T-EMU: SpaceWire Bus Modelling

Prepared

Alberto Ferrazzi
Software Engineer

Checked

Dan Søren Nielsen
QA Manager

Approved

Michela Alberti
General Manager

Record of Changes

Author	Description	Rev	Date
Mattias Holm	Fix typos in example code	1.1	2017-01-10
Alberto Ferrazzi	Initial Version	1.0	2016-11-22

Table of Contents

1. Introduction

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

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

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

2. Interfaces

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

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

typedef struct temu_SpwPacket {
  temu_SpwPacketType MsgType;
  temu_Buff PktData;
```

```
  uint8_t Flags;
} temu_SpwPacket;

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

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

While the SpaceWire protocol is character based, to have better performances T-EMU 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 T-EMU 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.

Name	Description
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. 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.

4. 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

5. Classes

5.1. SpwRouter

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

6. Attributes

6.1. Properties

Name	Type	Description
internal.linkState	[32 x int32_t]	Holds the link state of the ports
object.timeSource	object	Time source object (a cpu or machine object)
ports	[32 x iref]	Connected SpaceWire devices

Name	Type	Description
routingTable	[256 x uint8_t]	Configure packet-address to forwarding-port mapping

6.2. Interfaces

Name	Type	Description
SpwPortIface	SpwPortIface	Input spacewire ports interfaces

6.3. Ports

Prop	Iface	Description
-	-	-

7. Examples

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

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

The next example shows how to implement a simple SpaceWire device

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

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

typedef struct {
  temu_Object Super;

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

void*
create(const char *Name,
       int Argc TEMU_UNUSED,
       const temu_CreateArg *Argv TEMU_UNUSED)
{
```

```
  void *Obj = malloc(sizeof(SpwDevice));
  memset(Obj, 0, sizeof(SpwDevice));

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

  return Obj;
}

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

static void
spwDeviceChangeLinkState(SpwDevice *Device, temu_SpwLinkState LinkState)
{
  Device->LinkState = LinkState;
  if ((Device->Uplink.Iface != NULL) && (Device->Uplink.Obj != NULL)) {
      Device->Uplink.Iface->signalLinkStateChange(
        Device->Uplink.Obj, LinkState);
  }
}

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

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

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

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

static temu_SpwLinkState
spwPortIfaceGetOtherSideLinkState0(void *Obj)
```

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

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

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

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

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

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

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

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

  // Reference to the port interface of the other end.
```

```
    temu_addProperty(Cls, "Uplink",
                     offsetof(SpwDevice, Uplink),
                     teTY_IfaceRef,
                     1, // Number of elements (1 = scalar)
                     NULL, NULL,
                     "Other end port interface");

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