Using a WCET Analysis Tool in Real-Time Systems Education

Samuel Petersson*, Andreas Ermedahl*, Anders Pettersson*, Daniel Sundmark*, and Niklas Holsti#

*Mälardalen Real-Time Research Center (MRTC)
#Tidorum Ltd

Status of WCET analysis

ADC analysis is now mature enough to be used in real industrial settings

Examples:

- Avionics software
- Software controlling in-vehicle communication networks
- Real-time operating system code
- Space applications

Timing analysis research has developed into companies

- Tidorum (static analysis)
- AbsInt (static analysis)
- Rapita Systems (measurement/static analysis)
**WCET analysis in practice**

🌟 WCET analysis tools have a potential to be a standard part of the embedded system developer’s tool chest

🌟 The problem:
Too few developers are yet aware of WCET analysis tools

**WCET analysis in education**

🌟 Embedded system developers need to be educated on the benefits of WCET analysis tools

🌟 The solution: Make WCET analysis tools a standard part of the education of these developers
Lego Mindstorms

* An off-the-shelf kit of Lego bricks for building and controlling Lego robots
* Including the fundamental necessities of an embedded real-time system:
  - The RCX unit – including a programmable Renesas H8/3292 microprocessor
  - Sensors & actuators – motors, touch- and light sensors, ...
  - Limited I/O – LCD display, IR-transceiver, ...
* Used in many real-time courses in academia!

The H8/3292 microprocessor

* Features a H8/300 CPU core
  - Single-chip RISC
  - Runs at 16 MHz
* 57 instructions
* 8 addressing modes
* 64 kB address space
  - 16 kB of ROM – containing code for reading sensors, controlling motors, etc
  - 512 bytes of on-chip RAM
  - 16 kB external RAM
* 16 8-bit registers or 8 16-bit registers
* No cache or pipeline
The Bound-T tool

★ A commercially available WCET tool
  ➢ Provided by Tidorum Ltd, (Niklas Holsti)
★ Supported targets:
  Intel 8051, Sparc V7, and Analog Devices 21020 DSP
  ➢ Ports to Atmel AVR and ARM7 under way
★ Works directly upon the binary executable

Some Bound-T details

★ The Bound-T tool is written in Ada
★ Clear division into general- and target-specific modules
★ To port Bound-T to a new processor only target-specific modules need to be added
More Bound-T details

- Decodes instructions, construct CFGs, and call-graph directly from the executable
- Presburger analysis is used to find loop bounds, resolve dynamic jumps and to compute stack usage bounds
  - Requires description of arithmetical effects of decoded instructions
- IPET is used to calculate the WCET bound(s)
  - Requires timing on the nodes and/or edges in the CFGs
  - One WCET per function and/or a WCET for the whole program

Porting Bound-T to H8/300

- Work performed by one MSc student (Samuel Petersson)
  - Supervised by Ermedahl, Holsti and Lisper
  - Took about 5 months
- Four main steps performed:
  1. Decode executable into (abstract) instructions
  2. Derive timing for instructions
  3. Give arithmetical effect of instructions
  4. Add representation of decoded instructions to the Bound-T program model (CFGs + CG)
Step 1: Binary decoding

- Input is one or two 16-bit binary words
  - Provided by Bound-T's COFF reader
- Instruction type, addressing mode, and operands needs to be identified
  - Example: 7903 0006 ⇒ mov.w #6, r3
- Considerable job, since 57 instructions with eight different addressing modes
  - No instruction allow all addressing modes

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Mem.</th>
<th>Operands</th>
<th>Instruction code</th>
<th>No. of states</th>
</tr>
</thead>
<tbody>
<tr>
<td>Immediate</td>
<td>MOV.W</td>
<td>W16, R3</td>
<td>0 0 0 0 0 0 0 0</td>
<td>1MB</td>
</tr>
<tr>
<td>Register indirect</td>
<td>MOV.W</td>
<td>QRS, Rd</td>
<td>6 9 0 0 0 0 0 0</td>
<td>4</td>
</tr>
<tr>
<td>Register indirect, with displacement</td>
<td>MOV.W</td>
<td>W16, R3</td>
<td>6 5 0 0 0 0 0 0</td>
<td>6</td>
</tr>
<tr>
<td>Register indirect, with post-increment</td>
<td>MOV.W</td>
<td>QRS, Rd</td>
<td>6 6 0 0 0 0 0 0</td>
<td>6</td>
</tr>
<tr>
<td>Absolute address</td>
<td>MOV.W</td>
<td>W16, R3</td>
<td>6 6 0 0 0 0 0 0</td>
<td>6</td>
</tr>
</tbody>
</table>

Step 2: Instruction timing

- Processor manual gives time in execution states
  - An execution state ≈ clock cycle = 62.5 µs
  - In between 2 to 20 executions states per instruction
  - No caches or pipelines
- The actual execution time depends on:
  - Instruction length
  - Addressing mode
  - Data width
  - Memory areas accessed
- Most required information can be fetched from the processor manual
  - Some depends on the run-time state
- Result as timing on entities (nodes an edges) in Bound-T’s CFGs
  - Used in IPET calculation
Troublesome instructions

- **EEPMOV = MOVe data to EEPROM**
  - Moves block of data in R5 to R6, length in R4L
  - Execution time of instruction depends on value in R4L
- **MOVFPE and MOVTPE**
  - Synchronizes with the peripheral clock
  - The peripheral clock has a variable access time
  - Worst access time assumed (usually an overestimation)
- **Memory configuration change during run-time**
  - Dynamically changed by setting special mode pins
  - Might have huge effect on the timing of instructions
  - Assumed that the programmer does not use this feature

Step 3: Arithmetical effect

- **Description on how executed instructions update different hardware resources**
  - Such as registers and memory locations
  - The "semantics" of the instructions
- **Example: move.w r3, r4**
  - Updates the 16-bit r3 register with the value in the r4 register
  - Updates the 8-bit r3H and r3L register values
  - Updates the condition code register, by setting the N, Z and V bits
- **Information fetched from the processor manual and the instruction decoding**
- **Needed by loop bound analysis**
Decoded instructions, arithmetical effects and timings are all added to Bound-T’s internal program model. CFGs and CG are constructed on-the-fly. Interface provided by Bound-T.

Remaining WCET calculation steps are handled by non-target specific Bound-T modules. WCETs can now be calculated.

**Step 4: Constructing CFGs**

**Instruction**
move.w r3, r4

**Timing**
4 cycles

**Arithmetical effect**
r3.value := r4.value
r3L.value := r4L.value
r3H.value := r4H.value

**WCET and System Timing**

To provide system timing guarantees we need WCET of all system components, including tasks, OS routines, interrupts etc. Used in e.g. schedulability analysis.

\[ R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \]

Provided by the Bound-T WCET tool.
The Asterix & Obelix RT laboratory environment

- A laboratory environment used in real-time courses given at Mälardalens University, Sweden
  - So far 500 students have performed 200 robot projects
- Asterix – a small real-time kernel
- Obelix – a system configuration tool
- Clear separation of functionality and configuration
  - Application might be implemented late, focusing on theoretical aspects of designing robust real-time system
- Suited for schedulability analysis!

Bound-T & the RT laboratory framework

- The framework uses Lego Mindstorms as target platform
- Bound-T will add the following:
  - WCET for student application code
  - WCET for OS calls
  - Illustrative flowgraphs of the code
**System calls in Asterix**

- Asterix has a small set of function calls
- Bound-T was used to derive their WCET
  - These WCETs will be given to the students
  - Students must derive WCET of their own tasks

<table>
<thead>
<tr>
<th>Call</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>Returns tasks own identification number</td>
</tr>
<tr>
<td>getSemaphore</td>
<td>Try to take specified semaphore</td>
</tr>
<tr>
<td>releaseSemaphore</td>
<td>Releases specified semaphore</td>
</tr>
<tr>
<td>raiseSignal</td>
<td>Wakes up all tasks waiting for a signal</td>
</tr>
<tr>
<td>getReadPointerWF</td>
<td>Returns a pointer to a buffer where the value to read is.</td>
</tr>
<tr>
<td>getWritePointerWF</td>
<td>Returns a pointer to a buffer where the user can write a value.</td>
</tr>
<tr>
<td>writeChannel</td>
<td>Writes to a waitfree-channel specified by pointer</td>
</tr>
</tbody>
</table>

**WCET analysis example**

- Some OS-call WCETs were constant:
  - self: \[
  \begin{array}{|c|c|}
  \hline
  self & WCET \ 34 \text{ (cycles)} \ \hline
  \end{array}
  \]
- Some OS-call WCETs were parametrical:
  - getSemaphore:
    - Table:
      | Semaphores | WCET(cycles) |
      |------------|--------------|
      | 1          | 900          |
      | 2          | 1258         |
      | 3          | 1616         |
      | 4          | 1974         |
    - WCET formulas obtained (manually):
      - \[ WCET_{\text{getSemaphore}}: 358 \times (#Sems – 1) + 900 \]
      - #Sems can be taken from the Obelix config file
Conclusions

- Porting a WCET-tool to a new target platform require much work
  - Even though a simple processor
- The resulting WCET tool will be used in education
  - Giving WCETs both for student- and OS- code
- Both the Bound-T WCET tool and the RT laboratory framework will be freely available for academia
  - Together they should provide a solid foundation for development of good RT labs
  - Please contact us for details!

Future work

- Validating the H8/300 Bound-T timing model
  - Against the hardware
- Evaluate the Bound-T tool on students in real-time courses
  - Should provide a lot of valuable feedback
- Investigate if Bound-T can analyze BrickOS
  - An alternative OS for Lego Mindstorms
  - However, not a hard real-time OS
Robot demo & Questions!