Wireless Firmware Execution Control in Computational RFID Systems

Wenyu Yang*, Die Wu*, Muhammad Jawad Hussain*†, and Li Lu*
*School of Computer Science and Engineering, University of Electronic Science and Technology of China
†School of Communication and Information Engineering, University of Electronic Science and Technology of China

***Abstract*—Current Computational RFID Tags (CRFIDs) are pre-programmed with only a single firmware instance in their flash memories for runtime operation where the functionality of CRFID is pre-determined by the firmware at the time of programming. As a result, the current CRFIDs require wired interface to re-program a new firmware which strictly limits their use to easy-to-reach places. We address this issue by remotely changing the behavior of CRFIDs by switching their firmware through commercial RFID reader and the EPC protocol, without demanding any hardware upgrades to CRFID tags or modification to EPC standard.**

We articulate the design, implementation and evaluation of FirmSwitch - a wireless scheme that equips CRFIDs with the capability of switching their firmware during runtime. This is achieved by wirelessly passing the encoded switching parameters to CRFID tag through RFID reader which leverages the tag to switch among firmwares and execute them for intended cycles. We further employ the schemes of *pre-defined EPC* and *pre-calculated CRC* for computational liberty and energy efficiency. For concept validation, we develop a User Interface to switch between four firmwares and extensively test our scheme. The results show that FirmSwitch offers a minimal energy overhead of 11.5nJ to 2.037μJ, and incurs a switching delay of 7.8 to 1498 μsec. As overall, our system achieves a success rate of 87% for an interrogation range of 0.5 meter.

***Index Terms*—Computational RFID; Execution Flexibility; Firmware Execution; EPC.**

I. Introduction

Computational RFID (CRFID) is an emerging technology which not only extends the capabilities of RFID tags for enhanced operations like sensing and computation [1]–[3], but in parallel, opens new opportunities for Internet-of-Things and ubiquitous computing [4]–[6]. The CRFID tag is a PCB based far-field passive RFID tag comprising Microcontroller Unit (MCU), EEPROM, sensors and discrete architectures for transmitter, receiver, power harvester and else. Based on WISP as the pioneering CRFID project [7], many other variants have been developed on the same design [6], [8]–[11].

We observe that present versions of CRFID are programmed with a single firmware which greatly restricts their functionality to a single and specific operation during runtime. In real scenarios, the CRFIDs maybe required to execute multiple operations, such as sensor polling and data collection in one firmware and encryption in another firmware. For such cases, any firmware update requires the MCU in the tag to be physically connected (through programming wires) to Flash Programmer, which often becomes infeasible. Like, in [12], the UMASS-MOO CRFID is buried in concrete to measure strain and temperature.

In order to leverage CRFID tags with multiple functionalities, a straightforward way is to pre-program the tag with a single firmware that contains the source codes of all required operations (each individual firmware). This may not be possible for many reasons. Like, each individual firmware may have distinct logic and architecture that can present problems in integration of diverse codes; combining the codes may increase the size; developers may be unwilling to open their source codes. We present the viability of a multi-firmware approach, named FirmSwitch, that intelligently combines various firmwares for diverse operations. Furthermore, we consider our work to be an elementary step towards "over-the-air (OTA) programming" concept. Like, if a CRFID is equipped with higher voltage ($\geq 2.2V$ in Intel-WISP V4.1) to re-program the MCU flash, then our work can be extended to wirelessly receive, load and execute new firmwares without any wired access which we will further elaborate in Section VII. We highlight that the term "firmware" in our discussion refers to the code that can realize hardware initialization, EPC protocol communication, sensor polling and execute specific function within the resources of CRFID tags.

In essence, our endeavour can be visualized as a subset to "OTA programming" whereby we employ the commercial RFID reader to remotely change the behaviour of a CRFID tag through EPC protocol without requiring physical access or hardware upgrades to CRFID tags. Initially, the user stores all intended firmwares inside MCU flash during the deployment phase and selects the most useable firmware as a default firmware. Upon adequate harvested power, the CRFID tag starts the execution of default firmware. Upon completion, the user can select and switch the operation of CRFID to a particular firmware by instructing the tag through a commercial RFID reader and EPC protocol. Such selection is realized by passing the switching parameters (like firmware ID) through *Write* Access command in EPC protocol. To make our system more flexible, the user can also specify the number of firmware executions and next intended firmware through a "switch command" sent from the reader. The whole process is regulated under energy supervisor whereby system moves between sleep and active modes upon power outage and resumes the execution from the last point.

More in detail, we realize FirmSwitch in shape of *Firmware Arrangement* at deployment phase, *Instruction Encoding* in

EPC protocol and *Firmware Loading and Execution* for the tag. While deploying the CRFID tag, the Firmware Arrangement step appends each and every individual firmware with a Decoding Module and compresses them to a single firmware by means of public code, callback routines and a look-up table. The finally compiled firmware code is downloaded to flash memory through wired interface. To pass switching parameters to the tag, the *Instruction Encoding* process encodes the switching instructions in EPC $Write$ command which contains the memory address of the target firmware and its number of executions. On the tag side, the *Firmware Loading and Execution* module extracts and decodes these parameters and switches the firmwares and executes them for required cycles.

A quick glance of faced challenges and their realized solutions is given herein: 1) The $Success$ response from the tag is transferred in $Write$ command requiring 16-bits of CRC, which a CRFID cannot compute in runtime [13]. As a solution, we use pre-computed CRC as in [14]. 2) Tag-ID (EPC number) changes each time the data is sent back (as legacy CRFIDs embed their sensor data within 96-bits of EPC field to conserve power). In result, the reader foresees each EPC field as a new tag ID. We resolve this issue by using pre-defined EPC which also indicates the user about the start of the switching operation. 3) Tag should not be left in unstable state, e.g., PC pointer points to wrong memory address. We use energy supervisor and memory pointers to put system to sleep mode and to restart properly.

We demonstrate our scheme implementing a prototype on Intel-WISP and realize a User Interface for commercial RFID reader. At system level, we test FirmSwitch for switching among Accelerometer firmware, Temperature Sensor firmware and a firmware to response EPC number. We further evaluate our scheme against four different firmwares to validate its efficiency for interrogation range, success rate, execution time and energy overhead. The results show that FirmSwitch provides a minimal energy overhead of 11.5 nJ to 2.037 μJ and incurs switching delay of 7.8 to 1498 μsec. Moreover, while switching amongst firmwares and corresponding with the reader in runtime, our system achieves a success rate of 87% for 0.5 meter.

We believe that FirmSwitch can enormously reduce the deployment and maintenance overheads of CRFID systems. Following are the chief features:

- FirmSwitch is a wireless firmware switching approach for CRFID tokens without requiring any hardware upgrades. The system conforms to EPC-C1G2 standard and can be realized through commercial RFID reader.
- It equips CRFID tags with firmware flexibility whereby the user can load several firmwares in tag's flash memory during deployment phase which can be switched during system runtime.
- It is light-weight in terms of computation, communication time and power consumption.
- System starts execution with a default firmware while resumes its operation from last execution point during duty cycle mode.

Admittedly, the chief limitation of FirmSwitch is the number of firmwares that can be stored in flash memory, which we emphasize is the limiting factor because of the size of flash memory. Our scheme can be scaled to more number of firmwares if UMASS-MOO CRFID is used with 116 Kbyte of flash. In essence, our system is directly scalable to higher onboard flash memories. As a future extension, we foresee to implement the complete over-the-air programming, i.e., a user can re-write the onboard flash with new firmware through RFID reader. Since, current CRFIDs operate on 1.8V which is inadequate to program the MCU flash, therefore, we envision two viable modifications to CRFID tag: it can be the re-design of harvester topology and power regulation to offer higher voltage, or, the change in MCU which can re-program the flash at low voltages (like MSP430FR5969 [15]).

The remainder of this paper is structured as followed: Section II comprehends various firmware approaches for CRFID systems. The design challenges are explained in Section III while the FirmSwitch is elaborated in Section IV. Section V illustrates the system implementation and evaluation on Intel-WISP while the evaluation metrics and the results are discussed in Section VI. The paper concludes at Section VII.

II. Related Work

To best of our knowledge, FirmSwitch is the first approach to offer firmware flexibility for CRFID tags through wireless medium using RFID reader and EPC protocol. We only find two relevant works on MSP430 series of microcontrollers that tend to manage the firmware execution. In [13], the authors presented an energy aware schedular which maps the harvested voltage with appropriate firmware to execute. The selection of new firmware is based upon the extent of energy it utilizes. However, the said scheme is restricted only to energy aware scheduling and user does not have flexibility to select a specific firmware as per the requirement. The authors in [16] presented a pre-programmed look-up table schedular. The MCU follows a pre-defined look-up table to execute the tasks in a cycle which is restricted to the scheduling of look-up table only.

Various firmware approaches have been presented for CRFIDs as far as their firmware execution, energy-aware task scheduling and other functionalities are concerned. The Memento [13] enables the CRFID tag to complete the long-running computations by breaking a single firmware into interruptible executions. The Harmony [14] pre-calculates the CRC to save harvested energy and the execution time. In [17], [18], the authors proposed a scheme to enhance the security of proximity cards with CRFID enabled secret handshakes. Several techniques are presented in [14], [19]–[21] which allow efficient storage and secure data transfer between a CRFID tag and the commercial reader.

III. Design Challenges

We faced three challenges during implementation of FirmSwitch. First two challenges relate to Inventory and Access Rounds of EPC protocol for using the "empty" tag-ID field

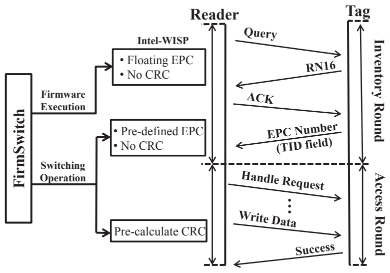


Figure 1. FirmSwitch follows the scheme of *Floating EPC* during routine firmware execution for *N* times, which is in analogy to legacy Intel-WISP CRFID. For switching operation, it incorporates *Pre-defined EPC* during Inventory Round and *Pre-calculated CRC* during Access Round.

and calculating the CRC value. The third challenge relates to preventing the tag to move into unstable state.

In Fig. 1, we illustrate the Inventory and Access Rounds of EPC standard [22]. The current CRFIDs embed their sensor data inside 96-bits of EPC number (TID field) during Inventory Round to backscatter the information. This, in result, changes the TID during each operation which we term as *Floating EPC*. For FirmSwitch, we have two scenarios. In first case, it executes a firmware for N times and backscatters the data during Inventory Round just like current CRFIDs. In here, we adopt the same procedure of *Floating EPC* and embed the data in TID field. In second case towards the successful execution of a firmware, FirmSwitch has to read the switching parameters during Access Round. This requires the execution of Inventory Round before the Access Round which involves the TID reply. But the TID field is empty as there is no sensor data. In resolution, we use a pre-defined number as the tag-ID in "empty" TID field which we term as *Pre-defined EPC*. Such an architecture of floating and pre-defined EPC greatly simplifies the understanding and debugging of FirmSwitch, as illustrated in Section V-A.

Next, once the tag has successfully executed the firmware operation and received new switching parameters, it needs to reply a $Success$ message which requires a CRC value. However, we observe that current CRFIDs cannot calculate CRC within the strict time slots prescribed by the EPC protocol because of low computational capability and power constraints [14]. To this end, we pre-compute CRC values in line with [14] and resultantly, FirmSwitch can backscatter $Success$ reply during system runtime.

The third challenge relates to preventing the tag to go into unstable state once system moves from low power modes to active mode. We use memory pointers (Program Counter Pointer and Reset Vectors) for proper firmware switching, execution and restart operation. Moreover, following the approach of current CRFIDs, we use energy supervisor to move our system between active and power down states.

IV. FirmSwitch Design Approach

FirmSwitch consists of three steps: *Firmware Arrangement*, *Instruction Encoding*, and *Decoding and Execution*. Firmware Arrangement step efficiently integrates and compresses the individual firmwares to a single firmware and places them in specific memory segments. Moreover, each firmware is also appended with a Decoding Module. It is the key module which actually performs the switching operation, as shown in Fig. 2 and later explained in Section IV-C. The Instruction Encoding process embeds the switching parameters inside *Write* command of EPC protocol. The Decoding and Execution process parses the switching parameters passed by the reader and selects new firmware for intended operations.

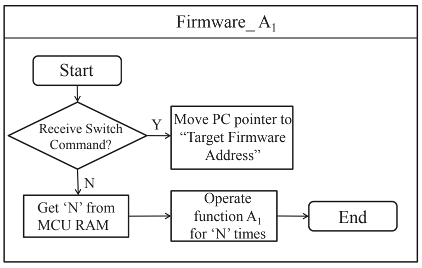

Firmware_ A1
Start
Receive Switch Command?
Y
Move PC pointer to "Target Firmware Address"
N
Get 'N' from MCU RAM
Operate function A1 for 'N' times
End

Figure 2. Decoding Module is the key block in FirmSwitch. It parses the switching instructions and either executes the same firmware for specified operations, or, selects and executes the next firmware for intended cycles.

A. Firmware Arrangement

The prime function of Firmware Arrangement step is to embed multiple firmwares in space-limited flash memory during deployment phase. First, each firmware is appended with a Decoding Module of 210 bytes. Next, all firmwares are *compressed* by inspecting and segregating the common routines as "public code" which are called back using a look-up table. The finalized firmware is saved in flash memory as a single firmware. Both steps are explained below.

1) Firmware Compression: Once we compile each individual firmware to its binary code, we observe some duplicate parts which can be taken out and defined as "public code". For example, the size of Accelerometer and Temperature Sensor firmwares is 2.90 and 2.78 Kbyte while both firmwares include 1.1 Kbyte of EPC protocol and 210 bytes of Decoding Module. We first extract these two common codes and save them at a memory location, say 0xA5F0. Next, we "refer" this public code in Accelerometer and Temperature Sensor firmwares by moving the PC pointer to 0xA5F0 through look-up table. The look-up table stores the callback routines of each firmware and the public codes. Upon successful completion, the execution is returned back to original firmware.

2) Firmware Storage in Memory Segments: Intel-WISP and its most variants are embedded with MSP430F2132 MCU with 8 Kbyte flash memory. This is divided into segments of 512 bytes whereby each segment is also the smallest unit of data erasure. In firmware storage step, we assign memory segments to each firmware and record its address. We stipulate that every firmware should be placed at the beginning of a memory segment. The main aim of such placement is to have the *Firmware Flexibility*. This relates to our future work which aims to re-write the firmware through EPC protocol, like, we want to erase firmware-A and add a new firmware-D. Following our placement scheme, we only need to erase the memory segments of firmware-A and rewrite firmware-D (provided latter is smaller in size). Without the use of memory segmentation, we need to update the complete flash memory to

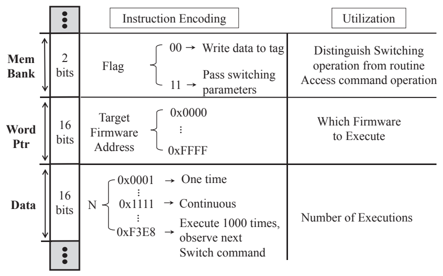


Figure 3. Instruction Encoding in the $Write$ Command.

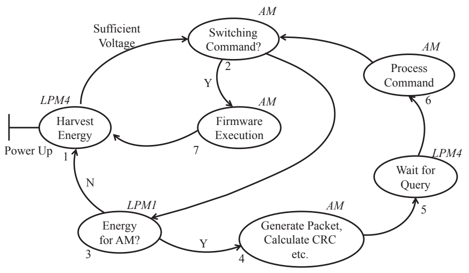


Figure 4. Illustration of FirmSwitch transitioning between Active Mode and various Low Power Modes.

replace even one small firmware which we envision to increase the update overhead in terms of time and energy.

B. Instruction Encoding on Reader Side

In EPC protocol, the RFID reader can read or write the data, lock and disable the tag using Access commands. We use $Write$ Access command to pass our switching instructions to the tag. We embed our data in three fields of $Write$ command: 2 bits of *MemBank*, 16 bits of *WordPtr*, and 16 bits of *Data* field. Fig. 3 gives an overview of *MemBank*, *WordPtr* and *Data* fields along with their utilization for Instruction Encoding process in FirmSwitch.

The *Instruction Encoding* step aims to embed and encode the switching parameters in the $Write$ Access command. The switching parameters include the $Flag$, N and $firmware$ $address$. The $Flag$ is used to distinguish the "switch" command from routine commands of EPC protocol, N illustrates the number of executions and $firmware\ address$ indicates the memory address of target firmware.

Unlike RFID tags, CRFIDs do not have reserved memories for EPC, TID, and User Memory, and therefore, we can use two bits of *MemBank* as a $Flag$ to distinguish between switching operation (coded as 11_b) and routine Access command operation (00_b), as illustrated in Fig. 3. We encode 16 bits of *WordPtr* to pass the memory address of the target firmware which we want to switch (Section IV-A2). Similarly, the 16-bits of *Data* field are used for encoding the number of firmware executions. In our evaluation, we encode these 16-bits as 0x0001 for one time, 0x1111 for continuous and 0xF3E8 for 1000 time execution followed by the termination of firmware.

C. Decoding and Execution on Tag Side

As shown in Fig. 2, the Decoding and Execution process decodes the switching instructions passed through *Write* command and in result, executes the same firmware for specified cycles or selects the new firmware for specific number of executions.

The job of $Decoding$ process is to parse the switching parameters ($Flag$, N and $firmware\ address$) and save them in MCU RAM. It first implements the EPC-C1G2 standard till Access Round and interprets the value of $Flag$ from $Write$ command to check whether the user wants to use the Access command in a routine way ($Flag = 00_b$) or pass the switching parameters ($Flag = 11_b$). For $Flag = 00_b$, FirmSwitch continues to run EPC protocol. For $Flag = 11_b$, FirmSwitch stores the value of N and $firmware\ address$ in MCU RAM.

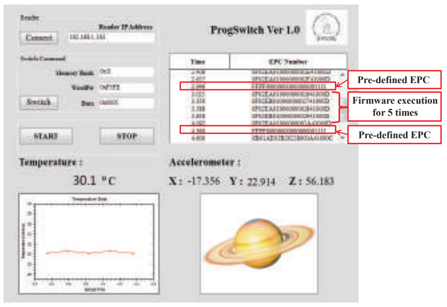
The $Execution$ process reads the variables N and $firmware\ address$ from system RAM and moves the PC pointer to target firmware reset vector which would initialize the firmware execution. Moreover, the execution process keeps the track of number of successful executions and updates the parameter N to N-1 in the RAM upon each firmware execution. For N successful executions, the tag sends a $Success$ acknowledgement to the reader. If the system runs out of power and goes into sleep mode (LPM4 mode which retains the RAM contents), the $Execution$ module learns the parameters N and $firmware\ address$ from the RAM once harvested power is adequate, and resumes its execution from the last point.

We highlight that our system uses pre-defined EPC and pre-calculated CRC, and operates upon various power saving modes upon inadequate harvested power, as explained next.

1) Pre-defined EPC and Pre-calculated CRC: During routine firmware execution, FirmSwitch uses *Floating EPC* to backscatter the sensor data, as explained in Section III. Our scheme differs once the reader wants to pass the switching parameters through $Write$ Access command. In here, the FirmSwitch embeds the *Pre-defined EPC* in "empty" TID field to execute the Inventory Round and moves to Access Round to read the $Write$ instruction.

During Access Round, FirmSwitch is required to compute the CRC value to reply a $Success$ message. If tag fails to reply $Success$, the reader will consider it as a failure. However, because of low computational capability and power constraints, the current CRFIDs cannot calculate CRC value during system runtime [14]. In resolution, we employ the method of pre-computing CRC as introduced in [14]. Following this approach, whenever the CRFID tag is powered up, the FirmSwitch will first check whether all CRC values have been pre-calculated or not. If not, it will pre-compute CRC values and store them in one list which will be used by the tag to reply the $Success$ message to the reader.

2) Transition between Active and Power saving Modes: Our system executes between Active and Power Saving modes for energy conservation (Fig. 4). The MSP430F2132 microcontroller used in our evaluation has two types of operating modes: Active mode (AM) and software selectable Low-Power Mode (LPM). The low power mode is further categorized to five types, LPM0 to LPM4, which offer various functionalities. We use AM, LPM1 and LPM4 modes. In brief, only the CPU is stopped in LPM1 mode while only the RAM is retained in



Reader

Reader IP Address: [illegible]

Switch Command

Memory Bank: 0x3

WordPtr: 0xFFE

Data: 0x0005

Time	EPC Number
[illegible]	[illegible]

Temperature : 30.1 °C

Accelerometer : X : -17.356 Y : 22.914 Z : 56.183

Figure 5. User Interface passes switching parameters to CRFID tag through RFID reader, receives the reply and displays the results.

LPM4 mode. In FirmSwitch, the MCU moves between active and low power modes depending upon interrupts generated by the energy supervisor.

The execution starts from LPM4 (Step-1) from power-up. When sufficient power is harvested and energy supervisor generates an interrupt, the system enters into Active Mode (Step-2) and decoding module checks the RAM contents for any previous process (N and $firmware\ address$). If RAM does not contain any contents, the MCU moves through Step-3 to Step-4 to pre-calculate CRC. Next, the system remains in LPM4 mode at Step-5 and waits for a hardware trigger to indicate the Query command from the reader. Upon receiving the Query and having adequate power, the system executes the EPC protocol in Active Mode during Step-6 to process the $Write$ command, and then moves to Step-2.

Once switching instructions are received, the system moves from Step-2 to Step-7 (both Active modes) to execute the target firmware. During N executions, the tag backscatters the sensor data. Upon N successful executions, system returns a $Success$ message and moves to Step-1. Once harvested power becomes inadequate during firmware execution in Step-7, the system moves to Step-1 till adequate power is harvested. It will resume the execution while moving from Step-2 to Step-7 to complete the remaining number of executions before finally going to Step-1. For the case once the harvested power is completely drained, the system will restart from Step-1, same like power-on reset.

V. Implementation and Evaluation

We implement FirmSwitch on Intel-WISP CRFID (DL WISP V4.1) as shown in left part of Fig. 6, and develop a User Interface in C# (CSharp) for commercial RFID reader (Impinj Speedway R420) shown in Fig. 5. To measure the power dissipation with and without FirmSwitch, we make a small modification to Intel-WISP and put a series resistor of 30Ω in power path of the MCU, like in [23]. We find out that a value between 30-33Ω 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 value results in a voltage drop in least microvolts which is difficult to measure with high resolution from a general purpose Multimeter (Voltage Meter).

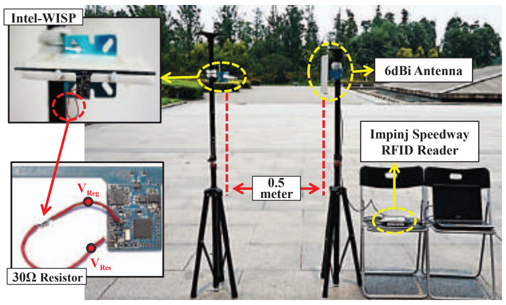


Figure 6. Experimental Setup for overall System Evaluation.

A. User Interface

The User Interface connects to commercial reader and passes the switching parameters through $Write$ command. It also reads the backscattered CRFID tag's data and displays the results. We demonstrate our User Interface in Fig. 5 which switches firmwares inside Intel-WISP CRFID tag having Accelerometer, Temperature Sensor and EPC (TID) firmwares. These firmwares are firstly downloaded as a single firmware inside MCU flash through a wired interface. Next, the User Interface is connected to RFID reader and we manually enter the parameters of $Write$ command ($MemoryBank = 11$ means to switch firmware, $WordPtr$ field gives the address of the target firmware and $Data$ field determines the number of executions). In our evaluation, we first read the temperature for five times using the onboard temperature sensor. Then, we manually edit the $Memory\ Bank$, $WordPtr$ and $Data$ fields on User Interface to switch firmware execution to Accelerometer sensing. The Decoding Module switches to Accelerometer firmware and backscatters the motion values along three axis (the movement of “Jupiter” icon gives the pictorial illustration of x,y,z measurements).

The User Interface also illustrates the pre-defined and floating EPC values in top right window of Fig. 5. The pre-defined EPC value is visible at 2.946 sec whereby we instruct the CRFID to select the Temperature Sensor firmware and execute it for 5 times. Next five subsequent floating EPC values (from 3.025 to 4.092sec) give us the temperature sensor values. Then, the reader passes next $Write$ instruction for which the tag switches to Accelerometer firmware (the pre-defined EPC value shown at 4.360sec). During this operation, we use pre-calculated CRC values, as explained earlier.

B. Evaluation Metrics

We evaluate our system at an outdoor place with both the reader and the tag placed at 0.5 meter distance, as shown in Fig. 6. The CRFID tag is equipped with a 30Ω resistor to measure the power overhead by the FirmSwitch, as discussed before. The evaluation is aimed for following three aspects:

1) Overall System Evaluation: The time delay of a single firmware (Accelerometer code) is measured with and without FirmSwitch. This also checks the conformance of FirmSwitch to EPC standard with commercial RFID reader.

2) Time and Energy Overhead: The time delay and energy overhead is measured once FirmSwitch switches two firmwares. The evaluation excludes the overhead caused by

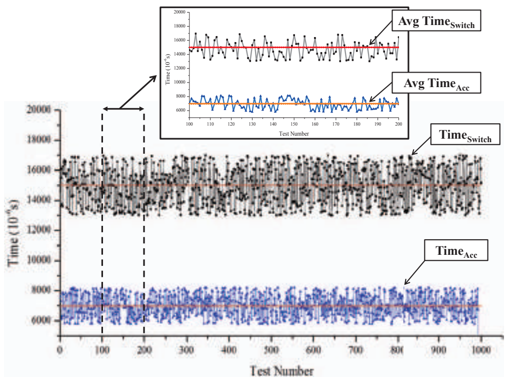


Figure 7. System evaluation once Intel-WISP executes Accelerometer code without FirmSwitch (annotated by $Time_{Acc}$ and $AvgTime_{Acc}$) and with FirmSwitch ($Time_{Switch}$ and $AvgTime_{Switch}$) for 1000 cycles. The former executes the EPC protocol till Inventory Round while the latter executes both the Inventory and the Access Rounds.

EPC protocol and is specific to the functionality of Decoding Module at two MCU clock frequencies, 1.127 and 3.472 MHz.

3) Success Rate and Interrogation Range: We calculate the success rate in line with interrogation range for FirmSwitch and compare it with two firmwares, the first backscatters the sensor data in TID field while the second firmware calculates the CRC value.

VI. Evaluation Results

A. Overall System Evaluation

We check the conformity of our scheme with EPC protocol. Since, EPC protocol puts strict limits on interrogation and reply timings of the reader and the tag, therefore in this evaluation, we restrict ourself to time measurements of overall system with and without FirmSwitch. The time and energy overhead of FirmSwitch's switching operation (Decoding Module) would be illustrated in next section. In here, we first use the CRFID tag without FirmSwitch and continuously read the Accelerometer data through RFID reader for 1000 times. Next, we employ FirmSwitch firmware whereby the Decoding Module selects and executes the Accelerometer firmware during every cycle of interrogation. This depicts the overall system delay introduced by FirmSwitch in selecting, switching and executing a firmware.

We evaluate our system at an outdoor place with both reader and tag placed at 0.5 meter distance, as shown in Fig. 6. The results are shown in Fig. 7. The average time delay for overall system without FirmSwitch is annotated as $AvgTime_{Acc}$ which comes out to be 6988.721μsec. Similarly, the average time delay with FirmSwitch is annotated as $AvgTime_{Switch}$ which comes out to be 15004.174μsec. The time difference, 8015.453μsec is the time taken by Decoding Module to execute the EPC protocol till Access Round to read the switching parameters from the $Write$ command, and resultantly select, load and execute the Accelerometer firmware. In brief, the overall delay introduced by FirmSwitch is well confined within time limits defined by the EPC standard.

Table I
Summery of Time and Energy measurements

Switching Operation	Clock *(MHz)*	V_{Res} *(V)*	V_{Reg} *(V)*	$\triangle$V *(V)*	T_{switch} *(μs)*	Energy *(nJ)*
A→B	1.127	1.7851	1.8009	0.0158	1491	1401.8
A←B	1.127	1.7883	1.8015	0.0132	496	390.3
A→B	3.462	1.7848	1.8051	0.0203	478	577.3
A←B	3.462	1.7859	1.8073	0.0214	160	203.8
C→D	1.127	1.7813	1.8044	0.0231	19.6	26.9
C←D	1.127	1.7826	1.8049	0.0223	20.2	26.8
C→D	3.462	1.7808	1.8057	0.0249	7.8	11.5
C←D	3.462	1.7795	1.8048	0.0253	8.0	12.0
B→B	1.127	1.7817	1.8046	0.0229	1498	2037.3
B←B	3.462	1.7816	1.8055	0.0239	480	681.3

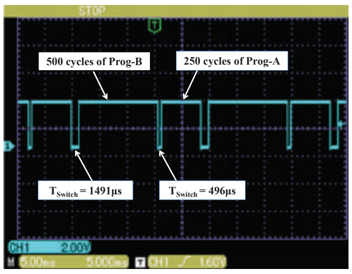
B. Time and Energy Overhead by FirmSwitch

To measure the time delay and energy overhead presented by FirmSwitch, we devise a *Scheduling Scheme* which switches the firmware execution between four different firmwares. This gives us the overhead of Decoding module which performs the actual switching procedure.

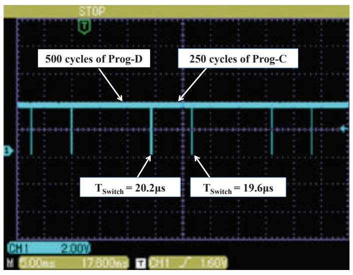
1) Time delay measurements: To demonstrate time consumption, we devise a *Scheduling Scheme* which gives us good illustration about time delay on the Oscilloscope. We take four different firmwares, firmware-A, B, C and D with 2.14, 3.07, 1.216 and 1.735 Kbytes, respectively. Next, we compile two firmwares to flash memory at one time, and check the time delay between switching the codes, as explained below:

For illustration, we explain *Switching Scheme* for two firmwares, A and B which are switched to each other in a repetitive manner for 1000 cycles. During each cycle, firmware-B executes for 500 times followed by the Decoding Module to select and load firmware-A to execute for 250 times. The choice of 500 and 250 executions is empirical with an aim to have a clear view on the Oscilloscope to differentiate two firmwares, and also adequate enough to measure the time delay. To visualize the firmware execution and switching delay in form of voltage waves, we set a flag to high state whenever decoding module selects and loads a firmware. Similarly, whenever a firmware completes its execution, the flag is set low. The toggling of flag before and after the loading and execution of each firmware renders us the indication that an event has started and completed. We use MCU's GPIO P3.0 as a flag and view its output as a voltage wave on the Oscilloscope, as shown in Fig. 8. We repeat same procedure for all four firmwares to measure the time delay and energy consumption. Also, each experiment is performed for two clock frequencies, 1.127 and 3.472 MHz, to see the impact of MCU clock on the execution time.

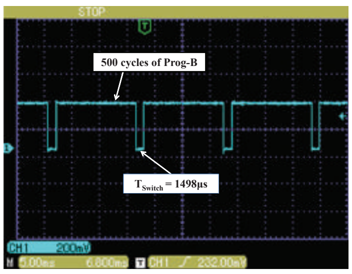
The results in Table I show a maximum switching time of 1498 μsec for switching firmware-A to firmware-B and a minimum switching time of 7.8 μsec for switching firmware-C to firmware-D. The difference in time delay is caused by the size and initialization time taken by each firmware. Same is more illustrative in Table I and Fig 8 once firmware-B is switched to itself. In here, the time delay is 1498 μsec at 1.127 MHz clock and the Oscilloscope waveform depicts the same dip for switching between same firmware, which is different in other cases.



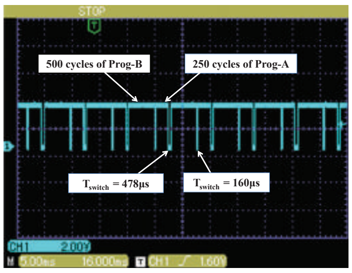
(a) $Prog_A$ and $Prog_B$ at 1MHz



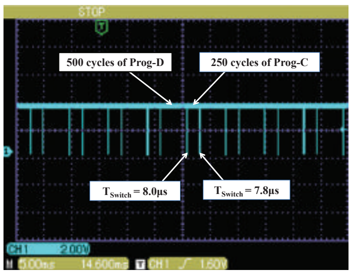
(b) $Prog_C$ and $Prog_D$ at 1MHz



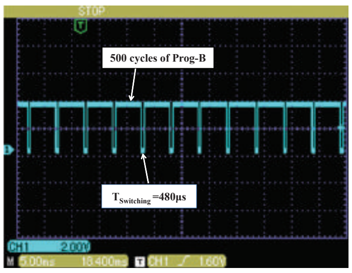
(c) $Prog_B$ and $Prog_B$ at 1MHz



(d) $Prog_A$ and $Prog_B$ at 3MHz



(e) $Prog_C$ and $Prog_D$ at 3MHz



(f) $Prog_B$ and $Prog_B$ at 3MHz

Figure 8. Delay measurements of FirmSwitch. The figures illustrate the time consumed by Decoding Module to switch among four firmwares, A, B, C and D. The evaluation is performed for 1.127 and 3.462 MHz clock frequencies.

The results of clock frequency are quite understandable. As frequency is increased from 1.127 to 3.472 MHz (roughly three times), the number of computations per second also increase. Since, all four firmwares are intended for different functions (they differ in number of variables and execution flow), therefore, the execution time is also different. For example, once the clock frequency is increased, the time taken by firmware-A switching to firmware-B reduces by a factor of 3.11. Similarly, the time consumed in firmware-C switching to firmware-D reduces by 2.51. We observe nearly the same time factors once firmwares are switched in the reversed order, i.e. once we decrease clock frequency, the time for firmware-B switching to firmware-A increases by a factor of 3.1, and, firmware-D switching to firmware-C increases by 2.52. The time factor for firmware-B switching to itself remains 3.12. The results of time and clock frequency are quite illustrative from Fig. 8 and Table I.

2) FirmSwitch Energy overhead: We employ the same procedure to measure the energy overhead of FirmSwitch. The modified Intel-WISP tag (with 30Ω resistor) is used to measure the power consumed by the MCU. In Fig. 6, the output of Voltage Regulator (LDO) is 1.8V which we call as V_{Reg}. The voltage dropped by 30Ω resistor is ΔV while the resultant voltage input to MCU is termed as V_{Res}. Our evaluation begins once the FirmSwitch switches firmwares as shown in the Table I at two clock frequencies, 3.462 and 1.127 MHz. We switch the firmwares for 100 rounds and measure the voltage drop across resistor with a Multimeter. Our method of measuring voltage drop across resistor follows the procedure from [23] which is already used to measure the power consumption of Intel-WISP CRFID during encryption. We highlight that each time we power-on MCU, the output of LDO, V_{Reg}, is around 1.8V with some tolerance (as per our experiences we measure tolerance from 1.8073V to 1.8009V). This is because of tolerances in Load Regulation (variations in MCU's current), Line Regulation (non-linear behaviour of power harvester) and internal tolerance of the LDO [24].

The power consumption is calculated by $\Delta V^2/R$. Our results illustrate a maximum energy consumption of 2.037μJ once firmware-B is switched to firmware-B at 1.127 MHz clock, while the lowest energy consumption is 11.5nJ once firmware-C is switched to firmware-D at 1.127 MHz. In general, we observe that lower MCU clocks consume more power than higher clock frequencies as shown in Table I.

Lastly, we emphasize that the energy consumption of power harvesting designs is a critical parameter. The situation for CRFID tags is more serious as a commercial tag typically operates at 150μW power [25] whereas only the MCU in WISP-CRFID consumes power in amounts of 960μW or more [23]. From the results in Table I, we observe that energy performance of FirmSwitch spans from 11.5nJ to 2.037μJ for switching four firmwares of different sizes. Such a low power consumption is the reason that we get adequate success rate and range performance as illustrated in next section.

3) Success Rate vs. Interrogation Range: We evaluate our system against two firmwares. First firmware only executes the EPC protocol till TID (like legacy Intel-WISP tag) while the second pre-calculates the CRC like in Harmony [14]. The former implements the EPC protocol till Inventory Round to backscatter the data in TID field. The latter executes the EPC protocol till Access Round to backscatter data with pre-calculated CRC. For FirmSwitch, we use Accelerometer switching code which we used during overall system evaluation (Section VI-A). The evaluation is performed at an open place similar to setup in Fig. 6 but during this experiment, we radially move the tag away from the reader in small intervals.

The evaluation results are shown in Fig. 9 which depict 87% success rate for 0.5m. FirmSwitch consumes more energy

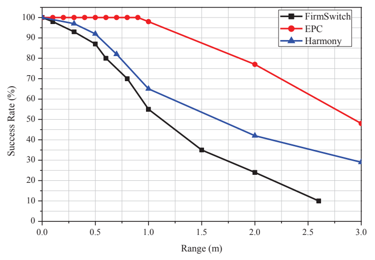


Figure 9. Range comparison for FirmSwitch with "Harmony" and EPC backscatter (the CRFID backscatters the TID only).

than other two schemes because it backscatters sensor data as well as pre-calculates CRC, which results in lower range. Till 1m distance, our system closely follows Harmony [14] but performance degrades within 1-2.5m. The decrease in success rate and interrogation distance over 1m is quite obvious; FirmSwitch keeps the CRFID in active mode for Decoding Module to select and execute new firmware and backscatter the sensor data. It is not only energy but power $(energy \times time)$ which is the reason that below 1m, FirmSwitch competes Harmony because the CRFID tag harvests power at lower distances (more time to harvest). As the distance increases, the harvested power decrease while the Decoding Module in FirmSwitch has to stay in Active Mode for firmware switching and data backscattering operations (Section IV-C2), which in result, decreases the range and the success rate.

VII. Conclusion And Future Work

We present the design, implementation and evaluation of FirmSwitch scheme which is used for wirelessly switching the firmwares on CRFIDs. The user compiles the intended firmwares in a once-for-all fashion during the deployment phase which can be switched to each other during system run-time through commercial RFID reader and the EPC protocol. To this end, an in-depth discussion is carried on for three phases of FirmSwitch, Firmware Arrangement, Instruction Encoding, and Decoding and Execution. Incase the switching process fails, the reader can send another switch command to the tag or wait till the time tag is drained out of energy and restarts again.

We evaluate FirmSwitch on Intel-WISP CRFID tag and evaluate our scheme from two aspects. First, we test Intel-WISP with and without FirmSwitch to check system overhead in terms of time delay, success rate and the interrogation range. Second, we employ a Scheduling Scheme for four different firmwares and evaluate time delay and energy overhead as incurred by the FirmSwitch itself. As a result, we propose FirmSwitch as a viable and practical approach for firmware flexibility in CRFID systems without any modifications to CRFID tag, RFID reader or the EPC protocol.

In future, we plan to equip the CRFID with a modified harvester, with which it would be possible to harvest higher voltage ($\geq 2.2V$ in Intel-WISP V4.1) to re-program the MCU flash. Then our work can be extended to wirelessly receive, load and execute new firmwares without any wired access. In brief, we plan to re-program a new firmware using the "OTA programming" concept.

Acknowledgments: Many thanks to the anonymous reviewers and to our shepherd, Damith C. Ranasinghe, for their insightful suggestions. This work is supported by National Natural Science Foundation of China (Grant No. 61472068, 61173171), and the Project funded by China Postdoctoral Science Foundation (Grant No. 2014M550466).

References

[1] A. Dementyev and J.R. Smith. A Wearable UHF RFID-Based EEG system. In *IEEE RFID Conference, 2013*.
[2] E. Hoque, R.F. Dickerson and J.A. Stankovic. Monitoring Body Positions and Movements During Sleep Using WISPs. In *Wireless Health*, pages 44–53, 2010.
[3] N. Saxena and J. Voris. Still and Silent: Motion Detection for Enhanced RFID Security and Privacy without Changing The Usage Model. In *Radio Frequency Identification: Security and Privacy Issues*. 2010.
[4] V. Liu et. al. Ambient Backscatter: Wireless Communication Out of Thin Air. In *ACM SIGCOMM*, 2013.
[5] A.N. Parks and J.R. Smith. Sifting Through The Airwaves: Efficient and Scalable Multiband RF Harvesting. In *IEEE RFID Conference, 2014*.
[6] M. Buettner et. al. Recognizing Daily Activities with RFID-Based Sensors. In *ACM Conference on Ubiquitous computing, 2009*.
[7] M. Philipose et. al. Battery-free Wireless Identification and Sensing. *Pervasive Computing, IEEE*, 4(1):37–45, 2005.
[8] H. Zhang and J. Gummeson et. al. MOO: A Batteryless Computational RFID and Sensing Platform. *University of Massachusetts Computer Science Technical Report UM-CS-2011-020*, 2011.
[9] J. Gummeson, S.S. Clark and K. Fu et. al. On The Limits of Effective Hybrid Micro-Energy Harvesting on Mobile CRFID Sensors. In *ACM MobiSys, 2010*.
[10] M. Marroncelli, D. Trinchero and V. Lakafosis et. al. Concealable, Low-Cost Paper-Printed Antennas for WISP-Based RFIDs. In *IEEE RFID Conference, 2011*.
[11] A. Dementyev, J. Gummeson and D. Thrasher et. al. Wirelessly Powered Bistable Display Tags. In *ACM Ubicomp, 2013*.
[12] UMASS MOO in Concrete, https://spqr.eecs.umich.edu/moo/apps/concrete/ [Accessed December 04, 2014].
[13] B. Ransford and S.S. Clark et. al. Getting Things Done on Computational RFIDs with Energy-Aware Checkpointing and Voltage-Aware Scheduling. In *HotPower*, 2008.
[14] Y. Zheng and M. Li. Read Bulk Data from Computational RFIDs. In *IEEE INFOCOM*, 2014.
[15] MSP403FR5969 Datasheet. Texas Instruments, 16 MHz Ultra-Low-Power Microcontroller. , http://www.ti.com/product/MSP430FR5969/datasheet [Accessed December 04, 2014].
[16] Benjamin Ransford, A Rudimentary Bootloader for Computational RFIDs https://web.cs.umass.edu/publication/docs/2010/UM-CS-2010-061.pdf [Accessed December 04, 2014].
[17] A. Czeskis, K. Koscher and J.R. Smith et. al. RFIDs and Secret Handshakes: Defending Against Ghost-and-Leech Attacks and Unauthorized Reads with Context-Aware Communications. In *ACM CCS, 2008*.
[18] S.A. Ahson and M. Ilyas. *RFID Handbook: Applications, Technology, Security, and Privacy*. CRC press, 2010.
[19] M. Salajegheh, Y. Wang and K. Fu et. al. Exploiting Half-Wits: Smarter Storage for Low-Power Devices. In *FAST, 2011*.
[20] D.E. Holcomb, W.P. Burleson and K. Fu. Power-Up SRAM State As An Identifying Fingerprint and Source of True Random Numbers. *Computers, IEEE Transactions on*, 58(9):1198–1210, 2009.
[21] P. Zhang, J. Gummeson and D. Ganesan. BLINK: A High Throughput Link Layer for Backscatter Communication. In *ACM MobiSys, 2012*.
[22] EPCglobal. Class 1 Generation 2 UHF Air Interface Protocol Standard Version 1.0.9. 2005.
[23] J.R. Smith. *Wirelessly Powered Sensor Networks and Computational RFID*. Springer, 2013.
[24] NCP583 Datasheet, Ultra Low current LDO Regulator. OnSemiconductor, http://www.onsemi.com/pub_link/Collateral/NCP583-D.pdf [Accessed December 04, 2014].
[25] S. Wong and C. Chen. Power Efficient Multi-Stage CMOS Rectifier Design for UHF RFID Tags. *INTEGRATION, the VLSI journal*, 44(3):242–255, 2011.