Soft Core Embedded Processor-Based Built-In Self-Test of FPGAs

Bradley Dutton & Charles Stroud
Dept. of Electrical & Computer Engineering

AUBURN UNIVERSITY
SAMUEL GINN
COLLEGE OF ENGINEERING

Presentation Outline

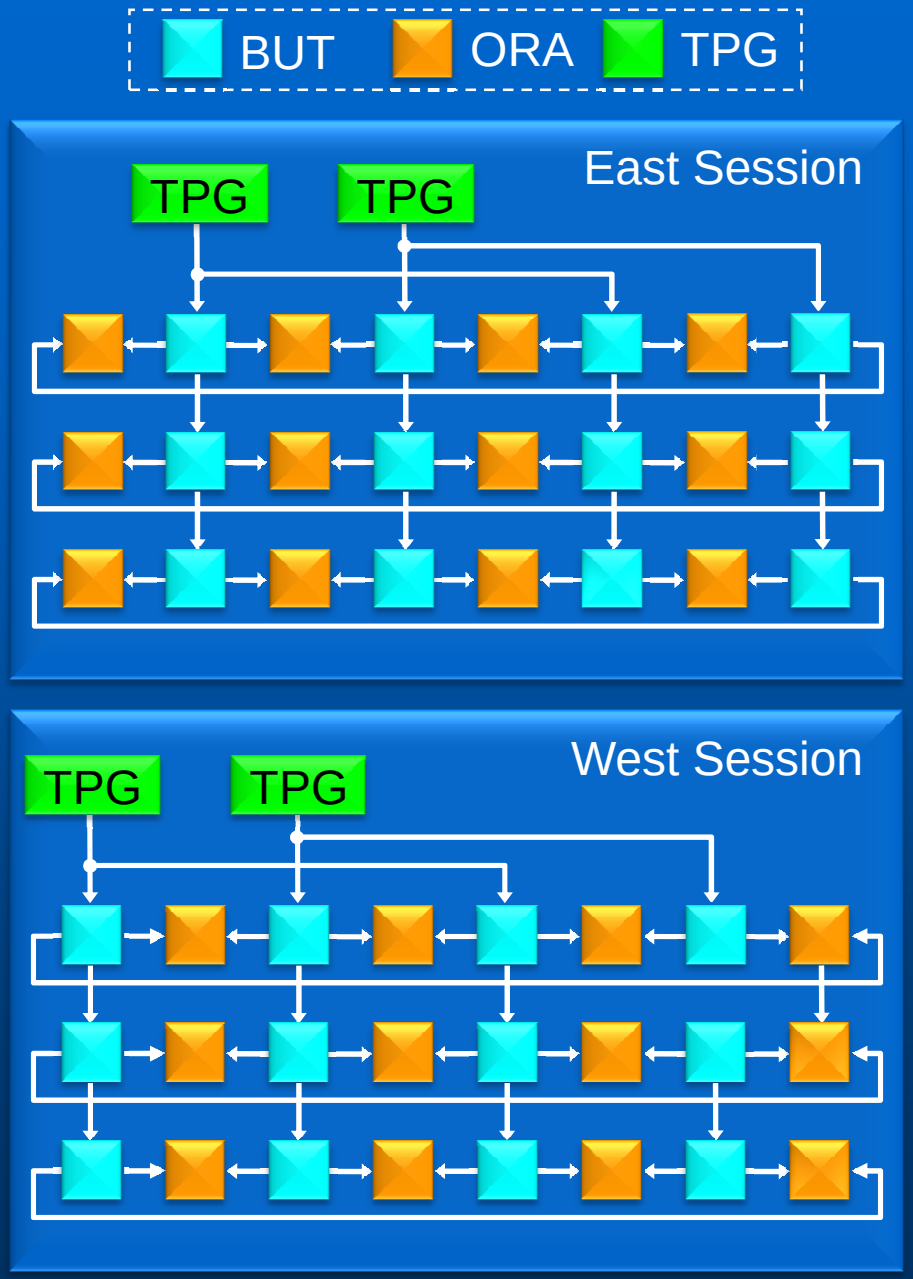
- Background
  - What is Built-In Self-Test (BIST) for FPGAs?
  - Typical BIST architecture
  - Hard processor-based BIST and diagnosis
    - Atmel AT94K SoC
- Embedded Soft Processor-based BIST
  - Hardware development
  - Software development
    - Configuration data compression
- Implementation Results in Virtex-5 FPGAs
  - Number of downloads and test time
- Conclusions

BIST for FPGAs

- Basic idea: reprogram FPGA to test itself
  - No area overhead or performance penalties
- Applicable to all levels of testing
  - Application independent testing
    - A generic test approach for a generic component
  - Good diagnostic resolution
- Cost:
  - Memory to store BIST configurations
    - Goal: minimize number and size of configurations
- Test time = download + execute + results
  - Dominated by download time
    - Goal: minimize downloads and/or download time
  - Results retrieval is second
    - Minimal if high diagnostic resolution is not required

Objective

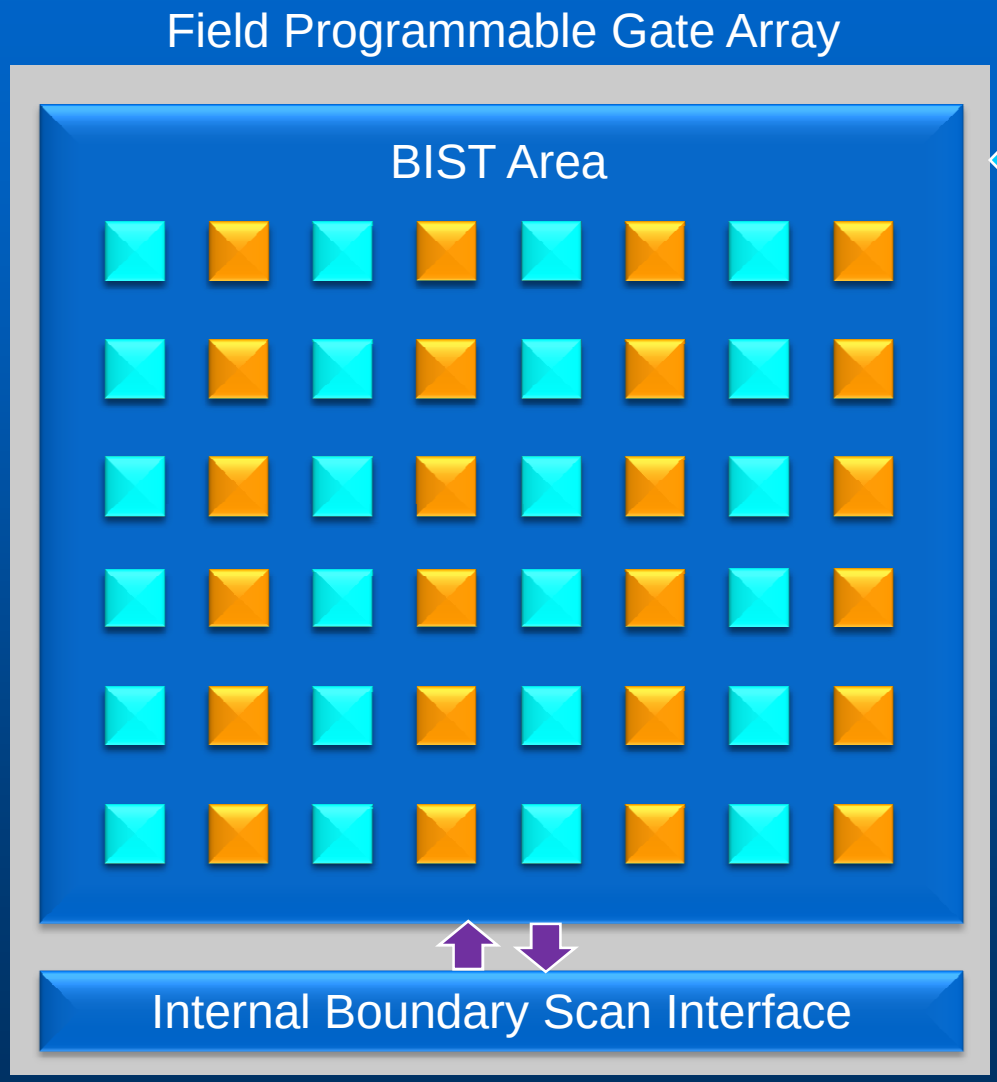
- Reduce test time and minimize system requirements for BIST of FPGAs
  - Minimize downloads via slower external configuration interface
    - Use embedded soft-core processor to perform reconfiguration for all test *phases* of a particular resource under test
  - Use “intelligent” reconfiguration techniques to significantly compress configuration files
  - Move complex BIST control logic into the FPGA fabric
    - Desirable for in-system fault-tolerant applications where resources are limited



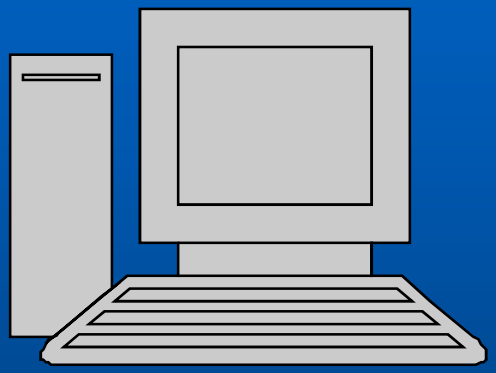
Typical BIST Architecture

- Configurable logic block (CLB) BIST
- 2 test *sessions*
  - East & West
  - Every CLB configured as both a Block Under Test (BUT) and comparison-based Output Response Analyzer (ORA)
- Multiple test *phases* per test *session*
  - Test all modes of operation in CLB
  - Virtex-5 requires 6 *phases*

Traditional BIST Approach



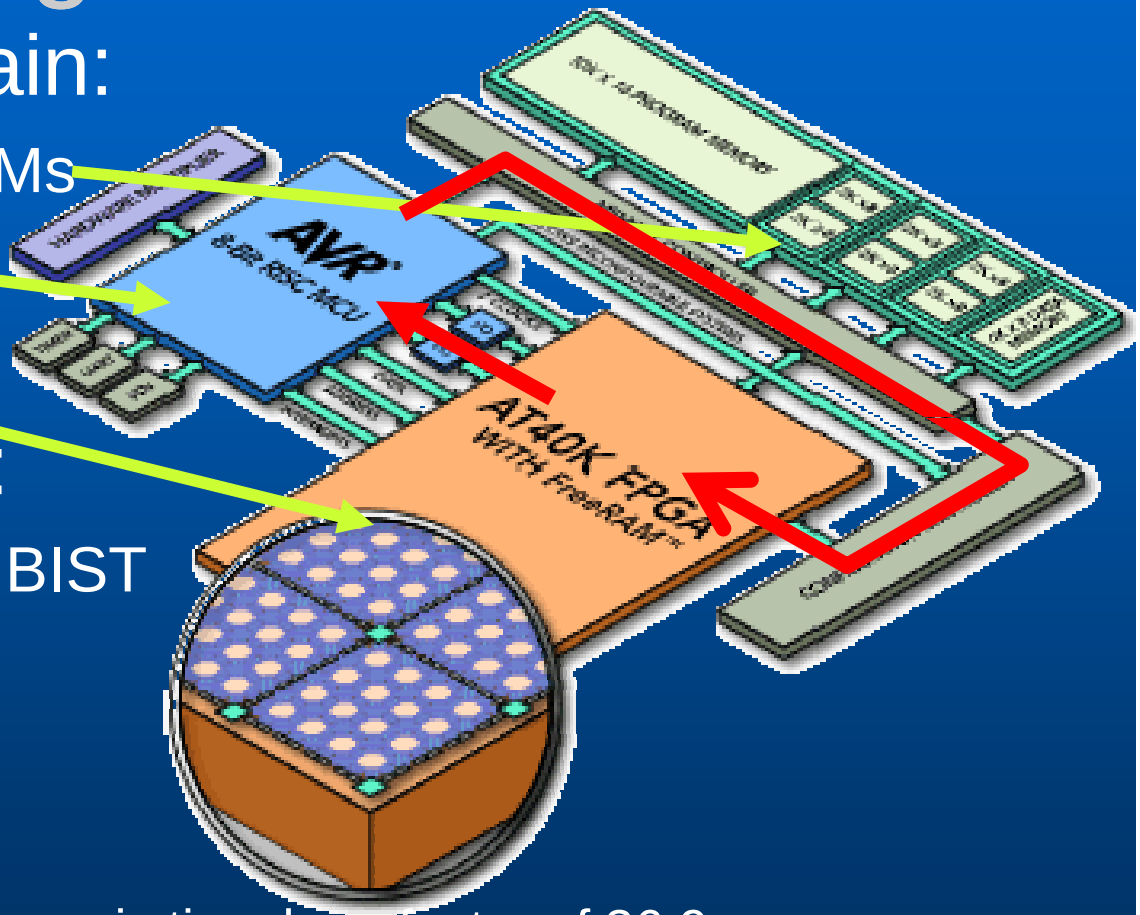
Boundary Scan



- External hardware
  - Performs configuration
  - Controls BIST execution
  - Performs results read back and fault diagnosis

Atmel Hard Processor-Based BIST and Diagnosis

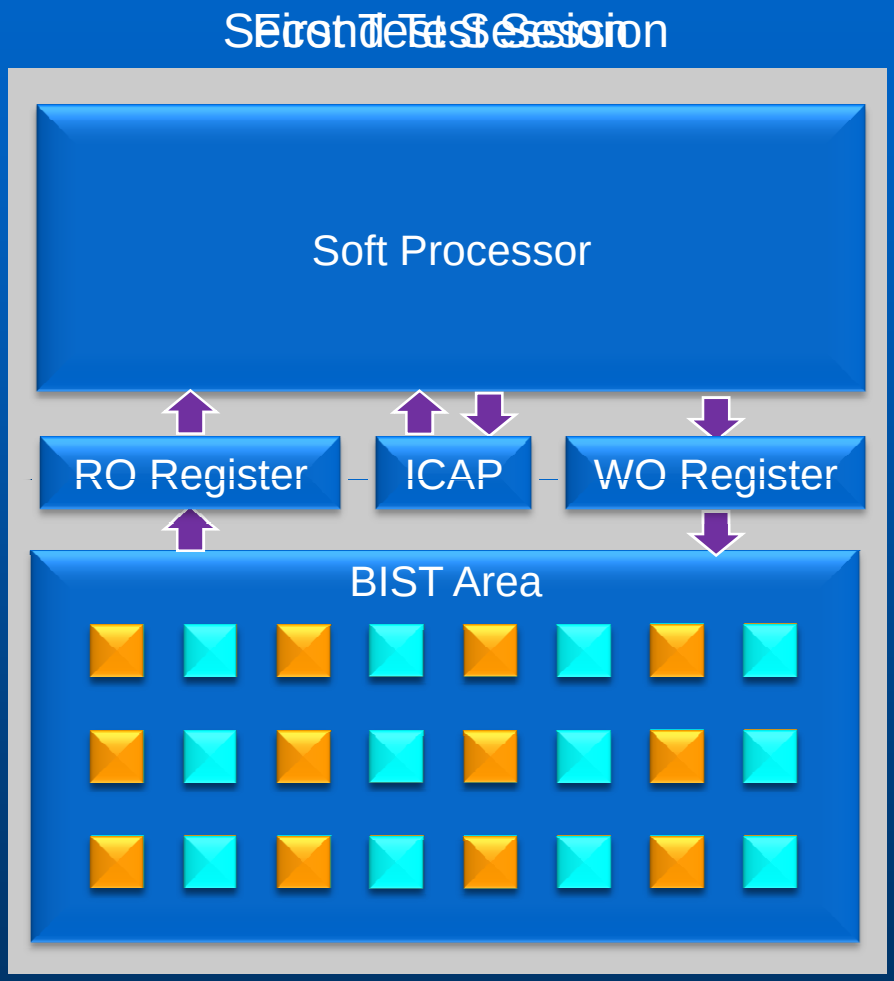
- Atmel SoCs contain:
  - Program & Data RAMs
  - Processor core
  - FPGA core
- Use processor to:
  - Configure FPGA for BIST
  - Run BIST
  - Get BIST results
  - Perform diagnosis
    - Reduces test and diagnosis time by a factor of 36.9
    - Store only one BIST and diagnostic program on-chip



Soft Processor-Based BIST

- Perform partial reconfiguration, control BIST execution, determine pass/fail status, and fault diagnosis in device under test
  - Reduces number of configurations via external configuration interface
  - Eliminates complex processing and control in external hardware
  - Requirements:
    - Internal access to the configuration memory (ICAP)
    - Sufficient resources to implement a processor in half of the programmable resources of the smallest supported device

Embedded Processor-Based BIST



- 4 test *sessions*
  - West & East Top
  - West & East Bottom
  - Worst case, BIST for all other resources require less sessions
- All test *phases* configured by soft processor
- Custom memory mapped registers included for control of BIST logic
- Results reported by processor via user defined interface

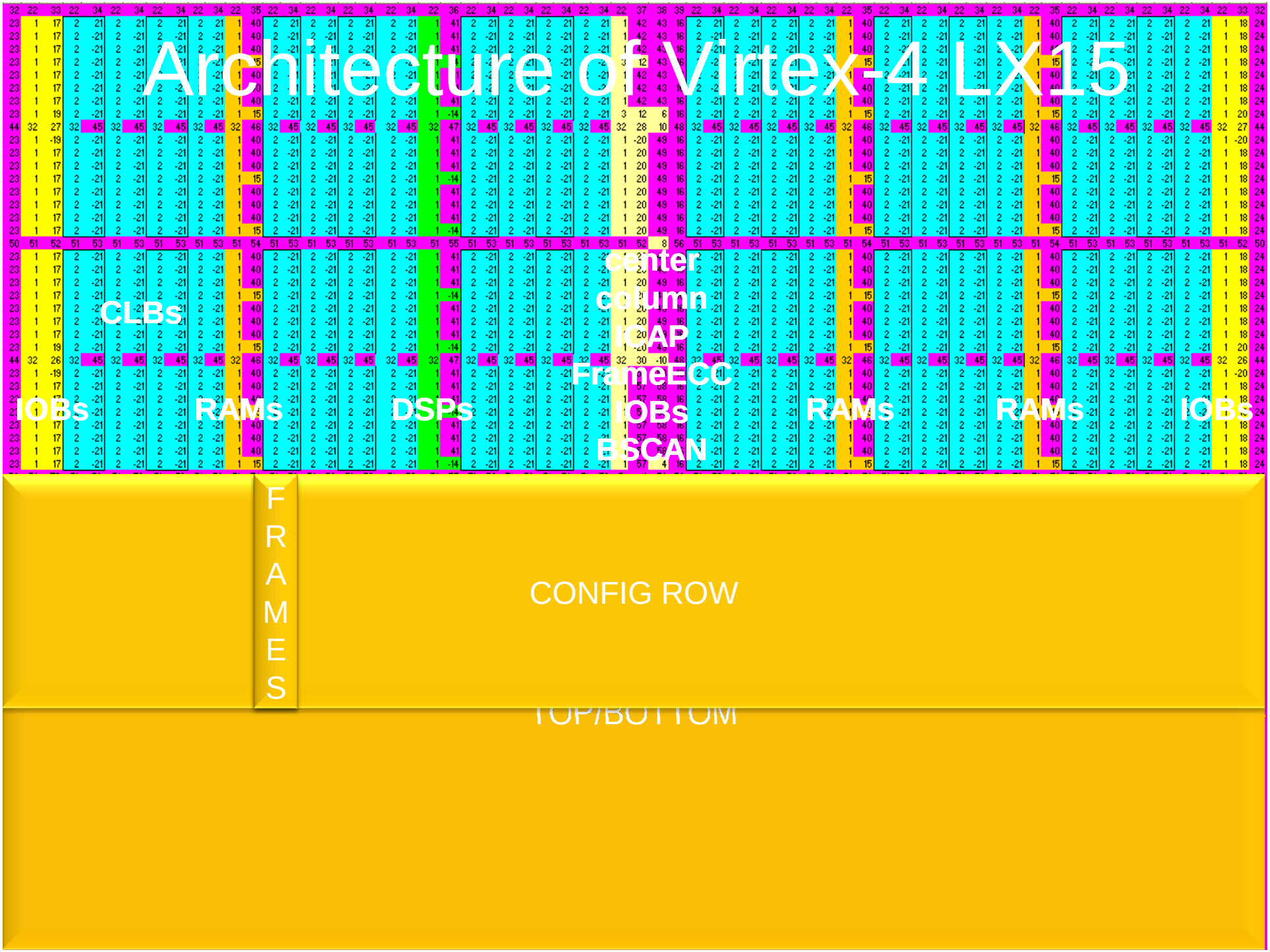
MicroBlaze Software Development

- MicroBlaze is a 32-bit RISC soft processor optimized for Xilinx FPGAs
  - Highly configurable using Xilinx EDK (Embedded Development Kit) and royalty free
  - Software developed in C language
  - Compiled machine code stored in Block RAMs in FPGA fabric (i.e. program memory)
  - Additional Block RAMs used for data memory
  - Only the software changes for BIST of different FPGA resources in a particular device

Embedded BIST Algorithm

- MicroBlaze Soft Processor reconfigures BUTs for all test *phases* after a single download
  - All device specific information is inserted in the compiled program using C preprocessor directives

```
1:      for all test phases do
2:          for all configuration rows in BIST half do
3:              for all frames in reconfiguration structure do
4:                  construct configuration frame
5:                  muti-frame write to all BUTs in half & row
6:              end for
7:          end for
8:          execute BIST phase
9:          get BIST results
10:     end for
11:     set done bit in WO control register
```


Architecture of Virtex-4 LX15
CLBs
IOBs
RAMs
DSPs
center column
IOBs
RAMs
RAMs
IOBs
IOB FrameECC
BSCAN
FRAMES
CONFIG ROW
TOP/BOTTOM

Configuration File Compression

- Required because partial reconfiguration files are device dependent and too large to store on device
- Performing “intelligent” configuration using embedded processor makes possible further compression of configuration files
  - Common instructions stored once
  - Store only 1 row of configuration data
  - Store smallest repeating pattern of 32-bit configuration words in 41-word frame
    - 2-8 words, depending on resource under test
  - Generate frame addresses algorithmically
  - Discard Hamming code in word 20
- Eliminates all device dependencies
  - Size of compressed configuration data is independent of the device under test

Frame Addressing Algorithm

```
struct framedata {
   unsigned int numword;              //# of words
   unsigned int word[MAXWORD];        //config data
   unsigned int numminor;             //# of addresses
   unsigned int minor[MAXMINOR]; //minor addr
};
struct partialconfig {
   unsigned int numframe;
   struct framedata frame[MAXFRAME];
} config[NRECONFIG] = {
//compressed frame data placed here
};
```

- Compressed configuration files are reconstructed algorithmically given the type of device under test

```
1:      for all configuration columns do
2:          if column is block under test then
3:              for all minor addresses in compressed config (numminor) do
4:                  multi-frame write to row, column, & minor
6:              end for
7:          end if
8:      end for
```

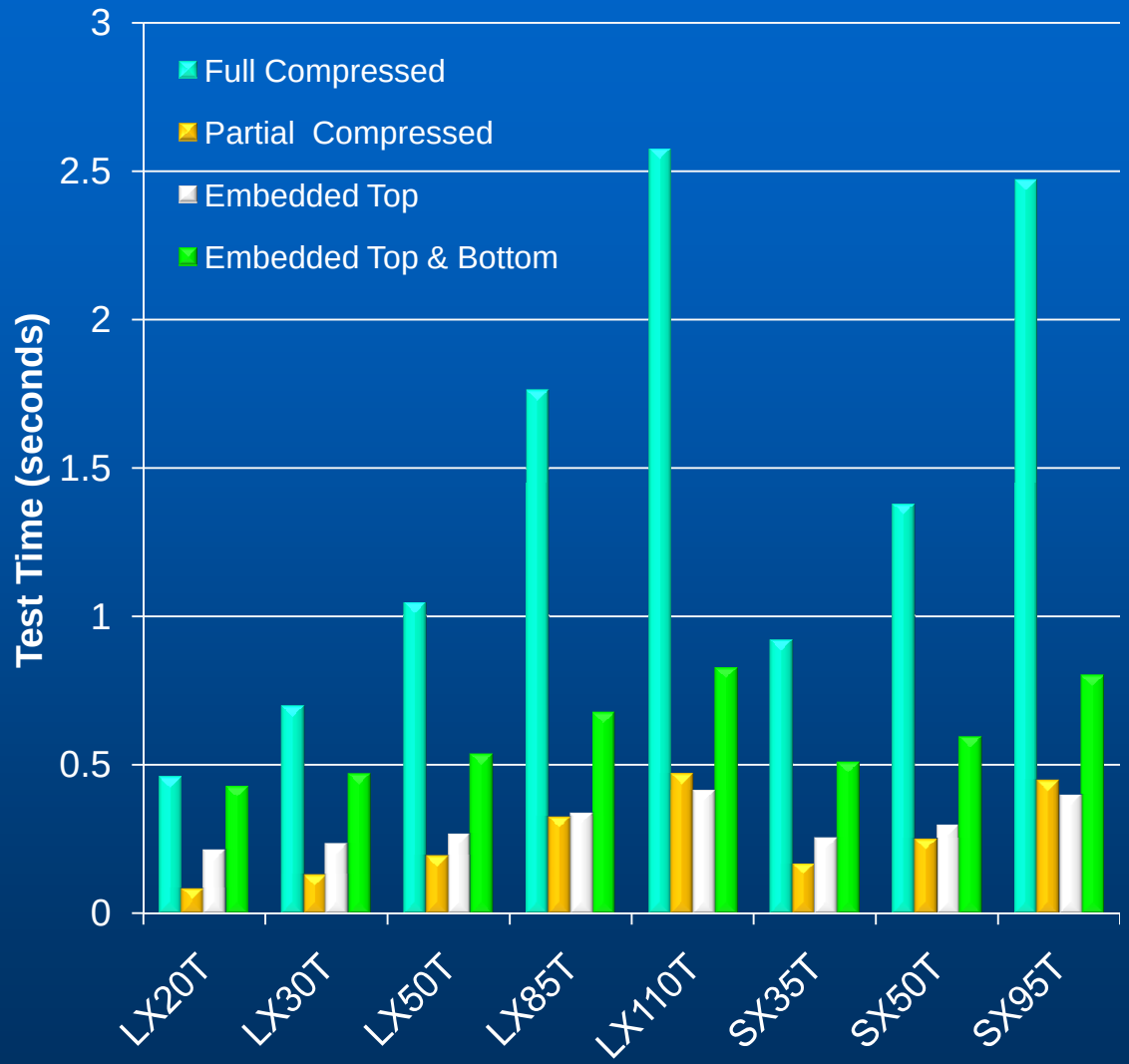
Configuration File Compression

- Size of our compressed configuration files is independent of the device under test

Numbers for Virtex-5 LX30T

BIST Session	External Reconfigurations (Sessions)	Emb. Processor Reconfigurations (Phases)	Original File Size (Bytes)	Compressed Size (Bytes)
CLB East	2	5	41,360	820
CLB West	2	5	41,360	820
LUT-RAM	2	4	10,944	1,232
I/O Logic	1	5	11,308	1,236
I/O SerDes	1	8	94,432	2,680
CRC	1	1	4,716	184
DSP	1	9	28,836	1152
Block RAM	2	5	285,040	4920
ECC RAM	2	2	19,384	1200
FIFO	2	3	29,076	1800
FIFO ECC	2	1	9,692	600

Total CLB Test Time in Virtex-5



- Worst case 3x increase in test time versus external configuration with partial reconfiguration
  - Due to the highly irregular structure of the MicroBlaze processor
- 2.2x speed-up versus full compressed configurations

Implementation in Virtex-5

- MicroBlaze configured with
  - Hardware integer multiplier
  - Five-stage pipeline
  - 32 KB program memory
  - 32 KB data memory
- MicroBlaze occupies
  - 3 DSPs
  - 16 36 Kbit Block RAMs
  - 400 CLBs
- Less than 50% of resources in smallest Virtex-5 device
- Test results and timing information reported via UART interface

Conclusions

- First BIST approach utilizing a soft-processor for reconfiguration of resources under test
  - Reduces the complexity of external test controller
  - Reduces number of external downloads to maximum of two per test session
  - Overall test time is not improved when fault diagnosis is not required
- Configuration file compression techniques can reduce memory requirements for BIST
  - Applicable to any system where a processor controls FPGA configuration
- Good approach for in-system testing
  - Requires only external memory and simple configuration controller

Thank You

Configuration File Compression