```
-- I2C_CONTROL
-- version 3 12/10/2002
library IEEE;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
library synplify;
use synplify.attributes.all;

entity I2C_CONTROL is port

(clock              : in  std_logic;
resetn              : in  std_logic;
state_out           : out std_logic_vector(5 downto 0);

-- error register connections
respond_error       : out std_logic;--_vector(7 downto 0);
--clear_error         : in  std_logic;

-- i/o to serial controller
CONTROL_REG         : in  std_logic_vector(6 downto 0);
NUM_TRANSFERS_REG   : in  std_logic_vector(3 downto 0);
POINTER_REGISTER    : in  std_logic_vector(7 downto 0);
DATA_REGISTER       : in  std_logic_vector(7 downto 0);
operate             : in  std_logic;
R_W_control         : in  std_logic;
done                : out std_logic;
busy                : out std_logic;


-- I2C core connections
start_bit           : out std_logic;
stop_bit            : out std_logic;
read_bit            : out std_logic;
write_bit           : out std_logic;
ack_in              : out std_logic;
ack_out             : in std_logic;
command_ack         : in std_logic;
I2C_busy            : in std_logic;
data_wr_slave       : out std_logic_vector(7 downto 0);
data_rd_slave       : in  std_logic_vector(7 downto 0);

-- EXTERNAL FRAM CONNECTIONS
fram_address        : out std_logic_vector(9 downto 0);
fram_OE             : out std_logic;
fram_WE             : out std_logic;
fram_CE             : out std_logic;
fram_D              : out std_logic_vector(7 downto 0);
fram_Q              : in std_logic_vector(7 downto 0);

-- enable for I2C output pins
I2C_enable          : out std_logic_vector(6 downto 0);

-- RAM BLOCK connections
RAM_ADDRESS         : out std_logic_vector(4 downto 0);
RAM_DATA_D          : out std_logic_vector(7 downto 0);
RAM_WRITE_ENABLE    : out std_logic;
RAM_READ_ENABLE     : out std_logic;
RAM_DATA_Q          : in  std_logic_vector(7 downto 0));


end I2C_CONTROL;



architecture rtl of I2C_CONTROL is
attribute syn_radhardlevel of rtl : architecture is "tmr";
```

```
type state_values is (st0, st1, st2, st3, st4, st5, st6, st7, st8, st9, st10, st11, st12,
st13, st14, st15, st16, st17, st18, st19, st20, st21, st22, st23, st24, st25, st26, st27,
st28, st29, st30, st31, st32, st33);

signal pres_state           : state_values;
signal error_ack            : std_logic;
signal device_id            : std_logic_vector(6 downto 0);
signal num_transfers        : std_logic_vector(3 downto 0);
signal RAM_ADDRESS_TEMP     : std_logic_vector(4 downto 0);
signal fram_address_temp    : std_logic_vector(9 downto 0);

begin
    -- fsm register
    state_reg: process (clock, resetn)

    begin

    if (resetn = '0') then
        state_out <= "000000";
        RAM_DATA_D <= "00000000";
        fram_address <= "0000000000";
        RAM_ADDRESS <= "00000";
        fram_OE <= '1';
        fram_WE <= '1';
        fram_CE <= '1';
        fram_D <= "00000000";
        RAM_READ_ENABLE <= '0';
        RAM_WRITE_ENABLE <= '0';
        busy <= '0';
        done <= '0';
        num_transfers <= "0000";
        fram_address_temp(9 downto 0) <= "0000000000";
        RAM_ADDRESS_TEMP <= "00000";
        start_bit <= '0';
        stop_bit  <= '0';
        read_bit  <= '0';
        write_bit <= '0';
        ack_in <= '0';
        respond_error <= '0';--"00000000";
        error_ack <= '0';
        data_wr_slave <= "00000000";
        pres_state <= st0;
        I2C_enable <= "0000000";


    elsif clock'event AND clock = '1' then

        RAM_ADDRESS     <= RAM_ADDRESS_TEMP;
        fram_address    <= fram_address_temp;



        case pres_state is

            when st0 =>
                state_out <= ("000000");
                num_transfers  <= NUM_TRANSFERS_REG;
                fram_OE <= '1';
                fram_WE <= '1';
                fram_CE <= '1';
                RAM_READ_ENABLE  <= '0';
                RAM_WRITE_ENABLE <= '0';
                start_bit <= '0';
                stop_bit  <= '0';
                read_bit  <= '0';
                write_bit <= '0';
                if operate = '1' then -- perform operation
                   busy <= '1';
```

```
                   done <= '0';
                   pres_state <= st1;
                else
                   busy <= '0';
                   done <= '0';
                   pres_state <= st0;
                end if;

            when st1 => --  decode operation
                state_out <= ("000001");
                fram_address_temp(7 downto 0)  <= POINTER_REGISTER(7 downto 0);
                fram_address_temp(9 downto 8) <= DATA_REGISTER(1 downto 0);
                RAM_ADDRESS_TEMP <= "00100"; -- register 4 (Always starts at 4 for an oper
ation)

                case CONTROL_REG is

                    when "0000000" => I2C_enable <= "0000000"; device_id(6 downto 4) <= "1
11";-- FRAM
                    when "0000001" => I2C_enable <= "0000001"; device_id(6 downto 4) <= "0
00";-- I2C core 1
                    when "0000010" => I2C_enable <= "0000010"; device_id(6 downto 4) <= "0
01";-- I2C core 2
                    when "0000100" => I2C_enable <= "0000100"; device_id(6 downto 4) <= "0
10";-- I2C core 3
                    when "0001000" => I2C_enable <= "0001000"; device_id(6 downto 4) <= "0
11";-- I2C core 4
                    when "0010000" => I2C_enable <= "0010000"; device_id(6 downto 4) <= "1
00";-- I2C core 5
                    when "0100000" => I2C_enable <= "0100000"; device_id(6 downto 4) <= "1
01";-- I2C core 6
                    when "1000000" => I2C_enable <= "1000000"; device_id(6 downto 4) <= "1
10";-- I2C core TTCrx
                    when others   => I2C_enable <= "0000000"; device_id(6 downto 4) <= "11
1";-- FRAM
                end case;

                case POINTER_REGISTER(5 downto 0) is

                    when "000000" =>  device_id(3 downto 0) <= "0000";-- board 1 chip 1

                    when "000001" =>  device_id(3 downto 0) <= "0001";-- board 1 chip 2
                    when "000010" =>  device_id(3 downto 0) <= "0010";-- board 1 chip 3
                    when "000011" =>  device_id(3 downto 0) <= "0011";-- board 1 chip 4
                    when "000100" =>  device_id(3 downto 0) <= "0100";-- board 1 chip 5
                    when "010000" =>  device_id(3 downto 0) <= "0101";-- board 2 chip 1
                    when "010001" =>  device_id(3 downto 0) <= "0110";-- board 2 chip 2
                    when "010010" =>  device_id(3 downto 0) <= "0111";-- board 2 chip 3
                    when "010011" =>  device_id(3 downto 0) <= "1000";-- board 2 chip 4
                    when "010100" =>  device_id(3 downto 0) <= "1001";-- board 2 chip 5
                    when "100000" =>  device_id(3 downto 0) <= "1010";-- board 3 chip 1
                    when "100001" =>  device_id(3 downto 0) <= "1011";-- board 3 chip 2
                    when "100010" =>  device_id(3 downto 0) <= "1100";-- board 3 chip 3
                    when "100011" =>  device_id(3 downto 0) <= "1101";-- board 3 chip 4
                    when "100100" =>  device_id(3 downto 0) <= "1110";-- board 3 chip 5
                    when others   =>  device_id(3 downto 0) <= "1111";-- board 0 chip 0 TT
Crx
                end case;

                if CONTROL_REG = "0000000" then
                   pres_state <= st2; -- FRAM operation
                else
                   pres_state <= st16; -- I2C operation
                end if;

            when st2 => -- let temp address get registered into address
                    state_out <= ("000010");
                    pres_state <= st3;
```

```
            when st3 => --  FRAM enabled
                    state_out <= ("000011");
                    if R_W_control = '0' then       -- write FRAM
                        RAM_READ_ENABLE <= '1';     -- get data from RAM and put data in F
RAM
                        fram_OE <= '1';
                        fram_WE <= '0';
                        fram_CE <= '1';
                        pres_state <= st4;
                    else                            -- read FRAM
--                        RAM_WRITE_ENABLE <= '1';    -- get data from FRAM and put data i
n RAM
                        fram_OE <= '1';
                        fram_WE <= '1';
                        fram_CE <= '1';
                        pres_state <= st6;
                    end if;

            when st4 => -- write FRAM
                    state_out <= ("000100");
                    pres_state <= st5;

           when st5 => -- write FRAM
                    state_out <= ("000101");
                    fram_D(7 downto 0) <= RAM_DATA_Q(7 downto 0);
                    fram_OE <= '1';
                    fram_WE <= '0';
                    fram_CE <= '0'; -- needs to go low to latch address of FRAM
                    pres_state <= st8;

           when st6 => -- read FRAM
                    state_out <= ("000110");
                    fram_OE <= '0';
                    fram_WE <= '1';
                    fram_CE <= '0'; -- needs to go low to latch address of FRAM
                    pres_state <= st7;

           when st7 =>
                    state_out <= ("000111");
                    pres_state <= st8;

           when st8 =>
                    state_out <= ("001000");
                    RAM_READ_ENABLE <= '0';
                    pres_state <= st9;

           when st9 =>
                    state_out <= ("001001");
                    pres_state <= st10;

           when st10 =>
                    state_out <= ("001010");
                    if R_W_control = '0' then       -- write FRAM
                        pres_state <= st11;
                    else
                        RAM_DATA_D(7 downto 0) <= fram_Q(7 downto 0);
                        RAM_WRITE_ENABLE <= '1';
                        pres_state <= st12;
                    end if;

           when st11 =>
                    state_out <= ("001011");
                    fram_OE <= '1';
                    fram_WE <= '1';
                    fram_CE <= '1';
                    pres_state <= st12;
```

```
            when st12 =>
                     state_out <= ("001100");
                     pres_state <= st13;

            when st13 =>
                     RAM_WRITE_ENABLE <= '0';
                     state_out <= ("001101");
                     pres_state <= st14;

            when st14 =>
                     state_out <= ("001110");
                     fram_OE <= '1';
                     fram_WE <= '1';
                     fram_CE <= '1';
                     pres_state <= st15;

            when st15 =>
                     state_out <= ("001111");
                     if num_transfers = 1 then
                         pres_state <= st33;
                         done <= '1';
                     else
                         num_transfers <= num_transfers - 1;
                         fram_address_temp <= fram_address_temp + 1;
                         RAM_ADDRESS_TEMP <=  RAM_ADDRESS_TEMP + 1;
                         pres_state <= st2;
                     end if;

                         -- setup I2C mission
             when st16 => -- write address of I2C device pointer register
                     state_out <= ("010000");
                     if (I2C_BUSY = '0') then
                         data_wr_slave(7 downto 2) <=  POINTER_REGISTER(5 downto 0);
                         data_wr_slave(1) <= '0'; -- selects pointer register
                         data_wr_slave(0) <= '0'; -- write
                         start_bit <= '1';
                         write_bit <= '1';
                         pres_state <= st17;
                     else
                         pres_state <= st16;
                     end if;


             when st17 => -- check if transfer over
                     state_out <= ("010001");
                     if (command_ack = '1') then
                         start_bit <= '0';
                         write_bit <= '0';
                         if ack_out = '0' then
                             pres_state <= st18; -- slave responded
                         else
                             pres_state <= st31; -- slave didn't respond
                             stop_bit <= '1';
                             error_ack <= '1';
                         end if;
                     else
                         pres_state <= st17;
                     end if;

             when st18 => -- write I2C register number to I2C device pointer register of I2
C device addressed
                     state_out <= ("010010");
                     if (I2C_BUSY = '0') then
                         data_wr_slave(7 downto 0) <=  DATA_REGISTER(7 downto 0);

                         write_bit <= '1';
                         pres_state <= st19;
                     else
```

```
                    pres_state <= st18;
                end if;

        when st19 => -- check if transfer over
                state_out <= ("010011");
                if (command_ack = '1') then
                    write_bit <= '0';
                    if ack_out = '0' then
                        pres_state <= st20; -- slave responded
                    else
                        pres_state <= st31; -- slave didn't respond
                        stop_bit <= '1';
                        error_ack <= '1';
                    end if;
                else
                    pres_state <= st19;
                end if;

       when st20 => -- end of setup mission
                 state_out <= ("010100");
                if R_W_control = '0' then
                      pres_state <= st21; -- I2C write
                else
                      pres_state <= st26; -- I2C read
                end if;


                    -- begin write mission
        when st21 => -- write address of I2C data register
                state_out <= ("010101");
                RAM_ADDRESS_TEMP <= "00100"; -- register 4
                RAM_READ_ENABLE <= '1'; -- get data from RAM register 4
                if (I2C_BUSY = '0') then
                    data_wr_slave(7 downto 2) <=  POINTER_REGISTER(5 downto 0);
                    data_wr_slave(1) <= '1'; -- selects data register address
                    data_wr_slave(0) <= '0'; -- write command
                    start_bit <= '1'; -- repeat start condition
                    write_bit <= '1';
                    pres_state <= st22;
                else
                    pres_state <= st21;
                end if;



        when st22 => -- transfer over
                state_out <= ("010110");
               if (command_ack = '1') then
                    start_bit <= '0';
                    write_bit <= '0';
                    if ack_out = '0' then
                        pres_state <= st23; -- slave responded
                    else
                        pres_state <= st31; -- slave didn't respond
                        stop_bit <= '1';
                        error_ack <= '1';
                    end if;
                else
                    pres_state <= st22;
                end if;

        when st23 => -- write RAM data to I2C data register
                state_out <= ("010111");
                if (I2C_BUSY = '0') then
                    data_wr_slave(7 downto 0) <=  RAM_DATA_Q(7 downto 0); -- ram data

                    write_bit <= '1';
                    if num_transfers = 1 then
```

```
                        stop_bit <= '1';
                    else
                        stop_bit <= '0';
                    end if;
                    pres_state <= st24;
                else
                    pres_state <= st23;
                end if;

            when st24 => -- initiate transfer
                state_out <= ("011000");
               if (command_ack = '1') then
                    write_bit <= '0';
                    RAM_READ_ENABLE <= '0';
                    if ack_out = '0' then
                        pres_state <= st25; -- slave responded
                    else
                        pres_state <= st31; -- slave didn't respond
                        stop_bit <= '1';
                        error_ack <= '1';
                    end if;
                else
                    pres_state <= st24;
                end if;

            when st25 =>
                state_out <= ("011001");
                if num_transfers = 1 then -- end write mission
                    stop_bit <= '0';
                    done <= '1';
                    pres_state <= st33;
                else                      -- next address
                    num_transfers <= num_transfers - 1;
                    RAM_ADDRESS_TEMP <=  RAM_ADDRESS_TEMP + 1;
                    RAM_READ_ENABLE <= '1';
                    pres_state <= st23;
                end if;


                     -- begin read mission
            when st26 => -- write I2C device address of data register with read command
                state_out <= ("011010");
                RAM_ADDRESS_TEMP <= "00100"; -- register 4 (always begins at RAM addre
ss 4.
                if (I2C_BUSY = '0') then
                    data_wr_slave(7 downto 2) <=  POINTER_REGISTER(5 downto 0);
                    data_wr_slave(1) <= '1'; -- selects data register address
                    data_wr_slave(0) <= '1'; -- read command
                    start_bit <= '1'; -- repeat start condition
                    write_bit <= '1';
                    pres_state <= st27;
                else
                    pres_state <= st26;
                end if;


            when st27 => -- transfer over
                state_out <= ("011011");
               if (command_ack = '1') then
                    start_bit <= '0';
                    write_bit  <= '0';
                    if ack_out = '0' then
                        pres_state <= st28; -- slave responded
                    else
                        pres_state <= st31; -- slave didn't respond
                        stop_bit <= '1';
                        error_ack <= '1';
                    end if;
```

```
                    else
                        pres_state <= st27;
                    end if;


            when st28 => --  read command  - read back from I2C
                    state_out <= ("011100");
                        read_bit    <= '1';
                        if num_transfers = 1 then
                            stop_bit <= '1';
                            Ack_in      <= '1';
                        else
                            stop_bit <= '0';
                            Ack_in      <= '0';
                        end if;
                        pres_state <= st29;


            when st29 => -- write data read back to RAM
                    state_out <= ("011101");
                   if (command_ack = '1') then
                        Ack_in   <= '0';
                        read_bit <= '0';
                        RAM_DATA_D(7 downto 0) <= data_rd_slave(7 downto 0); -- write I2C
data read back at RAM
                        RAM_WRITE_ENABLE <= '1'; -- enable write to RAm

                        pres_state <= st30;
                    else
                        pres_state <= st29;
                    end if;

            when st30 =>
                    state_out <= ("011110");
                    if num_transfers = 1 then -- end read mission
                        done <= '1';
                        pres_state <= st33;
                    else
                        num_transfers <= num_transfers - 1;
                        RAM_ADDRESS_TEMP <=  RAM_ADDRESS_TEMP + 1;
                        pres_state <= st28;
                    end if;

            when st31 => -- error_ack
                    state_out <= ("011111");
                    respond_error <= error_ack;
                    --respond_error(6 downto 0) <= device_id;
                    pres_state <= st32;

            when st32 => --
                    state_out <= ("100000");
                    error_ack <= '0';
                    --if clear_error = '1' then
                        respond_error <= '0';
                        done <= '1';
                        pres_state <= st33;
                    --else
                    --    pres_state <= st32;
                   -- end if;

            when st33 => -- end of mission
                    state_out <= ("100001");
                    stop_bit <= '0';
                    pres_state <= st0;

        end case;
        end if;
        end process;
```

```
 end rtl;
```