Vol.2011-SLDM-149 No.44
Vol.2011-EMB-20 No.44
2011/3/19

HW/SW Cosimulation Framework Based on Software Component System

Takuya Azumi,[†1] Yuko Hara-Azumi,[†1,†2]
Shinya Honda[†2] and Hiroaki Takada[†2]

As the size and complexity of software applications increase, software component systems are getting used due to its reusability, reduced time-to-market, and hence, reduced software production cost. This paper proposes SysTECS, a highly automated hardware/software (HW/SW) cosimulation framework which integrates a system-level design tool, SystemBuilder, and a software component system for embedded systems, TECS. In existing HW/SW codesign technologies, designers need to manually add or modify HW/SW communication sources (e.g., their size, direction, and allocator) in input behavioral descriptions, which is complex to specify. On the contrary, in SysTECS, the designers can design the overall system at higher abstraction level and have no need to specify the HW/SW communication in the input descriptions because TECS specifically defines the interface between components and the communication sources are automatically generated. This paper demonstrates the effectiveness of SysTECS through a case study with an SHA application.

1. Introduction

Increasing embedded system complexities and strict time-to-market schedules are critical issues in the today's system-level design. To improve the design productivity, designing the systems at a higher abstraction level is necessary[13].

In embedded software domains, meanwhile, software component technologies have become popular[1),2),5),10)]. It has many advantages such as increased reusability, reduced time-to-market, reduced software production cost, and hence, improved productivity[9]. We, therefore, propose a highly automated hardware/software (HW/SW) cosimulation framework named SysTECS integrating a system-level design tool, SystemBuilder[8], and a software component system, TECS (TOPPERS Embedded Component System[5]) .

The contribution of this work is to present a system-level design method based on TECS which provides a higher abstraction level design environment than existing works such as 6), 8), 12). In the existing HW/SW codesign technologies, designers need to manually add or modify HW/SW communication sources (e.g., their size, direction, and allocator) in input behavioral descriptions, which is complex to specify. On the contrary, in SysTECS, the designers can design the overall system at higher abstraction level and have no need to specify the HW/SW communication in the input descriptions because TECS specifically defines the interface between components and the communication sources are automatically generated.

The rest of this paper is organized as follows. Section 2 depicts an overview of SysTECS. Section 3 describes a case study with an SHA application by using SysTECS. Section 4 summarizes this paper.

2. Overview of SysTECS

In this section, a design flow of SysTECS, and a problem analysis and approaches of SysTECS are described in detail.

2.1 Design Flow of SysTECS

Fig. 1 shows the entire design flow using SysTECS. There are three stages until RTL descriptions are generated: stage 1 (TECS stage), stage 2 (SystemBuilder stage), and stage 3 (behavioral synthesis stage).

In Fig. 1, at the stage 1, starting form component descriptions and component sources in TECS[5], (1) HW/SW communication sources and (2) mapping and process definition files are automatically generated by an interface generator. The HW/SW communication is either non-blocking, blocking, or memory primitive, all of which are primitive communication channels provided by SystemBuilder. (3) Each communication primitive defines access functions which can be used as a function call in C programs. SW and HW components specified with TECS communicate with each other through the access functions. Software components are mapped to processes defined in SystemBuilder. A process in SystemBuilder will be implemented as either a software task or a HW module with an FSM, depending on the HW/SW partitioning. SysTECS also generates a specific file

†1 Colleage of Information Science and Engineering, Ritsumeikan University
†2 Graduate School of Information Science, Nagoya University

Vol.2011-SLDM-149 No.44
Vol.2011-EMB-20 No.44
2011/3/19

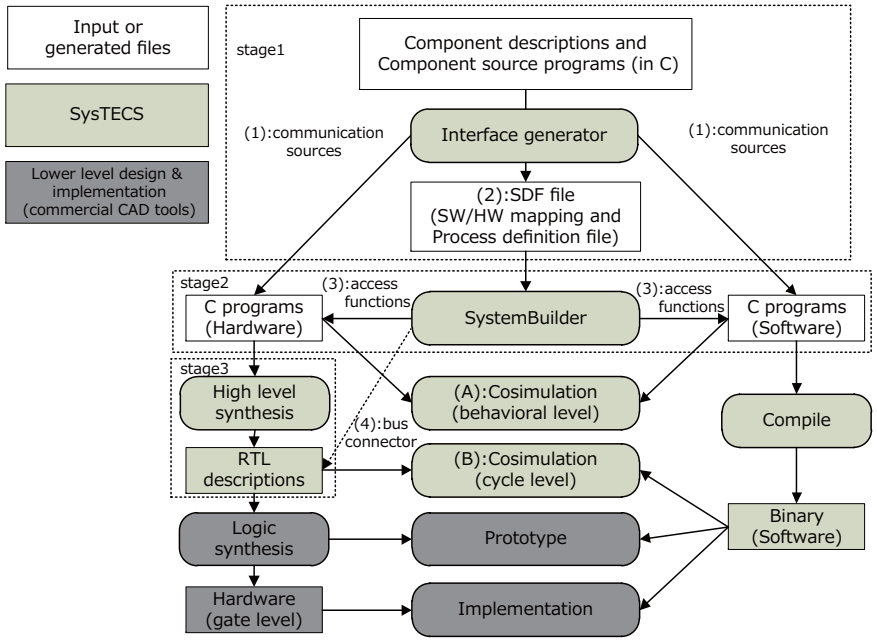

Fig. 1 Entire design flow using SysTECS.

named System DeFinition (SDF) file including processes information and the overall structure, and source code for HW/SW communications. The SDF file is explained in Section 3.

At the stage 2, based on the generated SDF file and sources, (3) the access functions and (4) RTL descriptions for cosimulation are generated by SystemBuilder. At the stage 3, based on the C programs for HW which includes communication sources, eXCite[15], a commercial behavioral synthesis tool, generates RTL descriptions of the HW.

SysTECS supports HW/SW cosimulation at different abstraction levels. SysTECS features cosimulation with (A) functional simulation models of hardware written in C (at a behavioral level) and (B) cosimulation through HDL simulators (at an RTL level).

2.2 Problem Analysis and Approaches of SysTECS

TECS adapts a static configuration to estimate the memory consumption of an entire application and to reduce overhead at runtime. The static configuration implies that both the configuration of the component behavior and the interconnections between components are static. Moreover, TECS supports a variety of component granularities, and a diversity of components, such as an allocator or an RPC channel. TECS adopts a static model that statically instantiates and connects components. The attributes of the components and interface sources for connecting the components are statically generated by the interface generator. Furthermore, TECS optimizes the interface sources. Hence, no instantiation overhead is introduced at runtime, and the runtime overhead of the interface code is minimized[4]. Therefore, these attributes of TECS are suitable for system-level designs. Meanwhile, there are problems to be solved for using SW components without modifying of component sources following: calling components between SW and HW, HW modules translated from SW components, and HW/SW mapping.

SysTECS supports to solve these problems.

- Calling components between SW and HW
  It is realized that an RPC mechanism of TECS[3] adapts calling components between SW and HW. The RPC mechanism generates components between caller and callee components. In particular, a marshaller component for the caller component is to convert function information into a standard data type and to send the data, and an unmarshaller component for the callee component is to receive the data and to reverse the standard data type. To send or receive the data, access functions are used in the marshaller and the unmarshaller components.
- HW modules translated from SW components
  Since TECS components are implemented for embedded software, it is nessesary to translate from these components to HW descriptions. The behavioral synthesis tool synthesizes RTL descriptions from software components mapped to HW modules.
- HW/SW mapping
  As mentioned above, the interface generator also generates HW/SW mapping

Vol.2011-SLDM-149 No.44
Vol.2011-EMB-20 No.44
2011/3/19

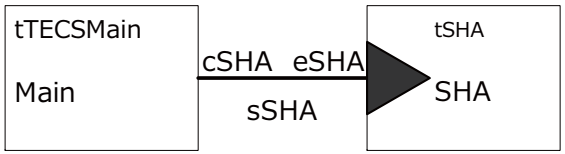

Fig. 2 Component diagram.

information for SystemBuilder from component descriptions in TECS. Then, TECS components are mapped to the processes defined by SystemBuilder. Furthermore, in system-level design, parallelism and pipeline processing should be considered. TECS supports parallelism and pipeline processing on a real-time OS for multi-processors in embedded software. The *oneway* calling is provided to support the parallelism. It means that caller component does not need to wait until a callee component finishes executing. At software level, the parallelism has been already supported in TECS. Therefore, it is possible to adapt the feature for system-level design.

Next section mainly explains calling components between SW and HW, HW modules translated from SW components, and HW/SW mapping.

3. A Case Study: SHA Application

We conducted a case study on SysTECS for an SHA application, which is one of the CHStone benchmarks[7]. In this case study, it is assumed that a componentized SHA application is provided.

Fig. 2 shows a component diagram for the SHA application. Each rectangle represents a *cell* which is an instance of a component in TECS. A *cell* has *entry port* and *call port* interfaces. The right *cell* (*SHA*), which is for an SHA calculation, has an *entry port* which is an interface to provide services (functions) to other *cell*s. The left *cell* (*Main*) which uses the *SHA cell* has a *call port* which is an interface to use services (functions) to other *cell*s.

There are three type of component descriptions to define components in TECS as an input of SysTECS: a *signature* description, a *cell type* description, and a *build* description. In 5), each description is explained in detail.

3.1 Stage 1: Component Description of the Case Stdudy

Fig. 3 shows a *signature* description for the SHA application. A set of function

```
1 signature sSHA{
2  void   transform(
            [in,size_is(length)] const int8_t *indata,
            [in] int32_t length,
            [out,size_is(5)] int32_t *messageDigest);
3  /* omission of other functions */
4 };
```

Fig. 3 SHA signature description.

```
1 [singleton,active]
2 celltype tTECSMain{
3   call sSHA cSHA;
4 };
5
6 celltype tSHA{
7   entry sSHA eSHA;
8 };
```

Fig. 4 SHA celltype description.

heads is enumerated in the body of *signature* keyword in the *signature* description. The *in* and *out* keywords are used to distinguish whether a parameter is input or output. A pointer indicates an array or a value in TECS. In this case, the *indata* and *messageDigest* parameters represent an array. It is necessary to describe the size of an array by using the *size_is* keyword in TECS.

Fig. 4 represents a *celltype* description for the SHA application. A *cell type* is the definition of a *cell*, which is as well as the *Class* of an object-oriented language. A *cell* is an entity of a *cell type*. A *cell type* name, such as *tTECSMain* and *tSHA*, follows a *celltype* keyword to define *cell type*. To declare *entry port*, an *entry* keyword is used (Line 7). Two words follow the *entry* keyword: a *signature* name, such as *sSHA*, and an *entry port* name, such as *eSHA*. Likewise, to declare a *call port*, a *call* keyword is used (Line 3).

The left part of **Fig. 5** describes a *build* description for the SHA application component diagram as shown in Figure 2. The *build* description is used to declare

```
1 cell tSHA SHA{                    1 cell tSHA SHA{
2 };                                2 };
3                                   3
4 cell tTECSMain Main{              4 cell tTECSMain Main{
5                                   5  [through(SysTECSPlugin,"")]
6  cSHA = SHA.eSHA;                 6  cSHA = SHA.eSHA;
7 };                                7 };
```

Fig. 5 SHA build description.

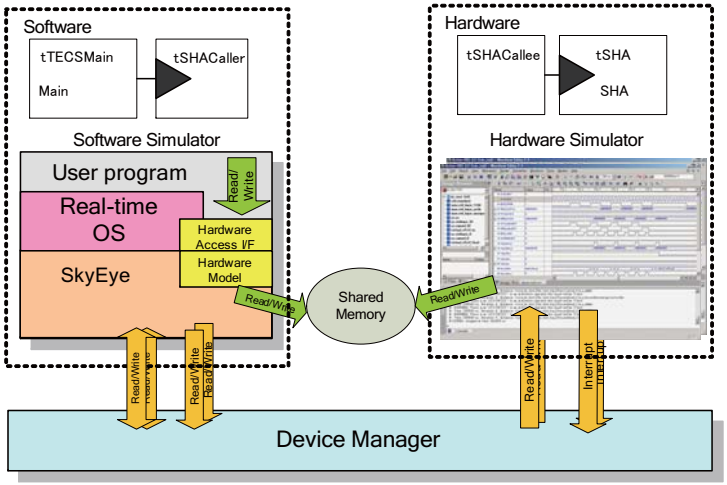

Fig. 6 Cosimulation enviroment in SysTECS.

*cell*s and to connect between the *cell*s for creating an application.

The right part of Fig. 5 represents *build* descriptions for cosimulation as shown in the above part of **Fig. 6**. In this case, the *through* keyword is described just before the description ($cSHA = SHA.eSHA$) of the connection, which includes "SysTECSPlugin" and "". "SysTECSPlugin" represents the type of plug-in. "" gives additional information for the plug-in. In this case, the plug-in is used to generate an SDF file, component descriptions of *tSHACaller* including marshaller sources for SW and *tSHACallee* including unmarshaller sources for HW,

```
 1 SYS_NAME = sp_hw #Design Name
 2 SW = tecs_main #Software partition
 3 HW = tSysTECSPlugin_SHA_callee #Hardware partition
 4 #Communication primitives
 5 SMEM SDRAM, ADDR=0x02100000, DEPTH=0x100000,
              READ_LATENCY=2, WRITE_LATENCY=1
 6 BCPRIM SW_HW_SSHA_BC,  SIZE = 32
 7 BCPRIM HW_SW_SSHA_BC,  SIZE = 32
 8 MEMPRIM SW_HW_SSHA_MEM,  SIZE = 8,
           DEPTH = 16384, LOC=SDRAM
 9 #Process for software
10 BEGIN_FU
11  NAME   = tecs_main #Process name
12  FILE   = "tTECSMain.c", "tTECSMain_tecsgen.c",
             "tSysTECSPlugin_SHACaller.c",
             "tSysTECSPlugin_SHACaller_tecsgen.c"
13  USE_CP = SW_HW_SSHA_BC(OUT), HW_SW_SSHA_BC(IN),
             SW_HW_SSHA_MEM(OUT)
14 END
15 #Process for hardware
16 BEGIN_FU
17  NAME   = tSysTECSPlugin_SHA_callee #Process name
18  FILE   = "tSHA.c", "tSHA_tecsgen.c",
             "tSysTECSPlugin_SHACallee.c",
             "tSysTECSPlugin_SHACallee_tecsgen.c"
19  USE_CP = SW_HW_SSHA_BC(IN), HW_SW_SSHA_BC(OUT),
             SW_HW_SSHA_MEM(IN)
20 END
```

Fig. 7 Generated SDF file.

and C sources including HW/SW communication sources. The below part of Fig. 6 shows the cosimulation environment. The SW parts run on SkyeEye[14], a simulator targeted to ARM based embedded systems, and the HW parts run on Modelsim[11], an HDL simulator.

3.2 Stage2: SystemBuilder Stage and Generated Sources

Fig. 7 depicts a generated SDF file. The overall structure of application specification is described in the SDF file. The application consists of two processes and three communication primitives. Lines 6-7 define blocking communication primitives. Line 8 defines a memory primitive. The *tecs_main* process is mapped

```
1 void
2 eThroughEntry_transform(CELLIDX idx, const int8_t* indata, int32_t length,
                          int32_t* messageDigest)
3 {
4  int32_t tmp_retval;
5   tSysTECSPlugin_SHACaller_CB *p_cellcb;
6  if( VALID_IDX( idx ) ){
7   p_cellcb = tSysTECSPlugin_SHACaller_GET_CELLCB(idx);
8  }
9  /* send function ID  */
10 SW_HW_SSHA_BC_WRITE(FUNC_TRANSFORM);
11 /* send arguments */
12 SW_HW_SSHA_BC_WRITE(length);
13 { /* indata */
14  int32_t i__0;
15  for( i__0 = 0; i__0 < length; i__0+=1 ){
16   SW_HW_SSHA_MEM_WRITE(i__0, indata[i__0]);
17  } /* for ( i__0 ) */
18  /* blocking communication */
19  SW_HW_SSHA_BC_WRITE(1);
20 }
21 /* receive arguments */
22 { /* messageDigest */
23  int32_t i__0;
24  for( i__0 = 0; i__0 < 5; i__0+=1 ){
25   HW_SW_SSHA_BC_READ(&messageDigest[i__0]);
26  } /* for ( i__0 ) */
27 }
28 /* blocking communication */
29 HW_SW_SSHA_BC_READ(&tmp_retval);
30 }
```

Fig. 8 A part of generated component sources for tSHACaller.

```
1 void tSysTECSPlugin_SHA_callee(){
2  /* omit declaration of variables */
3  while(1){
4   /* receive function ID */
5   SW_HW_SSHA_BC_READ(&func_id);
6   switch(func_id){
7    case FUNC_TRANSFORM:/* transform */{
8     /* receive arguments */
9     SW_HW_SSHA_BC_READ(&length);
10    SW_HW_SSHA_BC_READ(&tmp_indata);
11    SW_HW_SSHA_MEM_GET_POINTER(0 ,&indata);
12    /* calling component function */
13    cCall_transform(indata, length, messageDigest);
14    /* send results */
15    { /* messageDigest */
16    int32_t i__0;
17     for( i__0 = 0; i__0 < 5; i__0+=1 ){
18      HW_SW_SSHA_BC_WRITE(messageDigest[i__0]);
19     } /* for ( i__0 ) */
20    }
21    HW_SW_SSHA_BC_WRITE(1);
22    break;
23   } /* case */
24   /* omission of other functions */
25  } /* switch */
26 } /* while */
27 }
```

Fig. 9 A part of generated component sources of tSHACallee.

to SW (Line 2) and the *tSysTECSPlugin_SHA_callee* process is mapped to a HW module (Line 3), respectively. The file lines from *BEGIN_FU* to *END* define each process. The *tecs_main* process is written in four files and uses three communication primitives. For each communication primitive, the direction of the communication is specified.

HW/SW communication sources are described in **Figs. 8** and **9**. Based on communication primitives in the SDF file, HW/SW communication sources include the access functions generated by SystemBuilder.

Two types of access functions for blocking communication primitive are generated: *XXX_READ* and *XXX_WRITE*, where *XXX* denotes a unique name given to the communication primitive such as *SW_HW_SSHA_BC*. For example, *SW_HW_SSHA_BC_WRITE* (Line 10 in Fig. 8) and *SW_HW_SSHA_BC_READ* (Line 5 in Fig. 9) are used to send and receive the function ID, respectively. Three types of access functions for memory primitive are generated: *XXX_READ*, *XXX_WRITE*, and *XXX_GET_POINTER*.

Table 1 shows comparison of lines of code excluding comment and empty lines after each stage in the case study. The second column represents input of

Vol.2011-SLDM-149 No.44
Vol.2011-EMB-20 No.44
2011/3/19

Table 1 Comparison of lines of code.

	input	stage 1	stage 2
# of source code	232	673	866
# of other description	20 (com)	20 (sdf)	0
Total	252	693	866

SysTECS which includes component descriptions and component sources for the SHA application. The third column describes the SDF file and communication sources generated by SysTECS. The fourth column depicts access functions and C source code for HW modules generated by SystemBuilder. The designers only add a *through* mechanism to realize the HW/SW communication. An important advantage of using the *through* mechanism is that it is not necessary to modify the legacy components. Therefore, it is possible to maintain the reusability of components.

4. Conclusion

This paper proposed SysTECS, a highly automated HW/SW cosimulation framework. The advantage of SysTECS is to use software components for system-level design without modifying of input C sources (component sources). Moreover, since TECS supports applications of a real-time OS for multi-processors and SystemBuilder assumes MPSoC as target architecture, SysTECS can deal with more complicated applications. Currently, we are working on adapting a memory allocator in TECS [3] without copying input/output data for more efficient HW/SW communications.

Acknowledgments This research was partially supported by the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Research Activity Start-up 2010-2012(22800071).

References

1) Åkerholm, M., Carlson, J., Fredriksson, J., Hansson, H., Håkansson, J., Möller, A., Pettersson, P. and Tivoli, M.: The SAVE approach to component-based development of vehicular systems, *Journal of Systems and Software*, Vol.80, No.5, pp. 655–667 (2007).
2) AUTOSAR: AUTOSAR Specification, http://www.autosar.org/.
3) Azumi, T., Oyama, H. and Takada, H.: Memory Allocator for Efficient Task Communications by Using RPC Channels in an Embedded Component System, *Proc. the 12th IASTED International Conference on Software Engineering and Applications*, pp.204–209 (2008).
4) Azumi, T., Oyama, H. and Takada, H.: Optimization of Component Connections for an Embedded Component System, *Proc. IEEE/IFIP 7th International Conference on Embedded and Uniquitous Computing*, pp.182–188 (2009).
5) Azumi, T., Yamamoto, M., Kominami, Y., Takagi, N., Oyama, H. and Takada, H.: A New Specification of Software Components for Embedded Systems, *Proc. 10th IEEE International Symposium on Object/component/service-Oriented Real-Time Distributed Computing*, pp.46–50 (2007).
6) Dömer, R., Gerstlauer, A., Peng, J., Shin, D., Cai, L., Yu, H., Abdi, S. and Gajski, D.D.: System-on-Chip Environment: A SpecC-Based Framework for Heterogeneous MPSoC Design, *EURASIP Journal on Embedded Systems*, Vol.2008, pp.1–3 (2008).
7) Hara, Y., Tomiyama, H., Honda, S. and Takada, H.: Proposal and Quantitative Analysis of the CHStone Benchmark Program Suite for Practical C-based High-level Synthesis, *Journal of Information Processing*, Vol.17, pp..242–254 (2009).
8) Honda, S., Tomiyama, H. and Takada, H.: RTOS and Codesign Toolkit for Multiprocessor Systems-on-Chip, *Proc.In 12th Asia and South Pacific Design Automation Conference*, pp..336–341 (2007).
9) Lau, K.-K. and Wang, Z.: Software Component Models, *IEEE Transactions on Software Engineering*, Vol.33, No.10, pp.709–724 (2007).
10) Loiret, F., Navas, J., Babau, J.-P. and Lobry, O.: Component-Based Real-Time Operating System for Embedded Applications, *Proc. 12th InternationalSymposium on Component-Based Software Engineering*, pp.209–226 (2009).
11) Mentor Graphics: . `http://model.com/`.
12) Nikolov, H., Thompson, M., Stefanov, T., Pimentel, A.D., Polstra, S., Bose, R., Zissulescu, C. and Deprettere, E.F.: Daedalus: toward composable multimedia MP-SoC design., *Proc. International 45th Design Automation Conference*, pp.574–579 (2008).
13) Sangiovanni-Vincentelli, A.: Quo vadis, SLD? reasoning about the trends and challenges of system level design, *PROCEEDINGS-IEEE*, Vol.95, No.3, p.467 (2007).
14) SkyEye: . `http://www.skyeye.org/index.shtml`.
15) Y Explorations: . `http://www.yxi.com/`.