Swapping of FPGA-runtime data in non-volatile memory

Auslagerung von FPGA Laufzeitdaten in nicht-volatilen Speicher
Bachelor-Thesis von Jan Hohmann aus Offenbach am Main
Tag der Einreichung:

1. Gutachten: Prof. Dr.-Ing. Andreas Koch
2. Gutachten: Dr.-Ing. Andreas Engel

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Swapping of FPGA-runtime data in non-volatile memory
Auslagerung von FPGA Laufzeitdaten in nicht-volatilen Speicher

Vorgelegte Bachelor-Thesis von Jan Hohmann aus Offenbach am Main

1. Gutachten: Prof. Dr.-Ing. Andreas Koch
2. Gutachten: Dr.-Ing. Andreas Engel

Tag der Einreichung:

Erklärung zur Bachelor-Thesis gemäß § 23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Jan Hohmann, die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.
Mir ist bekannt, dass im Falle eines Plagiats (§ 38 Abs.2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten düfen nur einmal wiederholt werden.
Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung überein.

Darmstadt, den 3. April 2018

(Jan Hohmann)

Zusammenfassung

Mobile Sensorknoten und Geräte im Internet of Things (IoT) haben häufig eine begrenzte Energieversorgung und sind auf Batterien angewiesen. Daher ist es von essentieller Bedeutung, dass der Energieverbrauch dieser Geräte so gering wie möglich ist. Dies kann durch die Verwendung moderner Microcontroller Units (MCUs) mit tiefen Schlafzuständen erreicht werden. Sie sind aufgrund ihrer geringen Rechenleistung aber nicht für rechenintensive Anwendungen geeignet.

Application Specific Integrated Circuits (ASICs) hingegen können spezifische Berechnungen sehr schnell erledigen und können bei entsprechendem Design auch sehr energieeffizient sein. Dafür sind sie aber vergleichsweise teuer im Design und sind daher für kleine Stückzahlen nicht rentabel. Auch können sie nicht nachträglich für eine andere Aufgabe angepasst werden. Deshalb werden sie in dieser Bachelor-Thesis nicht weiter behandelt.

Auch Field Programmable Gate Arrays (FPGAs) können spezifische Berechnungen sehr schnell erledigen, sie unterstützen aber keine tiefen Schlafzustände und benötigen daher im unbenutzten Zustand wesentlich mehr Energie als eine MCU. Die in dieser Thesis verwendete MCU benötigt beispielsweise in tieferen Schlafzuständen 2.5 μW, während der FPGA mindestens 52.8 μW benötigt.

Um den Energieverbrauch von FPGAs weiter zu senken, müssen sie daher vollständig abgeschaltet werden. Da dies meist vom Hersteller nicht vorgesehen ist, muss der FPGA dafür von seiner Stromversorgung getrennt werden. Laufzeitdaten, die sich im internen flüchtigen Speicher des FPGAs befinden, müssen dabei jedoch in einem FPGA-externen Speicher gesichert werden, um nach einem Neustart des FPGAs auch weiterhin verfügbar zu sein. Durch diesen Mehraufwand ist ein Abschalten nur dann sinnvoll, wenn der FPGA einige Sekunden bis Minuten im Ruhezustand verbleibt.

Im Rahmen dieser Arbeit werden verschiedene Typen von nicht-flüchtigem Speicher auf ihre Tauglichkeit als Pufferspeicher zwischen den einzelnen Berechnungszyklen verglichen. Auch müssen verschiedene Arten von Schaltern für das Abschalten des FPGAs evaluiert werden, da Schaltstrom und Übertragungswiderstand Einfluss auf den Energieverbrauch des Gesamtsystems haben. Mit den ausgewählten Bauteilen wird ein Prototyp gebaut. Auf diesem wird eine Backup- und Wiederherstellungsfunktion für den FPGA entwickelt und diese an eine Beispielanwendung angebunden. Danach wird durch Messungen ermittelt, unter welchen Umständen ein Abschalten des FPGAs effizienter ist als ein Übergang in einen Standby-State. Um die Messungen steuern zu können und die korrekte Funktion der Beispielanwendung zu verifizieren, wird zusätzlich ein Serial Peripheral Interface (SPI) Controller implementiert, der die Kommunikation mit einer MCU übernimmt. Die MCU überwacht die Berechnungen der Beispielanwendung und steuert die Schalter für die Stromversorgung des FPGAs.

Durch die Verwendung von Ferroelectric Random-Access Memory (FeRAM) als Backup-Speicher und eines Metal-Oxide-Semiconductor Field-Effect Transistors (MOSFETs) als Schalter, ist schon bei einer Inaktivität von wenigen Sekunden ein Abschalten des FPGAs effizienter, als ein Wechsel in den Standby-State. Zwar benötigt der Backup- und Wiederherstellungsprozess mit circa 7 ms deutlich mehr Zeit, als ein Wechsel in den Standby-State, der nur 200 μs dauert. Dafür kann aber die Leistungsaufnahme des FPGAs nach dem Backup auf null Watt gesenkt werden. Der gesamte Backup- und Wiederherstellungsprozess für 450 Byte an Daten braucht mit 52.6 μJ weniger Energie als die 52.8 μJ, die der FPGA im Schlafzustand pro Sekunde benötigt.

Abstract

Mobile sensor nodes and Internet of Things (IoT) devices are often connected to a finite power source and rely on batteries. Therefore, it is very important, to keep the power consumption as low as possible. This can be achieved, by using modern Microcontroller Units (MCUs), which support low power states. But due to their limited computing power, they are not suitable for compute-intensive applications.

Whereas Application Specific Integrated Circuits (ASICs) can handle certain calculations very fast. If they are designed properly, they can also be very energy efficient. With the drawback, that the development of their hardware design is comparatively expensive and thus they are not cost-efficient in small numbers. Furthermore, they cannot be adapted to new tasks afterwards. That is why they are not addressed in this bachelor thesis.

Field Programmable Gate Arrays (FPGAs) can also handle certain calculations very fast, but they do not support low power states. This results in a higher power drain in idle state, than an MCU would have. The MCU used in this thesis, for example, needs 2.5 μW in lower power states, while the FPGA still needs 52.8 μW.

To further lower the energy usage of FPGAs, they have to be switched off completely. Since this is usually not supported by the vendor, the power supply of the FPGA must be cut. Any runtime data residing in the internal volatile memory of the FPGA must be backed up into some FPGA-external memory. Otherwise it would be lost after a power cycle of the FPGA. Due to this overhead, a power cut is only reasonable, if the FPGA can stay a few seconds or minutes in this suspended state.

Within this thesis, several technologies of non-volatile memory are compared in terms of their suitability as buffer memory for in between calculation cycles. Also different types of switches are evaluated. They are used to switch the FPGA. Differences in their switching current and transfer resistance have influence on the energy usage of the overall system. A prototype is built with the selected components to determine, in which cases a power cut of the FPGA and the swapping of the data into an external memory is more efficient, than a transition into a standby state would be. For this purpose, backup and restore functionality is implemented for the FPGA. A sample application is used to generate pageable data. Then the power drain is measured, to compare the energy needed by the swapping process with the standby drain. To control the evaluation and to monitor the sample application, a Serial Peripheral Interface (SPI) controller is implemented. It is responsible for the communication with an MCU. This MCU supervises the sample application and controls the power supply of the FPGA with a switch.

Due to the use of Ferroelectric Random-Access Memory (FeRAM) as backup memory and a Metal-Oxide-Semiconductor Field-Effect Transistor (MOSFET) as switch, a suspend duration of only a few seconds is enough to make the power cut more efficient than a transition into the standby state. The backup and restore process takes about 7 ms, which is much more as the 200 μs a transition into the standby takes. But this allows to reduce the power drain to zero watt after the backup is finished. The whole backup and restore process takes 52.6 μJ for 450 byte of data. This is less than the 52.8 μJ the FPGA needs in its standby state.

Contents

1 Introduction

Wireless Sensor Networks (WSNs) have gained in importance in recent years. These clusters from a few through to many smart sensor devices (further called sensor nodes) can be utilized in many different areas and form a large part in the Internet of Things (IoT). The sensor nodes are typically small, have limited computing power, and are inexpensive compared to traditional sensors. A sensor node typically has a Microcontroller Unit (MCU) with limited computing capabilities and one or more sensors, which are connected to it. In WSNs a sensor node transmits its collected data wirelessly and is not attached to a theoretically inexhaustible power source (power adapter). Most sensors use batteries, either as buffer for a staggering power source, such as a solar panel, or as the only power source. Both cases require the nodes to ration its available energy.

There are several applications for WSNs today. They are used to track people, animals, and vehicles. They monitor the weather, patients, and factories. They can be integrated into existing infrastructure, or can work completely independent from it [1].

To fit the needs of compute-intensive sensor nodes with a low power drain, the Hardware-Accelerated Low Energy Wireless Embedded Sensor Node (HaLOEWEn) was developed [2]. It combines the efficiency and performance of a Field Programmable Gate Array (FPGA) with the low power drain of an IEEE 802.15.4 based Radio Frequency System-on-Chip. This allows the platform, to perform complex calculations, such as lossless compression with the FPGA, while using a low power MCU for wireless communication. The FPGA can be powered down, while no operations are pending to keep the power consumption low. With an Application Specific Integrated Circuit (ASIC) the efficiency could be further improved, but for a scientific prototype, these are too expensive in design and construction.

Compared with an MCU, the standby power consumption of an FPGA is much higher. While the MCU is needed for wireless communication and therefore cannot be omitted, the FPGA can be switched off completely to overcome the high power consumption while it is idle. Data residing in the distributed registers and block-RAM of the FPGA must then be backed up if it is needed later. Otherwise, it will be lost due to the volatile nature of the integrated memory.

A further requirement to decrease the power consumption of the FPGA is to use flash-based FPGAs instead of Static Random-Access Memory (SRAM)-based. Flash-based FPGAs can be initialized faster, since they have their persistent configuration pattern on-chip and do not need to be programmed with specifications residing in external memory each startup. The power consumption of the total system is therefore lower compared to an SRAM-based FPGA, since no additional memory needs to be supplied. And it is possible to disable the FPGA on a regular basis. These benefits come along with drawbacks in package density and with a higher price for the chip itself [3].

1.1 Contribution of this Work

The primary aim of this work is to minimize the power consumption of the sensor node, by switching off the FPGA in idle times, while still conserving the relevant content of its on-chip memory. Therefore, a suitable memory chip must be selected to keep runtime data available over multiple power cycles of the FPGA. This memory has to be non-volatile and

should consume as little energy as possible. That means it must be fast, while having a low power consumption.

Furthermore, different types of switches are compared to find the most efficient way to switch off the FPGA. Those two components are connected to a Microsemi IGLOO development board [4], which is used to build a prototype. A Texas Instruments CC2530 MCU is used to control and monitor the FPGA. It uses an Serial Peripheral Interface (SPI) bus to communicate with the FPGA. Therefore, an SPI slave module is implemented on FPGA-side.

To backup and restore the relevant data to the external memory a swap-handler module is developed. This module takes the backup worthy data from the application modules and writes them to the memory. After a reset of the FPGA, it restores the data back to the application modules. The modules for themselves decide, which data is required to be saved. While the backup or restore process is active, all other operations of the FPGA are halted.

As a sample application a finite impulse response (FIR) filter is implemented, which processes data provided by the MCU. The MCU monitors the results of this filter, to verify that the backup and restore processes work flawlessly. The final results are transferred to an attached computer.

After this verification of the functionality the setup is ported to the HaLOEWEn platform, which is described in the next chapter. On this platform the power consumption is measured, to examine the additional consumption for the backup and restore process. This energy consumption overhead is compared with the specifications of the FlashFreeze mode. The power drain of the MCU is not regarded, since it has to be active anyway.

1.2 Structure of this Work

The main parts of this thesis are organized as follows. Chapter 2 will take a look at existing approaches to minimize the power usage of sensor nodes. It also addresses hibernation techniques and ways to minimize the data to be transferred. The HaLOEWEn platform is presented in this chapter, too. Chapter 3 presents currently available non-volatile memory (NVM) technologies and compares them in terms of power consumption. In this chapter the decision for a technology is made. Chapter 4 describes the implementation of an energy efficient HaLOEWEn prototype with reduced standby power consumption. It compares different kinds of switches and describes the implementation details of the swapping controller. Chapter 5 documents the measurement setup and presents the evaluation results. Also the conditions which must be fulfilled for the implementation to be more efficient than the regular standby-state, are described here. Chapter 6 summarizes the results. In addition, it shows optimization potentials, which, due to the high complexity of the subject, are not handled in this thesis.

2 Related Work

2.1 HaLOEWEn

The hardware architecture of this work is based on the HaLOEWEn implementation in version 3 as it can be seen in figure 2.1. It combines a software-programmable RF-SoC and an FPGA. The RF-SoC is a Texas Instruments CC2531 with an integrated 2.4 GHz IEEE 802.15.4 transceiver and an 8 bit MCU, which is based on the Intel 8051 architecture[5]. It uses the same protocol stack like many ZigBee, 6LoWPAN, WirelessHART and ISA 100.11a devices. The main tasks of the CC2531 are to handle the radio and other low complex computations like time-keeping. To preserve energy, the radio is switched off while not transmitting.

As FPGA a flash-based Microsemi IGLOO AGL1000 is used. This chip has a standby power drain of 52.8 μW[6]. For a quick dynamic power management it is beneficial that this FPGA can be woken up in a few microseconds. But the needed standby current is still at least an order of magnitude larger than the current drawn by typical sleeping WSN software processors. The CC2531 in comparison only has a power drain of 2.5 μW in its second lowest power mode and 1.0 μW in its lowest mode.

The FPGA is connected to the MCU with an SPI bus and some additional multifunction wires. All these connections use the third Input/Output (I/O)-bank (I/O-bank #2) of the FPGA. It is set to 2.5 V and cannot be switched off or measured. The other 3 I/O-banks can be driven with either 3.3 V or 2.5 V. They can be individually switched on and off and it is also possible to measure the power drain of every I/O-bank separately. Sensors can be attached to these banks via multiple board-to-board connectors.

The whole system can be powered with a 3 V battery. And it is possible to supply the FPGA with an separate power supply. [2]

2.2 Energy Conservation in Wireless Sensor Networks

There are several approaches to minimize the power consumption of WSNs. Since the radio is often the unit with the highest power consumption, there are several ways to minimize its utilization. The so called *Duty Cycling* approach turns the radio in a low power state

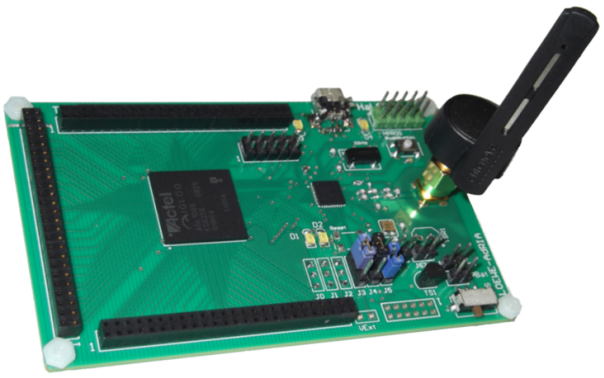

Figure 2.1.: HaLOEWEn v3 Prototype[2]

whenever it is not needed. The HaLOEWEn architecture already takes advantage of the technique. In networks, where single nodes do not only transmit their own collected data, but also need to forward information from other nodes it is important, that adjacent nodes synchronize their wake-up periods to make transmissions feasible. In networks, where nodes are redundant, the sensing area can be divided into virtual grids. Nodes in the same cell can rotate their wake-up times, so only one node is active at a time. An alternative approach is to use coordinator nodes, which are responsible for routing and forwarding. A coordinator node stays permanently awake, while other sensing nodes can go back to sleep after each measurement. The coordinators organize themselves autonomous.

There are also *Data-driven* approaches to conserve energy. One way is to minimize the data to be transmitted. Either by compressing the collected data or by reducing the measuring points. The missing data can then be supersampled. Both approaches need more computation than a simple collection of the data, but they reduce the needed air-time of the transmission. [7]

The selection of the MCU can also have a huge impact on the power consumption of the whole system. The variation in power consumption between different products is in the range of several hundred milliwatts. To further reduce the power consumption of the MCU, Dynamic Power Management (DPM) can be used. This technique can put the microcontroller in a low power state, while it is idling. But the utilized software has to be adapted to this. Since often only a timer is still running, there cannot be an event driven wake-up. The MCU can only wake up periodically to perform calculations. [8]

At last there is the relative new way of using hybrid sensor nodes, which contain a FPGA for compute intensive tasks, while using the MCU only for management and wireless transmission. The power of the FPGA can be used to compress the data or to further analyze it so the raw data does not have to be transmitted. These boards may use more energy than implementations with a low-power MCU, but they have far more computational power while being more efficient than an MCU with comparable processing power. [9] [2]

2.3 Data swapping

To back up the data from volatile memory into the NVM, there is the need of a backup process. The simplest way is to back up the whole memory before each shutdown. This is inefficient in terms of energy and time consumption. And time consumption also means energy consumption. For non-hybrid platforms there is already a solution based on statistics based dead blocks prediction (SBDP). This platform uses a capacitor to provide energy after a power loss to backup relevant data until reboot. To minimize writing time after the power loss, an adaptive writeback scheme (AWS) was designed. With little overhead in performance and energy a preemptive write-back is fulfilled. This reduces the backup time and therefore decreases the size needed of the capacitor. But since there is a chance that not all relevant information are backed up correctly, the technology is not directly applicable to this work. [10] Also the memory has an overhead to address and access its data so it is less efficient to read or write non-sequential data.

3 Non-Volatile Memory

3.1 Flash

Since its introduction in the mid-1980s flash memory gained much popularity. In comparison to Erasable Programmable Read-Only Memories (EPROMs), it does not need to be erased completely before being rewritten. Depending on the architecture, it can be written in single words or only in pages. There are two types of flash memory available. NAND flash is available in high densities, but its design does not allow addressing single bytes. Whereas NOR flash is byte addressable with the penalty of a lower density. NOR flash can also replace Read-Only Memories (ROMs) without a change in the interface.[11]

3.1.1 NAND flash

NAND flash is currently the most compact available memory technology. With triple level cells there are chip sizes up to 6 terabit possible [12]. But these flash cells do not have a high endurance. While single-level cells (SLCs) can handle up to 10k program/erase cycles [13], multi-level cells (MLCs) can break after 3k program/erase cycles. Triple-level cells (TLCs) have an even worse endurance with only 500 program/erase cycles [12]. Every level increases the density of the chip with quad level cells as the densest currently available packaging. The level describes how many bits can be stored in a single cell [14].

The single cells are arranged in pages and unlike in NOR flash, the cells cannot be individually addressed. A SLC flash page typically has a size of 512, 2048 or 4096 bytes. Multiple pages are combined to blocks. The read operation works on page level. To get the data of a single cell, the whole page must be read. The write operation also works on page level, but only previously erased pages can be written. So to update a single byte the whole page must be read, erased, and rewritten. A simple write (without a preceding erase) needs an energy of $4\,\mu$J for a 512 byte page, which is the minimal addressable unit size of a Toshiba 16MB NAND flash [15]. While the read process without addressing overhead only needs 1.69 nJ per byte [16].

The erase operation only works on block level. To clear a page, the whole block must be erased and unrelated pages have to be rewritten. An alternative would be the implementation of a wear leveling function, which only writes data in empty pages. If the used memory is large enough, this would make the erase process expendable and preserves energy.

But even with the implementation of a wear leveling function, NAND flash stays really expensive in terms of energy consumption. Therefore, it will not further be discussed in this work.

3.1.2 NOR flash

NOR flash was first proposed in the 1980s. It was developed as a pin-compatible replacement for Erasable Programmable Read-Only Memory (EPROM). Every word can directly addressed, read and written. Alternatively there are flash chips with an SPI available, which can only be addressed on page level. While EPROM needs an UV-light source to be erased, NOR flash is electronically erasable at block or even page level. But the erase

process in NOR flash takes several times longer than in NAND flash, since every byte must be previously written with zeros. [17]

NOR flash has a durability of up to 10^5 program/erase cycles per cell. Depending on the implementation it needs about 1 μJ for a single write on a serial NOR chip with a page size of 256 byte [15].

3.2 Phase Change Memory

Phase change memory (PCM) (also called PCRAM or PRAM) is one of the emerging memory technologies. Its key characteristics are a high density, a data retention time of over 10 years and a write endurance in magnitude of 10^9. While read operations in Flash are slightly faster than in PCM, write and erase operations are significantly faster in PCM. The technology is based on the physical effect, that the electrical resistance of a material depends on its phase. If a material is in crystalline phase, its resistance is several magnitudes lower than if it is in amorphous phase. To change the phase of a material it has to be heated up. A material in amorphous phase can be transformed into crystalline phase by heating it up to a temperature between the crystallization temperature and the melting temperature for a time period long enough to crystallize. Since the amorphous material has a high resistance, the Poole-Frenkel Effect has to be used. It lowers the resistance by applying a electrical field to the material, so a crystallizing current can be reached [18]. The crystalline phase is called the SET state. To reset it back into amorphous phase it has to be melted and then quickly quenched. This is achieved by applying a large current to the material. The amorphous phase is called the RESET state. To read the value of a cell, its resistance is measured with a low current.

While almost every material has these two phases only a few are suitable for data storage technologies. The difference in resistance between amorphous and crystalline phase must be in a specific range. The crystallization temperature has to be above the operational temperature and the phase should be changeable in nanoseconds. These requirements narrow the usable materials down to a few alloys basing on germanium and silicon. [19]

Current phase change materials need up to 512 nJ per byte written,]which make them more energy expensive than NAND flash if more than 8 byte need to be written in a flash page. The read operation however is very cheap in terms of power consumption. It only needs 0.6 nJ to read a single byte[20]. This can be optimized with a data-comparison write scheme, which will compare the data to be written with the existing data on bit level, so only required transformations will be done. With this technology up to 64 bits can be set, before PCM requires more energy than NAND flash, which makes it still very expensive.

3.3 Magnetoresistive memory

An other non-volatile memory technology is Magnetoresistive Random-Access Memory (MRAM). MRAM-cells store their information in the polarization of a magnetic field. They use the effect, that the resistance of the magnetic material depends on the polarization of its surrounding field. If the polarization of the sensing layer is parallel to the surrounding field, its resistance is low and the bit is interpreted as a "1". If the polarization is anti-parallel to the surrounding filed, the layers resistance is high and the bit is interpreted as a "0". [21]

MRAM needs less energy than flash or PCM. In read modes the module needs typically 25 mA. At a frequency of about 22 MHz it will be 3.7 nJ per byte. In write modes the

MRAM needs typically a current of 55 mA. This results in an energy of 8.2 nJ consumed per byte written and in a maximal data rate of 176 Mbit. The standby current of 7 mA is too high to be ignorable, so the device needs to be switched off if it is not in use. [22]

There are newer approaches based on Magnetic Tunnel Junctions (MTJs), which need less energy, but they are not yet available as commercial products. This technology uses two magnetic layers: A pinned layer with a constant field direction, and a free layer, which is used to store the data by alternating its polarization. To read the information from the cell, a constant voltage or a constant current has to be applied. If a constant voltage is applied, the current difference is read with the aid of a comparator. If a constant current is applied, the voltage difference needs to be amplified to extract the state.

To write data into a MRAM cell, the polarization of the free layer must be altered. This can be done the traditional way by applying a directed external field at the cell, or with the spin-torque phenomenon, which uses spin polarized currents to alter the field orientation. This technology can operate with lower write currents than the traditional one, which results in a write energy of 2.9 pJ per byte without considering peripheral needs. [23]

3.4 Ferroelectric memory

Ferroelectric Random-Access Memory (FeRAM) is a comparably old technology. It works with the changeable polarization of ferroelectric crystals. A thin ferroelectric film is used to store the data. Data is written into a cell by applying a magnetic field, which changes the polarization of the material. The so written bits are now non-volatile and even insensible to radiation. A problem is, that the readout of the data is destructive, so read data must be rewritten. To read from a cell, a zero is written and by sensing the needed current it can be distinguished, if the cell previously contained a one or a zero [24].

Since the early research in the 1950s the technology was improved in terms of density, endurance, and retention time. Currently it is available in a package size up to 4 Mibit. With an endurance of 10^{14} read/writes it should be sufficient for most applications. Even though every read includes a write operation, the typical active current of 2.5 mA at 40 MHz is still lower than the needs of an MRAM or even PCM module. A single byte read or write only needs 1.5 nJ with an SPI bus[25] or 3 nJ with an parallel interface[26]. Depending on the bus frequency, the maximal data rate is up to 40 Mbit/s. The standby current of 150 μA may be negligible, but since the memory is non-volatile it can also be switched it off. With a lowered frequency of 25 MHz the FeRAM can also work with a voltage of 2.5 V instead of 3.3 V. This reduces the requirements for the utilized platform.

3.5 3D XPoint™

3D XPoint™, also known as Intel® Optane™or Micron QuantX™, was not available on the free market while the decision for the used memory technology was made. Solid State Drives (SSDs) with Intel Optane technology are currently as of March 2018 becoming available, but the memory chips themselves are not distributed. Also, there are no datasheets available, but the SSDs are promoted to be faster than current flash based drives[27]. No further information about this memory technology are currently available and since the chips are also unavailable this technology will further be ignored.

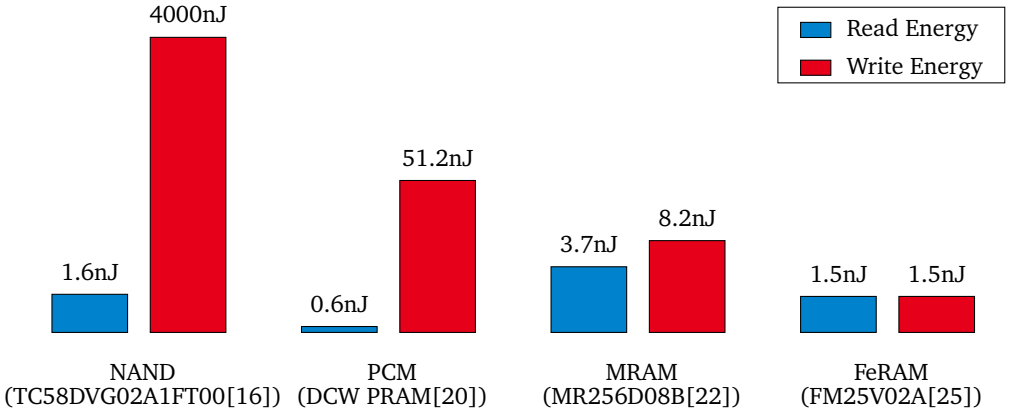

Figure 3.1.: Energy consumed per byte by non-volatile memory
Without erase operations

3.6 Conclusion

Compared with the other shown technologies, FeRAM is best suited as swapping memory. As to be seen in figure 3.1 it has the lowest energy usage per byte written and the second lowest per byte read. Because of the cheap write process, there is no need to implement a complex differential of incremental backup process. Also the memory pages can be overwritten without a separate erase operation, which would be needed with Flash and PCM.

Furthermore, it can be driven with 2.5 V, which is the main voltage of the HaLOEWEn board. And its typical supply current of 2.5 mA is low enough to be directly driven by the I/O-bank of the FPGA.

If the power-consumption of the FPGA itself is considered, the higher speed of the MRAM could make a difference. This is not further discussed in this work. Also if only a few bytes change every cycle, FeRAM is not the best option. In this case PCM in combination with an incremental backup scheme should be considered.

4 Implementation

4.1 FPGA Power Cutting

To decrease the power consumption of the sensor node the integrated FPGA has to be switched off. Therefore, it has to be disconnected from its power source. The Microsemi IGLOO AGL1000 has two supply voltages. Its core is powered with 1.2 V or alternatively with 1.5 V. The second supply voltage is used to power the I/O banks. This voltage can be configured to be 2.5 or 3.3 Volt [6]. In this work, the ARM Cortex-M1-Enabled IGLOO Development Kit board is used for a proof of concept. The core voltage is set to 1.2 V and the I/O voltage is set to 3.3 V [4]. For power-consumption measurement and further validation the HaLOEWEn v3 board [2] is used.

These comparable low voltages make it challenging, to find a suitable semiconductor switch. Alternatively it is possible to switch off the linear regulators. The line regulators are powered with 3.3 V. This is a voltage, that is easier to switch for a Metal-Oxide-Semiconductor Field-Effect Transistor (MOSFET), but it has the drawback, that the capacitors, which are located after the regulators, need to be recharged every time, the FPGA is switched back on.

The power consumption of the FPGA while active can be estimated approximately 30 mW [2][Table 17]. While the FPGA is suspended, it only needs 53 μW. A simple strand with a profile of 0.8 mm^2, for example, has a resistance of 20 mΩ per meter. This causes that a wire of 10 cm length has a power dissipation of 2 μW. So the suspended FPGA consumes only 25 times more energy than a simple wire, which makes line resistance a problem.

Bipolar Transistors

Bipolar junction transistor (BJT) were the first transistors widely used in integrated circuits before they were replaced by the Complementary metal–oxide–semiconductor (CMOS) technology. A bipolar transistor is current regulated, which means that a current between Base and Emitter is needed, to switch the transistor on.

They need at least 1 mW switching current, to keep the FPGA powered on. This is about 15 times more, than the 53 μW the AGL1000 needs as standby power. To be effective, the FPGA has to have an off-time 15 times as long as its active time, just to compensate the energy consumption of the transistor. Since our setup hast two switchable voltages, this proportion even increases to 30 times.

Metal-Oxide-Semiconductor Field-Effect Transistors

In comparison to bipolar transistors, MOSFETs need much less energy to keep their state. They generate an electrical field to induce a tunnel from the source to the drain substrate. Therefore, its gate is charged with several nano-coulomb.

There are p-channel and n-channel MOSFETs. P-channel MOSFETs close the circuit, if the gate voltage is below the source voltage, while n-channel MOSFETs are activated with a higher gate voltage.

In our application a n-channel MOSFET can be used to switch the 1.2 V core voltage, while the I/O banks and the memory have to switched with a p-channel MOSFET.

	BJT	MOSFET	Signal Relay
Switching energy	0 μJ	54 nJ	300 μJ
Active power	60 μW	7 μW	32 μW

Table 4.1.: Overhead of the different switching technologies

The drain-source resistance ($R_{DS(on)}$) cannot be neglected. It increases the power needed in active state and has to be added to the active power consumption. A good n-channel MOSFET has a $R_{DS(on)}$ of 10 mΩ at 1.8 V [28]. This causes an additional power consumption of 7 μW, while the FPGA is active, assuming the 30 mW, declared in the datasheet, are completely used on the 1.2 V supply.

Since the HaLOEWEn platform does not provide voltages larger than 3.3 V, a p-channel MOSFET has to be used for theses lines. In this case a good $R_{DS(on)}$ value is approximately 11 mΩ at 2.5 V [29]. This results in an additional power consumption of 1 μW, while the FPGA is active, assuming the 30 mW are completely used on the 3.3 V supply.

The gate charge of 30 nC can be ignored since the effective energy of 54 nJ is lower than the energy needed to write the data into the external memory.

But the best transistors are only available in very small packages like the PowerPAK SC-70-6L. This makes them unsuitable for prototypes. With worse specifications (31 mΩ at 1.8 V), there are npn-transistors available on the market in TSOP6 packages.[30]

For the prototype smd adapters boards are needed, which will increase the contact resistance, but the impact of those is not predictable.

Signal Relays

Relays are electromechanical switches. A coil induces a magnetic field, which changes the state of the switch. There are bistable and monostable relays. While monostable relays need energy to keep their on-state, bistable relays only need to be powered a few milliseconds. Modern signal relays can be powered with voltages down to 3 V and can therefore be driven by TTL or CMOS logic devices. But the switching current of 66.7 mA is larger than the current, the ATMEGA MCU can provide through its I/O-pins [31]. A so called H bridge is needed to power the relays. The benefit of a relays in comparison to a MOSFET is, that the switched voltage is irrelevant. The drawback is that the switching energy of approximately 300 μJ is comparably high. With this energy, the FPGA can stay in suspend state for six seconds. Also the typical contact resistance of 50 mΩ is larger than that of the best transistors [31]. This results in a power dissipation of 32 μW at 1.2 V, with an assumed FPGA power consumption of 30 mW.

Conclusion

As it can be seen in Table 4.1, there is no technology without a penalty in power consumption. In comparison to the signal relay, the transistors do not need a significant amount of switching energy. And because of the high active power of the bipolar transistor, the MOSFET is the only reasonable choice.

4.2 Hardware Architecture

All functionality is implemented on a Microsemi AGL1000 development kit[4]. It provides the FPGA, FLASH memory, SRAM, and some light-emitting diodes (LEDs) to display debug

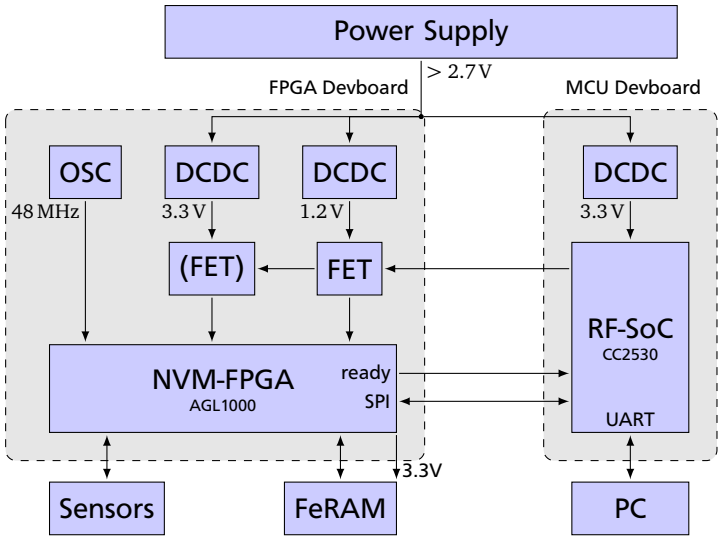

Figure 4.1.: Hardware Architecture of the first prototype based on the AGL1000 and CC2530 development kits

information. Several lines from each I/O-bank of the FPGA are provided through BT224 compatible boxed headers. The power is provided by a 5 V@3 A wall plug transformer. The power supply for the FPGA core can be cut with a jumper. This jumper is replaced by a Vishay Si4465ADY MOSFET [29], to make the power supply switchable. A 256-Kbit Cypress FM25V02A[25] FeRAM is used as FPGA-external swapping memory. It is connected as an SPI slave to the FPGA and can be operated with a clock of up to 40 MHz. With an operating voltage of 2 V to 3.6 V and a maximum current of 2.5 mA for read and write operations, it can be powered by a single I/O-pin of the FPGA. An additional switch is not needed.

A Texas Instruments SmartRF05 Evaluation Board with an integrated CC2530 MCU[5] is used to control the transistor and the FPGA. To enable communication between the MCU and the FPGA, the FPGA is connected as an SPI slave to the MCU. An additional “ready”-wire is used by the FPGA, to inform the MCU about the completion of the backup process. The whole setup is controlled by a host PC, which communicates via the universal asynchronous receiver-transmitter (UART)-protocol with the MCU. This hardware setup is visualized in Figure 4.1.

The MOSFET bracketed out is only included in the later used prototype based on the HaLOEWEn platform, since it is not needed to cut the power of the I/O-banks for the functional evaluation. Also the power supply is not shared between the development boards. They only have a common ground.

4.3 Data swapping

All runtime data residing in the FPGA is stored in its distributed registers and block-RAM. Due to its volatile nature, all information, that is needed after a power cycle of the FPGA, must be backed up prior to shutdown. Therefore, a swapping controller module is implemented, to copy all relevant information into an external FeRAM-chip. Besides this swapping controller module, there are application modules implemented on the FPGA, which can hold data, that has to be backed up. Directly after the device is reset, the swapping controller connects to the external memory and reads all stored information via SPI and forwards them to the application modules. To achieve this, all application modules

State	Bitmask	Description
IDLE	2'b00	Module should perform its work
BACKUP	2'b10	Module should provide data if addressed
RESTORE	2'b01	Module should restore data if addressed

Table 4.2.: Swapping controller states and their bitmasks

must be kept in a halt state, so that they do not corrupt the data, while it is still being restored. The swapping controller module is placed in a separate clock domain, to allow the FeRAM to be read and written with a maximum rate. Therefore the internal data bus, which is used to forward the data between the application modules and the swapping controller, has a configurable width. With a width of two, the clock of an application module can be half the clock of the swapping controller. After all data is restored, the controller ends the halt state and the application modules can continue their work at the exact same state, they were before the shutdown.

To shutdown the FPGA, a signal is issued by the MCU via an SPI command. The controller again halts the modules and collects their data, to write it to the external memory. In this process, the whole memory will be overwritten. There is no differentiation between altered and unaltered data. After the backup is finished the controller module signals the MCU, that the FPGA is now ready to be switched off. With the maximum frequency of 40 MHz, a whole backup or restore process with an 256 Kibit FeRAM takes no more than 6.6 ms.

4.3.1 Application Module Interface

Every application module that holds data, which must be preserved amongst power cycles, must implement an interface to the swapping controller. The listing in Appendix A shows a sample module, which contains all logic, that is required to backup and restore its data. The interface contains an incoming and an outgoing data bus, an address bus, and a status bus.

The status bus is used by the swapping controller, to notify the modules about active backup or restore processes. Its possible values are show in Table 4.2. While the state is not *IDLE*, all operations of the modules must be halted. The other states signalize, that a backup or restore process is in progress and that the modules should await their addresses.

The data and the address buses should have a width, configurable via external parameters. This allows a subsequent modification of the clock domains or the amount of data to be backed up. The configured address has to be globally unique. This means two modules must not use the same address for a word. So an address offset should be configurable to allow local addresses and to make it possible, to move a module to an other position in the external memory. It makes it easier to reuse a module in other implementations, since otherwise there is a risk, that two modules would use the same memory address range. Also a configurable offset allows multiple instances of the same module. When the offset values are configured, it should be minded, that there are no unused gaps in the address range. The address translation is done in line 27 and the check, if the provided address is in the module-specific range, is performed in line 55 of the in Appendix A provided listing.

The *DATA_IN* bus is only used during restore. Its data should be ignored, if the set address does not match the address range of the module, or if the swapping controller is not in restore state. If these constraints are fulfilled, the data can be restored into the local registers. An example is provided in lines 58 to 62 of the listing in Appendix A. The

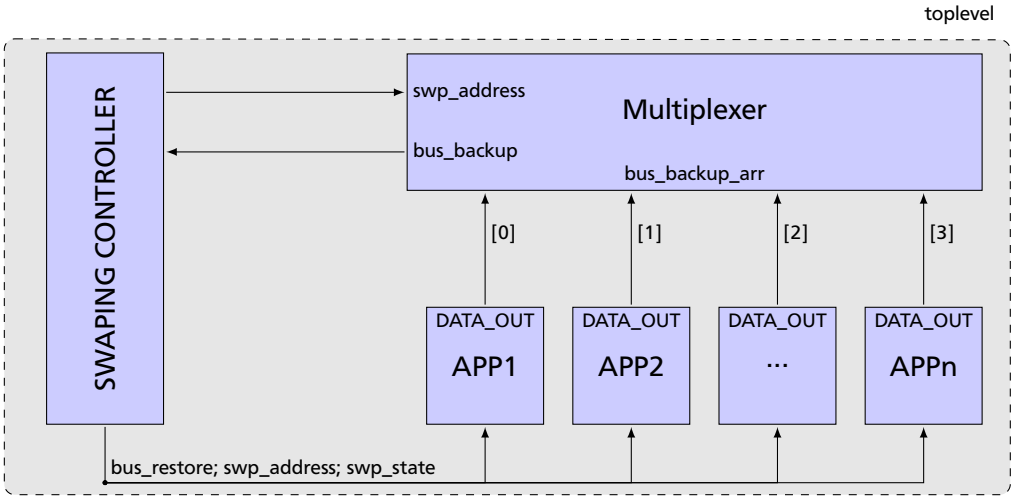

Figure 4.2.: Toplevel Design

address and data may change every clock cycle. This must be considered, if the data must be stored in slower memory.

To backup the data, the *DATA_OUT* bus is used. Again the global address provided by the swapping controller should be translated into local address. The data must be available in the same clock cycle as the address changes. This can be achieved by using a multiplexer, which uses the relative address, to provide the requested data. Data residing in the block-RAM must be prefetched, to be available in time. This prefetch should be done, while the address bus is set to one address before the actual data address. In the listing, all data is aggregated into the wire *datasum*, while the wire *datashift* in line 34 provides the requested word. If the address is not valid, the *DATA_OUT* bus may be set to zero.

4.3.2 Toplevel Requirements

The toplevel module, as to be seen in Appendix B, must provide the swapping controller direct access to the FeRAM. The external wires to the FeRAM use the same names, as they are provided in the pin definitions of the FM25V02A chip [25]. Additionally, the V_{DD} supply wire is also connected to the FPGA. Its internal name is *FRAM_POWER* and it gives the swapping controller control over the power supply of the FeRAM.

To inform the swapping controller about a pending shutdown, the *SHUTDOWN_TRIGGER* wire is used. This wire should only be set, if the FPGA has no pending calculations and can safely be shut down. It causes the swapping controller, to start a backup and to halt all other modules. When the shutdown is complete, the *SHUTDOWN_COMPLETE* wire is set by the swapping controller. In this state, the FeRAM is already switched off and the FPGA awaits its power to be cut.

The internal bus configuration between the swapping controller and the application modules is visualized in Figure 4.2. The *swp_state*, *bus_restore*, and *swp_address* wires can be directly attached to the application modules. For the *bus_backup* wire, a multiplexer is used to switch between the application modules. It uses the *swp_address* wire to select the correct application module and forwards its data to the swapping controller. In the listing, a helper array, called *bus_backup_arr*, is used, to simplify the multiplexer.

4.3.3 Logical Functionality

Backup

Before the FPGA can be safely switched off, the application data, residing in the internal registers and block-RAM, must be backed up. Therefore the swapping controller must be notified about the pending shutdown via the *SHUTDOWN_TRIGGER* wire. The swapping controller then powers the FeRAM and transmits a suspend signal to all application modules, to prevent a corrupt backup. It then writes the addresses, from zero up to the maximum defined address, in ascending order to the address bus. The application modules should now await their assigned addresses on the address bus. A multiplexer at toplevel ensures, that at a time only one module can transmit its data to the FeRAM. After all data is backed up, the swapping controller signals via the *SHUTDOWN_COMPLETE* wire, that the FPGA is now ready to be powered off. The FeRAM is also switched off, to conserve energy.

Restore

After the RESET line is asserted high, the controller powers the FeRAM and sets a signal on *swp_state* bus, that it will now issue a restore. This bus must be attached to all modules that should be backed up and restored. The modules must interrupt their normal operation, if the *BACKUP* or *RESTORE* signal is provided by this bus. After the restore is started, the controller reads sequentially data from the FeRAM and sends it over the *bus_restore* bus, while it increments the address. The sequential read is stopped, when the address reaches the value configured in *DATA_LEN*. While the address *0* is reserved for the controller, to do internal operation, any other address can be used by the application modules for their data. This address is used to identify the words on the internal data bus and can be different from the memory address for the same data. The width of the bus is configurable and should compensate different clocks, used by the controller and the application modules. It should be as small as possible, to prevent empty bits, which can occur when a module needs a number of bits, that is not divisible by the buswidth. The modules should use local addresses for their registers, so that every applicable global address maps on a local address. This allows a module, to be moved to a different position in the memory without a required modification of the module. When an address, used by the module, is set on the address-bus, the data from the data-bus should be written into the corresponding registers. If an address outside of the address range of the application module is provided by the address-bus, the data laying on the data-bus must be ignored, since the address may be assigned to another module. After all modules are restored, the controller removes the restore-signal from the control-bus and all modules can start working. At the same moment the FeRAM is disconnected from its power source, since it will not be needed until the next backup process.

4.3.4 FeRAM Communication

The FeRAM is accessed and controlled by the swapping controller. The *HOLD* input pin of the FeRAM is only set if the FeRAM is powered on. The *HOLD* pin suspends the FeRAM, if it is set to low. If the FeRAM is not powered on, this pin is shorted to ground internally.

The swapping controller communicates via the SPI protocol with the FeRAM. It ensures, that all timing requirements are satisfied and that the FeRAM is initialized properly. To backup the FPGA-internal registers, the first word of the FeRAM is addressed, and all data is transferred sequentially. After all data is written, an zero byte is written and the FeRAM

is switched off. The final zero byte ensures, that all data is correctly written to the FeRAM prior to shutdown.

To restore the data, a *FASTREAD* command is issued to the FeRAM. In this mode the memory transfers data sequentially, while the clock signal is active. After transmitting the first address, an zero byte must be written to the FeRAM. This is required, since the FeRAM implements an interface that is compatible to SPI FLASH memories. Then all data can be received and restored to the application modules. When all information are received, the swapping controller suspends the clock signal and switches the FeRAM off.

4.3.5 Design Considerations

FPGA

The Microsemi AGL1000 was chosen, since it has integrated flash memory and does not need to be configured by an external controller. This allows the FPGA to power up in less than $1\,\mu s$. Also the AGL1000 supports core voltages of 1.2 V and 1.5 V, which makes it predestined for low-power environments. At this moment, the Microsemi IGLOO FPGAs are the only available flash-based FPGAs on the market. Competing non-volatile FPGAs like the Intel Altera MAX10 series have the drawback, that they need to copy their configuration from an internal non-volatile memory. This comes along with a higher initial configuration time of at least $300\,\mu s$ [32].

Differential Backup

It was considered, to only backup data that has been altered since the last backup cycle. This could improve the duration and thereby the power efficiency of the backup process. This idea was abolished, since the FeRAM can only be written in whole bytes and for a non-continuous write the address must be retransmitted. Therefore, the write process must be restarted, which has an overhead of 4 bytes to be transmitted. So a differential backup is more efficient than a full backup, only if more than four continues bytes can be left unaltered in the memory. In comparison to FLASH memory, there is no need to erase the data first and overwriting an unaltered segment does not cause any unproportional power consumption. Also the implementation of a differential backup needs additional logic inside the application modules. A dirty flag must be implemented, to track changes of the data. It is not an option, that the swapping controller compares the local data with the data residing in the FeRAM, since a comparison takes the same time and energy as the backup itself.

4.4 Demo Application

To validate the correct functionality of the swapping controller, a demo application is implemented. A simple FIR filter is used to generate data that can be backed up. It stores a finite amount of 32-bit integers in an array and calculates their average after each appended entry. If the array is full, the oldest entry is replaced. The entries are provided by the MCU which transmits them via the control interface, described in Section 4.5. With each new entry, the calculated average is sent back to the MCU.

The size of the FIR filter can be configured. All its data can be completely backed up. And it operates with half the clock rate as the swapping controller.

The demo application is required for the evaluation, but the explanation of its basic functionality is also required for a better understanding of the following control interface.

Command	Bitmask	Description
SHUTDOWN	0xF1	Triggers the swapping controller to start the backup process
RESET	0xF2	Resets all application modules to default values
FIR	0xF3	Provides data for the FIR filter
NOP	0x00	Does noting

Table 4.3.: SPI commands understood by the FPGA

4.5 Control Interface

To allow the FPGA to be controlled, a SPI slave module is implemented. It awaits commands from the MCU and forwards them to the corresponding module. This way, the *SHUTDOWN_TRIGGER* can be set and the FIR filter is filled with data. An overview of the accepted command can be found in Table 4.3.

If *SHUTDOWN* command is received, the FPGA is backed up, as described in Section 4.3.3. After the shutdown, the MCU is signaled by the FPGA through an additional wire, that the power can now be cut. The additional wire is needed, since the FPGA cannot transfer data through the SPI bus on its own initiative.

The *RESET* command is needed, since the swapping handler always restores all data from the FeRAM at startup. To reset the application modules to default state, a manual reset has to be performed. This reset allows the evaluation process, described in Chapter 5, to start each step with a clean data set. The reset signal should be connected to all application modules and must not be connected to the SPI slave module itself. The generated reset signal is cleared with the next received datum on the SPI bus, which can be the *NOP* command or the *SHUTDOWN* command. If the *SHUTDOWN* command is used, the FeRAM is filled with the default data of the modules.

To fill the FIR filter, which is required for the evaluation, the *FIR* command is used. It is the only command that expects further parameters. The first byte after the command is used to declare the number of integers, that are transmitted. This is used, to allow the filter to be initially filled at a faster rate, without the need of retransmitting the command for every integer. The following bytes are aggregated to 32-bit integers and then forwarded to the FIR filter module. This command cannot be aborted prematurely. It only terminates, if the declared number of integers is received. While the integers are received from the MCU, the module expects the calculated average from the FIR filter. With each received integer, the previous average is transmitted to the MCU.

This SPI slave module works with the FPGA-internal clock and can handle any SPI clock, as long as the external clock has a frequency lower than half of the internal clock.

4.6 MCU Implementation

The MCU is used to control the FIR filter and the power supply of the FPGA. To communicate with the FPGA, the commands listed in Table 4.3 are sent via an SPI bus. It also calculates the data for the FIR filter and compares its results with the data committed by the FPGA. The MCU can be controlled via an UART interface. The available commands are listed in Table 4.4. If the FIR filter is started, the MCU sends an batch of data to the FPGA to have a initial setup of the internal registers. Then it sends a new integer to the FPGA every second and compares the received calculation result with the expected result. The

Command	Description
powerUp	Switch the power for the FPGA to state ON
powerDown	Trigger backup process and switch off the FPGA on finish
reset	Send RESET signal to FPGA
nop	Send NOP signal to FPGA
startFir	Start FIR filter and continue to send data to the FPGA
stopFir	Stop FIR calculations

Table 4.4.: UART commands understood by the MCU

state and the sent and received integers are also send via UART to the host PC. This allows the user to monitor the operation of the FPGA.

To power cycle the FPGA and to test the swapping controller, the commands *powerDown* and *powerUp* can be used. The *powerDown* command automatically stops the FIR filter. To continue the FIR filter after the next start of the FPGA, the *startFir* command must be resubmitted.

5 Evaluation

5.1 Functional Verification

To verify the correct operation of the swapping controller and the FeRAM, the development architecture described in Chapter 4 is used. A counter as evaluation module is implemented, which displays its current value on a led bar graph, residing on the development board. Via the control interface, a power cycle of the FPGA is then issued. To ensure that there is no phantom data left in the internal SRAM, the development board is completely cut from its power source in further iterations. The final iteration includes an switched off time of 24 hours. After each iteration, the restored state is compared with the pre-backup state.

Additionally a FIR filter, which calculates the average of 32 entries, is used. Each entry has a size of 32 bit. The FIR filter is implemented on the FPGA as well as on the MCU. This way, the MCU can verify, that the results received from the FPGA are correct. For the communication between the FPGA and the MCU, a SPI bus with a rate of 28800 baud is used.

For the following power measurement, the filter runs automatically. After 100 submitted entries, the FPGA is shut down for 20 seconds. After the restore, 80 further entries are submitted, before the test cycle ends. A LED on the HaLOEWEn board is active, while the received results are correct.

The correct operation of the backup and restore process could be verified.

5.2 Consumption Measuring

5.2.1 Hardware Architecture

To increase the precision of the evaluation the development boards must be replaced by a platform with less interfering components. Therefore the whole codebase is ported to the HaLOEWEn v3 board. Besides the Microsemi AGL1000 FPGA[6] and the Texas Instruments CC2531 MCU and their voltage regulators there are no further active components integrated. The power of the FPGA-core and of three out of its four I/O-banks can be completely cut. But since the last I/O-bank is hardwired to the MCU, which requires a fixed voltage of 2.5 V, it cannot be switched off. Whereas the other I/O-banks are configurable to be driven with either 3.3 V or 2.5 V, by placing a jumper. This jumper can be replaced with an n-channel MOSFET, to control the power supply of the I/O-banks.

The FeRAM is directly attached to the FPGA. Its power supply is also provided by an FPGA pin. The FPGA-core and the I/O-banks can be configured and controlled with jumpers. By replacing these jumpers with multiple n-channel MOSFETs, the power supply of the FPGA can be controlled by the MCU. The switched power must be provided by a separate DC power supply, because of the high current that is needed to refill the stabilizing capacitors of the FPGA. Otherwise, the supply voltage of the development board would drop below the operating voltage of the MCU. This could be mitigated by placing the transistor after the voltage stabilization unit, or by installing more powerful voltage regulators. Both concepts

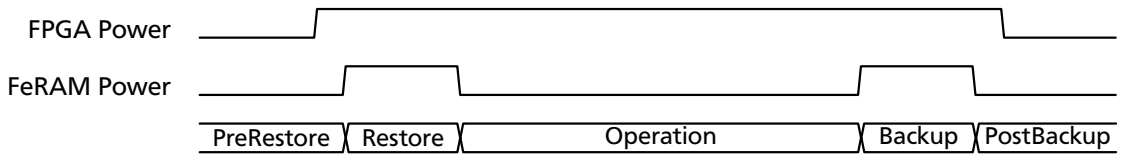

Figure 5.1.: Backup Process cycle

could not be implemented in this work, since they require a redesign of the printed circuit board (PCB) of the HaLOEWEn board.

To generate some data to be backed up and restored the previously explained FIR filter is used. The MCU monitors the calculated results to guarantee that the backup/restore-process works flawlessly. It also controls the whole test cycle by initiating the restore and backup process of the FPGA.

But even the HaLOEWEn v3 board has some drawbacks concerning the evaluation. The UART connection to the computer is missing, so it is not possible to get extensive information about the current state. Admittedly, it provides an USB interface, but its usage required extensive adaptations of the sourcecode. To monitor the correct execution of the FIR filter and of the backup and restore process, two LEDs are used. These LEDs are connected to the MCU and therefore do not affect the measured power drain of the FPGA. Also the SPI bus controller on the MCU, used with the development boards, cannot be used, due to a bug in the HaLOEWEn board. Therefore, the program running on the MCU has to be altered to use the second available controller. This results in huge changes in the source code, because all SPI related status registers including the interrupt routines had to be adapted.

FPGA Logic Cell Requirements

The listing in Appendix C shows the compile report for the swapping controller. The controller is configured with a bus width of 8 bit and a backup size of 32 kbit. Additionally, the SPI bus controller and a counter as demo application module are synthesized. These modules combined, require 4.5 percent of the core logic gates and 7.81 percent of the I/O-lines. Since the FPGA has only one phase-locked loop (PLL) module, this sample uses all available PLL modules. But two out of three available clock signals can be configured for application modules.

With this simple setup, the swapping controller can be driven at 40 MHz. This is the maximum frequency supported by the FeRAM.

5.2.2 Measurement

To get the power consumption of the single components, a HAMEG HMO3224 [33] digital oscilloscope is used. It has a DC gain accuracy of two percent. In combination with an 100 Ω shunt resistor for the I/O-banks and a 10 Ω shunt resistor for the core, the input current and input voltage is recorded. These shunt resistors are installed between the DC/DC converters and the switching MOSFETs. Since the export of the trace only allows single precision floats, the recorded current is multiplied with the recorded voltage by the oscilloscope to reduce the loss of information due to rounding. The precision of the two probes adds up to a four percent error. The resistance of the shunts has been verified to be precise with an error of 0.1 percent. The data has 6 significant digits, which is still more, than the guaranteed precision of the oscilloscope. The trace is then exported as a CSV-file to be further analyzed with GNU Octave.

The test cycle is divided in multiple phases illustrated in Figure 5.1:

- First the *pre-restore* phase which starts 4 ms before powering up the FeRAM and ends just before powering up the external memory.
- Afterwards, the *restore* phase starts, in which the copy process from the external memory to the internal registers takes place. It ends with powering down the FeRAM.
- Then the *operation* phase begins where the external memory is shut down and the FPGA does its normal work. This phase is covered later, since its duration is independent from the amount of data, that needs to be backed up.
- In the *backup* phase all relevant data from the internal registers is copied back to the FeRAM.
- The *post-backup* phase starts just after powering down the FeRAM and lasts for 4 ms.

The buffer capacitors for the FPGA core need 20 ms to discharge below 1 mV. This is far less that the 20 s waiting time, between shutdown and power up, used by the test cycle.

Every single phase of the process is measured independently and multiple times. This allows the detection of observation errors. And it shows, which phase is associated with a high energy consumption.

5.2.3 Results

FPGA-core and I/O-bank power consumption are measured independently which allows to decide which switches are really needed. Backup and restore were performed with 450 bytes of data and with an SPI bus frequency of 25 MHz. This is the maximum frequency, the FeRAM supports, if it is powered with 2.5 V instead of 3.3 V. The energy consumption with different amounts of data is calculated. Therefore, the power consumption during the backup phase is lineary scaled to the time, which is required for a different amount of data.

FPGA Core Energy Consumption

Table 5.1 show the energy consumed solely by the FPGA-core. First it must be decided, if the recharging of the buffer capacitors could be compensated. The first row, titled with “switched”, shows the energy consumed, if the MOSFET is used to cut the power. The second row, titled with “non-switched”, shows the energy consumed, if the core is not switched off. This data is collected, to determine the impact of the buffer capacitors on the energy consumption. Additionally, Figure 5.2 shows, that a large amount of Energy is consumed to power up the FPGA. In comparison to the second measurement series, the start of the FPGA consumes additional 39.426 μJ. (12.469 μJ + 33.140 μJ - 6.183 μJ)

The transition into the FlashFreeze state was not separately evaluated. The comparison is made with the datasheet values. With the energy, which is required to charge the buffer capacitors, the FPGA can stay nearly one second in the FlashFreeze state. Therefore, independent from the amount of data to be backed up, it is not efficient to switch off the FPGA for less than one second. The FPGA core needs 9.4 mW, while it is performing the restore. With an clock of 25 MHz, the transfer of one byte data takes 320 ns. Therefore, for each byte of data, an additional energy of 6 nJ is needed, only by the core, for backup and restore combined. The data independent energy consumption totals up to 48.29 μJ. (12.469 μJ + 33.140 μJ + 5.381 μJ - 450 * 0.006 μJ) This can be split to initialization time of the FeRAM before restore and backup, and to the start of the FPGA. After the FPGA is switched off, there is no further energy consumed by the FPGA.

	Pre-Restore	Restore	Backup	Post-Backup
switched	12.469 µJ	33.140 µJ	5.381 µJ	0.061 µJ
non-switched	37.537 µJ	6.183 µJ	5.381 µJ	38.310 µJ

Table 5.1.: FPGA-Core Energy consumption for 450 byte of data

	Pre-Restore	Restore	Backup	Post-Backup
switched	5.758 µJ	37.230 µJ	0.235 µJ	0.097 µJ
non-switched	0.002 µJ	1.267 µJ	0.235 µJ	2.144 µJ

Table 5.2.: I/O-Bank Energy consumption for 450 byte of data

I/O-Bank Energy Consumption

For the I/O-banks, Table 5.2 shows, that the non-switched variant consumes significant less energy than the switched variant. The I/O-banks are switched off by the FPGA core, when it is powered down. Therefore, there is no need to switch off the I/O-banks of the FPGA, since they consume no energy, while the core is switched off. The I/O-banks require 420.6 µW while performing a backup. A single byte needs less than 0.3 nJ to be backed up and restored. The data independent overhead is 3.51 µJ. (1.267 µJ + 0.235 µJ + 2.144 µJ - 450 * 0.0003 µJ)

Total Swapping Energy Consumption

The data idependent overhead of core and I/O-banks combined sums up to 51.80 µJ, which is slightly less, than the FlashFreeze mode consumes per second. Figure 5.3 shows, how long the FPGA must be idle, that the swapping process is more efficent than the Flash-Freeze. This time depends on the amount of data to be backed up. The figure does not consider the energy consumption of the MOSFET.

MOSFET Penalty

At last, the power dissipation of the utilized MOSFET is considered. Therefore, the power drain during the *operation* phase is measured. This shows, that the core consumes 10.11 mW with an integrated MOSFET and 10.007 mW without an integrated MOSFET. The difference of 103 µW is twice the consumption, the FPGA consumes in FlashFreeze state. Therefore, the FPGA must be twice as long switched off, than it is powered, to compensate the energy loss on the transistor.

Following formula can be used to determine, how long the FPGA has to be idle, so that the power down pays off:

$$\begin{aligned} b &:= \text{data to be backed up (byte)} \\ f &:= \text{memory frequency (hertz)} \\ t &:= \text{active time (seconds)} \\ s &:= \text{minimal sleep time (seconds)} \end{aligned} \tag{5.1}$$

$$s = \underbrace{\left(\frac{1}{f} \times 16 \times b + t\right) \times \frac{103}{52.8}}_{1} + \underbrace{\frac{51.8}{52.8}}_{2} + \underbrace{\frac{0.0063}{52.8} \times b}_{3}$$

The first addend regards the power dissipation of the MOSFET. This is affected by the run-time of the application modules as well as by the amount of data to be backed up. The

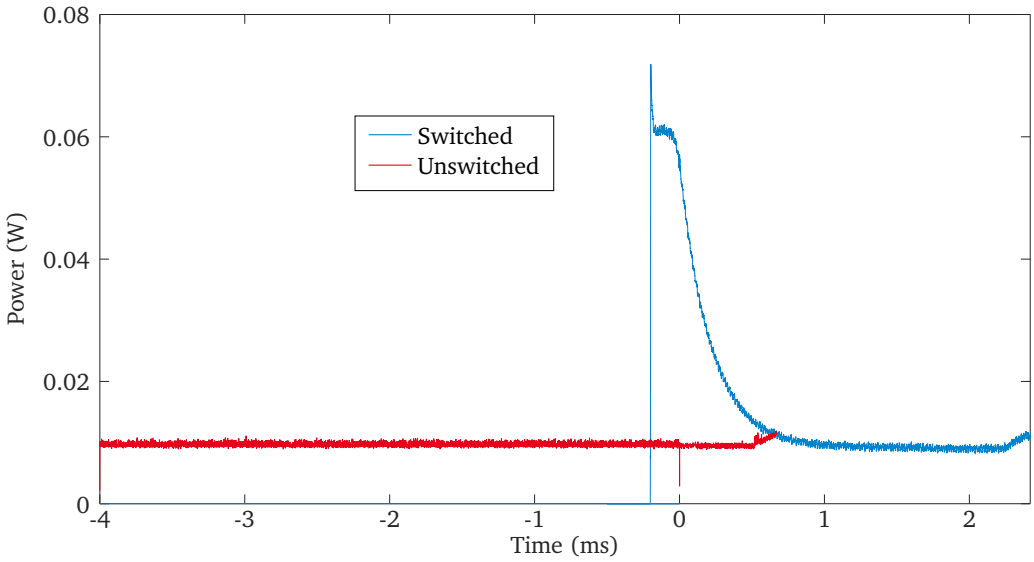

Figure 5.2.: Power usage of the core during restore process
The FeRAM is powered on at 0 ms

second addend compensates the data independent power consumption, arisen from the initialization of the FPGA and FeRAM. The third addend is used to calculate the compensation time, which is needed to backup and restore the data.

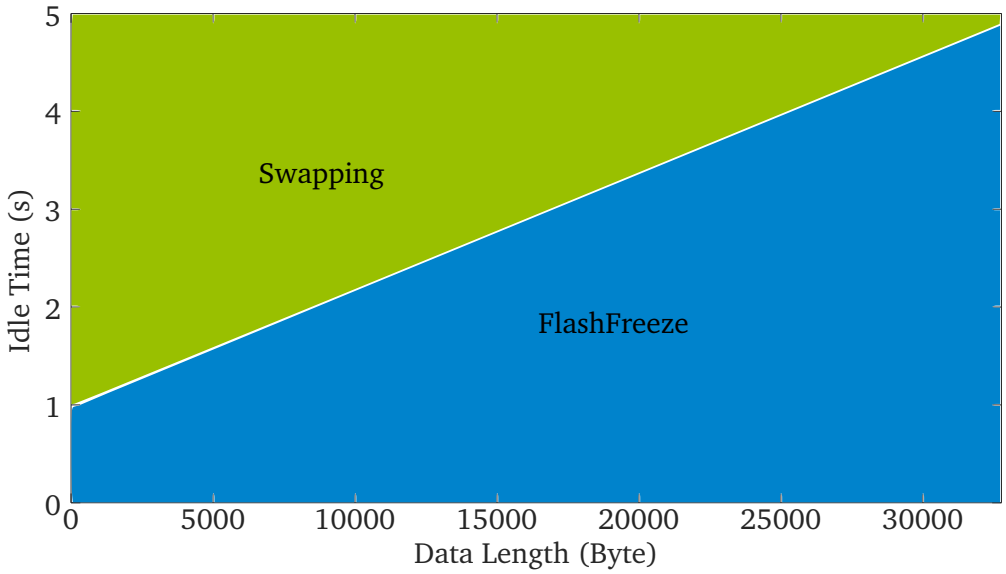

Figure 5.3.: Idle time, depending on the amount of data to be backed up, for which swapping is more efficient than a transition into FlashFreeze

6 Summary and Future Work

In this thesis, a prototype based on Microsemi and Texas Instruments development boards was developed to reduce the standby power drain of an FPGA by switching it off. It uses an external FeRAM as backup memory to preserve to information located in the FPGA-internal SRAM. The FeRAM technology has been evaluated to be superior in terms of energy consumption to current alternatives like FLASH or MRAM.

To preserve the relevant infomation, a swapping controller was implemented on the FPGA which undertakes the communication with the FeRAM and provides an interface for the application modules. Additionally a controll protocol between the FPGA and an MCU based on the SPI bus was designed. It is used to initiate the shutdown and backup and to control a FIR filter which is used to verify the correct operation of the swapping controller.

For the evaluation the prototype was ported to the HaLOEWEn v3 plattform. Because of its high energy efficiency, it allowed precise tests to determine the energy usage of the backup and restore process. With this plattform it was possible to reduce the power drain of the FPGA to zero watt, while it is not used. The time overhead of 10 ms per power cycle as well as the additionally energy usage of 52 μJ must be considered, to determine if the implementation into a WSN application is reasonable. Additionally the MOSFET switch, used to controll the power of the FPGA, has a noticable resistance, which results in an additional power drain of 103 μW. With the used Microsemi AGL1000, the data swapping is rentable if the idle time exceeds twice the active time.

6.1 Future Improvements

There are two main issues which require further improvements. At first the PCB design of the HaLOEWEn plattform should be altered. It needs the direct integration of the MOSFET. Then even more efficient transistors like the Vishay SiA427DJ[34] can be used. Since it is only available in the PowerPAK®package, which is unsiutable for a development prototype, this MOSFET was not evaluated. But its specifications promise a higher power efficiency than the used Vishay Si4465ADY[29]. Additionally the transistor could be positioned logically behind the buffer capacitors. This removes the large initial power drain during powering up the FPGA.

Also an incremental backup scheme could be implemented and evaluated. This could shorten the backup time and may allow a changeover to PCM. But this approach only pays off, if the application data mostly stays the same during multiple power cycles. For the FeRAM an improved alignment of the application data inside the memory can make an incremental backup scheme more efficient, with only a few bytes of “static” data. But in the end, it is highly dependent on the type of data to be backed up.

As additional minor issue, the swapping controller expects the data, to be backed up, in the same clock cycle, as it updates the address. This requires the application modules to implement prefetching logic, if they have data stored in the block-RAM. This could be improved, by implementing a halt bit, which allows the modules to suspend the backup process, until the data is fetched from the block-RAM.

A Verilog Module Implementation

```
1  module demo_application(CLK, RESET, DATA_ADDR, DATA_IN, DATA_OUT, SWP_STATE);
2
3  // Parameter to control swapping bus
4  parameter BUS_WIDTH = 4;
5  parameter ADDR_WIDTH = 16;
6  parameter ADDR_OFFSET = 1;
7
8  // Constants that define amount of data to be backed up
9  localparam BITCOUNT = 1032;
10 localparam MAX_ADDR = div_up(BITCOUNT, BUS_WIDTH) + ADDR_OFFSET-1;
11
12 input    wire                      CLK;
13 input    wire                      RESET;
14 input    wire[1:0]                 SWP_STATE;
15 input    wire[ADDR_WIDTH-1:0]      DATA_ADDR;
16 input    wire[BUS_WIDTH-1:0]       DATA_IN;     //  Restore this
17 output   wire[BUS_WIDTH-1:0]       DATA_OUT;    //  Backup this
18
19 // Constants for backup state
20 // Can be omitted
21 localparam SWP_IDLE    = 2'b00;
22 localparam SWP_BACKUP  = 2'b10;
23 localparam SWP_RESTORE = 2'b01;
24
25 // Calculate local address
26 wire[ADDR_WIDTH-1:0] address;
27 assign address  = DATA_ADDR - ADDR_OFFSET;
28
29 // Combination of all registers to be backed up
30 wire [BITCOUNT-1:0] datasum;
31
32 // Select data
33 wire [9:0] datashift;
34 assign datashift = datasum >>> (address  * BUS_WIDTH);
35 // Set output if address is in {1,2,3}
36 assign DATA_OUT = ((DATA_ADDR >= ADDR_OFFSET ) &&
37                    (DATA_ADDR <= MAX_ADDR) &&
38                    (SWP_STATE == SWP_BACKUP))
39 ? datashift[BUS_WIDTH-1:0] : {BUS_WIDTH{1'b0}};
40
41
42 // Register containing data to be preserved
43 reg [7:0] localreg;
44
45     always @(posedge CLK)
46     begin
47         if (RESET)
48         begin
49           // Reset registers
50         end
51         else if (SWP_STATE == SWP_IDLE)
52         begin
53             // Do your work
54         end
55         else if (SWP_STATE == SWP_RESTORE && DATA_ADDR >= ADDR_OFFSET &&
56                  DATA_ADDR <= MAX_ADDR)
```

```
57          begin
58            begin case (address)
59                // Restore data
60                0: localreg[3:0]          <= DATA_IN;
61                1: localreg[7:4]          <= DATA_IN;
62            endcase end
63          end
64      end
65
66    /********* FUNCTIONS *********/
67    //divide and round up
68    function integer div_up;
69      input [31:0] divident;
70      input [31:0] divisor;
71      integer product;
72      begin
73          div_up = 1;
74          for (product=divisor; product < divident;
75                                    product = product + divisor)
76              div_up = div_up+1;
77      end
78    endfunction
79  endmodule
```

B Verilog Toplevel Implementation

```
1  module toplevel
2  (
3    input wire CLK,
4    input wire RESET,
5    output wire SHUTDOWN_COMPLETE,
6    output wire FRAM_CLK,
7    output wire FRAM_CS,
8    input wire FRAM_SO,
9    output wire FRAM_SI,
10   output wire FRAM_WP,
11   output wire FRAM_HOLD,
12   output wire FRAM_POWER
13 );
14
15 wire [1:0] swp_state;
16 wire [7:0] swp_address;
17 wire [3:0] bus_backup;
18 wire [3:0] bus_backup_arr [6:0];
19 wire [3:0] bus_restore;
20
21 wire SHUTDOWN_TRIGGER;
22 wire RST;
23 assign RST = (!RESET);
24
25
26 assign bus_backup_arr[0] = 0;
27 assign bus_backup_arr[2] = 15;
28
29 assign bus_backup =
30 (swp_address < 3) ? bus_backup_arr[2]  :
31 (swp_address < 261) ? bus_backup_arr[3] : bus_backup_arr[0];
32
33
34 swap_handler #(
35     .BUS_WIDTH (4),
36     .ADDR_WIDTH (11),
37     .DATA_LEN (900)
38 )
39 SWAP_HANDLER_i (
40   .CLK (CLK),
41   .RESET (RST),
42   .BACKUP (SHUTDOWN_TRIGGER),
43   .BACKUP_FINISHED (SHUTDOWN_COMPLETE),
44   .DATA_IN (bus_backup),
45   .DATA_OUT (bus_restore),
46   .DATA_ADDR (swp_address),
47   .SWP_STATE (swp_state),
48   .FRAM_SO (FRAM_SO),
49   .FRAM_CLK (FRAM_CLK),
50   .FRAM_CS (FRAM_CS),
51   .FRAM_SI (FRAM_SI),
52   .FRAM_WP (FRAM_WP),
53   .FRAM_HOLD (FRAM_HOLD),
54   .FRAM_POWER (FRAM_POWER)
55 );
56
```

```
57
58 demo_application #(
59     .BUS_WIDTH (4),
60     .ADDR_WIDTH (8),
61     .ADDR_OFFSET (3)
62 )
63 DEMO_APPLICATION_i (
64   .CLK (CLK),
65   .RESET (RST),
66   .DATA_IN (bus_restore),
67   .DATA_OUT (bus_backup_arr[3]),
68   .DATA_ADDR (swp_address),
69   .SWP_STATE (swp_state)
70 );
71
72 endmodule
```

C Libero Compile Report

```
Compile report:
===============

    CORE                          Used:    1107  Total:  24576   (4.50%)
    IO (W/ clocks)                Used:      14  Total:    177   (7.91%)
    Differential IO               Used:       0  Total:     44   (0.00%)
    GLOBAL (Chip+Quadrant)        Used:       3  Total:     18   (16.67%)
    PLL                           Used:       1  Total:      1   (100.00%)
    RAM/FIFO                      Used:       0  Total:     32   (0.00%)
    Low Static ICC                Used:       0  Total:      1   (0.00%)
    FlashROM                      Used:       0  Total:      1   (0.00%)
    User JTAG                     Used:       0  Total:      1   (0.00%)
```

Bibliography

[1] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless sensor network survey", *Computer Networks*, vol. 52, no. 12, pp. 2292–2330, Apr. 2008, ISSN: 1389-1286. DOI: 10.1016/j.comnet.2008.04.002.

[2] A. Engel, "A heterogeneous system architecture for low-power wireless sensor nodes in compute-intensive distributed applications", PhD thesis, Technische Universität Darmstadt, 2015. [Online]. Available: http://tuprints.ulb.tu-darmstadt.de/5778/.

[3] F. Mafie. (Mar. 2014). Comparing flash- and sram-based fpgas, [Online]. Available: http://www.electronicproducts.com/Digital_ICs/Standard_and_Programmable_Logic/Comparing_flash-_and_SRAM-based_FPGAs.aspx (visited on 04/23/2017).

[4] *ARM Cortex-M1-Enabled IGLOO*, Rev. 1, Microsemi, Jun. 2011. [Online]. Available: http://www.digchip.com/datasheets/download_datasheet.php?id=4246116&part-number=M1AGL1000-DEV-KIT.

[5] *A USB-Enabled System-On-Chip Solution for 2.4-GHz IEEE 802.15.4 and ZigBee Applications*, CC2531, Texas Instruments, 2010. [Online]. Available: http://www.ti.com/lit/ds/symlink/cc2531.pdf (visited on 03/13/2018).

[6] *IGLOO Low Power Flash FPGAs*, DS0095, Rev. 27, Microsemi, May 2016. [Online]. Available: https://www.microsemi.com/document-portal/doc_download/130694-ds0095-igloo-low-power-flash-fpgas-datasheet (visited on 03/21/2018).

[7] G. Anastasi, M. Conti, M. di Francesco, and A. Passarella, "Energy conservation in wireless sensor networks: A survey", *Ad Hoc Networks*, vol. 7, no. 3, pp. 537–568, Jun. 2008, ISSN: 1570-8705. DOI: 10.1016/j.adhoc.2008.06.003.

[8] V. Raghunathan, C. Schurgers, S. Park, and M. B. Srivastava, "Energy-aware wireless microsensor networks", *IEEE Signal Processing Magazine*, vol. 19, no. 2, pp. 40–50, Mar. 2002, ISSN: 1053-5888. DOI: 10.1109/79.985679.

[9] J. Valverde, A. Otero, M. Lopez, J. Portilla, E. de la Torre, and T. Riesgo, "Using sram based fpgas for power-aware high performance wireless sensor networks", *Sensors*, vol. 12, no. 3, pp. 2667–2692, Feb. 2012. DOI: 10.3390/s120302667.

[10] H. Li, Y. Liu, Q. Zhao, Y. Gu, X. Sheng, G. Sun, C. Zhang, M.-F. Chang, R. Luo, and H. Yang, "An energy efficient backup scheme with low inrush current for nonvolatile sram in energy harvesting sensor nodes", in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, ser. DATE '15, Grenoble, France: EDA Consortium, 2015, pp. 7–12, ISBN: 978-3-9815370-4-8. [Online]. Available: http://dl.acm.org/citation.cfm?id=2755753.2755756.

[11] R. Novotný, J. Kadlec, and R. Kuchta, "Nand flash memory organization and operations", *Journal of Information Technology & Software Engineering*, vol. 5, no. 1, 2015. DOI: 10.4172/2165-7866.1000139.

[12] *Micron® 3d nand flash memory*, Micron Technology Inc, 2016. [Online]. Available: https://www.micron.com/~/media/documents/products/product-flyer/3d_nand_flyer.pdf (visited on 04/23/2017).

[13] Y. Koh, "Nand flash scaling beyond 20nm", in *2009 IEEE International Memory Workshop*, Apr. 2009, pp. 1–3. DOI: 10.1109/IMW.2009.5090600.

[14] R. Micheloni, A. Marelli, and S. Commodaro, "Nand overview: From memory to systems", in *Inside NAND Flash Memories*. Dordrecht: Springer Netherlands, 2010, pp. 19–53, ISBN: 978-90-481-9431-5. DOI: 10.1007/978-90-481-9431-5_2.

[15] G. Mathur, P. Desnoyers, P. Chukiu, D. Ganesan, and P. Shenoy, "Ultra-low power data storage for sensor networks", *ACM Trans. Sen. Netw.*, vol. 5, no. 4, 33:1–33:34, Nov. 2009, ISSN: 1550-4859. DOI: 10.1145/1614379.1614385.

[16] *1-GBIT (128M x 8 BITS) CMOS NAND E^2 PROM*, TC58DVG02A1FT00, Toshiba, Oct. 2003. [Online]. Available: http://www.alldatasheet.com/datasheet-pdf/pdf/538804/TOSHIBA/TC58DVG02A1FT00.html (visited on 03/21/2018).

[17] R. Bez, E. Camerlenghi, A. Modelli, and A. VIsconti, "Introduction to flash memory", *Proceedings of the IEEE*, vol. 91, no. 4, pp. 489–502, Apr. 2003, ISSN: 0018-9219. DOI: 10.1109/JPROC.2003.811702.

[18] D. Ielmini, "Threshold switching mechanism by high-field energy gain in the hopping transport of chalcogenide glasses", *Phys. Rev. B*, vol. 78, p. 035 308, 3 Jul. 2008. DOI: 10.1103/PhysRevB.78.035308.

[19] H. S. P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson, "Phase change memory", *Proceedings of the IEEE*, vol. 98, no. 12, pp. 2201–2227, Dec. 2010, ISSN: 0018-9219. DOI: 10.1109/JPROC.2010.2070050.

[20] B. D. Yang, J. E. Lee, J. S. Kim, J. Cho, S. Y. Lee, and B. G. Yu, "A low power phase-change random access memory using a data-comparison write scheme", in *2007 IEEE International Symposium on Circuits and Systems*, May 2007, pp. 3014–3017. DOI: 10.1109/ISCAS.2007.377981.

[21] S. Tehrani, J. M. Slaughter, E. Chen, M. Durlam, J. Shi, and M. DeHerren, "Progress and outlook for mram technology", *IEEE Transactions on Magnetics*, vol. 35, no. 5, pp. 2814–2819, Sep. 1999, ISSN: 0018-9464. DOI: 10.1109/20.800991.

[22] *MR256D08B Dual Supply 32k x 8 MRAM*, MR256D08B, Rev. 3.2, Everspin Technologies, Jun. 2015. [Online]. Available: https://www.everspin.com/file/207/download (visited on 06/10/2017).

[23] C. Augustine, N. N. Mojumder, X. Fong, S. H. Choday, S. P. Park, and K. Roy, "Spin-transfer torque mrams for low power memories: Perspective and prospective", *IEEE Sensors Journal*, vol. 12, no. 4, pp. 756–766, Apr. 2012, ISSN: 1530-437X. DOI: 10.1109/JSEN.2011.2124453.

[24] J. F. Scott and C. A. P. de Araujo, "Ferroelectric memories", *Science*, vol. 246, pp. 1400–1405, 4936 Dec. 1989. DOI: 10.1126/science.246.4936.1400.

[25] *256-Kbit (32K × 8) Serial (SPI) F-RAM*, FM25V02A, Cypress Semiconductor Corporation, Sep. 2016. [Online]. Available: http://www.cypress.com/file/139666/download (visited on 03/15/2018).

[26] *4-Mbit (256K × 16) F-RAM Memory*, FM22LD16, Cypress Semiconductor Corporation, Jan. 2016. [Online]. Available: http://www.cypress.com/file/136506/download (visited on 03/15/2018).

[27] ShroudResearch, "Intel optane storage performance and implications on testing methodology", Oct. 2017, [Online]. Available: https://static1.squarespace.com/static/57fdb580ff7c50274b1387ef/t/59f24b308e7b0f2fb05e73f3/1509051187047/IntelOptaneSSD900p_Performance_Testing_Methodology_v10.pdf (visited on 03/15/2018).

[28] *N-Channel 8 V (D-S) MOSFET*, SiA436DJ, Rev. A, Vishay, Nov. 2011. [Online]. Available: http://www.farnell.com/datasheets/2050059.pdf.

[29] *P-Channel 1.8 V (G-S) MOSFET*, Si4465ADY, Rev. B, Vishay, Mar. 2009. [Online]. Available: http://www.farnell.com/datasheets/2049699.pdf.

[30] *Bsl802sn*, Rev. 2.2, Infineon Technologies AG, Mar. 2010. [Online]. Available: http://www.farnell.com/datasheets/1648222.pdf (visited on 05/31/2017).

[31] *TQ RELAYS*, TQ2-L-3V, Panasonic Corporation, Jul. 2014. [Online]. Available: http://www.farnell.com/datasheets/2243479.pdf (visited on 05/24/2017).

[32] *Intel® MAX® 10 FPGA Device Datasheet*, Intel, Dec. 2017. [Online]. Available: https://www.altera.com/en_US/pdfs/literature/hb/max-10/m10_datasheet.pdf (visited on 03/31/2018).

[33] *350Mhz 4 Channel Digital Oscilloscope*, HMO3524, Hameg Instruments GmbH. [Online]. Available: https://mcs-testequipment.com/resources/Datasheets_Downloads/Hameg/HMO3522%203524%20Datasheet.pdf (visited on 03/06/2018).

[34] *P-Channel 8 V (D-S) MOSFET*, SiA427DJ, Rev. C, Vishay, May 2012. [Online]. Available: http://www.farnell.com/datasheets/1698160.pdf.

Abbreviations

ASIC	Application Specific Integrated Circuit
AWS	adaptive writeback scheme
BJT	bipolar junction transistor
CMOS	Complementary metal–oxide–semiconductor
CSV	comma separated values
DPM	Dynamic Power Management
EPROM	Erasable Programmable Read-Only Memory
FeRAM	Ferroelectric Random-Access Memory
FIR	finite impulse response
FPGA	Field Programmable Gate Array
HaLOEWEn	Hardware-Accelerated Low Energy Wireless Embedded Sensor Node
I/O	Input/Output
IoT	Internet of Things
LED	light-emitting diode
MCU	Microcontroller Unit
MLC	multi-level cell
MOSFET	Metal-Oxide-Semiconductor Field-Effect Transistor
MRAM	Magnetoresistive Random-Access Memory
MTJ	Magnetic Tunnel Junction
NVM	non-volatile memory
PCB	printed circuit board
PCM	phase change memory
PLL	phase-locked loop
$R_{DS(on)}$	drain-source resistance
RF-SoC	Radio Frequency System-on-Chip
ROM	Read-Only Memory
SBDP	statistics based dead blocks prediction
SLC	single-level cell

SPI	Serial Peripheral Interface
SRAM	Static Random-Access Memory
SSD	Solid State Drive
TLC	triple-level cell
UART	universal asynchronous receiver-transmitter
WSN	Wireless Sensor Network

List of Tables

List of Figures