A CoDesign Experience with the MCSE Methodology

J.P. Calvez, D. Isidoro

IRESTE La Chantrerie CP 3003 44087 Nantes cedex 03 France
fax (33).40.68.30.66, email: jcalvez@ireste.fr

Abstract

In this paper, we describe the experimentation of our codesign process to develop a communication system which needs to correctly mix hardware and software parts to satisfy required performance. The system design process is based on the MCSE methodology and we show its usefulness for CoDesign. CoDesign is shown as an enhancement of the implementation specification step of MCSE. System partitioning is the result of an interactive procedure based on performance and cost evaluations. The complete description of the implementation is obtained by transformations of the functional description: C or C++ for the software part, VHDL for the hardware part. The links between the hardware and software parts are also synthesized.

Such a procedure and associated tools lead to efficiently obtain system prototypes in an incremental manner.

1: Introduction

The increase in demand and complexity of systems requiring electronics leads to consider concurrently two technologies which are software and hardware. A suitable solution to a given problem is in this case the result of trade-offs between both technologies which is derived from an analysis of imposed constraints such as performance, manufacturing cost, time-to-market.

Developing such kind of systems is not new, but the process to design them has to be improved to significantly increase the number and variety of designed embedded systems with a notable development cost reduction. The design of such systems according to an integrated development process require to know and to use a complete design methodology useful for heterogeneous systems as well as computer tools to assist designers for the description, evaluation, implementation, realization steps.

CoDesign (Concurrent Design) is defined as the use of a complete top-down design process and related tools (CASHE for Computer-Aided Software and Hardware Engineering) to transform a system specification into an operational product which satisfies required performances, cost, time-to-market and quality constraints [1], [11].

The main problem CoDesign must solve is to determine, from a specification which explains the functional objective to satisfy for a given application, the appropriate hardware architecture and for its programmable hardware part the software to run on it. A diversity of technological solutions exists based on available components.

The CoDesign problem concerns, in the short term, the definition and use of a suitable development process which leads to correctly use existing tools. Following such a design process, new tools will then be conceived.

Codesign problems to be solved also depend on the kind of application and more specifically of the needed hardware architecture. In most cases, a template architecture on which the functional solution has to be mapped is first chosen. It is what we have done in our experimentation. In codesign related works, the mainly used architecture is a master/slave architecture using a conventional microprocessor or microcontroller as the master and an ASIC (FPGA or other) as the slave part to implement the time-constraint functions. The partitioning is often easy and defined interactively. The main problem concerns the synthesis of the interface between hardware and software (Hw/Sw interfaces) [7], [9], [10], [11]. Moreover, in most cases, the functionality is explained by an algorithm (mostly sequential) from which the partitioning is determined according to timing constraints [10].

The approach we have followed for our example is different. First, a master/master architecture has to be used to obtain the best performance for the message transfer protocol. The ASIC shares the memory with the microprocessor, this implies an arbitration of the bus. Synchronizations have to be implemented between the Asic and the microprocessor in both directions. Moreover, the microprocessor is supposed to run other software tasks, therefore, the Hw/Sw interface must use interrupts and not always a polling technique in order to release the microprocessor as often as possible. Second, the functionality is described by a set of sequential cyclic tasks, each of them enough detailed to be completely implemented in hardware or in software (coarse-grain partitioning).

0-8186-6315-4/94 $04.00 © 1994 IEEE

In this paper we describe a codesign experience and show that a system design methodology called MCSE (Méthodologie de Conception des Systèmes Electroniques), we have developed for real-time systems [3] is appropriate for CoDesign. The contents of the paper is as follows. Section 2 briefly describes the MCSE methodology and its conceptual description model. Our illustrative example is given in Section 3, for which the explanation uses the MCSE functional model. This presentation is enough to show the importance of the implementation specification step for CoDesign, step which is detailed in Section 4. Section 5 describes two solutions we have implemented for our example. Section 6 presents tools we are using for our codesign experience. Section 7 describes our future work.

2: Design process and description model of the MCSE methodology

For efficiency and quality reasons, every design must be done according to a methodological process which is based on a hierarchy of description models which go from the preliminary idea to the final product. The design process described hereby and which is relevant to system design is based on the MCSE methodology specifically conceived to design embedded real-time systems [3], and which has been enhanced to support ASIC design [4]. Hereafter we only describe briefly the MCSE methodology. Readers more interested are invited to see [3], [6].

System designers must proceed according to four steps: system specification, functional design, implementation specification, realization. The result, at the end of each step, must be in conformance to an appropriate class of description model.

The purpose of the first specification step is to express a purely external view of the system (WHAT) starting from needs and user requirements.

The purpose of the functional design step is to find an appropriate internal architecture (which explains the HOW) but according to an application-oriented viewpoint. The description based on a functional structure or the behavior of each function has to be technology-independent.

The third step called the implementation specification step, consists in searching firstly for the executive support or hardware architecture and secondly for the way of implementing software functions. Timing constraints and performance are then analyzed to determine the hardware/ software distribution.

The implementation step leads to an operational system. Implementation which includes testing, debugging and validation, is a bottom-up process since it consists of assembling individual parts, bringing out more and more abstract functionalities.

The conceptual model on which MCSE is based. includes two views, each corresponding to a specific aspect of the solution:

- the *functional model* allows to describe a system by a set of interacting functional elements (functional organization dimension or functional structure) and the behavior of each one. Described with a hierarchical and graphical model, functions interact using relations of one of the three types: the shared variable relation which defines a data exchange without temporal dependencies, the synchronization relation which specifies temporal dependency, the message transfer by port which implies a producer/consumer type relation.
- the *executive model* describes the architectural structure based on active components (microprocessors, specific processors, hardware components) and interconnections between them.

These two views, when separately considered, are not sufficient to completely describe a solution. It is necessary to add the mapping between the functional viewpoint and the executive viewpoint by defining an integration or allocation correspondence also called configuration.

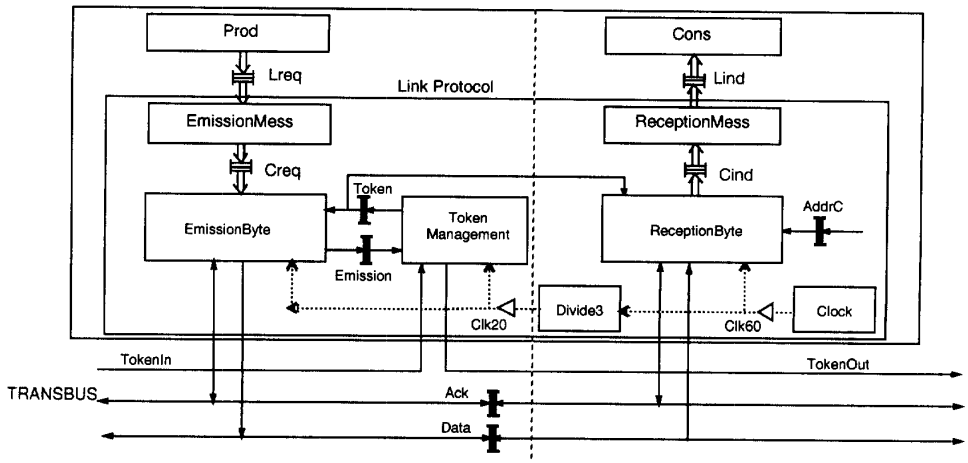
The functional model, located between models appropriate to express specifications and the architectural model, is suitable to describe the internal organization of an object by explaining all necessary functions and couplings between them accordingly to the problem viewpoint. Designing with this method leads to a technology-independent solution. As a matter of fact, with this kind of model, all or a part of the description may be implemented either in software or in hardware. Therefore, this model is interesting as a basis for CoDesign.

3: Description of our example

The chosen example is interesting to illustrate our method and to show the problems related to codesign.

Let us consider a communication system used to send short messages or packets (256 bytes max) between producers and consumers. Producers and consumers are software tasks which are implemented on several similar boards interconnected through a 20 Mbits/s serial bus called Transbus [2]. The bus includes 4 lines: a DATA line as the data medium (bit protocol similar to the transputer link protocol except for acknowledgment), an ACK line for acknowledging each transmitted byte, 2 lines (TokenIn and TokenOut) to control the bus access according to a hardware token ring. The objective is to develop the system corresponding to one board. A performance constraint is considered which can vary between a low performance and the highest performance for message throughput.

To explain the needed functionality, we first describe the solution resulting from the use of the two first steps of the MCSE methodology (specification and functional



-Figure 1 - Functional structure for the communication protocol example.

design steps). Figure 1 describes the internal (functional) structure of the communication board on the 20 Mbits/s serial bus. The way followed to obtain this solution is not the objective of this paper. See [3].

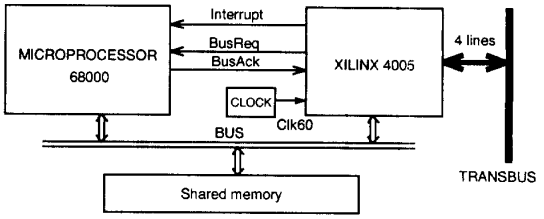
If we analyze this result, first the functional model is a structured model which leads to describe the complete organization of the application in a hierarchical manner. The model is graphical and hierarchical, so the designer can proceed by searching for his internal solution by stepwise refinements. The hierarchical approach is suitable to easily manage the complexity. Functions (or processes) of the model (rectangle) are active functional components which transform input data and produce output values, are easily described by a sequential cyclic algorithm.

The meaning of the solution for our example (Fig 1) is quite easy to understand. The functional structure is a technology-independent solution. It is completely understandable and can be validated in VHDL after the behavior of each function has been described. Graphical tools we have developed [8] are useful to generate a complete VHDL model which is simulatable. Performances can be analyzed by this way. For that we use the simulation technique to evaluate the average time between two bytes.

From this design result, the designer has to select an appropriate executive structure after having defined the partitioning. One can easily imagine that the performance constraints will modify the partitioning. For a low performance level, a minimum of hardware is necessary to convert a byte into a bit stream. It is the same to receive each byte. The remainder can be implemented in software. For a high performance level, EmissionMess and ReceptionMess have also to be implemented in hardware because a byte is sent or received in less than 600 ns. Messages have to be read or written in a memory accessible by producers and consumers (software functions).

For this design problem, we have considered a generic hardware architecture shown in Figure 2. A board has been developed which includes a well-known microprocessor (68000), a Xilinx 4005 FPGA, and a memory shared by both components. The microprocessor can have access to the FPGA and the memory, the FPGA can also have access to the memory and can generate an interrupt to the 68000. The bus arbitration is done by the two signals BusReq and BusAck. This board has been chosen only to be able to prototype the application and to make CoDesign experiences. If industrial constraints (cost, speed, time-to-market,...) are considered, better solutions may be found.

Based on this hardware choice, the CoDesign problem is how to convert the functional description into all necessary parts to obtain through generation and synthesis the FPGA programmation and the software part on the 68000. Our proposed CoDesign process is described in the next section.



-Figure 2 - Generic hardware architecture for the example.

4: The implementation specification step for CoDesign

The aim of this step is to find the most appropriate mapping of the functional description onto a hardware architecture which has to be selected in order to best satisfy performance and cost criteria. This step is decomposed into three phases shown in Figure 3.

The codesign process evolves as follows:

Phase 1: Detailing functional design

- Take account of geographic distribution constraints,
- Addition of physical and man/machine interfaces.

Phase 2: Partitioning

- Decomposition into sub-sets corresponding to software parts and hardware parts based on performance and timing constraints,
- Specification of the executive structure and allocation of functions on components.

Phase 3: Generation, synthesis, evaluation

- Architectural design and synthesis of the hardware part,
- Specification and generation of the software part,
- Synthesis and/or generation of the hardware/software interfaces,
- Behavior and performance evaluation.

Hereafter we succinctly describe each phase and explain its application to the example.

4.1: Detailing functional design

The phase 1 is necessary to transform the functional description which is technology-independent into a description physically compatible with the system environment. The functional description depicted in Figure 1 shows this kind of result. Indeed, the functional structure concerns only one board. The geographic distribution constraint stipulates that producers and consumers are distributed on different boards. As the bus (Transbus specification) is imposed, each board has also to satisfy the physical bus interface. An optimisation is also often done to reduce the number of clock event functions. The result is a complete, detailed and optimized functional description useful for partitioning.

4.2: Partitioning

Phase 2 concerns the search of an appropriate hardware as a support to the detailed functional description which leads to an operational solution satisfying performance, timing constraints, cost, etc. Partitioning of the functional structure is the first task to identify functions which can be implemented in software, and functions which are compulsorily implemented in hardware. With MCSE, we suggest to follow an interactive partitioning procedure driven by the designer who can easily decide for each function the best choice between a hardware or a software implementation according to event, message and function activation frequencies. The executive structure can then be decided: hardware functions will be implemented in an ASIC part (one or more components), software functions will be implemented in one or more microprocessors depending on timing and cost constraints. Necessarily links between processor(s) and ASIC(s) are also specified from corresponding relations of the functional structure.

For our example, the performance of the communication board is expressed by the bus throughput or number of bytes/s transmitted and received. This performance will be determined by the rate at which EmissionMess and ReceptionMess can exchange each byte with EmissionByte and ReceptionByte. We have to consider two cases mentioned in the specification:

- For *low performance*: TokenManagement, EmissionByte, ReceptionByte driven by a 20 Mhz and 60 Mhz clocks, Clock and Divide3 functions are necessarily implemented in hardware to satisfy the 20 Mbits/s rate. It is easy for a designer to determine that these functions are the minimum to implement in hardware. The criterion for each function is its activation frequency or period. All other functions can be implemented in software on the same

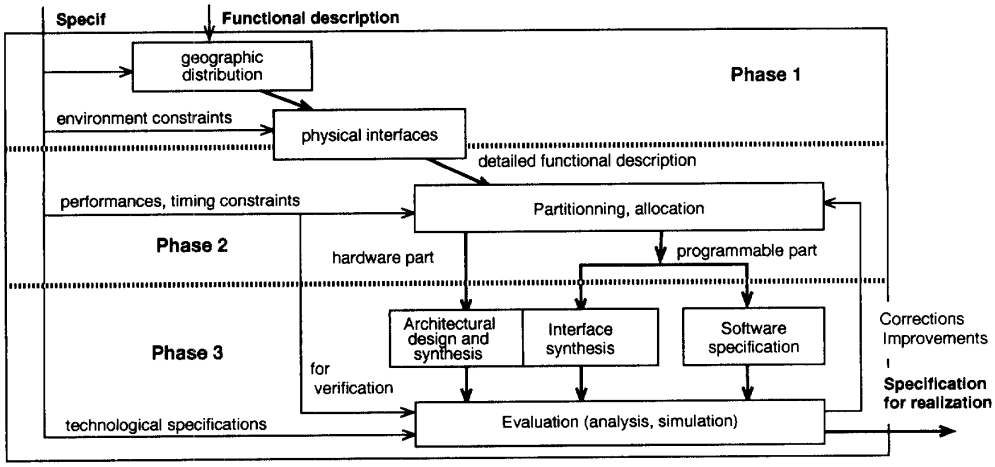


Figure 3. The implementation specification step for CoDesign.

microprocessor. The lowest bound of the activation period (Tactivation) for a feasible implementation of a function in software is between 20 and 50 µs without considering its execution time. Its execution time (Texec) is also to be considered. The ratio Texec/ Tactivation defines the utilization ratio of a microprocessor for the function.

- For the *highest performance*, because a byte can be sent or received in about 600 ns, EmissionMess and ReceptionMess must be activated at 1.8 Mhz. This implies a hardware implementation for these two functions. All other functions implemented in software determine the system performance.

The chosen hardware architecture shown in Figure 2 is well-suited to prototype both cases because it is a generic architecture. For the first case, the shared memory is not necessary (the same for BusReq, BusAck and Interrupt). In the second case, the links between software functions (Producer and Consumer) and hardware functions (EmissionMess and ReceptionMess) i.e. Lreq and Lind ports, are implemented with the shared memory and the interrupt line. The size of these ports is also an implementation parameter. In Section 5, we detail the two solutions for the emission part.

4.3: Generation, synthesis, evaluation

Phase 3 concerns the generation of the whole solution including software and hardware parts. For the hardware, two parts (and so two levels) are concerned: first the schematics of the system based on off-the-shelf components such as microprocessor(s), memories, EPLD, FPGA, etc., which are conventional and easy to draw with EDA tools, second the description of the ASIC part. This second part has to be described in VHDL in order to use available VHDL synthesizers. For that, each function is described in VHDL at a RT level. The pads description has also to be added.

For the software part, very often, a multi-task organization scheme has to be decided due to the implementation of many asynchronous functions on the same microprocessor. One way is to define a software implementation scheme (see MCSE) without using a real-time kernel (RT kernel): some functions such as EmissionMess and ReceptionMess are implemented on interrupts. Another way is to use a RT kernel. In such a case, each function is implemented as a task and relations between functions use semaphores, mailbox and shared resource mechanisms. Writing the software in C or C++ is easy: most part can be generated by tools in a semi-automatic manner.

Correct interfaces between the hardware and the software parts have also to be generated to correctly implement functional relations. From the 3 kinds of the MCSE functional relations, few transformation rules can be specified and the implementation results from using a VHDL package for the hardware, and C procedures for the software.

The hardware part described first in VHDL and then in a gate model after synthesis can be simulated and correctly verified with the software part and the system environment if you have previously written the complete functional model in VHDL during the functional design step. Critical parts and performance are easily evaluated by this way by analyzing the simulation waveforms.

From the result of this CoDesign step, the system can be prototyped. An important question is then how to test and debug such a prototype with software and hardware parts. One answer is to monitor both parts. Monitoring the hardware with a logic analyzer is more difficult for ASIC than for a board including standard LSI components. Moreover, errors can be due to the link between the hardware and the software parts of the solution. The CoDesign process we have detailed above is helpful for this task. Because we propose to use generators to produce the software and the hardware description in VHDL, it is therefore easy to add specific statements (software statements or VHDL statements) to obtain a way of capturing an event trace in real time which permits to understand the behavior of the system [6]. These statements can then be removed at the end of the development.

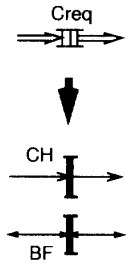
5: Detail of our solution for the transmission part

To illustrate Our CoDesign procedure, in this section we describe the solution developed only for the transmission part of the communication protocol but for the two performance levels. The objective is to show the advantage of the functional description for deciding the partitioning and to present some transformation rules of Hw/Sw functional relations and the corresponding VHDL description style.

From the functional structure described in Figure 1, we extract the necessary functions (the left side) to define the hardware and the interface with the application software. For high performance, only the producer function is implemented in software. For low performance, EmissionMess is also a software function. For both cases, the problem is to translate a port relation (Lreq or Creq) as the interface between the hardware and the software parts. Shared variables implemented as flip-flops and registers are used for that.

5.1: Low-performance solution

The EmissionMess function has to send each message byte after byte in Creq. EmissionByte driven at 20 Mhz




```
Procedure EmissionMess(Mess:def_mess);
  Const addr of CH&BF=F80000h;
  begin
    for I=1 to Mess[2]+1 do begin
        CH&BF:=Mess[I]&false&true;
        repeat until not BF;
        end;
    CH&BF:=Mess[Mess[2]]&true&true;
    repeat until not BF;
  end;

Mess[2] represents the message length
BF and CH are written concurrently
because BF and CH are at the same address
```

```
EmissionByte: Process (Clk20, Reset);
  begin
  if (Reset = '1') then
        RstBF <= '1'; .......;
  elsif (Clk20'event and clk20='1') then
    RstBF <= '0';
    case STATE is
      when Idle =>
          if BF='1' then
                CHR := CH;
                RstBF <= '1';
                Emission <= '1';
                STATE <= TokenWait;
      when TokenWait =>
          .............
```

```
VarCH: Process (WrCH, Reset);
  begin
  if (Reset = '1') then CH <= "000000000";
  elseif (WrCH'event and WrCH='0') then
    CH <= Din; endif;
```

```
VarBF: Process (WrBF, RstBF);
  begin
  if (RstBF = '1') then BF <= '0';
  elseif (WrBF'event and WrBF='0') then
    BF <= '1'; end if;
```

Figure 4 The software driver and a part of the VHDL description for the interface.

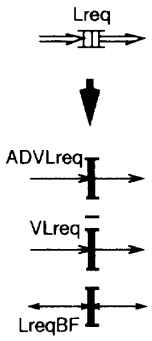
sends each byte in 600 ns when it has the token (Emission is used to ask for the token when receiving the first byte of a message). Even in the polling mode, the 68000 is not able to send a byte faster than 5 to 8 µs. Therefore, the necessary Creq port size is only one byte and Creq can be transformed in two variables: one to store the byte (CH) and another one to indicate the status Buffer full or not (BF) of CH. For the software interface, the best solution is to write EmissionMess as a driver procedure called by Producer, which includes a busy wait synchronization on the BF flag. The solution is given in Figure 4.

CH (and Creq) is 9 bit wide, CH[8] (the MSB) is used to indicate the last byte of each message. The shared variables CH and BF are described as asynchronous processes, solution which is necessary for BF because the resolution function is not synthesizable. EmissionByte is described at the RTL level synchronous to Clk20 to be synthesizable with the Compass synthesizer we are using.

5.2: high-performance solution

EmissionMess has now to be implemented in hardware. Therefore, the Lreq port is again transformed in two variables: one to store a message (VLreq) and a boolean LreqBF for synchronizations (Fig 5). The VLreq is implemented in the shared memory which is 16 bit wide. The address where the message is written in the memory is sent to the hardware (ADVLreq in 16 bits to simplify, the 8 least significant bits are always 0). To avoid the CPU waiting the end of the message transmission in a polling mode, the software is driven by LreqBF↓ as an interrupt. The Wait primitive is used to explain that the CPU is released for another task until LreqBF=false.

EmissionMess is described synchronous to Clk20 or to another lower clock derived from Clk20. It waits until the LreqBF flag is true and then starts the transmission by reading each 16 bit word of VLreq in the memory. Before reading, it has to ask the bus to the CPU with BusReq and then has to wait BusAck. The 2 bytes are successively sent




```
Procedure EmissMess(Mess:def_Mess);
  Const ADVLreq =F80000h;
  begin
    if LreqBF then wait(LreqBFfalse);
    VLreq:=Mess;
    ADVLreq:= addr of VLreq in memory;
    LreqBF:= true;
  end;

ADVLreq is the VLreq address in the memory
given to EmissionMess function.
wait(LreqBFfalse) is a primitive of a real-time
kernel to wait on the LreqBF interrupt.
LreqBF is described in VHDL in the same
manner as BF in Fig 5.
```

```
EmissionMess: Process (Clk20, Reset);
  begin
  if (Reset = '1') then
        RstLreqBF <= '1'; .......;
  elsif (Clk20'event and clk20='1') then
    RstLreqBF <= '0';
    case STATE is
      when Idle => if LreqBF='1' then
                STATE <= waitBus;
                BusReq <= '1'; end if;
      when waitBus => if BusAck then
                STATE <=readFirstRd; end if;
      when readFirstRd =>
                Read(ADVLreq, Wrd);
          .............
```

Figure 6 The new software driver and a part of the VHDL description for EmissionMess.

to the EmissionByte function according to the preceding protocol (CH and BF). The solution is given in Figure 6.

5.3: Results for the two solutions

In this section, we briefly describe the results of our experience by indicating in the following table, the length of the C software source code necessary for the 68000 CPU to simulate the producer and to load the Xilinx configuration, the length of the VHDL program (300 lines used to instantiate the pads), the size of the resulting ASIC and its performance expressed by the time between 2 bytes on the transbus.

Solution	Software	Hardware VHDL	Complexity in gates	performance
Low performance	90 lines	800 lines	500	17 µs
High performance	60 lines	1130 lines	1870	0.6 µs

5.4: Remarks

For our example, we have observed some limitations. The ReceptionByte function has to work at 60 Mhz and EmissionByte at 20 Mhz. By using synthesizers and specifically the Xilinx technology, it is not possible to work higher then 10 Mhz. Only a design by hand of a standard cell ASIC can lead to a good result at the 20 Mbits/s rate on the bus. Another limitation is due to the small complexity of the Xilinx 4005. Only the Emission part of the example can be implemented.

The chosen generic architecture is used only to evaluate the ASIC complexity in gate number and the communication performance. In the case of a real product, all components (kind of CPU, ASIC technology) have to be selected based on cost and performance criteria.

The VHDL description is very dependent of the used synthesizer: the Compass Asic synthesizer was available for us. The VHDL description of the pads is specific to the chosen ASIC and the synthesizer to be used. This task we have done by hand is made only once. When developing CoDesign applications, it is important to take care of the asynchronous characteristics of the Hw/Sw linked parts. Therefore, some constraints must be met to avoid race cycles and to respect flip-flop setup time.

At present, we are developing a new board for prototyping more complex CoDesign applications. This board includes a 68332 microcontroller, a Xilinx 4010, a shared memory and a specific ASIC we have designed to monitor both the hardware and the software during the test phase.

6: CoDesign procedure and tools used

Because the MCSE functional model allows to describe functions and inter-function couplings in a graphical and hierarchical manner, it is easy to understand that interactive and graphical tools can be developed to capture the functional structure, the behavior of each function, the executive structure with the technology (hard or soft) of each component, the mapping between the functional and the executive.

For the hardware part, we have developed a VHDL generator (G_VHDL) which produces a simulatable model [8] and a synthesizable model [5]. This model is a translation of the functional structure according to some rules describing the implementation of relations.

The codesign procedure and tools we are using are shown in Figure 6. This figure also shows the work to be done by the designer.

The partitioning and allocation step is done by the designer based on timing constraints and execution time estimation. This is not a complex task.

The transcription of the software part in VHDL is also quite easy because only the software tasks linked to the hardware (the interface or driver) have to be translated. Delays are inserted between VHDL statements to

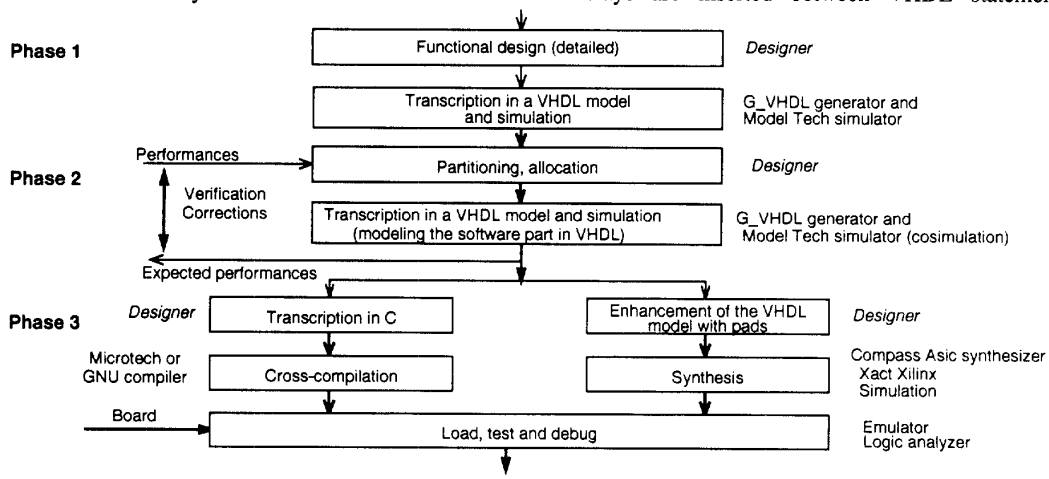


Figure 6 The CoDesign procedure and tools .

approximately model the execution time. A macroscopic modeling is often enough. This technique is used for cosimulation (concurrent simulation of the hardware and the software).

The transcription of the software part in C or C++ is at present done by hand. This task is easy and fast because the necessary software part concerned by codesign is small.

7: Future work

When experimenting different solutions or when requirements are changing, the designer's objective is to rapidly produce and test a new description. Based on the MCSE methodology, our solution is to use graphical tools to quickly modify the description, the partitioning and select the appropriate generic models for the interfaces, and then automatically produce the complete description. It is why we are focusing our research on developing such kind of tools both for the software and the hardware. The semi-automatic generation of Hw/Sw interfaces is possible based on generic VHDL models which are parametrized from the selected functional relations and appropriate transformation rules. The described example has shown the basis of such rules.

From the knowledge of inputs and outputs, the pads description in VHDL can also be generated in a semi-automatic manner. Some parameters have to be specified by the designer (kind and pin number of the pad, fanout, ...).

The codesign procedure we are following seems long due to the use of many tools from different vendors. It is necessary to lead to a more homogeneous and easier to use set of tools in order to shorten the development, test and evaluation cycle.

8: Conclusion

In this paper we have described an example and two different solutions for the transmission part of it. The objective was to explain the system design methodology we are using and to present our specific CoDesign procedure as an enhancement of the implementation specification step. We have also indicated tools we are using to efficiently develop hardware/software systems and more specifically for control-oriented applications: commercial tools for simulation, synthesis, cross-compilation, combined with prototype tools we have been developing these last years.

One of the most important conclusion is the necessity to start a CoDesign development with a complete methodology based on a system approach which permits to describe the solution at a functional and technology-independent level. The design part concerned by CoDesign appears therefore as a sub-set which includes tightly coupled links between hardware and software. The partitioning task is important but without existing automatic tools, it is easily managed by the designer according to an analysis procedure based on timing. One way is to develop an interactive partitioning procedure with a tool useful to evaluate the main characteristics for decision. The choice of the hardware architecture is also very important and has a great influence on the quality of the result and the complexity of the work. Many solutions exist. Designers are specifically concerned and this task is interesting and creative.

Our experience has shown us that it is necessary to have more integrated tools to improve the efficiency of transforming a functional description into a prototype, of testing and evaluating the system in real-time and embedded in its real environment. Interactive and easy to use tools is one way to keep the designer in the CoDesign loop to decide for good solutions based on evaluation.

9: References

[1] K. Buchenrieder, Focus on HW/SW CoDesign,, CODES/CASHE'93 , Second IFIP International Workshop on Hardware/Software CoDesign, Innsbruck, Austria, May 24-27 1993

[2] J.P. Calvez, O. Pasquier, A TRANSputer interconnection BUS for real-time systems, TRANSPUTERS'92 , Arc-et-Senans, France, May 20-22 1992, M. Becker et al., Ed. IOS Press, pp 273-283

[3] J.P. Calvez, Embedded Real-Time Systems. A Specification and Design Methodology, John Wiley 1993, 670 p

[4] J.P. Calvez, Spécification et conception des ASICs, Masson, June 1993, 580 p., to be published in english by Chapman&Hall, April 1995

[5] J.P. Calvez, D. Heller, P. Bakowski, Functional-level synthesis with VHDL, EUROVHDL'93, Hamburg, Sept. 20-24 1993

[6] J.P. Calvez, O. Pasquier, D. Isidoro, D. Jeuland, CoDesign with the MCSE methodology, EUROMICRO 94, Liverpool, Sept. 5-7 1994

[7] M. Edwards, J. Forrest, A development environment for the Cosynthesis of embedded software/Hardware systems, Proceedings of the European Design and Test Conference 1994, Paris, Feb. 28-March 3 1994, pp 466-473

[8] D. Fermy, B. Rossignol, P. Bakowski, J.P. Calvez, Tools to Design at a Functional Scheme Level using VHDL, EURO-VHDL'91, Stockholm, sept 1991

[9] R.K. Gupta, G. De Micheli, Hardware-Software cosynthesis for digital systems, IEEE Design & test of computers, Sept 1993, pp 29-41

[10] R.K. Gupta, C.N. Coelho, G. De Micheli, Program implementation schemes for Hardware-Software systems, IEEE Computer, Jan 1994, pp 48-55

[11] D.E. Thomas, J.K. Adams, H. Schmit, A model and methodology for Hardware-Software Codesign, IEEE Design & test of computers, Sept 1993, pp 6-15