Hardware Interface Design For Real Time Embedded Systems

Adel BAGANNE Jean Luc PHILIPPE Eric MARTIN
LESTER LAB, 10 Rue Jean Zay 56100 LORIENT FRANCE
Email: name@univ-ubs.fr; Tel: (33)02/97/87/28/34; FAX: (33) 02/97/87/28/62
http://lester.univ-ubs.fr:8080/

Abstract

In this paper, we address the problem of hardware interface design in a Codesign approach for real time Digital Signal Processing (DSP) applications. We focus on the allocation problem of necessary storage components needed for data communication between hardware-software components. First, we present a modeling style for I/O data exchanged between both components and we describe our generic model of the hardware interface. Second, we describe a formal technique to the necessary storage components allocation. Our design strategy starts from the hardware I/O transfer sequences computed by a high level synthesis tool, like GAUT[3]. It incorporates some interface specification (I/O transfer order, timing constraints) obtained by any cosynthesis tool such as [6]. The proposed allocation procedure assigns for each I/O data a time interval at which its transfer could occur. The results that we have presented are based on FFTs algorithms implemented on Asics.

1. Introduction

In this paper, we address the problem of Hardware-Software (H/S) interface synthesis task which is a key issue in Codesign. It specifies how the software and hardware components communicate. Interface synthesis can be divided into two major subtasks, namely *communication synthesis* and *H/S synchronisation. Communication synthesis* consists in determining : *(i) the communication protocol* used for data transfer between components (blocking or non-blocking, master-slave, special protocols[7]) *(ii) timing requirements* for each data transfer (transfer delay estimation, date of transfer *(iii) the transfer mode* between hardware and software (Memory mapped I/O scheme, direct connection).

The *H/S synchronization* subtask refers to making data transfer happen in a specified time order, or more especially, coordinating the real time presentation of data and maintaining the time-ordered relation among the Asic and Processors. We present in this paper a design methodology of a hardware interface that supports functional constraints description of I/O data exchanged between hardware and software.

We refer to the hardware component as Asics (Applied Specific Integrated Circuits) and the software component as processors. Our work domain is the real time implementation of Digital Signal Processing and Image applications. These applications are characterized by a parameter T_0 called the *iteration period.* All computations are repeated every T_0 time. Real time implementation of these applications on mixed H/S architecture should satisfy the following periodic constraints : *(i) execution timing constraints* on each component *(ii) I/O data transfer sequences* between both components with their timing specifications. Such constraints can be obtained by any cosynthesis tool[6].

We consider the hardware interface synthesis task as a part of the Asic generation process by means of a High Level Synthesis (HLS) tool GAUT [3]. This tool involves several techniques such as structural transformations or hardware selection [4] that allows a large exploration of the hardware design process. Our hardware design methodology is based on the following points :

1. the **Processing unit** (PU) of the Asic is the first functional unit synthesised by GAUT because it undergoes the most important constraints from the real time execution constraints.
2. the **hardware interface** is synthesized after PU and takes into account both I/O PU constraints and Processors I/O constraints specified by any cosynthesis tool used for system partitioning.

This paper is organized as follows: In section 2, we state the assumptions we make about our system target architecture and design methodology. In section 3, we briefly describe the main steps of our hardware design by the Gaut HLS and we give our formulation of the hardware interface synthesis problem. In section 4, a generic architecture of the interface is presented and discussed. In section 5, we introduce a timing and communication model for the I/O data transferred between the Asic and the processors. In section 6, we present our design technique for analysis and synthesis of the interface buffers under I/O

timing requirements. In section 7, we present some results based on FFTs algorithms implemented on Asics.

2. Preliminaries

Our system target architecture (see figure 3.1) consists of an **Asic** *(hardware component)* and a **general purpose processor** *(software component).* We consider the design of real time application that is described at the behavioral level by some specification language[1] such as SDL, Statecharts etc. We assume that a cosynthesis tool such as [6,8] is used to partition the specification into three components: Software, Hardware and Interface. The software partition is composed of threads (T_1 ,...,T_N) which are sets of computations that execute in deterministic time and other sets of computation that execute in non-deterministic time (such as loops and synchronization tasks). A control-data driven scheduler is used in the software that allows both hardware and software to schedule threads over time and to exchange I/O data [9]. The hardware partition is specified with VHDL at behavioral level and constitutes the entry point to the high level synthesis tool GAUT. The hardware design process is described in the next section.

The H/S interface specification is assumed to be further annotated with detailed information. The following lists a description style of such information:

1. The Asic-processor interconnection is realized by *direct links*. Communications between both components are based on the *message passing model[8]* and are *buffered* in order to minimize synchronization delays.
2. The communication protocol is slave-master where the software component (Processor) works as a master. The hardware and software modules interacts with an interrupt mechanism. The software running on processor is capable of communicating with the Asic and performing an interrupt-driven I/O.
3. The communication channels between threads and hardware partition are mapped to I/O busses such as in[10]. Therefore bus transactions and between the processor and the Asic could be described at the lowest level: I/O data structure (scalar, vector) and their timing constraints are known. In the rest of the paper, we will refer to these constraints as **IOCC** (Input Output Cosynthesis Constraints).

3. Hardware Design Overview

In this section, we briefly describe the main steps of our hardware design by the Gaut compiler and we give our formulation of the hardware interface synthesis problem.

3.1 Asic Design

Figure 3.1 illustrates the design procedure of both the Asic and the hardware interface. The hardware partition produced after patrtitionning is described in VHDL at

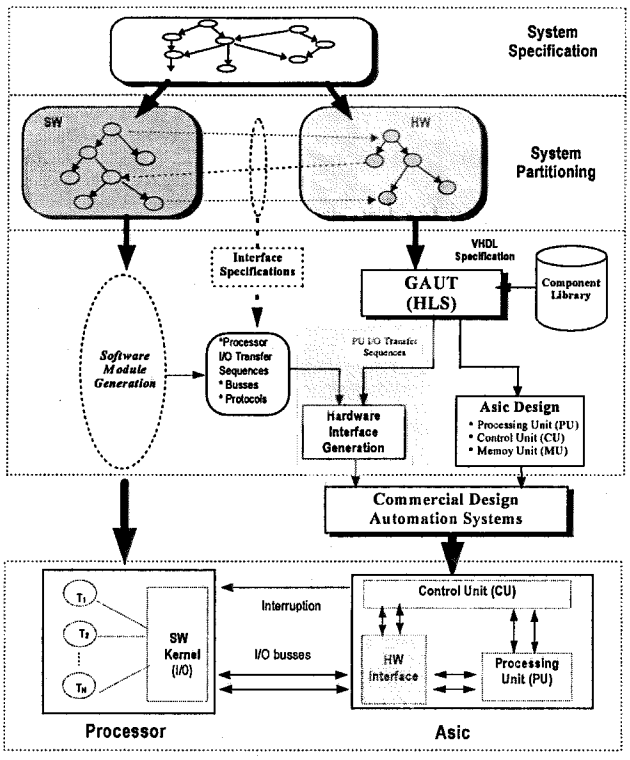

Figure 3-1 Hardware module design overview

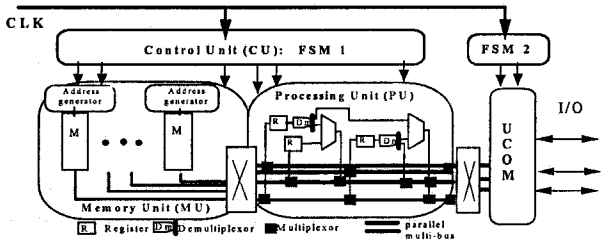

Figure 3-2 Generic Architecture of the Asic

behavioral level. From this description, the timing constraints (computation delays) imposed after partitioning, and a technological driven library (standard cells, FPGA), the hardware architecture will be synthesized by GAUT HLS. This architecture is based on four functional units (see figure 3.2): the **Processing Unit** (PU), the **Control Unit** (CU) the **Memory Unit** (MU) and the **Communication Unit** (UCOM) that implements the H/S interface in the hardware side. The topology of the processing unit is based on elementary cells including an operator, registers and interconnecting components. The memory unit implements registers, memory banks and their associated address generators, as well as channel for data transfer [3].

After the PU synthesis step, GAUT produce a set of functional constaints relative to the PU I/O sequence of transfers : the production and consumption dates of the I/Odata. This sequence thus constitutes the background and the schedule of conditions for the Asic's interface synthesis procedure.

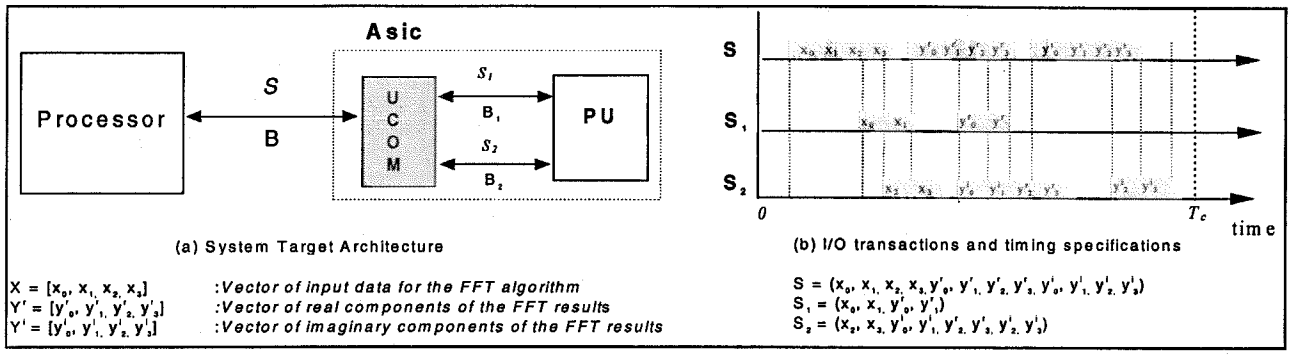

Figure 3-3 Example: Hardware FFT implementation and Asic-processor data transfer sequences

3.2 Problem formulation

Let us consider a mixed implementation of a DSP application where the hardware component consists in a FFT implementation. For simplicity, we consider a four points FFT. Figure 3.3 shows a target architecture composed by an Asic (hardware component) and a processor (software component). At each iteration, an I/O sequence S is transferred over the bus B between both components. Each communication data may be described by different ways *: (i) transaction order (ii) timing constraints* determined by some communication synthesis process [10, 11]. These constraints can include strict timing specifications (fixed dates of transfers, transfer deadlines) [see section 5.1] or only just I/O data transaction order. As mentioned previously our architectural synthesis tool GAUT starts Asic design cycle by the processing unit (PU) and gives the consumption and gives the production and consumption dates of I/O data. S_1 and S_2 are the fully specified I/O sequences transferred respectively over B1 and B2 (internal busses of the PU). The problem of the hardware interface generation can be explained in the following way :

How to generate the smallest amount of hardware (datapath and control logic) that still meets the I/O sequence constraints (timing requirement, data transfer order) ?

The aims are the following:

1. *mixing and demixing* of data busses of processor towards PU busses : it concerns to ensure all the necessary link between the Processor and the PU busses.
2. *synthesis of I/O data storage* under **IOCC** constraints.

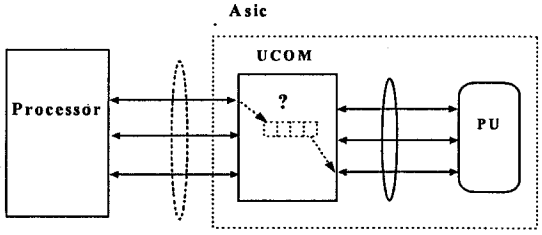

Figure 3-4 Hardware Interface synthesis problem

4. Generic model of the Hardware Interface

The hardware interface model that we have adopted can be seen in figure 4.1. It is composed of datapath and an FSM based controller. The datapath of the interface includes 3 major components : buffers for storing I/O data, buses and interconnection components such as multiplexers, demultiplexers and tristate buffers. The storage buffers are composed of FIFOs, LIFOs and registers. Busses included in the interface have two types: external and internal. External busses have two categories. The first one is I/O busses which represent the physical links between ASIC and processor. The second category is the *Interface-Processing Unit* busses. Their number is determined by the synthesis step of the processing unit. However , it should be greater or equal to the maximum number of simultaneous data transfer between the interface and PU. The FSM based controller implements the communication protocol defined by the cosynthesis tool and generates the adequate control signals for the datapath components.

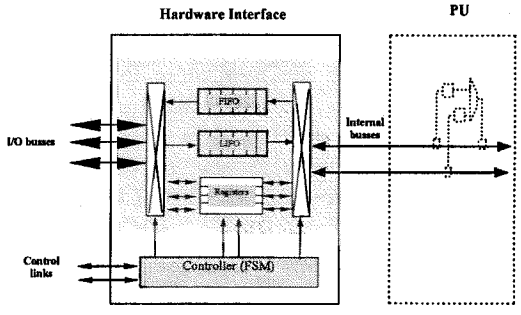

Figure 4-1 Generic model of the hardware interface

5. Modeling and specification of I/O sequences

In this section, we will consider that processor and Asic have the same reference of time. In the following sections, we will describe the timing specification of the Asic-processor I/O data transfer.

5.1 Asic-Processor I/O transfer sequences:

In our model (see figure 5.1) the processor has N_B busses connected to the Asic. As mentioned previously, the set ordering of communication data is provided by a cosynthesis tool and therefore known in advance. Let us have an ordered set P of all I/O data transferred over N_B busses in one iteration period T_e. Some of the data item may be structured (vectors).

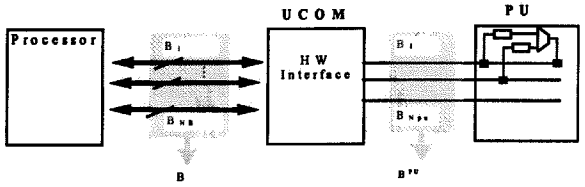

Figure 5-1 Topology model of the hardware interface

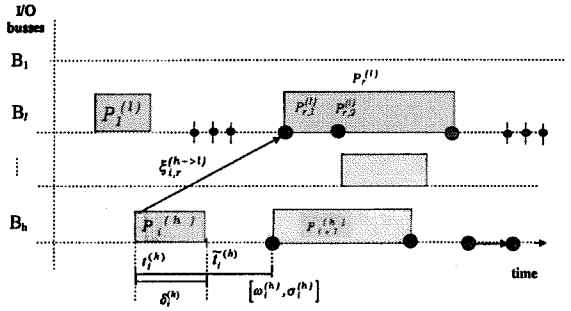

Figure 5-2 Timing specifications of I/O processor data

Let us define the set P as an ordered set of data subsets:

- $P = \bigcup_{1 \le h \le N_B} P^{(h)}$ *where* $P^{(h)}$ *denotes the ordered sets of data subsets (vectors) transferred over the h-th bus.* $P^{(h)} = \left[P_1^{(h)} \ldots P_{N_h}^{(h)}\right]$
- *The single subset* $P_i^{(h)}$ *is defined as a set of* ordered scalar *transferred over the* h-th *bus.* $P_i^{(h)} = \left[P_{i,1}^{(h)} \ldots P_{i,I_h}^{(h)}\right] = \left[P_{i,j}^{(h)}\right]_{1 \le j \le I_h}$

Let t be the common reference of time and let $t_i^{(h)}$ and $\tilde{t}_i^{(h)}$ be respectively the start and the end of transfer of the *i-th* data item $P_i^{(h)}$ on the *h-th* bus (see figure 5.2). The communication time for $P_i^{(h)}$ transfer is defined as $\delta_i^{(h)} = \tilde{t}_i^{(h)} - t_i^{(h)}$. Note that our a-priori knowledge only concerns the mutual time constraints of data, and not the time values themselves, i.e. $\tilde{t}_j^{(h)} \prec \tilde{t}_k^{(h)} \; \forall j < k$. However, our model can be refined and extended to cases where all data ordering and their timing constraints are fully satisfied. Timing constraints are then introduced to define upper and lower bounds between the start times of two data transfers. The following equations express the main (Min/Max) timing constraints that can be specified for each pair of the exchanged data unit :

$$t_i^{(h)} + \omega_i^{(h)} \le t_{i+1}^{(h)} \le t_i^{(h)} + \sigma_j^{(h)} \text{ where } t_i^{(k)} + \xi_{i,j}^{(k->h)} \le t_j^{(h)}$$

- $\omega_i^{(h)}$ *and* $\sigma_j^{(h)}$ *define respectively the minimum and maximum delays between two consecutive subset transfers over the* h-th *bus.*
- $\xi_{i,r}^{(h->l)}$ *defines the minimum delay between transfer starts of the data units* $P_i^{(h)}$ *and* $P_r^{(l)}$

5.2 Processing Unit I/O transfer

We have mentioned previously that all the PU I/O transfer sequences are fully specified by GAUT HLS. The hardware interface exchanges data with the PU over N_{pu} internal busses (see Figure 5.1).

Let B^{PU} be the *set of PU I/O busses and* $X^{PU,B_i} = \{x_{1,i} \ldots x_{n_i,i}\}$ $1 \le i \le N_{pu}$ an ordered *I/O sequence* transferred over the *i-th* bus B_i.

Each *input data x* (towards the Asic) that belongs to X^{PU,B_i} is described by the following information : (Label, Receive step, First read step, Last read step, source, destination). The label is basically an integer identifier. The receive step (denoted as *R(x)*) is the control step (c-step) at which x is received by the communication unit from the processor. The first read step (denoted as $\tau_{\min}(x)$) is the c-step of the first use of x by the PU. Similarly, the last read step (denoted as $\tau_{\max}(x)$) is the c-step of the last use of x by the PU. The source and destination are integer values to represent the I/O ports of the UCOM or the PU that produces and uses the data x.

Each *output data x* (towards the processor) that belongs to X^{PU,B_i} is described by the following information : (Label, Production step, Emission step, source, destination). The production step (denoted as *Tp(x)*) is the c-step at which x is produced by the PU. The Emission step is the c-step at which x is sent to the processor. The Other information of output data have the same meaning of those of input data. All variable descriptions presented above are resumed in Table 5.1

Asic I/O		$S=(x_1, x_2, \ldots x_n)$	*an ordered PU I/O data sequence*
		x	*I/O data x of the PU sequence S*
	Output data (x)	$T_P(x)$	*production date of the output data by the processing unit (PU)*
		$W(x)$	*emission date of the output data x from the Asic to the processor*
		$\Gamma(x)=[T_P(x), \tau_{max}(x)]$	*lifetime for the output data x in UCOM*
	Input data (x)	$R(x)$	*reception time of the input data x from the processor to Asic*
		$\tau_{\min}(x)$	*time of first read of data x by the PU*
		$\tau_{\max}(x)$	*time of last read of data x by the PU*
		$\Gamma(x)=[\tau_{min}(x), \tau_{max}(x)]$	*interval of life time for the input data x*

Table 5-1 Asic I/O data Description

5.3 Definitions:

- *Let T_c be the execution time constraint of the hardware module.*
- *Let Φ be a storage module (Φ may be a Register, FIFO or LIFO)*
- *Let x_1 and x_2 be respectively an input and output data that belong to the PU I/O data sequence S*

Definition 1: *the **PU lifetime** of I/O data is defined by $\Delta(x_i)=[\tau_{min}(x_i),\tau_{max}(x_i)]$ for input data and $\Delta(x_2)=T_p(x_2)$ for output data. To keep homogenous notations, we give an equivalent notation to $\Delta(x_2)$ as $\Delta(x_2)=[T_p(x_2),T_p(x_2)]$.*

Definition 2: *the **UCOM lifetime** of I/O data is defined by $\Gamma(x_i)=[R(x_i),\tau_{max}(x_i)]$ for input data and $\Gamma(x_2)=[Tp(x_2),W(x_2)]$ for output data. Note that $\Delta(x_i)\subset\Gamma(x_i)$ $i=1,2$.*

Definition 3: *the **Data Test Function (DTF)** of the Φ module is defined by the Boolean function Φ_S: $S \rightarrow \{0,1\}$ where $\Phi_S(x)=1$ if the I/O data x can share the resource Φ with all data of the S sequence.*

Definition 4: the ***Sequence Storage Function (SSF)*** of *the Φ module is defined by Φ^S_T: $S \rightarrow [0, Tc]_{(Modulo\ Tc)}$ where $\Phi^S_T(x_i)$ represents the time interval at which the transfer of data x_i assumed to be stored in Φ, can be occurred.*

6. I/O data storage: Allocation Procedure

In this section, we are interested in allocation and mapping of I/O data sequences to specific storage modules which are queues (Fifo) stacks (Lifo) and registers.

The allocation problem of I/O data may be solved by means of traditional techniques used for resources sharing [2] if all I/O timing are fully specified. However a full I/O timing specification can hardly be obtained because H/S communications synthesis is based on execution-time estimations, and not guaranteed to be cycle-accurate. The allocation procedure we propose here starts from the PU I/O sequence timing (fully specified by GAUT HLS) and incorporates the I/O constraints presented in section 5.1. Our goal is to assign, for each I/O data, a time interval at which the transfer to/from the processor could occur. This approach gives the designer a fine grain description of data transfer timing and such information could be exploited to refine and improve the granularity of system partitioning.

We have developed temporal transformations ***SSF*** (Φ^S_T, see definition 4) based on functional properties of each storage module Φ. Each function Φ^S_T takes into account conditions that should be fulfilled by each pair of data to be allocated to the same storage component Φ. For example, conditions relative to data storage in the same register must have disjoint lifetimes (the two intervals have empty intersection). All data to be allocated to Fifo or Lifo buffers must have the same type, e.g. all of them are input data or output data and must have disjoint lifetimes.

Here are the main steps of our procedure:

1. *First, given the PU I/O sequence S and its relative constraints (I/O timing requirements and data ordering), we determine which data that can be grouped in the same storage module. Table 6.1 shows PU conditions that should be fulfilled by each pair of data to be allocated to the same storage component (Register, Fifo, Lifo). This step produces a set C of subsequences Sk that could be stored into the module Φ_k: C={Sk}k , Sk = {xi ∈ S/ Φ^t_{Sk}(xi) =1}, Φ^t_{Sk} is the DTF function of the Φ_k module.*
2. *Second, for each subsequences S_k, some accessing constraints of the storage component Φ_k are added. These constraints (denoted as access conditions in Table 6.1) express the relationship between input and output sequence relative to each storage module. For example, the input sequence of Register or Fifo must be the same as the output sequence.*
3. *Third, we apply the* SSF function *on each data subsequence S_k. This function takes into account all the conditions discussed above and produce for each data a time interval in which the data transfer could occur. In Table 6.2, we describe the* SSF *transformation of each storage module and its respective derived constraints.*

These derived constraints can be merged with some user constraints and therefore the allocation procedure carries out, by means of some recursive heuristics we have developed, the following tasks:

(i) Constraints analysis and checking
(ii) I/O buffer sizing
(iii) determination for each I/O data a time interval.

Storage components Φ		PU timing conditions	Access conditions
Register		$\Delta(x_i)\cap\Delta(x_j)=\varnothing$	$\Gamma(x_i)\cap\Gamma(x_j)=\varnothing$
Fifo	*Input data*	$\Delta(x_i)\cap\Delta(x_j)=\varnothing$	• $\Delta(x_i)\cap\Delta(x_j)=\varnothing$ • $\tau_{min}(x_i)\prec\tau_{min}(x_j)\Rightarrow R(x_i)\prec R(x_j)$
	output data	$\Delta(x_i)\cap\Delta(x_j)=\varnothing$	• $\Delta(x_i)\cap\Delta(x_j)=\varnothing$ • $Tp(x_i)\prec Tp(x_j)\Rightarrow W(x_i)\prec W(x_j)$
Lifo	*Input data*	$\Delta(x_i)\cap\Delta(x_j)=\varnothing$	• $\Delta(x_i)\cap\Delta(x_j)=\varnothing$ • $\tau_{min}(x_i)\prec\tau_{min}(x_j)\Rightarrow R(x_i)\succ R(x_j)$
	output data	$\Delta(x_i)\cap\Delta(x_j)=\varnothing$	• $\Delta(x_i)\cap\Delta(x_j)=\varnothing$ • $Tp(x_i)\prec Tp(x_j)\Rightarrow W(x_i)\succ W(x_j)$

Table 6-1 Timing conditions relative to storage module allocation

Storage Modules (Φ)	Data Type (x_k)	SSF: $\Phi_T{}^S(x_k)$	
		Transformation	*Derived Constraints*
Register	*Input data*	$\left]\tau_{max}(x_{k-1}),\tau_{min}(x_k)\right]$ and $\Phi^S{}_T(x_1)=\left]\tau_{max}(x_N)-T_c,\tau_{min}(x_1)\right]$	$\tau_{max}(x_{k-1}) \prec R(x_k) \prec \tau_{min}(x_k)$ and $\tau_{max}(x_N)-T_c \prec R(x_1) \prec \tau_{min}(x_1)$
	output data	$\left[\tau_{max}(x_k),\tau_{min}(x_{k+1})\right[$	$\tau_{max}(x_k) \prec W(x_k) \prec \tau_{min}(x_{k+1})$
Fifo	*Input data*	$\left]\tau_{max}(x_{ni})-T_c,\tau_{min}(x_k)\right]$	$R(x_{k-1}) \prec R(x_k)$
	output data	$\left]\tau_{max}(x_{no})-T_c,\tau_{min}(x_k)\right]$	$W(x_{k-1}) \prec W(x_k)$
Lifo	*Input data*	$\left]\tau_{max}(x_{ni})-T_c,\tau_{min}(x_1)\right]$	$R(x_{k-1}) \succ R(x_k)$
	output data	$\left[T_p(x_{no}),T_p(x_1)+T_c\right[$	$W(x_{k-1}) \succ W(x_k)$

Table 6-2 SSF functions: transformations and derived constraints

7. Example: FFT implementation on Asic

Table 8.1 shows the sets of timing intervals (R(x) and W(x)) resulting from our allocation procedure on the example presented in section 3. In this case we assumed that the IOCC constraints are represented by an ordered transaction over the Asic-Processor bus. The hardware architectures of both the processing unit (PU) and the hardware interface (UCOM) are presented in Table 8.2. The necessary storage components of the hardware interface are composed of two registers and a two points FIFO.

8. Conclusion:

We presented in this paper the problem of hardware interface synthesis taking into account two types of design constraints. The first one consists in some interface specifications produced by a Cosynthesis tool. The second one is represented by I/O transfer sequence constraints resulting from the Asic's processing unit design by a high level synthesis tool GAUT. Our hardware interface design ensures : *(i) real time execution* of DSP applications *(ii) I/O data coherence*, by means of formal transformations we have developed, between Processor and PU I/O transfer sequences.

We described a global modeling of data transactions between the software (Processor) and hardware (Asic) modules and we presented a generic model of on-chip hardware interface. The necessary storage components of the interface are composed of queues, stacks and registers. The allocation procedure we described is based on timing transformation relative to each storage component. Each I/O data is assigned to a time interval at which the transfer to/from the processor could occur. As future work, we intend to fully automate the design process. We also intend to extend the hardware interface synthesis under cost constraints (surface and power consumption).

I/O data	Data type	Data transfer order	Receive step $R(x)$	First use step $\tau_{min}(x)$	Production step $T_P(x)$	Emission step $W(x)$
x0	In	1	[0, 80]	80	-	-
x1	In	2	[80,100]	120	-	-
x2	In	3	[100,120]	120	-	-
x3	In	4	[120,180]	180	-	-
y'_0	Out	5	-	-	180	[180, 260]
y'_1	Out	6	-	-	260	[260,380]
y'_2	Out	7	-	-	380	[380,460]
y'_3	Out	8	-	-	460	[460,540]
y'_0	Out	9	-	-	260	[260,360]
y'_1	Out	10	-	-	260	[360,460]
y'_2	Out	11	-	-	540	[540,620]
y'_3	Out	12	-	-	620	[620,1000]

Table 8-1 FFT Example: timing interval sets (R(x) and W(x)) resulting from the allocation procedure

	Processing Unit Architecture (PU)	Hardware Interface Architecture (UCOM)
Hardware Partition: FFT (N=4) T_c=1000 ns	- 1 adder , - 1 substractor - 1 multiplier - 8 registers - 4 multiplexers - 3 demultiplexers - 2 I/O busses	- 1 FIFO (2 points) - 2 registers - 1 multiplexor - 1 demultiplexor

Table 8-2 Hardware design results: PU and Interface architectures

References

[1] De Micheli Giovanni, M. Sami 'Hardware/Software Codesign' Ed Kluwer Academic Publisher 95.

[2] D. D. Gajski, N.Dutt, 'High Level Synthesis', Ed Kluwer Academic Publisher 92.

[3] E. Martin, O. Sentieys, H. Dubois, J.L. Philippe, 'GAUT, An Architectural Synthesis Tool for Dedicated Signal Processors', In *proceedings of EURO-DAC 93.*

[4] Towards a Multi-Formalism framework for Architectural Synthesis: the ASAR Project Asar Project members'. In *proceedings of* the 3rd International Workshop on H/S Codesign. Grenoble, Sep 1994.

[5] G. Boriello and R. Katz. ' Synthesis of interface transducer logic'. In *proceedings of ICCAD*, 1987.

[6] P.H. Chou, R.B Ortega, G .Boriello. 'The Chinook Hardware/Software Co-Synthesis System ' In proceeedings of the ISSS. 1995.

[7] D. Filo, D.C. Ku, C.N. Coelho, and G. De Micheli. 'Interface optimization for concurrent systemls under timin constraints', *IEEE Trans. on VLSI systems*, Pages 268, -281, September 1993.

[8] R.K.Gupta, C.N. Coelho, and G. De Micheli. 'Synthesis and simulation of digital systems containing interacting hardware and software components'. In Proceedings of DAC 1992.

[9] S. Narayan and D.D Gajski, 'Protocol generation for communication channels' In proceedings of DAC, pages 547-548, 1994.

[10] S. Narayan and D.D Gajski, 'Synthesis of system level bus interfaces' In proceedings of DAC, pages 547-548, 1994.

[11] C. N. Coelho Jr & al 'Redesigning Hardware-Software Systems'. In *proceedings of* the 3rd International Workshop on H/S Codesign. Grenoble, Sep 1994.