An Automated, Reconfigurable, Low-Power RFID Tag

Alex K. Jones, Raymond R. Hoare, Swapna R. Dontharaju, Shenchih Tung
Ralph Sprang, Josh Fazekas, James T. Cain, and Marlin H. Mickle
{akjones,hoare,cain,mickle}@ece.pitt.edu

Department of Electrical and Computer Engineering
University of Pittsburgh
Pittsburgh, PA 15261

ABSTRACT

This paper describes an ultra low power active RFID tag and its automated design flow. *RFID primitives* to be supported by the tag are enumerated with *RFID macros* and the behavior of each primitive is specified using ANSI-C within the template to automatically generate the tag controller. Two power saving components, a passive transceiver/burst switch and a smart buffer, are presented to save power and increase tag lifetime. Based on a test program, the processors required 183, 43, and 19 μJ per transaction for StrongARM, XScale, and EISC processors, respectively. Three hardware controllers using a Fusion FPGA, Coolrunner II CPLD, and ASIC required 13 nJ, 1.3 nJ, and 0.07 nJ per transaction.

Categories and Subject Descriptors

C.3 [**Special Purpose and Application-Based Systems**]: Real-time and embedded systems; B.7.1 [**Integrated Circuits**]: Types and Design Styles—*Gate arrays*

General Terms

Performance, Design, Human Factors, Languages

1. INTRODUCTION

RFID tags generally come in two types, active and passive. Active tags require an internal power source (usually a battery) to power the transceiver for receiving queries and transmitting responses. The power supply also powers the tag's controller, which may be an ASIC or an embedded microprocessor. Passive tags do not contain an internal power source. As a result, these tags not only receive information from a query, they also receive energy. This energy is used to power the tag to determine and send a response to the query. While passive tags are generally cheaper than active tags, they have two major disadvantages: (1) the range of passive tags is significantly lower than active tags and (2) the complexity of response is significantly reduced over active tags due to the limited energy budget. However, active tags in addition to being more costly than passive tags also require maintenance, often to change the battery.

DAC 2006, July 24–28, 2006, San Francisco, California, USA.

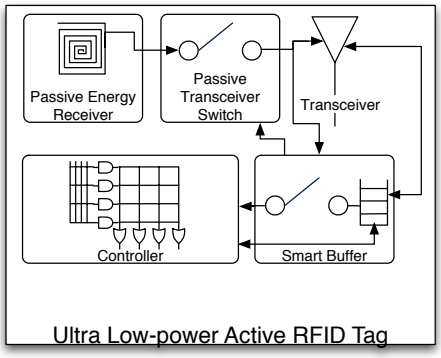

Figure 1: Extensible, low-power RFID tag.

The tag shown in Figure 1 addresses two current problems in the field of RFID. The first problem is that many applications require a "custom" tag, either active or passive. Standards have been developed but in some cases additional capabilities are required for a specific application. Currently, the cost of designing and implementing such a custom tag is prohibitive. Our tag consists of a programmable controller that can be easily customized to work with different standards combined with custom proprietary commands. The second problem is that there are many applications that require the capability of an active RFID tag but the application is such that battery replacement is at least inconvenient, if not impossible. Our tag provides the capability of an active tag but with several power saving components that allow the tag to operate on a single battery throughout its lifetime.

To program the controller, a design automation tool has been developed that allows *RFID primitives* to be specified using *RFID macros*, an assembly-like format. These RFID macros are processed to generate a template file to specify the behavior for each macro. All behavior is specified using ANSI-C allowing the user to create arbitrarily complex behaviors. Finally, the RFID compiler generates the final controller used for managing the tag. In order to reduce the power consumption of the tag we present a technique called a burst switch that allows the tag to remain asleep with its transceiver and controller either completely off or in a hibernate mode. Additionally, during a transaction, the power can be further reduced by a smart buffer that keeps the tag in a standby mode with the controller remaining asleep until it is needed. Because RFID

packets are transmitted serially with kHz speed clocks, even allowing the controller to remain off can contribute significant power savings.

To prove this design methodology, several RFID tag systems are considered. The first class of tags employ an embedded microprocessor core as the controller. This requires the RFID compiler to generate a C program that is compiled onto the microprocessor with its embedded C compiler. The second class replaces the microprocessor with a custom hardware controller requiring the RFID compiler to generate a controller in synthesizable VHDL. Both techniques integrate the burst switch and smart buffer with an existing transceiver. To prototype the system, the smart buffer was implemented in an FPGA and a burst switch was constructed..

The remainder of the paper is organized as follows: Section 2 describes related work in the RFID systems domain. The RFID compiler is presented in Section 3. Section 4 briefly introduces the power saving technologies. Results are related in Section 5. Section 6 discusses conclusions and future directions.

2. RELATED WORK

Research and development in RFID has been focused on hardware and firmware components such as active and passive RFID tags, readers, and embedded software, for the purpose of its deployment in specific application domains. RFID is being incorporated in supply chain management, giving enterprises a real-time view on the location and integrity of their inventories. RFID technology is used in a location sensing prototype system for locating objects inside buildings [14]. RFID tags have been adopted in the Vatican Library in Rome to identify and manage its extensive book and document collection [20]. RFID tags and intelligent transponders are widespread for vehicle to roadside communications, road tolling and vehicle access control [1]. The medical industry has deployed RFID technology in "Mobile Healthcare Systems" for positioning and identifying persons and objects inside the hospitals [13]. Different types of RFID prototype systems are being developed to support all aspects of aviation baggage tracking, sorting and reconciliation [2]. Recent research has focused on the collection and storage of RFID data using Geo-Time visualization [16]. Distributed Application Specification Language (DASL) has been used to model and deploy software applications to process RFID event data [12]. Most of the above RFID systems use proprietary hardware and software that cannot tolerate changes to the application or standard. However, the use of design automation for the development of RFID systems and the combination of passive and active technology together for increased capability and reduced power have not been studied.

3. RFID TAG CONTROLLER AUTOMATION

The RFID communication system consists of a transponder or tag and an interrogator or reader. The format for exchanges between the interrogator and the transponder is a set of commands or primitives that requests that the transponder perform a set of actions. The specifications of these commands varies from one standard to another. The flow of the RFID compiler is specified in Figure 2. This particular compiler can accept virtually any set of commands as input and target a microprocessor or hardware device to provide the RFID tag controller functionality.

3.1 Specification of Macros

The descriptions corresponding to the *RFID primitives* appear as simple, assembly-like instructions called *RFID macros*. For example, the *RFID macro* representation of the *primitive* "Owner id write" of the ISO standard is shown here. The format of the respective fields of the *primitive* and its corresponding response are both illustrated in Figure 3.

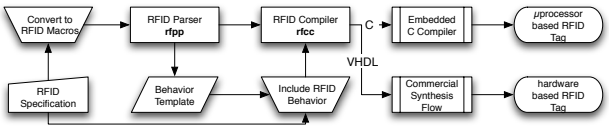

Figure 2: Specification Methodology and Compilation Flow

The command code of each *RFID primitive* is a unique field or *opcode* that serves as the identifier. Each of the RFID *primitives* also contains a subset of fields with varying lengths providing positions for data present in a command as can be inferred from Figure 3. Similarly, the tag response to each *RFID primitive* has fields of varying lengths.

"OWNERID WRITE" COMMAND FORMAT

PREFIX	TYPE	OWNER ID	TAG ID	INTERROGATOR ID	COMMAND CODE	CRC
8 BITS	8 BITS	24 BITS	32 BITS	16 BITS	8 BITS	16 BITS

RESPONSE FORMAT

TAG STATUS	MESSAGE LENGTH	INTERROGATOR ID	TAG ID	COMMAND CODE	CRC
16 BITS	8 BITS	16 BITS	32BITS	8 BITS	16 BITS

TAG STATUS FORMAT

MODE FIELD	RESERVED	ACKNOWLEDGE	RESERVED	TAG TYPE	RESERVED	USER ID	BATTERY
4 BITS	3 BITS	4 BITS	2 BITS	3 BITS	1 BIT	1 BIT	1 BIT

Figure 3: Owner id write primitive and response format (ISO)

Each *RFID macro* description contains a relatively short character string corresponding to the specific name of the *primitive*, a distinct number corresponding to the value of the *opcode*, a set of operands will correspond to the *primitive* and finally, a set of operands are included that correspond to the standard response.

Figure 4 shows an example *RFID macros* file containing the Owner Id Write *primitive*. The macros file has a *declarations* section and a *main* section. The *declarations* section allows the user to pre-declare the lengths of all of the corresponding fields that occur in each of the *primitives* and responses. In the section identified as *main*, the *primitives* and their specific responses are defined in terms of their fields.

```
declarations
prefix(8)
type(8)
ownerid(24)
interid(16)
tagid(32)
comcode(8)
siz(16)
res(8)
crc(16)
tagstatus(16) [
      modefield(4)
      reserved1(3)
      acknowledge(4)
      reserved2(2)
      tagtype(3)
      reserved3(1)
      userid(1)
      battery(1)
]
mesglen(8)

main
ionw (137)     prefix     type     ownerid   tagid    interid   comcode  crc
               tagstatus  mesglen  interid   tagid    comcode   crc
icol (16)      prefix     type     ownerid   interid  comcode   siz      res   crc
               tagstatus  mesglen  interid   tagid    ownerid   crc
```

Figure 4: Macros specification

In a number of cases, the *primitive* fields or the response will have multiple nested fields with a variety of lengths. The specific fields can be easily described as illustrated in Figure 4. This provides the user with the ability to adopt any level of granularity in or-

der to manipulate the *primitives* and their corresponding responses. In the *macro* illustrated below, the string denoting the *owner id write* command is `ionw`. The decimal value of the command code for this specific command is "137".

Initially, the Extensible Markup Language (XML) [18] was evaluated as a candidate for the macro specification. In this specific example, the form presented in Figure 4 does not require a full parser and was considered easier for the end user to understand. In addition, it was noted that the nested structures were hard to work with when employing the pre-existing XML parsers.

3.2 Behavior Template

The communications from the RFID interrogator (Reader) is accomplished by transmitting the *primitive* to the RFID tag using a standard air interface. The tag will respond by changing the current state and / or transmitting a designated response message to the interrogator. The user has the capability of specifying the tag behavior in ANSI C.

In order to simplify the user interaction, a template is generated by the RFID parser to assist the user in providing the response behavior code. C language constructs (conditionals, loops, etc.) can be included by the user in order to check the values of the fields of an incoming *RFID primitive* to specify the field values of the response. The resulting template, which has been generated as a result of the collection command (*icol* in the *macros* specification from Figure 3) is shown in Figure 5(a). Similar templates are contained in a file for each of the macros that are a part of the macros specification file.

The details involving size and field position in the command of the interrogator and the corresponding response packet are handled by the compiler. Therefore, complexities encountered in unpacking the command and subsequently packing the response can be abstracted from the user as shown in Figure 5. However, the ability for the user to manipulate each of the individual fields in the response is intact. Therefore, the response customization along with the corresponding state changes can increase in complexity with user ease.

```
// Behavioral C of ICOL follows
TAGSTATUS.modefield =
TAGSTATUS.reserved1 =
TAGSTATUS.acknowledge =
TAGSTATUS.reserved2 =
TAGSTATUS.tagtype =
TAGSTATUS.reserved3 =
TAGSTATUS.userid =
TAGSTATUS.battery =

RESPONSE.mesglen =
RESPONSE.interid =
RESPONSE.tagid =
RESPONSE.ownerid =
RESPONSE.crc =
```

(a) Template Generated

```
TAGSTATUS.modefield = 0;
TAGSTATUS.reserved1 = 7;

if(commandValid)
  TAGSTATUS.acknowledge = 0;
else
  TAGSTATUS.acknowledge = 1;

TAGSTATUS.reserved2 = 3;
TAGSTATUS.tagtype = 2;
TAGSTATUS.reserved3 = 1;
TAGSTATUS.userid = 1;
TAGSTATUS.battery = 0;
RESPONSE.mesglen = 112;
RESPONSE.interid = interid;
RESPONSE.tagid = tagid;
RESPONSE.ownerid = ownerid;
RESPONSE.crc = crc;
```

(b) After User Input

Figure 5: Tag Behavior for Collection Command

3.3 Compiler-Generated RFID Tag Program

The code generation is the final compiler phase determined by the tag behavior and the input macros specification. Code generation can be in the form of ANSI-C for general purpose microprocessor controllers or VHDL in the case of hardware controllers. The decode instructions generated by the compiler identify the received *RFID primitive*. For each incoming command, routines are

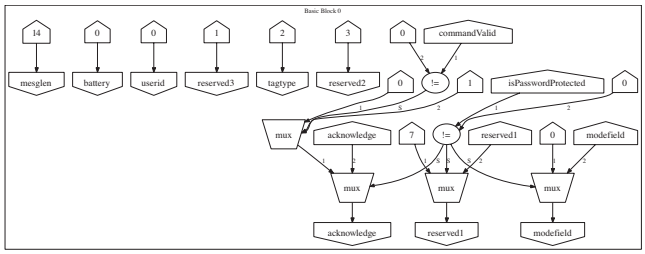

Figure 6: SDFG for the "Owner id write" (`ionw`) command.

generated by the compiler, which unpack the command generating the fields that it is expected to contain. The corresponding behavior is attached to each field, and the corresponding routines for packing the response are then generated. By virtue of the fact that C is a significant and universally known language than classical hardware description languages, the primitive behaviors are specified in C in the case of a specified VHDL target. Therefore the C code must be converted to a readily synthesizable hardware code.

During the hardware conversion, the C code is converted into a control and data flow graph (CDFG). Compilers commonly use CDFGs to perform optimizations and transformations. Typically, behavioral synthesis tools will also use CDFGs as an internal representation [19]. In many cases, the control dependencies present in a CDFG create cycle boundries during high-level synthesis.

In contrast, in our RFID compiler the CDFG is transformed into an entirely combinational representation by the SuperCISC Compiler. The result is a super data flow graph (SDFG). The SuperCISC compiler [10, 6] takes advantage of well known compiler transformations including loop unrolling and function inlining and *hardware predication* in order to convert each control dependency into a data dependency creating a combinational representation. The SDFG for the `ionw` is shown in Figure 6. The need for many potentially high-power consuming sequential constructs such as registers and clock trees are removed by this technique. Thus, the resulting SDFG based hardware implementations are extremely power efficient [11].

4. POWER REDUCTION TECHNIQUES

A typical RFID tag consists of three primary elements; (1) transmitter, (2) controller, and (3) receiver. From the classical operational scenario, the transmitter is only needed when there is data to transmit. Thus, the transmitter can easily be turned off when not needed. The controller can be put in a low power/sleep mode to save energy. The only element that is not in a low power mode is the receiver, which is kept active to know when there is an inquiry from the interrogator. Thus, the receiver becomes is the determinant of the *shelf life* of an active RFID tag.

To reduce the power of the RFID tag receiver two main technologes are employed, a passive transceiver or *burst switch* allows the tag to remain in a sleep mode until activated with RF energy and a *smart buffer*, which allows the controller to remain asleep while an incoming packet is buffered.

The burst switch, shown in Figure 7, is a passive receiver that is powered by RF energy from the air, which allows the active receiver to be turned completely off. Thus, the shelf life is determined primarily by the controller sleep power. The active receiver and smart buffer are turned only when the burst switch is activated by the wake-up signal(s) from the interrogator. The controller is managed by the smart buffer. The result is an active tag with minimal power consumption that can operate on a single battery for its lifetime.

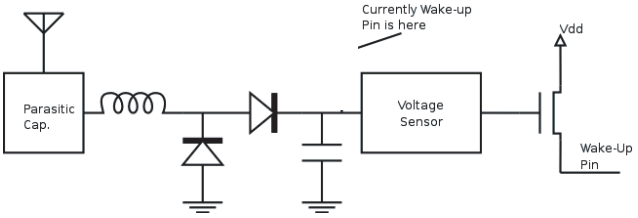

Figure 7: Burst switch schematic.

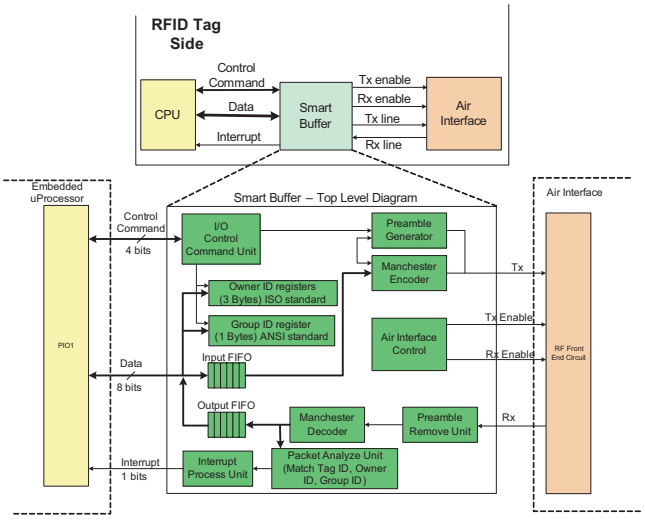

Figure 8: Block diagram for the smart buffer architecture.

The RF transceiver coprocessor or *smart buffer* assists the embedded controller to sleep or stay in a low-power idle mode while any non-relevant packets arriving at the RFID tag are ignored and valid packets wake up the processor to respond appropriately. Figure 8 depicts the top level diagram of the smart buffer. The preamble removal unit detects a valid incoming preamble signal. The Manchester decoder translates the Manchester encoded data immediately following a valid preamble. Based on the header information in the RFID packet, the packet analysis unit determines specific characteristics of the packet needed to decide whether to wake up the controller for response generation. The interrupt process unit wakes up the controller only when a complete RFID packet is stored into the output FIFO and the analysis result forwarded by the packet analysis unit is positive. The command control unit fetches control commands from the controller, such as read and write data to FIFO, update Tag ID or Group/Owner ID, and so forth. The preamble generator generates a preamble signal followed by the final sync pulse. When the final sync pulse is transmitted, the preamble generator informs the Manchester encoder to begin to output its serial encoded data. The air interface unit has output control signals which are enabled when the smart buffer needs to listen for incoming signals from the air, or when it is ready to transmit a response to readers. Refer to [9] for details.

5. RESULTS

The prototype microprocessor system was simulated using an Altera APEX 20 FPGA for smart buffer implementation and three different processor cores: the Intel StrongARM at 206 MHz [7], the Intel XScale 80200 at 733 MHz [8], and the 16-bit EISC microprocessor at 50 MHz from ADChips [3]. A prototype system was built with an Avnet development board, an EISC development board, and an air interface prototype board fabricated using PCB Express. The system was tested with a variety of automatically generated controller programs including anywhere from 1 to 14 Primitives. Additional primitives were not fit in the prototype system due to limitations of available memory on the board.

The prototype hardware-based system was implemented using Oki ASIC cells for the smart buffer and two FPGA/CPLD targets, a Xilinx Coolrunner II XC2C512 and an Altera Fusion AFS600 as well as direct ASIC implementation with the Oki cells for the primitive logic. The entire system was also prototyped with a single Spartan 3 XC3S400 and tested with an Opal Kelly board. The system was tested with up to 40 primitives, which includes all primitives from ANSI and ISO standards and several custom primitives. The area and and performance results are shown in Table 1 for the XC3S400 and Table 2 for the XC2C512. Due to space constraints of the Coolrunner II it was not possible to include the smart buffer logic and primitives in single device.

Table 1: Area and performance for implementing smart buffer and primitive logic on a Spartan 3 XC3S400. Maximum frequency (FMax) is in MHz.

Prims	2	4	6	8	10	12	15	20	24	30	35	40
LUTs	1946	1948	1958	1971	2003	2061	2420	2576	2594	2612	2692	2704
% used	27%	27%	27%	27%	27%	28%	33%	35%	36%	36%	37%	37%
FMax	70	70	70	70	68	70	68	67	66	65	65	67

Table 2: Area and performance for implementing the primitive logic on a Coolrunner II XC2C512. Macrocells (MCs), product terms (PTs), registers (Regs), function block inputs (FBIs) are included. Maximum Frequency (FMax) is in MHz.

Prims	2	4	6	8	10	12	15	20	24	30	35	40
MCs	332	335	338	340	350	366	426	447	447	447	449	447
% used	66%	66%	67%	67%	69%	72%	84%	88%	88%	88%	88%	88%
PTs	444	477	514	506	477	552	772	953	993	1106	1181	1213
% used	25%	27%	29%	29%	27%	31%	44%	54%	56%	62%	66%	68%
Regs	262	267	271	271	283	307	422	443	443	443	443	443
% used	52%	53%	53%	53%	56%	60%	83%	87%	87%	87%	87%	87%
FBIs	360	379	408	391	338	396	611	767	801	870	900	914
% used	29%	30%	32%	31%	27%	31%	48%	60%	63%	68%	71%	72%
FMax	49	41	41	40	49	41	41	29	33	33	26	30

5.1 Power Estimation

Power optimization is critical in RFID systems as the power supplied to the tags is limited and battery drain needs to be limited. Because active systems are designed for extremely low-cost large-scale applications, frequent replacement of batteries is not feasible. Current research in power consumption has only focused minimizing power consumption of anti collision protocols [21] and to implement energy conserving access protocols [4] in RFID systems. Our burst switch and smart buffer provide a power down capability to the transceiver and controller, respectively. Unfortunately, The power savings due to these technologies are highly scenario dependent (e.g. the number of accesses per day, number of tags in the system, etc.). Therefore the remaining results are for the active case of the controller using different implementation strategies.

5.1.1 Microprocessor-based Tag

The RFID compiler was used to generate three programs: A with 24 primitives, B with 12 primitives, and C with 4 primitives. Ex-

Table 3: Power consumption results for StrongARM (SA), XScale (XS) and EISC based tags during response generation.

Description	Average Power (mW)			Energy (μJ)		
	SA	XS	EISC[1]	SA	XS	EISC[2]
Program A (1 primitive)	314	244	30	140	32	-
Program B (1 primitive)	322	257	30	183	43	19
Program C (1 primitive)	314	249	30	140	32	17

periments were conducted by executing 1 primitive of Program A, 1 primitive of B, and 1 primitive of C.

The sim-panalyzer [17] and XTREM [5] tools were used to estimate the power dissipation of the microprocessor based tag for the ARM based cores. Sim-panalyzer is a cycle accurate, architecture level power simulator built on the SimpleScalar simulator. XTREM is a SimpleScalar-based power and performance simulator tailored for Intel XScale micro-architecture. We used SimpleScalar's sim-profile to obtain ARM instruction and instruction class profiles for our software. Because an instruction set simulator was not available for the EISC, the application was run on the development board and the execution time was measured by setting a pin output from low to high upon each iteration, and measuring duration using an oscilloscope. The energy consumed by the EISC was based on a static power estimate from ADC [3], which should be within about 10% accuracy of an instruction level power estimation approach [15].

Table 3 shows the power consumption of our program on the StrongArm, XScale, and EISC processors. These results show the power consumption during the active phase of the RFID transaction (e.g. the time after the entire packet is received by the smart buffer and is determined to be relevant). During this time, the smart buffer is active, however as seen for the ASIC implementation in Table 5 this power is negligible compared to the processor power. Both ARM based processors operate in the 250-400 mW range, while the XScale uses significantly less energy. The EISC processor uses an order of magnitude less power, but operates much slower. However, the energy consumed is still less than half of XScale.

5.1.2 *Hardware-based tag*

The power consumption of the the Spartan 3 and Coolrunner II was estimated using Xpower from Xilinx. The Fusion was estimated using SmartPower from Actel. The ASIC implementation was estimated with PrimePower from Synopsys. Switching statistics for the tool were generated from cycle-accurate, post place and route simulations of actual test data.

While the Spartan 3 provides plenty of capacity, its power consumption, albeit potentially lower than the ARM based processors, is not as low as desired. Much of this is due to the quiescent power of 92 mW. In order to further reduce power, the Fusion and Coolrunner II were explored as they have much lower Quiescent powers of 7.5mW and 50 μWs, respectively. The Fusion is of particular interest because it can be entirely powered off while not in use due to its flash memory for storage and configuration, making it idea to work with the burst switch. The smart buffer is segregated from the reconfigurable logic and only the user defined primitive controller generated by the compiler is implemented in the CoolRunner and the Fusion. The power/energy consumption is shown in Table 4.

[1]Static power estimation provided by ADChips [3]

[2]Energy calculation is static power consumption multiplied by measured execution time. - means program did not fit within instruction memory.

Table 4: Primitive logic on a Coolrunner II XC2C512, a Fusion AFS600, and for 0.16 μm ASIC.

Number of Primitives	Target	Power (mW) Dynamic	Quiescent	Total	Energy (nJ) Total
2	Coolrunner	1.06	0.05	1.11	1.11
	Fusion	3.76	7.50	11.26	11.26
	ASIC	0.063	0.0004	0.063	0.063
4	Coolrunner	1.06	0.05	1.11	1.11
	Fusion	3.82	7.50	11.32	11.32
	ASIC	0.063	0.0004	0.063	0.063
6	Coolrunner	1.06	0.05	1.11	1.11
	Fusion	3.87	7.50	11.37	11.37
	ASIC	0.063	0.0004	0.063	0.063
8	Coolrunner	1.07	0.05	1.12	1.12
	Fusion	4.40	7.50	11.90	11.90
	ASIC	0.063	0.0004	0.063	0.063
10	Coolrunner	1.06	0.05	1.11	1.11
	Fusion	4.41	7.50	11.91	11.91
	ASIC	0.063	0.0004	0.063	0.063
20	Coolrunner	1.24	0.05	1.29	1.29
	Fusion	5.23	7.50	12.73	12.73
	ASIC	0.064	0.0004	0.064	0.064
30	Coolrunner	1.24	0.05	1.29	1.29
	Fusion	5.35	7.50	12.85	12.85
	ASIC	0.064	0.0004	0.064	0.064
40	Coolrunner	1.24	0.05	1.29	1.29
	Fusion	5.48	7.50	12.98	12.98
	ASIC	0.065	0.0004	0.065	0.065

5.1.3 *Burst Switch Implementation*

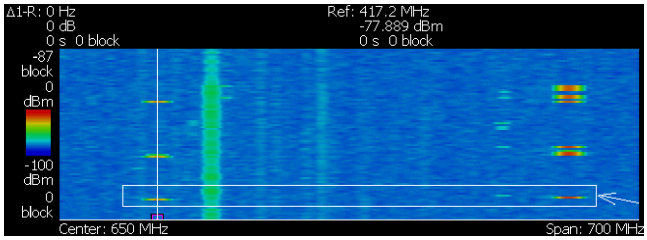

Figure 9: Burst switch verification with a real-time spectrum analyzer.

To verify the burst switch capability, the system was tested using a real-time spectrum analyzer (RTSA) from Tektronix, which is shown in Figure 9. This test was taken with the reader at a distance of 1 m from the tag. In this plot, time is on the Y axis, and frequency is on the X axis. Power is indicated by the color blue being lowest and red being highest. The burst switch activation can be seen in a column near the right at a frequency of approximately 933 MHz. The responses are shown on the left in a column with slightly less intensity at approximately 433 MHz. The white box indicates a burst switch activation which triggered a tag response. The columns of green are radio interference that show up at low intensity within the lab and are not related to the experiment.

5.1.4 *Smart Buffer Implementation*

The smart buffer was prototyped on a Spartan 3 FPGA and studied for ASIC implementation using 0.16 μm Oki standard cells. While it was possible to implement FIFO blocks on the Spartan 3 using Xilinx IP blocks, ASIC versions of these FIFOs were not available. Three components, preamble detection, a 30 kHz wake-up signal detection, and Manchester decoder, were separated from the larger design and synthesized and power profiled independently.

Table 5: ASIC vs. FPGA power for portions of Smart Buffer

Component	Spartan 3	0.16 μm ASIC
Wake-up Signal	0.01 mW	0.001 mW
Preamble Detection	0.87 mW	0.005 mW
Manchester Decoder	0.95 mW	0.284 mW
Quiescent Power	92 mW	0 mW
Total	93.83 mW	0.29 mW

The power estimates of the ASIC and Spartan 3 based smart buffer components are displayed in Table 5. The analyses are based on post synthesis simulation with the exact same stimuli. The FPGA power results were computed in a similar manner to Section 5.1.2. Based on the results from Table 5 the ASIC version of the implementation uses orders of magnitude less dynamic power for all components. Interestingly, the quiescent power of the FPGA alone is nearly three orders of magnitude greater than the dynamic power of the ASIC.

6. CONCLUSIONS AND FUTURE WORK

This paper presents an ultra low power active RFID tag system and an automated tag controller design flow. By providing a burst switch front end, the active transceiver can remain off until requested by a interrogator. Finally, during a communication, the smart buffer allows the controller to remain off until required to process a packet. This system provides the capabilities of an active RFID tag with the maintainability and power consumption similar to passive RFID tag. The RFID compiler can automatically generate software or hardware for both microprocessor and hardware based tags from a simple description of the standards and behavior in C. The system is extensible, as it allows for addition (or removal) of a set of custom *primitives* that may be a subset or superset of the original standard(s).

To select the appropriate tag architecture for an extensible low-power tag, consider the power consumption for the different tag systems described in Section 5.1. Based on the results from Section 5.1.4 coupled with the fact that smart buffer does not require reconfiguration except in drastic redesign of RFID systems, implementing the smart buffer directly in silicon is the preferred option.

Three low-power embedded microprocessors were power profiled, StrongARM, XScale, and EISC. The XScale provided capacity advantages with reasonable power, while the EISC was more power efficient, but could handle relatively few primitives. Three custom hardware controllers were considered, an Actel Fusion, Xilinx Coolrunner II, and direct ASIC implementation. The ASIC approach provides a large power advantage, but is not extensible. The Coolrunner II requires less energy per transaction than the Fusion (1.3 nJ versus 13 nJ); however, the Coolrunner requires the 50 μW of static power to remain programmed during the sleep mode while the Fusion can be entirely disconnected from power. The appropriate choice depends on the RFID system deployment scenerio.

We plan to introduce security into the RFID tag design in the future. Because our tag provides the capability of active tags with the power consumption and maintainability of passive tags, it may be possible to encrypt the communication without significantly degrading the lifetime of the tag. The design automation makes this task easier allowing C-based encryption algorithms to be leveraged. To prevent a malicious interrogator from running down the tag battery by constantly activating the burst switch, it is possible to create activation codes using burst duration and combining multiple burst switches at different frequencies to prevent this denial of service attack.

7. REFERENCES

[1] P. Blythe. RFID for road tolling, road-use pricing and vehicle access control. *IEE Colloquium on RFID Technology*, 1999.

[2] A. Cerino and W. P. Walsh. Research and application of radio frequency identification (RFID) technology to enhance aviation security. In *Proc. IEEE NAECON*, 2000.

[3] Y. Cha. EISC core. ADChips Presentation, February 2005.

[4] I. Chlamtac, C. Petrioli, and J. Redi. Energy-conserving access protocols for identification networks. *IEEE/ACM Transactions on Networking (TON)*, 7(1), February 1999.

[5] C. Gilberto, M. Martonosi, J. Peng, R. Ju, and G. Lueh. XTREM: A power simulator for the Intel XScale core. In *Proc. ACM LCTES*, 2004.

[6] R. Hoare, A. K. Jones, D. Kusic, J. Fazekas, J. Foster, S. Tung, and M. McCloud. Rapid VLIW processor customization for signal processing applications using combinational hardware functions. *EURASIP Journal on Applied Signal Processing*, 2006.

[7] Intel. SA-110 microprocessor technical reference manual, 1998. ftp://download.intel.com.

[8] Intel. Intel PXA27x processor family developers manual, 2004. ftp://download.intel.com.

[9] A. K. Jones, R. Hoare, S. R. Dontharaju, S. Tung, , R. Sprang, J. Fazekas, J. T. Cain, and M. H. Mickle. An automated, FPGA-based reconfigurable, low-power RFID tag. *Journal of Microprocessors and Microsystems*, to appear.

[10] A. K. Jones, R. Hoare, D. Kusic, J. Fazekas, and J. Foster. An FPGA-based VLIW processor with custom hardware execution. In *Proc. of FPGA*, 2005.

[11] A. K. Jones, R. Hoare, D. Kusic, G. Mehta, J. Fazekas, and J. Foster. Reducing power while increasing performance with supercisc. *ACM TECS*, to appear.

[12] M. Kaundinya and A. Syed. Modeling event driven applications with a specification language (MEDASL). In *Proc. ACM OOPSLA*, 2004.

[13] C. Li, L. Liu, S. Chen, C. C. Wu, C. Huang, and X. Chen. Mobile healthcare service system using RFID. In *Proc. IEEE ICNSC*, volume 2, 2004.

[14] L. M. Ni, Y. Liu, Y. C. Lau, and A. P. Patil. LANDMARC: indoor location sensing using active RFID. *Wireless Networks*, 10, November 2004.

[15] J. Russell and M. Jacome. Software power estimation and optimization for high performance, 32-bit embedded processors. In *Proc. of ICCD*, 1998.

[16] D. Shuping and W. Wright. Geotime visualization of RFID supply chain data. RFID Journal, March 2005.

[17] Sim-panalyzer. Simplescalar-ARM power modeling project. http://www.eecs.umich.edu/ panalyzer.

[18] T. Bray and J. Paoli and C. M. Sperberg-McQueen. Extensible markup language (XML) 1.0. W3C Recommendation, 1998. http://www.w3.org/TR/1998/REC-xml-19980210.

[19] X. Tang, T. Jiang, A. Jones, and P. Banerjee. Compiler optimizations in the PACT behavioral synthesis tool for ASICs and FPGAs. In *Proc. of SoC*, September 2003.

[20] TI. Texas instruments' RFID technology streamlines management of vatican library's treasured collections, 2004.

[21] F. Zhou, C. Chen, D. Jin, C. Huang, and H. Min. Evaluating and optimizing power consumption of anti-collision protocols for applications in RFID systems. In *Proc. ISLPED*, 2004.