# TEMU

## *Model Reference*

Version 2.2, 2020-06-18

# Table of Contents

# Chapter 1. Models

This is the reference manual for the TEMU models.

Each model description is normally structured using the following sections:

- Introduction
- Loading the Plugin
- Configuration
- Limitations

In addition, models have reference content such as properties, interfaces and registers listed.

# Chapter 2. AMBA

TEMU provides support for AMBA plug-and-play as used in the Gaisler GRLIB. The AMBA bus support and interfaces are defined in `temu-c/Bus/Amba.h`. In addition to the interfaces implemented by device models, the `AhbCtrl` and `ApbCtrl` classes are provided.

## 2.1. Interfaces

The interesting interfaces are defined in the `temu-c/Bus/Amba.h` header. This header provides support constants and helper functions, used to work with the PnP info structs.

```
typedef struct {
  uint32_t IdentReg;
  uint32_t UserDef[3];
  uint32_t Bar[4];
} temu_AhbPnpInfo;

typedef struct temu_AhbIface {
  temu_AhbPnpInfo* (*getAhbPnp)(void *Obj);
} temu_AhbIface;

typedef struct {
  uint32_t ConfigWord;
  uint32_t Bar;
} temu_ApbPnpInfo;

typedef struct temu_ApbIface {
  temu_ApbPnpInfo* (*getApbPnp)(void *Obj);
} temu_ApbIface;
```

## 2.2. Classes

There are two important classes provided, the AhbCtrl and ApbCtrl classes. These are available in `libTEMUAhbCtrl.so` and `libTEMUApbCtrl.so`.

When configuring a non-standard LEON3 / LEON4 based processor, the AHB and APB controllers must be instantiated and connected to devices implementing the plug and play interfaces. For the `AhbCtrl` class, connections is done using the `masters` and `slaves` array properties. For the `ApbCtrl` class, only the `slaves` property exist.

### 2.2.1. Attributes

#### Properties

| Name | Type | Description |
|---|---|---|
| masters | [64 x iref / <unknown>] | |
| object.timeSource | object | Time source object (a cpu or machine object) |
| slaves | [64 x iref / <unknown>] | |

#### Interfaces

| Name | Type | Description |
|---|---|---|
| DeviceIface | DeviceIface | |
| MemAccessIface | MemAccessIface | |
| ResetIface | ResetIface | |

#### Ports

| Prop | Iface | Description |
|---|---|---|
| - | - | - |

### 2.2.2. Attributes

#### Properties

| Name | Type | Description |
|---|---|---|
| object.timeSource | object | Time source object (a cpu or machine object) |
| pnp.bar | [4 x uint32_t] | |
| pnp.identReg | uint32_t | |
| pnp.userDef | [3 x uint32_t] | |
| slaves | [512 x iref / <unknown>] | |

#### Interfaces

| Name | Type | Description |
|---|---|---|
| AhbIface | AhbIface | |
| DeviceIface | DeviceIface | |
| MemAccessIface | MemAccessIface | |
| ResetIface | ResetIface | |

#### Ports

| Prop | Iface | Description |
|---|---|---|
| - | - | - |

## 2.3. Examples

The first example shows how to create and connect the AHB and APB bus controllers.

```
import AhbCtrl
import ApbCtrl

# Create two bus objects
object-create class=AhbCtrl name=ahbctrl0
object-create class=ApbCtrl name=apbctrl0

# Map to the normal addresses
memory-map memspace=mem0 addr=0x800ff000 length=0x1000 object=apbctrl0
memory-map memspace=mem0 addr=0xfffff000 length=0x1000 object=ahbctrl0

# Connect various APB devices to the APB controller
connect a=apbctrl0.slaves b=ftmctrl0:ApbIface
connect a=apbctrl0.slaves b=apbuart0:ApbIface
connect a=apbctrl0.slaves b=irqMp0:ApbIface
connect a=apbctrl0.slaves b=gpTimer0:ApbIface
connect a=apbctrl0.slaves b=ahbstat0:ApbIface

# Connect various AHB devices to the AHB controller
connect a=ahbctrl0.masters b=cpu0:AhbIface
connect a=ahbctrl0.slaves b=ftmctrl0:AhbIface
connect a=ahbctrl0.slaves b=apbctrl0:AhbIface
```

The next example shows how to implement a simple APB device.

```
#include "temu-c/Bus/Amba.h"

// This is the model type, we need to add the Pnp info.
typedef struct MyDevice {
  temu_ApbPnpInfo Pnp;
  // ...
} MyDevice;


// Implement the APB PNP interface
temu_ApbPnpInfo*
getApbPnp(void *Obj)
{
   MyDevice *Dev = (MyDevice*)Obj;
```

```
   return &Dev->Pnp;
}

temu_ApbIface ApbIface = {
  .getApbPnp = getApbPnp
};


// Define functions to allocate and destroy the object
void*
create(int Argc, const temu_CreateArg *Argv)
{
  MyDevice *Dev = malloc(sizeof(MyDevice));
  memset(Dev, 0, sizeof(MyDevice));

  // PNP init
  temu_apbSetVendorId(&MyDevice->Pnp, 0x99);
  temu_apbSetDeviceId(&MyDevice->Pnp, 0x001);
  temu_apbSetVersion(&MyDevice->Pnp, 1);

  temu_apbSetAddr(&MyDevice->Pnp, 0);
  temu_apbSetCP(&MyDevice->Pnp, 0);
  temu_apbSetMask(&MyDevice->Pnp, 0xfff);
  temu_apbSetType(&MyDevice->Pnp, 1); // APB I/O space

  return MyDevice;
}

void
dispose(void *Obj)
{
  MyDevice *Dev = (MyDevice*)Obj;
  free(Irq);
}

// Define the device interface
void
reset(void *Obj, int ResetKind)
{
}

void
mapDevice(void *Obj, uint64_t Addr, uint64_t Len)
{
  MyDevice *Dev = (MyDevice*)Obj;
  temu_apbSetAddr(&Dev->Pnp, Addr);
}
```

```
temu_DeviceIface DeviceIface = {
  reset, // Called on resets
  mapDevice, // Called when a device is mapped to a memory location.
};



TEMU_PLUGIN_INIT
{
  temu_Class *cls = temu_registerClass("MyClass", create, dispose);

  temu_addInterface(cls, "ApbIface", "ApbIface", &ApbIface);
  temu_addInterface(cls, "DeviceIface", "DeviceIface", &DeviceIface);

}
```

# Chapter 3. APBUART

The ApbUart model is available in the ApbUart plugin. That plugin is part of the GRLIB device library feature. The ApbUart model supports both FIFO simualtion and infinite speed UARTs. In infinite speed mode bytes are sent directly when they are written to the data register.

## 3.1. Loading the Plugin

```
import ApbUart
```

## 3.2. Attributes

### 3.2.1. Properties

| Name | Type | Description |
|---|---|---|
| config.clockDivider | uint32_t | |
| config.fifoSize | uint8_t | |
| config.infiniteUartSpeed | uint8_t | |
| config.interrupt | uint8_t | |
| control | uint32_t | |
| data | uint32_t | |
| fifo_debug | uint32_t | |
| irqCtrl | iref / <unknown> | |
| object.timeSource | object | Time source object (a cpu or machine object) |
| pnp.bar | uint32_t | |
| pnp.config | uint32_t | |
| rxFifo.data | [32 x uint8_t] | |
| rxFifo.size | uint8_t | |
| rxFifo.start | uint8_t | |
| rxFifo.usage | uint8_t | |
| scaler | uint32_t | |
| status | uint32_t | |
| tx | iref / <unknown> | |
| txFifo.data | [32 x uint8_t] | |
| txFifo.size | uint8_t | |

| Name | Type | Description |
|---|---|---|
| txFifo.start | uint8_t | |
| txFifo.usage | uint8_t | |
| txShift | uint8_t | |

### 3.2.2. Interfaces

| Name | Type | Description |
|---|---|---|
| ApbIface | ApbIface | |
| DeviceIface | DeviceIface | |
| MemAccessIface | MemAccessIface | |
| ResetIface | ResetIface | |
| UartIface | SerialIface | |

### 3.2.3. Ports

| Prop | Iface | Description |
|---|---|---|
| tx | UartIface | serial port |

## 3.3. Limitations

- Loop back mode is not presently supported.
- Control flow (cts) is not supported

# Chapter 4. CAN

TEMU provides support for CAN bus based devices. The bus model interfaces are available in: "temu-c/Bus/Can.h". In addition to the interfaces one CAN bus model is provided.

As CAN is a multi-node bus, a bus model object is needed to route messages to the relevant destination.

There are two types of CAN classes that can be created, firstly bus models and secondly device models. The difference is that a bus model is responsible for routing messages. To the device models, and the device models implement CAN message reception logic.

The standard SimpleCANBus bus model, provides fairly dumb logic. It routes a sent message to all devices connected to the CAN bus (except the sender device). However, CAN devices often implements filtering of message IDs in hardware, and this filtering (which is typically based on a mask and code pair) can be used to define a smart CAN bus model which can route frames using internal routing tables.

However, a smart CAN bus model is not necessarily faster for a small CAN network. Currently, TEMU is not delivered with a smart bus model, in fact the optimal routing algorithm depends on the allocation of message IDs and whether or not extended message IDs are used and how many filters are supported per device. While a smart bus model may be provided in the future, none is provided at present.

## 4.1. Interfaces

The interesting interfaces are defined in the temu-c/Bus/Can.h header. This header also define inline functions to help construct CAN frames.

```
typedef struct {
  uint8_t Data[8];
  uint32_t Flags;
  uint8_t Length;
  uint8_t Error;
} temu_CanFrame;

struct temu_CanDevIface {
  void (*connected)(void *Dev, temu_CanBusIfaceRef Bus);
  void (*disconnected)(void *Dev);
  void (*receive)(void *Dev, temu_CanFrame *Frame);
};

struct temu_CanBusIface {
  void (*connect)(void *Bus, temu_CanDevIfaceRef Dev);
  void (*send)(void *Bus, void *Sender, temu_CanFrame *Frame);
  void (*enableSendEvents)(void *Bus);
  void (*disableSendEvents)(void *Bus);
```

```
  void (*reportStats)(void *Bus);
  void (*setFilter)(void *Bus, temu_CanDevIfaceRef Dev, int FilterID,
                    uint32_t Mask, uint32_t Code);
};
```

The CAN frame is central to the transmission of CAN data. It is not a bit by bit representation of the CAN protocol, rather it is a simplified format that omit bits that are implicit and ensures that relevant bits such as RTR is fixed in location.

If a real CAN frame is needed, you need to transform the frame struct to the needed representation. Note that the struct is optimised for performance (e.g. Data is first and can be bitcopied as a uint64).

Device models are typically simple, they implement the connected, disconnected and receive functions. Of-course, if the device also need registers and MMIO handling, it tend to get more complex.

As can be seen, the device and bus interface support connect and disconnect events. The purpose of these are to support hot-plugging of CAN devices. As these connect and disconnect events are supported, the normal connect command should not be used when connecting a CAN device, rather the "can-connect" command is to be used.

## 4.2. Commands

Two CAN bus related commands are provided:

| Name | Description |
|---|---|
| can-connect | Connect a CAN device to a CAN bus. |
| can-disconnect | Disconnect CAN device from a CAN bus. |

## 4.3. Classes

The SimpleCAN bus class provides a CAN bus model. In the SimpleCANBus class, messages are forwarded to all connected devices (except the sending one). If this results in performance issues, it is possible to write a filtering CAN bus model.

### 4.3.1. Attributes

#### Properties

| Name | Type | Description |
|---|---|---|
| devices | irefarray / <unknown> | CAN devices attached to bus |
| object.timeSource | object | Time source object (a cpu or machine object) |
| stats.lastReportSentBits | uint64_t | Statistics |

| Name | Type | Description |
|---|---|---|
| stats.sentBits | uint64_t | Statistics |

### Interfaces

| Name | Type | Description |
|---|---|---|
| CanBusIface | CanBusIface | CAN Bus Interface |

### Ports

| Prop | Iface | Description |
|---|---|---|
| - | - | - |

## 4.4. Examples

This example shows how to create a simple CAN device and connect it to a bus model.

```
exec ut700.temu
import MyCanDevice
# Create a can bus
create class=SimpleCANBus name=canbus0
create class=MyCANClass name=mycan0

can-connect bus=canbus0:CanBusIface dev=occan0:CanDevIface # From ut700
can-connect bus=canbus0:CanBusIface dev=mycan0:CanDevIface
```

The next example shows how to implement a simple CAN device

```
#include "temu-c/Bus/Can.h"
#include "temu-c/Bus/Objsys.h"

// This is a device / RTU model, it needs to know about its CAN bus
typedef struct MyCanDevice {
  temu_Object Super;
  temu_CanBusIfaceRef Bus;
} MyCanDevice;


void*
create(const char *Name, int Argc, const temu_CreateArg *Argv)
{
  MyCanDevice *Dev = malloc(sizeof(MyCanDevice));
  memset(Dev, 0, sizeof(MyCanDevice);
  return Dev;
}
```

```
void
dispose(void *Obj)
{
  MyCanDevice *Dev = (MyCanDevice*)Obj;
  free(Dev);
}

// Implement the CAN Device interface

void
connected(void *Obj, temu_CanBusIfaceRef Bus)
{
  MyCanDevice *Dev = (MyCanDevice*)Obj;
  Dev->Bus = Bus;
  temu_logInfo(Dev, "connected to CAN bus");
}

void
disconnected(void *Obj)
{
  MyCanDevice *Dev = (MyCanDevice*)Obj;
  Dev->Bus = {NULL, NULL};
  // NOTE: This should also stop any pending events related to
  // message transmissions
  temu_logInfo(Dev, "disconnected from CAN bus");
}

void
receive(void *Dev, temu_CanFrame *Frame)
{
  temu_logInfo(Dev, "received CAN message with msg id %u",
               temu_canGetIdent(Frame));
}

temu_CanDevIface CanIface = {
  connected,
  disconnected,
  receive,
};

TEMU_PLUGIN_INIT
{
  temu_Class *cls = temu_registerClass("MyCANClass", create, dispose);

  temu_addProperty(cls, "CANBus", teTY_IfaceRef, 1);
  temu_addInterface(cls, "CanDevIface", "CanDevIface", &CanIface);
```

```
}
```

# Chapter 5. CAN_OC

The CAN_OC device is part of the OpenCores and the GRLIB IP libraries. It is available in `libTEMUOpenCores.so`.

## 5.1. Loading the Plugin

```
import OpenCores
```

## 5.2. Configuration

There are two configuration parameters in the CAN device. Firstly the `config.interrupt` property can be set to influence the interrupt that is raised with the IRQ controller. Setting that property also updates the AHB PnP info.

The second configuration property is `config.infiniteSpeed`. If that property is set, messages will be sent immediately instead of being scheduled.

The device should be connected to an interrupt controller and a CAN bus, to work properly.

### 5.2.1. Attributes

#### Properties

| Name | Type | Description |
|---|---|---|
| basiccan.acceptCode | uint8_t | Accept Code register for BasicCAN mode. |
| basiccan.acceptMask | uint8_t | Accept Mask register for BasicCAN mode. |
| basiccan.ctrl | uint8_t | Control register for BasicCAN mode. |
| basiccan.txID | [2 x uint8_t] | TxID registers for BasicCAN mode. |
| bus | iref / <unknown> | CAN bus the device is connected to. |
| busTiming | [2 x uint8_t] | Bus Timing registers. |
| clockDivider | uint8_t | Clock Divider register. |
| command | uint8_t | Command register. |
| config.infiniteSpeed | uint8_t | Enable infinite speed mode (no delays when sending messages). |

| Name | Type | Description |
|---|---|---|
| config.interrupt | uint8_t | External interrupt raised with IRQ controller. |
| fifo.data | [64 x uint8_t] | RX FIFO data buffer. |
| fifo.start | uint32_t | RX FIFO buffer start location. |
| fifo.usage | uint32_t | RX FIFO buffer usage. |
| interrupt | uint8_t | Interrupt register. |
| irqCtrl | iref / <unknown> | Interrupt controller. |
| object.timeSource | object | Time source object (a cpu or machine object) |
| pelican.acceptCode | [4 x uint8_t] | Accept Code registers for PeliCAN mode. |
| pelican.acceptMask | [4 x uint8_t] | Accept Mask registers for PeliCAN mode. |
| pelican.arbLostCaputure | uint8_t | Arbitration Lost Capture register for PeliCAN mode. |
| pelican.errCodeCapture | uint8_t | Error Code Capture register for PeliCAN mode. |
| pelican.errWarnLimit | uint8_t | Error Warning Limit register for PeliCAN mode. |
| pelican.interruptEnable | uint8_t | Interrupt Enable register for PeliCAN mode. |
| pelican.mode | uint8_t | Mode register for PeliCAN mode. |
| pelican.rxErrCounter | uint8_t | RX Error Counter register for PeliCAN mode. |
| pelican.rxMsgCounter | uint8_t | RX Message Counter register for PeliCAN mode. |
| pelican.txErrCounter | uint8_t | TX Error Counter register for PeliCAN mode. |
| pelican.txFI | uint8_t | TX Frame Info register for PeliCAN mode. |
| pelican.txID | [4 x uint8_t] | TxID registers for PeliCAN mode. |
| status | uint8_t | Status register. |
| txData | [8 x uint8_t] | TX data buffer (excluding TX FI and TX ID registers). |

### Interfaces

| Name | Type | Description |
|---|---|---|
| AhbIface | AhbIface | AHB interface |
| CanDevIface | CanDevIface | CAN device interface. |
| DeviceIface | DeviceIface | Device interface. |
| MemAccessIface | MemAccessIface | Memory access interface for memory mapped registers. |
| ResetIface | ResetIface | |

### Ports

| Prop | Iface | Description |
|---|---|---|
| - | - | - |

## 5.3. Limitations

The following deviations from real hardware are known to exist with this model:

- The controller clears the RX and TX buffers on reset. This is not the proper behaviour and may have an impact on FDIR. Let us know if this is an issue.
- With all CAN models, there is no arbitration of messages in the simulated world and busses are not synchronised.
- The model does at present not register filters with the CAN bus model.
- The model currently ignores the error field in the CAN frame objects.
- The model currently assumes the CAN bus is running at 1 Mb/s (in non-infinite speed mode). This is arguably incorrect and the timing should be picked from the bus timing register, this has however not yet been done. Contact Terma if this is this is critical for your needs.

# Chapter 6. Ethernet

TEMU provides support for Ethernet bus based devices. To support the development of custom MAC controllers, TEMU provides three generic models.

The `MDIOBus` model implements MDIO routing. As multiple MDIO devices can be connected to the same bus, a bus model is needed.

A `GenericPHY` model is implemented to expose the MDIO interface to the MAC models.

The `GenericPHY` model can be attached to the `EthernetLink` model. `EthernetLink` is responsible for routing EthernetFrames between registered nodes. It has two routing lists. Firstly, a list of *promiscuous* nodes that will receive all messages. Secondly, a routing map for non-promiscuous nodes.

When the `EthernetLink` model receives a frame, it forwards the frame to all the promiscuous nodes. Then, it routes it to the destination MAC.

The `EthernetLink` assumes unique MACs, thus it will emit a warning in the case of a MAC address collision.

## 6.1. Connections

An ethernet link must be connected to its attached PHYs. Connection is done using the `connect` command.

*Example 1. Connect Syntax*

```
ethernet-connect link=ethlink0 phy=phy0:PHYIface
```

*Example 2. Disconnect Syntax*

```
ethernet-disconnect link=ethlink0 phy=phy0:PHYIface
```

## 6.2. Checksums

Ethernet frames typically have a checksum that is generated and checked by hardware. To optimise the bus model, it is expected that MAC models supports opt in control on checksum generation and checking. This applies to all checksums, including Ethernet frame CRCs and IP header, TCP, UDP checksums. Since the Ethernet link is fully virtual, data cannot normally be corrupted in transit. Thus checksum checking and generation would be a waste of cycles.

There are still several usecases where one want to enable checksums:

- When viewing capture files with *Wireshark*, the tool will complain if ethernet CRCs are invalid.

- When receiving frames in a device which do not have hardware assisted CRC checking.

Thus, normally Ethernet CRC generation and checking will be disabled, while TCP/UDP/IP checksum generation (but not hardware checking) will be enabled.

## 6.3. Auto Negotiation

The ethernet model supports autonegotiation for transfer speed capabilities.

The process is based on issuing an auto-negotiation request to the ethernet link model. The link will then issue autonegotiationg requests to each attached PHY, and finally call autonegotiateDone for all attached PHYs.

Each PHY will be called with the current known capabilities. It should return the same capabilities with potentially some of them cleared.

The actual final capabilities are reported with autonegotiateDone.

There, a PHY will select the highest priority common mode. Which by the standard is:

1. 40GBASE T FD
2. 25GBASE T FD
3. 10GBASE T FD
4. 5GBASE T FD
5. 2.5GBASE T FD
6. 1000BASE T FD
7. 1000BASE T HD
8. 100BASE T2 FD
9. 100BASE TX FD
10. 100BASE T2 HD
11. 100BASE T4
12. 100BASE TX HD
13. 10BASE T FD
14. 10BASE T HD

Note that TEMU does not support emulation of 2.5 GBASE and above at this moment.

## 6.4. Frames

Ethernet frames in TEMU are structs containing a flag field, data and an optional preamble.

The data field is a COW buffer which contains the level 2 ethernet frame data.

The preamble will typically be ignored and not set for most MACs. However if it is set to somthing non-standard, a device can indicate this by setting the flag `TEMU_ETH_NON_STANDARD_PREAMBLE`.

# 6.5. Ethernet Link

## 6.5.1. Frame Capture

The ethernet link can be instructed to dump all traffic to a PCAPNG file.

Wireshark may flag frames as having invalid CRCs. To avoid this you can enable CRC generation in the MAC, or turn off checking in Wireshark.

To enable capture execute the enableCapture command ont he ethernet link.

*Example 3. Enable Capture Command*

```
ethernet-link-enable-capture link=ethlink0 file="foo.pcap"
```

## 6.5.2. Attributes

### Properties

| Name | Type | Description |
|---|---|---|
| object.timeSource | object | Time source object (a cpu or machine object) |

### Interfaces

| Name | Type | Description |
|---|---|---|
| EthernetIface | temu::EthernetIface | |

### Ports

| Prop | Iface | Description |
|---|---|---|
| - | - | - |

# 6.6. PHY Model

The `GenericPHY` is a PHY / MII device which supports both the MDIO interface and the PHY interface for sending/receiving ethernet frames.

The `GenericPHY` device class by default enables support for BASE10, BASE100 and BASE1000 transfers. To only enable specific speed modes, the constructor accepts arguments:

- base10:1

- base100:1
- base1000:1

If any of these are set, the unset ones will be disabled.

Thus by default a PHY supports all BASE10, BASE100 and BASE1000 modes. By setting the base10 argument, only BASE10 modes will be supported. By setting base10 and base 100 arguments, only BASE10 and BASE100 will be supported.

At present it is not possible to control the support on a lower level.

### 6.6.1. Attributes

#### Properties

| Name | Type | Description |
|---|---|---|
| autoNegAdvertisment | uint16_t | Auto negotiation advertisment register |
| autoNegotiationExpansion | uint16_t | Auto negotiation expansion register |
| basicModeConfig | uint16_t | Basic mode config register |
| basicModeStatus | uint16_t | Basic mode status register |
| ethernetLink | iref / <unknown> | Ethernet link |
| linkPartnerAbility | uint16_t | Link partner ability register |
| macDevice | iref / <unknown> | MAC device |
| object.timeSource | object | Time source object (a cpu or machine object) |
| phyID | [2 x uint16_t] | Physical ID registers |

#### Interfaces

| Name | Type | Description |
|---|---|---|
| MDIOIface | temu::MDIOIface | |
| PHYIface | temu::PHYIface | |

#### Ports

| Prop | Iface | Description |
|---|---|---|
| - | - | - |

## 6.7. MDIO Model

The MDIO bus distributes MDIO control messages and supports routing of them. The MDIO bus use the same interface as an MDIO device. Thus, if only one MDIO device (e.g. GenericPHY) is available no MDIO bus instance is needed.

### 6.7.1. Attributes

#### Properties

| Name | Type | Description |
|---|---|---|
| macDevice | iref / <unknown> | |
| object.timeSource | object | Time source object (a cpu or machine object) |
| phyDevices | [32 x iref / <unknown>] | |

#### Interfaces

| Name | Type | Description |
|---|---|---|
| MDIOIface | temu::MDIOIface | |

#### Ports

| Prop | Iface | Description |
|---|---|---|
| - | - | - |

## 6.8. MAC Models

TEMU comes with some bundled MAC models. In some cases it will be needed to implement additional project specific MAC models.

Consult the `eth-device` example for more info.

# Chapter 7. Generic Cache

TEMU supports the use of cache models. However, cache models, at least when they are non-statistical have a significant impact on performance, and therefor, normally cache models are not used when running the emulator.

For the cases where a cache model is needed, the generic cache model is likely to be useful (see limitations for when it is less useful). It is a highly configurable cache model and supports being used, both as Harward style caches (separate I- and D-caches) and as a unified cache.

CAUTION : When connecting the generic cache model in the memory hierarchy, it will intercept every memory transaction, and disable the ATC for any fetched, read or written data. This means that performance is significantly impacted firstly due to the need to visit the memory system for every fetch, read and write, but also and especially in a system with an enabled MMU, in these systems, the CPU will need to do a VM table walk for every memory access, which is very costly in terms of performance. Note that these table walks may be optimised in the future.

The cache model will handle memory accesses with the TEMU_MT_CACHEABLE flag set. This flag can be set when mapping in a device (e.g. RAM or ROM).

## 7.1. Configuration

### 7.1.1. Attributes

#### Properties

| Name | Type | Description |
|---|---|---|
| data.lineBits | uint32_t | |
| data.lineMask | uint32_t | |
| data.lineSize | uint32_t | line size in bytes |
| data.lineWordSizeLg2 | uint32_t | log 2 of line-size in words |
| data.replacementPolicy | int32_t | data cache replacement policy (0=none, 1=lru, 2=lrr, 3=rnd) |
| data.rndReplaceWay | int32_t | |
| data.setBits | uint32_t | |
| data.setMask | uint32_t | |
| data.setShift | uint32_t | |
| data.sets | uint32_t | number of sets |
| data.status | uint32_t | status of data cache |
| data.ways | uint32_t | number of ways in the cache |
| dcacheCtrl | iref / <unknown> | data cache controller |

| Name | Type | Description |
|---|---|---|
| fetchHits | uint64_t | |
| fetchMisses | uint64_t | |
| fetchPenalty | int32_t | |
| icacheCtrl | iref / <unknown> | instruction cache controller |
| instr.lineBits | uint32_t | |
| instr.lineMask | uint32_t | |
| instr.lineSize | uint32_t | line size in bytes |
| instr.lineWordSizeLg2 | uint32_t | log 2 of line-size in words |
| instr.replacementPolicy | int32_t | instruction cache replacement policy (0=none, 1=lru, 2=lrr, 3=rnd) |
| instr.rndReplaceWay | int32_t | |
| instr.setBits | uint32_t | |
| instr.setMask | uint32_t | |
| instr.setShift | uint32_t | |
| instr.sets | uint32_t | number of sets |
| instr.status | uint32_t | status of instruction cache |
| instr.ways | uint32_t | number of ways in the cache |
| isSplitCache | int32_t | |
| isWriteAllocate | int32_t | |
| isWriteBack | int32_t | |
| object.timeSource | object | Time source object (a cpu or machine object) |
| postTransaction | iref / <unknown> | |
| preTransaction | iref / <unknown> | |
| readHits | uint64_t | |
| readMisses | uint64_t | |
| readPenalty | int32_t | |
| wordSize | int32_t | |
| writeHits | uint64_t | |
| writeMisses | uint64_t | |
| writePenalty | int32_t | |

#### Interfaces

| Name | Type | Description |
|---|---|---|
| DCacheIface | CacheIface | |
| ICacheIface | CacheIface | |
| ObjectIface | ObjectIface | |
| PostAccessIface | MemAccessIface | |
| PreAccessIface | MemAccessIface | |

#### Ports

| Prop | Iface | Description |
|---|---|---|
| - | - | - |

### 7.1.2. Arguments

**size**

Unified cache size in bytes.

**instrSize**

Instruction cache size in bytes.

**dataSize**

Data cache size in bytes.

**ways**

Number of ways in a unified cache (must be power of 2)

**instrWays**

Number of ways in instruction cache (must be power of 2)

**dataWays**

Number of ways in data cache (must be power of 2)

**lineSize**

Line size for unified cache

**dataLineSize**

Line size for data cache

**instrLineSize**

Line size for instruction cache

**wordSize**

Size of a word in bytes (defaults to 4)

**separate**

Set to 1 to turn the cache model to separate I- and D-caches. Set to 0 to make the cache a unified cache. This option affects the interpretation of the size, ways and lineSize arguments (see above).

### 7.1.3. Interfaces

The following interfaces can be used to connect the generic cache model:

**PreAccessIface**

A MemAccessIface that receives memory access events before they reach the target device.

**PostAccessIface**

A MemAccessIface that handles memory access events after they reach the target device.

### 7.1.4. Properties

The following properties are used for configuring the cache model and to connect the model in the object graph.

**preTransaction**

Memory access interface reference for next pre-access handler.

**postTransaction**

Memory access interface reference for next post-access handler.

**icacheCtrl**

Optional interface reference for a instruction cache controller object.

**dcacheCtrl**

Optional interface reference for a data cache controller object.

**instr.replacementPolicy**

Replacement policy used when fetching instructions. Set to 0 = NONE (or directly mapped / 1-way set associative cache). 1 = LRU, 2 = LRR and 3 = RND. Automatically set to 0 when ways is set to 1.

**data.replacementPolicy**

Replacement policy used when accessing data. Set to 0 = NONE (or directly mapped / 1-way set associative cache). 1 = LRU, 2 = LRR and 3 = RND. Automatically set to 0 when ways is set to 1.

**isSplitCache**

Cache is split and has separate instruction and data caches.

**isWriteBack**

Cache is write-back cache, not supported at the moment.

**isWriteAllocate**

Set to non-zero to have the cache allocate a line in case of a write miss. Set to zero to avoid line allocation.

**fetchPenalty**

Cost for fetching from a cached line.

**readPenalty**

Cost for reading from a cached line.

**writePenalty**

Cost for writing to a cached line.

**wordSize**

Word size for cache (defaults to 4, do not modify unless connecting to 64-bit processor architectures).

**instr.sets**

Number of sets in the instruction cache.

**instr.ways**

Number of ways in the instruction cache.

**instr.lineSize**

Instruction line size in bytes.

**data.sets**

Number of sets in the data cache.

**data.ways**

Number of ways in the data cache.

**data.lineSize**

Data line size in bytes.

## 7.2. Properties

The generic cache model contains the following counters that can be inspected to get an idea of hit and miss-rates.

**fetchHits**

Number of cache hits when fetching instructions.

**fetchMisses**

Number of cache misses when fetching instructions.

**readHits**

Number of cache hits when reading data.

**readMisses**

Number of cache misses when reading data.

**writeHits**

Number of cache hits when writing data.

**writeMisses**

Number of cache misses when writing data.

## 7.3. Limitations

- The cache does not emulate write-back penalties for write-back caches at present. This means that the evict functions will behave as the invalidate functions.
- Number of ways must be a power of 2. That means that 1- 2- and 4- way set associative caches are fine, but 3-way set associative caches are not emulated by the generic cache model.

# Chapter 8. GPIO Bus

This bus model is deprecated. Users should migrate to the SignalIface.

The GPIO bus model is one of the standard bus models available in TEMU. The bus model maintains the values of 64 GPIO pins, and a notification list where pin updates can be forwarded to an arbitrary number of models when pin values have changed.

This does place a limitation, in that a model must know which pin it is connected to, which may not be ideal. The recommended approach is to ensure that the model maintains its own user configurable mask for filtering out the relevant bits.

## 8.1. Configuration

The GpioBus model can be configured by connecting GPIO clients to the Clients property. No other configuration capabilities are provided.

## 8.2. Class Info

### 8.2.1. Attributes

#### Properties

| Name | Type | Description |
|---|---|---|
| Bits | uint64_t | |
| Clients | irefarray / <unknown> | |
| object.timeSource | object | Time source object (a cpu or machine object) |

#### Interfaces

| Name | Type | Description |
|---|---|---|
| GpioBusIface | GpioBusIface | |

#### Ports

| Prop | Iface | Description |
|---|---|---|
| - | - | - |

## 8.3. Limitations

The primary limitation of the GPIO bus model is that pin updates using the GpioBusIface will be distributed to all GpioClients that have been connected to the GPIO bus. If requested the bus model can be augmented with direct distribution properties for forwarding individual pin changes to

predetermined objects. This has not been implemented yet though, contact us if you need support for this.

# Chapter 9. GPTIMER

The GPTIMER is part of the GRLIB device library from Gaisler. The timer runs using synchronised events in order to ensure that would a timer tick be broadcasted by the interrupt controller, then the IRQ should be taken at roughly the same time.

## 9.1. Loading the Plugin

```
import GpTimer
```

## 9.2. Limitations

The following deviations from real hardware are known to exist with this model:

- The Disable Timer Freeze bit is always 1 and cannot be configured.
- The Debug Halt bit for each timer is always 0 and cannot be altered.
- Chained timers are not supported at the moment.
- The last timer does not work as a watchdog.
- As the timer utilise synchronised events, the minimum time for a timer expiration on a multi-core CPU would be equal to the time-quanta that the machine has been configured with.

# Chapter 10. GRCAN

The GRCAN model is available in the GrCan plugin.

## 10.1. Loading the Plugin

```
import GrCan
```

## 10.2. Attributes

### 10.2.1. Properties

| Name | Type | Description |
|---|---|---|
| bus | iref / CanBusIface | CAN bus |
| cfg | uint32_t | Congifuation register |
| config.irq | uint8_t | Interrupt number |
| config.singleIrq | uint8_t | Single interrupt |
| ctrl | uint32_t | Control register |
| irqCtrl | iref / IrqCtrlIface | IRQ controller |
| irqMask | uint32_t | Interrupt register |
| mem | iref / MemoryIface | Memory |
| object.timeSource | object | Time source object (a cpu or machine object) |
| pendIrq | uint32_t | Pending interrupt register |
| rxChanAddr | uint32_t | RX channel address register |
| rxChanCode | uint32_t | RX channel code register |
| rxChanCtrl | uint32_t | RX channel control register |
| rxChanIrq | uint32_t | RX channel irq register |
| rxChanMask | uint32_t | RX channel mask register |
| rxChanRd | uint32_t | RX channel read register |
| rxChanSize | uint32_t | RX channel size register |
| rxChanWr | uint32_t | RX channel write register |
| stat | uint32_t | Status register |
| syncCodeFilt | uint32_t | SYNC code filter register |
| syncMaskFilt | uint32_t | SYNC mask filter register |

| Name | Type | Description |
|---|---|---|
| txChanAddr | uint32_t | TX channel address register |
| txChanCtrl | uint32_t | TX channel control register |
| txChanIrq | uint32_t | TX channel irq register |
| txChanRd | uint32_t | TX channel read register |
| txChanSize | uint32_t | TX channel size register |
| txChanWr | uint32_t | TX channel write register |

### 10.2.2. Interfaces

| Name | Type | Description |
|---|---|---|
| ApbIface | ApbIface | APB P&P interface |
| CanDevIface | CanDevIface | CAN device interface |
| MemAccessIface | MemAccessIface | Memory access interface (registers) |

### 10.2.3. Ports

| Prop | Iface | Description |
|---|---|---|
| - | - | - |

## 10.3. Registers

Register support is currently experimental!

### 10.3.1. Register Bank default

#### Register cfg

Congifuation register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| - | - | - | - | - |

#### Register stat

Status register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| - | - | - | - | - |

### Register ctrl

Control register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| - | - | - | - | - |

### Register syncMaskFilt

SYNC mask filter register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| - | - | - | - | - |

### Register syncCodeFilt

SYNC code filter register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| - | - | - | - | - |

### Register pendIrq

Pending interrupt register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| - | - | - | - | - |

### Register irqMask

Interrupt register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| - | - | - | - | - |

### Register txChanCtrl

TX channel control register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| - | - | - | - | - |

### Register txChanAddr

TX channel address register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| - | - | - | - | - |

### Register txChanSize

TX channel size register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| - | - | - | - | - |

### Register txChanWr

TX channel write register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| - | - | - | - | - |

### Register txChanRd

TX channel read register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| - | - | - | - | - |

### Register txChanIrq

TX channel irq register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| - | - | - | - | - |

### Register rxChanCtrl

RX channel control register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| - | - | - | - | - |

### Register rxChanAddr

RX channel address register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| - | - | - | - | - |

### Register rxChanSize

RX channel size register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| - | - | - | - | - |

### Register rxChanWr

RX channel write register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| - | - | - | - | - |

### Register rxChanRd

RX channel read register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| - | - | - | - | - |

### Register rxChanIrq

RX channel irq register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| - | - | - | - | - |

### Register rxChanMask

RX channel mask register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| - | - | - | - | - |

### Register rxChanCode

RX channel code register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| - | - | - | - | - |

## 10.4. Limitations

- None

# Chapter 11. GRETH

The GRETH model is available in the GrEth plugin. The model needs to be combined with a MDIOBus, PHY and Ethernet model.

The GRETH model implements the behaviour of both `GRETH` and `GRETH_GBIT`.

## 11.1. Loading the Plugin

```
import BusModels
import GrEth
GRETH.new name=greth0
GenericPHY.new name=phy0
EthernetLink.new name=eth0
connect a=greth0.phy b=phy0:PHYIface
connect a=greth0.mdioBus b=phy0:MDIOIface
connect a=apbctrl0.slaves b=greth0:ApbIface
greth0.setMAC mac="00:00:00:00:00:01"
connect a=phy0.mac b=greth0:MACIface
eth0.connect device=phy0:PHYIface
```

## 11.2. Limitations

The `config.gbitVariant` property can be set to enable `GRETH_GBIT` extensions. The extensions includes:

- Gigabit speed.
- IP header checksum offloading
- TCP checksum offloading
- UDP checksum offloading
- Scatter / gather send lists.

### 11.2.1. Attributes

#### Properties

| Name | Type | Description |
|---|---|---|
| ETHCTR | uint32_t | Ethernet Control Register |
| ETHMDC | uint32_t | Ethernet MDIO Control and Status Register |
| ETHRDP | uint32_t | Ethernet Receiver Descriptor Pointer Register |

| Name | Type | Description |
|---|---|---|
| ETHSIS | uint32_t | Ethernet Status and Interrupt Source Register |
| ETHTDP | uint32_t | Ethernet Transmitter Descriptor Pointer Register |
| MACLSB | uint32_t | Ethernet MAC Address LSB |
| MACMSB | uint32_t | Ethernet MAC Address MSB |
| config.checkCrc | uint8_t | Enable ethernet frame CRC checking. |
| config.checkIpCrc | uint8_t | Enable IP header CRC checking. |
| config.checkTcpCrc | uint8_t | Enable TCP header CRC checking. |
| config.checkUdpCrc | uint8_t | Enable UDP header CRC checking. |
| config.gbitVariant | uint8_t | Enable GRETH_GBIT behaviour. |
| config.generateCrc | uint8_t | Enable ethernet frame CRC generation. |
| config.irq | uint8_t | IRQ |
| config.logTraffic | uint8_t | Enable traffic logging |
| irqCtrl | iref / IrqCtrlIface | IRQ controller |
| mac | cstring | Set MAC by string |
| mdioBus | iref / temu::MDIOIface | MDIO bus |
| memory | iref / MemoryIface | Memory |
| object.timeSource | object | Time source object (a cpu or machine object) |
| phy | iref / temu::PHYIface | PHY device |

### Interfaces

| Name | Type | Description |
|---|---|---|
| ApbIface | ApbIface | APB P&P interface |
| DeviceIface | DeviceIface | |
| MACIface | temu::MACIface | MAC interface |
| MemAccessIface | MemAccessIface | Mem access interface |
| ResetIface | ResetIface | |

### Ports

| Prop | Iface | Description |
|---|---|---|
| - | - | - |

## 11.2.2. Registers

Register support is currently experimental!

### Register Bank registers

#### Register ETHCTR

Ethernet Control Register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| EA | 0x80000000 | 0x0 | 0x0 | EDCL available |
| BS | 0x70000000 | 0x0 | 0x0 | EDCL buffer size |
| GA | 0x8000000 | 0x0 | 0x0 | Gigabit MAC |
| MA | 0x4000000 | 0x0 | 0x0 | MDIO interrupts supported |
| MC | 0x2000000 | 0x0 | 0x0 | Multicast supported |
| SP | 0x80 | 0x0 | 0x0 | Speed |
| RS | 0x40 | 0x0 | 0x0 | Reset |
| PM | 0x20 | 0x0 | 0x0 | Open Packet Mode |
| FD | 0x10 | 0x0 | 0x0 | Full Duplex |
| RI | 0x8 | 0x0 | 0x0 | Enable Receiver Interrupts |
| TI | 0x4 | 0x0 | 0x0 | Enable Transmitter Interrupts |
| RE | 0x2 | 0x0 | 0x0 | Receive Enable |
| TE | 0x1 | 0x0 | 0x0 | Transmit Enable |

#### Register ETHSIS

Ethernet Status and Interrupt Source Register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| IA | 0x80 | 0x0 | 0x0 | Invalid Address |
| TS | 0x40 | 0x0 | 0x0 | Too Small |
| TA | 0x20 | 0x0 | 0x0 | Transmitter AHB Error |
| RA | 0x10 | 0x0 | 0x0 | Receiver AHB Error |
| TI | 0x8 | 0x0 | 0x0 | Transmitter Interrupt |
| RI | 0x4 | 0x0 | 0x0 | Receiver Interrupt |
| TE | 0x2 | 0x0 | 0x0 | Transmitter Error |
| RE | 0x1 | 0x0 | 0x0 | Receiver Error |

#### Register MACMSB

Ethernet MAC Address MSB

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| MSB | 0xffff | 0x0 | 0x0 | Two MSB of MAC |

#### Register MACLSB

Ethernet MAC Address LSB

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| LSB | 0xffffffff | 0x0 | 0x0 | Four LSB of MAC |

#### Register ETHMDC

Ethernet MDIO Control and Status Register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| Data | 0xffff0000 | 0x0 | 0x0 | Data for MMI read / write |
| PHY_ADDR | 0xf800 | 0x0 | 0x0 | PHY address |
| REG_ADDR | 0x7c0 | 0x0 | 0x0 | MII reg addr |
| NV | 0x10 | 0x0 | 0x0 | Not valid |
| BU | 0x8 | 0x0 | 0x0 | Busy |
| LF | 0x4 | 0x0 | 0x0 | Link fail |
| RD | 0x2 | 0x0 | 0x0 | Read |
| WR | 0x1 | 0x0 | 0x0 | Write |

#### Register ETHTDP

Ethernet Transmitter Descriptor Pointer Register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| TXDTRA | 0xfffff800 | 0x0 | 0x0 | Tx desc base address |
| TX_DESCRIPTOR_PTR | 0x3f8 | 0x0 | 0x0 | Tx desc offset |

#### Register ETHRDP

Ethernet Receiver Descriptor Pointer Register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| RXDTRA | 0xfffff800 | 0x0 | 0x0 | Rx desc base address |
| RX_DESCRIPTOR_PTR | 0x3f8 | 0x0 | 0x0 | Rx desc offset |

## 11.3. Limitations

- Multicast groups are not yet supported.

- ECDL mode is not supported.

# Chapter 12. GRGPIO

The GRGPIO device is part of the GRLIB device library from Gaisler. The GrGPIO model simulates a 16 pin GPIO device by providing input and output via the SignalIface.

## 12.1. Loading the Plugin

```
import GrGPIO
```

## 12.2. Limitations

- Only the UT700 based configuration is supported at the moment. That means that the bypass and capabilities registers are missing. Further the IRQ map registers are not available.

# Chapter 13. GRSPW1

The GRSPW1 is part of the GRLIB IP library. It is available in libTEMUGrspw1.so.

## 13.1. Loading the Plugin

```
import Grspw1
```

## 13.2. Configuration

To work correctly, the device should be connected to an interrupt controller, the memory and another SpaceWire device.

There are several configuration parameters in the GrSpw1 device, summarized in the following table:

| Name | Description |
|---|---|
| config.infiniteSpeed | With this set, messages are sent immediately instead of being scheduled for the future based on the message length. This is the default option. |
| config.transmitter.frequency | Specify the SpaceWire transmitter frequency in Hz. Affects transfer speed when infinite speed is disabled. |
| config.transmitter.dataRate | SpaceWire port datarate: 1=single, 2=double, etc. Affects transfer speed when infinite speed is disabled. |
| config.interrupt | Influences the interrupt that is raised with the IRQ controller (setting this property also updates the APB PnP info). |
| config.realCrcCheck | Set to use real crc check instead of packet crc flags. Real crc costs in terms of performance. |

### 13.2.1. Attributes

#### Properties

| Name | Type | Description |
|---|---|---|
| config.infiniteSpeed | uint8_t | Set to use infinite speed for transfers. |
| config.interrupt | uint8_t | The interrupt index |

| Name | Type | Description |
|---|---|---|
| config.realCrcCheck | uint8_t | Set to use real crc check instead of packet crc flags |
| config.transmitter.dataRate | uint8_t | SpaceWire port datarate: 1=single, 2=double,... |
| config.transmitter.frequency | uint32_t | SpaceWire transmitter frequency in Hz |
| internal.linkState | int32_t | Link state |
| internal.txDAddr | uint32_t | Data address for the scheduled dma engine transfer |
| internal.txDLength | uint32_t | Data length for the scheduled dma engine transfer |
| internal.txFlags | uint32_t | Flags for the scheduled dma engine transfer |
| internal.txHAddr | uint32_t | Header address for the scheduled dma engine transfer |
| internal.txType | uint8_t | Scheduled transmission type (dma engine/rmap) |
| internal.uplinkNsPerByte | uint32_t | Transmitter speed |
| irqCtrl | iref / <unknown> | Irq controller |
| memory | iref / <unknown> | Memory used for DMA accesses |
| object.timeSource | object | Time source object (a cpu or machine object) |
| pnp.bar | uint32_t | Pnp BAR |
| pnp.config | uint32_t | Pnp configuration |
| regs.clockDiv | uint32_t | Clock Divisor register |
| regs.control | uint32_t | Control register |
| regs.destKey | uint32_t | Destination Key register |
| regs.dmaControl | uint32_t | Dma control registers |
| regs.dmaRxDescTableAddr | uint32_t | Dma receive descriptor table address registers |
| regs.dmaRxMaxLen | uint32_t | Dma rx maximum length registers |
| regs.dmaTxDescTableAddr | uint32_t | Dma transmit descriptor table address registers |
| regs.nodeAddress | uint32_t | Node address register |

| Name | Type | Description |
|---|---|---|
| regs.statusIrqSrc | uint32_t | Status / Interrupt-source register |
| regs.time | uint32_t | Time register |
| spwUplink | [2 x iref / <unknown>] | SpaceWire devices connected to the port |

#### Interfaces

| Name | Type | Description |
|---|---|---|
| ApbIface | ApbIface | Apb interface |
| DeviceIface | DeviceIface | Device interface |
| MemAccessIface | MemAccessIface | Memory Access Interface |
| ResetIface | ResetIface | |
| SpwPortIface | SpwPortIface | SpaceWire ports interfaces |

#### Ports

| Prop | Iface | Description |
|---|---|---|
| - | - | - |

## 13.3. Limitations

The following limitations/deviations from real hardware are known to exist with this model:

- No spill is currently not implemented
- Althougth the device already provides two spacewire ports, dual port is not yet implemented. This correspond to a device where the port VHDL parameter is set to 1. Therefore, in the control register, PO will be 0 and PS / NP bit are not available. Let us know if you need this feature implemented.
- The link interface currently effectively uses only ErrorReset, Ready, Connecting and Run states. Therefore, those are the only values that will be visible on the status register.
- RMAPEN signals not available

## 13.4. Examples

This example shows how to create two Grspw1 devices and connect them.

```
import BusModels
import TEMUGrspw1
object-create class=Grspw1 name=grspw0
```

```
object-create class=Grspw1 name=grspw1
spw-connect port1=grspw0:SpwPortIface[0] port2=grspw1:SpwPortIface[0]
```

# Chapter 14. GRSPW2

The Grspw2 model is part of the GRLIB IP library feature. It is available in the Grspw2 plugin.

## 14.1. Loading the Plugin

```
import Grspw2
```

## 14.2. Configuration

To work correctly, the device should be connected to an interrupt controller, the memory and another SpaceWire device.

There are several configuration parameters in the GrSpw2 device, summarized in the following table:

| Name | Description |
|---|---|
| config.infiniteSpeed | With this set, messages are sent immediately instead of being scheduled for the future based on the message length. This is the default option. |
| config.transmitter.frequency | Specify the SpaceWire transmitter frequency in Hz. Affects transfer speed when infinite speed is disabled. |
| config.transmitter.dataRate | SpaceWire port datarate: 1=single, 2=double, etc. Affects transfer speed when infinite speed is disabled. |
| config.dma.rxdescnum | Specifies the amount of rx description (0=128, 1=256, 2=512, 3=1024). This affect the regs.dmaRxDescTableAddr |
| config.dma.txdescnum | Specifies the amount of tx descriptors (0=64, 1=128, 2=256, 3=512). This affect the regs.dmaTxDescTableAddr |
| config.interrupt | Influences the interrupt that is raised with the IRQ controller (setting this property also updates the APB PnP info). |
| config.realCrcCheck | Set to use real crc check instead of packet crc flags. Real crc costs in terms of performance. |

### 14.2.1. Attributes

#### Properties

| Name | Type | Description |
|---|---|---|
| config.dma.rxdescnum | uint8_t | Number of rx descriptors |
| config.dma.txdescnum | uint8_t | Number of tx descriptors |
| config.infiniteSpeed | uint8_t | Set to use infinite speed for transfers. |
| config.interrupt | uint8_t | The interrupt index |
| config.realCrcCheck | uint8_t | Set to use real crc check instead of packet crc flags |
| config.transmitter.dataRate | uint8_t | SpaceWire port datarate: 1=single, 2=double,... |
| config.transmitter.frequency | uint32_t | SpaceWire transmitter frequency in Hz |
| internal.linkState | int32_t | Link state |
| internal.txCurrChan | uint8_t | Channel scheduled for transmission |
| internal.txDAddr | uint32_t | Data address for the scheduled dma engine transfer |
| internal.txDLength | uint32_t | Data length for the scheduled dma engine transfer |
| internal.txFlags | uint32_t | Flags for the scheduled dma engine transfer |
| internal.txHAddr | uint32_t | Header address for the scheduled dma engine transfer |
| internal.txType | uint8_t | Scheduled transmission type (dma engine/rmap) |
| internal.uplinkNsPerByte | uint32_t | Transmitter speed |
| irqCtrl | iref / <unknown> | Irq controller |
| memory | iref / <unknown> | Memory used for DMA accesses |
| object.timeSource | object | Time source object (a cpu or machine object) |
| pnp.bar | uint32_t | Pnp BAR |
| pnp.config | uint32_t | Pnp configuration |
| regs.clockDiv | uint32_t | Clock Divisor register |
| regs.control | uint32_t | Control register |
| regs.destKey | uint32_t | Destination Key register |
| regs.dmaAddr | [4 x uint32_t] | Dma address registers |

| Name | Type | Description |
|---|---|---|
| regs.dmaControl | [4 x uint32_t] | Dma control registers |
| regs.dmaRxDescTableAddr | [4 x uint32_t] | Dma receive descriptor table address registers |
| regs.dmaRxMaxLen | [4 x uint32_t] | Dma rx maximum length registers |
| regs.dmaTxDescTableAddr | [4 x uint32_t] | Dma transmit descriptor table address registers |
| regs.nodeAddress | uint32_t | Node address register |
| regs.statusIrqSrc | uint32_t | Status / Interrupt-source register |
| regs.time | uint32_t | Time register |
| spwUplink | [2 x iref / <unknown>] | SpaceWire devices connected to the port |

### Interfaces

| Name | Type | Description |
|---|---|---|
| ApbIface | ApbIface | Apb interface |
| DeviceIface | DeviceIface | Device interface |
| MemAccessIface | MemAccessIface | Memory Access Interface |
| ResetIface | ResetIface | |
| SpwPortIface | SpwPortIface | SpaceWire ports interfaces |

### Ports

| Prop | Iface | Description |
|---|---|---|
| - | - | - |

## 14.3. Limitations

The following deviations from real hardware are known to exist with this model:

- Althougth the device already provides two ports, dual port is not yet implemented. Let us know if you need this feature implemented.
- The link interface currently effectively uses only ErrorReset, Ready, Connecting and Run states. Therefore, those are the only values that will be visible on the status register.
- RMAPEN and PNPEN signals not available

## 14.4. Examples

This example shows how to create two Grspw2 devices and connect them.

```
import BusModels
import Grspw2
object-create class=Grspw2 name=grspw0
object-create class=Grspw2 name=grspw1
spw-connect port1=grspw0:SpwPortIface[0] port2=grspw1:SpwPortIface[0]
```

# Chapter 15. IRQMP

The IrqMP is part of the GRLIB device library from Gaisler. It is a multiprocessor capable interrupt controller.

The controller supports among things the routing of interrupts to different processor cores, and also broadcasted interrupts.

## 15.1. Loading the Plugin

```
import IrqMp
```

## 15.2. Configuration

**config.nCpu**

Number of processors supported.

**config.enExtIrq**

Enable extended IRQs.

**pnp.config**

Plug and play configuration word for APB plug-and-play.

**cpu**

Up to 16 CPUs supported. IfaceRef property should be connected to the different CPUs.

### 15.2.1. Attributes

#### Properties

| Name | Type | Description |
|---|---|---|
| broadcast | uint32_t | |
| config.enExtIrq | uint8_t | |
| config.logInterrupts | uint8_t | |
| config.nCpu | uint8_t | |
| config.traceReads | uint8_t | |
| config.traceWrites | uint8_t | |
| cpu | [16 x iref / <unknown>] | |
| extIntAck | [16 x uint32_t] | |
| force | [16 x uint32_t] | |

| Name | Type | Description |
|---|---|---|
| irqClear | uint32_t | |
| irqCtrl | [16 x iref / <unknown>] | |
| irqForce0 | uint32_t | |
| irqLevel | uint32_t | |
| irqPending | uint32_t | |
| mask | [16 x uint32_t] | |
| mpStatus | uint32_t | |
| object.timeSource | object | Time source object (a cpu or machine object) |
| pnp.bar | uint32_t | |
| pnp.config | uint32_t | |

#### Interfaces

| Name | Type | Description |
|---|---|---|
| ApbIface | ApbIface | |
| DeviceIface | DeviceIface | |
| IrqClientIface | IrqClientIface | uptree interrupt handlers (e.g. CPUs) |
| IrqIface | IrqCtrlIface | |
| MemAccessIface | MemAccessIface | |
| ResetIface | ResetIface | |

#### Ports

| Prop | Iface | Description |
|---|---|---|
| irqCtrl | IrqClientIface | irq port |

## 15.3. Limitations

The following deviations from real hardware are known to exist with this model:

- Broadcasted interrupts are broadcasted at the current time to all CPUs, if it was triggered by a non-synchronised event, the interrupt is raised at different times on the different cores. Depending on the IRQ frequency and the configured quanta length, this may result in problems.

# Chapter 16. LEON2 SoC

The Leon2SoC class implements a model of the LEON2 on chip devices (i.e. memory controller, interrupt controller, UARTs and timers). The model must be combined with a LEON2 CPU to be really useful.

## 16.1. Loading the Plugin

```
import Leon2SoC
```

## 16.2. Configuration

### 16.2.1. Interrupt Delivery

Set the irqControl property to point out the processor's irq interface. The model will deliver normal SPARC interrupts (1 up to 15). The LEON2 also exports the IrqCtrlIface as IrqIface. IrqClientIface should be wired from the CPU the LEON2 model is connected to.

The IrqIface enables the use of external interrupts using the raise and lower functions. The LEON2 has 8 external IRQs mapped according to the following table (the mappings cannot be customised at present):

*Table 1. External to Internal IRQ Mapping*

| External | Internal (Sparc IRL) |
|---|---|
| 0 | 4 |
| 1 | 5 |
| 2 | 6 |
| 3 | 7 |
| 4 | 10 |
| 5 | 12 |
| 6 | 13 |
| 7 | 15 |

The rules for IRQ raising is controlled by the GPIO IRQ config registers (it is also possible to raise IRQs by setting and lowering GPIO pins).

### 16.2.2. UART Connections

The UARTs are connected to the destination using the uarta and uartb properties. For the remote end points, these should be connected to UartAIface and UartBIface.

### 16.2.3. Infinite UART Speed

The UARTs can run either at infinite speed, or at simulated real-time speed. This can be configured using the infiniteUartSpeed property. Set this property to non-zero to enable infinite UART speed.

Note that this controls the speed of both UARTs.

When infinite speed is enabled, bytes are emitted to the destination serial device as soon as they have been written by the OBSW.

### 16.2.4. GPIO

The GPIO support in the LEON2 model supports interrupt generation using the GPIO interface instead of the IRQ controller interface. Model implements both the GpioClientIface and a property with a GpioBusIface reference (called gpioBus). The GPIO bus connection is not mandatory to set. If it is set, writes to the GPIO data register's out bits will be forwarded over the GPIO port. Note that the LEON2 only have 16 GPIO pins.

Both the legacy multipin GpioBusIface and the new single pin SignalIface are supported. The model will prioritise the legacy interface for backwards compatibility. If you wish to use the SignalIface interface you should not set the gpioBus property.

### 16.2.5. Caches

The LEON2 SoC can act as a cache controller. That means that a cache model can notify the SoC about when it starts an evict/flush operation. The controller will also notify any connected caches about enabling, disabling and freezing events happening.

The cache parameters in the cache control register and the product configuration register are set automatically when connecting the dCache and iCache interface references to conforming objects.

When connecting the cache references, make sure the caches are configured before they are connected.

The caches that these interface references are connected to should normally be compliant with the supported LEON2 cache parameters. That is, there is a limitation on the sizes, lines and ways.

While the model does a best effort in trying to report errors when a miss-configured cache model is supplied, take care to ensure that the model is correctly configured.

### 16.2.6. Attributes

#### Properties

| Name | Type | Description |
|---|---|---|
| ahbfailaddr | uint32_t | Fail address register |
| ahbstat | uint32_t | Fail status register |

| Name | Type | Description |
|---|---|---|
| behaviour | uint8_t | Set to 1 for COLE mode |
| cachectrl | uint32_t | Cache control register |
| cpu | iref / CpuIface | CPU to control with powerdown |
| dCache | iref / <unknown> | |
| gpioBus | iref / <unknown> | |
| gpioIrqLevel | uint32_t | |
| gpioIrqMask | uint32_t | |
| gpioIrqPolarity | uint32_t | |
| gpiodir | uint32_t | I/O port direction register |
| gpioinout | uint32_t | I/O port data register |
| gpioirqcfg | uint32_t | I/O port interrupt register 1 |
| gpioirqcfg2 | uint32_t | I/O port interrupt register 2 |
| iCache | iref / <unknown> | |
| infiniteUartSpeed | uint32_t | |
| irqControl | iref / IrqCtrlIface | Next level IRQ controller object (e.g. CPU) |
| irqclear | uint32_t | Interrupt clear register |
| irqforce | uint32_t | Interrupt force register |
| irqmask | uint32_t | Interrupt mask and priority register |
| irqpend | uint32_t | Interrupt pending register |
| leoncfg | uint32_t | Product configuration register |
| memcfg1 | uint32_t | Memory configration register 1 |
| memcfg2 | uint32_t | Memory configuration register 2 |
| memcfg3 | uint32_t | Memory configuration register 3 |
| memcfg4 | uint32_t | Memory configuration 4 (COLE) |
| memcfg5 | uint32_t | Memory configuration 5 (COLE) |
| mr | uint32_t | Map register (COLE) |
| object.timeSource | object | Time source object (a cpu or machine object) |
| outSignals | [8 x iref / SignalIface] | |

| Name | Type | Description |
|---|---|---|
| powerdown | uint32_t | Idle register |
| presccntr | uint32_t | Prescaler counter register |
| prescrld | uint32_t | Prescaler reload register |
| timer1cntr | uint32_t | Timer 1 counter register |
| timer1ctrl | uint32_t | Timer 1 control register |
| timer1rld | uint32_t | Timer 1 reload register |
| timer2cntr | uint32_t | Timer 2 counter register |
| timer2ctrl | uint32_t | Timer 2 control register |
| timer2rld | uint32_t | Timer 2 reload register |
| uart1DatTxHold | uint32_t | UART1 data TX hold register |
| uart1DatTxShift | uint32_t | UART 1 data TX shift |
| uart1ctrl | uint32_t | UART 1 control register |
| uart1datrx | uint32_t | UART 1 RX data register |
| uart1scal | uint32_t | UART 1 scaler register |
| uart1stat | uint32_t | UART 1 status register |
| uart2DatTxHold | uint32_t | |
| uart2DatTxShift | uint32_t | |
| uart2ctrl | uint32_t | UART 2 control register |
| uart2datrx | uint32_t | UART 1 RX data register |
| uart2scal | uint32_t | UART 2 scaler register |
| uart2stat | uint32_t | UART 2 status register |
| uarta | iref / <unknown> | |
| uartb | iref / <unknown> | |
| watchdog | uint32_t | Watchdog register |
| writeprot1 | uint32_t | Write protection register 1 |
| writeprot2 | uint32_t | Write protection register 2 |
| writeprotstart1 | uint32_t | Write protection start address 1 |
| writeprotstart2 | uint32_t | Write protection start address 2 |
| writeprotstop1 | uint32_t | Write protection end address 1 |
| writeprotstop2 | uint32_t | Write protection end address 2 |

### Interfaces

| Name | Type | Description |
|---|---|---|
| DCacheCtrlIface | CacheCtrlIface | D-cache to control |
| DeviceIface | DeviceIface | |
| GpioClientIface | GpioClientIface | |
| ICacheCtrlIface | CacheCtrlIface | I-cache to control |
| IrqClientIface | IrqClientIface | IRQ acknowledgement (from CPU) |
| IrqIface | IrqCtrlIface | IRQ controller, post your IRQs here. |
| MemAccessIface | MemAccessIface | |
| ResetIface | ResetIface | |
| SignalIface | SignalIface | Incomming signals |
| UartAIface | SerialIface | UART A |
| UartBIface | SerialIface | UART B |

#### Ports

| Prop | Iface | Description |
|---|---|---|
| irqControl | IrqClientIface | Interrupt |
| uarta | UartAIface | uart a |
| uartb | UartBIface | uart b |

### 16.2.7. Registers

Register support is currently experimental!

#### Register Bank registers

##### Register memcfg1

Memory configration register 1

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| pbrdy | 0x40000000 | 0x0 | 0x0 | PROM area bus-ready enable |
| abrdy | 0x20000000 | 0x0 | 0x0 | Asynchronous bus ready |

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| iowdh | 0x18000000 | 0x0 | 0x0 | I/O bus width |
| iobrdy | 0x4000000 | 0x0 | 0x0 | I/O area bus ready enable |
| bexc | 0x2000000 | 0x0 | 0x0 | Bus error enable for RAM PROM and I/O access |
| iows | 0xf00000 | 0x0 | 0x0 | I/O waitstates |
| ioen | 0x80000 | 0x0 | 0x0 | I/O area enable |
| prwen | 0x800 | 0x0 | 0x0 | PROM write enable |
| prwdh | 0x300 | 0x0 | 0x0 | PROM width |
| prwws | 0xf0 | 0x0 | 0x0 | PROM write waitstates |
| prrws | 0xf | 0x0 | 0x0 | PROM read waitstates |

#### Register memcfg2

Memory configuration register 2

Cold reset value: 0x7

Warm reset value: 0x7

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| sdrref | 0x80000000 | 0x0 | 0x0 | SDRAM refresh |
| trp | 0x40000000 | 0x1 | 0x1 | SDRAM t_rp timing |
| trfc | 0x38000000 | 0x7 | 0x7 | SDRAM t_rfp timing |
| sdrcas | 0x4000000 | 0x1 | 0x1 | SDRAM CAS delay |
| sdrbs | 0x3800000 | 0x0 | 0x0 | SDRAM bank size |
| sdrcls | 0x600000 | 0x2 | 0x2 | SDRAM column size |
| sdrcmd | 0x180000 | 0x0 | 0x0 | SDRAM command |
| se | 0x4000 | 0x0 | 0x0 | SDRAM enable |
| si | 0x2000 | 0x0 | 0x0 | SDRAM disable |
| rambs | 0x1e00 | 0x0 | 0x0 | SRAM bank size |

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| rambrdy | 0x80 | 0x2 | 0x2 | SRAM area bus ready enable |
| ramrmw | 0x40 | 0x2 | 0x2 | SRAM read-modify-write |
| ramwdh | 0x30 | 0x2 | 0x2 | SRAM bus width |
| ramwws | 0xc | 0x0 | 0x0 | SRAM write waitstates |
| ramrws | 0x3 | 0x0 | 0x0 | SRAM read waitstates |

#### Register memcfg3

Memory configuration register 3

Cold reset value: 0x3

Warm reset value: 0x3

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| rfc | 0xc0000000 | 0x3 | 0x3 | Register file checkbits |
| me | 0x8000000 | 0x1 | 0x1 | Memory EDAC |
| srcrv | 0x7fff000 | 0x0 | 0x0 | SDRAM refresh counter reload value |
| wb | 0x800 | 0x0 | 0x0 | EDAC diagnostic write bypass |
| rb | 0x400 | 0x0 | 0x0 | EDAC diagnostic read |
| re | 0x200 | 0x0 | 0x0 | RAM EDAC enable |
| pe | 0x100 | 0x0 | 0x0 | PROM EDAC enable |
| tcb | 0xff | 0x0 | 0x0 | Test checkbits |

#### Register ahbfailaddr

Fail address register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| - | - | - | - | - |

#### Register ahbstat

Fail status register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| eed | 0x200 | 0x0 | 0x0 | EDAC-correctable error detected |
| hed | 0x100 | 0x0 | 0x0 | Hardware error detected |
| het | 0x80 | 0x0 | 0x0 | Hardware error type |
| hem | 0x78 | 0x0 | 0x0 | Hardware error module |
| hes | 0x7 | 0x0 | 0x0 | Hardware error size |

#### Register cachectrl

Cache control register

Cold reset value: 0x3

Warm reset value: 0x3

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| drepl | 0xc0000000 | 0x3 | 0x3 | Data cache replacement policy |
| irepl | 0x30000000 | 0x3 | 0x3 | Instruction cache replacement policy |
| isets | 0xc000000 | 0x3 | 0x3 | Instruction cache associativity |
| dsets | 0x1000000 | 0x1 | 0x1 | Data cache associativity |

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| ds | 0x800000 | 0x0 | 0x0 | Data cache snoop enable |
| fd | 0x400000 | 0x0 | 0x0 | Flush data cache |
| fi | 0x200000 | 0x0 | 0x0 | Flush instruction cache |
| cpc | 0x180000 | 0x2 | 0x2 | Cache parity bits |
| cptb | 0x60000 | 0x3 | 0x3 | Cache parity test bits |
| ib | 0x10000 | 0x3 | 0x3 | Instruction burst fetch |
| ip | 0x8000 | 0x3 | 0x3 | Instruction cache flush pending |
| dp | 0x4000 | 0x0 | 0x0 | Data cache flush pending |
| ite | 0x3000 | 0x0 | 0x0 | Instruction cache tag error counter |
| ide | 0xc00 | 0x0 | 0x0 | Instruction cache data error counter |
| dte | 0x300 | 0x0 | 0x0 | Data cache tag error counter |
| dde | 0xc0 | 0x0 | 0x0 | Data cache data error counter |
| df | 0x20 | 0x0 | 0x0 | Data cache freeze on interrupt |
| if | 0x10 | 0x0 | 0x0 | Instruction cache freeze on interrupt |
| dcs | 0xc | 0x0 | 0x0 | Data cache state |
| ics | 0x3 | 0x0 | 0x0 | Instruction cache state |

#### Register powerdown

Idle register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| - | - | - | - | - |

#### Register writeprot1

Write protection register 1

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| en | 0x80000000 | 0x0 | 0x0 | Enable |
| bp | 0x40000000 | 0x0 | 0x0 | Block protect |
| tag | 0x1fff8000 | 0x0 | 0x0 | Address tag |
| mask | 0x3fff | 0x0 | 0x0 | Address mask |

#### Register writeprot2

Write protection register 2

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| en | 0x80000000 | 0x0 | 0x0 | Enable |
| bp | 0x40000000 | 0x0 | 0x0 | Block protect |
| tag | 0x1fff8000 | 0x0 | 0x0 | Address tag |
| mask | 0x3fff | 0x0 | 0x0 | Address mask |

#### Register writeprotstart1

Write protection start address 1

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| start | 0x3ffffffc | 0x0 | 0x0 | Start address |
| bp | 0x2 | 0x0 | 0x0 | Block protect |

#### Register writeprotstart2

Write protection start address 2

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| start | 0x3ffffffc | 0x0 | 0x0 | Start address |
| bp | 0x2 | 0x0 | 0x0 | Block protect |

#### Register writeprotstop1

Write protection end address 1

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| end | 0x3ffffffc | 0x0 | 0x0 | End address |
| us | 0x2 | 0x0 | 0x0 | User mode |
| su | 0x1 | 0x0 | 0x0 | Supervisor mode |

#### Register writeprotstop2

Write protection end address 2

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| end | 0x3ffffffc | 0x0 | 0x0 | End address |
| us | 0x2 | 0x0 | 0x0 | User mode |
| su | 0x1 | 0x0 | 0x0 | Supervisor mode |

#### Register leoncfg

Product configuration register

Cold reset value: 0x7

Warm reset value: 0x7

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| mmu | 0x80000000 | 0x0 | 0x0 | Memory management unit |
| dsu | 0x40000000 | 0x1 | 0x1 | Debug support unit |
| sdrctrl | 0x20000000 | 0x1 | 0x1 | SDRAM controller |
| wtpnb | 0x1c000000 | 0x4 | 0x4 | IU watchpoints |
| imac | 0x2000000 | 0x0 | 0x0 | UMAC/SMAC instructions |
| nwin | 0x1f00000 | 0x7 | 0x7 | IU register file windows |
| icsz | 0xe0000 | 0x3 | 0x3 | Instruction cache set size |
| ilsz | 0x18000 | 0x3 | 0x3 | Instruction cache line size |
| dcsz | 0x7000 | 0x3 | 0x3 | Data cache set size |
| dlsz | 0xc00 | 0x2 | 0x2 | Data cache line size |
| divinst | 0x200 | 0x1 | 0x1 | UDIV/SDIV instructions |
| mulinst | 0x100 | 0x1 | 0x1 | UMUL/SMUL instructions |
| wdog | 0x80 | 0x1 | 0x1 | Watchdog |
| memstat | 0x40 | 0x1 | 0x1 | Memory status and address failing register |
| fpu | 0x30 | 0x1 | 0x1 | FPU type |
| pci | 0xc | 0x1 | 0x1 | PCI core type |
| wprt | 0x3 | 0x1 | 0x1 | Write protections |

#### Register timer1cntr

Timer 1 counter register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| - | - | - | - | - |

#### Register timer1rld

Timer 1 reload register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| - | - | - | - | - |

#### Register timer1ctrl

Timer 1 control register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| ld | 0x4 | 0x0 | 0x0 | Load counter |
| rl | 0x2 | 0x0 | 0x0 | Reload counter |
| en | 0x1 | 0x0 | 0x0 | Enable counter |

#### Register watchdog

Watchdog register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| - | - | - | - | - |

#### Register timer2cntr

Timer 2 counter register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| - | - | - | - | - |

#### Register timer2rld

Timer 2 reload register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| - | - | - | - | - |

#### Register timer2ctrl

Timer 2 control register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| ld | 0x4 | 0x0 | 0x0 | Load counter |
| rl | 0x2 | 0x0 | 0x0 | Reload counter |
| en | 0x1 | 0x0 | 0x0 | Enable counter |

#### Register presccntr

Prescaler counter register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| cnt | 0x3ff | 0x0 | 0x0 | Prescaler counter value |

#### Register prescrld

Prescaler reload register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| rv | 0x3ff | 0x0 | 0x0 | Prescaler reload value |

#### Register uart1datrx

UART 1 RX data register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| rtd | 0xff | 0x0 | 0x0 | Received/transmit data |

#### Register uart1stat

UART 1 status register

Cold reset value: 0x1

Warm reset value: 0x1

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| fe | 0x40 | 0x0 | 0x0 | Framing error |
| pe | 0x20 | 0x0 | 0x0 | Parity error |
| ov | 0x10 | 0x0 | 0x0 | Overrun |
| br | 0x8 | 0x0 | 0x0 | Break received |
| th | 0x4 | 0x1 | 0x1 | Transmitter hold register empty |
| ts | 0x2 | 0x1 | 0x1 | Transmitter shift register empty |
| dr | 0x1 | 0x0 | 0x0 | Data ready |

#### Register uart1ctrl

UART 1 control register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| ec | 0x100 | 0x0 | 0x0 | External clock |
| lb | 0x80 | 0x0 | 0x0 | Loop back |
| fl | 0x40 | 0x0 | 0x0 | Flow control |
| pe | 0x20 | 0x0 | 0x0 | Parity enable |
| ps | 0x10 | 0x0 | 0x0 | Parity select |
| ti | 0x8 | 0x0 | 0x0 | Transmitter interrupt enable |
| ri | 0x4 | 0x0 | 0x0 | Receiver interrupt enable |
| te | 0x2 | 0x0 | 0x0 | Transmitter enable |
| re | 0x1 | 0x0 | 0x0 | Receiver enable |

#### Register uart1scal

UART 1 scaler register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| - | - | - | - | - |

#### Register uart2datrx

UART 1 RX data register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| rtd | 0xff | 0x0 | 0x0 | Received/transmit data |

#### Register uart2stat

UART 2 status register

Cold reset value: 0x1

Warm reset value: 0x1

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| fe | 0x40 | 0x0 | 0x0 | Framing error |
| pe | 0x20 | 0x0 | 0x0 | Parity error |
| ov | 0x10 | 0x0 | 0x0 | Overrun |
| br | 0x8 | 0x0 | 0x0 | Break received |
| th | 0x4 | 0x1 | 0x1 | Transmitter hold register empty |
| ts | 0x2 | 0x1 | 0x1 | Transmitter shift register empty |
| dr | 0x1 | 0x0 | 0x0 | Data ready |

#### Register uart2ctrl

UART 2 control register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| ec | 0x100 | 0x0 | 0x0 | External clock |
| lb | 0x80 | 0x0 | 0x0 | Loop back |
| fl | 0x40 | 0x0 | 0x0 | Flow control |
| pe | 0x20 | 0x0 | 0x0 | Parity enable |
| ps | 0x10 | 0x0 | 0x0 | Parity select |
| ti | 0x8 | 0x0 | 0x0 | Transmitter interrupt enable |
| ri | 0x4 | 0x0 | 0x0 | Receiver interrupt enable |
| te | 0x2 | 0x0 | 0x0 | Transmitter enable |
| re | 0x1 | 0x0 | 0x0 | Receiver enable |

#### Register uart2scal

UART 2 scaler register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| - | - | - | - | - |

#### Register irqmask

Interrupt mask and priority register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| ilevel_io7 | 0x80000000 | 0x0 | 0x0 | Interrupt level |
| ilevel_pci | 0x40000000 | 0x0 | 0x0 | Interrupt level |
| ilevel_io6 | 0x20000000 | 0x0 | 0x0 | Interrupt level |
| ilevel_io5 | 0x10000000 | 0x0 | 0x0 | Interrupt level |
| ilevel_dsu | 0x8000000 | 0x0 | 0x0 | Interrupt level |
| ilevel_io4 | 0x4000000 | 0x0 | 0x0 | Interrupt level |
| ilevel_timer2 | 0x2000000 | 0x0 | 0x0 | Interrupt level |
| ilevel_timer1 | 0x1000000 | 0x0 | 0x0 | Interrupt level |
| ilevel_io3 | 0x800000 | 0x0 | 0x0 | Interrupt level |
| ilevel_io2 | 0x400000 | 0x0 | 0x0 | Interrupt level |
| ilevel_io1 | 0x200000 | 0x0 | 0x0 | Interrupt level |
| ilevel_io0 | 0x100000 | 0x0 | 0x0 | Interrupt level |
| ilevel_uart1 | 0x80000 | 0x0 | 0x0 | Interrupt level |
| ilevel_uart2 | 0x40000 | 0x0 | 0x0 | Interrupt level |
| ilevel_amba | 0x20000 | 0x0 | 0x0 | Interrupt level |
| imask_io7 | 0x80000000 | 0x0 | 0x0 | Interrupt mask |
| imask_pci | 0x40000000 | 0x0 | 0x0 | Interrupt mask |
| imask_io6 | 0x20000000 | 0x0 | 0x0 | Interrupt mask |
| imask_io5 | 0x10000000 | 0x0 | 0x0 | Interrupt mask |
| imask_dsu | 0x8000000 | 0x0 | 0x0 | Interrupt mask |
| imask_io4 | 0x4000000 | 0x0 | 0x0 | Interrupt mask |
| imask_timer2 | 0x2000000 | 0x0 | 0x0 | Interrupt mask |
| imask_timer1 | 0x1000000 | 0x0 | 0x0 | Interrupt mask |
| imask_io3 | 0x800000 | 0x0 | 0x0 | Interrupt mask |

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| imask_io2 | 0x400000 | 0x0 | 0x0 | Interrupt mask |
| imask_io1 | 0x200000 | 0x0 | 0x0 | Interrupt mask |
| imask_io0 | 0x100000 | 0x0 | 0x0 | Interrupt mask |
| imask_uart1 | 0x80000 | 0x0 | 0x0 | Interrupt mask |
| imask_uart2 | 0x40000 | 0x0 | 0x0 | Interrupt mask |
| imask_amba | 0x20000 | 0x0 | 0x0 | Interrupt mask |

#### Register irqpend

Interrupt pending register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| - | - | - | - | - |

#### Register irqforce

Interrupt force register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| - | - | - | - | - |

#### Register irqclear

Interrupt clear register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| - | - | - | - | - |

#### Register gpioinout

I/O port data register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| - | - | - | - | - |

#### Register gpiodir

I/O port direction register

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| - | - | - | - | - |

#### Register gpioirqcfg

I/O port interrupt register 1

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| - | - | - | - | - |

#### Register gpioirqcfg2

I/O port interrupt register 2

Cold reset value: 0x0

Warm reset value: 0x0

| Field | Mask | Cold | Warm | Description |
|---|---|---|---|---|
| - | - | - | - | - |

## 16.3. Limitations

The Leon2 Device model simulates the AT697F chip. There are some deviations to the AT697E chip (e.g. the size of the counters). If you need the AT697E behaviour, please contact us for more info.

The following deviations from real hardware are known to exist, if you need the correct behaviour (or simulation of it, contact us for more info):

- No support for Ethernet at present
- No support for PCI at present
- The UARTs do not support external clocks.
- The UARTs do not support parity, framing errors and break signals.
- GPIO pin configurations are ignored for UARTs, the UARTs are assumed to be on separate dedicated I/O pins. However, a warning will be issued if the UART pins do not have the correct GPIO configuration.
- GPIO databus control is not supported (i.e. meddat and lowdat fields).
- Write protection registers have no effect
- Timer values are lazily computed on reads, the content in the case a timer is disabled is estimated on disabling time. This is in principle correct. However, the prescaler counter write has no effect, only the reload value has an effect when written. This may cause an offset of 1024 cycles when re-enabling a timer.
- In general the MEMCFG registers are ignored

# Chapter 17. Machine

The machine class is used to assemble and group related processors in machines. The machine class is intended to be used for SMP and multi-core systems. It provides the following capabilities:

1. A multi-CPU scheduler that executes all the CPUs in the machine in sequence (for a fixed time quanta).
2. A synchronised event queue. CPUs can post events in the next time quanta to be executed after all the processors have reached a specific time point.
3. A scheduling interface enabling the machine to be run for a time specified in seconds, not cycles.

Note that the machine class supports the scheduling of different CPUs with different clock frequencies.

Synchronised events are posted on a CPUs event queue by adding the flag TEMU_EVENT_SYNC to the posting function, this will bypass the CPU event queue and put it in the machine object's queue.

## 17.1. Configuration

### 17.1.1. Attributes

#### Properties

| Name | Type | Description |
|---|---|---|
| cpus | irefarray / <unknown> | Processors in the machine |
| currentCPU | iref / <unknown> | Current CPU |
| currentCPUIdx | int32_t | Current CPU Index |
| devices | irefarray / <unknown> | Devices to reset when machine is reset |
| object.timeSource | object | Time source object (a cpu or machine object) |
| quanta | uint64_t | Quanta length in nanoseconds |
| quantaEnd | uint64_t | End point of current quanta in nanoseconds |
| quantaStart | uint64_t | Quanta start in nanoseconds |
| syncMask | uint64_t | Synchronised CPU mask |

#### Interfaces

| Name | Type | Description |
|---|---|---|
| EventIface | EventIface | |
| LegacyIface | LegacyIface | |
| MachineIface | MachineIface | |
| ObjectIface | ObjectIface | |
| ResetIface | ResetIface | |

### Ports

| Prop | Iface | Description |
|---|---|---|
| - | - | - |

## 17.2. Limitations

- The machine class cannot have more than 64 CPU cores connected.

# Chapter 18. MEC

The MEC (Memory Controller) device is used with the ERC32 processor. The device provides two UARTs, two timers and an interrupt interface. The interrupt interface allows for the raising and lowering of the 5 external interrupts provided by the ERC32 (IRQ 0 through (including) 4). The device model takes care of converting these to the relevant internal interrupts (i.e. SPARC IRQs 2,3,10,11 and 14). When raising (or lowering) a MEC interrupt you need to use numbers 0-4.

## 18.1. Loading the Plugin

```
import Mec
```

## 18.2. Configuration

### 18.2.1. Interrupt Delivery

The property irqControl should be connected to the device which the MEC raises interrupts on, this is normally a CPU object. The connection should be made to the CPU-object's interface of type IrqIface. Note that the CPU must support interrupts 1 through 15, this is in general case correct for SPARC based processors, but other CPUs may not be compatible.

### 18.2.2. UART Connections

Two serial interfaces exist, the UartAIface and the UartBIface, these can be connected to in order to receive data from remote serial port terminals (i.e. this is the RX direction). The uarta and uartb properties can be used to connect the TX direction of the UARTs.

### 18.2.3. Infinite UART Speed

Set config.infiniteUartSpeed to nonzero to enable infinite speed on the Tx channels. With infinite speed, a written byte is immediately forwarded to the destination device, with limited UART speed (the variable being zero) the timing due to UART scaler bits (upper 8 bits of the MecCtrlReg) will be simulated, leading to realistic byte rates over the serial port device. Note that individual bits are not transmitted only the bytes.

### 18.2.4. Attributes

#### Properties

| Name | Type | Description |
|---|---|---|
| accessProtSegment1Base | uint32_t | |
| accessProtSegment1End | uint32_t | |
| accessProtSegment2Base | uint32_t | |

| Name | Type | Description |
|---|---|---|
| accessProtSegment2End | uint32_t | |
| config.infiniteUartSpeed | uint32_t | |
| cpu | iref / <unknown> | |
| errorAndResetStatus | uint32_t | |
| failingAddr | uint32_t | |
| gpiConfig | uint32_t | |
| gpiData | uint32_t | |
| gptCounter | uint32_t | |
| gptCounterProgramReg | uint32_t | |
| gptScaler | uint32_t | |
| gptScalerProgramReg | uint32_t | |
| ioConfig | uint32_t | |
| irqClear | uint32_t | |
| irqControl | iref / <unknown> | |
| irqForce | uint32_t | |
| irqMask | uint32_t | |
| irqPending | uint32_t | |
| irqShape | uint32_t | |
| mecCtrl | uint32_t | |
| memoryConfig | uint32_t | |
| object.timeSource | object | Time source object (a cpu or machine object) |
| outSignals | [8 x iref / SignalIface] | |
| powerDown | uint32_t | |
| rtcCounter | uint32_t | |
| rtcCounterProgramReg | uint32_t | |
| rtcScaler | uint32_t | |
| rtcScalerProgramReg | uint32_t | |
| softwareReset | uint32_t | |
| systemFaultStatus | uint32_t | |
| testControl | uint32_t | |
| timerControl | uint32_t | |

| Name | Type | Description |
|---|---|---|
| uartChanARxTx | uint32_t | |
| uartChanBRxTx | uint32_t | |
| uartStatus | uint32_t | |
| uarta | iref / <unknown> | |
| uartb | iref / <unknown> | |
| waitStateConfig | uint32_t | |
| wdogProgAndTimeoutAck | uint32_t | |
| wdogTrapDoorSet | uint32_t | |

### Interfaces

| Name | Type | Description |
|---|---|---|
| DeviceIface | DeviceIface | |
| IrqClientIface | IrqClientIface | |
| IrqIface | IrqIface | |
| MemAccessIface | MemAccessIface | |
| ResetIface | ResetIface | |
| SignalIface | SignalIface | Incomming signals |
| UartAIface | SerialIface | |
| UartBIface | SerialIface | |

### Ports

| Prop | Iface | Description |
|---|---|---|
| irqControl | IrqClientIface | uart a |
| uarta | UartAIface | uart a |
| uartb | UartBIface | uart b |

## 18.3. Notes

The MEC sets the interrupt pending register bit when an interrupt is raised even when the interrupt is masked. The mask is only applied when evaluating whether to raise an IRQ with the CPU.

## 18.4. Limitations

The following deviations from real hardware are known to exist, if you need the correct behaviour

(or simulation of it, contact us for more info):

- The UARTs do not support external (watchdog) clocks.
- The UARTs do not support parity, framing errors, break signals or stop bit configuration (although the transmission times are computed based on stop bit count and parity bit embedding).
- Write protection registers have no effect
- Timer values are lazily computed on reads, the content in the case a timer is disabled is estimated on disabling time. This is in principle correct. However, the prescaler counter write has no effect, only the reload value has an effect when written. This may cause an offset of 1024 cycles when re-enabling a timer.

# Chapter 19. MIL-STD-1553

This document describes the TEMU MIL-STD-1553 bus model and its interfaces. The MIL-STD-1553 standard is often referred to as simply milbus or 1553.

The 1553 protocol is described in detail in the well known "MIL-STD-1553 Tutorial" document from AIM GmbH (formerly published by Condor). It is recommended that persons involved with modelling bus controllers and remote terminals keep a copy of that document at close hand.

The TEMU support for the 1553 protocol consist of a bus interface (Mil1553BusIface), a bus model (MilStd1553Bus) and a bus client interface (Mil1553DevIface).

This approach enables the user to not only implement remote terminal models, but also to implement their own bus models would the bundled one not be found suitable (e.g. if the user have existing remote terminal models that must be integrated with specific interfaces).

The most common task for the end user will normally be to implement remote terminal models, but bus controllers are also possible as they use the same interface.

## 19.1. Bus Model

The 1553 bus model is available as a class with the name MilStd1553Bus in the TEMU "BusModels" plugin.

## 19.2. Configuration

The bus model is configured using the Mil1553BusIface. The main work is to call the connect function to insert a remote terminal at the given subaddress.

SetBusController should be called to set the current bus controller (note, this can be done at runtime).

The construction of a network with 1553 devices are simplified by using the following commands in the command line interface:

- mil-std-1553-connect bus=b rt=rt addr=1
- mil-std-1553-disconnect bus=b addr=1
- mil-std-1553-setbc bus=b bc=bc

### 19.2.1. Attributes

#### Properties

| Name | Type | Description |
|---|---|---|
| bc | iref / <unknown> | |

| Name | Type | Description |
|---|---|---|
| devices | [32 x iref / <unknown>] | |
| lastCmd | uint16_t | |
| object.timeSource | object | Time source object (a cpu or machine object) |
| receiverRT | int8_t | |
| stats.lastReportSentWords | uint64_t | |
| stats.sentWords | uint64_t | |
| transmitterRT | int8_t | |

#### Interfaces

| Name | Type | Description |
|---|---|---|
| Mil1553BusIface | Mil1553BusIface | |

#### Ports

| Prop | Iface | Description |
|---|---|---|
| - | - | - |

### 19.2.2. Attributes

#### Properties

| Name | Type | Description |
|---|---|---|
| bus | object | Bus object to monitor. |
| object.timeSource | object | Time source object (a cpu or machine object) |
| statPeriod | double | Statistics report period in seconds, set to positive enables reports. |

#### Interfaces

| Name | Type | Description |
|---|---|---|
| - | - | - |

#### Ports

| Prop | Iface | Description |
|---|---|---|
| - | - | - |

### 19.2.3. Notifications

The default TEMU milbus model issues the following notifications:

| Name | Description | Param Type |
|---|---|---|
| temu.mil1553Stat | Statistics notification. | temu_Mil1553Stats* |
| temu.mil1553Send | Valid message in transit. | temu_Mil1553Msg* |

The statistics notification is issued when calling the reportStats function in the bus interface. The user can call this function from a timed event handler if needed. Another interesting calling point is to force statistics reporting at a PPS tick, i.e. a PPS device issues the call to the milbus object to report the statistics, and can attempt to post other events at minor cycle intervals for example. This way the stat event can be used to monitor whether the system keeps the milbus budget.

The send notification receives a pointer with the actual message in transit, but before it has been delivered to the remote terminal (but after the bus object has rejected any messages transmitted illegally). The notification handler is free to modify the message, for example it is possible to set the Err field in the message struct to inject a transfer error, the RT can then set the message error bit in the status word.

## 19.3. Limitations

The bus object does not support bus monitors in the normal sense, however, it is possible to turn on the temu.mil1553Send notification and listen in on all traffic using this notification interface.

For the command line support, only models with one and only one device interface with the name Mil1553DevIface is supported. This may change in the future.

## 19.4. API

# Chapter 20. Interfaces

```
typedef struct temu_Mil1553BusIface {
  void (*connect)(void *Bus, int Subaddr, temu_Mil1553DevIfaceRef Device);
  void (*disconnect)(void *Bus, int Subaddr);
  void (*reportStats)(void *Bus);
  void (*send)(void *Bus, void *Sender, temu_Mil1553Msg *Msg);

  // Controls whether events should be issued at send calls
  void (*enableSendEvents)(void *Bus);
  void (*disableSendEvents)(void *Bus);
  void (*setBusController)(void *Bus, temu_Mil1553DevIfaceRef Device);
} temu_Mil1553BusIface;
```

```
typedef struct temu_Mil1553DevIface {
  void (*connected)(void *Device, temu_Mil1553BusIfaceRef Bus, int SubAddr);
  void (*disconnected)(void *Device, temu_Mil1553BusIfaceRef Bus, int SubAddr);
  void (*receive)(void *Device, temu_Mil1553Msg *Msg);
} temu_Mil1553DevIface;
```

## 20.1. Writing Clients

### Bus Controllers and Remote Terminals

Bus controllers and remote terminals can be implemented using the Mil1553BusIface interface. This interface is defined in "temu-c/Bus/MilStd1553.h".

The interface consist of the connected, disconnected and receive functions. These are all mandatory and they are called whenever a virtual cable is connected and disconnected, or when a 1553 bus message is received.

A remote terminal needs to know about the bus it is connected to so it can use the send function in the Mil1553BusIface interface.

Do not call the bus send function from the device receive function, doing so will result in undefined behaviour. If a response is to be issued due to handling of a receive, ensure that an event is posted on the model's event queue source.

The TEMU 1553 API follows the standard fairly well and subdivides 1553 transactions in phases which are command, data, status and mode command phases. To send a receive command, the bus controller will first send a message of the type teMT_Cmd, followed by a teMT_Data message. The remote terminal is then expected to respond with a teMT_Stat message. The remote terminal and bus controller model is responsible for issuing the different messages with delays. Delays can be computed using the temu_mil1553TransferTime() function.

Messages should be sent in whole when they are supposed to arrive. This means that the bus

controller model can immediately raise any needed interrupts when a message is complete.

The TEMU default 1553 bus model will print error messages if a remote terminal does not follow the 1553 protocol phases properly (e.g. sending a status response to a broadcast message).

```
void
receive(void *Device, temu_Mil1553Msg *Msg)
{
  MyRT *RT = (MyRT*)Device;
  //...
  // Start sending response
  temu_eventPostNanos(RT->Super.TimeSource, RT->TransferCompleteEvent,
                      temu_mil1553TransferTime(1), // One word for status message
                      teSE_Cpu);
}

void
transferComplete(temu_Event *Ev)
{
   MyRT *RT = (MyRT*)Ev->Obj;

   uint16_t Stat = computeStatWord(RT);
   temu_Mil1553Msg Msg = temu_mil1553CreateStatMsg(&Stat);

   RT->Bus.Iface->send(RT->Bus.Obj, RT, &Msg); // Send the message
}
```

### Bus Monitors

The 1553 bus interface does not support the implementation of bus monitors directly at this moment. The reason for this is that, the message notification interface already allows the system to inspect all the bus traffic executed. The notification interface can also be used to modify traffic in situ (e.g. to flip the error flags in the message object). Terma appreciates that there may be need for some users to support modelling of bus monitors, please contact Terma if this is needed.

## 20.2. Capture Device

TEMU is bundled with a MILBUS capture device that enables capturing of the bus traffic. There are three supported options for message capture:

- Logging command words issued to the TEMU log with partial decoding
- CSV output with command words and partial decodes of them
- PCAPNG file with all data transferred. File can be loaded in Wireshark if needed.

To create a logging capture device, create the bus capture instance using:

For logging: object-create class=MilStd1553BusCapturer name=milbus-cap0 \ args=fmt:log,bus:milbus0

For CSV output (into milbus0.csv): object-create class=MilStd1553BusCapturer name=milbus-cap0 \ args=fmt:csv,bus:milbus0

For PCAPNG output (into milbus0.pcapng): object-create class=MilStd1553BusCapturer name=milbus-cap0 \ args=fmt:pcapng,bus:milbus0

Do not forget to set the time source for the capture device: connect-timesource obj=milbus-cap0 ts=cpu0

While the logging and CSV modes should be clear enough, there are some notes to be providede regarding the PCAPNG format.

Firstly, the capture model captures logical units in the protocol, that is, command words are captured by themselves, as is status messages and data messages.

Secondly, the capture model use the flags in the frame block to mark where the data came from. That is, it flags unicast, and broadcast messages as such, and it also flags the direction as outbound for frames emitted by the BC (e.g. command words, mode codes, data sent to RTs etc) and inbound for data sent from RTs.

Thirdly, LINKTYPE_USER0 is used for the device type (there is no standardised milbus link type), this linktype is not supported directly by Wireshark, and a dissector needs to be implemented to make frames more human readable.

Due to these caveats, interpreting the 1553 protocol in Wireshark is a bit tricky, but in general, we can say that command sequences starts with outbound frames, which are followed by inbound frames. A dissector (or human viewing without a dissector) needs to be clever about decoding these frames and take into account the previous frames sent, it is likely also necessary in case of failed transfers to take into account such flags as well as buscontrollers tend to retry message transfers if they fail.

# Chapter 21. Serial Console

The serial console is a simple endpoint for serial traffic that you can connect a device's UART to. It echos received data to stdout and optionally logs the data in an unbounded log.

## 21.1. Loading the Plugin

```
import Console
```

## 21.2. API

There is a dedicated API for accessing the console log. Note that the functions are defined in libTEMUConsole.so.

This API does not work on macOS.

```c
// Include the Console API
#include "temu-c/Models/Console.h"

// These functions are defined in libTEMUConsole.so
uint64_t temu_consoleGetLineCount(void *Con);
const char* temu_consoleGetLine(void *Con, uint64_t Line);
```

## 21.3. Configuration

### 21.3.1. Creation

The Console class is defined in libTEMUConsole.so. The constructor takes no parameters.

### 21.3.2. Options

`config.caretControl` can be used to eliminate some VT100 characters that are printed to the console otherwise.

`config.recordTraffic` can be set to enable data recording in the console model, this data can then be extracted with the API.

### 21.3.3. Attributes

#### Properties

| Name | Type | Description |
|---|---|---|
| config.caretControl | uint8_t | |

| Name | Type | Description |
|---|---|---|
| config.outFile | cstring | File name to write TTY log to. |
| config.recordTraffic | uint8_t | |
| config.reformatNonPrintable | uint8_t | |
| lastByte | uint8_t | |
| object.timeSource | object | Time source object (a cpu or machine object) |
| outByte | uint8_t | |
| serial | iref / <unknown> | |

### Interfaces

| Name | Type | Description |
|---|---|---|
| LineDataLoggerIface | LineDataLoggerIface | |
| SerialIface | SerialIface | |

### Ports

| Prop | Iface | Description |
|---|---|---|
| serial | SerialIface | serial port |

## 21.4. Limitations

- The record buffer cannot be cleaned without deleting the console object.
- Caret control only omits caret sequences from being put on stdout (especially nice when booting Linux). It doesn't act on the sequences in any way at the moment e.g. a delete character will be ignored and not actually delete anything.
- The record buffer will not be snapshotted.

# Chapter 22. Serial Console UI

The serial console ui is a simple graphical endpoint for serial traffic that you can connect a device's UART to. It forks of a separate process which display a new window with the serial port output. This window also handles interactive input, meaning that you can for example type commands to a command line interface provided by the software running in the emulated environment.

The console window supports limited VT100 emulation.

## 22.1. Loading the Plugin

```
import ConsoleUI
```

## 22.2. Limitations

As with all other models, problems not listed here should be reported to Terma as they may indicate bugs in the software.

- The Console UI requires QT 4 to be installed (e.g. with your package manager) and any needed support libraries for QT. Thus the console in particular has a lot of extra dependencies over the rest of the emulator. If you are running this on specific systems and the console does not work, please report this to Terma.
- The console always do VT100 emulation, the emulation cannot be disabled.
- Only partial VT100 support exists. The supported CSIs include colors and cursor movements. Some CSIs may be missing.
- The console does not echo input back automatically, this is typically done by the remote serial end. Consequently, you will not see any characters if you type them in the console and the remote does not echo back.
- The console model will in normal mode poll the input at 100 Hz as the emulator at the moment of writing does not support the injection of asynchronous events. The 100 Hz polling is good enough for interactive use.

# Chapter 23. SpaceWire

TEMU provides support for SpaceWire based devices. It also provides helpful functions for RMAP commands decoding. The bus model interfaces are available in: `temu-c/Bus/Spacewire.h`. In addition to the interfaces a simple SpaceWire Router model is provided.

Spacewire is a point to point bus. Two devices can be connected directly while multiple devices can be connected through a Router. A SpaceWire Route receives a packet on a port and forward it to another, where the destination device is connected.

Spacewire uses wormhole routing. The sender device provides the list of addresses (each address is an 8-bit value) required to reach the destination. Each node in the middle is supposed to strip the first address and use it to select the port used to forward the packet.

## 23.1. API

The interesting interfaces are defined in the `temu-c/Bus/Spacewire.h` header.

```
typedef enum {
  teSMT_Data = 1,
  teSMT_Err = 2,
  teSMT_Time = 3,
} temu_SpwPacketType;

typedef struct temu_SpwPacket {
  temu_SpwPacketType MsgType;
  temu_Buff PktData;
  uint8_t Flags;
} temu_SpwPacket;

typedef enum {
  teSPWLS_ErrorReset = 1,
  teSPWLS_Ready = 2,
  teSPWLS_Started = 3,
  teSPWLS_Connecting = 4,
  teSPWLS_Run = 5
} temu_SpwLinkState;

struct temu_SpwPortIface {
  void (*receive)(void *Device, void *Sender, temu_SpwPacket *Pkt);
  void (*signalLinkStateChange)(void* Device, temu_SpwLinkState LinkState);
  temu_SpwLinkState (*getOtherSideLinkState)(void* Device);
  void (*connect)(void *Device, temu_SpwPortIfaceRef Dest);
  void (*disconnect)(void *Device);
  uint64_t (*timeToSendPacketNs)(void* Device, uint64_t PacketLength);
};
```

While the SpaceWire protocol is character based, to have better performances TEMU transfers full messages with a single call on the port interface. Example of messages are a data packet, an RMAP packet and a time code. Control characters like FCT (flow control) are abstracted away.

The SpaceWire packet structure is used to pass a packet between nodes. The MsgType field identifies if the packet is a timecode, a complete data packet (ending with EOP) or an incomplete data packet (ending with EEP). The PktData field contains the packet data or the time code value.

A TEMU buffer is used to hold the data. This data structure has been implemented to handle SpaceWire packets in a performant way. It allows to acquire a reference to a part of the original data so that a copy of data is not required for each node due to wormhole routing stripping. It also free the memory used to store the original message when no more references are active. This way, destination devices can maintain the data as long as needed without coping it.

SpaceWire links are full-duplex. The SpaceWire link is modeled by simply having each device implementing a port interface and holding a reference to other end port. This allows comunication in both directions simultaneously.

SpaceWire devices often have several connections port. The SpwPortIface is meant to be implement for each port a device intends to provide.

`temu-c/Bus/Spacewire.h` header also define functions to help decode RMAP packets:

| Name | Description |
|---|---|
| temu_spwRmapDecodePacket | Provided a SpaceWire Rmap packet attempts to decode it. |
| temu_spwRmapDecodeBuffer | Provided a buffer containing a SpaceWire Rmap packet attempts to decode it. |
| temu_spwRmapHeaderReplySize | Returns the total packet-size required to reply to the command. |
| temu_spwRmapEncodeReadReplyHeaderForPacket | Encodes the reply for a read command. |
| temu_spwRmapEncodeRmwHeaderForPacket | Encodes the reply for a rmw command. |
| temu_spwRmapEncodeWriteReplyHeaderForPacket | Encodes the reply for a write command. |
| temu_spwRmapCRCNextCode | Provided the previous calculated crc and a the current byte returns the next CRC value. |
| temu_spwRmapCRC | Calculates the CRC over the specified data. |

## 23.2. Limitations

The following deviations from real hardware are known to exist with this model:

- When two different devices try to access the same device the two accesses will happend

simultaneously. This should not be the case, the accesses should be sequential (the second device should wait for the bus to be free). This issue will be solved in the future when bus-reservation feature will be implemented.

# 23.3. Commands

The following commands are provided:

| Name | Description |
|---|---|
| spw-connect | Connect the two SpaceWire port interfaces provided as parameters |
| spw-disconnect | Disconnect the two SpaceWire port interfaces provided as parameters |

# 23.4. Models

## 23.4.1. SpwRouter

The SpwRouter class provides a simple SpaceWire Router that lets the user configure the mapping between the packet-address and the port that will be used to forward the packet. More advanced features like Group Adaptive Routing or Packet Distribution are not implemented.

## 23.4.2. Attributes

### Properties

| Name | Type | Description |
|---|---|---|
| internal.linkState | [32 x int32_t] | Holds the link state of the ports |
| object.timeSource | object | Time source object (a cpu or machine object) |
| ports | [32 x iref / <unknown>] | Connected SpaceWire devices |
| routingTable | [256 x uint8_t] | Configure packet-address/forwarding-port mapping |

### Interfaces

| Name | Type | Description |
|---|---|---|
| SpwPortIface | SpwPortIface | Input spacewire ports interfaces |

### Ports

| Prop | Iface | Description |
|---|---|---|
| - | - | - |

### 23.4.3. Attributes

#### Properties

| Name | Type | Description |
|---|---|---|
| count.Rx | uint32_t | Counter for received messages. |
| count.Tx | uint32_t | Counter for received messages. |
| enabled | uint8_t | Enable/Disable UDP. Required to change properties. |
| internal.linkState | int32_t | Holds the link state of the port |
| object.timeSource | object | Time source object (a cpu or machine object) |
| port | iref / <unknown> | Connected SpaceWire device. |
| protocolId | uint8_t | Protocol ID to be used. |
| rxUdpPort | uint16_t | Udp port used to receive. |
| targetAddr | [16 x uint8_t] | Addresses to use to forard a packet received via UDP. |
| targetAddrLength | uint8_t | Number of valid addresses in targetAddr array. |
| txHost | cstring | File name to write TTY log to. |
| txUdpPort | uint16_t | Udp port used to send. |

#### Interfaces

| Name | Type | Description |
|---|---|---|
| SpwPortIface | SpwPortIface | Input spacewire port interfaces |

#### Ports

| Prop | Iface | Description |
|---|---|---|
| - | - | - |

## 23.5. Examples

This example shows how to create a simple SpaceWire Router and a Grspw2 device and connect them.

```
import BusModels
import TEMUGrspw2
object-create class=Grspw2 name=grspw0
object-create class=SpwRouter name=spwRouter
spw-connect port1=grspw0:SpwPortIface[0] port2=spwRouter:SpwPortIface[0]
```

The next example shows how to implement a simple SpaceWire device

```
#include <stdint.h>
#include <stdio.h>
#include <string.h>

#include "temu-c/Support/Objsys.h"
#include "temu-c/Support/Attributes.h"
#include "temu-c/Support/Logging.h"
#include "temu-c/Bus/Spacewire.h"

typedef struct {
  temu_Object Super;

  int TransmitterDataRate;
  temu_SpwLinkState LinkState;
  temu_SpwPortIfaceRef Uplink;
} SpwDevice;

void*
create(const char *Name,
       int Argc TEMU_UNUSED,
       const temu_CreateArg *Argv TEMU_UNUSED)
{
  void *Obj = malloc(sizeof(SpwDevice));
  memset(Obj, 0, sizeof(SpwDevice));

  printf("Creating Object '%s'\n", Name);

  return Obj;
}

void
destroy(void *Obj)
{
  free(Obj);
}

static void
spwDeviceChangeLinkState(SpwDevice *Device, temu_SpwLinkState LinkState)
{
```

```
  Device->LinkState = LinkState;
  if ((Device->Uplink.Iface != NULL) && (Device->Uplink.Obj != NULL)) {
      Device->Uplink.Iface->signalLinkStateChange(
        Device->Uplink.Obj, LinkState);
  }
}

////////////////////////////////////////////////////////////////////////////
// SpwPortIface 0 implementation
////////////////////////////////////////////////////////////////////////////

static void
spwPortIfaceReceive0(void *Obj, void *Sender, temu_SpwPacket *Pkt)
{
  // Handle packet received.
  SpwDevice *Dev = (SpwDevice*)(Obj);
  temu_logInfo(Dev, "Received SpaceWire packet");
}

static void
spwPortIfaceSignalLinkStateChange0(void *Obj, temu_SpwLinkState LinkState)
{
  // The other side notified us that its link state changed.
  SpwDevice *Dev = (SpwDevice*)(Obj);
  temu_logInfo(Dev, "Other side link state changed");

  // Depending on the other side link state change update this
  // device link state.
}

static temu_SpwLinkState
spwPortIfaceGetOtherSideLinkState0(void *Obj)
{
  // Other side request this device state.
  SpwDevice *Dev = (SpwDevice*)(Obj);
  return (temu_SpwLinkState)Dev->LinkState;
}

static void
spwPortIfaceConnect0(void *Obj, temu_SpwPortIfaceRef PortIf)
{
  SpwDevice *Dev = (SpwDevice*)(Obj);
  Dev->Uplink = PortIf;

  // When two ports are connected the device goes to ready state.
  spwDeviceChangeLinkState(Dev, teSPWLS_Ready);
}

static void
```

```
spwPortIfaceDisconnect0(void *Obj)
{
  SpwDevice *Dev = (SpwDevice*)(Obj);
  Dev->Uplink.Iface = NULL;
  Dev->Uplink.Obj = NULL;

  // When two ports are diconnected the device goes to error reset state.
  spwDeviceChangeLinkState(Dev, teSPWLS_ErrorReset);
}

static uint64_t
spwPortIfaceTimeToSendPacketNs0(void* Obj, uint64_t PacketSize)
{
  SpwDevice *Dev = (SpwDevice*)(Obj);
  // Return the time required to transmit the packet through this port.
  return PacketSize / Dev->TransmitterDataRate;
}

temu_SpwPortIface SpwPortIface0 = {
  spwPortIfaceReceive0,
  spwPortIfaceSignalLinkStateChange0,
  spwPortIfaceGetOtherSideLinkState0,
  spwPortIfaceConnect0,
  spwPortIfaceDisconnect0,
  spwPortIfaceTimeToSendPacketNs0
};

TEMU_PLUGIN_INIT
{
  temu_Class *Cls = temu_registerClass("SpwDevice", create, destroy);

  // Reference to the port interface of the other end.
  temu_addProperty(Cls, "Uplink",
                   offsetof(SpwDevice, Uplink),
                   teTY_IfaceRef,
                   1, // Number of elements (1 = scalar)
                   NULL, NULL,
                   "Other end port interface");

  // Port interface.
  temu_addInterface(Cls, "SpwPortIface", "SpwPortIface", &SpwPortIface0,
                    0, "SpaceWire port interface");
}
```