Task1 - RISC-V Toolchain Setup Tasks & Uniqueness Test

Ubuntu (native or in VirtualBox on Windows) • Spike • Proxy Kernel • Icarus Verilog

Before you start

Linux (native): Ubuntu 22.04 LTS (64-bit) is recommended.

Windows: Do not change your existing OS. Install Ubuntu 22.04 LTS in Oracle VirtualBox (2 vCPUs, 4–8 GB RAM, 30 GB disk). Then run the tasks inside that Ubuntu VM.

Tip: Run `sudo apt-get update` once before Task 1.

Task 1 — Install base developer tools

Why: These are common build prerequisites (compilers, linkers, autotools) and libraries required by the RISC-V simulator, proxy kernel, and other tooling. GTKWaves is included for waveform viewing in digital design flows.

```
sudo apt-get install -y git vim autoconf automake autotools-dev curl \
  libmpc-dev libmpfr-dev libgmp-dev gawk build-essential bison flex \
  texinfo gperf libtool patchutils bc zlib1g-dev libexpat1-dev gtkwave
```

Task 2 — Create a clean workspace and capture your home path

Why: Keeps everything contained in ~/riscv_toolchain, making it easy to remove, update, or archive. Storing $pwd (your home) ensures consistent paths for later steps.

```
cd
pwd=$PWD
mkdir -p riscv_toolchain
cd riscv_toolchain
```

Task 3 — Get a prebuilt RISC-V GCC toolchain

Why: Provides riscv64-unknown-elf-gcc (newlib) to compile bare-metal/user-space RISC-V programs. Using a prebuilt toolchain avoids a long source build.

```
wget "https://static.dev.sifive.com/dev-tools/riscv64-unknown-elf-gcc-
8.3.0-2019.08.0-x86_64-linux-ubuntu14.tar.gz"
tar -xvzf riscv64-unknown-elf-gcc-8.3.0-2019.08.0-x86_64-linux-
ubuntu14.tar.gz
```

Task 4 — Add toolchain to your PATH (current shell + persistent)

Why: Allows riscv64-unknown-elf-gcc and related binaries to be available without typing full paths, both now and after you reopen the terminal.

Command (current shell):

```
export PATH=$pwd/riscv_toolchain/riscv64-unknown-elf-gcc-8.3.0-
2019.08.0-x86_64-linux-ubuntu14/bin:$PATH
```

Command (persistent for new terminals):

```
echo 'export PATH=$HOME/riscv_toolchain/riscv64-unknown-elf-gcc-8.3.0-
2019.08.0-x86_64-linux-ubuntu14/bin:$PATH' >> ~/.bashrc
source ~/.bashrc
```

Task 5 — Install Device Tree Compiler (DTC)

Why: Some RISC-V components expect DTC to be present; it's a common dependency in SoC/simulator flows.

```
sudo apt-get install -y device-tree-compiler
```

Task 6 — Build and install Spike (RISC-V ISA simulator)

Why: Spike is the reference ISA simulator. You'll run your compiled RISC-V ELF on Spike to validate correctness. Installing into the same prefix keeps everything together.

```
cd $pwd/riscv_toolchain
git clone https://github.com/riscv/riscv-isa-sim.git
cd riscv-isa-sim
mkdir -p build && cd build
../configure --prefix=$pwd/riscv_toolchain/riscv64-unknown-elf-gcc-
8.3.0-2019.08.0-x86_64-linux-ubuntu14
make -j$(nproc)
sudo make install
```

Task 7 — Build and install the RISC-V Proxy Kernel (riscv-pk)

Why: pk provides a minimal runtime so you can run newlib ELF binaries under Spike with 'spike pk ./your_prog'. It bridges your compiled program to the simulator.

```
cd $pwd/riscv_toolchain
git clone https://github.com/riscv/riscv-pk.git
cd riscv-pk
mkdir -p build && cd build
../configure --prefix=$pwd/riscv_toolchain/riscv64-unknown-elf-gcc-
8.3.0-2019.08.0-x86_64-linux-ubuntu14 --host=riscv64-unknown-elf
make -j$(nproc)
sudo make install
```

Task 8 — Ensure the cross bin directory is in PATH

Why: Some installs place pk and related utilities into a nested riscv64-unknown-elf/bin. Adding this ensures pk is found by 'which pk'.

Command (current shell):

```
export PATH=$pwd/riscv_toolchain/riscv64-unknown-elf-gcc-8.3.0-
2019.08.0-x86_64-linux-ubuntu14/riscv64-unknown-elf/bin:$PATH
```

Command (persistent):

```
echo 'export PATH=$HOME/riscv_toolchain/riscv64-unknown-elf-gcc-8.3.0-
2019.08.0-x86_64-linux-ubuntu14/riscv64-unknown-elf/bin:$PATH' >>
~/.bashrc
source ~/.bashrc
```

Task 9 — (Optional) Install Icarus Verilog

Why: For digital design education, you'll often verify RTL with Icarus Verilog and view waveforms with GTKWaves. This is not required for the C uniqueness test but is part of a complete RISC-V learning toolchain.

```
cd $pwd/riscv_toolchain
git clone https://github.com/steveicarus/iverilog.git
cd iverilog
git checkout --track -b v10-branch origin/v10-branch
git pull
chmod +x autoconf.sh
./autoconf.sh
./configure
make -j$(nproc)
sudo make install
```

Task 10 — Quick sanity checks

Why: Confirms the toolchain and simulator are visible and runnable from your shell.

```
which riscv64-unknown-elf-gcc
riscv64-unknown-elf-gcc -v

which spike
spike --version || spike -h

which pk
```

Final Deliverable: A Unique C Test (Username & Machine Dependent)

We need every participant's output to be unique for evaluation. We'll compile the program while injecting your username and hostname as compile-time constants. The program computes a 64-bit FNV-1a hash of 'USERNAME@HOSTNAME' and prints it.

1) Create unique_test.c

```
#include <stdint.h>
#include <stdio.h>

#ifndef USERNAME
#define USERNAME "unknown_user"
#endif
#ifndef HOSTNAME
#define HOSTNAME "unknown_host"
#endif

// 64-bit FNV-1a
static uint64_t fnv1a64(const char *s) {
    const uint64_t FNV_OFFSET = 14695981039346656037ULL;
    const uint64_t FNV_PRIME  = 1099511628211ULL;
    uint64_t h = FNV_OFFSET;
    for (const unsigned char *p = (const unsigned char*)s; *p; ++p) {
        h ^= (uint64_t)(*p);
        h *= FNV_PRIME;
    }
    return h;
}

int main(void) {
    const char *user = USERNAME;
    const char *host = HOSTNAME;

    char buf[256];
    int n = snprintf(buf, sizeof(buf), "%s@%s", user, host);
    if (n <= 0) return 1;

    uint64_t uid = fnv1a64(buf);

    printf("RISC-V Uniqueness Check\n");
    printf("User: %s\n", user);
    printf("Host: %s\n", host);
    printf("UniqueID: 0x%016llx\n", (unsigned long long)uid);

#ifdef __VERSION__
    unsigned long long vlen = (unsigned long long)sizeof(__VERSION__) -
1;
    printf("GCC_VLEN: %llu\n", vlen);
#endif
```

```
    return 0;
}
```

2) Compile with injected identity and RISC-V flags

```
riscv64-unknown-elf-gcc -O2 -Wall -march=rv64imac -mabi=lp64 \
  -DUSERNAME="$(id -un)" -DHOSTNAME="$(hostname -s)" \
  unique_test.c -o unique_test
```

3) Run on Spike with the proxy kernel

```
spike pk ./unique_test
```

Expected output format:

```
RISC-V Uniqueness Check
User: <your-username>
Host: <your-hostname>
UniqueID: 0x<64-bit-hex>
GCC_VLEN: <number>
```

What to submit for evaluation

1) Source file unique_test.c

2) The exact compile command you used

3) The program output from spike pk ./unique_test

Uniqueness rule: Your UniqueID must differ from other participants because it's derived from your user/machine identity at compile time. Changing user or hostname will change the output. Evaluation will be based on this uniqueness.

Notes & Troubleshooting

• If spike or pk aren't found, re-open the terminal or re-run:

```
source ~/.bashrc
hash -r
```

• If configure fails, ensure Task 1 completed without errors and try:

```
sudo apt-get update && sudo apt-get install -y build-essential autoconf
automake libtool
```

• If you prefer a from-source GCC/newlib build instead of the prebuilt tarball, that's allowed—but it will take significantly longer. Keep the same Spike/pk steps and the same uniqueness test.