TSTE12 Design of Digital Systems Lecture 6

Kent Palmkvist

LINKÖPING UNIVERSITY

Agenda

- Practical issues
  - Handins
  - Audio codec function and interface
- High abstraction level modelling

TSTE12 Deadlines Y,D,ED

- Final version of Requirement specification today 8 September
- Initial version design sketch and project plan tuesday 12 September
  - Design sketch (proper document) describe block structure and functions
  - Project plan describe activities (what, who and when)
- Weekly meetings should start
  - Internal weekly meeting with transcript sent to supervisor
- Lab 2 soft deadline Wednesday 13 September at 21.00
  - Lab 2 results will be checked after project end

TSTE12 Deadlines MELE, erasmus

- First project meeting no later than Monday 11 September
- Tuesday 12 September: First version of requirement specification
- Wednesday 13 September 21.00: Lab 1 deadline
  - Pass required to be allowed continued project participation

Handin (homework), Individual!

- 1st handin deadline Monday 18 September
  - Available on web from Monday 11 September
- Submit answers using Lisam assignment function (individual work!!!)
- Theory question answers entered direct as text (see assignments on web)
- Use your own home directory for code answer testing (Not lab group directory)
  - Use ~/TSTE12/
- Use a special terminal window when working with handins

  module load TSTE12 ; TSTE12handin

Individual handin task, cont.

- Create handin code answers using a plain text editor
  - emacs, vim, or the built-in editor in modelsim
  - See in the tutorial how start and use modelsim
- Upload the answers of the coding tasks onto Lisam (TSTE12 course room submission)
- Remember to compile and simulate the design
  - Will use source code for checking handin results
- Do not use handin directory for anything else but handin code and answers you make yourself!

Individual handin, cont.

- Hand-ins are **individual** work!
  - Ask me if there are questions about the handin
- Hand-ins are checked automatically (using scripts)
  - Make sure all names and types are correct in code answers
    - Datatype bit is **not** the same as std_logic!
  - Test your code! Do not assume that you have written correct code.
  - NOTE: Do NOT use hdl-designer (do not start the software using TSTE12lab or TSTE12proj)
  - See modelsim tutorial on exercise page
    - www.isy.liu.se/edu/kurs/TSTE12/kursmaterial

DE2-115 board components

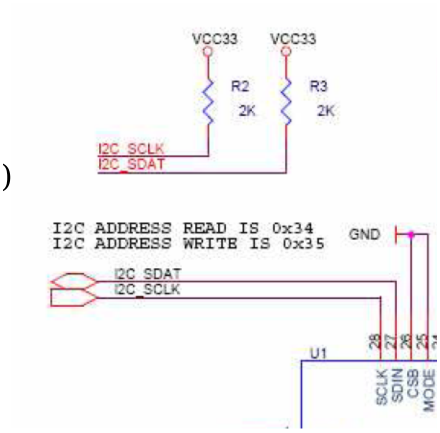
- Audio codec used to input/output analog audio signal
- Codec function
  - Codec can be used in multiple configurations
  - Contains clock generators, A/D, D/A and filters
  - Loopback, volume control
- Codec configuration
  - Default configuration defined in documentation

Individual

- DE2-115 FPGA default design
  - DE2-115 loads a default design at power on
    - Microprocessor design running (NIOS II Soft process, i.e. written i VHDL)
    - Check switches SW3 downto SW0 to select what to do with the SRAM contents and Codec init
    - Infinite loop: read switches, update LEDs, updates 7-segment display.
    - Help text shown on VGA screen
  - Default design is not the standard design described in the DE2-115 user manual
  - Do not depend on power-on defaults
  - Allows configuration of memory and codec for testing

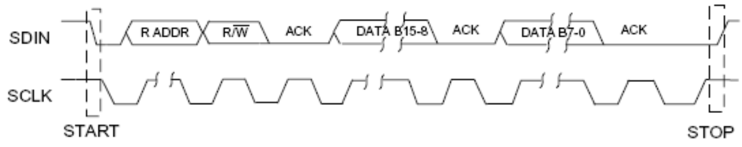
Codec programming (I2C)

- Codec configured using an I2C bus
- General structure
  - Multichip bus (all chips connect to the same pins)
- Pullup
  - Only assign 'Z' or '0' on pins
  - Pullup will translate 'Z' to '1' if other chips does not sink that pin
  - Separate output and input signals to/from pin
  - Values do not change immediately (slow voltage changes)



I2C protocol

- Bidirectional protocol
  - Wired-and using pull-up
  - Send byte by byte
  - MSB first



- FPGA work as master
- Slave (codec) responds with ack after each byte
  - Pulls down SDA in ack cycle
  - Simple solution: assume ack (do not check)

Numeric calculations

- Bit-vectors (and std_logic_vectors) does not directly correspond to a value
  - ”1011” could mean 11 in decimal (unsigned), or -5 in decimal (2’s complement)
- Datatypes are included in supporting packages to enable arithmetic on bit-vectors
  - ieee.numeric_bit.all
  - ieee.numeric_std.all
- Must use defined types signed or unsigned to allow calculations
  - Same definitions as bit_vector and std_logic_vector
  - Can copy values between types due to same element type

Numeric calculations example

- Counter incrementing 3-bit count value each clock cycle
  - Asynchronous reset

```
library ieee;
use ieee.numeric_bit.all;

entity INL3_KB is
  port (
    C : in bit;
    R : in bit;
    Q : out bit_vector(1 to 3));
  end entity;
```

```
architecture KB of INL3_KB is
  begin

    process(C,R)
      variable count : unsigned(1 to 3);
    begin
      if R = '1' then
        count := (others => '0');
      elsif C'event and (C='1') then
        count := count + 1;
      end if;
      Q <= bit_vector(count);
    end process;

  end architecture;
```


Including integers

- Integers can be used for synthesis
  - If synthesis tool cannot figure out the limits, the result is 32-bit arithmetic
  - Subtypes (limiting range) help to reduce hardware and catch unexpected use
- Integers will be implemented as bitvectors
  - Either unsigned or signed (2’s complement)
  - Translation between integer and bitvectors exist
    x_signed := to_signed(y_int,x_signed’size);
  - Translation other way around (unsigned to integer value)
    y_int = to_integer(x_signed);

Another aspect of signal assignment

- One signal can be assigned from different parts of the code
  - Support multiple entities driving the same wire
  - Example: Databus in a computer connecting multiple memories and CPU
- Modelling must be strict and clear
  - Same result independant of simulator tool
  - Should not be able to detect the order the processes where calculated
- Not all data types support multiple sources for the value

Multiple assignment on one signal

- Each process containing a signal assignment will have a driver in the simulator generating a contribution to the final signal value
  - Concurrent signal assignments will have one driver each
  - Processes only have one driver for each signal (even with multiple assignment)
  - The signal update seen before is done individually on each driver
  - One driver does not know anything about other drivers
- When the value of a signal is fetched, the contributions from the different drivers current values are collected.
  - The resulting signal value depends on the definition of how to combine the values from the different drivers, using a resolution function

Example of data types supporting multiple drivers

- Signals driven by multiple drivers must be resolved
  - Use a special function that resolves multiple drivers
- Resolution function
  - Example: Wired-OR
    - `signal X1 : WIRED_OR Bit;`
    - `subtype STD_LOGIC is RESOLVED STD_ULOGIC;`
    - `signal Y2 : STD_LOGIC;`
  - RESOLVED is the resolution function name
    - Called every time the value of the signal is calculated
    - Gets all driver values as input

Example of implementing Multivalued logic in VHDL

- Alternative to data type BIT but simpler than std_logic

```
Type MVL4 is ('X', '0', '1', 'Z');
Type MVL4_VECTOR is array(NATURAL range <>) of MVL4;
```

- X leftmost to make it the initial value unless explicitly initialized in the code

Multidriver signals

- Requires a resolution signal
- Different combinations possible
  - X always overrides others
  - 0 and 1 at the same time gives X
  - Z and Z gives Z

Resolution function definition

```
Subtype DotX is wiredX MVL4;
```

- WiredX is the name of the resolution function

```
Function WiredX (V:MVL4_VECTOR) return MVL4;
```

- Where V is a vector containing all values of all drivers of a signal

Resolution function implementation

- Implement as a loop and lookup table

```
Function wiredX (V: MVL4_VECTOR) return MVL4 is
   Variable result: MVL4:= ’Z’;
Begin
  For i in V’RANGE loop  -- range not known in advance
    Result = table_WIREDX(result,V(i));
    Exit when result = ’X’;
  End loop;
  Return result;
End wiredX;
```


Resolution function impl., cont.

- Check of X in loop is not necessary, but speed up simulation
- Table should then look like:

```
Type MVL4_TABLE is array (MVL4, MVL4) of MVL4;
Constant table_WIREDX : MVL4_TABLE :=
--
--         X    0    1    Z
--
        ((’X’, ’X’, ’X’, ’X’),  --    X
         (’X’, ’0’, ’X’, ’0’),  --    0
         (’X’, ’X’, ’1’, ’1’),  --    1
         (’X’, ’0’, ’1’, ’Z’)); --    Z
```

Resolution function impl. Cont.

- Table lookup may be used for most functions
  - Not possible to know the order of the value in V, may therefore require a more complex algorithm

Bus data type

Type busX is array (Natural range <>) of DotX;

- However, a new data type requires all logic operations to be specified
  - Complicated
- Better approach: conversion function
  - Only read bus using a call to a Sense function

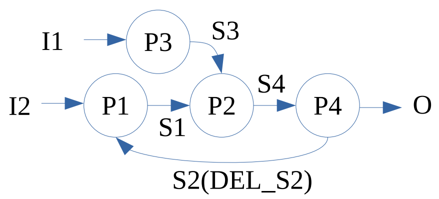
    Function Sense (value : busX) return bit_vector;
  - Only assign value to the bus using the Drive function

    Function Drive (value : bit_vector) return busX

Algorithmic level development

- Specification in many cases in natural language
  - Ambigous description in many cases
- Want an executable specification
  - Allows testing of the behavior the description describes
- Use VHDL to capture the specification
  - Use the full language capabilities
  - Description not intended for synthesis

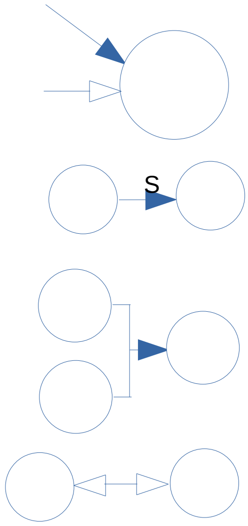
Process Model Graph (PMG)



- Typical example
- Arcs describes signals with names and delays
  - example process 4 to 1
- VHDL Code example: S <= xxx after DEL_S;
- Physical or functional partitioning
  - Single process may map to multiple hardware units
  - Multiple process may map to single hardware unit

Graph Elements

- Two types of signals
  - Triggering signal, put in sensitivity list
  - Sampled signal
- Signals without delay information has a delay of one delta-t
- Signals may be driven by multiple processes. Requires a resolution function
- Signals may be bidirectional. Requires also a resolution function



Example approach

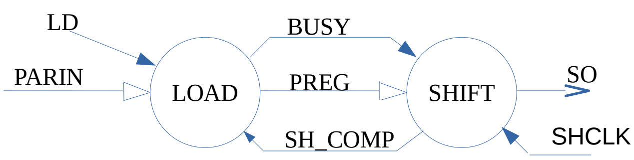
- Map groups of sentences onto VHDL processes
- Assign each process an activity list
- Develop VHDL code that implements each activity

Example: Serial to Paralell converter

- English text description
  - The 8-bit parallel word (PARIN) is loaded into the converter when the control signal LD makes a zero to one transition At this time the status signal BUSY is set high. The data is shifted out serially at a rate controlled by the input shift clock SHCLK. Shifting occurs at the rise of the clock. BUSY remains high until shifting is complete. While BUSY is high, no further loads will be accepted.
- Note some sentences are shared between functions
- Two processes: LOAD and SHIFT

Serial to Paralell converter, cont.

- LOAD: (a) 8-bit parallel word (PARIN) load when LD makes a zero to one transition. Set BUSY high. (b) BUSY remains high until shift complete. No new loads while BUSY high
- SHIFT: (a) Data shifted out controlled by rising edge of SHCLK. (b) BUSY remain high until shift complete




```
entity PAR_TO_SER is
  port(LD,SHCLK: in BIT;
       PARIN: in BIT_VECTOR(0 to 7);
       BUSY: inout BIT := '0';
       SO: out BIT);
end PAR_TO_SER;
```

PMG version

- Corresponding code based on processes
- PMG defines interface of each process + signals between the processes
- Code start by defining processes and comments about activities

```
architecture TWO_PROC of PAR_TO_SER is
  signal SH_COMP: BIT :='0';
  signal PREG: BIT_VECTOR(0 to 7);
begin

  LOAD:process(LD,SH_COMP)
  begin
    ---- Activities:
      ----1)Register Load
      ----2)Busy Set
      ----3)Busy Reset
  end process LOAD;

  SHIFT:process(BUSY,SHCLK)
    variable COUNT: INTEGER;
    variable OREG: BIT_VECTOR(0 to 7);
  begin
    ----Activities:
      ----1)Shift Initialize
      ----2)Shift
      ----3)Shift Complete
  end process SHIFT;

end TWO_PROC;
```


PMG -> Code

- Each process has a check for an event, and then a part that execute the data operations

```
SHIFT:process(BUSY,SHCLK)
  variable COUNT: INTEGER;
  variable OREG: BIT_VECTOR(0 to 7);
begin
  ----Activities:
  if BUSY'EVENT and
     BUSY = '1' then
    ----1)Shift Initialize
    COUNT := 7;
      OREG := PREG;
    SH_COMP <= '0';
  end if;
  if SHCLK'EVENT and
     SHCLK= '1'and
     BUSY='1' then
    ----2)Shift
    SO<=OREG(COUNT);
    COUNT := COUNT - 1;
    ----3)Shift Complete
    if COUNT < 0 then
      SH_COMP <= '1';
    end if;
  end if;
end process SHIFT;
```

```
LOAD:process(LD,SH_COMP)
begin
  ---- Activities:
  if LD'EVENT and LD='1'
     and BUSY='0' then
    ----1)Register Load
    PREG <=  PARIN;
    ----2)Busy Set
    BUSY <= '1';
  end if;
    if SH_COMP'EVENT
       and SH_COMP='1' then
      ----3)Busy Reset
      BUSY <= '0';
    end if;
end process LOAD;
```

Timing example

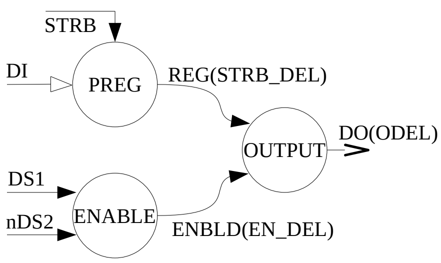
- New model: Buffered register
  - Loaded on rise of the strobe (STRB)
- English description:
  - The register is loaded on the rise of the strobe (STRB), and assuming that the output buffers are enabled, the output of the buffers will change t_{SD} nanoseconds later. The enable condition for the register buffer is the AND of the DS1 and invers of DS2 inputs. Any change in the enable condition will cause the outputs to change t_{ED} nanoseconds later.

Timing example, cont.

- Three processes: PREG, ENABLE, OUTPUT
- Add delay on wires

  t_{SD} = STRB_DEL + ODEL

  t_{ED} = EN_DEL + ODEL



Timing example, cont.

```
entity BUFF_REG is
  generic(
    STRB_DEL,EN_DEL,ODEL: TIME);
  port(
    DI:  in BIT_VECTOR(1 to 8);
    STRB: in BIT;
    DS1:  in BIT;
    NDS2: in BIT;
    DO: out BIT_VECTOR(1 to 8));
end BUFF_REG;


architecture THREE_PROC of BUFF_REG is
    signal REG: BIT_VECTOR(1 to 8);
    signal ENBLD: BIT;
  begin
```

```
  PREG: process(STRB)
    begin
      if (STRB = '1') then
        REG <=DI after STRB_DEL;
      end if;
  end process PREG;

  ENABLE: process(DS1,NDS2)
    begin
      ENBLD <= DS1 and not NDS2 after EN_DEL;
  end process ENABLE;

  OUTPUT: process(REG,ENBLD)
    begin
      if (ENBLD = '1') then
        DO <= REG after ODEL;
      else
        DO <= "11111111" after ODEL;
      end if;
  end process OUTPUT;
end THREE_PROC;
```


Process complexity trade-off

- Number of signals
  - Many signals => slow simulation
- Large processes
  - Complex behavior may not match specification
- Ease of mapping to hardware
  - More processes may simplify mapping

Checking timing

- Additional requirements
  - DI stable SUT ns before STRB rise
  - DI stable HT ns after STRB rise
  - STRB minimum high duration MPW ns
- Implement checks using assert statements

```
assert not (not STRB'stable and (STRB = '1')
            and not DI'stable(SUT))
    report “Setup Time Failure”;
```


Timing Check placement

- Tests in architecture must be copied between architectures
  - May introduce errors
  - If changed, many architectures must be changed
- Solution: Place checks in the entity
  - Check always executed, independent of selected architecture

Timing check example

```
Entity BUFF_REG is
    Generic (STRB_DEL, EN_DEL, ODEL,SUT,HT,MPW: TIME);
    Port (DI: in bit_vector(1 to 8);
          STRB : in bit ; DS1 : in bit;
          NDS2 : in bit;
          DO : out bit_vector(1 to 8));
  Begin
    Assert STRB'stable or (STRB = '0') or DI'stable(SUT)
         Report "Setup time Failure";

    Assert STRB'delayed(HUT)'stable or
          (STRB'delayed(HT) = '0') or DI'STABLE(HT)
         Report "Hold Time Failure";

    Assert STRB'stable or (STRB = '1') or
           STRB'delayed'stable(MPW)
         Report "Minimum pulse width failure";

End BUFF_REG;
```


LINKÖPING UNIVERSITY