
SPI Communication Protocol

The GOLDI Warehouse V2 is configured through an SPI interface. The SPI interface allows the independent reading and writing of values from and into the registers of the model.

SPI Signals

The GOLDI Warehouse V2 has four signals:

SPI0_SCLK	SPI clock
SPI0_MOSI	SPI serial data input
SPI0_MISO	SPI serial data output
SPI0_nCE0	SPI chip select input (active low)

The module is enabled for an SPI transaction by a logic low on the chip select input nCE0. Bit transfer is synchronous to the bus clock SCLK, with the slave latching the data from MOSI on the rising edge of SCLK and driving data to MISO on the falling edge. The most significant bit is sent first. A minimum of 24 SCLK clock cycles is required for a bus transaction.

(CONFIGURATION_WORD[15:0] + DATA_WORD[7:0])

If more than 24 clocks are driven, the additional bits shifted into MOSI are assigned to increasing lower addresses. If a read transaction with $24+n*8$ clocks is performed then the MISO shifts the data of the selected register and the registers with the addresses (adr-n). If a write transaction with $24+n*8$ clocks is performed the MOSI shifts the data to the selected registers and the registers with the address (adr-n)

nCE0 must be low during the whole SPI transaction. When nCE0 goes high, the unfinished transaction is discarded. The MOSI data is latched once the DATA_WORD is transferred

The configuration word length is based on the value "BUS_ADDRESS_WIDTH" in the GOLDI_COMM_STANDARD package. This corresponds to the address width + 1 bit for the write enable flag. The data word length is based on the value "SYSTEM_DATA_WIDTH" in the GOLDI_COMM_STANDARD package. This value corresponds to the number of data bits

Default configuration for the GOLDI Warehouse V2 model

<i>BUS_ADDRESS_WIDTH</i>	15
<i>SYSTEM_DATA_WIDTH</i>	8

Configuration Word [15:0]			Data Word[7:0]
Bit 15	Bit [14:8]	Bit [7:0]	Bit [7:0]
WE	REGISTER_ADDRESS		DATA[MSBF]
0	READ_ADDRESS[15:0]		[MOSI: dc] [MISO: Register data]
1	WRITE_ADDRESS[15:0]		[MOSI: New data] [MISO: Register data]

GOLDI Control Unit hardware pinout

Control Unit Pinout					
Hardware Pinout		FPGA System			
Signal Name	Schematic Name	Pin Type	Pin Number	Pin Mode	Entity name
Clock FPGA	ClockFPGA	in	128	LVC MOS33	ClockFPGA
Reset	FPGA_nReset	in	126	LVC MOS33	FPGA_nReset
SCLK	SPI0_SCLK	out	138	LVC MOS33	SPI0_SCLK
MOSI	SPI0_MOSI	out	133	LVC MOS33	SPI0_MOSI
MISO	SPI0_MISO	in	139	LVC MOS33	SPI0_MISO
nCE	SPI0_nCE0	out	127	LVC MOS33	SPI0_nCE0
GPIO0	CMGPIO0	inout	125	LVC MOS33	IO_DATA[0]
GPIO1	CMGPIO1	inout	122	LVC MOS33	IO_DATA[1]
X-Axis Limit Left	Stepper0_LSDir0	inout	84	LVC MOS33	IO_DATA[2]
X-Axis Limit Right	Stepper0_LSDir1	inout	85	LVC MOS33	IO_DATA[3]
Y-Axis Limit Outside	HBridge0A_LS	inout	15	LVC MOS33	IO_DATA[4]
Y-Axis Limit Inside	HBridge0B_LS	inout	17	LVC MOS33	IO_DATA[5]
Z-Axis Limit Bottom	Stepper1_LSDir1	inout	92	LVC MOS33	IO_DATA[6]
Z-Axis Limit Top	Stepper1_LSDir0	inout	91	LVC MOS33	IO_DATA[7]
Inductive sensor signal	Input7	inout	57	LVC MOS33	IO_DATA[8]
Encoder X Channel A	Stepper0_EncA	inout	87	LVC MOS33	IO_DATA[9]
Encoder X Channel B	Stepper0_EncB	inout	89	LVC MOS33	IO_DATA[10]
Encoder X Channel I	Stepper0_EncI	inout	86	LVC MOS33	IO_DATA[11]
Encoder Z Channel A	Stepper1_EncA	inout	104	LVC MOS33	IO_DATA[12]
Encoder Z Channel B	Stepper1_EncB	inout	105	LVC MOS33	IO_DATA[13]
Encoder Z Channel I	Stepper1_EncI	inout	106	LVC MOS33	IO_DATA[14]
X Motor Clock	Stepper0_CLK	inout	78	LVC MOS33	IO_DATA[15]
X Motor Enable	Stepper0_ENN	inout	77	LVC MOS33	IO_DATA[16]
X Motor Stall Guard	Stepper0_SG	inout	83	LVC MOS33	IO_DATA[17]
X Motor Step	Stepper0_STEP	inout	81	LVC MOS33	IO_DATA[18]
X Motor Direction	Stepper0_DIR	inout	82	LVC MOS33	IO_DATA[19]
X Motor Spi nCS	Stepper0_nCS	inout	76	LVC MOS33	IO_DATA[20]
X Motor Spi SCLK	Stepper0_SCK	inout	75	LVC MOS33	IO_DATA[21]
X Motor Spi MOSI	Stepper0_MOSI	inout	74	LVC MOS33	IO_DATA[22]
X Motor Spi MISO	Stepper0_MISO	inout	73	LVC MOS33	IO_DATA[23]
Y Motor Enable	HBridge0AB_Enable	inout	1	LVC MOS33	IO_DATA[24]
Y Motor Out Left	HBridge0A_PWM	inout	2	LVC MOS33	IO_DATA[25]
Y Motor Out Right	HBridge0B_PWM	inout	6	LVC MOS33	IO_DATA[26]
Z Motor Clock	Stepper1_CLK	inout	103	LVC MOS33	IO_DATA[27]
Z Motor Enable	Stepper1_ENN	inout	100	LVC MOS33	IO_DATA[28]
Z Motor Stall Guard	Stepper1_SG	inout	95	LVC MOS33	IO_DATA[29]
Z Motor Step	Stepper1_STEP	inout	93	LVC MOS33	IO_DATA[30]
Z Motor Direction	Stepper1_DIR	inout	94	LVC MOS33	IO_DATA[31]
Z Motor Spi nCS	Stepper1_nCS	inout	99	LVC MOS33	IO_DATA[32]
Z Motor Spi SCLK	Stepper1_SCK	inout	98	LVC MOS33	IO_DATA[33]

Z Motor Spi MOSI	Stepper1_MOSI	inout	97	LVCMOS33	IO_DATA[34]
Z Motor Spi MISO	Stepper1_MISO	inout	96	LVCMOS33	IO_DATA[35]
LED Power Red	LEDPowerR	inout	141	LVCMOS33	IO_DATA[36]
LED Power Green	LEDPowerG	inout	140	LVCMOS33	IO_DATA[37]
Environment Light Red	LightRed	inout	33	LVCMOS33	IO_DATA[38]
Environment Light White	LightWhite	inout	34	LVCMOS33	IO_DATA[39]
Environment Light Green	LightGreen	inout	35	LVCMOS33	IO_DATA[40]

Register Map

Document Version 1
Hardware Version V2.00.00
Date 30.04.2023

All registers have a base address located on the package GOLDI_MODULE_CONFIG. This can be changed to move the modules in case the configuration word width is changed.

Register Name	Address (Dec)	Address (Hex)	Default	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
System Configuration	1	0x01	0x00							ENC_rst	BUS_sel
Sensors: model	2	0x02	0x00		inductive	z_top	z_bottom	y_inside	y_outside	x_right	x_left
Sensors: virtual low	3	0x03	0x00	virtual_x[8]	virtual_x[7]	virtual_x[6]	virtual_x[5]	virtual_x[4]	virtual_x[3]	virtual_x[2]	virtual_x[1]
Sensors: virtual high	4	0x04	0x00		virtual_z[5]	virtual_z[4]	virtual_z[3]	virtual_z[2]	virtual_z[1]	virtual_x[10]	virtual_x[9]
Error list 1	5	0x05	0x00	error_7	error_6	error_5	error_4	error_3	error_2	error_1	error_0
Error list 2	6	0x06	0x00				error_12	error_11	error_10	error_9	error_8
GPIO0 Driver	7	0x07	0x00							Out_enb	Data
GPIO1 Driver	8	0x08	0x00							Out_enb	Data
X Encoder low	9	0x09	0x00	X_VAL[7:0]							
X Encoder high	10	0x0A	0x00	X_VAL[15:8]							
Z Encoder low	11	0x0B	0x00	Z_VAL[7:0]							
Z Encoder high	12	0x0C	0x00	Z_VAL[15:8]							
X Motor Control	13	0x0D	0x00	Pow_off					Stall	Dir1	Dir0
X Motor Speed	14	0x0E	0x00	FRQ_VAL[7:0]							
X Motor Speed	15	0x0F	0x00	FRQ_VAL[15:8]							
X Motor SPI 0	16	0x10	0x07	CONFIG_WORD[7:0]							
X Motor SPI 1	17	0x11	0x00	CONFIG_WORD[15:8]							
X Motor SPI 2	18	0x12	0x00	CONFIG_WORD[23:16]							
Y Motor Direction	19	0x13	0x00							Y_Inside	Y_Outside
Y Motor Speed	20	0x14	0x00	PWM[7:0]							
Z Motor Control	21	0x15	0x00	Pow_off					Stall	Dir1	Dir0
Z Motor Speed	22	0x16	0x00	FRQ_VAL[7:0]							
Z Motor Speed	23	0x17	0x00	FRQ_VAL[15:8]							
Z Motor SPI 0	24	0x18	0x07	CONFIG_WORD[7:0]							
Z Motor SPI 1	25	0x19	0x00	CONFIG_WORD[15:8]							
Z Motor SPI 2	26	0x20	0x00	CONFIG_WORD[23:16]							
Power LED Red	27	0x21	0x00	on/off	Blink_enb	Delay_on			Delay_off		
Power LED Green	28	0x22	0x00	on/off	Blink_enb	Delay_on			Delay_off		
Environment Light Red	29	0x23	0x00	on/off	Blink_enb	Delay_on			Delay_off		
Environment Light White	30	0x24	0x00	on/off	Blink_enb	Delay_on			Delay_off		
Environment Light Green	31	0x25	0x00	on/off	Blink_enb	Delay_on			Delay_off		

Error List

Error code	Error definition
error_0	Limit sensors left and right active
error_1	Limit sensors y-Outside and y-Inside active
error_2	Limit sensors bottom and top active
error_3	Motor x drive to left active and limit left active
error_4	Motor x drive to right active and limit right active
error_5	Motor y drive to outside active and limit outside active
error_6	Motor y drive to inside active and limit inside active
error_7	Motor z drive to bottom active and limit bottom active
error_8	Motor z drive to top active and limit top active
error_9	Crane out of the horizontal virtual box in the left limit
error_10	Crane out of the horizontal virtual box in the right limit
error_11	Crane out of the vertical virtual box in the bottom limit
error_12	Crane out of the vertical virtual box in the top limit

Stepper Motor utilization

General description

The stepper motors are driven by the TMC2660. The TMC2660 is a driver for two-phase stepper motors with multiple industrial features. The driver includes high-resolution microstepping, sensorless mechanical load measurement, load measurement, load-adaptive power optimization, and low-resonance chopper operation. It is operated through either a standard SPI interface or a STEP/DIRECTION interface.

Configuration process:

The TMC2660 requires setting configuration parameters and mode bits through the SPI interface before the motor can be driven. The SPI interface also allows reading back status values and bits.

The warehouse_v2 hardware provides the TMC2660 with an initial configuration when the module is started. 5 SPI 24-bit transactions are performed automatically after initialization to write the data to the 5 registers of the TMC2660 and activate the TMC2660. The default values for the initial configuration are located in the GOLDI_MODULE_CONFIG module in the TMC2660_rom structure. Once the hardware is started a FIFO structure loads the data into the SPI transmitter until the module has been programmed. After this initial configuration, the module can be operated through the STEP/DIRECTION interface.

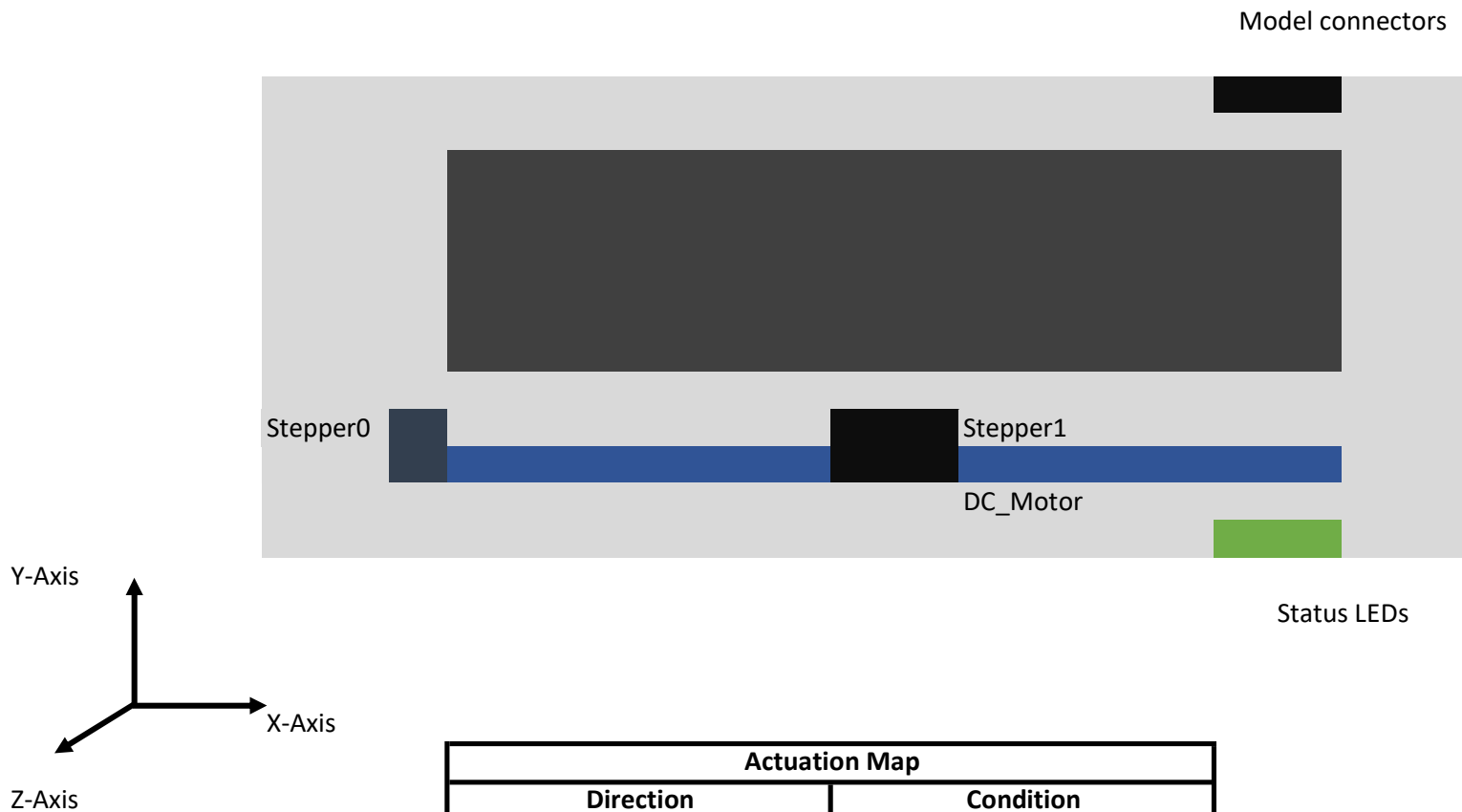
After the initial configuration, the TMC2660 data can be modified by the user mid-operation using the three SPI registers in the TMC2660_DRIVER. Given that the TMC2660 returns the same data regardless of the rewritten register, three registers are enough to efficiently communicate with the chip. The data is passed to the FIFO structure to queue the SPI transmitter. The SPI stream interface reacts to the write strobe of the lower register, meaning the FIFO structure is loaded with the register's data when the register "SPI 0" data is modified. The FIFO structure has been placed between the registers and SPI transmitter to allow the user to program up to 5 configuration words with the faster FPGA SPI interface without data losses.

STEP/DIRECTION interface (normal actuation of motor):

To drive the stepper motor the STEP/DIRECTION interface of the TMC2660 is used. This interface is operated by the lower three registers in the TMC2660_DRIVER (the "motor speed" and "motor control") registers.

The motor speed registers contain the step signal frequency expressed in Hertz and the motor control register has two direction dependent enable signals. (If both are active the Dir1 takes precedent) Additionally, the stall bit reads the StallGuard2 information of the TMC2660 and flags a stall of the stepper motor. The power off bit temporarily disables the TMC2660 and allows the stepper motor to rotate freely in case it has to be moved manually and should be activated before the FPGA reprogramming. The STEP/DIRECTION interface takes the speed value at the beginning of the operation. To change the speed of the motor the driver must first be stopped.

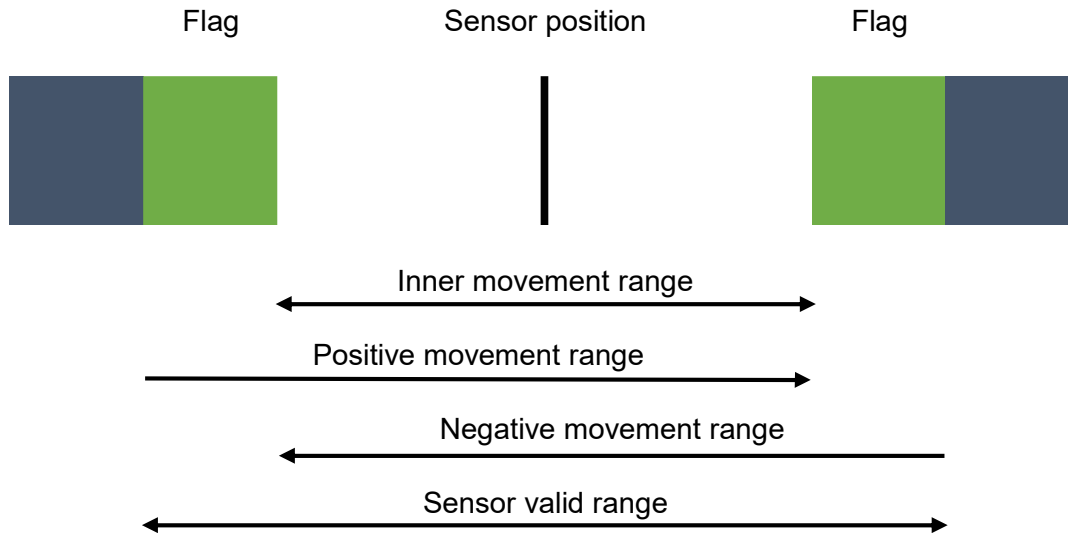
Physical model reference map



Actuation Map	
Direction	Condition
x_neg left	Stepper0 -> Dir0
x_pos right	Stepper0 -> Dir1
y_neg Outside	DC -> Outside
y_pos Inside	DC -> Inside
z_neg bottom	Stepper1 -> Dir0
z_pos top	Stepper1 -> Dir1

Virtual limit principle for the model protection

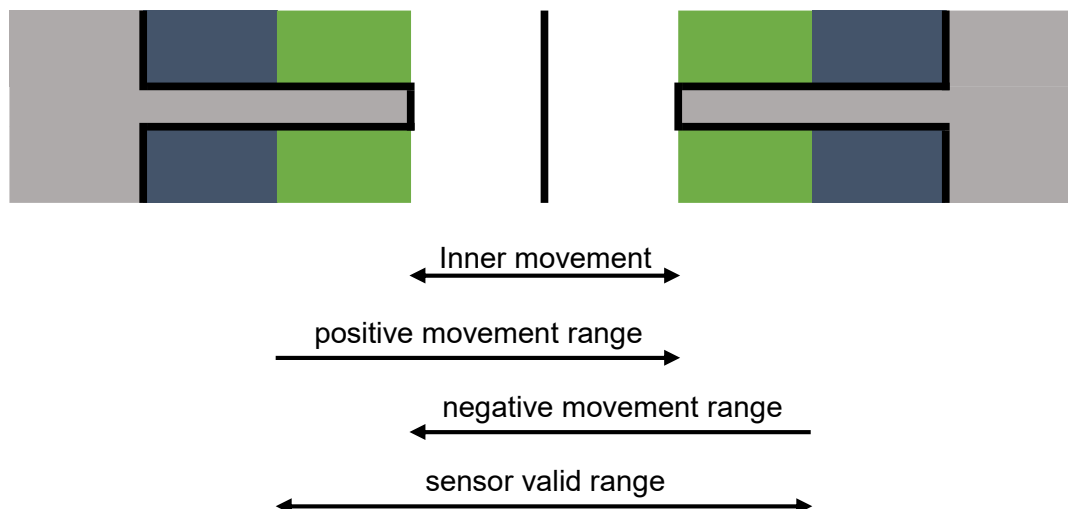
Virtual limit description:



The principle of virtual limits works by processing the data provided by the incremental encoders to calculate regions in which a virtual sensor is detected, the "sensor valid range". Additionally, the virtual limit enables a margin flag that indicates a closeness to either the negative or positive limit of the sensor valid range.

To protect the model from damage when it is actuated within the shelves the virtual limits are used to create a virtual box in which the x- and z- motors can move freely. The virtual box is created by a virtual limit in the x- and z-axis. The actuation mask that protects the system blocks the movement of the motor in a direction if the corresponding margin flag is asserted and if the position is not inside a virtual box. The addition of the flag margin allows the crane to be inserted in an out of bounds position and be aligned with the correct margin in both the x- and z-axis.

Shelf diagram



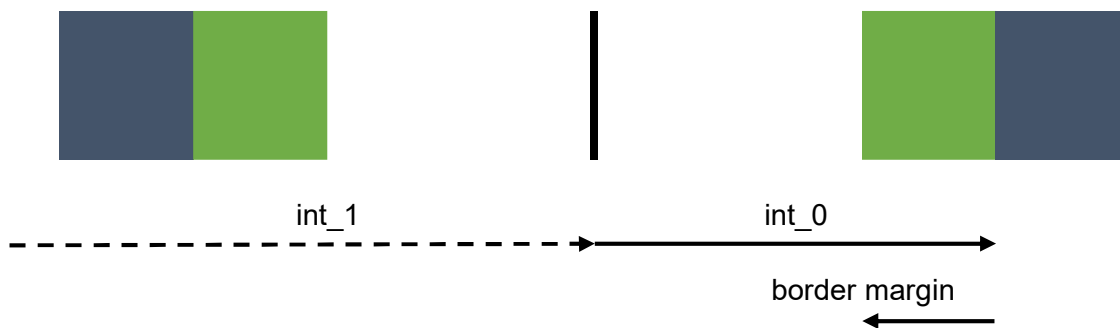
Configuration of virtual limits:

The configuration of the virtual limits is programmed using three values. The sensor position, sensor width and border margin. The border margin is the same for all virtual limits in an array, however the sensor width can be configured independently using the "sensor_limit" and "sensor_limit_array" structures defined in the GOLDI_DATA_TYPE package and instantiated in the GOLDI_MODULE_CONFIG package. The "sensor_limit" is a set of two integers (int_1,int_0).

int_1: Corresponds with the position of the limit center line

int_0: Corresponds with the width from a lateral limit to the center line

border margin: Corresponds with the width of the flag range from the lateral limit to the inside of the box



negative limit = $\text{int_1} - \text{int_0}$

positive limit = $\text{int_1} + \text{int_0}$

negative inner limit = $\text{int_1} - \text{int_0} + \text{border_margin}$

positive inner limit = $\text{int_1} + \text{int_0} - \text{border_margin}$