MODELING FRAMEWORK FOR HIGH-LEVEL EVALUATION OF EMBEDDED SYSTEMS

Klemen Perko[1], Andrej Trost [2]

[1] Sipronika d.o.o.
1000 Ljubljana, Tržaška 2, Slovenia
[2] University of Ljubljana, Faculty of Electrical Engineering,
1000 Ljubljana, Tržaška 25, Slovenia

klemen.perko@sipronika.si (Klemen Perko)

Abstract

As technology advances, options for realization of heterogeneous systems increase. Traditional approach to embedded systems design does not offer satisfactory support for building efficient contemporary designs. Nowadays designers use a variety of hardware (HW) and software (SW) co-design methodologies in order to meet application constraints as fast as possible. The paper presents a graphical modeling framework used for high-level modeling, evaluation and design-space exploration of heterogeneous systems. The framework provides designer graphical elements for using modeling concepts from system modeling libraries. Graphical modeling relieves the designer of the manual-typing source code and thus hides many details of system-level design languages that normally need to be taken care of. The graphical framework also provides different constraint checks during modeling and automatically generates an executable model for evaluation of a heterogeneous system.
The applicability of the modeling framework is illustrated within a case study where a system-level modeling of a simplified digital camera is presented. Case study exemplifies the use of the framework and shows what information is obtained from an executable model built on a high-level of abstraction. Evaluation of results serves as a basis for further design decisions. Graphical modeling enables rapid changes in the model and thus speeds-up design-space exploration.

Keywords: Embedded systems, High-level design, Graphical modeling, System-level simulation, Design-space exploration.

Presenting Author's biography

Klemen Perko received his B.Sc. degree in electrical engineering from the Faculty of Electrical Engineering, University of Ljubljana, in 2004. Since then, he has been working as HW and System Design Engineer. In October 2004, he started working towards the Ph.D. degree at the Faculty of Electrical Engineering, supported by Ministry of Higher Education, Science and Technology. His current research interests include HW/SW codesign, high-level modeling and design of embedded systems and systems on a single chip.

ISBN 978-3-901608-32-2

1 Introduction

System modeling and model evaluation is an important step in the embedded system design process. Embedded systems design flow comprises many steps from initial system specifications toward actual implementation. Designers are encouraged to use a variety of HW and SW implementation technologies in order to meet application constraints and provide quick time-to-market solutions.

Traditional approach of embedded systems design is based on selection of an architectural platform and development of software algorithms for the specified functionality. First assumption is that execution of algorithm is performed on microprocessors. After programming is finished obtained system performance is evaluated. If real-time constraints are not met, profiling of algorithm is performed to identify which parts of algorithm present bottlenecks. Identified parts are then considered for parallel implementation in additional circuits (FPGA or ASIC) [1].

For many different reasons traditional design approach gives suboptimal implementation results. Some ad-hoc decisions about architecture are made at the earliest stage of design process. Algorithm profiling also gives suboptimal results, since implementation of algorithm can substantially differ if it is implemented in sequential or parallel architectural resources. One of weaknesses of traditional approach is that first results of performance evaluation can not be obtained until actual programming is finished. Consequently process of design space exploration is very time consuming [2,3].

Contemporary methodologies for designing embedded systems offer conceptual shift away from solving the problem in a traditionally sequential manner and concentrate on system-level modeling. Modeling at system-level assures that heterogeneous information of HW architectural resources and SW functionality is collected in one common model. Many different HW/SW codesign methodologies have been presented [4]. HW/SW codesign process enables adequate model evaluation at early stages of design and though helps avoiding premature and ad-hoc decisions as they unjustifiably narrow the available design space and eliminate potentially better design solutions.

Design flow of system-level modeling is presented in Fig. 1. This is iterative process. Considering specifications initial system-level model is built. Models on different abstraction levels are used in order to manage design complexity [5]. Initial system model is described on high abstraction level and evaluated. If evaluation results do not meet constraints, model is revised in the process of design space exploration. When satisfactory model at selected level ob abstraction is obtained, additional information is added and though level of abstraction is lowered. This is repeated until implementable model is obtained.

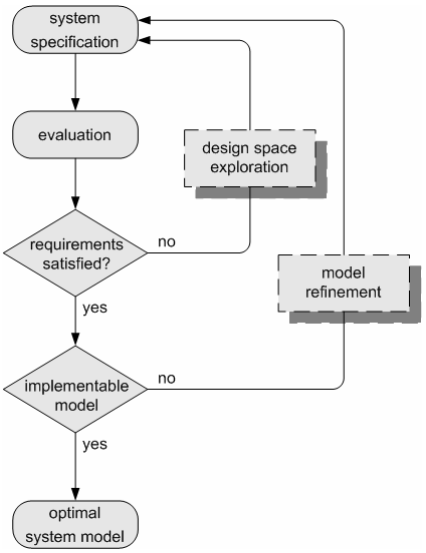

Fig. 1 System-level design flow

In this paper we will focus on graphical modeling of embedded system on the highest abstraction level. We present graphical modeling framework used for high-level modeling of heterogeneous systems. The framework was designed in order to relieve designers of the burden of repeatedly implementing models of some basic concepts. Graphical environment enables rapid changes in the model and thus speeds-up design-space exploration.

For modeling embedded systems at the earliest stage of design, we identified basic elements of these systems that are repeatedly needed by designers. UML static class diagram notation in Fig. 2 illustrates basic elements of embedded system model on high abstraction level.

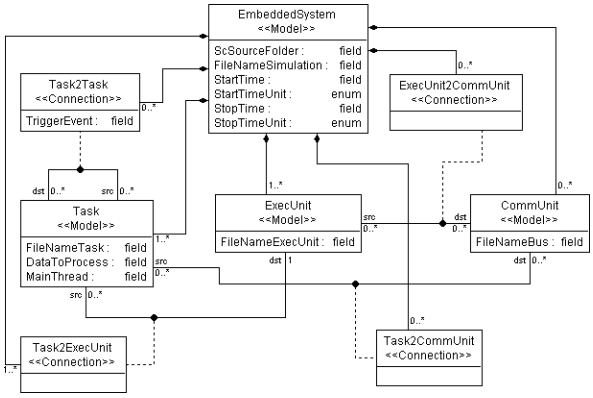

Fig. 2 Basic elements of embedded system model on high level of abstraction

Embedded systems are composed from HW architecture resources and SW functionality. Functionality can be represented as a composition of several *tasks*. Order of *task* execution is defined with connections between them (*Task2Task*). On the other hand architecture on high abstraction level can be represented as a composition of several execution units (*ExecUnit*), communication units (*CommUnit*) and connections between them

(*ExecUnit2CommUnit*). SW functionality is executed on HW architecture and utilizes its resources. With mapping process, merging of functionality and architecture heterogeneity is performed. To define what architectural resources could specific task utilize *Task2ExecUnit* and *Task2CommUnit* connections are needed.

2 Graphical modeling framework

Since process of developing graphical framework from the ground-up is very complex, time-consuming and expensive we decided to use one of already developed generic graphical frameworks that can be configured for our specific needs. Open source graphical modeling environments found suitable for us are Eclipse Graphical Editing Framework [6] and Generic Modeling Environment (GME) [7]. We decided to use GME since it is more mature, offers very good user support through online forum and provides tools for easy integration of the interpreter for translating the graphical model.

2.1 Generic Modeling Environment - GME

The Generic Modeling Environment (GME) [7] is a configurable toolkit used for creating domain-specific modeling, model analysis, model transformation and program synthesis environments. The configuration is accomplished through meta-models specifying the modeling paradigm (modeling language) of the application domain. The modeling paradigm contains all the syntactic, semantic and presentation information regarding the application domain. It defines concepts used to construct models, their relationship, organization and graphical presentation, and rules governing model construction.

The modeling paradigm is created by configuring a meta-model using the GME meta-modeling language. Meta-models are used to automatically generate target domain-specific environment. An interesting aspect of this approach is that the environment itself is used to build meta-models. This top-level environment is called a Meta-metamodel.

The meta-modeling paradigm is based on the Unified Modeling Language (UML) [8]. The syntactic definitions are modeled using pure UML class diagrams and the static semantics are specified with constraints using the Object Constraint Language (OCL). This process needs to be done just once. Users of this domain-specific framework can build their specific models according to rules defined in the meta-model.

Fig. 3 illustrates a snippet of the UML meta-modeling paradigm and its actual corresponding presentation in GME. The curvy arrows show how individual modeling elements and their relations are defined by different parts of the meta-model.

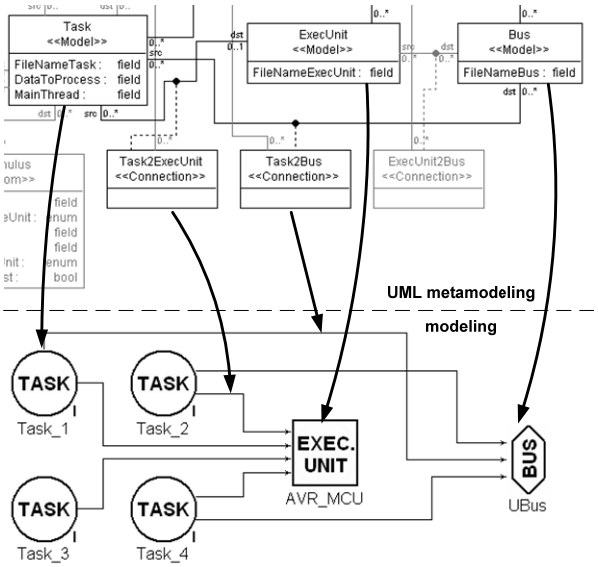

Fig. 3 Creating a domain-specific modeling framework

GME has a built-in set of generic concepts: folders, models, atoms, connections, roles, constraints and aspects. These concepts are the main elements used by the meta-model developer. We will not make a detailed presentation of all of them as this would exceed the scope of this paper. The reader can find it in [9,10]. We will just point out the concept of aspects. Aspects provide visibility control. They are used to allow models to be constructed and observed from different viewpoints. Existence of parts of the domain in a particular aspect is determined by the meta-model. Each part can be either visible or hidden. The concept of aspects allows the user to employ just the parts suited for a selected viewpoint and hide all the others irrelevant for it.

The generated domain-specific environment is then used to build domain models that are stored in the model database. GME also provides high-level C++ and Java interfaces for writing plug-in components to traverse, manipulate and interpret graphical models into an appropriate text description suiting as input to Commercial Off-The-Shelf (COTS) analysis tools. The interpreter needs to be written by the meta-user because interpreter must be able to translate graphical models built according to the meta-model.

2.2 Building paradigm

To configure GME for specific needs, we built a meta-model containing information of the basic elements for modeling embedded systems on high abstraction level.

As mentioned above, embedded systems at high abstraction level can be modeled with basic elements presented in Fig. 2. For clarity of presentation the only most important elements of our system-level modeling methodology are presented. The meta-model enables a model of a typical embedded system to be made-up as a composition of:

ISBN 978-3-901608-32-2

- at least one execution unit (*ExecUnit*),
- any number of communication units (*CommUnit*), and
- at least one task (*Task*).

The restrictions for the numbers of instances in the actual model are set by multiplicity constraints (e.g. constraint for the *ExecUnit* is set to: "1..*"). The meta-model also defines possible connections between these elements. The designer can make just the connections permitted in the meta-model. The connections shown in Fig. 2 are:

ExecUnit2CommUnit: with these connections the designer defines the communication units available for a selected execution unit. Generally an execution unit can have more than one communication unit and many different execution units can share the same communication units. Instances of execution and communication units connected together compose system architectural resources.

Task2Task: with these connections the designer defines the order of task execution. The order is governed by the tasks' data dependency and the direction from the source to destination has to be followed. Instances of the tasks connected together with the *Task2Task* connections compose system functional description.

Task2ExecUnit: with these connections the designer assigns execution units responsible for execution of a selected task. Each task can be assigned to only one execution unit.

Task2CommUnit: with these connections the designer defines the communication units available for data-manipulation operations of tasks. Generally, a task can use more than one communication unit, but only those available to the assigned execution unit can be used. This means that the designer can select only between those communication units that have been previously attached with *ExecUnit2CommUnit* to the execution unit.

Besides the presented blocks, the meta-model contains also some other elements required for model construction and simulation setup. All of them are listed in Table 1. The event splitter and event joiner are used for defining the order of task execution. The event joiner performs an addition of multiple input events when starting a specific task depends on execution ending of multiple tasks. Event splitter triggers multiple tasks in a certain order and can be used for modeling a SW scheduler. Start and stop events are used for control of the simulation process. External event-generator elements serve for imitating input signals coming from the surroundings where our system will be operating. Waveform trace and console log elements serve for setting which data will be collected from the model during the simulation.

The concept of aspects in GME provides visibility control. The aspects allow models to be constructed and viewed from different viewpoints. They show only elements relevant in a particular aspect. In our meta-model we implemented four different aspects in which a model of an embedded system can be viewed.

In the task triggering aspect, the designer enters functionality of the system by placing and connecting task instances. The simulation setup elements (start and stop events) and external-event generators are also defined in this aspect.

In the architecture aspect, instances of hardware resources (execution and communication units) are placed and connections *ExecUnit2CommUnit* are defined.

The mapping aspect serves for mapping tasks to appropriate hardware resources. Only connections among the already defined instances can be made.

In the simulation setting aspect, *WaveformTrace* and *ConsoleLog* set elements are instantiated and their appropriate members defined.

Table 1 lists all of the implemented elements of our meta-model in conjunction with the visibility aspects. Even if a specific element is visible in more than one aspect, it can be instantiated or modified only in its primary aspect. The primary aspect is denoted with a shadowed cell.

Table 1. Visibility of elements depends on the aspect

Visibility \ Aspect	Task Triggering	Architecture	Mapping	Simulation Settings
Task	•		•	
Event Splitter	•			
Event Joiner	•			
Execution Unit		•	•	•
Bus		•	•	•
Start Event	•			
Stop Event	•			
External Event Gen.	•			•
Waveform Trace				•
Console Log (usage)				•

For connecting all the elements together, we defined proper connections in the meta-model. As mentioned above, we will not describe all of them since this is not crucial for understanding the idea of our approach. At this point it needs just to be noted that the possibility of making connections also depends on the aspect.

2.3 Model interpretation

Important part of our graphical modeling framework is the model interpreter. The purpose of the interpreter is to translate all information captured in the graphical model into a system-level textual description.

For modeling on the system-level, different system-level design languages have been developed [4]. These languages enable textual description of system's HW an SW components.

Our modeling framework is based on SystemC. SystemC is implemented as a C++ class library and is standardized by IEEE-1666. The SystemC extends the capabilities of the C++ by enabling modeling of

hardware descriptions [11,12]. It adds important concepts to the C++ such as concurrent processes execution, modeling timed events and hardware data types.

For modeling on a high abstraction level, system level modeling libraries were developed in our Laboratory [13,14]. They provide wrappers for modeling functionality and architecture on a high-level of abstraction. One of the integral parts of system modeling libraries is also a built-in support for logging relevant information about the system during execution of simulation.

Model interpreter translates graphical elements into model instances in SystemC. During interpretation it uses attributes and high level task description in SystemC code provided by system model designer. Detailed overview of high level task description will be explained in next section. Interpreter performs also different syntactic and semantic checks in order to verify the graphical model. Errors are reported and the designer is guided to repair the model. The interpreter generates the SystemC source code together with appropriate project files for automatic compilation and linking. Finally, an executable description of the system model is obtained.

3 Application of modeling framework

To see how our graphical modeling framework operates in practice, system-level modeling of a simplified digital camera will be presented. A digital camera is a complex system comprising both mechanistic and electronic components and is very well-suited as an application case study for our modeling framework. In accordance with the focus of this paper we concentrate on high-level aspects of the design space exploration. We will show that using our graphical framework for modeling the observed system on a high abstraction level enables performance estimation before the implementation is made. The framework enables rapid changes in the model (e.g. changes in the mapping aspect can give better results) and allows very easy exploration of different system implementations.

3.1 Digital camera system

A digital camera captures and stores images in digital format on a storage device. Fig 4 depicts a block diagram of a proposed simplified digital camera system.

Normal digital camera operations commence with the process of determining proper settings for the scene or subject to be photographed. Such tasks typically involve adjusting the focus, setting image quality, measuring and gathering shooting parameters, and selecting appropriate shutter duration and aperture opening. Once the required parameters are set and the shutter button is pressed, the following sequence of operations typically ensues:

- The shutter is closed; the sensor becomes temporarily inactive, and is instantly flushed off all residual charges. This step is to prepare the sensor to capture a new image.
- Depending on the camera and the settings, the residual charges that are flushed off the sensor may be analyzed to acquire the proper settings for automatic point-and-shoot operations.
- The sensor becomes active and, at the same time, the shutter opens, exposing the sensor to light-charging it as a result. The shutter remains open for the specified exposure duration, before closing again. The image can now be captured and streamed off to the Image Conditioning module.
- The shutter re-opens, and the camera is ready to take another picture.

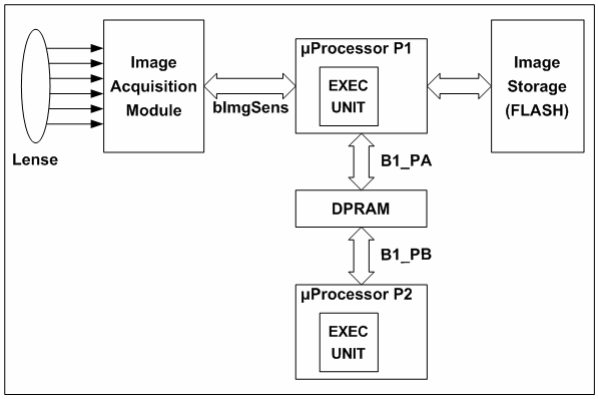

Fig. 4 Block diagram of a proposed digital camera system

3.2 Model construction

Construction of the initial digital camera model on the high abstraction level was performed in four aspects using graphical elements from the meta-model.

3.2.1 Functionality aspect

In the first aspect, functionality of the digital camera is defined. Simplified functionality of the system can be divided into nine different tasks presented in Fig. 5. For simulating the "shutter" button triggering, a start event element *Event_T0* is used.

Tasks *readImgSens* serves for simulation of the first reading of data from image sensor. Depending on the data obtained, task *setCptrParams* sets parameters for appropriate shutter duration, aperture opening and calculates factors for color correction (e.g. white balance). In task *takePicture* processor waits for image sensor to actually capture the image. In task *readImgPlCorr* reading of data from image sensor and color correction with previously calculated factors is performed. The following four tasks (*rgb2yuv-entrCod*) server for compression of obtained image according to JPEG standard [15]. This encoding process consists of four consecutive stages: color-space conversions (*rgb2yuv*), forward discrete cosine transform (FDCT) (*frwrdDCT*), coefficient quantization (*quant*) and entropy coding (*entrCod*). Other pre and post processing stages (e.g. down-

sampling) are omitted here for clarity. In the last task *writeToFlash* writing of obtained compressed image to flash storage media is performed.

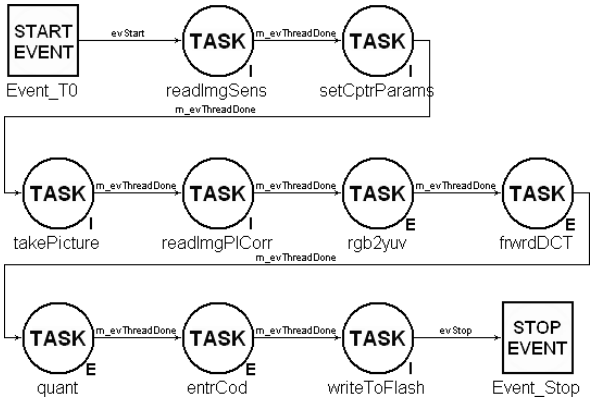

Fig. 5 Functionality description in the task triggering aspect

3.2.2 Architecture aspect

Previously described tasks need appropriate HW architecture units so they can utilize their resource to perform execution of requested operations. In our case-study we examine impact of three different architectures on time needed to perform desired functionality of digital camera.

The first architecture implementation on Fig. 6 contains only one processor *P1*, image sensor is connected to *P1* via memory bus *bImgSens* and memory for image processing during compression is connected to *P1* via memory bus *B1*. The second architecture contains two processors – *P1* and *P2*. Both of them are connected to a shared single port memory via bus *B1*. In the third architecture implementation differs from the second by using dual-port memory for image processing. Since dual-port memory allows independent memory access on both ports it is modeled with two memory buses: *B1_PA* is connected to *P1* and *B1_PB* is connected to *P2*.

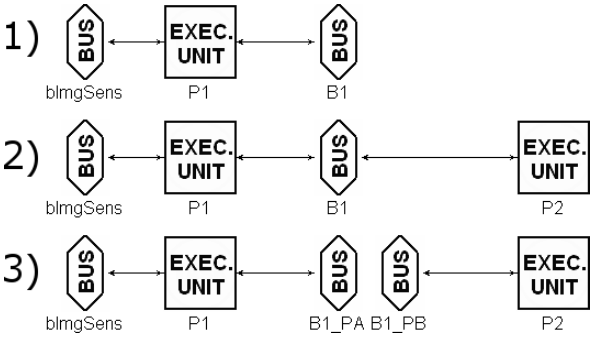

Fig. 6 Three different architecture implementations of digital camera on a high abstraction level

3.2.3 Mapping aspect

Merging of functionality and architecture is performed in mapping aspect. Each task from functionality aspect is connected to specific execution unit (e.g. processor and memory bus). For each architecture implementation, separate mapping needs to be performed. E.g. for first architecture implementation all functionality tasks are mapped on execution unit *P1*, and those that perform any data transfer also need to be mapped to appropriate memory buses (*bImgSens* and/or *B1*). For instance tasks *readImgSens* and *readImgPlCorr* both need to be mapped on both memory buses.

JPEG compression algorithm allows some parallel operations. Since the same algorithm needs to be performed for each of three color planes, there is possible to perform some tasks in parallel (e.g. when *rgb2yuv* the first plane is finished, *frwrdDCT* for this plane and *rgb2yuv* for the second plane can run in parallel). The possible parallel execution of the tasks is limited by their data dependency and available HW resources. In general processors present one execution unit capable of just sequentially executing many different tasks. Since there is only one processor in the first architecture, there is no possibility for any parallel execution of tasks.

The other two architectures with two processors allow such parallel operation. As mentioned before they differ by memory bus configurations.

On Fig. 7 mapping of the tasks on architecture No. 3 is presented. The processor P2 executes only one task: *fwrdDCT*, the other tasks are mapped to P1.

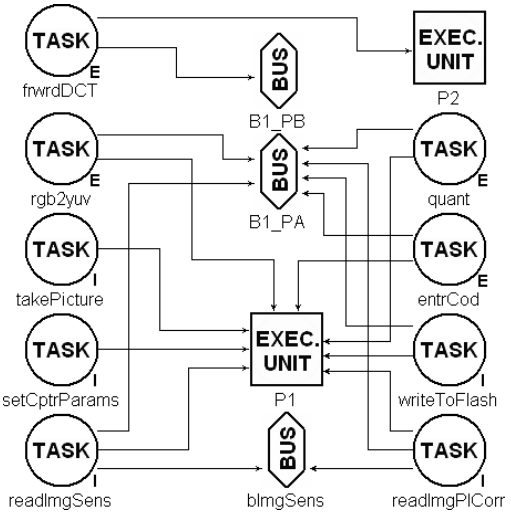

Fig. 7 Mapping of tasks on architecture resources for architecture No. 3

At this point let us briefly explain how the tasks are actually described. The task model is defined in terms of architectural resource usage Fig. 8 lists a high-level algorithm description for the forward DCT encoding stage (task `frwrdDCT`). When studying a specific DCT realization (e.g. [16]), it can be concluded that 11 multiplications and 20 additions are needed to apply the 1-D DCT transform. These operations represent abstract functions. The purpose of the triple nested for loops of Fig. 6 listing, explained from the inside out, is as follows. The 2-D DCT transform (of an 8x8 pixel block) is obtained by applying the 1-D DCT transform to rows first (loop 3) and columns second (loop 2). The pixels to be transformed must be grouped in segments of 8x8 pixels (structured block in data

domain) (loop 1).

The predefined methods Add, Mult, GetData and WriteData from our system modeling libraries are used for modeling architectural resource (i.e. processor) usage. All method requests are performed through execution unit interface (m_pExecUnit). The libraries provide estimations of the execution time for each method on particular execution unit.

```
void jpg_frwrdDCT::MainThread()
{// divide requested size (bytes) into blocks
of 64 bytes
 // do 2 pass 1D DCT
 int blocks = ROUND_UP( m_bytes, 64 );
 if( m_bytes % 64 )
 cout << sc_time_stamp() << " input size to
frwrdDCT is not 8x8 aligned\n";
 for( int i = 0; i < blocks; i++ )
 {// pass 0 - horizontal, pass 1 - vertical
   for( int pass = 0; pass < 2; pass++ )
   {
     for( int line = 0; line < 8; line++ )
     { m_pExecUnit-GetData(this,true,&b1_pb,
       -1);
       // 11MUL, 20ADD
       m_pExecUnit->Mult(this, 11 );
       m_pExecUnit->Add(this, 20 );
       m_pExecUnit-
>WriteData(this,true,&b1_pb,-1);
 } } } }
```

Fig. 8 High-level description of DCT. Although functionally incomplete, algorithm features are captured.

3.2.4 Simulation settings aspect

An image from the simulation setting aspect is shown in Fig. 9. A set named *camera_2p2b* is selected and its members (both execution units *P1*, *P2*, and communication units – dual-port memory *B1_PA*, *B1_PB*) are shown. All the other architectural resources are shadowed. In this way, the designer defines resources used for producing value-change-dump (VCD) traces and console log files during model execution.

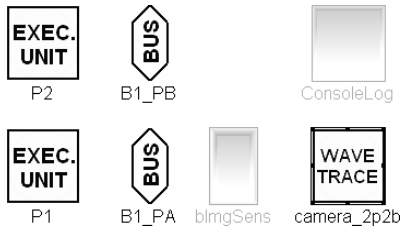

Fig. 9 Selecting architecture resources for waveform traces

3.3 Results

System model evaluation results are obtained after interpretation, compilation and execution of the designed graphical model for each of three architectures. Analysis of these results provides basis for further design decisions.

Table 2 summarizes utilization results of architecture resources and total amount of time for single image processing for all three implementations. The table is split into the architectural part and functionality part (Tasks). The architectural part shows the resource execution time (RET), i.e. summation of the time the services are required from a specific resource. Complementary, the functionality part presents tasks' active and wait timings. The numbers stated in the *active* column represent the percentage of the time a specific task is being actively executed on a specific resource. Similarly, the numbers stated in the *wait* column represent the time a specific task has to wait for a specific resource to become available – this is the time interval during which a task may be executed regarding data dependency, but its execution is not started because of the unavailability of HW resources.

Analysis of the results from Table 2 for architecture No. 1 shows that tasks needed involved in JPEG compression wait for their execution quite large amount of time because of high utilization of processor. It can also be seen that the most time consuming task (52%) is *frwdDCT*. Consequently this also shows that it would be good decision if this task could be mapped on another processor. Analysis of the results for architecture No. 2 shows that total amount of time needed for single image processing has dropped for 17,1%. It can also be seen that contentions on single port memory for image processing present bottleneck, since task *frwdDCT* waits for memory to be freed for 21,35ms. Results for architecture No. 3 shows that total amount of time dropped for 34,3% regarding to architecture No. 1.

Graphical framework also enables generation of VCD waveforms with timing details about tasks execution and usage of architecture resources.

4 Conclusion and future work

We present a modeling framework used for high-level modeling of heterogeneous systems. It provides graphical design elements for using modeling wrappers from system modeling libraries. Graphical modeling relieves the designer of manual typing the source code and thus hides many details of the SystemC code that normally need to be taken care of. Thus the designer can put more effort on actual modeling. Our modeling framework also provides different constraint checks during modeling and integrates support for simulation settings. When modeling is completed, an executable model is automatically generated to simulate the system behavior on a high abstraction level. Our case study exemplifies the use of our framework and shows information obtained from the executable model built on a high-abstraction level. Results of simulation are estimations of architecture resource contention and utilization, time needed for task execution and timing diagrams. Evaluation of these results serves as a basis for model evaluation and further design decisions.

Table 2. Utilization results and total amount of time spent for single image processing

	One processor, single-port RAM		Two processors, single-port RAM			Two processors, dual-port RAM			
Architecture	**P1**	**BUS1**	**P1**	**P2**	**BUS1**	**P1**	**BUS1_PA**	**P2**	**BUS1_PB**
RET[ms]	123,496ms	86,502ms	73,686ms	89,058ms	86,502ms	55,789ms	37,350ms	67,707ms	49,152ms
Tasks [ms]	active/wait	active	active/wait	active/wait	active/wait	active/wait	active	active/wait	active
readImgSens	1,64/0,00	1,64	1,64/0,00	/	1,64/0,00	1,64/0,00	1,64	/	/
setCptrParams	0,00/0,00	/	0,00/0,00	/	/	0,00/0,00	/	/	/
takePicture	1,00/0,00	/	1,00/0,00	/	/	1,00/0,00	/	/	/
readImgPlCorr	9,83/0,00	4,91	9,83/0,00	/	4,91/0,00	9,83/0,00	4,92	/	/
rgb2yuv	15,89/33,25	9,91	26,86/23,57	/	9,91/10,97	15,89/19,27	9,91	/	/
frwrdDCT	67,71/42,26	49,15	/	89,06/0,00	49,15/21,35	/	/	67,71/0,00	49,15
quant	19,98/6,77	14,75	22,10/5,26	/	14,75/2,13	19,98/2,54	14,75	/	/
entrCod	6,84/29,59	5,53	11,59/11,28	/	5,53/4,75	6,84/5,31	5,53	/	/
writeToFlash	0,61/36,52	0,61	0,66/35,87	/	0,61/0,05	0,61/22,51	0,61	/	/
Total [ms]	**123,496ms**		**102,431 ms**			**81,080ms**			

Graphical modeling enables rapid changes in the model (e.g. changes in the mapping aspect can give better results) without time-consuming manual SystemC code rewriting.

Even though high-level and functionally incomplete models are used, it is shown that the results obtained by our modeling framework offer solid further design guidance. The benefits include a greatly reduced exploration time and higher-level algorithm capturing that is especially suitable for component-based data capturing.

In our future work we intend to include support for modeling task interruption and to handle priorities of tasks execution in order to achieve efficient modeling of operating systems. We plan to provide a hierarchical approach to model building as it will significantly improve handling complexity of large models. Our intention is to enable a graphical composition of the task description by providing a library of commonly used methods. The library will provide cost estimations for execution of the methods on various architectural resources.

5 References

[1] M. Finc, A. Žemva. A systematic approach to profiling for hardware/software partitioning. *Computers & electrical engineering*, 31:93-111,2005

[2] A. A. Jerraya. Long Term Trends for Embedded System Design. *CEPA 2 Workshop Digital Platforms for Defence*, Brussels, March 15-16 2005

[3] A. A. Jerraya, W. Wolf. Hardware/Software Interface Codesign for Embedded Systems. *IEEE Computer Society*, 38:63-69, 2005

[4] A. Habibi, S. Tahar. A Survey on System-On-a-Chip Design Languages. *Proc. IEEE 3rd International Workshop on System-on-Chip (IWSOC'03)*, page 212-215, Calgary, Canada, June-July 2003, IEEE Computer Society Press,

[5] L. Cai, D. Gajski. Transaction Level Modeling: An Overview. *CODES+ISSS'03*, page 19-24, Newport Beach, California, USA, October 2003, ISBN:1-58113-742-7

[6] http://www.eclipse.org/gef/

[7] http://www.isis.vanderbilt.edu/Projects/gme

[8] http://www.uml.org/

[9] A. Ledeczi, et al. The Generic Modeling Environment. *Workshop on Intelligent Signal Processing*, Budapest, Hungary, May 17, 2001.

[10] Institute for Software Integrated Systems, Vanderbilt University. *A Generic Modeling Environment: GME 6 User's Manual, Version 6.0.* Vanderbilt University, Nashville, 2006

[11] http://www.systemc.org/

[12] D. C. Black & J. Donovan. SystemC from the ground up. Springer. 2004

[13] J. Dedič. Enovito razvojno okolje za sočasno načrtovanje strojne in programske opreme, doktorska disertacija, Univerza v Ljubljani, Fakulteta za elektrotehniko, 2006

[14] J. Dedič, M. Finc, A Trost. A Framework For High-Level System Design Exploration. *Informacije MIDEM*, 36:151-160, 2006

[15] V. Bhaskaran, K. Konstantinides. Image and Video Compression Standards. Second Edition. Kluwer Academic Publishers. 1997

[16] http://www.ijg.org

