$\mathcal{R}^3$: Reliable Over-the-Air Reprogramming on Computational RFIDs

DIE WU, LI LU, MUHAMMAD JAWAD HUSSAIN, and SONGFAN LI, University of Electronic Science and Technology of China
MO LI, Nanyang Technological University
FENGLI ZHANG, University of Electronic Science and Technology of China

Computational Radio Frequency Identification (CRFID) tags operate solely on harvested energy and have emerged as viable platforms for a variety of ubiquitous sensing and computation applications. Due to their battery-less nature, these tags can be permanently deployed in hard-to-reach places where the possibility of tag access is eliminated. In such scenarios, maintaining and upgrading the tag's firmware becomes infeasible because programming tools, including wired interface and PC-based software, are required to erase, modify, or reprogram the microcontroller unit's memory. Such limitations necessitate the demand for an over-the-air (OTA) scheme, which can wirelessly reprogram or upgrade the firmware in CRFID tags.

In this article, we present $\mathcal{R}^3$—a reliable OTA reprogramming scheme that is compliant with EPC protocol and requires no hardware upgrade to RFID reader or CRFID tag. We demonstrate our scheme on three platforms, which include both software-defined as well as chip-based CRFID tags, that is, WISP5.1 and Optimized WISP (Opt-WISP), and Spider tag, respectively. The selection also includes both the FLASH- and FRAM-based microcontrollers. We extensively evaluate our scheme in terms of several metrics, including overall system delay, time and energy overhead, and success rate in line with interrogation range. We foresee our endeavor to offer the viability of OTA reprogramming and firmware upgrade for CRFID tokens under practical situations.

CCS Concepts: • **Computer systems organization → Firmware**; *Sensors and actuators*; • **Software and its engineering → Software configuration management and version control systems**;

Additional Key Words and Phrases: Computational RFID, OTA reprogramming, firmware upgrade, EPC

ACM Reference format:
Die Wu, Li Lu, Muhammad Jawad Hussain, SongFan Li, Mo Li, and Fengli Zhang. 2017. $\mathcal{R}^3$: Reliable Over-the-Air Reprogramming on Computational RFIDs. *ACM Trans. Embed. Comput. Syst.* 17, 1, Article 9 (September 2017), 25 pages.
https://doi.org/10.1145/3070720

A preliminary version of this work appeared in *IEEE International Conference on RFID* under a slightly different title [52].

This work is jointly supported by the National Natural Science Foundation of China under Grant 61472068 and 61502085, and China Postdoctoral Science Foundation under Grant 2015M570775.

Authors' addresses: D. Wu, Li Lu, M. J. Hussain, and S. Li, School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu, 611731, China; emails: wudie.uestc@gmail.com, luli2009@uestc.edu.cn, {yinuodie, yinuodie}@gmail.com; M. Li, School of Computer Science and Engineering, Nanyang Technological University, 639798, Singapore; email: limo@ntu.edu.sg; F. Zhang, School of Information and Software Engineering, University of Electronic Science and Technology of China, Chengdu, 610054, China; email: fzhang@uestc.edu.cn.

1 INTRODUCTION

With fully passive and radio frequency (RF)-powered architecture, computational radio frequency identification (CRFID) tags are equipped with ultra-low power microcontroller units (MCUs) and multiple different sensors, which can be used for numerous computational and sensing scenarios. Since the inception of the WISP project [33], CRFIDs have emerged as an appealing platform for both academia and industry. Generally, we categorize WISP-based systems as "software-defined" CRFID tags in which both EPC protocol [13] and computational tasks are executed in the MCU. The other family, which we term as "chip-based" CRFID tags, uses commercial chips to execute EPC protocol and provide auxiliary power and communication interface for external MCU and sensors. Over the past decade, a variety of diverse sensing and computation applications have been demonstrated on these systems, including movement recognition [4, 6, 21], health monitoring [10, 53], environment sensing [17, 29], access control [19, 41], cardinality estimation [16, 60, 61], and ubiquitous computing [24, 26, 31, 32].

A prime limitation in these systems is that the MCU in the CRFID tag is programmed with only a specific firmware during deployment time. To update or replace a firmware, users have to physically connect each tag with a specific programming adapter and download the patch or new firmware through the PC-based software. The situation would further deteriorate if CRFIDs are deployed in hard-to-reach places or when their scale grows. Known examples include structural monitoring [23, 57], bio-signal sensing [55], implanted glucose monitoring [53], and RFID sensor network [20]. In these applications, maintaining the firmware through physical access becomes cumbersome. Such limitation necessitates the demand for a flexible over-the-air (OTA) reprogramming scheme, whereby a commercial RFID reader is able to wirelessly reprogram CRFID tags through EPC protocol.

Despite the fact that the concept of OTA programming is a hot topic specifically in wireless sensor networks (WSNs), the study on wireless reprogramming in CRFID systems has not been performed before. Unlike battery-assisted wireless sensor nodes, transiently powered CRFIDs offer no guaranteed reprogramming operations as they cannot afford the high-energy budget required for reprogramming the MCU memories. Furthermore, RFID reader is restricted to transfer the data in few-byte segments, for which the tag has to calculate the CRC and acknowledge the successful receipt within the stringent link timings of EPC protocol. Importantly, without an operating system to coordinate the firmware execution and wireless reprogramming, the pre-installed firmware on CRFID tag cannot perform a self-modification as the memory of the firmware is protected during its execution. Therefore, we need an energy-aware OTA scheme that is not only able to reliably reprogram the CRFID, but also compatible with EPC protocol and commercial RFID reader.

To address this issue, this article presents $\mathcal{R}^3$, a $\mathcal{R}$eliable OTA $\mathcal{R}$eprogramming scheme for computational $\mathcal{R}$FID tags. The faced challenges and their solutions are given herein: (a) Current EPC protocol does not provide any specific command for reprogramming operation. In our scheme, we encode routine *Write* command to transmit instructions and customized data. On the tag side, a boot-loader is designed to communicate with RFID reader and coordinate the tag operations including firmware execution, reprogramming and error correction. (b) The communication rate in RFID system is very low, it might not be possible to transfer complete firmware image in a single operation. As a solution, we partition the target firmware image in multiple segments in accordance with link timings and MCU memory. (c) The computational state will be completely lost once the power failure occurs. In our scheme, CRFID tags transit between active mode and low power mode to conserve harvested power. (d) For tags with FLASH based MCU, the writing operation is not reliable when the supply voltage drops below 2.2V. To address this problem, we propose an error detection and correction mechanism based on the concept of *bitmap*.

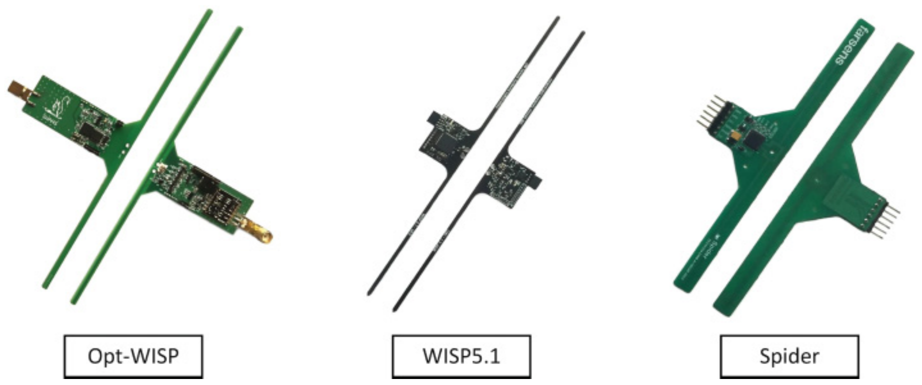


Fig. 1. Software-defined CRFIDs Opt-WISP and WISP5.1 and chip-based CRFID Spider.

We implement and evaluate $\mathcal{R}^3$ on three tags, which belong to both families of CRFIDs. As shown in Figure 1, the software-defined CRFID tags include Opt-WISP (left) and WISP5.1 (middle). The former is primarily based on Intel-WISP4.1DL and uses MSP430F2132 as its MCU while latter includes MSP430FR5969. Third is the chip-based CRFID tag Spider (right) by Farsens that uses Andy100 CRFID chip [14], and we interface MSP430F2132 as MCU. Similarly, we also evaluate $\mathcal{R}^3$ with different embedded memories, that is, FLASH memory in MSP430F2132 and Ferroelectric RAM (FRAM) in MSP430FR5969. For all three systems, we develop a user interface based on Impinj Software Development Kit (SDK) [22]. We evaluate our system from following aspects: overall system delay in reprogramming 512-byte firmware image, time and energy overhead for reprogramming operation, and success rate within the reader's interrogation range. All said parameters are evaluated for three CRFID tags and a comparison is presented.

In summary, our endeavor offers viability of OTA reprogramming and upgrade for CRFID systems. Following are the chief contributions of our work:

- $\mathcal{R}^3$ is an OTA reprogramming scheme for both software-defined and chip-based CRFID tags. It is fully compatible with EPC-C1G2 protocol without demanding any modifications to RFID reader or hardware upgrade to CRFID tags.
- $\mathcal{R}^3$ efficiently maps the firmware image in accordance with link timings and MCU memories. Such memory arrangement not only offers flexibility in reprogramming operation but also conserves harvested energy and reprogramming time.
- Based on the concept of *bitmap*, we present an error detection and correction mechanism to ensure the reliability of OTA reprogramming.
- We implement and evaluate our system for three CRFID tags including Opt-WISP, WISP5.1 and Spider. Our selection also includes two type of MCU memories, including FRAM and FLASH.

Admittedly, CRFID system becomes vulnerable as any reader can reprogram the CRFID tag as far as it follows the EPC protocol. This shortcoming can be resolved by authenticating the RFID reader, which we foresee as our future work.

The rest of this article is organized as follows. We introduce the related work in Section 2 and describe the design of our system in Section 3. The error detection and correction is discussed in Section 4. The implementation and evaluation are explained in Section 5. The article concludes in Section 6.

2 RELATED WORK

OTA Programming in WSNs. We find promising OTA programming schemes in WSNs. XNP [8] is a single hop protocol that broadcasts the intended firmware, while an energy efficient scheme is presented in [38] that only distributes the changes to currently running programs. Based on a multi-hop network, Deluge [7] provides a reliable data dissemination for all sensor nodes while [43] proposed a code distribution mechanism specifically for Mica-2 Motes. Aqueduct [34] and Tiny-Cubus [28] proposed a scheme to distribute programs to the selected nodes. Pando [11, 12] accomplishes the data dissemination process by transmitting fountain-encoded packet over constructive interference and pipelining. As industrial products, WaspMote can be wirelessly programmed through Zigbee [25], while a user can use Bluetooth-enabled smartphone to reprogram the nRF51822 chip [1].

Firmware Execution in CRFIDs. As OTA reprogramming is a new concept in CRFID field, we only find few relevant works on software-defined CRFID tags. In FirmSwitch [54], the authors remotely switch the behavior of CRFID tag by selecting amongst the pre-programmed firmwares. A pre-determined look-up table approach is followed in [35], however, the MCU is restricted to follow a pre-determined execution flow. Moreover, the authors presented an energy aware scheduling scheme [36] that maps the harvested voltage with appropriate firmware for execution. Mementos [37] enables the CRFID tag to complete long-running computations by breaking a single program into interruptible executions. DINO [27] is a software-based model that addresses the intermittent execution through checkpointing and reducing the complexity of programming. Wisent [44] transfers data through changing the frame length of *BlockWrite* and stores firmware only in WISP 5. DewDrop [3] effectively makes use of harvested energy while treating iterative tasks as a scheduling problem to balance the task demands in relation to available energy. Researchers carried out the performance evaluation for link layer of RFID systems [5].

Hybrid Designs for CRFIDs. Pioneered by Intel WISP project [33], a large variety of software-defined CRFID systems have been developed. BlueDevil WISP [39], EEGWISP [9], SoCWISP [30], WISPCam [29], WISPs/g [15], SolarWISP and FrankenWISP [17], and Moo [57] are some of the chief designs. Besides WISP and its various hybrids, we find two commercial CRFID chips, Andy100 by Farsens and SL-series by AMS AG. Both chips execute the EPC standard and additionally provide the power and communication interface (SPI) for external modules like MCU or the digital sensors. However, no OTA operation is provided in both cases. The CRFID tags based upon Andy100 chip are used to interact with mobile robots [51]. The SL900A chip is used for soil moisture monitoring [2] and reading the tagged objects [49]. Moreover, a recent work [56] describes the design of a CRFID chip, which follows EPC C1G2 protocol and collects sensor data through SPI interface.

Data Transfer on CRFIDs. Recent works have considered the data transmission between the CRFID tag and the reader. Buzz exploits the PHY layer collisions to reduce the message loss rate [50]. In Harmony [62], researchers proposed an efficient data transfer scheme that enables CRFID tag to transfer high volume data to a commercial reader. With an aim to improve the throughput of RFID communication, BLINK [59] utilizes the RSSI and package loss rate while Flit [18] leverages the idle slots. QuarkNet [58] enables continuous communication under severe harvesting conditions.

3 SYSTEM DESIGN

We discuss our design from both RFID reader and CRFID tag sides. Broadly speaking, our system has three main phases: reprogramming, execution, and verification. An overview of the proposed scheme is shown in Figure 2. For reprogramming, on the reader side, the firmware image is

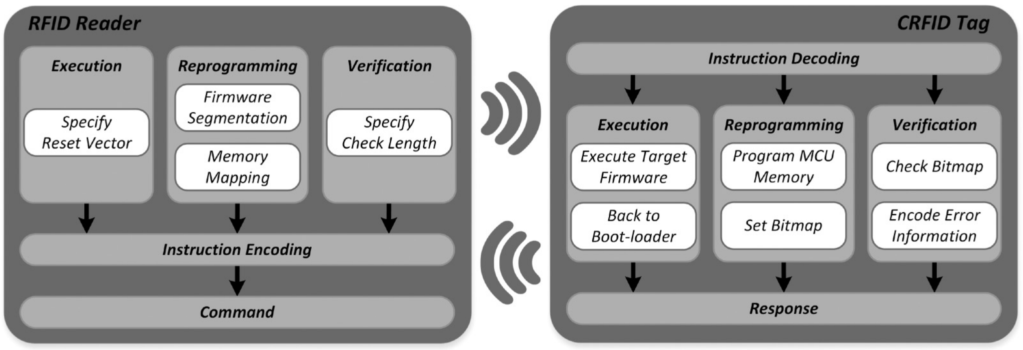


Fig. 2. An overview of the proposed OTA reprogramming scheme $\mathcal{R}^3$.

partitioned into multiple segments in accordance with the type of CRFID tag. Then, the reprogramming instructions, firmware segments and their memory addresses are embedded in the *Write* command. Towards the tag side, the boot-loader decodes the reprogramming instruction, programs the MCU memory, and sets the corresponding bits of the *Bitmap*. To verify the programmed segments, the check length is embedded in the verification instruction. Upon reception, CRFID tag checks the *Bitmap* and response the encoded error information. Moreover, the user can select an intended firmware through embedding its reset vector address in the execution instruction. In this case, the CRFID tag will execute the target firmware and switch back to boot-loader when the execution completes.

3.1 Firmware Segmentation

The battery-assisted devices, like WSNs, can afford the high-energy budget required for reprogramming the MCU memory. However, CRFID tags are fully passive devices that solely function on transient harvested power. If the size of the transmitted firmware is too large, then the reprogramming process might be terminated by the power failure. Consequently, the CRFID tag will lose its running state and RFID reader has to restart the reprogramming operation. Therefore, the firmware image should be fragmented into multiple small segments that can be practically transferred and reprogrammed within the energy budget constraints.

In first intuition, the size of the fragmented segments should be as small as possible. However, as MCU memories are byte- and word-programmable, the firmware reprogramming operation is the combination of multiple byte-wise or word-wise writing operations. In practice, we observe that the byte-wise write operation consumes nearly the same time and energy as that for word-wise write. Therefore, the firmware image is fragmented into multiple word-length segments, so we cannot only reduce the overall reprogramming time, but also provide reliability to our scheme. Particularly, the CRFID chip in chip-based tags communicates with external MCU through 8-bit SPI, the firmware image for chip-based CRFID tags is fragmented into byte-length segments.

Algorithm 1 gives an illustration for the proposed firmware image segmentation procedure. Once a specific tag and firmware image is selected, RFID reader loads the firmware image and measures its *Length* in bytes. Then, the reader searches the received tag's ID (*TagID*) from the sets of EPC numbers for both software-defined and chip-based CRFID tags (Set_s and Set_c, respectively). In case the tag's ID belongs to Set_s, the firmware image is appended with a *0xFF* when the *Length* is odd. After that, the firmware image is partitioned into multiple word-length segments and saved in

ALGORITHM 1: Firmware Image Segmentation (Executed on Reader Side)

Input: EPC number of the tag (*TagID*), firmware image to be reprogrammed (*FirmImage*[ ]), firmware image length in bytes (*Length*), set of EPC numbers for software-defined CRFIDs (Set_s) and set of EPC numbers for chip-based CRFIDs (Set_c).

Output: The i^{th} firmware image segment for software-defined CRFIDs ($SSegment_i$) and chip-based CRFIDs ($CSegment_i$).

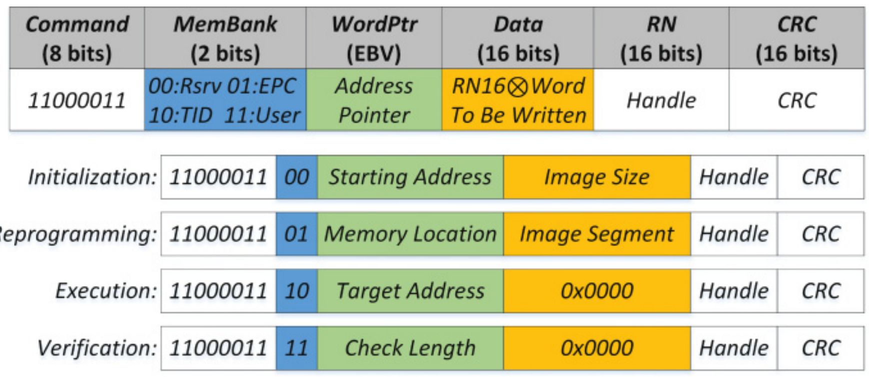
```
if $TagID \in Set_s$ then
    if $Length \bmod 2 = 1$ then
        $FirmImage[Length] = 0xFF$
    end
    for $i \leftarrow 0$ to $\lceil Length/2 \rceil - 1$ do
        $SSegment_i = FirmImage[2i] \,||\, FirmImage[2i+1]$
    end
else if $TagID \in Set_c$ then
    for $i \leftarrow 0$ to $Length-1$ do
        $CSegment_i = FirmImage[i]$
    end
else
    return
end
```

an array of *SSegment*. When the received *TagID* belongs to Set_c, the firmware binary would further be partitioned into multiple byte-length segments and saved in the array of *CSegment*.

3.2 Instruction Encoding

3.2.1 Selection between Write and BlockWrite. With an aim to provide an EPC-compliant OTA reprogramming scheme, we are restricted to use the existing command offered by EPC protocol. However, existing EPC protocol offers no specific instructions for OTA reprogramming. In EPC protocol, only two of the Access Commands (*Write* and *BlockWrite*) can be used to transmit customized data from the reader to the tag [13]. The *Write* command is one of the mandatory commands in *Access Round*, which can transfer a single 16-bit customized data to the tag. The *BlockWrite* command can transfer any number of 16-bit words so there is less back-and-forth communication between the reader and the tag, as compared with *Write* command. Nevertheless, *Write* command is more suitable for our scheme than *BlockWrite* command. On one hand, the firmware image is fragmented into multiple byte- or word-length segments based on the type of the CRFID tag to be reprogrammed. The 16-bit data field of *Write* is enough for either byte-length segment or word-length segment. The time and energy overhead for processing *Write* command is lower than that for *BlockWrite* command as the command length of former is shorter. On the other hand, *BlockWrite* is an optional command that is not supported by all CRFID tags, for example, chip-based Spider tag. As a result, we adopt the mandatory *Write* command and use it to transfer the fragmented segments to the CRFID tag.

3.2.2 Instruction Encoding for Software-defined CRFID Tags. For software-defined CRFIDs, we encode the instructions in *Write* command by embedding the parameters in three fields: 2 bits of *MemBank* field, 16 bits of *WordPtr* field and 16 bits of *Data* field, as shown in Figure 3. Normally, the tag memory is logically separated into four distinct banks, that is, Reserved memory, EPC memory, TID memory, and User Memory. However, software-defined CRFID tags execute the EPC protocol in the MCU and the RAM is used instead of the aforementioned memories. Therefore, we can



Command (8 bits)	MemBank (2 bits)	WordPtr (EBV)	Data (16 bits)	RN (16 bits)	CRC (16 bits)
11000011	00:Rsrv 01:EPC 10:TID 11:User	Address Pointer	RN16⊗Word To Be Written	Handle	CRC

Initialization:	11000011	00	Starting Address	Image Size	Handle	CRC
Reprogramming:	11000011	01	Memory Location	Image Segment	Handle	CRC
Execution:	11000011	10	Target Address	0x0000	Handle	CRC
Verification:	11000011	11	Check Length	0x0000	Handle	CRC

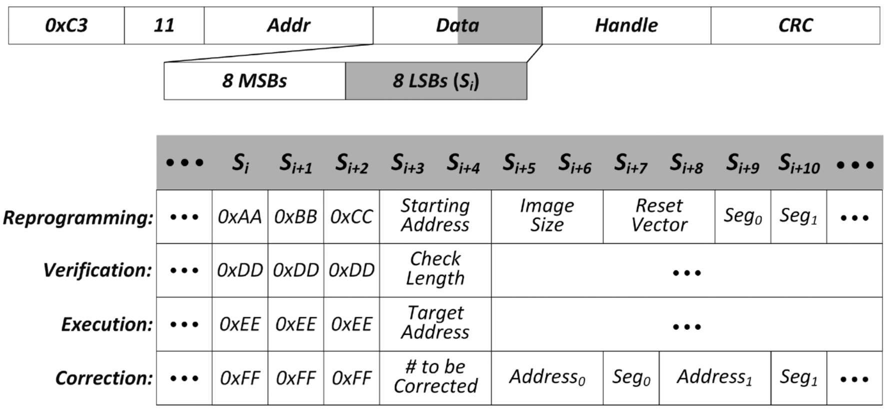
Fig. 3. Instruction encoding for software-defined CRFIDs.

utilize the *MemBank* field as a sign to differentiate between following instructions: initialization instruction, reprogramming instruction, execution instruction, and verification instruction.

For initialization instruction, the *MemBank* field is coded as 00_b, and the following *WordPtr* and *Data* fields indicate the starting address and size of the firmware image to be reprogrammed. For reprogramming instruction, the *MemBank* field is coded as 01_b, and the *WordPtr* and *Data* fields represent the memory location and its corresponding 16-bit firmware image segment, which is partitioned during the firmware image segmentation phase. For execution instruction, the *MemBank* field is coded as 10_b and the *WordPtr* field represents the address of reset vector for target firmware to be executed. For verification instruction, that is, the reader instructs the tag to check the erroneously reprogrammed image segments, the *MemBank* field is coded as 11_b and the *WordPtr* field represents the number of image segments to be verified. Before transmission, the *Data* field for both execution and verification instruction is set as 0x0000. Particularly, the reprogramming instruction can also be used for error correction. In this case, the erroneous reprogrammed segment and its corresponding memory location are embedded in *Data* and *WordPtr* fields, respectively.

3.2.3 *Instruction Encoding for Chip-based CRFID Tags.*

Unlike software-defined CRFID tags, the chip-based CRFID tag (i.e., Spider tag in our scheme) is interfaced with an external MCU through SPI interface. The internal architecture of the Andy100 chip restricts the data transfer to a maximum of 8 bits at one time. When the *MemBank* field of a *Write* command is set as 11_b, the Andy100 chip will transfer the 8-bit least-significant bits (LSBs) of the *Data* field to the external MCU through 8-bit SPI. Based on such a working scheme, the following procedure shown in Figure 4 is devised: we define the 8 LSBs of *Data* field as S_i in the ith *Write* command when the *MemBank* field is set as 11_b. To start the reprogramming operation and initialize the memory to be reprogrammed, the instruction is defined as $S_i||S_{i+1}||S_{i+2} = AA_h||BB_h||CC_h$, which is followed by the starting address (contained in $S_{i+3}||S_{i+4}$) and the number of firmware image segments (contained in $S_{i+5}||S_{i+6}$). The following two 8 LSBs ($S_{i+7}||S_{i+8}$) represent the reset vector of the firmware image to be reprogrammed. Then, the fragmented image segments are embedded one by one from S_{i+9}. For firmware verification and error detection, the opcode is defined as $S_i||S_{i+1}||S_{i+2} = DD_h||DD_h||DD_h$, the following two 8 LSBs represent the number of firmware segments to be checked. To execute the firmware, the default instruction is defined as $S_i||S_{i+1}||S_{i+2} = EE_h||EE_h||EE_h$, which is followed by the reset vector address of the target firmware embedded in $S_{i+3}||S_{i+4}$. In particular, a correction instruction is used to correct the erroneously programmed



	•••	S_i	S_{i+1}	S_{i+2}	S_{i+3}	S_{i+4}	S_{i+5}	S_{i+6}	S_{i+7}	S_{i+8}	S_{i+9}	S_{i+10}	•••
Reprogramming:	•••	0xAA	0xBB	0xCC	Starting Address		Image Size		Reset Vector		Seg_0	Seg_1	•••
Verification:	•••	0xDD	0xDD	0xDD	Check Length		•••						
Execution:	•••	0xEE	0xEE	0xEE	Target Address		•••						
Correction:	•••	0xFF	0xFF	0xFF	# to be Corrected		$Address_0$		Seg_0	$Address_1$		Seg_1	•••

Fig. 4. Instruction encoding for chip-based CRFIDs.

image segments. To start the error correction phase, the instruction is pre-fixed with $S_i||S_{i+1}||S_{i+2} = FF_h||FF_h||FF_h$, while the number of total segments to be corrected are passed in the next byte $(S_{i+3}||S_{i+4})$. The following bytes from S_{i+5} contain the memory address and correct segment of each identified error. In case a large number of error segments are encountered, reader repeats the same procedure in a sequential manner.

3.3 Instruction Decoding and Execution

In software-defined CRFIDs, the boot-loader executes the EPC protocol, decodes the commands and parses the parameters embedded in three fields of *Write* command. In contrast, chip-based CRFID tag first decodes the EPC commands within its CRFID chip and triggers the external MCU when the *Write* command in which the *MemBank* field is set as 11_b is received. Based on different working schemes, the instruction decoding process is executed differently for both types of CRFIDs.

In case of software-defined CRFIDs, once the *Write* command is received, the boot-loader parses the command and checks the CRC value. If an invalid CRC is received, then boot-loader ignores the command and waits for the next instruction. Otherwise, it executes the instruction and replies a *Success* to the reader after the execution is complete. As stated earlier, the value of *MemBank* is the opcode that indicates the next task on the tag, that is, for *MemBank*=00_b, the boot-loader initializes the memory to be reprogrammed based on the starting address and image size parsed from *WordPtr* and *Data* fields, respectively. For *MemBank*=01_b, boot-loader sets the tag to reprogramming mode and writes the image segment of *Data* field to the address stored in the *WordPtr* field. Before replying the *Success*, boot-loader verifies the reprogramming results and sets the corresponding bit in the bitmap region (the detailed process for bitmap is discussed in Section 4). For *MemBank*=10_b, the boot-loader shifts the content in *WordPtr* field (reset vector address of target firmware) to the *Program Counter* register that holds the address of the next instruction to be executed. For *MemBank*=11_b, boot-loader sets the tag to verification mode, collects the positions of the erroneously reprogrammed segments based on the number contained in *WordPtr* (number of segments to be verified) and encodes the error information using collected data. The verification and error correction process will be explained in detail in Section 4.

For chip-based CRFIDs, the CRFID chip will initialize the SPI communication with external MCU once a valid *Write* command accessing the data stored in *User Memory* is received. In the external MCU, the boot-loader receives the SPI data, labels the *i*th received 8-bit data segment as

S_i and stores them in the RAM. Whenever the reprogramming opcode (AA_h, BB_h, CC_h) is received, the boot-loader sets the MCU to reprogramming mode and initializes the memory based upon next four bytes (S_{i+3}, S_{i+4}, S_{i+5} and S_{i+6} that contain the starting address and the length of the firmware image). Then, the reset vector ($S_{i+7} \| S_{i+8}$) is placed at the end of the initialized memory segment. After that, the following received image segments (from S_{i+9}) will be written one after the other from the starting address ($S_{i+3} \| S_{i+4}$). If an execution opcode (three consecutive EE_h) is received, then the boot-loader stores the next two bytes of SPI data in *Program Counter* to execute the target firmware. In case a verification opcode (three consecutive DD_h) is received, boot-loader moves the tag into verification mode, prepares for any error information based the next two bytes ($S_{i+3} \| S_{i+4}$). When the correction opcode (three consecutive FF_h) is received, boot-loader corrects the erroneous firmware segments based on the number to be corrected ($S_{i+3} \| S_{i+4}$) and the image segments along with their corresponding memory locations.

3.4 Memory Management

To minimize the reprogramming overhead and provide our scheme with stability, the *memory management* phase relates to allocating the MCU memory to boot-loader, firmware images and their corresponding reset vectors in accordance with the MCU memory before the CRFID tag is deployed. In particular, a specific memory region is reserved for the purpose of error detection and correction. In our case, the Opt-WISP and Spider tags include FLASH-based MCU (MSP430F2132), whereas WISP5.1 includes an FRAM-based MCU (MSP430FR5969), which is one of the latest additions to the MSP family of ultra-low power MCUs.

For FLASH-based MCU, the main memory is physically divided into multiple 512-byte segments. Despite single byte or word can be written to the FLASH memory, the segment is the smallest unit for data erasure [45, 46]. To reprogram the tag with FLASH-based MCU, we first need to erase the complete memory segments to be reprogrammed. On the contrary, for FRAM-based MCU, there is no specific memory segmentation and it can be logically partitioned by the memory protection unit [47, 48]. To reprogram the tag with FRAM-based MCU, we simply need to write the new firmware image at the specific memory location without performing the erase operation, as in FLASH-based MCUs.

In our scheme, the memory management for CRFIDs with FLASH-based MCU (Opt-WISP and Spider) and FRAM-based MCU (WISP5.1) is shown in Figure 5. For tags with FLASH-based MCU, our scheme places each firmware image at the beginning of a memory segment, and the reset vector of each firmware image is required to be placed at the last two bytes of each segment. As a result, there is no overlapping in memory segments for different firmware images. Such placement cannot only provide the flexibility of erasing each individual firmware image, but also minimize the erasing overhead in terms of time and energy. For tags with FRAM-based MCU, as the FRAM module has no specific memory segmentation and it can be overwritten in a similar fashion to RAM without any special requirements. The firmware images and their reset vectors are stacked from the beginning of FRAM.

As the execution of CRFID tags can be terminated by the frequent power failure, our system should be able to restart its execution from a preset point. By default, whenever the MCU is powered-up, the *Program Counter* in either FLASH-based MCU or FRAM-based MCU is loaded with the address contained at reset vector location (*0xFFFE*). We leverage this feature for stability and place the reset vector of boot-loader at *0xFFFE*. As a result, whenever the tag is powered on for the first time or a power failure occurs, the execution will start from boot-loader. In addition, to reduce the time and energy overhead incurred by verifying the reprogramming results, a specific memory region (*Bitmap*), which stores the verification results of the reprogrammed firmware image is allocated during *memory management* phase. In our case, one memory segment

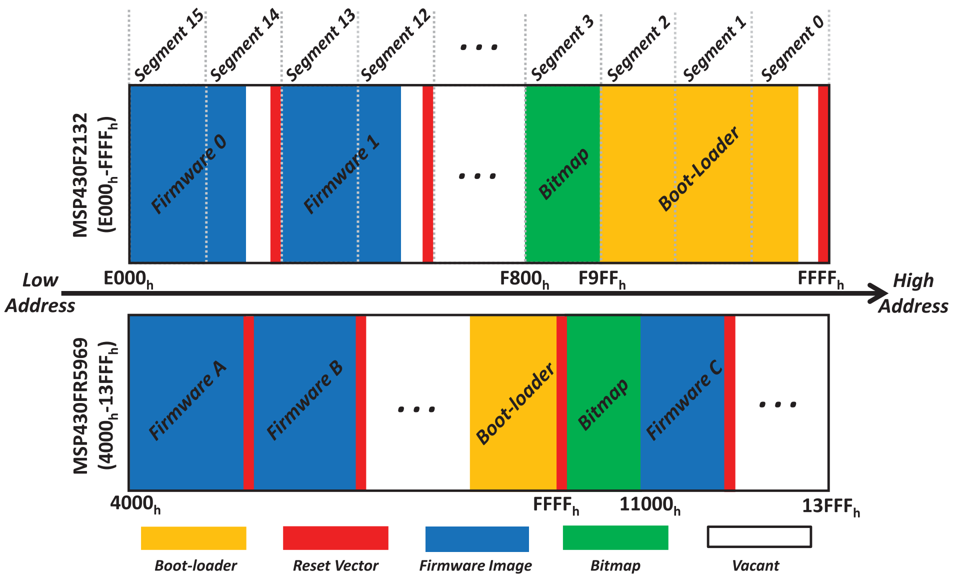


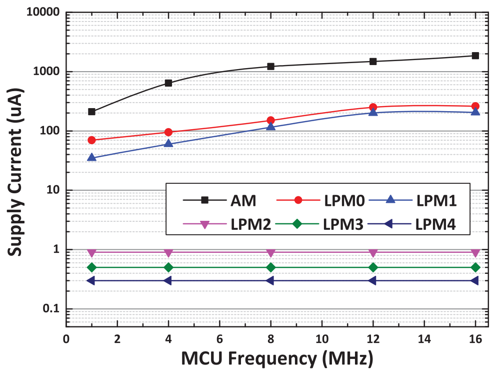
Fig. 5. Memory management for CRFIDs with FLASH-based MCU (MSP430F2132) and FRAM-based MCU (MSP430FR5969).

(*Segment 3*) is allocated for Opt-WISP and Spider tags, while a 4KByte memory region (from *0x10000* to *0x10FFF*) is allocated for WISP5.1.

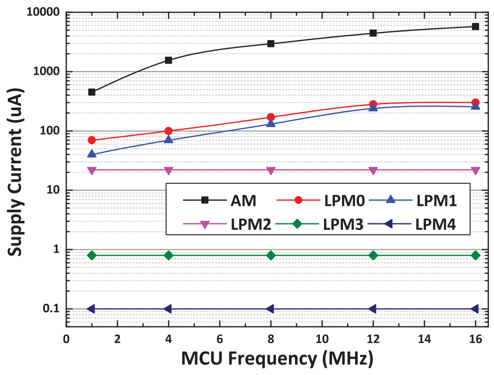
3.5 Power Management

CRFID tags are power constrained devices that use the MSP430 series of MCUs to perform the computation and sensing tasks. A key challenge of CRFIDs system is that the execution on existing tags is not guaranteed as the power in the energy storage capacitor might drain before the task completes. In fact, CRFID tags receive nearly a constant amount of power when the interrogation range is fixed and the tag is not performing any task (sleep mode). However, when the CRFID tags are performing computation or sensing tasks, the power consumption is significantly higher than harvested, and the energy storage capacitor discharges to provide power for the tasks. In case the power in the energy storage capacitor is totally drained, the task will be terminated.

To address this issue, we first look into the current consumption of the MCU. Broadly speaking, the MSP430 MCUs provide one active mode and several software-selectable low power modes (from LPM0 to LPM4) to extend the lifetime of the energy constrained devices. Figure 6 shows the current consumption of these operating modes for MSP430FR5969 and MSP430F2132, respectively. The figure shows that the supply current in active mode ranges from 210μA to 1845μA for MSP430FR5969 when the working frequency increases from 1MHz to 16MHz. Similarly, for MSP430F2132, the supply current in active mode are 450μA at 1MHz and 5750μA at 16MHz. On the contrary, the supply current for LPM4 is 0.3μA for MSP430FR5969 and 0.1μA for MSP430F2132. Moreover, the current consumption for LPM4 is the lowest among all these operating modes and it is orders of magnitude lower than that for active mode for both MCUs. If the CRFID tag is in LPM4, then the CPU is stopped and all clocks are disabled. Therefore, the total current consumption would be in a few micro-amps so the harvested power can charge the energy storage capacitor.

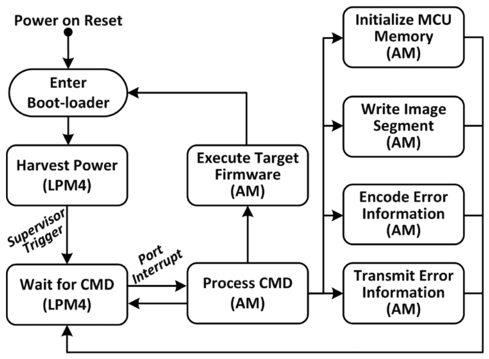


(a) Supply current for MSP430FR5969.

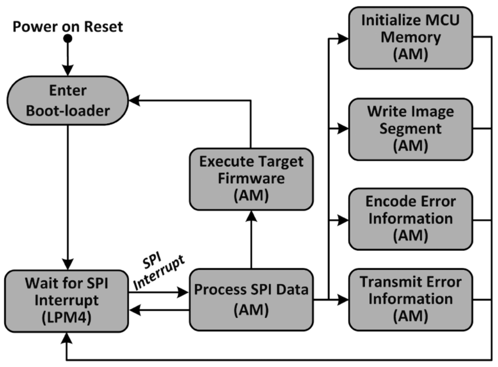


(b) Supply current for MSP430F2132.

Fig. 6. Supply current vs. MCU frequency for active and low power modes.



(a) State diagram for software-defined CRFIDs.


Power on Reset
Enter Boot-loader
Wait for SPI Interrupt (LPM4)
SPI Interrupt
Process SPI Data (AM)
Execute Target Firmware (AM)
Initialize MCU Memory (AM)
Write Image Segment (AM)
Encode Error Information (AM)
Transmit Error Information (AM)

(b) State diagram for chip-based CRFIDs.

Fig. 7. State diagram of power management for software-defined CRFID and chip-based CRFID tags.

In our scheme, we minimize the energy consumption of the CRFID tags by repetitively transiting between LPM4 and active mode. Despite the CPU is stopped in LPM4, the proposed scheme is still feasible because the interrupt event can wake up the MCU from LPM4. Once waked up, the MCU works in active mode to serve the request and restores back to LPM4 on its return from the interrupt service routine.

The state diagram of power management for both software-defined and chip-based CRFIDs is shown in Figure 7. For software-defined CRFID tags, when they are within the interrogation range of a reader, they start to harvest power and the voltage supplied to MCU begins to rise. When the voltage touches approximately 1.75V, the MCU is powered on and starts to load and execute the boot-loader. In our design, the boot-loader immediately enters LPM4 to wait for the hardware interrupt (supervisor trigger) from the energy supervisor. Such design avoids the oscillation of the system during the startup. When sufficient power is harvested (the voltage increases to around 1.9V), the energy supervisor generates a hardware trigger and boot-loader starts to wait for the EPC command from the reader in LPM4. When the communication starts from the reader's side, each symbol of the command would result in a port interrupt, which wakes up the MCU to work in active mode. The boot-loader stores the received symbol and moves back to LMP4 to wait for the next symbol. Once enough symbols are received, the boot-loader processes the command following

Table 1. Minimum Voltage Required for CPU Operation and Memory Writing

Microcontroller	Memory Type	Minimum V_{cpu}	Minimum V_{mem}
MSP430F149/413/1232	FLASH	1.8V	2.7V
MSP430F2012/2132/2247	FLASH	1.8V	2.2V
MSP430FR5736/5969	FRAM	1.8V	1.5V

EPC protocol if the command belongs to either select or inventory commands (i.e., *Select*, *Query*, *QueryAdjust*, *QueryRep*, *ACK*, and *NAK*). In case a *Write* command is received, the boot-loader executes the instruction based on the parameters parsed from *MemBank* field and moves back to LPM4 to wait for the next command. In particular, after processing the execution instruction, the MCU cannot directly move back to LPM4 to wait for the next command as the MCU needs to reboot to load the boot-loader after executing the target firmware. In case a *Read* command is received, tag replies the encoded error information in active mode and moves back to LPM4 upon completion.

For chip-based CRFID tag, the storage capacitor charges when the tag is brought within the read range. Different from software-defined CRFID tags, a voltage monitor is included to manage the power supply in chip-based CRFIDs. As a result, the voltage monitor feeds the MCU when the voltage of the energy storage capacitor is higher than 2.4V and disconnects the MCU when the voltage is lower than 1.8V. Due to this architecture, the power is sufficient to load the boot-loader and the repeated brownouts during the system startup are therefore avoided. After loading the boot-loader, the MCU moves into LPM4 and waits for the SPI interrupt from CRFID chip. During both *Write* and *Read* commands, the *MemBank* field is set as *User Memory*, which results in an SPI interrupt to wake up the MCU from LPM4. For *Write* command, the 8 LSBs of *Data* field are transferred from CRFID chip to the MCU. As a result, MCU stores these bits and returns to LPM4 to wait the next SPI interrupt. Once the encoded instruction is received, MCU moves to active mode, processes the instruction and finally moves back to LPM4. For *Read* command, the CRFID chip wakes up the MCU from LPM4, pulls 8 bits of encoded error information and transmits the data to the reader.

4 ERROR DETECTION AND CORRECTION

4.1 Error Source—Insufficient Power

To provide our system with reliability, whenever the CRFID tags check the packet integrity after receiving the instructions from the reader by verifying the CRC. However, in practice, the correctness of reprogramming results still cannot be guaranteed. Typically, the on-chip memory (FLASH or FRAM) in MCU shares a common voltage supply with the CPU instead of using separate power rails [40]. For most of the MCUs used in the CRFID tags, the minimum working voltage of CPU specified by the manufacturer is different from the voltage required for writing on-chip memories, as shown in Table 1. Generally, for FLASH-based MCUs, memory writing requires a higher voltage than the CPU operation ($V_{mem} > V_{cpu}$). If the supply voltage is higher than V_{mem}, then consistent and correct writing operation will be achieved. However, if the supply voltage is between V_{mem} and V_{cpu}, the FLASH memory can still be programmed but the correctness of memory writing is not guaranteed. Once the supply voltage is lower than V_{cpu}, the MCU will lose power and eventually stop working. In case of FRAM-based MCUs, the lowest voltage required for memory writing is lower than that for CPU ($V_{mem} < V_{cpu}$). If the supplied voltage is higher than V_{cpu}, then the results of memory writing would be consistently correct, while if the supplied voltage is lower than V_{cpu}, the MCU will stop working.

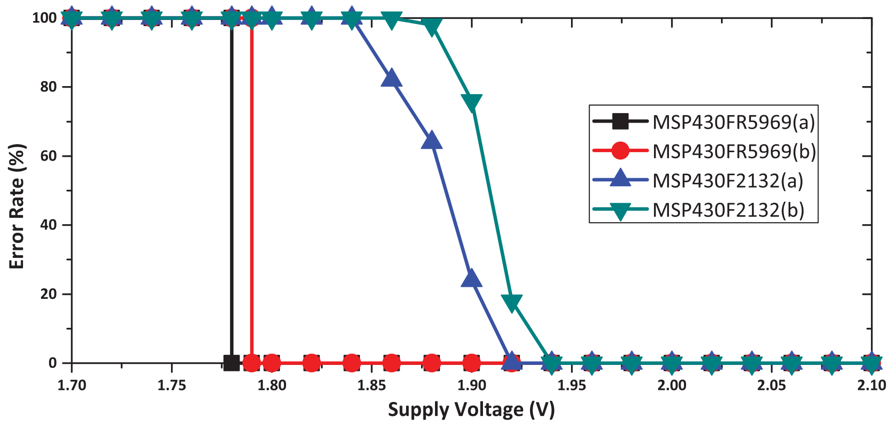


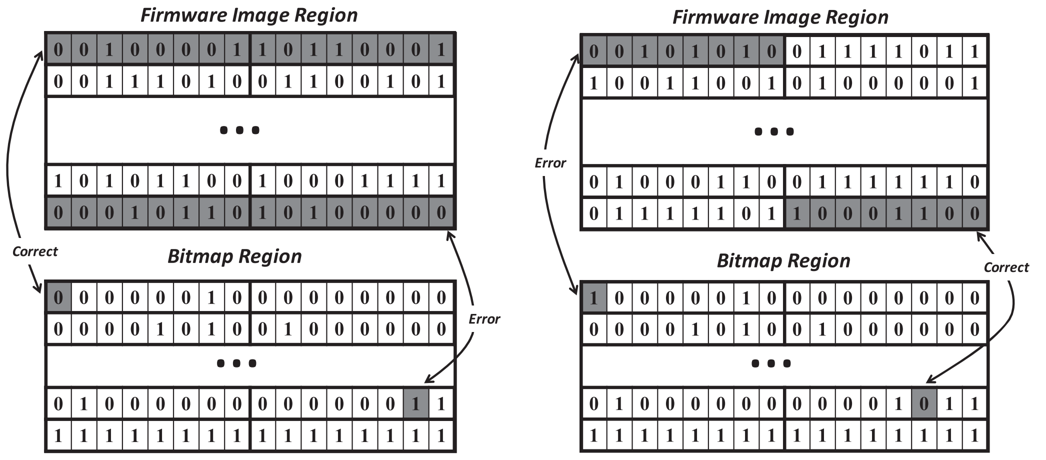
Fig. 8. Error rates for writing data to MSP430FR5969 and MSP430F2132 with different supply voltages.

To further understand the behavior of memory write operation under low voltage against the extend of errors incurred, we investigate the reprogramming scenario under decreasing voltage. We find that writing data to a memory cell is actually applying an electric field across the cell. When the supply voltage is lower than the minimum requirement specified by the manufacturer, the strength of the applied electric field might not be enough to change the logic state of the memory cell. By default, the logic state of the memory cells for both FLASH and FRAM are initialized with a logic state of "1." As a result, transition from "1" to "1" can obviously succeed and the error will only occur during the transition from "1" to "0." Therefore, to check the error rate at low voltage inputs, we write "0"s to the on-chip memory of both types of MCUs (MSP430F2132 and MSP430FR5969). As shown in Figure 8, for MSP430F2132, the error rate varies when the supply voltage is lower than the manufacturer-specified threshold. For MSP430FR5969, as the voltage requirement for FRAM is lower than that for CPU, the writing operation will always succeed if the CPU is working properly. In addition, we observe that the error rate slightly varies for different chips of the same MCU type.

As discussed in Section 3.5, once the tag is working in active mode, the power harvested is far from enough to meet the power consumed. Consequently, the energy stored in the storage capacitor is continuously consumed and voltage supplied to the MCU decreases. Take Opt-WISP as an example, the minimum voltage required for memory writing is 2.2V whereas the CPU necessitates 1.8V during program execution. When supply voltage drops between 3.3V and 2.2V, the reprogramming instruction will be executed successfully and writing results are correct. However, once supply voltage drops between 1.8V and 2.2V, the reprogramming instruction can be executed properly but the reprogramming errors might occur. When supply voltage drops below 1.8V, the CPU cannot work properly and as a result, the reader will lose the tag.

4.2 Error Information Encoding

The errors incurred in writing memory at low voltages necessitate an efficient mechanism for error detection and correction. As CRFID tags are highly constraint in harvested energy and limited by tight communication link timings, any error detection and correction mechanism should therefore be lightweight. To this end, we devise an error detection mechanism that maps the result of each reprogrammed image segment to a single corresponding bit at a specific memory area, which we term as *bitmap*. This way, the results of a success or failed write operation are mapped to a single



(a) Bitmap in software-defined CRFID tag.

(b) Bitmap in chip-based CRFID tag.

Fig. 9. Bitmap for software-defined and chip-based CRFID tags.

bit in the *bitmap*. The major advantage of bitmap is that the reader can validate the reprogramming operation by merely checking the bitmap, instead of reading all the written firmware segments. In a similar manner, the error correction becomes handy as the reader just re-sends the erroneously written (and identified) segments instead of sending whole firmware image again. In case of FRAM where $V_{mem} < V_{cpu}$, all the bits will be written correctly however our mechanism is still applicable to other families of MCUs in which the CPU voltage is lower than FRAM.

During firmware segmentation phase, the firmware image is partitioned into multiple byte-length or word-length segments based on the type of CRFID tag (Section 3.1). As shown in Figure 9, each bit in bitmap region is mapped to the reprogramming result of 16-bit and 8-bit firmware image segments for software-defined CRFID tags and chip-based CRFID tags, respectively. Since the memory reading operation can be realized across full voltage range of the MCU. Therefore, each time the boot-loader performs the reprogramming operation, it reads the address of the reprogrammed segment and compares the writing result with the data received from the reader. If the writing result is correct, then the corresponding bit in the bitmap region is set to "0"; otherwise, it is set to "1" (left unchanged). Consequently, even if the tag is starved of harvested power and cannot write the result into bitmap, the bit under operation will remain in its default state ("1"). Working this way, the tag validates the written segments against the image received from the reader, and writes the results to the bitmap region. We highlight that the bitmap is placed in the FLASH or FRAM memory instead of RAM, since the tag will lose the checking results once the power in the energy storage capacitor is drained.

Despite the use of bitmap can efficiently reduce the time overhead in checking the reprogramming errors, a more efficient transmission method is required to reduce the back and forth communication in error information acquirement. More in detail, once the verification instruction is received, boot-loader sets the CRFID tag into verification mode, loads the bitmap results based on the check length (number of reprogrammed firmware segments), and encodes the error information for erroneously reprogrammed segments. For encoding process, multiple 16-bit binaries are utilized to encapsulate the error information for N reprogrammed segments. Each 16-bit stack indicates the location of all the erroneous image segments amongst every 8 reprogrammed image segments. The first 8 bits represent the value of the bitmap while the last

ALGORITHM 2: Error Information Encoding (Executed on Tag Side)

Input: Number of Segments to be verified (N) and Byte read from bitmap region ($x_i, x_{i+1}, \ldots, x_{i+7}$), where x_i is defined as the i^{th} bit of bitmap region, $i \bmod 8 = 0$.

Output: 16-bit Error Information Piece (*ErrorInfo*).

```
$n = 1 + N/8$
while $n > 0$ do
    if $\exists x_j \neq 0,\ (j \in i, i+1, \ldots, i+7)$ then
        $offset = i/8$
        $(y_0, y_1, \ldots, y_7) = DecToBin(offset)$
        $ErrorInfo = (x_i, x_{i+1}, \ldots, x_{i+7}, y_0, y_1, \ldots, y_7)$
        $n = n - 1$
    end
    else
        $n = n - 1$
    end
end
```

Note: *DecToBin*(x) *is a function transforming* x *from Decimal to Binary format.*

8 bits represent the address offset of that value, which is denoted as *offset*. Algorithm 2 explains the encoding procedure for error information. The algorithm transforms the data format of address offset from decimal to binary, and appends the result to the end of the values read from bitmap region. For example, if the reprogramming error occurs in 19^{th} and 21^{th} image segment, the value of the third byte in bitmap would be *0x10*, that is, the value from 16^{th} bit to the 24^{th} bit of bitmap region is (0,0,0,1,0,1,0,0), the input of the encoding algorithm would be $(x_8, x_9, \ldots, x_{15}) =$ (0,0,0,1,0,1,0,0). In this case, the address offset in bitmap region is *offset* = 2, and the output of error information encoding would be (0,0,0,1,0,1,0,0,0,0,0,0,0,0,1,0). In such encoding algorithm, each piece of error information (16 bits) could indicate eight in maximum while only one in worst case position(s) of erroneously reprogrammed segment(s).

4.3 Error Location and Correction

As EPC protocol adopts an Interrogator-Talks-First (ITF) procedure, the tag cannot actively reply the encoded error information back to the reader. Before transferring the error information, RFID reader has to first send a verification instruction so the boot-loader can generate the encoded error information. After that, reader acquires the information through *Read* command in Access round while is followed by the reply from the tag's side. In the first intuition, the *Read* can be used to transfer any customized data from the tag to the reader. However, due to the computational constraints and tight link timings of EPC protocol, the data size that could be successfully transferred from the tag to the reader is quite limited [62]. Therefore, for software-defined tags, the 16-bit encoded error information is replied during each during each *Read* command. For chip-based CRFID tags, the data size is restrained by the 8-bit SPI interface between the external MCU and the CRFID chip. Therefore, each time the *Read* command is received from the reader, the CRFID chip initializes the SPI communication and pulls 8 bits from MCU. For each piece of 16-bit encoded error information, these tags repeat the reply operation twice. To indicate the completion of the reply (encoded error information), the tag sends the end-of-signaling bits in which first 8 bits are initialized to "0"s.

After the transmission of continuous *Read* command to request for the error information, the reader decodes the received information following Algorithm 3 if the end-of-signaling bits are

ALGORITHM 3: Error Information Decoding (Executed on Reader Side)

```
Input: Error Information get from the tag's reply to a successful Read command ($(z_0, z_1, \ldots, z_{15})$), where $z_i$ is
       defined as the $i^{th}$ bit in the received error information, $i \in \{0, 1, \ldots, 15\}$.
Output: Vector of positions for erroneous segments ($\overline{Z}$)
while $\exists z_i \neq 0,\ (i \in 0, 1, \ldots, 7)$ do
    $offset = z_{15} \cdot 2^0 + z_{14} \cdot 2^1 + \cdots + z_8 \cdot 2^7$
    $\alpha_k = offset \times 8 + k \quad (k \in 0, 1, \ldots, 7)$
    $\overline{Z} = (z_0 \cdot \alpha_0, z_1 \cdot \alpha_1, \ldots, z_7 \cdot \alpha_7)$
end
```

Note: non-zero element in $\overline{Z}$ represents the sequence number of the erroneously reprogrammed segments.

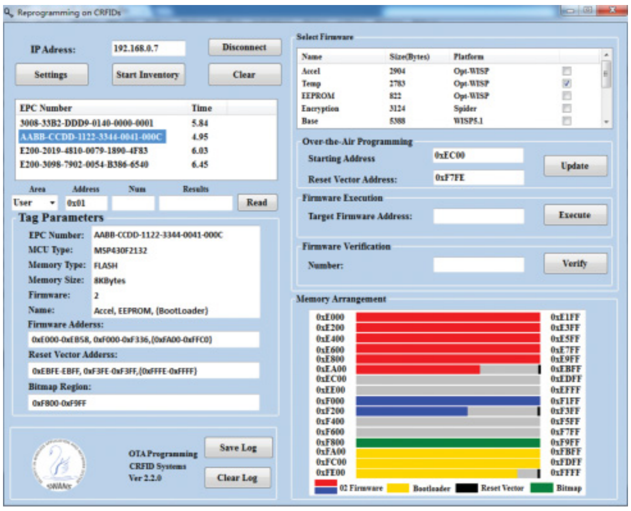
received. For each piece of 16-bit error information, the decoding result is an error position vector, which indicates the sequence numbers of the erroneous image segments. For example, if the 16-bit error information is *0x1402*, that is, $(z_0, z_1, \ldots, z_{15})$ = (0,0,0,1,0,1,0,0,0,0,0,0,0,0,1,0), the decoding result would be (0,0,0,19,0,21,0,0), which indicates reprogramming error in the 19^{th} and 21^{th} image segments. As the reprogramming errors only occur in the transition from "1" to "0", the erroneous firmware segments can be corrected without erasing the memory location in advance. Therefore, for software-defined CRFID tags, the reader can utilize reprogramming instruction (*MemBank*=01_b) to embed each image segment in the *Data* field and its memory location in the *WordPtr* field. For chip-based CRFIDs, the correction instruction to retransmit the image segments.

5 IMPLEMENTATION AND EVALUATION

We evaluate $\mathcal{R}^3$ on three platforms, that is, WISP5.1, Opt-WISP and Spider CRFID tags. The Opt-WISP is based upon the design of WISP4.1DL but it is retrofitted with two independent antennas and power harvesters. The first antenna-harvester pair is used for realizing receiving EPC commands, power harvesting and backscattering operation similar to WISP4.1DL. The second antenna-harvester pair is used exclusively for sensing applications. Such architecture results to higher output ranges and high-end sensing capabilities. Moreover, the MCU is fed with 3.3V instead of 1.8V (as in WISP4.1DL). The higher supply voltage is the prime reason that we use Opt-WISP in place of WISP4.1DL because the supplied voltage for WISP4.1DL is always much lower than the minimal writing requirement for MCU (2.2V). In case of Spider tag, the CRFID chip is interfaced with an evaluation board of MSP430F2132. This way, we evaluate $\mathcal{R}^3$ on MCUs with two different on-chip memories, FLASH in MSP430F2132 and FRAM in MSP430FR5969. A Graphical User Interface based on the software development kit of Impinj Speedway R420 is developed. To have a comprehensive evaluation of our system, we evaluate the overall system delay, energy overhead and the success rates at different interrogation ranges.

5.1 User Interface

As shown in Figure 10, the User Interface connects to a commercial reader and displays the information regarding the tags in the top left window. Once a specific tag is selected, its parameters are displayed in *Tag Parameters* panel. The memory addresses of the firmwares inside MCU memory are displayed towards the bottom of this panel in *Firmware Address* field. The memory address of boot-loader is displayed within braces to distinguish from other firmware images, for example, *{0xFA00-0xFFC0}*, which shows the starting and ending address of the boot-loader. To give a graphic illustration, we also portray the memory addresses of all existing firmwares in *Memory*



IP Adress: 192.168.0.7

EPC Number	Time
3008-33B2-DDD9-0140-0000-0001	5.84
AABB-CCDD-1122-3344-0041-000C	4.95
E200-2019-4810-0079-1890-4F83	6.03
E200-3098-7902-0054-B386-6540	6.45

Area	Address	Num	Results
User	0x01		

Tag Parameters

EPC Number: AABB-CCDD-1122-3344-0041-000C
MCU Type: MSP430F2132
Memory Type: FLASH
Memory Size: 8KBytes
Firmware: 2
Name: Accel, EEPROM, (BootLoader)
Firmware Adderss: 0xE000-0xEB58, 0xF000-0xF336,(0xFA00-0xFFC0)
Reset Vector Adderss: 0xEBFE-EBFF, 0xF3FE-0xF3FF,(0xFFFE-0xFFFF)
Bitmap Region: 0xF800-0xF9FF

OTA Programming
CRFID Systems
Ver 2.2.0

Select Firmware

Name	Size(Bytes)	Platform	
Accel	2904	Opt-WISP	☐
Temp	2783	Opt-WISP	✓
EEPROM	822	Opt-WISP	☐
Encryption	3124	Spider	☐
Base	5388	WISP5.1	☐

Over-the-Air Programming
Starting Address 0xEC00
Reset Vector Address: 0xF7FE

Firmware Execution
Target Firmware Address:

Firmware Verification
Number:

0xE000	0xE1FF
0xE200	0xE3FF
0xE400	0xE5FF
0xE600	0xE7FF
0xE800	0xE9FF
0xEA00	0xEBFF
0xEC00	0xEDFF
0xEE00	0xEFFF
0xF000	0xF1FF
0xF200	0xF3FF
0xF400	0xF5FF
0xF600	0xF7FF
0xF800	0xF9FF
0xFA00	0xFBFF
0xFC00	0xFDFF
0xFE00	0xFFFF

Fig. 10. User Interface passes encoded instructions through commercial RFID reader, receives the tag's reply and displays the results.

Arrangement panel. As shown in bottom right of the figure, the colored area marks the firmware images, reset vector is shown at the end of the memory segment while the blank area indicates the location of empty memory. Particularly, the green region indicates the area allocated for bitmap. To execute the firmware, the user can input the address of intended firmware in *Firmware Execution* panel.

For reprogramming operation, the user first needs to select the desired firmware in *Select Firmware* Panel. Then, the *Over-the-Air Programming* panel provides accessibility to specify the memory location for the selected firmware that comprising of *Starting Address* and *Reset Vector* fields. This way, a new firmware can be remotely transmitted to a specific memory location of the tag through *Write* Command. For firmware verification, the user inputs the number of segments that has been successfully transferred to tag, then uses the read command on the top of *Tag Parameters* panel to get the error information from the tag.

5.2 Overall System Delay

Since EPC protocol strictly limits the interrogation timings, the reader will lose the tag once these link timings are not followed because of computational overhead or inadequate energy towards the tag's end. Under such case, it becomes important to evaluate the time overhead incurred by $\mathcal{R}^3$ during reprogramming operation. The experimental setup is shown in Figure 11. We use an off-the-shelf Impinj Speedway R420 RFID reader with a 6 dBi circularly polarized antenna while the transmission power is set to 30 dBm. In this evaluation, we measure the overall time consumption for reprogramming a 512 bytes firmware image on Opt-WISP, WISP5.1 and Spider. The evaluation includes the time consumption incurred by uplink and downlink communication, memory reprogramming and error detection and correction. The evaluation is carried out once the reader and tag are spaced at 0.5 and 2 meters and the MCU clock for three CRFID tags is set to minimum and maximum frequencies, that is, 1 *MHz* and 16 *MHz*. At each interrogation range, the experiment is repeated for 50 times.

The experiment results are shown in Figure 12. For the interrogation range at 0.5 meter, the average time delay for Opt-WISP, WISP5.1 and Spider tags at 1 *MHz* is 15.21, 15.61 and 29.83 seconds

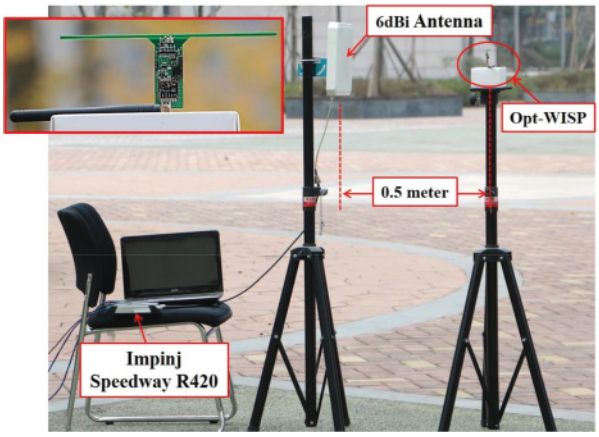
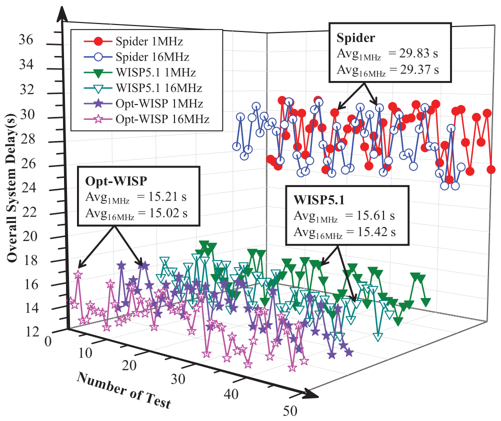
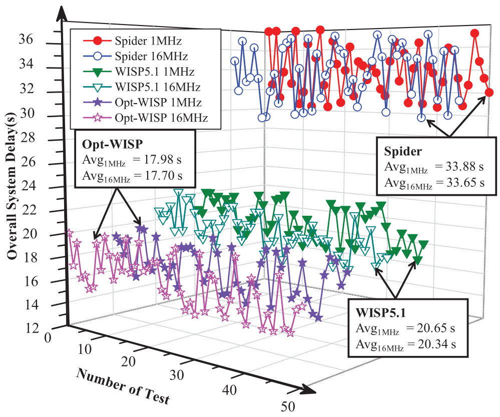


Fig. 11. Experimental setup for overall system delay and success rate evaluations.



(a) Reprogramming at 0.5 meters.

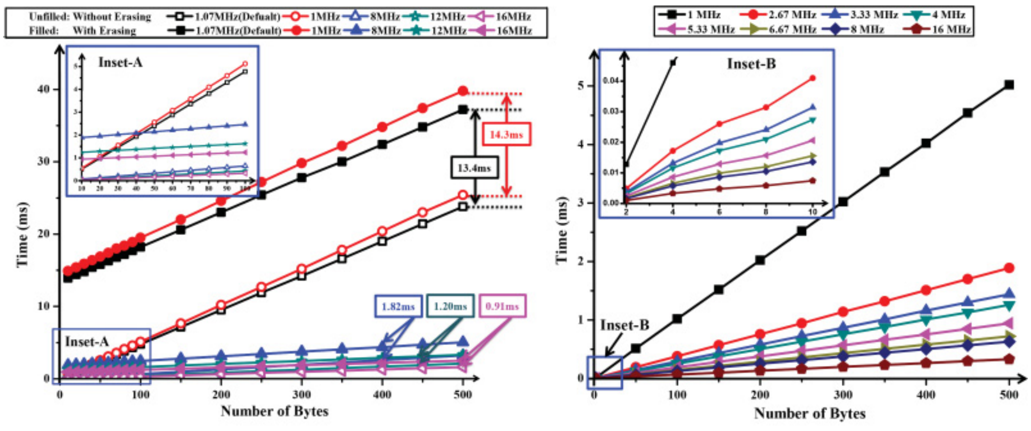


(b) Reprogramming at 2 meters.

Fig. 12. Overall system delay measurement. The distance between RFID reader antenna and CRFID tag is 0.5 and 2 meters, respectively. Each experiment is repeated for 50 times.

respectively, while at 16 *MHz* the results are 15.02, 15.42 and 29.37 seconds. For the distance of 2 meters, the average time at 1 *MHz* are 17.98, 20.65 and 33,88 seconds while at 16 *MHz* the time are 17.70, 20.34 and 33.65 seconds.

From the results, we observe that the overall time delay for Spider tag is roughly two times as that for the other two tags at both interrogation ranges. This is because only 8 bits can be transferred to MCU through SPI interface and the length of image segment is one byte for chip-based CRFID tags and two bytes for software-defined CRFID tags. As a result, the number of commands sent by a reader for spider is approximately doubled as compared with the other two tags. For WISP5.1 and Opt-WISP, they consume almost the same time because of their same instruction encoding and decoding mechanism. When the tag is placed at a distance of 2 meters, the inadequate harvested power results to higher reprogramming error rates. Therefore, it takes more time for the error detection and correction mechanism that is employed to correct the erroneously reprogrammed bits. More importantly, as the interrogation range increase, the overall round trip



(a) Writing data on MSP430F2132. (b) Writing data on MSP430FR5969.

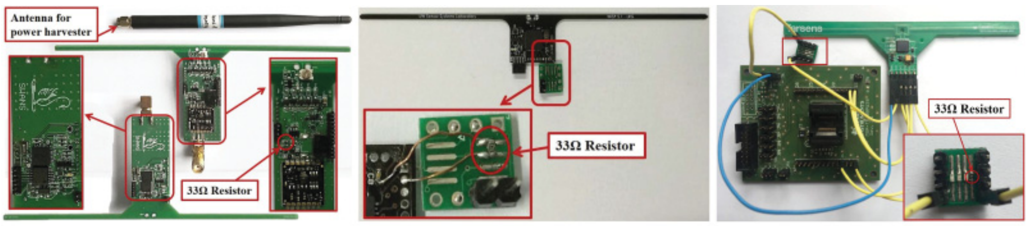
Fig. 13. Time consumption for writing data on MSP430F2132 and MSP430FR5969.

communication time increases because of the low communication success rates between RFID reader and CRFID tag. Comparing the results of Opt-WISP, WISP5.1 and Spider tags, we find that both the clock frequency and the type of memory have no serious effect on the overall system delay.

5.3 Energy Overhead

5.3.1 Time Delay Measurements. To investigate the energy overhead incurred by the reprogramming operation, we first need to measure the time consumption incurred by writing data to MCU. In this experiment, we study the time consumption of writing different bytes of data to the MCUs used in CRFID tags. For MSP430F2132, the time is measured at 5 different clock frequencies: default frequency (1.07 *MHz*), calibrated frequencies, which are 1, 8, 12, and 16 *MHz* under two conditions. One is writing the FLASH memory directly without erasing, another is writing the uninitialized FLASH memory after erasing. Such experiment design is to compare the time consumption of initializing and reprogramming instruction with correction instruction. Due to the nature of the FLASH memory cell, the state of each cell can be transformed from "1" to "0" individually but to reprogram from "0" to "1" requires an erase cycle. Therefore, the MCU needs to erase the FLASH segments before reprogramming. On the contrary, the correction instruction can directly rewrite the memory cell without erasing because the reprogramming error only occurs when the state is transiting from "1" to "0". As shown in Figure 13(a), the unfilled markers represent the time consumption of writing the memory without erasing the FLASH segment, while the filled markers show the time delay introduced by writing the memory after erasing the FLASH segment. In addition, the shown results pertain to writing a single memory segment. We observe a significant linear relationship between the time consumption and the number of bytes written to the memory. In particular, the time difference between writing with and without erasing operation is the distance between two parallel lines with the same color in Figure 13(a). Therefore, the time consumption for erasing operation (corresponding difference) at five clock frequencies is 13.4, 14.3, 1.82, 1.2 and 0.91 *ms* as annotated in the figure.

For MSP430FR5969, we evaluate our system on following clock frequencies: 1, 2.67, 3.33, 4, 5.33, 6.67, 8 and 16 *MHz*. However, the maximum write speed of FRAM on MSP430 devices is 8 *MHz* and a "wait state" is required if the CPU clock exceeds the FRAM access speed requirements.



(a) Opt-WISP (b) WISP5.1 (c) Spider

Fig. 14. Experimental setup for Opt-WISP, WISP5.1 and Spider.

The resultant speed depends upon factors like "cache hit ratio" and settings of NWAITSx register, the discussion of which is beyond the scope of our research. For evaluation at 16 *MHz*, we use the NWAITSx value of 01_b [47, 48]. The experiment results are shown in Figure 13(b). Similar to MSP430F2132, the time grows linearly as the number of bytes increases. However, when comparing the results for two MCU at the same system frequency, we can find that the time consumption for MSP430FR5969 is significantly lower than that for MSP430F2132.

5.3.2 Energy Overhead Measurements. To measure the energy overhead incurred by reprogramming operation, we use the similar method that is used to measure the power consumption of WISP4.1DL during encryption process [42]. We add a series resistor of 33 Ω in the power path of MCU in three CRFIDs and observe the voltage drop across the resistor. We find out that a value between 30-33 Ω serves our purpose: a high resistor value adequately drops the voltage below 1.8V, which is the minimum operating voltage for MCU, while a low resistance results in a voltage drop in least microvolts, which is difficult to measure with high resolution. The evaluation is performed for various clock frequencies. The insets in Figure 14 highlight the small modification to Spider tag and WISP5.1, whereas we put an on-board resistor in Opt-WISP. The voltage before and after the resistor (denoted as V_{in} and V_{out}) are measured using 8846A Digit Precision Multimeter, and the time consumption (denoted as t) is measured using a 5 GS/s oscilloscope. The power consumption is calculated by

$$Energy = \frac{(V_{in} - V_{out}) \cdot V_{out}}{R} \cdot t. \tag{1}$$

As the size of the firmware can scale and data is transferred through multiple *Write* commands for all three platforms, we evaluate the time and energy overhead of writing a single image segment to the on-chip memory. The energy overhead for Opt-WISP, Spider and WISP5.1 is shown in Table 2. We observe that despite the length of the image segment is 2 bytes for Opt-WISP and 1 byte for Spider, the energy consumption for the two tags is nearly similar. This is because Opt-WISP writes the on-chip memory in a word-wise fashion whereas Spider adopts the byte-wise writing, and the time difference between these two writing operations is negligible. In addition, for Opt-WISP and Spider the maximum energy overhead of writing an image segment without erasing is less than one microjoule. For WISP5.1 and Opt-WISP, the former consumes approximately an order of magnitude lower energy than latter. Among all three CRFID tags, the maximum time and energy consumption is 14.4ms and 118.639μJ for Spider at 1MHz, while the minimum values are 1.01μs and 7.264nJ for WISP5.1 at 16MHz. Moreover, we observe that the higher clock frequencies will result in lower energy consumption and otherwise.

Table 2. Time and Energy Measurements of Writing an Image Segment on Opt-WISP, WISP5.1, and Spider

	Clock (MHz)	V_{in} (V)	V_{out} (V)	Time		Energy	
				without Erasing	with Erasing	without Erasing	with Erasing
Opt-WISP (2 Bytes)	Default	3.346013	3.260247	113 μs	13.5 ms	957.481 nJ	114.389 μJ
	1	3.346012	3.263726	121 μs	14.4 ms	984.716 nJ	117.189 μJ
	8	3.346022	3.220431	15.4 μs	1.84 ms	188.747 nJ	22.551 μJ
	12	3.346017	3.194128	10.1 μs	1.21 ms	148.486 nJ	17.789 μJ
	16	3.346017	3.163778	7.68 μs	0.92 ms	134.182 nJ	16.074 μJ
Spider (1 Byte)	Default	3.163521	3.071369	113 μs	13.5 ms	969.173 nJ	115.786 μJ
	1	3.163518	3.075104	121 μs	14.4 ms	996.902 nJ	118.639 μJ
	8	3.163524	3.026743	15.4 μs	1.84 ms	193.201 nJ	23.084 μJ
	12	3.163516	2.998503	10.1 μs	1.21 ms	151.436 nJ	18.142 μJ
	16	3.163522	2.966066	7.68 μs	0.92 ms	136.301 nJ	16.328 μJ
WISP5.1 (2 Bytes)	1	2.216182	2.172403	12.8 μs		36.889 nJ	
	2.67	2.216181	2.163687	4.88 μs		16.796 nJ	
	3.33	2.216179	2.158987	3.76 μs		14.069 nJ	
	4	2.216183	2.155213	3.24 μs		12.901 nJ	
	5.33	2.216181	2.147463	2.48 μs		11.090 nJ	
	6.67	2.216174	2.131627	1.86 μs		10.158 nJ	
	8	2.216188	2.124673	1.62 μs		9.545 nJ	
	16	2.216185	2.103351	1.01 μs		7.264 nJ	

5.4 Success Rate

Since our scheme employs *Write* command to transmit the encoded instructions from RFID reader to CRFID tag. In the final experiment, we evaluate the success rates for the *Write* command on three platforms. The experiment setup is similar to Figure 11 as in case of overall system delay. The distance between the reader and CRFID tag is increased from 0.2 to 4.0m during the course of the experiment. Once the tag receives the *Write* command from the reader, it checks the CRC of the received packet. If a correct CRC is received, then a *Success* will be replied to the reader, otherwise, tag will ignore the command. Moreover, we compare our results with FirmSwitch [54], which uses WISP4.1DL and switches between multiple pre-programmed firmwares through *Write* command using the commercial RFID reader.

The evaluation results are shown in Figure 15. We observe that Spider tag has the best performance, since it is analogous to commercial RFID tags as far as the execution of EPC protocol is concerned, that is, whole EPC protocol is implemented in the embedded chip. Compared with Spider, the success rates for WISP5.1 and Opt-WISP are lower because the EPC protocol is realized in the MCU and the harvest power might not enough for the energy consuming computations when the interrogation range increases. In addition, we observe that Opt-WISP, WISP5.1, and Spider CRFIDs show better performance than FirmSwitch. There is a sharp decline in the success rate of FirmSwitch after 0.5 meters because WISP4.1DL backscatters sensor data as well as pre-calculates the CRC, which consumes a large amount of energy. As Opt-WISP makes use of an additional antenna-harvester pair exclusive for sensing/computational tasks, the interrogation range is increased and observed success rate is higher. In the case of WISP5.1, it uses an optimized and more sensitive power harvester that increases both the range as well as the success rate.

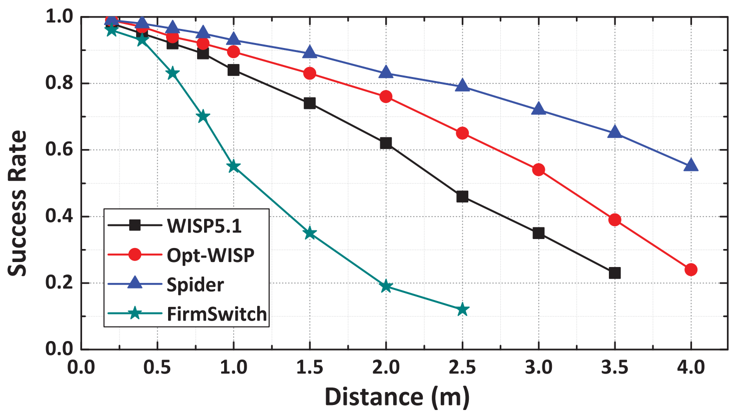


Fig. 15. Success rates at different interrogation ranges.

6 CONCLUSION

In this article, we present the $\mathcal{R}^3$ scheme, which can reliably perform an OTA reprogramming operation. The scheme is fully compatible with EPC protocol and does not require any hardware modifications to either CRFID tags or RFID reader. To this end, the system architecture is explained with an in-depth discussion for two topologies of CRFIDs, including three tags and two kinds of MCUs. Moreover, we alleviate the power failure though leveraging the transition between active mode and low-power mode. In particular, we investigate the underlying reasons for reprogramming errors and propose a simple yet efficient mechanism based on the concept of *Bitmap* for error detection and correction. We implement our scheme on two topologies of CRFIDs including three tags and two kinds of MCUs. Finally, we evaluate our scheme from three aspects: overall system delay in reprogramming firmware image, time and energy overhead for reprogramming operation, and success rate at different the reader's interrogation range. Experimental results show that the average overall system delay amounts to 29.83, 15.21, and 15.61s at 0.5m for Spider tag, Opt-WISP, and WISP5.1 CRFID tags when reprogramming 512 bytes at 1MHz. The energy consumption of writing a single image segment at 16MHz is 134.2, 136.3, and 7.2nJ for Opt-WISP, Spider, and WISP5.1, respectively. A success rate of 93%, 89.5%, and 84% is achieved for *Write* command with an interrogation range of 1m.

REFERENCES

[1] ARMmbed. 2015. Firmware over the air FOTA updates. (March 2015). Retrieved March 21, 2017. Retrieved from https://developer.mbed.org/teams/Bluetooth-Low-Energy/wiki/Firmware-Over-the-Air-FOTA-Updates.

[2] C. Bauer-Reich, Kay Chen Tan, F. Haring, N. Schneck, A. Wick, L. Berge, Jesse Hoey, Rudolf Sailer, and C. Ulven. 2014. An investigation of the viability of UHF RFID for subsurface soil sensors. In *Proceedings of the IEEE International Conference on Electro/Information Technology (EIT'14)*. IEEE, 577–580.

[3] Michael Buettner, Benjamin Greenstein, and David Wetherall. 2011. Dewdrop: An energy-aware runtime for computational RFID. In *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI'11)*.

[4] Michael Buettner, Richa Prasad, Matthai Philipose, and David Wetherall. 2009. Recognizing daily activities with RFID-based sensors. In *Proceedings of the 11th International Conference on Ubiquitous Computing*. ACM, 51–60.

[5] Michael Buettner and David Wetherall. 2011. A software radio-based UHF RFID reader for PHY/MAC experimentation. In *Proceedings of the IEEE International Conference on RFID (RFID'11)*. IEEE, 134–141.

[6] Rohit Chaudhri, Jonathan Lester, and Gaetano Borriello. 2008. An RFID based system for monitoring free weight exercises. In *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems*. ACM, 431–432.

[7] Adam Chlipala, Jonathan Hui, and Gilman Tolle. 2004. Deluge: Data dissemination for network reprogramming at scale. University of California, Berkeley, Technical Report (2004).

[8] Crossbow Technology, Inc. 2003. Mote In-network programming user reference (Jan. 2003).

[9] Artem Dementyev and Joshua R. Smith. 2013. A wearable UHF RFID-based EEG system. In *Proceedings of the IEEE International Conference onRadio Frequency Identificaion (RFID'13)*. IEEE, 1–7.

[10] Robert F. Dickerson, Eugenia I. Gorlin, and John A. Stankovic. 2011. Empath: A continuous remote emotional health monitoring system for depressive illness. In *Proceedings of the 2nd Conference on Wireless Health*. ACM, 5.

[11] Wan Du, Zhenjiang Li, Jansen Christian Liando, and Mo Li. 2016. From rateless to distanceless: Enabling sparse sensor network deployment in large areas. *IEEE/ACM Transactions on Networking* 24, 4 (2016), 2498–2511.

[12] Wan Du, Jansen Christian Liando, Huanle Zhang, and Mo Li. 2015. When pipelines meet fountain: Fast data dissemination in wireless sensor networks. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*. ACM, 365–378.

[13] EPCglobal. 2015. EPC Radio-Frequency Identity Protocols, Generation-2 UHF RFID, Specification for RFID Air Interface Protocol for Communications at 860MHz-960MHz, Version 2.0.1 Ratified. (April 2015). Retrieved May 29, 2016 from http://www.gs1.org/sites/default/files/docs/epc/Gen2_Protocol_Standard.pdf.

[14] Farsens. 2015. ANDY100 evaluation board with integrated start-up circuit. (Sept. 2015). Retrieved May 29, 2016 from http://www.farsens.com/media/document/26/ds-spider-h254-v01.pdf.

[15] Federico Gasco, Paolo Feraboli, Jeff Braun, Joshua Smith, Patrick Stickler, and Luciano DeOto. 2011. Wireless strain measurement for structural testing and health monitoring of carbon fiber composites. *Comp. Part A: Appl. Sci. Manufactur.* 42, 9 (2011), 1263–1274.

[16] Wei Gong, Kebin Liu, Xin Miao, and Haoxiang Liu. 2014. Arbitrarily accurate approximation scheme for large-scale RFID cardinality estimation. In *Proceedings of IEEE INFOCOM*. IEEE, 477–485.

[17] Jeremy Gummeson, Shane S. Clark, Kevin Fu, and Deepak Ganesan. 2010. On the limits of effective hybrid microenergy harvesting on mobile CRFID sensors. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*. ACM, 195–208.

[18] Jeremy Gummeson, Pengyu Zhang, and Deepak Ganesan. 2012. Flit: A bulk transmission protocol for RFID-scale sensors. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*. ACM, 71–84.

[19] Daniel Halperin, Thomas S. Heydt-Benjamin, Benjamin Ransford, Shane S. Clark, Benessa Defend, Will Morgan, Kevin Fu, Tadayoshi Kohno, and William H. Maisel. 2008. Pacemakers and implantable cardiac defibrillators: Software radio attacks and zero-power defenses. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P'08)*. IEEE, 129–142.

[20] Shibo He, Jiming Chen, Fachang Jiang, David K. Y. Yau, Guoliang Xing, and Youxian Sun. 2013. Energy provisioning in wireless rechargeable sensor networks. *IEEE Trans. Mobile Comput.* 12, 10 (2013), 1931–1942.

[21] Enamul Hoque, Robert F. Dickerson, and John A. Stankovic. 2010. Monitoring body positions and movements during sleep using WISPs. In *Proceedings of the 2010 Conference on Wireless Health*. ACM, 44–53.

[22] Impinj. 2015. Octane SDK. (Oct. 2015). Retrieved May 29, 2016 from https://support.impinj.com/hc/en-us/articles/202755268-Octane-SDK.

[23] Shan Jiang and Stavros V. Georgakopoulos. 2011. Optimum wireless power transmission through reinforced concrete structure. In *Proceedings of the IEEE International Conference on Radio Frequency Identification (RFID'11)*. IEEE, 50–56.

[24] Bryce Kellogg, Aaron Parks, Shyamnath Gollakota, Joshua R. Smith, and David Wetherall. 2015. Wi-Fi backscatter: Internet connectivity for RF-powered devices. *ACM SIGCOMM Comput. Commun. Rev.* 44, 4 (2015), 607–618.

[25] Libelium. 2016. Over the Air Programming (OTAP). (Jan. 2016). Retrieved May 29, 2016 from http://www.libelium.com/products/waspmote/ota/.

[26] Vincent Liu, Aaron Parks, Vamsi Talla, Shyamnath Gollakota, David Wetherall, and Joshua R. Smith. 2013. Ambient backscatter: Wireless communication out of thin air. *ACM SIGCOMM Comput. Commun. Rev.* 43, 4 (2013), 39–50.

[27] Brandon Lucia and Benjamin Ransford. 2015. A simpler, safer programming and execution model for intermittent systems. In *ACM SIGPLAN Notices*, Vol. 50. ACM, 575–585.

[28] Pedro José Marrón, Andreas Lachenmann, Daniel Minder, Matthias Gauger, Olga Saukh, and Kurt Rothermel. 2005. Management and configuration issues for sensor networks. *Int. J. Network Manage.* 15, 4 (2005), 235–253.

[29] Saman Naderiparizi, Aaron N. Parks, Zerina Kapetanovic, Benjamin Ransford, and Joshua R. Smith. 2015. Wispcam: A battery-free RFID camera. In *Proceedings of the IEEE International Conference on Radio Frequency Identification (RFID'15)*. IEEE, 166–173.

[30] Brian Otis and Dan Yeager. 2009. SoCWISP: Ultra-low power wireless sensing RFID chip. In *Proceedings of the WISP Summit Workshop*.

[31] Aaron N. Parks, Angli Liu, Shyamnath Gollakota, and Joshua R. Smith. 2014. Turbocharging ambient backscatter communication. In *ACM SIGCOMM Comput. Commun. Rev.* Vol. 44. ACM, 619–630.

[32] Aaron N. Parks and Joshua R. Smith. 2014. Sifting through the airwaves: Efficient and scalable multiband RF harvesting. In *Proceedings of the IEEE International Conference on Radio Frequency Identification (RFID'14).* IEEE, 74–81.

[33] Matthai Philipose, Joshua R. Smith, Bing Jiang, Alexander Mamishev, Sumit Roy, and Kishore Sundara-Rajan. 2005. Battery-free wireless identification and sensing. *IEEE Pervas. Comput.* 4, 1 (2005), 37–45.

[34] Lane A. Phillips. 2005. *Aqueduct: Robust and Efficient Code Propagation in Heterogeneous Wireless Sensor Networks.* Ph.D. Dissertation. University of Colorado.

[35] Benjamin Ransford. 2010. A rudimentary bootloader for computational RFIDs. *UMass Amherst, Technical Report UM-CS-2010-061* (2010).

[36] Benjamin Ransford, Shane S. Clark, Mastooreh Salajegheh, and Kevin Fu. 2008. Getting things done on computational RFIDs with energy-aware checkpointing and voltage-aware scheduling. *HotPower* 8 (2008), 5–5.

[37] Benjamin Ransford, Jacob Sorber, and Kevin Fu. 2012. Mementos: System support for long-running computation on RFID-scale devices. *ACM SIGPLAN Notices* 47, 4 (2012), 159–170.

[38] Niels Reijers and Koen Langendoen. 2003. Efficient code distribution in wireless sensor networks. In *Proceedings of the 2nd ACM International Conference on Wireless Sensor Networks and Applications.* ACM, 60–67.

[39] Matt Reynolds and Stewart Thomas. 2009. The blue devil WISP: Expanding the frontiers of the passive RFID physical layer. In *Proceedings of the WISP Summit Workshop.*

[40] Mastooreh Salajegheh, Yue Wang, Anxiao Andrew Jiang, Erik Learned-Miller, and Kevin Fu. 2013. Half-wits: Software techniques for low-voltage probabilistic storage on microcontrollers with NOR flash memory. *ACM Trans. Embed. Comput. Syst. (TECS)* 12, 2s (2013), 91.

[41] Yuanchao Shu, Yu Jason Gu, and Jiming Chen. 2014. Dynamic authentication with sensory information for the access control systems. *IEEE Trans. Parallel Distrib. Syst.* 25, 2 (2014), 427–436.

[42] Joshua R. Smith. 2013. *Wirelessly Powered Sensor Networks and Computational RFID.* Springer Science & Business Media.

[43] Thanos Stathopoulos, John Heidemann, and Deborah Estrin. 2003. *A Remote Code Update Mechanism for Wireless Sensor Networks.* Technical Report. DTIC Document.

[44] Jethro Tan. 2015. *Robust Downstream Communication and Storage for Computational RFIDs.* Ph.D. Dissertation. Department of Software Technology, Delft University of Technology.

[45] Texas Instruments. 2012. Mixed Signal Microcontroller. (Jan. 2012). Retrieved May 29, 2016 from http://www.ti.com/lit/ds/symlink/msp430f2132.pdf.

[46] Texas Instruments. 2013. MSP430x2xx Family User's Guide. (July 2013). Retrieved May 29, 2016 from http://www.ti.com/lit/ug/slau144j/slau144j.pdf.

[47] Texas Instruments. 2015. MSP430FR59xx Mixed-Signal Microcontrollers. (March 2015). Retrieved May 29, 2016 from http://www.ti.com/lit/ds/symlink/msp430fr5969.pdf.

[48] Texas Instruments. 2016. MSP430FR58xx, MSP430FR59xx,MSP430FR68xx, and MSP430FR69xx Family User's Guide. (May 2016). Retrieved May 29, 2016 from http://www.ti.com/lit/ug/slau367j/slau367j.pdf.

[49] Arnaud Vena, Brice Sorli, Alain Foucaran, and Yassin Belaizi. 2014. A RFID-enabled sensor platform for pervasive monitoring. In *Proceedings of the 9th International Symposium on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC'14).* IEEE, 1–4.

[50] Jue Wang, Haitham Hassanieh, Dina Katabi, and Piotr Indyk. 2012. Efficient and reliable low-power backscatter networks. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication.* ACM, 61–72.

[51] Jennifer Wang, Erik Schluntz, Brian Otis, and Travis Deyle. 2015. A new vision for smart objects and the internet of things: Mobile robots and long-range UHF RFID sensor tags. *arXiv:1507.02373* (2015).

[52] Die Wu, Muhammad Jawad Hussain, Songfan Li, and Li Lu. 2016. R2: Over-the-air reprogramming on computational RFIDs. In *Proceedings of the IEEE International Conference on Radio Frequency Identification (RFID'16).* IEEE, 1–8.

[53] Zhibin Xiao, Xi Tan, Xianliang Chen, Sizheng Chen, Zijian Zhang, Hualei Zhang, Junyu Wang, Yue Huang, Peng Zhang, Lirong Zheng, and Hao Min. 2015. An implantable RFID sensor tag toward continuous glucose monitoring. *IEEE J. Biomed. Health Info.* 19, 3 (2015), 910–919.

[54] Wenyu Yang, Die Wu, Muhammad Jawad Hussain, and Li Lu. 2015. Wireless firmware execution control in computational RFID systems. In *Proceedings of the IEEE International Conference on Radio Frequency Identification (RFID'15).* IEEE, 129–136.

[55] Daniel Yeager, Fan Zhang, Azin Zarrasvand, Nicole T. George, Thomas Daniel, and Brian P. Otis. 2010. A 9 uA, addressable gen2 sensor tag for biosignal acquisition. *IEEE J. Solid-State Circ.* 45, 10 (2010), 2198–2209.

[56] Ibon Zalbide, Eduardo D'Entremont, Ainara Jimenez, Hector Solar, Andoni Beriain, and Roc Berenguer. 2014. Battery-free wireless sensors for industrial applications based on UHF RFID technology. In *Proceedings of the IEEE 2014 Conference on Sensors.* IEEE, 1499–1502.

[57] Hong Zhang, Jeremy Gummeson, Benjamin Ransford, and Kevin Fu. 2011. Moo: A batteryless computational RFID and sensing platform. Department of Computer Science, University of Massachusetts Amherst, Technical Report (2011).

[58] Pengyu Zhang and Deepak Ganesan. 2014. Enabling bit-by-bit backscatter communication in severe energy harvesting environments. In *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI'14)*. 345–357.

[59] Pengyu Zhang, Jeremy Gummeson, and Deepak Ganesan. 2012. Blink: A high throughput link layer for backscatter communication. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*. ACM, 99–112.

[60] Yuanqing Zheng and Mo Li. 2013. ZOE: Fast cardinality estimation for large-scale RFID systems. In *Proceedings of the 2013 IEEE INFOCOM*. IEEE, 908–916.

[61] Yuanqing Zheng and Mo Li. 2014. Towards more efficient cardinality estimation for large-scale RFID systems. *IEEE/ACM Trans. Network.* 22, 6 (2014), 1886–1896.

[62] Yuanqing Zheng and Mo Li. 2016. Read bulk data from computational RFIDs. *IEEE/ACM Trans. Network.* 24, 5 (2016), 3098–3108.

Received June 2016; revised December 2016; accepted March 2017