A framework for verification of Program Control Unit of VLIW processors

By
Santhosh Billava, Saankhya Labs
Sharangdhar M Honwadkar, Saankhya Labs

accellera SYSTEMS INITIATIVE

Saankhya Labs

2014 DESIGN AND VERIFICATION DVCON CONFERENCE AND EXHIBITION INDIA

Agenda

- Introduction
- VLIW processor overview
- Program control unit of a VLIW processor
- Verification challenges
- Framework for test case generation
- Results and Conclusions
- Questions

Agenda

- Introduction
- VLIW processor overview
- Program control unit of a VLIW processor
- Verification challenges
- Framework for test case generation
- Results and Conclusions
- Questions

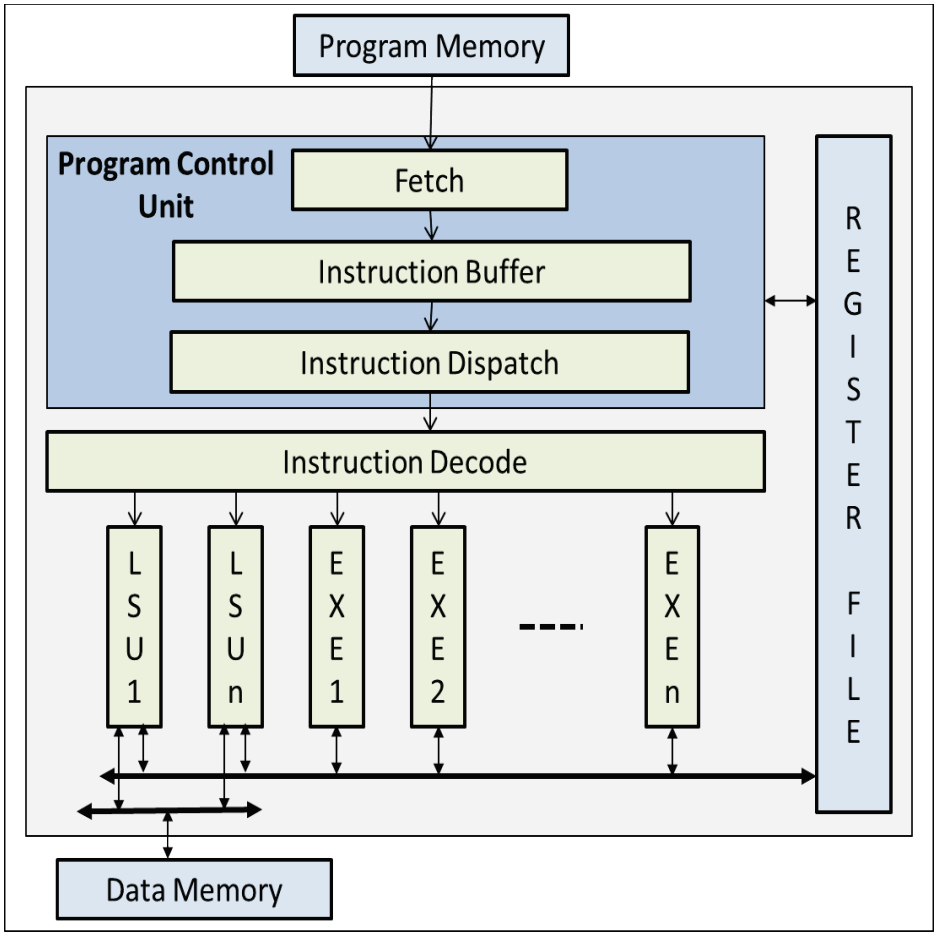
Introduction

- Work done at Saankhya Labs – India
  - An innovative fabless semiconductor company providing SDR(Software Defined Radio) based ICs and modules for video, audio and data communication
- VLIW processors are part of Saankhya ’s SDR Platform
- Product success is defined by how well the individual VLIW processors are verified at all levels
- This work is done to achieve the above mentioned objective

Agenda

- Introduction
- VLIW processor overview
- Program control unit of a VLIW processor
- Verification challenges
- Framework for test case generation
- Results and Conclusions
- Questions

VLIW Processor Overview



- Pipelined processors
- Simple instruction set
- Parallel execution
- Compile time scheduling
- Multiple execution units
- VLIW instruction length depends on number of execution units

Agenda

- Introduction
- VLIW processor overview
- Program control unit of a VLIW processor
- Verification challenges
- Framework for test case generation
- Results and Conclusions
- Questions

Program Control Unit

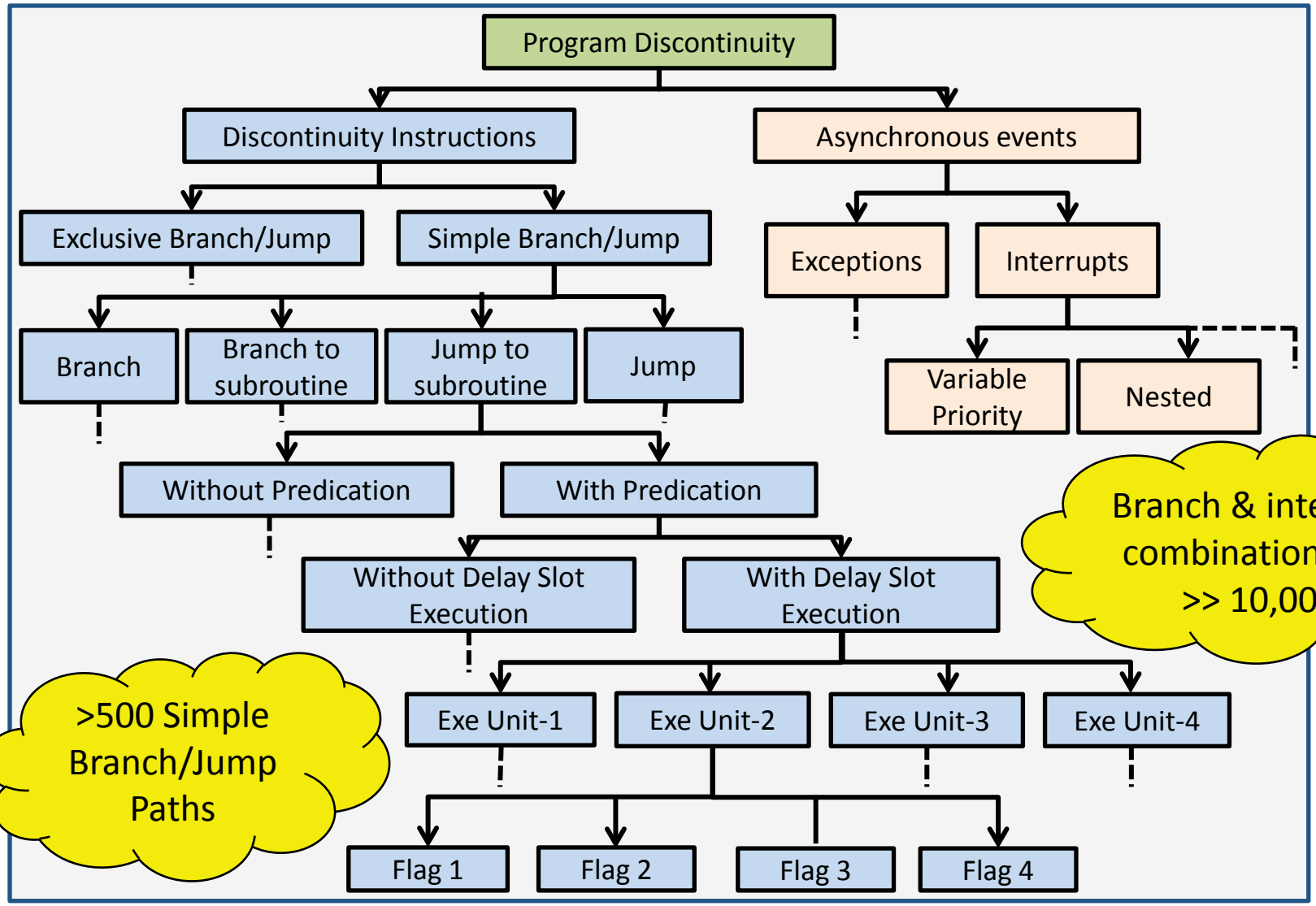
- Controls program execution flow
- Instruction buffer to decouple fetch and execution units
- Delay in instruction fetch introduces execution stalls
- Branch, jump, interrupt & exceptions cause program discontinuity
- Program discontinuity introduces instruction delay slots
- Zero overhead software pipelined loops
- Software debug features

Control intensive unit of a VLIW processor

Agenda

- Introduction
- VLIW processor overview
- Program control unit of a VLIW processor
- Verification challenges
- Framework for test case generation
- Results and Conclusions
- Questions

Program Discontinuity



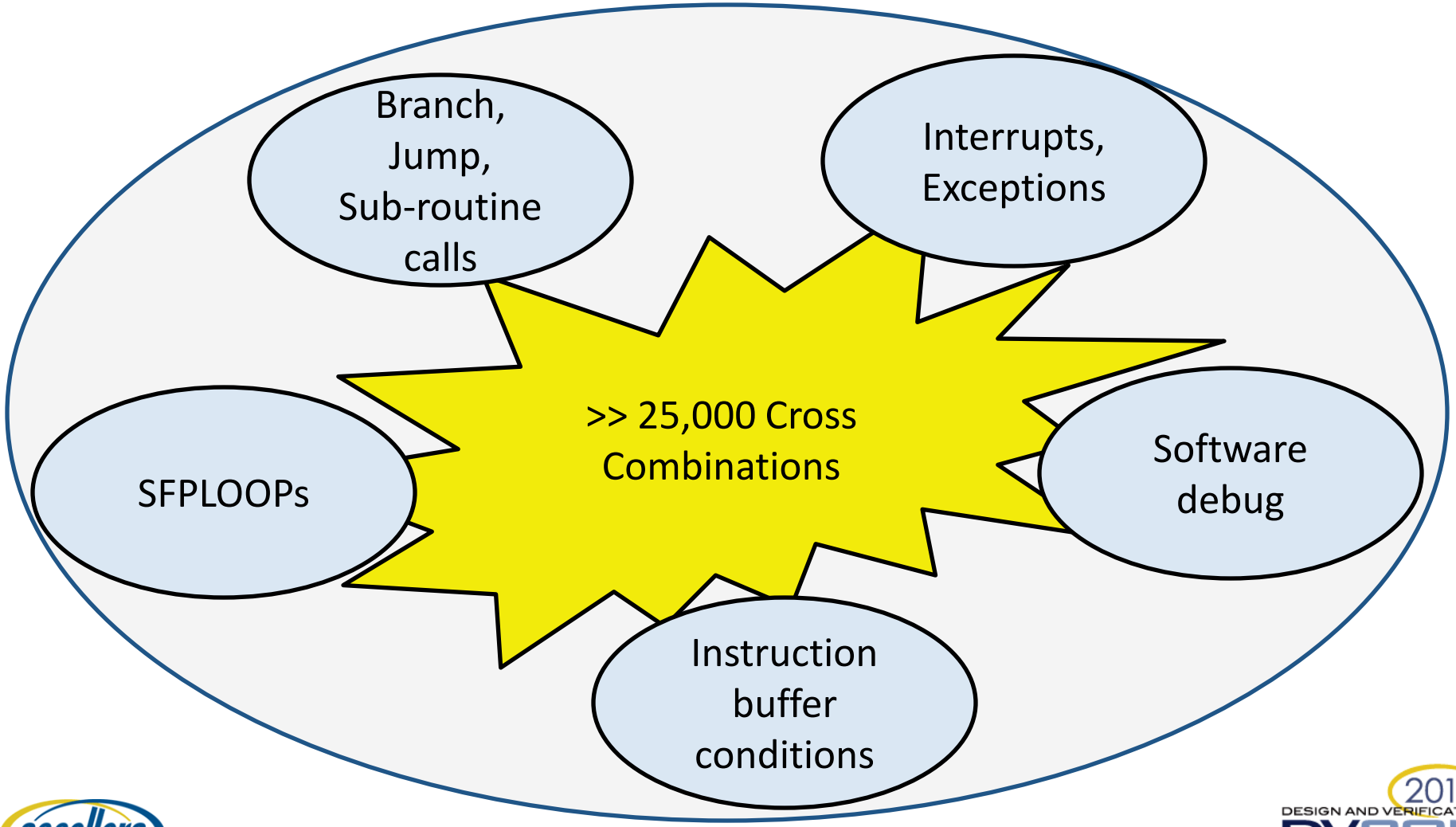
Zero overhead software pipelined loops (SFPLOOP)

- Loop sizes less than, equal, and more than the instruction buffer size
- Interrupts and exceptions before, after and inside the loop
- Nested loops
- Inside branch or jump nested loops
- Branch, jump, and subroutine calls to exit the loop
- Exclusive branch/jumps inside loops

Instruction Buffer & Software Debug

- To be tested with execution stalls, debug stalls, and instruction fetch delays
- Occurrence of program discontinuity when instruction buffer is empty or full
- Software debug features
  - Break point
  - Trace buffer
  - Watch points
  - Single steps

Program Control Test Combinations



Agenda

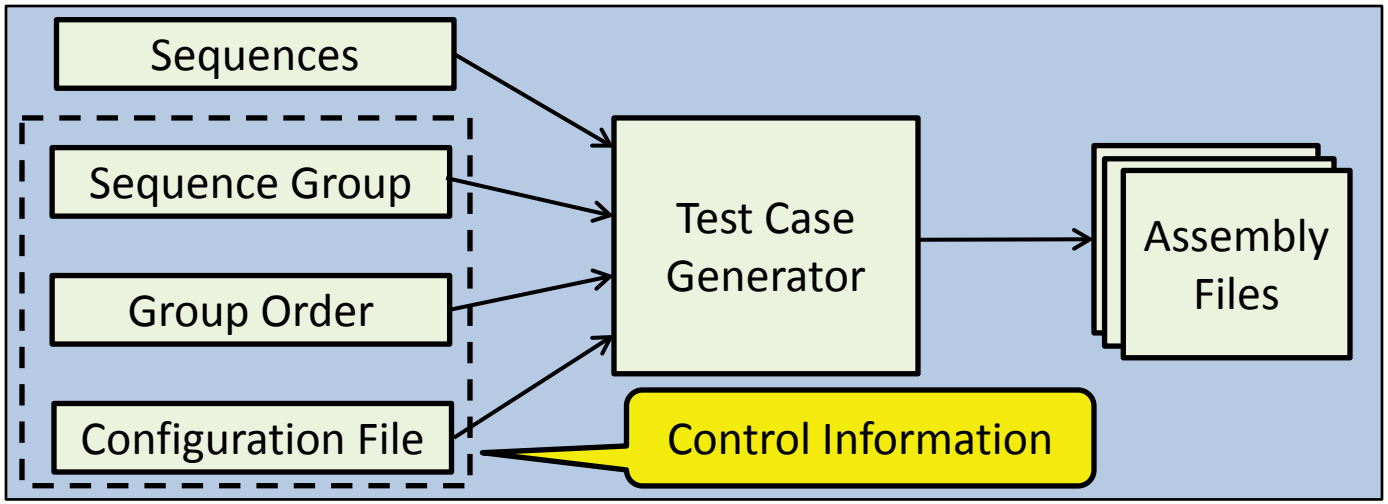
- Introduction
- VLIW processor overview
- Program control unit of a VLIW processor
- Verification challenges
- Framework for test case generation
- Results and Conclusions
- Questions

Why do we need a framework for program control verification?

- Program control feature combinations are huge
  - More than 25,000 test combinations to be verified
- Directed test case development and debug is time and resource consuming
- Generating targeted combinations of branch, jump, interrupt, exceptions and other events are difficult in constrained random simulation
- Need for re-use of test cases for FPGA and Silicon validation
- Shorter silicon validation time

Components of the framework

- Inputs
  - Sequences
  - Sequence group
  - Group order
  - Configuration file
- Test case generator
  - Generates self-checking assembly test cases



Sequences

- Basic elements of the generator
- Specific to targeted features of instruction set
- Multiple sequences per feature
- Sequence consists
  - Control variables
  - Assembly code
  - Behavioral code

Sequences

```
task conditional_branch_seq1;
        bit cb_cbnop;
        cb_cbnop  = $urandom_range(0, 1);

        rmac2_exp(R16, R17, ACC1, ACC0, ACC1, ACC0);        // Complex MAC operation
        if(!eq_flag_set || !cb_cbnop)                       // Call functions based on flag
        begin                                               // and other conditions
                cmul2_exp(R17, R16, R23, R22, 1, R25, R24); // Complex multiplication
                cnmul2_exp(R17, R16, R19, R18, 1, R27, R26); // Complex* multiplica
                accx_truncate(ACC0, 16'd15, R20);           // Data movement
        end

        $fwrite(ASM, “{\n");
        $fwrite(ASM, "S3.RMAC2 R16, R17\n");
        if(cb_cbnop)
                $fwrite(ASM, "[S1.EQ] S1.CBNOP #L%-0d\n", label_count);   // Branch without
        else                                                              // and with
                $fwrite(ASM, "[S1.EQ] S1.CB #L%-0d\n", label_count);      // Delay Slot
        $fwrite(ASM, "}\n");                                              // instructions
        $fwrite(ASM, "S3.CMUL2 SV8, SV11, SV12, #1 \n");
        $fwrite(ASM, "S3.CNMUL2 SV8, SV9, SV13, #2 \n");
        $fwrite(ASM, "S2.MOV ACC0, #0, R20\n");
endtask
```

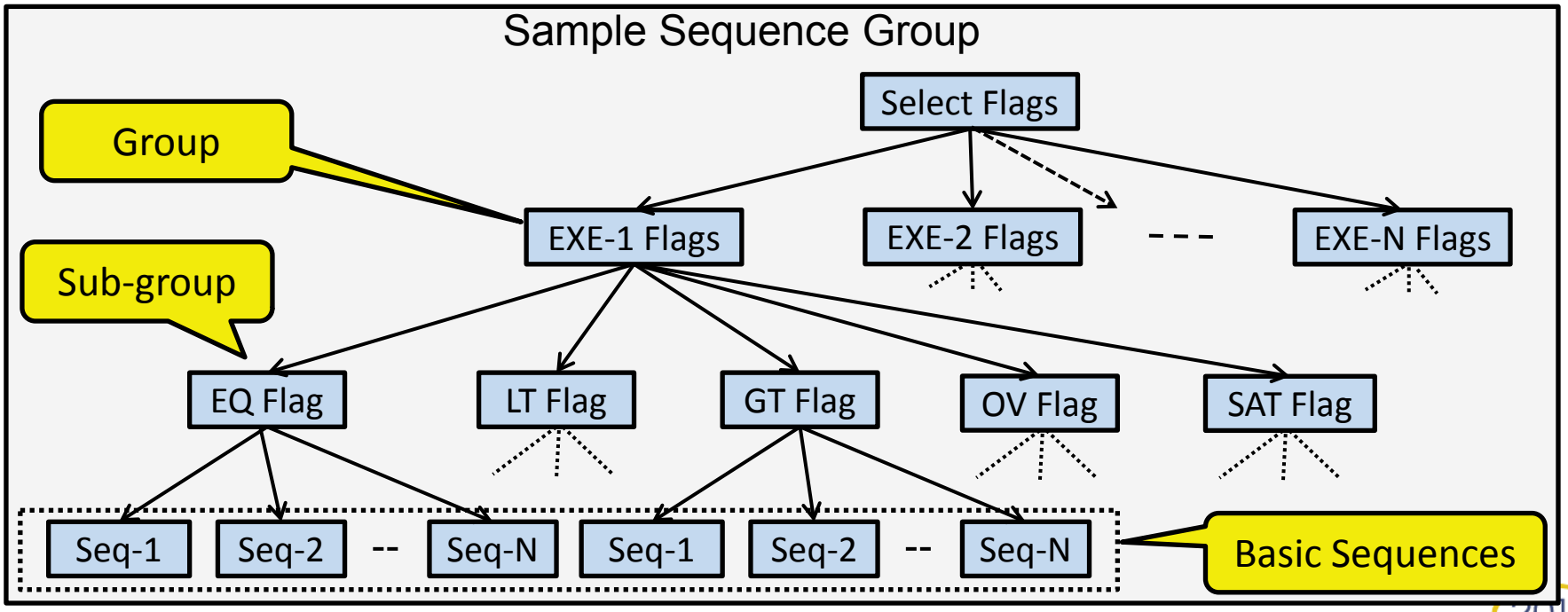
Control Variables

Equivalent behavioral of assembly code

Assembly code

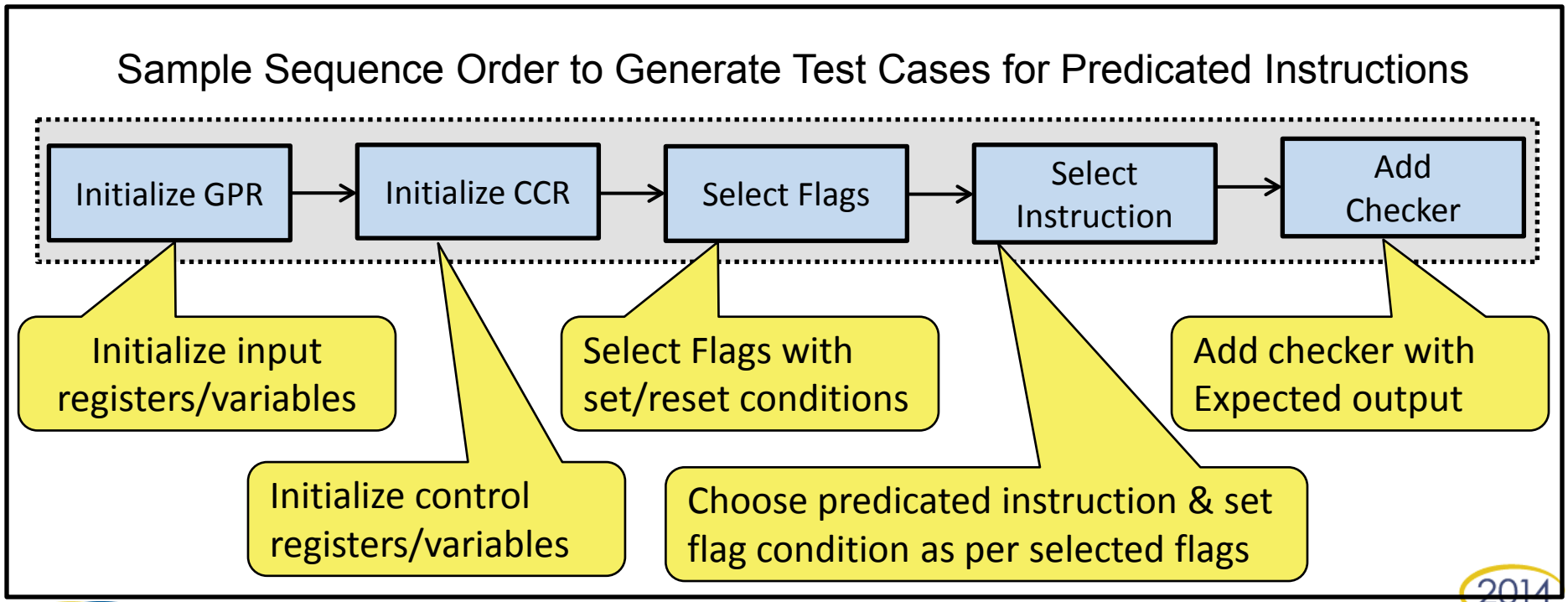
Sequence Group

- Hierarchical group of sequences
- Consists of sequences, sub-groups and groups
- Groups are created based on features and its variants



Sequence Group Order

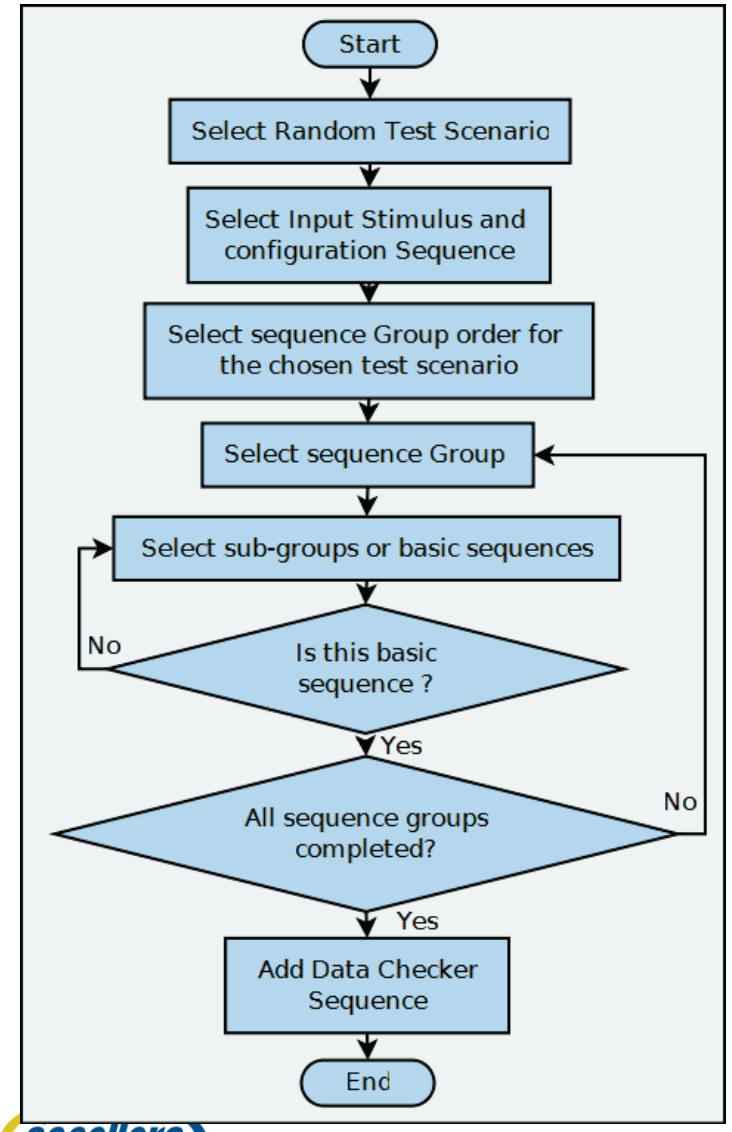
- Information to select groups in a defined order
- Specific to targeted test scenarios
- Used for Randomizing and stitching sequences



Test Case Generation

- Test case generated by randomizing and stitching sequences as per Sequence Group Order
- Input stimulus through pre-defined GPRs
- Outputs re-directed to pre-defined GPRs
- Test case generation constrained through *configuration file*
- SV *randsequence* or similar features are used for randomization
- Checkers for checking test case status

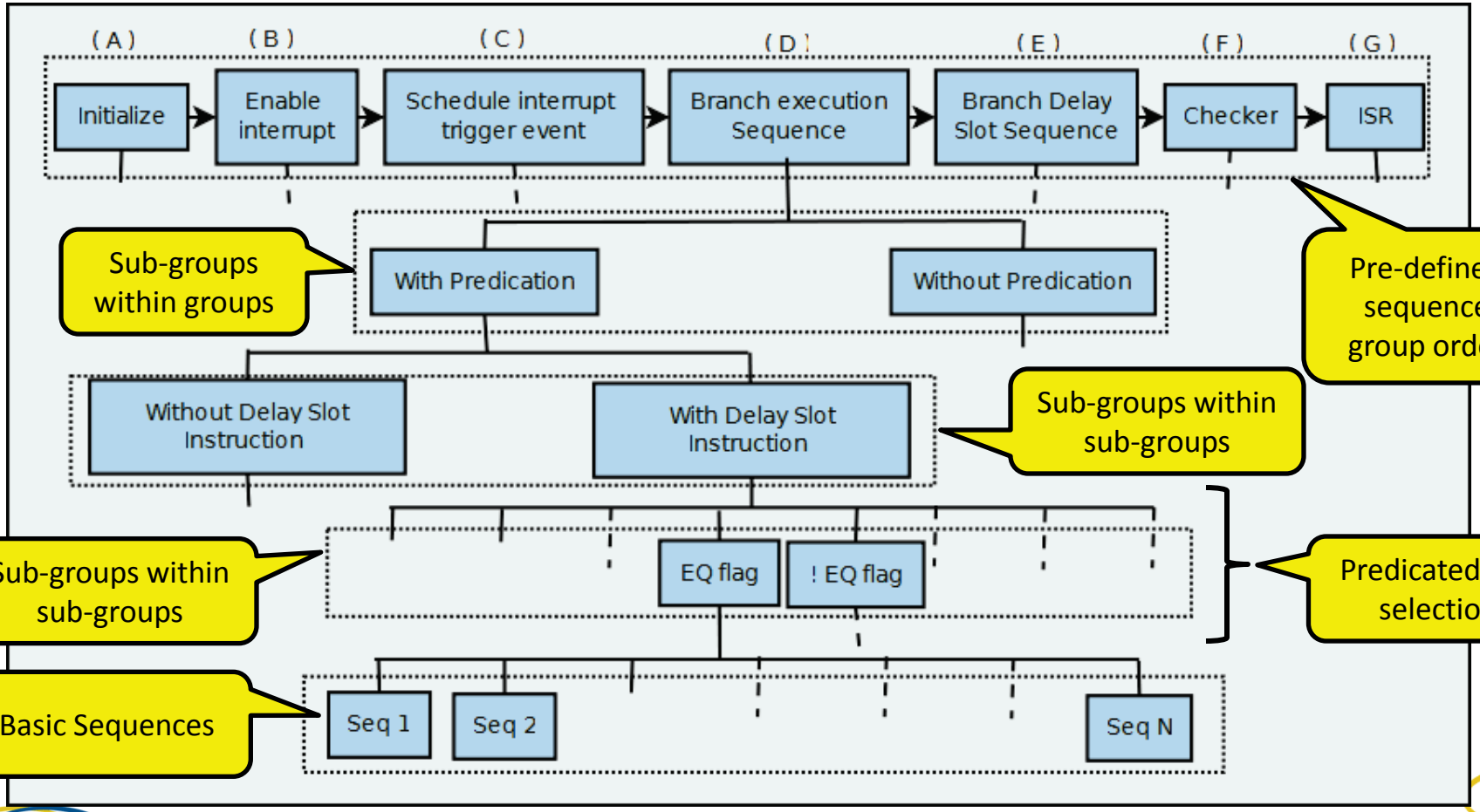
Generation Flow



- Selects test scenario
- Selects sequence group order
- Initializes input and control variables
- Traverses through groups, sub-groups and sequences hierarchically
- Stitches sequences as per sequence group order
- Calculates expected output
- Adds checker for self-checking

Example

- Interrupt before/after/along with branch execution



Agenda

- Introduction
- VLIW processor overview
- Program control unit of a VLIW processor
- Verification challenges
- Framework for test case generation
- Results and Conclusions
- Questions

Results and Conclusions

- Used for verification of Program Control Unit of multiple VLIW cores
- Tests were re-used for FPGA and Silicon validation
- Achieved 100% code and targeted functional coverage
- No bugs found during system level verification and Silicon validation

Questions