# Reduce runtime memory requirements using overlays

IN THIS DOCUMENT



An overlay is a block of code and data that is loaded into memory on demand during program execution. The overlay runtime is responsible for loading overlays into memory before the functions in the overlay are called. Each overlay has two addresses - an external location it is loaded from and a predetermined address in xCORE memory it is copied to.

The area of memory an overlay is copied to is called an overlay region. Several overlays may be associated with the same overlay region, but only one of these overlays can be loaded at once. Using overlays reduces runtime memory requirements since it is no longer necessary to reserve space in memory for all the code and data used in the application, instead it is sufficient to only reserve enough space for the largest overlay that can be loaded into each overlay region.

## 1 Create an overlay

An overlay is created by adding the overlay attribute to a function as shown in Figure 1.

```
// Add overlay attribute to function, supported in xC only.
[[overlay]]
void f() {
  ...
}

// Alternative overlay attribute syntax, works in C, C++ and xC.
__attribute__((overlay))
void g() {
  ...
}
```

**Figure 1:** Overlay attributes

This creates an overlay containing that function. Functions marked with the overlay attribute are known as overlay roots. The tools will automatically place code and read only data that is only referenced from the overlay root in the overlay. The



XMOS © 2013, All Rights Reserved

overlay is loaded on demand when the overlay root is called. Calls to the overlay root may be slow (since the overlay may need to be loaded), but once inside the overlay, code will execute at full speed.

Overlays may optionally be named:

```
[[overlay(foo)]]
void f() {
  ...
}
```

You can use this to force two functions into the same overlay by specifying the same overlay name for both functions.

## 2 Enable overlays

To enable the use of overlays in the xTIMEcomposer tools add the `-foverlay` option. You can specify the overlay runtime to use after the option, for example `-foverlay=flash` enables overlays and links against the flash overlay runtime.

**Figure 2:** Overlay runtimes

| Option | Description |
|---|---|
| `-foverlay=flash` | SPI flash overlay runtime. |
| `-foverlay=syscall` | Syscall overlay runtime. |

## 3 Use the SPI flash overlay runtime

Add the option `-foverlay=flash` to use the SPI flash overlay runtime. You should also call the `overlay_flash_init` function in your application to initialize the runtime, as shown in Figure 3.

To use the flash overlay runtime the application must be booted from flash. See XM-000949-PC. Overlays are only enabled on tiles that are attached to a SPI flash used for boot.

The second and third arguments to `overlay_flash_init` specify the maximum SPI clock frequency in MHz as a fractional number where the second argument is the numerator and the third argument is the denominator.

## 4 Share SPI ports with SPI flash overlay runtime

The call to `overlay_flash_init` gives ownership of the SPI ports to the runtime. This prevents the ports from being used for any other purpose by the application (e.g. reading and writing data using libflash). If necessary an application can temporarily take ownership of the SPI ports back from the runtime as shown in Figure 4

Call `overlay_flash_claim_ports` to take ownership of the SPI ports and call `overlay_flash_return_ports` to transfer ownership of the ports back

```
#include <overlay_flash.h>

// Declare the SPI ports.
fl_SPIPorts spi_ports = {
  PORT_SPI_MISO,
  PORT_SPI_SS,
  PORT_SPI_CLK,
  PORT_SPI_MOSI,
  XS1_CLKBLK_1
};
// Declare a movable pointer to the SPI ports.
fl_SPIPorts * movable spi_ports_ptr = &spi_ports;

int main()
{
  // Call overlay_flash_init() to initialize the flash overlay runtime,
  // passing ownership of the SPI ports to the runtime.
  overlay_flash_init(move(spi_ports_ptr), 100, 8);

  return 0;
}
```

**Figure 3:** Initializing flash overlay runtime

```
// Take ownership of the SPI ports back from the overlay runtime.
fl_SPIPorts * movable spi_ports = overlay_flash_claim_ports();

// Connect to the flash using libflash.
fl_connect(*spi_ports);
// Read and write flash here...
fl_disconnect();

// Transfer ownership of the ports back to the application.
overlay_flash_return_ports(move(spi_ports));
```

**Figure 4:** Taking ownership of SPI flash ports.

to the runtime. If the overlay runtime is loading an overlay the call to `overlay_flash_claim_ports` will block until the overlay runtime has finished loading the overlay. While the application has ownership of the SPI ports a call to an overlay root function that is not in a currently loaded overlay will block until ownership of the SPI ports is returned to the overlay runtime.

## 5 Use the syscall overlay runtime

Add the option `-foverlay=syscall` to use the syscall overlay runtime. Overlays are loaded from a host machine using a system call. No initialization of the runtime is required in the application code.

To use the syscall overlay runtime the application must be loaded over JTAG and it must be run while a debug adapter is attached. Applications using the syscall overlay runtime can also be run in simulation using XSIM.

Copyright © 2013, All Rights Reserved.

Xmos Ltd. is the owner or licensee of this design, code, or Information (collectively, the “Information”) and is providing it to you “AS IS” with no warranty of any kind, express or implied and shall have no liability in relation to its use. Xmos Ltd. makes no representation that the Information, or any particular implementation thereof, is or will be free from any claims of infringement and again, shall have no liability in relation to any such claims.