February 16, 2011
Berkeley, CA

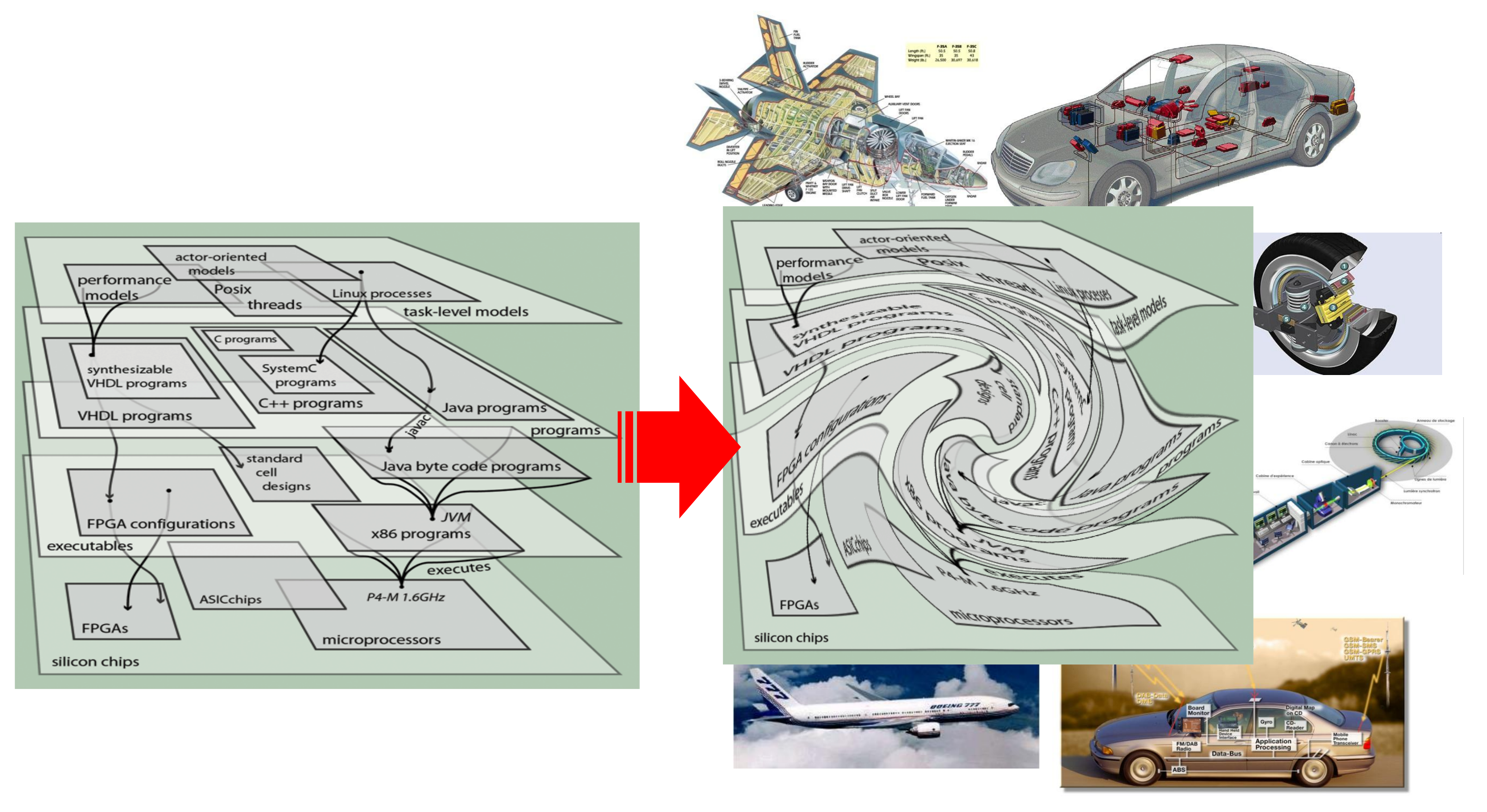
Ptolemy Miniconference

Deadline Instructions in a PRET Architecture

Gage Eads, Edward A. Lee, Isaac Liu, Hiren D. Patel, Jan Reineke

PRET Philosophy

The traditional computing abstractions only concern themselves with the “functional” aspects of a program and not its timing properties. This allows the use of techniques like speculative execution, caches, interrupts, and dynamic compilation that offer improved average-case performance at the expense of predictable execution times. The PRET project aims to improve the timing predictability at all layers of abstraction by carefully reexamining and reworking various architectural and compiler advancements with an eye toward their effects on timing behavior and worst-case bounds.



C Level Constructs

Real-time software engineers require expressive timing constructs at the programming language level. We present three possible control blocks, constructed from PRET deadline instructions: **tryin expired{}**, **tryin catch{}**, and **tryfor{}**.

tryin{} - expired{}

```
tryin(500ms) {
    // Code block to execute
    read_sensor();
}
expired{ patchup(); }
```

Upper Bound (Late Miss Detection)

```
ldr r1, 1                  ; Set seconds register to 1
set_time r1, r2            ; Set r1 to current time + r1,
…                          ;Code block to execute
ldr r3, <TARGET>           ; Load r3 with TARGET address
branch_expired r1, r2, r3  ; Jump to TARGET if
                           ; deadline was missed
```

tryin{} - catch{}

```
tryin(500ms) {
    //Code block to execute
    control_actuator();
}
catch{ panic(); }
```

Upper Bound (Immediate Miss Detection)

```
ldr r1, 1                       ; Set seconds register to 1
set_time r1, r2                 ; Set r1 to current time + r1,
exception_on_expire r1, 1       ; Enable exception ID #1
…                               ;Code block to execute
deactivate_exception 1          ; Deactivate exception ID #1
```

tryfor{}

```
tryfor(500ms) {
    //Code block to execute
    read_sensor();
}
catch{ panic(); }
```

Lower Bound

```
ldr r1, 1            ; Set seconds register to 1
set_time r1, r2      ; Set r1 to current time + r1,
…                    ;Code block to execute
delay_until r1, r2   ; Stall until lower bound met
```

Assembly Timing Instructions

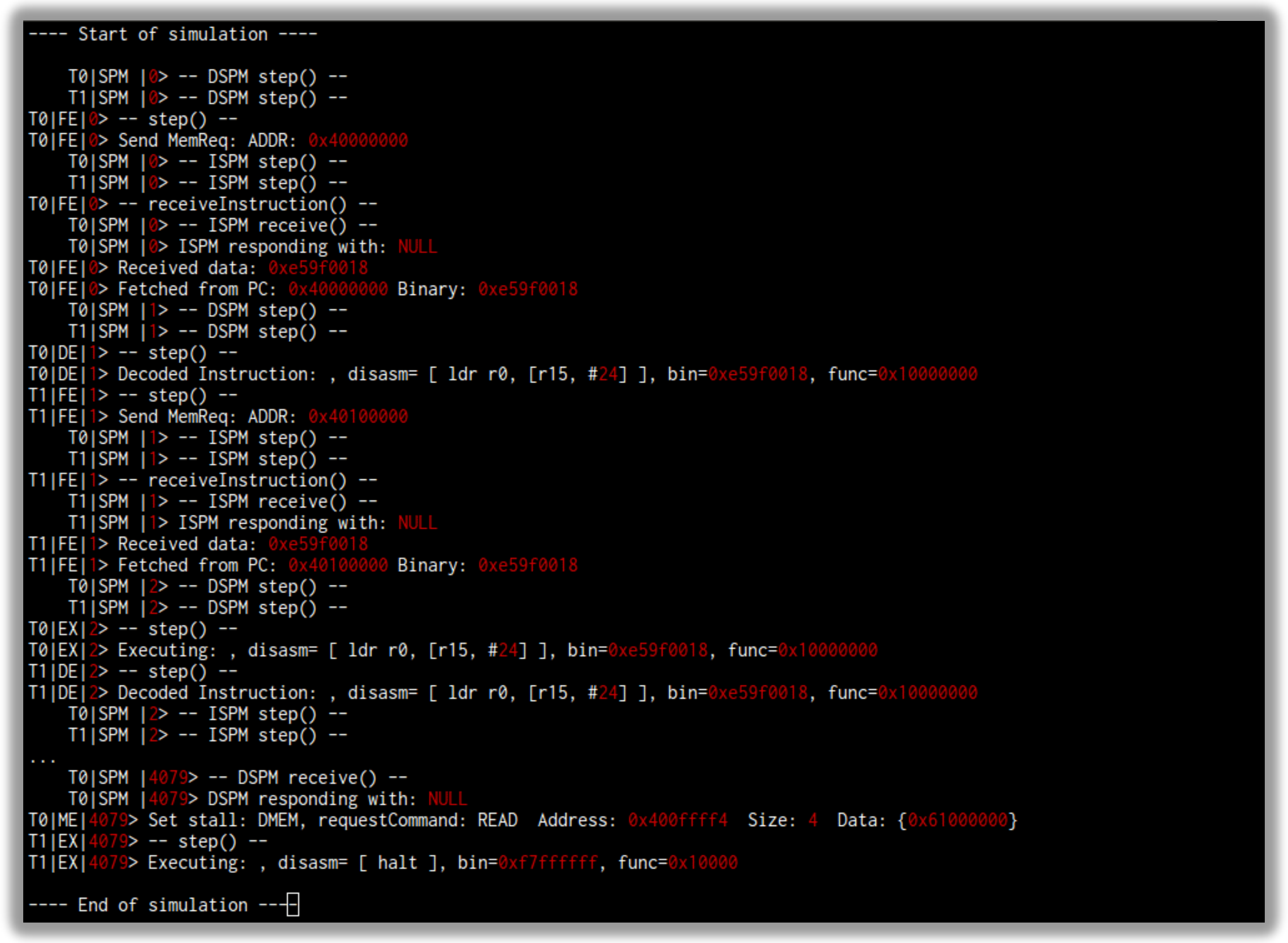
We augment the ARM ISA with five powerful timing instructions. The instructions are implemented as coprocessor accesses to the 64-bit timer coprocessor, which is partitioned into a 32-bit second counter and a 32-bit nanosecond counter. Register operands *rd* and *rm* contain the **second** and **nanosecond** values, respectively.

Assembly Instruction	Behavior
SET_TIME rd, rm	Loads the current timer value plus register contents into the register.
DELAY_UNTIL rd, rm	Stall CPU until timer value > *rd* + *rm*
BRANCH_EXPIRED rd, rm, rn	Branch to target address in register *rn* if timer value > *rd* + *rm*
EXCEPTION_ON_EXPIRE rd, <id>	Arm the processor to throw an exception immediately with id = <id> when the timer > *rd* + *rm*
DEACTIVATE_EXCEPTION <id>	Deactivate exceptions of id = <id>

We design timing constructs such as lower and upper bounded execution through a combination of PRET timing instructions. A fourth construct, MTFD, would statically determine whether a code block can meet its deadline on a given predictable system.

PRET Simulator

We released the PRET Simulator v1.0 in February 2009. A number of class projects and papers leveraged the simulator to explore precision timed software design, such as an automated mapping from a timed functional specification (Giotto) to a PRET architecture.



```
---- Start of simulation ----

    T0|SPM |0> -- DSPM step() --
    T1|SPM |0> -- DSPM step() --
T0|FE|0> -- step() --
T0|FE|0> Send MemReq: ADDR: 0x40000000
    T0|SPM |0> -- ISPM step() --
    T1|SPM |0> -- ISPM step() --
T0|FE|0> -- receiveInstruction() --
    T0|SPM |0> -- ISPM receive() --
    T0|SPM |0> ISPM responding with: NULL
T0|FE|0> Received data: 0xe59f0018
T0|FE|0> Fetched from PC: 0x40000000 Binary: 0xe59f0018
    T0|SPM |1> -- DSPM step() --
    T1|SPM |1> -- DSPM step() --
T0|DE|1> -- step() --
T0|DE|1> Decoded Instruction: , disasm= [ ldr r0, [r15, #24] ], bin=0xe59f0018, func=0x10000000
T1|FE|1> -- step() --
T1|FE|1> Send MemReq: ADDR: 0x40100000
    T0|SPM |1> -- ISPM step() --
    T1|SPM |1> -- ISPM step() --
T1|FE|1> -- receiveInstruction() --
    T1|SPM |1> -- ISPM receive() --
    T1|SPM |1> ISPM responding with: NULL
T1|FE|1> Received data: 0xe59f0018
T1|FE|1> Fetched from PC: 0x40100000 Binary: 0xe59f0018
    T0|SPM |2> -- DSPM step() --
    T1|SPM |2> -- DSPM step() --
T0|EX|2> -- step() --
T0|EX|2> Executing: , disasm= [ ldr r0, [r15, #24] ], bin=0xe59f0018, func=0x10000000
T1|DE|2> -- step() --
T1|DE|2> Decoded Instruction: , disasm= [ ldr r0, [r15, #24] ], bin=0xe59f0018, func=0x10000000
    T0|SPM |2> -- ISPM step() --
    T1|SPM |2> -- ISPM step() --
...
    T0|SPM |4079> -- DSPM receive() --
    T0|SPM |4079> DSPM responding with: NULL
T0|ME|4079> Set stall: DMEM, requestCommand: READ  Address: 0x400ffff4  Size: 4  Data: {0x61000000}
T1|EX|4079> -- step() --
T1|EX|4079> Executing: , disasm= [ halt ], bin=0xf7ffffff, func=0x10000

---- End of simulation ---
```

Debug output from PRET Simulator running threads 0 and 1

Since then, we have made significant changes to the PRET architecture: the ISA changed from SPARCv8 to ARMv4, a predictable memory controller subsumed the memory wheel, and the core became a 4-stage pipeline. Version 2.0 of the PRET Simulator is functional and features all of the aforementioned deadline instructions. It will be released for public use under an open-source license in the near future.

Acknowledgments

This work was supported in part by the Center for Hybrid and Embedded Software Systems (CHESS) at UC Berkeley, which receives support from the National Science Foundation (NSF awards #0720882 (CSR-EHS:PRET) and #0720841 (CSR-CPS)), the U. S. Army Research Office (ARO#W911NF-07-2-0019), the U. S. Air Force Office of Scientific Research (MURI #FA9550-06-0312), the Air Force Research Lab (AFRL), the State of California Micro Program, and the following companies: Agilent, Bosch, HSBC, Lockheed-Martin, National Instruments, and Toyota. The authors acknowledge the support of the Multiscale Systems Center, one of six research centers funded under the Focus Center Research Program.