Desynchronization: Design For Verification

Sudarshan K. Srinivasan and Raj S. Katti

***Abstract*—Desynchronization is used to synthesize asynchronous circuits from synchronous specifications. Controller networks used for desynchronization are highly nondeterministic and are not easily amenable for verification. We adapt the desynchronization controllers for verifiability by imposing additional sequential dependencies among controller events that reduces nondeterminism. We deduce properties of the adapted controllers, which we use to develop methods for reachability analysis and verification of desynchronized circuits. The methods are demonstrated using seven desynchronized processor models.**

I. Introduction

The impact of persistent technology scaling results in a previously ignored set of design challenges such as manufacturing and process variability, and increased significance of wire delays. The challenges threaten to invalidate the effectiveness of synchronous design paradigms at the system-level. Asynchronous circuits provide several alluring properties—over their synchronous counterparts—that pose solutions to many of these challenges. Such properties include locally generated timing signals in the place of global clocks, potential performance speedups, robustness towards variability in the manufacturing process and operating conditions, etc. [7], [5], [6], [27], [28], [29], [32], [30], [36], [39]. However, design of asynchronous circuits has been a challenge and currently lacks support of Computer-Aided Design tools. Desynchronization [7], [10] is proposed as a design solution, where pipelined circuits and systems with a high degree of asynchronocity are synthesized from synchronous parents in a manner that exploits existing CAD tool support for synchronous designs. Desynchronization methods have in fact been successfully used to design and fabricate circuits that implement the DLX architecture and the DES encryption/decryption algorithm [7].

For desynchronization to be a feasible design solution, one of the critical challenges however is verification. Verification becomes a challenge when the desynchronized circuits are pipelined. For example, the controller of a desynchronized 5-stage pipeline can have more than $16*15^{10}$ states [7]. Thus, our approach is focused on verifying desynchronized pipelines against their non-pipelined synchronous specifications.

One of the more effective formal methods to verify pipelined circuits and systems is refinement-based verification. Refinement is a formal correctness notion that can be used to check the equivalence of an implementation system and a specification system, even if the implementation and specification are at very disparate levels of abstraction. In the context of pipelines, refinement is used to verify the pipelined system (such as a pipelined processor model) against a high-level non-pipelined specification (such as an instruction set architecture machine). A number of refinement-based verification solutions have been developed for synchronous pipelines [23][25] [26][16] [13] [12] [11] [15] [33] [34] [35] [19] [41] [18] [1] [2]. These verification solutions have been developed in the context of pipelined microprocessor models. While there have been some efforts toward the verification of asynchronous pipelines [20], this area has not been explored as much.

The desynchronized pipeline controller network is highly non-deterministic in nature and also exhibits a large state space. These two factors make it hard to track the states of the controller network that are reachable from the initial or reset states (reachable states). Identifying reachable states is important as unreachable states are often inconsistent and can cause spurious counter examples. More specifically, there are two approaches to compute reachable states of the desynchronized controller network.

1) The first approach is based on symbolic simulation. The idea is to start from the set of reset states and perform symbolic simulation until no new states are discovered, *i.e.*, until a fixed point is reached. Or, start from the set of all states and perform symbolic simulation until a fixed point is reached where no new states are eliminated [25]. Approaches based on symbolic simulation of the implementation model cannot be used because the complexity of the desynchronized controller network requires a prohibitively large number of symbolic simulations of the model.
2) The second approach is to compute invariant properties of the desynchronized controller network that characterize the set of reachable states. We explored this approach and found that because of the large state space and the high degree of nondeterminism of the controller networks, we could not find a systematic approach to generate invariants to characterize the reachable states of the controller networks. The primary problem is that the behavior of a desynchronized controller depends on the state of controllers on its output side, but, does not have any dependencies on the input side *i.e.*, the source controllers. This lack of dependency on the source side results in a high degree of nondeterminism of the resulting controller networks.

Since verifiability is an important consideration for design, we propose changes to the desynchronized controllers introduced by Cortadella et al. [7]. These changes add additional sequential dependencies between controller events so that the state space of each controller and resulting controller networks are simplified and reduced. We refer to the modified con-

Sudarshan K. Srinivasan and Raj S. Katti are with the Department of Electrical & Computer Engineering, North Dakota State University, Fargo, North Dakota 58105. Email: sudarshan.srinivasan@ndsu.edu and rajendra.katti@ndsu.edu. This research was funded in part by NSF grant CCF-1117164.

trollers as Design For Verification Desynchronization (DFVD) controllers. We also develop a refinement-based verification method for desynchronized pipelines. We show that when the DFVD controllers are used, the resulting desynchronized pipelines can be verified using our approach. The specific contributions of our work are:

1) The controller used for desynchronization can hold zero, one or two tokens. The controller is initialized with one token and can transition to states with zero tokens or two tokens. Our contribution is the DFVD controller that satisfies the following property. If the controller currently holds one token, then the controller will hold on to that token until a new token is accepted on its input. This is not a property satisfied by the original controller used for desynchronization. Therefore, when the DFVD controller is initialized with one token, it will always remain in states where it has one or two tokens. This property of the DFVD controller makes it possible to compute reachable states of pipeline controller networks, which is a requirement for refinement-based verification. However, the verifiability of the controller is achieved by trading with performance. We estimate that in the worst case, pipeline throughput is degraded by the delay of four transitions of a muller-C element.
2) We analyzed and deduced 15 properties of the DFVD controller. These properties (an important contribution of our work) can be used as rules and applied to systematically generate invariants and characterize the reachable states of any DFVD controller network.
3) We have also developed a refinement-based verification procedure for desynchronized pipelined systems. Proving refinement requires a refinement map, which (in this context) is a function that maps states of the implementation (desynchronized pipelined system) to states of the specification (non-pipelined synchronous machine). Defining the refinement map in this context requires identifying duplicate information in the pipeline (which is possible in desynchronized pipelines). Our specific contribution here is to identify conditions of the controller network state that correspond to duplicate information in the data path. These duplicate conditions are generic and can be applied to any DFVD controller network. The key here is that these conditions are applicable only for reachable states (which we have been able to characterize using the DFVD properties).
4) We developed 7 desynchronized processor models of varying pipeline length (between 5 and 7 stages) and controller complexity. The models used the DFVD controllers. Our design for verification and verification approaches are demonstrated by checking the correctness of these models.

The rest of the paper is organized as follows. Related work is given in Section II. Desynchronization and DFVD controllers are described in Section III. The properties of the DFVD controllers and reachability analysis of desynchronization controller networks are described in Section IV. The desynchronized processor models used for experiments are described in Section V. The refinement-based verification procedure is detailed in Section VI. Experimental results are given in Section VII and we conclude in Section VIII. The benchmarks and tools required to reproduce our results are available in [9].

II. Related Work

Current verification technology for asynchronous circuits can be classified as property checking approaches [3][43] or methods based on trace theory [29]. The trace theory approaches target the verification of gate-level asynchronous circuits. Our focus is on the verification of desynchronized pipelined circuits and systems. Verifying pipelines has been a challenge and warrants specialized techniques. Approaches based on property checking can be used for desynchronized pipelined circuits, but, are cumbersome because a large number of properties are required and also the properties themselves can be hard to write leading to erroneous specifications [29].

Loewenstein [20] verified some properties of a counter-flow pipeline using the HOL theorem prover. Counter-flow pipelines are asynchronous in nature with results flowing in the pipeline in a direction opposite to that of instruction flow. The desynchronized pipelines we verify do not use the counter-flow mechanism. Also, our correctness proofs are based on the use of decision procedures and are highly automated.

Cortadella et al. [7] have used flow equivalence (FE) to prove the correctness of their desynchronization method and FE is well suited for this purpose. However, they have not demonstrated verification based on FE. Why do we use refinement instead of FE? Refinement is a more general notion. For example, one requirement of FE is that the specification and implementation should have the same set of latches. This requirement is not satisfied when comparing pipelined systems with non-pipelined specifications. Also, if after desynchronization, optimizations such as retiming or pipelining is applied, then the design cannot be related back to its synchronous specification using FE.

In previous work, we have developed a refinement-based verification method for desynchronized pipelines [38]. The original desynchronization controllers are used. The approach for reachability is based on performing symbolic simulation of the implementation model, starting from reset states, until no new states are discovered, *i.e.*, until a fixed point is reached. However, this approach is not viable because the complexity of the desynchronized controller network requires a prohibitively large number of symbolic simulations of the model. As a result, we were only able to verify a small subset of the reachable states and therefore, the verification method is only partial. In the current work, our approach is to generate constraints (also known as inductive invariants) on the state variables that characterize the reachable states of the system. This approach is also not viable for the original desynchronized controllers. Hence we use the design for verification approach to develop the DFVD controller. For the DFVD controller network, the latter approach for reachability is viable as shown in Section IV.

We proposed the idea of using completion functions to define the refinement map for desynchronized pipelines in [38]. We adopt this approach in our current work. However, the key difference is how duplicate data is identified in the pipeline. In [38], we relied on observing the flow of tokens in the controller network, as symbolic simulation was used for reachability analysis. However, as stated earlier, using symbolic simulation for reachability analysis is not viable. In this work, we have deduced generic conditions of the state of controller network that identify duplication in the data path, which can be used for complete safety verification and is described in Section VI.

III. Desynchronization and DFVD Controllers

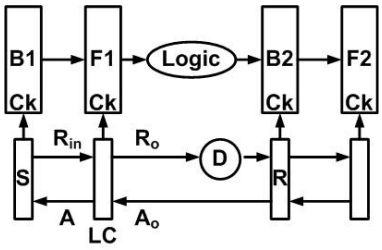

Fig. 1. A Pipeline Stage with A Latch Controller Network.

Desynchronization is the process of converting a synchronous circuit into an asynchronous one by replacing the clock network with a network of handshaking latch controllers. The edge-triggered D-flipflops of the synchronous circuit are replaced by two D-latches which are transparent when their clock input is a 1 and are in the hold mode otherwise. The clock signals or triggers (Ck) for the latches are obtained by latch controllers with two inputs (R_{in},A_o), and two outputs (R_o,$A = \neg Ck$). $R's$ denote a request signal and $A's$ denote an acknowledge signal. Consider a synchronous pipeline stage with a logic block whose inputs are provided by a flipflop and whose outputs are input to another flipflop. A desynchronized version of such a stage is shown in Figure 1. Each flipflop is converted into two latches shown on either side of the logic block. In a pair of consecutive latches, the left latch is the back latch and the right is the front latch, indicated by subscripts "b" (or "B") and "f" (or "F"), respectively. Each latch has a controller associated with it. The latch controller used by us is the semi-decoupled controller in [10]. If G is a signal then G^+ corresponds to a rising edge on G and G^- corresponds to a falling edge on G. We now describe the operation of the latch controller labeled, LC, in Fig 1 with the help of the two latch controllers , S (for sender), and R (for receiver). LC receives R_{in}^+ from S indicating the availability of data at the input of the LC latch (F1). LC sends A^+ to S indicating that the data has been captured by F1. LC then sends R_o^+ to R to indicate that its output is valid and will be stable until A_o^+ is received. S sees A^+ and puts out R_{in}^-. LC sees R_{in}^- and A_o^+ and puts out A^- and R_o^-. R puts out A_o^- when it receives R_o^-. The above description of the latch controller (called the 4-phase controller) can be converted to the following five logic equations.

1. $A^+ = R_{in} \wedge \neg R_o$
2. $A^- = \neg R_{in} \wedge R_o \wedge A_o$
3. $R_o^+ = A \wedge \neg A_o$
4. $R_o^- = \neg A$
5. $Ck = \neg A$

Note that the clock input to the latch is Ck and the delay element D in Fig. 1 mimics the delay of the logic block. This kind of desynchronization is similar to that performed in [7] and has been proven to work well.

One of the important aspects of desynchronization is that it leads to duplicated tokens in the data path. When a token is passed from a source latch to a destination latch, the source latch holds on to a copy of the token until it receives an acknowledge signal indicating that the token has reached its destination. Thus, for a period of time, the source and destination latches both have copies of the same token. Duplication leads to some issues with refinement-based verification, which we discuss in Section VI.

A. DFVD Controller

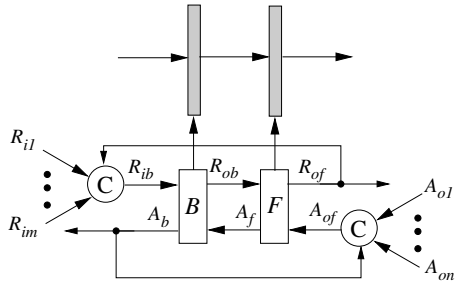

Fig. 2. DFVD Controller

The proposed Design For Verification Desynchronization (DFVD) controllers for a pipeline latch pair is shown in Figure 2. The difference between the proposed controller and the controller in the desynchronized circuit of [7] is the feedback of R_{of} to the muller-C element that generates R_{ib} and the feed forward of A_b to the muller-C element that generates A_{of}. The connections are not part of the original controller. The A_b connection enforces the property that if the controller currently holds only one token in the F latch, then the controller will hold on to that token until it has received a new token in the B latch. The F latch will drop its token when it receives an acknowledge (A_{of}^+). Since A_b is connected to the muller-C element that generates A_{of}, unless latch B acknowledges the receipt of a new token by asserting A_b, the F latch will not drop its token.

An additional dependency is enforced by the R_{of} connection that allows the B latch to receive a new token only when the F latch has signaled a request on the output side. The new controller results in only minor delays but satisfies properties that allow for reachability analysis (see Section IV).

We now estimate the *worst case* increase in delay for the new controller by estimating the maximum delay between consecutive R_{ib}^+ transitions in the controller of latch B. This

TABLE I
WORST CASE DELAY ANALYSIS OF DFVD CONTROLLER

State Label	State $\langle A_b R_{ib} R_{ob} A_f R_{of} A_{of} \rangle$	Event	Delay
S1	$\langle$ 000100 $\rangle$	R_{ob}^-	N
S2	$\langle$ 000110 $\rangle$	R_{of}^+	N
S3	$\langle$ 010110 $\rangle$	R_{ib}^+	Y
S4	$\langle$ 110110 $\rangle$	A_b^+	N
S5	$\langle$ 100110 $\rangle$	R_{ib}^-	Y
S6	$\langle$ 100111 $\rangle$	A_{of}^+	Y
S7	$\langle$ 100011 $\rangle$	A_f^-	N
S8	$\langle$ 101011 $\rangle$	R_{ob}^+	N
S9	$\langle$ 101001 $\rangle$	R_{of}^-	N
S10	$\langle$ 101101 $\rangle$	A_f^+	N
S11	$\langle$ 001101 $\rangle$	A_b^-	N
S12	$\langle$ 001100 $\rangle$	A_{of}^-	Y

delay gives us the minimum time between consecutive sets of data getting stored into a latch. In the new controller circuit R_{ib} cannot change to 1 (or 0) unless R_i's (R_{i1}–R_{im}) and R_{of} are all 1 (or 0). Similarly A_{of} cannot change to 1 (or 0) unless A_o's (A_{o1}–A_{on}) and A_b are all 1 (or 0). The set of transitions that lead to *worst case* delay for the proposed controller circuit is shown in Table I. This has been derived from the state diagram of an individual semi-decoupled 4-phase controller of [10] (see Figure 8 in [10]). The controller transitions from state S_i to S_{i+1}, starting at state S_1 and until it reaches S_{12}. From S_{12}, it transitions back to S_1. The events that causes the state transition is also shown in Table I. Delays occur when a transition of R_{ib} or A_{of} has to occur.

From the state diagram it is clear that it takes 12 state transitions for two consecutive R_{ib}^+ transitions to occur. However without the new connections to R_{ib} and A_{of} it takes 8 state transitions for two consecutive R_{ib}^+ transitions to occur. Thus we obtain a *worst case* delay of 4 state transitions for the new controller to have two consecutive R_{ib}^+ compared to the existing semi-decoupled 4-phase controller of [10], which is usually negligible compared to the delay of pipeline processing logic in a stage. Also, note that many of the transitions of the additional muller-C elements can take place simultaneously with other events in the circuit and on average the performance degradation could be much lower.

IV. REACHABILITY ANALYSIS OF DFVD CONTROLLER NETWORK

To perform verification, we need to compute the reachable states of the desynchronized pipeline controller network. Computing reachable states has two ends. First, unreachable states can be inconsistent w.r.t. the correctness property and flag spurious counter examples that hinder the verification process. Identifying reachable states of the implementation solves this problem as verification properties can now be checked only on the reachable states ensuring that spurious counter examples are eliminated. Second, our procedure for computing refinement maps for desynchronized pipelined machines is based on reachability analysis. ***Note that the reachability method eliminates unreachable states that hinder verification, which is what is required. The reachable states of the controller network may in fact only be a subset of the set of states computed by the reachability method.***

We now describe the general invariant generation rules. The first 8 rules (P1-P8) apply to the DFVD controller shown in Figure 2. Note that these rules are properties of the DFVD controller, and should be applied to each of the DFVD controllers in a DFVD pipeline controller network.

The acknowledge signal for the front and back latches A_f and A_b, respectively, also act as the clock for the front and back latches. When A_f or A_b are asserted, the corresponding latches are in a hold state, and when A_f or A_b are de-asserted, the corresponding latches are transparent (not holding any data tokens). Thus Property P_1 is a significant property as it implies that the DFVD controller will always be in a state where one or both of the latches is in a hold state. In other words, the DFVD controller will never reach a state where both latches are empty/transparent. This is not a property satisfied by the desynchronization controller proposed by [7], which allows the state where both latches are transparent. Property P_1 makes it possible for us to compute reachable states and define refinement maps in a systematic manner for desynchronized pipelines.

P_1: $A_b \vee A_f$

Property P_1 is not an invariant by itself, because there are states of the DFVD controller, which satisfy the property, but which can transition to states that do not satisfy P_1. Therefore, we need low-level properties P_2–P_8 that eliminate all such states.

P_2: $\langle A_b \wedge A_f \wedge R_{of} \rangle \rightarrow (\neg R_{ob})$

Properties P_3–P_5 identify the conditions under which the muller-C element corresponding to A_{of} should hold values of 0 and 1.

P_3: $\langle A_b \wedge A_f \wedge (\neg R_{of}) \rangle \rightarrow (\neg A_{of})$
P_4: $\langle (\neg A_b) \wedge A_f \wedge R_{of} \rangle \rightarrow (\neg A_{of})$
P_5: $\langle A_b \wedge (\neg A_f) \rangle \rightarrow A_{of}$

Properties P_6–P_8 identify the conditions under which the muller-C element corresponding to R_{ib} should hold values of 0 and 1.

P_6: $\langle A_b \wedge (\neg A_f) \wedge (\neg R_{of}) \rangle \rightarrow R_{ib}$
P_7: $\langle (\neg A_b) \wedge A_f \wedge (\neg R_{of}) \rangle \rightarrow (\neg R_{ib})$
P_8: $\langle A_b \wedge A_f \wedge R_{of} \rangle \rightarrow R_{ib}$

The conjunction of properties P_1–P_8 form an inductive invariant, which we have verified using the ACL2-SMT verification system [37] by proving that for every state of the DFVD controller that satisfies the conjunction of properties P_1–P_8, its successor also satisfies the conjunction of P_1–P_8.

Properties P_9–P_{15} apply to the circuit shown in Figure 3 that occurs in the desynchronized pipeline controller network when data is passing from one stage of the pipeline to another. In this situation, the front latch of the source stage is connected to the back latch of the destination stage. Hence the front controller of the source stage (labeled F_1 in the figure) is connected to

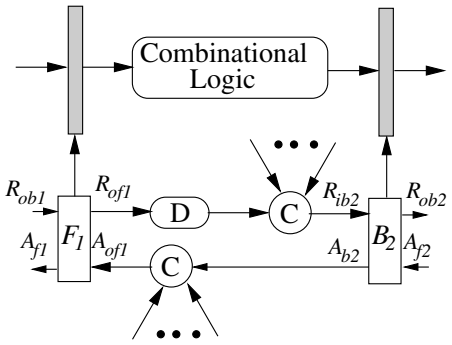

Fig. 3. DFVD Controller Circuit for Data Transfer

the back controller of the destination stage (labeled B_2 in the figure).

Properties/rules P_9–P_{15} should be applied to every source to destination connection in the pipeline including feedback connections as well. The properties P_9–P_{15} are used to eliminate inconsistent states by identifying the conditions in which the muller-C elements corresponding to R_{ib2} and A_{of1} hold values of 1 and 0.

P_9: $\langle(\neg A_{f1}) \wedge (\neg A_{b2})\rangle \rightarrow (\neg R_{ib2})$
P_{10}: $\langle(\neg A_{f1}) \wedge R_{of1}\rangle \rightarrow R_{ib2}$
P_{11}: $\langle A_{f1} \wedge (\neg A_{b2}) \wedge (\neg R_{of1})\rangle \rightarrow (\neg R_{ib2})$
P_{12}: $\langle A_{f1} \wedge A_{b2} \wedge R_{of1}\rangle \rightarrow R_{ib2}$
P_{13}: $\langle A_{f1} \wedge A_{b2} \wedge (\neg R_{of1})\rangle \rightarrow A_{of1}$
P_{14}: $\langle A_{f1} \wedge (\neg A_{b2}) \wedge R_{of1}\rangle \rightarrow (\neg A_{of1})$
P_{15}: $\langle(\neg A_{f1}) \wedge A_{b2}\rangle \rightarrow A_{of1}$

The conjunction of properties P_9–P_{15} also form an inductive invariant, which we have verified using the ACL2-SMT system by proving that for every state of the circuit shown in Figure 3 that satisfies the conjunction of properties P_9–P_{15}, its successor also satisfies the conjunction of P_9–P_{15}.

V. Desynchronized Pipelined Models

Five desynchronized pipelined processor models were developed and used as benchmarks to demonstrate the applicability and efficiency of the proposed verification solution for desynchronized systems. The models are specified using the ACL2 programming language [17]. First, a 5-stage synchronous pipelined processor model based on the DLX pipeline [31] was constructed. Three desynchronized versions of the synchronous pipeline were developed, including DPM5-1, DPM5-2, and DPM5-5. In DPM5-1, one desynchronization controller is used to control all the stages of the pipeline using the idea of clustering [8]. Clustering is also used in DPM5-2, where two desynchronization controllers are employed (one controller for the fetch and decode stages, and the second controller for the execute, memory, and write back stages). DPM5-5 is a fully desynchronized model, where 5 controllers are used (one for each stage of the pipeline). The fetch stage in DPM5-5 is further pipelined (resulting in a short instruction queue) to create DPM6-6 and DMP7-7, both of which are fully desynchronized models employing one controller for each pipeline stage. The high-level organization of DPM6-6 is shown in Figure 4.

The models are specified at the term-level [4], [40], an abstraction level in which the bit-vector data path is abstracted using integers (also called terms in this context). Also, functions that operate on data are abstracted using Uninterpreted Functions (black box functions that only satisfy the property that equal inputs produce equal outputs). Term-level abstraction is used as it drastically improves the efficiency of verification.

VI. Refinement-Based Verification

The goal of our verification procedure is to show equivalence between a pipelined desynchronized circuit/system and its non-pipelined synchronous specification. The notion of equivalence that we use is Well Founded Equivalence Bisimulation (WEB) refinement [22] and is based on stuttering bisimulation. Proving refinement guarantees that every behavior of the implementation is matched by behavior of the specification and vice versa. A detailed description of the theory of refinement can be found in [22]. It is enough to check the following correctness formula [21] to establish refinement (thereby establish equivalence) between an implementation and its specification.

Definition 1: (Core WEB Refinement Correctness Formula)

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

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 (a mechanism not found in stuttering bisimulation) is a function that maps implementation states to specification states thereby making it easy to compare systems at different abstraction levels. *rank*, used for deadlock detection, is a witness function from implementation states to natural numbers whose value decreases when there is stutter. The proof obligation that $s = r(v)$ is the safety component and guarantees that if the implementation makes progress, then the result of that progress is correct as given by the specification. The proof obligation that $rank(next\text{-}impl) < rank(impl)$ is the liveness component and guarantees that the machine will not deadlock, *i.e.*, will always make forward progress. In this work, we solve the problem of safety verification for desynchronized pipelines and reserve liveness verification for future work.

The specific steps involved in a refinement-based verification methodology for checking safety are: (a) Compute the states of the implementation model that are reachable from reset (known as reachable states). ***We use the rules given in Section IV to generate invariant properties that characterize the reachable states of any desynchronized pipeline controller network.*** (b) Construct a refinement map. (c) The models and the refinement map can now be used to state the safety component of the refinement-based correctness formula for the implementation model, which can then be automatically checked for the set of all reachable states using

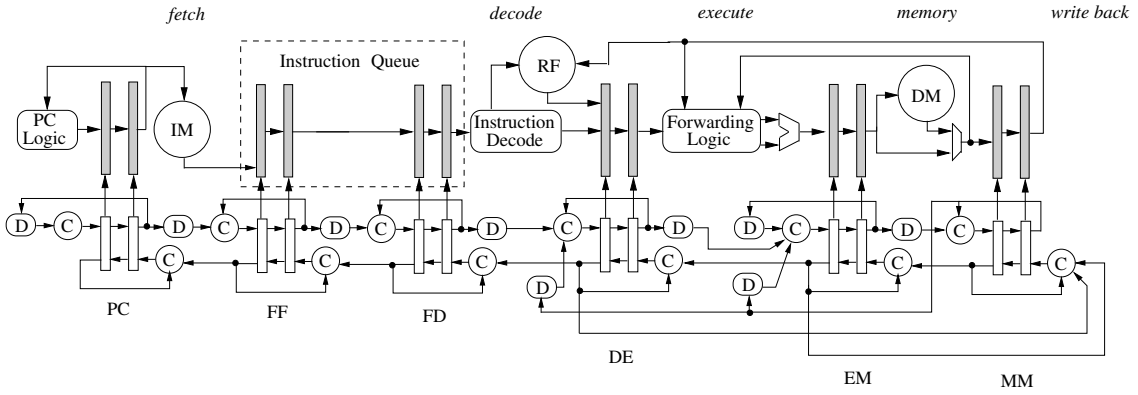

Fig. 4. High-Level Organization of Desynchronized 6-Stage Pipelined Processor Model

a decision procedure. Verification is performed using ACL2-SMT [37], a system developed by combining the ACL2 theorem prover [17] with the Yices Satisfiability Modulo Theories (SMT) solver [42].

Therefore, to perform verification, in addition to the implementation and specification models, and reachability analysis, we also require a refinement map. Next, we provide a procedure for computing refinement maps for desynchronized pipelined systems.

A. Refinement Maps

Our approach for defining refinement maps for desynchronized pipelines is based on the technique of completion functions proposed by Hosabettu et al. [14]. The idea with the completion functions approach is as follows. We use the DLX pipeline as an example, but the approach can be applied to any pipelined system. For a DLX pipeline, the non-pipelined specification is its instruction set architecture (ISA) machine. The state elements of this ISA machine includes its program counter, register file, data memory, and instruction memory. In addition to these state elements, a pipelined machine state also includes pipeline latches that have inflight instructions. The completion functions approach constructs a refinement map by completing the partially executed instructions in a pipelined machine state without fetching any new instructions. In the resulting, pipelined machine state, the pipeline latches are empty. Therefore, projecting out the ISA state elements from such a state would give the corresponding ISA state.

The partially executed instructions are completed by defining one function for each latch in the pipeline that observes the contents of that latch and computes how that instruction will update the ISA state elements. As instructions in the pipeline can depend on older instructions, the older instructions (instructions towards the end of the pipeline) are completed first. The values of the state elements obtained from completing older instructions are then used to complete younger instructions. Therefore, this approach allows younger instructions access to results of older instructions. Note that refinement maps based on symbolic simulation of the implementation model such as commitment [21] [23] or flushing [4] are not viable because a prohibitively large number of symblolic simulations are required for desynchronized pipelines.

The completion functions approach is efficient in terms of computational complexity and works well for synchronous pipelines. However, for desynchronized pipelines, the problem is that desynchronization allows for duplication of data in the data path. This is an issue with the completion functions approach. Completing the same instruction twice (from two different latches) will lead to erroneous results. Therefore, the duplicate instructions/data in the pipeline latches need to be identified and omitted from being completed.

Since duplication is a result of desynchronization, the latches that have duplicate data in a desynchronized pipelined machine state can be determined by observing the controller state. There are two conditions of the pipeline latch controller that identify duplicate data in the pipeline, which are given below.

D_1: $A_b \wedge R_{ob} \wedge A_f$

The first duplication condition (D_1) occurs between the latch pair used to separate two stages of a pipeline and is depicted in Figure 2. The condition occurs when the B latch is holding its data (indicated by A_b), which has also been transmitted to the F latch (indicated by $R_{ob} \wedge A_f$).

D_2: $A_{f1} \wedge R_{of1} \wedge R_{ib2} \wedge A_{b2}$

The second duplication condition (D_2) occurs between the F latch of a source latch pair (controller 1) and B latch of a destination latch pair (controller 2). The corresponding circuit is shown in Figure 3. The condition occurs when the F latch of controller 1 is holding its data (indicated by A_{f1}), which has also been transmitted to the B latch of controller 2 (indicated by $R_{of1} \wedge R_{ib2} \wedge A_{b2}$).

VII. Results

Table II reports the results for safety verification of the five desynchronized processor models described in Section V. One indicator of the complexity of the processor models is the number of lines of term-level ACL2 code required to specify the models, which is reported in the table. Note that the models are quite complex. For example, the size of the model DPM7-7 is 949 lines of term-level ACL2 code (obtained after abstracting combinational circuits blocks such as the ALU).

TABLE II
VERIFICATION TIMES AND SMT STATISTICS

Processor Model	No. Of Lines ACL2 Code	ACL2-SMT Verification Times (sec)	SMT Statistics			
			Decisions	Conflicts	Bool Vars	Memory Used (MB)
DPM5-1	687	1.19	6,292	1,202	2,723	9.41
DPM5-2	708	1.98	5,558	1,548	2,798	11.56
DPM5-5	783	4.37	70,662	16,950	3,456	13.09
DPM6-6	866	53.21	834,187	219,160	4,542	17.97
DPM7-7	949	2417.74	25,231,948	7,304,751	5,940	32.49
DPM-B1-5-2	708	1.91	5,665	1,058	2,940	11.62
DPM-B2-5-2	708	1.74	358	49	1,529	11.01

Table II reports the verification time for checking safety for the desynchronized processor models. The experiments were conducted using an Intel(R) Core(TM)2 CPU 6400, with a cache size of 2048 KB. Verification was performed using the ACL2-SMT system [37], obtained by combining ACL2 (version 3.3) and the Yices decision procedure (version 1.0.10).

The table also provides SMT statistics that are indicative of the complexity of the verification problem. Note that overall, verification is efficient as safety is verified for all the models with a maximum running time of 2417.74 seconds required by the DPM7-7 model. A well-known trend in verification of pipelined machines is the exponential increase in verification times with increase in the number of pipeline stages [24]. The results exhibit this trend as well. Compositional approaches have been successfully demonstrated to improve the scalability of refinement-based verification of synchronous pipelines [24]. For future work, we plan to explore compositional approaches for desynchronized pipelines.

Buggy Models: Two buggy variations of the DPM5-2 model were also checked using the verification procedure and resulted in the ACL2-SMT tool flagging counter examples that pointed to the source of the bug. The buggy models are DPM-B1-5-2 and DPM-B2-5-2. In DPM-B1-5-2, we injected a bug in the data path. In the forwarding path for source operand 2 from memory stage to execute stage, the destination operand address is compared with the source address of operand 1 instead of operand 2. In DPM-B2-5-2, we injected a bug in the desynchronized pipeline controller network. The DPM5-2 model has 2 DFVD controllers. The R_{ob} signal instead of the R_{of} signal of controller 1 is connected to the R_{ib} muller-C element of controller 2. The verification statistics are also reported for the buggy models in Table II.

VIII. CONCLUSIONS

Formal verification methods have become an integral part of the design cycle to ensure reliable IC designs. Therefore, verifiability has become an important consideration for any design paradigm. In this work, we propose improved verifiability for desynchronization, which is achieved with a worst case performance penalty of 4 muller-C element delay in pipeline throughput.

For future work, we plan to address liveness verification of desynchronized pipelines and explore compositional methods to improve scalability. We also plan to explore design for verification solutions for desynchronization with lower performance degradation.

REFERENCES

[1] T. Arons and A. Pnueli. Verifying tomasulo's algoithm by refinement. In *Proc. 12th International Conference on VLSI Design*, 1999.

[2] T. Arons and A. Pnueli. A comparison of two verification methods for speculative instruction execution. In *TACAS00: Tools and Algorithms for the Construction and Analysis of Systems*, pages 487–502, 2000.

[3] J. R. Burch. Combining ctl, trace theory and timing models. In J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*, pages 334–348. Springer, 1989.

[4] J. R. Burch and D. L. Dill. Automatic verification of pipelined microprocessor control. In *CAV '94*, pages 68–80.

[5] J. Cortadella. Petrify: A tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *IEICE Trans. Inf. Syst.*, E80-D(2):315–325, 1997.

[6] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. *Logic Synthesis of Asynchronous Controllers and Interfaces*. New York: Springer-Verlag, 2002.

[7] J. Cortadella, A. Kondratyev, L. Lavagno, and C. P. Sotiriou. Desynchronization: Synthesis of asynchronous circuits from synchronous specifications. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 25(10):1904–1921, 2006.

[8] A. Davare, K. Lwin, A. Kondratyev, and A. L. Sangiovanni-Vincentelli. The best of both worlds: the efficient asynchronous implementation of synchronous specifications. In S. Malik, L. Fix, and A. B. Kahng, editors, *(DAC'04)*, pages 588–591. ACM, 2004.

[9] Benchmarks and tools for: Desynchronization: Design for verification, 2011. See URL `http://-venus.ece.ndsu.nodak.edu/~s.srinivasan/-fmcad11.tar.gz`.

[10] S. B. Furber and P. Day. Four-phase micropipeline latch control circuits. *IEEE Trans. VLSI Syst.*, 4(2):247–253, 1996.

[11] R. Hosabettu, G. Gopalakrishnan, and M. Srivas. A proof of correctness of a processor implementing Tomasulo's algorithm without a reorder buffer. In L. Pierre and T. Kropf, editors, *Correct Hardware Design and Verification Methods, 10th IFIP WG10.5 Advanced Research Working Conference, (CHARME '99)*, volume 1703 of *LNCS*, pages 8–22. Springer-Verlag, 1999.

[12] R. Hosabettu, M. Srivas, and G. Gopalakrishnan. Decomposing the proof of correctness of a pileplined microprocessors. In *CAV '99*.

[13] R. Hosabettu, M. Srivas, and G. Gopalakrishnan. Proof of correctness of a processor with reorder buffer using the completion functions approach. In N. Halbwachs and D. Peled, editors, *Computer-Aided Verification–CAV '99*, volume 1633 of *LNCS*. Springer-Verlag, 1999.

[14] R. Hosabettu, M. K. Srivas, and G. Gopalakrishnan. Decomposing the proof of correctness of pipelined microprocessors. In A. J. Hu and M. Y. Vardi, editors, *Computer Aided Verification*, volume 1427 of *Lecture Notes in Computer Science*, pages 122–134. Springer, 1998.

[15] W. A. Hunt, Jr. Microprocessor design verification. *Journal of Automated Reasoning*, 5(4):429–460, 1989.

[16] R. Kane, P. Manolios, and S. K. Srinivasan. Monolithic verification of deep pipelines with collapsed flushing. In G. G. E. Gielen, editor, *Design, Automation and Test in Europe, (DATE'06)*, pages 1234–1239. European Design and Automation Association, Leuven, Belgium, 2006.

[17] M. Kaufmann, P. Manolios, and J. S. Moore. *Computer-Aided Reasoning: An Approach*. Kluwer Academic Publishers, July 2000.

[18] D. Kroning. *Formal Verification of Pipelined Microprocessors*. PhD thesis, Universität des Saarlandes, 2001.

[19] S. Lahiri, S. Seshia, and R. Bryant. Modeling and verification of out-of-order microprocessors using UCLID. In *Formal Methods in Computer-*

Aided Design (FMCAD'02), volume 2517 of *LNCS*, pages 142–159. Springer-Verlag, 2002.

[20] P. Loewenstein. Formal verification of counterflow pipeline architecture. In *Proceedings of the 8th International Workshop on Higher Order Logic Theorem Proving and Its Applications*, 1995.

[21] P. Manolios. Correctness of pipelined machines. In W. A. Hunt, Jr. and S. D. Johnson, editors, *FMCAD 2000*, volume 1954 of *LNCS*, pages 161–178. Springer-Verlag, 2000.

[22] P. Manolios. *Mechanical Verification of Reactive Systems*. PhD thesis, University of Texas at Austin, August 2001. See URL `http://-www.cc.gatech.edu/~manolios/publications.html`.

[23] P. Manolios and S. K. Srinivasan. Automatic verification of safety and liveness for xscale-like processor models using web refinements. In *Design, Automation and Test in Europe (DATE'04)*, pages 168–175. IEEE Computer Society, 2004.

[24] P. Manolios and S. K. Srinivasan. A complete compositional reasoning framework for the efficient verification of pipelined machines. In *ICCAD'05*, pages 863–870, 2005.

[25] P. Manolios and S. K. Srinivasan. A computationally efficient method based on commitment refinement maps for verifying pipelined machines. In *Formal Methods and Models for Co-Design (MEMOCODE'05)*, pages 188–197. IEEE, 2005.

[26] P. Manolios and S. K. Srinivasan. Refinement maps for efficient verification of processor models. In *Design, Automation and Test in Europe (DATE'05)*, pages 1304–1309. IEEE Computer Society, 2005.

[27] A. Martin and M. Nystrom. Asynchronous techniques for system-on-chip design. *Proc. IEEE*, 94(6):1089–1120, 2006.

[28] A. J. Martin, A. Lines, R. Manohar, M. Nyström, P. I. Pénzes, R. Southworth, and U. Cummings. The design of an asynchronous mips r3000 microprocessor. In *ARVLSI*, pages 164–181. IEEE Computer Society, 1997.

[29] C. J. Myers. *Asynchronous Circuit Design*. New York: Wiley, 2001.

[30] S. M. Nowick, M. B. Josephs, and C. H. V. Berkel. Special issue: Asynchronous circuits and systems. *Proc. IEEE*, 87(2):217–396, 1999.

[31] D. A. Patterson and J. L. Hennessy. *Computer Organization and Design: The Hardware/Software Interface*. Elsevier Morgan Kaufmann, 2009.

[32] J. Rabaey, A. Chandrakasan, and B. Nikolic. *Digital Integrated Circuits: A Design Perspective*. Prentice-Hall, 2003.

[33] J. Sawada. *Formal Verification of an Advanced Pipelined Machine*. PhD thesis, University of Texas at Austin, Dec. 1999. See URL `http://-www.cs.utexas.edu/users/sawada/dissertation/`.

[34] J. Sawada and W. A. Hunt, Jr. Trace table based approach for pipelined microprocessor verification. In *Computer Aided Verification (CAV '97)*, volume 1254 of *LNCS*, pages 364–375. Springer-Verlag, 1997.

[35] J. Sawada and W. A. Hunt, Jr. Processor verification with precise exceptions and speculative execution. In A. J. Hu and M. Y. Vardi, editors, *Computer Aided Verification (CAV '98)*, volume 1427 of *LNCS*, pages 135–146. Springer-Verlag, 1998.

[36] J. Sparso and E. S. Furber. *Principles of Asynchronous Circuit Design: A Systems Perspective*. Boston, MA: Kluwer, 2001.

[37] S. K. Srinivasan. *Efficient Verification of Bit-Level Pipelined Machines Using Refinement*. PhD thesis, Georgia Institute of Technology, December 2007. See URL `http://etd.gatech.edu/theses/-available/etd-08242007-111625/`.

[38] S. K. Srinivasan and R. S. Katti. Verification of desynchronized circuits. In *ISCAS'09*, 2009.

[39] C. H. van Berkel and R. Saejis. Compilation of communicating processes into delay insensitive circuits. In *Proc. Int. Conf. Computer Design (ICCD)*, pages 157–162. IEEE Computer Society, 1988.

[40] M. N. Velev and R. E. Bryant. Bit-level abstraction in the verfication of pipelined microprocessors by correspondence checking. In *FMCAD'98*, pages 18–35, 1998.

[41] M. N. Velev and R. E. Bryant. Superscalar processor verification using efficient reductions of the logic of equality with uninterpreted functions to propositional logic. In L. Pierre and T. Kropf, editors, *Correct Hardware Design and Verification Methods, 10th IFIP WG10.5 Advanced Research Working Conference, (CHARME '99)*, volume 1703 of *LNCS*, pages 37–53. Springer-Verlag, 1999.

[42] Yices homepage, 2007. See URL `http://fm.csl.sri.com/-yices`.

[43] T. Yoneda and B.-H. Schlingloff. Efficient verification of parallel real-time systems. *Formal Methods in System Design*, 11(2):187–215, 1997.