# Exploration of Hardware/Software Design Space through a Codesign of Robot Arm Controller

Mohamed Abid[1], Adel Changuel, Ahmed Jerraya

Laboratoire TIMA/INPG, System-level Synthesis Group,
46 Av. Félix Viallet, 38031 Grenoble Cédex, FRANCE
Email: abid@verdon.imag.fr

## Abstract

*This paper deals with exploration of hardware/software design space. The analysis is illustrated using a design of robot arm controller. The controller makes use of an adaptive speed control in real-time. Several architectural solutions will be discussed with regard to their performance and cost. The goal is to select the best solution that satisfies the real-time constraints and minimizes the cost.*

## 1. Introduction

Real-time systems are specified not only by a set of tasks but also by constraints dependent on the application context. System design is a critical foundation since many systems include performance constraints as a part of their requirements. It is a complete system that performs all the functions properly in such a way that all performances can be met.

In recent years, real-time electronic systems used for dedicated applications consist of general-purpose processors or application specific hardware. When designing such a system, the designers have to decide among a full software solution, a full hardware solution and a mixed hardware/software solution. A purely software implementation reduces the design time and cost by implementing tasks as programs. This solution also makes easier the maintainability. Depending on the specification and system operating environment, a software solution can range from a simple single microprocessor to a multiprocessor configuration. Unfortunately software implementation does not always satisfy all requirements, in particular, the timing performance. Hardware implementations produce faster solutions which are however more expensive moreover they require a longer to be designed. To optimize the performance/cost, it is recommended to use a mixed hardware/software implementation [3].

[1] Assistant professor from ENIM (Ecole Nationale d'Ingénieurs de Monastir) - 5000 Monastir - TUNISIA

The renewed interest in such an implementation is driven by advances in technologies that support hardware and software parts. These technologies are needed in order to handle high complexity and to allow mixed implementations required for codesign. The main motivation behind this research is to allow the design and implementation of modern systems including both hardware and software. Several projects currently in progress (SpecSyn at Irvine [7], CODES at Siemens [4], SDW at Italtel [1], Thomas approach at CMU [2], Gupta and De Micheli approach at Stanford [8], Wolf approach at Princeton University [13], Chinook at the university of Washington [6], Ptolemy at Berkeley [10]) are trying to integrate both hardware and software in the same design process.

The long term objective of this research work is to develop methods and tools for hardware/software codesign. The objective of this paper is to explore the hardware/software design space. An analysis will be performed using a codesign of a robot arm controller. The controller makes use of an adaptive speed control in real-time. Several architectural solutions will be discussed with regard to their performance and cost. The aim is to select the optimal hardware/software solution and illustrate the advantages of mixed hardware-software implementations.

The next section describes the robot arm controller system used to illustrate the hardware/software partitioning. Section 3 gives an outline of the hardware/software codesign flow and a target architecture that will be served as a platform onto which a mixed hardware/software system is mapped. The hardware-software partitioning of robot arm controller system and discussions of different solutions will be described in section 4. Finally, section 5 draws some conclusions and outlines future directions on hardware/software partitioning.

## 2. The Robot Arm Controller

The robot arm controller under discussion acts between a "Host Machine" and a robot arm that makes use of 18 stepper motors (figure 1). The host fixes the trajectory of the arm. It then sends the commands to the robot arm. The controller makes use of an adaptive speed control to smoothen the motion of the arm.

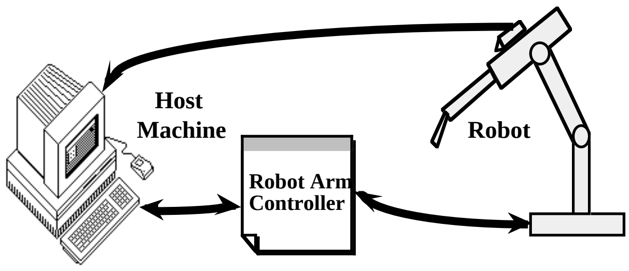

figure: 1 Control System in Robotics

## 2.1. Principle of Adaptive Control

The trajectory of the arm is decomposed into short distance segments to go. In order to realize a given segment, each motor will have to perform a given number of steps determined by the host (in the form of packets of pulses). We assume that all the motors have to perform the corresponding segment in the same lapse of time with a constant speed. Then the speed of each motor has to be adapted according to the number of steps required by the performed segment.

Besides, in order to avoid the motors from stopping brutally between two successive segments, an anticipation is performed. Thus, the speed of a motor depends not only on the current segment but also on the precision and on the next neighbour segments. This anticipation creates an error. The magnitude of this error is controlled through the amount of anticipation.

The distribution system feeds the adaptive controller in parallel. The motor controllers send back continuously informations about the distance they still have to go. This information is used by the distribution system in order to decide whether it should send a new segment to the motors. So, the system performs two principal tasks: adaptive distribution of pulse packets and adaptive speed control in real-time.

*** Adaptive Distribution**

To each motor, is associated a "Tank" which acts as a "Buffer" memory. The latter is used to stock the received packets of pulses from the host machine (Ri). We assume that the maximum number of pulses (Pmax), which may be used by each speed control, is known. After the arrival of each pulse packet:

- the "Tank" having the maximum of pulses is identified (Rmax),
- the number of pulses needed to be sent to each speed control is:

$$P_i = (R_i * P_{max})/R_{max}$$

The "Tanks" are, of course, actualized after each supply in order to determine the one having the highest number of pulses in order to compute the new proportions of pulses.

With this approach, the distribution of pulses for different motors is always proportional to the number of pulses kept in the "Tank". This ensures the continuous operation of the motors and a smooth control of their speed.

*** Adaptive speed control in real-time**

The actual speeds of the motors are calculated and adapted according to the number of pulses to be realized and the number of pulses already achieved. This permits anticipation.

To each motor, is also associated a "Tank" of pulses memorising the proportional number of pulses from the distributor. The speed of the motor should respect a well defined curve for acceleration and deceleration. The number of pulses may be consumed before the motor reaches its minimum speed; this situation introduces an anticipation (negative number of pulses) to be taken into account during the next step. The speed of each motor would then be calculated with respect to:

- the remaining number of pulses to be executed,
- the current speed of the motor, and
- its acceleration/deceleration curve.

At each step, we have to decide whether to accelerate, to decelerate or to maintain the speed of each motor.

## 2.2. Real-time constraints

The response time of each motor is 6 ms. Consequently, in order to ensure the command in real-time, the speed variation must be ensured within 6 ms. The time needed to update the speed must be less than 6 ms.

We will explore a several implementations. The goal is to determine which parts of the system should be implemented as hardware and which ones with software. The solution has to satisfy the real-time constraint.

Before the design space exploration of this example, let us give a brief description of COSMOS environment for hardware/software codesign.

## 3. Hardware/Software Codesign Flow

Figure 2 shows the codesign flow performed in order to transform a system-level specification of a mixed hardware-software implementation. The codesign process is composed of four steps [9]: system-level specification, system-level partitioning, communication synthesis and software and hardware synthesis.

The goal of hardware/software codesign is to produce an efficient implementation that satisfies the performance and minimizes the cost, starting from the initial specification. However, there exists a diversity of technological solutions based on available hardware-software components. Of course the design process will include lots of feedback loops in order to redesign parts of the system or even the full system. Different solutions are analyzed to choose the optimal solution; it results from trade-offs between both technologies which satisfy the required performances. However, at any stage of the design, the user can cancel one or several design steps in order to explore new choices.

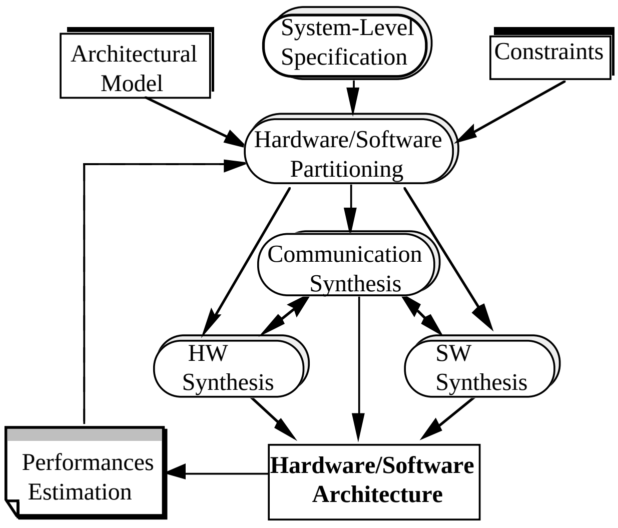

**Figure 2: co-design flow**

The architecture model specified earlies in the codesign process guides the detailed implementation choices. In fact, the differents architectural solutions depend on the input description and the architecture model. To explore the hardware/software design space, an example of target architecture onto which a mixed hardware/software system will be mapped, is performed. This architecture is based on a PC (for software implementation) with an FPGAs extension board (for hardware implementation).

The platform include several standard peripheral components such as I/O ports (serial, parallel), Direct Memory Access, Interrupt controllers etc. Their drivers may be kept either in EPROM or may be down loaded from an external memory. The system bus is used for the transfer of data, address and control between different components.

Hardware parts is composed of a set of FPGAs (Xilinx). The FPGA's parameters may be read from an EPROM or an external memory system, in which case a routine is executed on the microprocessor.

Standard and FPGAs components used in this platform are designed to cooperate with the microprocessor. These components work with the interface driver running on the microprocessor, and translate reads and writes to and from components into the proper handshake and data transfers. Many communication modes (such as synchronous and asynchronous modes using acknowledgement or not etc.) may be used. The designer can use the functions that are achieved by the operating system or perform others functions to transfer control between the microprocessor and the components.

The mixed hardware/software system will be mapped from behavior description. COSMOS uses unified model for co-simulation and co-synthesis of mixed hardware/software systems [12]. A modular description is composed of three parts: a set of hardware components described in VHDL, a set of software components as C programs, and a set of communication component(s) to connect the above two parts. The latter, namely the communication components, corresponds to a library of components, which helps to hide the possibly complex behavior of the platform. This description will be used for co-synthesis to produce hardware/software prototypes.

## 4. Design space Exploration of the Robot Arm Controller:

The controller will be designed as two parts. The 18 Speed control sub-systems are independent and may therefore be thought of as concurrent state machines. Distribution sub-system computes the proportional pulses for each motor, this is done concurrently with the Speed control sub-systems. Different hardware/software implementations may be explored.

In this section, we well present the functional specification of the application, followed by the discussion of several architecture solutions.

### 4.1. Functional specification

As shown in figure 3, the distribution sub-system receives data from the Host Machine and provides the travelling distance to the Speed Control sub-system. This sub-system computes the number of pulses for speed control and translates them into motor control signals. The motor returns the pulses already realized. Thus the speed control sub-system controls the motors step by step in real-time.

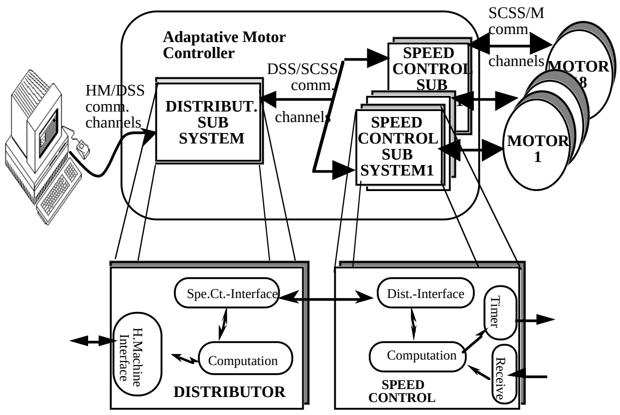

**Figure 3: Specification of Robot Arm Controller**

The distributor sub-system fulfil's the three following tasks:

i) it receives the command or data (the packets of pulses) from the host machine through the HM/DSS channel,

ii) it computes the proportional number of pulses for each motor,

iii) it supplies to the speed control sub-systems the corresponding pulses when they are ready through the DSS/SCSS channel.

For each motor, the speed control sub-system receives the corresponding number of pulses in the form of a packet, computes the new speed and translates them into control signals (Sequence of Modulated Pulse) while the motor returns the realised impulses.

The speed control sub-system includes four tasks:

i) the receipt of the command or data (proportional number of pulses) from the distributor sub-system through the DSS/SCSS channel,

ii) After the communication with the distributor sub-system, the information is decoded. The latter can be either a command or data. The speed control sub-system accordingly treats the above in one of the following ways:

- If there is a Control command (initialization, starting, etc.), then the sub-system initializes the corresponding operation,
- If there are new pulse data, they are added to the remaining pulses. Then the new speed of a motor will be calculated on the basis of the remaining number of pulses as well as the present speed. The above would thus result in either an acceleration or a deceleration.
- In the case when the received command inquires the state of the speed control of the motor, the sub-system returns the answer.

iii) The sub-system transmits the calculated speed to the motors through the SCSS/M channel.

iv) Pulses already executed are simultaneously counted through the SCSS/M channel. This enables the update of the pulses to be realised for the calculation of new speeds.

For implementation, there are different partitioning solutions: pure-software, pure-hardware or hardware-software implementation. In fact, the system is specified not only by a set of tasks but also by the response time of the motor. Thus, the functional behaviour of this system must be logically and temporally correct. In order to ensure the control in real-time, the speed variation must be ensured during a lapse of time smaller than the response time of the motor. In other words, the time needed to calculate the next speed must be less than the maximum response time of the motor (6 ms).

### 4.2. Exploration of the hradware/software partitioning space

Starting from the specification given in figure 3 several solutions may implement the Robot Arm Controller. Each solution will be composed of a set of hardware components and/or software components and of communication components.

For the sake of simplicity we will restrict the type of components used. A software component will be made of a 80286-based architecture; a hardware component will be made of a Xilinx 4013 FPGA, while communication components may be existing components or Xilinx 4013 FPGAs.

An additional assumption is that a cost is associated with each component. For instance, an FPGA will cost 10 units, while software will cost only 5 units. Since we have fixed the components, we can predict the performance of each task whether it is implemented in hardware or in software.

Although, in order to be able to measure the performance of codesign solution, a metrics of execution time is used. In fact, the execution time is decomposed into two components: the computation time and the communication time. The computation time is defined as the execution time for the processor to perform all its internal computations during a single pass through the behavior. The communication time is defined as the access time of external data to the processor (during a single pass through the behavior of the processor too).

In order to determine the total execution time of the system, the number of executions for each task is calculated with regard to *one packet of pulse*. The frequency of each task is then deduced according to the *speed*. For this, we assume that:

- the speed variation corresponds to 16 pulses at most,
- the supply from the Host Machine is 150 pulses,
- the maximum number of pulses to be sent to a speed controller of the motor is 15.

For software, a dynamic simulation [11] is used to estimate the execution time. The time is determined by calculating the maximum execution time of real-time programs. For Hardware, the execution time is deduced using SYNOPSYS with VHDL description.

Table 1 lists the call frequency and the execution time for each task according to the implementation (software or hardware). These number corresponds to cycle count obtained by evaluation.

| | Call Frequency | Software (cycles) | Hardware (cycles) |
|---|---|---|---|
| **Distributor** | | | |
| H.Mach. Interface | 0.106 | 12 | 2 |
| Computation | 1.066 | 1250 | 5 |
| Spe.Ctr. Interface | 1.066 | 11 | 3 |
| **Speed Controller** | | | |
| Dis. Interface | 1.066 | 10 | 3 |
| Computation | 3 | 1480 | 6 |
| Timer | 16 | 25 | 4 |
| Receive | 16 | 30 | 6 |

**Table 1: Execution time**

|  | Software (cycles) | Hardware (cycles) |
|---|---|---|
| **Distributor** | **1333.41** | **8.56** |
| H.Machine Interface | 0.19 | 0.03 |
| Computation | 1321.50 | 5.33 |
| Spe.Ctr. Interface | 11.73 | 3.20 |
| **Speed Controller** | **5330.06** | **70.33** |
| Dis. Interface | 10.06 | 3.20 |
| Computation | 4440.00 | 18 |
| Timer | 400 | 64 |
| Receive | 480 | 96 |

**table 2: Normalised execution time**

By combining the informations those given by table 1 (the frequency of calls and execution time), we deduce the normalized execution time (number of execution cycles to calculate the speed) of each task for hardware and software implementation (table 2).

The discussion about hardware/software solutions considers 6 ms as being the real-time performance and a cost constraint of value 80.

### 4.2.1. Pure Software Implementation Using One Software Processor

We assume that all the tasks will be executed on the same processor. This will need an extra scheduling step that will arrange the execution order of the tasks. Let us now deduce the execution time for the distributor and the 18 motor controllers. In fact, the total execution time is about 9.7 ms (thus larger than 6 ms). However, the cost of this solution is 5 (less than 80). Although, this solution meets the cost constraint, it does not meet the real-time requirement.

### 4.2.2. Pure Software Implementation Using Two Software Components

The system can be designed as two asynchronous parts: the distributor may be done concurrently to the 18 motor Speed Control Sub-Systems. Thus each part can be run on a separate microprocessor. In this case, the execution time for each part may reach 6 ms at most. After the analysis, the execution time of the 18 Speed Control Sub-Systems is almost 9.5 ms (>6 ms). Thus this software solution too does not meet the real-time requirement. Note however that the cost of this solution is 10 (< 80).

### 4.2.3. Pure-Software Implementation Using 19 Software Components

Another solution using a microprocessor for each speed control sub-system may be considered. We associate a software component to the distributor sub-system and a software component for each motor controller. This solution meets the real-time requirement (with an execution time almost of 0.95 ms). However it does not meet the cost constraint (95>80).

### 4.2.4. A Pure Hardware Solution

All the functions could be realized in hardware with FPGAs modules. In fact, the 18 Speed Control Sub-Systems are independent and may therefore be designed as concurrent state machines. On considering the FPGA capacity, the number of speed control sub-systems integrating on one chip is 3. Therefore this implementation has to use one FPGA for the distributor and 6 FPGA chips for the 18 Speed Control Sub-Systems. The cost of this solution is 70 (less than 80). Thus this solution meet the cost constraint.

However, it can be advantageous to make some parts of the design in software, while time critical parts are realized as hardware. Of course, the mixed implementation has to meet the real-time requirement. These alternatives will be analyzed in the next paragraph.

### 4.2.5. Hardware/Software Solutions

Because external real-time signals are better analyzed in hardware, we associate hardware components to the 18 Speed Control Sub-Systems. However, if we associate a microprocessor to the distributor, the execution time is 1.33 ms. Thus, this implementation has to use one microprocessor for the distributor and 6 FPGA chips for the 18 Speed Control Sub-Systems (figure 4). The cost of this solution is 65 (less than 80).

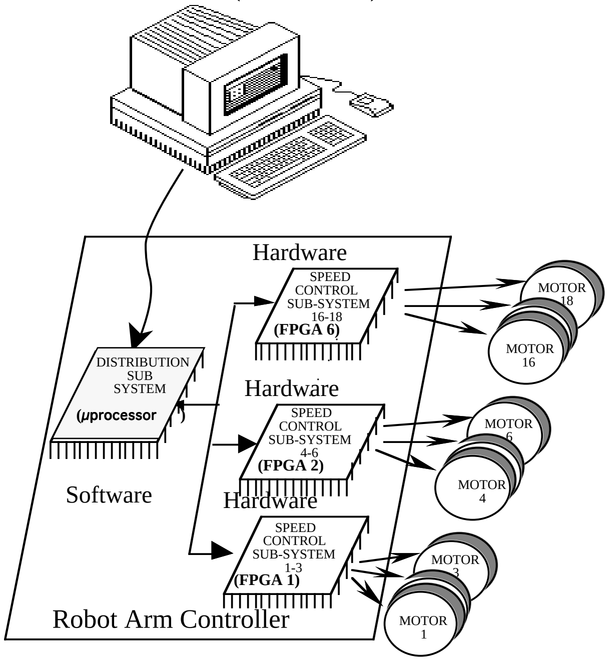

**Figure 4: Hardware/software solution**

### 4.2.6 Summary of the Codesign Exploration

Three solutions met the real time requirements and four solutions satisfied the cost constraints as illustrated in Table 3. However only two of them met both cost and real-time constraints. In this case we select the optimal solution. In fact, the last solution is cheaper than the fourth solution. It has a better cost compared to solution 4.

| Solution | Normalised Execution Time | Real-Tim eviolatio n | Cost | Cost violation | possible Solution |
|---|---|---|---|---|---|
| **1** | 9.7 ms | Yes | **5** | **No** | KO |
| **2** | 9.5 ms | Yes | **10** | **No** | KO |
| **3** | **0.95 ms** | **No** | 95 | Yes | KO |
| **4** | **0.21 ms** | **No** | **70** | **No** | **OK** |
| **5** | **1.33 ms** | **No** | **65** | **No** | **OK** |

**Table 3: result of analysis**

A prototype is performed using an Intel 286-based platform and a development board that consists of Xilinx 4013 FPGAs. The design of the system based on fuzzy logic is given in a companion paper [5]. An analysis of the prototype indicates that this solution correctly implements the system functionality while meeting real-time requirements. This prototype has been experimented with the stepper motors. The result is satisfactory.

## 5. Conclusion

This paper discussed a hardware/software codesign of a Robot Arm Controller with 18 motors. We showed that several solutions may implement an initial specification. The use of a cost and performance measurement model allowed to select a solution that meet all the requirement of the application. The advantages of mixed hardware-software implementation are illustrated.

Of course several other key issues are still needed in order to explore a large design space. These include the automatic algorithms for hardware/software. These are needed in order to allow fast and large exploration of the design space. The efficiency of such techniques will depend on the objective function used. This brings us again to the previous point: the need for good estimation methods. Research efforts will be currently deployed to formulate estimation models and objectives functions.

## References

[1] S.Antoniazzi, M.Mastretti, "An Interactive Environment For Hardware/Software System Design at the Specification Level", Microprocessing & Microprogramming, Vol. 30, pp. 545-554, 1990.

[2] J. K. Adams and D. Thomas "Multiple-Process Behavioral Synthesis for Mixed Hardware/Software Systems", 8th International Symposium on System Synthesis, Septembre 1995, Canne, France.

[3] G. Boriello, K. Buchenrieder, R. Camposano, E. Lee, R. Waxman, W. Wolf, "Hardware/Software Codesign", IEEE Design and Test of Computers , pp. 83-91, March 1993.

[4] K. Buchenrieder, A. Sedlmeier, C. Veith, "HW/SW Co-Design With PRAMs Using CODES", Proc. CHDL '93, Ottawa, Canada, April 1993.

[5] A. Changuel, R. Rolland, A. Jerraya "Design of an Adaptative Motors Controller based on Fuzzy Logic using Behavioural Synthesis", EuroDAC ' 96, September 16-20 1996, Geneva, Switzerland.

[6] Pai H. Chou, Ross B. Ortega, Gaetano Borriello, "The Chinook Hadware/Software co-synthesis system" International Symposium on Synthesis System, Sept. 95, Cannes, France.

[7] D. D. Gajski, F. Vahid "Specification and Design of Embedded Hardware-Software Systems, IEEE Design & Test Computers, Spring 1995.

[8] R.Gupta, G. DeMicheli, "Hardware-Software Co-synthesis for Digital Systems", IEEE Design and Test of Computers, pp. 29-41, September 1993.

[9] T. B. Ismail, M. Abid, A. A. Jerraya, "An Approach For Hardware-Software Codesign", IEEE Inter. Wchop in Rapid System Prototyping, Juin 1994, Grenoble, France,

[10] A. Kalavade, E. A. Lee, "Manifestations of Heterogeneity in Hardware/Software Codesign", Proc. of 31st Design Automation Conference (DAC'94), pp. 437-438, June 1994.

[11] M. Nouira, "Simulateur et estimateur des performances sur une architecture specifique", Final Year Projet Report, Juin 1995, ENIM, Monastir University, Tunisia,

[12] C. A. Valderrama, A. Changuel, P. V. Raghavan, M. Abid, T. B. Ismai, A. A. Jerraya, "unified model for co-simulation and co-synthesis of mixed hardware/software systems", Proc. EDAC'95, Paris, France, February-March 1995.

[13] W. H. Wolf, "Hardware-Software Codesign of Embedded Systems", Proceedings of the IEEE. Vol. 82, NO. 7., Iuly 1994