

DESIGN OF COMPLEX EMBEDDED SYSTEMS BASED ON DIFFERENT PETRI-NET INTERPRETATIONS

W. Fengler, A. Karg
Technical University of Ilmenau
Faculty of Computer Science and Automation
PF 100 565, 98684 Ilmenau, Germany
e-mail: wfengler;ankarg@theoinf.tu-ilmenau.de

Keywords: Object-orientation, Petri Nets, Embedded System, Discrete simulation, Verification

Abstract

A holistic approach for the design of complex embedded systems is described. Its design flow combines hardware and software components. All flow phases are based upon different Petri Net interpretations which use the same theoretical fundamentals. This allows to simulate and verify system parts as well as the whole system. A special simulation method allows to include transition actions. Verification of safety and time properties becomes possible. This is important because embedded systems are often a major part of safety critical real-time applications. To hide the difficulties of different high-level Petri Net interpretations and to use the advantages of modern software technologies an object-oriented design method called Concurrent Object Nets is introduced. This method allows engineers to design and implement safety-critical embedded systems without knowing the Petri Net theory.

1. Introduction

An embedded system consists of specific hard- and software which forms a component of some larger system and which is expected to work properly without human intervention [FOD98].

In general it is designed for a single specific application and carries out well defined functions within the complete system. Embedded systems can contain standard microprocessors and microcontrollers as well as special hard- and software adapted for the particular application. For high performance applications embedded systems are often realised distributed with more than one processor and a communication system.

The correctness and the compliance with real-time requirements is often very important for embedded systems, because they control systems in safety and time critical areas (e.g. vehicles).

The continuing drop in prices and the increase in efficiency for hardware components like microprocessors have made accessible new types of application for embedded systems and, therefore, stimulated research on this subject. Also, it is remarkable, that embedded system design requires methods, which differ basically from the ones already known.

Embedded system design means to combine standard hardware circuits with individual application specific hard- and software components. New methods and algorithms are necessary to design such complex embedded systems with reasonable costs and as quick as possible. Such new methods and algorithms will allow the designer to predict design and production costs in a very early phase and to obtain error free results rapidly.

Although there are differences between several embedded systems depending on the particular field of application, there are some characteristics in their design flows they have in common. Figure 1 shows the main phases of such design flows.

The specification is the first step of the design flow. It must be distinguished between functional and non-functional requirements. Design faults, made in these early steps of the system development will be hard and often expensive to adjust in later phases. An important research topic is the development of formal specification techniques which allow to describe functional and safety properties. The well known Petri Nets can be enhanced to fulfil these requirements, whereas the original Petri Nets, known from C.A. Petri [PET62] are not sufficient to satisfy them completely.

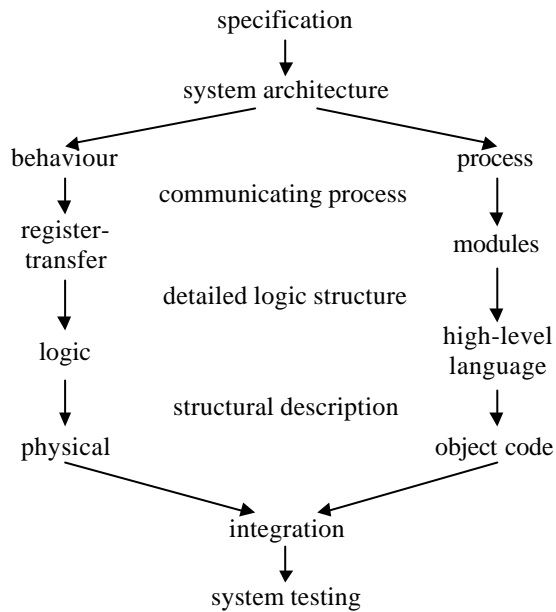


Figure 1: Embedded System Design Flow

For a complete system specification, one single technique is not adequate, but different Petri Net interpretations can be used to describe all aspects of a system. That gives the possibility of a special treatment of each system view, but all interpretations refer to the same fundamental theoretical basis. To make the system description more comfortable for people who are not familiar with Petri Nets, it might be helpful to hide them by an additional specification layer.

The definition of an initial system architecture follows. This phase is marked by partitioning the system in hard- and software. The following phases are separated into a hard- and software design flow. Figure 1 shows a common approach for this separation.

During the integration phase the complete software and the real destination hardware components are combined. The main focus in this phase lies on the check of the error free co-operation of hard- and software.

Only if hard- and software co-operate, the validation can ensure, that the system specification has been kept. Due to the complex structure of embedded systems, the check by simulation is very important.

2. Object-Oriented System- and Software Design

For the design and verification of distributed embedded applications a new object-oriented design method considering specific requirements of engineers has been

developed. This method is called Concurrent Object Nets (CON) [NDF98],[NFB97]. It provides a graphical representation for the semantics of classes and their dynamic behaviour as well as for the interaction of their objects. UML [UML97] diagrams are used to describe static inheritance relationships. An internal hidden Petri Net representation of the objects supplies the designer with simulation and verification facilities.

Within the CON method, three different meta classes, which can be distinguished by their internal behaviour, exist:

- An abstract ON class (AONC) has no specific behaviour. AONCs like all other ON classes have a port interface and an optional list of time and logical constraints. There are four kinds of ports: synchronous send (SSP) and receive ports (SRP) an asynchronous send (ASP) and receive ports (ARP). During the design flow all instances of AONCs will be substituted by instances with refined behaviour.
- A hierarchical ON class (HONC) encapsulates an Object Net which consists of several ON instances connected by a number of message links. These aggregated instances may also be either of AONCs, HONCs or EONCs.
- An elementary ON class (EONC) encapsulates a hierarchical extended state machine (SM) [HAR87] with time delays and time constraints (min./max. time for software actions). The SM is the fundamental building block for the ON design model. This leads to a conflict-free communication between different ONs. The only kind of conflict, which may exist within an ON, is within one SM and can be found easily. The SM is defined by states, an initial state, actions and a set of state changes. In this case, actions are software functions, which are assigned to a state change. The state change of the SM is forced either by incoming messages or by the termination of special states.

The design flow in the Concurrent Object Net method is based on the principle of refinement through inheritance. The software designer starts to discuss the problem with the customer. As a result of this discussion the designer creates a first Object Net specification. This specification formalises the non-formal requirement of the customer. It includes instances of AONCs. During the refinement (through inheritance) process the designer overwrites the instances from the first specification with instances which are more specific.

The mechanism of refinement through inheritance is very restrictive in the CON method. Two fundamental

inheritance rules (interface inheritance rule and constraint inheritance rule) define which refinements are allowed.

If the designers refinements follow these inheritance rules the properties of the first specification will be preserved. An automatic (formalised) check determines whether the designer has violated the rules.

At the end of the refinement process a specification with all details will be available. This specification can be simulated graphically. The last step in the design flow transforms the fully refined Object Net specification into a specification which can be implemented on a distributed platform.

For simulation and implementation every hierarchical ON will be automatically transformed into a flat ON including only EON instances coupled by message links. In the second step these flat ONs will be transformed into corresponding high-level Petri Nets to

be run in the integrated back-end simulator.

This back-end Petri Net simulator is coupled with the script language Tcl (Tool Command Language) [OUS94]. This allows to include the program code of the actions into the simulation without any compilation. The modular design of our design tool-set with its back-end simulator allows also offline or remote simulation e.g. on a UNIX workstation cluster or other platforms.

3. Object-oriented Design Example

To demonstrate our design method, we will discuss a small application example. This example is a special approach to drill holes into circuit boards. To place the drill at a specific x-y-position, the controller has to calculate two angles (phi1 and phi2) from the x/y-coordinates before (see Figure 2).

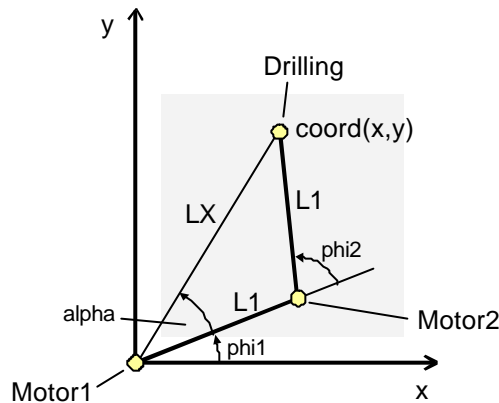


Figure 2: The Drilling Example

To calculate these angles, the following formulas have to be applied:

$$\begin{aligned}
 LX &= \text{hypot}(\text{coord.x}, \text{coord.y}) \\
 \cos(\alpha) &= LX / 2 * L1 \\
 \cos(\alpha + \text{phi1}) &= \text{coord.x} / LX \\
 \text{phi2} &= 2 * \alpha
 \end{aligned}$$

These angles will be used to control two motors (*Motor1* and *Motor2*). In our approach the motor control and the angle calculation will be done distributed. Therefore, both motors work independently. As soon as both motors have reached their final position, the drilling will be triggered. After drilling is finished, the drilling cycle for the next hole starts.

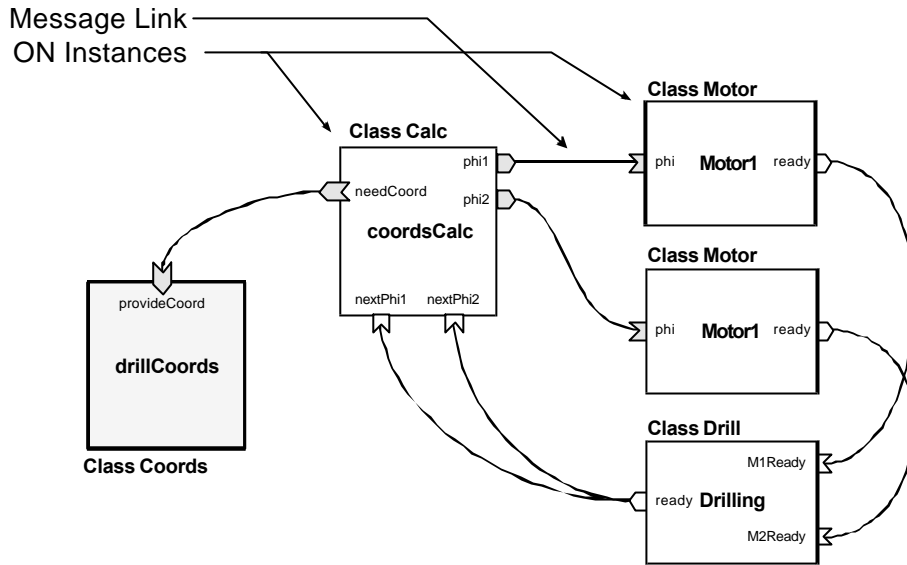


Figure 3: The ON Specification for the Drilling Example

As you can see in Figure 3 the ONS contains two Object Net Instances (ONI) of the class *Motor* (*Motor1* and *Motor2*), one ONI of the class *Drill* (*Drilling*), one of the class *Calc* (*coordsCalc*) and a last instance of the class *Coords* (*drillCoords*). This last ONI is for simulation purposes only. The ONI *coordsCalc* is the central part of the ONS. Its class *Calc* (see Figure 4) is a HONC, which contains five ONIs of five different EONCs. The class *Phi1Buffer*, which is shown in Figure 5, expects at two ARPs two float values for the

final angle calculation. These two values will be combined. After the reception of a trigger message from port *sendPhi1* the result will be sent through the ASP *phi1*. After this send action, the class returns to its idle state and waits for two new values. These two ports, *phi1* and *sendPhi1*, are visible in the surrounding HONC *Calc*.

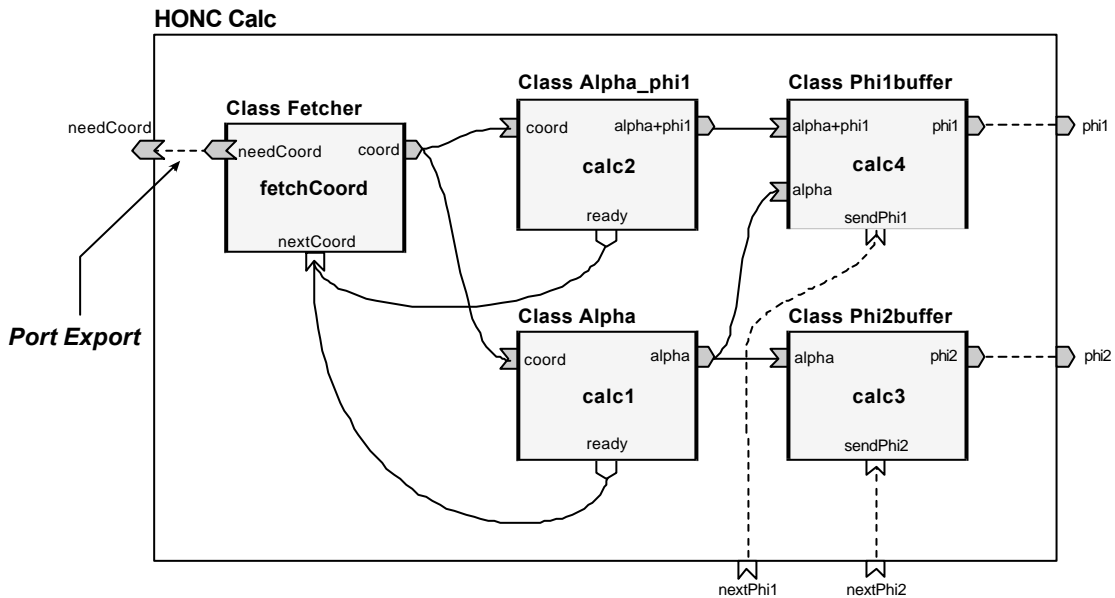


Figure 4: The Hierarchical Object Net Class (HONC) *Calc*

Messages coming from port *phi1* will be directed through a message link to the ONI *Motor1*. The message with a new angle position is the trigger for the motor to enter this new position. *Motor2* works analogous to *Motor1* with a different angle. The ONI *drilling* waits for two messages, one from each motor, to start the drilling action. The completion of the drilling is the signal for the ONI *coordsCalc* to send

new values to the two motors. During motor movement and drilling, the ONI *coordsCalc* fetches new x/y-coordinates from a special instance, which provides test coordinates for simulation purposes only. In case of implementation, this instance will be replaced by the real drilling control environment. To give an idea about simple EONCs Figure 5 (EONCs) shows two EONCs of the example.

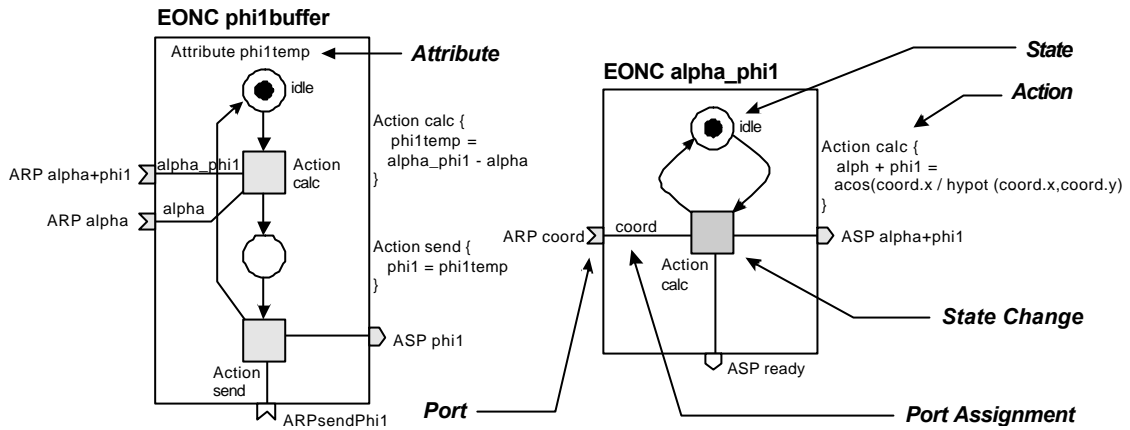


Figure 5: Two EONCs of the Example

The result of the transformation into the corresponding HLPN is shown exemplarily (see Figure 6). It is only a sub-section of the complete Petri Net. It shows the ONI *fetchCoord* and its connected message links. These links have been transformed for the case

that every ONI was placed on a separate controller which is communicating with the other ones via time consuming (*delay*-transitions) communication networks (e.g. fieldbuses).

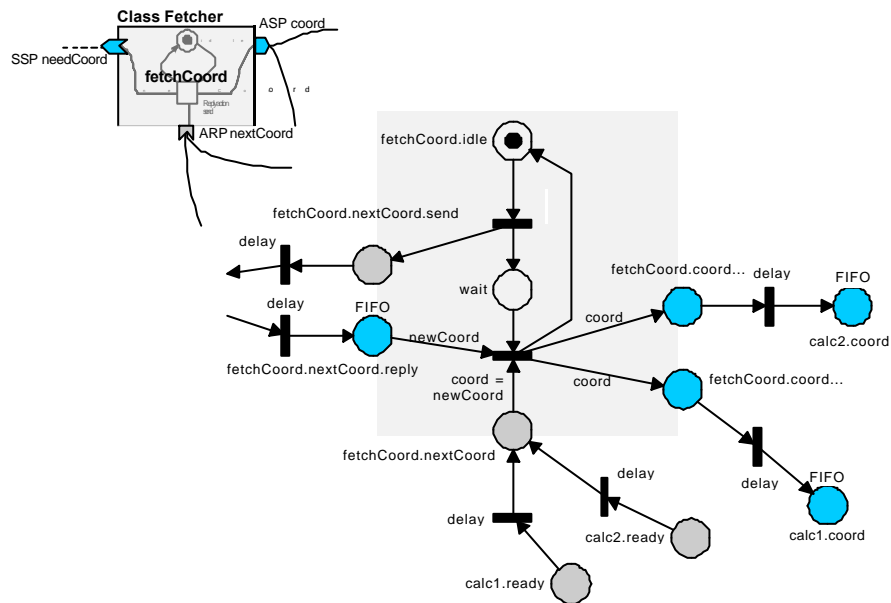


Figure 6: The Corresponding Petri Net

4. Communication Design

For the design of communication protocols, extended high-level Petri Nets (HLPN) [NFE95], [JEN92], [KNO94] are used. These are a compact form of the original Place-Transition Nets with the speciality of individual tokens which can carry additional data values. The advantages of this Petri Net interpretation are:

- Possibility to integrate the control and synchronisation with the description of data manipulation.

- Interactive simulation whereas the results are represented directly in the net.
- The HLPNs can be transformed back into original Place-Transition Nets.
- Therefore, a large number of existing formal analysis methods to verify the model can be used.

The communication protocols which are needed for embedded systems are mostly fieldbuses (e.g. CAN or Profibus), which are specified for the layers 1, 2 and 7 of the ISO/OSI reference model.

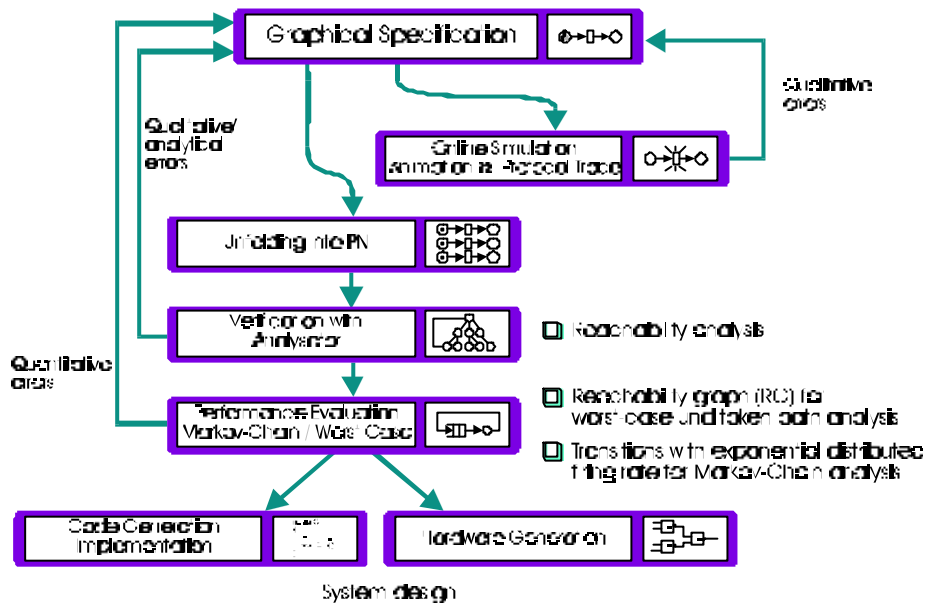


Figure 7: Method for Specification and Design of Communication Protocols based on HLPN

Figure 7 shows the detailed steps from an HLPN to the code of a communication protocol. First, the structure of the system has to be modelled. Then, the HLPN needs to be verified by simulation and afterwards to be unfolded into a normal Petri Net. This Petri Net needs to be verified again, e.g. by reachability analysis. After that, a worst-case analysis can be performed. As soon as all analysis checks have returned a positive result, the code and also the hardware can be generated.

In our ON system design message links between ONIs which are located on distributed controllers connected by a fieldbus system use the services of layers 7 and 2 respectively. The protocol designs described in this chapter replace these communication structures. For a full system simulation and verification the specification of these services have to be integrated.

5. Hardware Design

To design some parts of the embedded system in hardware, Application Specific Integrated Circuits (ASIC's) can be used. The main problem is, to find a way to design systems in general, that means to design the system in a way which makes it independent from a special environment of a special company. The best known way to realise this is to use VHSIC Hardware Description Language (VHDL) [IEEE88], [CAR91]. To make this easier for the designer, Electronic System Design Automation Tools (ESDA) [ESDA97] can be used. They are placed in the hierarchy directly above the design with VHDL (see Figure 8). The main advantage of ESDA is the graphical user interface for design, simulation and verification. Also, error search and redesign can be done in a single environment. This makes work easier and more clear, because it illustrates the system in a better way.

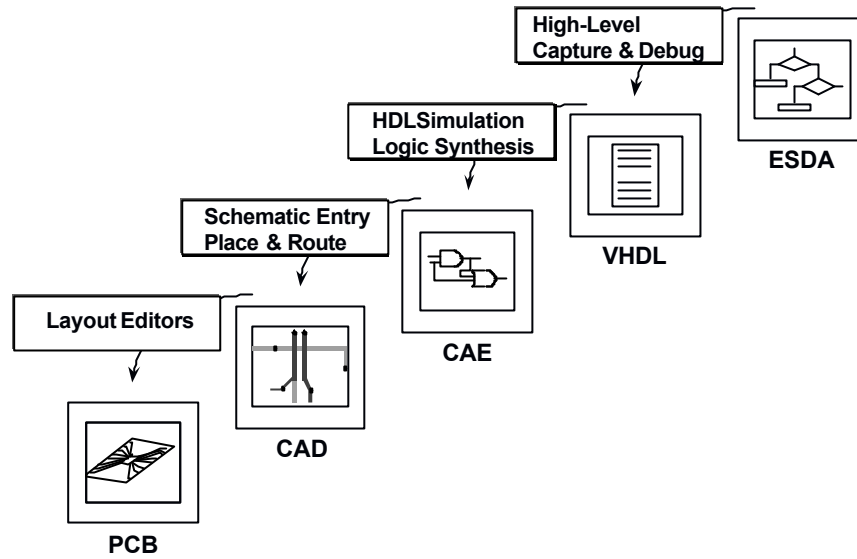


Figure 8: Levels of Hardware Design

The first step is to design the system with formalised graphical models. Those can be state diagrams, flow diagrams, truth tables or similar models. Normally, this is done with an ESDA tool. The next step is the transformation of these graphical models into automatically synthesised VHDL code. In Figure 9 these steps are shown in more detailed. With ESDA it is

possible to design hardware apart from the real system in a general manner. Formally, the specification step was separated from the design flow itself. Now it is situated within this flow and became an important part of it. The disadvantage of this automatic synthesis of VHDL code is the only "good" quality of the code.

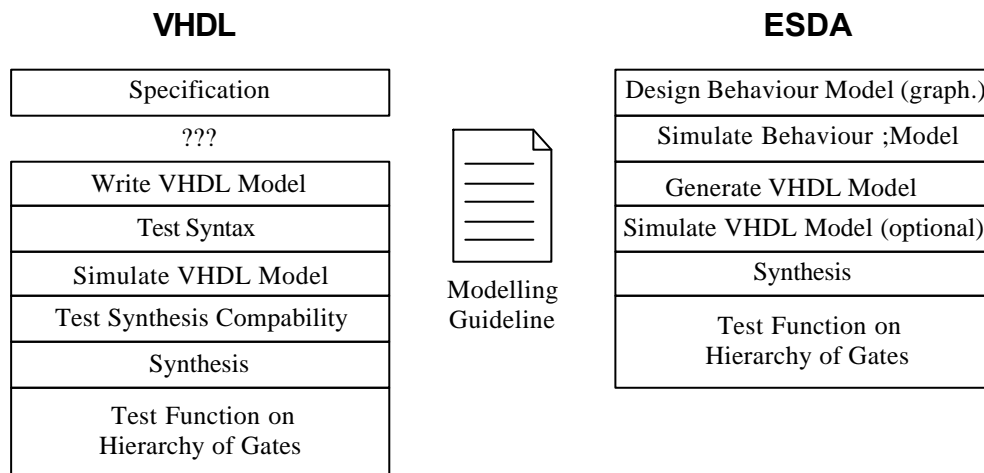


Figure 9: Use of ESDA to Design Simulatable Specifications

Since the ESDA specification consists of several graphical state oriented models, Petri Nets can be used, too [FPA95] [FWAM96]. They give more efficiency in describing parallelism and data flow. It is possible to use different high-level Petri Net interpretations like

hierarchical, coloured and special extended Petri Nets, depending on the system, which is to be designed. With Petri Nets, the designed model can be analysed and verified. On the other side state machines are a special

class of normal Petri Nets and the use of them is included.

In our system design method using Object Nets (ON) the lowest level is given by the elementary ON classes (EONC's, see chapter 2). The control structure of the EONC is defined by a state machine with a port interface. These ports can be transformed into PN structures. Places with capacity one are modelling normal logical signals or triggers, places with higher capacity model buses or registers.

Within hardware classes actions triggered by the firing of transitions are modelled by VHDL functions. It

is necessary for automatic logic synthesis that these VHDL functions are synthesisable. Then, it is possible to generate the hardware structure of Figure 10 using the method of Rokyta. There are elements for computing the switchability of the transitions, solving the conflicts and switch the transitions. If the hardware part consumes or produces external signals asynchronous or synchronous input/output logic blocks are necessary. The VHDL functions of the transitions written by the designer are treated similar to the synchronous I/O logic [ROK97].

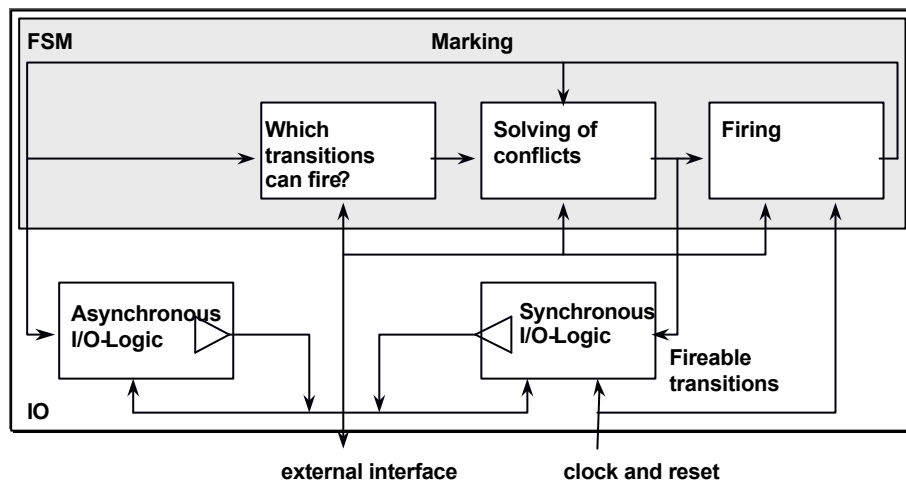


Figure 10: Finite State Machine (FSM) for Realisation of a Petri Net Model

6. Analysis Methods

Embedded systems mostly have higher demands on real-time and security behaviour. Therefore, the problems of validation and verification are more important. The general use of different interpretations of Petri Nets in the design areas described earlier gives us the chance to use the variety of methods and tools existing for this model. We generate an overall model of our design result as follows:

- The application software was designed with the Concurrent Object Net method (see Chapter 2). The elementary Object Net classes are described by state machines extended by net constructions modelling the communication (from the application view). The time behaviour of the actions is given by a minimum and a maximum time or a stochastic average time (distributed exponentially) [MAC87].
- The communication software has an underlying coloured Petri Net model (see Chapter 3). We use it to substitute the more general communication net

structures on the application layer (message links) through the real structures realising the services of the used communication system, in most cases a fieldbus. The time behaviour is given by an average time and, if used, a deterministic protocol by a minimum and maximum time [KNO94].

- Special hardware sections are designed by using high-level Petri Nets on the ESDA level (see Chapter 4). Compared with the software running times and the communication times it is possible to ignore the time consumption of digital hardware actions. If there are some counters, analog or mixed analog-digital sections, it is possible that the time data of these functions have to be taken into account, too.

After generation of the net model of the design it is possible to investigate it by simulation. It can be shown by this method that sub-states of interest are reachable and the transitions between them are correct. It is possible to get statistic values about the time behaviour

(e.g. throughput) by the use of stochastic simulation of deterministic stochastic nets

Using formal analysis as described in [STA90] and implemented in the tool INA [STA97] we get information about reachable markings (sub states) and the paths between them, liveness (dead and life locks), conflicts and the safety of the used asynchronous communication connections. These analysis techniques have been investigated for a long time. A major

problem is that the reachability graph is used for many analyses and it can be very large.

In the last time some work has been done on the field of worst case timing analysis and the results are implemented in the tool INA, too. The following example derived from the design example in chapter 2 shows the problem (see Figure 11). To make it less complicated we use a direct parallel communication without time consumption.

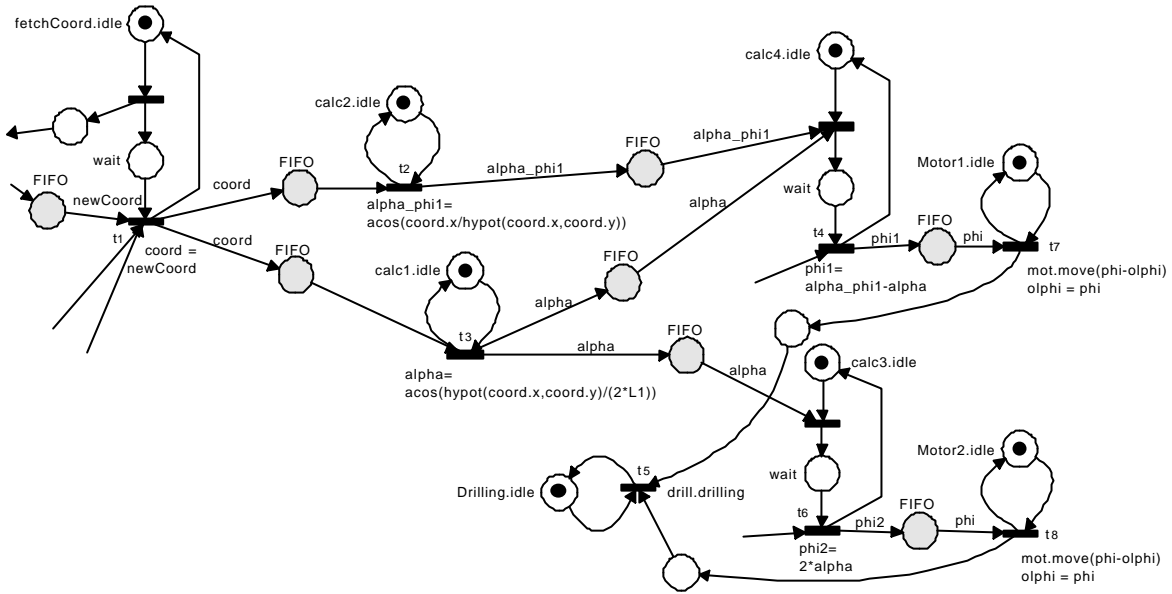


Figure 11: Part of the Corresponding Petri Net (See Example, Chapter 3)

After giving a new coordinate (*newCoord*) there is a co-operation of four calculation and two motor objects (*calc1*, *calc2*, *calc3*, *calc4*, *Motor1*, *Motor2*) necessary before the drilling object can be activated. The worst case time (t_{max}) for this process is of interest for the application. There are the minimum/maximum time intervals ($t_x := [t_{xmin}, t_{xmax}]$) of the actions given (determined from the software sources and the mechanical models) and there are some possible critical ways in the net. The tool calculates the longest one by the given time values. It will not be a sum of maximum values in all cases.

The following formulas show the calculation of the shortest time t_{min} and the worst time t_{max} :

$$t_{bmax} = t_{3max} + t_{6max} + t_{8max}$$

$$t_{bmin} = t_{3min} + t_{6min} + t_{8min}$$

$$t_{amax} = \max(t_{2max}, t_{3max}) + t_{4max} + t_{7max}$$

$$t_{amin} = \max(t_{2min}, t_{3min}) + t_{4min} + t_{7min}$$

$$t_{max} = t_{1max} + \max(t_{amax}, t_{bmax}) + t_{5max}$$

$$t_{min} = t_{1min} + \max(t_{amin}, t_{bmin}) + t_{5min}$$

7. Summary and Conclusions

Complex embedded systems consist of more than one processor and a communication system. They are increasingly often applied to safety and time critical processes. Their design involves system specification, hard- and software design and various validation and verification methods. A combination of object-oriented methods based on the widely accepted Unified Modelling Language (UML) extended with special Object Nets leads to satisfying results in the area of system and software specification. The Concurrent Object Nets combine methods normally used by engineers, like data flow diagrams and state machines

(SM), with object-orientation and the theory of high-level Petri Nets (HLPN). The use of HLPN and SM provides the possibility to integrate design methods for communication protocols as well as for hardware parts. The application of HLPN throughout the entire design process allows to utilise analysis methods of these nets. Extending the HLPNs with time intervals gives access to worst case analysis, thus, verifying the real-time properties of the designed systems. The integration of all the developed methods in a tool system is crucial in order to benefit from their application. With this integration it will be a step towards safer and faster design of embedded systems with adequate expenditure of engineering costs.

References

- [CAR91] Carlson, S.: *Introduction to HDL-based Design Using VHDL*. Synopsys, Inc., 1991
- [ESDA97] *Visual HDL for VHDL on UNIX: Edition 4.0*, Summit Design, Inc., 1997
- [FOD98] Howe, D.: *Free On-line Dictionary of Computing*. 1998: <http://wombat.doc.ic.ac.uk/>
- [FPA95] Fernandes, J. M.; Proeneca, A. J.; Adamski, M. A.: Vhdl Generation from Petri Net Parallel Controller Specification. Proceeding of EURO-VHDL'95, Brighton, GB, 18.-22.9.1995
- [FWAM96] Fengler, W.; Wendt, A.; Adamski, M. A.; Monteiro, J. L. M. P.: Petri Net Based Program Design for Controller Systems. Proceeding of 13th IFAC World Congress, San Francisco, June 30 – July 5 1996
- [HAR87] Harel, D.: Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, Vol. 8, p. 231-274, 1987
- [IEEE88] *IEEE Standard VHDL Language Reference Manual*. The Institute of Electrical and Electronics Engineers, Inc., 1988
- [JEN92] Jensen, K.: *Coloured Petri Nets*. Berlin, Heidelberg, New York, Tokyo: Springer-Verlag 1992
- [KNO94] Knorr, R.: *Specification, Verification, Performance Evaluation and Implementation of Communication Protocols with Hierarchical High-Level Petri Nets*. Ph.D. theses, TU Ilmenau 1994 (in German)
- [MAC87] Marsan, M.A.; Chiola, G.: *On Petri Nets with Deterministic and Exponentially Distributed Firing Times*. *Advances in Petri Nets 1987*, Berlin, Heidelberg, New York, Tokyo: Springer-Verlag 1987, pp. 132-145
- [NDF98] Nützel, J.; Däne, B.; Fengler, W.: *Object Nets for the Design and Verification of Distributed and Embedded Systems*. EHPN'98, Orlando, Florida, USA, 3th April 1998
- [NFB97] Nützel, J.; Fengler, W.; Böhme, T.: *Object-oriented Design Model for Control Systems based on the Petri Net Theory*. EKA'97, May 1997, Braunschweig, Germany (in German)
- [NFE95] Nützel, J.; Fengler, W.: *Analysis and Verification of High-Level-Nets in Combination with Formal Estelle Specification Petri Nets applied to Protocols*. Workshop of the 16th Int. Conf. on Application and Theory of Petri Nets, Torino, Italy, 26 June 1995
- [OUS94] Ousterhout, J. K.: *Tcl and the Tk Toolkit*. Addison-Wesley, 1994
- [PET62] Petri, C.A.: *Communication with Automatas*. Hemel Hempstadt: Prentice-Hall, Inc.
- [ROK97] Rokyta, P.: Synthesis Oriented ASIC Design with Petri Nets. 3rd ITG/GI/GMM-Workshop, February 1997, Chemnitz, Germany (in German)
- [STA90] Starke, P. H.: *Analysis of Petri Net Models*. Stuttgart : Teubner, 1990 (in German)
- [STA97] Starke, P. H.: *Manuals to the Integrated Net Analyser INA*. HU Berlin, 1997. <http://www.informatik.hu-berlin.de/lehrstuehle/automaten/ina/>
- [UML97] Rational Software Corporation: *Unified Modeling Language*. <http://www.rational.com/uml/>