

The Sail Programming Language

A Sail Cookbook

William C. McSpadden, Martin Berger

Table of Contents

List of programming examples (in increasing complexity)	4
1. Introduction	5
2. How to contribute (Bill)	6
2.1. pull requests (Bill)	6
2.2. RISC-V working groups (Bill)	6
2.3. style and contribution guide	6
2.4. brevity	6
2.5. short executable	6
2.6. standalone	6
2.7. maintainership (when something breaks)	6
2.8. Syntax highlighting for Sail	6
3. Sail installation	7
3.1. Docker	7
3.2. Ubuntu (Bill Mc.)	7
3.3. MacOS (Martin)	7
3.4. Windows	7
3.5. Windows: Cygwin (Bill Mc., low priority)	7
3.6. Other?	7
4. Basic description	8
4.1. what sail is	8
4.2. what sail is not	8
4.3. version management and what to expect	8
5. “Hello, World” example program (Bill)	9
5.1. "Hello, World" and the Sail interactive interpreter	13
6. Data types	14
6.1. effect annotations	14
6.2. Integers	14
6.3. type variables	14
6.4. Bits	14
6.5. Strings	14
6.6. Lists	14
6.7. Structs	14
6.8. mappings	14
6.9. Liquid data types (Martin)	14
7. Execution	15
7.1. Functions	15
7.2. The ``my_replicate_bits()`` function from the Sail manual (Bill)	15
7.3. Control flow	18

7.4. Iteration	18
7.5. matches	18
8. Other stuff	19
8.1. <i>FILE</i> , <i>LINE</i> , <i>LOC</i>	19
9. Description prelude.sail	20
9.1. description of print, sext, equility etc. standard template stuff	20
9.2. the C interface	20
10. CPU example	21
11. Formal tools that analyze Sail source code	22
12. FAQs (Frequently Asked Questions)	23
13. Frequently Asked Questions about the Sail RISC-V Golden Model.	24
13.1. Q: Is there support for multi-HART or multi-Core simulation?	24
13.2. Q: What are .ml files? What are their purpose?	24
14. Q:	25
14.1. Q: What is the meaning of life, the universe and everything?	25
14.2. Q: What does the answer to "What is the meaning of life, the universe and everything" mean?	25

List of programming examples (in increasing complexity)

The main purpose of this document, is to give the user a quick reference to Sail coding examples. The following is a list of all the programming examples found in this document.

[“Hello, World” example program \(Bill\)](#)

[The ``my_replicate_bits\(\)`` function from the Sail manual \(Bill\)](#)

Chapter 1. Introduction

Sail is a programming language that was developed for the purpose of clearly, concisely and completely describing a computer's Instruction Set Architecture (ISA). This includes... - specifying the opcodes/instructions and their behaviours - specifying the general purpose registers - specifying the control space registers

Sail was the language chosen by RISC-V International to formally specify the RISC-V open source ISA. This document, while not RISC-V specific, is especially targeted for engineers who are working on specifying the RISC-V ISA.

This cookbook is intended to supply the beginning Sail programmer with some simple, well-commented, bite-size program fragments that can be compiled and run.

github is used to host the development of Sail. You can find the repository at the following URL:

<https://github.com/rem-s-project/sail>

Currently, the work on this cookbook can be found on a branch in the above repo. This branch is:

https://github.com/billmcspadden-riscv/sail/tree/cookbook_br

So this is the place you should probably clone. (Eventually, this branch will be merged to the release branch.)

Other documentation:

There is another useful Sail document that you should know about. It is "The Sail instruction-set semantics specification language" by Armstrong, et. al. It can be found at:

https://github.com/billmcspadden-riscv/sail/blob/cookbook_br/manual.pdf

While useful, the document does not contain a useful set of programming examples. That is the purpose of **this** document.

Chapter 2. How to contribute (Bill)

2.1. pull requests (Bill)

2.2. RISC-V working groups (Bill)

2.3. style and contribution guide

2.4. brevity

2.5. short executable

2.6. standalone

2.7. maintainership (when something breaks)

2.8. Syntax highlighting for Sail

Syntax highlighting for several editors (emacs, vim, Visual Studio, etc) can be found at:

<https://github.com/rem-s-project/sail/tree/sail2/editors>

It is beyond the scope of this document to describe how to use the syntax highlighting for the various editors.

Chapter 3. Sail installation

3.1. Docker

TBD

3.2. Ubuntu (Bill Mc.)

TBD

3.3. MacOS (Martin)

TBD

3.4. Windows

Support of a native command line interface is not planned. If you want to run Sail under Windows, plan on running it under Cygwin.

3.5. Windows: Cygwin (Bill Mc., low priority)

If there is a demand, a port to Cygwin will be attempted.

3.6. Other?

Are there other OS platforms that should be supported? Other Linux distis? Or will Docker support?

Chapter 4. Basic description

4.1. what sail is

- sequential model only
- non-parallel

4.2. what sail is not

- not a RTL language, etc
- Sail does not support any parallelism. No threads. No event sequences. No clocking.

4.3. version management and what to expect

Chapter 5. “Hello, World” example program (Bill)

All example programs associated with this cookbook, can be found in `<sail_git_root>/cookbook/functional_code_snippets/`

The purpose of this simple program is to show some of the basics of Sail and to ensure that you have the Sail compiler (and the other required tools) installed in your environment.

It is assumed that you have built the sail compiler in the local area. The Makefiles in the coding examples depend on this.

The following code snippet comes from:

https://github.com/billmcspadden-riscv/sail/tree/cookbook_br/cookbook/functional_code_snippets/hello_world

hello_world.sail:

```
// Two types of comments...
// This type and ...

/*
...block comments
*/

// Whitespace is NOT significant. Yay!

default Order dec           // Required. Defines whether bit vectors are
increasing (inc)            // (MSB is index 0; AKA big-endian) or decreasing
(dec)                       // (LSB is index 0; AKA little-endian)
// default Order inc

// The $include directive is used to pull in other Sail code.
// It functions similarly, but not exactly the same, as the
// C preprocessor directive.

// Sail is a very small language. In order to get a set
// of useful functionality (eg - print to stdout), a set
// of functions and datatypes are defined in the file
// "prelude.sail"
#include <prelude.sail>

// =====
// Function signatures (same idea as C's function prototype)
// =====
```

```

val "print" : string -> unit

val main : unit -> unit

// =====
// The entry point into the program starts at the function, main.
// =====
function main() =
{
    print("hello, world!\n") ;
    print("hello, another world!\n") ;
}

```

So... that's the code we want to compile. But how do we compile it? Remember, we want to use the sail compiler that was built in this sandbox. We use a *make* methodology for building. The first Makefile (in the same directory as the example code example) is very simple. It includes a generic Makefile (../Makefile.generic) that is used for building most of the program examples.

[Note] If you want to create and contribute your own example program and you need to deviate from our make methodology, you would do that in your own test directory by writing your own Makefile.

The basic flow is:

*.sail --(Sail)-> out.c, *.c --(gcc)-> executable

Makefile:

```

# vim: set tabstop=4 shiftwidth=4 noexpandtab
# =====
# Filename:      Makefile
#
# Description:   Makefile for building example code
#
# Author(s):     Bill McSpadden (bill@riscv.org)
#
# Revision:      See revision control log
#
# =====

#=====
# Includes
#=====

include ../Makefile.generic

```

Makefile.generic is the Makefile that does the work for compilation. It depends on a local compilation of sail. See the [Installation](#sail-installation) section to understand how to install in the tools for your platform.

Makefile:

```
# vim: set tabstop=4 shiftwidth=4 noexpandtab
# =====
# Filename:      Makefile
#
# Description:   Makefile for building Sail example code fragments
#
#               NOTE: in order to render this file in an asciidoc
#               for the Sail cookbook, keep the line length less
#               than 86 characters, the width of the block comment line
#               of this section
#
# Author(s):     Bill McSpadden (bill@riscv.org)
#
# Revision:      See revision control log
#
# =====

#=====
# Includes
#=====

#=====
# Make variables
#=====

# The sail compiler expects that SAIL_DIR is set in the environment.
# The sh env var, SAIL_DIR, is set and exported using the make
# variable, SAIL_DIR. I hope this is not too confusing.
SAIL_DIR      := ../../..
SAIL_LIB      := ${SAIL_DIR}/lib/sail
SAIL          := ${SAIL_DIR}/sail
SAIL_OUTFILE  := out
SAIL_FLAGS    := -c -o ${SAIL_OUTFILE}

SAIL_SRC      := $(wildcard *.sail)

CC            := gcc
CCFLAGS       := -lgmp -lz -I ${SAIL_DIR}/lib/

# out.c is the file that sail generates as output from the
# sail compilation process. It will be compiled with
# other C code to generate an executable
# ${SAIL_DIR}/lib/*.c is a set of C code used for interaction
# with the programming environment. It also provides
# functionality that cannot be natively supported by sail.
#
C_SRC         := out.c ${SAIL_DIR}/lib/*.c
```

```

TARGET          := out

#=====
# Targets and Rules
#=====

all: run

build: out

install:

run: out
    ./out

out: out.c
    gcc ${C_SRC} ${CCFLAGS} -o $@

#   gcc out.c ${SAIL_DIR}/lib/*.c -lgmp -lz -I ${SAIL_DIR}/lib -o $@

# In the following rule, the environment variable, SAIL_DIR, must be
# set in order for the sail compilation step to work correctly.
out.c: ${SAIL_SRC}
    SAIL_DIR=${SAIL_DIR} ; export SAIL_DIR ; \
    ${SAIL} ${SAIL_FLAGS} ${SAIL_SRC}

# clean: cleans only local artifacts
clean:
    rm -f out out.c out.ml

# Cleans local artifacts and the install location
clean_all:

```

What does the compilation process look like? Under Ubuntu Linux, this is the output you can expect for compiling and running the "hello world" example program.

```

ubuntu-VirtualBox 227> make
SAIL_DIR=../../.. ; export SAIL_DIR ; \
../../..sail -c -o out hello_world.sail
gcc out.c ../../..lib/*.c -lgmp -lz -I ../../..lib/ -o out
./out
hello, world!
hello, another world!
ubuntu-VirtualBox 228>

```

Now that we've examined the Makefiles, we will make little mention of them in the rest of this document (except for the example where we discuss the C foreign function interface where we will show how Sail can call C functions).

5.1. "Hello, World" and the Sail interactive interpreter

TBD

Chapter 6. Data types

6.1. effect annotations

6.2. Integers

- Int
- int
- Multi-precision

6.3. type variables

What does " 'n " mean?

6.4. Bits

6.5. Strings

6.6. Lists

6.7. Structs

6.8. mappings

6.9. Liquid data types (Martin)

Chapter 7. Execution

7.1. Functions

7.2. The ``my_replicate_bits()`` function from the Sail manual (Bill)

First, let's look at the code that is described in the Sail manual for the function, `my_replicate_bits()`.

Note: The following code actually comes from the file `doc/examples/my_replicate_bits.sail`. It is a little bit different than what is shown in the manual for reasons that will be covered here.

```
//default Order dec          // billmc

#include <prelude.sail>

// billmc
#include "my_replicate_bits_function_signatures.sail"

infixl 7 <<
infixl 7 >>

val operator << = "shiftrl" : forall 'm. (bits('m), int) -> bits('m)
val "shiftrl" : forall 'm. (bits('m), int) -> bits('m)

val operator >> = {
  ocaml: "shiftr_ocaml",
  c: "shiftr_c",
  lem: "shiftr_lem",
  _: "shiftr"
} : forall 'm. (bits('m), int) -> bits('m)

//val "or_vec" : forall 'n. (bits('n), bits('n)) -> bits('n)
val or_vec = {c: "or_bits" } : forall 'n. (bits('n), bits('n)) -> bits('n)      //
billmc

val zero_extend = "zero_extend" : forall 'n 'm, 'm >= 'n. (bits('n), atom('m)) ->
bits('m)

overload operator | = {or_vec}

//val my_replicate_bits : forall 'n 'm, 'm >= 1 & 'n >= 1. (int('n), bits('m)) ->
bits('n * 'm)      // billmc

val zeros = "zeros" : forall 'n. atom('n) -> bits('n)

function my_replicate_bits(n, xs) = {
```



```

ys = zeros(n * length(xs));
foreach (i from 1 to n) {
  ys = ys << length(xs);
  ys = ys | zero_extend(xs, length(ys))
};
ys
}

val my_replicate_bits_2 : forall 'n 'm, 'm >= 1 & 'n >= 1. (int('n), bits('m)) ->
bits('n * 'm)

function my_replicate_bits_2(n, xs) = {
  ys = zeros('n * 'm);
  foreach (i from 1 to n) {
    ys = (ys << 'm) | zero_extend(xs, 'n * 'm)
  };
  ys
}

val cast_extz : forall 'n 'm, 'm >= 'n. (implicit('m), bits('n)) -> bits('m)

function extz(m, xs) = zero_extend(xs, m)

val my_replicate_bits_3 : forall 'n 'm, 'm >= 1 & 'n >= 1. (int('n), bits('m)) ->
bits('n * 'm)

function my_replicate_bits_3(n, xs) = {
  ys = zeros('n * 'm);
  foreach (i from 1 to n) ys = ys << 'm | xs;
  ys
}

```

You will see in this code, that there is no *main* function, and as such, will not compile into a C Sail model. You will get the following error message:

TODO: get the error message.

In order to get this to compile into a C Sail model, you will need to provide a main function. The following code shows the implementation of a ``main()`` function that calls `my_replicate_bits()`.

```

// vim: set tabstop=4 shiftwidth=4 expandtab
// =====
// Filename:    main.sail
//
// Description: Example sail file
//
// Author(s):   Bill McSpadden (bill@riscv.sail)
//
// Revision:    See revision control log
// =====

```

```

default Order dec
$include <prelude.sail>

val "print" : string -> unit

//val my_replicate_bits : forall 'n 'm, 'm >= 1 & 'n >= 1. (int('n), bits('m)) ->
bits('n * 'm)
$include "my_replicate_bits_function_signatures.sail"

val main : unit -> unit
function main() =
{
  v1 : bits(8) = 0x55;
  v2 : bits(32) = 0x00000000;

  // Sail has a powerful type-checking system, but understanding it
  // is best learned by examining some examples.

  //   num : int = 4;           // CE
  //   let num : int(4) = 4;     // Works
  //   let num : int(4) = 5;     // CE
  //   let num : int(5) = 5;
  //   let num : int(4) = 3;     // CE
  let num : int(4) = 3 + 1;

  print("calling my_replicate_bits() .....\\n");

  // The compiler needs to evaluate
  //   v2 = my_replicate_bits (num, v1);
  v3 : bits(32) = my_replicate_bits (num, v1);
  //   v3 : bits(32) = my_replicate_bits (4, v1);

  print_bits("replicated bits: ", v3);

  print("returned from my_replicate_bits() .....\\n");
}

```

Because both the files, `my_replicate_bits.sail` and `main.sail`, need to have the function signatures in order to compile (and we want them to be consistent), the function signatures have been put into a separate file that is include by both. Here is the function signature file, `my_replicate_bits_function_signatures.sail`:

```

// vim: set tabstop=4 shiftwidth=4 expandtab
// =====
// Filename:   my_replicate_bvits_function_signatures.sail
//
// Description:
//

```

```
// Author(s):   Bill McSpadden (bill@riscv.org)
//
// Revision:     See revision control log
// =====

#include <prelude.sail>

val "print" : string -> unit

val my_replicate_bits : forall 'n 'm, 'm >= 1 & 'n >= 1. (int('n), bits('m)) ->
bits('n * 'm)

val main : unit -> unit
```

7.3. Control flow

7.4. Iteration

- for
- while
- lambda function

7.5. matches

Chapter 8. Other stuff

8.1. *FILE* , *LINE* , *LOC*

Chapter 9. Description prelude.sail

prelude.sail contains the function signatures and implemenmtations of many support functions.

9.1. description of print, sext, equility etc. standard template stuff

9.2. the C interface

Chapter 10. CPU example

- From nand2tetris

Chapter 11. Formal tools that analyze Sail source code

coverage

Chapter 12. FAQs (Frequently Asked Questions)

Following are a set of FAQs that were generated via set of questions to the Sail developers.

Chapter 13. Frequently Asked Questions about the Sail RISC-V Golden Model

- [Q: Is there support for multi-HART or multi-Core simulation?](#is-there-support-for-multi-hart-multi-core-simulation)
- [Q: What is the meaning of life, the universe and everything?](#q-what-is-the-meaning-of-life-the-universe-and-everything)
- [Q: What does the answer to "What is the meaning of life, the universe and everything" mean?](#q-what-does-the-answer-to-"what-is-the-meaning-of-life-the-universe-and-everything"-mean)

13.1. Q: Is there support for multi-HART or multi-Core simulation?

A: There is no inherent support for multi-HART or multi-Core within the existing RISC-V Sail model. There are future plans for adding this kind of simulation. It is needed in order to simulate (in a meaningful way) the atomic memory operations and to evaluate memory consistency and coherency.

The model isn't directly about testing. Testing is a separate activity. The point of the model is to be as clear as possible. and we should keep testing and the model separate.

13.2. Q: What are .ml files? What are their purpose?

A: These are OCaml files. They are to the ocaml emulator what the .c files are to the c emulator. I question the need for an OCaml emulator ,see also <https://github.com/riscv/sail-riscv/issues/138>

Chapter 14. Q:

14.1. Q: What is the meaning of life, the universe and everything?

A: 42

14.2. Q: What does the answer to "What is the meaning of life, the universe and everything" mean?

A: One must construct an experimental, organic computer to compute the meaning. Project *Earth* is one such computer. Timeframe for an expected answer is... soon.