A Consistent Design Methodology for Configurable HW/SW-Interfaces in Embedded Systems

Embedded Systems Design

Stefan Ihmor, Markus Visarius, Wolfram Hardt
{ihmor | visi | hardt}@upb.de
University of Paderborn, Warburger Str.100, D-33098

Abstract: In the embedded systems domain predictability, fault tolerance and high-speed data transmission rates are key challenges for the interface design. Multiple tasks and channels communicate through different protocols with each other. In this paper we present a consistent design approach for configurable real-time interfaces. An interface design methodology therefore should regard the relationship between distributed tasks, channels and supported protocols within a HW/SW Codesign scenario. The model dependent parameters are important information for this process and are represented in a formal UML-based way. As result of the design process an interface-block (IFB) is generated which considers all these parameters. A complex embedded system in the context of a case study implements a collision avoidance algorithm for two interacting robots. It demonstrates the usability of this concept for an implementation of HW/SW-interfaces with respect to the real-time restrictions..

Key words: dynamic reconfigurable interface design and modeling, HW/SW Interfaces

1. INTRODUCTION

In this paper we present a new modeling approach for configurable real-time interfaces. The interface design is driven by high needs in data transmission techniques and rising computing performance in terms of operations per second in the embedded system domain. Much effort has been spent in improving these aspects but the affected interfaces also have to be discussed to avoid weak points in system-architecture. Especially embedded systems (ES) include several kinds of interfaces, mostly in form of HW/HW-

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35599-3_29

B. Kleinjohann et al. (eds.), *Design and Analysis of Distributed Embedded Systems*

or HW/SW-interfaces. In many cases modern ES are distributed systems, need to be fault tolerant, have to cope with several kinds of media and hard real-time restrictions of multiple tasks have to be met. As a result the large amount of different interfaces leads to the need for an automated design process for reconfigurable real-time interfaces. This process requires a consistent and integrated design methodology, from the specification to the implementation.

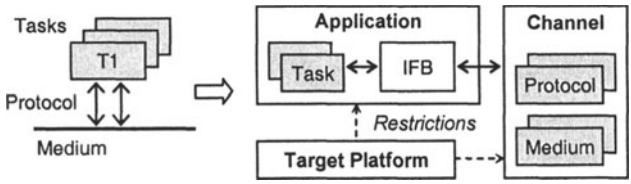

Figure 1. Overview of interface design relevant aspects

The main components of our interface design view can be structured into four parts: task, medium, protocol and target platform dependencies (see Figure 1). This is a refinement of the very general view symbolized on the left side of Figure 1. The traditional parts protocol and medium are merged to a logical block named "Channel". Functional or spatial dependent tasks are joined together with an interface-block (IFB) to the "Application". All relationships between the application, the channel and the target platform with respect to the design process can be modeled in UML. The structure and the dependencies between these items are represented by class diagrams. Timing aspects are modeled in form of sequence charts and activity diagrams. The charts and diagrams have to include sufficient information to specify the IFB that implements the reconfigurable thus dynamic interface. Tasks as well as the IFB may be implemented in hardware or software.

Every communication between two distributed tasks is processed by the dedicated IFB, which handles the medium access on the one hand and controls the task activation on the other hand. This means that the IFB is an active interface. To guarantee hard real-time restrictions the whole system uses a static pre-processed scheduling for medium access. The parameters which are needed to specify the IFB are separated from the UML models of the different tasks, the media and the target platform.

Reconfiguration of the IFB means to change the scheduling strategy implemented in the control unit. The executed IFB behavior from the task activation to the medium access and the internal IFB control may be varied. Our approach also supports the usage of full custom protocols. These are defined by the appropriate task specification and are only restricted by the data transfer rate of the used medium through its physical properties. As these full custom protocols need to be adaptable to the target platform the

platform dependent parameters have to be considered in the design process as well. A design methodology from an abstract formal specification model (UML) to a dedicated hardware description language (HDL) or a programming language (SystemC) is considered to realize this approach.

2. APPLICATION RELEVANCE

Several techniques for interface design like SLIF or OCB have been presented in the past. But all these ideas are limited to a partial viewpoint on the design process. Using an IFB-model includes several advantages: Different channels, protocols and tasks are supported and all implementation relevant parameters can be extracted from one abstract modeling language. The requirements are modeled in the well-known and easy-to-understand UML [2, 3, 7] technique in form of class diagrams, sequence charts and statecharts. Therefore it is important to define an exact semantic meaning for the specification language in addition to a non-ambiguous syntax description. An automaton based approach in form of statecharts offers a compact methodology to specify the behavior of the IFB. Afterwards the transformation to the Timing Dependency Graph (TDG) [5] can be done to include timing constraints within the automaton representation. So the time constraints can be modeled high-level, here by UML sequence charts and then transferred to the TDG. The VHDL code which is generated by a transformation of the IFB is the input for the succeeding synthesis process. Special restrictions of the synthesis process have to be considered by the VHDL code generation as well. This can be done by modeling these parameters within the platform dependent diagram. Eventually the restrictions and properties of used synthesis tools can be counted to the platform parameters itself.

Full custom protocols which have been mentioned in the introduction are a further advantage of the IFB model and can be calculated automatically from the specification. Although SDL is one of the common ways for modeling protocols and interfaces [11, 13], this paper considers a UML conform approach to possess a consistent description formalism to specify all necessary information needed for the design process [3, 4, 12].

3. MODELING OF INTERFACE PARAMETERS

There are three possible fields of application for an IFB-model: For medium access of tasks, task adaptation and protocol conversion. The first case covers a consistent bus access of several tasks to multiple channels. If

you want to include one task to an existing design the IFB works as an adapter (see Figure 2) and in case there are only channels the IFB acts as a protocol-gateway.

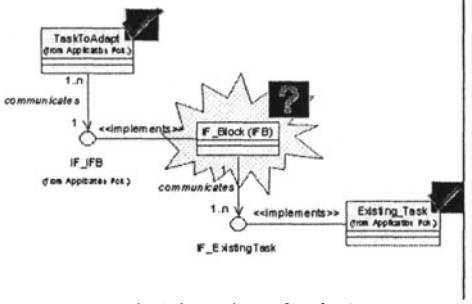

Figure 2. Adaptation of tasks into an existing context

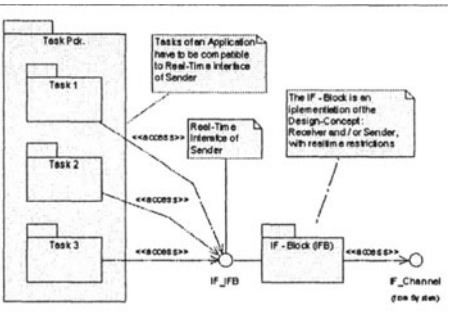

Figure 3. Task-IFB-Level as packet diagram

A set of UML packet diagrams is used to describe the hierarchical content of the interface structure. An excerpt of the packet structure is shown in Figure 3. You can see the "Task-IFB-Level", where several tasks access the "IF_IFB" of the IFB. Also the connection from the IFB to the channel interface "IF_Channel" is symbolized in the lower right of the figure.

As UML was chosen as the modeling language a class diagram is the adequate form to represent structures and dependencies of the required IFB parameters. Figure 4 shows a task's diagram which can either be a HW-block or a SW-program. "Task" is a class that communicates with the abstract class "IF_IFB" which is implemented within the IFB and defines its interface. The parameters in the class model are extensible and have to be adjusted to the appropriate model.

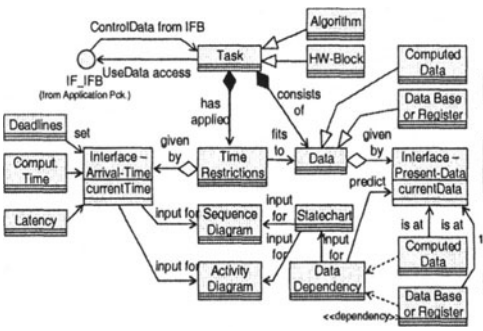

Figure 4. Task modeled as UML class diagram

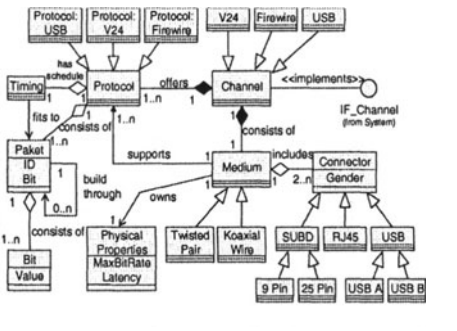

Figure 5. Channel modeled as UML class diagram

Figure 5 represents a channel which includes the medium as well as the protocol aspect. The class "Channel" that implements the abstract class "IF_Channel" defines the access point for the IFB. Some specific

realizations of denoted classes are modeled using inheritance for the class channel, protocol and medium with the connectors.

The third class diagram represents the target platform dependent parameters for SW or HW (see Figure 6). Main class of the diagram is "Target Platform". It is associated to classes "HW-Target" and "SW-Target". "Technical Requirements" and "Synthesis Process" are connected to the HW-development by compositions. Prohibited and recommended kind of source code can be modeled in the "Synthesis Process".

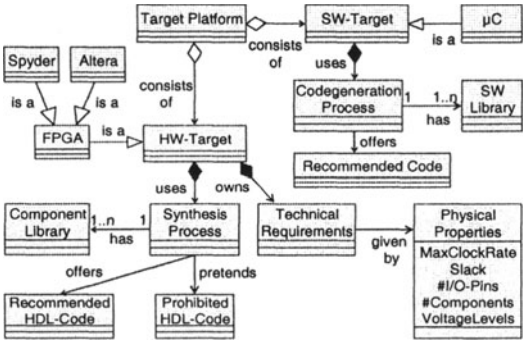

Figure 6. Target platform modeled as UML class diagram

Another important aspect for HW-target code generation is the used component library. A SW-target consists of a "Code Generation Process" that offers "Recommended Code" and uses a specific "SW Library".

The next step is to explain the IFB-model [6, 8] which is parameterized by the information given in the diagrams. The succeeding chapter deals with the representation of an IFB by statecharts and an overview of the functional concept. Statecharts are extended automata and seem to be good candidates for an automated code generation for FPGAs [5, 10]. Sequence charts are used to specify the timing aspect of the included statecharts and the control unit within the IFB.

4. PROTOTYPING

An FPGA has been chosen for the hardware implementation of the IFB. A micro-controller can be taken for a software implementation as well. The medium, e.g. twisted pair cable is connected to the prototyping platform. Tasks can be implemented separated from the platform or integrated as well. The protocol as an abstract element is implemented within the IFB which consist of a control unit and a generator pipeline – the sequence- and the protocol generator. The job of the control unit is to activate tasks and handle

the generator pipe. The sequence generator communicates with the tasks to fetch and collect their data and prepare it for the protocol unit. Further on predefined pieces of data can automatically be inserted here which allows complex features instantiated in sequence generation. That is very useful if a receiving task (actor) expects a dedicated data stream and the sending task, like simple sensor, only generates fragments of the data content available. In the protocol generator these sequences are adapted to the appropriate protocol. Another feature of the protocol generator is the medium access which is guarded by the control unit.

An automata based approach has been selected to implement the IFB-model because the synthesis process is supported very well for this design methodology [10]. Therefore Figure 7 represents the resulting structure of the communicating automata. On top the "Control Unit" controls the sequence generator (SG), the protocol generator (PG) and as here included the task (Task). The control unit requires feedback information in form of status signals. To realize dynamic interfaces different behavior is implemented by sub-automata of the SG, PG and where necessary of the task.

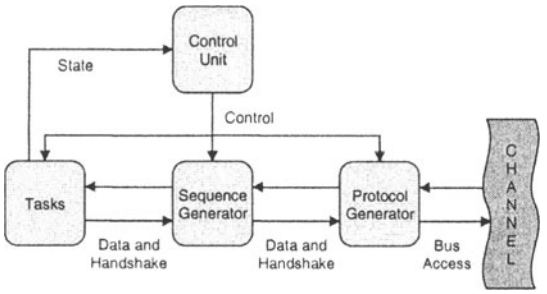

Figure 7. Automata based way to implement an IFB

The sub-automata of SG and PG realize the "generator modes". To execute different behavior the control unit activates dedicated sub-automata through the SG and PG and the task itself, respectively. The communication flow between task and channel is routed through the SG- and PG-FSM using data signals and handshake wires. Figure 8 represents the dependencies as a UML class diagram.

As the main class of the diagram, "Control Unit" implements the abstract interface-class "IFB_Control". On the left "IF_IFB", the interface to the tasks is implemented by the sequence generator and on the right the protocol generator accesses the interface of class "Channel". The "Protocol" defined in the "Channel" class diagram is a template for the possible instances of the generator pipe. To the sequence generator a "data template" is offered in this way, which is used for data preparation for the protocol generator. The

requirements for the creation of the supported protocols within the PG are extracted from the "protocol template".

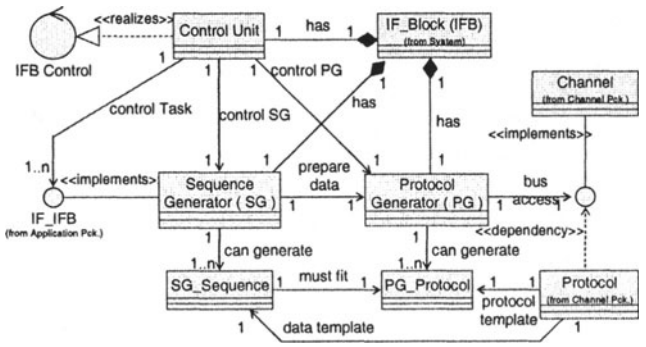

Figure 8. UML class diagram of an IFB

The illustrated kind of implementation is for the access of tasks to different media. In case of adaptation or protocol conversion the IFB fulfills nearly the same job which differs only in its meaning.

The design process, in combination with the IFB is a step towards an automated code generation, because it offers a unified and capable solution for the explained problems. It will be necessary to refine the demonstrated class models to accomplish an entire high-level and automated design process. For some constructs it's obvious how a transformation could look like, e.g. state charts to VHDL automata. But therefore it's necessary to define the syntax and semantic meaning for a unique mapping.

5. EXPERIMENTAL RESULTS

A scenario with two integrated interacting industry robots demonstrates that the concept can be realized for practice relevant application (see Figure 9). One of the robots (R1) is manually controlled by a joystick, whereas robot R2 executes a predefined job. In case of an potential collision R2 has to swerve to avoid any contact between R1 and R2.

To fulfill this task, a TTP/A communication between the controlling FPGAs, using a serial V24 connection, has been established [9] in form of an IFB. On this way the motor data of robot R1 is submitted from FPGA1 to FPGA2. The FPGAs are connected to the robots also using a serial V24 channel which is realized by an IFB as well. The serial V24 (RS232) interface is a common interface in the embedded systems domain and can be used to set up a real-time communication. The design methodology of an IFB (see Figure 10) is illustrated by the V24 interface as an example.

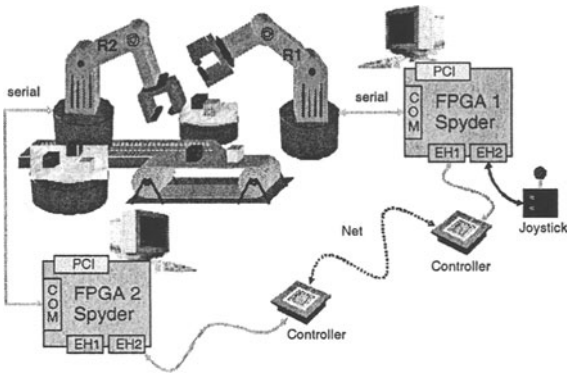

Figure 9. UML Scenario of the interacting robots and the different communication channels

Figure 10 and 11 show the specification of the used V24 serial interface for manual robot control. The aspects "Task, Control Unit, Sequence Generator and Protocol Generator" are represented as statecharts. The task differentiates between an active joystick control and a homing state.

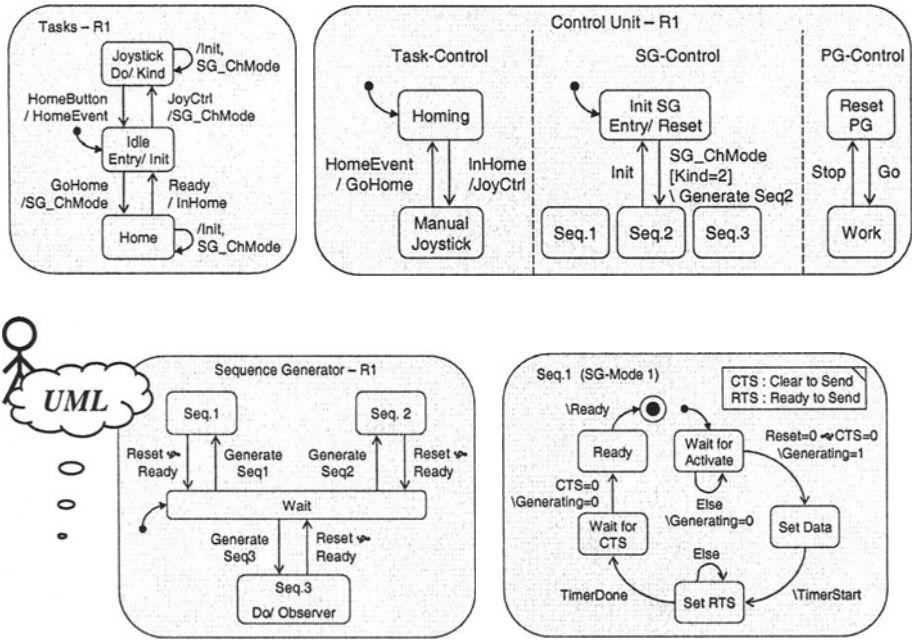

Figure 10. Specification of the Task, CU and SG as statechart

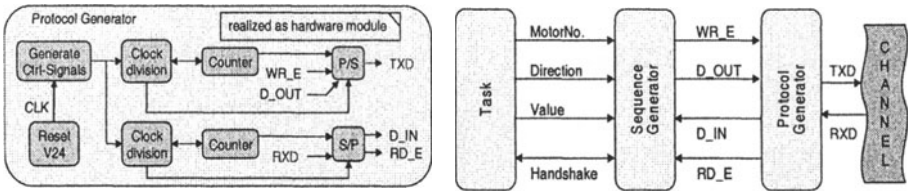

Figure 11. Specification of the PG and the communication flow

The resulting data stream is prepared in the SG and transmitted by the PG that implements a standard V24 UART HW block. A control unit is used to control the other units. The right side in Figure 11 characterizes the control- and data flow within the generator pipe. The submitted data is received by robot R1 which is directly connected to the channel. To get the joystick data to the "Task"-FSM another IFB is realized that serves as physical interface.

6. CONCLUSION AND FUTURE WORK

In this paper we have presented a new concept for reconfigurable real-time interfaces. It has been shown that the concept offers a consistent, level-comprehensive, easy-to-extend and well-defined approach for interface design. In this way interfaces with widespread properties can be consistently modeled and implemented by an IFB.

The demonstrator's serial interface has been used to illustrate that our approach covers all demanded requirements in the application domain, from modeling to the implementation of real-time interfaces. Furthermore our approach is dedicated to automatic code generation based on abstract modeling techniques.

Future work will concentrate on the aspect of code generation towards HW (VHDL or Verilog) and SW (C). This includes aspects of graph transformation from formal abstract modeling languages as well as the computation of full custom protocols. These can be deployed beside standard protocols to fit best to the demands of the comprised tasks.

In addition, validation of timing restrictions has to be considered. Therefore a schedule has to be computed from all the information derived from the task requirements. On this basis automatic evaluation can decide whether the required system performance meets the offered bandwidth or not. In case of success a schedule for the distributed IFB nodes has to be computed. This will be important input for the HW/SW repartitioning. The mentioned extensions will advance our approach to automatic design of HW/SW real-time interfaces.

REFERENCES

[1] A. Burns, *Real Time Systems and Programming Languages*, Addison-Wesley, Harlow [u.a.], third edition, Ada 95, real time Java and real time POSIX., 2001

[2] G. C. Buttazzo, *Hard Real Time Computing Systems: Predictable Scheduling Algorithms and Applications*, Kluwer, Boston [u.a.], third edition, 2000

[3] B. P. Douglas, *Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks and Patterns*, Addison-Wesley, Reading Massachusetts [u.a.], first edition, 2000

[4] B. P. Douglas, *Real-time: Developing Efficient Objects for Embedded Systems*, Addison-Wesley, Reading Massachusetts [u.a.], third edition, 1998

[5] W. Hardt, T. Lehmann, M. Visarius, *Towards a Design Methodology Capturing Interface Synthesis*, University Paderborn, Computer Science Department, 2000

[6] W. Hardt, M.Visarius, S. Ihmor, Rapid Prototyping of Real-Time Interfaces, FPL – Field Programmable Logic conference in Belfast, 2001

[7] G. Hassan, *Designing Concurrent, Distributed, and Real-Time Applications with UML*, Addison-Wesley, Boston [u.a.], 2000

[8] S. Ihmor, *Entwurf von Echtzeitschnittstellen am Beispiel interagierender Roboter*, Master Thesis, University of Paderborn, 2001

[9] H. Kopez, *Principles for Distributed Embedded Applications*, Kluwer Academic Publ., Boston [u.a.], fourth edition, 2001

[10] D. J. Smith, *HDL Chip Design, A Practical Guide For Designing, Synthesizing and Simulating ASICs and FPGAs using VHDL or Verilog*, Doone Publications, Madison, AL, USA, seventh edition, 2000

[11] J. Teich, *Digitale Hardware / Software-Systeme, Synthese und Optimierung*, Springer-Verlag, Berlin Heidelberg, 1997

[12] K. Tindell, *Analysis of Hard Real-Time Communications*, Real-Time Systems, pp. 147-171, 1995

[13] P. Verissimo, *Real-Time Communication*, Addison Wesley – ACM Press, Reading, Mass., 1993