Pipeline Datapath Models from RISC-V based cores

Samira Ait Bensaid, Mihail Asavoae, Farhat Thabet and Mathieu Jan
Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France

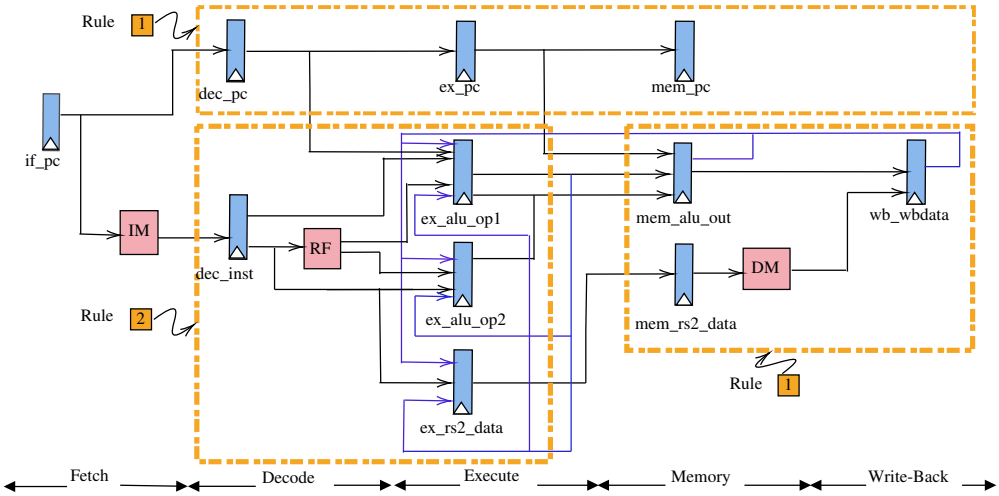

Fig. 1: RISC-V Sodor 5-stages pipeline datapath model.

Motivation. Certification of safety-critical systems requires the use of timing analyses to estimate Worst-Case Execution Times (WCET). These timing analyses reason about the executions of a program on an underlying computer architecture. WCET analyzers (such as [4], [7], [12]) use architecture models, generally built by hand, and static analysis to estimate these WCETs. Generating hardware models have mainly focused on functional verification of Verilog/VHDL designs ([6], [8], [10], [11]) and not for such WCET analyses, except [14]. However, hardware designers tend to use higher-level and more expressive languages, such as Chisel [3] or SpinalHDL for instance. There is thus a need for a fully automated construction of (abstract) datapath pipeline models from these higher-level hardware construction languages.

Contributions. We currently design a custom FIRRTL [9] pass to automate the construction of datapath pipeline models from Chisel [3] HDL processor designs. This pass focuses on the registers involved in the pipeline datapath, determines its depth and generates a model of it (see Fig. 1 for an example). We explore both the combinatorial and sequential logics of a pipeline in order to extract dependency relations between registers. Next, we assign a pipeline stage to the identified registers using two different rules. Rule 1 relies on register dependencies and takes into account forwarding mechanisms within micro-architectures. Rule 2 relies on a heuristic by taking advantage of a common practice of hardware designers to simultaneously update registers of a same pipeline stage within a same conditional block. This procedure is described in more details in [5]. Specific abstractions can then be applied to match input (timing) models used by WCET analyzers.

Results. The following table reports the effectiveness of our pass on a set of in-order RISC-V processors, ranging from 3 to 5 pipline stages. The first column describes the code size of each (datapath) pipeline, the next column presents the number of registers ($\#Regs$) and the last two columns summarize the number of registers successfully placed by each rule. For each processor, the depth of each pipeline is correctly computed and Fig 1 illustrates a subset of the datapath pipeline model of RISC-V Sodor. Starting from a given register located in the fetch stage (e.g. `if_pc`), our pass assigns registers `dec_pc`, `ex_pc` and `mem_pc` to respectively decode, execute and memory pipeline stages thanks to rule 1. Then, it assigns register `dec_inst` using rule 2, as it is updated in the same context as `dec_pc`. The procedure is then recursively applied to the next registers (`ex_alu_op1`, etc.) and identifies forwarding mechanisms represented by the blue edges on Fig 1.

	LOC	$\#Regs$	Rule 1	Rule 2
RISC-V Mini [1]	241	15	5	10
Sodor [2]	646	48	34	14
KyogenRV [13]	4567	93	47	36

References

[1] Risc-v mini. `https://github.com/ucb-bar/riscv-mini`,
[2] Risc-v sodor. `https://github.com/ucb-bar/riscv-sodor`,
[3] Bachrach, J., Vo, H., Richards, B., Lee, Y., Waterman, A., Avižienis, R., Wawrzynek, J., Asanović, K.: Chisel: Constructing hardware in a scala embedded language. In: DAC. p. 1216–1225 (2012)
[4] Ballabriga, C., Cassé, H., Rochange, C., Sainrat, P.: OTAWA: an open toolbox for adaptive WCET analysis. In: SEUS (2010)
[5] Bensaid, S.A., Asavoae, M., Thabet, F., Jan, M.: WiP: Automatic Construction of Pipeline Datapaths from High-Level HDL Code. In: RTASS (2022), to appear.
[6] Charvát, L., Smrcka, A., Vojnar, T.: HADES: microprocessor hazard analysis via formal verification of parameterized systems. In: MEMICS. EPTCS, vol. 233, pp. 87–93 (2016)
[7] Hardy, D., Rouxel, B., Puaut, I.: The heptane static worst-case execution time estimation tool. In: WCET. OASICS, vol. 57, pp. 8:1–8:12 (2017)
[8] Irfan, A., Cimatti, A., Griggio, A., Roveri, M., Sebastiani, R.: Verilog2SMV: A tool for word-level verification. In: DATE (2016)
[9] Izraelevitz, A.M., Koenig, J., Li, P., Lin, R., Wang, A., Magyar, A., Kim, D., Schmidt, C., Markley, C., Lawson, J., Bachrach, J.: Reusability is FIRRTL ground: Hardware construction languages, compiler frameworks, and transformations. In: ICCAD. pp. 209–216 (2017)
[10] Jain, H., Kroening, D., Sharygina, N., Clarke, E.: VCEGAR: Verilog CounterExample Guided Abstraction Refinement. In: TACAS (2007)
[11] Lee, S., Sakallah, K.A.: Unbounded scalable verification based on approximate property-directed reachability and datapath abstraction. In: CAV. pp. 849–865 (2014)
[12] Li, X., Yun, L., Mitra, T., Roychoudhury, A.: Chronos: A timing analyzer for embedded software. Sci. Comput. Program. **69**(1-3), 56–67 (2007)
[13] Saitoh, A.: KyogenRV: simple 5-staged pipeline RISC-V. `https://github.com/panda5mt/KyogenRV`,
[14] Schlickling, M., Pister, M.: A framework for static analysis of VHDL code. In: WCET. OASICS, vol. 6 (2007)