BEHAVIORAL DESCRIPTION MODEL BDM FOR DESIGN SPACE EXPLORATION: A CASE STUDY OF HIS ALGORITHM FOR MC-CDMA SYSTEM

F. Thabet, P. Coussy, D. Heller , E. Martin

LESTER – Université de Bretagne Sud – BP 92116 - 56321 LORIENT Cedex, France
{firstname.lastname}@univ-ubs.fr

ABSTRACT

The design of complex Digital Signal Processing (DSP) hardware accelerators implies to minimize architectural cost and to maximize timing performances. Exploring different communication architectures and timing behaviors is thus a key step in the modern system design flows. In this paper, we present a methodology that permits the design space exploration of DSP applications. The proposed approach consists in embedding a sequential function into a Behavioral Description Model (BDM) that includes a set of I/O and control processes. Communication architectures and timing behaviors (I/O scheduling, I/O parallelism…) can be varied and easily explored by adding I/O and synchronization code into the dedicated concurrent processes while keeping the functionality description unchanged throughout the exploration step. A high-level synthesis tool is next used to generate the architectures that respect the design constraints. We show the interest of our approach in the case study of a Hyper-plane Intersection and Selection HIS algorithm for MC-CDMA system.

Index Terms— System analysis and design, Design automation, High-level synthesis.

1 INTRODUCTION

Consumer electronic devices are more and more oriented towards multimedia and communication applications. The design of real-time embedded SoC architectures is currently achieved by using system-level descriptions, electronic-system level ESL tools and by re-using pre-designed IP-cores. Typical SoC architectures include several processors, memories, I/O devices, communication media and dedicated HW accelerators. Indeed, the increasing complexity, the low-power design constraints and the growing data rates of applications from the digital signal processing (DSP) domain still often requires hardwired implementations to be used as dedicated accelerators in the final system-on-chip (SoC). In [1] authors introduced the Interface-based design methodology. The underlying idea of this work is to split the communication and the functionality to allow an independent refinement of their timing behavior. The SystemC methodology [2] is based on this concept and implements it with the Interface Method Call (IMC) mechanism. In [3] the authors propose a similar approach that uses the SpecC language. Many commercial tools (such as Cadence NCSystemC [5], CoCentric Synopsys System Studio [6], CoWare N2C [7]) have started to support system exploration at the Transactional Level Modeling TLM level. However, while these approaches target software dominated design, they do not address precise timing issues and do not define how to refine and to model detailed communication architectures and behaviors which are required for the design space exploration of hardware DSP functions (see [8]). Some other approaches are more oriented toward the hardware design of DSP applications. Thus, in [9] and [10] the authors propose approaches that use Matlab/Simulink/Stateflow tools for the system specification and that produce a VHDL RTL architecture of the system. Based on hardware macro/generators that use the "generic"/"generate" mechanisms, the synthesis process can be summarized as a block instantiation [11] which does not permit a real design space exploration.

In [12], [19], [20], we show the interest of reusing algorithmic IP cores to design hardware accelerator of the DSP domain. Based on high-level synthesis techniques, the approach aims at synthesizing an algorithm (a C function) by taking into account integration constraints: application rate, technology, I/O timing diagram, memory mapping... The proposed approach does not however propose any support for the exploration of the design constraints. In [18], we introduce a behavioral description approach (BDM) that allows to explore, by easily modifying some architectural and I/O timing constraints, the design of complex DSP systems.

This paper is organized as follows: Section 2 introduces the problem of communication refinement and specification in the design space exploration. Section 3 briefly reminds the main concepts of the BDM approach. In section 4, we show the interest of our approach by presenting the design space exploration of a Hyper-plane Intersection and Selection HIS algorithm [16].

2 PROBLEM FORMULATION

Let us consider the typical design of a DSP application including a function *add_v* that adds two vectors (A and B) of N elements. At the untimed functional level (UTF), the application is developed in a C or MATLAB like environment. The intrinsic concurrency of the application is next exhibited through a process network where modules (that embed the functions) are connected by FIFOs. Each computation is processed in zero time using blocking read functions which arguments are structured data: vector type (1D-array) in our *add_v* example. FIFO sizes are then bounded and timing information is added for the

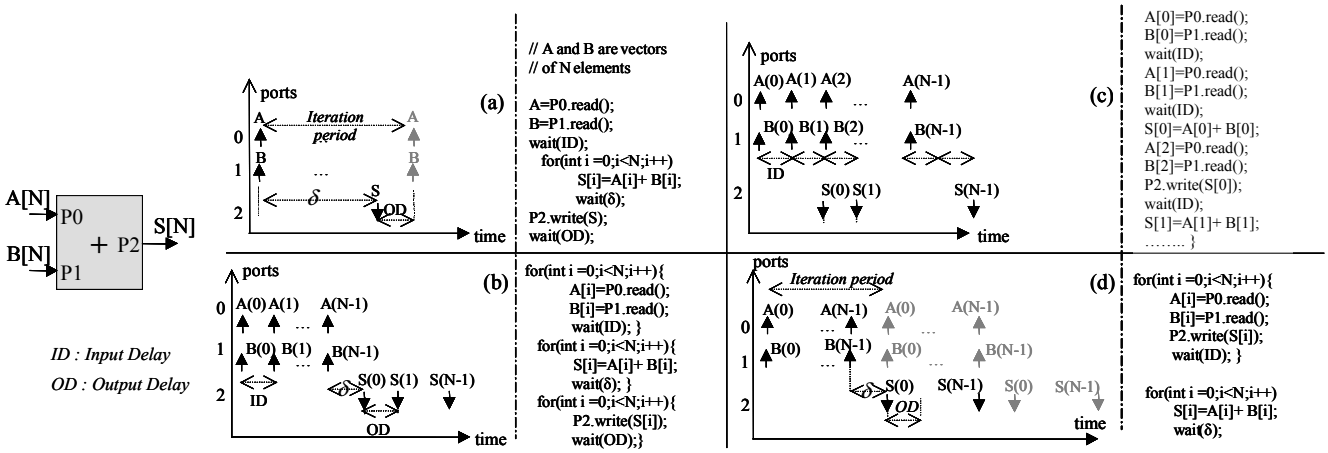


Figure 1: Communication timing behaviors of an adder component: (a) sequential behavior with abstract data type, (b) sequential behavior with refined I/O data type, (c) pipelined behavior, (d) interleaved behavior with refined I/O data type.

computation latency of each function by using a *wait* statement (Timed Functional description TF). Communication delays are next added for each I/O data transfer by using the same annotation technique. Note that I/O data types have not yet been refined and are still referred as atomic structures. At the Bus Cycle Accurate (BCA) level, the communications are realized through cycle accurate component interfaces. Read and write operations to memory use physical addresses to access fine grain data (scalar). This implies that both the data types and the timing refinements have been done which is true only for pre-designed core. In this context, the main problem confronted by the designer is how to easily explore communication architectures and timing behaviors of components that compose a complex system, to identify an ideal design point in earlier design stages. Traditional approaches describe the reading, the processing and the writing operations inside a single process which limits the capability to efficiently and quickly explore different behaviors (see Figure 1.a and Figure 1.b). However, refining a sequential behavior description (a pure C function for example) into a concurrent one requires the designer to modify its specification source code. For simple algorithms, a basic behavior like the interleaving between reading, processing and writing operations, with or without pipeline, can be modeled by modifying the code as presented in Figure 1.c and Figure 1.d. However, this task becomes rapidly too complex, time consuming and error prone for real world examples. This complexity needed to describe concurrent, overlapping and pipeline communication behaviors between I/O ports and computation, is not acceptable in the context of efficient design space exploration which requires rapid and safe modifications.

A new approach that allows a simple and rapid exploration of communication architectures and timing behaviors of DSP functions is thus needed. We proposed, in [18], a refinement model called *Behavioral-Description-Model (BDM)* which is summarized in the next section. For a given communication architecture (i.e. number of I/O ports which can also easily be modified), the BDM approach allows the designer to model several timing behaviors: sequential (first reading all the inputs, then computing and finally writing all the outputs), interleaved (reading some inputs, computing and writing some outputs), and pipelined, without modifying the original computation function. Moving from one behavior to one other can be done by simply modifying I/O processes in which the designer sets the delay values, port assignment and synchronization between ports. Moving from an abstraction level to an other can easily be realized by refining the port type and the I/O data granularity (matrix, vector or scalar type) and/or set or reset delays (Untimed Functional vs. Timed Functional). All these refinements and behavioral specifications are done without modifying the computation function. In the next section, we present an overview of our BDM approach.

3 BDM APPROACH OVERVIEW

A behavioral description model is to be used in the context of Process Network PN to describe a concurrent DSP application. Hence, a BDM has a set of read and write ports connected to channels allowing the BDM components to be chained and to communicate with each other. The behavior of a BDM consists of reading inputs *I,* writing outputs *O* and executing a computation function which uses intermediate storage variables *SV*. Each BDM includes:

- A set of input, output and aging shared-variables respectively named *isv*, *osv* and *asv*, with *SV={isv, osv, asv},*
- A set of input processes $IP=\{ip_0,...,ip_N\}$, which allows the variables of the set *isv* to be assigned from the input values (belonging to *I*) coming from the input channels,
- A set of output processes $OP = \{op_0, op_1,..., op_M\}$, which allows the variables of *osv* to be computed and next written as output values (belonging to *O*) into the output channels,

- A set of two control processes *CP = {icp, ocp}* composed of one input control process *icp* and one output control process *ocp* which control and synchronize *IP* and *OP* execution and which handle the state of the shared variables *SV*.
- One computation function f_{comp} that determines the functionality of a BDM component and gives results to the output data in *osv,* depending on the input storage variables of *isv* and aging storage variables of *asv.*

The BDM processes can exchange control events which allow the designer to specify reactivity, synchronization... between inputs and/or outputs. Figure 2 presents the possible architecture of the BDM for the *vector adder* component described in fig. 1.

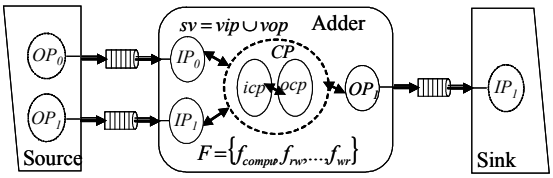


Figure 2 : BDM architecture of adder component.

The traditional specification model that uses only one process which includes all the read, compute and write operations is thus decomposed into a set of concurrent processes, which allow the parallel execution of read, processing and write tasks. Note that in our model, only one computing function exists. Communication architectures and timing behaviors (I/O scheduling, I/O parallelism…) can be varied and easily explored by adding I/O and control instructions into the dedicated concurrent processes while keeping the functionality description unchanged throughout the refinement steps.

Behavioral Description Model has been implemented using the SystemC language. A BDM component is a SystemC module that has a set of I/O ports through which it connects to other modules via FIFO channels. Designer can thus easily model a complex digital signal processing DSP application as a process network PN. Input, output and control processes are implemented using *SC_THREAD* processes. Read, write and control primitives are provided as polymorph functions regrouped in a C++ class named "BDM_toolbox" from which each BDM module inherits. Synchronization between processes is realized using the dynamic event primitives provided by SystemC. Read and write primitives are instantiated into I/O processes and allow a BDM module to access communication channels via its I/O ports. The SystemC read and write methods, that we provide in the toolbox, also allow the generation of both (1) simulation traces relative to I/O transfers and (2) design constraint files for high-level synthesis.

More details about the BDM approach can be found in [18].

4 A CASE STUDY OF A HIS ALGORITHM FOR MC-CDMA SYSTEMS

In this section, we show the BDM efficiency in exploring communication architectures and timing behaviors, and their impact on the final design, for an industrial application: an *Hyper-plane Intersection and Selection HIS* algorithm used in MC-CDMA systems (see [16] for algorithm details).

An efficient sub-optimal algorithm, called HIS (Hyper-plane Intersection and Selection) detection algorithm [16], has been proposed to solve the problem of joint detection of *K* users in a MC-CDMA system. This algorithm has three characteristics very attractive for practical systems. Firstly, it has nearly optimal performance. Secondly, it has a low computational complexity ($O(K^2)$ multiplications and $O(K^3)$ additions). Third, the algorithm has an inherent parallelism level which allows us to predict an efficient hardware implementation.

4.1 HIS Application overview

In the following, we give all steps of the HIS detection algorithm with parameters *D* (the number of studied dominant axis noise) and *M* (the number of selected candidates):

Input: the received vector *Y*, the channel matrix *H*.

Output: a nearly Maximum Likelihood (ML) solution X_{sol} of (3).

$$\hat{X}(t)_{ml} = \arg\min_{X\in\xi} \|Y(t) - H(t)\times X\| \quad (3)$$

Pre - processing:

- Find the *D* singular vectors $(v_p)_{p=1...D}$ associated to the smallest singular values of the channel matrix *H*. Compute the $D \times N$ values $(1/v_p(i))$ and compute H^{-1}. Note that this pre-processing step is performed only once for every new channel matrix *H*.
- Calculate an initial sub-optimal solution ρ using ZF (Zero Forcing) algorithm.
- Generate the *D* references lines $\{\Delta_1,..., \Delta_D\}$ defined by the point X_{ZF} and vectors $\{v_p\}_p$=1..*D*.

$$\Delta_k = \left\{ z \in \Re^n / z = \rho + \alpha.\mathbf{v}_k, \quad \alpha \in \Re \quad (4) \right\}$$

Step 1: Geometrical Intersection (GI)

For each lines Δ_k, find all intersection points between this line and all hyper-plane *P* defined as **(5).** There are 2n points to be calculated as shown in Figure 3. Finally project all intersection point on $\{-1,1\}^n$, suppress redundant points and generate candidates points Ip_k.

$$P = \bigcup_{i=1}^{n} \left[\left\{ z \in \Re^n / z(i) = 1 \right\} \cup \left\{ z \in \Re^n / z(i) = -1 \right\} \right] \quad (5)$$

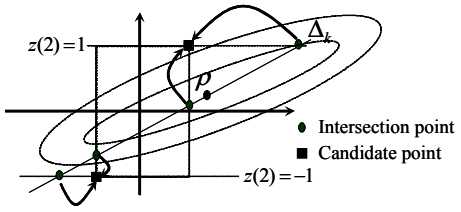


Figure 3: example of HIS algorithm for D=1 and n=2.

Step 2: Evaluation (Eva)
For each direction *k (k=1….D),* evaluate the cost function, second member of relation (3), for all Ip_k.

Step 3: Sort and Select (SS)
For each Ip_k, extract the *M* candidate points having the lowest cost function.

Step 4: Search and Select (GS)
Use each candidate point $b_{k,t}$ (*k=1…..D and t =1…..M*) as a starting point of a greedy function (local search function) and select the Quasi-optimal solution X_{sol} for problem **(3)**.

4.2 Design space exploration

We present in this sub-section the design space exploration we did by using our BDM models to specify several communication architectures and several timing behaviors for each sub-block that composes the HIS application. We first described the *HIS* algorithm as a set of purely sequential C functions, each performing one of the four algorithm steps described in the previous section. We next encapsulated each function into a *BDM* SystemC object as shown in Figure 4. This new description of the application was an untimed and concurrent specification of the HIS application that included four BDM components: *GI* (*Geometrical Intersection)*, *EVA (Evaluation)*, *SS (Sort and Select)* and *GS (Sort and Select)*. Each component had a set of input ports and output ports connected to SystemC FIFOs. The first sequential specification written in C language has thus been parallelized thanks to the concurrency introduced by the set of BDM modules. We then refined at different abstraction levels this parallel specification and defined several timing behaviors. This allowed us to explore different architectures and to analyze their impact on the timing performances and the area of the HIS system. The communication constraints specified for the simulation were next used for the high-level synthesis of each block.

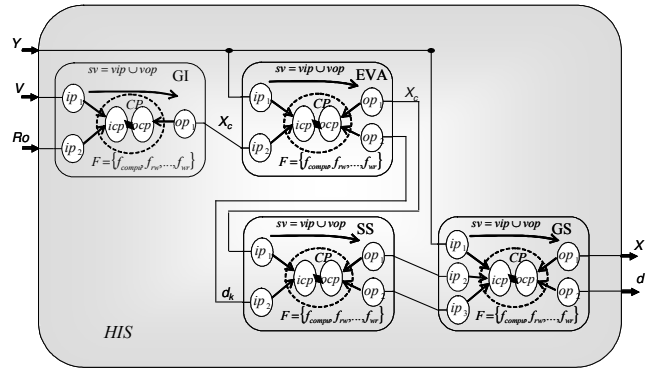


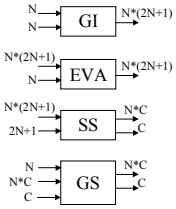
Figure 4: A hierarchical HIS BDM architecture.

At both the untimed and timed coarse grain BDM levels (UTF-CG and TF-CG), the read (write) primitive gets (puts) all the scalar data simultaneously since the FIFOs handle data with structured types (vectors/matrix of integers). We next refined the type of both the FIFOs and ports to handle integers (at the fine grain description level FG), and we specified the scheduling of inputs and outputs by using *for* loops instructions in the input and output processes. At the TF-FG level, four timing behaviors have been specified, when possible, for each component:

- one sequential behavior with no overlapping between I/O: the first output data is produced in an output FIFO only when all the inputs have been read like in a Synchronous Data-Flow SDF actor. Thus, the original sequential function is executed only one time.
- one "interleaved" behavior where the I/O were overlapping: some results are written in the output FIFOs before all the inputs had been read. Until all the input data have not been read, the computation function is executed each time a result has to be produced.
- One "pipelined" behaviors with I/O data overlapping where the component starts input data read for the next algorithm iteration before finishing the production of output data of the current iteration.
- One "pipelined" behaviors without I/O data overlapping.

These four types of behavior have been specified by using the parameterized communication primitives that we provide in the SystemC toolbox.

For each timing behavior of each BDM component, an I/O trace file has been automatically generated during the functional SystemC simulation. These traces have next been used as I/O timing constraints for high level synthesis of the components. Synthesis results have been obtained by using the HLS tool GAUT [17] with a 10ns clock period constraint. This allowed us to estimate the impact of different communication strategies on the final hardware implementation of the GI, EVA and GS components. SS which is an interleaver, has been used for simulation purpose but was not synthesized in this experiment. Table 1 presents the results of the design space exploration we performed for the HIS algorithm. The algorithmic parameters of each components were the matrix dimension N=4 and the number of selected candidates C=4. The synthesis of each block was performed for the three following I/O sampling rates: 1 data/4 cycles, 1 data/cycle and 2 data/cycle. For each I/O sampling rate, each of the 4 types of behavior (sequential, interleaved…) has been synthesized whenever possible (according to the data dependencies of the algorithms) while targeting the best timing performance (lowest latency and iteration period, maximum throughput).
When the behavior is not pipeline, the iteration period must be greater or equal to the latency which depends on the sampling rate. In this case, the I/O overlapping (when it is possible, see table1) reduces the total latency and thus enhances the maximum throughput but leads to larger component areas. The overlapping ratio between I/O depends on the data dependencies of the algorithm and on the I/O schedule. In Table 1 the interleaved behavior of the EVA block gives a larger area than the sequential behavior since the architecture must be more parallel in order to respect a shorter latency. The overlapping ratio between I/O depends also on the sampling rate since a slow sampling rate will result in a large timing delay during which operations can be scheduled. Hence, the EVA



		GI				EVA				GS			
		Latency (ns)	Iteration Period(ns)	Area (u²)	Throughput (Kdata/s)	Latency (ns)	Iteration Period(ns)	Area (u²)	Throughput (Kdata/s)	Latency (ns)	Iteration Period(ns)	Area (u²)	Throughput (Kdata/s)
Rate 1: 1 data/4cycles	Sequential	9000	= 9 000	31 619	888	2 300	= 2 300	34 813	17 391	3 750	= 3 750	15 032	6 400
	Interleaved	-	-	-	-	1 700	= 1 700	42 187	23 529	-	-	-	-
	Pipelined	9000	310	47 380	25 806	-	-	-	-	3 750	3 000	30 261	8 000
Rate 2: 1 data/cycle	Sequential	2400	= 2400	26 601	3 333	1 550	= 1550	11 980	25 806	2 830	= 2 830	13 484	8 480
	Pipelined	2400	300	41 886	26 666	-	-	-	-	-	-	-	-
Rate 3: 2 data/cycle	Sequential	1 380	= 1 380	27 337	5 797	1 550	= 1550	10 694	25 806	2 600	= 2 600	35 951	9 236
	Pipelined	1 380	300	44 446	26 666	-	-	-	-	-	-	-	-

Table 1: The HIS component behaviors.

interleaved behavior is no more possible for rate2 and rate3. When the behavior is pipelined, the throughput depends on the iteration period which depends itself on the data dependencies of the algorithm. The area depends on the iteration period but also on the latency which is function of the I/O sampling rate (see table GI and GS in table1).

Hence, the architecture performances of digital processing applications rely on a set of parameters that are interdependent which needs a dedicated design space exploration environment.

5 CONCLUSION

In this paper, we described a behavioral description model BDM which allows the design space exploration of DSP applications. The proposed approach consists in encapsulating a sequential function into a BDM object which also embeds input, output and control processes. By using these processes, the designer defines a communication architecture for which he next specifies and refines the timing behavior. The BDM approach which has been implemented in SystemC has been used in this paper to highlight the complexity of the design space exploration task. The system level modeling and refinement of a HIS system has been realized using a set of BDMs and has next been synthesized by using a high-level synthesis tool.

We are currently working on the development of a tool named DsXplore that automates the encapsulation of a pure C function and the generation of its corresponding SystemC BDM. Through a graphical interface (an Eclipse plug-in [21]), this tool allows the specification and the exploration of BDM networks. This API will be next connected to our HLS tool GAUT that already use the design constraints generated by the BDM models.

6 REFERENCES

[1] A. Rowson and A. Sangiovanni-Vincentelli. "Interface-based design". in *Proc. of the 34th Design Automation Conf. (DAC-97)*, 1997

[2] T. Grotker, S. Liao et al, "System Design with SystemC", Kluwer Academic Publishers, USA 2002.

[3] L. Cai and D. Gajski. "Transaction Level Modeling: An Overview". *In Proc. of the Int. conference CODES+ISSS, 2003*

[4] A. K. Deb, et al. "System Design for DSP Applications in Transaction Level Modelling Paradigm" in *Proc. of the 42th Design Automation Conf. (DAC-04)*, 2004

[5] Cadence NCSystemC. http://www.cadence.com.

[6] CoCentric, System Studio.http://www.synopsys.com

[7] CoWare. http://www.coware.com.

[8] A. Donlin "Transaction level modeling: flows and use models", *In CODES+ISSS'04, Proc. of the 2nd IEEE international conference on Hardware/Software co-design and system synthesis, 2004*

[9] L. Reyneri et al. "A hardware/software co-design flow and IP library based on Simulink", *in Proc. 39th Design Automation Conf. (DAC),* 2001.

[10] J. Ruiz-Amaya, et al., " MATLAB/SIMULINK-Based High-Level Synthesis of Discrete-Time and Continuous-Time SD Modulators", *In Proc. of Design Automation and Test in Europe DATE'04,* 2004

[11] Codesimulink,http://polimage.polito.it/groups/codesimulink.html

[12] P. Coussy, E. Casseau, P. Bomel, A. Baganne, and E. Martin, "A Formal Method for Hardware IP Design and Integration under I/O and Timing Constraints", ACM Transactions on Embedded Computing System (TECS), 2006, vol 5, N° 1, pp. 29-53.

[13] Edward A. Lee and Alberto Sangiovanni-Vincentelli, "A Framework for Comparing Models of Computation", *IEEE Transactions on Computer Aided Design (TCAD)*, Vol. 17, No. 12, December 1998

[14] H. Lehr D. Gajski "Modeling Custom Hardware in VHDL", Technical report ICS-99-29, July 6, 1999.

[15] P. Bomel et al. "Synchronization Processor Synthesis for Latency Insensitive Systems", In Proceedings of Design Automation and Test in Europe DATE'05, 2005.

[16] A. Nafkha, C. Roland et al. "A Near-Optimal Multiuser Detector for MC-CDMA systems Using Geometrical Approach", *ICASSP'05*.

[17] GAUT - HLS Tool for DSP, http://web.univ-ubs.fr/gaut/

[18] Thabet F., Coussy P., Heller D., Martin E., "Design Space Exploration of DSP Applications Based on Behavioral Description Models", *In IEEE Workshop on Signal Processing Systems (SIPS), 2006*.

[19] P. Coussy , D. Gnaedig, and al., " A Methodology for IP Integration into DSP SoC: A Case Study of a MAP Algorithm for Turbo Decoder", *In Proc. of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2004*.

[20] P. Coussy, G. Corre, P. Bomel, E. Senn, E. Martin, "A More Efficient And Flexible Dsp Design Flow From Matlab-Simulink", *In Proc. of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2005*.

[21] Eclipse - an open development platform: www.**eclipse**.org