



## Projeto - Fase 3

# Coprocessador em VHDL

Bruno Susko Marcellini

Eric Rodrigues Pires

Mateus Nakajo de Mendonça

Tiago Koji Castro Shibata

Data: 09/08/2017

## 1. Introdução

O objetivo desta etapa do trabalho é prosseguir o desenvolvimento do coprocessador proposto, mas desta vez realizando a implementação na linguagem VHDL. Também é proposta a realização de testes do algoritmo implementado para a verificação de seu correto funcionamento e por último a sintetização do mesmo na placa Nexys 4 DDR.

Em partes do relatório, apenas trechos do código foram reproduzidos. O código completo pode ser encontrado no seguinte site:

<https://github.com/tiogoshibata/AES-FPGA>

Mais especificamente, códigos em VHDL estão disponíveis no diretório [include/vhdl](#) do repositório; do Microblaze (para acesso ao IP) em [include/microblaze](#).

## 2. Atividades

O trabalho foi dividido em três etapas distintas:

- Execução do algoritmo de referência no Microblaze.

## Projeto Fase 3

- Execução de testbenches em SystemC para verificação do funcionamento.
- Implementação do coprocessador em VHDL.
- Execução de testbenches em VHDL para verificação do funcionamento.
- Sintetizar para Nexys 4 DDR.
- Execução do algoritmo de referência no projeto sintetizado.
- Identificar métricas de desempenho, comparar implementações em hardware.

### 2.1 Execução do algoritmo de referência no Microblaze.

Para a execução do algoritmo de referência no Microblaze foi seguido o tutorial para a montagem básica de uma instância do mesmo no Vivado. Após todos os passos de validação, síntese, implementação e geração do bitstream, foi aberto o SDK do Vivado para a realização dos testes com a placa.

No SDK foram utilizadas as seguintes flags na compilação:

Name	Value	Default
archiver	mb-ar	mb-ar
compiler	mb-gcc	mb-gcc
compiler_flags	-O2 -c	-O2 -c
extra_compiler_flags	-ffunction-sections -fdata-sections -Wall -Wextra	-ffunction-sections -fdata-sections -Wall -Wextra

Rodamos a mesma bateria de testes usada em desktop, que realiza uma cifragem em modo ECB e três testes de cifragem e decifragem de blocos de 128 bits em modo CTR. Não foram realizadas modificações relevantes no código da bateria de testes. Para fins de verificar o correto funcionamento e ainda sem se preocupar com a questão de desempenho, mantivemos as funções de impressão para o terminal.

Após a execução, foi possível observarmos na saída serial do Microblaze:

```
AES-ECB-128 (enc): passed
AES-CTR-128 (dec): passed
AES-CTR-128 (enc): passed
AES-CTR-128 (dec): passed
AES-CTR-128 (enc): passed
AES-CTR-128 (dec): passed
AES-CTR-128 (enc): passed
```

```
~/opt/Xilinx_Vivado
```

Captura de tela com o fim dos testes no softcore

Em seguida, desativamos a serial e os comandos de print para não influenciarem no tempo de execução. Ao executar a análise de performance do Vivado, observamos 27.809 segundos para executar toda a bateria de testes puramente no softcore.

### 2.2 Execução de testbenches em SystemC para verificação do funcionamento.

A implementação em SystemC e testes com testbenches foi mostrada no último relatório. Para completude, repetimos aqui os testes (código fonte disponível no repositório do projeto). Implementamos o testbench do sistema com os mesmos valores de teste do "golden":

**main.cc**

```
include <systemc>
#include <iostream>

#include "aes_ctr.h"

#define tick() do{ \
    std::cout << "Running 1 cycle\n"; \
    sc_core::sc_start(half_clock_time); \
    clock = 1; \
    sc_core::sc_start(half_clock_time); \
    clock = 0; \
}while(0)

static const unsigned char aes_test_ctr_nonce_counter[16] =
    { 0x00, 0x00, 0x00, 0x30, 0x00, 0x00, 0x00, 0x00,
```

## Projeto Fase 3

```
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01 };;

static const unsigned char aes_test_ctr_key[16] =
{ 0xAE, 0x68, 0x52, 0xF8, 0x12, 0x10, 0x67, 0xCC,
  0x4B, 0xF7, 0xA5, 0x76, 0x55, 0x77, 0xF3, 0x9E };;

static const unsigned char aes_test_ctr_ct[16] =
{ 0xE4, 0x09, 0x5D, 0x4F, 0xB7, 0xA7, 0xB3, 0x79,
  0x2D, 0x61, 0x75, 0xA3, 0x26, 0x13, 0x11, 0xB8 };;

static const unsigned char aes_test_ctr_pt[16] =
{ 0x53, 0x69, 0x6E, 0x67, 0x6C, 0x65, 0x20, 0x62,
  0x6C, 0x6F, 0x63, 0x6B, 0x20, 0x6D, 0x73, 0x67 };;

int sc_main(int argc, char* argv[])
{
    std::cout << "Starting simulation: " << sc_core::sc_time_stamp() << "\n";

    sc_core::sc_time half_clock_time(0.5, sc_core::sc_time_unit::SC_MS);
    AES_CTR ctr("AES_CTR");

    sc_core::sc_signal<bool> clock, start, clear;
    sc_core::sc_vector<sc_core::sc_signal<unsigned char>> input("AES_input", 16),
        nonce_counter("AES_nonce_counter", 16), key("AES_key", 16);
    sc_core::sc_vector<sc_core::sc_signal<unsigned char>> output("AES_output", 16);
    sc_core::sc_signal<bool> done;

    for (int i = 0; i < 16; i++) {
        key[i] = aes_test_ctr_key[i];
        nonce_counter[i] = aes_test_ctr_nonce_counter[i];
        input[i] = aes_test_ctr_pt[i];
    }

    ctr.clock(clock);
    ctr.start(start);
    ctr.clear(clear);
    ctr.input(input);
    ctr.nonce_counter(nonce_counter);
    ctr.key(key);
    ctr.output(output);
    ctr.done(done);

    start = 0;
    clear = 1;
    tick();
    clear = 0;
    tick();
    start = 1;
    tick();
    start = 0;
    while (!ctr.done) {
        tick();
    }

    std::cout << "Ended simulation: " << sc_core::sc_time_stamp() << "\n";
    for (int i = 0; i < 16; i++) {
        std::cout << "Position " << i << ": expected " << +aes_test_ctr_ct[i] << ", found " <<
+output[i] << "\n";
        assert(aes_test_ctr_ct[i] == output[i]);
    }

    std::cout << "Congratulations, it worked.\n";
    return 0;
}
```

## 2.3 Implementação do co-processador em VHDL.

A implementação em VHDL foi feita baseada no código em SystemC. As características de timing usadas e os componentes criados foram similares aos usados em SystemC.

aes\_ctr.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity AES_CTR is
    port (
        clock, start, clear: in STD_LOGIC;
        input, nonce_counter, key: in STD_LOGIC_VECTOR(127 downto 0);
        output: out STD_LOGIC_VECTOR(127 downto 0);
        done: out STD_LOGIC
    );
end AES_CTR;

architecture arch of AES_CTR is

    -- State machine
    type estados is (
        aes_ctr_st_wait, aes_ctr_st_rk_start, aes_ctr_st_rk_wait,
        aes_ctr_st_enc_start, aes_ctr_st_enc_wait,
        aes_ctr_st_nc_inc, aes_ctr_st_end_wait, aes_ctr_st_end
    );
    signal sreg, snext: estados;

    -- Internal signals
    signal rk0, rk1, rk2, rk3: UNSIGNED(31 downto 0);
    signal rk_addr: UNSIGNED(5 downto 0);
    signal cipher, curr_nc: STD_LOGIC_VECTOR(127 downto 0);

    -- Modules
    signal start_rk, done_rk: STD_LOGIC;
    component AES_RoundKey
        port (
            clock, start, clear: in STD_LOGIC;
            rk_addr: in UNSIGNED(5 downto 0);
            key: in STD_LOGIC_VECTOR(127 downto 0);
            rk0, rk1, rk2, rk3: out UNSIGNED(31 downto 0);
            done: out STD_LOGIC
        );
```

```

end component;
signal start_enc, done_enc: STD_LOGIC;
component AES_Encrypt
    port (
        clock, start, clear: in STD_LOGIC;
        rk0, rk1, rk2, rk3: in UNSIGNED(31 downto 0);
        input: in STD_LOGIC_VECTOR(127 downto 0);
        output: out STD_LOGIC_VECTOR(127 downto 0);
        rk_addr: out UNSIGNED(5 downto 0);
        done: out STD_LOGIC
    );
end component;
signal inc_nc: STD_LOGIC;
component AES_Nonce_Counter
    port (
        clock, clear, increment: in STD_LOGIC;
        original_nc: in STD_LOGIC_VECTOR(127 downto 0);
        current_nc: out STD_LOGIC_VECTOR(127 downto 0)
    );
end component;

begin

roundkey0: AES_RoundKey port map (
    clock, start_rk, clear,
    rk_addr,
    key,
    rk0, rk1, rk2, rk3,
    done_rk
);
aes_encrypt0: AES_Encrypt port map (
    clock, start_enc, clear,
    rk0, rk1, rk2, rk3,
    curr_nc,
    cipher,
    rk_addr,
    done_enc
);
nonce_ctr: AES_Nonce_Counter port map (
    clock, clear, inc_nc,
    nonce_counter,
    curr_nc
);

process (clock)
begin
    if clock'event and clock = '1' then

```

```

sreg <= snext;

case sreg is

    when aes_ctr_st_wait =>
        done <= '0';
        output <= (others => '0');
        inc_nc <= '0';
        start_rk <= '0';
        start_enc <= '0';
    when aes_ctr_st_rk_start =>
        start_rk <= '1';
    when aes_ctr_st_rk_wait =>
        start_rk <= '0';
    when aes_ctr_st_enc_start =>
        start_enc <= '1';
    when aes_ctr_st_enc_wait =>
        start_enc <= '0';
    when aes_ctr_st_nc_inc =>
        output <= input xor cipher;
        inc_nc <= '1';
    when aes_ctr_st_end_wait =>
        inc_nc <= '0';
        done <= '1';
    when aes_ctr_st_end =>
        done <= '1';

end case;
end if;
end process;

process (sreg, start, clear, done_rk, done_enc)
begin
    if (clear = '1') then
        snext <= aes_ctr_st_wait;
    else case sreg is

        when aes_ctr_st_wait =>
            if (start = '1') then
                snext <= aes_ctr_st_rk_start;
            else
                snext <= aes_ctr_st_wait;
            end if;
        when aes_ctr_st_rk_start =>
            snext <= aes_ctr_st_rk_wait;
    when aes_ctr_st_rk_wait =>
        if (done_rk = '1') then

```

```

        snext <= aes_ctr_st_enc_start;
    else
        snext <= aes_ctr_st_rk_wait;
    end if;
when aes_ctr_st_enc_start =>
    snext <= aes_ctr_st_enc_wait;
when aes_ctr_st_enc_wait =>
    if (done_enc = '1') then
        snext <= aes_ctr_st_nc_inc;
    else
        snext <= aes_ctr_st_enc_wait;
    end if;
    when aes_ctr_st_nc_inc =>
        snext <= aes_ctr_st_end_wait;
    when aes_ctr_st_end_wait =>
        if (start = '1') then
            snext <= aes_ctr_st_end_wait;
        else
            snext <= aes_ctr_st_end;
        end if;
    when aes_ctr_st_end =>
        if (start = '1') then
            snext <= aes_ctr_st_enc_start;
        else
            snext <= aes_ctr_st_end;
        end if;
end case;
end if;
end process;

end arch;

```

aes\_encrypt.vhd

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity AES_Encrypt is
    port (
        clock, start, clear: in STD_LOGIC;
        rk0, rk1, rk2, rk3: in UNSIGNED(31 downto 0);
        input: in STD_LOGIC_VECTOR(127 downto 0);
        output: out STD_LOGIC_VECTOR(127 downto 0);
    );
end entity AES_Encrypt;

```



```

        rk_addr: out UNSIGNED(5 downto 0);
        done: out STD_LOGIC
    );
end AES_Encrypt;

architecture arch of AES_Encrypt is

-- State machine
type estados is (
    aes_enc_st_wait, aes_enc_st_read, aes_enc_st_addroundkey,
    aes_enc_st_round_odd_in, aes_enc_st_round_odd_out,
    aes_enc_st_round_even_in, aes_enc_st_round_even_out,
    aes_enc_st_round_last_in, aes_enc_st_round_last_out,
    aes_enc_st_subbytes_in, aes_enc_st_subbytes_out,
    aes_enc_st_end
);
signal sreg, snext: estados;

-- Internal signals
signal x0, x1, x2, x3, y0, y1, y2, y3: UNSIGNED(31 downto 0);
signal aes_round, Irk_addr: UNSIGNED(5 downto 0);

-- Modules
signal froundin0, froundin1, froundin2, froundin3, froundout0, froundout1,
froundout2, froundout3: UNSIGNED(31 downto 0);
component AES_FROUND
    port (
        y0, y1, y2, y3, rk0, rk1, rk2, rk3: in UNSIGNED(31 downto 0);
        x0, x1, x2, x3: out UNSIGNED(31 downto 0)
    );
end component;
signal fsbin0, fsbin1, fsbin2, fsbin3, fsbout0, fsbout1, fsbout2, fsbout3:
UNSIGNED(31 downto 0);
component AES_FSb
    port (
        y0, y1, y2, y3, rk0, rk1, rk2, rk3: in UNSIGNED(31 downto 0);
        x0, x1, x2, x3: out UNSIGNED(31 downto 0)
    );
end component;

begin

fround0: AES_FROUND port map (
    froundin0, froundin1, froundin2, froundin3, rk0, rk1, rk2, rk3,
    froundout0, froundout1, froundout2, froundout3
);
fsb0: AES_FSb port map (

```

### Projeto Fase 3

```
fsbin0, fsbin1, fsbin2, fsbin3, rk0, rk1, rk2, rk3,
fsbout0, fsbout1, fsbout2, fsbout3
);

process (clock)
begin
    if clock'event and clock = '1' then
        sreg <= snext;

        case sreg is

            when aes_enc_st_wait =>
                done <= '0';
                aes_round <= (others => '0');
                Irk_addr <= (others => '1');
            when aes_enc_st_read =>
                Irk_addr <= (others => '0');
                aes_round <= "000011"; -- (10 >> 1) - 2
                x0 <= unsigned(input(103 downto 96) & input(111 downto
104) & input(119 downto 112) & input(127 downto 120));
                x1 <= unsigned(input( 71 downto 64) & input( 79 downto 72) &
input( 87 downto 80) & input( 95 downto 88));
                x2 <= unsigned(input( 39 downto 32) & input( 47 downto 40) &
input( 55 downto 48) & input( 63 downto 56));
                x3 <= unsigned(input( 7 downto 0) & input( 15 downto 8) &
input( 23 downto 16) & input( 31 downto 24));
            when aes_enc_st_addroundkey =>
                x0 <= x0 xor rk0;
                x1 <= x1 xor rk1;
                x2 <= x2 xor rk2;
                x3 <= x3 xor rk3;

                --AES_FROUND( y0, y1, y2, y3, x0, x1, x2, x3 );
            when aes_enc_st_round_odd_in =>
                Irk_addr <= Irk_addr + 4;
                froundin0 <= x0; froundin1 <= x1; froundin2 <= x2;
froundin3 <= x3;
            when aes_enc_st_round_odd_out =>
                y0 <= froundout0; y1 <= froundout1; y2 <= froundout2; y3
<= froundout3;
            when aes_enc_st_round_last_in =>
                Irk_addr <= Irk_addr + 4;
                froundin0 <= x0; froundin1 <= x1; froundin2 <= x2;
froundin3 <= x3;
            when aes_enc_st_round_last_out =>
                y0 <= froundout0; y1 <= froundout1; y2 <= froundout2; y3
<= froundout3;
```

### Projeto Fase 3

```
--AES_FROUND( x0, x1, x2, x3, y0, y1, y2, y3 );
when aes_enc_st_round_even_in =>
    Irk_addr <= Irk_addr + 4;
    froundin0 <= y0; froundin1 <= y1; froundin2 <= y2;
froundin3 <= y3;

when aes_enc_st_round_even_out =>
    x0 <= froundout0; x1 <= froundout1; x2 <= froundout2; x3
<= froundout3;

    aes_round <= aes_round - 1;

--AES_FSb( x0, x1, x2, x3, y0, y1, y2, y3 );
when aes_enc_st_subbytes_in =>
    Irk_addr <= Irk_addr + 4;
    fsbin0 <= y0; fsbin1 <= y1; fsbin2 <= y2; fsbin3 <= y3;
when aes_enc_st_subbytes_out =>
    x0 <= fsbout0; x1 <= fsbout1; x2 <= fsbout2; x3 <=
fsbout3;

    when aes_enc_st_end =>
        output <= std_logic_vector(x0( 7 downto 0) & x0(15 downto
8) & x0(23 downto 16) & x0(31 downto 24) &
x1( 7 downto 0) & x1(15 downto
8) & x1(23 downto 16) & x1(31 downto 24) &
x2( 7 downto 0) & x2(15 downto
8) & x2(23 downto 16) & x2(31 downto 24) &
x3( 7 downto 0) & x3(15 downto
8) & x3(23 downto 16) & x3(31 downto 24));
        done <= '1';

    end case;
end if;
end process;

process (sreg, start, clear, aes_round)
begin
    if (clear = '1') then
        snext <= aes_enc_st_wait;
    else case sreg is

        when aes_enc_st_wait =>
            if (start = '1') then
                snext <= aes_enc_st_read;
            else
                snext <= aes_enc_st_wait;
            end if;
        when aes_enc_st_read =>
```

```

        snext <= aes_enc_st_addroundkey;
    when aes_enc_st_addroundkey =>
        snext <= aes_enc_st_round_odd_in;
    when aes_enc_st_round_odd_in =>
        snext <= aes_enc_st_round_odd_out;
    when aes_enc_st_round_odd_out =>
        snext <= aes_enc_st_round_even_in;
    when aes_enc_st_round_even_in =>
        snext <= aes_enc_st_round_even_out;
    when aes_enc_st_round_even_out =>
        if (aes_round > 0) then
            snext <= aes_enc_st_round_odd_in;
        else
            snext <= aes_enc_st_round_last_in;
        end if;
    when aes_enc_st_round_last_in =>
        snext <= aes_enc_st_round_last_out;
    when aes_enc_st_round_last_out =>
        snext <= aes_enc_st_subbytes_in;
    when aes_enc_st_subbytes_in =>
        snext <= aes_enc_st_subbytes_out;
    when aes_enc_st_subbytes_out =>
        snext <= aes_enc_st_end;
    when aes_enc_st_end =>
        snext <= aes_enc_st_wait;

    end case;
    end if;
end process;

rk_addr <= Irk_addr;

end arch;
```

### aes\_fround.vhd

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity AES_FROUND is
    port (
        y0, y1, y2, y3, rk0, rk1, rk2, rk3: in UNSIGNED(31 downto 0);
        x0, x1, x2, x3: out UNSIGNED(31 downto 0)
    );
```

```

end AES_FROUND;

architecture arch of AES_FROUND is

-- ROM
type ft_type is ARRAY(0 to 1023) of STD_LOGIC_VECTOR(7 downto 0);
constant ft : ft_type := (
    0 => X"A5", 1 => X"63", 2 => X"63", 3 => X"C6", 4 => X"84", 5 => X"7C", 6 =>
X"7C", 7 => X"F8", 8 => X"99", 9 => X"77", 10 => X"77", 11 => X"EE", 12 => X"8D", 13
=> X"7B", 14 => X"7B", 15 => X"F6",
    16 => X"0D", 17 => X"F2", 18 => X"F2", 19 => X"FF", 20 => X"BD", 21 => X"6B",
22 => X"6B", 23 => X"D6", 24 => X"B1", 25 => X"6F", 26 => X"6F", 27 => X"DE", 28 =>
X"54", 29 => X"C5", 30 => X"C5", 31 => X"91",
    32 => X"50", 33 => X"30", 34 => X"30", 35 => X"60", 36 => X"03", 37 => X"01",
38 => X"01", 39 => X"02", 40 => X"A9", 41 => X"67", 42 => X"67", 43 => X"CE", 44 =>
X"7D", 45 => X"2B", 46 => X"2B", 47 => X"56",
    48 => X"19", 49 => X"FE", 50 => X"FE", 51 => X"E7", 52 => X"62", 53 => X"D7",
54 => X"D7", 55 => X"B5", 56 => X"E6", 57 => X"AB", 58 => X"AB", 59 => X"4D", 60 =>
X"9A", 61 => X"76", 62 => X"76", 63 => X"EC",
    64 => X"45", 65 => X"CA", 66 => X"CA", 67 => X"8F", 68 => X"9D", 69 => X"82",
70 => X"82", 71 => X"1F", 72 => X"40", 73 => X"C9", 74 => X"C9", 75 => X"89", 76 =>
X"87", 77 => X"7D", 78 => X"7D", 79 => X"FA",
    80 => X"15", 81 => X"FA", 82 => X"FA", 83 => X"EF", 84 => X"EB", 85 => X"59",
86 => X"59", 87 => X"B2", 88 => X"C9", 89 => X"47", 90 => X"47", 91 => X"8E", 92 =>
X"0B", 93 => X"F0", 94 => X"F0", 95 => X"FB",
    96 => X"EC", 97 => X"AD", 98 => X"AD", 99 => X"41", 100 => X"67", 101 =>
X"D4", 102 => X"D4", 103 => X"B3", 104 => X"FD", 105 => X"A2", 106 => X"A2", 107 =>
X"5F", 108 => X"EA", 109 => X"AF", 110 => X"AF", 111 => X"45",
    112 => X"BF", 113 => X"9C", 114 => X"9C", 115 => X"23", 116 => X"F7", 117 =>
X"A4", 118 => X"A4", 119 => X"53", 120 => X"96", 121 => X"72", 122 => X"72", 123 =>
X"E4", 124 => X"5B", 125 => X"C0", 126 => X"C0", 127 => X"9B",
    128 => X"C2", 129 => X"B7", 130 => X"B7", 131 => X"75", 132 => X"1C", 133 =>
X"FD", 134 => X"FD", 135 => X"E1", 136 => X"AE", 137 => X"93", 138 => X"93", 139 =>
X"3D", 140 => X"6A", 141 => X"26", 142 => X"26", 143 => X"4C",
    144 => X"5A", 145 => X"36", 146 => X"36", 147 => X"6C", 148 => X"41", 149 =>
X"3F", 150 => X"3F", 151 => X"7E", 152 => X"02", 153 => X"F7", 154 => X"F7", 155 =>
X"F5", 156 => X"4F", 157 => X"CC", 158 => X"CC", 159 => X"83",
    160 => X"5C", 161 => X"34", 162 => X"34", 163 => X"68", 164 => X"F4", 165 =>
X"A5", 166 => X"A5", 167 => X"51", 168 => X"34", 169 => X"E5", 170 => X"E5", 171 =>
X"D1", 172 => X"08", 173 => X"F1", 174 => X"F1", 175 => X"F9",
    176 => X"93", 177 => X"71", 178 => X"71", 179 => X"E2", 180 => X"73", 181 =>
X"D8", 182 => X"D8", 183 => X"AB", 184 => X"53", 185 => X"31", 186 => X"31", 187 =>
X"62", 188 => X"3F", 189 => X"15", 190 => X"15", 191 => X"2A",
    192 => X"0C", 193 => X"04", 194 => X"04", 195 => X"08", 196 => X"52", 197 =>
X"C7", 198 => X"C7", 199 => X"95", 200 => X"65", 201 => X"23", 202 => X"23", 203 =>
X"46", 204 => X"5E", 205 => X"C3", 206 => X"C3", 207 => X"9D",
    208 => X"28", 209 => X"18", 210 => X"18", 211 => X"30", 212 => X"A1", 213 =>

```

### Projeto Fase 3

X"96", 214 => X"96", 215 => X"37", 216 => X"0F", 217 => X"05", 218 => X"05", 219 =>  
 X"0A", 220 => X"B5", 221 => X"9A", 222 => X"9A", 223 => X"2F",  
 224 => X"09", 225 => X"07", 226 => X"07", 227 => X"0E", 228 => X"36", 229 =>  
 X"12", 230 => X"12", 231 => X"24", 232 => X"9B", 233 => X"80", 234 => X"80", 235 =>  
 X"1B", 236 => X"3D", 237 => X"E2", 238 => X"E2", 239 => X"DF",  
 240 => X"26", 241 => X"EB", 242 => X"EB", 243 => X"CD", 244 => X"69", 245 =>  
 X"27", 246 => X"27", 247 => X"4E", 248 => X"CD", 249 => X"B2", 250 => X"B2", 251 =>  
 X"7F", 252 => X"9F", 253 => X"75", 254 => X"75", 255 => X"EA",  
 256 => X"1B", 257 => X"09", 258 => X"09", 259 => X"12", 260 => X"9E", 261 =>  
 X"83", 262 => X"83", 263 => X"1D", 264 => X"74", 265 => X"2C", 266 => X"2C", 267 =>  
 X"58", 268 => X"2E", 269 => X"1A", 270 => X"1A", 271 => X"34",  
 272 => X"2D", 273 => X"1B", 274 => X"1B", 275 => X"36", 276 => X"B2", 277 =>  
 X"6E", 278 => X"6E", 279 => X"DC", 280 => X"EE", 281 => X"5A", 282 => X"5A", 283 =>  
 X"B4", 284 => X"FB", 285 => X"A0", 286 => X"A0", 287 => X"5B",  
 288 => X"F6", 289 => X"52", 290 => X"52", 291 => X"A4", 292 => X"4D", 293 =>  
 X"3B", 294 => X"3B", 295 => X"76", 296 => X"61", 297 => X"D6", 298 => X"D6", 299 =>  
 X"B7", 300 => X"CE", 301 => X"B3", 302 => X"B3", 303 => X"7D",  
 304 => X"7B", 305 => X"29", 306 => X"29", 307 => X"52", 308 => X"3E", 309 =>  
 X"E3", 310 => X"E3", 311 => X"DD", 312 => X"71", 313 => X"2F", 314 => X"2F", 315 =>  
 X"5E", 316 => X"97", 317 => X"84", 318 => X"84", 319 => X"13",  
 320 => X"F5", 321 => X"53", 322 => X"53", 323 => X"A6", 324 => X"68", 325 =>  
 X"D1", 326 => X"D1", 327 => X"B9", 328 => X"00", 329 => X"00", 330 => X"00", 331 =>  
 X"00", 332 => X"2C", 333 => X"ED", 334 => X"ED", 335 => X"C1",  
 336 => X"60", 337 => X"20", 338 => X"20", 339 => X"40", 340 => X"1F", 341 =>  
 X"FC", 342 => X"FC", 343 => X"E3", 344 => X"C8", 345 => X"B1", 346 => X"B1", 347 =>  
 X"79", 348 => X"ED", 349 => X"5B", 350 => X"5B", 351 => X"B6",  
 352 => X"BE", 353 => X"6A", 354 => X"6A", 355 => X"D4", 356 => X"46", 357 =>  
 X"CB", 358 => X"CB", 359 => X"8D", 360 => X"D9", 361 => X"BE", 362 => X"BE", 363 =>  
 X"67", 364 => X"4B", 365 => X"39", 366 => X"39", 367 => X"72",  
 368 => X"DE", 369 => X"4A", 370 => X"4A", 371 => X"94", 372 => X"D4", 373 =>  
 X"4C", 374 => X"4C", 375 => X"98", 376 => X"E8", 377 => X"58", 378 => X"58", 379 =>  
 X"B0", 380 => X"4A", 381 => X"CF", 382 => X"CF", 383 => X"85",  
 384 => X"6B", 385 => X"D0", 386 => X"D0", 387 => X"BB", 388 => X"2A", 389 =>  
 X"EF", 390 => X"EF", 391 => X"C5", 392 => X"E5", 393 => X"AA", 394 => X"AA", 395 =>  
 X"4F", 396 => X"16", 397 => X"FB", 398 => X"FB", 399 => X"ED",  
 400 => X"C5", 401 => X"43", 402 => X"43", 403 => X"86", 404 => X"D7", 405 =>  
 X"4D", 406 => X"4D", 407 => X"9A", 408 => X"55", 409 => X"33", 410 => X"33", 411 =>  
 X"66", 412 => X"94", 413 => X"85", 414 => X"85", 415 => X"11",  
 416 => X"CF", 417 => X"45", 418 => X"45", 419 => X"8A", 420 => X"10", 421 =>  
 X"F9", 422 => X"F9", 423 => X"E9", 424 => X"06", 425 => X"02", 426 => X"02", 427 =>  
 X"04", 428 => X"81", 429 => X"7F", 430 => X"7F", 431 => X"FE",  
 432 => X"F0", 433 => X"50", 434 => X"50", 435 => X"A0", 436 => X"44", 437 =>  
 X"3C", 438 => X"3C", 439 => X"78", 440 => X"BA", 441 => X"9F", 442 => X"9F", 443 =>  
 X"25", 444 => X"E3", 445 => X"A8", 446 => X"A8", 447 => X"4B",  
 448 => X"F3", 449 => X"51", 450 => X"51", 451 => X"A2", 452 => X"FE", 453 =>  
 X"A3", 454 => X"A3", 455 => X"5D", 456 => X"C0", 457 => X"40", 458 => X"40", 459 =>  
 X"80", 460 => X"8A", 461 => X"8F", 462 => X"8F", 463 => X"05",

### Projeto Fase 3

464 => X"AD", 465 => X"92", 466 => X"92", 467 => X"3F", 468 => X"BC", 469 => X"9D", 470 => X"9D", 471 => X"21", 472 => X"48", 473 => X"38", 474 => X"38", 475 => X"70", 476 => X"04", 477 => X"F5", 478 => X"F5", 479 => X"F1",  
480 => X"DF", 481 => X"BC", 482 => X"BC", 483 => X"63", 484 => X"C1", 485 => X"B6", 486 => X"B6", 487 => X"77", 488 => X"75", 489 => X"DA", 490 => X"DA", 491 => X"AF", 492 => X"63", 493 => X"21", 494 => X"21", 495 => X"42",  
496 => X"30", 497 => X"10", 498 => X"10", 499 => X"20", 500 => X"1A", 501 => X"FF", 502 => X"FF", 503 => X"E5", 504 => X"0E", 505 => X"F3", 506 => X"F3", 507 => X"FD", 508 => X"6D", 509 => X"D2", 510 => X"D2", 511 => X"BF",  
512 => X"4C", 513 => X"CD", 514 => X"CD", 515 => X"81", 516 => X"14", 517 => X"0C", 518 => X"0C", 519 => X"18", 520 => X"35", 521 => X"13", 522 => X"13", 523 => X"26", 524 => X"2F", 525 => X"EC", 526 => X"EC", 527 => X"C3",  
528 => X"E1", 529 => X"5F", 530 => X"5F", 531 => X"BE", 532 => X"A2", 533 => X"97", 534 => X"97", 535 => X"35", 536 => X"CC", 537 => X"44", 538 => X"44", 539 => X"88", 540 => X"39", 541 => X"17", 542 => X"17", 543 => X"2E",  
544 => X"57", 545 => X"C4", 546 => X"C4", 547 => X"93", 548 => X"F2", 549 => X"A7", 550 => X"A7", 551 => X"55", 552 => X"82", 553 => X"7E", 554 => X"7E", 555 => X"FC", 556 => X"47", 557 => X"3D", 558 => X"3D", 559 => X"7A",  
560 => X"AC", 561 => X"64", 562 => X"64", 563 => X"C8", 564 => X"E7", 565 => X"5D", 566 => X"5D", 567 => X"BA", 568 => X"2B", 569 => X"19", 570 => X"19", 571 => X"32", 572 => X"95", 573 => X"73", 574 => X"73", 575 => X"E6",  
576 => X"A0", 577 => X"60", 578 => X"60", 579 => X"C0", 580 => X"98", 581 => X"81", 582 => X"81", 583 => X"19", 584 => X"D1", 585 => X"4F", 586 => X"4F", 587 => X"9E", 588 => X"7F", 589 => X"DC", 590 => X"DC", 591 => X"A3",  
592 => X"66", 593 => X"22", 594 => X"22", 595 => X"44", 596 => X"7E", 597 => X"2A", 598 => X"2A", 599 => X"54", 600 => X"AB", 601 => X"90", 602 => X"90", 603 => X"3B", 604 => X"83", 605 => X"88", 606 => X"88", 607 => X"0B",  
608 => X"CA", 609 => X"46", 610 => X"46", 611 => X"8C", 612 => X"29", 613 => X"EE", 614 => X"EE", 615 => X"C7", 616 => X"D3", 617 => X"B8", 618 => X"B8", 619 => X"6B", 620 => X"3C", 621 => X"14", 622 => X"14", 623 => X"28",  
624 => X"79", 625 => X"DE", 626 => X"DE", 627 => X"A7", 628 => X"E2", 629 => X"5E", 630 => X"5E", 631 => X"BC", 632 => X"1D", 633 => X"0B", 634 => X"0B", 635 => X"16", 636 => X"76", 637 => X"DB", 638 => X"DB", 639 => X"AD",  
640 => X"3B", 641 => X"E0", 642 => X"E0", 643 => X"DB", 644 => X"56", 645 => X"32", 646 => X"32", 647 => X"64", 648 => X"4E", 649 => X"3A", 650 => X"3A", 651 => X"74", 652 => X"1E", 653 => X"0A", 654 => X"0A", 655 => X"14",  
656 => X"DB", 657 => X"49", 658 => X"49", 659 => X"92", 660 => X"0A", 661 => X"06", 662 => X"06", 663 => X"0C", 664 => X"6C", 665 => X"24", 666 => X"24", 667 => X"48", 668 => X"E4", 669 => X"5C", 670 => X"5C", 671 => X"B8",  
672 => X"5D", 673 => X"C2", 674 => X"C2", 675 => X"9F", 676 => X"6E", 677 => X"D3", 678 => X"D3", 679 => X"BD", 680 => X"EF", 681 => X"AC", 682 => X"AC", 683 => X"43", 684 => X"A6", 685 => X"62", 686 => X"62", 687 => X"C4",  
688 => X"A8", 689 => X"91", 690 => X"91", 691 => X"39", 692 => X"A4", 693 => X"95", 694 => X"95", 695 => X"31", 696 => X"37", 697 => X"E4", 698 => X"E4", 699 => X"D3", 700 => X"8B", 701 => X"79", 702 => X"79", 703 => X"F2",  
704 => X"32", 705 => X"E7", 706 => X"E7", 707 => X"D5", 708 => X"43", 709 => X"C8", 710 => X"C8", 711 => X"8B", 712 => X"59", 713 => X"37", 714 => X"37", 715 =>

### Projeto Fase 3

X"6E", 716 => X"B7", 717 => X"6D", 718 => X"6D", 719 => X"DA",  
 720 => X"8C", 721 => X"8D", 722 => X"8D", 723 => X"01", 724 => X"64", 725 =>  
 X"D5", 726 => X"D5", 727 => X"B1", 728 => X"D2", 729 => X"4E", 730 => X"4E", 731 =>  
 X"9C", 732 => X"E0", 733 => X"A9", 734 => X"A9", 735 => X"49",  
 736 => X"B4", 737 => X"6C", 738 => X"6C", 739 => X"D8", 740 => X"FA", 741 =>  
 X"56", 742 => X"56", 743 => X"AC", 744 => X"07", 745 => X"F4", 746 => X"F4", 747 =>  
 X"F3", 748 => X"25", 749 => X"EA", 750 => X"EA", 751 => X"CF",  
 752 => X"AF", 753 => X"65", 754 => X"65", 755 => X"CA", 756 => X"8E", 757 =>  
 X"7A", 758 => X"7A", 759 => X"F4", 760 => X"E9", 761 => X"AE", 762 => X"AE", 763 =>  
 X"47", 764 => X"18", 765 => X"08", 766 => X"08", 767 => X"10",  
 768 => X"D5", 769 => X"BA", 770 => X"BA", 771 => X"6F", 772 => X"88", 773 =>  
 X"78", 774 => X"78", 775 => X"F0", 776 => X"6F", 777 => X"25", 778 => X"25", 779 =>  
 X"4A", 780 => X"72", 781 => X"2E", 782 => X"2E", 783 => X"5C",  
 784 => X"24", 785 => X"1C", 786 => X"1C", 787 => X"38", 788 => X"F1", 789 =>  
 X"A6", 790 => X"A6", 791 => X"57", 792 => X"C7", 793 => X"B4", 794 => X"B4", 795 =>  
 X"73", 796 => X"51", 797 => X"C6", 798 => X"C6", 799 => X"97",  
 800 => X"23", 801 => X"E8", 802 => X"E8", 803 => X"CB", 804 => X"7C", 805 =>  
 X"DD", 806 => X"DD", 807 => X"A1", 808 => X"9C", 809 => X"74", 810 => X"74", 811 =>  
 X"E8", 812 => X"21", 813 => X"1F", 814 => X"1F", 815 => X"3E",  
 816 => X"DD", 817 => X"4B", 818 => X"4B", 819 => X"96", 820 => X"DC", 821 =>  
 X"BD", 822 => X"BD", 823 => X"61", 824 => X"86", 825 => X"8B", 826 => X"8B", 827 =>  
 X"0D", 828 => X"85", 829 => X"8A", 830 => X"8A", 831 => X"0F",  
 832 => X"90", 833 => X"70", 834 => X"70", 835 => X"E0", 836 => X"42", 837 =>  
 X"3E", 838 => X"3E", 839 => X"7C", 840 => X"C4", 841 => X"B5", 842 => X"B5", 843 =>  
 X"71", 844 => X"AA", 845 => X"66", 846 => X"66", 847 => X"CC",  
 848 => X"D8", 849 => X"48", 850 => X"48", 851 => X"90", 852 => X"05", 853 =>  
 X"03", 854 => X"03", 855 => X"06", 856 => X"01", 857 => X"F6", 858 => X"F6", 859 =>  
 X"F7", 860 => X"12", 861 => X"0E", 862 => X"0E", 863 => X"1C",  
 864 => X"A3", 865 => X"61", 866 => X"61", 867 => X"C2", 868 => X"5F", 869 =>  
 X"35", 870 => X"35", 871 => X"6A", 872 => X"F9", 873 => X"57", 874 => X"57", 875 =>  
 X"AE", 876 => X"D0", 877 => X"B9", 878 => X"B9", 879 => X"69",  
 880 => X"91", 881 => X"86", 882 => X"86", 883 => X"17", 884 => X"58", 885 =>  
 X"C1", 886 => X"C1", 887 => X"99", 888 => X"27", 889 => X"1D", 890 => X"1D", 891 =>  
 X"3A", 892 => X"B9", 893 => X"9E", 894 => X"9E", 895 => X"27",  
 896 => X"38", 897 => X"E1", 898 => X"E1", 899 => X"D9", 900 => X"13", 901 =>  
 X"F8", 902 => X"F8", 903 => X"EB", 904 => X"B3", 905 => X"98", 906 => X"98", 907 =>  
 X"2B", 908 => X"33", 909 => X"11", 910 => X"11", 911 => X"22",  
 912 => X"BB", 913 => X"69", 914 => X"69", 915 => X"D2", 916 => X"70", 917 =>  
 X"D9", 918 => X"D9", 919 => X"A9", 920 => X"89", 921 => X"8E", 922 => X"8E", 923 =>  
 X"07", 924 => X"A7", 925 => X"94", 926 => X"94", 927 => X"33",  
 928 => X"B6", 929 => X"9B", 930 => X"9B", 931 => X"2D", 932 => X"22", 933 =>  
 X"1E", 934 => X"1E", 935 => X"3C", 936 => X"92", 937 => X"87", 938 => X"87", 939 =>  
 X"15", 940 => X"20", 941 => X"E9", 942 => X"E9", 943 => X"C9",  
 944 => X"49", 945 => X"CE", 946 => X"CE", 947 => X"87", 948 => X"FF", 949 =>  
 X"55", 950 => X"55", 951 => X"AA", 952 => X"78", 953 => X"28", 954 => X"28", 955 =>  
 X"50", 956 => X"7A", 957 => X"DF", 958 => X"DF", 959 => X"A5",  
 960 => X"8F", 961 => X"8C", 962 => X"8C", 963 => X"03", 964 => X"F8", 965 =>



```
X"A1", 966 => X"A1", 967 => X"59", 968 => X"80", 969 => X"89", 970 => X"89", 971 =>
X"09", 972 => X"17", 973 => X"0D", 974 => X"0D", 975 => X"1A",
    976 => X"DA", 977 => X"BF", 978 => X"BF", 979 => X"65", 980 => X"31", 981 =>
X"E6", 982 => X"E6", 983 => X"D7", 984 => X"C6", 985 => X"42", 986 => X"42", 987 =>
X"84", 988 => X"B8", 989 => X"68", 990 => X"68", 991 => X"D0",
    992 => X"C3", 993 => X"41", 994 => X"41", 995 => X"82", 996 => X"B0", 997 =>
X"99", 998 => X"99", 999 => X"29", 1000 => X"77", 1001 => X"2D", 1002 => X"2D", 1003
=> X"5A", 1004 => X"11", 1005 => X"0F", 1006 => X"0F", 1007 => X"1E",
    1008 => X"CB", 1009 => X"B0", 1010 => X"B0", 1011 => X"7B", 1012 => X"FC",
1013 => X"54", 1014 => X"54", 1015 => X"A8", 1016 => X"D6", 1017 => X"BB", 1018 =>
X"BB", 1019 => X"6D", 1020 => X"3A", 1021 => X"16", 1022 => X"16", 1023 => X"2C"
);
```

**begin**

**process** (y0, y1, y2, y3, rk0, rk1, rk2, rk3)

**begin**

```
    x0 <= unsigned(std_logic_vector(rk0) xor ((ft(to_integer(y0( 7 downto 0) &
"00")) & ft(to_integer(y0( 7 downto 0) & "01")) & ft(to_integer(y0( 7 downto 0) &
"10")) & ft(to_integer(y0( 7 downto 0) & "11")))) xor
        ((ft(to_integer(y1(15 downto 8) &
"01")) & ft(to_integer(y1(15 downto 8) & "10")) & ft(to_integer(y1(15 downto 8) &
"11")) & ft(to_integer(y1(15 downto 8) & "00")))) xor
        ((ft(to_integer(y2(23 downto 16) &
"10")) & ft(to_integer(y2(23 downto 16) & "11")) & ft(to_integer(y2(23 downto 16) &
"00")) & ft(to_integer(y2(23 downto 16) & "01")))) xor
        ((ft(to_integer(y3(31 downto 24) &
"11")) & ft(to_integer(y3(31 downto 24) & "00")) & ft(to_integer(y3(31 downto 24) &
"01")) & ft(to_integer(y3(31 downto 24) & "10"))))));
```

```
    x1 <= unsigned(std_logic_vector(rk1) xor ((ft(to_integer(y1( 7 downto 0) &
"00")) & ft(to_integer(y1( 7 downto 0) & "01")) & ft(to_integer(y1( 7 downto 0) &
"10")) & ft(to_integer(y1( 7 downto 0) & "11")))) xor
        ((ft(to_integer(y2(15 downto 8) &
"01")) & ft(to_integer(y2(15 downto 8) & "10")) & ft(to_integer(y2(15 downto 8) &
"11")) & ft(to_integer(y2(15 downto 8) & "00")))) xor
        ((ft(to_integer(y3(23 downto 16) &
"10")) & ft(to_integer(y3(23 downto 16) & "11")) & ft(to_integer(y3(23 downto 16) &
"00")) & ft(to_integer(y3(23 downto 16) & "01")))) xor
        ((ft(to_integer(y0(31 downto 24) &
"11")) & ft(to_integer(y0(31 downto 24) & "00")) & ft(to_integer(y0(31 downto 24) &
"01")) & ft(to_integer(y0(31 downto 24) & "10"))))));
```

```
    x2 <= unsigned(std_logic_vector(rk2) xor ((ft(to_integer(y2( 7 downto 0) &
"00")) & ft(to_integer(y2( 7 downto 0) & "01")) & ft(to_integer(y2( 7 downto 0) &
"10")) & ft(to_integer(y2( 7 downto 0) & "11")))) xor
```

### Projeto Fase 3

```

                                ((ft(to_integer(y3(15 downto 8) &
"01")) & ft(to_integer(y3(15 downto 8) & "10")) & ft(to_integer(y3(15 downto 8) &
"11")) & ft(to_integer(y3(15 downto 8) & "00")))) xor
                                ((ft(to_integer(y0(23 downto 16) &
"10")) & ft(to_integer(y0(23 downto 16) & "11")) & ft(to_integer(y0(23 downto 16) &
"00")) & ft(to_integer(y0(23 downto 16) & "01")))) xor
                                ((ft(to_integer(y1(31 downto 24) &
"11")) & ft(to_integer(y1(31 downto 24) & "00")) & ft(to_integer(y1(31 downto 24) &
"01")) & ft(to_integer(y1(31 downto 24) & "10"))));

    x3 <= unsigned(std_logic_vector(rk3) xor ((ft(to_integer(y3( 7 downto 0) &
"00")) & ft(to_integer(y3( 7 downto 0) & "01")) & ft(to_integer(y3( 7 downto 0) &
"10")) & ft(to_integer(y3( 7 downto 0) & "11")))) xor
                                ((ft(to_integer(y0(15 downto 8) &
"01")) & ft(to_integer(y0(15 downto 8) & "10")) & ft(to_integer(y0(15 downto 8) &
"11")) & ft(to_integer(y0(15 downto 8) & "00")))) xor
                                ((ft(to_integer(y1(23 downto 16) &
"10")) & ft(to_integer(y1(23 downto 16) & "11")) & ft(to_integer(y1(23 downto 16) &
"00")) & ft(to_integer(y1(23 downto 16) & "01")))) xor
                                ((ft(to_integer(y2(31 downto 24) &
"11")) & ft(to_integer(y2(31 downto 24) & "00")) & ft(to_integer(y2(31 downto 24) &
"01")) & ft(to_integer(y2(31 downto 24) & "10"))));

end process;

end arch;
```

aes\_fsb.vhd

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity AES_FSB is
    port (
        y0, y1, y2, y3, rk0, rk1, rk2, rk3: in UNSIGNED(31 downto 0);
        x0, x1, x2, x3: out UNSIGNED(31 downto 0)
    );
end AES_FSB;

architecture arch of AES_FSB is

    -- ROM
    type fsb_type is ARRAY(0 to 255) of STD_LOGIC_VECTOR(7 downto 0);
    constant fsb : fsb_type := (
```

### Projeto Fase 3

0 => X"63", 1 => X"7C", 2 => X"77", 3 => X"7B", 4 => X"F2", 5 => X"6B", 6 => X"6F", 7 => X"C5",  
8 => X"30", 9 => X"01", 10 => X"67", 11 => X"2B", 12 => X"FE", 13 => X"D7", 14 => X"AB", 15 => X"76",  
16 => X"CA", 17 => X"82", 18 => X"C9", 19 => X"7D", 20 => X"FA", 21 => X"59", 22 => X"47", 23 => X"F0",  
24 => X"AD", 25 => X"D4", 26 => X"A2", 27 => X"AF", 28 => X"9C", 29 => X"A4", 30 => X"72", 31 => X"C0",  
32 => X"B7", 33 => X"FD", 34 => X"93", 35 => X"26", 36 => X"36", 37 => X"3F", 38 => X"F7", 39 => X"CC",  
40 => X"34", 41 => X"A5", 42 => X"E5", 43 => X"F1", 44 => X"71", 45 => X"D8", 46 => X"31", 47 => X"15",  
48 => X"04", 49 => X"C7", 50 => X"23", 51 => X"C3", 52 => X"18", 53 => X"96", 54 => X"05", 55 => X"9A",  
56 => X"07", 57 => X"12", 58 => X"80", 59 => X"E2", 60 => X"EB", 61 => X"27", 62 => X"B2", 63 => X"75",  
64 => X"09", 65 => X"83", 66 => X"2C", 67 => X"1A", 68 => X"1B", 69 => X"6E", 70 => X"5A", 71 => X"A0",  
72 => X"52", 73 => X"3B", 74 => X"D6", 75 => X"B3", 76 => X"29", 77 => X"E3", 78 => X"2F", 79 => X"84",  
80 => X"53", 81 => X"D1", 82 => X"00", 83 => X"ED", 84 => X"20", 85 => X"FC", 86 => X"B1", 87 => X"5B",  
88 => X"6A", 89 => X"CB", 90 => X"BE", 91 => X"39", 92 => X"4A", 93 => X"4C", 94 => X"58", 95 => X"CF",  
96 => X"D0", 97 => X"EF", 98 => X"AA", 99 => X"FB", 100 => X"43", 101 => X"4D", 102 => X"33", 103 => X"85",  
104 => X"45", 105 => X"F9", 106 => X"02", 107 => X"7F", 108 => X"50", 109 => X"3C", 110 => X"9F", 111 => X"A8",  
112 => X"51", 113 => X"A3", 114 => X"40", 115 => X"8F", 116 => X"92", 117 => X"9D", 118 => X"38", 119 => X"F5",  
120 => X"BC", 121 => X"B6", 122 => X"DA", 123 => X"21", 124 => X"10", 125 => X"FF", 126 => X"F3", 127 => X"D2",  
128 => X"CD", 129 => X"0C", 130 => X"13", 131 => X"EC", 132 => X"5F", 133 => X"97", 134 => X"44", 135 => X"17",  
136 => X"C4", 137 => X"A7", 138 => X"7E", 139 => X"3D", 140 => X"64", 141 => X"5D", 142 => X"19", 143 => X"73",  
144 => X"60", 145 => X"81", 146 => X"4F", 147 => X"DC", 148 => X"22", 149 => X"2A", 150 => X"90", 151 => X"88",  
152 => X"46", 153 => X"EE", 154 => X"B8", 155 => X"14", 156 => X"DE", 157 => X"5E", 158 => X"0B", 159 => X"DB",  
160 => X"E0", 161 => X"32", 162 => X"3A", 163 => X"0A", 164 => X"49", 165 => X"06", 166 => X"24", 167 => X"5C",  
168 => X"C2", 169 => X"D3", 170 => X"AC", 171 => X"62", 172 => X"91", 173 => X"95", 174 => X"E4", 175 => X"79",  
176 => X"E7", 177 => X"C8", 178 => X"37", 179 => X"6D", 180 => X"8D", 181 => X"D5", 182 => X"4E", 183 => X"A9",  
184 => X"6C", 185 => X"56", 186 => X"F4", 187 => X"EA", 188 => X"65", 189 =>

```

X"7A", 190 => X"AE", 191 => X"08",
    192 => X"BA", 193 => X"78", 194 => X"25", 195 => X"2E", 196 => X"1C", 197 =>
X"A6", 198 => X"B4", 199 => X"C6",
    200 => X"E8", 201 => X"DD", 202 => X"74", 203 => X"1F", 204 => X"4B", 205 =>
X"BD", 206 => X"8B", 207 => X"8A",
    208 => X"70", 209 => X"3E", 210 => X"B5", 211 => X"66", 212 => X"48", 213 =>
X"03", 214 => X"F6", 215 => X"0E",
    216 => X"61", 217 => X"35", 218 => X"57", 219 => X"B9", 220 => X"86", 221 =>
X"C1", 222 => X"1D", 223 => X"9E",
    224 => X"E1", 225 => X"F8", 226 => X"98", 227 => X"11", 228 => X"69", 229 =>
X"D9", 230 => X"8E", 231 => X"94",
    232 => X"9B", 233 => X"1E", 234 => X"87", 235 => X"E9", 236 => X"CE", 237 =>
X"55", 238 => X"28", 239 => X"DF",
    240 => X"8C", 241 => X"A1", 242 => X"89", 243 => X"0D", 244 => X"BF", 245 =>
X"E6", 246 => X"42", 247 => X"68",
    248 => X"41", 249 => X"99", 250 => X"2D", 251 => X"0F", 252 => X"B0", 253 =>
X"54", 254 => X"BB", 255 => X"16"
);

begin

process (y0, y1, y2, y3, rk0, rk1, rk2, rk3)
begin

    x0 <= unsigned(std_logic_vector(rk0) xor (fsb(to_integer(y3(31 downto 24))) &
fsb(to_integer(y2(23 downto 16))) &
fsb(to_integer(y1(15 downto 8))) &
fsb(to_integer(y0( 7 downto 0))))));

    x1 <= unsigned(std_logic_vector(rk1) xor (fsb(to_integer(y0(31 downto 24))) &
fsb(to_integer(y3(23 downto 16))) &
fsb(to_integer(y2(15 downto 8))) &
fsb(to_integer(y1( 7 downto 0))))));

    x2 <= unsigned(std_logic_vector(rk2) xor (fsb(to_integer(y1(31 downto 24))) &
fsb(to_integer(y0(23 downto 16))) &
fsb(to_integer(y3(15 downto 8))) &
fsb(to_integer(y2( 7 downto 0))))));

    x3 <= unsigned(std_logic_vector(rk3) xor (fsb(to_integer(y2(31 downto 24))) &
fsb(to_integer(y1(23 downto 16))) &
fsb(to_integer(y0(15 downto 8))) &
fsb(to_integer(y3( 7 downto 0))))));

end process;

end arch;

```

aes\_nonce\_counter.vhd

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity AES_Nonce_Counter is
    port (
        clock, clear, increment: in STD_LOGIC;
        original_nc: in STD_LOGIC_VECTOR(127 downto 0);
        current_nc: out STD_LOGIC_VECTOR(127 downto 0)
    );
end AES_Nonce_Counter;

architecture arch of AES_Nonce_Counter is

    -- Internal signal
    signal nc: UNSIGNED(127 downto 0) := (others => '0');

begin

    process (clock, clear, increment)
    begin
        if clock'event and clock = '1' then
            if clear = '1' then
                nc <= unsigned(original_nc);
            elsif increment = '1' then
                nc <= nc + 1;
            end if;
        end if;
    end process;

    current_nc <= std_logic_vector(nc);

end arch;

```

aes\_roundkey.vhd

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity AES_RoundKey is

```

```

    port (
        clock, start, clear: in STD_LOGIC;
        rk_addr: in UNSIGNED(5 downto 0);
        key: in STD_LOGIC_VECTOR(127 downto 0);
        rk0, rk1, rk2, rk3: out UNSIGNED(31 downto 0);
        done: out STD_LOGIC
    );
end AES_RoundKey;

architecture arch of AES_RoundKey is

-- State machine
type estados is (aes_rk_st_wait, aes_rk_st_read, aes_rk_st_generate_begin,
aes_rk_st_generate_end, aes_rk_st_end);
signal sreg, snext: estados;

-- RAM
type memoria_type is ARRAY(0 to 43) of STD_LOGIC_VECTOR(31 downto 0);
signal memoria: memoria_type := (others => (others => '0'));

-- RCON ROM
type rcon_type is ARRAY(0 to 9) of STD_LOGIC_VECTOR(7 downto 0);
constant rcon : rcon_type := (
    0 => X"01", 1 => X"02", 2 => X"04", 3 => X"08", 4 => X"10",
    5 => X"20", 6 => X"40", 7 => X"80", 8 => X"1B", 9 => X"36"
);

-- FSb ROM
type fsb_type is ARRAY(0 to 255) of STD_LOGIC_VECTOR(7 downto 0);
constant fsb : fsb_type := (
    0 => X"63", 1 => X"7C", 2 => X"77", 3 => X"7B", 4 => X"F2", 5 => X"6B", 6 =>
X"6F", 7 => X"C5",
    8 => X"30", 9 => X"01", 10 => X"67", 11 => X"2B", 12 => X"FE", 13 => X"D7", 14
=> X"AB", 15 => X"76",
    16 => X"CA", 17 => X"82", 18 => X"C9", 19 => X"7D", 20 => X"FA", 21 => X"59",
22 => X"47", 23 => X"F0",
    24 => X"AD", 25 => X"D4", 26 => X"A2", 27 => X"AF", 28 => X"9C", 29 => X"A4",
30 => X"72", 31 => X"C0",
    32 => X"B7", 33 => X"FD", 34 => X"93", 35 => X"26", 36 => X"36", 37 => X"3F",
38 => X"F7", 39 => X"CC",
    40 => X"34", 41 => X"A5", 42 => X"E5", 43 => X"F1", 44 => X"71", 45 => X"D8",
46 => X"31", 47 => X"15",
    48 => X"04", 49 => X"C7", 50 => X"23", 51 => X"C3", 52 => X"18", 53 => X"96",
54 => X"05", 55 => X"9A",
    56 => X"07", 57 => X"12", 58 => X"80", 59 => X"E2", 60 => X"EB", 61 => X"27",
62 => X"B2", 63 => X"75",
    64 => X"09", 65 => X"83", 66 => X"2C", 67 => X"1A", 68 => X"1B", 69 => X"6E",

```

```

70 => X"5A", 71 => X"A0",
    72 => X"52", 73 => X"3B", 74 => X"D6", 75 => X"B3", 76 => X"29", 77 => X"E3",
78 => X"2F", 79 => X"84",
    80 => X"53", 81 => X"D1", 82 => X"00", 83 => X"ED", 84 => X"20", 85 => X"FC",
86 => X"B1", 87 => X"5B",
    88 => X"6A", 89 => X"CB", 90 => X"BE", 91 => X"39", 92 => X"4A", 93 => X"4C",
94 => X"58", 95 => X"CF",
    96 => X"D0", 97 => X"EF", 98 => X"AA", 99 => X"FB", 100 => X"43", 101 =>
X"4D", 102 => X"33", 103 => X"85",
    104 => X"45", 105 => X"F9", 106 => X"02", 107 => X"7F", 108 => X"50", 109 =>
X"3C", 110 => X"9F", 111 => X"A8",
    112 => X"51", 113 => X"A3", 114 => X"40", 115 => X"8F", 116 => X"92", 117 =>
X"9D", 118 => X"38", 119 => X"F5",
    120 => X"BC", 121 => X"B6", 122 => X"DA", 123 => X"21", 124 => X"10", 125 =>
X"FF", 126 => X"F3", 127 => X"D2",
    128 => X"CD", 129 => X"0C", 130 => X"13", 131 => X"EC", 132 => X"5F", 133 =>
X"97", 134 => X"44", 135 => X"17",
    136 => X"C4", 137 => X"A7", 138 => X"7E", 139 => X"3D", 140 => X"64", 141 =>
X"5D", 142 => X"19", 143 => X"73",
    144 => X"60", 145 => X"81", 146 => X"4F", 147 => X"DC", 148 => X"22", 149 =>
X"2A", 150 => X"90", 151 => X"88",
    152 => X"46", 153 => X"EE", 154 => X"B8", 155 => X"14", 156 => X"DE", 157 =>
X"5E", 158 => X"0B", 159 => X"DB",
    160 => X"E0", 161 => X"32", 162 => X"3A", 163 => X"0A", 164 => X"49", 165 =>
X"06", 166 => X"24", 167 => X"5C",
    168 => X"C2", 169 => X"D3", 170 => X"AC", 171 => X"62", 172 => X"91", 173 =>
X"95", 174 => X"E4", 175 => X"79",
    176 => X"E7", 177 => X"C8", 178 => X"37", 179 => X"6D", 180 => X"8D", 181 =>
X"D5", 182 => X"4E", 183 => X"A9",
    184 => X"6C", 185 => X"56", 186 => X"F4", 187 => X"EA", 188 => X"65", 189 =>
X"7A", 190 => X"AE", 191 => X"08",
    192 => X"BA", 193 => X"78", 194 => X"25", 195 => X"2E", 196 => X"1C", 197 =>
X"A6", 198 => X"B4", 199 => X"C6",
    200 => X"E8", 201 => X"DD", 202 => X"74", 203 => X"1F", 204 => X"4B", 205 =>
X"BD", 206 => X"8B", 207 => X"8A",
    208 => X"70", 209 => X"3E", 210 => X"B5", 211 => X"66", 212 => X"48", 213 =>
X"03", 214 => X"F6", 215 => X"0E",
    216 => X"61", 217 => X"35", 218 => X"57", 219 => X"B9", 220 => X"86", 221 =>
X"C1", 222 => X"1D", 223 => X"9E",
    224 => X"E1", 225 => X"F8", 226 => X"98", 227 => X"11", 228 => X"69", 229 =>
X"D9", 230 => X"8E", 231 => X"94",
    232 => X"9B", 233 => X"1E", 234 => X"87", 235 => X"E9", 236 => X"CE", 237 =>
X"55", 238 => X"28", 239 => X"DF",
    240 => X"8C", 241 => X"A1", 242 => X"89", 243 => X"0D", 244 => X"BF", 245 =>
X"E6", 246 => X"42", 247 => X"68",
    248 => X"41", 249 => X"99", 250 => X"2D", 251 => X"0F", 252 => X"B0", 253 =>
X"54", 254 => X"BB", 255 => X"16"

```

```
);

-- Internal signals
signal round: UNSIGNED(5 downto 0);

begin

process (rk_addr, memoria)
begin
    if (rk_addr <= 40) then
        rk0 <= unsigned(memoria(to_integer(rk_addr)));
        rk1 <= unsigned(memoria(to_integer(rk_addr + 1)));
        rk2 <= unsigned(memoria(to_integer(rk_addr + 2)));
        rk3 <= unsigned(memoria(to_integer(rk_addr + 3)));
    else
        rk0 <= (others => '0');
        rk1 <= (others => '0');
        rk2 <= (others => '0');
        rk3 <= (others => '0');
    end if;
end process;

process (clock)
begin
    if clock'event and clock = '1' then
        sreg <= snext;

        case sreg is
            when aes_rk_st_wait =>
                round <= to_unsigned(0, round'length);
                done <= '0';
            when aes_rk_st_read =>
                round <= to_unsigned(0, round'length);
                memoria(0) <= key(103 downto 96) & key(111 downto 104) & key(119
downto 112) & key(127 downto 120);
                memoria(1) <= key( 71 downto 64) & key( 79 downto 72) & key( 87
downto 80) & key( 95 downto 88);
                memoria(2) <= key( 39 downto 32) & key( 47 downto 40) & key( 55
downto 48) & key( 63 downto 56);
                memoria(3) <= key(  7 downto  0) & key( 15 downto  8) & key( 23
downto 16) & key( 31 downto 24);
            when aes_rk_st_generate_begin =>
                if (round < 10) then
                    memoria(4 + to_integer(round sll 2)) <= (memoria(0
+ to_integer(round sll 2)) xor ("000000" & rcon(to_integer(round))) xor
(
                        fsb( to_integer(unsigned(memoria(3 +
```



```

(to_integer(round sll 2))( 7 downto 0) )) ) &
        fsb( to_integer(unsigned(memoria(3 +
(to_integer(round sll 2))(31 downto 24) )) ) &
        fsb( to_integer(unsigned(memoria(3 +
(to_integer(round sll 2))(23 downto 16) )) ) &
        fsb( to_integer(unsigned(memoria(3 +
(to_integer(round sll 2))(15 downto 8) )) )
    );
    end if;
    when aes_rk_st_generate_end =>
        if (round < 10) then
            memoria(5 + to_integer(round sll 2)) <= memoria(1 +
to_integer(round sll 2)) xor memoria(4 + to_integer(round sll 2));
            memoria(6 + to_integer(round sll 2)) <= memoria(1 +
to_integer(round sll 2)) xor memoria(2 + to_integer(round sll 2)) xor memoria(4 +
to_integer(round sll 2));
            memoria(7 + to_integer(round sll 2)) <= memoria(1 +
to_integer(round sll 2)) xor memoria(2 + to_integer(round sll 2)) xor memoria(3 +
to_integer(round sll 2)) xor memoria(4 + to_integer(round sll 2));
            round <= round + 1;
        end if;
    when aes_rk_st_end =>
        done <= '1';

    end case;
end if;
end process;

process (sreg, start, clear, round)
begin
    if (clear = '1') then
        snext <= aes_rk_st_wait;
    else case sreg is
        when aes_rk_st_wait =>
            if (start = '1') then
                snext <= aes_rk_st_read;
            else
                snext <= aes_rk_st_wait;
            end if;
        when aes_rk_st_read =>
            snext <= aes_rk_st_generate_begin;
        when aes_rk_st_generate_begin =>
            snext <= aes_rk_st_generate_end;
        when aes_rk_st_generate_end =>
            if (round < 10) then
                snext <= aes_rk_st_generate_begin;
            else

```

```

        snext <= aes_rk_st_end;
    end if;
    when aes_rk_st_end =>
        snext <= aes_rk_st_wait;
    end case;
    end if;
end process;

end arch;

```

### 2.4 Execução de testbenches em VHDL para verificação do funcionamento.

Testbenches foram escritos e rodados no Vivado. Os testes inicialmente falharam e utilizamos as funcionalidades de depuração do Vivado para visualizar sinais internos ao circuito. Com mais etapas de depuração e desenvolvimento, foi possível efetuar o testbench e confirmar o funcionamento do hardware.

O código do testbench pode ser encontrado a seguir:

tb\_aes\_ctr.vhd

```

-----
--! @file tb_aes_ctr.vhd
--! @date 20170730
--! @brief Testbench para o encrypt
-----

entity tb_aes_ctr is end;

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

architecture stim of tb_aes_ctr is
    component AES_CTR is
    port (
        clock, start, clear: in STD_LOGIC;
        input, nonce_counter, key: in STD_LOGIC_VECTOR(127 downto 0);
        output: out STD_LOGIC_VECTOR(127 downto 0);
        done: out STD_LOGIC
    );
    end component;

    constant periodoClock : time := 1 ms;

    signal clock, start, clear: std_logic := '0';
    signal input, nonce_counter, key: STD_LOGIC_VECTOR(127 downto 0) := (others =>

```

```
'0');
signal output : STD_LOGIC_VECTOR(127 downto 0);
signal done : STD_LOGIC;

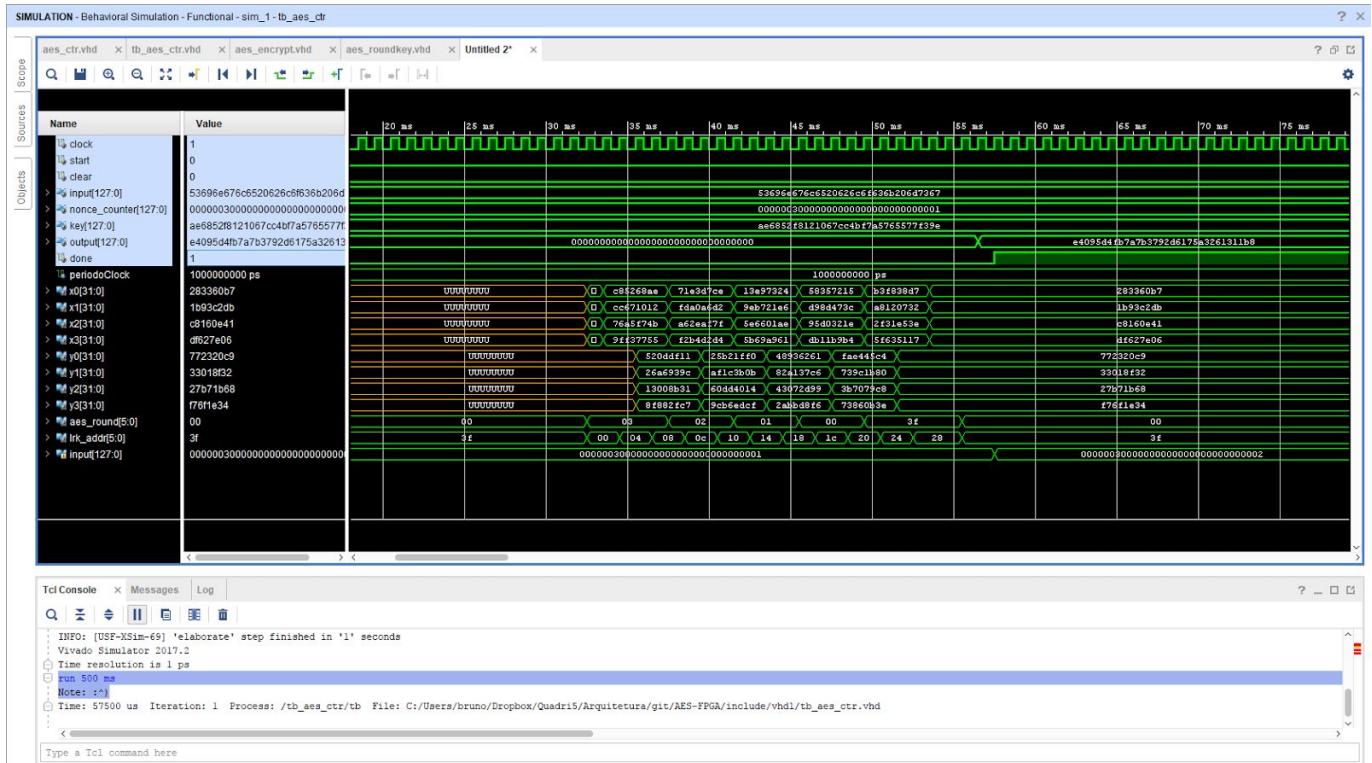
begin
    uut: AES_CTR port map(clock, start, clear, input, nonce_counter, key, output,
done);
    clock <= not clock after periodoClock/2;

    tb: process
    begin
        input <= x"53696E676C6520626C6F636B206D7367";
        nonce_counter <= x"00000030000000000000000000000001";
        key <= x"AE6852F8121067CC4BF7A5765577F39E";
        -- Ciclo de reset
        clear <= '1';
        wait for 2 ms;
        clear <= '0';
        start <= '1';
        wait for 2 ms;
        start <= '0';
        wait until done = '1';

        assert output = x"E4095D4FB7A7B3792D6175A3261311B8"
            report "Nope"
            severity failure;

        report "(:^)"
            severity note;
        wait;
    end process;
end architecture stim;
```

Um screenshot das formas de ondas obtidas do testbench pode ser encontrado a seguir:



## 2.5 Sintetizar para Nexys 4 DDR.

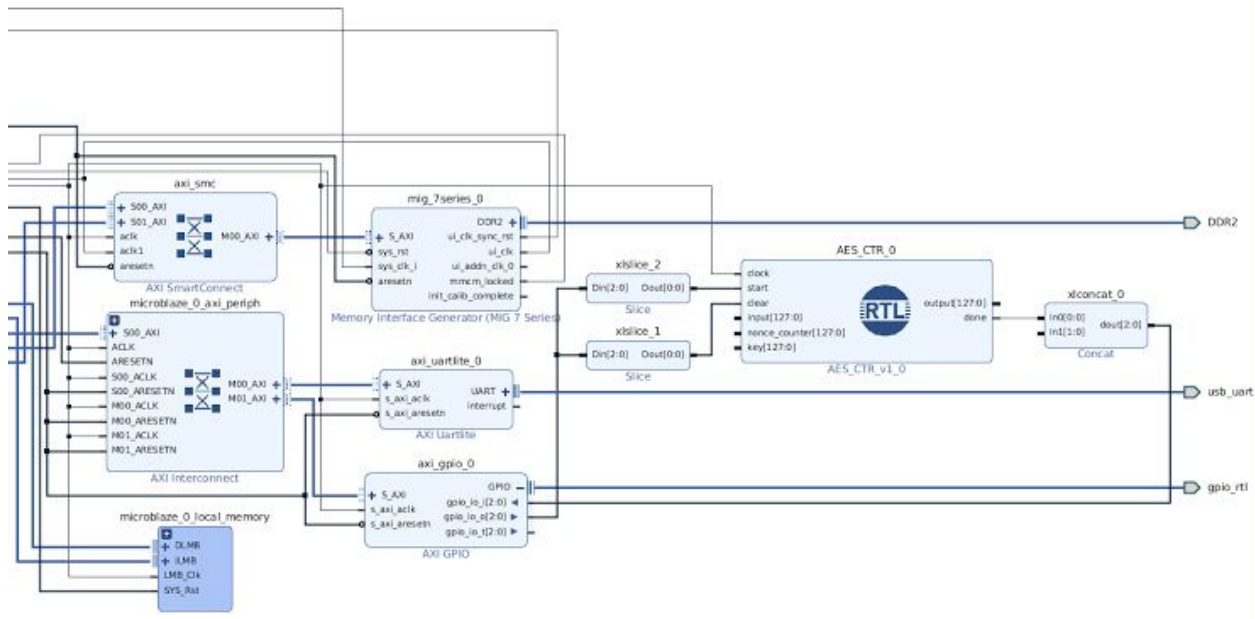
Para sintetizar para a Nexys 4 DDR, fizemos alterações nos periféricos conectados ao software e mapeamento de memória.

Criamos o IP do projeto abrindo Design Sources e selecionando botão direito -> “Add Module To Block Design” no nosso componente.

Adicionamos um componente AXI GPIO, usando os pinos de GPIO conectados no barramento AXI para enviar e receber sinais de controle. A referência **AXI GPIO v2.0 - LogiCORE IP Product Guide**, encontrada no site da Xilinx, foi usada. O componente, conectado no barramento AXI, converte valores recebidos em seu mapa de memória para sinais GPIO simples, que poderão ser facilmente lidos pelo nosso circuito. O componente teve seu sinal S\_AXI conectado aos sinais do barramento, s\_axi\_aclk ao clock, e s\_axi\_aresetn ao sinal de resetn do barramento. Os sinais gpio\_rtl foram deixados custom e conectados ao nosso IP.

O componente AXI GPIO fornece saídas e entradas em um barramento (vetor), que não podem ser conectadas diretamente aos pinos do nosso componente. Após trabalho bastante árduo para decifrar o motivo das conexões não funcionarem, encontramos os componentes *slice* e *concat*, que convertem um vetor de sinais a vetores menores ou vice-versa. Dois componentes slice foram usados para gerar *start* e *clear*, e um concat para gerar um barramento a partir de *done*:

## Projeto Fase 3

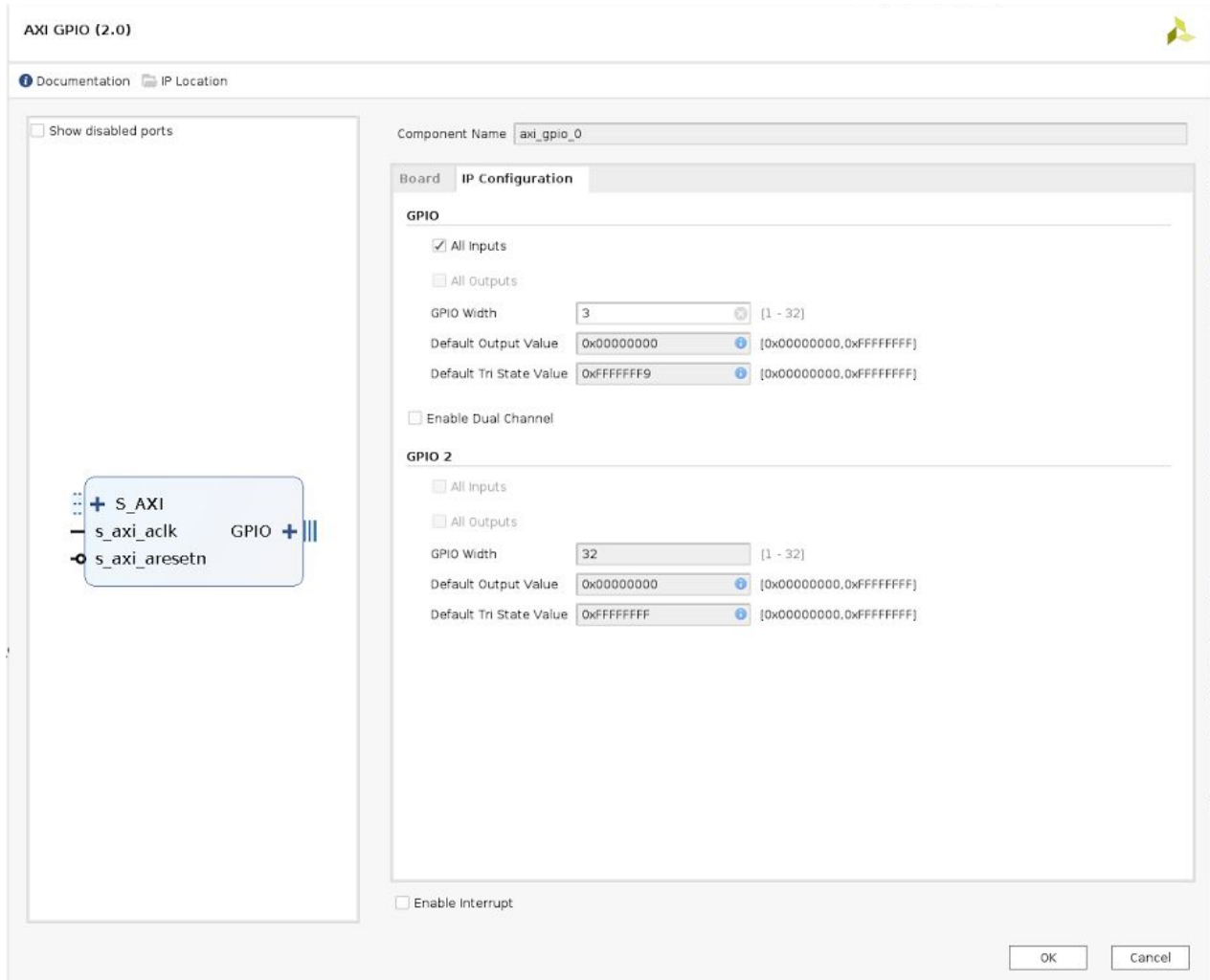


O componente `axi_gpio` foi mantido com o mapeamento de memória gerado pelo Vivado:

Cell	Slave Interface	Base Name	Offset Address	Range	High Address
<div> <div> <div></div> <div>microblaze_0</div> </div> <div> <div></div> <div>Data (32 address bits : 4G)</div> </div> </div>					
axi_uartlite_0	S_AXI	Reg	0x4060_0000	64K	0x4060_FFFF
microblaze_0_local_memory/dlmb_bram_if_cntlr	SLMB	Mem	0x0000_0000	32K	0x0000_7FFF
mig_7series_0	S_AXI	memaddr	0x8000_0000	128M	0x87FF_FFFF
axi_gpio_0	S_AXI	Reg	0x4000_0000	64K	0x4000_FFFF
<div> <div> <div></div> <div>Instruction (32 address bits : 4G)</div> </div> <div> <div></div> <div></div> </div> </div>					
microblaze_0_local_memory/ilmb_bram_if_cntlr	SLMB	Mem	0x0000_0000	32K	0x0000_7FFF
mig_7series_0	S_AXI	memaddr	0x8000_0000	128M	0x87FF_FFFF

Configuramos as conexões da GPIO como custom, sem conectar a nenhum sinal já definido da placa, utilizamos a configuração em single channel, sem suporte a interrupções e limitamos o controlador a 3 pinos. O valor padrão de saída foi colocado em 0. Dos 3 últimos pinos, 2 foram usados como saída e 1 como entrada (tri-state = 1), resultando em máscara terminando em  $1001_2 = 9_{16}$ . A saída foi colocada como 0 por padrão:

## Projeto Fase 3



A partir da referência do componente, extraímos o espaço de registradores do mesmo:

Table 2-4: Registers

Address Space Offset <sup>(3)</sup>	Register Name	Access Type	Default Value	Description
0x0000	GPIO_DATA	R/W	0x0	Channel 1 AXI GPIO Data Register.
0x0004	GPIO_TRI	R/W	0x0	Channel 1 AXI GPIO 3-state Control Register.
0x0008	GPIO2_DATA	R/W	0x0	Channel 2 AXI GPIO Data Register.
0x000C	GPIO2_TRI	R/W	0x0	Channel 2 AXI GPIO 3-state Control.
0x011C	GIER <sup>(1)</sup>	R/W	0x0	Global Interrupt Enable Register.
0x0128	IP IER <sup>(1)</sup>	R/W	0x0	IP Interrupt Enable Register (IP IER).
0x0120	IP ISR <sup>(1)</sup>	R/TOW <sup>(2)</sup>	0x0	IP Interrupt Status Register.

**Notes:**

1. Interrupt registers are available only if AXI GPIO is compiled using the **Enable Interrupt** parameter.
2. Toggle-On-Write (TOW) access toggles the status of the bit when a value of 1 is written to the corresponding bit.
3. Address Space Offset is relative to C\_BASEADDR assignment.

Lembrando que, na configuração em single-channel, o segundo barramento GPIO está desativado, e não ativamos interrupções, portanto apenas os registradores GPIO\_DATA e GPIO\_TRI nos interessam:

Table 2-5: AXI Parameter-Register Dependency

Parameters		Register Retainability				GIER, IP IER, IP ISR
		GPIO_DATA	GPIO_TRI	GPIO2_DATA	GPIO2_TRI	
<b>Enable Dual Channel</b>	0	Yes	Yes	No	No	NA
	1	Yes	Yes	Yes	Yes	NA
<b>Enable Interrupt</b>	0	NA				No
	1	NA				Yes

Seguindo o manual, escritas em GPIO\_DATA mudam o valor de pinos em modo saída e leituras retornam o valor atual de pinos em modo entrada. GPIO\_TRI configura o pino como saída (se 0) ou entrada (se 1).

Para concluir a integração com a interface de GPIO, escrevemos código para o softcore realizar acessos:

<pre> axi_gpio.h  #include &lt;stdint.h&gt;  volatile uint32_t *axi_gpio_base = (uint32_t *)0x40000000; #define GPIO_DATA    0 #define GPIO_TRI     1  static inline char axi_gpio_get_done_bit() { </pre>
--

```
// Read GPIO done bit
return axi_gpio_base[GPIO_DATA] & 1;
}

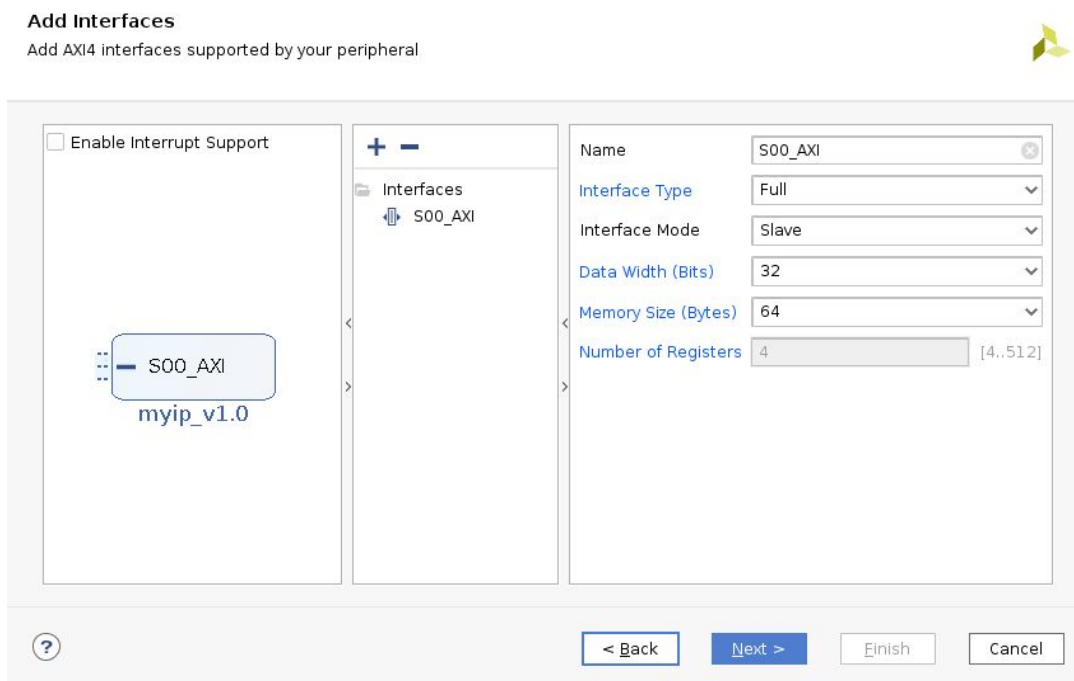
static inline void axi_gpio_set_clear_bit(char value) {
    value = !!value; // constrain to 0 or 1
    axi_gpio_base[GPIO_DATA] |= (value << 1);
}

static inline void axi_gpio_set_start_bit(char value) {
    value = !!value; // constrain to 0 or 1
    axi_gpio_base[GPIO_DATA] |= (value << 2);
}
```

Mesmo após descobrirmos o uso de *slice* e *concat* e seguindo referências da Xilinx online, não conseguimos que o módulo de GPIO funcionasse. Nossa ideia inicial, de utilizar um módulo de GPIO para sinais de controle e DMA para transferência dos dados brutos, foi descartada, e passamos a utilizar apenas DMA, inclusive para sinais de controle.

Em seguida, precisamos que nosso IP suportasse transferências pelo barramento AXI por DMA para receber e enviar os dados usados no algoritmo. Usamos o barramento diretamente com nosso IP por recomendação dos professores e para melhor performance. Cada um vetores de dados (entrada, saída, chave e nonce) possui 128 bits = 16 bytes, para um total de 64 bytes que deve ser mapeados.

Começando utilizando o wizard de criação de IP conectado a AXI4. Escolhemos um dispositivo com interface full, suportando transferências burst, DMA e alta vazão, em modo slave e com 64 bytes mapeados:

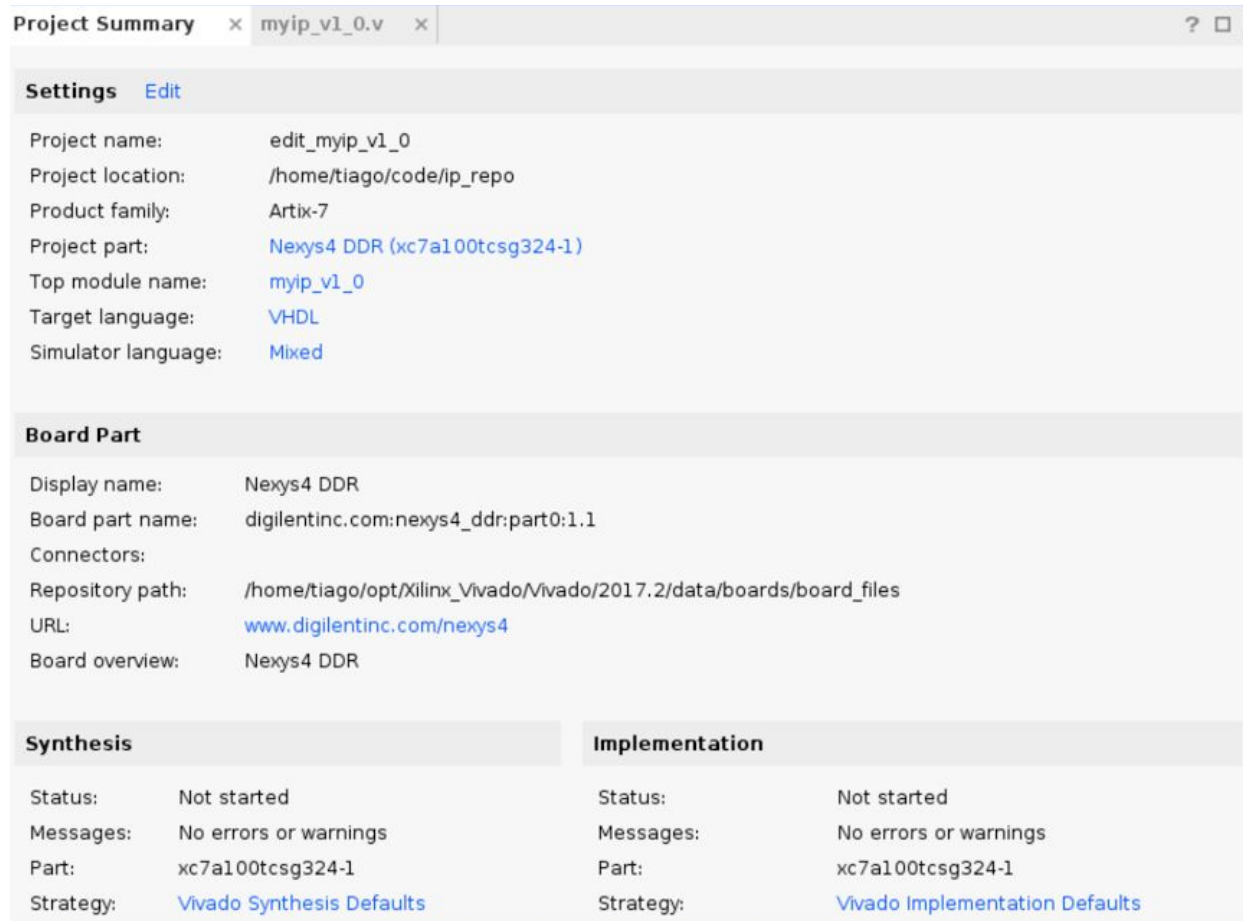




## Projeto Fase 3

No projeto criado para o novo IP, selecionamos “Add Sources” e adicionamos as fontes do nosso IP.

Começamos a editar o IP gerado com tratamento do barramento AXI para ligá-lo aos sources adicionados para AES. Por padrão, o código foi gerado em Verilog, e tivemos que editar as propriedades do projeto top-level para usar VHDL:



**Project Summary** x myip\_v1\_0.v x ? □

**Settings** Edit

Project name: edit\_myip\_v1\_0  
Project location: /home/tiago/code/ip\_repo  
Product family: Artix-7  
Project part: Nexys4 DDR (xc7a100tcsg324-1)  
Top module name: myip\_v1\_0  
Target language: VHDL  
Simulator language: Mixed

**Board Part**

Display name: Nexys4 DDR  
Board part name: digilentinc.com:nexys4\_ddr:part0:1.1  
Connectors:  
Repository path: /home/tiago/opt/Xilinx\_Vivado/Vivado/2017.2/data/boards/board\_files  
URL: [www.digilentinc.com/nexys4](http://www.digilentinc.com/nexys4)  
Board overview: Nexys4 DDR

Synthesis		Implementation	
Status:	Not started	Status:	Not started
Messages:	No errors or warnings	Messages:	No errors or warnings
Part:	xc7a100tcsg324-1	Part:	xc7a100tcsg324-1
Strategy:	<a href="#">Vivado Synthesis Defaults</a>	Strategy:	<a href="#">Vivado Implementation Defaults</a>

Ao editamos o IP gerado automaticamente para instanciar o nosso IP e conectar aos vetores gerados com dados do barramento AXI, percebemos que o componente AXI Full gerado não possui memória interna, apenas repassando os sinais e dados de leitura e escrita para nosso IP. Queríamos utilizar o Full que parece mais adequado ao projeto, já que suporta transferência em burst mode de dados sequenciais, mas como desejamos um jeito fácil de receber os valores por DMA, refizemos o IP com AXI Lite. O IP gerado com AXI Lite salva os valores em registradores internamente, e precisamos apenas conectá-los ao nosso IP. Além disso, usamos 65 registradores de 32 bits, mantendo um para ser usado para sinais de controle:

## Projeto Fase 3

### Peripheral Details

Specify name, version and description for the new peripheral



Name:	<input type="text" value="axi_interface_lite"/>
Version:	<input type="text" value="1.0"/>
Display name:	<input type="text" value="axi_interface_lite_v1.0"/>
Description:	<input type="text" value="65 32-bit register interface"/>
IP location:	<input type="text" value="/home/tiago/code/ip_repo"/>
<input type="checkbox"/> Overwrite existing	

? < Back Next > Finish Cancel

### Add Interfaces

Add AXI4 interfaces supported by your peripheral



☐ Enable Interrupt Support

S00\_AXI

axi\_interface\_lite\_v1.0

+ -

Interfaces

S00\_AXI

Name

S00\_AXI

Interface Type

Lite

Interface Mode

Slave

Data Width (Bits)

32

Memory Size (Bytes)

64

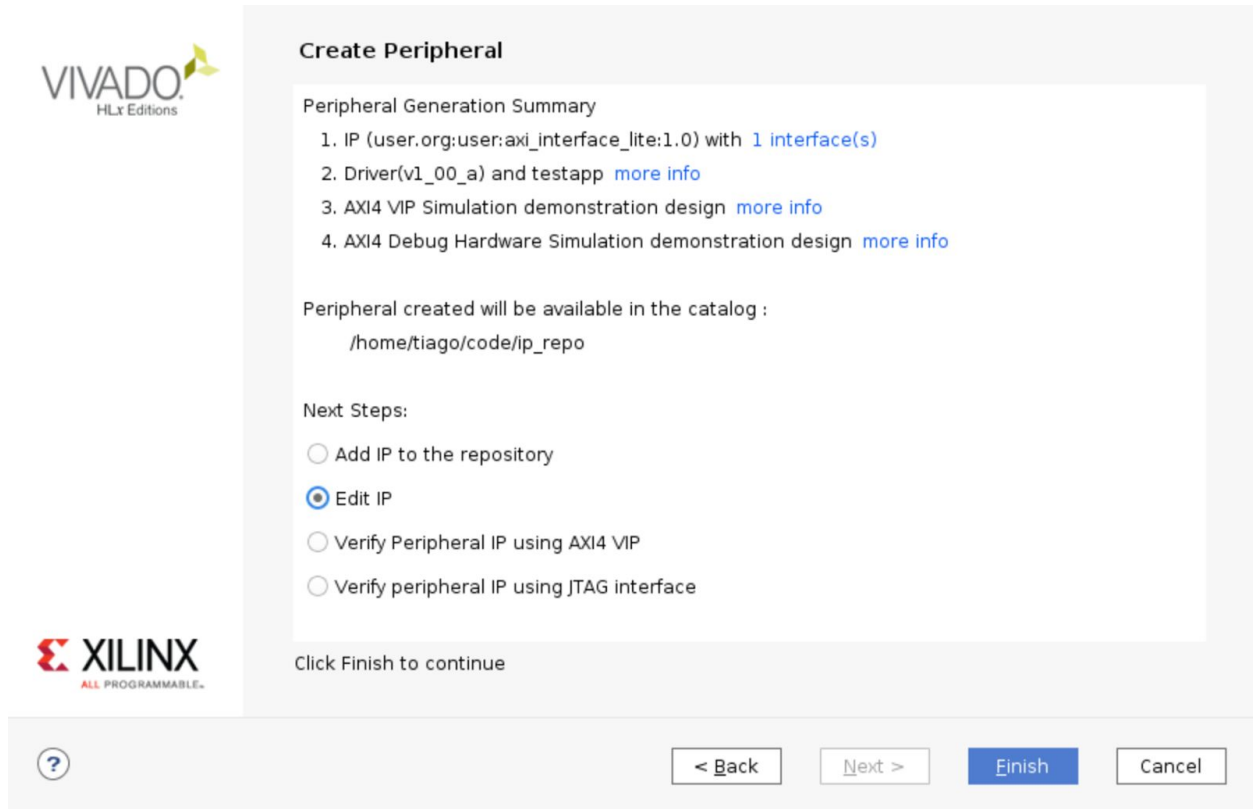
Number of Registers

65

[4..512]

? < Back Next > Finish Cancel

## Projeto Fase 3



Na instanciação de `vhdl_axi_lite_65_v1_0_S00_AXI`, adicionamos um componente `AES_CTR` e sinais necessários:

```
signal pt, nonce_counter, key, ct : std_logic_vector(127 downto 0);

component AES_CTR is
port (
    clock, start, clear: in STD_LOGIC;
    pt, nonce_counter, key: in STD_LOGIC_VECTOR(127 downto 0);
    ct: out STD_LOGIC_VECTOR(127 downto 0);
    done: out STD_LOGIC
);
end component AES_CTR;
```

E na implementação, conectamos os sinais:

```
pt <= slv_reg3 & slv_reg2 & slv_reg1 & slv_reg0;
nonce_counter <= slv_reg7 & slv_reg6 & slv_reg5 & slv_reg4;
key <= slv_reg11 & slv_reg10 & slv_reg9 & slv_reg8;
ct <= slv_reg15 & slv_reg14 & slv_reg13 & slv_reg12;

AES_CTR_impl : AES_CTR
port map (
    clock => S_AXI_ACLK,
    clear => slv_reg16(0),
    start => slv_reg16(1),
    done => slv_reg16(2),
    pt => pt,
    nonce_counter => nonce_counter,
    key => key,
    ct => ct
);
```

Depois disso, abrimos a definição da memória gerada e corrigimos erros de duplicate driver, que surgiram pois a memória gerada pelo wizard do Vivado tenta escrever nos endereços que são saídas do nosso circuito. Retiramos o suporte a escrita das palavras usadas pelo bloco de saída do AES e sinais de controle. O seguinte patch indica todas as mudanças feitas:

```
diff --git a/include/vhdl/AXI_ip/vhdl_axi_aes_v1_0_AXI_AES.vhd
b/include/vhdl/AXI_ip/vhdl_axi_aes_v1_0_AXI_AES.vhd
index 268e796..b61ff7e 100644
--- a/include/vhdl/AXI_ip/vhdl_axi_aes_v1_0_AXI_AES.vhd
+++ b/include/vhdl/AXI_ip/vhdl_axi_aes_v1_0_AXI_AES.vhd
@@ -298,11 +298,11 @@ begin
    slv_reg9 <= (others => '0');
    slv_reg10 <= (others => '0');
    slv_reg11 <= (others => '0');
-    slv_reg12 <= (others => '0');
-    slv_reg13 <= (others => '0');
-    slv_reg14 <= (others => '0');
-    slv_reg15 <= (others => '0');
-    slv_reg16 <= (others => '0');
+    -- slv_reg12 <= (others => '0');
+    -- slv_reg13 <= (others => '0');
+    -- slv_reg14 <= (others => '0');
+    -- slv_reg15 <= (others => '0');
+    -- slv_reg16 <= (others => '0');
    slv_reg17 <= (others => '0');
    slv_reg18 <= (others => '0');
    slv_reg19 <= (others => '0');
@@ -454,41 +454,36 @@ begin
    when b"0001100" =>
        for byte_index in 0 to (C_S_AXI_DATA_WIDTH/8-1) loop
            if ( S_AXI_WSTRB(byte_index) = '1' ) then
-                -- Respective byte enables are asserted as per write strobes
-                -- slave register 12
-                slv_reg12(byte_index*8+7 downto byte_index*8) <=
```

```

S_AXI_WDATA(byte_index*8+7 downto byte_index*8);
+           -- Read only register
+           -- slv_reg12(byte_index*8+7 downto byte_index*8) <=
S_AXI_WDATA(byte_index*8+7 downto byte_index*8);
        end if;
    end loop;
    when b"0001101" =>
        for byte_index in 0 to (C_S_AXI_DATA_WIDTH/8-1) loop
            if ( S_AXI_WSTRB(byte_index) = '1' ) then
                -- Respective byte enables are asserted as per write strobes
                -- slave register 13
                slv_reg13(byte_index*8+7 downto byte_index*8) <=
S_AXI_WDATA(byte_index*8+7 downto byte_index*8);
+           -- Read only register
+           -- slv_reg13(byte_index*8+7 downto byte_index*8) <=
S_AXI_WDATA(byte_index*8+7 downto byte_index*8);
            end if;
        end loop;
    when b"0001110" =>
        for byte_index in 0 to (C_S_AXI_DATA_WIDTH/8-1) loop
            if ( S_AXI_WSTRB(byte_index) = '1' ) then
                -- Respective byte enables are asserted as per write strobes
                -- slave register 14
                slv_reg14(byte_index*8+7 downto byte_index*8) <=
S_AXI_WDATA(byte_index*8+7 downto byte_index*8);
+           -- Read only register
+           -- slv_reg14(byte_index*8+7 downto byte_index*8) <=
S_AXI_WDATA(byte_index*8+7 downto byte_index*8);
            end if;
        end loop;
    when b"0001111" =>
        for byte_index in 0 to (C_S_AXI_DATA_WIDTH/8-1) loop
            if ( S_AXI_WSTRB(byte_index) = '1' ) then
                -- Respective byte enables are asserted as per write strobes
                -- slave register 15
                slv_reg15(byte_index*8+7 downto byte_index*8) <=
S_AXI_WDATA(byte_index*8+7 downto byte_index*8);
+           -- Read only register
+           -- slv_reg15(byte_index*8+7 downto byte_index*8) <=
S_AXI_WDATA(byte_index*8+7 downto byte_index*8);
            end if;
        end loop;
    when b"0010000" =>
        for byte_index in 0 to (C_S_AXI_DATA_WIDTH/8-1) loop
            if ( S_AXI_WSTRB(byte_index) = '1' ) then
                -- Respective byte enables are asserted as per write strobes
                -- slave register 16
                slv_reg16(byte_index*8+7 downto byte_index*8) <=
S_AXI_WDATA(byte_index*8+7 downto byte_index*8);
+           -- Read only register
+           -- slv_reg16(byte_index*8+7 downto byte_index*8) <=
S_AXI_WDATA(byte_index*8+7 downto byte_index*8);
            end if;
        end loop;
    when b"0010001" =>
@@ -888,11 +883,11 @@ begin
        slv_reg9 <= slv_reg9;
        slv_reg10 <= slv_reg10;
        slv_reg11 <= slv_reg11;
        -
        slv_reg12 <= slv_reg12;

```

## Projeto Fase 3

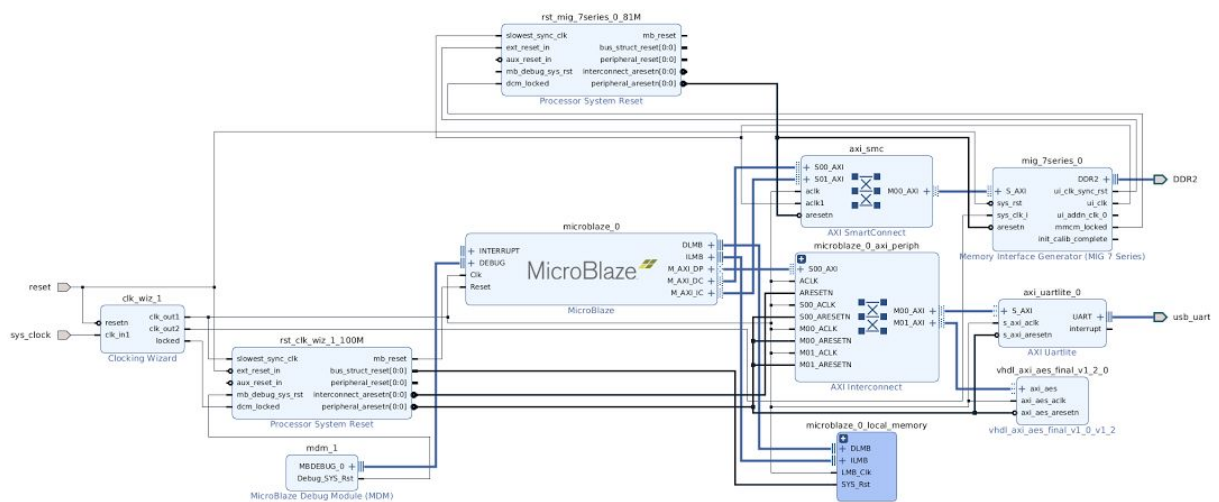
```

-      slv_reg13 <= slv_reg13;
-      slv_reg14 <= slv_reg14;
-      slv_reg15 <= slv_reg15;
-      slv_reg16 <= slv_reg16;
+      -- slv_reg12 <= slv_reg12;
+      -- slv_reg13 <= slv_reg13;
+      -- slv_reg14 <= slv_reg14;
+      -- slv_reg15 <= slv_reg15;
+      -- slv_reg16 <= slv_reg16;
      slv_reg17 <= slv_reg17;
      slv_reg18 <= slv_reg18;
      slv_reg19 <= slv_reg19;

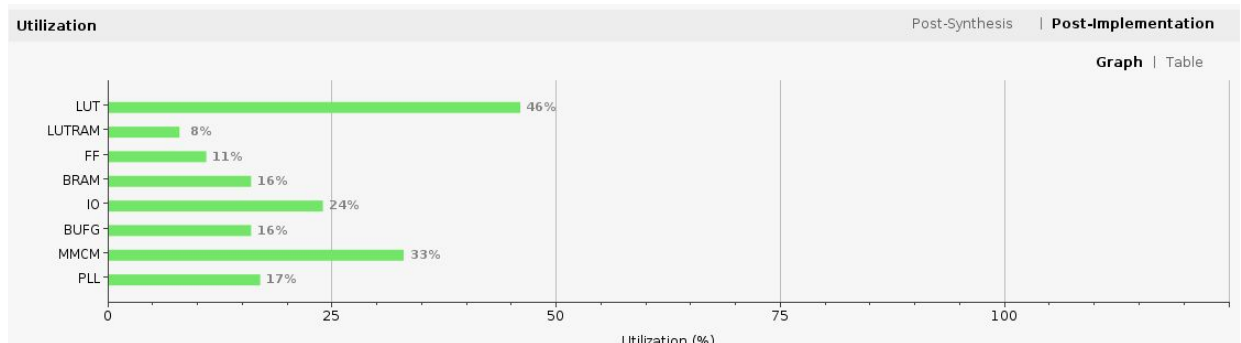
```

Depois de mais alterações para ajustar o IP à interface de memória, o projeto sintetizou e foi implementado com sucesso.

Importamos o projeto do IP com AES conectado por AXI no projeto principal. Conectamos o IP no barramento AXI:



Refizemos a síntese e implementação do softcore. Com o módulo de AES com AXI é possível notar que sobrou ainda muito espaço na FPGA:



Implementamos o código para o softcore:

axi\_dma.h

```

/*
 * axi_dma.h
 *
 * Created on: Aug 8, 2017
 * Author: tiago
 */

#ifndef SRC_AXI_DMA_H_
#define SRC_AXI_DMA_H_

#include <stdint.h>

volatile uint32_t *axi_dma_base = (uint32_t *)0x44a00000;

static inline char axi_get_done_bit() {
    // Read done bit
    return axi_dma_base[16];
}

static inline void axi_set_clear_bit() {
    axi_dma_base[17] = 1;
}

static inline void axi_reset_status_word() {
    axi_dma_base[17] = 0;
}

static inline void axi_set_start_bit() {
    axi_dma_base[17] |= 1 << 1;
}

static inline void axi_set_pt(uint16_t n, uint32_t v) {
    axi_dma_base[3 - n] = v;
}

static inline void axi_set_nonce(uint16_t n, uint32_t v) {
    axi_dma_base[4 + 3 - n] = v;
}

static inline void axi_set_key(uint16_t n, uint32_t v) {
    axi_dma_base[8 + 3 - n] = v;
}

static inline uint32_t axi_get_ct(uint16_t n) {
    return axi_dma_base[12 + 3 - n];
}

#endif /* SRC_AXI_DMA_H_ */

```

## 2.6 Execução do algoritmo de referência no projeto sintetizado

Utilizamos o seguinte código para teste, entrando com vetores encontrados online e comparando as saídas:

main.c

```
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"

#include "axi_dma.h"

int main()
{
    init_platform();

    axi_set_nonce(0, 0x01020304);
    axi_set_nonce(1, 0x05060708);
    axi_set_nonce(2, 0x090a0b0c);
    axi_set_nonce(3, 0x0d0e0f10);

    axi_set_pt(0, 0);
    axi_set_pt(1, 0);
    axi_set_pt(2, 0);
    axi_set_pt(3, 0);

    axi_set_key(0, 0x11121314);
    axi_set_key(1, 0x15161718);
    axi_set_key(2, 0x191a1b1c);
    axi_set_key(3, 0x1d1e1f20);

    axi_set_clear_bit();
    axi_reset_status_word();
    axi_set_start_bit();
    while (!axi_get_done_bit()) ;

    for (int i = 0; i < 4; i++)
        printf("%lx ", axi_get_ct(i));

    cleanup_platform();
    return 0;
}
```



Após arrumar pequenas falhas, rodamos também os casos de teste do Golden e medimos seu desempenho. Como o tempo de execução é muito menor, decidimos executar o AES 10000 vezes e realizar um benchmark.

### 2.7 Identificar métricas de desempenho, comparar implementações em hardware.

Comparando os resultados, percebemos que foi obtido um ganho significativo na implementação em hardware. O tempo de execução da bateria de testes totalmente em software foi de 27.809 segundos, já com o coprocessador sintetizado foi de 14872.20 microssegundos. Isto representa um ganho de cerca de  $2.664 \times 10^6$  vezes.

Obtivemos o seguinte resultado:

Implementação	Tempo	Tempo médio
Software	27.809 s (7 execuções)	3.97 s
VHDL	14872.20 us (10000 execuções)	1.49 us
		Relação: 2664430 s/s

## 3. Conclusão

Nesta etapa do projeto foi possível realizar uma implementação com sucesso do coprocessador em VHDL e verificar seu correto funcionamento. Também foi possível sintetizar com sucesso um modelo do Microblaze na placa Nexys 4 DDR, integrar um módulo do coprocessador ao mesmo e executar o algoritmo de referência.

Foi possível realizar a comparação de desempenho entre o algoritmo de referência totalmente em software com um módulo do coprocessador sintetizado, chegando-se a conclusão que de fato houve um ganho significativo, com o desempenho esperado para um hardware especializado. O Microblaze, por ser um softcore de baixo poder computacional, é muito mais lento que a implementação puramente em FPGA.

Vale mencionar que durante o desenvolvimento do trabalho, não vimos grandes vantagens na utilização do SystemC, uma vez que para protótipos iniciais em hardware o Vivado HLS parece capaz de sintetizar versões HDL de código em C/C++. Além disso, a linguagem contém documentação pouco clara, uma vez que é relativamente pouco utilizada. Acreditamos que o uso dessa linguagem facilita parcialmente a reformulação do golden como hardware, servindo meramente como passo intermediário, o que auxiliou na criação do VHDL – porém, julgamos que a passagem rápida por SystemC nesse trabalho não foi suficiente para justificar o uso da linguagem.

O Vivado mostrou-se um ambiente de desenvolvimento extremamente completo, mas também complexo e com uma considerável curva de aprendizado, uma vez que nunca havíamos utilizado o mesmo ou realizado o desenvolvimento de um projeto deste nível.

### 4. Bibliografia

- ARMbed. **AES source code**. Disponível em: <https://tls.mbed.org/aes-source-code>. Acesso em: 13/06/2017.
- National Institute of Standards and Technology. **Recommendation for Block Cipher Modes of Operation**. Disponível em: <http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf>. Acesso em: 14/06/2017.
- Accellera Standards. **SystemC**. Disponível em: <http://accellera.org/downloads/standards/systemc>. Acesso em: 27/06/2017.
- Doulos. **SystemC Tutorial**. Disponível em: <https://www.doulos.com/knowhow/systemc/tutorial/>. Acesso em: 28/06/2017.
- Xilinx. **MicroBlaze Soft Processor Core**. Disponível em: <https://www.xilinx.com/products/design-tools/microblaze.html>. Acesso em: 03/07/2017.
- Xilinx. **Creating a Simple MicroBlaze Design in IP Integrator**. Disponível em: <https://www.xilinx.com/video/hardware/creating-a-simple-microblaze-design-in-ip-integrator.html>. Acesso em: 03/07/2017.
- Xilinx. **Using Vivado HLS SW Libraries in your C, C++, System-C Code**. Disponível em: <https://www.xilinx.com/video/hardware/vivado-hls-sw-libraries-in-your-c-system-c-code.html>. Acesso em: 03/07/2017.
- Digilent. **Simulating a Custom IP core using a Zynq processor**. Disponível em: <https://reference.digilentinc.com/learn/programmable-logic/tutorials/zybo-custom-ip-simulation/start>. Acesso em: 03/07/2017.
- Digilent. **Nexys 4 DDR - Getting Started with Microblaze**. Disponível em: <https://reference.digilentinc.com/learn/programmable-logic/tutorials/nexys-4-ddr-getting-started-with-microblaze/start>. Acesso em: 03/07/2017.
- CHAPMAN, Ken. **Implementing a Simple PicoBlaze Design in Vivado**. Disponível em: [http://www-classes.usc.edu/engr/ee-s/254/ee254l\\_lab\\_manual/PicoBlaze/Picoblaze\\_KCPSM6\\_Release9\\_30Sept14/PicoBlaze\\_Design\\_in\\_Vivado.pdf](http://www-classes.usc.edu/engr/ee-s/254/ee254l_lab_manual/PicoBlaze/Picoblaze_KCPSM6_Release9_30Sept14/PicoBlaze_Design_in_Vivado.pdf). Acesso em: 06/07/2017.
- Xilinx. **Getting Started with Vivado High-Level Synthesis**. Disponível em: <https://www.xilinx.com/video/hardware/getting-started-vivado-high-level-synthesis.html>. Acesso em: 06/07/2017.
- HU, Yu Hen. **SystemC and System Level Design Resources**. Disponível em: <http://homepages.cae.wisc.edu/~ece734/SystemC/>. Acesso em: 08/07/2017.
- Archlinux User Repository. **Systemc**. Disponível em: <https://aur.archlinux.org/packages/systemc/>. Acesso em: 09/07/2017.
- Xilinx. **AXI GPIO v2.0 - LogiCORE IP Product Guide** (Vivado Design Suite PG144 - October 5, 2016). Disponível em: [https://www.xilinx.com/support/documentation/ip\\_documentation/axi\\_gpio/v2\\_0/pg144-axi-gpio.pdf](https://www.xilinx.com/support/documentation/ip_documentation/axi_gpio/v2_0/pg144-axi-gpio.pdf)

### Projeto Fase 3

- FPGA Developer. **Creating a custom IP block in Vivado**. Disponível em: <http://www.fpgadeveloper.com/2014/08/creating-a-custom-ip-block-in-vivado.html>. Acesso em: 30/07/2017.

As imagens utilizadas estão em domínio público.