Uitleg bij de gegeven VHDL code

inleiding

De bedoeling van dit document is om je op weg te zetten in de gegeven VHDL code. Je zal merken dat het niet evident is om bestaande code te debuggen, zeker als je ze zelf niet geschreven hebt. Dit is echter iets wat je ongetwijfeld nog veel zal moeten doen in het bedrijfsleven en zelfs met eigen code, die je lang geleden geschreven hebt.

Dit document probeert niet om de volledige code in detail te verklaren, maar je enkel een eerste aanzet te geven, zodat je in grote lijnen begrijpt wat de blokken code doen.

Om de fout(en) in deze code te kunnen vinden, zal je ze dus eerst goed moeten begrijpen om vervolgens via de verschillende debugmethoden te achterhalen wat er juist fout loopt, zodat je tenslotte op de juiste plaats(en) in de code veranderingen kan aanbrengen.

entity

```
entity I2CIntf is
  generic(
    g_ClkFreq   : integer := 100_000_000;       -- input clock speed from user logic in Hz
    g_BusFreq   : integer := 100_000);          -- speed the i2c bus (scl) will run at in Hz
  port(
    Clk         : in std_logic;                 -- system clock
    ResetN      : in std_logic;                 -- active low reset
    SDA         : inout std_logic;              -- serial data output of i2c bus
    SCL         : inout std_logic;              -- serial clock output of i2c bus
    Leds        : out std_logic_vector(7 downto 0)); -- led outputs
end I2CIntf;
```

Aan de hand van de twee generic parameters kan de kloksnelheid van externe klok en die van de I^2C bus ingesteld worden. Het kristal op het FPGA bord werkt aan 100 MHz, dus als je deze rechtstreeks gebruikt (wat standaard zo is), moet dit niet veranderd worden.

SDA en SCL vormen de I^2C bus en de Leds uitgangen sturen de 8 LED's aan op het FPGA bord.

bit timing

Zoals je ondertussen zou moeten weten van de uitleg over het I^2C protocol, is de volgorde waarin SCL en SDA van waarde veranderen belangrijk. Daarom wordt in het p_BitTiming process gewerkt met 4 stappen, waarbij iedere stap dus een vierde van de I^2C busperiode duurt. Bij de standaard busfrequentie van 100 KHz, duurt zo één stap dus 2,5 microseconden.

```
------------------------------------------------------------------
-- bit timing:
------------------------------------------------------------------
--       +-+ +-+ +-+ +-+ +-
--       | | | | | | | | |
--      -+ +-+ +-+ +-+ +-+
--      (0) (1) (2) (3) (0)
--      -+       +-------+
--  SCL  |       |       |
--       +-------+       +-
--
--      ----\ /------------
--  SDA      X     stable
--      ----/ \------------
--
p_BitTiming: process (Clk, ResetN)
begin
  if ResetN = '0' then
```

In bovenstaande figuur kan je dus zien dat SCL bijgevolg een mooie blokgolf wordt met een periode van 10 microseconden (100 KHz) door van waarde te veranderen op stap 0 en 2.

Volgens het I^2C protocol mag SDA enkel veranderen wanneer SCL laag is, op stap 1. De enige uitzondering is de stop-bit, waarbij SDA verandert op stap 3.

byte timing

Het process p_ByteTiming gaat uit van een correcte werking van het p_BitTiming process en geeft enkel bitgewijs de gewenste waarden door.

Om deze waarden door te geven, worden de signalen SclEn, SdaEn en SdaInt gebruikt.

Zulke onderverdeling in bit timing en byte timing kan je vergelijken met een laag 0 en laag 1 van het OSI-model, waarbij laag 0 zich enkel bezig houdt met het correct overbrengen van bits, terwijl laag 1 een data frame probeert op te bouwen, steunend op laag 0.

```
------------------------------------------------------------------
-- byte timing:
------------------------------------------------------------------
-- There are 3 possibilities for access to the i2c slave address (SADDR):
--
-- 1) Writing data (DATAW) to register address (RADDR)
--
-- +-------+-------------+------+-------+------+-------+------+------+
-- | Start | SADDR+write | ACKA | RADDR | ACKD | DATAW | ACKD | Stop |
-- +-------+-------------+------+-------+------+-------+------+------+
--
-- 2) Reading data (DATAR) from register address (RADDR)
--
-- +-------+-------------+------+-------+------+------+-------+------------+------+-------+-------+------+
-- | Start | SADDR+write | ACKA | RADDR | ACKD | Stop | Start | SADDR+read | ACKA | DATAR | NACKD | Stop |
-- +-------+-------------+------+-------+------+------+-------+------------+------+-------+-------+------+
--
-- 2) Burst reading data (DATAR1,2,...) from register address (RADDR)
--
-- +-------+-------------+------+-------+------+------+-------+------------+------+--------+------+--------+------+-----+-------+-------+------+
-- | Start | SADDR+write | ACKA | RADDR | ACKD | Stop | Start | SADDR+read | ACKA | DATAR1 | ACKD | DATAR2 | ACKD | ... | DATARX| NACKD | Stop |
-- +-------+-------------+------+-------+------+------+-------+------------+------+--------+------+--------+------+-----+-------+-------+------+
--
p_ByteTiming: process (Clk, ResetN)
begin
  if ResetN = '0' then
```

In bovenstaande figuur staan de drie mogelijke frames die opgebouwd worden in deze state machine:

1) data schrijven naar een bepaald register adres.
2) data lezen van een bepaald register adres.
3) data lezen “in burst” van opeenvolgende register adressen.

Merk op dat in elk van deze gevallen steeds gestart wordt met het I²C slave adres van de specifieke component op de bus. Dit adres is niet hetzelfde als het interne registeradres waar je mee communiceert.

In elk van de drie gevallen wordt er ook begonnen met te schrijven naar het specifieke interne registeradres, ook als er gelezen moet worden van dat adres. Het is pas in het tweede gedeelte van de frame dat de eigenlijke leesbewerking wordt uitgevoerd. Dit is eigen aan het I²C protocol.

In dit process wordt er gewerkt met verschillende toestanden (states), waarbij er soms meerdere klokcycli lang in dezelfde toestand moet verbleven worden (bijvoorbeeld e_Addr, waarin meerdere adresbits doorgestuurd moeten worden). Om te kunnen bijhouden hoeveel klokcycli er al gepasseerd zijn in diezelfde toestand, wordt er gewerkt met het BitCnt signaal.

Er is ook nog een ReadCnt signaal dat enkel gebruikt wordt tijdens het burst lezen van meerdere opeenvolgende adressen.

programma

```
-- this "program" works for one specific device. You will have to change this for your specific device.
p_Program: process (Clk, ResetN)
begin
  if ResetN = '0' then
    Program   <= e_Idle;
    Addr      <= (others => '0');
    Data      <= (others => '0');
    Writing   <= false;
    Reading   <= false;
    Burst     <= false;
    Status    <= false;
  elsif Clk'event and Clk = '1' then
    if ClkDivCnt = c_ClkDiv * 4 - 3 then -- one clk cycle before the Program FSM updates
      if Status and Program = e_ReadStatus then -- Program has just read the status register
        if DataRead(0)(0) = '0' then -- the RDY bit of the status register is not set
          Program     <= e_ReadStatus; -- read the status register again
          Addr        <= x"09";
          Writing     <= false;
          Reading     <= true;
          Burst       <= false;
          Status      <= false;
        end if;
      end if;
    end if;
    if ClkDivCnt = c_ClkDiv * 4 - 2 then -- one clk cycle before p_ByteTiming updates
      if BusState = e_Idle then
        case Program is
          when e_SetModeSingle =>
            Program   <= e_ReadStatus;
            Addr      <= x"09"; -- this specific device has a "status" register at address 09
```

Het laatste process p_Program staat eigenlijk bovenop de byte timing laag. Het is als het ware laag 2, die telkens bepaalde registerbewerkingen aanvraagt aan laag 1, die op zijn beurt laag 0 aanspreekt.

Deze gegeven VHDL code werd geschreven voor één bepaalde I²C module, die een eigen verzameling aan interne registers heeft (bijvoorbeeld een status register op adres 9). Ongetwijfeld is dit anders voor de module die jij gekregen hebt, dus dit zal je sowieso moeten wijzigen.

Het programma in deze VHDL code, voor deze specifieke I²C module doet het volgende:

- Schrijf waarde 1 in adres 2 (een mode register), wat de module in single mode plaatst.
- Lees het status register (op adres 9).
- Wacht tot bit 0 van dit status register gelijk is aan 1, wat aangeeft dat de module “klaar” is.
- Lees “in burst” 6 opeenvolgende data registers (adressen 3 t.e.m. 8) en plaats deze data in de matrix DataRead (array of std_logic_vector)

Nogmaals, dit programma is NIET geschikt voor jouw I^2C module. Je zal zelf (in opdracht 2) een eigen versie hiervan moeten schrijven. Je kan je hierbij wel baseren op het gegeven state machine.

Indien je een burst nodig hebt van een andere lengte (bijvoorbeeld 4 opeenvolgende adressen lezen), dan zal je op twee plaatsen in de code een wijziging moeten aanbrengen, namelijk in het process p_ByteTiming en in de type definitie van t_DataRead.