Using a WCET Analysis Tool in Real-Time Systems Education

Samuel Petersson*♮, Andreas Ermedahl*♮, Anders Pettersson*, Daniel Sundmark*, and Niklas Holsti†
*Dept. of Computer Science and Electronics
Mälardalen University, S-72123 Västerås, Sweden
spn99007@student.mdh.se, andreas.ermedahl@mdh.se
anders.petterson@mdh.se, daniel.sundmark@mdh.se
†Tidorum Ltd
Tiirasaarentie 32
FI-00200 Helsinki, Finland
niklas.holsti@tidorum.fi

Abstract

To reach a more widespread use, WCET analysis tools need to be a standard part in the education of embedded systems developers. Many real-time courses in academia use Lego Mindstorms, an off-the-shelf kit of Lego bricks for building and controlling small prototype robots. We describe work on porting the Bound-T WCET analysis tool to the Lego Mindstorms microprocessor; the Renesas H8/3292. We believe that this work will make students, and indirectly the industry of tomorrow, aware of the benefits of WCET analysis tools.

We also present the real-time laboratory framework in which this WCET analysis tool will be used. The framework has been developed with schedulability and timing predictability in mind, and is already used in a number of real-time courses given at Mälardalen University in Sweden. The developed WCET tool and the real-time laboratory framework will be freely available for academic use.

1 Introduction

Today, tools for static *Worst-Case Execution Time* (WCET) analysis, such as Bound-T [4] and aiT [1], are starting to be used in embedded system development and timing verification [9, 10, 14, 15, 18]. We believe that such tools have a potential to be part of the embedded real-time developer's tool chest, in the same way as profilers, hardware emulators, compilers, and source-code debuggers already are today. By providing easier verification of timing behavior they should provide improvements in product quality and safety, as well as reduced development time.

Unfortunately, too few embedded system developers are yet aware of WCET analysis tools and the functionality they offer. This article describes an attempt to improve this situation. We are currently porting an existing WCET analysis tool, Bound-T, to the Renesas H8/3292 microprocessor. This microprocessor is used in Lego Mindstorms [11], an off-the-shelf kit of Lego bricks for building and controlling small prototype robots. This kit is used in many real-time courses in academia. Thereby static WCET analysis could be regularly used in the education of the embedded system developers of tomorrow.

However, for achieving system predictability and to make best use of calculated WCET estimates, the complete system needs to be developed with timing predictability in mind. To make students aware of this fact, the developed WCET analysis tool will be used in a real-time laboratory framework targeting such system predictability. This framework is already used in a number of real-time courses given at Mälardalen University in Sweden. It consists of a small real-time micro-kernel and an operating system called Asterix [16], a configuration tool called Obelix, and a GNU GCC cross-compiler for the H8/3292.

Both the WCET analysis tool and the real-time laboratory framework will be freely available for academic use. This should provide a valuable foundation for teaching students how to construct better and safer real-time systems.

2 WCET Analysis for Mindstorms

Lego Mindstorms is an off-the-shelf kit of Lego bricks for building and controlling small prototype robots. The simplicity in the design of Lego Mindstorms makes it suitable for educational purposes in ages from 12 years and up. Out of the box, the Mindstorms kit is not considered to be a platform for real-time systems. However, the set of Lego bricks includes sensors and actuators such as pressure sensors, a light sensor and small motors, i.e., the fundamental necessities for real-time systems. Also, the sparse hardware resources and few interfaces for hardware access places the Mindstorms construction in the embedded system category.

The RCX, illustrated in Figure 1, is the processing unit

♮ This research has been supported by the Advanced Software Technology Center (ASTEC) in Uppsala [2]. ASTEC is a Vinnova (Swedish Agency for Innovation Systems) initiative [19].

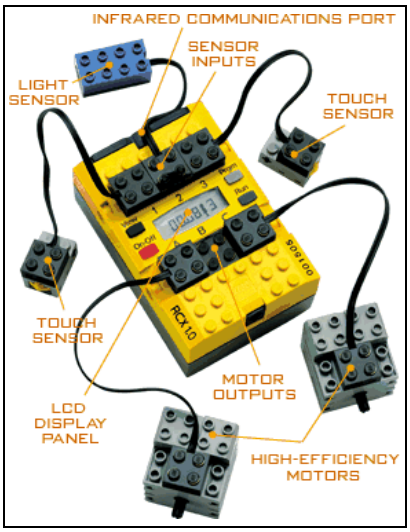


Figure 1. The Lego Mindstorms RCX unit

of Lego Mindstorms. The RCX is based on the single-chip H8/3292 [8], a RISC microcomputer running at 16 MHz. It features a H8/300 CPU core and a complement of on-chip supporting modules. The H8/3292 has 16 kBytes of read-only memory (ROM) and 512 bytes of on-chip random-access memory (RAM) and an additional 16 kBytes of external RAM in a separate circuit. The ROM includes several functions for reading of sensors, controlling the motors, display segments and numbers on a LCD-display. Located on-chip are one 16-bit timer, two 8-bit timers, a watchdog-timer, a serial communication interface and an 8-channel 10-bit analog-digital converter. The RCX also contains an IR-transceiver, useful for downloading programs and for communicating with other RCX units.

2.1 H8/300 Hardware Timing

Instructions in the H8/300 architecture generally have a fixed execution time, the exception being the `EEPMOV` instruction that copies a block of data in memory. There is no cache (at least not on-chip) and no visible pipeline. The fastest instructions take two clock cycles; for example an `ADD.B` or `ADD.W` executed from the on-chip memory using register operands. Complex instructions that access external memory may have execution times of 20 or more cycles, depending on the length of the instruction, the addressing mode, the data width and the memory areas that are accessed. The time does not depend on execution history. Two instructions, `MOVFPE` and `MOVTPE`, synchronize with the ”peripheral” clock and have somewhat variable execution time.

The simple instruction timing means that the high-level flow-path analysis becomes the main problem in WCET analysis for Lego Mindstorms. We chose the Bound-T WCET tool [4] as the basis for our work because its low-level analysis is sufficient for the H8/300, it has a fairly powerful high-level analysis, and its modular structure let us divide the porting work between the WCET analysis group at Mälardalen University and the company behind Bound-T, Tidorum Ltd [4].

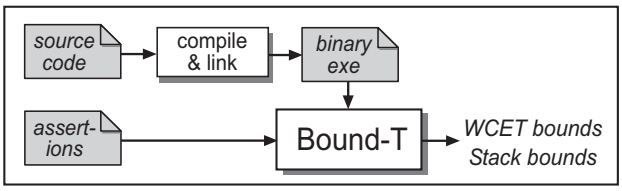


Figure 2. The Bound-T WCET analysis tool

2.2 The Bound-T WCET Analysis Tool

Bound-T, see Figure 2, performs WCET analysis from machine code (binary, linked executables). To find loop bounds Bound-T models the computations and branch conditions with Presburger arithmetic. Bound-T examines the model to find loop-counter variables and computes a bound on loop iterations from the counter’s initial value, its final value implied by the exit condition, and its change (step) in the loop body.

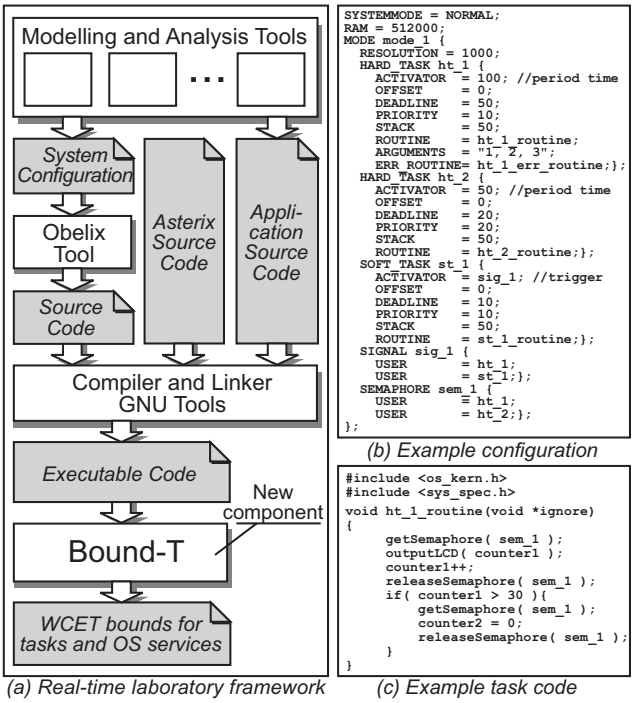
Loop bounds can be context-sensitive, i.e., dependent on the call-path. The Presburger model is used also to resolve dynamic jumps, for example from switch/case statements, and to compute stack usage bounds. The worst-case path is found with implicit path enumeration (IPET).

Bound-T is implemented as a single Ada program with a strict division into modules specific to the target processor (e.g., instruction decoding) and modules independent of the target processor (e.g., analysis of loop bounds). For Presburger analysis Bound-T uses the Omega Calculator [13]. For IPET the lp_solve program [3] is used. Control-flow and call graphs can be emitted in DOT form [6]. The automatic loop analysis can be supported or replaced by user assertions in a separate input file (not as source annotations).

Target processors supported by Bound-T include Intel 8051 [9], SPARC V7 (in its ERC32 implementation) and Analog Devices 21020 DSP [10]. Ports to ATMEL AVR and ARM7 are under way.

2.3 Porting Bound-T to the H8/300

Bound-T is based on an internal model of the target program as a set of control-flow graphs (one for each subprogram) connected into a call-graph. The flow-graph nodes have attributes for the execution time and the arithmetic (computational) effect of the node. The structure of the program model is independent of the target processor but the model is parameterized by target-specific types and operations that are defined in target-specific Ada packages.



(a) Real-time laboratory framework

```
SYSTEMMODE = NORMAL;
RAM = 512000;
MODE mode_1 {
  RESOLUTION = 1000;
  HARD_TASK ht_1 {
    ACTIVATOR  = 100; //period time
    OFFSET     = 0;
    DEADLINE   = 50;
    PRIORITY   = 10;
    STACK      = 50;
    ROUTINE    = ht_1_routine;
    ARGUMENTS  = "1, 2, 3";
    ERR_ROUTINE= ht_1_err_routine;};
  HARD_TASK ht_2 {
    ACTIVATOR  = 50; //period time
    OFFSET     = 0;
    DEADLINE   = 20;
    PRIORITY   = 20;
    STACK      = 50;
    ROUTINE    = ht_2_routine;};
  SOFT_TASK st_1 {
    ACTIVATOR  = sig_1; //trigger
    OFFSET     = 0;
    DEADLINE   = 10;
    PRIORITY   = 10;
    STACK      = 50;
    ROUTINE    = st_1_routine;};
  SIGNAL sig_1 {
    USER       = ht_1;
    USER       = st_1;};
  SEMAPHORE sem_1 {
    USER       = ht_1;
    USER       = ht_2;};
};
```

(b) Example configuration

```
#include <os_kern.h>
#include <sys_spec.h>
void ht_1_routine(void *ignore)
{
     getSemaphore( sem_1 );
     outputLCD( counter1 );
     counter1++;
     releaseSemaphore( sem_1 );
     if( counter1 > 30 ){
         getSemaphore( sem_1 );
         counter2 = 0;
         releaseSemaphore( sem_1 );
     }
}
```

(c) Example task code

Figure 3. The real-time laboratory framework

To port Bound-T to a new target processor, one must implement these target-specific types and operations. The main operation is the one that decodes instructions and inserts them in the program model. This operation is given the address of an instruction and must ”fetch” the binary instruction from the program’s memory image (a COFF file from GCC in our case), decode the instruction to find out the instruction type and the operands, and call Bound-T operations that create new nodes and edges in the control-flow graph.

The porting of the H8/300 to Bound-T took about five months and was performed by the first author, Samuel Petersson, as an MSc project in his computer science studies [12]. The instruction decoding process was divided into two steps: first from the binary instruction to an ”abstract” instruction, and then from the abstract instruction to the Bound-T model. The abstract instruction is a model of the H8/300 architecture built from Ada types. This model depends only on the H8/300, not on Bound-T.

The decoding from binary instructions to abstract instructions included a lot of processor manual reading. It was a considerable job because there are 57 different instructions which can execute in eight different addressing modes. Furthermore, no instructions allow all the addressing modes.

The next step was the conversion of abstract instructions to the Bound-T model. This included creation of flow-graph nodes with timing information and arithmetic effects of the included instructions. The decoding process expands the (abstract) `EEPMOV` instruction into three flow-graph nodes that model the block-copy loop. Bound-T’s usual loop-bound analysis applies here. For the `MOVFPE` and `MOVTPE` instructions we assume the worst-case execution time.

The first version of Bound-T for the H8/300 supports the H8/3297 chip series (which includes the 3292). Tidorum plans to support other H8/300 chips and perhaps other members of the H8 family such as the 32-bit H8/300H processor.

3 Mindstorms in RT Systems Education

The Bound-T tool will be used in a real-time laboratory framework, see Figure 3(a), developed at Mälardalen University. The framework replaces the software architecture and programming environment that are delivered together with the Lego Mindstorms kit. It consists of a small real-time micro-kernel and operating system called Asterix [16], a configuration tool called Obelix, and a GNU GCC cross-compiler for the H8/3292.

3.1 Asterix - the Real-Time Kernel

The Asterix real-time kernel handles execution strategies ranging from strictly statically scheduled systems via fixed priority scheduled systems to event-triggered systems, or any combination of these. To fulfill the needs for embedded systems we have minimized the kernel and the application memory footprints.

A built-in monitoring function facilitates the use of state-of-the-art testing and debugging tools like deterministic testing, replay debugging, and visualization [17]. The kernel also provides on-line facilities for measuring execution times (via testing). For every system reconfiguration the kernel must be recompiled, leading to an efficient usage of memory and other limited resources. Task properties (e.g., deadline, priority, offset) are defined outside the source code in an Obelix configuration file.

3.2 Obelix - the System Configuration Tool

The Obelix system configuration tool allows static off-line definition of system resource demands. The demands are set in a configuration file separated from the source code, clearly separating the system functionality from the system configuration and requirements. Furthermore, Obelix allows cleaner source code in the sense that no special tags and system calls are needed for initialisation and system set up. This gives the possibility of moving the code to the target development environment after testing without modifications.

The configuration files include descriptions of all necessary resource requirements as well as configuration information for e.g., the task schedule, synchronization of tasks, inter-task communication and inter-node communication.

In addition, the configuration files contain all necessary information of task attributes and time resolution. Examples of a configuration file and implemented task functionality are depicted in Figure 3(b) and Figure 3(c) respectively.

3.3 The Real-Time System Student Assignments

On a yearly basis, the laboratory framework is used in two real-time courses (a regular course and a distance course) given at Mälardalen University. For each course, two student assignments are given: a preparatory assignment and a robot project. Because of the straightforward programming (basic C-programming) the preparatory assignments are easily done in a few hours, giving the programming knowledge required for the subsequent robot project. Since the start approximately 500 students have performed 200 robot projects using Lego Mindstorms and Asterix.

Due to the separation of functionality and configuration in the Asterix framework, the students may implement their application in a late stage, and focus more on the theoretical aspects of designing a robust real-time system (e.g., timing analysis, scheduling, etc.). Furthermore, the simplicity of configuring and programming leaves room for proper software design. However, students have previously experienced difficulties to estimate proper execution times by measurements. By instead using Bound-T for this purpose, we believe that the students will be able to derive more accurate timing bounds.

In order to perform response-time analysis and to derive overall system timing guarantees the students need WCET bounds both for tasks and for OS services. To simplify the assignment, WCET bounds for all OS calls will be derived beforehand, and presented in an off-line table. However, the students will be required to use Bound-T to derive WCET bounds for their own robot application task code.

4 Conclusions, Related and Future Work

We have described work on porting the Bound-T WCET analysis tool to the Lego Mindstorms and the H8/3292 microprocessor. The tool will initially be used in assignments in real-time systems courses given at Mälardalen University. This should allow students to get familiar with WCET analysis and should provide valuable feedback on the functionality required for WCET tools to be applicable in real-time system development. The developed WCET tool and the real-time laboratory framework will be freely available for academia and under license for industry.

An alternative OS for Lego Mindstorms, and the developed WCET analysis tool, is BrickOS from Sourceforge [5]. BrickOS supports preemptive multitasking, dynamic memory management, POSIX semaphores, as well as native to display, buttons, IR communication, motors and sensors. However, compared to Asterix, BrickOS is not a hard real-time OS, making it more difficult to provide overall system timing guarantees.

The H8/300 has previously been ported to the Heptane WCET tool [7] and the BrickOS. However, we have not seen any reports on using the tool in education.

Future work includes a systematic validation of the developed Bound-T H8/300 timing model using measurement tools such as oscilloscopes and logic analyzers. This will minimize the possibility of implementation faults and verify that the timing given in the H8/300 processor manual [8] actually corresponds to the real hardware timing.

References

[1] AbsInt company homepage, 2005. `www.absint.com`.

[2] ASTEC homepage, 2005. `www.astec.uu.se`.

[3] M. Berkelaar. *lp_solve: (Mixed Integer) Linear Programming Problem Solver*, 2004. `ftp://ftp.es.ele.tue.nl/pub/lp_solve`.

[4] Bound-T tool homepage, 2005. `www.tidorum.fi/bound-t/`.

[5] BrickOS homepage, 2005. `brickos.sourceforge.net`.

[6] Homepage for the Graphviz tool, Aug 1997. `www.graphviz.org`.

[7] Homepage for the Heptane WCET analysis tool, 2005. `www.irisa.fr/aces/work/heptane-demo/heptane.html`.

[8] Hitachi. Hitachi Single-Chip Microcomputer H8/3297 series. *Hardware manual, 3rd edition*, 2000.

[9] N. Holsti, T. Långbacka, and S. Saarinen. Using a Worst-Case Execution-Time Tool for Real-Time Verification of the DEBIE software. In *Proc. of the DASIA 2000 Conference (Data Systems in Aerospace 2000, ESA SP-457)*, Sep 2000.

[10] N. Holsti, T. Långbacka, and S. Saarinen. Worst-Case Execution-Time Analysis for Digital Signal Processors. In *Proc. of the EUSIPCO 2000 Conference (X European Signal Processing Conference)*, Sep 2000.

[11] Lego Mindstorms homepage, 2005. `www.legomindstorms.com`.

[12] S. Petersson. Porting the Bound-T WCET tool to Lego Mindstorms and the Asterix RTOS. Master's thesis, Mälardalens University, Västerås, Sweden, May 2005.

[13] William Pugh. The Omega test: a Fast and Practical Integer Programming Algorithm for Dependence Analysis. In *Supercomputing*, pages 4–13, 1991.

[14] S. Byhlin, A. Ermedahl, J. Gustafsson, B. Lisper. Applying Static WCET Analysis to Automotive Communication Software. In *Proc. 17^{th} Euromicro Conference of Real-Time Systems, (ECRTS'05)*, July 2005.

[15] D. Sandell, A. Ermedahl, J. Gustafsson, and B. Lisper. Static Timing Analysis of Real-Time Operating System Code. In *Proc. 1^{st} International Symposium on Leveraging Applications of Formal Methods (ISOLA'04)*, Oct 2004.

[16] H. Thane, A. Pettersson, and D. Sundmark. The Asterix Real-Time Kernel. In *Proc. 13^{th} Euromicro Conference of Real-Time Systems, (ECRTS'01)*, June 2001.

[17] Henrik Thane. *Monitoring, Testing and Debugging of Distributed Real-Time Systems*. PhD thesis, Royal Institute of Technology, Stockholm, May 2000.

[18] S. Thesing. *Safe and Precise WCET Determination by Abstract Interpretation of Pipeline Models*. PhD thesis, Saarland University, 2004.

[19] Vinnova homepage, 2005. `www.vinnova.se`.