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)

We are hopeful that as you learn the Sail programming language, that you too would want to create some code snippets that you think someone might find helpful.

The simple "hello world" program (found in cookbook/functional_code_snippets/hello_world/) provides a template for writing a new code snippet. For an example that lives in a single Sail file, this should be sufficient. Create a test directory (with a useful name), copy the Makefile and the .sail file into that directory, and then write your code. And finally, edit this .adoc file and give a description of what the example file is intended to do.

Once you have completed your snippet and verifies that it works, you should make an entry in this document. Pleas see "Hello, World" example program (Bill) to see how you should include your snippet in this document. You should at least include the .sail file and give a brief descritpion. Also, please make an entry in List of programming examples (in increasing complexity) for quick perusal by readers.

2.1. Coding and indentation style

We do not have a preferred coding style for these little code snippets. With regards to indentation style, the RISC-V modle follows a vaguely K&R style. Some of the program snippets (those originating with Bill McSpadden) follow the Whitesmiths indentation style. All styles are welcome.

For a list and description of popular indentation styles, steer your browser to... https://en.wikipedia.org/wiki/Indentation_style.

2.2. Brevity

Program examples should be short, both in terms of number-of-lines and in terms of execution time. Each example should focus on one simple item. And the execution of the example item should be clear. The example should be short, standalone and easy to maintain.

Now, we do have one example in this Cookbook that somewhat violates this request. The programming example, [simple_cpu_exampl], is more complex. But it is meant to demonstrate the usefulness of Sail in defining the functionality of an ISA.

2.3. Maintainership (when something breaks)

We would also ask that if you contribute a code example, that you would maintain it.

2.4. 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

Sail is supported on a number of different platforms. MacOs and Lunix/Ubuntu seem to be the most used platforms.

TBD

3.1. Ubuntu (Bill Mc.)

TBD

3.2. MacOS (Martin)

TBD

3.3. Docker

Docker is used as a ….

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)

// 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.generic:

```
# 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
#               then 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 interpretter

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

// The following comment is of interest for reasons other than
//  functionality.  The Sail syntax is still being developed.
//  Attention should be paid to the issues reported to the Sail
//  team (via github) and when releases are made (again via github).

// The following is deprecated per Alasdair Armstrong:
//  I would just remove that example as the cast feature is now
//  deprecated in the latest version (and the risc-v model has
//  always used a flag fully disabling it anyway)

// 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 seperate 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
```

Chapter 8. Control flow

8.1. Iteration

- for
- while
- lamba function

8.2. List iteration example

```
// vim: set tabstop=4 shiftwidth=4 expandtab
// ================================================================
// File:        test.sail
//
// Description: test file for figuring out how to iterate through
//              a Sail list.
//
//              Using code and structure for Ben Marshall's implemetation
//              of RISC-V crypto-scalar code.  (riscv_types_kext.sail).
//
// Author(s):   Bill McSpadden
//
// History:     See git log
// ================================================================

default Order dec

$include <prelude.sail>

overload operator - = sub_bits

val not_vec = {c: "not_bits", _: "not_vec"} : forall 'n. bits('n) -> bits('n)

let aes_sbox_inv_table : list(bits(8)) =
    [|
    0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38,
    0xbf, 0x40, 0xa3, 0x9e, 0x81, 0xf3, 0xd7, 0xfb,
    0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87,
    0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde, 0xe9, 0xcb,

    0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d,
    0xee, 0x4c, 0x95, 0x0b, 0x42, 0xfa, 0xc3, 0x4e,
    0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2,
    0x76, 0x5b, 0xa2, 0x49, 0x6d, 0x8b, 0xd1, 0x25,

    0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16,
    0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65, 0xb6, 0x92,
```

```
    0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda,
    0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d, 0x9d, 0x84,

    0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a,
    0xf7, 0xe4, 0x58, 0x05, 0xb8, 0xb3, 0x45, 0x06,
    0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02,
    0xc1, 0xaf, 0xbd, 0x03, 0x01, 0x13, 0x8a, 0x6b,

    0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea,
    0x97, 0xf2, 0xcf, 0xce, 0xf0, 0xb4, 0xe6, 0x73,
    0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85,
    0xe2, 0xf9, 0x37, 0xe8, 0x1c, 0x75, 0xdf, 0x6e,

    0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89,
    0x6f, 0xb7, 0x62, 0x0e, 0xaa, 0x18, 0xbe, 0x1b,
    0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20,
    0x9a, 0xdb, 0xc0, 0xfe, 0x78, 0xcd, 0x5a, 0xf4,

    0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31,
    0xb1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xec, 0x5f,
    0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d,
    0x2d, 0xe5, 0x7a, 0x9f, 0x93, 0xc9, 0x9c, 0xef,

    0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0,
    0xc8, 0xeb, 0xbb, 0x3c, 0x83, 0x53, 0x99, 0x61,
    0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26,
    0xe1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0c, 0x7d
    |]


// Lookup function - takes an index and a list, and retrieves the
// x'th element of that list.

val sbox_lookup : (bits(8), list(bits(8))) -> bits(8)
function sbox_lookup(x, table) =
    {
    match (x, table)
        {
        (0x00, head::tail) => head,
        (   y, head::tail) => sbox_lookup(x - 0x01, tail)
        }
    }


val main : unit -> unit
function main() =
    {
    let x : bits(8) = 0x03;

    print_bits("lookup results: ", sbox_lookup(x, aes_sbox_inv_table));
```

```
    }
```

8.3. matches

Chapter 9. Other stuff

9.1. *FILE* , *LINE* , *LOC*

Chapter 10. Description prelude.sail

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

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

10.2. the C interface

Chapter 11. Simple CPU example (Martin)

- From nand2tetris

Chapter 12. Formal tools that analyze Sail source code

coverage

Chapter 13. FAQs (Frequently Asked Questions)

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

Chapter 14. Frequently Asked Questions about the Sail RISC-V Golden Model

14.1. 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.

14.2. 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 15. Q:

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

A: 42

15.2. 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.