Modeling and Formal Verification of a Commercial Microcontroller for Embedded System Applications

Subhashini Balakrishnan and Sofiène Tahar

Concordia University, ECE Dept., Montreal, Quebec, H3H 1M8 Canada
Email: {subhar, tahar}@ ece.concordia.ca

Abstract

Embedded systems are finding widespread application. A formal model for the underlying hardware (RT level) of the microcontroller (PIC 16C71) in commercial use, along with its Instruction Set Architecture (ISA) using Multiway Decision Graphs (MDG) is proposed. Using the MDG verification tools, we verified if the instructions in the instruction set are implemented correctly in the microcontroller hardware. Models for the flowchart specification and the assembly language implementation of an embedded software, used in a mouse controller application are portrayed. Applying the platform of the ISA verification, the correctness of the embedded software has been verified at a higher level of abstraction. Inconsistencies in the assembly code with respect to the specification, as published in the application notes of the manufacturer, were uncovered through our experiments.

I. Introduction

Interest in hardware/software codesign [2] has been on the rise for the past few years, and this interest has been manifesting itself in the emergence of tools to facilitate the design of entire systems. Recently, attention has been given to the verification of embedded systems using formal methods. In this paper we present a methodology and application of formal verification of a microcontroller, using Abstract State Machines (ASMs) [5] based on Multiway Decision Graphs (MDGs) [5]. We demonstrate our approach on the embedded software used to program the microcontroller, PIC16C71, commercialized by Microchip Technology Inc., [9] for a serial mouse controller application [8].

Some work have been done on the formal verification of the hardware of microprocessors and digital signal processor algorithms. These include theorem prover methodologies [10, 12] or Decision Diagram based approaches [3, 5]. Thiry and Claesen [11] suggested a methodology for modeling the software and hardware of an embedded system using the CTL temporal logic representation, implemented in SMV [7]. They presented a way for checking properties on the embedded software. SMV uses the ROBDD symbolic model checking algorithm to find out whether CTL specifications are satisfied. More recently, in [1] the verification of an assembly code for the Motorola Complex Arithmetic Digital Signal Processor using ACL2 theorem-proving system is described. Unlike model checkers, theorem provers are scalable to large software programs but are not automatic. Our work is motivated by the fact that a symbolic model checker is restricted to representation at the boolean level and a theorem prover to users with a lot of expertise and experience. We illustrate the ability to carry out equivalence checking, in addition to checking properties, using MDGs. We thus pave a way for automatic verification of an embedded system at higher levels of abstraction.

The rest of the paper is structured as follows: In Section II, we introduce Multiway Decision Graphs briefly. In Section III, we describe the microcontroller architecture, its modeling in MDG and our formal verification approach. The mouse controller software application is outlined in Section IV, along with its model and formal verification. Section V concludes the paper.

II. Multiway Decision Graphs

Multiway Decision Graphs (MDGs) have been proposed recently [5] as a solution to the state space explosion problem of ROBDD based verification tools. The MDG tools combine the advantages of representing a circuit at higher abstract levels as is possible in a theorem prover, and of the automation offered by ROBDD based tools. MDGs, a new class of decision graphs, comprises, but is much broader than the class of ROBDDs. It is based on a subset of many-sorted first order logic, augmented with a distinction between *abstract* and *concrete* sorts.

An MDG is a finite, directed acyclic graph (DAG). An internal node of an MDG can be a variable of concrete sort with its edge labels being the individual constants in the enumeration of the sort; or it can be a cross-term (whose function symbol is a cross-operator). An MDG may only have one leaf node denoted as T, which means all paths in the MDG are true formulae. Thus, MDGs essentially represent relations rather than functions. MDGs incorporate variables of abstract type to denote data signals and *uninterpreted function* symbols to denote data operations. MDGs can also represent sets of states. They are thus much more compact than ROBDDs for designs containing datapath and sequential circuits can be verified independently of the width of the datapath.

In MDG-based verification, abstract descriptions of state machines, called *abstract state machines* (ASM) [5] are used to model the systems. This makes it possible to verify a circuit at the register transfer (RT) level without getting bogged down with the details of a gate level implementation. Thereby, the use of ASMs raises the level of abstraction of automated verification methods to approach those of interactive theorem proving methods, without sacrificing automation.

MDGs are used as the underlying representation for a set of hardware verification tools, providing both validity checking and verification based on state-space exploration [5]. The MDG tools package the basic MDG operators and verification procedures. The MDG tools run on a Prolog platform and they accept a Prolog-style HDL, MDG-HDL [13], which allows the use of abstract variables and uninterpreted function symbols. MDG-HDL supports structural and behavioral ASM descriptions, or a mixture of structural and behavioral descriptions. It also comes with a large set of basic components provided in a library. The MDG-HDL description is then compiled into the ASM model in internal MDG data structures. Interested readers are referred to [4, 5, 13] for more details on MDGs and related tools.

III. The Microcontroller Architecture

The hardware device we investigate in this work is the PIC16C71 microcontroller [9], commercialized by Microchip Technology Inc. We give a simple, brief description of the microcontroller architecture and implementation, with their MDG models.

A. MDG Modeling of the RTL Implementation

The PIC16C71 is an 8-bit controller, employing a RISC-like architecture (Figure 1). There are 36 8-bit wide general purpose registers, 15 special function registers and a hardware stack. The hardware stack is 8-level deep and has 36 bytes of RAM. A total of 35 instructions (reduced instruction set) are available, each instruction being 14-bit wide. An instruction cycle consists of eight Q cycles (Q1 to Q8). A fetch cycle begins with the program counter (PC) incrementing in Q1. The instruction is fetched from the program memory and latched into the instruction register in Q5. This instruction is then decoded and executed during the Q6, Q7, and Q8 cycles. Data memory is read during Q6 (operand read) and written during Q8 (destination write).

Using MDG-HDL, we described the RTL netlist implementation of the microcontroller of Figure 1. We adopted a hierarchical description down to the MDG-HDL library of basic components. As the MDG system can handle abstract descriptions, it avoids all the cumbersome procedure of defining each bit of a register. Rather, a register can be viewed as an abstract variable. Thus, an 8-bit general purpose register of the microcontroller can be modeled as a variable of abstract sort *worda8* instead of a concrete sort with enumeration {0, ..., 255}. In MDG-HDL, the system bus can be modeled using the basic component, driver. The register file is described in terms of access functions *read* and *write*, modeled as uninterpreted function symbols. The uninterpreted function symbol, *fetch* is used to model the program memory. The ALU can be viewed as a black box, where ALU functions are expressed using uninterpreted function symbols e.g., *add*, *sub*, *inc*, *or* etc. The Q cycles are modeled using a variable of concrete sort *wordc4*, with enumeration {0, ..., 15}, which can accommodate the 8 cycles.

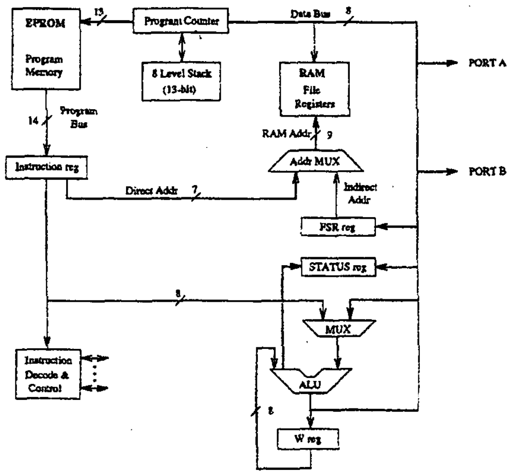

Figure 1. Microcontroller architecture (RT-level)

B. MDG Modeling of the ISA

The instruction set architecture (ISA) consists of byte-oriented, bit-oriented, literal and control operations. The following are the four general formats of instructions:

- instructions acting on two registers
- instructions acting on the working register, *w*
- instructions acting on one bit
- instructions acting on the program counter, *pc*

Using MDGs we can model the assembly instructions as predicates. We illustrate this on the following two instructions: the inclusive-OR instruction (IORLW K) [9] between a literal (*k*) and a working register (*w*), and the decrement instruction (DECF W) [9] of the working register (*w*). The operations (e.g., inclusive-OR, decrement) can be modeled using uninterpreted functions (e.g., *or*, *dec*), applied to the arguments of the instructions (predicates). The instructions mentioned above and their models, written in MDG-HDL, are as follows:

Assembly Instruction: IORLW K

Definitions:

```
var(w, worda8).
var(k, worda8).
function(or,inputs[worda8,worda8],output[worda8]).
```

Instruction:

```
transform(inputs([w,k]), function(or), output(w)).
```

Assembly Instruction: DECF W

Definitions:

```
var(w, worda8).
function(dec,input[worda8], output[worda8]).
```

Instruction:

```
transform(inputs([w]), function(dec), output(w)).
```

C. Formal Verification of the ISA

The instruction set architecture of a microcontroller is the specification of the effect that each instruction is intended to have on the visible state which consists of the visible registers and memory. To verify a microcontroller implementation against its instruction set is to verify that the hardware execution of every instruction has the intended effect.

The control FSM of a conventional microcontroller has a distinguished ready state that is the starting point of instruction execution. In PIC 16C71 the instruction execution is signalled by the starting of the Q1 cycle. To verify we compare the circuit M consisting of the microcontroller with an ideal state machine M' whose state is the visible state of M and where each transition corresponds to the execution of an instruction as specified by the architecture. M' is synchronized with M by a ready signal extracted from M: when $ready = 1$ (when $Q = Q1$) the specified transition takes place, otherwise M' remains in the same state.

Equivalence verification between the RTL implementation (machine M) and the ISA (machine M') is done within the MDG tools through sequential equivalence checking, which ensures if the outputs of the two machines are the same at every clock cycle.

Applying our models, we verified whether each of the instructions in the ISA is executed correctly in the microcontroller RTL hardware. Experiments were conducted on a SUN SPARC ULTRA 1 with 124 MB of main memory. The performance statistics for the equivalence checking are given in [6], including CPU time, memory usage and number of MDG nodes generated.

IV. Embedded System Application

The implementation of a serial mouse using the PIC16C71 [8] is taken as the example for our purpose. We started with the data sheets of PIC16C71 by Microchip Technology Inc. [9].

The major tasks performed by the embedded software for the mouse controller are Button scanning, X and Y motion scanning, and formatting and sending data to the host. The software is composed of three parts:

- Main program
- Subroutine *Byte*
- Subroutine *Bit*

The main program detects any changes in the button status and in the movement counts. There are two subroutines *Byte* and *Bit*. The *Byte* is called in the main program five times to send five bytes of data. The *Byte*, in turn calls the routine *Bit* periodically. The *Bit* counts the number of pulses from the outputs of the photo detectors and determines the direction of movement. In this paper, we concentrate on the specification of the routine *Bit* [8] for our demonstration.

The routine *Bit*, shown in Figure 2, consists of two subroutines, *Bitx* and *Bity*, wherein the *Right Flag* being set indicates a movement to the right and *Up Flag* being set indicates an upward movement. The *XCount* and *YCount* give the speed of the movement in X and Y directions respectively.

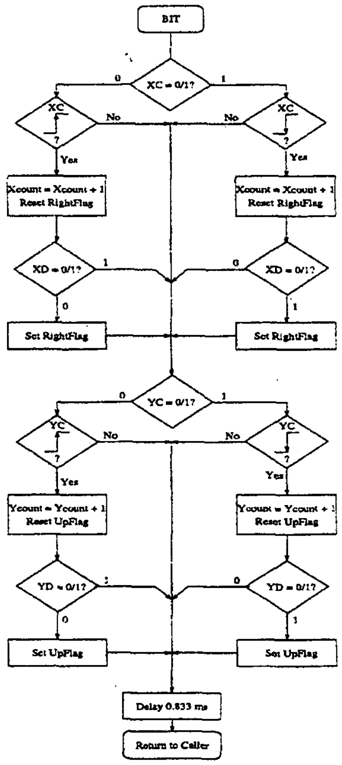

Figure 2. Flowchart specification of *Bit* routine

A. MDG Modeling of the Flowchart Specification

Our main goal is to verify whether the programmed embedded software confirms with the intended behavior. One common way of describing the intended behavior is using algorithmic flowcharts. Hence, we consider the flowchart of the embedded software program as the behavioral specification. We model the flowchart as an ASM (given mainly by a tabular representation of the transition and output relations). In our application, each state in the ASM characterizes the contents of the microcontroller registers. The instructions in an assembly code, or a microcode, or the statements of a specification control the transitions from one state to another. The model uses the abstract sorts *worda8* and *label* for the *XCount* and (branch) target address labels, respectively. We thus do abstract away from defining any concrete bit width for the *XCount* and any enumeration for the address labels. We also use two concrete sorts *bool* and *wordc4*. The former is the boolean sort while the latter designates 4 bit words with enumeration {0, ..., 15} and is used for *pc* values.

```
....
BTFSS RA.b2
GOTO BITC
....
BTFSS RA.b3
....
BSF FLAGB.b3
GOTO BITY
....
```

Notations:

```
BTFSS Ri.bj: test bit j of Ri. skip next inst. if set
GOTO L: jump to the address indicated by label L
BSF Ri.bj: set bit j of register Ri
```

Figure 3. Assembly code portion of *Bit1* of *Bitx*

B. MDG Modeling of the Assembly Implementation

The assembly code implementation is modeled as a set of instructions implementing the control behavior of a routine, which is also represented as an ASM. Instruction variables such as *RA.b2* and *FLAGB.b3* (see Figure 3) are of sort *bool*, *pc* of sort *wordc4*, and *Bitx* and *Bity* of sort *label*. Furthermore, we use the same abstract and concrete functions as in the flowchart specification. This is mandatory when checking equivalence between the specification and the implementation using the MDG system.

Interested readers are referred to [6] for further details on the mouse controller software and its MDG modeling.

C. Formal Verification of the Embedded Software

Equivalence checking between the specification (or behavior) of the embedded software and its assembly implementation is done to ensure that the implemented assembly code reflects its intended behavior. Based on the above results of the ISA verification (Section III-C), we verified the embedded assembly language software of the entire routine *Bit* against its behavioral specification. We also validated our models and verification results by checking key properties of the *Right* and *Up Flags*.

The equivalence checking between the behavioral specification and the embedded software showed an error in the assembly language code, indicated by a counterexample generated by the MDG tools, providing a trace leading to the software error. Accordingly, one of the instructions, BTFSS (highlighted in Figure 3) is to be replaced by BTFSC. This result confirms with the one obtained in [11] using SMV. The erroneous instruction is corrected and the equivalence checking is run successfully. The performance statistics and more details on the embedded software verification can be found in [6].

V. Conclusions

Embedded systems are gaining widespread applications by the fact that they allow for more flexibility and reconfigurability (the degree of which depends on the degree of the partition) and reduced design cycle time. This paper attempts to emphasize the imperative need for verifying the correctness of the target system and its embedded application software. Models are outlined for the hardware and its instruction set architecture of a commercial microcontroller using MDGs. MDG models for expressing the behavioral specification of its embedded software application and its implementation in the assembly code level are also established. The verifications of the PIC 16C71 microcontroller ISA as well as the embedded mouse controller software are demonstrated using the MDG tools. The experiments concluded in few seconds of CPU time. With this work, we have shown the efficiency of the use of abstract data types and uninterpreted functions, as provided by MDGs, to achieve a high level verification in an automated environment.

References

[1] B. Brock and W. Hunt. "Formally Specifying and Mechanically Verifying Programs for the Motorola Complex Arithmetic Processor DSP". Proc. IEEE International Conference on Computer Design (ICCD'97), Austin, Texas, USA, October 1997, pp. 31-36.

[2] K. Buchenrieder, A. Sedlmeier, and C. Vieth. "HW/SW Co-Design with PRAM's using CODES". Proc. Computer Hardware Description Languages and their Applications (CHDL'93). Elsevier Science Publishers B. V., 1993, pp. 65-78.

[3] J. Burch and D. Dill. "Automatic Verification of Pipelined Microprocessor Control". Computer Aided Verification, Lecture Notes in Computer Science 818, springer Verlag, 1994, pp. 68-80.

[4] E. Cerny, F. Corella, M. Langevin, X. Song, S. Tahar, and Z. Zhou. "Automated Verification with Abstract State Machines Using Multiway Decision Graphs". Formal Hardware Verification: Methods and Systems in Comparison, LNCS 1287, State-of-the-Art Survey, Springer Verlag, 1997, pp. 79-113.

[5] F. Corella, Z. Zhou, X. Song, M Langevin, and E. Cerny. "Multiway Decision Graphs for Automated Hardware Verification". Formal Methods in System Design, Vol. 10, No. 1, 1997, pp. 7-46.

[6] S. Balakrishnan and S. Tahar. "On the Formal Verification of Embedded Software Using Multiway Decision Graphs". Technical Report No. 402, Concordia University, Dept. of Electrical and Computer Engineering, December 1997.

[7] K. McMillan. "Symbolic Model Checking". Kluwer Academic Publishers, Boston, Massachusetts, 1993.

[8] Microchip Technology Inc. "Embedded Control Handbook", 1993, pp. 2.121-2.133.

[9] Microchip Technology Inc. "PIC16C71", 1994, pp. 2.328-2.372.

[10] S. Tahar and R. Kumar. "Implementing a Methodology for Formally Verifying RISC Processors in HOL". Higher Order Logic Theorem Proving and its Applications, LNCS 780, Springer Verlag, 1994, pp. 281-294.

[11] O. Thiry and L. Claesen. "A Formal Verification Technique for Embedded Software". Proc. IEEE International Conference on Computer Design (ICCD'96), Austin, Texas, USA, October 1996, pp. 352-357.

[12] P.Windley. "Formal Modeling and Verification of Microprocessors". IEEE Transactions on Computers, Vol. 44, No. 1, 1995, pp. 57-72.

[13] Z. Zhou and N. Boulerice. "MDG Tools (V1.0) User's Manual". Dept. D'IRO, University of Montreal, 1996.