AN APPROACH TO THE SYNTHESIS OF HW AND SW IN CODESIGN

Vincenza Carchiolo, Michele Malgeri, Giuseppe Mangioni
Istituto di Informatica e telecomunicazioni - Universita' di Catania
Viale Andrea Doria, 6 - I95125 Catania (Italy)
E-mail {car, mm, gmangioni} @ iit.unict.it

Abstract

The main aim of codesign is to be able to design a whole system without excessive preliminary constraints on the mapping or partitioning of the hardware and software parts. At present, given the availability of CAD tools and hardware devices, the sector which seems to offer most prospects of codesign methodology application is that of embedded systems. This paper presents a codesign approach based on the formal technique called TTL. It shows how the synthesis of both the HW and SW modules described by TTL into RTL or C respectively can be performed thanks to a semantic based translation.

1 Introduction

Embedded systems are often used in life critical situations, where reliability and safety are more important criteria than performance. For this reason we believe that the design approach should be based on the use of a formal model to describe the behaviour of the system before a decision on its implementation is taken (see also [1][2][3]).

In this paper we propose a codesign methodology suitable for control-dominated systems. In order to achieve the final partitioning it is necessary to define the processor, hardware components and handshake technique - generally referred to as the target architecture. The methodology proposed currently refers to an architecture including a single general-purpose processor and a few application-specific hardware components, a single bus master software component and a single-level memory hierarchy.

As the methodology proposed intends to cover the development of the whole system, that is, from the specifications in terms of both time and behavior to implementation of its components (the software components by using a programming language and the hardware ones by synthesis) certain choices have to be made, especially that of the technique used to describe the system.

*This work is partially supported by MURST project
Metodologie e Strumenti di Progetto per Sistemi Distribuiti e Paralleli

The issues this paper deals with are the way in which the specification is synthesized into both hardware and software parts. Section 2 discusses the approach to codesign and the steps making the methodology. Section 3 presents the approach to translation of the system specification into RTL and C modules and outlines the scheduling methodology. Finally a brief overview of the current approaches to codesign and the conclusions are presented in Section 4.

2 Methodology Overview

This section outlines the methodology proposed. This methodology aims to provide the developer with a common framework and useful tools to validate the results.

Fig. 1 schematically shows the steps by which methodology is composed. The first step is the description of the behaviour of the system being developed. The tool used for this purpose is the TTL (Templated T-LOTOS) [4] language, a language which has a formal syntax and semantics (FDT), derived from T-LOTOS [5]. T-LOTOS is a language which extends LOTOS [6] adding the capability to describe time explicitly. The properties of TTL (such as modularity, a high level of abstraction, formal semantics, etc.) make it suitable to handle control dominated systems. These are systems which do not require a lot of input signal processing as opposed to data-dominated systems.

The second step in the proposed methodology is the refinement of the TTL description of the system. During this step the specification of the system, which has been developed in the previous step, is concretized through a series of intermediate phases. During these phases each component of the system is decomposed into elementary ones. Throughout all these steps the correctness of the refinement operations performed on the system is guaranteed thanks to the formal base of the source language which also allows mathematical proofs of correctness.

The third step is the -composition of the specification into tasks. They represent elementary blocks which will be used during the subsequent step (partitioning). The decomposition of the specification
does not rely on the target architecture which will be used to implement the system.

Between decomposition and partitioning there is pre-clustering whose purpose is to reduce the complexity of partitioning. This is obtained by grouping the tasks produced by the decomposition into clusters in order to minimize a parameter which takes into account the data-flow among the tasks (this parameter is called coupling-degree) [7].

Then each cluster is translated into both hardware and software. The algorithm used translates the TTL source description into C and a Register Transfer Level language.

Thereafter the implementation cost (mainly the area and delay cost) of each cluster is calculated for both the hardware and software realization. These values are used as parameters for the cost function of the system minimization of which gives the final partitioning.

Finally, after the partitioning, the software-hardware interfaces are generated together with the operating system.

This paper shows the steps dealing with the translation of the clusters into C and RTL. Implementation of the operating system is also shown. A description of the steps with which this paper does not deal can be found in [8]

3 Translation

The algorithms implementing the clusters in hardware and software use an intermediate structure. This structure, called behavioural graph represents the behaviour of each process composing the specification of the system. The behavioural graph is similar to a syntactical tree; it contains all the information needed to perform the translation. For instance, we introduced the connection node which allows us to link two behaviour expressions without duplicating the states which could take place in trivial translation of the system description. Fig. 3 shows the graph of the process exml given in Fig. 2.

```
process exml[t2, t3]: noexit :=
t2 ! i; exit
\[
\]
t3 ? g; int; i; stop >> [g \geq 0] >> i; t2 ! i;
exml[t2, t3]
endproc
```

Figure 2: Specification of exml

As can be seen in the figure each behavioural graph related to a process has one starting node and one ending node. The nodes subsequent to the node labelled 0 (starting node) represent the initial events offered by process exml. Depending on the events which take place, the corresponding branch is fired. All branches which successfully end (denoted by exit) must enable the behavioural expression which follows the >> operator. This fact is represented in the graph by the connection node which links the branches coming from the node labelled exit. The nodes following the connection node represent the starting point of the subsequent behavioural expression.

Figure 3: Behavioural Graph

3.1 TTL to RTL

In this section we will deal with the translation of the TTL specification into RTL. In order to reach
this goal we will introduce the rules to map the basic TTL operator onto a hardware device described by an RTL specification.

The synthesis of the hardware device is based on a direct semantic translation from a formal TTL specification into an RTL one using automatic tools which guarantee the correspondence between the two models. In this way the requisites of the high-level specification are also present in the RTL specification obtained.

This approach is extremely simple and direct. The fact that we have only presented examples for the basic constructs is not a limit, in that the other TTL constructs can be semantically derived from the basic ones [9]. It is of particular interest as it makes a realistic evaluation of the (monetary) costs of the implementation and delay of the hardware circuit possible. This is very important in designing mixed hardware/software systems where knowledge of the costs is fundamental for the partitioning process.

Translation of the complex TTL synchronization into RTL requires the use of several signals to guarantee the semantic correctness of the translation. So each synchronization operation (event) is associated with three signals: one for the exchange of the data itself, and two others to manage the synchronization (a ready and an acknowledgement signal). The need for two signals for synchronization is due to the fact that communication in TTL is a rendez-vous between events.

In this case of one-to-one synchronization, with a value exchange, irrespective of the value actually exchanged, two gates are involved in the synchronization, one of which offers a value (condition expressed by the symbol "$!\$"), while the other accepts a value (expressed by "$?\$").

Let us assume we have two processes, T and R, which respectively offer ($g!v$) and are able to accept a value $v$ through a gate $g?v$.

Schematically, translation of the event $g!v$ can be represented as in Figure 4 (a). Figure 4 (b) is a scheme of the event $g?v$.

![Figure 4: Scheme of basic interaction events](image)

The signals $in_k$ and $out_k$ represent the signals enabling execution of the block $k$ and terminating execution of the block $k$ (which coincides with the signal enabling execution of the block $k + 1$). The signal $g_n$ has no function in this specific case but is necessary to translate the choice operator. In this phase the signal is introduced because it allows us to make a description of the event which is valid in any situation.

Translation of block $i$ into hardware is represented in Figure 5.

```
x  : if ( not g_4_i; g_4_j ) goto(z; z + 1)
x + 1: g_4_k = 1
x + 2: if ( not g_n; g_n ) goto(z; z + 1)
x + 3: g_n = v_r
```

Figure 5: Translation of Transmitter

As can be seen from Figure 5, the transmitter waits for the receiver to be available for synchronization, after which it acknowledges the synchronization and exchanges the value (if any).

The behaviour of the receiver complements that of the transmitter (see Fig. 6). A complete RTL description and a circuit scheme equivalent to the RTL descriptions can be found in [10]. In Figure 5 $v_r$ represents the variable containing the value to be transferred, which in RTL is equivalent to a register. Likewise, in Figure 6 $v_r$ is the register which, following synchronization, will contain the value exchanged.

```
y  : g_4_k:if (not g_4_k; g_4_k) goto(y; y + 1)
y + 1: g_k = 1
y + 2: v_r := g_v
```

Figure 6: Translation of Receiver

According to the translation scheme used, the transmitter is translated into 4 RTL steps. The number of steps is strictly proportional to the complexity of the equivalent circuit as each is implemented by a flip-flop.

The action prefix operator is a pure form of synchronization, i.e. without an exchange of values between the gates involved and can be seen as a particular case of one-to-one synchronization without an exchange of values.

In one-to-many synchronization a transmitter synchronizes with several receivers, which all have to be available for the event. The RTL coding of the receivers remains unchanged with respect to the coding in Fig. 6. Coding of the transmitter is similar to that of the previous case, the only exception being that this time the transmitter has to ascertain, before sending the ack signal, that all the receivers are available for synchronization.

The choice operator allows branches to be introduced into the specification. The branch to be taken is chosen according to the type of event oc-
The state array stores the current state of each process. Signal is an array whose dimension is \( Number\ of\ processes \times Number\ of\ events\). If element \( i,j \) of signal is set to one it means that the process \( i \) is ready to synchronize on the event \( j \). Lastly the mask array stores the event which each process is able to emit/receive.

### 3.3 Operating System

The operating system activates tasks ready to run, in a round-robin fashion. A task is ready when it is not available to engage an event. In this case the task will be unlocked when all the other tasks participating in the same event, are able to engage that event. The Operating System manages the hardware tasks through registers for exchanging signals. The signals are directly managed by the Operating System without using any interrupt, but using a polling policy [10].

### 4 Related Work and Conclusions

In this section we will examine the different approaches that can be found in literature to solve each aspect of codesign. Different techniques have been proposed to tackle the specification of hardware and software; in the following we will sketch the characteristics of some of them. Esterel [11] is a synchronous language based on FSM. The synchronous hypothesis states that time is described as a sequence of instants, between which no action can take place. This hypothesis permits the system to be modelled using only a single FSM exhibiting a totally predictable behaviour. Unfortunately the resulting FSM is generally fairly large, thus making it difficult to specify systems with a large amount of concurrency. Another technique belonging to FSM is CFSM [12]. It is an extension of the FSM model tailored for codesign.

Among the other languages used for co-specification we can cite two examples: C*, the entry language for COSYMA [13], which extends ANSI C with delays, tasks and task communication, HardwareC [14] which can be translated into a flow graph, and VERILOG[3]. In the methodology introduced in this paper the specification language used is TTL. It is derived from T-LOTOS, an FDT based on the CCS and CSP process algebras.

A key problem in codesign methodologies is validation of the model of the system being developed. Simulation is still the main tool used for this purpose and consists of comparing the model against a set of specifications. Many methods have been proposed in literature; they differ in their method of coupling hardware and software components. For example, in [15] a single custom simulator is used for both hardware and software, whereas another approach proposes using a software process run-
ning on a host computer loosely connected with a hardware simulator [16]. TTL aims to perform verification on the specification. It also allows the congruence between two successive refinement steps to be checked without using a simulation approach.

Several solutions to the partitioning problem are proposed in literature. Some use a graph model to represent the operations performed by devices and associate a cost to them [13]. Others perform the partitioning together with the implementation of the scheduling algorithm as, for instance, in [17] where the specification is made with a hardware description language and synthesis tools are used to estimate the costs. The basic idea of performing scheduling and partitioning together is to minimize the response time.

Our methodology divides the partitioning stage into two steps. The first (preclustering) is based only on the properties of the system and aims to reduce the complexity of problems. This is obtained by a simple algorithm whose complexity is very low especially compared with that of the mapping algorithm. The second step groups the remaining clusters and maps them onto the target architecture. The strategy used to reduce the complexity of mapping is based on minimization of the interaction among clusters. Finally some problems dealing with mapping have been discussed, including the choice of the scheduling algorithms needed to allow hardware and software modules to coexist.

References


