A Formal Approach To Design Space Exploration Of Protocol Converters

Karin Avnit, Arcot Sowmya
School of Computer Science and Engineering
The University of New South Wales, Sydney Australia
{kavnit,sowmya}@cse.unsw.edu.au

Abstract

In the field of chip design, hardware module reuse is a standard solution to the increasing complexity of chip architecture and the pressures to reduce time to market. In the absence of a single module interface standard, integration of pre-designed modules often requires the use of protocol converters. For an arbitrary pair of incompatible protocols it is likely that there exist more than one possible converter. However, existing approaches to automatic synthesis of protocol converters either produce a single suggested converter or provide a general nondeterministic solution, out of which a designer is required to extract a deterministic converter.

In this work we present a novel approach for design space exploration of FSM based protocol converters. We present algorithms for extraction of minimal converters for a given pair of incompatible protocols. We demonstrate the process through a simple example, and report on results of experiments with converters for commercial protocols AMBA ASB, APB and the Open Core Protocol (OCP). The experiments show a reduction in the number of states in the converter of as much as 62% (with an average reduction of 42%) and a reduction in the number of transitions of as much as 85% (with an average reduction of 61%), demonstrating the benefits of design space exploration.

1 Introduction

Hardware module reuse is a standard solution to the problem of increasing complexity of chip architectures and growing pressure to reduce time to market. Much research has been dedicated to problem of automatic synthesis of converters to mediate between incompatible protocols. Practical converter synthesis algorithms are crucial to realizing the dream of “plug-n-play” style SoC design. Complete automation of the converter synthesis process can save time and effort in both design and verification phases and reduce the risk of human error.

For an arbitrary pair of incompatible protocols it is likely that there exists more than one possible converter, and there can be great differences in performance. Current solutions to automatic converter synthesis either provide methods for constructing a single specific converter, or produce a general nondeterministic converter. Such a general converter constitutes a design space for possible deterministic converters and requires exploration of the design space for extraction of such converters.

Our work focuses on design space exploration of a general nondeterministic converter and automatic extraction of smaller more deterministic converters. We present formal definitions for converter pruning and show that the suggested pruning method guarantees correctness of the resulting converters. We report on experiments with converters for commercial protocols such as AMBA ASB and APB [3] and the Open Core Protocol [9].

The system model assumed in this work is illustrated in Figure 1. The modules communicate through control and data channels and the converter is assumed to be constructed from FSM logic and a set of buffers, which is used for storage of data items provided by one protocol until it can be read by the other protocol.

1.1 Related Work

The focus of most research in the field of automatic converter synthesis has been, expectedly, on finding methods for automating the design of protocol converters. Little attention has been devoted to the fact that for a pair of incompatible protocols there may exist more than one correct converter, and a designer should have methods to explore the different options for correct conversion. Most methods proposed concentrate on generating a general nondeterministic converter, that includes internal choices, and that still requires manual resolution at the end of the process.

In early work [1], protocols were represented as state
machines and their cross product was used to construct a converter. This work was highly innovative but preliminary. This approach was later extended [2, 5, 6] and more recently [4]. These methods build a converter out of all possible behaviors of the protocols and produce a general nondeterministic converter. In addition to producing the general converter, Androutsopoulos et al. [22] propose a method for optimizing the converter by the removal of internal choice to maximize data transfer rate.

Passerone et al. [11] specify mismatched synchronous protocols as regular expressions and in later work [10] attempt a game theory formalization. Relying on this work, recent work [14] proposes partitioning protocols into sets of transactions and synthesizing a converter by merging small converters for individual transactions. Again, these methods do not address internal choice of the converter and require further processing to arrive at a deterministic converter.

Methods relying on a standard communication scheme [7, 12, 13] suggest communicating with the protocols by mirroring their specifications, and this also introduce internal choice into their converter, that is not resolved.

As far as we know, no work has been published, that solves the problem of automatic design space exploration for protocol converters. The design space exploration method proposed in this paper should not be confused with the field of design space exploration in system design, that deals with mapping system specifications to given architecture templates (eq. [8]). The proposed method also differs from the research of finite state machine optimization for hardware design. The latter focuses on lower levels of abstraction, such as state coding and the minimization of the number of states in machines, by providing a state machine that implements some behaviors with a minimal number of states (eq.[15]). The problem addressed in this paper is to find the behaviors to be implemented, in the design space of all possible behaviors.

1.2 Original Contribution

The contribution of this work is in its ability to bring automatic converter synthesis yet another step closer to realization. The definitions and algorithms presented in this work allow efficient design exploration that produces provably correct deterministic converters. Comparing the characteristics of these converters lead to an optimization of the converter over the compared metrics.

2 Background

As the base of our work, we rely on the formal foundations already presented ([4]). In that work protocols were modeled as FSMs and formal definitions presented for protocol compatibility and correct protocol conversion. In addition, an algorithm for automatic synthesis of the Most General Correct Converter (MGC²) was presented. The MGC² is a nondeterministic converter which includes all possible correct behaviors for conversion between a given pair of protocols. In this section we recap the definition of compatibility and the process of generation of the MGC² via a simple example.

 Compatibility: Two protocols (P₁ and P₂) are defined to be compatible if and only if when they communicate:
1. Data items are read only when written,
2. A data item is read as distinct exactly once,
3. No deadlocks occur and livelocks can be avoided.
A parallel composition (P₁ ∥ P₂) is defined as a machine that includes all behaviors that can be reached when the two protocols communicate, and compatibility is decided by examining the behaviors in P₁ ∥ P₂.

Consider the pair of protocols illustrated in Figure 2. This pair is incompatible, because when they run concurrently it is possible that P₂ will try to read from the data channel when it is not guaranteed to hold a valid value. The reachable part of the parallel composition is illustrated in Figure 3, and the violation of the compatibility definition is manifested in the transition from state (0,2) to state (0,0). For such a pair of protocols it is still possible to synthesize a correct converter, with the use of buffers to store data items that are provided by one protocol until they can be read by the other protocol.

 Correct conversion: A correct converter is one that:
1. Is compatible with both protocols,
2. Never causes an overflow or an underflow of its buffers.
For a pair of incompatible protocols there may be many different correct converters. The algorithm previously presented ([4]) produces a correct-by-construction converter which includes all possible correct behaviors - the MGC₂. The MGC₂ for the protocols in Figure 2 is illustrated in Figure 4. The ranges specified in the transition labels represent restrictions over the number of items stored in the address and data buffers (also referred to data state). The restriction \([x, y]\) means that a transition is enabled only if the number of data items stored in the buffer is between \(x\) and \(y\), inclusive, and \(B\) represents the buffer size.

### 3 Design Space Exploration

The MGC₂ is not guaranteed to be deterministic. In the MGC₂ in Figure 4, for example, nondeterminism occurs in states \((0,0),(0,1)\) and \((0,2)\). In each of these states there are input control values that enable more than one transition.

Given the MGC₂, we would like to be able to extract out of it smaller more deterministic converters that are included in the MGC₂, that are still as correct as the MGC₂ and are also optimal under given criteria. In order to achieve this, we define pruning rules, which are used to remove non-deterministic choices without affecting the correctness of the converters.

#### 3.1 Pruning Rules

To prune a converter while still maintaining its correctness, we need to be able to identify states at which non-determinism occurs, and the converter can make a choice between two or more transitions. In such states there are Input Control Values (ICVs) and data states (the number of data items in each buffer) for which more than one transition is enabled. By making a prior decision on this choice we can remove the non-determinism and possibly reduce the converter size.

Let \(ICV(t)\) denote the set of ICVs for which a transition \(t\) is enabled. Let \(range(t)\) denote the data states for which the transition \(t\) is enabled. We now define formally the notion of redundancy of transitions.

**Definition 1 (A Redundant Transition).** A transition \(t\) outgoing state \(q\) in a converter \(C\) is a redundant transition iff
\[
\forall v \in ICV(t) \text{ there exist transitions } t_1 \ldots t_n \neq t \text{ also outgoing state } q, \text{ such that } v \in ICV(t_i) \text{ for } 1 \leq i \leq n, \text{ and range}(t) \subseteq \text{range}(t_1) \cup \text{range}(t_2) \ldots \cup \text{range}(t_n).
\]

In other words, a transition \(t\) is redundant if for every control input value that enables it, there are also other transitions from the same source state that are enabled for the same control input value, and they cover all data states that are enabling \(t\). Note that the redundancy here is in the sense that the transition \(t\) can be removed without affecting the correctness of the converter, and does not imply redundancy in the behaviors that it enables. A transition can be Partially Redundant if it can be split into two transitions such that one of them is redundant and the other is not. This is possible when some of the ranges in the transitions can be covered by other transitions that are enabled for its control input while other ranges cannot. Alternatively, a transition can be partially redundant if only some of its enabling ICVs are common with other transitions.

In Figure 5, transitions \(t_1\) and \(t_2\) are redundant as they are covered by \(t_3\). Transition \(t_3\) is only partially redundant, since it can be split into transition 0 \(\xrightarrow{c_1\{1,3\}} 3\) which is redundant, and transition 0 \(\xrightarrow{c_1\{4,5\}} 3\) which is not redundant.

Examples of correct and incorrect pruning of state 0 from Figure 5 are provided in Figure 6. In Figure 6(a) the choice was made to use transition \(t_3\) only where it is the only one enabled. The example of Figure 6(b) represents a choice to take transition \(t_3\) for data states 2 to 5 but to use the other transition if there is only one item in the buffer. In Figure 6(c) transition \(t_3\) is always taken, causing the removal of states 1 and 2. In these three examples all ICVs that enabled a transition before pruning (Figure 5) still enable some transition. In contrast, the example of Figure 6(d) demonstrates incorrect pruning, as the reaction of the converter to some ICVs (such as \(c_1?, c_2\#, [1, 3]\)) that was defined before pruning is now no longer defined.
Theorem 3.1. Removal of a redundant transition from a correct converter preserves correctness.

Proof. Let $C'$ be the machine resulting from the removal of a redundant transition $t$ from the correct converter $C$.

In order to prove that $C'$ is a correct converter we examine the effect of the removal of a redundant transition $t$ from $C$, on the definition of a correct converter as in [4], and as explained in Section 2. According to the definition, $C'$ needs to be compatible with both protocols and needs to prevent overflows and underflows of the buffers.

Given that $C$ is a correct converter, it cannot cause any overflows or underflows of the buffers in any transition. All behaviors of $C'$ are included in $C$ (all transitions are included), therefore $C'$ cannot cause any overflows or underflows of the buffers as well. It remains to be shown that $C'$ is compatible with the protocols.

The compatibility definition refers to the three constraints presented in Section 2.

As all transitions of $C'$ are included in $C$, it is clear that there can be no transition in $C'$ in which a data item is read even though it is not written. In the same way, as a removal of a transition cannot add behaviors that are not possible in $C$, a violation of item 2 is not possible. It remains to show that removal of a redundant transition cannot introduce deadlocks or livelocks into the converter.

In this framework, deadlocks and livelocks are manifested as reachable states (with respect to the initial state) that cannot reach the final state. A removal of a transition from a state $q$ in a correct converter can introduce a deadlock or a livelock at the state if a combination of an ICV and a data state restriction that had enabled a transition in the original correct converter no longer enables any transition in the resulting converter. However, by definition of a redundant transition, for every ICV and data state for which the transition is enabled, there is some other transition from the same source state, that is also enabled.

Theorem 3.2. Removal of a transition that is not redundant from a correct converter impairs the converter correctness.

Proof. Removal of a transition that is not redundant from a state $q$ in a correct converter $C$, by definition of a non redundant transition, results in a combination of an ICV and a buffer state that used to enable a transition but no longer does. As the $MGC^2$ is designed to react to all possible inputs of the protocols and only to possible inputs of the protocols, the removal of a non-redundant transition means that there is an ICV that may be introduced by the protocols, which that does not enable any transition in the converter. This introduces a deadlock into the converter and violates the compatibility of the converter with the protocols.

3.2 Automatic Design Space Exploration

An algorithm for automatic design space exploration needs to identify states in the converter in which redundant or partially redundant transitions exist, and explore all possible pruning options for removal of nondeterminism. The output of such algorithm would be a list of all possible correct and deterministic converters for a specific pair of incompatible protocols.

Algorithm 1 is a recursive algorithm that is designed to identify nondeterminism for a specific ICV ($icv$) in a state $q$ and return a list of structures of type choice ($icvs$) for which one transition is enabled, a list of ranges and the ICV for which a choice is possible. It takes as input the set of transitions, a list of ranges and the ICV for which a choice is possible. It returns a list of choices, each holding a list of transitions, a list of ranges and the ICV for which a choice is possible. The output of such algorithm would be a list of all possible correct and deterministic converters for a specific pair of incompatible protocols.

Consider a state as illustrated in Figure 7. Applying Algorithm 1 to this state produces the following choice:
Algorithm 1 ICV_CHOICE(T, icv)
Input: T = \{t_1, t_2 \ldots t_n\} - a group of transitions, icv - an input control value.
Output: a list of choices.

static Free_Range = full range
Common = (\bigcap_{t \in T} range(t)) \cap Free_Range
if Common \neq \emptyset then
    Free_Range = Free_Range \cap Common
    [T, T'] = \text{SPLIT_TRANSITIONS}(T)
    Add new choice(T', Common, icv)
end if
if Free_Range = \emptyset \lor |T| = 2 then
    return
end if
for i = 1 to |T| do
    ICV_CHOICE(T/t_i, icv)
end for

Algorithm 2 SPLIT_TRANSITIONS(t, icv)
Input: A transition t = q \xrightarrow{S} q' and a control input value.
Output: Two mutually exclusive transitions

Others = S/input_control_actions(S)
S_{icv} = icv \cup Others
if input_control_actions(S) = icv then
    return [\emptyset, q \xrightarrow{S_{icv}} q']
else
    Common = S \cap icv
    Exclusive = icv/Common
    S' = Others \cup Common \cup (\bigcup_{action \in Exclusive} action)
end if
return [q \xrightarrow{S'} q', q \xrightarrow{S_{icv}} q']

\[
\left\{ 0 \xrightarrow{c1?[0,7]} 1, 0 \xrightarrow{c1?[3,10]} 2 \right\}, [3,7], c1? \right\}.
\]

Theoretically, this choice represents different solutions to the nondeterminism, as every value in the common range (of size 5) can be assigned to each of the two transitions.

Once Algorithm 1 is run for every state q in the converter and every icv \in ICV(q), it remains to generate the different converters that arise from the different possible choices. Going over every one of the choice structures, resolution of a specific nondeterminism is possible in a few ways. The most general way is to produce all possible assignments of state values that are common to a set of transitions. In this case, a choice with a range of size n that is common to m transitions yields \( m^n \) different solutions, as each value in the range can be attached to each of the transitions. However, in many cases there are no practical differences (such as in the number of states and transition and design area) between different choice solutions, and the complete set of solutions may not be worth exploring. Instead, a more restricted exploration can be used. An easy exploration may be to retain the entire range that is common to the set of transitions, and attach it to each of the transitions - yielding m different solutions. A compromise between the two approaches may be to choose possible m sub-ranges and attach each to the different transitions. The number of different possible assignments in this case can be bounded as follows: the number of possible sub-ranges is equivalent to choosing m – 1 positions out of a possible n + 1 positions, including repetitions. Each set of positions can be assigned to every possible permutation of the m transitions, and in total the number of possible options is bounded by \( m \times (m + n - 1)/n! \). It is important to note that the number of transitions involved in a choice (m) is normally very small (typically 2 or 3 transitions), while the common ranges may be much bigger and depend on the buffer sizes.

The possible assignments of the common range [3,7] from Figure 7, to either of the transitions, according to the three approaches discussed above, are listed in Table 1. The Full Range Assignment (FRA) yields two options - attaching the common range to either the first or the second transition. In The Sub-Range Assignment (SRA) the range [3,7] is split into two sub-ranges and each sub-range is attached to either of the transitions. In the Single Value Assignment (SVA) all possible assignments of each value in the common range are explored.

<table>
<thead>
<tr>
<th>Table 1. Possible Range Assignments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full Range Assignment (FRA): [{\emptyset}, {3 - 7}]</td>
</tr>
<tr>
<td>Sub-Range Assignment (SRA): FRA \cup {{3}, {3, 4}, {3 - 5}, {3 - 6}, {4 - 7}, {5 - 7}, {6, 7}, {7} }</td>
</tr>
<tr>
<td>Single Value Assignment (SVA): SRA \cup {{4}, {5}, {6}, {5, 6}, {5, 7}, {4, 5}, {4, 6}, {4, 7}, {4, 6, 7}, {4, 5, 7}, {3, 7}, {3, 6}, {3, 5}, {3, 6, 7}, {3, 5, 7}, {3, 5, 6, 7}, {3, 4, 7}, {3, 4, 6}, {3, 4, 6, 7}, {3, 4, 5, 7} }</td>
</tr>
</tbody>
</table>

It is easy to see that if the purpose of the design space exploration is to minimize the number of states and transitions, then the only type of exploration needed is that of Full Ranges Assignment, as it includes the most empty-set assignments, and these are the assignments that may lead to a reduction in the number of transitions and states.
Table 2. Experimental Results

<table>
<thead>
<tr>
<th>Initiator</th>
<th>Reactor</th>
<th>$MGC^2$</th>
<th>$C_{min}$</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMBA ASB</td>
<td>AMBA APB</td>
<td>10,19</td>
<td><strong>9,15</strong></td>
<td>10%, 21%</td>
</tr>
<tr>
<td>OCP</td>
<td>AMBA APB</td>
<td>4,15</td>
<td>2,3</td>
<td>50%, 80%</td>
</tr>
<tr>
<td>OCP</td>
<td>AMBA ASB</td>
<td>16,40</td>
<td><strong>7,10</strong></td>
<td>56%, 75%</td>
</tr>
<tr>
<td>$p_1^+$</td>
<td>$p_2^+$</td>
<td>6,12</td>
<td><strong>3,4</strong></td>
<td>50%, 61%</td>
</tr>
</tbody>
</table>

* The protocols in Figure 2.

Figure 8. A Minimal Converter For $p_1$ and $p_2$

4 Experimental Results

The proposed methods and algorithms were applied to converters mediating between different pairs of protocols as listed in Table 2. In all listed experiments, the $MGC^2$ was produced with buffer size 10 and the exploration was conducted for finding the deterministic converter with the smallest number of states. In the table, the first and second columns detail the protocols used. The $MGC^2$ column shows the number of states and transitions in the most general correct converter, the number of states and transitions in the column also represent the maximal values for a deterministic converter. $C_{min}$ shows the number of states and transitions in the smallest converter found in the exploration. The last column of the table show the improvement in the number of states and transitions as a percentage of $C_{min}$ states and transitions relative to those of the $MGC^2$.

The table shows that exploring the design space can lead to great improvements in the number of states and transitions and this improvement is generally greater when the protocols involved are more complex (as AMBA ASB and OCP). This improvement is because more complex protocols are usually more flexible and allow for more choices on behalf of the communicating party, and in our case - the converter. A minimal converter found for the $MGC^2$ of Figure 4 is presented in Figure 8.

5 Conclusions

In this work we have presented a method for design space exploration of SOC buffer based converters within the framework of finite state machine modeling. The presented method relies on formal foundations and is guaranteed to produce correct-by-construction, deterministic converters. The results presented in this work show that design space exploration can assist in optimizing the converter and the improvements gained can be very significant. The experiments show that the reduction in the number of states can be over 60% between different converters, and the reduction in the number of transitions can be even greater.

6 Acknowledgments

The authors would like to thank Vijay D’Silva of Oxford university, and Sri Parameswaran of UNSW Australia, for their technical and presentational contributions. This research is supported by the Australian Research Council discovery project grant since 2005.

References