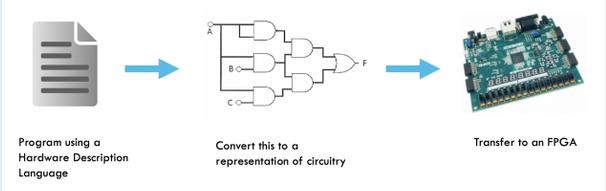
A Hardware Engine For Generating Number-Theoretic Sequences

Madeline Burbage Advisor Duane Bailey Williams College Computer Science

Background - Reconfigurable Computing

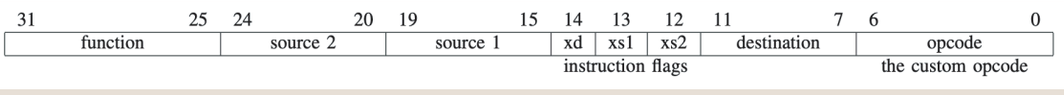
Reconfigurable Computing allows developers to design, and re-design, hardware circuitry as many times as they like. There have been many recent advancements in this field, allowing for quick hardware design to speed up specific software tasks [4].
Open-source hardware design tools make this process more accessible [2]. We used Chisel, a high-level hardware description language, to build a hardware accelerator for specific functions.



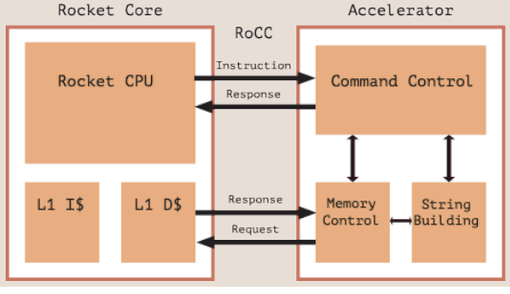
This accelerator targets the highly configurable, open-source Rocket Chip processor, which implements the RISC-V open ISA [1]. The instruction above is customized to send sequence-generating constraints to our accelerator when called.

Approach - Designing and Simulating the Hardware Engine

Our hardware accelerator implements the cool, cooler, and coolest sequence generating rules. It receives custom instructions from the Rocket Chip, containing the rule to use, the string length to generate, and the weight ranges allowed. We also created two methods of string generation: either returning the next string in a sequence to the CPU, or saving a full sequence of strings to a section of the cache.



Since our instructions use register addressing, we limited the length of possible strings to 32 or less. On 64-bit systems this could easily be extended.

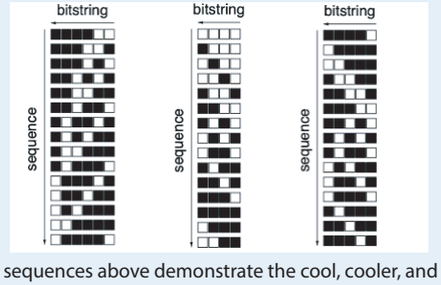


At left is the layout of the chip and our accelerator:
- Command Control receives chip input and handles accelerator functionality.
- Memory control interfaces with the Rocket Chip's cache to send strings there when called.
- The String Building unit controls actual string generation for a sequence.

We implemented our accelerator with the lowRISC variant of the Rocket Chip on the Nexys A7 FPGA board. We also tested it on the Rocket Chip with a cycle-accurate simulator, Verilator. In simulations, we measured speedup in our accelerator as compared to the software version of the 3 string-generating functions and both return styles. We tested string lengths of 2, 4, 8, 16, and 24 for immediate-response functions, and 2, 4, 8, and 13 for memory-response functions.

Motivation - Combinatoric Sequences

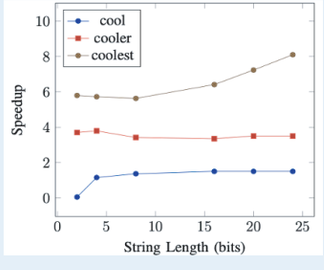
One task that can benefit from hardware acceleration is Combinatorial Sequence Generation. These generated sequences of binary strings have applications in software and hardware [3].
Each string describes a way to choose items from a set given certain constraints (if bit i is high, item i is selected). Together, a sequence of strings enumerates all possible selections for a set of constraints.



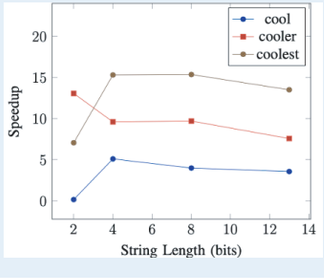
The three sequences above demonstrate the cool, cooler, and coolest rules for sequence generation, for strings of different lengths and weight ranges [8]. These rules can be implemented in constant amortized time.
The three rules can also be implemented statelessly, at a cost in speed [5]. However, this loss can be overcome by converting these software methods into hardware.

Results and Conclusions

Immediate Response Functions



Memory Response Functions



The cool, cooler, and coolest functions all show speedup but not to the same extent. This is because the three functions grow progressively more complex, increasing latency in software. However, all the functions take the same amount of time in hardware: 1 cycle to generate a string, and usually several cycles to transfer control back from the accelerator to the CPU.

Speedup in memory-response functions is around twice as high as with immediate-response functions, which is explained by the removal of costly synchronizing cycles between hardware and software. With memory-response functions, strings are streamed to memory as fast as the cache can accept them.

The speedup found for every function type indicates the effectiveness of hardware acceleration for mathematically interesting successor functions. Our approach demonstrates the efficiency of using an open-source toolchain to accelerate specific software problems. We also see this work as a well-documented contribution to a growing collection of accelerators for RISC-V processors.

References

[1] Asanović, K., Avizienis, R., Bachrach, J., Beamer, S., Biancolin, D., Celio, C., Cook, H., Dabbelt, D., Hauser, J., Izraelevitz, A., Karandikar, S., Keller, B., Kim, D., Koenig, J., Lee, Y., Love, E., Maas, M., Magyar, A., Mao, H., Moreto, M., Ou, A., Patterson, D. A., Richards, B., Schmidt, C., Twigg, S., Vo, H., and Waterman, A. 2016. The Rocket Chip Generator. Technical Report. University of California, Berkeley.
[2] Bachrach, J., Vo, H., Richards, B., Lee, Y., Waterman, A., Avižienis, R., Wawryzynek, J., and Asanović, K. 2012. Chisel: Constructing hardware in a Scala embedded language. In DAC Design Automation Conference (June 2012). 1212-1221.
[3] Golomb, S. W. 2017. Shift Register Sequences, 3rd revised ed. World Scientific, 2017.
[4] Hauck, S., and DeHon, A. 2010. Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation. ISSN. Elsevier Science.
[5] Knuth, D. 2005. The Art of Computer Programming, Volume 4, Fascicle 3. Pearson Education.
[6] Mao, H. 2017. Hardware acceleration for memory to memory copies. Master's Thesis. University of California, Berkeley.
[7] Schoeberl, M. 2019. Digital Design with Chisel. Kindle Direct Publishing.
[8] Stevens, B., and Williams, A. 2014. The coolest way to generate binary strings. Theory of Computing Systems 54, 4 (May 2014), 551-557