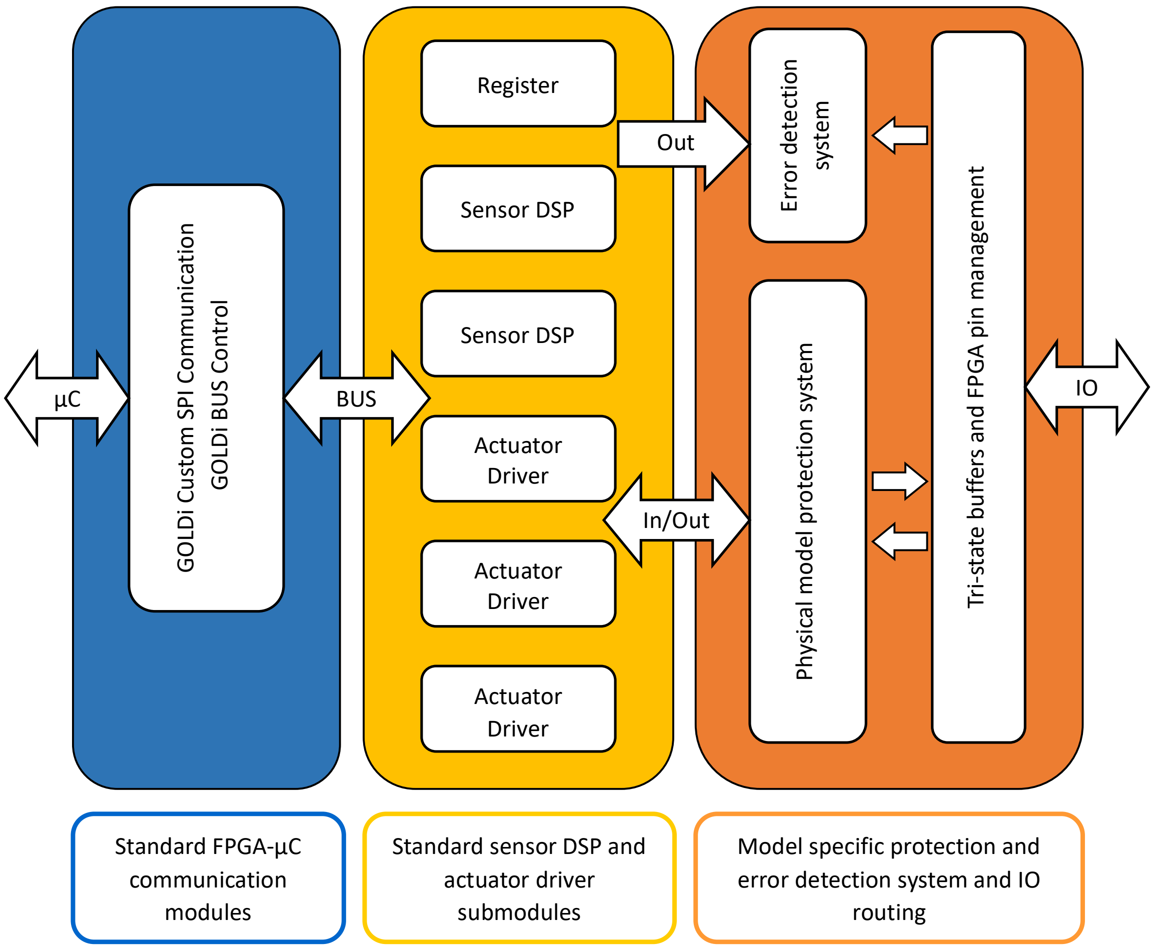
GOLDi FPGA Control Unit – System architecture

Overview

The Grid of Online Laboratory Devices Ilmenau (GOLDi) allows users to develop and test control algorithms for virtual and physical models such as an elevator, a warehouse, etc... At the core of these devices, also called a *GOLDi Model Unit*, lies an *FPGA Control Unit* dedicated to processing the signals needed to operate and control the actuators and sensors in the physical system. Given that multiple Model Units share similar elements like stepper motors, incremental encoders, LED lighting, etc.. a common digital architecture for all FPGAs has been designed. This unified system seeks to simplify the construction of new GOLDi Model Units, provide a standard interface for the microcontroller dedicated to the simulation of the user algorithm, and protect the physical model from damage caused by a user error.

The architecture of the system

The developed architecture is based on a three-tier hierarchical structure that reduces the model unit-specific elements and ensures the reusability of significant sections of the VHDL code. (Figure 1.)



µC
GOLDi Custom SPI Communication
GOLDi BUS Control
BUS
Register
Sensor DSP
Sensor DSP
Actuator Driver
Actuator Driver
Actuator Driver
Out
In/Out
Error detection system
Physical model protection system
Tri-state buffers and FPGA pin management
IO
Standard FPGA-µC communication modules
Standard sensor DSP and actuator driver submodules
Model specific protection and error detection system and IO routing

Figure 1. Architecture of the FPGA Control Unit for a standard GOLDi Model Unit

The first tier of the architecture is comprised of the primary communication modules of the FPGA and encompasses the microcontroller/FPGA interface and the system's internal bus master interface. The communication with the microcontroller, the device in charge of the main algorithmic tasks, is achieved through a serial peripheral interface SPI using a dedicated customizable protocol that grants access to the register structures that configure the FPGA system. The SPI protocol is based on several customizable parameters, like the address and data vector widths. Additionally, multiple data transfer modes are available. The protocol as well as the internal BUS architecture are defined in the *GOLDI_COMM_STANDARD* VHDL package.

The communication modules are located in the common/comms/goldi_spi directory. The GOLDI_SPI_SMODULE acts as the top module and uses two SP_CONVERTES to transform the configuration and data words from serial to parallel format. The BUS_ADAPTOR assigns the input data to the correct sub-bus and acts as the master interface of the system BUS. Any changes or extensions to the SPI protocol require, in theory, changes to the BUS_ADAPTOR only.

The second tier of the architecture consists of the generalized actuation and DSP submodules. These modules are principal elements of the Control Unit. Each submodule contains the digital logic needed to operate one of the Model Unit's elements. For example, the *HBRIDGE_SMODULE* is a DC-Motor driver (H-Bridge driver) including the digital logic needed to turn the motor on and off, selecting the motor's direction, and modifying the motor's speed. The data needed to configure a given submodule is stored in configurable register structures accessible through the system's main BUS. An additional characteristic of the submodules is that the output and input signals related to the model unit's elements are routed using custom bi-directional structures defined in the *GOLDI_IO_STANDARD* VHDL package. Using this structures allows the pins of an FPGA can be configured as "in/out" pins regardless of the specific function of it[1]. By connecting the pins to a tri-state buffer the submodules automatically select the required "in", "out" or "in/out" configuration and allow the signals to be re-routed dynamically using a crossbar element.

The submodules can be found in the common directory. They are characterized by the suffix "_SMODULE" and the use of the "sys_bus_i / sys_bus_o" signals in the port declaration. Additional modules often with the suffix "_DRIVER" or "_ANALYSER" are either encapsulated processed or even DSP modules that are missing the register structures and therefore can not be connected to the BUS directly. However, many of this modules can be easily repurposed for use in other submodules.

The third tier of the architecture is the only device specific part of the system and it consists of the protection system, error detection system, the tri-state buffer array, and synchronization components. The protection and error detection systems are particular for each Model Unit, taking the submodule outputs and FPGA inputs and disabling certain signals in case an error condition occurs. A more detailed explanation of each protection and error detection system can be found in the Model Unit's datasheet. In addition every Model Unit counts with an additional package, the *GOLDI_MODULE_CONFIG* package, containing the addresses of each submodule and the constant values required to instantiate them.

The device specific modules can be found in the source directory of each board. The modules are named "ACTUATOR_MASK" and "ERROR_DETECTOR". In the case of the Warehouse_V2 board there are additional "ACTUATOR_MASK_D", "ERROR_DETECTOR_D", and "WH_SENSOR_ARRAY" modules.

[1] Pins with additional or uncommon configurations are an exception and must be deliberately routed. For example I2C pins that must be configured as "open-drain" to prevent damage.

The described architecture has the It allows developers to extend the primary communication modules without changing each signal driver, it lets new actuator or sensor drivers be introduced for multiple Model Units at once, and it prevents damage to the physical model regardless of modifications to the submodules.

GOLDi SPI Protocol and GOLDi BUS

The GOLDi SPI Protocol consist of a configuration word and one or more data words transferred in an SPI transaction, defined as the data transferred in the period between a falling and rising edge on the chip select (nCE) signal. The communication with the FPGA is controller by the configuration word at the beginning of each SPI transaction. This input data configures the type of transaction in progress, the registers to be accessed during the data transfer, and additional tag modifiers that change the behavior of the stored data.

The configuration word starts with the write enable (WE) tag, which selects if the operation is read-only (WE=0) or a read-write (WE=1) type operation.

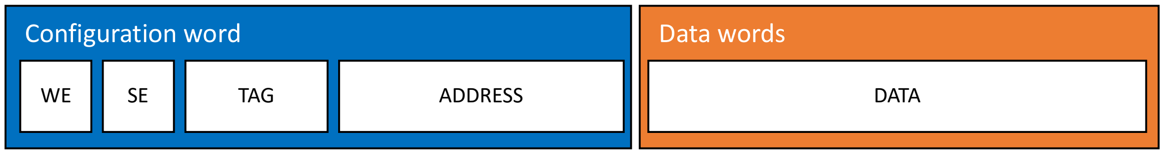


Figure 2. GOLDi SPI Protocol

The second bit, the stream-enable (SE) tag, selects the behavior of the SPI communication module when multiple data words are transferred in a single SPI transaction. When the multi-register mode is selected (SE=0), the provided address acts as the initial register to be accessed. After the data word has been transferred, the internal communication modules decrease the address value by one and the next register is accessed. This data transfer mode simplifies the data transfer to a submodule with multiple registers and data formats. In contrast, when the stream mode is selected (SE=1), only the addressed register is accessed and the stored data is overwritten/read after every data word has been transferred. This data transfer mode is used primarily to transfer large data vectors to secondary communication submodules that configure ICs or other electronics. The submodules often have queue structures that prevent data loses.

The steam-enable tag is followed by a series of multi-purpose tags that are applied to all the data words in the transaction. The configuration word is finished by the address vector.

The GOLDi SPI Protocol is a customizable protocol, therefore the exact widths of the address, tag, and data words can be modified in the GOLDI_COMM_STANDARD package when change to the BUS_ADDRESS_WIDTH, BUS_TAG_BITS, or SYSTEM_DATA_WIDTH is performed.

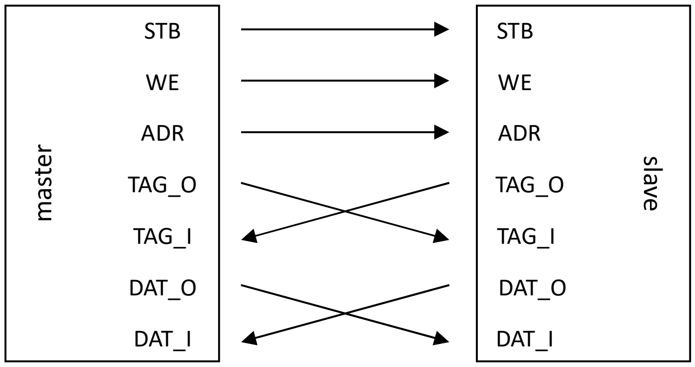


Figure 3. GOLDi system BUS

The GOLDi main communication modules take the data transferred by the SPI transaction and assign it to the master sub-buses WE, ADR, TAG, DAT_O of the GOLDi system BUS. The modules also take the data in the DAT_O slave sub-bus and transfer it to the microcontroller. The slave sub-bus TAG_O can be used internally and the slave MUX flag can be used to multiplex a vector of slave interfaces.

Crossbar and Crossbar BUS

The crossbars structure for IO tri-state signal allows the GOLDi FPGA Control Unit to perform multiple functions for a selected subgroup of FPGA pins. The structure allows multiple GOLDi submodules to access the data present on a pin simultaneously and also to drive the tri-state pin.

The crossbar works by storing two matrices containing the correspondence between the left and right sides of the structure. The first matrix, the "left_port_layout", contains the routing of the FPGA input data. The second matrix, the "right_port_layout", contains the routing of the internal system output data.

The dimensions of the crossbar structure are customizable under the assumption that the left side is always larger or equal the right side. The default layout of the crossbar after reset is defined in by the "g_default_left_layout" and "g_default_right_layout" constants in the GOLDI_CROSSBAR_CONFIG package.

V4.00.00 introduces the use of two matrices to store the initial configuration of the crossbar. Previously only the right side of the port was used and the initial configuration and the left side was inferred by inverting the matrix. Attention must be given to the definition of the left layout since multiple assignments of a left port to the right side will cause an error.

The configuration of the crossbar i.e. the matrices contents can be modified through a crossbar BUS structure. The BUS interface is the same as the GOLDi BUS standard, however the data is interpreted differently. The crossbar BUS translates the "adr" vector into the right pin number and the "dat" vector as the corresponding left pin to be connected. To ensure that a control register can be used to multiplex the main system BUS and multiple instances of the crossbar, the right pin number has been offset by 2.

A detailed explanation with examples can be found in the MOLE's datasheet.