A Formal Verification Technique For Embedded Software.

Olivier Thiry, Luc Claesen
IMEC/K.U.Leuven, Kapeldreef 75, B-3001 Leuven Belgium
phone: +32-16-281203, fax: +32-16-281501, email:claesen@imec.be

Abstract

A method for the verification of embedded software correctness is presented1. A formal model for an actual commercial microprocessor is established. This is done by modeling the instruction set and processor architecture. Embedded software takes the form of the assembly program code to be run on the processor. Specifications are given as CTL temporal logic formulae. The method has been implemented in the SMV model checker and is illustrated by a practical embedded system application: a mouse controller. The inconsistency of the specification and the implementation as an assembly language program as it has been published in the applications book of the manufacturer has been uncovered.

1. Introduction

VLSI technology has enabled that more and more complex systems can be integrated on single chips. Besides dedicated hardware, several systems are currently built up as integrated embedded systems, encompassing a dedicated microprocessor with embedded software together with specific peripheral circuits [13]. This trend is motivated by the fact that the embedded software component allows to postpone specific implementation and functionality decisions until late in the development process. Having the availability of a microprocessor and embedded software, allows for design changes later in the product life cycle without too drastic changes in the circuit structures. Very often only a ROM has to be updated. This has led to the problem of what is commonly known as Hardware/Software CoDesign [10].

The main emphasis in the Hardware/Software CoDesign area has gone into either the partitioning problem [11] to make decisions on which parts of the design go into hardware and which go into software, or in the estimation of timing behavior for the implementation of certain parts in hardware or software. Very little has been achieved yet in the area of verification of embedded systems or mixed hardware/software systems.

There is however a definite need to be able to verify the correct behavior of embedded systems. This can have as aspects, the verification of the correct implementation of systems as mixed hardware/software systems or the verification of certain properties (safety or lifeness properties) of these systems. In cases where systems are used in safety critical applications the thorough verification of the correct behavior is definitely needed. Examples are numerous: car-electronics, aircraft electronics, medical applications. A number of the tasks to be performed (e.g. ABS, automatic suspension etc...) have to satisfy certain safety criteria. The ABS should not start breaking at the wrong occasions.

This paper presents a methodology and application of the formal verification of embedded software [14]. The instruction set of a commercial microcontroller [5] has been modeled. A method for the symbolic verification of software in terms of assembly code [5] on this processor is worked out. The application of a mouse controller [6] with Microsoft compatible RS232 interface has been used as an example. Inconsistencies of the assembly code in [6] and the flow chart specification in [6] has been uncovered, illustrating the usefulness of the approach presented.

2. The CTL Temporal Logic.

For the modeling and verification of the embedded software the CTL temporal logic was chosen. Temporal logic allows to reason with finite state machines that evolve over time. The intention is to model the machine architecture as an instruction interpreter and the assembly code as a finite state machine. CTL is a subset of temporal logic defined by Clarke, Emerson and Sistla [3]. Ken McMillan e.a. [1,4] have introduced symbolic state space traversal techniques that allow to evaluate CTL formulae on finite state machines in an efficient way by means of

[1]Research sponsored in part by the European Union under the ESPRIT 8776 CHARME working group.

symbolic fixed point calculations. This is implemented in the SMV program [12].

Figure 1 represents the CTL formula : "for all paths, there exists a state in the future where the boolean formula g is true (AF g)

In our application, each state is characterized by the contents of the microcontroller registers and the transitions are due to the instructions of the assembly code

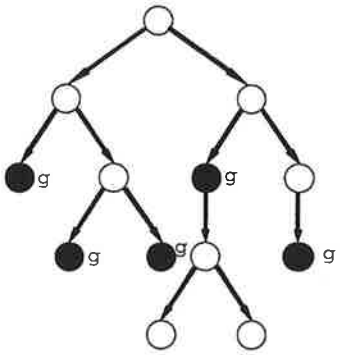

Figure 1 : AF g

3.Modeling of embedded software

The SMV language defined by K. Mc Millan, provides operation at a suitable high level to allow the description of finite states machines. It uses the Oriented Binary Decision Diagram symbolic model checking algorithm to find out whether the CTL specifications are satisfied.

3.1 Description of the microcontroller

We give a description of the microcontroller in order to estimate what characteristics must be modeled : registers, instructions, data path.

This microcontroller [5] is a RISC-like CPU having only 35 instructions. Each instruction takes one clock cycle, except for program branches which takes two cycles. The 1K EPROM memory contains the 14-bit instructions which compose the program.

We also have 36 general purpose registers (8-bit wide) and 15 special function registers (status register, low order 8 bit of the program counter, 8 bits real time clock counter, ...) .

Finally, this microcontroller has an 8-level deep hardware stack: 8 13-bit wide registers.

We will only elaborate a structure including the instruction set (instruction decode and control, ALU), the stack, the program counter, the working register, and some other registers (RAM File registers).

These simplifications have been made in order to focus better on our modeling. Indeed, our goals are simply to verify that our approach is possible but not to make a complete and absolutely general model.

Figure 2 represents the architecture of the microcontroller

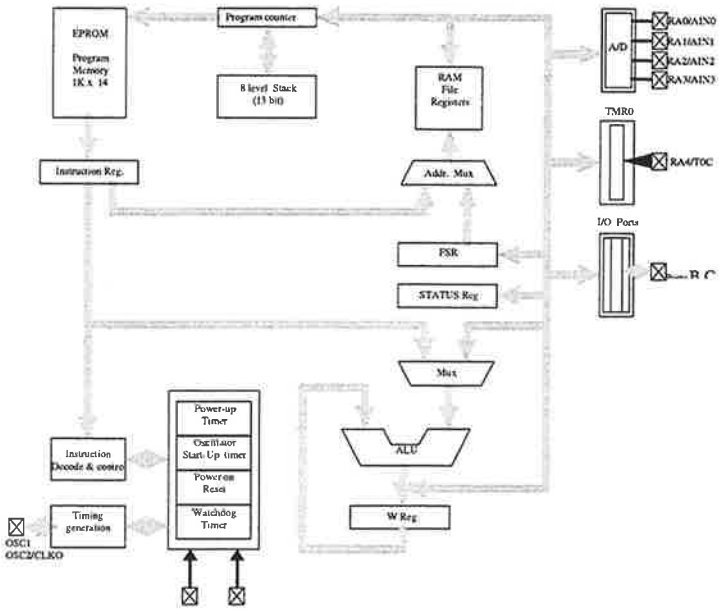

Figure 2 : block diagram of the microcontroller

Mainly, a model is set up for :
the registers
the instructions

3.2 The registers

The physical registers contain a number between 0 and 255. They are declared as a module containing 8 Boolean variables. In the example below, the syntax of the SMV language [12] is used.

ramX represents a 8-bit general purpose register and b0, b1 , ..., b7 the 8 bits of this register.

```
MODULE main
VAR
        ramX:process REGISTER;
        ...
MODULE  REGISTER
VAR
        b0:boolean;
        b1:boolean;
        ...
        b7:boolean;
```

Figure 3 :Description of a general purpose register

3.3 Instructions

We can distinguish four different types of instructions as represented in the following figures :
instructions acting on two registers
instructions acting on the working register
instructions acting on one bit
instructions manipulating the program counter

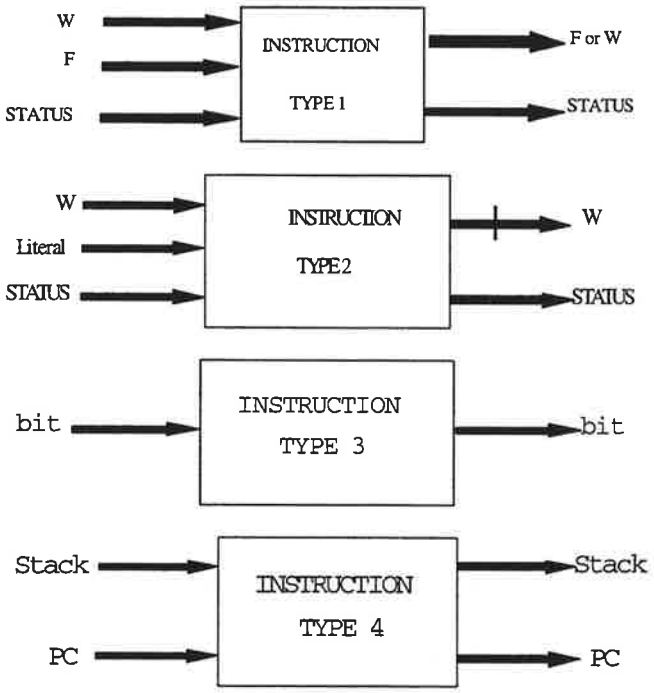

Figure 4 : Representation of the 4 types of instructions

3.4 Suppression of parallelism

SMV is a language which allows several modules to evolve in parallel. For a single processor application, only one instruction is executed at a time. So a way to suppress this parallel capability of SMV is required.

In order to achieve this goal, each instruction, in other words each module, will receive the program counter as parameter and a number which corresponds to the address of the instruction. It is like at the butcher's. Each customer has a number and looks at the counter. When his turn comes, the client acts and modifies the counter.

3.5 Example

In figure 5, the representation of the instruction OR between a literal and the working register (W) is illustrated.

There are two modules :
IOR, an auxiliary module, which computes the ORing operation between two parameters a and b

IORLW, the real instruction which applies IOR to W.bi and Ki (i=0, ...7) where W.bi represents the ith bit of the working register and Ki the ith bit of the literal.

Z represents the bit ZERO of the status register and is set to 0 if the result of the operation is zero.
Pc is the program counter and Inr is the instruction number(see 3.4)

This code simply defines the next value of each bit of the working register in function of the value of the program counter :
If this value is equal to the instruction number the OR operation is done
Otherwise the value of the W register is left unchanged.
The same test is also applied to Z and Pc

```
 MODULE IOR(a,b)
DEFINE
        or:=alb;

MODULE IORLW(Inr, Pc, W, K0, K1, K2, K3, K4, K5, K6, K7, Z)
VAR

        bi0:IOR(W.b0, K0);
        ...
        bi7:IOR(W.b7, K7);

ASSIGN
next(W.b0):=
        case
                Inr=Pc          :bi0.or;
                1               :W.b0;
        esac;
....

next(W.b7):=
        case
                Inr=Pc          :bi7.or;
                1               :W.b7;
        esac;

next(Z):=
        case
                Inr=Pc          :!(bi0.orl . . . | bi7.or);
                1               :Z;
        esac;
```

```
next(Pc):=
case
        Inr=Pc          :Pc+1;
        1               ;Pc;
esac;
```

Figure 5 : description in the SMV language of the instruction IORLW

4. Application

4.1 Introduction.

The model is illustred by a simple mouse controller [6] which is mainly composed of five functional blocks :

- Microcontroller
- Button detection
- Motion detection
- RS232 signal generation
- 5V DC power supply unit .

For this example, we suppose we have three push buttons. When a switch changes or any X or Y movement is detected, a message is formatted and sent to the host.

4.2 Software details[6]

The software has to :

- scan buttons, X and Y motion
- format data
- send this serial data to the host.

To achieve these three goals, the program is composed of three parts :

the main part tests any changes in the button status and in the movement counts. Then, it calls five times the routine Byte.

Byte sends data and periodically calls the routine bit

Bit counts X and Y pulscs.

In this paper we will restrain us to one specification of the routine bit.

4.3 Specification

From the flow chart [6] some specifications can be deduced on what this program is supposed to carry out. We restrict ourselves to a partial description of the specifications. Indeed, it is not necessary to describe the program specifications totally to insure that our approach is valid.

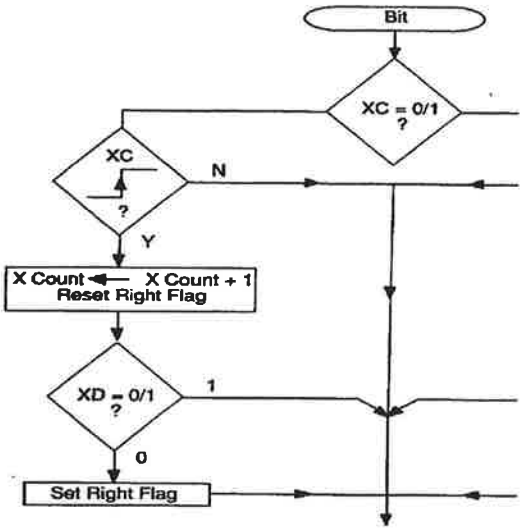

Figure 6: partial representation of the flow chart concerning the bit routine

Figure 6 is a part of the flow chart of the routine bit.

We can deduce from this that if XDATA =0 is read during a rising edge of the clock (Ra.b2=1 and CSTAT.b2=0) then the right flag (FLAGB.b3) is set to 1 in all cases in the future.

We obtain in CTL :

AG (RA.b2=1 & CSTAT.b2=0 & RA.b3 =0 -> AF FLAGb.b3 =1)

where (AG p) means for all paths, p is true in each state and (AF p) means for all paths, p is true for one state in the future.

5. Experimental results

5.1 Our specific example

In the figure 7, the piece of code corresponding to the flow chart is shown.

NOTATIONS :
BIT0 and BITY are the addresses of sub routines
BTFSC (a, b,c) tests bit c and skip the next instruction if it is cleared
BTFSS (a,b,c) tests bit c and skip the next instruction if it is set
INCF (a,b,c,d,e,f) increments register d and place the result in d if e=1 otherwise in c
BCF (a,b,c) clears bit c
BSF(a,b,c) sets bit c
GOTO (a,b,c) go to address c

```
INST0   :process BTFSS      (0, Pc, RA.b2);
INST1   :process GOTO       (1, Pc, BIT0);
INST2   :process BTFSC      (2, Pc, CSTAT.b2);
INST3   :process GOTO       (3, Pc, BITY);
INST4   :process INCF       (4, Pc, W, XCOUNT, 1, STATUS.b2);
```

```
INST5    :process BCF      (5, Pc, FLAGB.b3);
INST6    :process BTFSS    (6, Pc, RA.b3);
INST7    :process GOTO     (7, Pc, BITY);
INST8    :process BSF      (8, Pc, FLAGB.b3);
INST9    :process GOTO     (9, Pc, BITY);

rising edge =instructions 0 to 3

XCOUNT+1
clear right flag
If XDATA is set then set right flag
```

Figure 7 : First instructions of the bit routine

We applied our model to the complete routine bit (40 instructions). It took 23 seconds on a 486DX33 with 16 MB RAM to verify the specification described in 4.3 .The listing was found incorrect and the system produces a counter-example which is a succession of states (registers) and transitions (instructions). Indeed, it is easy to see that instruction 6 should be a BTFSC instead of a BTFSS.

5.2 Discussion

The method explained in this paper has shown that it is possible to model the instruction execution and verification of instruction properties based on temporal logic CTL and implement this in the SMV system. The complexity to model the whole processor and the whole embedded program is still too high for execution with SMV. This is mainly caused by the fact that a single state transition relation is being set up in SMV that needs to be represented by a canonical OBDD characteristic function. The modeling and verification of larger embedded programs will require new symbolic analysis techniques able to cope with the inherent complexities.

6. Conclusions

Embedded systems are becoming more and more important as a means to integrate versatile and cheap systems. The correct behavior of embedded software and hardware is crucial in most applications. This paper describes a method for the verification of embedded software on a commercial embedded processor. It has been applied to the verification of an example software implementation at the level of assembly code. Inconsistencies in the implementation of a published application have been shown. The application of traditional techniques for symbolic state-space traversal is limited due to complexity problems in the symbolic state space representation and manipulation. Therefore future research is oriented towards applying other techniques such as SFG-Tracing [7,8] that have proven to be able to cope with practical problems of complexities of 230,000 transistor chips.

Acknowledgments

This work has been supported in part under the ESPRIT CHARME working group 8776.

References

[1] K.L. McMillan, "Symbolic model checking", Carnegie Mellon University, Kluwer Academic Publishers, 1993.
[2] R.E. Bryant, "Graph-Based Algorithm for Boolean Function Manipulation", IEEE Transactions on Computers, vol. C-35, No. 8, 1986.
[3] E.M. Clarke, E.A. Emerson, A.P. Sistla, "Automatic Verification of Finite-State Concurrent systems using Temporal Logic Specifications", ACM Transactions on Programming Languages and Systems, Vol. 8, No. 2, April 1986, page 244-263.
[4] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, J. Hwang, "Symbolic Model Checking: 1020 and beyond", in Proc. of the Fifth Annual Symposium on Logic in Computer Science, June 1990.
[5] Microchip Technology Incorporated, "PIC16C71", 1994, page 2.328-2.372.
[6] Microchip Technology Incorporated, "Embedded Control Handbook", 1993, page 2.121-2.133.
[7] L. Claesen, F. Proesmans, E. Verlind, H. De Man, "SFG-Tracing: a methodology for the automatic verification of MOS transistor level implementations from high-level behavioral specifications", Proceedings 1991 International Workshop on Formal Methods in VLSI Design, ed. P.A. Subrahmanyam, ACM-SIGDA, Miami, Jan 9-11, 1991.
[8] L. Claesen, M. Genoe, "Multi-Level Formal Verification of High Level Synthesis: a Reality", Proceedings SASIMI'95, August 25-26, 1995, Nara Japan, pp. 106-113.
[9] H. De Man, J. Rabaey, P. Six, L. Claesen, "Cathedral-II: a silicon compiler for digital signal processing", IEEE Design and Test of Computers, December 1986, Vol. 3, No. 6, pp.73-85.
[10] K. Buchenrieder, A. Sedlmeier, C. Vieth, "HW/SW Co-Design with PRAM's using CODES", in Proceedings CHDL'93, Computer Hardware Description Languages and their Applications (A-32), ed. D. Agnew, L. Claesen, R. Camposano, Elsevier Science Publishers B.V., 1993, pp. 65-78.

[11] J. Henkel, Th Benner, R. Ernst,W. Ye , N. Serafimov and G. Glawe,"COSYMA : A software-oriented approach to hardware/software Codesign", The journal of Computer and Software Engineering, Vol.2, No.3, pp. 293-314, 1994.

[12] K.L. Mc Millan, "The SMV system DRAFT", Carnegie-Mellon University, February 2, 1992.
[13] L. Claesen, D. Beullens, R. Martens, R. Mertens, S. De Schrijver, W. De Jong, "SmartPen : An Application of Integrated Microsystem and Embedded Hardware/Software CoDesign", proceedings ED&TC, User Forum, Paris 11-14 March 1996, pp201-205
[14] O. Thiry, "Symbolic Verification Technique of Embedded Software" Masters Thesis, K.U. Leuven (Belgium), 1994-1995