



This section provides a comprehensive reference to the Nios[®] II hardware abstraction layer (HAL) application program interface (API) and the utilities, scripts, and settings that constitute the Nios II Software Build Tools. This section includes the following chapters:

- [Chapter 14, HAL API Reference](#)
- [Chapter 15, Nios II Software Build Tools Reference](#)
- [Appendix A, Using the Nios II Integrated Development Environment](#)

Introduction

This chapter provides an alphabetically ordered list of all the functions in the hardware abstraction layer (HAL) application program interface (API). Each function is listed with its C prototype and a short description. Each listing provides information about whether the function is thread-safe when running in a multi-threaded environment, and whether it can be called from an interrupt service routine (ISR).

This chapter only lists the functionality provided by the HAL. The complete newlib API is also available from within HAL systems. For example, newlib provides `printf()`, and other standard I/O functions, which are not described here.

-  Each function description lists the C header file that your code must include to access the function. Because header files include other header files, the function prototype might not be defined in the listed header file. However, you must include the listed header file in order to include all definitions on which the function depends.
-  For more details about the newlib API, refer to the newlib documentation. On the Windows **Start** menu, click **Programs > Altera > Nios II > Nios II Documentation**.

HAL API Functions

The HAL API functions are shown on the following pages.

`_exit()`

Prototype: `void _exit (int exit_code)`


Commonly called by: Newlib C library

Thread-safe: Yes.

Available from ISR: No.

Include: `<unistd.h>`

Description: The newlib `exit()` function calls the `_exit()` function to terminate the current process. Typically, `exit()` calls this function when `main()` completes. Because there is only a single process in HAL systems, the HAL implementation blocks forever.

 Interrupts are not disabled, so ISRs continue to execute.

The input argument, `exit_code`, is ignored.

Return: `-`

See also: Newlib documentation

`_rename()`

Prototype:	<code>int _rename(char *existing, char* new)</code>
Commonly called by:	Newlib C library
Thread-safe:	Yes.
Available from ISR:	Yes.
Include:	<code><stdio.h></code>
Description:	The <code>_rename()</code> function is provided for newlib compatibility.
Return:	It always fails with return code <code>-1</code> , and with <code>errno</code> set to <code>ENOSYS</code> .
See also:	Newlib documentation

alt_alarm_start()

Prototype:	<pre>int alt_alarm_start (alt_alarm* alarm, alt_u32 nticks, alt_u32 (*callback) (void* context), void* context)</pre>
Commonly called by:	C/C++ programs Device drivers
Thread-safe:	Yes.
Available from ISR:	Yes.
Include:	<sys/alt_alarm.h>
Description:	<p>The <code>alt_alarm_start()</code> function schedules an alarm callback. Refer to “Using Timer Devices” in the <i>Developing Programs Using the Hardware Abstraction Layer</i> chapter of the <i>Nios® II Software Developer’s Handbook</i>. The HAL waits <code>ntick</code> system clock ticks before calling the <code>callback()</code> function. When the HAL calls <code>callback()</code>, it passes it the input argument <code>context</code>.</p> <p>The <code>alarm</code> argument is a pointer to a structure that represents this alarm. You must create it, and it must have a lifetime that is at least as long as that of the alarm. However, you are not responsible for initializing the contents of the structure pointed to by <code>alarm</code>. This action is done by the call to <code>alt_alarm_start()</code>.</p>
Return:	<p>The return value for <code>alt_alarm_start()</code> is zero on success, and negative otherwise. This function fails if there is no system clock available.</p>
See also:	<pre>alt_alarm_stop() alt_nticks() alt_sysclk_init() alt_tick() alt_ticks_per_second() gettimeofday() settimeofday() times() usleep()</pre>

alt_alarm_stop()

Prototype: `void alt_alarm_stop (alt_alarm* alarm)`

Commonly called by: C/C++ programs
Device drivers

Thread-safe: Yes.

Available from ISR: Yes.

Include: `<sys/alt_alarm.h>`

Description: You can call the `alt_alarm_stop()` function to cancel an alarm previously registered by a call to `alt_alarm_start()`. The input argument is a pointer to the alarm structure in the previous call to `alt_alarm_start()`.

On return the alarm is canceled, if it is still active.

Return: —

[alt_alarm_start\(\)](#)

[alt_nticks\(\)](#)

[alt_sysclk_init\(\)](#)

[alt_tick\(\)](#)

See also: [alt_ticks_per_second\(\)](#)

[gettimeofday\(\)](#)

[settimeofday\(\)](#)

[times\(\)](#)

[usleep\(\)](#)

alt_dcache_flush()

Prototype:	<code>void alt_dcache_flush (void* start, alt_u32 len)</code>
Commonly called by:	C/C++ programs Device drivers
Thread-safe:	Yes.
Available from ISR:	Yes.
Include:	<code><sys/alt_cache.h></code>
Description:	The <code>alt_dcache_flush()</code> function flushes the data cache for a memory region of length <code>len</code> bytes, starting at address <code>start</code> . Flushing the cache consists of writing back dirty data and then invalidating the cache. In processors without data caches, it has no effect.
Return:	–
See also:	<code>alt_dcache_flush_all()</code> <code>alt_icache_flush()</code> <code>alt_icache_flush_all()</code> <code>alt_remap_cached()</code> <code>alt_remap_uncached()</code> <code>alt_uncached_free()</code> <code>alt_uncached_malloc()</code>

alt_dcache_flush_all()

Prototype: `void alt_dcache_flush_all (void)`

Commonly called by: C/C++ programs
Device drivers

Thread-safe: Yes.

Available from ISR: Yes.

Include: `<sys/alt_cache.h>`

Description: The `alt_dcache_flush_all()` function flushes, that is, writes back dirty data and then invalidates, the entire contents of the data cache.

In processors without data caches, it has no effect.

Return: —

[alt_dcache_flush\(\)](#)

[alt_icache_flush\(\)](#)

[alt_icache_flush_all\(\)](#)

See also: [alt_remap_cached\(\)](#)

[alt_remap_uncached\(\)](#)

[alt_uncached_free\(\)](#)

[alt_uncached_malloc\(\)](#)

alt_dev_reg()

Prototype: `int alt_dev_reg(alt_dev* dev)`

Commonly called by: Device drivers

Thread-safe: No.

Available from ISR: No.

Include: `<sys/alt_dev.h>`

The `alt_dev_reg()` function registers a device with the system. After it is registered, you can access a device using the standard I/O functions. Refer to the *Developing Programs Using the Hardware Abstraction Layer* chapter of the *Nios II Software Developer's Handbook*.

The system behavior is undefined in the event that a device is registered with a name that conflicts with an existing device or file system.

Description: The `alt_dev_reg()` function is not thread-safe in the sense that no other thread can use the device list at the time that `alt_dev_reg()` is called. Call `alt_dev_reg()` only in the following circumstances:

- When running in single-threaded mode.
- From a device initialization function called by `alt_sys_init()`. `alt_sys_init()` may only be called by the single-threaded C startup code.

Return: The return value is zero upon success. A negative return value indicates failure.

See also: [alt_fs_reg\(\)](#)

alt_dma_rxchan_close()

Prototype: `int alt_dma_rxchan_close (alt_dma_rxchan rxchan)`

Commonly called by: C/C++ programs
Device drivers

Thread-safe: Yes.

Available from ISR: No.

Include: `<sys/alt_dma.h>`

Description: The `alt_dma_rxchan_close()` function notifies the system that the application has finished using the direct memory access (DMA) receive channel, `rxchan`. The current implementation always succeeds.

Return: The return value is zero on success and negative otherwise.

`alt_dma_rxchan_depth()`
`alt_dma_rxchan_ioctl()`
`alt_dma_rxchan_open()`
`alt_dma_rxchan_prepare()`
`alt_dma_rxchan_reg()`

See also: `alt_dma_txchan_close()`
`alt_dma_txchan_ioctl()`
`alt_dma_txchan_open()`
`alt_dma_txchan_reg()`
`alt_dma_txchan_send()`
`alt_dma_txchan_space()`

alt_dma_rxchan_depth()

Prototype:	<code>alt_u32 alt_dma_rxchan_depth(alt_dma_rxchan dma)</code>
Commonly called by:	C/C++ programs Device drivers
Thread-safe:	Yes.
Available from ISR:	No.
Include:	<code><sys/alt_dma.h></code> The <code>alt_dma_rxchan_depth()</code> function returns the maximum number of receive requests that can be posted to the specified DMA transmit channel, <code>dma</code> .
Description:	Whether this function is thread-safe, or can be called from an ISR, depends on the underlying device driver. In general it safest to assume that it is not thread-safe.
Return:	Returns the maximum number of receive requests that can be posted. <code>alt_dma_rxchan_close()</code> <code>alt_dma_rxchan_ioctl()</code> <code>alt_dma_rxchan_open()</code> <code>alt_dma_rxchan_prepare()</code> <code>alt_dma_rxchan_reg()</code>
See also:	<code>alt_dma_txchan_close()</code> <code>alt_dma_txchan_ioctl()</code> <code>alt_dma_txchan_open()</code> <code>alt_dma_txchan_reg()</code> <code>alt_dma_txchan_send()</code> <code>alt_dma_txchan_space()</code>

alt_dma_rxchan_ioctl()

Prototype: `int alt_dma_rxchan_ioctl (alt_dma_rxchan dma, int req, void* arg)`

Commonly called by: C/C++ programs
Device drivers

Thread-safe: See description.

Available from ISR: See description.

Include: `<sys/alt_dma.h>`

The `alt_dma_rxchan_ioctl()` function performs DMA I/O operations on the DMA receive channel, `dma`. The I/O operations are device specific. For example, some DMA drivers support options to control the width of the transfer operations. The input argument, `req`, is an enumeration of the requested operation; `arg` is an additional argument for the request. The interpretation of `arg` is request dependent.

Description: [Table 14-1](#) shows generic requests defined in `alt_dma.h`, which a DMA device might support. Whether a call to `alt_dma_rxchan_ioctl()` is thread-safe, or can be called from an ISR, is device dependent. In general it safest to assume that it is not thread-safe.

Do not call the `alt_dma_rxchan_ioctl()` function while DMA transfers are pending, or unpredictable behavior could result.

For device-specific information about the Altera® DMA controller core, refer to the [DMA Controller Core](#) chapter in the *Embedded Peripherals IP User Guide*.

Return: A negative return value indicates failure. The interpretation of nonnegative return values is request specific.

`alt_dma_rxchan_close()`
`alt_dma_rxchan_depth()`
`alt_dma_rxchan_open()`
`alt_dma_rxchan_prepare()`
`alt_dma_rxchan_reg()`

See also: `alt_dma_txchan_close()`
`alt_dma_txchan_ioctl()`
`alt_dma_txchan_open()`
`alt_dma_txchan_reg()`
`alt_dma_txchan_send()`
`alt_dma_txchan_space()`

Table 14-1. Generic Requests

Request	Meaning
ALT_DMA_SET_MODE_8	Transfer data in units of 8 bits. The value of <code>arg</code> is ignored.
ALT_DMA_SET_MODE_16	Transfer data in units of 16 bits. The value of <code>arg</code> is ignored.
ALT_DMA_SET_MODE_32	Transfer data in units of 32 bits. The value of <code>arg</code> is ignored.
ALT_DMA_SET_MODE_64	Transfer data in units of 64 bits. The value of <code>arg</code> is ignored.
ALT_DMA_SET_MODE_128	Transfer data in units of 128 bits. The value of <code>arg</code> is ignored.
ALT_DMA_GET_MODE	Return the transfer width. The value of <code>arg</code> is ignored.

Table 14-1. Generic Requests

Request	Meaning
ALT_DMA_TX_ONLY_ON	The ALT_DMA_TX_ONLY_ON request causes a DMA channel to operate in a mode in which only the transmitter is under software control. The other side writes continuously from a single location. The address to which to write is the argument to this request.
ALT_DMA_TX_ONLY_OFF	Return to the default mode, in which both the receive and transmit sides of the DMA can be under software control.
ALT_DMA_RX_ONLY_ON	The ALT_DMA_RX_ONLY_ON request causes a DMA channel to operate in a mode in which only the receiver is under software control. The other side reads continuously from a single location. The address to read is the argument to this request.
ALT_DMA_RX_ONLY_OFF	Return to the default mode, in which both the receive and transmit sides of the DMA can be under software control.

alt_dma_rxchan_open()

Prototype:	<code>alt_dma_rxchan alt_dma_rxchan_open (const char* name)</code>
Commonly called by:	C/C++ programs Device drivers
Thread-safe:	Yes.
Available from ISR:	No.
Include:	<code><sys/alt_dma.h></code>
Description:	The <code>alt_dma_rxchan_open()</code> function obtains an <code>alt_dma_rxchan</code> descriptor for a DMA receive channel. The input argument, <code>name</code> , is the name of the associated physical device, for example, <code>/dev/dma_0</code> .
Return:	The return value is null on failure and non-null otherwise. If an error occurs, <code>errno</code> is set to <code>ENODEV</code> .
See also:	<code>alt_dma_rxchan_close()</code> <code>alt_dma_rxchan_depth()</code> <code>alt_dma_rxchan_ioctl()</code> <code>alt_dma_rxchan_prepare()</code> <code>alt_dma_rxchan_reg()</code> <code>alt_dma_txchan_close()</code> <code>alt_dma_txchan_ioctl()</code> <code>alt_dma_txchan_open()</code> <code>alt_dma_txchan_reg()</code> <code>alt_dma_txchan_send()</code> <code>alt_dma_txchan_space()</code>

alt_dma_rxchan_prepare()

```

int alt_dma_rxchan_prepare (alt_dma_rxchan  dma,
                           void*          data,
                           alt_u32        length,
                           alt_rxchan_done* done,
                           void*          handle)

```

Commonly called by: C/C++ programs
Device drivers

Thread-safe: See description.

Available from ISR: See description.

Include: <sys/alt_dma.h>

Description: The `alt_dma_rxchan_prepare()` posts a receive request to a DMA receive channel. The input arguments are: `dma`, the channel to use; `data`, a pointer to the location that data is to be received to; `length`, the maximum length of the data to receive in bytes; `done`, callback function that is called after the data is received; `handle`, an opaque value passed to `done`.

Whether this function is thread-safe, or can be called from an ISR, depends on the underlying device driver. In general it safest to assume that it is not thread-safe.

Return: The return value is zero upon success. A negative return value indicates that the request cannot be posted.

See also: `alt_dma_rxchan_close()`
`alt_dma_rxchan_depth()`
`alt_dma_rxchan_ioctl()`
`alt_dma_rxchan_open()`
`alt_dma_rxchan_reg()`
`alt_dma_txchan_close()`
`alt_dma_txchan_ioctl()`
`alt_dma_txchan_open()`
`alt_dma_txchan_reg()`
`alt_dma_txchan_send()`
`alt_dma_txchan_space()`

alt_dma_rxchan_reg()

Prototype: `int alt_dma_rxchan_reg (alt_dma_rxchan_dev* dev)`

Commonly called by: C/C++ programs
Device drivers

Thread-safe: No.

Available from ISR: No.

Include: `<sys/alt_dma_dev.h>`

The `alt_dma_rxchan_reg()` function registers a DMA receive channel with the system. After it is registered, a device can be accessed using the functions described in “Using DMA Devices” in the *Developing Programs Using the Hardware Abstraction Layer* chapter of the *Nios II Software Developer’s Handbook*.

System behavior is undefined in the event that a channel is registered with a name that conflicts with an existing channel.

Description: The `alt_dma_rxchan_reg()` function is not thread-safe if other threads are using the channel list at the time that `alt_dma_rxchan_reg()` is called. Call `alt_dma_rxchan_reg()` only in the following circumstances:

- When running in single-threaded mode.
- From a device initialization function called by `alt_sys_init()`. `alt_sys_init()` may only be called by the single-threaded C startup code.

Return: The return value is zero upon success. A negative return value indicates failure.

`alt_dma_rxchan_close()`
`alt_dma_rxchan_depth()`
`alt_dma_rxchan_ioctl()`
`alt_dma_rxchan_open()`
`alt_dma_rxchan_prepare()`

See also: `alt_dma_txchan_close()`
`alt_dma_txchan_ioctl()`
`alt_dma_txchan_open()`
`alt_dma_txchan_reg()`
`alt_dma_txchan_send()`
`alt_dma_txchan_space()`

alt_dma_txchan_close()

Prototype:	<code>int alt_dma_txchan_close (alt_dma_txchan txchan)</code>
Commonly called by:	C/C++ programs Device drivers
Thread-safe:	Yes.
Available from ISR:	No.
Include:	<code><sys/alt_dma.h></code>
Description:	The <code>alt_dma_txchan_close</code> function notifies the system that the application has finished using the DMA transmit channel, <code>txchan</code> . The current implementation always succeeds.
Return:	The return value is zero on success and negative otherwise.
See also:	<code>alt_dma_rxchan_close()</code> <code>alt_dma_rxchan_depth()</code> <code>alt_dma_rxchan_ioctl()</code> <code>alt_dma_rxchan_open()</code> <code>alt_dma_rxchan_prepare()</code> <code>alt_dma_rxchan_reg()</code> <code>alt_dma_txchan_ioctl()</code> <code>alt_dma_txchan_open()</code> <code>alt_dma_txchan_reg()</code> <code>alt_dma_txchan_send()</code> <code>alt_dma_txchan_space()</code>

alt_dma_txchan_ioctl()

Prototype: `int alt_dma_txchan_ioctl (alt_dma_txchan dma,
int req,
void* arg)`

Commonly called by: C/C++ programs
Device drivers

Thread-safe: See description.

Available from ISR: See description.

Include: `<sys/alt_dma.h>`

The `alt_dma_txchan_ioctl()` function performs device specific I/O operations on the DMA transmit channel, `dma`. For example, some drivers support options to control the width of the transfer operations. The input argument, `req`, is an enumeration of the requested operation; `arg` is an additional argument for the request. The interpretation of `arg` is request dependent.

Description: Refer to [Table 14-1 on page 14-11](#) for the generic requests a device might support.

Whether a call to `alt_dma_txchan_ioctl()` is thread-safe, or can be called from an ISR, is device dependent. In general it safest to assume that it is not thread-safe.

Do not call the `alt_dma_txchan_ioctl()` function while DMA transfers are pending, or unpredictable behavior could result.

Return: A negative return value indicates failure; otherwise the interpretation of the return value is request specific.

`alt_dma_rxchan_close()`
`alt_dma_rxchan_depth()`
`alt_dma_rxchan_ioctl()`
`alt_dma_rxchan_open()`
`alt_dma_rxchan_prepare()`

See also: `alt_dma_rxchan_reg()`
`alt_dma_txchan_close()`
`alt_dma_txchan_open()`
`alt_dma_txchan_reg()`
`alt_dma_txchan_send()`
`alt_dma_txchan_space()`

alt_dma_txchan_open()

Prototype:	<code>alt_dma_txchan alt_dma_txchan_open (const char* name)</code>
Commonly called by:	C/C++ programs Device drivers
Thread-safe:	Yes.
Available from ISR:	No.
Include:	<code><sys/alt_dma.h></code>
Description:	The <code>alt_dma_txchan_open()</code> function obtains an <code>alt_dma_txchan()</code> descriptor for a DMA transmit channel. The input argument, <code>name</code> , is the name of the associated physical device, for example, <code>/dev/dma_0</code> .
Return:	The return value is null on failure and non-null otherwise. If an error occurs, <code>errno</code> is set to <code>ENODEV</code> .
See also:	<code>alt_dma_rxchan_close()</code> <code>alt_dma_rxchan_depth()</code> <code>alt_dma_rxchan_ioctl()</code> <code>alt_dma_rxchan_open()</code> <code>alt_dma_rxchan_prepare()</code> <code>alt_dma_rxchan_reg()</code> <code>alt_dma_txchan_close()</code> <code>alt_dma_txchan_ioctl()</code> <code>alt_dma_txchan_reg()</code> <code>alt_dma_txchan_send()</code> <code>alt_dma_txchan_space()</code>

alt_dma_txchan_reg()

Prototype: `int alt_dma_txchan_reg (alt_dma_txchan_dev* dev)`

Commonly called by: C/C++ programs
Device drivers

Thread-safe: No.

Available from ISR: No.

Include: `<sys/alt_dma_dev.h>`

The `alt_dma_txchan_reg()` function registers a DMA transmit channel with the system. After it is registered, a device can be accessed using the functions described in “Using DMA Devices” in the *Developing Programs Using the Hardware Abstraction Layer* chapter of the *Nios II Software Developer’s Handbook*.

System behavior is undefined in the event that a channel is registered with a name that conflicts with an existing channel.

Description: The `alt_dma_txchan_reg()` function is not thread-safe if other threads are using the channel list at the time that `alt_dma_txchan_reg()` is called. Call `alt_dma_txchan_reg()` only in the following circumstances:

- When running in single-threaded mode.
- From a device initialization function called by `alt_sys_init()`. `alt_sys_init()` may only be called by the single-threaded C startup code.

Return: The return value is zero upon success. A negative return value indicates failure.

`alt_dma_rxchan_close()`
`alt_dma_rxchan_depth()`
`alt_dma_rxchan_ioctl()`
`alt_dma_rxchan_open()`
`alt_dma_rxchan_prepare()`

See also: `alt_dma_rxchan_reg()`
`alt_dma_txchan_close()`
`alt_dma_txchan_ioctl()`
`alt_dma_txchan_open()`
`alt_dma_txchan_send()`
`alt_dma_txchan_space()`

alt_dma_txchan_send()

Prototype:	<pre>int alt_dma_txchan_send (alt_dma_txchan dma, const void* from, alt_u32 length, alt_txchan_done* done, void* handle)</pre>
Commonly called by:	C/C++ programs Device drivers
Thread-safe:	See description.
Available from ISR:	See description.
Include:	<sys/alt_dma.h>
Description:	<p>The <code>alt_dma_txchan_send()</code> function posts a transmit request to a DMA transmit channel. The input arguments are: <code>dma</code>, the channel to use; <code>from</code>, a pointer to the start of the data to send; <code>length</code>, the length of the data to send in bytes; <code>done</code>, a callback function that is called after the data is sent; and <code>handle</code>, an opaque value passed to <code>done</code>.</p> <p>Whether this function is thread-safe, or can be called from an ISR, depends on the underlying device driver. In general it safest to assume that it is not thread-safe.</p>
Return:	The return value is negative if the request cannot be posted, and zero otherwise.
See also:	<pre>alt_dma_rxchan_close() alt_dma_rxchan_depth() alt_dma_rxchan_ioctl() alt_dma_rxchan_open() alt_dma_rxchan_prepare() alt_dma_rxchan_reg() alt_dma_txchan_close() alt_dma_txchan_ioctl() alt_dma_txchan_open() alt_dma_txchan_reg() alt_dma_txchan_space()</pre>

alt_dma_txchan_space()

Prototype: `int alt_dma_txchan_space (alt_dma_txchan dma)`

Commonly called by: C/C++ programs
Device drivers

Thread-safe: See description.

Available from ISR: See description.

Include: `<sys/alt_dma.h>`

Description: The `alt_dma_txchan_space()` function returns the number of transmit requests that can be posted to the specified DMA transmit channel, `dma`. A negative value indicates that the value cannot be determined.

Whether this function is thread-safe, or can be called from an ISR, depends on the underlying device driver. In general it safest to assume that it is not thread-safe.

Return: Returns the number of transmit requests that can be posted.

`alt_dma_rxchan_close()`
`alt_dma_rxchan_depth()`
`alt_dma_rxchan_ioctl()`
`alt_dma_rxchan_open()`
`alt_dma_rxchan_prepare()`
See also: `alt_dma_rxchan_reg()`
`alt_dma_txchan_close()`
`alt_dma_txchan_ioctl()`
`alt_dma_txchan_open()`
`alt_dma_txchan_reg()`
`alt_dma_txchan_send()`

alt_erase_flash_block()

Prototype: `int alt_erase_flash_block(alt_flash_fd* fd,
int offset,
int length)`

Commonly called by: C/C++ programs
Device drivers

Thread-safe: No.

Available from ISR: No.

Include: `<sys/alt_flash.h>`

Description: The `alt_erase_flash_block()` function erases an individual flash erase block. The parameter `fd` specifies the flash device; `offset` is the offset within the flash of the block to erase; `length` is the size of the block to erase. No error checking is performed to check that this is a valid block, or that the length is correct. Refer to “Using Flash Devices” in the *Developing Programs Using the Hardware Abstraction Layer* chapter of the *Nios II Software Developer’s Handbook*.

Call the `alt_erase_flash_block()` function only when operating in single-threaded mode.

The only valid values for the `fd` parameter are those returned from the `alt_flash_open_dev` function. If any other value is passed, the behavior of this function is undefined.

Return: The return value is zero upon success. A negative return value indicates failure.

See also: [alt_flash_close_dev\(\)](#)
[alt_flash_open_dev\(\)](#)
[alt_get_flash_info\(\)](#)
[alt_read_flash\(\)](#)
[alt_write_flash\(\)](#)
[alt_write_flash_block\(\)](#)

alt_exception_cause_generated_bad_addr()

Prototype: `int alt_exception_cause_generated_bad_addr
(alt_exception_cause cause)`

Commonly called by: Instruction-related exception handlers

Thread-safe:

Available from ISR:

Include: `<sys/alt_exceptions.h>`

This function validates the `bad_addr` argument to an instruction-related exception handler. The function parses the handler's `cause` argument to determine whether the `bad_addr` register contains the exception-causing address.

Description: If the exception is of a type that generates a valid address in `bad_addr`, this function returns a nonzero value. Otherwise, it returns zero.

If the `cause` register is unimplemented in the Nios II processor core, this function always returns zero.

Return: A nonzero value means `bad_addr` contains the exception-causing address.

Zero means the value of `bad_addr` is to be ignored.

See also: [alt_instruction_exception_register\(\)](#)

alt_flash_close_dev()

Prototype: `void alt_flash_close_dev(alt_flash_fd* fd)`

Commonly called by: C/C++ programs
Device drivers

Thread-safe: No.

Available from ISR: No.

Include: `<sys/alt_flash.h>`

Description: The `alt_flash_close_dev()` function closes a flash device. All subsequent calls to `alt_write_flash()`, `alt_read_flash()`, `alt_get_flash_info()`, `alt_erase_flash_block()`, or `alt_write_flash_block()` for this flash device fail.

Call the `alt_flash_close_dev()` function only when operating in single-threaded mode.

The only valid values for the `fd` parameter are those returned from the `alt_flash_open_dev` function. If any other value is passed, the behavior of this function is undefined.

Return: —

[alt_erase_flash_block\(\)](#)

[alt_flash_open_dev\(\)](#)

[alt_get_flash_info\(\)](#)

See also:

[alt_read_flash\(\)](#)

[alt_write_flash\(\)](#)

[alt_write_flash_block\(\)](#)

alt_flash_open_dev()

Prototype: `alt_flash_fd* alt_flash_open_dev(const char* name)`

Commonly called by: C/C++ programs
Device drivers

Thread-safe: No.

Available from ISR: No.

Include: `<sys/alt_flash.h>`

The `alt_flash_open_dev()` function opens a flash device. After it is opened, you can perform the following operations:

Description:

- Write to a flash device using `alt_write_flash()`
- Read from a flash device using `alt_read_flash()`
- Control individual flash blocks using `alt_get_flash_info()`, `alt_erase_flash_block()`, or `alt_write_flash_block()`.

Call the `alt_flash_open_dev` function only when operating in single-threaded mode.

Return: The return value is zero upon failure. Any other value indicates success.

See also:

- `alt_erase_flash_block()`
- `alt_flash_close_dev()`
- `alt_get_flash_info()`
- `alt_read_flash()`
- `alt_write_flash()`
- `alt_write_flash_block()`

alt_fs_reg()

Prototype: `int alt_fs_reg (alt_dev* dev)`

Commonly called by: Device drivers

Thread-safe: No.

Available from ISR: No.

Include: `<sys/alt_dev.h>`

The `alt_fs_reg()` function registers a file system with the HAL. After it is registered, a file system can be accessed using the standard I/O functions. Refer to the *Developing Programs Using the Hardware Abstraction Layer* chapter of the *Nios II Software Developer's Handbook*.

System behavior is undefined in the event that a file system is registered with a name that conflicts with an existing device or file system.

Description:

`alt_fs_reg()` is not thread-safe if other threads are using the device list at the time that `alt_fs_reg()` is called. Call `alt_fs_reg()` only in the following circumstances:

- When running in single-threaded mode.
- From a device initialization function called by `alt_sys_init()`. `alt_sys_init()` may only be called by the single-threaded C startup code.

Return: The return value is zero upon success. A negative return value indicates failure.

See also: [alt_dev_reg\(\)](#)

alt_get_flash_info()

Prototype:

```
int alt_get_flash_info(alt_flash_fd* fd,
                      flash_region** info,
                      int*          number_of_regions)
```

Commonly called by: C/C++ programs
Device drivers

Thread-safe: No.

Available from ISR: No.

Include: `<sys/alt_flash.h>`

Description: The `alt_get_flash_info()` function gets the details of the erase region of a flash part. The flash part is specified by the descriptor `fd`, a pointer to the start of the `flash_region` structures is returned in the `info` parameter, and the number of flash regions are returned in `number_of_regions`.

Call this function only when operating in single-threaded mode.

The only valid values for the `fd` parameter are those returned from the `alt_flash_open_dev` function. If any other value is passed, the behavior of this function is undefined.

Return: The return value is zero upon success. A negative return value indicates failure.

See also: [alt_erase_flash_block\(\)](#)
[alt_flash_close_dev\(\)](#)
[alt_flash_open_dev\(\)](#)
[alt_read_flash\(\)](#)
[alt_write_flash\(\)](#)
[alt_write_flash_block\(\)](#)

alt_ic_irq_disable()

Prototype: `int alt_ic_irq_disable (alt_u32 ic_id, alt_u32 irq)`

Commonly called by: C/C++ programs
Device drivers

Thread-safe: Yes.

Available from ISR: No.

Include: `<sys/alt_irq.h>`

The `alt_ic_irq_disable()` function disables a single interrupt.

The function arguments are as follows:

- Description:
- `ic_id` is the interrupt controller identifier (ID) as defined in **system.h**, identifying the external interrupt controller in the daisy chain. This argument is ignored if the external interrupt controller interface is not implemented.
 - `irq` is the interrupt request (IRQ) number, as defined in `system.h`, identifying the interrupt to enable.
 - A driver for an external interrupt controller (EIC) must implement this function.

Return: This function returns zero if successful, or nonzero otherwise. The function fails if the `irq` parameter is greater than the maximum interrupt port number supported by the external interrupt controller.

[alt_irq_disable_all\(\)](#)

[alt_irq_enable\(\)](#)

[alt_irq_enable_all\(\)](#)

[alt_irq_enabled\(\)](#)

See also:

[alt_irq_register\(\)](#)

[alt_irq_disable\(\)](#)

[alt_ic_irq_enable\(\)](#)

[alt_ic_irq_enabled\(\)](#)

[alt_ic_isr_register\(\)](#)

alt_ic_irq_enable()

Prototype: `int alt_ic_irq_enable (alt_u32 ic_id, alt_u32 irq)`

Commonly called by: C/C++ programs
Device drivers

Thread-safe: Yes.

Available from ISR: No.

Include: `<sys/alt_irq.h>`

The `alt_ic_irq_enable()` function enables a single interrupt.

The function arguments are as follows:

- Description:
- `ic_id` is the interrupt controller ID as defined in **system.h**, identifying the external interrupt controller in the daisy chain. This argument is ignored if the external interrupt controller interface is not implemented.
 - `irq` is the IRQ number, as defined in **system.h**, identifying the interrupt to enable.
 - A driver for an EIC must implement this function.

Return: This function returns zero if successful, or nonzero otherwise. The function fails if the `irq` parameter is greater than the maximum interrupt port number supported by the external interrupt controller.

See also: `alt_irq_disable()`
`alt_irq_disable_all()`
`alt_irq_enable_all()`
`alt_irq_enabled()`
`alt_irq_register()`
`alt_irq_enable()`
`alt_ic_irq_disable()`
`alt_ic_irq_enabled()`
`alt_ic_isr_register()`

alt_ic_irq_enabled()

Prototype: `int alt_ic_irq_enabled (alt_u32 ic_id, alt_u32 irq)`

Commonly called by: Device drivers

Thread-safe: Yes.

Available from ISR: Yes.

Include: `<sys/alt_irq.h>`

This function determines whether a specified interrupt is enabled.

The function arguments are as follows:

- Description:
- `ic_id` is the interrupt controller ID as defined in **system.h**, identifying the external interrupt controller in the daisy chain. This argument is ignored if the external interrupt controller interface is not implemented.
 - `irq` is the IRQ number, as defined in **system.h**, identifying the interrupt to enable.
 - A driver for an EIC must implement this function.

Return: Returns zero if the specified interrupt is disabled, and nonzero otherwise.

[alt_irq_disable\(\)](#)

[alt_irq_disable_all\(\)](#)

[alt_irq_enable\(\)](#)

[alt_irq_enable_all\(\)](#)

See also:

[alt_irq_register\(\)](#)

[alt_irq_enabled\(\)](#)

[alt_ic_irq_disable\(\)](#)

[alt_ic_irq_enable\(\)](#)

[alt_ic_isr_register\(\)](#)

alt_ic_isr_register()

Prototype:

```
int alt_ic_isr_register (alt_u32 ic_id,
                       alt_u32 irq,
                       alt_isr_func isr,
                       void*   isr_context,
                       void*   flags)
```

Commonly called by: Device drivers

Thread-safe: Yes.

Available from ISR: No.

Include: <sys/alt_irq.h>

The `alt_ic_isr_register()` function registers an ISR. If the function is successful, the requested interrupt is enabled on return, and `isr` and `isr_context` are inserted in the vector table.

The function arguments are as follows:

- `ic_id` is the interrupt controller ID as defined in **system.h**, identifying the external interrupt controller in the daisy chain. This argument is ignored if the external interrupt controller interface is not implemented.
- `irq` is the IRQ number, as defined in **system.h**, identifying the interrupt to register.
- `isr` is the function that is called when the interrupt is accepted.
- `isr_context` is the input argument to `isr`. `isr_context` points to a data structure associated with the device driver instance.
- `flags` is reserved.

Description:

The ISR function prototype is defined as follows:

```
typedef void (*alt_isr_func) (void* isr_context);
```

Calls to `alt_ic_isr_register()` replace previously registered handlers for interrupt `irq`.

If `isr` is set to null, the interrupt is disabled.

- A driver for an EIC must implement this function.

Return:

This function returns zero if successful, or nonzero otherwise. The function fails if the `irq` parameter is greater than the maximum interrupt port number supported by the external interrupt controller.

See also:

```
alt_irq_disable()
alt_irq_disable_all()
alt_irq_enable()
alt_irq_enable_all()
alt_irq_enabled()
alt_irq_register()
alt_ic_irq_disable()
alt_ic_irq_enable()
alt_ic_irq_enabled()
```

alt_icache_flush()

Prototype:	<code>void alt_icache_flush (void* start, alt_u32 len)</code>
Commonly called by:	C/C++ programs Device drivers
Thread-safe:	Yes.
Available from ISR:	Yes.
Include:	<code><sys/alt_cache.h></code>
Description:	The <code>alt_icache_flush()</code> function invalidates the instruction cache for a memory region of length <code>len</code> bytes, starting at address <code>start</code> . In processors without instruction caches, it has no effect.
Return:	— alt_dcache_flush() alt_dcache_flush_all() alt_icache_flush_all()
See also:	alt_remap_cached() alt_remap_uncached() alt_uncached_free() alt_uncached_malloc()

alt_icache_flush_all()

Prototype:	<code>void alt_icache_flush_all (void)</code>
Commonly called by:	C/C++ programs Device drivers
Thread-safe:	Yes.
Available from ISR:	Yes.
Include:	<code><sys/alt_cache.h></code>
Description:	The <code>alt_icache_flush_all()</code> function invalidates the entire contents of the instruction cache. In processors without instruction caches, it has no effect.
Return:	— <code>alt_dcache_flush()</code> <code>alt_dcache_flush_all()</code> <code>alt_icache_flush()</code>
See also:	<code>alt_remap_cached()</code> <code>alt_remap_uncached()</code> <code>alt_uncached_free()</code> <code>alt_uncached_malloc()</code>

alt_instruction_exception_register()

```
void alt_instruction_exception_register (
    alt_exception_result (*handler)
    ( alt_exception_cause cause,
      alt_u32             exception_pc,
      alt_u32             bad_addr ))
```

Prototype:

Commonly called by: C/C++ programs

Device drivers

Thread-safe: No.

Available from ISR: Yes.

Include: <sys/alt_exceptions.h>

The HAL API function `alt_instruction_exception_register()` registers an instruction-related exception handler. The `handler` argument is a pointer to the instruction-related exception handler.

You can only use this API function if you have enabled the `hal.enable_instruction_related_exceptions_api` setting in the board support package (BSP). For details, refer to “Settings” in the *Nios II Software Build Tools Reference* chapter of the *Nios II Software Developer’s Handbook*.

Description: Register the instruction-related exception handler as early as possible in function `main()`. This allows you to handle abnormal conditions during startup.

You can register an exception handler from the `alt_main()` function.

A call to `alt_instruction_exception_register()` replaces the previously registered exception handler, if any. If `handler` is set to null, the instruction-related exception handler is removed.

For further usage details, refer to the *Exception Handling* chapter of the *Nios II Software Developer’s Handbook*.

Return: —

See also: [alt_irq_register\(\)](#)
[alt_exception_cause_generated_bad_addr\(\)](#)

alt_irq_disable()

Prototype: `int alt_irq_disable (alt_u32 id)`


Commonly called by: C/C++ programs
Device drivers


Thread-safe: Yes.

Available from ISR: No.

Include: `<sys/alt_irq.h>`

The `alt_irq_disable()` function disables a single interrupt.

Description:  This function is part of the legacy HAL interrupt API, which is deprecated. Altera recommends using the enhanced HAL interrupt API.

 For details about using the enhanced HAL interrupt API, refer to “Interrupt Service Routines” in the *Exception Handling* chapter of the *Nios II Software Developer's Handbook*.

Return: The return value is zero.

[alt_irq_disable_all\(\)](#)

[alt_irq_enable\(\)](#)

[alt_irq_enable_all\(\)](#)

[alt_irq_enabled\(\)](#)

See also:

[alt_irq_register\(\)](#)

[alt_ic_irq_disable\(\)](#)

[alt_ic_irq_enable\(\)](#)

[alt_ic_irq_enabled\(\)](#)

[alt_ic_isr_register\(\)](#)

alt_irq_disable_all()

Prototype:	<code>alt_irq_context alt_irq_disable_all (void)</code>
Commonly called by:	C/C++ programs Device drivers
Thread-safe:	Yes.
Available from ISR:	No.
Include:	<code><sys/alt_irq.h></code>
Description:	The <code>alt_irq_disable_all()</code> function disables all maskable interrupts. Nonmaskable interrupts (NMIs) are unaffected.
Return:	Pass the return value as the input argument to a subsequent call to <code>alt_irq_enable_all()</code> . <code>alt_irq_disable()</code> <code>alt_irq_enable()</code> <code>alt_irq_enable_all()</code> <code>alt_irq_enabled()</code>
See also:	<code>alt_irq_register()</code> <code>alt_ic_irq_disable()</code> <code>alt_ic_irq_enable()</code> <code>alt_ic_irq_enabled()</code> <code>alt_ic_isr_register()</code>

alt_irq_enable()

Prototype:	<code>int alt_irq_enable (alt_u32 id)</code>
Commonly called by:	C/C++ programs Device drivers
Thread-safe:	Yes.
Available from ISR:	No.
Include:	<code><sys/alt_irq.h></code>
Description:	The <code>alt_irq_enable()</code> function enables a single interrupt.
Return:	The return value is zero.
See also:	<code>alt_irq_disable()</code> <code>alt_irq_disable_all()</code> <code>alt_irq_enable_all()</code> <code>alt_irq_enabled()</code> <code>alt_irq_register()</code> <code>alt_ic_irq_disable()</code> <code>alt_ic_irq_enable()</code> <code>alt_ic_irq_enabled()</code> <code>alt_ic_isr_register()</code>

alt_irq_enable_all()

Prototype: `void alt_irq_enable_all (alt_irq_context context)`

Commonly called by: C/C++ programs
Device drivers

Thread-safe: Yes.

Available from ISR: Yes.

Include: `<sys/alt_irq.h>`

Description: The `alt_irq_enable_all()` function enables all interrupts that were previously disabled by `alt_irq_disable_all()`. The input argument, `context`, is the value returned by a previous call to `alt_irq_disable_all()`. Using `context` allows nested calls to `alt_irq_disable_all()` and `alt_irq_enable_all()`. As a result, `alt_irq_enable_all()` does not necessarily enable all interrupts, such as interrupts explicitly disabled by `alt_irq_disable()`.

Return: —

`alt_irq_disable()`
`alt_irq_disable_all()`
`alt_irq_enable()`
`alt_irq_enabled()`

See also: `alt_irq_register()`
`alt_ic_irq_disable()`
`alt_ic_irq_enable()`
`alt_ic_irq_enabled()`
`alt_ic_isr_register()`

alt_irq_enabled()

Prototype: `int alt_irq_enabled (void)`


Commonly called by: Device drivers


Thread-safe: Yes.

Available from ISR: Yes.

Include: `<sys/alt_irq.h>`

Determines whether maskable exceptions (`status.PIE`) are enabled.

Description:  This function is part of the legacy HAL interrupt API, which is deprecated. Altera recommends using the enhanced HAL interrupt API.



 For details about using the enhanced HAL interrupt API, refer to “Interrupt Service Routines” in the *Exception Handling* chapter of the *Nios II Software Developer’s Handbook*.

Return: Returns zero if interrupts are disabled, and non-zero otherwise.

`alt_irq_disable()`
`alt_irq_disable_all()`
`alt_irq_enable()`
`alt_irq_enable_all()`
`alt_irq_register()`
`alt_ic_irq_disable()`
`alt_ic_irq_enable()`
`alt_ic_irq_enabled()`
`alt_ic_isr_register()`

See also:

alt_irq_register()

Prototype:	<pre>int alt_irq_register (alt_u32 id, void* context, void (*isr)(void*, alt_u32))</pre>
Commonly called by:	Device drivers
Thread-safe:	Yes.
Available from ISR:	No.
Include:	<pre><sys/alt_irq.h></pre> <p>The <code>alt_irq_register()</code> function registers an ISR. If the function is successful, the requested interrupt is enabled on return.</p> <p>The input argument <code>id</code> is the interrupt to enable. <code>isr</code> is the function that is called when the interrupt is active. <code>context</code> and <code>id</code> are the two input arguments to <code>isr</code>.</p>
Description:	<p>Calls to <code>alt_irq_register()</code> replace previously registered handlers for interrupt <code>id</code>. If <code>irq_handler</code> is set to null, the interrupt is disabled.</p> <p> This function is part of the legacy HAL interrupt API, which is deprecated. Altera recommends using the enhanced HAL interrupt API.</p> <p> For details about using the enhanced HAL interrupt API, refer to “Interrupt Service Routines” in the <i>Exception Handling</i> chapter of the <i>Nios II Software Developer’s Handbook</i>.</p>
Return:	<p>The <code>alt_irq_register()</code> function returns zero if successful, or non-zero otherwise.</p> <p><code>alt_irq_disable()</code> <code>alt_irq_disable_all()</code> <code>alt_irq_enable()</code> <code>alt_irq_enable_all()</code></p>
See also:	<p><code>alt_irq_enabled()</code> <code>alt_ic_irq_disable()</code> <code>alt_ic_irq_enable()</code> <code>alt_ic_irq_enabled()</code> <code>alt_ic_isr_register()</code></p>

alt_llist_insert()

Prototype: `void alt_llist_insert(alt_llist* list,
alt_llist* entry)`

Commonly called by: C/C++ programs
Device drivers

Thread-safe: No.

Available from ISR: Yes.

Include: `<sys/alt_llist.h>`

Description: The `alt_llist_insert()` function inserts the doubly linked list entry `entry` in the list `list`. This operation is not reentrant. For example, if a list can be manipulated from different threads, or from within both application code and an ISR, some mechanism is required to protect access to the list. Interrupts can be locked, or in MicroC/OS-II, a `mutex` can be used.

Return: `-`

See also: [alt_llist_remove\(\)](#)

alt_llist_remove()

Prototype: `void alt_llist_remove(alt_llist* entry)`

Commonly called by: C/C++ programs

Device drivers

Thread-safe: No.

Available from ISR: Yes.

Include: `<sys/alt_llist.h>`

Description: The `alt_llist_remove()` function removes the doubly linked list entry `entry` from the list it is currently a member of. This operation is not reentrant. For example if a list can be manipulated from different threads, or from within both application code and an ISR, some mechanism is required to protect access to the list. Interrupts can be locked, or in MicroC/OS-II, a `mutex` can be used.

Return: —

See also: [alt_llist_insert\(\)](#)

alt_load_section()

Prototype: `void alt_load_section(alt_u32* from,
alt_u32* to,
alt_u32* end)`

Commonly called by: C/C++ programs

Thread-safe: No.

Available from ISR: No.

Include: `<sys/alt_load.h>`

When operating in run-from-flash mode, the sections `.exceptions`, `.rodata`, and `.rwddata` are automatically loaded from the boot device to RAM at boot time. However, if there are any additional sections that require loading, the `alt_load_section()` function loads them manually before these sections are used.

Description: The input argument `from` is the start address in the boot device of the section; `to` is the start address in RAM of the section, and `end` is the end address in RAM of the section.

To load one of the additional memory sections provided by the default linker script, use the macro `ALT_LOAD_SECTION_BY_NAME` rather than calling `alt_load_section()` directly. For example, to load the section `.onchip_ram`, use the following code:

```
ALT_LOAD_SECTION_BY_NAME(onchip_ram);
```

The leading `'.'` is omitted in the section name. This macro is defined in the header `sys/alt_load.h`.

Return: `-`

See also: `-`

alt_nticks()

Prototype:	alt_u32 alt_nticks (void)
Commonly called by:	C/C++ programs Device drivers
Thread-safe:	Yes.
Available from ISR:	Yes.
Include:	<sys/alt_alarm.h>
Description:	The alt_nticks() function.
Return:	Returns the number of elapsed system clock tick since reset. It returns zero if there is no system clock available. alt_alarm_start() alt_alarm_stop() alt_sysclk_init() alt_tick()
See also:	alt_ticks_per_second() gettimeofday() settimeofday() times() usleep()

alt_read_flash()

Prototype:	<pre>int alt_read_flash(alt_flash_fd* fd, int offset, void* dest_addr, int length)</pre>
Commonly called by:	C/C++ programs Device drivers
Thread-safe:	No.
Available from ISR:	No.
Include:	<sys/alt_flash.h> The <code>alt_read_flash()</code> function reads data from flash. <code>length</code> bytes are read from the flash <code>fd</code> , starting <code>offset</code> bytes from the beginning of the flash and are written to the location <code>dest_addr</code> .
Description:	Call this function only when operating in single-threaded mode. The only valid values for the <code>fd</code> parameter are those returned from the <code>alt_flash_open_dev</code> function. If any other value is passed, the behavior of this function is undefined.
Return:	The return value is zero on success and nonzero otherwise.
See also:	alt_erase_flash_block() alt_flash_close_dev() alt_flash_open_dev() alt_get_flash_info() alt_write_flash() alt_write_flash_block()

alt_remap_cached()

Prototype:	<pre>void* alt_remap_cached (volatile void* ptr, alt_u32 len);</pre>
Commonly called by:	C/C++ programs Device drivers
Thread-safe:	Yes.
Available from ISR:	No.
Include:	<code><sys/alt_cache.h></code>
Description:	The <code>alt_remap_cached()</code> function remaps a region of memory for cached access. The memory to map is <code>len</code> bytes, starting at address <code>ptr</code> . Processors that do not have a data cache return uncached memory.
Return:	The return value for this function is the remapped memory region.
See also:	alt_dcache_flush() alt_dcache_flush_all() alt_icache_flush() alt_icache_flush_all() alt_remap_uncached() alt_uncached_free() alt_uncached_malloc()

alt_remap_uncached()

Prototype:	<pre>volatile void* alt_remap_uncached (void* ptr, alt_u32 len);</pre>
Commonly called by:	C/C++ programs Device drivers
Thread-safe:	Yes.
Available from ISR:	No.
Include:	<code><sys/alt_cache.h></code>
Description:	The <code>alt_remap_uncached()</code> function remaps a region of memory for uncached access. The memory to map is <code>len</code> bytes, starting at address <code>ptr</code> . Processors that do not have a data cache return uncached memory.
Return:	The return value for this function is the remapped memory region.
See also:	<code>alt_dcache_flush()</code> <code>alt_dcache_flush_all()</code> <code>alt_icache_flush()</code> <code>alt_icache_flush_all()</code> <code>alt_remap_cached()</code> <code>alt_uncached_free()</code> <code>alt_uncached_malloc()</code>

alt_sysclk_init()

Prototype: `int alt_sysclk_init (alt_u32 nticks)`

Commonly called by: Device drivers

Thread-safe: No.

Available from ISR: No.

Include: `<sys/alt_alarm.h>`

The `alt_sysclk_init()` function registers the presence of a system clock driver. The input argument is the number of ticks per second at which the system clock is run.

Description: The expectation is that this function is only called from within `alt_sys_init()`, that is, while the system is running in single-threaded mode. Concurrent calls to this function might lead to unpredictable results.

Return: This function returns zero on success; otherwise it returns a negative value. The call can fail if a system clock driver is already registered, or if no system clock device is available.

[alt_alarm_start\(\)](#)

[alt_alarm_stop\(\)](#)

[alt_nticks\(\)](#)

[alt_tick\(\)](#)

See also: [alt_ticks_per_second\(\)](#)

[gettimeofday\(\)](#)

[settimeofday\(\)](#)

[times\(\)](#)

[usleep\(\)](#)

alt_tick()

Prototype: `void alt_tick (void)`

Commonly called by: Device drivers

Thread-safe: No.

Available from ISR: Yes.

Include: `<sys/alt_alarm.h>`

Description: Only the system clock driver may call the `alt_tick()` function. The driver is responsible for making periodic calls to this function at the rate specified in the call to `alt_sysclk_init()`. This function provides notification to the system that a system clock tick has occurred. This function runs as a part of the ISR for the system clock driver.

Return: —

`alt_alarm_start()`

`alt_alarm_stop()`

`alt_ticks()`

`alt_sysclk_init()`

See also: `alt_ticks_per_second()`

`gettimeofday()`

`settimeofday()`

`times()`

`usleep()`

alt_ticks_per_second()

Prototype:	alt_u32 alt_ticks_per_second (void)
Commonly called by:	C/C++ programs Device drivers
Thread-safe:	Yes.
Available from ISR:	Yes.
Include:	<sys/alt_alarm.h>
Description:	The alt_ticks_per_second() function returns the number of system clock ticks that elapse per second. If there is no system clock available, the return value is zero.
Return:	Returns the number of system clock ticks that elapse per second.
See also:	alt_alarm_start() alt_alarm_stop() alt_nticks() alt_sysclk_init() alt_tick() gettimeofday() settimeofday() times() usleep()

alt_timestamp()

Prototype: `alt_u32 alt_timestamp (void)`

Commonly called by: C/C++ programs

Thread-safe: See description.

Available from ISR: See description.

Include: `<sys/alt_timestamp.h>`

Description: The `alt_timestamp()` function returns the current value of the timestamp counter. Refer to “Using Timer Devices” in the *Developing Programs Using the Hardware Abstraction Layer* chapter of the *Nios II Software Developer’s Handbook*. The implementation of this function is provided by the timestamp driver. Therefore, whether this function is thread-safe and or available at interrupt level depends on the underlying driver.

Always call the `alt_timestamp_start()` function before any calls to `alt_timestamp()`. Otherwise the behavior of `alt_timestamp()` is undefined.

Return: Returns the current value of the timestamp counter.

See also: [alt_timestamp_freq\(\)](#)
[alt_timestamp_start\(\)](#)

alt_timestamp_freq()

Prototype: `alt_u32 alt_timestamp_freq (void)`

Commonly called by: C/C++ programs

Thread-safe: See description.

Available from ISR: See description.

Include: `<sys/alt_timestamp.h>`

Description: The `alt_timestamp_freq()` function returns the rate at which the timestamp counter increments. Refer to “Using Timer Devices” in the *Developing Programs Using the Hardware Abstraction Layer* chapter of the *Nios II Software Developer’s Handbook*. The implementation of this function is provided by the timestamp driver. Therefore, whether this function is thread-safe and or available at interrupt level depends on the underlying driver.

Return: The returned value is the number of counter ticks per second.

See also: [alt_timestamp\(\)](#)
[alt_timestamp_start\(\)](#)

alt_timestamp_start()

Prototype: `int alt_timestamp_start (void)`

Commonly called by: C/C++ programs

Thread-safe: See description.

Available from ISR: See description.

Include: `<sys/alt_timestamp.h>`

Description: The `alt_timestamp_start()` function starts the system timestamp counter. Refer to “Using Timer Devices” in the *Developing Programs Using the Hardware Abstraction Layer* chapter of the *Nios II Software Developer’s Handbook*. The implementation of this function is provided by the timestamp driver. Therefore, whether this function is thread-safe and or available at interrupt level depends on the underlying driver.

This function resets the counter to zero, and starts the counter running.

Return: The return value is zero on success and nonzero otherwise.

See also: [alt_timestamp\(\)](#)
[alt_timestamp_freq\(\)](#)

alt_uncached_free()

Prototype:	<code>void alt_uncached_free (volatile void* ptr)</code>
Commonly called by:	C/C++ programs Device drivers
Thread-safe:	Yes.
Available from ISR:	No.
Include:	<code><sys/alt_cache.h></code>
Description:	The <code>alt_uncached_free()</code> function causes the memory pointed to by <code>ptr</code> to be deallocated, that is, made available for future allocation through a call to <code>alt_uncached_malloc()</code> . The input pointer, <code>ptr</code> , points to a region of memory previously allocated through a call to <code>alt_uncached_malloc()</code> . Behavior is undefined if this is not the case.
Return:	—
See also:	<code>alt_dcache_flush()</code> <code>alt_dcache_flush_all()</code> <code>alt_icache_flush()</code> <code>alt_icache_flush_all()</code> <code>alt_remap_cached()</code> <code>alt_remap_uncached()</code> <code>alt_uncached_malloc()</code>

alt_uncached_malloc()

Prototype:	<code>volatile void* alt_uncached_malloc (size_t size)</code>
Commonly called by:	C/C++ programs Device drivers
Thread-safe:	Yes.
Available from ISR:	No.
Include:	<code><sys/alt_cache.h></code>
Description:	<p>The <code>alt_uncached_malloc()</code> function allocates a region of uncached memory of length <code>size</code> bytes. Regions of memory allocated in this way can be released using the <code>alt_uncached_free()</code> function.</p> <p>Processors that do not have a data cache return uncached memory.</p>
Return:	<p>If sufficient memory cannot be allocated, this function returns null, otherwise a pointer to the allocated space is returned.</p> <p><code>alt_dcache_flush()</code> <code>alt_dcache_flush_all()</code> <code>alt_icache_flush()</code></p>
See also:	<p><code>alt_icache_flush_all()</code> <code>alt_remap_cached()</code> <code>alt_remap_uncached()</code> <code>alt_uncached_free()</code></p>

alt_write_flash()

Prototype:	<pre>int alt_write_flash(alt_flash_fd* fd, int offset, const void* src_addr, int length)</pre>
Commonly called by:	C/C++ programs Device drivers
Thread-safe:	No.
Available from ISR:	No.
Include:	<sys/alt_flash.h> The <code>alt_write_flash()</code> function writes data to flash. The data to be written is at address <code>src_addr</code> . <code>length</code> bytes are written to the flash <code>fd</code> , <code>offset</code> bytes from the beginning of the flash device address space.
Description:	Call this function only when operating in single-threaded mode. This function does not preserve any unwritten areas of any flash sectors affected by this write. Refer to “Using Flash Devices” in the <i>Developing Programs Using the Hardware Abstraction Layer</i> chapter of the <i>Nios II Software Developer’s Handbook</i> . The only valid values for the <code>fd</code> parameter are those returned from the <code>alt_flash_open_dev</code> function. If any other value is passed, the behavior of this function is undefined.
Return:	The return value is zero on success and nonzero otherwise.
See also:	alt_erase_flash_block() alt_flash_close_dev() alt_flash_open_dev() alt_get_flash_info() alt_read_flash() alt_write_flash_block()

alt_write_flash_block()

Prototype:

```
int alt_write_flash_block(alt_flash_fd* fd,
                          int           block_offset,
                          int           data_offset,
                          const void   *data,
                          int           length)
```


Commonly called by: C/C++ programs
Device drivers

Thread-safe: No.

Available from ISR: No.

Include: <sys/alt_flash.h>

The `alt_write_flash_block()` function writes one block of data of flash. The data to be written is at address `data`. `length` bytes are written to the flash `fd`, into the block starting at offset `block_offset` from the beginning of the flash address space. The data starts at offset `data_offset` from the beginning of the flash address space.

Description:  No check is performed on any of the parameters. Refer to “Using Flash Devices” in the *Developing Programs Using the Hardware Abstraction Layer* chapter of the *Nios II Software Developer’s Handbook*.

Call this function only when operating in single-threaded mode.

The only valid values for the `fd` parameter are those returned from the `alt_flash_open_dev` function. If any other value is passed, the behavior of this function is undefined.

Return: The return value is zero on success and nonzero otherwise.

[alt_erase_flash_block\(\)](#)
[alt_flash_close_dev\(\)](#)
[alt_flash_open_dev\(\)](#)
[alt_get_flash_info\(\)](#)
[alt_read_flash\(\)](#)
[alt_write_flash\(\)](#)

See also:

close()

Prototype:	<code>int close (int fd)</code>
Commonly called by:	C/C++ programs Newlib C library
Thread-safe:	See description.
Available from ISR:	No.
Include:	<code><unistd.h></code> The <code>close()</code> function is the standard UNIX-style <code>close()</code> function, which closes the file descriptor <code>fd</code> .
Description:	Calls to <code>close()</code> are thread-safe only if the implementation of <code>close()</code> provided by the driver that is manipulated is thread-safe. Valid values for the <code>fd</code> parameter are: <code>stdout</code> , <code>stdin</code> , and <code>stderr</code> , or any value returned from a call to <code>open()</code> .
Return:	The return value is zero on success, and <code>-1</code> otherwise. If an error occurs, <code>errno</code> is set to indicate the cause.
See also:	<code>fcntl()</code> <code>fstat()</code> <code>ioctl()</code> <code>isatty()</code> <code>lseek()</code> <code>open()</code> <code>read()</code> <code>stat()</code> <code>write()</code> Newlib documentation

execve()

Prototype: int execve(const char *path,
 char *const argv[],
 char *const envp[])

Commonly called by: Newlib C library

Thread-safe: Yes.

Available from ISR: Yes.

Include: <unistd.h>

Description: The `execve()` function is only provided for compatibility with newlib.

Return: Calls to `execve()` always fail with the return code `-1` and `errno` set to `ENOSYS`.

See also: Newlib documentation

fcntl()

Prototype: `int fcntl(int fd, int cmd)`

Commonly called by: C/C++ programs

Thread-safe: No.

Available from ISR: No.

Include: `<unistd.h>`
`<fcntl.h>`

Description: The `fcntl()` function is a limited implementation of the standard `fcntl()` system call, which can change the state of the flags associated with an open file descriptor. Normally these flags are set during the call to `open()`. The main use of this function is to change the state of a device from blocking to nonblocking (for device drivers that support this feature).

The input argument `fd` is the file descriptor to be manipulated. `cmd` is the command to execute, which can be either `F_GETFL` (return the current value of the flags) or `F_SETFL` (set the value of the flags).

If `cmd` is `F_SETFL`, the argument `arg` is the new value of flags, otherwise `arg` is ignored. Only the flags `O_APPEND` and `O_NONBLOCK` can be updated by a call to `fcntl()`. All other flags remain unchanged. The return value is zero on success, or `-1` otherwise.

Return: If `cmd` is `F_GETFL`, the return value is the current value of the flags. If an error occurs, `-1` is returned.

In the event of an error, `errno` is set to indicate the cause.

[close\(\)](#)

[fstat\(\)](#)

[ioctl\(\)](#)

[isatty\(\)](#)

See also: [lseek\(\)](#)

[read\(\)](#)

[stat\(\)](#)

[write\(\)](#)

[Newlib documentation](#)

fork()

Prototype:	<code>pid_t fork (void)</code>
Commonly called by:	Newlib C library
Thread-safe:	Yes.
Available from ISR:	no
Include:	<code><unistd.h></code>
Description:	The <code>fork()</code> function is only provided for compatibility with newlib.
Return:	Calls to <code>fork()</code> always fails with the return code <code>-1</code> and <code>errno</code> set to <code>ENOSYS</code> .
See also:	Newlib documentation

fstat()

Prototype: `int fstat (int fd, struct stat *st)`

Commonly called by: C/C++ programs

Newlib C library

Thread-safe: See description.

Available from ISR: No.

Include: `<sys/stat.h>`

The `fstat()` function obtains information about the capabilities of an open file descriptor. The underlying device driver fills in the input `st` structure with a description of its functionality. Refer to the header file **sys/stat.h** provided with the compiler for the available options.

Description: By default, file descriptors are marked as character devices, unless the underlying driver provides its own implementation of the `fstat()` function.

Calls to `fstat()` are thread-safe only if the implementation of `fstat()` provided by the driver that is manipulated is thread-safe.

Valid values for the `fd` parameter are: `stdout`, `stdin`, and `stderr`, or any value returned from a call to `open()`.

Return: The return value is zero on success, or `-1` otherwise. If the call fails, `errno` is set to indicate the cause of the error.

[close\(\)](#)

[fcntl\(\)](#)

[ioctl\(\)](#)

[isatty\(\)](#)

[lseek\(\)](#)

See also:

[open\(\)](#)

[read\(\)](#)

[stat\(\)](#)

[write\(\)](#)

Newlib documentation

getpid()

Prototype:	<code>pid_t getpid (void)</code>
Commonly called by:	Newlib C library
Thread-safe:	Yes.
Available from ISR:	No.
Include:	<code><unistd.h></code>
Description:	The <code>getpid()</code> function is provided for newlib compatibility and obtains the current process id.
Return:	Because HAL systems cannot contain multiple processes, <code>getpid()</code> always returns the same id number.
See also:	Newlib documentation

gettimeofday()

Prototype:	<pre>int gettimeofday(struct timeval *ptimeval, struct timezone *ptimezone)</pre>
Commonly called by:	C/C++ programs Newlib C library
Thread-safe:	See description.
Available from ISR:	Yes.
Include:	<pre><sys/time.h></pre>
Description:	<p>The <code>gettimeofday()</code> function obtains a time structure that indicates the current time. This time is calculated using the elapsed number of system clock ticks, and the current time value set by the most recent call to <code>settimeofday()</code>.</p> <p>If this function is called concurrently with a call to <code>settimeofday()</code>, the value returned by <code>gettimeofday()</code> is unreliable; however, concurrent calls to <code>gettimeofday()</code> are legal.</p>
Return:	<p>The return value is zero on success. If no system clock is available, the return value is <code>-ENOTSUP</code>.</p> <pre>alt_alarm_start() alt_alarm_stop() alt_nticks() alt_sysclk_init() alt_tick() alt_ticks_per_second() settimeofday() times() usleep()</pre>
See also:	<pre>Newlib documentation</pre>

ioctl()

Prototype: `int ioctl (int fd, int req, void* arg)`

Commonly called by: C/C++ programs

Thread-safe: See description.

Available from ISR: No.

Include: `<sys/ioctl.h>`

The `ioctl()` function allows application code to manipulate the I/O capabilities of a device driver in driver-specific ways. This function is equivalent to the standard UNIX `ioctl()` function. The input argument `fd` is an open file descriptor for the device to manipulate, `req` is an enumeration defining the operation request, and the interpretation of `arg` is request specific.

For file subsystems, `ioctl()` is wrapper function that passes control directly to the appropriate device driver's `ioctl()` function (as registered in the driver's `alt_dev` structure).

Description: For devices, `ioctl()` handles `TIOCEXCL` and `TIOCNXCL` requests internally, without calling the device driver. These requests lock and release a device for exclusive access. For requests other than `TIOCEXCL` and `TIOCNXCL`, `ioctl()` passes control to the device driver's `ioctl()` function.

Calls to `ioctl()` are thread-safe only if the implementation of `ioctl()` provided by the driver that is manipulated is thread-safe.

Valid values for the `fd` parameter are: `stdout`, `stdin`, and `stderr`, or any value returned from a call to `open()`.

Return: The interpretation of the return value is request specific. If the call fails, `errno` is set to indicate the cause of the error.

`close()`

`fcntl()`

`fstat()`

`isatty()`

`lseek()`

See also:

`open()`

`read()`

`stat()`

`write()`

Newlib documentation

isatty()

Prototype: `int isatty(int fd)`

Commonly called by: C/C++ programs
Newlib C library

Thread-safe: See description.

Available from ISR: No.

Include: `<unistd.h>`

Description: The `isatty()` function determines whether the device associated with the open file descriptor `fd` is a terminal device. This implementation uses the driver's `fstat()` function to determine its reply.

Calls to `isatty()` are thread-safe only if the implementation of `fstat()` provided by the driver that is manipulated is thread-safe.

Return: The return value is 1 if the device is a character device, and zero otherwise. If an error occurs, `errno` is set to indicate the cause.

See also: `close()`
`fcntl()`
`fstat()`
`ioctl()`

`lseek()`
`open()`
`read()`
`stat()`
`write()`

Newlib documentation

kill()

Prototype:	<code>int kill(int pid, int sig)</code>
Commonly called by:	Newlib C library
Thread-safe:	Yes.
Available from ISR:	Yes.
Include:	<code><signal.h></code> <p>The <code>kill()</code> function is used by newlib to send signals to processes. The input argument <code>pid</code> is the id of the process to signal, and <code>sig</code> is the signal to send. As there is only a single process in the HAL, the only valid values for <code>pid</code> are either the current process id, as returned by <code>getpid()</code>, or the broadcast values, that is, <code>pid</code> must be less than or equal to zero.</p>
Description:	<p>The following signals result in an immediate shutdown of the system, without call to <code>exit()</code>: <code>SIGABRT</code>, <code>SIGALRM</code>, <code>SIGFPE</code>, <code>SIGILL</code>, <code>SIGKILL</code>, <code>SIGPIPE</code>, <code>SIGQUIT</code>, <code>SIGSEGV</code>, <code>SIGTERM</code>, <code>SIGUSR1</code>, <code>SIGUSR2</code>, <code>SIGBUS</code>, <code>SIGPOLL</code>, <code>SIGPROF</code>, <code>SIGSYS</code>, <code>SIGTRAP</code>, <code>SIGVTALRM</code>, <code>SIGXCPU</code>, and <code>SIGXFSZ</code>.</p> <p>The following signals are ignored: <code>SIGCHLD</code> and <code>SIGURG</code>.</p> <p>All the remaining signals are treated as errors.</p>
Return:	The return value is zero on success, or <code>-1</code> otherwise. If the call fails, <code>errno</code> is set to indicate the cause of the error.
See also:	Newlib documentation

link()

Prototype:	<pre>int link(const char *_path1, const char *_path2)</pre>
Commonly called by:	Newlib C library
Thread-safe:	Yes.
Available from ISR:	Yes.
Include:	<unistd.h>
Description:	The <code>link()</code> function is only provided for compatibility with newlib.
Return:	Calls to <code>link()</code> always fails with the return code <code>-1</code> and <code>errno</code> set to <code>ENOSYS</code> .
See also:	Newlib documentation

lseek()

Prototype: `off_t lseek(int fd, off_t ptr, int whence)`

Commonly called by: C/C++ programs
Newlib C library

Thread-safe: See description.

Available from ISR: No.

Include: `<unistd.h>`

The `lseek()` function moves the read/write pointer associated with the file descriptor `fd`. `lseek()` is wrapper function that passes control directly to the `lseek()` function registered for the driver associated with the file descriptor. If the driver does not provide an implementation of `lseek()`, an error is reported.

`lseek()` corresponds to the standard UNIX `lseek()` function.

You can use the following values for the input parameter, `whence`:

Description:

- `SEEK_SET`—The offset is set to `ptr` bytes.
- `SEEK_CUR`—The offset is incremented by `ptr` bytes.
- `SEEK_END`—The offset is set to the end of the file plus `ptr` bytes.

Calls to `lseek()` are thread-safe only if the implementation of `lseek()` provided by the driver that is manipulated is thread-safe.

Valid values for the `fd` parameter are: `stdout`, `stdin`, and `stderr`, or any value returned from a call to `open()`.

Return:

On success, the return value is a nonnegative file pointer. The return value is `-1` in the event of an error. If the call fails, `errno` is set to indicate the cause of the error.

`close()`

`fcntl()`

`fstat()`

`ioctl()`

`isatty()`

See also:

`open()`


`read()`

`stat()`

`write()`

Newlib documentation

open()

Prototype:	<code>int open (const char* pathname, int flags, mode_t mode)</code>
Commonly called by:	C/C++ programs
Thread-safe:	See description.
Available from ISR:	No.
Include:	<code><unistd.h></code> <code><fcntl.h></code>
Description:	<p>The <code>open()</code> function opens a file or device and returns a file descriptor (a small, nonnegative integer for use in read, write, etc.)</p> <p><code>flags</code> is one of: <code>O_RDONLY</code>, <code>O_WRONLY</code>, or <code>O_RDWR</code>, which request opening the file in read-only, write-only, or read/write mode, respectively.</p> <p>You can also bitwise-OR <code>flags</code> with <code>O_NONBLOCK</code>, which causes the file to be opened in nonblocking mode. Neither <code>open()</code> nor any subsequent operation on the returned file descriptor causes the calling process to wait.</p> <p> Not all file systems/devices recognize this option.</p> <p><code>mode</code> specifies the permissions to use, if a new file is created. It is unused by current file systems, but is maintained for compatibility.</p> <p>Calls to <code>open()</code> are thread-safe only if the implementation of <code>open()</code> provided by the driver that is manipulated is thread-safe.</p>
Return:	<p>The return value is the new file descriptor, and <code>-1</code> otherwise. If an error occurs, <code>errno</code> is set to indicate the cause.</p>
See also:	<p><code>close()</code> <code>fcntl()</code> <code>fstat()</code> <code>ioctl()</code> <code>isatty()</code> <code>lseek()</code> <code>read()</code> <code>stat()</code> <code>write()</code></p> <p>Newlib documentation</p>

read()

Prototype: `int read(int fd, void *ptr, size_t len)`

Commonly called by: C/C++ programs
Newlib C library

Thread-safe: See description.

Available from ISR: No.

Include: `<unistd.h>`

The `read()` function reads a block of data from a file or device. `read()` is wrapper function that passes control directly to the `read()` function registered for the device driver associated with the open file descriptor `fd`. The input argument, `ptr`, is the location to place the data read and `len` is the length of the data to read in bytes.

Description: Calls to `read()` are thread-safe only if the implementation of `read()` provided by the driver that is manipulated is thread-safe.

Valid values for the `fd` parameter are: `stdout`, `stdin`, and `stderr`, or any value returned from a call to `open()`.

Return: The return argument is the number of bytes read, which might be less than the requested length. The return value is `-1` upon an error. In the event of an error, `errno` is set to indicate the cause.

`close()`

`fcntl()`

`fstat()`

`ioctl()`

`isatty()`

See also:

`lseek()`

`open()`

`stat()`

`write()`

Newlib documentation

sbrk()

Prototype: `caddr_t sbrk(int incr)`

Commonly called by: Newlib C library

Thread-safe: No.

Available from ISR: No.

Include: `<unistd.h>`

Description: The `sbrk()` function dynamically extends the data segment for the application. The input argument `incr` is the size of the block to allocate. Do not call `sbrk()` directly. If you wish to dynamically allocate memory, use the newlib `malloc()` function.

Return: `-`

See also: Newlib documentation

settimeofday()

Prototype:	<code>int settimeofday (const struct timeval *t, const struct timezone *tz)</code>
Commonly called by:	C/C++ programs
Thread-safe:	No.
Available from ISR:	Yes.
Include:	<code><sys/time.h></code>
Description:	If the <code>settimeofday()</code> function is called concurrently with a call to <code>gettimeofday()</code> , the value returned by <code>gettimeofday()</code> is unreliable.
Return:	The return value is zero on success. If no system clock is available, the return value is -1, and <code>errno</code> is set to <code>ENOSYS</code> .
See also:	<code>alt_alarm_start()</code> <code>alt_alarm_stop()</code> <code>alt_nticks()</code> <code>alt_sysclk_init()</code> <code>alt_tick()</code> <code>alt_ticks_per_second()</code> <code>gettimeofday()</code> <code>times()</code> <code>usleep()</code>

stat()

Prototype:	<pre>int stat(const char *file_name, struct stat *buf);</pre>
Commonly called by:	C/C++ programs Newlib C library
Thread-safe:	See description.
Available from ISR:	No.
Include:	<pre><sys/stat.h></pre> <p>The <code>stat()</code> function is similar to the <code>fstat()</code> function—It obtains status information about a file. Instead of using an open file descriptor, like <code>fstat()</code>, <code>stat()</code> takes the name of a file as an input argument.</p>
Description:	<p>Calls to <code>stat()</code> are thread-safe only if the implementation of <code>stat()</code> provided by the driver that is manipulated is thread-safe.</p> <p>Internally, the <code>stat()</code> function is implemented as a call to <code>fstat()</code>. Refer to “fstat()” on page 14-62.</p>
Return:	— close() fcntl() fstat() ioctl() isatty()
See also:	lseek() open() read() write() Newlib documentation

times()

Prototype: `clock_t times (struct tms *buf)`

Commonly called by: C/C++ programs
Newlib C library

Thread-safe: Yes.

Available from ISR: Yes.

Include: `<sys/times.h>`

This `times()` function is provided for compatibility with newlib. It returns the number of clock ticks since reset. It also fills in the structure pointed to by the input parameter `buf` with time accounting information. The definition of the `tms` structure is:

```
typedef struct
{
    clock_t tms_utime;
    clock_t tms_stime;
    clock_t tms_cutime;
    clock_t tms_cstime;
};
```

Description: The structure has the following elements:

- `tms_utime`: the processor time charged for the execution of user instructions
- `tms_stime`: the processor time charged for execution by the system on behalf of the process
- `tms_cutime`: the sum of the values of `tms_utime` and `tms_cutime` for all child processes
- `tms_cstime`: the sum of the values of `tms_stime` and `tms_cstime` for all child processes

In practice, all elapsed time is accounted as system time. No time is ever attributed as user time. In addition, no time is allocated to child processes, as child processes cannot be spawned by the HAL.

Return: If there is no system clock available, the return value is zero, and `errno` is set to `ENOSYS`.

[alt_alarm_start\(\)](#)

[alt_alarm_stop\(\)](#)

[alt_nticks\(\)](#)

[alt_sysclk_init\(\)](#)

[alt_tick\(\)](#)

See also: [alt_ticks_per_second\(\)](#)

[gettimeofday\(\)](#)

[settimeofday\(\)](#)

[usleep\(\)](#)

Newlib documentation

unlink()

Prototype:	<code>int unlink(char *name)</code>
Commonly called by:	Newlib C library
Thread-safe:	Yes.
Available from ISR:	Yes.
Include:	<code><unistd.h></code>
Description:	The <code>unlink()</code> function is only provided for compatibility with newlib.
Return:	Calls to <code>unlink()</code> always fails with the return code <code>-1</code> and <code>errno</code> set to <code>ENOSYS</code> .
See also:	Newlib documentation

usleep()

Prototype:	<code>int usleep (unsigned int us)</code>
Commonly called by:	C/C++ programs Device drivers
Thread-safe:	Yes.
Available from ISR:	No.
Include:	<code><unistd.h></code>
Description:	The <code>usleep()</code> function blocks until at least <code>us</code> microseconds have elapsed.
Return:	The <code>usleep()</code> function returns zero on success, or <code>-1</code> otherwise. If an error occurs, <code>errno</code> is set to indicate the cause. The current implementation always succeeds.
See also:	<code>alt_alarm_start()</code> <code>alt_alarm_stop()</code> <code>alt_nticks()</code> <code>alt_sysclk_init()</code> <code>alt_tick()</code> <code>alt_ticks_per_second()</code> <code>gettimeofday()</code> <code>settimeofday()</code> <code>times()</code>

wait()

Prototype:	<code>int wait(int *status)</code>
Commonly called by:	Newlib C library
Thread-safe:	Yes.
Available from ISR:	Yes.
Include:	<code><sys/wait.h></code>
Description:	Newlib uses the <code>wait()</code> function to wait for all child processes to exit. Because the HAL does not support spawning child processes, this function returns immediately. On return, the content of <code>status</code> is set to zero, which indicates there is no child processes.
Return:	The return value is always <code>-1</code> and <code>errno</code> is set to <code>ECHILD</code> , which indicates that there are no child processes to wait for.
See also:	Newlib documentation

write()

Prototype: `int write(int fd, const void *ptr, size_t len)`

Commonly called by: C/C++ programs
Newlib C library

Thread-safe: See description.

Available from ISR: no

Include: `<unistd.h>`

The `write()` function writes a block of data to a file or device. `write()` is wrapper function that passes control directly to the `write()` function registered for the device driver associated with the file descriptor `fd`. The input argument `ptr` is the data to write and `len` is the length of the data in bytes.

Description: Calls to `write()` are thread-safe only if the implementation of `write()` provided by the driver that is manipulated is thread-safe.

Valid values for the `fd` parameter are: `stdout`, `stdin`, and `stderr`, or any value returned from a call to `open()`.

The return argument is the number of bytes written, which might be less than the requested length.

Return:

The return value is `-1` upon an error. In the event of an error, `errno` is set to indicate the cause.

`close()`

`fcntl()`

`fstat()`

`ioctl()`

`isatty()`

See also:

`lseek()`

`open()`

`read()`

`stat()`

Newlib documentation

Standard Types

In the interest of portability, the HAL uses a set of standard type definitions in place of the ANSI C built-in types. Table 14-2 describes these types, which are defined in the header file `alt_types.h`.

Table 14-2. Standard Types

Type	Description
<code>alt_8</code>	Signed 8-bit integer.
<code>alt_u8</code>	Unsigned 8-bit integer.
<code>alt_16</code>	Signed 16-bit integer.
<code>alt_u16</code>	Unsigned 16-bit integer.
<code>alt_32</code>	Signed 32-bit integer.

Table 14–2. Standard Types

Type	Description
alt_u32	Unsigned 32-bit integer.
alt_64	Signed 64-bit integer.
alt_u64	Unsigned 64-bit integer.

Document Revision History

Table 14–3 shows the revision history for this document.

Table 14–3. Document Revision History

Date	Version	Changes
February 2011	10.1.0	Removed “Referenced Documents” section.
July 2010	10.0.0	<ul style="list-style-type: none"> ■ Clarify purpose of listed C header file for functions. ■ Correction: <code>alt_irq_enabled()</code> is not a legacy function.
November 2009	9.1.0	<ul style="list-style-type: none"> ■ Document new enhanced HAL interrupt API functions: <code>alt_ic_irq_disable()</code>, <code>alt_ic_irq_enable()</code>, <code>alt_ic_irq_enabled()</code>, and <code>alt_ic_isr_register()</code>. ■ Deprecate legacy HAL interrupt API functions <code>alt_irq_disable()</code>, <code>alt_irq_enable()</code>, <code>alt_irq_enabled()</code>, and <code>alt_irq_register()</code>.
March 2009	9.0.0	<ul style="list-style-type: none"> ■ Corrected minor typographical errors.
May 2008	8.0.0	<ul style="list-style-type: none"> ■ Advanced exceptions added to Nios II core. ■ Instruction-related exception handling added to HAL. ■ Added <code>alt_instruction_exception_register()</code> and <code>alt_exception_cause_generated_bad_addr()</code> for instruction-related exception handlers.
October 2007	7.2.0	Maintenance release.
May 2007	7.1.0	<ul style="list-style-type: none"> ■ Added table of contents to “Introduction” section. ■ Added Referenced Documents section.
March 2007	7.0.0	Maintenance release.
November 2006	6.1.0	Function <code>open()</code> requires <code>fcntl.h</code> .
May 2006	6.0.0	Maintenance release.
October 2005	5.1.0	Added API entries for “ <code>alt_irq_disable()</code> ” and “ <code>alt_irq_enable()</code> ”, which were previously omitted by error.
May 2005	5.0.0	<ul style="list-style-type: none"> ■ Added <code>alt_load_section()</code> function. ■ Added <code>fcntl()</code> function.
December 2004	1.2	Updated names of DMA generic requests.
September 2004	1.1	<ul style="list-style-type: none"> ■ Added <code>open()</code>. ■ Added <code>ERRNO</code> information to <code>alt_dma_txchan_open()</code>. ■ Corrected <code>ALT_DMA_TX_STREAM_ON</code> definition. ■ Corrected <code>ALT_DMA_RX_STREAM_ON</code> definition. ■ Added information to <code>alt_dma_rxchan_ioctl()</code> and <code>alt_dma_txchan_ioctl()</code>.
May 2004	1.0	Initial release.

This chapter provides a complete reference of all available commands, options, and settings for the Nios® II Software Build Tools (SBT). This reference is useful for developing your own software projects, packages, or device drivers.

- Before using this chapter, read the *Getting Started from the Command Line* chapter of the *Nios II Software Developer's Handbook*, and familiarize yourself with the parts of the *Nios II Software Build Tools* chapter of the *Nios II Software Developer's Handbook* that are relevant to your tasks.

This chapter includes the following sections:

- “Nios II Software Build Tools Utilities” on page 15-1
- “Nios II Example Design Scripts” on page 15-24
- “Settings” on page 15-27
- “Application and User Library Makefile Variables” on page 15-68
- “Tcl Commands” on page 15-71
- “Path Names” on page 15-117

Nios II Software Build Tools Utilities

The build tools utilities are an entry point to the Nios II SBT. Everything you can do with the tools, such as specifying settings, creating makefiles, and building projects, is made available by the utilities.

All Nios II SBT utilities share the following behavior:

- Sends error messages and warning messages to `stderr`.
- Sends normal messages (other than errors and warnings) to `stdout`.
- Displays one error message for each error.
- Returns an exit value of 1 if it detects any errors.
- Returns an exit value of 0 if it does not detect any errors. (Warnings are not errors.)
- If the `help` or `version` command-line option is specified, returns an exit value of 0, and takes no other action. Sends the output (`help` or `version` number) to `stdout`.
- If no command-line arguments are specified, returns an exit value of 1 and sends a help message to `stderr`. All commands require at least one argument.
- When an error is detected, suppresses all subsequent operations (such as writing files).

Logging Levels

All the utilities support multiple status-logging levels. You specify the logging level on the command line. [Table 15-1](#) shows the logging levels supported. At each level, the utilities report the status as listed under **Description**. Each level includes the messages from all lower levels.

Table 15-1. Nios II SBT Logging Levels

Logging Level	Description
silent (lowest)	No information is provided except for errors and warnings (sent to <code>stderr</code>).
default	Minimal information is provided (for example, start and stop operation of SBT phases).
verbose	Detailed information is provided (for example, lists of files written).
debug (highest)	Debug information is provided (for example, stack backtraces on errors). This level is for reporting problems to Altera.

[Table 15-2](#) shows the command-line options used to select each logging level. Only one logging level is possible at a time, so these options are all mutually exclusive.

Table 15-2. Selecting Logging Level

Command-Line Option	Logging Level	Comments
none	default	If there is no command-line option, the default level is selected.
<code>--silent</code>	silent	Selects silent level of logging.
<code>--verbose</code>	verbose	Selects verbose level of logging.
<code>--debug</code>	debug	Selects debug level of logging.
<code>--log <fname></code>	debug	All information is written to <code><fname></code> in addition to being sent to the <code>stdout</code> and <code>stderr</code> devices.

Setting Values

The value of a setting is specified with the `--set` command-line option to `nios2-bsp-create-settings` or `nios2-bsp-update-settings`, or with the `set_setting` Tcl command. The value of a setting is obtained with the `--get` command-line option to `nios2-bsp-query-settings` or with the `get_setting` Tcl command.

For more information about settings values and formats, refer to [“Settings” on page 15-27](#).

Utility and Script Summary

The following command-line utilities and scripts are available:

- [“nios2-app-generate-makefile”](#)
- [“nios2-bsp-create-settings” on page 15-4](#)
- [“nios2-bsp-generate-files” on page 15-6](#)
- [“nios2-bsp-query-settings” on page 15-7](#)
- [“nios2-bsp-update-settings” on page 15-8](#)

- “nios2-lib-generate-makefile” on page 15–10
- “nios2-bsp-editor” on page 15–11
- “nios2-app-update-makefile” on page 15–12
- “nios2-lib-update-makefile” on page 15–14
- “nios2-swexample-create” on page 15–16
- “nios2-elf-insert” on page 15–17
- “nios2-elf-query” on page 15–18
- “nios2-convert-ide2sbt” on page 15–19
- “nios2-c2h-generate-makefile” on page 15–22
- “nios2-bsp” on page 15–23

nios2-app-generate-makefile

Usage

```
nios2-app-generate-makefile [--app-dir <directory>]
  --bsp-dir <directory> [--debug]
  [--elf-name <filename>] [--extended-help] [--help]
  [--log <filename>] [--no-src] [--set <name> <value>]
  [--silent] [--src-dir <directory>]
  [--src-files <filenames>] [--src-rdir <directory>]
  [--use-lib-dir <directory>] [--verbose]
  [--version] [--c2h]
```

Options


- `--app-dir <directory>`: Directory to place the application makefile and executable and linking format file (.elf). If omitted, it defaults to the current directory.
- `--bsp-dir <directory>`: Specifies the path to the BSP generated files directory (populated using the `nios2-bsp-generate-files` command).
- `--debug`: Output debug, exception traces, verbose, and default information about the command’s operation to stdout.
- `--elf-name <filename>`: Name of the .elf file to create. If omitted, it defaults to the first source file specified with the file name extension replaced with .elf and placed in the application directory.
- `--extended-help`: Displays full information about this command and its options.
- `--help`: Displays basic information about this command and its options.
- `--log <filename>`: Create a debug log and write to specified file. Also logs debug information to stdout.
- `--no-src`: Allows no sources files to be set in the Makefile. User must add source files in manually before compiling
- `--set <name> <value>`: Set the makefile variable called `<name>` to `<value>`. If the variable exists in the managed section of the makefile, `<value>` replaces the default settings. If the variable does not already exist, it is added. Multiple `--set` options are allowed.

- `--silent`: Suppress information about the command's operation normally sent to stdout.
- `--src-dir <directory>`: Searches for source files in *<directory>*. Use `.` to look in the current directory. Multiple `--src-dir` options are allowed.
- `--src-files <filenames>`: Adds a list of space-separated source file names to the makefile. The list of file names is terminated by the next option or the end of the command line. Multiple `--src-files` options are allowed.
- `--src-rdir <directory>`: Same as `--src-dir` option but recursively search for source files in or under *<directory>*. Multiple `--src-rdir` options are allowed and can be freely mixed with `--src-dir` options.
- `--use-lib-dir <directory>`: Specifies the path to a dependent user library directory. The user library directory must contain a makefile fragment called **public.mk**. Multiple `--use-lib-dir` options are allowed.
- `--verbose`: Output verbose, and default information about the command's operation to stdout.
- `--version`: Displays the version of this command and exits with a zero exit status.


Description

The **nios2-app-generate-makefile** command generates an application makefile (called Makefile). The path to a BSP created by **nios2-bsp-generate-files** is a mandatory command-line option.

You can enable support for the Nios II C2H Compiler with the `--c2h` option, which creates a null C2H makefile fragment in your project, and includes it in the application makefile. This makefile fragment, `c2h.mk`, contains comments to help you fill in the makefile variables by hand.

 This **c2h.mk** overwrites any existing **c2h.mk**.

You can use the command-line tool **nios2-c2h-generate-makefile** to generate a populated C2H makefile fragment.

 The **nios2-c2h-generate-makefile** script is available to support pre-existing command-line C2H projects. Create new C2H projects using the Nios II IDE.

For more details about this command, use the `--extended-help` option to display comprehensive usage information.

nios2-bsp-create-settings

Usage

```
nios2-bsp-create-settings [--bsp-dir <directory>]
  [--cmd <tcl command>] [--cpu-name <cpu name>]
  [--debug] [--extended-help] [--get-cpu-arch]
  [--help] [--jdi <filename>]
  [--librarian-factory-path <directory>]
```

```
[--librarian-path <directory>] [--log <filename>]
[--script <filename>] [--set <name> <value>]
--settings <filename> [--silent]
--sopc <filename> --type <OS name> [--type-version <version>]
[--verbose] [--version]
```

Options

- `--bsp-dir <directory>`: Path to the directory where the BSP files are generated. Use `.` for the current directory. The directory `<directory>` must exist. This command overwrites preexisting files in `<directory>` without warning.
- `--cmd <tcl command>`: Runs the specified Tcl command. Multiple `--cmd` options are allowed.
- `--cpu-name <cpu name>`: The name of the Nios II processor that the BSP supports. Optional for a single-processor system.
- `--debug`: Sends debug information, exception traces, verbose output, and default information about the command's operation, to stdout.
- `--extended-help`: Displays full information about this command and its options.
- `--get-cpu-arch`: Queries for processor architecture from the processor specified. Does not create a BSP.
- `--help`: Displays basic information about this command and its options.
- `--jdi <filename>`: The location of the JTAG Debugging Information File (`.jdi`) generated by the Quartus® II software. The `.jdi` file specifies the name-to-node mappings for the JTAG chain elements. The tool inserts the `.jdi` path in `public.mk`. If no `.jdi` path is specified, the command searches the directory containing the `.sopcinfo` file, and uses the first `.jdi` file found.
- `--librarian-factory-path <directory>`: Comma separated librarian search path. Use `$` for default factory search path.
- `--librarian-path <directory>`: Comma separated librarian search path. Use `$` for default search path.
- `--log <filename>`: Creates a debug log and write to specified file. Also logs debug information to stdout.
- `--script <filename>`: Run the specified Tcl script with optional arguments. Multiple `--script` options are allowed.
- `--set <name> <value>`: Sets the setting called `<name>` to `<value>`. Multiple `--set` options are allowed.
- `--settings <filename>`: File name of the BSP settings file to create. This file is created with a `.bsp` file extension. It overwrites any existing settings file.
- `--silent`: Suppresses information about the command's operation normally sent to stdout.
- `--sopc <filename>`: The SOPC Information File (`.sopcinfo`) used to create the BSP.
- `--type <OS name>`: BSP type. Use `?` to list available BSP types for this option.
- `--type-version <version>`: BSP software component version. By default the latest version is used. `default` value can be used to reset to this default behavior. Use `?` to list available BSP types and versions.

- `--verbose`: Sends verbose output, and default information about the command's operation, to stdout.
- `--version`: Displays the version of this command and exits with a zero exit status.

Description

If you use `nios2-bsp-create-settings` to create a settings file without any command-line options, Tcl commands, or Tcl scripts to modify the default settings, it creates a settings file that fails when running `nios2-bsp-generate-files`. Failure occurs because the `nios2-bsp-create-settings` command is able to create reasonable defaults for most settings, but the command requires additional information for system-dependent settings. The default Tcl scripts set the required system-dependent settings. Therefore it is better to use default Tcl scripts when calling `nios2-bsp-create-settings` directly. For an example of how to use the default Tcl scripts, refer to the `nios2-bsp` script.

For more details about this command, use the `--extended-help` option to display comprehensive usage information.

Example

```
nios2-bsp-create-settings --settings my_settings.bsp --sopc \
  ../my_sopc.sopcinfo --type hal --script default_settings.tcl
```

nios2-bsp-generate-files

Usage

```
nios2-bsp-generate-files --bsp-dir <directory>
  [--debug] [--extended-help] [--help]
  [--librarian-factory-path <directory>]
  [--librarian-path <directory>] [--log <filename>]
  --settings <filename> [--silent] [--verbose]
  [--version]
```

Options

- `--bsp-dir <directory>`: Path to the directory where the BSP files are generated. Use `.` for the current directory. The directory `<directory>` must exist. This command overwrites preexisting files in `<directory>` without warning.
- `--debug`: Sends debug, exception trace, verbose, and default information about the command's operation to stdout.
- `--extended-help`: Displays full information about this command and its options.
- `--help`: Displays basic information about this command and its options.
- `--librarian-factory-path <directory>`: Comma separated librarian search path. Use `$` for default factory search path.
- `--librarian-path <directory>`: Comma separated librarian search path. Use `$` for default search path.
- `--log <filename>`: Creates a debug log and writes to specified file. Also logs debug information to stdout.

- `--settings <filename>`: File name of an existing BSP Settings File (`.bsp`) to generate files from.
- `--silent`: Suppresses information about the command's operation normally sent to stdout.
- `--verbose`: Sends verbose and default information about the command's operation to stdout.
- `--version`: Displays the version of this command and exits with a zero exit status.

Description

The `nios2-bsp-generate-files` command populates the files in a BSP directory. The path to an existing `.bsp` file and the path to the BSP directory are mandatory command-line options. Files are written to the specified BSP directory. Generated files are created unconditionally. Copied files are copied from the Nios II EDS installation folder only if they are not present in the BSP directory, or if the existing files differ from the installation files.

```
--set APP_INCLUDE_DIRS \"lcd board\"
```

For more details about this command, use the `--extended-help` option to display comprehensive usage information.

nios2-bsp-query-settings

Usage

```
nios2-bsp-query-settings [--cmd <tcl command>]
  [--debug] [--extended-help] [--get <name>]
  [--get-all] [--help]
  [--librarian-factory-path <directory>]
  [--librarian-path <directory>] [--log <filename>]
  [--script <filename>] --settings <filename>
  [--show-descriptions] [--show-names] [--silent]
  [--verbose] [--version]
```

Options

- `--cmd <tcl command>`: Run the specified Tcl command. Multiple `--cmd` options are allowed.
- `--debug`: Output debug, exception traces, verbose, and default information about the command's operation to stdout.
- `--extended-help`: Displays full information about this command and its options.
- `--get <name>`: Display the value of the setting called `<name>`. Multiple `--get` options are allowed. Each value appears on its own line in the order the `--get` options are specified. Mutually exclusive with the `--get-all` option.
- `--get-all`: Display the value of all BSP settings in order sorted by option name. Each option appears on its own line. Mutually exclusive with the `--get` option.
- `--help`: Displays basic information about this command and its options.
- `--librarian-factory-path <directory>`: Comma separated librarian search path. Use `$` for default factory search path.

- `--librarian-path <directory>`: Comma separated librarian search path. Use \$ for default search path.
- `--log <filename>`: Create a debug log and write to specified file. Also logs debug information to stdout.
- `--script <filename>`: Run the specified Tcl script with optional arguments. Multiple `--script` options are allowed.
- `--settings <filename>`: File name of an existing BSP settings file to query settings from.
- `--show-descriptions`: Displays the description of each option after the value.
- `--show-names`: Displays the name of each option before the value.
- `--silent`: Suppress information about the command's operation normally sent to stdout.
- `--verbose`: Output verbose, and default information about the command's operation to stdout.
- `--version`: Displays the version of this command and exits with a zero exit status.

Description

The `nios2-bsp-query-settings` command provides information from a .bsp file. The path to an existing .bsp file is a mandatory command-line option. The command does not modify the settings file. Only information requested by the user is displayed on stdout; no informational messages are displayed.

For more details about this command, use the `--extended-help` option to display comprehensive usage information.

nios2-bsp-update-settings

Usage

```
nios2-bsp-update-settings [--bsp-dir <directory>]
  [--cmd <tcl command>] [--cpu-name <cpu name>]
  [--debug] [--extended-help] [--help] [--jdi <filename>]
  [--librarian-factory-path <directory>]
  [--librarian-path <directory>] [--log <filename>]
  [--script <filename>] [--set <name> <value>]
  --settings <filename> [--silent]
  [--sopc <filename>] [--verbose] [--version]
```

Options

- `--bsp-dir <directory>`: Path to the directory where the BSP files are generated. Use . for the current directory. The directory `<directory>` must exist.
- `--cmd <tcl command>`: Run the specified Tcl command. Multiple `--cmd` options are allowed.
- `--cpu-name <cpu name>`: The name of the Nios II processor that the BSP supports. This argument is useful if the hardware design contains multiple Nios II processors. Optional for a single-processor design.

- `--debug`: Output debug, exception traces, verbose, and default information about the command's operation to stdout.
- `--extended-help`: Displays full information about this command and its options.
- `--help`: Displays basic information about this command and its options.
- `--jdi <filename>`: The location of the `.jdi` file generated by the Quartus II software. The `.jdi` file specifies the name-to-node mappings for the JTAG chain elements. The tool inserts the `.jdi` path in `public.mk`. If no `.jdi` path is specified, the command searches the directory containing the `.sopcinfo` file, and uses the first `.jdi` file found.
- `--librarian-factory-path <directory>`: Comma separated librarian search path. Use `$` for default factory search path.
- `--librarian-path <directory>`: Comma separated librarian search path. Use `$` for default search path.
- `--log <filename>`: Create a debug log and write to specified file. Also logs debug information to stdout.
- `--script <filename>`: Run the specified Tcl script with optional arguments. Multiple `--script` options are allowed.
- `--set <name> <value>`: Set the setting called `<name>` to `<value>`. Multiple `--set` options are allowed.
- `--settings <filename>`: File name of an existing BSP settings file to update.
- `--silent`: Suppress information about the command's operation normally sent to stdout.
- `--sopc <filename>`: The `.sopcinfo` file to update the BSP with. It is recommended to create a new BSP if the design has changed significantly. This argument is useful if the path to the original `.sopcinfo` file has changed.
- `--verbose`: Output verbose, and default information about the command's operation to stdout.
- `--version`: Displays the version of this command and exits with a zero exit status.

Description

The `nios2-bsp-update-settings` command updates an existing Nios II `.bsp` file. The path to an existing `.bsp` file is a mandatory command-line option. The command modifies the settings file so the file must have write permissions. You might want to use the `--script` option to pass the default Tcl script to the `nios2-bsp-update-settings` command to make sure that your BSP is consistent with your system (this is what the `nios2-bsp` command does).

For more details about this command, use the `--extended-help` option to display comprehensive usage information.

nios2-lib-generate-makefile

Usage

```
nios2-lib-generate-makefile [--bsp-dir <directory>]
  [--debug] [--extended-help] [--help]
  [--lib-dir <directory>] [--lib-name <filename>]
  [--log <filename>] [--no-src]
  [--public-inc-dir <directory>] [--set <name> <value>]
  [--silent] [--src-dir <directory>]
  [--src-files <filenames>] [--src-rdir <directory>]
  [--use-lib-dir <directory>] [--verbose]
  [--version]
```

Options

- `--bsp-dir <directory>`: Path to the BSP generated files directory (populated using the **nios2-bsp-generate-files** command).
- `--debug`: Output debug, exception traces, verbose, and default information about the command's operation to stdout.
- `--extended-help`: Displays full information about this command and its options.
- `--help`: Displays basic information about this command and its options.
- `--lib-dir <directory>`: Destination directory for the user library archive file (**.a**), the user library makefile, and **public.mk**. If omitted, it defaults to the current directory.
- `--lib-name <filename>`: Name of the user library being created. The user library file name is the user library name with a **lib** prefix and **.a** suffix added. Do not include these in the user library name itself. If the user library name option is omitted, the user library name defaults to the name of the first source file with its extension removed.
- `--log <filename>`: Create a debug log and write to specified file. Also logs debug information to stdout.
- `--no-src`: Allows no sources files to be set in the Makefile. User must add source files in manually before compiling
- `--public-inc-dir <directory>`: Path to a directory that contains C header files (**.h**) that are to be made available (that is, public) to users of the user library. This directory is added to the appropriate variable in **public.mk**. Multiple `--public-inc-dir` options are allowed.
- `--set <name> <value>`: Set the makefile variable called `<name>` to `<value>`. If the variable exists in the managed section of the makefile, `<value>` replaces the default settings. It adds the makefile variable if it does not already exist. Multiple `--set` options are allowed.
- `--silent`: Suppress information about the command's operation normally sent to stdout.
- `--src-dir <directory>`: Search for source files in `<directory>`. Use `.` to look in the current directory. Multiple `--src-dir` options are allowed.

- `--src-files <filenames>`: A list of space-separated source file names added to the makefile. The list of file names is terminated by the next option or the end of the command line. Multiple `--src-files` options are allowed.
- `--src-rdir <directory>`: Same as `--src-dir` option but recursively search for source files in or under `<directory>`. Multiple `--src-rdir` options are allowed and can be freely mixed with `--src-dir` options.
- `--use-lib-dir <directory>`: Path to a dependent user library directory. The user library directory must contain a makefile fragment called **public.mk**. Multiple `--use-lib-dir` options are allowed.
- `--verbose`: Output verbose, and default information about the command's operation to stdout.
- `--version`: Displays the version of this command and exits with a zero exit status.

Description

The **nios2-lib-generate-makefile** command generates a user library makefile (called Makefile). The path to a BSP created by **nios2-bsp-generate-files** is an optional command-line option.

For more details about this command, use the `--extended-help` option to display comprehensive usage information.

nios2-bsp-editor

Usage

```
nios2-bsp-editor [--extended-help]
  [--fontsize <point size>] [--help]
  [--librarian-factory-path <directory>]
  [--librarian-path <directory>] [--log <filename>]
  [--settings <filename>] [--version]
```

Options

- `--extended-help`: Displays full information about this command and its options.
- `--fontsize <point size>`: The default point size for GUI fonts is 11. Use this option to adjust the point size.
- `--help`: Displays basic information about this command and its options.
- `--librarian-factory-path <directory>`: Comma separated librarian search path. Use \$ for default factory search path.
- `--librarian-path <directory>`: Comma separated librarian search path. Use \$ for default search path.
- `--log <filename>`: Create a debug log and write to specified file.
- `--settings <filename>`: File name of an existing BSP settings file to update.
- `--version`: Displays the version of this command and exits with a zero exit status.

Description

The `nios2-bsp-editor` command is a GUI application for creating and editing board support packages for Nios II designs.

For more details about this command, use the `--extended-help` option to display comprehensive usage information.

nios2-app-update-makefile

Usage

```
nios2-app-update-makefile --app-dir <directory>
  [--add-lib-dir <directory>] [--add-src-dir <directory>]
  [--add-src-files <filenames>] [--add-src-rdir <directory>] [--debug]
  [--extended-help] [--force] [--get <name>] [--get-all]
  [--get-asflags] [--get-bsp-dir] [--get-debug-level]
  [--get-defined-symbols] [--get-elf-name] [--get-optimization]
  [--get-undefined-symbols] [--get-user-flags] [--get-warnings]
  [--help] [--list-lib-dir] [--list-src-files] [--lock]
  [--log <filename>] [--no-src] [--remove-lib-dir <directory>]
  [--remove-src-dir <directory>] [--remove-src-files <filenames>]
  [--remove-src-rdir <directory>] [--set <name>]
  [--set-asflags <value>] [--set-bsp-dir <directory>]
  [--set-debug-level <value>] [--set-defined-symbols <value>]
  [--set-elf-name <name>] [--set-optimization <value>]
  [--set-undefined-symbols <value>] [--set-user-flags <value>]
  [--set-warnings <value>] [--show-managed-section] [--show-names]
  [--silent] [--unlock] [--verbose] [--version]
```

Options

- `--app-dir <directory>`: Path to the Application Directory with the generated makefile.
- `--add-lib-dir <directory>`: Add a path to dependent user library directory
- `--add-src-dir <directory>`: Add source files in `<directory>`. Use `.` to look in the current directory. Multiple `--add-src-dir` options are allowed.
- `--add-src-files <filenames>`: A list of space-separated source file names to be added to the makefile. The list of file names is terminated by the next option or the end of the command line. Multiple `--src-files` options are allowed.
- `--add-src-rdir <directory>`: Same as `--add-src-dir` option but recursively search for source files in or under `<directory>`. Multiple `--add-src-rdir` options are allowed and can be freely mixed with `--src-dir` options.
- `--debug`: Output debug, exception traces, verbose, and default information about the command's operation to stdout.
- `--extended-help` Displays full information about this command and its options.
- `--force`: Update the Makefile even if it's locked
- `--get <name>`: Get the values of Makefile variables
- `--get-all` Get all variables in the managed section of the Makefile
- `--get-asflags` Get user assembler flags
- `--get-bsp-dir` Get the BSP generated files directory

- `--get-debug-level` Get debug level flag
- `--get-defined-symbols` Get defined symbols flag
- `--get-elf-name` Get the name of .elf file
- `--get-optimization` Get optimization flag
- `--get-undefined-symbols` Get undefined symbols flag
- `--get-user-flags` Get user flags
- `--get-warnings` Get warnings flag
- `--help`: Displays basic information about this command and its options.
- `--list-lib-dir` List all paths to dependent user library directories
- `--list-src-files` List all source files in the makefile.
- `--lock`: Lock the Makefile to prevent it from being updated
- `--log <filename>`: Create a debug log and write to specified file. Also logs debug information to stdout.
- `--no-src` Remove all source files in the makefile
- `--remove-lib-dir <directory>`: Remove a path to dependent user library directory
- `--remove-src-dir <directory>`: Remove source files in *<directory>*. Use `.` to look in the current directory. Multiple `--remove-src-dir` options are allowed.
- `--remove-src-files <filenames>`: A list of space-separated source file names to be removed from the makefile. The list of file names is terminated by the next option or the end of the command line. Multiple `--src-files` options are allowed.
- `--remove-src-rdir <directory>`: Same as `--remove-src-dir` option but recursively search for source files in or under *<directory>*. Multiple `--remove-src-rdir` options are allowed and can be freely mixed with `--src-dir` options.
- `--set <name> <value>`: Set the value of a Makefile variable called *<name>*
- `--set-asflags <value>`: Set user assembler flags
- `--set-bsp-dir <directory>`: Set the BSP generated files directory
- `--set-debug-level <value>`: Set debug level flag
- `--set-defined-symbols <value>`: Set defined symbols flag
- `--set-elf-name <name>`: Set the name of .elf file
- `--set-optimization <value>`: Set optimization flag
- `--set-undefined-symbols <value>`: Set undefined symbols flag
- `--set-user-flags <value>`: Set user flags
- `--set-warnings <value>`: Set warnings flag
- `--show-managed-section` Show the managed section in the Makefile
- `--show-names` Show name of the variables

- `--silent`: Suppress information about the command's operation normally sent to stdout.
- `--unlock`: Unlock the Makefile
- `--verbose`: Output verbose, and default information about the command's operation to stdout.
- `--version`: Displays the version of this command and exits with a zero exit status.

Description

The `nios2-app-update-makefile` command updates an application makefile to add or remove source files.

For more details about this command, use the `--extended-help` option to display comprehensive usage information.



The `--add-src-dir`, `--add-src-rdir`, `--remove-src-dir`, and `--remove-src-rdir` options add and remove files found in `<directory>` at the time the command is executed. Files subsequently added to or removed from the directory are not reflected in the makefile.

nios2-lib-update-makefile

Usage

```
nios2-lib-update-makefile --lib-dir <directory>
  [--add-lib-dir <directory>] [--add-public-inc-dir <directory>]
  [--add-src-dir <directory>] [--add-src-files <filenames>]
  [--add-src-rdir <directory>] [--debug] [--extended-help] [--force]
  [--get <name>] [--get-all] [--get-asflags] [--get-bsp-dir]
  [--get-debug-level] [--get-defined-symbols] [--get-lib-name]
  [--get-optimization] [--get-undefined-symbols] [--get-user-flags]
  [--get-warnings] [--help] [--list-lib-dir] [--list-public-inc-dir]
  [--list-src-files] [--lock] [--log <filename>] [--no-src]
  [--remove-lib-dir <directory>] [--remove-public-inc-dir <directory>]
  [--remove-src-dir <directory>] [--remove-src-files <filenames>]
  [--remove-src-rdir <directory>] [--set <name> <value>]
  [--set-asflags <value>] [--set-bsp-dir <directory>]
  [--set-debug-level <value>] [--set-defined-symbols <value>]
  [--set-lib-name <name>] [--set-optimization <value>]
  [--set-undefined-symbols <value>] [--set-user-flags <value>]
  [--set-warnings <value>] [--show-managed-section] [--show-names]
  [--silent] [--unlock] [--verbose] [--version]
```

Options

- `--add-lib-dir <directory>`: Add a path to dependent user library directory
- `--add-public-inc-dir <directory>`: Add a directory that contains C-language header files
- `--add-src-dir <directory>`: Add source files in `<directory>`. Use `.` to look in the current directory. Multiple `--add-src-dir` options are allowed.
- `--add-src-files <filenames>`: A list of space-separated source file names to be added to the makefile. The list of file names is terminated by the next option or the end of the command line. Multiple `--src-files` options are allowed.

- `--add-src-rdir <directory>`: Same as `--add-src-dir` option but recursively search for source files in or under `<directory>`. Multiple `--add-src-rdir` options are allowed and can be freely mixed with `--src-dir` options.
- `--debug`: Output debug, exception traces, verbose, and default information about the command's operation to stdout.
- `--extended-help`: Displays full information about this command and its options.
- `--force`: Update the Makefile even if it is locked
- `--get <name>`: Get the values of Makefile variables
- `--get-all`: Get all variables in the managed section of the Makefile
- `--get-asflags`: Get user assembler flags
- `--get-bsp-dir`: Get the BSP generated files directory
- `--get-debug-level`: Get debug level flag
- `--get-defined-symbols`: Get defined symbols flag
- `--get-lib-name`: Get the name of user library
- `--get-optimization`: Get optimization flag
- `--get-undefined-symbols`: Get undefined symbols flag
- `--get-user-flags`: Get user flags
- `--get-warnings`: Get warnings flag
- `--help`: Displays basic information about this command and its options.
- `--list-lib-dir`: List all paths to dependent user library directories
- `--list-public-inc-dir`: List all public include directories
- `--list-src-files`: List all source files in the makefile.
- `--lock`: Lock the Makefile to prevent it from being updated
- `--log <filename>`: Create a debug log and write to specified file. Also logs debug information to stdout.
- `--no-src`: Remove all source files
- `--remove-lib-dir <directory>`: Remove a path to dependent user library directory
- `--remove-public-inc-dir <directory>`: Remove a include directory
- `--remove-src-dir <directory>`: Remove source files in `<directory>`. Use `.` to look in the current directory. Multiple `--remove-src-dir` options are allowed.
- `--remove-src-files <filenames>`: A list of space-separated source file names to be removed from the makefile. The list of file names is terminated by the next option or the end of the command line. Multiple `--src-files` options are allowed.
- `--remove-src-rdir <directory>`: Same as `--remove-src-dir` option but recursively search for source files in or under `<directory>`. Multiple `--remove-src-rdir` options are allowed and can be freely mixed with `--src-dir` options.
- `--set <name> <value>`: Set the value of a Makefile variable called `<name>`

- `--set-asflags <value>`: Set user assembler flags
- `--set-bsp-dir <directory>`: Set the BSP generated files directory
- `--set-debug-level <value>`: Set debug level flag
- `--set-defined-symbols <value>`: Set defined symbols flag
- `--set-lib-name <name>`: Set the name of user library
- `--set-optimization <value>`: Set optimization flag
- `--set-undefined-symbols <value>`: Set undefined symbols flag
- `--set-user-flags <value>`: Set user flags
- `--set-warnings <value>`: Set warnings flag
- `--show-managed-section`: Show the managed section in the Makefile
- `--show-names`: Show name of the variables
- `--silent`: Suppress information about the command's operation normally sent to stdout.
- `--unlock`: Unlock the Makefile
- `--verbose`: Output verbose, and default information about the command's operation to stdout.
- `--version`: Displays the version of this command and exits with a zero exit status.

Description

The `nios2-lib-update-makefile` command updates a user library makefile to add or remove source files.

For more details about this command, use the `--extended-help` option to display comprehensive usage information.



The `--add-src-dir`, `--add-src-rdir`, `--remove-src-dir`, and `--remove-src-rdir` options add and remove files found in `<directory>` at the time the command is executed. Files subsequently added to or removed from the directory are not reflected in the makefile.

nios2-swexample-create

Usage

```
nios2-create-swexample [--name] --sopc-dir --type [--list] [--app-dir]
  [--bsp-dir] [--no-app] [--no-bsp] [--elf-name] [--describe]
  [--describeX] [--describeAll] [--search] [--help] [--silent]
  [--version]
```

Options

- `--name`: Specify the name of the software project to create.
- `--sopc-dir` Specify the SOPC builder hardware design directory. Required.
- `--type`: Specify the software example design template type. Required.

- `--list`: List all valid software example design template types.
- `--app-dir`: Specify the application directory to create. Default: `<sopc-dir>/software_examples/app/<name>`
- `--bsp-dir`: Specify the bsp directory to create. Default: `<sopc-dir>/software_examples/bsp/<bsp-type>`
- `--no-app` Do not generate the **create-this-app** script
- `--no-bsp` Do not generate the **create-this-bsp** script
- `--elf-name` Name of the .elf file to create.
- `--describe`: Describe the software example type specified and exit.
- `--describeX`: Verbosely describe the software example type specified and exit.
- `--describeAll`: Describe all the software example types and exit.
- `--search`: Search for software example templates in the specified directory. Default: `<Nios II EDS install path>/examples/software`
- `--help`: Displays basic information about this command and its options.
- `--silent`: Do not echo commands.
- `--version`: Print the version number of nios2-create-swexample and exit.

Description

This utility creates a template software example for a given SOPC system.

nios2-elf-insert

Usage

```
nios2-elf-insert <filename> [--cpu_name <cpuName>]
  [--stderr_dev <stderrDev>] [--stdin_dev <stdinDev>]
  [--stdout_dev <stdoutDev>] [--sidp <sysidBase>] [--id <sysidHash>]
  [--timestamp <sysidTime>] [--sof <sofFile>]
  [--sopcinfo <sopcinfoFile>] [--jar <jarFile>] [--jdi <jdiFile>]
  [--quartus_project_dir <quartusProjectDir>]
  [--sopc_system_name <sopcSystemName>]
  [--profiling_enabled <profilingEnabled>]
  [--simulation_enabled <simulationEnabled>]
```

Options

- `<filename>`: the ELF filename
- `--cpu_name <cpuName>`
- `--stderr_dev <stderrDev>`
- `--stdin_dev <stdinDev>`
- `--stdout_dev <stdoutDev>`
- `--sidp <sysidBase>`
- `--id <sysidHash>`
- `--timestamp <sysidTime>`

- `--sof <sofFile>`
- `--sopcinfo <sopcinfoFile>`
- `--jar <jarFile>`
- `--jdi <jdiFile>`
- `--quartus_project_dir <quartusProjectDir>`
- `--sopc_system_name <sopcSystemName>`
- `--profiling_enabled <profilingEnabled>`
- `--simulation_enabled <simulationEnabled>`

nios2-elf-query

Usage

```
nios2-elf-query <filename> [--cpu_name] [--stderr_dev] [--stdin_dev]
  [--stdout_dev] [--sidp] [--id] [--timestamp] [--sof] [--sopcinfo]
  [--jar] [--jdi] [--quartus_project_dir] [--sopc_system_name]
  [--profiling_enabled] [--simulation_enabled]
```

Options

- `<filename>`: the ELF filename
- `--cpu_name`
- `--stderr_dev`
- `--stdin_dev`
- `--stdout_dev`
- `--sidp`
- `--id`
- `--timestamp`
- `--sof`
- `--sopcinfo`
- `--jar`
- `--jdi`
- `--quartus_project_dir`
- `--sopc_system_name`
- `--profiling_enabled`
- `--simulation_enabled`

nios2-convert-ide2sbt

Usage

```
nios2-convert-ide2sbt --input-dir --output-dir [--build-config]
  [--help] [--silent] [--version]
```

Options

- `--input-dir`: Specify the input application project directory to create. Required.
- `--output-dir`: Specify the directory to create your output projects. Required.
- `--build-config`: Specify the build configuration type: for example Release, Debug.
- `--help`: Print this message and exit.
- `--silent`: Suppress information about the command's operation normally sent to stdout.
- `--version`: Displays the version of this command and exits.

Description

Convert a Nios II IDE project to a Nios II SBT project.

nios2-flash-programmer-generate

Usage

```
nios2-flash-programmer-generate [--accept-bad-sysid]
  [--add-bin <fname> <flash-slave-desc> <offset>]
  [--add-elf <fname> <flash-slave-desc> <extra-elf2flash-arguments>]
  [--add-sof <fname> <flash-slave-desc> <offset>
  <extra-sof2flash-arguments>]
  [--cable <cable name>] [--cpu <processor_name>] [--debug]
  [--device <device name>] [--erase-first] [--extended-help]
  --flash-dir <directory> [--go] [--help] [--id <address>]
  [--instance <instance value>] [--log <filename>] [--mmu]
  [--program-flash] [--script-dir <directory>] [--sidp <address>]
  [--silent] --sopcinfo <filename> [--verbose] [--version]
```

Options

- `--accept-bad-sysid`: Continue even if the system identifier (ID) comparison fails.
- `--add-bin <fname> <flash-slave-desc> <offset>`: Specify a binary file to convert and program. The filename, target flash slave descriptor, and target offset amount are required. This option can be used multiple times for `.sof` files.

- `--add-elf <fname> <flash-slave-desc> <extra-elf2flash-arguments>`: Specify a `.elf` file to convert and program. The filename and target flash slave descriptor are required. This option can be used multiple times for `.elf` files. `<extra-elf2flash-arguments>` can be any of the following options supported by **elf2flash**:

- `save`
- `sim_optimize`

The following **elf2flash** options have default values computed, but are also supported as `<extra-elf2flash-arguments>` for manual override of those defaults:

- `base`
- `boot`
- `end`
- `reset`

- `--add-sof <fname> <flash-slave-desc> <offset> <extra-sof2flash-arguments>`: Specify a `.sof` file to convert and program. The filename, target flash slave descriptor, and target offset arguments are required. This option can be used multiple times for `.sof` files. `<extra-sof2flash-arguments>` can be any of the following options supported by **sof2flash**:

- `activeparallel`
- `compress`
- `save`
- `timestamp`
- `options`

- `--cable <cable name>`: Specifies which JTAG cable to use (not needed if you only have one cable). Not used without `--program-flash` option.
- `--cpu <processor_name>`: The SOPC Builder system file Nios II processor name. Not required if only one Nios II processor in the system.
- `--debug`: Sends debug information, exception traces, verbose output, and default information about the command's operation, to stdout.
- `--device <device name>`: Specifies in which device you want to look for the Nios II debug core. Device 1 is the device nearest TDI. Not used without `--program-flash` option.
- `--erase-first`: Erase entire flash targets before programming them. Not used without `--program-flash` option.
- `--extended-help`: Displays full information about this command and its options.
- `--flash-dir <directory>`: Path to the directory where the flash files are generated. Use `.` for the current directory. This command overwrites pre-existing files in `<directory>` without warning.
- `--go`: Run processor from reset vector after program.
- `--help`: Displays basic information about this command and its options.

- `--id <address>`: Unique ID code for target system. Not used without `--program-flash` option.
- `--instance <instance value>`: Specifies the INSTANCE value of the debug core (not needed if there is exactly one on the chain). Not used without `--program-flash` option.
- `--log <filename>`: Creates a debug log and write to specified file. Also logs debug information to stdout.
- `--mmu`: Specifies if the processor with the corresponding INSTANCE value has an MMU (not needed if there is exactly one processor in the system). Not used without `--program-flash` option.
- `--program-flash`: Providing this flag causes calls to **nios2-flash-programmer** to be generated and executed. This results in flash targets being programmed.
- `--script-dir <directory>`: Path to the directory where a shell script of this tool's executed command lines is generated. This script can be used in place of this **nios2-flash-programmer-generate** command. Use `.` for the current directory. This command overwrites pre-existing files in `<directory>` without warning.
- `--sidp <address>`: Base address of system ID peripheral on the target. Not used without `--program-flash` option.
- `--silent`: Suppresses information about the command's operation normally sent to stdout.
- `--sopcinfo <filename>`: The **.sopcinfo** file.
- `--verbose`: Sends verbose output, and default information about the command's operation, to stdout.
- `--version`: Displays the version of this command and exits with a zero exit status.

Description

The **nios2-flash-programmer-generate** command converts multiple files to a **.flash** in Motorola S-record format, and programs them to the designated target flash devices (`--program-flash`). This is a convenience utility that manages calls to the following command line utilities

- **bin2flash**
- **elf2flash**
- **sof2flash**
- **nios2-flash-programmer**

This utility also generates a script that captures the sequence of conversion and flash programmer commands.

Example

```
nios2-flash-programmer-generate --sopcinfo=C:\my_design.sopcinfo \  
  --flash-dir=. \  
  --add-sof C:\my_design\test.sof 0x0C000000 memory_0 compress save \  
  --add-elf C:\my_app\my_app.elf 0x08000000 memory_0 \  
  --program-flash
```

nios2-c2h-generate-makefile

Usage

```
nios2-c2h-generate-makefile --sopc=<sopc>
  --app-dir=<dir>
  --accelerator=<function>
  --enable_quartus=<0/1>
  --analyze_only=<0/1>
  --use_existing_accelerators=<0/1>
```

Options

- `--sopc`: The path to the `.sopcinfo` file.
- `--app-dir`: Directory to place the application Makefile and executable file. If omitted, it defaults to the current directory.
- `--accelerator`: Specifies a function to be accelerated.
- `--enable_quartus`: Building the application compiles the associated Quartus II project. Defaults to 0.
- `--analyze_only`: Disables hardware generation, SOPC Builder system generation, and Quartus II compilation for all accelerators in the application. Building the project with this option only updates the report files. Defaults to 0.
- `--use_existing_accelerators`: Disables all hardware generation steps. The build behaves as if `c2h.mk` did not exist, with the exception of possible accelerator linking as specified in the `--accelerator` option. Defaults to 0.

Description

The `nios2-c2h-generate-makefile` command creates a C2H makefile fragment, `c2h.mk`, that specifies all accelerators and accelerator options for an application.

This command creates a new `c2h.mk` each time it is called, overwriting the existing `c2h.mk`

The `--accelerator` argument specifies a function to be accelerated. This argument accepts up to four comma-separated values:

- Target function name.
- Target function file.
- Link hardware accelerator instead of original software. 1 or 0. Defaults to 1.
- Flush data cache before each call. 1 or 0. Defaults to 1.



The `nios2-c2h-generate-makefile` script is available to support pre-existing command-line C2H projects. Create new C2H projects using the Nios II IDE.

Example

```
nios2-c2h-generate-makefile \
  --sopc=../../NiosII_stratix_ls40_standard.sopcinfo \
  --app_dir=./ \
  --accelerator=filter,filter.c \
  --accelerator=xmath,../../xmath.c,1,0 \
  --use_existing_accelerators
```

nios2-bsp

Usage

```
nios2-bsp <bsp-type> <bsp-dir> [<sopc>] [<override>]...
```

Options

- <bsp-type>: hal or ucosii.
- <bsp-dir>: Path to the BSP directory.
- <sopc>: The path to the .sopcinfo file or its directory.
- <override>: Options to override defaults.

Description

The **nios2-bsp** script calls **nios2-bsp-create-settings** or **nios2-bsp-update-settings** to create or update a BSP settings file, and the **nios2-bsp-generate-files** command to create the BSP files. The Nios II Embedded Design Suite (EDS) supports the following BSP types:

- hal
- ucosii

BSP type names are case-insensitive.

This utility produces a BSP of <bsp-type> in <bsp-dir>. If the BSP does not exist, it is created. If the BSP already exists, it is updated to be consistent with the associated hardware system.

The default Tcl script is used to set the following system-dependent settings:

- stdio character device
- System timer device
- Default linker memory
- Boot loader status (enabled or disabled)

If the BSP already exists, **nios2-bsp** overwrites these system-dependent settings.

The default Tcl script resides at:

```
<Nios II EDS install path>/sdk2/bin/bsp-set-defaults.tcl
```

When creating a new BSP, this utility runs **nios2-bsp-create-settings**, which creates **settings.bsp** in <bsp-dir>.

When updating an existing BSP, this utility runs **nios2-bsp-update-settings**, which updates **settings.bsp** in <bsp-dir>.

After creating or updating the **settings.bsp** file, this utility runs **nios2-bsp-generate-files**, which generates files in <bsp-dir>

Required arguments:

- `<bsp-type>`: Specifies the type of BSP. This argument is ignored when updating a BSP. This argument is case-insensitive. The **nios2-bsp** script supports the following values of `<bsp-type>`:
 - hal
 - ucosii
- `<bsp-dir>`: Path to the BSP directory. Use `“.”` to specify the current directory.


Optional arguments:

- `<sopc>`: The path name of the **.sopcinfo** file. Alternatively, specify a directory containing a **.sopcinfo** file. In the latter case, the tool finds a file with the extension **.sopcinfo**. This argument is ignored when updating a BSP. If you omit this argument, it defaults to the current directory.
- `<override>`: Options to override defaults. The **nios2-bsp** script passes most overrides to **nios2-bsp-create-settings** or **nios2-bsp-update-settings**. It also passes the `--silent`, `--verbose`, `--debug`, and `--log` options to **nios2-bsp-generate-files**.

nios2-bsp passes the following overrides to the default Tcl script:

- `--default_stdio <device>|none|DONT_CHANGE`
Specifies stdio device.
- `--default_sys_timer <device>|none|DONT_CHANGE`
Specifies system timer device.
- `--default_memory_regions DONT_CHANGE`
Suppresses creation of new default memory regions when updating a BSP. Do not use this option when creating a new BSP.
- `--default_sections_mapping <region>|DONT_CHANGE`
Specifies the memory region for the default sections.
- `--use_bootloader 0|1|DONT_CHANGE`
Specifies whether a boot loader is required.

On a preexisting BSP, the value `DONT_CHANGE` prevents associated settings from changing their current value.

 The `“--”` prefix is stripped when the option is passed to the underlying utility.

Nios II Example Design Scripts

The Nios II SBT includes scripts that allow you to create sample BSPs and applications. This section describes each script and its location in the example design directory structure. Each hardware example design in the Nios II EDS includes a **software_examples** directory with **app** and **bsp** subdirectories.

The **bsp** subdirectory contains a variety of example BSPs. Table 15-3 lists all potential BSP examples provided in the **bsp** directory. The **bsp** directory for each hardware example only includes BSP examples supported by the associated hardware example.

Table 15-3. BSP Examples

Example BSP (1)	Summary
hal_reduced_footprint	Hardware abstraction layer (HAL) BSP configured to minimize memory footprint
hal_default	HAL BSP configured with all defaults
hal_zipfs	HAL BSP configured with the Altera® read-only zip file system
ucosii_net	MicroC/OS-II BSP configured with networking
ucosii_net_zipfs	MicroC/OS-II BSP configured with networking and the Altera read-only zip file system
ucosii_net_tse	MicroC/OS-II BSP configured with networking support for the Altera triple-speed Ethernet media access control (MAC)
ucosii_net_tse_zipfs	MicroC/OS-II BSP configured with networking support for the Altera triple-speed Ethernet MAC and the Altera read-only zip file system
ucosii_default	MicroC/OS-II BSP configured with all defaults

Note to Table 15-3:

(1) Some BSP examples might not be available on some hardware examples.

The **app** subdirectory contains a separate subdirectory for each software example supported by the hardware example, as listed in Table 15-4.

Table 15-4. Application Examples (1)

Application Name	Summary
Hello World	Prints "Hello from Nios II"
Board Diagnostics	Tests peripherals on the development boards
Count Binary	Displays a running count of 0x00 to 0xff
Hello Freestanding	Prints "Hello from Nios II" from a free-standing application
Hello MicroC/OS-II	Prints "Hello from Nios II" using the MicroC/OS-II RTOS
Hello World Small	Prints "Hello from Nios II" from a small footprint program
Memory Test	Runs diagnostic tests on both volatile and flash memory
Memory Test Small	Runs diagnostic tests on volatile memory from a small footprint
Simple Socket Server	Runs a TCP/IP socket server
Web Server	Runs a web server from a file system in flash memory
Zip File System	Reads from a file system in flash memory

Note to Table 15-4:

(1) Some application examples might not be available on some hardware examples, depending on BSP support.

create-this-bsp

Each BSP subdirectory contains a **create-this-bsp** script. The **create-this-bsp** script calls the **nios2-bsp** script to create a BSP in the current directory. The **create-this-bsp** script has a relative path to the directory containing the **.sopcinfo** file. The **.sopcinfo** file resides two directory levels above the directory containing the **create-this-bsp** script.

The **create-this-bsp** script takes no command-line arguments. Your current directory must be the same directory as the **create-this-bsp** script. The exit value is zero on success and one on error.

create-this-app

Each application subdirectory contains a **create-this-app** script. The **create-this-app** script copies the C/C++ application source code to the current directory, runs **nios2-app-generate-makefile** to create a makefile (named **Makefile**), and then runs **make** to build the Executable and Linking Format File (**.elf**) for your application. Each **create-this-app** script uses a particular example BSP. For further information, refer to the script to determine the associated example BSP. If the BSP does not exist when **create-this-app** runs, **create-this-app** calls the associated **create-this-bsp** script to create the BSP.

The **create-this-app** script takes no command-line arguments. Your current directory must be the same directory as the **create-this-app** script. The exit value is zero on success and one on error.

Finding create-this-app and create-this-bsp

The **create-this-app** and **create-this-bsp** scripts are installed with your Nios II example designs. You can easily find them from the following information:

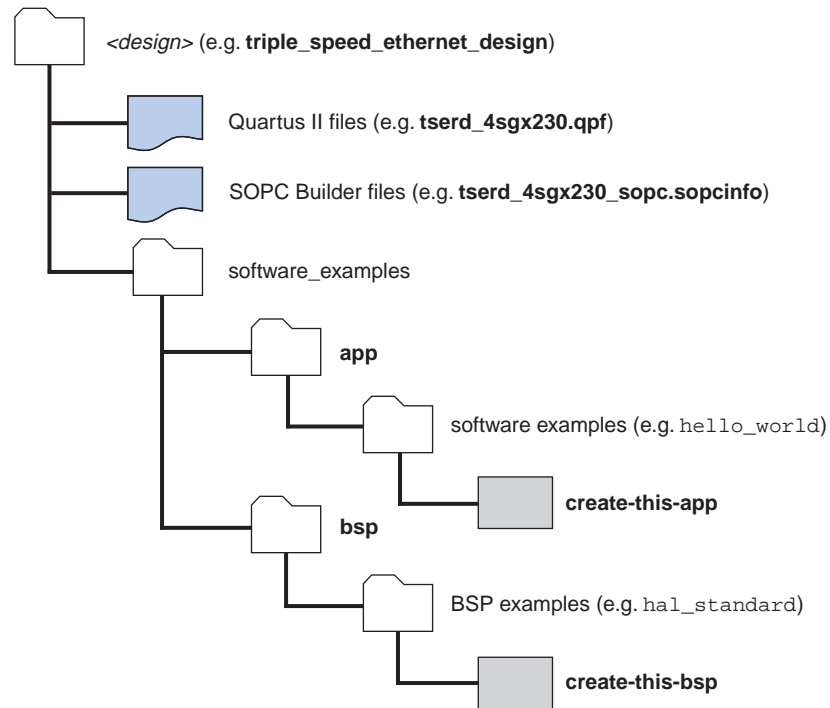
- Where the Nios II EDS is installed
- Which Altera® development board you are using
- Which HDL you are using
- Which Nios II hardware example design you are using
- The name of the Nios II software example

The **create-this-app** script for each software example design is in `<Nios II EDS install path>/examples/<HDL>/niosII_<board type>/<design name>/software_examples/app/<example name>`. For example, the **create-this-app** script for the **Hello World** software example running on the triple-speed ethernet example design for the Stratix® IV GX FPGA Development Kit might be located in `c:/altera/100/nios2eds/examples/verilog/niosII_stratixIV_4sgx230/triple_speed_ethernet_design/software_examples/app/hello_world`.

Similarly, the **create-this-bsp** script for each software example design is in `<Nios II EDS install path>/examples/<HDL>/niosII_<board type>/<design name>/software_examples/bsp/<BSP_type>`. For example, the **create-this-bsp** script for the basic HAL BSP to run on the triple-speed ethernet example design for the Stratix IV GX FPGA Development Kit might be located in `c:/altera/100/nios2eds/examples/verilog/niosII_stratixIV_4sgx230/triple_speed_ethernet_design/software_examples/bsp/hal_default`.

Figure 15-1 shows the directory structure under each hardware example design.

Figure 15-1. Software Example Design Directory Structure



Settings

Settings are central to how you create and work with BSPs, software packages, and device drivers. You control the characteristics of your project by controlling the settings. The settings determine things like whether or not an operating system is supported, and the device drivers and other that packages are included.

Every example in the *Getting Started from the Command Line* and *Nios II Software Build Tools* chapters of the *Nios II Software Developer's Handbook* involves specifying or manipulating settings. Sometimes these settings are specified automatically, by scripts such as **create-this-bsp**, and sometimes explicitly, with Tcl commands. Either way, settings are always involved.

This section contains a complete list of available settings for BSPs and for Altera-supported device drivers and software packages. It does not include settings for device drivers or software packages furnished by Altera partners or other third parties. If you are using a third-party driver or component, refer to the supplier's documentation.

Settings used in the Nios II SBT are organized hierarchically, for logical grouping and to avoid name conflicts. Each setting's position in the hierarchy is indicated by one or more prefixes. A prefix is an identifier followed by a dot (.). For example, `hal.enable_c_plus_plus` is a hardware abstraction layer (HAL) setting, while `ucosii.event_flag.os_flag_accept_en` is a member of the event flag subgroup of MicroC/OS-II settings.

Setting names are case-insensitive.

Overview of BSP Settings

A BSP setting consists of a name/value pair.

The format in which you specify the setting value depends on the setting type. Several settings types are supported. [Table 15-5](#) shows the allowed formats for each setting type.

Table 15-5. Setting Formats

Setting Type	Format When Setting	Format When Getting
boolean	0/1 or false/true	0/1
number	0x prefix for hexadecimal or no prefix for a decimal number	decimal
string	Quoted string Use "none" to set empty string. In the SBT command line, to specify a string value with embedded spaces, surround the string with a backslash-double-quote sequence (\"). For example: <code>--set APP_INCLUDE_DIRS \"lcd board\"</code>	Quoted string

There are several contexts for BSP settings, as shown in [Table 15-6](#).

Table 15-6. BSP Setting Contexts

Setting Context	Description
Altera HAL	Settings available with the Altera HAL BSP or any BSP based on it (for example, Micrium MicroC/OS-II).
Micrium MicroC/OS-II	Settings available if using the Micrium MicroC/OS-II BSP. All Altera HAL BSP settings are also available because MicroC/OS-II is based on the Altera HAL BSP.
Altera BSP Makefile Generator	Settings available if using the Altera BSP makefile generator (generates the Makefile and public.mk files). These settings control the contents of makefile variables. This generator is always present in Altera HAL BSPs or any BSPs based on the Altera HAL.
Altera BSP Linker Script Generator	Settings available if using the Altera BSP linker script generator (generates the linker.x and linker.h files). This generator is always present in Altera HAL BSPs or any BSPs based on the Altera HAL.

Do not confuse BSP settings with BSP Tcl commands. This section covers BSP settings, including their types, meanings, and legal values. The Tcl commands, which include tools for manipulating the settings, are covered in [“Tcl Commands for BSP Settings” on page 15-72](#).

Overview of Component and Driver Settings

The Nios II EDS includes a number of standard software packages and device drivers. All of the software packages, and several drivers, have settings that you can manipulate when creating a BSP. This section lists the packages and drivers that have settings.

You can enable a software package or driver in the Nios II BSP Editor. In the SBT command line flow, use the `enable_sw_package` Tcl command, described in [“Tcl Commands for BSP Settings” on page 15-72](#).

Altera Host-Based File System Settings

The Altera host-based file system has one setting. If the Altera host-based file system is enabled, you must debug (not run) applications based on the BSP for the host-based file system to function. The host-based file system relies on the GNU debugger running on the host to provide host-based file operations.

The host-based file system enables debugging information in your project, by setting `APP_CFLAGS_OPTIMIZATION` to `-g` in the makefile.

To include the host-based file system in your BSP, enable the `altera_hostfs` software package.

Altera Read-Only Zip File System Settings

The Altera read-only zip file system has several settings. If the read-only zip file system is enabled, it adds `-DUSE_RO_ZIPFS` to `ALT_CPPFLAGS` in **public.mk**.

To include the read-only zip file system in your BSP, enable the `altera_ro_zipfs` software package.

Altera NicheStack TCP/IP - Nios II Edition Stack Settings

The Altera NicheStack[®] TCP/IP Network Stack - Nios II Edition has several settings. The stack is only available in MicroC/OS-II BSPs. If the NicheStack TCP/IP stack is enabled, it adds `-DALT_INICHE` to `ALT_CPPFLAGS` in **public.mk**.

To include the NicheStack TCP/IP networking stack in your BSP, enable the `altera_iniche` software package.

Altera Avalon-MM JTAG UART Driver Settings

The Altera Avalon Memory-Mapped[®] (Avalon-MM) JTAG UART driver settings are available if the `altera_avalon_jtag_uart` driver is present. By default, this driver is used if your SOPC Builder system has an `altera_avalon_jtag_uart` module connected to it.

Altera Avalon-MM UART Driver Settings

The Altera Avalon-MM UART driver settings are available if the `altera_avalon_uart` driver is present. By default, this driver is used if your SOPC Builder system has an `altera_avalon_uart` module connected to it.

Settings Reference

This section lists all settings for BSPs, software packages, and device drivers.

hal.enable_instruction_related_exceptions_api

Identifier	none
Type	Boolean definition
Default Value	false
Destination File	none
Description	Enables application program interface (API) for registering handlers to service instruction-related exceptions. These exception types can be generated if various processor options are enabled, such as the memory management unit (MMU), memory protection unit (MPU), or other advanced exception types. Enabling this setting increases the size of the exception entry code.
Restrictions	none

hal.max_file_descriptors

Identifier	none
Type	Decimal number
Default Value	32
Destination File	none
Description	Determines the number of file descriptors statically allocated.
Restrictions	If hal.enable_lightweight_device_driver_api is true, there are no file descriptors so this setting is ignored. If hal.enable_lightweight_device_driver_api is false, this setting must be at least 4 because HAL needs a file descriptor for /dev/null, /dev/stdin, /dev/stdout, and /dev/stderr. This setting defines the value of ALT_MAX_FD in system.h .

hal.exclude_default_exception

Identifier	ALT_EXCLUDE_DEFAULT_EXCEPTION
Type	Boolean definition
Default Value	false
Destination File	system.h
Description	Excludes default exception vector. If true, this setting defines the macro ALT_EXCLUDE_DEFAULT_EXCEPTION in system.h.
Restrictions	none

hal.sys_clk_timer

Identifier	none
Type	Unquoted string
Default Value	none
Destination File	none
Description	Slave descriptor of the system clock timer device. This device provides a periodic interrupt (“tick”) and is typically required for RTOS use. This setting defines the value of ALT_SYS_CLK in system.h .
Restrictions	none

hal.timestamp_timer

Identifier	none
Type	Unquoted string
Default Value	none
Destination File	none
Description	Slave descriptor of timestamp timer device. This device is used by Altera HAL timestamp drivers for high-resolution time measurement. This setting defines the value of ALT_TIMESTAMP_CLK in system.h .
Restrictions	none

ucosii.os_max_tasks

Identifier	OS_MAX_TASKS
Type	Decimal number
Default Value	10
Destination File	system.h
Description	Maximum number of tasks
Restrictions	none

ucosii.os_lowest_prio

Identifier	OS_LOWEST_PRIO
Type	Decimal number
Default Value	20
Destination File	system.h
Description	Lowest assignable priority
Restrictions	none

ucosii.os_thread_safe_newlib

Identifier	OS_THREAD_SAFE_NEWLIB
Type	Boolean assignment
Default Value	1
Destination File	system.h
Description	Thread safe C library
Restrictions	none

ucosii.miscellaneous.os_arg_chk_en

Identifier	OS_ARG_CHK_EN
Type	Boolean assignment
Default Value	1
Destination File	system.h
Description	Enable argument checking
Restrictions	none

ucosii.miscellaneous.os_cpu_hooks_en

Identifier	OS_CPU_HOOKS_EN
Type	Boolean assignment
Default Value	1
Destination File	system.h
Description	Enable MicroC/OS-II hooks
Restrictions	none

ucosii.miscellaneous.os_debug_en

Identifier	OS_DEBUG_EN
Type	Boolean assignment
Default Value	1
Destination File	system.h
Description	Enable debug variables
Restrictions	none

ucosii.miscellaneous.os_sched_lock_en

Identifier	OS_SCHED_LOCK_EN
Type	Boolean assignment
Default Value	1
Destination File	system.h
Description	Include code for OSSchedLock() and OSSchedUnlock()
Restrictions	none

ucosii.miscellaneous.os_task_stat_en

Identifier	OS_TASK_STAT_EN
Type	Boolean assignment
Default Value	1
Destination File	system.h
Description	Enable statistics task
Restrictions	none

ucosii.miscellaneous.os_task_stat_stk_chk_en

Identifier	OS_TASK_STAT_STK_CHK_EN
Type	Boolean assignment
Default Value	1
Destination File	system.h
Description	Check task stacks from statistics task
Restrictions	none

ucosii.miscellaneous.os_tick_step_en

Identifier	OS_TICK_STEP_EN
Type	Boolean assignment
Default Value	1
Destination File	system.h
Description	Enable tick stepping feature for uCOS-View
Restrictions	none

ucosii.miscellaneous.os_event_name_size

Identifier	OS_EVENT_NAME_SIZE
Type	Decimal number
Default Value	32
Destination File	system.h
Description	Size of name of Event Control Block groups
Restrictions	none

ucosii.miscellaneous.os_max_events

Identifier	OS_MAX_EVENTS
Type	Decimal number
Default Value	60
Destination File	system.h
Description	Maximum number of event control blocks
Restrictions	none

ucosii.miscellaneous.os_task_idle_stk_size

Identifier	OS_TASK_IDLE_STK_SIZE
Type	Decimal number
Default Value	512
Destination File	system.h
Description	Idle task stack size
Restrictions	none

ucosii.miscellaneous.os_task_stat_stk_size

Identifier	OS_TASK_STAT_STK_SIZE
Type	Decimal number
Default Value	512
Destination File	system.h
Description	Statistics task stack size
Restrictions	none

ucosii.task.os_task_change_prio_en

Identifier	OS_TASK_CHANGE_PRIO_EN
Type	Boolean assignment
Default Value	1
Destination File	system.h
Description	Include code for OSTaskChangePrio()
Restrictions	none

ucosii.task.os_task_create_en

Identifier	OS_TASK_CREATE_EN
Type	Boolean assignment
Default Value	1
Destination File	system.h
Description	Include code for OSTaskCreate()
Restrictions	none

ucosii.task.os_task_create_ext_en

Identifier	OS_TASK_CREATE_EXT_EN
Type	Boolean assignment
Default Value	1
Destination File	system.h
Description	Include code for OSTaskCreateExt()
Restrictions	none

ucosii.task.os_task_del_en

Identifier	OS_TASK_DEL_EN
Type	Boolean assignment
Default Value	1
Destination File	system.h
Description	Include code for OSTaskDel()
Restrictions	none

ucosii.task.os_task_name_size

Identifier	OS_TASK_NAME_SIZE
Type	Decimal number
Default Value	32
Destination File	system.h
Description	Size of task name
Restrictions	none

ucosii.task.os_task_profile_en

Identifier	OS_TASK_PROFILE_EN
Type	Boolean assignment
Default Value	1
Destination File	system.h
Description	Include data structure for run-time task profiling
Restrictions	none

ucosii.task.os_task_query_en

Identifier	OS_TASK_QUERY_EN
Type	Boolean assignment
Default Value	1
Destination File	system.h
Description	Include code for OSTaskQuery
Restrictions	none

ucosii.task.os_task_suspend_en

Identifier	OS_TASK_SUSPEND_EN
Type	Boolean assignment
Default Value	1
Destination File	system.h
Description	Include code for OSTaskSuspend() and OSTaskResume()
Restrictions	none

ucosii.task.os_task_sw_hook_en

Identifier	OS_TASK_SW_HOOK_EN
Type	Boolean assignment
Default Value	1
Destination File	system.h
Description	Include code for OSTaskSwHook()
Restrictions	none

ucosii.time.os_time_tick_hook_en

Identifier	OS_TIME_TICK_HOOK_EN
Type	Boolean assignment
Default Value	1
Destination File	system.h
Description	Include code for OSTimeTickHook()
Restrictions	none

ucosii.time.os_time_dly_resume_en

Identifier	OS_TIME_DLY_RESUME_EN
Type	Boolean assignment
Default Value	1
Destination File	system.h
Description	Include code for OSTimeDlyResume()
Restrictions	none

ucosii.time.os_time_dly_hmsm_en


Identifier	OS_TIME_DLY_HMSM_EN
Type	Boolean assignment
Default Value	1
Destination File	system.h
Description	Include code for OSTimeDlyHMSM()
Restrictions	none

ucosii.time.os_time_get_set_en

Identifier	OS_TIME_GET_SET_EN
Type	Boolean assignment
Default Value	1
Destination File	system.h
Description	Include code for OSTimeGet and OSTimeSet()
Restrictions	none


ucosii.os_flag_en

Identifier	OS_FLAG_EN
Type	Boolean assignment
Default Value	1
Destination File	system.h
Description	Enable code for Event Flags.


 This setting is enabled by default in MicroC-OS/II BSPs, because it is required for correct functioning of Altera device drivers and the HAL in a multithreaded environment. Avoid disabling it.

Restrictions	none
---------------------	------


ucosii.event_flag.os_flag_wait_clr_en**Identifier** OS_FLAG_WAIT_CLR_EN**Type** Boolean assignment**Default Value** 1**Destination File** system.h**Description** Include code for Wait on Clear Event Flags.

 This setting is enabled by default in MicroC-OS/II BSPs, because it is required for correct functioning of Altera device drivers and the HAL in a multithreaded environment. Avoid disabling it.


Restrictions none**ucosii.event_flag.os_flag_accept_en****Identifier** OS_FLAG_ACCEPT_EN**Type** Boolean assignment**Default Value** 1**Destination File** system.h**Description** Include code for OSFlagAccept().

 This setting is enabled by default in MicroC-OS/II BSPs, because it is required for correct functioning of Altera device drivers and the HAL in a multithreaded environment. Avoid disabling it.

Restrictions none**ucosii.event_flag.os_flag_del_en****Identifier** OS_FLAG_DEL_EN**Type** Boolean assignment**Default Value** 1**Destination File** system.h**Description** Include code for OSFlagDel().

 This setting is enabled by default in MicroC-OS/II BSPs, because it is required for correct functioning of Altera device drivers and the HAL in a multithreaded environment. Avoid disabling it.

Restrictions none**ucosii.event_flag.os_flag_query_en****Identifier** OS_FLAG_QUERY_EN**Type** Boolean assignment**Default Value** 1**Destination File** system.h**Description** Include code for OSFlagQuery().

 This setting is enabled by default in MicroC-OS/II BSPs, because it is required for correct functioning of Altera device drivers and the HAL in a multithreaded environment. Avoid disabling it.

Restrictions none

ucosii.event_flag.os_flag_name_size

Identifier	OS_FLAG_NAME_SIZE
Type	Decimal number
Default Value	32
Destination File	system.h
Description	Size of name of Event Flags group. CAUTION: This is required by the HAL and many Altera device drivers; use caution in reducing this value.
Restrictions	none

ucosii.event_flag.os_flags_nbits

Identifier	OS_FLAGS_NBITS
Type	Decimal number
Default Value	16
Destination File	system.h
Description	Event Flag bits (8,16,32). CAUTION: This is required by the HAL and many Altera device drivers; use caution in changing this value.
Restrictions	none

ucosii.event_flag.os_max_flags

Identifier	OS_MAX_FLAGS
Type	Decimal number
Default Value	20
Destination File	system.h
Description	Maximum number of Event Flags groups. CAUTION: This is required by the HAL and many Altera device drivers; use caution in reducing this value.
Restrictions	none

ucosii.os_mutex_en

Identifier	OS_MUTEX_EN
Type	Boolean assignment
Default Value	1
Destination File	system.h
Description	Enable code for Mutex Semaphores
Restrictions	none

ucosii.mutex.os_mutex_accept_en

Identifier	OS_MUTEX_ACCEPT_EN
Type	Boolean assignment
Default Value	1
Destination File	system.h
Description	Include code for OSMutexAccept()
Restrictions	none

ucosii.mutex.os_mutex_del_en


Identifier	OS_MUTEX_DEL_EN
Type	Boolean assignment
Default Value	1
Destination File	system.h
Description	Include code for OSMutexDel()
Restrictions	none

ucosii.mutex.os_mutex_query_en

Identifier	OS_MUTEX_QUERY_EN
Type	Boolean assignment
Default Value	1
Destination File	system.h
Description	Include code for OSMutexQuery
Restrictions	none

ucosii.os_sem_en

Identifier	OS_SEM_EN
Type	Boolean assignment
Default Value	1
Destination File	system.h
Description	Enable code for semaphores.

 This setting is enabled by default in MicroC-OS/II BSPs, because it is required for correct functioning of Altera device drivers and the HAL in a multithreaded environment. Avoid disabling it.

Restrictions	none
---------------------	------

ucosii.semaphore.os_sem_accept_en


Identifier OS_SEM_ACCEPT_EN

Type Boolean assignment

Default Value 1

Destination File system.h

Description Include code for OSSemAccept().

 This setting is enabled by default in MicroC-OS/II BSPs, because it is required for correct functioning of Altera device drivers and the HAL in a multithreaded environment. Avoid disabling it.

Restrictions none

ucosii.semaphore.os_sem_set_en


Identifier OS_SEM_SET_EN

Type Boolean assignment

Default Value 1

Destination File system.h

Description Include code for OSSemSet().

 This setting is enabled by default in MicroC-OS/II BSPs, because it is required for correct functioning of Altera device drivers and the HAL in a multithreaded environment. Avoid disabling it.

Restrictions none

ucosii.semaphore.os_sem_del_en


Identifier OS_SEM_DEL_EN

Type Boolean assignment

Default Value 1

Destination File system.h

Description Include code for OSSemDel().

 This setting is enabled by default in MicroC-OS/II BSPs, because it is required for correct functioning of Altera device drivers and the HAL in a multithreaded environment. Avoid disabling it.

Restrictions none

ucosii.semaphore.os_sem_query_en


Identifier OS_SEM_QUERY_EN

Type Boolean assignment

Default Value 1

Destination File system.h

Description Include code for OSSemQuery().

 This setting is enabled by default in MicroC-OS/II BSPs, because it is required for correct functioning of Altera device drivers and the HAL in a multithreaded environment. Avoid disabling it.

Restrictions none

ucosii.os_mbox_en

Identifier	OS_MBOX_EN
Type	Boolean assignment
Default Value	1
Destination File	system.h
Description	Enable code for mailboxes
Restrictions	none

ucosii.mailbox.os_mbox_accept_en

Identifier	OS_MBOX_ACCEPT_EN
Type	Boolean assignment
Default Value	1
Destination File	system.h
Description	Include code for OSMboxAccept()
Restrictions	none

ucosii.mailbox.os_mbox_del_en

Identifier	OS_MBOX_DEL_EN
Type	Boolean assignment
Default Value	1
Destination File	system.h
Description	Include code for OSMboxDel()
Restrictions	none

ucosii.mailbox.os_mbox_post_en

Identifier	OS_MBOX_POST_EN
Type	Boolean assignment
Default Value	1
Destination File	system.h
Description	Include code for OSMboxPost()
Restrictions	none

ucosii.mailbox.os_mbox_post_opt_en

Identifier	OS_MBOX_POST_OPT_EN
Type	Boolean assignment
Default Value	1
Destination File	system.h
Description	Include code for OSMboxPostOpt()
Restrictions	none

ucosii.mailbox.os_mbox_query_en

Identifier	OS_MBOX_QUERY_EN
Type	Boolean assignment
Default Value	1
Destination File	system.h
Description	Include code for OSMboxQuery()
Restrictions	none

ucosii.os_q_en

Identifier	OS_Q_EN
Type	Boolean assignment
Default Value	1
Destination File	system.h
Description	Enable code for Queues
Restrictions	none

ucosii.queue.os_q_accept_en

Identifier	OS_Q_ACCEPT_EN
Type	Boolean assignment
Default Value	1
Destination File	system.h
Description	Include code for OSQAccept()
Restrictions	none

ucosii.queue.os_q_del_en

Identifier	OS_Q_DEL_EN
Type	Boolean assignment
Default Value	1
Destination File	system.h
Description	Include code for OSQDel()
Restrictions	none

ucosii.queue.os_q_flush_en

Identifier	OS_Q_FLUSH_EN
Type	Boolean assignment
Default Value	1
Destination File	system.h
Description	Include code for OSQFlush()
Restrictions	none

ucosii.queue.os_q_post_en

Identifier	OS_Q_POST_EN
Type	Boolean assignment
Default Value	1
Destination File	system.h
Description	Include code of OSQFlush()
Restrictions	none

ucosii.queue.os_q_post_front_en

Identifier	OS_Q_POST_FRONT_EN
Type	Boolean assignment
Default Value	1
Destination File	system.h
Description	Include code for OSQPostFront()
Restrictions	none

ucosii.queue.os_q_post_opt_en

Identifier	OS_Q_POST_OPT_EN
Type	Boolean assignment
Default Value	1
Destination File	system.h
Description	Include code for OSQPostOpt()
Restrictions	none

ucosii.queue.os_q_query_en

Identifier	OS_Q_QUERY_EN
Type	Boolean assignment
Default Value	1
Destination File	system.h
Description	Include code for OSQQuery()
Restrictions	none

ucosii.queue.os_max_qs

Identifier	OS_MAX_QS
Type	Decimal number
Default Value	20
Destination File	system.h
Description	Maximum number of Queue Control Blocks
Restrictions	none

ucosii.os_mem_en

Identifier	OS_MEM_EN
Type	Boolean assignment
Default Value	1
Destination File	system.h
Description	Enable code for memory management
Restrictions	none

ucosii.memory.os_mem_query_en

Identifier	OS_MEM_QUERY_EN
Type	Boolean assignment
Default Value	1
Destination File	system.h
Description	Include code for OSMemQuery()
Restrictions	none

ucosii.memory.os_mem_name_size

Identifier	OS_MEM_NAME_SIZE
Type	Decimal number
Default Value	32
Destination File	system.h
Description	Size of memory partition name
Restrictions	none

ucosii.memory.os_max_mem_part

Identifier	OS_MAX_MEM_PART
Type	Decimal number
Default Value	60
Destination File	system.h
Description	Maximum number of memory partitions
Restrictions	none

ucosii.os_tmr_en

Identifier	OS_TMR_EN
Type	Boolean assignment
Default Value	0
Destination File	system.h
Description	Enable code for timers
Restrictions	none

ucosii.timer.os_task_tmr_stk_size

Identifier	OS_TASK_TMR_STK_SIZE
Type	Decimal number
Default Value	512
Destination File	system.h
Description	Stack size for timer task
Restrictions	none

ucosii.timer.os_task_tmr_prio

Identifier	OS_TASK_TMR_PRIO
Type	Decimal number
Default Value	2
Destination File	system.h
Description	Priority of timer task (0=highest)
Restrictions	none

ucosii.timer.os_tmr_cfg_max

Identifier	OS_TMR_CFG_MAX
Type	Decimal number
Default Value	16
Destination File	system.h
Description	Maximum number of timers
Restrictions	none

ucosii.timer.os_tmr_cfg_name_size

Identifier	OS_TMR_CFG_NAME_SIZE
Type	Decimal number
Default Value	16
Destination File	system.h
Description	Size of timer name
Restrictions	none

ucosii.timer.os_tmr_cfg_ticks_per_sec

Identifier	OS_TMR_CFG_TICKS_PER_SEC
Type	Decimal number
Default Value	10
Destination File	system.h
Description	Rate at which timer management task runs (Hz)
Restrictions	none

ucosii.timer.os_tmr_cfg_wheel_size

Identifier	OS_TMR_CFG_WHEEL_SIZE
Type	Decimal number
Default Value	2
Destination File	system.h
Description	Size of timer wheel (number of spokes)
Restrictions	none

altera_avalon_uart_driver.enable_small_driver

Identifier	ALTERA_AVALON_UART_SMALL
Type	Boolean definition
Default Value	false
Destination File	public.mk
Description	Small-footprint (polled mode) driver
Restrictions	none

altera_avalon_uart_driver.enable_ioctl

Identifier	ALTERA_AVALON_UART_USE_IOCTL
Type	Boolean definition
Default Value	false
Destination File	public.mk
Description	Enable driver ioctl() support. This feature is not compatible with the small driver; ioctl() support is not compiled if either the UART <code>enable_small_driver</code> or the HAL <code>enable_reduced_device_drivers</code> setting is enabled.
Restrictions	none

altera_avalon_jtag_uart_driver.enable_small_driver

Identifier	ALTERA_AVALON_JTAG_UART_SMALL
Type	Boolean definition
Default Value	false
Destination File	public.mk
Description	Small-footprint (polled mode) driver
Restrictions	none

altera_hostfs.hostfs_name

Identifier	ALTERA_HOSTFS_NAME
Type	Quoted string
Default Value	/mnt/host
Destination File	system.h
Description	Mount point
Restrictions	none

altera_iniche.iniche_default_if

Identifier	INICHE_DEFAULT_IF
Type	Quoted string
Default Value	NOT_USED
Destination File	system.h
Description	Deprecated setting: Default media access control (MAC) interface. This setting was formerly used by Altera networking example applications. It need not be assigned a valid value for the use of networking example designs in Nios II 8.0 or later. If you are using this setting in a project, it is recommended that you remove the dependency. This setting might be removed in a future release.
Restrictions	none

altera_iniche.enable_dhcp_client

Identifier	DHCP_CLIENT
Type	Boolean definition
Default Value	true
Destination File	system.h
Description	Use dynamic host configuration protocol (DHCP) to automatically assign Internet protocol (IP) address
Restrictions	none

altera_iniche.enable_ip_fragments

Identifier	IP_FRAGMENTS
Type	Boolean definition
Default Value	true
Destination File	system.h
Description	Reassemble IP packet fragments
Restrictions	none

altera_iniche.enable_include_tcp

Identifier	INCLUDE_TCP
Type	Boolean definition
Default Value	true
Destination File	system.h
Description	Enable Transmission Control Protocol (TCP)
Restrictions	none

altera_iniche.enable_tcp_zerocopy

Identifier	TCP_ZEROCOPY
Type	Boolean definition
Default Value	false
Destination File	system.h
Description	Use TCP zero-copy
Restrictions	none

altera_iniche.enable_net_stats

Identifier	NET_STATS
Type	Boolean definition
Default Value	false
Destination File	system.h
Description	Enable statistics
Restrictions	none

altera_ro_zipfs.ro_zipfs_name

Identifier	ALTERA_RO_ZIPFS_NAME
Type	Quoted string
Default Value	/mnt/rozipfs
Destination File	system.h
Description	Mount point
Restrictions	none

altera_ro_zipfs.ro_zipfs_offset

Identifier	ALTERA_RO_ZIPFS_OFFSET
Type	Hexadecimal number
Default Value	0x100000
Destination File	system.h
Description	Offset of file system from base of flash
Restrictions	none

altera_ro_zipfs.ro_zipfs_base

Identifier	ALTERA_RO_ZIPFS_BASE
Type	Hexadecimal number
Default Value	0x0
Destination File	system.h
Description	Base address of flash memory device
Restrictions	none

hal.linker.allow_code_at_reset

Identifier	none
Type	Boolean assignment
Default Value	0
Destination File	none
Description	Indicates if initialization code is allowed at the reset address. If true, defines the macro ALT_ALLOW_CODE_AT_RESET in linker.h .
Restrictions	This setting is typically false if an external bootloader (e.g. flash bootloader) is present.

hal.linker.enable_alt_load

Identifier	none
Type	Boolean assignment
Default Value	1
Destination File	none
Description	Enables the alt_load() facility. The alt_load() facility copies sections from the .text memory into RAM. If true, this setting sets up the VMA/LMA (virtual memory address/low memory address) of sections in linker.x to allow them to be loaded into the .text memory.
Restrictions	This setting is typically false if an external bootloader (e.g. flash bootloader) is present.

hal.linker.enable_alt_load_copy_exceptions

Identifier	none
Type	Boolean assignment
Default Value	0
Destination File	none
Description	Causes the alt_load() facility to copy the .exceptions section. If true, this setting defines the macro ALT_LOAD_COPY_EXCEPTIONS in linker.h.
Restrictions	none

hal.linker.enable_alt_load_copy_rodata

Identifier	none
Type	Boolean assignment
Default Value	0
Destination File	none
Description	Causes the alt_load() facility to copy the .rodata section. If true, this setting defines the macro ALT_LOAD_COPY_RODATA in linker.h.
Restrictions	none

hal.linker.enable_alt_load_copy_rwdata

Identifier	none
Type	Boolean assignment
Default Value	0
Destination File	none
Description	Causes the initialization code to copy the .rwdata section. If true, this setting defines the macro ALT_LOAD_COPY_RWDATA in linker.h.
Restrictions	none

hal.linker.enable_exception_stack

Identifier	none
Type	Boolean assignment
Default Value	0
Destination File	none
Description	Enables use of a separate exception stack. If true, defines the macro ALT_EXCEPTION_STACK in <code>linker.h</code> , adds a memory region called <code>exception_stack</code> to <code>linker.x</code> , and provides the symbols <code>__alt_exception_stack_pointer</code> and <code>__alt_exception_stack_limit</code> in <code>linker.x</code> .
Restrictions	The <code>hal.linker.exception_stack_size</code> and <code>hal.linker.exception_stack_memory_region_name</code> settings must also be valid. This setting must be false for MicroC/OS-II BSPs. The exception stack can be used to improve interrupt and other exception performance if an EIC is not implemented.

hal.linker.exception_stack_memory_region_name

Identifier	none
Type	Unquoted string
Default Value	none
Destination File	none
Description	Name of the existing memory region to be divided up to create the <code>exception_stack</code> memory region. The selected region name is adjusted automatically when the BSP is generated to create the <code>exception_stack</code> memory region.
Restrictions	Only used if <code>hal.linker.enable_exception_stack</code> is true.

hal.linker.exception_stack_size

Identifier	none
Type	Decimal number
Default Value	1024
Destination File	none
Description	Size of the exception stack in bytes.
Restrictions	Only used if <code>hal.linker.enable_exception_stack</code> is true.

hal.linker.enable_interrupt_stack

Identifier	none
Type	Boolean assignment
Default Value	0
Destination File	none
Description	Enables use of a separate interrupt stack. If true, defines the macro <code>ALT_INTERRUPT_STACK</code> in <code>linker.h</code> , adds a memory region called <code>interrupt_stack</code> to <code>linker.x</code> , and provides the symbols <code>__alt_interrupt_stack_pointer</code> and <code>__alt_interrupt_stack_limit</code> in <code>linker.x</code> .
Restrictions	The <code>hal.linker.interrupt_stack_size</code> and <code>hal.linker.interrupt_stack_memory_region_name</code> settings must also be valid. This setting must be false for MicroC/OS-II BSPs. Only enable this setting for systems with an EIC. If an EIC is not implemented, use the separate exception stack to improve interrupt and other exception performance.

hal.linker.interrupt_stack_memory_region_name

Identifier	none
Type	Unquoted String
Default Value	none
Destination File	none
Description	Name of the existing memory region that is divided up to create the <code>interrupt_stack</code> memory region. The selected region name is adjusted automatically when the BSP is generated to create the <code>interrupt_stack</code> memory region.
Restrictions	Only used if <code>hal.linker.enable_interrupt_stack</code> is true.

hal.linker.interrupt_stack_size

Identifier	none
Type	Decimal Number
Default Value	1024
Destination File	none
Description	Size of the interrupt stack in bytes.
Restrictions	Only used if <code>hal.linker.enable_interrupt_stack</code> is true.

hal.make.ar

Identifier	AR
Type	Unquoted string
Default Value	nios2-elf-ar
Destination File	BSP makefile
Description	Archiver command. Creates library files.
Restrictions	none

hal.make.ar_post_process

Identifier	AR_POST_PROCESS
Type	Unquoted string
Default Value	none
Destination File	BSP makefile
Description	Command executed after archiver execution.
Restrictions	none

hal.make.ar_pre_process

Identifier	AR_PRE_PROCESS
Type	Unquoted string
Default Value	none
Destination File	BSP makefile
Description	Command executed before archiver execution.
Restrictions	none

hal.make.as

Identifier	AS
Type	Unquoted string
Default Value	nios2-elf-gcc
Destination File	BSP makefile
Description	Assembler command. Note that CC is used for Nios II assembly language source files (.S).
Restrictions	none

hal.make.as_post_process

Identifier	AS_POST_PROCESS
Type	Unquoted string
Default Value	none
Destination File	BSP makefile
Description	Command executed after each assembly file is compiled.
Restrictions	none

hal.make.as_pre_process

Identifier	AS_PRE_PROCESS
Type	Unquoted string
Default Value	none
Destination File	BSP makefile
Description	Command executed before each assembly file is compiled.
Restrictions	none

hal.make.bsp_arflags

Identifier	BSP_ARFLAGS
Type	Unquoted string
Default Value	-src
Destination File	BSP makefile
Description	Custom flags only passed to the archiver. This content of this variable is directly passed to the archiver rather than the more standard ARFLAGS. The reason for this is that GNU Make assumes some default content in ARFLAGS. This setting defines the value of BSP_ARFLAGS in Makefile.
Restrictions	none

hal.make.bsp_asflags

Identifier	BSP_ASFLAGS
Type	Unquoted string
Default Value	-Wa,-gdwarf2
Destination File	BSP makefile
Description	Custom flags only passed to the assembler. This setting defines the value of BSP_ASFLAGS in Makefile.
Restrictions	none

hal.make.bsp_cflags_debug

Identifier	BSP_CFLAGS_DEBUG
Type	Unquoted string
Default Value	-g
Destination File	BSP makefile
Description	C/C++ compiler debug level. <code>-g</code> provides the default set of debug symbols typically required to debug a typical application. Omitting <code>-g</code> removes debug symbols from the ELF. This setting defines the value of BSP_CFLAGS_DEBUG in Makefile.
Restrictions	none

hal.make.bsp_cflags_defined_symbols

Identifier	BSP_CFLAGS_DEFINED_SYMBOLS
Type	Unquoted string
Default Value	none
Destination File	BSP makefile
Description	Preprocessor macros to define. A macro definition in this setting has the same effect as a <code>#define</code> in source code. Adding <code>-DALT_DEBUG</code> to this setting has the same effect as <code>#define ALT_DEBUG</code> in a source file. Adding <code>-DFOO=1</code> to this setting is equivalent to the macro <code>#define FOO 1</code> in a source file. Macros defined with this setting are applied to all <code>.S</code> , C source (<code>.c</code>), and C++ files in the BSP. This setting defines the value of <code>BSP_CFLAGS_DEFINED_SYMBOLS</code> in the BSP makefile.
Restrictions	none

hal.make.bsp_cflags_optimization

Identifier	BSP_CFLAGS_OPTIMIZATION
Type	Unquoted string
Default Value	-O0
Destination File	BSP makefile
Description	C/C++ compiler optimization level. <code>-O0</code> = no optimization, <code>-O2</code> = normal optimization, etc. <code>-O0</code> is recommended for code that you want to debug since compiler optimization can remove variables and produce nonsequential execution of code while debugging. This setting defines the value of <code>BSP_CFLAGS_OPTIMIZATION</code> in Makefile.
Restrictions	none

hal.make.bsp_cflags_undefined_symbols

Identifier	BSP_CFLAGS_UNDEFINED_SYMBOLS
Type	Unquoted string
Default Value	none
Destination File	BSP makefile
Description	Preprocessor macros to undefine. Undefined macros are similar to defined macros, but replicate the <code>#undef</code> directive in source code. To undefine the macro <code>FOO</code> use the syntax <code>-u FOO</code> in this setting. This is equivalent to <code>#undef FOO</code> in a source file. Note: the syntax differs from macro definition (there is a space, i.e. <code>-u FOO</code> versus <code>-DFOO</code>). Macros defined with this setting are applied to all <code>.S</code> , <code>.c</code> , and C++ files in the BSP. This setting defines the value of <code>BSP_CFLAGS_UNDEFINED_SYMBOLS</code> in the BSP Makefile.
Restrictions	none

hal.make.bsp_cflags_user_flags

Identifier	BSP_CFLAGS_USER_FLAGS
Type	Unquoted string
Default Value	none
Destination File	BSP makefile
Description	Custom flags passed to the compiler when compiling C, C++, and <code>.S</code> files. This setting defines the value of <code>BSP_CFLAGS_USER_FLAGS</code> in Makefile.
Restrictions	none

hal.make.bsp_cflags_warnings

Identifier	BSP_CFLAGS_WARNINGS
Type	Unquoted string
Default Value	-Wall
Destination File	BSP makefile
Description	C/C++ compiler warning level. <code>-Wall</code> is commonly used. This setting defines the value of <code>BSP_CFLAGS_WARNINGS</code> in Makefile.
Restrictions	none

hal.make.bsp_cxx_flags

Identifier	BSP_CXXFLAGS
Type	Unquoted string
Default Value	none
Destination File	BSP makefile
Description	Custom flags only passed to the C++ compiler. This setting defines the value of <code>BSP_CXXFLAGS</code> in Makefile.
Restrictions	none

hal.make.bsp_inc_dirs

Identifier	BSP_INC_DIRS
Type	Unquoted string
Default Value	none
Destination File	BSP makefile
Description	Space separated list of extra include directories to scan for header files. Directories are relative to the top-level BSP directory. The <code>-I</code> prefix is added by the makefile, therefore you must not include it in the setting value. This setting defines the value of <code>BSP_INC_DIRS</code> in the makefile.
Restrictions	none

hal.make.build_post_process

Identifier	BUILD_POST_PROCESS
Type	Unquoted string
Default Value	none
Destination File	BSP makefile
Description	Command executed after BSP built.
Restrictions	none

hal.make.build_pre_process

Identifier	BUILD_PRE_PROCESS
Type	Unquoted string
Default Value	none
Destination File	BSP makefile
Description	Command executed before BSP built.
Restrictions	none

hal.make.cc

Identifier	CC
Type	Unquoted string
Default Value	nios2-elf-gcc -xc
Destination File	BSP makefile
Description	C compiler command
Restrictions	none

hal.make.cc_post_process

Identifier	CC_POST_PROCESS
Type	Unquoted string
Default Value	none
Destination File	BSP makefile
Description	Command executed after each .c or .S file is compiled.
Restrictions	none

hal.make.cc_pre_process

Identifier	CC_PRE_PROCESS
Type	Unquoted string
Default Value	none
Destination File	BSP makefile
Description	Command executed before each .c or .S file is compiled.
Restrictions	none

hal.make.cxx

Identifier	CXX
Type	Unquoted string
Default Value	nios2-elf-gcc -xc++
Destination File	BSP makefile
Description	C++ compiler command
Restrictions	none

hal.make.cxx_post_process

Identifier	CXX_POST_PROCESS
Type	Unquoted string
Default Value	none
Destination File	BSP makefile
Description	Command executed before each C++ file is compiled.
Restrictions	none

hal.make.cxx_pre_process

Identifier	CXX_PRE_PROCESS
Type	Unquoted string
Default Value	none
Destination File	BSP makefile
Description	Command executed before each C++ file is compiled.
Restrictions	none

hal.make.ignore_system_derived.big_endian

Identifier	none
Type	Boolean assignment
Default Value	0
Destination File	public.mk
Description	Enable BSP generation to query if SOPC system is big endian. If true ignores export of 'ALT_CFLAGS += -EB' to public.mk if big endian system. If true ignores export of 'ALT_CFLAGS += -EL' if little endian system.
Restrictions	none

hal.make.ignore_system_derived.fpu_present

Identifier	none
Type	Boolean assignment
Default Value	0
Destination File	public.mk
Description	Enable BSP generation to query if SOPC system has FPU present. If true ignores export of 'ALT_CFLAGS += -mhard-float' to public.mk if FPU is found in the system. If true ignores export of 'ALT_CFLAGS += -mhard-soft' if FPU is not found in the system.
Restrictions	none

hal.make.ignore_system_derived.hardware_divide_present

Identifier	none
Type	Boolean assignment
Default Value	0
Destination File	public.mk
Description	Enable BSP generation to query if SOPC system has hardware divide present. If true ignores export of 'ALT_CFLAGS += -mno-hw-div' to public.mk if no division is found in system. If true ignores export of 'ALT_CFLAGS += -mhw-div' if division is found in the system.
Restrictions	none

hal.make.ignore_system_derived.hardware_fp_cust_inst_divider_present

Identifier	none
Type	Boolean assignment
Default Value	0
Destination File	public.mk
Description	Enable BSP generation to query if SOPC system floating point custom instruction with a divider is present. If true ignores export of 'ALT_CFLAGS += -mcustom-fpu-cfg=60-2' and 'ALT_LDFLAGS += -mcustom-fpu-cfg=60-2' to public.mk if the custom instruction is found in the system.
Restrictions	none

hal.make.ignore_system_derived.hardware_fp_cust_inst_no_divider_present

Identifier	none
Type	Boolean assignment
Default Value	0
Destination File	public.mk
Description	Enable BSP generation to query if SOPC system floating point custom instruction without a divider is present. If true ignores export of 'ALT_CFLAGS += -mcustom-fpu-cfg=60-1' and 'ALT_LDFLAGS += -mcustom-fpu-cfg=60-1' to public.mk if the custom instruction is found in the system.
Restrictions	none

hal.make.ignore_system_derived.sopc_simulation_enabled

Identifier	none
Type	Boolean assignment
Default Value	0
Destination File	public.mk
Description	Enable BSP generation to query if SOPC system has simulation enabled. If true ignores export of 'ELF_PATCH_FLAG += --simulation_enabled' to public.mk.
Restrictions	none

hal.make.ignore_system_derived.debug_core_present

Identifier	none
Type	Boolean assignment
Default Value	0
Destination File	public.mk
Description	Enable BSP generation to query if SOPC system has a debug core present. If true ignores export of 'CPU_HAS_DEBUG_CORE = 1' to public.mk if a debug core is found in the system. If true ignores export of 'CPU_HAS_DEBUG_CORE = 0' if no debug core is found in the system.
Restrictions	none

hal.make.ignore_system_derived.hardware_multiplier_present

Identifier	none
Type	Boolean assignment
Default Value	0
Destination File	public.mk
Description	Enable BSP generation to query if SOPC system has multiplier present. If true ignores export of 'ALT_CFLAGS += -mno-hw-mul' to public.mk if no multiplier is found in the system. If true ignores export of 'ALT_CFLAGS += -mhw-mul' if multiplier is found in the system.
Restrictions	none

hal.make.ignore_system_derived.hardware_mux_present

Identifier	none
Type	Boolean assignment
Default Value	0
Destination File	public.mk
Description	Enable BSP generation to query if SOPC system has hardware mux present. If true ignores export of 'ALT_CFLAGS += -mno-hw-mulx' to public.mk if no mux is found in the system. If true ignores export of 'ALT_CFLAGS += -mhw-mulx' if mux is found in the system.
Restrictions	none

hal.make.ignore_system_derived.sopc_system_base_address

Identifier	none
Type	Boolean assignment
Default Value	0
Destination File	public.mk
Description	Enable BSP generation to query SOPC system for system ID base address. If true ignores export of 'SOPC_SYSID_FLAG += --sidp=<address>' and 'ELF_PATCH_FLAG += --sidp=<address>' to public.mk.
Restrictions	none

hal.make.ignore_system_derived.sopc_system_id

Identifier	none
Type	Boolean assignment
Default Value	0
Destination File	public.mk
Description	Enable BSP generation to query SOPC system for system ID. If true ignores export of 'SOPC_SYSID_FLAG += --id=<sysid>' and 'ELF_PATCH_FLAG += --id=<sysid>' to public.mk.
Restrictions	none

hal.make.ignore_system_derived.sopc_system_timestamp

Identifier	none
Type	Boolean assignment
Default Value	0
Destination File	public.mk
Description	Enable BSP generation to query SOPC system for system timestamp. If true ignores export of 'SOPC_SYSID_FLAG += --timestamp=<timestamp>' and 'ELF_PATCH_FLAG += --timestamp=<timestamp>' to public.mk.
Restrictions	none

hal.make.rm

Identifier	RM
Type	Unquoted string
Default Value	rm -f
Destination File	BSP makefile
Description	Command used to remove files when building the <code>clean</code> target.
Restrictions	none

hal.custom_newlib_flags

Identifier	CUSTOM_NEWLIB_FLAGS
Type	Unquoted string
Default Value	none
Destination File	public.mk
Description	Build a custom version of newlib with the specified space-separated compiler flags.
Restrictions	The custom newlib build is placed in the <i><bsp root>/newlib</i> directory, and is used only for applications that utilize this BSP.

hal.enable_c_plus_plus

Identifier	ALT_NO_C_PLUS_PLUS
Type	Boolean assignment
Default Value	1
Destination File	public.mk
Description	Enable support for a subset of the C++ language. This option increases code footprint by adding support for C++ constructors. Certain features, such as multiple inheritance and exceptions are not supported. If false, adds <code>-DALT_NO_C_PLUS_PLUS</code> to <code>ALT_CPPFLAGS</code> in public.mk , and reduces code footprint.
Restrictions	none

hal.enable_clean_exit

Identifier	ALT_NO_CLEAN_EXIT
Type	Boolean assignment
Default Value	1
Destination File	public.mk
Description	When your application exits, close file descriptors, call C++ destructors, etc. Code footprint can be reduced by disabling clean exit. If disabled, adds <code>-DALT_NO_CLEAN_EXIT</code> to <code>ALT_CPPFLAGS</code> and <code>-Wl,--defsym,exit=_exit</code> to <code>ALT_LDFLAGS</code> in public.mk .
Restrictions	none

hal.enable_exit

Identifier	ALT_NO_EXIT
Type	Boolean assignment
Default Value	1
Destination File	public.mk
Description	Add <code>exit()</code> support. This option increases code footprint if your <code>main()</code> routine returns or calls <code>exit()</code> . If false, adds <code>-DALT_NO_EXIT</code> to <code>ALT_CPPFLAGS</code> in public.mk , and reduces footprint.
Restrictions	none

hal.enable_gprof

Identifier	ALT_PROVIDE_GMON
Type	Boolean assignment
Default Value	0
Destination File	public.mk
Description	Causes code to be compiled with gprof profiling enabled and the application ELF to be linked with the GPROF library. If true, adds <code>-DALT_PROVIDE_GMON</code> to <code>ALT_CPPFLAGS</code> and <code>-pg</code> to <code>ALT_CFLAGS</code> in public.mk .
Restrictions	none

hal.enable_lightweight_device_driver_api

Identifier	ALT_USE_DIRECT_DRIVERS
Type	Boolean assignment
Default Value	0
Destination File	public.mk
Description	Enables lightweight device driver API. This reduces code and data footprint by removing the HAL layer that maps device names (e.g. <code>/dev/uart0</code>) to file descriptors. Instead, driver routines are called directly. The <code>open()</code> , <code>close()</code> , and <code>lseek()</code> routines always fail if called. The <code>read()</code> , <code>write()</code> , <code>fstat()</code> , <code>ioctl()</code> , and <code>isatty()</code> routines only work for the stdio devices. If true, adds <code>-DALT_USE_DIRECT_DRIVERS</code> to <code>ALT_CPPFLAGS</code> in public.mk .
Restrictions	The Altera Host and read-only ZIP file systems cannot be used if <code>hal.enable_lightweight_device_driver_api</code> is true.

hal.enable_mul_div_emulation

Identifier	ALT_NO_INSTRUCTION_EMULATION
Type	Boolean assignment
Default Value	0
Destination File	public.mk
Description	Adds code to emulate multiply and divide instructions in case they are executed but are not present in the processor. Normally this is not required because the compiler does not use multiply and divide instructions that are not present in the processor. If false, adds <code>-DALT_NO_INSTRUCTION_EMULATION</code> to <code>ALT_CPPFLAGS</code> in public.mk .
Restrictions	none

hal.enable_reduced_device_drivers

Identifier	ALT_USE_SMALL_DRIVERS
Type	Boolean assignment
Default Value	0
Destination File	public.mk
Description	Certain drivers are compiled with reduced functionality to reduce code footprint. Not all drivers observe this setting. The <code>altera_avalon_uart</code> and <code>altera_avalon_jtag_uart</code> drivers switch from interrupt-driven to polled operation. CAUTION: Several device drivers are disabled entirely. These include the <code>altera_avalon_cfi_flash</code> , <code>altera_avalon_epcs_flash_controller</code> , and <code>altera_avalon_lcd_16207</code> drivers. This can result in certain API (HAL flash access routines) to fail. You can define a symbol provided by each driver to prevent it from being removed. If true, adds <code>-DALT_USE_SMALL_DRIVERS</code> to <code>ALT_CPPFLAGS</code> in public.mk .
Restrictions	none

hal.enable_runtime_stack_checking

Identifier	ALT_STACK_CHECK
Type	Boolean assignment
Default Value	0
Destination File	public.mk
Description	Turns on HAL runtime stack checking feature. Enabling this setting causes additional code to be placed into each subroutine call to generate an exception if a stack collision occurs with the heap or statically allocated data. If true, adds <code>-DALY_STACK_CHECK</code> and <code>-mstack-check</code> to <code>ALT_CPPFLAGS</code> in public.mk .
Restrictions	none

hal.enable_sim_optimize

Identifier	ALT_SIM_OPTIMIZE
Type	Boolean assignment
Default Value	0
Destination File	public.mk
Description	The BSP is compiled with optimizations to speedup HDL simulation such as initializing the cache, clearing the <code>.bss</code> section, and skipping long delay loops. If true, adds <code>-DALY_SIM_OPTIMIZE</code> to <code>ALT_CPPFLAGS</code> in public.mk .
Restrictions	When this setting is true, the BSP cannot run on hardware.

hal.enable_small_c_library

Identifier	none
Type	Boolean assignment
Default Value	0
Destination File	public.mk
Description	Causes the small newlib (C library) to be used. This reduces code and data footprint at the expense of reduced functionality. Several newlib features are removed such as floating-point support in <code>printf()</code> , <code>stdin</code> input routines, and buffered I/O. The small C library is not compatible with Micrium MicroC/OS-II. If true, adds <code>-msmallc</code> to <code>ALT_LDFLAGS</code> and adds <code>-DSMALL_C_LIB</code> to <code>ALT_CPPFLAGS</code> in public.mk .
Restrictions	none

hal.enable_sopc_sysid_check

Identifier	none
Type	Boolean assignment
Default Value	1
Destination File	public.mk
Description	Enable SOPC Builder System ID. If a System ID SOPC Builder component is connected to the processor associated with this BSP, it is enabled in the creation of command-line arguments to download an ELF to the target. Otherwise, system ID and timestamp values are left out of public.mk for the application makefile <code>download-elf</code> target definition. With the system ID check disabled, the Nios II EDS tools do not automatically ensure that the application <code>.elf</code> file (and BSP it is linked against) corresponds to the hardware design on the target. If false, adds <code>--accept-bad-sysid</code> to <code>SOPC_SYSID_FLAG</code> in public.mk .
Restrictions	none

hal.log_port

Identifier	LOG_PORT
Type	Unquoted string
Default Value	none
Destination File	system.h
Description	Slave descriptor of debug logging character-mode device. If defined, it enables extra debug messages in the HAL source. This setting is used by the ALT_LOG_PORT family of defines in system.h .

hal.log_flags

Identifier	ALT_LOG_FLAGS
Type	Decimal Number
Default Value	0
Destination File	public.mk
Description	The value is assigned to ALT_LOG_FLAGS in the generated public.mk . Refer to hal.log_port for further details. The valid range of this setting is 1 through 4.

hal.stderr

Identifier	STDERR
Type	Unquoted string
Default Value	none
Destination File	public.mk
Description	Slave descriptor of STDERR character-mode device. This setting is used by the ALT_STDERR family of defines in system.h .

hal.stdin

Identifier	STDIN
Type	Unquoted string
Default Value	none
Destination File	system.h
Description	Slave descriptor of STDIN character-mode device. This setting is used by the ALT_STDIN family of defines in system.h .

hal.stdout

Identifier	STDOUT
Type	Unquoted string
Default Value	none
Destination File	system.h
Description	Slave descriptor of STDOUT character-mode device. This setting is used by the ALT_STDOUT family of defines in system.h .

Application and User Library Makefile Variables

The Nios II SBT constructs application and makefile libraries for you, inserting makefile variables appropriate to your project configuration. You can control project build characteristics by manipulating makefile variables at the time of project generation. You control a variable with the `--set` command line option, as in the following example:

```
nios2-bsp hal my_bsp --set APP_CFLAGS_WARNINGS "-Wall" ←
```

The following utilities and scripts support modifying makefile variables with the `--set` option:

- **nios2-app-generate-makefile**
- **nios2-lib-generate-makefile**
- **nios2-app-update-makefile**
- **nios2-lib-update-makefile**
- **nios2-bsp**

Application Makefile Variables

You can modify the following application makefile variables on the command line:

- **CREATE_OBJDUMP**—Assign 1 to this variable to enable creation of an object dump file (**.objdump**) after linking the application. The **nios2-elf-objdump** utility is called to create this file. An object dump contains information about all object files linked into the **.elf** file. It provides a complete view of all code linked into your application. An object dump contains a disassembly view showing each instruction and its address.
- **OBJDUMP_INCLUDE_SOURCE**—Assign 1 to this variable to include source code inline with disassembled instructions in the object dump. When enabled, this includes the **--source** switch when calling the object dump executable. This is useful for debugging and examination of how the preprocessor and compiler generate instructions from higher level source code (such as C) or from macros.
- **OBJDUMP_FULL_CONTENTS**—Assign 1 to this variable to include a raw display of the contents of the **.text** linker section. When enabled, this variable includes the **--full-contents** switch when calling the object dump executable.
- **CREATE_ELF_DERIVED_FILES**—Setting this variable to 1 creates the HDL simulation and onchip memory initialization files when you invoke the makefile with the **all** target. When this variable is 0 (the default), these files are only created when you make the **mem-init-install** target.




Creating the HDL simulation and onchip memory initialization files increases project build time.

- **CREATE_LINKER_MAP**—Assign 1 to this variable to enable creation of a link map file (**.map**) after linking the application. A link map file provides information including which object files are included in the executable, the path to each object file, where objects and symbols are located in memory, and how the common symbols are allocated.
- **APP_CFLAGS_DEFINED_SYMBOLS**—This variable allows you to define macros using the **-D** argument, for example **-D <macro name>**. The contents of this variable are passed to the compiler and linker without modification.
- **APP_CFLAGS_UNDEFINED_SYMBOLS**—This variable allows you to remove macro definitions using the **-U** argument, for example **-U <macro name>**. The contents of this variable are passed to the compiler and linker without modification.
- **APP_CFLAGS_OPTIMIZATION**—The C/C++ compiler optimization level. For example, **-O0** provides no optimization and **-O2** provides standard optimization. **-O0** is recommended for debugging code, because compiler optimization can remove variables and produce non-sequential execution of code while debugging.
- **APP_CFLAGS_DEBUG_LEVEL**—The C/C++ compiler debug level. **-g** provides the default set of debug symbols typically required to debug an application. Omitting **-g** omits debug symbols from the **.elf**.
- **APP_CFLAGS_WARNINGS**—The C/C++ compiler warning level. **-Wall** is commonly used, enabling all warning messages.
- **APP_CFLAGS_USER_FLAGS**


- **APP_INCLUDE_DIRS**—Use this variable to specify paths for the preprocessor to search. These paths commonly contain C header files (**.h**) that application code requires. Each path name is formatted and passed to the preprocessor with the **-I** option.

You can add multiple directories by enclosing them in double quotes, for example `--set APP_INCLUDE_DIRS "../my_includes ../../other_includes"`.

- **APP_LIBRARY_DIRS**—Use this variable to specify paths for additional libraries that your application links with.


 When you specify a user library path with **APP_LIBRARY_DIRS**, you also need to specify the user library names with the **APP_LIBRARY_NAMES** variable.

APP_LIBRARY_DIRS specifies only the directory where the user library file(s) are located, not the library archive file (**.a**) name.


 Do not use this variable to specify the path to a BSP or user library created with the SBT. The paths to these libraries are specified in **public.mk** files included in the application makefile.

You can add multiple directories by enclosing them in double quotes, for example `--set APP_LIBRARY_DIRS "../my_libs ../../other_libs"`.

- **APP_LIBRARY_NAMES**—Use this variable to specify the names of additional libraries that your application must link with. Library files are **.a** files.

 You do not specify the full name of the **.a** file. Instead, you specify the user library name **<name>**, and the SBT constructs the filename **lib<name>.a**. For example, if you add the string "math" to **APP_LIBRARY_NAMES**, the SBT assumes that your library file is named **libmath.a**.

Each specified user library name is passed to the linker with the **-l** option. The paths to locate these libraries must be specified in the **APP_LIBRARY_DIRS** variable.

 You cannot use this variable to specify a BSP or user library created with the SBT. The paths to these libraries are specified in **public.mk** file included in the application makefile.

- **BUILD_PRE_PROCESS**—This variable allows you to specify a command to be executed prior to building the application, for example, `cp *.elf ../lastbuild`.
- **BUILD_POST_PROCESS**—This variable allows you to specify a command to be executed after building the application, for example, `cp *.elf //production/test/nios2executables`.

User Library Makefile Variables

You can modify the following user library makefile variables on the command line:

- `LIB_CFLAGS_DEFINED_SYMBOLS`—This variable allows you to define macros using the `-D` argument, for example `-D <macro name>`. The contents of this variable are passed to the compiler and linker without modification.
- `LIB_CFLAGS_UNDEFINED_SYMBOLS`—This variable allows you to remove macro definitions using the `-U` argument, for example `-U <macro name>`. The contents of this variable are passed to the compiler and linker without modification.
- `LIB_CFLAGS_OPTIMIZATION`—The C/C++ compiler optimization level. For example, `-O0` provides no optimization and `-O2` provides standard optimization. `-O0` is recommended for debugging code, because compiler optimization can remove variables and produce non-sequential execution of code while debugging.
- `LIB_CFLAGS_DEBUG_LEVEL`—The C/C++ compiler debug level. `-g` provides the default set of debug symbols typically required to debug an application. Omitting `-g` omits debug symbols from the `.elf`.
- `LIB_CFLAGS_WARNINGS`—The C/C++ compiler warning level. `-Wall` is commonly used, enabling all warning messages.
- `LIB_CFLAGS_USER_FLAGS`—
- `LIB_INCLUDE_DIRS`—You can add multiple directories by enclosing them in double quotes, for example `--set LIB_INCLUDE_DIRS
"../my_includes ../../other_includes"`

Standard Build Flag Variables

The SBT creates makefiles supporting the following standard makefile command-line variables:

- `CFLAGS`
- `CPPFLAGS`
- `ASFLAGS`
- `CXXFLAGS`

You can define flags in these variables on the makefile command line, or in a script that invokes the makefile. The makefile passes these flags on to the corresponding GCC tool.

Tcl Commands

Tcl commands are a crucial component of the Nios II SBT. Tcl commands allow you to exercise detailed control over BSP generation, as well as to define drivers and software packages. This section describes the Tcl commands, the environments in which they run, and how the commands work together.

Tcl Command Environments

The Nios II SBT supports Tcl commands in the following environments:


- **BSP setting specification**—In this environment, you manipulate BSP settings to control the static characteristics of the BSP. BSP setting commands are executed before the BSP is generated.
- **BSP generation callbacks**—In this environment, you exercise further control over BSP details, managing settings that interact with one another and with the hardware design. BSP callbacks run at BSP generation time.
- **Device driver and software package definition**—In this environment, you bundle source files into a custom driver or package. This process prepares the driver or package so that a BSP developer can include it in a BSP using the SBT.

The following sections describe each Tcl environment in detail, listing the available commands.

Tcl Commands for BSP Settings

[“Settings” on page 15-27](#) describes settings that are available in a Nios II project. This section describes the tools that you use to specify and manipulate these settings.

You manipulate project settings with BSP Tcl commands. The commands in this section are used with the utilities **nios2-bsp-create-settings**, **nios2-bsp-update-settings**, and **nios2-bsp-query-settings**. You can call the Tcl commands directly on a utility command line using the `--cmd` option, or you can put them in a Tcl script, specified with the `--script` option. For details about how to call Tcl commands from utilities, refer to [“Nios II Software Build Tools Utilities” on page 15-1](#).

-  For more information about creating Tcl scripts, refer to [“Tcl Scripts for BSP Settings”](#) in the *Nios II Software Build Tools* chapter of the *Nios II Software Developer’s Handbook*. This chapter includes a discussion of the default Tcl script, which provides excellent usage examples of many of the Tcl commands described in this section.

The following commands are available to manipulate BSP settings:

- [“add_memory_device” on page 15-74](#)
- [“add_memory_region” on page 15-75](#)
- [“add_section_mapping” on page 15-75](#)
- [“add_systemh_line” on page 15-76](#)
- [“are_same_resource” on page 15-76](#)
- [“delete_memory_region” on page 15-77](#)
- [“delete_section_mapping” on page 15-77](#)
- [“disable_sw_package” on page 15-77](#)
- [“enable_sw_package” on page 15-78](#)
- [“get_addr_span” on page 15-78](#)
- [“get_assignment” on page 15-78](#)

- “get_available_drivers” on page 15-79
- “get_available_sw_packages” on page 15-79
- “get_base_addr” on page 15-80
- “get_break_offset” on page 15-80
- “get_break_slave_desc” on page 15-80
- “get_cpu_name” on page 15-81
- “get_current_memory_regions” on page 15-81
- “get_current_section_mappings” on page 15-81
- “get_default_memory_regions” on page 15-82
- “get_driver” on page 15-82
- “get_enabled_sw_packages” on page 15-83
- “get_exception_offset” on page 15-83
- “get_exception_slave_desc” on page 15-83
- “get_fast_tlb_miss_exception_offset” on page 15-84
- “get_fast_tlb_miss_exception_slave_desc” on page 15-84
- “get_interrupt_controller_id” on page 15-85
- “get_irq_interrupt_controller_id” on page 15-85
- “get_irq_number” on page 15-85
- “get_memory_region” on page 15-85
- “get_module_class_name” on page 15-86
- “get_module_name” on page 15-86
- “get_reset_offset” on page 15-86
- “get_reset_slave_desc” on page 15-87
- “get_section_mapping” on page 15-87
- “get_setting” on page 15-88
- “get_setting_desc” on page 15-88
- “get_slave_descs” on page 15-88
- “is_char_device” on page 15-89
- “is_connected_interrupt_controller_device” on page 15-89
- “is_connected_to_data_master” on page 15-89
- “is_connected_to_instruction_master” on page 15-90
- “is_ethernet_mac_device” on page 15-90
- “is_flash” on page 15-90
- “is_memory_device” on page 15-90
- “is_non_volatile_storage” on page 15-91

- “is_timer_device” on page 15-91
- “log_debug” on page 15-91
- “log_default” on page 15-92
- “log_error” on page 15-92
- “log_verbose” on page 15-92
- “set_driver” on page 15-93
- “set_ignore_file” on page 15-93
- “set_setting” on page 15-94
- “update_memory_region” on page 15-94
- “update_section_mapping” on page 15-95
- “add_default_memory_regions” on page 15-95
- “create_bsp” on page 15-95
- “generate_bsp” on page 15-96
- “get_available_bsp_type_versions” on page 15-96
- “get_available_bsp_types” on page 15-96
- “get_available_cpu_architectures” on page 15-96
- “get_available_cpu_names” on page 15-97
- “get_available_software” on page 15-97
- “get_available_software_setting_properties” on page 15-97
- “get_available_software_settings” on page 15-98
- “get_bsp_version” on page 15-98
- “get_cpu_architecture” on page 15-98
- “get_sopcinfo_file” on page 15-98
- “get_supported_bsp_types” on page 15-99
- “is_bsp_hal_extension” on page 15-99
- “is_bsp_lwhal_extension” on page 15-99
- “open_bsp” on page 15-99
- “save_bsp” on page 15-99
- “set_bsp_version” on page 15-100
- “set_logging_mode” on page 15-100

add_memory_device

Usage

`add_memory_device <device name> <base address> `

Options

- *<device name>*: String with the name of the memory device.
- *<base address>*: The base address of the memory device. Hexadecimal or decimal string.
- **: The size (span) of the memory device. Hexadecimal or decimal string.

Description

This command is provided to define a user-defined external memory device, outside the SOPC Builder system. Such a device would typically be mapped through a bridge component. This command adds an external memory device to the BSP's memory map, allowing the BSP to define memory regions and section mappings for the memory as if it were part of the system. The external memory device parameters are stored in the BSP settings file.

add_memory_region

Usage

```
add_memory_region <name> <slave_desc> <offset> <span>
```

Options

- *<name>*: String with the name of the memory region to create.
- *<slave_desc>*: String with the slave descriptor of the memory device for this region.
- *<offset>*: String with the byte offset of the memory region from the memory device base address.
- **: String with the span of the memory region in bytes.

Description

Creates a new memory region for the linker script. This memory region must not overlap with any other memory region and must be within the memory range of the associated slave descriptor. The offset and span are decimal numbers unless prefixed with 0x.

Example

```
add_memory_region onchip_ram0 onchip_ram0 0 0x100000
```

add_section_mapping

Usage

```
add_section_mapping <section_name> <memory_region_name>
```

Options

- *<section_name>*: String with the name of the linker section.
- *<memory_region_name>*: String with the name of the memory region to map.

Description

Maps the specified linker section to the specified linker memory region. If the section does not already exist, `add_section_mapping` creates it. If it already exists, `add_section_mapping` overrides the existing mapping with the new one. The linker creates the section mappings in the order in which they appear in the linker script.

Example

```
add_section_mapping .text onchip_ram0
```

add_systemh_line

Usage

```
add_systemh_line <sw> <name> <value>
```

- `<sw>`: The software (OS) that the **system.h** text is associated with
- `<name>`: Name of macro to write into **system.h** (left-hand side of `#define`)
- `<value>`: Name of value to assign to macro in **system.h** (right-hand side of `#define`)

Description

Adds a line of text to the **system.h** file. The `<sw>` argument is the name of the software type (typically an operating system name) that the **system.h** text applies to. In the context of an operating system Tcl script, the name in the `create_os <name>` command must be used. The text is a name-value pair that creates a macro (`#define` statement) in the **system.h** file.



This command can only be used by Tcl scripts that are registered to run at BSP generation time by an operating system.

Example

```
add_systemh_line UCOSII OS_TICKS_PER_SEC 100
```

are_same_resource

Usage

```
are_same_resource <slave_desc1> <slave_desc2>
```

Options

- `<slave_desc1>`: String with the first slave descriptor to compare.
- `<slave_desc2>`: String with the second slave descriptor to compare.

Description

Returns a boolean value that indicates whether the two slave descriptors are connected to the same resource. To connect to the same resource, the two slave descriptors must be associated with the same module. The module specifies whether two slaves access the same resource or different resources within that module. For example, a dual-port memory has two slaves that access the same resource (the memory). However, you could create a module that has two slaves that access two different resources such as a memory and a control port.

delete_memory_region**Usage**

```
delete_memory_region <region_name>
```

Options

- *<region_name>*: String with the name of the memory region to delete.

Description

Deletes the specified memory region. The region must exist to avoid an error condition.

delete_section_mapping**Usage**

```
delete_section_mapping <section_name>
```

Options

- *<section_name>*: String with the name of the section.

Description

Deletes the specified section mapping.

Example

```
delete_section_mapping .text
```

disable_sw_package**Usage**

```
disable_sw_package <software_package_name>
```

Options

- *<software_package_name>*: String with the name of the software package.

Description

Disables the specified software package. Settings belonging to the package are no longer available in the BSP, and associated source files are not included in the BSP makefile. It is an error to disable a software package that is not enabled.

enable_sw_package

Usage

```
enable_sw_package <software_package_name>
```

Options

- *<software_package_name>*: String with the name of the software package, with the version number optionally appended with a ':'.

Description

Enables a software package. Adds its associated source files and settings to the BSP. Specify the desired version in the form *<software_package_name>:<version>*. If you do not specify the version, `enable_sw_package` selects the latest available version.

Examples

- Example 1:

```
enable_sw_package altera_hostfs:7.2
```

- Example 2:

```
enable_sw_package my_sw_package
```

get_addr_span

Usage

```
get_addr_span <slave_desc>
```

Options

- *<slave_desc>*: String with the slave descriptor to query.

Description

Returns the address span (length in bytes) of the slave descriptor as an integer decimal number.

Example

```
puts [get_addr_span onchip_ram_64_kbytes]
```

Returns:

```
65536
```

get_assignment

Usage

```
get_assignment <module_name> <assignment_name>
```

Options

- *<module_name>*: Module instance name to query for assignment
- *<assignment_name>*: Module instance assignment name to query for

Description

Returns the name of the value of the assignment for a specified module instance name.

Example

```
puts [get_assignment "cpu0" "embeddedsw.configuration.breakSlave"]
```

Returns:

```
memory_0.s0
```

get_available_drivers

Usage

```
get_available_drivers <module_name>
```

Options

- *<module_name>*: String with the name of the module to query.

Description

Returns a list of available device driver names that are compatible with the specified module instance. The list is empty if there are no drivers available for the specified slave descriptor. The format of each entry in the list is the driver name followed by a colon and the version number (if provided).

Example

```
puts [get_available_drivers jtag_uart]
```

Returns:

```
altera_avalon_jtag_uart_driver:7.2 altera_avalon_jtag_uart_driver:6.1
```

get_available_sw_packages

Usage

```
get_available_sw_packages
```

Options

None

Description

Returns a list of software package names that are available for the current BSP. The format of each entry in the list is a string containing the package name followed by a colon and the version number (if provided).

Example

```
puts [get_available_sw_packages]
```

Returns:

```
altera_hostfs:7.2 altera_ro_zipfs:7.2
```

get_base_addr

Usage

```
get_base_addr <slave_desc>
```

Options

- *<slave_desc>*: String with the slave descriptor to query.

Description

Returns the base byte address of the slave descriptor as an integer decimal number.

Example

```
puts [get_base_addr jtag_uart]
```

Returns:

```
67616
```

get_break_offset

Usage

```
get_break_offset
```

Options

None

Description

Returns the byte offset of the processor break address.

Example

```
puts [get_break_offset]
```

Returns:

```
32
```

get_break_slave_desc

Usage

```
get_break_slave_desc
```

Options

None

Description

Returns the slave descriptor associated with the processor break address. If null, then the break device is internal to the processor (debug module);

Example

```
puts [get_break_slave_desc]
```

Returns:

```
onchip_ram_64_kbytes
```

get_cpu_name

Usage

```
get_cpu_name
```

Options

None

Description

Returns the name of the BSP specific processor.

Example

```
puts [get_cpu_name]
```

Returns:

```
cpu_0
```

get_current_memory_regions

Usage

```
get_current_memory_regions
```

Options

None

Description

Returns a sorted list of records representing the existing linker script memory regions. Each record in the list represents a memory region. Each record is a list containing the region name, associated memory device slave descriptor, offset, and span, in that order.

Example

```
puts [get_current_memory_regions]
```

Returns:

```
{reset onchip_ram0 0 32} {onchip_ram0 onchip_ram0 32 1048544}
```

get_current_section_mappings

Usage

```
get_current_section_mappings
```

Options

None

Description

Returns a list of lists for all the current section mappings. Each list represents a section mapping with the format {section_name memory_region}. The order of the section mappings matches their order in the linker script.

Example

```
puts [get_current_section_mappings]
```

Returns:

```
{.text onchip_ram0} {.rodata onchip_ram0} {.rwddata onchip_ram0}  
  {.bss onchip_ram0} {.heap onchip_ram0} {.stack onchip_ram0}
```

get_default_memory_regions**Usage**

```
get_default_memory_regions
```

Options

None

Description

Returns a sorted list of records representing the default linker script memory regions. The default linker script memory regions are the best guess for memory regions based on the reset address and exception address of the processor associated with the BSP, and all other processors in the system that share memories with the processor associated with the BSP. Each record in the list represents a memory region. Each record is a list containing the region name, associated memory device slave descriptor, offset, and span, in that order.

Example

```
puts [get_default_memory_regions]
```

Returns:

```
{reset onchip_ram0 0 32} {onchip_ram0 onchip_ram0 32 1048544}
```

get_driver**Usage**

```
get_driver <module_name>
```

Options

- *<module_name>*: String with the name of the module instance to query.

Description

Returns the driver name associated with the specified module instance. The format is *<driver name>* followed by a colon and the version (if provided). Returns the string "none" if there is no driver associated with the specified module instance name.

Examples

- Example 1:

```
puts [get_driver jtag_uart]
```

Returns:

```
altera_avalon_jtag_uart_driver:7.2
```

■ Example 2:

```
puts [get_driver onchip_ram_64_kbytes]
```

Returns:

```
none
```

get_enabled_sw_packages

Usage

```
get_enabled_sw_packages
```

Options

None

Description

Returns a list of currently enabled software packages. The format of each entry in the list is the software package name followed by a colon and the version number (if provided).

Example

```
puts [get_enabled_sw_packages]
```

Returns:

```
altera_hostfs:7.2
```

get_exception_offset

Usage

```
get_exception_offset
```

Options

None

Description

Returns the byte offset of the processor exception address.

Example

```
puts [get_exception_offset]
```

Returns:

```
32
```

get_exception_slave_desc

Usage

```
get_exception_slave_desc
```

Options

None

Description

Returns the slave descriptor associated with the processor exception address.

Example

```
puts [get_exception_slave_desc]
```

Returns:

```
onchip_ram_64_kbytes
```

get_fast_tlb_miss_exception_offset**Usage**

```
get_fast_tlb_miss_exception_offset
```

Options

None

Description

Returns the byte offset of the processor fast translation lookaside buffer (TLB) miss exception address. Only a processor with an MMU has such an exception address.

Example

```
puts [get_fast_tlb_miss_exception_offset]
```

Returns:

```
32
```

get_fast_tlb_miss_exception_slave_desc**Usage**

```
get_fast_tlb_miss_exception_slave_desc
```

Options

None

Description

Returns the slave descriptor associated with the processor fast TLB miss exception address. Only a processor with an MMU has such an exception address.

Example

```
puts [get_fast_tlb_miss_exception_slave_desc]
```

Returns:

```
onchip_ram_64_kbytes
```

get_interrupt_controller_id

Usage

```
get_interrupt_controller_id <slave_desc>
```

Options

- *<slave_desc>*: String with the slave descriptor to query.

Description

Returns the interrupt controller ID of the slave descriptor (-1 if not a connected interrupt controller).

get_irq_interrupt_controller_id

Usage

```
get_irq_interrupt_controller_id <slave_desc>
```

Options

- *<slave_desc>*: String with the slave descriptor to query.

Description

Returns the interrupt controller ID connected to the IRQ associated with the slave descriptor (-1 if none).

get_irq_number

Usage

```
get_irq_number <slave_desc>
```

Options

- *<slave_desc>*: String with the slave descriptor to query.

Description

Returns the interrupt request number of the slave descriptor, or -1 if no interrupt request number is found.

get_memory_region

Usage

```
get_memory_region <name>
```

Options

- *<name>*: String with the name of the memory region.

Description

Returns the linker script region information for the specified region. The format of the region is a list containing the region name, associated memory device slave descriptor, offset, and span in that order.

Example

```
puts [get_memory_region reset]
```

Returns:

```
reset onchip_ram0 0 32
```

get_module_class_name**Usage**

```
get_module_class_name <module_name>
```

Options

- *<module_name>*: String with the module instance name to query.

Description

Returns the name of the module class associated with the module instance.

Example

```
puts [get_module_class_name jtag_uart0]
```

Returns:

```
altera_avalon_jtag_uart
```

get_module_name**Usage**

```
get_module_name <slave_desc>
```

Options

- *<slave_desc>*: String with the slave descriptor to query.

Description

Returns the name of the module instance associated with the slave descriptor. If a module with one slave, or if it has multiple slaves connected to the same resource, the slave descriptor is the same as the module name. If a module has multiple slaves that do not connect to the same resource, the slave descriptor consists of the module name followed by an underscore and the slave name.

Example

```
puts [get_module_name multi_jtag_uart0_s1]
```

Returns:

```
multi_jtag_uart0
```

get_reset_offset**Usage**

```
get_reset_offset
```

Options

None

Description

Returns the byte offset of the processor reset address.

Example

```
puts [get_reset_offset]
```

Returns:

```
0
```

get_reset_slave_desc

Usage

```
get_reset_slave_desc
```

Options

None

Description

Returns the slave descriptor associated with the processor reset address.

Example

```
puts [get_reset_slave_desc]
```

Returns:

```
onchip_ram_64_kbytes
```

get_section_mapping

Usage

```
get_section_mapping <section_name>
```

Options

- *<section_name>*: String with the section name to query.

Description

Returns the name of the memory region for the specified linker section. Returns null if the linker section does not exist.

Example

```
puts [get_section_mapping .text]
```

Returns:

```
onchip_ram0
```

get_setting

Usage

```
get_setting <name>
```

Options

- *<name>*: String with the name of the setting to get.

Description

Returns the value of the specified BSP setting. `get_setting` returns boolean settings with the value 1 or 0. If the value of the setting is an empty string, `get_setting` returns "none".

The `get_setting` command is equivalent to the `--get` command-line option.

Example

```
puts [get_setting hal.enable_gprof]
```

Returns:

```
0
```

get_setting_desc

Usage

```
get_setting_desc <name>
```

Options

- *<name>*: String with the name of the setting to get the description for.

Description

Returns a string describing the BSP setting.

Example

```
puts [get_setting_desc hal.enable_gprof]
```

Returns:

```
"This example compiles the code with gprof profiling enabled and links \  
the application ELF with the GPROF library. If true, adds \  
-DALT_PROVIDE_GMON to ALT_CPPFLAGS and -pg to ALT_CFLAGS in \  
public.mk."
```

get_slave_descs

Usage

```
get_slave_descs
```

Options

None

Description

Returns a sorted list of all the slave descriptors connected to the Nios II processor.

Example

```
puts [get_slave_descs]
```

Returns:

```
jtag_uart0 onchip_ram0
```

is_char_device

Usage

```
is_char_device <slave_desc>
```

Options

- *<slave_desc>*: String with the slave descriptor to query.

Description

Returns a boolean value that indicates whether the slave descriptor is a character device.

Examples

- Example 1:

```
puts [is_char_device jtag_uart]
```

Returns:

```
1
```

- Example 2:

```
puts [is_char_device onchip_ram_64_kbytes]
```

Returns:

```
0
```

is_connected_interrupt_controller_device

Usage

```
is_connected_interrupt_controller_device <slave_desc>
```

Options

- *<slave_desc>*: String with the slave descriptor to query.

Description

Returns a boolean value that indicates whether the slave descriptor is an interrupt controller device that is connected to the processor so that the interrupt controller sends interrupts to the processor.

is_connected_to_data_master

Usage

```
is_connected_to_data_master <slave_desc>
```

Options

- *<slave_desc>*: String with the slave descriptor to query.

Description

Returns a boolean value that indicates whether the slave descriptor is connected to a data master.

is_connected_to_instruction_master**Usage**

```
is_connected_to_instruction_master <slave_desc>
```

Options

- *<slave_desc>*: String with the slave descriptor to query.

Description

Returns a boolean value that indicates whether the slave descriptor is connected to an instruction master.

is_ethernet_mac_device**Usage**

```
is_ethernet_mac_device <slave_desc>
```

Options

- *<slave_desc>*: String with the slave descriptor to query.

Description

Returns a boolean value that indicates whether the slave descriptor is an Ethernet MAC device.

is_flash**Usage**

```
is_flash <slave_desc>
```

Options

- *<slave_desc>*: String with the slave descriptor to query.

Description

Returns a boolean value that indicates whether the slave descriptor is a flash memory device.

is_memory_device**Usage**

```
is_memory_device <slave_desc>
```

Options

- *<slave_desc>*: String with the slave descriptor to query.

Description

Returns a boolean value that indicates whether the slave descriptor is a memory device.

Examples

- Example 1:

```
puts [is_memory_device jtag_uart]
```

Returns:

```
0
```

- Example 2:

```
puts [is_memory_device onchip_ram_64_kbytes]
```

Returns:

```
1
```

is_non_volatile_storage

Usage

```
is_non_volatile_storage <slave_desc>
```

Options

- *<slave_desc>*: String with the slave descriptor to query.

Description

Returns a boolean value that indicates whether the slave descriptor is a non-volatile storage device.

is_timer_device

Usage

```
is_timer_device <slave_desc>
```

Options

- *<slave_desc>*: String with the slave descriptor to query.

Description

Returns a boolean value that indicates whether the slave descriptor is a timer device.

log_debug

Usage

```
log_debug <message>
```

Options

- *<message>*: String with message to log.

Description

Displays a message to the host's stdout when the logging level is debug.

log_default**Usage**

```
log_default <message>
```

Options

- *<message>*: String with message to log.

Description

Displays a message to the host's stdout when the logging level is default or higher.

Example

```
log_default "This is a default message."
```

Displays:

```
INFO: Tcl message: "This is a default message."
```

log_error**Usage**

```
log_error <message>
```

Options

- *<message>*: String with message to log.

Description

Displays a message to the host's stderr, regardless of logging level.

log_verbose**Usage**

```
log_verbose <message>
```

Options

- *<message>*: String with message to log.

Description

Displays a message to the host's stdout when the logging level is verbose or higher.

set_driver

Usage

```
set_driver <driver_name> <module_name>
```

Options

- *<driver_name>*: String with the name of the device driver to use.
- *<module_name>*: String with the name of the module instance to set.

Description

Selects the specified device driver for the specified module instance. The *<driver_name>* argument includes a version number, delimited by a colon (:). If you omit the version number, `set_driver` uses the latest available version of the driver which is compatible with the SOPC Builder module specified by the *<module_name>* argument.

If *<driver_name>* is `none`, the specified module instance does not use a driver. If *<driver_name>* is not `none`, it must be the name of the associated component class.

Examples

- Example 1:

```
set_driver altera_avalon_jtag_uart_driver:7.2 jtag_uart
```

- Example 2:

```
set_driver none jtag_uart
```

set_ignore_file

Usage

```
set_ignore_file <software_component_name> <file_name> <ignore>
```

Options

- *<software_component_name>*: Name of the driver, software package, or operating system to which the file belongs.
- *<file_name>*: Name of the file.
- *<ignore>*: Set to `true` to ignore (not generate or copy) the file, `false` to generate or copy the file normally.

Description

You can use this command to have a specific BSP file ignored (not generated or copied) during BSP generation. This command allows you to take ownership of a specific file, modify it, and prevent the SBT from overwriting your modifications.

<software_component_name> can have one of the following values:

- *<driver_name>*—The name of a driver, as specified with the `create_driver` command in the `*_sw.tcl` file that defines the driver. Specifies that *<file_name>* is a copied file associated with a device driver.

- `<software_package_name>`—The name of a software package, specified with the `create_sw_package` command in the `*_sw.tcl` file that defines the package. Specifies that `<file_name>` is a copied file associated with a software package.
- `<OS_name>`—The name of an OS, specified with the `create_os` command in the `*_sw.tcl` file that defines the OS, and is used in the `nios2-bsp-create-settings` to specify the BSP type. Specifies that `<file_name>` is a copied file associated with an OS.
- `generated`—Specifies that `<file_name>` is a generated top-level BSP file. The list of generated BSP files depends on the BSP type. For a list of generated files associated with HAL and MicroC/OS-II BSPs, refer to “Details of BSP Creation” in the *Nios II Software Build Tools* chapter of the *Nios II Software Developer’s Handbook*. For a list of generated files associated with a third-party OS, refer to the OS supplier’s documentation.

set_setting

Usage

```
set_setting <name> <value>
```

Options

- `<name>`: String with the name of the setting.
- `<value>`: String with the value of the setting.

Description

Sets the value for the specified BSP setting. Legal values for boolean settings are true, false, 1, and 0. Use the keyword none instead of an empty string to set a string to an empty value. The `set_setting` command is equivalent to the `--set` command-line option.

Example

```
set_setting hal.enable_gprof true
```

update_memory_region

Usage

```
update_memory_region <name> <slave_desc> <offset> <span>
```

Options

- `<name>`: String with the name of the memory region to update.
- `<slave_desc>`: String with the slave descriptor of the memory device for this region.
- `<offset>`: String with the byte offset of the memory region from the memory device base address.
- ``: String with the span of the memory region in bytes.

Description

Updates an existing memory region for the linker script. This memory region must not overlap with any other memory region and must be within the memory range of the associated slave descriptor. The offset and span are decimal numbers unless prefixed with 0x.

Example

```
update_memory_region onchip_ram0 onchip_ram0 0 0x100000
```

update_section_mapping**Usage**

```
update_section_mapping <section_name> <memory_region_name>
```

Options

- *<section_name>*: String with the name of the linker section.
- *<memory_region_name>*: String with the name of the memory region to map.

Description

Updates the specified linker section. The linker creates the section mappings in the order in which they appear in the linker script.

Example

```
update_section_mapping .text onchip_ram0
```

add_default_memory_regions**Usage**

```
add_default_memory_regions
```

Description

Defaults the BSP to use default linker script memory regions. The default linker script memory regions are the best guess for memory regions based on the reset address and exception address of the processor associated with the BSP, and all other processors in the system that share memories with the processor associated with the BSP.

create_bsp**Usage**

```
create_bsp <bspType> <bsp version> <processor name> <sopcinfo>
```

Options

- *<bspType>*: Type of BSP to create.
- *<bsp version>*: Version of BSP software element to utilize.
- *<processor name>*: Name of processor instance for BSP
- *<sopcinfo>*: .sopcinfo generated file that describes the system the BSP is for.

Description

Creates a new BSP.

generate_bsp**Usage**

```
generate_bsp <bspDir>
```

Options

- <bspDir>: BSP directory to generate files to.

Description

Generates a new BSP.

get_available_bsp_type_versions**Usage**

```
get_available_bsp_type_versions <bsp_type> <sopcinfolpath>
```

Options

- <bsp_type>: BSP type identifier (e.g. hal, ucousii).
- <sopcinfolpath>: SOPC Information File path. Its parent folder might include custom BSP IP software components (*_sw.tcl).

Description

Gets the available BSP type versions.

get_available_bsp_types**Usage**

```
get_available_bsp_types <sopcinfolpath>
```

Options

- <sopcinfolpath>: SOPC Information File path. Its parent folder might include custom BSP IP software components (*_sw.tcl).

Description

Gets the available BSP type identifiers.

get_available_cpu_architectures**Usage**

```
get_available_cpu_architectures
```

Description

Gets the available processor architectures.

get_available_cpu_names

Usage

```
get_available_cpu_names <sopcinfolpath>
```

Options

- <sopcinfolpath>: SOPC Information File path that contains processor instances

Description

Gets the processor names given a SOPC system.

get_available_software

Usage

```
get_available_software <bsp_type> <filter> <sopcinfolpath>
```

Options

- <bsp_type>: BSP type identifier (e.g. hal, ucousii).
- <sopcinfolpath>: SOPC Information File path. Its parent folder might include custom BSP IP software components (*_sw.tcl).
- <filter>: A filter can be applied to restrict results. Filters are "all", "drivers", "sw_packages", and "os_elements". Comma separated tokens are acceptable.

Description

Gets the available software (drivers, software packages, and bsp components) for a given BSP type.

get_available_software_setting_properties

Usage

```
get_available_software_setting_properties <setting_name> \  
  <software_name> <software_version> <sopcinfolpath>
```

Options

- <software_name>: Name of a software component (e.g. "altera_avalon_uart_driver", or "hal").
- <software_version>: Enter "default" for latest version, or a specific version number.
- <setting_name>: Name of a selected software component setting to get properties for (e.g. hal.linker.allow_code_at_reset).
- <sopcinfolpath>: SOPC Information File path. Its parent folder might include custom BSP IP software components (*_sw.tcl).

Description

Gets the available setting names for a software component.

get_available_software_settings

Usage

```
get_available_software_settings <software_name> <software_version> \  
  <sopcinfol_path>
```

Options

- <software_name>: Name of a software component (e.g. altera_avalon_uart_driver).
- <software_version>: Enter "default" for latest version, or a specific version number.
- <sopcinfol_path>: SOPC Information File path. Its parent folder can include custom BSP IP software components (*_sw.tcl).

Description

Gets the available setting names for a software component.

get_bsp_version

Usage

```
get_bsp_version
```

Description

Gets the version of the BSP operating system software element.

get_cpu_architecture

Usage

```
get_cpu_architecture <processor_name> <sopcinfol_path>
```

Options

- <processor_name>: processor instance name
- <sopcinfol_path>: SOPC Information File path that contains processor_name instance

Description

Gets the processor architecture (e.g. nios2) of a specified processor instance given a SOPC system.

get_sopcinfol_file

Usage

```
get_sopcinfol_file
```

Description

Returns the path of the BSP specific SOPC Information File.

get_supported_bsp_types

Usage

```
get_supported_bsp_types <processor_name> <sopcinfolpath>
```

Options

- <processor_name>: processor instance name
- <sopcinfolpath>: SOPC Information File path. Its parent folder can include custom BSP IP software components (*_sw.tcl).

Description

Gets the BSP types supported for a given processor and SOPC system.

is_bsp_hal_extension

Usage

```
is_bsp_hal_extension
```

Description

Returns a boolean value that indicates whether the BSP instantiated is of a type based on Altera HAL.

is_bsp_lwhal_extension

Usage

```
is_bsp_lwhal_extension
```

Description

Returns a boolean value that indicates whether the BSP instantiated is of a type based on Altera Lightweight HAL.

open_bsp

Usage

```
open_bsp <settingsFile>
```

Options

- <settingsFile>: .bsp settings file to open.

Description

Opens an existing BSP.

save_bsp

Usage

```
save_bsp <settingsFile>
```

Options

- `<settingsFile>`: .bsp settings file to save BSP to.

Description

Saves a new BSP.

set_bsp_version**Usage**

```
set_bsp_version <version>
```

Options

- `<version>`: Version of BSP type software element to use.

Description

Sets the version of the BSP operating system software element to a specific value. The value "default" uses the latest version available. If this call is not used, the BSP is created using the 'default' BSP software element version.

set_logging_mode**Usage**

```
set_logging_mode <mode>
```

Options

- `<mode>`: Logging Mode: 'silent', 'default', 'verbose', 'debug'

Description

Sets the verbosity level of the logger. Useful to eliminate tool informative messages

Tcl Commands for BSP Generation Callbacks

If you are defining a device driver or a software package, you can define Tcl callback functions to run whenever a BSP is generated containing your driver or package. Tcl callback functions enable you to create settings dynamically for the driver or package. This capability is essential when the driver or package settings must be customized to the hardware configuration, or to other BSP settings.

Tcl callback scripts are defined and controlled from the `*_sw.tcl` file associated with the driver or package. In `*_sw.tcl`, you can specify where the Tcl functions come from, when function runs, and the scope of each Tcl function's operation.


When the BSP is generated with your driver or software package, the settings you define in the callback scripts are inserted in `settings.bsp`.

You specify the source of the callback functions with the `set_sw_property` command, using the `callback_source_file` property.

A Tcl callback function can run at one of the following times:

- BSP initialization

- BSP generation
- BSP validation

 Although you can specify a new setting's value when you create the setting at BSP initialization, the setting's value can change between initialization and generation. For example, the BSP developer might edit the setting in the Nios II BSP Editor.

A Tcl callback can function in either of the following scopes:

- Component class
- Component instance

You specify each callback function's runtime environment by using the appropriate property in the `set_sw_property` command, as shown in [Table 15-7](#).

Table 15-7. Callback Properties


Property as specified in <code>set_sw_property</code>	Run time	Scope	Callback Arguments
<code>initialization_callback</code>	Initialization	Component instance	Component instance name
<code>validation_callback</code>	Validation	Component instance	Component instance name
<code>generation_callback</code>	Generation	Component instance	Component instance name, BSP generate target directory, driver BSP subdirectory (1)
<code>class_initialization_callback</code>	Initialization	Component class	Driver class name
<code>class_validation_callback</code>	Validation	Component class	Driver class name
<code>class_generation_callback</code>	Generation	Component class	Driver class name, BSP generate target directory, driver BSP subdirectory (1)

Note to Table 15-7:

(1) The BSP subdirectory into which the driver or package files are copied

Tcl callbacks have access to a specialized set of commands, described in this section. In addition, Tcl callbacks can use any read-only BSP setting Tcl command.

 Refer to [“Tcl Commands for BSP Settings” on page 15-72](#) for details about BSP setting Tcl commands.

 When a Tcl callback creates a setting, it can specify the value. However, callbacks cannot change the value of a pre-existing setting.

add_class_sw_setting

Usage

`add_class_sw_setting <setting-name> <setting-type>`

Options

- `<setting-name>`: Name of the setting to persist in the BSP settings file. This is prepended with the driver class name associated with this callback script

- *<setting-type>*: Type of the setting to persist in the BSP settings file.

Description

Creates a BSP setting that is associated with a particular software driver element class. The `set_class_sw_setting_property` command is required to set the values for fields pertaining to a BSP software setting definition. This command is only valid for a callback script. A callback script is set in the driver's `*_sw.tcl` file, using the command `set_sw_property callback_source_file <filename>`.

Example

```
add_class_sw_setting MY_FAVORITE_SETTING String
```

add_class_systemh_line

Usage

```
add_class_systemh_line <macro-name> <macro-value>
```

Options

- *<macro-name>*: Macro to be added to the system.h file for the generated BSP
- *<macro-value>*: Value associated with the macro-name to be added to the system.h file for the generated BSP

Description

This adds a system.h assignment or macro during a driver callback execution. The BSP typically uses this during the generate phase depending on the generator. This command is only valid for a callback script. A callback script is set in the driver's `*_sw.tcl` file, using the command `set_sw_property callback_source_file <filename>`.

Example

```
add_class_systemh_line MY_MACRO "Macro_Value";
```

add_module_sw_property

Usage

```
add_module_sw_property <property-name> <property-value>
```

Options

- *<property-name>*: Name of the property to add to the BSP for a module instance
- *<property-value>*: Value of the property to add to the BSP for a module instance

Description

This adds a software property to the BSP driver of this module instance. The BSP typically uses this during the generate phase depending on the generator. This command is only valid for a callback script. A callback script is set in the driver's `*_sw.tcl` file, using the command `set_sw_property callback_source_file <filename>`.

Example

```
add_module_sw_setting MY_FAVORITE_SETTING String
```

add_module_sw_setting**Usage**

```
add_module_sw_setting <setting-name> <setting-type>
```

Options

- *<setting-name>*: Name of the setting to persist in the BSP settings file. This is prepended with the module name associated with this callback script
- *<setting-type>*: Type of the setting to persist in the BSP settings file.

Description

Creates a BSP setting that is associated with a particular instance of hardware module in a SOPC system. The `set_module_sw_setting_property` command is required to set the values for fields pertaining to a BSP software setting definition. This command is only valid for a callback script. A callback script is set in the driver's `*_sw.tcl` file, using the command `set_sw_property callback_source_file <filename>`.

Example

```
add_module_sw_setting MY_FAVORITE_SETTING String
```

add_module_systemh_line**Usage**

```
add_module_systemh_line <macro-name> <macro-value>
```

Options

- *<macro-name>*: Macro to be added to the system.h file for the generated BSP
- *<macro-value>*: Value associated with the macro-name to be added to the system.h file for the generated BSP

Description

This adds a system.h assignment or macro during a driver callback execution. The BSP typically uses this during the generate phase depending on the generator. This command is only valid for a callback script. A callback script is set in the driver's `*_sw.tcl` file, using the command `set_sw_property callback_source_file <filename>`.

Example

```
add_module_systemh_line MY_MACRO "Macro_Value";
```

get_class_peripheral**Usage**

```
get_class_peripheral <instance-name> <irq-number>
```

Options

- *<instance-name>*: Name of EIC module instance to find connected peripheral for.
- *<irq-number>*: IRQ number to locate connected peripheral device

Description

This command is used on an EIC instance callback to obtain a peripheral slave descriptor connected to a specific IRQ port number. This command is only valid for a callback script.

Example

```
get_class_peripheral eic_1 $irq_2;
```

get_module_assignment**Usage**

```
get_module_assignment <assignment-name>
```

Options

- *<assignment-name>*: Name of the module assignment to retrieve the value for, as defined for the module instance in the **.sopcinfo** file

Description

Given a module assignment key, return the assignment value of a module associated with the callback script using this command. The callback script must be set in the ***_sw.tcl** file using the following command:

```
set_sw_property callback_source_file <filename>
```

Example

```
puts [get_module_assignment embeddedsw.configuration.isMemoryDevice]
```

Returns:

```
true
```

get_module_name**Usage**

```
get_module_name
```

Options

None

Description

Returns the name of the module associated with the callback script using this command. The callback script must be set in the ***_sw.tcl** file using the following command:

```
set_sw_property callback_source_file <filename>
```

Example

```
puts [get_module_name]
```

Returns:

```
jtag_uart
```

get_module_peripheral

Usage

```
get_module_peripheral <irq-number>
```

Options

- *<irq-number>*: IRQ number to locate connected peripheral device

Description

This command is used on an EIC instance callback to obtain a peripheral slave descriptor connected to a specific IRQ port number. This command is only valid for a callback script.

Example

```
get_module_peripheral 2;
```

get_module_sw_setting_value

Usage

```
get_module_sw_setting_value <setting-name>
```

Options

- *<setting-name>*: Name of the module software setting to retrieve the value for, as defined by the `add_module_sw_setting` command.

Description

Given a module software setting name, return the setting value. The callback script using this command must be set in the `*_sw.tcl` file using the following command:

```
set_sw_property callback_source_file <filename>
```

You can use this command in a generation or validation callback to retrieve the current value of a setting created in an initialization callback.

Example

```
puts [get_module_sw_setting_value MY_SETTING]
```

Returns:

```
"My setting value"
```

get_peripheral_property

Usage

```
get_peripheral_property <slave-descriptor> <property-name>
```

Options

- *<slave-descriptor>*: Slave descriptor of a connected peripheral device
- *<property-name>*: Property name to query from the connected peripheral device

Description

This command is used on an EIC instance callback to obtain a connected peripheral property value. This command is only valid for a callback script. A callback script is set in the driver's *_sw.tcl file, using the command `set_sw_property callback_source_file <filename>`.

Example

```
get_peripheral_property jtag_uart supports_preemption;
```

remove_class_systemh_line**Usage**

```
remove_class_systemh_line <macro-name>
```

Options

- *<macro-name>*: Macro to be removed to the system.h file for the generated BSP

Description

This removes a system.h assignment or macro during a driver callback execution. The BSP typically uses this during the generate phase depending on the generator. This command is only valid for a callback script. A callback script is set in the driver's *_sw.tcl file, using the command `set_sw_property callback_source_file <filename>`.

Example

```
remove_class_systemh_line MY_MACRO;
```

remove_module_systemh_line**Usage**

```
remove_module_systemh_line <macro-name>
```

Options

- *<macro-name>*: Macro to be removed to the system.h file for the generated BSP

Description

This removes a system.h assignment or macro during a driver callback execution. The BSP typically uses this during the generate phase depending on the generator. This command is only valid for a callback script. A callback script is set in the driver's *_sw.tcl file, using the command `set_sw_property callback_source_file <filename>`.

Example

```
remove_module_systemh_line MY_MACRO;
```

set_class_sw_setting_property

Usage

```
set_class_sw_setting_property <setting-name> <property> <value>
```

Options

- *<setting-name>*: Name of the setting to persist in the BSP settings file associated with the driver class of this callback script
- *<property>*: Name of the setting property to update
- *<value>*: Value of the setting property to update

Description

Update a driver class software setting property. The setting must be added using the `add_class_sw_setting` command before calling this method. This command is only valid for a callback script. A callback script is set in the driver's `*_sw.tcl` file, using the command `set_sw_property callback_source_file <filename>`.

You can set the following setting properties:

- destination
- identifier
- value
- default_value
- description
- restrictions
- group

Example

```
set_class_sw_setting_property MY_FAVORITE_SETTING default-value '42'
```

set_module_sw_setting_property

Usage

```
set_module_sw_setting_property <setting-name> <property> <value>
```

Options

- *<setting-name>*: Name of the setting to persist in the BSP settings file associated with the SOPC module of this callback script
- *<property>*: Name of the setting property to update
- *<value>*: Value of the setting property to update

Description

Update a module's software setting property. The setting must be added using the `add_module_sw_setting` command before calling this method. This command is only valid for a callback script. A callback script is set in the driver's `*_sw.tcl` file, using the command `set_sw_property callback_source_file <filename>`.

You can set the following setting properties:

- destination
- identifier
- value
- default_value
- description
- restrictions
- group

Example


```
set_module_sw_setting_property MY_FAVORITE_SETTING default-value '42'
```

Tcl Commands for Drivers and Packages

This section describes the tools that you use to specify and manipulate the settings and characteristics of a custom software package or driver. Typically, when creating a custom software package or device driver, or importing a package or driver from another development environment, you need these more powerful tools. To manipulate settings on existing software packages and device drivers, refer to [“Settings” on page 15-27](#) and [“Tcl Commands for BSP Settings” on page 15-72](#).

A device driver and a software package are both collections of source files added to the BSP. A device driver is associated with a particular component class (for example, `altera_avalon_jtag_uart`). A software package is not associated with any particular component class, but implements a functionality such as TCP/IP.

To define a device driver or software package, you create a Tcl script defining its characteristics. This section describes the Tcl commands available to define device drivers and software packages.

 For more information about creating Tcl scripts, refer to [“Tcl Scripts for BSP Settings”](#) in the *Nios II Software Build Tools* chapter of the *Nios II Software Developer’s Handbook*.

The following commands are available for device driver and software package creation:

- [“add_sw_property” on page 15-108](#)
- [“add_sw_setting” on page 15-110](#)
- [“create_driver” on page 15-113](#)
- [“create_os” on page 15-113](#)
- [“create_sw_package” on page 15-113](#)
- [“set_sw_property” on page 15-114](#)

add_sw_property

Usage

```
add_sw_property <property> <value>
```

Options

- *<property>*: Name of property.
- *<value>*: Value assigned, or appended to the current value.

Description

This command defines a property for a device driver or software package. A property is a list of values (for example, a list of file names). The `add_sw_property` command defines a property if it is not already defined. The command appends a new value to the list of values if the property is already defined.

In the case of a property consisting of a file name or directory name, use a relative path. Specify the path relative to the directory containing the Tcl script.

This command supports the following properties:

- `asm_source`—Adds a Nios II assembly language source file (`.s` or `.S`) to BSPs containing your package. **nios2-bsp-generate-files** copies assembly source files into a BSP and adds them to the source build list in the BSP makefile. This property is optional.
- `c_source`—Adds a C source file (`.c`) to BSPs containing your package. **nios2-bsp-generate-files** copies C source files into a BSP and adds them to the source build list in the BSP makefile. This property is optional.
- `cpp_source`—Adds a C++ source file (`.cpp`, `.cc`, or `.cxx`) to BSPs containing your package. **nios2-bsp-generate-files** copies the C++ source files into a BSP and adds them to the source build list in the BSP makefile. This property is optional.
- `include_source`—Adds an include file (typically `.h`) to BSPs containing your package. **nios2-bsp-generate-files** copies include files into a BSP, but does not add them to the generated makefile. This property is optional.
- `include_directory`—Adds a directory to the `ALT_INCLUDE_DIRS` variable in the BSP's **public.mk** file. Adding a directory to `ALT_INCLUDE_DIRS` allows all source files to find include files in this directory. `add_sw_property` adds the path to the generated public makefile shared by the BSP and applications or libraries referencing it. `add_sw_property` compiles all files with the include directory listed in the compiler arguments.
This property is optional.
- `lib_source`—Adds a precompiled library file (typically `.a`) to each BSP containing the driver or package. **nios2-bsp-generate-files** copies the precompiled library file into the BSP directory and adds both the library file name and the path (required to locate the library file) into the BSP's **public.mk** file. Applications using the BSP link with the library file.
The library file name must conform to the following pattern:
lib<name>.a
where *<name>* is a nonempty string.
Example:
`add_sw_property lib_source HAL/lib/libcomponent.a`
This property is optional.

- `specific_compatible_hw_version`—Specifies that the device driver only supports the specified component hardware version. See the `version` property of the `set_sw_property` command for information about version strings. This property applies only to device drivers (see the `create_driver` command), not to software packages. If your driver supports all versions of a peripheral after a specific release, use the `set_property min_compatible_hw_version` command instead. This property is optional. This property is only available for device drivers.
- `supported_bsp_type`—Adds a specific BSP type (operating system) to the list of supported operating systems that the driver or software package supports. Specify `HAL` if the software supports the Altera HAL, or operating systems that extend it. If your software is operating system-neutral and works on multiple HAL-based operating systems, state `HAL` only. If your software or driver contains code that depends on a particular operating system, state compatibility with that operating system only, but not `HAL`.
The name of another operating system to support must match the name of the operating system exactly. This operating system name string is the same as that used to create a BSP with the `nios2-bsp-*` commands, as well as in the `.tcl` script that describes the operating system, in its `create_os` command.
When a user creates a BSP with an operating system that extends HAL, such as `UCOSII` and the BSP tools select a driver for a particular hardware module, precedence is given to drivers which state compatibility with a that specific operating system (OS) before a more generic driver stating `HAL` compatibility. This property is only available for device drivers and software packages. This property must be set to at least one operating system.
- `alt_cppflags_addition`—Adds a line of arbitrary text to the `ALT_CPPFLAGS` variable in the BSP `public.mk` file. This technique can be useful if you wish to have a static compilation flag or definition that all BSP, application, and library files receive during software build. This property is optional.
- `excluded_hal_source`—Specifies a file to exclude from the a BSP generated with an operating system that extends HAL. The value is the path to a BSP file to exclude, with respect to the BSP root. This property is optional.
- `systemh_generation_script`—Specifies a `.tcl` script to execute during generation of the BSP `system.h` file. This script runs with the tcl commands available to other BSP settings tcl scripts, and allow you to influence the contents of the `system.h` file. This property is available only to operating systems, created with the `create_os` command. This property is optional.

add_sw_setting

Usage

```
add_sw_setting <type> <destination> <displayName>
               <identifier> <value> <description>
```

Options

- `<type>`: Setting type - Boolean, QuotedString, UnquotedString.
- `<destination>`: The destination BSP file associated with the setting, or the module generator that processes this setting.

- *<displayName>*: Setting name.
- *<identifier>*: Name of the macro created for a generated destination file.
- *<value>*: Default value of the setting.
- *<description>*: Setting description.

Description

This command creates a BSP setting associated with a software package or device driver. The setting is available whenever the software package or device driver is present in the BSP. **nios2-bsp-generate-files** converts the setting and its value into either a C preprocessor macro or BSP makefile variable. `add_sw_setting` passes macro definitions to the compiler using the `-D` command-line option, or adds them to the **system.h** file as `#define` statements.

The setting only exists once even if there are multiple instances of a software package. Set or get the setting with the `--set` and `--get` command-line options of the **nios2-bsp**, **nios2-bsp-create-settings**, **nios2-bsp-query-settings**, and **nios2-bsp-update-settings** commands. You can also use the BSP Tcl commands `set_setting` and `get_setting` to set or get the setting. The value of the setting persists in the BSP settings file.

To create a setting, you must define each of the following parameters:

- `type`—This parameter formats the setting value during BSP generation. The following supported types and usage restrictions apply:
 - `boolean_define_only`—Defines a macro if the setting's value is 1 or true. Example: `#define LCD_PRESENT`. No macro is defined if the setting's value is 0 or false. This setting type supports the `system_h_define` and `public_mk_define` generators.
 - `boolean`—Defines a macro or makefile variable to 1 (if the value is 1 or true) or 0 (if the value is 0 or false). Example: `#define LCD_PRESENT 1`. This type supports all generators.
 - `character`—Defines a macro with a single character with single quotes around the character. Example: `#define DELIMITER ':'`. This type supports the `system_h_define` destination.
 - `decimal_number`—Decimal numbers define a macro or makefile variable with an unquoted decimal (integer) number. Example: `#define NUM_COPROCESSORS 3`. This type supports all destinations.
 - `double`—Double numbers have a macro name and setting value in the destination file including decimal point. Example: `#define PI 3.1416`. This type supports the `system_h_define` destination.
 - `float`—Float numbers have a macro name and setting value in the destination file including decimal point and `f` character. Example: `#define PI 3.1416f`. This type supports the `system_h_define` destination.
 - `hex_number`—Hex numbers have a macro name and setting value in the destination file with `0x` prepended to the value. Example: `#define LCD_SIZE 0x1000`. This type supports the `system_h_define` destination.

- `quoted_string`—Quoted strings always have the macro name and setting value added to the destination files. In the destination, the setting value is enclosed in quotation marks. Example:


```
#define DFLT_ERR "General error"
```

 If the setting value contains white space, you must also place quotation marks around the value string in the Tcl script.
This type supports the `system_h_define` destination.
- `unquoted_string`—Unquoted strings define a macro or makefile variable with setting name and value in the destination file. In the destination file, the setting value is not enclosed in quotation marks. Example:


```
#define DFLT_ERROR Error
```

 This type supports all destinations.
- `destination`—The destination parameter specifies where `add_sw_setting` puts the setting in the generated BSP. `add_sw_settings` supports the following destinations:
 - `system_h_define`—With this destination, `add_sw_settings` formats settings as `#define <setting name> [<setting value>]` macros in the **system.h** file
 - `public_mk_define`—With this destination, `add_sw_settings` formats settings as `-D<setting name>[=<setting value>]` additions to the `ALT_CPPFLAGS` variable in the BSP **public.mk** file. **public.mk** passes the flag to the C preprocessor for each source file in the BSP, and in applications and libraries using the BSP.
 - `makefile_variable`—With this destination, `add_sw_settings` formats settings as makefile variable additions to the BSP makefile. The variable name must be unique in the makefile.
- `displayName`—The name of the setting. Settings exist in a hierarchical namespace. A period separates levels of the hierarchy. Settings created in your Tcl script are located in the hierarchy under the driver or software package name you specified in the `create_driver` or `create_sw_package` command. Example: `my_driver.my_setting`. The Nios II SBT adds the hierarchical prefix to the setting name.
- `identifier`—The name of the macro or makefile variable being defined. In a setting added to the **system.h** file at generation time, this parameter corresponds to the text immediately following the `#define` statement.
- `value`—The default value associated with the setting. If the user does not assign a value to the option, its value is this default value. Valid initial values are `true`, `1`, `false`, and `0` for `boolean` and `boolean_define_only` setting types, a single character for the `character` type, integer numbers for the `decimal_number` setting type, integer numbers with or without a `0x` prefix for the `hex_number` type, numbers with decimals for `float_number` and `double_number` types, or an arbitrary string of text for `quoted` and `unquoted_string` setting types. For string types, if the value contains any white space, you must enclose it in quotation marks.
- `description`—Descriptive text that is inserted along with the setting value and name in the **summary.html** file. You must enclose the description in quotation marks if it contains any spaces. If the description includes any special characters (such as quotation marks), you must escape them with the backslash (`\`) character. The description field is mandatory, but can be an empty string (`" "`).

create_driver

Usage

```
create_driver <name>
```

Options

- <name>: Name of device driver.

Description

This command creates a new device driver instance available for the Nios II SBT. This command must precede all others that describe the device driver in its Tcl script. You can only have one `create_driver` command in each Tcl script. If the `create_driver` command appears in the Tcl script, the `create_sw_package` and `create_os` commands cannot appear.

The name argument is usually distinct from all other device drivers and software packages that the SBT might locate. You can specify driver name identical to another driver if the driver you are describing has a unique version number assignment.

If your driver differs for different operating systems, you need to provide a unique name for each BSP type.

This command is required, unless you use the `create_sw_package` or `create_os` commands, as appropriate.

create_os

Usage

```
create_os <name>
```

Options

- <name>: Name of operating system (BSP type).

Description

This command creates a new operating system (OS) instance (also known as a BSP type) available for the Nios II BSP tools. This command must precede all others that describe the OS in its Tcl script. You can only have one `create_os` command in each Tcl script. If the `create_os` command appears in the Tcl script, the `create_driver` or `create_sw_package` commands cannot appear.

The name argument is usually distinct from all other operating systems that the SBT might locate. You can specify an OS name identical to OS if the OS you are describing has a unique version number assignment.

This command is required, unless you use the `create_driver` or `create_sw_package` commands, as appropriate.

create_sw_package

Usage

```
create_sw_package <name>
```

Options

- *<name>*: Name of the software package.

Description

This command creates a new software package instance available for the Nios II SBT. This command must precede all others that describe the software package in its Tcl script. You can only have one `create_sw_package` command in each Tcl script. If the `create_sw_package` command appears in the Tcl script, the `create_driver` or `create_os` commands cannot appear.

The name argument is usually distinct from all other device drivers and software packages that the SBT might locate. You can specify a name identical to another software package if the software package you are describing has a unique version number assignment.

If your software package differs for different operating systems, you need to provide a unique name for each BSP type.

This command is required, unless you use the `create_driver` or `create_os` commands, as appropriate.

set_sw_property

Usage

```
set_sw_property <property> <value>
```


Options

- *<property>*: Type of software property being set.
- *<value>*: Value assigned to the property.

Description

Sets the specified value to the specified property. The properties this command supports can only hold a single value. This command overwrites the existing (or default) contents of a particular property with the specified value. This command applies to device drivers and software packages.

This command supports the following properties:

- `hw_class_name`—The name of the hardware class which your device driver supports. The hardware class name is also the Component Name shown in SOPC Builder Component Editor. Example: `altera_avalon_uart`. This property is only available for device drivers.
 If your driver supports a user-defined component created with SOPC Builder 7.0 or earlier, you must append `_classic` to the class name. If you create (or update) your component with the SOPC Builder 7.1 (or later) component editor, there is no need to append `_classic`.
This property is required for all drivers.

- `version`—The version number of this package. `set_sw_property` uses version numbers to determine compatibility between hardware (peripherals) and their software (drivers), as well as to choose the most recent software or driver if multiple compatible versions are available. A version can be any alphanumeric string, but is usually a major and one or more minor revision integers. The dot (.) character separates major and minor revision numbers. Examples: 9.0, 5.0sp1, 3.2.11. This property is optional, but recommended. If you do not specify a version, the newest version of the package is used.
- `min_compatible_hw_version`—Specifies that the device driver supports the specified hardware version, or all greater versions. This property is only available for device drivers. If your device driver supports only one or more specific versions of a hardware class, use the `add_sw_property specific_compatible_hw_version` command instead. See the `version` property documentation for information about version strings. This property is optional. This property is only available for device drivers.
- `auto_initialize`—Boolean value that specifies `alt_sys_init.c` needs to initialize your package. If enabled, you must provide a header file containing `INSTANCE` and `INIT` macros per the instructions in the *Nios II Software Developer's Handbook*. This property is optional; if unspecified, `alt_sys_init.c` does not contain references to your driver or software. This property is only available for device drivers and software packages.
- `bsp_subdirectory`—Specifies the top-level directory where `nios2-bsp-generate-files` copies all source files for this package. This property is a path relative to the top-level BSP directory. This property is optional; if unspecified, `nios2-bsp-generate-files` copies the driver or software package into the `drivers` subdirectory of any BSP including this software.
- `alt_sys_init_priority`—This property assigns a priority to the software package or device driver. The value of this property must be a positive integer. Use this property to customize the order of macro calls in the BSP `alt_sys_init.c` file. Specifying the priority is useful if your software or driver must be initialized before or after other software in the system. For example, your driver might depend on another driver already being initialized. This property is optional. The default priority is 1000. This property is only available for device drivers and software packages.
- `display_name`—This property is used for user interfaces and other tools that wish to show a human-readable name to identify the software being described in the `.tcl` script. `display_name` is set to a few words of text (in quotes) that name your software. For example: Altera Nios II driver. This property is optional. If not set, tools that attempt to use the display name use the package name created with the appropriate `create_` command.
- `extends_bsp_type`—This property specifies which BSP type that an operating system (created with the `create_os` command) extends (if any). Currently, only the Altera HAL (HAL) is supported. This command is required for all operating systems that wish to use HAL-compatible generators in the Nios II BSP tools. It is also required for


operating systems that require the Altera HAL, device driver, or software package source files that are HAL compatible in BSPs created with that operating system. An operating system that extends HAL is presumed to be compatible with device drivers that support HAL.


This command is only available for operating systems.

- `callback_source_file`—This property specifies a Tcl source file containing callback functions.
- `initialization_callback`—This property specifies the name of a Tcl callback function which is intended to run in the following environment:
 - Run time: initialization
 - Scope: component instance
 - Function argument(s): component instance name
- `validation_callback`—This property specifies the name of a Tcl callback function which is intended to run in the following environment:
 - Run time: validation
 - Scope: component instance
 - Function argument(s): component instance name
- `generation_callback`—This property specifies the name of a callback function which is intended to run in the following environment:
 - Run time: generation
 - Scope: component instance
 - Function argument(s): component instance name, BSP generate target directory, driver BSP subdirectory
- `class_initialization_callback`—This property specifies the name of a callback function which is intended to run in the following environment:
 - Run time: initialization
 - Scope: component instance
 - Function argument(s): driver class name
- `class_validation_callback`—This property specifies the name of a callback function which is intended to run in the following environment:
 - Run time: validation
 - Scope: component instance
 - Function argument(s): driver class name
- `class_generation_callback`—This property specifies the name of a callback function which is intended to run in the following environment:
 - Run time: generation
 - Scope: component instance
 - Function argument(s): driver class name, BSP generate target directory, driver BSP subdirectory

- `supported_interrupt_apis`—Specifies the interrupt API that the device driver supports. Specify `legacy_interrupt_api` if the device driver supports the legacy API only or `enhanced_interrupt_api` if the device driver supports the enhanced API only. Specify both using a quoted list if the device driver supports both APIs.

If you do not specify which API your device driver supports, the Nios II SBT assumes that only the legacy interrupt API is supported. The Nios II SBT analyzes this property for each driver in the system to determine the appropriate API to be used in the system.

 This property is only available for device drivers.

 For more information about the legacy and enhanced APIs, refer to “Exception Handling” in the *Nios II Software Build Tools* chapter of the *Nios II Software Developer’s Handbook*.

- `isr_preemption_supported`—Specify true if your device driver ISR can be preempted by a higher priority ISR. If you do not specify whether ISR preemption is supported, the Nios II SBT assumes that your device driver does not support preemption. If your driver does not have an ISR, but the associated device has an interrupt port, you can set this property to true.

 This property is valid for operating systems and device drivers.

Path Names

There are some restrictions on how you can specify file paths when working with the Nios II SBT. The tools are designed for the maximum possible compatibility with a variety of computing environments. By following the restrictions in this section, you can ensure that the build tools work smoothly with other tools in your tool chain.

Command Arguments

Many Nios II software build tool commands take file name and directory path arguments. You can provide these arguments in any of several supported cross-platform formats. The Nios II SBT supports the following path name formats:

- **Quoted Windows**—A drive letter followed by a colon, followed by directory names delimited with backslashes, surrounded by double quotes. Example of a quoted Windows absolute path:

```
"c:\altera\72\nios2eds\examples\verilog\niosII_cyclone_1c20\standard"
```

Quoted Windows relative paths omit the drive letter, and begin with two periods followed by a backslash. Example:

```
"..\niosII_cyclone_1c20\standard"
```

- **Escaped Windows**—The same as quoted Windows, except that each backslash is replaced by a double backslash, and the double quotes are omitted. Examples:

```
c:\\altera\\72\\nios2eds\\examples\\verilog\\niosII_cyclone_1c20\\standard  
..\\niosII_cyclone_1c20\\standard
```

- **Linux**—An optional forward slash, followed by directory names delimited with forward slashes. Examples:

```
/altera/72/nios2eds/examples/verilog/niosII_cyclone_1c20/standard
verilog/niosII_cyclone_1c20/standard
```

Linux relative paths begin with two periods followed by a forward slash.

Example:

```
../niosII_cyclone_1c20/standard
```

- **Mixed**—The same as quoted Windows, except that each backslash is replaced by a forward slash, and the double quotes are omitted. Examples:

```
c:/altera/72/nios2eds/examples/verilog/niosII_cyclone_1c20/standard
../niosII_cyclone_1c20/standard
```

- **Cygwin**—An absolute Cygwin path consists of the pseudo-directory name "/cygdrive/", followed by the lower case Windows drive name, followed by directory names delimited with forward slashes. Example:

```
/cygdrive/c/altera/72/nios2eds/examples/verilog/niosII_cyclone_1c20/standard
```

Cygwin relative paths are the same as Linux relative paths. Example:

```
../niosII_cyclone_1c20/standard
```

The Nios II SBT accepts both relative and absolute path names.

Table 15-8 shows the supported path name formats for each platform, for Nios II SBT utilities and makefiles.

Table 15-8. Path Name Format Support

Context	Formats supported on Linux (1)	Formats supported on Windows with Cygwin
Utilities and scripts	Linux	<ul style="list-style-type: none"> ■ Quoted Windows (2) ■ Mixed (2) ■ Escaped Windows (2) ■ Cygwin
Makefiles	Linux	<ul style="list-style-type: none"> ■ Mixed (3) ■ Cygwin (3)
<p>Notes to Table 15-8:</p> <p>(1) These rules apply to any Unix-like platform.</p> <p>(2) These rules apply to other Unix-like shells running on Windows. The Nios II Command Shell, provided with the Nios II EDS, is based on Cygwin. Examples in this chapter are designed for the Nios II Command Shell.</p> <p>(3) The build tools automatically convert path names to Cygwin format</p>		

Object File Directory Tree


The makefile created by the Nios II SBT creates a new directory tree for generated object files. To the extent possible, the object file directory tree retains the structure of the corresponding source directory.

For example, if you specify the path to a source file as

```
src/util/special/tools.c
```

the makefile places the corresponding object code in

```
obj/util/special/tools.o
```

 The object file directory structure is illustrated in “Nios II Software Projects” in the *Nios II Software Build Tools* chapter of the *Nios II Software Developer’s Handbook*.

The makefile does not create object directories outside the project directory root. If the source file path you specify is a relative path beginning with “..”, the Nios II SBT flattens the path name prior to creating the object directory structure.

For example, if you specify the path to a source file as

```
../special/tools.c
```

the makefile places the corresponding object code in

```
obj/tools.o
```

If you specify an absolute path to source files under Cygwin, the Nios II SBT creates the obj directory structure as if you had used the Cygwin form of the path name. For example, if you specify the path to a source file as

```
c:/dev/app/special/tools.c
```

the Nios II SBT places the corresponding object code in

```
obj/cygdrive/c/dev/app/special/tools.o
```

Document Revision History

Table 15-9 shows the revision history for this document.

Table 15-9. Document Revision History (Part 1 of 2)

Date	Version	Changes
February 2011	10.1.0	<ul style="list-style-type: none"> ■ Correction to <code>add_memory_device</code> Tcl command arguments. ■ New functionality in <code>nios2-bsp-create-settings</code> command. ■ Removed “Referenced Documents” section.
July 2010	10.0.0	<ul style="list-style-type: none"> ■ Update documentation of <code>hal.enable_small_c_library</code> setting. ■ Describe new BSP Tcl commands: <ul style="list-style-type: none"> ■ <code>add_memory_device</code> ■ <code>set_ignore_file</code> ■ Correct missing properties in <code>set_sw_property</code> Tcl command: <ul style="list-style-type: none"> ■ <code>supported_interrupt_apis</code> ■ <code>isr_preemption_supported</code>

Table 15–9. Document Revision History (Part 2 of 2)


Date	Version	Changes
November 2009	9.1.0	<ul style="list-style-type: none"> ■ Support for external interrupt controller. ■ Add documentation for the following utilities: <ul style="list-style-type: none"> ■ nios2-lib-update-makefile ■ nios2-app-update-makefile ■ nios2-convert-ide2sbt ■ nios2-example-sw-create ■ nios2-elf-insert ■ nios2-elf-query ■ nios2-flash-programmer ■ Add documentation for <code>--jdi</code> command-line argument. ■ Add documentation for example design scripts. ■ Add documentation for hal.log_flags setting. ■ Add documentation for interrupt stack settings. ■ Clarify information about default settings for MicroC/OS-II. ■ Clarify information about default settings for host file system. ■ Add documentation for makefile variables. ■ Add documentation for the following BSP Tcl commands: <ul style="list-style-type: none"> ■ <code>add_systemh_line</code> ■ <code>get_assignment</code> ■ <code>get_cpu_name</code> ■ <code>get_interrupt_controller_id</code> ■ <code>get_irq_interrupt_controller_id</code> ■ <code>is_connected_interrupt_controller_device</code> ■ Describe Tcl callback functions.
March 2009	9.0.0	<ul style="list-style-type: none"> ■ Reorganized and updated information and terminology to clarify role of Nios II Software Build Tools. ■ Described usage of custom Tcl scripts. ■ Corrected minor typographical errors.
May 2008	8.0.0	<ul style="list-style-type: none"> ■ Advanced exceptions added to Nios II core. ■ Instruction-related exception handling added to HAL. ■ Describe new BSP setting <code>hal.enable_instruction_related_exceptions_api</code>.
October 2007	7.2.0	Initial release. Reference material moved here from former <i>Nios II Software Build Tools</i> chapter.

Introduction

This chapter familiarizes you with the main features of the Nios® II integrated development environment (IDE).


 In most cases, you should create new projects using either the Nios II Software Build Tools (SBT) for Eclipse™ or the SBT command line. IDE support is for the following situations:

- Working with pre-existing Nios II IDE software projects
- Creating new projects for the Nios II C2H compiler
- Debugging with the FS2 console

 If your hardware design was created with SOPC Builder 7.0 or earlier, you must either use the Nios II IDE development flow, or update your hardware design.

This chapter contains the following sections:

- “Differences from the Nios II Software Build Tools”
- “Getting Started with the Nios II IDE” on page A-4
- “Developing Software with the Nios II IDE” on page A-8
- “Porting Nios II IDE Projects to the SBT” on page A-18
- “Archiving Nios II IDE Software Projects” on page A-21
- “Help System” on page A-23

 For more information on all topics related to the Nios II IDE, refer to the Nios II IDE help system.

The Nios II IDE Tools

Table A-1 describes the tools provided by the Nios II IDE user interface.

Differences from the Nios II Software Build Tools

The Nios II Embedded Design Suite (EDS) offers two software development tool flows, as described in “Nios II Software Development Environment” in the *Overview* chapter of the *Nios II Software Developer’s Handbook*. The Nios II IDE is the key part of the Nios II IDE development flow. This section describes some importance differences between the SBT development flow and the Nios II IDE development flow.

Table A-1. The Nios II IDE and Associated Tools

Tools	Description
The Nios II IDE	The Nios II IDE is a software development user interface for the Nios II processor. All software development tasks can be accomplished in the IDE, including editing, building, and debugging programs. For more information, refer to the Nios II IDE help system.
Flash programmer	The Nios II IDE includes a flash programmer utility that allows you to program flash memory chips on a target board. The flash programmer supports programming flash on any board, including Altera® development boards and your own custom boards. The flash programmer facilitates programming flash for the following purposes: <ul style="list-style-type: none"> ■ Executable code and data ■ Bootstrap code to copy code from flash to RAM, and then run from RAM ■ Hardware Abstraction Layer (HAL) file subsystems ■ FPGA hardware configuration data For more information, refer to the <i>Nios II Flash Programmer User Guide</i> .
Instruction set simulator	Altera provides an instruction set simulator (ISS) for the Nios II processor. The ISS is available in the Nios II IDE, and the process for running and debugging programs on the ISS is the same as for running and debugging on target hardware. For more information, refer to the Nios II IDE help system.
Quartus® II Programmer	The Quartus II programmer is part of the Altera Complete Design Suite, however the Nios II IDE can start the Quartus II programmer directly. The Quartus II programmer allows you to download new FPGA configuration files to the board. For more information, refer to the Nios II IDE help system, or to the Quartus II help system.

Nios II IDE Makefiles

A major difference between the Nios II IDE software development flow and the Nios II SBT flow is the difference in makefile implementation. The Nios II SBT generates user-managed makefiles that you can read, and modify in detail using the SBT. In the Nios II IDE development flow, the IDE creates and manages your project makefiles for you.

The key differences between user-managed makefiles and Nios II IDE makefiles are as follows:

- The Nios II IDE has control over the contents of a makefile in an IDE project.
- In a Nios II IDE makefile, the structure and syntax are optimized for automation rather than for human readability.
- It is not normally necessary or recommended for you to read or modify a Nios II IDE makefile.

Nios II IDE Terminology

The Nios II SBT and the Nios II IDE are described with somewhat different project terminology. Where the meaning is unambiguous, this handbook uses the SBT terminology for both development flows. The IDE terminology is used where needed to distinguish the Nios II IDE development flow from the SBT development flow.

The terminology differences are listed in [Table A-2](#).

Table A-2. Nios II IDE Terminology

Nios II IDE Terminology	Nios II SBT Terminology
Nios II C/C++ application	Nios II application
Nios II C/C++ library	Nios II user library
System library	Board support package (BSP)
System library option	BSP setting
Software component	Software package

Altera Nios II Instruction Set Simulator


The Nios II Instruction Set Simulator (ISS) allows you to begin developing programs before the target hardware platform is ready. The Nios II IDE allows you to run programs on the ISS as easily as running on a real hardware target.

Command-Line Tools

Although the Nios II IDE is primarily a GUI, it includes some commands for use at the Nios II Command Shell. This section describes those commands.

Nios II IDE Command-Line Tools

[Table A-3](#) shows the command-line utilities that form the basis of the Nios II IDE. These tools can create and build Nios II IDE projects without launching the Nios II IDE GUI. However, Altera recommends that you use the Nios II SBT to address command-line needs, as well as for all new projects.

 For detailed information about the Nios II SBT, refer to the [Nios II Software Build Tools](#) chapter of the *Nios II Software Developer's Handbook*.

Each of the Nios II IDE command-line tools launches the Nios II IDE in the background, without displaying the GUI. You cannot use these utilities while the IDE is running, because only one instance of the Nios II IDE can be active at a time.


The Nios II IDE command-line tools are in the `<Nios II EDS install path>/bin/` directory.

Table A-3. Nios II IDE Command-Line Tools

Tool	Description
<code>nios2-create-system-library</code>	Creates a new system library project.
<code>nios2-create-application-project</code>	Creates a new C/C++ application project.

Table A-3. Nios II IDE Command-Line Tools

Tool	Description
nios2-build-project	Builds a project using the Nios II IDE managed-make facilities. Creates or updates the makefiles to build the project, and optionally runs make. nios2-build-project operates only on projects that exist in the current Nios II IDE workspace.
nios2-import-project	Imports a previously-created Nios II IDE project into the current workspace.
nios2-delete-project	Removes a project from the Nios II IDE workspace, and optionally deletes files from the file system.


 The Nios II IDE command-line tools must be supplied with a workspace location. This location is supplied by means of the `-data <path to workspace>` command-line argument. The path to the workspace must not contain whitespace. Otherwise, any valid disk location can be used for the workspace. The workspace shown in [Example A-1](#) is the default workspace which is used by the IDE.

Example A-1. Specifying a Workspace on the Command Line

```
nios2-create-project \
  -data c:/altera/80/nios2eds/bin/eclipse/nios2-ide-workspace-8.0 \
  <other arguments>
```

FS2 Command-Line Interface

The `nios2-console` command opens the FS2 command-line interface, connects to the Nios II processor, and optionally downloads code.

 The FS2 console is not compatible with the Nios II SBT for Eclipse.

GCC Toolchain

Starting in v. 10.0, the Nios II EDS includes two versions of the GNU Compiler Collection (GCC) toolchain: GCC 3.4.6 and GCC 4.1.2. The Nios II IDE uses GCC 3, and IDE projects can only be built with GCC 3.

Getting Started with the Nios II IDE

This section describes the key components of the Nios II IDE, and describes how to create and debug a software project.

The Nios II IDE Workbench

The term “workbench” refers to the desktop development environment for the Nios II IDE. The workbench is where you edit, compile and debug your programs in the IDE.

Perspectives, Editors, and Views

Each workbench window contains one or more perspectives. Each perspective provides a set of capabilities for accomplishing a specific type of task.

Most perspectives in the workbench comprise an editor area and one or more views. An editor allows you to open and edit a project resource (i.e., a file, folder, or project). Views support editors, provide alternative presentations, and ways to navigate the information in your workbench.

Any number of editors can be open at once, but only one can be active at a time. The main menu bar and toolbar for the workbench window contain operations that are applicable to the active editor. Tabs in the editor area indicate the names of resources that are currently open for editing. An asterisk (*) indicates that an editor has unsaved changes. Views can also provide their own menus and toolbars, which, if present, appear along the top edge of the view. To open the menu for a view, click the drop-down arrow icon at the right of the view's toolbar or right-click in the view. A view might appear on its own, or stacked with other views in a tabbed notebook.

EDS Development Flows and the Nios II IDE

The main distinction between the two development flows is in the management of the project.

Nios II IDE Projects and Makefiles

In the Nios II IDE development flow, the IDE manages Nios II C/C++ application and board support package (BSP) projects and makefiles that you create with the **New Project** wizard in Nios II IDE. The best way to modify and build an IDE project is through the IDE. You manage the BSP project settings with the **System Library** page of the **Properties** dialog box.



In the Nios II IDE, the term “system library” is used for a BSP project.

SBT Projects and Makefiles

In the Nios II SBT development flow, you manage Nios II application, user library, and BSP projects and makefiles, giving you total control. Typically, you create SBT projects outside of the Nios II IDE and then import them into the IDE for debugging.



SBT projects and Nios II IDE projects are not interchangeable. However, you can manually convert an IDE project to an SBT project.



For details, refer to [“Porting Nios II IDE Projects to the SBT” on page A-18](#).

Creating a New Nios II IDE-Managed Project

The Nios II IDE provides a **New Project** wizard that guides you through the steps to create new IDE projects. To start the **New Project** wizard for Nios II C/C++ application projects, on the File menu in the Nios II C/C++ perspective, point to **New**, and then click **Nios II C/C++ Application**.

The Nios II C/C++ application **New Project** wizard prompts you to specify:

1. A name for your new Nios II project.
2. The target hardware.
3. A template for the new project.

Project templates are ready-made, working software projects that serve as examples to show you how to structure your own Nios II projects. It is often easier to start with a working “Hello World” project, than to start a blank project from scratch.

When the Nios II IDE creates the new application project, it also creates a BSP project. If the name of the application project is *<name>*, the default name of the BSP project is *<name>*_syslib* (for example, **dhystone_0_syslib**). These projects appear in the Nios II C/C++ Projects view of the workbench.



The first time you create or build a Nios II project, the Nios II IDE automatically creates a project in your workspace called **altera.components**. This project contains links to the source code files for all Altera-provided device drivers and software packages, enabling you to step through system code in the debugger, set breakpoints, and use other debugger features. The **altera.components** project appears in the Nios II C/C++ Projects view. The Nios II C/C++ view protects the source files in **altera.components** from accidental deletion, because they are shared among all software projects. Do not attempt to circumvent this protection.

Building and Managing Projects

Right-clicking on any resource (a file, folder, or project) opens a context-sensitive menu containing commands that you can perform on the resource. Right-clicking is usually the quickest way to find the command you need, though commands are also available in menus and toolbars.

To compile a Nios II project, right-click the project in the Nios II C/C++ Projects view, and click **Build Project**. When building, the Nios II IDE first builds the BSP project (and any other project dependencies), and then compiles the main project. Any warnings or errors are displayed in the Tasks view.

Right-clicking a project in the Nios II C/C++ Projects view also allows you to access the following important options for managing the project:

- **Properties**—Manage the dependencies on target hardware and other projects
- **System Library Properties**—Manage hardware-specific settings, such as communication devices and memory partitioning
- **Build Project**—i.e., make
- **Run As**—Run the program on hardware or under simulation
- **Debug As**—Debug the program on hardware or under simulation

Debug and Release Configurations

You can select a Debug or Release configuration in the **Project Properties** dialog box, under **C/C++ Build**. The project configuration controls the optimization level and debug compiler options.

Running and Debugging Programs

Run and debug operations are available by right-clicking the Nios II project. The Nios II IDE allows you to run or debug the project either on a target board, under the Nios II ISS, or using the ModelSim[®] logic simulator. For example, to run the program on a target board, right-click the project in the Nios II C/C++ Projects view, point to Run As, and then click **Nios II Hardware**. Character I/O to stdout and stderr are displayed in the Console view.

Starting a debug session is similar to starting a run session. For example, to debug the program on the ISS, right-click the project in the Nios II C/C++ Projects view, point to **Debug As**, and then click **Nios II Instruction Set Simulator**.

Launching the debugger changes the workbench perspective to the debug perspective. You can easily switch between the debug perspective and the Nios II C/C++ development perspective by clicking on the **Open Perspective** icon at the upper right corner of the workbench window.

After you start a debug session, the debugger loads the program, sets a breakpoint at `main()`, and begins executing the program. You use the usual controls to step through the code: Step Into, Step Over, Resume, Terminate, etc. To set a breakpoint, double click in the left-hand margin of the code view, or right-click in the margin and then click **Add Breakpoint**.

The Nios II IDE offers many views that allow you to examine the status of the processor while debugging, such as the Variables, Expressions, Registers, and Memory views.

Programming Flash

Many Nios II processor systems use external flash memory to store one or more of the following items:

- Program code
- Program data
- FPGA configuration data
- File systems

The Nios II IDE provides a Flash Programmer utility to help you manage and program the contents of flash memory.



To program an SBT C/C++ application to flash memory, you must first specify an SOPC Builder System File, as follows:

1. Click **Browse** at the right of the **SOPC Builder System PTF File** box.
2. Locate the SOPC Builder System File on which the application's BSP is based. For example, if you are using a Nios II SBT example, the SOPC Builder System File is three levels up in the directory tree from the software project.

Developing Software with the Nios II IDE

In many ways, Nios II software development with the Nios II IDE is the same as development with the SBT. The processor architecture, Hardware Abstraction Layer (HAL), software packages and drivers are the same. However, there are a few limitations and differences in tool flow details. This section discusses those differences and limitations.

Using the HAL in an IDE Project

Like the Nios II SBT development flow, the Nios II IDE flow can automatically keep your system library up to date with the SOPC Builder system. In an IDE project, the Nios II IDE manages the system library and updates the driver configurations to accurately reflect the system hardware. If the SOPC Builder system changes — i.e., the SOPC Builder system file (.ptf) is updated — the IDE rebuilds the system library the next time you build or run your C/C++ application program.

Generated Files

The Nios II IDE development flow uses different file name and directory structure conventions for some generated system library files, as described in this section.

generated.x

In a Nios II IDE project, the **generated.x** file is the same as the **linker.x** file created by the Nios II build tools.

generated.gdb

In a Nios II IDE project, the **generated.gdb** file is the same as the **memory.gdb** file created by the Nios II build tools.

alt_main.c

In a Nios II IDE project, you can find **alt_main.c** in
<Nios II EDS install path>/components/altera_hal/HAL/src.

System Library Settings

In a Nios II IDE project, you manage the system library project settings with the **System Library** page of the **Properties** dialog box.



For details of how to control system library settings, refer to the Nios II IDE help system.

Reducing Code Footprint

The basic techniques for reducing code footprint are the same in the Nios II IDE flow as in the SBT flow, but you use a different procedure to specify the system library options. You control the following system library options through the Nios II IDE system properties dialog box:

Table A-4. System Library Options for Reducing Code Footprint

Technique	System Library Option Name
Use Reduced Device Drivers	Reduced device drivers
Reduce the File Descriptor Pool	Max file descriptors
Use a Smaller File I/O Library	Small C library
Use the Lightweight Device Driver application programming interface (API)	Lightweight device driver API
Eliminate Clean Exit	Clean exit (flush buffers)
Eliminate All Exit Code	Program never exits
Turn off C++ Support	Support C++

Paths to Hardware Abstraction Layer Files

In Nios II IDE projects, HAL source files are in several directories. You can find HAL-related files in the following locations:

- The `<Nios II EDS install path>/components` directory contains most HAL source files.
- `<Nios II EDS install path>/components/altera_hal/HAL/inc/sys` contains header files defining the HAL generic device models. In a `#include` directive, reference these files relative to `<Nios II EDS install path>/components/altera_hal/HAL/inc/`. For example, to include the direct memory access (DMA) drivers, use `#include sys/alt_dma.h`
- `<Nios II EDS install path>/bin` contains the newlib ANSI C library header files.
- The Altera Complete Design Suite includes HAL drivers for SOPC Builder components distributed with the Altera Complete Design Suite. For example, if the design suite is installed in `c:\altera\80`, you can find the drivers under `c:\altera\80\ip\sopc_builder_ip`.

Overriding HAL Functions

In the Nios II IDE build flow, you can override any HAL source file, including `alt_sys_init.c`, by placing your own implementation in your system project directory. When building the executable, the Nios II IDE finds your function, and uses it in place of the HAL version.

Device Drivers for Nios II IDE Projects

HAL device drivers work the same in the Nios II IDE flow as in the SBT flow. However, there are slight differences in how you create a device driver.

Compared with the Nios II IDE, Nios II SBT provides a less rigid set of file naming and location requirements for your drivers. However, Altera recommends using the Nios II IDE conventions to maintain build-flow compatibility. Provided you use the file hierarchy described in “Integrating a Device Driver in the HAL” in the *Developing Device Drivers for the Hardware Abstraction Layer* chapter of the *Nios II Software Developer’s Handbook*, your device driver is compatible with the Nios II IDE development flow.

This section describes how to develop device drivers for Nios II IDE projects.

Integrating a Device Driver in the HAL

This section discusses how to take advantage of the HAL’s ability to instantiate and register device drivers during system initialization. You can take advantage of this service, whether you created a device driver for one of the HAL generic device models, or you created a peripheral-specific device driver. Taking advantage of the automation provided by the HAL is mainly a process of placing files in the appropriate place in the HAL directory structure.

Device Driver Files for the HAL

This section describes how to provide appropriate files to integrate your device driver into the HAL.

- **A Device’s HAL Header File and `alt_sys_init.c`**—At the heart of the HAL is the autogenerated source file, `alt_sys_init.c`. `alt_sys_init.c` contains the source code that the HAL uses to initialize the device drivers for all supported devices in the system. In particular, this file defines the `alt_sys_init()` function, which is called before `main()` to initialize all devices and make them available to the program.


 Refer to “Creating a Custom Device Driver for the HAL” in the *Developing Device Drivers for the Hardware Abstraction Layer* chapter of the *Nios II Software Developer’s Handbook* for more information about `alt_sys_init.c`.

- **A Device’s HAL Header File and `alt_sys_init.c`**—In the Nios II IDE development flow, for each device visible to the processor, the generator utility searches for an associated header file in the device’s HAL/inc directory. The name of the header file depends on the SOPC Builder component name. For example, for Altera’s JTAG UART component, the generator finds the file `altera_avalon_jtag_uart/HAL/inc/altera_avalon_jtag_uart.h`. If the generator utility finds such a header file, it inserts code into `alt_sys_init.c` to perform the following actions:
 - Include the device’s header file.
 - Call the macro `<name of device>_INSTANCE` to allocate storage for the device.
 - Call the macro `<name of device>_INIT` inside the `alt_sys_init()` function to initialize the device.

- **Device Driver Source Code**—Place any required source code in the `HAL/src` directory. In addition, you must include a makefile fragment, `component.mk`. The `component.mk` file lists the source files to include in the system library. You can list multiple files by separating filenames with a space. [Example A-2](#) shows an example makefile fragment for Altera’s JTAG UART device.

The Nios II IDE includes the `component.mk` file into the top-level makefile when compiling system library projects and application projects. `component.mk` can only modify the make variables listed in [Table A-5](#).

`component.mk` can add additional make rules and macros as required, but interoperability macro names must conform to the namespace rules.

 For details about namespace rules, refer to “Namespace Allocation” in the *Developing Device Drivers for the Hardware Abstraction Layer* chapter of the *Nios II Software Developer’s Handbook*.

Example A-2. component.mk for a UART Driver

```
C_LIB_SRCS += altera_avalon_uart.c
ASM_LIB_SRCS +=
INCLUDE_PATH +=
```

Table A-5. Make Variables Defined in component.mk

Make Variable	Meaning
C_LIB_SRCS	The list of C source files to build into the system library.
ASM_LIB_SRCS	The list of assembler source files to build into the system library (these are preprocessed with the C preprocessor).
INCLUDE_PATH	A list of directories to add to the include search path. The directory <code><component>/HAL/inc</code> is added automatically and so does not need to be explicitly defined by the component.

Overriding the Default Device Drivers

The Nios II IDE locates all include and source files using search paths. The system library project directory is always searched first. If you place an alternative driver in the system library project directory, it overrides drivers installed with the Nios II EDS. For example, if a component provides the header file `alt_my_component.h`, and the system library project directory also contains a file `alt_my_component.h`, the version provided in the system library project directory is used at compile time. This same mechanism can override C and assembler source files.


Exception Handling in a Nios II IDE Project


Exception handling with the internal interrupt controller in Nios II IDE projects is largely the same as in SBT projects. This section discusses the differences.

 The Nios II IDE development flow does not support external interrupt controllers.

Software Trap Handling

If your software is compiled for release, the exception handler makes a distinction between traps and other exceptions. If your software is compiled for debug, traps and other exceptions are handled identically, by executing a break instruction.

 For more information about HAL software exception handling, refer to “HAL Exception Handling System Implementation” in the *Exception Handling* chapter of the *Nios II Software Developer’s Handbook*.


 The instruction-related exception handler is unavailable in Nios II IDE projects.


Advanced Exceptions

Advanced exception support, including the instruction-related exception handler, is not available in the Nios II IDE development flow.

Using the Unimplemented Instruction Handler

To include the unimplemented instruction handler in a Nios II IDE project, turn on **Emulate multiply and divide instructions** on the **System properties** page of the Nios II IDE.

 You do not normally need the unimplemented instruction handler, because the HAL includes software emulation for unimplemented instructions from its run-time libraries if you are compiling for a Nios II processor that does not support the instructions.

 For further information about the unimplemented instruction handler, refer to “HAL Exception Handling System Implementation” in the *Exception Handling* chapter of the *Nios II Software Developer’s Handbook*.

Configuring MicroC/OS-II Projects with the Nios II IDE

In the Nios II IDE, the displayed MicroC/OS-II setting names are different from the equivalent BSP setting names. This section lists the meanings of the IDE setting names.

For step-by-step instructions on how to create a MicroC/OS-II project in the Nios II IDE, refer to *Using the MicroC/OS-II RTOS with the Nios II Processor Tutorial*.

MicroC/OS-II General Options

Table A-6 shows the general MicroC/OS-II options available through the Nios II IDE.

Table A-6. General Options (Part 1 of 2)

Option	Description
Maximum number of tasks	Specifies the value of the <code>OS_MAX_TASKS</code> preprocessor symbol. Must be at least 2
Lowest assignable priority	Specifies the value of the <code>OS_LOWEST_Prio</code> preprocessor symbol. Maximum allowable value is 63.
Thread safe C library	Enable thread-safe C library

Table A-6. General Options (Part 2 of 2)

Option	Description
Enable code for event flags	Specifies the value of the <code>OS_FLAG_EN</code> preprocessor symbol. When this option is disabled (set to 0), event flag settings are also disabled. Refer to “ Event Flag Settings ” on page A-13.
Enable code for mutex semaphores	Specifies the value of the <code>OS_MUTEX_EN</code> preprocessor symbol. When this option is disabled (set to 0), mutual exclusion semaphore settings are also disabled. Refer to “ Mutex Settings ” on page A-14
Enable code for semaphores	Specifies the value of the <code>OS_SEM_EN</code> preprocessor symbol. When this option is disabled (set to 0), semaphore settings are also disabled. Refer to “ Semaphore Settings ” on page A-14.
Enable code for mailboxes	Specifies the value of the <code>OS_MBOX_EN</code> preprocessor symbol. When this option is disabled (set to 0), mailbox settings are also disabled. Refer to “ Mailbox Settings ” on page A-14.
Enable code for queues	Specifies the value of the <code>OS_Q_EN</code> preprocessor symbol. When this option is disabled (set to 0), queue settings are also disabled. Refer to “ Queue Settings ” on page A-15.
Enable code for memory management	Specifies the value of the <code>OS_MEM_EN</code> preprocessor symbol. When this option is disabled (set to 0), memory management settings are also disabled. Refer to “ Memory Management Settings ” on page A-15.
Enable code for timers	Enable code for timers

Event Flag Settings

Table A-7 shows the event flag settings available through the Nios II IDE.

Table A-7. Event Flags Settings

Setting	Description
Include code for wait on clear in the event flags	Specifies the value of the <code>OS_FLAG_WAIT_CLR_EN</code> preprocessor symbol. This setting includes code to wait for the specified bits to be cleared in the event flag group.
Include code for OSFlagAccept()	Specifies the value of the <code>OS_FLAG_ACCEPT_EN</code> preprocessor symbol.
Include code for OSFlagDel()	Specifies the value of the <code>OS_FLAG_DEL_EN</code> preprocessor symbol.
Include code for OSFlagQuery()	Specifies the value of the <code>OS_FLAG_QUERY_EN</code> preprocessor symbol.
Maximum number of event flag groups	Specifies the value of the <code>OS_MAX_FLAGS</code> preprocessor symbol.
Size of name of event flags group	Specifies the value of the <code>OS_FLAG_NAME_SIZE</code> preprocessor symbol.
Event flag bits (8, 16, 32)	Specifies the number of event flag bits

Mutex Settings

Table A-8 shows the mutex settings available through the Nios II IDE.

Table A-8. Mutex Settings

Setting	Description
Include code for OSMutexAccept()	Specifies the value of the <code>OS_MUTEX_ACCEPT_EN</code> preprocessor symbol.
Include code for OSMutexDel()	Specifies the value of the <code>OS_MUTEX_DEL_EN</code> preprocessor symbol.
Include code for OSMutexQuery()	Specifies the value of the <code>OS_MUTEX_QUERY_EN</code> preprocessor symbol.

Semaphore Settings

Table A-9 shows the semaphore settings available through the Nios II IDE.

Table A-9. Semaphores Settings

Setting	Description
Include code for OSSemAccept()	Specifies the value of the <code>OS_SEM_ACCEPT_EN</code> preprocessor symbol.
Include code for OSSemSet()	Specifies the value of the <code>OS_SEM_SET_EN</code> preprocessor symbol.
Include code for OSSemDel()	Specifies the value of the <code>OS_SEM_DEL_EN</code> preprocessor symbol.
Include code for OSSemQuery()	Specifies the value of the <code>OS_SEM_QUERY_EN</code> preprocessor symbol.

Mailbox Settings

Table A-10 shows the mailbox settings available through the Nios II IDE.

Table A-10. Mailboxes Settings

Setting	Description
Include code for OSMboxAccept()	Specifies the value of the <code>OS_MBOX_ACCEPT_EN</code> preprocessor symbol.
Include code for OSMBoxDel()	Specifies the value of the <code>OS_MBOX_DEL_EN</code> preprocessor symbol.
Include code for OSMboxPost()	Specifies the value of the <code>OS_MBOX_POST_EN</code> preprocessor symbol.
Include code for OSMboxPostOpt()	Specifies the value of the <code>OS_MBOX_POST_OPT_EN</code> preprocessor symbol.
Include code for OSMBoxQuery()	Specifies the value of the <code>OS_MBOX_QUERY_EN</code> preprocessor symbol.

Queue Settings

Table A-11 shows the queue settings available through the Nios II IDE.

Table A-11. Queues Settings

Setting	Description
Include code for OSQAccept()	Specifies the value of the OS_Q_ACCEPT_EN preprocessor symbol.
Include code for OSQDel()	Specifies the value of the OS_Q_DEL_EN preprocessor symbol.
Include code for OSQFlush()	Specifies the value of the OS_Q_FLUSH_EN preprocessor symbol.
Include code for OSQPost()	Specifies the value of the OS_Q_POST_EN preprocessor symbol.
Include code for OSQPostFront()	Specifies the value of the OS_Q_POST_FRONT_EN preprocessor symbol.
Include code for OSQPostOpt()	Specifies the value of the OS_Q_POST_OPT_EN preprocessor symbol.
Include code for OSQQuery()	Specifies the value of the OS_Q_QUERY_EN preprocessor symbol.
Maximum number of Queue Control blocks	Specifies the value of the OS_MAX_QS preprocessor symbol.

Memory Management Settings

Table A-12 shows the memory management settings available through the Nios II IDE.

Table A-12. Memory Management Settings

Setting	Description
Include code for OSMemQuery()	Specifies the value of the OS_MEM_QUERY_EN preprocessor symbol.
Maximum number of memory partitions	Specifies the value of the OS_MAX_MEM_PART preprocessor symbol.
Size of memory partition name	Specifies the value of the OS_MEM_NAME_SIZE preprocessor symbol.

Miscellaneous Settings

Table A-13 shows the miscellaneous settings available through the Nios II IDE.

Table A-13. Miscellaneous Settings (Part 1 of 2)

Setting	Description
Enable argument checking	Specifies the value of the OS_ARG_CHK_EN preprocessor symbol.
Enable uCOS-II hooks	Specifies the value of the OS_CPU_HOOKS_EN preprocessor symbol.
Enable debug variables	Specifies the value of the OS_DEBUG_EN preprocessor symbol.
Include code for OSSchedLock() and OSSchedUnlock()	Specifies the value of the OS_SCHED_LOCK_EN preprocessor symbol.

Table A-13. Miscellaneous Settings (Part 2 of 2)

Setting	Description
Enable tick stepping feature for uCOS-View	Specifies the value of the OS_TICK_STEP_EN preprocessor symbol.
Enable statistics task	Specifies the value of the OS_TASK_STAT_EN preprocessor symbol.
Check task stacks from statistics task	Specifies the value of the OS_TASK_STAT_STK_CHK_EN preprocessor symbol.
Statistics task stack size	Specifies the value of the OS_TASK_STAT_STK_SIZE preprocessor symbol.
Idle task stack size	Specifies the value of the OS_TASK_IDLE_STK_SIZE preprocessor symbol.
Maximum number of event control blocks	Specifies the value of the OS_MAX_EVENTS preprocessor symbol.
Size of semaphore, mutex, mailbox, or queue name	Specifies the value of the OS_EVENT_NAME_SIZE preprocessor symbol.

Task Management Settings

Table A-14 shows the task management settings available through the Nios II IDE.

Table A-14. Task Management Settings

Setting	Description
Include code for OSTaskChangePrio()	Specifies the value of the OS_TASK_CHANGE_PRIO_EN preprocessor symbol.
Include code for OSTaskCreate()	Specifies the value of the OS_TASK_CREATE_EN preprocessor symbol.
Include code for OSTaskCreateExt()	Specifies the value of the OS_TASK_CREATE_EXT_EN preprocessor symbol.
Include code for OSTaskDel()	Specifies the value of the OS_TASK_DEL_EN preprocessor symbol.
Include variables in OS_TCB for profiling	Specifies the value of the OS_TASK_PROFILE_EN preprocessor symbol.
Include code for OSTaskQuery()	Specifies the value of the OS_TASK_QUERY_EN preprocessor symbol.
Include code for OSTaskSuspend() and OSTaskResume()	Specifies the value of the OS_TASK_SUSPEND_EN preprocessor symbol.
Include code for OSTaskSwHook()	Specifies the value of the OS_TASK_SW_HOOK_EN preprocessor symbol.
Size of task name	Specifies the value of the OS_TASK_NAME_SIZE preprocessor symbol.

Time Management Settings

Table A-15 shows the time management settings available through the Nios II IDE.

Table A-15. Time Management Settings

Setting	Description
Include code for OSTimeDlyHMSM()	Specifies the value of the <code>OS_TIME_DLY_HMSM_EN</code> preprocessor symbol.
Include code for OSTimeDlyResume()	Specifies the value of the <code>OS_TIME_DLY_RESUME_EN</code> preprocessor symbol.
Include code for OSTimeGet() and OSTimeSet()	Specifies the value of the <code>OS_TIME_GET_SET_EN</code> preprocessor symbol.
Include code for OSTimeTickHook()	Specifies the value of the <code>OS_TIME_TICK_HOOK_EN</code> preprocessor symbol.

Timer Management Settings

Table A-16 shows the timer management settings available through the Nios II IDE.

Table A-16. Timer Management Settings

Setting	Description
Maximum number of timers	Specifies the maximum number of timers
Determine the size of a timer name	Specifies the size of a timer name
Size of timer wheel (#Spokes)	Specifies the size of the timer wheel
Rate at which timer management task runs (Hz)	Specifies the rate at which the timer management task runs
Stack size for timer task	Specifies the stack space allocated for the timer task
Priority of timer task (0=highest)	Specifies the timer task priority

Using NicheStack in a Nios II IDE Project

This section discusses the details of how to use the NicheStack TCP/IP Stack in the Nios II IDE.

get_mac_addr() and get_ip_addr()

The NicheStack TCP/IP Stack system code calls `get_mac_addr()` and `get_ip_addr()` during the device initialization process. These functions are necessary for the system code to set the media access control (MAC) and IP addresses for the network interface, which you select through **MAC interface** in the **NicheStack TCP/IP Stack** tab of the **Software Components** dialog box.

`INICHE_DEFAULT_IF`, defined in `system.h`, identifies the network interface that you defined in SOPC Builder. In the Nios II IDE, you can set `INICHE_DEFAULT_IF` through the **MAC interface** control in the **NicheStack TCP/IP Stack** tab of the **Software Components** dialog box.

`DHCP_CLIENT`, also defined in `system.h`, specifies whether to use the dynamic host configuration protocol (DHCP) client application to obtain an IP address. You can set or clear this setting in the Nios II IDE (with the **Use DHCP to automatically assign IP address** check box)

Configuring the NicheStack TCP/IP Stack in the Nios II IDE

The Nios II IDE allows you to configure certain options (i.e. modify the `#defines` in `system.h`) without editing source code. The most commonly accessed options are available through the **NicheStack TCP/IP Stack** tab of the **Software Components** dialog box.



If you modify the `ippport.h` file directly, be careful not to select the **Clean Project** build option in the Nios II IDE. Selecting **Clean Project** results in your modified `ippport.h` file being replaced with the starting template version of this file.

Porting Nios II IDE Projects to the SBT

The Nios II SBT uses a different directory structure and settings file format than the Nios II IDE. Therefore, if you wish to take advantage of the Nios II SBT, you need to port your IDE projects to the Nios II SBT development flow.

This appendix describes the steps required to port a Nios II IDE project to the Nios II SBT development flow. The Nios II EDS includes a utility to convert Nios II IDE projects to the SBT flow.



You do not need to rewrite your Nios II IDE project's C/C++ code for use with the SBT development flow.


The Nios II SBT development flow provides a number of advantages over the Nios II IDE development flow. You might want to port an IDE project to the SBT to take advantage of the following improvements:

- Fully repeatable control over all build options using command line options, Tcl scripts, or both
- Simplified project file management and naming
- Simplified makefiles
- Versioned device drivers
- Independence from Eclipse code and Eclipse projects
- Self-contained BSPs, making hand-off and version control easier than is possible with Nios II IDE-created BSPs (system library projects)
- Upwards compatibility with future releases of the Nios II EDS
- GCC toolchain upgraded to version 4.1.2


Converting a Nios II IDE Project

This section describes how to convert a Nios II IDE project to an SBT project using the `nios2-convert-ide2sbt` utility.

1. Build the original project in the Nios II IDE, using either the Debug or the Release configuration, depending on your preference. Ensure that the project builds without errors.

 The Nios II SBT flow does not include separate Debug and Release builds as implemented in the Nios II IDE development flow.

2. Launch the Nios II Command Shell.

 For details about the Nios II Command Shell, refer to the *Getting Started from the Command Line* chapter of the *Nios II Software Developer's Handbook*.

3. Run **nios2-convert-ide2sbt**. The command syntax is as follows:

```
nios2-convert-ide2sbt --input-dir=<source directory> \  
  --output-dir=<target directory> \  
  --build-config=<configuration>
```

The command arguments are as follows:

- *<source directory>*—directory containing the original Nios II IDE project.
- *<target directory>*—directory where **nios2-convert-ide2sbt** places the converted SBT project. If *<target directory>* does not exist, **nios2-convert-ide2sbt** creates it.
- *<configuration>*—Debug or Release, designating the Nios II IDE project configuration.


Nios II IDE project types are converted as shown in [Table A-17](#).

Table A-17. Conversion of Project Types by nios2-convert-ide2sbt


Nios II IDE Project Type	SBT Conversion
Nios II C/C++ system library	BSP
Nios II C/C++ library	User library (1)
Nios II C/C++ application	Application project (1)
Note to Table A-17:	
(1) At the same time, nios2-convert-ide2sbt converts any Nios II C/C++ system library or Nios II C/C++ libraries on which the converted project depends.	

For example, suppose you have a Nios II C/C++ application in the Release configuration, located in the `./software/hello_world` directory. To convert the project and its associated system library, and put the resulting SBT project in the `./software_sbt` directory, type:

```
nios2-convert-ide2sbt --input-dir=software/hello_world \  
  --output-dir=software_sbt --build-config=Release ↵
```

 If you need to reconvert a project you previously converted, you must delete the previous target directory, or specify a new target directory.

nios2-convert-ide2sbt converts your Nios II IDE software project to equivalent SBT projects. **nios2-convert-ide2sbt** ports compiler flags, like the optimization level, debug, and custom instruction options, to the new project. During conversion, **nios2-convert-ide2sbt** displays the converted project settings on the console, in the form of a sample SBT command line.

 For details about SBT command usage, refer to the *Nios II Software Build Tools* and *Nios II Software Build Tools Reference* chapters of the *Nios II Software Developer's Handbook*.


Other Software Modules

This section describes how to convert and incorporate the following kinds of software modules that you might need to include in your converted BSP:

- Custom device drivers and software components
- Precompiled libraries and non-HAL device drivers

Custom Device Drivers and Software Components

In the Nios II IDE development flow, a makefile fragment named **component.mk** specifies device drivers. By contrast, in the Nios II SBT development flow, a Tcl script defines the device driver structure. If you have custom device drivers and software components, including third-party device drivers, convert them to Tcl scripts manually.


 For more information about implementing device drivers and software packages for the Nios II SBT, refer to “Device Drivers and Software Packages” in the *Nios II Software Build Tools* chapter of the *Nios II Software Developer's Handbook*.

Precompiled Libraries and Non-HAL Device Drivers

If you have precompiled libraries and non-HAL device drivers, including third-party libraries and device drivers, convert them manually.

The best way to convert a typical precompiled library is to create a software package. If the precompiled library is dependent on a specific hardware device, it is better to incorporate the library into the device driver. Library archive files (**.a**) can be incorporated into a device driver just as C source files (**.c**) and header files (**.h**) are.

 Non-HAL device drivers do not support initialization through `alt_sys_init()`.

 For information about creating software packages and drivers, refer to “Integrating a Device Driver in the HAL” in the *Developing Device Drivers for the Hardware Abstraction Layer* chapter of the *Nios II Software Developer's Handbook*.

nios2-convert-ide2sbt does not convert GNU Compiler Collection (GCC) command-line options for external include file paths (**-I**) or external library file paths (**-L, -l**). You must handle these cases manually.

To add include paths to a BSP, use the `hal.make.bsp_inc_dirs` BSP setting in your Tcl script.

To add libraries to an application, add or modify one of the following variables in the application makefile:


- `APP_LIBRARY_DIRS`—Specifies a list of paths to directories in which `lib<libname>.a` files reside.
- `APP_LIBRARY_NAMES`—Specifies a list of the names of the libraries being added. If the library file is named `lib<libname>.a`, specify `<libname>` as the name of the library.


Using Your Converted Project

When you have finished porting your project, you can manage the makefiles, build and run the project, and perform all other project tasks exactly as if the project were created with the SBT. You can also import the project to the SBT for Eclipse for debugging.

The Nios II EDS includes two versions of the GCC toolchain: GCC 3.4.6 and GCC 4.1.2. GCC 4, introduced with the Nios II EDS v. 10.0, is fully backwards-compatible with GCC 3, and provides substantially faster build times.

After conversion, your project is configured to use GCC 3 by default. To upgrade to GCC 4, import the project to the Nios II SBT for Eclipse and change the toolchain in the **Properties** dialog box, or simply build the project in the GCC 4 Nios II Command Shell.

 For information about managing GCC toolchains in the SBT for Eclipse, refer to “Managing Toolchains in Eclipse” in the *Getting Started with the Graphical User Interface* chapter of the *Nios II Software Developer’s Handbook*. For information about selecting the toolchain on the command line, refer to the *Getting Started from the Command Line* chapter of the *Nios II Software Developer’s Handbook*.

 Nios II IDE projects cannot be directly imported to the SBT for Eclipse. You must first convert the project to the SBT, according to the procedures in this section.

Archiving Nios II IDE Software Projects

This section helps you identify the files you must include when archiving a Nios II IDE software project. With this information, you can archive a Nios II application project and its associated Nios II BSP project.

You might want to archive your projects for one of the following reasons:

- To place them under source control
- To create backups
- To bundle the projects for transfer to another location

This section covers the following information:

- How to find and identify the files that you must include in an archived Nios II IDE software project.
- Which files must have write permission to allow the software projects to be built.

Required Files

This section describes the files required by Nios II IDE software projects. This is the minimum set of files needed to completely rebuild a software project, including the Executable and Linking Format File (.elf).

Archive your Nios II IDE software projects together with the SOPC Builder system on which they are based. You cannot rebuild a Nios II IDE software project without its associated SOPC Builder system.

Nios II Application Project Files

The files listed in [Table A-18](#) are located in the Nios II application project directory.

Table A-18. Files Required for a Nios II Application Project

File Description	File Name	Write Permission Required? ⁽¹⁾
All source files	for example: <code>app.c</code> , <code>header.h</code> , <code>assembly.s</code> , <code>lookuptable.dat</code>	No
Eclipse project file	<code>.project</code>	No
C/C++ Development Toolkit project file	<code>.cdtproject</code>	Yes
C/C++ Development Toolkit option file	<code>.cdtbuild</code>	No
Software configuration file	<code>application.stf</code>	No

Note to [Table A-18](#):

(1) For further information about write permissions, refer to “[File Write Permissions](#)”.

Nios II BSP Project

The files listed in [Table A-19](#) are located in the Nios II BSP (system library) project directory.

Table A-19. Files Required for a Nios II BSP Project

File description	File name	Write permission required? ⁽¹⁾
Eclipse project file	<code>.project</code>	Yes
C/C++ Development Toolkit project file	<code>.cdtproject</code>	Yes
C/C++ Development Toolkit option file	<code>.cdtbuild</code>	No
System software configuration file	<code>system.stf</code>	Yes

Note to [Table A-19](#):

(1) For further information about write permissions, see “[File Write Permissions](#)”.

File Write Permissions

You must have write permission for certain files, shown in [Table A-18](#) and [Table A-19](#). The tools write to these files as part of the build process. If the files are not writable, the tool chain fails.

Many source control tools mark local files read-only by default. In this case, you must override this behavior. You do not have to check the files out of source control unless you are modifying the Nios II software project.

Help System

The Nios II IDE help system provides documentation on all IDE topics. To launch the help system, click **Help Contents** on the Help menu. You can also press F1 on Windows (Shift-F1 on Linux) at any time for context-sensitive help. The Nios II IDE help system contains hands-on tutorials that guide you step-by-step through the process of creating, building, and debugging Nios II projects.

Document Revision History

Table A-20 shows the revision history for this document.

Table A-20. Document Revision History (Part 1 of 2)

Date	Version	Changes
February 2011	10.1.0	Removed “Referenced Documents” section.
July 2010	10.0.0	<ul style="list-style-type: none"> ■ Introduction of GCC 4 toolchain for Nios II Software Build Tools. ■ Nios II IDE projects limited to GCC 3. ■ Upgrading converted projects to GCC 4.
November 2009	9.1.0	<ul style="list-style-type: none"> ■ Nios II Software Build Tools for Eclipse introduced. ■ <i>Nios II Integrated Development Environment</i> chapter combined with <i>Appendix A, Porting Nios II IDE Projects to the Software Build Tools</i>. Chapter replaced by <i>Getting Started with the Graphical User Interface</i> chapter). ■ Include Nios II IDE-specific content from the following chapters: <ul style="list-style-type: none"> ■ Overview ■ Developing Programs Using the Hardware Abstraction Layer ■ Developing Device Drivers for the Hardware Abstraction Layer ■ Exception Handling ■ MicroC/OS-II Real-Time Operating System ■ Ethernet and the NicheStack TCP/IP Stack - Nios II Edition
March 2009	9.0.0	<ul style="list-style-type: none"> ■ Reorganized and updated information and terminology to clarify role of Nios II Software Build Tools. ■ Corrected minor typographical errors.
May 2008	8.0.0	Maintenance release.
October 2007	7.2.0	altera.components project added.
May 2007	7.1.0	<ul style="list-style-type: none"> ■ Nios II Software Build Tools introduced. ■ Added instructions for importing Software Build Tools projects. ■ Changed chapter title. ■ Added table of contents to “Introduction” section. ■ Added Referenced Documents section.
March 2007	7.0.0	Maintenance release.

Table A-20. Document Revision History (Part 2 of 2)

Date	Version	Changes
November 2006	6.1.0	Updated look and feel based on Eclipse 3.2, including Nios II C/C++ perspective and Nios II C/C++ Projects views, renamed project types.
May 2006	6.0.0	Maintenance release.
October 2005	5.1.0	Updated for the Nios II IDE version 5.1.
May 2005	5.0.0	Maintenance release.
September 2004	1.1	Updated screen shots.
May 2004	1.0	Initial release.