

Ascon HW Design Document

Erich Wenger and Hannes Groß

Graz University of Technology
Institute for Applied Information Processing and Communications
Inffeldgasse 16a, 8010 Graz, Austria
`{Erich.Wenger,Hannes.Gross}@iaik.tugraz.at`

1 Introduction

Our goal was it to explore the design space of the ASCON hardware design by Dobraunig, Eichlseder, Mendel and Schl  ffer [1]. Within this paper, we document the common hardware bus interface as well as the memory layout of all our ASCON implementations. In Section 4, we check the implementations for their hardware characteristics.

2 Assumptions

Researchers found many ways to let their hardware implementations shine using many different semi-advisable techniques. In the following, a discussion of the different techniques is given.

Technology. Hardware results highly depend on the used manufacturing technology.

As a comparison of designs, manufactured in different technologies, in terms of μm^2 is not possible, researchers use the unit Gate Equivalents (GE). A GE is defined as the size of a 2-input NAND gate with 0.5 drive. This normalized unit makes the chip area more comparable than μm^2 . However, there are technologies with bigger NAND gates and there are technologies with smaller NAND gates. Let's guess which technology is preferred by researchers...

Interface. Having a design without bus interface might reduce the size and the complexity of a hardware implementation significantly. However, there exist only a very limited number of applications that use custom architectures that may not be based on a microprocessor or do not use a synchronous bus interface. Therefore, one has to question the significance of a general purpose hardware implementation without bus interface.

“Signal is magically created by test bench” or “key comes from the outside”.

Testing is one of the most critical practical requirements of most implementations. Usually that is done with a test bench that is written in VHDL, Verilog, or TCL. The test bench is the source of the key, random numbers, the plaintext, etc. However, in the most generic case, both the key and the random numbers have to be written via the bus.

RAM/ROM Macros are a legible option for many practical implementations. However, for better comparison at least the synthesis results without using a macros should be reported. Also, even a proper use of latches to save area is *not* good practice.

Toolchain. Cadence, Synopsys, Mentor offer differently clever tools with different advantages and merits. As it can hardly be expected that a researcher has access to all design flows, a minimum is to state the used design flow and the used versions of the tools.

Post synthesis vs. post place-and-route. It is common practice to report post-synthesis area results. However, a designer has to make sure that her design will be actually routable without significant area overhead (e.g., less than ten percent).

Maximum clock frequencies are highly dependent on the used technology, the operating temperature, and the used toolchain. For high-throughput designs it is reasonable to assume at least a 100 MHz clock. Multiple clock domains usually want to be avoided and are only used when absolutely necessary. For resource constrained implementations a minimum maximum-clock-frequency of 10 MHz in a “modern” technology seems reasonable.

Comparisons of different implementations done in different technologies, with different toolchains, etc., must be doubted for their fairness. A fair comparison can only be done if all designs were made under nearly-identical assumptions and tools. Within the CAESAR competition it is advisable to publish the used VHDL/Verilog code such that other researchers may verify the results and use it for fair comparisons.

3 Bus Interface

A synchronous 32-bit bus interface is used for all our implementations. Although there exist potentially interesting microprocessors with smaller memory interfaces (e.g., the Atmel AVR ATmega series or the Texas Instruments MSP430 series) and some dedicated applications (e.g., Internet-of-Things, RFID, Wireless Sensor Networks) with very unique requirements, the majority of industry will use a 32-bit processor with a synchronous bus interface. Anyways, it is straightforward to strip away the 32-bit bus interface and replace it with the custom interface needed for the custom design goals.

3.1 Bus Signals

Table 1 specifies the interface signals to access the hardware accelerator. It is closely related to the AXI-lite bus specification [1] with some minor differences that ease the development of the designs competing within the CAESAR competition.

The bus has a synchronous interface with the following use cases:

- A global asynchronous active-low reset sets all registers to their initial value.
- If `CSxSI` and `WExSI` are ‘1’, then `DataWritexDI` is written to `AddressxDI` when the clock changes from ‘0’ to ‘1’.
- If `CSxSI` is ‘1’ and `WExSI` is ‘0’, then `AddressxDI` an internal multiplexer is used to update `DataReadxD0` asynchronously.

Thus every read and every write operation is finished in a single clock cycle. The AXI-lite bus always needs to consecutive cycles to perform read and write operations. The modification of the specification from above to support the AXI-lite specification is straightforward: add registers to `CSxSI`, `WExSI`, and `AddressxDI` to introduce a one-cycle delay and add an output register to `DataReadxD0`.

Table 1. Bus signals used for 32-bit bus interface.

Name	I/O	Functionality
ClkxCi	in	clock signal
RstxRBI	in	asynchronous active-low reset
CSxSI	in	active-high chip select/enable
WExSI	in	high: write, low: read
AddressxDI	in	20-bit word address signal
DataWritexDI	in	32-bit data write bus
DataReadxD0	out	32-bit data read bus

3.2 Memory Layout

In general, an Authentication Encryption with Associated Data (AEAD) hardware design must support the following functionality:

- Initialization
- Authentication of data
- Encryption of plaintext
- Decryption of cipher text
- Finalization
- Reading of authentication tag

In particular, the interface for our Ascon hardware designs is shown in Table 2.

Table 2. Memory layout.

Word Address	R/W	Functionality
0000	read	0xdeadbeef to check basic connectivity
0001	read	Status Register: reads 1 if busy, otherwise 0
0002	write	Control Register: <ul style="list-style-type: none"> - bit 0: Initialize - bit 1: Associate - bit 2: Encrypt - bit 3: Decrypt - bit 4: Final Encrypt - bit 5: Final Decrypt - bit 6: Final Associate
0004-0007	write	128-bit key, little endian
0008-000B	write	128-bit nonce, little endian
000C-000D	write	64-bit data to encrypt/decrypt/associate, little endian
000C-000D	read	64-bit encrypted/decrypted data, little endian
0010-0013	read	128-bit tag, little endian

3.3 Software Access

The usage of the interface within software will work as follows. Listing 1.1 defines the structure to access the hardware. Listing 1.2 converts the interface from the CAESAR

competition to access the proposed hardware designs. `CAST_B_I` and `SET_I_B` are two macros that convert the endianness. For an ease of pseudo-code, Listing 1.2 does not handle the padding of data and does not perform wait-and-start optimizations (where CPU and HW work in parallel).

Listing 1.1. Memory Interface defined in C. Accessible at address 0x40000000.

```
typedef struct CAESAR_INTERFACE_ {
    uint32_t unique_value;
    uint32_t status;
    uint32_t control;
    uint32_t filler1;
    uint32_t key[4];
    uint32_t nonce[4];
    uint32_t data[2];
    uint32_t filler2[2];
    uint32_t tag[4];
} CAESAR_INTERFACE;

volatile CAESAR_INTERFACE interface = (volatile CAESAR_INTERFACE*) 0x40000000;

#define CTRL_INIT (1 << 0)
#define CTRL_ASSOCIATE (1 << 1)
#define CTRL_ENCRYPT (1 << 2)
#define CTRL_DECRYPT (1 << 3)
#define CTRL_ENCRYPT_FINAL (1 << 4)
#define CTRL_DECRYPT_FINAL (1 << 5)
#define CTRL_ASSOCIATE_FINAL (1 << 6)
```

4 Results

The hardware was synthesized using Cadence Encounter v08.10 and a UMC 90 nm manufacturing technology. The main runtime and area results are summarized in Table 3.

Table 3. Characteristics of Ascon hardware implementations.

Design	Ascon-fast	Ascon-small-64bit	Ascon-super-small
Area			
Cycles per Round	1		
Cycles per Round plus interface	1		
Cycles per Byte	0.75		
Maximum frequency	TBD		
Maximum throughput	TBD		

Listing 1.2. Simple way of using the hardware to authenticate and encrypt data.

```
int crypto_aead_encrypt(
    uint8_t *c, uint64_t *cLen,
    const uint8_t *m, uint64_t mLen,
    const uint8_t *ad, uint64_t adLen,
    const uint8_t *nsec,
    const uint8_t *npub,
    const uint8_t *k) {
    interface->key[0] = CAST_B_I(k+0);
    interface->key[1] = CAST_B_I(k+4);
    interface->key[2] = CAST_B_I(k+8);
    interface->key[3] = CAST_B_I(k+12);
    interface->nonce[0] = CAST_B_I(npub+0);
    interface->nonce[1] = CAST_B_I(npub+4);
    interface->nonce[2] = CAST_B_I(npub+8);
    interface->nonce[3] = CAST_B_I(npub+12);
    interface->control = CTRL_INIT;
    while(interface->status != 1);

    // TODO: specially handle padding and adlen % 8 != 0
    while(adlen >= 8) {
        interface->data[0] = CAST_B_I(ad); ad += 4;
        interface->data[1] = CAST_B_I(ad); ad += 4; adlen -= 8;
        interface->control = CTRL_ASSOCIATE;
        while(interface->status != 1);
    }
    interface->control = CTRL_ASSOCIATE_FINAL; // single cycle

    // TODO: specially handle padding and mlen % 8 != 0
    while(mLen > 8) {
        interface->data[0] = CAST_B_I(m); m+=4;
        interface->data[1] = CAST_B_I(m); m+=4;
        interface->control = CTRL_ENCRYPT;
        while(interface->status != 1);
        SET_I_B(c, interface->data[0]); c+=4;
        SET_I_B(c, interface->data[1]); c+=4;
    }
    interface->data[0] = CAST_B_I(m); m+=4;
    interface->data[1] = CAST_B_I(m); m+=4;
    interface->control = CTRL_ENCRYPT_FINAL;
    while(interface->status != 1);
    SET_I_B(c, interface->data[0]); c+=4;
    SET_I_B(c, interface->data[1]); c+=4;
    SET_I_B(c, interface->tag[0]); c+=4;
    SET_I_B(c, interface->tag[1]); c+=4;
    SET_I_B(c, interface->tag[2]); c+=4;
    SET_I_B(c, interface->tag[3]);
}
```