# *VME_SCV64*

The VME_SCV64 module is a bus controller for the VMEbus controlled by a Tundra SCV64 chip.

## Process Information

| | |
|---|---|
| Prototype Name | vme |
| Link Order | before any VMEbus drivers |
| Process Name | unused (‘initonly’ process). |

## Target File Definitions

| | |
|---|---|
| VME_SCV64_BASE | The base address of the memory-mapped registers for the SCV6 chip. |
| VME_IACK*n* | (*n* = 0..7). The address of the *n*th VMEbus interrupt-acknowledge location . |
| VME_IPL_REG | The address of the Interrupt Priority Level Register. |
| VME_[VEC_]XINT0 | The interrupt pin [vector] number of the high-priority error interrupt. |
| VME_[VEC_]XINT1 | The interrupt pin [vector] number of the low-priority error and device interrupt. |
| VME_SIZE_SHARED | The size of the VME shared memory region, in bytes. |

## Process Operation

The initialisation routine configures the VMEbus for operating and installs the interrupt handlers for the two interrupts. It sets the VMEbus interrupt handlers to their default values and clear the user-settable ‘DMA-done’ handler.

The interrupt handler interprets VMEbus errors, creating trace records as appropriate. It calls the registered handler for device interrupts and the *done* handler for DMA completions.

There is no main process, and the module does not handle any messages.

## Shared Library Macros and Routines

The following routines are used by drivers for devices connected through the VMEbus to handle interrupts, DMA requests, and shared-memory allocations.

## vme_add_done

**int** *vme_add_done*(
  **void (****routine_ptr***)**())

The *vme_add_done* routine sets *routine_ptr* to be the routine called whenever a ‘DMA done’ interrupt is signalled to the VMEbus interrupt handler. The routine returns 0 if the handler was added successfully, and -1 otherwise.

## vme_add_handler

**int** *vme_add_handler*(
  **uint** vector**,**
  **void (****func_ptr***)(int)**,
  **int** *parameter*)

The *vme_add_handler* routine sets *func_ptr* to be the handler for the VMEbus interrupt at *vector.* The handler is called with *parameter* as its argument. The routine returns 0 if the handler was installed and -1 otherwise.

## vme_disable_interrupt

**int** *vme_disable_interrupt*(
  **uint** *level*)

The *vme_disable_interrupt* routine masks off the VMEbus interrupt at *level.* The routine returns 0 if the interrupt was sucessfully masked, -1 otherwise.

## vme_enable_interrupt

**int** *vme_enable_interrupt*(
  **uint** *level*)

The *vme_enable_interrupt* routine allows the VMEbus interrupt at *level* to reach the processor. The routine returns 0 if the interrupt was sucessfully enabled, -1 otherwise.

## vme_generate_interrupt

**int** *vme_generate_interrupt*(
  **uint** *vector,*
  **uint** *level*)

The *vme_generate_interrupt* routine allows software to generate a VMEbus interrupt with code *vector* at priority *level.* The routine returns 0 if the interrupt was sucessfully generated, and non-zero if there was already an interrupt in progress at this level.

## vme_remove_done

**void** *vme_remove_done*(**void**)

The *vme_remove_done* routine resets the ‘DMA done’ interrupt handler to its default internal value.

### vme_remove_handler

> **int** *vme_remove_handler*(
>     **uint** *vector*)

The *vme_remove_handler* routine resets the handler for the VMEbus interrupt at *vector* to its default internal value. The routine returns 0 if the handler was succesfully reset, -1 otherwise.

### vme_shared_malloc

> **ptr** *vme_shared_malloc*(
>     **int** *size*)

The *vme_shared_malloc* routine returns a pointer to *size* bytes of memory within the shared VMEbus address space, that is accessible both from the processor and from devices on the bus. As the usual purpose of this memory is be to given to a device as an input or output buffer, it is generally difficult to know at what point there are no references to the address being kept by devices, so there is no corresponding *free* routine.

## Debug Support

The *vme_view_errors* routine is callable from within the debugger, and will print the details of the most recent 32 VME event traces added to the trace buffer.

The *scv64_debug* routine formats all the SCV64 chip registers, interpreting the various bit fields for the status and error conditions.