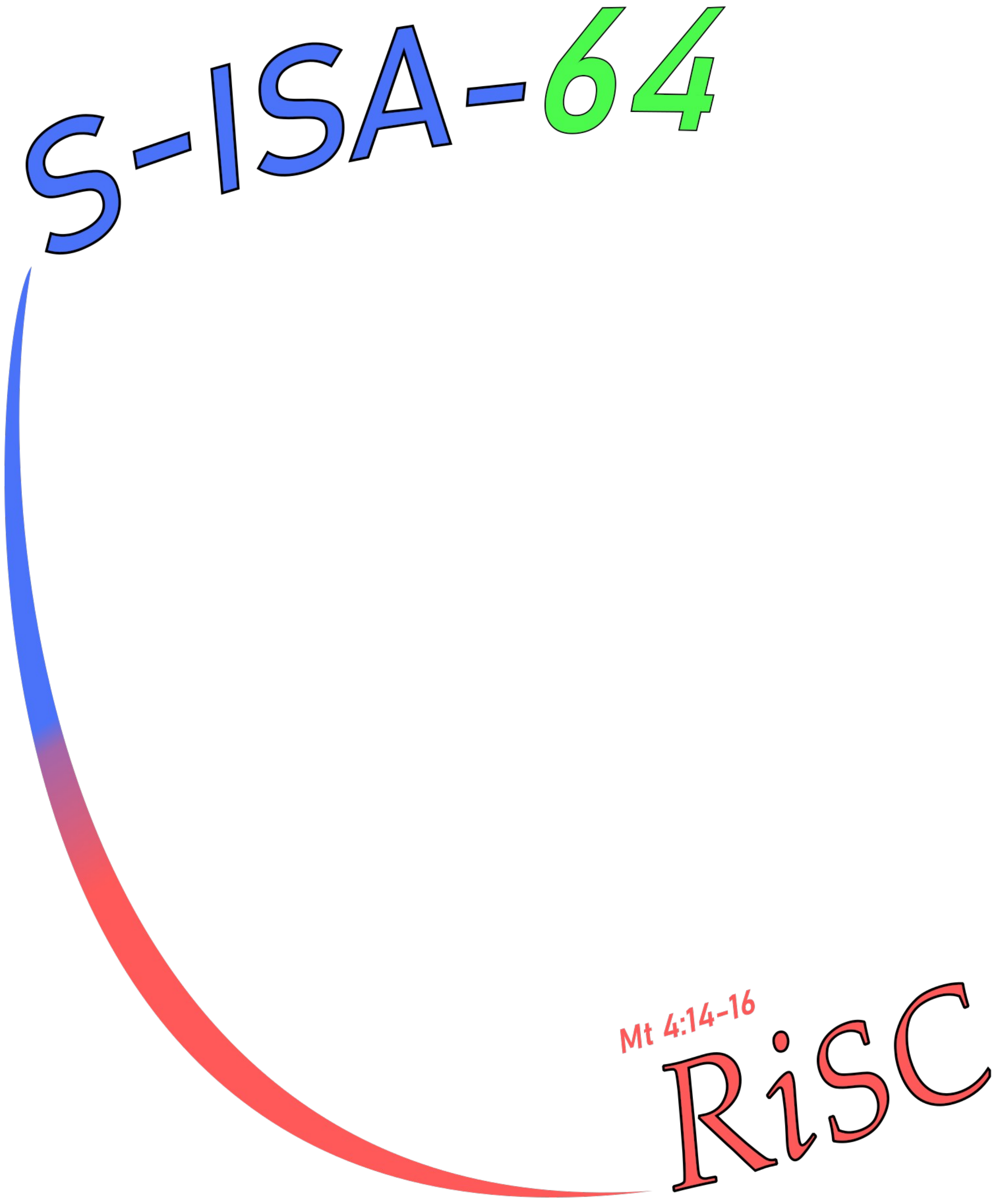
S-ISA-64 Device Standard



Contents

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 unimplemented addresses from 0 to 8 inclusive, which are not implemented as according to this standard, shall safely return zero, which is considered the error value.***

Finally, and most importantly, the operating system may search for device descriptors by checking address 8, and if the return value is non-zero it may check the device descriptor address array starting at 0x100.

3. ***If there are no peripheral device descriptor tables on the system, the return value of reading address 8 on the device bus shall be zero.***
4. ***If the return value of address 8 is non-zero, then reading addresses starting at 0x100 shall return pointers to standards-compliant device descriptors.***

Each device descriptor table may be queried and interacted with according to the standard, and the complete tree (as other device descriptor tables may be provided) may be searched.

Default Device Behavior

The default devices in addresses 0-5 are considered vital to the functionality of the S-ISA-64 implementation, especially 1 and 2. *It is safe to assume that addresses 1 and 2 will always be available and function normally unless specifically configured otherwise through device descriptors.*

Serial IO Device (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 Device (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 Device (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 Address Device (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.

Mutual Exclusion Device Count Device (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 Address Device (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 Address Device (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.

Device Descriptor Count Device (Address 8)

Address 8 shall yield the number of device descriptor tables whose device bus addresses are accessible by reading 0x100 (device bus) and onward. The return value of Address 8 is dubbed “**NeX**”.

The return value may be zero if number of peripheral devices available on the system is zero.

If **NeX** is zero, then the operating system may safely assume that there are no peripherals on the system.

Device Descriptor Table Address Array (Addresses 0x100 – 0x100+NeX-1)

Addresses 0x100 (256 decimal) through 0x100 + NeX -1 shall return valid device bus addresses, each of which is the address of a **Device Descriptor Table** which is a series of contiguous device bus addresses that can be used to configure a device or query its properties. It may also provide the ability to interact with that device, and for some Device types, it may provide functionality not defined in this standard.

Device Descriptor Table Format

Every non-default device on the system, including virtual devices used only for configuring other devices, must expose itself using a “device descriptor table”.

What is a device descriptor table?

A series of contiguous device bus addresses which can be read from or written to, to query, discover, or interact with a device. It can be thought of like a struct in C or class in C++, but with potentially volatile members.

What do you mean “Contiguous addresses?”

Let’s say there’s only one peripheral on the system. Device Bus Address 8 will return the value 1 and address 0x100 will return another address, let’s say that it returns the value 0x100000000100.

0x100000000100 is the first address of the device descriptor table for the only peripheral on the system. Reading from it returns the “Device Descriptor Identifier” or “DescID” of that device.
0x100000000101 is the second address of the device descriptor table, which based on the DescID returned from 0x100000000100, may do a variety of different things if read from or written to.
It may do nothing at all or may not even be implemented depending on what kind of device it is.

(continued on next page)

You can think of it as a (packed) struct in C with a variable length array member (NOTE: Not a pointer!)
Here is a C99 style declaration:

```
typedef struct{
    uint64_t DescID;
    uint64_t Stuff[];
} Device_Descriptor_Entry;
```

the addresses retrieved by reading 0x100,0x101, etcetera can each be thought of as being a `volatile Device_Descriptor_Entry*` (pointer to device descriptor entry).

In the above example, 0x100000000100 was the address of “DescID” and 0x100000000101 was the address of “Stuff[0]”.
0x100000000102 would be the address of “Stuff[1]” and so on and so forth.

For readability, the rest of the document will use the notation defined by the above struct declaration.

Recall that the device bus in S-ISA-64 is addressed by the word size, not by byte.

Each device descriptor table listing in this document gives a listing of what each of the entries in Stuff[] do for a particular Device Descriptor ID.

Common Conventions and Notations:

Feature Bitmask. Stuff[0] typically contains a value where each bit in the value is to be treated as a **boolean** indicating whether some feature is supported by the device or not. **Writing** to the feature bitmask is *implementation defined.* Bits are identified by their power of two- for example, bit 0 is the bit which can be on or off to indicate a “1” in the bottom binary place. Bit 1 is the bit which can be on or off to indicate a “2”, etc.

Feature Bitmask Configurator. A later entry in Stuff[] may contain a *configurator* for the device, which can be written to modify the behavior of the device. A subsequent *read* of the feature bitmask can be done by the calling code to check if the change went through. If the requested feature set change is not supported, **no change will occur**, not even the ones that are supported.

Returns a Pointer to…
Entries in Stuff[] in Device descriptor tables usually don't encapsulate the functionality of the device itself, but rather return device bus addresses which can be read from or written to, to provide the device's main functionality.
So, for a Fully standardized *Null Device*, Stuff[2] returns a *valid device bus address* which itself can be passed back into *dread* to get a value from the Null Device.

(READ ONLY -UB)/(WRITE ONLY -IDB)/(UNUSED -IDB)
Only Reading, or only Writing that device bus address are standardized behavior by this document. Doing the wrong operation could be dangerous (undefined behavior).

Unused indicates that the standard has no requirement on how that entry must be used.

The -UB or -IDB indicate whether doing the opposite function is <u>undefined behavior</u> or simply <u>implementation defined behavior.</u>

<u>BITMASK DEPENDENT: -UB</u>
If the behavior of an entry in Stuff[] is dependent on the configuration of a bit in the Feature Bitmask, then this will appear in its subsection.
-UB or -IDB once again indicate how this field is supposed to behave.

Pointer to **Vendor String.**
Pointer to the first of a contiguous series of addresses each returning an 8-bit ASCII value, comprising a null-terminated string meant to specify the vendor of the device. Every entry is (READ ONLY -UB)

Device Descriptor Table Safety Rules.

1) it must always be safe (causing no side effects other than the return value) to read the first entry of a device descriptor table (DescID). Writing is *implementation defined behavior.* This includes multi-threaded concurrent accesses.

2) Reading or writing to the second or farther (Stuff[0], Stuff[1]) entries in a device descriptor table is safe or unsafe as defined by this standard. It may be undefined, strictly defined, or implementation defined behavior depending on DescID.

Device Descriptor Identifiers

For a complete listing of all valid Device Descriptor Identifier values (DescID values), see *Device Descriptor Identifier Table* in the Programmer's Manual.

As described before, it is considered a *Necessary Companion* for this document.

Multiprocessor Essentials

The S-ISA-64 standard allows for systems that have multiple physical processors running concurrently on the same memory and using the same device bus.

The boot process, including how processors are started, is documented in the *Programmer's Manual*.

Here are the fundamental rules of how multiprocessor interactions work:

1) Separate processors do not share registers, not even the stack pointer or addressing registers. They may each be in privileged or user mode at any particular time.

2) Separate processors access same the main system memory or device bus, and may do so concurrently.

3) Concurrent reads to the same memory location are allowed, but any access by a processor concurrent with a write on another is implementation defined behavior (For the definitions of implementation defined behavior and undefined behavior, See *Terminology* in the Programmer's Manual).

4) Processor 0 (the "boot processor") is the only one running on system reset or power-on, and it has all its registers cleared, running in privileged mode, executing code from address 0 when started.

5) Other processors are powered off, and should be available to be kick-started, at the end of a system reset, or at power-on.

6) Concurrent accesses to an address pertaining to any particular device on the device bus is *undefined behavior* unless otherwise specified.

Default Device Standardized Extensions

<u>0 – Null Device</u>

Description:

Does nothing, returns zero. Some configurations may also allow returning random numbers.

Stuff[0] – Feature bitmask (READ ONLY -UB)

- Bit 0: Unused.
- Bit 1: Customizable Value bit (1- Can customize value)
- Bit 2: Random Number Generator. (1- Returns random numbers)
- Bit 3: Can set bitwidth. (1- Can set bit width)

Stuff[1] – Feature Bitmask Configurator. (WRITE ONLY -UB)

Stuff[2] – Get pointer to Null Device. (READ ONLY -UB)

> Returns a valid device bus address, which can be read to get a value of zero. If the device has a “value setter” then you can write a 64 bit value to the value setter and it will be returned (possibly with some of the high bits set to zero, based on the bit width)
>
> *Returned Pointer*- (READ ONLY- UB)

Stuff[3] – Get pointer to Null Value Setter. (READ ONLY -UB)

> <u>BITMASK DEPENDENT: -UB</u>
> Bit 1 == 1
>
> Returns a valid device bus address, which can be written to, to set the value returned by the Null Device.
>
> *Returned Pointer*- (WRITE ONLY- UB)

Stuff[4] – Current Bit Width (READ ONLY -UB)

Get the number of bits that the Null Device returns.

Stuff[5] – Set Bit Width (WRITE ONLY -UB)

<u>BITMASK DEPENDENT: -UB</u>

Bit 3 == 1

Write to set the bit width. Invalid bit widths do not change the width.

Stuff[6] – Pointer to **Vendor String**.

1 - Serial I/O Extension

Description:

Serial Input/Output. Meant to mimic the behavior of a UART. Defaults to 8 bit value transmission.

Stuff[0] – Feature bitmask. (READ ONLY- UB)

Bit 0- Non-blocking I/O. (0- blocking, 1- blocking)

(For non-blocking I/O the value received when there is no value, is that all the bits are set to 1.)

Bit 1- Supports other bitwidth sizes. (0- Fixed size, 1- Changeable size)

Bit 2- Input Is Unbuffered. (0- Buffered, 1- Unbuffered)

Bit 3- Can set Input buffer size. (0- Can't set, 1- Can set)

Bit 4- Input Can be flushed. (1- Can flush, 0- Cannot flush)

If the implementation supports flushing the input, this will be set to 1.

Bit 5- Input Buffer Overflow Guaranteed Behavior.

If set to 1, the behavior of an Input Buffer Overflow is guaranteed. If set to zero, what happens when the serial line's input buffer overflows is partially implementation defined. It may not cause the system to reset or crash.

Bit 6- Input Buffer Overflow Behavior.

If set to Zero, the behavior is to mimick the behavior of a shift register. When the buffer overflows, old values are discarded. If set to One, the behavior is to discard new values until the buffer is emptied by a flush or by reading the serial line.

Stuff[1] – Feature Bitmask configurator (WRITE ONLY- UB)

Stuff[2] – Bit Width. (READ ONLY- UB)

<u>BITMASK DEPENDENT: -UB</u>

Bit 1 == 1

Returns current bit width of serial line.

Stuff[3] – Bit Width Setter. (WRITE ONLY- UB)

<u>BITMASK DEPENDENT: -UB</u>

Bit 1 == 1

Set the width of the serial line in bits. If an unsupported value is written, the bit width does not change.

Stuff[4] – Input Buffer Size. (READ ONLY- UB)

BITMASK DEPENDENT: -UB
Bit 2 == 0

Read to get the current size of the input buffer for input values. The unit is input entries. **Not** bytes.

Stuff[5] – Input Buffer Maximum Size (READ ONLY- UB)

BITMASK DEPENDENT: -UB
Bit 3 == 1

Maximum possible size of input buffer for input values. Again, the unit is *input entries* and **not *bytes*.**

Stuff[6] – Input Buffer Change Size. (WRITE ONLY- UB)

BITMASK DEPENDENT: -UB
Bit 3 == 1

Change the size of the input buffer. Invalid values do not change the size.

Stuff[7] – Pointer to flush input. (READ ONLY -UB)

BITMASK DEPENDENT: -UB
Bit 4 == 1

Returns a pointer to an address which can be written to, to clear the input buffer.

Returned Pointer: (WRITE ONLY- UB)

Write to this address to flush the input buffer of the Serial Line without reading it.

Stuff[8] – Pointer to **Vendor String**.

Memory Extension

Description:

Changes the behavior of Address 1.

Get the size of the available memory, change the amount of available memory, enable/disable a memory cache (which applies to all processors, not just the current one), and more.

Stuff[0] – Feature Bitmask (READ ONLY -UB)
Bit 0- Supports changing available memory size.
Bit 1- Has Cache (1- Has Cache, 0- No Cache)
Bit 2- Can toggle cache (1- Can toggle, 0- Can’t toggle)
Bit 3- Cache Enabled/Disabled (1- Enabled, 0- disabled)
Stuff[1] – Feature Bitmask Configurator (WRITE ONLY -UB)
Stuff[2] – Pointer to Memory Size Changer (WRITE ONLY -UB)
Recall that the memory size can be queried from address 1.

Returned Pointer: (WRITE ONLY -UB)
Writing to this address sets the amount of available system memory.
Invalid values do not change the amount of available system memory.
Stuff[3] – Pointer to **Vendor String**.

Clock Extension

Description:

Change the behavior of Address 2.

Get the current time in Milliseconds, Nanoseconds, or Seconds. Also allows you to set the current time value in the currently set units.

Stuff[0] – Feature Bitmask
- Bit 0- Supports changing the unit (1- Can change)
- Bit 1- Supports Nanoseconds (1- Supports)
- Bit 2- Supports Microseconds (1- Supports)
- Bit 3- Supports Seconds (1- Supports)
- Bit 4- Nanoseconds Mode (1- Is in nanoseconds mode)
- Bit 5- Microseconds Mode (1- Is in microseconds mode)
- Bit 6- Seconds Mode (1- is in seconds mode)
- Bit 7- Can Set Time (1- Can set)
- Bit 8- Continues running on shutdown (1- Non-volatile)
- Bit 9- Battery Powered. (1- Has Battery)

Stuff[1] – Feature Bitmask Configurator.

Stuff[2] – Pointer to Time Setter.

BITMASK DEPENDENT: -UB
Bit 7 == 1

Returned Pointer:(WRITE ONLY -UB)
Write the desired set time to this address to set the time of the system clock.

Stuff[3] – Pointer to **Vendor String**.