Proactive Energy-Aware Programming with PEEK

Timo Hönig, Heiko Janker, Oliver Mihelic[1], Christopher Eibel,
Rüdiger Kapitza[1], and Wolfgang Schröder-Preikschat

FAU Erlangen-Nuremberg [1]TU Braunschweig

TRIOS'14, Broomfield, CO, USA
October 5, 2014



FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

Technische
Universität
Braunschweig

DFG

SFB/TRR 89 „Invasive Computing"

Energy-Aware Programming: Status Quo





- - -

- - -

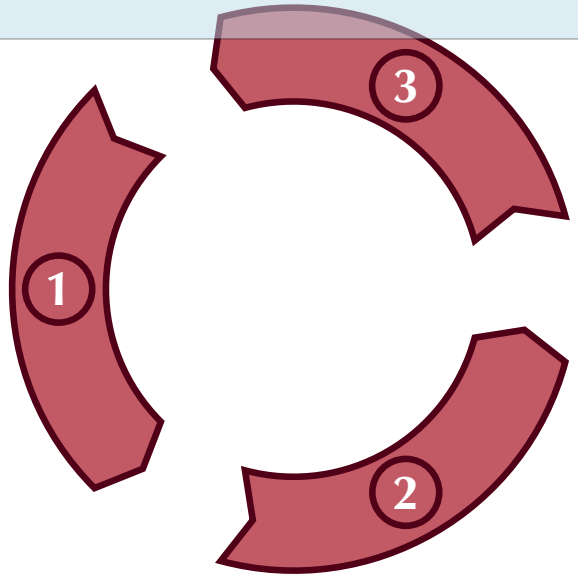




Energy-Aware Programming: Status Quo

Recurring Tasks to Optimize Code Manually

1. Write, modify, and build application code
2. Run code with defined input (e. g., parameters, use case)
3. Perform energy analysis (hardware/software)

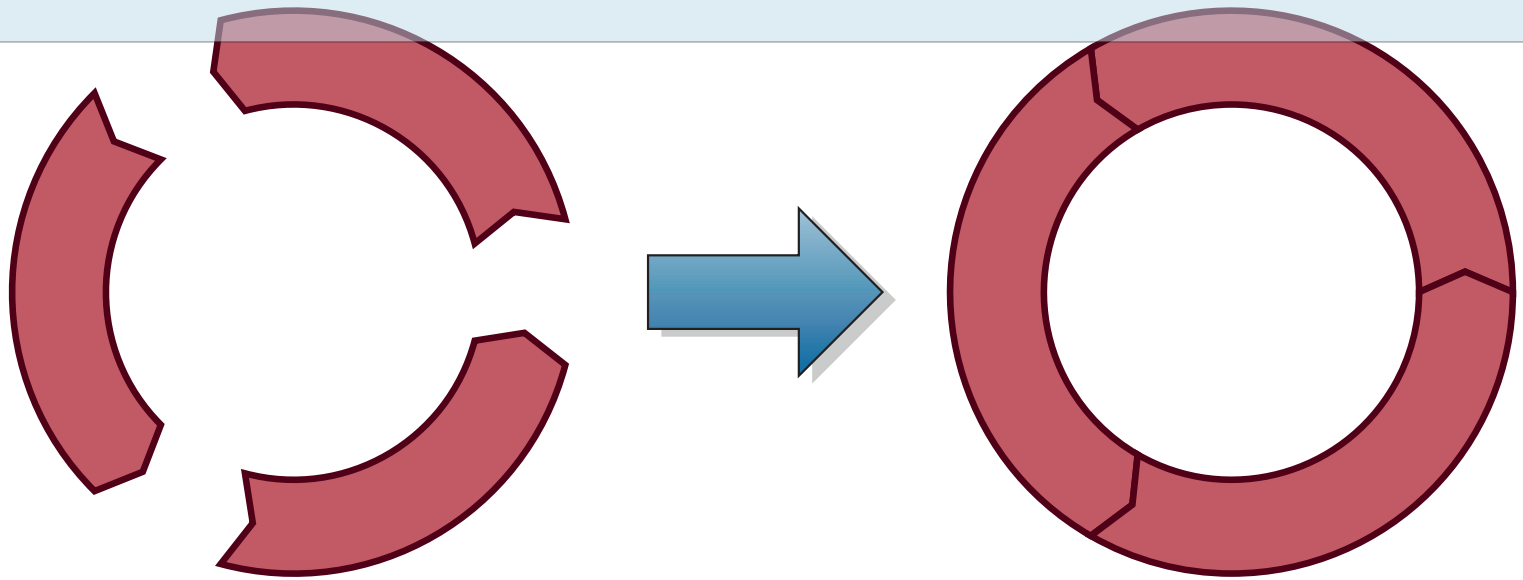




Energy-Aware Programming: Challenges

Challenges

- TODO Integrated tooling support (e. g., IDEs)
- TODO Fully automated and accurate energy measurements
- TODO Automatic suggestions on how to improve code





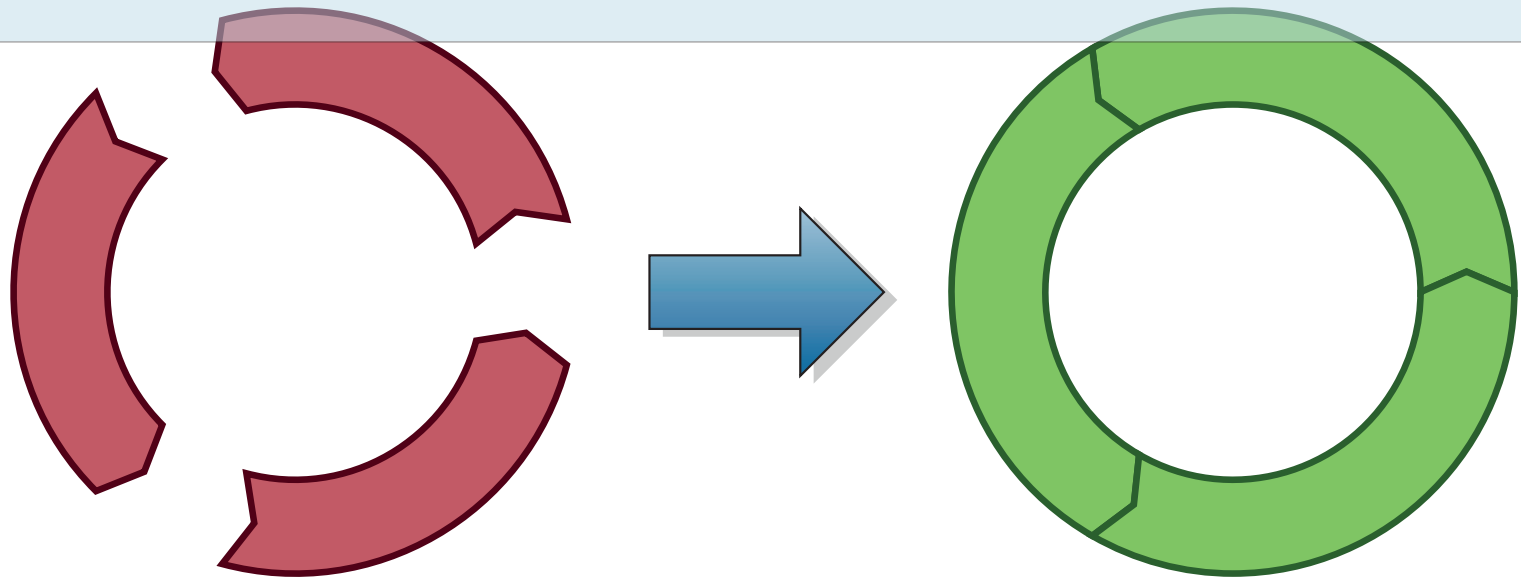
Proactive Energy-Aware Programming with PEEK

Proactive Energy-Aware Programming Kit (PEEK)

→ Systems approach to proactive energy-aware programming

→ Energy analysis and optimization right at development time

→ Evaluation: 8.4x speed-up, decrease energy demand by 25.3 %





Agenda

Energy-Aware Programming
- Status Quo
- Challenges

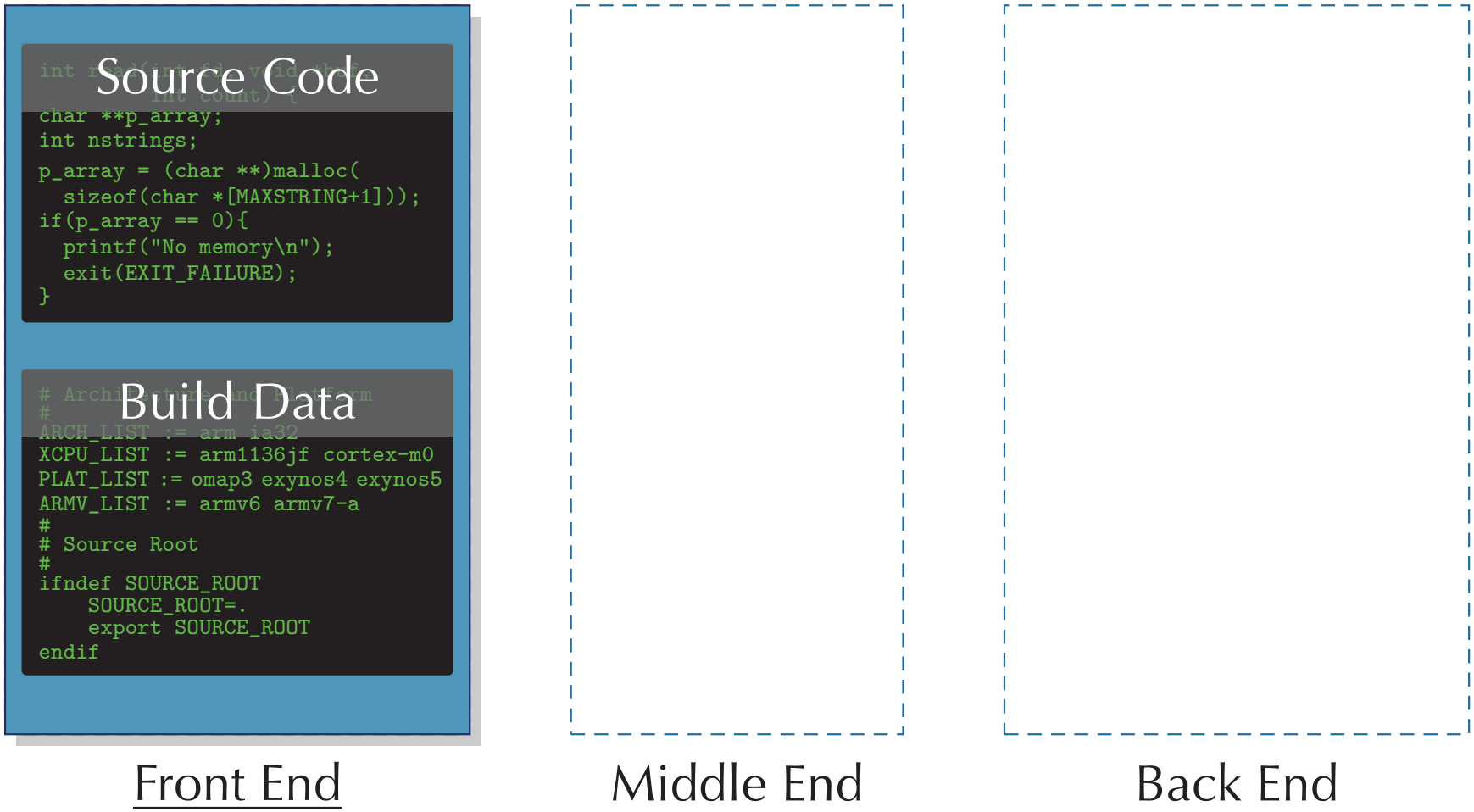
Proactive Energy-Aware Programming Kit
- Design and System Architecture
- Implementation and Operation
- Energy Optimization Hints
- Electronic Measuring Device

PEEK

Evaluation
- Scenarios
- Results



Design and System Architecture

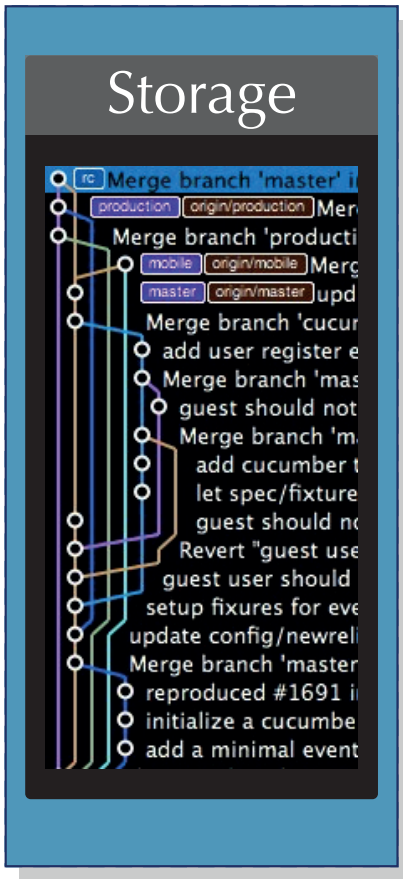


- User interface (command line, IDE)
- Main system control (acquisition of data, analysis operations)



Design and System Architecture







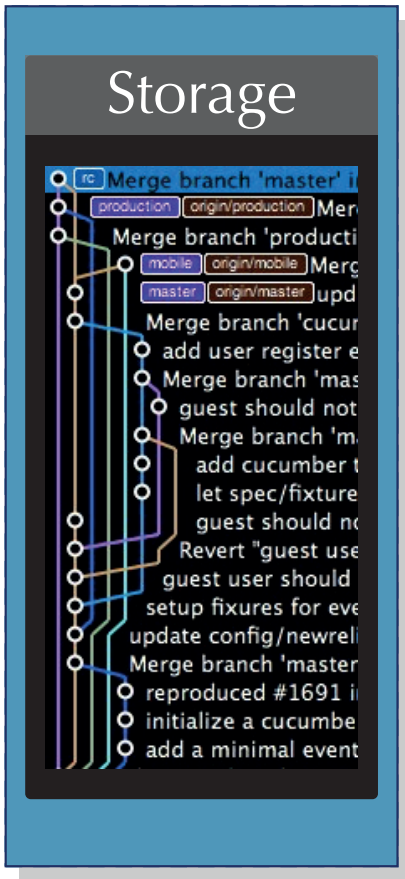
Front End | Middle End | Back End

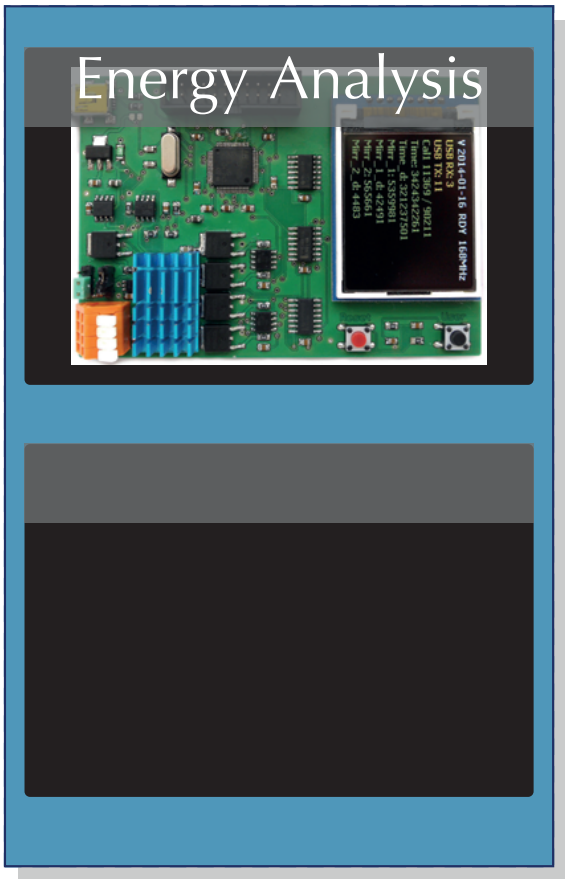
- Data storage (i. e., source code, build data, meta data)
- Passive component (used from existing development environments)



Design and System Architecture







Front End | Middle End | Back End

- Direct and indirect energy analysis (i. e., hardware, software)
- Energy consumption values at function level



Design and System Architecture

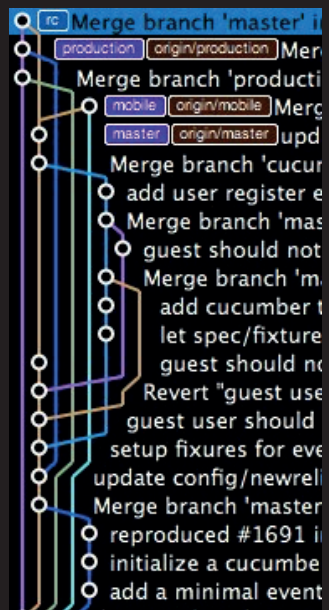
Source Code

```
char **p_array;
int nstrings;
p_array = (char **)malloc(
  sizeof(char *[MAXSTRING+1]));
if(p_array == 0){
  printf("No memory\n");
  exit(EXIT_FAILURE);
}
```

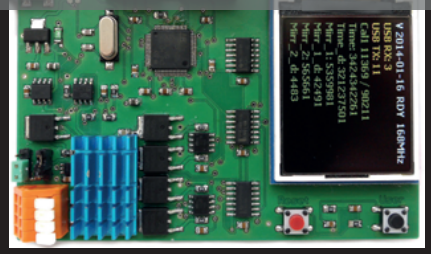
Build Data

```
ARCH_LIST := arm ia32
XCPU_LIST := arm1136jf cortex-m0
PLAT_LIST := omap3 exynos4 exynos5
ARMV_LIST := armv6 armv7-a
#
# Source Root
#
ifndef SOURCE_ROOT
    SOURCE_ROOT=.
    export SOURCE_ROOT
endif
```

Storage



Energy Analysis



Optimizations

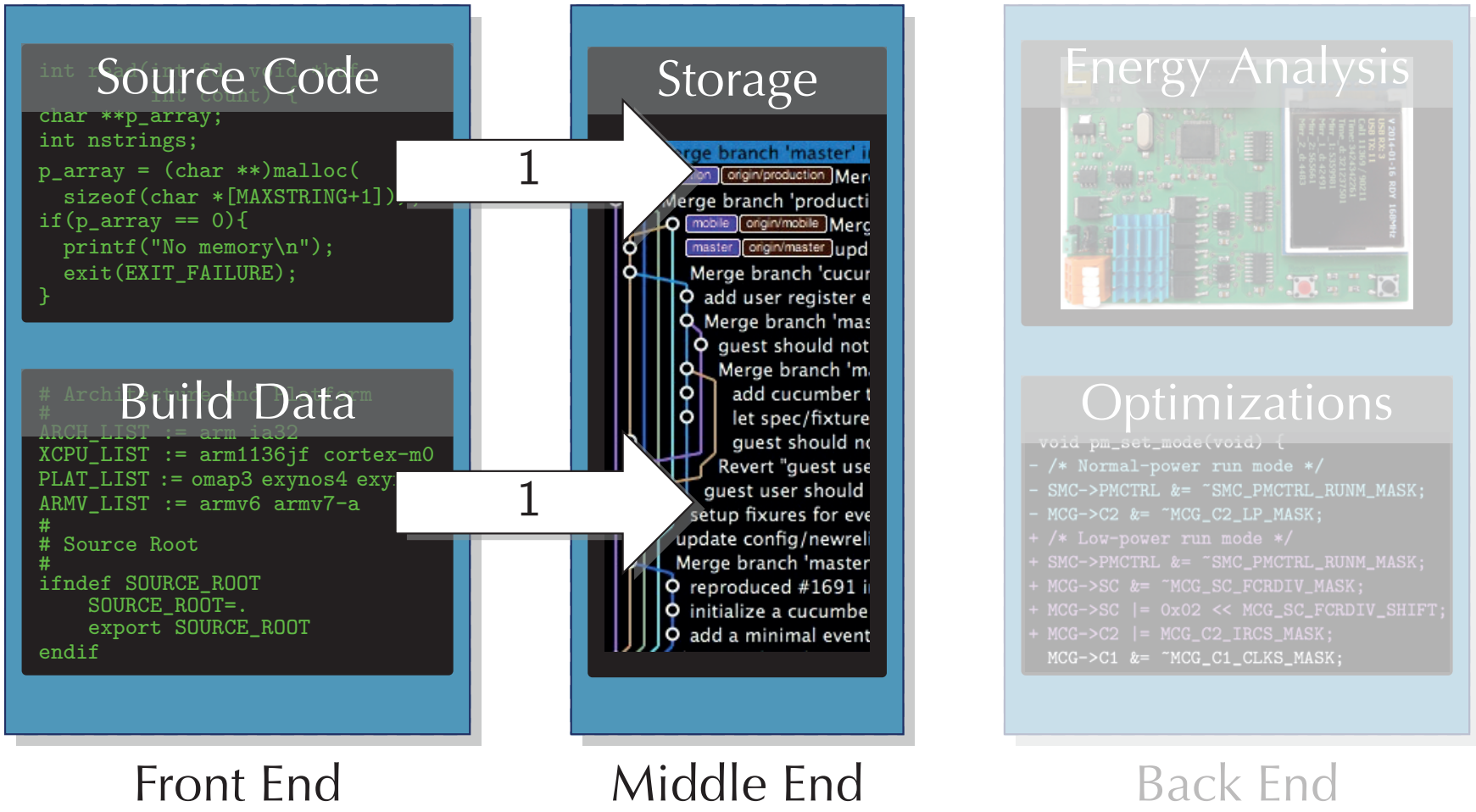
```
void pm_set_mode(void) {
- /* Normal-power run mode */
- SMC->PMCTRL &= ~SMC_PMCTRL_RUNM_MASK;
- MCG->C2 &= ~MCG_C2_LP_MASK;
+ /* Low-power run mode */
+ SMC->PMCTRL &= ~SMC_PMCTRL_RUNM_MASK;
+ MCG->SC &= ~MCG_SC_FCRDIV_MASK;
+ MCG->SC |= 0x02 << MCG_SC_FCRDIV_SHIFT;
+ MCG->C2 |= MCG_C2_IRCS_MASK;
  MCG->C1 &= ~MCG_C1_CLKS_MASK;
```

Front End | Middle End | Back End

- Energy optimization hints (suggestions to reduce energy footprint)
- Source code patches or build system modifications



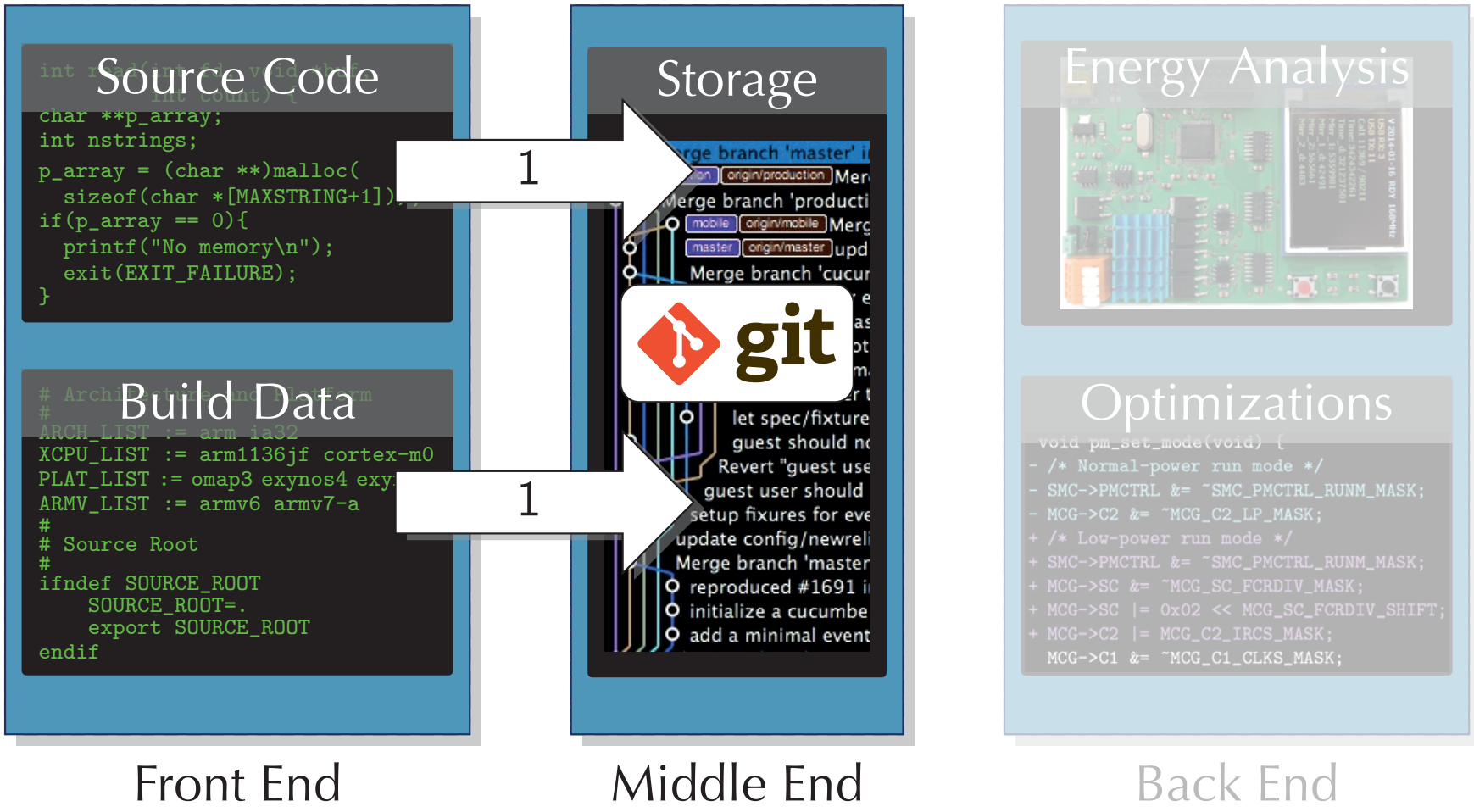
Implementation and Operation



- Snapshot commit (source code, build data)
- Energy analysis configuration (meta data)



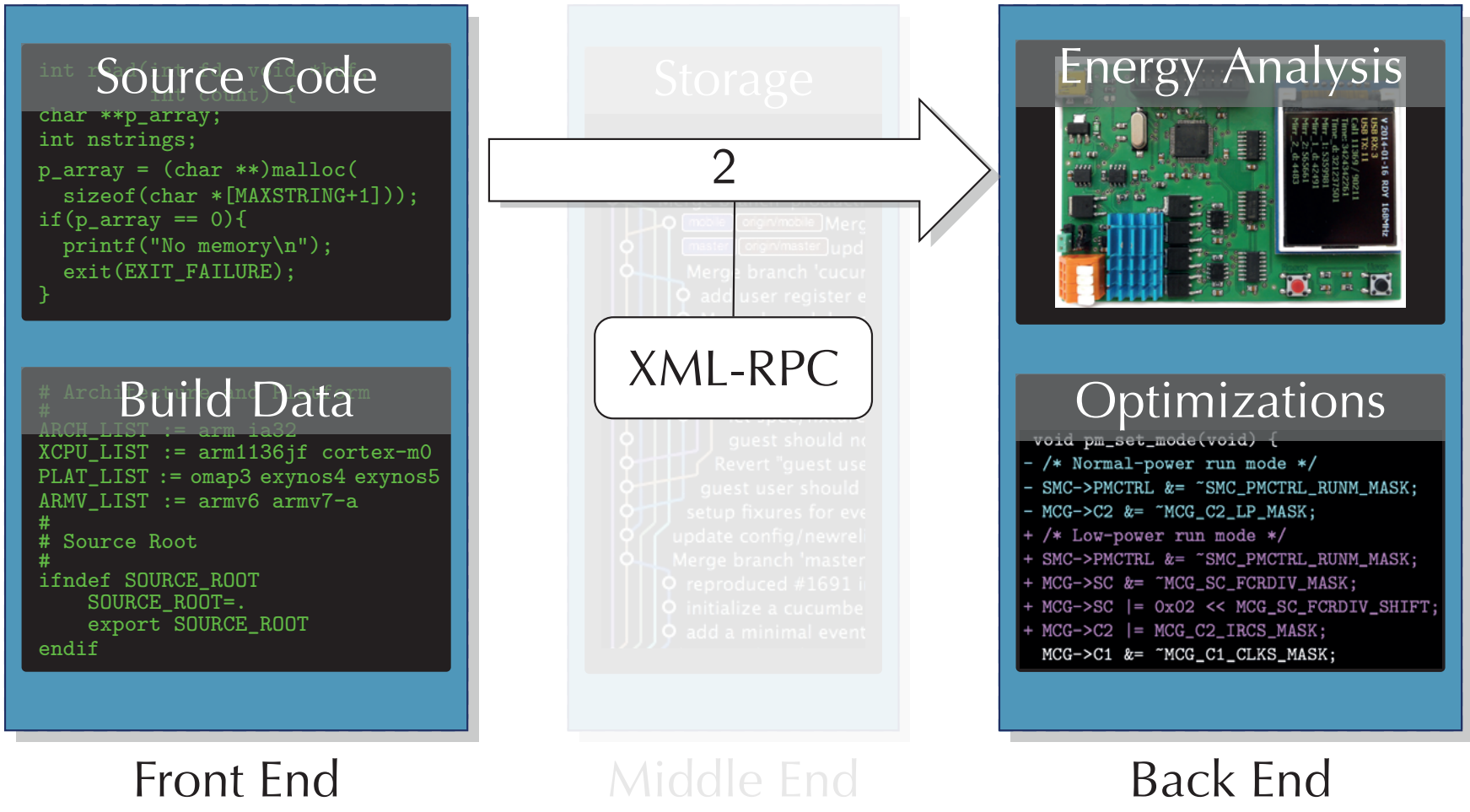
Implementation and Operation



- PEEK snapshots are mapped to Git branches
- Analysis of multiple source code revisions via batch operation

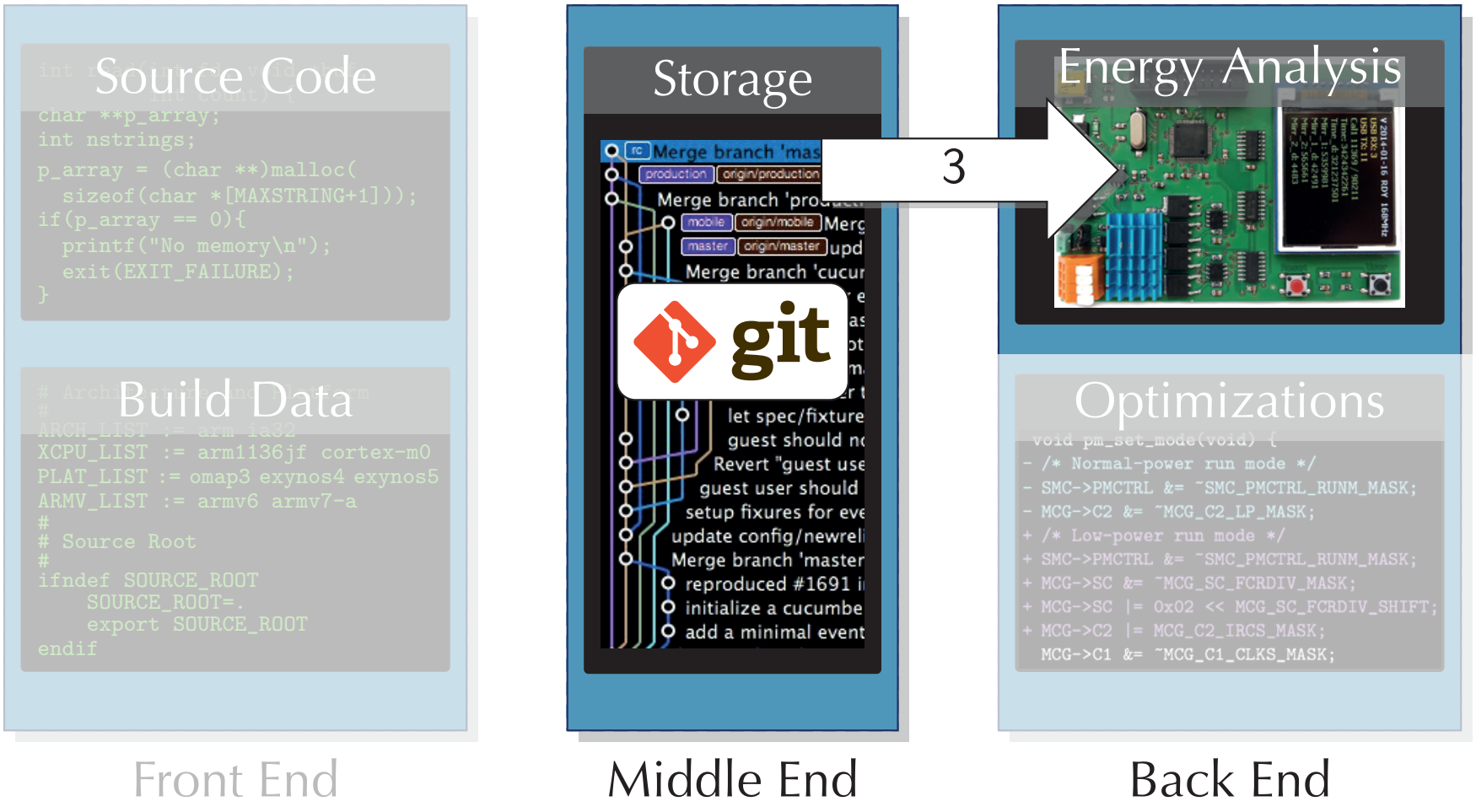


Implementation and Operation



- Front end triggers energy analysis via XML-RPC
- Asynchronous operation

Implementation and Operation



- Back end pulls Git branches from the repository
- Prepares energy analysis (direct or indirect)



Implementation and Operation: Energy Analysis

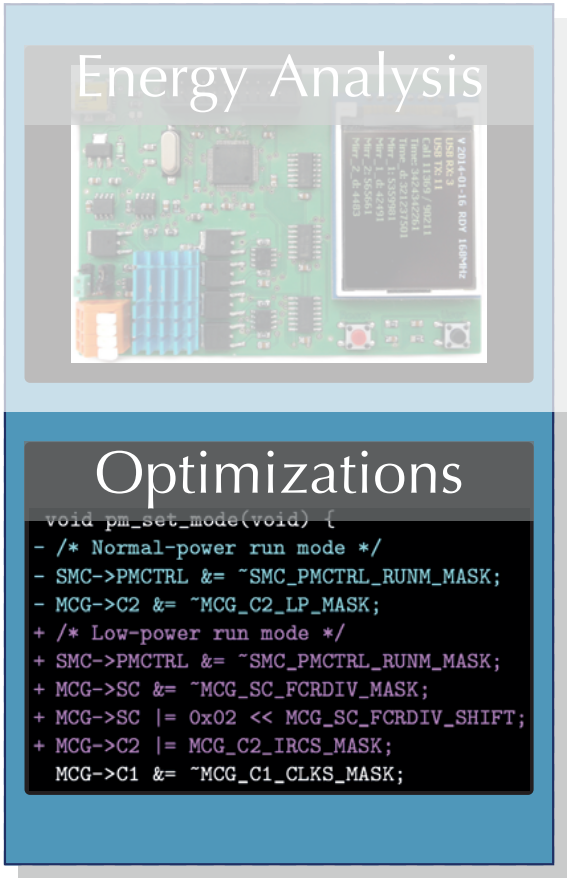
- Direct Energy Analysis (Hardware)
  - initialize energy measurement device (i. e., oscilloscope, multimeter, ADC)
  - deploy binary to target platform
  - execute application, run energy analysis
- Indirect Energy Analysis (Software)
  - setup environment for energy profiler
  - deploy binary to profiler
  - execute application, run energy analysis





Implementation and Operation: Optimizations

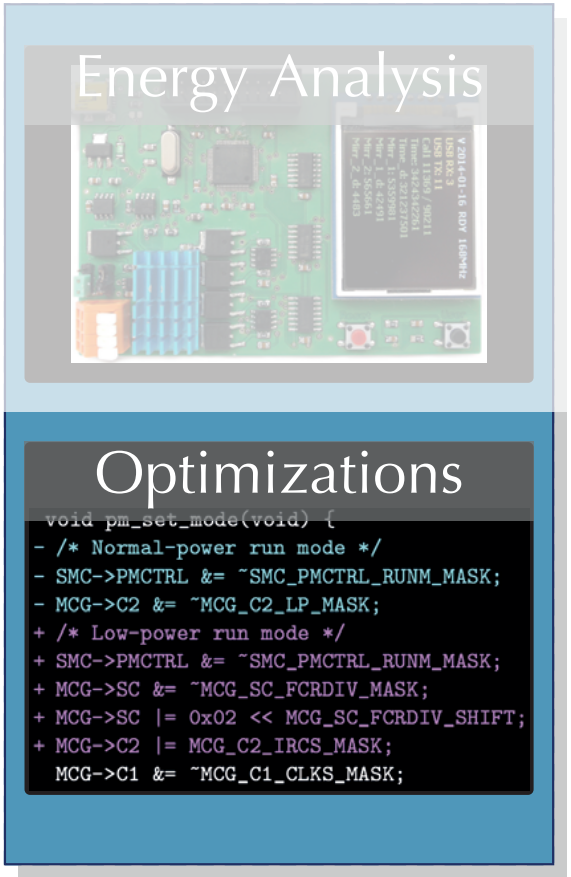
- Optimizing for Low Energy
  - exploit hardware energy saving features: find optimal set
  - apply software changes: restructure, modify and align timers, use different algorithms





Implementation and Operation: Optimizations

- Optimizing for Low Energy
  - exploit hardware energy saving features: find optimal set
  - apply software changes: restructure, modify and align timers, use different algorithms
- Back-End Snapshot Analysis
  - modify original snapshot, examine changes for potential energy savings
  - push modified snapshot as a new branch to Git (private), initiate energy analysis

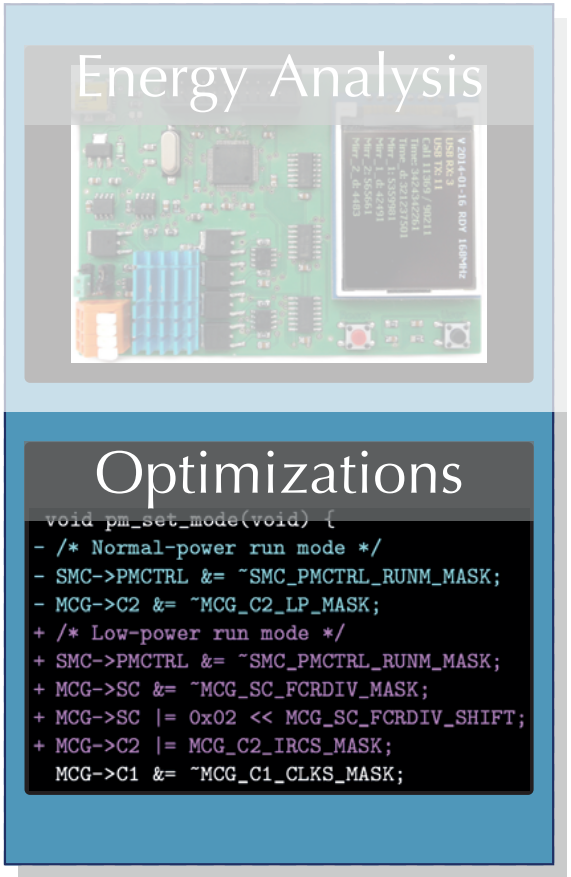


Back End



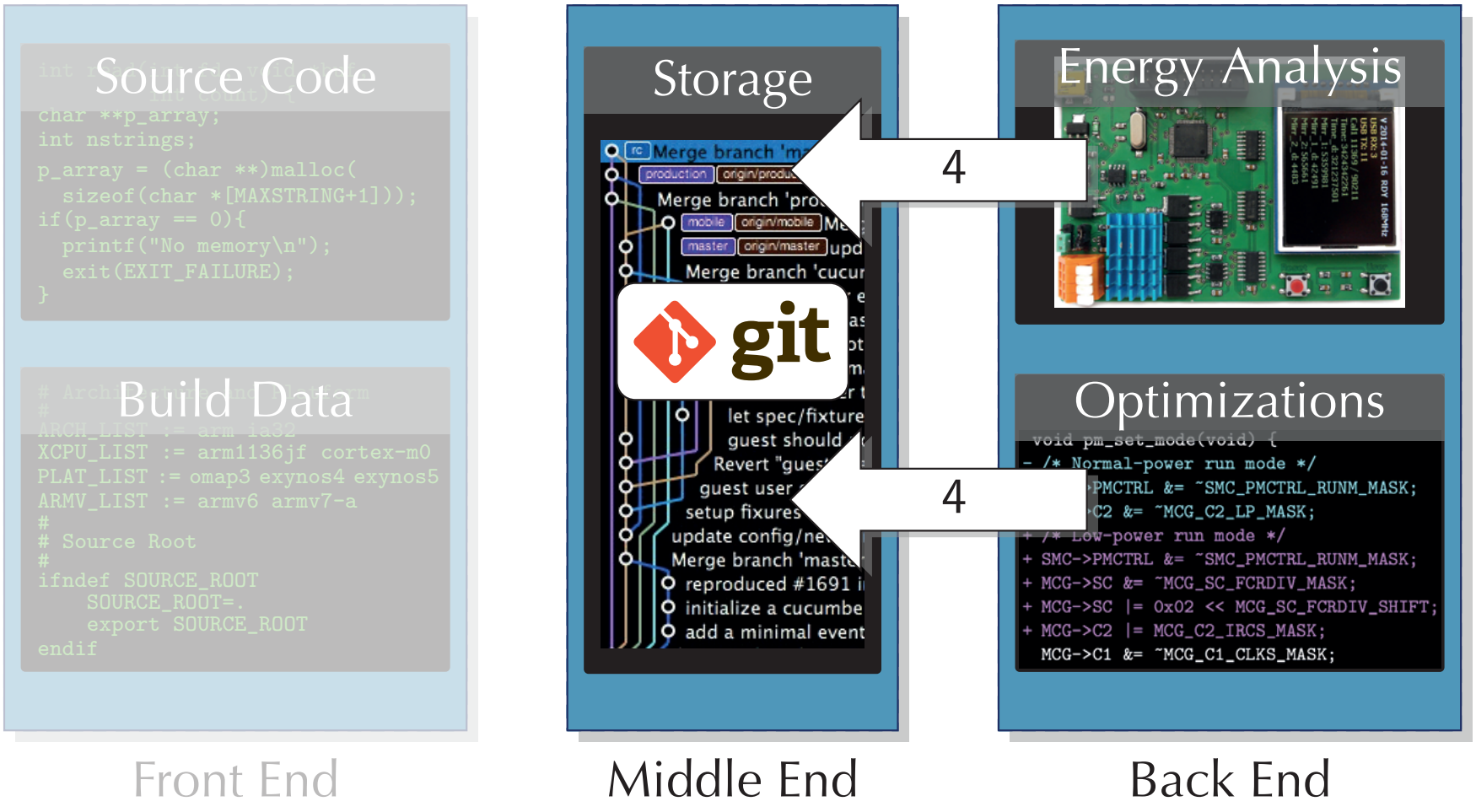
Implementation and Operation: Optimizations

- Optimizing for Low Energy
  - exploit hardware energy saving features: find optimal set
  - apply software changes: restructure, modify and align timers, use different algorithms
- Back-End Snapshot Analysis
  - modify original snapshot, examine changes for potential energy savings
  - push modified snapshot as a new branch to Git (private), initiate energy analysis
- Optimization Candidates
  - private snapshots with improved energy footprint become optimization candidates
  - all other private snapshots are omitted





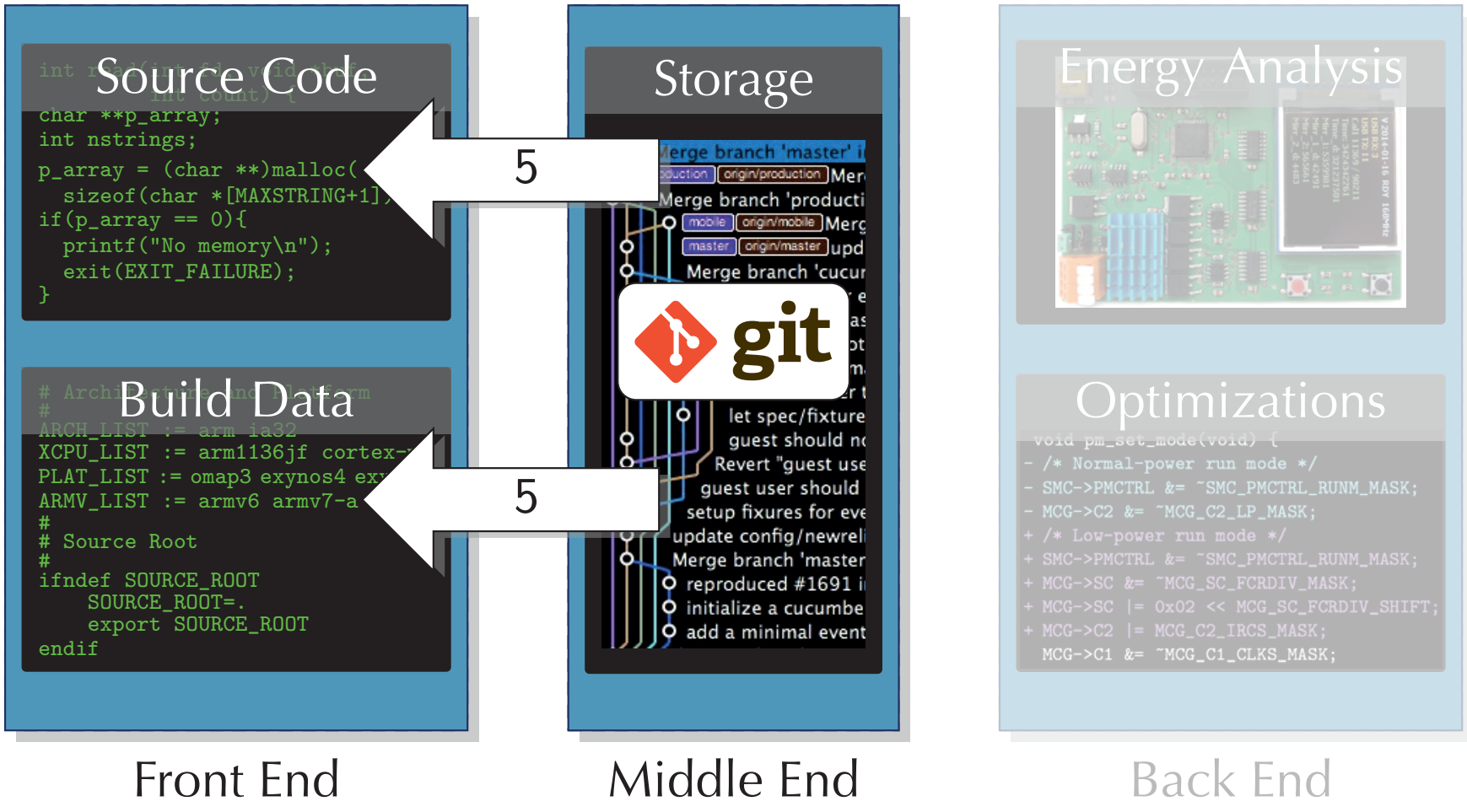
Implementation and Operation



- Energy analysis results are pushed to the middle end
- Candidate snapshots provide base data for optimization hints

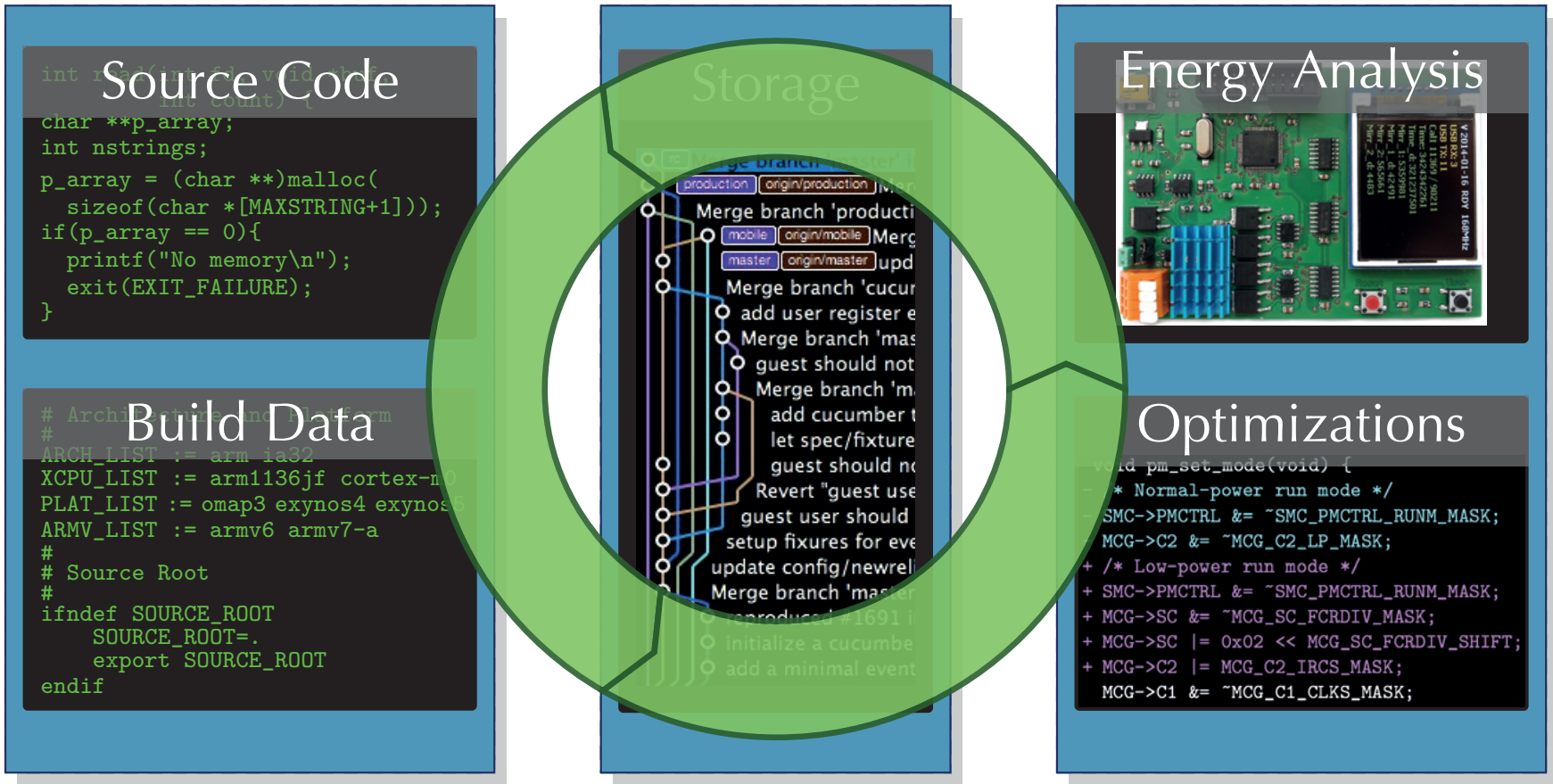


Implementation and Operation



- Front end pulls energy analysis results from middle end
- Generate energy optimization hints: process base data → patches

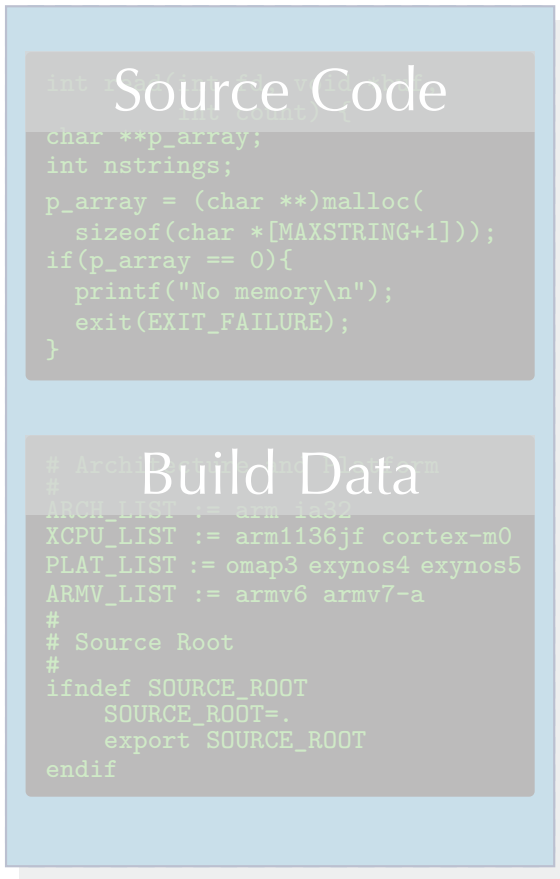
Implementation and Operation

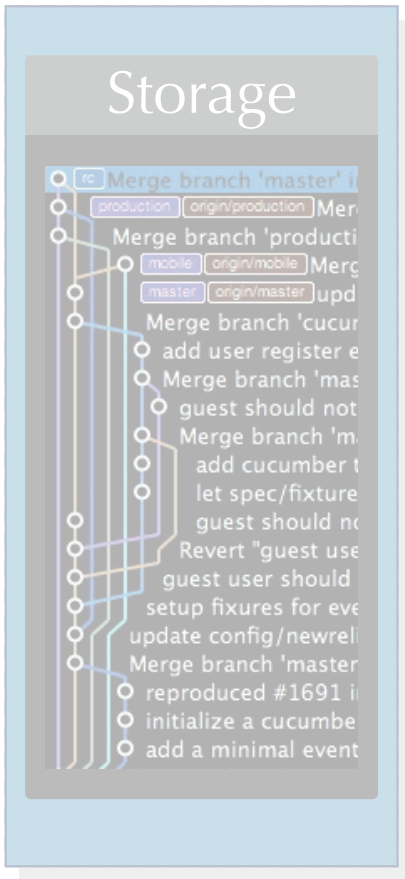


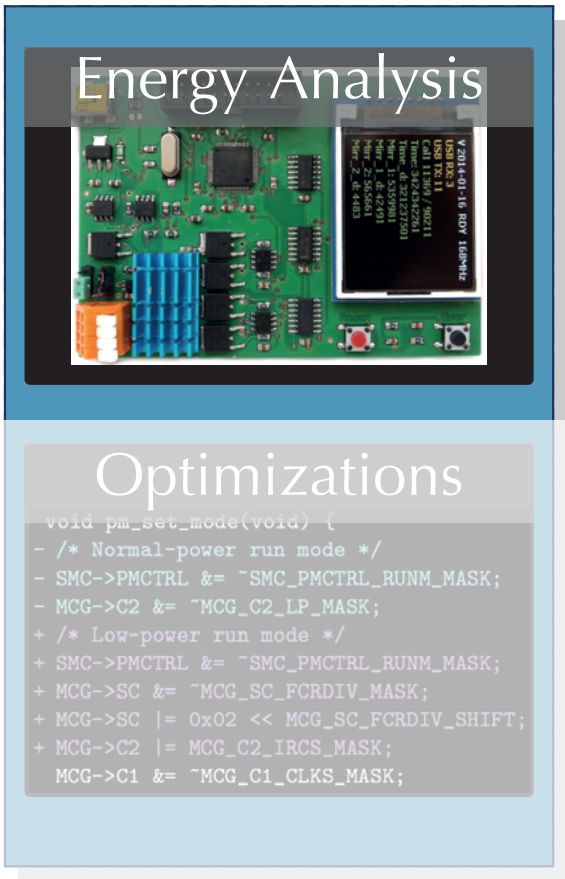
- Implementation and operation of the PEEK software stack



Implementation and Operation







Front End | Middle End | Back End

- Implementation and operation of the PEEK software stack
- Lightweight, yet powerful electronic measuring device for PEEK



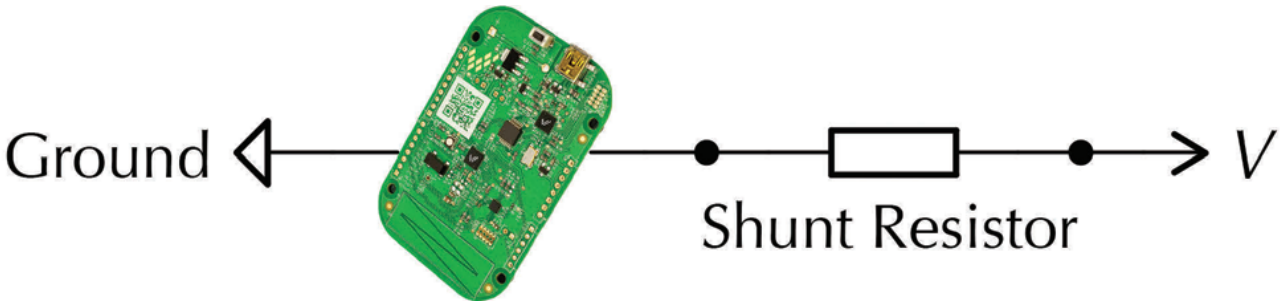
Electronic Measuring Device: Motivation

- Measuring energy consumption
  - analog-to-digital converter (ADC)
  - multimeter
  - oscilloscope











Electronic Measuring Device: Motivation

- Measuring energy consumption
  - analog-to-digital converter (ADC)
  - multimeter
  - oscilloscope





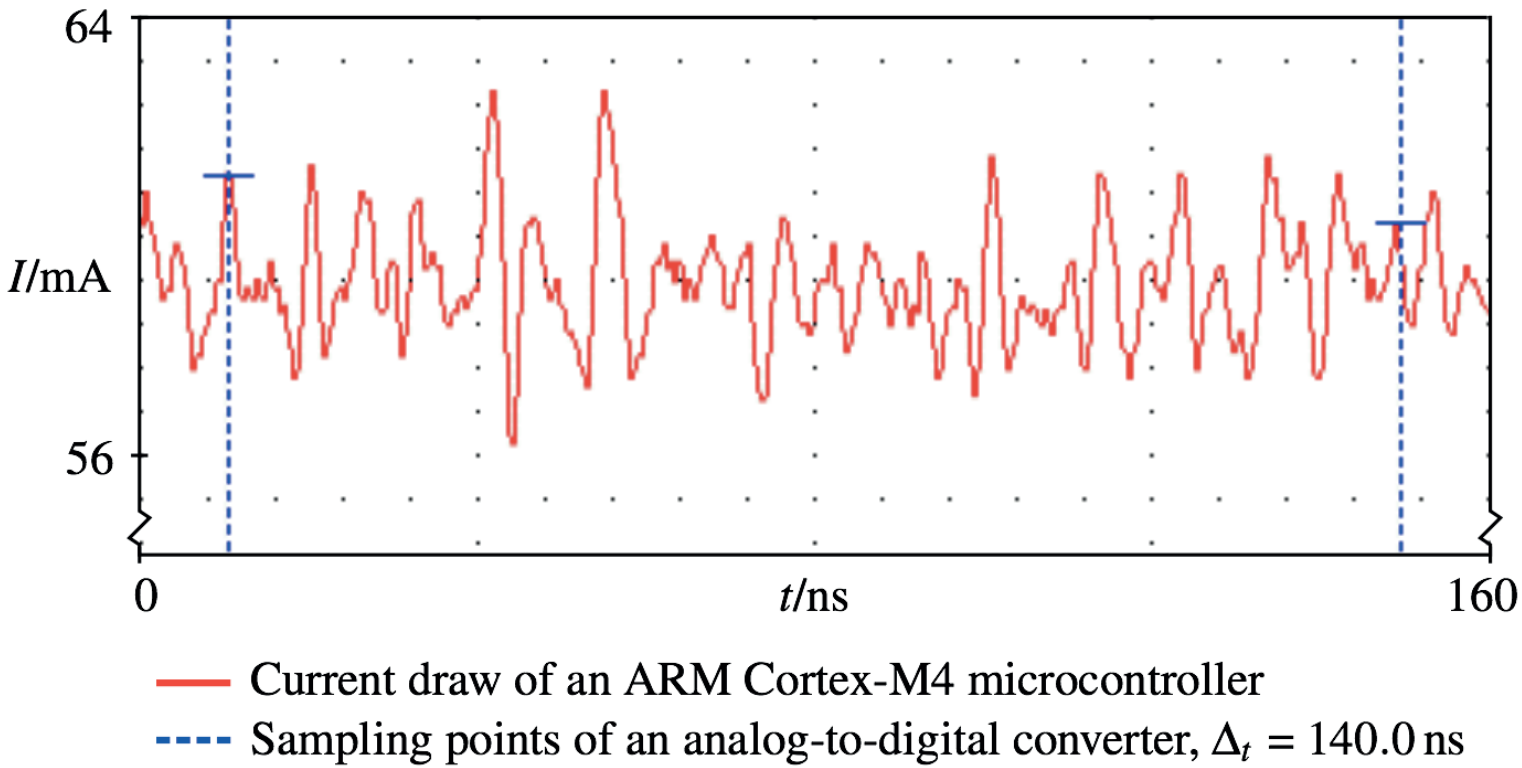


- Limitations
  - control interfaces are cumbersome to use, if available at all
  - difficult to setup, calibrate, and use
  - **sampling may lead to inaccurate measurement results**



Measuring Energy: Sampling Constraints

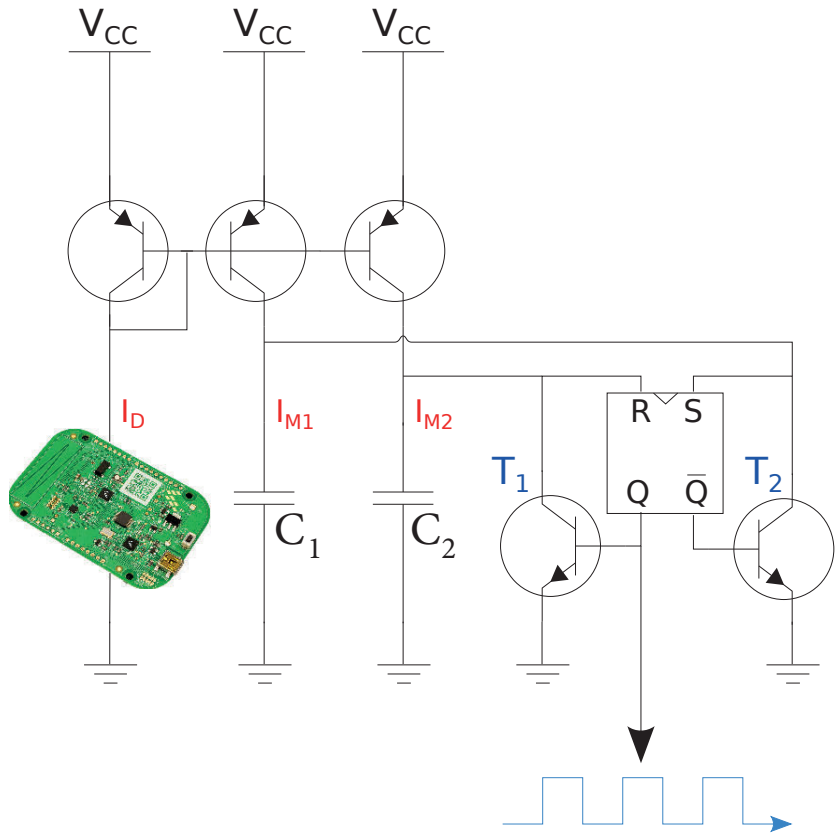
- Measuring Energy with an ADC
  - STMicroelectronics STM32 F4 (ARM Cortex-M4)
  - three ADCs (chained)
  - 7.2 million samples per second (MSPS) $\rightarrow$ sampling time $\sim$ 140 ns





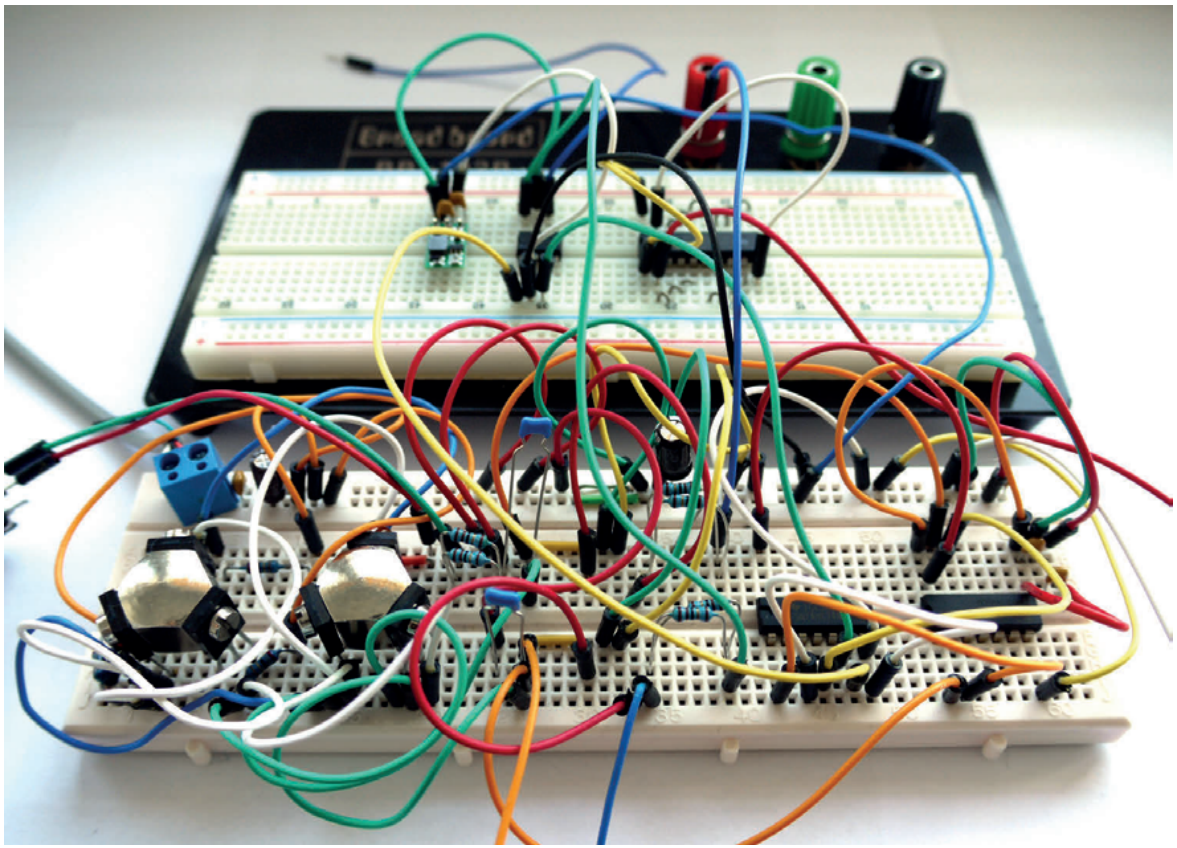
Measuring Energy: PEEK Energy Measurement Device

- Exploit a *current mirror* to measure energy consumption
  - current to frequency conversion
  - two capacitors are being charged and discharged on an alternating basis
  - switching events of the controlling flip-flop are counted



Measuring Energy: PEEK Energy Measurement Device

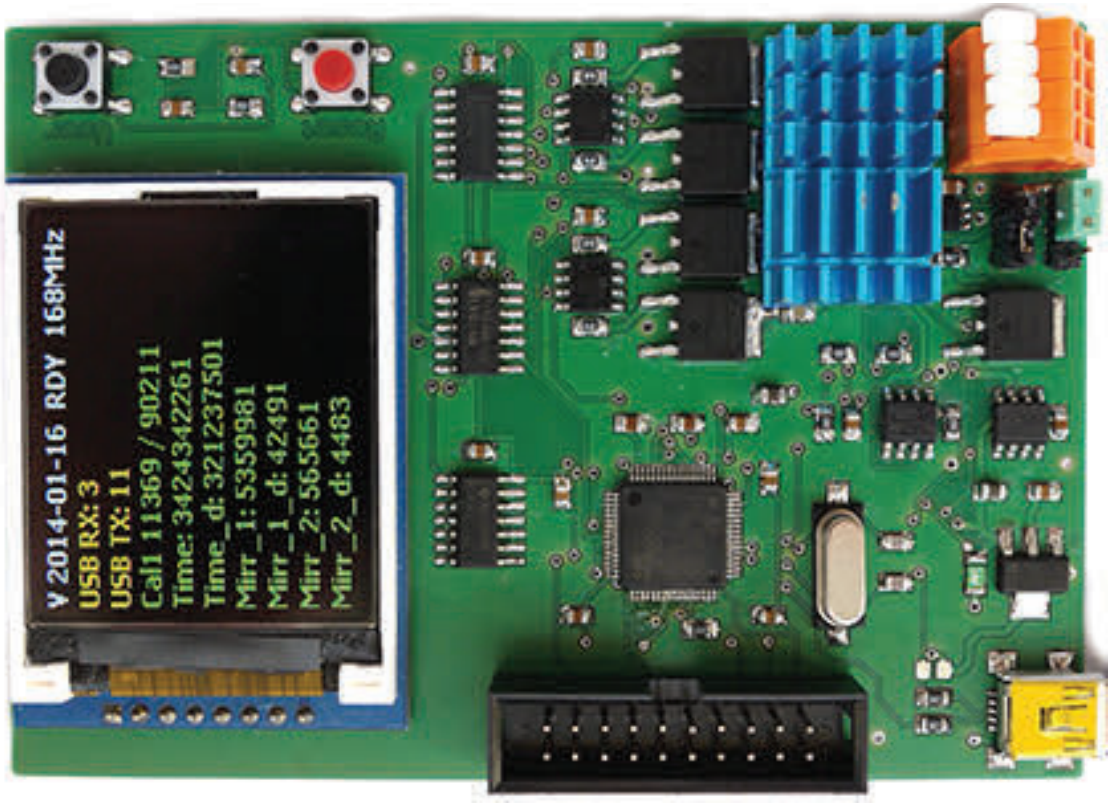
- Exploit a *current mirror* to measure energy consumption
  - current to frequency conversion
  - two capacitors are being charged and discharged on an alternating basis
  - switching events of the controlling flip-flop are counted





Measuring Energy: PEEK Energy Measurement Device

- PEEK energy measurement device
  - controlled by a ARM Cortex-M4 MCU, programmable via USB
  - fully automated energy measurements
  - energy measurement resolution: 0.1 µJ, temporal resolution: 6 ns





Agenda

Energy-Aware Programming
- Status Quo
- Challenges

Proactive Energy-Aware Programming Kit
- Design and System Architecture
- Implementation and Operation
- Energy Optimization Hints
- Electronic Measuring Device

Evaluation
- Scenarios
- Results



Evaluation Scenarios

- Energy Measurements
  - run automated energy measurements using PEEK

  → runtime vs. energy demand

  → compiler impact (GCC vs. Clang)
- Energy Optimization Hints
  - generate energy optimization hints for an application

  → evaluate different run modes

  → automatically reduce energy demand
- Time Savings (→ Paper)
  - compare time savings to existing approaches
- Hands-On Experiment (→ Paper)
  - practical evaluation with a group of students

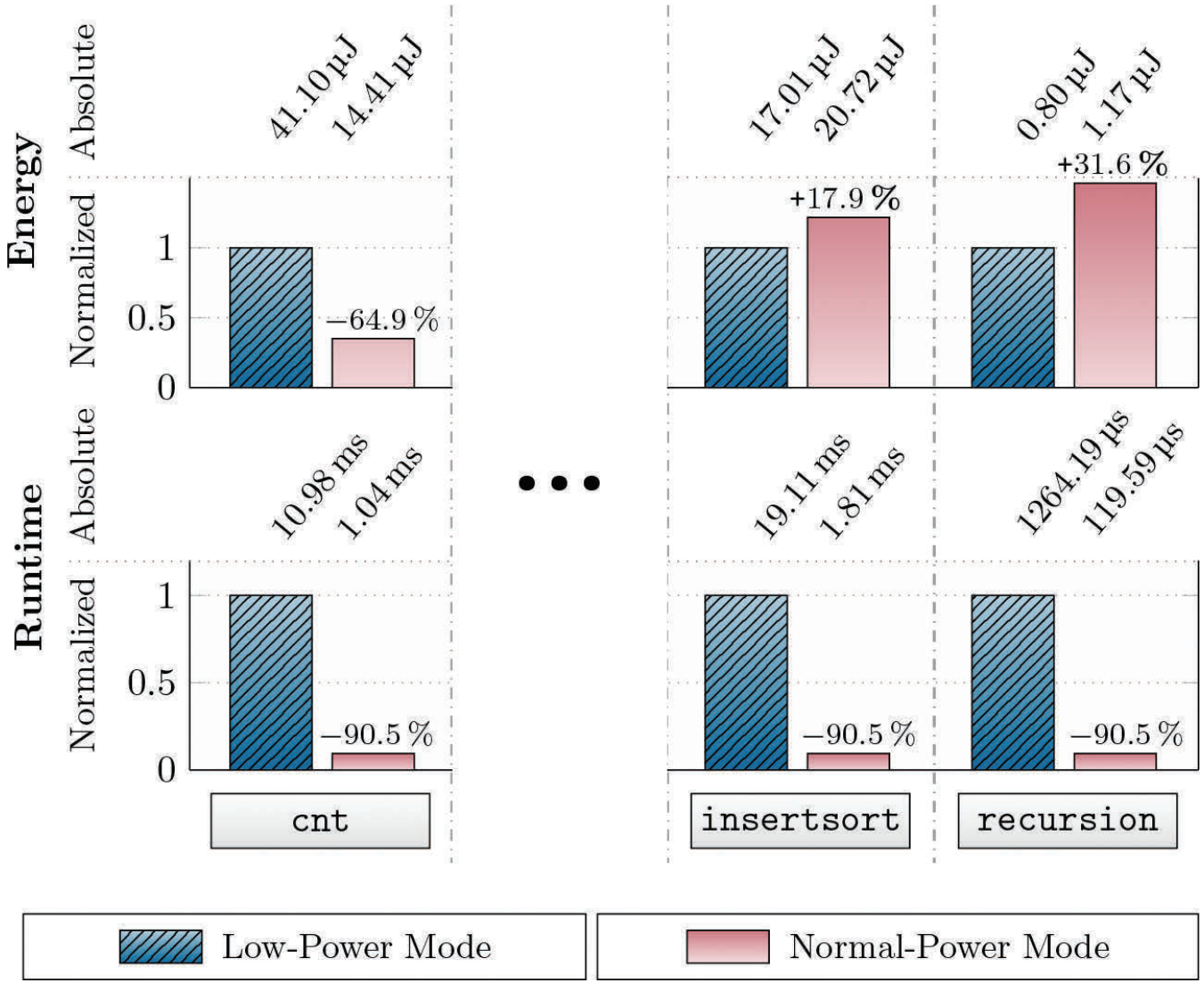


Evaluation Platform

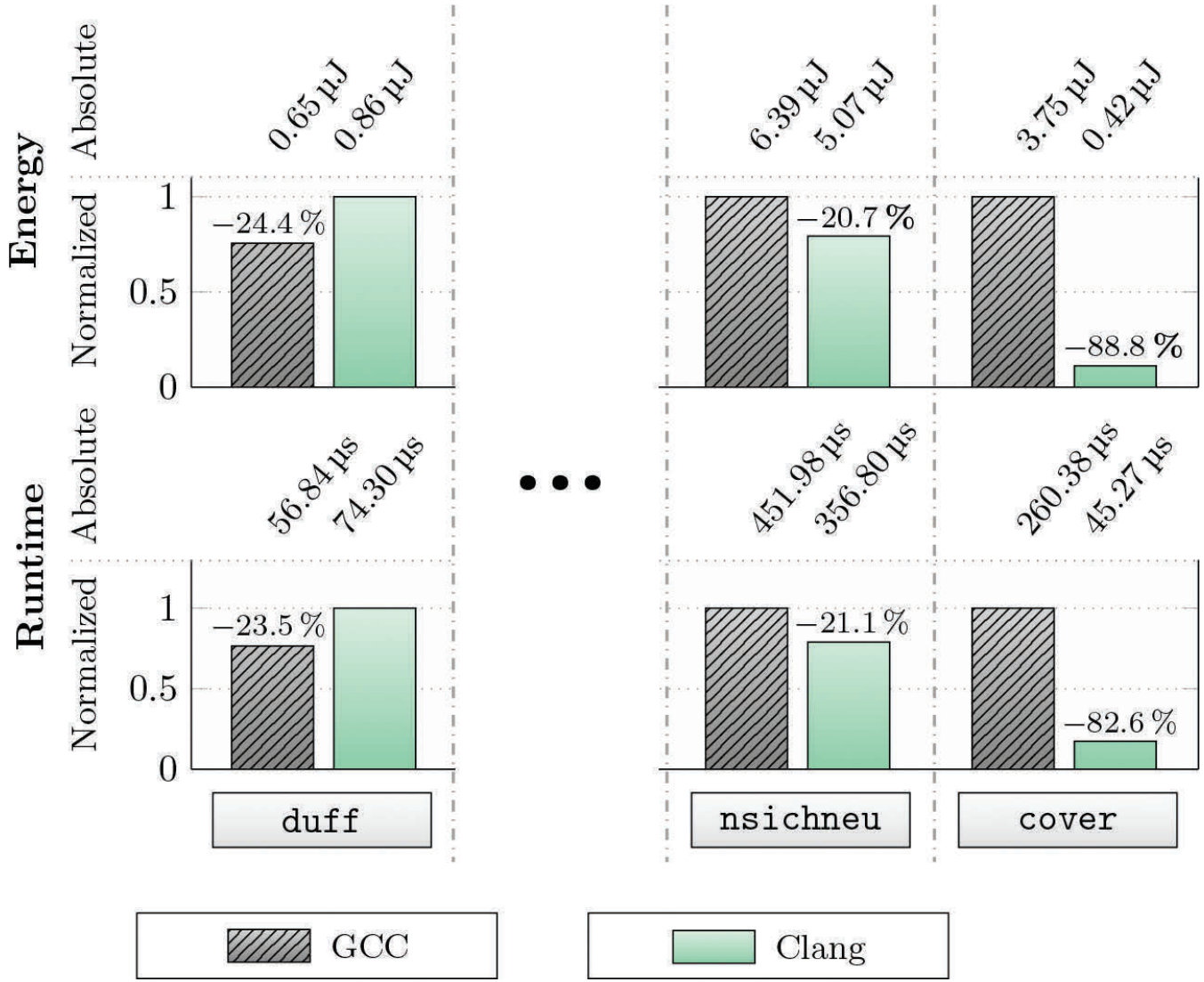
- Freescale Kinetis KL02
- ARM Cortex-M0+ MCU
- 11 different power modes (2 run modes, 9 sleep modes)



Evaluation Results: Power Modes

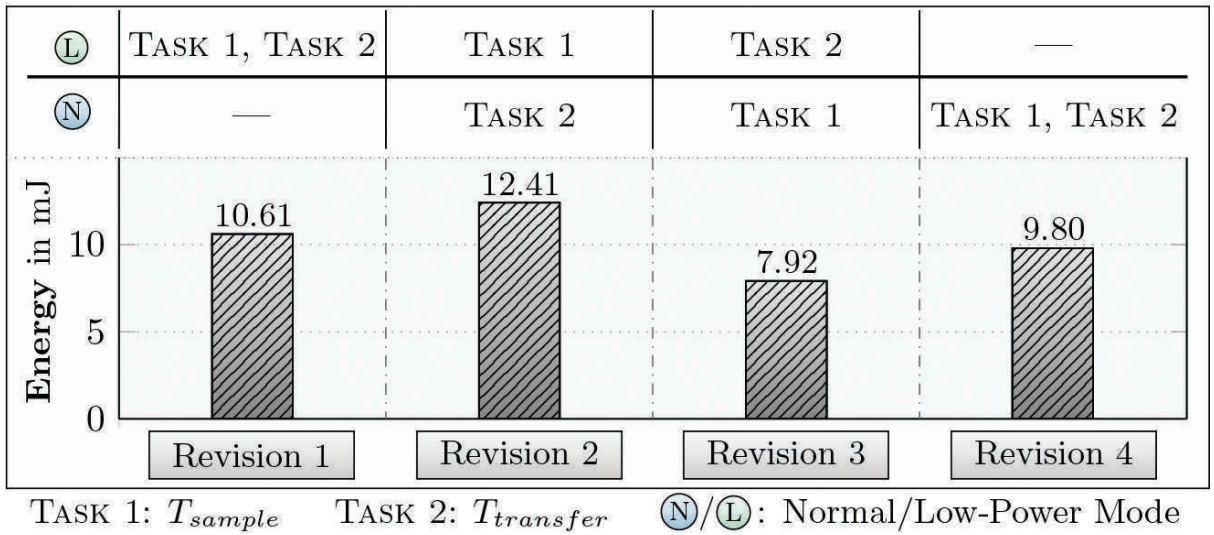


Evaluation Results: Compiler Impact



Evaluation Results: Energy Optimization Hints

- Generate energy optimization hint(s)
  - Two tasks (T_{sample} and $T_{transfer}$)
  - T_{sample}: read sensor data from a triple-axis accelerometer
  - $T_{transfer}$: encrypted data transmission (AES)



- Revision 3 improves energy demand by 25.3 %



Conclusion

- Proactive Energy-Aware Programming with PEEK
  - joint software and hardware infrastructure for energy-aware programming
  - seamlessly integrates into existing development environments
  - assists developers with automated energy analysis
  - generates energy optimization hints to increase energy efficiency



`https://www4.cs.fau.de/Research/PEEK`



Discussion

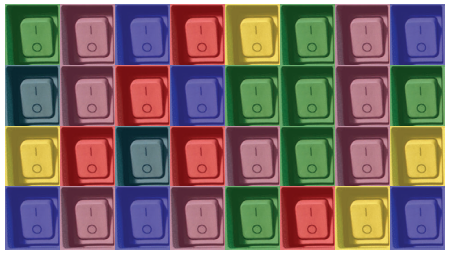












We see PEEK as a key concept for
energy-aware programming in the future.

