1 Application Programming Interface

In this appendix we will describe the application programming interface that programmers should use to write applications on systems using the xTask Operating System.

Applications that make use of the xTask API should include the following header file: `/include/xtask.h`.

1.1 xtask_kernel

void xtask_kernel(init_tasks, idle_task, tick_rate, notification_chan, service_chan)

Initialize and start a kernel. The kernel is connected to a Communication Server through the service and notification channels. This function should be called in a `par` statement from the main function (using a C wrapper function to circumvent xC's dislike of function pointers).

Arguments:

init_tasks	Pointer to function that initializes all initial tasks. This function has the signature: `void function(void)`.
idle_task	Pointer to idle task function. This function has the signature: `void function(void *)`.
unsigned int tick_rate	kernel tick rate (by default in 10ns resolution).
chanend notification_chan	channel for notifications from Communication Server.
chanend service_chan	channel to get service from Communication Server.

Return value:

This function never returns, it will switch to the first task to be ran.

1.2 xtask_comserver

void xtask_comserver(service_chan[], noficication_chan[], nr_kernels, ring_in, ring_out, id)

Initialize and start the communication server. This function should be called in a `par` statement from the main function.

Arguments:

service_chan[]	Array with the service channels connecting to each kernel.
notification_chan[]	Array with the notification channels connecting to each kernel.
unsigned int nr_kernels	Number of kernels connected to this Communication Server. `service_chan[]` and `notification_chan[]` should have `nr_kernels` elements.
chanend ring_in	Ring bus incoming chanend. This value can be `null` to disable ring bus. If either `ring_in` or `ring_out` is `null`, the ring bus will not be enabled.
chanend ring_in	Ring bus incoming chanend. This value can be `null` to disable ring bus. If either `ring_in` or `ring_out` is `null`, the ring bus will not be enabled.
unsigned int id	Globally unique ID for Communication Server.

Return value:

This function never returns, it will process events infinitely.

1.3 xtask_create_init_task

int xtask_create_init_task(code, stack_size, priority, tid, args)

Create an initial task (before the Operating System starts). One or more of these function calls should be wrapped in another function and passed to `xtask_kernel()`.

Arguments:

code	Pointer to the function that should be ran as a task. The function has the following signature: `void function(void *)`.
unsigned int stack_size	Stack size in 32-bit words.
unsigned int priority	Task priority, a number between 0 and 6. Lower number means higher priority.
unsigned int tid	Unique task ID.
void * args	Arguments passed to the new task (can be NULL).

Return value:

0	Always returns 0 currently.

1.4 xtask_create_task

int xtask_create_task(code, stack_size, priority, tid, args)

Create a new task by another task.

Arguments:

code	Pointer to the function that should be ran as a task. The function has the following signature: `void function(void *)`.
unsigned int stack_size	Stack size in 32-bit words.
unsigned int priority	Task priority, a number between 0 and 6. Lower number means higher priority.
unsigned int tid	Unique task ID.
void * args	Arguments passed to the new task (can be NULL).

Return value:

0	Always returns 0 currently.

1.5 xtask_delay_ticks

void xtask_delay_ticks(unsigned int ticks)

Delay task for a certain amount of kernel ticks. The task will not be scheduled while it is delayed, giving other tasks the oportunity to run.

Arguments:

ticks	Number of kernel ticks to delay the task.

Return value:

void

1.6 xtask_create_mailbox

unsigned int xtask_create_mailbox(id, inbox_size, outbox_size)

Create a new mailbox for inter-task communication.

Arguments:

unsigned int id	Globally unique mailbox identifier.
unsigned int inbox_size	Size of inbox in bytes.
unsigned int outbox_size	size of outbox in bytes.

Return value:

0	Currently always returns 0.

1.7 xtask_get_outbox

struct vc_buf * xtask_get_outbox(id)

Get access to outbox buffer.

Arguments:

unsigned int id	Globally unique mailbox identifier.

Return value:

Pointer to vc_buf structure that contains the buffer information.

1.8 xtask_send_outbox

unsigned int xtask_send_outbox(sender, recipient)

Send outbox to recipient. The sending task will be blocked until the message is delivered to the recipient task and the recipient task has actively received it.

Arguments:

unsigned int sender	Globally unique mailbox identifier of sender.
unsigned int recipient	Globally unique mailbox identifier of recipient.

Return value:

0	Message delivered.
1	Recipient could not be found.

1.9 xtask_get_outbox

struct vc_buf * xtask_get_inbox(id, location)

Receive a message. The task will be blocked if no message is available.

Arguments:

unsigned int id	Globally unique mailbox identifier.
unsigned int location	Location can be `LOCAL_TILE` or `ALL_TILES`. If one or more tasks have tried to send a message while the recipient task was not waiting for it, a flag will be set in the recipient task's mailbox to indicate that another task tried to send a message. When the recipient task calls this function it will inform all tasks that tried to send a message that it is now ready to receive a message. If `LOCAL_TILE` is given as argument it will only search for pending senders that make use of the same Communication Server. If `ALL_TILES` is given, the ring bus will be used to inform all Communication Servers. If all possible sending tasks make use of the same Communication Server, `LOCAL_TILE` should be used.

Return value:

Pointer to vc_buf structure that contains the buffer information of the received message.

1.10 xtask_create_thread

unsigned int xtask_create_thread(code, stackwords, args, obj_size, rx_buf_size, tx_buf_size)

Create a new dedicated hardware thread (local, same tile).

Arguments:

code	Pointer to the function that should be ran as a dedicated hardware thread. The function has the following signature: `void function(void *, chanend)`.
unsigned int stackwords	Stack size in 4-byte words.
void * args	Arguments passed to the new task (can be NULL).
unsigned int obj_size	Size in bytes of objects transferred through the channel. Must be a multiple of 4 bytes.
unsigned int rx_buf_size	Receive buffer size. Must be a multiple of obj_size.
unsigned int tx_buf_size	Transfer buffer size. Must be a multiple of obj_size.

Return value:

A handle to the new dedicated hardware thread.

1.11 xtask_create_remote_thread

unsigned int xtask_create_remote_thread(code, stackwords, args, obj_size, rx_buf_size, tx_buf_size)

Create a new dedicated hardware thread (different tile).
This function is highly expirimental!

Arguments:

unsigned int code	Function number to execute (not a function pointer!). The function has the following signature: `void function(void *, chanend)`.
unsigned int stackwords	Stack size in 4-byte words.
void * args	Arguments passed to the new task (can be NULL).
unsigned int obj_size	Size in bytes of objects transferred through the channel. Must be a multiple of 4 bytes.
unsigned int rx_buf_size	Receive buffer size. Must be a multiple of obj_size.
unsigned int tx_buf_size	Transfer buffer size. Must be a multiple of obj_size.

Return value:

A handle to the new dedicated hardware thread.

1.12 xtask_vc_get_write_buf

struct vc_buf * xtask_vc_get_write_buf(handle)

Receive a write buffer that can be filled by the task and transmitted to the dedicated hardware thread. This function should only be called once prior to the first transmission to the dedicated hardware thread. The function that sends the buffer to the dedicated hardware thread will return a new empty buffer.

Arguments:

unsigned int handle	Dedicated hardware thread handle.

Return value:

A pointer to a vc_buf struct that contains the information about the buffer that can be filled by the task and transmitted to the dedicated hardware thread. Null pointer when no buffer was available but this should not happen in regular operation.

1.13 xtask_vc_receive

struct vc_buf * xtask_vc_receive(handle, min_size)

Receive from a hardware thread through a virtual channel. The task will block if there is no (sufficient) data available.

Arguments:

unsigned int handle	Dedicated hardware thread handle.
unsigned int min_size	Minimum amount of data to receive in bytes. If set to 0, the minimum amount is a full buffer.

Return value:

Pointer to a vc_buf struct which contains all the information about the buffer such as the actual pointer to the buffer and the amount of data in the buffer.

1.14 xtask_vc_send

struct vc_buf * xtask_vc_send(buf)

Instruct the Communication Server to send the write buffer to the dedicated hardware thread. Receive a new empty write buffer that can be immediately filled by the task.

Arguments:

struct vc_buf * buf	Pointer to write buffer.

Return value:

A pointer to a vc_buf struct that contains the information about the buffer that can be filled by the task and transmitted to the dedicated hardware thread. Null pointer when no buffer was available but this should not happen in regular operation.