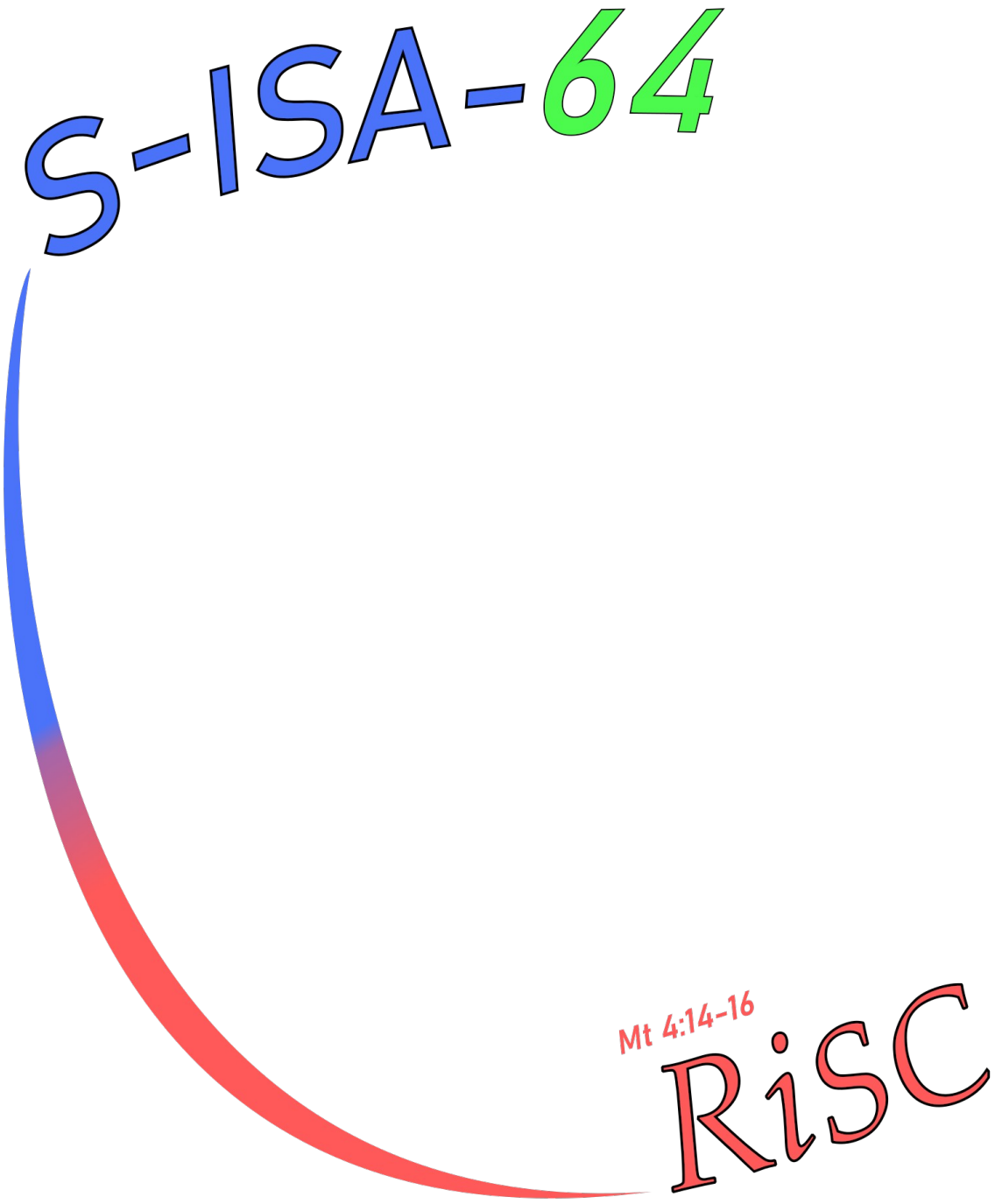
S-ISA-64 Device Standard



Contents

Overview....3
Prerequisite Terminology and Concepts....4
Device Interaction- Execution Flow....5
Default Device Behavior....8
Peripheral Enabled Table....13
Secondary Device Standard Layout....14
Peripheral Address Space Layout....15

Overview

S-ISA-64 accesses non-RAM components through two instructions: the `dread` and `dwrite` instructions, which send data and commands on a special bus dedicated for interacting with peripheral components.

The Programmer's Manual covers the overall functionality of these two instructions in the instruction set, while this document aims to document how components on the system (such as non-volatile storage, a graphics card, keyboard, display, microphone, ethernet controller, or USB hub). Are discovered and interacted with in terms of `dread` and `dwrite`.

It also documents the errata of multi-threaded access to the device bus and what interactions must be implemented.

The Programmer's Manual is considered a *Necessary Companion* for understanding this document.

Prerequisite Terminology and Concepts

This is what this whole document is actually about:

`dread`- instruction in the S-ISA-64 instruction set which **read**s 64 bits from a **d**evice on the **d**evice bus- a special address space which behaves like memory that can be read or written to. The actual instruction takes a single register ID operand and writes the return value to the same, but the effective functionality is to take in an address and return a 64 bit integer. Thus, the prototype is

```
uint64 dread(uint64 address);
```

`dwrite`- instruction in the S-ISA-64 instruction set which **write**s 64 bits to a **d**evice on the **d**evice bus. The actual instruction takes two register ID's as operands, but the effective functionality is to pass in an address and a value, and return nothing. Thus, the prototype is:

```
void dwrite(uint64 address, uint64 val);
```

These two operations form the basis of all interactions with components (excluding main memory) on the system. Components on the system may also be configured (through dwrite operations) to take data from main memory, if supported. (How devices with such functionality behave and how they are configured shall be documented in this standard.)

This document's primary goal is to create a hardware-software interaction standard for the S-ISA-64 architecture **before** hundreds of hours of work has been spent to design the actual hardware, the emulator, operating system development kit, and system libraries.

Unless otherwise specified, Anything called an "address" in this document is an address on the *device bus* and not a main-memory address.

Device Interaction- Execution Flow

Upon boot, the operating system on the computer will check its own memory size by reading address 1, check for the presence of standards-compliant coprocessor cores in address 3, and mutual exclusion devices (“mutexes”) in address 5. Using these values, if applicable, it may read the pointer to the *processor kickstart addresses* in address 4. After reading 1 and 4, the operating system may also try to use one of the *processor kickstart addresses* to start a secondary core. How processor kickstart addresses work shall be documented later. It may also read the pointer to the *Mutual Exclusion Device Addresses* in Address 6. It may also try to retrieve the starting address of Processor Status Variables from Address 7.

Here, we have our first standard rules:

1. ***The memory size and millisecond clock functionalities of addresses 1 and 2 are considered vital to a system and shall not be absent or mangled in a compliant implementation.***
2. ***Any default device addresses which are not implemented as according to this standard, shall safely return zero, or do nothing on a write.***

Finally, and most importantly, the operating system may check for attached devices by checking address 8, and if the return value is non-zero it may check the 0x1000000 device enabled table, and the individual address spaces of the available peripherals, especially Address 0, which is the “DeviceType”.

To enable this functionality...

3. ***If there are no secondary peripherals on the system, address 8 returns 0.***
4. ***If the return value of address 8 is non-zero, then reading the first addresses in the address spaces of the corresponding peripherals must be defined behavior.***
5. ***It shall never result in a system crash, random shutdown, or other “unsafe” behavior to read the first 256 bytes within the address space of any peripheral other than the default device (0).***
6. ***1,048,576 addresses starting at 0x1000000 must return 1 or 0 corresponding to whether that peripheral ID currently has a functioning***

peripheral in it. For addresses not corresponding to the number of peripherals supported by the system, the return value is always zero.

After power on, the operating system may periodically check the Device Enabled Table, and the Device Types of each and every peripheral on the system, so that it can update what services it can provide to user programs.

The User may decide to swap components on the system. For some critical devices, this would most likely require a power off. However, some devices, like keyboards, mice, touchscreens, microphones, gamepads, and flash storage devices may be disconnected or connected at runtime.

Properly disconnecting peripherals in a way that prevents the Operating System from performing *undefined behavior* may require notifying the operating system by the user, however connecting a new peripheral should never change the functionality of other devices. It should not, for instance, cause a still-plugged-in device's Peripheral ID to change, which might cause a write operating in-progress to be moved from a USB stick to a hard drive.

Reading and writing to a disabled peripheral is undefined behavior that may result in hardware damage. Reading and writing to a peripheral in a way in which it was not designed to be read from or written to is undefined behavior.

On a real system, then, **a Peripheral ID corresponds more to a physical expansion port on the system, rather than the peripheral itself.**

The implementation is required to keep track of the number of active, functioning peripherals on a system at any given time and where they are on the system.

What types of peripherals and ports are considered "safe" to hot-swap at a given time is implementation defined. How an operating system handles hot-swapped peripherals to prevent itself from executing undefined behavior is implementation defined.

Other cores on the system may also access the device bus, however accessing the same device asynchronously is not necessarily safe unless the device indicates that it is thread-safe (Having atomic-style behavior on a read/write).

Default Device Behavior

The default device, Peripheral 0, is the heart of the implementation and is vital to the functionality of any operating system.

Serial I/O UART (Address 0)

The Serial I/O device provides blocking read/write of ASCII character values for debugging purposes. It is not considered essential for a S-ISA-64 implementation as its primary purpose is debugging, but this address must only be used for that purpose. When dread is invoked on the address, the processor is halted until a value is retrieved from the input device. An 8-bit ASCII character (the bottom 7 being the 7-bit ASCII standard) value is returned in the bottom 8 bits of the 64 bit return value. When dwrite is invoked on the address, the bottom 8 bits of the input are treated as an 8-bit ASCII value, and should be displayed, transmitted, or broadcast as such to the appropriate serial port.

Memory Size (Address 1)

The Memory Size query device returns the amount of available system RAM for the implementation. Writing to this address is undefined behavior, but reading it shall return the number of available addressable contiguous bytes to the S-ISA-64 processor starting at address zero.

Clock (Address 2)

The Clock device returns a measure of the number of milliseconds passed since some point in the past to the caller on a dread. It must provide at least 32 bits of wrap-around (It may wrap around every 4,294,967.296 seconds).

Processor Count (Address 3)

The processor count device returns the number of available compliant processors on the system, which is dubbed **Np**. If the implementation is single-threaded, it may return either 0 (unimplemented) or 1. The behavior of Processors shall be documented in *Multiprocessor Essentials.*

Processor Kickstart Address Array Pointer (Address 4)

Reading address 4 shall return a valid device bus address (Dubbed "**Kickstart[0]**"), which, if written to, attempts to start the first processor on the system (Which is the ***boot processor***). The next address ("**Kickstart[1]**") attempts to start the second processor on the system, if available, and so on, up to **Kickstart[Np-1]** where **Np** is the number of compliant processors on the system, as provided by address 3.

If there are no available kickstart addresses, or they are unimplemented, Address 4 shall return 0.

If reading address 3 returns 0, then reading address 4 must also return 0.

Mutex Count (Address 5)

Address 5 shall return the number of available contiguous mutual exclusion devices on the system, which is dubbed **Nm**. If none are available, zero is returned.

Mutual Exclusion Device Array Pointer (Address 6)

Address 6 shall return a valid device bus address (Dubbed "**Mutex[0]**"), which, if a value of 0 is written to it, shall attempt to "unlock" it. And if a value of 1 is written to it, shall attempt to "lock" it. Writing other values is *implementation defined behavior.* The next address, ("**Mutex[1]**") shall behave identically to Mutex[0] if **Nm** is at least 2.

If reading address 5 returns 0, then address 6 must also return 0.

Processor Status Variable Array Pointer (Address 7)

Reading Address 7 shall return a valid device bus address (Dubbed "**Pstat[0]**") which, if read, shall return a value indicating whether or not the processor is running and/or currently unavailable to be kickstarted by writing to its corresponding Kickstart address (which in this case would be Kickstart[0])

If the processor is running, and not able to be kickstarted, the value returned shall be 1.
if the processor is available to be kickstarted, the value returned shall be 0.
if the processor is for any reason unable to be kickstarted, the value returned shall be 0.

The next address, **Pstat[1]** shall behave identically to **Pstat[0]** if available.

The number of processor status variables, if implemented, must be at least as large as the number of processors, **Np**.

If the return value of Address 3 is zero, the return value of Address 7 must also be zero.

Peripheral Count (Address 8)
Address 8 shall yield the number of secondary devices (Dubbed "**NeX**") on the system (Dubbed "**Peripherals**"). Each secondary device (Peripherals 1 – **NeX**) has a series of attributes stored at the beginning of its address space. If only the default device is available, the return value is 0.

Peripherals being plugged in or unplugged from a system does not change after power-on, rather, they just become disabled.

Pointer to Main Memory Mirror (Address 9)
Returns Device Bus Address where a 64-bit-aligned mirror of main memory can be accessed. This exists to allow DMA controllers to read and write on the device address bus rather than using main memory.

Set Is Serial NonBlocking (Address 10)
Reading this will reveal whether Serial is nonblocking or not (0 or 1)
Writing 0 will make it blocking, and 1 will make it non-blocking.

Set System Time (Address 11)
Reading this will reveal whether the system time can be set (0 or 1).
Writing will set the system time in milliseconds to the value written.

Default Device Vendor String Pointer (Address 12)
Returns a Device Bus Address where a null-terminated ASCII string specifying vendor-specific information (vendor name, model, extensions, whatever) can be read. Vendor strings are read 8 bits at a time, so contiguous addresses yield increasing bytes.
If the vendor string started at 0x100 and the string was "Hi!" then reads would look like this:

```
read(0x100) → 'H'
read(0x101) → 'i'
```

```
read(0x102) → '!'
read(0x103) → 0
```

This model of how a “vendor string” works shall be mimicked by all compliant secondary devices.

Peripheral Enabled Table

At address 0x1000000 on the device bus, a series of boolean values can be queried to indicate whether an associated peripheral is enabled. The values correspond to the peripherals 0-NeX. Peripheral 0 is always available so it is locked at 1. If peripheral 1 is enabled, 0x1000001 will also yield a 1.

This is, essentially, how plug-and-play is handled. If a device is unplugged, it become disabled. The peripheral count does not change during the runtime of the computer. However, a peripheral may **change functionality** or **become disabled** during runtime, and if that happens, its boolean value in this table will be zero.

In order to tell what type a device is, the device must itself be queried from its address space.

Secondary Device Standard Layout

Peripherals on a system are identified by a peripheral ID, from 0 (the default device) to NeX (inclusive), which has a maximum value of 1,048,575.

Any address on peripheral 1, then, looks like this in bits:
`0000 0000 0000 0000 0001 XXXX XXXX XXXX (…)`

Where (…) is filled in with X's corresponding to the remaining bits of the address.

Address 1 on peripheral 1 replaces all those X's with zeroes, except for the very last X, which is a 1.
To *safely* convert from a Peripheral ID and a Peripheral Address to a Device Bus address, you do the following transformation...
in C pseudocode...

```
dev_addr =
    ((uint64_t)peripheral_ID << 44)
    |
    (peripheral_addr & 0xFffFFffFFff);
```

where dev_addr is the resulting device bus address, Peripheral ID is the ID of the peripheral (0 being the default device) and peripheral_addr being the address within that peripheral's address space.

Peripheral Address Space Layout

Within a peripheral's address space, the following addresses shall be reserved, and required to be implemented, for the following purposes:

Address 0- Device Type. (Read only- Write is UB)

This address shall yield a Device Type ID which tells the operating system what type of device this peripheral is. The specification of Device Type IDs shall be given later. 0 is returned if the device is disabled. 0xFFffFFffFFffFFff is returned if the device does not use a device type ID.

Address 1- Device Vendor ID (Read only- Write is UB)

This address shall yield a Vendor ID. Vendor ID 0 is "None". The list of reserved Vendor IDs shall be listed later.

Address 2- Pointer to Device Vendor String (Read only- Write is UB)

Similar to the Default Device's vendor string system. Returns a peripheral address you can read sequentially within the peripheral's address space (Not prefixed, the top 20 bits are empty) to get an ascii string identifying various vendor-specific properties of the peripheral as ASCII text. The string is null-terminated.

Address 3- Write-to-Disable (Read/Write)

Reading this address says whether or not the peripheral can be disabled, in preparation for it being safely removed from a system during runtime. If the return value is '0' then writing to this address does nothing, but if the return value is '1' then writing will cause the peripheral to ***eventually*** disable itself. The OS should regularly check the appropriate entry in the **Device Enabled Table** to discover when the device has safely disabled itself so that the user's unplugging the device does not result in *undefined behavior.*

Address 4- Is Thread Safe.

Reading this address says whether or not the peripheral is thread-safe. If every single functional address on the device can be read from or written to atomically, then this returns 1. Otherwise, reads and writes must be guarded by a mutex, or only ever happen on a single core.

The rest of the address space is free for the peripheral to implement.