A Novel Synthesis Technique for Communication Controller Hardware from declarative Data Communication Protocol Specifications

Robert Siegmund, Dietmar Müller
Chemnitz University of Technology
Professorship Circuit and Systems Design
09107 Chemnitz, Germany
{rsie,mueller}@infotech.tu-chemnitz.de

ABSTRACT

An innovative methodology for the efficient design of communication controller hardware for popular protocols such as ATM, USB or CAN is proposed. In our approach, controller hardware in form of RTL models is synthesized from a formal specification of a communication protocol. The difference to previously published work related to hardware synthesis techniques from protocol specifications is that in our approach a complete communication architecture consisting of *both* the interacting transaction producer and the consumer controllers, as well as the interconnect between them, are synthesized from *one single* protocol specification in the *same* synthesis tool run, thus ensuring conformity of all producer and consumer controllers to the protocol specification while tremendously reducing the modeling effort for the controller specifications. The formalism used for protocol specification and a corresponding hardware synthesis algorithm from such specifications are presented. The methodology has been applied to the design of various communication controllers including IEC14443 Wireless SmartCard, ATM and CAN. The novelty and efficiency of our methodology is demonstrated through comparison to State-of-The-Art protocol synthesis tools such as [10].

Categories and Subject Descriptors

B.4.4 [**Hardware**]: Input/Output and Data Communications Devices—*Performance Analysis and Design Aids*; B.5.2 [**Hardware**]: Register-Transfer-Level-Implementation—*Design Aids*

General Terms

Algorithms, Design, Languages

Keywords

Protocol Specification, Controller Hardware Synthesis, Interface-based Design

DAC 2002, June 10-14, 2002, New Orleans, Louisiana, USA.

1. INTRODUCTION

Today's digital systems are communication-dominated designs. They comprise a large number of modules of high functional complexity which communicate with each other by means of high-performance data protocols. Especially in order to minimize the physical interconnect between system modules, bit-serial protocols using a single wire for data transmission such as USB, CAN or FireWire are extensively applied. However, corresponding protocol controllers are control-oriented designs which include complex state machines. The manual design of such protocol controllers is thus a tedious, error-prone and time-consuming task.

In this paper, a novel methodology for the efficient design of communication controller hardware which is especially suited for (but not limited to) complex, bit-serial protocols is presented. The methodology is based on synthesis of controller hardware from a formal high-level specification of the protocol. Compared to previous work [6],[8],[4],[5],[10], our approach is unique in two aspects: First, the formalism that is used for protocol specification has been realized as an extension to the system description language SystemC, called SystemC^{SV} . This ensures that protocol specifications become an integral part of the system specification and can be simulated and verified directly in the system context. State-of-the-Art approaches published to date require the generation of corresponding behavioural models for transaction producer and consumer in a HDL or C from the protocol specification which have then to be inserted manually into the system model prior to simulation. The major drawback is that errors in the protocol description are only reflected in the simulated behaviour of these modules so that from the erroneous behaviour the necessary corrections of the protocol specification must be concluded.

The second aspect is related to the fact that in any kind of communication between two or more system modules usually two distinct types of interacting controllers are involved: Transaction producing controllers, which structure and encode the information to be transmitted into a signal sequence on a physical communication medium, and transaction consuming controllers which assemble this signal sequence and extract the relevant information (see Fig 1). The approach presented in this paper is to the author's knowledge the first one that generates *both* transaction producer and consumer controllers in form of RTL models from *one single* protocol specification in the same run of the synthesis algorithm. Beside a tremendous raise in design efficiency and reduced modeling effort for the protocol specifications, this approach ensures that the behaviours of both transaction producer and consumer conform to the protocol specification, a property which has to be otherwise verified

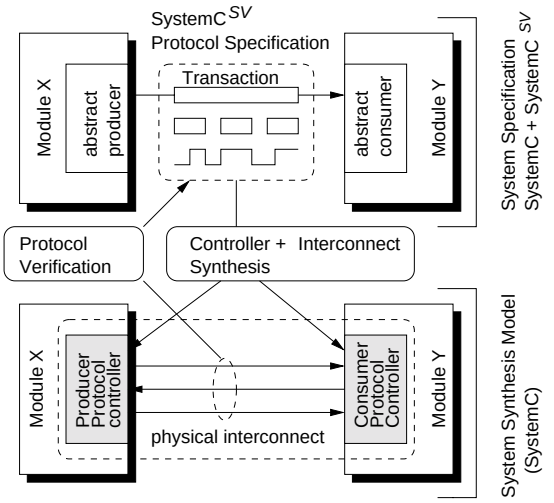


Figure 1: Overview over the proposed Methodology

in extensive simulations using protocol reference models. However, in case verification of the controller implementation against the protocol specification is required, in the presented approach the protocol specification also serves as a protocol checker during system simulations (Fig. 1). Details on this can be found in [9].

1.1 Related Work and Motivation

High-level communication modeling concepts are an integral part of a large number of State-of-the-Art system modeling languages: So do SystemC [11] and SpecC [2] provide the concept of interfaces and channels which are an abstraction of the signal level communication and enable the specification of hierarically structured communication protocols. These modeling concepts support an interface-based design style [7] through separation of communication and computation. However, the communication modeling concepts in these languages have been primarily designed for simulation purposes and to date no feasible hardware synthesis methodologies have been presented for related interface protocol descriptions.

A number of publications have addressed synthesis of communication controller hardware from protocol descriptions. In [8] the system Clairvoyant was presented which generates controller hardware from production-based protocol specifications. An extension to this system is the Protocol Compiler [10], which uses a graphical hierarchical regular expression based language for protocol specification. In [4],[5] regular grammars are used for protocol descriptions which can be synthesized into a VHDL description of a corresponding protocol controller by means of the PRO-GRAM system.

The drawback of these synthesis approaches is that the formalisms used to capture the protocol specification are not part of a system modeling language. Therefore, no verification support for the protocol is offered by the standard simulation tools, and the methods are complicated to integrate into existing design flows. Another disadvantage that is common to all formalisms for communication protocol specification presented so far is that in order to synthesize a transaction producer/consumer pair, the protocol must be described from two different views: the transaction producer view and the transaction consumer view. Therefore, for the same protocol two different specifications have to be developed which results in a decrease in design efficiency and increases verification effort.

The rest of the paper is organized as follows: In Sect. 2 the formalism used for protocol specification is presented. Section 3 describes our approach to synthesize controller hardware from such specifications. The synthesis results obtained for various protocols such as IEC14443 SmartCard, ATM and CAN are given in Sect. 4. Section 5 concludes the paper.

2. SPECIFICATION OF DATA COMMUNICATION PROTOCOLS

The formalism which is used to capture specifications of data communication protocols is an extension to SystemC and named SystemC^{SV} [9]. In SystemC^{SV} , system modules (which themselves can be either hardware or software components) communicate through transmission of **communication items** over abstract channels. The information payload carried by an item is specified by a set of item parameters. Further item attributes describe the direction and duration of item transmission [9]. Items are used to describe communication at different levels of abstraction. So, taking the USB bus as an example, a communication item may describe a complete USB transaction, the data field within this transaction or just a single bit of information in the serial USB bit stream. SystemC^{SV} provides three types of interface items which are distinguished by the level of communication abstraction: **TRANSACTION** items are used to describe multi-directional communication such as transfers with acknowledge phase. **MESSAGE** items describe unidirectional information transfers from a source to a target. Finally, the **PHYMAP** item provides a means for clock-synchronous mapping of transactions and messages and their parameters to states of physical signals which eventually represent the interconnect between hardware modules. Sample specifications for these items are given in Lst. 1, which is a description of the ISO/IEC14443 serial protocol used for wireless communication between a SmartCard transponder and a terminal[1].

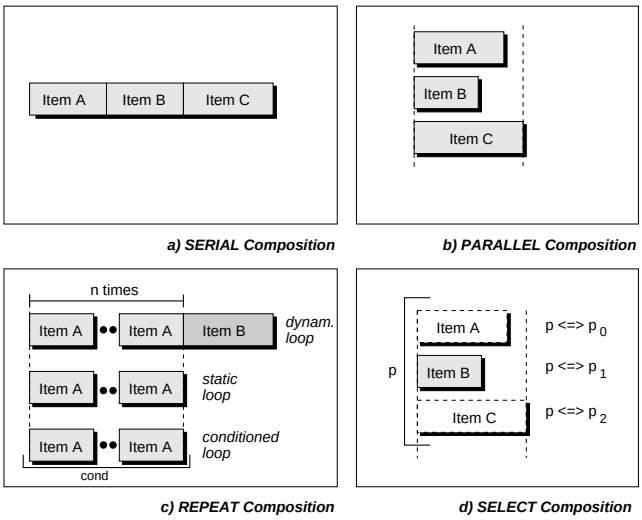


Figure 2: Item Composition Schemes used for Protocol Specification in SystemC^{SV}

For transaction and message items a *transmission protocol* can be defined in form of a **COMPOSITION** of constituting lower level items. For that purpose SystemC^{SV} provides four different composition schemes (Fig. 2), which are used to describe a transmission protocol of an item in terms of an execution schedule of a set of constituting lower level items. The **SERIAL** and **PARALLEL** schemes execute a number of items sequentially or in parallel, respectively. In addition to that there are two parameter-controlled schemes which allow for alternative (**SELECT** scheme)

or repeated item execution (**REPEAT**). The latter two schemes enable the specification of protocols which involve data-dependent branching and looping. Extensive investigations into State-of-the-Art serial communication protocols showed that virtually any non-pipelined, point-to-point or single-master broadcast protocol can be modeled through combination of these four schemes.

```
SV_TRANSACTION(IEC14443Frame) {

  /* parameters */
  SV_Param<sc_uint<8> >  framelen;
  SV_ParamArray<sc_uint<8>, 8> data;

  SV_TRANSACTION_CTOR(IEC14443Frame) {
     SV_FROM << "TX"; SV_TO << "RX";
     ...
     /* protocol declaration */
     SV_COMPOSITION(
        SV_SERIAL(
           SoF_M(),
           SV_REPEAT(z,framelen,
                     Data_M(framelen,data)),
           EOF_M()));
  } };

/* specifications of SoF,Data,EoF left out*/

SV_PHYMAP(PHYBit) {
  SV_Param<bool> b;
  SV_SignalRef<bool> XD;
  SV_PHYMAP_CTOR(PHYBit) {
  ...
  SV_ASSOCIATE(XD-b);
};
```

Listing 1: Specification of the ISO/IEC14443 Protocol

A further SystemC^{SV} language construct used for protocol specification is the SV_SEQBEHAVIOUR statement which is used in composition schemes to embed the execution of sequential behaviours given as a C++ method in between item executions. Such sequential behaviours must execute in zero time and are generally used to modify local state of communication items, so that with the aid of parameter-controlled composition schemes which evaluate the item state the execution of items can be made dependent on previous executions of this or other items. Example applications for embedded sequential behaviours are CRC computations or automatic bit stuffing inside bit transfer messages. In fact, the class of protocols which can be described with this approach is a a superset of the class of protocols which can be specified with regular expressions. In particular all protocols which can be expressed as context-free grammars recognized by a stack automaton of stack size 1 can be modeled.

It has to be pointed out that SystemC^{SV} protocol specifications are purely declarative specifications and, in contrast to behavioural specifications, are not directly executable. Instead, it is the task of a corresponding simulation or synthesis algorithm to derive in the elaboration phase two *distinct, executable behaviours* from such a protocol specification:

Definition 1. The **Decompositional Behaviour** defines how a communication item is decomposed over time into a set of constituting lower level items according to its protocol. It furthermore defines how the set of item parameters are mapped to the formal parameters of these lower level items. For synthesis, it specifies the behaviour of the *transaction producer* controller.

Definition 2. The **Compositional Behaviour** corresponds to the *inverse* decompositional behaviour and defines how a set of constituting lower level items are assembled over time according to the protocol in order to generate a higher level item. It furthermore defines how the information payload contained in the parameters of the assembled item is reconstructed from the parameter values of the constituting items. For synthesis, it specifies the behaviour of the *transaction consumer* controller.

Modules which communicate using items such as IEC14443 Frames would then be specified in SystemC^{SV} as follows:

```
SC_MODULE(Transponder) {
   SV_InterfacePort<IEC14443> RX;
   SV_InterfacePort<IEC14443> TX;

   void send_thread() {
     TX.sv_send(TX->IEC14443Frame(32,data));
   }
   void receive_thread() {
     RX.sv_receive(RX->IEC14443Frame(len,data));
   }
}
```

Listing 2: Module Specification with Item Communication

3. CONTROLLER SYNTHESIS

3.1 Overview

Fig. 3 visualizes the steps required to automatically transform a SystemC specification containing SystemC^{SV} protocol specifications into a model which can be further synthesized with e.g. the Synopsys CoCentric Compiler. The input to our synthesis algorithm is a complete SystemC system model, consisting of two or more SystemC modules which communicate using one or more SystemC^{SV} interface items. The item transmission protocols are specified in terms of a SystemC^{SV} protocol description. Module and interface descriptions are parsed and transformed into an intermediate representation. For the protocol description a protocol flow analysis is then performed which results in a protocol flow graph (PFG). In the next step both transaction producer and consumer are generated from this PFG in form of extended FSM models. These models replace the abstract channel interfaces in the communicating modules. The last step is the generation of synthesizable SystemC code for the communicating modules which have the descriptions of the protocol controllers included and which are connected by physical wires and signal level interfaces.

3.2 Generation of Protocol Flow Graphs

In the protocol analysis step, a protocol flow graph (PFG) is constructed from a SystemC^{SV} item protocol description.

Definition 3. A transaction protocol flow graph is a directed cyclic graph defined by the tuple

$$PFG := < S, T, P, C, I, F >$$

where S denotes the set of protocol states with $I \in S$ being the initial state and $F \in S$ the final state, $T \subseteq S \times S$ the set of state transitions, P the set of transaction parameters and C the set of signals which constitute the physical communication channel.

PFG transitions $t \in T$ are labeled with composition and decomposition guards [C/D Cond], composition and decomposition actions [C/D Action] and signal associations χ, denoted by the symbol <=>. The latter describe an unambiguous mapping of a parameter $p \in P$ or a constant value to a channel signal $c \in C$. Transition guards

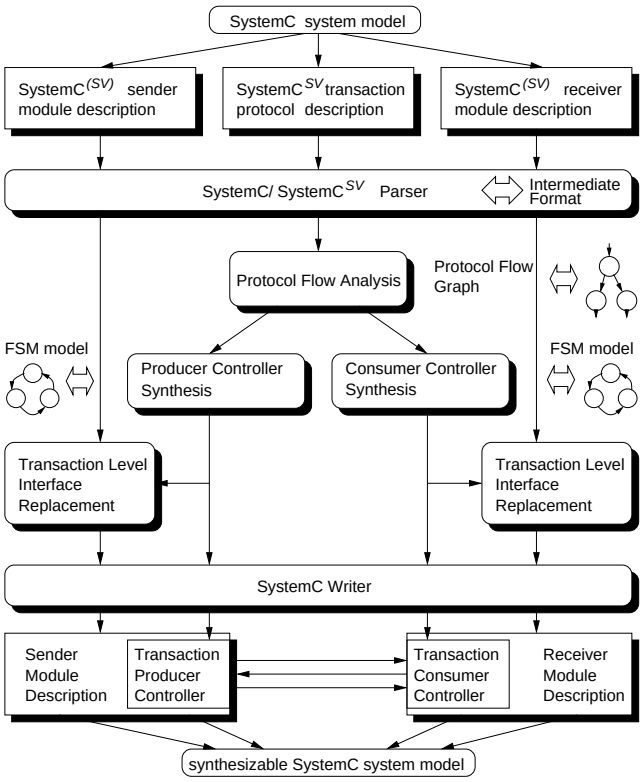
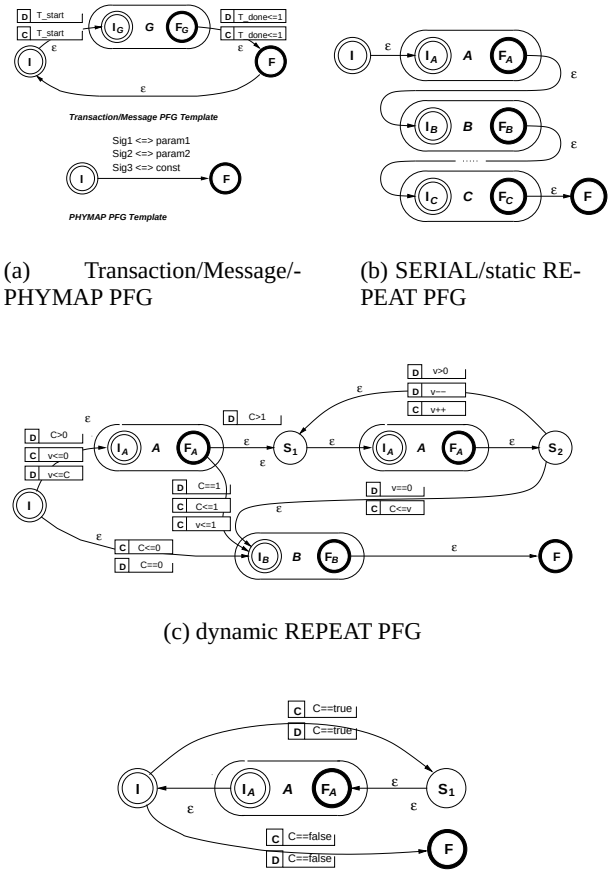


Figure 3: Transformation of a SystemC^{SV} model into a synthesizable SystemC model



(a) Transaction/Message/-PHYMAP PFG

(b) SERIAL/static REPEAT PFG

(c) dynamic REPEAT PFG

(d) WHILE..REPEAT PFG

(e) SELECT PFG (I)

(f) SELECT PFG (II)

Figure 4: Protocol Flow Graph Templates used for PFG construction

condition the transitions between graph states. Conditions can be described in terms of values of parameters or state variables. Transition actions describe actions to be performed when a transition is executed. Such actions include the modification of local state variables or the execution of embedded sequential behaviours. If a transition has multiple actions attached, the order of execution is not determined. Guards and actions are classified into composition specific, denoted by *C*, and decomposition specific ones, denoted by *D*, which accounts for the fact that a PFG represents both compositional and decompositional behaviour of a transaction protocol. For PFG generation from a SystemC^{SV} transaction protocol specification we have chosen a template-based construction approach. For each SystemC^{SV} composition scheme as well as for each item type, a corresponding PFG template is stored in a template library. The contents of this library is shown in Fig. 4. For the **SELECT** scheme two different templates exist. Which one is chosen depends on the selection parameter being a reversible parameter (e.g. during composition the parameter value is established depending on the item matched by the SELECT scheme) or being a constant (during composition a certain item is expected to be matched). PFG generation is done simply through appropriate recursive nesting of these templates, starting with the transaction/message template. This recursive nesting process completes with the insertion of PHYMAP item templates containing the channel signal associations. Care must be taken when transaction item protocols involve a turnaround in the direction of information flow (e.g. handshaking protocol). This is indicated by the fact that lower level items differ in their `FROM,TO` direction attributes (see Lst. 1). In this case, for all items which differ in their logical transmission direction from the direction of the first item in the transaction composition, in the further PFG construction the *C* and *D* labels in the guards and actions must be swapped for this item instance. Finally, when PHYMAP items are inserted, all signal associations are labeled with *C* if such swapping has been performed for the instancing higher level item. Otherwise they are labeled with *D*. In order to be able to generate PFG's also for non-deterministic protocols (specified through dynamic REPEAT or SELECT compositions), PFG templates are modeled using ε-transitions which corresponds to the Thompson construction of NDFA-based pattern matchers [3]. In our case, ε-transitions describe PFG state transitions which are executed in zero time while non-ε transitions in the PFG take exactly one clock cycle to execute. In Fig. 5 an example PFG is shown for IEC14443 frame protocol which was given in Lst. 1.

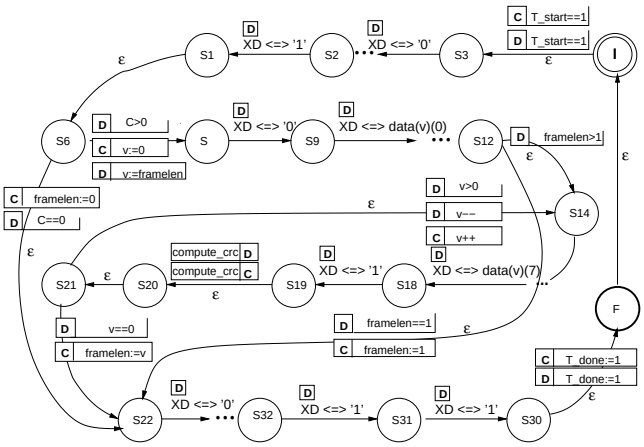


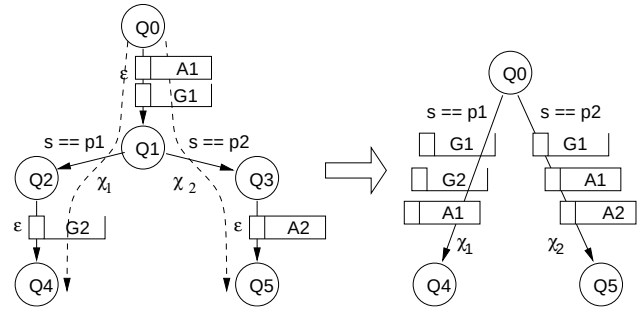
Figure 5: PFG for the IEC14443 Frame Protocol

3.3 Controller Generation from PFG

After successful generation of the PFG from a SystemC^{SV} protocol description, the two distinct controllers representing transaction producer and transaction consumer have to be synthesized from the PFG. The first step is to generate two distinct finite automata for producer FA_D and consumer FA_C. FA_D is generated in the following way: (FA_C is obtained in an analog fashion with C and D labels swapped)

- erase all PFG composition guards and actions (labeled C) from the PFG
- transform signal associations labeled with D into a channel signal assignment
- transform signal associations labeled with C into a transition condition.

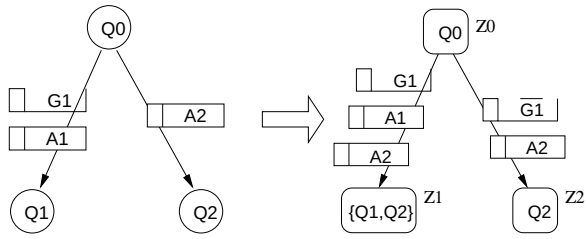
The resulting automaton graphs still contain ε-transitions which must not exist in the final controller RTL description in order to be synthesizable into hardware. Thus, in the next step a transformation is applied to the two automaton graphs which generates equivalent graphs without ε-transitions. A corresponding algorithm used in DFA construction for lexicographical scanners would compute the ε-closure of a state, e.g. the set of all states reachable from this state by ε transitions and then simply re-target non-ε transitions ending in this state to all states in the ε-closure. However, since in our case ε-transitions may have guards and actions attached which need to be handled appropriately, a different approach is chosen. The idea is to traverse the FA graph from each state having outgoing ε-transitions in order to find all paths of the form $Q_S \xrightarrow{\varepsilon} Q_A \xrightarrow{\chi} Q_B \xrightarrow{\varepsilon} Q_T$. For each such path, a new transition $Q_S \xrightarrow{\chi} Q_T$ is created. Furthermore all guards and actions along the path are collected and annotated on this transition. After all such paths have been identified, ε-transitions and consequently isolated states are removed from the automaton graph. The result is an equivalent condensed graph as depicted in Fig. 6. The condensed graphs for FA_C,FA_D are in most cases non-deterministic e.g. they contain states which have outgoing transitions to different target states with the same transition conditions. Non-determinism results from the use of the dynamic REPEAT or SELECT composition in protocol descriptions. The construction of a deterministic FSM from the possibly non-deterministic automaton graphs poses no problem since a deterministic FSM can be obtained using the subset construction algorithm [3]. This algorithm computes all clusters of target states reachable from a source state for a certain transition condition. We extended this algorithm for our purposes in order to correctly handle transition guards and reassign actions to the resulting FSM graph. An example for this transformation is shown in Fig. 7. The non-deterministic FA has two transitions leaving State $Q0$, where one transition is conditioned with guard G1 and the other is unconditioned. Furthermore are actions A1, A2 attached to these transitions. In case the condition of guard G1 evaluates to true, both states $Q1$ and $Q2$ are reachable from $Q0$, resulting in the state subset cluster $Z1 = \{Q0, Q1\}$. When this transition is taken, both A1 and A2 must be executed which are consequently attached to the transition $Z0 \longrightarrow Z1$. If G1 evaluates to false, the only reachable state is $Q2$, thus $Z2 = Q2$ and A2 must be executed which is annotated to transition $Z0 \longrightarrow Z2$.



(a) FA graph with labeled ε edges

(b) condensed FA graph without ε edges

Figure 6: Transformation of FA graph with ε transitions into a condensed FA without ε transitions



(a) non-deterministic FA graph

(b) equivalent deterministic FSM graph

Figure 7: Transformation of non-deterministic FA into deterministic FSM

3.4 Controller Integration

The last step in the synthesis process is the integration of the generated controller models into the models of the communicating modules. In this step, also the transaction level interfaces of these modules, represented by the `sv_InterfacePort` objects (see Lst. 2), are replaced by a signal level interface with a set of physical pins which are then interconnected between the modules. (see also Fig 1). The `sv_send(),sv_receive()` methods used for communication in the SystemC system specification are replaced by a conditional loop containing a single SystemC `wait()` state-

Table 1: Synthesis Results for various Protocol Specifications (Area in Virtex Slices, Clock Frequency in MHz)

Protocol	COSYNE								Protocol Compiler							
	Producer Controller				Consumer Controller				Producer Controller				Consumer Controller			
	Stats	Tran	f_{max}	A	Stats	Tran	f_{max}	A	Stats	Tran	f_{max}	A	Stats	Tran	f_{max}	A
IEC 14443	43	46	85.3	94	77	131	78.5	126	49	116	83.6	91	256	410	77.2	134
ATM	56	62	85.3	52	9	25	63.8	56	56	62	85.9	51	9	22	65.7	54
CAN	452	1022	58.2	271	421	1217	51.7	334	512	1132	50.3	258	476	1342	48.2	321

ment. This loop tests the T_done signal of the generated controllers and blocks the execution of the current process until the corresponding protocol controller which now performs the functionality of the former send() or receive() statement has either sent or received a transaction.

4. EXPERIMENTAL RESULTS

The proposed protocol synthesis algorithm has been implemented with about 25000 lines of C++ code in a tool called **COSYNE** (**Co**ntroller **Syn**thesis **E**nvironment) and has been applied to synthesis of hardware controllers for transaction producer/consumer pairs for three different protocols of sufficient complexity. The synthesized protocol examples include the IEC14443 SmartCard communication protocol as well as popular peripheral device interconnect protocols such as CAN. Furthermore was an ATM protocol specification synthesized as an example for a byte-serial protocol. For the ATM example, the protocol compiler models available as demos in the Synopsys distribution were used and the corresponding SystemC^{SV} protocol description has been developed on their basis. Finally, for the CAN example controllers were generated from a protocol description which defines the four basic CAN messages (Data/Remote/Overload/Errorframe).

Table 1 shows the results of the controller synthesis with COSYNE in terms of the number of controller states and transitions in order to give an impression of the complexity of the generated RTL controller models. Furthermore are area and clock frequency measures shown which were obtained through further logic synthesis of the controller models for a XILINX Virtex V100CS144-4 using Synopsys FPGA II compiler. For that purpose, the SystemC controller models generated by COSYNE had to be manually converted to equivalent VHDL RTL models since SystemC synthesis tools were not available yet to the authors. Functional equivalence of the VHDL models resulting from COSYNE and from Protocol Compiler, respectively, was verified through pairwise simulation of a COSYNE generated producer controller in combination with a Protocol Compiler generated consumer controller and vice versa. For comparison purposes, the synthesis results obtained for the same protocols using Synopsys Protocol Compiler are listed. In order to get comparable results, Protocol Compiler options were set to generation of single process controller architectures with binary min-encoded state machines. The results suggest that the controllers synthesized with COSYNE are in terms of performance and area consumption comparable to those generated with protocol compiler. In some cases (IEC14443 and CAN producers) even a smaller number of states and transitions were obtained with COSYNE, resulting in performance and area improvements of the controller implementations.

5. CONCLUSIONS AND FUTURE WORK

We presented a novel design methodology for communication controller hardware based on synthesis from declarative protocol descriptions. A formalism for protocol specification was presented as integral part of the system modeling language SystemC which enabled the specification, verification and synthesis of protocols in the context of the system description. Furthermore we presented a synthesis algorithm which generates from a single SystemC^{SV} protocol specification all interacting transaction producer/consumer controllers. The approach could be especially interesting for the design of future *Networks on Chips* which are protocol-dominated designs and integrate transaction producer and consumer of a protocol onto a single chip. The feasibility of our methodology was shown by means of various protocol examples which were captured in SystemC^{SV} and synthesized into RTL controller models using the COSYNE tool. Further work will address the problem of modeling and synthesis of pipelined protocols such as AMBA as well as protocols which involve split transactions. Other work will address the extension of the proposed methodology to the synthesis of interfaces between software and HW/SW components.

6. REFERENCES

[1] Fujitsu FME GmbH, Dreicheich/Buchschlag. *BabyFace2 Chip Specification*, 1999.

[2] D. Gajski, J. Zhu, R. Doemer, A. Gerstlauer, and S. Zhao. *SpecC: Specification Language and Methodology*. Kluwer Academic Publishers, Boston/Dordrecht/London, 2000.

[3] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, MA, 1986.

[4] J.Oberg, A. Kumar, and A. Hemani. Grammar-based hardware synthesis of data communication protocols. In *Proceedings of the 9th International Symposium on System Synthesis*, La Jolla, CA, November 1996.

[5] J. Oberg, A. Kumar, and A. Hemani. Grammar-based Hardware Synthesis from Port Size Independent Specifications. In *IEEE Transactions on VLSI*, volume Vol. 8, pages pp. 184–194, April 2000.

[6] P. Probert and K. Salesh. Synthesis of communication protocols: survey and assessment. In *IEEE Transactions on Computers, Special Issue on Protocol Engineering*, volume Vol. 40, pages pp. 468–476, 1991.

[7] J. Rowson and A. Sangiovanni-Vincentelli. Interface-based design. In *Proceedings of the 34th Design Automation Conference*, Anaheim, CA, 1997.

[8] A. Seawright. *Grammar-based Specifications and Synthesis for Synchronous Digital Hardware Design*. PhD thesis, University of California, Santa Barbara, June 1994.

[9] R. Siegmund and D. Mueller. SystemC-SV: Extension of SystemC for Mixed Multi Level Communication Modeling and Interface-based System Design. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE 2001)*, Munich, Germany, 2001.

[10] SYNOPSYS Inc. *Protocol Compiler User Manual*, 1998.

[11] Synopsys Inc., CoWare Inc., Frontier Design, Inc., http://www.systemc.org. *SystemC User's Guide*, 2000.