

THE MHDL / VHDL CONNECTION

Lorna Carmichael
Jared Brodsky
U.S. Army Research Laboratory
Ft. Monmouth, N.J. 07703
Attention: AMSRL-EP-MA
lorna@monmouth-etdl1.army.mil brodsky@monmouth-etdl1.army.mil

Abstract

Techniques to capture complete systems are currently being investigated. The concept of a single hardware description language for all aspects of hardware design is euphoric. This paper addresses a plausible solution for complete system design. Use of the MIMIC and VHSIC Hardware Description Languages to represent and describe a typical system problem is presented. Particularly, the paper focuses on the use and interaction of the MHDL and VHDL interface.

I. INTRODUCTION.

As the complexity of electronic systems increase, and size is compacted, greater demands are imposed on CAD tools and the users which support the systems. In one case, tools and techniques familiar to the engineer are no longer sufficient to approximate behavior of the system, requiring the engineer to become more knowledgeable of the tool in order to take full advantage of it. On the other hand, the tools are no longer sufficient to perform the desired operations. Nevertheless, the user is constantly required to learn more and more, be it increasing knowledge of a familiar tool or mastering a new tool. Faster clock and bus speeds, circuit compaction, new material technology, and multi-domain chips require description techniques beyond the scope and ability of any current single language. MHDL is offered as a partner in tackling such problems. Indeed, MHDL is another "tool" to learn; the challenge is to make the learning curve as small as possible.

II. MHDL OVERVIEW.

MHDL is an IEEE effort to establish a common language for the description, documentation, and design of electronic systems. Initially, the effort began under sole sponsorship of ARPA and DoD in 1988. MHDL is now under the auspices of IEEE Standards Coordinating Committee (SCC) 30 for further development and standardization, with continued support from ARPA and DoD. Interfaces to existing CAD tools (Success and WREN) and language specific tools (i.e., language-sensitive editor, syntax checker, evaluator, analyzer, and librarian) are currently sponsored by ARPA.

MHDL is a functionally based, object-oriented language, borrowing most of its concepts from the functional language Haskell. Functional languages treat functions the same as any other parameter; functions can be returned as values. MHDL is also an objected oriented language; it is hierarchical, it supports polymorphism, inheritance(multiple), and abstract data types. Some of the majors features and elements of the language (printed in ***bold italic***) are: ***Models***: represents the design entity; ***Packages***: group related definitions for multiple use by other ***packages*** and/or ***libraries***; ***Libraries***: group of MHDL ***models*** or ***packages***; ***Components***: an actual use of a ***structure***, representing a tangible object; ***Structures***: describes the contents of models and their associated connections; ***Connectors***: the external connections of the ***model***, representing information flow; ***Configuration***: identifies the ***structure*** to be used for a ***component model***; ***Signals***: values with associated physical unites; ***Constraints***: specifies an assertion to be tested; ***Alternatives***: one or more mutually exclusive selections. A detailed description of MHDL can be found in [2].

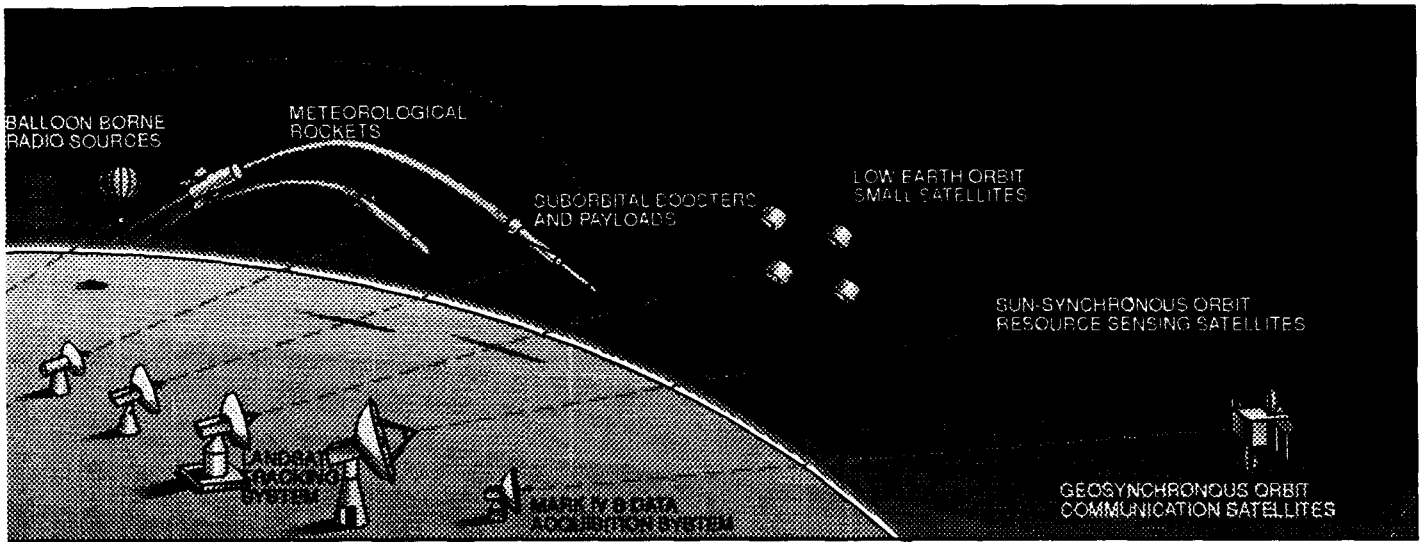


Figure 1. Geosynchronous Orbiting Satellite System.

The MHDL-VHDL interface and naming conventions are described in [2]. VHDL language constructs will be represented in *italic print*. A MHDL *model* can instantiate a VHDL *component* or a VHDL *entity* can instantiate a MHDL *component*. MHDL recognizes a VHDL *entity* by assigning the name of the VHDL *entity-architecture* pair to a MHDL *model* using an *abstract property definition*:

attribute VHDL_entity = 'entity_arch_pair_name';

In the case of a VHDL *entity* instantiating a MHDL *component*, the MHDL *model* serves as the *architecture* for the VHDL *entity* and MHDL must provide initial values for the *signals* on the VHDL *ports* and MHDL *connectors*. In either case, *connector* names in MHDL must correspond and be named the same as *ports* in VHDL and the *attributes* on the *connectors* in MHDL must be compatible with the *types* of the VHDL *ports*. The example presented captures the system in MHDL and utilizes VHDL *components*.

III. TVRO TRACKING SYSTEM.

The example presented is based on a real satellite system (figure 1) but has been simplified to illustrate the interfacing of MHDL and VHDL. A television receive only (TVRO) system, receives video/audio signals from a geosynchronous satellite orbiting 35,881 kilometers from the equator. Signal emissions from the satellite are weak and power limited, suffer from path loss, atmospheric signal attenuation, and degradations due to the receiving hardware such as, antenna

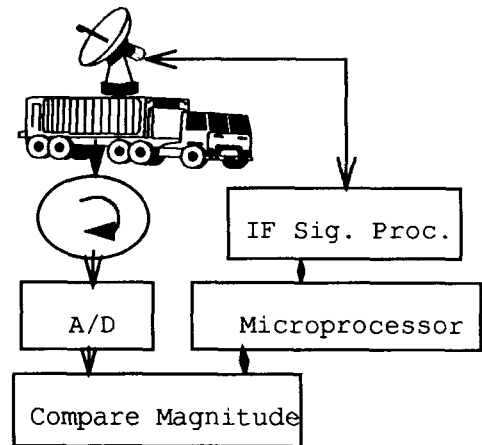


Figure 2. TVRO Block Diagram

pointing accuracy. Following path loss, possible interference and carrier-to-noise ratio are key factors in received signal quality. Poor signal quality would generate shaky images or "sparklies." If the receiving antenna could always point directly at the satellite, loss could be minimized.

Consider a mobile TVRO, mounted on a truck. As the truck travels, the antenna must be constantly realigned with the satellite in order to have maximum signal strength. The TVRO tracking system accomplishes this task by splitting the received signal, digitally converting the amplitude spectrum of the signal, and comparing it to a previously know maximum signal

strengths. The antenna position is then adjusted to maintain maximum signal strength. A block diagram of the simplified system is presented in fig. 2.

The antenna receives a radio frequency (RF) signal, which is connected to a power splitter and conditioner. A portion of the signal is guided to a RF signal processing block, and the other, to an A/D converter. The amplitude spectrum (as a result of signal conditioning) of the signal is digitally converted to a corresponding power level to check the intensity of the signal. The converted signal is compared to a given minimum signal strength, which corresponds to an acceptable picture quality. If the signal falls below the minimum signal strength, the system begins to realign the antenna with the satellite. Alignment states are 'A', 'B', 'C', and 'D', which corresponds to search right, search left, strong signal, and reset coordinates. Based on the current system position and known satellite position, a fairly good estimate of the proper antenna positioning can be determined (the exact position if the system was stationary). The output of the searching/tracking processor is connected to the antenna motor control, which positions the antenna. The antenna is positioned by an actuator. For each revolution in the screw (positioning arm), a Reed switch generates a pulse. Thus a corresponding digital word determines the positioning of the antenna.

Models for the actuator arm, motor, controller, and intermediate frequency (IF) signal processor block are not presented in this publishing, they are of little significance to the MHDL/VHDL interface. However, behavioral models of the Reed switch and A/D converter are of interest.

```

model TVRO_Sys
  includes VHDL;
  includes electrical;
  structure TVRO_struct
    VHDL_entity = decode1_4;
    VHDL_entity = sig_generator;
    connectors rco_antenna, output;
  components
    ASys :: antenna_sys.antenna_struct;
    A_D :: Ana_to_Dig.A_to_D_struct;
    Split :: Split_cond.split_struct;
    Decode1 :: decode1_4;
    Sig1 :: sig_generator;
    IF_proc :: IF_sig_proc.IF_struct;
  end components;

```

```

  connect ASys.antenna_struct.receive,
    rco_antenna;
  connect A_D.A_to_D_struct.analog_in,
    Split.split_struct.out2;
  connect A_D.A_to_D_struct.digital_out,
    Decode1.decode1_4.select_out;
  connect Split.split_struct.in1,
    ASys.antenna_struct.sig_out;
  connect Split.split_struct.out1,
    IF_proc.IF_struct.IH_in;
  connect ASys.antenna_struct.reed_sw_out,
    Sig1.sig_generator.pulse_level;
  connect ASys.antenna_struct.motor_ctrl,
    output;
end TVRO_struct;
end TVRO_Sys;

```

```

model antenna_sys
  includes electrical;
  signal rf_in: Time -> Float;
  structure antenna_struct
    connectors receive, sig_out, motor_ctrl,
    reed_sw_out;
  -- receive.rf_in = continuous and periodic
  -- sig_out.rf_in = signal after low noise
  -- amplification, down conversion, and filtering.
  components
    A_sys :: ant_sub_sys.rf_process;
    act1 :: actuator.vonweiss;
  end components;
  connect act1.base, A_sys.rf_process.positioner;
  connect receive, A_sys.rf_process.sig_in;
  connect motor_ctrl, act1.vonweiss.ctrl_lines;
  connect reed_sw_out, act1.vonweiss.reed_out;
  end Antenna_struct;
end antenna_sys;

```

```

model ant_sub_sys
  includes IF_process;
  includes VHDL;
  structure sub_sys_struct
    connectors proc_ctrl, positioner, sig_in, sig_out;
  -- other components, definitions, descriptions
  end sub_sys_struct;
end ant_sub_sys;

```

```

model actuator
  structure thomson_sagniaiw_ppa
    -- definitions/descriptions
  end thomson_sagniaiw_ppa;
  structure vonweiss
    connectors pw_supply, gnd, reed_out,
    ctrl_lines, base;

```

```

components
  arm :: act_arm.acme;
  mtr :: motor.elect
  definitions
    mtr.elect.vsupply = 36 'v';
  end mtr;
  reed_sw :: switch.reed_struct;
  central_ctrl:: controls.act_ctrl;
end components;
-- more definitions/descriptions
connect reed_out, reed_sw.reed_struct.out;
connect base, arm.acme.mount;
connect central_ctrl, cw1, mtr.elect.red;
connect gnd, mtr.black, mtr.elect.white;
connect central_ctrl.cw2, reed_sw.reed_struct.on;
end vonweiss;
end actuator;

```

```

model reed_sw
includes VHDL;
attribute rev : Float;
attribute reed : VHDL_Signal (Float);
structure reed_struct
  connectors pw, out;
  VHDL_entity = sig_generator;
end reed_struct;
end reed_sw;

```

```

model split_cond
includes electrical;
structure split_struct
  connectors in1, out1, out2;
  signal rf, temp_sig : Time -> Float;
  out1.rf = 0.43 * temp_sig;
  out2.rf = sqrt((realPart(out1.rf) +
    imagPart(out1.rf)) *
    (realPart(out1.rf) - imagPart(out1.rf)));
  end P_split_struct;
end split_cond;

```

```

model Ana_to_Dig
includes electrical;
includes VHDL;
attribute Sperm: Time;
attribute thresh: Float;
attribute tmp_sig: Time -> Float;
attribute select_out: VHDL_Signal (fourletter);
type fourletter = 'A' | 'B' | 'C' | 'D';
samp:: (Time -> a) -> Time -> [(Time, a)];
samp (f, per) = [ (per * x, f (per * x)) |
  x <- [0 .. ] ];
threshold:: [(Time -> a)] -> a
-> VHDL_Signal (fourletter);

```

```

threshold ([], th) = [];
threshold ([x], th) = [];
threshold (((t1,v1) :- x@((t2,v2) :- xs)], th) =
  if v1 < th && v2 > th then (t1, 0, 'A') :-
threshold (x, th)
  else if v1 > th && v2 < th then (t1, 0, 'B') :-
threshold (x, th)
  else if v1 > th && v2 > th then (t1, 0, 'C') :-
threshold (x, th)
  else if v1 < th && v2 < th then (t1, 0, 'D') :-
threshold (x, th)
  else threshold (x, th);
structure A_to_D_struct
  connectors analog_in, digital_out;
  VHDL_entity = decode1_4;
  select_out = threshold ( samp ( tmp_sig, Sperm),
thresh);
end A_to_D_struct;
end Ana_to_Dig;

```

----- VHDL CODE-----

```

PACKAGE four is
  TYPE fourletter IS ('A', 'B', 'C', 'D');
END four;
LIBRARY IEEE;
USE IEEE.STD_logic_1164.ALL
USE WORK.four.ALL;
TYPE fourletter IS ('A', 'B', 'C', 'D');
ENTITY decode1_4 IS
  PORT (select_out: IN fourletter;
    enable: IN BIT := '0';
    q0: OUT std_logic;
    q1: OUT std_logic;
    q2: OUT std_logic;
    q3: OUT std_logic);
END decode1_4;
ARCHITECTURE resolve4 OF decode1_4 IS
BEGIN
  q0 <= 'H' WHEN select_out = 'A' ELSE 'L';
  q1 <= 'H' WHEN select_out = 'B' ELSE 'L';
  q2 <= 'H' WHEN select_out = 'C' ELSE 'L';
  q3 <= 'H' WHEN select_out = 'D' ELSE 'L';
END resolve4;

ENTITY sig_generator IS
  PORT ( rev: IN real;
    pulse: OUT real:= 0.0);
END sig_generator;
ARCHITECTURE generation OF sig_generator IS
CONSTANT period: Time:= 2 ns;
CONSTANT halfperiod: Time:= 1 ns;
CONSTANT pulse_level: integer:= 5;

```

```

BEGIN
PROCESS
  VARIABLE i: real;
  BEGIN
    i:= rev;
    WHILE i > 0.0 LOOP
      pulse <= real(pulse_level), 0.0 AFTER halfperiod;
      WAIT FOR period;
      i:= i - 1.0;
      IF (i < 1.0 AND i > 0.5) THEN
        pulse <= real(pulse_level), 0.0 AFTER halfperiod;
        i:= 0.0;
      ELSIF (i < 0.5 AND i > 0.0) THEN
        pulse <= real(pulse_level);
        i:= 0.0;
      ELSE
        null;
      END IF;
    END LOOP;
    WAIT FOR period;
  END PROCESS;
END generation;

```

In *model Ana_to_Dig*, the “includes VHDL” statement, includes the *package* VHDL which is required in describing the MHDL and VHDL descriptions. The *package* defines the MHDL *type* for a VHDL *signal*. The *type* is a three tuple list of time, delta cycle, and native VHDL *type*. Although in VHDL, this three tuple list is hidden from the user, MHDL must recognize it. The *abstract property definition* for *select_out* uses *package* VHDL to define its *type*. The enumerated type “fourletter” must be defined in both MHDL and VHDL. A sampling function was defined to convert the continuous signal to a discrete signal. The function threshold actually generates the appropriate VHDL *signal*. Note, the delta cycle is set to zero for the function. The generated VHDL *signal* and all of its updates are present at the VHDL interface at the beginning of the VHDL simulation cycle. Timing is the only noted concern for the interface. Delta cycles will have to be carefully selected to avoid posting values at VHDL ports improperly. A more complex and tightly integrated, e.g., sending VHDL *signal* values to a MHDL *model*, may require accurate specification of the delta cycle. Specification of the delta cycle should be taken into account by the discrete to continuous conversion function.

V. CONCLUSION.

The integration of the MIMIC and VHISIC hardware description language establishes a forum for a seamless modeling environment for systems incorporating analog and digital components. The simple, well defined interface between MHDL and VHDL should allow the description and modeling of more complicated structures easier. The organization and many of the MHDL language constructs are quite similar to VHDL. There is an almost one to one correspondence (i.e. VHDL *entity* and MHDL *model*, VHDL *architecture* and MHDL *Structure*, VHDL *assert* and MHDL *constraint*). Although few differences were found, some harmonization is desired. For example, subtle differences such as, the type specification for real numbers in VHDL is *real* and in MHDL it is *Float*. Additionally, recognition of the MHDL/VHDL interface and adoption of standard naming conventions by the VHDL community would aid in standardization of the interface. Once harmonization has occurred, learning an additional language would not be as difficult.

ACKNOWLEDGMENTS.

We express sincere gratitude to our fellow co-workers Eric Lenzing, Elmer Freibergs, Arnie Bard and Gerald Michael of ARL EPSPD; and especially Dave Barton of Intermetrics for his MHDL/VHDL expertise.

REFERENCES.

- [1] S.Prentiss, TVRO Technology. Prentice Hall, Englewood Cliffs, NJ 1989.
- [2] MHDL Language Reference Manual, V2.0, Intermetrics, Inc. March 1994.
- [3] IEEE VHDL Language Reference Manual, Std. 1076-1987, IEEE press, March 31, 1988.
- [4] Department of Defense, DARPA/Tri-Service MHDL Requirements Document. U.S. Army, LABCOM ETDL, 1991.
- [5] L. Carmichael, A. Mangum, "Automating the design process: A design process model implementation," Autotescon '94, Proceedings of the IEEE Systems Readiness Technology Conference 1994.
- [6] D.L. Perry, VHDL. McGraw-Hill 1991.