Team IEEE Nexus

Project Proposal, Stage Two

Revision 0
Edited 5/31/2016

Prepared by:

Daniel Luncasu-Rolea (dclr@uw.edu)
Treyven Chin (crimchin@uw.edu)
Joshua Yang (ybd1992@hotmail.com)
Renqing Li (renli@uw.edu)

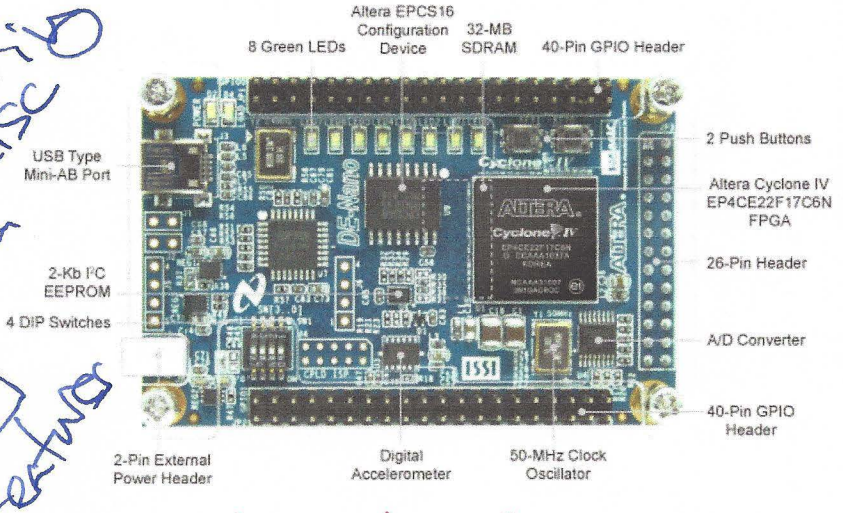
I've given you lots of red ink but I love it. You're going to have an actual product you could easily pitch turn Adafruit etc, and Guerilla market with articles for mags like Circuit Cellar.

Nicole

Table of Contents

Introduction

This project is an effort to construct a device based on the mips32r1 architecture with a debugging interface that conforms to Class I IEEE Nexus standards. The chosen platform is the Terasoc DE-Nano, featuring an Altera Cyclone IV FPGA with a 32mb SDRAM chip, dual GPIO header, 50 MHz clock, 8 green LED array, 2 debounced buttons, and 4 DIP switch inputs.



Objective

Our objective for this project is to continue the development of an educational platform for microprocessor design and/or programming, equipped with a debugging interface that conforms to real industry standards. Future Capstone groups in the University of Washington Electrical Engineering program will be able to build on this project through adding additional layers of debugging capability and/or further modifications to the processor.

The starting points for this project are:

1) The open-source mips32r1 architecture made available by Grant Ayers online on Github[1]
2) Instructions on building the GCC for cross-compiling mips32r1 code, also by Grant Ayers
3) USB-UART hardware to communicate with a XUM bootloader on PC
4) A dual-ported SDRAM controller provided by the previous Capstone team, not yet integrated

In this project, we will:

1) Create a stable build of the mips32r1 core and crosstools for the DE0-Nano
2) Integrate last team's SDRAM controller into the mips32r1 code and produce a stable build
3) Design a top shield with LEDs, switches, keys, and a MicroSD jack for non-volatile storage
4) Create a bottom shield for interfacing Arduino shields and products with our design
5) Implement Arudino libraries and allow use of the Arduino IDE

[1] https://github.com/grantae/mips32r1_soc_nano

SELL THIS → this is an Arduino with a much faster 32-bit softcore processor, far more RAM, and an FPGA for I/O devices, and a control panel.

Hardware Comparison

The mips32r1 processor is a softcore pipelined Reduced-Instruction-Set-Computing based system. The previous Capstone team chose it due to its flexibility, simple architecture, easily-available cross-compilation tools, and 32-bit architecture[2]. Our product uses a Terasic DE0-Nano prototyping board, which was chosen by the previous Capstone team due to its low price, 50Mhz clock, small size, numerous expansion headers, and 32mb onboard SDRAM[3].

The Arduino is a similar product (a microcontroller) used by hobbyists around the world. It uses an 8-bit, 16Mhz ATMega processor and features 54 digital pins and 16 analog inputs (Ardino Mega 2560) or 14 digital pinsand 6 analog inputs. [4]

Table of features? 32 bit vs 8 bit RAM ...

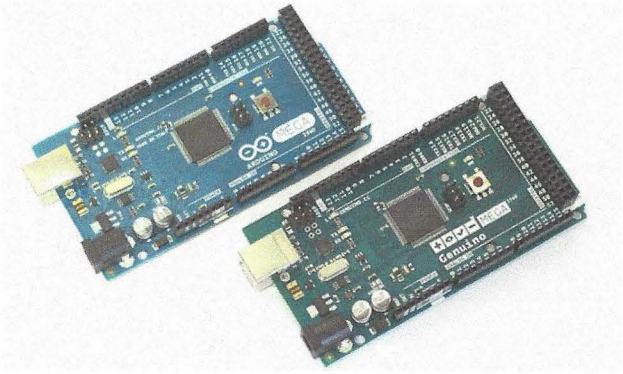


Figure 1: Arduino Mega 2560

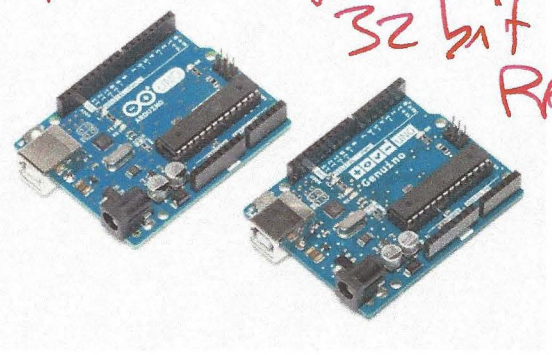


Figure 2: Arduino Uno

much easier to understand as a table

By default, mips32r1 is a bare-metal processor with no inbuilt memory management. Grant Ayers' implementation does, however, include memory and I/O mapping to memory locations, and features 64kb of block RAM built into the processor using logic gates[5]. The DE0-nano contains a 32mb SDRAM chip and we plan to include a MicroSD card jack for storing programs and long-term data.

The Mega 2560 features 256kb of Flash memory, 8kb of SRAM used for variables and 4kb of EEPROM for long-term data storage.[6]

future

Our product will utilize an FPGA which can be reprogrammed, allowing modifications to the processor to facilitate debugging or extend the features of the processor. The processor source contains a 64kb block RAM implementation which takes a lot of space, but the SDRAM controller does not take much space in comparison and allows us to remove the block RAM entirely.

---

[2] https://github.com/grantea/mips32r1
[3] https://www.altera.com/b/de0-nano-dev-board.html
[4] https://www.arduino.cc/en/Main/ArduinoBoardMega2560
[5] https://github.com/grantae/mips32r1_soc_nano
[6] https://www.arduino.cc/en/Tutorial/Memory

Feel free to borrow the comparison charts from my IEEE presentation, just cite it

Part 1: Mips32r1 on DE0-Nano

The first step toward building a basic mips32r1 implementation on a DE0-Nano was to set up cross-compilation tools and upload a basic "Hello World" program onto the board. The processor itself was compiled using Altera's Quartus Prime software, while the crosstools were compiled using a Cygwin terminal.

This work is largely done.

GCC Compiler

To upload C code to the processor, we needed to build a C compiler variant configured for cross-compilation. Grant Ayers' implementation[7] contained a set of commands and prerequisites, allowing us to build a working implementation after documenting all the required (and unspecified) steps. See Table 1 for a general list of the tools and prerequisite archives to build the tools.

UART Protocol

The bootloader attached to Grant Ayers' implementation used a UART protocol to connect to a computer via USB. To talk to the processor initially, we used CP2102 bridges[8] to communicate with the bootloader.

Test Program

The test program we used to verify the board and compiler were working included a linker script and several assembly files dictating interrupt functionality. We hope to include these files by default in future implementations, for a clean and user-friendly interface.

Table 1: Required Archives and Cygwin Packages

Required Archives	Required Cygwin Libraries
binutils-2.24.tar.bz2	*Gcc*
gcc-4.9.1.tar.bz2	*G++*
mpfr-3.1.2.tar.bz2	*libgmp-devel*
mpc-1.0.2.tar.gz	*libmpfr-devel*
gmp-6.0.0a.tar.bz2	*libmpc-devel*
newlib-2.1.0.tar.gz	*binutils*
	diffutils
	texinfo
	xxd
	make

[7] https://github.com/grantae/mips32r1_soc_nano

[8] https://www.silabs.com/products/mcu/Pages/USBtoUARTBridgeVCPDrivers.aspx

Part 2: SDRAM Controller

The previous Capstone team built an SDRAM controller to interface with the DE0-Nano's onboard 32mb memory chip. This gives us much more memory to work with at the cost of additional complexity. We will need to configure the SDRAM to replace block RAM in its entirely on Grant Ayers' implementation.

The SDRAM controller is useful for a number of reasons:

1) The existing block RAM only contains 64kb of memory, enough for relatively small programs. The SDRAM chip on the board is 32mb, allowing for much larger programs
2) The block RAM requires a large number of logic elements on the chip. The SDRAM controller, on the other hand, is much smaller, and the SDRAM chip itself does not require additional logic elements on the chip.

There are also a number of disadvantages:

1) The SDRAM chip is much slower than BRAM and will introduce several clock cycles of delay between read commands on the instruction memory
2) The implementation may introduce new bugs into the code and slow future work

The SDRAM wrapper provided by the previous capstone team did not fit the protocol used by the processor. The processor operates by issuing a read command first, then waiting for an "Ack" signal from the memory controller. The current SDRAM controller uses a "data ready" signal instead which remains high when idle and therefore halts the processor.

The memory controller will be paged into identically-sized 16mb segments, each of which will be used as instruction memory. It may be prudent later to include two modes for the processor to run in: SDRAM mode with instruction memory written to the memory chip, and BRAM mode wherein instructions are read from internal block RAM similar to the kind included with the processor.

What would you do to solve this? Could you use a cache? If you don't, maybe the next team might build one.

(Caching lets you use burst mode to transfer large blocks)

What is this for (except perhaps as a build option)?

Btw: On subject of SMD below, would you like to try to do SMD if we could get the IEEE to buy us an actual hot-air SMD rework station?

Part 3: Top Board

The top board will have the following features:

1) LED's as I/O for programs and for debugging purposes
2) Switches and keys for control and user input
3) Seven-segment displays for displaying numbers and text
4) A MicroSD jack for non-volatile memory storage
5) A single free GPIO header for 3.3V TTL logic

Board dimensions? Likely # of Layers? One? Two-sided parts? Any SMD?

Artwork?

Power

The Cyclone IV chip on the DE0-nano supports 40 mA per I/O pin, which is more than enough for individual LED's. The power requirements for the seven-segment displays may depend on the individual model. Our design will use bubble displays which magnify a small image for increased visibility and decreased power consumption.

The DE0-nano can supply 1.2 amps on the 5V jack and 1.5 amps on the 3.3V jack[9], which will be enough for our needs.

MicroSD Card

What does this mean? Are we talking about boot or file system or what?

The top board will contain a MicroSD card jack which will interface with an I/O controller in our design, separate from the processor. Upon program upload, the MicroSD controller will upload the current program onto the MicroSD card if one is present and use it as a form of extended RAM.

The system, in its first implementation, will disregard any filesystem present on the SD Card. In order to write useful files, users will need to provide an additional SD card and interface it with a C program given to the processor.

No idea what this means either.

Connection

One of the GPIO headers will be used to control the I/O devices on the top board, as well as the MicroSD card jack.

We plan to use a virtual UART connection to directly interface with the DE0-nano. However, if this is rendered infeasible somehow we can include dedicated UART pins on the top board.

Term? You mean using the same USB used by Quartus to program the board?

[9] From support@terasic.com

Board Specs
Dimensions
headers
Rough sketch (or actual PCB layout)

Part 4: Arduino Expansion

The Arduino community contains a large volume of official and user-created tools which are useful for educators and hobbyists. Ensuring compatibility with Arduino libraries will allow users to save time porting code to a different platform.

Arduino Shield Header

We plan to include a shield that maps all of the pins on an Ardunio to the 26-pin header on the bottom of the DE0-nano. This includes analog pins and digital pins. We will use BSS138 transistors to ensure compatibility between 3.3V and 5V logic.

One difficulty we may encounter is the lack of a variable analog reference on the DE0-nano, and the possibility of burning out the Cyclone IV. This can be fixed by using voltage dividers to scale all signals down and using a software solution to adjust the value given to the processor, using the AREF pin on the Arduino as an input itself to determine what scaling needs to be done.

Best solution? Alternatives?

or transistors that cutoff or saturate?

Arduino Libraries

We plan to implement common Arduino functions such as DigitalWrite and DigitalRead in order to allow seamless integration of Arduino libraries.

Circuit protection and leave it at that?

Standards

Our primary objective this quarter is to implement IEEE Nexus functionality on our working microprocessor system. There are a number of standards which we are working with/towards.

MIPS Instruction set

The mips32r1 processor uses bytecode from the MIPS instruction set. The GCC cross-compilation tools convert C programs into bytecode used by the processor, along with any Assembly language files included with the program. We are not modifying the instruction set.

C Language

Our end-goal is to produce a Class I Nexus-capable device that runs C programs. Regardless of what USB-JTAG interface and computer software we use, the intent will be to run and debug programs written in C and uploaded to the processor's memory.

Verilog

The processor and hardware-side I/O components are written in SystemVerilog using Altera's Quartus Prime software for editing and project management. The project archive contains a Quartus Project File (.qpf) which links together all the related Verilog files under the top-level-entity "mips.sv."

USB/UART

In order to set breakpoints and upload code easily, we plan to connect to our device using a virtual UART connection to negate the need for dedicated UART hardware.

The current processor implementation uses a simple UART bootloader and a program written in C# for uploading to the board. It requires dedicated USB hardware, but the DE0-nano can also use USB.

IEEE Nexus Class I

The first NEXUS compliance class consists of runtime control using the JTAG interface. A Class I device is able to do the following [2]:

1) Running and halting the processor
2) Memory R/W while processor is halted
3) Breakpoints
4) Register R/W

Are you really promising this?

JTAG

The JTAG IEEE 1149.1 standard uses a serial connection interface used by debugging tools with a clock pin, mode select pin, and data in/out pins, along with a 5th optional reset pin.

Would you prefer to do this or a cache if you have this much time on your hands?

Timeline

Milestone 1: Functional mips32r1

After the SDRAM controller is finished, we will have a fully-functional microprocessor with program loading via UART and a XUM bootloader. Not all of the I/O pins on the FPGA will be usable by C programs, as two pins plus a ground pin are needed to connect the UART hardware.

This version will have no debugging capability. It will, however, have capabilities similar to that of an Arduino aside from retaining flashed programs in memory. Our solution could potentially replace Arduinos for embedded systems education.

Projected Completion: July 7th

Milestone 2: Virtual UART

The UART-to-USB adapter connected to GPIO pins is an inelegant solution. Furthermore we will require a JTAG interface in the future, reserving even more pins. Using a virtual solution that interfaces with the microUSB port on the DE0-nano allows us to free those pins.

Fabulous

Projected Completion: July 14th

Milestone 3: I/O and Arduino Library Integration

We will begin mapping I/O signals such as the ADC and accelerometer into the processor's memory space, allowing C programs to use those signals.

We also need to implement simple Arduino functions and test Arduino programs/shields on the DE0-nano for cross-compatibility.

Projected Completion: Mid July

Milestone 4: Prototype PCBs

We plan to build prototype PCB's for both the top and the bottom and verify that they work, in addition to creating drivers for the various devices on each board.

Projected Completion: Late July

References

[1] Standard for a Global Embedded Processor Debug Interface. IEEE-ISTO. April 2012.

[2] The NEXUS Debug Standard: Gateway to the Embedded Systems of the Future. Ashling Microsystems.

[3] *256Mb Synchronous DRAM*. IS42S83200G. Integrated Silicon Solution, Inc. Dec 2013.

[4]

[5] Using the SDRAM on Altera's DE0-Nano Board with Verilog Designs. Altera Corporation. May 2011 .

[6] S. Brown and Z. Vranesic, *Fundamentals of Digital Logic With Verilog,* 3rd ed. London, United States: McGraw-Hill Education, 2013.

[7] D. Patterson and J. Hennessy, *Computer Organization and Design*, 5th ed. Morgan Kaufmann, 2013.

[8] N. Hamilton, D. Blankenburg, R. Garcia, T. Nguyen, K. Chao. University of Washington Bothell Capstone Project. University of Washington.

[9] N. Hamilton, D. Blankenburg, R. Garcia, T. Nguyen, K. Chao. Team IEEE Nexus Project Proposal. University of Washington.

~~Slide~~ Presentation to IEEE?

I'm not sure I understand these two refs.