Analysis of Picojava Microprocessor

D. Niharika[1] S. Purna Syam Chand[2] P. Vijaya Durga Reddy[3] B. Jhansi[4] Mr. A. Raja[5]
[1,2,3,4,5]Saveetha School of Engineering, India

Abstract— PicoJava is a Java chip created by Sun miniaturized scale frameworks to accelerate execution of Java in implanted frameworks and a regularly referred to reference plan for other Java processors. Data about usage of picoJava is uncommon be that as it may. As opposed to a number on new Java processors which are focused at FPGAs, picoJava was intended for ASICs, and no usage in a FPGA is known forward-thinking. In this paper we show the execution and assessment of Sun's picoJava-II microchip in a FPGA.Java is broadly applied in current implanted frameworks because of its article arranged highlights and points of interest, for example, security, strength, and stage autonomy. A Java virtual machine is expected to execute Java programs. Nonetheless, in the greater part of the current answers for Java virtual machines, the overhead of executing object-arranged related directions is critical and turns into the bottleneck of framework execution. To take care of this issue, a novel Java processor called jHISC is proposed, which for the most part targets J2ME and installed applications. In jHISC, the item situated related guidelines are actualized by equipment legitimately, as an equipment decipherable information structure is utilized to speak to the article. The total framework with 4 KB guidance reserve and 8 kB information store is depicted by VHDL and executed in a Xilinx Virtex FPGA. It involves 601 859 identical entryways and the greatest clock recurrence of the framework is around 30 MHz. Contrasted and PicoJava II, the general execution is accelerate 1 to 7.4 occasions and the execution proficiency of article situated related byte codes is improved by 0.91 to 13.2 occasions for a similar clock recurrence.

Keywords: Picojava Microprocessor, Java Processors

I. JAVA PROCESSORS

Since the presence of Java in 1995 numerous tasks, both in industry and the scholarly community, have been dedicated to speedup Java byte code execution through equipment usage. In spite of promising first outcomes Java processors are as yet a specialty item. Numerous mechanical undertakings vanished following a couple of Years. The reality can be clarified by the advances in the nick of time (JIT) compilers. JIT arrangement is the standard execution method of the JVM in work area and server situations. In the inserted area, where memory assets are rare, a Java processor is as yet a choice. Particularly progressively frameworks JIT assemblage isn't plausible. The assemblage during run time is difficult to foresee and presents a high fluctuation in the execution time. In the accompanying segment we give a diagram of Java processors.

A. Picojava:

A minimal effort RISC chip devoted to executing Java - based bytecodes without the requirement for a mediator or JIT compiler.picoJava legitimately executes the Java Virtual Machine guidance set. Accordingly, Java programming applications are up to multiple times littler in code size and up to multiple times quicker - in this manner lessening memory prerequisites - and multiple times quicker than Java translators running on standard CPUs.It does exclude any memory or I/O interfacelogic. Or maybe, engineers can add their very own rationale to modify memory and an interface.

B. ASIC Designs:

Sun presented the main adaptation of picoJava [19] in 1997. Sun's picoJava is the Java processor frequently refered to in research papers. It is utilized as a source of perspective for new Java processors and as the reason for examination into improving different parts of a Java processor. Incidentally, this processor was never discharged as an item by Sun. An update followed in 1999, known as picoJava-II that is presently uninhibitedly accessible with a rich arrangement of documentation [23, 24]. The design of picoJava is a stack-based CISC processor actualizing 341different directions [19] and is the most unpredictable Java processor accessible. As per [10] the processor can be actualized in around 440 K entryways.

II. PICOJAVA ARCHITECTURE

Sun microsystems presented the main rendition of picoJava [19] in 1997, directed at the installed frameworks showcase as an unadulterated Java processor with confined help of C. An overhaul followed in 1999, known as picoJava-II; this is the adaptation portrayed underneath. After Sun chose to not deliver picoJava in silicon, Sun authorized picoJava to Fujitsu, IBM, LG Semicon and NEC. Be that as it may, these organizations likewise didn't create a chip and Sun microsystems at long last gave the full Verilog code under an open-source permit. Java byte codes with low multifaceted nature are legitimately actualized in equipment, a large portion of them execute in one to three cycles. Execution basic guidelines with higher multifaceted nature, for example summoning a technique, are actualized in microcode. picoJava traps on the staying complex directions, and imitates this guidance. To get to memory, inner registers and for reserve the executives picoJava actualizes 115 broadened guidelines with 2-byte opcodes.

These guidelines are important to compose framework level code to help the JVM. Traps are created on intrudes on, exemptions and for guidance imitating. A snare is somewhat costly and has a base overhead of 16 clock cycles:

- 6 tickers trap execution
- n timekeepers trap code
- 2 timekeepers set VARS register
- 8 timekeepers come back from trap

This base worth must be accomplished if the snare table passage is in the information store and the main guidance of the snare routine is in the guidance reserve. The most pessimistic scenario interfere with idleness is 926 clock cycles .The significant capacity units of picoJava. The whole number unit translates and executes picoJava directions. The guidance reserve is immediate mapped, while the information store is two-way set acquainted, both with a line size of 16 bytes. The stores can be designed somewhere in the range of 0 and 16 KB. A guidance support decouples the guidance

reserve from the interpret unit. The FPU is sorted out as a microcode motor with a 32-piece information way supporting single-and twofold exactness activities. Most single-exactness tasks require four cycles. Twofold accuracy tasks require multiple times the quantity of cycles as single-exactness activities. For minimal effort plans, the FPU can be evacuated and the center snares on skimming direct guidelines toward a product routine to copy these guidelines. picoJava gives a 64-passage stack store as a register record. The center deals with this register document as a roundabout support, with a pointer to the highest point of stack. The stack the executives unit consequently performs spill to and fill from the information store to dodge flood and sub-current of the stack support. To give this usefulness the register document contains five memory ports. Calculation needs two read ports and one compose port, the simultaneous spill and fill tasks the two extra read and compose ports.

The processor center comprises of following six pipeline stages:

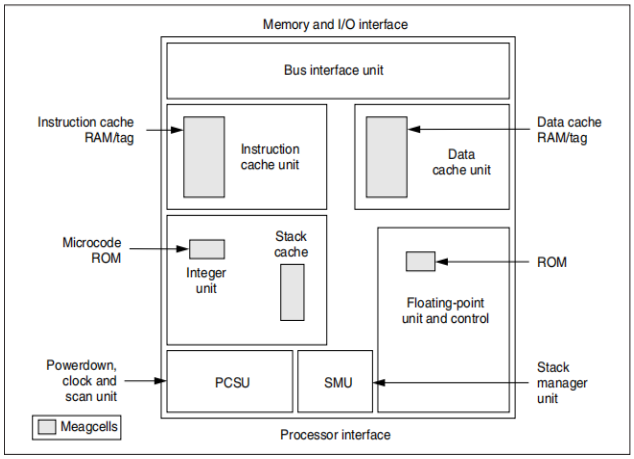
1) Fetch: Fetch 8 bytes from the guidance store or 4 bytes from the transport interface to the 16-byte-profound prefetch cushion.
2) Decode: Group and pre code guidelines (up to 7 bytes) from the prefetch cushion. Guidance collapsing is performed on up to four byte codes.
3) Register: Read up to two operands from the register record (stack reserve).
4) Execute: Execute straightforward guidelines in a single cycle or microcode for multi-cycle directions.
5) Write back: Access the information store.
6) Compose back: Write the outcome once again into the register record.

The whole number unit together with the stack unit gives a system, called guidance collapsing, to accelerate basic code examples found in stack designs, as appeared in Figure 2. The initial phase in A Java guidance

c = a + b;

means the accompanying bytecodes:

iload_1
iload_2
iadd
istore_3



A. *Mapping a Delicate Center to Silicon*

As clarified before, a delicate center gives its client the adaptability of designing the parts to all the more likely fit client's needs and re-focusing to the picked library. Consequently, the center client needs to play out the perplexing errands of blend, improvement, design, testing, and approval of the delicate center. Prior to closing down the delicate center (RTL determination), a delicate center supplier plays out the above undertakings utilizing a configuration instruments stream focused to a specific innovation library (normally a conventional library) to check the structure and give execution/control portrayals. In this procedure, the supplier can bolster a portion of the above undertakings performed utilizing the specific apparatuses. For instance, the picoJava center dispersion gives order content and imperative records for blend utilizing Synopsys Design Compiler. Nonetheless, it's unrealistic for the center supplier to help the above errands for all conceivable plan streams and target innovation libraries. Thus, center clients with various devices and target libraries face issues in mapping the delicate center to the physical level. Furthermore, a delicate center may incorporate some large scale squares or super cells, regularly planned at the physical level. While the delicate center supplier supplies social or utilitarian models for the uber cells utilized in the center (for recreating the center at elevated levels), the center client needs to execute the innovation subordinate super cells—a possibly testing assignment. A delicate center client faces difficulties because of changes in both the plan instruments stream and the objective innovation library. On the off chance that a plan device is utilized for which the delicate center circulation gives no help, the center client must create the vital order records just as set up the earth for that apparatus in the delicate center setting. Changes in the objective innovation library present progressively extreme difficulties, as they influence the whole procedure of mapping the center to silicon. Another innovation library may require change in the content documents and the structure itself, and furthermore countless cycles in configuration stream to meet the necessary execution.

While rationale BIST may perform well on modern application indicated coordinated circuits (ASICs), its practicality on microchips is yet to be researched. To start with, the structure changes required for preparing a chip BIST may accompany unsuitable cost, for example, considerable manual exertion and critical execution corruption. Also, chip are particularly irregular example safe. Because of the planning basic nature of microchips, test focuses may not be worthy as an answer for this issue, as they could present execution debasement on basic ways. Deterministic BIST, then again, may prompt unsatisfactory zone overhead, as the size of the on-chip equipment for encoding deterministic test examples relies upon the testability of the circuit.

Programming based self-testing, which includes the testing of chip utilizing processor instructions. A Software-Based Self-Test Methodology Targeting at Structural Faults Unlike equipment based self-testing, programming based testing empowers the utilization of irregular example age programs with different designs without presenting any test overhead. Additionally, programming directions will have the option to guide test designs through the intricate processor, maintaining a strategic distance from the blockage of the test information due to non-practical control signals. In this paper, we propose a novel programming based processor

self-testing philosophy that conveys auxiliary tests to parts utilizing processor directions. Our self-testing plan incorporates two stages. The test readiness step incorporates the age of feasible tests for segments of the processor and the embodiment of segment tests into individual test marks. Oneself testing step includes the utilization of a product analyzer, which comprises of an on-chip test design age program, a test design application program, and a test reaction investigation program.

In the event that individual test marks are utilized, an on-chip Bus Interface Unit (335) Power down, Clock, and Scan Unit (102) Stack Manager Unit (3061) Floating Point Unit and control (23365) ICRAM (135037) ITAG (13731) DCRAM (89260) DTAG (42374) Stack Cache ucode ROM FP-ROM Integer Unit (83637) Instr. Reserve Unit (32165) Data Cache Unit (8884)

B. Inserted memory Source:

Sun Micro frameworks Figure 3. PicoJava-II processor center Table 2. PicoJava-II: full output versus LBIST LFSR # Test focuses size MISR size Control Observe Area overhead # Test examples Fault inclusion Full Scan - 11.13% 12,736 95.54% LBIST-1 24 41 0 13.06% 32,767 58.81% LBIST-2 24 41 100 13.29% 32,767 82.53% LBIST-3 32 41 100 13.30% 32,767 82.93% LBIST-4 24 41 100 13.30% 1,000,000 84.11%

In the part tests are created under the limitations forced by the processor guidance set, it will consistently be conceivable to discover guidelines for applying the segment tests. In any case, exceptional consideration should be taken for gathering part test reaction. Information yields and status yields have distinctive discernibleness and ought to be dealt with diversely during reaction gathering. Here we represent the engendering of status yields with the ALU in the PARWAN processor. The ALU has four status yields, v (flood), c (convey), z (zero), and n (negative), which can be seen by the guidance succession Instructions 0 – 2 apply a test vector to ALU. The status yields become accessible after guidance 1. Directions 3 – 11 make a picture of the status yields in the gatherer. Initial, an each of the one vector is stacked to the aggregator. In the event that v is one, the every one of the one vector is left immaculate. Something else, a zero replaces the one at the fourth piece from right. Different status bits are dealt with also. After the execution of guidance 11, a picture of the status yield is made in the gatherer. Guidance 12 stores this picture to memory. When all is said in done, however there are no guidelines for putting away the status yields of a segment straightforwardly to memory, the picture of the status yields can be made in memory utilizing contingent directions. This method can be utilized to watch the status yields of any segments.

```
0             lda addr(y)   //load AC
1             add addr(x)
2             sta data_out  //store AC
3             lda 11111111
4             brav ifv      //branch if overflow
5             and 11110111
6  label ifv  brac ifc      //branch if carry
7             and 11111011
8  label ifc  braz ifz      //branch if zero
9             and 11111101
10 label ifz  bran ifn      //branch if negative
11            and 11111110
12 label ifn  sta flag_out
```

III. METHODOLOGY

The picoJava-II CPU core, a reusable core for a variety of Java™ processors, is a synthesizable RTL netlist and can be targeted to different libraries.

There are two levels of verification:

– Stand-alone testing (SAT)
– Full core-level testing, the methodology for which is co -simulation between the architecture simulator and the RTL net list

In addition, you can synthesize the RTL net list into a gate-level net list to ensure that the design meets the timing budget. Usually, several iterations are necessary to shorten the longest timing paths to meet the goals. Finally, gate-level simulation is recommended to ensure that the synthesis is correct.

A. Strategy:

You can use directed tests and Verilog-based random events to achieve high confidence within a reasonable time frame.

picoJava-II Processor Models

The picoJava-II processor core can be simulated at two different levels of abstraction:

– A detailed RTL model which can be simulated using a Verilog language.
– A coarse-grained instruction-accurate simulator written in C.

The RTL model is used for implementing the design in silicon and verifying its functionality prior to tape out, while the instruction-accurate simulator (ias) is used for software development and debugging and as a golden reference model for verifying the RTL design

The picoJava-II distribution contains a system simulation environment to run and debug programs on both the ias and RTL models. The simulation environment, called DSV (Decaf System Verification), is embedded in these two models, and provides functionality such as simulating memory accesses, loading class files into memory, and controlling simulation.

The RTL model is invoked using the command pj2vlog or pj2vcs (depending on which Verilog simulator you use). The instruction-accurate model is invoked using the command ias.

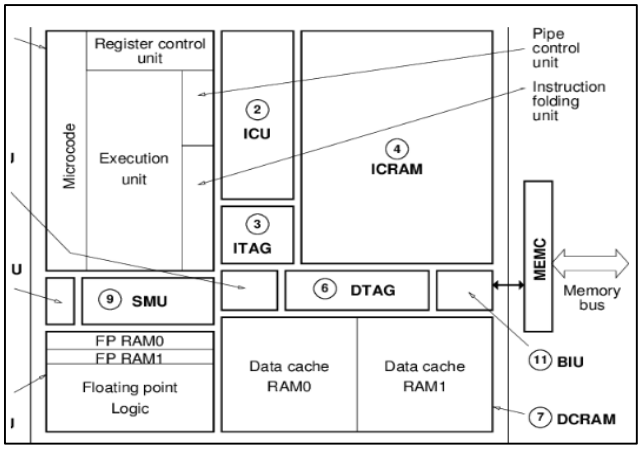
B. Memory Model:

picoJava-II main memory functionality is simulated using a C library (libloader.a). The picoJava-II RTL test bench includes a simple memory controller, which acts as a broker between requests made by the picoJava-II core and the C memory model. The memory controller calls the C library using PLI (Programming Language Interface) functions. PLI functions are defined in the file tools/ldr/src/rw.c. For descriptions of the PLI functions, see Programming Language Interface (PLI) The structure of the C main memory model is such that exactly the same library is linked into the instruction-accurate simulator. ias uses the same interface functions to access main memory as the RTL. However, ias and RTL have independent implementations for cache memories. ias implements a high-level cache simulator (implemented in tools/cache/src/cache.c) while RTL models detailed cache control and datapath logic, using behavioral

models of memory mega cells to simulate storage for data, tag and status RAMs in both caches. Some locations in the picoJava-II memory address space have special meaning in the simulation environment (for both Ias and the RTL model). FIGURE 2-1 and TABLE 2-1 for a list of these special addresses.

C. RTL Verification:

The RTL model is validated via self-checking tests, co-simulation, and RTL monitors two



– To Run Simulation with Verilog-XL
– Provide rad.v to Verilog as input (instead of the Verilog source). For example,

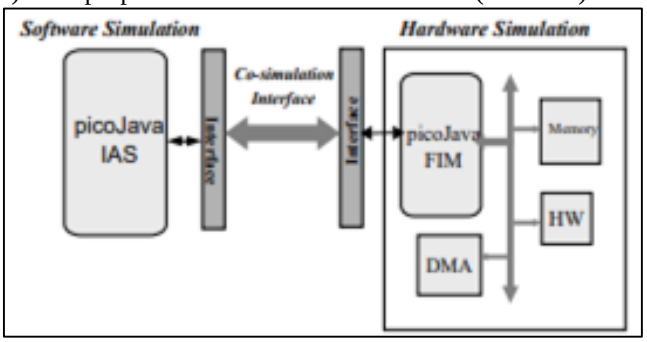
type the following command:
% pj2vlog rad.v +cosim+ias

D. Co-simulation Environment:

The interface and conversation mechanism between a software program factor and a hardware component can be validated with the aid of simulating the software program factor using a hardware simulation model of the processor. However, simulation the usage of a processor hardware model can be prohibitively slow. As the processor core is pre-validated, hardware simulation can be prevented via the usage of an abstract, Instruction Accurate Simulator (IAS) mannequin of the processor. In this section, we evaluate the simulation overall performance of an RTL hardware model of the picoJava processor with an IAS model. We additionally existing a speedy and environment friendly co-simulation environment, using the IAS mannequin of the picoJava processor core, for validating the Hardware/Software interface issues of picoJava based totally SoCs. Table 1 suggests the time taken (ms) in simulating quite a few check packages furnished in the picoJava validation test suite [10], the usage of RTL simulation as well as IAS primarily based simulation. From the outcomes in Table 1, it can be considered that the IAS simulation is countless orders of magnitude faster than the RTL simulation. Table 1. Comparison of IAS model with RTL model Program RTL Simulation IAS Simulation Time(ms) Time(ms) arithm int 4460 1.20 dcu 2 7 3960 0.99 fadd zero 109340 forty six icu 3 1 1 3990 1.10 push pop 1 11200 7.50 To successfully use the IAS model of the picoJava processor to validate the interface between software and hardware components, we developed a Hw/Sw co-simulation framework proven in Figure 5. In this framework, the software program simulation makes use of the IAS mannequin of the picoJava processor. For hardware simulation, as a substitute than the use of a RTL model of the processor, a Fast Interface Model (FIM) is used for simulating the interface of the processor with different elements such as the bus interface, interrupts, and timers. However, the FIM hardware simulation have to be synchronized with the IAS software simulation for correct Hw/Sw system simulation. An efficient co-simulation interface has been developed to synchronize between the software simulation (the IAS).

We simulated each of the above mistakes the use of three distinctive simulation methodologies:

1) complete device simulation at RT Level(RTL),
2) system simulation the usage of co-simulation methodology described in part 4 (Cosim), and
3) the proposed interface based simulation (Interface).



and the hardware simulation (the FIM model). Whenever external reminiscence is accessed in IAS, IAS sends the request to FIM using the co-simulation interface, which in turn initiates the read/write cycle on the bus and acknowledges lower back to IAS with the fetched data. Additionally, whenever the FIM receives an interrupt from other hardware components, it is communicated to IAS in software so that the software can execute the gorgeous entice handler. The interfaces between a software factor and different factors in a picoJava based SoC can be verified very quick the usage of this co-simulation surroundings as it avoids the time ingesting hardware mannequin simulation, and as a substitute simulates solely the interface of picoJava with different factors in hardware. To show the effectiveness of the proposed co-simulation, we simulated the interface between the picoJava processor, the DCT aspect and the memory component, the use of each RTL simulation as properly as the co-simulation environment proposed. For a precise photograph (COWWEI) of size 64x64 pixels, the RTL simulation took 12975 seconds while the gadget simulation the usage of the co-simulation environment took solely 138 seconds. From this and other experimental results that we are getting, cosimulation surroundings offers nearly 100X improvement in system simulation speed.

E. Running Simulations:

The picoJava-II distribution presents a take a look at suite containing subsuites that exhaustively take a look at all functionality in the processor. A test is usually a application written in picoJava-II assembly language and assembled using the picoJava-II assembler, or a program written in the Java programming language, and compiled the usage of a Java compiler. The exams may also be run both on ias alone,

or on the RTLmodel and ias collectively (in cosimulation mode).

To run all the checks in the verification suite, you want to perform the following fundamental steps:

1) Build ias.
   The distribution offers a default binary for SPARC/Solaris platforms.
2) If you want to run the assessments on RTL, construct the RTL model.
   This creates a binary called pj2vcs or pj2vlog.
3) Compile the check suites you would like to run.
   The check suites are provided in source form (either picoJava-II assembly language or the Java programming language). The source should be compiled to create .class archives that can then be loaded in the simulators and run.
4) Run the tests.

Run the checks on either the ias or the RTL model directly, or use the Steam script to run tests. Steam is a perl script that offers a flexible surroundings in which to run large batches of tests and document their results.

Example of Interrupt Handler Code in Verilog

```
// define the interrupt level (1-16), 16 is nmi
`define INT_LEVEL 16
// define the trigger signal
`define INT_TRIGGER (neg edge sys .pico.iu .datapath. ucode.u_done)
always @`INT_TRIGGER begin
pending_reg[`INT_LEVEL] <= #1 1'b1;
```

For details, see the following source directories:

Sim/test/pico_vts/basic
sim/test/pico_vts/basic_java
sim/test/pico_vts/instr_tests

Architecture tests for the IU verify the following features:

- Hazard detection and prevention
- Branch prediction
- Folding
- Handling of polymorphic guidelines and boundary conditions

The above test cases verify the following FPU functionalities:

- Corner vectors such as not-a-numbers (NaNs), zeroes, and infinities
- Underflow and overflow
- Operations that result in rounding to zero
- Operations that result in rounding to the nearest mode
- Widening and narrowing conversions
- Operations that contain denormals
- Back-to-back FPU operations
- Bypassing of information logic for all FPU operations
- Holds while FPU operations are underway
- Interrupts while FPU operations are underway

F. Traps and Interrupts

Exceptions in picoJava-II operations are caused by:

- A subset of the Java virtual computer instructions which the picoJava-II core emulates, the usage of software
- picoJava-specific exceptions, such as runtime exceptions and hardware errors
- Hardware-generated interrupts

G. Exception Handlers:

When the picoJava-II core takes a entice for any reason, it units up a lure body and transfers manipulate to the entice handler for that lure type.

The picoJava-II core verification surroundings offers reference implementations of Java digital machine byte codes which cause an emulation trap. These entice handlers are furnished for the motive of verification only.

The verification surroundings also offers reference implementations of handlers for different traps (such as exceptions and interrupts) for the purpose of verification. B default, these entice handlers genuinely abort the test that is running. However, they also have a mode to depend how many times an exception took place at some stage in a test. This mode is beneficial for checking out exceptions. In this mode, the trap handlers, as an alternative of one hundred ten picoJava-II Verification Guide

– March 1999 aborting the program, simply increment a counter in a predefined memory place and return to the test application to continue execution. At the stop of the test, the test software exams these counters and compares the result with the predicted variety ofexceptions of each type.

The counters for every lure kind are saved in reminiscence from place 0xf000 onward, one word for each lure type (including unused entice types). The address of thecounter for a unique entice kind is (0xf000 + trap_type * 4).

To use the mode the place the exception handlers matter the range of instances the exception occurred (instead of aborting the program), specify the +expcount option. This alternative shops the fee 0xf0 to reminiscence area 0xfff0. Exception handlers check this vicinity to determine whether or not to increment the counter and return to the test, or abort it.

Note – When exams are run only on the simulator, this reminiscence region should be explicitly initialized when the mode in which exception handlers count the number of exceptions is used.

To initialize this memory location, enter the following command:% memPoke 0xfff0 0xf0

Note that co-simulation between the RTL and the simulator continues throughout the time the entice is taken and whilst the exception handler is being executed. However, the co-simulation surroundings can't cope with asynchronous exceptions, because it is

Non deterministic when the exception will be mentioned in the RTL. This makes it challenging to compare nation with the simulator after each and every instruction. For checks that workout such exceptions, you can disable co-simulation and alternatively rely on self-checking exams walking on the RTL standalone.

H. Power-On Reset (POR)

Triggered by way of an exterior reset request, a power-on reset (POR) reasons a switch of manipulate to tackle zero When a POR is active, the picoJava-II core ignores all different resets and traps

Runtime Arithmetic

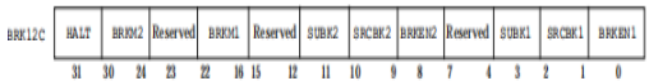
runtime_ArithmeticException is an great arithmetic situation, such as an integer division or the rest operation with zero.
Test with idiv and irem directions (exp_4_1_14_1.code):

– To create this exception
– To take the following exceptions: tt = 0x6d, 0x71, which are runtime_arithmetic exceptions

I. Breakpoints

The core helps statistics and education breakpoints. Data and training breakpoint traps are brought about when there is a suit between an education or data address with the tackle saved in one of the two breakpoint tackle registers, BRK1A and
BRK2A. The BRK12C register controls precisely how the address contrast is performed.
Note – Do no longer set a breakpoint to set off the instruction that sets it. Unexpected results may occur.
To allow a breakpoint:

1) Set up the tackle breakpoint registers.
2) Write to the breakpoint manage register



J. Break point Handlers

The break point handler exception occurs when the program counter (PC) register values suits the ruin point address.
The take a look at case is exp_4_1_18_1.code

1) Memory Protection Fault

The reminiscence safety fault is a synchronous trap that happens when the picoJava-II core accesses outside of the tackle tiers of the USERRANGE1 and USERRANGE2 registers.

Note – The picoJava-I core can take asynchronous traps when it accesses outside of the tackle degrees of the URSERRANGE1 and USERRANGE2 registers.

Test Case	Description
exp_4_1_22_1.code Count/addr1miss.	Tests Lock Count/addr0 overflow and Lock
exp_4_1_22_2.code Count/addr0miss.	Tests Lock Count/addr1 overflow and Lock
exp_4_1_22_3.code Count/addr1.miss.	Tests Lock Count/addr0 underflow and Lock
exp_4_1_22_4.code Count/addr0 miss.	Tests Lock Count/addr1 underflow and Lock

The lock depend overflow lure occurs if the LOCKCOUNT register overflows or below flows when incremented or decremented while entering or exiting a monitor.

K. Lock Enter Miss Traps

The lock enter omit entice takes place if the display that is entering does now not exist in either of the LOCKADDR registers (Lockaddr0 and Lockaddr1).
The check case is exp_4_1_23_1.code

L. Opcodes:

The Cycles column shows a regular range of cycles the directions take, assuming cache hits and no pipeline stalls or exceptions. If an instruction is marked Trap, it is no longer done by way of the hardware however traps to a software program emulation entice handler.

Here is a key to the acronyms in the tables:
LV Local variable load or load from international register or push constant
OP An operation that makes use of the top two entries of stack
BG2 An operation that makes use of the pinnacle two entries of the stack and breaks the group
BG1 An operation that makes use of only the topmost entry of stack and breaks the group
MEM Local variable stores, world register stores, and memory loads
NF Nonfoldable instruction
LDUSE Addition of an more cycle if subsequent guidelines use the load results
Here is a key to the footnotes in the tables:

1) Assumes a non handle reference.
2) Assumes a cope with reference.
3) Optionally traps, relying on the PSR.DRT bit. See Instruction Emulation on page 54.
4) Assumes conditional department is now not taken.
5) Assumes conditional department is taken.
6) Depends on the index and desk change bounds: If the index is less than the decrease bound, then table swap take 10 cycles; if the index is higher than the top bound, then table change takes eleven cycles.
7) May lure if the object reference is now not held in LOCKADDR registers. See Chapter 8, Monitors.
8) May trap if the hardware would have required an examination of the splendid class kind of the checked object.

M. Outlook

In this paper we the implementation of the first industrial Java processor, Sun's picoJava, in an FPGA. To pleasant of our knowledge this is the first successful strive to put into effect the in the beginning ASIC focused layout in an FPGA.

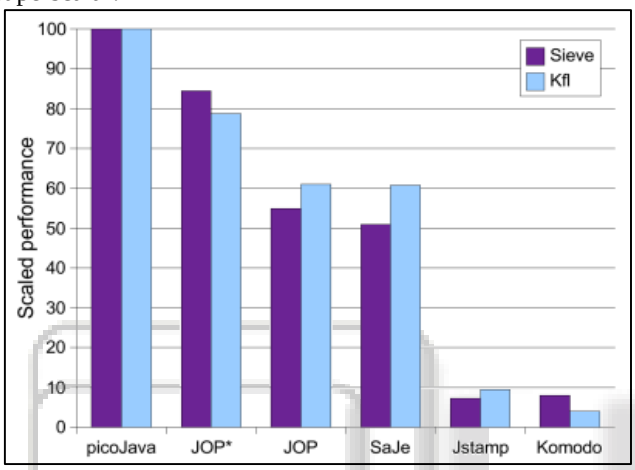
During the port to an FPGA we have encountered various solvable issues in the route of implementing the interior recollections for the a number caches. Results show that picoJava outperforms different Java processors, however that it is two drastically large too.

The diagram adjustments can be overly complex for designs like microprocessors, main to not only significant make bigger in layout time, but additionally unacceptable performance degradation. We have proposed a novel software-based self-testing technique that allows at-speed self-testing using the performance of the processor beneath test. Structural faults are targeted at some point of the self-test whilst the functionality of the processor is used as a car for applying structural tests.

IV. FUTURE SCOPE OF PICOJAVA

Future work will include finishing the implementation of all lure functions and a bigger subset of the Java classification library. With these prerequisites, more complicated benchmarks will be possible and a as a result better understanding of the architecture. We grant a package deal named Harvey to adapt the original sources to the Altera DE2 board (see Appendix for download location and build instructions). We hope that the availability of a pico-Java implementation in an FPGA on a low priced prototyping

board is a groundwork where other researches can consider their ideas within an industrial electricity Java processor. With a real implementation enhancements can be effortlessly tested the usage of a quantitative approach. . The benefits of the proposed approach include enabling at-speed testing with low velocity testers as well as accomplishing excessive fault insurance barring sacrificing area or performance. By breaking up a complicated machine into manageable portions and targeting at man or woman components, we anticipate to apply this method to giant processors and structures in the future. Currently, by way of applying it to the picoJava processor core, we are looking forward to extend the proposed self-test approach to tackle problems related to complicated architectural aspects such as pipelining and superscalar.



Benchmarks for distinct processors

V. CONCLUSION

In this paper, we tackle the issue of validating complicated core-based SoCs. From realistic ride in designing a SoC, we identify various frequent issues in integration of elements of a SoC, and classify them into ordinary interface validation problems. We proposed a methodology to validate the verbal exchange mechanisms and physical interfaces between the components earlier than the whole validation of the machine is performed. Our experimental effects exhibit the full-size reduction in validation time that can be performed the use of our proposed methodology.java possesses language points as protection and object orientation that can substantially enhance improvement of embedded systems.however, implementation as interpreter with JIT compiler are normally no longer possible in useful resource constraint embedded systems. This paper presented the architecture of the JVM. The flexibility of FPGAs and HW/SW co-design makes it feasible to adapt the resource usage of the processor for exceptional applications. JOP has been used in three real-

- world applications displaying that it can complete with widespread microcontrollers. JOP encourages utilization of java in embedded systems. The important aspects are
- Small core that fits in a low cost FPGA
- Configurable resources usage through HW/SW co-design
- Predictable execution time of java bytecodes.
- Fast execution of java bytecodes except JIT -compiler
- Flexibility for embedded systems through FPGA implementation.