Chipsmall Limited consists of a professional team with an average of over 10 year of expertise in the distribution of electronic components. Based in Hongkong, we have already established firm and mutual-benefit business relationships with customers from,Europe,America and south Asia,supplying obsolete and hard-to-find components to meet their specific needs.

With the principle of “Quality Parts,Customers Priority,Honest Operation,and Considerate Service”,our business mainly focus on the distribution of electronic components. Line cards we deal with include Microchip,ALPS,ROHM,Xilinx,Pulse,ON,Everlight and Freescale. Main products comprise IC,Modules,Potentiometer,IC Socket,Relay,Connector.Our parts cover such applications as commercial,industrial, and automotives areas.

We are looking forward to setting up business relationship with you and hope to provide you with the best service and solution. Let us make a better world for our industry!

Contact us

Tel: +86-755-8981 8866 Fax: +86-755-8427 6832
Email & Skype: info@chipsmall.com Web: www.chipsmall.com
Address: A1208, Overseas Decoration Building, #122 Zhenhua RD., Futian, Shenzhen, China

MAX21000 USER GUIDE

Revision 2.6, May 2015

maxim integrated™

Rev. 2.6, May 2015

CONTENTS

TABLES

FIGURES

1 Revision History

Date	Revision	Description
1.0	12/12/2012	Inital Release
2.0	3/4/2015	Initial release with new format
2.1	3/5/2015	First review with new format
2.2	3/6/2015	Misspellings and mis-references are fixed
2.3	3/6/2015	Registers are renamed
2.4	4/13/2015	DSYNC section is added Bit fields are updated Detailed endian information is included Detailed if_parity information is included Added new programming examples
2.5	05/01/2015	Re-formatting Typos and mis-references fixed Updated register definitions
2.6	05/18/2015	Re-formatting Fixed broken links Fixed register fields name Review for errors

2 Introduction

MEMS sensors are revolutionizing the way people interact with everyday technology, making it easier and more user-friendly. Maxim can leverage its analog integration expertise to develop and manufacture new breakthrough MEMS sensors being smaller, lower power and more accurate than ever.

Owning the entire supply chain, Maxim brings its customers complete, reliable and cost-effective solutions, ensuring prompt time-to-volume and time-to-market to effectively address high-volume applications in consumer and industrial market segments.

Thanks to its leadership in analog integration and its manufacturing experience in MEMS, Maxim is capable of high-volume production to meet the market's demands. Maxim’s manufacturing expertise and highest quality standards also guarantee high performance and product reliability.

Every MEMS sensor is tested and trimmed in factory so that for most consumer applications, no additional sensor calibrations are required. The end user can quickly verify the sensor’s operation without physically tilting or rotating the sensor thanks to the built-in self-test feature, which allows accelerating the time-to-market for mass production.

This User Guide will provide a clear picture of the guidelines for its use in consumer applications and a comprehensive description of his unique features. The final section of this guide will present the structure of the register file, the purpose of each field or every register, including two examples about typical programming sequences.

3 Nomenclature

ODR	Output Data Rate
BW	Bandwidth
FS	Fulllscale
UI	User Interface
OIS	Optical Image Stabilization
MSB	Most Significant Bit or Byte
LSB	Least Significant Bit or Byte
HPF	Highpass Filter
LPF	Lowpass Filter
dps	Degrees per seconds
RFU	Reserved for future uses

4 MAX21000 Description

The MAX21000 is a low-power, low-noise, three-axis angular rate sensor able to offer unprecedented accuracy and sensitivity over temperature and time.

It is capable of working with a supply voltage as low as 1.71V for minimum power consumption. It includes a sensing element and an IC interface capable of providing the measured angular rate to the external world (I^2C/SPI).

The MAX21000 has a configurable full scale of ±31.25/±62.5/±125/±250/±500/±1000/±2000dps and is capable of measuring rates with a finely tunable user-selectable bandwidth. The high output data rate (ODR) and the large bandwidth (BW), together with the low phase delay, make the MAX21000 suitable for both user interface (UI) and optical image stabilization (OIS) applications.

The MAX21000 is a highly integrated solution requiring only two external capacitors, available in a compact 3mm x 3mm x 0.9mm plastic land grid array (LGA) package and can operate within a temperature range of -40°C to +85°C.

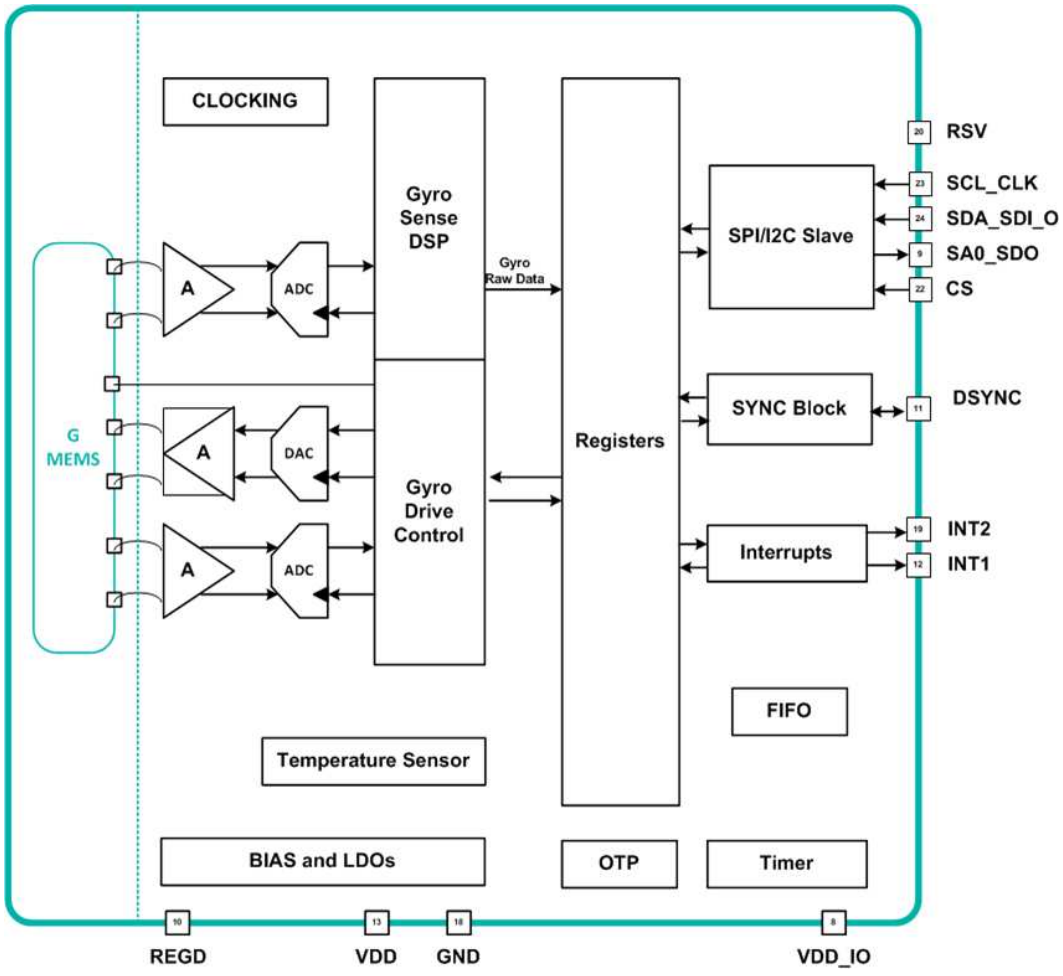

Figure 1: Block Diagram

5 Pin Description

Table 1: Pin Description

PIN	NAME	FUNCTION
1	V_{DDIO}	Interface and Interrupt Pad Supply Voltage. Same range of V_{DD}. $V_{DDIO} \leq V_{DD}$ (diode).
2	N.C.	Not Internally Connected
3	N.C.	Not Internally Connected
4	SCL_CLK	SPI and I^2C Clock. When in I^2C mode, the IO has selectable anti-spike filter and delay to ensure correct hold time.
5	GND	Power-Supply Ground
6	SDA_SDI_O	SPI In/Out Pin and I^2C Serial Data. When in I^2C mode, the IO has selectable anti-spike filter and delay to ensure correct hold time.
7	SA0_SDO	SPI Serial Data Out or I^2C Slave Address LSB
8	CS	SPI Chip Select/Serial Interface Selection
9	INT2	Interrupt Line #2
10	RESERVED	Must be connected to GND
11	INT1	Interrupt Line #1
12	DSYNC	Data Synchronization Pin to wake up MAX21000 from power down/standby or to synchronize data with an external device.
13	RESERVED	Leave unconnected
14	V_{DD}	Analog Power Supply Pin: Bypass to GND with a 0.1µF capacitor and one 10µF capacitor.
15	V_{DD}	Must be tied to V_{DD} in the application.
16	N.C.	Not Internally Connected

6 I²C Interface

To connect a MAX21000 device to an I²C master, the SDA_SDI_O pin of the MAX21000 device must be connected to the SDA pin of the I²C master and the SCL_CLK pin of the MAX21000 device must be connected to the SCL pin of the I²C master. Both SDA and SCL lines must be connected to a pullup resistor. The SA0_SDO pin must be connected to VDD or GND to configure the MAX21000 I²C slave address (see Table 3: I²C Device Addresses).

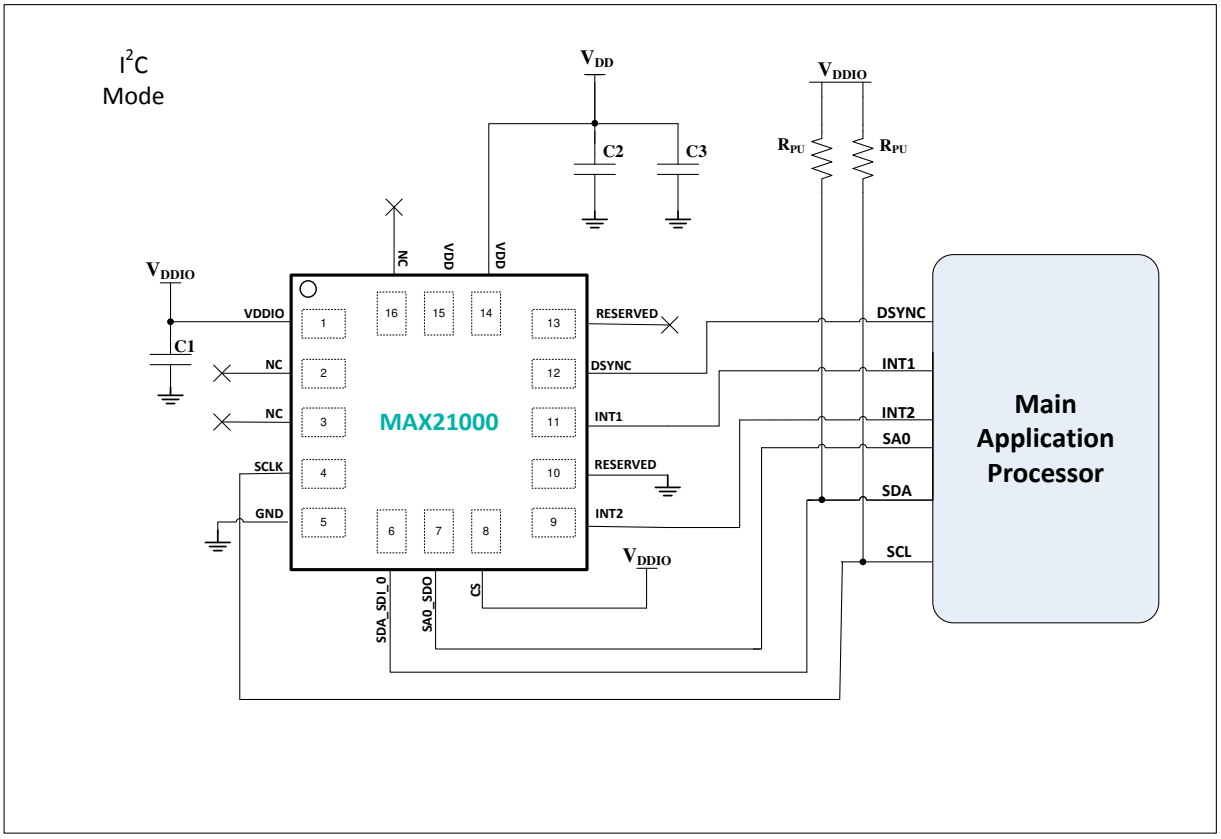

Figure 2: I²C Interface Connection to an Application Processor

Table 2: I²C External Component Properties

COMPONENT	LABEL	SPECIFICATION	QUANTITY
V_{DDIO} /V_{DD} Bypass Capacitor	C1,C2	Ceramic, X7R, 100nF ±10%, 4V	2
V_{DD} Bypass Capacitor	C3	Ceramic, X7R, 1uF ±10%, 4V	1
Pullup Resistor (I²C Mode only)	R_{PU}	1.1kΩ - 10kΩ (min - max)	2

6.1 I²C Protocol

To start an I²C request, the master sends a START condition (S), followed by the MAX21000's I²C address. Then, the master sends the address of the register to be programmed. The master then terminates the communication by issuing a STOP condition (P) to relinquish the control of the bus, or a repeated START condition (Sr) to keep controlling it.

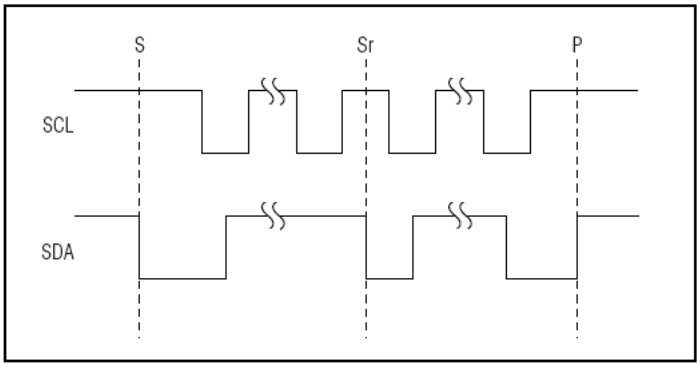

Figure 3: START (S), STOP (P), and Repeated START (Sr) Conditions

6.2 Slave Address

The slave address is used to identify the MAX21000 device in I²C communications. The address is defined as the seven most significant bits followed by the read/write bit. Set the read/write bit to 1 to request a read operation, or 0 to request a write operation (see Table 3).

Table 3: I²C Device Addresses

I²C Base Address	SA0_SDO pin	R/W bit	Resulting Address
0x2C (6bit)	0	0	0xB0
0x2C	0	1	0xB1
0x2C	1	0	0xB2
0x2C	1	1	0xB3

6.3 Acknowledge

The acknowledge bit is sent after every byte. This bit allows the receiver to notify the transmitter that the byte has been received correctly and another byte may be sent. The master generates all clock pulses, including the acknowledge's ninth clock pulse (8 bits of data + ACK).

To allow the receiver to send the acknowledge, the transmitter releases the SDA line during the acknowledge pulse, so that the receiver can pull the SDA line LOW during the ninth clock pulse to signal an Acknowledge (ACK), or release the SDA line (HIGH) to signal a Not Acknowledge (NACK).

The NACK is sent if the device is busy or a system fault occurs. It is also used by the master to signal the end of the transfer during a read operation.

6.4 Register Address

The I²C register address for the MAX21000 is composed by 6 bits of address and 1 bit (if parity) whose meaning can be configured as:

Auto-increment:	if 0, in case of burst operation the initial register address is auto-incremented after every data byte; if 1, the operation is executed always on the same register;
Even parity	this bit represents the even parity computed on the 6 bits of the register address;
Odd Parity	this bit represents the odd parity computed on the 6 bits of the register address;

6.5 I²C Operations

6.5.1 Write One Byte

To write one byte, the following steps must be executed:

1: The master sends a START condition.
2: The master sends the 7 bits slave ID plus a write bit (low)
3: The addressed slave asserts an ACK on the data line.
4: The master sends 8 bits of the Register Address.
5: The slave asserts an ACK on the data line ***only if the address is valid (NACK if not)***.
6: The master sends 8 bits of data.
7: The slave asserts an ACK on the data line.
8: The master generates a STOP condition.

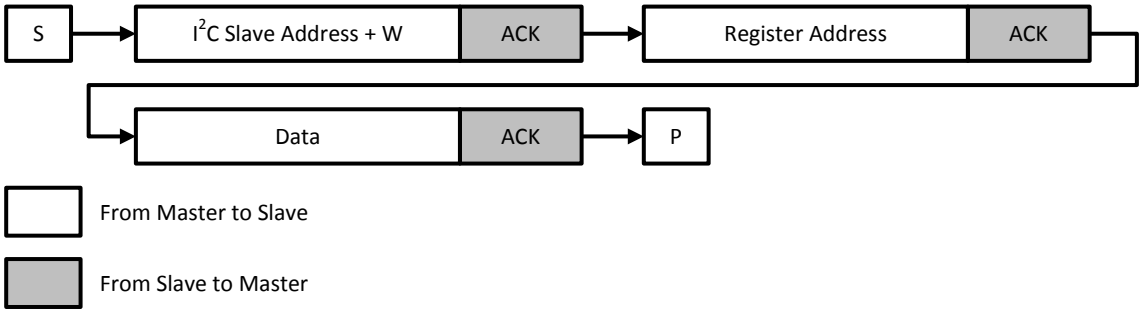

Figure 4: I²C Write One Byte

6.5.2 Write a Burst of Data

To execute a write of a burst of data, the following steps must be executed:

1: The master sends a START condition.
2: The master sends the 7 bits slave ID plus a write bit (low).
3: The addressed slave asserts an ACK on the data line.
4: The master sends 8 bits of the Register Address.
5: The slave asserts an ACK on the data line ***only if the address is valid (NACK if not)***.
6: The master sends 8 bits of data.
7: The slave asserts an ACK on the data line.
8: Repeat 6 and 7 as long as needed.

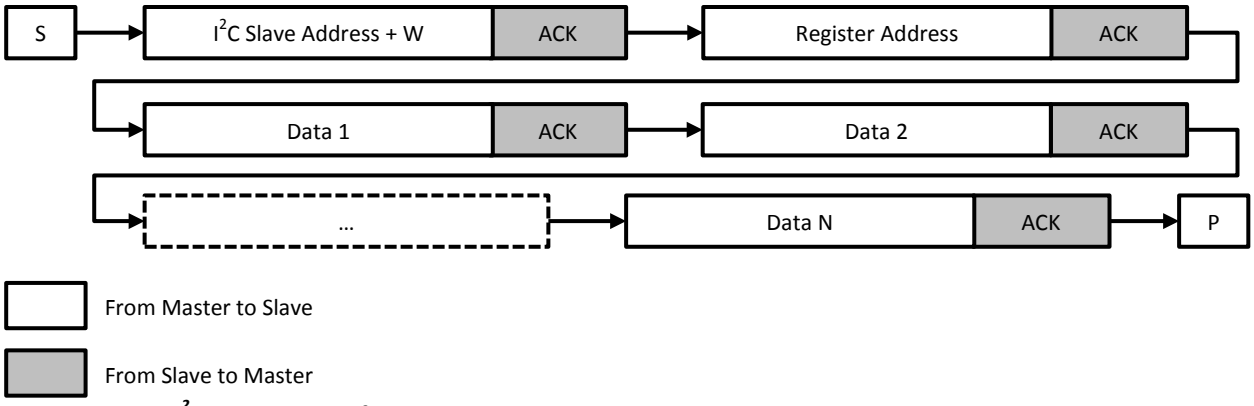

Figure 5: I²C Write a Burst of Data

6.5.3 Read One Byte

To read one byte, the following steps must be executed:

1: The master sends a START condition.
2: The master sends the 7 bits slave ID plus a write bit (low).
3: The addressed slave asserts an ACK on the data line.
4: The master sends 8 data bits.
5: The active slave asserts an ACK on the data line ***only if the address is valid (NACK if not)***.
6: The master sends a restart condition.
7: The master sends the 7 bits slave ID plus a read bit (high).
8: The addressed slave asserts an ACK on the data line.
9: The slave sends 8 data bits.
10: The master asserts a NACK on the data line.
11: The master generates a STOP condition.

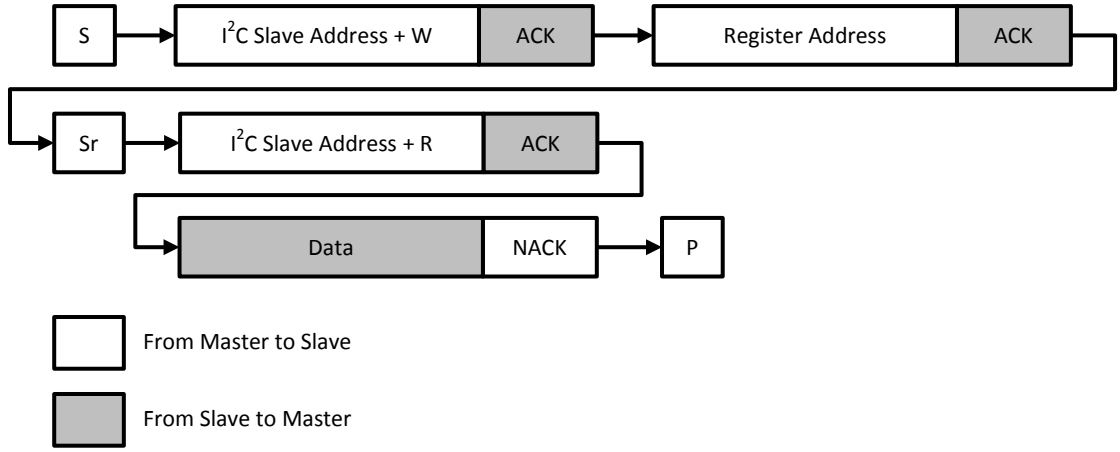

Figure 6: I²C Read One Byte

6.5.4 Read a Burst of Data

To execute a read of a burst of data, the following steps must be executed:

1: The master sends a START condition.
2: The master sends the 7 bits slave ID plus a write bit (low).
3: The addressed slave asserts an ACK on the data line.
4: The master sends 8 bits of the Register Address.
5: The slave asserts an ACK on the data line **only if the address is valid (NACK if not)**.
6: The master sends a repeated START condition.
7: The master sends the 7 bits slave ID plus a read bit (high).
8: The slave asserts an ACK on the data line.
9: The slave sends 8 bits of data.
10: The master asserts an ACK on the data line.
11: Repeat 9 and 10 as long as needed.
12: The master generates a STOP condition.

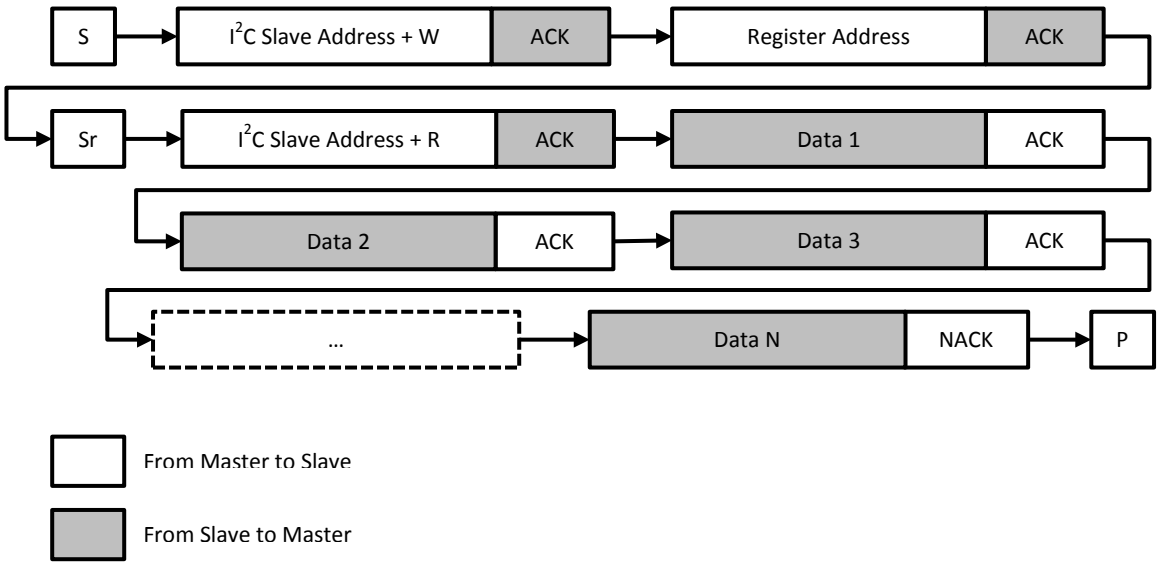

Figure 7: I²C Read a Burst of Data

7 SPI Interface

To connect a MAX21000 device to an SPI master, the CS pin of the MAX21000 device must be connected to the CSn pin of the SPI master, the SDA_SDI_O pin of the MAX21000 device must be connected to the MOSI pin of the SPI master, the SA0_SDO pin of the MAX21000 device must be connected to the MISO pin of the SPI master and the SCL_CLK pin of the MAX21000 device must be connected to the SCLK pin of the SPI master. For external component parameters, refer to Table 4: SPI External Component Properties.

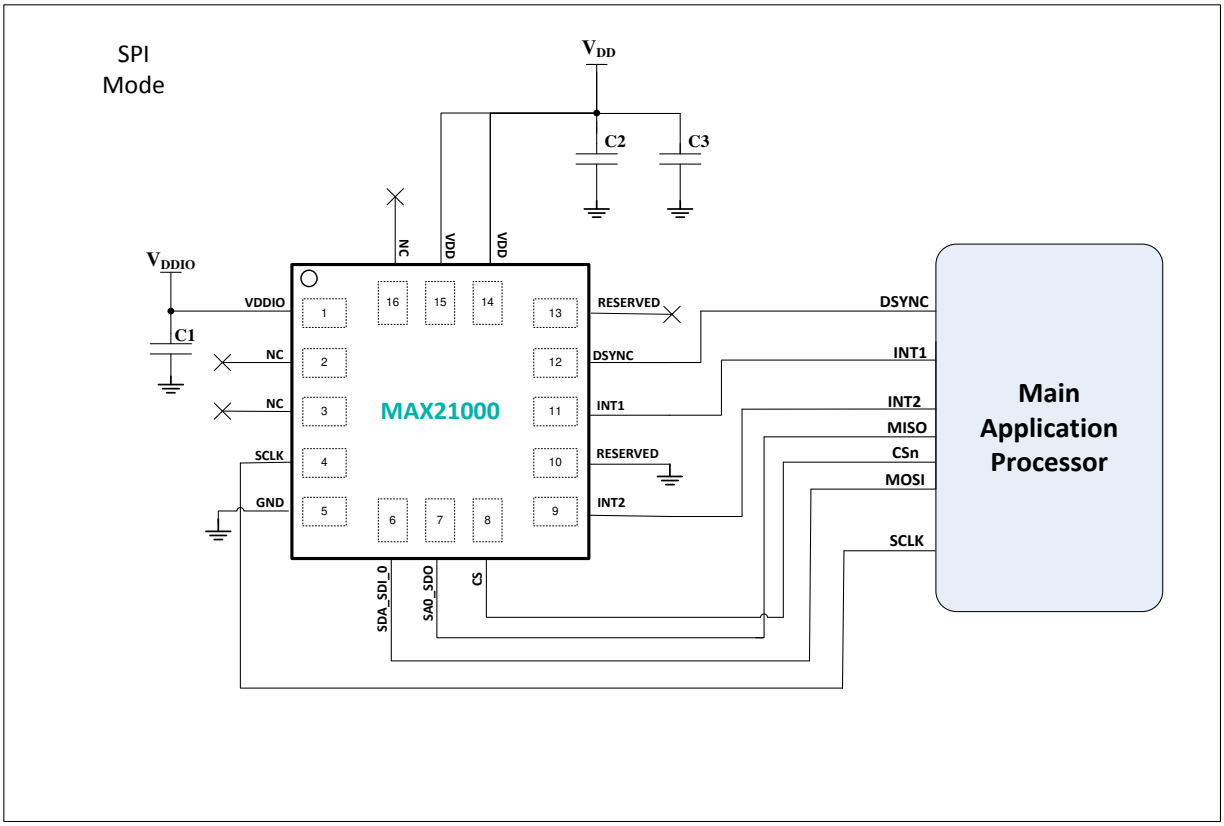

Figure 8: SPI Interface Connection to an Application Processor

Table 4: SPI External Component Properties

COMPONENT	LABEL	SPECIFICATION	QUANTITY
V_{DDIO} /V_{DD} Bypass Capacitor	C1,C2	Ceramic, X7R, 100nF ±10%, 4V	2
V_{DD} Bypass Capacitor	C3	Ceramic, X7R, 1µF ±10%, 4V	1

7.1 SPI Protocol

CSn is the serial port enable and is controlled by the SPI master. It goes low at the start of the transmission and returns to high at the end.

SCLK is the serial port clock and is controlled by the SPI master. It is kept high when CSn is high (no transmission). MISO and MOSI are, respectively, the serial port data input and output. These lines are driven at the falling edge of SCLK and are sampled at the rising edge of SCLK.

Both the read register and write register commands are completed in 16 clock pulses, or in multiples of 8 in case of multiple read/write bytes. Bit duration is the time between two falling edges of SCLK.

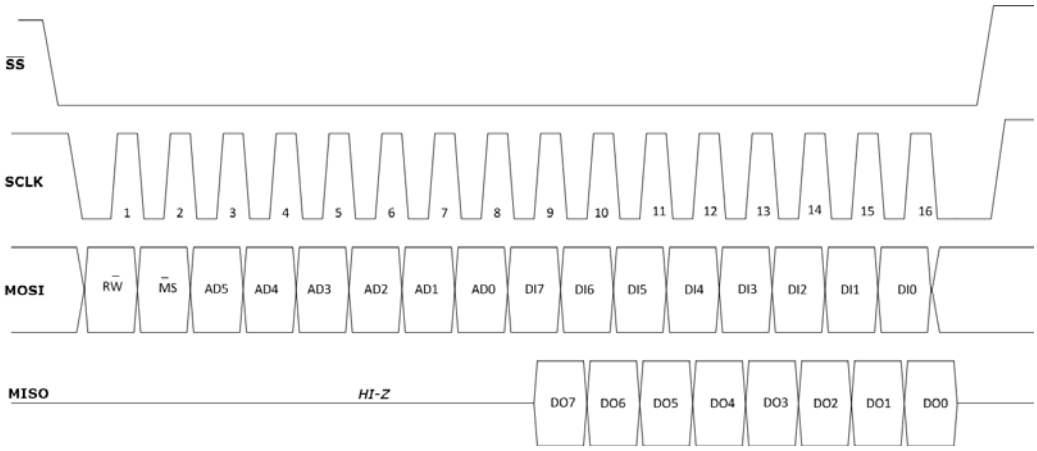

Figure 9: SPI Protocol

7.2 Register Address

The SPI register address for the MAX21000 is composed by 6 bits of address, 1 bit to select the direction of the operation (Read/Write) and 1 bit whose meaning can be configured as:

Auto-increment:	if 0, in case of burst operation the initial register address is auto-incremented after every data byte; if 1, the operation is executed always on the same register;
Even parity	this bit represents the even parity computed on the 6 bits of the register address;
Odd Parity	this bit represents the odd parity computed on the 6 bits of the register address;

parity_error and if_parity are used to manage the parity bit during the SPI communication. The parity bit is an additional bit added to the end of a digital word and it indicates whether the number of bits in the word with the value one is even or odd. Parity bit is used to verify if there was a communication error. According to the datasheet, during a SPI communication, the first byte is the register address you want to read/write:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
R/W	MS/Parity	A5	A4	A3	A2	A1	A0

Bit 7:	Is used to define if you want to read or write a register. 0 = write, 1 = read;
Bit 6:	Can have 2 different functionalities, in according with the if_parity (see below)
Bit [5:0]	Address of the register you want to read or write;
If_parity = '00':	Bit 6 is used to set the **multi-addressing standard mode** (MS). MS = 0 -> the address is auto incremented in multiple read/write command MS = 1 -> the address remains unchanged in multiple read/write commands;
if_parity = '01':	Bit 6 is used to check the **even parity** with the register address (A[5:0]);
if_parity = '10':	Bit 6 is used to check the **odd parity** with the register address (A[5:0]);

Here are shown some examples:

Address	count of 1 bits	8 bits including parity	
		Even	Odd
000000 (0x00)	0	x0000000	x1000000
100000 (0x20)	1	x1100000	x0100000
100011 (0x23)	3	x1100011	x0100011
111111 (0x3F)	6	x0111111	x1111111

Bit 7 – indicated with x, can be 0 or 1, depending on if you want to write or read the correspondent register

For further explanation:

- to **write** register **0x00**, using **odd parity,** you have to send from your MCU the byte '**01000000**' (0x40);
- to **read** register **0x20**, using **even parity,** you have to send from your MCU the byte '**11100000**' (0xE0);
- to **read** register **0x23**, using **odd parity,** you have to send from your MCU the byte '**10100011**' (0xA3);
- to **read** register **0x3F**, using **even parity,** you have to send from your MCU the byte '**10111111**' (0xBF);

When the device receives the above bytes from the MCU, it will try to calculate the parity bit, in according with the rule set in the ***if_parity*** field. If the parity calculated by the device is the same of the parity bit, the communication was good and the content of ***parity_error*** bit will be 0. If instead the parity calculated by the device is different from the parity bit, the content of ***parity_error*** bit is set to '1', reporting that there was an error in the serial communication.

8 Interrupts

The MAX21000 is equipped with an interrupt module to control a set of interrupt flags and two interrupt lines (INT1 and INT2).

This module allows to:

1: Configure the behavior of the interrupt lines (INT1 and INT2)
2: Map each interrupt flag to one of both the interrupt lines
3: Create a conditional interrupt (Rate interrupt) based on four different thresholds

8.1 Interrupt Flags

The interrupt module provides several interrupt flags in the INT_STS_UL register. Each flag reports the occurrence of a significant event inside the device.

8.2 Interrupt Lines

An interrupt line is a dedicated pin where a notification to an external application processor can be provided. The MAX21000 is equipped with two interrupt lines that can be configured independently.

For each interrupt line, it is possible to (refer to INT_CFG2):

- Enable/disable them
- Set the active level to low
- Set the output type to push-pull or open drain configuration

To map an interrupt flag to an interrupt line it is necessary to enable it through the INT1_MSK for INT1 and INT2_MSK for INT2: by setting its corresponding bit to 1 on the bit-mask, the interrupt flag is mapped to the related interrupt line. The output of the two INT1 and INT2 interrupt lines is then computed by applying the OR operator to all the enabled interrupt flags contained in the INT1_STS and INT2_STS registers, respectively. Please note that the enable is not applied to the **INT1_STS** and **INT2_STS** registers, so those registers contain also the value of the interrupt flags that are not enabled.

An interrupt line can be configured in order to keep the status until the master requests to clear it (latched) or after a timeout. Those modes can be selected through the ***int1_latch_mode*** and ***int2_latch_mode*** register fields (INT_TMO). The only exception is the ***gyro_dr*** flag that is always unlatched regardless of the ***int1_latch_mode*** and ***int2_latch_mode*** register fields setting.

The lines can be configured also to auto-clear its status after a period of time where the duration to clear the interrupt is programmable (see Timed Mode at INT_TMO).

8.3 Rate Interrupts

The *rate interrupt* allows generating an interrupt event when the gyroscope data falls inside a range of values defined by a set of thresholds. By setting the absolute value of the threshold (|TH|) for each gyroscope axis, four ranges are defined:

Figure 10: Conditional Ranges

For each range, it is possible to select if it contributes to the generation of the rate interrupt for each axis. Then, the rate interrupts are computed as follow:

- **int_and:** Active if the defined conditions are satisfied for all the gyroscope axes at the same time;
- **int_or:** Active if the defined conditions are satisfied for at least one of the gyroscope axes.

The threshold absolute value can be set by writing the **INT_REF_X**, **INT_REF_Y** and **INT_REF_Z** registers. Those registers represent the most significant byte of the real threshold; so, to compute the actual absolute value of the threshold, the register value must be multiplied by 256. If the application requires a better resolution, it is possible to specify a 16-bit threshold by setting the int_single_ref register field; in this case, all the axes share the same threshold that is defined as the combination of INT_REF_X and INT_REF_Y where the first one is the most significant byte and the last one is the least significant byte of the threshold.

Once the thresholds are defined, the INT_MSK_X, INT_MSK_Y and INT_MSK_Z permits to select which range contributes to the generation of the rate interrupts for each *axis* as follow:

- **int_***axis***_high_pos_en:** if enabled, the condition is satisfied if the data of the *axis* falls in the *high_pos* range;
- **int_***axis***_high_neg_en:** if enabled, the condition is satisfied if the data of the *axis* falls in the *high_neg* range;
- **int_***axis***_low_pos_en:** if enabled, the condition is satisfied if the data of the *axis* falls in the *low_pos* range;
- **int_***axis***_low_neg_en:** if enabled, the condition is satisfied if the data of the *axis* falls in the *low_neg* range.

Then, the desired axes must be enabled by setting to 1 the corresponding bit of the int_mask_xyx_and for the **int_and** and int_mask_xyz_or for the **int_or**.

Through the INT_MSK_X, INT_MSK_Y and INT_MSK_Z registers, it is also possible to read the current value of the conditions, regardless of the content of the int_mask_xyz_or and int_mask_xyz_and register fields. Each of these values can be configured as latched by setting the int_freeze register field. For the rate interrupts, it is possible also to define a de-bounce value by defining the number of samples the axis data has to satisfy the condition before asserting the corresponding interrupt. Those values can be set in the INT_DEB_X, INT_DEB_Y and INT_DEB_Z registers. If the required value is greater than 255, it is possible to define a 16-bit debounce value by setting int_single_deb register field; in this case, all the conditions share the same de-bounce value that is defined as the combination of INT_DEB_X and INT_DEB_Y where the first one is the most significant byte and the last one is the least significant byte of the debounce value.

9 Reading Data from MAX21000 and FIFO Operation

MAX21000 sensor output data can be read by the processor in two different mechanisms: synchronous (DATA_READY interrupt) and asynchronous (polling) reading methods.

9.1 Synchronous Reading

In this mode, the processor reads the data set (e.g. 6 bytes for a 3 axes configuration) generated by the MAX21000 every time that gyro_dr flag is set. The processor must read the output data only once the gyro_dr flag is set in order to avoid data inconsistencies. Benefits of using this approach include the perfect reconstruction of the signal coming from the sensors and minimization of the data traffic at the interface.

9.2 Asynchronous Reading

In this mode, the processor reads the data generated by the MAX21000 regardless the status of the gyro_dr flag. To minimize the error caused by different samples being read a different number of times, the access frequency to be used must be higher than the selected ODR.

9.3 FIFO Modes Description

The MAX21000 also embeds a 256-slot of a 16-bit data FIFO for each of the three output channel; X, Y and Z. This allows a consistent power saving for the system since the host processor does not need to continuously poll data from the sensor, but it can wake up only when needed and burst the data out from the FIFO. When configured in Snapshot mode, it offers the ideal mechanism to capture the data following a Rate Interrupt event. The data order in FIFO depends on the endian setting (endian):

Big Endian: *GYRO_X_H, GYRO_X_L, GYRO_Y_H, GYRO_Y_L, GYRO_Z_H, GYRO_Z_L*
Little Endian: *GYRO_X_L, GYRO_X_H, GYRO_Y_L, GYRO_Y_H, GYRO_Z_L, GYRO_Z_H*

The FIFO buffer can work according to four main different modes (see FIFO_CFG): off, normal, interrupt and snapshot. Both normal and Interrupt modes can be optionally configured to operate in overrun Mode, depending on whether, in case of buffer under-run, newer or older data are lost.

9.3.1 FIFO OFF Mode

In this mode, the FIFO is turned off; data are stored only in the data registers. No data are available from FIFO if read. When the FIFO is turned OFF, there are two options to use the device: synchronous/asynchronous reading.

9.3.2 Normal Mode

The behavior of the FIFO in Normal mode varies depending on the **Overrun** settings (***fifo_overrun*** register field). The following paragraphs show a description of the behavior for both settings of the overrun. For the next sections, the following descriptions are useful for the user's understanding of FIFO:

Write Pointer (WP): *Write pointer is the address number where the next data will be written in FIFO, and whenever there is a new data is written, the write pointer is incremented by 1,*
Read Pointer (RP): *Read pointer is the address number from where the first data in FIFO is read, and whenever there is a new data is read, the read pointer is incremented by 1,*
Level *Level tells the difference between Write pointer and Read pointer, which also gives you the total number of data available in FIFO*

9.3.2.1 Stop on Full

- FIFO is turned on.
- FIFO is filled with the data at the selected Output Data Rate (ODR).
- When FIFO is full, an interrupt can be generated.
- When FIFO is full, all the new incoming data will be discharged. Reading only a subset of the data already stored into the FIFO keeps locked the possibility for new data to be written.
- Only if all the data are read the FIFO will restart saving data. If the communication speed is high, data loss can be prevented.
- In order to prevent a FIFO full condition, the required condition is to complete the reading of the data set before the next DATA_READY occurs.
- If this condition is not guaranteed, data can be lost.

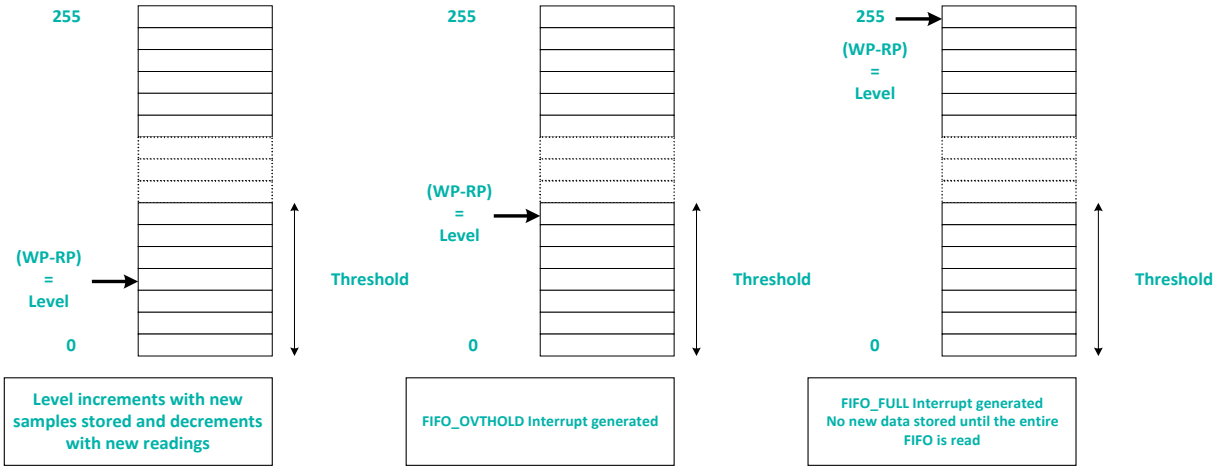

Figure 11: FIFO Normal Mode, Stop on Full

9.3.2.2 Overwrite

- FIFO is turned on.
- FIFO is filled with the data at the selected ODR.
- When FIFO is full, an interrupt can be generated.
- When FIFO is full, the oldest data will be overwritten with the new ones.
- If communication speed is high, data integrity can be preserved.
- In order to prevent a data lost condition, the requirement is to complete the reading of the data set before the next DATA_READY occurs.
- If this condition is not guaranteed, data can be overwritten.
- When an overrun condition occurs the Reading pointer is forced to Writing Pointer -1, to make sure only older data are discarded and newer data have a chance to be read.

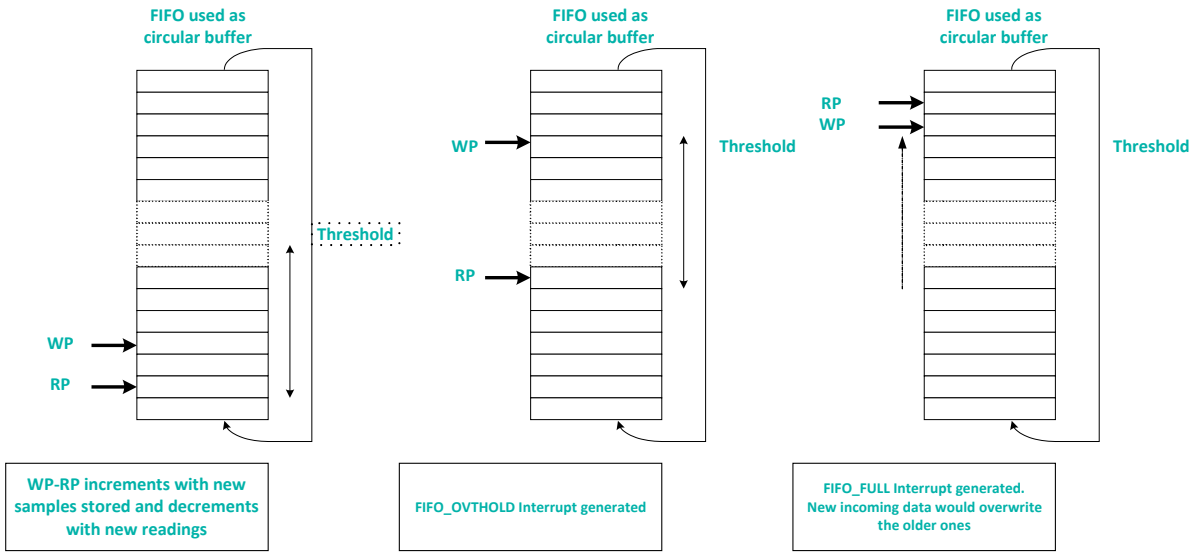

Figure 12: FIFO Normal Mode, Overwrite

9.3.3 Interrupt Mode

The behavior of the FIFO in Interrupt mode varies depending on the **Overrun** settings (***fifo_overrun*** register field). The following paragraphs show a description of the behavior for both settings of the overrun.

9.3.3.1 Stop on Full

- FIFO is initially disabled. Data is stored only in the data registers.
- When a Rate Interrupt (either OR or AND) is generated, the FIFO is turned on automatically. It stores the data at the selected ODR. Rate interrupt are documented starting from INT_REF_X.
- When FIFO is full, all the new incoming data will be discharged. Reading only a subset of the data already stored into the FIFO keeps locked the possibility for new data to be written.
- Only if all the data are read the FIFO will restart saving data.
- If communication speed is high, data loss can be prevented.
- In order to prevent a FIFO full condition, the required condition is to complete the reading of the data set before the next DATA_READY occurs.
- If this condition is not guaranteed, data can be lost.

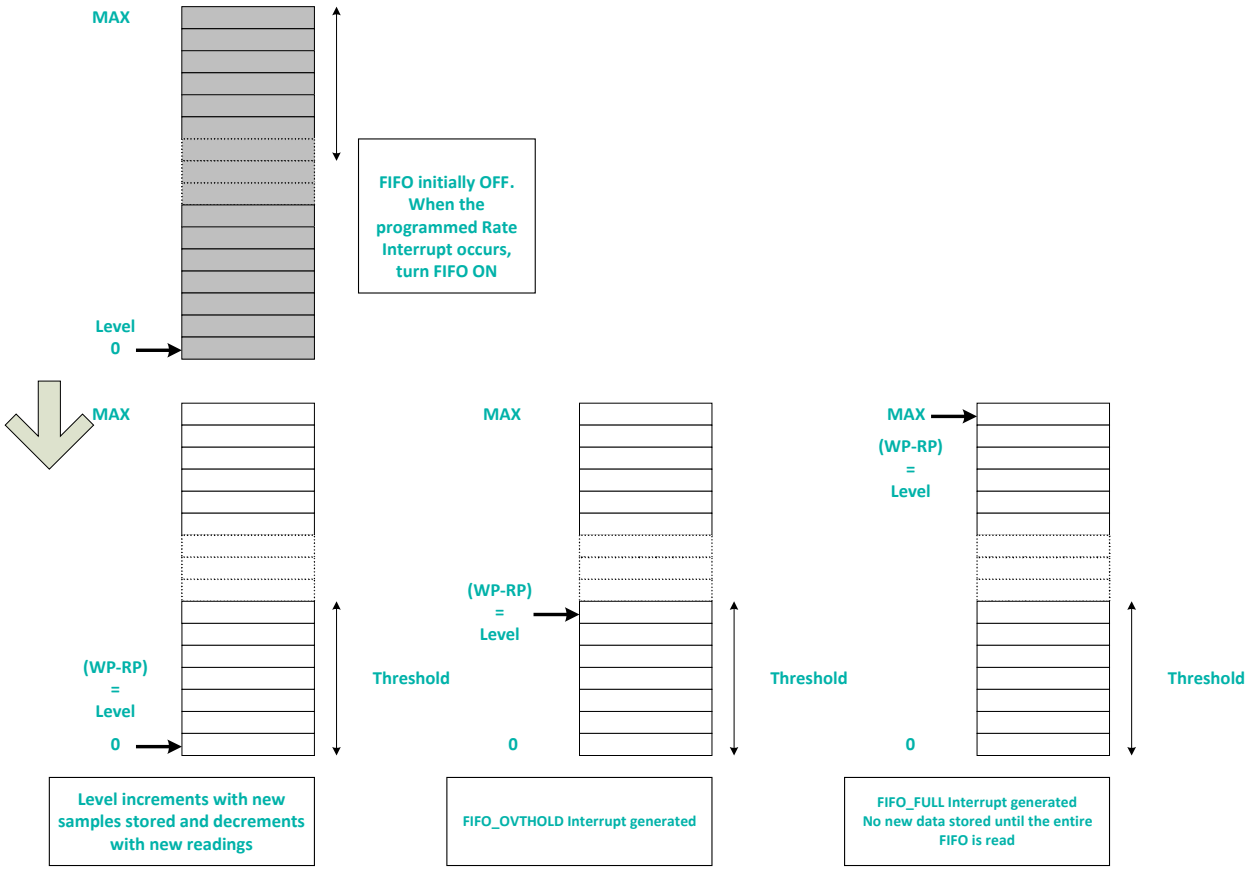

Figure 13: FIFO Interrupt Mode, Stop on Full

9.3.3.2 Overwrite

- FIFO is initially disabled. Data is stored only in the data registers.
- When a Rate Interrupt (either OR or AND) is generated, the FIFO is turned on automatically. It stores the data at the selected ODR.
- When FIFO is full, an interrupt can be generated.
- When FIFO is full, the oldest data will be overwritten with the new ones.
- If communication speed is high, data integrity can be preserved.
- In order to prevent a data lost condition, the required condition is to complete the reading of the data set before the next DATA_READY occurs.
- If this condition is not guaranteed, data can be overwritten.
- When an overrun condition occurs the Reading pointer is forced to Writing Pointer -1, to make sure only older data are discarded and newer data have a chance to be read.