midi was included as standard from the very first Atari St throughout the entire series and played an important role in establishing the computer as a serious computer for music production. In the Atari ST MIDI is implemented with another ACIA MC6850 as used to communicate with the keyboard. It uses a 31250 baud serial protocol over a current loop and is optically isolated. The MIDI could be provided with another instance of the MC6850 IP Core. The game port to MIDI adapters that is commonly available for PCs could easily be used to provide the electrical specification MIDI needs. The game port is

a 15 way D-Type connector and the panel cut out is provided on many PC cases, unlike that of the 5 pin DIN connectors MIDI uses.

### 7.3
### IDE

Although there exists a simple IDE interface for the Atari ST that is compatible to the Atari Falcons IDE port, it only works in programmed I/O mode. This is where the CPU has to do the work of moving the data to and from the IDE bus to RAM. Although this is fine for most small chunks of data, large files will inevitably tie up the CPU stopping it from doing more useful processing. A solution could exist in using the original DMA component with a bridge layer to IDE protocol. This would provide complete software compatibility, appearing as a DMA device to the Atari ST but using plentiful and inexpensive IDE drives. The conversion from Atari's own protocol of the DMA bus, ACSI could be converted to IDE internally as a custom IP Core. It also opens up the possibility for more than one IDE port, as ACSI protocol can support 7 devices.

### 7.4
### Unification of mass storage

In the design there were a lot of areas and different device technologies for storage of data.

- Floppy drive using 3.5" floppy disks
- Configuration of the FPGA held in custom Xilinx serial PROM
- Operating System held in parallel FLASH memory
- PIC microcontroller firmware held within itself

Each of the above could be contained in one single device like an inexpensive SD Card. The PIC microcontroller could contain firmware to read a FAT 32 formatted SD Card to program its self with new firmware, program the FPGA configuration and store the Operating System in a portion of RAM. The floppy drive image could be retrieved on

request from the Floppy Drive Controller with the PIC acting as a bridge. This greatly reduces cost and component count at the expensive of development time to guarantee correct functionality [42], [43].

### 7.5
### Reconfigurable systems

The previous paragraph leads nicely onto the ability for reconfigurable systems. By allowing the PIC to read from an SD Card, it is possible to store multiple versions for each part of the system. This could allow the system to change functionality entirely within seconds. Using an SD Card also removes the need for special programming hardware for the various parts onboard and simplifies any updates needed to a system. It would be as simple as inserting the SD Card into a PC to copy new firmware packages on [44], [45].

### 7.6
### Commercial viability

There is scope for a design like this to be made in to a sellable item. By having a design that supports many projects needs, and a framework wrapper to ease migration and porting from other development boards it could attract many designers. From looking at other FPGA development boards and needs for projects the following specification ahs been concluded.

- Composite Video/ S Video for connection to domestic Televisions
- VGA connection with high quality DAC for high resolution monitors
- SD Card for storage of data
- One fixed clock, one software programmable for pixel clocks etc
- PS/2 mouse and keyboard ports
- LCD header pins for embedded designs
- Ethernet port
- Audio Codec for sound in and out

- IDE interface port
- CPU expansion slot to allow for different CPU architectures and/or addition IO
- SDRAM main memory
- PIC Microcontroller for standby operation

A design with these features opens the market to many diverse applications, not just an Atari ST design. The addition of a wrapper for using the inexpensive Analogue and Digital Sony Playstation controllers could very be useful for the control in robotics.

Even with an Atari ST design, it is possible to run the 68k Debian Linux port, or uClinux are even Atari's own UNIX derivative, MultiTOS/MINT.

# Chapter 8

# Summary

**8.1**

I personally found implementing an entire computer system in an FPGA highly challenging and incredibly rewarding. From the overall design blocks to the intricate details at logic gate level presented many opportunities for problem solving.

Every IP Core (bar the ACIA) of the system had to be studied in great detail to understand the inner workings and fixing numerous problems. Not every detail that was changed could be mentioned, as some of them took so long to discover the root of the problem and the length of detailing the solution. The best example of this is the Floppy Drive Controller IP Core. Even in its current state it is not fully functional, it has problems with games that have complex copy protection techniques or formatted with non standard numbers of sectors per track. Writing to a disk was not even attempted and was disabled in hardware for fear of ruining many disks. The FDC IP Core is by far the most complex IP Core in the design, having to deal with a MFM encoded bit stream from the floppy drive that varies every so slightly in bit rate as the disk drive motor RPM fluctuates.

One point I found from doing this project is understanding how far computing has come, the amount of complexities there are in modern computer chipsets, graphics and micro processors. One thing I feel is that by understanding the past, an insight into the future of computing can be seen clearer and many ideas will come back around.

A few things that could have helped greatly would be better access to tools. Having something along the lines of a logic analyser embedded into the design, like Xilinx's own ChipScope would have been incredibly useful. A normal logic analyser would help to some extent, but every time you want to view a different internal signal it requires a rebuild of the project. On that point, even on my relatively new computer the process of

building the project and programming through the JTAG interface took approximately 20 minutes. In total there were approximately 15000 lines of VHDL code.

I found the internet a great resource of information, as there were very few books published about the hardware of the Atari ST. Much of the information has come from archives of past magazines articles and documents written about certain aspects of the hardware.

Although the design did not feature the CPU as an IP Core, I feel this was a wise decision as it was the one part of the design that could be trusted as working from the very start. Trying to debug a CPU IP Core along with everything else would have been incredibly time consuming.

I've learnt a lot about the inner workings of FPGAs, and the VHDL language. Most importantly is that FPGAs are in some ways like a group of individual integrated circuits. It is just as important to make sure that the interconnections between these blocks of logic are constrained to certain paths, distances and delays as it is with a traditional design.

Overall I've found systems on chip incredibly interesting and something of real importance for the future. Their ability is to make designs smaller, consume less power and most importantly faster.

# References

All online references sited 29th April 2008.

[1] Old-Computers.com The Museam. [Online].
Available: http://www.old-computers.com/museum/company.asp?st=1&m=10

[2] Wikipedia. (unknown). TV Boy. [Online].
Available: http://en.wikipedia.org/wiki/TV_Boy

[3] Howard Wen. (2007). Curt Vendel: The Escapist Interview. [Online].
Available: http://www.escapistmagazine.com/articles/view/issues/issue_100/555-Curt-Vendel-The-Escapist-Interview

[4] Benjamin J Heckendorn. (2006). Still looking for NES on a Chips . [Online].
Available: http://benheck.com/01-05-2006/still-looking-for-nes-on-a-chips-noacs

[5] Rieks Warendorp Torringa/ Sander Zuidema. (2006). Bazix Homepage. [Online]
Available: http://www.bazix.nl

[6] Dennis Van Weeren. (2008). MiniMig Homepage. [Online].
Available: http://home.hetnet.nl/~weeren001/

[7] Old-Computers.com The Museam. [Online].
Available: http://en.wikipedia.org/wiki/Jeri_Ellsworth

[8] JOHN MARKOFF. (2004). The New York Times: A Toy With a Story. [Online].
Available: http://www.nytimes.com/2004/12/20/technology/20joystick.html

[9] IGN. (unknown). Atari Inc. (1972-1984). [Online]
Available: http://uk.games.ign.com/objects/764/764953.html

[10] Wikipedia. (2008). Atari ST. [Online].
Available: http://en.wikipedia.org/wiki/Atari_ST

[11] Bob Lash. (2002). ATARI ST Prototype. [Online].
Available: http://www.bambi.net/atari/atari_st_prototype.html

[12] K.Gerits, L.Englisch, R.Bruckmann, “The ST Operating System” in *Atari ST Internals*, 3rd ed. , Miami: Abacus Software Inc., May 1988, pp. 105-106

[13] Hans-Dieter Jankowski, Julian F.Reschke,Dietmar Rabich, “Der “Power-UP Ablaufplan” in *Atari ST Profibuch*, 2nd ed. , Düsseldorf: Sybex-Verlag, 1989, pp. 917-919

[14] K.Gerits, L.Englisch, R.Bruckmann, “The BIOS Listing” in *Atari ST Internals*, 3rd ed. , Miami: Abacus Software Inc., May 1988, pp. 271-461

[15] Kevin Morris. (2005). World's Best FPGA Article. [Online].
Available: http://www.fpgajournal.com/articles_2005/20050510_worldsbest.htm

[16] Peter Clarke. (1999). Early users of IP cores could gain an edge from design reuse. [Online].
Available: http://www.eetimes.com/story/OEG19990622S0014

[17] Xilinx, Inc. (unknown). ISE WebPACK 8.2i FAQ. [Online].
Available: http://www.xilinx.com/ise/logic_design_prod/webpack_faq.htm#3a

[18] Gennadiy Shvets. (2007). Motorola 68000 microprocessor family. [Online].
Available: http://www.cpu-world.com/CPUs/68000/

[19] John Townsend/Richard Davey. (unknown). Town's Little Guide to TOS Revisions. [Online].
Available: http://www.atari.st/content.php?type=t&file=toslist

[20] Micron Technology, Inc. (2000). Migrating from FPM/EDO To SDRAM. [Online].
Available: http://download.micron.com/pdf/technotes/ZT07.pdf

[21] Freescale Semiconductor, Inc. (1997). Addendum to MC68000 User Manual. [Online].
Available: http://www.freescale.com/files/32bit/doc/ref_manual/M68000UMAD.pdf

[22] Ken Chapman. (2007). M29DW323DT ST Microelectronics FLASH Programmer. [Online].
Available: http://www.xilinx.com/products/boards/s3astarter/reference_designs.htm

[23] David Small. (1986). Probing the FDC, START Vol.1,No.2,Page 96. [Online].
Available: http://www.atarimagazines.com/startv1n2/ProbingTheFDC.html

[24] Didier Méquignon, Laurent Favard. (2005). Atari Eiffel 3 Interface PS/2. [Online].
Available: http://pagesperso-orange.fr/didierm/eiffel-e.htm

[25] Pera Putnik. (1998). Atari IDE disk interface. [Online].
Available: http://members.tripod.com/~piters/atari/astide.htm

[26] Mike J. (2006). VGA Display Test. [Online].
Available: http://home.freeuk.com/fpgaarcade/displaytest.htm

[27] Freescale Semiconductor, Inc. (2006). MC68000 User Manual. [Online].
Available: http://www.freescale.com/files/32bit/doc/ref_manual/M68000UM.pdf

[28] Sam Duncan. (2003). Article 53214. [Online].
Available: www.fpga-faq.com/archives/53100.htm

[29] Craig Abramson. (unknown). Determining Clock Skew when the Virtex DLL Drives Multiple Copies of a Clock Off Chip. [Online].
Available:
http://www.nalanda.nitc.ac.in/industry/appnotes/xilinx/documents/xcell/xl32/xl32_53.pdf

[30] Xilinx, Inc. (2000). Using Delay-Locked Loops in Spartan-II FPGAs. [Online].
Available: http://www.xilinx.com/support/documentation/application_notes/xapp174.pdf

[31] Brian Jackson. (2007). Improving FPGA on PCB Integration with PlanAhead Design and Analysis Tool. [Online].
Available: http://www.fpgajournal.com/whitepapers_2007/q2_xilinx_1.htm

[32] David Gahris. (1995). WD1772 Programming information. [Online].
Available: http://www.buchty.net/ensoniq/files/wd1772.txt

[33] Greg Cook. (2005). Register summary for Western Digital FDC. [Online].
Available: http://homepages.tesco.net/~rainstorm/fdc-combined.htm

[34] Western Digital Corporation. (unknown). WD177X-00 Floppy Disk Controller. [Online].
Available: http://dev-docs.atariforge.org/files/WD1772.pdf

[35] Andries Brouwer. (2004). Keyboard scancodes. [Online].
Available: http://www.win.tue.nl/~aeb/linux/kbd/scancodes.html

[36] Andrew J McCubbin. (1998). Sony PlayStation Controller Information. [Online].
Available: http://www.gamesx.com/controldata/psxcont/psxcont.htm

[37] Dan Hollis. (1994). Atari ST/STe/MSTe/TT/F030 Hardware Register Listing. [Online].
Available: http://dev-docs.atariforge.org/files/hardware.zip

[38] CompactFlash Association. (2008). CF and CompactFlash FAQ. [Online].
Available: http://www.compactflash.org/faqs/faq.htm

[39] Arnaud Carré. (unknown). YM2149 Special Effects. [Online].
Available: http://leonard.oxg.free.fr/ymformat.html

[40] Dr Sengan Baring-Gould. (1991). Overscan Techniques. [Online].
Available: http://alive.atari.org/alive9/ovrscn1.php

[41] Jean François. (2008). Emulateur de lecteur de disquette. [Online].
Available: http://jeanfrancoisdelnero.free.fr/floppy_drive_emulator/

[42] Claudi Chiculita. (2007). Tiny PIC bootloader. [Online].
Available: http://www.etc.ugal.ro/cchiculita/software/picbootloader.htm

[43] Philip Freidin. (2003). Configuring an FPGA from a processor. [Online].
Available: http://www.fpga-faq.com/FAQ_Pages/0038_Config_FPGA_from_a_processor.htm

[44] Lauro Rizzatti. (2002). Hardware emulation for everyone. [Online].
Available:http://www.eetimes.com/news/design/columns/eda/showArticle.jhtml?articleID=17407881

[45] Wikipedia. (2008). Hardware emulation. [Online].
Available: http://en.wikipedia.org/wiki/Hardware_emulation

# Appendices

# A

MAIN.UCF constraints file
MAIN.VHD component file

```
#PACE: Start of Constraints generated by PACE

#PACE: Start of PACE I/O Pin Assignments
NET "CLK_PCI"  LOC = "A11"  ;
NET "EEPROM_SCL"  LOC = "U7"  ;
NET "EEPROM_SDA"  LOC = "U10"  ;
NET "EEPROM_WP"  LOC = "V7"  ;
NET "FLASH_A<0>"  LOC = "Y10"  ;
NET "FLASH_A<10>"  LOC = "U12"  ;
NET "FLASH_A<11>"  LOC = "AB15"  ;
NET "FLASH_A<12>"  LOC = "AB9"  ;
NET "FLASH_A<13>"  LOC = "AB14"  ;
NET "FLASH_A<14>"  LOC = "AA13"  ;
NET "FLASH_A<15>"  LOC = "AB10"  ;
NET "FLASH_A<16>"  LOC = "AB11"  ;
NET "FLASH_A<17>"  LOC = "AB13"  ;
NET "FLASH_A<18>"  LOC = "Y12"  ;
NET "FLASH_A<1>"  LOC = "W10"  ;
NET "FLASH_A<2>"  LOC = "V10"  ;
NET "FLASH_A<3>"  LOC = "W9"  ;
NET "FLASH_A<4>"  LOC = "W8"  ;
NET "FLASH_A<5>"  LOC = "AB8"  ;
NET "FLASH_A<6>"  LOC = "AA8"  ;
NET "FLASH_A<7>"  LOC = "AA9"  ;
NET "FLASH_A<8>"  LOC = "V9"  ;
NET "FLASH_A<9>"  LOC = "AA15"  ;
NET "FLASH_CE"  LOC = "V14"  ;
NET "FLASH_IO<0>"  LOC = "AA10"  ;
NET "FLASH_IO<1>"  LOC = "W11"  ;
NET "FLASH_IO<2>"  LOC = "Y11"  ;
NET "FLASH_IO<3>"  LOC = "U11"  ;
NET "FLASH_IO<4>"  LOC = "W13"  ;
NET "FLASH_IO<5>"  LOC = "V13"  ;
NET "FLASH_IO<6>"  LOC = "Y13"  ;
NET "FLASH_IO<7>"  LOC = "W14"  ;
NET "FLASH_OE"  LOC = "U13"  ;
NET "FLASH_WE"  LOC = "W12"  ;
NET "J01_2"  LOC = "AA12"  ;
NET "J01_3"  LOC = "AB12"  ;
NET "J01_4"  LOC = "V16"  ;
NET "J01_5"  LOC = "W16"  ;
NET "J02_2"  LOC = "V8"  ;
NET "J02_3"  LOC = "Y6"  ;
NET "J02_4"  LOC = "AA6"  ;
NET "J02_5"  LOC = "U6"  ;
NET "J2_1"  LOC = "B19"  ;
```

```
NET "J2_10"  LOC = "C17"  ;
NET "J2_11"  LOC = "B17"  ;
NET "J2_12"  LOC = "E15"  ;
NET "J2_13"  LOC = "D17"  ;
NET "J2_14"  LOC = "E13"  ;
NET "J2_15"  LOC = "D15"  ;
NET "J2_16"  LOC = "F13"  ;
NET "J2_17"  LOC = "D14"  ;
NET "J2_18"  LOC = "A15"  ;
NET "J2_19"  LOC = "F12"  ;
NET "J2_20"  LOC = "B14"  ;
NET "J2_21"  LOC = "B15"  ;
NET "J2_22"  LOC = "F16"  ;
NET "J2_23"  LOC = "A14"  ;
NET "J2_24"  LOC = "D13"  ;
NET "J2_25"  LOC = "F17"  ;
NET "J2_26"  LOC = "A13"  ;
NET "J2_27"  LOC = "C13"  ;
NET "J2_28"  LOC = "E12"  ;
NET "J2_29"  LOC = "B13"  ;
NET "J2_3"  LOC = "A19"  ;
NET "J2_31"  LOC = "A12"  ;
NET "J2_32"  LOC = "D12"  ;
NET "J2_33"  LOC = "A9"  ;
NET "J2_34"  LOC = "B12"  ;
NET "J2_35"  LOC = "B10"  ;
NET "J2_36"  LOC = "C10"  ;
NET "J2_37"  LOC = "A8"  ;
NET "J2_38"  LOC = "B9"  ;
NET "J2_39"  LOC = "F11"  ;
NET "J2_4"  LOC = "C18"  ;
NET "J2_40"  LOC = "E10"  ;
NET "J2_41"  LOC = "F10"  ;
NET "J2_42"  LOC = "E9"  ;
NET "J2_43"  LOC = "F9"  ;
NET "J2_44"  LOC = "B8"  ;
NET "J2_45"  LOC = "D7"  ;
NET "J2_46"  LOC = "E7"  ;
NET "J2_47"  LOC = "C6"  ;
NET "J2_48"  LOC = "B6"  ;
NET "J2_49"  LOC = "E6"  ;
NET "J2_5"  LOC = "D18"  ;
NET "J2_50"  LOC = "D6"  ;
NET "J2_51"  LOC = "A5"  ;
NET "J2_52"  LOC = "B5"  ;
NET "J2_6"  LOC = "B18"  ;
NET "J2_8"  LOC = "E17"  ;
NET "J2_9"  LOC = "A18"  ;
NET "JL1_10"  LOC = "M21"  ;
NET "JL1_11"  LOC = "K19"  ;
NET "JL1_12"  LOC = "K20"  ;
NET "JL1_13"  LOC = "K22"  ;
NET "JL1_14"  LOC = "K21"  ;
NET "JL1_15"  LOC = "G19"  ;
NET "JL1_16"  LOC = "F19"  ;
NET "JL1_17"  LOC = "F20"  ;
NET "JL1_18"  LOC = "F21"  ;
```

```
NET "JL1_19"  LOC = "E21"  ;
NET "JL1_2"  LOC = "Y21"  ;
NET "JL1_20"  LOC = "E22"  ;
NET "JL1_3"  LOC = "T22"  ;
NET "JL1_4"  LOC = "T21"  ;
NET "JL1_5"  LOC = "T18"  ;
NET "JL1_6"  LOC = "R18"  ;
NET "JL1_7"  LOC = "M17"  ;
NET "JL1_8"  LOC = "M18"  ;
NET "JL1_9"  LOC = "M22"  ;
NET "JL2_1"  LOC = "W21"  ;
NET "JL2_10"  LOC = "T17"  ;
NET "JL2_11"  LOC = "M19"  ;
NET "JL2_12"  LOC = "M20"  ;
NET "JL2_13"  LOC = "L20"  ;
NET "JL2_14"  LOC = "L19"  ;
NET "JL2_15"  LOC = "G18"  ;
NET "JL2_16"  LOC = "G17"  ;
NET "JL2_17"  LOC = "F18"  ;
NET "JL2_18"  LOC = "E18"  ;
NET "JL2_19"  LOC = "D19"  ;
NET "JL2_2"  LOC = "W20"  ;
NET "JL2_20"  LOC = "D20"  ;
NET "JL2_3"  LOC = "V22"  ;
NET "JL2_4"  LOC = "V21"  ;
NET "JL2_5"  LOC = "V19"  ;
NET "JL2_6"  LOC = "W19"  ;
NET "JL2_7"  LOC = "V20"  ;
NET "JL2_8"  LOC = "U19"  ;
NET "JL2_9"  LOC = "U18"  ;
NET "JL3_10"  LOC = "L21"  ;
NET "JL3_11"  LOC = "L22"  ;
NET "JL3_12"  LOC = "L18"  ;
NET "JL3_13"  LOC = "L17"  ;
NET "JL3_14"  LOC = "G21"  ;
NET "JL3_15"  LOC = "G22"  ;
NET "JL3_16"  LOC = "E20"  ;
NET "JL3_17"  LOC = "E19"  ;
NET "JL3_18"  LOC = "D22"  ;
NET "JL3_19"  LOC = "D21"  ;
NET "JL3_2"  LOC = "Y22"  ;
NET "JL3_20"  LOC = "C22"  ;
NET "JL3_3"  LOC = "W22"  ;
NET "JL3_4"  LOC = "U21"  ;
NET "JL3_5"  LOC = "U20"  ;
NET "JL3_6"  LOC = "N19"  ;
NET "JL3_7"  LOC = "N20"  ;
NET "JL3_8"  LOC = "N21"  ;
NET "JL3_9"  LOC = "N22"  ;
NET "JR1_10"  LOC = "L2"  ;
NET "JR1_11"  LOC = "L1"  ;
NET "JR1_12"  LOC = "K3"  ;
NET "JR1_13"  LOC = "K4"  ;
NET "JR1_14"  LOC = "G1"  ;
NET "JR1_15"  LOC = "G2"  ;
NET "JR1_16"  LOC = "D3"  ;
NET "JR1_17"  LOC = "D2"  ;
```

```
NET "JR1_18"  LOC = "D1"  ;
NET "JR1_19"  LOC = "C1"  ;
NET "JR1_2"  LOC = "W4"  ;
NET "JR1_20"  LOC = "C2"  ;
NET "JR1_3"  LOC = "W3"  ;
NET "JR1_4"  LOC = "V3"  ;
NET "JR1_5"  LOC = "V4"  ;
NET "JR1_6"  LOC = "N4"  ;
NET "JR1_7"  LOC = "N3"  ;
NET "JR1_8"  LOC = "N2"  ;
NET "JR1_9"  LOC = "N1"  ;
NET "JR2_1"  LOC = "W1"  ;
NET "JR2_10"  LOC = "T6"  ;
NET "JR2_11"  LOC = "M4"  ;
NET "JR2_12"  LOC = "M3"  ;
NET "JR2_13"  LOC = "L3"  ;
NET "JR2_14"  LOC = "L4"  ;
NET "JR2_15"  LOC = "H5"  ;
NET "JR2_16"  LOC = "G5"  ;
NET "JR2_17"  LOC = "G6"  ;
NET "JR2_18"  LOC = "F5"  ;
NET "JR2_19"  LOC = "E4"  ;
NET "JR2_2"  LOC = "W2"  ;
NET "JR2_20"  LOC = "D4"  ;
NET "JR2_3"  LOC = "V5"  ;
NET "JR2_4"  LOC = "U5"  ;
NET "JR2_5"  LOC = "V2"  ;
NET "JR2_6"  LOC = "V1"  ;
NET "JR2_7"  LOC = "U4"  ;
NET "JR2_8"  LOC = "T4"  ;
NET "JR2_9"  LOC = "T5"  ;
NET "JR3_10"  LOC = "M2"  ;
NET "JR3_11"  LOC = "L5"  ;
NET "JR3_12"  LOC = "L6"  ;
NET "JR3_13"  LOC = "K1"  ;
NET "JR3_14"  LOC = "K2"  ;
NET "JR3_15"  LOC = "F4"  ;
NET "JR3_16"  LOC = "E3"  ;
NET "JR3_17"  LOC = "F2"  ;
NET "JR3_18"  LOC = "F3"  ;
NET "JR3_19"  LOC = "E2"  ;
NET "JR3_2"  LOC = "Y1"  ;
NET "JR3_20"  LOC = "E1"  ;
NET "JR3_3"  LOC = "U2"  ;
NET "JR3_4"  LOC = "U3"  ;
NET "JR3_5"  LOC = "T1"  ;
NET "JR3_6"  LOC = "T2"  ;
NET "JR3_7"  LOC = "M6"  ;
NET "JR3_8"  LOC = "M5"  ;
NET "JR3_9"  LOC = "M1"  ;
NET "LED1_1"  LOC = "AA17"  ;
NET "LED1_11"  LOC = "AA18"  ;
NET "LED1_12"  LOC = "Y18"  ;
NET "LED1_13"  LOC = "V18"  ;
NET "LED1_14"  LOC = "AB20"  ;
NET "LED1_15"  LOC = "W18"  ;
NET "LED1_16"  LOC = "AA20"  ;
```

```
NET "LED1_2"  LOC = "U17"  ;
NET "LED1_3"  LOC = "Y17"  ;
NET "LED1_4"  LOC = "V17"  ;
NET "LED1_5"  LOC = "AB18"  ;
NET "LED1_6"  LOC = "U16"  ;
NET "LED1_7"  LOC = "W17"  ;
NET "LED1_8"  LOC = "U14"  ;
NET "LED2"  LOC = "AB5"  ;
NET "LED3"  LOC = "AA5"  ;
NET "LED4"  LOC = "AA4"  ;
NET "LED5"  LOC = "AB4"  ;
NET "S1"  LOC = "AA3" | PULLUP ;
NET "S2"  LOC = "Y4" | PULLUP ;
NET "TEMP_A<0>"  LOC = "V12"  ;
NET "TEMP_A<1>"  LOC = "V11"  ;
NET "TEMP_A<2>"  LOC = "V6"  ;
NET "TEMP_INT"  LOC = "W5"  ;
NET "TEMP_SCL"  LOC = "Y5"  ;
NET "TEMP_SDA"  LOC = "W6"  ;
NET "USER_CLK"  LOC = "AA11"  ;

#PACE: Start of PACE Area Constraints

#PACE: Start of PACE Prohibit Constraints

#PACE: End of Constraints generated by PACE



library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.std_logic_ARITH.ALL;
use IEEE.std_logic_UNSIGNED.ALL;

entity main is
    Port(
                -- JL1 Header (pin 1 0v) BANK3&2
                   JL1_2 : in std_logic;
                   JL1_3 : in std_logic;
                   JL1_4 : in std_logic;
                   JL1_5 : in std_logic;
                   JL1_6 : in std_logic;
                   JL1_7 : in std_logic;
                   JL1_8 : in std_logic;
                   JL1_9 : in std_logic;
                   JL1_10 : in std_logic;
                   JL1_11 : in std_logic;
                   JL1_12 : in std_logic;
                   JL1_13 : in std_logic;
                   JL1_14 : in std_logic;
                   JL1_15 : in std_logic;
                   JL1_16 : in std_logic;
                   JL1_17 : in std_logic;
                   JL1_18 : in std_logic;
                   JL1_19 : in std_logic;
                   JL1_20 : in std_logic;
```

```
-- JL2 Header BANK3&2
JL2_1 : in std_logic; -- CCLK with R999
JL2_2 : in std_logic;
JL2_3 : in std_logic;
JL2_4 : in std_logic;
JL2_5 : in std_logic;
JL2_6 : in std_logic;
JL2_7 : in std_logic;
JL2_8 : in std_logic;
JL2_9 : in std_logic;
JL2_10 : in std_logic;
JL2_11 : in std_logic;
JL2_12 : in std_logic;
JL2_13 : in std_logic;
JL2_14 : in std_logic;
JL2_15 : in std_logic;
JL2_16 : in std_logic;
JL2_17 : in std_logic;
JL2_18 : in std_logic;
JL2_19 : in std_logic;
JL2_20 : in std_logic;

-- JL3 Header (pin 1 3.3v) BANK3&2
JL3_2 : in std_logic;
JL3_3 : in std_logic;
JL3_4 : in std_logic;
JL3_5 : in std_logic;
JL3_6 : in std_logic;
JL3_7 : in std_logic;
JL3_8 : in std_logic;
JL3_9 : in std_logic;
JL3_10 : in std_logic;
JL3_11 : in std_logic;
JL3_12 : in std_logic;
JL3_13 : in std_logic;
JL3_14 : in std_logic;
JL3_15 : in std_logic;
JL3_16 : in std_logic;
JL3_17 : in std_logic;
JL3_18 : in std_logic;
JL3_19 : in std_logic;
JL3_20 : in std_logic;

-- JR1 Header (pin 1 0v) BANK6&7
JR1_2 : in std_logic;
JR1_3 : in std_logic;
JR1_4 : in std_logic;
JR1_5 : in std_logic;
JR1_6 : in std_logic;
JR1_7 : in std_logic;
JR1_8 : in std_logic;
JR1_9 : in std_logic;
JR1_10 : in std_logic;
JR1_11 : in std_logic;
JR1_12 : in std_logic;
JR1_13 : inout std_logic;
```

```
JR1_14 : in std_logic;
JR1_15 : in std_logic;
JR1_16 : in std_logic;
JR1_17 : in std_logic;
JR1_18 : in std_logic;
JR1_19 : in std_logic;
JR1_20 : in std_logic;

-- JR2 Header BANK6&7
JR2_1 : in std_logic;
JR2_2 : in std_logic;
JR2_3 : in std_logic;
JR2_4 : in std_logic;
JR2_5 : in std_logic;
JR2_6 : in std_logic;
JR2_7 : in std_logic;
JR2_8 : in std_logic;
JR2_9 : in std_logic;
JR2_10 : in std_logic;
JR2_11 : in std_logic;
JR2_12 : in std_logic;
JR2_13 : in std_logic;
JR2_14 : in std_logic;
JR2_15 : in std_logic;
JR2_16 : in std_logic;
JR2_17 : in std_logic;
JR2_18 : in std_logic;
JR2_19 : in std_logic;
JR2_20 : in std_logic;

-- JR3 Header (pin 1 3.3v) BANK6&7
JR3_2 : in std_logic;
JR3_3 : in std_logic;
JR3_4 : in std_logic;
JR3_5 : in std_logic;
JR3_6 : in std_logic;
JR3_7 : in std_logic;
JR3_8 : in std_logic;
JR3_9 : in std_logic;
JR3_10 : in std_logic;
JR3_11 : in std_logic;
JR3_12 : in std_logic;
JR3_13 : in std_logic;
JR3_14 : in std_logic;
JR3_15 : in std_logic;
JR3_16 : out std_logic;
JR3_17 : out std_logic;
JR3_18 : out std_logic;
JR3_19 : inout std_logic;
JR3_20 : inout std_logic;

-- J01 (pin 1 0v) R99 links pin2&3
J01_2 : in std_logic;
J01_3 : in std_logic;
J01_4 : in std_logic;
J01_5 : in std_logic;
```

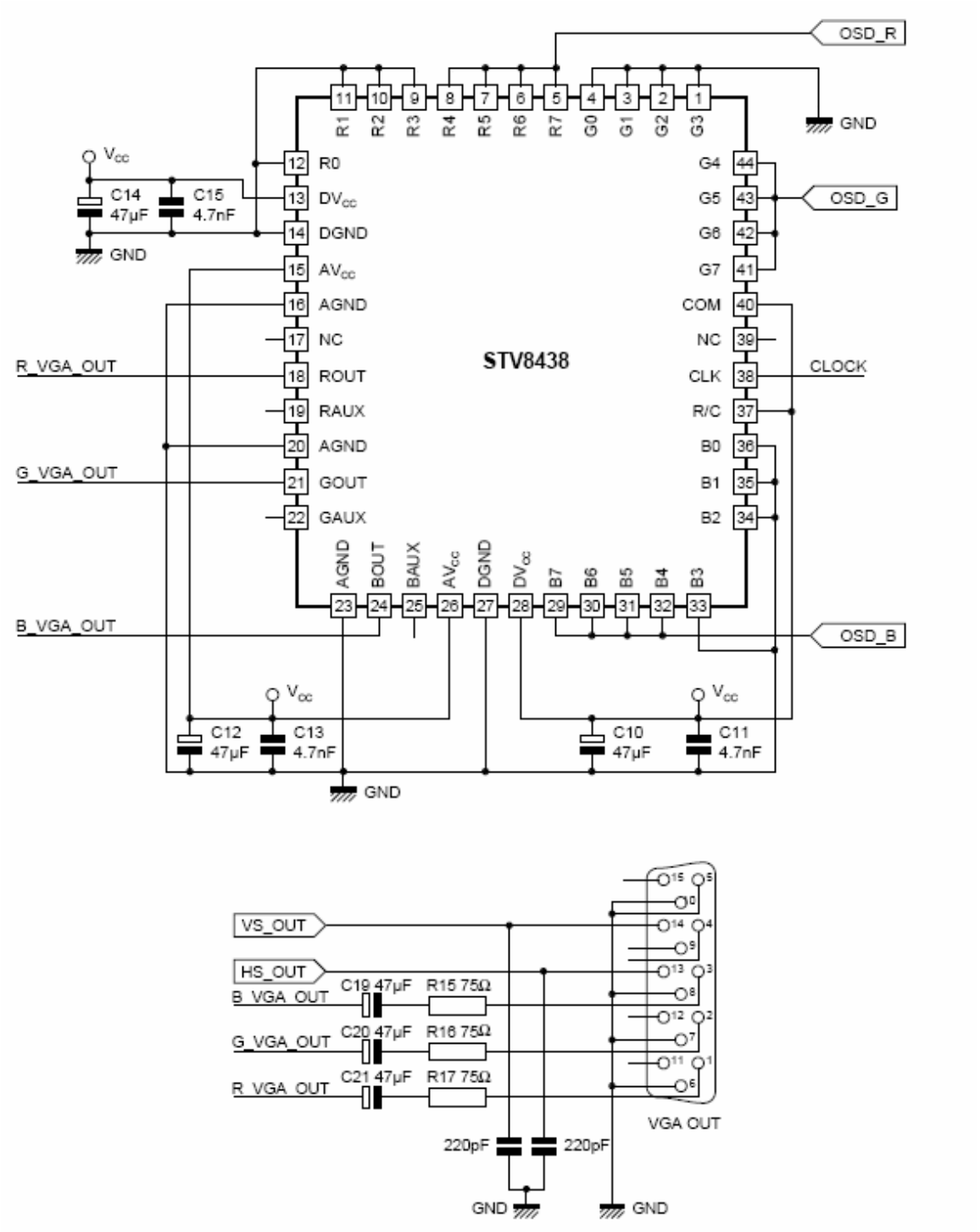
```
          -- J02 (pin 1 3.3v)
          J02_2 : in std_logic;
          J02_3 : in std_logic;
          J02_4 : in std_logic;
          J02_5 : in std_logic;

          -- JB1-4 ????


---------- PCI IO -----------------------

          -- J2 (pin 2 is just PCI clock)
          --    last pin is NC
          J2_1 : in std_logic;
        --J2_2 : out std_logic;
          J2_3 : in std_logic;
          J2_4 : in  std_logic;
          J2_5 : in  std_logic;
          J2_6 : in  std_logic;
        --J2_7 : out std_logic; NC
          J2_8 : in  std_logic;
          J2_9 : out std_logic;
          J2_10 : out std_logic;
          J2_11 : in  std_logic;
          J2_12 : in  std_logic;
          J2_13 : in  std_logic;
          J2_14 : in  std_logic;
          J2_15 : in  std_logic;
          J2_16 : in  std_logic;
          J2_17 : out std_logic;
          J2_18 : out std_logic;
          J2_19 : in  std_logic;
          J2_20 : in  std_logic;
          J2_21 : in  std_logic;
          J2_22 : in  std_logic;
          J2_23 : in  std_logic;
          J2_24 : in  std_logic;
          J2_25 : out std_logic;
          J2_26 : out std_logic;
          J2_27 : out std_logic;
          J2_28 : out std_logic;
          J2_29 : in std_logic;
        --J2_30 : out std_logic; NC
          J2_31 : in std_logic;
          J2_32 : in std_logic;
          J2_33 : in std_logic;
          J2_34 : in std_logic;
          J2_35 : in std_logic;
          J2_36 : in std_logic;
          J2_37 : in std_logic;
          J2_38 : in std_logic;
          J2_39 : in std_logic;
          J2_40 : in std_logic;
          J2_41 : in std_logic;
          J2_42 : in std_logic;
          J2_43 : in std_logic;
          J2_44 : in std_logic;
```

```
J2_45 : in std_logic;
J2_46 : in std_logic;
J2_47 : in std_logic;
J2_48 : in std_logic;
J2_49 : in std_logic;
J2_50 : in std_logic;
J2_51 : in std_logic;
J2_52 : in std_logic;
--J2_53 : out std_logic;      NC

----------  Stuff onboard  ----------------

-- Clocks
--CCLK : in std_logic;
USER_CLK : in std_logic;
CLK_PCI : in std_logic;

-- U12 (Flash?)
FLASH_A : out std_logic_VECTOR(18 DOWNTO 0);
FLASH_WE : out std_logic;
FLASH_OE : out std_logic;
FLASH_CE : out std_logic;
FLASH_IO : in std_logic_VECTOR(7 DOWNTO 0);

-- Temp Sensor
TEMP_SDA : in std_logic;
TEMP_SCL : in std_logic;
TEMP_INT : in std_logic;
TEMP_A : in std_logic_VECTOR(2 DOWNTO 0);

-- Serial EEPROM
EEPROM_WP : in std_logic;
EEPROM_SCL : in std_logic;
EEPROM_SDA : in std_logic;

-- On board switchs (active low)
S1 : in std_logic;
S2 : in std_logic;

--  On board LEDs (active high)
LED2 : out std_logic;
LED3 : out std_logic;
LED4 : out std_logic;
LED5 : out std_logic;

-- LED display
LED1_1 : out std_logic; --Active high column 1 select
LED1_2 : out std_logic; --Active high column 2 select
LED1_3 : out std_logic; --Active low digit D select
LED1_4 : out std_logic; --Active high dots select
LED1_5 : out std_logic; --Active low digit E select
LED1_6 : out std_logic; --Active high column 3 select
LED1_7 : out std_logic; --Active low DP select
LED1_8 : out std_logic; --Active high column 4 select
LED1_11 : out std_logic; -- Active low digit F select
LED1_12 : out std_logic; -- NC???!!!
LED1_13 : out std_logic; -- Active low digit C select
```

```
                LED1_14 : out std_logic; -- Active low digit A select
                LED1_15 : out std_logic; -- Active low digit G select
                LED1_16 : out std_logic  -- Active low digit B select
                );
end main;

architecture rtl of main is


begin


end;
```

# B

Schematic of ST Microelectronics Video DAC for 3 x 4bit colour use



# C

An excerpt from the PicoBlaze source code for programming a 192K byte raw binary file to a parallel Flash memory device.

```
;******************************************************************
; Program BIN Command - Program FLASH memory with data defined in an
; BIN file
;******************************************************************

program_bin_command: CALL send_CR
                     CALL send_Waiting_MCS_file
                     LOAD s9, 00   ;load start address of programming
                     LOAD s8, 00
                     LOAD s7, 00
                     CALL program_BIN
                     CALL send_OK
                     JUMP prompt


;******************************************************************
; Program FLASH memory with data defined in an BIN file
;******************************************************************
;
; Reads the BIN file from the UART and programs the FLASH device at
; 00000 location.
;
;
; This routine will continue until an end of file record is detected.
; For each line of BIN received, the current address will be output so
; that progress can be monitored.

program_BIN:         CALL read_from_UART  ;read character

                     LOAD sB, UART_data   ; load in data

                     CALL program_byte

                     ADD s7, 01           ;increment address
                     ADDCY s8, 00
                     ADDCY s9, 00


                     COMPARE s9, 03       ; check for 196608 bytes
                     ;COMPARE s9, 04      ; check for 262144 bytes

                     JUMP NZ, program_BIN

                     RETURN               ;finnished
```

# D

Startup.vhd, a VHDL component for reset and power up reset generation.

```
----------------------------------------------------------------------------
-- Company:
-- Engineer: Lyndon Amsdon
--
-- Create Date:    00:02:39 10/10/2007
-- Design Name:
-- Module Name:    startup - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:  Holds reset line low for n clocks on powerup or when -
-  reset pressed
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Revision 0.02 - Changed to run from master clock
-- Revision 0.03 - Added an extra reset output that only occurs during
-- powerup
-- Revision 0.03 - Changed powerup reset to be shorter than normal
-- reset
-- Additional Comments:
--
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity startup is
    Port (
                  RESET_IN : in bit;
         CLOCK : in bit;
                  POWER_UP_RESET_OUT : out  bit;
                  RESET_OUT : out  bit
                  );
end startup;

architecture Behavioral of startup is

signal r_counter       : std_logic_vector(24 downto 0);
signal pwr_counter       : std_logic_vector(25 downto 0);
signal int_resetn       : bit;

begin
      process(RESET_IN,CLOCK) is
            begin
            if (RESET_IN = '0' or int_resetn = '0') then -- if reset
switch pushed, reset counter
                  r_counter <= (others => '0');
                  RESET_OUT <= '0';
```

```
            elsif r_counter = '1' & x"ffbeef" then -- timer
                  RESET_OUT <= '1';
            elsif CLOCK = '1' and CLOCK' event then
                  RESET_OUT <= '0';
                  r_counter <= r_counter + 1;
            end if;
      end process;

      process(CLOCK) is
            begin
            if pwr_counter = "11" & x"ffcafe" then -- powerup timer
approx 1sec
                  POWER_UP_RESET_OUT <= '1';
                  int_resetn <= '1';
            elsif CLOCK = '1' and CLOCK' event then
                  int_resetn <= '0';
                  POWER_UP_RESET_OUT <= '0';
                  pwr_counter <= pwr_counter + 1;
            end if;
      end process;

end Behavioral;
```

# E

Clock.vhd, a VHDL component for clock and synchronous bus cycle generation.

```
----------------------------------------------------------------------------
-- Company:
-- Engineer: Lyndon Amsdon
--
-- Create Date:    12:22:03 10/10/2007
-- Design Name:
-- Module Name:    clocks - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description: Distributes Clocks of different frequencies from master
-- 32MHZ clock
-- E_CLK is simulated version of old 6800 8bit clock.  Frequency 1/10 -
-- of CPU clock,
-- Duty cycle 60% low, 40% high.
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Revision 0.02 - Strange issue in simulation using counter(1) as
-- clock for process
-- Changed to an internal signal
-- Revision 0.03 - Changed the 8, 2 and 0.5 MHz clock to be inverse in
-- relation to master clock
-- Revision 0.04 - Changed to use the DCM/PLL for 8 and 16 MHz clocks
-- Revision 0.04 - Added 27Mhz divide by 11 = 2.45 Mhz clock for MFP
-- Additional Comments:
--
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity clocks is
      Port (
                  RESET : in bit;
                  CLOCK_IN : in bit;
                  CLOCK32 : out  bit;
                  CLOCK16 : out  bit;
                  CLOCK8 : out  bit;
         CLOCK4 : out  bit;
         CLOCK2 : out  bit;
         CLOCK0_5 : out  bit;
                  CLOCK24576 : out  bit;
                  CLOCK_E : out bit;
                  VPAn  : in bit;
                  VMAn : out bit;
                  DTACKn : out bit
        );
end clocks;
```

```
architecture Behavioral of clocks is

signal pcounter               : std_logic_vector(3 downto 0);
signal e_counter              : std_logic_vector(3 downto 0);
signal dcmreset,dcmclock      : std_logic;
signal dcm8,dcm16,dcm32       : std_logic;
signal int_vman                    : bit;
signal dcm27,dcm24576         : std_logic;

component dcm_16
      PORT
      (
            CLKIN_IN                      : IN STD_LOGIC  ;
            RST_IN                        : IN STD_LOGIC  ;
            CLKDV_OUT               : OUT STD_LOGIC ;
            CLKFX_OUT               : out   std_logic;
            --CLKIN_IBUFG_OUT : OUT STD_LOGIC ;
            CLK0_OUT                      : OUT STD_LOGIC ;
            LOCKED_OUT              : OUT STD_LOGIC
      );
end component;
component dcm_8
      PORT
      (
            CLKIN_IN                      : IN STD_LOGIC  ;
            RST_IN                        : IN STD_LOGIC  ;
            CLKDV_OUT               : OUT STD_LOGIC ;
            --CLKIN_IBUFG_OUT : OUT STD_LOGIC ;
            CLK0_OUT                      : OUT STD_LOGIC ;
            LOCKED_OUT              : OUT STD_LOGIC
      );
end component;
component dcm_24576
      PORT
      (
            CLKIN_IN                      : IN STD_LOGIC  ;
            RST_IN                        : IN STD_LOGIC  ;
            CLKDV_OUT               : OUT STD_LOGIC ;
            --CLKIN_IBUFG_OUT : OUT STD_LOGIC ;
            CLK0_OUT                      : OUT STD_LOGIC ;
            LOCKED_OUT              : OUT STD_LOGIC
      );
end component;


begin

      process(dcm8,RESET) is
            begin
            if RESET = '0' then
                  pcounter <= "0000";
            elsif rising_edge(dcm8) then
                  pcounter <= pcounter + '1';
            end if;
      end process;
```

```
      process(RESET,dcm8) is
            begin
            if RESET = '0' then
                  e_counter <= "0000";
            elsif e_counter = "1010" then
                  e_counter <= "0000";
            elsif rising_edge(dcm8) then
                  e_counter <= e_counter + 1;
            end if;
      end process;

-- DCM PLL clock management
dcmclock <= to_stdulogic(CLOCK_IN);
dcmreset <= not to_stdulogic(RESET);
clock32 <= to_bit(dcm32);
clock16 <= to_bit(dcm16);
clock8  <= to_bit(dcm8);
clock24576  <= to_bit(dcm24576);

I_DCM_16 : dcm_16
      port map
            (
            CLKIN_IN => dcmclock,
      RST_IN => dcmreset,
      CLKDV_OUT => dcm16,
            CLKFX_OUT => dcm27,
      CLK0_OUT => dcm32,
      LOCKED_OUT => open
            );

I_DCM_8 : dcm_8
      port map
            (
            CLKIN_IN => dcm32,
      RST_IN => dcmreset,
      CLKDV_OUT => dcm8,
      CLK0_OUT => open,
      LOCKED_OUT => open
            );

I_DCM_24576 : dcm_24576
      port map
            (
            CLKIN_IN => dcm27,
      RST_IN => dcmreset,
      CLKDV_OUT => dcm24576,
      CLK0_OUT => open,
      LOCKED_OUT => open
            );

--CLOCK16 <= to_bit(pcounter(0));
--CLOCK8  <= to_bit(ncounter(1));
CLOCK4  <= to_bit(pcounter(0));
CLOCK2  <= to_bit(pcounter(1));
CLOCK0_5  <= to_bit(pcounter(3));

-- Produce 60/40 6800 syncronous E clock
```

```
CLOCK_E <= '1' when e_counter = "0110" or e_counter = "0111" or
                    e_counter = "1000" or e_counter = "1001" else '0';

--SR FlipFlop
process(dcm8,RESET)
begin
      if RESET = '0' then
            int_vman <= '1';
      elsif rising_edge(dcm8) then
            if (e_counter = "0010" and VPAn = '0') then
                  int_vman <= '0';
            elsif VPAn = '1' then
                  int_vman <= '1';
            end if;
      end if;
end process;

--Generate DTACK to end bus cycle
DTACKn <= '0' when (int_vman ='0' and (e_counter = "1000" or e_counter
="1001")) else '1';

VMAn <= int_vman;

end Behavioral;
```

# F

rom_control.vhd, a VHDL component for resizing 8 bit Flash memory bus to 16 bit.

```
----------------------------------------------------------------------------
-- Company:
-- Engineer: Lyndon Amsdon
--
-- Create Date:    00:34:12 10/12/2007
-- Design Name:
-- Module Name:    rom_control - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description: A small state machine to use 8 bit flash ROM on a 16bit
-- bus
-- Stores low byte then increments A0
-- 2x CPU clock needed
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - Changed Endian as file is transferred to target from
-- Intel Host
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity rom_control is
      Port (
                  FLASH_DATA : in  bit_VECTOR (7 downto 0);
         FLASH_ADDR : out bit_VECTOR (17 downto 0);
         D_BUS : out std_logic_VECTOR (15 downto 0);
         CLOCK : in  bit;
         A_BUS : in  bit_VECTOR (17 downto 1);
         CS : in  bit;
                  RESET : in  bit
                  );
end rom_control;

architecture Behavioral of rom_control is

-- define states
type state_type is (idle,botbyte,topbyte,incaddr);
signal state : state_type;

signal a0 : bit;
signal lowbyte : bit_vector(7 downto 0);
signal highbyte : bit_vector(7 downto 0);

begin
```

```
process (RESET,CLOCK)
  begin
      if RESET = '0' then state <= idle;
      elsif CLOCK = '1' and CLOCK' event then
      case state is

        when idle =>
          if CS = '0' then -- cycle started
            a0 <= '0';
            state <= botbyte;
          else
                        a0 <= '0';
            state <= idle;
          end if;

        when botbyte =>
                        highbyte <= FLASH_DATA;
                        a0 <= '0';
                        state <= incaddr;

             when incaddr =>
                        a0 <= '1'; -- increment address
                        state <= topbyte;

             when topbyte =>
                        a0 <= '1';
                        lowbyte <= FLASH_DATA;
                        if CS = '0' then -- cycle not finnished yet
                              state <= topbyte;
                        else
                              state <= idle;
                        end if;

        when others => null;
       end case;
    end if;
end process;

FLASH_ADDR <= A_BUS & a0; -- add the new A0 to the bus
D_BUS <= to_stdlogicvector(highbyte) & to_stdlogicvector(lowbyte) when
         CS ='0' else (others => '0');

end Behavioral;
```

# G

led_debug.vhd, a VHDL component for driving the value of the CPU address and data bus onto 7 segment displays. Hex2seg.vhd is a subcomponent.

```
----------------------------------------------------------------------------
-- Company:
-- Engineer: Lyndon Amsdon
--
-- Create Date:    16:41:38 10/12/2007
-- Design Name:
-- Module Name:    led_debug - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description: Displays current data and address but on a 7 Segment
-- Also provides a flashing led to show clocks are up and running
--
-- Dependencies:
--
-- Revision:
-- Revision 0.09 - Support to show data bus on bus error cycles
-- Revision 0.08 - Data byte enables used to display blank character in
-- byte transfers
-- Revision 0.07 - UDSn used to display missing A0
-- Revision 0.06 - Improved so data is latched one CPU cycle after
-- dtackn low
-- Two clocks needed, one for display (slow ~0.5Mhz) and another for
-- CPU clock speed
-- Revision 0.05 - Added latch for data bus as data from RAM is cyclic
-- Revision 0.04 - Slowed display down a little bit
-- Revision 0.03 - Modified to display full address and data buses
-- Revision 0.02 - Modified to display current address
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity led_debug is
    Port (
            ADDRESS : in  bit_VECTOR (23 downto 1);
            DATA : in  std_logic_VECTOR (15 downto 0);
            LEDCOL : out  bit_VECTOR (4 downto 0);
            LEDROW : out  bit_VECTOR (7 downto 0);
            CLOCK : in  bit;
            CLOCK8 : in  bit;
            LEDFLASH : out  bit;
            UDSn: in  bit;
            LDSn: in  bit;
            DTACKn: in  bit;
            BERRn: in  bit;
```

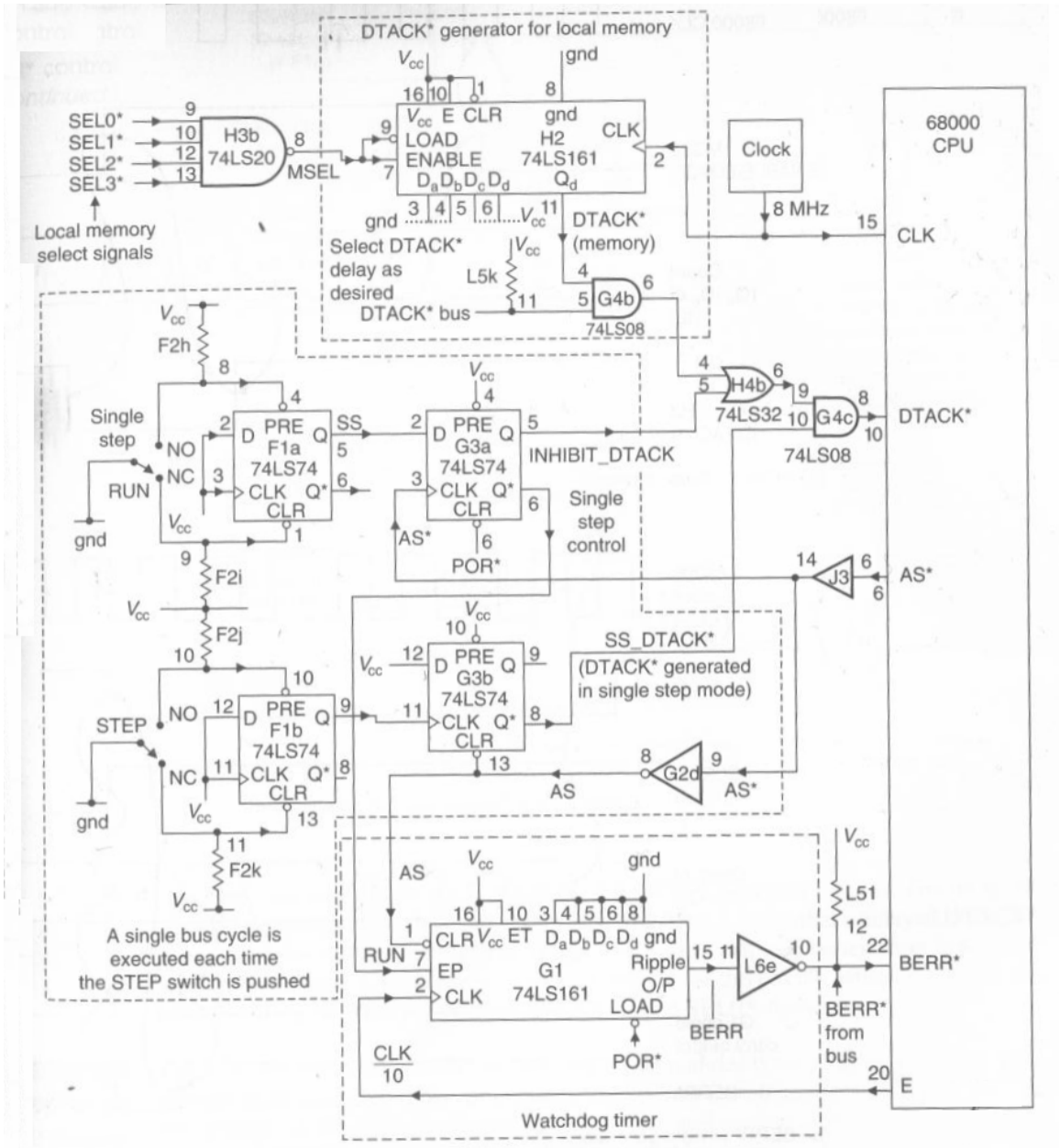
```
            RESET : in  bit
       );
end led_debug;

architecture Behavioral of led_debug is

signal counter       : std_logic_vector(20 downto 0);
signal temphex :  bit_vector (3 downto 0);
signal tempseg :  bit_vector (6 downto 0);
signal blank :  bit;
signal temp_latch :  bit_vector (15 downto 0);

-- define states
type state_type is (S4,S6,S8);
signal state : state_type;


-- LED Driver
begin
process(RESET,CLOCK) is
      begin
      if (RESET = '0') then -- reset signals
            counter <= (others => '0');
      elsif CLOCK = '1' and CLOCK' event then
            counter <= counter + 1; -- increment counter
            LEDFLASH <= to_bit(counter(18));
                  case  counter(1 downto 0)  is
                  when "00" => --DIGIT1
                        blank <= '0';
                        if counter(20 downto 19) = "00" then
                              blank <= '1'; -- display blank char
                        elsif counter(20 downto 19) = "01" then
                              temphex <= ADDRESS (15 downto 12);
                        else
                              blank <= UDSn;
                              temphex <= temp_latch (15 downto 12);
                        end if;
                        LEDROW <= '1' & tempseg;
                        LEDCOL <= "00001";
                  when "01" => --DIGIT2
                        blank <= '0';
                        if counter(20 downto 19) = "00" then
                              blank <= '1'; -- display blank char
                        elsif counter(20 downto 19) = "01" then
                              temphex <= ADDRESS (11 downto 8);
                        else
                              blank <= UDSn;
                              temphex <= temp_latch (11 downto 8);
                        end if;
                        LEDROW <= '1' & tempseg;
                        LEDCOL <= "00010";
                  when "10" => --DIGIT3
                        blank <= '0';
                        if counter(20 downto 19) = "00" then
                              temphex <= ADDRESS (23 downto 20);
                        elsif counter(20 downto 19) = "01" then
                              temphex <= ADDRESS (7 downto 4);
```

```
                        else
                              blank <= LDSn;
                              temphex <= temp_latch (7 downto 4);
                        end if;
                        LEDROW <= '1' & tempseg;
                        LEDCOL <= "00100";
                  when "11" => --DIGIT4
                        blank <= '0';
                        if counter(20 downto 19) = "00" then
                              temphex <= ADDRESS (19 downto 16);
                        elsif counter(20 downto 19) = "01" then
                              temphex <= ADDRESS (3 downto 1) & UDSn;
                        else
                              blank <= LDSn;
                              temphex <= temp_latch (3 downto 0);
                        end if;
                        LEDROW <= '1' & tempseg;
                        LEDCOL <= "01000";

                  when others =>
                        null;
                  end case;
            end if;
      end process;

process (CLOCK8)
  begin
      if CLOCK8 = '0' and CLOCK8' event then
      case state is
        when S4 =>
                  if DTACKn = '0' or BERRn = '0' then -- cycle started
            state <= S6;
          else
            state <= S4;
          end if;
             when S6 =>
                        temp_latch <= to_bitvector(DATA);
                        state <= S8;
             when S8 =>
                  state <= S4;
        when others => null;
      end case;
    end if;
end process;

I_HEX2SEG : entity work.hex2seg
      port map
            (
            BLANK => blank,
         HEX => temphex,
         SEG => tempseg
            );


end Behavioral;
```

```
----------------------------------------------------------------------------
-- Company:
-- Engineer: Lyndon Amsdon
--
-- Create Date:    21:53:10 10/15/2007
-- Design Name:
-- Module Name:    hex2seg - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description: Converts a Hex value to a 7 segment display
-- Dependencies:
-- Revision:
-- Revision 0.02 - Added option to display a '-' character
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity hex2seg is
    Port (
                  BLANK : in bit;
                  HEX : in bit_vector (3 downto 0);
         SEG : out bit_vector (6 downto 0)
                  );
end hex2seg;
architecture Behavioral of hex2seg is
signal conversion : bit_vector (6 downto 0);
begin

SEG <= "1111110" when BLANK = '1' else conversion;
with HEX select
            conversion <= "0000001" when "0000",
                          "1001111" when "0001",
                          "0010010" when "0010",
                          "0000110" when "0011",
                          "1001100" when "0100",
                          "0100100" when "0101",
                          "0100000" when "0110",
                          "0001111" when "0111",
                          "0000000" when "1000",
                          "0000100" when "1001",
                          "0001000" when "1010",
                          "1100000" when "1011",
                          "0110001" when "1100",
                          "1000010" when "1101",
                          "0110000" when "1110",
                          "0111000" when "1111",
                          "0111000" when others;


end Behavioral;
```

# H

Schematic from ‘Microprocessor System Design’ by Alan Clements showing single step control of the Motorola MC68000.



# I

Single_step.vhd, a VHDL component for controlling single stepping of the CPU through bus cycles.

```
----------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    15:09:37 01/15/2008
-- Design Name:
-- Module Name:    single_step - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
-- Used to intercept the bus cycle handshaking to the CPU so
-- that a switch can be used to single step the SPU
--
-- Dependencies:
--
-- Revision:
-- Revision 0.04 - Added reset for state machine
-- Revision 0.03 - Modified to work with pulsing dtack in (eg from mmu)
-- Revision 0.02 - Getting stuck in Busend State, changed to delay
-- rather
-- than checking for AS going high.
-- Revision 0.01 - File Created, use state machine
-- Additional Comments:
--
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity single_step is
    Port ( CLK16 : in  bit;
                    CLK8 : in  bit;
                    RESET : in  bit;
                    RAM : in  bit;
                    AS : in  bit;
           DTACK_I : in bit;
           BERR_I : in  bit;
           SWITCH : in  bit;
           DTACK_O : out  bit;
           BERR_O : out  bit);
end single_step;

architecture Behavioral of single_step is

signal counter       : std_logic_vector(23 downto 0);
```

```
-- define states
type state_type is (idle,button,busend,delay,ramaccess);
signal state : state_type;

begin

process (CLK16,RESET)
begin
if reset = '0' then
      state <= idle;
      BERR_O <= '1';
      DTACK_O <= '1';
elsif CLK16 = '1' and CLK16' event then

case state is
      when idle =>
      BERR_O <= '1';
      DTACK_O <= '1';
      counter <= "00000000000000000000000000"; --reset counter
            if AS = '0' then -- cycle started
                  state <= button;
            else
                  state <= idle;
            end if;

when button =>
      if switch ='0' and RAM ='0' and CLK8 ='0' and DTACK_I ='1' then
            DTACK_O <= DTACK_I;
            BERR_O <= BERR_I;
            state <= ramaccess;
      elsif switch='0' and RAM='1' and (DTACK_I='0' or BERR_I='0') then
            DTACK_O <= DTACK_I;
            BERR_O <= BERR_I;
            state <= busend;
      else
            state <= button;
      end if;

when busend =>
      --if AS = '1' then -- cycle ended
      DTACK_O <= DTACK_I;
      BERR_O <= BERR_I;
      state <= delay;
          --else
          --  state <= busend;
          --end if;

when delay =>
      if counter = "11111111111111111111111111" then -- debounce counter
            state <= idle;
      else
            BERR_O <= '1';
            DTACK_O <= '1';
            counter<=counter+1; --increment for 1 second delay counter
            state <= delay;
      end if;
```

```
when ramaccess =>
      DTACK_O <= DTACK_I;
      BERR_O <= BERR_I;
      if RAM = '1' then -- wait for cycle to end
            state <= delay;
      else
            state <= ramaccess;
      end if;

when others => null;
end case;
end if;
end process;

end Behavioral;
```

# J

Change to MFP IP core wf68901ip_timers.vhd for correct generation of x_CNTSTRB where x is the Timer letter.

**Old:**

```
wait until CLK = '1' and CLK' event;
if PRESCALE > x"00" and XTAL_STRB = '1' then
      PRESCALE := PRESCALE - '1';
else
      case TACR(2 downto 0) is
            when "111" => PRESCALE := x"C7"; -- Prescaler = 200.
            when "110" => PRESCALE := x"63"; -- Prescaler = 100.
            when "101" => PRESCALE := x"3F"; -- Prescaler = 64.
            when "100" => PRESCALE := x"31"; -- Prescaler = 50.
            when "011" => PRESCALE := x"0F"; -- Prescaler = 16.
            when "010" => PRESCALE := x"09"; -- Prescaler = 10.
            when "001" => PRESCALE := x"03"; -- Prescaler = 4.
            when "000" => PRESCALE := x"00"; -- Timer stopped or event
                                             -- count mode.
      end case;
end if;
case PRESCALE is
      when x"00" => A_CNTSTRB <= '1';
      when others => A_CNTSTRB <= '0';
end case;
```

**New:**

```
wait until CLK = '1' and CLK' event;
if PRESCALE > x"00" and XTAL_STRB = '1' then
      PRESCALE := PRESCALE - '1';
elsif XTAL_STRB = '1' then
      case TACR(2 downto 0) is
            when "111" => PRESCALE := x"C7"; -- Prescaler = 200.
            when "110" => PRESCALE := x"63"; -- Prescaler = 100.
            when "101" => PRESCALE := x"3F"; -- Prescaler = 64.
            when "100" => PRESCALE := x"31"; -- Prescaler = 50.
            when "011" => PRESCALE := x"0F"; -- Prescaler = 16.
            when "010" => PRESCALE := x"09"; -- Prescaler = 10.
            when "001" => PRESCALE := x"03"; -- Prescaler = 4.
            when "000" => PRESCALE := x"00"; -- Timer stopped or event
                                             -- count mode.
      end case;
end if;
case PRESCALE is
      when x"00" => A_CNTSTRB <= XTAL_STRB;
      when others => A_CNTSTRB <= '0';
end case;
```

# K

Change to Glue IP core wf25915ip_interrupts.vhd for correct generation of IPL level from interrupt sources, MFP, HBL and VBL.

**Old:**

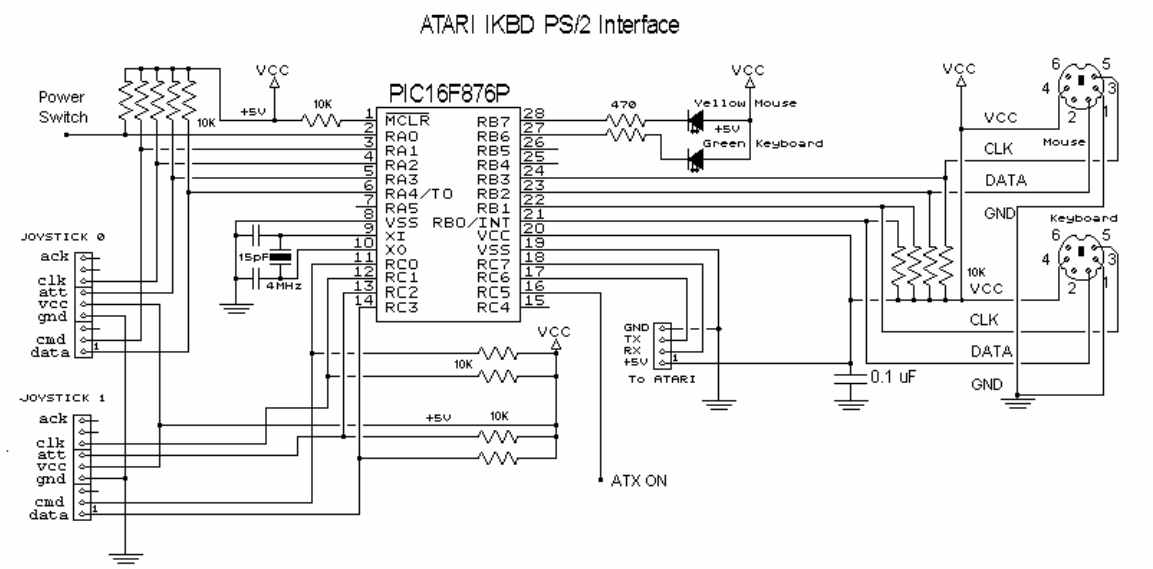
```
PRIODECODER: process(EINT3n, EINT5n, EINT7n, GI_In)
      begin
            if EINT7n = '0' then -- Highest priority.
                  IPLn <= "000";
            elsif GI_In(1) = '0' and GI_In(2) = '0' then -- MFPINT.
                  IPLn <= "001";
            elsif EINT5n = '0' then
                  IPLn <= "010";
            elsif GI_In(1) = '0' and GI_In(2) = '1' then -- H-Blank.
                  IPLn <= "011";
            elsif EINT3n = '0' then
                  IPLn <= "100";
            elsif GI_In(1) = '1' and GI_In(2) = '0' then -- V-Blank.
                  IPLn <= "101";
            else
                  IPLn <= "111";
            end if;
end process PRIODECODER;
```

**New:**

```
PRIODECODER: process(EINT3n, EINT5n, EINT7n, GI_In)
    begin
        if EINT7n = '0' then -- Highest priority.
            IPLn <= "000";
        elsif GI_In(2) = '0' and GI_In(1) = '0' then -- MFPINT.
            IPLn <= "001";
        elsif EINT5n = '0' then
            IPLn <= "010";
        elsif GI_In(2) = '0' and GI_In(1) = '1' then -- V-Blank.
            IPLn <= "011"; -- 011
        elsif EINT3n = '0' then
            IPLn <= "100";
        elsif GI_In(2) = '1' and GI_In(1) = '0' then -- H-Blank.
            IPLn <= "101"; -- 101
        else
            IPLn <= "111";
        end if;
end process PRIODECODER;
```

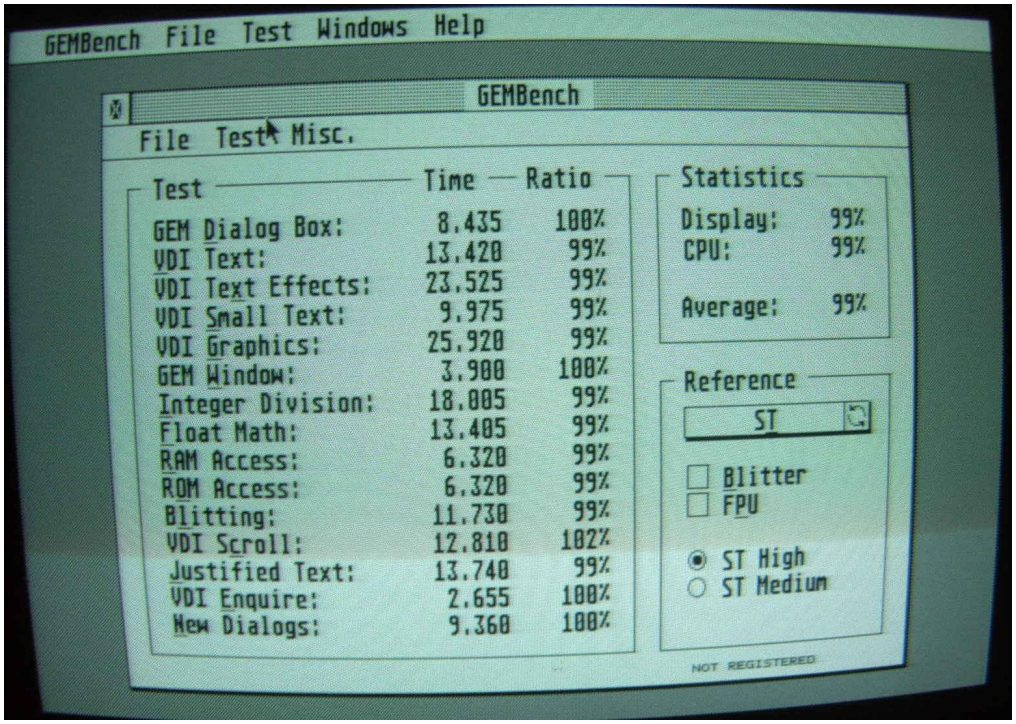
# L

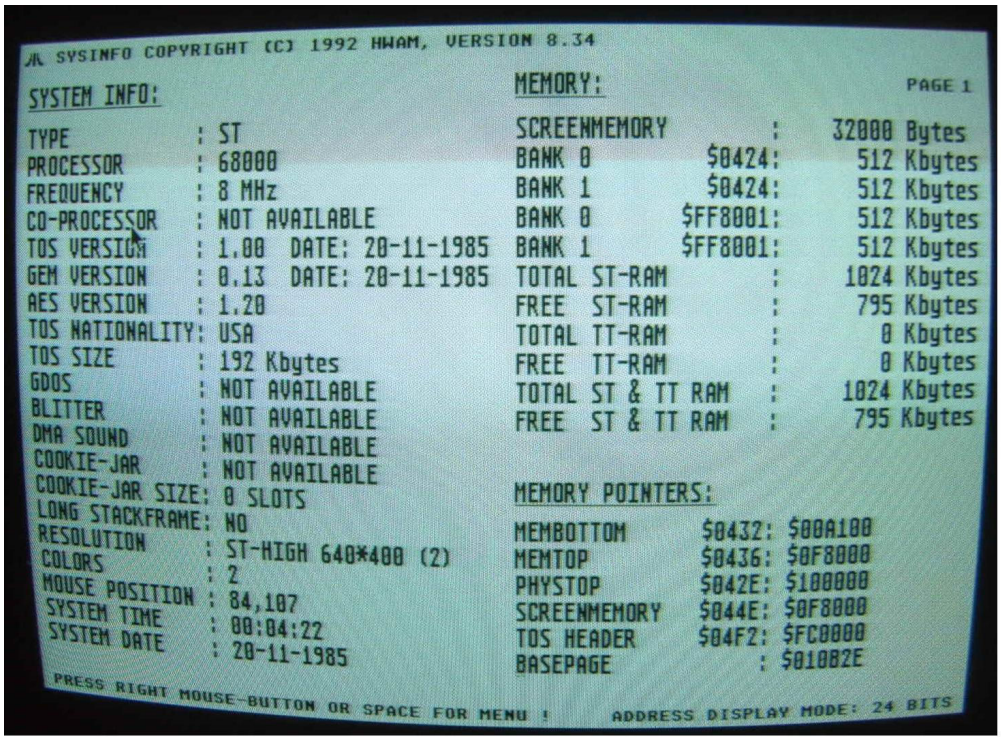
Schematic of the revised Eiffel project, to support control of an ATX power supply and use of Sony Playstation controllers.



# M

Gembench and SysInfo Results.



# N

Two sections from wf25915ip_video_timing.vhd, first is the original and the second is the revised version with equality relational operators.

```
DE_CTRL: process(CLK, RESETn)
begin
if RESETn = '0' then
       HDE <= '0'; -- Blanking out.
       VDE <= '0'; -- Blanking out.
elsif CLK = '1' and CLK' event then
       -- Horizontal controls:
       if SHIFTMODE = "10" then -- 35.714 kHz.
              if HTEMP > "000101000" and HTEMP <= "001101000" then
                     HDE <= '0'; -- 8us low, 3.0 before and 2.0 after HSYNC.
              else
                     HDE <= '1';
              end if;
       else -- 15.625 kHz.
              if HTEMP > "010110100" and HTEMP <= "101110100" then
                     HDE <= '0'; -- 24us low,14 before and 5 after HSYNC.
              else
                     HDE <= '1';
              end if;
       end if;

       -- Vertical controls:
       if SHIFTMODE = "10" then -- 72Hz.
              if VTEMP > "011010001" and VTEMP <= "100110010" then
                     VDE <= '0'; -- 97 lines low,47 before and 49 after VSYNC.
              else
                     VDE <= '1';
              end if;
       elsif (SHIFTMODE = "01" or SHIFTMODE = "00") and SYNCMODE(1) = '0' then
              -- 60.00Hz.
              if VTEMP > "000011000" and VTEMP <= "001010111" then
                     VDE <= '0'; -- 63 lines low,40 before and 20 after VSYNC.
              else
                     VDE <= '1';
              end if;
       elsif (SHIFTMODE = "01" or SHIFTMODE = "00") and SYNCMODE(1) = '1' then
              -- 50.00Hz.
              if VTEMP > "000000111" and VTEMP <= "001111010" then
                     VDE <= '0'; -- 115 lines low,72 before and 40 after VSYNC.
              else
                     VDE <= '1';
              end if;
       else
              VDE <= '1';
       end if;
end if;
end process DE_CTRL;
```

```
DE_CTRL: process(CLK, RESETn)
begin
if RESETn = '0' then
	HDE <= '0'; -- Blanking out.
	VDE <= '0'; -- Blanking out.
elsif CLK = '1' and CLK' event then
	-- Horizontal controls:
	if SHIFTMODE = "10" then -- 35.714 kHz.
		if HTEMP = "000000100" then --4
			HDE <= '1'; -- 8us low, 3.0 before and 2.0 after HSYNC.
		elsif HTEMP = "010100100" or HTEMP = "010101100" then --164 or 172
			HDE <= '0';
		end if;
	elsif (SHIFTMODE = "01" or SHIFTMODE = "00") and SYNCMODE(1) = '0' then
	-- 60.00Hz.
		if HTEMP = "000110100"   then --52
			HDE <= '1'; -- 24us low,14 before and 5 after HSYNC.
		elsif HTEMP = "101110100" or HTEMP = "111001100" then --372 or 460
			HDE <= '0';
		end if;
	elsif (SHIFTMODE = "01" or SHIFTMODE = "00") and SYNCMODE(1) = '1' then
	-- 50.00Hz.
		if HTEMP = "000111000"  then --56
			HDE <= '1'; -- 24us low,14 before and 5 after HSYNC.
		elsif HTEMP = "101111000" or HTEMP = "111010000" then --376 or 464
			HDE <= '0';
		end if;
	end if;

	-- Vertical controls:
	if SHIFTMODE = "10" and HTEMP = "011010000" then -- 72Hz.
		if VTEMP = "111000000" or VTEMP = "111011001" then -- 448 or 473
			VDE <= '0'; -- 97 lines low,47 before and 49 after VSYNC.
		elsif VTEMP = "000110000" then --48
			VDE <= '1';
		end if;
	elsif (SHIFTMODE = "01" or SHIFTMODE = "00") and HTEMP = "111101100" and
SYNCMODE(1) = '0' then -- 60.00Hz.
		if (VTEMP = "011101010" or VTEMP = "100000010") then --234 or 258
			VDE <= '0';
		elsif VTEMP = "000100010" then --34
			VDE <= '1';
		end if;
	elsif (SHIFTMODE = "01" or SHIFTMODE = "00") and HTEMP = "111110000" and
SYNCMODE(1) = '1' then -- 50.00Hz.
		if (VTEMP = "100000111" or VTEMP = "100110100")  then --263 or 308
			VDE <= '0';
		elsif VTEMP = "000111111" then --63
			VDE <= '1';
		end if;
	end if;
end if;
end process DE_CTRL;
```