



# AES (Rijndael) IP-Cores

## Encryption/Decryption and Key Expansion

### Data Sheet

Revision 1.3

April 2006



## Revision History

Date	Version	Description
24 February 2006	1.0	Initial draft.
15 March 2006	1.1	Block diagrams added.
26 March 2006	1.2	Timing diagrams added.
03 April 2006	1.3	Implementation statistics completed.

ErSt is disclosing this Intellectual Property (hereinafter "IP") to you for use in the development of designs to operate on FPGAs. Any unauthorized use of the IP may violate copyright laws, trademark laws, the laws of privacy and publicity, and communications regulations and statutes.

ErSt does not assume any liability arising out of the application or use of the IP; nor does ErSt convey any license under its patents, copyrights, or any rights of others. You are responsible for obtaining any rights you may require for your use or implementation of the IP. ErSt reserves the right to make changes, at any time, to the IP as deemed desirable in the sole discretion of ErSt. ErSt assumes no obligation to correct any errors contained herein or to advise you of any correction if such be made. ErSt will not assume any liability for the accuracy or correctness of any engineering or technical support or assistance provided to you in connection with the IP.

THE IP IS PROVIDED "AS IS" WITH ALL FAULTS, AND THE ENTIRE RISK AS TO ITS FUNCTION AND IMPLEMENTATION IS WITH YOU. YOU ACKNOWLEDGE AND AGREE THAT YOU HAVE NOT RELIED ON ANY ORAL OR WRITTEN INFORMATION OR ADVICE, WHETHER GIVEN BY ERST, OR ITS AGENTS OR EMPLOYEES. ERST MAKES NO OTHER WARRANTIES, WHETHER EXPRESS, IMPLIED, OR STATUTORY, REGARDING THE IP, INCLUDING ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE, AND NONINFRINGEMENT OF THIRD-PARTY RIGHTS.

IN NO EVENT WILL ERST BE LIABLE FOR ANY CONSEQUENTIAL, INDIRECT, EXEMPLARY, SPECIAL, OR INCIDENTAL DAMAGES, INCLUDING ANY LOST DATA AND LOST PROFITS, ARISING FROM OR RELATING TO YOUR USE OF THE IP, EVEN IF YOU HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THE TOTAL CUMULATIVE LIABILITY OF ERST IN CONNECTION WITH YOUR USE OF THE IP, WHETHER IN CONTRACT OR TORT OR OTHERWISE, WILL IN NO EVENT EXCEED THE AMOUNT OF FEES PAID BY YOU TO ERST HEREUNDER FOR USE OF THE IP. YOU ACKNOWLEDGE THAT THE FEES, IF ANY, REFLECT THE ALLOCATION OF RISK SET FORTH IN THIS AGREEMENT AND THAT ERST WOULD NOT MAKE AVAILABLE THE IP TO YOU WITHOUT THESE LIMITATIONS OF LIABILITY.

The IP is not designed or intended for use in the development of on-line control equipment in hazardous environments requiring fail-safe controls, such as in the operation of nuclear facilities, aircraft navigation or communications systems, air traffic control, life support, or weapons systems ("High-Risk Applications"). ErSt specifically disclaims any express or implied warranties of fitness for such High-Risk Applications. You represent that use of the IP in such High-Risk Applications is fully at your risk.

Spartan and Virtex are trademarks of Xilinx Inc.

© ErSt Electronic GmbH, 2006. All rights reserved. All specifications are subject to change without notice.

## Table of Contents

<b>1. Overview</b>	<b>4</b>
<b>2. What is AES and how does it work?</b>	<b>4</b>
<b>3. Implementation Statistics</b>	<b>5</b>
<b>4. Standard Encryption Core AES_ENC</b>	<b>6</b>
4.1. Functional Description	6
4.2. Interface Signals	6
4.3. Timing Diagram	7
4.4. VHDL Entity Declaration	8
<b>5. Standard Decryption Core AES_DEC</b>	<b>9</b>
5.1. Functional Description	9
5.2. Interface Signals	9
5.3. Timing Diagram	10
5.4. VHDL Entity Declaration	11
<b>6. Standard Encryption/Decryption Core AES_ENC_DEC</b>	<b>12</b>
6.1. Functional Description	12
6.2. Interface Signals	12
6.3. Timing Diagram	13
6.4. VHDL Entity Declaration	14
<b>7. Standard Key Expander Core KEY_EXPANDER_WITH_STORAGE</b>	<b>15</b>
7.1. Functional Description	15
7.2. Interface Signals	16
7.3. Timing Diagram	17
7.4. VHDL Entity Declaration	17
<b>8. Test Bench</b>	<b>18</b>
8.1. Known Answer Tests	18
8.2. Random Tests	18
<b>9. Deliverables</b>	<b>19</b>
<b>10. Export</b>	<b>20</b>
<b>11. Related Information</b>	<b>20</b>
11.1. Advanced Encryption Standard Specification	20
11.2. Xilinx FPGAs and Software	20

## 1. Overview

These cores implement the AES (Advanced Encryption Standard, Rijndael) encryption standard, as described in the NIST (National Institute of Standards and Technology) Federal Information processing Standard (FIPS) Publication 197 document. The cores cover both encryption/decryption functions and key expansion, supporting any or all of the proposed key sizes (128/192/256-bit). Our cores implement all the building blocks individually giving the user the greatest possible flexibility. These cores are designed to be simple to use and can be integrated into any AES design with minimum effort.

## 2. What is AES and how does it work?

AES is suitable for any application that requires strong encryption technology. This new encryption standard may replace the previously used triple-DES where the superior efficiency of Rijndael can be used to gain much increased data throughput for less logic real-estate. Typical applications might include secure communications, program content protection for digital media applications, storage area, networks, VPN, secure VoIP, wireless LAN, electronic banking etc..

The general Rijndael algorithm is a block cipher with multiple options for its block and key size. The NIST approved AES is a subset of these options with a fixed block size of 128-bits, but the key may be either 128, 192 or 256-bits in length. This means, that a basic AES engine is capable of encrypting plain text data in blocks of 128-bits using any of the specified key sizes. Higher levels of security can be achieved by using bigger key sizes.

The AES algorithm consists of a complex non-linear core function, which is iterated multiple times starting from the incoming plain text data block. Each iteration is called a "round". The round function is slightly modified for the final round and there is an additional pre-processing round at the start of every encryption. The number of "rounds" required depends on the selected key size. For a key size of 128-bit there are 10 rounds, for a 192-bit key there are 12 rounds, and for a 256-bit key there are 14 rounds. The consequence of this is that the longer key sizes do take slightly more time to process.

Each round of AES requires a unique 128-bit round key schedule that is generated from the supplied 128-bit, 192-bit or 256-bit AES key using a key expansion algorithm. For 128-bit keys one needs 11 key schedules, for 192-bit keys one needs 13 key schedules and for 256-bit keys one needs 15 key schedules. The key expansion process can be accomplished in one of two ways. For the encryption the round key schedules can be generated "on the fly" in real-time when they are required by the encryption algorithm. This is especially useful if the AES keys need to change on a regular basis. If the AES keys do not get changed too often then the round key schedules may be generated off-line and stored in internal RAM for subsequent use.

### Features

- ✓ Implements AES (Rijndael) to latest NIST FIPS PUB 197
- ✓ Full support for all AES key sizes (128, 192 and 256-bits)
- ✓ Separate cores for encryption, decryption and round key generation
- ✓ Simple external interface
- ✓ Code optimized for use in Xilinx FPGA technologies
- ✓ Data throughput up to 2 Gbps in Virtex-4 FPGAs
- ✓ Low cost

### Deliverables

- ✓ Fully synthesizable RTL VHDL code and NGC netlists for Xilinx FPGAs
- ✓ VHDL simulation model and test bench with FIPS test vectors and random tests
- ✓ User documentation

### 3. Implementation Statistics

To get an estimate of the required resources and maximum system frequencies we simulate the environment of each entity with a wrapper. It is placed around the core and registers all the inputs and outputs. This detaches any external timing influences and allows a single PERIOD constraint to cover all delays within and in and out of the core. We can use "area groups" to force the core to place within as small an area as possible (this mimics the FPGA being very full) which allows the user to replicate our performance in their own design by using a similar area group. In addition, we can separate out the wrapper registers (which are outside of the area group) from the core itself, to give slice numbers (the Xilinx ISE MAP report gives the correct core slice area in the form of the area group results).

**Table 1:** Implementation statistics for the AES\_ENC encryption core with external key expander.

FPGA Family	Example Device	F <sub>max</sub> (MHz) <sup>1</sup>	Data Throughput (Mbps) <sup>2</sup>	GCLK	DCM/DLL	Slices	BRAM
Spartan-3™	XC3S1000-4	100	1163	1	0	280	8
Virtex-2™	XC2V500-4	118	1373	1	0	335	8
Virtex-2 Pro™	XC2VP7-5	154	1792	1	0	302	8
Virtex-4™	XC4VLX25-11	174	2025	1	0	289	8

**Table 2:** Implementation statistics for the AES\_DEC decryption core with external key expander.

FPGA Family	Example Device	F <sub>max</sub> (MHz) <sup>1</sup>	Data Throughput (Mbps) <sup>2</sup>	GCLK	DCM/DLL	Slices	BRAM
Spartan-3™	XC3S1000-4	80	931	1	0	469	8
Virtex-2™	XC2V500-4	95	1105	1	0	467	8
Virtex-2 Pro™	XC2VP7-5	126	1466	1	0	464	8
Virtex-4™	XC4VLX25-11	174	2025	1	0	461	8

**Table 3:** Implementation statistics for the AES\_ENC\_DEC encryption/decryption core with external key expander.

FPGA Family	Example Device	F <sub>max</sub> (MHz) <sup>1</sup>	Data Throughput (Mbps) <sup>2</sup>	GCLK	DCM/DLL	Slices	BRAM
Spartan-3™	XC3S1000-4	75	873	1	0	593	8
Virtex-2™	XC2V500-4	91	1059	1	0	594	8
Virtex-2 Pro™	XC2VP7-5	115	1338	1	0	582	8
Virtex-4™	XC4VLX25-11	157	1827	1	0	681	8

**Table 4:** Implementation statistics for the KEY\_EXPANDER\_WITH\_STORAGE key expander core.

FPGA Family	Example Device	F <sub>max</sub> (MHz) <sup>1</sup>	Clock Cycles for Key Expansion <sup>3</sup>	GCLK	DCM/DLL	Slices	BRAM
Spartan-3™	XC3S1000-4	102	44/52/60	1	0	244	2
Virtex-2™	XC2V500-4	118	44/52/60	1	0	240	2
Virtex-2 Pro™	XC2VP7-5	161	44/52/60	1	0	225	2
Virtex-4™	XC4VLX25-11	187	44/52/60	1	0	247	2

Notes:

- 1) F<sub>max</sub> is quoted assuming all core inputs are sourced from registers and all core outputs drive registers. This has been done by putting a wrapper around the core to best represent real applications. The keep hierarchy flag has been set in the synthesis tool.
- 2) Quoted values are calculated for 128-bit keys. The maximum data throughput in ECB mode is as follows:  
Max Throughput (Mbps) = (128 / (Nr+1)) x Master Clock Frequency (MHz) where Nr=10, 12, 14 for 128/192/256-bit keys.
- 3) The number of clock cycles required for a complete key expansion depends on the key size and is N = 4\*(Nr+1), where Nr=10, 12, 14 for 128/192/256-bit keys.

## 4. Standard Encryption Core AES\_ENC

### 4.1. Functional Description

This entity is an AES encryption component that uses an external key expander. The component processes each round in a single clock cycle. Plain text input, round key schedule input and cipher text output ports are 128 bits wide.

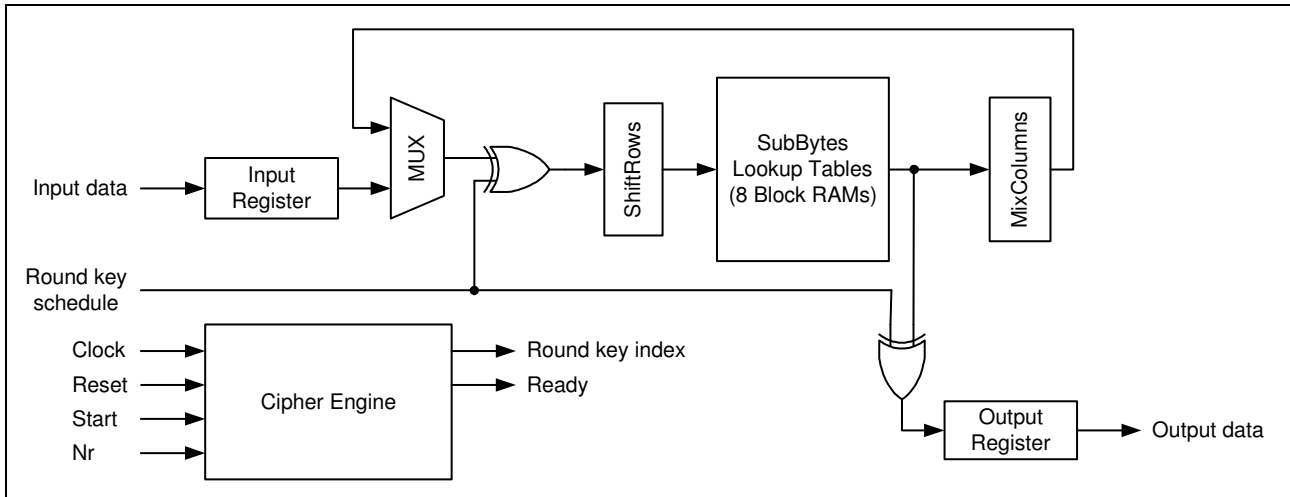


Figure 1: Block diagram of the AES\_ENC encryption core with external key expander.

When the start signal is asserted, input data is loaded and a new encryption operation is started. After a latency of 11, 13 or 15 master clock cycles (depending on the key size of 128, 192 or 256 bits) the ready signal is asserted and the cipher text output is valid. The round key index cycles through all needed values and is valid one clock cycle before the round key schedule data is required. This allows the use of external synchronous RAM to store the round key schedules.

A new encryption operation can be started whenever the round key index is zero. One clock cycle later the output of a previous operation becomes available.

### 4.2. Interface Signals

The port map of the encryption core is shown in Figure 2.

The master clock signal CLK drives the core at the rising edge. An active high synchronous reset RST is provided to initialize the component.

NR specifies the number of rounds. For a key size of 128-bits there are 10 rounds, for a 192-bit key there are 12 rounds, and for a 256-bit key there are 14 rounds.

Input START enables a register that stores DIN (the plain text) and initiates the encryption operation.

Output CT\_READY indicates valid DOUT (cipher text) and can be used to enable a register that stores DOUT.

RK\_VALUE must be provided by a register or RAM one clock cycle after RK\_INDEX.

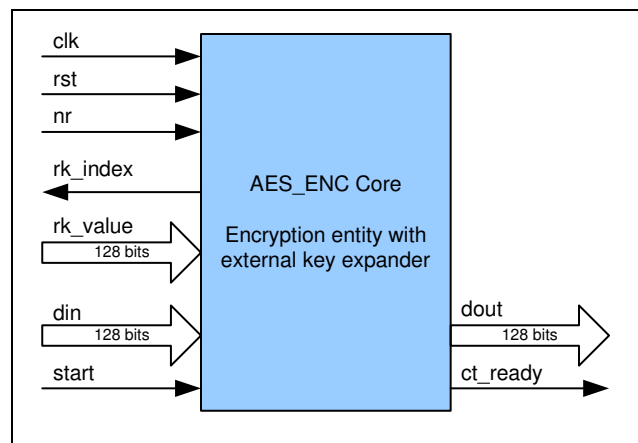


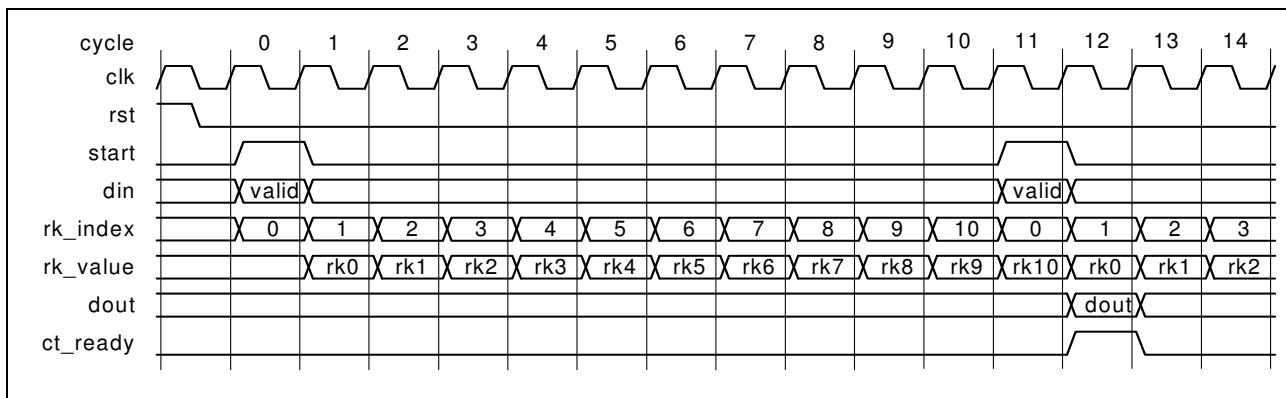
Figure 2: Port map of the AES\_ENC encryption core with external key expander.

**Table 5:** Port signals of the AES\_ENC encryption core with external key expander.

Signal	Width	I/O	Description
clk	1	I	<b>Clock:</b> Master clock signal that drives the core at the rising edge. All signals are synchronous with respect to the rising edge of this clock.
rst	1	I	<b>Reset:</b> Active high synchronous reset. It is provided to initialize the component any time after power on reset.
nr	4	I	<b>Number of rounds:</b> The number of rounds performed depending on the used key size. For a key size of 128-bits there are 10 rounds, for a 192-bit key there are 12 rounds, and for a 256-bit key there are 14 rounds.  <b>Note:</b> Only values 10, 12 or 14 are valid. Device operation is not specified for other values.
rk_index	4	O	<b>Round key index:</b> The number of the next round key schedule that is needed by the encryption component. The index is the address into an external RAM that stores the round key schedules or can be connected with the key expander directly.
rk_value	128	I	<b>Round key value:</b> This is the 128-bit round key schedule for the current round and must be provided by a register or RAM one clock cycle after RK_INDEX.
din	128	I	<b>Input data:</b> 128-bit plain text input data. It must be valid at the same time as START is asserted.
start	1	I	<b>Start:</b> An active high input that enables a register that stores the plain text input DIN and initiates the encryption operation.
dout	128	O	<b>Output data:</b> 128-bit cipher text output data.
ct_ready	1	O	<b>Cipher text ready:</b> An active high output that indicates valid cipher text output DOUT and can be used to enable a register that stores the output data.

### 4.3. Timing Diagram

The timing diagram of the encryption core is shown in Figure 3 below.



**Figure 3:** Timing diagram of the AES\_ENC encryption core with external key expander. Shown is the timing for a 128-bit key. For 192/256-bit keys the ct\_ready output goes high in cycle 14 or 16, respectively.

## 4.4. VHDL Entity Declaration

The VHDL entity declaration of the AES\_ENC encryption core is shown in Figure 4 below.

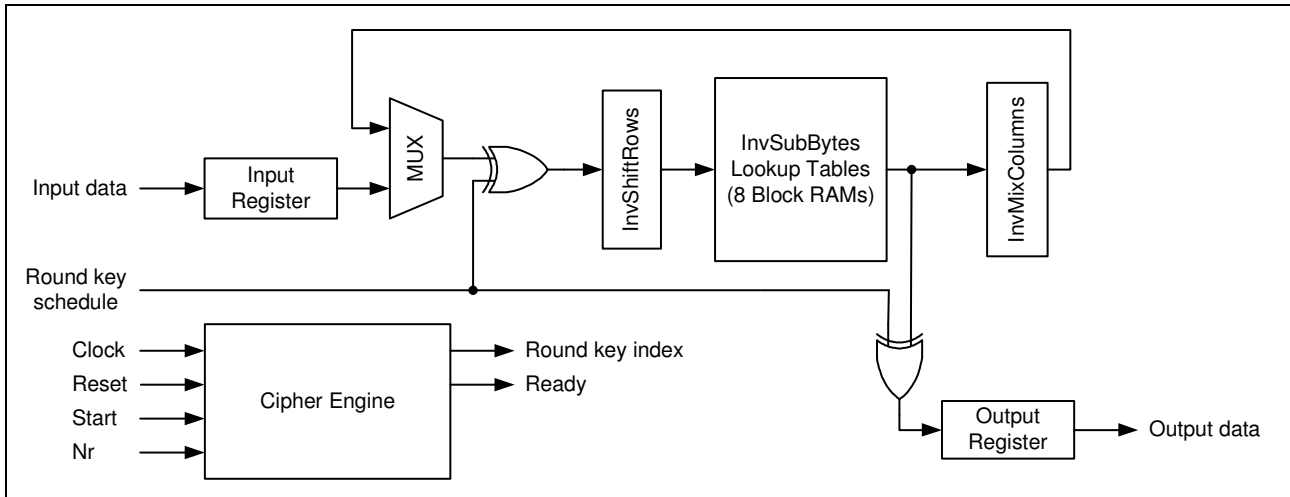
```
entity AES_ENC is
  port (
    clk      : in  std_logic;           -- master clock, rising edge
    rst      : in  std_logic;           -- active high reset
    nr       : in  std_logic_vector(3 downto 0); -- number of AES rounds (10, 12 or 14)
    din      : in  std_logic_vector(0 to 127); -- plain text input, 16 bytes
    start    : in  std_logic;           -- load plain text and start new encryption
    dout     : out std_logic_vector(0 to 127); -- cipher text output, 16 bytes
    ct_ready : out std_logic;           -- indicates that cipher text is ready
    rk_index : out std_logic_vector(3 downto 0); -- round key index
    rk_value : in  std_logic_vector(0 to 127)); -- round key value
end AES_ENC;
```

**Figure 4:** VHDL entity declaration of the AES\_ENC encryption core with external key expander.

## 5. Standard Decryption Core AES\_DEC

### 5.1. Functional Description

This entity is an AES decryption component that uses an external key expander. The component processes each round in a single clock cycle. Cipher text input, round key schedule input and plain text output ports are 128 bits wide.



**Figure 5:** Block diagram of the AES\_DEC decryption core with external key expander.

When the start signal is asserted, input data is loaded and a new decryption operation is started. After a latency of 11, 13 or 15 master clock cycles (depending on the key size of 128, 192 or 256 bits) the ready signal is asserted and the plain text output is valid. The round key index cycles through all needed values and is valid one clock cycle before the round key schedule data is required. This allows the use of external synchronous RAM to store the round key schedules.

A new decryption operation can be started whenever the round key index is zero. One clock cycle later the output of a previous operation becomes available.

### 5.2. Interface Signals

The port map of the decryption core is shown in Figure 6.

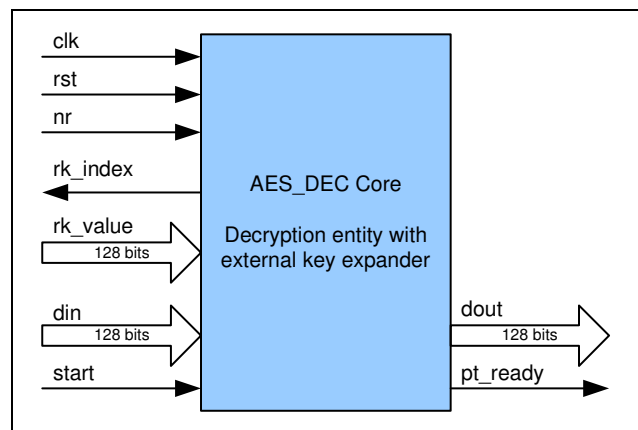
The master clock signal CLK drives the core at the rising edge. An active high synchronous reset RST is provided to initialize the component.

NR specifies the number of rounds. For a key size of 128-bits there are 10 rounds, for a 192-bit key there are 12 rounds, and for a 256-bit key there are 14 rounds.

Input START enables a register that stores DIN (the cipher text) and initiates the decryption operation.

Output PT\_READY indicates valid DOUT (plain text) and can be used to enable a register that stores DOUT.

RK\_VALUE must be provided by a register or RAM one clock cycle after RK\_INDEX.



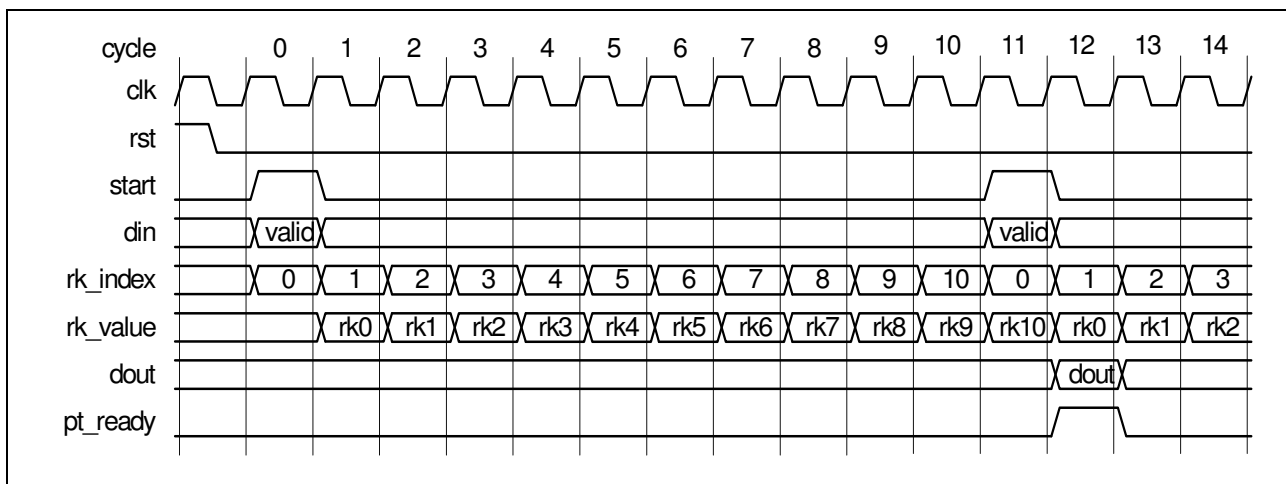
**Figure 6:** Port map of the AES\_DEC decryption core with external key expander.

**Table 6:** Port signals of the AES\_DEC decryption core with external key expander.

Signal	Width	I/O	Description
clk	1	I	<b>Clock:</b> Master clock signal that drives the core at the rising edge. All signals are synchronous with respect to the rising edge of this clock.
rst	1	I	<b>Reset:</b> Active high synchronous reset. It is provided to initialize the component any time after power on reset.
nr	4	I	<b>Number of rounds:</b> The number of rounds performed depending on the used key size. For a key size of 128-bits there are 10 rounds, for a 192-bit key there are 12 rounds, and for a 256-bit key there are 14 rounds.  <b>Note:</b> Only values 10, 12 or 14 are valid. Device operation is not specified for other values.
rk_index	4	O	<b>Round key index:</b> The number of the next round key schedule that is needed by the decryption component. The index is the address into an external RAM that stores the round key schedules or can be connected with the key expander directly.
rk_value	128	I	<b>Round key value:</b> This is the 128-bit round key schedule for the current round and must be provided by a register or RAM one clock cycle after RK_INDEX.
din	128	I	<b>Input data:</b> 128-bit cipher text input data. It must be valid at the same time as START is asserted.
start	1	I	<b>Start:</b> An active high input that enables a register that stores the cipher text input DIN and initiates the decryption operation.
dout	128	O	<b>Output data:</b> 128-bit plain text output data.
pt_ready	1	O	<b>Plain text ready:</b> An active high output that indicates valid plain text output DOUT and can be used to enable a register that stores the output data.

### 5.3. Timing Diagram

The timing diagram of the decryption core is shown in Figure 7 below.



**Figure 7:** Timing diagram of the AES\_DEC decryption core with external key expander. Shown is the timing for a 128-bit key. For 192/256-bit keys the pt\_ready output goes high in cycle 14 or 16, respectively.

## 5.4. VHDL Entity Declaration

The VHDL entity declaration of the AES\_DEC decryption core is shown in Figure 8 below.

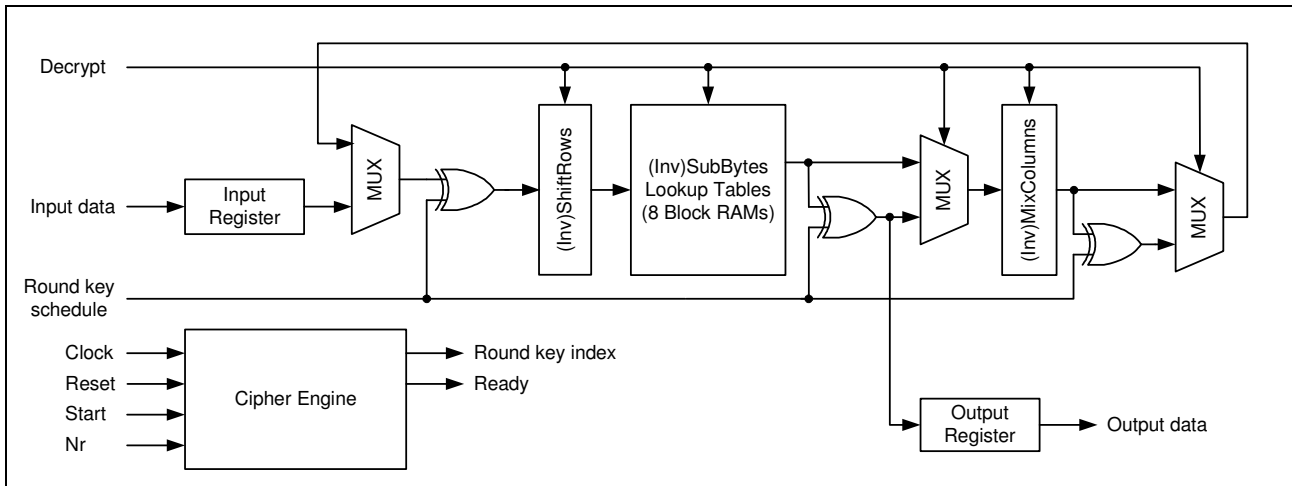
```
entity AES_DEC is
  port (
    clk      : in  std_logic;           -- master clock, rising edge
    rst      : in  std_logic;           -- active high reset
    nr       : in  std_logic_vector(3 downto 0); -- number of AES rounds (10, 12 or 14)
    din      : in  std_logic_vector(0 to 127); -- cipher text input, 16 bytes
    start    : in  std_logic;           -- load plain text and start new decryption
    dout     : out std_logic_vector(0 to 127); -- plain text output, 16 bytes
    pt_ready : out std_logic;           -- indicates that plain text is ready
    rk_index : out std_logic_vector(3 downto 0); -- round key index
    rk_value : in  std_logic_vector(0 to 127)); -- round key value
end AES_DEC;
```

**Figure 8:** VHDL entity declaration of the AES\_DEC decryption core with external key expander.

## 6. Standard Encryption/Decryption Core AES\_ENC\_DEC

### 6.1. Functional Description

This entity is a combined encryption/decryption component with external key expander. The component processes each round in a single clock cycle. Plain/cipher text input, round key schedule input and cipher/plain text output ports are 128 bits wide.



**Figure 9:** Block diagram of the AES\_ENC\_DEC encryption/decryption core with external key expander.

When the start signal is asserted, input data is loaded and a new operation is started. Depending on the state of a select signal the operation is either encryption or decryption.

After a latency of 11, 13 or 15 master clock cycles (depending on the key size of 128, 192 or 256 bits) the ready signal is asserted and the plain text output is valid. The round key index cycles through all needed values and is valid one clock cycle before the round key schedule data is required. This allows the use of external synchronous RAM to store the round key schedules.

A new operation can be started whenever the round key index is zero. One clock cycle later the output of a previous operation becomes available.

### 6.2. Interface Signals

The port map of the decryption core is shown in Figure 10.

The master clock signal CLK drives the core at the rising edge. An active high synchronous reset RST is provided to initialize the component.

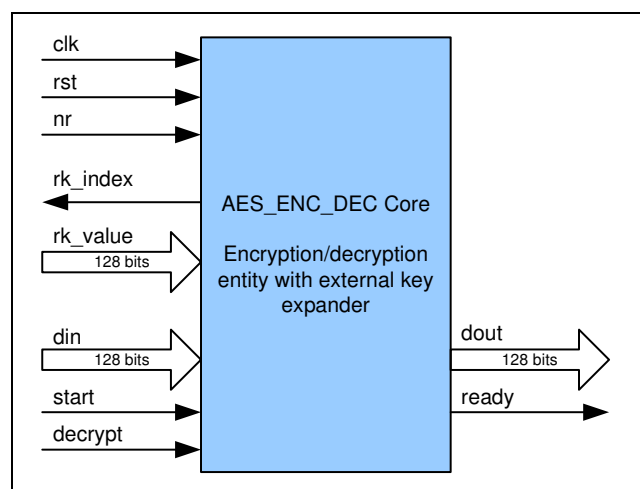
NR specifies the number of rounds. For a key size of 128-bits there are 10 rounds, for a 192-bit key there are 12 rounds, and for a 256-bit key there are 14 rounds.

Input START enables a register that stores DIN (the cipher text) and initiates the operation.

DECRYPT selects encryption (when '0') or decryption (when '1') operation.

Output READY indicates valid DOUT (cipher text for encryption or plain text for decryption operation) and can be used to enable a register that stores DOUT.

RK\_VALUE must be provided by a register or RAM one clock cycle after RK\_INDEX.



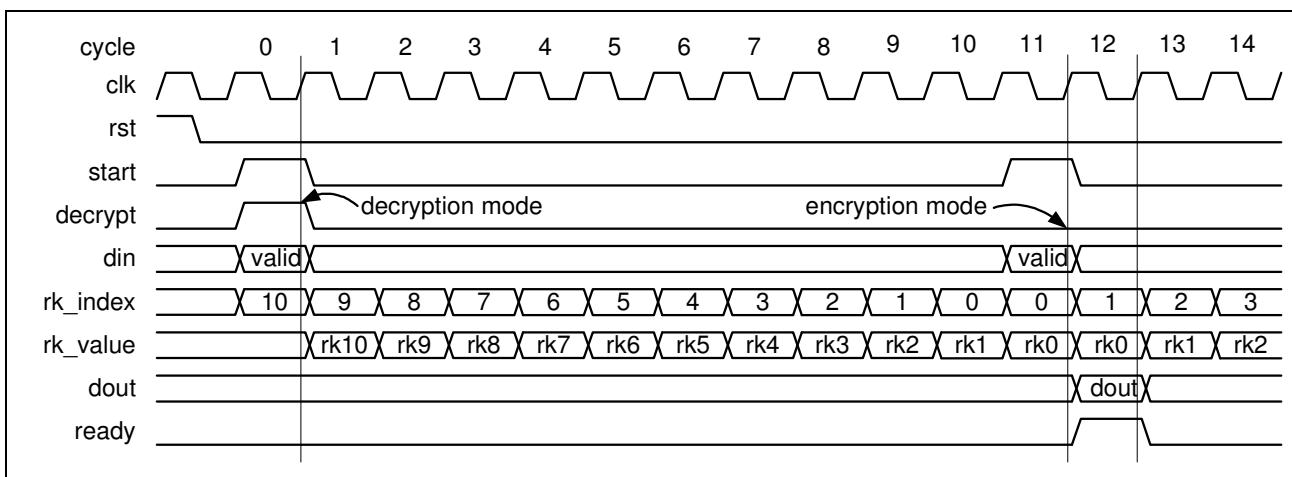
**Figure 10:** Port map of the AES\_ENC\_DEC encryption/decryption core with external key expander.

**Table 7:** Port signals of the AES\_ENC\_DEC encryption/decryption core with external key expander.

Signal	Width	I/O	Description
clk	1	I	<b>Clock:</b> Master clock signal that drives the core at the rising edge. All signals are synchronous with respect to the rising edge of this clock.
rst	1	I	<b>Reset:</b> Active high synchronous reset. It is provided to initialize the component any time after power on reset.
nr	4	I	<b>Number of rounds:</b> The number of rounds performed depending on the used key size. For a key size of 128-bits there are 10 rounds, for a 192-bit key there are 12 rounds, and for a 256-bit key there are 14 rounds.  <b>Note:</b> Only values 10, 12 or 14 are valid. Device operation is not specified for other values.
rk_index	4	O	<b>Round key index:</b> The number of the next round key schedule that is needed by the encryption/decryption component. The index is the address into an external RAM that stores the round key schedules or can be connected with the key expander directly.
rk_value	128	I	<b>Round key value:</b> This is the 128-bit round key schedule for the current round and must be provided by a register or RAM one clock cycle after RK_INDEX.
din	128	I	<b>Input data:</b> 128-bit plain/cipher text input data. It must be valid at the same time as START is asserted.
start	1	I	<b>Start:</b> An active high input that enables a register that stores the input data DIN and initiates the encryption or decryption operation.
decrypt	1	I	<b>Decrypt:</b> Selects encryption or decryption operation: '0': encryption mode '1': decryption mode
dout	128	O	<b>Output data:</b> 128-bit cipher/plain text output data.
ready	1	O	<b>Ready:</b> An active high output that indicates valid cipher or plain text output DOUT and can be used to enable a register that stores the output data.

### 6.3. Timing Diagram

The timing diagram of the decryption core is shown in Figure 11 below.



**Figure 11:** Timing diagram of the AES\_ENC\_DEC encryption/decryption core with external key expander. Shown is the timing for a 128-bit key. For 192/256-bit keys the ready output goes high in cycle 14 or 16, respectively.

## 6.4. VHDL Entity Declaration

The VHDL entity declaration of the AES\_ENC\_DEC encryption/decryption core is shown in Figure 12 below.

```
entity AES_ENC_DEC is
  port (
    clk      : in  std_logic;           -- master clock, rising edge
    rst      : in  std_logic;           -- active high reset
    nr       : in  std_logic_vector(3 downto 0); -- number of AES rounds (10, 12 or 14)
    din      : in  std_logic_vector(0 to 127); -- plain/cipher text input, 16 bytes
    decrypt  : in  std_logic;           -- select decryption when '1' else encryption
    start    : in  std_logic;           -- load input data and start new encryption
    dout     : out std_logic_vector(0 to 127); -- cipher/plain text output, 16 bytes
    ready    : out std_logic;           -- indicates that result is ready
    rk_index : out std_logic_vector(3 downto 0); -- round key index
    rk_value : in  std_logic_vector(0 to 127)); -- round key value
end AES_ENC_DEC;
```

Figure 12: VHDL entity declaration of the AES\_ENC\_DEC encryption/decryption core with external key expander.

## 7. Standard Key Expander Core KEY\_EXPANDER\_WITH\_STORAGE

### 7.1. Functional Description

This entity implements a key expander for 128/192/256-bit AES keys. The key is entered as a series of 32-bit words. All round keys are stored in internal dual port RAM which can be accessed directly from the encryption and decryption entities.

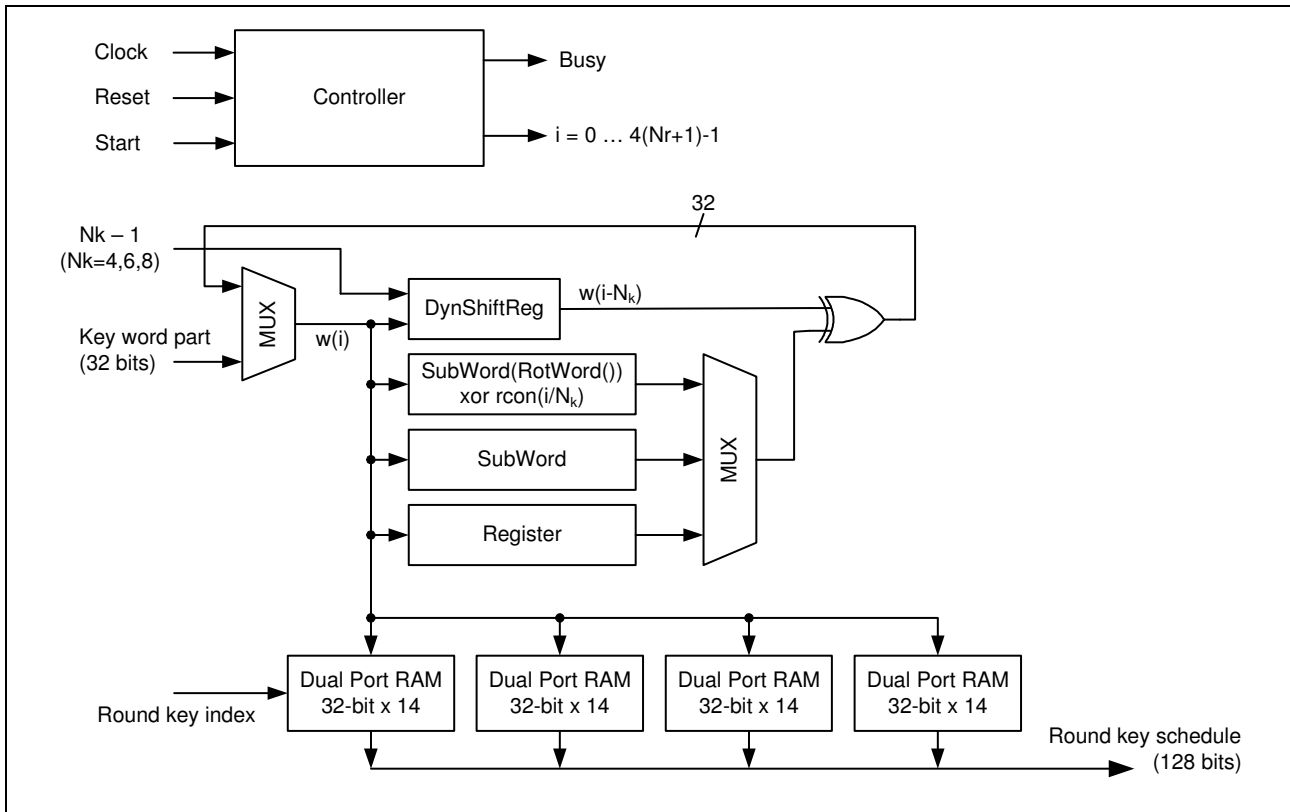


Figure 13: Block diagram of the KEY\_EXPANDER\_WITH\_STORAGE component.

When the start signal is asserted, the first 32-bit key word part is loaded and a new expansion operation is started. The remaining key word parts are loaded in the following cycles until the whole key is loaded.

A new operation can be started whenever the busy output is inactive (low).

## 7.2. Interface Signals

The master clock signal CLK drives the core at the rising edge. An active high synchronous reset RST is provided to initialize the component.

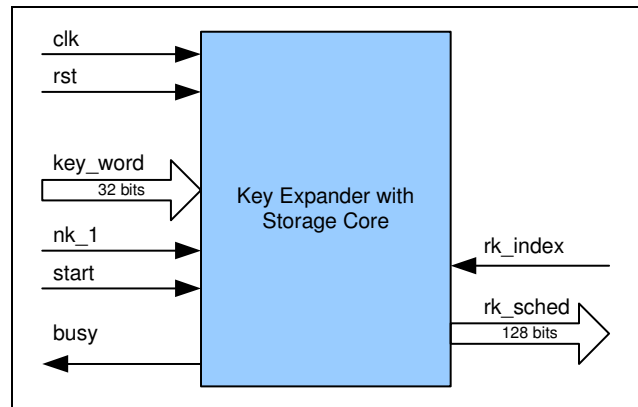
KEY\_WORD is a 32-bit part of the key. The complete key is entered into the key expander by feeding 4, 6 or 8 32-bit parts during consecutive clock cycles.

NK\_1 specifies the number of key parts minus one. For a key size of 128-bits there are 4 key parts (NK\_1=3), for a 192-bit key there are 6 key parts (NK\_1=5) and for a 256-bit key there are 8 key parts (NK\_1=7).

Input START registers the first key word part and starts the key expansion. The BUSY output goes high to indicate an ongoing key expansion process one clock cycle later.

BUSY stays active high during key expansion. After busy goes inactive low again, all expanded key schedules have been stored in the internal RAM.

The external round key schedule access is controlled by RK\_INDEX which addresses the appropriate key schedule. The round key schedule is provided by the internal RAM and is available on the RK\_SCHED output one clock cycle later.



**Figure 14:** Port map of the KEY\_EXPANDER\_WITH\_STORAGE component.

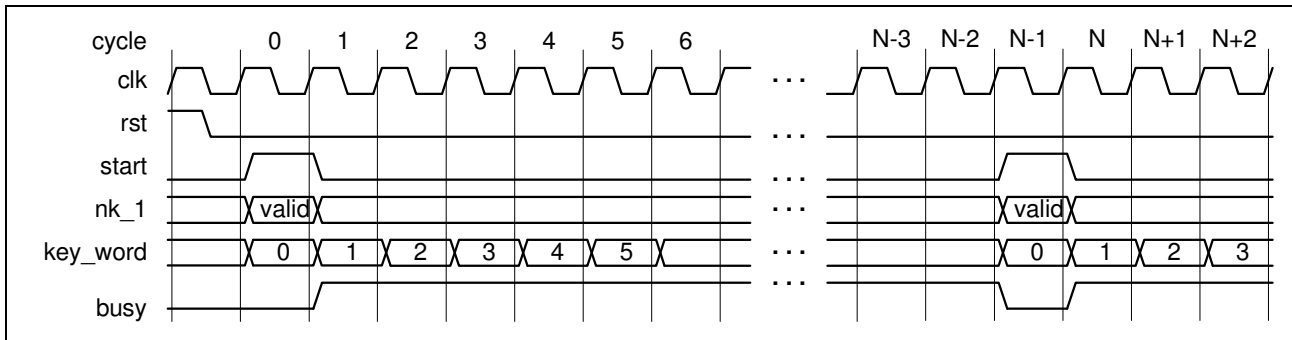
**Table 8:** Port signals of the AES\_ENC\_DEC encryption/decryption core with external key expander.

Signal	Width	I/O	Description
clk	1	I	<b>Clock:</b> Master clock signal that drives the core at the rising edge. All signals are synchronous with respect to the rising edge of this clock.
rst	1	I	<b>Reset:</b> Active high synchronous reset. It is provided to initialize the component any time after power on reset.
nk_1	3	I	<b>Number of key word parts:</b> The number of 32-bit key word parts minus one. For a key size of 128-bits there are 4 parts (nk_1=3), for a 192-bit key there are 6 parts (nk_1=5) and for a 256-bit key there are 8 parts (nk_1=7). <b>Note:</b> Only values 3, 5 or 7 are valid. Device operation is not specified for other values.
key_word	32	I	<b>Key word part:</b> This is a 32-bit key word part of the cipher key to be expanded.
start	1	I	<b>Start:</b> An active high input that starts the key expansion.
busy	1	O	<b>Busy:</b> An active high output that indicates that key expansion is going on.
rk_index	4	I	<b>Round key index:</b> The number of the next round key schedule that is needed by the encryption/decryption component. The index is the address into the internal RAM that stores the round key schedules.
rk_sched	128	O	<b>Round key schedule:</b> This is the 128-bit round key schedule for the current round and is provided by the internal RAM one clock cycle after RK_INDEX.

### 7.3. Timing Diagram

The timing diagram of the key expander core is shown in Figure 15 below.

The number of clock cycles required for a complete key expansion depends on the key size and is  $N = 4 \cdot (N_r + 1)$ , where  $N_r = 10, 12, 14$  for 128/192/256-bit keys. The key size is measured in multiples of 32-bit words denoted with  $NK$ .  $NK - 1$  ( $= 3, 5$  or  $7$ ) selects the key size and must be valid when  $START$  is active.



**Figure 15:** Timing diagram of the KEY\_EXPANDER\_WITH\_STORAGE key expander with storage core. The timing shown is for a 192-bit key with six 32-bit parts.

### 7.4. VHDL Entity Declaration

The VHDL entity declaration of the KEY\_EXPANDER\_WITH\_STORAGE key expander core is shown in Figure 16 below.

```
entity KEY_EXPANDER_WITH_STORAGE is
  port (
    clk      : in  std_logic;           -- master clock
    rst      : in  std_logic;           -- asynchronous reset (active high)

    -- key expansion
    key_word : in  std_logic_vector(0 to 31); -- 32-bit word part of key
    nk_1     : in  std_logic_vector(2 downto 0); -- key size Nk-1 (Nk=4,6,8)
    start    : in  std_logic;           -- start key expansion (active high)
    busy     : out std_logic;           -- key expander is busy (active high)

    -- round key schedule access
    rk_index : in  std_logic_vector(3 downto 0); -- key schedule index
    rk_sched : out std_logic_vector(0 to 127)); -- round key schedule
end KEY_EXPANDER_WITH_STORAGE;
```

**Figure 16:** VHDL entity declaration of the KEY\_EXPANDER\_WITH\_STORAGE key expander core.



## 8. Test Bench

All AES cores have been thoroughly verified under simulation using the NIST FIPS submission test vectors for known answer tests and using random data for Monte Carlo test procedures. The test suite and random data logs are also supplied as part of the core deliverables to demonstrate operation.

All tests were performed on the functional level using the RTL VHDL description and on the gate level using a post-place & route simulation model.

### 8.1. Known Answer Tests

These are basic tests that compare the result with the known answer. They are used to verify the basic operation of each component. A log file that lists the used keys, input and output data is produced for each test block.

### 8.2. Random Tests

These tests generate a series of random keys and plain texts using a simple linear feedback shift register random bit generator with primitive polynomial  $p(x) = x^{60} + x + 1$ . Note that although this technique has good statistical properties and leads to very efficient hardware implementations, the polynomial can be deduced from the output sequence, making it unsuitable for cryptographic applications. The primary purpose here is to generate sufficiently random test data.

The plain text is first encrypted with the encryption component giving an intermediate cipher text. This data is then decrypted with the decryption component to give a resulting plain text. The test then verifies that the decrypted cipher text is equal to the original plain text.

Several blocks perform these random tests for 128/192/256-bit keys separately. Random keys and plain texts, as well as the intermediate cipher texts are reported in a log file for later reference.

## 9. Deliverables

The deliverables of the standard AES encryption/decryption IP core suite consist of a set of files that are structured as follows:

Folder and File Structure	Contents Description
<ul style="list-style-type: none"> <li>▶ <b>bench</b></li> <li> <ul style="list-style-type: none"> <li>▶ <b>vhdl</b></li> <li> <ul style="list-style-type: none"> <li>tb_aes.vhd</li> <li>kat_tables.vhd</li> </ul> </li> </ul> </li> <li>▶ <b>doc</b></li> <li> <ul style="list-style-type: none"> <li>aes_standard_cores.pdf</li> </ul> </li> <li>▶ <b>rtl</b></li> <li> <ul style="list-style-type: none"> <li>▶ <b>vhdl</b></li> <li> <ul style="list-style-type: none"> <li>aes_enc.vhd</li> <li>aes_dec.vhd</li> <li>aes_enc_dec.vhd</li> <li>key_expander_with_storage.vhd</li> </ul> </li> </ul> </li> <li>▶ <b>syn</b></li> <li> <ul style="list-style-type: none"> <li>▶ <b>xilinx</b></li> <li> <ul style="list-style-type: none"> <li>▶ <b>spartan3</b></li> <li> <ul style="list-style-type: none"> <li>aes_enc.ngc</li> <li>aes_dec.ngc</li> <li>aes_enc_dec.ngc</li> <li>key_expander_with_storage.ngc</li> </ul> </li> </ul> </li> <li> <ul style="list-style-type: none"> <li>▶ <b>virtex2</b></li> <li>...</li> </ul> </li> <li> <ul style="list-style-type: none"> <li>▶ <b>virtex2pro</b></li> <li>...</li> </ul> </li> <li> <ul style="list-style-type: none"> <li>▶ <b>virtex4</b></li> <li> <ul style="list-style-type: none"> <li>▶ <b>FX</b></li> <li>...</li> <li>▶ <b>LX</b></li> <li>...</li> <li>▶ <b>SX</b></li> <li>...</li> </ul> </li> </ul> </li> </ul> </li> </ul>	<p><b>Test bench</b></p> <p><b>VHDL code of the test bench</b></p> <p>Test bench is used to simulate the behavior of all components Tables of Known Answer Tests for Electronic Codebook (ECB) Mode</p> <p><b>User documentation</b></p> <p>Complete data sheet (this document)</p> <p><b>Synthesizable RTL descriptions of the cores</b></p> <p><b>VHDL code of the cores</b></p> <p>AES encryption component using an external key expander AES decryption component using an external key expander AES encryption/decryption component using an external key expander Key expander with storage</p> <p><b>Synthesized code of the cores</b></p> <p><b>Xilinx specific netlists</b></p> <p><b>NGC netlists for Spartan-3 FPGAs</b></p> <p>AES encryption component using an external key expander AES decryption component using an external key expander AES encryption/decryption component using an external key expander Key expander with storage</p> <p><b>Netlists for Virtex-2 FPGAs</b></p> <p>NGC netlists for all cores</p> <p><b>Netlists for Virtex-2 Pro FPGAs</b></p> <p>NGC netlists for all cores</p> <p><b>Netlists for Virtex-4 FPGAs</b></p> <p>NGC netlists of all cores for Virtex-4 FX family</p> <p>NGC netlists of all cores for Virtex-4 LX family</p> <p>NGC netlists of all cores for Virtex-4 SX family</p>

The exact extent of the deliverables depends on the license type. E.g. VHDL code is not present when only netlists are purchased.



## 10. Export

Strong encryption technology such as AES is governed internationally by export regulations. Please note that licensees are responsible for complying with the applicable requirements for re-export of electronics containing AES technology.

Immediate export from Switzerland is permitted to the following countries (according to Annex4 of the Good Control Ordinance 946.202.1):

Argentina	Germany	Italy	Norway	Czech Republic
Australia	Finland	Japan	Poland	Turkey
Austria	France	Canada	Portugal	Hungary
Belgium	Greece	Luxembourg	Sweden	Ukraine
Bulgaria	Great Britain	New Zealand	Spain	United States of America
Denmark	Ireland	Netherlands	South Korea	

## 11. Related Information

### 11.1. Advanced Encryption Standard Specification

More detailed information about the Advanced Encryption Standard algorithm can be found in the original specification (NIST FIPS Publication 197). The document may be downloaded from the NIST website at

<http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>

### 11.2. Xilinx FPGAs and Software

For information on Xilinx FPGAs or development system software, contact your local Xilinx sales office, or see

<http://www.xilinx.com>