The YML2HDL Tool

Thiago Costa de Paiva

1st FPGA Developer's Forum

Jun 2024

Motivation

Technology advances

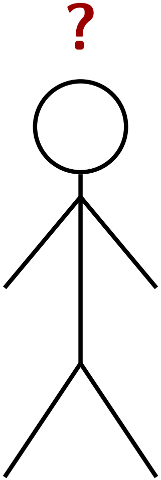
- FPGA devices become increasingly powerful
- More-complex projects are enabled
- Larger teams, developers with different backgrounds

Mixed language projects

- Custom types/constants not shared between languages
- No structured types when instantiating
  - VHDL: `bit`, `bit_vector`, `std_logic`, `std_ulogic`, `std_logic_vector`, `std_ulogic_vector`
  - SystemVerilog: `wire`, `reg`

Conversion between structured types and basic types

- SystemVerilog: `logic [$bits(structured_type)-1:0] bit_vector`
- VHDL: `???`



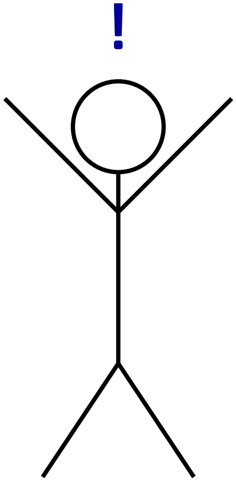
YML2HDL

Main goals

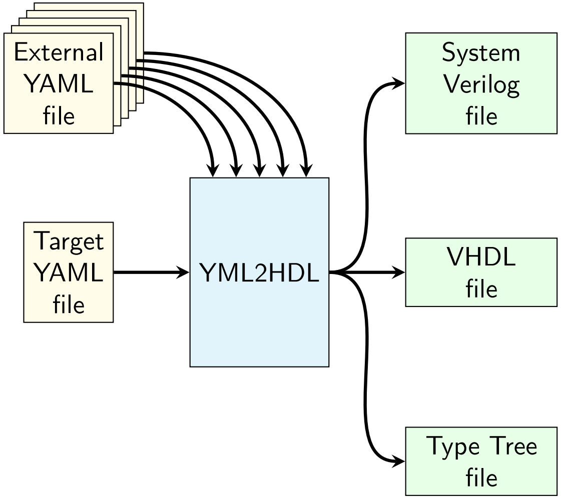
- Centralizes constants and custom types database for VHDL/SystemVerilog
- Provides VHDL helpers to convert between structured types and basic types

However...

- It is human-friendly (YAML format)
- It is git-friendly (plain text)
- It allows organization of the database in multiple files
- It is automation-friendly
- It can help with documentation
- It is customizable



YML2HDL Process Flow



```
$ python yml2hdl                    \
    -e ext_types_1.yml              \
    -e ext_types_1.yml              \
    -a ieee.std_logic_1164          \
    -a my_lib_1.my_package_1        \
    -a my_lib_2.my_package_2        \
    target.yml
```

- Only types in `target.yml` file are generated
- External `.yml` contain type definitions required for the `target.yml`
- Type tree for documentation purposes

Format Examples – Constants

YAML

```yaml
- MY_CONSTANT: {qualifier: constant, type: integer, value: 6, comment: My constant}
```

SystemVerilog

```systemverilog
// My constant
parameter int MY_CONSTANT = 6;
```

VHDL

```vhdl
-- My constant
constant MY_CONSTANT : integer := 6;
```

Alternatively (part of YAML specification):

YAML

```yaml
- MY_CONSTANT:
    qualifier: constant
    type: integer
    value: 6
    comment: |
        better for comments that
        span more than one line
```

Format Examples – Bit Types

YAML

```yaml
- bit0 : {type: logic, comment: a bit}
```

SystemVerilog

```systemverilog
// a bit
typedef int bit0;
```

VHDL

```vhdl
-- a bit
subtype bit0 is std_logic;
attribute w of bit0 : subtype is 1;
```

Format Examples – Bit Vector Types

Constrained bit vector

YAML

```
- vector0 : {type: logic, range: [3,0], comment: a vector}
```

SystemVerilog

```
// a vector
typedef logic vector0[3,0];
```

VHDL

```
-- a vector
subtype vector0 is std_logic_vector(3 downto 0);
attribute w of vector0 : subtype is 4;
```

Unconstrained bit vector

YAML

```
- vector1 : {type: logic, range: open, comment: another vector}
```

SystemVerilog

```
// another vector
typedef logic vector1;
```

VHDL

```
-- another vector
subtype vector1 is std_logic_vector;
```

Format Examples – Structured Types (1/2)

YAML

```yaml
- record0:
    members:
        - bit1: {type: logic, comment: member in a record}
        - bit2: {type: bit0}
        - vector2: {type: logic, range: [0,9]}
        - vector3: {type: vector0}
    comment: about this record
```

SystemVerilog

```systemverilog
// about this record
typedef struct packed {
   // bit1: member in a record
   logic        bit1;
   bit0         bit2;
   logic        vector2[0,9];
   vector0      vector3;
} record0;
```

Format Examples – Structured Types (2/2)

YAML

```yaml
- record0:
    members:
        - bit1: {type: logic, comment: member in a record}
        - bit2: {type: bit0}
        - vector2: {type: logic, range: [0,9]}
        - vector3: {type: vector0}
    comment: about this record
```

VHDL

```vhdl
-- about this record
type record0 is record
    -- bit1: member in a record
    bit1 : std_logic;
    bit2 : bit0;
    vector2 : std_logic_vector(0 to 9);
    vector3 : vector0;
end record record0;
attribute w of record0 : type is 1+1+10+4;
function convert(x: record0; tpl: std_logic_vector) return std_logic_vector;
function convert(x: std_logic_vector; tpl: record0) return record0;
```

Format Examples – Array Types

YAML

```yaml
- array0 : {array: [4,2], type: logic, range: [MY_CONSTANT-1,0]}
```

SystemVerilog

```systemverilog
typedef logic[4:2] array0[MY_CONSTANT-1,0];
```

VHDL

```vhdl
type array0 is array(4 downto 2) of std_logic_vector(0 to MY_CONSTANT);
attribute w of array0 : type is Abs(2*MY_CONSTANT);
function convert(x: array0; tpl: std_logic_vector) return std_logic_vector;
function convert(x: std_logic_vector; tpl: array0) return array0;
function convert(x: array0; tpl: std_logic_vector_array) return std_logic_vector_array;
function convert(x: std_logic_vector_array; tpl: array0) return array0;
```

YAML

```yaml
- array1 : {array: open, type: record0}
```

SystemVerilog

```systemverilog
typedef record0 array1[];
```

VHDL

```vhdl
type array1 is array(integer range <>) of record0;
function convert(x: array1; tpl: std_logic_vector) return std_logic_vector;
function convert(x: std_logic_vector; tpl: array1) return array1;
function convert(x: array1; tpl: std_logic_vector_array) return std_logic_vector_array;
function convert(x: std_logic_vector_array; tpl: array1) return array1;
```

Usage in VHDL

VHDL declaration

```vhdl
signal record0_s  : record0;
signal record0_vs : std_logic_vector(record0'w-1 downto 0);

signal array1_s   : array1(3 downto 0);
signal array1_vs  : std_logic_vector(array1'length*record0'w-1 downto 0);
signal array1_avs : std_logic_vector_array(3 downto 0)(record0'w-1 downto 0);
```

VHDL assignment

```vhdl
record0_vs <= convert(record0_s, record0_vs);
array_vs <= convert(array_s, array_vs);
array_avs <= convert(array_s, array_avs);
```

Documentation Friendliness (type tree)

```
- array1 : {array: open, type: record0}
```

```
A    array1: array[open] record0
R        record0:
E            bit1: logic
E            bit2: bit0
E                bit0: logic
E            vector2: logic[9,0]
E            vector3: vector0
E                vector0: logic[3,0]
```

Other Features – Command Line Options

- Conversion functions for basic VHDL types

  `-b`/`--basic`

- Overwrite type definitions temporarily

  `-c`/`--custom`

- Destination directory

  `-d`/`--output-dir`

- External YAML files with type definitions

  `-e`/`--external`

- Allow overwriting of generated files

  `-f`/`--force`

- Redefine package name

  `-p`/`--package`

- Limit generation of files
  - VHDL only (`-V`/`--vhdl`)
  - SystemVerilog only (`-S`/`--systemverilog`)
  - Type tree (`-T`/`--tree`)

- Add timestamp on generated files

  `-w`/`--when`

Hint: `-h`/`--help` works!

Final Remarks

The YML2HDL tool:

- Facilitates team work between developers with different backgrounds
- Prevents error prone approaches (centralized database)
- Is user and Git friendly
- Adds VHDL conversion functions to enable interface between languages
  - (It also can be used to store signals in memory modules)
- Is "open source"
- Is in active development
- Plans to support HLS as well
- Is used in two CERN ATLAS projects:
  - the Monitored Drift Tube trigger processor
  - the New Small Whell trigger processor
- `https://gitlab.com/tcpaiva/yml2hdl`