INTERNATIONAL JOURNAL OF CIRCUIT THEORY AND APPLICATIONS
Int. J. Circ. Theor. Appl. (2012)
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cta.1874

LETTER TO THE EDITOR

Synthesizing VHDL from Activity Models in UML 2

Tomás Balderas-Contreras*,†, René Cumplido and Gustavo Rodríguez

Computer Science Department, Instituto Nacional de Astrofísica, Óptica y Electrónica, Tonantzintla, Puebla, Mexico

ABSTRACT

This document describes a synthesis technology that generates structural VHDL code from models describing the flow of data required to perform algorithms operating on bit-blocks. The models are built using restricted activity diagrams in the Unified Modeling Language version 2. The code generator is developed using Acceleo, a technology to implement transformations from models to text. The technology described in this paper exploits the principles of object orientation and model-driven engineering. The primary aim is to improve productivity and alleviate complexity during the design of digital hardware systems that implement demanding operations used by a wide variety of computing devices. The use of the technology is illustrated with the generation of VHDL code from models describing a block cipher algorithm. Copyright © 2012 John Wiley & Sons, Ltd.

Received 19 February 2012; Revised 22 July 2012; Accepted 17 October 2012

KEY WORDS: model-driven engineering; UML 2; domain-specific modeling; meta-modeling; code generation; VHDL

1. INTRODUCTION

Electrical and electronics engineers have used several technologies, methodologies, and levels of abstraction to design computer-based systems in the last decades. Implementing advanced architectural techniques in early computers based on vacuum tubes and solid-state transistors was very difficult. With the introduction of integrated circuits (IC), the designer developed schematics that described a computer as a set of interconnected ICs. The increasing complexity of modern systems implemented in VLSI ICs encouraged the use of hardware-description languages, like Verilog and VHDL, to describe their functionality. Not only has VHDL been used for describing digital hardware systems, but also to describe analog and mixed-signal circuits [1].

Current efforts to raise the level of abstraction promote the use of languages commonly employed to develop software systems, like C and Java, to describe the functionality of digital hardware systems. At the electronic system level of abstraction, the designer 'utilizes the appropriate abstractions in order to increase comprehension about a system and to enhance the probability of a successful implementation of functionality in a cost-effective manner, while meeting necessary constraint' [2]. In this context, we explore the possibility of applying recent advances in software engineering to aid in the design of digital hardware systems.

Model-driven engineering (MDE) is a recent paradigm intended to raise the level of abstraction further when developing software systems. This approach conceives the solution to a problem as a set of models expressed in terms of concepts in the problem's domain space, those that the designers and/or customers know very well, instead of concepts in the solution space, those related to software and hardware

*Correspondence to: Tomás Balderas-Contreras, Computer Science Department, Instituto Nacional de Astrofísica, Óptica y Electrónica, Tonantzintla, Puebla, Mexico.
†E-mail: balderas@ccc.inaoep.mx

technologies [3]. The intention is to translate the designer's models into the appropriate implementation for a specific platform and to hide the complexities of such platform's hardware and software. The motivation behind this paradigm is to improve both short-term productivity (increase functionality) and long-term productivity (lengthen longevity) during the development process [4].

The model-driven architecture (MDA) initiative, proposed by the Object Management Group (OMG), is a realization of the MDE paradigm [5]. It attempts to define a MDE-based framework using the OMG's standards, including the Unified Modeling Language version 2 (UML 2) [6, 7]. The OMG also maintains standards describing transformations between modeling languages [8] and between modeling languages and programming languages [9].

This document describes the first steps towards the implementation of a MDA-based design flow to implement digital hardware systems from domain-specific modeling languages. It is possible to build UML 2 models that represent certain algorithms and transform them automatically into a functional description suitable for implementation in a silicon platform (FPGA or ASIC). As an example, we show how activity diagrams in UML 2 can be adapted to model block cipher algorithms and then processed by a transformation tool that generates VHDL code from them. The ultimate goal is to contribute to alleviate the complexity of current systems and increase the productivity of the designers. There are some other proposals to synthesize VHDL from UML [10, 11]; however, such proposals do not raise the level of abstraction when building models in UML.

The rest of this document describes our proposal. Section 2 describes how to tailor UML 2 activity diagrams to model block cipher algorithms. This application domain was selected to illustrate our technology because block cipher algorithms become more important as computer-based systems require more strict security mechanisms. Section 3 illustrates the process of generating VHDL code from the diagrams. Finally, Section 4 concludes.

2. HIGH-LEVEL MODELING OF BLOCK CIPHERS USING UML 2

KASUMI is a block cipher algorithm consisting of a Feistel structure with eight rounds, with each round invoking other operations that also have a Feistel structure [12]. Figure 1 shows the elements that make up the algorithm, how the input plaintext block is split into sections, the operations performed on every

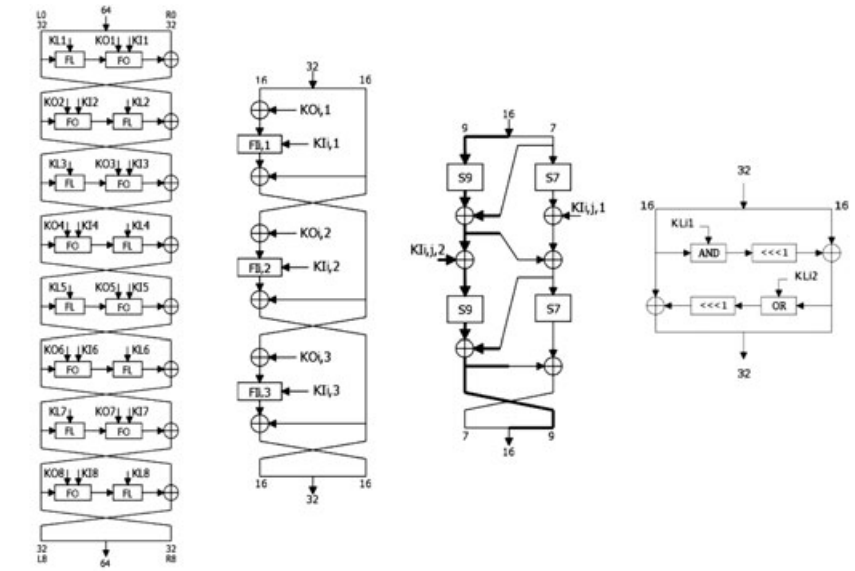


Figure 1. The components of the block cipher algorithm KASUMI.

DOI: 10.1002/cta

section, and the flow of data along every component. Notice that the diagrams in Figure 1 are neither UML diagrams nor block diagrams of a hardware architecture that implements the algorithm. It is possible to manipulate the structure of KASUMI to build implementations that consume fewer resources or achieve higher performance [13, 14]. An activity diagram in UML 2 may be used to represent the flow of data and the operations required by the components of KASUMI. The structure of these models could be modified according to the designer's strategy, and the resulting diagrams could be transformed to a representation ready to be implemented in a hardware platform.

UML 2 diagrams are commonly used to describe the structure and behavior of object-oriented software systems. However, not only that, it turns out that UML 2 is a language that has object orientation at its foundations. Every diagram in UML 2 is a composite object made up of simpler objects linked to each other. These objects are instances of the classes that populate the meta-model of UML 2 [7], which is the definition of UML 2 and can be thought of as the model of all of the models built using UML 2. By studying the meta-model, it is possible to determine how to adapt UML 2 diagrams for accurate modeling of block ciphers. The activity diagram in UML 2 is the perfect candidate for adaptation because it allows modeling data flow.

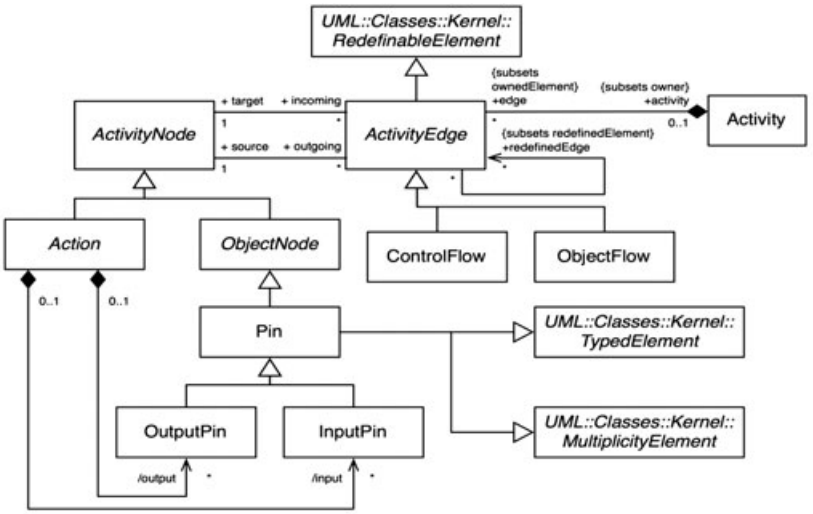
An activity consists of a number of nodes interconnected by edges. There are nodes representing the execution of operations, input and output parameters to the activity, decisions, constant values, concurrent flow of control, and other concepts. The edges in an activity represent transfers of data or control between nodes. Figure 2 shows the classes in the meta-model that define some of the nodes and edges that are useful to model block cipher algorithms. These classes form a specialization hierarchy where every class is a specialization of a more abstract concept or class. Describing the meta-model is beyond the scope of this paper, the interested reader is referred to the references for detailed information [4,15].

Since an activity diagram may be used to model flows of data in a wide variety of areas, it is necessary to define a number of constraints and precisions in order for the diagram to model block cipher algorithms accurately. To adapt the activity diagram to this application domain, we shall restrict the values of the attributes of the objects that make up the model and establish what kind of nodes can be connected together. The transformation tool determines if an input model meets such restrictions before generating the corresponding VHDL code. The following list enumerates some of the restrictions:

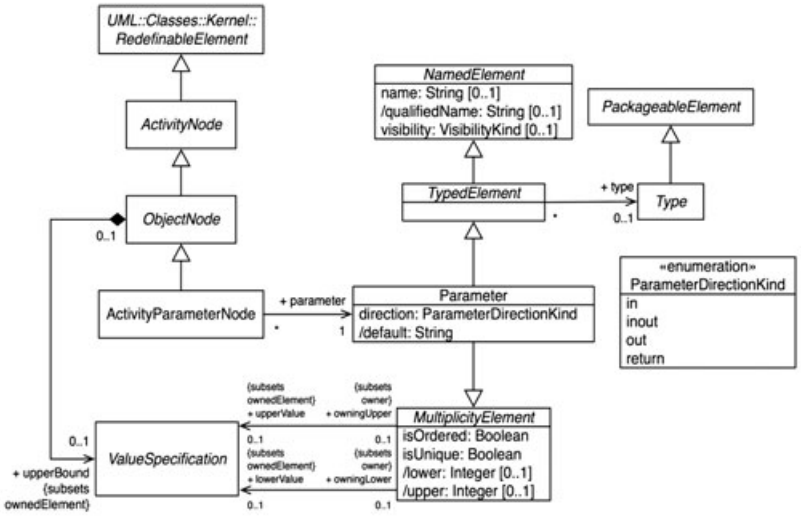
1. A well-formed bit-block is an ordered sequence of objects that are instances of the class Bit. The length of the sequence shall be non-zero and finite.
2. Every activity shall be named.
3. Every activity shall have parameters (instances of the class ActivityParameterNode) that send/receive a continuous flow of bit-blocks of a fixed length. The length of the bit-blocks may differ from one parameter node to another.
4. The only edges allowed are those that model flow of data (instances of the class ObjectFlow).
5. Two nodes connected by an edge shall process bit-blocks of the same length.
6. Every operand (an instance of the class Pin) of an operation node (an instance of concrete sub-classes of the class Action) shall receive a continuous flow of well-formed bit-blocks coming from other operation nodes through edges.
7. A node shall not be the source or target node of multiple edges.
8. Every operation node shall specify an operation supported by block cipher algorithms (xor, and, or, shift left, shift right, rotate left, rotate right, split, merge). In case of nodes specifying constants, these shall be integer values.
9. Every kind of operation node shall have the appropriate number of input and output operands:

a. An operation node indicating a binary bitwise logic operation (xor, and, or, nand, nor, xnor) shall have two input operands and one output operand. All of the operands shall process bit-blocks of the same length.
b. An operation node indicating a shift or rotate operation (shift left, shift right, rotate left and rotate right) shall have one input operand and one output operand processing bit-blocks of the same length. A second input operand is needed to specify the number of bits to shift/rotate.
c. An operation node indicating a constant value shall have a single output parameter and no input parameters.

DOI: 10.1002/cta



(a) An instance of the class Activity owns a number of nodes (instances of the class ActivityNode) interconnected by edges (instances of the class ActivityEdge)



(b) The nodes representing input and output values are instances of the class ActivityParameterNode and are related to instances of the class Parameter

Figure 2. Fragment of the meta-model of UML 2 for activity diagrams.

d. An operation node indicating the split of its single input operand shall have one or more output operands. The sum of the lengths of the output operands shall equal the length of the input operand.
e. An operation node indicating the merge of the multiple input operands shall have a single output operand. The sum of the lengths of the input operands shall equal the length of the output operand.

10. The restrictions defined for operands shall be met as part of the validation of the owning operation.

The activity models in Figure 3 observe the previous restrictions; they represent the components of the algorithm shown in Figure 1 and indicate the operations required to carry out the encryption process. For simplicity, the algorithm was not manipulated before generating VHDL code, but the manipulation of models comes in handy when the designer requires evaluating multiple design

DOI: 10.1002/cta

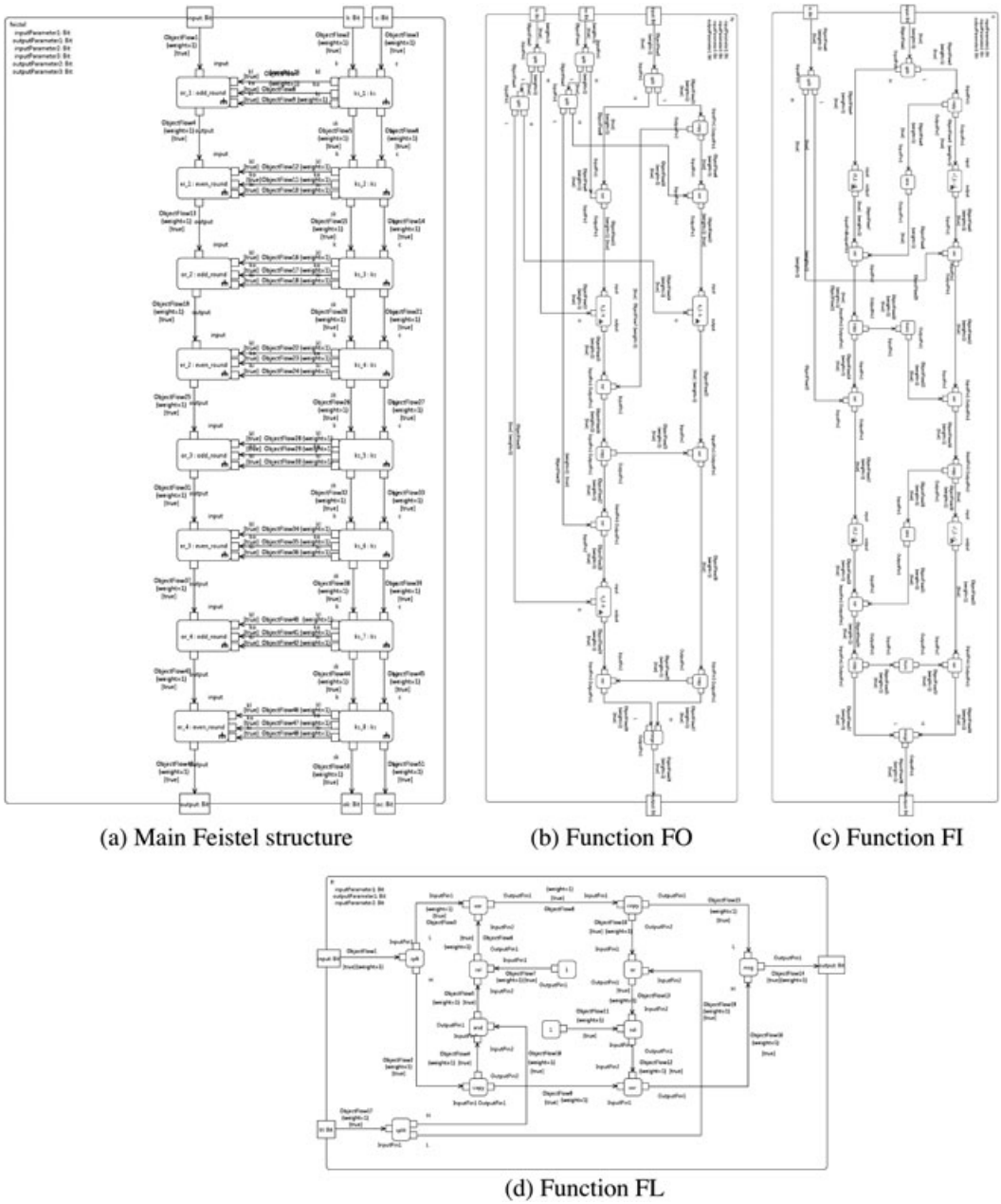


Figure 3. Activity diagrams in UML 2 for the KASUMI block cipher.

strategies to implement a block cipher algorithm. The transformation tool validates that all of the restrictions are met before generating code. The formal mechanism to extend UML 2, known as profiling, allows adding restrictions to the semantics of the modeling elements, new notation, and additional attributes that could be needed.

3. TRANSFORMATION OF MODELS INTO VHDL CODE

The transformation tool is developed using Acceleo [16], a technology built on top of Eclipse that implements the OMG standard for model to text transformation (M2T) [9]. Our project consists of a number of modules containing templates that generate the skeleton of the source code. The templates are filled out with values from attributes of the modeling elements or other

DOI: 10.1002/cta

complex values computed by queries. The restrictions stated previously are implemented using queries written in Acceleo, which are applied to the incoming activity model to validate it before transforming it into VHDL code.

The query shown in Table I is applied to every action in the model to validate the restriction 6 stated above; it returns true if the model meets the restriction. The query is written in a declarative language that is reminiscent of the Object Constraint Language used to specify constraints in the meta-model of UML 2 [17].

The query checks that every input pin (input parameter) of an action is connected to another node and receives a continuous supply of well-formed bit-blocks. The value of the query is true when the operation forAll returns true, which occurs when the condition inside the operation evaluates to true for every input pin (collection input) of the action on which the query is applied (anAction). The condition determines whether the current input pin receives bit-blocks of a finite length (validatePinType() and validatePinMultiplicity()), the number of received bit-blocks is unbounded (validatePinUpperBound()) and is connected to another node through an edge (isConnected()). The transformation tool contains about 200 queries that validate the model and retrieve information from the modeling elements.

In addition to queries, the modules comprising the transformation tool also contain templates that generate structural VHDL code in the output files. The transformation tool generates one design file in VHDL for every activity in the model. The body of a template contains fragments of the ultimate code, invocations to other templates to generate other segments of code, invocations to queries to get values that are to be included in the code, direct accesses to the attributes of the modeling elements to get additional values to complete the code, and control structures like loops and conditionals. The transformation tool contains six modules, each generating a specific section of the VHDL code describing the design of the component.

The templates generate a sub-set of VHDL that conforms to a reduced version of VHDL's grammar [18]. The grammar was simplified because the structural description of block ciphers does not require all of the language's constructs. Table II shows a reduced version of the grammar that generates the entity declaration section in a VHDL file.

The symbols in bold in the previous production rules are terminal and indicate actual code. Terminal symbols are included in the body of the template to indicate Acceleo the code to generate. The non-terminal symbols in the rules indicate the application of other rules. Similarly, a template implementing a production rule may call other templates that generate different parts of the code.

Table I. A query in Acceleo that validates restriction 6.

```
[query public validateInputPins(anAction: Action): Boolean =
  anAction.input
  ->forAll(i: InputPin |
                          i.validatePinType() and
                          i.validatePinMultiplicity() and
                          i.validatePinUpperBound() and
                          i.isConnected())
/]
```

Table II. Simplified grammar that generates the entity declaration section in a VHDL design file.

```
entity_declaration ::= ENTITY identifier IS
                       port_clause
                       BEGIN
                       END ENTITY identifier;
port_clause ::= PORT (interface_list) ;
interface_list ::= interface_signal_declaration { ; interface_signal_declaration }
interface_signal_declaration ::= identifier_list: mode subtype_indication
```


DOI: 10.1002/cta

Templates employ loops and conditional structures to generate code iteratively or selectively. The template in Table III generates the entity declaration section in a VHDL file; it maps the parameter nodes in an activity directly to ports in the entity declaration.

From the activities in Figure 3 the transformation tool synthesizes structural VHDL code describing a pipelined architecture for KASUMI. Each stage of the architecture corresponds to a pair of round module and key-scheduler module. The architecture requires eight clock cycles to encrypt the first 64-bit plaintext block and then issues one ciphertext block every clock cycle. Figure 4 illustrates simulation results using the standard test vectors provided by the Third Generation Partnership Program [19]. During the first clock cycle (Figure 4(a)), the architecture is fed with the first plaintext block and the first key; during the second and third clock cycles (Figure 4(a) and Figure 4(b)), new data is provided to the architecture. From the eighth clock cycle to the tenth clock cycle (Figure 4(d) and Figure 4(e)), the architecture generates the resulting ciphertext blocks. These results prove that the transformation tool generates correct hardware descriptions in VHDL from activity models; thus, we have a complete design flow from high level descriptions to lower level representations.

The previous example illustrated the synthesis of VHDL code from a description of KASUMI intended to optimize the performance of the resulting digital hardware system. Alternatively, the designer may manipulate the structure of the model of KASUMI to simplify the algorithm and produce area-efficient systems like those described by the authors previously [13, 14]. Thus, the designer is responsible for building models according to a strategy for reaching a specific design goal. The structure of these models shall produce designs optimized for either performance or number of hardware resources. Power consumption may also benefit from such design strategy, especially when the systems produced operate on low frequencies.

Our design infrastructure allows modeling complex algorithms and synthesizing the corresponding VHDL descriptions as long as two conditions are met. First, every model must manipulate bit-blocks and use supported operations. Second, a model may invoke other models or not; if it does, the invoked models must meet these two conditions. As a result, our synthesis technology is able to validate and transform hierarchical models made up an arbitrary number of different models. Also, UML 2 provides modeling constructs that enable behaviors to invoke one another in a hierarchical manner; the only limitation may be the extent as to which the modeling tools ease sharing and reusing modeling projects and diagrams. Therefore, the synthesis tool is applicable to the development of complex systems.

Table III. Template in Acceleo that generates the entity declaration section in a VHDL design file.

```
[template public generateEntityDeclaration(anActivity: Activity)]
entity [anActivity.generateEntityName()/] is
  [anActivity.generatePortClause()/]
begin
end entity [anActivity.generateEntityName()/];
[/template]
[template public generateEntityName(anActivity: Activity)]
  [anActivity.getActivityName().concat('_entity')/]
[/template]
[template public generatePortClause(anActivity: Activity)]
port (
  [anActivity.generatePortList()/]
);
[/template]
[template public generatePortList(anActivity: Activity)]
[for (pn: ActivityParameterNode | anActivity.getParameterNodes()) separator(';\n')]
[pn.getParameterNodeName()/]: [pn.parameter.direction.toString()/][pn.generateSubtypeIndication()/][/for]
[if (anActivity.isSynchronized())]
;
clk: in bit[/if]
[/template]
```


DOI: 10.1002/cta

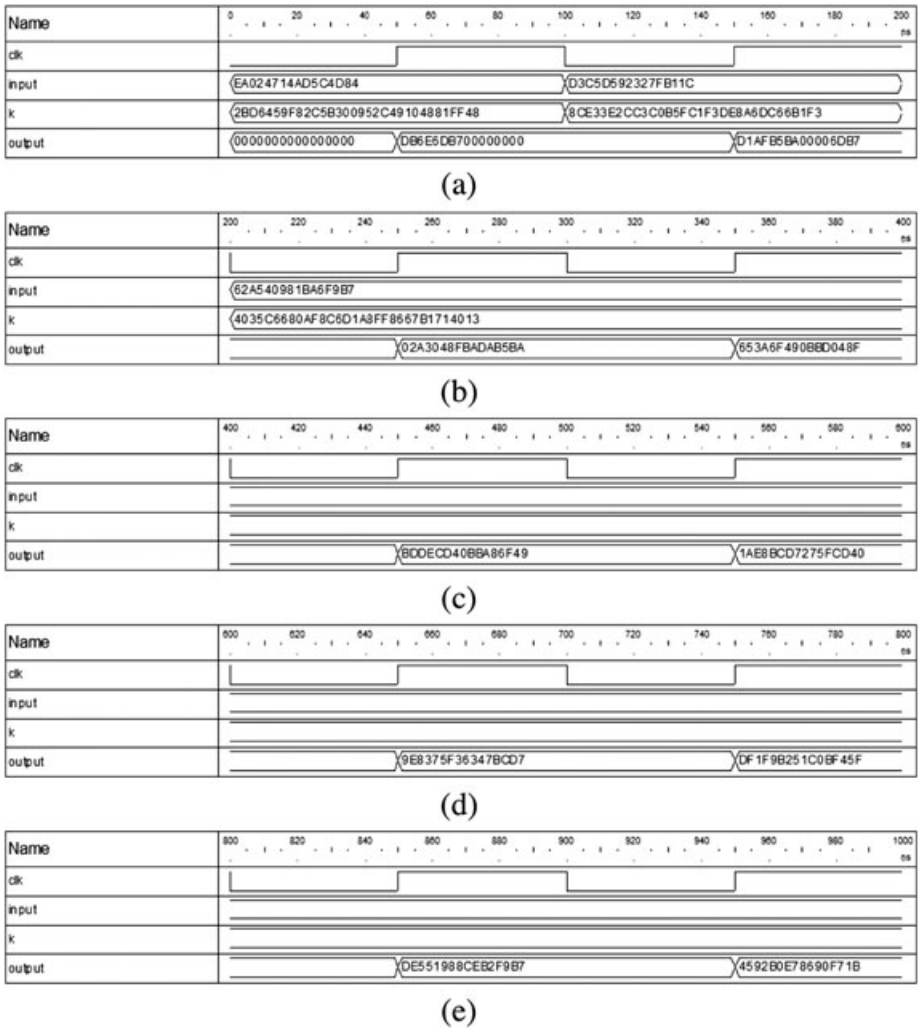


Figure 4. Waveforms for the pipelined architecture for the KASUMI block cipher.

4. CONCLUSIONS

We described a functional prototype of a synthesis technology that transforms activity diagrams in UML 2 to source code in VHDL. This technology intends to alleviate the complexity of digital hardware design by raising the level of abstraction and automating the production of VHDL code. It is expected that the use of this technology also increases the productivity of the designers. A shortcoming of the overall design environment is that the models cannot be executed to validate its results before synthesis. Execution would require adding support for execution to the UML 2 modeling infrastructure, which is not a mature technology at this time. Also, the expressiveness of the prototype is limited at this phase of its development.

The next version of the technology will include new syntax for the modeling elements to make them more comprehensible and readable. The use of UML 2 profiles will enrich the proposed design flow by making the concepts in the application domain recognizable and comprehensible. A complete transformation tool will also generate code including components and intellectual property cores for specific ASICs or FPGAs. Thus, it is possible to develop a family of domain-specific modeling languages for different application domains and a family of transformation tools, each transforming models in a modeling language to an implementation for the corresponding platform.

DOI: 10.1002/cta

ACKNOWLEDGEMENTS

This research was funded by CONACyT, the Mexican council for science and technology through the scholarship number 41722. Thanks to the anonymous reviewers for their valuable comments.

REFERENCES

1. Doménech-Asensi G, Díaz-Madrid JA, Ruiz-Merino R. Synthesis of CMOS Analog Circuit VHDL-AMS Descriptions Using Parameterizable Macromodels. *International Journal of Circuit Theory and Applications*; 2011; DOI: 10.1002/cta.820.
2. Bailey B, Martin G, Piziali A. *ESL Design and Verification. A Prescription for Electronic System-Level Methodology*. Morgan Kaufmann Publishers: San Francisco, 2007.
3. Kent S. Model Driven Engineering. *Lecture Notes in Computer Science* 2002; 2335/2002, pp. 286–298. DOI: 10.1007/3-540-47884-1_16.
4. Atkinson C, Kühne T. Model-Driven Development: A Metamodeling Foundation. *IEEE Software* 2003; **20**(5):36–41. DOI: 10.1109/MS.2003.1231149.
5. Miller J, Mukerji J. Model Driven Architecture. *Object Management Group*, 2001; ormsc/2001-07-01.
6. OMG Specification. OMG Unified Modeling Language (OMG UML) Infrastructure. Version 2.1.2. *Object Management Group*, 2007; formal/2007-11-04.
7. OMG Specification. OMG Unified Modeling Language (OMG UML) Superstructure. Version 2.1.2. *Object Management Group*, 2007; formal/2007-11-02.
8. OMG Specification. Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification. Version 1.1. *Object Management Group*, 2011; formal/2011-01-01.
9. OMG Specification. MOF Model to Text Transformation Language. Version 1.0. *Object Management Group* 2008; formal/2008-01-16.
10. Björklund D, Lilius J. From UML Behavioral Descriptions to Efficient Synthesizable VHDL. *In Proceedings of the 20th IEEE Norchip Conference*, 2002.
11. Coyle FP, Thornton MA. From UML to HDL: A Model Driven Architectural Approach to Hardware/Software Co-design. *In Proceedings of Information Systems*: *New Generations Conference*, 2005.
12. 3GPP Specification. Universal Mobile Telecommunications System (UMTS), Specification of the 3GPP Confidentiality and Integrity Algorithms, Document 2: KASUMI Specification. Version 5.0.0. *3rd Generation Partnership Program*, 2002; 3GPP TS 35.202.
13. Balderas-Contreras T, Cumplido R. *An Efficient Reuse-based Approach to Implement the 3GPP KASUMI Block Cipher*. In Proceedings of the First International Conference on Electrical and Electronics Engineering 2004.
14. Balderas-Contreras T, Cumplido R. High Performance Encryption Cores for 3G Networks. *In Proceedings of the 42nd Design Automation Conference*, 2005.
15. Olivé A. *Conceptual Modeling of Information Systems*. Springer Publishing Company, Incorporated: New York, 2007.
16. Musset J, Juliot E, Lacrampe S. Acceleo User Guide. *Obeo*, 2010.
17. OMG Specification. Object Constraint Language. Version 2.0. *Object Management Group*, 2006; formal/06-05-01.
18. IEEE Standard. VHDL Language Reference Manual. *The Institute of Electrical and Electronics Engineers*, *Inc*. 2002; IEEE Std 1076–2002.
19. 3GPP Specification. Universal Mobile Telecommunications System (UMTS), Specification of the 3GPP Confidentiality and Integrity Algorithms, Document 3: Implementors' Test Data. Version 5.0.0. *3rd Generation Partnership Program*, 2002; 3GPP TS 35.203.