A Prototype Implementation of the TinyOS Hardware Abstraction Architecture for Ferroelectric RAM

Sebastian A. Bachmaier
Material Testing Institute
Universität Stuttgart
Stuttgart, Germany
sebastian.bachmaier@mpa.uni-stuttgart.de

***Abstract*—This report presents a hardware driver for the Ramtron Ferroelectric RAM (FRAM, FeRAM) chips for use in TinyOS according to TinyOS' Hardware Abstraction Architecture. FRAM is a replacement for flash memory, suitable for usage in Wireless Sensor Networks (WSNs) for its properties. The properties of FRAM and flash are shortly depicted and compared. The design of the driver implementation is described, including a chip clustering method to circumvent the capacity limitation. The driver offers the DirectStorage interface and the BlockStorage interface for usage by applications. Comments on the suitability of the provided interfaces, intended for flash memory originally, for FRAM are given.**

***Index Terms*—TinyOS, FRAM, driver, hardware abstraction architecture**

I. INTRODUCTION

In this report, TinyOS, TinyOS drivers and FRAM are introduced shortly. Then, the architecture of the driver implementation is shown, the clustering of several chips is described and finally the suitability of existing TinyOS flash storage abstractions is assessed.

A. TinyOS and its Driver Model

TinyOS is an operating system and library of code components for sensor networks. The further development is done by working groups and by user contributions. Working groups can issue TinyOS Extension Proposals (TEPs), specifying best practices for new code contributions.

TEP2 [1] is one of the central TEPs for TinyOS 2.0. It describes a *hardware abstraction architecture* (HAA). The HAA specifies a three-layered architecture for driver implementations. The three layers comprise the *hardware presentation layer* (HPL), which exposes the hardware's capabilities directly, the *hardware adaptation layer* (HAL) which abstracts the hardware and allows to maintain states in software, and the *hardware interface layer* (HIL) which offers a standardized platform-independent interface for applications, irrespective of the underlying hardware components. Drivers reside in a *chip* directory by convention, with some additional code in a *platform* directory where code is placed which states the platform specifics, like specific hardware pins.

B. Ferroelectric RAM (FRAM)

FRAM is a relatively new memory technology which combines the best from static RAM memory (fast, energy efficient) and flash memory (non-volatile). It is based on a ferroelectric material which retains its state even when currentless. FRAM is a suitable replacement for flash.

TABLE I. shows a simplified comparison between flash and FRAM memory properties. Please note that the values for a specific application have to be taken from the actual datasheet of the actually used chip. Values may vary greatly, especially for the energy consumption per stored bit as this depends not only on the used chip, but also on the calculation model, e.g. the assumptions made with respect to read-write cycle times and the assumed bus speed. Hence, the data is only given to stress some main differences. These are: (1) the durability in terms of write cycles. This is irrelevant for many applications however. (2) The capacity which is in favor of flash memory since being in a later stadium of the development cycle and the smaller manufacturing processes. (3) The energy effort to store a bit. This is an important quantity in WSNs for the power limitations imposed to the system owing to the desired autonomous operation over long time periods.

TABLE I. SIMPLIFIED FLASH–FRAM COMPARISON

Comparison with respect to …	**Type of Non-Volatile Memory**	
	flash	***FRAM***
Available Interfaces	SPI/I2C/Parallel	SPI/I2C/Parallel
Sleep Mode Current	1 µA	1 µA
Data Retention	> 10 a	> 10 a
Write Cycles	~ 10^5	~ 10^{10}
Capacity[a]	≤ 32 Gibit (parallel) ≤ 128 Mibit (SPI)	≤ 4 Mibit (parallel) ≤ 2 Mibit (SPI) ≤ 1 Mibit (I2C)
Energy Consumption[b]	90 nJ/bit	1.1 nJ/bit
Write Speed/byte[c]	~ 10 µs	~ 400 ns

a. Development is making rapid progress. This is a snapshot view only. Capacity varies with physical chip/die size.

b. These values differ greatly with the usage model used for calculation and the actual chip.

c. Depending on bus speed, data unit size and others.

Ramtron, Colorado Springs, CO offers FRAM chips with SPI bus which are meant to replace serial flash memory. The SPI protocol used is similar to the one of flash chips. Pin compatibility is also given. It is therefore easy to replace flash by FRAM. The realization here is for the FM25H20 type.

This work was funded by the 7th Framework Programme of the European Commission.

II. IMPLEMENTATION

The implementation follows that of the STM25P flash chip by Hui [2]. The components should reside in tinyos-2.x/tos/chips/fm25h and platform/<platform-name>/chips/fm25h. However, the contribution resides at tinyos-2.x-contrib/ustutt where it can be retrieved from.

A. Hardware Presentation Layer (HPL)

The HPL offers no *erase* and *pageProgram* commands, but offers a *write* command instead. In FRAM writes are possible without prior erase. The write command operates on data units of down to single bytes. Flush is not implemented since data is always written through. Sleep mode support is available.

B. Hardware Adaptation Layer (HAL)

Two HAL implementations are offered: one simpler HAL for single chip mode and a ClusterHAL component for clustered operation of several chips under a flat, continuous address space. Standard wiring uses the single chip HAL. The single chip HAL is similar to the STM25P implementation.

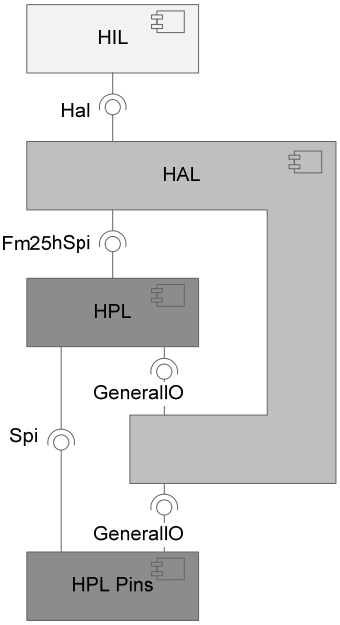

Figure 1. HAA of the clustered operation: the HPL is ignorant of the actually used chip as this is dispatched by the HAL (GeneralIO interface)

C. ClusterHAL

While the FRAM offers some advantages over flash memory it still offers less capacity. This is due to the smaller packing density, which is caused by the larger manufacturing process sizes and the ferroelectric material properties.

We therefore had to bundle several chips to get a memory size comparable to the 1 MiB of the TelosB which were used as reference. The resulting cluster was desired to act like one big memory under a unified address space. This means a dispatcher has to handle accesses to the unified address space and direct them to the corresponding chip. The dispatcher is provided on the HAL layer. This has the advantage of having the HPL unchanged for cluster or single chip operation. Furthermore, HPL can be stateless and "present" just the operations the FRAM offers. However, the layering in this approach is not strictly adhered to as for clustered operation the IO pins are handled by the HAL, transparently to the HPL (see Figure 1). The HPL just accesses the chip select (CS) to activate the large virtual chip (of which the HPL is ignorant of) and the HAL activates the appropriate physical chip determined by the memory address that is accessed. Memory accesses across chips are split into several separate operations. Other approaches are conceivable and can be implemented later. E.g., a strictly layered architecture would access several individual HPLs, but code is replicated then.

D. Hardware Interface Layer (HIL)

For a description of two prototype implementations of HIL refer to chapter III. The implementation here follows the STM25P implementation.

III. SUITABILTY ASSESMENT OF FLASH ABSTRACTIONS

A. BlockStorage Interface (TEP103)

TEP103 [3] standardizes three fundamental storage abstractions found in typical sensor network applications: BlockStorage for program memory, ConfigStorage for little chunks of configuration data and LogStorage for data logging application. TEP103 aims solely at flash memory and incorporates specialties of flash memory. However, the flash functionality is a subset of FRAM functionality, i.e. FRAM has fewer restrictions to consider. It should therefore be possible to realize these storage abstractions for FRAM. For workload restriction, of the three storage abstraction of TEP103, only the BlockStorage was implemented exemplarily.

B. DirectStorage Interface (TEP128)

TEP 128 ([4], draft version) describes an interface for direct access to non-volatile storage. It offers read, write, erase, flush and crc commands. It differs primarily in two points from the abstractions of TEP103: (1) the interface is an application indifferent general purpose interface, and (2) the implementation (in conjunction with the VolumeSettings interface) is platform independent. TEP 129 describes a new set of BlockStorage, ConfigStorage and LogStorage which resides above the platform-independent intermediate DirectStorage interface.

There are some comments to the DirectStorage interface which occurred during implementation of the interface. Due to the missing sector size of FRAM, the sector size can artificially be set to either an arbitrary size (for easier programming a fraction of a power of two) or it can be set to 1 which is the natural sector size of FRAM. This leads to two consequences: (1) the volume information structure *fm25h_volume_info_t* which is set in the tool *tos-storage-fm25h* should define both *base* and *size* as *uint32_t* instead of *unit8_t* to accommodate the larger size numbers (this is internal to the chip specific tool chain and has no consequences to the interfaces), and (2) the *erase* command's parameter *eraseUnitIndex* (and the corresponding *eraseDone* event's) should likewise be *uint32_t*, instead of *uint16_t*. This results from the erase unit size which is of size 1 as well. However, here the advantage of an artificially introduced larger erase unit size becomes obvious. For erasing larger memory regions less function calls were necessary then.

A last comment is given on the DirectModify interface signature. It is perhaps preferable to name all completion events in the same manner and so rename the completion event of *modify(...)* to *modifyDone(...)*.

IV. CONCLUSION

FRAM can replace flash memory when low-power operation in combination with short write access times is of importance. Drawback is the smaller maximum capacity per chip. If the capacity is a limiting factor, several chips can be clustered as proposed. Several HIL interfaces originally intended for flash memory were successfully implemented. Recommenda-

tions were made to the DirectStorage and DirectModify interfaces, both being still in draft status and open for changes.

ACKNOWLEDGMENT

S. B. likes to thank Helmut Ernst for his support in the hardware development of the sensor nodes used for this work.

REFERENCES

[1] V. Handziski, J. Polastre, J.-H. Hauer, C. Sharp, A. Wolisz, D. Culler, D. Gay, "TEP2 – Hardware Abstraction Architecture". http://www.tinyos.net/tinyos-2.x/doc/html/tep2.html. Date of access: 2009-05-25

[2] J. Hui, "Implementation of the TEP103 for the ST M25P serial code flash". http://tinyos.cvs.sourceforge.net/viewvc/tinyos/tinyos-2.x/tos/chips/ stm25p. Date of access: 2009-05-28

[3] D. Gay, J. Hui, "TEP103 – Permanent Data Storage (flash)". http://www.tinyos.net/tinyos-2.x/doc/html/tep103.html. Date of access: 2009-05-25

[4] D. Moss, J. Du, P. Dutta, D. Ganesan, K. Klues, A. Martin, G. Mathur, "TEP128 - Platform Independent Non-Volatile Storage Abstractions". http://www.tinyos.net/tinyos-2.1.0/doc/html/tep128.html. Date of access: 2009-05-25