Sha256

Generated by Doxygen 1.8.18

1 Namespace Index **1**

1.1 Packages 1

2 Hierarchical Index **3**

2.1 Design Unit Hierarchy 3

3 Design Unit Index **5**

3.1 Design Unit List 5

4 File Index **7**

4.1 File List 7

5 Namespace Documentation **9**

5.1 constants Namespace Reference 9

5.1.1 Detailed Description 9

5.2 sha_function Namespace Reference 9

5.2.1 Detailed Description 9

6 Class Documentation **11**

6.1 sha_function Package Body Reference 11

6.1.1 Detailed Description 12

6.1.2 Member Function Documentation 12

6.1.2.1 adding() 12

6.1.2.2 CH() 12

6.1.2.3 code_a() 13

6.1.2.4 code_e() 13

6.1.2.5 EP0() 13

6.1.2.6 EP1() 13

6.1.2.7 MAJ() 13

6.1.2.8 SIG0() 14

6.1.2.9 SIG1() 14

6.2 behavior Architecture Reference 14

6.2.1 Member Function Documentation 15

6.2.1.1 COMMANDER() 16

6.2.1.2 to_std_logic_vector() 16

6.2.1.3 TRANSMITTER() 16

6.2.2 Member Data Documentation 16

6.2.2.1 BAUD 16

6.2.2.2 BYTE_FILE_TYPE 16

6.2.2.3 char_buffer 17

6.2.2.4 Clk 17

6.2.2.5 CLK_FREQUENCY 17

6.2.2.6 command_buffer 17

6.2.2.7 command_counter 17

6.2.2.8 Data_File 17
6.2.2.9 data_length 18
6.2.2.10 DATA_WIDTH 18
6.2.2.11 f_finish 18
6.2.2.12 f_full 18
6.2.2.13 file_buffer 18
6.2.2.14 FILENAME 18
6.2.2.15 Reset 19
6.2.2.16 Rx 19
6.2.2.17 state 19
6.2.2.18 STATE_TYPE 19
6.2.2.19 state_uart 19
6.2.2.20 Tx 19
6.2.2.21 uart_counter 19
6.2.2.22 uart_data 20
6.2.2.23 uart_enable 20
6.2.2.24 UART_STATE_TYPE 20
6.2.2.25 uut 20
6.3 behavior Architecture Reference 20
6.3.1 Member Function Documentation 21
6.3.1.1 Clk_process() 21
6.3.1.2 stim_proc() 21
6.3.2 Member Data Documentation 21
6.3.2.1 Clk 22
6.3.2.2 Clk_period 22
6.3.2.3 Hash_input 22
6.3.2.4 Hash_ready 22
6.3.2.5 Reset 22
6.3.2.6 Tx 22
6.3.2.7 uut 22
6.4 behavior Architecture Reference 23
6.4.1 Member Function Documentation 23
6.4.1.1 Clk_process() 23
6.4.1.2 stim_proc() 24
6.4.2 Member Data Documentation 24
6.4.2.1 able 24
6.4.2.2 Clk 24
6.4.2.3 Clk_period 24
6.4.2.4 Reset 24
6.4.2.5 Tx 24
6.4.2.6 TX_Data_In 25
6.4.2.7 TX_Ready 25

6.4.2.8 TX_Start 25
6.4.2.9 uut 25
6.5 Behavioral Architecture Reference 25
6.5.1 Member Function Documentation 26
6.5.1.1 PROCESS_1() 26
6.5.2 Member Data Documentation 26
6.5.2.1 bit_counter 26
6.5.2.2 f_ready 26
6.5.2.3 state 27
6.5.2.4 STATE_TYPE 27
6.5.2.5 word_buffer 27
6.5.2.6 word_counter 27
6.6 Behavioral Architecture Reference 27
6.6.1 Member Function Documentation 28
6.6.1.1 PROCESS_0() 28
6.6.2 Member Data Documentation 28
6.6.2.1 a 28
6.6.2.2 b 28
6.6.2.3 c 29
6.6.2.4 d 29
6.6.2.5 e 29
6.6.2.6 f 29
6.6.2.7 g 29
6.6.2.8 h 29
6.6.2.9 hash 29
6.6.2.10 state 30
6.6.2.11 STATE_TYPE 30
6.6.2.12 working_vars 30
6.7 Behavioral Architecture Reference 30
6.7.1 Member Function Documentation 31
6.7.1.1 OUTPUT() 31
6.7.2 Member Data Documentation 31
6.7.2.1 ack 31
6.7.2.2 schedule 31
6.7.2.3 word_t 31
6.8 Behavioral Architecture Reference 31
6.8.1 Member Function Documentation 32
6.8.1.1 ascii_to_integer() 32
6.8.1.2 RX_COMMANDER() 32
6.8.2 Member Data Documentation 33
6.8.2.1 counter 33
6.8.2.2 data_buffer 33

6.8.2.3 f_ready 33
6.8.2.4 state 33
6.8.2.5 STATE_TYPE 33
6.9 Behavioral Architecture Reference 34
6.9.1 Member Data Documentation 34
6.9.1.1 Clk 34
6.9.1.2 Hash 35
6.9.1.3 Hash_ready 35
6.9.1.4 Reset 35
6.9.1.5 Reset_All 35
6.9.1.6 Rx 35
6.9.1.7 RX_Data 35
6.9.1.8 RX_Ready 35
6.9.1.9 s1 36
6.9.1.10 S1_Word_id 36
6.9.1.11 S1_Word_output 36
6.9.1.12 S1_Word_ready 36
6.9.1.13 s2 36
6.9.1.14 S2_Word_out_nr 36
6.9.1.15 S2_Word_output 36
6.9.1.16 s3 37
6.9.1.17 S3_Word_req_id 37
6.9.1.18 Tx 37
6.9.1.19 u1 37
6.9.1.20 u2 37
6.9.1.21 uc 37
6.9.1.22 UC_Output_data 37
6.9.1.23 UC_Output_finish 38
6.9.1.24 UC_Output_length 38
6.9.1.25 UC_Output_ready 38
6.10 Behavioral Architecture Reference 38
6.10.1 Member Function Documentation 38
6.10.1.1 PROCESS_2() 39
6.10.2 Member Data Documentation 39
6.10.2.1 new_d 39
6.10.2.2 TX_Data_Out 39
6.10.2.3 TX_Ready 39
6.10.2.4 TX_Start 39
6.10.2.5 uart_tx 39
6.11 Behavioral Architecture Reference 40
6.11.1 Member Function Documentation 40
6.11.1.1 RX_PROCESS() 40

6.11.1.2 TO_OUTPUT() . . . 40
6.11.2 Member Data Documentation . . . 41
6.11.2.1 count . . . 41
6.11.2.2 data_buf . . . 41
6.11.2.3 data_ready . . . 41
6.11.2.4 freq_count . . . 41
6.11.2.5 last_Rx . . . 41
6.11.2.6 MAX_FREQ_COUNT . . . 42
6.11.2.7 receiving . . . 42
6.12 Behavioral Architecture Reference . . . 42
6.12.1 Member Function Documentation . . . 42
6.12.1.1 TX_PROCESS() . . . 42
6.12.2 Member Data Documentation . . . 43
6.12.2.1 count . . . 43
6.12.2.2 freq_count . . . 43
6.12.2.3 MAX_FREQ_COUNT . . . 43
6.13 Behavioral Architecture Reference . . . 43
6.13.1 Member Function Documentation . . . 43
6.13.1.1 PROCESS_3() . . . 44
6.14 COMPUTE_HASH Entity Reference . . . 44
6.14.1 Member Data Documentation . . . 45
6.14.1.1 Clk . . . 45
6.14.1.2 constants . . . 45
6.14.1.3 Hash_output . . . 45
6.14.1.4 Hash_ready . . . 45
6.14.1.5 IEEE . . . 45
6.14.1.6 NUMERIC_STD . . . 45
6.14.1.7 Output_finish . . . 46
6.14.1.8 Reset . . . 46
6.14.1.9 sha_function . . . 46
6.14.1.10 STD_LOGIC_1164 . . . 46
6.14.1.11 Word_in_nr . . . 46
6.14.1.12 Word_input . . . 46
6.14.1.13 Word_req_id . . . 46
6.15 constants Package Reference . . . 47
6.15.1 Detailed Description . . . 48
6.15.2 Member Data Documentation . . . 48
6.15.2.1 constant_initials . . . 48
6.15.2.2 constant_values . . . 48
6.15.2.3 constant_values_sha256 . . . 48
6.15.2.4 DWORD . . . 48
6.15.2.5 hash_array . . . 49

6.15.2.6 IEEE 49
6.15.2.7 message_block 49
6.15.2.8 message_schedule 49
6.15.2.9 NUMERIC_STD 49
6.15.2.10 STD_LOGIC_1164 49
6.16 MAIN Entity Reference 50
6.16.1 Member Data Documentation 51
6.16.1.1 BAUD 51
6.16.1.2 CLK_FREQUENCY 51
6.16.1.3 Clk_input 51
6.16.1.4 constants 51
6.16.1.5 DATA_WIDTH 51
6.16.1.6 IEEE 51
6.16.1.7 NUMERIC_STD 52
6.16.1.8 Reset_input 52
6.16.1.9 Rx_input 52
6.16.1.10 sha_function 52
6.16.1.11 STD_LOGIC_1164 52
6.16.1.12 Tx_output 52
6.17 PADDING_MESSAGE Entity Reference 53
6.17.1 Member Data Documentation 54
6.17.1.1 Clk 54
6.17.1.2 constants 54
6.17.1.3 DATA_WIDTH 54
6.17.1.4 IEEE 54
6.17.1.5 Input_data 54
6.17.1.6 Input_finish 54
6.17.1.7 Input_length 55
6.17.1.8 Input_ready 55
6.17.1.9 NUMERIC_STD 55
6.17.1.10 Reset 55
6.17.1.11 sha_function 55
6.17.1.12 STD_LOGIC_1164 55
6.17.1.13 Word_id 55
6.17.1.14 Word_output 56
6.17.1.15 Word_ready 56
6.18 PREPARE_SCHEDULE Entity Reference 56
6.18.1 Member Data Documentation 57
6.18.1.1 Clk 57
6.18.1.2 constants 57
6.18.1.3 IEEE 57
6.18.1.4 NUMERIC_STD 57

6.18.1.5 Reset 57
6.18.1.6 sha_function 58
6.18.1.7 STD_LOGIC_1164 58
6.18.1.8 Word_in_id 58
6.18.1.9 Word_in_ready 58
6.18.1.10 Word_input 58
6.18.1.11 Word_out_nr 58
6.18.1.12 Word_output 58
6.18.1.13 Word_req_id 59
6.19 sha_function Package Reference 59
6.19.1 Detailed Description 60
6.19.2 Member Function Documentation 60
6.19.2.1 adding() 60
6.19.2.2 CH() 61
6.19.2.3 code_a() 61
6.19.2.4 code_e() 61
6.19.2.5 EP0() 61
6.19.2.6 EP1() 62
6.19.2.7 MAJ() 62
6.19.2.8 SIG0() 62
6.19.2.9 SIG1() 62
6.19.3 Member Data Documentation 62
6.19.3.1 constants 62
6.19.3.2 IEEE 63
6.19.3.3 numeric_std 63
6.19.3.4 STD_LOGIC_1164 63
6.20 SHA_TX Entity Reference 63
6.20.1 Detailed Description 64
6.20.2 Member Data Documentation 64
6.20.2.1 BAUD 64
6.20.2.2 Clk 65
6.20.2.3 CLK_FREQUENCY 65
6.20.2.4 constants 65
6.20.2.5 DATA_WIDTH 65
6.20.2.6 Hash_input 65
6.20.2.7 Hash_ready 65
6.20.2.8 IEEE 65
6.20.2.9 NUMERIC_STD 66
6.20.2.10 Reset 66
6.20.2.11 sha_function 66
6.20.2.12 STD_LOGIC_1164 66
6.20.2.13 Tx 66

6.21 test_main Entity Reference . . . 66
6.21.1 Member Data Documentation . . . 67
6.21.1.1 constants . . . 67
6.21.1.2 ieee . . . 67
6.21.1.3 numeric_std . . . 67
6.21.1.4 sha_function . . . 67
6.21.1.5 std . . . 68
6.21.1.6 std_logic_1164 . . . 68
6.21.1.7 std_logic_textio . . . 68
6.21.1.8 textio . . . 68
6.22 test_sha_tx Entity Reference . . . 68
6.22.1 Member Data Documentation . . . 69
6.22.1.1 constants . . . 69
6.22.1.2 ieee . . . 69
6.22.1.3 sha_function . . . 69
6.22.1.4 std_logic_1164 . . . 69
6.23 test_uart_tx Entity Reference . . . 70
6.23.1 Member Data Documentation . . . 70
6.23.1.1 ieee . . . 70
6.23.1.2 std_logic_1164 . . . 70
6.24 UART_COMMANDER Entity Reference . . . 71
6.24.1 Member Data Documentation . . . 72
6.24.1.1 Clk . . . 72
6.24.1.2 constants . . . 72
6.24.1.3 DATA_WIDTH . . . 72
6.24.1.4 IEEE . . . 72
6.24.1.5 NUMERIC_STD . . . 72
6.24.1.6 Output_data . . . 72
6.24.1.7 Output_finish . . . 73
6.24.1.8 Output_length . . . 73
6.24.1.9 Output_ready . . . 73
6.24.1.10 Reset . . . 73
6.24.1.11 Reset_all . . . 73
6.24.1.12 RX_Data . . . 73
6.24.1.13 RX_Ready . . . 74
6.24.1.14 sha_function . . . 74
6.24.1.15 STD_LOGIC_1164 . . . 74
6.25 UART_RX Entity Reference . . . 74
6.25.1 Detailed Description . . . 75
6.25.2 Member Data Documentation . . . 75
6.25.2.1 BAUD . . . 75
6.25.2.2 Clk . . . 76

6.25.2.3 CLK_FREQUENCY 76
6.25.2.4 constants 76
6.25.2.5 DATA_WIDTH 76
6.25.2.6 IEEE 76
6.25.2.7 NUMERIC_STD 76
6.25.2.8 Reset 76
6.25.2.9 Rx 77
6.25.2.10 RX_Data_Out 77
6.25.2.11 RX_Ready 77
6.25.2.12 sha_function 77
6.25.2.13 STD_LOGIC_1164 77
6.26 uart_tx Entity Reference 78
6.26.1 Detailed Description 79
6.26.2 Member Data Documentation 79
6.26.2.1 BAUD 79
6.26.2.2 Clk 79
6.26.2.3 CLK_FREQUENCY 79
6.26.2.4 constants 79
6.26.2.5 DATA_WIDTH 80
6.26.2.6 IEEE 80
6.26.2.7 NUMERIC_STD 80
6.26.2.8 Reset 80
6.26.2.9 sha_function 80
6.26.2.10 STD_LOGIC_1164 80
6.26.2.11 Tx 81
6.26.2.12 TX_Data_In 81
6.26.2.13 TX_Ready 81
6.26.2.14 TX_Start 81
6.27 WORD_T Entity Reference 81
6.27.1 Member Data Documentation 82
6.27.1.1 Ack 82
6.27.1.2 Clk 82
6.27.1.3 constants 83
6.27.1.4 IEEE 83
6.27.1.5 NUMERIC_STD 83
6.27.1.6 Ready 83
6.27.1.7 Reset 83
6.27.1.8 Schedule 83
6.27.1.9 sha_function 83
6.27.1.10 STD_LOGIC_1164 84
6.27.1.11 Word_id 84
6.27.1.12 Word_in 84

7 File Documentation **85**

7.1 constants.vhd File Reference 85

7.2 elbertv2_pin.ucf File Reference 85

7.2.1 Variable Documentation 85

7.2.1.1 ""Clk_input[0]"" 85

7.2.1.2 ""Rx_input[0]"" 85

7.2.1.3 ""Tx_output[0]"" 86

7.2.1.4 VCCAUX 86

7.3 main.vhd File Reference 86

7.4 sha_compute_hash.vhd File Reference 86

7.5 sha_function.vhd File Reference 86

7.6 sha_padding_message.vhd File Reference 86

7.7 sha_prepare_schedule.vhd File Reference 87

7.8 sha_tx.vhd File Reference 87

7.9 sha_word_t.vhd File Reference 87

7.10 test_main.vhd File Reference 87

7.11 test_sha_tx.vhd File Reference 87

7.12 test_uart_tx.vhd File Reference 87

7.13 uart_commander.vhd File Reference 88

7.14 uart_rx.vhd File Reference 88

7.15 uart_tx.vhd File Reference 88

Index **89**

Chapter 1

Namespace Index

1.1 Packages

Here are the packages with brief descriptions (if available):

constants
Packet with definitions of new types of values and constant variables 9
sha_function
Packet with definition function used in sha algorithm 9

Chapter 2

Hierarchical Index

2.1 Design Unit Hierarchy

Here is a hierarchical list of all entities:

test_main 66
    MAIN 50
        UART_RX 74
        SHA_TX 63
            uart_tx 78
        UART_COMMANDER 71
        PADDING_MESSAGE 53
        PREPARE_SCHEDULE 56
            WORD_T 81
        COMPUTE_HASH 44
test_sha_tx 68
    SHA_TX 63
test_uart_tx 70
    uart_tx 78

Chapter 3

Design Unit Index

3.1 Design Unit List

Here is a list of all design unit members with links to the Entities they belong to:

architecture behavior 14
architecture behavior 20
architecture behavior 23
architecture Behavioral 25
architecture Behavioral 27
architecture Behavioral 30
architecture Behavioral 31
architecture Behavioral 34
architecture Behavioral 38
architecture Behavioral 40
architecture Behavioral 42
architecture Behavioral 43
entity COMPUTE_HASH 44
entity MAIN 50
entity PADDING_MESSAGE 53
entity PREPARE_SCHEDULE 56
entity SHA_TX
    Divides the final sha result by eight bits and sending to uart 63
entity test_main 66
entity test_sha_tx 68
entity test_uart_tx 70
entity UART_COMMANDER 71
entity UART_RX
    Definition of UART RX 74
entity uart_tx
    Definition of UART TX 78
entity WORD_T 81

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

constants.vhd 85
elbertv2_pin.ucf 85
main.vhd 86
sha_compute_hash.vhd 86
sha_function.vhd 86
sha_padding_message.vhd 86
sha_prepare_schedule.vhd 87
sha_tx.vhd 87
sha_word_t.vhd 87
test_main.vhd 87
test_sha_tx.vhd 87
test_uart_tx.vhd 87
uart_commander.vhd 88
uart_rx.vhd 88
uart_tx.vhd 88

Chapter 5

Namespace Documentation

5.1 constants Namespace Reference

Packet with definitions of new types of values and constant variables.

5.1.1 Detailed Description

Packet with definitions of new types of values and constant variables.

5.2 sha_function Namespace Reference

Packet with definition function used in sha algorithm.

5.2.1 Detailed Description

Packet with definition function used in sha algorithm.

Chapter 6

Class Documentation

6.1 sha_function Package Body Reference

Hash values from the current iteration.

Package >> sha_function

Functions

- **std_logic_vector** CH**(**
  x: in std_logic_vector
  y: in std_logic_vector
  z: in std_logic_vector
)
- **std_logic_vector** MAJ**(**
  x: in std_logic_vector
  y: in std_logic_vector
  z: in std_logic_vector
)
- **std_logic_vector** EP0**(x: in std_logic_vector)**
- **std_logic_vector** EP1**(x: in std_logic_vector)**
- **std_logic_vector** SIG0**(x: in std_logic_vector)**
- **std_logic_vector** SIG1**(x: in std_logic_vector)**
- **std_logic_vector** code_e**(**
  h: in std_logic_vector
  e: in std_logic_vector
  f: in std_logic_vector
  g: in std_logic_vector
  d: in std_logic_vector
  M: in std_logic_vector
  K: in std_logic_vector
)

- **std_logic_vector** code_a**(**
  h: in std_logic_vector
  e: in std_logic_vector
  f: in std_logic_vector
  g: in std_logic_vector
  a: in std_logic_vector
  b: in std_logic_vector
  c: in std_logic_vector
  M: in std_logic_vector
  K: in std_logic_vector
)

Procedures

- adding(**signal h: inouthash_array,signal w_v: inhash_array**)

  Hash values from the previous iteration.

6.1.1 Detailed Description

Hash values from the current iteration.

6.1.2 Member Function Documentation

6.1.2.1 adding()

```
 adding(
             signal  h inouthash_array ,
             signal  w_v inhash_array )   [Procedure]
```

Hash values from the previous iteration.

6.1.2.2 CH()

```
std_logic_vector CH(
              xin std_logic_vector ,
              yin std_logic_vector ,
              zin std_logic_vector )   [Function]
```

6.1.2.3 code_a()

```
std_logic_vector code_a(
              hin std_logic_vector ,
              ein std_logic_vector ,
              fin std_logic_vector ,
              gin std_logic_vector ,
              ain std_logic_vector ,
              bin std_logic_vector ,
              cin std_logic_vector ,
              Min std_logic_vector ,
              Kin std_logic_vector )  [Function]
```

6.1.2.4 code_e()

```
std_logic_vector code_e(
              hin std_logic_vector ,
              ein std_logic_vector ,
              fin std_logic_vector ,
              gin std_logic_vector ,
              din std_logic_vector ,
              Min std_logic_vector ,
              Kin std_logic_vector )  [Function]
```

6.1.2.5 EP0()

```
std_logic_vector EP0(
              xin std_logic_vector )  [Function]
```

6.1.2.6 EP1()

```
std_logic_vector EP1(
              xin std_logic_vector )  [Function]
```

6.1.2.7 MAJ()

```
std_logic_vector MAJ(
              xin std_logic_vector ,
              yin std_logic_vector ,
              zin std_logic_vector )  [Function]
```

6.1.2.8 SIG0()

```
std_logic_vector SIG0(
            xin std_logic_vector )   [Function]
```

6.1.2.9 SIG1()

```
std_logic_vector SIG1(
            xin std_logic_vector )   [Function]
```

The documentation for this class was generated from the following file:

- sha_function.vhd

6.2 behavior Architecture Reference

Functions

- **std_logic_vector** to_std_logic_vector**(input: in string)**

    main process for reading files and sending data to work.MAIN function which changes string (ascii) to std_logic_vector

Processes

- COMMANDER**(Clk)**

    when file_buffer is not full yet
- TRANSMITTER**(Clk)**

    unlocks uart_enable at the end of uart transmission

Constants

- CLK_FREQUENCY **positive:= 12000000**

    constatnts
- DATA_WIDTH **positive:= 8**
- BAUD **positive:= 19200**
- FILENAME **string:=" example.bin "**

    name of the read file

Types

- BYTE_FILE_TYPE **fileofcharacter**

    file type, constant filename and buffer
- UART_STATE_TYPE **(IDLE,START,DATA,STOP)**

    uart transmitter for testing
- STATE_TYPE **(COMMAND,TRANSMIT,IDLE,STOP)**

    states of sending data to tested component

Signals

- Clk **std_logic:=' 0 '**

  IO signals of 'sha_main.vhd'.
- Reset **std_logic:=' 0 '**
- Rx **std_logic:=' 1 '**
- Tx **std_logic:=' 1 '**
- file_buffer **std_logic_vector(511 downto 0)**

  buffer for storing chunks of input file
- state_uart **UART_STATE_TYPE :=IDLE**

  determinates the state of the uart process
- uart_enable **std_logic:=' 0 '**

  locks pushing data to uart
- uart_data **std_logic_vector(0 toDATA_WIDTH - 1)**

  stores data for transmitting, 'to' is used for reverse trick on output assignment
- uart_counter **naturalrange 0 toDATA_WIDTH + 3 := 0**

  counts transmitted bits
- f_full **std_logic:=' 0 '**

  flag which states if file_buffer is full or finished
- f_finish **std_logic:=' 0 '**

  flag which states if all data was sent to MAIN module

Shared Variables

- data_length **integer:=:= 0**

  counts length of incoming data up to 512, used for dividing data into 512-bit long blocks
- char_buffer **character**

  byte long buffer for reading files
- state **STATE_TYPE :=:=IDLE**

  determinates the state of testing
- command_buffer **std_logic_vector(63 downto 0)**

  buffer which holds ascii commands in binary
- command_counter **integer**

  holds the length of command_buffer, downcounted while transmitting

Files

- Data_File **BYTE_FILE_TYPE openread_modeisFILENAME**

  file declaration in read mode

Instantiations

- uut **MAIN**

  main clock signal

6.2.1 Member Function Documentation

6.2.1.1 COMMANDER()

```
 COMMANDER(
              Clk   )  [Process]
```

when file_buffer is not full yet

6.2.1.2 to_std_logic_vector()

```
std_logic_vector to_std_logic_vector (
              string  input )
```

main process for reading files and sending data to work.MAIN function which changes string (ascii) to std_logic_↩
vector

6.2.1.3 TRANSMITTER()

```
 TRANSMITTER(
              Clk   )  [Process]
```

unlocks uart_enable at the end of uart transmission

6.2.2 Member Data Documentation

6.2.2.1 BAUD

```
BAUD positive:= 19200    [Constant]
```

6.2.2.2 BYTE_FILE_TYPE

```
BYTE_FILE_TYPE fileofcharacter   [Type]
```

file type, constant filename and buffer

6.2.2.3 char_buffer

`char_buffer character [Shared Variable]`

byte long buffer for reading files

6.2.2.4 Clk

`Clk std_logic:=' 0 ' [Signal]`

IO signals of 'sha_main.vhd'.

6.2.2.5 CLK_FREQUENCY

`CLK_FREQUENCY positive:= 12000000 [Constant]`

constatnts

6.2.2.6 command_buffer

`command_buffer std_logic_vector( 63 downto 0 ) [Shared Variable]`

buffer which holds ascii commands in binary

6.2.2.7 command_counter

`command_counter integer [Shared Variable]`

holds the length of command_buffer, downcounted while transmitting

6.2.2.8 Data_File

`Data_File BYTE_FILE_TYPE openread_modeisFILENAME [File]`

file declaration in read mode

6.2.2.9 data_length

```
data_length integer:=:= 0    [Shared Variable]
```

counts length of incoming data up to 512, used for dividing data into 512-bit long blocks

6.2.2.10 DATA_WIDTH

```
DATA_WIDTH positive:= 8    [Constant]
```

6.2.2.11 f_finish

```
f_finish std_logic:=' 0 '    [Signal]
```

flag which states if all data was sent to MAIN module

6.2.2.12 f_full

```
f_full std_logic:=' 0 '    [Signal]
```

flag which states if file_buffer is full or finished

6.2.2.13 file_buffer

```
file_buffer std_logic_vector( 511 downto 0 )    [Signal]
```

buffer for storing chunks of input file

6.2.2.14 FILENAME

```
FILENAME string:=" example.bin "    [Constant]
```

name of the read file

6.2.2.15 Reset

Reset **std_logic:=' 0 '** [Signal]

6.2.2.16 Rx

Rx **std_logic:=' 1 '** [Signal]

6.2.2.17 state

state **STATE_TYPE :=:=IDLE** [Shared Variable]

determinates the state of testing

6.2.2.18 STATE_TYPE

STATE_TYPE **(COMMAND,TRANSMIT,IDLE,STOP)** [Type]

states of sending data to tested component

6.2.2.19 state_uart

state_uart **UART_STATE_TYPE :=IDLE** [Signal]

determinates the state of the uart process

6.2.2.20 Tx

Tx **std_logic:=' 1 '** [Signal]

6.2.2.21 uart_counter

uart_counter **naturalrange 0 toDATA_WIDTH + 3 := 0** [Signal]

counts transmitted bits

6.2.2.22 uart_data

`uart_data std_logic_vector( 0 toDATA_WIDTH - 1 ) [Signal]`

stores data for transmitting, 'to' is used for reverse trick on output assignment

6.2.2.23 uart_enable

`uart_enable std_logic:=' 0 ' [Signal]`

locks pushing data to uart

6.2.2.24 UART_STATE_TYPE

`UART_STATE_TYPE (IDLE,START,DATA,STOP) [Type]`

uart transmitter for testing

6.2.2.25 uut

`uut MAIN [Instantiation]`

main clock signal

entity of tested component

The documentation for this class was generated from the following file:

- test_main.vhd

6.3 behavior Architecture Reference

Processes

- Clk_process
  Clock process definitions.
- stim_proc
  Stimulus process.

Constants

- Clk_period **time:= 83 ns**

  Clock period definitions.

Signals

- Clk **std_logic:=' 0 '**

  Inputs sha_tx.
- Reset **std_logic:=' 0 '**
- Hash_ready **std_logic:=' 0 '**
- Hash_input **hash_array**
- Tx **std_logic**

  Outputs sha_tx.

Instantiations

- uut **SHA_TX**

  declaration sha_tx

6.3.1 Member Function Documentation

6.3.1.1 Clk_process()

```
Clk_process
```

Clock process definitions.

6.3.1.2 stim_proc()

```
 stim_proc ( )   [Process]
```

Stimulus process.

6.3.2 Member Data Documentation

6.3.2.1 Clk

`Clk std_logic:=' 0 '   [Signal]`

Inputs sha_tx.

6.3.2.2 Clk_period

`Clk_period time:= 83 ns   [Constant]`

Clock period definitions.

6.3.2.3 Hash_input

`Hash_input hash_array    [Signal]`

6.3.2.4 Hash_ready

`Hash_ready std_logic:=' 0 '   [Signal]`

6.3.2.5 Reset

`Reset std_logic:=' 0 '   [Signal]`

6.3.2.6 Tx

`Tx std_logic   [Signal]`

Outputs sha_tx.

6.3.2.7 uut

`uut SHA_TX   [Instantiation]`

declaration sha_tx

The documentation for this class was generated from the following file:

- test_sha_tx.vhd

6.4 behavior Architecture Reference

Processes

- Clk_process

    Clock process definitions.
- stim_proc(**Clk** , **TX_Start**)

    Stimulus process.

Constants

- Clk_period **time:= 83 ns**

    Clock period definitions.

Signals

- Clk **std_logic:=' 0 '**

    Inputs uart_tx.
- Reset **std_logic:=' 0 '**
- TX_Data_In **std_logic_vector(7 downto 0):=(others=>' 0 ')**
- TX_Ready **std_logic:=' 0 '**
- TX_Start **std_logic**

    Outputs uart_tx.
- Tx **std_logic**
- able **std_logic:=' 0 '**

Instantiations

- uut **uart_tx**

    declaration uart_tx

6.4.1 Member Function Documentation

6.4.1.1 Clk_process()

```
Clk_process
```

Clock process definitions.

6.4.1.2 stim_proc()

```
stim_proc(
            Clk ,
            TX_Start  ) [Process]
```

Stimulus process.

6.4.2 Member Data Documentation

6.4.2.1 able

```
able std_logic:=' 0 '  [Signal]
```

6.4.2.2 Clk

```
Clk std_logic:=' 0 '  [Signal]
```

Inputs uart_tx.

6.4.2.3 Clk_period

```
Clk_period time:= 83 ns  [Constant]
```

Clock period definitions.

6.4.2.4 Reset

```
Reset std_logic:=' 0 '  [Signal]
```

6.4.2.5 Tx

```
Tx std_logic  [Signal]
```

6.4.2.6 TX_Data_In

```
TX_Data_In std_logic_vector( 7 downto 0 ):=(others=>' 0 ')   [Signal]
```

6.4.2.7 TX_Ready

```
TX_Ready std_logic:=' 0 '   [Signal]
```

6.4.2.8 TX_Start

```
TX_Start std_logic   [Signal]
```

Outputs uart_tx.

6.4.2.9 uut

```
uut uart_tx   [Instantiation]
```

declaration uart_tx

The documentation for this class was generated from the following file:

- test_uart_tx.vhd

6.5 Behavioral Architecture Reference

Processes

- PROCESS_1(**Clk**)

  pointer for the buffer, informs how many bits are free in the buffer

Types

- STATE_TYPE **(NORMAL,FINISH,STOP)**

  types of states for process of PADDING_MESSAGE

Signals

- state **STATE_TYPE :=NORMAL**

  determinates the state of the process
- word_buffer **DWORD :=(others=>' 0 ')**

  buffer used for forming output word
- word_counter **naturalrange 0 to 15 := 0**

  for counting the number of all formed words
- bit_counter **unsigned(63 downto 0):=(others=>' 0 ')**

  for counting the length of the message
- f_ready **std_logic:=' 0 '**

  flag for blocking conditions

6.5.1 Member Function Documentation

6.5.1.1 PROCESS_1()

```
PROCESS_1 (
              Clk    )  [Process]
```

pointer for the buffer, informs how many bits are free in the buffer

6.5.2 Member Data Documentation

6.5.2.1 bit_counter

```
bit_counter unsigned( 63 downto 0 ):=(others=>' 0 ')   [Signal]
```

for counting the length of the message

6.5.2.2 f_ready

```
f_ready std_logic:=' 0 '   [Signal]
```

flag for blocking conditions

6.5.2.3 state

```
state STATE_TYPE :=NORMAL   [Signal]
```

determinates the state of the process

6.5.2.4 STATE_TYPE

```
STATE_TYPE (NORMAL,FINISH,STOP)   [Type]
```

types of states for process of PADDING_MESSAGE

6.5.2.5 word_buffer

```
word_buffer DWORD :=(others=>' 0 ')   [Signal]
```

buffer used for forming output word

6.5.2.6 word_counter

```
word_counter naturalrange 0 to 15 := 0    [Signal]
```

for counting the number of all formed words

The documentation for this class was generated from the following file:

- sha_padding_message.vhd

6.6 Behavioral Architecture Reference

Processes

- PROCESS_0(**Clk** , **Reset**)
  varaible for temporarly storing constant

Types

- STATE_TYPE **(COMPUTE,ADD,INITIALIZE)**
  types of states for process of COMPUTE_HASH

Signals

- state **STATE_TYPE** **:=COMPUTE**

  determinates the state of the process
- working_vars **hash_array** **:=constant_initials**

  working variables used in computation
- hash **hash_array** **:=constant_initials**

  a working variable for computing hash

Aliases

- a **DWORD** **isworking_vars (0)**
- b **DWORD** **isworking_vars (1)**
- c **DWORD** **isworking_vars (2)**
- d **DWORD** **isworking_vars (3)**
- e **DWORD** **isworking_vars (4)**
- f **DWORD** **isworking_vars (5)**
- g **DWORD** **isworking_vars (6)**
- h **DWORD** **isworking_vars (7)**

6.6.1 Member Function Documentation

6.6.1.1 PROCESS_0()

```
PROCESS_0 (
                Clk    ,
                Reset    )   [Process]
```

varaible for temporarly storing constant

6.6.2 Member Data Documentation

6.6.2.1 a

```
a DWORD isworking_vars ( 0 )   [Alias]
```

6.6.2.2 b

```
b DWORD isworking_vars ( 1 )   [Alias]
```

6.6.2.3 c

```
c DWORD isworking_vars ( 2 )   [Alias]
```

6.6.2.4 d

```
d DWORD isworking_vars ( 3 )   [Alias]
```

6.6.2.5 e

```
e DWORD isworking_vars ( 4 )   [Alias]
```

6.6.2.6 f

```
f DWORD isworking_vars ( 5 )   [Alias]
```

6.6.2.7 g

```
g DWORD isworking_vars ( 6 )   [Alias]
```

6.6.2.8 h

```
h DWORD isworking_vars ( 7 )   [Alias]
```

6.6.2.9 hash

```
hash hash_array :=constant_initials    [Signal]
```

a working variable for computing hash

6.6.2.10 state

`state STATE_TYPE :=COMPUTE [Signal]`

determinates the state of the process

6.6.2.11 STATE_TYPE

`STATE_TYPE (COMPUTE,ADD,INITIALIZE) [Type]`

types of states for process of COMPUTE_HASH

6.6.2.12 working_vars

`working_vars hash_array :=constant_initials [Signal]`

working variables used in computation

The documentation for this class was generated from the following file:

- sha_compute_hash.vhd

6.7 Behavioral Architecture Reference

Processes

- OUTPUT(**Clk**)

  handling COMPUTE_HASH requests

Signals

- schedule **message_schedule**

  message schedule with 64 32-bit wordsjak
- ack **integer:= 0**

  acknowledgement which returns number of prepared words in schedule

Instantiations

- word_t **WORD_T**

  module for preparing 64-long word schedule

6.7.1 Member Function Documentation

6.7.1.1 OUTPUT()

```
OUTPUT (
              Clk )
```

handling COMPUTE_HASH requests

6.7.2 Member Data Documentation

6.7.2.1 ack

`ack` **`integer:=`** `0` `[Signal]`

acknowledgement which returns number of prepared words in schedule

6.7.2.2 schedule

`schedule` **`message_schedule`** `[Signal]`

message schedule with 64 32-bit wordsjak

6.7.2.3 word_t

`word_t` **`WORD_T`** `[Instantiation]`

module for preparing 64-long word schedule

The documentation for this class was generated from the following file:

- sha_prepare_schedule.vhd

6.8 Behavioral Architecture Reference

Functions

- **integer** ascii_to_integer**(input: in std_logic_vector)**
  funcition which changes ascii digits to integers

Processes

- RX_COMMANDER(**Clk** , **Reset**)

    waiting for commands

Types

- STATE_TYPE **(WAITING,RECEIVING)**

    types of states for RX_COMMANDER

Signals

- state **STATE_TYPE :=WAITING**

    determinates the state of RX_COMMANDER
- f_ready **std_logic:=' 0 '**

    flag for blocking condition

Shared Variables

- counter **naturalrange 0 to 512**

    stores the length of incoming data, downcounted
- data_buffer **std_logic_vector(63 downto 0):=:=(others=>' 0 ')**

    buffer for storing last 8 bytes in WAITING state

6.8.1 Member Function Documentation

6.8.1.1 ascii_to_integer()

```
integer ascii_to_integer(
                inputin std_logic_vector )  [Function]
```

funcition which changes ascii digits to integers

6.8.1.2 RX_COMMANDER()

```
 RX_COMMANDER(
                Clk    ,
                Reset    )  [Process]
```

waiting for commands

6.8.2 Member Data Documentation

6.8.2.1 counter

`counter naturalrange 0 to 512 [Shared Variable]`

stores the length of incoming data, downcounted

6.8.2.2 data_buffer

`data_buffer std_logic_vector( 63 downto 0 ):=:=(others=>' 0 ') [Shared Variable]`

buffer for storing last 8 bytes in WAITING state

6.8.2.3 f_ready

`f_ready std_logic:=' 0 ' [Signal]`

flag for blocking condition

6.8.2.4 state

`state STATE_TYPE :=WAITING [Signal]`

determinates the state of RX_COMMANDER

6.8.2.5 STATE_TYPE

`STATE_TYPE (WAITING,RECEIVING) [Type]`

types of states for RX_COMMANDER

The documentation for this class was generated from the following file:

- uart_commander.vhd

6.9 Behavioral Architecture Reference

Signals

- Clk **std_logic**

  signals for external ports
- Reset **std_logic**
- Rx **std_logic**
- Tx **std_logic**
- Reset_All **std_logic**

  buffer for reset
- RX_Data **std_logic_vector(DATA_WIDTH - 1 downto 0)**

  signals between UART_COMMANDER
- RX_Ready **std_logic**
- Hash **hash_array**

  signals between COMPUTE_HASH and SHA_TX
- Hash_ready **std_logic**
- UC_Output_data **std_logic_vector(DATA_WIDTH - 1 downto 0)**

  signals between UART_COMMANDER and PADDING_MESSAGE
- UC_Output_length **positiverange 1 toDATA_WIDTH**
- UC_Output_ready **std_logic**
- UC_Output_finish **std_logic**
- S1_Word_output **DWORD**

  signals between PADDING_MESSAGE and PREPARE_SCHEDULE
- S1_Word_id **naturalrange 0 to 15**
- S1_Word_ready **std_logic**
- S2_Word_output **DWORD**

  signals between PREPARE_SCHEDULE and COMPUTE_HASH
- S2_Word_out_nr **naturalrange 0 to 64**
- S3_Word_req_id **naturalrange 0 to 63**

Instantiations

- u1 **UART_RX**
- u2 **SHA_TX**
- uc **UART_COMMANDER**
- s1 **PADDING_MESSAGE**
- s2 **PREPARE_SCHEDULE**
- s3 **COMPUTE_HASH**

6.9.1 Member Data Documentation

6.9.1.1 Clk

```
Clk std_logic   [Signal]
```

signals for external ports

6.9.1.2 Hash

`Hash hash_array [Signal]`

signals between COMPUTE_HASH and SHA_TX

6.9.1.3 Hash_ready

`Hash_ready std_logic [Signal]`

6.9.1.4 Reset

`Reset std_logic [Signal]`

6.9.1.5 Reset_All

`Reset_All std_logic [Signal]`

buffer for reset

6.9.1.6 Rx

`Rx std_logic [Signal]`

6.9.1.7 RX_Data

`RX_Data std_logic_vector(DATA_WIDTH - 1 downto 0 ) [Signal]`

signals between UART_COMMANDER

6.9.1.8 RX_Ready

`RX_Ready std_logic [Signal]`

6.9.1.9 s1

`s1 PADDING_MESSAGE [Instantiation]`

6.9.1.10 S1_Word_id

`S1_Word_id naturalrange 0 to 15 [Signal]`

6.9.1.11 S1_Word_output

`S1_Word_output DWORD [Signal]`

signals between PADDING_MESSAGE and PREPARE_SCHEDULE

6.9.1.12 S1_Word_ready

`S1_Word_ready std_logic [Signal]`

6.9.1.13 s2

`s2 PREPARE_SCHEDULE [Instantiation]`

6.9.1.14 S2_Word_out_nr

`S2_Word_out_nr naturalrange 0 to 64 [Signal]`

6.9.1.15 S2_Word_output

`S2_Word_output DWORD [Signal]`

signals between PREPARE_SCHEDULE and COMPUTE_HASH

6.9.1.16 s3

```
s3 COMPUTE_HASH    [Instantiation]
```

6.9.1.17 S3_Word_req_id

```
S3_Word_req_id naturalrange 0 to 63    [Signal]
```

6.9.1.18 Tx

```
Tx std_logic    [Signal]
```

6.9.1.19 u1

```
u1 UART_RX    [Instantiation]
```

6.9.1.20 u2

```
u2 SHA_TX    [Instantiation]
```

6.9.1.21 uc

```
uc UART_COMMANDER    [Instantiation]
```

6.9.1.22 UC_Output_data

```
UC_Output_data std_logic_vector(DATA_WIDTH - 1 downto 0 )    [Signal]
```

signals between UART_COMMANDER and PADDING_MESSAGE

6.9.1.23 UC_Output_finish

```
UC_Output_finish std_logic   [Signal]
```

6.9.1.24 UC_Output_length

```
UC_Output_length positiverange 1 toDATA_WIDTH    [Signal]
```

6.9.1.25 UC_Output_ready

```
UC_Output_ready std_logic   [Signal]
```

The documentation for this class was generated from the following file:

- main.vhd

6.10 Behavioral Architecture Reference

Processes

- PROCESS_2(**Clk** , **Reset**)

  count to set when next uart packet should start

Signals

- TX_Data_Out **std_logic_vector(7 downto 0)**

  Inputs from uart_tx.
- TX_Ready **std_logic:=' 0 '**
- TX_Start **std_logic**

  Outputs from uart_tx.
- new_d **std_logic:=' 0 '**

Instantiations

- uart_tx **uart_tx**

  declaration uart_tx

6.10.1 Member Function Documentation

6.10.1.1 PROCESS_2()

```
PROCESS_2 (
              Clk,
              Reset )
```

count to set when next uart packet should start

6.10.2 Member Data Documentation

6.10.2.1 new_d

```
new_d std_logic:=' 0 '   [Signal]
```

6.10.2.2 TX_Data_Out

```
TX_Data_Out std_logic_vector( 7 downto 0 )   [Signal]
```

Inputs from uart_tx.

6.10.2.3 TX_Ready

```
TX_Ready std_logic:=' 0 '   [Signal]
```

6.10.2.4 TX_Start

```
TX_Start std_logic   [Signal]
```

Outputs from uart_tx.

6.10.2.5 uart_tx

```
uart_tx uart_tx   [Instantiation]
```

declaration uart_tx

The documentation for this class was generated from the following file:

- sha_tx.vhd

6.11 Behavioral Architecture Reference

Processes

- RX_PROCESS(**Clk** , **Reset**)

  waiting for data frame
- TO_OUTPUT(**Clk**)

  receiving data frame

Constants

- MAX_FREQ_COUNT **positive:=CLK_FREQUENCY /BAUD**

  length of one bit in clock cycles

Signals

- freq_count **naturalrange 0 toMAX_FREQ_COUNT - 1**

  used for counting clock cycles
- count **naturalrange 0 toDATA_WIDTH + 2**

  counting received bits
- last_Rx **std_logic**

  temporarily keeps last state of Rx input
- receiving **std_logic:=' 0 '**

  determinates if process is in receiving state
- data_buf **std_logic_vector(0 toDATA_WIDTH + 2)**

  buffer for incoming uart frame, 'to' is used for reverse trick on output assignment
- data_ready **std_logic:=' 0 '**

  determinates if data is ready to send to the output

6.11.1 Member Function Documentation

6.11.1.1 RX_PROCESS()

```
RX_PROCESS (
              Clk    ,
              Reset    )   [Process]
```

waiting for data frame

6.11.1.2 TO_OUTPUT()

```
TO_OUTPUT (
              Clk    )   [Process]
```

receiving data frame

6.11.2 Member Data Documentation

6.11.2.1 count

```
count naturalrange 0 toDATA_WIDTH + 2    [Signal]
```

counting received bits

6.11.2.2 data_buf

```
data_buf std_logic_vector( 0 toDATA_WIDTH + 2 )   [Signal]
```

buffer for incoming uart frame, 'to' is used for reverse trick on output assignment

6.11.2.3 data_ready

```
data_ready std_logic:=' 0 '   [Signal]
```

determinates if data is ready to send to the output

6.11.2.4 freq_count

```
freq_count naturalrange 0 toMAX_FREQ_COUNT - 1    [Signal]
```

used for counting clock cycles

6.11.2.5 last_Rx

```
last_Rx std_logic   [Signal]
```

temporarily keeps last state of Rx input

6.11.2.6 MAX_FREQ_COUNT

`MAX_FREQ_COUNT positive:=CLK_FREQUENCY /BAUD [Constant]`

length of one bit in clock cycles

6.11.2.7 receiving

`receiving std_logic:=' 0 ' [Signal]`

determinates if process is in receiving state

The documentation for this class was generated from the following file:

- uart_rx.vhd

6.12 Behavioral Architecture Reference

Processes

- TX_PROCESS(**Clk** , **Reset**)

Constants

- MAX_FREQ_COUNT **positive:=CLK_FREQUENCY /BAUD**

  length of one bit in clock cycles

Signals

- freq_count **naturalrange 0 toMAX_FREQ_COUNT - 1**

  used for counting clock cycles
- count **naturalrange 0 to 11 := 11**

  counting received bits

6.12.1 Member Function Documentation

6.12.1.1 TX_PROCESS()

```
TX_PROCESS(
             Clk    ,
             Reset    )  [Process]
```

6.12.2 Member Data Documentation

6.12.2.1 count

```
count naturalrange 0 to 11 := 11    [Signal]
```

counting received bits

6.12.2.2 freq_count

```
freq_count naturalrange 0 toMAX_FREQ_COUNT - 1    [Signal]
```

used for counting clock cycles

6.12.2.3 MAX_FREQ_COUNT

```
MAX_FREQ_COUNT positive:=CLK_FREQUENCY /BAUD    [Constant]
```

length of one bit in clock cycles

The documentation for this class was generated from the following file:

- uart_tx.vhd

6.13 Behavioral Architecture Reference

Processes

- PROCESS_3(**Clk** , **Reset**)

  variable used as buffer to change contents of schedule in loop

6.13.1 Member Function Documentation

6.13.1.1 PROCESS_3()

```
PROCESS_3 (
                Clk   ,
                Reset   )   [Process]
```

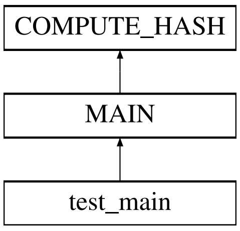
variable used as buffer to change contents of schedule in loop

The documentation for this class was generated from the following file:

- sha_word_t.vhd

6.14 COMPUTE_HASH Entity Reference

Inheritance diagram for COMPUTE_HASH:



Entities

- Behavioral architecture

Libraries

- IEEE

Use Clauses

- STD_LOGIC_1164
- NUMERIC_STD
- sha_function
- constants

Ports

- Clk **in std_logic**
- Word_input **in DWORD**
- Word_in_nr **in naturalrange 0 to 64**

  number of output word (id + 1, 0 means output is not ready)
- Word_req_id **out naturalrange 0 to 63**

  id of requested word
- Output_finish **in std_logic**

  states if all data was already transmitted
- Hash_output **out hash_array**
- Hash_ready **out std_logic:=' 0 '**

  states is hash on output is ready to be read
- Reset **in std_logic**

6.14.1 Member Data Documentation

6.14.1.1 Clk

```
Clk in std_logic   [Port]
```

6.14.1.2 constants

```
constants  [use clause]
```

6.14.1.3 Hash_output

```
Hash_output out hash_array    [Port]
```

output interface calculated hash on output

6.14.1.4 Hash_ready

```
Hash_ready out std_logic:=' 0 '   [Port]
```

states is hash on output is ready to be read

6.14.1.5 IEEE

```
IEEE  [Library]
```

6.14.1.6 NUMERIC_STD

```
NUMERIC_STD  [use clause]
```

6.14.1.7 Output_finish

```
Output_finish in std_logic   [Port]
```

states if all data was already transmitted

6.14.1.8 Reset

```
Reset in std_logic   [Port]
```

6.14.1.9 sha_function

```
sha_function  [use clause]
```

6.14.1.10 STD_LOGIC_1164

```
STD_LOGIC_1164  [use clause]
```

6.14.1.11 Word_in_nr

```
Word_in_nr in naturalrange 0 to 64    [Port]
```

number of output word (id + 1, 0 means output is not ready)

6.14.1.12 Word_input

```
Word_input in DWORD    [Port]
```

input interface 32-bit inpuy word

6.14.1.13 Word_req_id

```
Word_req_id out naturalrange 0 to 63    [Port]
```

id of requested word

The documentation for this class was generated from the following file:

- sha_compute_hash.vhd

6.15 constants Package Reference

Packet with definitions of new types of values and constant variables.

Libraries

- IEEE

  Use standart library.

Use Clauses

- STD_LOGIC_1164

  use logic elements
- NUMERIC_STD

  use numeric elements

Constants

- constant_values **constant_values_sha256** **:=(x" 428a2f98 ",x" 71374491 ",x" b5c0fbcf ",x" e9b5dba5 ",x" 3956c25b ",x" 59f111f1 ",x" 923f82a4 ",x" ab1c5ed5 ",x" d807aa98 ",x" 12835b01 ",x" 243185be ",x" 550c7dc3 ",x" 72be5d74 ",x" 80deb1fe ",x" 9bdc06a7 ",x" c19bf174 ",x" e49b69c1 ",x" efbe4786 ",x" 0fc19dc6 ",x" 240ca1cc ",x" 2de92c6f ",x" 4a7484aa ",x" 5cb0a9dc ",x" 76f988da ",x" 983e5152 ",x" a831c66d ",x" b00327c8 ",x" bf597fc7 ",x" c6e00bf3 ",x" d5a79147 ",x" 06ca6351 ",x" 14292967 ",x" 27b70a85 ",x" 2e1b2138 ",x" 4d2c6dfc ",x" 53380d13 ",x" 650a7354 ",x" 766a0abb ",x" 81c2c92e ",x" 92722c85 ",x" a2bfe8a1 ",x" a81a664b ",x" c24b8b70 ",x" c76c51a3 ",x" d192e819 ",x" d6990624 ",x" f40e3585 ",x" 106aa070 ",x" 19a4c116 ",x" 1e376c08 ",x" 2748774c ",x" 34b0bcb5 ",x" 391c0cb3 ",x" 4ed8aa4a ",x" 5b9cca4f ",x" 682e6ff3 ",x" 748f82ee ",x" 78a5636f ",x" 84c87814 ",x" 8cc70208 ",x" 90befffa ",x" a4506ceb ",x" bef9a3f7 ",x" c67178f2 ")**

  Array of constant values used in sha algoritm.
- constant_initials **hash_array** **:=(x" 6a09e667 ",x" bb67ae85 ",x" 3c6ef372 ",x" a54ff53a ",x" 510e527f ",x" 9b05688c ",x" 1f83d9ab ",x" 5be0cd19 ")**

  Initial hash values.

Types

- message_block **(0 to 15)DWORD**

  Type definition for storing message block's array of 32-bit words.
- message_schedule **(0 to 63)DWORD**

  Type definition for storing message schedule's array of 32-bit words.
- constant_values_sha256 **(0 to 63)DWORD**

  Type definition for storing the constants array.
- hash_array **(0 to 7)DWORD**

  Type definition for constants initial hash values.

Subtypes

- DWORD **std_logic_vector(31 downto 0)**

  Type definition of 32-bit word.

6.15.1 Detailed Description

Packet with definitions of new types of values and constant variables.

6.15.2 Member Data Documentation

6.15.2.1 constant_initials

```
constant_initials hash_array :=(x" 6a09e667 ",x" bb67ae85 ",x" 3c6ef372 ",x" a54ff53a ",x"
510e527f ",x" 9b05688c ",x" 1f83d9ab ",x" 5be0cd19 ")   [Constant]
```

Initial hash values.

6.15.2.2 constant_values

```
constant_values constant_values_sha256 :=(x" 428a2f98 ",x" 71374491 ",x" b5c0fbcf ",x" e9b5dba5
",x" 3956c25b ",x" 59f111f1 ",x" 923f82a4 ",x" ab1c5ed5 ",x" d807aa98 ",x" 12835b01 ",x" 243185be
",x" 550c7dc3 ",x" 72be5d74 ",x" 80deb1fe ",x" 9bdc06a7 ",x" c19bf174 ",x" e49b69c1 ",x" efbe4786
",x" 0fc19dc6 ",x" 240ca1cc ",x" 2de92c6f ",x" 4a7484aa ",x" 5cb0a9dc ",x" 76f988da ",x" 983e5152
",x" a831c66d ",x" b00327c8 ",x" bf597fc7 ",x" c6e00bf3 ",x" d5a79147 ",x" 06ca6351 ",x" 14292967
",x" 27b70a85 ",x" 2e1b2138 ",x" 4d2c6dfc ",x" 53380d13 ",x" 650a7354 ",x" 766a0abb ",x" 81c2c92e
",x" 92722c85 ",x" a2bfe8a1 ",x" a81a664b ",x" c24b8b70 ",x" c76c51a3 ",x" d192e819 ",x" d6990624
",x" f40e3585 ",x" 106aa070 ",x" 19a4c116 ",x" 1e376c08 ",x" 2748774c ",x" 34b0bcb5 ",x" 391c0cb3
",x" 4ed8aa4a ",x" 5b9cca4f ",x" 682e6ff3 ",x" 748f82ee ",x" 78a5636f ",x" 84c87814 ",x" 8cc70208
",x" 90befffa ",x" a4506ceb ",x" bef9a3f7 ",x" c67178f2 ")   [Constant]
```

Array of constant values used in sha algoritm.

6.15.2.3 constant_values_sha256

```
constant_values_sha256 ( 0 to 63 )DWORD   [Type]
```

Type definition for storing the constants array.

6.15.2.4 DWORD

```
DWORD std_logic_vector( 31 downto 0 )   [Subtype]
```

Type definition of 32-bit word.

6.15.2.5 hash_array

```
hash_array ( 0 to 7 )DWORD    [Type]
```

Type definition for constants initial hash values.

6.15.2.6 IEEE

```
IEEE  [Library]
```

Use standart library.

6.15.2.7 message_block

```
message_block ( 0 to 15 )DWORD    [Type]
```

Type definition for storing message block's array of 32-bit words.

6.15.2.8 message_schedule

```
message_schedule ( 0 to 63 )DWORD    [Type]
```

Type definition for storing message schedule's array of 32-bit words.

6.15.2.9 NUMERIC_STD

```
NUMERIC_STD  [use clause]
```

use numeric elements

6.15.2.10 STD_LOGIC_1164

```
STD_LOGIC_1164  [use clause]
```

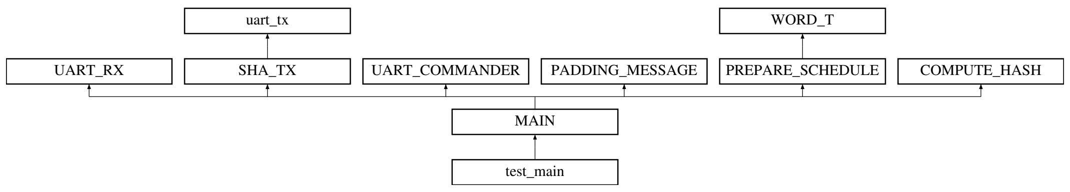
use logic elements

The documentation for this class was generated from the following file:

- constants.vhd

6.16 MAIN Entity Reference

Inheritance diagram for MAIN:



Entities

- Behavioral architecture

Libraries

- IEEE

  use standard library

Use Clauses

- STD_LOGIC_1164

  use logic elements
- NUMERIC_STD

  use numeric elements
- sha_function

  use work packages
- constants

Generics

- CLK_FREQUENCY **positive:= 12000000**

  clock frequency
- DATA_WIDTH **positive:= 8**

  UART message length.
- BAUD **positive:= 19200**

  UART baud rate.

Ports

- Clk_input **in std_logic_vector(0 downto 0)**

  clock external input
- Reset_input **in std_logic_vector(0 downto 0)**

  reset external input
- Rx_input **in std_logic_vector(0 downto 0)**

  Rx external input.
- Tx_output **out std_logic_vector(0 downto 0)**

  Tx external output.

6.16.1 Member Data Documentation

6.16.1.1 BAUD

`BAUD positive:= 19200 [Generic]`

UART baud rate.

6.16.1.2 CLK_FREQUENCY

`CLK_FREQUENCY positive:= 12000000 [Generic]`

clock frequency

6.16.1.3 Clk_input

`Clk_input in std_logic_vector( 0 downto 0 ) [Port]`

clock external input

6.16.1.4 constants

`constants [use clause]`

6.16.1.5 DATA_WIDTH

`DATA_WIDTH positive:= 8 [Generic]`

UART message length.

6.16.1.6 IEEE

`IEEE [Library]`

use standard library

6.16.1.7 NUMERIC_STD

`NUMERIC_STD [use clause]`

use numeric elements

6.16.1.8 Reset_input

`Reset_input in std_logic_vector( 0 downto 0 ) [Port]`

reset external input

6.16.1.9 Rx_input

`Rx_input in std_logic_vector( 0 downto 0 ) [Port]`

Rx external input.

6.16.1.10 sha_function

`sha_function [use clause]`

use work packages

6.16.1.11 STD_LOGIC_1164

`STD_LOGIC_1164 [use clause]`

use logic elements

6.16.1.12 Tx_output

`Tx_output out std_logic_vector( 0 downto 0 ) [Port]`

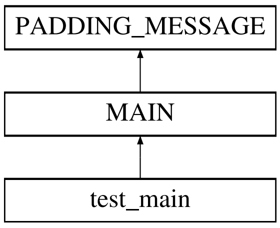
Tx external output.

The documentation for this class was generated from the following file:

- main.vhd

6.17 PADDING_MESSAGE Entity Reference

Inheritance diagram for PADDING_MESSAGE:



Entities

- Behavioral architecture

Libraries

- IEEE

Use Clauses

- STD_LOGIC_1164
- NUMERIC_STD
- sha_function
- constants

Generics

- DATA_WIDTH **positive:= 8**

Ports

- Clk **in std_logic**
- Input_data **in std_logic_vector(DATA_WIDTH - 1 downto 0)**
- Input_length **in positiverange 1 toDATA_WIDTH**

  length of data counted from the highest bit
- Input_ready **in std_logic**

  states is data on output is ready to be read
- Input_finish **in std_logic**

  states if all data was transmited for hash calculation
- Word_output **out DWORD**
- Word_id **out naturalrange 0 to 15**

  id of the word in the message block
- Word_ready **out std_logic**

  states if output word is ready to be read
- Reset **in std_logic**

6.17.1 Member Data Documentation

6.17.1.1 Clk

`Clk in std_logic [Port]`

6.17.1.2 constants

`constants [use clause]`

6.17.1.3 DATA_WIDTH

`DATA_WIDTH positive:= 8 [Generic]`

6.17.1.4 IEEE

`IEEE [Library]`

6.17.1.5 Input_data

`Input_data in std_logic_vector(DATA_WIDTH - 1 downto 0 ) [Port]`

input interface received data for hash calculation

6.17.1.6 Input_finish

`Input_finish in std_logic [Port]`

states if all data was transmited for hash calculation

6.17.1.7 Input_length

```
Input_length in positiverange 1 toDATA_WIDTH    [Port]
```

length of data counted from the highest bit

6.17.1.8 Input_ready

```
Input_ready in std_logic   [Port]
```

states is data on output is ready to be read

6.17.1.9 NUMERIC_STD

```
NUMERIC_STD  [use clause]
```

6.17.1.10 Reset

```
Reset in std_logic   [Port]
```

6.17.1.11 sha_function

```
sha_function  [use clause]
```

6.17.1.12 STD_LOGIC_1164

```
STD_LOGIC_1164  [use clause]
```

6.17.1.13 Word_id

```
Word_id out naturalrange 0 to 15    [Port]
```

id of the word in the message block

6.17.1.14 Word_output

```
Word_output out DWORD    [Port]
```

output interface input data merged into 32-bit word

6.17.1.15 Word_ready

```
Word_ready out std_logic   [Port]
```

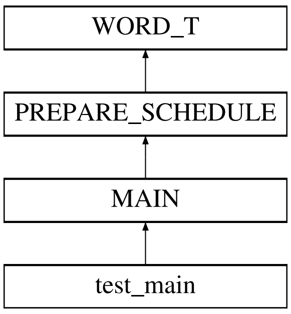
states if output word is ready to be read

The documentation for this class was generated from the following file:

- sha_padding_message.vhd

6.18 PREPARE_SCHEDULE Entity Reference

Inheritance diagram for PREPARE_SCHEDULE:



Entities

- Behavioral architecture

Libraries

- IEEE

Use Clauses

- STD_LOGIC_1164
- NUMERIC_STD
- sha_function
- constants

Ports

- Clk **in std_logic**
- Word_input **in DWORD**
- Word_in_id **in naturalrange 0 to 15**

  id of the word in the message block
- Word_in_ready **in std_logic**

  states if output word is ready to be read
- Word_output **out DWORD**
- Word_out_nr **out naturalrange 0 to 64**

  number of output word (id + 1, 0 means output is not ready)
- Word_req_id **in naturalrange 0 to 63**

  id of requested word
- Reset **in std_logic**

6.18.1 Member Data Documentation

6.18.1.1 Clk

```
Clk in std_logic  [Port]
```

6.18.1.2 constants

```
constants  [use clause]
```

6.18.1.3 IEEE

```
IEEE  [Library]
```

6.18.1.4 NUMERIC_STD

```
NUMERIC_STD  [use clause]
```

6.18.1.5 Reset

```
Reset in std_logic  [Port]
```

6.18.1.6 sha_function

```
sha_function  [use clause]
```

6.18.1.7 STD_LOGIC_1164

```
STD_LOGIC_1164  [use clause]
```

6.18.1.8 Word_in_id

```
Word_in_id in naturalrange 0 to 15    [Port]
```

id of the word in the message block

6.18.1.9 Word_in_ready

```
Word_in_ready in std_logic   [Port]
```

states if output word is ready to be read

6.18.1.10 Word_input

```
Word_input in DWORD    [Port]
```

input interface 32-bit input word

6.18.1.11 Word_out_nr

```
Word_out_nr out naturalrange 0 to 64    [Port]
```

number of output word (id + 1, 0 means output is not ready)

6.18.1.12 Word_output

```
Word_output out DWORD    [Port]
```

output interface 32-bit output word

6.18.1.13 Word_req_id

```
Word_req_id in naturalrange 0 to 63    [Port]
```

id of requested word

The documentation for this class was generated from the following file:

- sha_prepare_schedule.vhd

6.19 sha_function Package Reference

Packet with definition function used in sha algorithm.

Package Body >> sha_function

Functions

- **std_logic_vector CH(**
  x: in std_logic_vector
  y: in std_logic_vector
  z: in std_logic_vector
)

  Function with mathemeatical operations used in sha.
- **std_logic_vector MAJ(**
  x: in std_logic_vector
  y: in std_logic_vector
  z: in std_logic_vector
)

  Function with mathemeatical operations used in sha.
- **std_logic_vector EP0(x: in std_logic_vector)**

  function used because of lack unicode support
- **std_logic_vector EP1(x: in std_logic_vector)**

  function used because of lack unicode support
- **std_logic_vector SIG0(x: in std_logic_vector)**

  function used because of lack unicode support
- **std_logic_vector SIG1(x: in std_logic_vector)**

  function used because of lack unicode support
- **std_logic_vector code_e(**
  h: in std_logic_vector
  e: in std_logic_vector
  f: in std_logic_vector
  g: in std_logic_vector
  d: in std_logic_vector
  M: in std_logic_vector
  K: in std_logic_vector
)

  Function with calculatete temporary values used to code value.

- **std_logic_vector** code_a**(**
  h: in std_logic_vector
  e: in std_logic_vector
  f: in std_logic_vector
  g: in std_logic_vector
  a: in std_logic_vector
  b: in std_logic_vector
  c: in std_logic_vector
  M: in std_logic_vector
  K: in std_logic_vector
)

  Function with calculatete temporary values used to code value.

Procedures

- adding(**signal h: inouthash_array ,signal w_v: inhash_array**)

  add previous and current value

Libraries

- IEEE

  Use standart library.

Use Clauses

- STD_LOGIC_1164

  use logic elements
- numeric_std

  use numeric elements
- constants

  use constants value

6.19.1 Detailed Description

Packet with definition function used in sha algorithm.

6.19.2 Member Function Documentation

6.19.2.1 adding()

```
adding(
            signal   h inouthash_array   ,
            signal   w_v inhash_array   )   [Procedure]
```

add previous and current value

6.19.2.2 CH()

```
std_logic_vector CH(
             xin std_logic_vector ,
             yin std_logic_vector ,
             zin std_logic_vector )  [Function]
```

Function with mathemeatical operations used in sha.

6.19.2.3 code_a()

```
std_logic_vector code_a(
             hin std_logic_vector ,
             ein std_logic_vector ,
             fin std_logic_vector ,
             gin std_logic_vector ,
             ain std_logic_vector ,
             bin std_logic_vector ,
             cin std_logic_vector ,
             Min std_logic_vector ,
             Kin std_logic_vector )  [Function]
```

Function with calculatete temporary values used to code value.

6.19.2.4 code_e()

```
std_logic_vector code_e(
             hin std_logic_vector ,
             ein std_logic_vector ,
             fin std_logic_vector ,
             gin std_logic_vector ,
             din std_logic_vector ,
             Min std_logic_vector ,
             Kin std_logic_vector )  [Function]
```

Function with calculatete temporary values used to code value.

6.19.2.5 EP0()

```
std_logic_vector EP0(
             xin std_logic_vector )  [Function]
```

function used because of lack unicode support

6.19.2.6 EP1()

```
std_logic_vector EP1(
              xin std_logic_vector )  [Function]
```

function used because of lack unicode support

6.19.2.7 MAJ()

```
std_logic_vector MAJ(
              xin std_logic_vector ,
              yin std_logic_vector ,
              zin std_logic_vector )  [Function]
```

Function with mathemeatical operations used in sha.

6.19.2.8 SIG0()

```
std_logic_vector SIG0(
              xin std_logic_vector )  [Function]
```

function used because of lack unicode support

6.19.2.9 SIG1()

```
std_logic_vector SIG1(
              xin std_logic_vector )  [Function]
```

function used because of lack unicode support

6.19.3 Member Data Documentation

6.19.3.1 constants

```
constants  [use clause]
```

use constants value

6.19.3.2 IEEE

`IEEE [Library]`

Use standart library.

6.19.3.3 numeric_std

`numeric_std [use clause]`

use numeric elements

6.19.3.4 STD_LOGIC_1164

`STD_LOGIC_1164 [use clause]`

use logic elements

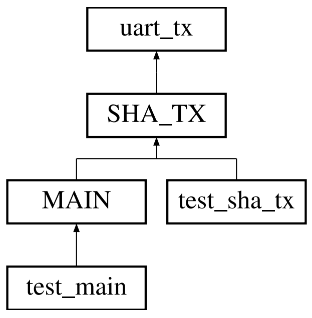
The documentation for this class was generated from the following file:

- sha_function.vhd

6.20 SHA_TX Entity Reference

divides the final sha result by eight bits and sending to uart

Inheritance diagram for SHA_TX:



Entities

- Behavioral architecture

Libraries

- IEEE

  use standard library

Use Clauses

- STD_LOGIC_1164

  use logic elements
- NUMERIC_STD

  use numeric elements
- sha_function
- constants

Generics

- CLK_FREQUENCY **positive:= 12000000**
- DATA_WIDTH **positive:= 8**

  UART message length.
- BAUD **positive:= 19200**

  UART baud rate.

Ports

- Clk **in std_logic**
- Reset **in std_logic**
- Hash_input **in hash_array**

  input SHA-256 hash
- Hash_ready **in std_logic**

  states if incoming data is ready
- Tx **out std_logic**

  Tx pin for transmitting.

6.20.1 Detailed Description

divides the final sha result by eight bits and sending to uart

6.20.2 Member Data Documentation

6.20.2.1 BAUD

```
BAUD  positive:= 19200    [Generic]
```

UART baud rate.

6.20.2.2 Clk

`Clk in std_logic [Port]`

6.20.2.3 CLK_FREQUENCY

`CLK_FREQUENCY positive:= 12000000 [Generic]`

6.20.2.4 constants

`constants [use clause]`

6.20.2.5 DATA_WIDTH

`DATA_WIDTH positive:= 8 [Generic]`

UART message length.

6.20.2.6 Hash_input

`Hash_input in hash_array [Port]`

input SHA-256 hash

6.20.2.7 Hash_ready

`Hash_ready in std_logic [Port]`

states if incoming data is ready

6.20.2.8 IEEE

`IEEE [Library]`

use standard library

6.20.2.9 NUMERIC_STD

`NUMERIC_STD [use clause]`

use numeric elements

6.20.2.10 Reset

`Reset in std_logic [Port]`

6.20.2.11 sha_function

`sha_function [use clause]`

6.20.2.12 STD_LOGIC_1164

`STD_LOGIC_1164 [use clause]`

use logic elements

6.20.2.13 Tx

`Tx out std_logic [Port]`

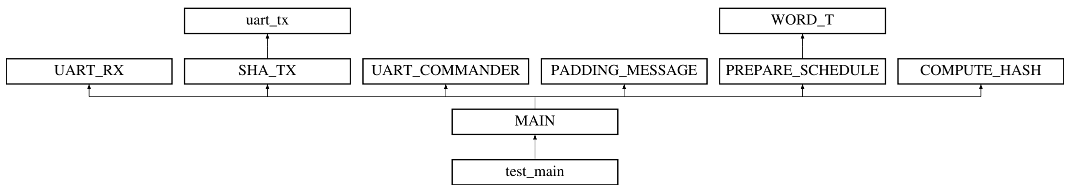
Tx pin for transmitting.

The documentation for this class was generated from the following file:

- sha_tx.vhd

6.21 test_main Entity Reference

Inheritance diagram for test_main:



Entities

- behavior architecture

Libraries

- ieee
- std

Use Clauses

- std_logic_1164
- numeric_std
- std_logic_textio

  for reading/writing files
- textio
- sha_function
- constants

6.21.1 Member Data Documentation

6.21.1.1 constants

```
constants  [use clause]
```

6.21.1.2 ieee

```
ieee  [Library]
```

6.21.1.3 numeric_std

```
numeric_std  [use clause]
```

6.21.1.4 sha_function

```
sha_function  [use clause]
```

6.21.1.5 std

`std [Library]`

6.21.1.6 std_logic_1164

`std_logic_1164 [use clause]`

6.21.1.7 std_logic_textio

`std_logic_textio [use clause]`

for reading/writing files

6.21.1.8 textio

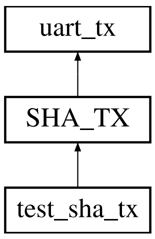
`textio [use clause]`

The documentation for this class was generated from the following file:

- test_main.vhd

6.22 test_sha_tx Entity Reference

Inheritance diagram for test_sha_tx:



Entities

- behavior architecture

Libraries

- ieee

Use Clauses

- std_logic_1164
- sha_function
- constants

6.22.1 Member Data Documentation

6.22.1.1 constants

`constants [use clause]`

6.22.1.2 ieee

`ieee [Library]`

6.22.1.3 sha_function

`sha_function [use clause]`

6.22.1.4 std_logic_1164

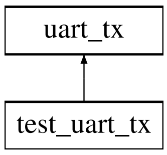
`std_logic_1164 [use clause]`

The documentation for this class was generated from the following file:

- test_sha_tx.vhd

6.23 test_uart_tx Entity Reference

Inheritance diagram for test_uart_tx:



Entities

- behavior architecture

Libraries

- ieee

Use Clauses

- std_logic_1164

6.23.1 Member Data Documentation

6.23.1.1 ieee

```
ieee  [Library]
```

6.23.1.2 std_logic_1164

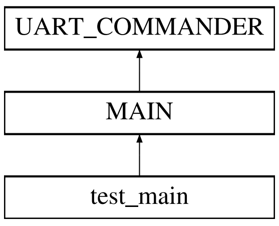
```
std_logic_1164  [use clause]
```

The documentation for this class was generated from the following file:

- test_uart_tx.vhd

6.24 UART_COMMANDER Entity Reference

Inheritance diagram for UART_COMMANDER:



Entities

- Behavioral architecture

Libraries

- IEEE

  use standard library

Use Clauses

- STD_LOGIC_1164

  use logic elements
- NUMERIC_STD

  use numeric elements
- sha_function

  use work packages
- constants

Generics

- DATA_WIDTH **positive:= 8**

Ports

- Clk **in std_logic**
- RX_Data **in std_logic_vector(DATA_WIDTH - 1 downto 0)**
- RX_Ready **in std_logic**

  states if RX_Data is ready to be read
- Output_data **out std_logic_vector(DATA_WIDTH - 1 downto 0)**
- Output_length **out positiverange 1 toDATA_WIDTH**

  length of data counted from the highest bit
- Output_ready **out std_logic:=' 0 '**

  states is data on output is ready to be read
- Output_finish **inout std_logic:=' 0 '**

  states if all data was transmited for hash calculation
- Reset **in std_logic**
- Reset_all **out std_logic:=' 1 '**

  resets blocks after receiving "RESET" command

6.24.1 Member Data Documentation

6.24.1.1 Clk

`Clk in std_logic [Port]`

6.24.1.2 constants

`constants [use clause]`

6.24.1.3 DATA_WIDTH

`DATA_WIDTH positive:= 8 [Generic]`

6.24.1.4 IEEE

`IEEE [Library]`

use standard library

6.24.1.5 NUMERIC_STD

`NUMERIC_STD [use clause]`

use numeric elements

6.24.1.6 Output_data

`Output_data out std_logic_vector(DATA_WIDTH - 1 downto 0 ) [Port]`

Output interface received data for hash calculation

6.24.1.7 Output_finish

```
Output_finish inout std_logic:=' 0 '   [Port]
```

states if all data was transmited for hash calculation

6.24.1.8 Output_length

```
Output_length out positiverange 1 toDATA_WIDTH   [Port]
```

length of data counted from the highest bit

6.24.1.9 Output_ready

```
Output_ready out std_logic:=' 0 '   [Port]
```

states is data on output is ready to be read

6.24.1.10 Reset

```
Reset in std_logic   [Port]
```

6.24.1.11 Reset_all

```
Reset_all out std_logic:=' 1 '   [Port]
```

resets blocks after receiving "RESET" command

6.24.1.12 RX_Data

```
RX_Data in std_logic_vector(DATA_WIDTH - 1 downto 0 )   [Port]
```

Rx interface Raw data from UART_RX

6.24.1.13 RX_Ready

```
RX_Ready in std_logic   [Port]
```

states if RX_Data is ready to be read

6.24.1.14 sha_function

```
sha_function  [use clause]
```

use work packages

6.24.1.15 STD_LOGIC_1164

```
STD_LOGIC_1164  [use clause]
```

use logic elements

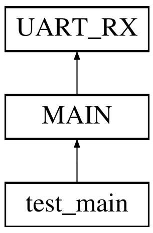
The documentation for this class was generated from the following file:

- uart_commander.vhd

6.25 UART_RX Entity Reference

Definition of UART RX.

Inheritance diagram for UART_RX:



Entities

- Behavioral architecture

Libraries

- IEEE
  use standard library

Use Clauses

- STD_LOGIC_1164

  use logic elements
- NUMERIC_STD

  use numeric elements
- sha_function

  use work packages
- constants

Generics

- CLK_FREQUENCY **positive:= 12000000**
- DATA_WIDTH **positive:= 8**

  UART message length.
- BAUD **positive:= 19200**

  UART baud rate.

Ports

- Clk **in std_logic**
- Reset **in std_logic**
- Rx **in std_logic**

  Rx pin for receiving.
- RX_Data_Out **out std_logic_vector(DATA_WIDTH - 1 downto 0)**

  received data
- RX_Ready **out std_logic:=' 0 '**

  determinates if data on output is ready

6.25.1 Detailed Description

Definition of UART RX.

6.25.2 Member Data Documentation

6.25.2.1 BAUD

```
BAUD  positive:= 19200    [Generic]
```

UART baud rate.

6.25.2.2 Clk

`Clk` **`in std_logic`** `[Port]`

6.25.2.3 CLK_FREQUENCY

`CLK_FREQUENCY` **`positive:= 12000000`** `[Generic]`

6.25.2.4 constants

`constants [use clause]`

6.25.2.5 DATA_WIDTH

`DATA_WIDTH` **`positive:= 8`** `[Generic]`

UART message length.

6.25.2.6 IEEE

`IEEE [Library]`

use standard library

6.25.2.7 NUMERIC_STD

`NUMERIC_STD [use clause]`

use numeric elements

6.25.2.8 Reset

`Reset` **`in std_logic`** `[Port]`

6.25.2.9 Rx

```
Rx in std_logic   [Port]
```

Rx pin for receiving.

6.25.2.10 RX_Data_Out

```
RX_Data_Out out std_logic_vector(DATA_WIDTH - 1 downto 0 )   [Port]
```

received data

6.25.2.11 RX_Ready

```
RX_Ready out std_logic:=' 0 '   [Port]
```

determinates if data on output is ready

6.25.2.12 sha_function

```
sha_function  [use clause]
```

use work packages

6.25.2.13 STD_LOGIC_1164

```
STD_LOGIC_1164  [use clause]
```

use logic elements

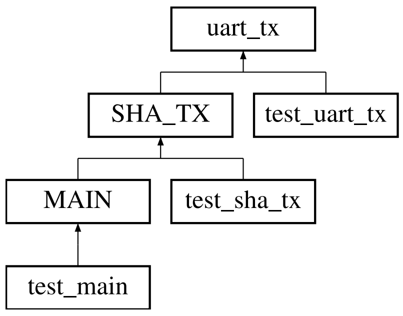
The documentation for this class was generated from the following file:

- uart_rx.vhd

6.26 uart_tx Entity Reference

Definition of UART TX.

Inheritance diagram for uart_tx:



Entities

- Behavioral architecture

Libraries

- IEEE

  use standart library

Use Clauses

- STD_LOGIC_1164

  use logic elements
- NUMERIC_STD

  use numeric elements
- sha_function

  use work packages
- constants

Generics

- CLK_FREQUENCY **positive:= 12000000**
- DATA_WIDTH **positive:= 8**

  UART message length.
- BAUD **positive:= 19200**

  UART baud rate.

Ports

- Clk **in std_logic**
- Reset **in std_logic**
- TX_Data_In **in std_logic_vector(DATA_WIDTH - 1 downto 0)**

  data to transmit
- TX_Ready **in std_logic**

  definition when new data come
- Tx **out std_logic:=' 1 '**

  Tx pin for transmitting.
- TX_Start **out std_logic:=' 1 '**

  definition when new data can come

6.26.1 Detailed Description

Definition of UART TX.

6.26.2 Member Data Documentation

6.26.2.1 BAUD

`BAUD  positive:= 19200   [Generic]`

UART baud rate.

6.26.2.2 Clk

`Clk in std_logic   [Port]`

6.26.2.3 CLK_FREQUENCY

`CLK_FREQUENCY  positive:= 12000000   [Generic]`

6.26.2.4 constants

`constants  [use clause]`

6.26.2.5 DATA_WIDTH

```
DATA_WIDTH positive:= 8  [Generic]
```

UART message length.

6.26.2.6 IEEE

```
IEEE [Library]
```

use standart library

6.26.2.7 NUMERIC_STD

```
NUMERIC_STD [use clause]
```

use numeric elements

6.26.2.8 Reset

```
Reset in std_logic  [Port]
```

6.26.2.9 sha_function

```
sha_function [use clause]
```

use work packages

6.26.2.10 STD_LOGIC_1164

```
STD_LOGIC_1164 [use clause]
```

use logic elements

6.26.2.11 Tx

```
Tx out std_logic:=' 1 '   [Port]
```

Tx pin for transmitting.

6.26.2.12 TX_Data_In

```
TX_Data_In in std_logic_vector(DATA_WIDTH - 1 downto 0 )   [Port]
```

data to transmit

6.26.2.13 TX_Ready

```
TX_Ready in std_logic   [Port]
```

definition when new data come

6.26.2.14 TX_Start

```
TX_Start out std_logic:=' 1 '   [Port]
```

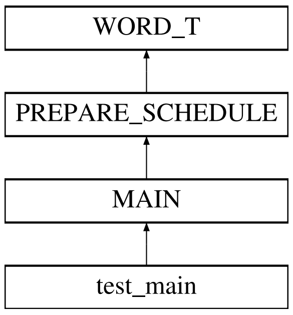
definition when new data can come

The documentation for this class was generated from the following file:

- uart_tx.vhd

6.27 WORD_T Entity Reference

Inheritance diagram for WORD_T:



Entities

- Behavioral architecture

Libraries

- IEEE

Use Clauses

- STD_LOGIC_1164
- NUMERIC_STD
- sha_function
- constants

Ports

- Clk **in std_logic**
- Word_in **in DWORD**
- Word_id **in naturalrange 0 to 15**

  id of the input word
- Ready **in std_logic**

  states if word is ready to be read
- Schedule **inout message_schedule**

  schedule with all words
- Ack **out integerrange 0 to 64 := 0**

  counts Ready words, 0 words Ready in Schedule at the beginning
- Reset **in std_logic**

6.27.1 Member Data Documentation

6.27.1.1 Ack

```
Ack out integerrange 0 to 64 := 0    [Port]
```

counts Ready words, 0 words Ready in Schedule at the beginning

6.27.1.2 Clk

```
Clk in std_logic    [Port]
```

6.27.1.3 constants

`constants [use clause]`

6.27.1.4 IEEE

`IEEE [Library]`

6.27.1.5 NUMERIC_STD

`NUMERIC_STD [use clause]`

6.27.1.6 Ready

`Ready in std_logic [Port]`

states if word is ready to be read

6.27.1.7 Reset

`Reset in std_logic [Port]`

6.27.1.8 Schedule

`Schedule inout message_schedule [Port]`

schedule with all words

6.27.1.9 sha_function

`sha_function [use clause]`

6.27.1.10 STD_LOGIC_1164

```
STD_LOGIC_1164  [use clause]
```

6.27.1.11 Word_id

```
Word_id in naturalrange 0 to 15    [Port]
```

id of the input word

6.27.1.12 Word_in

```
Word_in in DWORD    [Port]
```

input data 32-bit long input word

The documentation for this class was generated from the following file:

- sha_word_t.vhd

Chapter 7

File Documentation

7.1 constants.vhd File Reference

Entities

- constants package

  Packet with definitions of new types of values and constant variables.

7.2 elbertv2_pin.ucf File Reference

Constraints

- VCCAUX **" 3 . 3 "**
- "Clk_input[0]" **LOC=P129|IOSTANDARD=LVCMOS33|PERIOD=12MHz**
- "Rx_input[0]" **LOC=P125|IOSTANDARD=LVCMOS33|SLEW=SLOW|DRIVE= 12**
- "Tx_output[0]" **LOC=P127|IOSTANDARD=LVCMOS33|SLEW=SLOW|DRIVE= 12**

7.2.1 Variable Documentation

7.2.1.1 ""Clk_input[0]""

```
[Constraints]
```

7.2.1.2 ""Rx_input[0]""

```
[Constraints]
```

7.2.1.3 ""Tx_output[0]""

```
[Constraints]
```

7.2.1.4 VCCAUX

```
[Constraints]
```

7.3 main.vhd File Reference

Entities

- MAIN entity
- Behavioral architecture

7.4 sha_compute_hash.vhd File Reference

Entities

- COMPUTE_HASH entity
- Behavioral architecture

7.5 sha_function.vhd File Reference

Entities

- sha_function package

  Packet with definition function used in sha algorithm.
- sha_function package body

  Hash values from the current iteration.

7.6 sha_padding_message.vhd File Reference

Entities

- PADDING_MESSAGE entity
- Behavioral architecture

7.7 sha_prepare_schedule.vhd File Reference

Entities

- PREPARE_SCHEDULE entity
- Behavioral architecture

7.8 sha_tx.vhd File Reference

Entities

- SHA_TX entity

  divides the final sha result by eight bits and sending to uart
- Behavioral architecture

7.9 sha_word_t.vhd File Reference

Entities

- WORD_T entity
- Behavioral architecture

7.10 test_main.vhd File Reference

Entities

- test_main entity
- behavior architecture

7.11 test_sha_tx.vhd File Reference

Entities

- test_sha_tx entity
- behavior architecture

7.12 test_uart_tx.vhd File Reference

Entities

- test_uart_tx entity
- behavior architecture

7.13 uart_commander.vhd File Reference

Entities

- UART_COMMANDER entity
- Behavioral architecture

7.14 uart_rx.vhd File Reference

Entities

- UART_RX entity

  Definition of UART RX.
- Behavioral architecture

7.15 uart_tx.vhd File Reference

Entities

- uart_tx entity

  Definition of UART TX.
- Behavioral architecture

Index

"Clk_input[0]"
    elbertv2_pin.ucf, 85
"Rx_input[0]"
    elbertv2_pin.ucf, 85
"Tx_output[0]"
    elbertv2_pin.ucf, 85

a
    Behavioral, 28
able
    behavior, 24
Ack
    WORD_T, 82
ack
    Behavioral, 31
adding
    sha_function, 12, 60
ascii_to_integer
    Behavioral, 32

b
    Behavioral, 28
BAUD
    behavior, 16
    MAIN, 51
    SHA_TX, 64
    UART_RX, 75
    uart_tx, 79
behavior, 14, 20, 23
    able, 24
    BAUD, 16
    BYTE_FILE_TYPE, 16
    char_buffer, 16
    Clk, 17, 21, 24
    CLK_FREQUENCY, 17
    Clk_period, 22, 24
    Clk_process, 21, 23
    command_buffer, 17
    command_counter, 17
    COMMANDER, 15
    Data_File, 17
    data_length, 17
    DATA_WIDTH, 18
    f_finish, 18
    f_full, 18
    file_buffer, 18
    FILENAME, 18
    Hash_input, 22
    Hash_ready, 22
    Reset, 18, 22, 24
    Rx, 19
    state, 19
    STATE_TYPE, 19
    state_uart, 19
    stim_proc, 21, 23
    to_std_logic_vector, 16
    TRANSMITTER, 16
    Tx, 19, 22, 24
    TX_Data_In, 24
    TX_Ready, 25
    TX_Start, 25
    uart_counter, 19
    uart_data, 19
    uart_enable, 20
    UART_STATE_TYPE, 20
    uut, 20, 22, 25
Behavioral, 25, 27, 30, 31, 34, 38, 40, 42, 43
    a, 28
    ack, 31
    ascii_to_integer, 32
    b, 28
    bit_counter, 26
    c, 28
    Clk, 34
    count, 41, 43
    counter, 33
    d, 29
    data_buf, 41
    data_buffer, 33
    data_ready, 41
    e, 29
    f, 29
    f_ready, 26, 33
    freq_count, 41, 43
    g, 29
    h, 29
    Hash, 34
    hash, 29
    Hash_ready, 35
    last_Rx, 41
    MAX_FREQ_COUNT, 41, 43
    new_d, 39
    OUTPUT, 31
    PROCESS_0, 28
    PROCESS_1, 26
    PROCESS_2, 38
    PROCESS_3, 43
    receiving, 42
    Reset, 35

Reset_All, 35
Rx, 35
RX_COMMANDER, 32
RX_Data, 35
RX_PROCESS, 40
RX_Ready, 35
s1, 35
S1_Word_id, 36
S1_Word_output, 36
S1_Word_ready, 36
s2, 36
S2_Word_out_nr, 36
S2_Word_output, 36
s3, 36
S3_Word_req_id, 37
schedule, 31
state, 26, 29, 33
STATE_TYPE, 27, 30, 33
TO_OUTPUT, 40
Tx, 37
TX_Data_Out, 39
TX_PROCESS, 42
TX_Ready, 39
TX_Start, 39
u1, 37
u2, 37
uart_tx, 39
uc, 37
UC_Output_data, 37
UC_Output_finish, 37
UC_Output_length, 38
UC_Output_ready, 38
word_buffer, 27
word_counter, 27
word_t, 31
working_vars, 30

bit_counter
Behavioral, 26

BYTE_FILE_TYPE
behavior, 16

c
Behavioral, 28

CH
sha_function, 12, 60

char_buffer
behavior, 16

Clk
behavior, 17, 21, 24
Behavioral, 34
COMPUTE_HASH, 45
PADDING_MESSAGE, 54
PREPARE_SCHEDULE, 57
SHA_TX, 64
UART_COMMANDER, 72
UART_RX, 75
uart_tx, 79
WORD_T, 82

CLK_FREQUENCY
behavior, 17
MAIN, 51
SHA_TX, 65
UART_RX, 76
uart_tx, 79

Clk_input
MAIN, 51

Clk_period
behavior, 22, 24

Clk_process
behavior, 21, 23

code_a
sha_function, 12, 61

code_e
sha_function, 13, 61

command_buffer
behavior, 17

command_counter
behavior, 17

COMMANDER
behavior, 15

COMPUTE_HASH, 44
Clk, 45
constants, 45
Hash_output, 45
Hash_ready, 45
IEEE, 45
NUMERIC_STD, 45
Output_finish, 45
Reset, 46
sha_function, 46
STD_LOGIC_1164, 46
Word_in_nr, 46
Word_input, 46
Word_req_id, 46

constant_initials
constants, 48

constant_values
constants, 48

constant_values_sha256
constants, 48

constants, 9, 47
COMPUTE_HASH, 45
constant_initials, 48
constant_values, 48
constant_values_sha256, 48
DWORD, 48
hash_array, 48
IEEE, 49
MAIN, 51
message_block, 49
message_schedule, 49
NUMERIC_STD, 49
PADDING_MESSAGE, 54
PREPARE_SCHEDULE, 57
sha_function, 62
SHA_TX, 65
STD_LOGIC_1164, 49

test_main, 67
test_sha_tx, 69
UART_COMMANDER, 72
UART_RX, 76
uart_tx, 79
WORD_T, 82
constants.vhd, 85
count
Behavioral, 41, 43
counter
Behavioral, 33

d
Behavioral, 29
data_buf
Behavioral, 41
data_buffer
Behavioral, 33
Data_File
behavior, 17
data_length
behavior, 17
data_ready
Behavioral, 41
DATA_WIDTH
behavior, 18
MAIN, 51
PADDING_MESSAGE, 54
SHA_TX, 65
UART_COMMANDER, 72
UART_RX, 76
uart_tx, 79
DWORD
constants, 48

e
Behavioral, 29
elbertv2_pin.ucf, 85
"Clk_input[0]", 85
"Rx_input[0]", 85
"Tx_output[0]", 85
VCCAUX, 86
EP0
sha_function, 13, 61
EP1
sha_function, 13, 61

f
Behavioral, 29
f_finish
behavior, 18
f_full
behavior, 18
f_ready
Behavioral, 26, 33
file_buffer
behavior, 18
FILENAME
behavior, 18
freq_count
Behavioral, 41, 43

g
Behavioral, 29

h
Behavioral, 29
Hash
Behavioral, 34
hash
Behavioral, 29
hash_array
constants, 48
Hash_input
behavior, 22
SHA_TX, 65
Hash_output
COMPUTE_HASH, 45
Hash_ready
behavior, 22
Behavioral, 35
COMPUTE_HASH, 45
SHA_TX, 65

IEEE
COMPUTE_HASH, 45
constants, 49
MAIN, 51
PADDING_MESSAGE, 54
PREPARE_SCHEDULE, 57
sha_function, 62
SHA_TX, 65
UART_COMMANDER, 72
UART_RX, 76
uart_tx, 80
WORD_T, 83
ieee
test_main, 67
test_sha_tx, 69
test_uart_tx, 70
Input_data
PADDING_MESSAGE, 54
Input_finish
PADDING_MESSAGE, 54
Input_length
PADDING_MESSAGE, 54
Input_ready
PADDING_MESSAGE, 55

last_Rx
Behavioral, 41

MAIN, 50
BAUD, 51
CLK_FREQUENCY, 51
Clk_input, 51
constants, 51
DATA_WIDTH, 51

IEEE, 51
NUMERIC_STD, 51
Reset_input, 52
Rx_input, 52
sha_function, 52
STD_LOGIC_1164, 52
Tx_output, 52
main.vhd, 86
MAJ
sha_function, 13, 62
MAX_FREQ_COUNT
Behavioral, 41, 43
message_block
constants, 49
message_schedule
constants, 49

new_d
Behavioral, 39
NUMERIC_STD
COMPUTE_HASH, 45
constants, 49
MAIN, 51
PADDING_MESSAGE, 55
PREPARE_SCHEDULE, 57
SHA_TX, 65
UART_COMMANDER, 72
UART_RX, 76
uart_tx, 80
WORD_T, 83
numeric_std
sha_function, 63
test_main, 67

OUTPUT
Behavioral, 31
Output_data
UART_COMMANDER, 72
Output_finish
COMPUTE_HASH, 45
UART_COMMANDER, 72
Output_length
UART_COMMANDER, 73
Output_ready
UART_COMMANDER, 73

PADDING_MESSAGE, 53
Clk, 54
constants, 54
DATA_WIDTH, 54
IEEE, 54
Input_data, 54
Input_finish, 54
Input_length, 54
Input_ready, 55
NUMERIC_STD, 55
Reset, 55
sha_function, 55
STD_LOGIC_1164, 55
Word_id, 55
Word_output, 55
Word_ready, 56
PREPARE_SCHEDULE, 56
Clk, 57
constants, 57
IEEE, 57
NUMERIC_STD, 57
Reset, 57
sha_function, 57
STD_LOGIC_1164, 58
Word_in_id, 58
Word_in_ready, 58
Word_input, 58
Word_out_nr, 58
Word_output, 58
Word_req_id, 58
PROCESS_0
Behavioral, 28
PROCESS_1
Behavioral, 26
PROCESS_2
Behavioral, 38
PROCESS_3
Behavioral, 43

Ready
WORD_T, 83
receiving
Behavioral, 42
Reset
behavior, 18, 22, 24
Behavioral, 35
COMPUTE_HASH, 46
PADDING_MESSAGE, 55
PREPARE_SCHEDULE, 57
SHA_TX, 66
UART_COMMANDER, 73
UART_RX, 76
uart_tx, 80
WORD_T, 83
Reset_All
Behavioral, 35
Reset_all
UART_COMMANDER, 73
Reset_input
MAIN, 52
Rx
behavior, 19
Behavioral, 35
UART_RX, 76
RX_COMMANDER
Behavioral, 32
RX_Data
Behavioral, 35
UART_COMMANDER, 73
RX_Data_Out
UART_RX, 77
Rx_input

MAIN, 52
RX_PROCESS
Behavioral, 40
RX_Ready
Behavioral, 35
UART_COMMANDER, 73
UART_RX, 77

s1
Behavioral, 35
S1_Word_id
Behavioral, 36
S1_Word_output
Behavioral, 36
S1_Word_ready
Behavioral, 36
s2
Behavioral, 36
S2_Word_out_nr
Behavioral, 36
S2_Word_output
Behavioral, 36
s3
Behavioral, 36
S3_Word_req_id
Behavioral, 37
Schedule
WORD_T, 83
schedule
Behavioral, 31
sha_compute_hash.vhd, 86
sha_function, 9, 11, 59
adding, 12, 60
CH, 12, 60
code_a, 12, 61
code_e, 13, 61
COMPUTE_HASH, 46
constants, 62
EP0, 13, 61
EP1, 13, 61
IEEE, 62
MAIN, 52
MAJ, 13, 62
numeric_std, 63
PADDING_MESSAGE, 55
PREPARE_SCHEDULE, 57
SHA_TX, 66
SIG0, 13, 62
SIG1, 14, 62
STD_LOGIC_1164, 63
test_main, 67
test_sha_tx, 69
UART_COMMANDER, 74
UART_RX, 77
uart_tx, 80
WORD_T, 83
sha_function.vhd, 86
sha_padding_message.vhd, 86
sha_prepare_schedule.vhd, 87
SHA_TX, 63
BAUD, 64
Clk, 64
CLK_FREQUENCY, 65
constants, 65
DATA_WIDTH, 65
Hash_input, 65
Hash_ready, 65
IEEE, 65
NUMERIC_STD, 65
Reset, 66
sha_function, 66
STD_LOGIC_1164, 66
Tx, 66
sha_tx.vhd, 87
sha_word_t.vhd, 87
SIG0
sha_function, 13, 62
SIG1
sha_function, 14, 62
state
behavior, 19
Behavioral, 26, 29, 33
STATE_TYPE
behavior, 19
Behavioral, 27, 30, 33
state_uart
behavior, 19
std
test_main, 67
STD_LOGIC_1164
COMPUTE_HASH, 46
constants, 49
MAIN, 52
PADDING_MESSAGE, 55
PREPARE_SCHEDULE, 58
sha_function, 63
SHA_TX, 66
UART_COMMANDER, 74
UART_RX, 77
uart_tx, 80
WORD_T, 83
std_logic_1164
test_main, 68
test_sha_tx, 69
test_uart_tx, 70
std_logic_textio
test_main, 68
stim_proc
behavior, 21, 23

test_main, 66
constants, 67
ieee, 67
numeric_std, 67
sha_function, 67
std, 67
std_logic_1164, 68
std_logic_textio, 68

textio, 68
test_main.vhd, 87
test_sha_tx, 68
constants, 69
ieee, 69
sha_function, 69
std_logic_1164, 69
test_sha_tx.vhd, 87
test_uart_tx, 70
ieee, 70
std_logic_1164, 70
test_uart_tx.vhd, 87
textio
test_main, 68
TO_OUTPUT
Behavioral, 40
to_std_logic_vector
behavior, 16
TRANSMITTER
behavior, 16
Tx
behavior, 19, 22, 24
Behavioral, 37
SHA_TX, 66
uart_tx, 80
TX_Data_In
behavior, 24
uart_tx, 81
TX_Data_Out
Behavioral, 39
Tx_output
MAIN, 52
TX_PROCESS
Behavioral, 42
TX_Ready
behavior, 25
Behavioral, 39
uart_tx, 81
TX_Start
behavior, 25
Behavioral, 39
uart_tx, 81

u1
Behavioral, 37
u2
Behavioral, 37
UART_COMMANDER, 71
Clk, 72
constants, 72
DATA_WIDTH, 72
IEEE, 72
NUMERIC_STD, 72
Output_data, 72
Output_finish, 72
Output_length, 73
Output_ready, 73
Reset, 73
Reset_all, 73
RX_Data, 73
RX_Ready, 73
sha_function, 74
STD_LOGIC_1164, 74
uart_commander.vhd, 88
uart_counter
behavior, 19
uart_data
behavior, 19
uart_enable
behavior, 20
UART_RX, 74
BAUD, 75
Clk, 75
CLK_FREQUENCY, 76
constants, 76
DATA_WIDTH, 76
IEEE, 76
NUMERIC_STD, 76
Reset, 76
Rx, 76
RX_Data_Out, 77
RX_Ready, 77
sha_function, 77
STD_LOGIC_1164, 77
uart_rx.vhd, 88
UART_STATE_TYPE
behavior, 20
uart_tx, 78
BAUD, 79
Behavioral, 39
Clk, 79
CLK_FREQUENCY, 79
constants, 79
DATA_WIDTH, 79
IEEE, 80
NUMERIC_STD, 80
Reset, 80
sha_function, 80
STD_LOGIC_1164, 80
Tx, 80
TX_Data_In, 81
TX_Ready, 81
TX_Start, 81
uart_tx.vhd, 88
uc
Behavioral, 37
UC_Output_data
Behavioral, 37
UC_Output_finish
Behavioral, 37
UC_Output_length
Behavioral, 38
UC_Output_ready
Behavioral, 38
uut
behavior, 20, 22, 25

VCCAUX

elbertv2_pin.ucf, 86

word_buffer
Behavioral, 27
word_counter
Behavioral, 27
Word_id
PADDING_MESSAGE, 55
WORD_T, 84
Word_in
WORD_T, 84
Word_in_id
PREPARE_SCHEDULE, 58
Word_in_nr
COMPUTE_HASH, 46
Word_in_ready
PREPARE_SCHEDULE, 58
Word_input
COMPUTE_HASH, 46
PREPARE_SCHEDULE, 58
Word_out_nr
PREPARE_SCHEDULE, 58
Word_output
PADDING_MESSAGE, 55
PREPARE_SCHEDULE, 58
Word_ready
PADDING_MESSAGE, 56
Word_req_id
COMPUTE_HASH, 46
PREPARE_SCHEDULE, 58
WORD_T, 81
Ack, 82
Clk, 82
constants, 82
IEEE, 83
NUMERIC_STD, 83
Ready, 83
Reset, 83
Schedule, 83
sha_function, 83
STD_LOGIC_1164, 83
Word_id, 84
Word_in, 84
word_t
Behavioral, 31
working_vars
Behavioral, 30