Virtualisation in Use

An AURIX/***T1*** demo

Dipl.-Ing. (BA) **Peter Gliwa**, Gliwa GmbH

Dipl.-Ing. **Jens Harnisch**, Infineon Technologies AG

Contents

- **Motivation**: analyse variants of a function
- **How it works**: virtual function substitution
- **How it works**: virtual function migration
- **Summary**

Use case: analyse function variants

- Need to gather *actual performance data* in order to select between a number of variants of a function
  - Performance of the function itself
  - Impact of the function
    - stack usage
    - shared memory conflicts
    - cache usage

- A build, flash and run cycle takes several hours

- Virtual function substitution allows a large number of variants to be trialled with one build

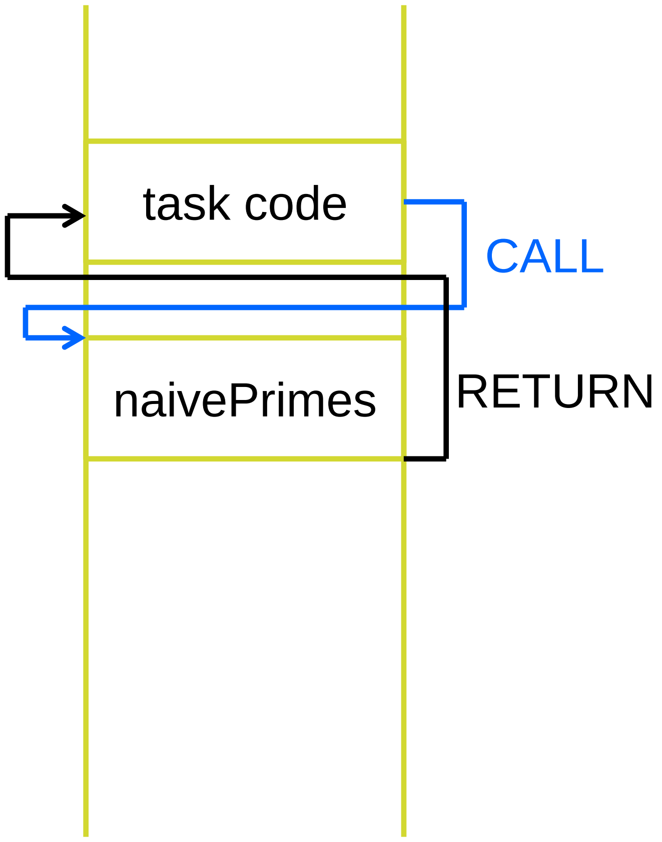
Use case: calculate prime numbers

- Any computationally intensive function will demonstrate the principles.
- Naïve algorithm tests successive number *N* to see if they have a factor such that $1 < \text{factor} <= \sqrt{N}$
- Alternative algorithm uses lazy, sparse Sieve of Eratosthenes with fewer arithmetic operations but more memory accesses
- Which performs better in a real system?

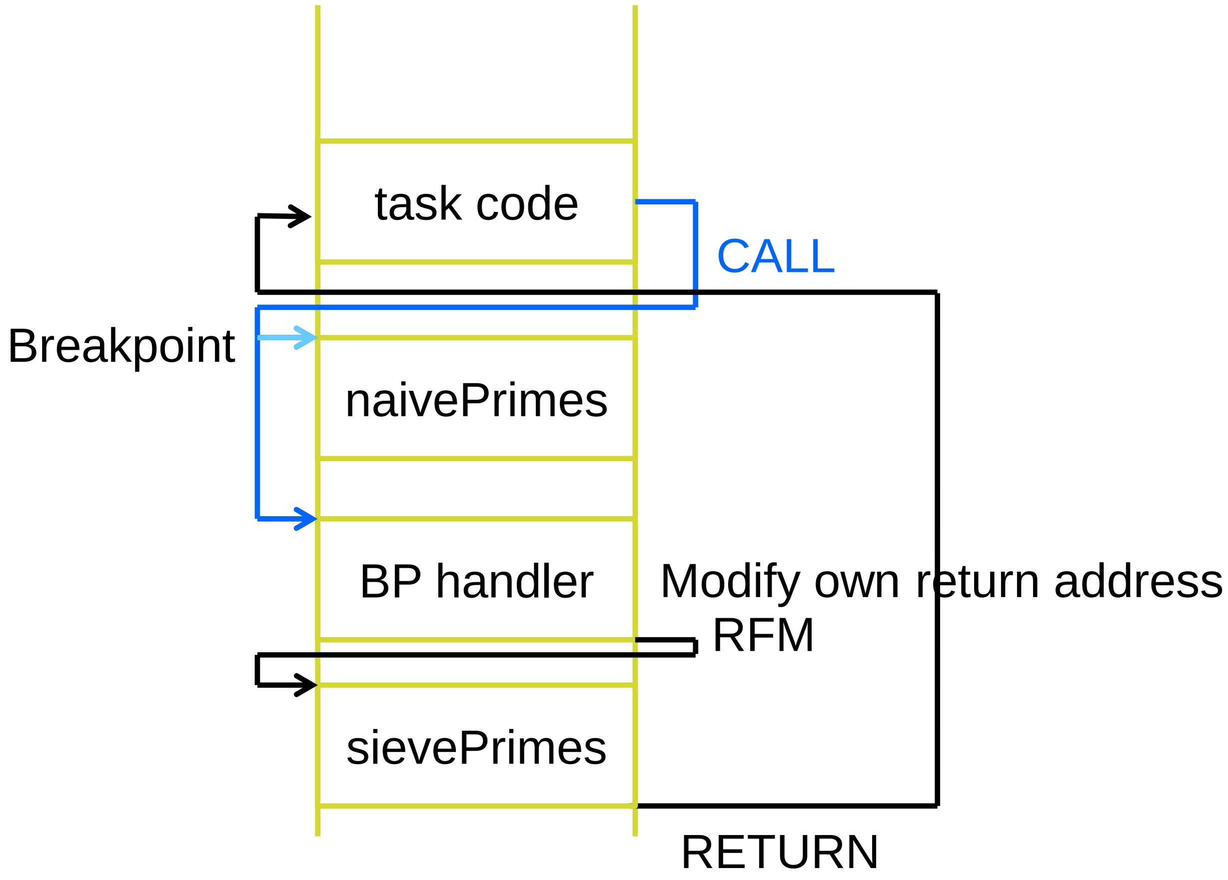
Use case: environment

- Infineon AURIX with 3 TriCore CPUs:
  - CPU0, V1.6E (Efficiency) core, mostly event-driven schedule
  - CPU1, V1.6P (Performance) core, mostly periodic schedule
  - CPU2, V1.6P (Performance) core, reserved in this demo
- Tasking v4.1r1 TriCore compiler
- Gliwa ***T1*** triggers demo phases and visualises timing effects

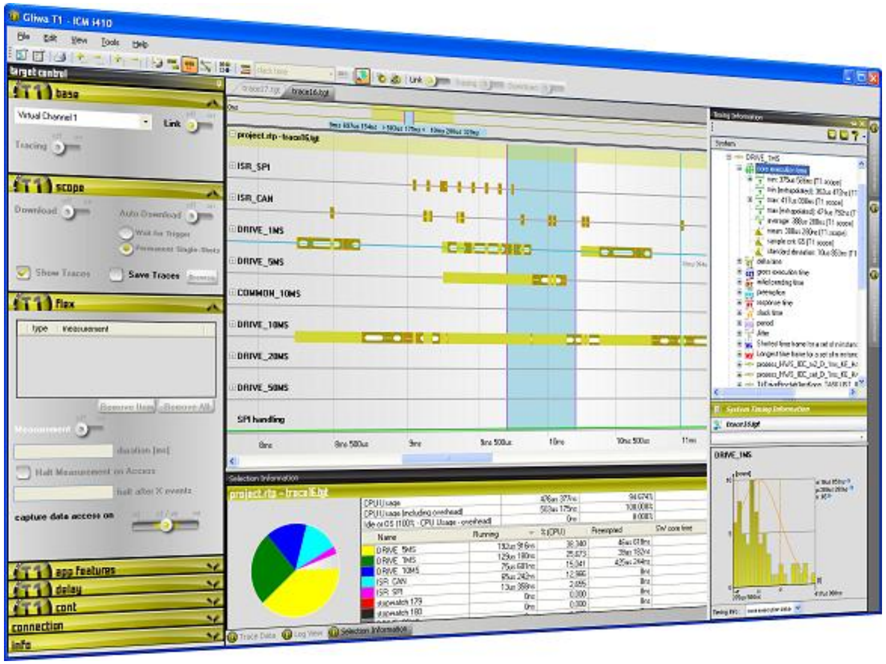
Function substitution disabled



Function substitution enabled



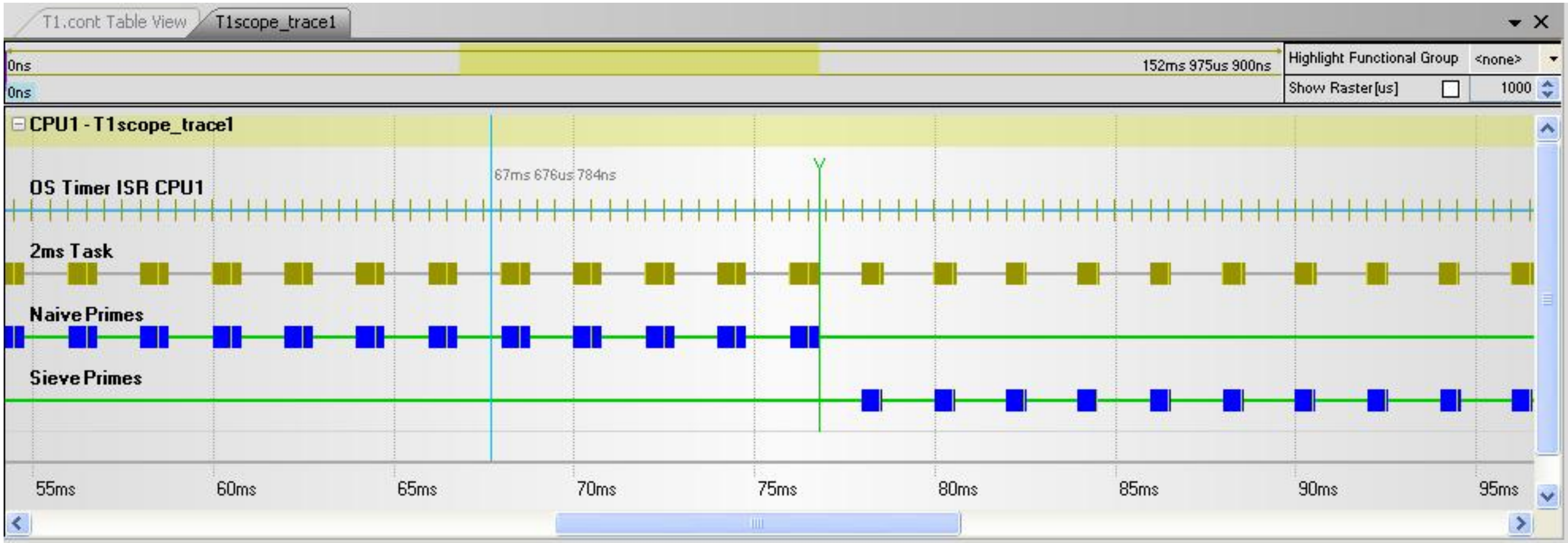
Used for example analysis: ***T1*** timing suite



- Runtime measuring, debugging, verification and optimisation
- System and code level timing analysis
- Oscilloscope-like visualisation of runtime scenarios
- Net run times for tasks, interrupts, functions or any code fragment
- CPU load measurement
- On-target measurement and supervision
- On-line instrumentation of code
- Easy connection to target hardware – no HW modification required
- Interfaces to static code- and scheduling analysis tools
- Embedded in AUTOSAR processes

AUTOSAR REAL-TIME EXPERTS Symtavision Gliwa Absint ALL|times

Results of substitution on CPU1

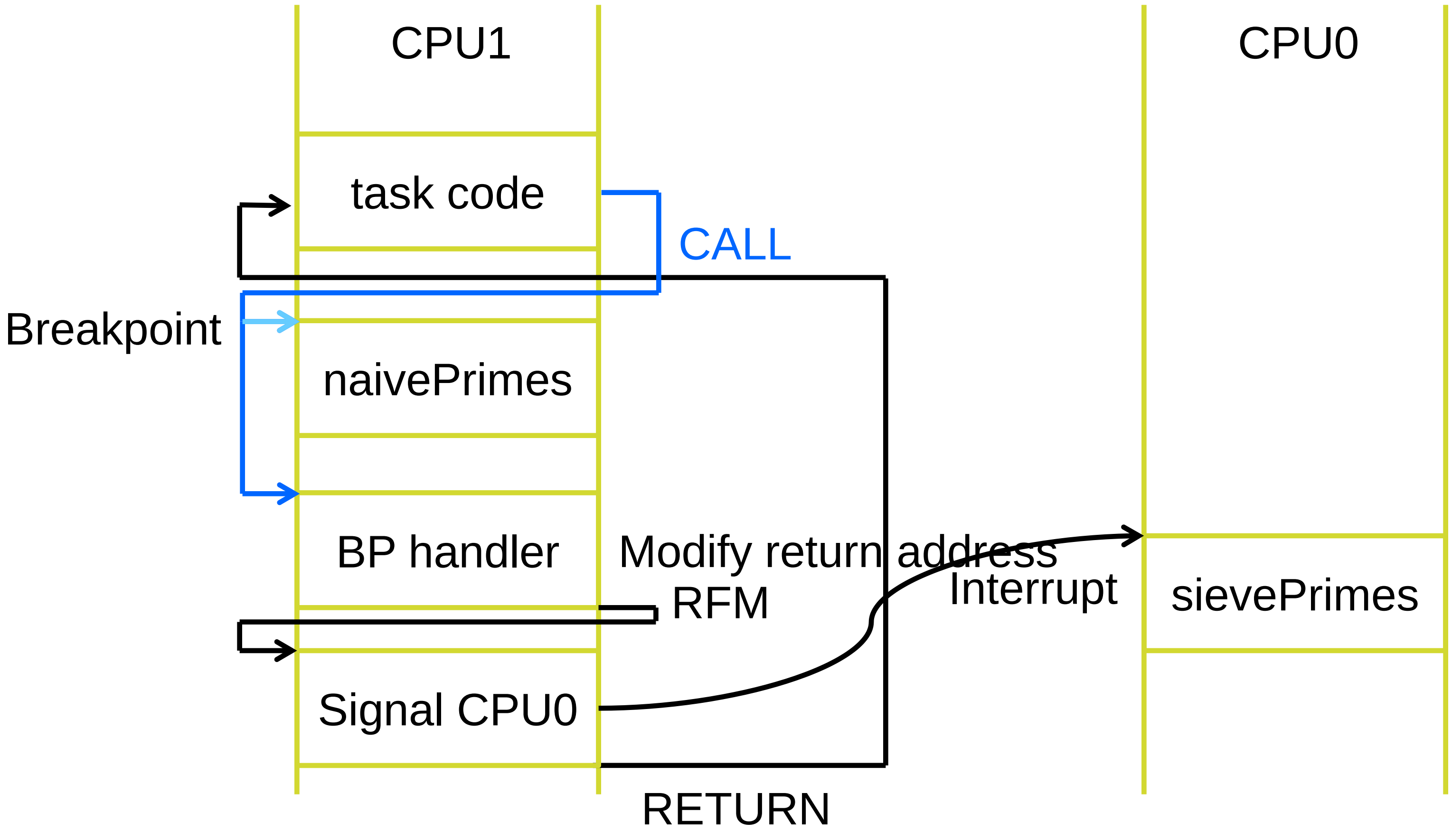


- Naïve Primes max CET = 825µs
- Sieve Primes max CET = 560µs
  - 32% reduction in execution time

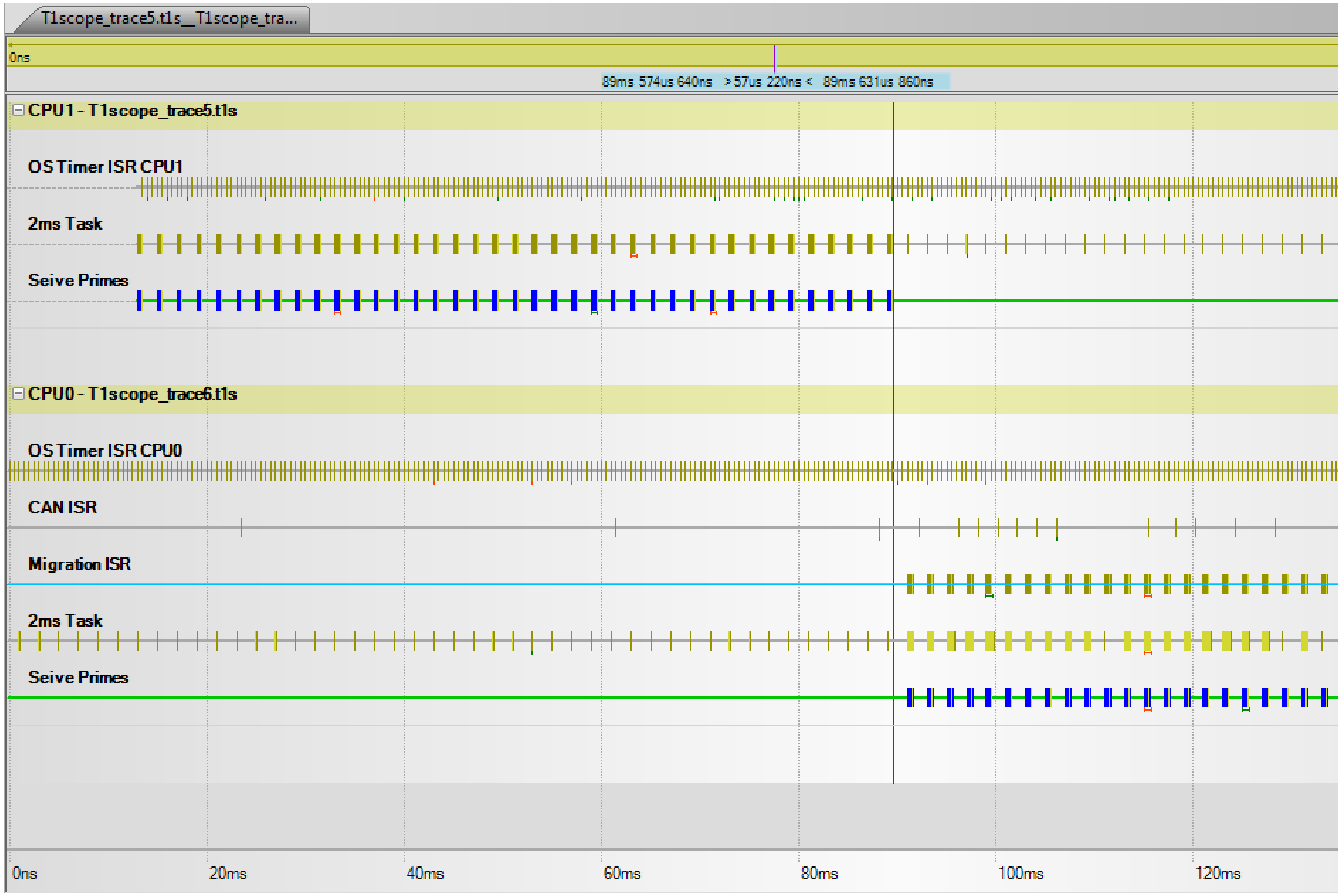
Use case: analyse multicore load balance

- If we can substitute one function with another, we can just as well migrate the whole function to another core
- The optimised prime calculator is fast enough to run on CPU0’s slower 1.6E core
- So let us migrate sievePrimes to CPU0, freeing CPU load on CPU1

Function migration



Results of migration from CPU1 to CPU0



Migration results 2

- Sieve Primes on CPU1 max CET = 560µs
- Sieve Primes on CPU0 max CET = 680µs
  - CPU load increased by about 35%
- CPU0 can manage the extra CPU load
- Prioritised interrupts mean that CPU0 is unaffected with regard to previous interrupts, which have higher priority than the migration interrupt

Summary of example, outlook

- Using simple HW features, we can
  - substitute one function with another
  - migrate a function from one core to another

- The replacement function could equally be compiled, located and loaded to RAM *while the system is operating*