Учреждение образования

«Белорусский государственный университет

информатики и радиоэлектроники»

Кафедра ПОИТ

Отчет

по ра:

«Методы защиты проектных HDL-описания от несанкционированного использования. Применение водяных знаков и отпечатков пальцев (watermarking & fingerprint)»

Выполнил: Верховцов Павел Андреевич

магистрант кафедры ПОИТ

группа № 7М2221

Проверил: Иванюк Александр Александрович

доктор технических наук

Минск 2018

**Stack**

1. **Описание ISA**

|  |  |  |
| --- | --- | --- |
| Мнемоника | Код операции (3 бита) | Операнд (5 бит) |
| ADD | 000 | xxxxx |
| SUBT | 001 | xxxxx |
| SHIFT | 010 | xxxxx |
| JNZ | 011 | aaaaa |
| PUSH | 100 | aaaaa |
| POP | 101 | aaaaa |
| POPIN | 110 | aaaaa |
| HALT | 111 | xxxxx |

ADD – сложение значений TOS и TOS+1, результат записывается на вершину стека

SUBT – вычитание значений TOS+1 из TOS, результат записывается на вершину стека

SHIFT – логический сдвиг вправо значения TOS, результат записывается на вершину стека

JNZ addr – переход на операции по адресу addr, при сброшенном флаге нуля

PUSH addr – пересылка значения хранящегося по адресу addr на вершину стека TOS

POP addr – пересылка значения с вершины стека по адресу addr

POPIN addr – пересылка значения с вершины стека по адресу хранящемуся по адресу addr.

HALT - останова

1. **Микропрограмма**

constant STUB: mem\_addr := "00000";

constant LENGTH: mem\_addr := "10000";

constant COUNTER: mem\_addr := "10001";

constant CURRENT\_VALUE: mem\_addr := "10010";

constant DATA\_START: mem\_addr := "00000";

constant INIT\_VALUE: mem\_addr := "10011";

constant ZERO: mem\_addr := "10100";

constant ONE: mem\_addr := "10101";

constant EMPTY\_SPACE: mem\_addr := "10110";

constant LOADDR: mem\_addr := "01010";

PUSH & ZERO,

POP & COUNTER,

PUSH & INIT\_VALUE,

POP & CURRENT\_VALUE,

PUSH & INIT\_VALUE,

POP & DATA\_START,

PUSH & ONE,

PUSH & LENGTH,

SUBT & STUB,

POP & LENGTH,

PUSH & COUNTER,

PUSH & ONE,

ADD & STUB,

POP & COUNTER,

PUSH & CURRENT\_VALUE,

SHIFT & STUB,

POP & CURRENT\_VALUE,

PUSH & CURRENT\_VALUE,

POPIN & COUNTER,

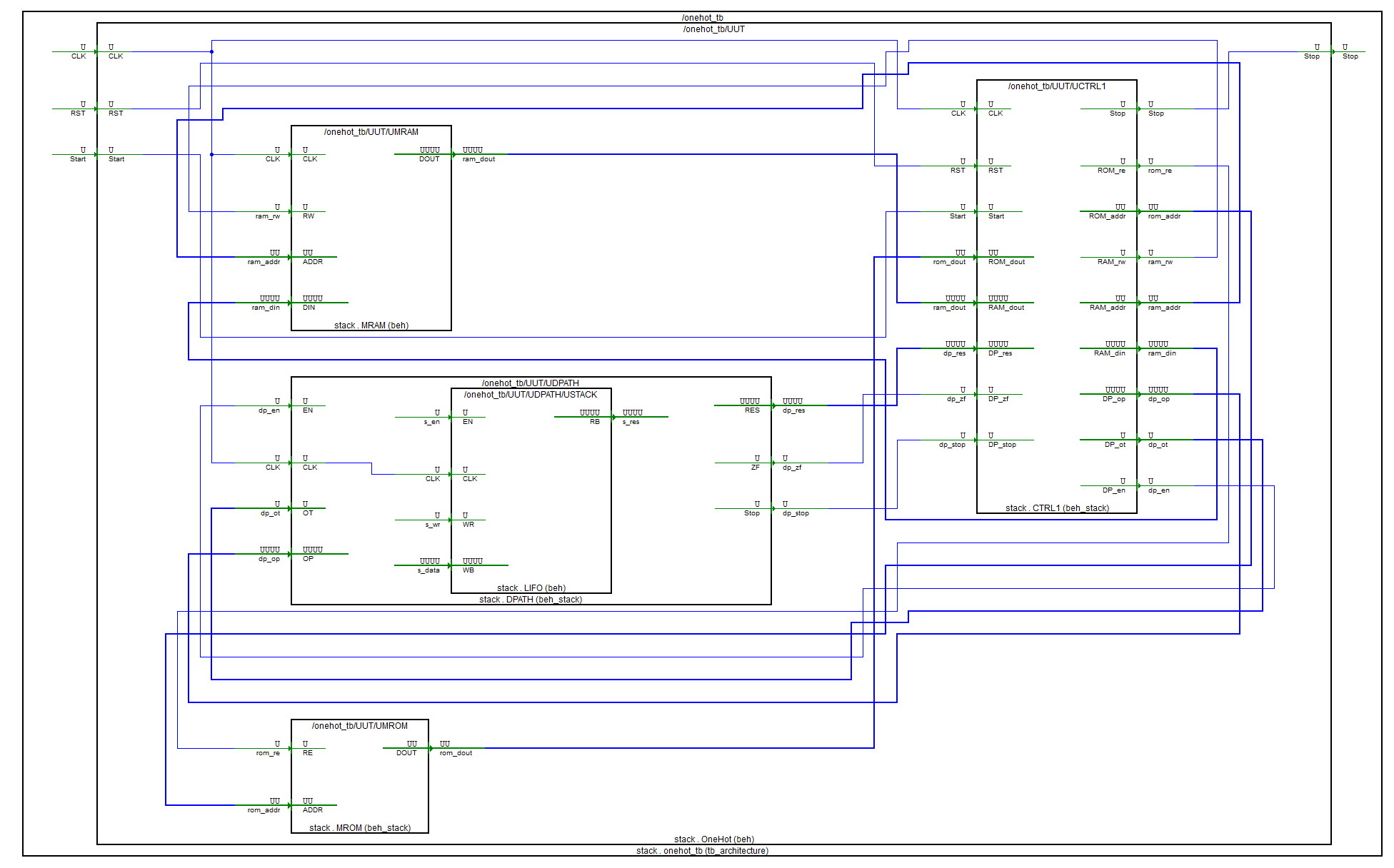
PUSH & LENGTH,

PUSH & COUNTER,

SUBT & STUB,

JNZ & LOADDR

1. **Общая структурная схема**



1. **Описание микроархитектуры**
2. **Block Level**
   1. Блок MROM
      1. **HDL**

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

use IEEE.STD\_LOGIC\_UNSIGNED.all;

use OneHotStack.all;

entity MROM is

port (

RE: in std\_logic;

ADDR: in mem\_addr;

DOUT: out command

);

end MROM;

architecture Beh\_Stack of MROM is

type tROM is array (0 to 31) of command;

constant STUB: mem\_addr := "00000";

constant LENGTH: mem\_addr := "10000";

constant COUNTER: mem\_addr := "10001";

constant CURRENT\_VALUE: mem\_addr := "10010";

constant DATA\_START: mem\_addr := "00000";

constant INIT\_VALUE: mem\_addr := "10011";

constant ZERO: mem\_addr := "10100";

constant ONE: mem\_addr := "10101";

constant EMPTY\_SPACE: mem\_addr := "10110";

constant LOADDR: mem\_addr := "01010";

constant ROM: tROM :=(

-- OP CODE | RAM ADDR

-- Init

PUSH & ZERO,

POP & COUNTER,

PUSH & INIT\_VALUE,

POP & CURRENT\_VALUE,

PUSH & INIT\_VALUE,

POP & DATA\_START,

PUSH & ONE,

PUSH & LENGTH,

SUBT & STUB,

POP & LENGTH,

-- Loop

PUSH & COUNTER,

PUSH & ONE,

ADD & STUB,

POP & COUNTER,

PUSH & CURRENT\_VALUE,

SHIFT & STUB,

POP & CURRENT\_VALUE,

PUSH & CURRENT\_VALUE,

POPIN & COUNTER,

PUSH & LENGTH,

PUSH & COUNTER,

SUBT & STUB,

JNZ & LOADDR,

others => HALT & STUB );

signal data: command;

begin

data <= ROM(conv\_integer(ADDR));

zbufs: process (RE, data)

begin

if (RE = '1') then

DOUT <= data;

else

DOUT <= (others => 'Z');

end if;

end process;

end Beh\_Stack;

* + 1. **TestBench**

library stack;

use stack.OneHotStack.all;

library ieee;

use ieee.STD\_LOGIC\_UNSIGNED.all;

use ieee.std\_logic\_1164.all;

entity mrom\_tb is

end mrom\_tb;

architecture TB\_ARCHITECTURE of mrom\_tb is

component mrom

port(

RE : in STD\_LOGIC;

ADDR : in mem\_addr;

DOUT : out command );

end component;

signal RE : STD\_LOGIC;

signal ADDR : mem\_addr;

signal DOUT : command;

constant WAIT\_period: time := 10 ns;

begin

UUT : mrom

port map (

RE => RE,

ADDR => ADDR,

DOUT => DOUT

);

main: process

begin

re <= '0';

addr <= "00010";

wait for 1 \* WAIT\_period;

re <= '1';

wait for 1 \* WAIT\_period;

addr <= "00000";

re <= '1';

wait for 1 \* WAIT\_period;

re <= '0';

wait for 100 \* WAIT\_period;

wait;

end process;

end TB\_ARCHITECTURE;

configuration TESTBENCH\_FOR\_mrom of mrom\_tb is

for TB\_ARCHITECTURE

for UUT : mrom

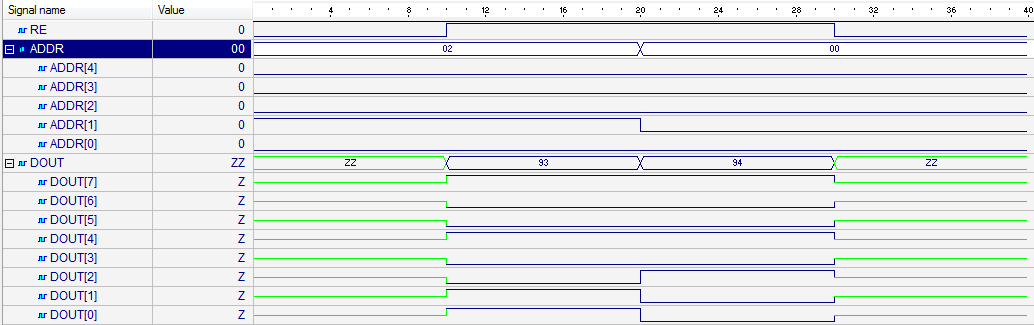
use entity work.mrom(beh\_stack);

end for;

end for;

end TESTBENCH\_FOR\_mrom;

* + 1. **Временные диаграммы функционирования**



* 1. Блок MRAM
     1. **HDL**

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

use IEEE.STD\_LOGIC\_UNSIGNED.all;

use OneHotStack.all;

entity MRAM is

port (

CLK: in std\_logic;

RW: in std\_logic;

ADDR: in mem\_addr;

DIN: in operand;

DOUT: out operand

);

end MRAM;

architecture Beh of MRAM is

type tRAM is array (0 to 31) of operand;

signal RAM: tRAM:= (

-- | BIN | ADR BIN

"0000000000000000", -- | 00000 |

"0000000000000000", -- | 00001 |

"0000000000000000", -- | 00010 |

"0000000000000000", -- | 00011 |

"0000000000000000", -- | 00100 |

"0000000000000000", -- | 00101 |

"0000000000000000", -- | 00110 |

"0000000000000000", -- | 00111 |

"0000000000000000", -- | 01000 |

"0000000000000000", -- | 01001 |

"0000000000000000", -- | 01010 |

"0000000000000000", -- | 01011 |

"0000000000000000", -- | 01100 |

"0000000000000000", -- | 01101 |

"0000000000000000", -- | 01110 |

"0000000000000000", -- | 01111 |

"0000000000001000", -- | 10000 |

"0000000000000000", -- | 10001 |

"0000000000000000", -- | 10010 |

"0000000000000001", -- | 10011 |

"0000000000000000", -- | 10100 |

"0000000000000001", -- | 10101 |

others => "0000000000000000"

);

signal data\_in: operand;

signal data\_out: operand;

Begin

data\_in <= Din;

WRITE: process (CLK, RW, ADDR, data\_in)

begin

if (RW = '0') then

if (rising\_edge(CLK)) then

RAM(conv\_integer(ADDR)) <= data\_in;

end if;

end if;

end process;

data\_out <= RAM (conv\_integer(ADDR));

RDP: process (RW, RAM, data\_out)

begin

if (RW = '1') then

DOUT <= data\_out;

else

DOUT <= (others => 'Z');

end if;

end process;

end Beh;

* + 1. **TestBench**

library stack;

use stack.OneHotStack.all;

library ieee;

use ieee.STD\_LOGIC\_UNSIGNED.all;

use ieee.std\_logic\_1164.all;

entity mram\_tb is

end mram\_tb;

architecture TB\_ARCHITECTURE of mram\_tb is

component mram

port(

CLK : in STD\_LOGIC;

RW : in STD\_LOGIC;

ADDR : in mem\_addr;

DIN : in operand;

DOUT : out operand );

end component;

signal CLK : STD\_LOGIC;

signal RW : STD\_LOGIC;

signal ADDR : mem\_addr;

signal DIN : operand;

signal DOUT : operand;

constant CLK\_period: time := 10 ns;

begin

UUT : mram

port map (

CLK => CLK,

RW => RW,

ADDR => ADDR,

DIN => DIN,

DOUT => DOUT

);

CLK\_Process: process

begin

CLK <= '0';

wait for CLK\_Period/2;

CLK <= '1';

wait for CLK\_Period/2;

end process;

main: process

begin

wait for CLK\_PERIOD;

addr <= "00010";

din <= "0000000000000100";

wait for CLK\_PERIOD;

addr <= "00001";

rw <= '1';

wait for CLK\_PERIOD;

addr <= "00000";

din <= "0000000000010000";

rw <= '0';

wait for CLK\_PERIOD;

wait;

end process;

end TB\_ARCHITECTURE;

configuration TESTBENCH\_FOR\_mram of mram\_tb is

for TB\_ARCHITECTURE

for UUT : mram

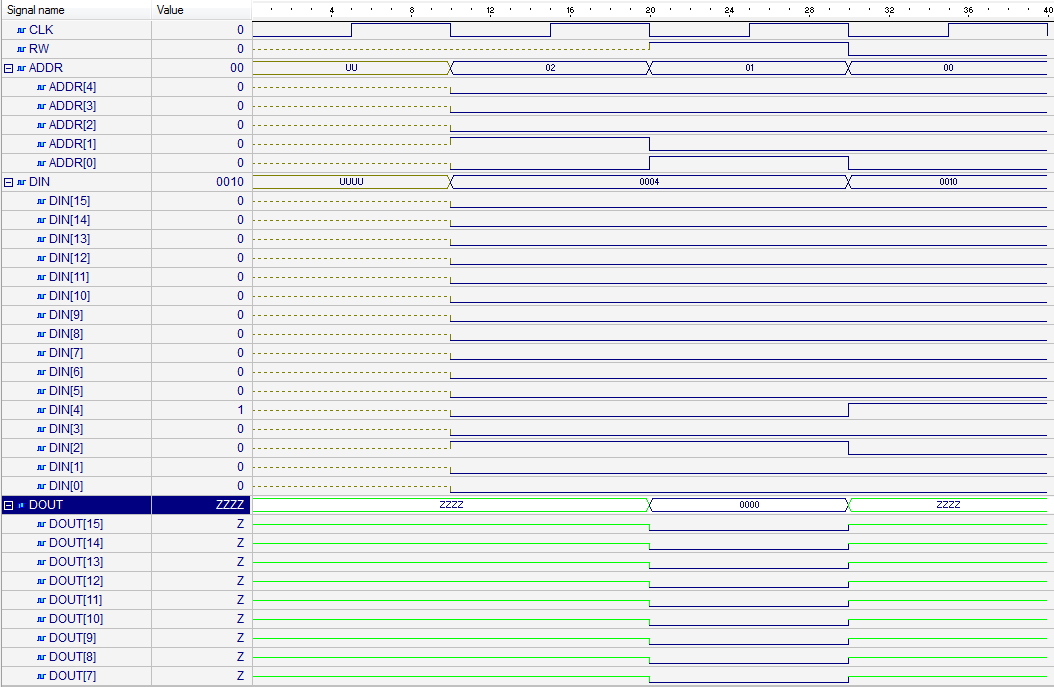
use entity work.mram(beh);

end for;

end for;

end TESTBENCH\_FOR\_mram;

* + 1. **Временные диаграммы функционирования**



* 1. Блок CTRL
     1. **HDL**

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

use IEEE.STD\_LOGIC\_UNSIGNED.all;

library stack;

use stack.OneHotStack.all;

entity CTRL1 is

port(

CLK, RST, Start: in std\_logic;

Stop: out std\_logic;

ROM\_re: out std\_logic;

ROM\_addr: out mem\_addr;

ROM\_dout: in command;

RAM\_rw: out std\_logic;

RAM\_addr: out mem\_addr;

RAM\_din: out operand;

RAM\_dout: in operand;

DP\_op: out operand;

DP\_ot: out operation;

DP\_en: out std\_logic;

DP\_res: in operand;

DP\_zf: in std\_logic;

DP\_stop: in std\_logic

);

end CTRL1;

architecture Beh\_Stack of CTRL1 is

type states is (I, F, D, R, S, H, LIN, J, DP, DPW);

signal nxt\_state, cur\_state: states;

signal RI: command;

signal IC: mem\_addr;

signal RO: operation;

signal RA: mem\_addr;

signal RD: operand;

begin

FSM: process(CLK, RST, nxt\_state)

begin

if (RST = '1') then

cur\_state <= I;

elsif rising\_edge(CLK) then

cur\_state <= nxt\_state;

end if;

end process;

COMB: process(cur\_state, start, RO, DP\_stop)

begin

case cur\_state is

when I =>

if (start = '1') then

nxt\_state <= F;

else

nxt\_state <= I;

end if;

when F => nxt\_state <= D;

when D =>

case RO is

when ADD | SUBT | SHIFT | POP => nxt\_state <= DP;

when HALT => nxt\_state <= H;

when JNZ => nxt\_state <= J;

when others => nxt\_state <= R;

end case;

when R =>

if (RO = PUSH) then

nxt\_state <= DP;

elsif (RO = POPIN) then

nxt\_state <= LIN;

else

nxt\_state <= I;

end if;

when LIN => nxt\_state <= DP;

when DP => nxt\_state <= DPW;

when DPW =>

if (DP\_stop = '0') then

nxt\_state <= DPW;

else

if (RO = POP or RO = POPIN) then

nxt\_state <= S;

else

nxt\_state <= F;

end if;

end if;

when S | J => nxt\_state <= F;

when H => nxt\_state <= H;

when others => nxt\_state <= I;

end case;

end process;

PSTOP: process (cur\_state)

begin

if (cur\_state = H) then

stop <= '1';

else

stop <= '0';

end if;

end process;

PMC: process (CLK, RST, cur\_state)

begin

if (RST = '1') then

IC <= "00000";

elsif falling\_edge(CLK) then

if (cur\_state = D) then

IC <= IC + 1;

elsif (cur\_state = J and DP\_ZF = '0') then

IC <= RA;

end if;

end if;

end process;

ROM\_addr <= IC;

PROMREAD: process (nxt\_state, cur\_state)

begin

if (nxt\_state = F or cur\_state = F) then

ROM\_re <= '1';

else

ROM\_re <= '0';

end if;

end process;

PROMDAT: process (RST, cur\_state, ROM\_dout)

begin

if (RST = '1') then

RI <= (others => '0');

elsif (cur\_state = F) then

RI <= ROM\_dout;

end if;

end process;

-- RO and RA control

PRORA: process (RST, nxt\_state, RI)

begin

if (RST = '1') then

RO <= (others => '0');

RA <= (others => '0');

elsif (nxt\_state = D) then

RO <= RI (7 downto 5);

RA <= RI (4 downto 0);

elsif (nxt\_state = LIN) then

RA <= RD (4 downto 0);

end if;

end process;

PRAMST: process (RA)

begin

if (cur\_state /= J) then

RAM\_addr <= RA;

end if;

end process;

PRAMREAD: process (cur\_state)

begin

if (cur\_state = S) then

RAM\_rw <= '0';

else

RAM\_rw <= '1';

end if;

end process;

PRAMDAR: process (cur\_state)

begin

if (cur\_state = R) then

RD <= RAM\_dout;

end if;

end process;

-- move the value from DPATH to RAM input bus

RAM\_din <= DP\_res;

-- move the value from RD to datapath

DP\_op <= RD;

-- move RO value to DP operation bus

DP\_ot <= RO;

pdpathen: process (cur\_state)

begin

if (cur\_state = DP) then

DP\_en <= '1';

else

DP\_en <= '0';

end if;

end process;

end Beh\_Stack;

* + 1. **TestBench**

library stack;

use stack.OneHotStack.all;

library ieee;

use ieee.STD\_LOGIC\_UNSIGNED.all;

use ieee.std\_logic\_1164.all;

entity ctrl1\_tb is

end ctrl1\_tb;

architecture TB\_ARCHITECTURE of ctrl1\_tb is

component MROM is

port (

RE: in std\_logic;

ADDR: in mem\_addr;

DOUT: out command

);

end component;

component MRAM is

port (

CLK: in std\_logic;

RW: in std\_logic;

ADDR: in mem\_addr;

DIN: in operand;

DOUT: out operand

);

end component;

component DPATH is

port(

EN: in std\_logic;

CLK: in std\_logic;

OT: in operation;

OP: in operand;

RES: out operand;

ZF: out std\_logic;

Stop: out std\_logic

);

end component;

-- Component declaration of the tested unit

component ctrl1

port(

CLK : in STD\_LOGIC;

RST : in STD\_LOGIC;

Start : in STD\_LOGIC;

Stop : out STD\_LOGIC;

ROM\_re : out STD\_LOGIC;

ROM\_addr : out mem\_addr;

ROM\_dout : in command;

RAM\_rw : out STD\_LOGIC;

RAM\_addr : out mem\_addr;

RAM\_din : out operand;

RAM\_dout : in operand;

DP\_op : out operand;

DP\_ot : out operation;

DP\_en : out STD\_LOGIC;

DP\_res : in operand;

DP\_zf : in STD\_LOGIC;

DP\_stop : in STD\_LOGIC );

end component;

signal CLK : STD\_LOGIC;

signal RST : STD\_LOGIC;

signal Start : STD\_LOGIC;

signal ROM\_dout : command;

signal RAM\_dout : operand;

signal DP\_res : operand;

signal DP\_zf : STD\_LOGIC;

signal DP\_stop : STD\_LOGIC;

signal Stop : STD\_LOGIC;

signal ROM\_re : STD\_LOGIC;

signal ROM\_addr : mem\_addr;

signal RAM\_rw : STD\_LOGIC;

signal RAM\_addr : mem\_addr;

signal RAM\_din : operand;

signal DP\_op : operand;

signal DP\_ot : operation;

signal DP\_en : STD\_LOGIC;

constant CLK\_period: time := 10 ns;

begin

-- Unit Under Test port map

UUT : ctrl1

port map (

CLK => CLK,

RST => RST,

Start => Start,

Stop => Stop,

ROM\_re => ROM\_re,

ROM\_addr => ROM\_addr,

ROM\_dout => ROM\_dout,

RAM\_rw => RAM\_rw,

RAM\_addr => RAM\_addr,

RAM\_din => RAM\_din,

RAM\_dout => RAM\_dout,

DP\_op => DP\_op,

DP\_ot => DP\_ot,

DP\_en => DP\_en,

DP\_res => DP\_res,

DP\_zf => DP\_zf,

DP\_stop => DP\_stop

);

UMRAM: MRAM

port map(

CLK => CLK,

RW => ram\_rw,

ADDR => ram\_addr,

DIN => ram\_din,

DOUT => ram\_dout

);

UMROM: entity MROM (Beh\_Stack)

port map (

RE => rom\_re,

ADDR => rom\_addr,

DOUT => rom\_dout

);

UDPATH: DPATH

port map(

EN => dp\_en,

CLK => CLK,

OT => dp\_ot,

OP => dp\_op,

RES => dp\_res,

ZF => dp\_zf,

STOP => dp\_stop

);

CLK\_Process: process

begin

CLK <= '0';

wait for CLK\_Period/2;

CLK <= '1';

wait for CLK\_Period/2;

end process;

main: process

begin

rst <= '1';

wait for 1 \* CLK\_PERIOD;

rst <= '0';

start <= '1';

wait for 100 \* CLK\_PERIOD;

wait;

end process;

end TB\_ARCHITECTURE;

configuration TESTBENCH\_FOR\_ctrl1 of ctrl1\_tb is

for TB\_ARCHITECTURE

for UUT : ctrl1

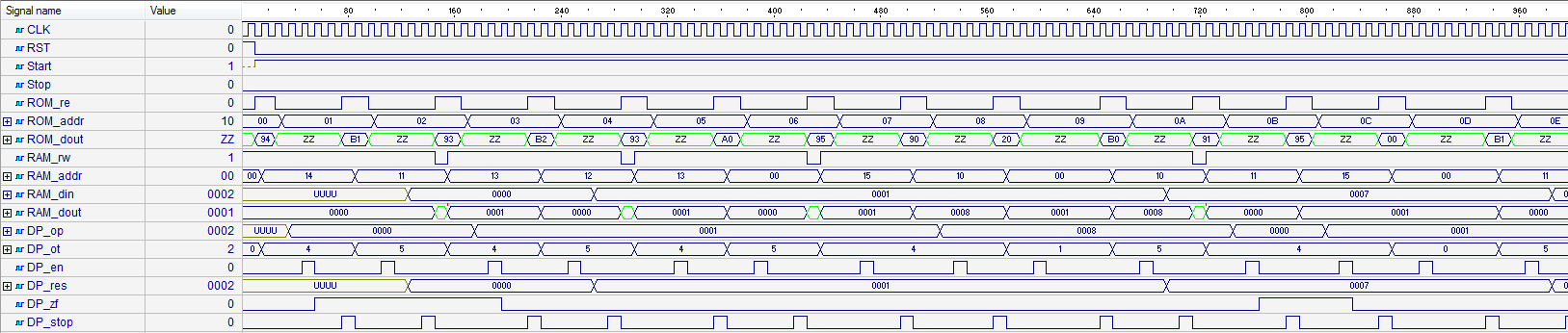
use entity work.ctrl1(beh\_stack);

end for;

end for;

end TESTBENCH\_FOR\_ctrl1;

* + 1. **Временные диаграммы функционирования**



* 1. Блок DPATH
     1. **HDL**

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

use IEEE.STD\_LOGIC\_ARITH.all;

use IEEE.STD\_LOGIC\_UNSIGNED.all;

library stack;

use stack.OneHotStack.all;

entity DPATH is

port(

EN: in std\_logic;

CLK: in std\_logic;

OT: in operation;

OP: in operand;

RES: out operand;

ZF: out std\_logic;

Stop: out std\_logic

);

end DPATH;

architecture Beh\_Stack of DPATH is

component LIFO

generic(

m: integer := 2;

n: integer := 2

);

port (

EN: in std\_logic;

CLK: in std\_logic;

WR: in std\_logic;

RB: out std\_logic\_vector(n-1 downto 0);

WB: in std\_logic\_vector(n-1 downto 0)

);

end component;

type states is (I, IPOP1, IPOP2, A, SB, SH, IPUSH, MOVERES, MOVERESOP, H);

signal nxt\_state, cur\_state: states;

signal res\_op: operand;

signal i\_op: operand;

signal i\_op1: operand;

signal i\_op2: operand;

signal i\_res: operand;

signal s\_en: std\_logic;

signal s\_wr: std\_logic;

signal s\_res: operand;

signal s\_data: operand;

signal t\_zf: std\_logic;

Begin

USTACK: LIFO

generic map(

m => 5,

n => 16

)

port map(

CLK => CLK,

EN => s\_en,

WR => s\_wr,

RB => s\_res,

WB => s\_data

);

i\_op <= OP;

FSM: process(CLK, nxt\_state)

begin

if rising\_edge(CLK) then

cur\_state <= nxt\_state;

end if;

end process;

COMB: process(cur\_state, EN, OT)

begin

case cur\_state is

when I =>

if (EN = '1') then

case OT is

when ADD | SUBT | SHIFT | POP | POPIN => nxt\_state <= IPOP1;

when others => nxt\_state <= MOVERESOP;

end case;

else

nxt\_state <= I;

end if;

when IPOP1 =>

if (OT = POP or OT = POPIN) then

nxt\_state <= MOVERES;

elsif (OT = SHIFT) then

nxt\_state <= SH;

else

nxt\_state <= IPOP2;

end if;

when IPOP2 =>

if (OT = ADD) then

nxt\_state <= A;

else

nxt\_state <= SB;

end if;

when A | SB | SH | MOVERESOP => nxt\_state <= IPUSH;

when MOVERES => nxt\_state <= H;

when IPUSH => nxt\_state <= H;

when H => nxt\_state <= I;

when others => nxt\_state <= I;

end case;

end process;

-- stop signal handler

PSTOP: process (cur\_state)

begin

if (cur\_state = H) then

stop <= '1';

else

stop <= '0';

end if;

end process;

STACKCTRL: process (cur\_state, nxt\_state)

begin

if (nxt\_state = IPOP1 or nxt\_state = IPOP2) then

s\_wr <= '1';

s\_en <= '1';

elsif (cur\_state = IPUSH) then

s\_wr <= '0';

s\_en <= '1';

else

s\_wr <= '1';

s\_en <= '0';

end if;

end process;

OP1CTRL: process (cur\_state, s\_res)

begin

if (cur\_state = IPOP1) then

i\_op1 <= s\_res;

end if;

end process;

OP2CTRL: process (cur\_state, s\_res)

begin

if (cur\_state = IPOP2) then

i\_op2 <= s\_res;

end if;

end process;

OPRESULTCTRL: process (cur\_state, i\_op1, i\_op2, i\_op)

begin

if (cur\_state = A) then

res\_op <= CONV\_STD\_LOGIC\_VECTOR(CONV\_INTEGER(i\_op1) + CONV\_INTEGER(i\_op2), 16);

elsif (cur\_state = SB) then

res\_op <= CONV\_STD\_LOGIC\_VECTOR(CONV\_INTEGER(i\_op1) - CONV\_INTEGER(i\_op2), 16);

elsif (cur\_state = SH) then

res\_op <= i\_op1(14 downto 0) & '0';

elsif (cur\_state = MOVERESOP) then

res\_op <= i\_op;

end if;

end process;

IRESCTRL: process (cur\_state, i\_op1)

begin

if (cur\_state = MOVERES) then

i\_res <= i\_op1;

end if;

end process;

FLAGS: process(res\_op)

begin

if res\_op = (res\_op'range => '0') then

t\_zf <= '1';

else

t\_zf <= '0';

end if;

end process;

s\_data <= res\_op;

RES <= i\_res;

ZF <= t\_zf;

End Beh\_Stack;

* + 1. **TestBench**

library stack;

use stack.OneHotStack.all;

library ieee;

use ieee.STD\_LOGIC\_UNSIGNED.all;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_arith.all;

entity dpath\_tb is

end dpath\_tb;

architecture TB\_ARCHITECTURE of dpath\_tb is

component dpath

port(

EN : in STD\_LOGIC;

CLK : in STD\_LOGIC;

OT : in operation;

OP : in operand;

RES : out operand;

ZF : out STD\_LOGIC;

Stop : out STD\_LOGIC );

end component;

signal EN : STD\_LOGIC;

signal CLK : STD\_LOGIC;

signal OT : operation;

signal OP : operand;

signal RES : operand;

signal ZF : STD\_LOGIC;

signal Stop : STD\_LOGIC;

constant CLK\_Period: time := 10 ns;

constant Stop\_WAIT: time := 5 \* CLK\_Period;

begin

-- Unit Under Test port map

UUT : dpath

port map (

EN => EN,

CLK => CLK,

OT => OT,

OP => OP,

RES => RES,

ZF => ZF,

Stop => Stop

);

CLK\_Process: process

begin

CLK <= '0';

wait for CLK\_Period/2;

CLK <= '1';

wait for CLK\_Period/2;

end process;

MAIN: process

begin

wait for clk\_period;

en <= '0';

op <= "0000000000000010";

ot <= PUSH;

wait for clk\_period;

en <= '1';

wait for clk\_period;

en <= '0';

wait for Stop\_WAIT;

en <= '1';

wait for clk\_period;

en <= '0';

wait for Stop\_WAIT;

ot <= ADD;

wait for clk\_period;

en <= '1';

wait for clk\_period;

en <= '0';

wait for Stop\_WAIT;

ot <= POP;

wait for clk\_period;

en <= '1';

wait for clk\_period;

en <= '0';

wait for Stop\_WAIT;

ot <= PUSH;

wait for clk\_period;

en <= '1';

wait for clk\_period;

en <= '0';

wait for Stop\_WAIT;

en <= '1';

wait for clk\_period;

en <= '0';

wait for Stop\_WAIT;

ot <= SUBT;

wait for clk\_period;

en <= '1';

wait for clk\_period;

en <= '0';

wait for Stop\_WAIT;

ot <= POP;

wait for clk\_period;

en <= '1';

wait for clk\_period;

en <= '0';

wait for Stop\_WAIT;

ot <= PUSH;

wait for clk\_period;

en <= '1';

wait for clk\_period;

en <= '0';

wait for Stop\_WAIT;

ot <= SHIFT;

wait for clk\_period;

en <= '1';

wait for clk\_period;

en <= '0';

wait for Stop\_WAIT;

ot <= POP;

wait for clk\_period;

en <= '1';

wait for clk\_period;

en <= '0';

wait for Stop\_WAIT;

wait;

end process;

end TB\_ARCHITECTURE;

configuration TESTBENCH\_FOR\_dpath of dpath\_tb is

for TB\_ARCHITECTURE

for UUT : dpath

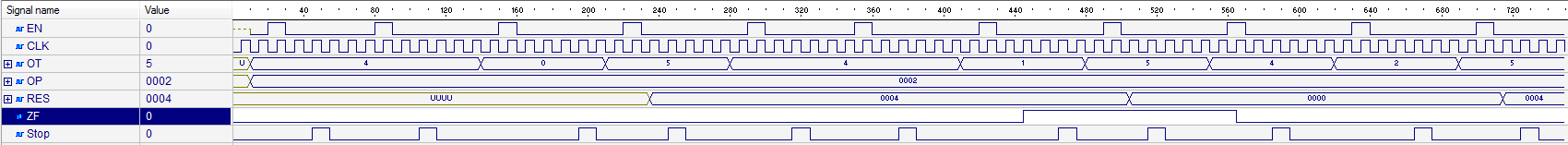
use entity work.dpath(beh\_stack);

end for;

end for;

end TESTBENCH\_FOR\_dpath;

* + 1. **Временные диаграммы функционирования**



1. **System Level**
   1. **HDL**

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

use IEEE.STD\_LOGIC\_UNSIGNED.all;

library stack;

use stack.OneHotStack.all;

entity OneHot is

port (

CLK, RST, Start: in std\_logic;

Stop: out std\_logic

);

end OneHot;

architecture Beh of OneHot is

component MROM is

port (

RE: in std\_logic;

ADDR: in mem\_addr;

DOUT: out command

);

end component;

component MRAM is

port (

CLK: in std\_logic;

RW: in std\_logic;

ADDR: in mem\_addr;

DIN: in operand;

DOUT: out operand

);

end component;

component DPATH is

port(

EN: in std\_logic;

CLK: in std\_logic;

OT: in operation;

OP: in operand;

RES: out operand;

ZF: out std\_logic;

Stop: out std\_logic

);

end component;

component CTRL1 is

port(

CLK, RST, Start: in std\_logic;

Stop: out std\_logic;

ROM\_re: out std\_logic;

ROM\_addr: out mem\_addr;

ROM\_dout: in command;

RAM\_rw: out std\_logic;

RAM\_addr: out mem\_addr;

RAM\_din: out operand;

RAM\_dout: in operand;

--datapath

DP\_op: out operand;

DP\_ot: out operation;

DP\_en: out std\_logic;

DP\_res: in operand;

DP\_zf: in std\_logic;

DP\_stop: in std\_logic

);

end component;

signal rom\_re: std\_logic;

signal rom\_addr: mem\_addr;

signal rom\_dout: command;

signal ram\_rw: std\_logic;

signal ram\_addr: mem\_addr;

signal ram\_din: operand;

signal ram\_dout: operand;

signal dp\_op: operand;

signal dp\_ot: operation;

signal dp\_en: std\_logic;

signal dp\_res: operand;

signal dp\_zf: std\_logic;

signal dp\_stop: std\_logic;

begin

UMRAM: MRAM

port map(

CLK => CLK,

RW => ram\_rw,

ADDR => ram\_addr,

DIN => ram\_din,

DOUT => ram\_dout

);

UMROM: MROM

port map (

RE => rom\_re,

ADDR => rom\_addr,

DOUT => rom\_dout

);

UDPATH: DPATH

port map(

EN => dp\_en,

CLK => CLK,

OT => dp\_ot,

OP => dp\_op,

RES => dp\_res,

ZF => dp\_zf,

STOP => dp\_stop

);

UCTRL1: CTRL1

port map(

CLK => CLK,

RST => RST,

START => Start,

STOP => STOP,

ROM\_re => rom\_re,

ROM\_addr => rom\_addr,

ROM\_dout => rom\_dout,

RAM\_rw => ram\_rw,

RAM\_addr => ram\_addr,

RAM\_din => ram\_din,

RAM\_dout => ram\_dout,

DP\_en => dp\_en,

DP\_ot => dp\_ot,

DP\_op => dp\_op,

DP\_res => dp\_res,

DP\_zf => dp\_zf,

DP\_stop => dp\_stop

);

end Beh;

* 1. **Test Bench**

library ieee;

use ieee.STD\_LOGIC\_UNSIGNED.all;

use ieee.std\_logic\_1164.all;

library stack;

use stack.OneHotStack.all;

entity onehot\_tb is

end onehot\_tb;

architecture TB\_ARCHITECTURE of onehot\_tb is

component onehot

port(

CLK : in STD\_LOGIC;

RST : in STD\_LOGIC;

Start : in STD\_LOGIC;

Stop : out STD\_LOGIC );

end component;

signal CLK : STD\_LOGIC;

signal RST : STD\_LOGIC;

signal Start : STD\_LOGIC;

signal Stop : STD\_LOGIC;

constant CLK\_period: time := 10 ns;

begin

UUT : onehot

port map (

CLK => CLK,

RST => RST,

Start => Start,

Stop => Stop

);

CLK\_Process: process

begin

CLK <= '0';

wait for CLK\_Period/2;

CLK <= '1';

wait for CLK\_Period/2;

end process;

main: process

begin

rst <= '1';

wait for 1 \* CLK\_PERIOD;

rst <= '0';

start <= '1';

wait for 100 \* CLK\_PERIOD;

wait;

end process;

end TB\_ARCHITECTURE;

configuration TESTBENCH\_FOR\_onehot of onehot\_tb is

for TB\_ARCHITECTURE

for UUT : onehot

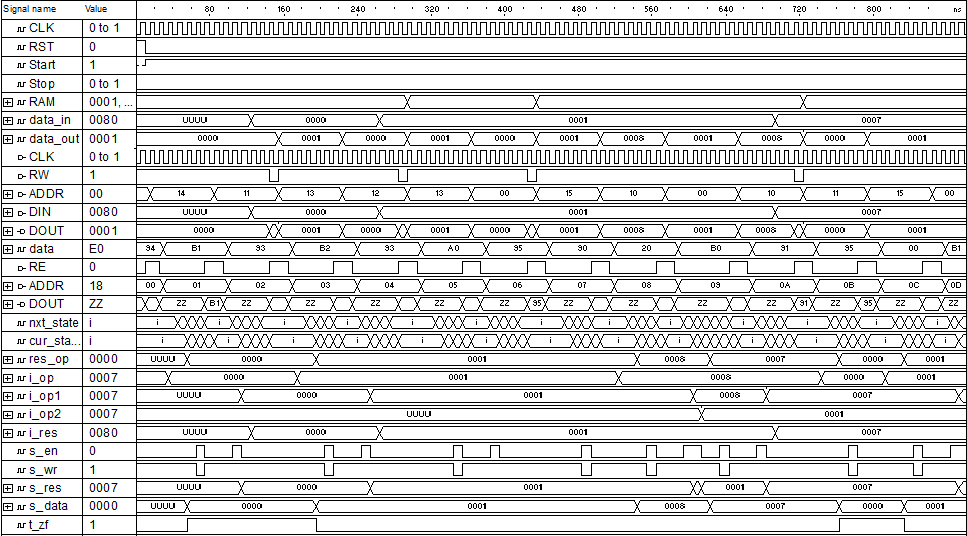
use entity work.onehot(beh);

end for;

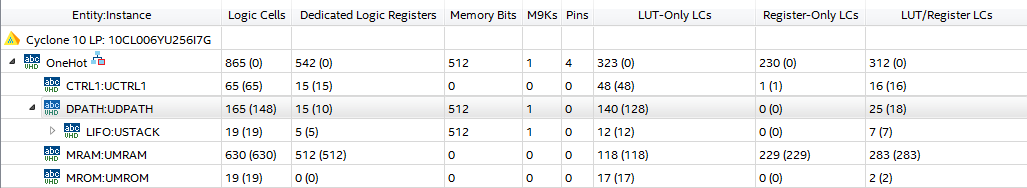
end for;

end TESTBENCH\_FOR\_onehot;

* 1. **Временные диаграммы функционирования**



1. **Анализ аппаратурных и временные затрат**
   1. Аппаратные затраты (Cyclone 10CL006YU256I7G / Inter Quartus Prime 17.1 Lite)



* 1. Временные затраты

Для генерации 8 элементов “One-Hot” последовательности требуется 710 тактов синхронизации.

1. Выводы

Accumulator

1. **Описание ISA**

|  |  |  |
| --- | --- | --- |
| Мнемоника | Код операции (3 бита) | Операнд (5 бит) |
| LOAD | 000 | aaaaa |
| STORE | 001 | aaaaa |
| ADD | 010 | aaaaa |
| SUBT | 011 | aaaaa |
| STOREIN | 100 | aaaaa |
| SHIFT | 101 | xxxxx |
| JNZ | 110 | aaaaa |
| HALT | 111 | xxxxx |

ADD addr – сложение значений хранящегося в аккумуляторе с значением хранящемся по адресу addr, результат записывается в аккумулятор

SUBT addr – вычитание значения хранящегося по адресу addr значения хранящегося в аккумуляторе, результат записывается в аккумулятор

SHIFT – логический сдвиг вправо значения хранящегося в аккумуляторе, результат записывается в аккумулятор

JNZ addr – переход на операции по адресу addr, при сброшенном флаге нуля

LOAD addr – пересылка значения хранящегося по адресу addr в аккумулятор

STORE addr – пересылка значения хранящегося в аккумуляторе по адресу addr

STOREIN addr – пересылка значения хранящегося в аккумуляторе по адресу хранящемуся по адресу addr.

HALT - останова

1. **Микропрограмма**

constant STUB: mem\_addr := "00000";

constant DATA\_START: mem\_addr := "00000";

constant LENGTH :mem\_addr := "10000";

constant COUNTER: mem\_addr := "10001";

constant CURRENT\_VALUE: mem\_addr := "10010";

constant INIT\_VALUE: mem\_addr := "10011";

constant ZERO: mem\_addr := "10100";

constant ONE: mem\_addr := "10101";

constant LOOP\_ADDR: mem\_addr := "01000";

---------------------------

LOAD & ZERO,

STORE & COUNTER,

LOAD & INIT\_VALUE,

STORE & DATA\_START,

STORE & CURRENT\_VALUE,

LOAD & LENGTH,

SUBT & ONE,

STORE & LENGTH,

LOAD & COUNTER,

ADD & ONE,

STORE & COUNTER,

LOAD & CURRENT\_VALUE,

SHIFT & STUB,

STORE & CURRENT\_VALUE,

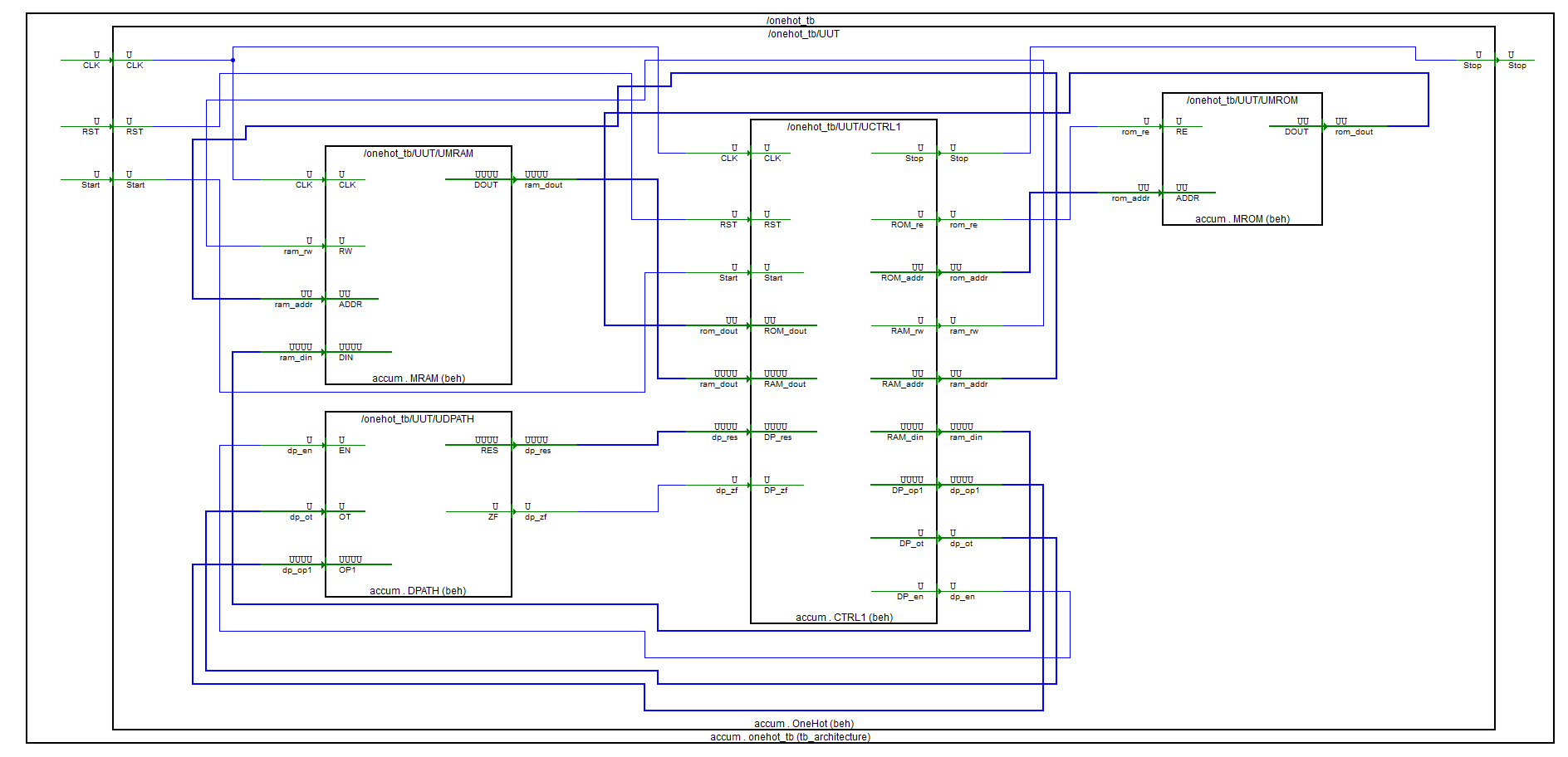
STOREIN & COUNTER,

LOAD & LENGTH,

SUBT & COUNTER,

JNZ & LOOP\_ADDR

1. **Общая структурная схема**

****

1. **Описание микроархитектуры**
2. **Block Level**
   1. Блок MROM
      1. **HDL**

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

use IEEE.STD\_LOGIC\_UNSIGNED.all;

library accum;

use accum.OneHotAccum.all;

entity MROM is

port (

RE: in std\_logic;

ADDR: in mem\_addr;

DOUT: out command

);

end MROM;

architecture Beh of MROM is

type tROM is array (0 to 31) of command;

constant STUB: mem\_addr := "00000";

constant DATA\_START: mem\_addr := "00000";

constant LENGTH :mem\_addr := "10000";

constant COUNTER: mem\_addr := "10001";

constant CURRENT\_VALUE: mem\_addr := "10010";

constant INIT\_VALUE: mem\_addr := "10011";

constant ZERO: mem\_addr := "10100";

constant ONE: mem\_addr := "10101";

constant LOOP\_ADDR: mem\_addr := "01000";

constant ROM: tROM :=(

-- OP CODE | OP1

-- Init

LOAD & ZERO,

STORE & COUNTER,

LOAD & INIT\_VALUE,

STORE & DATA\_START,

STORE & CURRENT\_VALUE,

LOAD & LENGTH,

SUBT & ONE,

STORE & LENGTH,

-- Loop

LOAD & COUNTER,

ADD & ONE,

STORE & COUNTER,

LOAD & CURRENT\_VALUE,

SHIFT & STUB,

STORE & CURRENT\_VALUE,

STOREIN & COUNTER,

LOAD & LENGTH,

SUBT & COUNTER,

JNZ & LOOP\_ADDR,

others => HALT & STUB

);

signal data: command;

begin

data <= ROM(conv\_integer(addr));

zbufs: process (RE, data)

begin

if (RE = '1') then

DOUT <= data;

else

DOUT <= (others => 'Z');

end if;

end process;

end Beh;

* + 1. **TestBench**

library accum;

use accum.OneHotAccum.all;

library ieee;

use ieee.STD\_LOGIC\_UNSIGNED.all;

use ieee.std\_logic\_1164.all;

entity mrom\_tb is

end mrom\_tb;

architecture TB\_ARCHITECTURE of mrom\_tb is

component mrom

port(

RE : in STD\_LOGIC;

ADDR : in mem\_addr;

DOUT : out command );

end component;

signal RE : STD\_LOGIC;

signal ADDR : mem\_addr;

signal DOUT : command;

constant WAIT\_period: time := 10 ns;

begin

UUT : mrom

port map (

RE => RE,

ADDR => ADDR,

DOUT => DOUT

);

main: process

begin

re <= '0';

addr <= "00010";

wait for 1 \* WAIT\_period;

re <= '1';

wait for 1 \* WAIT\_period;

addr <= "00000";

re <= '1';

wait for 1 \* WAIT\_period;

re <= '0';

wait for 100 \* WAIT\_period;

wait;

end process;

end TB\_ARCHITECTURE;

configuration TESTBENCH\_FOR\_mrom of mrom\_tb is

for TB\_ARCHITECTURE

for UUT : mrom

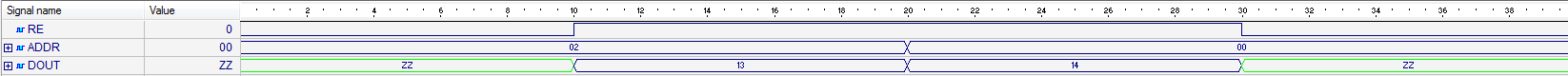
use entity work.mrom(beh);

end for;

end for;

end TESTBENCH\_FOR\_mrom;

* + 1. **Временные диаграммы функционирования**



* 1. Блок MRAM
     1. **HDL**

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

use IEEE.STD\_LOGIC\_UNSIGNED.all;

library accum;

use accum.OneHotAccum.all;

entity MRAM is

port (

CLK: in std\_logic;

RW: in std\_logic;

ADDR: in mem\_addr;

DIN: in operand;

DOUT: out operand

);

end MRAM;

architecture Beh of MRAM is

type tRAM is array (0 to 31) of operand;

signal RAM: tRAM:= (

-- | BIN | ADR BIN

"0000000000000000", -- | 00000 |

"0000000000000000", -- | 00001 |

"0000000000000000", -- | 00010 |

"0000000000000000", -- | 00011 |

"0000000000000000", -- | 00100 |

"0000000000000000", -- | 00101 |

"0000000000000000", -- | 00110 |

"0000000000000000", -- | 00111 |

"0000000000000000", -- | 01000 |

"0000000000000000", -- | 01001 |

"0000000000000000", -- | 01010 |

"0000000000000000", -- | 01011 |

"0000000000000000", -- | 01100 |

"0000000000000000", -- | 01101 |

"0000000000000000", -- | 01110 |

"0000000000000000", -- | 01111 |

"0000000000001000", -- | 10000 |

"0000000000000000", -- | 10001 |

"0000000000000000", -- | 10010 |

"0000000000000001", -- | 10011 |

"0000000000000000", -- | 10100 |

"0000000000000001", -- | 10101 |

others => "0000000000000000"

);

signal data\_in: operand;

signal data\_out: operand;

Begin

data\_in <= Din;

WRITE: process (CLK, RW, ADDR, data\_in)

begin

if (RW = '0') then

if (rising\_edge(CLK)) then

RAM(conv\_integer(ADDR)) <= data\_in;

end if;

end if;

end process;

data\_out <= RAM (conv\_integer(ADDR));

RDP: process (RW, RAM, data\_out)

begin

if (RW = '1') then

DOUT <= data\_out;

else

DOUT <= (others => 'Z');

end if;

end process;

end Beh;

* + 1. **TestBench**

library accum;

use accum.OneHotAccum.all;

library ieee;

use ieee.STD\_LOGIC\_UNSIGNED.all;

use ieee.std\_logic\_1164.all;

entity mram\_tb is

end mram\_tb;

architecture TB\_ARCHITECTURE of mram\_tb is

component mram

port(

CLK : in STD\_LOGIC;

RW : in STD\_LOGIC;

ADDR : in mem\_addr;

DIN : in operand;

DOUT : out operand );

end component;

signal CLK : STD\_LOGIC;

signal RW : STD\_LOGIC;

signal ADDR : mem\_addr;

signal DIN : operand;

signal DOUT : operand;

constant CLK\_period: time := 10 ns;

begin

UUT : mram

port map (

CLK => CLK,

RW => RW,

ADDR => ADDR,

DIN => DIN,

DOUT => DOUT

);

CLK\_Process: process

begin

CLK <= '0';

wait for CLK\_Period/2;

CLK <= '1';

wait for CLK\_Period/2;

end process;

main: process

begin

wait for CLK\_PERIOD;

addr <= "00010";

din <= "0000000000000100";

wait for CLK\_PERIOD;

addr <= "00001";

rw <= '1';

wait for CLK\_PERIOD;

addr <= "00000";

din <= "0000000000010000";

rw <= '0';

wait for CLK\_PERIOD;

wait;

end process;

end TB\_ARCHITECTURE;

configuration TESTBENCH\_FOR\_mram of mram\_tb is

for TB\_ARCHITECTURE

for UUT : mram

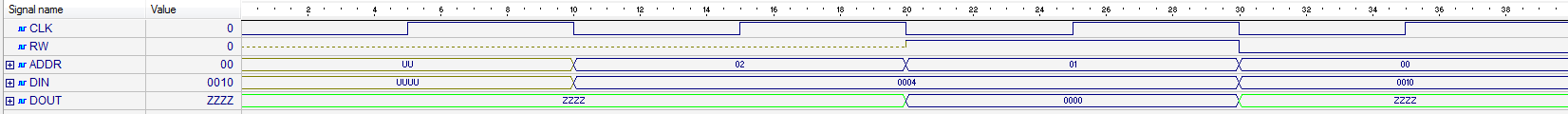
use entity work.mram(beh);

end for;

end for;

end TESTBENCH\_FOR\_mram;

* + 1. **Временные диаграммы функционирования**



* 1. Блок CTRL
     1. **HDL**

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

use IEEE.STD\_LOGIC\_UNSIGNED.all;

library accum;

use accum.OneHotAccum.all;

entity CTRL1 is

port(

CLK, RST, Start: in std\_logic;

Stop: out std\_logic;

-- ROM

ROM\_re: out std\_logic;

ROM\_addr: out mem\_addr;

ROM\_dout: in command;

-- RAM

RAM\_rw: out std\_logic;

RAM\_addr: out mem\_addr;

RAM\_din: out operand;

RAM\_dout: in operand;

--datapath

DP\_op1: out operand;

DP\_ot: out operation;

DP\_en: out std\_logic;

DP\_res: in operand;

DP\_zf: in std\_logic

);

end CTRL1;

architecture Beh of CTRL1 is

type states is (I, F, D, R, L, SH, S, A, SB, H, JNOTZ, SIN);

signal nxt\_state, cur\_state: states;

signal RI: command;

signal IC: mem\_addr;

signal RO: operation;

signal RA: mem\_addr;

signal RD: operand;

begin

FSM: process(CLK, RST, nxt\_state)

begin

if (RST = '1') then

cur\_state <= I;

elsif rising\_edge(CLK) then

cur\_state <= nxt\_state;

end if;

end process;

-- Next state

COMB: process(cur\_state, start, RO)

begin

case cur\_state is

when I =>

if (start = '1') then

nxt\_state <= F;

else

nxt\_state <= I;

end if;

when F => nxt\_state <= D;

when D =>

if (RO = HALT) then

nxt\_state <= H;

elsif (RO = STORE) then

nxt\_state <= S;

elsif (RO = JNZ) then

nxt\_state <= JNOTZ;

elsif (RO = SHIFT) then

nxt\_state <= SH;

else

nxt\_state <= R;

end if;

when R =>

if (RO = LOAD) then

nxt\_state <= L;

elsif (RO = ADD) then

nxt\_state <= A;

elsif (RO = SUBT) then

nxt\_state <= SB;

elsif (RO = STOREIN) then

nxt\_state <= SIN;

else

nxt\_state <= I;

end if;

when SIN => nxt\_state <= S;

when L | S | A | SB | SH | JNOTZ => nxt\_state <= F;

when H => nxt\_state <= H;

when others => nxt\_state <= I;

end case;

end process;

-- stop signal

PSTOP: process (cur\_state)

begin

if (cur\_state = H) then

stop <= '1';

else

stop <= '0';

end if;

end process;

-- instruction counter

PMC: process (CLK, RST, cur\_state)

begin

if (RST = '1') then

IC <= "00000";

elsif falling\_edge(CLK) then

if (cur\_state = D) then

IC <= IC + 1;

elsif (cur\_state = JNOTZ and DP\_ZF = '0') then

IC <= RA;

end if;

end if;

end process;

ROM\_addr <= IC;

-- ROM read signal

PROMREAD: process (nxt\_state, cur\_state)

begin

if (nxt\_state = F or cur\_state = F) then

ROM\_re <= '1';

else

ROM\_re <= '0';

end if;

end process;

-- read ROM value and put it into RI

PROMDAT: process (RST, cur\_state, ROM\_dout)

begin

if (RST = '1') then

RI <= "00000000";

elsif (cur\_state = F) then

RI <= ROM\_dout;

end if;

end process;

-- RO and RA control

PRORA: process (RST, nxt\_state, RI)

begin

if (RST = '1') then

RO <= "000";

RA <= "00000";

elsif (nxt\_state = D) then

RO <= RI (7 downto 5);

RA <= RI (4 downto 0);

elsif (nxt\_state = SIN) then

RA <= RD (4 downto 0);

end if;

end process;

PRAMST: process (RA)

begin

if (cur\_state /= JNOTZ) then

RAM\_addr <= RA;

end if;

end process;

-- RAM read/write control

PRAMREAD: process (cur\_state)

begin

if (cur\_state = S) then

RAM\_rw <= '0';

else

RAM\_rw <= '1';

end if;

end process;

-- read value from RAM and put it into RD

PRAMDAR: process (cur\_state)

begin

if (cur\_state = R) then

RD <= RAM\_dout;

end if;

end process;

-- move the value from DPATH to RAM input bus

RAM\_din <= DP\_res;

-- move the value from RD to datapath

DP\_op1 <= RD;

-- move RO value to DP operation bus

DP\_ot <= RO;

paddsuben: process (cur\_state)

begin

if (cur\_state = A or cur\_state = SB or cur\_state = SH or cur\_state = L) then

DP\_en <= '1';

else

DP\_en <= '0';

end if;

end process;

end Beh;

* + 1. **TestBench**

library accum;

use accum.OneHotAccum.all;

library ieee;

use ieee.STD\_LOGIC\_UNSIGNED.all;

use ieee.std\_logic\_1164.all;

entity ctrl1\_tb is

end ctrl1\_tb;

architecture TB\_ARCHITECTURE of ctrl1\_tb is

component MROM is

port (

RE: in std\_logic;

ADDR: in mem\_addr;

DOUT: out command

);

end component;

component MRAM is

port (

CLK: in std\_logic;

RW: in std\_logic;

ADDR: in mem\_addr;

DIN: in operand;

DOUT: out operand

);

end component;

component DPATH is

port(

EN: in std\_logic;

OT: in operation;

OP1 : in operand;

RES: out operand;

ZF: out std\_logic

);

end component;

component ctrl1

port(

CLK : in STD\_LOGIC;

RST : in STD\_LOGIC;

Start : in STD\_LOGIC;

Stop : out STD\_LOGIC;

ROM\_re : out STD\_LOGIC;

ROM\_addr : out mem\_addr;

ROM\_dout : in command;

RAM\_rw : out STD\_LOGIC;

RAM\_addr : out mem\_addr;

RAM\_din : out operand;

RAM\_dout : in operand;

DP\_op1 : out operand;

DP\_ot : out operation;

DP\_en : out STD\_LOGIC;

DP\_res : in operand;

DP\_zf : in STD\_LOGIC );

end component;

signal CLK : STD\_LOGIC;

signal RST : STD\_LOGIC;

signal Start : STD\_LOGIC;

signal ROM\_dout : command;

signal RAM\_dout : operand;

signal DP\_res : operand;

signal DP\_zf : STD\_LOGIC;

signal Stop : STD\_LOGIC;

signal ROM\_re : STD\_LOGIC;

signal ROM\_addr : mem\_addr;

signal RAM\_rw : STD\_LOGIC;

signal RAM\_addr : mem\_addr;

signal RAM\_din : operand;

signal DP\_op1 : operand;

signal DP\_ot : operation;

signal DP\_en : STD\_LOGIC;

constant CLK\_period: time := 10 ns;

begin

UMRAM: MRAM

port map(

CLK => CLK,

RW => ram\_rw,

ADDR => ram\_addr,

DIN => ram\_din,

DOUT => ram\_dout

);

UMROM: MROM

port map (

RE => rom\_re,

ADDR => rom\_addr,

DOUT => rom\_dout

);

UDPATH: DPATH

port map(

EN => dp\_en,

OT => dp\_ot,

OP1 => dp\_op1,

RES => dp\_res,

ZF => dp\_zf

);

UCTRL1: CTRL1

port map(

CLK => CLK,

RST => RST,

START => Start,

STOP => STOP,

ROM\_RE => rom\_re,

ROM\_ADDR => rom\_addr,

ROM\_DOUT => rom\_dout,

RAM\_RW => ram\_rw,

RAM\_ADDR => ram\_addr,

RAM\_DIN => ram\_din,

RAM\_DOUT => ram\_dout,

DP\_EN => dp\_en,

DP\_OT => dp\_ot,

DP\_OP1 => dp\_op1,

DP\_RES => dp\_res,

DP\_ZF => dp\_zf

);

UUT : ctrl1

port map (

CLK => CLK,

RST => RST,

Start => Start,

Stop => Stop,

ROM\_re => ROM\_re,

ROM\_addr => ROM\_addr,

ROM\_dout => ROM\_dout,

RAM\_rw => RAM\_rw,

RAM\_addr => RAM\_addr,

RAM\_din => RAM\_din,

RAM\_dout => RAM\_dout,

DP\_op1 => DP\_op1,

DP\_ot => DP\_ot,

DP\_en => DP\_en,

DP\_res => DP\_res,

DP\_zf => DP\_zf

);

CLK\_Process: process

begin

CLK <= '0';

wait for CLK\_Period/2;

CLK <= '1';

wait for CLK\_Period/2;

end process;

main: process

begin

rst <= '1';

wait for 1 \* CLK\_PERIOD;

rst <= '0';

start <= '1';

wait for 100 \* CLK\_PERIOD;

wait;

end process;

end TB\_ARCHITECTURE;

configuration TESTBENCH\_FOR\_ctrl1 of ctrl1\_tb is

for TB\_ARCHITECTURE

for UUT : ctrl1

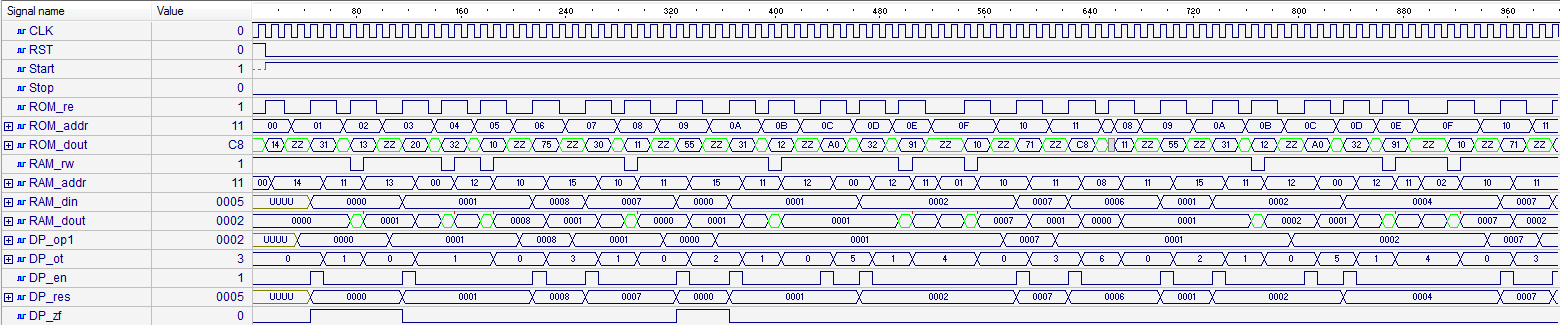
use entity work.ctrl1(beh);

end for;

end for;

end TESTBENCH\_FOR\_ctrl1;

* + 1. **Временные диаграммы функционирования**



* 1. Блок DPATH
     1. **HDL**

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

use IEEE.STD\_LOGIC\_ARITH.all;

use IEEE.STD\_LOGIC\_UNSIGNED.all;

library accum;

use accum.OneHotAccum.all;

entity DPATH is

port(

EN: in std\_logic;

-- operation type

OT: in operation;

-- operand

OP1: in operand;

RES: out operand;

-- zero flag

ZF: out std\_logic

);

end DPATH;

architecture Beh of DPATH is

signal ACCUM: operand;

signal res\_add: operand;

signal res\_sub: operand;

signal res\_shift: operand;

signal t\_zf: std\_logic;

Begin

res\_add <= CONV\_STD\_LOGIC\_VECTOR(CONV\_INTEGER(ACCUM) + CONV\_INTEGER(OP1), 16);

res\_sub <= CONV\_STD\_LOGIC\_VECTOR(CONV\_INTEGER(ACCUM) - CONV\_INTEGER(OP1), 16);

res\_shift <= ACCUM(14 downto 0) & '0';

REGA: process (EN, OT, OP1, res\_add, res\_sub, res\_shift)

begin

if rising\_edge(EN) then

case OT is

when LOAD => ACCUM <= OP1;

when ADD => ACCUM <= res\_add;

when SUBT => ACCUM <= res\_sub;

when SHIFT => ACCUM <= res\_shift;

when others => null;

end case;

end if;

end process;

FLAGS: process(ACCUM)

begin

if ACCUM = (ACCUM'range => '0') then

t\_zf <= '1';

else

t\_zf <= '0';

end if;

end process;

RES <= ACCUM;

ZF <= t\_zf;

End Beh;

* + 1. **TestBench**

library accum;

use accum.OneHotAccum.all;

library ieee;

use ieee.STD\_LOGIC\_UNSIGNED.all;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_arith.all;

-- Add your library and packages declaration here ...

entity dpath\_tb is

end dpath\_tb;

architecture TB\_ARCHITECTURE of dpath\_tb is

-- Component declaration of the tested unit

component dpath

port(

EN : in STD\_LOGIC;

OT : in operation;

OP1 : in operand;

RES : out operand;

ZF : out STD\_LOGIC );

end component;

-- Stimulus signals - signals mapped to the input and inout ports of tested entity

signal EN : STD\_LOGIC;

signal OT : operation;

signal OP1 : operand;

-- Observed signals - signals mapped to the output ports of tested entity

signal RES : operand;

signal ZF : STD\_LOGIC;

-- Add your code here ...

constant WAIT\_period: time := 10 ns;

begin

-- Unit Under Test port map

UUT : dpath

port map (

EN => EN,

OT => OT,

OP1 => OP1,

RES => RES,

ZF => ZF

);

-- Add your stimulus here ...

MAIN: process

begin

wait for WAIT\_period;

en <= '1';

wait for WAIT\_period;

en <= '0';

wait for WAIT\_period;

op1 <= "0000000000000010";

en <= '1';

ot <= LOAD;

wait for WAIT\_period;

en <= '0';

op1 <= "0000000000100000";

wait for WAIT\_period;

ot <= ADD;

en <= '1';

wait for WAIT\_period;

en <= '0';

wait for WAIT\_period;

ot <= SUBT;

en <= '1';

wait for WAIT\_period;

en <= '0';

wait for WAIT\_period;

wait;

end process;

end TB\_ARCHITECTURE;

configuration TESTBENCH\_FOR\_dpath of dpath\_tb is

for TB\_ARCHITECTURE

for UUT : dpath

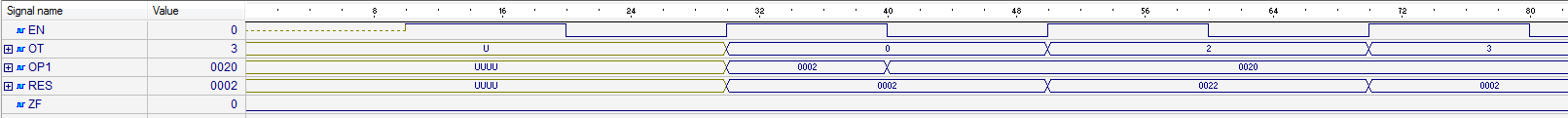
use entity work.dpath(beh);

end for;

end for;

end TESTBENCH\_FOR\_dpath;

* + 1. **Временные диаграммы функционирования**



1. **System Level**
   1. **HDL**

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

use IEEE.STD\_LOGIC\_UNSIGNED.all;

library accum;

use accum.OneHotAccum.all;

entity OneHot is

port (

CLK, RST, Start: in std\_logic;

Stop: out std\_logic

);

end OneHot;

architecture Beh of OneHot is

component MROM is

port (

RE: in std\_logic;

ADDR: in mem\_addr;

DOUT: out command

);

end component;

component MRAM is

port (

CLK: in std\_logic;

RW: in std\_logic;

ADDR: in mem\_addr;

DIN: in operand;

DOUT: out operand

);

end component;

component DPATH is

port(

EN: in std\_logic;

OT: in operation;

OP1 : in operand;

RES: out operand;

ZF: out std\_logic

);

end component;

component CTRL1 is

port(

CLK, RST, Start: in std\_logic;

Stop: out std\_logic;

-- ROM

ROM\_re: out std\_logic;

ROM\_addr: out mem\_addr;

ROM\_dout: in command;

-- RAM

RAM\_rw: out std\_logic;

RAM\_addr: out mem\_addr;

RAM\_din: out operand;

RAM\_dout: in operand;

-- datapath

DP\_op1: out operand;

DP\_ot: out operation;

DP\_en: out std\_logic;

DP\_res: in operand;

DP\_zf: in std\_logic

);

end component;

signal rom\_re: std\_logic;

signal rom\_addr: mem\_addr;

signal rom\_dout: command;

signal ram\_rw: std\_logic;

signal ram\_addr: mem\_addr;

signal ram\_din: operand;

signal ram\_dout: operand;

signal dp\_op1: operand;

signal dp\_ot: operation;

signal dp\_en: std\_logic;

signal dp\_res: operand;

signal dp\_zf: std\_logic;

begin

UMRAM: MRAM

port map(

CLK => CLK,

RW => ram\_rw,

ADDR => ram\_addr,

DIN => ram\_din,

DOUT => ram\_dout

);

UMROM: MROM

port map (

RE => rom\_re,

ADDR => rom\_addr,

DOUT => rom\_dout

);

UDPATH: DPATH

port map(

EN => dp\_en,

OT => dp\_ot,

OP1 => dp\_op1,

RES => dp\_res,

ZF => dp\_zf

);

UCTRL1: CTRL1

port map(

CLK => CLK,

RST => RST,

START => Start,

STOP => STOP,

ROM\_RE => rom\_re,

ROM\_ADDR => rom\_addr,

ROM\_DOUT => rom\_dout,

RAM\_RW => ram\_rw,

RAM\_ADDR => ram\_addr,

RAM\_DIN => ram\_din,

RAM\_DOUT => ram\_dout,

DP\_EN => dp\_en,

DP\_OT => dp\_ot,

DP\_OP1 => dp\_op1,

DP\_RES => dp\_res,

DP\_ZF => dp\_zf

);

end Beh;

* 1. **Test Bench**

library ieee;

use ieee.STD\_LOGIC\_UNSIGNED.all;

use ieee.std\_logic\_1164.all;

entity onehot\_tb is

end onehot\_tb;

architecture TB\_ARCHITECTURE of onehot\_tb is

component onehot

port(

CLK : in STD\_LOGIC;

RST : in STD\_LOGIC;

Start : in STD\_LOGIC;

Stop : out STD\_LOGIC );

end component;

signal CLK : STD\_LOGIC;

signal RST : STD\_LOGIC;

signal Start : STD\_LOGIC;

signal Stop : STD\_LOGIC;

constant CLK\_period: time := 10 ns;

begin

UUT : onehot

port map (

CLK => CLK,

RST => RST,

Start => Start,

Stop => Stop

);

CLK\_Process: process

begin

CLK <= '0';

wait for CLK\_Period/2;

CLK <= '1';

wait for CLK\_Period/2;

end process;

main: process

begin

rst <= '1';

wait for 1 \* CLK\_PERIOD;

rst <= '0';

start <= '1';

wait for 100 \* CLK\_PERIOD;

wait;

end process;

end TB\_ARCHITECTURE;

configuration TESTBENCH\_FOR\_onehot of onehot\_tb is

for TB\_ARCHITECTURE

for UUT : onehot

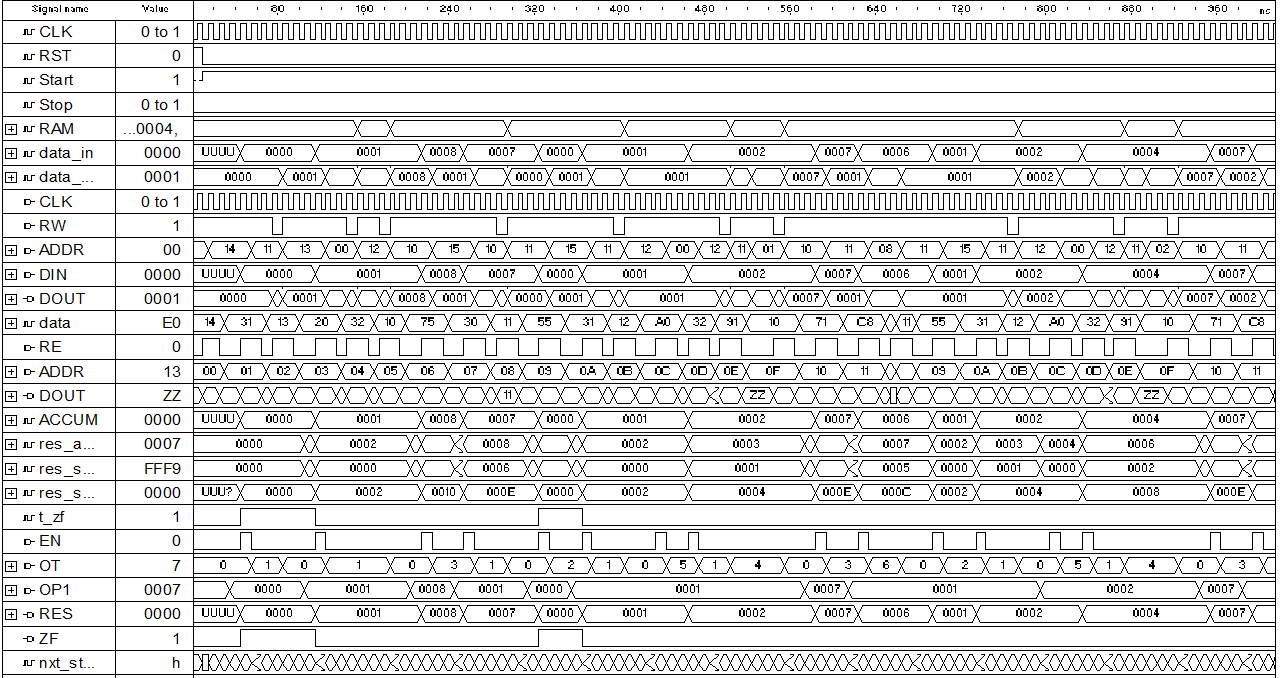
use entity work.onehot(beh);

end for;

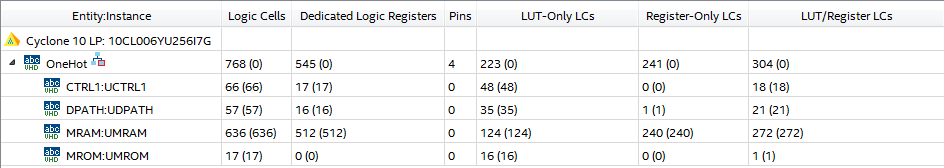
end for;

end TESTBENCH\_FOR\_onehot;

* 1. **Временные диаграммы функционирования**



1. **Анализ аппаратурных и временные затрат**
   1. Аппаратные затраты (Cyclone 10CL006YU256I7G / Inter Quartus Prime 17.1 Lite)



* 1. Временные затраты

Для генерации 8 элементов “One-Hot” последовательности требуется 290 тактов синхронизации.

1. Выводы

GPR

1. **Описание ISA**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Мнемоника | Код операции (3 бита) | Операнд #1 (5 бит) | Операнд #2 (5 бит) | Операнд #3 (5 бит) |
| ADD | 000 | aaaaa | aaaaa | aaaaa |
| SUBT | 010 | aaaaa | aaaaa | aaaaa |
| SHIFT | 011 | aaaaa | xxxxx | aaaaa |
| JNZ | 100 | aaaaa | xxxxx | xxxxxx |
| COPY | 101 | aaaaa | aaaaa | xxxxxx |
| HALT | 111 | xxxxx | xxxxx | xxxxx |

ADD addr1, addr2, addr3 – сложение значений хранящегося по адресу addr1 и addr2, результат записывается по адресу addr3

SUBT addr1, addr2, addr3 – вычитание из значений хранящегося по адресу addr1 значение хранящееся по адресу addr2, результат записывается по адресу addr3

SHIFT addr1, addr3 – логический сдвиг вправо значения хранящегося по адресу addr1, результат записывается по адресу addr3

JNZ addr1 – переход на операции по адресу addr1, при сброшенном флаге нуля

COPY addr1, adde3 – пересылка значения хранящегося по адресу addr1 по адресу хранящемуся по адресу addr3.

HALT - останова

1. **Микропрограмма**

constant STUB: mem\_addr := "00000";

constant LENGTH: mem\_addr := "10000";

constant COUNTER: mem\_addr := "10001";

constant CURRENT\_VALUE: mem\_addr := "10010";

constant DATA\_START: mem\_addr := "00000";

constant INIT\_VALUE: mem\_addr := "10011";

constant ZERO: mem\_addr := "10100";

constant ONE: mem\_addr := "10101";

constant EMPTY\_SPACE: mem\_addr := "10110";

constant LOOP\_ADDR: mem\_addr := "00100";

--------------

ADD & ZERO & INIT\_VALUE & DATA\_START,

ADD & ZERO & INIT\_VALUE & CURRENT\_VALUE,

ADD & ZERO & ZERO & COUNTER,

SUBT & LENGTH & ONE & LENGTH,

ADD & COUNTER & ONE & COUNTER,

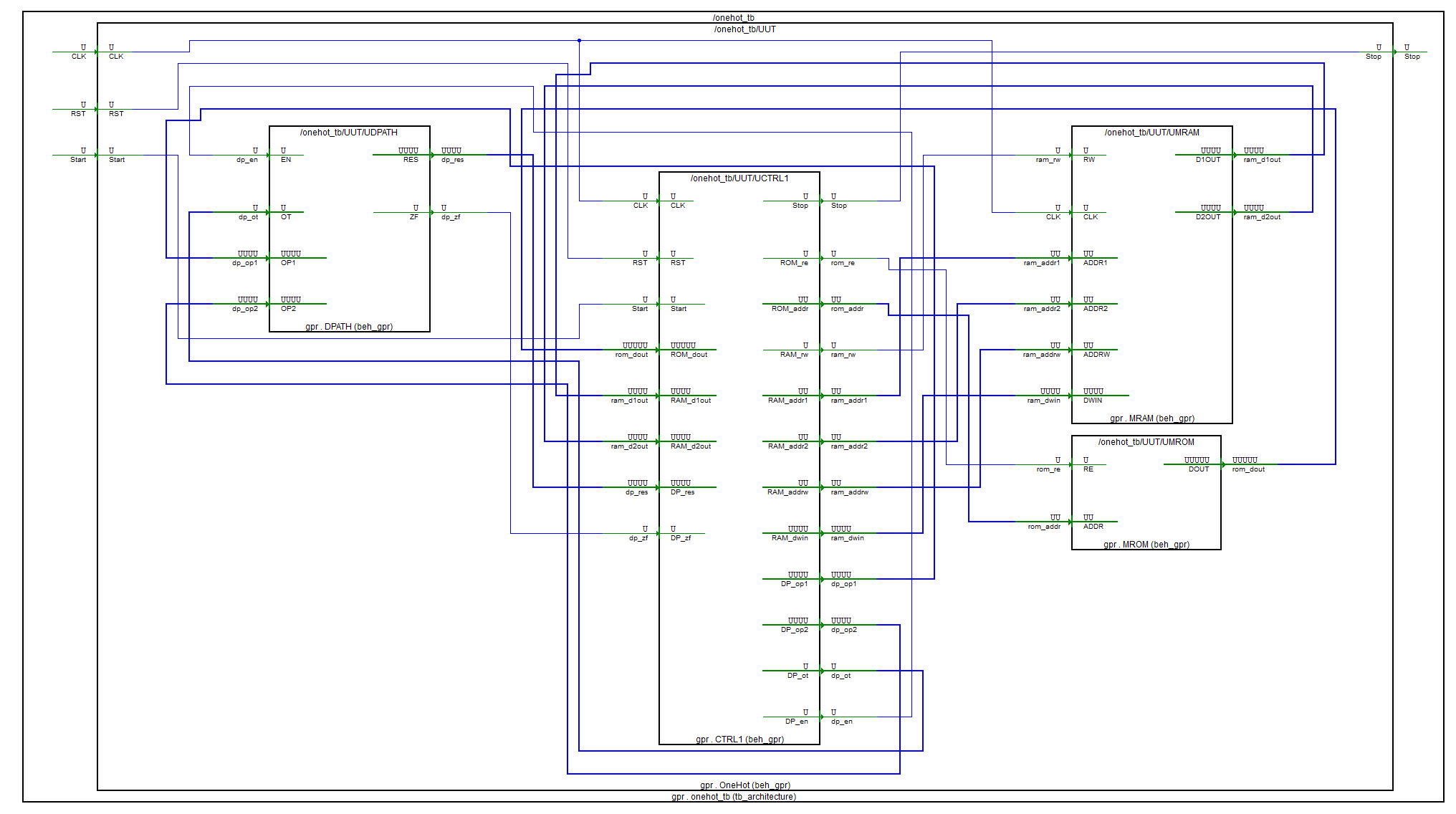
SHIFT & CURRENT\_VALUE & STUB & CURRENT\_VALUE,

COPY & CURRENT\_VALUE & COUNTER & STUB,

SUBT & LENGTH & COUNTER & EMPTY\_SPACE,

JNZ & LOOP\_ADDR & STUB & STUB

1. **Общая структурная схема**



1. **Описание микроархитектуры**
2. **Block Level**
   1. Блок MROM
      1. **HDL**

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

use IEEE.STD\_LOGIC\_UNSIGNED.all;

library gpr;

use gpr.OneHotGPR.all;

entity MROM is

port (

RE: in std\_logic;

ADDR: in mem\_addr;

DOUT: out command

);

end MROM;

architecture Beh\_GPR of MROM is

type tROM is array (0 to 31) of command;

constant STUB: mem\_addr := "00000";

constant LENGTH: mem\_addr := "10000";

constant COUNTER: mem\_addr := "10001";

constant CURRENT\_VALUE: mem\_addr := "10010";

constant DATA\_START: mem\_addr := "00000";

constant INIT\_VALUE: mem\_addr := "10011";

constant ZERO: mem\_addr := "10100";

constant ONE: mem\_addr := "10101";

constant EMPTY\_SPACE: mem\_addr := "10110";

constant LOOP\_ADDR: mem\_addr := "00100";

constant ROM: tROM :=(

-- OP CODE | OP1 | OP2 | RESULT

-- Init

ADD & ZERO & INIT\_VALUE & DATA\_START,

ADD & ZERO & INIT\_VALUE & CURRENT\_VALUE,

ADD & ZERO & ZERO & COUNTER,

SUBT & LENGTH & ONE & LENGTH,

-- Loop

ADD & COUNTER & ONE & COUNTER,

SHIFT & CURRENT\_VALUE & STUB & CURRENT\_VALUE,

COPY & CURRENT\_VALUE & COUNTER & STUB,

SUBT & LENGTH & COUNTER & EMPTY\_SPACE,

JNZ & LOOP\_ADDR & STUB & STUB,

others => HALT & STUB & STUB & STUB

);

signal data: command;

begin

data <= ROM(conv\_integer(ADDR));

zbufs: process (RE, data)

begin

if (RE = '1') then

DOUT <= data;

else

DOUT <= (others => 'Z');

end if;

end process;

end Beh\_GPR;

* + 1. **TestBench**

library gpr;

use gpr.OneHotGPR.all;

library ieee;

use ieee.STD\_LOGIC\_UNSIGNED.all;

use ieee.std\_logic\_1164.all;

-- Add your library and packages declaration here ...

entity mrom\_tb is

end mrom\_tb;

architecture TB\_ARCHITECTURE of mrom\_tb is

-- Component declaration of the tested unit

component mrom

port(

RE : in STD\_LOGIC;

ADDR : in mem\_addr;

DOUT : out command );

end component;

signal RE : STD\_LOGIC;

signal ADDR : mem\_addr;

signal DOUT : command;

constant WAIT\_period: time := 10 ns;

begin

-- Unit Under Test port map

UUT : mrom

port map (

RE => RE,

ADDR => ADDR,

DOUT => DOUT

);

-- Add your stimulus here ...

main: process

begin

re <= '0';

addr <= "00010";

wait for WAIT\_period;

re <= '1';

wait for WAIT\_period;

addr <= "00000";

re <= '1';

wait for WAIT\_period;

re <= '0';

wait for WAIT\_period;

wait;

end process;

end TB\_ARCHITECTURE;

configuration TESTBENCH\_FOR\_mrom of mrom\_tb is

for TB\_ARCHITECTURE

for UUT : mrom

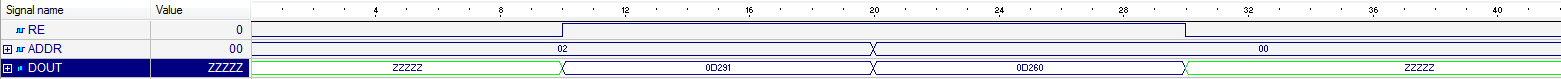
use entity work.mrom(beh\_gpr);

end for;

end for;

end TESTBENCH\_FOR\_mrom;

* + 1. **Временные диаграммы функционирования**



* 1. Блок MRAM
     1. **HDL**

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

use IEEE.STD\_LOGIC\_UNSIGNED.all;

library gpr;

use gpr.OneHotGPR.all;

entity MRAM is

port(

-- 0 - write; 1 - read

RW: in std\_logic;

CLK: in std\_logic;

-- address for the first operand

ADDR1: in mem\_addr;

-- address for the second operand

ADDR2: in mem\_addr;

-- address to write

ADDRW: in mem\_addr;

-- first operand

D1OUT: out operand;

-- second operand

D2OUT: out operand;

-- data to write

DWIN: in operand

);

end MRAM;

architecture Beh\_GPR of MRAM is

type tRAM is array (0 to 31) of operand;

signal RAM: tRAM:= (

-- | VALUE BIN | ADR BIN | CONST NAME |

"0000000000000000", -- | 00000 | STUB, ADDR\_DATA\_START |

"0000000000000000", -- | 00001 | |

"0000000000000000", -- | 00010 | |

"0000000000000000", -- | 00011 | |

"0000000000000000", -- | 00100 | |

"0000000000000000", -- | 00101 | |

"0000000000000000", -- | 00110 | |

"0000000000000000", -- | 00111 | |

"0000000000000000", -- | 01000 | |

"0000000000000000", -- | 01001 | |

"0000000000000000", -- | 01010 | |

"0000000000000000", -- | 01011 | |

"0000000000000000", -- | 01100 | |

"0000000000000000", -- | 01101 | |

"0000000000000000", -- | 01110 | |

"0000000000000000", -- | 01111 | |

"0000000000001000", -- | 10000 | ADDR\_LENGTH |

"0000000000000000", -- | 10001 | ADDR\_COUNTER |

"0000000000000000", -- | 10010 | ADDR\_CURRENT\_VALUE |

"0000000000000001", -- | 10011 | ADDR\_INIT\_VALUE |

"0000000000000000", -- | 10100 | ADDR\_ZERO |

"0000000000000001", -- | 10101 | ADDR\_ONE |

"0000000000000000", -- | 10110 | EMPTY\_SPACE |

others => "0000000000000000"

);

signal data\_win: operand;

signal data\_1out: operand;

signal data\_2out: operand;

Begin

data\_win <= DWIN;

WRITE: process(CLK)

begin

if (rising\_edge(CLK)) then

if (RW = '0') then

RAM(conv\_integer(ADDRW)) <= data\_win;

end if;

end if;

end process;

data\_1out <= RAM (conv\_integer(ADDR1));

data\_2out <= RAM (conv\_integer(ADDR2));

READ: process(CLK)

begin

if (rising\_edge(CLK)) then

if (RW = '1') then

D1OUT <= data\_1out;

D2OUT <= data\_2out;

else

D1OUT <= (others => 'Z');

D2OUT <= (others => 'Z');

end if;

end if;

end process;

End Beh\_GPR;

* + 1. **TestBench**

library gpr;

use gpr.OneHotGPR.all;

library ieee;

use ieee.STD\_LOGIC\_UNSIGNED.all;

use ieee.std\_logic\_1164.all;

entity mram\_tb is

end mram\_tb;

architecture TB\_ARCHITECTURE of mram\_tb is

component mram

port(

RW : in STD\_LOGIC;

CLK : in STD\_LOGIC;

ADDR1 : in mem\_addr;

ADDR2 : in mem\_addr;

ADDRW : in mem\_addr;

D1OUT : out operand;

D2OUT : out operand;

DWIN : in operand );

end component;

signal RW : STD\_LOGIC;

signal CLK : STD\_LOGIC;

signal ADDR1 : mem\_addr;

signal ADDR2 : mem\_addr;

signal ADDRW : mem\_addr;

signal DWIN : operand;

signal D1OUT : operand;

signal D2OUT : operand;

constant CLK\_period: time := 10 ns;

begin

-- Unit Under Test port map

UUT : mram

port map (

RW => RW,

CLK => CLK,

ADDR1 => ADDR1,

ADDR2 => ADDR2,

ADDRW => ADDRW,

D1OUT => D1OUT,

D2OUT => D2OUT,

DWIN => DWIN

);

CLK\_Process: process

begin

CLK <= '0';

wait for CLK\_Period/2;

CLK <= '1';

wait for CLK\_Period/2;

end process;

main: process

begin

wait for CLK\_PERIOD;

addrw <= "00010";

dwin <= "0000000000000100";

wait for CLK\_PERIOD;

addr1 <= "00001";

addr2 <= "00010";

rw <= '1';

wait for CLK\_PERIOD;

addrw <= "00000";

dwin <= "0000000000010000";

rw <= '0';

wait for CLK\_PERIOD;

wait;

end process;

end TB\_ARCHITECTURE;

configuration TESTBENCH\_FOR\_mram of mram\_tb is

for TB\_ARCHITECTURE

for UUT : mram

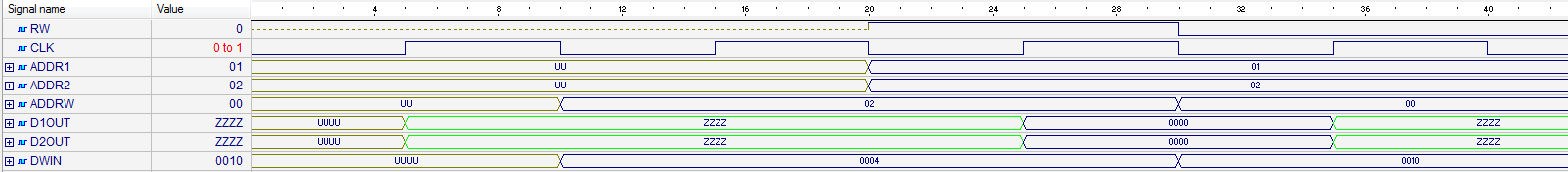
use entity work.mram(beh\_gpr);

end for;

end for;

end TESTBENCH\_FOR\_mram;

* + 1. **Временные диаграммы функционирования**



* 1. Блок CTRL
     1. **HDL**

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

use IEEE.STD\_LOGIC\_UNSIGNED.all;

library gpr;

use gpr.OneHotGPR.all;

entity CTRL1 is

port(

CLK, RST, Start: in std\_logic;

Stop: out std\_logic;

ROM\_re: out std\_logic;

ROM\_addr: out mem\_addr;

ROM\_dout: in command;

RAM\_rw: out std\_logic;

RAM\_addr1: out mem\_addr;

RAM\_addr2: out mem\_addr;

RAM\_addrw: out mem\_addr;

RAM\_dwin: out operand;

RAM\_d1out: in operand;

RAM\_d2out: in operand;

--datapath

DP\_op1: out operand;

DP\_op2: out operand;

DP\_ot: out operation;

DP\_en: out std\_logic;

DP\_res: in operand;

DP\_zf: in std\_logic

);

end CTRL1;

architecture Beh\_GPR of CTRL1 is

type states is (I, F, D, R, W, A, SB, SH, H, M, CP, CPR, JNOTZ);

signal nxt\_state, cur\_state: states;

-- instruction register

signal RI: std\_logic\_vector(17 downto 0);

-- instruction counter

signal IC: mem\_addr;

-- operation type register

signal RO: std\_logic\_vector(2 downto 0);

-- memory address register for the first operand

signal RA\_1: mem\_addr;

-- memory address register for the second operand

signal RA\_2: mem\_addr;

-- memory address register for the result

signal RA\_W: mem\_addr;

-- data register for the first operand

signal RD\_1: operand;

-- data register for the second operand

signal RD\_2: operand;

-- data register for the result

signal RD\_W: operand;

begin

-- synchronous memory

FSM: process(CLK, RST, nxt\_state)

begin

if (RST = '1') then

cur\_state <= I;

elsif rising\_edge(CLK) then

cur\_state <= nxt\_state;

end if;

end process;

-- Next state

COMB: process(cur\_state, start, RO)

begin

case cur\_state is

when I =>

if (start = '1') then

nxt\_state <= F;

else

nxt\_state <= I;

end if;

when F => nxt\_state <= D;

when D =>

if (RO = HALT) then

nxt\_state <= H;

elsif (RO = JNZ) then

nxt\_state <= JNOTZ;

else

nxt\_state <= R;

end if;

when R =>

if (RO = ADD) then

nxt\_state <= A;

elsif (RO = SUBT) then

nxt\_state <= SB;

elsif (RO = SHIFT) then

nxt\_state <= SH;

elsif (RO = COPY) then

nxt\_state <= CP;

else

nxt\_state <= I;

end if;

when CP => nxt\_state <= CPR;

when CPR => nxt\_state <= M;

when A | SB | SH | M => nxt\_state <= W;

when W | JNOTZ => nxt\_state <= F;

when H => nxt\_state <= H;

when others => nxt\_state <= I;

end case;

end process;

-- stop signal handler

PSTOP: process (cur\_state)

begin

if (cur\_state = H) then

stop <= '1';

else

stop <= '0';

end if;

end process;

-- instruction counter

PMC: process (CLK, RST, cur\_state)

begin

if (RST = '1') then

IC <= (others => '0');

elsif falling\_edge(CLK) then

if (cur\_state = D) then

IC <= IC + 1;

elsif (cur\_state = JNOTZ and DP\_ZF = '0') then

IC <= RA\_1;

end if;

end if;

end process;

ROM\_addr <= IC;

-- reading from ROM

PROMREAD: process (nxt\_state, cur\_state)

begin

if (nxt\_state = F or cur\_state = F) then

ROM\_re <= '1';

else

ROM\_re <= '0';

end if;

end process;

-- reading the instruction from ROM and put it into RI

PROMDAT: process (RST, cur\_state, ROM\_dout)

begin

if (RST = '1') then

RI <= (others => '0');

elsif (cur\_state = F) then

RI <= ROM\_dout;

end if;

end process;

-- fill RO and RA\_1, RA\_2, RA\_W registers

PRORA: process (RST, nxt\_state, RI)

begin

if (RST = '1') then

RO <= (others => '0');

RA\_1 <= (others => '0');

RA\_2 <= (others => '0');

RA\_W <= (others => '0');

elsif (nxt\_state = D) then

RO <= RI (17 downto 15);

RA\_1 <= RI (14 downto 10);

RA\_2 <= RI (9 downto 5);

RA\_W <= RI (4 downto 0);

elsif (nxt\_state = CP) then

RA\_2 <= RD\_2 (4 downto 0);

elsif (nxt\_state = CPR) then

RA\_W <= RA\_2;

end if;

end process;

PRAMST: process (RA\_1, RA\_2, RA\_W)

begin

if (cur\_state /= JNOTZ) then

RAM\_addr1 <= RA\_1;

RAM\_addr2 <= RA\_2;

RAM\_addrw <= RA\_W;

end if;

end process;

-- control read/write signal for RAM

PRAMREAD: process (cur\_state)

begin

if (cur\_state = W) then

RAM\_rw <= '0';

else

RAM\_rw <= '1';

end if;

end process;

-- read values from RAM and store them in RD\_1 and RD\_2 registers

PRAMDAR: process (cur\_state)

begin

if (cur\_state = R) then

RD\_1 <= RAM\_d1out;

RD\_2 <= RAM\_d2out;

end if;

end process;

-- pass the value from data path to RAM data bus

RAM\_dwin <= DP\_res;

-- pass the value from RD\_1 register to the data path first operand

DP\_op1 <= RD\_1;

-- pass the value from RD\_2 register to the data path second operand

DP\_op2 <= RD\_2;

-- pass the value from RO register to the data path operation type

DP\_ot <= RO;

paddsuben: process (cur\_state)

begin

if (cur\_state = A or cur\_state = SB or cur\_state = SH or cur\_state = M) then

DP\_en <= '1';

else

DP\_en <= '0';

end if;

end process;

end Beh\_GPR;

* + 1. **TestBench**

library accum;

use accum.OneHotAccum.all;

library ieee;

use ieee.STD\_LOGIC\_UNSIGNED.all;

use ieee.std\_logic\_1164.all;

entity ctrl1\_tb is

end ctrl1\_tb;

architecture TB\_ARCHITECTURE of ctrl1\_tb is

component MROM is

port (

RE: in std\_logic;

ADDR: in mem\_addr;

DOUT: out command

);

end component;

component MRAM is

port (

CLK: in std\_logic;

RW: in std\_logic;

ADDR: in mem\_addr;

DIN: in operand;

DOUT: out operand

);

end component;

component DPATH is

port(

EN: in std\_logic;

OT: in operation;

OP1 : in operand;

RES: out operand;

ZF: out std\_logic

);

end component;

component ctrl1

port(

CLK : in STD\_LOGIC;

RST : in STD\_LOGIC;

Start : in STD\_LOGIC;

Stop : out STD\_LOGIC;

ROM\_re : out STD\_LOGIC;

ROM\_addr : out mem\_addr;

ROM\_dout : in command;

RAM\_rw : out STD\_LOGIC;

RAM\_addr : out mem\_addr;

RAM\_din : out operand;

RAM\_dout : in operand;

DP\_op1 : out operand;

DP\_ot : out operation;

DP\_en : out STD\_LOGIC;

DP\_res : in operand;

DP\_zf : in STD\_LOGIC );

end component;

signal CLK : STD\_LOGIC;

signal RST : STD\_LOGIC;

signal Start : STD\_LOGIC;

signal ROM\_dout : command;

signal RAM\_dout : operand;

signal DP\_res : operand;

signal DP\_zf : STD\_LOGIC;

signal Stop : STD\_LOGIC;

signal ROM\_re : STD\_LOGIC;

signal ROM\_addr : mem\_addr;

signal RAM\_rw : STD\_LOGIC;

signal RAM\_addr : mem\_addr;

signal RAM\_din : operand;

signal DP\_op1 : operand;

signal DP\_ot : operation;

signal DP\_en : STD\_LOGIC;

constant CLK\_period: time := 10 ns;

begin

UMRAM: MRAM

port map(

CLK => CLK,

RW => ram\_rw,

ADDR => ram\_addr,

DIN => ram\_din,

DOUT => ram\_dout

);

UMROM: MROM

port map (

RE => rom\_re,

ADDR => rom\_addr,

DOUT => rom\_dout

);

UDPATH: DPATH

port map(

EN => dp\_en,

OT => dp\_ot,

OP1 => dp\_op1,

RES => dp\_res,

ZF => dp\_zf

);

UCTRL1: CTRL1

port map(

CLK => CLK,

RST => RST,

START => Start,

STOP => STOP,

ROM\_RE => rom\_re,

ROM\_ADDR => rom\_addr,

ROM\_DOUT => rom\_dout,

RAM\_RW => ram\_rw,

RAM\_ADDR => ram\_addr,

RAM\_DIN => ram\_din,

RAM\_DOUT => ram\_dout,

DP\_EN => dp\_en,

DP\_OT => dp\_ot,

DP\_OP1 => dp\_op1,

DP\_RES => dp\_res,

DP\_ZF => dp\_zf

);

UUT : ctrl1

port map (

CLK => CLK,

RST => RST,

Start => Start,

Stop => Stop,

ROM\_re => ROM\_re,

ROM\_addr => ROM\_addr,

ROM\_dout => ROM\_dout,

RAM\_rw => RAM\_rw,

RAM\_addr => RAM\_addr,

RAM\_din => RAM\_din,

RAM\_dout => RAM\_dout,

DP\_op1 => DP\_op1,

DP\_ot => DP\_ot,

DP\_en => DP\_en,

DP\_res => DP\_res,

DP\_zf => DP\_zf

);

CLK\_Process: process

begin

CLK <= '0';

wait for CLK\_Period/2;

CLK <= '1';

wait for CLK\_Period/2;

end process;

main: process

begin

rst <= '1';

wait for CLK\_PERIOD;

rst <= '0';

start <= '1';

wait for 100 \* CLK\_PERIOD;

wait;

end process;

end TB\_ARCHITECTURE;

configuration TESTBENCH\_FOR\_ctrl1 of ctrl1\_tb is

for TB\_ARCHITECTURE

for UUT : ctrl1

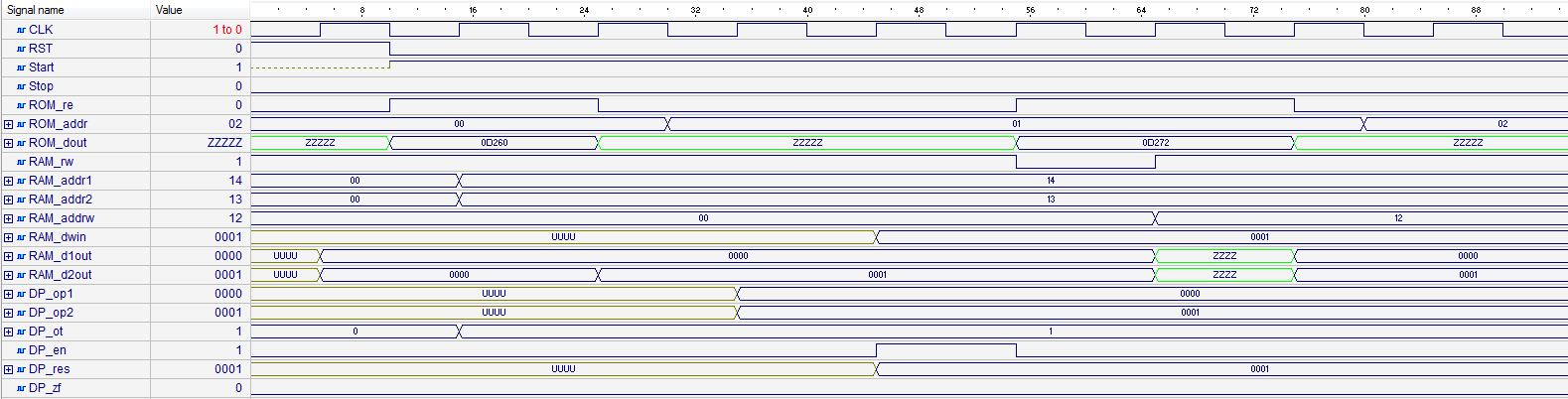
use entity work.ctrl1(beh);

end for;

end for;

end TESTBENCH\_FOR\_ctrl1;

* + 1. **Временные диаграммы функционирования**



* 1. Блок DPATH
     1. **HDL**

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

use IEEE.STD\_LOGIC\_ARITH.all;

use IEEE.STD\_LOGIC\_UNSIGNED.all;

library gpr;

use gpr.OneHotGPR.all;

entity DPATH is

port(

EN: in std\_logic;

OT: in operation;

OP1: in operand;

OP2: in operand;

RES: out operand;

ZF: out std\_logic

);

end DPATH;

architecture Beh\_GPR of DPATH is

signal res\_g: operand;

signal res\_add: operand;

signal res\_sub: operand;

signal res\_shift: operand;

signal res\_copy: operand;

signal t\_zf: std\_logic;

Begin

res\_add <= CONV\_STD\_LOGIC\_VECTOR(CONV\_INTEGER(OP1) + CONV\_INTEGER(OP2), 16);

res\_sub <= CONV\_STD\_LOGIC\_VECTOR(CONV\_INTEGER(OP1) - CONV\_INTEGER(OP2), 16);

res\_shift <= OP1(14 downto 0) & '0';

res\_copy <= OP1;

REGA: process (EN, OT, OP1, res\_add, res\_sub, res\_shift, res\_copy)

begin

if rising\_edge(EN) then

case OT is

when ADD => res\_g <= res\_add;

when SUBT => res\_g <= res\_sub;

when SHIFT => res\_g <= res\_shift;

when COPY => res\_g <= res\_copy;

when others => null;

end case;

end if;

end process;

FLAGS: process(res\_g)

begin

if res\_g = (res\_g'range => '0') then

t\_zf <= '1';

else

t\_zf <= '0';

end if;

end process;

RES <= res\_g;

ZF <= t\_zf;

End Beh\_GPR;

* + 1. **TestBench**

library gpr;

use gpr.OneHotGPR.all;

library ieee;

use ieee.STD\_LOGIC\_UNSIGNED.all;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_arith.all;

entity dpath\_tb is

end dpath\_tb;

architecture TB\_ARCHITECTURE of dpath\_tb is

component dpath

port(

EN : in STD\_LOGIC;

OT : in operation;

OP1 : in operand;

OP2 : in operand;

RES : out operand;

ZF : out STD\_LOGIC );

end component;

signal EN : STD\_LOGIC;

signal OT : operation;

signal OP1 : operand;

signal OP2 : operand;

signal RES : operand;

signal ZF : STD\_LOGIC;

constant WAIT\_period: time := 10 ns;

begin

UUT : dpath

port map (

EN => EN,

OT => OT,

OP1 => OP1,

OP2 => OP2,

RES => RES,

ZF => ZF

);

MAIN: process

begin

en <= '1';

wait for WAIT\_period;

en <= '0';

op1 <= "0000000000000010";

op2 <= "0000000000000010";

ot <= ADD;

wait for WAIT\_period;

en <= '1';

wait for WAIT\_period;

en <= '0';

op1 <= "0000000000000010";

op2 <= "0000000000000010";

ot <= SUBT;

wait for WAIT\_period;

en <= '1';

wait for WAIT\_period;

en <= '0';

op1 <= "0000000000010010";

ot <= SHIFT;

wait for WAIT\_period;

en <= '1';

wait for WAIT\_period;

en <= '0';

op1 <= "0000000001010010";

ot <= COPY;

wait for WAIT\_period;

en <= '1';

wait for WAIT\_period;

en <= '0';

wait for WAIT\_period;

wait;

end process;

end TB\_ARCHITECTURE;

configuration TESTBENCH\_FOR\_dpath of dpath\_tb is

for TB\_ARCHITECTURE

for UUT : dpath

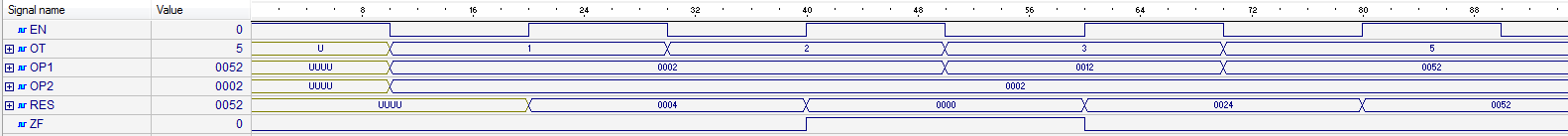
use entity work.dpath(beh\_gpr);

end for;

end for;

end TESTBENCH\_FOR\_dpath;

* + 1. **Временные диаграммы функционирования**



1. **System Level**
   1. **HDL**

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

use IEEE.STD\_LOGIC\_UNSIGNED.all;

library accum;

use accum.OneHotAccum.all;

entity OneHot is

port (

CLK, RST, Start: in std\_logic;

Stop: out std\_logic

);

end OneHot;

architecture Beh of OneHot is

component MROM is

port (

RE: in std\_logic;

ADDR: in mem\_addr;

DOUT: out command

);

end component;

component MRAM is

port (

CLK: in std\_logic;

RW: in std\_logic;

ADDR: in mem\_addr;

DIN: in operand;

DOUT: out operand

);

end component;

component DPATH is

port(

EN: in std\_logic;

OT: in operation;

OP1 : in operand;

RES: out operand;

ZF: out std\_logic

);

end component;

component CTRL1 is

port(

CLK, RST, Start: in std\_logic;

Stop: out std\_logic;

ROM\_re: out std\_logic;

ROM\_addr: out mem\_addr;

ROM\_dout: in command;

RAM\_rw: out std\_logic;

RAM\_addr: out mem\_addr;

RAM\_din: out operand;

RAM\_dout: in operand;

DP\_op1: out operand;

DP\_ot: out operation;

DP\_en: out std\_logic;

DP\_res: in operand;

DP\_zf: in std\_logic

);

end component;

signal rom\_re: std\_logic;

signal rom\_addr: mem\_addr;

signal rom\_dout: command;

signal ram\_rw: std\_logic;

signal ram\_addr: mem\_addr;

signal ram\_din: operand;

signal ram\_dout: operand;

signal dp\_op1: operand;

signal dp\_ot: operation;

signal dp\_en: std\_logic;

signal dp\_res: operand;

signal dp\_zf: std\_logic;

begin

UMRAM: MRAM

port map(

CLK => CLK,

RW => ram\_rw,

ADDR => ram\_addr,

DIN => ram\_din,

DOUT => ram\_dout

);

UMROM: MROM

port map (

RE => rom\_re,

ADDR => rom\_addr,

DOUT => rom\_dout

);

UDPATH: DPATH

port map(

EN => dp\_en,

OT => dp\_ot,

OP1 => dp\_op1,

RES => dp\_res,

ZF => dp\_zf

);

UCTRL1: CTRL1

port map(

CLK => CLK,

RST => RST,

START => Start,

STOP => STOP,

ROM\_RE => rom\_re,

ROM\_ADDR => rom\_addr,

ROM\_DOUT => rom\_dout,

RAM\_RW => ram\_rw,

RAM\_ADDR => ram\_addr,

RAM\_DIN => ram\_din,

RAM\_DOUT => ram\_dout,

DP\_EN => dp\_en,

DP\_OT => dp\_ot,

DP\_OP1 => dp\_op1,

DP\_RES => dp\_res,

DP\_ZF => dp\_zf

);

end Beh;

* 1. **Test Bench**

library ieee;

use ieee.STD\_LOGIC\_UNSIGNED.all;

use ieee.std\_logic\_1164.all;

entity onehot\_tb is

end onehot\_tb;

architecture TB\_ARCHITECTURE of onehot\_tb is

component onehot

port(

CLK : in STD\_LOGIC;

RST : in STD\_LOGIC;

Start : in STD\_LOGIC;

Stop : out STD\_LOGIC );

end component;

signal CLK : STD\_LOGIC;

signal RST : STD\_LOGIC;

signal Start : STD\_LOGIC;

signal Stop : STD\_LOGIC;

constant CLK\_period: time := 10 ns;

begin

UUT : onehot

port map (

CLK => CLK,

RST => RST,

Start => Start,

Stop => Stop

);

CLK\_Process: process

begin

CLK <= '0';

wait for CLK\_Period/2;

CLK <= '1';

wait for CLK\_Period/2;

end process;

main: process

begin

rst <= '1';

wait for 1 \* CLK\_PERIOD;

rst <= '0';

start <= '1';

wait for 100 \* CLK\_PERIOD;

wait;

end process;

end TB\_ARCHITECTURE;

configuration TESTBENCH\_FOR\_onehot of onehot\_tb is

for TB\_ARCHITECTURE

for UUT : onehot

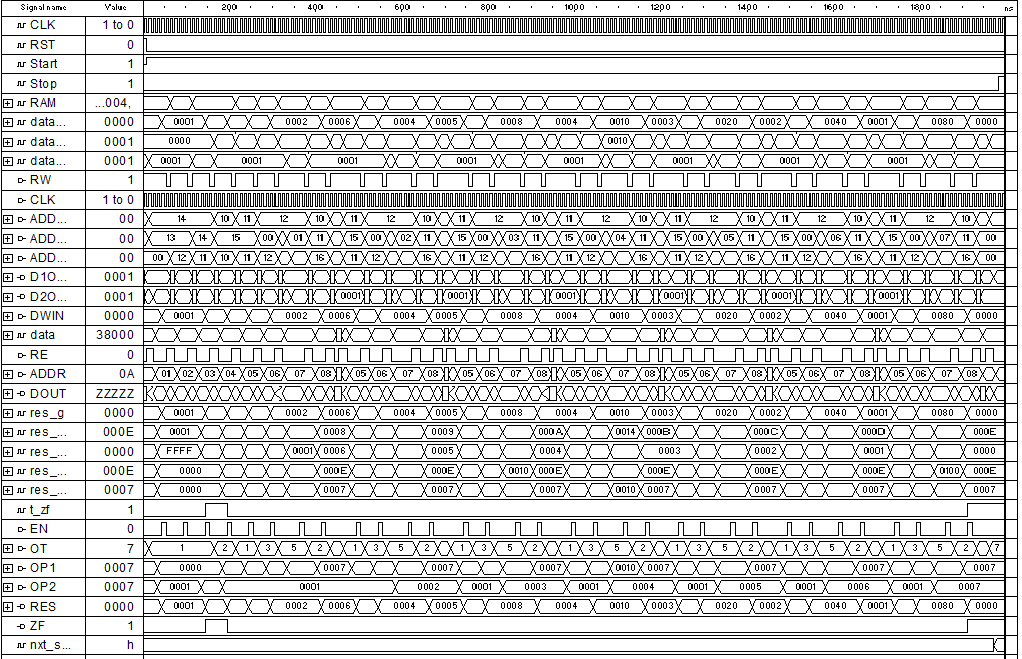
use entity work.onehot(beh);

end for;

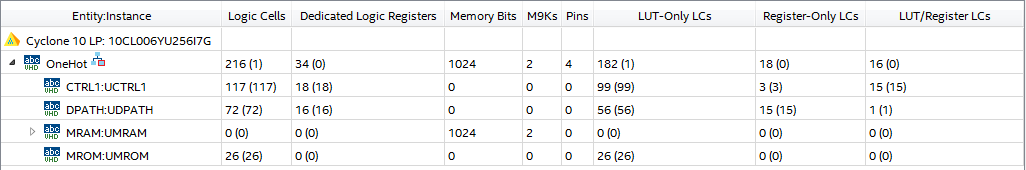
end for;

end TESTBENCH\_FOR\_onehot;

* 1. **Временные диаграммы функционирования**



1. **Анализ аппаратурных и временные затрат**
   1. Аппаратные затраты (Cyclone 10CL006YU256I7G / Inter Quartus Prime 17.1 Lite)



* 1. Временные затраты

Для генерации 8 элементов “One-Hot” последовательности требуется 198 тактов синхронизации.

1. Выводы