Modeling Software Interrupts using SystemC

by
David C Black

Formerly
Eklectic Ally

Electronic System Level Xperts

www.ESLX.com
Info@ESLX.com
Version 1.2

Agenda

- Problem statement
- Approaches
- An API
- Example
- Features
- Implementation Details
- Potential Issues
- Future Directions

Problem

- Platform based simulations usually have multiple interrupt sources simulated with events, but...
- When waiting for multiple events (including timeouts), it is not possible to know which event caused the wait to complete.

Approaches

- Instruction Set Simulator (ISS)
  - More accurate
  - Slower due to extra detail
  - Builtin Interrupt Handler (usually)
- Higher Level Modeling
  - High level native code
  - More general
  - Less granularity
  - Need detectable events

Considered solutions

- Use SystemC 2.1's dynamic events to create a special class that can track events.
  - Dr. David Long of Doulos presented one such a solution in the SystemC forum
  - Requires version 2.1
- Create a custom channel that uses request/update and STL map<>
  - This is what we chose to do
  - Only requires 2.0
  - Other features such as multiple scheduled notifications on the same event (like sc_event_queue)

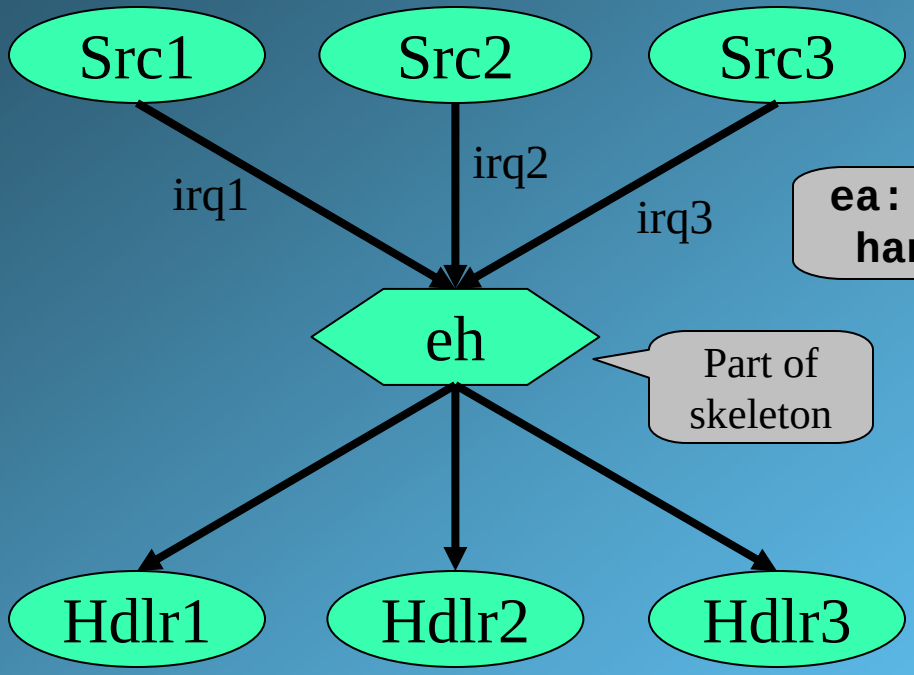
Intent of solution

- Allow events that are detectable

```
wait(evt1|evt2|evt3);
if ("evt1 happened") ...
```

- Allow multiple outstanding events
  - Test can post for multiple future times
- Useful for simulating hardware interrupt controllers as well

Example scenario




```
ea::event_handler eh;
sc_event irq1, irq2, irq3;

eh.notify(delay,irq3)
```

ea::event handler

SC_ZERO_TIME is allowable

sc_event goes here

```
wait(Timeout,irq1|irq2|irq
3);
if (timed_out()) {
  // Handle timeout
}
if (eh.event(irq1)) {
  // Handle interrupt 1

}…
```

Features

- Only interrupts that use the channel have any burden. Good for performance of most of the design.
- Except for reporting, uses 2.0.1 features + STL only.
- Supports **SC_ZERO_TIME** notifications
- Similar to **sc_event_queue** except that events scheduled for a single time are bundled. See following slide.
- Ability to notify, check active
- Able to check pending and cancel in multiple ways:
  - Time & event
  - Time only
  - Event only
  - All
- Built-in debug features -- outputs details as events are scheduled (notified) and become active with identification of sources

ea::event_handler API

- `event_handler()`
  - May specify name
  - May specify verbose*
- **`void`** `notify(delta,evt)`
- **`bool`** `event(evt)`
- **`int`** `events()`

* Requires #define `EVENT_HANDLER_DEBUG`

- **`int`** `pending(time,evt)`
- **`int`** `pending(evt)`
- **`int`** `pending(time)`
- **`int`** `pending_times()`
- **`int`** `pending_all()`

- **`void`** `cancel(time,evt)`
- **`void`** `cancel(evt)`
- **`void`** `cancel(time)`
- **`void`** `cancel_all()`

How notification differs

Ignore syntax

Example:

```
notify(my_time1, my_evt);
notify(my_time1, my_evt);
notify(my_time1, my_evt);
notify(my_time2, my_evt);
```

- 1 event under **`sc_event`** (nearest time kept)
- 4 events under **`sc_event_queue`** (2.1 only)
- 2 events under **`ea::event_handler`**

Syntax notes

- Use of handler
  - Declaration & instantiation hidden in skeleton
  - Only one handler in system. Even for multiple handler situations!

```
#include "ea_event_handler.h"
ea::event_handler eh;
```

- Modified **notify** syntax:

```
eh.notify(sc_time,sc_event)
```

- Test for occurrence event:

```
if (eh.event(sc_event)) {
```

Requirements

- Weak dependence on SystemC 2.1
  - **SC_REPORT_** macros - not functionality
    - Could use SCV
    - Simple macros
- Implementation uses **STL**
  - `#include <map>`
  - `#include <set>`
  - `#include <sstream>`

Implementation file structure

- Interface
  - ea_event_handler_if.h
- Channel specificaiton
  - ea_event_handler.h
- Implementation
  - ea_event_handler.cpp
- Testbench
  - ea_event_handler_tb.h

Implementation class structure

- Two channels in one

  - Hierarchical channel contains 1 **SC_METHOD**
    - Issues notifications to track future events

  - Primitive channel: **request()** & **update()**
    - Remove old events
    - Manages delta cycle requests
  - Share STL **map<set<>>**
  - One event

Possible Issues

- Performance
  - Probably not an issue for application
  - Address if/when problem
- Coding errors
  - Forgetting to insert **`event_handler`** prefix
    - **`ea.notify(`**
    - **`ea.event(`**
  - Debug features available
    - SystemC 2.1 required for debug
      - Direction of SystemC community

Future directions

- Inherit from `sc_event` or make similar
  - More natural coding reduces errors
    - `ea_event irq1;`
    - `irq1.notify(t1);`
    - `if (irq1.event())`
  - Why not now?
    - Risk/time
  - SystemC 3.0 enhancement
- Add `watch(evt)` method
  - Support events triggered externally

Aknowledgements & Availability

- **Users**
  - One of our advanced customers who wishes to remain anonymous
- **Reviewers**
  - Rene Lemoyne
  - Grant Martin
  - Jack Donovan
- **Source code (ESLX.com)**
  - <https://www.ESLX.com/Library_open_area/index.html>
  - Registration required
  - Feedback to <dcblack@ESLX.com>

Questions?

How events are managed

- ```
  #include <map>
  using std::map;
  #include <set>
  using std::set;
  ```
- ```
  map<sc_time,
              ,set<sc_event*> > m_schedule_map;
  ```
  - `sc_time_stamp()` entry contains currently active events (if any)
  - Other entries pertain to scheduled future events
- `std::set<sc_event*> m_delta_set`
  - Contains scheduled delta events
  - Transferred during **`update()`**

We use the C++ STL (Standard Template Library) For implementation

An associative array (map) contains time indexed sets of scheduled events.

Who is ESLX?

- ESLX is a consulting company that focuses on Electronic System Level (ESL) Design and specializes in SystemC.
- We offer a variety of SystemC services:
  - Experienced adoption planning
  - World-class training
  - Advanced methodology definition
  - Efficient flow design & automation
  - Certified team staffing & mentoring
  - Modern verification & testbench development
  - Behaviorally synthesizable model development
  - Fast & accurate software development platforms
- Contact us via Info@ESLX.com