SpaceWire-D: Deterministic Data Delivery over SpaceWire

Steve Parkes(1), David Gibson(1), and Albert Ferrer(2)

(1)University of Dundee, Space Technology Centre, Dundee, DD1 4HN, Scotland, UK.

Email: sparkes@computing.dundee.ac.uk; davidgibson@computing.dundee.ac.uk

(2)STAR-Dundee Ltd, STAR House, 166 Nethergate, Dundee, DD1 4EE, Scotland, UK.

Email: albert.ferrer@star-dundee.com;

ABSTRACT

1 INTRODUCTION

SpaceWire-D is a protocol that provides deterministic data delivery over an existing SpaceWire network [1]. It allows SpaceWire networks to be used for time-critical avionics control applications and for asynchronous payload data handling.

SpaceWire-D uses the SpaceWire Remote Memory Access Protocol (RMAP) [2] to provide the basic communication mechanism: transactions that can read or write to memory in a remote target node. These transactions are executed by an initiator, with the initiator sending the RMAP command, a target receiving, executing and replying to the command, and the initiator receiving the reply from the target, which contains any data read from the target or is an acknowledgement to a write command.

To provide determinism the network bandwidth is split into a series of time-slots. One or more initiators are allowed to send a group of transactions in a particular time-slot, provided that transactions from different initiators do not use the same network resources, i.e. common SpaceWire links on the paths from the initiators to the target devices being accessed. The group of transactions executed in a particular time-slot must complete before the end of the time-slot, or a fault will be signalled. This restriction avoids a group of transactions from disrupting the transactions in the next time-slot, if they were to overrun their time-slot.

2 PROTOCOL STACK

The SpaceWire-D protocol stack is illustrated in Figure 2-1. SpaceWire-D is an asymmetric set of protocols with one or more initiators sending or collecting information from one or more target devices. An initiator is typically a payload control processor. A target is typically an instrument, sensor or actuator.

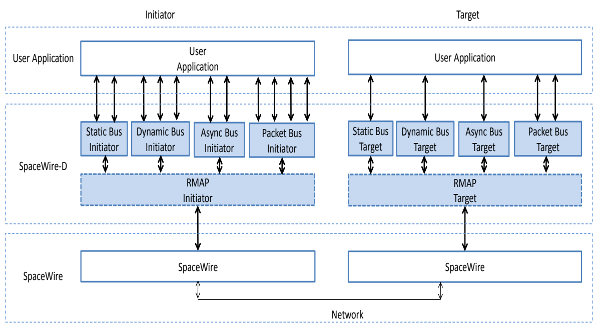

Figure 2-1 SpaceWire-D Protocol Stack

SpaceWire-D runs over existing SpaceWire networks with no modifications to existing SpaceWire routers. SpaceWire-D makes extensive use of the SpaceWire Remote Memory Access Protocol (RMAP). SpaceWire-D target devices are standard RMAP target devices.

SpaceWire-D provides four principal services: static bus, dynamic bus, asynchronous bus and packet bus.

- **Static bus:** Has a single slot associated to it. Accepts a single transaction group, one at a time, for execution in a specified slot. Provides a fully deterministic service.
- **Dynamic bus:** Has one or more slots associated to it. Accepts up to two transactions groups, the first will be sent in the next slot associated with the dynamic bus and the second one in the slot after that. Has double buffering so once the first transaction group has been executed and the second one is waiting to execute, another transaction group can be loaded. Is not fully deterministic since a transaction group can execute in one of possibly several slots associated with the dynamic bus.
- **Asynchronous bus:** Has one or more slots associated to it. Accepts individual transactions one at a time each with a priority setting. These transactions are stored in a queue with the highest priority transactions at the front. A transaction group to execute in the next slot for the

asynchronous bus is formed by taking transactions off the front of the queue, estimating their execution time, and forming a transaction group which will complete within the slot duration. This bus provides a prioritised means of asynchronously reading and writing to memory in the target devices associated with asynchronous bus, without upsetting the deterministic data delivery of static buses.

- **Packet bus:** Has one or more slots associated to it. Accepts requests to send a packet to a particular target or to receive a packet from a target. These requests are converted into a series of RMAP transactions that provide the packet bus service without upsetting the deterministic data delivery of the static bus.

3 HOW IT WORKS

In this section the way in which SpaceWire-D works is described.

3.1 Time-slots

SpaceWire-D uses time-division multiplexing of the network resources to provide deterministic data delivery. Time is split into a repeating sequence of time-slots. In a time-slot access to the network resources is given to one initiator so that there is no conflict on the network. Two or more initiators may access the network in a single time-slot, provided that they do not use any of the same SpaceWire links on the network.

The time-slots used in SpaceWire-D are illustrated in Figure 3-1.

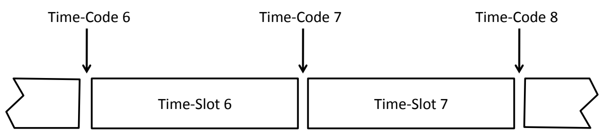

Figure 3-1 Time-Slots

A time-slot starts when a time-code is broadcast over the network and arrives at an initiator. It ends when the next time-code arrives. The time-slot is given the number of the time-code that starts the time-slot. There are therefore 64 time-slots, numbered 0 to 63. When time-slot 63 ends the sequence repeats again starting at time-slot 0. A time-slot is typically 1 ms to 20 ms in duration.

To provide resilience against a missing time-code and alternative means of defining the time-slots is provided which uses a local time-counter synchronised to the broadcast time-codes. This is illustrated in Figure 3-2.

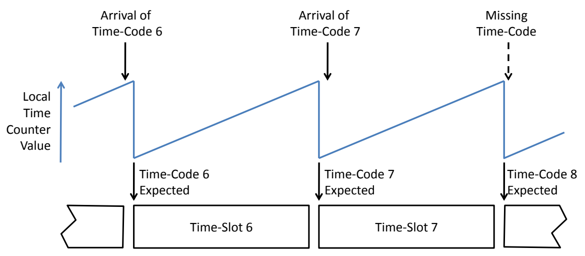

Figure 3-2 Time-Slots using a local time counter

The time-counter is used to determine the start and end of each time-slot in an initiator. The time-counter is synchronised to incoming time-codes. If a time-code goes missing, the local time-counter will start the next time-slot. The missing time-code is reported to a network management application. Several time-codes may be lost before time-slot synchronisation is lost completely.

3.2 Transaction

SpaceWire-D sends and receives information using SpaceWire RMAP transactions. An initiator executes an RMAP transaction by sending an RMAP command to a target and receiving the reply from that target. Existing RMAP target devices can be used with SpaceWire-D without any modification. The RMAP transaction provides a means of reading, writing or read-modify-writing to memory in a remote target.

3.3 Transaction Group

In a single time-slot it is possible to execute hundreds of RMAP transactions, depending on the amount of data being transferred in those RMAP transactions. The transactions that are to be executed in a given time-slot are called a transaction group. A transaction group is executed in a time-slot. If a defined transaction group is executed in a particular time-slot, this provides fully deterministic data delivery.

The transactions in a transaction group must all be completed before the end of the time-slot in which they are executed or they will start to affect traffic in other time-slots. When transaction groups are passed to SpaceWire-D for execution they are first checked to make sure that they are likely to complete execution in their designated time-slot.

3.4 Multi-slots

Suppose a SpaceWire network is operating at 200 Mbits/s and the SpaceWire-D time-slot duration is set to 10 ms. The maximum amount of data that can be sent is 200 kbytes. If a camera that provides 1 Mbyte images is to be used as a target device in a SpaceWire-D network a much larger time-slot will be required.

Rather than increase the time-slot period, which may have been set to provide responsive determinism for other data transfers, SpaceWire-D can concatenate a number of adjacent time-slots into a multi-slot. Time-slots and multi-slots are referred to generically as slots.

A transaction group can be executed in a slot, i.e. time-slot or multi-slot, provided that all the transactions in the group complete before the end of that slot.

3.5 Static Bus

The static bus is the simplest and most deterministic of the bus types that SpaceWire-D supports. When a static bus is opened it is allocated a single slot only. A static bus accepts a transaction group from an initiator application and will send that group in the next occurrence of the time-slot allocated to the static bus. A static bus is given the number of the lowest number time-slot in its allocated slot.

The operation of a static bus is illustrated in Figure 3-3.

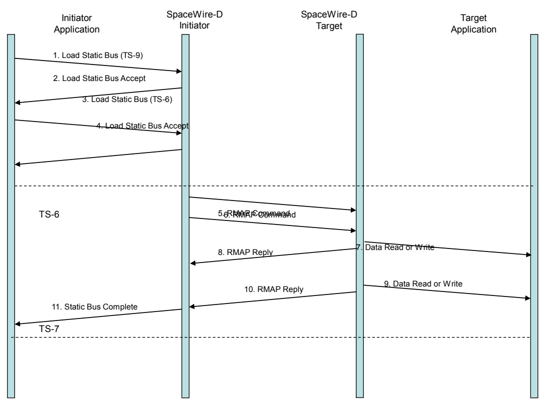

Figure 3-3: Transactions on a static bus

Once a static bus has been opened it can accept a transaction group for sending on that bus. It is called a bus because an initiator is master of the network for the duration of a time-slot that it has been allocated. It can access many target devices, but the initiator starts the transactions.

In Figure 3-3, two static buses have been opened; one for time-slot 6 and one for time-slot 9. The user application loads the transaction group to be executed on these two static buses using the Load Static Bus primitives. When time-slot 6 starts, the transaction group loaded for static bus 6 is executed; the RMAP commands in the transaction group are sent to the various targets and the replies received. The target application is informed every time an RMAP command is executed on the target. The initiator is informed when the complete transaction group has finished execution.

To support repeated access of sensor or housekeeping information a static bus can be set to repeat each time its allocated slot starts. This saves the user application having to reload the static bus with the same information repeatedly. If the repeating information does need to change, the user application can simply write the new transaction group into the static bus.

3.6 Dynamic Bus

The dynamic bus is similar to the static bus except that one or more slots can be allocated to a dynamic bus and two transaction groups are queued in the dynamic bus initiator; the current transaction group which will execute as soon as a time-slot allocated to the dynamic bus starts, and the next transaction group which becomes the current transaction group once the current transaction group has been executed. When a dynamic bus is opened it is associated with one or more slots.

The operation of the dynamic bus is illustrated in Figure 3-4.

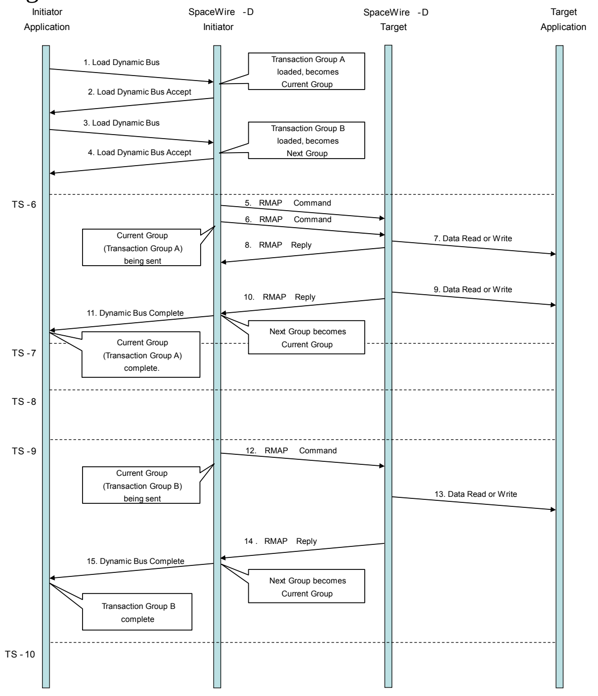
Initiator Application
SpaceWire -D Initiator
SpaceWire -D Target
Target Application
1. Load Dynamic Bus
Transaction Group A loaded, becomes Current Group
2. Load Dynamic Bus Accept
3. Load Dynamic Bus
Transaction Group B loaded, becomes Next Group
4. Load Dynamic Bus Accept
TS -6
5. RMAP Command
6. RMAP Command
Current Group (Transaction Group A) being sent
7. Data Read or Write
8. RMAP Reply
9. Data Read or Write
10. RMAP Reply
11. Dynamic Bus Complete
Next Group becomes Current Group
Current Group (Transaction Group A) complete.
TS -7
TS -8
TS -9
12. RMAP Command
Current Group (Transaction Group B) being sent
13. Data Read or Write
14. RMAP Reply
15. Dynamic Bus Complete
Next Group becomes Current Group
Transaction Group B complete
TS - 10

Figure 3-4: Transactions on a dynamic bus

Once a dynamic bus has been opened the user application can load it with a transaction group for execution. Once loaded this group (Group A in Figure 3-4) becomes the current group and is executed on the first slot to arrive of those allocated to the dynamic bus. If another transaction group is loaded (Group B), it becomes the next group and will become the current group, once the present current group (Group A) has been executed.

A transaction group that is loaded into a dynamic bus is not executed in a specific slot, but will execute in the next slot allocated to the dynamic bus. This makes it less deterministic than the static bus. Repeating transactions cannot be used with a dynamic bus.

3.7 Asynchronous Bus

The asynchronous bus is for executing individual transactions. Transactions for an asynchronous bus are loaded into the bus singly and queued ready for execution in the next slot allocated to the bus. A priority can be applied to these transactions with a higher priority transaction being placed before lower priority ones in the queue. When a slot allocated to an asynchronous bus arrives, transactions are pulled from the queue to form a transaction group which is able to be executed within the time limits of the slot. These transactions are then executed.

The operation of the asynchronous bus is illustrated in Figure 3-5.

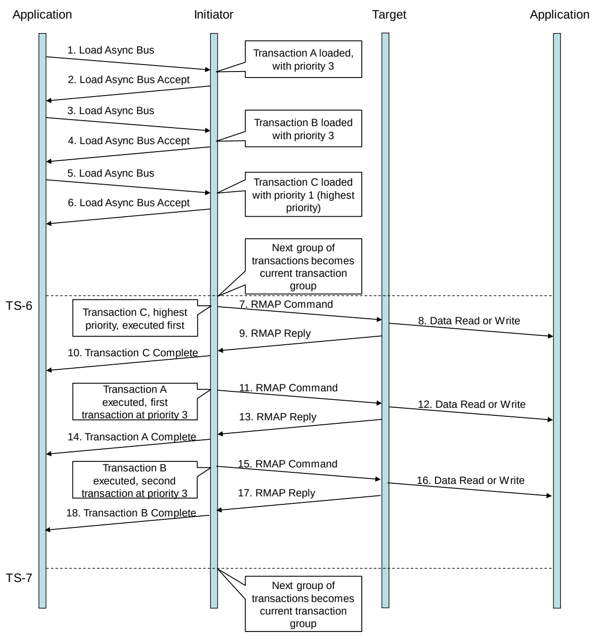
Application
Initiator
Target
Application
1. Load Async Bus
Transaction A loaded, with priority 3
2. Load Async Bus Accept
3. Load Async Bus
Transaction B loaded with priority 3
4. Load Async Bus Accept
5. Load Async Bus
Transaction C loaded with priority 1 (highest priority)
6. Load Async Bus Accept
Next group of transactions becomes current transaction group
TS-6
Transaction C, highest priority, executed first
7. RMAP Command
8. Data Read or Write
9. RMAP Reply
10. Transaction C Complete
Transaction A executed, first transaction at priority 3
11. RMAP Command
12. Data Read or Write
13. RMAP Reply
14. Transaction A Complete
Transaction B executed, second transaction at priority 3
15. RMAP Command
16. Data Read or Write
17. RMAP Reply
18. Transaction B Complete
TS-7
Next group of transactions becomes current transaction group

Figure 3-5: Transactions on an asynchronous bus

A series of individual transactions are loaded in to the asynchronous bus. These are placed on the transaction queue in priority order. When the next slot for the asynchronous bus arrives, the transactions are executed, starting with the highest priority transactions.

3.8 Packet Bus

The packet bus will be described in the full paper.

3.9 Schedule

The schedule is a means for coordinating the use of the SpaceWire network resources when multiple initiators are being used. There is a schedule table in each initiator which specifies which bus has access to the network. An example schedule is given in Figure 3-6.

Time-Slot	Bus
0	Static 0
1	Dynamic 1
2	Static 2
3	Async 3
4	Static 4
5	Async 5
6	Async 5
7	Dynamic 7
8	Empty
9	Dynamic 1
10	Dynamic 7
…	…
61	Static 61
62	Dynamic 7
63	Static 63

Figure 3-6: Example schedule table

Time-slot 0 is allocated to static bus 0. Similarly time-slots 2, 4 and 61 are allocated static buses each of one time-slot duration.

Time-slots 0 and 9 are allocated to dynamic bus 1. Time-slots 7, 10 and 62 are allocated to dynamic bus 7.

Time-slot 3 is allocated to asynchronous bus 3, which is a single slot of one time-slot duration. Time-slots 5 and 6 are allocated to asynchronous bus 5, which is a single slot of two time-slots duration.

When a particular time-slot arrives, the bus associated with that time-slot in the schedule table is allowed to execute some transactions.

When multiple initiators are being used the network resources (SpaceWire links) used in a particular time-slot by a bus in one initiator must not be used by any other initiator in that same time-slot. It is the responsibility of the system engineer to ensure that this is the case. Scheduling tools, to help with the definition of buses and the allocation of buses to the schedule in each initiator, are being researched by the University of Dundee,

4 FDIR

Each SpaceWire-D initiator monitors the transactions it has initiated and ensures that they all complete before the end of the slot in which they were executed. If they fail to complete in time a fault is reported. RMAP errors can occur without a fault being reported. For example, if an RMAP command is sent to a target and this command is not authorised by the target, the RMAP reply will contain the "command not authorised" error code. This will be received at the initiator and the transaction will be considered executed, even though the reply contained an error code. This has not broken the SpaceWire-D protocol so no fault is reported. It is up to the user application to determine what it wants to do in response to the error code received.

It is important that every transaction group completes in its allocated time-slot. If it doesn't, a fault will be reported. To prevent inadvertent faults, SpaceWire-D checks each transaction and transaction group it is given for execution on a particular bus to ensure that it is for a target associated with the bus, and that it will execute within the duration of the slot allocated to the bus. This error prevention mechanism means that the application will be told of any errors with the transactions and transaction groups. Erroneous transactions and transaction groups will not be executed, preventing faults occurring on the network. To check the target and anticipated execution time, the initiator uses information about the target provided when a bus was opened, together with transaction data length information from each transaction.

Time-codes are crucial for the operation of SpaceWire-D, so it is important to monitor them for errors, This can be done in the initiators by applying early and late watchdog timers for each time-code as illustrated in Figure 4-1. If the time-code arrives outside the time-code window or is missing altogether, a fault can be reported to the user application, which can take appropriate action.

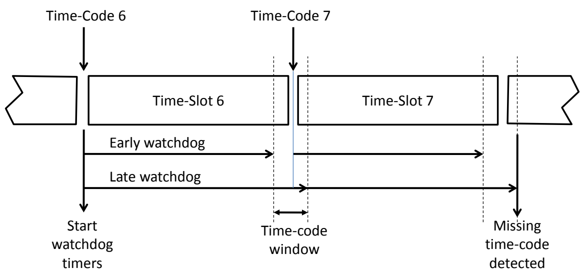

Figure 4-1 Time-Code Watchdog

5 CURRENT STATUS

The initial concept for SpaceWire-D was developed by University of Dundee [3] [4] [5], who are now writing the draft standard specification under ESA contract. An early version of SpaceWire-D developed in Japan is being used in missions like Hayabusa 2 [6][7].

The requirements for SpaceWire-D have been gathered with inputs from Airbus Defence GmbH and Space and Thales Alenia Space France. A draft SpaceWire-D standard specification has been written.

The final paper will provide updated information on the status of SpaceWire-D, including results of initial implementation.

6 CONCLUSIONS

SpaceWire-D is designed to support both avionics and payload data-handling applications on a single network. Avionics applications like attitude and orbit control require deterministic data delivery. Asynchronous payload data must not interfere with the deterministic delivery of time-critical data. SpaceWire-D uses time-division multiplexing to share the network resources between several initiators. It provides deterministic data delivery using existing SpaceWire networks and RMAP target devices. It supports common time-critical avionics and asynchronous payload data-handling on a single SpaceWire network.

7 ACKNOWLEDGMENTS

The research leading to these results has received funding the European Space Agency under ESA contract number 4000107346/12/NL/LvH/fe. We would also like to thank David Jameux the ESA project manager for the SpaceWire-D related activities for his help.

8 REFERENCES

[1] ECSS Standard ECSS-E-ST-50-12C, "SpaceWire, Links, Nodes, Routers and Networks", Issue 1, European Cooperation for Space Data Standardization, July 2008, available from http://www.ecss.nl.

[2] ECSS Standard ECSS-E-ST-50-52C, "SpaceWire – Remote memory access protocol", Issue 1, European Cooperation for Space Data Standardization, 5 February 2010, available from http://www.ecss.nl.

[3] S. Parkes and A. Ferrer Florit, "SpaceWire-D Deterministic Control and Data Delivery Over SpaceWire Networks", Draft B, April 2010, available from http://spacewire.esa.int/WG/SpaceWire/

[4] S. Parkes, A Ferrer, S. Mills, A. Mason, "SpaceWire-D: Deterministic Data Delivery with SpaceWire", International SpaceWire Conference, St Petersburg, Russia, June 2010.

[5] S. Parkes, A. Ferrer Florit, A. Gonzalez Villafranca and C. McClements, "SpaceWire-D Deterministic Control and Data Delivery Over SpaceWire Networks", Draft C, April 2012, available from http://spacewire.esa.int/WG/SpaceWire/

[6] Y. Chen, M. Takada, R. Kurachi, H. Takada,Satoko Kawakami, Yasuhiro Takeda, Hiroki Hihara, Ryu Funase, Tetsuya Masuda, Masatoshi Ebara and Takahiro

Yamada, “Real-time Data Recording System with SpaceWire for Asteroid Sample Return Mission HAYABUSA2”, International SpaceWire Conference, Gothenburg, 2013.

[7] Mitsutaka Takada, Hiroaki Takada, Yang Chen, Takayuki Yuasa and Tadayuki Takahashi, “Development of Software Platform Supporting a Protocol for Guaranteeing the Real-Time Property of SpaceWire”, International SpaceWire Conference, Gothenburg, 2013.