ARMv8 Simulator

Harshit Patel, Swarnadeep Saha

August 27, 2015

Introduction

ARMv8 simulator is developed to simulate limited instruction set of ARMv8 architecture in pipelined fashion. It is equipped with a debugger having a set of very useful functionalities.

Assumptions

1. There are five stages in processor pipeline, namely Instruction Fetch (IF), Instruction Decode (ID), Execution (EX), Memory Access (MA), Write Back (WB).
2. Instruction cache and data cache are saperate functional units.
3. All attempts to access data from instruction cache and data cache are hits.
4. All memory words are 0s unless something is stored into memory through program.
5. Functional unit to shift bits in operands of instruction is implemented as combinational circuit and has 0 latency.
6. Total 3 cycles are required to fetch an instruction from instruction cache and **2 extra cycles are not considered stall cycles.**
7. There are two separate register files:
   (a) Interger Register File (intRF) of 32 64-bit registers.
   (b) Float Register File (floatRF) of 32 128-bit registers.
8. Operands can be forwarded from EX/MA interstage register and MA/WB interstage register in operand forwarding mode.
9. Switching happens every cycle in a functional unit for the duration it is active.

Source Files

All source files are in [project_path]/ARMv8/ folder. Majority of source files have names of the format [stage_function]_[instruction_class].py where [stage_function] and [instruction_class] are as below:

- [stage_function]:
  - **opfetch**: These files contain code for decoding and operand fetching for instructions of [instruction_class] in ID stage.
  - **executor**: These files contain code for executing instructions of [instruction_class] in EX stage.
  - **memaccess**: These files contain code for memory read/write operations for instructions of [instruction_class] in MA stage.

  - **writeback**: These files contain code for writing results in destination registers for instructions of [instruction_class] in WB stage.

- [instruction_class]:
  - **ALU**: Implements CLS and CLZ instructions.
  - **FP_addSub**: Implements scalar and vector varients of FADD and FSUB instructions.
  - **FP_maxMin**: Implements scalar and vector varients of FMAX and FMIN instructions.
  - **adc**: Implements ADC instruction.
  - **addSub**: Implements ADD, ADDS, SUB and SUBS instructions.
  - **bitwise_shift**: Implements instructions.
  - **branch**: Implements B, BCOND, BL, BR, BLR, RET, CBZ and CBNZ instructions.
  - **conditional**: Implements CSET, CSINC, CNEG, CSNEG, CSINV and CSINV instructions.
  - **loadStore**: Implements LDR, LDRB, LDRSB, LDRH, LDRSH, LDRSW, LDP, STR and STP instructions.
  - **logical**: Implements AND and ANDS instructions.
  - **misc**: Implements ADR, ADRP and NOP instructions.
  - **mov**: Implements MOV, FMOV and FMOV general instructions.
  - **moveWide**: Implements MOVK, MOVN and MOVZ instructions.
  - **mulDiv**: Implements UMULL, UDIV and SDIV instructions.
  - **rotate**: Implements ROR instruction.
  - **shift**: Implements LSL, LSR and ASR instructions.

There are several other source files as follows:

- **armdebug.py**: Implements debugger.
- **utilfunc.py**: Implements utility functions for setting/getting registers from register file, storing/loading memory locations, integer and floating point addition/subtraction etc.
- **const.py**: Implements global constants and flags used for synchronization and decision making.
- **config.xml**: Contains hardware specifications of processor.
- **config.py**: Parses config.xml to read hardware specifications of processor into simulator.

Limitations of Assignment

1. SIMD instructions are unimplemented.
2. Actual number of pipeline stages in ARM processor are more than five.
3. Instructions reordering and similar techniques are not used to reduce number of stalls.

Testcases

```
1  _start:
2  mov w1,#0xffffffff
3  mov w30,#0xffffffff
4  umull x30,w1,w30
5  mov w2,#0xfff
6  cls w3,w2
7  sdiv w30,w1,w2
8  ands x1,x1,#0xf0
9  csneg w4,w1,w30,ne
10 fmov s1,#-5.25
11 fmov s2,#10.0
12 fmov v31.d[1], x2
13 fadd d3,d1,d2
14 fsub v3.4s,v1.4s,v2.4s
15 fmax d4,d3,d31
16 movk x15, #0xfff0, lsl #48
17 bics w0, w1, w2, lsl #3
18 fsub d3,d3,d2
19 movz x15, #32
20 ror w25,w3,w3
21 ccmn w25, w3, #6, eq
22 cls x4,x3
23 mov x10,#0xfffffffffffffff33
24 udiv x11,x10,x2
25 cset w10, eq
26 csneg x1,x2,x3,ne
27 ldrh  W3, [x30, #15]!
28 stp w2, w3, [x4, #16]!
29 ldp w1, w2, [x4], #16
30 cinv x30,x25,eq
31 fmin s31,s1,s31
```

```
Integer Register File
=====================
Register0: 0x0
Register1: 0xfff
Register2: 0x0
Register3: 0x0
Register4: 0x5a
Register5: 0x0
Register6: 0x0
Register7: 0x0
Register8: 0x0
Register9: 0x0
Register10: 0x0
Register11: 0x1001001001000f
Register12: 0x0
Register13: 0x0
Register14: 0x0
Register15: 0x20
Register16: 0x0
Register17: 0x0
Register18: 0x0
Register19: 0x0
Register20: 0x0
Register21: 0x0
Register22: 0x0
Register23: 0x0
Register24: 0x0
Register25: 0x26000
Register26: 0x0
Register27: 0x0
Register28: 0x0
Register29: 0x0
Register30: 0x26000
Register31: 0x0
```

```
Float Register File
====================
Register0: 0x0
Register1: 0xc0a80000
Register2: 0x41200000
Register3: 0x80540000
Register4: 0xc1740000
Register5: 0x0
Register6: 0x0
Register7: 0x0
Register8: 0x0
Register9: 0x0
Register10: 0x0
Register11: 0x0
Register12: 0x0
Register13: 0x0
Register14: 0x0
Register15: 0x0
Register16: 0x0
Register17: 0x0
Register18: 0x0
Register19: 0x0
Register20: 0x0
Register21: 0x0
Register22: 0x0
Register23: 0x0
Register24: 0x0
Register25: 0x0
Register26: 0x0
Register27: 0x0
Register28: 0x0
Register29: 0x0
Register30: 0x0
Register31: 0xc0a80000
```

Figure 1: Testcase-1 with output

```
1  _start:
2  mov x1,#-10
3  mov x2,#5
4  fmov d1,#7.5
5  fmov d2,x1
6  fmov x3,d1
7  movn x4,#5, lsl #32
8  movk x7,#10
9  fmin s31,s1,s2
10 fadd v30.2d,v31.2d,v2.2d
11 subs w4,w1,w2
12 csneg w1,w2,w3,ne
13 ccmn w1, w3, #15, eq
14 bics x1,x2,x3
15 umull x1,w2,w29
16 ldrsb  x3, [x2, #15]!
17 asr w25,w29,w30
18 lsl w25,w29,w30
19 fmax s30,s31,s1
20 fadd d29,d30,d3
21 csinc w0,w4,w2,eq
22 cinv w7,w1,ne
23 lsl w4,w3,#10
24 udiv x1,x2,x3
25 sdiv x1,x2,x3
26 LDP w7, W2, [X3], #16
27 csinc w0, w7, w2, EQ
28 STP w7, W2, [X3, #128]!
29 clz w0, w7
30 cmn w3, #5, LSL #12
31 adc w0, w7, w2
32 fmax s1,s2,s3
33 bics w0, w1, w2, lsr #5
```

```
Integer Register File
=====================
Register0: 0xffffffff
Register1: 0xffffffffffffffff
Register2: 0x0
Register3: 0x10000000000000034
Register4: 0xfffe9000
Register5: 0x0
Register6: 0x0
Register7: 0x0
Register8: 0x0
Register9: 0x0
Register10: 0x0
Register11: 0x0
Register12: 0x0
Register13: 0x0
Register14: 0x0
Register15: 0x0
Register16: 0x0
Register17: 0x0
Register18: 0x0
Register19: 0x0
Register20: 0x0
Register21: 0x0
Register22: 0x0
Register23: 0x0
Register24: 0x0
Register25: 0x0
Register26: 0x0
Register27: 0x0
Register28: 0x0
Register29: 0x0
Register30: 0x0
Register31: 0x0
```

```
Float Register File
===================
Register0: 0x0
Register1: 0xfffffff6
Register2: 0xfffffffffffffff6
Register3: 0x0
Register4: 0x0
Register5: 0x0
Register6: 0x0
Register7: 0x0
Register8: 0x0
Register9: 0x0
Register10: 0x0
Register11: 0x0
Register12: 0x0
Register13: 0x0
Register14: 0x0
Register15: 0x0
Register16: 0x0
Register17: 0x0
Register18: 0x0
Register19: 0x0
Register20: 0x0
Register21: 0x0
Register22: 0x0
Register23: 0x0
Register24: 0x0
Register25: 0x0
Register26: 0x0
Register27: 0x0
Register28: 0x0
Register29: 0xfffffff6
Register30: 0xfffffff6
Register31: 0xfffffff6
```

Figure 2: Testcase-2 with output

Energy vs Voltage Graphs for Testcase-1

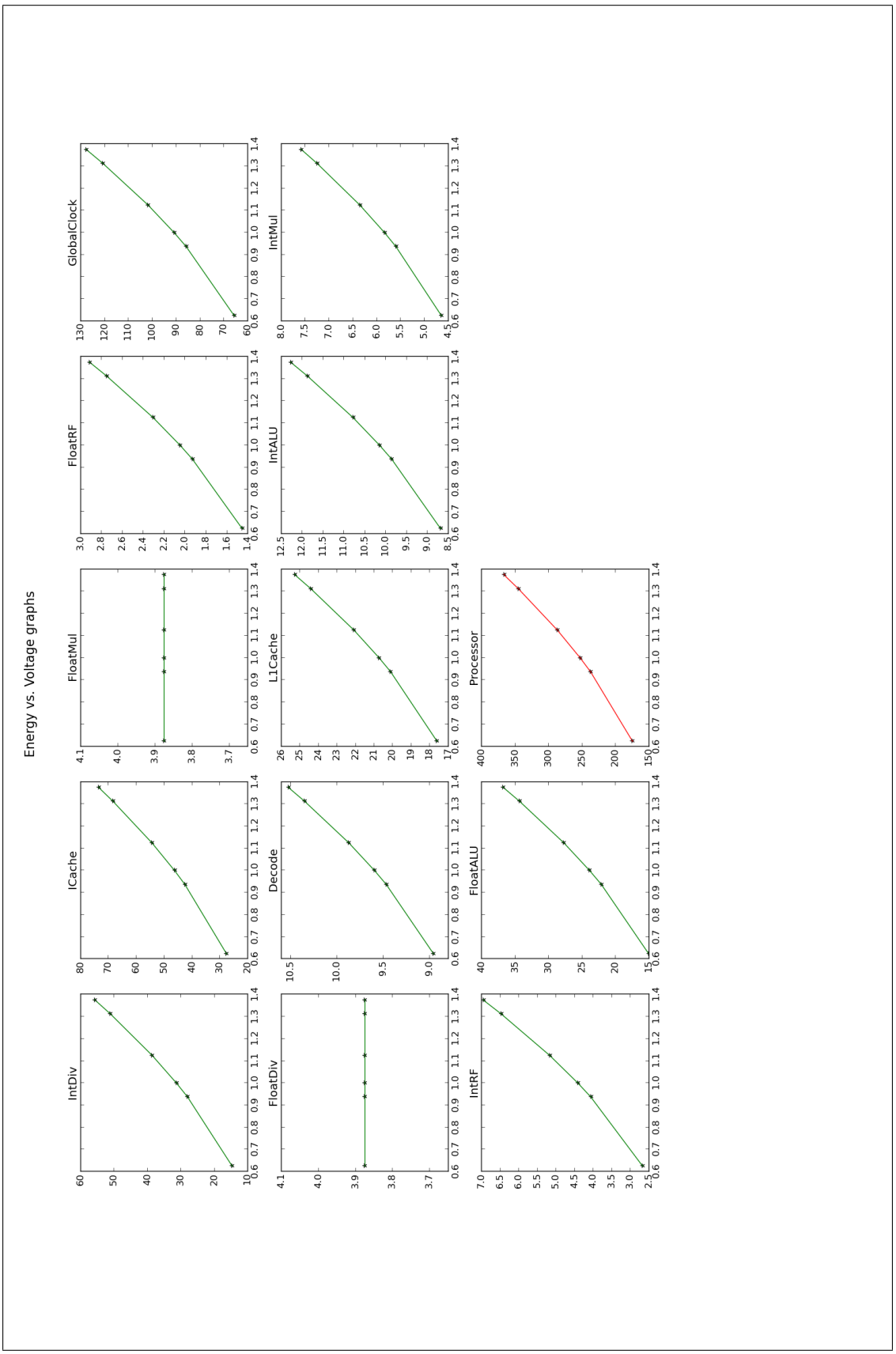

Figure 3: Testcase-1 Energy vs Voltage graphs

EDP vs Voltage Graphs for Testcase-1

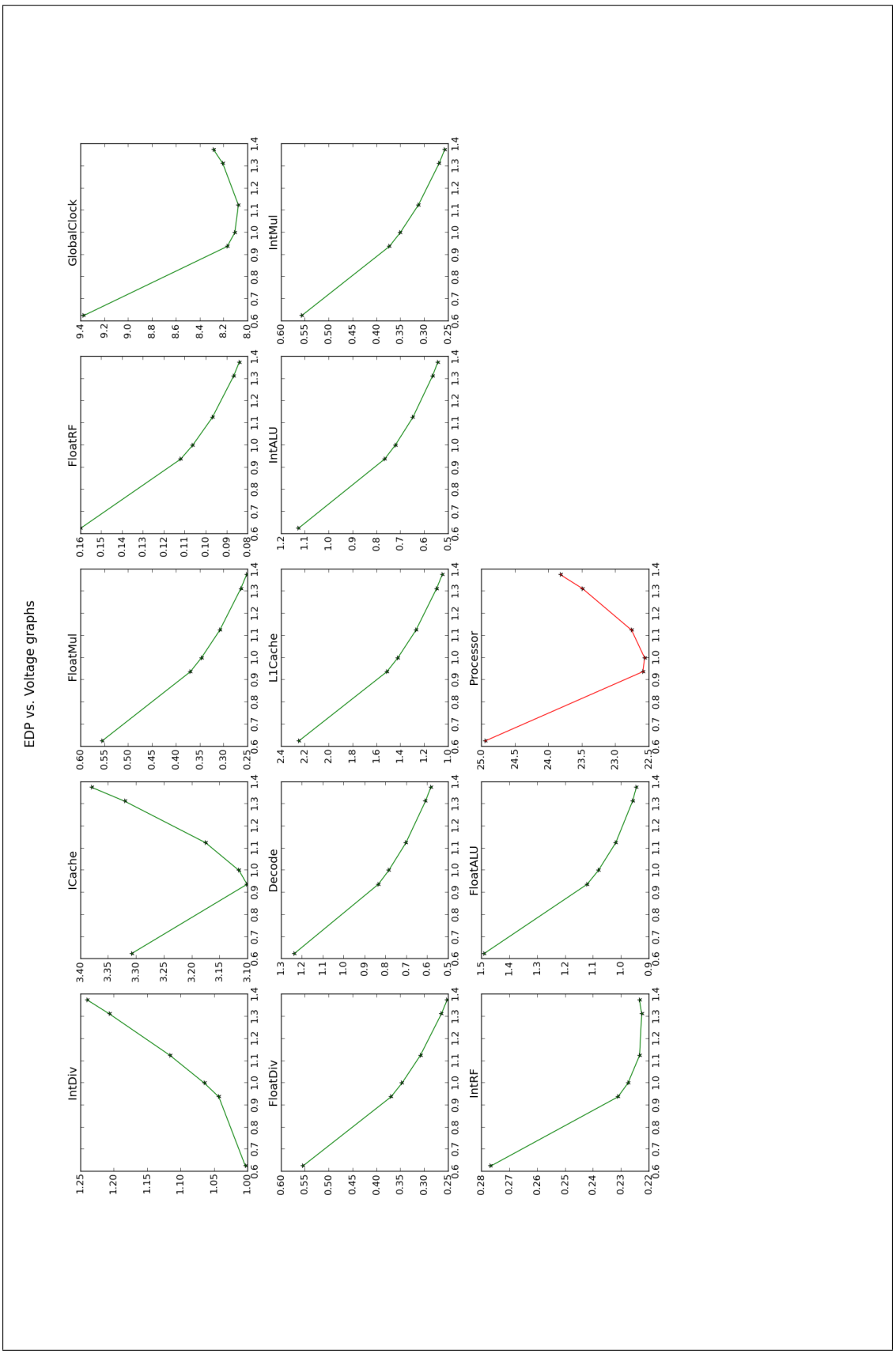

Figure 4: Testcase-1 EDP vs Voltage graphs

Energy vs Voltage Graphs for Testcase-2

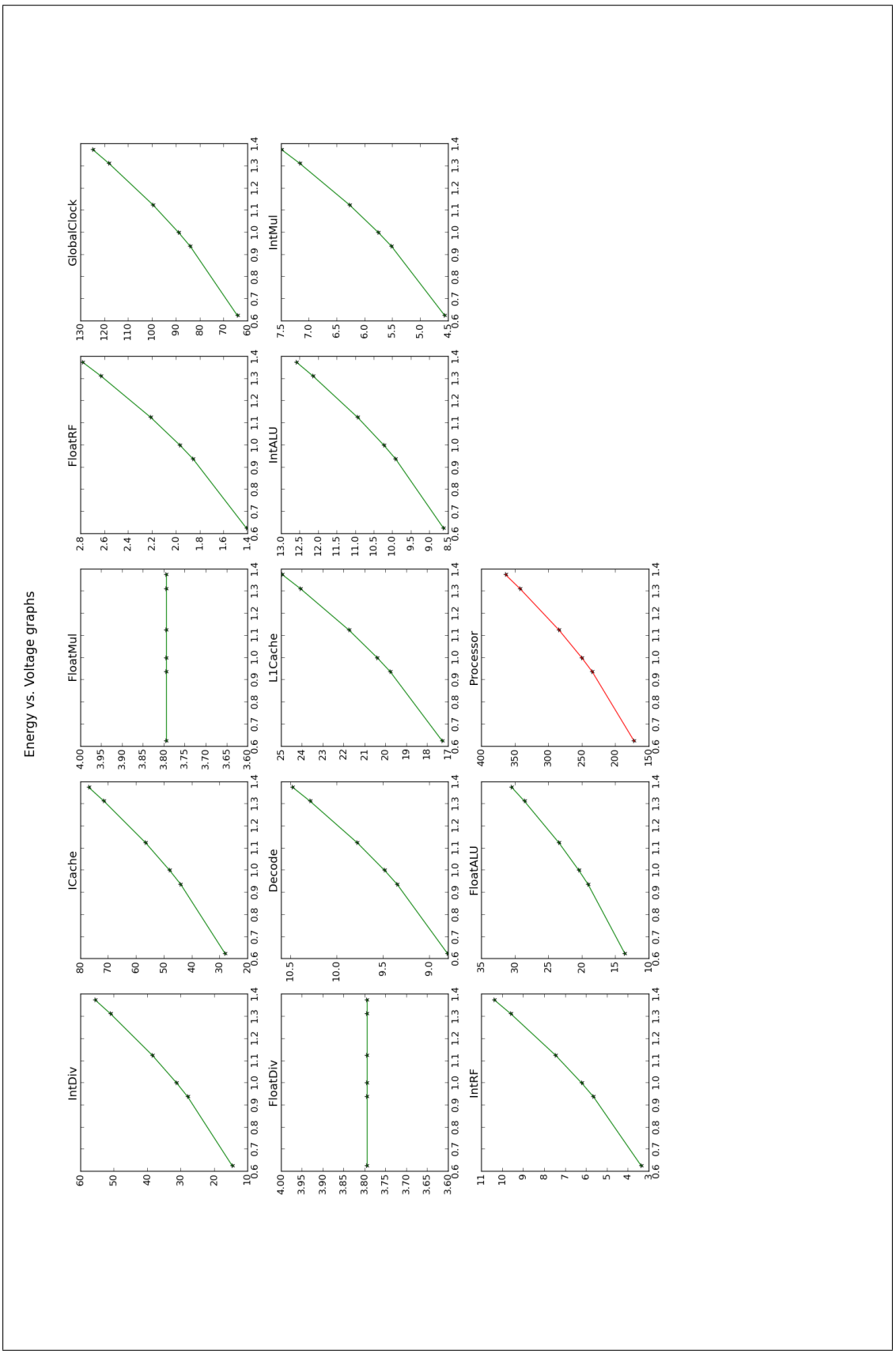

Figure 5: Testcase-2 Energy vs Voltage graphs

EDP vs Voltage Graphs for Testcase-2

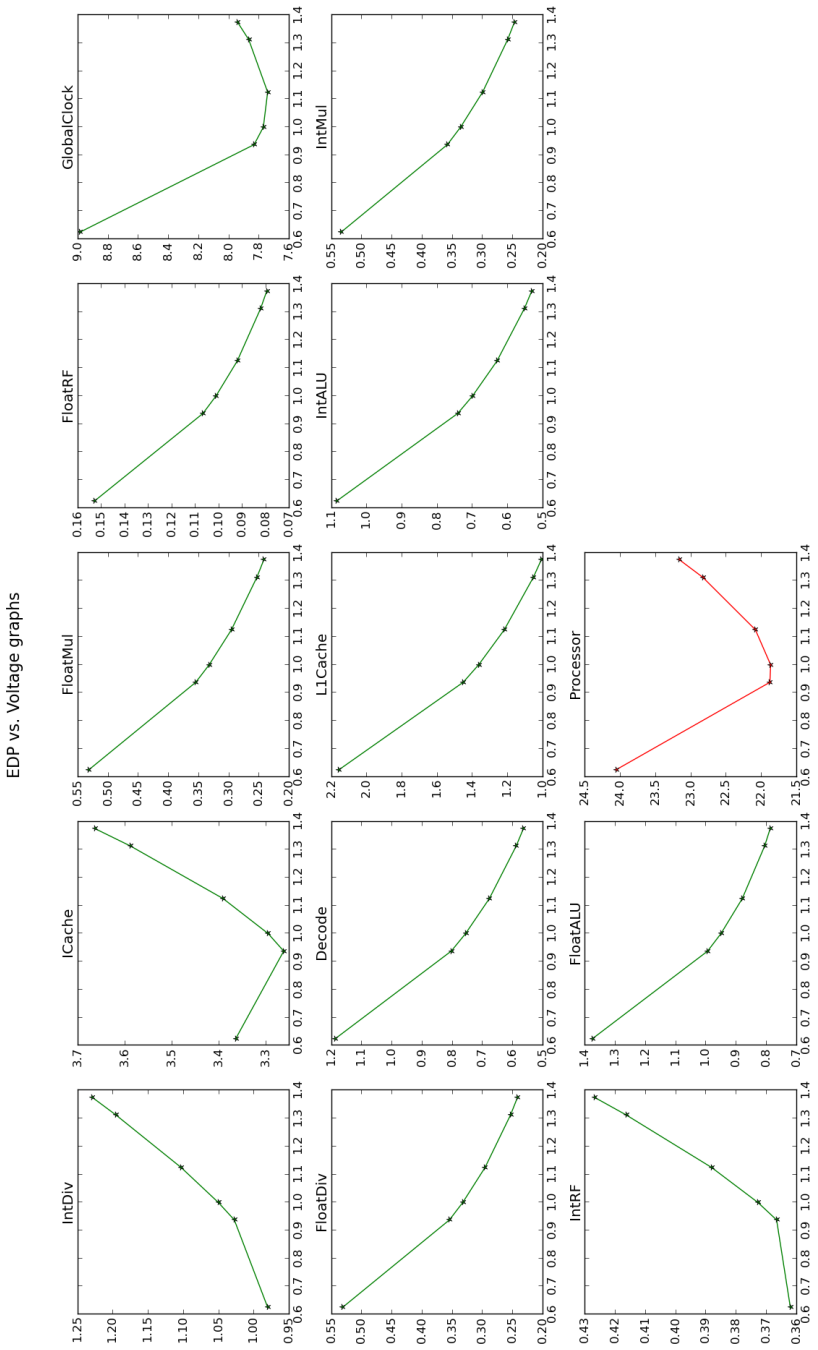

Figure 6: Testcase-2 EDP vs Voltage graphs