Concurrency in Hardware Description Languages

Chae Jubb
27 October 2014
Concurrency in HDLs

- More natural than in conventional programming languages
  - Because hardware is inherently parallel!
- Data Flow
- "Composition of Components" model
“Composition of Components”
Control Flow Paradigm

- if / goto
- loops
- subroutine
- e.g. C, Python, Haskell
Data Flow Paradigm

- If S depends on A, updating A automatically updates S
- Layperson’s example: spreadsheets
- Hardware Description Languages (HDLs)
Hardware Description Languages

- Simulate hardware
- Data Flow paradigm
- Explicit notion of time
- Synthesizable subset
- Inherently parallel simulations w.r.t. clock
- e.g. Verilog, VHDL, SystemC
Threaded Concurrency

- **Shared memory**
  - semaphores
  - waiting
- **Multiple threads**
  - Start and stop programmatically
- **Thread scheduling not consistent**
- e.g. pthreads in C
pthreads (C)

```c
static pthread_mutex_t mut;

void *body(void *args) {
  int *arg_in = (int *) args;
  pthread_mutex_lock(&mut);
  *arg_in = *arg_in + 5;
  pthread_mutex_unlock(&mut);
  return NULL;
}

int main() {
  pthread_t thread0;
  pthread_t thread1;
  int *argument = malloc(sizeof(int));
  *argument = 0;
  pthread_create(&thread0, NULL, body, (void*) argument);
  pthread_create(&thread1, NULL, body, (void*) argument);
  pthread_join(thread0, NULL);
  pthread_join(thread1, NULL);
  printf("Output: %d\n", *argument);
  free(argument);
  return 0;
}
```
Inherent Concurrency (Verilog)

- Hardware has multiple components
- These work concurrently
- Consider Content Addressable Memory (CAM)
  - Used in networking
    - Routers
    - DNS
  - Read, Write, Search simultaneously
Inherent Concurrency (Verilog)

module cam

/* write functionality */
decoder write_dec(.inp_i(d.write_index_i), .enable(d.write_i), .out_o(write_reg_enable));
always_comb begin for (int iter = 0; iter < 2**ARRAY_SIZE_LOG2; ++iter) begin cam_i[iter] = d.write_data_i; end end

/* read functionality */
mux #(SELECT_WIDTH(ARRAY_SIZE_LOG2), .DATA_WIDTH(ARRAY_WIDTH_LOG2)) read_data_mux(.inp_i(cam_o), .selector_i(d.read_index_i), .out_o(out_value));
mux #(SELECT_WIDTH(ARRAY_SIZE_LOG2), .DATA_WIDTH(0)) read_valid_mux(.inp_i(cam_v_o), .selector_i(d.read_index_i), .out_o(written));

/* search functionality */
equality_checker #(DATA_WIDTH(ARRAY_WIDTH_LOG2), .NUM_COMP(2**ARRAY_SIZE_LOG2)) eq_check_search (.inp_i(cam_o), .valid_i(cam_v_o), .data_i(d.search_data_i), .out_o(cam_found));
priorityencoder #(.SIZE(ARRAY_WIDTH_LOG2)) search_priorityenc (.inp_i(cam_found), .out_o(out_index), .valid_o(found));
...
How SystemC Handles Concurrency
SystemC (Naturally Concurrent)

- C++ Library that emulates HDL
  - C++ style syntax
- Analogues with hardware
  - Classes::Components
  - Class Variables::Inputs/Outputs of Components
  - Thread(s)::(Multiple) functions of component
SystemC Description of Adder

Interface

```c
SC_MODULE(adder) {
    sc_in<bool> clk;
    sc_in<bool> rst;
    sc_in<long int> in_0;
    sc_in<long int> in_1;
    sc_out<long long int> out_data;

    void beh(); /* behavior */

    SC_CTOR(adder) {
        SC_METHOD(beh);
        sensitive << clk.pos() << rst;
    }
};
```

Functionality

```c
void adder::beh()
{
    if (!rst.read()) {
        RESET:
        out_data.write(0);
    } else if (clk.event()) {
        long int tmpInput0 = in_0.read();
        long int tmpInput1 = in_1.read();
        out_data.write(tmpInput0 + tmpInput1);
    }
}
```
SystemC and Concurrency

- Methods (seen previously)
- Threads
- CThreads
SystemC Methods

- Run when triggered
  - Triggers set by sensitivity list
  - Similar to function call

- No lasting local storage across invocations

- Each triggering is independent of previous
SystemC (C)Threads

- Correspond to single functionality of component
- Different than POSIX Thread
  - Can have multiple SC_THREADs, but still have single threaded program
- Not explicitly called by user
SystemC (C)Threads

● Conventions
  ○ Infinite Loop
  ○ Wait
    ■ Positive clock edge
    ■ Reset signal

● CTHREADs (Clocked threads)
  ○ Can only wait on clock signal
  ○ Jumps to beginning of thread on reset
SystemC Simulation

1. Initialization
2. Eval and repeat
3. “Update Phase”
4. Goto 2 if necessary
5. Increment time and Goto 2 or terminate
SystemC Features

- Clock and Time
- Signals and Concurrency
- Events and Notifications
Concurrency in HDLs

- Different paradigm than control flow languages
- Explicit notion of time and a clock
  - HDLs concurrent with respect to this clock
- Easier to model because hardware is inherently parallel
References

● https://www.ida.liu.se/~TDDI08/labs/lab1.pdf
● SystemC 2.0.1 LRM
● Verilog Code: Chae Jubb and Tim Paine