Inverting Martin Synthesis for Verification

Stephen Longfield, Jr., Rajit Manohar
School of Electrical and Computer Engineering
Cornell University, Ithaca, NY, 14853, U.S.A.
Email: {slongfield, rajit}@csl.cornell.edu

***Abstract*—Quasi Delay-Insensitive (QDI) circuits can be created through the procedure of Martin Synthesis, a series of transformations that begin with an executable specification and end in a transistor network. If these transformations are properly applied the circuits will be correct by construction; however if they are improperly applied, finding design errors can be quite difficult. We show that the forward transformations of Martin Synthesis are reversible, and that the inversion of these steps recreates the specification when applied to correctly synthesized circuits. We have created a tool to apply these inversions, and show that it can also be used to verify other compilation methods for QDI circuits. This procedure presents an alternative approach to typical VLSI verification by requiring little designer effort and by reconstructing specifications through transformations.**

I. Introduction and Motivation

Verifying that VLSI circuits will operate correctly is an important area of research, due both to the very high costs of failure, and the high level of complexity. However, many of the best methods to verify circuits require a great deal of designer effort to set up the verification [1], and any errors in this specification can be very difficult to diagnose and debug.

Martin Synthesis for Quasi Delay-Insensitive (QDI) asynchronous VLSI circuits begins with a description of computation in the Communicating Hardware Processes (CHP) language, a derivative of Hoare's Communicating Sequential Processes (CSP) [2]. This description is transformed into several smaller CHP programs whose actions are synchronized by actions on communication channels. These channels are replaced with appropriate handshakes, which may be reshuffled before being transformed into circuits [3]. When properly applied, this procedure generates correct asynchronous circuits.

For circuits generated through Martin Synthesis, the specification is the original high-level description, which is itself simulatable. Currently, the majority of QDI circuits are verified by referencing their final simulated behavior against simulations of this original specification This is not sufficient for very large designs which may have unknown rare cases that are not exercised by simulations. A formal verification system for asynchronous circuits has long been desired [4], but most of the effort has gone to verifying that the synthesis steps can be correctly applied forwards [5].

In the field of computer science, there has been considerable work in the reverse engineering and decompilation of programs, principally for the purposes of understanding how they operate, either for modification of legacy programs or for understanding unknown programs [6]. These decompilation procedures work best when the program was produced through known program transformations.

We propose a step toward reverse-engineering QDI logic by developing techniques to invert Martin Synthesis for control circuits. This process can be used for a number of different applications, including (i) verifying that asynchronous circuits correspond to their higher-level specifications, especially in the context of manual design; (ii) evaluating the impact of circuit optimizations on high-level design; (iii) reverse-engineering circuits. We have built a tool to automatically apply the proposed inverse transformations, and have used it to reverse-engineer a number of different QDI circuits from the literature.

II. Martin Synthesis for Control

We summarize the stages of the Martin Synthesis approach to QDI circuit design. The procedure starts with circuits described using the CHP notation (see Appendix A). To create efficient circuits, backtracking between stages may be required, but for any circuit created by Martin Synthesis there exists a strictly forward path. In the description that follows, we have introduced two additional steps in the synthesis procedure: (i) the introduction of *Two-phase CHP* [7], and (ii) the introduction of the partitioned repetitive event-rule system (PRER), a derivation of structures used for timing analysis [8]. These steps make aspects of the synthesis procedure more explicit without changing the procedure itself. They also introduce certain limitations which restrict our description to the design of control circuits.[1]

A. CHP ⇒ Two-phase CHP

Most QDI systems implement channel actions with four-phase handshakes as these have been found to allow for effective and efficient circuits, especially when handshakes are allowed to overlap [9]. Two-phase CHP [7] was created to analyze these overlaps.

The translation from CHP to Two-phase CHP is straightforward. Each CHP channel action, C, is replaced with two Two-phase CHP channel actions $C^{\uparrow}; C^{\downarrow}$, known as the up-going and down-going channel action, respectively. For data-carrying channel actions, the up-going channel action is defined as the action which sets a delay-insensitive encoding of data to a valid state, and the down-going action is defined as the one that resets it to a neutral state. These communication actions may be passive (beginning with a wait) or active

[1] This is not a limitation of Martin synthesis, but rather a limitation we have introduced to simplify the inverse synthesis problem in this paper.

(beginning with a state change). Martin Synthesis permits delaying the down-going action of the handshake, though it cannot be delayed such that it passes the next up-going action on this channel [3]. This procedure is known as reshuffling, and incorrect reshuffling can lead to deadlock.

One difference between the procedure we present here and Martin Synthesis without the transformation to Two-phase CHP is the use of the $\star$ operator instead of the $\bullet$ operator for simultaneous composition of channel actions. The $\bullet$ operator does not have a well-defined Two-phase CHP definition, whereas $\star$ is defined as:

$$C \star D \quad \triangleright \quad C^{\uparrow} \star D^{\uparrow}; C^{\downarrow} \star D^{\downarrow}$$

If channel actions are composed together with $\star$ at most one of the channels can be active [7].

Example. The D-element is a commonly used component in QDI control circuits. This circuit corresponds to the CHP program and Two-phase CHP program shown below:

$$*[L; R] \quad \triangleright \quad *[L^{\uparrow}; L^{\downarrow} \star (R^{\uparrow}; R^{\downarrow})]$$

B. Two-phase CHP ⇒ HSE

The next step in Martin synthesis is the creation of handshaking expansions (HSE). HSE is a strict subset of CHP, where all variables are Booleans. This implies that all channel actions have been expanded into sets and waits on Boolean variables that implement the channel. When translating to HSE, all computations on non-Boolean variables are first translated into Boolean actions. This is done using standard data representation techniques [10]. After this translation has been made, delay-insensitive data encodings and passive/active assignments are determined for each channel, and each channel is translated into its respective representations. For dataless channels, with output c_o and input c_i, this expansion takes one of the following forms:

$$\begin{array}{lll} C^{\uparrow} & \triangleright & c_o\uparrow; [c_i] \text{ (active)} \\ C^{\uparrow} & \triangleright & [c_i]; c_o\uparrow \text{ (passive)} \\ C^{\downarrow} & \triangleright & c_o\downarrow; [\neg c_i] \text{ (active)} \\ C^{\downarrow} & \triangleright & [\neg c_i]; c_o\downarrow \text{ (passive)} \end{array}$$

Two further *invisible* transformations are allowed at this level: (i) Fresh, local variables can be added to the computation, but they cannot affect computation or be accessed by other processes. This is equivalent to the state variable insertion of Martin Synthesis [3]. (ii) Externally invisible ordering changes can be made. These include transformations that do not change the sequence of operations (such as loop peeling and unrolling [11]), as well as transformations which reorder sequences of assignments and sequences of waits on Boolean variables. Reordering a sequence of assignments or a sequence of waits is permissible because a QDI circuit must observe variables through wires with arbitrary delay, and thus would be unable to distinguish such changes.

These two transformations are used in the state assignment procedure of Martin Synthesis. In this procedure, the HSE is modified into one where no assignment to variables can be reached in two distinct program positions.

Example. The HSE corresponding to the Two-phase CHP for the D-element is given below. Note that it includes a state variable z introduced to enable production rule synthesis [3].

$$*[[l_i]; z\uparrow; l_o\uparrow; [\neg l_i]; r_o\uparrow; [r_i]; z\downarrow; r_o\downarrow; [\neg r_i]; l_o\downarrow]$$

C. HSE ⇒ PRER

Translation from HSE to a partitioned repetitive event-rule system (PRER) is only defined for non-terminating HSE (those that include a single infinite loop) or HSEs without any loops. This does not limit the class of circuits we can design, as programs can be re-written to satisfy this constraint [12]. The definition is also limited to HSE without vacuous sets and waits, which are defined as ones whose removal will not affect the reachable states or delay a computation. This also does not add any effective limits, and is a reasonable restriction for control circuits.

Partitioned Repetitive Event Rule (PRER) systems are a novel structure introduced here. They are a derivative of Repetitive Event Rule (RER) systems, which were created to analyze performance of asynchronous control circuits [8]. A PRER has two event types, *wait* type events, which wait for a variable to be equal to a Boolean value, and *set* type events, which set a variable to a Boolean value. These events are connected with *rules* that represent causality. We forbid any PRER from having both *wait* and *set* events on the same variable. This constraint is what makes these systems partitioned—a variable is either one that is set by the PRER, or one that is set by another and which may predicate events in the PRER. When events are represented as nodes and the rules are represented as directed edges, the PRER can be viewed as a directed graph.

To transform from HSE to PRER, we first create events for all set actions (set events), and wait events for all waits. As PRER do not have a representation for selection statements or logic, all selections statements and logical operations must be transformed into series or parallel waits and sets. For logical operations, `and` corresponds to parallel execution of waits while `or` operations where the arguments are exclusive result in waits on the variables that will be true. This limits the set of possible HSE that can be represented with a PRER, which is why this paper focuses on control circuits.

Sequencing constructs from the HSE are used to create edges. Sequencing between two actions creates direct edges, which distributes over parallel composition. The infinite loop construct, $*[P]$, creates an edge from the final event back to the first event which is labelled as a back edge.

These transformations will not construct a valid PRER if the HSE contained waits on variables that were also set by the computation. Fortunately, these waits are either superfluous or lead to deadlock in the case of most control circuits. If the variable is never set to a value that would satisfy the wait, the wait and all of its successors can be removed.

We allow two transformations to be made on the PRER, which are equivalent to buffer and inverter additions from Martin Synthesis [3]. Adding a buffer or inverter to a variable

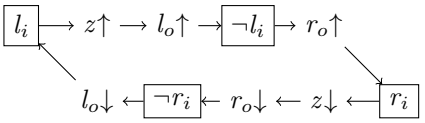


Fig. 1. D-Element PRER

is equivalent to adding a successor *set* event on a fresh variable to all events on the original variable, and transferring all of theses nodes' successors to the fresh nodes.

An isochronic branch of a forked wire is a branch where all transitions will be acknowledged by another transition. Wiring branches don't have a clear parallel in PRER systems, but there is a clear parallel to transitions. Adding a buffer or inverter on an isochronic branch is identical to adding a set event on a fresh variable as a successor to all events (either waits or sets) on a single variable, and then moving one or more of the outgoing edges of the original nodes onto these new nodes. If the new edges are not implied by transitivity, then it must be the case that all sets on the fresh variable will be acknowledged by sets on this successor.

Example. Figure 1 shows the PRER system that corresponds to the D-element handshaking expansion. In this diagram, *wait* events are boxed, while *set* events are not. The initial node is the one shown on the top left corner.

D. PRER ⇒ PRS

A Production Rule Set (PRS) that implements a PRER is the PRS that has the same requirements for ordering and causality as the PRER. The PRS that corresponds to a PRER need not be unique. The synthesis procedure from PRER to PRS can use the same technique as conventional Martin synthesis [3], since the PRER can be viewed as a graphical representation of the HSE.

Example. The production rules for the D-element are:

$$z \vee r_i \;\mapsto\; l_o\uparrow \qquad z \wedge \neg l_i \;\mapsto\; r_o\uparrow$$
$$\neg z \wedge \neg r_i \;\mapsto\; l_o\downarrow \qquad \neg z \vee l_i \;\mapsto\; r_o\downarrow$$

$$l_i \;\mapsto\; z\uparrow$$
$$r_i \;\mapsto\; z\downarrow$$

III. Inverse Synthesis

This section outlines the proposed methods for inverting all the steps in the synthesis of QDI circuits. We use the same intermediate representations as the synthesis procedure outlined in Section II. We also discuss limitations of the proposed inverse synthesis steps.

A. PRS ⇒ PRER

As PRER are similar to standard repetitive event-rule systems (RER) [8], algorithms to construct RER can also construct PRER systems. An example of such an algorithm is Lee's Index-Priority Simulation algorithm [13]. These algorithms need to be augmented to label events as being either sets or waits, but can otherwise be implemented as specified. This labeling must be specified by the designer for the inversion of this rule. For systems without shared variables, the wait event variables will be exactly the channel input variables, making this a straightforward step. As Index-Priority Simulation is only concerned with finding cycles, the algorithm was also augmented for finding the straight-line predecessors to cycles (pseudorepetitive PRER), and identifying the entry points to cycles. This was done by ensuring that the PRER initial state in matched the circuit reset state, and that there were as few PRER nodes possible between this state and the cycle entry. In the case where the reset state was in the cycle, edges connecting to it were labeled as back edges. We automatically reconstructed the D-element PRER from its production rule set using augmented index-priority simulation.

Limitations. All QDI PRS without inherent disjunctions can be transformed back into a PRER [13] though this procedure is computationally complex. This exposes another limitation of the approach outlined in this paper—namely, that we apply the inverse synthesis process to individual control processes rather than the full system. This is similar to an implicit limitation in Martin synthesis where handshaking expansions are converted into PRS one process at a time, thereby avoiding the construction of the state space for the entire system.

B. PRER ⇒ HSE

To transform from a PRER to an HSE, we first transform all of the PRER events into HSE events, wait events mapping to HSE waits, and set events mapping to HSE sets. If the PRER is pseudorepetitive, we consider the two components independently. These are combined using the following three rules:

- If two wait nodes have the same set of successors, these two nodes are fused together, and become a single node with their elements combined with `and`.
- If a HSE has only a single predecessor, and that predecessor has only a single successor, the two HSE nodes can be fused together, sequencing the predecessor before the successor.
- If two HSE nodes have the same set of successors and predecessors, these two HSE nodes are fused together with parallel composition.

This is sufficient to collapse all properly nested PRER into a single HSE node. If this node has no successors or predecessors, it is a straight line HSE. If it is its own successor and predecessor, it is an infinitely looping HSE. Note that if the PRER is cyclic, reverse edges must be either ignored or transformed with the lowest priority. If they are not, the HSE may be reconstructed incorrectly, as these are the only indicator for where a loop begins.

Repeatedly applying these transformations to the PRER in Figure 1 re-creates the handshaking expansion for the D-element.

Limitations. If the PRER is not properly nested [7], the transformations given will not be able to derive a HSE. As the HSE only has parallel and sequential composition operators, the original embedding must be properly nested. However,

the PRER system could correspond to circuits created by other procedures that do not use HSE. Finally, the limitations outlined in Section IIC also apply.

C. HSE ⇒ Two-phase CHP

To reverse the translation of channel actions from Two-phase CHP, each channel must be labeled as active or passive, and then match the projection onto each pair of channel action variable pairs against Table I, duplicated from [7]. This gives a partial ordering on Two-phase channel actions, which can be used to reconstruct the Two-phase CHP. We describe a few extensions to this process below.

First, if a channel is communicated on more than once in a program, each communication and and its associated channel variables can be given a fresh names for the sake of the transformation. Second, if the variables do not fit into the orderings shown in Table I, they can be reordered under certain circumstances. Parallel composition of actions on channel variables can be mapped to any of the possible interleavings. This parallel composition should be reconstructed to least restrictive Two-phase CHP allowed, where the Two-phase compositions are ordered as:

$$L^{\uparrow} \parallel R^{\uparrow} \;<\; L^{\uparrow}; R^{\uparrow} \;=\; R^{\uparrow}; L^{\uparrow} \;<\; L^{\uparrow} \star R^{\uparrow}$$

Finally, any *invisible* transformations, as defined in the Two-phase CHP to HSE section can be made. The transformation that was found to be the most common was loop peeling, where a reconstructed HSE had some channel variable set in its reset state, and subsequent occurrences of this set were scheduled in the body of the loop.

These transformations must be made such that when reconstructing the Two-phase CHP, the ordering on channel up and down going actions is $C^{\uparrow} \prec C^{\downarrow}$ for any channel C. If this is not true, it is not possible to translate back into a CHP program. This ordering must exist for any properly constructed HSE, but may not appear in a reconstructed HSE if the reentry point of the infinite loop was incorrectly determined. In these cases, a loop peeling that gives the proper channel ordering must exist if the decompilation is to continue.

Limitations. The methods presented here will not reconstruct any of the translations of data operations onto Boolean variables. As there are many possible origins for a Boolean operation, the specification cannot be reconstructed through purely uninformed inversions. Additionally, as was stated in the original paper on Two-phase CHP, to recreate the Two-phase CHP, the HSE must be properly nested, however, any HSE that was generated from the synthesis procedure will be properly nested [7]. It must also be the case that the channel actions in the HSE contain a correct ordering for all channels.

D. Two-phase CHP ⇒ CHP

In the majority of cases transforming from Two-phase CHP to CHP is done by replacing the up-going channel action with the full channel action, as its position in the computation must not have been modified.

TABLE II
EXAMPLE STAR COMPOSITION TO PROBES. IN THIS TABLE, × IS BEING USED TO RANGE OVER ALL NON-⋆ COMPOSITION OPERATORS (E.G. ;)

C Passive, D Active	$C^{\uparrow} \star D^{\uparrow}; C^{\downarrow} \times D^{\downarrow}$ $\triangleright[\overline{C}]; D; C$
C and D passive, E active	$C^{\uparrow} \star D^{\uparrow} \star E^{\uparrow}; C^{\downarrow} \times D^{\downarrow} \times E^{\downarrow}$ $\triangleright[\overline{C} \wedge \overline{D}]; E; (C \parallel D)$
C passive, D and E active	$C^{\uparrow} \star (D^{\uparrow} \parallel E^{\uparrow}); C^{\downarrow} \times D^{\downarrow} \times E^{\downarrow}$ $\triangleright[\overline{C}]; (D \parallel E); C$
C passive, D and E active	$C^{\uparrow} \star (D^{\uparrow}; E^{\uparrow}); C^{\downarrow} \times D^{\downarrow} \times E^{\downarrow}$ $\triangleright[\overline{C}]; (D; E); C$

The one exceptional case is when two up-going channel actions are composed with ⋆. If the down-going channel actions are also composed with ⋆, then the full channel actions can be consider to occur at the same program point as the up-going actions, composed with ⋆. If the down-going actions are not composed with ⋆, then the translation used is the following steps in sequence: (i) wait for all the probes on the passive channels to be true; (ii) execute the operations on the active channels (composed as in the original up-going actions); (iii) complete the communication on the passive channels in parallel. Examples are shown in Table II.

Limitations. The procedure defined here does not preserve deadlocking behavior (since the second half of the synchronization is erased). Any deadlock would have to be identified at the Two-phase CHP level of abstraction, through existing techniques [7].

IV. APPLICATIONS

We have created a tool which automatically applies the reverse steps detailed in the previous section, and have successfully applied it to the control circuits that are used in two compilation procedures in the literature: (i) Syntax Directed Translation [14], and (ii) the Tangram synthesis approach [15]. These two systems were chosen as the primitives are within the limits to the analysis proposed, their synthesis was not conducted by the authors, and their reverse engineerings are simple enough to present here. Beyond these examples, we have successfully applied the decompilation tool to dataless control buffers of the PCHB, PCFB, and WCHB families [9], as well as larger straight-line sequencing with several hundred possible reachable state assignments.

A. Syntax Directed Translation

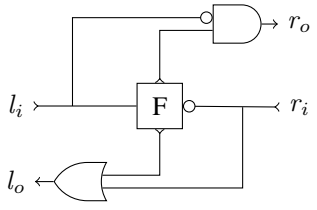


Fig. 2. The D-Element

In the Syntax Directed Translation process presented in [14] sub-processes are decomposed automatically into smaller pro-

TABLE I
HSE AND MAPPINGS TO RESHUFFLED TWO-PHASE COMMUNICATIONS

L Passive, R Passive			L Active, R Active			L Active, R Passive		
$[l_i];\ l_o\uparrow;[r_i];\ r_o\uparrow$	$\triangleright$	$L^\uparrow;\ R^\uparrow$	$l_o\uparrow;\ [l_i];\ r_o\uparrow;\ [r_i]$	$\triangleright$	$L^\uparrow;\ R^\uparrow$	$l_o\uparrow;\ [l_i \wedge r_i];\ r_o\uparrow$	$\triangleright$	$L^\uparrow;\ R^\uparrow$
$[l_i \wedge r_i];\ l_o\uparrow \parallel r_o\uparrow$	$\triangleright$	$L^\uparrow \star R^\uparrow$	$l_o\uparrow \parallel r_o\uparrow;\ [l_i \wedge r_i]$	$\triangleright$	$L^\uparrow \parallel R^\uparrow$	$l_o\uparrow;\ [r_i];\ r_o\uparrow;\ [l_i]$	$\triangleright$	$L^\uparrow \parallel R^\uparrow$
$[r_i];\ r_o\uparrow;[l_i];\ l_o\uparrow$	$\triangleright$	$R^\uparrow;\ L^\uparrow$	$r_o\uparrow;\ [r_i];\ l_o\uparrow;\ [l_i]$	$\triangleright$	$R^\uparrow;\ L^\uparrow$	$[r_i];\ l_o\uparrow;\ [l_i];\ r_o\uparrow$	$\triangleright$	$L^\uparrow \star R^\uparrow$
						$[r_i];\ r_o\uparrow \parallel l_o\uparrow;[l_i]$	$\triangleright$	$R^\uparrow;\ L^\uparrow$

In this table, an up-going action is being used as shorthand for up and down going actions. The same transforms apply. As $\parallel$ and $\star$ commute, L Passive, R Active is identical to L Active, R Passive with renaming.

cesses, and then re-connected with a D-element (see Figure 2). For sequential composition, this corresponds to the transformation:

$$\begin{array}{lll} *[S_0; S_1] & \triangleright & *[\bar{C_0} \rightarrow S_0; C_0?] \\ & \parallel & *[\bar{C_1} \rightarrow S_1; C_1?] \\ & \parallel & D(C_0, C_1) \end{array}$$

where the D-element is shown in Figure 2. The F-element in Figure 2 is simply:

$$\begin{array}{l} l_i \ \mapsto \ z\uparrow \\ r_i \ \mapsto \ z\downarrow \end{array}$$

This allows us to translate the D-element circuit into a PRS. In this PRS, the pair of wires on the left is addressed as l_i and l_o, and the ones on the right as r_i and r_o, making up the channels L and R, respectively.

We used this D-element as the working example when explaining Martin Synthesis, as well as the proposed inverse synthesis approach. When augmented with an environment and a reset on z, our tool can create the PRER system shown in Figure 1. This in turn was automatically inverted into the correct handshaking expansion, which in turn was converted into the correct Two-phase CHP. Finally, the Two-phase CHP

$$*[L^\uparrow; L^\downarrow \star (R^\uparrow;\ R^\downarrow)]$$

can be correctly converted into:

$$*[L; R]$$

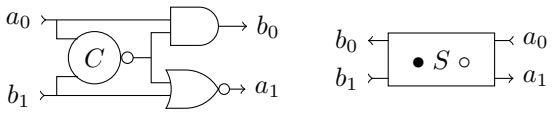
As these are dataless channels, they can be freely assigned to sends by labeling the outputs as data, creating the CHP:

$$*[L!; R!]$$

which matches the original CHP used for the D-element.

B. Tangram Control Circuits

The Tangram compilation method has been successfully used to create several asynchronous chips by mapping from a CSP-derived source language down to circuit primitives [15]. The validity of this mapping was argued by giving specifications for the primitives and showing that, if these specifications are followed, the final decomposition will be a valid implementation of the original program [16]. In this section, we use the decompilation tools and techniques we have developed to show that an interesting selection of the Tangram primitives implement their specifications. For the sake of compactness, we do not present the PRER for the Tangram primitives.



a) Circuit Diagram b) Schematic Icon

Fig. 3. Tangram S Element

TABLE III
MAPPING FROM TANGRAM SPECIFICATION TO TWO-PHASE CHP

Name	Tangram Specification	Two-phase CHP
Up-going Action*	$a\uparrow^\bullet \mid b\uparrow^\circ$	$A^\uparrow! \mid B^\uparrow?$
Down-going Action*	$a\downarrow^\bullet \mid b\downarrow^\circ$	$A^\downarrow! \mid B^\downarrow?$
Sequential Composition	$T; U$	$P; Q$
Parallel Composition	—	$P \parallel Q$
Simultaneous Composition	$a : b$	$A \star B$
Infinite Loops	$\#[T]$	$*[P]$

*Tangram specifications indicate if a channel is active or passive. Here, that is indicated in Two-phase CHP by labeling as a send or receive, respectively. This mapping is arbitrary.

The primitives are specified in terms of circuit diagrams, Tangram diagrams and a CSP-like specification. The circuit diagrams are directly translatable into PRS and the Tangram diagrams give indications for which channels should be considered passive (indicated with a bubble) and active (indicated with a filled-in circle). The written specifications have direct translations into Two-phase CHP, detailed in Table III. All of these circuits are specified to be *strongly initializing* meaning they need no additional circuitry to reset to the correct state given that their inputs were reset to the correct state.

1) S-Element: The S-Element is a circuit element that is used to build up other Tangram primitives. The circuit diagram and the symbol that will be used to represent it are shown in Figure 3. It is described as being similar to the D-element [14], and with Two-phase specification:

$$*[A^\uparrow? \star (B^\uparrow!; B^\downarrow!); A^\downarrow?]$$

When we pass the PRS for the S-element circuit through our tools, the HSE derived (where z indicates the output of the C-element, and with variables renamed to indicate input and

output) is:

$$*[[a_i]; b_o\uparrow; [b_i]; z\downarrow; b_o\downarrow; [\neg b_i]; a_o\uparrow; [\neg a_i]; z\uparrow; a_o\downarrow]$$

This HSE can be passed through the next phase of decompilation, creating the Two-phase CHP:

$$*[A^{\uparrow}? \star (B^{\uparrow}!; B^{\downarrow}!); A^{\downarrow}?]$$

which is identical to the Two-phase CHP specified by the Tangram flow. Using this description, we can see that it is subtly different than the D-element. The S-element requires the entire active handshake to be enclosed in the up-going action of the passive handshake, while the D-element requires the active handshake be enclosed by the down-going action. These are symmetric, but represent distinct CHP. The S-element CHP follows:

$$*[[\overline{A}]; B!; A?]$$

2) Repeater: The repeater is a source for active channel b, activated by passive channel a. The translation of its specification into Two-phase CHP follows:

$$A^{\uparrow}? \star (*[B^{\uparrow}!;\ \ B^{\downarrow}!])$$

Given an all-low reset, the circuit shown in Figure 4 can be inverted to result in the HSE:

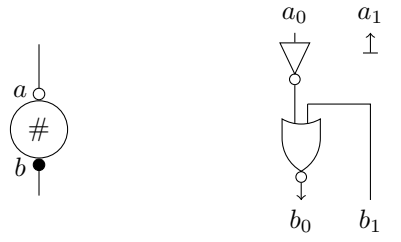
$$[a_0];\ \ *[b_0\uparrow;\ \ [b_1];\ \ b_0\downarrow;\ \ [\neg b_1]]]$$

This HSE cannot be decompiled further by the procedures presented above into pure channels. This is because the Two-phase action on A never completes. One way to proceed is to simply convert the wait into a probe on the channel. With this approach, we can obtain the Two-phase CHP and CHP shown below:

$$\begin{aligned} \text{Two-phase} &\equiv [\overline{A}]; *[B^{\uparrow}!; B^{\downarrow}!] \\ \text{CHP} &\equiv [\overline{A}]; *[B!] \end{aligned}$$

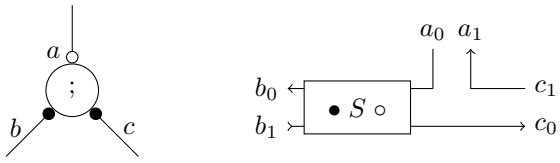
These are correctly infinite sources on B that are predicated on A. However, they will never acknowledge the first Two-phase action on A, so the Two-phase CHP does not precisely match the Tangram specification. If we erase unreachable actions from the Tangram specification, then the Two-phase CHP matches exactly.

3) Sequencer: Two specifications are given for the sequencer:



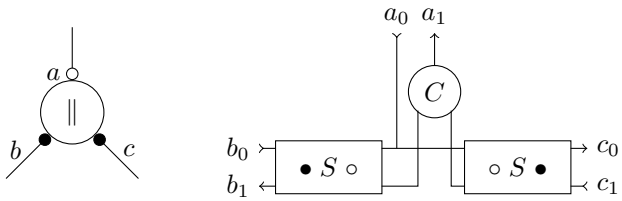
a) Tangram Diagram b) Circuit Diagram

Fig. 4. Tangram Repeater



a) Tangram Diagram b) Circuit Diagram

Fig. 5. Tangram Sequential Element



a) Tangram Diagram b) Circuit Diagram

Fig. 6. Tangram Parallel Element

$$*[A^{\uparrow}? \star (B^{\uparrow}!; B^{\downarrow}!; C^{\uparrow}!); A^{\downarrow}? \star C^{\downarrow}!]$$

$$*[A^{\uparrow}? \star B^{\uparrow}!; A^{\downarrow}? \star (B^{\downarrow}!; C^{\uparrow}!; C^{\downarrow}!)]$$

However, only the circuit shown in Figure 5 is given as an implementation. We can invert the production rules corresponding to the Tangram circuit into the following HSE:

$$\begin{aligned} *[&[[a_0]; b_0\uparrow; [b_1]; b_0\downarrow; [\neg b_1]; c_0\uparrow; \\ &[c_1]; a_1\uparrow; [\neg a_0]; c_0\downarrow; [\neg c_1]; a_1\downarrow] \end{aligned}$$

This was automatically inverted to construct the following Two-phase CHP:

$$*[A^{\uparrow}? \star (B^{\uparrow}!; B^{\downarrow}!; C^{\uparrow}!); A^{\downarrow}? \star C^{\downarrow}!]$$

Which represents the first specification, and is thus a correct implementation. This translates into the CHP:

$$*[[\overline{A}]; B!; C!; A?]$$

4) Par: The Par element, shown in Figure 6, is used to compose two actions in parallel. No explicit Two-phase specification is given, but it is described as a structure where a handshake on A encloses parallel handshakes on B and C.

We can decompile this circuit using our flow, representing the outputs of the S-elements connected to B as z, and to C as y, we derive the HSE:

$$\begin{aligned} *[&[[a_0]; ((b_0\uparrow; [b_1]; b_0\downarrow; [\neg b_1];\ \ z\uparrow) \\ &\quad \| \ \ (c_0\uparrow; [c_1]; c_0\downarrow; [\neg c_1];\ \ y\uparrow)); \\ &a_1\uparrow; [\neg a_0]; (z\downarrow \| y\downarrow); a_1\downarrow\ ] \end{aligned}$$

This decompiles to the Two-phase CHP

$$*[A^{\uparrow}? \star ((B^{\uparrow}!; B^{\downarrow}!) \| (C^{\uparrow}!; C^{\downarrow}!)); A^{\downarrow}?]$$

and finally, to the CHP:

$$*[[\overline{A}]; (B! \| C!); A?]$$

This CHP corresponds to the textual description of the Par operation.

5) Non-Receptive Mixer: The Tangram non-receptive mixer element depicted in Figure 7 is a structure for joining two channels into a single channel, it has a requirement on its usage that the environment must guarantee that the two input channels are accessed strictly sequentially. Given that, it can be considered as the superposition of two specifications:

$$*[A^{\uparrow}? \star C^{\uparrow}!; A^{\downarrow}? \star C^{\downarrow}!]$$

$$*[B^{\uparrow}? \star C^{\uparrow}!; B^{\downarrow}? \star C^{\downarrow}!]$$

Decompiling the circuit in conjunction with the partial environments corresponding to these two scenarios allows the tool to derive:

$$*[[a_0]; c_0{\uparrow}; [c_1]; a_1{\uparrow}; [\neg a_0]; c_0{\downarrow}; [\neg c_1]; a_1{\downarrow}]$$

$$*[[b_0]; c_0{\uparrow}; [c_1]; b_1{\uparrow}; [\neg b_0]; c_0{\downarrow}; [\neg c_1]; b_1{\downarrow}]$$

These can be decompiled into the Two-phase CHP processes:

$$*[A^{\uparrow}? \star C^{\uparrow}!; A^{\downarrow}? \star C^{\downarrow}!]$$

$$*[B^{\uparrow}? \star C^{\uparrow}!; B^{\downarrow}? \star C^{\downarrow}!]$$

which correspond to the specifications given. A more detailed reconstruction may recombine them to the CHP

$$\begin{array}{l} *[[\overline{A} \;\longrightarrow\; B!;\; A? \\ \;\;[]\overline{B} \;\longrightarrow\; C!;\; A? \\ ]] \end{array}$$

However, this selection-statement reconstruction is not performed by the current version of the decompilation tool that we have developed.

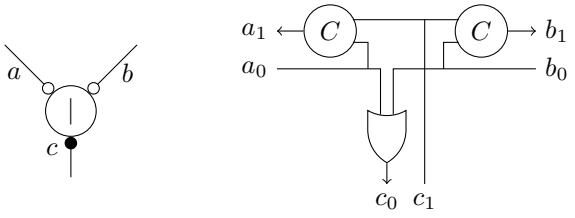
6) Join: The Tangram join element, shown in Figure 8, is the dual of the par element, taking communications on A and B and enclosing them in C. We can derive the HSE from the circuit to obtain:

$$\begin{array}{l} *[\;\; [a_0 \wedge b_0]; c_0{\uparrow}; [c_1]; (a_1{\uparrow} \parallel b_1{\uparrow}); \\ \quad [\neg a_0 \wedge \neg b_0]; c_0{\downarrow}; [\neg c_1]; (a_1{\downarrow} \parallel b_1{\downarrow}) \;\;] \end{array}$$

This decompiles to the Two-phase CHP:

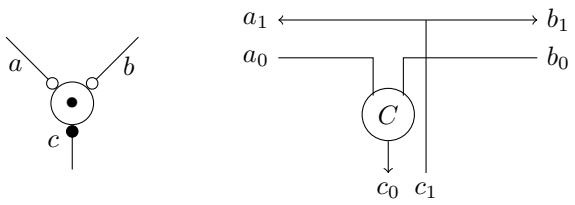
$$*[\;\; A^{\uparrow}? \star B^{\uparrow}? \star C^{\uparrow}!; A^{\downarrow}? \star B^{\downarrow}? \star C^{\downarrow}!]$$

which corresponds to the CHP:



a) Tangram Diagram b) Circuit Diagram

Fig. 7. Tangram Non-Receptive Mixer Element



a) Tangram Diagram b) Circuit Diagram

Fig. 8. Tangram Join Element

$$*[\;\; A? \star B? \star C! \;\;]$$

One given requirement of the join element is that when the B and C ports are connected to the B and C ports of the par element, the communication proceeds. This is the case for this CHP, and so we say that this circuit implements the specifications given.

V. Conclusions and Future Work

We have discussed and given inversions for each stage of Martin Synthesis, dividing up stages when this was found to give more natural inverse transformations. These inversions have been composed together to produce a decompilation tool, which has been successfully applied to reconstruct specifications of QDI circuits with only limited information about the original specification. We showed how the approach could be applied to construct CHP-language specifications for a number of different control circuits.

The analysis here is limited to non-inherently disjunctive systems, but could be extended using methods similar to the Extended Repetitive Event Rule extension of Repetitive Event Rule Systems [13]. Outside of the examples presented in this paper, it has been used to verify the sequencing of simple data-carrying circuits, but currently cannot reconstruct non-trivial data-carrying elements or complex selection statements. Future work will be focused on more general reconstruction that includes data as well as selection statements.

Acknowledgements

This research is supported by by the NSF GRFP under grant DGE-0707428, IARPA under award N66001-12-C-2009, and the NSF TRUST STC under CCF-0424422.

Appendix A

Here we informally present the notation we use to describe QDI asynchronous circuits. A formal trace-tree based semantics can be found in [17].

A. CHP

As convention, variables are ranged over by lowercase letters, $a, b, c \ldots$, channels are ranged over by uppercase letters from the beginning of the alphabet, $A, B, C \ldots$, and programs or program segments are ranged over by uppercase letters from the second half of the alphabet, $P, Q, R \ldots$

- **Skip**: `skip`. This statement does nothing.

- **Assignment**: $x := E$. This statement means "assign the value of expression E to x." The statements $x\uparrow$ and $x\downarrow$ are shorthand for $x :=$ **true** and $x :=$ **false**, respectively.
- **Communication**: $A!e$ is a statement meaning "send the value of e over channel A," and $B?x$ means "receive a value over channel B and store it in variable x." Both sending and receiving are blocking.
- **Selection**: $[G_1 \rightarrow P_1 [] ... [] G_n \rightarrow P_n]$, where each G_i is a boolean expression, and each P_i is a statement. This statement is executed by waiting for one of the guards to be true, and then executing one statement with a true guard. If the guards are not mutually exclusive, we use the thin bar (|) instead of the thick bar ([]). $[G]$ is used as a shorthand for $[G \rightarrow \texttt{skip}]$.
- **Repetition**: $*[G_1 \rightarrow P_1 [] ... [] G_n \rightarrow P_n]$. This statement is executed by choosing one of the true guards and executing the corresponding statement, repeating until all guards evaluate to false. If the guards are not mutually exclusive, we use the thin bar (|) instead of the thick bar ([]). $*[P]$ is used as a shorthand for $*[\textbf{true} \rightarrow P]$.
- **Probe**: The boolean $\overline{A}$ is true if and only if a communication on channel A can complete without suspending. Probes are only allowed to occur in the guards of selection statements.
- **Sequential Composition**: $P; Q$. This statement means "execute statement P, and then execute statement Q."
- **Parallel Composition**:$P \parallel Q$. This statement means "execute statement P, while simultaneously executing statement Q."
- **Simultaneous Composition**: $P \star Q$. This statement means "execute channel actions P and Q such that they begin and complete simultaneously." This composition was introduced in [7] and replaces $\bullet$ composition of Martin [3]. Unlike the $\bullet$ composition, it has a well-defined definition in Two-phase CHP, and will deadlock when composed in parallel with itself. $\star$ binds more tightly than ;, which in turn binds more tightly than $\parallel$.

Weak Fairness Concurrent execution of CHP processes is assumed to be *weakly fair*, meaning that every continuously enabled action will eventually be given a chance to execute.

B. HSE

Handshaking Expansion (HSE) is a subset of CHP, where all actions on channels (communication, probes) are translated into Boolean variables, and all non-Boolean data representations are given Boolean representations.

C. PRS

A Production Rule Set (PRS) is a set of production rules, each of the form:

$$G \mapsto t$$

where t is a simple boolean assignment, and G is a boolean expression, known as the guard of the production rule. We require a few properties of a valid PRS:

- **Non-Interference** Production rules $G^+ \mapsto x\uparrow$ and $G^- \mapsto x\downarrow$ are said to be non-interfering in a computation if and only if $\neg G^+ \vee \neg G^-$ is an invariant of the computation. A valid PRS must only contain non-interfering rules, as interfering rules correspond to a short circuit.
- **Stability** A production rule $G \mapsto t$ is said to be stable in a computation if and only if G can change from true to false only after the assignment on t has completed. A valid QDI PRS will only contain stable production rules.

A PRS is executed as a state transformer, where any of the rules whose guards are satisfied by the current state may execute, modifying the state with their associated boolean assignment.

REFERENCES

[1] J. OLeary, X. Zhao, R. Gerth, and C.-J. Seger, "Formally verifying ieee compliance of floating-point hardware," *Intel Technology Jounal*, Q1'99.

[2] C. Hoare, "Communicating sequential processes," in *Communications of the ACM*, vol. 21, no. 8, August 1978, pp. 666–677.

[3] A. J. Martin, "Compiling communicating processes into delay-insensitive VLSI circuits," vol. 1, no. 4, pp. 226–234, 1986.

[4] ——, "Tomorrow's digital hardware will be asynchronous and verified," in *Information Processing 92, Vol. I: Algorithms, Software, Architecture*, ser. IFIP Transactions, J. van Leeuwen, Ed., vol. A-12. Elsevier Science Publishers, 1992, pp. 684–695.

[5] S. F. Smith and A. E. Zwarico, "Correct compilation of specifications to deterministic asynchronous circuits," in *Formal Methods in System Design, 7:155–226*. Kluver Academic Publishers, 1995, pp. 7–155.

[6] I. D. Baxter and M. Mehlich, "Reverse engineering is reverse forward engineering," *Working Conference on Reverse Engineering*, 1997.

[7] R. Manohar, "An analysis of reshuffled handshaking expansions," in *Asynchronous Circuits and Systems, 2001. ASYNC 2001. Seventh International Symposium on*, 2001, pp. 96 –105.

[8] S. M. Burns, "Performance analysis and optimization of asynchronous circuits," Ph.D. dissertation, California Institute of Technology, December 1990.

[9] A. Lines, "Pipelined asynchronous circuits," Master's thesis, California Institute of Technology, 1998.

[10] M. Ercegovac and T. Lang, *Digital Arithmetic*. Morgan Kaufmann, 2004.

[11] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools (2nd Edition)*. Addison Wesley, August 2006.

[12] J. Tierno, R. Manohar, and A. Martin, "The energy and entropy of vlsi computations," in *Advanced Research in Asynchronous Circuits and Systems, 1996. Proceedings., Second International Symposium on*, mar 1996, pp. 188 –196.

[13] T. K. Lee, "A general approach to performance analysis and optimization of asynchronous circuits," Ph.D. dissertation, California Institute of Technology, May 1995.

[14] S. M. Burns, "Automated compilation of concurrent programs into self-timed circuits," Master's thesis, California Institute of Technology, 1988.

[15] K. van Berkel, J. Kessels, M. Roncken, R. Saeijs, and F. Schalij, "The vlsi-programming language tangram and its translation into handshake circuits," in *Proceedings of the conference on European design automation*, ser. EURO-DAC '91. Los Alamitos, CA, USA: IEEE Computer Society Press, 1991, pp. 384–389.

[16] K. Van Berkel, *Handshake Circuits: An Asynchronous Architecture for Vlsi Programming*, ser. Cambridge International Series on Parallel Computation : 5. Cambridge University Press, 1993.

[17] M. van der Goot, "Semantics of vlsi synthesis," Ph.D. dissertation, California Institute of Technology, May 1995.