FORCE-RISCV/R32 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.

1. **Requirements**

The feature requirements include:

* Create separate sets of 32-bit application and 64-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.

1. **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, vector register definitions |

**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

1. **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\_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 |

1. **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 |

1. **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.

1. **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*.

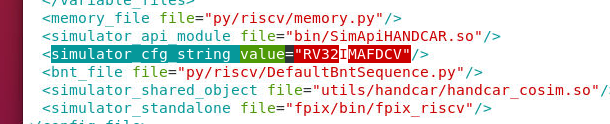
**Inclusion of 32-bit or 64-bit files**

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.

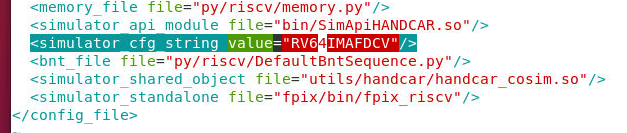
**Simulation configuration string**

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



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:



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):



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.

1. **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.

1. **Regression tests**

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



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.