SpaceWire on FPGA – Challenges and Solutions

Dr Barry M Cook, C Paul H Walker

4Links Limited, PO Box 816, Bletchley Park, Milton Keynes, MK3 6ZP England,
Email: [Barry, Paul] @4Links.co.uk

ABSTRACT

SpaceWire is characterised by transmission on two signal lines that encode both the data and its clock. Clock recovery and design of the receiver appears to be simple but presents two distinct implementation issues, both presenting serious challenges for FPGA implementation (and are non-trivial challenges for ASIC implementation).

The first challenge is that of the tight timing constraints required to use the recovered clock. The second is that the received data/tokens are in the clock domain of the remote transmitter (the recovered clock) and must be re-timed to the receive clock. These constraints and asynchronous interfaces are not easy to guarantee in FPGA – nor is their design well supported by design tools.

We explain these issues and then describe an alternative approach using a fully-synchronous technique. The result is a robust design that meets the assumptions of design tools and can be implemented very easily in FPGA – and ASIC.

1. INTRODUCTION

SpaceWire [1] is a high speed (2 to >400Mb/s) digital communication standard. It is characterised by transmission on two signal lines that encode both the data and its clock. Clock recovery at the receiver is as simple as an exclusive-or of the two signal lines. This recovered clock can be used to decode the serial data stream into its constituent tokens. The tight timing constraints required to use the recovered clock are difficult to guarantee as the commonly available FPGA design tools do not allow a simple specification of the required constraints and users must resort to hand-placement of the logic cells required for this function. The recovered clock net may also have to drive the logic decoding the data and may require the use of clock buffers further limiting placement choices. Explicit placement of SpaceWire logic restricts the placement and routing of other parts of the design leading to reduced usability of the silicon.

Received data/tokens are in the clock domain of the remote transmitter (the recovered clock) and must be re-timed to the receive clock. Transmit and receive clocks need not be the same frequency and, even if nominally the same, may drift relative to each other in frequency or phase. Passing data between clock domains is known to be difficult [2]. Issues such as meta-stability can be handled for single-bit signals – although at the expense of limiting the data rate between clock domains. Attempts to improve throughput using multi-bit parallel transfers bring issues of path delay matching that add further difficulties in FPGA implementations.

Of the many faulty SpaceWire implementations we have seen and analysed, most were due to the asynchronous interfaces. Unfortunately, asynchronous design is poorly supported by silicon design tools and simulation is particularly difficult. Asynchronous errors have been seen in practice even after extensive simulations detected no problems.

We present an alternative design strategy that is fully synchronous and, as a result, can be implemented in FPGA (and ASIC) with no more constraint than specifying the (single) system clock frequency. Design tools are thus not limited in their ability to place user logic and hence are able to fully utilize the silicon. 4Links has built a large number of SpaceWire interfaces with this strategy and all have proved both simple to design and extremely robust in operation at speeds up to 500Mb/s.

We begin by reviewing the commonly used asynchronous design technique, showing where difficulties arise, and proceed to describe the alternative, fully synchronous, approach that has proved to be easy to implement and robust in operation.

2. SPACEWIRE LINE CODING AND CLOCK/DATA RECOVERY

SpaceWire sends its data signal (D) along with a "Strobe" (S) signal such that there is a transition on one

signal for each bit period. An example sequence is shown if figure 1.

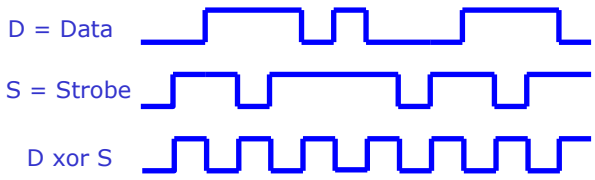

Figure 1 Signals D, S and D xor S

The source clock signal can be recovered by an exclusive-or of D and S and D must be sampled on both edges of the recovered clock. Re-timing the recovered data to a single clock edge is convenient. Suitable logic is shown in figure 2.

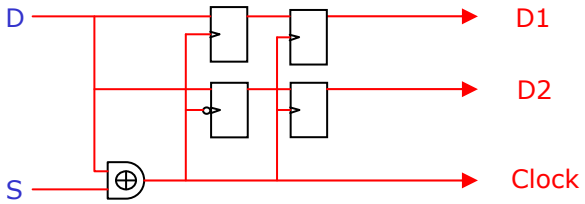

Figure 2 Logic to recover data from D and S

Although apparently simple, this logic demands care in obtaining suitable time delays through various paths:

- To maximise performance (data rate) the delays from the D and S pins to the exclusive-or gate must be matched;
- Set-up and hold times for the first data latches, FF1 and FF2 must be met
  - The time delay through the exclusive-or to the clock inputs must not be too short to prevent incorrect operation
  - The time delay through the exclusive-or to the clock inputs must not be too long to prevent performance degradation

FPGA implementations, where use is made of general-purpose interconnect to flip-flops within cells whose placement depends on other logic within the device rarely provide tools with sufficient control of placement to meet these requirements directly. Recourse then has to be made to hand placement of cells.

Further complication arises when the recovered clock is used for logic that decodes the data stream into SpaceWire tokens. Local (signal) nets often have too much skew for the clock signal resulting in

- The need to use a clock buffer, often at the edge of a chip with attendant restrictions on the location of the SpaceWire circuit, and/or
- More hand placement of logic

Hand placement results in a longer design cycle, more difficulty in making changes and more restrictions on the use of the rest of the chip.

3. CLOCK DOMAINS AND WHERE TO PLACE THE BOUNDARIES

Received data has to be decoded to SpaceWire tokens and results in two or three distinct signals

- Received data
- Received flow control tokens
- Received time codes (optional)

The most obvious implementation places all the decode logic in the transmitters time domain – using the recovered clock signal. The resulting tokens must then cross into the local clock domain.

It is also possible to divide the decode logic between domains, for example by collecting a few received bits, un-decoded, and passing then to a decoder in the local clock domain.

4. CROSSING CLOCK DOMAINS

Transferring data between clock domains is not easy, and it may not be fast.

In order to be received correctly, data presented to a flip-flop must not change within a critical period around the active edge of the clock signal. This is normally expressed as the data needing to be stable for a period before the clock edge – the setup time – and then not changing until after the clock edge – the hold time.

When the signal is produced in one clock domain and received in another there can be no guarantee of meeting these timing requirements. In general the clock frequencies will be different, even if nominally the same but differing by a few parts per million. Sooner or later, data that changes during the critical period will be presented.

When the setup and hold times are violated the output of the flip-flop will be unstable and will take time to fall into a stable state (metastability). The effect rapidly becomes more apparent as clock frequencies increase. The time to become stable is unbounded but the

likelihood of being in the unstable state reduces exponentially with time.

The most widely used technique to reduce the effect of metastability is to use two cascaded flip-flops clocked with the same signal – figure 3. This can reduce the problem by many orders of magnitude.

A single flip-flop at tens of Mega-Hertz is very likely to show metastability. A pair of flip-flops will normally reduce the incidence to something negligible. In extreme cases a third flip-flop will need to be added. It is possible to calculate the frequency of occurrence of this effect, see [2] and numerous technical notes from silicon vendors.

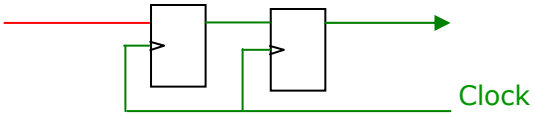

Figure 3 Two cascaded flip-flops, input from clock domain A, clocked in domain B

If it is necessary to know that the data arrived safely a handshake, reply, signal must be passed in the other direction – again requiring two, or more, flip-flops. A total round-trip delay of at least two clock cycles in each clock domain.

At first sight, some optimisation can be obtained by transferring multiple signals under the control of a single handshake. When a parallel set of signals is ready to be transferred a single-bit "ready" signal is transferred from A to B; B takes the parallel data and then sends a handshake back.

On further consideration, however, we must be sure that the routing delays on all the signals is the same as or less than the delay on the ready signal or we may latch data that is not yet ready. This also usually requires hand placement and its attendant blocking of chip area for other logic.

We must also be careful that the transfers are fast enough to meet the rate at which tokens are received. Data tokens at 10-bits and time-codes at 14 might be expected to cope with 2 cycles transfer delay in each clock domain but consecutive flow-control or end-of-packet tokens at 4-bits might not.

Use of asynchronous FIFO's is a possibility – but these must be carefully designed and do not seem to be the ideal choice for time-codes or flow-control. They represent a less effective use of silicon than should be possible.

5. SYNCHRONOUS DESIGN

If we could move the time domain boundary right up to the pins on the chip and do all SpaceWire decoding in the local clock domain. By using the local clock to sample D and S in the input buffers we can produce a completely synchronous design – see figures 4 and 5. We need only a single clock net for both transmit and any number of receivers.

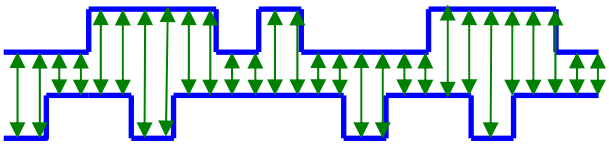

Figure 4 Sampling D and S with the local clock

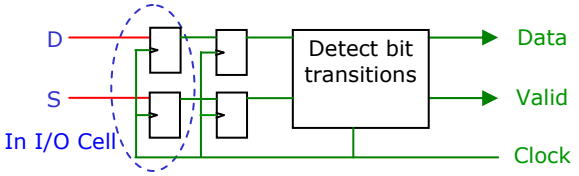

Figure 5 Sampling Logic

At a stroke we can eliminate all difficult issues – there is no need to hand place anything and the normal design tools can be left with a simple clock speed requirement and total freedom to use any part of the chip for SpaceWire and other logic.

Sampling at the pins is the only point at which there is any asynchronous logic – we still need two flip-flops to avoid metastability but on only two signals, D and S (the Gray code used ensures that only one of these will change per bit).

Having recovered the data and strobe signals we must determine the location of the bit boundaries. These occur whenever there is a change in level in either D or S and simple edge detectors can be used.

Instead of having a data and clock signal in an external time domain we have a data and valid signal in the local clock domain.

SpaceWire decoding proceeds as usual but all in the local clock domain – we have no performance limiting clock domain crossings to deal with.

We must sample the incoming bits fast enough not to miss any. We are used to thinking of sampling at many times the data rate – oversampling – maybe 8 to 16 samples per bit in order to be sure of correctly synchronizing to bit boundaries. One significant advantage of the Data-Strobe encoding that SpaceWire uses is that the clock signal is directly available to us

and, in fact, we need only sample at (just) greater than one sample per SpaceWire bit.

Imperfect transmission lines cause edges to move from their ideal locations and one sample per bit has no margin to cope with this. In practice, a sample rate of 1.5 times the bit rate works well and we have not found it worth exceeding 2 times.

Using both edges of the local clock (so called "double data rate" input) gives us two samples per clock cycle and thus we can use the simple rule-of-thumb that the local clock frequency should equal the SpaceWire bit rate.

If double-data-rate input is not supported we can use one sample per clock sample which has the advantage of simpler logic but a lower acceptable SpaceWire data rate.

It is also possible to implement more samples per clock cycle by using a multi-phase sampling scheme or a frequency multiplier from the local clock.

Table 1 shows a small selection of schemes that we have used, characterised by the ratio of sampling frequency to system clock frequency (Nf) and number of samples per clock cycle (Mp). A conservative ratio of sampling rate to data rate of >1.5:1 has been used in this table.

Scheme	Sample Rate	Data rate
1f1p	f	f/2
1f2p	2f	f
1f4p	4f	2f
3f2p	6f	4f

Table 1 Sampling Schemes

6. PERFORMANCE ON FPGA

Using far from the state-of-the-art FPGA's (Xilinx™ Virtex 2) we achieve a data rate of 500Mb/s with a 125MHz system clock.

We have recently ported designs to Actel™ devices and find the results most encouraging. Here are a few key findings:

- Using a 125 MHz system clock, sampled on both edges (250 Ms/s) for a nominal input data rate of 125 Mb/s we find that the maximum input data rate – after signal degradation through 10 metres of cable – is >190Mb/s. This is an ultra-conservative margin.
- Un-optimised, the Actel™ toolset prediction for the codec performance in RTAX, at 100 KRad dose, is >155MHz. We expect a version optimised for these devices to be capable of 200Mb/s. [The un-optimised design is already capable of exceeding 220Mb/s in commercial, AX, silicon.]
- The codec consumes just 460 cells – plus receive buffers. These buffers consume very little resource when implemented in block RAM but approximately double the size when implemented in registers.
- A complete 4-port routing switch, with FIFO's in registers (the resource-hungry option) requires 4300 cells (14% of an (RT)AX2000). It uses just *one* clock net for the whole design and *no* placement constraints.

7. CONCLUSION

Asynchronous SpaceWire designs can be implemented – many people, including ourselves, have done so – but there are many challenges to doing so on FPGA's. We have explained the worst of those challenges and suggested an easier alternative. Synchronous designs are easier to develop and implement in FPGA and our experience with them at the leading edge of performance in demanding applications gives confidence that this is a preferred route to success. It can also be usefully applied to ASIC design.

Benefits of the synchronous design include

- One clock net, regardless of the number of SpaceWire ports (Asynchronous designs require one clock net per port in addition to the system net);
- No cell placement is needed (asynchronous designs require careful placement to meet difficult timing relationships);
- Robustness due to the lack of difficult timing issues (placement is critical to asynchronous designs – they are fragile);
- Easy to mix designs with other IP – the tools have full freedom to intermix cells and routing (asynchronous design may have to reserve silicon area to ensure they meet their timing requirements).

8. REFERENCES

1. ECSS-E-50-12A (24 January 2003), SpaceWire, Links, Nodes, Routers and Networks
2. H. Johnson & M. Graham, *High Speed Digital Design*, Prentice Hall