Automatic Generation of Transaction Field Methods

Stuart Swan
Platform Architect
Siemens EDA
1 December 2022

SIEMENS

Introduction

- Matchlib and SystemC require that user-defined transactions implement several utility methods within the class/struct which defines the transaction.
- These methods are:

1. `template <unsigned int Size> void Marshall(Marshaller<Size>& m);`

   - This function is needed for Matchlib connections.

2. `inline friend void sc_trace(sc_trace_file *tf, const this_type &v, const std::string &NAME );`

   - This function is needed for SystemC standard waveform tracing support.

3. `inline friend std::ostream &operator<<(ostream &os, const this_type &rhs);`

   - This function is needed for SystemC standard transaction streaming/printing support.

4. `static const unsigned int width = ... ;`

   - This constant is needed for Matchlib connections.

5. `bool operator==(const this_type & rhs) const ;`

   - This function is needed for transactions that are used with SystemC sc_signal<T>

Example of Manual Coding of Field methods (example 07*)

```
struct engine_t
{
  static const int plugs = 4;
  sc_uint<16> engine;
  spark_plug_t spark_plugs[plugs];

  static const unsigned int width = 16 + (spark_plug_t::width * plugs);
  template <unsigned int Size> void Marshall(Marshaller<Size> &m) {
    m &engine;
    for (int i=0; i<plugs; i++)
      m &spark_plugs[i];
  }
  inline friend void sc_trace(sc_trace_file *tf, const engine_t& v, const std::string& NAME ) {
      sc_trace(tf,v.engine,  NAME + ".engine");
      for (int i=0; i<plugs; i++)
        sc_trace(tf,v.spark_plugs[i],  NAME + ".spark_plug" + std::to_string(i));
  }

  inline friend std::ostream& operator<<(ostream& os, const engine_t& rhs)
  {
    os << rhs.engine << " ";
    for (int i=0; i<plugs; i++)
      os << rhs.spark_plugs[i] << " ";
    return os;
  }
};
```

User's struct/transaction

Declare HW bitwidth

Pack/Unpack to bits

SystemC standard tracing (see LRM)

Stream to text, used for transaction logging

Problems with Manual Coding of Field Methods

- All field methods need to be updated when transaction fields are updated.
- Tedious and error-prone, especially for transactions with many fields.
- Manual coding of field methods often leads to inconsistent printing and naming behavior across different transactions within a large design.
- To solve these problems, we have provided a feature which automatically generates all these needed utility functions and parameters.

Automatic generation of field methods (example 07*)

```
#include "auto_gen_fields.h"

struct engine_t {
  static const int plugs = 4;
  sc_uint<16> engine;
  spark_plug_t spark_plugs[plugs];

  AUTO_GEN_FIELD_METHODS(engine_t, ( \
      engine \
    , spark_plugs \
  ) )
  //
};
```

Required include file

Open parenthesis here is required

List all the fields, separated by commas

Two closing parenthesis here are required

- For further information see examples 07* and 23* in the Matchlib examples kit.