KONGSBERG

**WORLD CLASS** – through people, technology and dedication

# Kommandostyrte testbenker i VHDL

KONGSBERG

- Historikk
- Ønskeliste
- Kommandospråk
- Adresseliste generering
- Kommandotolker
- Probe signaler
- Testbenk
- Eksempel
- Konklusjon

# Historikk

KONGSBERG

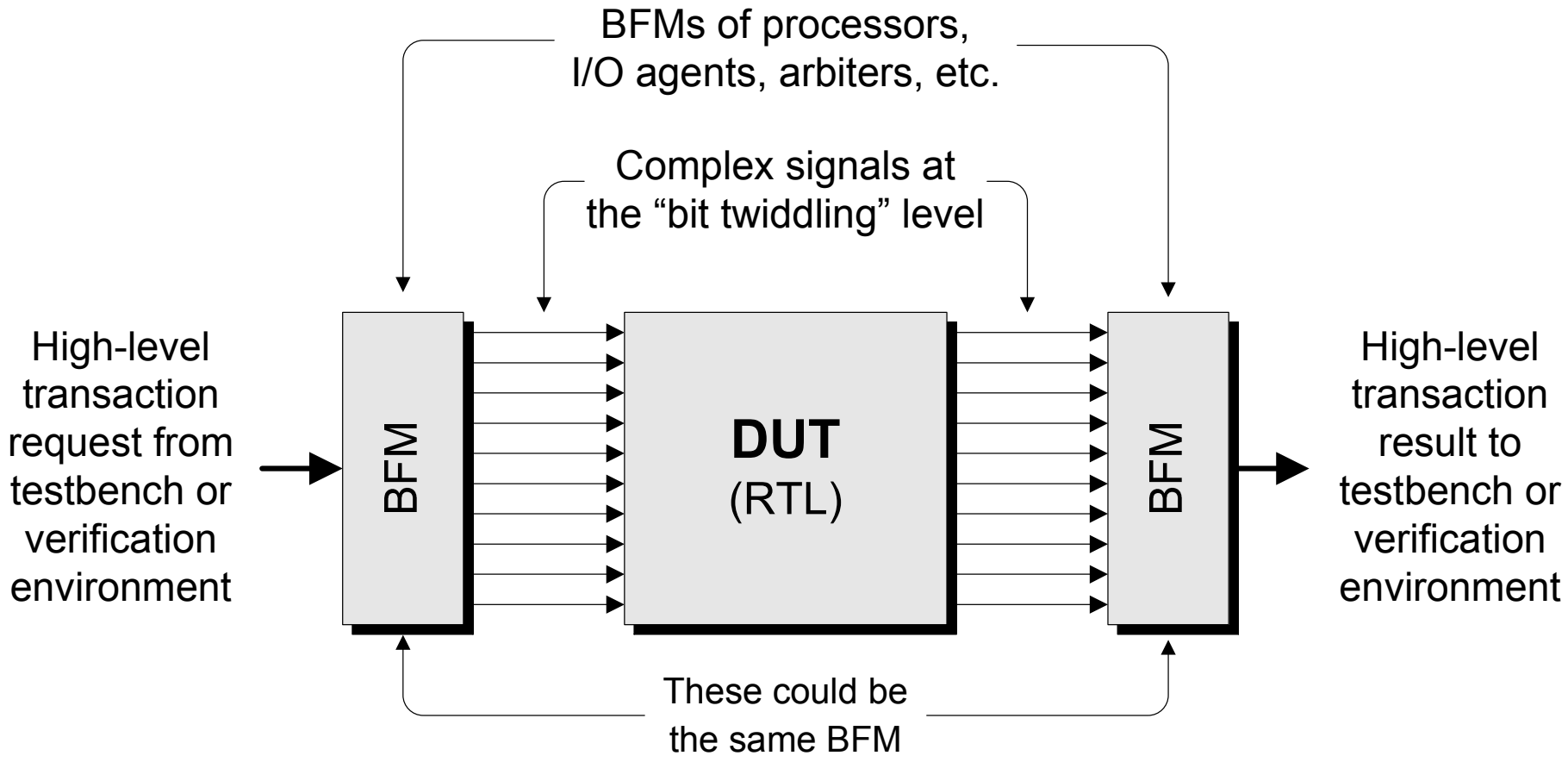
- Tidligere ble vanligvis programvare for verifikasjon (oppførselsnivå fasit) laget i et høynivåspråk som C/C++.
- Med VHDL kom et brukbart alternativ til høynivåspråk. VHDL ble valgt pga. Verilog var uegnet, og SystemVerilog har kommet senere.
  - KDA ASIC'ene Nova1 og Nova2 ASIC'er med kommandostyrte testbenker.
  - KDA FPGA'ene CHP3 og Epos2 sine testbenker ikke kommandostyrte, men skriv/les makroer og makroer for start/stopp av funksjoner.
- Ulemper med daværende kommandostyrte løsning:
  - Bare mulig med ett nivå med kommandofiler.
  - Alle adresser i memory map for prosessor interface må VHDL kodes i testbenken hvis kommandotolker skal få tilgang til adressene.
  - Alle signaler som skal påtrykkes eller avleses i testbenken må VHDL kodes i testbenken hvis kommandotolker skal få tilgang til signalene.
  - Lite gjenbrukbart.

# Ønskeliste

KONGSBERG

- Lage et svært enkelt kommandospråk.
- Testbenken leser kommandoer fra tekstfiler.
- Inkludere kommandofiler i kommandofiler (dvs. rekursivt).
- Testbenken leser memory map package ved oppstart med en eller flere «map» kommandoer slik at alle processor interface (PIF) adresser som inngår i designet kan brukes i testbenkens kommandofiler.
- Ved oppstart leser testbenken sin egen VHDL kode slik at deklarerte signaler i testbenken kan brukes i kommandofilene.
- Det bør også være mulig å bruke signaler i design filer (probing).
- Gjenbrukbare funksjoner for å effektivisere utvikling av testbenker.

# From INF3430: Bus Functional Model (BFM)



# Kommandospråk

KONGSBERG

- map filename -- Memory package filename or link (Linux) to file name
- include filename.txt
- m? –bh ADDR+offset value
  - mw –b ADDR value
  - mc –b ADDR value *-- returns TRUE if equal else FALSE*
  - mc –h ADDR+2 value
  - mr –h ADDR
- set signal value
- set vector value
- wait4 signal value *-- returns FALSE when timed out else TRUE*
- wait4 vector value

## Kommandospråk forts.

KONGSBERG

- check signal  value  *-- returns TRUE if equal else FALSE*
- check vector  value
- if  <when true> else <when false> end  *(else branch optional)*
- ifn <when false> else <when true> end  *(else branch optional)*
- report –n string
- report –e string
- run –c value
- quit
- finish
- break

# Kommandospråk videreutvikling

KONGSBERG

- mr –bh**w** ADDR+offset value
- mw –bh**wg** length ADDR+offset value (-g also for mc command)
  - *-- write to addr. RAM_ADDR to RAM_ADDR+3 values F000,F001,F002,F003*
  - mw –hg x'0004 RAM_ADDR x'F000
  - *-- check from addr. RAM_ADDR to RAM_ADDR+3 values F000,F001,F002,F003*
  - mc –hg x'0004 RAM_ADDR x'F000
- mc –bhw ADDR+offset value mask *-- returns TRUE if equal unmasked bits*
  *-- else FALSE. All bits unmasked default.*
- wait4 –bhw ADDR+offset value mask *-- memory check poll. Returns TRUE*
  *-- if equal unmasked bits else FALSE*
- check signal value mask *-- returns TRUE if equal unmasked bits else FALSE*
- inst –i path(arch) *-- include signals in entity and architecture to signal list*
- inst -d path(arch) *-- delete signals in entity and architecture from signal list*
- run –t value unit

# Memory Map Package

KONGSBERG

- Prosjekt VHDL memory map package for:
  - FPGA/ASIC syntetiserbar kode
  - Modul- og toppnivå testbenker
  - Kilde for SW memory map
- Eksempel:

```
subtype reg_addr is std_logic_vector(15 downto 0);
constant regA : reg_addr:= x”0000”;  -- RW U8,4
constant regB : reg_addr:= x”0001”;  -- RW U8,8
constant regC : reg_addr:= x”0002”;  -- WO U8,1
constant regD : reg_addr:= x”0004”;  -- RW U16,12
constant regE : reg_addr:= x”0008”;  -- RO U32,32
```

# Adresseliste generering

KONGSBERG

- **function addr_list_gen**(file log: text; infile : string) return addrmap_ptr;
  - Lager en lenket liste av alle adresser med navn og verdi i hvert element.
  - Returnerer peker til første element i lista som blir siste adresse i package.
- **procedure get_address**(

  variable address_name : in string;
  variable addrmap_list : in addrmap_ptr;
  variable address_value : out std_logic_vector;
  variable found : out boolean);

  - Søker i adresse lista og returnerer med adressens verdi som out parameter når adresser leses i mw/mc/mr kommandoer.
  - Alle adresser som brukes i kommando filene må ligge i en eller flere address map package
- Standard dynamisk liste generering med VHDL access type (Ashenden kap.17)
- VHDL tekstbehandling og std_developerskit.std_iopak

# Kommandotolker

KONGSBERG

```
procedure include( infile: string; forced_eof: inout boolean) is
    file_open(file_ok, command_file, infile, read_mode);
begin
    while not endfile(command_file) loop
        readline(command_file, command);
        readword(command, length, word);
        command_id:= new string'(word'low to length)));
        if command_id.all="include" then
            <les nytt filnavn fra kommando fil>
            include(filename, forced_eof);
        elsif command_id.all="mw" then
            <les type, addresse og data fra kommando fil>
            get_address(address_name, addrmap_list, address, found);
            mw(tsize, address, data);
        -- elsif  flere kommandoer .... ;
        else
            <ukjent kommando>
        end if;
    end loop;
    file_close(command_file);
end procedure include;
```

# Probe signaler

KONGSBERG

- Metode:
  - Lage lenket liste av signaler i testbenken
  - Elementene i lista har navn, type og vektor lengde.
    - Navn f. eks. ”bit1”, ”bit2”, ”vector1” eller ”vector2”.
    - Type f. eks. std_logic, std_logic_vector(1 downto 0), unsigned(2 downto 0), osv.
  - Søke i signal lista og returnere med signal type og lengde som out parametere når signal navn leses i for eksempel wait4 kommando.
  - Initialisere probe signal ved hjelp av ”**init_signal_spy**” eller ”**signal_force**” funksjonene i Modelsim.
- Problem:
  - Hvordan lese et signal navn som en tekst streng i kommando filen, og deretter konvertere det til en probe? Navnet kan være for eksempel av type std_logic, std_logic_vector, unsigned eller signed.

# Probe generering

KONGSBERG

- Alle signal typer må pga. probe signalet i Modelsim funksjonene være deklarert i testbenken som default declarations:

```
signal sl_spy          : std_logic;
signal slv1_sp         : std_logic_vector(0 downto 0);
signal slv2_spy        : std_logic_vector(1 downto 0);
signal slv3_spy        : std_logic_vector(2 downto 0);
….
signal slv128_spy      : std_logic_vector(127 downto 0);
```

- Søker i signal lista og finn navnet, typen og lengden.
- Generer probe signalet ved å kalle på funksjon for init_signal_spy eller signal_force; f.eks:

```
If <std_logic_vector og lengde=2> then
        init_signal_spy(name, slv2_spy'simple_name);  -- Bruker ´simple_name pga. konverterer til text
elsif <andre typer og lengder> ….
```

- På denne måten vil alle std_logic_vector signaler av lengde 2 bli slv2_spy signal. Signalet slv2_spy blir deretter parameter til f. eks. wait4 funksjonen.
- Dette blir en god del kode, men kan gjenbrukes i alle testbenker uten endring.

# Probe generering bieffekt

KONGSBERG

## MEN:

- Init_signal_spy og signal_force oppretter ny FLI prosess hver gang den kalles, og alle FLI prosessene vil forbli aktive under hele simuleringen.
- Det fører til økt memory bruk som jeg har målt til ~365 bytes/probegenerering og litt økt simuleringstid.
- Det vil gi 365 Mbytes for 1 mill. kall på init_signal_spy funksjonen.
- Dette unngås ved å kode navnene for de signalene som brukes ofte direkte i set, check og wait4 kommandoene (vanligvis unødvendig):

```
If name="signal_name1" then
      < use signal_name1 in check value procedure>
elsif name="signal_name2" then
      < use signal_name2 in check value procedure>
elsif <std_logic> then
       init_signal_spy(name, sl_spy'simple_name);
       < use sl_spy signal in check value procedure >
elsif <andre typer og lengder> ….
```

# Testbenk

KONGSBERG

<use address map design package>

<use testbench package>

<use KDA reuse package>

**architecture** driver **of** tb_top **is**

  <signal declarations>

  <component declarations: **UUT** and **verification** components>

  <signal declarations for UUT and verification components>

**begin**

    <component instantiations>

    tb_command_0: **process**

        <variable declarations>

        <subroutine and function declarations>

## Testbenk forts.

KONGSBERG

```
begin
   addrmap_list:= addr_list_gen(log, ”addrmap_package”);
   probe_list:= probe_list_gen(log, ”testbench”);
   include(”start.cmd”, forced_eof);
   if (error_no>0) then
     <report number of errors to log file>
   else
     <report simulation successful to log file>
   end if;
end process tb_command_0;

sim_limit: process
   <terminate simulation if longer than specified time limit! >
```

# Subprogrammer

KONGSBERG

- Robin gjenbrukspakke (robin_pck.vhd + robin_bdy.vhd):
  - addr_list_gen -- lager adresselisten
  - get_address -- returnerer adresseverdi hvis den finnes
  - probe_list_gen -- lager signal listen
  - get_probe -- returnerer signal med type og evnt. lengde hvis det finnes
  - rm_space -- fjerner blanke i en linje fra en fil frem til første ikke blanke
  - readword -- leser neste ord frem til neste blanke
  - find_char -- søker i linjen frem til spesifisert tegn
  - writef -- skriver melding til log. fil
  - div. for klokkesignal generering, strobesignal generering og string behandling

# Subprogrammer forts.

KONGSBERG

- Testbenk prosjekt pakke (chip_tb_pck.vhd + chip_tb_bdy.vhd):
  - PifWrite -- Prosessor IF skriv
  - PifRead -- Prosessor IF les
  - PifCheck -- Prosessor IF les og sjekk mot forventet verdi
- Funksjonene kalles fra procedures i kommando tolkeren:
  - MW(tsize, address, data) -- Memory Write
  - MR(tsize, address, data) -- Memory Read
  - MC(tsize, address, data) -- Memory Check
- Subprogrammer i testbenk ”tb_command” process:
  - MW, MR, MC, include, set, check og wait4.
- PifWrite, PifRead og PifCheck funksjonene må tilpasses prosessoren i systemet (PowerQUICC, ARM, SPI protokoll, MicroBlaze, osv.).

# PIF write 1

KONGSBERG

```
-- uP Write
    procedure MW (
      constant tsize    : in  transaction_size;
      constant addr     : in  std_logic_vector(ADDRESS_LENGTH-1 downto 0);
      constant data     : in  std_logic_vector(DATA_LENGTH-1 downto 0)) is
    begin
      PifWrite (tsize, addr, data,
                mclk,
                cs_n, rw, we_n, oe_n, psdval_n, ain, d, ta_n, log, cycle_no);
    end procedure MW;
```

# PIF write 2

KONGSBERG

```
procedure PifWrite (
  constant tsize    : in  transaction_size;              -- Halfword/Upper/Lower size
  constant addr     : in  std_logic_vector(ADDRESS_LENGTH-1 downto 0);   -- EPOS local address
  constant data     : in  std_logic_vector(DATA_LENGTH-1 downto 0);   -- data
  signal   clk      : in  std_logic;
  signal   cs_n     : out std_logic;                     -- Chip select
  signal   rw       : out std_logic;                     -- read/write access
  signal   we_n     : out std_logic_vector(1 downto 0);    -- write access
  signal   oe_n     : out std_logic;                     -- output ena. in read access
  signal   psdval_n : out std_logic;                     -- read access termination
  signal   ain      : out std_logic_vector(ADDRESS_LENGTH-1 downto 0);   -- address
  signal   d        : out std_logic_vector(DATA_LENGTH-1 downto 0);   -- ext. data out
  signal   ta_n     : in  std_logic;                     -- write complete
  file     log      :     text;
  signal   cycle    : in  natural ) is
begin
  wait until rising_edge(clk);
   …..
end procedure PifWrite;
```

# PIF write 3

KONGSBERG

```
cs_n <= '0' after cs_delay;
rw   <= '0' after rw_delay;
psdval_n <= '1' after rw_delay;
ain  <= addr after rw_delay;
d    <= data after rw_delay;
we_n <= (others => '1');
if (tsize=BYTE and addr(0)='0') or tsize=HALFWORD then
  we_n(0) <= '0' after we_delay;
end if;
if (tsize=BYTE and addr(0)='1') or tsize=HALFWORD then
  we_n(1) <= '0' after we_delay;
end if;
wait on clk until ta_n='0';
we_n <= (others => '1');
d <= (others => 'Z') after deactivate_delay;
wait on clk until ta_n=PULL_UP or ta_n='1';
writef(log, cycle,"Pif Write to   Stanley" & " @ " &
      lv2strx(addr,ADDRESS_LENGTH) & "<= " & lv2strx(data,16));
cs_n <= '1' after deactivate_delay;
rw   <= '1' after deactivate_delay;
ain  <= (others => '0') after deactivate_delay;
Tcycle(2);
```

# Eksempel: start.cmd & setup.cmd

KONGSBERG

```
map chip_addrmap
include setup.cmd
include rw.cmd
report -n Simulation completed
```

```
set arst_n 0
run -c 100
set arst_n 1
report -n Reset done
wait4 irq_n 0
ifn
  report -e Waited for clock stable interrupt. Simulation terminated
  quit
end
mw -b PIF_CLK_STATUS x'00
wait4 irq_n 1
if
  report -n Master clock stable
else
  report -e Waited for deactivation of clock stable interrupt.
       Simulation terminated
  quit
end
```

```
NOTE: Invalid addresses in address map package:     0
NOTE: Invalid data in address map package:    10
NOTE: Invalid probe signals in testbench:     3
    0     Start of simulation
    0       NOTE: Open file: setup.cmd
    0     Set signal arst_n to value: 0
  100   Set signal arst_n to value: 1
  100     NOTE: Reset done
61747  Waited until signal irq_n got value: 0
61758  Pif Write to   Stanley @ 0000010<= 0000
61760  Waited until signal irq_n got value: 1
61760    NOTE: Master clock stable
61760    NOTE: Close file: setup.cmd
   :
   :
   :
```

# Eksempel: rw.cmd med feil

KONGSBERG

```
mw -h ADIF_DUMMY x'555a
mc -h ADIF_DUMMY x'555a
mw -h ADIF_DUMMY x'aaa5
mc -h ADIF_DUMMY x'aa55
set hp_clk 0
run -c 1
check hp_clk 0
if
    report -n hp_clk signal set to zero.
else
    report -e hp_clk signal not set to zero.
end
set hp_clk 1
run -c 1
check hp_clk 0
ifn
    report -e hp_clk signal not set to one.
else
    report -n hp_clk signal set to one.
end
set hp_clk2 1
run -c 1
check hp_clk2 0
```

```
    :
    :
61760    NOTE: Open file: rw.cmd
61771 Pif Write to   Stanley @ 0020000<= 555A
61782 Pif Read  from Stanley @ 0020000<= 555A
61798 Pif Write to   Stanley @ 0020000<= AAA5
61809 Pif Read  from Stanley @ 0020000<= AAA5
61814    ERROR! data read: A5, expected : 55
61814 Set signal hp_clk to value: 0
61815 Checked signal hp_clk with value: 0 expected value: 0
61815    NOTE: hp_clk signal set to zero.
61815 Set signal hp_clk to value: 1
61816 Checked signal hp_clk with value: 1 expected value: 0
61816    ERROR: hp_clk signal not set to one.
61816    ERROR: Invalid set signal command with unknown signal:
                    hp_clk2 to value: 1
61817    ERROR: Checked signal hp_clk2, but signal not found!
61817    NOTE: Close file: rw.cmd
61817    NOTE: Simulation completed
62053    ERROR: Simulation failed! Number of errors: 4
```

# Eksempel: rw.cmd uten feil

KONGSBERG

```
mw -h ADIF_DUMMY x'555a
mc -h ADIF_DUMMY x'555a
mw -h ADIF_DUMMY x'aaa5
mc -h ADIF_DUMMY x'aaa5
set hp_clk 0
run -c 1
check hp_clk 0
if
    report -n hp_clk signal set to zero.
else
    report -e hp_clk signal not set to zero.
end
set hp_clk 1
run -c 1
check hp_clk 1
ifn
    report -e hp_clk signal not set to one.
else
    report -n hp_clk signal set to one.
end
set hp_clk 1
run -c 1
check hp_clk 1
```

```
    :
    :
61760    NOTE: Open file: rw.cmd
61771 Pif Write to   Stanley @ 0020000<= 555A
61782 Pif Read  from Stanley @ 0020000<= 555A
61798 Pif Write to   Stanley @ 0020000<= AAA5
61809 Pif Read  from Stanley @ 0020000<= AAA5
61814 Set signal hp_clk to value: 0
61815 Checked signal hp_clk with value: 0 expected value: 0
61815    NOTE: hp_clk signal set to zero.
61815 Set signal hp_clk to value: 1
61816 Checked signal hp_clk with value: 1 expected value: 1
61816    NOTE: hp_clk signal set to one.
61816 Set signal hp_clk to value: 1
61817 Checked signal hp_clk with value: 1 expected value: 1
61817    NOTE: Close file: rw.cmd
61817    NOTE: Simulation completed
62053    NOTE: Simulation successful!
```

# Konklusjon

KONGSBERG

- Laget et enkelt og fleksibelt kommandospråk.
- Ingen endring av VHDL testbenk ved endringer i adresse map packager.
- Ingen endringer av testbenk ved nye probe signaler, men ved meget lange simuleringer må mye brukte signalnavn kodes i VHDL funksjonene check, wait4 og set for å unngå økt memory bruk og økt simuleringstid.
- Gjenbrukbare funksjoner.
- Brukt i mange FPGA prosjekter.