Simplifying the management of multiple ADC data interfaces using PolarFire: a case study

Author

Roberto Capezzali
Thales Alenia Space
Specialist in VLSI Design
for Radar Applications
roberto.capezzali@thalesaleniaspace.com

1. ABSTRACT

A common scenario in hardware design for complex space systems is the need for simplification, driven by require+ments such as weight, volume, power consumption, and cost. This often leads to resource limitations that pose significant challenges for designers.
Modern FPGAs can offer resources that help to address these challenges through unconventional approaches. It is crucial for designers to have a comprehensive understan+ding of the toolbox that FPGA technology provides to explo+re alternative solutions when traditional methods are not feasible.
In this case study, **a large number of ADCs needed to be properly connected to a central processing FPGA imple-mented on a PolarFire device RTPF500T-1CG1509** (trials were also conducted on RT300).

Due to hardware design constraints, **the number of signals from the ADCs to the FPGA exceeded the avai-lable backplane connections**. Additionally, challenges arose in meeting the mutual timing constraints of the rou+table interface signals.
The problem was resolved by combining system-level control with PolarFire features, leading to substantial ex+ternal hardware simplification. The resulting solution meets all system requirements and has proven to be more robust than the originally intended architecture, both in minimi+zing points of failure and in its resilience to long-term varia+tions, such as aging and signal drift.

2. PROBLEM DESCRIPTION

2.1 Context

In an ever-evolving technological scenario, the requirement for data throughput to be exchanged among electronic components is continuously growing, and such growth is usually made feasible by new interface protocols able to manage higher and higher data rates.
For instance, in the DSP field, the development of the JESD protocol in its various versions has optimized hardware connections between ADCs, DACs, and FPGAs.
However, **sometimes architecture evolution is based on increased resources parallelism**, maintaining classical connection interfaces while increasing the number of con+nections themselves. This increase leads to a proliferation of hardware connections among components within the architecture, reaching numbers that pose a routing chal+lenge.
In the present case study, based on parameters of genuine interest for architectures under development, **an FPGA is tasked with processing data from a large number of ADCs** (48-72 as a range, distributed in groups of 8 on va+rious peripheral boards). Each interface between ADCs and FPGA is a typical co-directional interface consisting of:

- a Data Clock line (D_clk)
- a data line carrying the sampled signal at N serialized bits (Data)
- a Frame Clock line (F_clk) determining the word boun+dary of the data.

Frequency range of interest for D_clk are in the order of 200 to 400 MHz, with Data running at double data rate.
An interface description is provided in ***FIGURE 1***. As a fur+ther hardware complication, lines are typically differential, requiring each ADC to have 6 physical connections with the FPGA.
All ADCs receive the same sampling clock, generating both F_clk and D_clk, thus all F_clk run at the same frequency, as well as all D_clk. ***FIGURE 2*** describes a simplified block dia+gram of the system.

2.2 Hardware design constraints

Several issues arose during the hardware design phase of the architecture demonstrator:

- **The total number of connections required between ADCs and FPGA** (min 2x3x48=288, max 2x3x72=432) **exceeded the available connections** between periphe+ral boards and the central processing unit. Thus, it was necessary to find ways to decrease this number.
- There was obviously a **need for a phase relation among D_clock, F_clock, and Data** to ensure correct data sam+pling and accurate word reconstruction representing the N-bit sample by the FPGA.
- **It was essential that data from various interfaces, once deserialized inside the FPGA, be temporally ali-gned**. It was not acceptable to have random delays between reconstructed signals from different lines (Note: a deterministic delay, removable during processing, is acceptable).

2.3 Hypothetical simplifications and impact on hardware design

To make the hardware design feasible, the following simpli+fications were necessary:

- **Only one D_clock line for each peripheral board was connected to the FPGA.**
- **No F_Clock line was connected to the FPGA.**

This simplification allows for a **62.5% reduction in con-nections**, (min from 2x3x48=288 to min 2x1x48+2x(48/8)=108, max from 2x3x72=432 to 2x1x72+2x(72/8)=162)).
The focus of this study was to find an implementation solu+tion to meet the constraints mentioned in section 3 given the indicated simplifications.

3. PROBLEM SOLUTION

3.1 Bit alignment

The first problem to address was **the correct resampling of the serial data**. Without the availability of each line's ac+companying D_clock, a dynamic data\clock phase adjust+ment system was implemented **using the PF_IOD_GENERI-C_RX macro and developing a controller for fine-tuning the sampling point**.

3.1.1 D_clock mux tree

It was decided to operate the entire RX section with a **single clock**, chosen from the 6/8 available inputs. To this purpose, a clock mux tree was set up as shown in ***FIGURE 3***. This choice was based on simplifying data processing within the FPGA by using a single clock domain. Additionally, **making the processing clock selectable offers redundancy of clock sources to the system**, drastically reducing the chance of failure and providing an extra feature in terms of design safety.
However, the choice of a single clock necessitates **modifica-tions to the PolarFire RX macro**, which, in Microchip's pro+posed implementation, operates with a clock coming from dedicated pins and not from the FPGA fabric side. Additio+nally, the clock returned by the macro to the fabric side, along with the deserialized data, is generated within each of the FPGA IO banks. In order to use a single clock among lines from different banks, a modification of the PolarFire soft macro has been implemented.

3.1.2 PF_IOD_ GENERIC_RX

As shown in ***FIGURE 4***, the RX section of this macro is equip+ped with an array of 256 analog delay lines, each capable of applying roughly 25ps delay to the incoming signal. As shown in ***FIGURE 5***, the macro receives the data serial line from the pad and samples it with an interface clock working in DDR. **EYE_MONITOR_EARLY and EYE_MONITOR_LATE alarms are used to indicate when the clocking point is too close to data edges**. An external logic has been devel+ped for opportunely **adding or subtracting delay lines in order to center the correct sampling point**. The macro can signal, through EARLY and LATE controls, if the sampling point of the data downstream of the configured delays ap+proaches the data transition point (and the activation di+stance is configurable at runtime). Finally, the sampled data is deserialized with a configurable parallelism M and made available on the FPGA fabric side.

3.1.3 Bit alignment controller

When bit alignment is requested, the developed controller follows **this procedure**:
a) **Initialize** the macro's analog **delays to zero**.
b) **Start sampling incoming data, recording whether the violation signals (early or late) activate at the current delay**. The recording occurs by setting 1 (violation) or 0 (non-violation) on an array of 256 registers (one for each cu+mulative delay element).
c) **Increment** the analog delay by one.
d) **Repeat** points b) and c) until the last delay.
e) Analyze the violation array to **identify the maximum interval of absence of violations** (the longest sequence of zeros).
f) Configure the **sampling point at the center of this interval**.
At the procedure's end, the user can choose to disable or keep tracking the optimal sampling point active, based on parameters dependent on usage type.
FIGURE 6a and ***FIGURE 6b*** explain the described procedure.

3.2 Sample alignment

After obtaining bit alignment, the next essential operation is properly **reconstructing the sample boundary**. Typically related to the use of F_clock in the normal functioning of the ADC interface, this operation is not possible without F_clock and **requires a procedure involving both ADCs and FPGA**.

3.2.1 ADC pattern mode

The used ADC (TI ADC368), like many commercial others, in+cludes various debug modes, one of which, pattern mode, allows sending a configurable word on the Data line. The ADC formats the word under F_clock, letting the receiver synchronize with this word to deduce the F_clock position and correctly deserialize the data. **The pattern mode is set up during the alignment procedure**.
FIGURE 7 depicts one of the possible pattern modes for TI ADC368.

3.2.2 Pattern aligner

A **pattern aligner** was thus developed to determine the word boundaries of input data. It works similarly to classic parallel aligners (e.g., the ones used in SDH) utilizing the pa+rallel data from the macro and providing the reconstructed parallel sample along with an enable on the FPGA fabric side. This pattern aligner was designed with input parallelism of 4 and output of 16 or input parallelism of 7 and output of 14, with an enable valid every 4 or 2 phases, respectively, to match different ADC modes.

3.3 Datastream alignment

The final system constraint to match is the **temporal align-ment of samples from different ADCs** once reconstructed within the FPGA, meaning two samples taken simultaneou+sly at time T from two different ADCs must be aligned inside the FPGA. (Note: a deterministic delay, known during system calibration and repeatable with each system startup post-a+lignment procedure, is acceptable as well).
This requires two further procedures.

3.3.1 Alignment of all ADC sampling points

This is essential for obvious reasons, and is achieved by **en-suring phase-aligned arrival of the sampling clock at all ADCs**. It involves hardware tuning by compensating clock track lengths from distributors to ADCs during design, fol+lowed by calibrating analog delays programmable at the clock distributor outputs. During this phase, the FPGA can be used to measure differential delays in data paths.

3.3.2 Controlled start of datastream alignments for all RX interfaces

With assured simultaneous arrival of data streams at va+rious RX interfaces of the FPGA, **simultaneous sampling of the incoming lines must be ensured**. The PF_IOD_GENERI+C_RX macro provides the first M bits (4 or 7 in different ap+plications) starting from the first sample acquired by the IO macro after the reset release.
Unfortunately, the involved frequencies do not guarantee that the resynchronized resets of various macros will have simultaneous effect.
Thus, the **following procedure** has been introduced:
a) **Turn off the common reference clock** for all interfaces.
b) Assert the **asynchronous (non-resynchronized) reset**.
c) **Restart the reference clock**.
This way, zero or minimal phase differences (which can be absorbed by the pattern aligner) will ensure bit-aligned outputs from the PF_IOD_GENERIC_RX macro's parallel in+terfaces. ***FIGURE 8a*** explains how a wrong reset release impact the alignment behavior, while ***FIGURE 8b*** depicts the correct procedure.

4. CONCLUSIONS

The implemented solution has **guaranteed the system fea-sibility** and brought **additional positive effects**:
a) **Hardware connections** are significantly **reduced** (> 60%).
b) With DSP relying on a single clock received by the FPGA from 6 to 9 interfaces in various configurations, several **dan-gerous single failure points are eliminated**.
c) The data sampling and tracking mechanism simplifies hardware design, with the FPGA **not needing to guarantee strict phase constraints compliance** between data and clock.
d) As **this mechanism is dynamic**, it can remain active or can be repeated during operational phases, **absorbing medium and long-term variations** like wandering and aging.
The experience suggested **other fields of interest** in which unconventional use of last generation FPGAs' resources can make the difference. Just for instance, a kind of minimal in+terconnections control interface based on the above descri+bed implementation is under study for future applications.

Figure 1

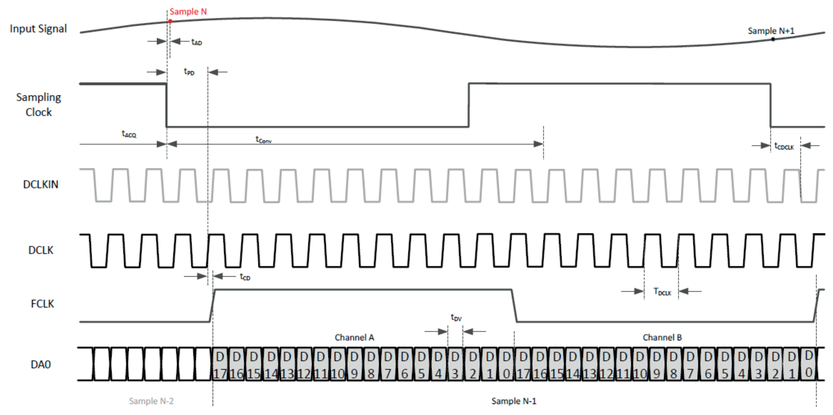

Figure 2
System block diagram

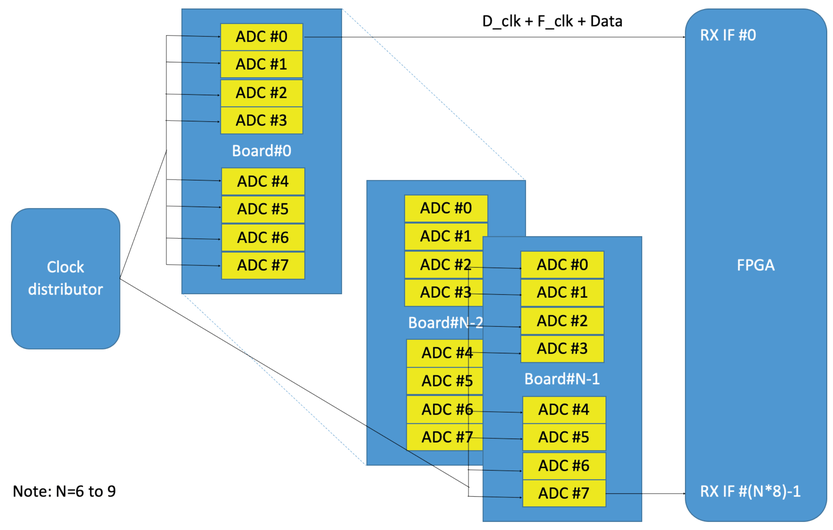

Figure 3
Clock tree

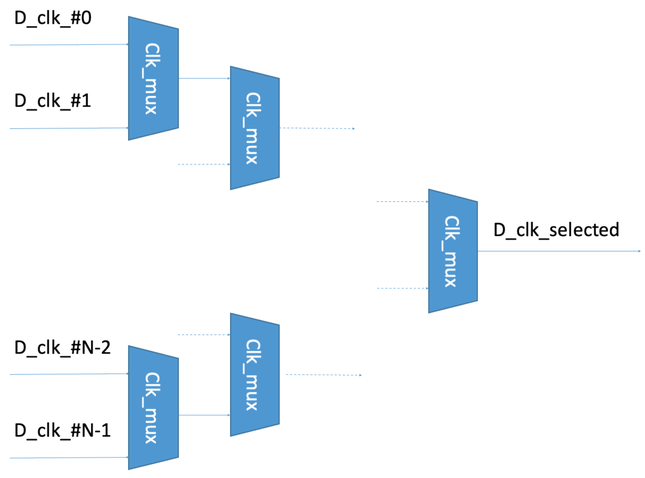

Figure 4

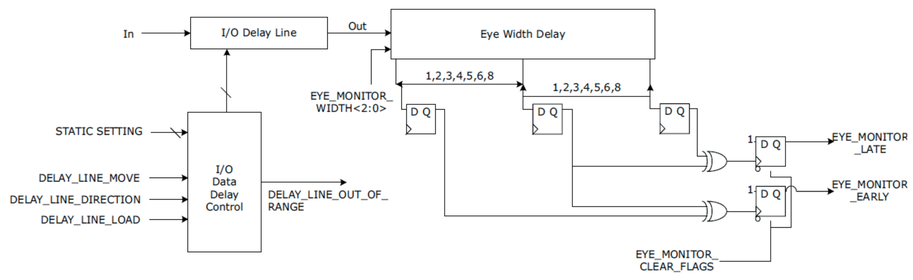

Figure 5

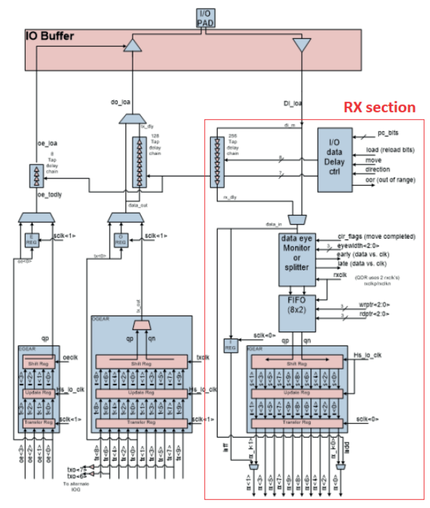

Figure 6a
Bit alignment controller procedure
a) Eye detection example

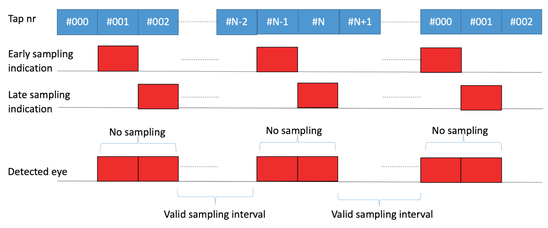

Figure 6b
Bit alignment controller procedure
a) Optimal sampling point choise

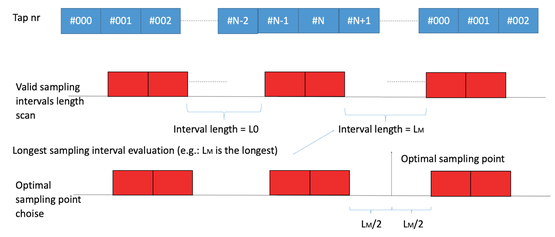

Figure 7
Pattern aligner

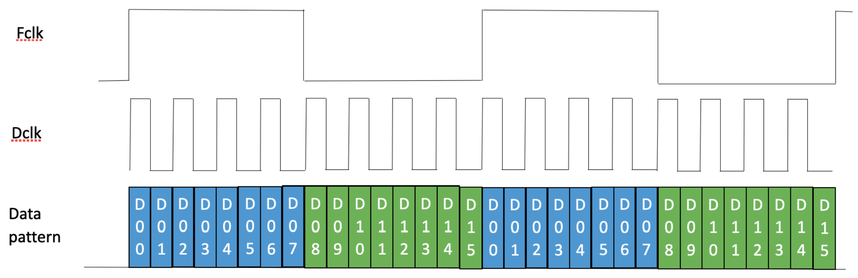

Figure 8a
Unsynchronized RX IOD deserializer start

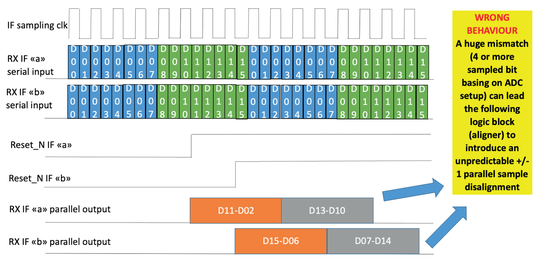

Figure 8b
Stopped clock RX IOD deserializer start