Hindawi Publishing Corporation
Research Letters in Electronics
Volume 2009, Article ID 480740, 5 pages
doi:10.1155/2009/480740

Research Letter

Token-Aware Completion Functions for Elastic Processor Verification

Sudarshan K. Srinivasan, Koushik Sarker, and Rajendra S. Katti

Department of Electrical & Computer Engineering, North Dakota State University, Fargo, ND 58105, USA

Correspondence should be addressed to Sudarshan K. Srinivasan, sudarshan.srinivasan@ndsu.edu

Received 2 June 2009; Accepted 31 July 2009

Recommended by Massimo Poncino

We develop a formal verification procedure to check that elastic pipelined processor designs correctly implement their instruction set architecture (ISA) specifications. The notion of correctness we use is based on refinement. Refinement proofs are based on refinement maps, which—in the context of this problem—are functions that map elastic processor states to states of the ISA specification model. Data flow in elastic architectures is complicated by the insertion of any number of buffers in any place in the design, making it hard to construct refinement maps for elastic systems in a systematic manner. We introduce token-aware completion functions, which incorporate a mechanism to track the flow of data in elastic pipelines, as a highly automated and systematic approach to construct refinement maps. We demonstrate the efficiency of the overall verification procedure based on token-aware completion functions using six elastic pipelined processor models based on the DLX architecture.

1. Introduction

The impact of persistent technology scaling results in a previously ignored set of design challenges such as manufacturing and process variability and increasing significance of wire delays. These challenges threaten to invalidate the effectiveness of synchronous design paradigms at the system level. Several alternate design paradigms to deal with these challenges are being proposed. One popular trend is latency-insensitive designs, which allows for variability in data propagation delays [1]. Synchronous Elastic Networks (SENs) [2, 3] hav been proposed as an effective approach to design latency-insensitive systems.

One of the critical challenges for any design approach to succeed is verification. We present a novel highly automated formal verification solution for latency-insensitive pipelined microprocessors developed using the SEN approach (here on referred to as elastic processors). Note that correctness proofs for methods to synthesize elastic designs from synchronous designs have been provided [2], but this is not a substitute for verification. The idea with the verification approach is to show that the elastic processor correctly implements all behaviors of its instruction set architecture (ISA) model, which is used as the high-level specification for the processor. The notion of correctness that we use is Well-Founded Equivalence Bisimulation (WEB) refinement, a detailed description of which can be found in [4]. It is sufficient to prove that the elastic processor (implementation) and its ISA (specification) satisfy the following core WEB refinement correctness formula to establish that the elastic processor refines (correctly implements) its ISA.

Definition 1.1 (Core WEB Refinement Correctness Formula).

$$\begin{aligned} \langle \forall w \in IMPL \quad &:: s = r(w) \wedge u = Sstep(s) \wedge \\ &v = Istep(w) \wedge u \neq r(v) \\ &\rightarrow s = r(v) \wedge rank(v) < rank(w) \rangle. \end{aligned} \quad (1)$$

In the formula above, *IMPL* denotes the set of implementation states, *Istep* is a step of the implementation machine, and *Sstep* is a step of the specification machine. The refinement map r is a function that maps implementation states to specification states. In fact, the refinement map can be thought of as an instrument to view the behaviors of the implementation machine at the specification level, thereby allowing verification tools to easily compare the behaviors of the two systems. *rank* is used for deadlock detection. Our focus in this work is to check safety, that is, to show that if the implementation makes progress, then, the result of that progress is correct as specified by the high-level specification. We plan to address deadlock detection in future work, and we therefore ignore *rank* for the present.

The specific steps involved in a refinement-based verification methodology are (a) construct models of the specification and implementation, (b) compute the states of the implementation model that are reachable from reset (known as reachable states), (c) construct a refinement map, and (d) the models and the refinement map can now be used to state the refinement-based correctness formula (excluding deadlock detection) for the implementation model, which can then be automatically checked for the set of all reachable states using a decision procedure. Modeling and verification are performed using ACL2-SMT [5], a system developed by combining the ACL2 theorem prover (version 3.3) with the Yices decision procedure (version 1.0.10) [6].

The primary challenge in applying the refinement-based approach to elastic pipelines is as follows. The very attractive property of elastic systems is that they allow for the insertion of buffers (known as elastic buffers) in any place in the data path to deal with propagation delays of long wires, without altering the functionality of the system. The insertion of these buffers however can drastically change the data flow patterns of the system, making it hard to compute refinement maps for these systems. *Our primary contribution is a procedure—we call token-aware completion functions—that computes refinement maps for elastic pipelined systems (described in Section 3) even after the insertion of elastic buffers in any place in the data path.* The procedure allows for highly automated and efficient verification of elastic pipelined systems. The effectiveness of our verification method is demonstrated using 6 DLX-based elastic pipelined processor models. The models are described in Section 2. Verification results are given in Section 4, and we conclude in Section 5. Due to limited space, we request the reader to refer to literature for background on synchronous elastic networks [2, 3] and refinement [4].

Note that this is the first known approach that aims to verify the correctness of elastic pipelined processors against their high-level nonpipelined ISA specifications. In previous work, we have developed an equivalence checking approach that is used to verify elastic pipelines against their synchronous parent pipelines [7].

2. Elastic Processor Models

The elastic processor models are based on the 5-stage DLX pipeline. The elastic processor models and their nonpipelined ISA-level specifications are described using the ACL2 programming language and are defined at the term-level, because term-level abstractions make the verification problem tractable. We use the ACL2-SMT system for verification as it can be used to reason at the term-level. Note that bit-level versions of these models were used in [7]. The models were obtained by first elasticizing a synchronous 5-stage DLX processor using the Synchronous Elastic Flow (SELF) protocol approach [2]. The main idea is to replace all flip flops with elastic buffers (EBs) that are constructed from two elastic half buffers (EHBs), namely, a master EHB and a slave EHB. The clock network is replaced by a network of elastic controllers, where each controller is used to control the elastic buffers in a pipeline stage and synchronized with the controllers of adjacent pipeline stages. The controllers are synchronized with the clock and are connected in accordance with connections between pipeline stages in the data path. Each controller has three possible states, *empty*, *half*, and *full*, which indicate that the corresponding elastic buffer has 0, 1, and 2 valid data tokens, respectively.

We call the processor model obtained by elasticizing the synchronous DLX $M0$. The main advantage of the elastic processor is that it permits the insertion of additional elastic buffers at any place in the data path to break long wires. We therefore inserted additional elastic buffers $l1, \dots, l5$ at various places in the model. We inserted $l1$ in model $M0$ to get model $M1$. We then inserted $l2$ in model $M1$ to get $M2$. We derived models $M3$, $M4$, and $M5$ in a similar manner. The model M5 is shown in Figure 1. The figure also shows the positions of the additional elastic buffers and how they are connected with the elastic buffers corresponding to the pipeline latches (namely pc, fd, de, em, and mm). The network of elastic controllers for the DLX processor with five additional elastic buffers in the data path is shown in Figure 2. These models are used to demonstrate the effectiveness of our verification approach.

3. Token-Aware Completion Functions

Flushing [8] is one standard approach used to compute refinement maps for pipelined processors. In this approach, partially executed instructions in the pipeline latches are forced to complete, without allowing the machine to fetch any new instructions. Projecting out the programmer visible components—which include the program counter, register file, instruction memory, and data memory for the models we consider—in the resulting state will give the corresponding ISA state.

Completion functions [9] were proposed as a computationally efficient approach to construct flushing refinement maps. One completion function for each pipeline latch in the machine is used to compute the effect on the programmer visible components of completing any partially executed instruction in that latch. The completion functions are composed to form the flushing refinement map. Note that older instructions in the pipeline are completed before younger instructions. For the DLX example, let *fdc*, *dec*, *emc*, and *mmc* be the completion functions for the latches *fd*, *de*, *em*, and *mm*, respectively. Let *rf*, *im*, and *dm* be the register file, instruction memory, and the data memory of the processor model. The ISA state s corresponding to a synchronous DLX processor state w ($\langle pc^w, fd^w, de^w, em^w, mm^w, rf^w, im^w, dm^w \rangle$) is $\langle pc^s, rf^s, im^s, dm^s \rangle = fdc(dec(emc(mmc(\langle pc^w, rf^w, im^s, dm^w \rangle, mm^w), em^w), de^w), fd^w)$.

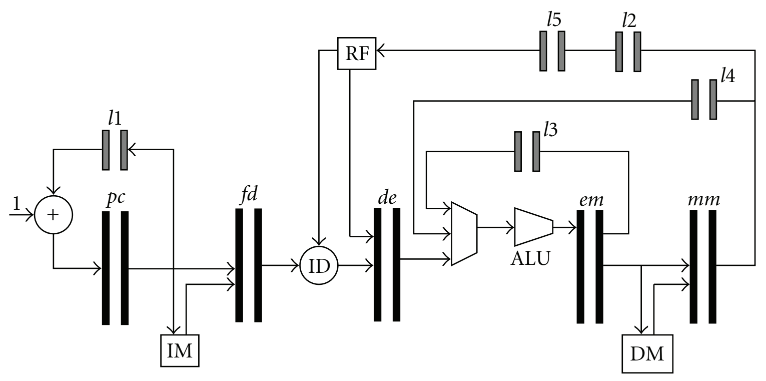

FIGURE 1: High-level organization of elastic 5-stage DLX processor with five additional elastic buffers $l1, l2, \ldots, l5$.

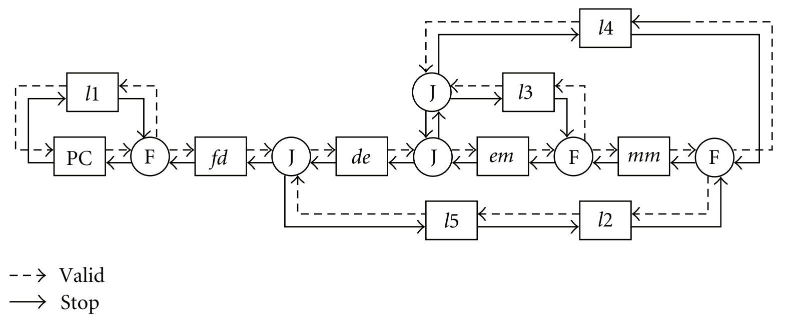

FIGURE 2: Network of elastic controllers for the elastic 5-stage DLX processor shown in Figure 1. The J and F blocks denote the join and fork circuits. Valid is used to indicate valid data in the datapath. Stop is used to indicate that EB cannot accept data at this time.

TABLE 1: Verification times and SMT statistics.

Elastic	Yices		Total
Models	Bool Vars	Time (sec)	Time (sec)
$M0$	887	0.22	1.07
$M1$	1,101	0.46	2.29
$M2$	1,889	1.05	5.07
$M3$	2,811	1.32	6.17
$M4$	3,096	3.29	16.65
$M5$	3,605	4.16	24.42

When we try to apply the completion functions approach to elastic pipelined processors, two issues arise. First, in some states of the elastic processor, instructions can be duplicated in the data path; that is, an instruction can reside in two pipeline latches. Such a situation can occur at a fork when the instruction in a buffer before the fork has proceeded along one path of the fork, but the other path is blocked. The latch before the fork has to retain the instruction until both paths are cleared. A direct application of the completion functions-based map to such a state will result in completing the same instruction twice leading to an erroneous refinement map. Second, Elastic Half Buffers (EHBs) need not have valid tokens. The contents of such EHBs should be ignored and should not be used to update the programmer visible components.

We introduce token-aware completion functions as a method to compute flushing-based refinement maps for elastic pipelined processors. The idea being that EHBs which are either holding duplicate instructions or are in an empty state should not be completed. This is achieved by first computing the reachable states of the elastic controller network. We use token-flow diagrams proposed in [7] to compute the reachable states of the system. The reachability analysis is performed by simulating how tokens flow in the elastic architecture using a form of symbolic simulation. The output of the token-flow diagrams is a set of token-states, one token-state for each reachable state. In a token-state, each EHB is assigned a numbered token, which is essentially a natural number. A value of “0” indicates a bubble; that is, the EHB is empty. Also, EHBs with the same instruction will be assigned the same token numbers. Thus, using the token-state, duplicate instructions and empty EHBs can be identified.

The token-aware completion functions approach works by first computing a two-dimensional array; we call *token-array*. Each row in the array corresponds to a reachable state of the elastic controller network. Each element in a row is a binary value. The number of elements in a row is $2n$, where n is the number of pipeline latches in the elastic system. If $\textit{token-array}(i, j) = 1$, then the contents of EHB H_j in the reachable state S_i should be completed. If $\textit{token-array}(i, j) = 0$, then the contents of EHB H_j in the reachable state S_i should be ignored when computing the refinement map. Given the set of token-states (which are the reachable states represented using numbered tokens) of the elastic controller network of an elastic system, Procedure 1 computes the *token-array* for the elastic system.

Procedure 3.1.

```
In: $S_R$, set of token-states of the elastic controller network and $PH$, the ordered set of pipeline half buffers. The number of token states ($|S_R|$) is $r$. The number of pipeline half buffers ($|PH|$) is $2n$, where $n$ is the number of pipeline latches. The order of the pipeline half buffers is determined by the position of the buffer in the pipeline; that is, buffers closer to the end of the pipeline have a higher index.
Out: token-array for the elastic system.
  (1) Initialize $i$ to $r$.
  (2) Initialize $V_t$ (the set of visited tokens) to $\{0\}$. The token number "0" represents a bubble. Note that initializing $V_t$ to $\{0\}$ causes the procedure to assign a "0" value to the empty EHBs in the token-array.
  (3) Initialize $j$ to $2n$.
  (4) Let $t = token(S_i, PH_j)$, where token is a look-up function that gives the token number for EHB $PH_j$ in token-state $S_i$.
  (5) $token\text{-}array(i, j) = \neg(t \in V_t)$
  (6) Assign $V_t = V_t \cup \{t\}$: add the token number of EHB $PH_j$ to the visited token set.
  (7) If $j - 1 \neq 0$, decrement $j$ and go to step 4.
  (8) If $i - 1 \neq 0$, decrement $i$ and go to step 2.
```

Procedure 2 takes as input the *token-array* and computes the flushing refinement map for the elastic system using completion functions.

Procedure 3.2.

```
In: Elastic processor state $w$: $\langle P_1, \ldots, P_m, H_1, \ldots, H_{2n} \rangle$. $P_1, \ldots, P_m$ are the programmer visible components, and $H_1, \ldots, H_{2n}$ are the half buffers in the pipeline latches of the elastic machine.
Out: ISA state $s$ obtained by applying the flushing refinement map to $w$.
  (1) Let $S_{2n+1} = \langle P_1, \ldots, P_m \rangle$.
  (2) Initialize $i$ to $2n$.
  (3) $r = reachable\text{-}state(w)$, gives the number of the reachable elastic controller network state of $w$, assuming that the reachable states are numbered.
  (4) One has
      $$S_i = \begin{cases} completion(S_{i+1}, H_i), & \text{if } token\text{-}array(r, i) = 1, \\ S_{i+1}, & \text{otherwise.} \end{cases} \quad (2)$$
  (5) If $i - 1 \neq 0$, decrement $i$ and go to step 3.
  (6) Then, $s = S_1$.
```

Example 3.3. The elastic controller network of the $M5$ processor model has two reachable states S_1 and S_2. The token-states T_1 and T_2 (given as a vector of token numbers for the EHBs in $M5$ in the order $\langle pc \mid fd \mid de \mid em \mid mm \mid l1 \mid l2 \mid l3 \mid l4 \mid l5 \rangle$) corresponding to these reachable states S_1 and S_2, respectively, are $\langle 0,7 \mid 0,6 \mid 0,5 \mid 0,4 \mid 0,0 \mid 0,0 \mid 0,0 \mid 0,3 \mid 0,0 \mid 0,3 \rangle$ and $\langle 0,0 \mid 7,6 \mid 0,5 \mid 0,0 \mid 0,4 \mid 0,7 \mid 0,4 \mid 0,3 \mid 0,3 \mid 0,0 \rangle$ [7]. Note that there are two tokens in the token-states for each EB, one corresponding to the master EHB and the other corresponding to the slave EHB. The completion function-based refinement map obtained using Procedures 3.1 and 3.2 for any state w of processor model $M5$ whose elastic controller network is state S_1 is $\langle pc^s, rf^s, im^s, dm^s \rangle = fdc(dec(emc(mmc(\langle pc^w, rf^w, im^s, dm^w \rangle, l5_s^w), em_s^w), de_s^w), fd_s^w)$. The completion function-based refinement map obtained using Procedures 3.1 and 3.2 for any state w of processor model $M5$ whose elastic controller network is state S_2 is $\langle pc^s, rf^s, im^s, dm^s \rangle = fdc(fdc(dec(mmc(\langle pc^w, rf^w, im^s, dm^w \rangle, mm_s^w), de_s^w), fd_s^w), fd_m^w)$.

4. Results

The token-aware completion functions approach was used to verify safety for six elastic pipelined processors $M0, \ldots, M5$. The results are shown in Table 1. Verification was performed using the ACL2-SMT system. The ACL2-SMT system incorporates a translator that reduces the correctness theorem to a decision problem in the form of a formula in a decidable logic that Yices can handle. The decision problem is then checked by Yices. Column "Bool Vars" gives the number of Boolean variables in the decision problem. The experiments were conducted on a 1.8 GHz Intel (R) Core(TM) Duo CPU, with an L1 cache size of 2048 KB. As can be seen from the table, each of the elastic 5-stage DLX-based processors was verified against the high-level instruction set architecture (ISA) within 25 seconds, thereby demonstrating the high efficiency of our approach.

5. Conclusions

We have developed a method for checking the correctness of elastic pipelined processors against their high-level instruction set architectures. The approach was demonstrated by verifying 6 DLX-based elastic processor models. For future work, we plan to further explore the scalability of the verification method.

References

[1] L. P. Carloni, K. L. McMillan, and A. L. Sangiovanni-Vincentelli, "Theory of latency-insensitive design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 9, pp. 1059–1076, 2001.

[2] J. Cortadella, M. Kishinevsky, and B. Grundmann, "Synthesis of synchronous elastic architectures," in *Proceedings of the 43rd annual Design Automation Conference (DAC '06)*, E. Sentovich, Ed., pp. 657–662, San Francisco, Calif, USA, July 2006.

[3] S. Krstic, J. Cortadella, M. Kishinevsky, and J. O'Leary, "Synchronous elastic networks," in *Formal Methods in Computer Aided Design (FMCAD '06)*, pp. 19–30, IEEE Computer Society, San Jose, Calif, USA, November 2006.

[4] P. Manolios, *Mechanical Verification of Reactive Systems*, Ph.D. thesis, University of Texas, Austin, Tex, USA, August 2001, http://www.ccs.neu.edu/home/pete/research/phd-dissertation .html.

[5] S. K. Srinivasan, *Efficient verification of bit-level pipelined machines using refinement*, Ph.D. thesis, Georgia Institute of Technology, December 2007, http://etd.gatech.edu/theses/available/etd-08242007-111625/.

[6] Yices, 2007, http://fm.csl.sri.com/yices/.

[7] S. K. Srinivasan, K. Sarker, and R. S. Katti, "Verification of synchronous elastic processors," to appear in *IEEE Embedded Systems Letters*.

[8] J. R. Burch and D. L. Dill, "Automatic verification of pipelined microprocessor control," in *Proceedings of the 6th International Conference on Computer Aided Verification (CAV '94)*, vol. 818 of *Lecture Notes in Computer Science*, pp. 68–80, Springer, Stanford, Calif, USA, June 1994.

[9] R. Hosabettu, M. Srivas, and G. Gopalakrishnan, "Proof of correctness of a processor with reorder buffer using the completion functions approach," in *Proceedings of the 11th International Conference Computer Aided Verification (CAV '99)*, N. Halbwachs and D. Peled, Eds., vol. 1633 of *Lecture Notes in Computer Science*, Springer, Trento, Italy, July 1999.