T-EMU: MIL-STD-1553 Bus Model Manual

Prepared	Checked
Mattias Holm Technical Manager	Dan Søren Nielsen QA Manager
Approved	
Michela Alberti General Manager	

Record of Changes

Author	Description	Rev	Date
Mattias Holm	Fix type in device interface.	1.2	2017-05-04
Mattias Holm	Auto gen tables	1.1	2016-05-12
Mattias Holm	Initial Version	1.0	2016-04-05

Table of Contents

1. Introduction

This document describes the T-EMU 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 T-EMU support for the 1553 protocol consist of a bus interface (Mil1553BusIface), a bus model (MilStd1553Bus) and a bus client interface (Mil1553DevIface).

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

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

2. Bus Model

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

3. Attributes

3.1. Properties

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

3.2. Interfaces

Name	Type	Description
Mil1553BusIface	Mil1553BusIface	

3.3. Ports

Prop	Iface	Description
-	-	-

3.4. Notifications

The default T-EMU milbus model issues the following notifications:

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

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

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

3.5. Configuration

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

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

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

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

3.6. 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.

4. Interfaces

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

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

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

5. Writing Clients

5.1. 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.

Warning

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

Messages should be sent in whole when they are supposed to arrive. This means that the bus controller model can immediately raise any needed interrupts when a message is complete.

Note

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

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

5.2. Bus Monitors

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