STSPIN logic core basics

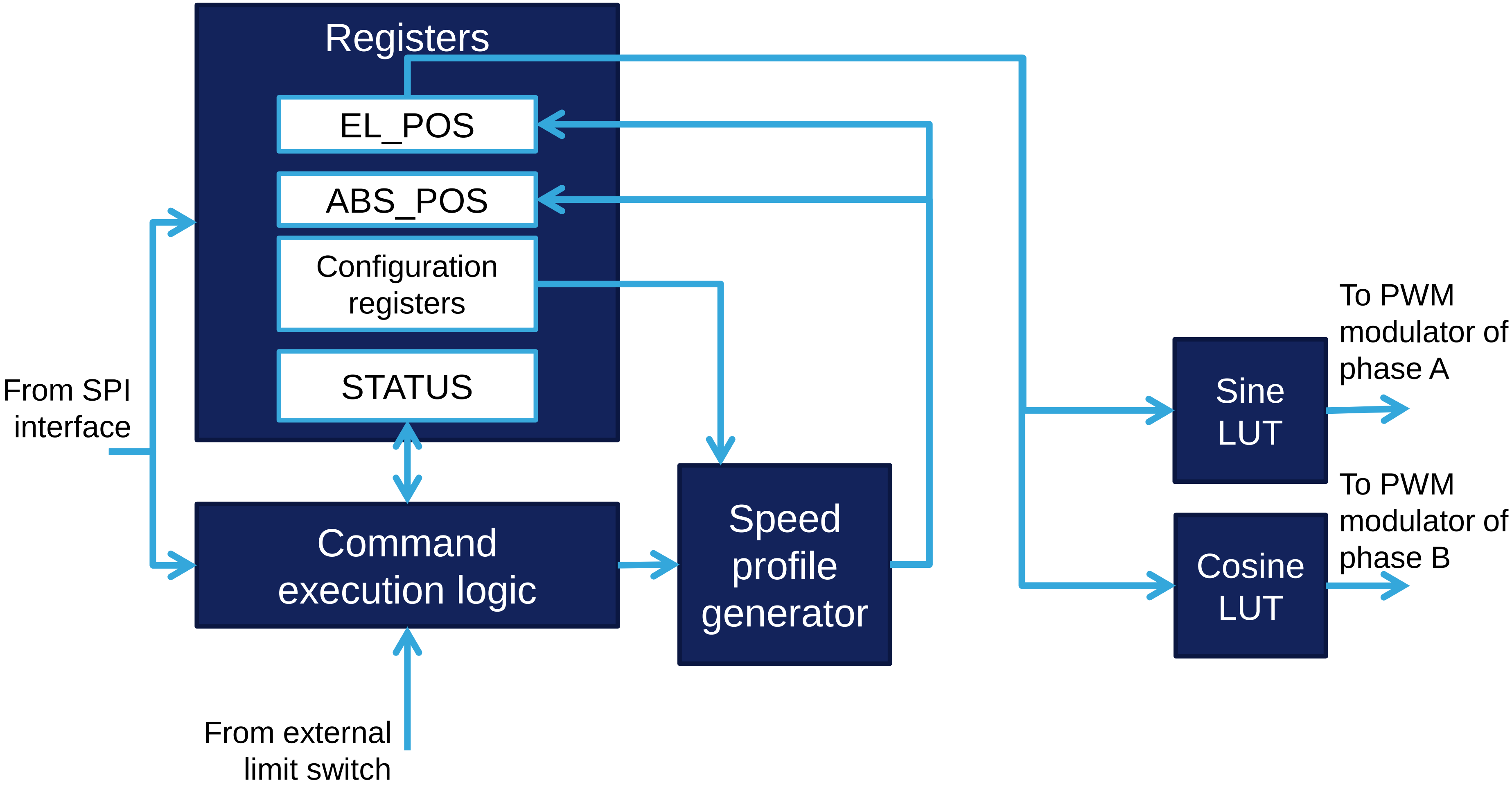
L6470\72, L6480\82 and powerSTEP01

Basic features

The logic core integrated into the L6470\72, L6480\82 and powerSTEP01 ICs provides advanced features for the control of the stepper motor position:

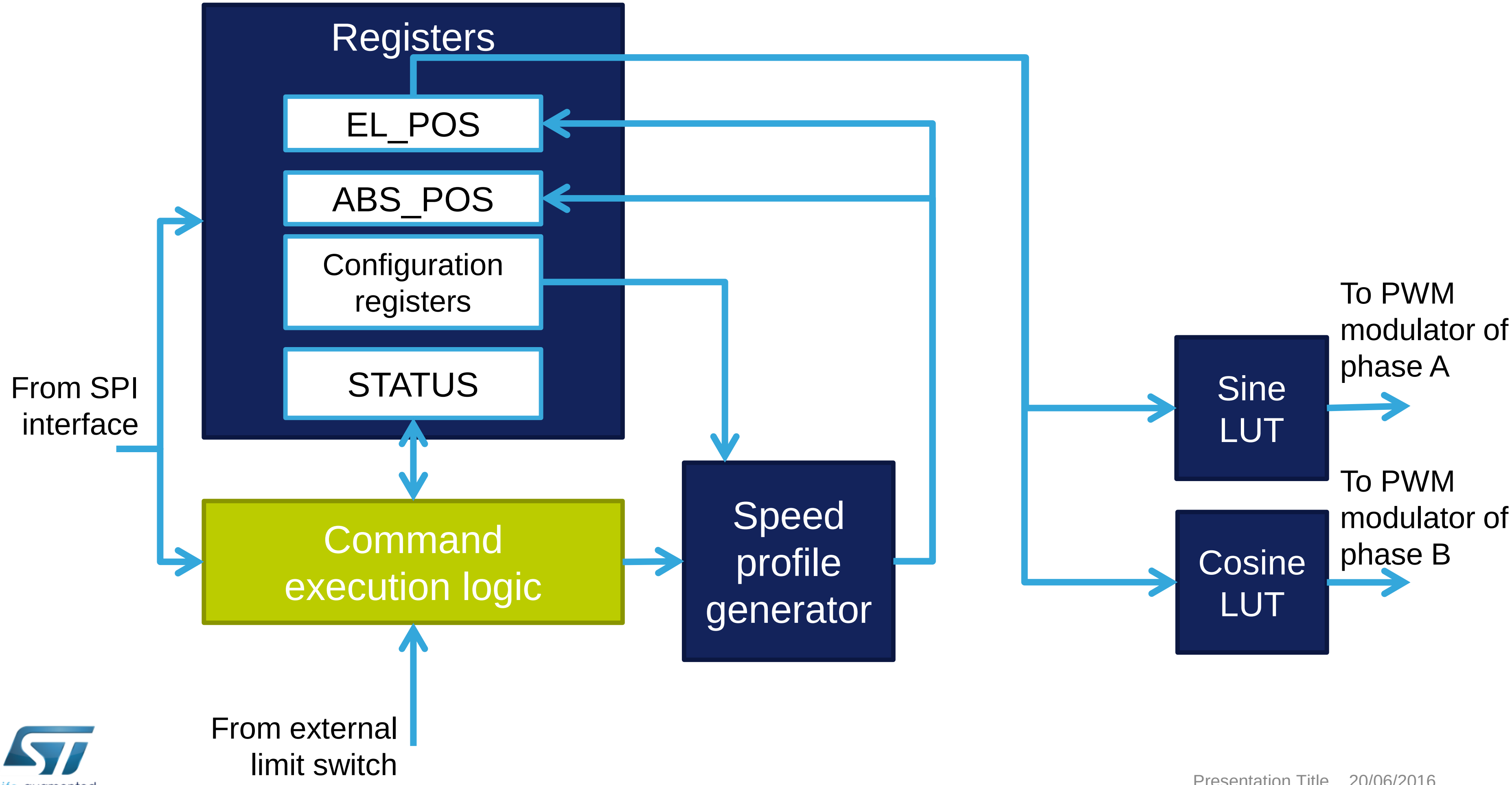
- **Programmable speed profile**
- **Absolute and relative positioning**
- **Speed tracking**
- **External limit switch management**

Basic features



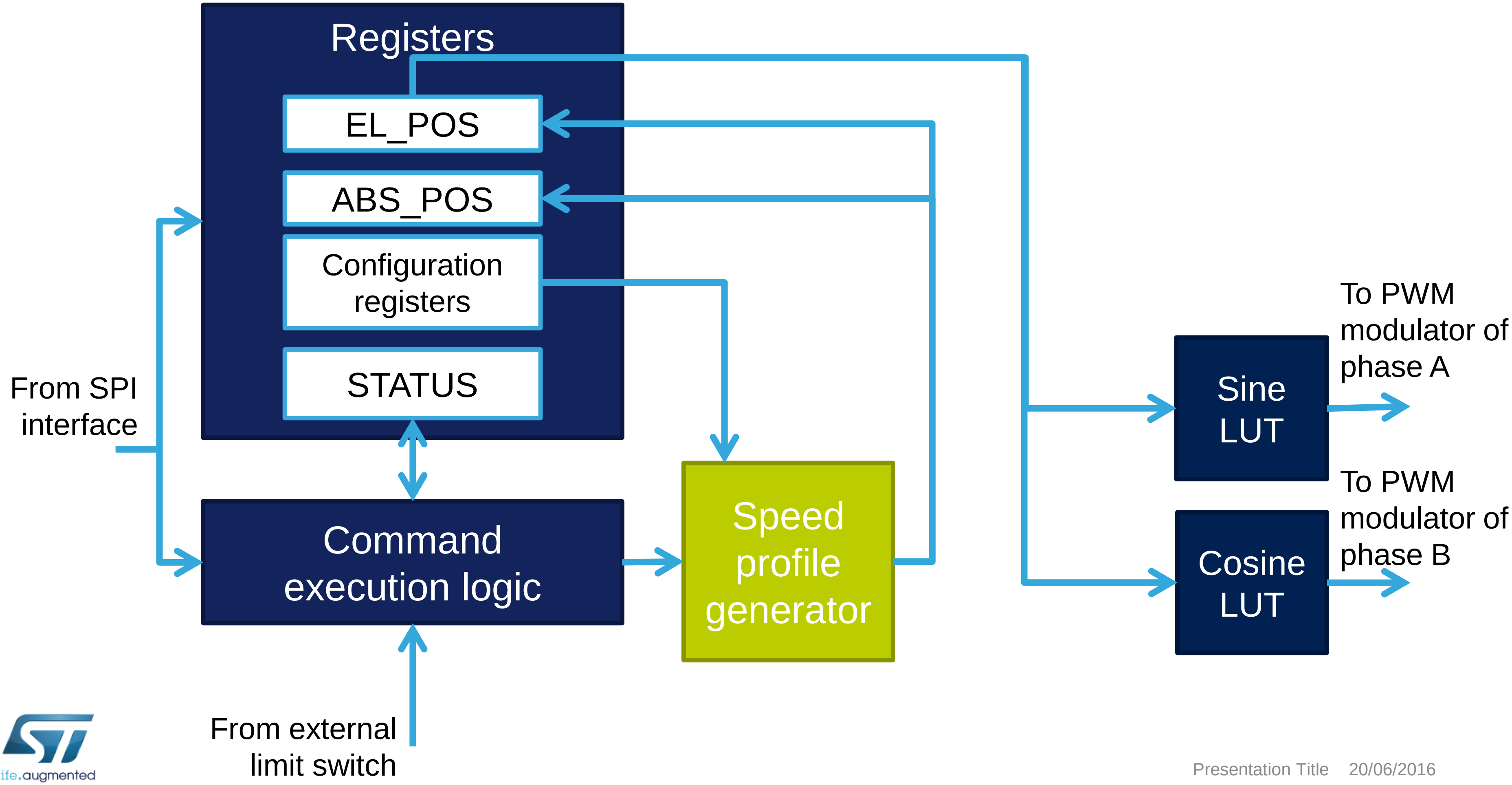
Basic features

The **command execution logic** converts the high-level commands from the SPI into indications for the speed profile generator.



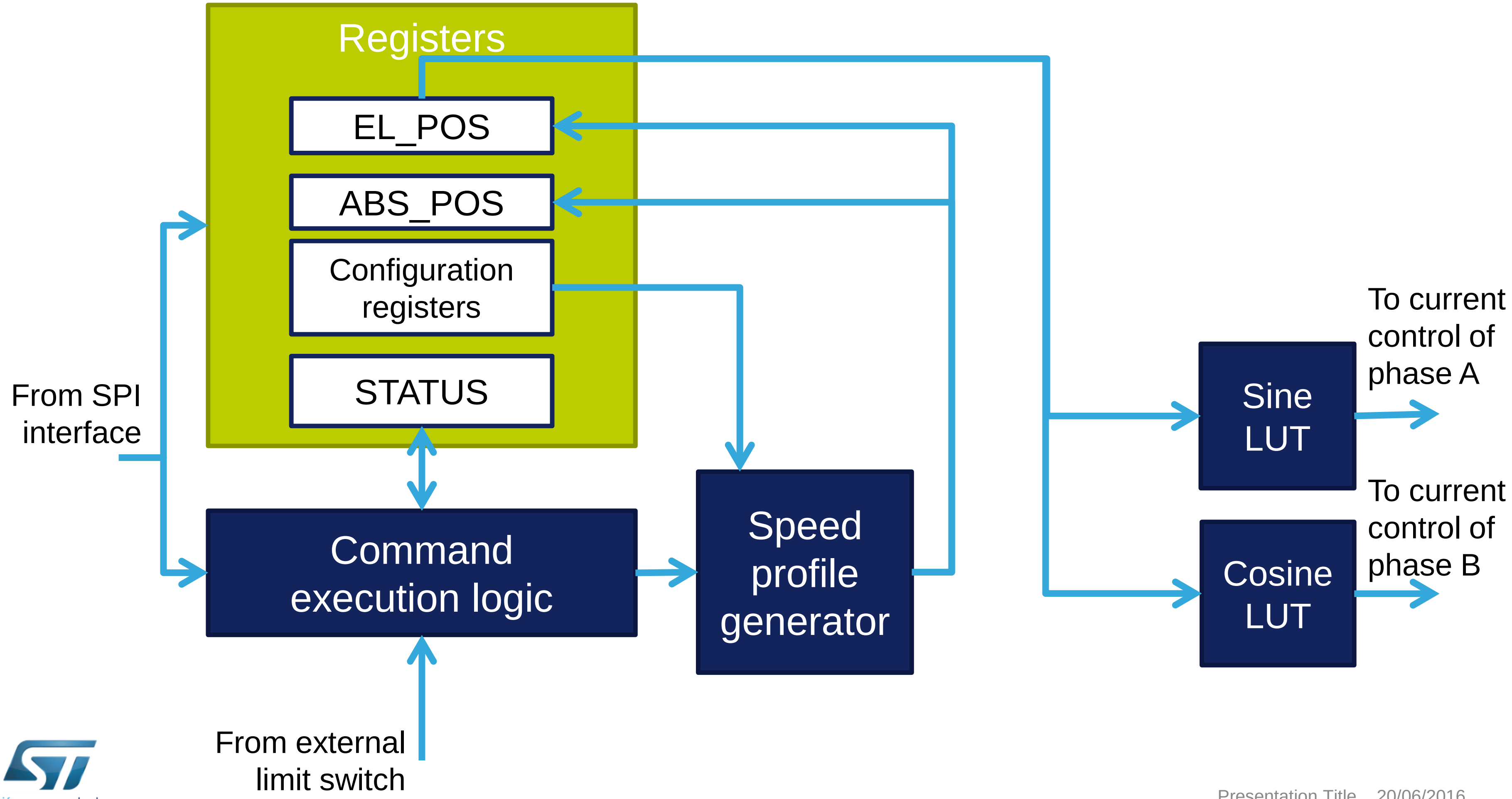
Basic features

The **speed profile generator** generates the step-clock and direction signal according to the indications of the command execution logic and the parameters of the configuration registers.



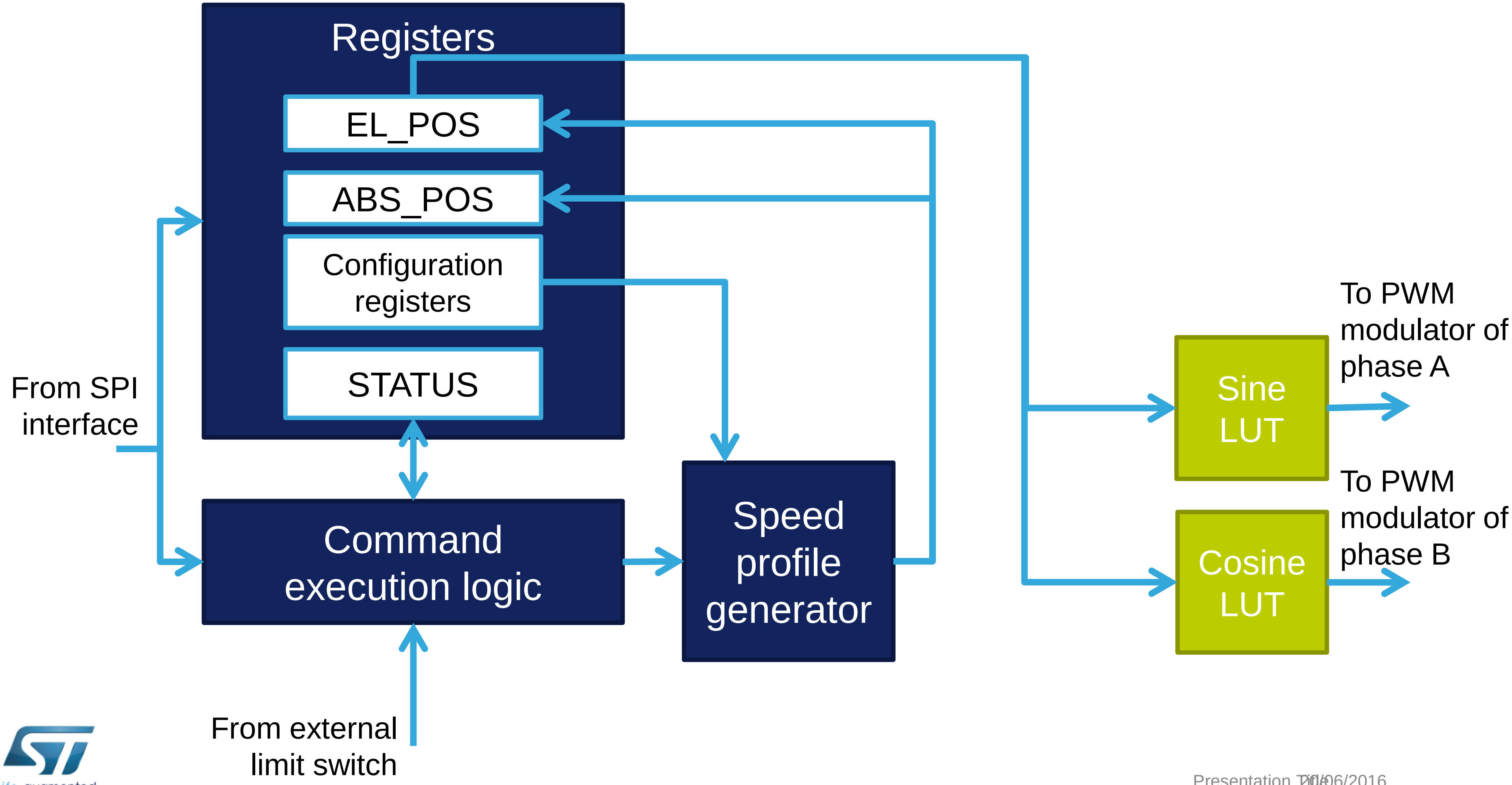
Basic features

The **absolute position** (ABS_POS) and **electrical position** (EL_POS) registers are counters controlled by the speed profile generator.
The **diagnostic register** (STATUS) returns the status of the device.



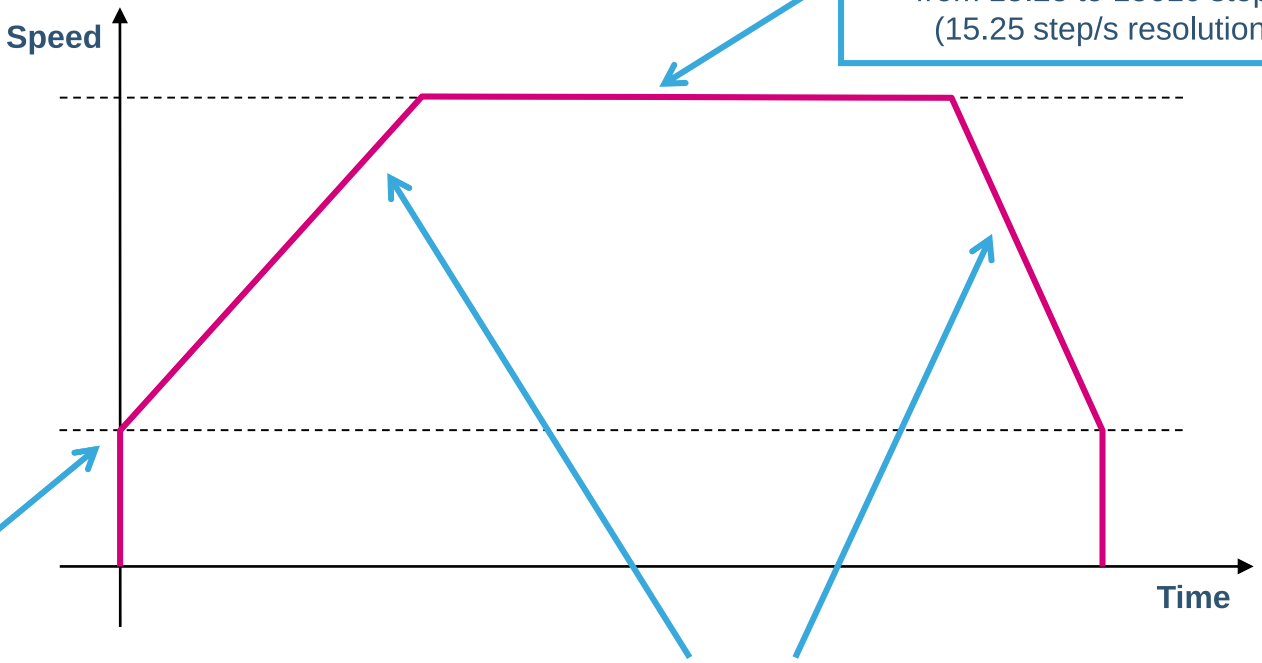
Basic features

The **sine and cosine LUT** are used to generate the full-step, half-step and microstepping driving sequences according to the EL_POS value.



Programmable speed profile

The motion engine allows independent setting of all the speed profile parameters.



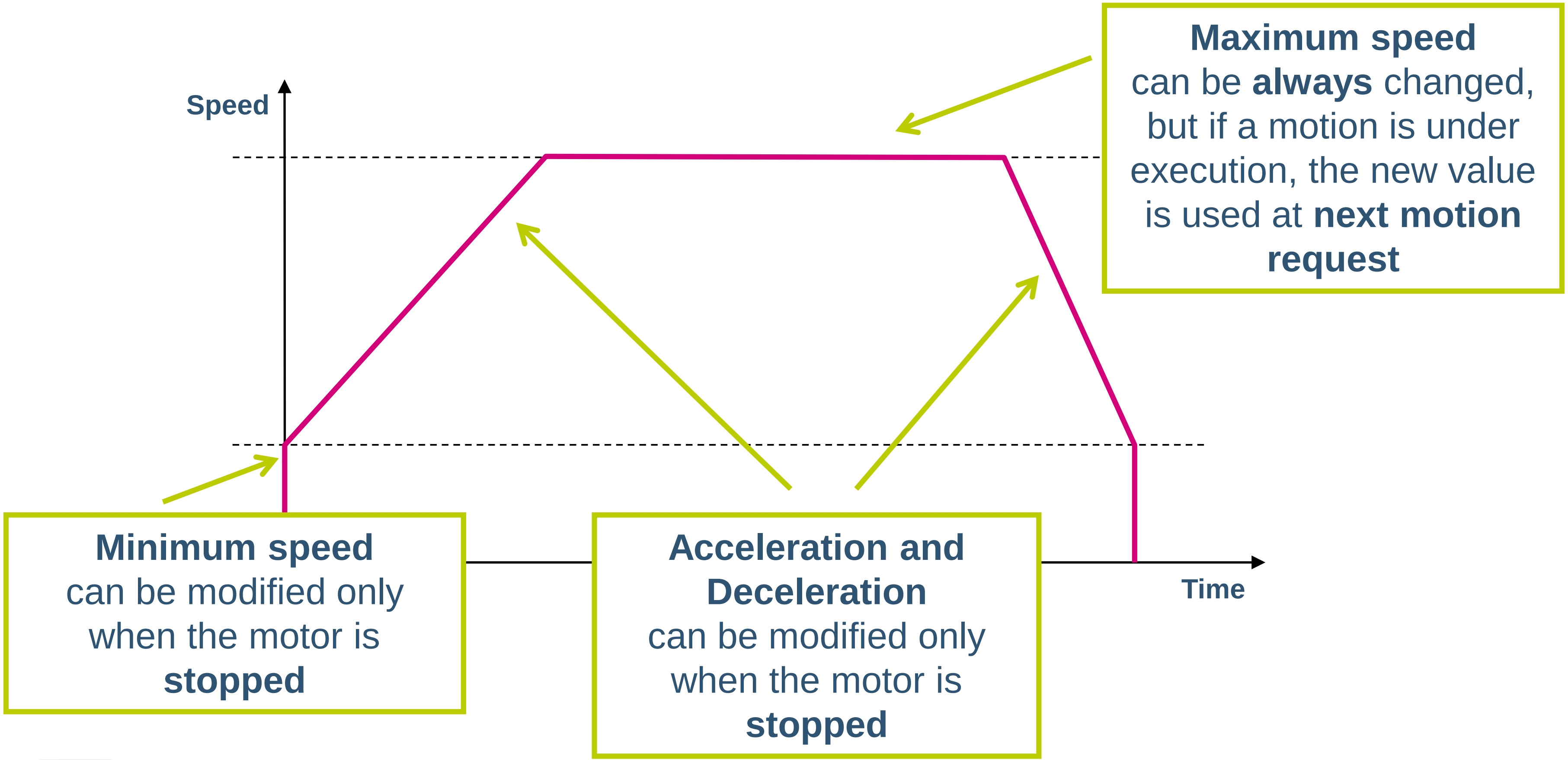
Maximum speed
from 15.25 to 15610 step/s
(15.25 step/s resolution)

Minimum speed
from 0 to 976 step/s
(0.24 step/s resolution)

Acceleration and Deceleration
from 14.55 to 59590 step/s^2
(14.55 step/s^2 resolution)

Programmable speed profile

The parameters of the speed profile can be changed only in specific conditions:



Maximum speed can be **always** changed, but if a motion is under execution, the new value is used at **next motion request**

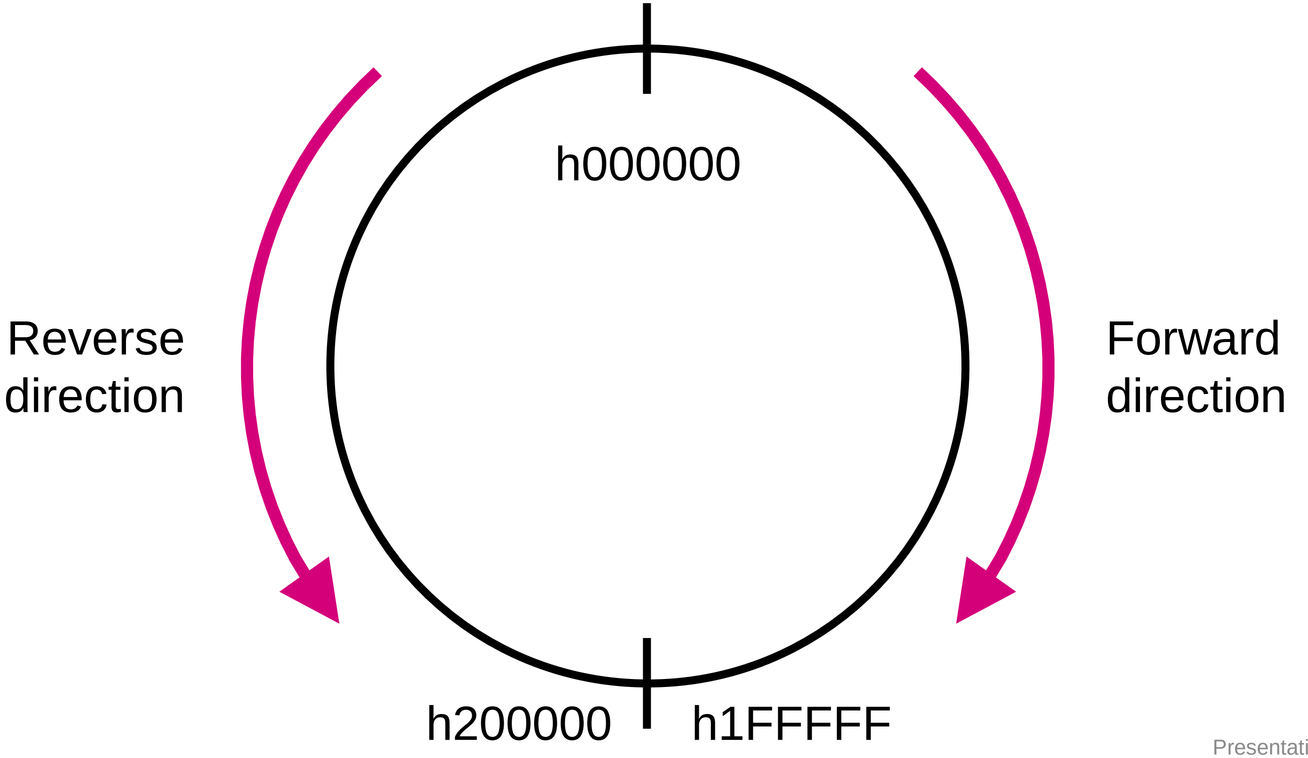
Minimum speed can be modified only when the motor is **stopped**

Acceleration and Deceleration can be modified only when the motor is **stopped**

Positioning commands

The motion engine integrates an **absolute position register** which indicates the **mechanical position** of the motor.

The absolute position register is a **22-bit up/down counter** connected to the step-clock signal generated by the speed profile generator.



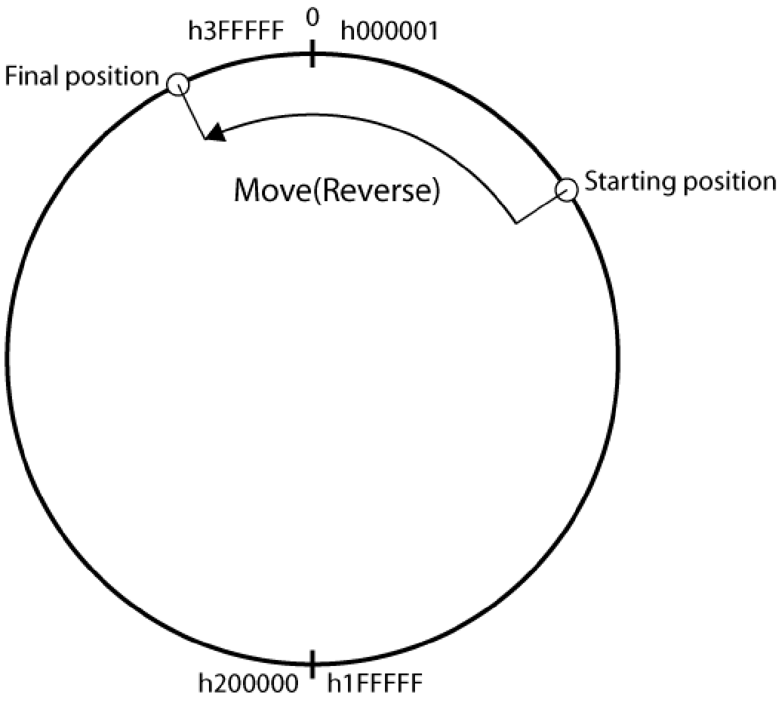
Positioning commands

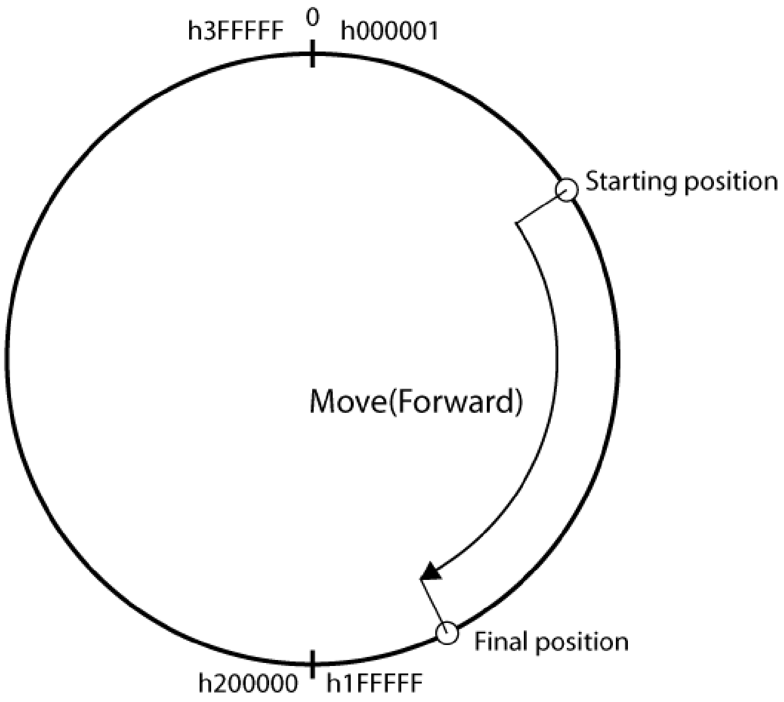
Name	Description	Notes
Move	Perform the target number of microsteps in the requested direction.	Only in stop condition.
GoTo	Reach the target absolute position (ABS_POS register) using the shortest path.	Only when not BUSY.
GoTo_DIR	Reach the target absolute position (ABS_POS register) running in the requested direction.	Only when not BUSY.
GoHome	Reach the home position (all zeroes) using the shortest path.	Only when not BUSY.
GoMark	Reach the position stored into the MARK register using the shortest path.	Only when not BUSY.

Positioning commands

Move

The Move command makes the motor perform the target number of steps in the indicated direction.





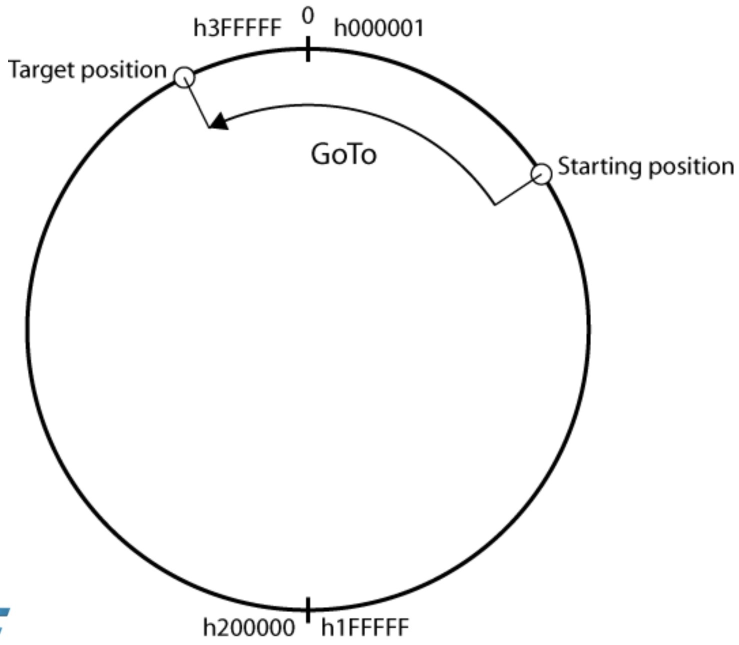
Positioning commands

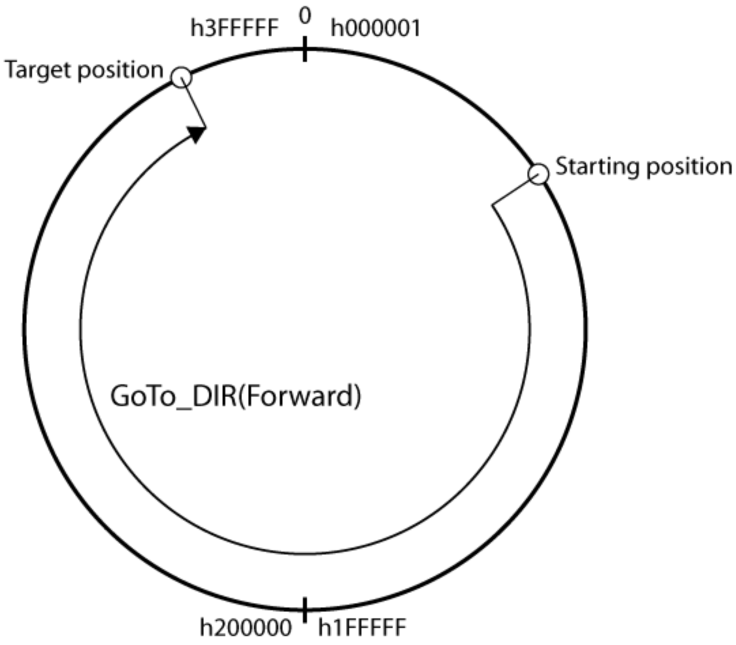
GoTo

The GoTo command makes the motor reach the target ABS_POS value using the shortest path.

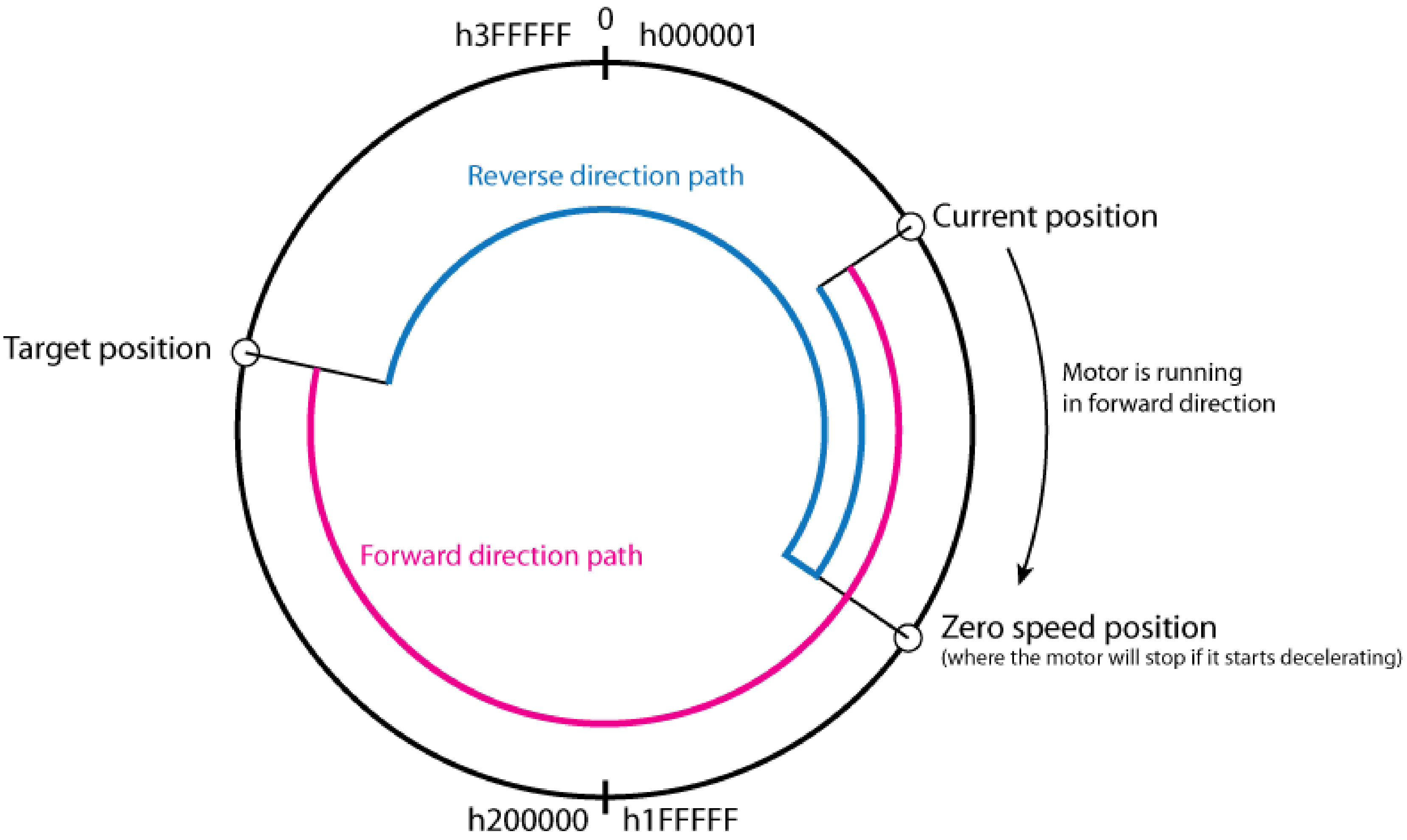
GoTo_DIR

The GoTo_DIR command makes the motor reach the target ABS_POS value using the indicated direction.





Positioning commands



Reverse direction path > Forward direction path then forward direction is used.

Positioning commands

GoHome

The GoHome command makes the motor reach the zero value of the ABS_POS register using the shortest path.
It is equivalent to the GoTo(0) command.

GoMark

The GoMark command makes the motor reach the value of the ABS_POS register stored into the MARK register using the shortest path.
It is equivalent to the GoTo(MARK) command.

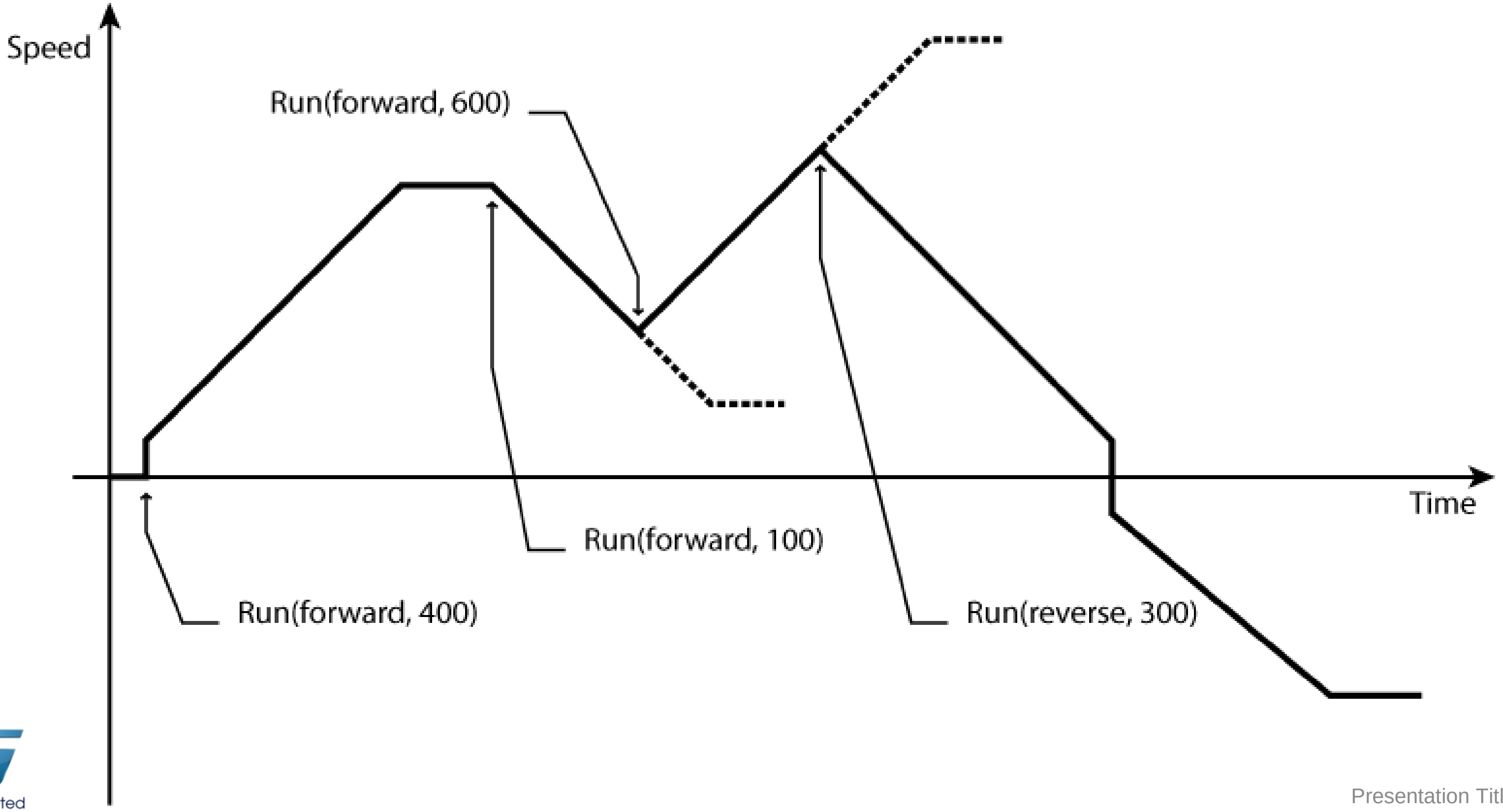
Speed tracking and stop commands

Name	Description	Notes
Run	Reach the target speed in the requested direction.	Always accepted and immediately executed (if present, the previous command is aborted)
SoftStop	Stop the motor in accordance to the programmed speed profile.	Always accepted and immediately executed (if present, the previous command is aborted)
HardStop	Immediately stop the motor (infinite deceleration).	Always accepted and immediately executed (if present, the previous command is aborted)
SoftHiZ	Stop the motor in accordance to the programmed speed profile and then disable the power bridges.	Always accepted and immediately executed (if present, the previous command is aborted)
HardHiz	Immediately disable the power bridges.	Always accepted and immediately executed (if present, the previous command is aborted)

Speed tracking and stop commands

Run

The Run command makes the motor reach the target speed following the programmed speed profile boundaries (acceleration and deceleration).



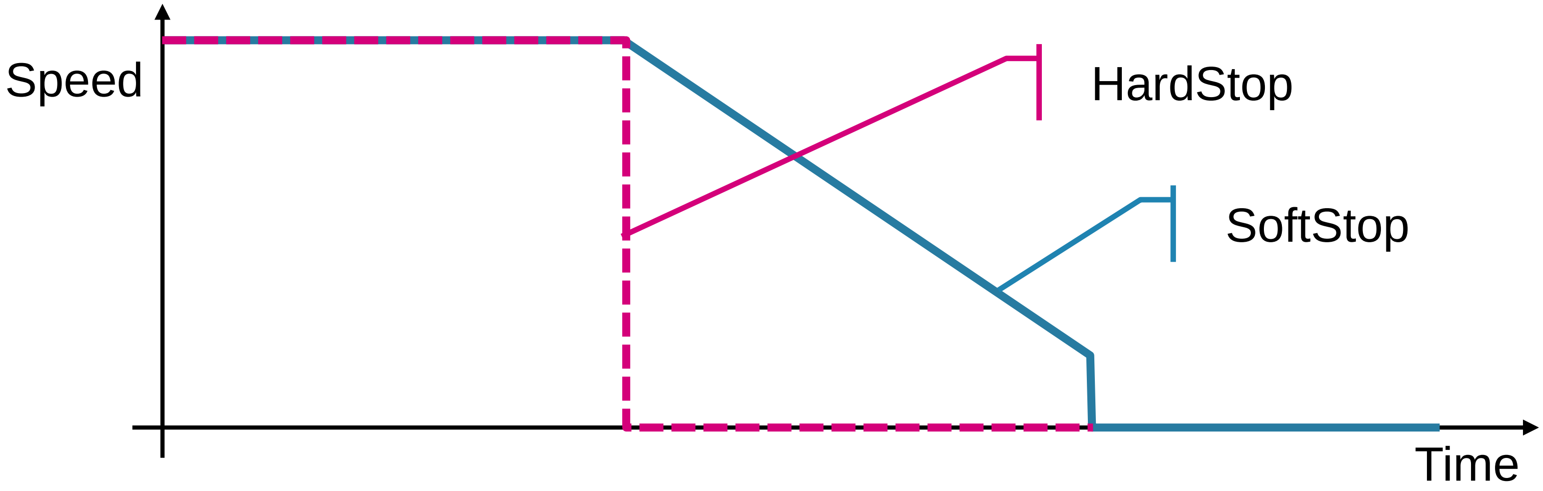
Speed tracking and stop commands

SoftStop

The SoftStop command makes the motor decelerate down to zero speed.

HardStop

The HardStop command immediately stops the motor.



Speed tracking and stop commands

SoftHiZ

The SoftHiz command executes a SoftStop and then disables the power bridges.

HardHiZ

The HardHiZ command executes a HardStop and then disables the power bridges.

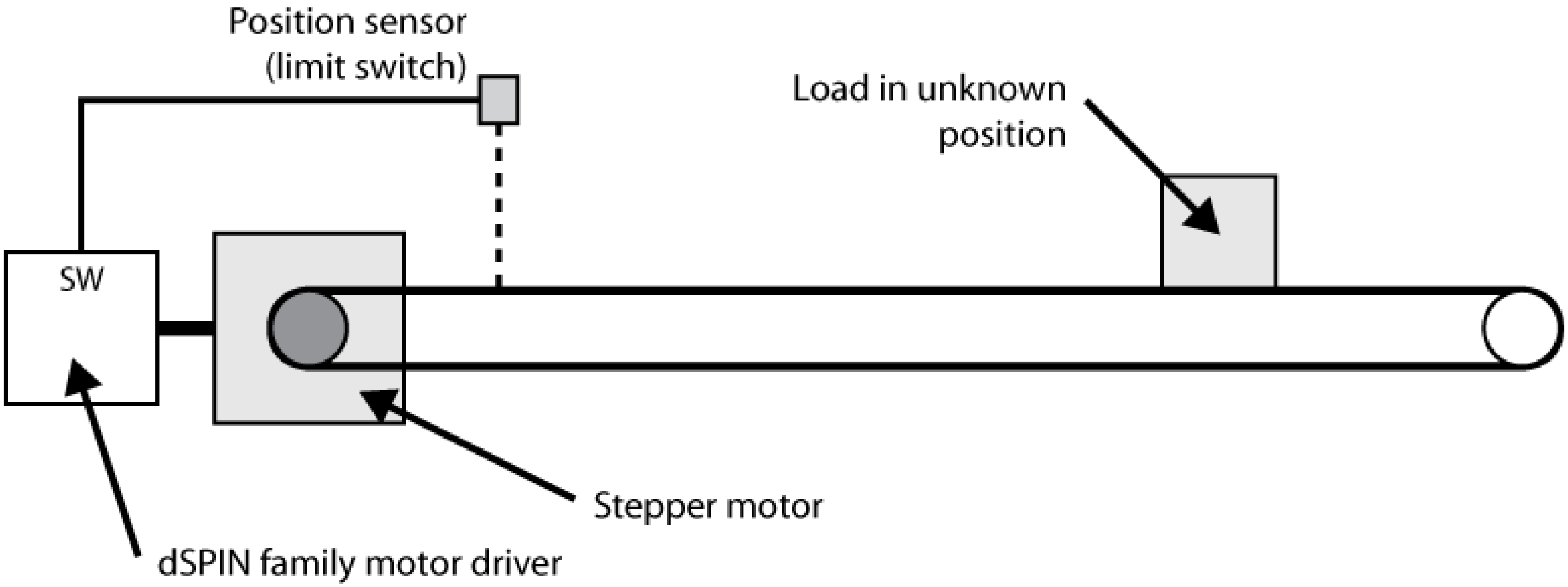
External limit switch management

The motion engine can manage the presence of an external limit switch (mechanical position sensor) providing dedicated motion commands.

Name	Description	Notes
GoUntil	Reach the target speed in the requested direction and stop when SW input is forced low (falling edge).	Always accepted and immediately executed (if present, the previous command is aborted)
ReleaseSW	Run the motor at low speed in the requested direction and stop when SW input is forced high (rising edge).	Always accepted and immediately executed (if present, the previous command is aborted)

External limit switch management

A practical example...

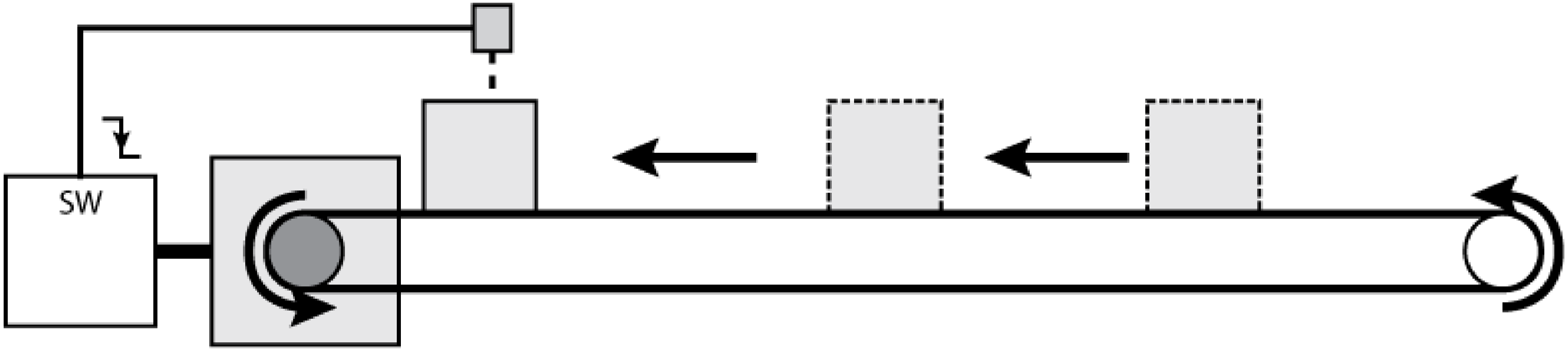


External limit switch management

The **GoUntil** command is executed and the load reaches the limit switch at programmed speed.

When a falling edge on the SW is detected, the motor is stopped with a SoftStop and one of this two actions can be automatically executed:

- The absolute position is reset to zero
- The absolute position is stored into the a service register (MARK)

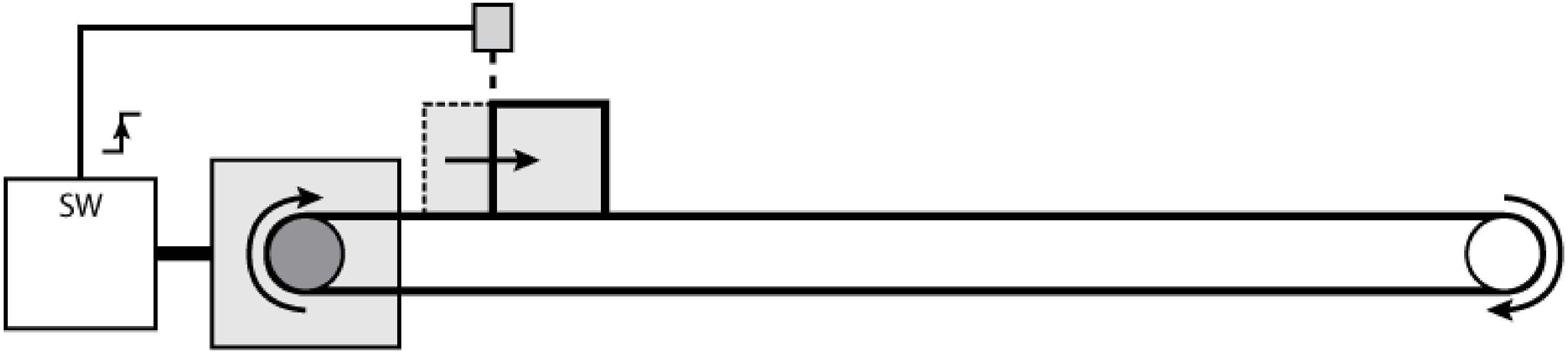


External limit switch management

The **ReleaseSW** command is executed and the load is moved at low speed and positioned exactly on the threshold point of the limit switch.

When a rising edge on the SW is detected, the motor is stopped with a HardStop and one of this two actions can be automatically executed:

- The absolute position is reset to zero
- The absolute position is stored into the a service register (MARK)



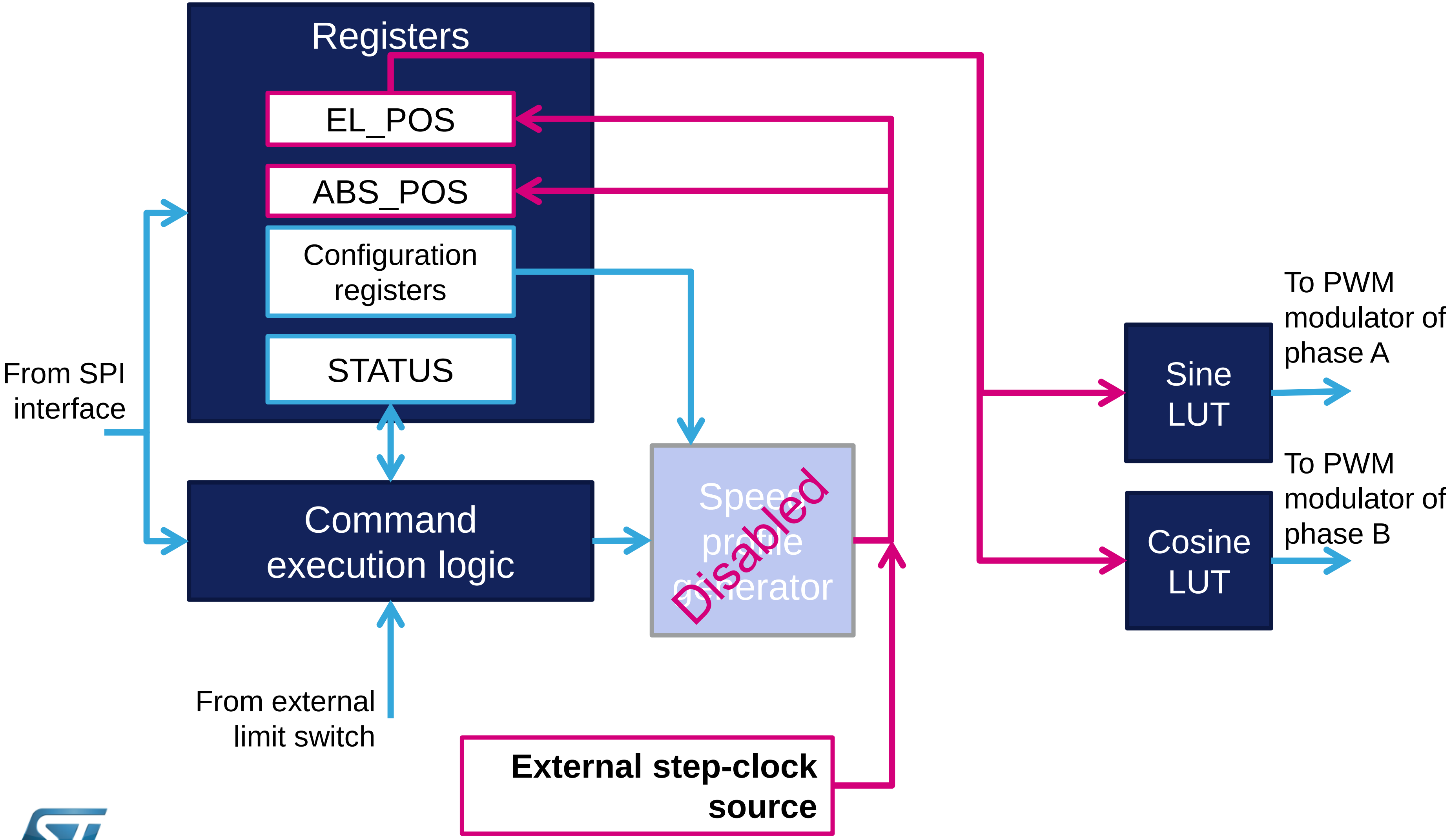
Step clock mode

The motion engine can be forced to use an external step clock.

Name	Description	Notes
StepClock	Switch the device in step-clock mode imposing the direction.	Only in stop condition.

When the step-clock mode is enabled, the motion engine considers the motor as **stopped** and the system is considered at **zero speed**, so all the respective parameters/limits depending on the motor status or speed are applied.

Step clock mode



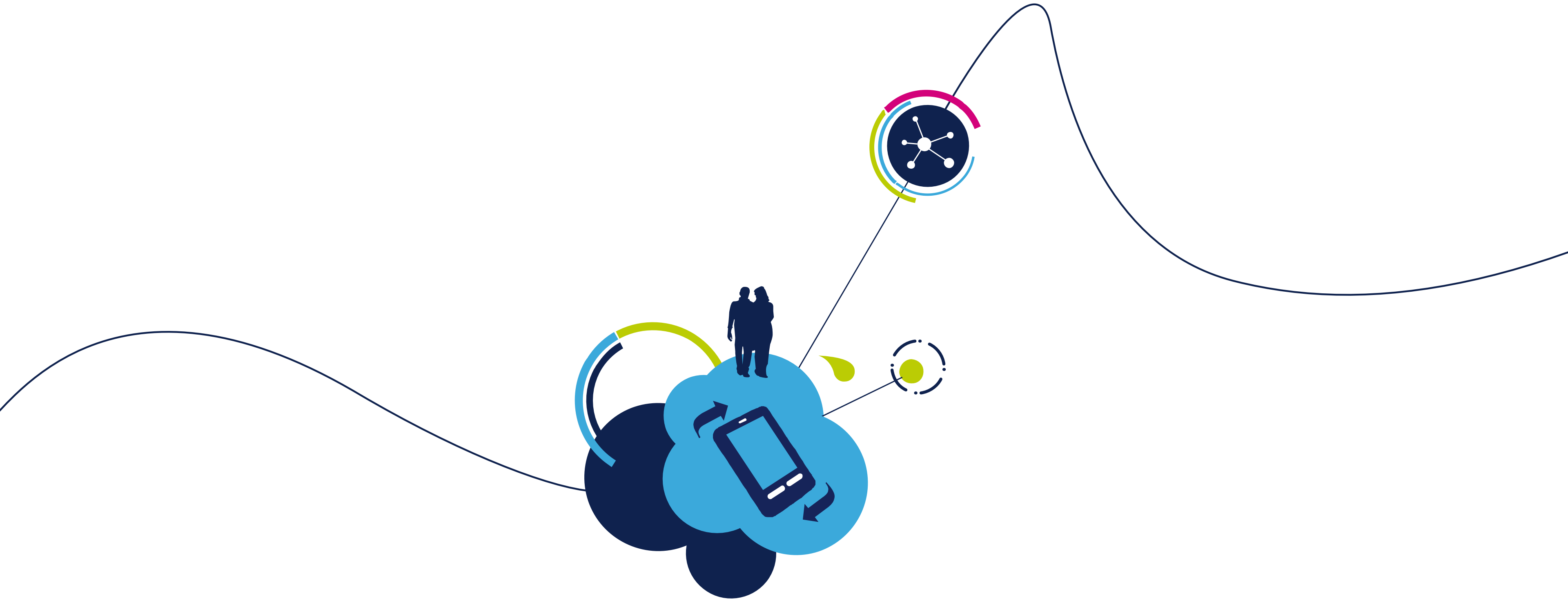
Configuration and diagnostic commands

In order to set up the motion engine and the other functions of the device, a set of commands for the reading and the writing of the internal registers is provided.
A command checking the device diagnostic register (STATUS) is also present.

Name	Description	Notes
SetParam	Write the target register	Executed only if the target register can be written in the current condition.
GetParam	Read the target register	Always accepted and immediately executed.
GetStatus	Read the diagnostic register (STATUS) and release the failure condition	Always accepted and immediately executed.
ResetPos	Reset the absolute position to zero	Executed only if the absolute position register can be written in the current condition.
ResetDevice	Reset all the parameters to the default value (power stage outputs are forced in high impedance status)	Always accepted and immediately executed (HANDLE WITH CARE).

What you can\cannot do

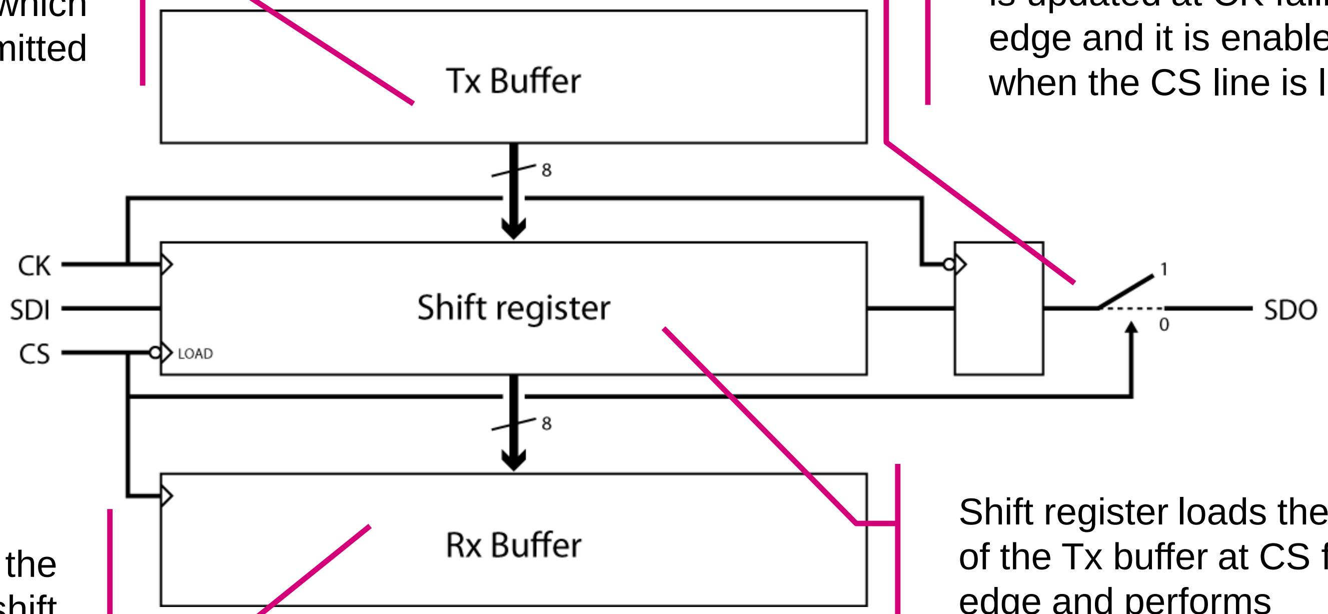
When…	You can…
Power stage is disabled	• Read and write all the registers • Perform any motion command
Power stage is enabled and the motor is stopped	• Read all registers • Write most of the registers but the critical ones (IC configuration, current control setup, step mode, electrical position) • Perform any motion command
Power stage is enabled, the motor is moving and the BUSY line is high (no command under execution)	• Read all registers • Write some registers • Perform absolute positioning, speed tracking and stop commands
Power stage is enabled, the motor is moving and the BUSY line is low (command under execution)	• Read all registers. • Write some registers • Perform speed tracking and stop commands



SPI interface

SPI interface

The motion engine commands and the configuration parameters are sent to the devices through an 8-bit SPI interface.



Tx buffer stores the response byte which must be transmitted

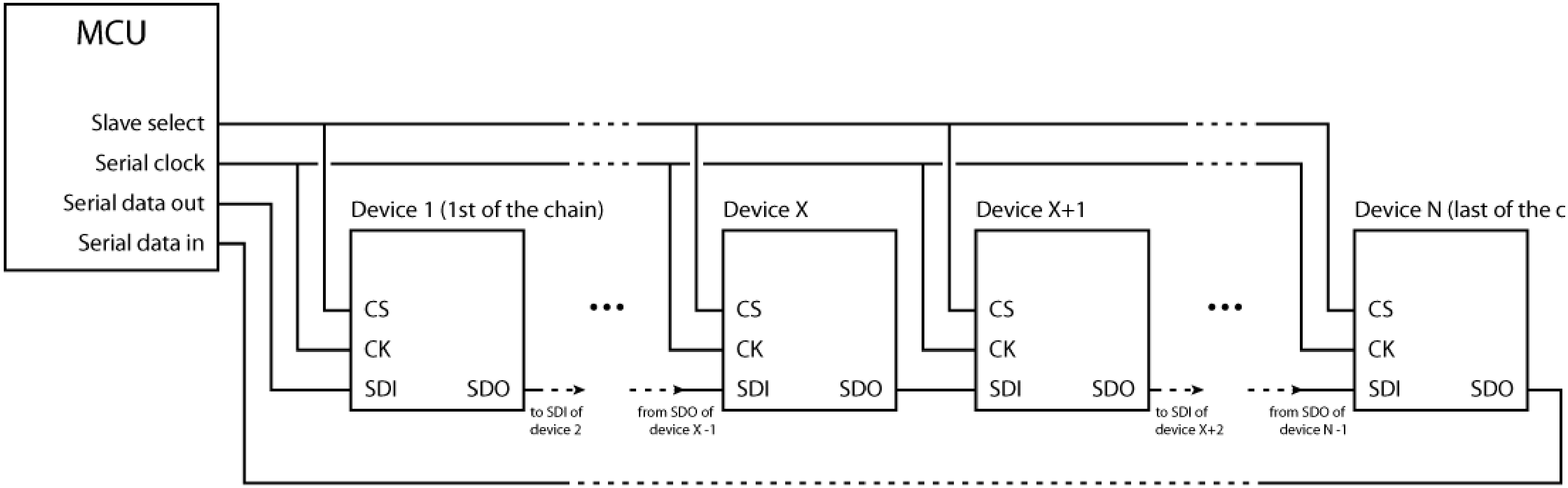
Serial output of the device is updated at CK falling edge and it is enabled only when the CS line is low

Rx buffer loads the byte of the shift register at CS rising edge

Shift register loads the byte of the Tx buffer at CS falling edge and performs the bit-by-bit transmission (parallel to serial and serial to parallel converter).

SPI Interface and daisy-chaining

The SPI of the motion engine is compliant with the daisy-chain configuration.

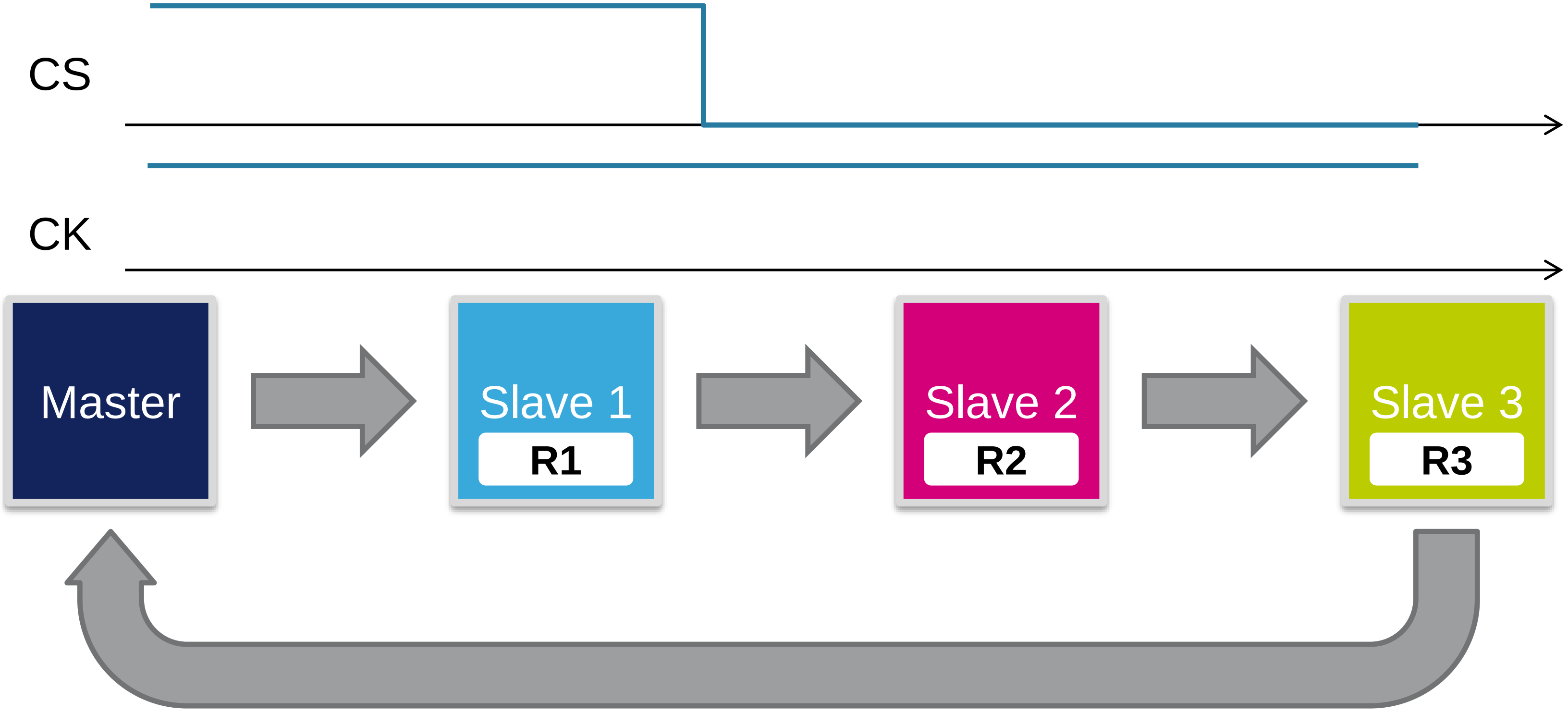


Using this configuration, a **single SPI master** can drive **multiple SPI slaves**.

SPI Interface and daisy-chaining

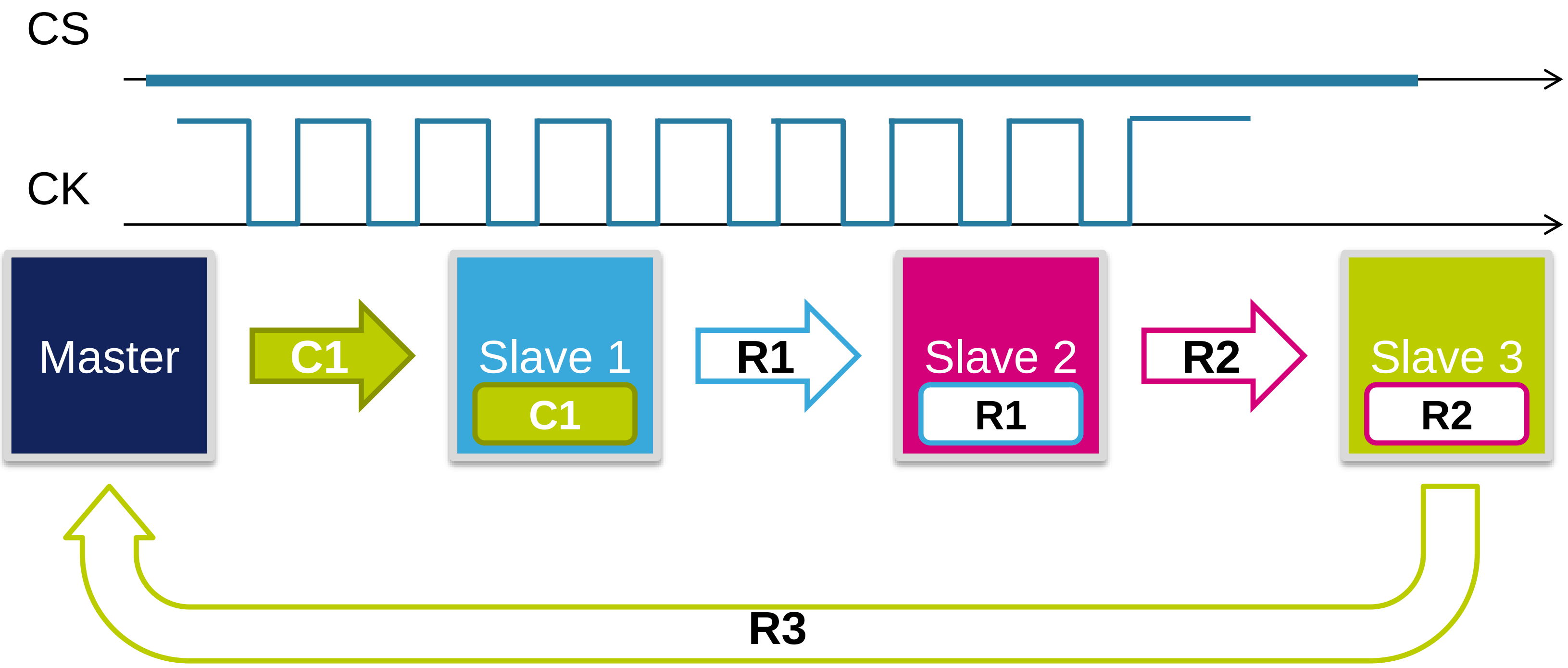
Driving sequence example with 3 slaves

Step 1: The master starts the communication cycle forcing the CS line low. The slaves load into the shift register of the SPI interface the **response bytes**.



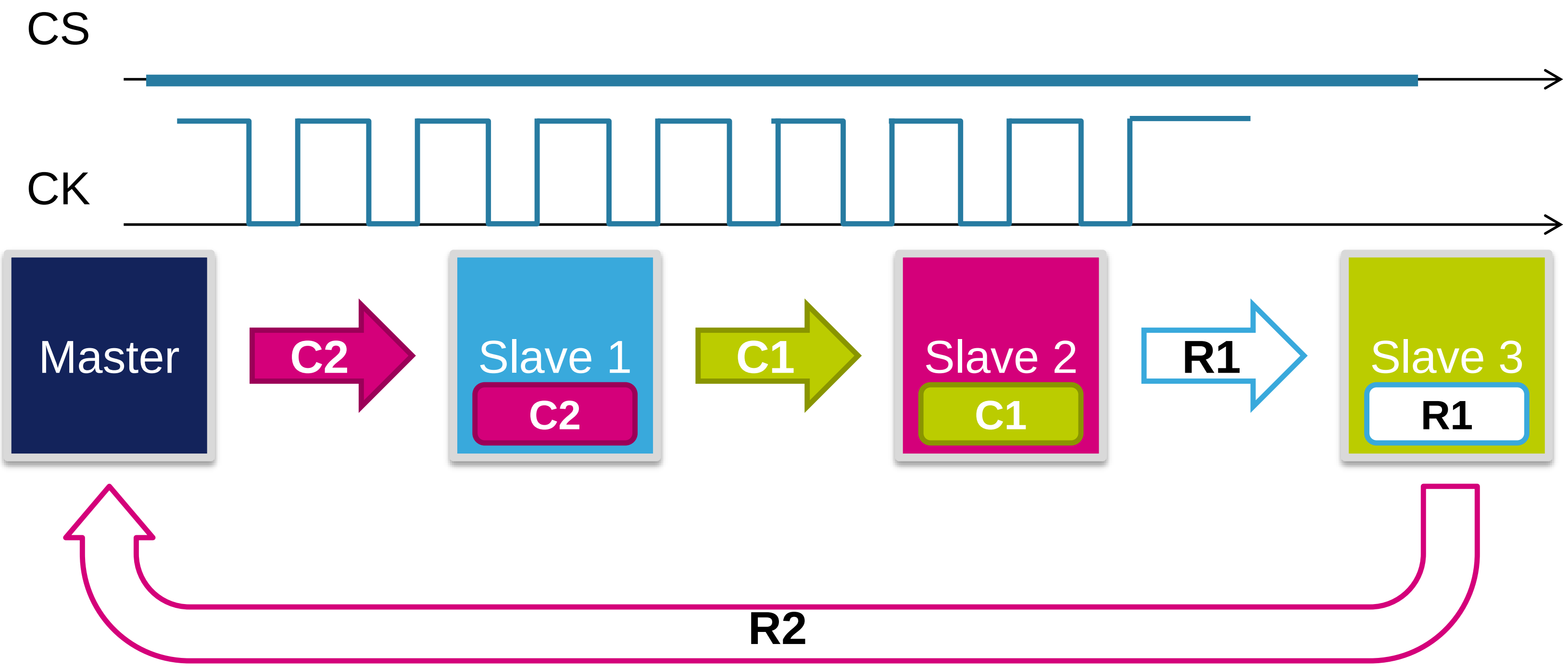
SPI Interface and daisy-chaining

Step 2: The master transmits the **first command byte**. At the same time, each slave transmits the byte into the SPI shift register to the next device of the chain (right side) and stores into the shift register the byte from the previous one (left side).
At the end of byte transmission, the master receives the **response byte of the slave 3**.
The CS line is kept low and it is not raised at the end of byte transmission.



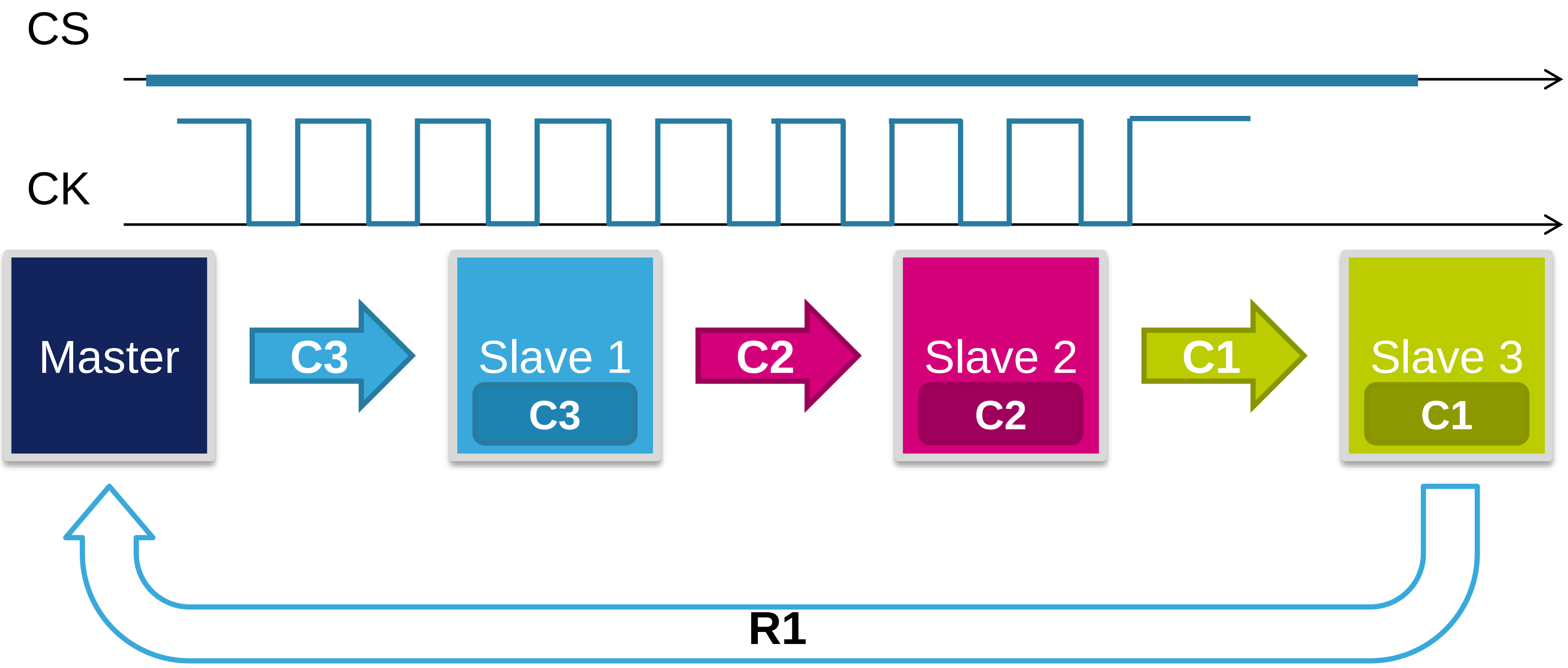
SPI Interface and daisy-chaining

Step 3: The master transmits the **second command byte**. At the same time, each slave transmits the byte into the SPI shift register to the next device of the chain (right side) and stores into the shift register the byte from the previous one (left side).
At the end of byte transmission, the master receives the **response byte of the slave 2**.
The CS line is kept low and it is not raised at the end of byte transmission.



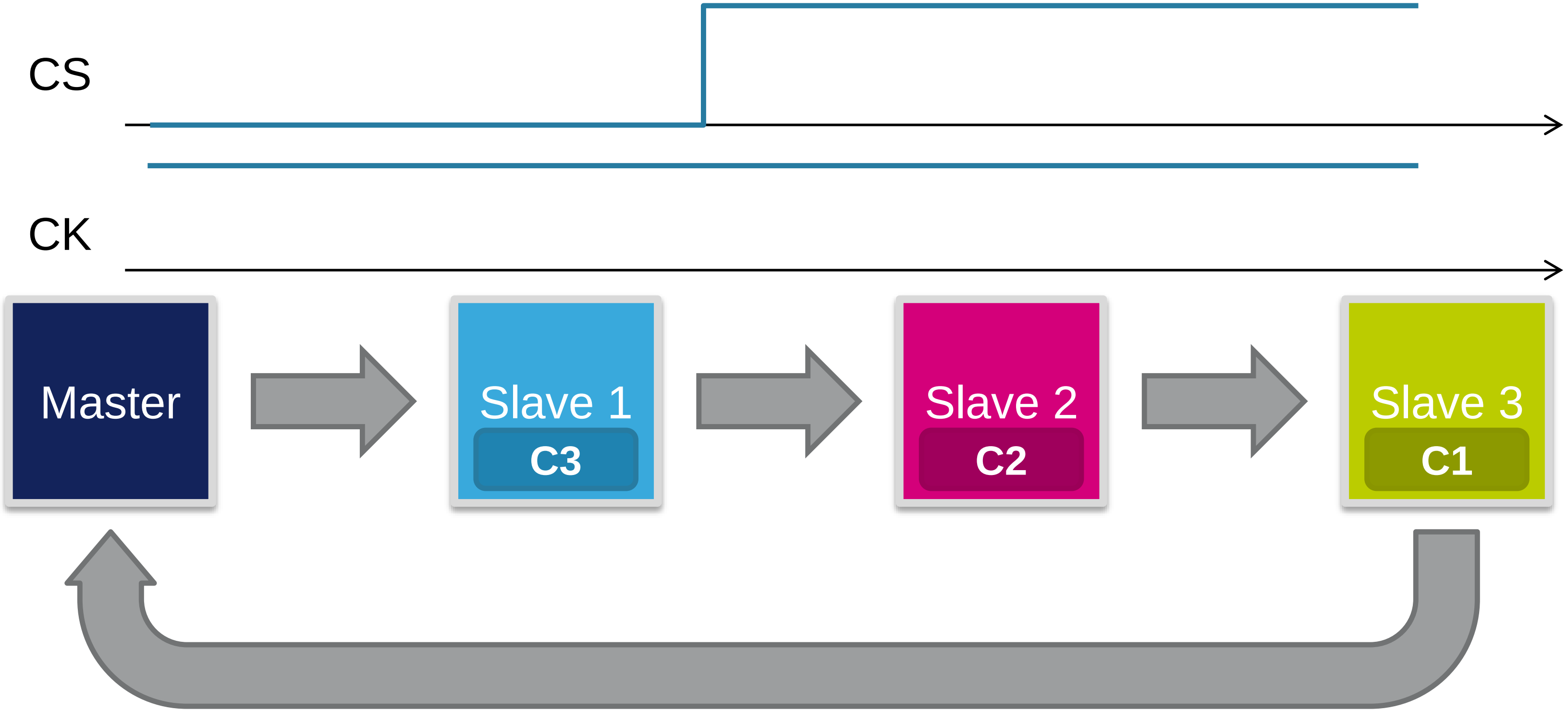
SPI Interface and daisy-chaining

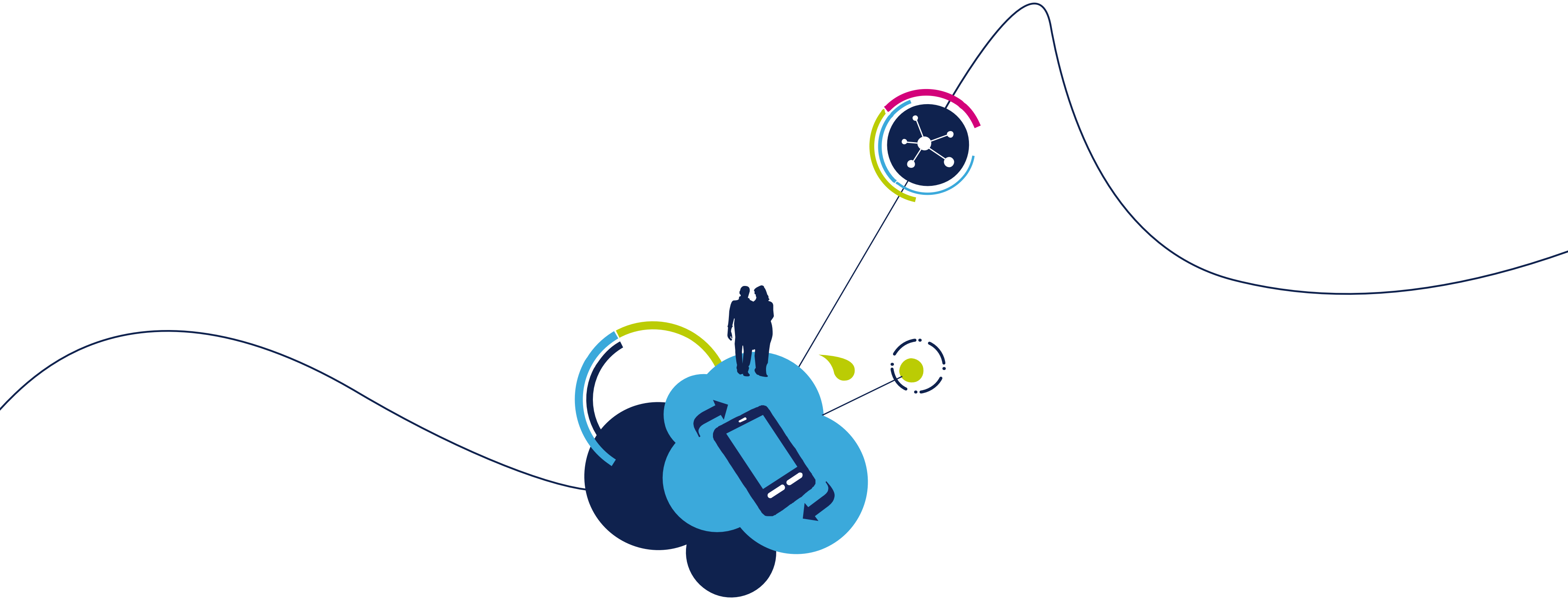
Step 4: The master transmits the **last command byte**. At the same time, each slave transmits the byte into the SPI shift register to the next device of the chain (right side) and stores into the shift register the byte from the previous one (left side).
At the end of byte transmission, the master receives the **response byte of the slave 1**.
The CS line is kept low and it is not raised at the end of byte transmission.



SPI Interface and daisy-chaining

Final step: The master stops the communication cycle forcing the CS line high. The slaves acquire the byte stored into the shift register of the SPI interface and decode it as a command byte.





Communication protocol

General protocol description

- All **commands** are composed by a **single byte**.
- After the command byte, **an argument could be needed**.
- The **argument** length can vary from **1 to 3 bytes**.
- A new command can be sent to the device only when the argument of the previous command is completed.
- **By default, the device returns an all zeroes response** for any received byte.
- When a **GetParam** or a **GetStatus** command is received, the **response bytes represent the related register value**.
- The **response** length can vary from **1 to 3 bytes**.
- The logic acquires and **executes the received byte** only when the **CS line is forced high**.

SetParam command

Transmission order

Command byte

7	6	5	4	3	2	1	0
0	**0**	**0**	**ADDR[4]**	**ADDR[3]**	**ADDR[2]**	**ADDR[1]**	**ADDR[0]**

Argument byte/s

	7	6	5	4	3	2	1	0
(1)	**BYTE 2**							
(2)	**BYTE 1**							
	BYTE 0							

The **SetParam** command byte is composed of the target register address with a 000 header.

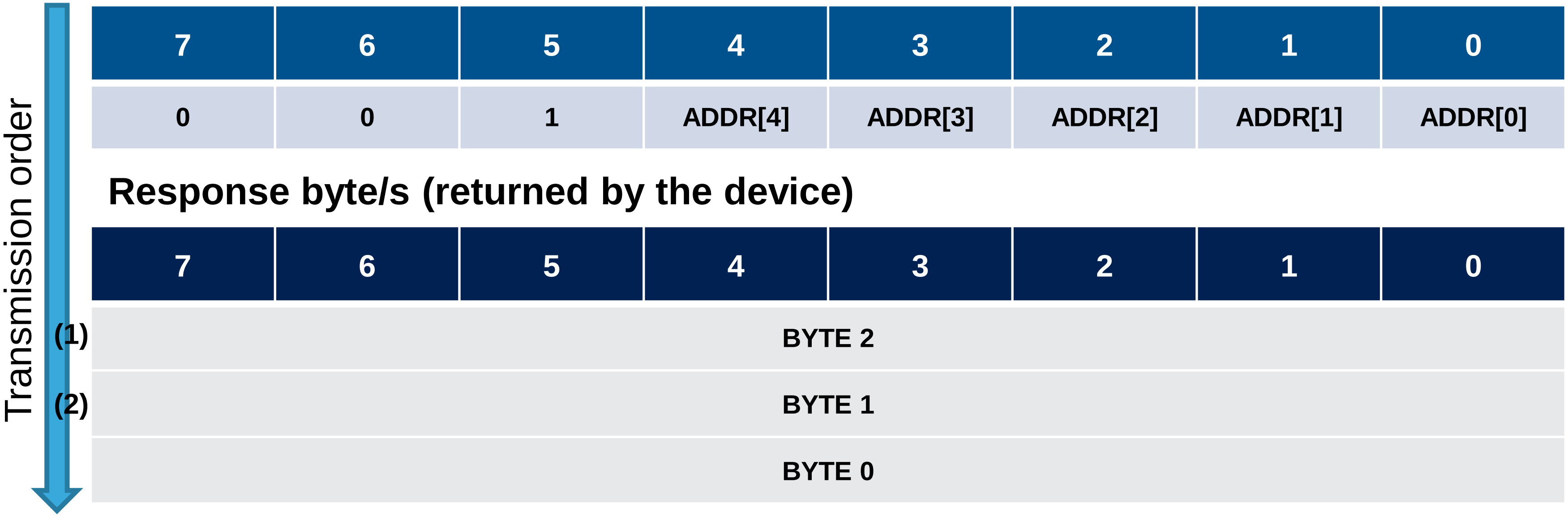
According to the target register length, the command has to be followed by a 1-, 2- or 3-byte argument: it is the value that has to be written into the target register (MSByte first).

Some argument bits could be ignored according to the register structure.

(1) Skip BYTE2 transmission if the length of the register is lower than 3 bytes

(2) Skip BYTE1 transmission if the length of the register is lower than 2 bytes

GetParam command



Command byte

7	6	5	4	3	2	1	0
0	0	1	ADDR[4]	ADDR[3]	ADDR[2]	ADDR[1]	ADDR[0]

Response byte/s (returned by the device)

	7	6	5	4	3	2	1	0
(1)	BYTE 2							
(2)	BYTE 1							
	BYTE 0							

The **GetParam** command byte is composed of the target register address with a 001 header.

The command needs no argument.

According to the target register length the command returns a 1-, 2- or 3-byte response: it is the current value of target register (MSByte first).

(1) The BYTE2 transmission is skipped if the length of the register is lower than 3 bytes

(2) The BYTE1 transmission is skipped if the length of the register is lower than 2 bytes

GetStatus command

Transmission order

Command byte

7	6	5	4	3	2	1	0
1	1	0	1	0	0	0	0

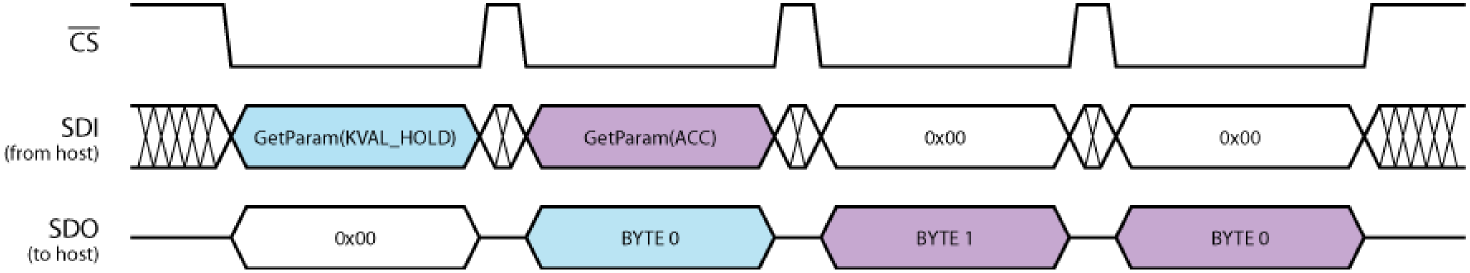
Response byte/s (returned by the device)

<table>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td colspan="8">STATUS MSB</td></tr>
<tr><td colspan="8">STATUS LSB</td></tr>
</table>

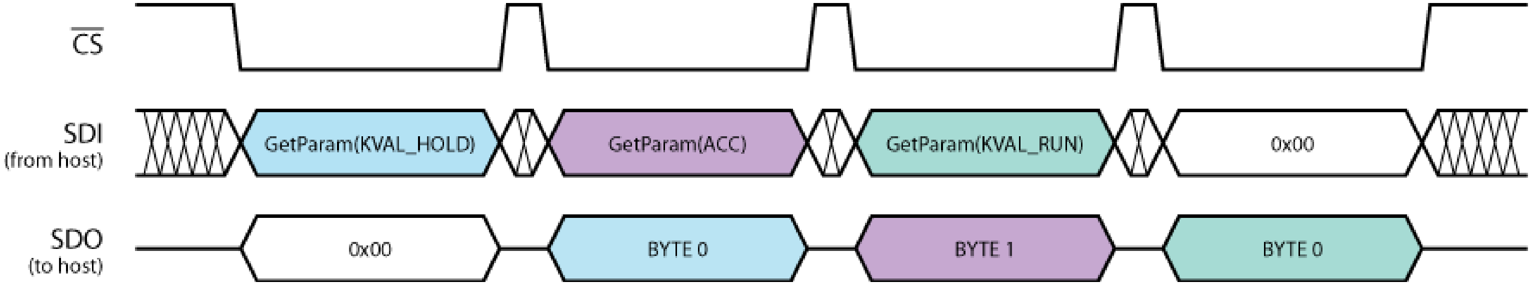
The **GetStatus** command byte is 0xD0.
The command needs no argument.
The command **returns a 2-byte response** containing the current value of the STATUS register (MSByte first).

GetParam and GetStatus sequences

If a new **GetParam** or **GetStatus** command is sent to the device before the previous response is completed, the new response replaces the previous one.



GetParam sequence without response interruption



GetParam sequence with response interruption

ResetPos command

Command byte

7	6	5	4	3	2	1	0
1	1	0	1	1	0	0	0

The **ResetPos** command byte is 0xD8.
The command needs no argument.

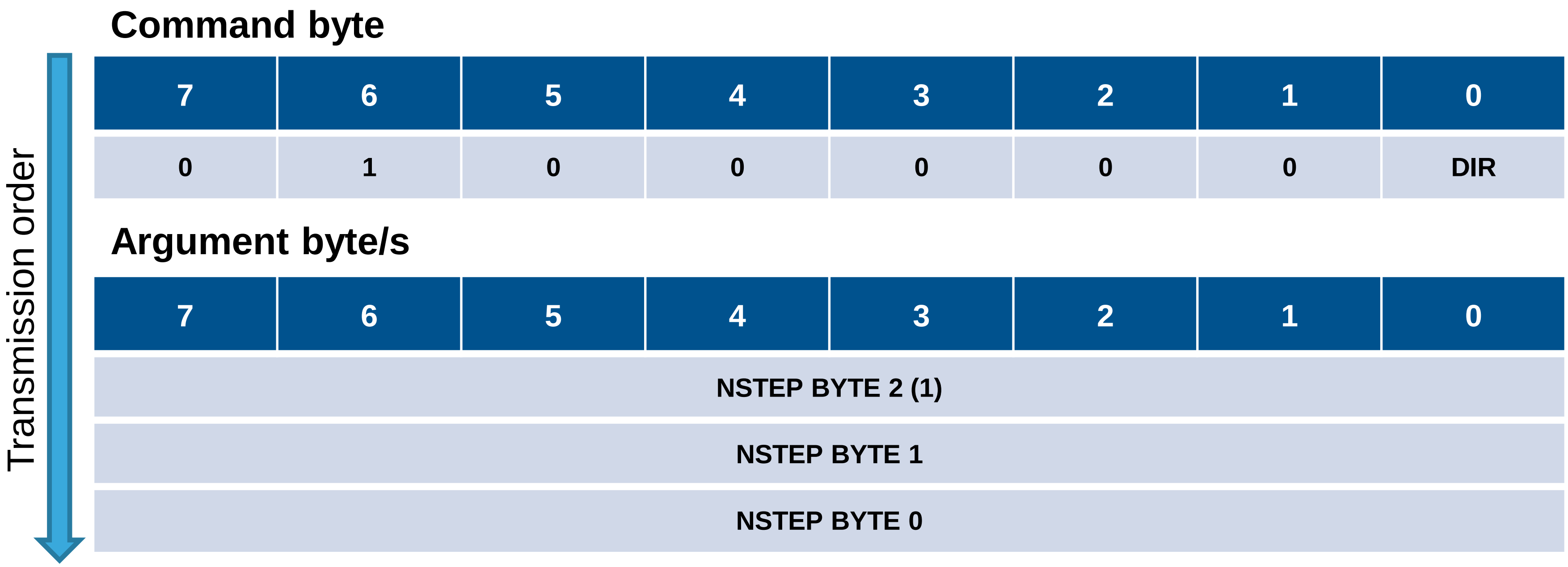
ResetDevice command

Command byte

7	6	5	4	3	2	1	0
1	1	0	0	0	0	0	0

The **ResetDevice** command byte is 0xC0.
The command needs no argument.

Move command



Command byte

7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	DIR

Argument byte/s

7	6	5	4	3	2	1	0
NSTEP BYTE 2 (1)							
NSTEP BYTE 1							
NSTEP BYTE 0							

The **Move** command byte is 0x40 for a reverse direction movement and 0x41 for a forward direction movement (the LSb defines the motion direction).
The command needs a **3-byte argument** indicating the **target number of microsteps** (or steps) which must be performed (MSByte first).

(1) Bits from 6 to 7 are ignored because NSTEP is a 22 bit value

GoTo command

Transmission order

Command byte

7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	0

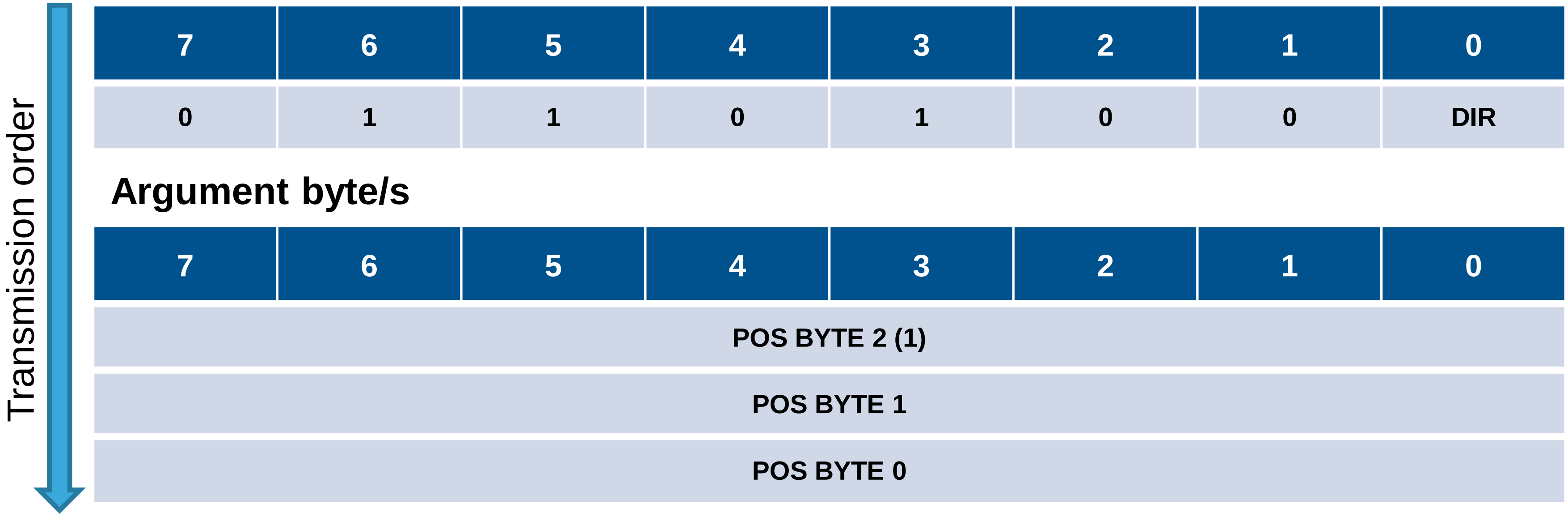
Argument byte/s

<table>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td colspan="8">POS BYTE 2 (1)</td></tr>
<tr><td colspan="8">POS BYTE 1</td></tr>
<tr><td colspan="8">POS BYTE 0</td></tr>
</table>

The **GoTo** command byte is 0x60.
The command needs a **3-byte argument** indicating the **target absolute position** value (MSByte first).

(1) Bits from 6 to 7 are ignored because POS is a 22 bit value

GoTo_DIR command



Transmission order

Command byte

7	6	5	4	3	2	1	0
0	1	1	0	1	0	0	DIR

Argument byte/s

7	6	5	4	3	2	1	0
POS BYTE 2 (1)							
POS BYTE 1							
POS BYTE 0							

The **GoTo_DIR** command byte is 0x68 for a reverse direction movement and 0x69 for a forward direction movement (the LSb defines the motion direction). The command needs a **3-byte argument** indicating the target absolute position value (MSByte first).

(1) Bits from 6 to 7 are ignored because POS is a 22 bit value

GoHome command

Command byte

7	6	5	4	3	2	1	0
0	1	1	1	0	0	0	0

The **GoHome** command byte is 0x70.
The command needs no argument.

GoMark command

Command byte

7	6	5	4	3	2	1	0
0	1	1	1	1	0	0	0

The **GoMark** command byte is 0x78.
The command needs no argument.

Run command

Transmission order

Command byte

7	6	5	4	3	2	1	0
0	1	0	1	0	0	0	DIR

Argument byte/s

7	6	5	4	3	2	1	0
SPEED BYTE 2 (1)							
SPEED BYTE 1							
SPEED BYTE 0							

The **Run** command byte is 0x50 for a reverse direction movement and 0x51 for a forward direction movement (the LSb defines the motion direction).
The command needs a **3-byte argument indicating the target speed** value (MSByte first).

(1) Bits from 4 to 7 are ignored because SPEED is a 20 bit value

StepClock command

Command byte

7	6	5	4	3	2	1	0
0	1	0	1	0	0	0	DIR

The **StepClock** command byte is 0x58 for a reverse direction movement and 0x59 for a forward direction movement (the LSb defines the motion direction). The command needs no argument.

SoftStop command

Command byte

7	6	5	4	3	2	1	0
1	0	1	1	0	0	0	0

The **SoftStop** command byte is 0xB0.
The command needs no argument.

HardStop command

Command byte

7	6	5	4	3	2	1	0
1	0	1	1	1	0	0	0

The **HardStop** command byte is 0xB8.
The command needs no argument.

SoftHiZ command

Command byte

7	6	5	4	3	2	1	0
1	0	1	0	0	0	0	0

The **SoftHiZ** command byte is 0xA0.
The command needs no argument.

HardHiZ command

Command byte

7	6	5	4	3	2	1	0
1	0	1	0	1	0	0	0

The **HardHiZ** command byte is 0xA8.
The command needs no argument.

GoUntil command

Transmission order

Command byte

7	6	5	4	3	2	1	0
1	0	0	0	ACT	0	1	DIR

Argument byte/s

7	6	5	4	3	2	1	0
SPEED BYTE 2 (1)							
SPEED BYTE 1							
SPEED BYTE 0							

The **GoUntil** command byte value depends on the target direction of the movement (bit 0 = low → reverse direction, bit 0 = high → forward direction) and the target action which should be performed at SW falling edge (bit 3 = low → reset the absolute position to zero, bit 3 = high → stores the absolute position into the MARK register).

The command needs a **3-byte argument indicating the target speed** value (MSByte first).

(1) Bits from 4 to 7 are ignored because SPEED is a 20 bit value

ReleaseSW command

Command byte

7	6	5	4	3	2	1	0
1	**0**	**0**	**1**	**ACT**	**0**	**1**	**DIR**

The **ReleaseSW** command byte value depends on the target direction of the movement (bit 0 = low → reverse direction, bit 0 = high → forward direction) and the target action which should be performed at SW falling edge (bit 3 = low → reset the absolute position to zero, bit 3 = high → stores the absolute position into the MARK register).
The command needs no argument.

Further information and full design support can be found at www.st.com/stspin

life.augmented