MGC3130 Summary

A MGC3130 system can be accessed at two software levels:
• by direct I2C access via message interface of GestIC Library (direct interface)
• by GestIC API as an abstraction layer of the messages (administered interface)

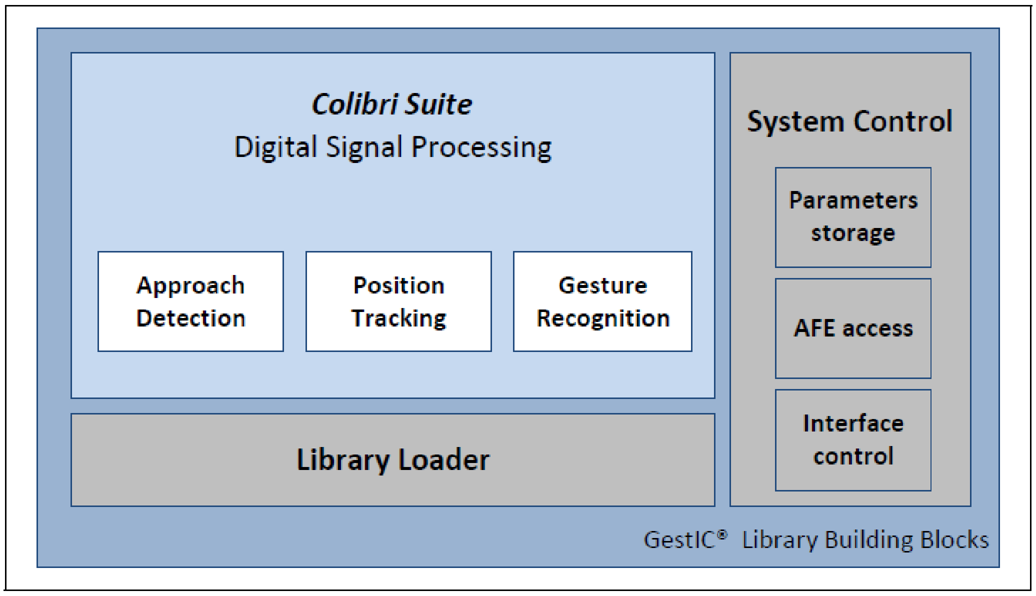
Direct access is recommended if a reduced set of sensor data are used by the application (e.g., gestures only, position only).

GestIC® LIBRARY
The GestIC Library is embedded firmware stored on the MGC3130's internal Flash memory. It contains:

- the Colibri Suite with the digital signal processing algorithms for GestIC features (i.e., GestIC core features Approach Detection, Position Tracking and Gesture Recognition)
- the System Control block providing full control of host interfaces, parameter storage and AFE access
- the Library Loader for updates of GestIC Library
-

The main building blocks are shown in Figure 1-2.
The GestIC Library incorporates a message-based interface that allows the configuration of the chip and the streaming of sensor data to the host application.

FIGURE 1-2: BUILDING BLOCKS OF GestIC® LIBRARY

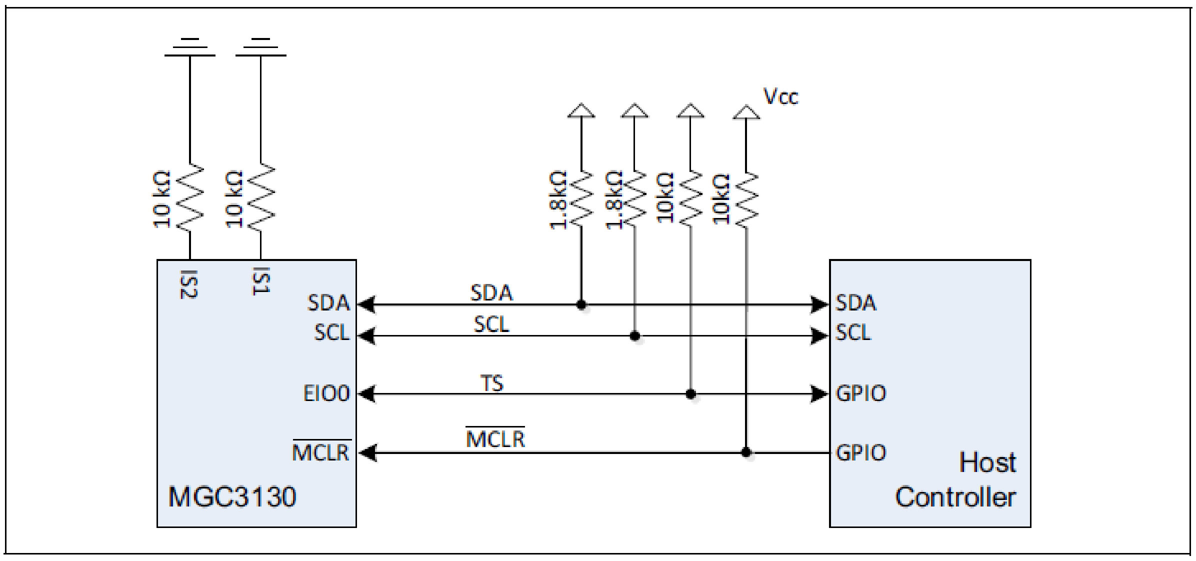


GestIC API

As an abstraction layer for MGC3130 messages, Microchip developed the GestIC API to provide a simplified user interface which can be easily integrated into the customer's application.
GestIC API comes along with a C reference code which includes message buffer, decoder and event handler to make the interface independent from the low-level protocol and its timing constraints.

MGC3130 HARDWARE INTERFACE
Communication with the MGC3130 is accomplished via a two-wire I2C compatible serial port, which allows the user to read the sensor data and to send control messages to the chip. It communicates via the serial interface with a master controller, which operates at speeds up to 400 kHz. One pin (IS2) is available for address selection and enables the user to connect up to two MGC3130 devices on the same bus without address conflict.
In addition, MGC3130 requires a dedicated transfer status line (TS), which features a data transfer status function. It is used by both I2C Master and Slave to control the data flow. I2C SCL, I2C SDA and TS lines require an open-drain connection on MGC3130 and the connected host controller. To function properly, I2C SCL and I2C SDA need to be pulled up to VCC with 1.8 kΩ resistors and the TS line needs to be pulled up to VCC with a 10 kΩ resistor.

FIGURE 2-1: HARDWARE INTERFACE TO HOST CONTROLLER



In order to complete the control options for MGC3130, it is recommended that the host controller controls the MGC3130 MCLR line. In particular, the hardware reset is necessary for the update procedure of the GestIC Library.

2.2 USAGE OF TRANSFER STATUS LINE (TS)

The transfer status line is used to check if I2C data are valid and if they can be sent from MGC3130 to the host controller.

The MGC3130 (I2C Slave) uses this line to inform the host controller (I2C Master) that there is data available which can be transferred. The host controller uses the TS line to indicate that data are being transferred and prevents MGC3130 from updating its data buffer.

Table 2-1 shows how the TS line is used in the different states of communication. MGC3130 can update the I2C buffer only when TS is released by both chips, and a data transfer can only be started when MGC3130 pulls TS low.

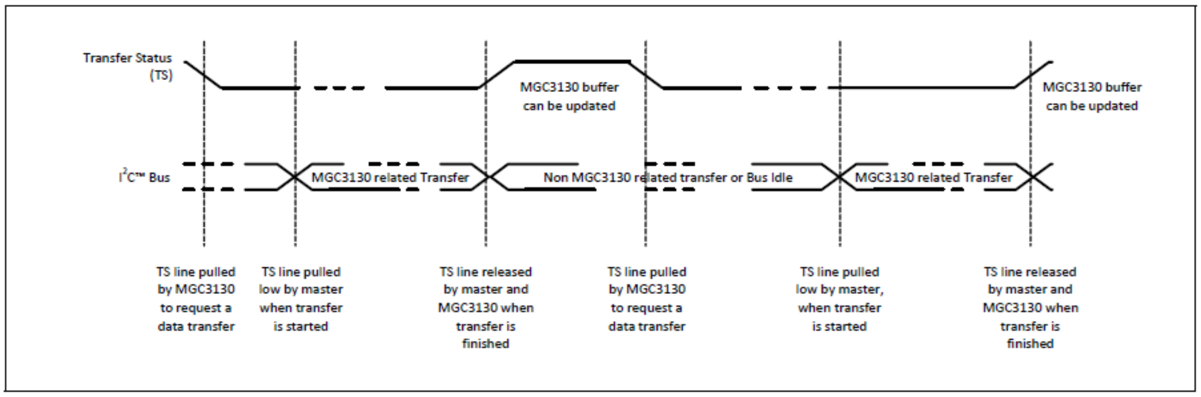
This procedure secures that:

- the host is always informed when new sensor data are available
- buffer updates in MGC3130 are always completed before data are sent to the I2C bus

TABLE 2-1: USAGE OF TRANSFER STATUS LINE

MGC3130	Host Controller	TS Line	Status
Released (H)	Released (H)	High	Host finished reading data (Transfer end). No more data to be transferred to the host. MGC3130 is allowed to update the data buffer.
Asserted (L)	Released (H)	Low	Data from MGC3130 is available to be sent, but the host has not yet started reading. If the host is busy and did not start reading before the next data update (5 ms), the MGC3130 will assert the TS line high while updating the data buffer.
Asserted (L)	Asserted (L)	Low	Host starts reading. MGC3130 data buffer will not be updated until the end of transfer (host releases TS high).
Released (H)	Asserted (L)	Low	MGC3130 is ready to update the data buffer, but the host is still reading the previous data. MGC3130 is allowed to update the data only when the host releases the TS high.

FIGURE 2-2: MGC3130 COMMUNICATION PROTOCOL


Transfer Status (TS)
MGC3130 buffer can be updated
MGC3130 buffer can be updated
I²C™ Bus
MGC3130 related Transfer
Non MGC3130 related transfer or Bus Idle
MGC3130 related Transfer
TS line pulled by MGC3130 to request a data transfer
TS line pulled low by master when transfer is started
TS line released by master and MGC3130 when transfer is finished
TS line pulled by MGC3130 to request a data transfer
TS line pulled low by master, when transfer is started
TS line released by master and MGC3130 when transfer is finished

Note 1: The stop condition after an I2C™ data transmission is generated by the host controller (I2C Master) after the data transfer is completed. Thus, it is recommended to verify the amount of bytes to be read in the message header (Size field).

2: Transfer Status is only needed for data transfer from MGC3130 to the host controller. Writing to MGC3130 does not require the additional TS signal.

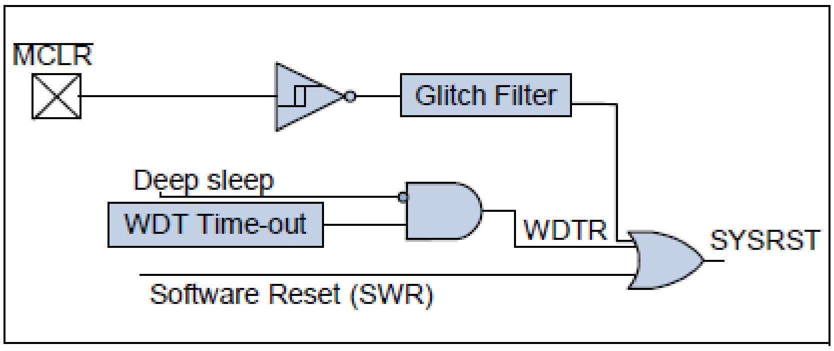
4.1 Reset Block

The Reset block combines all Reset sources. It controls the device system’s Reset signal (SYSRST). The following is a list of device Reset sources:

- MCLR: Master Clear Reset pin
- SWR: Software Reset available through GestIC Library
- WDTR: Watchdog Timer Reset

A simplified block diagram of the Reset block is illustrated in Figure 4-2.

FIGURE 4-2: SYSTEM RESET BLOCK DIAGRAM



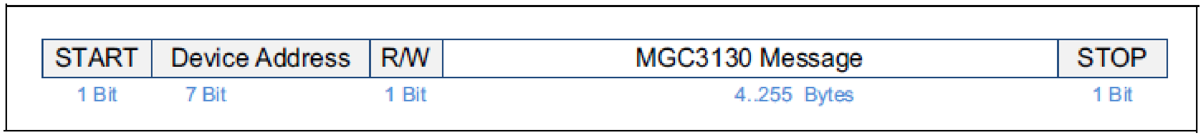
CODING EXAMPLE

EXAMPLE 2-1: CODE IMPLEMENTATION IN HOST CONTROLLER

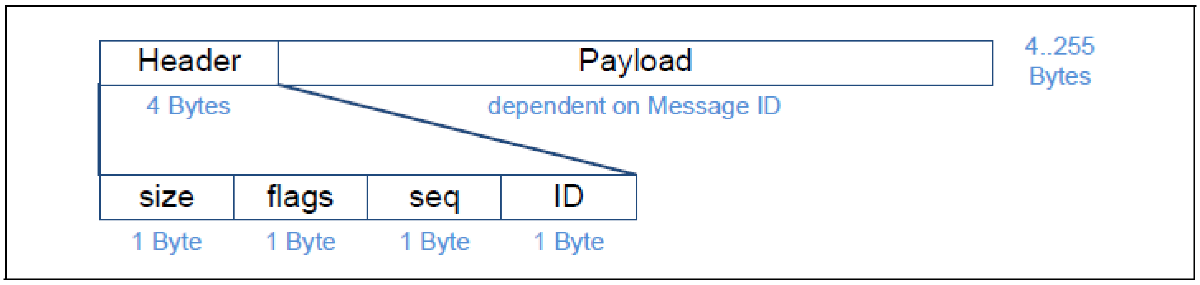
```
I²C™ Read Function – requires TS:
I²C™ Master read loop:
    Read TS
    If TS == 0:
        Assert TS
        Send I²C™ start condition
        Send I²C™ device address + read indication
        Receive I²C™ payload (the GestIC® Library message)
        Send I²C™ stop condition
        Release TS
    Wait 200 µs (to assure that MGC3130 released TS line, too)
I²C™ Write Function – does not require TS:
I²C™ Master write loop:
        Send I²C™ start condition
        Send I²C™ device address + write indication
        Send I²C™ payload (the GestIC® Library message)
        Send I²C™ stop condition
```

Message from MGC3130

MGC3130 MESSAGE EMBEDDED IN THE I^{2}C™ FRAME



Messages consist always of a 4-byte header and a variable payload.



Note: Payload elements are exchanged in little endian format. This means that the Lowest Significant Byte is written first.
Example: Element of 4 bytes: [Byte0]:[Byte1]:[Byte2]:[Byte3]

TABLE 3-4: DATA FIELDS OF MGC3130 MESSAGE HEADER

Field	Size (in bytes)	Description
Msg. Size	1	Complete size of the message in bytes including the header.
Flags	1	Reserved for future use.
Seq.	1	Sequence number which is increased for each message sent out by MGC3130. Range is 0…255. The host controller can use that information to verify if the messages got lost during I^{2}C™ transmission. MGC3130 ignores the sequence number in the received messages.
ID	1	ID of the message. For each ID, the GestIC® Library holds a dedicated structure containing the message direction, its payload elements and possible reply actions.

TABLE 3-14: MESSAGE FROM MGC3130 TO HOST: POSITION

Raw Message	`18 08 44 91 1E 01 41 8D 00 00 00 00 00 00 00 00 00 00 2F B2 E7 87 6A 35`				
Payload Element	`SystemInfo`	`GestureInfo`	`TouchInfo`	`Air-WheelInfo`	`xyzPosition`
Hex in little endian	`8D`	`00 00 00 00`	`00 00 00 00`	`00 00`	`2F B2 E7 87 6A 35`
Hex decoded	`0x8D`	`0x00000000`	`0x00000000`	`0x0000`	Byte 1 and 2: `0xB22F` Byte 3 and 4: `0x87E7` Byte 5 and 6: `0x356a`
Description	Bit 0: `PositionValid` Bit 2: `RawDataValid` Bit 3: `NoisePowerValid` Bit 7: `DSPRunning`	No Gesture Detected	Touch on Center Electrode	No AirWheel Data	x: `45615` y: `34791` z: `13674`