14th European Signal Processing Conference (EUSIPCO 2006), Florence, Italy, September 4-8, 2006, copyright by EURASIP

FAST PROTOTYPING OF DIGITAL SIGNAL PROCESSING SYSTEMS BY MEANS OF A MODEL-BASED CODESIGN ENVIRONMENT

Leonardo Maria Reyneri[*] and Fabio Ancona[**]

[*] Department of Electronics, Politecnico di Torino
C.so Duca degli Abruzzi, 24, 10129 TORINO (ITALY)
phone: + (39) 011 5644038, fax: + (39) 011 5644099, email: leonardo.reyneri@polito.it
web: polimage.polito.it/~lmr

[**] Sundance Italia, s.r.l.
Corso XXV Aprile 55/3, 16040 S. Salvatore di Cogorno (GE) (ITALY)
phone: + (39) 0185 385193, fax: + (39) 0185 385370, email: fabio.a@sundance.com
web: www.sundance.com

ABSTRACT

This paper presents a novel tool, based on Simulink™, for model-based high-level HW/SW codesign of high-performance digital signal processing systems. The tool has been tailored to support HW/SW configurable platforms, in particular those from Sundance Microprocessor Technology [1].

1. INTRODUCTION

Modern digital signal processing systems (**DSPS**) require increasingly performant HW supports to be able to meet tight speed, power and accuracy requirements.
Uptodate and cost effective DSPS's often require an ad-hoc integration of SW, digital and analog HW subsystems (**hybrid systems**), which often have to interact with an external environment providing input signals to and receiving output signals from the DSPS.
The design of hybrid DSPS's usually requires the development, testing and tuning of a set of *configuration codes* (for instance, one or more C --> EXE for the SW subsystem(s); one or more HDL --> fusemap for FPGA(s), schematic diagrams for analog subsystems).
Each configuration code usually has its own design flow, made of a different language, simulator, compiler and debugger. The integration of the different subsystems therefore requires, during the desing phase, the cosimulation of different codes, usually written in different languages (mostly, C, VHDL, SPICE). Unfortunately at present there is no commercial simulator, which integrates all these capabilities.

1.1 Model-based Hybrid Codesign and Cosimulation

As an alternative to the design flow outlined above, modern codesign environments are based on a common language, which allows the description, simulation, debugging and optimization of hybrid systems as a whole.
There are basically two families of such environments:

- **textual**, which use textual languages, like System-C, Handel-C, Precision-C. Textual languages are usually flexible and powerful, although they usually require a deep knowledge and experience in the field. The optimization phase with textual tools may be rather difficult, especially when interfacing analog environments
- **visual**, which use visual languages, like Ptolemy and Simulink. Visual languages are usually more user-friendly, as they can be mastered nearly by everybody, including also non-electric engineers.
  In particular, Simulink™ [2] is a well-known environment, which is commonly used in a large number of engineering fields and allows high-level description and simulation of virtually any system, among which DSPS's.
  Yet, Simulink as such is not a HW description language, therefore it does not support the generation of FPGA configuration codes.
  There are a number of add-ons to Simulink (for instance, Xilinx's System Generator [3]), which partially solve this problem, although they only support HW subsystems (no SW code can be generated); and only the compilation of proprietary FPGA's; in addition, they are not really high-level description languages, but rather a visual register-level environment to describe purely digital circuits.

Existing commercial tools are therefore seldom satisfactory for the design of high-performance hybrid DSPS's, creating a large gap between the performance and capabilities offered by commercial fast-prototyping boards and the ease of use which one would appreciate from them
This paper presents CodeSimulink/SMT6040 [4,5], a novel HW/SW codesign environment based on MATLAB and Simulink, which has been developped to fill the gap between the requirements of engineers and the capabilities of hybrid fast-prototyping HW/SW platforms, such as those manufactured by Sundance Microprocessor Technology [1] (see an example in fig. 1).

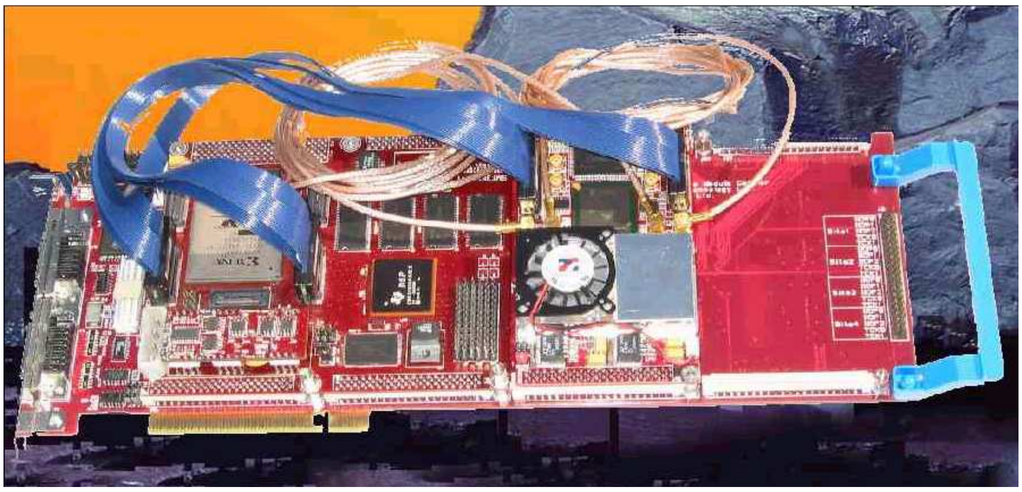


Figure 1 Sundance's Software Defined Radio kit (SMT8036E), hosting of a Texas Instrument 1GHz DSP, a Xilinx VIrtex-II FPGA, two 16-bits DAC' and two 14-bits ADC's

2. CODESIMULINK/SMT6040 AND FAST HW/SW PROTOTYPING

As mentioned above, at present, most existing design techniques for digital HW systems are based on either schematics at gate and register levels or on HW description languages like Verilog and VHDL, on accurate but slow simulators and on high performance synthesis and place&route tools.

Therefore, overall system simulations are not feasible due to the different languages used for each part (usually, VHDL for digital HW, C for SW, Spice for analog circuits, etc.). Design, simulation, tuning and optimization are often quite long steps, causing a long time-to-market, with a huge economical impact on the project.

Optimal cost/performance ratio and a short time-to-market can only be achieved by:

i) designing and simulating the whole system (both software, digital and analog parts) in its entirety, together with its interactions with the external world (for instance, mechatronical systems, human interaction, etc.);

ii) performing functional and system-level architectural optimization (for instance, signal resolution, sample rate, filter sizes, etc.);

iii) properly partitioning the system into digital HW, analog HW and SW for optimal cost/performance;

iv) automatically compiling and assembling: digital HW subsystems first into one or more VHDL code(s), then into FPGA fusemap(s); SW subsystems first into one or more C code(s), then into executable(s);

v) downloading fusemaps and executable into a fast prototyping platform, such as one of the Sundance products shown in fig. 1 [6].

This approach, which is usually called HW/SW *codesign*, was unaffordable until a few years ago, due to the lack of appropriate languages, simulators and tools.

Another drawback of the few commercial codesign tools is that they require a specific experience in electronic design, which is not always available and, they cannot easily model analog subsystems, like ADC's, DAC's, sensors, etc.

To overcome these and a few other problems, a novel tool has been developed at Polytechnic of Turin in cooperation with Sundace Italy, called *CodeSimulink/SMT6040* [4,5], which is based on MATLAB and Simulink™ [2] and allows a straightforward and accurate design, simulation and tuning of mixed-signal systems.

CodeSimulink/SMT6040 is more user-friendly than other environments, as it allows to design in a homogeneous way, with a common description and simulation language, any data-dominated system made of digital HW, analog HW and SW parts, together with any external system, like sensors, RF channels, etc., therefore it is perfectly suited to develop high performance DSPS's.

A DSPS can be designed as a Simulink model, therefore in a well-known visual environment. The user can then easily partition the system into, for instance, digital HW, analog HW and SW blocks, by flagging each block appropriately, by means of the dialog box shown in fig. 2. Each block assumes a different color according to its implementation.

System-level functional simulations can then be carried on:

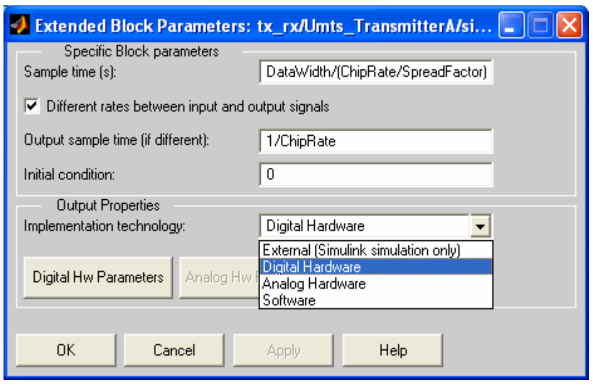
i) to decide whether the algorithm works as desired;

ii) to tune functional parameters (for instance, filter length, sample rates);

iii) to optimize architectural parameters (for instance, parallelism, resolution, virtualization, etc.) with the goal of optimizing a given cost function (for instance, complexity, cost, speed, power consumption).

HW/SW partitioning and architectural exploration (steps ii and iii above) can also be performed automatically by means of all the MATLAB capabilities and toolboxes.

The designer is also given the chance to specify, during the planning phase, which kind of prototyping board or FPGA is targeted, from a list of commercial boards or devices. All most popular Sundance boards are supported.


Extended Block Parameters: tx_rx/Umts_TransmitterA/si...

Specific Block parameters

Sample time (s): DataWidth/(ChipRate/SpreadFactor)

[✓] Different rates between input and output signals

Output sample time (if different): 1/ChipRate

Initial condition: 0

Output Properties

Implementation technology: Digital Hardware

External (Simulink simulation only)
[✓ Digital Hardware]
Analog Hardware
Software

Figure 2 CodeSimulink/SMT6040 dialog box to select block parameters and implementation technology. Other implementation-specific parameters (e.g. resolution parameters) are in another dialog box (not shown).

In addition, as CodeSimulink/SMT6040 runs under MATLAB-Simulink™, it has a tight interaction with the MATLAB Workspace. MATLAB can therefore be used, for instance: i) to supply input values/waveforms during simulation; ii) to evaluate, plot and post-process output signals (for instance, to compute spectra); iii) to interactively modify and assess the effects of functional parameters on system performance, etc.

CodeSimulink/SMT6040 supports numeric real-valued signals (either scalars or vectors or matrices), both integer, fixed- and floating-point of any resolution. In particular, low-resolution (less than 8 bits) floating point formats are quite effective, as they often offer an excellent cost/performance. Further details on CodeSimulink can be found in [4,5].

2.1 Compilation Model

As soon as performance are considered satisfactory (for instance, speed is enough, size fits into the chosen device, cost is compatible with the market, etc.), the system can automatically be compiled into C code and/or executables, for SW, and VHDL and/or fusemap, for HW, and downloaded onto the DSP and/or FPGA.

An appropriate *compiler block* must be placed into the model. The pression of a button on it triggers the activation of a sequence of automatic steps:

- block parameter calculation, as CodeSimulink/SMT6040 has a more powerful parameter handling than Simulink itself;
- hierarchy flattening;
- properly splitting the model into digital HW, analog HW and SW subsystems; HW/SW interfaces are duplicated;
- generation of a VHDL file from the digital HW subsystem; subsequent activation of the chosen synthesizer and place&route tool (e.g. Mentor Graphics Spectrum, or Xilinx ISE Foundation);
- activation of Real Time Workshop (a commercial Simulink --> C compiler) on the SW subsystem;
- all other blocks are disregarded.

2.2 Vectors and Matrices

The capability of handling vectors and matrices is a unique feature of CodeSimulink/SMT6040. Several DSPS's deal with groups of signals which have to be handled as a whole and can conveniently be arranged into either vectors or matrices. Examples of vectors are: the left and right channels of stereo sound; the channels of multi-antenna arrays. Examples of matrices are the images.

All Simulink-based HW generators (such as Xilinx System Generator [3]) do not treat vectors; the user must then consider them either as a time sequence or as the parallel of independent signals and take the burden to synchronize/duplicate vector/matrix components.

CodeSimulink/SMT6040, similarly to Simulink, considers vectors and matrices as a single vectorial signal, therefore it either sequences or duplicates vector/matrix components transparently to the user. This feature becomes increasingly useful when HW and SW subsytems interact, as described in next section.

Vectors and matrices are seen as a wire in the CodeSimulink/SMT6040 model, but they can be implemented in HW in either of two ways, transparently from the user: *serial*, when vector components are transferred sequentially on the same physical connection and computation resources are time-shared accordingly; *parallel*, when vector components and computing resources are parallelized, therefore automaticlaly replicated.

2.3 HW/SW Interfaces

The interconnection between HW and SW subsystems is the most critical issue in model-based codesign environments. A HW/SW interface has to handle, in a user-friendly manner, the interconnection between HW and SW subsystems, while keeping it as transparent to the user as possible.

Since the major task of a HW/SW interface is to transfer information between two environments with completely different computation and timing paradigms, it is worth to briefly outline both of them:

- HW subsytems are intrinsically parallel, in the sense that each physical resource remains allocated forever and resources are synchronized together either by a common clock or by a dataflow protocol, either synchronous or asynchronous;
- SW subsystems are intrinsically sequential, in the sense that a unique common resource (the arithmetic and logc unit), or a small set of them, is sequentially used by the algorithm steps and synchronization is intrisically guaranteed by the sequentiality of processes or by appropriate scheduling.

As a consequence of that, HW and SW subsystems may have a significantly different speed and/or throughput. Thus a HW/SW interface has to take care of transferring pieces of information across the boundary between two such different worlds, while keeping data consistency (that is, synchronization) and data structure (in particular, vectors and matrices).

We conceived two major communication paradigms, both compatible with the above constraints, that we call:

- **synchronous transfer**, whenever data are neither lost nor duplicated. That is (depending on transfer direction):
    a. the SW waits until the HW has new data to transfer, or it is ready to receive new data;
    b. the HW waits until the SW has new data to transfer, or it is ready to receive new data;
    c. in addition, the first, second, ... element of a vector/matrix for the HW shall be seen as the first, second, ... element also by the SW and viceversa (see also section 2.2).

  This transfer paradigm implies that the data transfer resource (made of a piece of C code on the DSP and a piece of VHDL code on the FPGA) may block both the HW and the SW. On the SW side this is a common concern, which is intrinsically solved by the SW sequentiality, while on the HW side it may create a number of non-trivial problems, since one should add some data validation protocol to guarantee that a HW block processes data if and only if these are valid.

  CodeSimulink/SMT6040 solves this problem in a transparent way, by using a dataflow computation paradigm (either synchronous or asynchronous) also for HW subsytems, therefore the user need not take care of the synchronization burden;
- **asynchronous transfer**, whenever data may be either lost or duplicated, that is (depending on transfer direction):
    a. the source of data (on one side of the interface; either SW or HW) writes data whenever it has a new one. The receiver side may read it whenever it needs. Therefore the receiver may repeatedly read the same data (duplication) if it is faster than the source; or it may read one data out of many, if it slower than the source (loss);
    b. yet, the first, second, ... element of a vector/matrix for the HW shall still be seen as the first, second, ... element also by the SW and viceversa (see also section 2.2).

  This transfer paradigm implies that the transfer resource (made of a piece of C code on the DSP and a piece of VHDL code on the FPGA) can block neither the HW nor the SW. There is therefore no need for synchronization, altough an appropriate number of registers is needed to store data, yet data consistency must be maintained.

  CodeSimulink/SMT6040 solves this problem in a transparent way, by inserting automatically an appropriate number of registers and storing/retrieving data according to vector/matrix structure.

HW/SW interfaces are a prerogative of CodeSimulink/SMT6040, which is the only tool based on Simulink which can cogenerate interacting HW and SW subsystems starting from a unique HW/SW model.

An example of all that in the field of Software Defined Radio (**SDR**; a particular case of DPSP's) is given in next section.

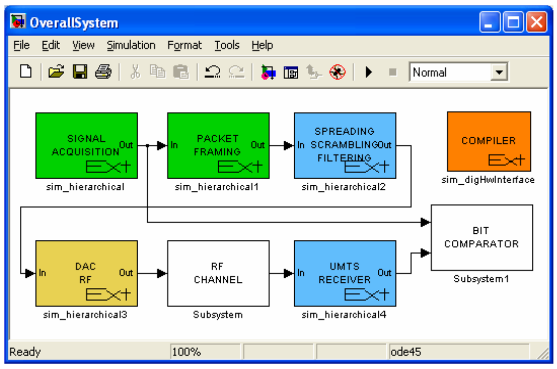


Figure 3 An application example: a complete UMTS uplink: transmitter, channel and receiver. Blue, green, yellow, orange blocks are *digital HW*, *SW*, *analog HW* and *interface* blocks, respectively

3. AN APPLICATION EXAMPLE: UMTS UPLINK

We have tested the proposed approach by developping a complete 8-channels UMTS uplink to be implemented on a Sundance SDR kit (SMT8036E) [6]. This is a commercial HW/SW fast prototyping board tailored to SDR systems, which is made of a Texas Instruments TMS320C64 DSP running at 1GHz, one Xilinx Virtex-II, two 140Msample/s DAC's and two 100Msamples/s ADC's, plus a PCI-compatible host board (see fig. 1).

Figure 3 shows the whole system implemented in CodeSimulink/SMT6040, made of six macro blocks:

1. the *signal acquisition* block runs on a host PC and is aimed at sampling input microphones and other sources of data (in our implementation, one stereo microphone is transmitted on two channels, while the other channels transmit synthetic data). It is therefore implemented as a CodeSimulink/SMT6040 *SW* subsystem;
2. the *packet framing* blocks runs on the DSP of the SDR-kit board. Its task is to assemble input samples into packets by properly adding error detection and framing headers. It is therefore implemented as another CodeSimulink/SMT6040 *SW* subsystem;
3. the *spreading, scrambling and filtering* block is the most computationally intensive subsystems, therefore it is advantageously implemented into the FPGA of the SDR-kit board. It is therefore implemented as a CodeSimulink/SMT5040 *digital HW* subsystem. Fig. 4 shows further the interior of this block, which is implemented using CodeSimulink/SMT6040 primitives and other hyerarchical blocks;
4. the *DAC and RF* block is the interface to the analog world, as it contains two 16-bits DAC's and the RF modulator; the former is part of the SDR-kit, while the other is an esternal 2.4GHz IQ modulator (demo board from Analog Devices). The block is implemented as a CodeSimulink *analog HW* subsystem;
5. the *RF channel* is a numerical model of the radio channel. Several models can be chosen (e.g. fading, Gaussian

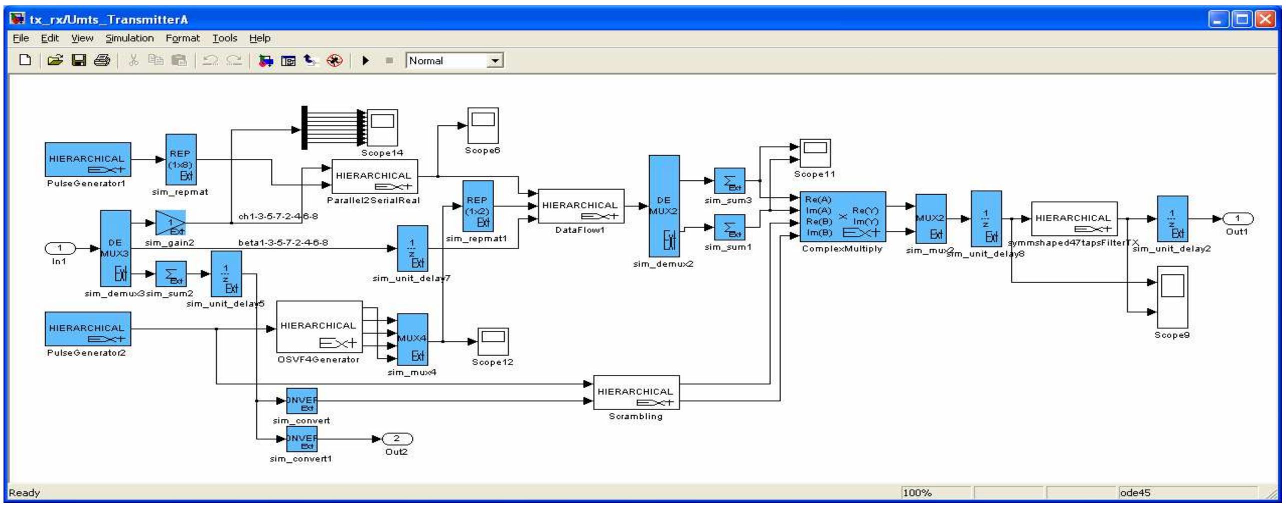


Figure 4 An application example: the digital HW subsystem of anUMTS uplink transmitter. Blue blocks are digital HW primitive blocks, while white blocks are hyerarchical blocks.

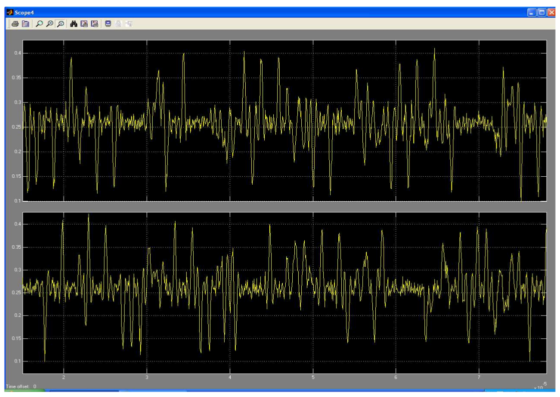


Figure 5 Simulation results: the UMTS signal after the RF demodulator with a Gaussian-noise channel

noise, multi-path, etc.), to assess by simulation the effects of different channel non-idealities. It is implemented as a *pure Simulink* model;

6. the *UMTS receiver* is there only to demodulate and decode received signals and measure bit error rate as a function of different system parameter (see further) . It is implemented as a hybrid CodeSimulink/SMT6040 subsystem (partly in *analog HW*, partly in *digital HW* and partly in *SW*).

Once described the whole system as outlined above (mind that each block is fully described, as it internally goes down to the level of Simulink/SMT6040 primitives), extensive simulations have been performed. Fig. 5 shown the I/Q channel after the RF demodulator; the classical UMTS waveforms, superposed to noise can easily be detected.

A bit comparator allows to measure bit error rate over long sequences, versus noise and implementation parameters

Simulations with different architectural parameters served to optimize the whole architecture and its parameters, among which: data width and oversample rate of output and input shaping filter; resolution of DAC's and ADC's; datawidth of receiver integrator; parallelism (that is, resource sharing) of digital HWsubsystem.

The extensive use of vectors (e.g. the eight input and output channels are an 8-elements vector; the I/Q channel are a 2-D vector; all generators of spreading and scrambling sequences are vectorial generators; the HW/SW interface transfer 8 data values per sample as a vector). This allowed verifying the cost/performance tradeoff for different parallel/sequential architectures, without having to redesign the whole system.

As soon as the system has been optimized, pressing on an appropriate button on the *compiler* block (see fig. 3) has automatically generated the executable for the PC, the DSP and the fusemap for the FPGA (obviously analog blocks are not compiled), which have been downloaded onto the SDR-kit board by means of the appropriate download tools.

The following table compares the time spent for each step of the design with traditional techniques (C+VHDL languages).

Development step	Devel. time with CodeSimulink	Devel. time with traditional tools
Design	2 days	>10 days
Simulation/debugging	1 day	3-5 days
Optimization	1 day	2-6 days
Code generation	20 min	-
Compilation/synthesis	1 hour	1 hour

REFERENCES

[1] Sundance web site, *www.sundance.com*

[2] -, "Simulink manual", http://www.themathworks.com

[3] -, "System Generator manual", http://www.xilinx.com

[4] -, "Codesimulink/SMT6040 user manual", http://polimage.polito.it/groups/codesimulink.html

[5] -, "SMT6040 product description", http://www.sundance.com/edge/files/productpage.asp?STRFilter=SMT6040

[6] -, "SMT8036 product description", http://www.sundance.com/edge/files/productpage.asp?STRFilter=SMT8036E