2016

Export to Arduino: A Tool to Teach Processor Design on Real Hardware

Michael Black
Bridgewater State University, michael.black@bridgew.edu

"EXPORT TO ARDUINO": A TOOL TO TEACH PROCESSOR DESIGN ON REAL HARDWARE *

Michael Black
Department of Computer Science
Bridgewater State University
michael.black@bridgew.edu

ABSTRACT

Many computer organization courses teach digital design and processor architecture without a hardware lab or physical equipment. This paper introduces a module to allow students to export digital designs as C programs that run on an inexpensive Arduino Uno, thereby allowing students to test and observe their designs in actual hardware with minimal setup time and equipment. The module runs within Emumaker86, an open-source digital design tool previously developed by the author for teaching microprocessor architecture, and can handle designs ranging from simple combinational circuits to a complete processor. Students were given this module in an undergraduate "Systems Computing" course, and developed a traffic light controller, a postfix evaluator, and a processor of their own design.

INTRODUCTION

A single course in "Computer Organization", as taught in many liberal arts computer science programs and recommended in the ACM core curriculum [1], is typically a major's sole exposure to computer hardware. This course must consequently cover a wide range of material, including digital design, assembly language, and processor microarchitecture. One of the challenges in teaching computer hardware is exposing students to physical hardware: chips, wires, lights, switches. Ideally students would gain a tangible appreciation for digital circuits by actually building them, but this requires equipment, expertise, lab time, and setup / takedown time. Another challenge is the sheer breadth and complexity of the material makes it difficult to assign design projects. In a

course on processor design it would be nice for students to design a processor, preferably out of NAND gates, but the scale of that project and limited time makes this impractical.

Inexpensive, widely used microcontroller boards, the Arduino in particular, offer a solution to the hardware challenge. A single Arduino Uno board, containing a C programmable Atmel microcontroller, USB serial interface, and breakout pins, can be purchased from Amazon for between $5 and $20. For approximately $60, less than a typical textbook, Arduino "starter kits" are sold that include a breadboard, wires, switches, and LEDs. Several digital design software tools have been developed for teaching computer hardware [4]. These range from circuit simulators at the lowest level to CPU emulators intended for learning assembly programming. Recently, a few tools have been developed to bridge the gap and teach processor microarchitecture design [7]. These include Emumaker86 [2,3], a digital simulator developed by the author that specifically teaches undergraduates to design processors. Emumaker86's processor design engine has two components: a datapath tool for placing registers, muxes, buses, and logic gates, and a control tool for scripting the control unit's state machine. Up until now, Emumaker86, like many other educational digital simulators, can only model the hardware behavior virtually on the user's computer screen. The product of the research described in this paper is a tool that takes a simulated Emumaker86 processor circuit and compiles it to an identical C program that runs on an Arduino board. A student, after composing and simulating a circuit, calls this "Export" tool, uploads its output to an Arduino board, connects LEDs and switches to the appropriate Arduino pins, and can physically see and test the design. Additionally, the student can now use this reprogrammed Arduino to integrate the student's design into a larger circuit.

A model for this work is programmable logic chips, such as FPGAs, that are programmed using a hardware descriptor language such as Verilog [6]. The disadvantage of these compared with Arduinos is cost and learning curve, both in installing and using the software and learning a new language. Here, in contrast, students draw their circuits graphically, and the Export tool autogenerates the C code, which the students are not required to understand. The author designed this Export tool to instruct a "Systems Computing" course with 16 students. Three multiweek design projects were created for this course and assigned to students in teams of two. These projects were: 1) a traffic light controller, 2) a stack-based postfix expression evaluator, 3) a complete 8-bit processor of the student's own design; and are explained in greater detail in this paper. The Export tool was developed concurrently with the course, and student experience and feedback was used in its development.

The Emumaker86 simulator with the Export tool integrated is posted at https://github.com/mdblack/simulator.html and are freely distributed under the GPL license.

THE EXPORT TOOL

The Export Tool, while written initially as a standalone application, is packaged as a module within Emumaker86 and called with a button-press within that program. It generates an Arduino .ino project file, which the student then loads into the Arduino utility and flashes to the board.

Emumaker86 designs are drawn in a graphical interface and saved in a custom XML format. Each component has a type, an identifier, a bit-width, and an X-Y coordinate for display. Buses include, additionally, a sink, provided the bus's output connects to a non-bus. This format, which was intended solely for reproducing the circuit visually, is used as input to the Export tool. The output is a C program, in the Arduino dialect, where each component is modeled in code. An Arduino program has two basic functions: *setup()* which runs once, and *loop()* which runs repeatedly as fast as possible. The Export tool generates this Arduino code as follows:

1. A global array *int table[]* is made to store the current value of each component in the circuit.

2. In *setup()*, Arduino pins are mapped to input and output pins in the circuit. Comments are provided in the code for the student that tell which Arduino pins map to which I/O pins.

3. In *loop()*, each combinational component is translated to a single line of C code. Figure 1 shows how a D flip-flop consisting of basic logic gates is converted into the equivalent C statements. Also supported are adders, negators, comparators, bus splitters and joiners, and lookup tables (ROMs). The contents of lookup tables are stored as global arrays.

4. In a final post-processing stage, all units are renumbered so that the table[] is as small as possible.

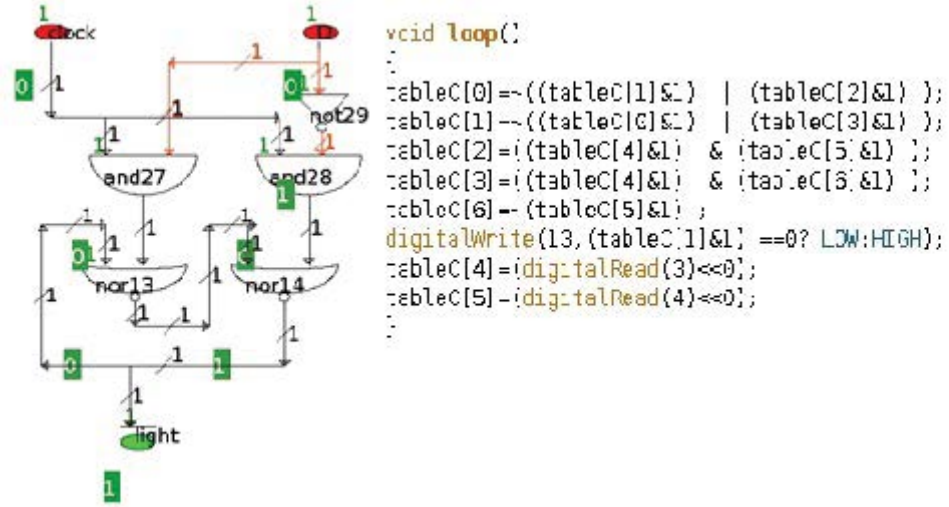


Figure 1: Example Emumaker86 circuit and the autogenerated C code

The C program models the circuit by iteratively converging on the correct result. Initially only the input pins have a correct value. The second time *loop()* runs, components sourced by the input pins produce a correct value, the third time components two away from the inputs are corrected, and so on. As the Arduino Uno has a clock rate of 16MHz, this appears instantaneous to the user. Sequential units — registers (edge-triggered D flip flops) and register files (tables of registers that can be indexed) — are handled as a special case. All registers are synchronized to a global clock pin assigned to Arduino pin 2. A second global *registerInput[]* holds incoming values. When the

clock pin changes from low to high, all registerInputs are copied to their appropriate table entries. An optional disable input to the register will prevent it from being clocked.

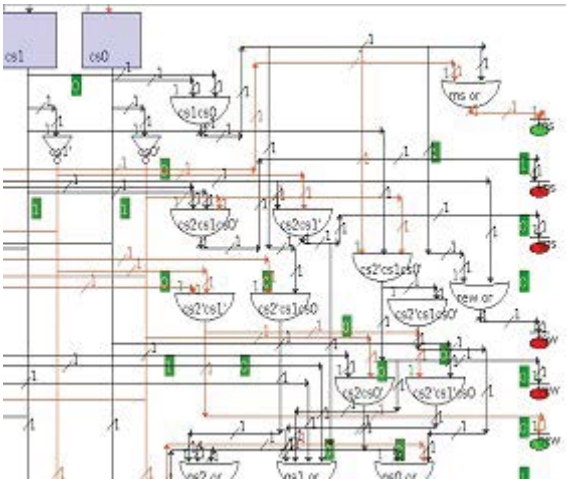
In the initial development, buses were also translated as simple assignment statements, such as *table[2]=table[1]*; This resulted, however, in massive programs that would not fit on an Arduino Uno, as many students would draw far more buses than they needed. An additional translation stage was consequently added. Each bus is traced back to the component sourcing it, that component is directly connected to the bus's output, and the bus is removed from the C program. This tended to reduce the program size by 60-70%.

THE COURSE

The author tested the tool on an elective course while it was being developed. "Systems Computing" was a one-off special topics course open to any student who had completed the basic programming requirements. 16 students took the class. Of these students, 8 reported in an anonymous survey having learned assembly language previously and 4 said that they had circuit wiring experience. For materials, students were required to purchase an Arduino Uno and encouraged to buy a "starter kit" with breadboard, wires, and LEDs. For students who did not get the starter kit, some breadboards and parts were made available.

PROJECTS

The author developed three projects for this course: a traffic light controller, a postfix evaluator, and a simple processor. The first project, assigned approximately three weeks into the semester, had students design a state machine for a T-junction traffic intersection between a busy road and a smaller road. The outputs of the circuit are six LEDs: red, yellow, and green for each direction; the input represents whether a car is present on the smaller road. A state diagram is provided to the students, but they must produce the equations, simplify them, and construct the circuit entirely out of basic gates and registers. Figure 2 shows, partially, a student Emumaker86 circuit and the student's Arduino circuit. The project was successfully completed by all 16 students.



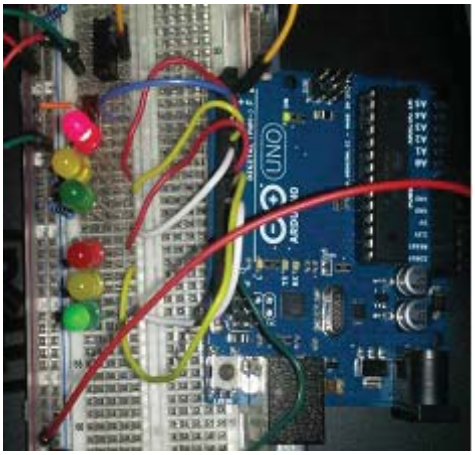


Figure 2: Traffic Light

The second project required students to build a 4-bit stack-based evaluator with a small 3-bit instruction set inspired by Java byte code:

```
000,001,010 iconst_0, iconst_1, iconst_5: push 0, 1, 5 to the stack
011,100 add, sub: pop two values, add or subtract, push the result
101 swap: pop two values and push them back in reverse
110 pop: pop a value and display it to an output pin
```

Unlike a true processor, there is no instruction storage or branch instructions. The input to the circuit are 3 push buttons for the instruction and a single "Clock" button, the output is the 4 LED result. However, like real processors, some instructions (add, sub, swap) took 4 cycles; the student must press Clock four times to see the result. Students demonstrated their work by running the test program:

```
1 1 + 5 SWAP - POP
```

If the lights show *0011 (3)*, the student was given full credit. For this assignment students were given detailed instructions: they started by building a stack with push, pop, and no-action operations, and then constructing an adder and swap mechanism around their stack. Unlike the first project, they were encouraged to use multiplexors instead of gates and program lookup tables to control them. Figure 3 shows part of a student's design and the resulting board. The project was successfully completed by 2/3 of the students.

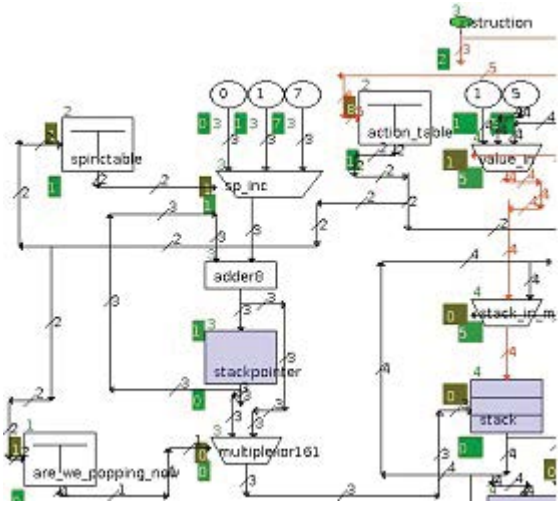
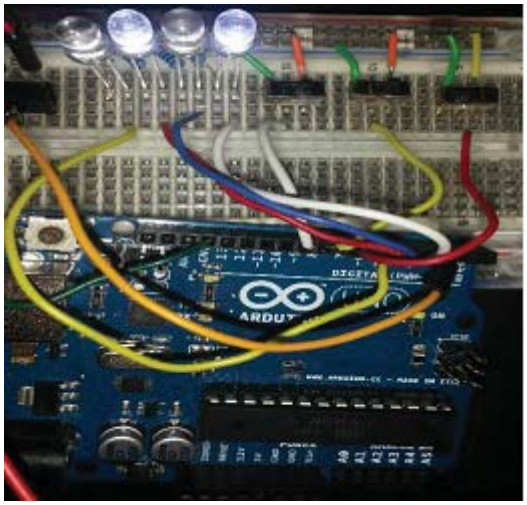


Figure 3: Postfix Evaluator

For the third project, students built an 8-bit processor to implement a simple instruction set. A small example instruction set was given to the students, but for full credit they had to design and

encode their own. The instruction set had to be capable of running a small test program that would iteratively sum numbers 1+2+...5, output the result, and then halt in an infinite while loop. Students implemented the processor using the same components as in the second project, and used a lookup table as an instruction ROM. They then programmed this instruction ROM with their machine code and uploaded it to an Arduino. A working project would have a push button Clock input and 8 LEDs. The student would press the Clock a large number of times (50-150 times, depending on their implementation), and the result 00001111 would appear on the output.



ACKNOWLEDGEMENTS

The author wishes to thank Dylan Tierney for allowing him to use pictures of his work in this paper.

REFERENCES

[1] ACM/IEEE Joint Task Force on Computing Curricula. Curriculum Guidelines for Undergraduate Degree Programs in Computer Science , 2013.

[2] Black, M., Komala, P. A full system x86 simulator for teaching computer organization. SIGCSE Technical Symposium on Computer Science Education, 2011.

[3] Black, M., Waggoner, N. Emumaker86: A Hardware Simulator for Teaching CPU Design, SIGCSE Technical Symposium on Computer Science Education, 2013.

[4] Burch, C. A graphical system for logic circuit design and simulation. Journal on Educational Resources in Computing 2:1, 2002, pages 5-16

[5] Garton, J. 2005. ProcessorSim - a visual MIPS R2000 Processor Simulator. http://jamesgart.com/procsim/

[6] Williams, S. http://iverilog.icarus.com/

[7] Wolffe, G. ,W.Yurcik,H.Osborne,M.Holliday. Teaching Computer Organization/Architecture with Limited Resources using Simulators. ACM SIGCSE Bulletin Volume 34, Issue 1, Pages 176-180.