ISSN 0002-306X. Изв. НАН РА и ГИУА. Сер. ТН. 2013. Т. LXVI, № 2.

UDC 530.15:004.032.2

COMPUTER SCIENCE AND INFORMATICS

A.A. KHZARJYAN

AN APPROACH OF STAR MEMORY SYSTEM USE FLOW AUTOMATION AND ITS VERIFICATION

Modern SoC design restricted with Time to Market and yield. A widely used IP block in SoC is an embedded memory which is more inclined to defects. One of the well-known Infrastructural IP is a STAR Memory System (SMS) which is a general solution of BIST and repair. This paper introduces an approach of SMS use flow template library construction with application of formal verification algorithm. It is implemented as a supporting tool to optimize the SMS use flow design and verify the customer needs.

Keywords: STAR Memory System, use flow, formal verification, System-on-Chip, build in self-test, intellectual property.

Introduction. Every new semiconductor technology node provides further miniaturization and higher performance. On the other hand, the growth in demand for System-on-Chips (SoCs) has spurred a flood of better, faster, smaller chips. The creation of such SoCs necessitates using several embedded IP blocks from different vendors. Most of the known IP blocks, though, are functional ones, such as embedded processor, embedded memory, embedded analog, etc. Rather, infrastructure IP is embedded in an IC solely to ensure its manufacturability and lifetime reliability [1].

It is reasonable to notice that embedded memory IPs become the major component of SoC that will occupy more than 94% SoC area in the year 2014 [2]. In the aspect of manufacturing yield, embedded memories are more inclined to defects than other SoC components. To improve the yield, the embedded memories should be armed with redundancy [3]. In general, SoC obtains the BIST that is used to perform only testing while BIRA and BISR [4]. Components of the engine are necessary for repairing the embedded memories. One of the well-known build in test and repair solutions is a STAR Memory System (SMS) [3] that is a complete solution of build in test and repair which provides a full set of infrastructural IP compilers with the corresponding generation, insertion and verification tools.

Customers usually use different IP blocks with a wide range of SMS components to build their SMS use flows. It means that the customers have to learn all SMS components with taking into account all their specific details. As a result, the use flow design can become time consuming and an error prone process. One of the possible ways of using the flow design optimization is encapsulation of its complexity by providing a standard mechanism of SMS usage. Analysis of various customer use flows has revealed that there are some standard use flows of SMS usage. Those use flows are templates of SMS usage flow customization that are similar to the well-

known ITIL templates [5]. Similarly, SMS use flow templates can form a template library which can be provided to simplify the customers' work. In general, the use flow template library (UFTL) will serve as a basis for designing specific use flows. The use flow design tool will be provided to the customer to adopt the proposed templates to their own cases. They can also extend the library by inserting new templates into the provided library. Modification of the basic template will require verification of changes.

The paper introduces an approach to the UFTL construction which is carried out by the language for building SMS components. A converter is proposed to transfer the use flows to workflow processes. An approach to the UFTL formal verification based on formal verification algorithm of workflow processes [6, 7] is also presented in the paper. This algorithm was previously used for ITIL verification. The illustration of the approach application is illustrated on the most useful use flow template which is a SMS usage default flow. The modification of the mentioned flow is a customer-driven case. The application of the formal verification on it has been performed to check correctness of the modified use flow template.

1. Template-based language of SMS use flow design. It is necessary to define a language that can be used to implement any custom use flow of SMS design. The language has to support implementation of each use flow of SMS design and verification (DVP). At first, it has to support the definition of each IP block that can be used during SMS DVP. The next requirement is to support the addition of definitions for each new SMS DVP flow by its automation language. It means that the language has to be general as much as it is possible. One of the well-known methods of language generalization is its construction based on templates. Usually template-based languages have possibilities to extend the set of their predefined templates. Our approach to the SMS DVP automation is based on template-based language implementation which will be introduced below.

A proposed language, called SMS DVP Template Language (DTL), implements SMS DVP requirements by supporting the IP compiler libraries for each vendor and the DVP flow modifiable definitions. DTL contains construction for describing the elements of DVP. It also provides presentation of IP compiler hierarchy, data hierarchy classification, Design and Verification Information. DTL constructs are based on the main concept: "Everything is an element" (Fig. 1).

BNF like syntax forms are used to describe the main constructs of DTL. The scheme of the DTL construct has the following structure:

<element> – is the element name which is defined as a template in DTL.

<item> – zero or more items that are specific for the described element. Each item also could be simple (atomic) or complex (list of sub items).

';', '\n' – corresponds to the dot comma punctuation mark or a new line respectively.

Generally, there are two types of elements that can be used in DTL: simple and complex. The simple element has only atomic items while the complex element container item has a nested hierarchy that can be made up of simple and complex elements. There is no restriction on the complexity or depth of the nested hierarchy.

The simple element scheme is presented in (Fig. 2).

```
<element> {<item>} (';'|'\n')
```

Fig. 1. Scheme of DTL construct

```
<simple_element> {<simple_item>} (';'|'\n')
```

Fig. 2. Simple element

The complex element general scheme, which is more suitable for describing vendors IP, is presented in (Fig. 3).

Besides the flexibility of data hierarchy description, DTL also provides a possibility of IP compiler classification which is necessary to support each aspect of various vendors' similar IP in one DTL template. For example, each IP compiler has its own specific parameters which are introduced as a set of simple elements. As a result, the classification enables to specify individual features of each IP compiler only in its classified template as shown in (Fig. 4).

```
<complex_element> {<simple_item>} '{'
    'class' <vendor>'.'<IP_compiler> [(';' | '\n')]
  | {{<simple_element>}
  | {<complex_element>}}
'}' [(';' | '\n')]
```

Fig. 3. Complex element

```
element ip_type {
        class vendor.ip_compiler_name;

        element parameter;
        ...
        element section {
            element subparameter;
            ...
        };
};
```

Fig. 4. Definition of IP compiler

IP compilers' infrastructural hierarchy is also possible to describe by using DTL. There are two ways of describing IP compilers' hierarchy in DTL. The first is designing the DTL templates by placing them in planar structure (Fig. 5). In this case, their dependencies will be realized through references implemented by simple elements. The second is designing the DTL templates in the nested structure (Fig. 6). Selecting the hierarchy representation structure is the vendors, preference.

```
element IP_type1 {
        class vendor.ip_compiler1;
        element ip_type2;
        ...
};
element IP_type2 {
        class vendor.ip_compiler2;
        element ip_type3;
        ...
};
...
```

Fig. 5. Planar Structure of IP compilers

```
element IP_type1 {
        class vendor.ip_compiler;
            element IP_type2 {
                    class vendor.ip_compiler;
                    element ip_type3;
            ...
            }
        ...
};
```

Fig. 6. Nested Structure of IP compilers

The IP compiler definition has to be used to define each IP block. The definition of IP block is supposed to identify all the necessary parameters of the corresponding IP compiler by their values. Each IP block contains reference to its IP compiler by using its classification. The scheme of IP block definition is presented in (Fig. 7).

If the IP block is a part of infrastructure hierarchy it has to be defined as part of it. In the case of planar structure, each IP block has a reference to its sub-block except the last one (Fig. 8).

```
ip_type ip_block_name {
        class vendor.ip_compiler_name;

        parameter1 value1;
        ...
};
```

Fig. 7. IP block definition

```
ip_type1 ip_block_name1 {
        class vendor.ip_compiler_name;

        subblock ip_block_name2;
        ...
};

ip_type2 ip_block_name2 {
        class vendor.ip_compiler_name;

        subblock ip_block_name3;
        ...
};
...
```

Fig. 8. Description of IP blocks in SoC

The generation step of SMS DVP can be automated based on the information which is described above. The insertion step of SMS DVP requires the definition of SoC by specifying its IP blocks and the necessary information for insertion. The general scheme of SoC description is presented in (Fig. 9).

An example of a simple use flow which implements DVP of SMS is presented in Fig. 10. The example is given based on the constructs of DTL.

```
soc soc_name {
        ip_blocks {
                ip_block1 ...
                ip_block2 ...
                …
        }
        …
};
```

Fig. 9. Description of IP blocks in SoC

```
memory mem1 {
        class vendor1.compiler1;
        NW 1024;
        NB 23;
        CM 8;
};
memory mem2 {
        class vendor2.compiler2;
        NW 512;
        NB 36;
        BK 16;
};
wrapper wr1 {
        class vendor3.compiler3;
        memory mem2;
};
wrapper wr2 {
        class vendor3.compiler4;
        memory mem1;
        FREQ 100MHZ;
};
processor proc1 {
        class vendor3.compiler4;
        wrapper {wr1 wr2};
};
server srv1 {
        class vendor3.compiler5;
        NVS efuse128;
        processor {{proc1 3}};
};
```

Fig. 10. Use flow example

2. Mapping of the use flow templates on workflow processes. The formal model of workflow processes is based on the model of IBM's MQSeries Workflow [8] that has been extended by adding the necessary formalism to consider the formal verification problem [6, 7].

The workflow model components are ***activities*** and ***connectors***. The activities are associated with a context being defined as data passing to an activity. It is called ***input container***. An activity also returns data called ***output container***. Control and data connectors provide connections between the activities. A control connector has an associated Boolean predicate called ***transition condition***. A directed graph based on sets of activities and control connectors is called ***control flow*** of a workflow/business process. Full details can be found in [6-8].

Mapping of DTL on the workflow process model can be described as follows:

- The IP blocks are mapped to activities.
- The IP blocks infrastructural hierarchy can be presented by control connectors.
- The parameters of each IP block can be presented as activity data container elements.

An example of a workflow process is presented in the next section.

3. An example of formal verification algorithm application on a SMS use flow template. Let us illustrate the application of formal verification algorithm on one of the use flows. Fig 11 shows the workflow of use flow template sample that is most used by our customers. It's a use flow template of SMS usage default flow. For the verification of the given process, a precondition and a postcondition should be specified [6,7]. The specific conditions are created based on the needs of verification against the definite aspects of the process behavior.

PreC = i(Read).Base $\neq \perp$,where $\perp$ denotes the unknown value of the variable.
PostC = (Serror = TRUE OR Sserver = PASS) AND Send = TRUE.

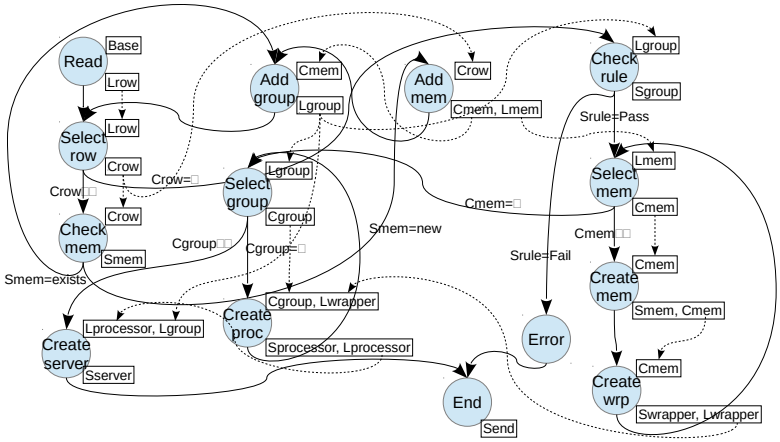

Fig. 11. Workflow process of default use flow

The application of the formal verification algorithm on the described process will identify the presence of cycles in it. A detailed analysis of cycles will show that process cycles are intervals. The first step of the algorithm will reduce the cyclic graph to the acyclic one [7]. It will initially construct the set of the first order intervals - S_C={*<Select row, Check row, Add mem Add group>, <Select Mem, Create mem, Create Wrap>, <Select group, Create proc>*. The next step is the replacement of intervals by corresponding equivalent activities *<Form group, Form mem wrap, Form proc>* [7]. The exit transitions of new activities have to contain branching information the corresponding cycle that is interpreted in terms of branching state registers. They are cycle invariants. For instance, *Form group* activity has a transition to *Check group*. Its transition condition is formulated from transition condition of *Select row* to *Check group* with addition of a branching register BrSr. *BrSr = 1* presents the execution path *<Read, Select row, Check group,…>* and *BrSr = 2* is presenting execution path *<Read, Select row, Check mem,…>*. Transition conditions of other new activities are constructed similarly. Reduction of the mentioned intervals by equivalent activities will result in a new process (Fig. 12). The second phase analysis of the graph means that it is acyclic. The acyclic process verification algorithm [6] has to be applied to the reduced process presented in Fig. 12. After the execution of the verification algorithm and after checking the correctness condition, we find that the process is *correct*.

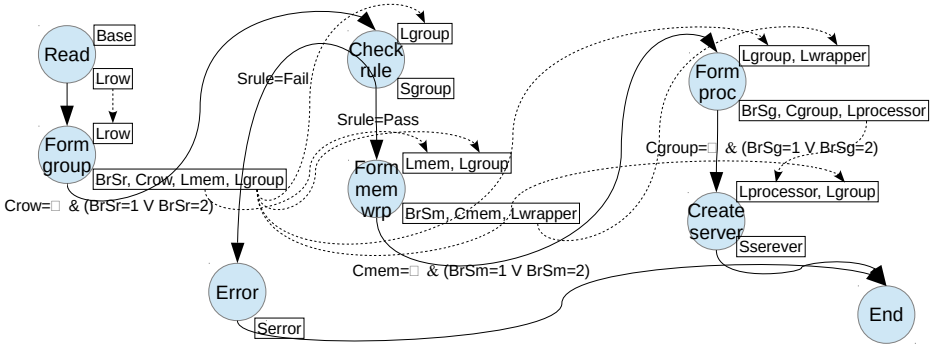

Fig. 12. Reduced process

Some of the presented activities can be modified by the user. They *are Create proc, Creat mem, Creat wrp, End* activities.

To improve the overall quality of this process *Analyze* and *Notify* additional tasks have been added to the process (Fig. 13) by one of our customers. This activity analyzes if everything execution of a process. In case of an abnormally executed process, the activity *Notify* would notify about it.

A new postcondition:

PostC = (Serror = TRUE OR Sserver = PASS) AND Sanalyzed=TRUE AND Spassed=TRUE AND Send = TRUE

As a result of applying similar steps of the process transformation and verification, the condition of **incorrect** processes will be satisfied [6]. The control connector between the activities *Notify* and *End* has to be removed to correct the modified template logic. The postcondition has to be changed to:

PostC = (Serror = TRUE OR Sserver = PASS) AND Sanalyzed =TRUE AND ((Spassed =TRUE AND Send = TRUE) OR (Spassed =FALSE AND Snotify = TRUE)).

Applying the algorithm to the corrected process will result in the satisfaction of the correct process condition [6].

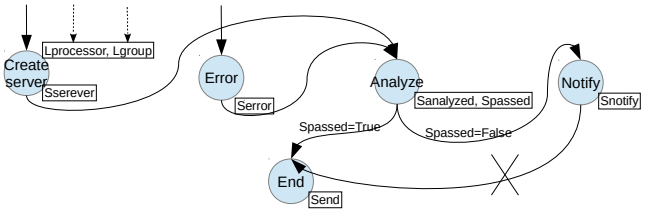

Fig. 13. Modified section of the process

Conclusion. The approach to the template-based language is proposed which is used as a basis for development of automated environment for design and verification use flows of the STAR Memory System. The main purpose of the proposed language is optimization of SMS use flows by encapsulation of its complexity by providing a common environment to SoC designer. A use flow template library (UFTL) is offered to design specific use flows by modification of the proposed templates by customers. A formal verification algorithm of workflow processes has been proposed to verify the correctness of the customers' use flows after the modification. The application of the presented approach is illustrated on a SMS default use flow.

REFERENCES

1. **Zorian, Y.** What is Infrastructure IP // IEEE Design & Test of Computers.- May-June, 2002.- Vol. 19, No 3.- P. 5-7.
2. The National Roadmap for Semiconductors. - 2000.
3. **Shoukourian S., Vardanian V., Zorian, Y.** SoC Yield Optimization via an Embedded-Memory Test and Repair Infrastructure // Design & Test of Computers IEEE. - 2004.- Vol. 21.- P.200 – 207.
4. **Rei-Fu Huang, Chen Chao-Hsun, Wu Cheng-Wen.** Economic Aspects of Memory Built-in Self-Repair // IEEE Design & Test of Computers.- 2007. – Vol. 24.- P. 164 - 172.
5. IT Infrastructure Library, IT Service Management, Office of Government Commerce, http://www.itil.co.uk/.
6. **Kostanyan, A., Varosyan, A.** Partial Recognizing Algorithm for Verification of Workflow Processes // FUBUTEC.- Porto Portugal, 2008.- P. 89-94.
7. **Kostanyan, A., Matevosyan V., Shoukourian S., Varosyan A.** An Approach for Formal Verification of Business Processes // BIS'09 SpringSim'09 San Diego USA.- 2009. - Article 134. – P. 1-8.

8. **Leymann, F., Roller D.** Production Workflow: Concepts and Techniques // Prentice Hall Press, 2000.- 479 p.

"Synopsys Armenia" CJSC. The material is received 12.04.2013.

Ա.Ա. ԽԶԱՐՋՅԱՆ

«ՍԹԱՐ» ՀԻՇՈՂՈՒԹՅԱՆ ՀԱՄԱԿԱՐԳԻ ՕԳՏԱԳՈՐԾՄԱՆ ԸՆԹԱՑՔԻ ԱՎՏՈՄԱՏԱՑՄԱՆ ԵՎ ՍՏՈՒԳՄԱՆ ՄՈՏԵՑՈՒՄ

Ժամանակակից համակարգ բյուրեղի վրա (ՀԲՎ)-ի նախագծումը սահմանափակված է դեպի շուկա ժամանակի և օգտակարության ելքով: Լայնորեն օգտագործվող մտավոր սեփականության (ՄՍ) կտորը ՀԲՎ-ում ներդրված հիշողությունն է, որն ավելի հակված է արատների: Հայտնի ենթակառուցվածքային ՄՍ է «ՍԹԱՐ» հիշողության համակարգը (ՍՀՀ), որը ընդհանուր լուծում է՝ ներդրված ինքնաթեստավորման և վերանորոգման համար: Ներկայացվում է ՍՀՀ օգտագործման ընթացքի կաղապարների գրադարանի կառուցումը՝ կիրառելով ձևային ստուգման ալգորիթմի մոտեցումը: Այն իրականացվել է որպես աջակցման գործիք՝ օպտիմալացնելով ՍՀՀ օգտագործման ընթացքի նախագծումը և ստուգումը՝ հաճախորդների կարիքներից ելնելով:

Առանցքային բառեր. «ՍԹԱՐ» հիշողության համակարգ, օգտագործման ընթացք, ձևական ստուգում, համակարգ բյուրեղի վրա, ներդրված ինքնաթեստավորում, մտավոր սեփականություն:

А.А. ХЗАРДЖЯН

ПОДХОД К АВТОМАТИЗАЦИИ И ПРОВЕРКЕ ПРОЦЕССА ИСПОЛЬЗОВАНИЯ СТАР - СИСТЕМЫ ПАМЯТИ

Современное проектирование системы на кристалле (СнК) ограничено временем выхода до рынка и полезным выходом. Широко используемый блок интеллектуальной собственности (ИС) в СнК - это встроенная память, которая более склонна к дефектам. Одна из известных инфраструктурных ИС СТАР - системы памяти (ССП) - это обобщенное решение для ВСТ и ремонта. Предлагается подход к конструированию библиотеки шаблонов процесса использования ССП и применения формального алгоритма проверки. Данный подход реализован в виде инструмента поддержки оптимизации проектирования использования ССП и проверки для нужд клиентов.

Ключевые слова: СТАР - система памяти, процесс использования, формальная проверка, система на кристалле, встроенное самотестирование, интеллектуальная собственность.