Journal of Signal Processing Systems (2021) 93:33–48
https://doi.org/10.1007/s11265-020-01544-z

Fast and Accurate Edge Computing Energy Modeling and DVFS Implementation in GEM5 Using System Call Emulation Mode

Yahya H. Yassin[1] · Magnus Jahre[2] · Per Gunnar Kjeldsberg[1] · Snorre Aunet[1] · Francky Catthoor[3,4]

Received: 27 November 2019 / Revised: 21 April 2020 / Accepted: 5 May 2020 / Published online: 29 May 2020

Abstract
Stringent power budgets in battery powered platforms have led to the development of energy saving techniques such as Dynamic Voltage and Frequency scaling (DVFS). For embedded system designers to be able to ripe the benefits of these techniques, support for efficient design space exploration must be available in system level simulators. The advent of the edge computing paradigm, with power constraints in the mW domain, has rendered this even more essential. Without a fast and accurate methodology for architecture simulation and energy estimation, the benefit of new ideas and solutions cannot be evaluated. In this paper, we propose a non-intrusive application controlled DVFS management implementation in the GEM5 simulator, used with GEM5's system call emulation mode. We also propose a novel architecture independent energy model based on categorization of different measurable workload classes. Our energy model is parametrized and calibrated with power measurements on a SAM4L microcontroller board, containing an ARM Cortex M4 processor. Together with the GEM5 output statistics, the model accurately estimates the total energy consumption of our simulated system. The results from our modified GEM5 simulator are validated with representative signal processing applications. After correction of systematic offset errors, our results deviate with less than 4% compared to measurements from the SAM4L microcontroller. Our contributions in this paper can easily be tailored to other processor models in GEM5 and to future versions of GEM5. It will therefore enable system architects to explore new techniques and compare the improvements relative to existing architectures.

Keywords DVFS · GEM5 · System call emulation · Edge computing · Embedded systems · Energy model · System scenario

1 Introduction

Recent advances in embedded systems and integrated circuit technology have enabled an unprecedented growth in features in mobile signal processing systems [18]. Even if battery technology has also improved significantly, the gains in available energy is much lower than the increase in demand from the more powerful algorithms. Hence, we have a major power management challenge on battery powered platforms, especially given that battery capacity in any case is a finite resource that must be used efficiently. This is particularly so in the edge computing domain [22], e.g., miniaturized surveillance systems and wearable devices, where you can have ultra low power budgets in the order of a few mW [12]. Different energy saving approaches have been proposed as mitigations. A survey by Mittal et al. [18] divides them into four categories; 1) Dynamic Voltage and Frequency (DVFS) and power-aware scheduling techniques, 2) Power Mode Management (PMM) through dynamic use of low power modes, 3) Micro-architectural techniques, leveraging application properties or variation in workload to dynamically reconfigure components of the system to save energy, and 4) Use of accelerator cores, e.g., DSPs, GPUs and FPGAs. In this paper we focus on ultra low power edge computing systems, without the most complex processing units. The two first categories are then most relevant [23].

In order to efficiently exploit these energy saving techniques, custom circuitry is typically required for platform re-configuration, e.g., to dynamically tune the power configurations while a processor core is active. One example system is the SAM4L board [4] where its ATSAM4LC4C microcontroller [5] supports switching

✉ Yahya H. Yassin
yhyassin@gmail.com

Extended author information available on the last page of the article.

between two voltage states at run-time. The same is possible with the STM32F4 microcontroller [24]. Both are examples of the less complex types of processing units typically used for edge computing. What we also see is that the dynamic behavior in advanced signal processing applications is increasing [17], giving increased opportunities to implement finer granularity energy saving approaches. Finer granularity leads to the need for more run-time re-configuration settings, but also for efficient optimization and tradeoff approaches, since the re-configuration typically comes with an overhead both in time and energy. System designers are using computer architecture (CA) simulators, such as GEM5 [8], GEM5-GPU [20], and Multi2Sim [25], to model and analyze new processor designs and energy saving techniques [1]. A CA simulator incorporates detailed performance models and uses them to approximate the behavior of real hardware. They enable designers to experiment with a variety of different configurations at early stages of a system design and to investigate interesting tradeoffs before the prototype stage of the design process [23].

To be able to evaluate the energy related consequences of different architectural choices an energy model is also required. The model needs to be sufficiently accurate to compare and choose between alternative implementations, but also fast enough to allow investigation of a large number of alternative solutions. Detailed transistor level models will typically give accurate results, but be prohibitively slow, while abstract behavioral models can be fast but not sufficiently accurate. For efficient design space exploration, the model must also be easily integrated with the CA simulator and be flexible with respect to use with different technologies and use-cases. Current CA simulators and energy models are to a large extent optimized for complex microprocessors, and less suited for the simpler microcontrollers used in edge computing. The techniques and methodologies presented in this paper will enable designers to exploit the design-space exploration capabilities of CA simulators while adhering to the needs of our focus domain.

For compute bound workloads without known deadlines, active execution at high frequency alternating with deep sleep, known as Race-To-Halt (RTH), typically leads to a larger overall energy reduction than operating at lower frequency and V_{dd} without deep sleep [7]. In deep sub-micron technology nodes, the RTH technique is useful because the leakage and short-circuit current are increasing in magnitude, and in the end dominates. DVFS techniques are on the other hand more suitable for memory bound workloads, and for real time systems with deadlines and long idle times [7]. CA simulators need configurable frequency and voltage scaling functionality in order to further develop these types of energy saving techniques.

GEM5 [8] is a widely used CA simulator, and researchers have proposed different solutions for DVFS management in full system mode [13, 23]. However, the GEM5 full system mode simulates a complete system in an operating system (OS) based environment. This is not representative for many edge computing systems, which often run without an OS. The application itself then has to be in control of the DVFS and power management. Furthermore, for many realistic contexts the simulation time of a full system with an OS is unacceptably long. As an alternative, GEM5 provides a System call Emulation (SE) mode. This is an application-level simulation where it is only necessary to specify the statically linked binary file that is going to be simulated. This makes the simulator significantly faster to execute and also more applicable for edge computing design exploration.

The McPAT energy modeling framework [16] is often used together with GEM5 output statistics to estimate the overall energy consumption. Like the GEM5 full system mode, it focuses mainly on high performance processors and can be overly complex for edge computing systems. It also requires a detailed internal architecture model not necessarily available to the system designer.

As illustrated in Fig. 1, the main contribution of this paper is 1) a non-intrusive application controlled DVFS management implementation in GEM5 SE mode, and 2) an architecture independent energy model based on classification of different measurable workload classes. The energy model is parametrized by running a calibration application once on real hardware. The contributions enable efficient use of CA simulators in design exploration for energy optimization of ultra-low power edge computing systems.

To our knowledge, no published DVFS implementations exist in GEM5 SE mode or for application controlled DVFS in GEM5. A few implementations are available in full system mode, based on the Alpha processor [13], or implemented as a component responsible for setting clock frequencies according to OS policies [23]. Our DVFS mechanism is controlled by the application through custom pseudo instructions implemented in the GEM5 simulation kernel based on the ARM processor model. These custom pseudo instructions communicate with Python configuration scripts at the GEM5 user-level, where the DVFS controller is implemented.

Our architecture independent energy modeling framework uses known energy and power formulas available in literature [18, 21], split into subparts for, e.g., dynamic and static power using realistic device parameters. In our model,

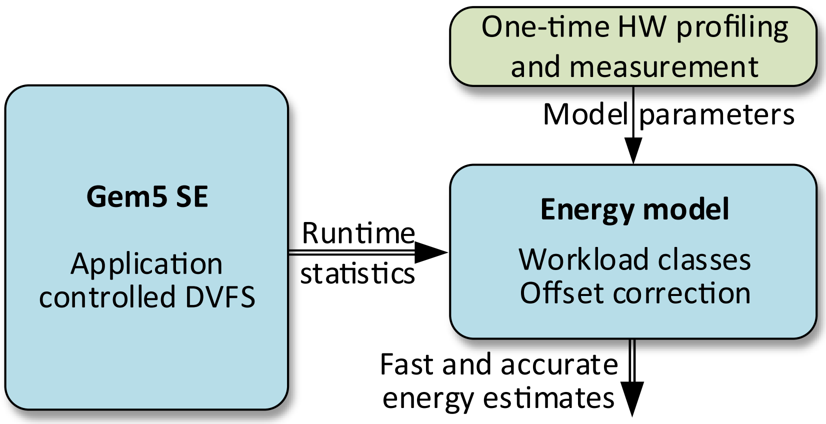

One-time HW profiling and measurement
Model parameters
Gem5 SE
Application controlled DVFS
Runtime statistics
Energy model
Workload classes
Offset correction
Fast and accurate energy estimates

Figure 1 Illustration of paper contribution, model setup and execution.

we separate different types of workloads into measurable classes, which are parametrized in our power formulas. The device parameters are calibrated using measurements from real HW in order to estimate the power consumption of different workloads accurately, resulting in a fast and accurate energy model. These calibrated parameters are stored in a simple XML interface which is read by our energy model script. In this paper, we calibrate our parameters with power measurements from the SAM4L microcontroller [30]. Other architectures can be modeled similarly by exchanging our calibrated values with new ones in our XML interface. The only information needed by the energy model script from the GEM5 simulator are the workload related statistics. Together this makes the setup fully reusable in a user-friendly manner for other processor platforms.

Section 2 presents an overview of CA simulators and existing DVFS implementations in GEM5. It also covers related work on energy models for use in CA simulators. Our DVFS implementation in GEM5 is described in Section 3, and the energy model is presented in Section 4. In Section 5 we introduce our experimental setup before we present and discuss our results in Sections 6 and 7, respectively. Finally, our conclusions are presented in Section 8.

2 Related Work

A number of CA simulators exist today, e.g., GEM5 [8], SimpleScalar [6], Sniper [10], and Multi2Sim [25]. Multi2Sim only supports out-of order execution and is mainly intended for CPU-GPU computing. SimpleScalar and Sniper are application-level simulators as opposed to GEM5, which is a full system simulator. The benefit of an application-level simulator is the ability to run only target applications instead of a full fledged target OS. The GEM5 simulator supports application-level simulation using the SE mode, the model of choice in our work as motivated in the introduction. Many of the techniques we present in this paper are agnostic to the choice of CA simulator, however.

Spiliopoulos et al. [23] extends the GEM5 simulator to support full-system DVFS modeling. They extend the GEM5 clock and voltage domains and use them with a kernel-level DVFS controller containing configurable memory-mapped registers that interacts with the software. Spiliopoulos et al. [23] rely on McPAT [16] for the power models and they extend it with a set of coefficients that describes the GEM5 modeled system with DVFS.

Haririan et al. [13] presents non-intrusive full system DVFS emulation in GEM5 based on the DEC Alpha processor model. They implement DVFS relevant performance monitors in the full system GEM5 model and transfer their values to the configuration script at user-level. The script controls DVFS based on custom instructions provided by the simulation kernel. The status of the main application is monitored by a concurrent utility application running periodically. This utility application communicates with the configuration scripts and compares its performance counters with previously measured values. The comparison results are used to trigger a DVFS switch accordingly. Their solution does not allow the application to have direct DVFS control of the platform.

Li et al. [16] introduce McPAT, an integrated power, area and timing modeling framework that supports comprehensive design space exploration for multi-core and many-core processor configurations. This framework includes models for the fundamental components of different types of processor cores. With a flexible XML interface, McPAT can be interfaced with many performance simulators. McPAT focuses mainly on complex high performance processors, and all parts of the system architecture must be modeled properly in order to extract the correct energy results. The architecture based energy model has also been seen to incur estimation errors that are difficult for designers to detect and compensate for [19].

A framework for analyzing and optimizing microprocessor power dissipation at the architectural level, WATTCH, is presented by Brooks et al. [9]. It is a fast and high accuracy framework tailored to work together with SimpleScalar [6]. Similar to McPAT, WATTCH requires the internal components of the system architecture to be modeled in detail in order to benefit from the energy model.

Compared to previous work, our DVFS controller is implemented in GEM5 SE mode, using a generic technique which can be applied to other simulators as well. The DVFS implementation is application controlled and focuses on simulating low power embedded systems with less complex architecture than high performance processors. Compared to energy modeling frameworks typically used in CA simulators, our energy model is architecture independent in the sense that it does not need details about internal components in the processing unit. Instead it uses energy and power formulas parametrized using measurements on real hardware. Coupled with statistics from GEM5 this gives fast and accurate energy estimates for microcontroller based edge computing systems.

3 Contribution I: GEM5 DVFS in SE Mode

In this section, we present our first contribution in this paper; SE mode DVFS implementation in GEM5. Figure 2 shows details of our GEM5 implementation. We will first give an overview of the functionality before we go into implementation details in the following subsections. When the application running on the simulated processor model (lower right corner of Fig. 2) decides that it wants to change DVFS state, it executes a custom instruction that calls the DVFS controller. The different clock and power settings are instantiated as separate CPU models and the DVFS controller activates the CPU switch mechanism, which then loads the correct model into each CPU core (upper left corner of Fig. 2). Activity statistics for cores and memories are now gathered by the simulator for the time period the selected DVFS state is running, and then dumped to text files for later post-processing. When the application stops executing, we thus have separate activity statistics for each DVFS state, which can be used by Python scripts to calculate the total energy consumption (right hand side of Fig. 2).

From a more formal perspective, the GEM5 simulator is comprised of C++ processor models at the simulator

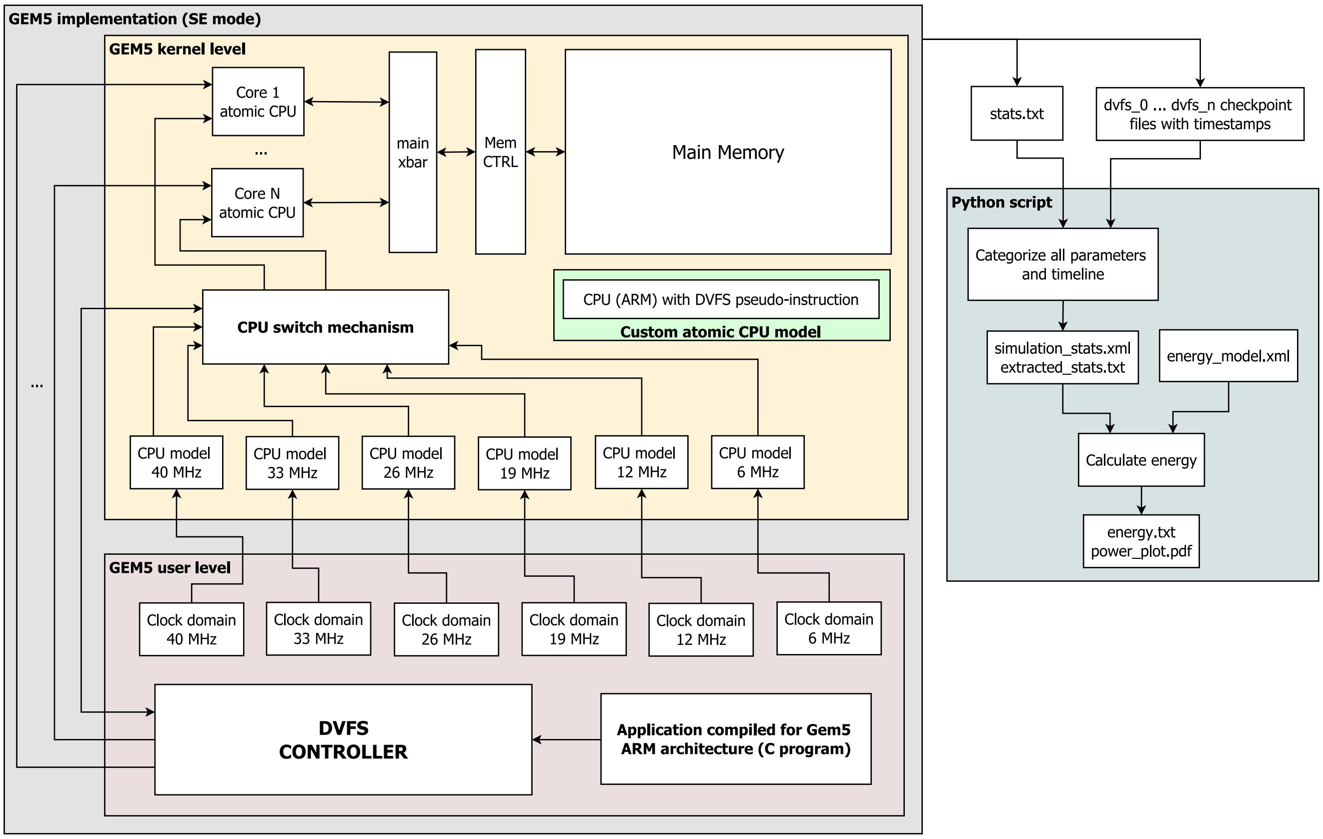


Figure 2 GEM5 DVFS simulator flow with Python scripted energy model.

kernel level (upper part of Fig. 2), which interacts with Python, Ruby, and Swig configuration scripts at the user level (lower part of Fig. 2). The kernel-level represent the physical system components through C++ objects, called simObjects. These simObjects are exposed to Python, such that they can be controlled by the user level configuration scripts. The user level thus represents the control part of the simulator, from which the different kernel level simObjects are configured and controlled through Python. The user level also links the compiled user application to the simObjects and runs the simulator.

We take advantage of GEM5's pseudo instruction functionality in the simulator kernel and let the application directly control the DVFS switch through a custom pseudo instruction we have implemented in the ARM processor model. This instruction interacts with the user level configuration scripts and switches the CPU model at run-time. A summary of our main user and kernel level modifications are listed in Table 1. Table 2 compares our generalized modifications to state-of-the-art DVFS implementations in GEM5. In the following sub-sections, we will describe our GEM5 DVFS simulator flow in more detail.

3.1 GEM5 Kernel Level Modifications

Our DVFS mechanism is implemented in a generic way so that it can be tailored to work with any in-order CPU model. Only CPU specific mechanisms, such as our custom DVFS instruction, must be ported to the target instruction set architecture, provided there is space available for additional instructions. To demonstrate our methodology, we apply our changes to the ARM processor model.

The DVFS instruction, *gem5Dvfs* shown in Listing 1, is implemented in an unused location within the GEM5 model of the ARM instruction set (Row 3 of Table 1). This instruction combines available registers for storage of DVFS state, delay and period variables of our DVFS function. The functionality of the DVFS instruction is modeled in the pseudo_inst files (Row 2 of Table 1). In Fig. 2 we find this custom atomic CPU model with DVFS instruction depicted below the Main Memory.

The GEM5 version we are using does not have native support for multiple clock domains per CPU model in SE mode. Consequently, we have implemented one custom CPU model for each performance level. These custom CPU models are copies of the custom atomic CPU model with modified file and variable names (Row 1 of Table 1).

At the end of *gem5Dvfs* in Listing 1 the simulation exits its loop with "gem5_DVFS" as the cause of exit. This message is then handled by the user level configuration scripts to switch the current CPU model with another model based on the value of the *dvfs_state* variable from the application, before the simulation continues. The *when* and *repeat* variables in Listing 1 are optional configurations that can delay a DVFS switch with a number of ns or trigger a periodic DVFS switch depending on what is required by the application. In this paper, we set these parameters to zero because the energy delay is handled by our energy model described in Section 4. The *gem5Dvfs* instruction is connected to a new *gem5_dvfs_inst* function in the GEM5 utilities, which are included when compiling the application for the target platform in GEM5 (Row 4 of Table 1). The utility function makes the DVFS instruction available to the application.

In order to control the DVFS switches occurring while the application is running we implemented a copy of the GEM5 checkpoint function to create DVFS checkpoints with time stamps each time the DVFS instruction is used (Row 5 of Table 1).

Table 1 GEM5 modifications.

Row	File path	Type of modification
Kernel level modifications		
1	In src/cpu/:	
	scpu1800mv40mhz	
	scpu1650mv33mhz	Copies of src/cpu/simple
	scpu1500mv26mhz	with modified variable and
	scpu1350mv19mhz	function names
	scpu1200mv12mhz	
	scpu1050mv06mhz	
2	In src/sim/:	
	pseudo_inst.hh	Implementation of the
	pseudo_inst.cc	DVFS instruction function
3	In src/arch/arm/isa/:	
	operands.isa	Extension of the ARM
	insts/m5ops.isa	instruction set with
	formats/m5ops.isa	one DVFS instruction
4	In util/m5:	
	m5op_arm.S	Connecting the pseudo
	m5op.h	DVFS instruction to
	m5ops.h	the configuration scripts
5	In src/python/m5/:	DVFS checkpoint function
	simulate.py	used by the DVFS instruction
User level modifications		
6	In config/examples/:	Declaration of each clock
	se.py	domain for DVFS
7	In config/common/:	Connecting CPU models to
	Simulation.py	the system, adding the
	Options.py	DVFS options flag and
	CpuConfig.py	controller functionality

Table 2 Comparison to state-of-the-art DVFS implementations in GEM5.

State-of-the-art	Our implementation
Works only in full system mode	Implemented for SE mode
Custom versions of simple, detailed and timing CPU models used with DVFS	Copies of the simple atomic CPU model with modified variable and function names
Implementation of the DVFS pseudo-instructions used with performance monitors	Implementation of application controlled DVFS pseudo-instruction used together with GEM5's checkpoint mechanism
OS controlled or system monitor based DVFS policies through either kernel level mechanisms or by GEM5's *switchCpus* mechanism	DVFS policy triggered by a DVFS instruction from the application together with GEM5's *switchCpus* mechanism

The performance statistics are stored in the stats.txt output from GEM5 where each DVFS performance level is collected in separate simulation rounds. Before each DVFS instruction in the user-level we first dump the gathered performance statistics to a file and reset the statistics counters immediately after each DVFS instruction. This procedure allows us to force the simulator to start a new simulation round, i.e., a new collection of performance statistics, after each DVFS instruction. Each simulation round in the stats.txt file in GEM5 lists the statistics for all CPU models instantiated in the configuration script. In order to know which CPU was active in each simulation round, we also introduced a new statistics variable, *dvfsCPUActive*, in each of our custom CPU models. This variable is used in a Python script in a post-processing step to extract the performance history from the correct CPU (right hand side of Fig. 2).

3.2 GEM5 User Level Modifications

Our DVFS controller is implemented in the GEM5 Python configuration scripts at the user level. An advantage of having the DVFS controller in the configuration script is that the designer can change and experiment with different DVFS switching policies without recompiling the simulation kernel. Compilation of the GEM5 kernel is only required if new frequencies or CPU's are added or modified.

In GEM5's SE mode user level script, se.py, we have instantiated one CPU clock domain for each of our custom CPU models and assigned them their corresponding frequency (Row 6 of Table 1). As visualized at the boundary between the user level and kernel level parts of Fig. 2 these user level domains are then connected with their corresponding kernel level CPU models using the mechanism in the common user level Simulation.py script (Row 7 of Table 1). Our working version of the GEM5 simulator did not permit a frequency change after the processor was initialized. However, it allowed changing the processor model through the GEM5 *switchCpus* function, which worked when we assigned separate clock domains for each CPU model. Method switchCpus drains the simulation and switches out the old CPU. When the simulation is drained, all the components are notified to come to a consistent state that can safely be serialized, in a similar way checkpoints are written to file. The old CPU continues to run until it has committed all instructions still residing in the pipeline. When this process is finished the CPU models are exchanged with the help of built-in functions in the gem5 simulator, and all statistics from the old CPU are written to the stats.txt file. The simulation then continues with the new CPU model. It is thus possible to run a fully functional simulation in order to achieve accurate energy estimates. The DVFS functionality is only activated when our "--gem5-dvfs" option is enabled at the start of the simulation. We modified the GEM5 run function used by se.py in Simulation.py to include our DVFS controller as shown in Listing 2. We implemented the DVFS controller

Listing 1 DVFS pseudo instruction.

```
void gem5Dvfs(ThreadContext *tc, uint64_t
     dvfs_state, Tick delay, Tick period){
 if (!tc->getCpuPtr()->params()->
      do_checkpoint_insts)
  return;

 // 1 Tick = 1 ps. Delay is in ns
 Tick when = curTick() + delay *
                         SimClock::Int::ns;
 Tick repeat = period * SimClock::Int::ns;
 exitSimLoop("gem5_DVFS", dvfs_state, when,
           repeat);
}
```

Listing 2 Modification in the run function.

```
if options.repeat_switch and maxtick > options.
    repeat_switch:
 exit_event = repeatSwitch(testsys,
     repeat_switch_cpu_list, maxtick, options.
     repeat_switch)
else:
 if options.gem5_dvfs:
  exit_event = gem5_dvfs_switch(options, maxtick,
      cptdir, testsys)
 else:
  exit_event = benchCheckpoints(options, maxtick,
      cptdir)
```

itself in the *gem5_dvfs_switch* function as shown in Listing 3.

The functionality of our DVFS controller is implemented in our Platform Adaption Manager (PAM) function called *PAM_select*. This function selects which CPU model to choose based on the application request given by the *dvfs_state* variable and triggers a frequency change from our *dfs* function. The *PAM_select* and *dfs* functions are shown in Listings 4 and 5.

Our current DVFS controller handles homogeneous multi-core switching, i.e., all active and parallel processor cores of the same kind change their models simultaneously when the application executes the DVFS instruction. As shown in Listings 5 and 6, our *dfs* function generates a CPU switch list. This switch list is used by the GEM5 *switchCpus* function to switch the active CPU with another model. An extension of our *dfs* function to support different models to be simultaneously active is considered future work and requires heterogeneous multi-core support in GEM5.

3.3 Usage of DVFS in the Application

Listing 7 shows the DVFS function, *dvfs_scenario*, which must be included in the application for DVFS control. In our tests we also trigger a DVFS switch at the beginning and end of the application in order to isolate the power

Listing 3 DVFS Controller.

```
def gem5_dvfs_switch(options, maxtick, dvfsdir,
    testsys):
 switch_cpu_list = []
 np = options.num_cpus

 exit_event = m5.simulate(maxtick - m5.curTick())
 exit_cause = exit_event.getCause()

 num_checkpoints = 1
 max_checkpoints = options.max_checkpoints

 # Initial event needs to be handled
 # before the while loop
 if "gem5_DVFS" in exit_cause:
  exit_cause_base = exit_cause
  dvfs_state = exit_event.getCode()

  switch_cpu_list = PAM_select(dvfs_state,
                               testsys, np)
  if len(switch_cpu_list) > 0:
   m5.switchCpus(testsys, switch_cpu_list)
 else:
  exit_cause_base = exit_cause

 while exit_cause_base == "gem5_DVFS":
  m5.dvfs_checkpoint(joinpath(dvfsdir, "dvfs_"+
                   str(num_checkpoints)+".%d"))
  num_checkpoints += 1

  exit_event = m5.simulate(maxtick - m5.curTick())
  exit_cause = exit_event.getCause()
  if "gem5_DVFS" in exit_cause:
   exit_cause_base = exit_cause
   dvfs_state = exit_event.getCode()

   switch_cpu_list = PAM_select(dvfs_state,
                                testsys, np)
   if len(switch_cpu_list) > 0:
    m5.switchCpus(testsys, switch_cpu_list)
  else:
    exit_cause_base = exit_cause

 return exit_event
```

Listing 4 PAM_select function.

```
def PAM_select(state, testsys, np):
 if state == DVFS_STATES.MHz_40_V_1_80:
  return dfs(testsys,np,"40")
 # ...
 elif state == DVFS_STATES.MHz_06_V_1_05:
  return dfs(testsys,np,"06")
 else:
  return dfs(testsys,np,"40")
```

estimation of the application. *dvfs_scenario* initially dumps the performance statistics of the current performance level, before switching the CPU model through our DVFS function. Immediately after the switch all performance statistics are reset, and the performance statistics for the new CPU model are then collected in a separate simulation round in stats.txt.

Compared to state-of-the-art solutions, we introduce direct application control of the voltage and frequency mechanisms in the simulated platform. Our implemented mechanisms are generic because we mainly add extensions to architecture independent parts of the simulator. Only the implementation of the *gem5Dvfs* instruction needs to be ported to an available location in the instruction set of other architectures of interest.

3.4 Limitations of Our DVFS Mechanisms

The current implementation of our DVFS mechanisms is limited to the simple atomic in-order CPU model in GEM5. Our mechanisms focus on CPU voltage and frequency scaling, and do not support memory hierarchy voltage and frequency scaling. For our focus on microcontroller based edge computing, this is not a limitation. However, doing the following modifications will enable using our technique for other domains as well. To support DVFS in the memory hierarchy, mechanisms similar to the *switchCpus* mechanism is required for the memory hierarchy in GEM5 to enable memory level DVFS. Similarly, a heterogeneous multi-core support in GEM5 is required before our DVFS mechanisms can be extended to support heterogeneous multi-core DVFS. These extensions are considered out of the scope of this paper.

4 Contribution II: Energy Model

The second contribution in this paper is our energy model, based on equations divided into subparts explicitly exposing essential device parameters. These are parametrized through power measurements from the SAM4L microcontroller with different workloads (arithmetic and memory) and power modes (active and sleep). The theory and assumptions behind our energy model and power consumption formulas are presented in Section 4.1. How we combine the GEM5 statistics with SAM4L power measurements through our

Listing 5 dfs function.

```
def dfs(testsys, np, new_frequency):
 switch_cpu_list = []

 for j in xrange(np):
  # Select the first (and only) active CPU type
  if not testsys.cpu[j].switchedOut():
   old_cpu = "cpu"
   switch_cpu_list = generate_switch_list(testsys,
                old_cpu, new_frequency, j, np)
  elif not testsys.switch_cpus_40mhz[0].
                               switchedOut():
   if new_frequency is "40":
    switch_cpu_list = []
   else:
    old_cpu = "switch_cpus_40mhz"
    switch_cpu_list = generate_switch_list(
        testsys, old_cpu, new_frequency, j, np)
  # ...
  elif not testsys.switch_cpus_06mhz[j].
      switchedOut():
   if new_frequency is "06":
    switch_cpu_list = []
   else:
    old_cpu = "switch_cpus_06mhz"
    switch_cpu_list = generate_switch_list(
        testsys, old_cpu, new_frequency, j, np)
  else:
   fatal('All CPUs are switched out!')
 return switch_cpu_list
```

Listing 6 generate_switch_list function.

```
def generate_switch_list(testsys, old_cpu,
    new_frequency, j, np):
 cmd = "testsys.switch_cpus_" + new_frequency +
                                   "mhz[j].workload"
 exec("%s = %s" % (cmd,"testsys." + old_cpu +
                                  "[j].workload"))
 cmd = "[(testsys." + old_cpu + "[i], testsys.
     switch_cpus_" + new_frequency + "mhz[i])
                           for i in xrange(np)]"
 exec("%s = %s" % ("switch_cpu_list",cmd))
 return switch_cpu_list
```

energy modeling scripts is presented in Section 4.2. Section 4.3 describes how we apply offset error corrections in GEM5.

4.1 Estimation of the Power Consumption

Our model for power consumption is based on Eqs. 1 and 2 [21].

$$P_{\text{Total}} = P_{\text{Static}} + P_{\text{Dynamic}} \tag{1}$$

$$P_{\text{Dynamic}} = C_E f V_{\text{dd}}^2, \text{ where } C_E = \alpha C \tag{2}$$

In Eq. 2, C_E is the effective load capacitance, V_{dd} is the supply voltage, f is the clock frequency, α is the activity factor (between 0 and 1), and C is the load capacitance. The static power can be modeled as shown in Eq. 3 [11], where $P_{\text{Short-circuit}}$ is the short-circuit (direct path) power and I_{Leak} is the leakage current. Even though gate- and junction-leakage currents exacerbate I_{Leak}, it is dominated by the sub-threshold current I_{ds}, giving Eqs. 4 [26] and 5 [3], respectively. In Eq. 4, β is the gain factor of a MOS transistor ($\mu A/V^2$), τ is the rise or fall time of a signal, f is the device frequency, V_{Th} is the threshold voltage, and V_{dd} is the supply voltage. According to Alioto [2], the overall energy per clock cycle of a VLSI system consists mainly of the dynamic and the leakage energy. The short-circuit current energy contribution is usually negligible, due to the exponential MOS I-V characteristics. Rabaey et al. [21] similarly shows that the short-circuit power in Eq. 4 can be ignored for well designed circuits, because it is normally less than 10% of the dynamic power dissipation, except for slow input signals (large τ). We hence assume that $P_{\text{Leak}} >> P_{\text{Short-circuit}}$ and leave out the short circuit power consumption from our model.

$$P_{\text{Static}} = P_{\text{Short-circuit}} + I_{\text{Leak}} \cdot V_{\text{dd}} \tag{3}$$

$$P_{\text{Short-circuit}} = \frac{\beta}{12} \tau f (V_{\text{dd}} - 2V_{\text{Th}})^3 \tag{4}$$

$$I_{\text{Leak}} = I_{\text{ds}} = I_0 e^{(1-\kappa)\frac{V_{\text{bs}}}{V_T}} e^{\kappa \frac{V_{\text{gs}}}{V_T}} (1 - e^{-\frac{V_{\text{ds}}}{V_T}} + \frac{V_{\text{ds}}}{V_0}) \tag{5}$$

In Eq. 5, V_{bs} is the substrate-to-source potential, V_{gs} is the gate-to-source potential, V_{ds} is the drain-to-source potential, $V_T = k \cdot T/q$ is the thermal voltage (26 mV at room temperature [3]), I_0 is the zero-bias current, V_0 is the early voltage (proportional to channel length), and κ is the effectiveness of the gate potential in controlling the channel current. Assuming that our target device operates in saturation, which is the case for $V_{\text{ds}} > 3 \cdot V_T$ [3], we can ignore the body effect. Equation 6 assumes $V_{gs} = 0$, and results in a simplified formula for the leak current when $V_{\text{ds}} >> V_{\text{Th}}$, which is the case for our target platform architecture.

$$I_{\text{Leak}} = I_0 + \frac{I_0}{V_0} V_{\text{ds}} \tag{6}$$

In our model we assume that the drain-to-source potential (V_{ds}) is equivalent to the supply voltage (V_{dd}). Hence, our model for total power consumption valid within our target domain, is as shown in Eq. 7.

$$P_{\text{Total}} = I_0 V_{\text{dd}} + \frac{I_0}{V_0} V_{\text{dd}}^2 + C_E f V_{\text{dd}}^2 \tag{7}$$

In order to estimate the values of C_E, I_0 and V_0, we measure the total power consumption of our SAM4L microcontroller at 40 MHz and 20 MHz with $V_{\text{dd}} = 1.8$ V, and similarly at 12 MHz with $V_{\text{dd}} = 1.2$ V. The measurements were done directly on the SAM4L board using an oscilloscope and ampere meter. Details on a similar experimental setup can be found in [29]. With these three measurements we use Eq. 7 to find the values of C_E, I_0 and V_0. We also observe that these values differ when the SAM4L is executing different types of workloads. When the application is reading and writing to memory, the measured C_E value is, as expected, higher than when the microcontroller is only executing arithmetic instructions (a memory access has higher load capacitance than the ALU). We made two separate measurements on

Listing 7 Inline DVFS instruction implementation.

```
inline void* dvfs_scenario( int state ){
  m5_dump_stats(0,0); // Split the statistics
  gem5_dvfs_inst(state,0,0); // DVFS checkpoint
  m5_reset_stats(0,0); // Start new statistics
}
```

our SAM4L microcontroller; one with a computational workload, and one with a memory access workload. From these measurements, we calculate the C_E, I_0 and V_0 values shown in Table 3.

Our energy model takes into account energy required for DVFS switching and also includes use of sleep modes in addition to DVFS. The SAM4L microcontroller has multiple sleep modes, but in this paper we only consider the deepest sleep13 mode. It is, however, easy to extend it to take into account others as well. The sleep13 power consumption and the energy consumption of the DVFS scale down and scale up overhead (E_{S_D} and E_{S_U}) were again measured directly on the SAM4L board with the oscilloscope and ampere meter. The oscilloscope was used to find the time between two pin pull-up signals from the microcontroller; one before and one after a DVFS switch. The ampere meter was used to measure the current consumption of the chip during this time, and to measure the deep-sleep power consumption. Table 4 gives the results from the measurements.

4.2 GEM5 Statistics and Energy Modeling Scripts

The GEM5 simulator outputs a stat.txt file containing statistics such as the number of instructions executed, the type of instructions executed, the number of memory accesses (reads and writes), and the number of simulated cycles at a given frequency. In our implementation of the DVFS mechanism, the simulator also outputs checkpoint files for each executed DVFS instruction. These checkpoint files contain an ID number and a time stamp, which is used together with the stats.txt file to map a time line of simulation events when using DVFS. Our DVFS mechanism splits the statistics in the stat.txt files into different chunks in order to separate the activity occurring at different frequencies into simulation sets. Together with our power model and parameters from Tables 3 and 4, we calculate the total energy consumption for the GEM5 simulation as shown in Eq. 8.

$$E_{\text{Total}} = E_{S_U} \cdot S_U + E_{S_D} \cdot S_D + \sum_{i=E0}^{En} E_{\text{Sim}_i} \tag{8}$$

In Eq. 8, $E0$, $E1$, ..., En are the energy consumptions of all the simulation sets derived from our model of the power consumption, Eq. 7, and the output stat.txt file from

Table 3 Variables derived from measurements for use in work load based total power calculation.

Type of workload	C_E (nF)	V_0 (V)	I_0 (mA)
Computation	0.19	–0.56	–0.84
Memory access	0.21	–0.49	–0.75

Table 4 Measured sleep mode related power and DVFS switching energy values.

$P_{sleep13}$ (μW)	E_{S_D} (nJ)	E_{S_U} (nJ)
51	28	62

GEM5. This is explained further later in this section. E_{S_D} and E_{S_U} are the energy scale down and up overhead for DVFS, respectively. S_U and S_D are the number of scale up and scale down events (i.e., DVFS switches) occurring throughout the simulation.

We calculate the application's total energy consumption with a Python script. Our script can calculate the energy consumption in two ways, defined by a NOINTERVAL input parameter. If NOINTERVAL is zero the energy model assumes our application processes a set of workloads arriving in uniform time intervals (e.g., every second). The application is in this case assumed to complete its workload, go to sleep and wake up just before the next workload arrives. Otherwise, when NOINTERVAL is one, our model assumes that the application executes one single non-periodic workload and terminates its execution when finished.

For simplicity, only the active execution is modeled in our GEM5 simulator. The sleep energy consumption is added for each simulation set through the script-based postprocessing. Equation 9 shows how we calculate the total energy per simulated frequency.

$$E_{\text{Sim i}} = P_{\text{active}_i} \cdot t_{\text{active}_i} + P_{\text{sleep}_i} \cdot t_{\text{sleep}_i} \tag{9}$$

P_{active} in Eq. 9 is calculated using the total power model of Eq. 7. P_{sleep} is taken from Table 4. One of the benefits of the GEM5 simulator output statistics is the modeling of the number and types of instructions executed, including memory accesses. This can be used for separation between different workload classes and allows us to model the effective capacitance C_E more accurately taking into consideration how it varies with different classes.

The total effective capacitance C_E is split up as shown in Eq. 10. C_{E_C} and C_{E_M}, for effective computation and memory capacitance, respectively, are combined to estimate the effective capacitance for our specific experiment. K_C and K_M are the fractions of the execution that are computational and memory access instructions, as calculated in Eq. 11 where x is the type of instruction and the numbers are collected from the stat.txt file generated during the experiment.

$$C_E = K_C \cdot C_{E_C} + K_M \cdot C_{E_M} \tag{10}$$

$$K_x = \frac{\text{\#Instructions of type x}}{\text{\#Sum of all executed instructions}} \tag{11}$$

The overall methodology presented in this paper can handle any number of workload classes that can be differentiated through estimated effective capacitance numbers and GEM5 statistics.

4.3 Offset Error Corrections

Parts of the circuitry in the SAM4L microcontroller are hard to incorporate into our model because it requires vendor specific information which is not publicly available, e.g., the clock circuitry. This will be the same for most commercial processors, and our methodology hence has to be able to handle this. We compensate for the power consumption of this additional circuitry through systematic offset corrections to the I_0, V_0 and E_C parameters in Eq. 7. The original parameter values were found as described in Section 4.1 running two different workloads on the microcontroller (one computational and one memory access workload). The offset correction parameters included in Eq. 12 are obtained by running the same workloads on GEM5 and comparing the power consumption results. The offset corrections are then approximated experimentally until the deviation in the simulated GEM5 results is minimized.

$$P_{\text{Total}} = I_0 X_{I_0} V_{\text{dd}} + \frac{I_0 X_{I_0}}{V_0 X_{V_0}} V_{\text{dd}}^2 + C_E X_{C_E} f V_{\text{dd}}^2 \qquad (12)$$

We use different error correction factors for different workloads as shown in Table 5. We can observe that for the SAM4L microcontroller no error corrections were needed for the static power, which implies that our assumptions in the static power model are sufficiently accurate in this case.

5 Experimental Setup

Our DVFS extension is implemented on the stable 2014-12-14 release[1] of GEM5. Our modular approach makes it easy to port to newer GEM5 releases through adaption of new or modified system components.

The DVFS GEM5 simulator is tested using two applications obtained from the Mediabench II website [14]; H264 and JPEG. Such applications can be found in the upper range of edge computing devices [31], not being among the latest more complex and compute intensive codecs. Their signal processing behavior is also relevant for application in less complex devices. From H264, we extracted the encoder control structure from the source code and modeled it under the assumption that all memory accesses take one clock cycle. The resulting code was implemented in a complete experimental system using our framework for system scenarios (FSS) [28]. Test data for three video streams simulated using different wireless networks (WLAN, LTE and WCDMA) was ported to GEM5 from our previous work [30]. The overhead of the scenario mechanisms in GEM5 were estimated using a separate program where all other code is removed. The energy consumption of the scenario mechanisms was averaged over 1 million frames in GEM5, resulting in an overhead of 1.54 μJ per frame simulated at 1.8 V and 40 MHz. This equals less than 0.01% of the total energy consumed while processing the stream, which we consider to be negligible.

[1]https://github.com/gem5/gem5/releases

Table 5 GEM5 offset correction parameters.

Workload	X_{C_E}	X_{I_0}	X_{V_0}
Arithmetic	1.18	1.0	1.0
Memory access	1.17	1.0	1.0

Like our H264 encoder control structure application, we extracted the JPEG compression control structure from the Mediabench II JPEG application, and the energy consumption of the JPEG compression application is measured on the SAM4L microcontroller with three different configurations. In the first configuration, we compress 500 consecutive 600x400 frames at 1.8 V and 40 MHz (JPEG FAST). In the second configuration we compress the same frames at 1.2 V and 12 MHz (JPEG SLOW). In the final configuration, we compress the frames using DVFS (JPEG DVFS), where we switch between our two voltage and frequency settings after every 10th compressed frame (i.e., 50 DVFS switches for 500 frames) in order to simulate a sequence with multiple DVFS switches.

6 Results

6.1 Correction of Systematic Offset Errors

The energy consumption of our applications is measured on the SAM4L microcontroller and estimated using our modified GEM5 simulator. The H264 application is run with the three data sets and with and without the scenario framework (FSS) while for JPEG the three different configurations are used. The deviation of the GEM5 simulator relative to the SAM4L measurements before and after correction of systematic offset errors are shown in Table 6.

Table 6 GEM5 energy estimate deviation from SAM4L measurements.

Application	Diff in value (%) without correction	Diff in value (%) with correction
H264 WLAN Scenario	14.5	1.2
H264 LTE Scenario	11.3	–0.6
H264 WCDMA Scenario	7.9	–4.0
H264 WLAN No Scenario	13.5	–0.01
H264 LTE No Scenario	13.0	–0.6
H264 WCDMA No Scenario	11.8	–1.9
JPEG FAST	13.5	0.06
JPEG SLOW	10.7	–0.9
JPEG DVFS	12.7	–0.1

6.2 Comparing Energy Reduction Techniques Using GEM5

Our GEM5 DVFS simulator can be used by designers to efficiently evaluate alternative strategies for energy efficient hardware and software implementations. As an example, Table 7 compares the relative energy savings of using the system scenario based design methodology over a brute-force RTH technique. We presented these experiments in our previous work [30] based on time consuming measurements on the SAM4L microcontroller board. The same experiments have now been performed using the GEM5 DVFS simulator. The percentage increase and decrease in energy consumption measured on the SAM4L board are listed in Table 7 together with the values obtained from the GEM5 simulator (with and without systematic error corrections). Negative values mean reduction in the energy consumption with the system scenario technique compared to the RTH technique. From Table 7 we can observe that the estimated improvements are not affected significantly by the systematic offset errors. This indicates that the fidelity of the estimates is good even without error correction and can be used to select the best solution among different alternatives.

In our previous work [30], we motivated that more available DVFS modes could result in further energy improvements. Again, we repeat the experiment using the GEM5 simulator and compare with the results from our previous estimates. Figure 3 shows the relative energy improvements with 6 compared to only 2 DVFS modes. Note that a stricter bandwidth sensitivity requirement of the LTE network was used for the 6 DVFS mode experiment, as discussed in [30]. The 6 DVFS modes energy improvements are also compared to an RTH solution with one constant voltage and frequency.

Figure 4 shows a plot from LabVIEW of the measured power consumption over time while Fig. 5 shows the corresponding GEM5 simulation plot.

Table 7 Energy consumption of H264 application with scenarios compared to no scenario.

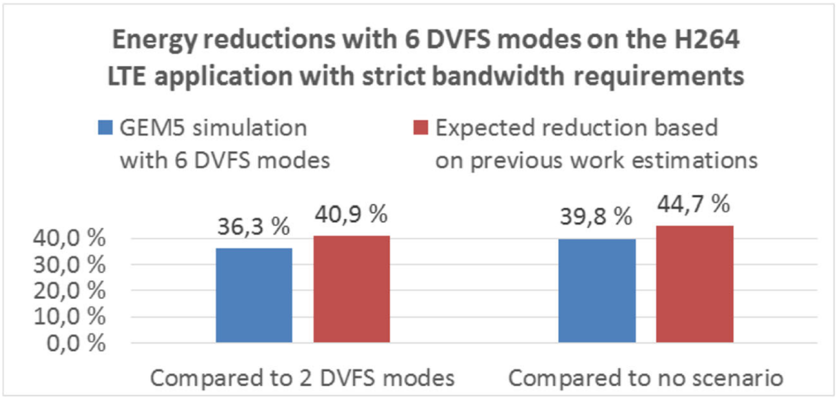
Application	Measurement on SAM4L	Estimated in GEM5	Estimated in GEM5 with correction
H264 WLAN	0.7%	-0.5%	–0.5%
H264 LTE	–44.2%	–43.1%	–44.2%
H264 WCDMA	–49%	–46.7%	–47.9%

7 Discussion of Results

Compared to measurements done with the same applications on the SAM4L microcontroller the results of our H264 and JPEG applications simulated in GEM5 show at most 4% deviation in values after offset error corrections. Such an accuracy is acceptable for most practical use cases in our edge computing target domain. Small errors occur for a number of reasons, however, and our measured deviation varied with the supply voltage. The main cause for these deviations is systematic offset errors, originating from additional processor and clock circuitry in the SAM4L board which are hard to incorporate into our model. Our offset correction parameters in Table 5 shows at most 18% offset error in the C_E estimation. This offset scales well with different V_{dd} values compared to our SAM4L measurements. The I_0 and V_0 variables are not offset corrected and implies that our assumptions related to the static power model have good enough accuracy to match the SAM4L microcontroller.

Other reasons for the deviations are the platform and model differences between the SAM4L microcontroller and the ARM model in GEM5. The SAM4L microcontroller board contains an in-order ARM Cortex M4 processor, while the ARM processor model in GEM5 is based on a simplified ARM Cortex A processor. Even if we use the simple atomic in-order CPU model in GEM5, these architectural differences will result in a small difference in the number of executed instructions. They will also result in different settings being used in the SAM4L ARM compiler compared to the ARM compiler used with GEM5 (such as platform specific optimization flags). Slight measurement errors and noise from the wires and probes on the SAM4L board also affect the measured

Figure 3 Energy reduction with 6 DVFS modes.



power consumption, which would deviate slightly from expected values. Future improved GEM5 processor models and measurement techniques will increase the estimation accuracy without having to change the overall methodology.

After our offset corrections, the deviation is reduced significantly, to an average of 1% and at most 4%. As shown in [27], direct use of McPAT can result in overestimates of several hundred percent. The average error can be reduced down to 2 to 5% through learning-based post-silicon calibration [15]. It requires substantially more effort than our approach, which we show is not needed in our domain of interest. In general, our offset corrected energy reduction results from the GEM5 simulator coincide well with the measured improvements from the SAM4L microcontroller for the H264 application.

In our previous work [30] we investigated, through rough manual calculations, the consequences of increasing the number of DVFS modes. Using inter- and extrapolation from the 2 DVFS modes on the SAM4L microcontroller, we calculated the dynamic power consumption that could be expected if 6 DVFS modes were available. RTH was not exploited in either case. With stricter bandwidth requirements and a fixed $C_E = 0.165nF$, the system scenario technique with 6 DVFS modes was roughly calculated to improve the energy consumption by 59% compared to not using scenarios [30]. We now use the same parameters as in [30] and extended this rough calculation with a combination of system scenario and RTH techniques. We predict 40.9% energy reduction when combining RTH with the system scenario technique with 6 DVFS modes compared to combining it with only 2 DVFS modes available. The GEM5 simulation of the same two alternatives results in 36.3% improvement, which is slightly less than our roughly estimated value of 40.9%. Our GEM5 energy model takes into account the static and sleep energy consumption, which is not taken into account by our rough calculation. The finer granularity of our power model, with different workload classes (and C_E values) also contributes to the difference from the calculated values.

Our DVFS implementation and energy modeling approach differs from previous techniques presented in Section 2, e.g., through introduction of workload classes and through calibration using measurements from real hardware. The energy model can easily be ported to other types of architectures, by taking new measurements of the power consumption for different workload classes and measuring the DVFS switching energy. These measurements can then be used to re-calculate the C_E, I_0 and V_0 parameters for the new target architecture, easily extending the practical use of the methodology. Our changes to the GEM5 simulator kernel and configuration scripts are for the most part architecture independent and hence fully reusable, except for the DVFS instruction. GEM5 supports different CPU models, which can be used with our GEM5 extension by applying the modifications mentioned in Section 3. This enables the system designer to experiment with new techniques on

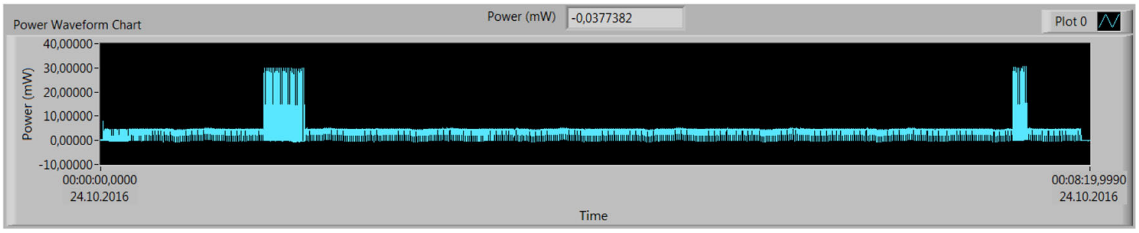


Figure 4 SAM4L H264 measurements plot.

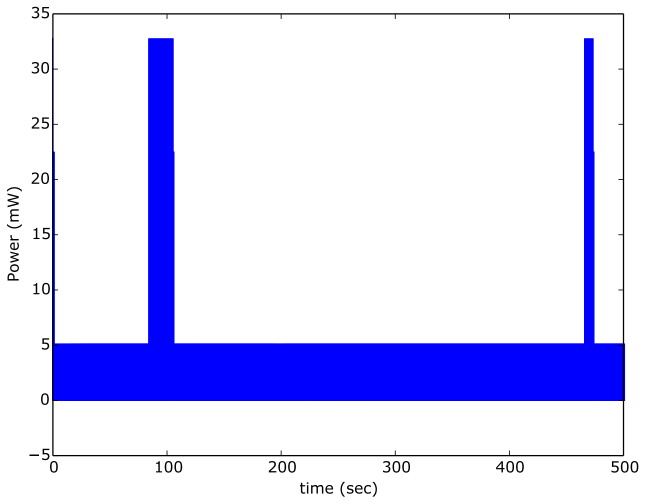


Figure 5 GEM5 H264 simulation plot.

non-ARM architectures and add more DVFS modes or different switching policies.

8 Conclusions

We present a non-intrusive application controlled DVFS management in GEM5 SE mode, and a Python script-based energy model used together with GEM5. Together this gives edge computing system designers tools needed for ultra low power design space exploration. The application triggers DVFS through one custom pseudo-instruction, implemented in GEM5's ARM instruction-set model. This instruction takes advantage of GEM5's utility functions, making it easily portable to other processor architectures. The rest of our DVFS mechanisms are architecture independent. Our energy model accurately parametrizes different workload classes, which are calibrated with power measurements from real HW. The results from our modified GEM5 simulator are validated with representative signal processing applications. The energy results from our GEM5 simulation are compared to measurements on a SAM4L microcontroller, resulting in less than 4% deviation after systematic offset error correction. Our changes to the GEM5 simulator kernel and configuration scripts are for the most part architecture independent and hence fully reusable, except from the DVFS instruction. Our modifications to GEM5 mentioned in this paper can be applied to other CPU models supported by GEM5 and allow for easy and accurate power and energy estimation for a range of practical processors and applications.

Acknowledgments Open Access funding provided by NTNU Norwegian University of Science and Technology (incl St. Olavs Hospital - Trondheim University Hospital).

References

1. Aleem, M., Islam, M.A., Iqbal, M.A. (2016). A comparative study of heterogeneous processor simulators. *International Journal of Computer Applications* 148(12).
2. Alioto, M. (2012). Ultra-low power VLSI circuit design demystified and explained: a tutorial. *IEEE Transactions on Circuits and Systems I: Regular Papers*, *59*(1), 3–29. https://doi.org/10.1109/TCSI.2011.2177004.
3. Andreou, A.G., Boahen, K.A., Pouliquen, P.O., Pavasovic, A., Jenkins, R.E., Strohbehn, K. (1991). Current-mode subthreshold MOS circuits for analog VLSI neural systems. *IEEE Transactions on Neural Networks*, *2*(2), 205–213. https://doi.org/10.1109/72.80331.
4. Atmel, S. (2015). AM4L Xplained pro user guide. http://ww1.microchip.com/downloads/en/devicedoc/Atmel-42074-SAM4L-Xplained-Pro_User-Guide.pdf.
5. Atmel (2016). ATSAM ARM-based flash MCU SAM4L series datasheet. http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42023-ARM-Microcontroller-ATSAM4L-Low-Power-LCD_Datasheet-Summary.pdf. 42023HS-SAM-11/2016.
6. Austin, T., Larson, E., Ernst, D. (2002). Simplescalar: an infrastructure for computer system modeling. *Computer*, *35*(2), 59–67. https://doi.org/10.1109/2.982917.
7. Awan, M.A., & Petters, S.M. (2014). Race-to-halt energy saving strategies. *Journal of Systems Architecture*, *60*(10), 796–815. https://doi.org/10.1016/j.sysarc.2014.10.001.
8. Binkert, N., Beckmann, B., Black, G., Reinhardt, S.K., Saidi, A., Basu, A., Hestness, J., Hower, D.R., Krishna, T., Sardashti, S., Sen, R., Sewell, K., Shoaib, M., Vaish, N., Hill, M.D., Wood, D.A. (2011). The gem5 simulator. *SIGARCH Computer Architecture News*, *39*(2), 1–7.
9. Brooks, D., Tiwari, V., Martonosi, M. (2000). Wattch: a framework for architectural-level power analysis and optimizations. *SIGARCH Compututer Architecture News*, *28*(2), 83–94. https://doi.org/10.1145/342001.339657.
10. Carlson, T.E., Heirmant, W., Eeckhout, L. (2011). Sniper: exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In *2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)* (pp. 1–12). https://doi.org/10.1145/2063384.2063454.
11. Chandrakasan, A.P., Sheng, S., Brodersen, R.W. (1992). Low-power CMOS digital design. *IEEE Journal of Solid-State Circuits*, *27*(4), 473–484. https://doi.org/10.1109/4.126534.
12. Ghasemzadeh, H., & Jafari, R. (2013). Ultra low-power signal processing in wearable monitoring systems: a tiered screening architecture with optimal bit resolution. *ACM Transactions on Embedded Computing Systems (TECS)*, *13*(1), 9.

13. Haririan, P., & Garcia-Ortiz, A. (2014). Non-intrusive DVFS emulation in gem5 with application to self-aware architectures. In *2014 9th International Symposium on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)* (pp. 1–7).
14. Lee, C., Potkonjak, M., Mangione-Smith, W. (1997). Mediabench: a tool for evaluating and synthesizing multimedia and communications systems. In *Thirtieth annual IEEE/ACM international symposium on microarchitecture, 1997. Proceedings* (pp. s330–335). https://doi.org/10.1109/MICRO.1997.645830.
15. Lee, W., Kim, Y., Ryoo, J.H., Sunwoo, D., Gerstlauer, A., John, L.K. (2015). Powertrain: a learning-based calibration of McPAT power models. In *2015 IEEE/ACM International symposium on low power electronics and design (ISLPED)* (pp. 189–194). https://doi.org/10.1109/ISLPED.2015.7273512.
16. Li, S., Ahn, J.H., Strong, R.D., Brockman, J.B., Tullsen, D.M., Jouppi, N.P. (2009). McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of the 42nd annual IEEE/ACM international symposium on microarchitecture, MICRO 42* (pp. 469–480). New York: ACM, https://doi.org/10.1145/1669112.1669172.
17. Ma, Z., Marchal, P., Scarpazza, D., Yang, P., Wong, C., Gomez, J., Himpe, S., Ykman-Couvreur, C., Catthoor, F. (2007). *Systematic methodology for real-time cost-effective mapping of dynamic concurrent task-based systems on heterogeneous platforms*. Springer.
18. Mittal, S. (2014). A survey of techniques for improving energy efficiency in embedded computing systems. *International Journal of Computer Aided Engineering and Technology*, *6*(4), 440–459.
19. Nowatzki, T., Menon, J., han Ho, C., Sankaralingam, K. (2014). gem5, GPGPUSim, McPAT, GPUWattch, "your favorite simulator here" considered harmful. In *11th Annual workshop on duplicating, deconstructing and debunking* (pp. 1–10).
20. Power, J., Hestness, J., Orr, M.S., Hill, M.D., Wood, D.A. (2015). gem5-gpu: a heterogeneous CPU-GPU simulator. *IEEE Computer Architecture Letters*, *14*(1), 34–36. https://doi.org/10.1109/LCA.2014.2299539.
21. Rabaey, J.M., & Pedram, M. (1996). *Low power design methodologies* (Vol. 336). The Springer International Series in Engineering and Computer Science.
22. Shi, W., Cao, J., Zhang, Q., Li, Y., Xu, L. (2016). Edge computing: vision and challenges. *IEEE Internet of Things Journal*, *3*(5), 637–646. https://doi.org/10.1109/JIOT.2016.2579198.
23. Spiliopoulos, V., Bagdia, A., Hansson, A., Aldworth, P., Kaxiras, S. (2013). Introducing DVFS-management in a full-system simulator. In *2013 IEEE 21st International symposium on modelling, analysis and simulation of computer and telecommunication systems* (pp. 535–545).
24. STM (2016). STM32F4 datasheet. https://www.st.com/resource/en/datasheet/dm00037051.pdf. DocID022152 Rev 8.
25. Ubal, R., Jang, B., Mistry, P., Schaa, D., Kaeli, D. (2012). Multi2sim: a simulation framework for CPU-GPU computing. In *Proc. of the 21st international conference on parallel architectures and compilation techniques*.
26. Veendrick, H.J.M. (1984). Short-circuit dissipation of static CMOS circuitry and its impact on the design of buffer circuits. *IEEE Journal of Solid-State Circuits*, *19*(4), 468–473. https://doi.org/10.1109/JSSC.1984.1052168.
27. Xi, S.L., Jacobson, H., Bose, P., Wei, G., Brooks, D. (2015). Quantifying sources of error in McPAT and potential impacts on architectural studies. In *2015 IEEE 21st International symposium on high performance computer architecture (HPCA)* (pp. 577–589). https://doi.org/10.1109/HPCA.2015.7056064.
28. Yassin, Y., Kjeldsberg, P., Catthoor, F. (2015). System scenario framework evaluation on EFM32 using the H264/AVC encoder control structure. In *2015 European conference on circuit theory and design (ECCTD)* (pp. 1–4). https://doi.org/10.1109/ECCTD.2015.7300121.
29. Yassin, Y., Kjeldsberg, P., Perkis, A., Catthoor, F. (2018). Techniques for dynamic hardware management of streaming media applications using a framework for system scenarios. *Elsevier Microprocessors and Microsystems*, *56*, 157–168. https://doi.org/10.1016/j.micpro.2017.12.002.
30. Yassin, Y., Kjeldsberg, P.G., Perkis, A., Catthoor, F. (2016). Dynamic hardware management of the H264/AVC encoder control structure using a framework for system scenarios. In *19th EUROMICRO Conference on digital system design (DSD'16)* (pp. 1–8): IEEE.
31. Zhang, H., Zhao, S., Pattnaik, A., Kandemir, M.T., Sivasubramaniam, A., Das, C.R. (2019). Distilling the essence of raw video to reduce memory usage and energy at edge devices. In *Proceedings of the 52nd Annual IEEE/ACM international symposium on microarchitecture* (pp. 657–669). https://doi.org/10.1145/3352460.3358298.

Affiliations

Yahya H. Yassin[1] · **Magnus Jahre**[2] · **Per Gunnar Kjeldsberg**[1] · **Snorre Aunet**[1] · **Francky Catthoor**[3,4]

Magnus Jahre
magnus.jahre@ntnu.no

Per Gunnar Kjeldsberg
pgk@ntnu.no

Snorre Aunet
snorre.aunet@ntnu.no

Francky Catthoor
Francky.Catthoor@imec.be

1 Department of Electronic Systems, Norwegian University of Science and Technology (NTNU), Trondheim, Norway

2 Department of Computer Science, Norwegian University of Science and Technology (NTNU), Trondheim, Norway

3 IMEC, Kapeldreef 75, 3000 Leuven, Belgium

4 Department of Electrical Engineering (ESAT), Katholieke Universiteit Leuven (KULeuven), Leuven, Belgium