Exchanging data between the CPU and the FPGA

PN128 | Posted on April 16, 2026 | Updated on April 17, 2026

François LEDENT
Development Engineer
imperix

Table of Contents

On imperix controllers, the CPU exchanges data with the FPGA via the **SBIO bus**. This memory-mapped bus allows the CPU user app to read and write FPGA registers via the SBI and SBO blocks in Simulink and PLECS.

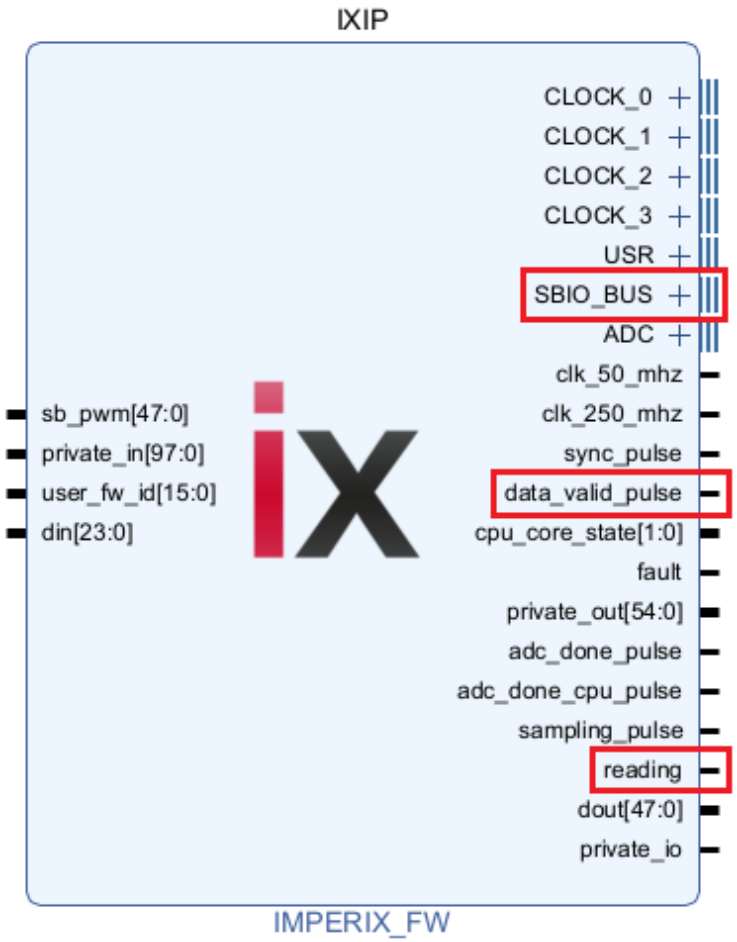
On the FPGA side, helper modules are provided to abstract the bus complexity and expose registers instead. An AXI4-Stream interface is also available, designed to facilitate the implementation of FPGA-based control algorithms.

Note on FPGA development for imperix controllers
Customizing the FPGA firmware involves instantiating the imperix firmware IP within Xilinx Vivado to edit the surrounding programmable logic, known as the sandbox. For step-by-step instructions on creating the required FPGA sandbox template, refer to the getting started guide.

SBIO bus

The **SBIO_BUS** (SandBox IO bus) is a 16-bit memory-mapped bus allowing the CPU to address up to 1024 SBI or SBO registers in the FPGA. The **`reading`** and **`data_valid_pulse`** signals serves to synchronize the user logic to the reading and writing cycles.

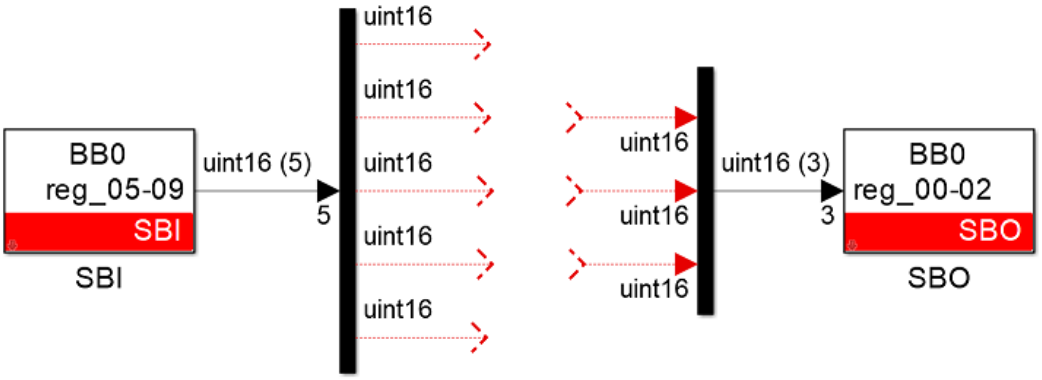
The bus operation details are documented in the imperix firmware IP product guide. However, direct interaction with these signals is rarely required due to the availability of the **SBIO helper modules** described in the following section.



SBIO_BUS and associated timing signals

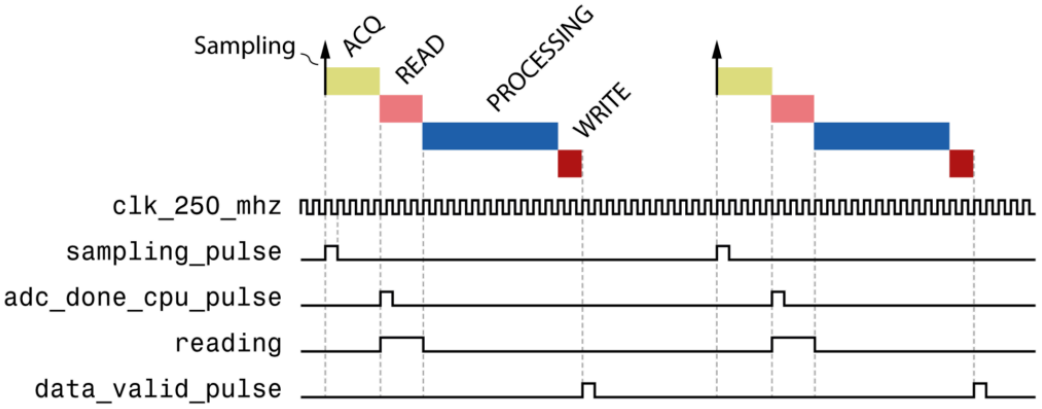
The user interacts with the bus using the following blocks:

- SBI blocks to **read** FPGA registers **before** the execution of the CPU task.
- SBO blocks to **write** FPGA registers **after** the execution of the CPU task.



The image below illustrates the CPU execution phases and related signals, which are further described in the [imperix firmware IP product guide].

- The **read** phase is indicated by the assertion of the `reading` signal. During that time, **SBI** registers are read and cached in the CPU buffer, making them available to the user application.
- At the end of the processing phase, **SBO** values are written to the FPGA. The end of the **write** phase is indicated by the assertion of the `data_valid_pulse` signal.



Execution timing signals

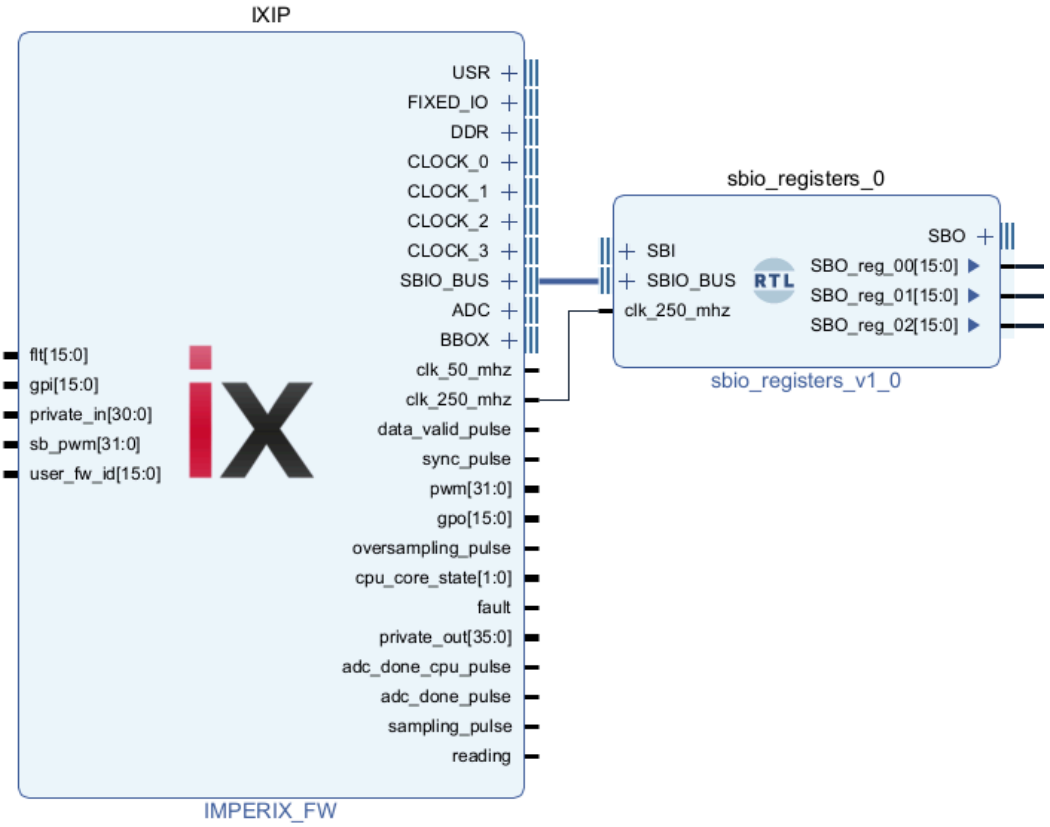
When set as *configuration registers*, the value of the SBO registers is only written once at the code start-up. More information is available in the [SBO] documentation page.

SBIO helper modules

The following SBIO helper modules are provided in the [imperix source files] to abstract the SBIO bus complexity.

SBIO registers

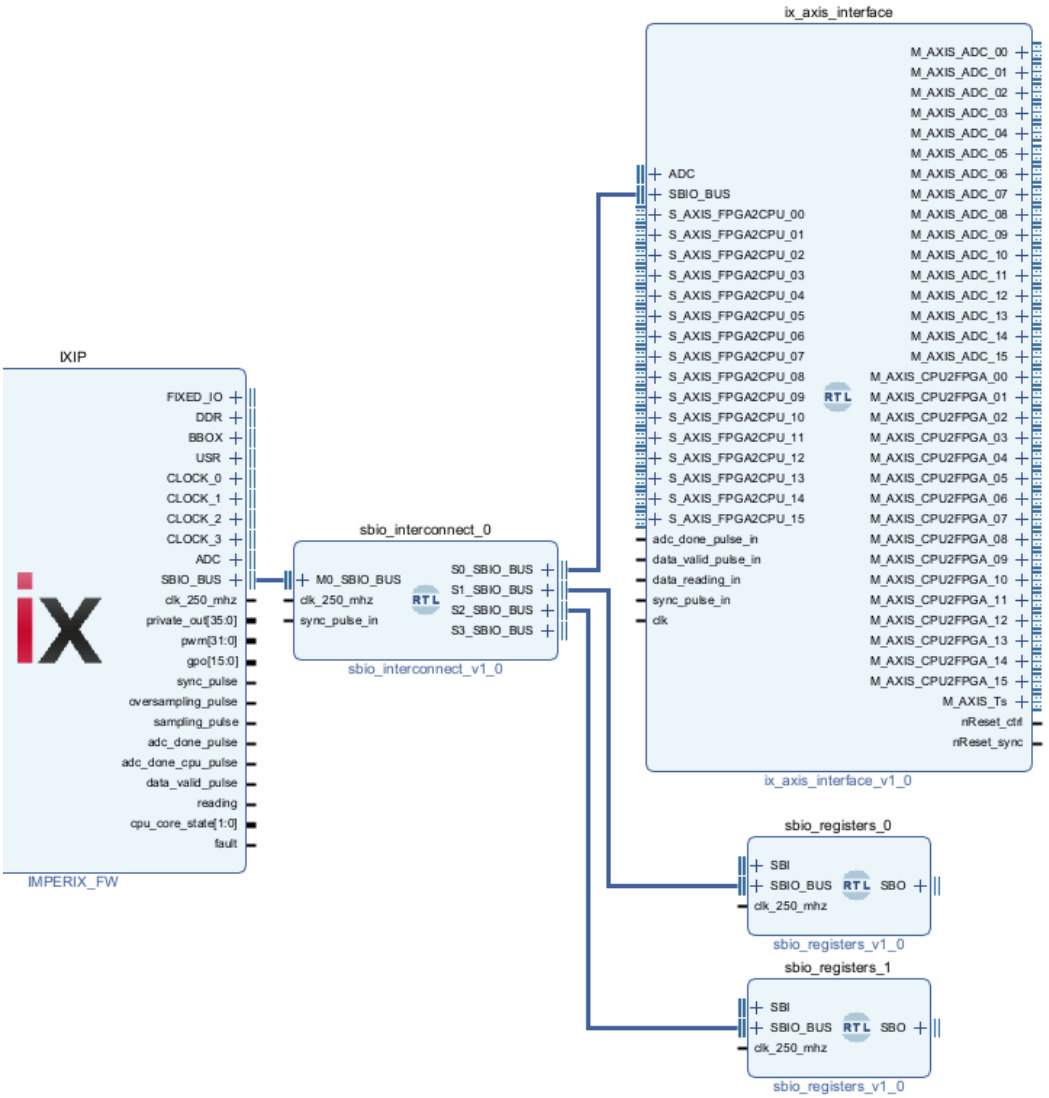
The **`sbio_registers`** and **`sbio_256_registers`** modules provide 64, respectively 256, addressable 16-bit registers accessible from the CPU using SBI and SBO blocks.



`sbio_registers` module

SBIO interconnect

The **`sbio_interconnect`** divides the full range of 1024 addresses of the SBIO bus into four ranges of 256 addresses each, allowing to connect multiple SBIO modules.

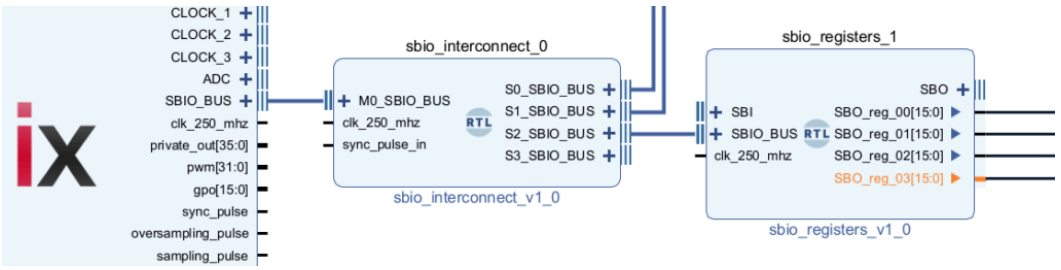


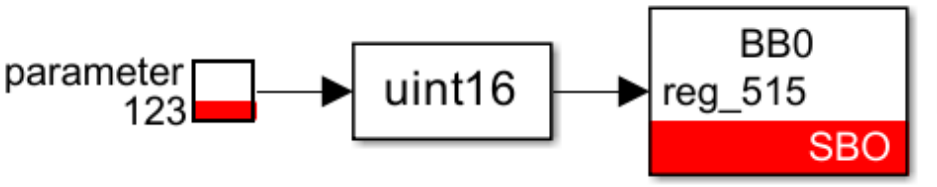
`sbio_interconnect` used to connect multiple SBIO modules to the SBIO_BUS

The address mapping of the SBIO interconnect is shown below, it divides the SBIO addressable range in 4 smaller areas.

`sbio_interconnect` memory mapping

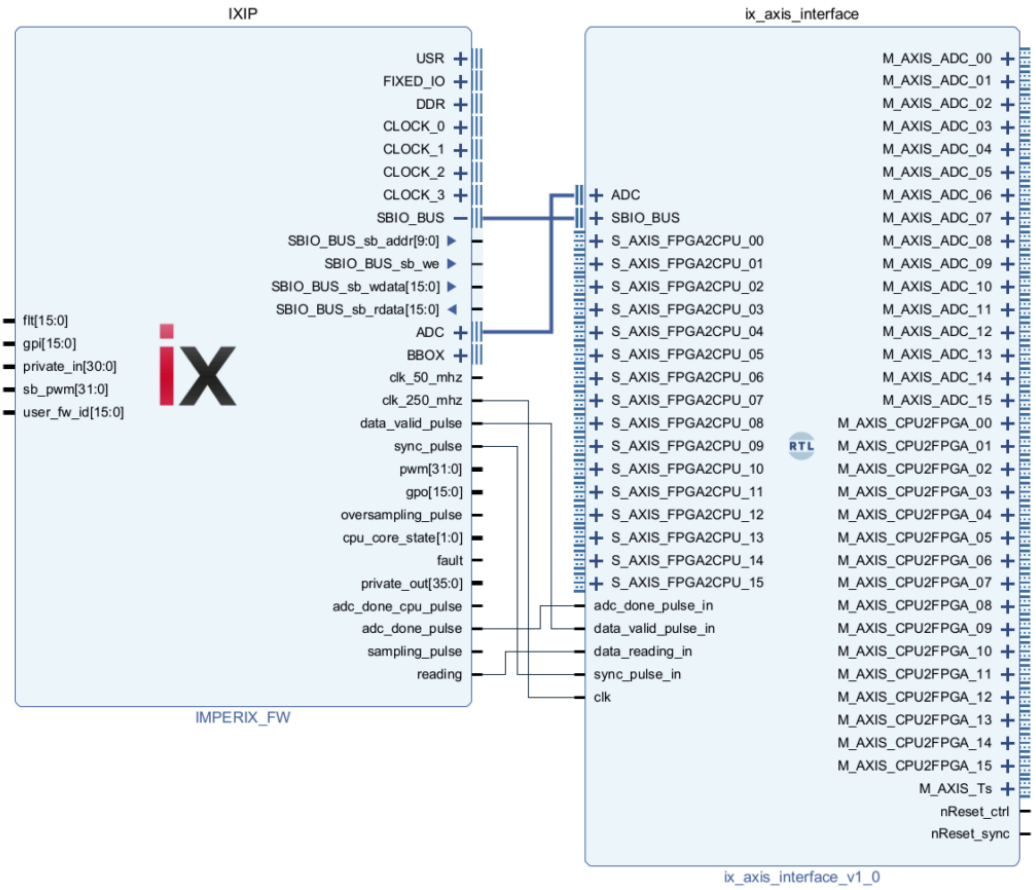
As an example, writing to **SBO_reg_03** of an **sbio_register** block connected to **S2_SBIO_BUS** requires an SBO block addressed to register 512+3=515.





AXI4-Stream interface

The AXI4-Stream interface provided with the sandbox template allows exchanging 32-bit data between the CPU and FPGA using the **M_AXIS_CPU2FPGA** and **S_AXIS_FPGA2CPU** interfaces.

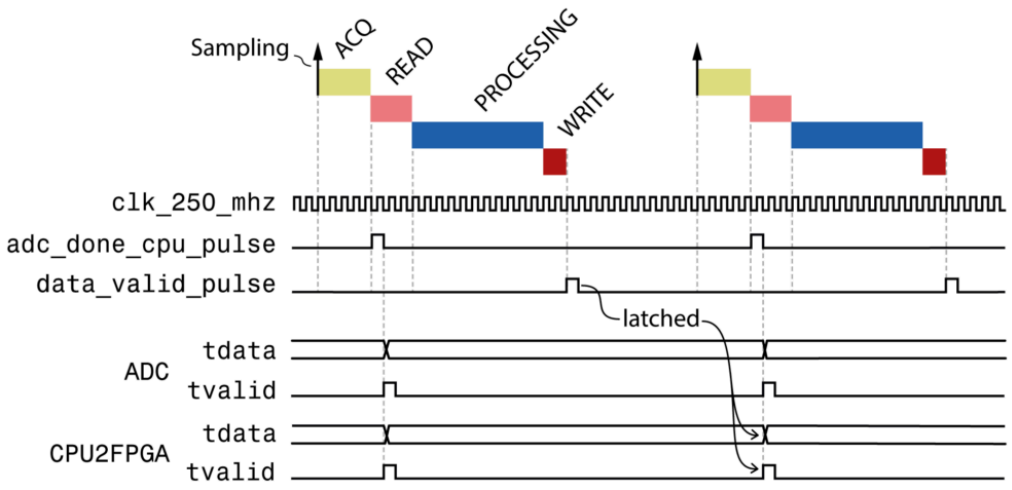


The AXI4-Stream interface was made specifically to facilitate implementing **control algorithm in FPGA** , such as the high-speed current control described in TN147. The interface is designed for algorithms that share the following characteristics:

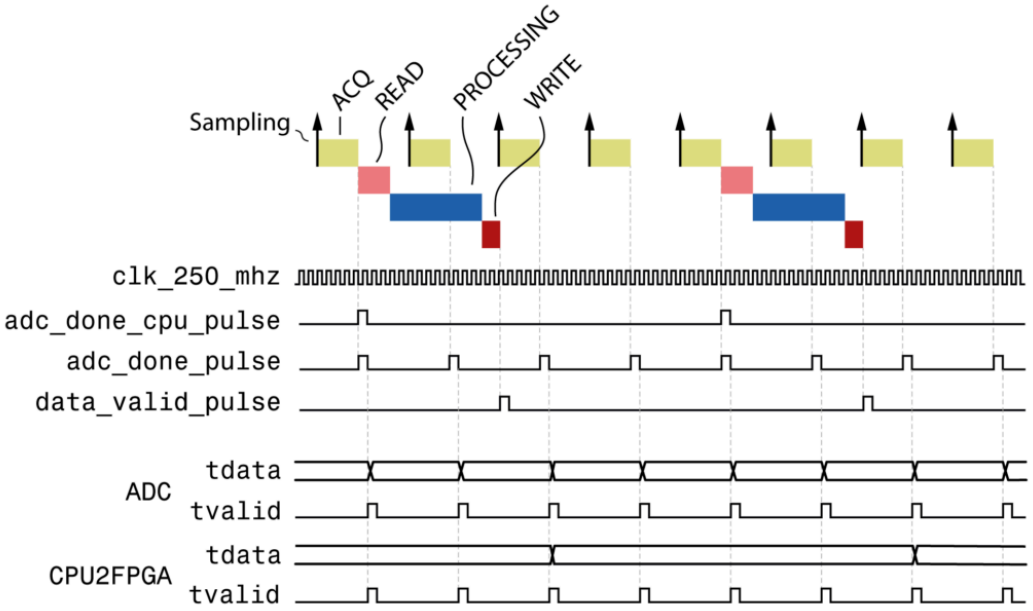
- The control algorithm is **fully implemented in the FPGA**.
- The control algorithm is executed at the **ADC sampling rate (SCLK rate)**, which may be faster than the CPU rate (postscaler > 1).
- The control algorithm operates with **32-bit floating-point** data.
- The CPU model is used to **writes slow-varying signals** such as configuration parameters (Kp, Ki, etc.) and slow-varying references, or serves for **monitoring/debugging**.

To facilitate the design of such algorithm, the following design choices were made:

- The M_AXIS_ADC and M_AXIS_CPU2FPGA interfaces output data **at the same time**, synchronized to the `adc_done_pulse`. The consequence is that **CPU2FPGA signals are delayed** to the end of the next acquisition phase.



- The M_AXIS_ADC and M_AXIS_CPU2FPGA interfaces output data **at the same rate.** That means that when the CPU execution rate is slower than the sampling clock (postscaler > 1) the same CPU2FPGA values are **outputted multiple times**.



Address mapping of the AXI4-Stream interface

The mapping between the 16-bit SBIO registers and the 32-bit AXI4-Stream interfaces is the following:

bit number	31 … 16	15 … 0
CPU2FPGA_00	SBO_01	SBO_00
CPU2FPGA_01	SBO_03	SBO_02
CPU2FPGA_02	SBO_05	SBO_04
CPU2FPGA_03	SBO_07	SBO_06
CPU2FPGA_04	SBO_09	SBO_08
CPU2FPGA_05	SBO_11	SBO_10
CPU2FPGA_06	SBO_13	SBO_12
CPU2FPGA_07	SBO_15	SBO_14
CPU2FPGA_08	SBO_17	SBO_16
CPU2FPGA_09	SBO_19	SBO_18
CPU2FPGA_10	SBO_21	SBO_20
CPU2FPGA_11	SBO_23	SBO_22
CPU2FPGA_12	SBO_25	SBO_24
CPU2FPGA_13	SBO_27	SBO_26
CPU2FPGA_14	SBO_29	SBO_28
CPU2FPGA_15	SBO_31	SBO_30

bit number	31 … 16	15 … 0
FPGA2CPU_00	SBI_01	SBI_00
FPGA2CPU_01	SBI_03	SBI_02
FPGA2CPU_02	SBI_05	SBI_04
FPGA2CPU_03	SBI_07	SBI_06
FPGA2CPU_04	SBI_09	SBI_08
FPGA2CPU_05	SBI_11	SBI_10
FPGA2CPU_06	SBI_13	SBI_12
FPGA2CPU_07	SBI_15	SBI_14
FPGA2CPU_08	SBI_17	SBI_16
FPGA2CPU_09	SBI_19	SBI_18
FPGA2CPU_10	SBI_21	SBI_20
FPGA2CPU_11	SBI_23	SBI_22
FPGA2CPU_12	SBI_25	SBI_24
FPGA2CPU_13	SBI_27	SBI_26
FPGA2CPU_14	SBI_29	SBI_28
FPGA2CPU_15	SBI_31	SBI_30

These registers are typically used to exchange floating-point values with the CPU. Ready-to-use helper functions for Simulink and PLECS are provided in the next section.

Exchanging floats signals

Algorithms typically operate using 32-bit floating-point values. The Simulink and PLECS templates provided below provide ready-to-use scripts to concatenate or decompose the 16-bit values and interpret the result as a floating-point variable.

Simulink model

Download **float_exchange_template.slx**

In Simulink, the following MATLAB functions are used.

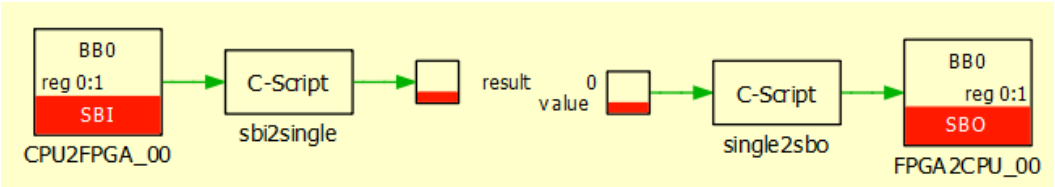
sbi2single

```
function y = sbi2single(u1,u2)
  y = single(0); % fix simulink bug: force compiled size of output
  y = typecast([uint16(u1) uint16(u2)], 'single');Code language: Matlab (matlab)
```

single2sbo

```
function [y1,y2] = single2sbo(u)
  temp = typecast(single(u),'uint16');
  y1 = temp(1);
  y2 = temp(2);Code language: Matlab (matlab)
```

PLECS model



Download **float_exchange_template.plecs**

The PLECS C-Script function are provided below.

sbi2single

```
float y1 = InputSignal(0,0);
float y2 = InputSignal(0,1);

union { unsigned int i; float f; } conv;
conv.i = ((unsigned short)y2 << 16) | (unsigned short)y1;

OutputSignal(0,0) = conv.f;Code language: C++ (cpp)
```

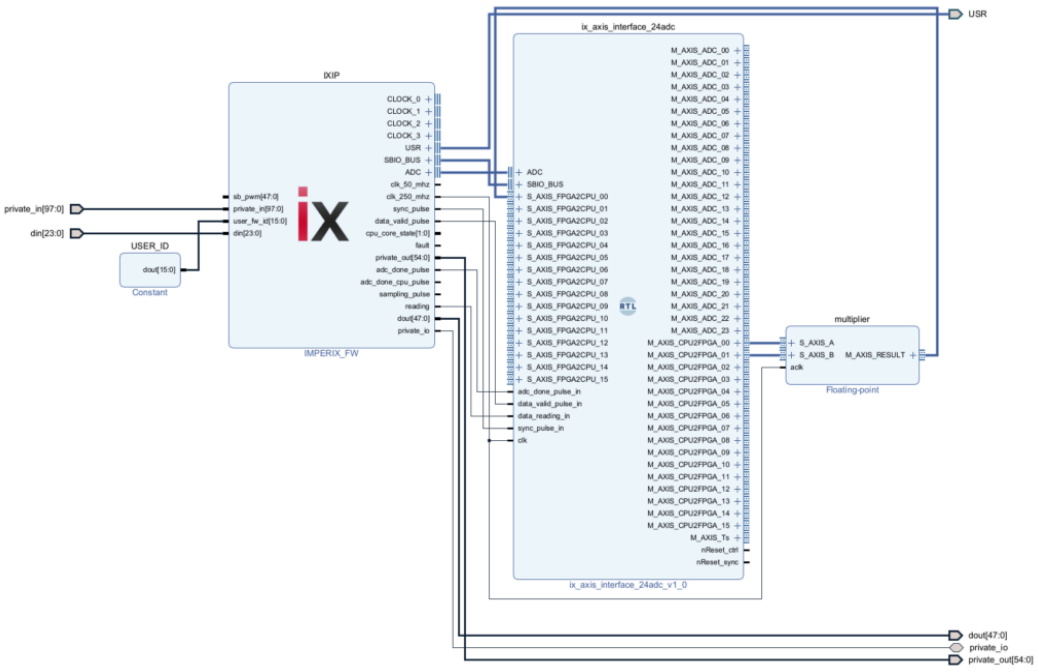
single2sbo

```
union { unsigned int i; float f; } conv;

conv.f = InputSignal(0,0);

OutputSignal(0,0) = (unsigned short)(conv.i & 0xFFFF);
OutputSignal(0,1) = (unsigned short)(conv.i >> 16);Code language: C++ (cpp)
```

Example using the AXI4-Stream interface

This step-by-step example implements the FPGA design shown below which:

- Receives two floating-point values from the user application (**`CPU2FPGA_00`** and **`CPU2FPGA_01`**).
- Multiplies the two values together using a **Xilinx Floating-Point IP**.
- Sends the result back to the user application running in the CPU (**`FPGA2CPU_00`**).



Two values are multiplied in the FPGA and sent back to the CPU

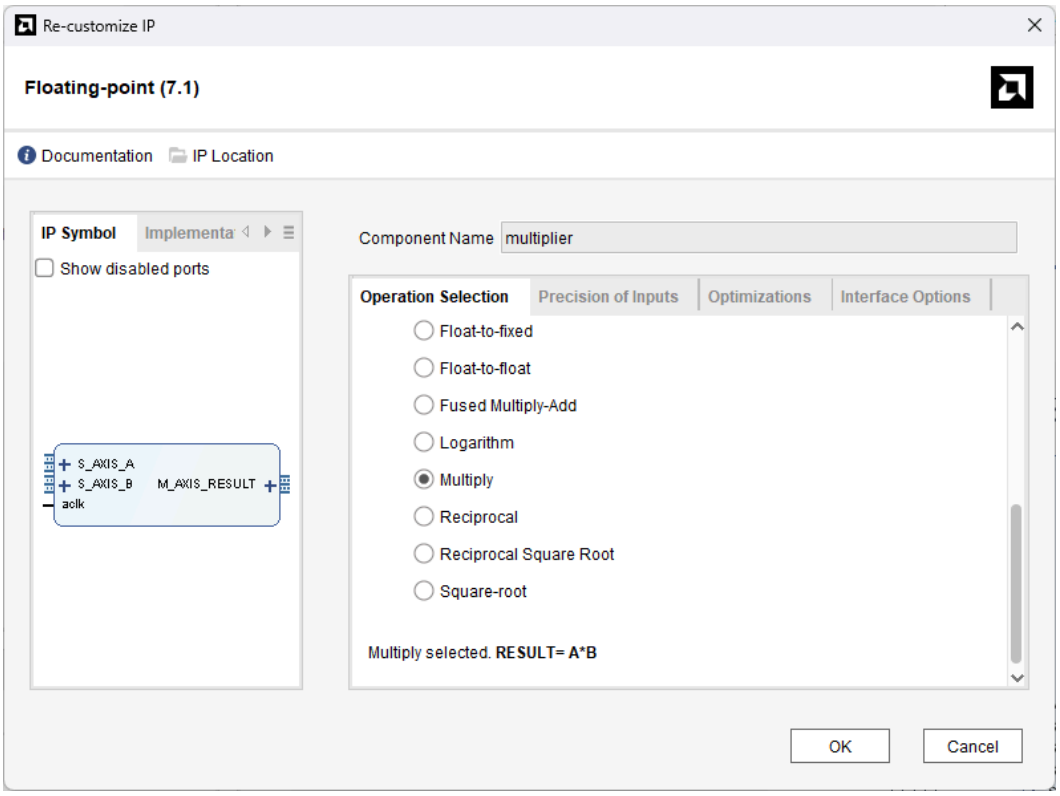
Start from template

Create the Vivado sandbox template following the “Creating the Vivado sandbox template” procedure detailed in the getting started.

Add a multiplier

To add an AXI4-Stream IP that multiplies two floating point value:

- Right-click somewhere in the block design, select **Add IP** and search for **Floating-point**. Press Enter.
- In the Block Properties panel, rename the block to **multiplier**.
- Double-click on the IP to open the configuration panel. In the **Operation Selection** tab, select **Multiply** for the operation. Other parameters can be let in their default configuration.

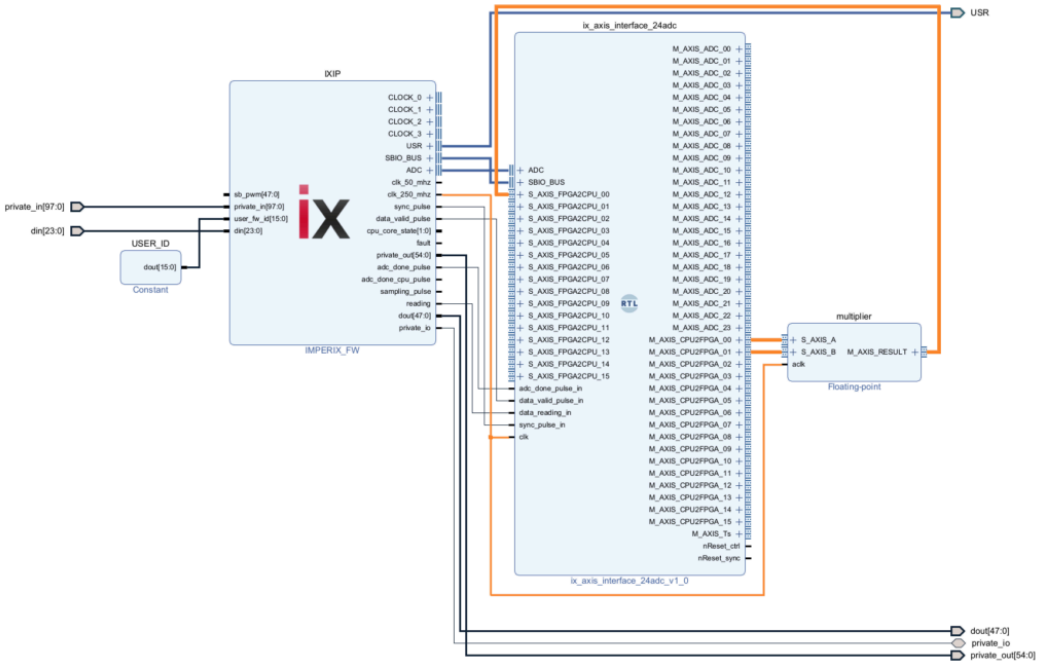


Connect the multiplier

The design can be finalized by applying the following connections between the AXI4-Stream interface and the multiplier:

- **M_AXIS_CPU2FPGA_00** to **S_AXIS_A**
- **M_AXIS_CPU2FPGA_01** to **S_AXIS_B**
- **M_AXIS_RESULT** to **S_AXIS_FPGA2CPU_00**

The **aclk** signal of the multiplier must be connected to the **clk_250_mhz** clock.



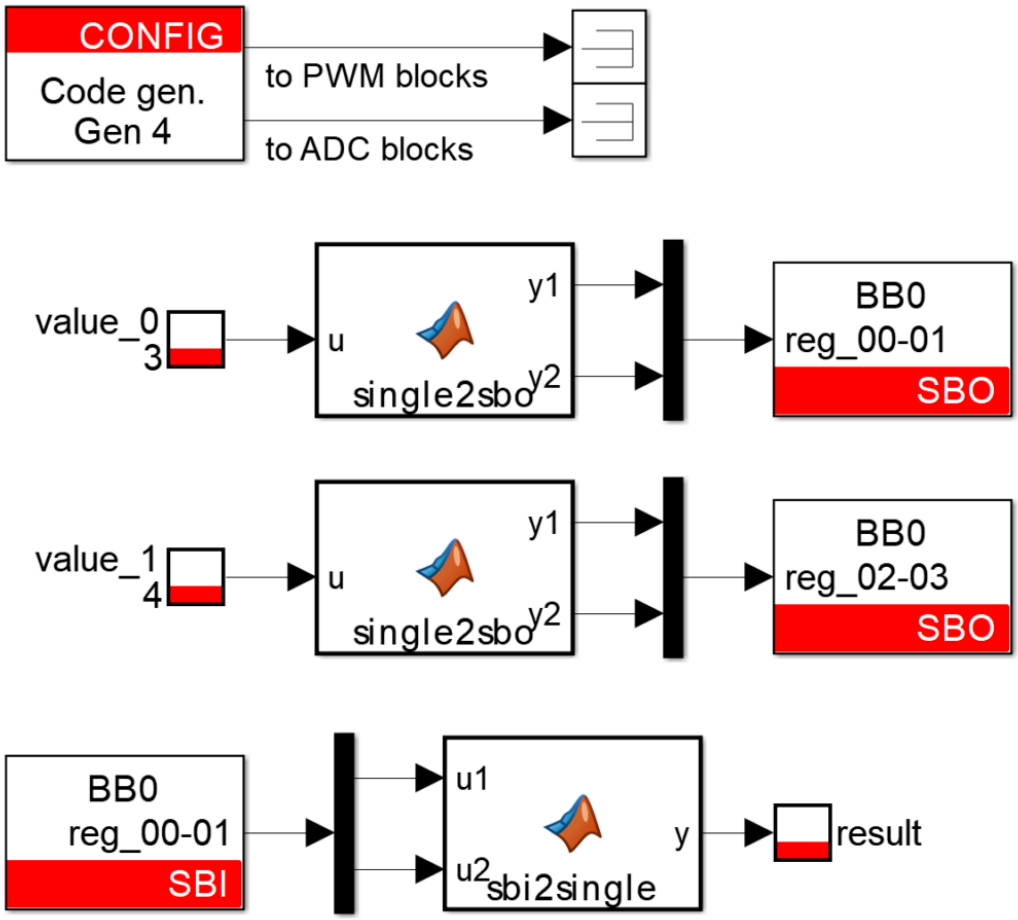
Save, build and load the design

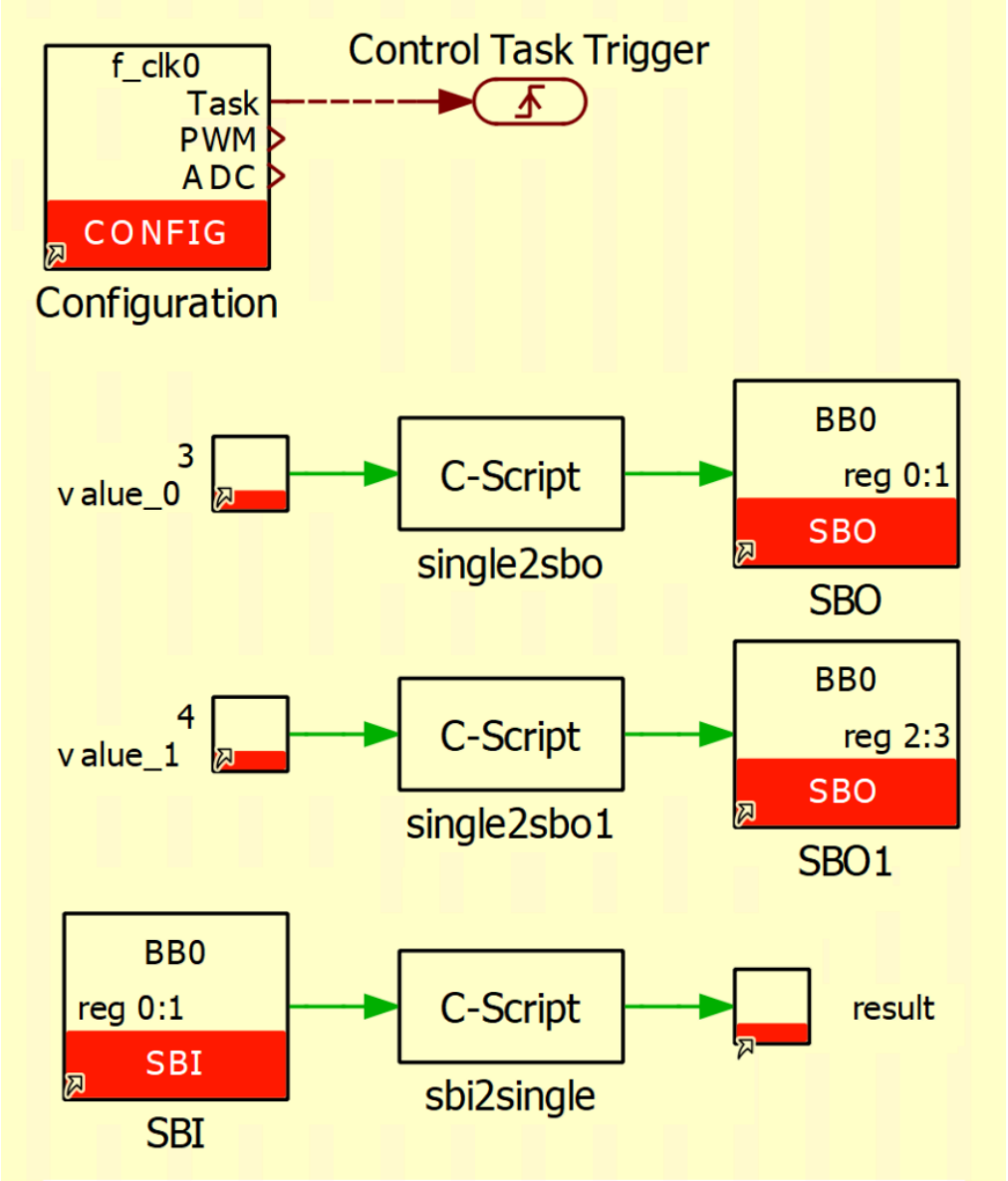
- Save the design and click on **Generate Bitstream** in the sidebar on the left.
- Load the bitstream onto the controller via Cockpit.

Experimental validation

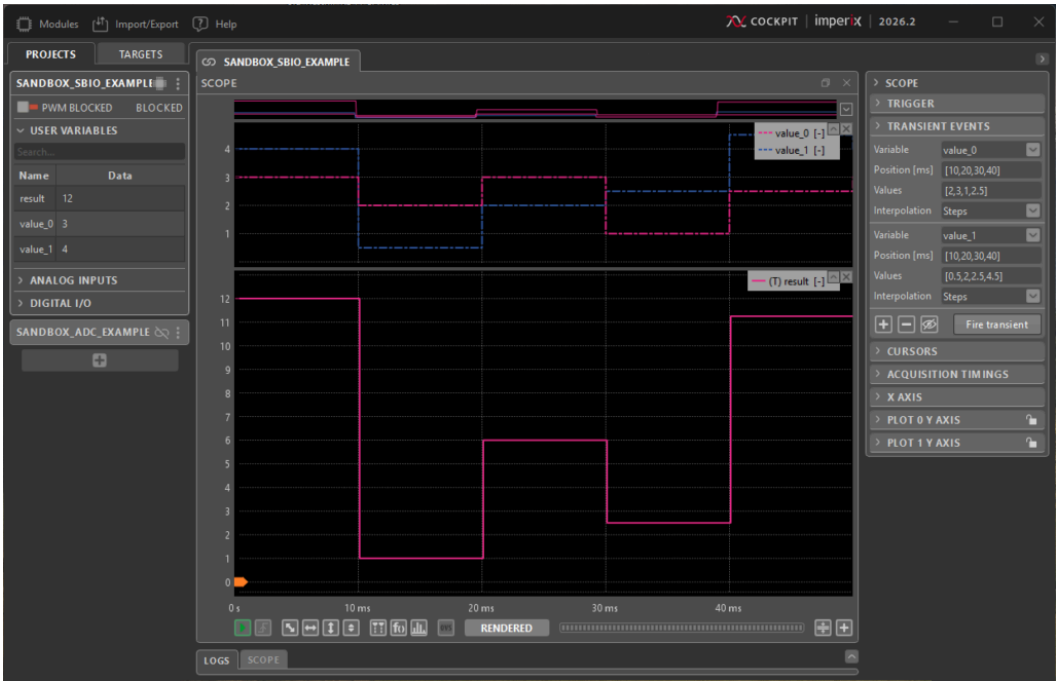
The FPGA design is tested using the CPU model provided below. It contains two tunable parameters for the two operands, which are transmitted to the FPGA through SBO blocks and the result is received from the FPGA through an SBI block. The **sbi2single** and **single2sbo** conversions are described in the Exchanging floats section above.

Download **pn128_sandbox_sbio_example.slx**
Download **pn128_sandbox_sbio_example.plecs**


CONFIG
Code gen.
Gen 4
to PWM blocks
to ADC blocks
value_0
3
u
single2sbo
y1
y2
BB0
reg_00-01
SBO
value_1
4
u
single2sbo
y1
y2
BB0
reg_02-03
SBO
BB0
reg_00-01
SBI
u1
u2
sbi2single
y
result



The expected results is shown in the figure below, which is obtained by checking different values for both operands thanks to the built-in Cockpit transient generator.



Going further

The TN147 example uses the **AXI4-Stream interface** for the CPU model to interact with the current control implemented in FPGA. The **CPU2FPGA** interfaces are used to write configuration signals (Kp, Ki, etc.) and slow-varying references (I_{ref}), while the **FPGA2CPU** interfaces are dedicated to monitoring and debugging.

In PN127, a PWM modulator is implemented in FPGA and the duty-cycle is provided by the CPU model. To avoid the additional delay introduced by the AXI4-Stream interface, the **sbio_register** helper is used instead. Two SBO register are concatenated to form a 32-bit floating-point value.