SSEL : Security Specification Language

Version: 0.1

Release Date: March 14, 2022

A Brief Description

In SSEL, we used a set of existing constructs from C++ and loops, functions, structures, and data types are used to define the constructs. We augmented SSEL with new data types to account for the binary and hexadecimal nature of the data transferred during on-chip transactions. Since SSEL is designed to provide abstraction of system-level communication, the technical details of various bus protocol implementations are encapsulated into APIs “baked” into the language abstraction. All inter-IP interactions are oblivious to existing bus implementations to enable reusability of SSEL specifications across platforms and interconnect protocols.

Current State of Development

Since this is the second release of SSEL, the language is still quite early in its development cycle. SSEL v0.2 supports inter-IP communication in an SoC using the below mentioned features and constructs. Not that the list below is not exhaustive. It also has support for authentication and logic-locking. SSEL also supports equivalence checking of concurrent programs (inter-IP communications) inside the SoC. SSEL models these multithreaded processes as a queue with appropriate interleaving to mimic multithreaded exeution and perfroms formal verification of these programs using KLEE. Note that these are C++ based constructs and the executables generated are g++ executables for rapid-prototyping and exercising security scenarios.
Each IP class can be derived from a base class (baseIP.cpp).

BaseIP Class

BaseIP class includes the following signals:

- A local memory map instance of an IP.

```
map<string,int> m_memory_map;
```

- System reset and System Clock

```
static bool m_sys_rst, m_sys_clk;
```

- Local IP reset

```
bool m_IP_rst;
```

- IP status signal

```
bool m_sts_busy;
```

BaseIP class has the following methods:

- **set_busy_sts()**: This method sets the busy status bit (m_sts_busy) of the IP instance.

```
void set_busy_sts( bool sts);
```

- **read_from_mem()**: This methods returns the pointer to the value of the register associated with the "key".

```
read_from_mem(string key);
```

- **modify_reg_value()**: This method modifies the register value associated with "key" with the value "value".

```
auto modify_reg_value(string key, int value);
```

- **print_map()**: This method prints the memory map associated with the local IP instance.

```
void print_map(map<string, int> const &m);
```

Each master or slave IP inherits from the baseIP class. Certain functions are overwritten in both the master and slave IP class.

MasterIP Class

This class extends the following classes:

- Memory Module (RAM instance)

```
class memory_model
```

- Global Memory Map

```
class system_memory_map
```

MasterIP has the following methods:

- **system_lvl_rst()**: This method sets the system level reset to true or false.

```
void system_lvl_rst(bool rst);
```

- **system_clk()**: This method sets the system level clock to true or false.

```
void system_clk(bool clk);
```

- **set_busy_sts()**: This method sets the IP busy status to true or false.

```
void set_busy_sts(bool sts);
```

- ***read_memory_val()**: This method reads the value of a register from the memory map pointed by calculating the address using the base and offset values passed in the arguments.

```
  int *read_memory_val(int clk, int rst, int start, int offset = 0){
          return memory_model::send_data_from_mem(clk, rst, start, offset
= 0);
        }
```

- **delete_memory_val()**: This method deletes the value of the register in the memory map pointed by calculating the address using the base and offset values passed in the arguments.

```
    void delete_memory_val(int clk, int rst, int offset, int base = 0){
        memory_model::delete_from_memory(clk, rst, offset, base = 0);
    }
```

- **write_memory_val()**: This method writes a value of a register to the address pointed by base and offset.

```
    void write_memory_val(int clk, int rst, int offset, int value, int
base = 0){
        memory_model::write_to_memory(clk, rst, offset, value, base = 0);
    }
```

SlaveIP Class

This class extends the following classes:

- Memory Module (RAM instance)

```
class memory_model
```

- Global Memory Map

```
class system_memory_map
```

SlaveIP has the following methods:

- **set_busy_sts()**: This method sets the IP busy status to true or false.

```
void set_busy_sts(bool sts);
```

- **reset_ip_instance()**: This method resets the IP instance or sets the reset value to True or False.

```
    void reset_ip_instance();
```

- **check_busy_sts()**: This method checks the IP busy status.

```
    bool check_busy_sts();
```

- **ingress_fifo_push()**: This method pushes a register value to the ingress FIFO of the IP instance.

```
    void ingress_fifo_push(int *value, int offset);
```

- **ingress_fifo_pop()**: This method pops a register value from the ingress FIFO of the IP instance.

```
    int  ingress_fifo_pop(int offset);
```

- **display_fifo()**: This method displays the contents of the ingress FIFO.

```
    void display_fifo();
```

Memory Module Class

SoC transactions need a memory model to perform read/write operations. SSEL comes with a memory model which enables this. This class extends the following classes:

- BaseIP

```
class baseIP
```

- The size of the memory model can be defined in the "memory_model.cpp" file by using the MEMORY_SIZE definition.

Memory Module has the following methods:

- **print_memory_val()**: This method prints the contents of the memory model.

```
    void print_memory_val();
```

- **delete_from_memory()**: This method deletes a value from the memory model pointed by the index using base + offset.

```
    void delete_from_memory(int clk, int rst, int offset, int base =
0);
```

- **write_to_memory()**: This method writes a value in the memory model pointed by the index using base + offset.

```
    void write_to_memory(int clk, int rst, int offset, int value, int
base);
```

- **send_data_from_mem()**: This method reads and sends a value in the memory model pointed by the index using base + offset.

```
    int *send_data_from_mem(int clk, int rst, int start, int offset =
0);
```

- **reset_memory()**: This method resets the entire memory model and sets the values to 0.

```
    void reset_memory();
```

- **fetch_sts()**: This method reads the fetch status of the memory. Fetch status is used to ensure that the memory is ready to send data when queried.

```
    bool fetch_sts();
```

Global Memory Map Class

SoC transactions need a memory model to perform read/write operations. SSEL comes with a memory model which enables this. This class extends the following classes:

```
    class Log
```

The global memory map is an unordered map.

```
    static unordered_map<string,int> *global_memory_map;
```

Global Memory Map has the following methods:

- **display_valid_bits()**: This method displays the valid bits of the memory map. If the valid bit is 1, the data is correct, else the data should not be read and is incorrect.

```
 void display_valid_bits();
```

- **check_valid_bit()**: This method checks the valid bits of the memory map.

```
    bool check_valid_bit(int offset);
```

- **print_map()**: This method displays the entire memory map.

```
    void print_map(unordered_map<string, int> const &global_memory_map);
```

- **write_to_system_map()**: This method writes the value of the register(reg) in the memory map.

```
    void write_to_system_map(string reg, int value);
```

- **read_system_map()**: This method reads the value of the register(key) in the memory map.

```
    int read_system_map(string key);
```

- **update_system_reg_value()**: This method updates the value of the register(key) in the memory map.

```
    void update_system_reg_value(string key, int value);
```

Security Constructs in SSEL

1. Contruct for authentication sequence. "uid" is the unique identifier for the IP and "vector" is the challenge response to authenticate the IP.

```
void auth_sequence(string uid, int vector) {}
```

2. Contruct for logic-locking sequence. "uid" is the unique identifier for the IP and "keys" is the key to unlock the IP.

```
void unlocking_sequence(string uid, long keys) {}
```

Configuration Parameters

1. uid.h : This contains the following:
    - Initial authentication status and requirement on an IP level.
    - Initial logic locking status and requirement on an IP level.
    - Golden Vectors of PUF-based authentication of IPs.
    - Unlocking keys of IPs.

```
  map<string, int> authentication_uid = {
        { "CCU",  1}, // 1 means IP does not require authentication
        { "UART", 0}, // 0 means IP does requires authentication
        { "GPIO", 1},
        { "PCM",  1}
};
map<string, int> authentication_vector = {
        { "CCU",  0},
        { "UART", 12341234},
        { "GPIO", 0},
        { "PCM",  0}
};
map<string, int> obfucated_uid = {
        { "CCU",  1}, // 1 means IP does not require unlocking
        { "UART", 0}, // 0 means IP does require unlocking
        { "GPIO", 1},
```

```
        { "PCM",  1}
};
map<string, long> obfuscated_keys = {
        { "CCU",  0},
        { "UART", 987987},
        { "GPIO", 0},
        { "PCM",  0}
};
```

Security Wrapper Configuration Parameters

SSEL can also be used for auto-generating Security Wrappers based on the provided configurable paramters here. A brief description of the configurable parameters is provided below:

- rstType : This can either be 0 or 1. 1 is the default value.
- rstSync : This can either be 0 or 1. 0 is the default value.
- clkEdg : This can either be 0 or 1. 1 is the default value.
- maxFanout : Any positive integer.
- nModCell : Any positive integer.
- HIRApercent : Any integer.
- nFFminPerFSM : Any value >= 3.
- nFFmaxPerFSM : Any integer >= nFFminPerFSM.
- nFFminPerSubFSM : Any integer >= 2.
- nFFmaxPerSubFSM : nFFminPerSubFSM <= Any integer <= nFFmaxPerFSM.
- reachability : Any integer >= 2^nFFminPerFSM.
- keyLength : Any positive integer.
- initSeed : Any integer.
- hasInPortFile : This can either be 0 or 1. 0 is the default value.
- fanoutHistIsTrue : This can either be 0 or 1. 0 is the default value.
- activityHistIsTrue : This can either be 0 or 1. 0 is the default value.
- simFlags : Any additional parameter passsed during simulation.
- pufElements : Number of PUF elements in the MeLPUF.

A detailed description of each of these parameters can be found in the ProtectIP user manual.
If SSEL does not detect any violations in the wrapper configuration for ProtectIP and ASTRA, it will pass these to the interface with ProtectIP and ASTRA to generate configurable security wrappers. This will be done in the coming quarters of this phase.

Requirements

This version of SSEL requires C++17 or above to build.

Future Work

1. The developers plan to extend SSEL to enable automatic generation of Security Wrappers for different threat levels.
2. Generation of RISC-V firmware will also be automated using SSEL specifications.