MemCork: Exploration of Hybrid Memory Architectures for Intermittent Computing at the Edge

Theo Soriano, David Novo, Guillaume Prenat, Gregory Di Pendina, Pascal Benoit

MemCork: Exploration of Hybrid Memory Architectures for Intermittent Computing at the Edge

Theo Soriano[1], David Novo[1], Guillaume Prenat[2], Gregory Di Pendina[2], and Pascal Benoit[1]
[1] LIRMM, University of Montpellier, CNRS - Montpellier, France - {firstname}.{lastname}@lirmm.fr
[2] SPINTEC - University of Grenoble-Alpes, CNRS, CEA - Grenoble, France - {firstname}.{lastname}@cea.fr

***Abstract*—Microcontroller units (MCUs) are often used in Internet of Things nodes that operate intermittently. Such nodes alternate active and inactive phases under strict energy constraints. Typically, the memory system has a significant impact on overall MCU energy consumption. Memory accesses and memory leakage power often dominate the consumption of active and inactive phases, respectively. Emerging Non-Volatile Memory (NVM) technologies have recently enabled the design of non-volatile MCUs that can significantly reduce energy consumption during inactive phases. However, replacing all memories with emerging NVMs is not necessarily the best solution, as it often results in dynamic power overhead during active phases. Instead, a hybrid memory architecture that combines volatile and non-volatile technologies is a promising alternative. However, designing hybrid memory MCUs is challenging because the technology that best fits a data segment depends on its access pattern during execution (*e.g.*, program memory experiences mostly reads while the stack alternates reads and writes). For a given intermittent application, our goal is to find the best memory architecture based on a data mapping that takes advantage of the different properties of the available memory technologies. To this end, we present MemCork, a tool for hybrid memory architecture exploration in intermittent computing devices. Based on an instrumented execution on a technology-agnostic FPGA prototype, our tool exhaustively explores the possible data mapping and memory architecture combinations to find the most energy-efficient solution. We evaluate MemCork on two representative intermittent applications and find a customised memory architecture and data mapping that reduces energy consumption by up to 23% compared to a fully NVM solution.**

***Index Terms*—IoT, intermittent computing, design exploration, MRAM, FPGA emulation, wireless sensor node, edge computing, tiny ML**

I. Introduction

For many applications, electronic systems operate intermittently. This is particularly the case in the context of the Internet of Things (IoT). IoT nodes can be used for sensing, actuating, processing, receiving, and/or transmitting data, but these functions are only executed periodically or following a trigger event, meaning that the device remains inactive for the rest of the time. Nodes include computing and storage resources embedded in a Microcontroller Unit (MCU) and usually operate in energy-constrained environments. To avoid unnecessary energy consumption during the inactive phases, IoT nodes can operate under different energy-saving modes, the most aggressive one being the total cut-off of the power supply. This powering down can be intended or not, depending on available energy management policies. Although cutting the power supply is the most radical solution for limiting energy consumption, it is also the one that penalises the most in terms of performance and reactivity.

As the memory can occupy up to 95% of the silicon area for some MCUs, the power it consumes represents the dominant part of the total circuit consumption [1]. An MCU is designed to run simple applications with one or more tasks. They include a very simple memory hierarchy with memories integrated in a flat address space (no cache). In general, the architecture includes at least three memories: a working one based on SRAM technology, a storage one based on Flash technology and a Read-Only Memory (ROM) that contains the boot code. Depending on the application parameters, such as the duration of the inactive phase, the right compromise must be found between keeping the volatile memories powered and saving and restoring their contents in non-volatile memory so that they can be switched off.

Emerging non-volatile memory technologies (NVM) such as FRAM, RRAM or MRAM have grown in maturity and their integration into MCU architecture has been the subject of a vast amount of research. These memories have SRAM-like performance while having the capability to hold saved data even if the power is turned off. Accordingly, they open new opportunities in the design of ultra-low-power MCUs for standby power critical applications and for energy harvesting-based systems. Non-volatile MCUs that rely on the use of emerging NVM are presented as interesting alternatives to SRAM/Flash-based MCUs. Prior works include MCUs implemented with FRAM [2] [3] [4] [5], RRAM [6] [7] [8] and MRAM [9] [10] [11]. Emerging NVMs are used to provide instant on/off capabilities to MCUs with near-zero power consumption during the inactive phases. These works have investigated the possibility of replacing SRAM and Flash with a single universal memory, or a mix of SRAM/Emerging NVM memories. Although these studies have succeeded in showing the advantages of using emerging NVM or hybrid solutions in specific applications, they usually present a particular solution that is difficult to generalise to different application and technology contexts. This non-systematic approach is costly in terms of design time and does not allow an easy and rapid exploration of different size/performance/power consumption trade-offs.

Our goal is to find the best memory architecture based on a data mapping that takes advantage of the different properties of the available technologies. We present MemCork, a framework for memory architecture and data mapping exploration in intermittent MCU applications. Based on an evaluation on a technology-agnostic FPGA prototype, it allows exhaustively exploring the possible data mapping and memory architecture combinations to find the most energy-efficient solution. To show the relevance of our approach, we use MemCork to evaluate two different intermittent applications and show that different application characteristics can influence the best design choice for the memory architecture and the data mapping.

II. MCU Design and Main Challenges

The design of an energy-efficient MCU implementation of an intermittent IoT application is a challenging task. It includes the definition of an architecture based on a set of memory technologies and the corresponding data and instruction mapping.

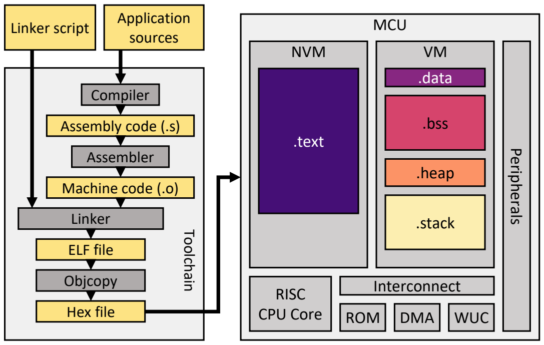

Fig. 1. Typical microcontroller compilation flow and architecture

MCU architecture and compilation flow. As shown in Fig. 1, most MCUs include a Reduced Instruction Set Computer (RISC) CPU core, a set of task-specific logic circuits such as Direct Memory Access (DMA) or Wake-Up Controller, peripherals to communicate with other components, Volatile Memories (VM) and NVM. NVM is usually based on a ROM for the boot code, and a storage memory (Flash or EEPROM) for the program and read-only data. The VM (SRAM) is used to store data during program execution. At the software level, the source code is compiled and then linked to generate the executable, using a dedicated toolchain. We focus on the linker script, which declares and defines the memories and their corresponding address ranges. At compile time, this script maps instructions and data to the memories by grouping them into typically five sections. The *.text* section contains the application code as a sequence of instructions, the read-only data and the default values of the application global variables initialised to specific values other than zero. The application global variables are also stored in the *.data* section where default values are initialised from *.text* section at startup and can be modified during execution. In the same way, the *.bss* section contains all uninitialised variables and is initialised to zero at startup. The *.heap* is used for dynamic memory allocation and the *.stack* for temporary data such as function parameters, return address and local variables. For both of them, their size is defined by the programmer in the linker script. As depicted in Fig. 1, the *.text* section is stored in NVM, while *.data*, *.bss*, *.heap* and *.stack* are located in VM.

Main challenges. The *first challenge* is the definition and design of high-level intermittent application and memory models. Accordingly, we define an intermittent application model that covers a wide range of possible applications by dividing them into several phases and defining the activity and operating modes of the memories in each phase. Regarding high-level memory energy models, we define and build them for different technologies and sizes in a consistent way for activation-based evaluation. For the mainstream technologies, we rely on information extracted from datasheets and memory compilers, while for MRAM we perform electrical simulations and characterisation of complete memory banks. The *second challenge* comes from the fact that in the case of an event-based IoT node, memory size and access pattern depend on the application type and on its environment. For example, a typical temperature sensor node application will sample data and do some processing (possibly min/max, average) before transmitting the data. This type of application requires a small memory and generates sporadic requests. However, if we take the example of an intelligent application based on machine learning, involving significantly more processing and the storage of many constant values, the memory needs and activity are considerably higher. Furthermore, in event-based applications, the wake-up probability depends on the environment in which the system is located. Regardless of the application, the memory usage will be strongly correlated with the wake-up frequency. All these parameters have an impact on the activity of the memory and therefore on its energy. To address this challenge, we propose MemCork, a framework for intermittent MCU application evaluation. We use an application evaluation platform allowing a fine-grained tracking of the memory activity, performed on a technology-agnostic FPGA prototype to characterise various applications running in real-time in a real-life environment. Finally, the *third challenge* relates to the design exploration of all the possible memory architectures and data mapping options considering multiple memory technologies for a given application. Accordingly, in MemCork, we introduce a memory architecture and data mapping exploration tool that takes advantage of the different properties of memory technologies to find a custom architecture and data mapping for minimal memory energy consumption.

III. Modelling Assumptions

In this section, we define a set of parameters to model and evaluate the energy of intermittent computing MCUs to enable hybrid memory architecture exploration. To this end, we first define an intermittent application model that covers a wide range of possible applications by dividing them into several phases. We approximate the overall energy as a function of key parameters that capture the activity and operating modes of each memory in each phase. We then build parametric memory models for different technologies and sizes to enable an activation-based energy consumption estimation.

A. Intermittent application model

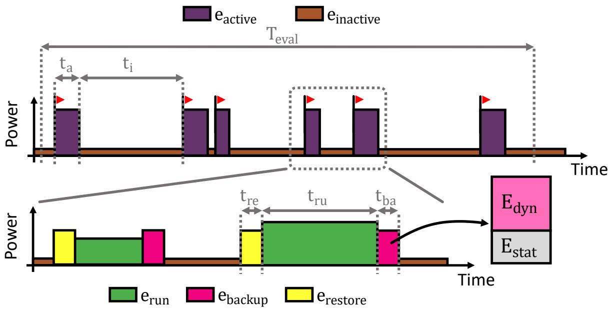

Fig. 2. Intermittent computing MCU power profile

Fig. 2 illustrates a possible MCU power profile of an IoT node running an intermittent application. The system is inactive by default and performs event-driven tasks. Events are depicted as flags in the picture; they can be periodic or aperiodic. Before transitioning from an active to an inactive phase, the node can secure the data stored in VM with a transfer to NVM (backup). In this case, it is necessary to add hardware and software support for backup and restore (the reverse operation) of volatile data. Based on these basic assumptions, we define several timing parameters to characterise the application profile. We consider the system in an evaluation time window denoted as $T_{eval} = T_a + T_i$, where T_a and T_i are the sums of all the active (t_a) and inactive (t_i) phases, respectively. We further split t_a into an execution phase t_{ru}, a backup phase t_{ba} and a restore phase t_{re}. Thus, t_a and t_i depend on the frequency of events and the duration of the corresponding phases.

Regarding the energy modeling, we define the MCU energy during T_{eval} as E_{mcu}, which corresponds to the sum of the consumption during T_a (E_{active}) and T_i ($E_{inactive}$). E_{active} is the sum of the energy of each active phase (e_{active}) and $E_{inactive}$ is the sum of each inactive phase ($e_{inactive}$). Thus, $e_{active} = e_{restore}+e_{run}+e_{backup}$. E_{mcu} is also the sum of the energy of the memory E_{mem} and the remaining logic E_{logic}. However, as justified in Section I, we assume $E_{logic} \ll E_{mem}$. Thus, we approximate the consumption of the MCU to that of all its memories, $i.e.$, $E_{mcu} \approx E_{mem}$. The memory energy during any phase is $E_{mem} = E_{dyn} + E_{stat}$, with E_{dyn} being the sum of the dynamic energy of each memory (e_{dyn}) and E_{stat} the sum of their static energy. We define the dynamic energy of a memory as $e_{dyn} = (n_{rd} \times E_{rd}) + (n_{wr} \times E_{wr})$, n_{rd} and n_{wr} being the number of read and write accesses, E_{rd} and E_{wr} the read and write energy, respectively. We define backup and restore content as $B_{br} = \sum B_{vm}$ with B_{vm} being the size of data and/or instructions stored in a VM. For backup activity, we consider the reading of B_{vm} in each of the VM and the writing of B_{br} in the destination NVM. For restore activity, we consider the reversed process. Backup and restore phases duration t_{ba} and t_{re} depend directly on the bandwidth of the source and destination memories and on the amount of data to be transferred B_{br}. We assume that the backup and restore procedures are executed by a dedicated device such as a DMA.

For static energy, we assume a typical memory level power management, *i.e.*, power modes can be controlled independently for each memory. We also assume `On` and `Off` modes. Memories can only be accessed in `On` mode, which corresponds to a static leakage power of P_{statOn}. The `Off` mode allows a lower static leakage power $P_{statOff}$ but leads to the loss of the VM content. The static energy of an execution phase of duration t_{phase} is $e_{stat} = P_{stat} \times t_{phase}$. For P_{stat} during a run phase, we consider all memories that contain data or instructions as `On` while other memories are `Off` (*e.g.*, empty memory or NVM used only for backup). Concerning backup and restore, we consider all VM and NVM involved in the process as `On` while other memories are `Off`. Finally, during an inactive phase all memories are `Off`.

B. Memory models

To calculate the energy for different memories, we model each technology in terms of the parameters defined in the previous subsection: E_{rd}, E_{wr}, P_{statOn}, $P_{statOff}$. As we explore memory sizing, we choose to express these parameters as a function of the memory size. We collect data for different memory sizes of each technology and approximate each parameter as an affine function $ax + b$ with x being the memory size, and a and b technology-dependent constants.

Parameters were extracted from commercial MCU datasheets for 1MB and 2MB Flash memory with a 90nm technology node, which was scaled down to 28nm. The same information was obtained for two 28nm FD-SOI SRAM (16kB and 64kB) built with the single-port high-density compiler supporting forward body bias (SPHD-BODYBIAS) from STMicroelectronics.

Emerging NVM, such as MRAM, are less mature than mainstream technologies. As a result, we have no access to datasheet information or memory compilers. Instead, we resort to the design of a full memory at the transistor level and its characterisation to extract the mentioned parameters. We choose to design a high-density 32-bit word single bank with an SRAM-like interface. The chosen CMOS process is 28nm FDSOI from STMicroelectronics and the considered MRAM processes are STT and SOT, with MTJ diameters of 28nm.

The first step is to calibrate the compact MTJ models of the two MRAM technologies for electrical simulations. For each of them, we calibrate the model to fit the state of the art for an MTJ diameter of 35nm, and then extrapolate to 28nm. Based on these compact models, we can design and characterise 128kB STT and SOT memories. Both memories include 1024 rows and columns. Since the word size is 32 bits, we have 32 words per row, resulting in a 32-to-one multiplexing to select the word. The address is thus composed of 15 bits: 10 to select the row and 5 to select the word in the row. We characterise the bit cell by evaluating the parasitic resistance and capacitance of the access lines depending on the memory size. This is evaluated by drawing a very simplified layout of the memory array to approximate the size of the access rows and parasitics. These parasitics are injected in the critical path of the memory, which contains the bit cell, made of the MTJ and access transistor, the writing drivers, reading amplifiers and access transistors such as activation transistors and multiplexing. Thus, the critical path is representative of the worst-case scenario for any bit cell and allows an exhaustive set of electrical simulations using standard electrical simulators such as Spectre from Cadence. We use these simulations to ensure the operation of the memory and extract the main parameters in terms of writing and reading currents, as well as timing data. Full memory operation is then performed and validated using a "fast-spice" simulator, such as UltraSim from Cadence, to reduce the simulation runtime and cover a large number of cases. From these other simulations, we can extract write and read energies, the `On` and `Off` static power due to leakage of a 128kB memory for both STT and SOT as shown in Table I.

TABLE I
128 kB MRAM HIGH-LEVEL ENERGY MODELS

	STT 28nm	SOT 28nm
Size [kB]	128	128
Byte read energy [J/B]	7.50e-12	7.50e-12
Byte write energy [J/B]	7.50e-12	3.75e-12
Static power in mode `On` [W]	1.50e-04	1.50e-04
Static power in mode `Off` [W]	2.40e-09	2.40e-09

To extend these estimations to other memory sizes, we build linear functions of the MRAM parameters. We assume that P_{statOn} and $P_{statOff}$ increase linearly with the memory size as all powered transistors contribute to leakage. For E_{rd} and E_{wr} we assume constant values because the bit-cells selected in a memory access remain constant for any memory size.

IV. MemCork

Now that we have defined the application and memory models, we look more closely at the memory exploration tool. In this section, we present MemCork, a framework for memory architecture and data mapping exploration in intermittent MCU applications. First, we propose an evaluation methodology based on a fine-grained tracking of the memory activity performed on a technology-agnostic FPGA prototype. It enables to retrieve both compile-time and run-time application parameters of real-time intermittent applications running in a real-life environment. Second, we combine the extracted activity information with the high-level models introduced in Section III in an exploration tool. This tool allows us to exhaustively explore the possible hybrid memory architectures and data mappings to select the most energy-efficient solution.

A. Application evaluation

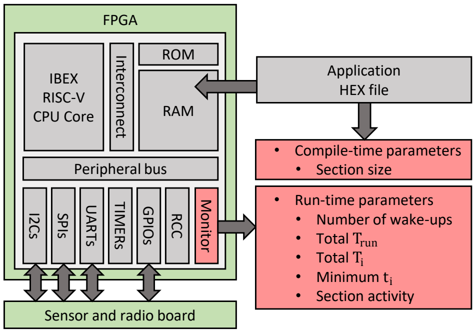

Fig. 3. Evaluation platform

We follow a methodology similar to [12], where an FPGA-based emulation platform for edge computing node design exploration is used to track the memory activity of a real execution. It includes a wireless sensor node prototype for architecture evaluation and exploration under real-life conditions. The prototype is based on a Nexys video FPGA development board connected to a radio module (with Bluetooth low energy and LoRa) and sensors (inertial measurement unit, microphone, temperature, humidity, and camera). It also integrates a power management unit for battery voltage regulation. As illustrated in Fig. 3, an open-source 32b RISC-V MCU architecture with various peripherals is embedded into the FPGA.

We extend the platform with an enhanced monitor, capable of retrieving run-time information and to evaluate the activity of memories with a fine granularity. Monitoring memory addresses individually is not realistic as it would generate excessively large traces. Thus, we change the activity monitoring to record at the section granularity (see Section II). The monitoring infrastructure performs two tasks: (1) configuration of the FPGA monitors based on the application compile-time parameters (sizes of the sections), and (2) recording of application run-time parameters (obtained by running the application on the platform during T_{eval}). The monitor, configured from software, tracks n_{rd} and n_{wr}, the number of reads and writes in several parameterizable sections. It also measures T_i as described in Section III. Once configured, the monitor is non-intrusive.

It must be noticed that the application execution on FPGA does not include backup and restore phases, as they depend on the memory technology. Their execution is only considered in the subsequent hybrid memory exploration step (Section IV-A). Since they result in an extra activity at the beginning (back-up), and the end (restore) of the inactive phases, it requires a minimum inactive duration t_i between two run phases to be feasible.

B. Hybrid memory exploration

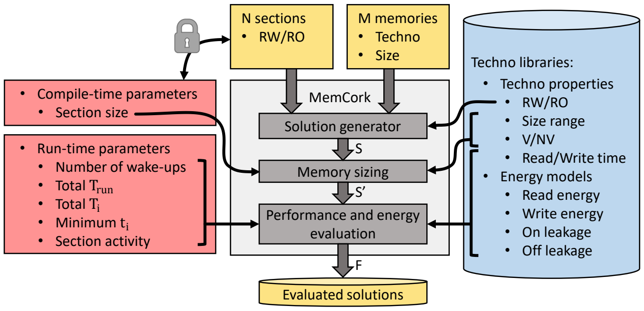

Fig. 4. MemCork exploration flow

As illustrated in Fig. 4, this exploration tool uses five inputs; first, a library containing the properties and energy models of each available memory technology (detailed in Section III); second, M, the set of memories, with their corresponding technology and size defined by the user; third, N, the set of sections and their properties; fourth, the compile-time parameters that contain information about sections such as their sizes; finally, the output of the monitor with the resulting run-time parameters (detailed in Section IV-A).

We propose an algorithm that explores all possible memory architecture and section mapping combinations for the evaluated application, with $m = |M|$ the number of memories, and $n = |N|$ the number of sections. A memory M_j can store $|M_j|$ bytes, and can be read-only (R), or read/write (R+W). $|N_i|$ is the size in bytes of a section N_i. A feasible mapping is an application $f : N \mapsto M$ where all sections can be mapped to the available memories. One or more N_i sections can be mapped into a memory M_j provided that R+W N_i sections are only mapped to a R+W M_j memory. Based on its inputs, the tool generates S, the set of solutions (feasible mappings) with cardinal $|S| \leq m^n$, depending on the constraints of the different sections and memory, *i.e.*, type (R or R+W) and size.

Based on the section sizes and on the required backup space, the tool computes the dimension of each memory for each solution of the S set. A user-defined upper and lower limit is assumed for each memory technology. According to the generated section mapping, it determines the minimal memory depth that minimises the total consumption. A memory size is valid only if it is within the range allowed in the corresponding memory technology library. Once calculated for each solution, the tool extracts S', the subset of S containing all solutions for which the memory size is valid.

Finally, based on the run-time parameters obtained with the monitor described above, the solutions of the set S' are evaluated following the energy modelling methodology proposed in Section III. As mentioned in Section IV-A, the memory activity and duration related to backup and restore is introduced at this stage. The energy of each memory for each application phase during the evaluation window is calculated.

The output of the tool is the solution set F, which is a subset of the set S'. It only includes the solutions where execution is feasible, *i.e.*, complies with the minimum t_i required for backup and restore.

V. Results

Recent studies have succeeded in showing the advantages of using emerging NVM or hybrid solutions in specific applications. MemCork is a systematic and automated exploration tool, able to cover a wide range of applications. To demonstrate this capability, we define two applications with very different characteristics: a simple *light* application representative of temperature-like sensor nodes, and a much more complex *heavy* application based on machine learning inference programmed in TensorFlow Lite for Microcontrollers. For both applications we assume a periodic wake-up with a period T and an identical execution phase at each wake-up. In this particular case, $T_{eval} = T$ is sufficient to evaluate the application.

Table II shows the application parameters for both applications. For the *light* one, the code is very simple resulting in a *.text* section smaller than 10kB and the amount of raw data generated by the sensor is very low (4B). The data section is empty while the *.bss* section is a bit more than 5kB because it contains all the user-level input and output buffers of communication peripherals. For the *heavy* application, the code is significantly more complex integrating multiple libraries (TensorFlow Lite + KissFFT), which results in a significantly larger *.text* section (236kB). Furthermore, the input raw data have a maximum size of 32kB, the associated features use 2kB and the TensorFlow Lite model uses 20kB.

TABLE II
Application parameters from monitor output

Section	*Light* app			*Heavy* app		
	Size [B]	Rd [B]	Wr [B]	Size [B]	Rd [B]	Wr [B]
.text	6.0e+03	2.6e+06	0	2.4e+05	4.8e+08	0
.data	0	0	0	1.4e+02	1.9e+02	2.0e+00
.bss	5.1e+03	4.6e+04	5.2e+02	1.7e+04	2.3e+06	1.5e+05
.heap	1.0e+03	0	0	3.3e+04	2.0e+07	1.5e+06
.stack	1.0e+03	4.6e+02	4.8e+02	3.3e+04	6.3e+07	2.9e+06

Concerning run-time parameters, the *light* application t_{ru} is 17ms versus the 3.53s for the *heavy* application. Table II shows that in both cases, reads in *.text* section represent a large part of the overall memory activity (from 83% to 98%). For the *light* application, there is no activity in the *.text* section as it is empty and the application does not use dynamic allocation resulting in no activity in the *.heap* section. The *.bss* and *.stack* sections are the most active after the *.text* section. This is because it contains the input and output buffers of the peripherals: in this type of application, the use of the peripherals represents a large part of the activity. Concerning the *heavy* application, the *.heap* and *.stack* sections show a high activity due to the complexity of the application. We also observe a certain activity in the *.bss* section. Indeed, this section contains an array of about 10kB generated by TensorFlow and called Tensor Arena, which is used to store input, output, and intermediate arrays during inference. Finally, we notice that the activity in the *.data* section is very low because the application uses very rarely initialised global variables.

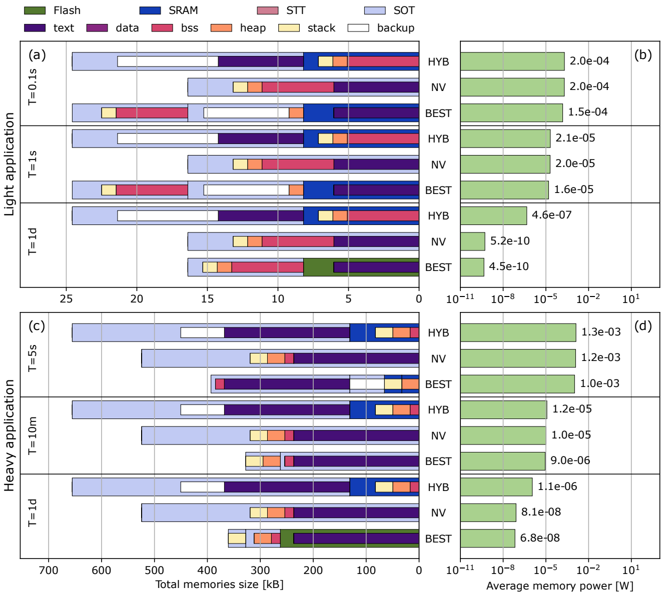

Fig. 5. Memory architecture (left) and corresponding average power (right) of a hybrid (HYB), a fully non-volatile (NV) and the optimal (BEST) solutions proposed by MemCork depending on different wake-up periods for the *light* (top) and *heavy* (bottom) applications

We used MemCork to study the influence of application parameters such as the wake-up period, section size and activity on the memory architecture and section mapping. The read/write accesses for each section were retrieved for the considered application, over the chosen evaluation window. The results are summarised in Fig. 5: the top bar graphs for the *light* application and the bottom ones for the *heavy* application. In this experiment, we considered 7 memories in the set M: one Flash (green), two SRAM (dark blue), two STT (pink) and two SOT (light blue); and 6 sections in the set N: *.text* (dark purple), *.data* (purple), *.bss* (dark pink), *.heap* (orange), *.stack* (yellow), and backup (white). Because of the page write granularity of Flash memory, it can only contain read-only sections and be used for backup. The exploration was performed on this set for both applications with different wake-up periods T_i: 0.1 second, 1 second, and 1 day for the light applications; 5 seconds, 10 minutes and 1 day for the heavy application. For each run, we extracted for analysis, among thousands of generated mappings, three relevant solutions: HYB (hybrid), NV (fully non-volatile) and BEST (optimal solution). The hybrid solution uses a SOT memory for *.text* section and backup content plus an SRAM memory for all other sections. The fully non-volatile solution uses a single SOT memory for all sections. We use these two solutions as a baseline to represent typical microcontroller architecture with SOT replacing Flash and both Flash and SRAM. Finally, the optimal solution is the one that results in the lowest average power consumption during T_{eval}. Each line of the graph represents a $N \mapsto M$ mapping solution on the left side, and the associated average memory power consumption on the right. For example, the first line of Fig. 5(a) represents the *light* application mapped in a hybrid architecture where *.text* section and backup content are located in a 16kB SOT memory while *.bss*, *.heap* and *.stack* sections are located in an 8kB SRAM. The corresponding average power for a 100ms period is 0.2 mW.

We observe for both applications, on the lowest considered duty cycles (1 day period), that the best solution relies on a Flash for the *.text* section. In the absence of precise figures, it should be noted here that we have not taken into account the static consumption $P_{statOff}$ of the Flash: we can therefore assume that this is an idealised consumption for this technology. This encourages the choice of Flash for very low duty cycle applications, even though its read energy is higher than that of other technologies. For the *light* application, for a 100ms and 1s period the BEST solution selected by MemCork was a hybrid solution with *.text* section in an 8kB SRAM and all other sections and backup content in two 8kB SOT. This solution allows up to a 23% reduction in average energy consumption compared to a fully non-volatile solution. Considering the *heavy* application, for a 5s period, the best solution is a 256kB SOT memory for *.text*, *.data* and *.bss* sections, a 64kB SRAM for *.heap* and *.stack* and a 64kB SOT for content backup/restore. This solution allows a 15% reduction in average memory power compared to a fully non-volatile approach. It combines the low active static power of SRAM in `On` mode during active phase and the low static power of SOT in `Off` mode. For a 10 min period, for the BEST solution, *.heap* and *.stack* sections are now directly mapped to the 64kB SOT and the use of a 64kB SRAM is no longer optimal. The resulting optimised fully non-volatile solution uses two non-volatile memories that are smaller than the single memory of the traditional solution, allowing a 12% reduction of the power consumption. For such a small duty cycle, this solution takes full advantage of the low inactive static power of SOT technologies. The solutions generated by the memory mapper are optimised to avoid large unused memory segments, which reduces the aggregate size of the memories.

VI. Related works

To the best of our knowledge, this is the first work to propose a systematic and automated exploration framework of hybrid memory MCU with custom data placement. Prior works target architecture exploration and/or data placement for cache-based System-on-Chip (SoC) with hybrid memory technologies. These works also rely on a methodology for the evaluation of performance, energy, and/or area of hybrid memory-based SoCs [13] [14]. In [15] and [16], the authors use an FPGA emulation method to evaluate the performance of hybrid memory architecture. However, these works target high-performance computing architectures and applications, which differ significantly in the intermittent edge computing domain targeted in this work. Patrigeon *et al.* [17] developed a memory technology exploration methodology that targets MRAM integration in ultra-low power MCU architecture. They also use an FPGA to emulate a complete microcontroller architecture in a real-world environment and evaluate memory consumption for different technologies using activation-based estimations. Assuming a two-section memory (one section for the application code and one section for the rest), they observe that a hybrid configuration with MRAM for code and SRAM for data is 26% more energy efficient than a traditional Flash + SRAM approach. However, this work only considers very basic applications and low complexity benchmarks. Furthermore, they only perform technology exploration for two memories with a constant size (128kB for code and 16kB for data) regardless of the application size.

VII. Conclusion

We introduced MemCork, a systematic and automated framework for hybrid memory architecture and data mapping exploration. This tool allows identifying the most energy-efficient solution for a given application based on a fine-grained tracking of the memory activity, performed on a technology-agnostic FPGA prototype running applications in real-time and in a real-life environment. We used MemCork on two typical applications to study the influence of the application type and its environment on the most energy-efficient solution. MemCork found automatically solutions for both types of applications, reducing memory consumption by up to 23% compared to a naive memory architecture based on a single NVM. We expect MemCork to encourage the use of emerging memory technologies in the design of low-power MCU for intermittent computing at the edge.

Acknowledgment

The authors acknowledge the support of the French National Research Agency (ANR), under grant ANR-19-CE24-0017 (NV-APROC project).

References

[1] G. Patrigeon, "Ultra low-power integrated systems for the internet-of-things," Ph.D. dissertation, Univ. Montpellier, 2020.

[2] M. Zwerg, A. Baumann *et al.*, "An 82 μA/MHz microcontroller with embedded FeRAM for energy-harvesting applications," in *Proceedings of ISSCC*, 2011.

[3] S. Khanna, S. C. Bartling *et al.*, "An FRAM-based nonvolatile logic MCU SoC exhibiting 100% digital state retention at VDD = 0 V achieving zero leakage with <400-ns wakeup time for ULP applications," *IEEE JSSC*, 2013.

[4] F. Su, Y. Liu *et al.*, "A ferroelectric nonvolatile processor with 46 μs system-level wake-up time and 14 μs sleep time for energy harvesting applications," *IEEE TCAS1*, 2016.

[5] Y. Liu, F. Su *et al.*, "A 130-nm ferroelectric nonvolatile system-on-chip with direct peripheral restore architecture for transient computing system," *IEEE JSSC*, 2019.

[6] F. Su, W.-H. Chen *et al.*, "A 462GOPs/J RRAM-based nonvolatile intelligent processor for energy harvesting IoE system featuring nonvolatile logics and processing-in-memory," in *Proceedings of VLSI Symposium*, 2017.

[7] Z. Wang, Y. Liu *et al.*, "A 65-nm ReRAM-enabled nonvolatile processor with time-space domain adaption and self-write-termination achieving >4x faster clock frequency and >6x higher restore speed," *IEEE JSSC*, 2017.

[8] T. F. Wu, B. Q. Le *et al.*, "A 43pJ/cycle non-volatile microcontroller with 4.7 μs shutdown/ wake-up integrating 2.3-bit/cell resistive RAM and resilience techniques," in *Proceedings of ISSCC*, 2019.

[9] N. Sakimura, Y. Tsuji *et al.*, "A 90nm 20MHz fully nonvolatile microcontroller for standby-power-critical applications," in *Proceedings of ISSCC*, 2014.

[10] M. Natsui, D. Suzuki *et al.*, "An FPGA-accelerated fully nonvolatile MCU for sensor-node applications in 40nm CMOS/MTJ-hybrid technology achieving 47.14μw operation at 200MHz," in *Proceedings of ISSCC*, 2019.

[11] M. B. Tahoori, S. M. Nair *et al.*, "A universal spintronic technology based on multifunctional standardized stack," in *Proceedings of DATE*, 2020.

[12] T. Soriano, D. Novo *et al.*, "An FPGA-based emulation platform for edge computing node design exploration," in *Proceedings of RSP*, 2021.

[13] T. R. Kumar, C. Ravikumar *et al.*, "Memory architecture exploration framework for cache based embedded SOC," in *Proceedings of VLSID*, 2008.

[14] M. Katsaragakis, L. Papadopoulos *et al.*, "Memory management methodology for application data structure refinement and placement on heterogeneous DRAM/NVM systems," in *Proceedings of DATE*, 2022.

[15] K. T. Malladi, M.-T. Chang *et al.*, "FAME: A fast and accurate memory emulator for new memory system architecture exploration," in *Proceedings of MASCOTS*, 2015.

[16] M.-T. Chang, I. S. Choi *et al.*, "Performance impact of emerging memory technologies on big data applications: A latency-programmable system emulation approach," in *Proceedings of GLSVLSI*, 2018.

[17] G. Patrigeon, P. Leloup *et al.*, "FlexNode: a reconfigurable internet of things node for design evaluation," in *Proceedings of SAS*, 2019.