```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;
entity FSM_MNvcS2 is
generic(
                nandBUSwidth: natural := 8;
                ErrorFetchCommand: std_logic_vector(7 downto 0) := x"23"
                );
        port(
                -- Debug signals
                DebugCURaddreg: out std_logic;
                DebugInternalCNT: out std_logic_vector(23 downto 0);

                -- Standard synchronous inputs
                clk : in std_logic;
                Reset_L : in std_logic;

                -- STANDARD SRAM interface
                ADDR : in std_logic_vector(11 downto 0);
                ST2outDATA : out std_logic_vector(7 downto 0);
                ST2inDATA : in std_logic_vector(7 downto 0);
                CE_L : in std_logic;
```

```
WE_L : in std_logic;

OE_L : in std_logic;

INT : out std_logic;

RB_L : out std_logic;


-- NAND or NAND ecc module interface

nandCE_L : out std_logic;

nandCLE_H : out std_logic;

nandALE_H : out std_logic;

nandWE_L : out std_logic;

nandRE_L : out std_logic;

nandWP_L : out std_logic;

nandPRE : out std_logic;

nandDATA : inout std_logic_vector(7 downto 0);

nandRB_L : in std_logic;

errINT : in std_logic;


-- Control signals


-- REGfile signals

--WE_regfile_H: out std_logic;

--REGnum: out std_logic_vector(3 downto 0);

--REGfile_DATA: out std_logic_vector(7 downto 0);

--REGfile_DATAout: in std_logic_vector(7 downto 0);


-- BUFFER1 signals

WE_buffer1_H: out std_logic;
```

```
		buffer1_ADDR: out std_logic_vector(11 downto 0);
		buffer1_DATA: out std_logic_vector(7 downto 0);
		buffer1_DATAout: in std_logic_vector(7 downto 0);

		-- BUFFER2 signals
		WE_buffer2_H: out std_logic;
		buffer2_ADDR: out std_logic_vector(11 downto 0);
		buffer2_DATA: out std_logic_vector(7 downto 0);
		buffer2_DATAout: in std_logic_vector(7 downto 0);

		-- disabling ECC module signal
		enableECCmodule: out std_logic
		);
	end FSM_MNvcS2;

	architecture syn of FSM_MNvcS2 is

		type state is (Start, enableTOG0, STreadstatusCMD, STreadstatusREAD, STreadID,
STreadIDadd00,
				STreadIDfourReads, STeraseblock, STeraseblockADDcycles,
STeraseblockCMDd0,
				STresetNAND, STreadpage, STreadpageADDlatch00,
STreadpageADDlatch3cycles,
				STreadpageCMDfinal30h, STreadpageREADS,
STreadpageSWAPbuf, STwaitforRB0,
				STwaitforRB1, STprogrampage, STprogrampageADDlatch00,
STprogrampageADDlatch3cycles,
				STprogrampageWRITES, STprogrampageCMDfinal10h,
STprogrampageCHbuf,
```

```
                                        STprogrampageSWstatusRB, STfetchERRORs,
STfetchERRORseightReads,        STreadpageCORwaitforRB,
                                        STreadpageCORRECT, STreadpageCORRECTaddr);

        type command is (ReadID, pageREAD, programPAGE, ReadSTATUS, erasepage, resetnand);


        -- state signals for state machine
        signal Cstate : state;
        signal Nstate : state;
        signal NEXTtonextST, NEXTtonextST2: state;
        signal internal_nandDATA: std_logic_vector(nandBUSwidth - 1 downto 0);

        -- FSM mux select signals
        signal buffer1_nandDATA_in, buffer2_nandDATA_in, nandDATA_from_buffer1,
nandDATA_from_buffer2, nandDATA_fsm : std_logic;
        signal outputVEC1, outputVEC2: std_logic_vector(4 downto 0);

        -- Repetition counter signals
        signal internalCNT, CNTupto, internalCNTaddr: std_logic_vector(11 downto 0);
        signal internalCNTRst_L, incintCNT, OnedelayedINCcnt : std_logic;

        -- Toggle state machine stuff
        signal enableTOGGLE, doneTOGGLE: std_logic;
        type TOGstate is (toggleWAIT, toggle1, toggle2, toggleDONE);
        signal cTOGstate: TOGstate;
        signal delayCNT: std_logic_vector(2 downto 0);
        signal incDcnt, rstDcnt_L: std_logic;
        signal nTOGstate: TOGstate;
```

```
-- System attributes i.e. Register FILE registers
signal Device8_not16, ECCpresent, buffer1or2: std_logic;
signal Device_Size:std_logic_vector(1 downto 0);
signal CommandREG, StatusREG: std_logic_vector(7 downto 0);

-- Error location array
type errorLOCtype is array (7 downto 0) of std_logic_vector(7 downto 0);
signal errorLOC: errorLOCtype;
type nandIDtype is array (3 downto 0) of std_logic_vector(7 downto 0);
type regADDRtype is array (2 downto 0) of std_logic_vector(7 downto 0);
type DOUBLEregADDRtype is array (1 downto 0) of regADDRtype;
signal nandID: nandIDtype;
signal REGaddress: DOUBLEregADDRtype;
signal currentADDreg, flipTHEt: std_logic;
signal DEBUGstatevector: std_logic_vector(7 downto 0);
signal nandRB_Lreg1, nandRB_Lfinal: std_logic;
--signal ADDressaddr: std_logic_vector(1 downto 0);

-- signals for error correction
signal errCORaddrIN, corWE: std_logic;
signal corINPUTdata: std_logic_vector(7 downto 0);
signal corADDR: std_logic_vector(11 downto 0);

-- Constants based on control registers
signal bytesCONTROLLED: std_logic_vector(11 downto 0);
```

```
	begin
		punchOUTconstants: process(ECCpresent)

	begin
		if(ECCpresent = '0') then
			bytesCONTROLLED <= x"840";

			else
				bytesCONTROLLED <= x"834";
		end if;

	end process punchOUTconstants;

doublebufnandRB_L:process(clk)

	begin
		if(rising_edge(clk)) then
			nandRB_Lreg1 <= nandRB_L;
			nandRB_Lfinal <= nandRB_Lreg1;
		end if;
	end process;
	DebugCURaddreg <= OneDelayedINCcnt;

	--DebugInternalCNT <= internalCNTaddr; --DebugInternalCNT <= "0000" &
DEBUGstatevector;
	--DebugInternalCNT <= buffer1or2 & nandDATA_fsm & nandDATA_from_buffer2 &
nandDATA_from_buffer1 & DEBUGstatevector;
	--DebugInternalCNT <= delayCNT & nandDATA_from_buffer1 & DEBUGstatevector;
```

```
	DebugInternalCNT <= corADDR & corWE & errCORaddrIN & errINT & nandRB_L &
DEBUGstatevector;

	TflipFLOPprocess:process(clk, Reset_L, flipTHEt, currentADDreg)


	begin

		if(rising_edge(clk)) then

			if(reset_L = '0') then

			currentADDreg <= '0';

			else

				if(flipTHEt = '1') then

					currentADDreg <= not(currentADDreg);

				end if;

			end if;

		end if;


end process TflipFLOPprocess;


enableECCmodule <= ECCpresent;

 statemachine: process(clk, Reset_L, cSTATE, nandRB_Lfinal, buffer1or2, buffer2_DATAout,
buffer1_DATAout)

	variable nandaddress1, nandaddress2, nandaddress3: std_logic_vector(7 downto 0);

	-- Format

	-- bit 0 - CE_L

	-- bit 1 - CLE_H

	-- bit 2 - ALE_H

	-- bit 3 - WE_L

	-- bit 4 - RE_L
```

```
	variable currentCMD: command;
	variable VARcorINPUTdata: std_logic_vector(7 downto 0);
	variable outputFROMbuf: std_logic_vector(7 downto 0);

	begin
		if (buffer1or2 = '0') then
			outputFROMbuf := buffer2_DATAout;
		else
			outputFROMbuf := buffer1_DATAout;
		end if;
			case errorLOC(to_integer(unsigned(internalCNT(1 downto 0) & '0')))(2
downto 0) is
		when "000" =>
			VARcorINPUTdata := outputFROMbuf(7 downto 1) &
not(outputFROMbuf(0));
		when "001" =>
			VARcorINPUTdata := outputFROMbuf(7 downto 2) & not(outputFROMbuf(1))
& outputFROMbuf(0);
		when "010" =>
			VARcorINPUTdata := outputFROMbuf(7 downto 3) & not(outputFROMbuf(2))
& outputFROMbuf(1 downto 0);
		when "011" =>
			VARcorINPUTdata := outputFROMbuf(7 downto 4) & not(outputFROMbuf(3))
& outputFROMbuf(2 downto 0);
		when "100" =>
			VARcorINPUTdata := outputFROMbuf(7 downto 5) & not(outputFROMbuf(4))
& outputFROMbuf(3 downto 0);
		when "101" =>
			VARcorINPUTdata := outputFROMbuf(7 downto 6) & not(outputFROMbuf(5))
& outputFROMbuf(4 downto 0);
```

```
                when "110" =>
                        VARcorINPUTdata := outputFROMbuf(7) & not(outputFROMbuf(6)) &
outputFROMbuf(5 downto 0);
                when "111" =>
                        VARcorINPUTdata := not(outputFROMbuf(7)) & outputFROMbuf(6 downto
0);
                when others =>
                        VARcorINPUTdata := outputFROMbuf;
                end case;

        if(rising_edge(clk)) then
                if(Reset_L = '0') then
                Cstate <= Start;
                NEXTtonextST <= Start;
                CNTupto <= x"000";
                internalCNTaddr <= x"000";

                -- New REGfile handling
                nandID <= (x"00", x"00", x"00", x"00");
                REGaddress <= ((x"00", x"00", x"00"), (x"00", x"00", x"00"));
                errorLOC <= (x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00");
                Device8_not16 <= '0';
                ECCpresent <= '0';
                buffer1or2 <= '0';
                Device_Size <= "00";
                CommandREG <= x"00";
                nandPRE <= '1';
        else
```

```
		nandPRE <= '1';
		if(ADDR = x"FF4" and WE_L = '0' and CE_L = '0') then
			REGaddress(to_integer(unsigned(currentADDreg)))(0) <= ST2inDATA;
		elsif(ADDR = x"FF5" and WE_L = '0' and CE_L = '0') then
			REGaddress(conv_integer(currentADDreg))(1) <= ST2inDATA;
		elsif(ADDR = x"FF6" and WE_L = '0' and CE_L = '0') then
			REGaddress(conv_integer(currentADDreg))(2) <= ST2inDATA;
		elsif(ADDR = x"FF7" and WE_L = '0' and CE_L = '0') then
			Device8_not16 <= ST2inDATA(0);
			Device_Size <= ST2inDATA(2 downto 1);
			ECCpresent <= ST2inDATA(6);
		elsif(ADDR = x"FF8" and WE_L = '0' and CE_L = '0') then
			buffer1or2 <= ST2inDATA(0);
		elsif(ADDR = x"FFA" and WE_L = '0' and CE_L = '0') then
			CommandREG <= ST2inDATA;
		end if;
		internalCNTaddr <= internalCNT;
		case cSTATE is
			when Start =>
				NEXTtonextST <= Start;
				cntUPTO <= x"000";
				NEXTtonextST2 <= Start;

-- Multiple Use states
when enableTOG0 =>
		cntUPTO <= x"000";
when STwaitforRB0 =>
```

```
            cntUPTO <= x"000";
when STwaitforRB1 =>
            cntUPTO <= x"000";

-- Program cycles
when STprogrampageCHbuf =>
            StatusREG(5) <= '0';
            if(CommandREG(0) = '1') then
                        buffer1or2 <= not(CommandREG(1));
            else
                        buffer1or2 <= not(buffer1or2);
            end if;
            NEXTtonextST <= STprogrampage;
            when STprogrampage =>
                        NEXTtonextST <= STprogrampageADDlatch00;
            when STprogrampageADDlatch00 =>
                        cntUPTO <= x"002";
                        NEXTtonextST <= STprogrampageADDlatch3cycles;
            when STprogrampageADDlatch3cycles =>
                        cntUPTO <= x"003";
                        NEXTtonextST <= STprogrampageWRITES;
            when STprogrampageWRITES =>
                        NEXTtonextST <= STprogrampageCMDfinal10h;
                        --cntUPTO <= x"834";
                        cntUPTO <= bytesCONTROLLED;
                        when STprogrampageCMDfinal10h =>
                                    NEXTtonextST <= STprogrampageSWstatusRB;
```

```
                when STprogrampageSWstatusRB =>
                        StatusREG(5) <= '1';
                        cntUPTO <= x"000";
                        NEXTtonextST <= Start;

-- Read cycles
when STreadpage =>
        StatusREG(5) <= '0';
        if(CommandREG(0) = '1') then
                -- shift control to the buffer which is free
                -- that is not the one in the command
                buffer1or2 <= not(CommandREG(1));
        end if;
        NEXTtonextST <= STreadpageADDlatch00;
when STreadpageADDlatch00 =>
        cntUPTO <= x"002";
        NEXTtonextST <= STreadpageADDlatch3cycles;
when STreadpageADDlatch3cycles =>
        cntUPTO <= x"003";
        NEXTtonextST <= STreadpageCMDfinal30h;
when STreadpageCMDfinal30h =>
        NEXTtonextST <= STreadpageREADS;
when STreadpageREADS =>
        NEXTtonextST <= Start;
        cntUPTO <= bytesCONTROLLED;
        --cntUPTO <= x"834";
when STreadpageSWAPbuf =>
```

```
            StatusREG(5) <= '1';
            cntUPTO <= x"000";
            buffer1or2 <= not(buffer1or2);
when STreadpageCORwaitforRB =>
            NEXTtonextST2 <= STreadpageCORRECTaddr;
when STreadpageCORRECT =>
            NEXTtonextST2 <= STreadpageCORRECTaddr;
when STreadpageCORRECTaddr =>
            if (delayCNT = "001") then
                        corINPUTdata <= VARcorINPUTdata;
            end if;
            cntUPTO <= x"004";

-- Erase cycles
when STeraseblock =>
            StatusREG(5) <= '0';
            --internal_nandDATA <= x"60";
            NEXTtonextST <= STeraseblockADDcycles;
when STeraseblockADDcycles =>
            --internal_nandDATA <= REGfile_DATAout;
            cntUPTO <= x"003";
            NEXTtonextST <= STeraseblockCMDd0;
when STeraseblockCMDd0 =>
            NEXTtonextST <= STprogrampageSWstatusRB;
            --internal_nandDATA <= x"D0";
            cntUPTO <= x"000";
```

```
-- Reset cycles
when STresetNAND =>
        --internal_nandDATA <= x"FF";
        NEXTtonextST <= Start;
        cntUPTO <= x"000";

-- Fetch Error cycles
when STfetchERRORs =>
        --internal_nandDATA <= x"90";
        cntUPTO <= x"000";
        NEXTtonextST <= STfetchERRORseightReads;
when STfetchERRORseightReads =>
        if(incintCNT = '1') then
                errorLOC(conv_integer(internalCNT(2 downto 0))) <= nandDATA;
        end if;
        cntUPTO <= x"008";
        NEXTtonextST <= NEXTtonextST2;

-- Read ID cycles
when STreadID =>
        --internal_nandDATA <= x"90";
        cntUPTO <= x"000";
        NEXTtonextST <= STreadIDadd00;
when STreadIDadd00 =>
        --internal_nandDATA <= x"00";
        NEXTtonextST <= STreadIDfourReads;
        cntUPTO <= x"000";
```

```
when STreadIDfourReads =>
        if(incintCNT = '1') then
                nandID(conv_integer(internalCNT(1 downto 0))) <= nandDATA;
        end if;
        cntUPTO <= x"004";
        NEXTtonextST <= Start;

-- Read Status cycles
when STreadstatusCMD =>
        cntUPTO <= x"000";
        NEXTtonextST <= STreadstatusRead;
        --internal_nandDATA <= x"70";
when STreadstatusRead =>
        if(incintCNT = '1') then
                StatusREG <= nandDATA;
        end if;
        cntUPTO <= x"000";
when others =>
        end case;
        Cstate <= Nstate;
        end if;
end if;
end process statemachine;

-- FallingEDGE
-- This process is to generate the latch signal on
-- RE going high.
```

```
	FallingEDGE: process(clk, Reset_L)
	begin
		if(falling_edge(clk)) then
			if (Reset_L='0') then
				OnedelayedINCcnt <= '0';
			else
				OnedelayedINCcnt <= incintCNT;
			end if;
		end if;
	end process FallingEDGE;
	combFSMmain: process(cState, ADDR, we_L, ST2inDATA, doneTOGGLE, NEXTtonextST,
incintCNT, CE_L, buffer1or2, OE_L,
				internalCNT, nandRB_Lfinal, OnedelayedINCcnt, buffer1_DATAout,
buffer2_DATAout, delayCNT,
				nandRB_L, errINT, corWE, errorLOC)

begin
	-- STANDARD SRAM interface
	INT <= '0';
	RB_L <= '0';
	nandWP_L <= '1';

	-- BUFFER 1 and 2 WE signals
	WE_buffer1_H <= not(buffer1or2) and not(WE_L) and not(CE_L);
	WE_buffer2_H <= buffer1or2 and not(WE_L) and not(CE_L);

	-- Internal signals
	buffer1_nandDATA_in <= '0';
```

```
buffer2_nandDATA_in <= '0';
nandDATA_from_buffer1 <= '0';
nandDATA_from_buffer2 <= '0';

-- Toggle state machine stuff
enableTOGGLE <= '0';
nandDATA_fsm <= '0';
outputVEC1 <= "11001";
outputVEC2 <= "11001";
flipTHEt <= '0';
Nstate <= Start;
internal_nandDATA <= x"00";
errCORaddrIN <= '0';
corADDR <= x"000";
corWE <= '0';
DEBUGstatevector <= x"0A";
case cSTATE is
	when Start =>
		if(ADDR = x"FFA" and WE_L = '0' and CE_L = '0') then
			if(ST2inDATA(7 downto 4) = x"7") then
				-- Status read
				Nstate <= STreadstatusCMD;
			elsif(ST2inDATA(7 downto 4) = x"9") then
				-- Read ID
				Nstate <= STreadID;
			elsif(ST2inDATA(7 downto 4) = x"6") then
				-- Erase Block
```

```
                    Nstate <= STeraseblock;
                elsif(ST2inDATA(7 downto 4) = x"F") then
                    -- Reset NAND flash
                    Nstate <= STresetNAND;
                elsif(ST2inDATA(7 downto 4) = x"8") then
                    -- Program Page
                    Nstate <= STprogrampageCHbuf;
                elsif(ST2inDATA(7 downto 4) = x"0") then
                    -- Read Page
                    Nstate <= STreadpage;
                elsif(ST2inDATA(7 downto 0) = x"23") then
                    -- Fetch Errors
                    Nstate <= STfetchERRORS;
                else
                    Nstate <= Start;
            end if;
        else
            Nstate <= Start;
        end if;
        RB_L <= '1';
        DEBUGstatevector <= x"11";

        -- The common done toggle state
        when enableTOG0 =>
            nandDATA_fsm <= '0';
            enableTOGGLE <= '0';
            outputVEC1 <= "11001";
```

```
		outputVEC2 <= "11001";
		if (doneTOGGLE = '0') then
			Nstate <= NEXTtonextST;
		else
			Nstate <= enableTOG0;
		end if;
		DEBUGstatevector <= x"12";

	when STwaitforRB0 =>
		internal_nandDATA <= x"00";
		outputVEC1 <= "11001";
		outputVEC2 <= "11001";
		nandDATA_fsm <= '0';
		enableTOGGLE <= '0';
		if (nandRB_Lfinal = '0') then
			Nstate <= STwaitforRB1;
		else
			Nstate <= STwaitforRB0;
		end if;
		DEBUGstatevector <= x"13";

	when STwaitforRB1 =>
		internal_nandDATA <= x"00";
		outputVEC1 <= "11001";
		outputVEC2 <= "11001";
		nandDATA_fsm <= '0';
		enableTOGGLE <= '0';
```

```
		if (nandRB_Lfinal = '1') then
			Nstate <= NEXTtonextST;
		else
			Nstate <= STwaitforRB1;
		end if;

-- Program cycles
when STprogrampageCHbuf =>
		flipTHEt <= '1';
		internal_nandDATA <= x"00";
		outputVEC1 <= "10010";
		outputVEC2 <= "11010";
		nandDATA_fsm <= '0';
		enableTOGGLE <= '0';
		Nstate <= STprogrampage;

when STprogrampage =>
		internal_nandDATA <= x"80";
		outputVEC1 <= "10010";
		outputVEC2 <= "11010";
		nandDATA_fsm <= '1';
		enableTOGGLE <= '1';
		if (doneTOGGLE = '1') then
			Nstate <= enableTOG0;
		else
			Nstate <= STprogrampage;
		end if;
```

```
when STprogrampageADDlatch00 =>
        internal_nandDATA <= x"00";
        outputVEC1 <= "10100";
        outputVEC2 <= "11100";
        nandDATA_fsm <= '1';
        enableTOGGLE <= '1';
        if (doneTOGGLE = '1') then
                Nstate <= enableTOG0;
        else
                Nstate <= STprogrampageADDlatch00;
        end if;

when STprogrampageADDlatch3cycles =>
        outputVEC1 <= "10100";
        outputVEC2 <= "11100";
        nandDATA_fsm <= '1';
        enableTOGGLE <= '1';
        if (doneTOGGLE = '1') then
                Nstate <= enableTOG0;
        else
                Nstate <= STprogrampageADDlatch3cycles;
        end if;
        if(internalCNT(1 downto 0) = "11") then
                internal_nandDATA <= x"00";
        else
                internal_nandDATA <=
```

```
REGaddress(conv_integer(not(currentADDreg)))(conv_integer(internalCNT(1 downto 0)));
                end if;

        when STprogrampageWRITES =>
                outputVEC1 <= "10000";
                outputVEC2 <= "11000";
                nandDATA_fsm <= '0';
                enableTOGGLE <= '1';
                if (doneTOGGLE = '1') then
                        Nstate <= enableTOG0;
                else
                        Nstate <= STprogrampageWRITES;
                end if;
                if(buffer1or2 = '0') then
                        -- This means the user has buffer 1
                        -- So use buffer 2 as the buffers have been
                        -- swapped
                        internal_nandDATA <= buffer2_DATAout;
                        nandDATA_from_buffer2 <= '1';
                        else
                                internal_nandDATA <= buffer1_DATAout;
                                nandDATA_from_buffer1 <= '1';
                        end if;

        when STprogrampageCMDfinal10h =>
                internal_nandDATA <= x"10";
                outputVEC1 <= "10010";
```

```
		outputVEC2 <= "11010";
		nandDATA_fsm <= '1';
		enableTOGGLE <= '1';
		if (doneTOGGLE = '1') then
			Nstate <= STwaitforRB0;
		else
			Nstate <= STprogrampageCMDfinal10h;
		end if;

	when STprogrampageSWstatusRB =>
		Nstate <= Start;

	-- Read Cycles
	when STreadpage =>
		internal_nandDATA <= x"00";
		outputVEC1 <= "10010";
		outputVEC2 <= "11010";
		nandDATA_fsm <= '1';
		flipTHEt <= not(delayCNT(0) or delayCNT(1) or delayCNT(2));
		enableTOGGLE <= '1';
		if (doneTOGGLE = '1') then
			Nstate <= enableTOG0;
		else
			Nstate <= STreadpage;
		end if;

	when STreadpageADDlatch00 =>
```

```
                internal_nandDATA <= x"00";
                outputVEC1 <= "10100";
                outputVEC2 <= "11100";
                nandDATA_fsm <= '1';
                enableTOGGLE <= '1';
                if (doneTOGGLE = '1') then
                        Nstate <= enableTOG0;
                else
                        Nstate <= STreadpageADDlatch00;
                end if;

        when STreadpageADDlatch3cycles =>
                outputVEC1 <= "10100";
                outputVEC2 <= "11100";
                nandDATA_fsm <= '1';
                enableTOGGLE <= '1';
                if (doneTOGGLE = '1') then
                        Nstate <= enableTOG0;
                else
                        Nstate <= STreadpageADDlatch3cycles;
                end if;
                -- set the Register address on the lines
                if(internalCNT(1 downto 0) = "11") then
                        internal_nandDATA <= x"00";
                else
                        internal_nandDATA <=

REGaddress(conv_integer(not(currentADDreg)))(conv_integer(internalCNT(1 downto 0)));
```

```
		end if;

	when STreadpageCMDfinal30h =>
		internal_nandDATA <= x"30";
		outputVEC1 <= "10010";
		outputVEC2 <= "11010";
		nandDATA_fsm <= '1';
		enableTOGGLE <= '1';
		if (doneTOGGLE = '1') then
			Nstate <= STwaitforRB0;
		else
			Nstate <= STreadpageCMDfinal30h;
		end if;

	when STreadpageREADS =>
		internal_nandDATA <= x"00";
		outputVEC1 <= "01000";
		outputVEC2 <= "11000";
		nandDATA_fsm <= '0';
		enableTOGGLE <= '1';
		if (doneTOGGLE = '1') then
			Nstate <= STreadpageCORwaitforRB;
		else
			Nstate <= STreadpageREADS;
		end if;
		if(buffer1or2 = '0') then
			-- This means the user has buffer 1
```

```
                -- to keep the default we do this:-
                WE_buffer1_H <= not(WE_L) and not(CE_L);
                -- to write to the other buffer
                WE_buffer2_H <= OneDelayedINCcnt;
                buffer2_nandDATA_in <= '1';
        else
                WE_buffer1_H <= OneDelayedINCcnt;
                -- to keep the default
                WE_buffer2_H <= not(WE_L) and not(CE_L);
                buffer1_nandDATA_in <= '1';
        end if;
        DEBUGstatevector <= x"BB";

when STreadpageCORwaitforRB =>
        if(nandRB_L = '0') then
                Nstate <= STreadpageCORRECT;
        elsif(ECCpresent = '0') then
                Nstate <= STreadpageSWAPbuf;
        else
                Nstate <= STreadpageCORwaitforRB;
        end if;
        DEBUGstatevector <= x"CC";

when STreadpageCORRECT =>
        if(nandRB_L = '1') then
                if(errINT = '1') then
                        Nstate <= STfetchERRORs;
```

```
                    else
                        Nstate <= STreadpageSWAPbuf;
                    end if;
                else
                    Nstate <= STreadpageCORRECT;
                end if;
                DEBUGstatevector <= x"CE";

            when STreadpageCORRECTaddr =>
                errCORaddrIN <= '1';
                if(buffer1or2 = '0') then
                    -- This means the user has buffer 1
                    -- to keep the default we do this:-
                    WE_buffer1_H <= not(WE_L) and not(CE_L);
                    -- to write to the other buffer
                    WE_buffer2_H <= corWE;
                    buffer2_nandDATA_in <= '1';
                else
                    WE_buffer1_H <= corWE;
                    -- to keep the default
                    WE_buffer2_H <= not(WE_L) and not(CE_L);
                    buffer1_nandDATA_in <= '1';
                end if;
                if(delayCNT = "011") then
                    corWE <= errorLOC(conv_integer(internalCNT(1 downto 0) & '1'))(4)
and
                        not(errorLOC(conv_integer(internalCNT(1 downto 0) &
'1'))(5));
```

```
                else
                        corWE <= '0';
                end if;
                corADDR <= '0' & internalCNT(1 downto 0) &
                        errorLOC(conv_integer(internalCNT(1 downto 0) & '1'))(3
downto 0)
                        & errorLOC(conv_integer(internalCNT(1 downto 0) & '0'))(7
downto 3);
                enableTOGGLE <= '1';
                if (doneTOGGLE = '1') then
                        Nstate <= STreadpageSWAPbuf;
                else
                        Nstate <= STreadpageCORRECTaddr;
                end if;
                DEBUGstatevector <= x"CF";

        when STreadpageSWAPbuf =>
                DEBUGstatevector <= x"DD";
                Nstate <= Start;

        -- Erase Block states
                when STeraseblock =>
                internal_nandDATA <= x"60";
                outputVEC1 <= "10010";
                outputVEC2 <= "11010";
                nandDATA_fsm <= '1';
                flipTHEt <= not(delayCNT(0) or delayCNT(1) or delayCNT(2));
                enableTOGGLE <= '1';
```

```
                if (doneTOGGLE = '1') then
                        Nstate <= enableTOG0;
                else
                        Nstate <= STeraseblock;
                end if;

        when STeraseblockADDcycles =>
                outputVEC1 <= "10100";
                outputVEC2 <= "11100";
                nandDATA_fsm <= '1';
                enableTOGGLE <= '1';
                if (doneTOGGLE = '1') then
                        Nstate <= enableTOG0;
                else
                        Nstate <= STeraseblockADDcycles;
                end if;
                if(internalCNT(1 downto 0) = "11") then
                        internal_nandDATA <= x"00";
                else
                        internal_nandDATA <=
                        REGaddress(conv_integer(not(currentADDreg)))(conv_integer(internalCNT(1
downto 0)));
                end if;

        when STeraseblockCMDd0 =>
                internal_nandDATA <= x"D0";
                outputVEC1 <= "10010";
                outputVEC2 <= "11010";
```

```
            nandDATA_fsm <= '1';
            enableTOGGLE <= '1';
            if (doneTOGGLE = '1') then
                  Nstate <= STwaitforRB0;
            else
                  Nstate <= STeraseblockCMDd0;
            end if;

-- Reset Nand state
when STresetNAND =>
            internal_nandDATA <= x"FF";
            outputVEC1 <= "10010";
            outputVEC2 <= "11010";
            nandDATA_fsm <= '1';
            enableTOGGLE <= '1';
            if (doneTOGGLE = '1') then
                  Nstate <= STwaitforRB0;
            else
                  Nstate <= STresetNAND;
            end if;

-- Fetch Error states
when STfetchERRORs =>
            internal_nandDATA <= ErrorFetchCommand;
            outputVEC1 <= "10010";
            outputVEC2 <= "11010";
            nandDATA_fsm <= '1';
```

```
		enableTOGGLE <= '1';
		if (doneTOGGLE = '1') then
			Nstate <= enableTOG0;
		else
			Nstate <= STfetchERRORs;
		end if;
		DEBUGstatevector <= x"20";

when STfetchERRORseightReads =>
		-- bit 0 - CE_L
		-- bit 1 - CLE_H
		-- bit 2 - ALE_H
		-- bit 3 - WE_L
		-- bit 4 - RE_L
		outputVEC1 <= "01000";
		outputVEC2 <= "11000";
		nandDATA_fsm <= '0';
		enableTOGGLE <= '1';
		if (doneTOGGLE = '1') then
			Nstate <= enableTOG0;
		else
			Nstate <= STfetchERRORseightReads;
		end if;
			DEBUGstatevector <= x"21";

-- Read ID states
when STreadID =>
```

```
            internal_nandDATA <= x"90";
            outputVEC1 <= "10010";
            outputVEC2 <= "11010";
            nandDATA_fsm <= '1';
            enableTOGGLE <= '1';
            if (doneTOGGLE = '1') then
                  Nstate <= enableTOG0;
            else
                  Nstate <= STreadID;
            end if;

when STreadIDadd00 =>
            internal_nandDATA <= x"00";
            outputVEC1 <= "10100";
            outputVEC2 <= "11100";
            nandDATA_fsm <= '1';
            enableTOGGLE <= '1';
            if (doneTOGGLE = '1') then
                  Nstate <= enableTOG0;
            else
                  Nstate <= STreadIDadd00;
            end if;

when STreadIDfourReads =>
            -- bit 0 - CE_L
            -- bit 1 - CLE_H
            -- bit 2 - ALE_H
```

```
		-- bit 3 - WE_L
		-- bit 4 - RE_L
		outputVEC1 <= "01000";
		outputVEC2 <= "11000";
		nandDATA_fsm <= '0';
		enableTOGGLE <= '1';
		if (doneTOGGLE = '1') then
			Nstate <= enableTOG0;
		else
			Nstate <= STreadIDfourReads;
		end if;

-- Status read states
when STreadstatusCMD =>
		internal_nandDATA <= x"70";
		outputVEC1 <= "10010";
		outputVEC2 <= "11010";
		nandDATA_fsm <= '1';
		enableTOGGLE <= '1';
		if (doneTOGGLE = '1') then
			Nstate <= enableTOG0;
		else
			Nstate <= STreadstatusCMD;
		end if;

when STreadstatusRead =>
		nandDATA_fsm <= '0';
```

```
                outputVEC1 <= "01000";
                outputVEC2 <= "11000";
                enableTOGGLE <= '1';
                if (doneTOGGLE = '1') then
                        Nstate <= Start;
                else
                        Nstate <= STreadstatusRead;
                end if;
        end case;
end process;

toggleFSM: process (clk, Reset_L)

begin
        if(rising_edge(clk)) then
                if(rstDcnt_L = '0') then
                        delayCNT <= "000";
                else
                        if(delayCNT = "101") then
                                delayCNT <= "000";
                        elsif(incDcnt = '1') then
                                delayCNT <= std_logic_vector(unsigned(delayCNT) + 1);
                        end if;
                end if;
        if(Reset_L = '0') then
                cTOGstate <= toggleWAIT;
```

```
-- NAND or NAND ecc module interface
--nandCE_L <= '1';
--nandCLE_H <= '0';
--nandALE_H <= '0';
--nandWE_L <= '1';
--nandRE_L <= '1';
else
	-- nand DATA bus MUX
	if(nandDATA_from_buffer1 = '1') then
		nandDATA <= buffer1_DATAout;
	elsif(nandDATA_from_buffer2 = '1') then
		nandDATA <= buffer2_DATAout;
	elsif(nandDATA_fsm = '1') then
		nandDATA <= internal_nandDATA;
	else
		nandDATA <= (others => 'Z');
	end if;
	case cTOGstate is
		when toggleWAIT =>
			nandCE_L <= outputVEC1(0);
			nandCLE_H <= outputVEC1(1);
			nandALE_H <= outputVEC1(2);
			nandWE_L <= outputVEC1(3);
			nandRE_L <= outputVEC1(4);
		when toggle1 =>
			nandCE_L <= outputVEC1(0);
			nandCLE_H <= outputVEC1(1);
```

```
                    nandALE_H <= outputVEC1(2);
                    nandWE_L <= outputVEC1(3);
                    nandRE_L <= outputVEC1(4);
                when toggle2 =>
                    nandCE_L <= outputVEC2(0);
                    nandCLE_H <= outputVEC2(1);
                    nandALE_H <= outputVEC2(2);
                    nandWE_L <= outputVEC2(3);
                    nandRE_L <= outputVEC2(4);
                when toggleDONE =>
                    nandCE_L <= '1';
                    nandCLE_H <= '0';
                    nandALE_H <= '0';
                    nandWE_L <= '1';
                    nandRE_L <= '1';
            end case;
            cTOGstate <= nTOGstate;
        end if;
        end if;
end process toggleFSM;

COMBtoggleFSM: process(cTOGstate, enableTOGGLE, delayCNT, internalCNT, CNTupto, outputVEC1,
outputVEC2)

begin
        doneTOGGLE <= '0';
        internalCNTRst_L <= '1';
        incintCNT <= '0';
```

```
rstDcnt_L <= '1';
incDcnt <= '0';
case cTOGstate is
	when toggleWAIT =>
		rstDcnt_L <= '0';
		if(enableTOGGLE = '1') then
			nTOGstate <= toggle1;
		else
			nTOGstate <= toggleWAIT;
		end if;
		internalCNTRst_L <= '0';

	when toggle1 =>
		incDcnt <= '1';
		-- this is swtiching at 2 as
		-- first it takes a clock cycle to switch
		-- states and then it takes another clock to switch the register

		if(delayCNT >= "011") then
			nTOGstate <= toggle2;
		else
			nTOGstate <= toggle1;
		end if;

	when toggle2 =>
		incDcnt <= '1';
		if (delayCNT = "100") then
			incintCNT <= '1';
```

```
                        end if;
                        -- switching at 4 as the same reason
                        -- 2 clocks before the signal goes back
                        if (delayCNT >= "101") then
                                if(internalCNT >= CNTupto) then
                                        nTOGstate <= toggleDONE;
                                else
                                        nTOGstate <= toggle1;
                                end if;
                        else
                                nTOGstate <= toggle2;
                        end if;

                when toggleDONE =>
                        rstDcnt_L <= '0';
                        doneTOGGLE <= '1';
                        if (enableTOGGLE = '0') then
                                nTOGstate <= toggleWAIT;
                        else
                                nTOGstate <= toggleDONE;
                        end if;
                end case;
        end process COMBtoggleFSM;

REPETITIONcounter: process(clk, internalCNTRst_L, incintCNT)

begin
```

```
	if(rising_edge(clk)) then
		if(internalCNTRst_L = '0') then
			internalCNT <= x"000";
		else
			if (incintCNT = '1') then
				internalCNT <= std_logic_vector(unsigned(internalCNT) + 1);
			end if;
		end if;
	end if;
end process REPETITIONcounter;

MUXoutsignals: process(buffer1_nandDATA_in, buffer2_nandDATA_in, nandDATA_from_buffer1,
nandDATA_from_buffer2,
			nandDATA, ST2inDATA, buffer1_DATAout, buffer2_DATAout, nandDATA_fsm,
internal_nandDATA, buffer1or2,
			ADDR, internalCNTaddr, nandID, ECCpresent, Device_Size, Device8_not16,
StatusREG, CommandREG, OE_L,
			CE_L, WE_L, currentADDreg, errorLOC, errCORaddrIN, corINPUTdata,
corADDR)

begin
	-- Buffer 1 and 2: data inputs
	if(buffer1_nandDATA_in = '1') then
		if(errCORaddrIN = '1') then
			buffer1_DATA <= corINPUTdata;
		else
			buffer1_DATA <= nandDATA;
		end if;
	else
```

```
		buffer1_DATA <= ST2inDATA;
end if;
if(buffer2_nandDATA_in = '1') then
		if(errCORaddrIN = '1') then
			buffer2_DATA <= corINPUTdata;
		else
			buffer2_DATA <= nandDATA;
		end if;
else
		buffer2_DATA <= ST2inDATA;
end if;

-- buffer addressing
if(buffer1or2 = '0') then
		if(ADDR < x"840") then
			buffer1_ADDR <= ADDR;
		else
			buffer1_ADDR <= x"840";
		end if;
if(errCORaddrIN = '1') then
			buffer2_ADDR <= corADDR;
		else
			buffer2_ADDR <= internalCNTaddr;
		end if;
else
		if(ADDR < x"840") then
			buffer2_ADDR <= ADDR;
```

```
                else
                        buffer2_ADDR <= x"840";
                end if;
        if(errCORaddrIN = '1') then
                        buffer1_ADDR <= corADDR;
                else
                        buffer1_ADDR <= internalCNTaddr;
                end if;
        end if;
        if(OE_L='0' and CE_L = '0' and WE_L = '1') then
                if(ADDR(11 downto 4) = x"FF") then
                        if(ADDR(3 downto 0) = x"0") then
                                ST2outDATA <= nandID(0);
                        elsif(ADDR(3 downto 0) = x"1") then
                                ST2outDATA <= nandID(1);
                        elsif(ADDR(3 downto 0) = x"2") then
                                ST2outDATA <= nandID(2);
                        elsif(ADDR(3 downto 0) = x"3") then
                                ST2outDATA <= nandID(3);
                        elsif(ADDR(3 downto 0) = x"4") then
                                ST2outDATA <= REGaddress(conv_integer(currentADDreg))(0);
                        elsif(ADDR(3 downto 0) = x"5") then
                                ST2outDATA <= REGaddress(conv_integer(currentADDreg))(1);
                        elsif(ADDR(3 downto 0) = x"6") then
                                ST2outDATA <= REGaddress(conv_integer(currentADDreg))(2);
                        elsif(ADDR(3 downto 0) = x"7") then
                                ST2outDATA <= '0' & ECCpresent & "000" & Device_Size &
Device8_not16;
```

```
                elsif(ADDR(3 downto 0) = x"8") then
                        ST2outDATA <= "000000" & currentADDreg & buffer1or2;
                elsif(ADDR(3 downto 0) = x"9") then
                        ST2outDATA <= StatusREG;
                elsif(ADDR(3 downto 0) = x"A") then
                        ST2outDATA <= CommandREG;
                else
                --ST2outDATA <= (others => 'Z');
                ST2outDATA <= x"AA";
                end if;
        elsif(ADDR(11 downto 3) = x"FE" & '0') then
                ST2outDATA <= errorLOC(conv_integer(ADDR(2 downto 0)));
        else
                if(buffer1or2 = '0') then
                        ST2outDATA <= buffer1_DATAout;
                else
                        ST2outDATA <= buffer2_DATAout;
                end if;
        end if;
        else
        ST2outDATA <= x"AA";
        --DATA <= (others => 'Z');
        end if;
end process MUXoutsignals;
end syn;
-------------------------------------------------
--                  FSM Ends
```
