# Embedded Linux Kernel Using a Synthesizable Open Source Soft Processor Core

Team - Arthur Miller, Lini Mestar, Jonathan Yap
Advisor - Halima El-Naga
California Polytechnic University - Pomona

***Abstract*—The purpose of this project was to run the Linux Operating System on a FPGA. This was accomplished by using a synthesizable open-source soft-processor core on a FPGA. The Intel Pentium based v586 project was chosen for the base processor system. Buildroot was then used to help create an embedded Linux kernel to run on the v586 core - as outlined in the v586 documentation. Reaching out to the developer of the v586 core helped detail a developer's guide to easily modify and implement his initial design. With his guidance the implemented design ran on a Nexys 4 FPGA and successfully booted the Linux kernel. The Linux terminal was displayed on a laptop connected via UART communications. Building on top of his work, microPython was added to the base kernel and successfully ran a script.**

***Index Terms*—i586, Synthesizable Processor, Embedded Linux, FPGA**

## I. Introduction

Our team approached the Senior Design Project from a R&D perspective. The goal of this project was to utilize as many aspects of our Computer Engineering curriculum which we have learned so far. Such as FPGA design using a HDL, Operating Systems, Computer Architecture and micro-controller Design. We settled on the idea of building a functional micro-computer. To develop and implement a micro-computer a full system level design is needed. It is common practice in Industry for a computer engineer to find a project that fits the needs of a problem and then customizes it according to user specifications. It is also the goal of a Computer Engineer to integrate both hardware and software [12]. Keeping these objectives in mind our team set out to accomplish these goals.

## II. Preliminary Research

There were several key software and hardware components that needed to be decided during our preliminary research phase. The soft processing unit that would be the base for our computer system was the first. With many open source and proprietary solutions for a synthesizable processor we had to chose the best fit for the team's needs. The FPGA to which we would synthesize the software on needed to be determined as well. Another consideration was the bus system in which we would use to connect the different HDL modules together. Finally we also had to consider which operating system that we would run. All of these preliminary research steps will be outlined in the following sub sections.

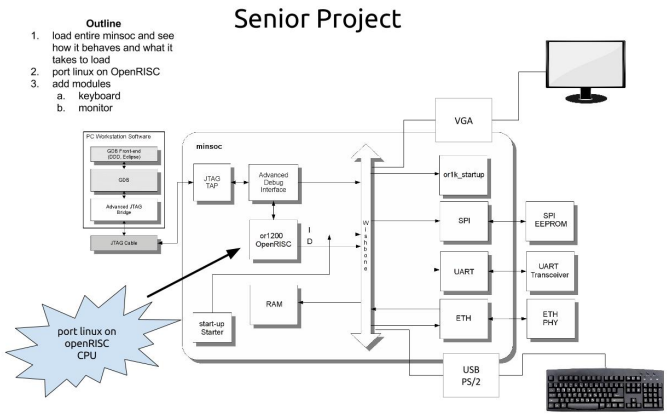
### A. Synthesizable Processor Core Options

*1) Custom Built Processor from Scratch:* The team's original idea was to continue development on the MIPS processor designed in Verilog during the ECE 425 course. The 16 bit processor was going to be extended to 32 bit. The instruction set was also going to be extended to a fully MIPS compatible instruction set. In addition, several advanced processing units were being considered to be developed and added to the design.

*2) MinSOC:* Minimal OpenRISC System On Chip (MinSOC) was the first open source soft processing module that we worked with. It is an open source project at the HDL IP core site OpenCores [5]. It is one of the few projects that is *Open Cores Certified* - which is the highest rating the site gives to indicate the stability of a project. It uses the OpenRISC 1200 processor core, which has one of the best development communities for open source processors. The OpenRISC processor core is a fully open sourced MIPS based processor written in Verilog. It also uses the Wishbone bus interface (to be described in detail in a later section) which allows for compatibility with all of the other project modules on OpenCores. We downloaded the source and attempted to get the hello world test working for this project - but was not able to. This was do to a complex makefile system that they used which was supposed to allow the design to be more easily ported to other FPGA hardware. This project was also only officially supported on Ubuntu 12.04 operating system and had many dependencies that were hard to locate. In addition, it is synthesizable only on the Spartan 3E FPGA board - which is an older design board. This was not a simple project that could be synthesized using Vivado or another single HDL IDE. It used tools from both Vivado and ISE within its makefile system - this sacrificed simplicity for the sake of compatibility. The documentation online was vague and in the end our team was not able to get this project working.

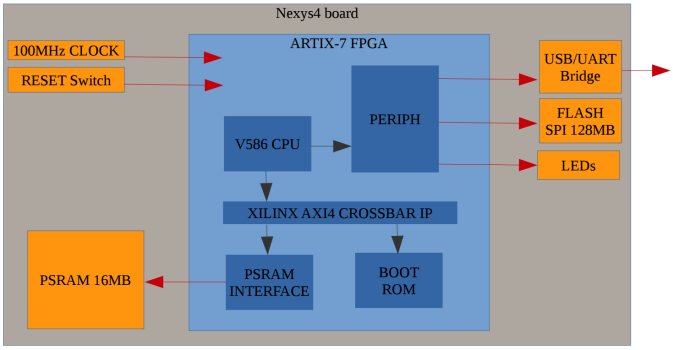
*3) v586:* The v586 project was the project that our team settled on using. It is an open source project that is on the OpenCores site as well [8]. This project was not as popular on the site as the MinSOC project, but its simplicity made it realizable for our needs. This CPU is based off the original Intel Pentium of the early 90s (generation i586). This is desirable because this architecture is popular and compatible with even modern software. Modern uses for this architecture includes the Intel Galileo which is a development board that runs Linux OS with Arduino software libraries [15]. The

Figure 1. MinSOC Block Diagram




Pentium ISA is compatible with BuildRoot which allows for easy kernel development [1]. This project is also supported for the newer Nexys 4 development board which is compatible with Xilinx's latest IDE Vivado [2]. A small critique is that using the v586 architecture does add the complexity of a CISC instruction set (see Appendix B for a detailed outline of the i586 processor). There were also several other limitations to what we could add or modify to this project due to the method of development by the developer. One being his use of a proprietary Cadence software (Virtuoso) to create the code for the peripheral interface. Secondly he used Xilinxs proprietary bus (AXI) to be described in detail in a later section. But perhaps the biggest benefit of choosing this project was the communication with the developer. There was very little documentation on the OpenCores site about this project so we reached out to the developer. He was a French engineer and was happy that students were taking interest in his work. He gave us advice and a thorough outline of his project to get us started.

Figure 2. v586 Block Diagram



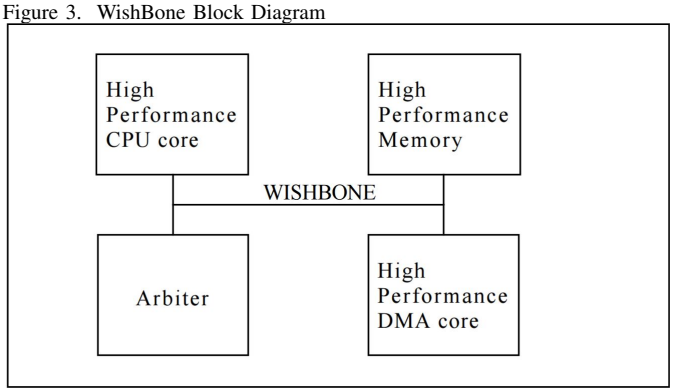

## B. Bus Selection

In order to connect soft cores to other peripherals, it is most common to use a unified bus system. Bus systems allows IP cores to connect to memory, a mouse, a keyboard, or even other IP cores. Having a unified system allows for quick changes to be made without having to remap all of the connections. An additional benefit is to future developers. For our project two bus systems caught our attention: Wishbone and AXI4; each providing different benefits.

*1) Wishbone:* Wishbone architecture is a non-copyrighted bus system released back in August 2002 on the OpenCores website [9]. It has become common for many designs for soft cores and other peripherals to come with Wishbone interfaces; especially in the open source community. The structure is relatively simple, and it is not uncommon for a complex SOC design to have multiple Wishbone interfaces. Under the wishbone architecture any module of a system can either be a master or a slave. Master interfaces are IP's which are capable of initiating bus cycles, while slave interfaces are capable of receiving bus cycles. Wishbone offers three types of interconnections including: shared bus, data flow, and crossbar switch [10].

Wishbone was the interface of choice for the MinSOC project, mentioned before. In Figure 1 the arrows coming to and from the Wishbone bus signify either a master or slave interface. For example the OpenRISC (OR1200) CPU core of MinSOC has both master and slave interfaces each 32 bits wide. It is through these interfaces it communicates to other peripherals via the Wishbone bus system. Of the three types of interconnections two are used in the MinSOC project: the crossbar switch and the shared bus. A shared bus interconnect only allows one master to communicate with one slave at the same time, while a crossbar switch may allow N masters connect to N slaves at the same time, according to the number of implemented buses [9]. MinSOC uses a double bus system at its current implementation.

Figure 3. WishBone Block Diagram




*2) Xilinx AXI4:* AXI4 is part of the ARM AMBA (Advanced Micro-controller Bus System). Family of micro-controller bus systems first created in 1996. AXI4 was included in the release of AMBA 4.0 in 2010 [20]. Xilinx has officially adopted this Advanced eXtensible Interface (AXI) protocol for IP cores beginning with the Spartan 6 and Virtex 6 devices. Comprising AXI4 are three types of general interfaces: AXI4, AXI4-Lite, and AXI4-Stream; each providing different capabilities depending on the needs of the developer. The first AXI4 is for high performance memory

mapped requirements; for up to 256 data transfer cycles for a single memory address. The second AXI-Lite is for lower throughput memory mapped requirements; made for single transaction memory transfers. The third Stream is for high speed streaming data, it removes the address phase altogether to allow unlimited data burst size. Similar to some features in wishbone, data can be transferred between master and slave simultaneously, and varied transfer sizes are available. However AXI allows for different clock cycles for each master slave pair.

In order to connect an AXI compliant master(s) to one or more memory mapped slave(s) the Xilinx AXI Interconnect IP is required. This interconnect is non-licensed and is available through the Xilinx software. Up to 16 masters and 16 slaves. AXI Interconnect provides all the connection types that wishbone offers; though named differently. From simple pass through communication or conversion; having only one master and one slave element. To complex configurations with more advanced capabilities: N to 1, 1 to N, N to M. There is even the option to send pipeline data, in the case of memory being shared by more than one master or slave.

For the v586 project a hybrid approach was chosen. The v586 is connected to both the PSRAM and the Boot ROM via the Xilinx AXI4 Crossbar IP. It is configured to be a 1 to N with bidirectional communication using the AXI4 interface.

## III. Implementation

The team documented a full implementation with pictures and a step by step instructions which can be found on GitHub [4]. Below is a quick outline of the project's implementation.

### A. Step by Step Tutorial of v586 Implementation

1) Download and install latest ISE IDE & Vivado IDE
2) Install cable drivers on Linux
3) Flash FPGA ROM using ISE iMPACT tool
4) Upload the synthesized bitstream to the FPGA using Vivado
5) Nexys 4 SW1/SW2 always on SW0 on/off button
6) Listen on Linux UART port using the program Screen and the following command:

```
sudo screen /dev/ttyUSB1 115200
```

7) Turn SW0 on
8) Now your computer console will show Linux booting on the FPGA
9) After boot sequence you can now login to the root account of the Linux machine

### B. Outline of Kernel Configuration For microPython Implementation

1) Modify kernel packages using Buildroot for microPython
2) Complete steps 1-9 as outlined above
3) You can now use python

## IV. Conclusion

The team was able to produce a working micro-computer on an FPGA running Linux. We used the open source project v586 as our synthesizable processor. We were able to use Buildroot to modify the system kernel to allow for running microPython on our device. The v586 project did end up being very constricting to our team when we tried to modify and add to the project. This was partially due to the nature of Verilog design but also to the original designers use of proprietary software. This project was humbling because our ambition drove us into a project that was much more difficult than expected. We had full control over the project and were naive as to what could be accomplished in the time frame. After much deliberation we were able to determine realistic goals. This project also gave the team a much better idea of the system level overview of a computer. We took the knowledge learned in various classes and got to see them work together. The entire design of a microcomputer must be understood, as well as a good understanding of version control, make files, Verilog, and operating systems.

## V. Future Goals

The final portion of the project that the team was not able to complete was adding peripheral devices. The teams priorities were to add a keyboard input, a VGA output, and a SD Card for added storage. The two methods to accomplish these goals are outlined below.

### A. Proprietary IP Cores Using Xilinx Bus

there are several IP cores that are already AXI4 connected that accomplish connecting to peripherals. Such as LCD displays, keyboard interfaces, as well as other sensors. The downside is cost. However, this does imitate how in the industry design elements can be outsourced depending on time and budget. Digital Blocks Semiconductor IP offers two IP cores that fit the project's needs.

### B. Open Source IP Cores Using Wishbone

The teams preferred method would have been to convert the v586 bus system to Wishbone. This would allow for the Open Source IP Cores on OpenCores to be used [7]. At the time of this project there were already projects for $I^2$C Keyboard Controllers, SD Card Controllers , and Ethernet Controllers [6]. There was not a controller for VGA communications, so some In-House design would still need to be done if the team did not want to pay for an IP Core to accomplish the VGA output.

### C. In-House Design

Another route that is to design modules for each device itself. This requires an intimate knowledge of communication between the device and the processor. Experience of Linux architecture is also needed in order to properly interface the corresponding device. This includes communication from kernel to I/O, and a different boot code. Making this sort of approach a time consuming and knowledge intensive one.

## References


[1] BuildRoot. *BuildRoot is for Everyone*. 2016. URL: https://buildroot.org/.

[2] Digilent. *Nexys 4 DDR Reference Manual*. 2016. URL: https://reference.digilentinc.com/reference/programmable-logic/nexys-4-ddr/reference-manual.

[3] Denis Howe. *hardware description language*. 2010. URL: http://www.dictionary.com/browse/hardware-description-language.

[4] Lini Mestar. *sp-i586*. 2016. URL: https://github.com/lmEshoo/sp-i586.

[5] OpenCores. *minsoc*. 2016. URL: http://opencores.org/project,minsoc.

[6] OpenCores. *Projects*. 2016. URL: http://opencores.org/projects.

[7] OpenCores. *SoC Interconnection: Wishbone*. 2016. URL: https://opencores.org/opencores,wishbone.

[8] OpenCores. *v586*. 2016. URL: http://opencores.org/project,v586,overview.

[9] OpenCores. *Wishbone B4*. 2010. URL: http://cdn.opencores.org/downloads/wbspec_b4.pdf.

[10] Rudolf Usselmann. "OpenCores SoC Bus Review". In: (2001).

[11] Wikipedia. *Complex instruction set computing*. 2016. URL: https://en.wikipedia.org/wiki/Complex_instruction_set_computing.

[12] Wikipedia. *Computer engineering*. 2016. URL: https://en.wikipedia.org/wiki/Computer_engineering.

[13] Wikipedia. *Field-programmable gate array*. 2016. URL: https://en.wikipedia.org/wiki/Field-programmable_gate_array.

[14] Wikipedia. *Integrated development environment*. 2016. URL: https://en.wikipedia.org/wiki/Integrated_development_environment.

[15] Wikipedia. *Intel Galileo*. 2016. URL: https://en.wikipedia.org/wiki/Intel_Galileo.

[16] Wikipedia. *MIPS instruction set*. 2016. URL: https://en.wikipedia.org/wiki/MIPS_instruction_set.

[17] Wikipedia. *Operating system*. 2016. URL: https://en.wikipedia.org/wiki/Operating_system.

[18] Wikipedia. *Serial Peripheral Interface Bus*. 2016. URL: https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus.

[19] Wikipedia. *Universal asynchronous receiver/transmitter*. 2016. URL: https://en.wikipedia.org/wiki/Universal_asynchronous_receiver/transmitter.

[20] Xilinx. *Vivado AXI Reference Guide*. 2014. URL: https://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/latest/ug1037-vivado-axi-reference-guide.pdf.


## Appendix A
## Jargon

**HDL**: Hardware Description Language (HDL) A kind of language used for the conceptual design of integrated circuits. Examples are VHDL and Verilog. [3]

**FPGA**: A field-programmable gate array (FPGA) is an integrated circuit designed to be configured by a customer or a designer after manufacturing hence "field-programmable". [13]

**OS**: An operating system (OS) is system software that manages computer hardware and software resources and provides common services for computer programs. [17]

**MIPS**: MIPS (originally an acronym for Microprocessor without Interlocked Pipeline Stages) is a reduced instruction set computer (RISC) instruction set architecture (ISA) developed by MIPS Technologies (formerly MIPS Computer Systems, Inc.). [16]

**CISC**: Complex instruction set computing (CISC) is a processor design where single instructions can execute several low-level operations (such as a load from memory, an arithmetic operation, and a memory store) or are capable of multi-step operations or addressing modes within single instructions. [11]

**UART**: A UART (Universal Asynchronous Receiver/Transmitter) is the microchip with programming that controls a computer's interface to its attached serial devices. [19]

**SPI**: is an interface that enables the serial (one bit at a time) exchange of data between two devices, one called a master and the other called a slave . An SPI operates in full duplex mode. This means that data can be transferred in both directions at the same time. [18]

**IDE**: An integrated development environment (IDE) is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of a source code editor, build automation tools and a debugger. Most modern IDEs have intelligent code completion. [14]

## Appendix B
## Intel Pentium

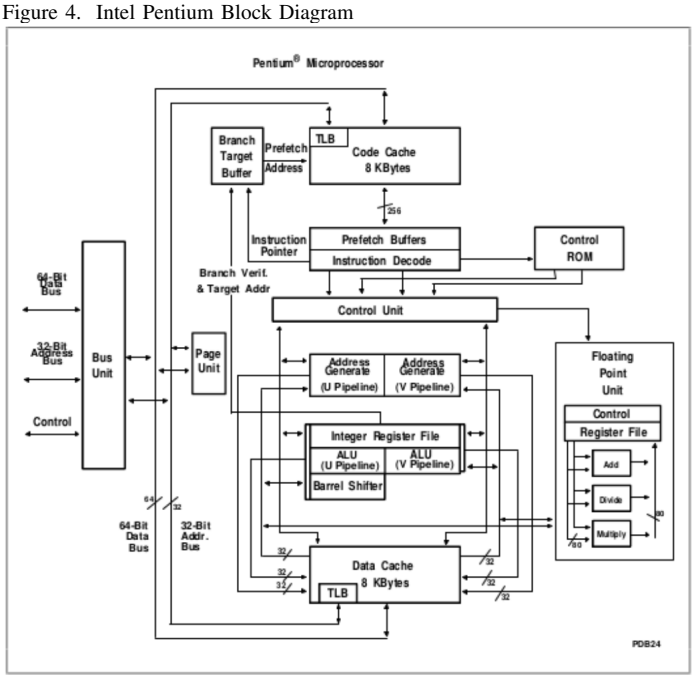
### A. Microprocessor Architecture Overview

The Pentium processor has an optimized superscalar micro-architecture capable of executing two instructions in a single clock. A 64-bit external bus, separate 8-Kbyte data and instruction caches, write buffers, branch prediction, and a pipe-lined floating-point unit combine to sustain the high execution rate. These architectural features and their operation are discussed in this chapter.

The Pentium processor is the next generation member of the Intel386 and Intel486 microprocessor family. It is 100% binary compatible with the 8086/88, 80286, Intel386 DX CPU, Intel386 SX CPU, Intel486 DX CPU, Intel486 SX and the Intel486 DX2 CPUs. The Pentium processor (510"60, 567"66) contains all of the features of the Intel486 CPU, and provides significant enhancements and additions including the following:

1) Superscalar Architecture
2) Dynamic Branch Prediction
3) Separate 8K Code and Data Caches with writeback MESI Protocol in the Data Cache
4) 64-Bit Data Bus

Figure 4. Intel Pentium Block Diagram




As well as these enhancements:

- Improved Instruction Execution Time
- Pipe-lined Floating-Point Unit
- Bus Cycle Pipe-lining
- Address Parity
- Internal Parity Checking
- Function Redundancy Checking
- Execution Tracing
- Performance Monitoring
- IEEE 1149.1 Boundary Scan
- System Management Mode
- Virtual Mode Extensions
- Upgradable with a future Pentium OverDrive processor

### B. Description of Enhancements

1) The Pentium processor family implements several enhancements to increase performance. The two instruction pipe-lines and floating-point unit on the Pentium processor are capable of independent operation. Each pipe-line issues frequently used instructions in a single clock. Together, the dual pipes can issue two integer instructions in one clock, or one floating-point instruction (under certain circumstances, 2 floating-point instructions) in one clock.
2) Branch prediction is implemented in the Pentium processor. To support this, the Pentium processor implements two prefetch buffers, one to prefetch code in a linear fashion, and one that prefetches code according to the BTB so the needed code is almost always prefetched before it is needed for execution.
3) The Pentium processor includes separate code and data caches integrated on chip to meet its performance goals. Each cache is 8 Kbytes in size, with a 32-byte line size and is 2-way set associative. Each cache has a dedicated Translation Lookaside Buffer (TLB) to translate linear addresses to physical addresses. The data cache is configurable to be writeback or writethrough on a line by line basis and follows the MESI protocol. The data cache tags are triple ported to support two data transfers and an inquire cycle in the same clock. The code cache is an inherently write protected cache. The code cache tags are also triple ported to support snooping and split line accesses. Individual pages can be configured as cacheable or non-cacheable by software or hardware. The caches can be enabled or disabled by software or hardware.
4) The Pentium processor has increased the data bus to 64-bits to improve the data transfer rate. Burst read and burst writeback cycles are supported by the Pentium processor. In addition, bus cycle pipe-lining has been added to allow two bus cycles to be in progress simultaneously. The Pentium processor Memory Management Unit contains optional extensions to the architecture which allow 4 Mbyte page sizes.

### C. Shortcomings

1) x86 overhead on the Pentium: According to an MPR article published at the time (see bibliography), Intel estimated that a whopping 30% of the Pentium's transistors were dedicated solely to providing x86 legacy support. When you consider the fact that the Pentium's RISC competitors with comparable transistor counts could spend those transistors on performance-enhancing hardware like execution units and cache, it's no wonder that the Pentium lagged behind.
2) Complicated CISC instruction set.