GOLDI Warehouse V2 - FPGA Control Unit

Document version:	3
Hardware Version	V4.00.00
Release date:	30.09.2023

Overview:

The GOLDI Warehouse V2 FPGA Control Unit is the main digital processing unit used in the GOLDi Warehouse V2 to gather and control the sensor and actuator signals used in the physical model. The control unit uses a Lattice MachX02 FPGA to operate the system and provides a standardized SPI interface for the microcontroller unit to acces the model's peripheral elements through a set of registers implemented in the FPGA

Calibration procedure:

The GOLDI Warehouse V2 FPGA Control Unit requires an initial reset of the encoders after power up to operate correctly. The protection system that prevents collisions when the model's crane is inside one of the warehouse's shelves is based on absolute encoder values. Therefore, the encoders must be zeroed when the crane is in the utmost left and down position, i.e. the negative endpoints of each axis of movement. This must be done manually after initialization by driving the crane to the limits and resetting the encoders using the control register.

Operation:

The GOLDI Warehouse V2 is operated by writing the corresponding values into the FPGA registers (See Register Table). To access one of the shelves the Warehouse V2 protection system must be reconfigured to allow the movement of the crane.

SPI Communication Protocol

The FPGA Control Unit 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 implemented signals:

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

The FPGA Control Unit is enabled for communication when a logic low is presented on the chip select input nCE0. The data transfered between a falling and rising edge of nCE0 is defined as a SPI transaction. The data bit transfer during a transaction is synchronous to the bus clock SCLK. The peripheral FPGA Control Unit registers data from MOSI on the rising edge of SCLK and drives data to MISO on the falling edge. The most significant bit is sent first. The SCLK signal is expected to remain high when the module is idle.

A valid SPI transaction consists of a 16-bit *configuration word* and one or more 8-bit *data words*, meaning a minimum of 24 SCLK clock cycles is required for a single valid transaction. If the nCE0 is driven high before the minimum 24 data bits have been registered or before an additional data word has been transfered the current data is discarded.

GOLDi SPI Protocol
The communication with the FPGA is controled by the *configuration word* at the begining 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 read-write (WE='1').
The next bit, the stream enable "SE" bit, selects the behavior of the SPI communication module when multiple *data words* are transfered in a single SPI transaction. When the multi-register mode is selected (SE='0'), the provided address acts as the inital register to be accessed. After the first operation has been performed, meaning the first data word has been transfered, the internal BUS address is drecreased by one and the next register is accessed. This data transfer mode simplifies the data transfer to a sub-module 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 overwriten/read after every *data word* has been transfered. This data transfer mode is used primarly to transfer large data vectors to secondary communication sub-modules that configure ICs or other electronics. The sub-modules often have queue structures that prevent data loses.

The stream-enable bit is followed by a 4-bit tag word that is applied to all data words in the transaction and the 10-bit address word. Up to V4.00.00, the tag word remains unused by any sub-module in the GOLDi Warehouse V2.

The GOLDi SPI Protocol is used by multiple GOLDi Model Units, therefore, the exact widths of the address, tag, and data words can be modified in the GOLDI_COMM_STANDARD package. A change to the "BUS_ADDRESS_WIDTH", "BUS_TAG_BITS", or "SYSTEM_DATA_WIDTH" automaticaly resizes the entire system.

Default configuration for the GOLDi Warehouse V2 model

BUS_ADDRESS_WIDTH	10
BUS_TAG_BITS	4
SYSTEM_DATA_WIDTH	8

Configuration Word [15:0]				Data Word[7:0]
Bit 15	**Bit [14]**	**Bit [13:10]**	**Bit [9:0]**	**Bit [7:0]**
WE	SE	TAG	ADDRESS	DATA[MSBF]
0	0	-	READ_START[9:0]	[MOSI: dc] | [MISO: Register data]
0	1	-	READ_ONLY[9:0]	[MOSI: dc] | [MISO: Register data]
1	0	-	WRITE_START[9:0]	[MOSI: New data] | [MISO: Register data]
1	1	-	WRITE_ONLY[9:0]	[MOSI: New data] | [MISO: Register data]

GOLDI Warehouse V2 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	LVCMOS33	ClockFPGA
Reset	FPGA_nReset	in	126	LVCMOS33	FPGA_nReset
SCLK	SPI0_SCLK	out	138	LVCMOS33	SPI0_SCLK
MOSI	SPI0_MOSI	out	133	LVCMOS33	SPI0_MOSI
MISO	SPI0_MISO	in	139	LVCMOS33	SPI0_MISO
nCE	SPI0_nCE0	out	127	LVCMOS33	SPI0_nCE0
GPIO0	CMGPIO0	inout	125	LVCMOS33	IO_DATA[0]
GPIO1	CMGPIO1	inout	122	LVCMOS33	IO_DATA[1]
X-Axis Limit Left	Stepper0_LSDir0	inout	84	LVCMOS33	IO_DATA[2]
X-Axis Limit Right	Stepper0_LSDir1	inout	85	LVCMOS33	IO_DATA[3]
Y-Axis Limit Outside	HBridge0A_LS	inout	15	LVCMOS33	IO_DATA[4]
Y-Axis Limit Inside	HBridge0B_LS	inout	17	LVCMOS33	IO_DATA[5]
Z-Axis Limit Bottom	Stepper1_LSDir1	inout	92	LVCMOS33	IO_DATA[6]
Z-Axis Limit Top	Stepper1_LSDir0	inout	91	LVCMOS33	IO_DATA[7]
Inductive sensor signal	Input7	inout	54	LVCMOS33	IO_DATA[8]
Encoder X Channel A	Stepper0_EncA	inout	87	LVCMOS33	IO_DATA[9]
Encoder X Channel B	Stepper0_EncB	inout	89	LVCMOS33	IO_DATA[10]
Encoder X Channel I	Stepper0_EncI	inout	86	LVCMOS33	IO_DATA[11]
Encoder Z Channel A	Stepper1_EncA	inout	104	LVCMOS33	IO_DATA[12]
Encoder Z Channel B	Stepper1_EncB	inout	105	LVCMOS33	IO_DATA[13]
Encoder Z Channel I	Stepper1_EncI	inout	106	LVCMOS33	IO_DATA[14]
X Motor Clock	Stepper0_CLK	inout	78	LVCMOS33	IO_DATA[15]
X Motor Enable	Stepper0_ENN	inout	77	LVCMOS33	IO_DATA[16]
X Motor Stall Guard	Stepper0_SG	inout	83	LVCMOS33	IO_DATA[17]
X Motor Step	Stepper0_STEP	inout	81	LVCMOS33	IO_DATA[18]
X Motor Direction	Stepper0_DIR	inout	82	LVCMOS33	IO_DATA[19]
X Motor Spi nCS	Stepper0_nCS	inout	76	LVCMOS33	IO_DATA[20]
X Motor Spi SCLK	Stepper0_SCK	inout	75	LVCMOS33	IO_DATA[21]
X Motor Spi MOSI	Stepper0_MOSI	inout	74	LVCMOS33	IO_DATA[22]
X Motor Spi MISO	Stepper0_MISO	inout	73	LVCMOS33	IO_DATA[23]
Y Motor Enable	HBridge0AB_Enable	inout	1	LVCMOS33	IO_DATA[24]
Y Motor Out Left	HBridge0A_PWM	inout	2	LVCMOS33	IO_DATA[25]
Y Motor Out Right	HBridge0B_PWM	inout	6	LVCMOS33	IO_DATA[26]
Z Motor Clock	Stepper1_CLK	inout	103	LVCMOS33	IO_DATA[27]
Z Motor Enable	Stepper1_ENN	inout	100	LVCMOS33	IO_DATA[28]
Z Motor Stall Guard	Stepper1_SG	inout	95	LVCMOS33	IO_DATA[29]
Z Motor Step	Stepper1_STEP	inout	93	LVCMOS33	IO_DATA[30]
Z Motor Direction	Stepper1_DIR	inout	94	LVCMOS33	IO_DATA[31]
Z Motor Spi nCS	Stepper1_nCS	inout	99	LVCMOS33	IO_DATA[32]
Z Motor Spi SCLK	Stepper1_SCK	inout	98	LVCMOS33	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 3
Hardware Version V4.00.00
Date 30.09.2023

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	ID=0100 (rst)					mask_unblock	ref_z_enc	rst_x_enc
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]
Protection mask: x neg low	5	0x05	0x00	x_lower_limit [7:0]							
Protection mask: x neg high	6	0x06	0x00	x_lower_limit [15:8]							
Protection mask: x pos low	7	0x07	0x00	x_upper_limit [7:0]							
Protection mask: x pos high	8	0x08	0x00	x_upper_limit [15:8]							
Protection mask: y neg low	9	0x09	0x00	y_lower_limit [7:0]							
Protection mask: y neg high	10	0x0A	0x00	y_lower_limit [15:8]							
Protection mask: y pos low	11	0x0B	0x00	y_upper_limit [7:0]							
Protection mask: y pos high	12	0x0C	0x00	y_upper_limit [15:89]							
Error list 1	13	0x0D	0x00	error_7	error_6	error_5	error_4	error_3	error_2	error_1	error_0
Error list 2	14	0x0E	0x00		error_14	error_13	error_12	error_11	error_10	error_9	error_8
GPIO0 Driver	15	0x0F	0x00							out_enb	data
GPIO1 Driver	16	0x10	0x00							out_enb	data
X Encoder low	17	0x11	0x00	x_val [7:0]							
X Encoder high	18	0x12	0x00	x_val [15:8]							
Z Encoder low	19	0x13	0x00	z_val [7:0]							
Z Encoder high	20	0x14	0x00	z_val [15:8]							
X Motor Control	21	0x15	0x00	pow_off					stall	dir1	dir0
X Motor Speed	22	0x16	0x00	frq_val [7:0]							
X Motor Speed	23	0x17	0x00	frq_val [15:8]							
X Motor SPI 0	24	0x18	0x07	config_word [7:0]							
X Motor SPI 1	25	0x19	0x00	config_word [15:8]							
X Motor SPI 2	26	0x1A	0x00	config_word [23:16]							
Y Motor Direction	27	0x1B	0x00							y_inside	y_outside
Y Motor Speed	28	0x1C	0x00	pwm [7:0]							
Z Motor Control	29	0x1D	0x00	pow_off					stall	dir1	dir0
Z Motor Speed	30	0x1E	0x00	frq_val [7:0]							
Z Motor Speed	31	0x1F	0x00	frq_val [15:8]							
Z Motor SPI 0	32	0x20	0x07	config_word [7:0]							
Z Motor SPI 1	33	0x21	0x00	config_word [15:8]							
Z Motor SPI 2	34	0x22	0x00	config_word[23:16]							
Power LED Red	35	0x23	0x00	on/off	blink_enb	delay_on			delay_off		
Power LED Green	36	0x24	0x00	on/off	blink_enb	delay_on			delay_off		
Environment Light Red	37	0x25	0x00	on/off	blink_enb	delay_on			delay_off		
Environment Light White	38	0x26	0x00	on/off	blink_enb	delay_on			delay_off		
Environment Light Green	39	0x27	0x00	on/off	blink_enb	delay_on			delay_off		

Error List

Error code	Error definition
error_0	X limit sensors negative and positive active
error_1	X motor active in negative direction and negative sensor active
error_2	X motor active in positive direction and positive sensor active
error_3	Y limit sensors negative and positive active
error_4	Y motor active in negative direction and negative sensor active
error_5	Y motor active in positive direction and positive sensor active
error_6	Z limit sensors negative and positive active
error_7	Z motor active in negative direction and negative sensor active
error_8	Z motor active in positive direction and positive sensor active
error_9	X lower virtual limit reached and motor active in negative direction
error_10	X upper virtual limit reached and motor active in positive direction
error_11	X virtual lower limit larger than virtual upper limit
error_12	Z lower virtual limit reached and motor active in negative direction
error_13	Z upper virtual limit reached and motor active in positive direction
error_14	Z virtual lower limit larger than virtual upper limit

TMC2660 Stepper Motor Driver Interface

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 si operated through either a standard SPI interface of 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 Axis Portal V2 hardware provides the TMC2660 with an initial configuration when the module is started. Five 24-bit SPI transactions are performed automatically after initialization to write the data to the 5 registers of the TMC2660 and activate the driver. The default values for the initial configuration are located in the GOLDI_MODULE_CONFIG package stored in an "array_16_bit" structure. This data structure stores 16-bit words in instantiated PLU ROM units. 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_SMODULE. Given that the TMC2600 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 that queues 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.

Dynamic Protection Mask (V4.00.00)

Overview:
The dynamic protection mask developed for V4.00.00 improves the old static protection mask implemented in V3.00.00 by reducing the logic complexity, simplifying the calibration process, and reducing the effects of a possible encoder drift. The dynamic mask takes a lower and upper limit value stored in registers and compares it to the crane's position and motor signals to limit the model and prevent damage. Additionlly, an error detector module mirrors the limit value data in the protection mask and flags the errors to the user.

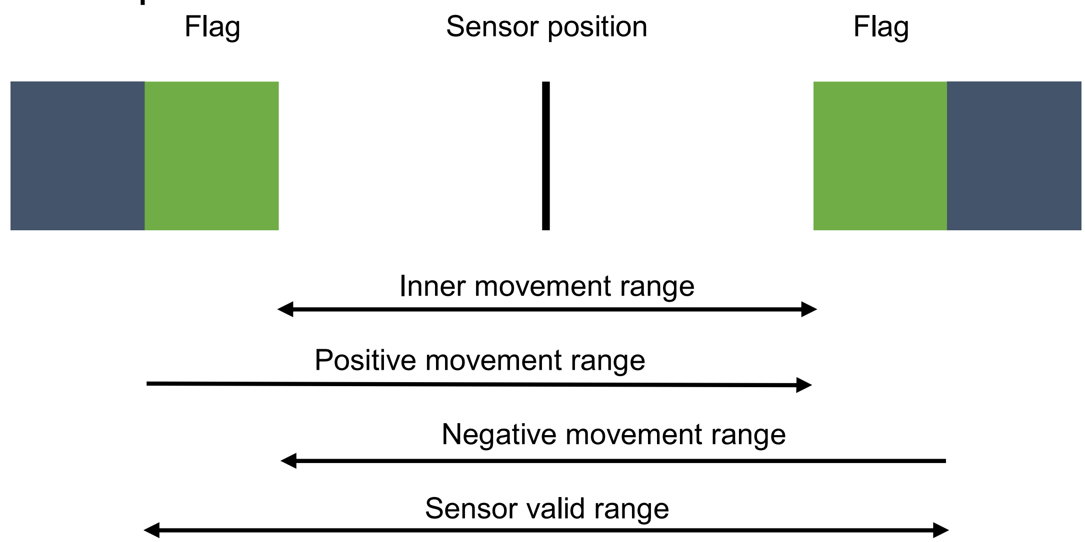
Operation of the protection mask:
The dynamic protection mask limits the crane's movement by comparing the current position data gathered by the x- and z-axis incremental encoders and comparing it to a lower and upper limit for each axis. The mask grounds the motor driver signals if the crane position is equal to or exceeds the values provided by the user. The y-axis motor is blocked if any virtual limit has been met. This ensures that the y-motor can only be activated when the crane is in a deliberately selected area of Warehouse V2.

The dynamic protection mask returns all virtual limits to zero after power-up. This automatically limits the movement of the crane in the x-, y- and z- directions and prevents a user mistake. To calibrate the system, the crane must be positioned at the negative limits of each axis (left limit and lower limit, see reference map); once there, the x- and z-encoder reset/reference flags in the control register must be asserted to zero the encoder DSP units. Given that the encoder DSP units use unsigned values to calculate the crane's position, it is possible that the negative limits can not be reached after a power-up. For this reason, an additional signal, the "ref_unblock" flag, has been added to the mask. If this signal is asserted, the protection mask will ignore the virtual limits and stop the motor only when the physical limits of the model have been reached. To prevent the accidental movement of the crane in the y-direction while the protection system is not active, the y-motor is blocked when the "ref_unblocked" signal is valid.

Once the calibration process is finished the normal operation of the mask can be assumed. To move the crane to a desired loading bay the coordinates of the center point of that bay are set as both the upper and lower limits. This prevents the y-motor from engaging prematurely and also eliminates the need to align the motor with the loading bay manually since the crane will be stopped once the correct position is reached. After the desired possition has been reached the virtual limits can be modified to form a perimeter around the shelf that enables the y-motor movement.

Virtual limit principle for the model protection (V3.00.00)

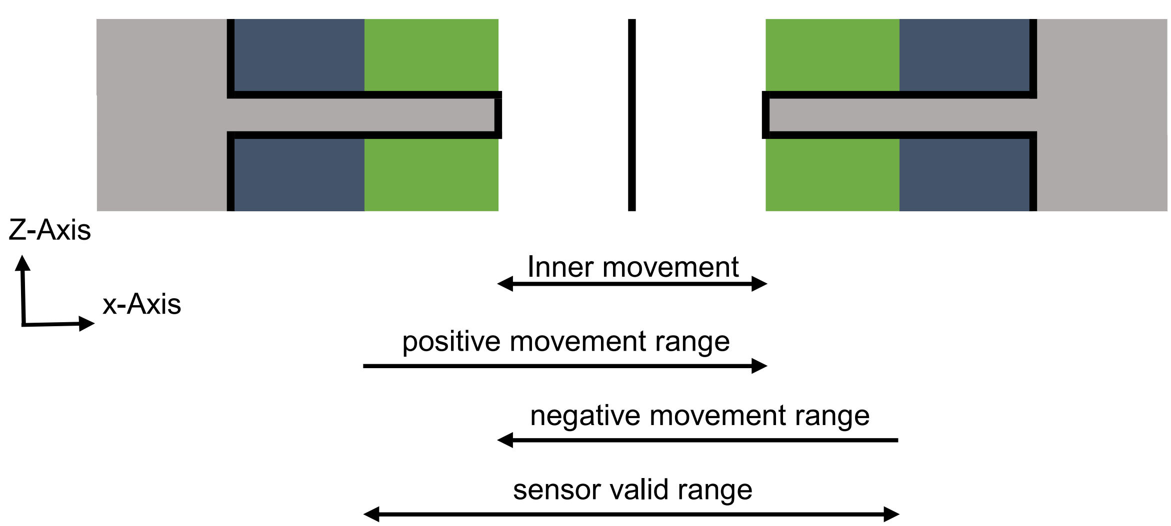
Virtual limit description:



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.

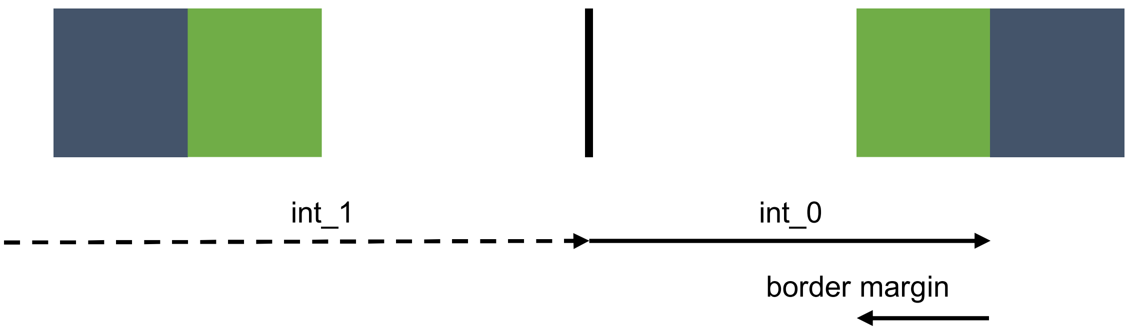
To protect the model from damage when it is actuated within the shelves the virtual limits are used to create a virtual box in which the x- and z- motors can move freely. The virtual box is created by a virtual limit in the x- and z-axis. The actuation mask that protects the system blocks the movement 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

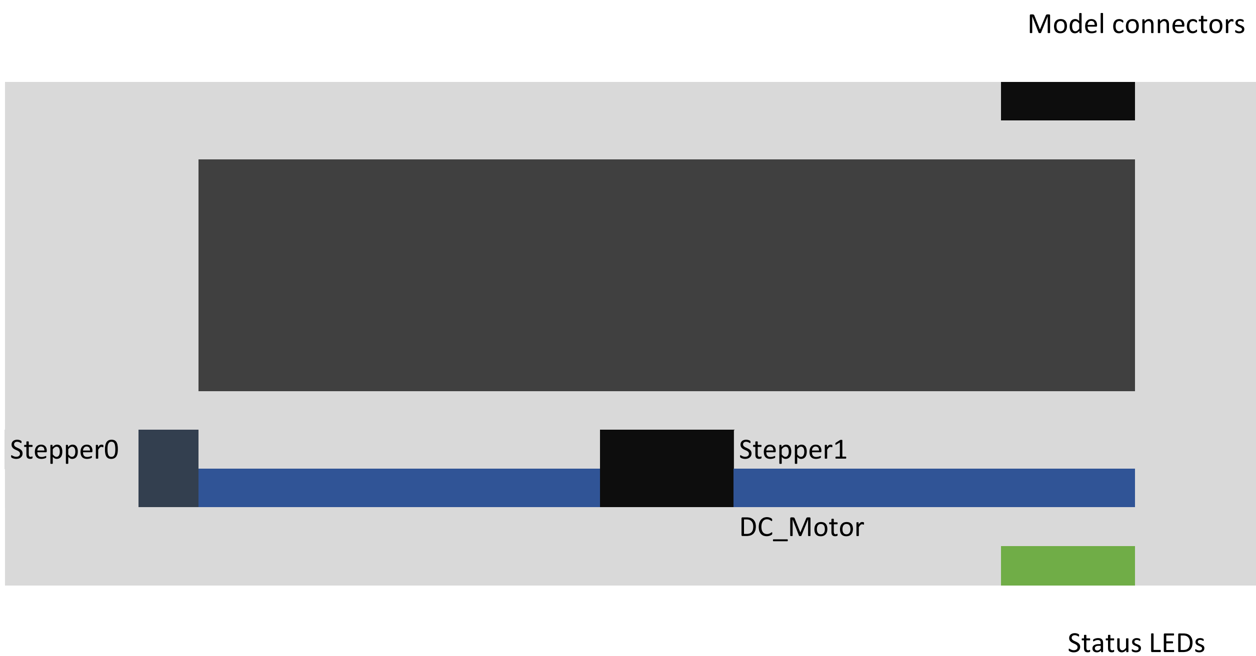


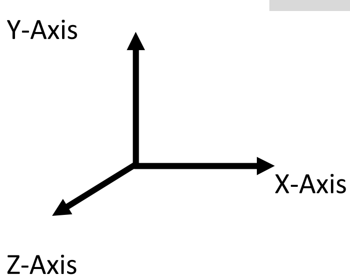
Configuration of virtual limits:
The configuration of the virtual limits is programmed using three values. The sensor position, sensor width and border margin. The border margin is the same for all virtual limits in an array, however the sensor width can be configured independently using the "sensor_limit" 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 = int_1- int_0
positive limit = int_1 + int_0
negative inner limit = int_1 - int_0 + border_margin
positive inner limit = int_1 + int_0 - border_margin





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