Standardisation of SpaceWire Software APIs

Stuart Mills, Alex Mason – STAR-Dundee
Steve Parkes – University of Dundee
Takayuki Yuasa – JAXA/ISAS

Fourth International SpaceWire Conference
10th November 2011

Introduction – APIs and STAR-Dundee

What's An API?

- Application Programming Interface
- According to Wikipedia:
  - *".. a particular set of rules ('code') and specifications that software programs can follow to communicate with each other."*
- If an API changes, then the software accessing the API must also change
- If one software module provides the same API as another, the two modules can be used interchangeably

STAR-Dundee's API Experience

- Developed APIs to provide interfaces to our devices
  - Our first SpaceWire API was released over 10 years ago, a few years before SpaceWire was standardised
  - Allow users to write software to perform unique tasks using our standard devices
- Provide similar APIs for different device types
  - SpaceWire PCI API
  - SpaceWire USB API
- APIs consistent across platforms
- Worked with NEC Toshiba Space Systems in porting USB API to Space Cube

Latest STAR-Dundee API

- Recently released a new software stack and API (STAR-System)
  - Will support all new and future STAR-Dundee devices

- Consistent interface for all device types
- Consistent interface and behaviour on all platforms
  - Windows, Linux, QNX, VxWorks, …
- Newer versions of the API will be consistent with older versions

- Designed to expose features required during development and testing of SpaceWire devices and networks

STAR-Dundee API Performance

- All STAR-Dundee APIs and drivers are written to provide high performance
- Allow traffic to be transmitted and received at very high speeds, without much load on the processor

- For example, if the transmit function only allows one packet to be transmitted:
  - The packet will be DMAd to the transmitting device
  - The device will be instructed to transmit the packet
  - The device will generate an interrupt when the packet is transmitted
  - The interrupt will be dealt with by the processor
  - Finally the user application will be informed the packet has been transmitted

- If the transmit function allows multiple packets to be transmitted:
  - The above steps only need to occur once for all packets

SpaceWire APIs

SpaceWire APIs

- No standards or even recommendations for SpaceWire APIs
- Each hardware manufacturer can provide a completely different API for accessing each device
- Greatly reduces opportunities for software reuse

- Test and development equipment will provide different features to flight equipment
- But likely to be a number of features which are consistent

Using Existing APIs

- POSIX Sockets API is most likely candidate

- STAR-Dundee's Router-USB and Brick supports the Sockets API on Linux
  - But strongly discourage users from using this
  - Other than when investigating TCP/IP over SpaceWire

Using The Sockets API

- Sockets API doesn't expose features specific to SpaceWire
- Additional APIs would also be needed to configure devices

- Some cases where Sockets API could be useful
  - `send()` and `recv()` functions would probably need modified to transmit/receive one SpaceWire packet

- Would allow developers to use familiar API
- But unlikely to provide high performance

Typical SpaceWire APIs

- Not just limited to transmitting and receiving packets
- Support for protocols carried over SpaceWire
- Functions for configuring devices

Packet Transfer APIs

- Easy to assume this is quite simple
- But important to provide a high performance interface
- Also need to provide functions for opening and closing connections to a device
- May also need to provide test and development functions

RMAP APIs

- Can be split up depending on functionality required
- RMAP Packet API
- RMAP Initiator API
- RMAP Target API

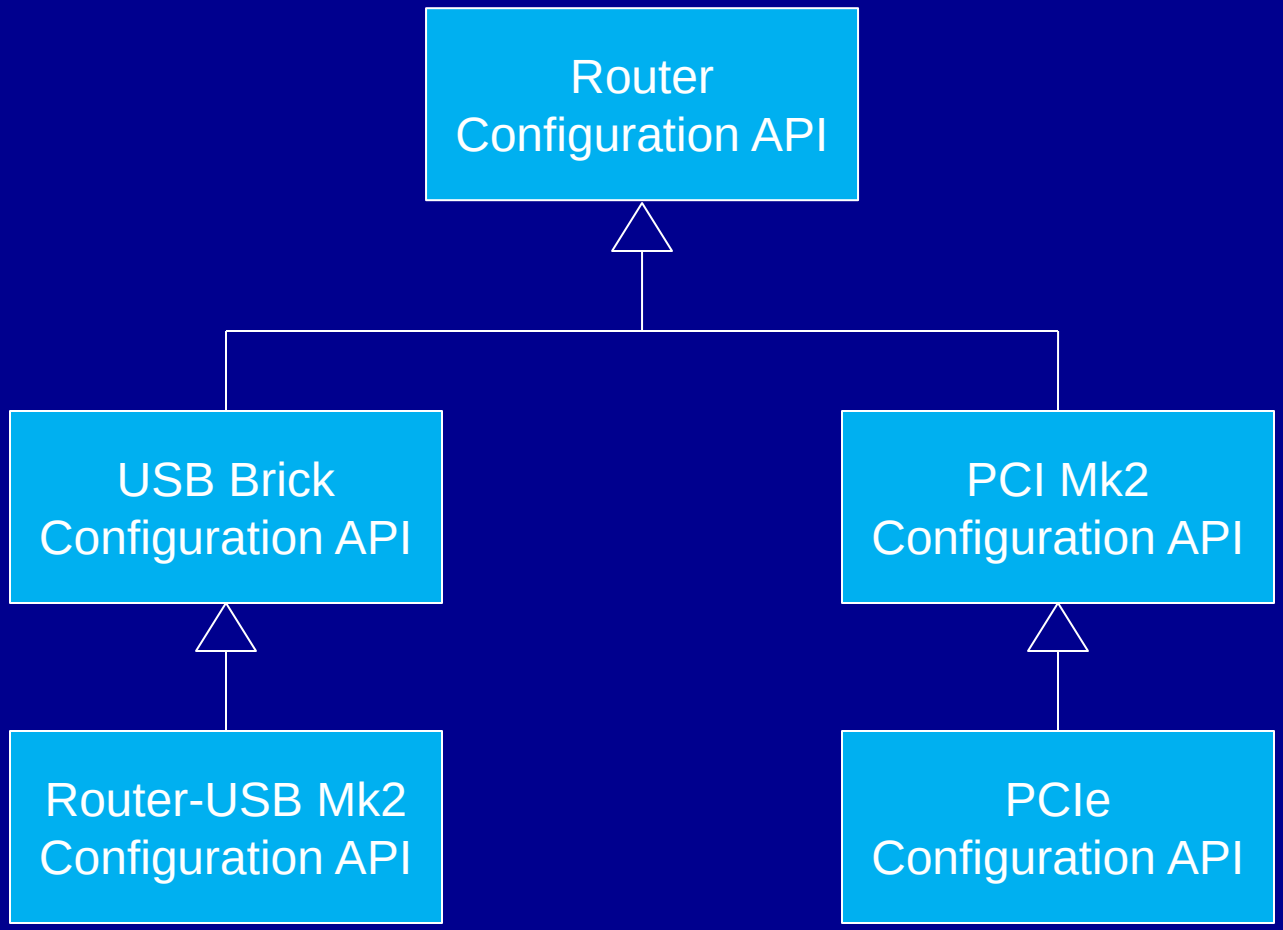
Other Protocol APIs

- E.g. CCSDS Packet Transfer Protocol, GOES-R RDDP, SpaceWire-PnP
- As with RMAP, can be split in to a packet building and interpreting API and an implementation API
- API required will depend on software being written
- Some protocols will already have a standardised API which can be used

Device Configuration APIs

- Functions to configure the features of devices
- Difficult to standardise due to differences between devices
- Some features common to a number of devices
- Additional functions specific to device types
- Some functions specific to an individual device
- SpaceWire-PnP will make things easier

STAR-System Device Configuration



Summary, Conclusions and Future

Summary

- Many different APIs exist to access SpaceWire devices
- Typical SpaceWire APIs:
  - Packet Transfer API
  - Protocol APIs
  - Device Configuration APIs
- Using existing APIs with SpaceWire is not ideal

Conclusions

- The time required for a developer to learn a new API can be considerable
- Mistakes made when developing with an unfamiliar API can be costly

- Standardisation would bring other benefits:
  - "Shim" layers would no longer be required to deal with differences between device types
  - Software could be developed and tested on existing test equipment before being moved to new flight equipment

- SpaceWire is intended to encourage reuse
- Software cannot easily be reused between projects unless software APIs are standardised

What Next?

- Japanese agencies, academia and industry have identified the importance of standard SpaceWire APIs
- STAR-Dundee has a “standard” API to be used by all future STAR-Dundee devices

- It is important that the rest of the SpaceWire community isn’t left behind, or is forced to accept standard APIs which do not meet their needs

- The entire SpaceWire community must therefore take responsibility for any software standardisation efforts