Chai-tea, Cryptographic Hardware Implementations of xTEA

Jens-Peter Kaps

Volgenau School of IT&E, George Mason University, Fairfax, VA, USA
jkaps@gmu.edu

Abstract. The tiny encryption algorithm (TEA) was developed by Wheeler and Needham as a simple computer program for encryption. This paper is the first design-space exploration for hardware implementations of the extended tiny encryption algorithm. It presents efficient implementations of XTEA on FPGAs and ASICs for ultra-low power applications such as RFID tags and wireless sensor nodes as well as fully pipelined designs for high speed applications. A novel ultra-low power implementation is introduced which consumes less area and energy than a comparable AES implementation. Furthermore, XTEA is compared with stream ciphers from the eSTREAM portfolio and lightweight ciphers. The high speed implementations of XTEA operate at 20.6 Gbps (FPGA) or 36.6 Gbps (ASIC).

Keywords: Efficient implementation, symmetric key algorithms, TEA, XTEA, FPGA, ASIC

1 Introduction

The Tiny Encryption Algorithm (TEA) was introduced by David Wheeler and Roger Needham [1, 2] in '94. Their main design goal was to produce a cipher that is simple, short and does not rely on large tables or pre-computations. Shortly after TEA was published, a few minor weaknesses were found [3]. The original authors eliminated those weaknesses in a new version of TEA called XTEA for extended TEA [4]. A positive side effect of the new version, also called *tean*, is that the main routine requires two fewer addition operations which results in a faster algorithm. Also in [4] the authors described a variant of TEA called *Block TEA* to cater for larger block sizes than the original algorithm's 64-bit. An attack on Block TEA was found and subsequently corrected in [5]. Other recent attacks [6, 7] target a reduced round version of TEA. The recommended 32 round version is still considered to be secure.

TEA uses only simple addition, XOR and shifts, and has a very small code size. This makes TEA an ideal candidate to provide data security services for wireless sensor network (WSN) nodes which have limited memory and computational power. Many software implementations of TEA for these nodes have been reported [8–10]. Kanamori compared software implementations of the Advanced

Encryption Standard (AES) to implementations of TEA on sensor nodes [11] and concluded that the memory requirements for TEA are one quarter the requirements of AES.

Even though TEA was proposed mainly for software implementations, its simple design makes it also very suitable for hardware implementations. The implementation designers can take advantage of the inherent parallelism of TEA to boost performance or, on the other hand, reduce its area and power consumption. This makes it possible to implement TEA for severe power constraint applications such as passive radio frequency identification (RFID) tags and the next generation WSN nodes. Customized hardware is currently the only option for inexpensive passive RFID tags as they do not have a processor. The same might become true for next generation sensor nodes which are predicted to be powered by energy scavenged from the environment [12].

For environments in which the power consumption is not the most important design criteria, field programable gate arrays (FPGAs) are an interesting option. FPGAs can be re-programmed in the field which makes it possible to update the cryptographic algorithm after deployment. This might be necessary if new cryptanalytic results lead to changes in the TEA algorithm as we have seen in the past.

Hardware implementations of the original TEA were first published in [13, 14] targetting RFID tags. To the knowledge of the authors, this is the first design space exploration of hardware implementations of XTEA. This paper covers implementations for application specific integrated circuits (ASIC) as well as for FPGAs, small, power and energy efficient implementations for ubiquitous computing devices with stringent power constraints and high speed implementations for server environments.

2 Extended Tiny Encryption Algorithm

The Extended Tiny Encryption Algorithm (XTEA) is a block cipher that uses a cryptographic key of 128 bits to encrypt or decrypt data in blocks of 64 bits. Each input block is split into two halves y and z which are then applied to a routine similar to a Feistel network for N rounds where N is typically 32. Most Feistel networks apply the result of a mixing function to one half of the data using XOR as a reversible function. XTEA uses for the same purpose integer addition during encryption and subtraction during decryption.

Figure 1 shows the C source code for XTEA as it was introduced in [4]. Additional parenthesis were added to clarify the precedence of the operators. The main variables y, z, and sum, which assists with the subkey generation, have a length of 32 bits. All additions and subtractions within XTEA are modulo 2^{32}. Logical left shifts of z by 4 bits are denoted as $z \ll 4$ and logical right shift by 5 bits as $z \gg 5$. The bitwise XOR function is denoted as "^" in the source code and $\oplus$ in this paper.

The first part of the algorithm is the encryption routine and the second part is the decryption routine. The *while*-loop constitutes the round function.

```
int tean(long * v, long * k, int N)
{
  unsigned long y=v[0], z=v[1], DELTA=0x9e3779b9;
  if (N>0) {    /* encryption */
    unsigned long limit=DELTA*N, sum=0;
    while (sum != limit) {
      y += ((z<<4 ^ z>>5) + z) ^ (sum + k[sum&3]);
      sum += DELTA;
      z += ((y<<4 ^ y>>5) + y) ^ (sum + k[sum>>11 &3]);
    }
  } else {    /* decryption */
    unsigned long sum=DELTA*(-N);
    while (sum) {
      z -= ((y<<4 ^ y>>5) + y) ^ (sum + k[sum>>11 &3]);
      sum -= DELTA;
      y -= ((z<<4 ^ z>>5) + z) ^ (sum + k[sum&3]);
    }
  }
  v[0]=y, v[1]=z;
  return;}
```

Fig. 1. Source code of XTEA

The formulae that compute the new values for y and z can be split into a permutation function $f(z) = (z \ll 4 \oplus z \gg 5) + z$ and a subkey generation function $sum + k(sum)$. The function $k(sum)$ selects one block out of the four 32-bit blocks that comprise the key, depending on either bits 1 and 0 or bits 12 and 11 of sum. The results of the permutation function and the subkey generation function are XORed and then applied to y and z respectively, by addition in the case of encryption or subtraction in the case of decryption.

This leads to the simplified block diagram shown in Fig. 2. For encryption, z is applied to the left side, y to the right side, and all adder/subtracters are in addition mode. For decryption, the opposite is applied. The permutation function is shown as $\boxed{\mathsf{f}}$ and the subkey generation as $\boxed{\mathsf{Keygen}}$. One round of xtea computes a new value for y and z. Therefore, we can view the computation of one value as a *halfround*. A new value for sum is computed between the first and the second halfround. It is incremented by a constant Δ during encryption and decremented during decryption. We included this computation into the first halfround as it can be performed concurrently with the final addition/subtraction of the data in this halfround.

3 Speed XTEA

Speed XTEA investigates how XTEA scales for fast hardware implementations. We implemented a fully loop-unrolled, pipelined architecture of XTEA on FPGA and ASIC. Each block of data can be associated with a new key and the mode can

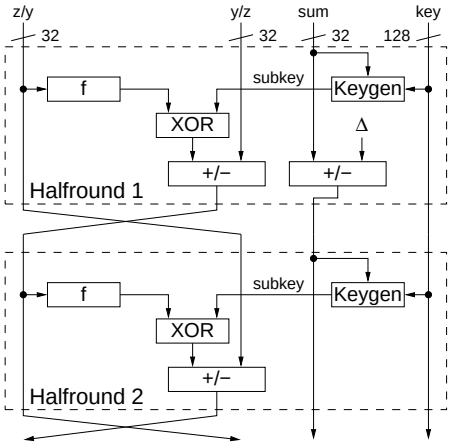

Fig. 2. Top level block diagram of XTEA

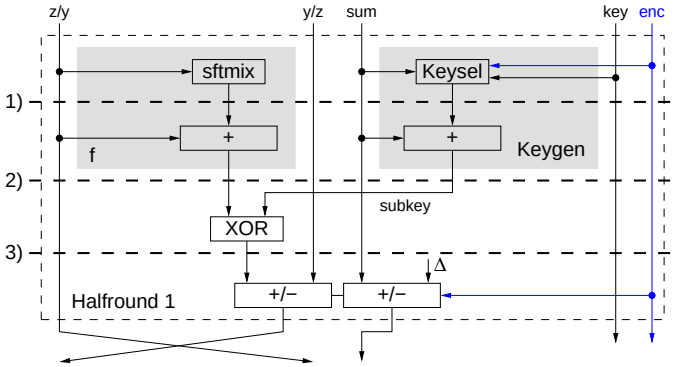

Fig. 3. Pipeline cuts for half rounds

be switched between encryption and decryption without loss of throughput. A natural location for the pipeline cuts is between the half rounds, so called outer round pipelining. However, due to the long delay of one halfround additional pipeline cuts within the halfround, called inner round pipelining, are necessary. Figure 3 indicates through thick dashed lines the location of three inner round cuts that we want to investigate.

3.1 FPGA Implementation

FPGAs are composed of configurable logic blocks (CLB) and a programmable interconnection network. We targeted the low cost Xilinx FPGA series Spartan 3 to compare our results with other reported block cipher implementations on the

same series. In addition, we implemented XTEA on the Xilinx Virtex 5 devices which are a natural choice for high speed applications. Xilinx divides each CLB into slices. Each slice on a Spartan-3 contains two sets of a 4-input look-up table (LUT) followed by a flip-flop. The Virtex 5 slice contains four sets of a 6-input LUT followed by a flip-flop. Slices in both families have dedicated circuitry and connections for fast carry propagation. Therefore, the result of additions can be stored within the same slices avoiding any additional wire delay. This makes this location ideal for a pipeline cut which is indicated by line number 2 in Fig. 3. Furthermore the LUT contained in these slices can accommodate the function sftmix and XOR.

Unfortunately the ideal clock speed of 3.18 ns on Spartan 3 (speed grade -5) and 1.00 ns on Virtex 5 (speed grade -3) cannot be reached due to wire delays at the input of the adder. Even though pipeline location 2 seems to be sufficient at first glance, the key select function Keysel $k(sum)$ can not be implemented with a single LUT per bit. Each output bit depends on 9 input bits: The encryption / decryption flag, four bits from sum, and four bits from the key. In addition, we noticed through timing analysis that the routing delay from the input of the half round to the inner round pipeline cut number 2 was much larger than the delay from the inner cut to the following outer round pipeline cut. Hence, we placed an additional pipeline cut, indicated as line number 1 in Fig. 3, before the first stage of adders.

Pipeline cut three is mainly of interest for ASIC implementation. The results for FPGA show that it has no effect on the critical path delay because the XOR can be performed in the LUTs of the slices occupied by the following adder.

We implemented variations of these possible cuts in a not loop-unrolled version of XTEA to analyze their effect on the delay and to estimate the throughput-area ratio of a fully unrolled implementation. Table 1 shows the results including an approximation of the ratio of delay due to logic cells "Logic Delay" versus routing delay. The throughput and the throughput area ratio are estimated for a fully unrolled design with the same delay and 32 times the area.

Placing pipeline cuts in either location 2 or locations 1 & 2 yields the smallest critical path delay and the best estimated throughput area ratio. We implemented two loop-unrolled pipelined version of XTEA called SpeedXTEA-1 with pipeline cut 2 and SpeedXTEA-2 with pipeline cuts 1 and 2. It is interesting to note that the 6-input LUTs of the Virtex 5 have no significant advantage over the 4-input LUTs on Spartan 3 in terms of critical path delay. The large speed improvement of Virtex 5 is mainly due to its newer CMOS technology.

3.2 ASIC Implementation

Our ASIC implementations of XTEA are based on the SpeedXTEA-1 and SpeedXTEA-2 FPGA designs and use the same pipeline cuts. SpeedXTEA-3 makes use of the pipeline cuts 1, 2, and 3. This takes the XOR as well as sftmix and keysel out of the critical path which is now determined solely by the delay of the adders. We used Brent-Kung [15] adders which are more than three times

Table 1. Results for pipeline cuts in Halfround 1 & 2 Implementations

FPGA	xc3s2000-5			xc5vlx100-3			
Pipeline Cuts	1	2	1&2	1	2	1&2	1&2&3
Critical Path Delay (ns)	8.200	7.372	5.471	3.097	2.999	2.193	2.199
Logic Delay	67%	53%	71%	62%	51%	64%	65%
Routing Delay (ns)	2.71	3.46	1.59	1.18	1.47	0.79	0.77
Clock Cycles for one round	4	4	6	4	4	6	8
Area including overhead (Slices)	597	577	701	410	383	445	522
Estimated Troughput (Mbps)	7,805	8,681	11,698	20,665	21,340	29,184	29,104
Estimated Ratio (Mbps/Slice)	0.409	0.470	0.521	1.575	1.741	2.049	1.742

faster than ripple-carry adders (RCA) even though they are only 32-bit wide. All pipeline cuts are implemented by using positive edge triggered flip-flops.

4 Tiny XTEA

The objective of Tiny XTEA was to develop an ultra-low power implementation of XTEA. We assume that the 128-bit key and 64 bits of data are stored in memory and can be accessed via an 8-bit data bus. This bus width selection and the fact that we store a copy of the key and one data block in registers enables us to compare our design with an ultra-low power AES design reported in [16]. Our implementation acts as a co-processor and has commands for loading a key from memory and encrypting or decrypting one block of data.

4.1 ASIC Implementation

Power consumption in CMOS devices is the sum of the leakage power P_{Leak} (also called static power) and dynamic power P_{Dyn}. P_{Leak} is caused by the leakage current of each gate and therefore proportional to the circuit size. P_{Dyn} is caused by gate output changes from '0' to '1' and vice versa and hence proportional to the switching activity and to the clock frequency of the circuit. Ultra low-power applications such as RFID tags and energy scavenger powered sensor nodes [12] operate at frequencies of 100 kHz to 500 kHz where the leakage power is dominant. Therefore, its reduction must be the primary design goal. This can be achieved by reducing the circuit size.

From Fig. 2 we can see that it is sufficient to implement halfround 1 as it can perform the same function as halfround 2. We chose to use a 32-bit datapath as this is the native width of all operations used in XTEA. The main area consuming parts, as far as ASIC design is concerned, are the registers required to store the data (x and y), the key, and the variable *sum*. Using a smaller datapath, for example 8 bits, would reduce the size of the adders and XORs by one fourth. However, it would require the use of additional multiplexers to select between four bytes of the the 32-bit words[1] and to select 17 bits from z

[1] The current design employs only a single such multiplexer.

in order to facilitate the $f(z) = (z \ll 4 \oplus z \gg 5) + z$ function. It would also increase the number of clock cycles needed for one encryption, and require a more complex control logic. The number of registers will not be affected. Hence, a smaller datapath will not necessarily lead to smaller circuit with a reduced power or energy consumption.

The halfround function performs four integer additions if it is used as halfround 1 and three integer additions as halfround 2. We can use the same adder for additions and subtractions by including a small circuit to compute the 2's-complement of the subtrahend. The design of the halfround function can be scaled to use only one adder or up to four adders. In order to determine the ideal solution for an ultra-low power implementation on ASIC, we implemented all four versions. From those results, we estimated the results of an implementation that combines halfround 1 and halfround 2 and therefore incorporates seven adders. We used simple ripple carry adders (RCA) which consume less power than faster adders of this width and at the targeted clock frequency [16].

Table 2. Results for Halfround 1 Implementations

Number of Adders in Architecture	1	2	3	4	7[a]
Critical Path Delay (ns)	9.91	9.84	10.08	9.97	11.70
Clock Cycles for one operation	224	128	96	64	32
Area (NAND Equiv.)	1220	1330	1170	1351	2521
Dynamic Power (at 100 kHz) (μW)	0.49	0.54	0.67	0.86	1.54
Static Power (μW)	6.34	7.21	6.57	7.56	13.84
Total Power (at 100 kHz) (μW)	6.83	7.75	7.25	8.42	15.38
Energy per bit (pJ)	239.0	154.9	108.8	84.24	76.89

[a] Estimated, includes halfround1 and halfround 2

Table 2 shows that the critical path delay of all implementations is almost equivalent. The main contributors to the delay are the RCAs. The surprising result is, that the area of the circuit does not grow linearly with the number of adders. In fact, the halfround 1 architecture with three adders is smaller than the one with two adders. The architectures *adder1* and *adder2* need a 32-bit wide register to store temporary results which is not necessary in the other architectures. Another reason for the nonlinearity is that with each additional adder, the need for multiplexers to switch their inputs is reduced. The static power consumption also has nonlinearities. This is due to the fact that not all gates have the same leakage power. RCAs have a very high switching activity due to glitches which is emphasized when multiple adders are in series as in the *adder3* and *adder4* architectures. This leads to a higher dynamic power consumption. Another factor of the dynamic power consumption is the utilization of the adders in each state of the computation, e.g. the state machine of the *adder3* architecture utilizes only one adder in every third state and three in all other states. This reduces the average dynamic power consumption.

We implemented TinyXTEA-1 using the *adder1* architecture which has the lowest power consumption and TinyXTEA-3 using *adder3* which is a good compromise between power consumption and speed.

4.2 FPGA Implementation

The FPGA implementation is based on the design of TinyXTEA-1 and TinyXTEA-3. The main modification was the replacement of ripple carry adders with fast carry propagate adders offered by the Xilinx FPGAs.

5 Results

All our designs were described in VHDL. The FPGA designs were implemented using Xilinx ISE 9.1 tools and verified through post-place and route simulation with ModelSim SE 6.3a and test vectors generated from the C-code given in [4]. All results reported in the FPGA section are from post-place and route analysis. The ASIC implementations were synthesized using Synopsys power_compiler and a 0.13 μm, $V_{DD} = 1.2V$ ASIC library from TSMC which is characterized for power. All results were reported at the gate level. Results for power, energy and maximum delay from implementations using different CMOS processes, e.g.: 0.35 μm, 0.18 μm, can not be compared to ours as these results are depending on the technology used.

5.1 ASIC Implementations

The results for the high speed ASIC implementation of XTEA are shown in Table 3. It is interesting to note that SpeedXTEA-1 with one pipeline cut per halfround has the same maximum path delay as SpeedXTEA-2 with two cuts. However, the results are from gate level implementation, before placing and routing. Depending on the routing paths we will see a difference but it might not be as significant as for the FPGA implementations. The result of SpeedXTEA-3 shows that our assumption that XOR is in the critical path is correct. It yields the fastest speed of 36.6 Mbps. However, its throughput area ratio is less favorable than SpeedXTEA-1. Unfortunately, the comparison of SpeedXTEA with high speed AES implementations is rather difficult. Satoh's [17] AES implementation supports 128, 192, and 256-bit keys and operates in Galois Counter Mode. Hodjat's AES implementation in [18] has a throughput of 77 Gbps, however in later publications [19, 20] no detailed results were shown. The numbers for those implementations in Table 3 were estimated from the published graphs.

Table 4 compares our tiny XTEA implementations with an ultra-low power AES implementation reported by Kaps in [16], the landmark AES implementations by Feldhofer [21, 22] which both use 0.35 μm technology, as well as several light-weight crypto algorithm implementations. Just recently the eSTREAM portfolio [29] was published recommending four different stream ciphers for hardware implementations. Common to these stream ciphers is that they use an 80-bit

Table 3. Results for Speed XTEA compared to fast AES implementations (ASIC)

Design	Maximum Delay (ns)	Clock Cycles	Clock Cycles Latency	Block Size (bits)	Area (Gate Equivalents (GE))	Throughput (Mbps)	Throughput/ Area (kbps/GE)
SpeedXTEA-1	2.87	1	128	64	307,190	22,300	72.6
SpeedXTEA-2	2.87	1	192	64	420,562	22,300	53.0
SpeedXTEA-3	1.75	1	256	64	529,987	36,571	69.0
AES (Satoh)[17]	3.00	1	11	128	297,542	42,667	143.4
AES (Hodjat) [18]	1.65	1	41	128	473,000	77,576	164.0
AES (composite) [19]	2.00	1	41	128	175,000	64,000	365.7
AES (LUT) [19]	1.91	1	21	128	275,000	67,000	143.4

Table 4. Results for Tiny XTEA compared to block ciphers and the eSTREAM ciphers (ASIC)

Design	Maximum Delay (ns)	Clock Cycles	Block Size (bits)	Key Size (bits)	Area (Gate Equivalents (GE))	Throughput at 100KHz (Kbps)	Throughput/ Area (Kbps/GE)	Power (μW)	Energy per bit (pJ)
TinyXTEA-1	11.28	240	64	128	3500	26.7	0.008	18.8	703
TinyXTEA-3	11.66	112	64	128	3490	57.1	0.016	19.5	341
AES 8-bit[16]	2.19	534	128	128	4070	24.0	0.006	23.8	994
AES 8-bit[21][a]	–	1016	128	128	3595	12.6	0.004	26.9	2135
AES 8-bit[22][a]	12.50	1032	128	128	3400	12.4	0.004	4.5	363
DESXL[23, 24][b]	–	144	64	184	2168	44.4	0.021	1.6	36
Camelia[25][a]	27.67	21	128	128	11350	609.5	0.054	–	–
Camelia[26]	8.93	44	128	128	6511	290.1	0.045	–	–
Present-80[27][b]	–	32	64	80	1570	200.0	0.127	5.0	25
Present-128[27]	–	32	64	128	1886[c]	200.0	0.106	–	–
F-FCSR-H v2 [28]	2.55	1	8	80	4760	800	0.168	10.6	13
F-FCSR-16 [28]	3.15	1	16	128	8072	1,600	0.198	18.3	11
Grain v1 [28]	1.38	1	1	80	1294	100	0.077	3.3	33
Grain 128 [28]	1.08	1	1	128	1857	100	0.054	4.3	43
MICKEY v2 [28]	1.83	1	1	80	3188	100	0.031	7.1	71
MICKEY 128 [28]	2.42	1	1	128	5039	100	0.020	11.2	112
Trivium [28]	3.05	1	1	80	2580	100	0.039	5.5	55
Trivium (x64) [28]	2.87	1	64	80	4921	6,400	1.301	14.3	2

[a] Results are from 0.35 μm CMOS process.
[b] Results are from 0.18 μm CMOS process.
[c] Estimate, was not implemented in [27]

key. Tables 5 and 4 list these ciphers including derivations of some of them that allow for a 128-bit key.

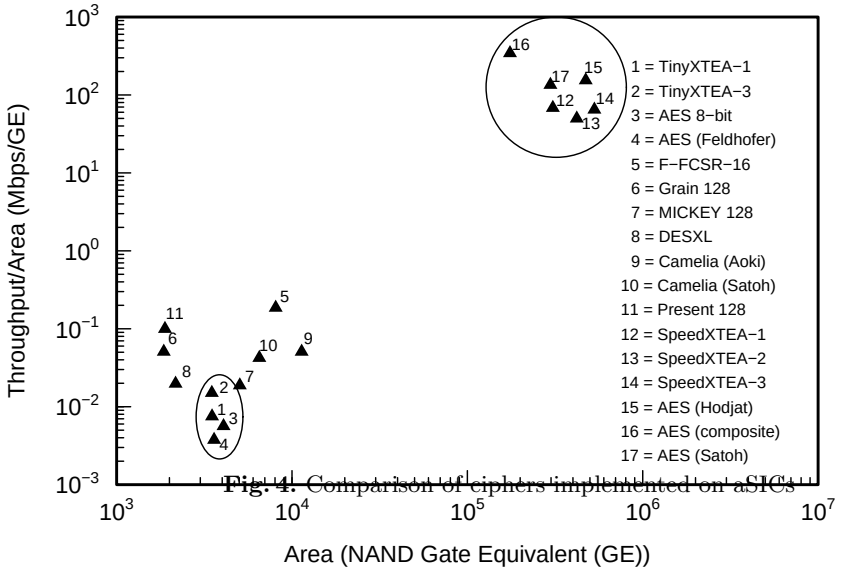

Fig. 4. Comparison of ciphers implemented on aSICs

The power consumption of TinyXTEA-1 is only marginally smaller than the one of TinyXTEA-3. This confirms our results from the implementations of only one halfround (see Table 2). TinyXTEA and the AES implementations have a similar area consumption. Hence, the power consumption of both implementations should also be similar if the same CMOS technology were used. The AES implementation in [22] consumes only a fifth of the power of the implementation reported in [21], even though both use the same CMOS technology. This is due to low level optimizations and voltage scaling. These techniques could also be employed for TinyXTEA. The surprising result is that the AES from [21] uses nine times more clock cycles than TinyXTEA-3 to encrypt twice as much data. This leads to a 4.5 times higher throughput for XTEA and could result in a 4.5 times lower energy consumption per bit.

The stream cipher implementation listed in Table 4 were published in [28] and have a much higher throughput than the XTEA or AES implementations at a similar power consumption. Therefore, they consume less energy per bit. Light weight ciphers like DESXL, Camelia and Present perform similarly well.

Figure 4 summarizes the results from Table 3 and Table 4 graphically. Two separate clusters can be easily identified, one for tiny XTEA and AES and one for fast XTEA and AES implementations. The results for tiny XTEA and AES

are very similar, the same applies to the results for high speed implementations. The results for the stream ciphers and light weight ciphers are scattered above the cluster for tiny XTEA and AES. Notably Present 128, Grain 128 and DESXL have a higher throughput while consuming a smaller sized area.

5.2 FPGA Implementations

The results of our XTEA implementations on FPGAs are summarized in Table 5 for tiny XTEA and Table 6 for speed XTEA. We expect from the ASIC analysis in Table 2 that TinyXTEA-1, using only one adder, consumes a slightly larger area than TinyXTEA-3 with three adders. Table 5 shows that this holds true for the FPGA implementation as well.

Table 5. Results for Tiny XTEA compared to 8-bit AES and the eSTREAM Portfolio ciphers (FPGA)

Design	Maximum Delay (ns)	Clock Cycles	Block Size (bits)	Key Size (bits)	Area (slices)	Throughput (Mbps)	Throughput/ Area (Mbps/slice)	Device
TinyXTEA-1	13.87	240	64	128	266	19	0.07	xc3s50-5
TinyXTEA-3	15.97	112	64	128	254	36	0.14	xc3s50-5
AES 8-bit[30]	14.93	3900	128	128	264	2	0.01	xc2s15-6
AES [31]	16.67	46	128	128	522	166	0.32	xc2s30-6
F-FCSR-H v2 [32]	7.25	1	8	80	342	1,104	3.23	xc3s50-5
F-FCSR-16 [32]	7.46	1	16	128	473	2,144	4.53	xc3s50-5
Grain v1 [32]	5.10	1	1	80	44	196	4.45	xc3s50-5
Grain 128 [32]	5.10	1	1	128	50	196	3.92	xc3s50-5
MICKEY v2 [32]	4.29	1	1	80	115	233	2.03	xc3s50-5
MICKEY 128 [32]	4.48	1	1	128	176	223	1.27	xc3s50-5
Trivium [32]	4.17	1	1	80	50	240	4.80	xc3s50-5
Trivium (x64) [32]	4.74	1	64	80	344	13,504	39.26	xc3s400-5

Figure 5 summarizes the results from Table 5 and Table 6 graphically. Three separate clusters can be easily identified, one for tiny XTEA and AES, one for stream ciphers, and one for fast XTEA and AES implementations. The results for tiny XTEA and AES are very similar, however stream ciphers have a higher throughput while consuming a smaller or similar sized area.

The throughput of TinyXTEA-3 is almost twice as fast as TinyXTEA-1 since it needs half as many clock cycles to encrypt one block of data. This is also reflected in the throughput to area ratio. The smallest AES implementation is the 8-bit AES by Good and Benaissa [30] which is similar in size to TinyXTEA. However, its throughput is almost 9 times lower than TinyXTEA-3. The AES by

Table 6. Results for Speed XTEA compared to fast AES implementations (FPGA)

Design	Maximum Delay (ns)	Clock Cycles	Clock Cycles Latency	Block Size (bits)	Area (slices)	Throughput (Mbps)	Throughput/ Area (Mbps/slice)	Device
SpeedXTEA-1	8.00	1	128	64	14,574	8,004	0.55	xc3s2000-5
SpeedXTEA-2	5.79	1	192	64	18,515	11,050	0.60	xc3s2000-5
SpeedXTEA-1 (V)	3.50	1	128	64	8,655	18,286	2.11	xc5vlx85-3
SpeedXTEA-2 (V)	3.10	1	192	64	9,647	20,645	2.14	xc5vlx85-3
AES 128-bit (S) [30]	5.10	1	70	128	17,425	25,101	1.44	xc3s2000-5
AES 128-bit (V) [30]	5.41	1	70	128	16,693	23,654	1.42	xcv2000E-8

Chodowiec and Gaj [31] is twice as large as TinyXTEA-3 and its throughput is almost twice as fast leading to a similar throughput area ratio. The 128-bit key versions of the stream ciphers [32] MICKEY and Grain have a higher throughput area ratio than XTEA or AES and occupy less area. We would like to remark that our implementation does not involve block RAMs.

The high speed XTEA implementations summarized in Table 6 confirm our predictions from Table 1. SpeedXTEA-2, with 2 cuts per halfround, has a much shorter critical path delay than SpeedXTEA-1, and even though it consumes more area, its throughput to area ratio is larger. Good and Benaissa report on two fully loop unrolled high speed AES implementations in [30], one on a Xilinx Spartan 3 (AES 128-bit (S)) and one on a Xilinx Virtex-E (AES 128-bit (V)). The AES implementation has a slightly shorter critical path delay than the SpeedXTEA-2 implementation on the same device and consume almost the same amount of area. However, due to its two times larger block size the throughput area ratio of AES is two times higher.

6 Conclusion

Our results on ASIC and FPGA show that XTEA is suitable for high-speed applications, however, it does not perform as fast as AES. Any speed improvement for XTEA would likely involve a significant increase in area and result in a lower throughput area ratio. Our ultra-low power implementation show that XTEA might be better suited for low resource environments than AES. Furthermore, XTEA's smaller block size of 64-bit is advantageous for applications where fewer than 128 bits of data have to be encrypted at a time. However, stream ciphers are a very interesting option as they have a higher throughput while consuming a smaller or similar sized area. The small code size of XTEA makes it an interesting choice in environments where some devices use software and other use hardware implementations.

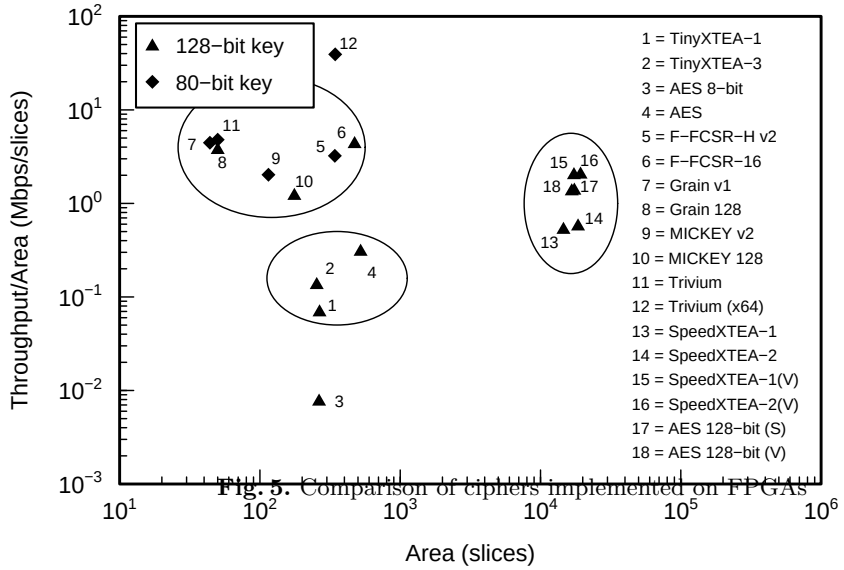

Fig. 5. Comparison of ciphers implemented on FPGAs

References

1. Wheeler, D., Needham, R.: TEA, a tiny encryption algorithm. Technical report, Cambridge University, England (Nov 1994)
2. Wheeler, D., Needham, R.: TEA, a tiny encryption algorithm. In: Fast Software Encryption, FSE 1994. Volume 1008 of LNCS., Springer-Verlag (1995) 363–366
3. Kelsey, J., Schneier, B., Wagner, D.: Related-key cryptanalysis of 3-WAY, biham-DES, CAST, DES-X, newDES, RC2, and TEA. In: Information and Communications Security ICIS '97. Volume 1334 of LNCS., Springer (1997) 233–246
4. Wheeler, D., Needham, R.: TEA extensions. Technical report, Cambridge University, England (Oct 1997)
5. Wheeler, D., Needham, R.: Correction to xtea. Technical report, Cambridge University (1998)
6. Castro, J.C.H., Viñuela, P.I.: New results on the genetic cryptanalysis of TEA and reduced-round versions of XTEA. In: Congress on Evolutionary Computation CEC2004. Volume 2. (2004) 2124–2129
7. Moon, D., Hwang, K., Lee, W., Lee, S., Lim, J.: Impossible differential cryptanalysis of reduced round XTEA and TEA. In Daemen, J., Rijmen, V., eds.: Fast Software Encryption, FSE 2002. Volume 2365 of LNCS., Springer (2002) 49–60
8. Liu, S., Gavrylyako, O.V., Bradford, P.G.: Implementing the TEA algorithm on sensors. In: ACM-SE 42: Proceedings of the 42nd annual Southeast regional conference, New York, NY, USA, ACM Press (2004) 64–69
9. Pavlin, M.: Encryption using low cost microcontrollers. In: 42nd International Conference on Microelectronics, Devices and Materials and the Workshop on MEMS and NEMS, Society for Microelectronics Electronic (2006) 189–194

10. Niati, R., Yazdani, N.: A more energy efficient network setup method for wireless sensor networks. In: Asia-Pacific Conference on Communications. (2005) 640–643
11. Kanamori, Y.: Reliability and security in a wireless body area network of intelligent sensors. Master's thesis, The University of Alabama in Huntsville (July 2002)
12. Amirtharajah, R., Chandrakasan, A.P.: Self-powered signal processing using vibration-based power generation. IEEE Journal of Solid-State Circuits **33**(5) (May 1998) 687–695
13. Israsena, P.: Securing ubiquitous and low-cost RFID using tiny encryption algorithm. In: Symp. on Wireless Pervasive Computing, IEEE (Jan 2006) 4 pp.
14. Israsena, P.: Design and implementation of low power hardware encryption for low cost secure RFID using TEA. In: Information, Communications and Signal Processing. (Dec 2005) 1402–1406
15. Brent, R.P., Kung, H.T.: A regular layout for parallel adders. IEEE Transactions on Computers **C-31**(3) (Mar 1982) 260–264
16. Kaps, J.P., Sunar, B.: Energy comparison of AES and SHA-1 for ubiquitous computing. In et al., X.Z., ed.: Embedded and Ubiquitous Computing (EUC-06) Workshop Proceedings. Volume 4097 of LNCS., Springer (Aug 2006) 372–381
17. Satoh, A.: High-speed hardware architectures for authenticated encryption mode GCM. In: International Symposium on Circuits and Systems (ISCAS) 2006. (May 2006) 4831–4834
18. Hodjat, A., Verbauwhede, I.: Speed-area trade-off for 10 to 100 Gbits/s throughput AES processor. In: Asilomar Conference on Signals, Systems and Computers. Volume 2. (Nov 2003) 2147–2150
19. Hodjat, A., Verbauwhede, I.: Area-throughput trade-offs for fully pipelined 30 to 70 Gbits/s AES processors. IEEE Trans. Computers **55**(4) (Apr 2006) 366–372
20. Hodjat, A., Verbauwhede, I.: Minimum area cost for a 30 to 70 Gbits/s AES processor. In: IEEE Comp. Soc. Annual Symposium on VLSI. (Feb 2004) 83–88
21. Feldhofer, M., Dominikus, S., Wolkerstorfer, J.: Strong authentication for RFID systems using the AES algorithm. In: Cryptographic Hardware and Embedded Systems – CHES 2004. Volume 3156 of LNCS., Springer (Aug 2004) 357–370
22. Feldhofer, M., Wolkerstorfer, J., Rijmen, V.: AES implementation on a grain of sand. Information Security, IEE Proceedings **152**(1) (Oct 2005) 13–20
23. Leander, G., Paar, C., Poschmann, A., Schramm, K.: New lightweight DES variants. In: Fast Software Encryption, FSE 2007. Volume 4593 of LNCS., Springer (2007) 196–210
24. Poschmann, A., Leander, G., Schramm, K., Paar, C.: New light-weight crypto algorithms for RFID. In: International Symposium on Circuits and Systems (ISCAS). (May 2007) 1843–1846
25. Aoki, K., Ichikawa, T., Kanda, M., Matsui, M., Moriai, S., Nakajima, J., Tokita, T.: Camellia: A 128-bit block cipher suitable for multiple platforms – design and analysis. In: Selected Areas in Cryptography, SAC 2000. Volume 2012 of LNCS., Springer (2001) 39–56
26. Satoh, A., Morioka, S.: Hardware-focused performance comparison for the standard block ciphers AES, Camellia, and Triple-DES. In: 6th Information Security Conference, ISC'03. Volume 2851 of LNCS., Springer (2003) 252–266
27. Bogdanov, A., Knudsen, L., Leander, G., Paar, C., Poschmann, A., Robshaw, M., Seurin, Y., Vikkelsoe, C.: PRESENT: An ultra-lightweight block cipher. In: Cryptographic Hardware and Embedded Systems – CHES 2007. Volume 4727 of LNCS., Springer (2007) 450–466

28. Good, T., Benaissa, M.: Hardware performance of eStream phase-iii stream cipher candidates. In: State of the Art of Stream Ciphers Workshop (SASC 2008). (Feb 2008) 163–173
29. Babbage, S., Canniére, C.D., Canteaut, A., Cid, C., Gilbert, H., Johansson, T., Parker, M., Preneel, B., Rijmen, V., Robshaw, M.: The eSTREAM portfolio. Technical report, eSTREAM, ECRYPT Stream Cipher Project (Apr 2008)
30. Good, T., Benaissa, M.: AES on FPGA from the fastest to the smallest. In: Cryptographic Hardware and Embedded Systems – CHES 2005. Volume 3659 of LNCS., Springer (2005) 427–440
31. Chodowiec, P., Gaj, K.: Very compact FPGA implementation of the AES algorithm. In: Cryptographic Hardware and Embedded Systems – CHES 2003. Volume 2779 of LNCS., Springer (2003) 319–333
32. Hwang, D., Chaney, M., Karanam, S., Ton, N., Gaj, K.: Comparison of FPGA-targeted hardware implementations of eSTREAM stream cipher candidates. In: State of the Art of Stream Ciphers Workshop (SASC 2008). (Feb 2008) 151–162