TEMU
Model Reference

Version 3.0-pre, 2022-04-12

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 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.

Listing 1. AMBA PnP API

```
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. @AhbCtrl Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object

Commands

Name	Description
delete	Dispose instance of @AhbCtrl
new	Create new instance of AhbCtrl

Command new Arguments

Name	Type	Required	Description
name	string	yes	Name of object to create

2.2.2. AhbCtrl Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object
masters	[temu_IfaceRef; 64]/ <unknown>	AHB master devices.
slaves	[temu_IfaceRef; 64]/ <unknown>	AHB slave devices.

Interfaces

Name	Type	Description
DeviceIface	DeviceIface	

Name	Type	Description
MemAccessIface	MemAccessIface	
ResetIface	ResetIface	

Commands

Name	Description
delete	Dispose instance of AhbCtrl

2.2.3. @ApbCtrl Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object

Commands

Name	Description
delete	Dispose instance of @ApbCtrl
new	Create new instance of ApbCtrl

Command new Arguments

Name	Type	Required	Description
name	string	yes	Name of object to create

2.2.4. ApbCtrl Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component

Name	Type	Description
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object
pnp.bar	[uint32_t; 4]	
pnp.identReg	uint32_t	
pnp.userDef	[uint32_t; 3]	
slaves	[temu_IfaceRef; 512]/ <unknown>	APB slaves.

Interfaces

Name	Type	Description
AhbIface	AhbIface	
DeviceIface	DeviceIface	
MemAccessIface	MemAccessIface	
ResetIface	ResetIface	

Commands

Name	Description
delete	Dispose instance of ApbCtrl

2.3. Examples

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

Listing 2. Connecting AHB and APB 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=0xffff000 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.

Listing 3. Simple Device with APB PNP Support

```
#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. Bus Models

3.1. 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. 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 simple 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. Such a model would route frames using internal routing tables.

A smart CAN bus model will by definition be system specific. It's routing will depend on the system specific allocation of message IDs.

3.1.1. 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, the frame struct need to be transformed 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. If the device needs registers and MMIO handling, it tends to get more complex.

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.

3.1.2. 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.

3.1.3. 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.

3.1.4. SimpleCANBus Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name

Name	Type	Description
TimeSource	*void	Time source object
devices	temu_IfaceRefArray	CAN devices attached to bus
stats.lastReportSentBits	uint64_t	Statistics
stats.sentBits	uint64_t	Statistics

Interfaces

Name	Type	Description
CanBusIface	CanBusIface	CAN Bus Interface

Commands

Name	Description
delete	Dispose instance of SimpleCANBus

3.1.5. Examples

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

Listing 4. Connecting CAN Devices

```
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

Listing 5. Simple CAN Device Example

```
#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);

}
```

3.2. 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 promiscous 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 collission.

3.2.1. Connecting Devices

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

Example 1. Connect Syntax

```
ethlink0.connect device=phy0:PHYIface
```

Example 2. Disconnect Syntax

```
ethlink0.connect device=phy0:PHYIface
```

3.2.2. 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.

3.2.3. 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 auto-negotiation 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

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

3.2.4. Ethernet 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`.

3.2.5. Ethernet Link

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. TEMU 3 Command Syntax

```
ethlink0.enableCapture file="foo.pcap"
```

Example 4. Legacy Global Command (TEMU 2) Syntax

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

3.2.6. @EthernetLink Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object

Commands

Name	Description
delete	Dispose instance of @EthernetLink

Name	Description
new	Create new instance of EthernetLink

Command new Arguments

Name	Type	Required	Description
name	string	yes	Name of object to create

3.2.7. EthernetLink Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object

Interfaces

Name	Type	Description
EthernetIface	temu::EthernetIface	

Commands

Name	Description
connect	Connect device to ethernet link
delete	Dispose instance of EthernetLink
disconnect	Disconnect device from ethernet link
enableCapture	Enable capture to PCAPNG file

Command connect Arguments

Name	Type	Required	Description
device	interface	yes	Device to connect

Command disconnect Arguments

Name	Type	Required	Description
device	interface	yes	Device to connect

Command enableCapture Arguments

Name	Type	Required	Description
file	path	yes	Name of capure file.

3.2.8. Generic PHY

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.

3.2.9. @GenericPHY Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object

Commands

Name	Description
delete	Dispose instance of @GenericPHY
new	Create new instance of GenericPHY

Command new Arguments

Name	Type	Required	Description
name	string	yes	Name of object to create

3.2.10. GenericPHY Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object
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	temu_IfaceRef/ <unknown>	Ethernet link.
linkPartnerAbility	uint16_t	Link partner ability register
macDevice	temu_IfaceRef/ <unknown>	MAC device.
phyID	[uint16_t; 2]	Physical ID registers

Interfaces

Name	Type	Description
MDIOIface	temu::MDIOIface	
PHYIface	temu::PHYIface	

Commands

Name	Description
delete	Dispose instance of GenericPHY

3.2.11. MDIO Bus

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.

3.2.12. @MDIOBus Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object

Commands

Name	Description
delete	Dispose instance of @MDIOBus
new	Create new instance of MDIOBus

Command new Arguments

Name	Type	Required	Description
name	string	yes	Name of object to create

3.2.13. MDIOBus Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component

Name	Type	Description
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object
macDevice	temu_IfaceRef/ <unknown>	MAC controller.
phyDevices	[temu_IfaceRef; 32]/ <unknown>	MDIO interface of PHYs.

Interfaces

Name	Type	Description
MDIOIface	temu::MDIOIface	

Commands

Name	Description
delete	Dispose instance of MDIOBus

3.2.14. Implementing a MAC Model

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.

3.3. 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. The latter is of interest if the bundled model is found not suitable.

Most users will be interested in implementing remote terminal models, but bus controllers are also possible as they use the same interface.

3.3.1. Bus Model

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

3.3.2. Configuration

The bus model is configured using the `Mil1553BusIface`. This is done by calling the connect function in order to attach remote terminal at given subaddress.

`SetBusController` should be called to set the current bus controller

It is possible to set the current bus controller at runtime, this is useful for handovers.

The construction of a network with 1553 devices is 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

3.3.3. @MilStd1553Bus Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object

Commands

Name	Description
delete	Dispose instance of @MilStd1553Bus
new	Create new instance of MilStd1553Bus

Command new Arguments

Name	Type	Required	Description
name	string	yes	Name of object to create

3.3.4. MilStd1553Bus Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object
bc	temu_IfaceRef/ <unknown>	Bus controller.
devices	[temu_IfaceRef; 32]/ <unknown>	Remote terminals.
lastCmd	uint16_t	
receiverRT	int8_t	
stats.lastReportSentWords	uint64_t	
stats.sentWords	uint64_t	
transmitterRT	int8_t	

Interfaces

Name	Type	Description
Mil1553BusIface	Mil1553BusIface	

Commands

Name	Description
connect	Connect device to 1553 bus.
delete	Dispose instance of MilStd1553Bus
disconnect	Disconnect device from 1553 bus.
setBC	Set bus controller for 1553 bus.

Command connect Arguments

Name	Type	Required	Description
addr	int	yes	RT address.
rt	object	yes	Connect RT to bus.

Command disconnect Arguments

Name	Type	Required	Description
addr	int	yes	RT address.

Command setBC Arguments

Name	Type	Required	Description
bc	object	yes	Bus controller object.

3.3.5. @MilStd1553BusLogger Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object

Commands

Name	Description
delete	Dispose instance of @MilStd1553BusLogger
new	Create new instance of MilStd1553BusLogger

Command new Arguments

Name	Type	Required	Description
bus	object	no	Bus object to attach to (see also attach/detach commands).
fmt	string	yes	Format (csv
pcap	pcapng	log).	name

3.3.6. MilStd1553BusLogger Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object
bus	*void	Bus object to monitor.
statPeriod	double	Statistics report period in seconds, set to positive enables reports.

Commands

Name	Description
attach	Attach to bus.
delete	Dispose instance of MilStd1553BusLogger
detach	Detach from bus.

Command attach Arguments

Name	Type	Required	Description
bus	object	yes	Bus object to attach to.

3.3.7. 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. It would be possible to force statistics reporting at a PPS tick, i.e. by having the a PPS device issue the call. 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. 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.

3.3.8. 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.

3.3.9. API

Interfaces

Listing 6. MIL-STD-1553B Bus Interface

```
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;
```

Listing 7. MIL-STD-1553B Device Interface

```
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;
```

3.3.10. 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 1553 API follows the standard and subdivides transactions in phases. The phases 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 RT and BC models are 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 an RT does not follow the 1553 protocol phases correctly. E.g. sending a status response to a broadcast message, will trigger a message.

Listing 8. Remote Terminal Example

```
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
}
```

3.3.11. Bus Monitors

The 1553 bus interface does not support the implementation of bus monitors directly. The reason for this is that, the notifications already allows the simulator to inspect all the bus traffic. The

notification interface can also be used to modify traffic in flight (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.

3.3.12. 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:

Listing 9. Logging Traffic

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

For CSV output (into milbus0.csv):

Listing 10. Capture to CSV

```
object-create class=MilStd1553BusCapturer name=milbus-cap0 \
  args='fmt:csv,bus:milbus0'
```

For PCAPNG output (into milbus0.pcapng):

Listing 11. Capture to 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:

Example 5. Setting The Time Source

```
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.

3.4. PCI

PCI emulation is experimental.

TEMU provides support for PCI bus based devices. This is based on a split between bridges, devices and buses.

A generic `PCIBus` class exists to provide bus connectivity. Devices of this class maintains the configuration space object.

When a BAR is written the memory and I/O spaces are updated automatically.

Normally the IO and memory spaces can be mapped directly into target memory. That said, som bridges may remap addresses. In that case the bridge should provide memory interfaces much like an MMU.

3.4.1. Bridge Modelling

TBW

3.4.2. Device Modelling

While it is sometimes necessary to implement PCI bridges (e.g. GRPCI2), the more common task is to implement PCI devices.

A PCI device in TEMU must support the following interfaces:

- PCIDevIface implementing the PCI device interface. Used for for accessing config info about the PCI device.
- MemAccessIface[6] implementing the PCI memory access functions for each BAR.
- ExpROMAccessIface implementing accesses to PCI expansion ROM.
- ConfigAccessIface implementing the memory access interface for the configuration space.

That means that MemAccessIface may be either IO or PCI mem. This depends on the hard coded value in the bar.

3.4.3. Examples

An example of a PCI device is provided in `share/temu/examples/pci-device/`.

3.5. Serial / UART

Serial ports are emulated using the `SerialIface`.

3.5.1. Console

The serial console is a simple endpoint for serial traffic. It echos received data to `stdout`, and optionally logs the data in an unbounded log.

Loading the Plugin

```
import Console
```

Configuration

Creation

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

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.

3.5.2. @Console Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object

Commands

Name	Description
delete	Dispose instance of @Console
new	Create new instance of Console

Command new Arguments

Name	Type	Required	Description
name	string	yes	Name of object to create

3.5.3. Console Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object
config.caretControl	uint8_t	
config.outFile	*char	File name to write TTY log to.
config.recordTraffic	uint8_t	
config.reformatNonPrintable	uint8_t	
lastByte	uint8_t	
outByte	uint8_t	
serial	temu_IfaceRef/ <unknown>	Serial connection.

Interfaces

Name	Type	Description
LineDataLoggerIface	LineDataLoggerIface	
SerialIface	SerialIface	

Ports

Prop	Iface	Description
serial	SerialIface	serial port

Commands

Name	Description
delete	Dispose instance of Console

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.

3.5.4. Console Graphical User Interface

The serial console ui is a simple graphical endpoint for serial traffic. 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.

Loading the Plugin

```
import ConsoleUI
```

Configuration

No specific configuration needed.

Creation

The ConsoleUI class is defined in `libTEMUConsoleUI.so`. The constructor takes no parameters.

3.5.5. @ConsoleUI Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name

Name	Type	Description
TimeSource	*void	Time source object

Commands

Name	Description
delete	Dispose instance of @ConsoleUI
new	Create new instance of ConsoleUI

Command new Arguments

Name	Type	Required	Description
name	string	yes	Name of object to create

3.5.6. ConsoleUI Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object
config.appendOutFile	uint8_t	
config.outFile	*char	File name to write TTY log to.
serial	temu_IfaceRef/ <unknown>	Serial connection.

Interfaces

Name	Type	Description
ObjectIface	ObjectIface	
SerialIface	SerialIface	

Ports

Prop	Iface	Description
serial	SerialIface	serial port

Commands

Name	Description
delete	Dispose instance of ConsoleUI

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.

3.6. Signal

The signal interface represents a single bit signal.

The interface can be used to implement GPIO and other pin based interfaces.

3.7. 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 path is supposed to strip the first byte and use it to select the port used to forward the packet.

3.7.1. Interfaces

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

Listing 12. SpaceWire Interfaces

```
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.

3.7.2. 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.

3.7.3. 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

3.7.4. Classes

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.

3.7.5. @SpwRouter Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object

Commands

Name	Description
delete	Dispose instance of @SpwRouter
new	Create new instance of SpwRouter

Command new Arguments

Name	Type	Required	Description
name	string	yes	Name of object to create

3.7.6. SpwRouter Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object
internal.linkState	[int32_t; 32]	Holds the link state of the ports

Name	Type	Description
ports	[temu_IfaceRef; 32]/ <unknown>	Connected SpaceWire devices.
routingTable	[uint8_t; 256]	Configure packet-address/forwarding-port mapping

Interfaces

Name	Type	Description
SpwPortIface	SpwPortIface	Input spacewire ports interfaces

Commands

Name	Description
delete	Dispose instance of SpwRouter

3.7.7. @SpwCssdsAdapter Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object

Commands

Name	Description
delete	Dispose instance of @SpwCssdsAdapter
new	Create new instance of SpwCssdsAdapter

Command new Arguments

Name	Type	Required	Description
name	string	yes	Name of object to create

3.7.8. SpwCssdsAdapter Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object
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
port	temu_IfaceRef/ <unknown>	Connected SpaceWire device.
protocolId	uint8_t	Protocol ID to be used.
rxUdpPort	uint16_t	Udp port used to receive.
targetAddr	[uint8_t; 16]	Addresses to use to forard a packet received via UDP.
targetAddrLength	uint8_t	Number of valid addresses in targetAddr array.
txHost	*char	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

Commands

Name	Description
delete	Dispose instance of SpwCssdsAdapter

3.7.9. Examples

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

them.

Listing 13. Creating and Connecting SpaceWire Router

```
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

Listing 14. SpaceWire Device Example

```
#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");
}
```

Chapter 4. GPIO Bus

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

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.

4.1. Configuration

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

4.2. Class Info

4.2.1. @GpioBus Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object

Commands

Name	Description
delete	Dispose instance of @GpioBus
new	Create new instance of GpioBus

Command new Arguments

Name	Type	Required	Description
name	string	yes	Name of object to create

4.2.2. GpioBus Reference

Properties

Name	Type	Description
Bits	uint64_t	
Class	*void	Class object
Clients	temu_IfaceRefArray	
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object

Interfaces

Name	Type	Description
GpioBusIface	GpioBusIface	

Commands

Name	Description
delete	Dispose instance of GpioBus

4.3. Limitations

Pin updates using the `GpioBusIface` will be distributed to all connected `GpioClients`.

Chapter 5. Generic Cache

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

For the cases where a cache model is needed, the *generic cache model* can be used (see limitations for more info). The generic cache model 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.

When connecting the generic cache model in the memory hierarchy, it will intercept every memory transaction, and disable the ATC for any memory access. This means that performance is significantly impacted, especially in systems with an MMU due to forced table walks.

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).

5.1. Configuration

5.1.1. @GenericCache Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object

Commands

Name	Description
delete	Dispose instance of @GenericCache
new	Create new instance of GenericCache

Command new Arguments

Name	Type	Required	Description
dataLineSize	int	no	Data cache line size
dataSize	int	no	Data cache size

Name	Type	Required	Description
dataWays	int	no	Data cache ways
instrLineSize	int	no	Instruction cache line size
instrSize	int	no	Instruction cache size
instrWays	int	no	Instruction cache ways
lineSize	int	no	Unified cache line size
name	string	yes	Name of object to create
separate	int	no	Separate instruction and data caches
size	int	no	Unified cache size
ways	int	no	Unified cache ways
wordSize	int	no	Word size

5.1.2. GenericCache Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object
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	

Name	Type	Description
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	temu_IfaceRef/ <unknown>	data cache controller
fetchHits	uint64_t	
fetchMisses	uint64_t	
fetchPenalty	int32_t	
icacheCtrl	temu_IfaceRef/ <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	
postTransaction	temu_IfaceRef/ <unknown>	Post transaction interface for chaining
preTransaction	temu_IfaceRef/ <unknown>	Pre transaction interface for chaining
readHits	uint64_t	
readMisses	uint64_t	
readPenalty	int32_t	
wordSize	int32_t	

Name	Type	Description
writeHits	uint64_t	
writeMisses	uint64_t	
writePenalty	int32_t	

Interfaces

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

Commands

Name	Description
delete	Dispose instance of GenericCache

5.1.3. 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).

5.1.4. 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.

5.1.5. 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.

5.2. 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.

5.3. 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 6. 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.

6.1. Loading the Plugin

```
import Leon2SoC
```

6.2. Configuration

6.2.1. 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).

6.2.2. 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.

6.2.3. 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.

6.2.4. 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.

6.2.5. 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.

6.2.6. @Leon2SoC Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component

Name	Type	Description
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object

Commands

Name	Description
delete	Dispose instance of @Leon2SoC
new	Create new instance of Leon2SoC

Command new Arguments

Name	Type	Required	Description
name	string	yes	Name of object to create

6.2.7. Leon2SoC Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object
ahbfailaddr	uint32_t	Fail address register
ahbstat	uint32_t	Fail status register
behaviour	uint8_t	Set to 1 for COLE mode
cachectrl	uint32_t	Cache control register
config.levelMask	uint32_t	Level triggered internal interrupts (mask)
config.logInterrupts	uint8_t	Enable interrupt logging
cpu	temu_IfaceRef/ <unknown>	CPU to control with powerdown
dCache	temu_IfaceRef/ <unknown>	Data cache (optional)

Name	Type	Description
gpioBus	temu_IfaceRef/ <unknown>	GPIO bus (deprecated, use signal interface instead)
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	temu_IfaceRef/ <unknown>	Instruction cache (optional)
infiniteUartSpeed	uint32_t	
inputBits	uint32_t	Input signals
irqControl	temu_IfaceRef/ <unknown>	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)
outSignals	[temu_IfaceRef; 16]/ <unknown>	Outgoing GPIO signals
powerdown	uint32_t	Idle register
presccntr	uint32_t	Prescaler counter register
prescrld	uint32_t	Prescaler reload register
timer1cntr	uint32_t	Timer 1 counter register

Name	Type	Description
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	temu_IfaceRef/ <unknown>	Serial port A
uartb	temu_IfaceRef/ <unknown>	Serial port B
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	

Name	Type	Description
ICacheCtrlIface	CacheCtrlIface	I-cache to control
InternalIrqIface	IrqCtrlIface	Internal IRQ controller (native LEON numbering), post your IRQs here.
IrqClientIface	IrqClientIface	IRQ acknowledgement (from CPU)
IrqIface	IrqCtrlIface	External 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

Registers

Register support is currently experimental!

Register Bank registers

Register ahbfailaddr

Description

Fail address register

Reset value

0x00000000

Warm reset mask

0x00000000

Field	Mask	Reset	Description
-	-	-	-

Register ahbstat

Description

Fail status register

Reset value

0x00000000

Warm reset mask

0x00000000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-																						eed	hed	het	hem				hes		
-																						rw	rw	rw	rw				rw		
-																						-	-	-	-				-		

Field	Mask	Reset	Description
eed	0x00000200	0x0	EDAC-correctable error detected
hed	0x00000100	0x0	Hardware error detected
het	0x00000080	0x0	Hardware error type
hem	0x00000078	0x0	Hardware error module
hes	0x00000007	0x0	Hardware error size

Register cachectrl

Description

Cache control register

Reset value

0xfd178000

Warm reset mask

0x00f9c00f

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
drepl		irepl		isets		-	dsets	ds	fd	fi	cpc		cptb		ib	ip	dp	ite		ide		dte		dde		df	if	dcs		ics	
ro		ro		ro		-	ro	rw	rw	rw	ro		ro		ro	ro	ro	ro		ro		ro		ro		ro	ro	rw		rw	
-		-		-		-	-	0	0	0	2		-		1	1	0	-		-		-		-		-	-	0		0	

Field	Mask	Reset	Description
drepl	0xc0000000	0x3	Data cache replacement policy
irepl	0x30000000	0x3	Instruction cache replacement policy
isets	0x0c000000	0x3	Instruction cache associativity
dsets	0x01000000	0x1	Data cache associativity
ds	0x00800000	0x0	Data cache snoop enable
fd	0x00400000	0x0	Flush data cache

Field	Mask	Reset	Description
fi	0x00200000	0x0	Flush instruction cache
cpc	0x00180000	0x2	Cache parity bits
cptb	0x00060000	0x3	Cache parity test bits
ib	0x00010000	0x1	Instruction burst fetch
ip	0x00008000	0x1	Instruction cache flush pending
dp	0x00004000	0x0	Data cache flush pending
ite	0x00003000	0x0	Instruction cache tag error counter
ide	0x00000c00	0x0	Instruction cache data error counter
dte	0x00000300	0x0	Data cache tag error counter
dde	0x000000c0	0x0	Data cache data error counter
df	0x00000020	0x0	Data cache freeze on interrupt
if	0x00000010	0x0	Instruction cache freeze on interrupt
dcs	0x0000000c	0x0	Data cache state
ics	0x00000003	0x0	Instruction cache state

Register gpiodir

Description

I/O port direction register

Reset value

0x00000000

Warm reset mask

0x00000000

Field	Mask	Reset	Description
-	-	-	-

Register gpioinout

Description

I/O port data register

Reset value

0x00000000

Warm reset mask

0x00000000

Field	Mask	Reset	Description
-	-	-	-

Register gpioirqcfg

Description

I/O port interrupt register 1

Reset value

0x00000000

Warm reset mask

0x00000000

Field	Mask	Reset	Description
-	-	-	-

Register gpioirqcfg2

Description

I/O port interrupt register 2

Reset value

0x00000000

Warm reset mask

0x00000000

Field	Mask	Reset	Description
-	-	-	-

Register irqclear

Description

Interrupt clear register

Reset value

0x00000000

Warm reset mask

0x00000000

Field	Mask	Reset	Description
-	-	-	-

Register irqforce

Description

Interrupt force register

Reset value

0x00000000

Warm reset mask

0x00000000

Field	Mask	Reset	Description
-	-	-	-

Register irqmask

Description

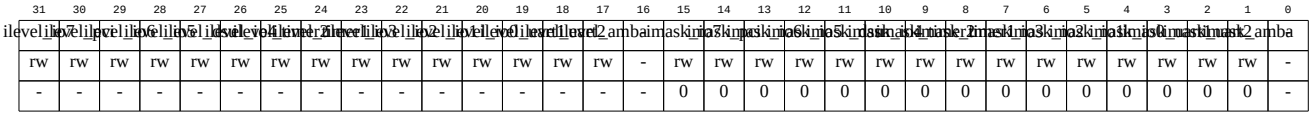
Interrupt mask and priority register

Reset value

0x00000000

Warm reset mask

0x0000fffe



Field	Mask	Reset	Description
ilevel_io7	0x80000000	0x0	Interrupt level
ilevel_pci	0x40000000	0x0	Interrupt level
ilevel_io6	0x20000000	0x0	Interrupt level
ilevel_io5	0x10000000	0x0	Interrupt level
ilevel_dsu	0x08000000	0x0	Interrupt level
ilevel_io4	0x04000000	0x0	Interrupt level
ilevel_timer2	0x02000000	0x0	Interrupt level
ilevel_timer1	0x01000000	0x0	Interrupt level
ilevel_io3	0x00800000	0x0	Interrupt level
ilevel_io2	0x00400000	0x0	Interrupt level
ilevel_io1	0x00200000	0x0	Interrupt level
ilevel_io0	0x00100000	0x0	Interrupt level

Field	Mask	Reset	Description
ilevel_uart1	0x00080000	0x0	Interrupt level
ilevel_uart2	0x00040000	0x0	Interrupt level
ilevel_amba	0x00020000	0x0	Interrupt level
imask_io7	0x00008000	0x0	Interrupt mask
imask_pci	0x00004000	0x0	Interrupt mask
imask_io6	0x00002000	0x0	Interrupt mask
imask_io5	0x00001000	0x0	Interrupt mask
imask_dsu	0x00000800	0x0	Interrupt mask
imask_io4	0x00000400	0x0	Interrupt mask
imask_timer2	0x00000200	0x0	Interrupt mask
imask_timer1	0x00000100	0x0	Interrupt mask
imask_io3	0x00000080	0x0	Interrupt mask
imask_io2	0x00000040	0x0	Interrupt mask
imask_io1	0x00000020	0x0	Interrupt mask
imask_io0	0x00000010	0x0	Interrupt mask
imask_uart1	0x00000008	0x0	Interrupt mask
imask_uart2	0x00000004	0x0	Interrupt mask
imask_amba	0x00000002	0x0	Interrupt mask

Register irqpend

Description

Interrupt pending register

Reset value

0x00000000

Warm reset mask

0x00000000

Field	Mask	Reset	Description
-	-	-	-

Register leoncfg

Description

Product configuration register

Reset value

0x7077bbd5

Warm reset mask

0x00000000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
mmu	dsu	sdrctrl	wtpnb			imac	nwin					icsz			ilsz		dcsz			dlsz		divinst	mulinst	wdog	memstat		fpu		pci		wprt	
ro	ro	ro	ro			ro	ro					ro			ro		ro			ro		ro	ro	ro	ro		ro		ro		ro	
-	-	-	-			-	-					-			-		-			-		-	-	-	-		-		-		-	

Field	Mask	Reset	Description
mmu	0x80000000	0x0	Memory management unit
dsu	0x40000000	0x1	Debug support unit
sdrctrl	0x20000000	0x1	SDRAM controller
wtpnb	0x1c000000	0x4	IU watchpoints
imac	0x02000000	0x0	UMAC/SMAC instructions
nwin	0x01f00000	0x7	IU register file windows
icsz	0x000e0000	0x3	Instruction cache set size
ilsz	0x00018000	0x3	Instruction cache line size
dcsz	0x00007000	0x3	Data cache set size
dlsz	0x00000c00	0x2	Data cache line size
divinst	0x00000200	0x1	UDIV/SDIV instructions
mulinst	0x00000100	0x1	UMUL/SMUL instructions
wdog	0x00000080	0x1	Watchdog
memstat	0x00000040	0x1	Memory status and address failing register
fpu	0x00000030	0x1	FPU type
pci	0x0000000c	0x1	PCI core type
wprt	0x00000003	0x1	Write protections

Register memcfg1

Description

Memory configration register 1

Reset value

0x00000000

Warm reset mask

0x7ef80bff

31	30	29	28 27	26	25	24	23 22 21 20	19	18 17 16 15 14 13 12	11	10	9 8	7 6 5 4	3 2 1 0
-	pbrdy	abrdy	iowdh	iobrdy	bexc	-	iows	ioen	-	prwen	-	prwdh	prwws	prrws
-	rw	rw	rw	rw	rw	-	rw	rw	-	rw	-	rw	rw	rw
-	0	0	0	0	0	-	0	0	-	0	-	0	0	0

Field	Mask	Reset	Description
pbrdy	0x40000000	0x0	PROM area bus-ready enable
abrdy	0x20000000	0x0	Asynchronous bus ready
iowdh	0x18000000	0x0	I/O bus width
iobrdy	0x04000000	0x0	I/O area bus ready enable
bexc	0x02000000	0x0	Bus error enable for RAM PROM and I/O access
iows	0x00f00000	0x0	I/O waitstates
ioen	0x00080000	0x0	I/O area enable
prwen	0x00000800	0x0	PROM write enable
prwdh	0x00000300	0x0	PROM width
prwws	0x000000f0	0x0	PROM write waitstates
prrws	0x0000000f	0x0	PROM read waitstates

Register memcfg2

Description

Memory configuration register 2

Reset value

0x7c400020

Warm reset mask

0xfff86000

31	30	29 28 27	26	25 24 23	22 21	20 19	18 17 16 15	14	13	12 11 10 9	8	7	6	5 4	3 2	1 0
sdrref	trp	trfc	sdrcas	sdrbs	sdrcls	sdrcmd	-	se	si	rambs	-	rambrdy	mrmw	ramwdh	ramwws	ramrws
rw	rw	rw	rw	rw	rw	rw	-	rw	rw	rw	-	rw	rw	rw	rw	rw
0	1	7	1	0	2	0	-	0	0	-	-	-	-	-	-	-

Field	Mask	Reset	Description
sdrref	0x80000000	0x0	SDRAM refresh
trp	0x40000000	0x1	SDRAM t_rp timing
trfc	0x38000000	0x7	SDRAM t_rfp timing
sdrcas	0x04000000	0x1	SDRAM CAS delay
sdrbs	0x03800000	0x0	SDRAM bank size
sdrcls	0x00600000	0x2	SDRAM column size
sdrcmd	0x00180000	0x0	SDRAM command

Field	Mask	Reset	Description
se	0x00004000	0x0	SDRAM enable
si	0x00002000	0x0	SDRAM disable
rambs	0x00001e00	0x0	SRAM bank size
rambrdy	0x00000080	0x0	SRAM area bus ready enable
ramrmw	0x00000040	0x0	SRAM read-modify-write
ramwdh	0x00000030	0x2	SRAM bus width
ramwws	0x0000000c	0x0	SRAM write waitstates
ramrws	0x00000003	0x0	SRAM read waitstates

Register memcfg3

Description

Memory configuration register 3

Reset value

0xc8000000

Warm reset mask

0xc8000c00

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
rfc				me	srcrv															wb	rb	re	pe	tcb							
rw				rw	rw															rw	rw	rw	rw	rw							
3				1	-															0	0	-	-	-							

Field	Mask	Reset	Description
rfc	0xc0000000	0x3	Register file checkbits
me	0x08000000	0x1	Memory EDAC
srcrv	0x07fff000	0x0	SDRAM refresh counter reload value
wb	0x00000800	0x0	EDAC diagnostic write bypass
rb	0x00000400	0x0	EDAC diagnostic read
re	0x00000200	0x0	RAM EDAC enable
pe	0x00000100	0x0	PROM EDAC enable
tcb	0x000000ff	0x0	Test checkbits

Register powerdown

Description

Idle register

Reset value

0x00000000

Warm reset mask

0x00000000

Field	Mask	Reset	Description
-	-	-	-

Register presccntr

Description

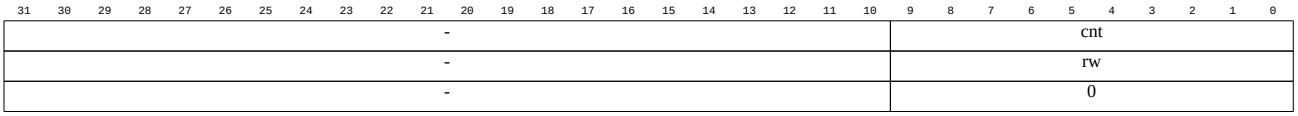
Prescaler counter register

Reset value

0x00000000

Warm reset mask

0x000003ff



Field	Mask	Reset	Description
cnt	0x000003ff	0x0	Prescaler counter value

Register prescrld

Description

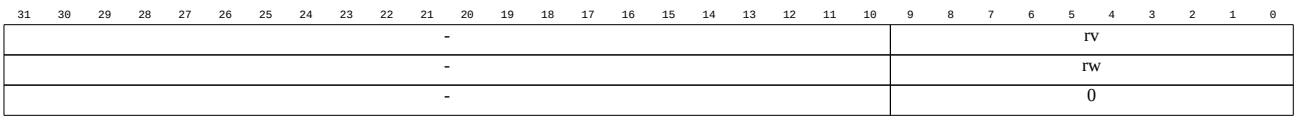
Prescaler reload register

Reset value

0x00000000

Warm reset mask

0x000003ff



Field	Mask	Reset	Description
rv	0x000003ff	0x0	Prescaler reload value

Register timer1cntr

Description

Timer 1 counter register

Reset value

0x00000000

Warm reset mask

0x00000000

Field	Mask	Reset	Description
-	-	-	-

Register timer1ctrl

Description

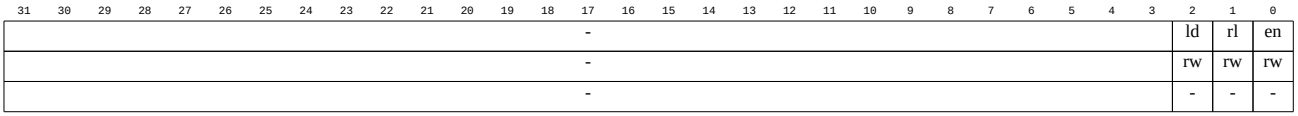
Timer 1 control register

Reset value

0x00000000

Warm reset mask

0x00000000



Field	Mask	Reset	Description
ld	0x00000004	0x0	Load counter
rl	0x00000002	0x0	Reload counter
en	0x00000001	0x0	Enable counter

Register timer1rld

Description

Timer 1 reload register

Reset value

0x00000000

Warm reset mask

0x00000000

Field	Mask	Reset	Description
-	-	-	-

Register timer2cntr

Description

Timer 2 counter register

Reset value

0x00000000

Warm reset mask

0x00000000

Field	Mask	Reset	Description
-	-	-	-

Register timer2ctrl

Description

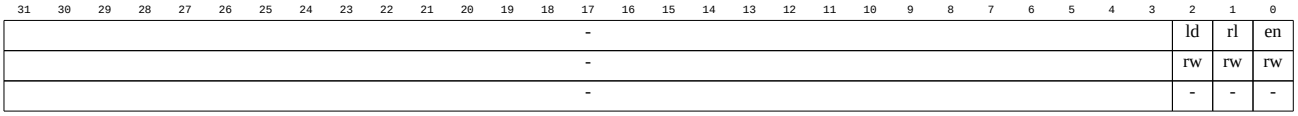
Timer 2 control register

Reset value

0x00000000

Warm reset mask

0x00000000



Field	Mask	Reset	Description
ld	0x00000004	0x0	Load counter
rl	0x00000002	0x0	Reload counter
en	0x00000001	0x0	Enable counter

Register timer2rld

Description

Timer 2 reload register

Reset value

0x00000000

Warm reset mask

0x00000000

Field	Mask	Reset	Description
-	-	-	-

Register uart1ctrl

Description

UART 1 control register

Reset value

0x00000000

Warm reset mask

0x00000143

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-																							ec	lb	fl	pe	ps	ti	ri	te	re
-																							rw	rw	rw	rw	rw	rw	rw	rw	rw
-																							0	-	0	-	-	-	-	0	0

Field	Mask	Reset	Description
ec	0x00000100	0x0	External clock
lb	0x00000080	0x0	Loop back
fl	0x00000040	0x0	Flow control
pe	0x00000020	0x0	Parity enable
ps	0x00000010	0x0	Parity select
ti	0x00000008	0x0	Transmitter interrupt enable
ri	0x00000004	0x0	Receiver interrupt enable
te	0x00000002	0x0	Transmitter enable
re	0x00000001	0x0	Receiver enable

Register uart1datrx

Description

UART 1 RX data register

Reset value

0x00000000

Warm reset mask

0x00000000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-																								rtd							
-																								rw							
-																								-							

Field	Mask	Reset	Description
rtd	0x000000ff	0x0	Received/transmit data

Register uart1scal

Description

UART 1 scaler register

Reset value

0x00000000

Warm reset mask

0x00000000

Field	Mask	Reset	Description
-	-	-	-

Register uart1stat

Description

UART 1 status register

Reset value

0x00000006

Warm reset mask

0x0000007f

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-																									fe	pe	ov	br	th	ts	dr
-																									rw	rw	rw	rw	ro	ro	ro
-																									0	0	0	0	1	1	0

Field	Mask	Reset	Description
fe	0x00000040	0x0	Framing error
pe	0x00000020	0x0	Parity error
ov	0x00000010	0x0	Overrun
br	0x00000008	0x0	Break received
th	0x00000004	0x1	Transmitter hold register empty
ts	0x00000002	0x1	Transmitter shift register empty
dr	0x00000001	0x0	Data ready

Register uart2ctrl

Description

UART 2 control register

Reset value

0x00000000

Warm reset mask

0x00000143

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-																							ec	lb	fl	pe	ps	ti	ri	te	re
-																							rw	rw	rw	rw	rw	rw	rw	rw	rw
-																							0	-	0	-	-	-	-	0	0

Field	Mask	Reset	Description
ec	0x00000100	0x0	External clock
lb	0x00000080	0x0	Loop back
fl	0x00000040	0x0	Flow control
pe	0x00000020	0x0	Parity enable
ps	0x00000010	0x0	Parity select
ti	0x00000008	0x0	Transmitter interrupt enable
ri	0x00000004	0x0	Receiver interrupt enable
te	0x00000002	0x0	Transmitter enable
re	0x00000001	0x0	Receiver enable

Register uart2datrx

Description

UART 1 RX data register

Reset value

0x00000000

Warm reset mask

0x00000000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-																								rtd							
-																								rw							
-																								-							

Field	Mask	Reset	Description
rtd	0x000000ff	0x0	Received/transmit data

Register uart2scal

Description

UART 2 scaler register

Reset value

0x00000000

Warm reset mask

0x00000000

Field	Mask	Reset	Description
-	-	-	-

Register uart2stat

Description

UART 2 status register

Reset value

0x00000006

Warm reset mask

0x0000007f

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-																									fe	pe	ov	br	th	ts	dr
-																									rw	rw	rw	rw	ro	ro	ro
-																									0	0	0	0	1	1	0

Field	Mask	Reset	Description
fe	0x00000040	0x0	Framing error
pe	0x00000020	0x0	Parity error
ov	0x00000010	0x0	Overrun
br	0x00000008	0x0	Break received
th	0x00000004	0x1	Transmitter hold register empty
ts	0x00000002	0x1	Transmitter shift register empty
dr	0x00000001	0x0	Data ready

Register watchdog

Description

Watchdog register

Reset value

0x00000000

Warm reset mask

0x00000000

Field	Mask	Reset	Description
-	-	-	-

Register writeprot1

Description

Write protection register 1

Reset value

0x00000000

Warm reset mask

0x80000000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
en	bp	-	tag														-	mask													
rw	rw	-	rw														-	rw													
0	-	-	-														-	-													

Field	Mask	Reset	Description
en	0x80000000	0x0	Enable
bp	0x40000000	0x0	Block protect
tag	0x1fff8000	0x0	Address tag
mask	0x00003fff	0x0	Address mask

Register writeprot2

Description

Write protection register 2

Reset value

0x00000000

Warm reset mask

0x80000000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
en	bp	-	tag														-	mask													
rw	rw	-	rw														-	rw													
0	-	-	-														-	-													

Field	Mask	Reset	Description
en	0x80000000	0x0	Enable
bp	0x40000000	0x0	Block protect

Field	Mask	Reset	Description
tag	0x1fff8000	0x0	Address tag
mask	0x00003fff	0x0	Address mask

Register writeprotstart1

Description

Write protection start address 1

Reset value

0x00000000

Warm reset mask

0x00000000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	start																													bp	-
-	rw																													rw	-
-	-																													-	-

Field	Mask	Reset	Description
start	0x3ffffffc	0x0	Start address
bp	0x00000002	0x0	Block protect

Register writeprotstart2

Description

Write protection start address 2

Reset value

0x00000000

Warm reset mask

0x00000000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	start																													bp	-
-	rw																													rw	-
-	-																													-	-

Field	Mask	Reset	Description
start	0x3ffffffc	0x0	Start address
bp	0x00000002	0x0	Block protect

Register writeprotstop1

Description

Write protection end address 1

Reset value

0x00000000

Warm reset mask

0x00000003

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	end																													us	su
-	rw																													rw	rw
-	-																													0	0

Field	Mask	Reset	Description
end	0x3ffffffc	0x0	End address
us	0x00000002	0x0	User mode
su	0x00000001	0x0	Supervisor mode

Register writeprotstop2

Description

Write protection end address 2

Reset value

0x00000000

Warm reset mask

0x00000003

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	end																													us	su
-	rw																													rw	rw
-	-																													0	0

Field	Mask	Reset	Description
end	0x3ffffffc	0x0	End address
us	0x00000002	0x0	User mode
su	0x00000001	0x0	Supervisor mode

Commands

Name	Description
delete	Dispose instance of Leon2SoC
lowerInternalIrq	Lower internally numbered interrupts
raiseExternalIrq	Raise externally numbered interrupts

Name	Description
raiseInternalIrq	Raise internally numbered interrupts

Command lowerInternalIrq Arguments

Name	Type	Required	Description
irq	int	yes	Interrupt number

Command raiseExternalIrq Arguments

Name	Type	Required	Description
irq	int	yes	Interrupt number

Command raiseInternalIrq Arguments

Name	Type	Required	Description
irq	int	yes	Interrupt number

6.3. 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 7. 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.

7.1. Configuration

7.1.1. @Machine Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object

Commands

Name	Description
delete	Dispose instance of @Machine
new	Create new instance of Machine

Command new Arguments

Name	Type	Required	Description
name	string	yes	Name of object to create

7.1.2. Machine Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object
cpus	temu_IfaceRefArray	Processors in the machine
currentCPU	temu_IfaceRef/ <unknown>	Current CPU
currentCPUIdx	int32_t	Current CPU Index
devices	temu_IfaceRefArray	Devices to reset when machine is reset
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	
MachineIface	MachineIface	
ObjectIface	ObjectIface	
ResetIface	ResetIface	

Commands

Name	Description
delete	Dispose instance of Machine

7.2. Limitations

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

Chapter 8. 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.

8.1. Loading the Plugin

```
import Mec
```

8.2. Configuration

8.2.1. 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.

8.2.2. 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.

8.2.3. 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.

8.2.4. @Mec Reference

Properties

Name	Type	Description
Class	*void	Class object

Name	Type	Description
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object

Commands

Name	Description
delete	Dispose instance of @Mec
new	Create new instance of Mec

Command new Arguments

Name	Type	Required	Description
name	string	yes	Name of object to create

8.2.5. Mec Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object
accessProtSegment1Base	uint32_t	
accessProtSegment1End	uint32_t	
accessProtSegment2Base	uint32_t	
accessProtSegment2End	uint32_t	
config.infiniteUartSpeed	uint32_t	
cpu	temu_IfaceRef/ <unknown>	Processor
errorAndResetStatus	uint32_t	
failingAddr	uint32_t	

Name	Type	Description
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	temu_IfaceRef/ <unknown>	Upstream interrupt controller (e.g. CPU)
irqForce	uint32_t	
irqMask	uint32_t	
irqPending	uint32_t	
irqShape	uint32_t	
mecCtrl	uint32_t	
memoryConfig	uint32_t	
outSignals	[temu_IfaceRef; 8]/ <unknown>	GPIO signals
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	
uartChanARxTx	uint32_t	
uartChanBRxTx	uint32_t	
uartStatus	uint32_t	
uarta	temu_IfaceRef/ <unknown>	Serial port A
uartb	temu_IfaceRef/ <unknown>	Serial port B
waitStateConfig	uint32_t	

Name	Type	Description
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

Commands

Name	Description
delete	Dispose instance of Mec

8.3. 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.

8.4. Limitations

The following deviations from real hardware are known to exist. If you need the correct behavior (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. Note that transmission times do take into account stop and parity bits.
- 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 9. GRLIB

9.1. 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.

9.1.1. Loading the Plugin

```
import ApbUart
```

@ApbUart Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object

Commands

Name	Description
delete	Dispose instance of @ApbUart
new	Create new instance of ApbUart

Command new Arguments

Name	Type	Required	Description
name	string	yes	Name of object to create

ApbUart Reference

Properties

Name	Type	Description
Class	*void	Class object

Name	Type	Description
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object
config.clockDivider	uint32_t	Clock divider
config.fifoSize	uint8_t	UART FIFO size
config.infiniteUartSpeed	uint8_t	Set to 1 to enable immediate UARTs
config.interrupt	uint8_t	Interrupt number
control	uint32_t	UART control register.
data	uint32_t	UART data register
fifo_debug	uint32_t	UART fifo debug register.
irqCtrl	temu_IfaceRef/ <unknown>	Interrupt controller.
pnp.bar	uint32_t	AMBA plug and play base address register
pnp.config	uint32_t	AMBA plug and play config word
rxFifo.data	[uint8_t; 32]	RX FIFO data
rxFifo.size	uint8_t	RX size
rxFifo.start	uint8_t	RX start index
rxFifo.usage	uint8_t	RX usage
scaler	uint32_t	Scaler register.
status	uint32_t	UART status register
tx	temu_IfaceRef/ <unknown>	Transmit target.
txFifo.data	[uint8_t; 32]	TX FIFO data
txFifo.size	uint8_t	TX size
txFifo.start	uint8_t	TX start index
txFifo.usage	uint8_t	TX usage
txShift	uint8_t	UART shift register

Interfaces

Name	Type	Description
ApbIface	ApbIface	AMBA plug and play interface.
DeviceIface	DeviceIface	Device interface.
MemAccessIface	MemAccessIface	Memory access interface.
ResetIface	ResetIface	Reset interface.
UartIface	SerialIface	Serial input interface.

Ports

Prop	Iface	Description
tx	UartIface	serial port

Commands

Name	Description
delete	Dispose instance of ApbUart

9.1.2. Limitations

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

9.2. CANOC

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

9.2.1. Loading the Plugin

```
import OpenCores
```

9.2.2. 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.

9.2.3. @CAN_OC Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object

Commands

Name	Description
delete	Dispose instance of @CAN_OC
new	Create new instance of CAN_OC

Command new Arguments

Name	Type	Required	Description
name	string	yes	Name of object to create

9.2.4. CAN_OC Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object
basiccan.acceptCode	uint8_t	Accept Code register for BasicCAN mode.
basiccan.acceptMask	uint8_t	Accept Mask register for BasicCAN mode.

Name	Type	Description
basiccan.ctrl	uint8_t	Control register for BasicCAN mode.
basiccan.txID	[uint8_t; 2]	TxID registers for BasicCAN mode.
bus	temu_IfaceRef/ <unknown>	CAN bus the device is connected to.
busTiming	[uint8_t; 2]	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).
config.interrupt	uint8_t	External interrupt raised with IRQ controller.
fifo.data	[uint8_t; 64]	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	temu_IfaceRef/ <unknown>	Interrupt controller.
pelican.acceptCode	[uint8_t; 4]	Accept Code registers for PeliCAN mode.
pelican.acceptMask	[uint8_t; 4]	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.

Name	Type	Description
pelican.txErrCounter	uint8_t	TX Error Counter register for PeliCAN mode.
pelican.txFI	uint8_t	TX Frame Info register for PeliCAN mode.
pelican.txID	[uint8_t; 4]	TxID registers for PeliCAN mode.
status	uint8_t	Status register.
txData	[uint8_t; 8]	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	

Commands

Name	Description
delete	Dispose instance of CAN_OC

9.2.5. 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 behavior and may have an impact on FDIR. Let us know if this is an issue.
- 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 (this assumption is in non-infinite speed mode). This is arguably incorrect and the timing should be picked from the bus timing register Contact Terma if this is this is critical for your needs.

9.3. 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.

9.3.1. Loading the Plugin

```
import GpTimer
```

9.3.2. Configuration

Separate Interrupts

Set the "config.separateInterrupts" property to non-zero.

Interrupt Index

The interrupt number can be set by configuring the "pnp.config" property. The lower 5 bits of the property is used for this.

Number of Timers

Set the "config.numTimers" property. By default this value is 4 to be compatible with the UT699.

Clear IRQ on Set

Set the "config.clearIrqOnSet" property changes the behaviour of bit 4 in the timer control registers. ClearOnSet can be non-zero (the default), in that case writing a 1 to the bit will clear bit 4 (i.e. it will read out as 0), if clearOnSet is zero, the bit is cleared if bit 4 is 0 in the written word.

The reason for this are ambiguities in the LEON3 and UT699 manuals.

9.3.3. @GpTimer Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object

Commands

Name	Description
delete	Dispose instance of @GpTimer
new	Create new instance of GpTimer

Command new Arguments

Name	Type	Required	Description
name	string	yes	Name of object to create

9.3.4. GpTimer Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object
config.clearIrqOnSet	uint8_t	Use behaviour in GRIP manual. Set to zero for beahviour as documented in (UT699 manual from 2012 (p77).
config.clockDivider	uint32_t	Clock divider for scaling event posting
config.irqNumber	uint8_t	Set interrupt number for first interrupt
config.numTimers	uint8_t	
config.separateInterrupts	uint8_t	Enable separate interrupts
config.traceReads	uint8_t	
config.traceWrites	uint8_t	
configReg	uint32_t	
control	[uint32_t; 7]	
counters	[uint32_t; 7]	
irqCtrl	temu_IfaceRef/ <unknown>	Interrupt controller.
pnp.bar	uint32_t	

Name	Type	Description
pnp.config	uint32_t	
reload	[uint32_t; 7]	
scaler	uint32_t	
scalerReload	uint32_t	

Interfaces

Name	Type	Description
ApbIface	ApbIface	
DeviceIface	DeviceIface	
MemAccessIface	MemAccessIface	
ObjectIface	ObjectIface	
ResetIface	ResetIface	

Commands

Name	Description
delete	Dispose instance of GpTimer

9.3.5. 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.
- The timer utilize synchronized events. The minimum time for a timer expiration on a multi-core CPU, will thus be equal to the time-quanta of the machine.

9.4. GRCAN

The GRCAN model is available in the GrCan plugin.

9.4.1. Loading the Plugin

```
import GrCan
```

9.4.2. Configuration

9.4.3. @GRCAN Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object

Commands

Name	Description
delete	Dispose instance of @GRCAN
new	Create new instance of GRCAN

Command new Arguments

Name	Type	Required	Description
name	string	yes	Name of object to create

9.4.4. GRCAN Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object
bus	temu_IfaceRef/ <unknown>	CAN bus.
cfg	uint32_t	Congifuation register
config.irq	uint8_t	Interrupt number

Name	Type	Description
config.singleIrq	uint8_t	Single interrupt
ctrl	uint32_t	Control register
irqCtrl	temu_IfaceRef/ <unknown>	IRQ controller.
irqMask	uint32_t	Interrupt register
mem	temu_IfaceRef/ <unknown>	Memory (deprecated)
memAccess	temu_IfaceRef/ <unknown>	Memory access for DMA
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
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

Interfaces

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

Registers

Register support is currently experimental!

Register Bank default

Register cfg

Description

Congifuation register

Reset value

0x00000000

Warm reset mask

0x00000000

Field	Mask	Reset	Description
-	-	-	-

Register ctrl

Description

Control register

Reset value

0x00000000

Warm reset mask

0x00000000

Field	Mask	Reset	Description
-	-	-	-

Register irqMask

Description

Interrupt register

Reset value

0x00000000

Warm reset mask

0x00000000

Field	Mask	Reset	Description
-	-	-	-

Register pendIrq

Description

Pending interrupt register

Reset value

0x00000000

Warm reset mask

0x00000000

Field	Mask	Reset	Description
-	-	-	-

Register rxChanAddr

Description

RX channel address register

Reset value

0x00000000

Warm reset mask

0x00000000

Field	Mask	Reset	Description
-	-	-	-

Register rxChanCode

Description

RX channel code register

Reset value

0x00000000

Warm reset mask

0x00000000

Field	Mask	Reset	Description
-	-	-	-

Register rxChanCtrl

Description

RX channel control register

Reset value

0x00000000

Warm reset mask

0x00000000

Field	Mask	Reset	Description
-	-	-	-

Register rxChanIrq

Description

RX channel irq register

Reset value

0x00000000

Warm reset mask

0x00000000

Field	Mask	Reset	Description
-	-	-	-

Register rxChanMask

Description

RX channel mask register

Reset value

0x00000000

Warm reset mask

0x00000000

Field	Mask	Reset	Description
-	-	-	-

Register rxChanRd

Description

RX channel read register

Reset value

0x00000000

Warm reset mask

0x00000000

Field	Mask	Reset	Description
-	-	-	-

Register rxChanSize

Description

RX channel size register

Reset value

0x00000000

Warm reset mask

0x00000000

Field	Mask	Reset	Description
-	-	-	-

Register rxChanWr

Description

RX channel write register

Reset value

0x00000000

Warm reset mask

0x00000000

Field	Mask	Reset	Description
-	-	-	-

Register stat

Description

Status register

Reset value

0x00000000

Warm reset mask

0x00000000

Field	Mask	Reset	Description
-	-	-	-

Register syncCodeFilt

Description

SYNC code filter register

Reset value

0x00000000

Warm reset mask

0x00000000

Field	Mask	Reset	Description
-	-	-	-

Register syncMaskFilt

Description

SYNC mask filter register

Reset value

0x00000000

Warm reset mask

0x00000000

Field	Mask	Reset	Description
-	-	-	-

Register txChanAddr

Description

TX channel address register

Reset value

0x00000000

Warm reset mask

0x00000000

Field	Mask	Reset	Description
-	-	-	-

Register txChanCtrl

Description

TX channel control register

Reset value

0x00000000

Warm reset mask

0x00000000

Field	Mask	Reset	Description
-	-	-	-

Register txChanIrq

Description

TX channel irq register

Reset value

0x00000000

Warm reset mask

0x00000000

Field	Mask	Reset	Description
-	-	-	-

Register txChanRd

Description

TX channel read register

Reset value

0x00000000

Warm reset mask

0x00000000

Field	Mask	Reset	Description
-	-	-	-

Register txChanSize

Description

TX channel size register

Reset value

0x00000000

Warm reset mask

0x00000000

Field	Mask	Reset	Description
-	-	-	-

Register txChanWr

Description

TX channel write register

Reset value

0x00000000

Warm reset mask

0x00000000

Field	Mask	Reset	Description
-	-	-	-

Commands

Name	Description
delete	Dispose instance of GRCAN

9.4.5. Limitations

- None

9.5. GRETH and GRETH_GBIT

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`.

9.5.1. 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
```

9.5.2. Configuration

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.

9.5.3. @GRETH Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object

Commands

Name	Description
delete	Dispose instance of @GRETH
new	Create new instance of GRETH

Command new Arguments

Name	Type	Required	Description
name	string	yes	Name of object to create

9.5.4. GRETH Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
ETHCTR	uint32_t	Ethernet Control Register
ETHMDC	uint32_t	Ethernet MDIO Control and Status Register
ETHRDP	uint32_t	Ethernet Receiver Descriptor Pointer Register
ETHSIS	uint32_t	Ethernet Status and Interrupt Source Register
ETHTDP	uint32_t	Ethernet Transmitter Descriptor Pointer Register
LoggingFlags	uint64_t	Flags for logging info
MACLSB	uint32_t	Ethernet MAC Address LSB
MACMSB	uint32_t	Ethernet MAC Address MSB
Name	*char	Object name
TimeSource	*void	Time source object
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

Name	Type	Description
config.logTraffic	uint8_t	Enable traffic logging
irqCtrl	temu_IfaceRef/ <unknown>	IRQ controller.
mac	*char	Set MAC by string
mdioBus	temu_IfaceRef/ <unknown>	MDIO bus.
memAccess	temu_IfaceRef/ <unknown>	Memory access (for DMA).
memory	temu_IfaceRef/ <unknown>	Memory (deprecated)
phy	temu_IfaceRef/ <unknown>	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	

Registers

Register support is currently experimental!

Register Bank registers

Register ETHCTR

Description

Ethernet Control Register

Reset value

0x00000000

Warm reset mask

0x00000000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EA	BS			GA	MA	MC	-																	SP	RS	PM	FD	RI	TI	RE	TE
ro	ro			ro	ro	ro	-																	ro	ro	ro	ro	ro	ro	ro	ro
-	-			-	-	-	-																	-	-	-	-	-	-	-	-

Field	Mask	Reset	Description
EA	0x80000000	0x0	EDCL available
BS	0x70000000	0x0	EDCL buffer size

Field	Mask	Reset	Description
GA	0x08000000	0x0	Gigabit MAC
MA	0x04000000	0x0	MDIO interrupts supported
MC	0x02000000	0x0	Multicast supported
SP	0x00000080	0x0	Speed
RS	0x00000040	0x0	Reset
PM	0x00000020	0x0	Open Packet Mode
FD	0x00000010	0x0	Full Duplex
RI	0x00000008	0x0	Enable Receiver Interrupts
TI	0x00000004	0x0	Enable Transmitter Interrupts
RE	0x00000002	0x0	Receive Enable
TE	0x00000001	0x0	Transmit Enable

Register ETHMDC

Description

Ethernet MDIO Control and Status Register

Reset value

0x00000000

Warm reset mask

0x00000000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Data																PHY_ADDR					REG_ADDR					-	NV	BU	LF	RD	WR
ro																ro					ro					-	ro	ro	ro	ro	ro
-																-					-					-	-	-	-	-	-

Field	Mask	Reset	Description
Data	0xffff0000	0x0	Data for MMI read / write
PHY_ADDR	0x0000f800	0x0	PHY address
REG_ADDR	0x000007c0	0x0	MII reg addr
NV	0x00000010	0x0	Not valid
BU	0x00000008	0x0	Busy
LF	0x00000004	0x0	Link fail
RD	0x00000002	0x0	Read
WR	0x00000001	0x0	Write

Register ETHRDP

Description

Ethernet Receiver Descriptor Pointer Register

Reset value

0x00000000

Warm reset mask

0x00000000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RXDTRA																					-	RX_DESCRIPTOR_PTR								-	
ro																					-	ro								-	
-																					-	-								-	

Field	Mask	Reset	Description
RXDTRA	0xfffff800	0x0	Rx desc base address
RX_DESCRIPTOR_PTR	0x000003f8	0x0	Rx desc offset

Register ETHSIS

Description

Ethernet Status and Interrupt Source Register

Reset value

0x00000000

Warm reset mask

0x00000000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-																								IA	TS	TA	RA	TI	RI	TE	RE
-																								ro	ro	ro	ro	ro	ro	ro	ro
-																								-	-	-	-	-	-	-	-

Field	Mask	Reset	Description
IA	0x00000080	0x0	Invalid Address
TS	0x00000040	0x0	Too Small
TA	0x00000020	0x0	Transmitter AHB Error
RA	0x00000010	0x0	Receiver AHB Error
TI	0x00000008	0x0	Transmitter Interrupt
RI	0x00000004	0x0	Receiver Interrupt
TE	0x00000002	0x0	Transmitter Error
RE	0x00000001	0x0	Receiver Error

Register ETHTDP

Description

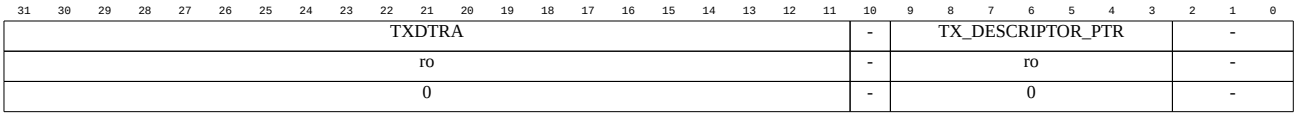
Ethernet Transmitter Descriptor Pointer Register

Reset value

0x00000000

Warm reset mask

0xfffffbf8



Field	Mask	Reset	Description
TXDTRA	0xfffff800	0x0	Tx desc base address
TX_DESCRIPTOR_PTR	0x000003f8	0x0	Tx desc offset

Register MACLSB

Description

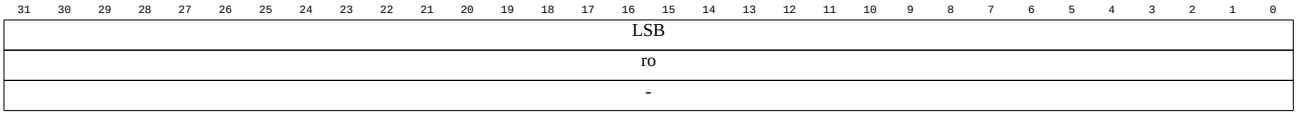
Ethernet MAC Address LSB

Reset value

0x00000000

Warm reset mask

0x00000000



Field	Mask	Reset	Description
LSB	0xffffffff	0x0	Four LSB of MAC

Register MACMSB

Description

Ethernet MAC Address MSB

Reset value

0x00000000

Warm reset mask

0x00000000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-																MSB															
-																ro															
-																-															

Field	Mask	Reset	Description
MSB	0x0000ffff	0x0	Two MSB of MAC

Commands

Name	Description
delete	Dispose instance of GRETH
sendFrame	Send frame
setMAC	Set MAC address

Command sendFrame Arguments

Name	Type	Required	Description
mac	string	yes	MAC address of target

Command setMAC Arguments

Name	Type	Required	Description
mac	string	yes	MAC address to set

9.5.5. Limitations

- Multicast groups are not yet supported.
- ECDL mode is not supported.

9.6. 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.

9.6.1. Loading the Plugin

```
import GrGPIO
```

9.6.2. Usage

The device can be connected to and from via the signal interface it implements. It implements 16 usable signals (signal 0 through 15). Signal 0 cannot raise interrupts.

You can connect the signal interface as follows:

Listing 15. Connecting via Command Line

```
# Connect GPIO device signal 0 to device model
connect a=gpio.outSignals[0] b=mydevice:SignalIface

# Connect a device signal interface ref to GPIO device
connect a=mydevice.signal b=gpio:SignalIface[1]
```

Listing 16. Connecting via API

```
// Connect GPIO device signal 0 to device model
temu_connect(gpio, "outSignals[0]", mydevice, "SignalIface");

// Connect a device signal interface ref to GPIO device
temu_connect(mydevice, "signal", gpio, "SignalIface[1]");
```

9.6.3. @GrGPIO Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object

Commands

Name	Description
delete	Dispose instance of @GrGPIO
new	Create new instance of GrGPIO

Command new Arguments

Name	Type	Required	Description
name	string	yes	Name of object to create

9.6.4. GrGPIO Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object
config.irqMask	uint32_t	
config.pinMask	uint32_t	
data	uint32_t	
direction	uint32_t	
edge	uint32_t	
irqCtrl	temu_IfaceRef/ <unknown>	Interrupt controller
mask	uint32_t	
outSignals	[temu_IfaceRef; 32]/ <unknown>	GPIO signals
output	uint32_t	
pnp.bar	uint32_t	
pnp.config	uint32_t	
polarity	uint32_t	

Interfaces

Name	Type	Description
ApbIface	ApbIface	
DeviceIface	DeviceIface	
MemAccessIface	MemAccessIface	
ResetIface	ResetIface	
SignalIface	SignalIface	Incomming signals

Commands

Name	Description
delete	Dispose instance of GrGPIO

9.6.5. 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.

9.7. GRIOMMU

The GRIOMMU model is available in the GrIoMmu plugin.

9.7.1. Loading the Plugin

```
import GrIoMmu
```

9.7.2. Configuration

The model should be attached in two directions:

Firstly, IO-devices need to have their memory access interface references routed through the IOMMU. To do this, connect the memory access iface ref in the device to `IOMMUAccessIface` in the IOMMU.

Secondly, the IOMMU needs to get access to the device's AMBA PNP info. The info is used to populate the `MasterConfig` registers. To set the PNP info, attach it to the `devicePnp` array.

The `IOMMUAccessIface` and `devicePnp` array assumes that the same device indexes are used. Not connecting devices the correct way is undefined behaviour.

Listing 17. Connecting the GRIOMMU

```
// Connect command
connect a=iommu.devicePnp[0] b=device:ApbIface ①
connect a=device.mem b=iommu:IOMMUAccessIface[0] ②

// Or with assigmment syntax
iommu.devicePnp[0] = device:ApbIface ①
device.mem = iommu:IOMMUAccessIface[0] ②
```

① Index should match index on next line.

② Index should match index on previous line.

9.7.3. @GRIOMMU Reference

Properties

Name	Type	Description
Class	*void	Class object

Name	Type	Description
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object

Commands

Name	Description
delete	Dispose instance of @GRIOMMU
new	Create new instance of GRIOMMU

Command new Arguments

Name	Type	Required	Description
name	string	yes	Name of object to create

9.7.4. GRIOMMU Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object
abhFailingAccess	uint32_t	AHB failing access register
asmpAccessControl	[uint32_t; 4]	ASMP access control register
capbility	[uint32_t; 3]	Capability register
config.interrupt	uint8_t	Interrupt number
control	uint32_t	Control register
dataRamErrorInjection	uint32_t	Data RAM error injection register

Name	Type	Description
devicePnp	[temu_IfaceRef; 16]/ <unknown>	Devices under IOMMU control
diagnosticCacheAccess	uint32_t	Diagnostic cache access register
diagnosticCacheAccessData	[uint32_t; 8]	Diagnostic cache access data register
diagnosticCacheAccessTag	uint32_t	Diagnostic cache access tag register
groupConfig	[uint32_t; 16]	Group config register
irq	temu_IfaceRef/ <unknown>	ASMP access control register
irqMask	uint32_t	Interrupt mask register
masterConfig	[uint32_t; 16]	Master config register
mem	temu_IfaceRef/ <unknown>	Main memory bus
status	uint32_t	Status register
tagRamErrorInjection	uint32_t	Tag RAM error injection register
tlbCacheFlush	uint32_t	TLB/cache flush register

Interfaces

Name	Type	Description
DeviceIface	DeviceIface	
IOMMUAccessIface	MemAccessIface	IOMMU memory access interfaces
MemAccessIface	MemAccessIface	
ResetIface	ResetIface	

Commands

Name	Description
delete	Dispose instance of GRIOMMU

9.7.5. Limitations

- Read and write combining is not simulated
- Bus select is ignored, only one memory space is used for memory and processor.
- APV and TLB caches are not implemented at the moment.
- Flushes are instantaneous. This means that the FLI and FCI will only result in one actual

interrupt being raised.

9.8. GRPCI2

The GRPCI2 model is available in the GrPci2 plugin.

9.8.1. Loading the Plugin

```
import GrPci2
```

9.8.2. Configuration

9.8.3. @GRPCI2 Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object

Commands

Name	Description
delete	Dispose instance of @GRPCI2
new	Create new instance of GRPCI2

Command new Arguments

Name	Type	Required	Description
name	string	yes	Name of object to create

9.8.4. GRPCI2 Reference

Properties

Name	Type	Description
AHB2PCI	uint32_t	AHB to PCI mapping for PCI I/O
AHBM2PCI	[uint32_t; 16]	DMA channel active
BCIM	uint32_t	PCI master prefetch burst limit
CTRL	uint32_t	Control register
Class	*void	Class object
Component	*void	Pointer to component object if part of component
DMABASE	uint32_t	DMA descriptor base address
DMACHAN	uint32_t	DMA channel active
DMACTRL	uint32_t	GRPCI2 DMA control and status register
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
PCI2AHB	[uint32_t; 6]	DMA channel active
STATCAP	uint32_t	Status and Capability register
TimeSource	*void	Time source object
ioMem	temu_IfaceRef/ <unknown>	PCI I/O space object
irqCtrl	temu_IfaceRef/ <unknown>	Upward interrupt controller (i.e. on AMBA bus)
pciBus	*void	PCI bus object
pciDevices	temu_IfaceRefArray	PCI devices
pciMem	temu_IfaceRef/ <unknown>	PCI memory space object

Interfaces

Name	Type	Description
ApbIface	ApbIface	APB P&P interface
ConfigAccessIface	MemAccessIface	PCI config access interface.
IrqIface	IrqCtrlIface	PCI IRQ interface
MemAccessIface	MemAccessIface	Memory access interface (registers)
PCIBridgeIface	temu::PCIBridgeIface	PCI bridge interface.

Registers

Register support is currently experimental!

Register Bank default

Register AHB2PCI

Description

AHB to PCI mapping for PCI I/O

Reset value

0x00000000

Warm reset mask

0x00000000

Field	Mask	Reset	Description
-	-	-	-

Register AHBM2PCI

Description

DMA channel active

Reset value

0x00000000

Warm reset mask

0x00000000

Field	Mask	Reset	Description
-	-	-	-

Register BCIM

Description

PCI master prefetch burst limit

Reset value

0x000000ff

Warm reset mask

0xffff00ff

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AHB_master_unmask																-								burst_length							
rw																-								rw							
0																-								255							

Field	Mask	Reset	Description
AHB_master_unmask	0xffff0000	0x0	Limit prefetch burst

Field	Mask	Reset	Description
burst_length	0x000000ff	0xff	Max number of beats / burst

Register CTRL

Description

Control register

Reset value

0x00000000

Warm reset mask

0xefff0fff

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RE	MR	TR	-	SI	PE	ER	EI	BusNumber								-				DFA	IB	CB	DIF	DeviceINTMask				HostINTMask			
rw	rw	rw	-	rw	rw	rw	rw	rw								-				rw	rw	rw	rw	rw				rw			
0	0	0	-	0	0	0	0	0								-				0	0	0	0	0				0			

Field	Mask	Reset	Description
RE	0x80000000	0x0	Reset
MR	0x40000000	0x0	Master Reset
TR	0x20000000	0x0	Target Reset
SI	0x08000000	0x0	System Error Interrupt Enable
PE	0x04000000	0x0	Parity Error
ER	0x02000000	0x0	Master / Target Error
EI	0x01000000	0x0	Master / Target Interrupt
BusNumber	0x00ff0000	0x0	Bus number for config cycles
DFA	0x00000800	0x0	DMA fair arbitration
IB	0x00000400	0x0	Burst I/O cycles
CB	0x00000200	0x0	Burst config cycles
DIF	0x00000100	0x0	Device Interrupt Force
DeviceINTMask	0x000000f0	0x0	Device interrupt mask
HostINTMask	0x0000000f	0x0	Host interrupt mask

Register DMABASE

Description

DMA descriptor base address

Reset value

0x00000000

Warm reset mask

0x00000000

Field	Mask	Reset	Description
-	-	-	-

Register DMACHAN

Description

DMA channel active

Reset value

0x00000000

Warm reset mask

0x00000000

Field	Mask	Reset	Description
-	-	-	-

Register DMACTRL

Description

GRPCI2 DMA control and status register

Reset value

0x80000000

Warm reset mask

0x800fffff

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SAFE	-											CHIRQ								MA	TA	PE	AE	DE	NumDMAChan			ACTIVE	DIS	IE	EN
rw	-											rw								rw	rw	rw	rw	rw	rw			ro	rw	rw	rw
1	-											0								0	0	0	0	0	0			0	0	0	0

Field	Mask	Reset	Description
SAFE	0x80000000	0x1	Arming of control field updates
CHIRQ	0x000ff000	0x0	Channel IRQ status
MA	0x00000800	0x0	Master abort
TA	0x00000400	0x0	Target abort
PE	0x00000200	0x0	Parity error
AE	0x00000100	0x0	AHB data error
DE	0x00000080	0x0	Descriptor error

Field	Mask	Reset	Description
NumDMAChans	0x00000070	0x0	Number of DMA channels
ACTIVE	0x00000008	0x0	DMA is active
DIS	0x00000004	0x0	DMA disable
IE	0x00000002	0x0	Interrupt enable
EN	0x00000001	0x0	DMA enable

Register PCI2AHB

Description

DMA channel active

Reset value

0x00000000

Warm reset mask

0x00000000

Field	Mask	Reset	Description
-	-	-	-

Register STATCAP

Description

Status and Capability register

Reset value

0x00000000

Warm reset mask

0x003ff000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Host	MST	TAR	DMA	DI	HI	IRQmode		Trace	-	FHC	FGD	DPGER	CoreInterruptStatus							HostInterruptStatus				-			FDEPTH			FNUM	
ro	ro	ro	ro	ro	ro	ro		ro	-	ro	ro	rw	rw							ro				-			ro			ro	
-	-	-	-	-	-	-		-	-	0	0	0	0							-				-			-			-	

Field	Mask	Reset	Description
Host	0x80000000	0x0	Core allowed ot act as system host
MST	0x40000000	0x0	Master implemented
TAR	0x20000000	0x0	Target implemented
DMA	0x10000000	0x0	DMA implemented
DI	0x08000000	0x0	Device drives PCI INTA

Field	Mask	Reset	Description
HI	0x04000000	0x0	Device samples PCI INTA..D
IRQmode	0x03000000	0x0	APB IRQ mode
Trace	0x00800000	0x0	PCI trace buffer implemented
FH	0x00200000	0x0	Fake device in system slot
CFGDO	0x00100000	0x0	PCI config access done
CFGER	0x00080000	0x0	Error during PCI config access
CoreInterruptStatus	0x0007f000	0x0	Interrupt status
HostInterruptStatus	0x00000f00	0x0	Host interrupt status
FDEPTH	0x0000001c	0x0	Words in each FIFO
FNUM	0x00000003	0x0	Number of FIFOs

Commands

Name	Description
delete	Dispose instance of GRPCI2

9.8.5. Limitations

- The device currently only supports one AHB master. Since most systems only have one (e.g. an AHB2AHB bridge), this limitation should not be a major issue.

9.9. GRSPW1

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

9.9.1. Loading the Plugin

```
import Grspw1
```

9.9.2. Configuration

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

There are several configuration parameters in the GrSpw1 device. They are 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.

9.9.3. @Grspw1 Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object

Commands

Name	Description
delete	Dispose instance of @Grspw1
new	Create new instance of Grspw1

Command new Arguments

Name	Type	Required	Description
name	string	yes	Name of object to create

9.9.4. Grspw1 Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object
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.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	temu_IfaceRef/ <unknown>	Irq controller
memAccess	temu_IfaceRef/ <unknown>	Memory used for DMA accesses
memory	temu_IfaceRef/ <unknown>	Memory used for DMA accesses (deprecated)
pnp.bar	uint32_t	Pnp BAR
pnp.config	uint32_t	Pnp configuration
regs.clockDiv	uint32_t	Clock Divisor register

Name	Type	Description
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
regs.statusIrqSrc	uint32_t	Status / Interrupt-source register
regs.time	uint32_t	Time register
spwUplink	[temu_IfaceRef; 2]/ <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

Commands

Name	Description
delete	Dispose instance of Grspw1

9.9.5. Limitations

The following 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 = 0* and *PS / NP* 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. These are the only values that will be visible on the status register.
- RMAPEN signals not available.

9.9.6. Examples

This example show how to create two Grspw1 devices and connect them:

Listing 18. Creating and Connecting GRSPW1 Devices

```
import BusModels
import TEMUGrspw1
Grspw1.new name=grspw0
Grspw1.new name=grspw1
spw-connect port1=grspw0:SpwPortIface[0] port2=grspw1:SpwPortIface[0]
```

9.10. GRSPW2

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

9.10.1. Loading the Plugin

```
import Grspw2
```

9.10.2. Configuration

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

There are several configuration parameters in the GrSpw2 device. These are 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.

Name	Description
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.

9.10.3. @Grspw2 Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object

Commands

Name	Description
delete	Dispose instance of @Grspw2
new	Create new instance of Grspw2

Command new Arguments

Name	Type	Required	Description
name	string	yes	Name of object to create

9.10.4. Grspw2 Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object
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	temu_IfaceRef/ <unknown>	Irq controller
memAccess	temu_IfaceRef/ <unknown>	Memory used for DMA accesses
memory	temu_IfaceRef/ <unknown>	Memory used for DMA accesses (deprecated)

Name	Type	Description
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	[uint32_t; 4]	Dma address registers
regs.dmaControl	[uint32_t; 4]	Dma control registers
regs.dmaRxDescTableAddr	[uint32_t; 4]	Dma receive descriptor table address registers
regs.dmaRxMaxLen	[uint32_t; 4]	Dma rx maximum length registers
regs.dmaTxDescTableAddr	[uint32_t; 4]	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	[temu_IfaceRef; 2]/ <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

Commands

Name	Description
delete	Dispose instance of Grspw2

9.10.5. Limitations

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

- 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. These are the only values that will be visible on the status register.
- RMAPEN and PNPEN signals not available.

9.10.6. Examples

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

Listing 19. Creating and Connecting GRSPW2 Devices

```
import BusModels
import TEMUGrspw2
Grspw2.new name=grspw0
Grspw2.new name=grspw1
spw-connect port1=grspw0:SpwPortIface[0] port2=grspw1:SpwPortIface[0]
```

9.11. IRQAMP

The IRQAMP 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.

The device supports the boot address registers.

9.11.1. Loading the Plugin

```
import IrqAMp
```

9.11.2. Configuration

9.11.3. @IRQAMP Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info

Name	Type	Description
Name	*char	Object name
TimeSource	*void	Time source object

Commands

Name	Description
delete	Dispose instance of @IRQAMP
new	Create new instance of IRQAMP

Command new Arguments

Name	Type	Required	Description
name	string	yes	Name of object to create

9.11.4. IRQAMP Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object
bootAddress	[uint32_t; 16]	Boot addresses
broadcast	uint32_t	
config.bootReg	uint8_t	Enable boot address registers
config.enExtIrq	uint8_t	
config.irqMapping	uint8_t	Enable interrupt mapping
config.logInterrupts	uint8_t	
config.nCpu	uint8_t	
config.traceReads	uint8_t	
config.traceWrites	uint8_t	
cpu	[temu_IfaceRef; 16]/ <unknown>	Processors

Name	Type	Description
dynamicReset	[temu_IfaceRef; 16]/ <unknown>	Dynamic reset address interface (typically implemented by CPU)
extIntAck	[uint32_t; 16]	
force	[uint32_t; 16]	
interruptMap	[uint32_t; 8]	
irqClear	uint32_t	
irqCtrl	[temu_IfaceRef; 16]/ <unknown>	Upstream interrupt controllers
irqForce0	uint32_t	
irqLevel	uint32_t	
irqPending	uint32_t	
mask	[uint32_t; 16]	
mpStatus	uint32_t	
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

Commands

Name	Description
delete	Dispose instance of IRQAMP

Name	Description
raiseExternalIrq	Raise interrupt

Command raiseExternalIrq Arguments

Name	Type	Required	Description
irq	int	yes	Interrupt number

9.11.5. 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 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 for some software.
- The model does not verify that register writes come from the correct CPU at the moment.

9.12. IRQMP

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

The controller supports routing interrupts to different processor cores and broadcasted interrupts.

9.12.1. Loading the Plugin

```
import IrqMp
```

9.12.2. 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.

9.12.3. @IrqMp Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object

Commands

Name	Description
delete	Dispose instance of @IrqMp
new	Create new instance of IrqMp

Command new Arguments

Name	Type	Required	Description
name	string	yes	Name of object to create

9.12.4. IrqMp Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object
bootAddress	[uint32_t; 16]	
broadcast	uint32_t	
config.bootReg	uint8_t	Enable boot address registers
config.enExtIrq	uint8_t	
config.irqMapping	uint8_t	Enable interrupt mapping
config.logInterrupts	uint8_t	

Name	Type	Description
config.nCpu	uint8_t	
config.traceReads	uint8_t	
config.traceWrites	uint8_t	
cpu	[temu_IfaceRef; 16]/ <unknown>	Processors
extIntAck	[uint32_t; 16]	
force	[uint32_t; 16]	
interruptMap	[uint32_t; 8]	
irqClear	uint32_t	
irqCtrl	[temu_IfaceRef; 16]/ <unknown>	Upstream interrupt controllers (e.g. processor)
irqForce0	uint32_t	
irqLevel	uint32_t	
irqPending	uint32_t	
mask	[uint32_t; 16]	
mpStatus	uint32_t	
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

Commands

Name	Description
delete	Dispose instance of IrqMp
raiseExternalIrq	Raise interrupt

Command raiseExternalIrq Arguments

Name	Type	Required	Description
irq	int	yes	Interrupt number

9.12.5. 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-synchronized event, the interrupt is raised at different times on the different cores. Consider the IRQ frequency and the configured quanta length.

Chapter 10. P2020

10.1. CCSRGU

This section describes the model of the P2020 CCSR General Utilities model. The device models the General Utilities part in CCSR space. CCSRGU includes the local configuration control and local access parts, handling the remapping of the dynamic physical address space.

10.1.1. Loading the Plugin

```
import P2020
```

10.1.2. Configuration

10.1.3. @CCSRGU Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object

Commands

Name	Description
delete	Dispose instance of @CCSRGU
new	Create new instance of CCSRGU

Command new Arguments

Name	Type	Required	Description
name	string	yes	Name of object to create

10.1.4. CCSRGU Reference

Properties

Name	Type	Description
ALTCAR	uint32_t	ALTCAR register
ALTCBAR	uint32_t	ALTCBAR register
BPTR	uint32_t	BPTR register
CCSRBAR	uint32_t	CCSRBAR register
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LAWAR	[uint32_t; 12]	LAWAR register
LAWBAR	[uint32_t; 12]	LAWBAR register
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object
ddr	temu_IfaceRef/ <unknown>	DDR memory controller
elbc	temu_IfaceRef/ <unknown>	Enhanced local bus
memSpace	temu_IfaceRef/ <unknown>	Memory space
pci	[temu_IfaceRef; 3]/ <unknown>	PCI bridges
srio	[temu_IfaceRef; 2]/ <unknown>	SRIO

Interfaces

Name	Type	Description
MemAccessIface	MemAccessIface	Memory access interface

Commands

Name	Description
delete	Dispose instance of CCSRGU

10.1.5. Limitations

10.2. DDR

This section describes the DDR controller model.

The model supports the handling of SEU and MEU events (i.e. correctable and uncorrectable ECC errors). Such events will update the counters and raise interrupts accordingly.

10.2.1. Loading the Plugin

```
import P2020
```

10.2.2. Configuration

To use SEU and MEU events connect the memoryspace upset/faulty handlers to the DDR model and allow interrupts as follows:

Listing 20. Connecting SEU and MEU events via Command Line

```
# Connect the memoryspace upset/faulty handlers to the ddr
connect a=mem.upsetHandlers b=ddr:CorrectableErrorIface
connect a=mem.faultyHandlers b=ddr:UncorrectableErrorIface

# Enable ECC error interrupts
ddr.ERR_INT_EN = 0x0000000c

# Set/clear faulty attribute on the given memory range
memory-set-faulty addr=0x08 obj=mem
memory-clear-faulty addr=0x08 obj=mem

# Set/clear upset attribute on the given memory range
memory-set-upset addr=0x08 obj=mem
memory-clear-upset addr=0x08 obj=mem
```

Listing 21. Connecting SEU and MEU events via API

```
// Connect the memoryspace upset/faulty handlers to the ddr
temu_connect(mem, "upsetHandlers", ddr, "CorrectableErrorIface");
temu_connect(mem, "faultyHandlers", ddr, "UncorrectableErrorIface");

// Enable ECC error interrupts
temu_writeValueU32(ddr, "ERR_INT_EN", 0x0000000c, 0);

// Set/clear faulty attribute on the given memory range
temu_memorySetAttr(mem, 0x08, 4, teMA_Faulty);
temu_memoryClearAttr(mem, 0x08, 4, teMA_Faulty);

// Set/clear upset attribute on the given memory range
temu_memorySetAttr(mem, 0x08, 4, teMA_Upset);
temu_memoryClearAttr(mem, 0x08, 4, teMA_Upset);
```

10.2.3. @DDR Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object

Commands

Name	Description
delete	Dispose instance of @DDR
new	Create new instance of DDR

Command new Arguments

Name	Type	Required	Description
name	string	yes	Name of object to create

10.2.4. DDR Reference

Properties

Name	Type	Description
CAPTURE_ADDRESS	uint32_t	
CAPTURE_ATTRIBUTES	uint32_t	
CAPTURE_DATA_HI	uint32_t	
CAPTURE_DATA_LO	uint32_t	
CAPTURE_ECC	uint32_t	
CAPTURE_EXT_ADDRESS	uint32_t	
CS_BNDS	[uint32_t; 4]	
CS_CONFIG	[uint32_t; 4]	
CS_CONFIG_2	[uint32_t; 4]	
Class	*void	Class object
Component	*void	Pointer to component object if part of component
DATA_ERR_INJECT_HI	uint32_t	

Name	Type	Description
DATA_ERR_INJECT_LO	uint32_t	
DATA_INIT	uint32_t	
DDRCDR	[uint32_t; 2]	
DDRDSR	[uint32_t; 2]	
ERR_DETECT	uint32_t	
ERR_DISABLE	uint32_t	
ERR_INJECT	uint32_t	
ERR_INT_EN	uint32_t	
ERR_SBE	uint32_t	
INIT_ADDR	uint32_t	
INIT_EXT_ADDR	uint32_t	
IP_REV	[uint32_t; 2]	
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
SDRAM_CFG	[uint32_t; 2]	
SDRAM_CLK_CNTL	uint32_t	
SDRAM_INTERVAL	uint32_t	
SDRAM_MD_CNTL	uint32_t	
SDRAM_MODE	[uint32_t; 2]	
SDRAM_RCW	[uint32_t; 2]	
SR_CNTR	uint32_t	
TIMING_CFG	[uint32_t; 6]	
TimeSource	*void	Time source object
WRLVL_CNTL	[uint32_t; 3]	
ZQ_CNTL	uint32_t	
config.IRQ	uint8_t	
irqCtrl	temu_IfaceRef/ <unknown>	Interrupt controller
memorySpace	temu_IfaceRef/ <unknown>	Memory space.

Interfaces

Name	Type	Description
CorrectableErrorIface	MemAccessIface	

Name	Type	Description
DeviceIface	DeviceIface	
MemAccessIface	MemAccessIface	
ResetIface	ResetIface	
UncorrectableErrorIface	MemAccessIface	

Commands

Name	Description
delete	Dispose instance of DDR

10.2.5. Limitations

Only registers handling SEU and MEU events are implemented, other parts of the model are dummy non-functional registers.

10.3. DMA

This section describes the P2020 DMA model.

The DMA controller transfers blocks of data between the interface and functional modules of this chip, independent of the cores or external hosts. It has four high-speed DMA channels. Both the cores and external devices can initiate DMA transfers.

The DMA module has two modes of operation: basic and extended. Basic mode is the DMA legacy mode, which does not support advanced features. Extended mode supports advanced features such as striding and flexible descriptor structures. These two basic modes allow users to initiate and end DMA transfers in various ways:

- Direct mode: no descriptors are involved, software/user must initialize such required fields as source address and attributes and destination address and attributes.
- Chaining mode: software must initialize descriptors in memory and such fields as link/list descriptor address before starting a transfer.
- Single-write start mode: DMA process can be started using a single-write command to either the descriptor address register in one of the chaining modes or the source/destination address registers in one of the direct modes.

10.3.1. Loading the Plugin

```
import P2020
```

10.3.2. Configuration

The DMA programmable registers that occupy memory-mapped space are named according to P2020 QorIQ integrated processor reference manual. All 4 DMA channels have the same set of registers, so the register index is the channels id (from 0 to 3).

Listing 22. Setting registers via Command Line

```
# set source address for channel 3
dma.SAR[3] = 0x40000000
```

Listing 23. Setting registers via API

```
// set source address for channel 3
temu_writeValueU32(dma, "SAR", 0x40000000, 3);
```

10.3.3. @DMA Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object

Commands

Name	Description
delete	Dispose instance of @DMA
new	Create new instance of DMA

Command new Arguments

Name	Type	Required	Description
name	string	yes	Name of object to create

10.3.4. DMA Reference

Properties

Name	Type	Description
BCR	[uint32_t; 4]	DMA n byte count register
CLNDAR	[uint32_t; 4]	DMA n current link descriptor address register
CLSDAR	[uint32_t; 4]	DMA n current list descriptor address register
Class	*void	Class object
Component	*void	Pointer to component object if part of component
DAR	[uint32_t; 4]	DMA n destination address register
DATR	[uint32_t; 4]	DMA n destination attributes register
DSR	[uint32_t; 4]	DMA n destination stride register
ECLNDAR	[uint32_t; 4]	DMA n current link descriptor extended address register
ECLSDAR	[uint32_t; 4]	DMA n extended current list descriptor address register
ENLNDAR	[uint32_t; 4]	DMA n extended next link descriptor address register
ENLSDAR	[uint32_t; 4]	DMA n extended next list descriptor address register
LoggingFlags	uint64_t	Flags for logging info
MR	[uint32_t; 4]	DMA n mode register
NLNDAR	[uint32_t; 4]	DMA n next link descriptor address register
NLSDAR	[uint32_t; 4]	DMA n next list descriptor address register
Name	*char	Object name
SAR	[uint32_t; 4]	DMA n source address register
SATR	[uint32_t; 4]	DMA n source attributes register
SR	[uint32_t; 4]	DMA n status register
SSR	[uint32_t; 4]	DMA n source stride register
TimeSource	*void	Time source object

Name	Type	Description
config.interrupt	[uint8_t; 4]	DMA IRQ number for each channel
irqCtrl	temu_IfaceRef/ <unknown>	Interrupt controller
memAccess	temu_IfaceRef/ <unknown>	Memory access for DMA.

Interfaces

Name	Type	Description
MemAccessIface	MemAccessIface	Memory access interface.

Commands

Name	Description
delete	Dispose instance of DMA

10.3.5. Limitations

None

10.4. DUART

This section describes the P2020 serial port (DUART) model. The DUART 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. FIFO mode for both transmitter and receiver provides 16-byte FIFOs.

10.4.1. Loading the Plugin

```
import P2020
```

10.4.2. Configuration

config.infiniteUartSpeed

Enable/disable immediate UART.

config.fifoSize

maximum UART FIFO size.

config.clockDivider

Clock devider (default 1).

To receive transmitted data from DUART device can be connected via the serial interface it implements:

Listing 24. Connecting via Command Line

```
# Connect DUART transmitter to device model
connect a=duart.tx b=mydevice:SerialIface
```

Listing 25. Connecting via API

```
// Connect DUART transmitter to device model
temu_connect(duart, "tx", mydevice, "SerialIface");
```

The DUART programmable registers that occupy memory-mapped space are named according to P2020 QorIQ integrated processor reference manual. All the DUART registers are one-byte wide.

To enables the transmitter and receiver FIFOs you need to set FIFO Control Register value as follows:

Listing 26. Setting registers via Command Line

```
duart.UFCR = 0x01;
```

Listing 27. Setting registers via API

```
temu_writeValueU8(duart, "UFCR", 0x01, 0);
```

10.4.3. @DUART Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object

Commands

Name	Description
delete	Dispose instance of @DUART
new	Create new instance of DUART

Command new Arguments

Name	Type	Required	Description
name	string	yes	Name of object to create

10.4.4. DUART Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object
UAFR	uint8_t	UART AlternateFunction register.
UDLB	uint8_t	UART DivisorLeastSignificantByte register.
UDMB	uint8_t	UART DivisorMostSignificantByte register.
UDSR	uint8_t	DUART DmaStatus register
UFCR	uint8_t	UART FifoControl register.
UIER	uint8_t	UART InterruptEnable register.
UIIR	uint8_t	UART InterruptId register.
ULCR	uint8_t	UART LineControl register.
ULSR	uint8_t	DUART LineStatus register.
UMCR	uint8_t	UART ModemControl register.
UMSR	uint8_t	DUART ModemStatus register.
URBR	uint8_t	UART ReceiverBuffer register.
USCR	uint8_t	DUART Scratch register
UTHR	uint8_t	UART TransmitterHolding register.
config.clockDivider	uint32_t	Clock divider
config.fifoSize	uint8_t	UART FIFO size

Name	Type	Description
config.infiniteUartSpeed	uint8_t	Set to 1 to enable immediate UARTs
config.interrupt	uint8_t	Interrupt number
irqCtrl	temu_IfaceRef/ <unknown>	Interrupt controller
rxFifo.data	[uint8_t; 32]	RX FIFO data
rxFifo.size	uint8_t	RX size
rxFifo.start	uint8_t	RX start index
rxFifo.usage	uint8_t	RX usage
tx	temu_IfaceRef/ <unknown>	Transmit target
txFifo.data	[uint8_t; 32]	TX FIFO data
txFifo.size	uint8_t	TX size
txFifo.start	uint8_t	TX start index
txFifo.usage	uint8_t	TX usage
txShift	uint8_t	UART shift register

Interfaces

Name	Type	Description
DeviceIface	DeviceIface	Device interface.
MemAccessIface	MemAccessIface	Memory access interface.
ResetIface	ResetIface	Reset interface.
UartIface	SerialIface	Serial input interface.

Ports

Prop	Iface	Description
tx	UartIface	serial port

Commands

Name	Description
delete	Dispose instance of DUART

10.4.5. Limitations

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

10.5. ECM

This section describes the P2020 ECM model. The model is a dummy providing only non-functional registers.

10.5.1. Loading the Plugin

```
import P2020
```

10.5.2. Configuration

10.5.3. @ECM Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object

Commands

Name	Description
delete	Dispose instance of @ECM
new	Create new instance of ECM

Command new Arguments

Name	Type	Required	Description
name	string	yes	Name of object to create

10.5.4. ECM Reference

Properties

Name	Type	Description
Class	*void	Class object

Name	Type	Description
Component	*void	Pointer to component object if part of component
EEATR	uint32_t	
EEBACR	uint32_t	
EEBPCR	uint32_t	
EEDR	uint32_t	
EEER	uint32_t	
EEHADR	uint32_t	
EELADR	uint32_t	
EIPBRR1	uint32_t	
EIPBRR2	uint32_t	
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object

Interfaces

Name	Type	Description
DeviceIface	DeviceIface	
MemAccessIface	MemAccessIface	
ResetIface	ResetIface	

Commands

Name	Description
delete	Dispose instance of ECM

10.5.5. Limitations

The model is a dummy providing only non-functional registers.

10.6. eSPI

This section describes the P2020 Enhanced Serial Peripheral Interface (eSPI) model.

The eSPI is a full-duplex, synchronous, character-oriented channel that supports a simple interface (receive, transmit and chip selects).

eSPI model contains names which include master/slave terminology. The TEMU

project does not use such terms normally, however given that the hardware specification, this header deals with, use these terms they are reused here for consistency.

10.6.1. Loading the Plugin

```
import P2020
```

10.6.2. Configuration

Master device does not have a direct access to slave devices(and vice versa). Transmission and reception process is done through bus, the bus route on chip select signals.

The eSPI bus model can be configured by connecting SPI slave device interface to the spiSlaveDevices array, and master device to the spiMasterDevice.

Listing 28. Connecting via Command Line

```
# Connect eSPI device to eSPI bus via SpiSlaveDeviceIface
connect a=spiBus.spiSlaveDevices b=spidev:SpiSlaveDeviceIface
# Connect master interface from eSPI model to the bus
connect a=spiBus.spiMasterDevice b=espi:MasterDeviceIface
# Connect bus to the eSPI model
connect a=espi.spiBus b=spiBus:SpiBusIface
# Connect bus and the eSPI device
connect a=spidev.spiBus, spiBus:SpiBusIface
```

Listing 29. Connecting via API

```
// Connect eSPI device to eSPI bus via SpiSlaveDeviceIface
temu_connect(spiBus, "spiSlaveDevices", spidev, "SpiSlaveDeviceIface");
// Connect master interface from eSPI model to the bus
temu_connect(spiBus, "spiMasterDevice", espi, "MasterDeviceIface");
// Connect bus to the eSPI model
temu_connect(espi, "spiBus", spiBus, "SpiBusIface");
// Connect bus and the eSPI device
temu_connect(spidev, "spiBus", spiBus, "SpiBusIface");
```

The eSPI programmable registers that occupy memory-mapped space are named according to P2020 QorIQ integrated processor reference manual. All the registers are 4-byte wide:

Listing 30. Setting registers via Command Line

```
# Configure SPMODE register to enable normal operation
espi.SPMODE = 0x8000100F;
```

Listing 31. Setting registers via API

```
// Configure SPMODE register to enable normal operation
temu_writeValueU32(espi, "SPMODE", 0x8000100F, 0);
```

10.6.3. @eSPI Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object

Commands

Name	Description
delete	Dispose instance of @eSPI
new	Create new instance of eSPI

Command new Arguments

Name	Type	Required	Description
name	string	yes	Name of object to create

10.6.4. eSPI Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
SPCOM	uint32_t	eSPI command register

Name	Type	Description
SPIE	uint32_t	eSPI event register
SPIM	uint32_t	eSPI mask register
SPIRF	uint32_t	eSPI receive FIFO access register
SPITF	uint32_t	eSPI transmit FIFO access register
SPMODE	uint32_t	eSPI mode register
SPMODE0	uint32_t	eSPI CS0 mode register
SPMODE1	uint32_t	eSPI CS1 mode register
SPMODE2	uint32_t	eSPI CS2 mode register
SPMODE3	uint32_t	eSPI CS3 mode register
TimeSource	*void	Time source object
chipType	uint8_t	Chip type
config.interrupt	uint8_t	eSPI IRQ number
irqCtrl	temu_IfaceRef/ <unknown>	Interrupt controller
rxFifo.data	[uint8_t; 32]	RX FIFO data
rxFifo.size	uint8_t	RX size
rxFifo.start	uint8_t	RX start index
rxFifo.usage	uint8_t	RX usage
spiBus	temu_IfaceRef/ <unknown>	Spi bus
txFifo.data	[uint8_t; 32]	TX FIFO data
txFifo.size	uint8_t	TX size
txFifo.start	uint8_t	TX start index
txFifo.usage	uint8_t	TX usage

Interfaces

Name	Type	Description
MasterDeviceIface	temu::SpiMasterDeviceIface	Communication interface.
MemAccessIface	MemAccessIface	Memory access interface.

Commands

Name	Description
delete	Dispose instance of eSPI

10.6.5. Limitations

- eSPI doesn't generate the transfer clock SPI_CLK signal and doesn't implement the eSPI baud rate generator (BRG).

10.7. eTSEC

This section describes the P2020 eTSEC Ethernet controller model.

The eTSECs of the device include these distinctive features:

- TCP/IP off-load:
  - IP v4 and IP v6 header recognition on receive;
  - IP v4 header checksum verification and generation;
  - TCP and UDP checksum verification and generation;
  - Per-packet configurable off-load.
- Support for different Ethernet physical interfaces (MII, GMII, RMII, RGMII, TBI and RTBI.).

10.7.1. Loading the Plugin

```
import P2020
```

10.7.2. Configuration

config.interfaceMode

Interface Mode (MII 10/100 Mbps, RMII 100 Mbps, RMII 10 Mbps, GMII 1Gbps, etc).

The GenericPHY is a PHY / MII device which supports both the MDIO interface and the PHY interface for sending/receiving ethernet frames. It is connected to MDIO bus via MDIOIface and to eTSEC via PHYIface. An ethernet link must be connected to its attached PHYs via EthernetIface;

The MDIO bus distributes MDIO control messages and supports routing of them. The MDIO bus use the same interface as an MDIO device. It is connected to eTSEC via MDIOIface.

Before starting the communications all mentioned above models should be created and connected as follows:

Listing 32. Setting Ethernet controller via API

```
// connect MDIO bus with MAC controller and PHY circuit.
temu_connect(miibus, "macDevice", etsec, "MACIface");
temu_connect(miibus, "phyDevices", phy, "MDIOIface");
// connect eTSEC with MDIO bus and PHY
temu_connect(etsec, "mdioBus", miibus, "MDIOIface");
temu_connect(etsec, "phy", phy, "PHYIface");
```

```
// connect MAC device to PHY
temu_connect(phy, "macDevice", etsec, "MACIface");
// connect ethernet link
temu_connect(phy, "ethernetLink", ethlink, "EthernetIface");
```

Listing 33. Setting Ethernet controller via Command Line

```
# connect MDIO bus with MAC controller and PHY circuit.
connect a=miibus.macDevice b=etsec:MACIface
connect a=miibus.phyDevices b=phy:MDIOIface

# connect eTSEC with MDIO bus and PHY
connect a=etsec.mdioBus b=miibus:MDIOIface
connect a=etsec.phy b=phy:PHYIface

#  connect MAC device to PHY
connect a=phy.macDevice b=etsec:MACIface
# connect ethernet link
connect a=phy.ethernetLink b=ethlink:EthernetIface
```

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

Listing 34. Connect Syntax

```
ethlink.connect device=phy:PHYIface
```

The eTSEC programmable registers that occupy memory-mapped space are named according to P2020 QorIQ integrated processor reference manual. All the eTSEC registers are 4-byte wide.

To set MAC address MACSTNADDR1 and MACSTNADDR2 registers should be filled. The value of the station address written into MACSTNADDR1 and MACSTNADDR2 is byte reversed from how it would appear in the DA field of a frame in memory. For example, for a station address of 0x12345678ABCD, MACSTNADDR1 is set to 0xCDAB7856 and MACSTNADDR2 is set to 0x34120000.

Listing 35. Setting registers via API

```
// Write MACSTNADDR1 register value to set MAC to 00:00:00:00:00:01
temu_writeValueU32(etsec, "MACSTNADDR1", 0x01000000, 0);
```

Listing 36. Setting registers via Command Line

```
# Write MACSTNADDR1 register value to set MAC to 00:00:00:00:00:01
etsec.MACSTNADDR1 = 0x01000000
```

MAC address also can be set via setMAC command:

Listing 37. Set MAC

```
etsec.setMAC mac=\"00:00:00:00:00:01\"
```

10.7.3. @eTSEC Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object

Commands

Name	Description
delete	Dispose instance of @eTSEC
new	Create new instance of eTSEC

Command new Arguments

Name	Type	Required	Description
name	string	yes	Name of object to create

10.7.4. eTSEC Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
ETSEC_1588_TMR_ACC	uint32_t	Timer accumulator register
ETSEC_1588_TMR_ADD	uint32_t	Timer rift compensation addend register

Name	Type	Description
ETSEC_1588_TMR_ALARM_H	[uint32_t; 2]	Timer offset high register
ETSEC_1588_TMR_ALARM_L	[uint32_t; 2]	Timer offset low register
ETSEC_1588_TMR_CNT_H	uint32_t	Timer counter high register
ETSEC_1588_TMR_CNT_L	uint32_t	Timer counter low register
ETSEC_1588_TMR_CTRL	uint32_t	Timer control register
ETSEC_1588_TMR_ETTS_H	[uint32_t; 2]	Time stamp of general purpose external trigger(high)
ETSEC_1588_TMR_ETTS_L	[uint32_t; 2]	Time stamp of general purpose external trigger(low)
ETSEC_1588_TMR_FIPER	[uint32_t; 2]	Timer fixed period interval
ETSEC_1588_TMR_PEMASK	uint32_t	Timer event mask register
ETSEC_1588_TMR_PEVENT	uint32_t	Timer event mask register
ETSEC_1588_TMR_PRSC	uint32_t	Timer prescale
ETSEC_1588_TMR_STAT	uint32_t	Time stamp status register
ETSEC_1588_TMR_TEMASK	uint32_t	Timer event mask register
ETSEC_1588_TMR_TEVENT	uint32_t	Time stamp event register
ETSEC_1588_TMR_TMROFF_H	uint32_t	Timer offset high register
ETSEC_1588_TMR_TMROFF_L	uint32_t	Timer offset low register
ETSEC_ATTR	uint32_t	ETSEC DMA Attribute register
ETSEC_ATTRELI	uint32_t	ETSEC DMA Attribute extract length and extract index register
ETSEC_CAM	[uint32_t; 2]	ETSEC Carry registers(0,1) mask
ETSEC_CAR	[uint32_t; 2]	ETSEC Carry registers(0,1)
ETSEC_DFVLAN	uint32_t	ETSEC Default VLAN control word
ETSEC_DMACTRL	uint32_t	ETSEC DMA control register
ETSEC_ECNTRL	uint32_t	ETSEC Ethernet control register
ETSEC_EDIS	uint32_t	ETSEC Error disabled register
ETSEC_GADDR	[uint32_t; 16]	ETSEC group address register n
ETSEC_HAFDUP	uint32_t	ETSEC Half-duplex control register
ETSEC_IEVENT	uint32_t	ETSEC Interrupt event register
ETSEC_IFSTAT	uint32_t	ETSEC Interface status

Name	Type	Description
ETSEC_IGADDR	[uint32_t; 16]	ETSEC Individual/group address register n
ETSEC_IMASK	uint32_t	ETSEC Interrupt mask register
ETSEC_IPGIFG	uint32_t	ETSEC Inter-packet/inter-frame gap register
ETSEC_MACCFG	[uint32_t; 2]	ETSEC MAC configuration register (0-1)
ETSEC_MACSTNADDR	[uint32_t; 2]	ETSEC MAC station address registers
ETSEC_MACnADDR1	[uint32_t; 16]	ETSEC MAC exact match address n, part 1
ETSEC_MACnADDR2	[uint32_t; 16]	ETSEC MAC exact match address n, part 2
ETSEC_MAXFRM	uint32_t	ETSEC Maximum frame length
ETSEC_MIIMADD	uint32_t	ETSEC MII management address
ETSEC_MIIMCFG	uint32_t	ETSEC MII management configuration
ETSEC_MIIMCOM	uint32_t	ETSEC MII management command
ETSEC_MIIMCON	uint32_t	ETSEC MII management control
ETSEC_MIIMIND	uint32_t	ETSEC MII management indicator
ETSEC_MIIMSTAT	uint32_t	ETSEC MII management status
ETSEC_MRBLR	uint32_t	ETSEC Maximum receive buffer length register
ETSEC_PVT	uint32_t	ETSEC Pause time value register
ETSEC_RALN	uint32_t	ETSEC Receive alignment error counter
ETSEC_RBASE	[uint32_t; 8]	ETSEC base address of ring n
ETSEC_RBASEH	uint32_t	ETSEC RxBD base address high bits
ETSEC_RBCA	uint32_t	ETSEC Receive broadcast packet counter
ETSEC_RBDBPH	uint32_t	ETSEC Rx data buffer pointer high bits register

Name	Type	Description
ETSEC_RBIFX	uint32_t	ETSEC Receive bit field extract control register
ETSEC_RBPTR	[uint32_t; 8]	ETSEC RxBD pointer for ring n
ETSEC_RBYT	uint32_t	ETSEC Receive byte counter
ETSEC_RCDE	uint32_t	ETSEC Receive code error counter
ETSEC_RCSE	uint32_t	ETSEC Receive carrier sense error counter
ETSEC_RCTRL	uint32_t	ETSEC Receive control register
ETSEC_RDRP	uint32_t	ETSEC Receive drop counter
ETSEC_RFBPTR	[uint32_t; 8]	ETSEC Receive Queue Parameters register n
ETSEC_RFCS	uint32_t	ETSEC Receive FCS error counter
ETSEC_RFLR	uint32_t	ETSEC Receive frame length error counter
ETSEC_RFRG	uint32_t	ETSEC Receive fragments counter
ETSEC_RJBR	uint32_t	ETSEC Receive jabber counter
ETSEC_RMCA	uint32_t	ETSEC Receive multicast packet counter
ETSEC_ROVR	uint32_t	ETSEC Receive oversize packet counter
ETSEC_RPKT	uint32_t	ETSEC Receive packet counter
ETSEC_RQFAR	uint32_t	ETSEC Receive queue filing table address register
ETSEC_RQFCR	uint32_t	ETSEC Receive queue filing table control register
ETSEC_RQFPR	uint32_t	ETSEC Receive queue filing table property register
ETSEC_RQPRM	[uint32_t; 8]	ETSEC Receive Queue Parameters register n
ETSEC_RQUEUE	uint32_t	ETSEC Receive queue control register
ETSEC_RREJ	uint32_t	ETSEC Receive filer rejected packet counter

Name	Type	Description
ETSEC_RSTAT	uint32_t	ETSEC Receive status register
ETSEC_RUND	uint32_t	ETSEC Receive undersize packet counter
ETSEC_RXCF	uint32_t	ETSEC Receive control frame packet counter
ETSEC_RXIC	uint32_t	ETSEC Receive interrupt coalescing register
ETSEC_RXPF	uint32_t	ETSEC Receive PAUSE frame packet counter
ETSEC_RXUO	uint32_t	ETSEC Receive unknown OP code counter
ETSEC_TBASE	[uint32_t; 8]	ETSEC TxBD base address of ring n
ETSEC_TBASEH	uint32_t	ETSEC TxBD base address high bits
ETSEC_TBCA	uint32_t	ETSEC Transmit broadcast packet counter
ETSEC_TBDBPH	uint32_t	ETSEC Tx data buffer pointer high bits
ETSEC_TBIPA	uint32_t	ETSEC TBI PHY address register
ETSEC_TBPTR	[uint32_t; 8]	ETSEC TxBD pointer for ring n
ETSEC_TBYT	uint32_t	ETSEC Transmit byte counter
ETSEC_TCTRL	uint32_t	ETSEC Transmit control register
ETSEC_TDRF	uint32_t	ETSEC Transmit deferral packet counter
ETSEC_TDRP	uint32_t	ETSEC Transmit drop frame counter
ETSEC_TEDF	uint32_t	ETSEC Transmit excessive deferral packet counter
ETSEC_TFCS	uint32_t	ETSEC Transmit FCS error counter
ETSEC_TFRG	uint32_t	ETSEC Transmit fragments frame counter
ETSEC_TJBR	uint32_t	ETSEC Transmit jabber frame counter

Name	Type	Description
ETSEC_TLCL	uint32_t	ETSEC Transmit late collision packet counter
ETSEC_TMCA	uint32_t	ETSEC Transmit multicast packet counter
ETSEC_TMCL	uint32_t	ETSEC Transmit multi collision packet counter
ETSEC_TMR_RXTS_H	uint32_t	ETSEC Rx timer time stamp register high
ETSEC_TMR_RXTS_L	uint32_t	ETSEC Rx timer time stamp register low
ETSEC_TMR_TXTS_H	[uint32_t; 2]	ETSEC Tx Tx time stamp high
ETSEC_TMR_TXTS_ID	[uint32_t; 2]	ETSEC Tx time stamp identification tag (set n) register
ETSEC_TMR_TXTS_L	[uint32_t; 2]	ETSEC Tx Tx time stamp low
ETSEC_TNCL	uint32_t	ETSEC Transmit total collision packet counter
ETSEC_TOVR	uint32_t	ETSEC Transmit oversize frame counter
ETSEC_TPKT	uint32_t	ETSEC Transmit packet counter
ETSEC_TQUEUE	uint32_t	ETSEC Transmit queue control register
ETSEC_TR03WT	uint32_t	ETSEC TxBD Rings 0-3 round-robin weightings
ETSEC_TR127	uint32_t	ETSEC Transmit and receive 65- to 127-byte frame counter
ETSEC_TR1K	uint32_t	ETSEC Transmit and receive 512- to 1023-byte frame counter
ETSEC_TR255	uint32_t	ETSEC Transmit and receive 128- to 255-byte frame counter
ETSEC_TR47WT	uint32_t	ETSEC TxBD Rings 4-7 round-robin weightings
ETSEC_TR511	uint32_t	ETSEC Transmit and receive 256- to 511-byte frame counter
ETSEC_TR64	uint32_t	ETSEC Transmit and receive 64-byte frame counter

Name	Type	Description
ETSEC_TRMAX	uint32_t	ETSEC Transmit and receive 1024- to 1518-byte frame counter
ETSEC_TSCL	uint32_t	ETSEC Transmit single collision packet counter
ETSEC_TSEC_ID	[uint32_t; 2]	ETSEC Controller ID registers (0, 1)
ETSEC_TSTAT	uint32_t	ETSEC Transmit status register
ETSEC_TUND	uint32_t	ETSEC Transmit undersize frame counter
ETSEC_TXCF	uint32_t	ETSEC Transmit control frame counter
ETSEC_TXCL	uint32_t	ETSEC Transmit excessive collision packet counter
ETSEC_TXIC	uint32_t	ETSEC Transmit interrupt coalescing register
ETSEC_TXPF	uint32_t	ETSEC Transmit PAUSE control frame counter
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object
config.checkCrc	uint8_t	Enable ethernet frame CRC checking.
config.generateCrc	uint8_t	Enable ethernet frame CRC generation.
config.interfaceMode	uint8_t	Set interface mode
config.irqError	uint8_t	IRQ number for eTSEC error
config.irqReceive	uint8_t	IRQ number for eTSEC receieve event
config.irqTransmit	uint8_t	IRQ number for eTSEC transmit event
config.logTraffic	uint8_t	Enable traffic logging
irqCtrl	temu_IfaceRef/ <unknown>	IRQ controller
mac	*char	Set MAC by string
mdioBus	temu_IfaceRef/ <unknown>	MDIO bus
memAccess	temu_IfaceRef/ <unknown>	Memory access (for DMA).

Name	Type	Description
memory	temu_IfaceRef/ <unknown>	Memory
phy	temu_IfaceRef/ <unknown>	PHY device

Interfaces

Name	Type	Description
MACIface	temu::MACIface	MAC interface
MemAccessIface	MemAccessIface	Mem access interface

Commands

Name	Description
delete	Dispose instance of eTSEC
sendFrame	Send frame
setMAC	Set MAC address

Command sendFrame Arguments

Name	Type	Required	Description
mac	string	yes	MAC address of target

Command setMAC Arguments

Name	Type	Required	Description
mac	string	yes	MAC address to set

10.7.5. Limitations

- Multicast groups are not yet supported.

10.8. GPIO

This section describes the P2020 GPIO device.

GPIO model simulates a 16 pin GPIO device by providing input and output via the SignalIface. Each signal can be set individually to act as input or output, according to application needs. All input ports can optionally generate an interrupt upon changing their state.

10.8.1. Loading the Plugin

```
import P2020
```

10.8.2. Configuration

Any device that implements SignalIface can be connected to GPIO and send signals via this interface to GPIO or receive them. GPIO implements 16 usable signals (signal 0 through 15). You can connect the signal interface as follows:

Listing 38. Connecting via Command Line

```
# Connect GPIO device signal 0 to device model
connect a=gpio.outSignals[0] b=mydevice:SignalIface

# Connect a device signal interface ref to GPIO device
connect a=mydevice.signal b=gpio:SignalIface[1]
```

Listing 39. Connecting via API

```
// Connect GPIO device signal 0 to device model
temu_connect(gpio, "outSignals[0]", mydevice, "SignalIface");

// Connect a device signal interface ref to GPIO device
temu_connect(mydevice, "signal", gpio, "SignalIface[1]");
```

The GPIO programmable registers that occupy memory-mapped space are named according to P2020 QorIQ integrated processor reference manual. You can set resgisters value as follows:

Listing 40. Setting registers via Command Line

```
# Set signal_0 as an output
gpio.GPDIR = 0x80000000;
```

Listing 41. Setting registers via API

```
// Set signal_0 as an output
temu_writeValueU32(gpio, "GPDIR", 0x80000000, 0);
```

10.8.3. @GPIO Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info

Name	Type	Description
Name	*char	Object name
TimeSource	*void	Time source object

Commands

Name	Description
delete	Dispose instance of @GPIO
new	Create new instance of GPIO

Command new Arguments

Name	Type	Required	Description
name	string	yes	Name of object to create

10.8.4. GPIO Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
GPDAT	uint32_t	
GPDIR	uint32_t	
GPICR	uint32_t	
GPIER	uint32_t	
GPIMR	uint32_t	
GPODR	uint32_t	
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object
inputData	uint16_t	Input data register.
irqCtrl	temu_IfaceRef/ <unknown>	
outSignals	[temu_IfaceRef; 16]/ <unknown>	
outputData	uint16_t	Output data register.

Interfaces

Name	Type	Description
DeviceIface	DeviceIface	
MemAccessIface	MemAccessIface	
ResetIface	ResetIface	
SignalIface	SignalIface	Incomming signals

Commands

Name	Description
delete	Dispose instance of GPIO

10.8.5. Limitations

- Open-drain capability is not supported, GPODR is a dummy register.

10.9. GUTS

This section describes the P2020 Global Utilities (GUTS) model.

10.9.1. Loading the Plugin

```
import P2020
```

10.9.2. Configuration

10.9.3. @GUTS Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object

Commands

Name	Description
delete	Dispose instance of @GUTS
new	Create new instance of GUTS

Command new Arguments

Name	Type	Required	Description
name	string	yes	Name of object to create

10.9.4. GUTS Reference

Properties

Name	Type	Description
AUTOSRTSR	uint32_t	Automatic reset status reg
CLKOCR	uint32_t	Clock out control reg
Class	*void	Class object
Component	*void	Pointer to component object if part of component
DDRCLKDR	uint32_t	DDR clock disable reg
DEVDISR	uint32_t	Device disable control reg
ECMCR	uint32_t	ECM control reg
ECTRSTCR	uint32_t	Exception reset control reg
IOVSELSR	uint32_t	IO voltage select status reg
LoggingFlags	uint64_t	Flags for logging info
MCPSUMR	uint32_t	Machine check summary reg
Name	*char	Object name
PMCDR	uint32_t	Power management disable reg
PMUXCR	uint32_t	Alternate function signal multiplex control
PORBMSR	uint32_t	POR boot mode status reg
PORDBGMSR	uint32_t	POR debug mode status reg
PORDEVSR	uint32_t	POR device status reg
PORDEVSR2	uint32_t	POR device status reg 2
PORGPPORCR	uint32_t	General-purpose POR configuration reg

Name	Type	Description
PORPLLSR	uint32_t	POR PLL ratio status reg
POWMGTCSR	uint32_t	Power management control and status reg
PVR	uint32_t	Processor version reg
RSTCR	uint32_t	Reset control reg
RSTRSCR	uint32_t	Reset request status and control reg
SDHCDCR	uint32_t	SDHC debug control reg
SRDSCR	[uint32_t; 7]	SRDS control reg
SVR	uint32_t	System version reg
TimeSource	*void	Time source object

Interfaces

Name	Type	Description
MemAccessIface	MemAccessIface	Memory access interface

Commands

Name	Description
delete	Dispose instance of GUTS

10.9.5. Limitations

10.10. PCIe

This section describes the P2020 PCIe controller model.

The PCI Express controller can function as either a root complex (RC) or an endpoint (EP) on the PCI Express link.

The PCI Express interface supports the following register types:

- Memory-mapped registers-these registers control PCI Express address translation, PCI error management, PCI Express configuration register access.
- PCI Express configuration registers contained within the PCI Express configuration space-these registers are specified by the PCI Express specification for every PCIExpress device.

PCIe Bus contains information about connected to it EP-devices and PCIe bridge (RC device). There are two configuration header types applicable to PCI Express. Type 0 headers are typically used by endpoints, Type 1 headers are used by root complexes and switches/bridges.

10.10.1. Loading the Plugin

```
import P2020
```

10.10.2. Configuration

Every RC or EP device should be connected to a bus PCIe devices array via PCIDevIface interface, buses should be connected via PCIeBusIface to PCIe model. All buses (exept the main upstream bus0) should have a pcie bridge as a parent device. Each PCIe device, including bridges, should point to the primary bus it is connected to, bridges should also point to the secondary bus immediately downstream of the bridge.

Listing 42. Setting PCIe buses and devices via API

```
// connect devices to buses
temu_connect(upstreamBus, "pcieDevices", pciebridge, "PCIDevIface");
temu_connect(downstreamBus, "pcieDevices", pciedev, "PCIDevIface");

// connect buses to the pcie model
temu_connect(pcie, "pcieBuses", upstreamBus, "PCIeBusIface");
temu_connect(pcie, "pcieBuses", downstreamBus, "PCIeBusIface");

// add upstream bridge to the bus model
temu_writeValue(downstreamBus, "parentBridge", temu_makePropObj(pciebridge), 0);

// set primary & secondary buses for devices
temu_writeValue(pciedev, "primaryBus", temu_makePropObj(downstreamBus), 0);
temu_writeValue(pciebridge, "primaryBus", temu_makePropObj(upstreamBus), 0);
temu_writeValue(pciebridge, "secondaryBus", temu_makePropObj(downstreamBus), 0);
```

Listing 43. Setting PCIe buses and devices via Command Line

```
# connect devices to buses
connect a=upstreamBus.pcieDevices b=pciebridge:PCIDevIface
connect a=downstreamBus.pcieDevices b=pciedev:PCIDevIface

# connect buses to the pcie model
connect a=pcie.pcieBuses b=upstreamBus:PCIeBusIface
connect a=pcie.pcieBuses b=downstreamBus:PCIeBusIface

# add upstream bridge to the bus model
downbus.parentBridge = pciebridge

# set primary & secondary buses for devices
pciedev.primaryBus = downstreamBus
pciebridge.primaryBus = upstreamBus
```

```
pciebridge.secondaryBus = downstreamBus
```

The PCIe programmable registers that occupy memory-mapped space are named according to P2020 QorIQ integrated processor reference manual. All the PCIe registers are 4-byte wide.

Listing 44. Setting registers via API

```
// Write Status & Command Register register from Configuration Space (device 0, bus 0,
offset = 4) via CONFIG_ADDR/CONFIG_DATA registers
temu_writeValueU32(pcie, "PEX_CONFIG_ADDR", 0x80000004, 0);
temu_writeValueU32(pcie, "PEX_CONFIG_DATA", 0x02000000, 0);
// Value = 0x00000002 will be written to pcieConfigStatusCommand of the device0
```

Listing 45. Setting registers via Command Line

```
# Write Status & Command Register register from Configuration Space (device 0, bus 0,
offset = 4) via CONFIG_ADDR/CONFIG_DATA registers
pcie.PEX_CONFIG_ADDR = 0x80000004
pcie.PEX_CONFIG_DATA = 0x02000000
# Value = 0x00000002 will be written to pcieConfigStatusCommand of the device0
```

10.10.3. @PCIe Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object

Commands

Name	Description
delete	Dispose instance of @PCIe
new	Create new instance of PCIe

Command new Arguments

Name	Type	Required	Description
name	string	yes	Name of object to create

10.10.4. PCIe Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
PEXITAR	[uint32_t; 3]	Inbound Translation Address Registers
PEXIWAR	[uint32_t; 3]	Inbound Window Attributes Registers
PEXIWBAR	[uint32_t; 3]	Inbound Window Base Address Registers
PEXIWBEAR	[uint32_t; 3]	Inbound Window Base Extended Address Registers
PEXOTAR	[uint32_t; 5]	Outbound Translation Address Registers
PEXOTEAR	[uint32_t; 5]	Outbound Translation Extended Address Registers
PEXOWAR	[uint32_t; 5]	Outbound Window Attributes Registers
PEXOWBAR	[uint32_t; 5]	Outbound Window Base Address Registers
PEX_CONFIG	uint32_t	Configuration Data Register
PEX_CONFIG_ADDR	uint32_t	Configuration Address Register
PEX_CONFIG_DATA	uint32_t	Configuration Data Register
PEX_CONF_RTY_TOR	uint32_t	Configuration Retry Timeout Register
PEX_ERR_CAP	[uint32_t; 4]	Error Capture Register
PEX_ERR_CAP_STAT	uint32_t	Error Capture Status Register
PEX_ERR_DISR	uint32_t	Error Disable Register

Name	Type	Description
PEX_ERR_DR	uint32_t	Error Detect Register
PEX_ERR_EN	uint32_t	Error Interrupt Enable Register
PEX_IP_BLK_REV	[uint32_t; 2]	IP Block Revision Register
PEX_OTB_CPL_TOR	uint32_t	Outbound Completion Timeout Register
PEX_PMCR	uint32_t	Power Management Command Register
PEX_PME_MES_DISR	uint32_t	PME and Message Disable Register
PEX_PME_MES_DR	uint32_t	PME and Message Detect Register
PEX_PME_MES_IER	uint32_t	PME and Message Interrupt Enable Register
TimeSource	*void	Time source object
deviceIOAccess	temu_IfaceRef/ <unknown>	Access to the pcie’s io space from inbound windows
deviceMemAccess	temu_IfaceRef/ <unknown>	Access to the pcie’s memory space from inbound windows
ioMem	temu_IfaceRef/ <unknown>	PCIe i/o space object
irqCtrl	temu_IfaceRef/ <unknown>	Interrupt controller
isEPMode	uint8_t	
pciBridge	*void	PCIe rc bridge
pciMem	temu_IfaceRef/ <unknown>	PCIe memory space object
pcieBuses	temu_IfaceRefArray	PCI busses
processorMem	temu_IfaceRef/ <unknown>	Access to the processor’s memory space from inbound windows

Interfaces

Name	Type	Description
IrqIface	IrqCtrlIface	PCI IRQ interface
MemAccessIface	MemAccessIface	Memory access interface (registers)
PCIExpressBridgeIface	temu::PCIExpressBridgeIface	PCIe bridge interface.

Name	Type	Description
WindowsAccessIface	MemAccessIface	PCIe windows and configuration access interface.

Commands

Name	Description
delete	Dispose instance of PCIe

10.10.5. Limitations

- Target interface for inbound windows can be set to local memory space only, Serial Rapid IO is not implemented.

10.11. PIC

This section describes the P2020 Programmable Interrupt Controller.

PIC implements two types of programmable interrupt outputs: critical interrupts (cint) and non-critical interrupts (int). Support for the following interrupt types:

- External — Off-chip signals (12 interrupts);
- Internal - On-chip sources from peripheral logic (64 interrupts);
- Global timers A (4 interrupts) and B (4 interrupts) internal to the PIC;
- Interprocessor interrupts (4 interrupts);
- Message registers, used for interprocessor communication (8 interrupts);
- Shared message signaled registers, used for cross-program communication (8 interrupts).

PIC has support for two processors, all interrups can be routed to processor core 0 or 1. Multi-cast delivery mode for interprocessor and global timer interrupts allowing these interrupts to be routed to either core 0 or 1, or both cores.

Each interrupt source routed to int is assigned a priority value (range from 0x00 to 0x0f. Therefore, setting int priority to zero inhibits that interrupt. Likewise, setting CTPR[TASKP] (task priority threshold) to 0x0f prevents the PIC from delivering interrupts to that core through the int signal. Note that this is the reset TASKP value, preventing the PIC from asserting int before the PIC is configured.

10.11.1. Loading the Plugin

```
import P2020
```

10.11.2. Configuration

config.nCpu

Number of processors supported (1 or 2).

config.CCBFrequency

CCB frequency in MHz (dafault 333 MHz), used for timer frequency reporting calculation.

config.RTCFrequency

RTC frequency in MHz (dafault 64 MHz), used for timer frequency reporting calculation.

config.logInterrupts

Additional logs with raised interrupts information.

The PIC programmable registers that occupy memory-mapped space are named according to P2020 QorIQ integrated processor reference manual. You can set resgisters value as follows:

Listing 46. Setting registers via Command Line

```
# Set mixed mode On. Interrupts are handled by the normal priority and delivery
mechanisms of the PIC.
pic.GCR = 0x20000000

# Set min task priority = 1 (max = 0x0f) for processor core 0, interrupt priority must
exceed this value for the interrupt request to be serviced
pic.CTPR[0]=0x00000001

# Allow interrupts from DUART (default internal interrupt number for DUART is 26), set
its priority to max value = 0x0f
pic.IIVPR[26]=0x000F0000

# Mark DUART interrupts as non-critical(int) and set processor core 1 as a recipient
pic.IIDR[26]=0x00000001

# Mark DUART interrupts as critical(cint) and set processor core 1 as a recipient
pic.IIDR[26]=0x40000000
```

Listing 47. Setting registers via API

```
// Set mixed mode On. Interrupts are handled by the normal priority and delivery
mechanisms of the PIC.
temu_writeValueU32(pic, "GCR", 0x20000000, 0);

// Mark DUART interrupts as critical(cint) and set processor core 1 as a recipient
temu_writeValueU32(pic, "IIDR", 0x40000000, 26);
```

10.11.3. @PIC Reference

Properties

Name	Type	Description
Class	*void	Class object
Component	*void	Pointer to component object if part of component
LoggingFlags	uint64_t	Flags for logging info
Name	*char	Object name
TimeSource	*void	Time source object

Commands

Name	Description
delete	Dispose instance of @PIC
new	Create new instance of PIC

Command new Arguments

Name	Type	Required	Description
name	string	yes	Name of object to create

10.11.4. PIC Reference

Properties

Name	Type	Description
BRR1	uint32_t	PIC Block revision register 1
BRR2	uint32_t	PIC Block revision register 2
CISR0	uint32_t	PIC Critical interrupt summary register 0
CISR1	uint32_t	PIC Critical interrupt summary register 1
CISR2	uint32_t	PIC Critical interrupt summary register 2
CTPR	[uint32_t; 2]	PIC CoreCurrentTaskPriorityReg register
Class	*void	Class object
Component	*void	Pointer to component object if part of component
EIDR	[uint32_t; 12]	

Name	Type	Description
EIVPR	[uint32_t; 12]	
EOI	[uint32_t; 2]	PIC End of interrupt register
ERQSR	uint32_t	PIC External interrupt summary register
GCR	uint32_t	PIC GlobalConfiguration register
GTBCR	[uint32_t; 8]	
GTCCR	[uint32_t; 8]	
GTDR	[uint32_t; 8]	
GTVPR	[uint32_t; 8]	
IACK	[uint32_t; 2]	PIC Interrupt acknowledge register
IIDR	[uint32_t; 64]	
IIVPR	[uint32_t; 64]	
IPIDR	[uint32_t; 4]	
IPIVPR	[uint32_t; 4]	
IRQSR0	uint32_t	PIC IRQ_OUT_B summary register 0
IRQSR1	uint32_t	PIC IRQ_OUT_B summary register 1
IRQSR2	uint32_t	PIC IRQ_OUT_B summary register 2
IRR	[uint32_t; 2]	Interrupt request register with IRQ number of raised interrupt for each core
LoggingFlags	uint64_t	Flags for logging info
MER	[uint32_t; 2]	
MIDR	[uint32_t; 8]	
MIVPR	[uint32_t; 8]	
MSGR	[uint32_t; 8]	
MSIDR	[uint32_t; 8]	
MSIIR	uint32_t	
MSIR	[uint32_t; 8]	
MSIVPR	[uint32_t; 8]	

Name	Type	Description
MSR	[uint32_t; 2]	
Name	*char	Object name
PIR	uint32_t	PIC Processor core initialization register
PMMR0	[uint32_t; 4]	Performance monitor n mask register 0
PMMR1	[uint32_t; 4]	Performance monitor n mask register 1
PMMR2	[uint32_t; 4]	Performance monitor n mask register 2
SVR	uint32_t	PIC Spurious vector register
TCR	[uint32_t; 2]	
TFRR	[uint32_t; 2]	
TimeSource	*void	Time source object
VIR	uint32_t	PIC Vendor identification register
bestIrqId0	uint32_t	First to act IRQ number (cpu = 0)
bestIrqId1	uint32_t	First to act IRQ number (cpu = 1)
config.CCBFrequency	uint32_t	CCB frequency in MHz
config.RTCFrequency	uint32_t	RTC frequency in MHz
config.logInterrupts	uint8_t	
config.nCpu	uint8_t	
config.traceReads	uint8_t	
config.traceWrites	uint8_t	
cpu	[temu_IfaceRef; 2]/ <unknown>	
irqCtrl	[temu_IfaceRef; 2]/ <unknown>	
irqVectors0	[uint16_t; 104]	IRQ vector (cpu = 0) that was raised, index is an interrupt priority value
irqVectors1	[uint16_t; 104]	IRQ vector (cpu = 1) that was raised, index is an interrupt priority value

Name	Type	Description
isIrqWithPriorityRaised	[uint16_t; 2]	Flags irqs with what priority was raised
lookupTable0	[uint64_t; 32]	Lookup value (cpu = 0) according priority: odd index begin of the uint128 value, even index - the second part of uint128
lookupTable1	[uint64_t; 32]	Lookup value (cpu = 1) according priority: odd index begin of the uint128 value, even index - the second part of uint128

Interfaces

Name	Type	Description
DeviceIface	DeviceIface	
ExternalIrqIface	IrqCtrlIface	
InternalIrqIface	IrqCtrlIface	
IrqClientIface	IrqClientIface	uptree interrupt handlers (e.g. CPUs)
MemAccessIface	MemAccessIface	
ResetIface	ResetIface	

Ports

Prop	Iface	Description
irqCtrl	IrqClientIface	irq port

Commands

Name	Description
delete	Dispose instance of PIC

10.11.5. Limitations

- External IRQ_OUT signal is not supported