Module ehal

This is the Erlang interface to the Epiphany eHAL NIF loadable library.

Version: 1.0.1

Authors: Mark A Fleming (mark_fleming@ieee.org).

Description

This is the Erlang interface to the Epiphany eHAL NIF loadable library. If the module is not in the current working directory, use

```
code:add_path("path to module directory")
```

to include the module directory in the Erlang node search path.

Introduction

This module provides an Erlang interface to the Epiphany SDK eHAL function set. Every effort has been made to hew to the name and argument list order of the original C functions. This was done to simplify the porting of C code to Erlang, to maintain familiarity as much as possible for the experienced SDK user, and to avoid the usual temptation to "improve" the original source arrangement.

Erlang Term Conventions

All SDK C structure values are represented as Erlang binaries. This is done to isolate Erlang code from any changes to the SDK structure definitions. An Erlang program will just pass these binary blobs around, leaving it up to the NIF functions to transform them to and from the corresponding C structures. An experienced programmer can use the bit syntax to prize apart the binary into the corresponding structure members.

Data written to or read from an eCore internal memory or Epiphany external memory is also represented as a binary. Erlang has a number of ways to convert between regular Erlang terms and binaries. For example, `list_to_binary("Hello World")` can be used to put a string in binary format and `binary_to_list(Bin)` can do the reverse. More detail can be found in `erlang(3)`

Many of the SDK's defined values and enumerations can be found in the Erlang 'ehal.hrl' include file. The file defines macros to represent such constants as E_REG_CONFIG (use ?E_REG_CONFIG). A small deviation from the SDK standard was to use the boolean atoms 'true', 'false', 'ok' and 'error' in place of the defined integer values E_TRUE, E_FALSE, E_OK and E_ERR respectively.

Module functions use the Erlang convention of returning the atoms 'ok' and 'error' or the tuple '{ok, Result}'. For example, most of the SDK C functions return a success flag and use pointer arguments to return other values. In Erlang, the 'e_open/4' function returns '{ok, Dev}' for the `e_epiphany_t` structure returned by the corresponding C function.

Data Types

epiphany_dev()

```
epiphany_dev() = binary()
```

epiphany_mem()

```
epiphany_mem() = binary()
```

platform_data()

```
platform_data() = {row, integer()} | {col, integer()} | {rows, integer()} | {cols,
integer()} | {version, string()} | {regs_base, integer()} | {num_chips, integer()} |
{num_emems, integer()}
```

platform_info()

```
platform_info() = [platform_data()]
```

Function Index

e_alloc/2	This function defines a buffer in external memory of size 'Size' bytes starting at offset 'Base' bytes from the beginning of external memory.
e_close/1	This function closes the eCode workgroup and releases resources allocated by the e_open function.
e_finalize/0	This function finalizes the connection to the Epiphany chip and releases resources allocated by the e_init

	function.
e_free/1	This function releases the resources allocated by the e_alloc/2 function.
e_get_coords_from_num/2	This function converts a workgroup's core number to the core coordinate relative to the workgroup origin.
e_get_num_from_coords/3	This function converts a workgroup's core coordinate to the core number within the workgroup.
e_get_platform_info/0	This function returns information about the Epiphany platform as a list of tuples of the form {attribute, Value}.
e_halt/3	This function halts a workgroup core's program execution.
e_init/1	This function initializes the HAL data structures, and establishes a connection to the Epiphany platform.
e_is_addr_on_chip/1	This function returns `true` if the given 32-bit address is within the Epiphany's physical address space, or `false` otherwise.
e_is_addr_on_group/2	This function returns `true` if the given 32-bit address is within the workgroup cores physical address space, or `false` otherwise.
e_load/5	This function loads a program into a workgroup core.
e_load_group/7	This function loads a program into a subgroup of a workgroup's cores.
e_open/4	This function defines a eCore workgroup.
e_read/5	This function reads data from a workgroup eCore local memory if Dev is of type `e_epiphany_t` or from an external memory buffer if Dev is of type `e_mem_t`.
e_reset_group/1	This function performs a soft reset of a eCore workgroup.
e_reset_system/0	This function performs a full hardware reset of the Epiphany platform, including the Epiphany chip and FPGA glue logic.
e_resume/3	This function resumes a workgroup core's program execution that was previously halted with a call to e_halt.
e_set_host_verbosity/1	This function sets the verbosity level of eHAL function calls.
e_set_loader_verbosity/1	This function sets the verbosity level of program loader function calls.

e_signal/3	This function sends a soft interrupt to a workgroup core.
e_start/3	This function starts a core, normally after a program has been loaded.
e_start_group/1	This function starts all cores in a workgroup, normally after a program has been loaded.
e_write/6	This function writes data to a workgroup core local memory if Dev is of type `e_epiphany_t` or to an external memory buffer if Dev is of type `e_mem_t`.
init/0	Search for and load the NIF loadable library.

Function Details

e_alloc / 2

```
e_alloc(Base, Size) -> {ok, Mbuf::epiphany_mem()} | error

    Base = integer()
    Size = integer()
```

This function defines a buffer in external memory of size 'Size' bytes starting at offset 'Base' bytes from the beginning of external memory. If successful, a binary (e_mem_t structure) is returned that can be used in subsequent read and write operations.

See Also

Epiphany SDK Reference

External Documentation Archive

e_close / 1

```
e_close(Dev) -> ok | error

    Dev = epiphany_dev()
```

This function closes the eCode workgroup and releases resources allocated by the e_open function. This function should be used before re-allocating an eCore member to a new workgroup.

See Also

Epiphany SDK Reference

External Documentation Archive

e_finalize / 0

```
e_finalize() -> ok | error
```

This function finalizes the connection to the Epiphany chip and releases resources allocated by the e_init function.

See Also

Epiphany SDK Reference

External Documentation Archive

e_free / 1

```
e_free(Mbuf) -> ok | error

     Mbuf = epiphany_mem()
```

This function releases the resources allocated by the e_alloc/2 function.

See Also

Epiphany SDK Reference

External Documentation Archive

e_get_coords_from_num / 2

```
e_get_coords_from_num(Dev, Corenum) -> {ok, {Row::integer(), Col::integer()}} |
error

     Dev = epiphany_dev()
     Corenum = integer()
```

This function converts a workgroup's core number to the core coordinate relative to the workgroup origin.

See Also

Epiphany SDK Reference

External Documentation Archive

e_get_num_from_coords / 3

```
e_get_num_from_coords(Dev, Row, Col) -> {ok, integer()} | error

     Dev = epiphany_dev()
```

```
    Row = integer()
    Col = integer()
```

This function converts a workgroup's core coordinate to the core number within the workgroup.

See Also

Epiphany SDK Reference

External Documentation Archive

e_get_platform_info/0

```
e_get_platform_info() -> {ok, Plat::platform_info()} | error
```

This function returns information about the Epiphany platform as a list of tuples of the form {attribute, Value}. Valid attributes at this time are:
row, col, rows, cols, version, num_chips, num_emems, regs_base

Use lists:keyfind/2 to extract a desired key-value pair.

e_halt/3

```
e_halt(Dev, Row, Col) -> ok | error

    Dev = epiphany_dev()
    Row = integer()
    Col = integer()
```

This function halts a workgroup core's program execution. The Row and Col eCore coordinate is relative to the workgroup given as Dev.

See Also

Epiphany SDK Reference

External Documentation Archive

e_init/1

```
e_init(HDF) -> ok | error

    HDF = string() | 0
```

This function initializes the HAL data structures, and establishes a connection to the Epiphany platform. The platform parameters are read from a Hardware Description File (HDF), whose path is given as the function argument. If the HDF argument is a 0 value or empty string, the function will use environment variable information

(EPIPHANY_HDF or EPIPHANY_HOME) to try to locate the default HDF file.

See Also

Epiphany SDK Reference

External Documentation Archive

e_is_addr_on_chip/1

```
e_is_addr_on_chip(Addr) -> true | false

    Addr = integer()
```

This function returns `true` if the given 32-bit address is within the Epiphany's physical address space, or `false` otherwise.

See Also

Epiphany SDK Reference

External Documentation Archive

e_is_addr_on_group/2

```
e_is_addr_on_group(Dev, Addr) -> true | false

    Dev = epiphany_dev()
    Addr = integer()
```

This function returns `true` if the given 32-bit address is within the workgroup cores physical address space, or `false` otherwise.

See Also

Epiphany SDK Reference

External Documentation Archive

e_load/5

```
e_load(Program, Dev, Row, Col, Start) -> ok | error

    Program = string()
    Dev = epiphany_dev()
    Row = integer()
    Col = integer()
    Start = true | false
```

This function loads a program into a workgroup core. The Program string points to a file in SREC format. The Row and Col address of the core is relative to the workgroup's origin. When the Start value is 'true', the program is launched after it is loaded. The function also accepts a null or zero Program parameter.

Program load should only be performed when a core is in an idle or halt state, ideally after a e_reset_system/0 or e_reset_group/1.

See Also

Epiphany SDK Reference

External Documentation Archive

e_load_group/7

```
e_load_group(Program, Dev, Row, Col, Rows, Cols, Start) -> ok | error

     Program = string()
     Dev = epiphany_dev()
     Row = integer()
     Col = integer()
     Rows = integer()
     Cols = integer()
     Start = true | false
```

This function loads a program into a subgroup of a workgroup's cores. The Program string points to a file in SREC format. The Row and Col address of the core group is relative to the workgroup's origin. The range of cores in the subgroup is given by Rows and Cols. When the Start value is 'true', the program is launched after it is loaded.

Program load should only be performed when a core is in an idle or halt state, ideally after a e_reset_system/0 or e_reset_group/1.

See Also

Epiphany SDK Reference

External Documentation Archive

e_open/4

```
e_open(Row, Col, Rows, Cols) -> {ok, Dev::epiphany_dev()} | error

     Row = integer()
     Col = integer()
     Rows = integer()
```

```
    Cols = integer()
```

This function defines a eCore workgroup. The Row and Col parameters defines the workgroup origin relative to the platform origin and the Rows and Cols parameters gives the workgroup size. The function returns a binary (e_epiphany_t structure) that can be used in subsequent calls to reference the workgroup. The e_get_platform_info/0 function can be used to determine the size of the Epiphany chip in rows and columns.

See Also

Epiphany SDK Reference

External Documentation Archive

e_read/5

```
e_read(Dev, Row, Col, From, Size) -> {ok, {Ssize::integer(), Data::binary()}} |
error

    Dev = epiphany_dev() | epiphany_mem()
    Row = integer()
    Col = integer()
    From = integer()
    Size = integer()
```

This function reads data from a workgroup eCore local memory if Dev is of type `e_epiphany_t` or from an external memory buffer if Dev is of type `e_mem_t`. If successful, a tuple containing the number of bytes read and a binary of the data read is returned. If accessing the core local registers, use one of the register symbol macros in ehal.hrl as the From offset.

The From argument specifies an offset from the base of the eCore internal memory or from the external memory buffer. The Row and Col arguments specify a particular eCore when reading from local memory and are ignored when reading from external memory.

See Also

Epiphany SDK Reference

External Documentation Archive

e_reset_group/1

```
e_reset_group(Dev) -> ok | error

    Dev = epiphany_dev()
```

This function performs a soft reset of a eCore workgroup. See SDK documentation for cautions on using this function.

See Also

Epiphany SDK Reference

External Documentation Archive

e_reset_system / 0

```
e_reset_system() -> ok | error
```

This function performs a full hardware reset of the Epiphany platform, including the Epiphany chip and FPGA glue logic.

See Also

Epiphany SDK Reference

External Documentation Archive

e_resume / 3

```
e_resume(Dev, Row, Col) -> ok | error

     Dev = epiphany_dev()
     Row = integer()
     Col = integer()
```

This function resumes a workgroup core's program execution that was previously halted with a call to e_halt. The Row and Col eCore coordinate is relative to the workgroup given as Dev.

See Also

Epiphany SDK Reference

External Documentation Archive

e_set_host_verbosity / 1

```
e_set_host_verbosity(Level) -> integer()

     Level = integer()
```

This function sets the verbosity level of eHAL function calls. The old diagnostic level is returned. Diagnostic levels are defined by the macros ?H_D0 to ?H_D4 in the 'ehal.hrl' include file. Level ?H_D0 means no diagnostics and each remaining level provides more detailed diagnostics.

See Also

Epiphany SDK Reference

External Documentation Archive

e_set_loader_verbosity/1

```
e_set_loader_verbosity(Level) -> integer()

     Level = integer()
```

This function sets the verbosity level of program loader function calls. The old diagnostic level is returned. Diagnostic levels are defined by the macros ?L_D0 to ?L_D4 in the 'ehal.hrl' include file. Level ?L_D0 means no diagnostics and each remaining level provides more detailed diagnostics.

See Also

Epiphany SDK Reference

External Documentation Archive

e_signal/3

```
e_signal(Dev, Row, Col) -> ok | error

     Dev = epiphany_dev()
     Row = integer()
     Col = integer()
```

This function sends a soft interrupt to a workgroup core. The Row and Col eCore coordinate is relative to the workgroup given as Dev.

See Also

Epiphany SDK Reference

External Documentation Archive

e_start/3

```
e_start(Dev, Row, Col) -> ok | error

     Dev = epiphany_dev()
     Row = integer()
     Col = integer()
```

This function starts a core, normally after a program has been loaded. The Row and Col eCore coordinate is relative to the workgroup given as Dev.

See Also

Epiphany SDK Reference

External Documentation Archive

e_start_group/1

```
e_start_group(Dev) -> ok | error

     Dev = epiphany_dev()
```

This function starts all cores in a workgroup, normally after a program has been loaded.

See Also

Epiphany SDK Reference

External Documentation Archive

e_write/6

```
e_write(Dev, Row, Col, To, Buf, Size) -> {ok, Ssize::integer()} | error

     Dev = epiphany_dev() | epiphany_mem()
     Row = integer()
     Col = integer()
     To = integer()
     Buf = binary()
     Size = integer()
```

This function writes data to a workgroup core local memory if Dev is of type `e_epiphany_t` or to an external memory buffer if Dev is of type `e_mem_t`. If successful, the number of bytes written is returned. If accessing the core local registers, use one of the register symbol macros in ehal.hrl as the To offset.

Data is passed to this function in the form of a binary. A binary can be created from common Erlang terms using such functions as atom_to_binary/2, float_to_binary/1, integer_to_binary/1 or list_to_binary/1. Use size/1 to get the size of a binary.

Since any character sequence within single quotes is an Erlang atom, atom_to_binary/2 is one method of encoding strings, for example atom_to_binary('Hello, World!', latin1)

See erlang(3) for more details and examples.

See Also

Epiphany SDK Reference

External Documentation Archive

init / 0

```
init() -> ok | error
```

Search for and load the NIF loadable library. This function is automatically called with the module is loaded. The library should be in the erlpiphany priv directory, which is searched first. If not found, check the default search paths for the library, and lastly, check the current working directory.
