SpaceWire EGSE: Simulating an Instrument

The SpaceWire Electronic Ground Support Equipment (EGSE) is a test and development unit that simulates instruments or other SpaceWire equipment in real-time. The EGSE is configured using a simple yet powerful scripting language designed specifically for SpaceWire applications. Once configured the EGSE operates independent of software resulting in real-time performance. This can be used to rapidly mimic the behaviour of SpaceWire equipment, vastly reducing traditional development time, risk and cost associated with writing equivalent software in a real-time operating system.

This application note provides an example of how an instrument may be simulated using a SpaceWire EGSE. Comparing this to traditional EGSE which requires complex and expensive real-time software development, the time saving, risk reduction and cost benefits provided by the SpaceWire EGSE should become clear.

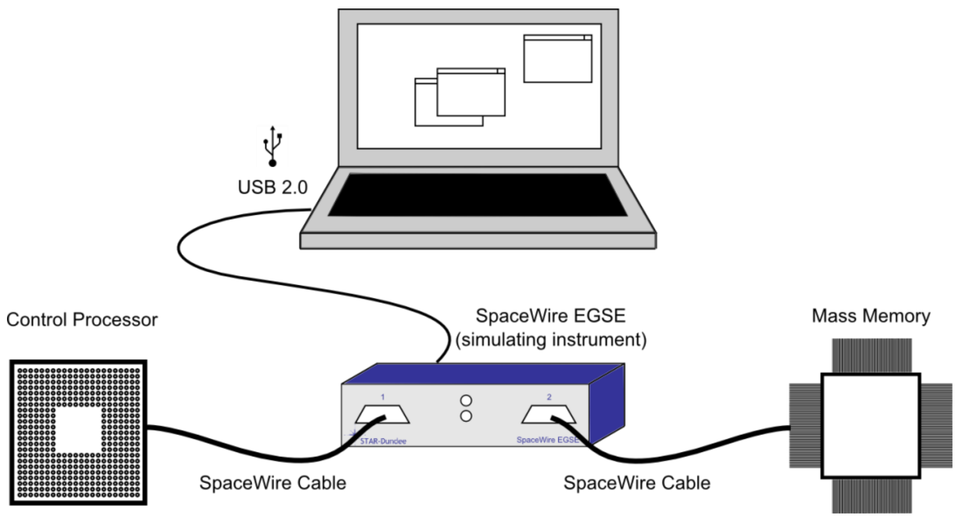
Scenario

A company needs to simulate a SpaceWire instrument in order to develop and test the SpaceWire components that will communicate with it: a control processor and mass memory module. The control processor is responsible for setting the instrument's mode. The mass memory module receives the SpaceWire traffic transmitted from the instrument.

The instrument operates at a link speed of 200Mbits/s and has three modes. In "Mode 0" the instrument is off. In "Mode 1" the instrument transmits 64kByte packets with random data at 150Mbits/s. In "Mode 2", 1kByte packets with walking ones data are transmitted at 50Mbits/s. The mode is determined by the two least significant bits of the instrument's control register at address 1 (Mode 0 = 0b00, Mode 1 = 0b01 and Mode 2 = 0b10). RMAP write commands transmitted from the control processor change the value of the control register and therefore govern the instrument's mode.

Test Setup

The SpaceWire EGSE is connected to the host PC via USB and powered by a 5V power brick. A SpaceWire cable connects interface one of the EGSE to the control processor. Interface two of the SpaceWire EGSE is connected to the mass memory module. The diagram below illustrates this configuration.



Instrument Simulation Test Setup

Scripting the Instrument Simulation

In order to configure the SpaceWire EGSE to simulate the instrument, a script must first be written that defines the instrument's behavior. In this example the link speed is first stipulated:

```
config
       spw_tx_rate(2, 200Mbps)
end config
```

The above statement sets the line rate of SpaceWire link two to 200Mbits/s.

Pattern matchers are then defined that match on the RMAP write commands that control the instrument's mode (note that comments are prefixed with the '#' character):

```
match matchMode0RMAPRegWrite
       sop
       0xfe          # Target logical address
       0x01          # Protocol ID                (0x01 for RMAP)
       0x60          # Instruction                (command, write single address, no reply)
       0x20          # Key
                     # No reply address
       0xfe          # Initiator logical address
       0x--          # Transaction ID      (MS)   (Ignore)
       0x--          # Transaction ID      (LS)   (Ignore)
       0x00          # Extended address           (0)
       0x00          # Address byte 3      (MS)
       0x00          # Address byte 2
       0x00          # Address byte 1
       0x01          # Address byte 0      (LS)   (1)
       0x00          # Data length byte 2  (MS)
       0x00          # Data length byte 1
       0x04          # Data length byte 0  (LS)   (4 bytes)
       0x--          # Header CRC                 (Ignore)
       0x--          # Data byte 3         (MS)   (Ignore)
       0x--          # Data byte 2                (Ignore)
       0x--          # Data byte 1                (Ignore)
       0b------00    # Data byte 0         (LS)   (Mode 0 = 0b00)
       0x--          # Data CRC                   (Ignore)
       eop
end match

match matchMode1RMAPRegWrite
       sop
       0xfe          # Target logical address
       0x01          # Protocol ID                (0x01 for RMAP)
       0x60          # Instruction                (command, write single address, no reply)
       0x20          # Key
                     # No reply address
       0xfe          # Initiator logical address
       0x--          # Transaction ID      (MS)   (Ignore)
       0x--          # Transaction ID      (LS)   (Ignore)
       0x00          # Extended address           (0)
       0x00          # Address byte 3      (MS)
       0x00          # Address byte 2
       0x00          # Address byte 1
       0x01          # Address byte 0      (LS)   (1)
       0x00          # Data length byte 2  (MS)
       0x00          # Data length byte 1
       0x04          # Data length byte 0  (LS)   (4 bytes)
       0x--          # Header CRC                 (Ignore)
       0x--          # Data byte 3         (MS)   (Ignore)
       0x--          # Data byte 2                (Ignore)
       0x--          # Data byte 1                (Ignore)
       0b------01    # Data byte 0         (LS)   (Mode 1 = 0b01)
       0x--          # Data CRC                   (Ignore)
       eop
end match

match matchMode2RMAPRegWrite
       sop
       0xfe          # Target logical address
       0x01          # Protocol ID              (0x01 for RMAP)
       0x60          # Instruction              (command, write single address, no reply)
       0x20          # Key
                     # No reply address
       0xfe          # Initiator logical address
       0x--          # Transaction ID      (MS)  (Ignore)
       0x--          # Transaction ID      (LS)  (Ignore)
       0x00          # Extended address          (0)
       0x00          # Address byte 3      (MS)
       0x00          # Address byte 2
       0x00          # Address byte 1
       0x01          # Address byte 0      (LS)  (1)
       0x00          # Data length byte 2 (MS)
       0x00          # Data length byte 1
       0x04          # Data length byte 0 (LS)  (4 bytes)
       0x--          # Header CRC                (Ignore)
       0x--          # Data byte 3          (MS) (Ignore)
       0x--          # Data byte 2               (Ignore)
       0x--          # Data byte 1               (Ignore)
       0b------10    # Data byte 0          (LS) (Mode 2 = 0b10)
       0x--          # Data CRC                  (Ignore)
       eop
end match
```

Three pattern matchers are defined above, each of which will match on an RMAP write command to the control register (address 1). The first pattern matcher is called “matchMode0RMAPRegWrite” and matches when the two least significant bits of the RMAP write data field equal 0b00. The second pattern matcher is called “matchMode1RMAPRegWrite” and matches when the two least significant bits of the RMAP write data field equal 0b01. The final pattern matcher is called “matchMode2RMAPRegWrite” and matches when the two least significant bits of the RMAP write data field equal 0b10.

The pattern matchers are then associated with events:

```
events
       mode0RMAPPkt = match_rx(1, matchMode0RMAPRegWrite)
       mode1RMAPPkt = match_rx(1, matchMode1RMAPRegWrite)
       mode2RMAPPkt = match_rx(1, matchMode2RMAPRegWrite)
end events
```

Three received pattern matched events are defined. The first is named “mode0RMAPPkt” and is raised when the traffic received on interface one matches that specified in the pattern matcher “matchMode0RMAPRegWrite”. The second is named “mode1RMAPPkt” and is raised when the traffic received on interface one matches that specified in the pattern matcher “matchMode1RMAPRegWrite”. The final received pattern matched event is named “mode2RMAPPkt” and is raised when the traffic received on interface one matches that specified in the pattern matcher “matchMode2RMAPRegWrite”.

Variables used to transmit packets with dynamic data (random and walking ones in this case) are then defined:

```
variables
           random = rnd08()
       rotateLeft = rol08(1)
end variables
```

A one byte random variable is declared named “random” along with a one byte bitwise rotate left variable named “rotateLeft” with an initial value of one.

The packets transmitted in each mode are then defined:

```
packet mode1Pkt
       random * 32768
       random * 32768
       eop
end packet

packet mode2Pkt
       rotateLeft * 1024
       eop
end packet
```

A packet named “mode1Pkt” is defined that consists of 65536 references of the “random” variable followed by an EOP marker. A packet named “mode2Pkt” is defined that consists of 1024 references of the “rotateLeft” variable followed by an EOP marker. Note that the “random” variable references are split over two lines because the maximum number of times a variable can be referenced on a single line is 65535.

A packet transmission schedule for each mode is then declared:

```
schedule mode0Schedule
end schedule

schedule mode1Schedule @ 150Mbps
       send mode1Pkt
end schedule

schedule mode2Schedule @ 50Mbps
       send mode2Pkt
end schedule
```

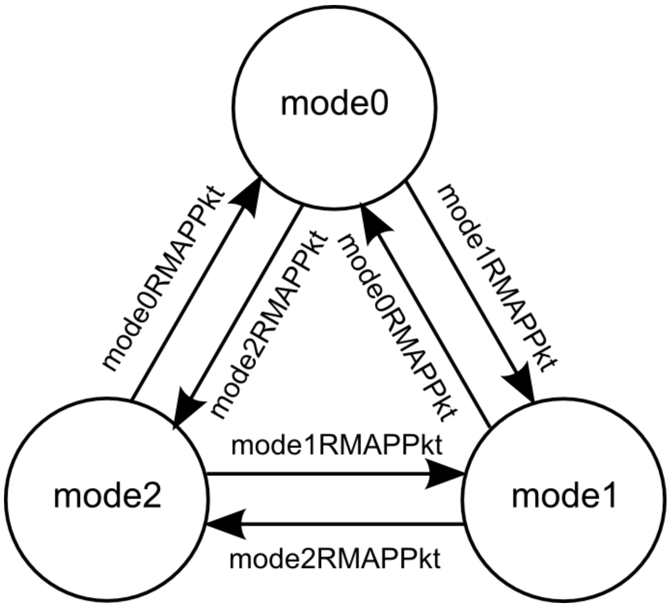
The first schedule is named “mode0Schedule” and transmits nothing. The second schedule is named “mode1Schedule” and specifies that the packet named “mode1Pkt” should be transmitted as soon as the schedule is executed at a data rate of 150Mbits/s. The third schedule is named “mode2Schedule” and specifies that the packet named “mode2Pkt” should be transmitted as soon as the schedule is executed at a data rate of 50Mbits/s.

Finally a state machine is defined:

```
statemachine 2
       initial state mode0
              do mode0Schedule
              LED colour is white
              on mode1RMAPPkt goto mode1 immediately
              on mode2RMAPPkt goto mode2 immediately
       end state
       state mode1
              do mode1Schedule repeatedly
              LED colour is green
              on mode0RMAPPkt goto mode0 immediately
              on mode2RMAPPkt goto mode2 immediately
       end state
       state mode2
```

```
                do mode2Schedule repeatedly
                LED colour is blue
                on mode0RMAPPkt goto mode0 immediately
                on mode1RMAPPkt goto mode1 immediately
        end state
end statemachine
```

A state machine is defined that is associated with SpaceWire interface two. It contains three states named “mode0”, “mode1” and “mode2”. The starting state named “mode0” executes the schedule “mode0Schedule” and transitions to “mode1” immediately if the “mode1RMAPPkt” event is detected and “mode2” immediately if the “mode2RMAPPkt” event is detected. The state named “mode1” executes the schedule “mode1Schedule” repeatedly and transitions to the “mode0” state immediately if the “mode0RMAPPkt” event is detected and the “mode2” state immediately if the “mode2RMAPPkt” event is detected. The state named “mode2” executes the schedule “mode2Schedule” repeatedly and transitions to the “mode0” state immediately if the “mode0RMAPPkt” event is detected and the “mode1” state immediately if the “mode1RMAPPkt” event is detected.



SpaceWire EGSE Instrument Simulation State Diagram

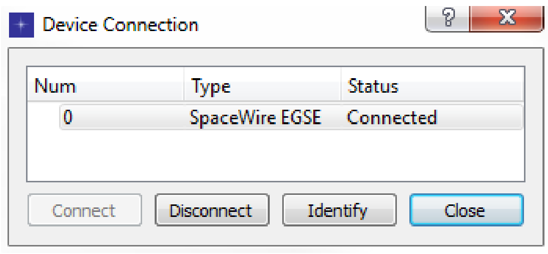
When the SpaceWire EGSE is configured using this script, it can operate in three states that correspond to the three modes described in the scenario above. In the initial “mode0” state it does not transmit any data. In the “mode1” state it transmits 64kByte packets with random data at a data rate of 150Mbits/s from SpaceWire interface two. In the “mode2” state it transmits 1kByte packets with walking ones data at a data rate of 50Mbits/s from SpaceWire interface two.

State transitions occur in response to RMAP write command packets received on SpaceWire interface one. Whilst in the “mode0” or “mode2” state, if an RMAP write command is received that writes 0b01 to the two least significant bits of the control register (address 1), a transition to the “mode1” state will occur. Whilst in the “mode0” or “mode1” state, if an RMAP write command is received that writes 0b10 to the two least significant bits of the control register, a transition to the “mode2” state will occur. Finally, whilst in the “mode1” or “mode2” state, if an RMAP write command is received that writes 0b00 to the two least significant bits of the control register, a transition to the “mode0” state will occur.

The optional “LED colour is white”, “LED colour is green” and “LED colour is blue” statements in the state machine provide a simple indicator of the current executing state. Whilst in the “mode0” state, the central LED above SpaceWire interface two is white, in the “mode1” state it is green and in the “mode2” state it is blue.

Compiling the Script

A script must be compiled before the SpaceWire EGSE can be configured. The SpaceWire EGSE comes with both a command line application and a GUI application that can be used to do this. In this example the GUI application will be used. Once the SpaceWire EGSE is connected and powered on, the “egse_gui” application is launched. A “Device Connection” window is presented where a connection to the SpaceWire EGSE is opened.

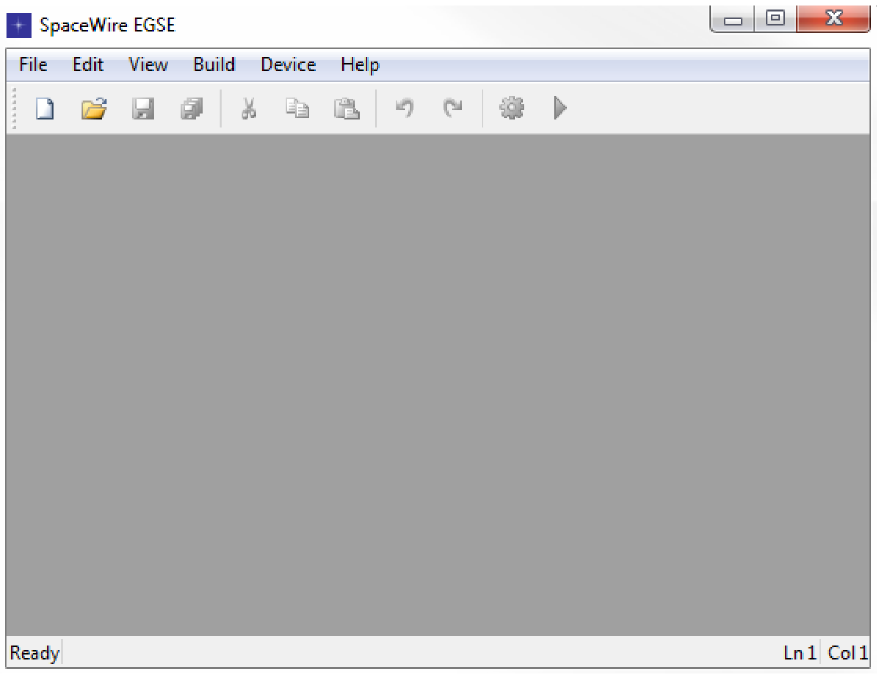


Device Connection

Num	Type	Status
0	SpaceWire EGSE	Connected

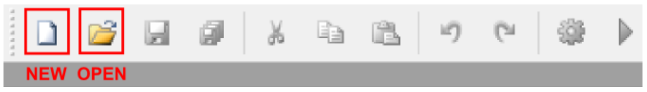
Device Connection Window

When the “Device Connection” window is closed the main window is displayed.



Main Window

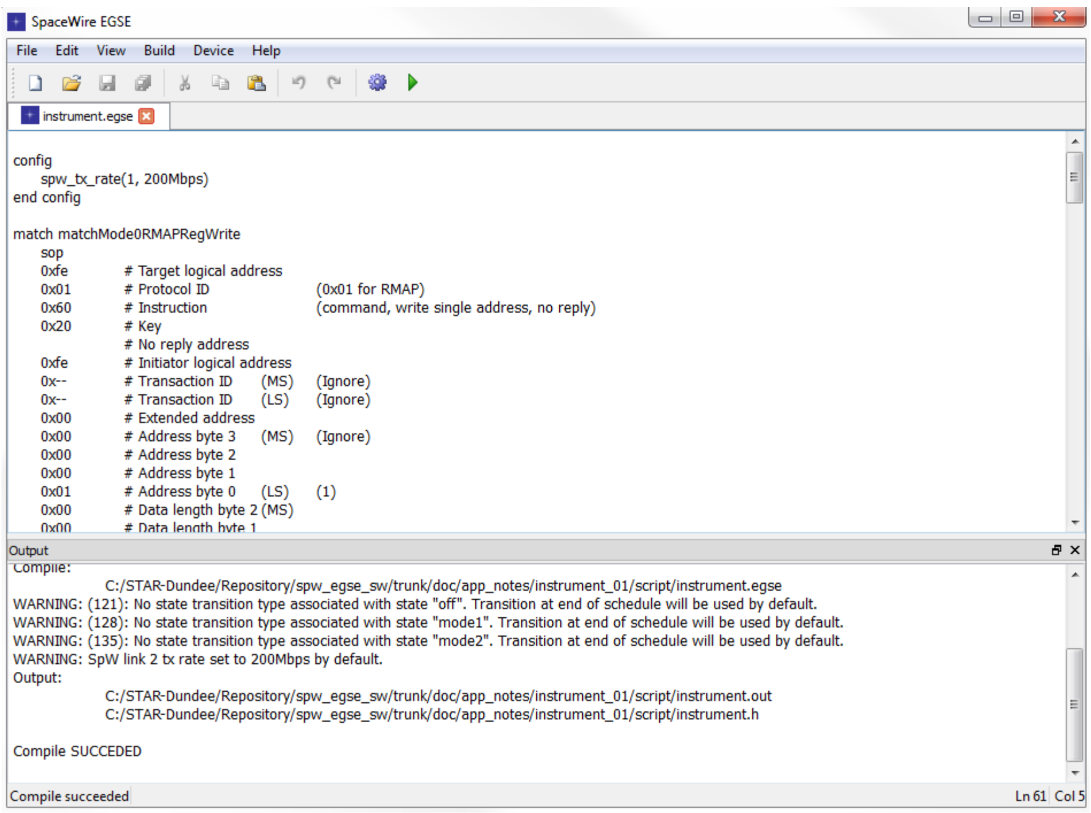
To create the new instrument script, the “New” toolbar button is selected. Alternatively if the script was already created using a different text editor it can be opened using the “Open” toolbar button.



New and Open Toolbar Buttons

Once the instrument simulation has been scripted, it is compiled using the “Compile” toolbar button. If the script has been newly created, a save window will prompt the user to save it. When the compile completes, an output window is displayed that shows any compiler errors or warnings along with the final compile status i.e. compile succeeded or failed.

STAR-Dundee



```
config
    spw_tx_rate(1, 200Mbps)
end config

match matchMode0RMAPRegWrite
    sop
    0xfe        # Target logical address
    0x01        # Protocol ID                 (0x01 for RMAP)
    0x60        # Instruction                 (command, write single address, no reply)
    0x20        # Key
                # No reply address
    0xfe        # Initiator logical address
    0x--        # Transaction ID     (MS)     (Ignore)
    0x--        # Transaction ID     (LS)     (Ignore)
    0x00        # Extended address
    0x00        # Address byte 3     (MS)     (Ignore)
    0x00        # Address byte 2
    0x00        # Address byte 1
    0x01        # Address byte 0     (LS)     (1)
    0x00        # Data length byte 2 (MS)
    0x00        # Data length byte 1
```

Output

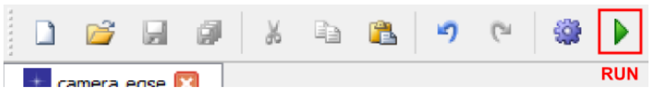
```
Compile:
            C:/STAR-Dundee/Repository/spw_egse_sw/trunk/doc/app_notes/instrument_01/script/instrument.egse
WARNING: (121): No state transition type associated with state "off". Transition at end of schedule will be used by default.
WARNING: (128): No state transition type associated with state "mode1". Transition at end of schedule will be used by default.
WARNING: (135): No state transition type associated with state "mode2". Transition at end of schedule will be used by default.
WARNING: SpW link 2 tx rate set to 200Mbps by default.
Output:
            C:/STAR-Dundee/Repository/spw_egse_sw/trunk/doc/app_notes/instrument_01/script/instrument.out
            C:/STAR-Dundee/Repository/spw_egse_sw/trunk/doc/app_notes/instrument_01/script/instrument.h

Compile SUCCEDED
```

Compile succeeded   Ln 61 Col 5

Compiler Output

Configure the SpaceWire EGSE

Once a script has been compiled successfully the SpaceWire EGSE can be configured. With a connection to the EGSE having previously been opened and the instrument script open, the “Run” toolbar button is selected.

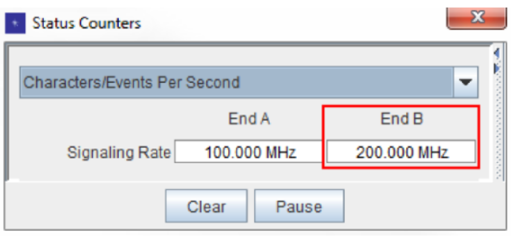


Run Toolbar Button

This configures the SpaceWire EGSE in such a way that it behaves as specified in the instrument script. Once configured it operates independent of software resulting in real-time behavior.

Resulting SpaceWire Traffic

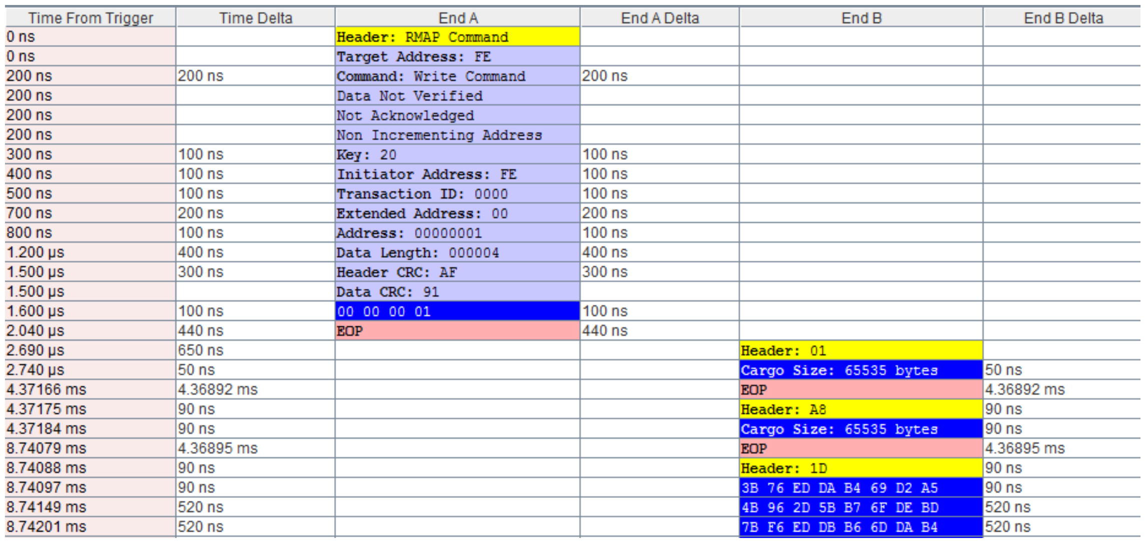
As soon as the SpaceWire EGSE is configured it operates as defined in the instrument script: the link speed of SpaceWire interface two is set to 200Mbits/s and the EGSE starts in “mode0”. The following screenshots were taken using a SpaceWire Link Analyser Mk2.



	End A	End B
Signaling Rate	100.000 MHz	200.000 MHz

200Mbits/s Link Speed

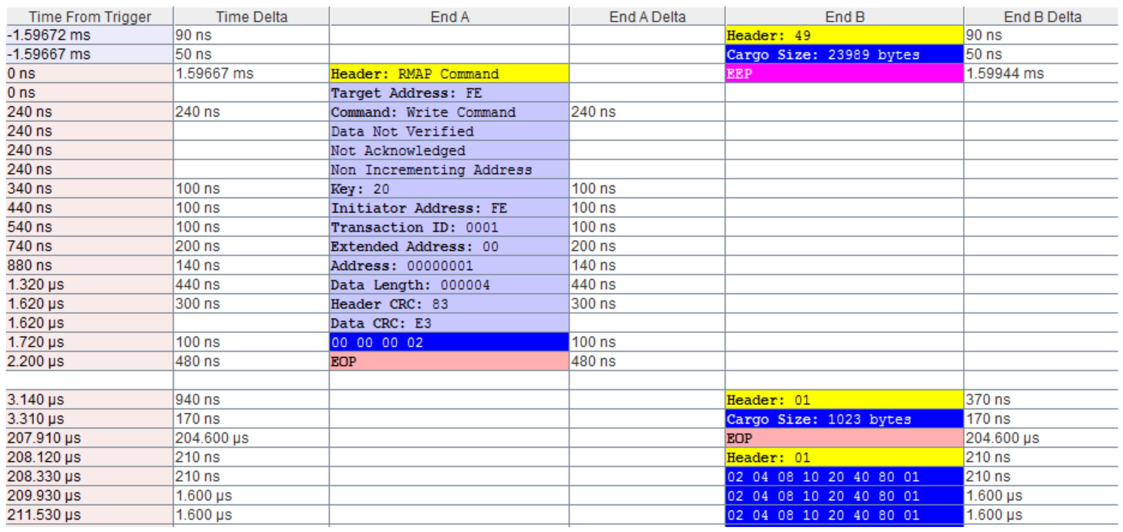
When a “mode1” RMAP write command is received on SpaceWire interface one, 64kByte packets of random data are transmitted from interface two repeatedly at a data rate of 150Mbits/s.



Time From Trigger	Time Delta	End A	End A Delta	End B	End B Delta
0 ns		Header: RMAP Command			
0 ns		Target Address: FE			
200 ns	200 ns	Command: Write Command	200 ns		
200 ns		Data Not Verified			
200 ns		Not Acknowledged			
200 ns		Non Incrementing Address			
300 ns	100 ns	Key: 20	100 ns		
400 ns	100 ns	Initiator Address: FE	100 ns		
500 ns	100 ns	Transaction ID: 0000	100 ns		
700 ns	200 ns	Extended Address: 00	200 ns		
800 ns	100 ns	Address: 00000001	100 ns		
1.200 µs	400 ns	Data Length: 000004	400 ns		
1.500 µs	300 ns	Header CRC: AF	300 ns		
1.500 µs		Data CRC: 91			
1.600 µs	100 ns	00 00 00 01	100 ns		
2.040 µs	440 ns	EOP	440 ns		
2.690 µs	650 ns			Header: 01	
2.740 µs	50 ns			Cargo Size: 65535 bytes	50 ns
4.37166 ms	4.36892 ms			EOP	4.36892 ms
4.37175 ms	90 ns			Header: A8	90 ns
4.37184 ms	90 ns			Cargo Size: 65535 bytes	90 ns
8.74079 ms	4.36895 ms			EOP	4.36895 ms
8.74088 ms	90 ns			Header: 1D	90 ns
8.74097 ms	90 ns			3B 76 ED DA B4 69 D2 A5	90 ns
8.74149 ms	520 ns			4B 96 2D 5B B7 6F DE BD	520 ns
8.74201 ms	520 ns			7B F6 ED DB B6 6D DA B4	520 ns

64kByte Packets with Random Data Transmitted at 150Mbits/s

When a “mode2” RMAP write command is received on SpaceWire interface one, 1kByte packets consisting of walking ones data are transmitted from interface two repeatedly at 50Mbits/s.



Time From Trigger	Time Delta	End A	End A Delta	End B	End B Delta
-1.59672 ms	90 ns			Header: 49	90 ns
-1.59667 ms	50 ns			Cargo Size: 23989 bytes	50 ns
0 ns	1.59667 ms	Header: RMAP Command		EEP	1.59944 ms
0 ns		Target Address: FE			
240 ns	240 ns	Command: Write Command	240 ns		
240 ns		Data Not Verified			
240 ns		Not Acknowledged			
240 ns		Non Incrementing Address			
340 ns	100 ns	Key: 20	100 ns		
440 ns	100 ns	Initiator Address: FE	100 ns		
540 ns	100 ns	Transaction ID: 0001	100 ns		
740 ns	200 ns	Extended Address: 00	200 ns		
880 ns	140 ns	Address: 00000001	140 ns		
1.320 µs	440 ns	Data Length: 000004	440 ns		
1.620 µs	300 ns	Header CRC: 83	300 ns		
1.620 µs		Data CRC: E3			
1.720 µs	100 ns	00 00 00 02	100 ns		
2.200 µs	480 ns	EOP	480 ns		
3.140 µs	940 ns			Header: 01	370 ns
3.310 µs	170 ns			Cargo Size: 1023 bytes	170 ns
207.910 µs	204.600 µs			EOP	204.600 µs
208.120 µs	210 ns			Header: 01	210 ns
208.330 µs	210 ns			02 04 08 10 20 40 80 01	210 ns
209.930 µs	1.600 µs			02 04 08 10 20 40 80 01	1.600 µs
211.530 µs	1.600 µs			02 04 08 10 20 40 80 01	1.600 µs

1kByte Packets with Walking Ones Data Transmitted at 50Mbits/s

When the “mode2” RMAP write command is received an immediate state transition occurs. This results in an EEP being appended to the current packet in transmission from SpaceWire interface two and the remaining packet cargo not being sent. Alternatively it is possible to specify transitions to occur once the current schedule completes or once the current packet transmission completes.

When a “mode0” RMAP write command is received on SpaceWire interface one, the EGSE stops transmitting data from interface two.

Conclusion

This application note demonstrates how the SpaceWire EGSE and its associated scripting language could be used to very quickly simulate a SpaceWire instrument. It has introduced some of the key concepts of the EGSE scripting language (link speed configuration, pattern matching, events, variables, packet definitions, scheduling and state machines), shown one way in which the EGSE can be operated (script creation, compilation and EGSE configuration via the GUI application) and shown the performance possible thanks to the EGSE's ability to operate independent of software.

This example is relatively simple and only touches on the range of features both the EGSE hardware and software provide. For more information please visit our website at www.star-dundee.com or contact us at enquiries@star-dundee.com.