FORCE-RISCV/RV32 Feature Document

1. Introduction

Proposal

Enhance the **FORCE-RISCV** *ISG* to support 32-bit RISC-V implementations.

Restated

Modify/adapt as required, the **FORCE-RISCV** architectural layer, source code, simulator API, in order to generate random tests compatible with 32-bit RISC-V designs.

Use cases.

1. Embedded 32-bit RISC-V designs.
2. Educators.

3. Futurewei-designed 32-bit RISC-V microcontrollers, SOCs?

How

Split existing **FORCE-RISCV** instruction set definition files, instruction tree files*, into 64-bit and 32-bit constituents. Use/modify existing **FORCE-RISCV** utilities, register starter files, to generate 32-bit application and system register definitions. Extend existing simulator API, to allow ISA and register width, to be specified.

*Can probably keep the instruction tree in its current form, ie, no need to split out RV32 instructions.

Benefits

Increase the flexibility of **FORCE-RISCV** insofar as supporting RISC-V designs.

FORCE-RISCV enhanced as an open source offering.

2. Requirements

The feature requirements include:

- Create separate sets of 32-bit application and 32-bit system registers.
    - Existing **FORCE-RISCV** provided register generation utilities must be used to generate 32-bit application/system registers.
- Split out the existing instruction definition files by ISA and register width.
    - Existing **FORCE-RISCV** provided instruction definition generation utilities must be used to generate the required instruction definition files.
- Derive 32-bit page table definitions from the *existing* (sv48) page table definition files.
- Enhance the **FORCE-RISCV** simulator APIs, to allow ISA and register width specification.

- Provide top-level **FORCE-RISCV** 32-bit and 64-bit configuration (xml) files.
- Other source code modifications as required, for 32-bit test generation.
- Provide build (Makefile) options to allow the user to compile the **FORCE-RISCV** sources to support a 64-bit RISC-V implementation (the default), or for a 32-bit RISC-V implementation.
- Corrections/additions to existing regression tests, to yield correct results for 64-bit and/or 32-bit **FORCE-RISCV** configurations.

3. Register files

Application register definition files

In the *riscv/arch_data* directory, 32-bit and 64-bit application register definition and floating point/vector register definitions, are to be defined and/or segregated into the following files:

`app_registers_rv32.xml`	32-bit versions of x0 through x31, PC
`app_registers_rv64.xml`	64-bit versions of x0 through x31, PC
`floating_pt_registers.xml`	single-precision, double-precision registers
`vector_registers.xml`	Vector registers

System register definition files

In the *riscv/arch_data* directory, rename and/or create separate sets of 32-bit and 64-bit system register definition files, as follows:

`system_registers_rv32.xml`	system register definitions, register choices, register field choices, 32-bit
`system_register_choices_rv32.xml`	
`register_field_choices_rv32.xml`	
`system_registers_rv64.xml`	system register definitions, register choices, register field choices, 64-bit
`system_register_choices_rv64.xml`	
`register_field_choices_rv64.xml`	

Note that the current system registers file, system_registers.xml, includes definitions for physical/logical RISC-v system register (CSR) definitions, configured for 64-bits. 64-bit RISC-V CSRs defined for 32-bit configurations, will require two 32-bit CSRs to be implemented (example: mstatus – low order 32 bits of mstatus; mstatush – high order 32 bits of mstatus). A (not necessarily comprehensive) list of CSRs that would need to be added to the system_registers_rv32.xml file includes:

> cycleh, mcycleh, timeh, instreth, minstreth, hpmcounter3h, hpmcounter4h, … hpmcounter31h, mhpmcounter3h, mhpmcounter4h, … mhpmcounter31h, htimedeltah, mstatush, pmpcfg1, pmpcfg3, … pmpcfg15

4. Instruction files

Instruction definitions are to be segregated into separate files in the *riscv/arch_data* directory, as follows:

riscv_instructions_int32.xml	RV32I, RV32M	Integer, 32-bit
riscv_instructions_int64.xml	RV64, RV64M	Integer, 64-bit
riscv_instructions_compressed_rv32.xml	RVC	Compressed instructions, 32-bit
riscv_instructions_compressed_rv64.xml		Compressed instructions, 64-bit
riscv_instructions_float.xml	RV32F, RV64F	Single precision floating point
riscv_instructions_double.xml	RV32D, RV64D	Double precision floating point
riscv_instructions_quad.xml	RV32Q, RV64Q	Quad precision floating point
riscv_instructions_vector.xml	RV64V	Vector extensions

5. Paging configuration files

FORCE-RISCV currently supports the RISC-V sv48 (48 bit) Virtual-Memory system. Renaming the existing paging related files, as well as adding new *sv32* paging files, should result in the following paging related file deliverables:

page_tables_sv48.xml	PTE definitions, paging related choices, sv48
paging_choices_sv48.xml	
page_tables_sv32.xml	PTE definitions, paging related choices, sv32
paging_choices_sv32.xml	

6. **Simulator API, simulator config**

Force-RISCV simulator API, Handcar simulator API, enhanced to allow register width, ISA extensions to be specified:

- **FORCE-RISCV** config xml parser, base `Config` class, enhanced to parse, include simulator config string.
- `ApiSimConfig` class enhanced to include simulator config string variable.
- `SimAPI::InitializeIss` method enhanced to support simulator config string
- `SimAPIHandcar::InitializeISS` method enhanced to accept simulator config string.

7. FORCE-RISCV configuration file

Naming convention

Rename the existing FORCE-RISCV config file, copy same to derive a 32-bit version…

`config/riscv.config64`	Existing 64-bit FORCE-RISCV configuration file, renamed.
`config/riscv.config32`	32-bit FORCE-RISCV configuration file.

The **FORCE-RISCV** default configuration file should be specified to be *config/riscv.config64*.

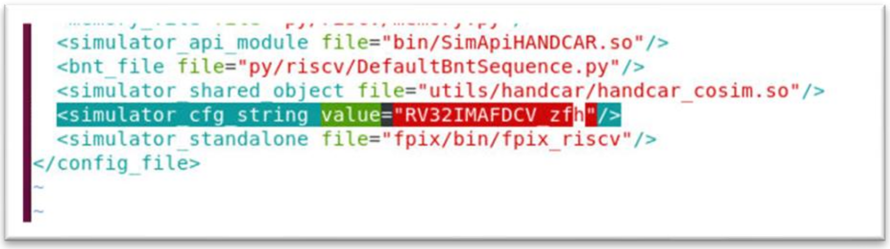
Inclusion of 32-bit or 64-bit files

The **FORCE-RISCV** 64-bit config file to include all 64-bit register, instruction, paging files, as detailed in sections 3, 4, and 5.

The **FORCE-RISCV** 32-bit config file to include all 32-bit register, instruction, paging files, as detailed in sections 3, 4, and 5.

Simulation configuration string

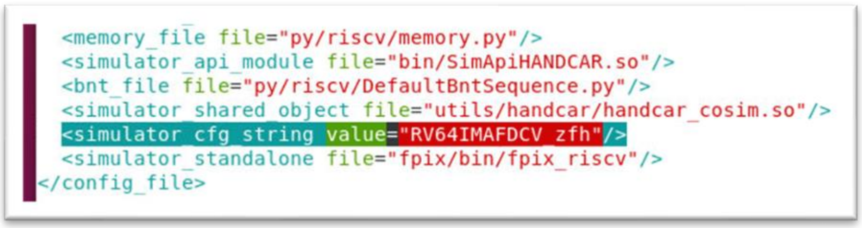
Add an entry to the **FORCE-RISCV** 32-bit configuration file to include an entry used to specify a simulator configuration string:



```
  <simulator_api_module file="bin/SimApiHANDCAR.so"/>
  <bnt_file file="py/riscv/DefaultBntSequence.py"/>
  <simulator_shared_object file="utils/handcar/handcar_cosim.so"/>
  <simulator_cfg_string value="RV32IMAFDCV_zfh"/>
  <simulator_standalone file="fpix/bin/fpix_riscv"/>
</config_file>
~
~
```

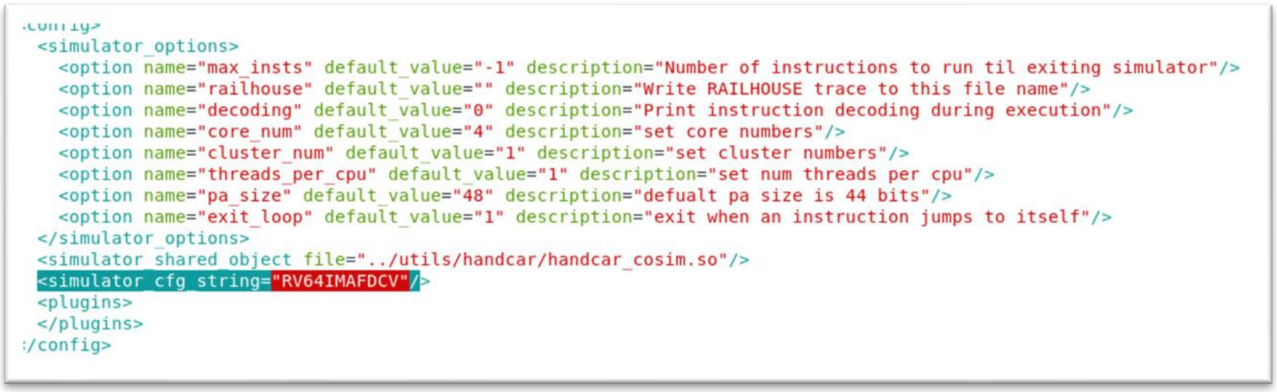
The config string prefix *RV32* indicates 32-bit register widths. The remaining characters identify which RISCV *ISA* extensions should be enabled.

Similarly, the **FORCE-RISCV** 64-bit config file will also need to include the simulator configuration string:



```
    <memory_file file="py/riscv/memory.py"/>
    <simulator_api_module file="bin/SimApiHANDCAR.so"/>
    <bnt_file file="py/riscv/DefaultBntSequence.py"/>
    <simulator_shared_object file="utils/handcar/handcar_cosim.so"/>
    <simulator_cfg_string value="RV64IMAFDCV_zfh"/>
    <simulator_standalone file="fpix/bin/fpix_riscv"/>
</config_file>
```

The **FORCE-RISCV** stand-alone simulator *fpix_riscv*, which also employs the **FORCE-RISCV** simulator APIs, will also need an addition to its RISC-V config file (fpix/config/riscv.config):



```
<config>
  <simulator_options>
    <option name="max_insts" default_value="-1" description="Number of instructions to run til exiting simulator"/>
    <option name="railhouse" default_value="" description="Write RAILHOUSE trace to this file name"/>
    <option name="decoding" default_value="0" description="Print instruction decoding during execution"/>
    <option name="core_num" default_value="4" description="set core numbers"/>
    <option name="cluster_num" default_value="1" description="set cluster numbers"/>
    <option name="threads_per_cpu" default_value="1" description="set num threads per cpu"/>
    <option name="pa_size" default_value="48" description="defualt pa size is 44 bits"/>
    <option name="exit_loop" default_value="1" description="exit when an instruction jumps to itself"/>
  </simulator_options>
  <simulator_shared_object file="../utils/handcar/handcar_cosim.so"/>
  <simulator_cfg_string="RV64IMAFDCV"/>
  <plugins>
  </plugins>
</config>
```

Similarly to **FORCE-RISCV**, separate 32-bit and 64-bit *fpix_riscv* config files will be required. The default *fpix_riscv* config file will be `fpix/config/riscv.config`).

Compile options (C defines) will need to be added to both the **FORCE-RISCV** *Makefiles* and the *fpix_riscv Makefiles* to allow the default config files to be specified at compile time.

8. Source code modifications

FORCE-RISCV Backend

The one known modification thus far to the *backend* code, ie, the *C++* code, that will be required (other than the changes related to the config file discussed in section 5), is in the GenSequenceAgentRISCV::*LoadGPRSequence* method, which issues 64-bit move immediate instructions to load a GPR.

This method will need to be enhanced, to issue 32-bit move immediate instructions, when either the value to load is only 32 bits, or the GPR to load to is 32 bits.

FORCE-RISCV Frontend

Front end, *python* code located in *py/riscv*, that will require modifications to operate correct in a 32-bit environment includes:

py/riscv/Utils.py

Various **FORCE-RISCV** python based utilities, and test templates make use of the py/*Utils/LoadGPR64,LoopControl* classes.

Enhance the *LoadGPR64::load* method to detect the GPR register width, to seamlessly load values into either 64-bit (the default) or 32-bit GPRs.

py/riscv/ModifierUtils.py

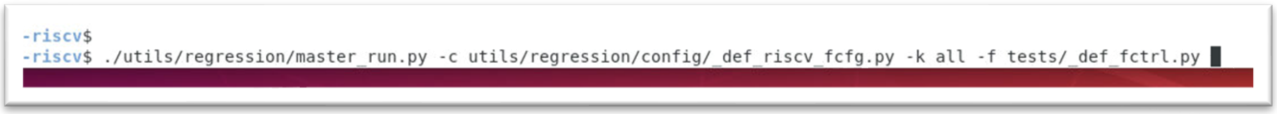
Define page fault types, levels based on current paging mode (sv32 vs sv39 vs sv48).

fpix_riscv stand-alone simulator

Code additions to *fpix_riscv* will be required to process the config file *simulator_cfg_string* variable, and pass its value to the `SimAPI::InitializeIss` method.

9. Regression tests

The *top level* **FORCE-RISCV** regression test suite is run using the *master_run.py* utility:



```
-riscv$
-riscv$ ./utils/regression/master_run.py -c utils/regression/config/_def_riscv_fcfg.py -k all -f tests/_def_fctrl.py
```

Segregate or modify tests, as required, to insure that the top level regression can be run on the bulk (or all?) of the current regression test suites, for a **FORCE-RISCV** 64-bit configurations or for a 32-bit configuration.