Open Source Formal Verification

Parametric design for formal

Yann Thoma

Reconfigurable and Embedded Digital Systems Institute
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

July 2025

Introduction

- Issues of formal verification:
  - Time
    - Related to the depth of search
  - Space
    - The bigger the data, the longer the formal verification
- Can we find ways of reducing both?

Reducing Space

- A system composed of n D flip flops has potentially 2^n states

$\Rightarrow$ Each D flip flop potentially multiplies the state space by 2

$\Rightarrow$ Reducing the number of D flip flops reduces the space

- Space as memory
  - Reduce memories sizes
  - Reduce FIFOs sizes
- Space as data representation
  - Can the design be verified with a smaller data size?
    - For instance, a FIFO can be verified with 1-bit data
  - Arithmetic computations do not necessarily need 32-bit data to check its behavior

Reducing time

- How to rapidly reach interesting moments?
- Reduce time by any means
  - Counter max values
  - Sampling frequency
  - Data rate
- Interesting corner cases: Counter overflow
  - What happens if the counter goes from max to 0?
    - Is it supposed to happen?
    - Is it handled correctly?

How to reduce?

1. Value as an input port
   - Use `assume` to fix the value for formal verification
2. Value as a generic parameter
   - Formal verification with small values
3. Value as a constant in a package
   - Use another version of the package for formal verification

Example: Counter

Bad

```
entity counter is
port (
  clk_i, rst_i : in  std_logic;
  enable_i     : in  std_logic;
  value_o      : out std_logic_vector(31 downto 0)
);
```

Good

```
entity counter is
generic (SIZE: integer := 32);
port (
  clk_i, rst_i : in  std_logic;
  enable_i     : in  std_logic;
  value_o      : out std_logic_vector(SIZE-1 downto 0)
);
```

Example: UART sender-receiver

- Baud rate as an input, generic parameter or constant in a package
- Size of the FIFO : idem
- Size of the data to be sent : idem

Package

Synthesis

```
package mydesign_pkg is

    constant COUNTER_SIZE : integer := 32;

    constant DATA_SIZE : integer := 16;

    constant FIFO_SIZE : integer := 256;

    subtype data_t is integer
        range 0 to 2**DATASIZE - 1;

    -- 9600 at 50MHz
    constant CLK_PER_BIT : integer := 5208;

end package;
```

Verification

```
package mydesign_pkg is

    constant COUNTER_SIZE : integer := 8;

    constant DATA_SIZE : integer := 4;

    constant FIFO_SIZE : integer := 4;

    subtype data_t is integer
        range 0 to 2**DATASIZE - 1;


    constant CLK_PER_BIT : integer := 8;

end package;
```

What values for generics?

- Impossible to test all values for generics
  - Especially if you have more than one generic parameter
- Then, how to choose the relevant ones?
- It depends on the meaning of the generics
  - Related to space
  - Related to time

What values for generics? - Space

- When generics represent
  - Size of registers
  - Number of elements in a memory
  - Size of the memory elements
  - Data values for mathematical computation
  - ...
- Choose
  - Short integers
    - But not too short (a FIFO of size 1 may not be sufficient)
  - Different than a power of 2 (if the design allows so)

What values for generics? - Time

- When generics represent
  - Max value of a counter
  - Timing values of a PWM
  - Number of clock cycles per bit for a serial transmitter
  - Minimum time between two events
  - Maximum time between two events
  - ...
- Choose
  - Short times
    - But not too short
  - Different than a power of 2

SBY limitation

- SBY does not accept generic parameters in SERE times

Do not work with `MAX_WAIT` as generic parameter

```
assert always {req} |=> {(not ack)[*0 to MAX_WAIT];ack};
```

- Solution: Declare a constant in a verification package

`CONST_MAX_WAIT` as a constant in a package

```
assert always {req} |=> {(not ack)[*0 to CONST_MAX_WAIT];ack};
```

  - The package constants have to agree with the generics set in the `.sby` file
  - Option: Automatically generate the package for each specific run (cf. CI and scripting)

Takeaway message

Whenever possible, use generic parameters or constants declared in packages

Not only for formal, but also for simulation-based verification