The SpaceWire CODEC

Chris McClements, Steve Parkes
University of Dundee

Augustin Leon
European Space Agency
Introduction

• **SpaceWire system**
  – Nodes connected indirectly through routers or directly node to node

• **SpaceWire CODEC**
  – Encodes and decodes bit-stream on physical medium, SpaceWire cable.
  – Part of the data-link layer for SpaceWire systems to communicate
  – Implemented in RTL level VHDL code.
  – Compliant with ECSS-50-12A SpaceWire standard

• **Goals**
  – Technology independent
  – High speed operation, Low area footprint
  – Configurability to users requirements and target technology
Initialisation state machine
- Establishes a connection with other end of link
- Enables and disables transmitter and receiver
- Timeout counters to determine state changes

Transmitter
- Bit-stream serialisation using shift register
- Variable data rate and transmit clock selection
- Next character selection dependent on initialisation state and current requests
- Allowed to send data characters when FCTs are received
• Receiver
  – Performs receiver clock recovery
  – Decodes input bit-stream.
  – Controls receiver buffer credit operations.
  – Resynchronises received characters to receive buffer clock

• Internal Error Recovery
  – Error recovery is performed when a link error is detected.
  – Recovers the tx and rx data buffers due to link disconnection
    • Transmitter may be in the middle of sending a packet
      – Packet is flushed from transmitter buffer.
    • Receiver may have been receiving a data packet.
      – Received packet is truncated with an error end of packet
    • Any outstanding FCT characters are added back to space available in receiver buffer counter.
Serial Interface

- **SpaceWire uses data-strobe encoding**
  - Strobe changes when data retains value for one bit-period
  - Efficient receiver clock recovery using Data xor Strobe at the expense of one extra signal wire.

- **Transmitter serial interface can be configured for**
  - **Single data rate**
    - One bit of information shifted for each transmit clock period
    - Simplest implementation
  - **Double data rate**
    - Two bits of information encoded for each transmit clock period
    - Extra complexity as two bits of data must be multiplexed onto the output pin for each clock period
• Main Goal of the VHDL SpaceWire CODEC
  – Tailor the VHDL to suit the users target hardware and application.

• Target hardware comes under two main headings
  – ASIC devices
  – PLD/FPGA devices

• Main factors affecting SpaceWire CODEC design
  – Number of clock nets or clock routing resources
  – Most efficient data storage method
  – Size of receiver buffer
  – Speed requirements
  – Area requirements
Pipelining Configuration

- Pipelining defines the insertion of flip-flops to increase performance
  - Extra logic is required for the pipeline flip-flops, greater area footprint
  - One cycle of latency is added to the transmitter when transmitting a data character and to the receiver when receiving a data character

- Increase in transmit speed compensates for extra logic and latency

- Simple data-rate comparison on a Xilinx Virtex-e device with no constraints on the logic

<table>
<thead>
<tr>
<th>Type</th>
<th>Area footprint</th>
<th>Max transmit rate</th>
<th>Max input bit rate, Receive clock*2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FLIP-FLOPS</td>
<td>LUTS</td>
<td>SDR</td>
</tr>
<tr>
<td>Non-pipelined</td>
<td>257</td>
<td>352</td>
<td>93 mbits/s</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>186 mbits/s</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>285 mbits/s</td>
</tr>
<tr>
<td>Pipelined</td>
<td>267</td>
<td>354</td>
<td>141 mbits/s</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>282 mbits/s</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>345 mbits/s</td>
</tr>
</tbody>
</table>
The transmit clock frequency determines the maximum serial output bit-rate.
  - Maximise target hardware capabilities to achieve maximum data rate

**CODEC transmitter constraints:**
  - Transmit clock
    - System clock or independent transmit clock
      - Set dependent on number of clock sources and nets available
  - Variable data rate
    - External, Internal clock divider or internal clock enable
      - Set dependent on the users variable data rate generation requirements
  - Default 10MBits/s transmit rate
    - Internally generated using variable data rate generator or external crystal
      - Set dependent on the clock sources, system and transmit clock frequencies.
### All transmit Clock Configurations

<table>
<thead>
<tr>
<th>Symbolic name</th>
<th>Input clock</th>
<th>Default 10/5MHz</th>
<th>Variable data rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS_DEFAULT</td>
<td>System Clock</td>
<td>System Clock</td>
<td>System Clock</td>
</tr>
<tr>
<td>SYS_SLOWCLK</td>
<td>System Clock</td>
<td>External 10/5 MHz</td>
<td>System Clock</td>
</tr>
<tr>
<td>SYS_SLOWCLK_DIV</td>
<td>System Clock</td>
<td>External 10/5 MHz</td>
<td>Clock divider</td>
</tr>
<tr>
<td>SYS_DIV</td>
<td>System Clock</td>
<td>Clock divider</td>
<td>Clock divider</td>
</tr>
<tr>
<td>SYS_EN</td>
<td>System Clock</td>
<td>Clock enable</td>
<td>Clock enable</td>
</tr>
<tr>
<td>TXCLK_DEFAULT</td>
<td>Transmit Clock</td>
<td>Transmit Clock</td>
<td>Transmit Clock</td>
</tr>
<tr>
<td>TXCLK_SLOWCLK</td>
<td>Transmit Clock</td>
<td>External 10/5 MHz</td>
<td>Transmit Clock</td>
</tr>
<tr>
<td>TXCLK_SLOWCLK_DIV</td>
<td>Transmit Clock</td>
<td>External 10/5 MHz</td>
<td>Clock divider</td>
</tr>
<tr>
<td>TXCLK_DIV</td>
<td>Transmit Clock</td>
<td>Clock divider</td>
<td>Clock divider</td>
</tr>
<tr>
<td>TXCLK_EN</td>
<td>Transmit Clock</td>
<td>Clock enable</td>
<td>Clock enable</td>
</tr>
</tbody>
</table>
Transmit default configuration

- Transmit clock is input from external clock generator
Internal Divider With External 10Mbits/s

- Internally generated variable transmit clock
- D input of transmitter flip-flops is gated by clock enable signals
• Default bit rate for all configurations
  – (No external 10Mbits/s clock)

\[
10Mbits = \frac{TxMaxBitRate}{(CFG\_SLOWRATE\_TXCLK + 1)}
\]

• Variable bit rate for all configurations

\[
TxNormalBitRate = \frac{TxMaxBitRate}{(TXRATE + 1)}
\]
Receive Buffer Configuration

- Receiver buffer stores received data characters
  - User sets size of receive buffer $2^n$
    - 8 bytes to maximum user buffer size (e.g. 32 bytes, 1Kbytes, 1Mbytes, etc.)
    - User sets maximum outstanding data characters, $\geq 8$ and $\leq 56$

- Receiver credit count operations performed internally by CODEC:
  - Uses an internal FCT pointer which moves ahead of read and write pointers for outstanding data characters and reserved buffer space
  - Request transmitter to send one FCT when:
    - unreserved space in buffer, up to maximum buffer size, and
    - not currently requesting maximum outstanding data characters

- Credit error is detected when a data character is received when not requested.
  - FCT pointer is moved back to write pointer position (No outstanding data characters)
Receive buffer data resynchronisation

• Received data characters resynchronised to receive buffer clock domain
  – Double flip-flop resynchronisation control signals used
  – At maximum bit rates the time taken to resynchronise data characters is greater than the time taken to receive
  – Internal 4 byte buffer used to store received data characters until the data is written to the receive buffer

• Design aims/goals
  – No internal data storage!

• Use most common storage methods
  – Latches for ASIC cell based devices
  – Flip-flops/RAM inference for FPGA based devices
Latch Implementation

• Three stage latch cell write is performed
  – Receiver clock cycle 1: Setup data, address
  – Receiver clock cycle 2: Perform write strobe
  – Receiver clock cycle 3: Hold data, address

• Reception of empty packet (double EOP/EEP) violates three clock cycle write rule.
  – CODEC can be configured to ignore empty packets

• Reception of <data><eop><data> violates three clock cycle write rule (<eop> = 2 receive clock cycles)
  – CODEC receiver delays EOP/EEP by one clock cycle
Host Interface

- Link control functions implemented as in SpaceWire standard
  - Auto-start link
    - Wait for other end of link to make connection
  - Start link
    - Attempt to make a connection with the other end of the link
  - Disable link and force disconnection

- Status outputs
  - Status outputs synchronous with system clock
  - Provide a snapshot of the CODEC
### Status Outputs

<table>
<thead>
<tr>
<th>Signal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATUS(0)</td>
<td>Disconnect error.</td>
</tr>
<tr>
<td>STATUS(1)</td>
<td>Parity error.</td>
</tr>
<tr>
<td>STATUS(2)</td>
<td>Escape error.</td>
</tr>
<tr>
<td>STATUS(3)</td>
<td>Receiver credit error.</td>
</tr>
<tr>
<td>STATUS(4)</td>
<td>Transmitter credit error.</td>
</tr>
<tr>
<td>STATUS(7:5)</td>
<td>Interface state encoded into three bits (6 states)</td>
</tr>
<tr>
<td>STATUS(8)</td>
<td>Interface state machine is in the <em>Run</em> state.</td>
</tr>
<tr>
<td>STATUS(9)</td>
<td>Receiver got NULL. Remains asserted after first NULL.</td>
</tr>
<tr>
<td>STATUS(10)</td>
<td>Receiver got FCT. Remains asserted after first FCT</td>
</tr>
<tr>
<td>STATUS(11)</td>
<td>Receiver got N-chars. Remains asserted after first N-Char</td>
</tr>
<tr>
<td>STATUS(12)</td>
<td>Receiver got Timecodes. Remains asserted after first Timecode</td>
</tr>
<tr>
<td>STATUS(13)</td>
<td>Transmitter has credit to send one more data character</td>
</tr>
<tr>
<td>STATUS(14)</td>
<td>N-char sequence error (N-char received before link state is Run)</td>
</tr>
<tr>
<td>STATUS(15)</td>
<td>Timecode sequence error (Timecode received before link state is Run)</td>
</tr>
</tbody>
</table>
Transmit host interface

- FIFO interface to host system
  - Transmit FIFO is implemented externally using technology specific data storage

- Synchronous empty and read signals
  - Synchronous to TXBITCLK which may be asynchronous to host clock

- Good implementation uses FIFO with independent read and write clocks
• FIFO interface to host system
  – Data buffer is implemented externally using users technology specific storage method
  – CODEC implements synchronous FIFO interface
• Parallel data output from the CODEC written to the receive buffer using:
  – system clock or independent read buffer clock

• Receive buffer clock can be slower to save power, for example.
  – 200Mbits/s input bit-rate
  – Stream of one byte packets causes, on average, one data character available every 35ns
    • Resynchronisation buffer smoothes data flow
  – Minimum receive buffer clock frequency to support data rate is this case is $1/35\text{ns} = \sim 28.571 \text{ MHz}$

• For the same setup
  – Stream of four byte packets causes, on average, one data character available to receive buffer every 44 ns.
  – Minimum receive buffer clock frequency to support input data rate in this case is $1/44\text{ns} = \sim 22.727\text{ns}$
• From previous slide it can be seen that the minimum size of packets and the input bit-rate determine the minimum receive buffer clock
• Minimum read clock =

\[
\frac{1}{\left( \frac{(\text{MinNumData} \times T_{data}) + T_{eop}}{\text{MinNumData} + 1} \right)}
\]

• In terms of maximum input bit rate =

\[
\frac{1}{\left( \frac{T_{rxbuf \_ clk} \times \text{NumPktNchars}}{\text{NumPktBits}} \right)}
\]
Performance and latency

• Expected performance
  – 200Mbits/s Router ASIC device
  – 200Mbits/s University of Dundee components
  – Placed and routed at 400Mbits/s in a Virtex 2 -6 speed grade device

• Latency
  – Minimum transmit latency is 2 transmit clock cycles
  – Minimum receive latency is
    • 2 receive clock cycles +
    • 2 receive buffer clock cycles
Conclusions

- SpaceWire standard ECSS-50-12A compliant SpaceWire CODEC
- Features
  - High speed operation
  - Configurable to meet users requirements
  - Ease host integration
  - Technology independent