Advanced Encryption Standard (AES) Tiny Version

November 15th, 2008 Product Specification V1.0

Features

- Fully compliant with NIST (FIPS-197) standards
- Supports key sizes of 128, 192 or 256 bits with hardware-based key expansion
- Supports stand-alone encryption, decryption or both using 128-bit data blocks in ECB format
  - Optional support for CBC, CFB and OFB formats
- Supports initialization of sbox ram tables by external source or internal source (ROM)
- 32-bit data interface simplifies loading of keys and data
- Supports burst operations with and without fifo
- Supports background scrubbing of sbox tables for improved reliability
- Testbench verifies FIPS compliance through known answer test (KAT)
- Deployed into multiple production designs

IP Core Facts	
Provided with Core	
Documentation	Core datasheet and testbench description
Design File Format	VHDL RTL or Verilog netlist
Constraint Files	SDC and PDC constraints
Verification	Testbench using Modelsim from Mentor
Synthesis Tool Used	
Synplify Version 9.4A1	
Support	
Provided by local sales channel	

Table 1 - Implementation Statistics for A3P/Fusion/Igloo[1]

Key Size	Encryption	Decryption	Fifo	RAM Blocks	Tiles (EXT init)[2]	Tiles (ROM init)[3]	Speed[4] (MHz)	Throughput (Mbps)
128	Yes	Yes	No	6	1724	2221	116	201
	Yes	No	No	6	1347	1843	129	223
	No	Yes	No	6	1406	1901	117	202
	Yes	Yes	Yes	8	1728	2224	120	208
	Yes	No	Yes	8	1358	1855	126	218
	No	Yes	Yes	8	1415	1908	121	209
192	Yes	Yes	No	6	1684	2188	117	170
	Yes	No	No	6	1347	1841	130	189
	No	Yes	No	6	1402	1900	121	176
	Yes	Yes	Yes	8	1689	2189	116	169
	Yes	No	Yes	8	1347	1845	125	182
	No	Yes	Yes	8	1402	1898	122	177
256	Yes	Yes	No	6	1767	2258	119	149
	Yes	No	No	6	1390	1883	122	153
	No	Yes	No	6	1447	1942	121	152
	Yes	Yes	Yes	8	1769	2264	110	138
	Yes	No	Yes	8	1393	1889	125	157
	No	Yes	Yes	8	1449	1947	113	142

Notes:

1) Igloo V5 performance is approximately 67% of the speed shown. Igloo V2 performance is approximately 40% of the speed shown
2) EXT refers to external initialization of the sbox ram tables – see the Generics section for more information
3) ROM refers to internal initialization of the sbox ram tables using ROM within the TinyAES core – see the Generics section
4) All performance numbers are based on A3P250-2PQ208 with single pass TDPR

AES Algorithm Overview

The Advanced Encryption Standard (AES) specifies a Federal Information Processing Standards (FIPS) approved cryptographic algorithm that can be used to protect electronic data. The AES algorithm is a symmetric block cipher that can encrypt (encypher) and decrypt (decypher) information. Encryption converts plaintext data to an unintelligible form called cipher-text. Decrypting the cipher-text converts the data back into its original plaintext form.

The AES algorithm is capable of using cryptographic keys of 128, 192, and 256 bits to encrypt and decrypt data in blocks of 128 bits. The algorithm is used with the three different key lengths indicated above, and therefore these different "flavors" are referred to as "AES-128", "AES-192", and "AES-256". For the AES algorithm, the amount of processing or number of rounds to be performed during the execution of the algorithm is dependent on the key size. The number of rounds is represented by ***Nr***, where

Nr =10 when for AES-128, ***Nr*** = 12 for AES-192, and ***Nr*** = 14 for AES-256.

Table 2 illustrates the breakdown of processing steps based on the different key sizes. The throughput is therefore decreased as the key size is increased. In other words, the two additional rounds for each increase in key size decreases the overall throughput of the TinyAES core due to the additional processing required.

Table 2 - AES Algorithm

Version	Key Size	Block Size	Rounds (Nr)
AES-128	128 bits	128 bits	10
AES-192	192 bits	128 bits	12
AES-256	256 bits	128 bits	14

The AES algorithm requires an expansion of the orginal key to then provide a unique key for each round of the cipher/decipher process. The step to create these additional keys is called key expansion. For the TinyAES core, the key expansion step is required each time a new key is used. Once a new key has been expanded, then 128-bit data can be input to the core continually. To simplify the loading of 128-bit block data and to provide a burst mode capability of up to 64 data blocks, a 256 x 32 bit fifo can be optionally added to the front-end of the core. Figure 1 below illustrates the architecture of the TinyAES core. Actel's flash-based FPGAs are a perfect fit for AES data security applications due to their inherent device security and non-volatile attributes.

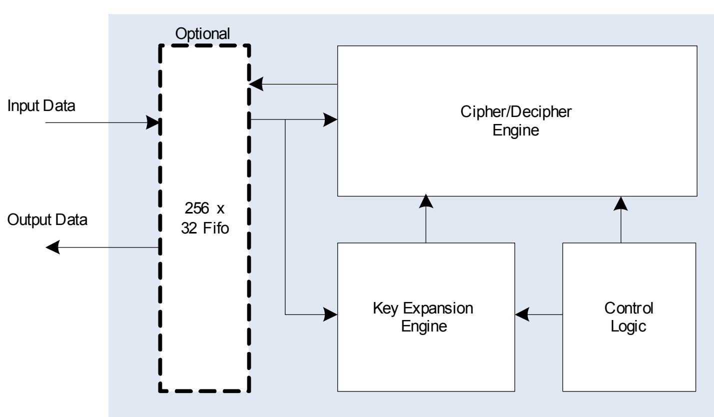

Figure 1 – TinyAES Block Diagram

Generic Definitions

Table 2 shows the generic settings that need to be configured for the desired user operation of the AES core. Using generics, maximum flexibility is obtained allowing a balance of area versus feature tradeoffs to be made.

Table 3 - Generics for TinyAES

Generic	Type	Values	Description
KEYSIZE	Integer	128, 192, 256	Specifies the key size for the AES core
INIT_SBOX	Text	"EXT", "ROM"	Specifies the method for initialization of the sbox ram tables "EXT" = external source loads the 512 bytes "ROM" = internal 512x8 ROM block loads the 512 bytes
FIFO	Integer	0 or 1	Specifies whether a 256x32bit fifo is used in the core for data into and out of the AES core 0 = no fifo used 1 = use fifo
ENCRYPTOR	Integer	0 or 1	Specifies whether to enable the encryption engine 0 = encryptor disabled 1 = encryptor enabled
DECRYPTOR	Integer	0 or 1	Specifies whether to enable the decryption engine 0 = decryptor disabled 1 = decryptor enabled
SCRUBBING	Integer	0 or 1	Specifies whether to enable background scrubbing of the sbox tables 0 = scrubbing disabled 1 = scrubbing enabled

Notes:

KEYSIZE

As the key size increases, there is a reduction in throughput based on the increased number of processing rounds. The TinyAES core size is mostly independent of the key size.

INIT_SBOX

In a processor-based system, or a device with internal flash memory, the option of EXT may be desirable because of the tile savings of over 20% versus using the internal ROM block. Since the ROM block consumes roughly 500 tiles, where possible, the EXT option is preferred since the tile count is reduced.

ENCRYPTOR AND DECRYPTOR

At least one of these generics must have a value of 1. 0 for both generics is an invalid combination. Size and speed tradeoffs can be made if one of the functions is not needed.

SCRUBBING

SCRUBBING is only available when the INIT_SBOX setting is ROM. This feature is useful when it is desired to ensure that the sbox tables have not been upset due to a neutron upset. When enabled, and the AES core is not busy, the sbox tables are automatically re-written from ROM to refresh the contents and ensure the highest reliability of the subsequent crypto functions. Impact to the AES core size is negligible, however dynamic power is increased slightly. Therefore, for the lowest power implementation, SCRUBBING should be disabled.

Signal Description

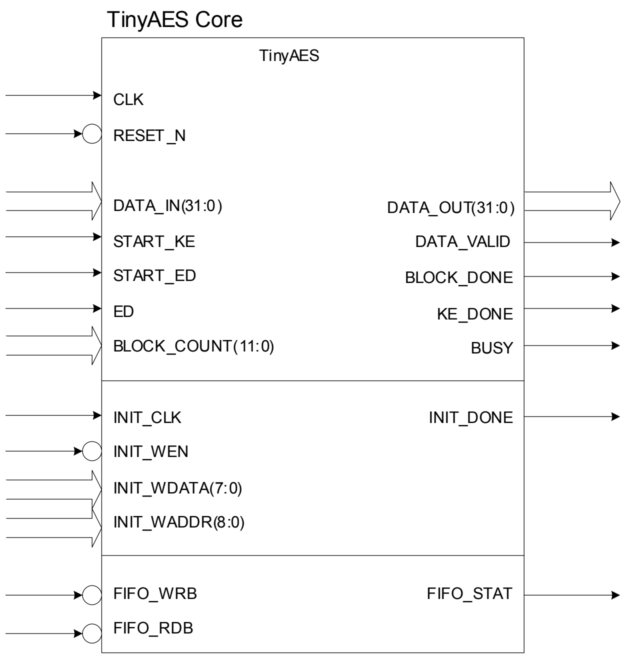

Figure 2 – TinyAES I/O Diagram

Table 4 - I/O Signal Description

Signal	Direction	Description
CLK	Input	Input clock to all registers and RAM
RESET_N	Input	LO active asynchronous clear of all registers
DATA_IN	Input	32-bit input data for key expansion, plaintext encryption or cipher-text decryption
START_KE	Input	HI active signal used to start the key expansion process. Only needs to be asserted for one clock cycle to start key expansion.
START_ED	Input	HI active signal used to start an encryption or decryption process. Only needs to be asserted for one clock cycle to start the process.
ED	Input	Specifies whether an encrypt or decrypt function is performed 0 = encrypt process; 1 = decrypt process
BLOCK_COUNT	Input	12-bit input specifies the number of consecutive 128-bit data block cryptographic operations. For single data block operations, block_count should be x"001".
DATA_OUT	Output	32-bit output data containing plaintext or cipher-text. 4 words are always output consecutively synchronous with DATA_VALID
DATA_VALID	Output	HI active signal indicating valid data on the DATA_OUT bus. This signal is always HI for 4 consecutive CLK cycles.
BLOCK_DONE	Output	HI active signal indicating that a block of crypto functions has been completed
KE_DONE	Output	HI active signal indicating completion of the key expansion process
BUSY	Output	HI active signal indicating that the TinyAES core is busy
INIT_CLK	Input	Input clock to SBOX ram. Must be less than or equal to CLK frequency.
INIT_WEN	Input	LO active write enable to the SBOX ram
INIT_WDATA	Input	8-bit write data to the SBOX ram
INIT_WADDR	Input	9-bit write address to the SBOX ram (512 byte addresses)
INIT_DONE	Output	HI active signal indicating the SBOX tables have been initialized after reset.
FIFO_WRB	Input	LO active signal that allows the fifo to be written to
FIFO_RDB	Input	LO active signal that allows the fifo to be read from
FIFO_STAT	Output	HI active signal indicating that the input fifo is empty

It is important to understand the effect of the generic settings on the user I/O of the TinyAES core. Table 5 below illustrates when an I/O pin is used based on the generic setting.

Table 5 - Generic Settings and Effect on I/O

Signal	SBOX_INIT=EXT	SBOX_INIT=ROM	FIFO=0	FIFO=1
INIT_CLK	YES	YES		
INIT_WEN	YES	NO		
INIT_WDATA	YES	NO		
INIT_WADDR	YES	NO		
INIT_DONE	NO	YES		
FIFO_WRB			NO	YES
FIFO_RDB			NO	YES
FIFO_STAT			NO	YES

The I/O signals not shown above are always necessary for proper operation of the TinyAES core. The I/Os above that have a NO entry can be left unconnected on the core given the generic setting shown. Shaded regions indicate that the generic has no effect on that particular I/O.

Functional Description

Initialization

The TinyAES core requires an initialization of its internal sbox ram tables before cryptographic functions can be started. There are two ways that these tables can be loaded based on the setting of the generic INIT_SBOX. If the ROM setting is chosen, then an internal ROM block is instantiated into the TinyAES core which is used to automatically load the sbox tables. This process is started immediately after the de-assertion of the RESET_N signal. After 1026 INIT_CLK cycles, the INIT_DONE signal is asserted HI to indicate completion of the initialization process. The INIT_DONE signal remains HI until the next assertion of RESET_N. Figure 3 below illustrates the timing for sbox initialization from internal ROM. No other actions are required to start the init process other than the de-assertion of the RESET_N signal

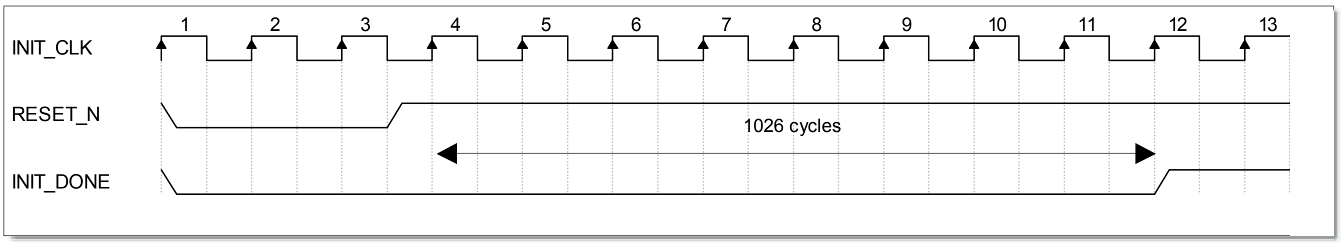

Figure 3 – Initialization Timing with INIT_SBOX=ROM

In an effort to offer optimal area efficiency for the TinyAES core, an external memory interface is available to load the sbox tables from external processor flash or from on-chip non-volatile flash memory (NVM). By setting the INIT_SBOX generic to EXT, the ROM block is excluded from the core build and a 20% tile count reduction is obtained. 512 bytes of data need to be loaded using the memory interface. During this setting, the INIT_DONE is unused since the memory interface control is outside of the core. Appendix A contains the data table that needs to be loaded via the memory interface. Figure 4 shows the timing required to initialize the sbox ram tables using the external memory interface.

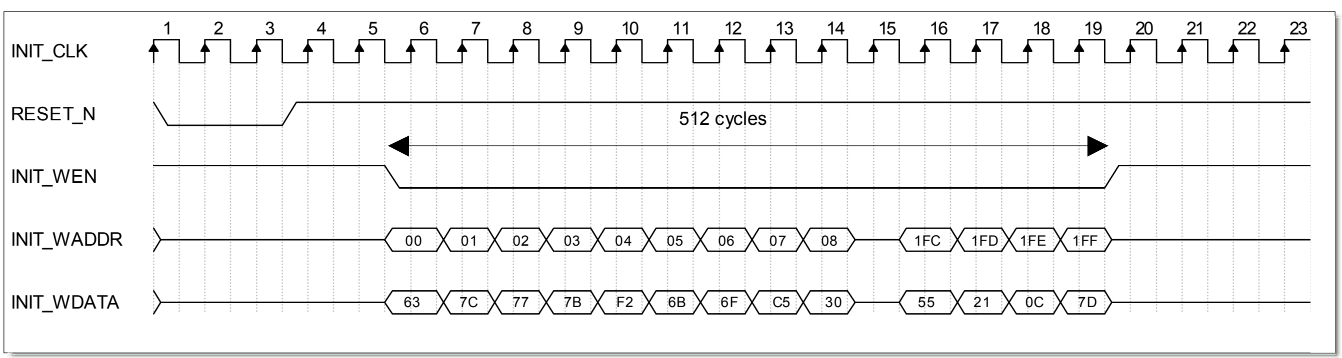

Figure 4 – Initialization Timing with INIT_SBOX=EXT

Key Expansion

The key expansion step is required each time a new key is to be used in the cryptographic process. Before a cryptographic process is performed, the AES algorithm requires the chosen key (regardless of size) to be expanded. During the key expansion step, the key is input to the core and stored in ram where it then gets passed through a logic chain ten times to produce ten sub-keys (in the case of a 128-bit key). These additional keys are also stored in ram to be later used in the actual cryptographic function (see Figure 5).

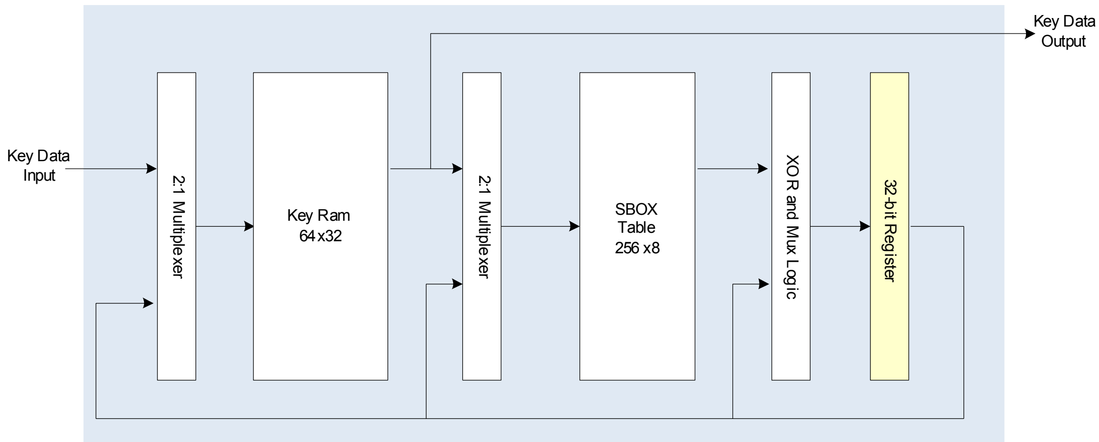

Figure 5 Key Expansion Engine Block Diagram

Key expansion only needs to be done once before cipher and decipher functions can be started. The only time it needs to be run again is when a different key is desired. The process of key expansion takes 68 (128 bit key), 74 (192-bit key), or 89 (256-bit key) clock cycles based on the key size selected. Key expansion and cryptographic functions can not be overlapped. The timing diagram shown below in Figure 6 shows that START_KE initiates the key expansion and KE_DONE indicates its completion.

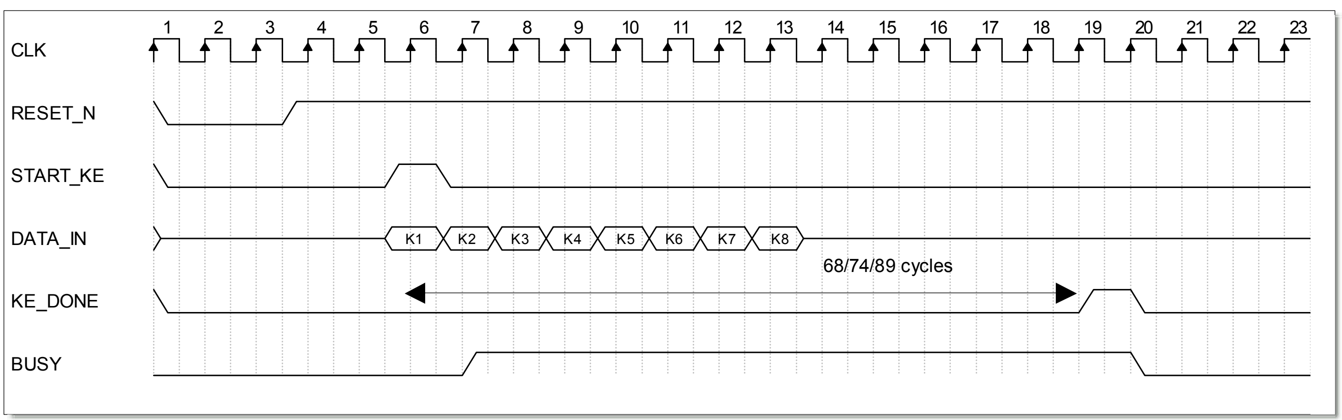

Figure 6 – Key Expansion Timing

Data Formatting

Given a 128-bit key arranged in four 32-bit long words, the load order would be from left to right starting with K1. K1(0) would be input on DATA_IN(0) with K1(31) input on DATA_IN(31), and then on the next CLK cycles K2, K3 and K4 in the bit order shown. Additional long words K5 and K6 are only needed for 192-bit key sizes, and likewise K7 and K8 required only for 256-bit key sizes.

K1	K2	K3	K4
0……………………31	32…………………..63	64…………………..95	96…………………127

Key Security

Since the expanded keys generated are central to the cryptographic integrity, the key ram is not accessible from outside the TinyAES core. In fact, even during calculation and storage of the sub-keys, no key information is exposed outside of the core.

128-bit Key Expansion Example

Key: 000102030405060708090a0b0c0d0e0f

Based on this key selection the Table 6 below shows the contents of the key ram after the key expansion.

Table 6 - Key Ram Contents Example

Key Ram Address	Contents	Expansion Round #
0	00010203	K1
1	04050607	K2
2	08090a0b	K3
3	0c0d0e0f	K4
4	d6aa74fd	1
5	d2af72fa	1
6	daa678f1	1
7	d6ab76fe	1
8	b692cf0b	2
9	643dbdf1	2
0A	be9bc500	2
0B	6830b3fe	2
0C	b6ff744e	3
0D	d2c2c9bf	3
0E	6c590cbf	3
0F	0469bf41	3
10	47f7f7bc	4
11	95353e03	4
12	f96c32bc	4
13	fd058dfd	4
14	3caaa3e8	5
15	a99f9deb	5
16	50f3af57	5
17	adf622aa	5
18	5e390f7d	6
19	f7a69296	6
1A	a7553dc1	6
1B	0aa31f6b	6
1C	14f9701a	7
1D	e35fe28c	7
1E	440adf4d	7
1F	4ea9c026	7
20	47438735	8
21	a41c65b9	8
22	e016baf4	8
23	aebf7ad2	8
24	549932d1	9
25	f0855768	9
26	1093ed9c	9
27	be2c974e	9
28	13111d7f	10
29	e3944a17	10
2A	f307a78b	10
2B	4d2b30c5	10

Cipher/Decipher Engine

This block is responsible for the core AES cryptographic algorithm processing. The stage sequencer shown in Figure 7 controls the AES processing functions. Similar to key expansion, multiple rounds are required for a complete AES encrypt or decrypt function (10,12 or 14 rounds per Table 2). Since this core contains one cipher engine, each round must pass through the engine sequentially. The TinyAES core uses a 32-bit data path for processing the 128-bit AES data block. Figure 7 shows how 4 clock cycles are needed to load the 128-bit state register plus 3 more clock cycles to complete the sbox translation and mix column or inverse mix column functions (see yellow shading to indicate register stages). Therefore a total of 7 clock cycles are required for each round.

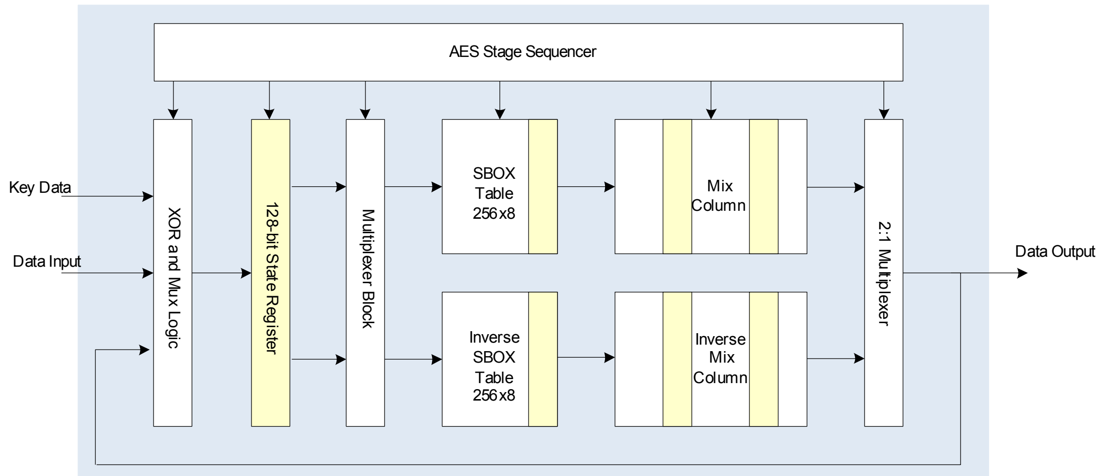

Figure 7 - Cipher/Decipher Engine Block Diagram

The equation below shows the total number of clock cycles required for processing a 128-bit AES data block through the TinyAES core.

(1) #cycles = (Nr x Ce) + Ohd

where Nr = the number of rounds (10 for 128-bit key)
where Ce = the number of cycles to pass through the cipher engine (7)
where Ohd = the number of overhead cycles (2 for single, 4 for burst)

Using equation (1), we can compute the number of clock cycles required to encrypt or decrypt a 128-bit block of data.

Table 7 - TinyAES Cycle Count

Key Size	# Cycles (Single operation)	# Cycles (Burst operation)
128 bits	(10 x 7) + 2 = 72	(10 x 7) + 4 = 74
192 bits	(12 x 7) + 2 = 86	(10 x 7) + 4 = 88
256 bits	(14 x 7) + 2 = 100	(10 x 7) + 4 = 102

The data throughput of TinyAES can be calculated using the results from Table 7 and equation (2) below. Together with the operating frequency values from Table 1, the data throughput rate for each of the core variants, can be calculated for a 128-bit block of data. The throughput is measured in bits-per-second (bps).

(2) Throughput(bps) = frequency x $(\#cycles)^{-1}$ x 128

For example with a 128-bit key size, if the speed of the chosen core variant runs at 120.8MHz in the device selected, then the throughput would be

$$208.9 \text{ Mbps} = 120.8 \times 10^6 \times (1/74) \times 128$$

The throughput number indicates the sustained input data rate that can be supported by the TinyAES core. This is possible running at 120.8MHz because the core operates on 128-bit data blocks and can perform a complete cryptographic function in as few as 74 cycles.

Figure 8 shows the timing required to perform an encryption or decryption operation. The value of the ED signal determines which type of operation is performed. The START_ED signal begins the operation on the next rising edge of the clock. The first of the four 32-bit plaintext words to be processed are input on the same rising edge as the START_ED with the next 3 plaintext words input on successive clock cycles as shown by P1, P2, P3 and P4. For a single operation (BLOCK_COUNT=001h), after 72 clock cycles (with 128-bit key size), the DATA_VALID signal is asserted for four clock cycles to indicate that the DATA_OUT bus contains valid data. For each DATA_VALID clock cycle, a new 32-bit word is driven out on the DATA_OUT bus as shown by E1, E2, E3, and E4.

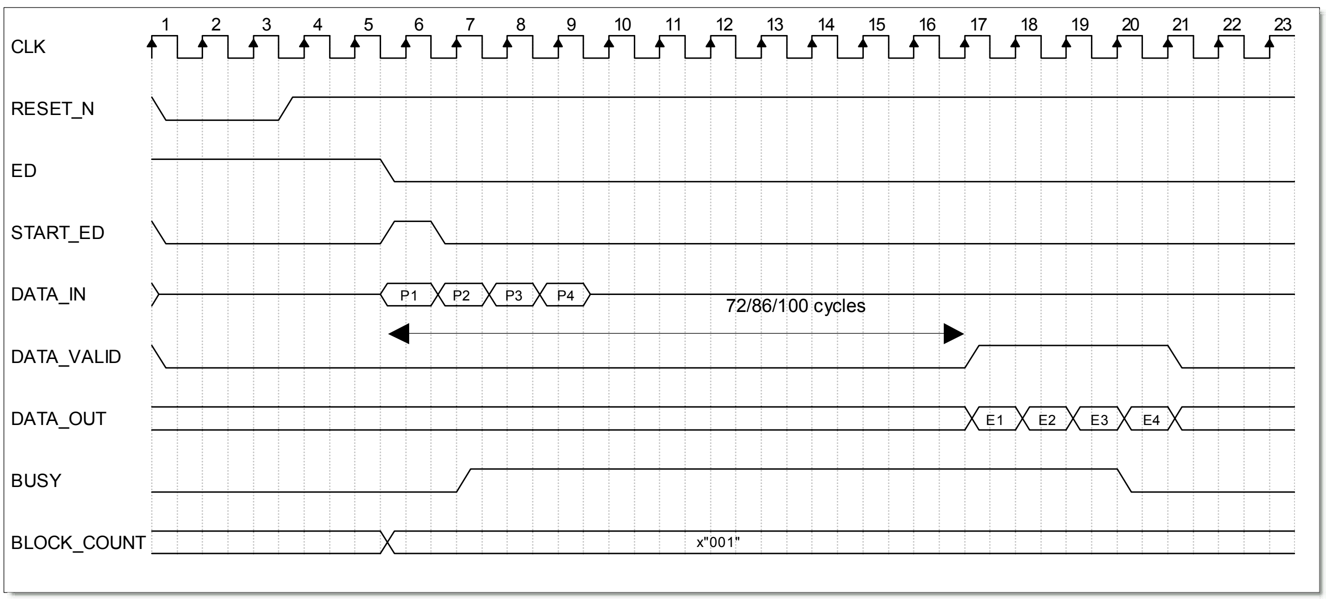

Figure 8 – Single Operation Encryption Timing (without fifo)

Figure 9 below shows the relative timing between DATA_VALID assertions during burst mode. For a 128-bit key size, from rising edge of DATA_VALID to the next rising edge is 74 cycles.

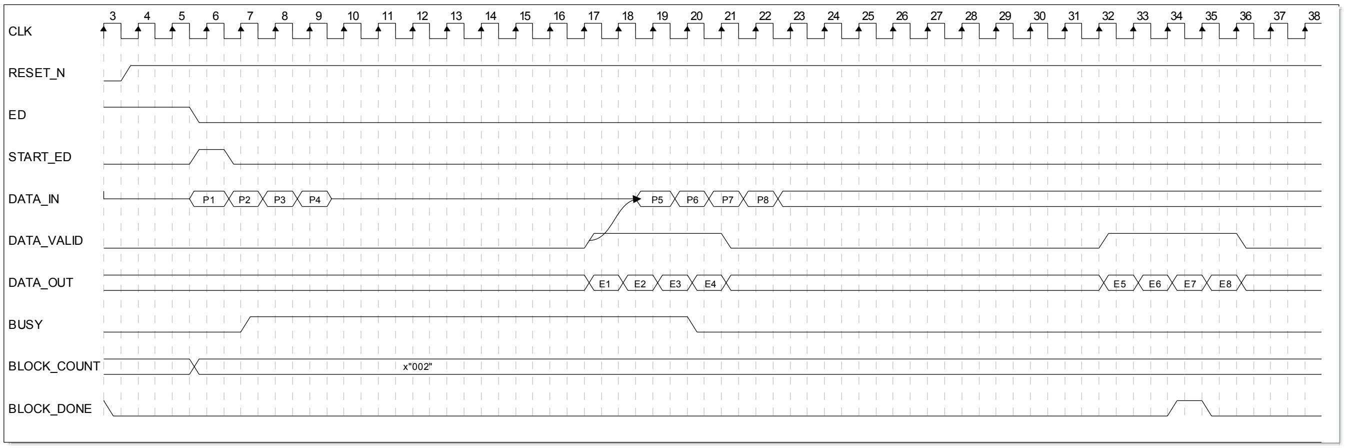

Figure 9 – Burst Operation Encryption Timing (without fifo)

As a note, the decryption timing is identical with the only difference being the polarity of the ED signal. Also, encryption and decryption can mixed in the same block sequence. In other words, in Figure 9, at the falling edge of clock 18, ED could be driven HI to perform a decryption operation of the next data block. There are no limitations on mixing these functions because the core handles them the same from a timing standpoint.

Fifo Option Timing
The FIFO option is currently implemented as a single 256x32 block. The user and the AES engine both share the fifo which minimizes the memory block utilization. If the user wants a larger fifo or full-duplex (unique input and output fifo blocks), then it is recommended to not use FIFO mode and simply use the DATA_VALID and START signals to control output and input fifos external to the core.

Figure 10 below shows the required signal timing to interface to the single fifo block. Up to 256 32-bit words (or 64 AES data blocks) can be preloaded into the fifo to be processed in a burst mode. During encryption or decryption of the fifo data, the BUSY signal is asserted to indicate that the fifo is busy being accessed by the AES engine. Once the core is not busy, then the FIFO_RDB signal can be asserted thereby allowing the fifo contents to be read by the user.

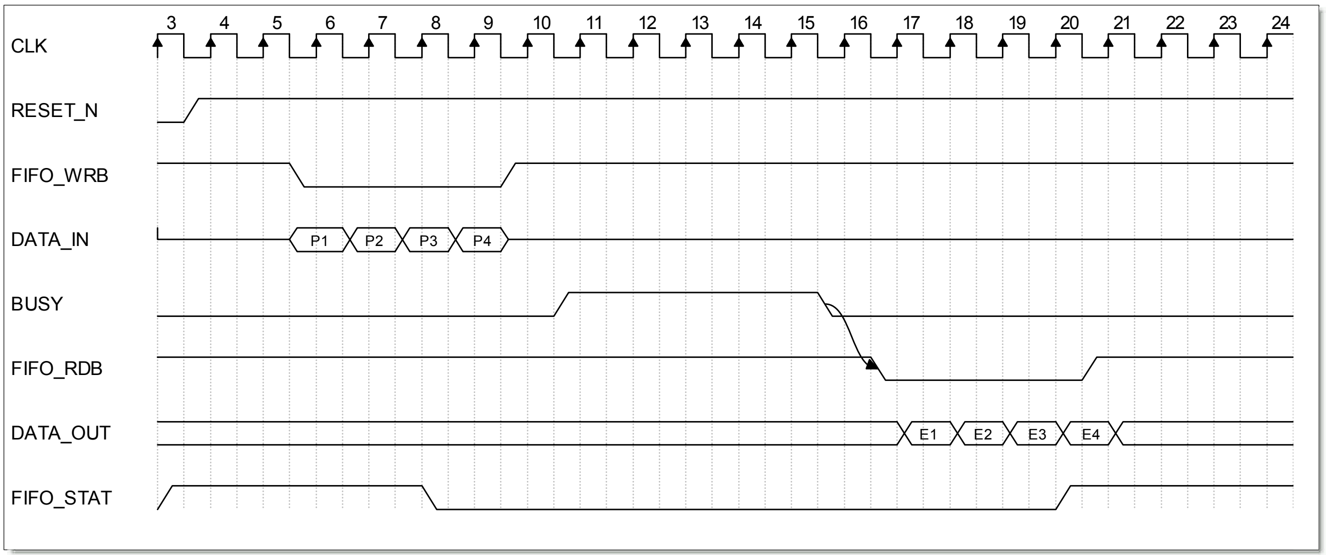

Figure 10 – Fifo Control Timing

Verification

Figure 11 shows the architecture of the verification testbench used to validate the TinyAES core variants.

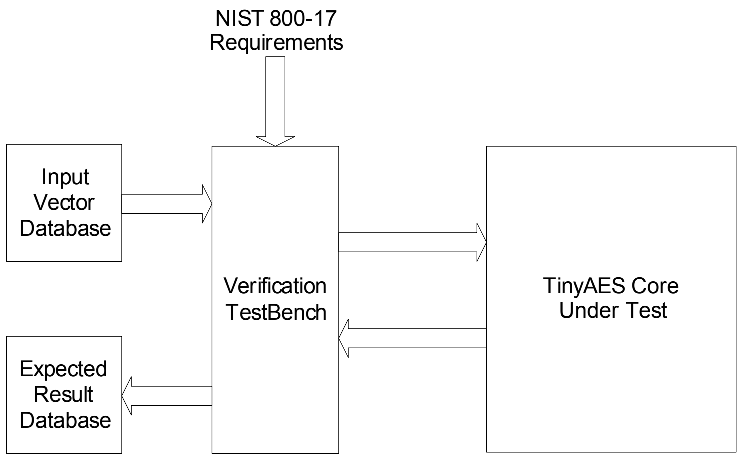

Figure 11 – Verification Block Diagram

Deliverables

Table 8 below shows the source code hierarchy of the TinyAES core. The current RTL deliverable is VHDL, however, verilog RTL can be provided on request.

Table 8 - Source Code Hierarchy

Item	Description
TinyAES	Top level file
ROM_sboxr_g3	ROM source code for internal initialization of sbox ram
sbox_table	256 byte ROM file used to load sbox ram with both normal and inverse sbox tables
TinyAESF_wrp	Fifo based wrapper file for the AES engine
fifo256x32	256x32 synchronous fifo block (2 ram blocks)
aes_top	Main AES function
cypher_engine	This level allows for the 3 key sizes to be passed to aes_cd_fast
aes_cd_fast	Main encryption/decryption engine
block_count12	12-bit block counter for burst operations
iGFmult	Inverse galois field multiplier for inverse mix column function
Gfmult	Galois field multiplier for mix column function
key_engine	Main key expansion block that specifies the proper key size expansion engine
aes_key_fast128	Performs 128-bit key expansion
aes_key_fast192	Performs 192-bit key expansion
aes_key_fast256	Performs 256-bit key expansion
ram64x32_sync	Key expansion RAM (used in all 3 key expansion options)
sbox_ram	512x32 of ram for storage of both sbox tables (total of 4 ram blocks)
ram512x8	Synchronous access (lower 256 is normal sbox table and upper 256 is inverse table)
TinyAES_wrp.vhd	Wrapper file for the AES engine
aes_top	Main AES function
cypher_engine	This level allows for the 3 key sizes to be passed to aes_cd_fast
aes_cd_fast	Main encryption/decryption engine
block_count12	12-bit block counter for burst operations
iGFmult	Inverse galois field multiplier for inverse mix column function
Gfmult	Galois field multiplier for mix column function
key_engine	Main key expansion block that specifies the proper key size expansion engine
aes_key_fast128	Performs 128-bit key expansion
aes_key_fast192	Performs 192-bit key expansion
aes_key_fast256	Performs 256-bit key expansion
ram64x32_sync	Key expansion RAM (used in all 3 key expansion options)
sbox_ram	512x32 of ram for storage of both sbox tables (total of 4 ram blocks)
ram512x8	Synchronous access (lower 256 is normal sbox table and upper 256 is inverse table)

Table 9 – Constraint Files

Item	Description
TinyAESext.sdc	SDC file needed when SBOX_INIT=EXT
TinyAESrom.sdc	SDC file needed when SBOX_INIT=ROM; multi-cycle constraints for INIT_CLK
TinyAESrom.pdc	PDC file needed when SBOX_INIT=ROM; puts INIT_CLK on a global
EXT_load.dat	External load file for sbox ram. 512 bytes of sbox data. (See appendix A)

Table 10 – Simulation Files – 128-bit key size

Item	Description
All_mc_aes128.do	Top level do file for running known answer test
TinyAES_tbmc128.vhd	Testbench for reading and comparing multiple cipher/decipher functions – Pass/Fail notification
ecb_vt_e.dat	Same key for each cipher but with varied plaintext
ecb_vt_d.dat	Same key for each decipher but with varied cipher-text
ecb_vk_e.dat	Same plaintext for each cipher but with varied key
ecb_vk_d.dat	Same cipher-text for each decipher but with varied key
ecb_tbl_128e.dat	Varied key and plaintext for each cipher
ecb_tbl_128d.dat	Varied key and cipher-text for each decipher
wave_key128.do	Waveform file for examining signals in Modelsim wave window
All_chip_aes128.do	Basic burst mode test without fifo. Examine Modelsim wave window for correct output
TinyAES_tb128.vhd	Basic burst mode testbench
wave_key128.do	Waveform file for examining signals in Modelsim wave window
All_chip_aes128f.do	Basic burst mode test with fifo. Examine Modelsim wave window for correct output
TinyAESf_tb128.vhd	Basic burst mode testbench with fifo
wave_std.do	Waveform file for examining signals in Modelsim wave window showing fifo signals
All_chip_aes128ext.do	Basic burst mode test without fifo; Uses external load of sbox tables
TinyAES_tb128ext.vhd	Basic burst mode testbench with external load of sbox tables
wave_key128.do	Waveform file for examining signals in Modelsim wave window

Table 11 – Simulation Files – (ZZZ = 192 or 256)

Item	Description
All_mc_aesZZZ.do	Top level do file for running known answer test
TinyAES_tbmcZZZ.vhd	Testbench for reading and comparing multiple cipher/decipher functions – Pass/Fail notification
ecb_tbl_ZZZe.dat	Varied key and plaintext for each cipher
ecb_tbl_ZZZd.dat	Varied key and cipher-text for each decipher
wave_keyZZZ.do	Waveform file for examining signals in Modelsim wave window
All_chip_aesZZZ.do	Basic burst mode test without fifo. Examine Modelsim wave window for correct output
TinyAES_tbZZZ.vhd	Basic burst mode testbench
wave_keyZZZ.do	Waveform file for examining signals in Modelsim wave window
All_chip_aesZZZf.do	Basic burst mode test with fifo. Examine Modelsim wave window for correct output
TinyAESf_tbZZZ.vhd	Basic burst mode testbench with fifo
wave_std.do	Waveform file for examining signals in Modelsim wave window showing fifo signals

Appendix A – External Initialization of Sbox

Sbox RAM Load Pattern (1st 256 bytes) – Sbox Table

ADDR	DATA
0	63
1	7C
2	77
3	7B
4	F2
5	6B
6	6F
7	C5
8	30
9	01
0A	67
0B	2B
0C	FE
0D	D7
0E	AB
0F	76
10	CA
11	82
12	C9
13	7D
14	FA
15	59
16	47
17	F0
18	AD
19	D4
1A	A2
1B	AF
1C	9C
1D	A4
1E	72
1F	C0
20	B7
21	FD
22	93
23	26
24	36
25	3F
26	F7
27	CC
28	34
29	A5
2A	E5
2B	F1
2C	71
2D	D8
2E	31
2F	15
30	04
31	C7
32	23
33	C3

ADDR	DATA
34	18
35	96
36	05
37	9A
38	07
39	12
3A	80
3B	E2
3C	EB
3D	27
3E	B2
3F	75
40	09
41	83
42	2C
43	1A
44	1B
45	6E
46	5A
47	A0
48	52
49	3B
4A	D6
4B	B3
4C	29
4D	E3
4E	2F
4F	84
50	53
51	D1
52	00
53	ED
54	20
55	FC
56	B1
57	5B
58	6A
59	CB
5A	BE
5B	39
5C	4A
5D	4C
5E	58
5F	CF
60	D0
61	EF
62	AA
63	FB
64	43
65	4D
66	33
67	85

ADDR	DATA
68	45
69	F9
6A	02
6B	7F
6C	50
6D	3C
6E	9F
6F	A8
70	51
71	A3
72	40
73	8F
74	92
75	9D
76	38
77	F5
78	BC
79	B6
7A	DA
7B	21
7C	10
7D	FF
7E	F3
7F	D2
80	CD
81	0C
82	13
83	EC
84	5F
85	97
86	44
87	17
88	C4
89	A7
8A	7E
8B	3D
8C	64
8D	5D
8E	19
8F	73
90	60
91	81
92	4F
93	DC
94	22
95	2A
96	90
97	88
98	46
99	EE
9A	B8
9B	14

ADDR	DATA
9C	DE
9D	5E
9E	0B
9F	DB
A0	E0
A1	32
A2	3A
A3	0A
A4	49
A5	06
A6	24
A7	5C
A8	C2
A9	D3
AA	AC
AB	62
AC	91
AD	95
AE	E4
AF	79
B0	E7
B1	C8
B2	37
B3	6D
B4	8D
B5	D5
B6	4E
B7	A9
B8	6C
B9	56
BA	F4
BB	EA
BC	65
BD	7A
BE	AE
BF	08
C0	BA
C1	78
C2	25
C3	2E
C4	1C
C5	A6
C6	B4
C7	C6
C8	E8
C9	DD
CA	74
CB	1F
CC	4B
CD	BD
CE	8B
CF	8A

ADDR	DATA
D0	70
D1	3E
D2	B5
D3	66
D4	48
D5	03
D6	F6
D7	0E
D8	61
D9	35
DA	57
DB	B9
DC	86
DD	C1
DE	1D
DF	9E
E0	E1
E1	F8
E2	98
E3	11
E4	69
E5	D9
E6	8E
E7	94
E8	9B
E9	1E
EA	87
EB	E9
EC	CE
ED	55
EE	28
EF	DF
F0	8C
F1	A1
F2	89
F3	0D
F4	BF
F5	E6
F6	42
F7	68
F8	41
F9	99
FA	2D
FB	0F
FC	B0
FD	54
FE	BB
FF	16

Sbox RAM Load Pattern (2^{ND} 256 bytes) - Inverse Sbox Table

ADDR	DATA
100	52
101	9
102	6A
103	D5
104	30
105	36
106	A5
107	38
108	BF
109	40
10A	A3
10B	9E
10C	81
10D	F3
10E	D7
10F	FB
110	7C
111	E3
112	39
113	82
114	9B
115	2F
116	FF
117	87
118	34
119	8E
11A	43
11B	44
11C	C4
11D	DE
11E	E9
11F	CB
120	54
121	7B
122	94
123	32
124	A6
125	C2
126	23
127	3D
128	EE
129	4C
12A	95
12B	0B
12C	42
12D	FA
12E	C3
12F	4E
130	8
131	2E
132	A1
133	66

ADDR	DATA
134	28
135	D9
136	24
137	B2
138	76
139	5B
13A	A2
13B	49
13C	6D
13D	8B
13E	D1
13F	25
140	72
141	F8
142	F6
143	64
144	86
145	68
146	98
147	16
148	D4
149	A4
14A	5C
14B	CC
14C	5D
14D	65
14E	B6
14F	92
150	6C
151	70
152	48
153	50
154	FD
155	ED
156	B9
157	DA
158	5E
159	15
15A	46
15B	57
15C	A7
15D	8D
15E	9D
15F	84
160	90
161	D8
162	AB
163	0
164	8C
165	BC
166	D3
167	0A

ADDR	DATA
168	F7
169	E4
16A	58
16B	5
16C	B8
16D	B3
16E	45
16F	6
170	D0
171	2C
172	1E
173	8F
174	CA
175	3F
176	0F
177	2
178	C1
179	AF
17A	BD
17B	3
17C	1
17D	13
17E	8A
17F	6B
180	3A
181	91
182	11
183	41
184	4F
185	67
186	DC
187	EA
188	97
189	F2
18A	CF
18B	CE
18C	F0
18D	B4
18E	E6
18F	73
190	96
191	AC
192	74
193	22
194	E7
195	AD
196	35
197	85
198	E2
199	F9
19A	37
19B	E8

ADDR	DATA
19C	1C
19D	75
19E	DF
19F	6E
1A0	47
1A1	F1
1A2	1A
1A3	71
1A4	1D
1A5	29
1A6	C5
1A7	89
1A8	6F
1A9	B7
1AA	62
1AB	0E
1AC	AA
1AD	18
1AE	BE
1AF	1B
1B0	FC
1B1	56
1B2	3E
1B3	4B
1B4	C6
1B5	D2
1B6	79
1B7	20
1B8	9A
1B9	DB
1BA	C0
1BB	FE
1BC	78
1BD	CD
1BE	5A
1BF	F4
1C0	1F
1C1	DD
1C2	A8
1C3	33
1C4	88
1C5	7
1C6	C7
1C7	31
1C8	B1
1C9	12
1CA	10
1CB	59
1CC	27
1CD	80
1CE	EC
1CF	5F

ADDR	DATA
1D0	60
1D1	51
1D2	7F
1D3	A9
1D4	19
1D5	B5
1D6	4A
1D7	0D
1D8	2D
1D9	E5
1DA	7A
1DB	9F
1DC	93
1DD	C9
1DE	9C
1DF	EF
1E0	A0
1E1	E0
1E2	3B
1E3	4D
1E4	AE
1E5	2A
1E6	F5
1E7	B0
1E8	C8
1E9	EB
1EA	BB
1EB	3C
1EC	83
1ED	53
1EE	99
1EF	61
1F0	17
1F1	2B
1F2	4
1F3	7E
1F4	BA
1F5	77
1F6	D6
1F7	26
1F8	E1
1F9	69
1FA	14
1FB	63
1FC	55
1FD	21
1FE	0C
1FF	7D