Verification of a SpaceWire C&DH Bus Using SystemC

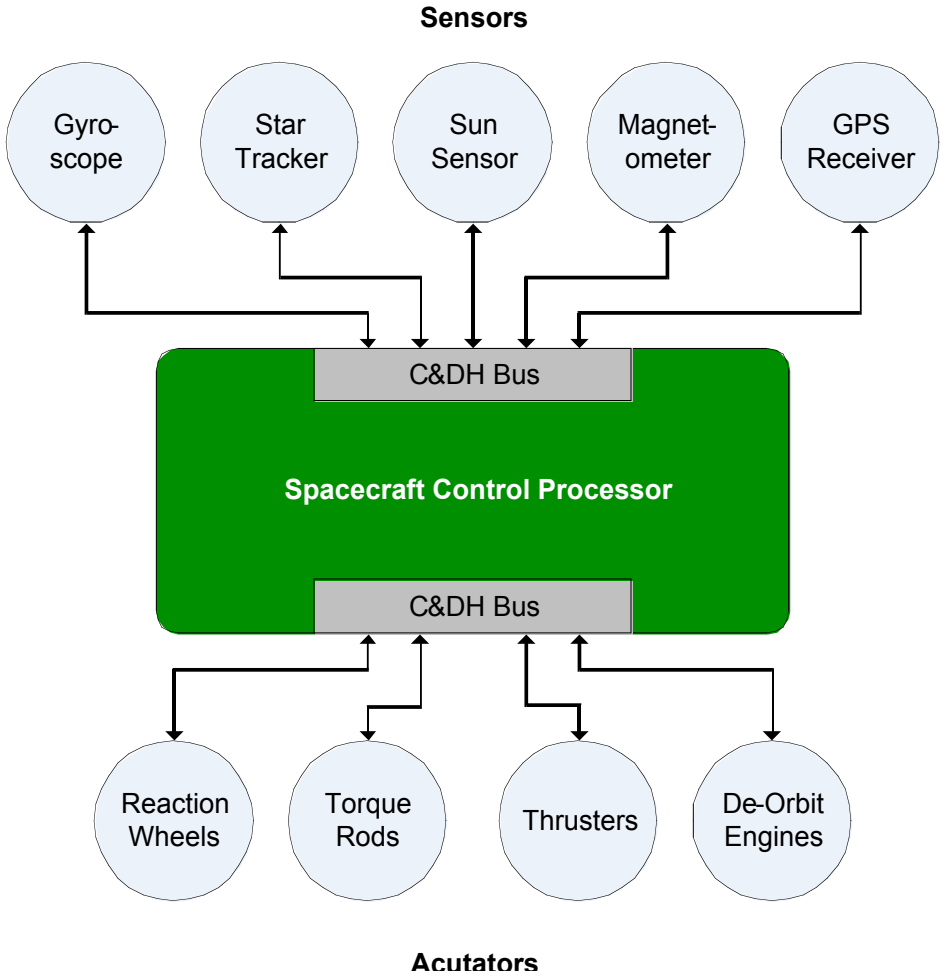
J. R. Petrus
james.r.petrus@lmco.com

Advanced Payload Electronics IRaD
Lockheed Martin Space Systems Company

June 4, 2007

What is a spacecraft C&DH bus?

- The spacecraft control processor (SCP) runs the flight software which is responsible for maintaining the vehicle's attitude (angular orientation).
- The Command and Data Handling (C&DH) Bus is the low to medium speed link between the SCP, sensors, and actuators.



SpaceWire Introduction

- Emerging standard proposed by the European Space Agency (ESA) for C&DH, derived from IEEE-1355
- Full duplex, point-to-point serial link
- Can create networks using nodes (endpoint with 1 port) and routers (standalone bridge with multiple ports)
- Error handling and redundant link capabilities
- Superior alternative to traditional C&DH buses: more bandwidth, longer distances, less power
- Performs comparably to IEEE-1394a (Firewire) and USB 2.0; typical bit rates are 10 to 200 Mbps and has been shown to operate up to 400 Mbps.

SpaceWire Packet

Logical Destination Address	Data Character 0	Data Character 1	Data Character 2	Data Character 3	...	Data Character N	EOP Control Character

*** SpaceWire Data Character = 1 Parity Bit, 1 Control Bit, and 8 Data Bits**

GAP Introduction

General Access Protocol is a layer-3 protocol proposed by LM to perform DMA transfers over SpaceWire networks. GAP enables processor cards to access peripherals using a simple read/write request/response bus.

GAP Write Request Packet

Logical Destination Address (1 char)	Protocol ID (1 char)	Command ID (1 char)	Command Options (1 char)	Return Address (1 char)	Reserved (1 char)	Transaction ID (2 chars)	Write Address (4 chars)

Length (3 chars)	Header Checksum (1 char)	Data Characters (N chars)	Data Character (1 char)	...	Data Character (1 char)	Packet Checksum (1 char)	EOP Control Character

Gap Write Response Packet

Logical Destination Address (1 char)	Protocol ID (1 char)	Command ID (1 char)	Command Options (1 char)	Return Address (1 char)	Result Code (1 char)	Transaction ID (2 chars)	Packet Checksum (1 char)	EOP Control Character

System Description

Sensor Data Processing Board:

- SpaceWire Router FPGA to facilitate communication with the SCP
- 3 FPGA processing elements each with
  - Unverified C&DH Subsystem (GAP/SpaceWire node)
  - Internal and external GAP-accessible memories
  - Additional mission-specific functionality

Spacecraft Control Processor:

- PPC single board computer
- Can control multiple sensor data processing boards
- On-board SpaceWire router + PCI bridge
- Runs QNX

System Diagram

Verified 3rd party IP:

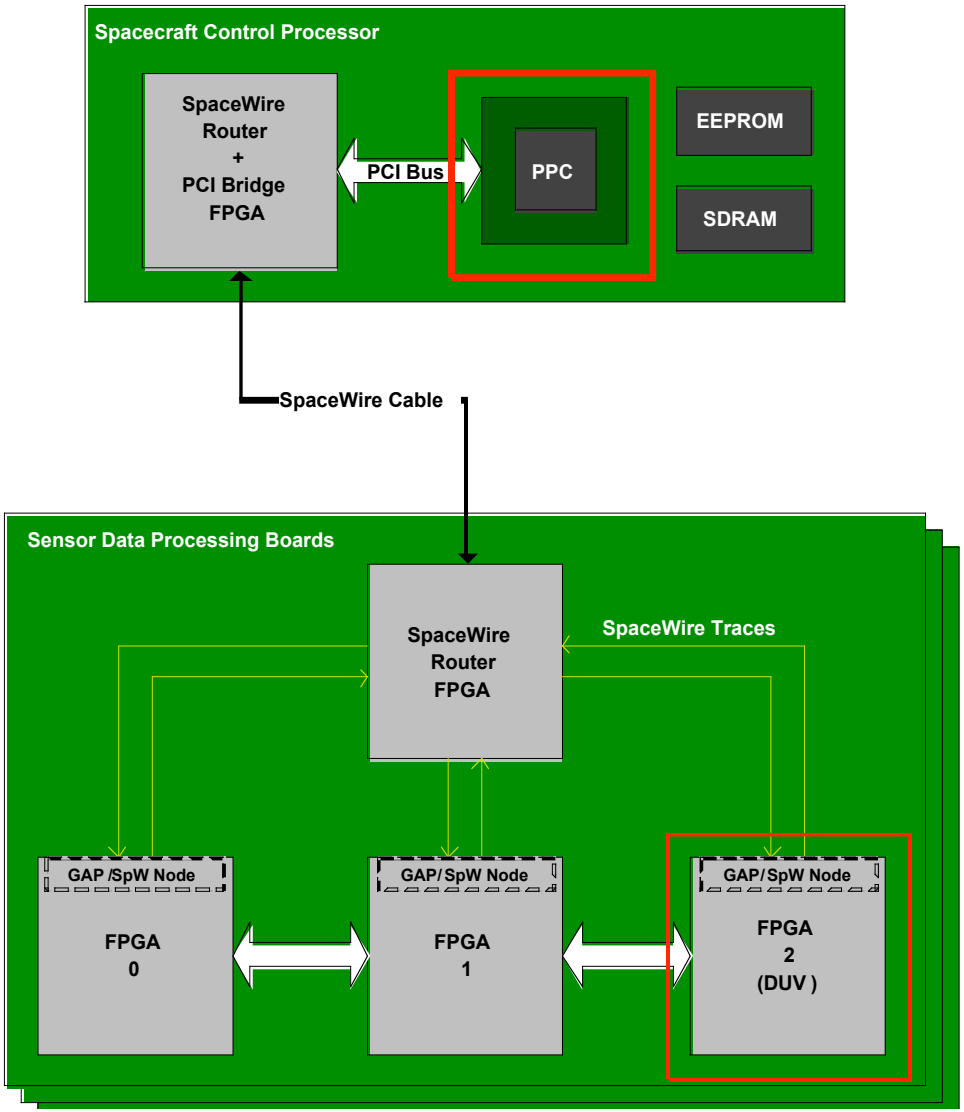
- SpaceWire Router
- PCI Bridge

Unverified RTL:

- SpaceWire Node IP
- GAP Node RTL
- Mission-specific RTL

Untested software:

- SpaceWire Device Driver
- SpaceWire Packet Class
- GAP Packet Class



Verification Goals for the C&DH Subsystem

- Prove that:
  - SpaceWire Node IP functions as promised (it was free)
  - C&DH Subsystem is functionally correct (GAP node RTL)
  - Mission specific FPGA 2 RTL is correct

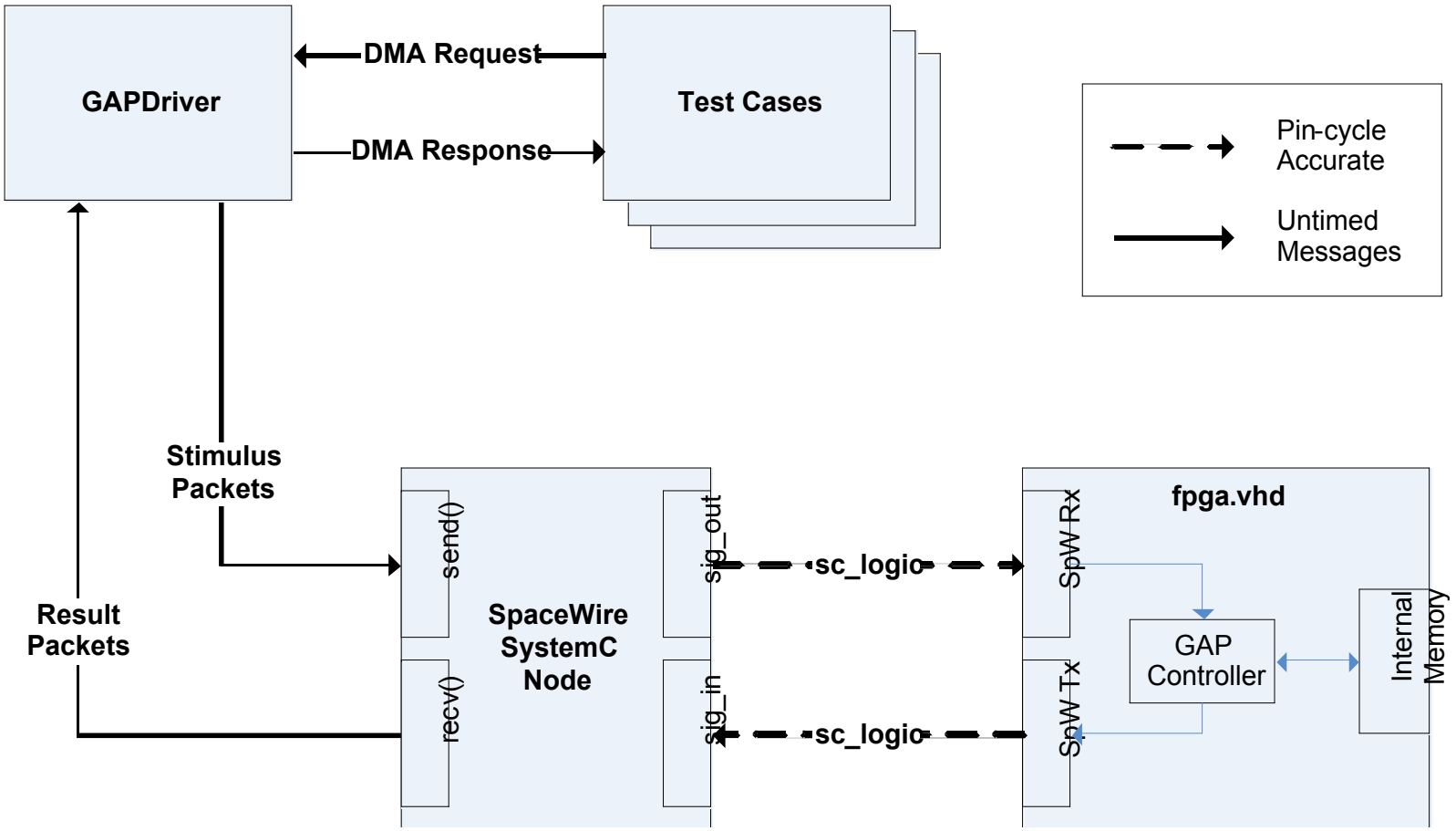
- Schedule/funding constraints: simulate behavioral RTL only

- Test that a stimulus packet yields an expected result some time later. We called this “packet-cycle-accuracy.”

- Develop a generic, reusable GAP/SpaceWire module

- Reduce risk by integrating existing software products into the verification environment

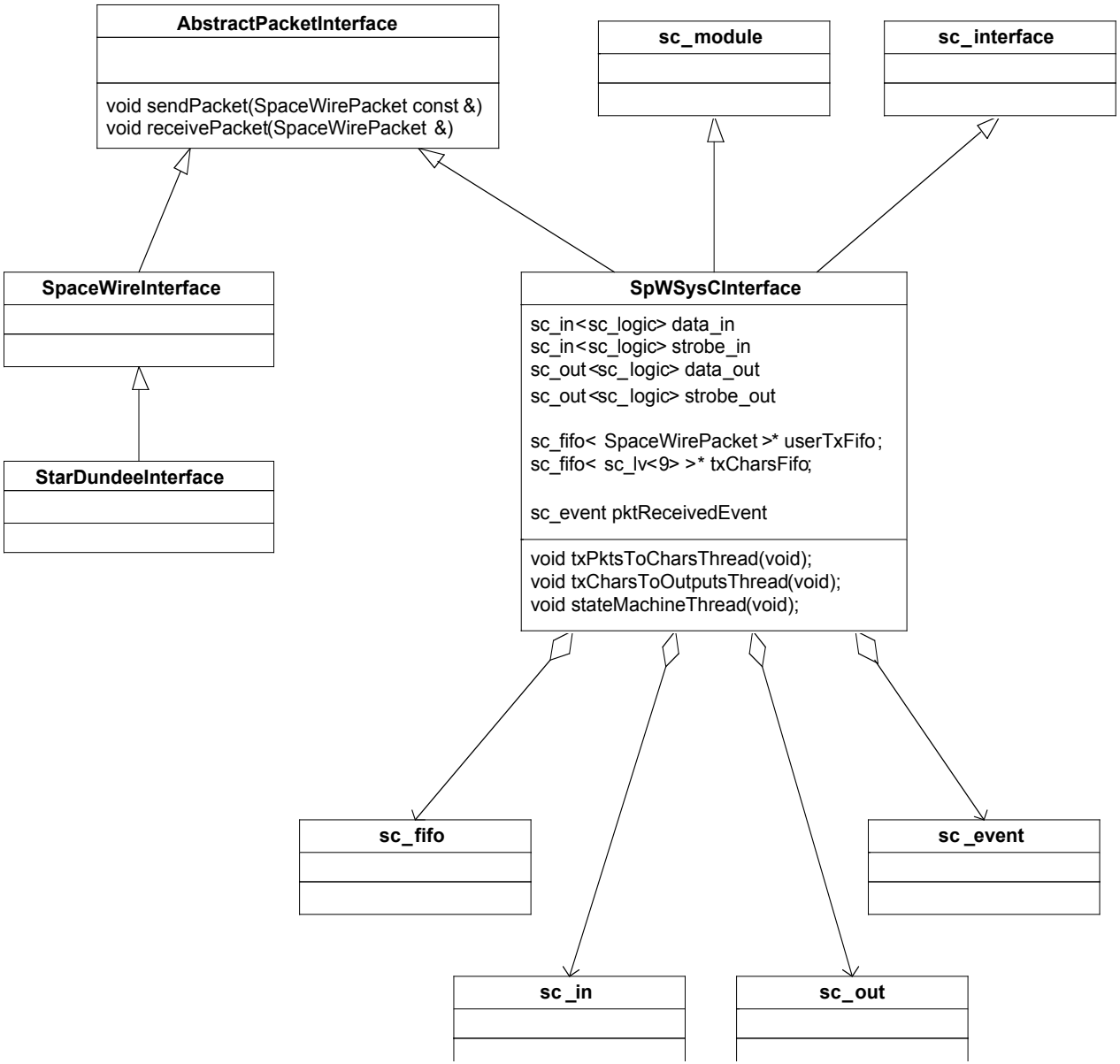
Verification Environment: packet-cycle-accuracy



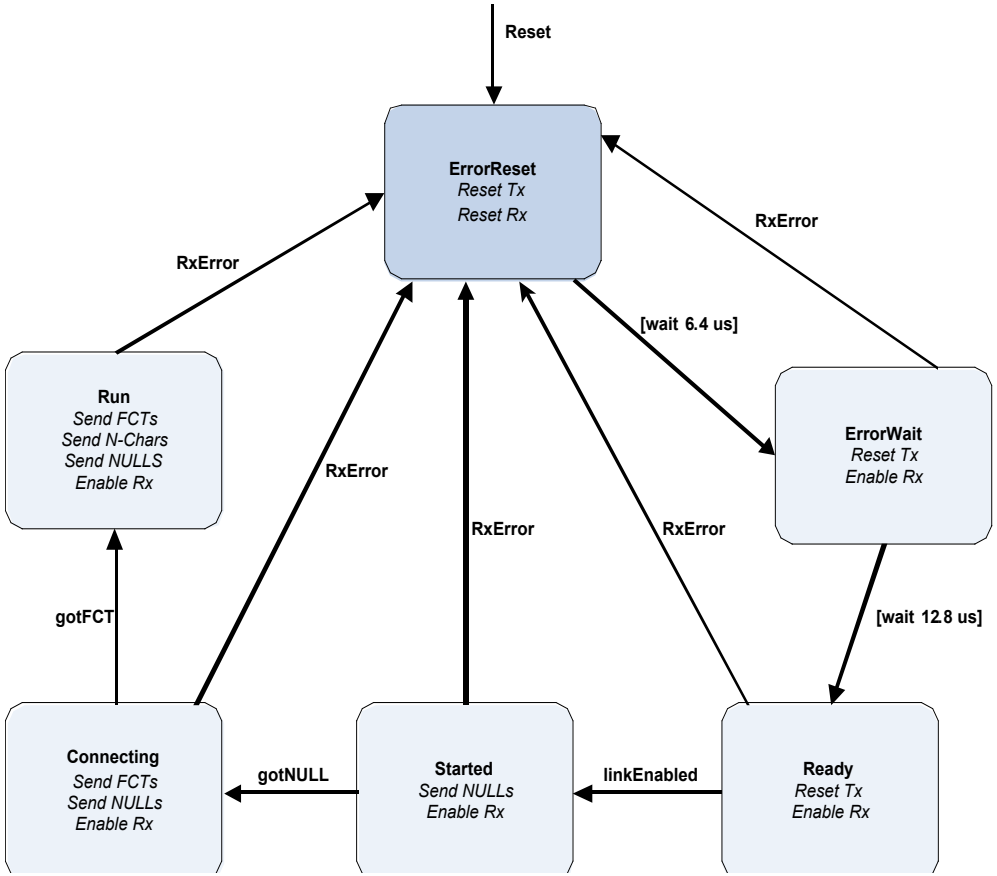
SystemC SpaceWire Module Requirements

- Transport SpaceWire packets between the untimed testbench and pin-cycle accurate RTL Design Under Verification (DUV)
- Act as an event-driven device driver for the testbench
- Utilize existing GAP/SpaceWire software written in C++
- Serialize and deserialize data to/from the RTL
- Error insertion capabilities
- Compliant with the European Space Agency specification (ECSS-E-50-12A)

SystemC SpaceWire Module



SpWSysCInterface: stateMachineThread()

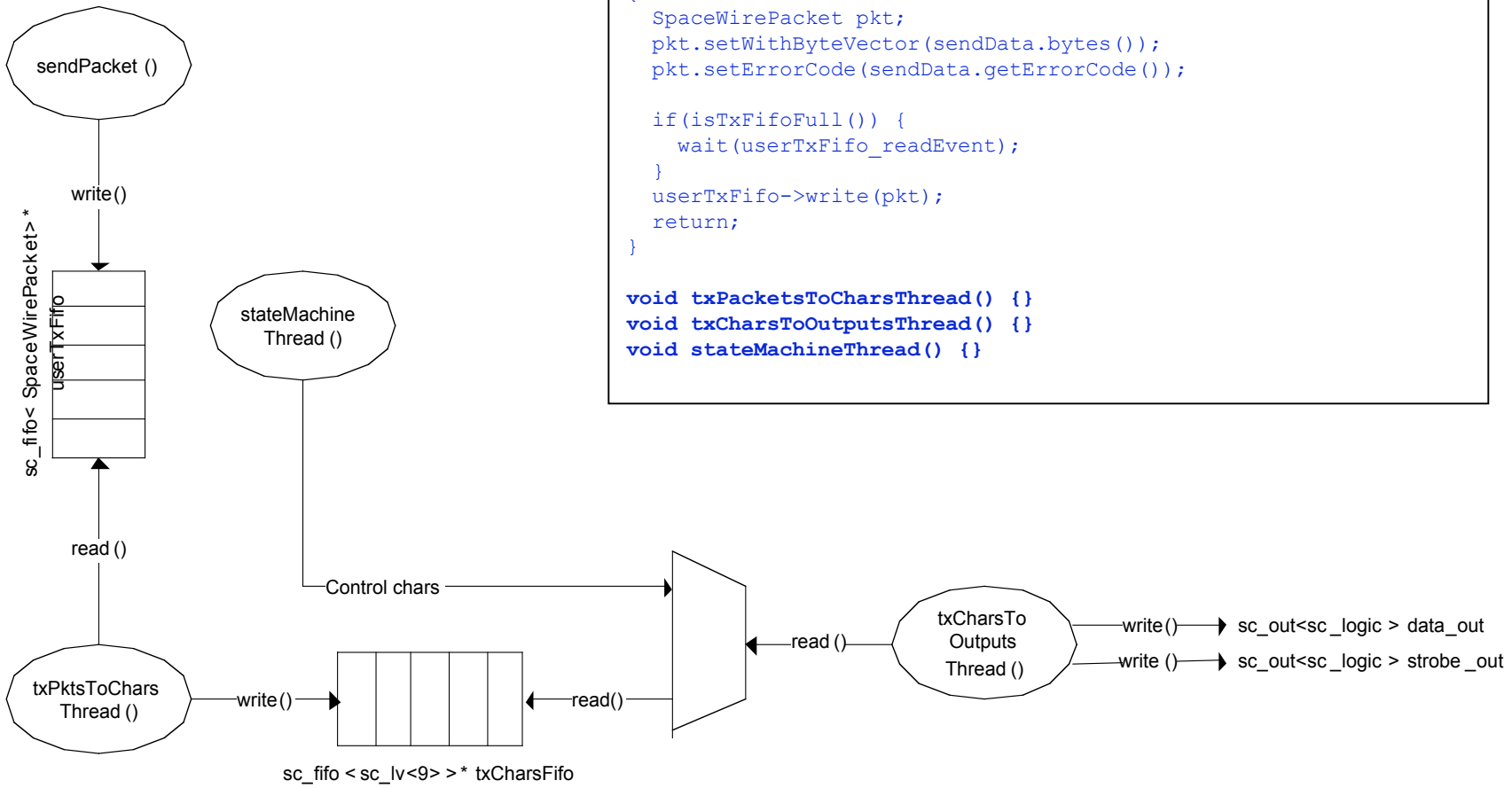


SpaceWire Link State Machine

(ECSS-E-50-12A, p. 60)

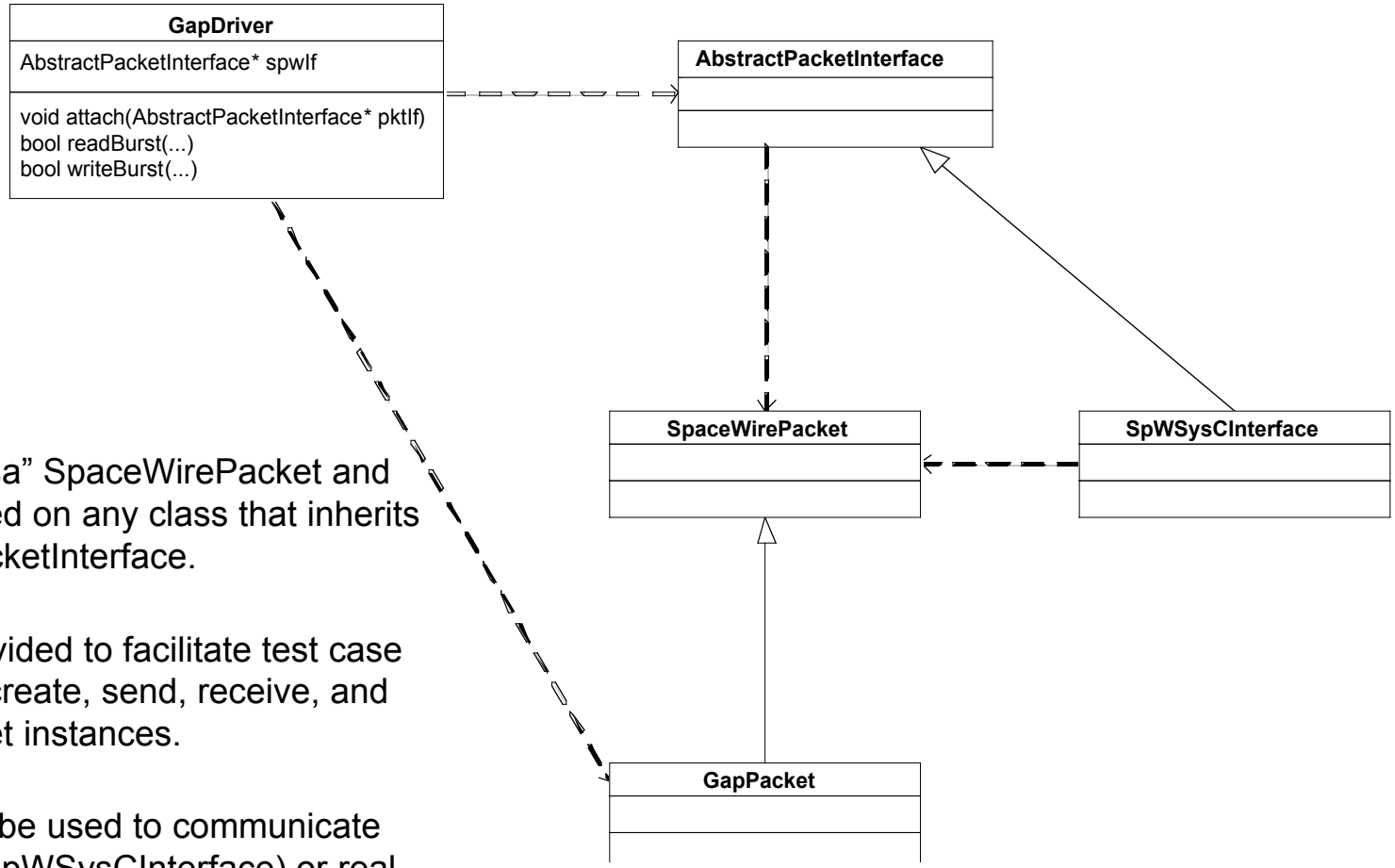
```
void SpWSysCInterface::stateMachineThread() {

  while(true){
    wait(clock.posedge_event() | reset_in.default_event());

    switch(state){
      case(ERROR_RESET):
        resetTx = SC_LOGIC_1;
        resetRx = SC_LOGIC_1;

        wait(6400, SC_NS, reset_in.posedge_event() );
        if(reset_in->read() == SC_LOGIC_1) {
          state = ERROR_RESET;
        }
        else {
          state = ERROR_WAIT;
        }
        break;

      // other states not shown

      case(ERROR_WAIT):

      case(READY):

      case(STARTED):

      case(CONNECTING):

      case(RUN):
```

SpWSysCInterface: Packet Transmission




```
void SpWSysCInterface::sendPacket(SpaceWirePacket const &sendData)
{
  SpaceWirePacket pkt;
  pkt.setWithByteVector(sendData.bytes());
  pkt.setErrorCode(sendData.getErrorCode());

  if(isTxFifoFull()) {
    wait(userTxFifo_readEvent);
  }
  userTxFifo->write(pkt);
  return;
}

void txPacketsToCharsThread() {}
void txCharsToOutputsThread() {}
void stateMachineThread() {}
```

GAP Transactions using SpWSysCInterface



- A GAP packet “isa” SpaceWirePacket and can be transmitted on any class that inherits from AbstractPacketInterface.
- GapDriver is provided to facilitate test case creation. It can create, send, receive, and parse GAPPacket instances.
- GapDriver could be used to communicate with simulated (SpWSysCInterface) or real (StarDundeeInterface) SpaceWire device drivers.

Interfacing with C++ Test Cases

```
// testcase.h

#include "systemc.h"
#include "gapdriver.h"
#include "abstractpacketinterface.h"

class TestCase: public sc_module, public sc_interface
{
  public:
    TestCase(sc_module_name n, AbstractPacketIf* pif);
    // etc…

  protected:
    GapDriver gapIf;
    void mainThread();
    // etc…

  SC_HAS_PROCESS(TestCase);
};

TestCase::TestCase(sc_module_name n,
  AbstractPacketIf* pif): sc_module(n)
{
  gapIf.attach(pif);
  SC_THREAD(mainThread);
}
```

```
// testcase.cpp
#include <string>
#include <iostream>
#include "testcase.h"

void TestCase::mainThread() {
  uint32_t i;
  uint32_t memAddr = 0x100;
  uint8_t wdata[1024];
  uint8_t* rdata = 0;

  wait(startEvent);

  if(!gapIf.writeBurst(memAddr, wdata, 1024)) {
    std::cerr << "GAP write error.";
  }

  if(!gapIf.readBurst(memAddr, rdata, 1024)) {
    std::cerr << "GAP read error.";
  }

  doneEvent.notify();
}
```

SystemC for Verification: Advantages

- Leveraged existing C++ knowledgebase: our engineers knew C++ but not Vera or SystemVerilog
- Compatibility with STL and standard C types allowed for easy integration with legacy code without the need for FLI/PLI/DPI.
- Top-down design flow encouraged refinement of abstract concepts
- SCV enabled constrained randomization and seed management: it was very easy to randomize packet length, payload data, and bit error rates
- High-level of abstraction enabled engineers to develop modules with more functionality than RTL in less time
- C&DH subsystem verification was a huge success: the SpaceWire core worked perfectly and the simulation was agile enough to help debug GAP problems observed in the lab.

Module		Lines
SpaceWire Node	VHDL	16,457
	SystemC	1,797
GAP Node	VHDL	2,461
	SystemC	1,526

Closing Remarks

- Our future verification environments will have even tighter integration with software; perhaps the flight software could be made abstract enough to run as a SystemC module?

- Definite market for SystemC verification intellectual property, especially communication interfaces. Most of our verification effort is spent writing standards-compliant interface modules. We've also created SystemC UART, SPI-3, and PPC-60x bus modules to name a few.

- More about SpaceWire: http://spacewire.esa.int

Questions?