TERMA

TEMU
CAN Bus Modelling

Mattias Holm

Version 1.0, 2016-04-01

Table of Contents

Table 1. Record of Changes

Rev	Date	Author	Note
1.0	2016-04-01	MH	Initial version.

1. Introduction

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

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

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

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

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

2. Interfaces

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

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

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

struct temu_CanBusIface {
  void (*connect)(void *Bus, temu_CanDevIfaceRef Dev);
  void (*send)(void *Bus, void *Sender, temu_CanFrame *Frame);
  void (*enableSendEvents)(void *Bus);
  void (*disableSendEvents)(void *Bus);
  void (*reportStats)(void *Bus);
  void (*setFilter)(void *Bus, temu_CanDevIfaceRef Dev, int FilterID,
                    uint32_t Mask, uint32_t Code);
};
```

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

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

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

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

3. Commands

Two CAN bus related commands are provided:

Name	Description
can-connect	Connect a CAN device to a CAN bus.

Name	Description
can-disconnect	Disconnect CAN device from a CAN bus.

4. Classes

4.1. SimpleCANBus

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

4.1.1. Properties

Name	Type	Description
devices	irefarray	CAN devices attached to bus
object.timeSource	object	Time source object (a cpu or machine object)
stats.lastReportSentBits	uint64_t	Statistics
stats.sentBits	uint64_t	Statistics

4.1.2. Interfaces

Name	Type	Description
CanBusIface	CanBusIface	CAN Bus Interface

4.1.3. Ports

Prop	Iface	Description
-	-	-

5. Examples

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

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

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

The next example shows how to implement a simple CAN device

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

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


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

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

// Implement the CAN Device interface

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

void
```

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

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

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

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

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

}
```