TUHH
Hamburg University of Technology

Institute of Embedded Systems

Bus-centric Optimization and Analysis for Multicore Hard Real-Time Systems

Heiko Falk and Dominic Oehlert

Institute of Embedded Systems
Electrical Engineering, Computer Science and Mathematics
Hamburg University of Technology

Motivation

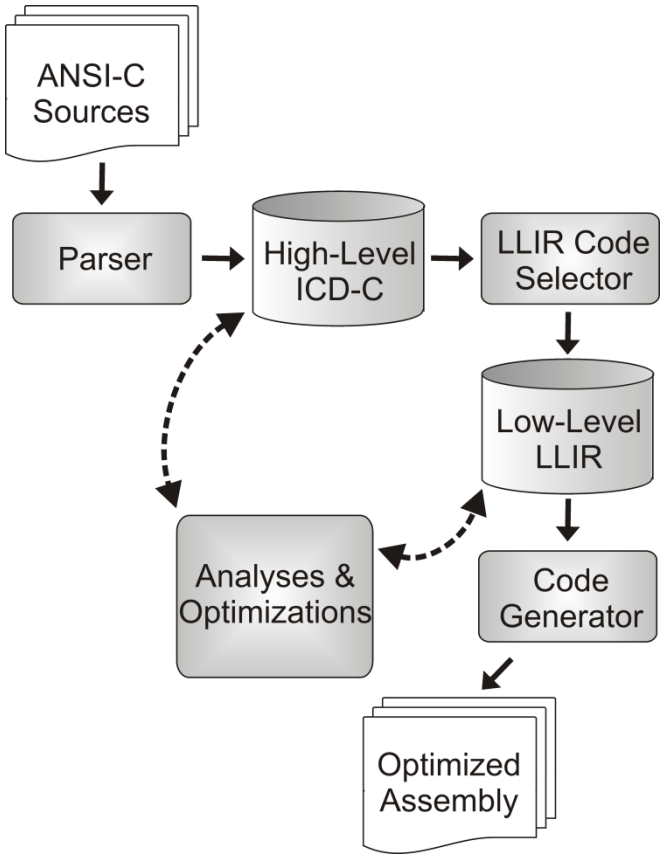
Analysis and Optimization of Real-Time Systems

- Only meaningful when done at code level (i.e., source or assembly code), in contrast to “black box” task- or system-level analyses
- Current industrial design flows for real-time software:
  - Code generation and timing analysis disrupted
  - No continuous automation, error-prone manual human intervention
  - No tool support for systematic optimization of real-time properties
- No methodology for parallel multi-task/multi-core real-time systems

Design of a Compiler that

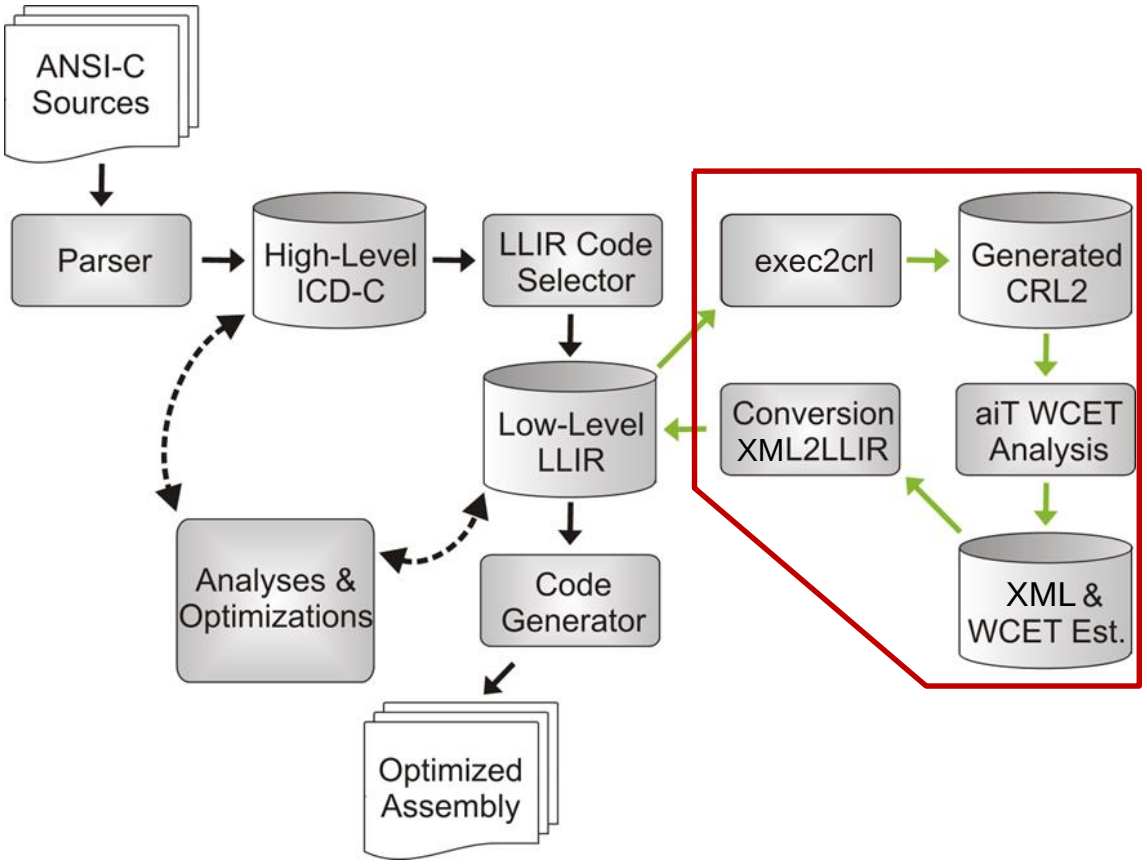
- allows formal guarantees on worst-case properties, instead of relying on observed execution times,
- applies fully automated optimizations to minimize $WCET_{EST}$, even for parallel real-time systems

WCC – The WCET-aware C Compiler



- ICD-C: High-level Intermediate Representation (IR)
- Processor-independent, close to C input
- Code Selector: Translates ICD-C into Low-Level Intermediate Representation (LLIR)
- LLIR: Processor-specific representation of assembly code
- Supported processors: Infineon TriCore TC1796 and TC1797; ARM7 single- and multi-core
  ARM Cortex-M and LEON3 ongoing

Integration of WCET Analysis into WCC Compiler



Relevant WCET data:

- $WCET_{EST}$ of entire program, function of basic block
- Worst-Case execution frequency per function, basic block or CFG edge
- Potential register contents
- Cache hits / misses per basic block

[http://www.tuhh.de/es/esd/research/wcc]

Memory Hierarchies and Execution Times

Memories and Execution Times

- Overall system performance largely dominated by memory and interconnect subsystem
- Large speed gap between slow memories/buses and fast processors
- Execution time of software depends on characteristics
  - ☞ of underlying memory and bus hierarchy and
  - ☞ of memory accesses performed by a program

☞ WCET estimates also heavily depend on memories and buses!

Consequences for WCET-aware Compiler

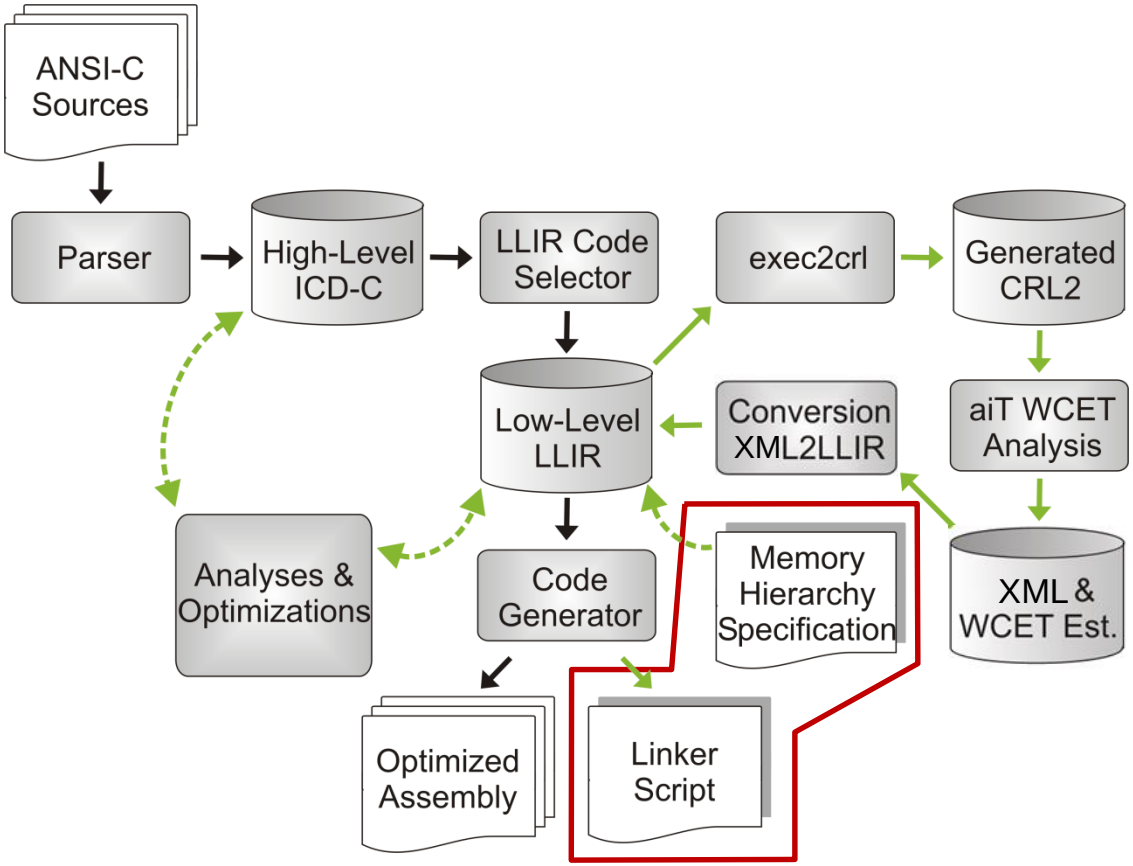
Compiler and Memory Hierarchies

- Compiler is responsible to pass memory-related information to WCET analyzer
- WCET estimates must adhere to a program's actual memory layout

The compiler...

- ... decides on a program's memory layout, and not the linker!
- ... needs detailed knowledge about memories

Specification of Memory Hierarchies



Plain-text interface per memory region:

- base address, length
- access attributes
- access times
- assembly-level sections

```
# Data SRAM (DMU)
[DMU-SRAM]
origin = 0xc0000000
length = 0x10000 #64k
attributes = RWA
cycles = 6
sections = .data.sram
```

Flow Facts (1)

Static WCET Analysis

– Estimates the longest possible execution path

– Such paths can contain cycles stemming from loops and/or recursions

☞ How many times are such cycles iterated in the worst case?

☞ Traversals of cycles have to be upper-bounded for WCET analysis!

Flow Facts...

– ... are (user-provided) meta-information containing such upper bounds

– ... have to be supported by a WCET-aware compiler

Flow Facts (2)

aiT's Flow Fact Support

- Via separate annotation file
- Contains loop bounds and recursion depths at machine code level, i.e., based on physical memory addresses

☞ Extremely low-level, tedious and error-prone to generate Flow Facts

☞ Cumbersome since Flow Facts must be (manually) updated if a program's memory layout changes

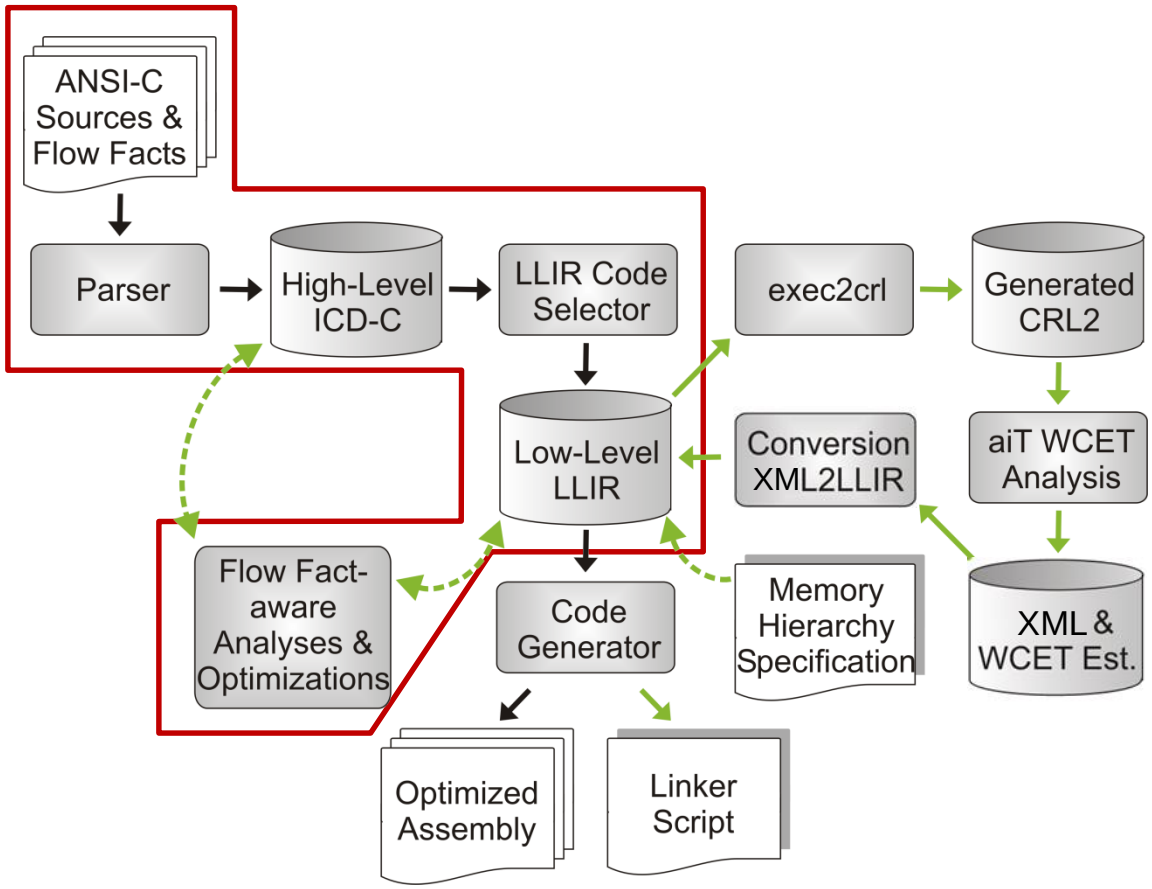
WCC's Flow Fact Support

- High-level, directly within ANSI-C source codes

```
_Pragma( "loopbound min 100 max 100" )
  for ( i = 1; i <= 100; i++ ) ...
```

- No relation to machine code level required for programmer
- Automatic Flow Fact update during code transformations

WCC's Flow Fact Mechanisms



Flow Fact-Awareness:

- Transversal concept in the entire compiler
- Starting in the front-end
- Touching all IRs and code selection
- Covering all optimizations
- Ending in the back-end during WCET analysis

Intermediate Wrap-Up

WCC's Design Flow

- Reduces degree of required manual interventions
- Raises abstraction level of flow facts
- Provides systematic approach for code optimization

Limitations: Simplifying Assumptions of static WCET Analyzers

- Single-Task systems
  - Code under analysis executes exclusively on a processor core
  - No preemptions, no interrupts
- Single-Core systems
  - Activities executed on parallel cores neglected
  - Shared hardware resources (buses, memories) and arbitration policies neglected