Automatic Synthesis of Interfaces between Incompatible Protocols

Roberto Passerone
Cadence Design Systems, Inc.

Problem formulation

Given the description of two protocols operating with different signaling conventions

Synthesize an interface so that data transfers are consistent with both protocols

Assumptions

- The protocols exchange the *same* token
- Data in the interface is stored in a register wide enough to contain an entire token, controlled by an FSM
- The protocols are independent of the data contents
Overview of the synthesis process

A protocol description

A protocol is the set of all *sequences* of values *admissible* at the ports of a module

Protocol description: Example

```plaintext
protocol serial of type yow {
    master bit start;
    master byte bus;

term null( ) { 0, - }
term one( byte b ) { 1, b }
term two( byte b ) { 0, b }
serial( yow y ) {
    null( ) *, one( y.a ), two( y.b )
}
}
```

Automata generation: Example

```plaintext
serial( yow y )
null( ) *, one( y.a ), two( y.b )
handshake( yow y )
wait( 0 ) *, get( 1, y.a )+, get( 0, y.b )+
```
PIG - Verilog Interface Synthesis from Protocols

```
module s2h(clock, s_start, s_byte, p_trigger, p_bus);
input clock, s_start, s_byte, p_trigger, p_bus;
reg [15:0] temp;
always @(posedge clock) begin
    if (state==0 && s_start==1) begin
        state = 1; temp[7:0] = s_byte;
        p_trigger = 0;
    end else if (state==0 && s_start==0) begin
        ...
    end
endmodule
```

```
void s2h(port s_start, port s_byte, ...)
static int state = 0; static int temp = 0;
if (state==0 && s_start.transition && s_start==1) begin
    state = 1;
    temp[7:0] = s_byte;
    p_trigger = 0;
end else if (state==0 && state==0) begin
    ...
end
```

Testbench

- Also synthesize “driver” and “monitor” modules
- Driver module is FSM that:
  - randomly generates token values
  - randomly selects from non-deterministic choices in protocol (Kleene closures, for instance)
- Monitor module is FSM that:
  - collects token and prints it out when received successfully
  - randomly selects non-deterministic responses
Adding streams

serial(yow y) handshake(yow y)

Adding streams: product

- Specify the size of the queues
  - Integer number of tokens
- During the product exploration, keep track of the iteration at which each protocol has arrived
- In the product, states built from the same pair of states are considered different whenever the difference in the iteration count is different
- The difference in iteration count shall not exceed the specified size of the queue
Adding streams: to study

- Resolution of non-determinism
  - Current heuristics may not find the best solution
  - May want to satisfy certain properties
  - Subset construction may actually be needed to explore even more possibilities

- State minimization
  - Synchronization information is no longer relevant after the product has been computed
  - Lots of opportunities for optimization
Adding streams: optimization

Multi-point communication

Processes specify the advancement of stream iterations

State transition diagrams

Asynchronous behavior

- A clock is a stream (so it is global to a composition)
- Transitions refer to clocks or to changes in values
- During the product computation, transitions on the same clock are considered synchronous
- All the others are considered asynchronous
  - In the product, all possible interleaving of asynchronous transitions are considered
- Implemented but not yet tested
Desiderata

- Examples!!
  - In the near future: PCI, Amba, VCI, PI, OCP
- State minimization
- Lossy communication
  - In the new specification, protocols can separately access and synchronize on data fields
- Distributed control
  - Partition the interface within the parties
- Data dependency
  - E.g. number of packets depends on header information
- Data-flow analysis
  - Size queues, direct connection

Conclusions

- Regular expressions used as a protocol description language.
- Product machine exploration as the basis for interface generation.
- System integration by assembling IP’s and generating correct glue logic.