CTL Modeling for Memory Test and Repair:
Part 1: The “what to” model aspects

S. Boutobza (Synopsys Inc)
June 19, 2006
Memory CTL
Plan

- Disclaimer
- CTL Modeling: Possibilities and Limitations
- Memory Test and Repair: Purpose and Scope
- Memory Test and Repair: what to model
Disclaimer

- The present slides are not intended to be a “full” specification

- Throw out some high-level recommendations regarding the “what to” model

- They are not tackling the “how to” model issue

- The goal: (with the help of this material) achieve a common agreement on the scope and the precise objective of this (standardization) activity
CTL Initial intent:

- Definition 1: “CTL is a language that is used to represent test information”
  
  [CTL for Test information of Digital ICs, R. Kapur03]

- Conclusion 1: CTL is the right candidate to model the memory test and repair and it presents a certain freedom for that
Definition 2: “CTL is the outcome of the need to solve the communication between multiple parties to create a successful test set for the SoC which involves testing the reused embedded design called a core and testing the logic of the SoC around the core in the presence of the core”

CTL limitations for memory test/repair modeling:
- Lack of several paradigms to model and represent the architectural/timing and topological information of an embedded memory
- Lack language constructs to describe memory test/algorithms procedures
- Descriptive Vs. Prescriptive

Conclusion 2: certain limitations in the CTL that imply some extensions
Purpose and Scope

- Purpose of the standardization activity:

  - Provide **accurate, simple, non-redundant** and **comprehensive** language(s) to describe and implement a memory test and/or repair algorithm/solution
Purpose and Scope

The scope of the standardization activity:

- Have a flexible and universal approach applied to the whole majority of memory test & repair approaches.

- The resulting language/extension should not be tied to any given memory test & repair:
  - procedure (stop at N errors, last cycle restart, first cycle restart, repair at error, repair at completion…),
  - architecture (BIST/BIST sub-modules),
  - or flow (steps for wrapping, hierarchical core integration..)
Memory Test and Repair: what to model

- What to model:
  1- Model embedded memories
  2- Model test and repair algorithms
  3- Model hardware realization of test and repair algorithms (BIST/ BISR)
  4- Model how to test/control the BIST/BISR IPs
Memory Test and Repair: what to model

1- Model embedded memories

- The memory, type, timing, architecture, access protocols and topology information

- The memory is a core and we need to model (describe) different views of that core

- Belong to CTL design (core) information

- Best candidate: CTL extension
Memory Test and Repair: what to model

STIL 1.0 {
  CTL P2001.10;
  Design P2001.1;}
Header {
  Date "04/23/06";
  Source "Hand Edited";
  History {Ann {* v1.0 tentative model by SB *}}}

Variables {IntegerConstant addrMAX = 16383; IntegerConstant addrMSB = 13; IntegerConstant dataMSB = 7;}
Signals {
  "QA"[dataMSB..0] Out; "CKA" In;; "MEAN" In; "TBYPASSA" In; "WEAN" In; "OEAN" In; "AA"[addrMSB..0] In;
  "DA"[dataMSB..0] In;}

Environment "MYSPRAM_16384x8m32" {
  CTL {
    MemoryProperties {ColumnMultiplexing 32; TopologicalOrg Distributed; SimultaneousReadWrite None;…} }

  CTL Mission_mode {
    TestMode Normal;
    Family SNPS_memory;}

  CTL BIST_mode {
    TestMode InternalTest;
    Internal {
      "QA"[dataMSB..0] {DataType MemoryData ; }
      "CKA" {DataType MasterClock {ActiveState ForceUp ; }}
      "MEAN" {DataType CoreSelect {ActiveState ForceDown ; }}
      "TBYPASSA" {DataType TestMode {ActiveState ForceDown ; }}
      "WEAN" {DataType MemoryWrite {ActiveState ForceDown ; }}
      "OEAN" {DataType TestMode {ActiveState ForceDown ; }}
      "AA"[addrMSB..0] {DataType MemoryAddress {ValueRange 0 addrMAX; }}
      "DA"[dataMSB..0] {DataType MemoryData Synchronous; }}

  CTL Bypass_mode {
    TestMode Bypass;
    Internal { "TBYPASSA" {DataType TestMode {ActiveState ForceUp; }}
  }}
}
Memory Test and Repair: what to model

2- Model test and repair algorithms

- No common language today to describe in consistent manner, memory test/repair algorithms (textual, programming languages, mathematical, OMTL..)
- Lot of misunderstanding in the EDA/IDM community and considerably limit tools interoperability (different inputs, parsers, reports formats, documentations…)
- A good candidate for standardization: describe a mathematical procedure (invariant, platform and tool independent)
- Ensure two objectives:
  - Hardware/software implementation (prescriptive usage)
  - Verification of the implementation (descriptive usage)
- Feasibility: a common and simple language to describe all memory test algorithms is not a fiction (based on previous experiences)
Memory Test and Repair: what to model

2- Model test and repair algorithms

- Two possible formats: textual (more readable), mathematical (more flexible)

- Can be a part of the CTL Pattern/protocol Information. But CTL reuse the STIL Protocol constructs

- Which language for test and repair algorithms?
  - Approach 1: CTL extension (a sub-language)
  - Approach 2: ATE test languages extension (STIL)
  - Approach 3: A new specific language
Memory Test and Repair: what to model

- Environment MemoryTestRepair {
  - MemoryTests {
    - Algorithm algo_1 {
      - MarchSweep {
        - Address {
          - CountDirection Down;
          - CountStyle FastRow;}
        - Data {
          - Pattern Solid;}
        - Port {
          - OffsetRow 0;
          - OffsetColumn 0;
          - Operation Write Background;
          - Operation Read Background;}
      - MarchSweep { …} 
    - Background algo_1 {
        - Type AlgorithmBased;
        - Pattern "0x00";
    - Test algo_1 {
        - Algorithm algo_1;
        - Background algo_1;
    - Algorithm algo_dp_1 {…}
    - }
- }
- }
- }
- }
}
Memory Test and Repair: what to model

3- Model hardware realization of test and repair algorithms (BIST/ BISR)

- Model the core interface (I/Os) as any other core
- The BIST is a core and the test structure in the same time: need special considerations
- Facilitate test synthesis/generation of sub-modules
- Facilitate test integration process (top-level control of several distributed BIST instances) which is one major goal of CTL
Memory Test and Repair: what to model

3- Model hardware realization of test and repair algorithms (BIST/ BISR)

- Internal, External, CoreInternal CTL blocks can be reused for this purpose

- Or create a new block statement “BISTInternal”, “BISTStructures”…

- Common modeling of some properties with LBIST
Memory Test and Repair: what to model

Environment "top" {
  CTL mbist_go_nogo{
    TestMode InternalTest;
    Family myFamily_MBIST_blabla;
    Focus CoreInstance "mbist_cntrl_1" {
      TestMode ExternalTest;
      CTL "mbist_cntrl_1" go_nogo;
      PatternBurst mbist_go_nogo;
    }
    Focus CoreInstance "mem_1" {...}
    DomainReferences {
      SignalGroups mbist_go_nogo;
      Variables mbist_go_nogo;
      ScanStructures mbist_go_nogo; ...
    }
  }
  Internal {
    "top_clk1" {DataType User bist_clk; }
    "my_bist_clk_lu" {DataType ScanMasterClock User bist_clk_lu {ActiveState ForceUp; }}
    "bist_run" {DataType User bist_run {ActiveState ForceUp; }}
    "bist_fail" {DataType User bist_fail; }
    "bist_done" {DataType User bist_done; }
    "bist_reset" {DataType Asynchronous User bist_reset {ActiveState ForceUp; }}
    // constrain other signals here...
    "test_si" {...}
    "test_mode1" {...}
  }
  CTL mbist_diagnosis{...}
  CTL Internal_scan {...}
}

Named Env block

Different CTL blocks to describe test modes

I/O interface of the top level module
Memory Test and Repair: what to model

4- Model how to test/control the BIST/BISR IPs (ATPG/ATE)

- Model how to control and test that core (the test protocol)

- Provides information for DRC, Pattern formatting (timing and vector relationships) and defines certain options (set clock, set pi constraint...)

- Set-up, resume Procedures, test, repair MacroDefs, WaveformTables...

- Almost implicit (supported by current CTL/STIL statements) → probably no need for CTL extension

- To ensure tight control and be more compliant, use the CTL PatternInformation block
Memory Test and Repair: what to model

MacroDefs mbist_go_nogo{
  "setup_mbist" {
    W "_default_WFT_";
    C {
      "top_addr[0]" = N; "mbist_prog_valid" = N;
      "mbist_diag_valid" = X; "bist_fail" = X;
      // other conditions
    }
    V {"my_bist_clk_lu" = 0; "bist_run" = 0;"bist_reset" = 1; "test_si" = 0;}
    V {"my_bist_clk_lu" = P; "bist_reset" = 0;}
  }
}
"start_mbist" {
  V {"my_bist_clk_lu" = P; "bist_run" = 1; "bist_done" = L; "bist_fail" = L; }
}
"check_mbist" {
  C {"bist_run" = 0; }
  V {"bist_done" = H; "bist_fail" = L; }
}
"run_mbist" {
  test_setup2 : Macro "setup_mbist";
  Macro "start_mbist";
  Macro "perform_sequence" {
    num_clock_cycles = 'hardwired_num_clock_cycles';
  }
  Macro "check_mbist";
}
Memory Test and Repair: what to model

- Issues to consider:
  - Possible conflicts with the P1500 standardization activity
  - Define clear boundaries with the STIL (1456.1)
  - Extension of CTL initial intent: from design and patterns description to more abstract paradigms (e.g., test algorithms)
Memory Test and Repair: what to model

Platforms and tools:

- DFT (BIST) Generation/integration
- Memory Compilers
- ATPG
- ATE
CTL Extension (what to model)
Flow Example

1-Memory Compiler → Memory Description view

Memory Test/Repair view

2-DFT Generator/Integrator

Memory BIST/BISR (Core) view

Memory BIST/BISR (Test) view –STIL–

3-ATPG / FS/ATE
Back to work (discussion)…