The Sail Programming Language

A Sail Cookbook

William C. McSpadden, Martin Berger

Table of Contents

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.

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 docuement, 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/rems-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/rems-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 preproessor directrive.

// 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 methodolgy, 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.....
#
# 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).

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. Control flow

7.3. Iteration

- for
- while
- lamba function

7.4. 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