Efficient Software Development Platforms for Multimedia Applications at Different Abstraction Levels

Katalin Popovici¹
Xavier Guerin¹
Frederic Rousseau¹
Pier Stanislao Paolucci²
Ahmed Jerraya³

¹TIMA Laboratory
46 Avenue Felix Viallet
F38031, Grenoble, FRANCE
{FirstName.LastName@imag.fr}

²ATMEL Roma Advanced DSP
Via V.G.Galati, 91
00155, Roma, ITALY
{pier.paolucci@atmelroma.it}

³CEA-LETI, MINATEC
17 rue des Martyrs, 38054
Grenoble Cedex 9, FRANCE
{ahmed.jerraya@cea.fr}

Abstract

Multimedia applications require heterogeneous multi-processor architectures with specific I/O components in order to achieve computation and communication performances. The different processors run different software stacks made of the application code and the hardware dependent software layer. Developing this software usually makes use of a high level programming environment that does not handle specific architecture capabilities. We propose abstract software development platforms allowing to debug incrementally the different software layers and able to accurately estimate the use of the resources of the architecture. The software development platform is an abstract model of the architecture allowing to execute the software with detailed hardware-software interaction, performance measurement and software debug. Different software development platforms are generated automatically from an initial Simulink model and are used to debug the different software components and to easily experiment with several mappings of the application onto the platform resources. In this paper we apply the proposed approach on a multimedia platform, involving a high performance DSP and a RISC processor, to validate the executable code for a MJPEG decoder application.

1 Introduction

Current multimedia applications require heterogeneous multiprocessor system on chip (MPSoC) architectures in order to achieve computation and communication performances [1]. Heterogeneous MPSoC includes different kinds of processing units (DSP, microcontroller, etc) and different communication schemes (fast links, non standard memory organization and access).

Typical heterogeneous platforms used in industry are TI OMAP [2], ST Nomadik [3] and Philips Nexperia [4]. They incorporate a DSP processor and a microcontroller, communicating via efficient, but sophisticated infrastructure.

To validate the multiple software stacks running on the heterogeneous architectures, classic software development platforms do not fit: (i) high level programming environment does not handle efficiently specific architecture capabilities, while (ii) hardware prototype is too detailed and time consuming for software validation. In practice, the validation of the software code running on these architectures is done by debugging separate low level codes for the different processors, without any global validation of the overall application with the hardware platform. The validation can be performed only when all the binary software is produced and can be executed on the hardware platform.

We use a structured model to represent this kind of heterogeneous MPSoC (figure 1) [16]. Each processor subsystem executes a specific software stack organized in 2 layers, the application and the hardware dependent software (HdS) layer. The HdS is made of lower software layers that may incorporate an operating system (OS), specific I/O communication and hardware abstraction layer (HAL) to allow the execution of the high level code on the hardware platform. This structured model allows validation of the different software components separately.

Figure 1. MPSoC Hardware/Software Architecture

Next generation MPSoC seems to integrate multiple DSP and microcontrollers on a single chip [5]. Hence, the key challenge is how to provide an efficient and fast software development platform for such architectures, that
allows debugging and validating the lower software layers required to map the high level application code on the architecture. The validation and debugging of HdS is the main bottleneck in MPSoC design [8] because each processor subsystem requires specific HdS implementation to be efficient.

As illustrated in figure 2, the software development platform is an abstract model of the architecture in the form of a run-time library or simulator aimed to execute the software. The combination of this platform with the software code produces an executable model that emulates the execution of the final system including hardware and software architecture. This executable model allows simulation of the software with detailed hardware-software interaction, performance measurement and software debug. The platform and the software may be combined using different schemes as it will be explained later in this paper.

Efficient software development platforms allow accurate measurement of the characteristics of the architecture. For instance, a data exchange between different processors may use different schemes (global memory accessible by both processing units, local memory of one of the processors, dedicated hardware fifo components, etc). Additionally, different synchronization schemes (polling, interrupts) may be used to coordinate this exchange. Each of these communication schemes has advantages and disadvantages in terms of performance (latency, throughput), resource sharing (multitasking, parallel I/O) and communication overhead (memory size, execution time). The ideal scheme would be able to produce an efficient software development platform, fast enough to allow exploration of different mapping schemes in order to evaluate the effect on performances of using the different architecture capabilities, and accurate enough to debug the different software components.

![Figure 2. Software development platform](image)

The key contribution of this paper is a software development platform, able to efficiently use the resources of the architecture and allowing easy experimentation of several mappings of the communication onto the platform resources. We use Simulink environment to capture both application and architecture initial representations. This model makes explicit the software distribution on different processors and the used communication schemes. Each software stack is structured into layers to allow easy software debugging, but also flexibility in terms of software components reuse (OS, communication protocol) and portability to other platforms. Specific software development platforms are generated automatically from the initial Simulink model to validate incrementally the different components of the software stacks. These different simulation models are used also to allow performance estimation.

The rest of the paper is organized as follows. Section II relates previous works. Section III presents the software development platform generation flow. Section IV deals with a case study of a complex architecture namely Diopsis tile, which executes an MJPEG decoder application. Section V gives some results and analysis, and finally some conclusions.

## 2 Related work

Previous work related to software development platforms can be classified in 3 categories: software oriented, hardware oriented or electronic system level (ESL) oriented.

The software oriented approaches make use of a software model in the form of a run-time library to model the interaction with hardware [9][10]. The application is written in a high level language. The software code validation consists in compiling and linking it with the libraries and natively executing or simulating it using a high level environment. The main disadvantage is the lack of accuracy. Moreover, the library is defined separately for each processor and can be very sophisticated. Thus, it implies high complexity for software debug. Such approaches have been already applied for SoC architectures (e.g. YAPI [11]) such as Trimedia platform with DSP and MIPS.

The hardware oriented approach executes the final software on a virtual platform and it corresponds to classic hardware-software cosimulation models using Instruction Set Simulators (ISS) [12] [13]. These techniques require that all the software and hardware are fully specified and designed down to RTL level if cycle accuracy is required. Thus, the software validation occurs too late and the debugging may be too expensive and fastidious.

The ESL oriented approaches use high level APIs (Application Programming Interface) to abstract the hardware-software interfaces, e.g. DSOC [14], TTL [15]. This approach enables the automatic generation of a virtual prototype from a system level model, but the generation of HdS software layer is performed in one step, which generally implies the use of predefined communication schemes. Moreover, the gap between the system level model and generated code makes the debugging of the generated software stacks more difficult and the identification of the different sources of bugs very difficult. Thus, this approach lacks the structuring of the hardware-software interfaces to
allow the debugging of the different parts of these interfaces separately.

The approach presented in this paper combines all the three previously described: it makes use of a high level simulation environment in Simulink, it allows the execution of the software code on a virtual platform and it abstracts the hardware-software interfaces by using high level APIs. The main difference of our approach related to the previous approaches consists in structuring the software stack in well defined layers and allowing the incremental validation of these different software components by using different software development platforms. This makes easier the debug of both application and HDS in addition to opening new opportunities like early performance estimation and communication mapping exploration.

3 Software development platforms generation flow

In order to allow separate validation of the different software components, we use different abstraction layers of the software. Each abstraction layer corresponds to a specific component of the software stack. Figure 3 illustrates these layers for a simplified application made of 3 tasks (T1, T2 and T3), that need to be mapped on an architecture made of 2 processors and hardware subsystems. For each level, figure 3 shows the software organization, the hardware-software interface and the software development platform that will be used to validate the software at the corresponding abstraction level. The key differentiation between these levels is the way of specifying the hardware-software interfaces and the communication.

The highest level is the system architecture level (SA). In this case, the software is made of a set of functions grouped into tasks. Several tasks may be mapped on the same software subsystem. The communication between functions, tasks and subsystems make use of abstract communication links, e.g. standard Simulink links or explicit communication units that correspond to specific communication paths of the target platform. The simulation at this level is made using standard Simulink environment to validate the application’s functionality.

The next level is called virtual architecture level (VA). The software is refined into tasks C code that contains the final application code and makes use of HDS API. The communication primitives of the HDS API access explicit communication components. Each data transfer specifies an end-to-end communication path. For example, the functional primitives send_reg(ch,src,size)/recv_reg(ch,dst,size) may be used to transfer data between the two processors using a register connected to the system bus. The software is executed using an abstract model of the platform that provides an emulation of the HDS API. The simulation at this level allows validation of the final code of tasks and may give useful statistics about communication requirements.

The next level is called the transaction accurate architecture level (TA). The software is linked with an explicit OS and specific I/O software to implement communication units. The resulting software makes use of hardware abstraction layer primitives (HAL_API). Data transfers use explicit addresses, e.g. read_reg(addr, dst, size)/write_reg(addr, src, size). The software is executed using a more detailed platform to emulate the interconnect, the explicit peripherals used by the HAL API and an abstract computation model of the processor. The simulation at this level allows to validate the integration of the application with the OS and the communication layer. It may also provide precise information about the communication performances.

Finally, the HAL API and processor are implemented through the use of a HAL software layer and the corresponding processor part for each software subsystem. This represents the virtual prototype level (VP). At the virtual prototype level the communication consists of physical I/Os, e.g. load/store. The simulation at this level allows performance validation and it corresponds to classical hardware/software cosimulation models [12] [13].

![Figure 3. Software abstraction levels](image-url)

3.1 Global Flow

As shown in figure 4, the software design flow starts with a hierarchical Simulink model to capture the grouping of functions into tasks and tasks into subsystems. The Simulink model may use explicit communication units to abstract the intra-subsystem and inter-subsystem communication. The flow starts also with an abstract model of the architecture to generate the execution model at each refinement step. Both initial application and abstract architecture are inputs of the system. The proposed flow validates incrementally the application tasks, the HDS and the
final binary code. At each step, the flow also generates a software development platform that allows debugging of the corresponding software component.

Figure 4. Simulink software design flow

The software development platforms depend on the application partitioning and mapping, as they capture the communication between the different subsystems and the different tasks of the same subsystem. The automatic generation of these platforms maps the communication buffers onto the memories, as required by the communication protocol.

3.2 Virtual Architecture Platform Generator

At the Virtual Architecture level, the platform is described using SystemC language. The software is represented by the task code as C program code (figure 3-b). This C code is very similar to that generated by Real Time Workshop [7], except that it is encapsulated into a SC_THREAD. To implement external communication, the inter-subsystem communication units become explicit resources visible by the HdS API used inside the task code. The intra-subsystem communication units become SystemC communication channels.

The Virtual Architecture Platform Generator generates a SA Execution for each task and a SC_MODULE for each processor that includes the tasks. It also makes explicit the interconnect between the different subsystems, connects all the hardware components and maps the communication buffers on the memory resources. During the generation, it makes use of a storage and interconnect components library. Compiled together, the task code and platform produce an executable model that makes use of the SystemC scheduler to execute the different tasks.

3.3 Transaction Accurate Architecture Platform Generator

At Transaction Accurate Architecture level, the task code is integrated with an OS and communication component that implements the HdS API and makes use of a HAL API (figure 3-c). The software tasks are scheduled by the OS scheduler and executed natively.

The Transaction Accurate Architecture Platform Generator builds a SystemC model of the abstract architecture. This contains all the resources that are visible by the HAL API. The communication protocol becomes explicit. The intra-subsystem communication units become software channels fully managed by the OS. The Transaction Accurate Architecture Platform Generator details the local architectures of the processors, including peripherals, synchronization components, memories or network interfaces. Then, the generator maps the different communication buffers on the different memory resources and creates the interrupt lines used for the communication protocol. It also connects all the hardware components. During the generation, it makes use of a library composed of the memories, peripherals, interconnect and abstract processor components.

4 Experiments

In this section, we present the results of mapping a Motion JPEG decoder application on a multimedia platform by using software development platforms at different abstraction levels. In this paper, we use as an architecture example the Shapes MPSoC architecture [6], which is a multi-tile architecture based on a Diopsis tile. The high level application model is made of a set of functions. The partitioning and mapping of the application on the target architecture is captured in a Simulink representation. Then, the different software development platforms are generated from this initial Simulink model, in order to validate gradually the different software components. During the experimentation, we also try 3 different communication schemes for performance measurement, as it will be detailed in the next section.

We use different execution models adapted to each software abstraction level, as shown in figure 4. The initial simulation at system architecture level is made in Simulink, while all others are made in SystemC. The execution models at different abstraction levels in the case of the Diopsis MPSoC architecture will be presented in the next sections. For the validation of the software components via simulation at different abstraction levels, we used a ten frames input bit stream encoded using the QVGA YUV 444 format.
4.1 Subset of Diopsis tile components

We consider a simplified Diopsis tile. The selection of components from the original architecture still captures all the possible communication schemes and specific I/O components.

The subset is shown in figure 5. It includes ARM and DSP subsystems. The ARM subsystem includes the processor core and local memories (SRAM, ROM); while the DSP subsystem includes the DSP core, data memory (DMEM), program memory (PMEM), registers (REG), DMA, interrupt controller and synchronization component. The hardware nodes consist of external distributed memory subsystem (DXM) and peripherals on tile (POT) subsystem. The POT includes system peripherals of the RISC processor, e.g. timer, interrupts controller (AIC), but also I/O components like the serial peripheral interface (SPI). The interconnection between these subsystems is made via AMBA bus.

For performance reasons, the ARM processor can access directly the data memory and control/status registers of the DSP processor via the AMBA slave interface of the DSP subsystem. In the same way, the DSP core can read/write directly on the local memory of the RISC processor by initiating a DMA transfer. Moreover, the processors can store and load data to/from DXM connected to the AMBA bus.

4.2 System architecture model

The system architecture model captures the functions of the application and the mapping information of the application on the target architecture. We built the MJPEG decoder Simulink model. During this step, the main functions of the MJPEG decoder were isolated into separate tasks (figure 6).

The VLD constitutes the first task. The zigzag scan, DPCM, RLD, and IQ are grouped into a second task. We mapped these first 2 tasks on the ARM processor. The IDCT computation was mapped on the DSP. The resulting decoded image is displayed on a LCD panel connected through the SPI peripheral of the POT (task 4). Thus, the Simulink hierarchy captures the mapping of the application on the architecture at a high abstraction level through the decomposition of the system into 4 tasks and 3 subsystems (ARM, DSP and POT). The system architecture model is illustrated in figure 7.

The Simulink model includes also 8 communication units to capture different communication protocols and resources provided by the architecture (comm1, comm2… comm8). In this case, we use software fifo (SWFIFO) as intra-subsystem communication protocol; for inter-subsystem communication we use DSP data memory (DMEM), DSP registers (REG), ARM local memory (SRAM) and distributed external memory (DXM). At this level, Simulink will use an abstract simulation model for each of these communication units. Later in design flow, each of these units can be mapped on a specific communication path and implements the communication protocol of the final architecture. The number of communication units depends on the application partitioning and mapping.
mechanism, or *NetworkType* which specifies the type of network used to interconnect the different subsystems, etc.

During experimentation, we attempted 3 communication schemes between DSP and RISC. In the first scheme, the data exchange is made only via DXM. The second communication scheme makes use of DXM and REG communication units between the processors and DMEM between DSP and POT. The third case uses the SRAM as communication unit between processors and DMEM between DSP and POT. In all the cases, the communication between tasks mapped on the same subsystem makes use of swfifo protocol.

We performed a simulation at this level to validate the functionality of the application. The simulation time of the 10 frames decoding process in Simulink was 35s.

### 4.3 Virtual architecture platform generation and simulation

From the Simulink system architecture model, we generated automatically the software development platform to validate the tasks C code at the virtual architecture level. The tasks code makes use of HdS API. Thus, the generated platform includes all the components accessible by HdS API and the resources to implement the required end-to-end communication paths. The inter-subsystem communication units are partially mapped on the memory modules (DXM, REG, SRAM, DMEM), attached as slave components to the AMBA bus. The address space of the components are automatically assigned and computed by using a template that contains the predefined address size for each component. The virtual architecture platform generation takes 3 seconds and generated 329 lines of code in SystemC.

At this level, the simulation model makes use of SystemC environment (figure 8). Each software tasks is a SystemC thread, the processor and memories are SystemC modules. The AMBA and software fifo’s are SystemC channels.

Besides task code validation, the simulation model allowed also gathering of important early performance measurements, e.g. using the second communication scheme with DXM, REG and DMEM, the bus was accessed to transfer 144,000 messages during the decoding process of the ten frames. The data transfer between the processors through DXM is performed in messages of 64 words for the IDCT coefficients and in 1 word for the coding pattern via REG; the data transfer between the DSP and POT via DMEM is performed in messages of 16 words. This simulation model was accurate enough to validate the functionality of the task code. The simulation time required to decode the ten frames was 14s on a PC running Linux OS at 1.73GHz.

### 4.4 Transaction accurate architecture platform generation and simulation

Then, we generated the Transaction Accurate Architecture platform, in order to validate the integration of the tasks code with an in house tiny OS and communication library for each processor. At the transaction accurate level, the platform contains the components that can be accessed by HAL API (figure 9). This includes the interrupt controllers of both processors, the DMA accessible by the DSP, synchronization components such as mailboxes, bus interfaces and other peripherals such as a timer or SPI.

![Figure 8. Virtual architecture simulation model](image)

The communication buffers between the different subsystems are mapped on the corresponding memory modules based on the protocol specified at Simulink level.

![Figure 9. Transaction accurate simulation model](image)

The assignment of addresses and mapping of the communication buffers into the memories with the corresponding interrupt mechanism used for synchronization is performed during the platform generation. The total number of generated lines of SystemC code is 443.

The simulation model at the transaction accurate architecture level enables native execution. Each software stack is a Unix process which interacts with the abstract processor via the Unix IPC layer. The HAL APIs implementation is also based on the Unix semantics. The hardware components, including the abstract processors, are described in SystemC. The communication between the tasks mapped...
on the same processor is implemented by the OS and it corresponds to the protocol information of the communication unit specified at the Simulink level. The address space of components is different from the Virtual Architecture platform, because the generated platform at Transaction Accurate level is more detailed and fully implements the communication protocol.

Besides the software debug, the execution model at this level also provided more precise idea on performances, that allowed some architecture experimentation. The simulation at transaction accurate level takes 5m10s. Using the simulation, we estimated the overall number of cycles required for the communication having the AMBA clock frequency as unit of measure. The performance estimation results, as well as the number of transactions to the memories required for communication in the 3 cases of different communication schemes are represented in table 1. The results prove that we can obtain a gain of 55% in communication performances by using the architecture capabilities. The communication protocol is specified in the initial Simulink model by annotating the communication units. Therefore, it is quite simple to experiment with different communication schemes, as the generation of the platforms in all the cases takes about 5 seconds.

<table>
<thead>
<tr>
<th>Communication Scheme</th>
<th>Transactions</th>
<th>Total cycles</th>
<th>—</th>
</tr>
</thead>
<tbody>
<tr>
<td>DXM</td>
<td>5256k</td>
<td>8856k 100%</td>
<td></td>
</tr>
<tr>
<td>DXM+REG+DMEM</td>
<td>4608k 72k</td>
<td>7884k 89%</td>
<td></td>
</tr>
<tr>
<td>SRAM+DMEM</td>
<td>0 4680k 576k</td>
<td>3960k 45%</td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Memory accesses

5 Results and analysis

The proposed software development platforms proved to be quite powerful. The different platforms were generated only in few seconds and allowed debugging of the different software components. The platforms also make efficient use of the architecture resources. Several mappings of the communication onto the platform resources can be experimented with using simple annotations of the Simulink model. We found that the key benefit of this approach is to generate simulation models that allow software debugging of different applications and performance estimation.

6 Conclusion

In this paper, we presented software development platforms at different abstraction levels that target a complex multimedia platform, namely a reduced Diopsis tile. These platforms are generated automatically from an initial Simulink model by annotating the communication units. Therefore, it is quite simple to experiment with different communication schemes, as the generation of the platforms in all the cases takes about 5 seconds.

References

[1] H. Meyr “Application Specific Processors (ASIP): On design and implementation Efficiency”, Proceeding of SASIMI 06, Nagoya, Japan