Asynchronous Verilog
interface between behavioural and pipeline synthesis

Wei Song

24/10/2011
Asynchronous Verilog

• A super set of Verilog (may or may not include some SystemVerilog features)
• A way to describe asynchronous pipelines (similar as RTL for sync circuits)
• Describe both sync and async in the same language
• Synthesisable using asynchronous synthesiser (into gate-level)
Motivations

• Proposal, layered design flow
  – Behavioural synthesis (high-level synthesiser)
    • Automatic async/sync partition
    • Petri-net, token flow synthesis
    • Pipeline style and interface definition
    • Fast architecture builder and behavioural simulation (SystemC based?)
  – Pipeline level synthesis (RTL) (DC-like synthesiser)
    • Power/speed/area optimisation
    • STG -> control circuits
    • Gate-level circuits generation
    • Detailed pipeline optimisation
    • Circuit-level simulation (compatible with Verilog ?)
Motivations

• Friendly to sync RTL designers
  – The handshake languages (Balsa) are not easy to understand by sync designers.
  – No clear clue where is pipeline stages.
  – No obvious DFT solution?
  – Lack of low-level speed analysis
  – Syntax-based direct map instead of logical synthesis
Motivation

• Asynchronous Verilog
  – A language to describe pipeline stages using always blocks
  – Using STG to describe pipeline control
  – Clear pipeline stage definition
  – Possible to analyse critical paths
  – Speed/area/power analysis
  – Real cell library
  – Constraint based synthesis (like DC)
Expected features

• Pipeline description

always @(posack acko) begin // define ack
    if (b > 5)
        a <= b - c; // general description of comb.
    else
        a <= b + c;
    acki <= a; // infer completion detection
end
Expected features

• Pipeline style (attribute)

always @(posack acko) begin
 (* pipe_style = 4p1o4 *)  // pipeline style
  a <= b - c;
  acki <= a;
end
Expected features

• Control circuits (using STGs as functions)

  STG pipe_ctl  // define the STG as a function block
  input ai, bi, ci;
  output eo, fo;
  begin
    ai+ eo+
    eo+ bi-
    ...
  end

  always @(ai or bi or ci)  // utilise the STG
  pipe_ctl (ai, bi, ci, eo, fo);
Features of Asynchronous Verilog

• Recognise all Verilog RTL level features
  – May support gate-level description
  – May support truth table or specify blocks
  – May support some synthesisable SystemVerilog

• Describe asynchronous circuits
  – always @(pos/negack ack) for pipeline stages
  – STG for control circuits
Design flow

• Behavioural level description
  – SystemC or ??
  – Galaxy like GUI for async/sync partition?
  – New features for token, petri-net?
  – SystemC simulation

• Behavioural synthesise
  – Asynchronous Verilog: both RTL and async pipes
  – Constraints: sync constraints, async pipe style, speed, skew margin, fork margin (automatic early evaluation infer?)
Design flow

- Asynchronous synthesis
  - Read in Async Verilog, constraints, cell lib
  - Generate RTL verilog including all sync circuits (asynchronous circuits defined as black boxes)
  - Generate constraints for sync circuits
  - Generate async gate-level netlist and constraints for P&R
Design flow

• RTL synthesis
  – DC for sync RTL netlists

• P&R
  – Sync netlist + constraints (DC)
  – Async netlist + constraints (Async Syn)
  – Generate GDSII
Remaining problems

• Boundary discovery
  – Identify signals as boundary (attributes)
  – Don’t touch blocks as sync blocks
  – Black box as sync blocks
  – Automatic infer
    • Signals from always @(posedge clk) are sync
    • All sync FFs to async pipes are async
    • All async pipes to FFs are sync
Remaining problems

• Manual designs (DLL, delay line, special)
  – dont_touch in constraint file
  – Balck box
  – DesignWare library (not at the first stage)

• Multi-pipeline stage control (difficult for timing analysis)
  – Use single pipeline stage control instead
    • Behavioural synthesis must divide complex STG into small ones
    • Use state machine instead of STG
Remaining problems

• Simulation
  – Before synthesis
    • Translate Async Verilog to System Verilog
      – Use equivalent tasks and functions to describe pipes and STGs
      – System Verilog is difficult to parse and synthesis yet
  – Post synthesis
    • Normal Verilog simulation
Asynchronous Synthesiser

• Read in cell library
  – Speed/area/power analysis and optimisation
  – First stage: read in logic and area info

• Read in async verilog
  – Specify async features
  – Parse RTL and async Verilog
  – Transfer async Verilog to internal data (block level graphs in bus, operation and state machines)
Asynchronous Synthesiser

- Parse constraints
- Parse user scripts (similar to DC)
- Automatic boundary discovery
- Logic synthesis
  - First stage, direct translation
- Optimization (none at first)
- Constraint generation
- GUI (like design vision, but ps files first, Balsa)
Asynchronous Synthesiser

- Possible features:
  - Automatic critical path calculation
  - Automatic equivalent logic replacement
  - Automatic pipeline style swapping
  - Automatic early ack evaluation
  - Power report by parse VCD/saif files
  - Automatic retiming (move pipeline stage forwards/backwards), pipeline duplication, bubble insertion
Summary

• Two-level design flow
  – Behavioural level:
    • S/A partition and async pipeline generation
  – Pipeline level
    • Logical synthesis

• A new Async Verilog language
  – Describe both sync and async
  – Unified design environment
Questions?