

VDFL: A Language for the Specification of DSP Systems

George S. Powley Jr.
powley@ee.eng.ohio-state.edu

Dr. Joanne E. DeGroat
degroat@ee.eng.ohio-state.edu

Department of Electrical Engineering
The Ohio State University
2015 Neil Avenue
Columbus, OH 43210-1272

Abstract

VDFL is a specification language developed to describe DSP systems.[†] Since the design of DSP systems is moving to higher levels of abstraction, VDFL allows DSP systems to be specified at a higher level of abstraction. The VDFL specification is translated into floating-point and fixed-point VHDL models for system simulation. This allows finite wordlength effects to be investigated. VDFL is also translated into a signal flow graph, for input to a behavioral synthesis tool. As the design moves towards implementation, the VDFL model can be used to validate lower level VHDL models. An example is presented and extensions for multirate systems and sequential constructs are discussed.

1 Introduction

As behavioral synthesis tools move the design of digital systems to higher levels of abstraction, the design specification language should also move to higher levels of abstraction. A higher level specification medium allows the designer to describe the design in a more natural format. An example of this is seen in state diagram editors. Although VHDL is capable of directly describing finite state machines, some designers feel more comfortable using a state diagram format. For DSP systems, Silage[1] and its commercial counterpart DFL[2] are more natural languages for design specification.

VHDL Data Flow Language (VDFL) is a specification language developed to describe Digital Signal Processing (DSP) systems. VDFL combines a

subset of VHDL with extensions from Silage/DFL to create a powerful and flexible DSP specification language. Since VDFL is not a simulation language language translators are used to convert VDFL to VHDL for simulation[3].

VDFL code is translated to fixed-point and floating point VHDL models, allowing finite wordlength effect to be investigated. VDFL is also translated into a signal flow graph (SFG) for input into a behavioral synthesis system. As the design moves towards implementation the VDFL model can be used to validate the lower-level VHDL models produced by behavioral synthesis and logic synthesis tools.

The current version of VDFL supports the description of single rate systems using concurrent signal assignments. In this paper, the motivation for VDFL is discussed, the language is described, and an example is presented. Extensions for multirate systems and sequential statement descriptions are also discussed.

2 Motivation

The design of DSP algorithms usually takes place at a high level of abstraction, by considering the manipulation of data streams. A natural format for the description of DSP algorithms is the Signal Flow Graph (SFG) [4]. An SFG contains nodes, which represent operations, and edges, which represent data streams connecting these operations. The SFG does not specify how operations are bound to processors or how data streams are bound to storage. Silage and DFL attempt to allow a more direct rendering of designs at this level.

VHDL has been shown to be an effective design environment for DSP [5]. VHDL can be used to model DSP algorithms at floating-point, fixed-point, or implementation levels. VHDL's ability to model all level

[†] This research has been funded by Harris Semiconductor, Melbourne, FL.

of the design process supports a top-down design methodology for the design of DSP systems.

The motivation for developing VDFL is to provide a DSP specification language with the features of DFL which will take advantage of VHDL as a powerful simulation language. There are several advantages to using a combination of these languages, as opposed to using VHDL or DFL alone, including:

- Using the DFL delay clause reduces the amount of programming overhead required to describe the same construct in VHDL.
- Using a subset of VHDL limits the number of VHDL constructs that must be supported by a behavioral synthesis tool.
- Using generics and the VHDL simulation engine provides more flexibility than is available from DFL alone.

VDFL is being developed as a language for a synthesis-driven design methodology. Therefore, the language designers must consider the behavioral synthesis aspects, as well as the specification and simulation aspects of the language. Designing the language in this manner results in a powerful specification and simulation language, which can be used to produce high quality behavioral synthesis results.

3 Language Description

The syntax of VDFL closely matches the syntax of VHDL[6]. This simplifies the translation of VDFL to VHDL, because many of the program constructs will not need to be changed during translation. Matching the syntax of VHDL also allows a VDFL parser to be easily created. Modifying an existing VHDL parser to handle the extensions from DFL and the limitations of a VHDL subset will produce a VDFL parser which can be used for language translation. In this section, a description of the current version of the VDFL language is provided.

3.1 Lexical elements

Lexical elements in VDFL correspond directly to lexical elements in the VHDL language, with some additional operators. A summary of the VDFL lexical elements is shown in Figure 1. By using the same lexical element definitions as VHDL, all identifiers, keywords, constants, and operators defined in a VDFL specification will be valid in the VHDL simulation models.

Lexical Element	Description
Identifiers	VHDL Identifiers (case insensitive)
Keywords	VHDL keywords VDFL types
Constants	VHDL literals
Operators	VHDL operators VDFL operators

Figure 1: VDFL Lexical Elements

3.2 Data types

Signals defined in a VDFL specification must use one of the types shown in Figure 2. These types correspond to the signal types defined in DFL. During source translation, the VDFL types are transformed into VHDL types for simulation.

For DFL float types, *m* corresponds to the mantissa length of a signal and *e* corresponds to its exponent length. Although DFL supports the specification of the mantissa and exponent length for floating-point numbers, the support for simulation of this feature is implementation dependent. Currently, VDFL does not support specification of the mantissa and exponent length for floating-point numbers, since it is not supported by VHDL simulation. This feature could be added to VDFL and supported using VHDL floating-point operations[7].

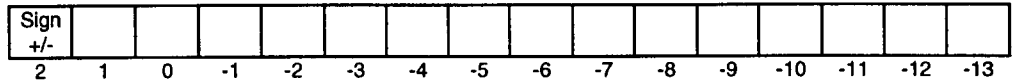
DFL	VDFL	VHDL
float< <i>m</i> , <i>e</i> >	real	real
fix< <i>w</i> , <i>f</i> >	fix< <i>w</i> , <i>f</i> >	fix(<i>w</i> - <i>f</i> -1 downto - <i>f</i>)
int< <i>w</i> >	int< <i>w</i> >	fix(<i>w</i> -1 downto 0)
unsigned< <i>w</i> , <i>f</i> >	ufix< <i>w</i> , <i>f</i> >	ufix(<i>w</i> - <i>f</i> -1 downto - <i>f</i>)
unsigned_int< <i>w</i> >	uint< <i>w</i> >	ufix(<i>w</i> -1 downto 0)
bool	bool	bool

Figure 2: VDFL Data Types

In DFL fixed-point types, *w* corresponds to the wordlength of the signal and *f* corresponds to its fraction length. Fixed point signal definitions in VDFL are translated into an array of STD_LOGIC in VHDL. By using type integer to define the array range, instead of type natural, the location of the binary point is preserved. An example fixed-point signal declaration, its bit representation, and corresponding VHDL signal declaration are shown in Figure 3.

Arrays of signals can also be declared in VDFL. In order to declare an array of signals, an array range is appended to the type indication, as shown in Figure 3.

Fixed-point signal representation:



VDFL fixed-point signal declaration:

```
SIGNAL x : fix<16, 13>;
```

Translated VHDL signal declaration:

```
SIGNAL x : fix(2 DOWNT0 -13);
```

VDFL signal array declaration:

```
SIGNAL a : fix<16,13>(0 TO 3);
```

Translated VHDL signal declaration:

```
TYPE a_T IS ARRAY (0 TO 3) OF fix(2 DOWNT0 -13);  
SIGNAL a : a_T;
```

Figure 3: Fixed-point Signal Declarations

The VDFL translator generates any intermediate array types required by the VHDL simulation models.

3.3 Operators

VDFL provides operators common to most programming languages. Unary operations, which require one operand, and binary operations, which require two operands, are provided. For fixed-point numbers, the result type of an operation is a function of the operand types and the operator. Operand type requirements and implicit type casting of results have been adopted from DFL, as shown in Figure 4.

VDFL also provides constructs unique to the description of DSP systems. The time delay clause **@**, brings the concept of samples into the specification language. For example, the following statement:

```
y <= x@4;
```

assigns to **y** the value of **x**, delayed by four samples. The time delay clause simplifies the specification of DSP algorithms compared to writing VHDL code to provide the same functionality, as we will show with an example in Section 4. A delay value can be initialized using a time delay initialization clause **@@**.

3.4 Expressions

An expression is a formula that defines the computation of a value. Expressions are composed of primaries with optional operators. A primary has a value and a type. The BNF description for VDFL primaries is:

```
primary ::=  
    name  
    | literal  
  
name ::=  
    ID [index][delay]  
  
index ::=  
    \(' simple_expression \)'  
  
delay ::=  
    \@' static_expression
```

There are three types of expressions defined in VDFL: unary, binary, and conditional. Unary expressions are expressions that take only one operand. Binary expressions are expressions that take two operands. The operators for these expressions are shown in Figure 4.

Conditional expressions define a signal value based on the result of condition expressions. This construct directly maps to the conditional signal assignment statement in VHDL.

Operator	Description	Left Operand	Right Operand	Result Type
NOT	Bitwise negation		fix<w, f> ufix<w, f>	fix<w, f> ufix<w, f>
+, -	Unary plus Unary minus		fix<w, f> REAL	fix<w, f> REAL
*	Multiplication	fix<w1, f1> ufix<w1, f1> REAL	fix<w2, f2> ufix<w2, f2> REAL	fix<w1+w2, f1+f2> ufix<w1+w2, f1+f2> REAL
/	Division	fix<w, f1> ufix<w, f1> REAL	fix<w, f2> ufix<w, f2> REAL	fix<w, f1> ufix<w, f1> REAL
+, -	Add Subtract	fix<w, f> ufix<w, f> REAL	fix<w, f> ufix<w, f> REAL	fix<w, f> ufix<w, f> REAL
>>, <<	Shift right Shift left	fix<w1, f> ufix<w1, f>	uint<w2> uint<w2>	fix<w1, f> ufix<w1, f>
<, >, <= >=, =, /=	Relational Operators	fix<w1, f1> ufix<w1, f1> REAL	fix<w2, f2> ufix<w2, f2> REAL	BOOL BOOL BOOL
AND, OR XOR	Logical Operators	fix<w, f> ufix<w, f>	fix<w, f> ufix<w, f>	fix<w, f> ufix<w, f>

Figure 4: VDFL Operators and Implicit Result Types

3.5 Generics and generate statements

Two other VHDL constructs have been adopted by the VDFL language: generics and generate statements. The generic construct provides a mechanism to pass static information to a VDFL model. By using generics, one VDFL specification can be used to model a large class of DSP algorithms. The generate statement allows iterative or conditional elaboration of concurrent statements. Generate statements provide a concise way to represent a large number of signal assignments.

In the next section, we present an example of how generics and generate statements are used to model a generic IIR filter.

4 Example

In this section, we present the VDFL specification for a generic IIR filter. The IIR filter in this example is the same filter structure used by Matlab for its filter design. Therefore, Matlab can be used to set the filter order and coefficients. The order and coefficients are passed to the VDFL model using generics, and the signal assignments are elaborated using generate statements. The VDFL specification is shown in Figure 5.

4.1 Floating-point VHDL model

A translation program is used to convert the VDFL specification to a floating-point VHDL model. During source translation, all fixed-point signals become floating-point signals, with type **REAL**. Operators that are not defined for type **REAL** are not supported in the floating-point VHDL model.

The floating-point VHDL model uses types and operators defined by VHDL. Therefore, no new signals are required to hold intermediate results.

The output of the VDFL to floating-point VHDL translation program is shown in Figure 6.

4.2 Fixed-point VHDL model

Another translation program is used to convert the VDFL specification into a fixed-point VHDL model. The fixed-point VHDL model uses the VDFL package to provide operations on fixed-point signals. The VDFL package provides the types, operations, rounding, and conversion routines to support fixed-point VHDL models. The VDFL package uses the IEEE NUMERIC_STD package[8] to provide arithmetic operations on **STD_LOGIC_VECTOR** signals.

The fixed-point VDFL model uses types and procedures defined in the VDFL package. Therefore, new

```

ENTITY dfII IS
  GENERIC(
    N      : INTEGER;
    a      : REAL();
    b      : REAL();
    width  : INTEGER;
    fraction : INTEGER
  );
  PORT(
    IN1 : IN  FIX<width, fraction>;
    OUT1 : OUT FIX<width, fraction>
  );
END dfII;

ARCHITECTURE vdf1 OF dfII IS
  SUBTYPE Data IS FIX<width, fraction>(0 TO N);

  CONSTANT coef_a : Data := a;
  CONSTANT coef_b : Data := b;

  SIGNAL prod_a : Data;
  SIGNAL prod_b : Data;
  SIGNAL sum    : Data;
BEGIN
  ALLN : FOR i IN 0 TO N GENERATE
    Least : IF i = 0 GENERATE
      sum(i) <= prod_b(i) + sum(i+1)@1;
    END GENERATE; -- Least
    Most : IF i = N GENERATE
      prod_a(i) <= coef_a(i) * sum(0);
      sum(i) <= prod_a(i) + prod_b(i);
    END GENERATE; -- Most
    Rest : IF i > 0 AND i < N GENERATE
      prod_a(i) <= coef_a(i) * sum(0);
      sum(i) <= prod_a(i) + prod_b(i) + sum(i+1)@1;
    END GENERATE; -- Rest
    prod_b(i) <= coef_b(i) * IN1;
  END GENERATE; -- ALLN
  OUT1 <= sum(0);
END vdf1;

```

Figure 5: VDFL Specification of an IIR Filter

signals must be created to hold intermediate values. The type of an intermediate signal is determined by the implicit result type of the corresponding operator.

The output of the VDFL to fixed-point VHDL translation program is shown in Figure 7.

4.3 VHDL simulation

A VHDL simulation was used to investigate the finite wordlength effects of a third order low-pass filter. The impulse response of the filter is shown in Figure 8. The graph shows the results for a floating-point simulation and two fixed-point simulations of `fix<16, 13>` and `fix<24, 20>`.

5 VDFL Extensions

The current version of VDFL is very useful for describing the functionality of single rate DSI algorithms, using concurrent signal assignments. Additional constructs could be added to VDFL to support multirate systems and sequential statements.

DFL contains additional constructs to describe multirate and irregular rate systems. The five functions added to support multirate systems are: `phasedelay`, `downsample`, `upsample`, `time_demux`, and `time_mux`. These functions are used to manipulate the period and phase of signals. Similar functions could be added to VDFL to provide the same functionality. In addition to adding these functions, the basic VHDL

```

USE WORK.VDFL.ALL;

ENTITY dfii IS
  GENERIC(
    N : INTEGER;
    a : REAL_ARRAY;
    b : REAL_ARRAY;
    width : INTEGER;
    fraction : INTEGER
  );
  PORT(
    CLK : IN BIT;
    IN1 : IN REAL;
    OUT1 : OUT REAL
  );
END dfii;

ARCHITECTURE vdf1 OF dfii IS
  SUBTYPE Data IS REAL_ARRAY(0 TO N);
  CONSTANT coef_a : Data := a;
  CONSTANT coef_b : Data := b;
  SIGNAL prod_a : Data;
  SIGNAL prod_b : Data;
  TYPE sum_D IS ARRAY(0 TO 1) OF Data;
  SIGNAL sum : sum_D;
BEGIN
  AllN : FOR i IN 0 TO N GENERATE
    Least : IF i = 0 GENERATE
      sum(0)(i) <= prod_b(i) + sum(1)(i + 1);
    END GENERATE;
    Most : IF i = N GENERATE
      prod_a(i) <= coef_a(i) * sum(0)(0);
      sum(0)(i) <= prod_a(i) + prod_b(i);
    END GENERATE;
    Rest : IF i > 0 AND i < N GENERATE
      prod_a(i) <= coef_a(i) * sum(0)(0);
      sum(0)(i) <= prod_a(i) + prod_b(i) + sum(1)(i + 1);
    END GENERATE;
    prod_b(i) <= coef_b(i) * IN1;
  END GENERATE;
  OUT1 <= sum(0)(0);

  -- delay process
  PROCESS(CLK)
  BEGIN
    IF (CLK = '1' AND CLK'EVENT) THEN
      sum(1) <= sum(0);
    END IF;
  END PROCESS;
END vdf1;

```

Figure 6: Floating-point VHDL Model

model would have to be changed to include period and phase information.

Sequential statements are more suitable for describing some algorithms, such as finding the maximum value of an array. Sequential statements and subprograms could easily be added to the VDFL language, using the VHDL definition of sequential statements.

6 Conclusion

In this paper, we have described a specification language for DSP systems. VDFL combines aspects of

VHDL and DFL to allow high level specification of DSP systems. Since VDFL is translated to VHDL for simulation, it takes advantage of commercially available VHDL simulators for powerful and flexible simulation. Floating-point and fixed-point models of the VDFL specification can be analyzed, in order to investigate fixed-point effects. The VDFL is also translated into a SFG, which is passed to a behavioral synthesis system.

The current version of VDFL supports single rate systems and concurrent signal assignments. Future work could include extending VDFL to support multirate systems and sequential constructs.

```

USE WORK.VDFL.ALL;
ENTITY dfii IS
  GENERIC(
    N : INTEGER;
    a : REAL_ARRAY;
    b : REAL_ARRAY;
    width : INTEGER;
    fraction : INTEGER
  );
  PORT(
    CLK : IN BIT;
    IN1 : IN FIXED(width-(fraction)-1 DOWNT0 -(fraction));
    OUT1 : OUT FIXED(width-(fraction)-1 DOWNT0 -(fraction))
  );
END dfii;

ARCHITECTURE vdf1 OF dfii IS
  TYPE Data IS ARRAY (0 TO N) OF FIXED(width-(fraction)-1 DOWNT0 -(fraction));
  SIGNAL coef_a : Data;
  SIGNAL coef_b : Data;
  SIGNAL prod_a : Data;
  SIGNAL prod_b : Data;
  TYPE sum_D IS ARRAY(0 TO 1) OF Data;
  SIGNAL sum : sum_D;
  TYPE temp_6_A IS ARRAY (0 TO N) OF FIXED(width-(fraction)-1 DOWNT0 -(fraction));
  SIGNAL temp_6 : temp_6_A;
BEGIN
  AllN : FOR i IN 0 TO N GENERATE
    Least : IF i = 0 GENERATE
      add(sum(1)(i + 1), prod_b(i), sum(0)(i));
    END GENERATE;
    Most : IF i = N GENERATE
      mult(sum(0)(0), coef_a(i), prod_a(i));
      add(prod_b(i), prod_a(i), sum(0)(i));
    END GENERATE;
    Rest : IF i > 0 AND i < N GENERATE
      mult(sum(0)(0), coef_a(i), prod_a(i));
      add(prod_b(i), prod_a(i), temp_6(i));
      add(sum(1)(i + 1), temp_6(i), sum(0)(i));
    END GENERATE;
    mult(IN1, coef_b(i), prod_b(i));
  END GENERATE;
  OUT1 <= sum(0)(0);

  -- signal delay process
  PROCESS(CLK)
  BEGIN
    IF(CLK = '1' AND CLK'EVENT) THEN
      sum(1) <= sum(0);
    END IF;
  END PROCESS;

  -- array initialization process
  PROCESS
  BEGIN
    FOR i IN a'RANGE LOOP
      toFixed(coef_a(i), a(i));
    END LOOP;
    FOR i IN b'RANGE LOOP
      toFixed(coef_b(i), b(i));
    END LOOP;
    WAIT;
  END PROCESS;
END vdf1;

```

Figure 7: Fixed-point VHDL Model

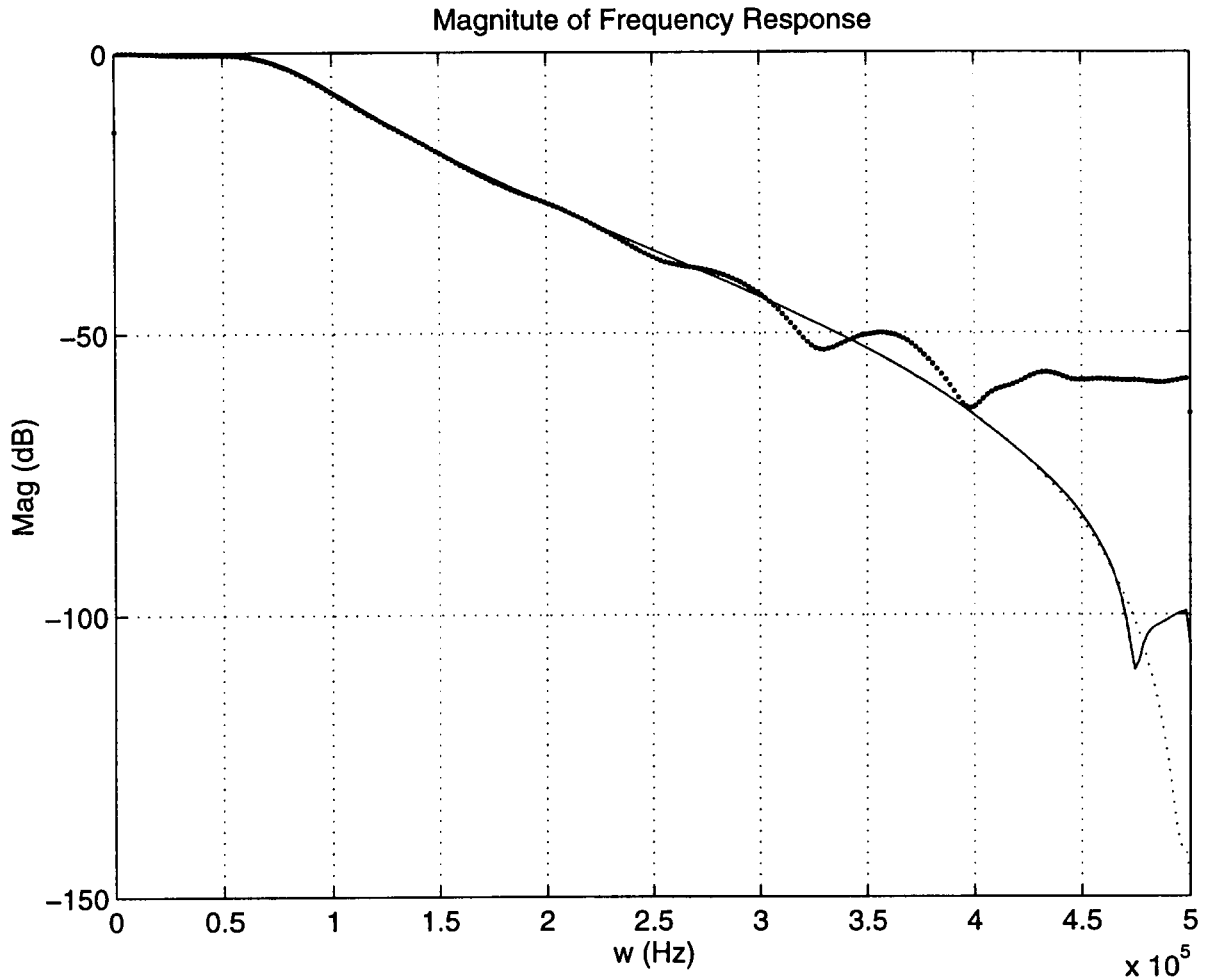


Figure 8: Impulse response for IIR Filter (Floating-point, 24 bit, and 16 bit data)

References

- [1] D. Genin, P. Hilfinger, J. Rabaey, C. Scheers, and H. De Man, "DSP Specification Using the Silage Language," *International Conference on Acoustics, Speech and Signal Processing*, vol. 2, 1990.
- [2] Mentor Graphics, Corp., *DSP Architect DFL User's and Reference Manual, V8.5_4*, Mentor Graphics, Corp., 1995.
- [3] D Gajski, "Essential Issues and Possible Solutions in High-Level Synthesis," in *High-Level VLSI Synthesis*, R. Camposano, W. Wolf, ed., Kluwer Academic Publishers, 1991.
- [4] A. Oppenheim and R. Schaffer, *Digital Signal Processing*, Prentice-Hall, 1975.
- [5] G. Porter and A. Whittaker, "System Design for DSP in an Integrated IC Design Environment," *Proc. VHDL International Users Forum*, May 1994.
- [6] The Institute of Electrical and Electronic Engineers, *IEEE Standard VHDL Language Reference Manual*, IEEE Std 1076-1987, IEEE, 1988.
- [7] G. Powley and J. DeGroat, "VHDL Floating Point Operations," in *Model Generation in Electronic Design*, J. Bergé, O. Levia, and J. Rouillard, ed., Kluwer Academic Publishers, 1995.