Efficient Hardware Implementations of the Warbler Pseudorandom Number Generator

Gangqiang Yang, Mark D. Aagaard, and Guang Gong

Department of Electrical and Computer Engineering
University of Waterloo
Waterloo, Ontario, N2L 3G1, CANADA
{g37yang, maagaard, ggong}@uwaterloo.ca

Abstract. Pseudorandom number generators (PRNGs) are very important for EPC Class 1 Generation 2 (EPC C1 G2) Radio Frequency Identification (RFID) systems. A PRNG is able to provide a 16-bit random number that is used in many commands of the EPC C1 G2 standard, and it can also be used in future security extensions of the EPC C1 G2 standard, such as mutual authentication protocols between the readers and tags. In this paper, we investigate efficient ASIC hardware implementations of Warbler (a lightweight PRNG), and demonstrate that Warbler can meet the area and power consumption requirements in passive RFID systems. Warbler is built upon three nonlinear feedback shift registers (NLFSRs) and four WG-5 transformation modules. We employ two design options to implement Warbler and three different compilation methods to further optimize the area, maximum operating frequency, and power consumption. We can achieve an area of 498 GEs after the place and route phase in a CMOS 65nm ASIC, with a maximum frequency of 1430 MHz and a total power consumption of 1.239 μW at 100 KHz. Accordingly, an area of 534 GEs after the place and route phase, with a maximum frequency of 250 MHz and a total power consumption of 0.296 μW at 100 KHz can be obtained in a CMOS 130nm ASIC. Our results show that the LFSR counter-based design is better than the binary counter-based one in terms of area and power consumption. In addition, we show that the areas of WG-5 transformation look-up tables depend on the specific decimation values.

Keywords: PRNG, Lightweight, Warbler, ASICs, Passive RFID

1 Introduction

Radio Frequency Identification (RFID) is an emerging technology widely used to perform automatic and unique identification of objects. For example, the RFID applications include animal identification, passports, access control, and supply chain management. A typical RFID system includes three parts: tags, readers, and a database. Each RFID tag is issued with a unique identification number, such as the Electronic Product Code (EPC) number in the EPC Class 1 Generation 2 (EPC C1 G2) standard [7]. These RFID tags are attached to the objects, and they communicate wirelessly with the RFID readers, which interrogate the tags and get their responses. After that, the readers search for more specific information about each object in the database with a secure channel. The tiny and inexpensive properties mean that the passive EPC C1 G2 RFID tags have very limited power consumption, constrained memory, and computing capability. Thus, it is impractical to apply the traditional cryptographic primitives to such RFID tags. However, the 16-bit random number (RN16) is used in many commands of the EPC C1 G2 standard. Furthermore, the random numbers can also be used in future potential security extensions of this standard, such as the challenge-response based mutual authentication protocols [6] between the readers and tags, where both the readers and tags use the random numbers as challenges.

NIST Lightweight Cryptography Workshop 2015

Motivated by the above applications, many lightweight pseudorandom number generators (PRNGs) have been devised in recent years, such as LAMED [16], Melia-Segui *et al.* [13], Warbler [11], J3Gen [14], and AKARI1B [12]. LAMED [16] is designed based on registers, arithmetic logic unit (ALU), XOR and modular operations. Melia-Segui *et al.*'s PRNG [13] and J3Gen [14] rely on the security of linear feedback shift registers (LFSRs) and a truly random number generator (TRNG). Warbler [11] is designed by using the properties of nonlinear feedback shift registers (NLFSRs) and the WG-5 transformation modules. The estimated areas of these four PRNGs are all below 2000 GEs, the maximum area limit for resource constrained applications [2,9]. However, there have been no actual hardware implementations for them until now. AKARI1B [12] is designed based on the T-function and a non-linear filter function, and it was synthesized using the UMC Faraday 90nm technology.

The sequences generated by Warbler can pass the EPC C1 G2 standard's statistical tests as well as the NIST randomness test suite [11]. In addition, this sequence has guaranteed randomness properties, such as period and linear span. Moreover, Warbler has been shown to be resistant to the algebraic attack, cube attack, time-memory-trade-off attack and so on [11] and it can be securely used in the EPC C1 G2 RFID systems.

In this paper, we pay attention to the low-area implementation of Warbler in CMOS 65nm and CMOS 130nm ASICs, and provide the area, maximum clock frequency, and total power consumption results. The maximum clock frequency is not important for the passive RFID tags but it is useful for high performance applications. We use two different design options to implement Warbler, one based on the binary counter and the other based on the LFSR counter. The LFSR counter-based design is better than the binary counter-based one in terms of area and total power consumption.

We compare our best results with those of other lightweight primitives (Table 1). We provide the key size, IV/Block size, and internal state size for all the designs, in order to have a fair comparison. From the table, we can see that the areas from both before and after the place and route phase of Warbler are smaller than the estimated areas of LAMED, Melia-Segui *et al.*'s PRNG, and J3Gen, and also smaller than the areas of AKARI1B, Grain, Trivium, SIMON, SPECK, PHOTON-80/20/16, and SPONGENT-88. The maximum frequency of Warbler is higher than Grain's and Trivium's in CMOS 65nm. The throughput of Warbler is less than that of Grain and Trivium, but greater than that of AKARI1B, SIMON, SPECK, PHOTON-80/20/16, and SPONGENT-88. The total power consumption is related to the operating frequency and the corresponding technology. However, as we can see, the total power consumption of Warbler is very small at 100 KHz, and it is smaller than Grain's, Trivium's, and SPONGENT-88's in CMOS 130nm. Hence, it is very suitable for passive RFID applications. Even though the key size is only 45-bit, Warbler has been proved to be sufficiently secure in the passive RFID tags [11].

This paper is organized as follows. In Section 2, we describe the specification of Warbler. Section 3 first presents our metrics and design flow in CMOS 65nm and CMOS 130nm ASICs. Then, we give our ASIC architecture of Warbler, including the top-level architecture, Finite State Machine (FSM), and datapath. Later, we present our ASIC results and analysis in both CMOS 65nm and CMOS 130nm in Section 4. Finally, Section 5 concludes this paper.

2 Description of Warbler

This section gives a detailed description of Warbler.

2.1 Specification of Warbler

The following terms and notations are used to describe Warbler and its hardware architecture.

- $\mathsf{p}(x) = x^5 + x^4 + x^3 + x + 1$, a primitive polynomial of degree 5 over $\mathbb{F}_2$. It is used to generate $\mathbb{F}_{2^5}$. Let α be a primitive element of $\mathbb{F}_{2^5}$ such that $p(\alpha) = 0$.

Table 1. Comparison of Hardware Implementations of Lightweight Primitives.

Algorithms		Key Size	IV/ Block Size*	Internal State Size	Area (GEs)	Max Frequency (MHz)	Throughput @100KHz (Kbps)	Total Power	Tech (nm)	Source
PRNG	Warbler	45	20	65	**491**† **534**‡	250	20	0.296 μW @100KHz	130	**here**
	Warbler	45	20	65	**464**† **498**‡	1430	20	1.239 μW @100KHz	65	**here**
	LAMED	32	32	64	1585$^{\triangle}$	–	–	–	–	[16]
	Melia-Segui *et al.*	16	0	16	761$^{\triangle}$	–	–	–	–	[13]
	J3Gen	64	0	64	1419$^{\triangle}$	–	–	–	–	[14]
	AKARI1B	–	–	64	1749†	–	14.2	0.182 $\mu W^{\triangledown}$ @100KHz	90	[12]
Stream cipher	Grain	80	64	160	1259‡	–	100	0.78 μW @100KHz	130	[1]
	Trivium	80	80	288	2088‡	–	100	1.44 μW @100KHz	130	[1]
	Grain	80	64	160	1126‡	1020	100	2.04 mW @1020MHz	65	[17]
	Trivium	80	80	288	1986‡	962	100	3.88 mW @962MHz	65	[17]
Block cipher	SIMON$^{\diamond}$	64	32	96	523†	–	5.6	–	130	[3]
	SPECK$^{\diamond}$	64	32	96	580†	–	4.2	–	130	[3]
Hash Function	PHOTON-80/20/16	–	–	100	865†	–	2.82	–	180	[8]
	SPONGENT-88	–	–	88	738†	–	0.81	1.57 μW @100KHz	130	[4]

* IV is for PRNGs and stream ciphers, and Block is for block ciphers.
† Areas are obtained before the place and route (P & R) phase and ‡ areas are obtained after the P & R phase.
– The corresponding value is not related or not provided by the authors.
$^{\triangle}$ The estimated area.
$^{\triangledown}$ The estimated power consumption in UMC Faraday 90nm library.
$^{\diamond}$ The smallest one in the SIMON and SPECK families.

- A polynomial basis of $\mathbb{F}_{2^5}$ over $\mathbb{F}_2$ is a basis of the form $\{1, \alpha, \alpha^2, \alpha^3, \alpha^4\}$. All the computations in Warbler are calculated based on this polynomial basis.
- $\mathsf{Tr}(x) = x + x^2 + x^{2^2} + x^{2^3} + x^{2^4}$, the trace function from $\mathbb{F}_{2^5} \rightarrow \mathbb{F}_2$.
- $\mathsf{WGP\text{-}5}(x) = x + (x+1)^5 + (x+1)^{13} + (x+1)^{19} + (x+1)^{21}$, $x \in \mathbb{F}_{2^5}$. The WG-5 permutation from $\mathbb{F}_{2^5} \rightarrow \mathbb{F}_{2^5}$.
- $\mathsf{WGT\text{-}5}(x) = \mathrm{Tr}(\mathsf{WGP\text{-}5}(x)) = \mathrm{Tr}(x^{19})$, $x \in \mathbb{F}_{2^5}$. The WG-5 transformation with decimation 1 from $\mathbb{F}_{2^5} \rightarrow \mathbb{F}_2$, which is the WGT2-5 module in Figure 1.
- $\mathsf{WGT\text{-}5}(x^3) = \mathrm{Tr}(\mathsf{WGP\text{-}5}(x^3)) = \mathrm{Tr}(x^{13})$, $x \in \mathbb{F}_{2^5}$. The WG-5 transformation with decimation 3 from $\mathbb{F}_{2^5} \rightarrow \mathbb{F}_2$, which is the WGT1-5 module in Figure 1.
- NLFSR1, a nonlinear feedback shift register with length $N_1 = 18$ and it generates a *span* N_1 sequence $\mathbf{a} = \{a_i\}_{i \geq 0}$, where $a_i \in \mathbb{F}_2$. The *span n* sequence is a binary sequence with period $2^n - 1$ and each non-zero n-tuple occurs exactly once in one period [11].
- NLFSR2, a nonlinear feedback shift register with length $N_2 = 17$ and it generates a *span* N_2 sequence $\mathbf{b} = \{b_i\}_{i \geq 0}$, where $b_i \in \mathbb{F}_2$.

– NLFSR3, a nonlinear feedback shift register with length $N_3 = 6$ and it generates a sequence $\mathbf{c} = \{c_i\}_{i\geq 0}$, where $c_i \in \mathbb{F}_{2^5}$.
– $\mathsf{g}(x) = x^6+x+\gamma$, a feedback primitive polynomial of degree 6 over $\mathbb{F}_{2^5}$ for NLFSR3, and $\gamma = \alpha^{15}$.

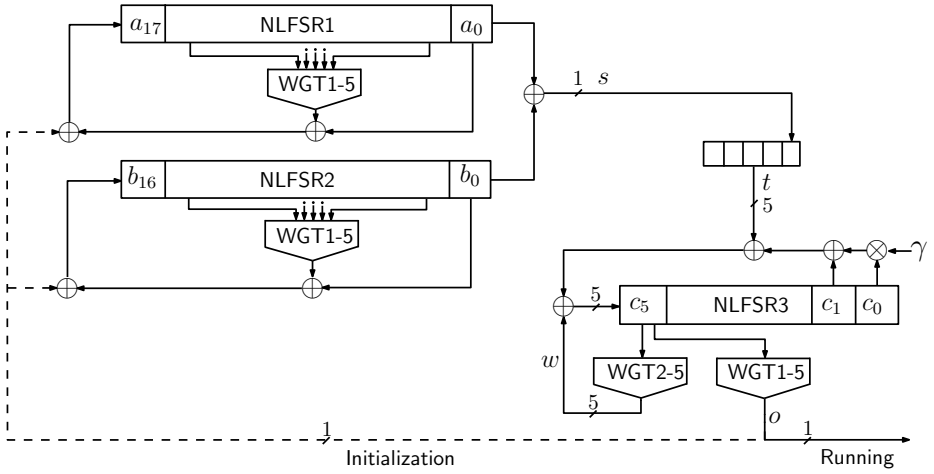

Fig. 1. Key/IV Initialization and Running Phases of Warbler

Warbler is mainly built upon three NLFSRs and four WG-5 transformation modules. It contains two phases as depicted in Figure 1: the Key and IV initialization phase and the running phase. In the initialization phase, the output of the WGT1-5 module in NLFSR3 is used to feed back to the inputs of NLFSR1 and NLFSR2. The random sequences are generated from the output of the WGT1-5 module in NLFSR3 in the running phase, and there is no feedback from WGT1-5 module in NLFSR3 to the inputs of NLFSR1 and NLFSR2 in this phase.

The nonlinear recurrence relations for NLFSR1 and NLFSR2 without the feedback from NLFSR3 are defined as follows:

$$a_{N_1+k} = a_k \oplus \mathsf{WGT\text{-}5}(x^3), \quad x = (a_{r_1+k}, a_{r_2+k}, \cdots, a_{r_5+k}) \in \mathbb{F}_{2^5}.$$
$$b_{N_2+k} = b_k \oplus \mathsf{WGT\text{-}5}(x^3), \quad x = (b_{r_1'+k}, b_{r_2'+k}, \cdots, b_{r_5'+k}) \in \mathbb{F}_{2^5}.$$

where, $k \geq 0$, $0 < r_i < N_1$, and $0 < r_i^{'} < N_2$ are the tap positions of NLFSR1 and NLFSR2 respectively. They are listed as below:

$$(r_1, r_2, r_3, r_4, r_5) = (4, 7, 8, 10, 15),$$
$$(r_1^{'}, r_2^{'}, r_3^{'}, r_4^{'}, r_5^{'}) = (4, 7, 8, 9, 12).$$

Based on the sequences $\mathbf{a} = \{a_i\}_{i\geq 0}$ and $\mathbf{b} = \{b_i\}_{i\geq 0}$, a new sequence $\mathbf{s} = \{s_i \mid s_i = a_i \oplus b_i, i \geq 0\}$ is generated and will be sent to a 5-bit shift register immediately. The element of this 5-bit shift register is used as one feedback for NLFSR3. This element can be represented as $t_k \in \mathbb{F}_{2^5},\ k \geq 0$.

The recursive relation for NLFSR3 is defined as follows:

$$c_{N_3+k} = \gamma c_k + c_{k+1} + w_k + t_k, \quad w_k = (0, 0, 0, 0, \mathsf{WGT\text{-}5}(c_{k+5})), \quad k \geq 0,$$

where, the least significant bit of w_k is generated by the WGT-5(x) module from the most significant element of NLFSR3, and the other bits of w_k are zeros.

2.2 The Behaviour of Warbler

Warbler has an internal state of 65 bits: a 45-bit Key ($K_0, K_1, K_2, \cdots, K_{44}$) and a 20-bit IV ($IV_0, IV_1, IV_2, \cdots, IV_{19}$). Initially, the Key and IV need to be loaded into the registers in NLFSR1, NLFSR2, and NLFSR3. The Key bits are loaded into the first consecutive 12, 11, and 22 positions of NLFSR1, NLFSR2, and NLFSR3 respectively. The remaining positions in each NLFSR are reserved for IV. More Specifically, the Key and IV loading process can be listed as follows.

$$\begin{aligned}
a_{11}, \cdots, a_0 &= K_{11}, \cdots, K_0, \\
a_{17}, \cdots, a_{12} &= IV_5, \cdots, IV_0, \\
b_{10}, \cdots, b_0 &= K_{22}, \cdots, K_{12}, \\
b_{16}, \cdots, b_{11} &= IV_{11}, \cdots, IV_6, \\
c_0 &= K_{27}, \cdots, K_{23}, \\
c_1 &= K_{32}, \cdots, K_{28}, \\
c_2 &= K_{37}, \cdots, K_{33}, \\
c_3 &= K_{42}, \cdots, K_{38}, \\
c_4 &= IV_{14}, IV_{13}, IV_{12}, K_{44}, K_{43}, \\
c_5 &= IV_{19}, IV_{18}, IV_{17}, IV_{16}, IV_{15}.
\end{aligned}$$

After we finish loading the Key and IV, a 36-round initialization phase is performed to mix the Key and IV properly. In this phase, the output signal o (Figure 1) from the WGT1-5 module in NLFSR3 is used as a feedback to NLFSR1 and NLFSR2 in every clock cycle. The Warbler initialization method is described as follows.

$$\text{NLFSR1}: \begin{cases} x = (a_{k+4}, a_{k+7}, a_{k+8}, a_{k+10}, a_{k+15}), \\ o_0 = 0, \\ a_{k+18} = a_k \oplus \text{WGT-5}(x^3) \oplus o_k, \quad 0 \le k \le 35. \end{cases}$$

$$\text{NLFSR2}: \begin{cases} y = (b_{k+4}, b_{k+7}, b_{k+8}, b_{k+9}, b_{k+12}), \\ o_0 = 0, \\ b_{k+17} = b_k \oplus \text{WGT-5}(y^3) \oplus o_k, \quad 0 \le k \le 35. \end{cases}$$

$$\text{5-bit shift register}: \begin{cases} s_j = 0, \quad j = 0, 1, 2, 3, \\ s_{k+4} = a_k \oplus b_k, \\ t_k = (s_k, s_{k+1}, s_{k+2}, s_{k+3}, s_{k+4}), \quad 0 \le k \le 34. \end{cases}$$

$$\text{NLFSR3}: \begin{cases} w_k = (0, 0, 0, 0, \text{WGT-5}(c_{k+5})), \\ c_{k+6} = \gamma c_k + c_{k+1} + w_k + t_k, \\ o_{k+1} = \text{WGT-5}(c^3_{k+5}), \quad 0 \le k \le 34. \end{cases}$$

The first output sequence bit o_0 from NLFSR3 is manually set to 0, which is used for the feedback from NLFSR3 to NLFSR1 and NLFSR2 in the first initialization clock cycle. The reason for this setting is that there is a 5-bit shift register between NLFSR1 and NLFSR2 together and NLFSR3. s_4 needs to take one clock cycle in order to shift into this 5-bit shift register. Therefore, NLFSR3 needs to wait for one clock cycle until the first element t_0 in the 5-bit shift register is ready for the feedback computation of NLFSR3. As a result, in the initialization phase, the NLFSR1 and NLFSR2 run for 36 clock cycles, and the NLFSR3 runs only for 35 clock cycles.

After the NLFSRs finish the initialization phase, they simultaneously go to the running phase, where the following 5-bit element t_k is used as a feedback element for NLFSR3.

$$t_k = (s_{5(k-27)-1}, s_{5(k-27)}, s_{5(k-27)+1}, s_{5(k-27)+2}, s_{5(k-27)+3}) \in \mathbb{F}_{2^5}, \quad k \ge 35.$$

t_k can be obtained by every five clock cycles from the 5-bit shift register, which results in a $1/5$ (i.e., 1-bit per five clock cycles) throughput of the Warbler output sequence $o_{k+1},\ k \geq 35$.

3 ASIC Implementation

We discuss the ASIC implementation of Warbler in this section, including the design flow and metrics, and the specific architecture.

3.1 Design Flow and Metrics

We use Synopsys Design Compiler Version D-2010.03-SP4 to synthesize the designs into netlist, based on the STMicroelectronics CMOS 65nm CORE65LPLVT_1.20V and IBM CMOS 130nm CMR8SF-LPLVT Process SAGE v2.0 standard cell library, with both having a typical 1.2V voltage and 25°C temperature. Cadence SoC Encounter v09.12-s159_1 is used to finish the place and route phase in order to generate the layout of the designs. We use Mentor Graphics ModelSim SE 10.1a to conduct functional simulation of the designs and perform timing simulation by using the timing delay information generated from SoC Encounter as well. The area of the design after the logic synthesis is provided for comparisons with previous designs, and a more accurate area after place and route is also provided for deploying the designs in practical cases. The densities used for the place and route phase for CMOS 130nm and 65nm are 0.92 and 0.93 respectively, in order to make a trade-off between area and maximum operating frequency when the densities are pretty high. We choose them because the area after the place and route phase will decrease when the density is higher. However, the corresponding critical path will increase; leading to potential DRC (Design Rule Check) and LVS (Layout Versus Schematic) violations. As usual, the area is measured in gate equivalents (GEs), and one GE is equivalent to the physical area required for the two-input one-output NAND gate with the lowest driving strength of the corresponding technology. The areas of one GE are 2.08 $(\mu m)^2$ and 5.76 $(\mu m)^2$ for ST CMOS 65nm and IBM CMOS 130nm respectively.

We use SoC Encounter v09.12-s159_1 to generate accurate power consumption based on the activity information generated from the timing simulation with a frequency of 100 KHz, and a duration time of 0.1s. We do so because the 100 KHz clock frequency is widely used for benchmarking in resource constrained applications and 0.1s is long enough to provide accurate activity information for all the signals. Moreover, the maximum clock frequency which can be operated for a specific design is obtained by using the critical path after the place and route phase.

In order to be fair enough to compare our results with the related work [3,5,15], we provide the areas of some basic gates in our specific libraries. All the areas of basic gates provided here are the smallest one in the library, and we normalize the two-input one-output NAND gate to be 1. The specific basic gates in our IBM CMOS 130nm and ST CMOS 65nm libraries are listed in Table 2.

Table 2. The Areas of Basic Gates in the Libraries

	NAND	AND	OR	NOT	XOR	XNOR	2-1 MUX	DFF	1-bit Full Adder	Scan FF
IBM CMOS 130nm	1	1.25	1.25	0.75	2	2	2.25	4.25	5.75	5.5
ST CMOS 65nm	1	1.25	1.5	0.75	2.25	2.25	2	3.75	4.5	4.75

3.2 ASIC Architecture

In this subsection, we target a low-area implementation of Warbler but still maintain a very high maximum operating frequency. We first provide the top-level architecture of Warbler and then present the architectures of FSM and datapath.

Entire Architecture

We provide a top-level architecture for Warbler in Figure 2, and the entire architecture includes two parts: FSM and datapath. FSM is used to provide state transition signals (Load, Init, Run) and the register chip-enable signal (NLFSR_ce3) for the datapath. The datapath is used to load the initial data for the registers (Section 2.2) using the input ports (d1, d2, d3), to process the internal sates in the registers, and then to output the sequence (o_Warbler) and the valid signal (o_valid).

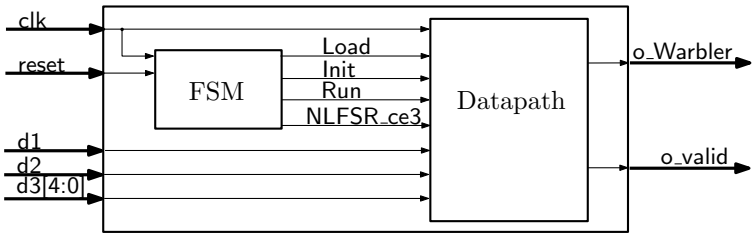

Fig. 2. The Top-level Architecture of Warbler

FSM

Our architecture has three states: loading, initialization, and running. The loading state takes 18 clock cycles, the initialization state lasts for 36 clock cycles, and the running state lasts forever unless Warbler is reset. Specifically, our FSM goes into loading state immediately when reset equals 1. Warbler reads the initial data from d1 into the datapath once reset goes to 0 again, and it reads data from d2 and data from d3 in the 2th and 13th clock cycles of the loading state respectively. Once the loading state is finished, the initialization and running states will run.

From the description of Warbler in Section 2, we know that the throughput of Warbler is $1/5$, because the Shift5 module takes five clock cycles for one feedback computation of NLFSR3 in the running state. Therefore, a register chip-enable signal (NLFSR_ce3) is required to control NLFSR3 in order to output the sequence correctly. However, NLFSR1 and NLFSR2 always run after reset, which makes them use only the standard registers without chip-enable signals. This property reduces the Warbler's area. Furthermore, it also explains why we have only one chip-enable signal in our architecture in Figure 2.

Recently, LFSR based counters have been used to replace the binary counter in the FSM in hardware implementations [10], because they only contain several registers and some combinational feedback logic without using a full-adder. In general, the combinational logic of the LFSR counter is smaller than the full-adder of the binary counter in terms of area. Therefore, we can achieve some hardware benefits by using the LFSR counter to replace the binary counter. The binary and LFSR counter-based designs are both provided in this paper. To design our LFSR counter, we use a primitive polynomial ($X^6 + X + 1$) with an initial value (1, 1, 1, 1, 1, 1).

We use one-hot encoding for the three states: loading (100), initialization (010), and running (001). For the binary counter-based design, the counter starts from 0 in each state. Similarly, the counter starts

from 63 in each state for the LFSR counter-based design. The states transition conditions for these two designs are summarized in Table 3. The Load signal stays at 1 when the FSM is in the loading state; otherwise, it equals 0. The similar case for the Init and Run signals.

Table 3. States Transition Conditions for FSM

States	Binary counter-based	LFSR counter-based
Loading (100) → initialization (010)	17	17
Initialization (010) → running (001)	35	39

The chip-enable signal (NLFSR_ce3) is generated as follows: it is set to 1 in the loading state, 1 in the initialization state except the first clock cycle, and $(0, 0, 0, 0, 1)$ in every five clock cycles in the running state.

Datapath

The datapath for Warbler in our ASIC architecture is shown in Figure 3. It includes five parts: NLFSR1, NLFSR2, NLFSR3, Shift5, and o_valid. NLFSR1 contains a 18-stage register, a WGT1-5 module, and other feedback logic. Similarly, NLFSR2 contains a 17-stage register, a WGT1-5 module, and other feedback logic. NLFSR3 contains a 6-stage register, a Gamma_Mult module, a WGT1-5 module, a WGT2-5 module, and other feedback logic. Shift5 is used for the 5-bit shift register and is comprised of a 5-stage register and other combinational logic. Moreover, o_valid provides a valid signal for the Warbler output sequence.

According to Section 2.2, the feedback values vary for different states. Therefore, Init, Load, and NLFSR_ce3 are used to select correct feedback values for NLFSRs in each state. Furthermore, the NLFSR_ce3 is used to control the throughput of output sequence (o_Warbler) and the output valid signal (o_valid). The datapath works with the FSM together to generate a pseudorandom sequence correctly.

The Gamma_Mult module is used for the calculation of γc_k in $\mathbb{F}_{2^5}$. Under the polynomial basis representation, the element $X \in \mathbb{F}_{2^5}$ ($X = x_0 + x_1\alpha + x_2\alpha^2 + x_3\alpha^3 + x_4\alpha^4$) multiplied by $\gamma = \alpha^{15}$ can be computed as follows:

$$\begin{aligned} X \cdot \alpha^{15} &= (x_0 + x_1\alpha + x_2\alpha^2 + x_3\alpha^3 + x_4\alpha^4) \cdot \alpha^{15} \\ &= x_0\alpha^{15} + x_1\alpha^{16} + x_2\alpha^{17} + x_3\alpha^{18} + x_4\alpha^{19} \\ &= (x_2 + x_4) + (x_2 + x_3 + x_4)\alpha + (x_0 + x_3 + x_4)\alpha^2 + \\ &\quad (x_0 + x_1 + x_2)\alpha^3 + (x_1 + x_3 + x_4)\alpha^4. \end{aligned}$$

Therefore, the result of $X \cdot \gamma$ is represented as a 5-bit vector $(x_2 \oplus x_4, x_2 \oplus x_3 \oplus x_4, x_0 \oplus x_3 \oplus x_4, x_0 \oplus x_1 \oplus x_2, x_1 \oplus x_3 \oplus x_4)$. Thus, we can implement our Gamama_Mult module using the finite field logic directly.

Similarly, we can compute WGT-5(x^3) and WGT-5(x) in polynomial basis for every $x \in \mathbb{F}_{2^5}$ by using the finite field logic directly or pre-storing them to two look-up tables (WGT1-5 and WGT2-5 respectively), as in [17]. However, the hardware implementations of WGT1-5 module (with decimation 3) and WGT2-5 module (with decimation 1) are more efficient if the look-up table method * is used rather than the finite field logic methods [17]. Therefore, we use the look-up table methods for implementing the WGT1-5 and WGT2-5 modules.

* The look-up table method here and the following refers to using the hardware tools to optimize the pre-stored tables of the corresponding modules to the logic but not to using the RAM directly.

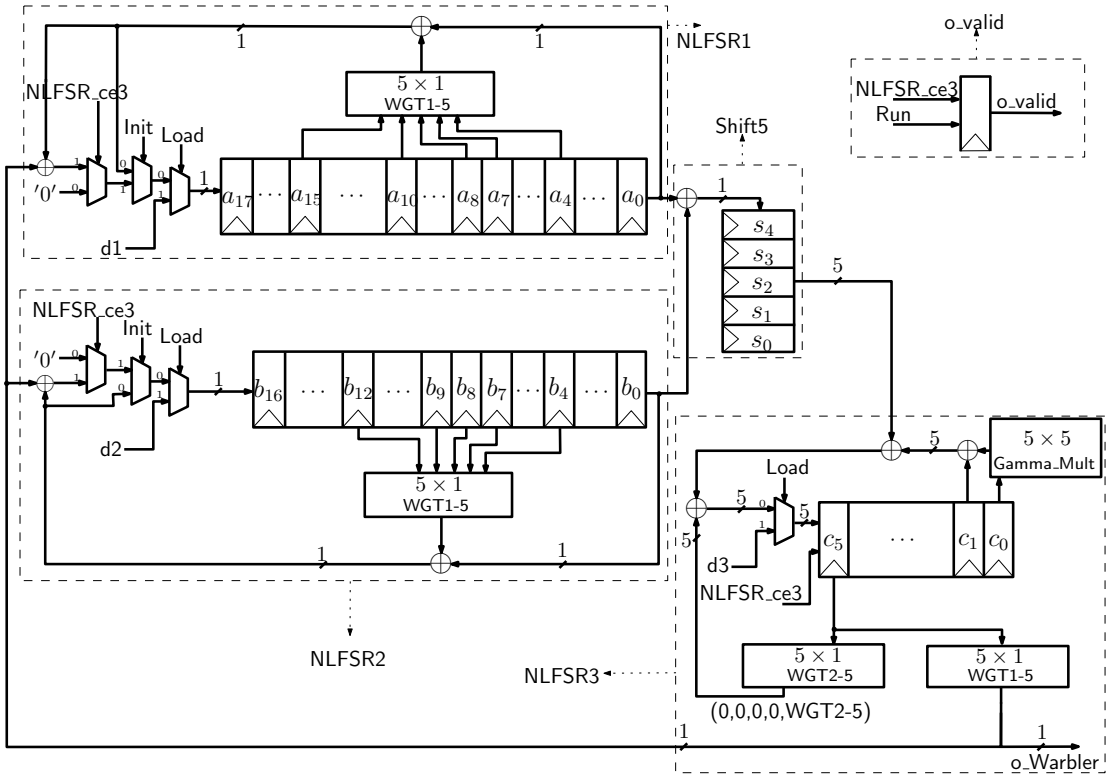

Fig. 3. Datapath of Warbler

4 ASIC Results and Analysis

In this section, we first give the ASIC implementation results of Warbler, using our architecture in CMOS 65nm and CMOS 130nm. Then, we provide a comprehensive analysis of the area that use different technologies and various compilation techniques.

4.1 ASIC Results

We use three different compilation techniques in Design Compiler to perform hardware optimizations: simple compile, compile ultra, and compile ultra with clock gating. The simple compile options can provide us with the hierarchal architectures of the design, and the area of specific submodules. The compile ultra option can make deeper optimizations by optimizing the entire module together, thereby reducing the area and power consumption significantly [5,10] . The clock gating technique can further reduce the area and power consumption [5].

Table 4 shows the ASIC implementation results of Warbler by using our architecture in CMOS 130nm and CMOS 65nm. The metric in our ASIC optimization is the low-area implementation, while still maintain a very high maximum frequency. In general, when the clock period constraint in the Design Compiler is very small, we can get a circuit with higher maximum frequency but also bigger area. However, in order to get a low-area implementation, we make a trade-off between the area and the maximum frequency by setting the clock period constraint loosely. In this way, we sacrifice a little bit maximum frequency because it is, generally, high enough for the passive RFID tags. The area results from before and after the place and route phase are both provided by using compile ultra and clock gating

Table 4. Our ASIC Implementation Results of Warbler in CMOS 65nm and CMOS 130nm.

Warbler	**Technology**	**Area** (GEs)		**Max Frequency** (MHz)	**Throughput** @100KHz (Kbps)	**Total Power** @100KHz (μW)	**Optimality** (MHz/#GEs)
		Before P&R	After P&R				
Binary counter-based	65nm	$475^{\dagger}$	$511^{\dagger}$	1370	20	1.274	2.68
LFSR counter-based	65nm	$464^{\dagger}$	$498^{\dagger}$	1430	20	1.239	2.87
Binary counter-based	130nm	$500^{\dagger}$	$543^{\dagger}$	270	20	0.298	0.50
LFSR counter-based	130nm	$491^{\dagger}$	$534^{\dagger}$	250	20	0.296	0.47

† Area obtained by using compile ultra and clock gating techniques.

techniques with a purpose of comparison with other designs. The corresponding maximum frequency is given by using the critical path. The throughput of Warbler is $1/5$, as mentioned in Section 2. Therefore, it is 20 Kbps at 100 KHz, the typical frequency for benchmarking lightweight primitives. Similarly, the total power consumption is provided at 100 KHz. Since the operating frequency is so small, the static power consumption dominates the total power consumption. However, the static power consumption is larger in CMOS 65nm than in CMOS 130nm. Therefore, the total power consumption of Warbler in CMOS 65nm is larger than in CMOS 130nm. In addition, the maximum frequency and total power consumption are both obtained after the place and router phase in order to be closer to the practical cases. The optimality is given based on our trade-off strategy.

From Table 4, we can see that the LFSR counter-based design is smaller than the binary counter-based design (i.e., 11 GEs and 9 GEs smaller in CMOS 65nm and CMOS 130nm respectively, for the areas before the place and route). Similarly, the total power consumption of the LFSR counter-based design is smaller than that of the binary counter-based design in both CMOS 65nm and CMOS 130 nm. The optimality for the LFSR counter-based design is higher than that for the binary counter-based design in CMOS 65nm, but opposite in CMOS 130nm.

Our another observation is that the sequential logic dominates the entire area. In the area before the place and route phase, for example, the proportion of sequential logic (Table 5), depends on the adopted technologies and compilation techniques. However, they are all above 65%.

Table 5. The Sequential Logic Ratios of Warbler

Warbler	**CMOS 65nm**			**CMOS 130nm**		
	Compile simple	Compile ultra	Compile ultra + clock gating	Compile simple	Compile ultra	Compile ultra + clock gating
Binary counter-based	65.0%	66.7%	65.0%	70.0%	71.5%	72.4%
LFSR counter-based	65.6%	67.4%	66.7%	71.3%	73.2%	73.4%

4.2 Results Analysis

In order to thoroughly analyze the constitution of the area of Warbler in CMOS 65nm and CMOS 130nm, we break down the entire area from before the place and route phase into separate submodules, as shown in Table 6. It is worth noting that the following analysis is based on the areas of the submodules which are obtained by using the compile simple technique.

The areas of the datapath for the binary counter-based and the LFSR counter-based designs are the same, and only the areas of the FSM are different. The LFSR counters are 17.4% and 23.7% smaller than the binary counters in CMOS 65nm and CMOS 130nm respectively. However, the total areas of the FSM

Table 6. Breakdown of the Implementation Results of Warbler before the Place and Route Phase

Components			CMOS 65nm		CMOS 130nm	
			Binary counter-based (GEs)	**LFSR counter-based** (GEs)	**Binary counter-based** (GEs)	**LFSR counter-based** (GEs)
FSM	State transitions + chip_enable logic		35.25	39.00	36.75	37.50
	Counter		47.50	39.25	51.75	39.50
Datapath	NLFSR1	18-stage register	67.50		76.50	
		WGT1-5	15.50		14.50	
		Other feedback logic	10.00		9.50	
	NLFSR2	17-stage register	63.75		72.25	
		WGT1-5	15.50		14.50	
		Other feedback logic	10.00		9.50	
	NLFSR3	6-stage register	150.00		180.00	
		Gamma_Mult	13.75		14.00	
		WGT1-5	15.50		14.50	
		WGT2-5	9.25		9.50	
		Other feedback logic	32.50		30.75	
	Shift5	5-stage register	18.75		21.25	
		Other combinational logic	8.00		8.00	
	O_valid		6.50		6.50	
Totals	Compile simple		519	515	570	558
	Compile ultra		505	493	555	541
	Compile ultra + clock gating		475	464	500	491

are only 5.5% and 13% smaller accordingly, due to the area of state transitions and chip-enable logic is bigger in the LFSR counter-based FSM than in the binary counter-based FSM.

For the registers in NLFSR1, NLFSR2, and Shift5, we can verify that they are indeed implemented by the standard registers without chip-enable signals. For example, the area of this type of register in CMOS 65nm is 3.75 GEs (Table 2), and the total area is 67.50 GEs for the 18-stage register in NLFSR1. This area is confirmed with our architecture as described in Section 3.2. The areas of combinational logic, such as the WGT1-5 module, depend on the areas of basic gates in the different technologies. For example, the areas of the WGT1-5 module are 15.50 GEs and 14.50 GEs in CMOS 65nm and CMOS 130nm respectively. The same situation exists for other submodules.

For the area of NLFSR3, the 6-stage register use registers with chip-enables, and moreover the WGT1-5 and WGT2-5 modules are different in the same technology (i.e., 15.50 GEs and 9.25 GEs respectively in CMOS 65nm, and 14.50 GEs and 9.50 GEs respectively in CMOS 130nm). We give the specific contents of these two look-up tables in Table 7. As we can see, the position distributions of 1 and 0 are different for the WGT1-5 and WGT2-5 modules, and they are computed based on WG-5 transforms with decimation 3 and 1 respectively. The synthesis tool is able to optimize WGT2-5 to the simpler logic in hardware than WGT1-5; therefore, the area of WGT1-5 is bigger than WGT2-5's as shown in Table 6. The distinct areas of different WG-5 transformation tables give us the worthwhile idea to select a decimation value in order to make the WG-5 transformation table as small as possible for a new design.

As shown in Table 6, the compile ultra and compile ultra plus clock gating techniques can indeed reduce the area. Table 8 shows the area reduction percentages by using compile ultra, and compile ultra plus clock gating techniques, compared to the compile simple technique. We can see that compile ultra

Table 7. The WGT1-5 and WGT2-5 Look-up Tables

Address	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
WGT1-5	0	0	1	0	0	1	1	1	0	1	1	1	1	0	0	0	1	0	1	0	0	0	1	1	0	1	1	0	0	1	0	1
WGT2-5	0	0	0	0	0	1	0	0	0	0	0	1	1	1	1	1	1	1	0	1	1	1	0	0	1	0	0	1	1	1	0	1

can reduce the area by at least 2.6%, and compile ultra plus clock gating can reduce the area by at least 8.5% for our Warbler designs.

Table 8. The Area Reduction Percentages by using Compile Ultra and Compile Ultra plus Clock Gating

Techniques	**CMOS 65nm**		**CMOS 130nm**	
	Binary counter-based	**LFSR counter-based**	**Binary counter-based**	**LFSR counter-based**
Compile ultra	2.7%	4.3%	2.6%	3.0%
Compile ultra + clock gating	8.5%	9.9%	12.3%	12.0%

5 Concluding Remarks

In this paper, we have presented hardware implementations of Warbler in CMOS 65nm and CMOS 130nm ASICs. We proposed an architecture that takes advantage of standard registers without chip-enable signals. In addition, we investigated two methods for designing the FSM: binary counter-based and LFSR counter-based. We used three different compilation techniques to optimize our designs. We can achieve the areas of 498 GEs and 534 GEs after the place and route phase in CMOS 65nm and CMOS 130nm respectively. The corresponding maximum frequencies are 1430 MHz and 250 MHz respectively, for CMOS 65nm and CMOS 130nm. The power consumption of Warbler is very small at 100 KHz: only 1.239 μW and 0.296 μW respectively, for CMOS 65nm and CMOS 130nm. From the ASIC results, we have determined that the LFSR counter-based design is better than the binary counter-based design in terms of smaller area and lower total power consumption. In addition, the sequential logic ratios for all our designs are larger than 65% for both CMOS 65nm and CMOS 130nm. Our analysis has verified that the areas of NLFSRs and combinational logic are dependent upon the type of registers and the adopted technologies. The area of the WG-5 transformation table depends upon the selected decimation value, giving us some suggestions for future ciphers and pseudorandom number generator designs using WG-5 transformations. When compared with other lightweight primitives, the area of our Warbler implementation is smaller than the estimated areas of LAMED, Melia-Segui *et al.*'s PRNG, and J3Gen, and also smaller than the areas of AKARI1B, Grain, Trivium, SIMON, SPECK, PHOTON-80/20/16, and SPONGENT-88. In conclusion, Warbler can fit into passive RFID systems.

References

1. M. D. Aagaard, G. Gong, and R. K. Mota. Hardware Implementations of the WG-5 Cipher for Passive RFID Tags. In *6th IEEE International Symposium on Hardware-Oriented Security and Trust*, pages 24–29, June 2013.
2. F. Armknecht, M. Hamann, and V. Mikhalev. Lightweight Authentication Protocols on Ultra-Constrained RFIDs-Myths and Facts. In *Radio Frequency Identification: Security and Privacy Issues*, pages 1–18. Springer, 2014.

3. R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers. The SIMON and SPECK Families of Lightweight Block Ciphers. Cryptology ePrint Archive, Report 2013/404, 2013. http://eprint.iacr.org/.
4. A. Bogdanov, M. Knežević, G. Leander, D. Toz, K. Varıcı, and I. Verbauwhede. SPONGENT: A Lightweight Hash Function. In *Cryptographic Hardware and Embedded Systems–CHES 2011*, pages 312–325. Springer, 2011.
5. C. De Canniere, O. Dunkelman, and M. Knežević. KATAN and KTANTAN: A Family of Small and Efficient Hardware-oriented Block Ciphers. In *Cryptographic Hardware and Embedded Systems-CHES 2009*, pages 272–288. Springer, 2009.
6. D. Engels, X. Fan, G. Gong, H. Hu, and E. M. Smith. Ultra-lightweight Cryptography for Low-cost RFID Tags: Hummingbird Algorithm and Protocol. *Centre for Applied Cryptographic Research (CACR) Technical Reports*, 29, 2009.
7. EPCglobal. EPC Radio Frequency Identity Protocols Class-1 Generation-2 UHF RFID Protocol for Communications at 860 MHZ - 960 MHz. http://www.gs1.org/sites/default/files/docs/uhfc1g2/uhfc1g2_2_0_0_standard_20131101.pdf, 2013.
8. J. Guo, T. Peyrin, and A. Poschmann. The PHOTON Family of Lightweight Hash Functions. In *Advances in Cryptology–CRYPTO 2011*, pages 222–239. Springer, 2011.
9. A. Juels and S. A. Weis. Authenticating Pervasive Devices with Human Protocols. In *Advances in Cryptology–CRYPTO 2005*, pages 293–308. Springer, 2005.
10. L. Knudsen, G. Leander, A. Poschmann, and M. J. Robshaw. PRINTcipher: A Block Cipher for IC-printing. In *Cryptographic Hardware and Embedded Systems, CHES 2010*, pages 16–32. Springer, 2010.
11. K. Mandal, X. Fan, and G. Gong. Warbler: A Lightweight Pseudorandom Number Generator for EPC C1 Gen2 Tags. In *Radio Frequency Identification System Security: RFIDsec'12 Asia Workshop Proceedings*, page 73. IOS Press, 2013.
12. H. Martin, E. San Millán, P. Peris-Lopez, and J. E. Tapiador. Efficient ASIC Implementation and Analysis of Two EPC C1 G2 RFID Authentication Protocols. *Sensors Journal, IEEE*, 13(10):3537–3547, 2013.
13. J. Melia-Segui, J. Garcia-Alfaro, and J. Herrera-Joancomarti. Analysis and Improvement of a Pseudorandom Number Generator for EPC Gen2 Tags. In *Financial Cryptography and Data Security*, pages 34–46. Springer, 2010.
14. J. Melià-Seguí, J. Garcia-Alfaro, and J. Herrera-Joancomartí. J3Gen: A PRNG for Low-cost Passive RFID. *Sensors*, 13(3):3816–3830, 2013.
15. A. Moradi, A. Poschmann, S. Ling, C. Paar, and H. Wang. Pushing the Limits: A Very Compact and a Threshold Implementation of AES. In *Advances in Cryptology–EUROCRYPT 2011*, pages 69–88. Springer, 2011.
16. P. Peris-Lopez, J. C. Hernandez-Castro, J. M. Estevez-Tapiador, and A. Ribagorda. LAMED–A PRNG for EPC Class-1 Generation-2 RFID Specification. *Computer Standards & Interfaces*, 31(1):88–97, 2009.
17. G. Yang, X. Fan, M. Aagaard, and G. Gong. Design Space Exploration of the Lightweight Stream Cipher WG-8 for FPGAs and ASICs. In *Proceedings of the Workshop on Embedded Systems Security*, page 8. ACM, 2013.