SDK
frizz firmware Specifications

Rev.1.0

January 23, 2015

MegaChips Corporation

CAUTION

Table of Contents

1. Introduction

This document describes functional specifications provided with frizz and frizz firmware.

2. frizz Overview

frizz is a 32-bit DSP equipped sensor HUB IC capable of high-speed arithmetic processing at low frequencies with instruction extension through ParaForce. It is based on Xtensa LX4 from Tensilica and is suitable for pedestrian dead reckoning (PDR) processing.

2.1. Overview of frizz Specifications

The overview of frizz specifications are shown in the following (Table 2-1).

Table 2-1 frizz Hardware Specifications

No.	Item	Details
1	CPU	Xtensa LX4 (40 MHz)
2	RAM	Instruction RAM (iram): 256 KB Data RAM (dram):256 KB
3	Power supply	Core: 1.2 V IO：1.8/2.5/2.8/3.0/3.3V
4	Power saving mode	Sleep/Stop/Standby Mode
5	Instruction extension through ParaForce *1	3way VLIW Floating Point 4way SIMD
6	Host I/F	SPI (slave) or I2C (slave) : 1ch
7	Device I/F	UART:1ch SPI（master）:1ch I2C（master）:1ch GPIO:4ch As Host I/F of frizz on Chignon, only SPI is supported.
8	Other systems	Timer 32bit × 4 JTAG Self Boot (SPI Flash) or Host CPU Download Boot

*1: For an overview of instruction extension through ParaForce, refer to “2.2 Overview of Instruction Extension through ParaForce”.

2.2. Overview of Instruction Extension through ParaForce

frizz extends instructions through ParaForce to enable high-speed arithmetic operations with each algorithm. The extended instructions can be used via the operation library described in “3.4 Arithmetic Math Libraries (libfrizz).”

2.2.1. Data Type Extensions

To support floating-point operations, the following floating-point data types are extended:

(1) Extension for handling a single precision floating-point type (32-bit: signed 1-bit, exponent 8-bit, and coefficient digits 23-bit).

(2) Extension for handling a 4-way single precision floating-point type (32-bit × 4-way) for vector operations.

2.2.2. Scalar Type Instruction Extensions (general purpose instruction extensions)

To support floating-point operations and comparisons, the following floating-point scalar type instructions are extended:

(1) Multiplication of a single precision floating-point number
(2) Multiplication and accumulation of a single precision floating-point number
(3) Multiplication and subtraction of a single precision floating-point number
(4) Multiplication and accumulation of a single precision floating-point number
(5) Addition of a single precision floating-point number
(6) Subtraction of a single precision floating-point number
(7) Comparison of a single precision floating-point number (<; less than)
(8) Comparison of a single precision floating-point number (<=; less than or equal to)
(9) Comparison of a single precision floating-point number (>; greater than)
(10) Comparison of a single precision floating-point number (>=; greater than or equal to)

2.2.3. Vector Type Instruction Extensions (matrix operation extensions)

To support matrix operations, the following floating-point scalar type instructions are extended:

(1) Zero initialization of a 4-way single precision floating-point number
(2) Multiplication of a 4-way single precision floating-point number
(3) Addition of a 4-way single precision floating-point number
(4) Subtraction of a 4-way single precision floating-point number
(5) Move instruction of a 4-way single precision floating-point number
(6) Addition of a 4-way single precision floating-point number shifting the elements of matrix
(7) Multiplication and accumulation of a 4-way single precision floating-point number
(8) Multiplication and subtraction of a 4-way single precision floating-point number
(9) Multiplication and subtraction of a 4-way single precision floating-point number

2.2.4. Scalar/Vector Mixed Type Instruction Extensions

To support matrix operations, the following scalar/vector mixed type instructions are extended:

(1) Addition of the element of a 4-way single precision floating-point matrix
(2) Scaling of a 4-way single precision floating-point matrix
(3) Scaling and addition of a 4-way single precision floating-point matrix
(4) Store instruction that unfolds a single precision floating-point in a vector
(5) Shift instruction that adds a single precision floating-point number to a vector after sliding the vector
(6) Pick up instruction that obtains a certain single precision floating-point number from a 4-way single precision floating-point matrix

2.2.5. Other Instruction Extensions

Other instructions that are also extended are listed below:

(1) Acquire frizz instruction extension version
(2) bit-swap instruction
(3) byte-swap instruction
(4) WAITI instruction (instruction to await interrupt)

3. Software Structure

A software block diagram for frizz firmware is shown in the following (Figure 3-1).

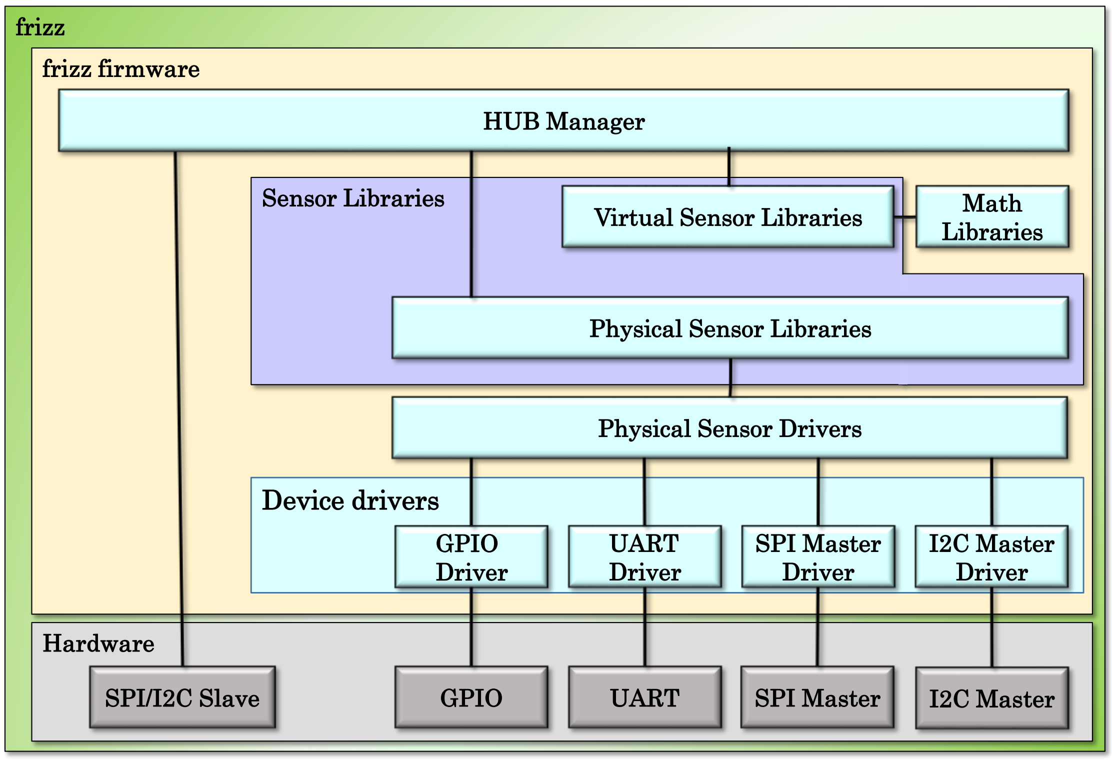


Figure 3-1 frizz firmware Software Block Diagram

frizz firmware consists of the following modules:

(1) HUB Manager
(2) Device Drivers (GPIO Driver, UART Driver, SPI Master Driver, I2C Master Driver)
(3) Sensor Libraries (Physical Sensor Libraries, Virtual Sensor Libraries)
(4) Math Libraries
(5) Physical Sensor Drivers

3.1. HUB Manager

The HUB manager is module that serve as interfaces with the Host CPU, control each of the sensor libraries, and has following functions:

(1) Initialize sensor libraries.
(2) Activate/deactivate for sensing of each sensor.
(3) Set the sampling periods of each sensor.
(4) Calculate the sampling period of the entire system.
(5) Receive and parse commands from the Host CPU and set each sensor library.
(6) Notify the Host CPU of sensor data acquired from each sensor library.
(7) Associate the virtual sensor library with the physical sensor libraries.
(8) Control the hardware used by each device to realize the above functions.

3.2. Device Drivers

frizz firmware has the following device drivers (GPIO Driver, UART Driver, SPI Master Driver and I2C Driver in Figure 3-1).

(1) I2C (master)
(2) SPI (master)
(3) UART
(4) GPIO
(5) TIMER

The HUB manager and physical sensor libraries use the above device drivers.
With frizz firmware, devices can be easily controlled in using the above device drivers.

3.3. Sensor Libraries

In addition to physical sensor libraries (Physical Sensor Libraries in Figure 3-1) that handle physical sensors connected physically to frizz, frizz firmware also has virtual sensor libraries (Virtual Sensor Libraries in Figure 3-1) that perform various operations based on sensor data obtained from the physical sensor libraries.

3.4. Math Libraries (libfrizz)

frizz firmware has the following math libraries(Math Library in Figure 3-1).

(1) Trigonometric function operation library
(2) Karman filter operation library
(3) Arithmetic operation library for using extended instructions

Using these operation libraries, frizz firmware can perform high-speed floating-point and vector operations even with a low operation clock frequency.

3.4.1. Trigonometric Function Operation Library

The trigonometric function operation library can perform the following operations related to trigonometric functions:

(1) Get absolute value
(2) Get square root
(3) Sine function
(4) Cosine function
(5) Arc sine function
(6) Arc cosine function
(7) Arc tangent function
(8) Arc tangent function for two variables

3.4.2. Karman Filter Operation Library

With the Karman filter operation library, arbitrary state equations can be designed in accordance with the following equation, and high-accuracy state estimations can be performed using the following functions.

State equation:

$$x(n) = \Phi(n)\, x(n-1) + G(n)\, u(n) + w$$

x: Vector of state variable
Φ: Transition matrix (normally [I + Fdt])
I: Identity Matrix
F: Dynamics matrix
dt: Sampling interval time
G: Control-input transition matrix
u: Vector of control-input
w: Process white-noise

$$z(n) = H(n)\, x(n) + v$$

z: Vector of measurement
H: Measurement matrix
v: Measurement white-noise

Library functions：

(1) Refer to state variable.
(2) Refer to predicted variable.
(3) Refer to difference of vector between measurement and predicted.
(4) Refer to Φ matrix (Transition matrix).
(5) Refer to G matrix (Control-input transition matrix).
(6) Refer to H matrix (Measurement matrix).
(7) Refer to error covariance matrix.
(8) Refer to covariance matrix for process noise at continuous time space.
(9) Refer to covariance matrix for sensor noise.
(10) Refer to covariance matrix for noise of control-input.
(11) Acquire identity matrix.
(12) Prepare system noise covariance matrix at discrete time space.
(13) Execute prediction.
(14) Update state (execute filtering).

3.4.3. Arithmetic Operation Library for Using Extended Instructions

The arithmetic operation library for using extended instructions has functions which operate arithmetic operations using extended instructions described in “2.2 Overview of Instruction Extension through ParaForce”.

3.5. Physical Sensor Drivers

Each physical sensor libraries, the setting value to each physical sensors received from the HUB manager, via each device drivers using the physical sensor drivers (I2C, SPI, UART, GPIO), it is set to each physical sensors.

Use the physical sensor drivers, each physical sensors data acquired through each device drivers, it notifies the HUB manager or each virtual sensors.

4. Overview of frizz Internal Control

Internal control of frizz is described below.
The internal functions of frizz can be controlled using the Host I/F and sensor HUB framework.

4.1. Status (RUN/STALL)

frizz (xtensa core) has two operation statuses, i.e., “RUN” and “STALL.”
In the “STALL” status, frizz can accept frizz firmware and unfold firmware in the internal RAM to prepare for boot up of frizz firmware.
Next, the system enters “RUN” status to boot up (operate) frizz firmware.
These statuses can be accessed from the Host CPU through SPI or I2C by enabling control with the frizz ctrl register.

4.2. Command Interface

To realize a frizz command interface with the Host CPU, the Host CPU writes a command in the frizz message register, and then the HUB manager analyzes and processes that command.
The Host CPU can access the message register through SPI.

4.3. FIFO

FIFO is used to transmit the ACK/NACK and response with respect to the command from the Host CPU, and to transmit sensor data.
The Host CPU reads the FIFO periodically or as triggered by the GPIO interruption from frizz.
The Host CPU can access the FIFO as a register, and thus the Host CPU can access the FIFO through the SPI.

4.4. GPIO

frizz has a 4-channel GPIO that can be used to notify the Host CPU and control peripheral devices including the PWM.

4.5. Interrupt/Power Saving Mode

Xtensa has an 8-channel interruption that is used by frizz for the following purposes:

Table4-1 Interruption List

No.	Factor	Priority level	Type
0	TIM0	1	Timer
1	TIM1	2	Timer
2	message	4	Level
3	TIM2	3	Timer
4	Awake	4	Level
5	I2C master	1	Level
6	UART Master	1	Level
7	GPIO3	1	Level
8	(not use)	-	-
9	SPI Master	1	Level
10	GPIO2	1	Level
11	(not use)	-	-
12	(not use)	-	-
13	FIFO RD	1	Edge
14	GPIO3	1	Edge
15	GPIO2	1	Edge

(*)Priority level are given priority in the order of precedence.

The Message interruption occurs when the Host CPU writes to the message register through the SPI.

The Awake interruption is a type of timer interruption that uses the Xtensa core external block, and it occurs when the value set in the Awake register, which is counted down in synchronization with the clock operation, becomes zero.

Depending on the setting, the Xtensa core can be stalled between the time that the occurrence of a WAITI instruction is detected and when an interruption occurs.

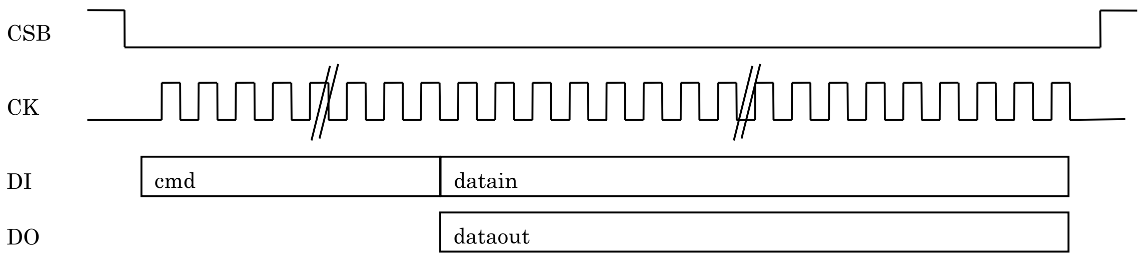
At this time, the clock in the Xtensa core is also stopped, so this WAITI instruction can be used for a deep sleep mode that causes frizz to operate in a power saving mode.

5. Host Interface (SPI, GPIO interruption)

frizz receives commands such as sensor settings from the Host CPU and notifies the Host CPU of sensor data through serial communication (SPI/I2C).
This module describes the communication specifications of the interface with the Host CPU.

5.1. SPI Communication Format

frizz performs serial communication with the Host CPU through SPI as an SPI slave mode device.
The SPI mode can be set to mode 0 or mode 3.
The SCLK of the SPI should be 1/10 or less than the frizz operation clock. (For example, when the operation clock is set to 40 MHz, the maximum SCLK is 4 MHz).
The SPI data format between the Host CPU and frizz is shown below.



*In case of SPI mode3

Figure 5-1 SPI Data Communication Format

The Host CPU and frizz transmit commands and data by sending cmd, datain, and dataout with MSB first.
The Read/Write flag and the frizz HOST I/F register address for reading and writing are stored in cmd.
In the case of a Read operation, frizz stores the register value in datain.
In case of a Write operation, dataout contains the data to write to the register.
Commands are sent to the sensor HUB by setting the frizz message register address to the cmd field.

Table5-1 Data Structure of the SPI Communication Format

Field	Bit	Description
cmd	Cycle (2bit)	Specify the access cycle (Read/Write) C: 2’b10 = Write, 2’b00 = Read, other = N/U
	Address (14bit)	frizz Host I/F register address to be accessed
datain/dataout	Data (32bit × N)	Data (Consecutive data transmission is possible while the CS signal is being continuously asserted. In this case, the next data becomes the register data for the next address.)

5.2. I2C Communication Format

I2C of host mode operates at Slave mode. There are Standard mode(Max.100kHz), Fast mode(Max.400kHz) and Fast mode+(Max.1MHz).

5.2.1. Write sequence

Figure 5-2 shows the sequence to write to the host interface register from Host I2C Device.

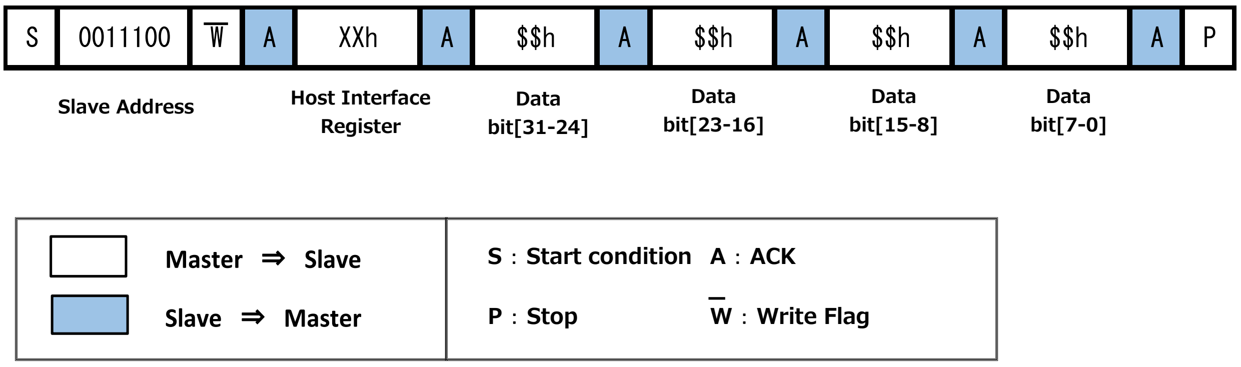


Figure 5-2 Host Interface I2C Write Sequence

5.2.2. Read sequence

Figure 5-3 shows the sequence to read the host interface register from Host I2C Device.

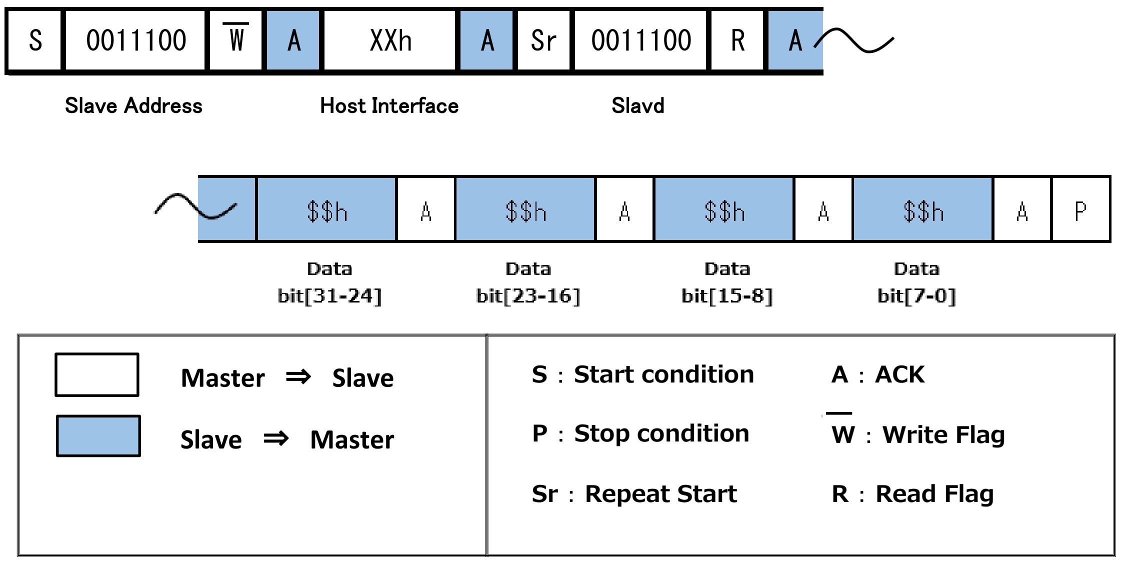


Figure 5-3 Host Interface I2C Read Sequence

5.3. frizz Host I/F registers specification

A list of frizz Host I/F registers specified in the SPI/I2C data format is in the following (Table5-2).

Table5-2 frizz Host I/F Registers

Register Address	Register name	Function
000h	ctrl	Operation control
001h	ver	HW version display
002h	mes	Message register
003h	mode	Refers to LSI operation modes
03Fh	fifo_cnr	fifo status display
040h	fifo	fifo data output
052h	ram_addr	iram/dram address
053h	ram_data	iram/dram data
0C0h–0FFh	snap_shot	Snap Shot Buffer

The details of each frizz Host I/F registers are shown below.

5.3.1. ctrl Register

Table5-3 ctrl Register

Register Name : ctrl (W)		Initial Value : {28’d0, 4’b0010}
D[31:4]	W	*1
D[3]	W	DPBUF FIFO reset 1 = Reset FIFO by writing 1.
D[2]	W	*1
D[1]	W	Stall Xtensa 0 = Set Xtensa core to Run state. 1 = Set Xtensa core to Stall state.
D[0]	W	System reset 1 = Reset blocks other than the Host I/F block (spi master/i2c master/Host I/F register) by writing 1.

*1 : Ignore when writing. Read “0” when reading.

5.3.2. mes Register

Table5-4 mes Register

Register Name : mes (W/R)		Reset Value : 32’h00000000
D[31:0]	W/R	Message register frizz firmware (Xtensa) reads the value written in this register using the XLMI (Xtensa Local Memory Interface).

5.3.3. ver Register

Table5-5 ver Register

Register Name : ver (R)		Initial Value : 32’h00000100
D[31:0]	R	Reads the HW frizz version (0x0000_0200).

5.3.4. mode Register

Table5-6 mode Register

Register Name：mode (W/R)		Initial Value：depends on the SEL[3:0] setting
D[31:18]	R	*1
D[17]	R	Refers to the xtensa Stall state.(0:Run 1: Stall)
D[16]	R	Detected error during self-boot.
D[15:12]	R	*1
D[11]	R	Internal oscillator (high-speed:40MHz) power down status 0:Normal 1: Power down
D[10]	R	Clock source setting monitor (Host) 0: Low-speed clock 1: High-speed clock
D[9]	W/R	Clock source setting monitor (ParaForce) 0: Low-speed clock 1: High-speed clock
D[8]	R	ParaForce Wait Mode 0:Normal operation 1:Waiting interrupt
D[7:3]	RW	*2
D[2:0]	R	Setting status monitor (SEL0,SEL1) by external terminal setting (SEL0,SEL1) [2]：Host Interface serial communication mode 0:mode0 1:mode3 [1]：Select Host Interface (0:SPI 1:I2C) [0]：SELFBOOT（0:Host DL Boot 1:SELFBOOT）

*1 : Ignore when writing. Read “0” when reading.

*2 : Initial Value is “0”, Read the written data after writing.

5.3.5. fifo_cnr Register

Table5-7 fifo_cnr Register

Register Name：fifo_cnr (W/R)		Initial Value：32’h00000000
D[31:16]	R	Shows the data count (word) in the FIFO.
D[15]	R	Shows that a FIFO overrun has occurred.
D[14]	R	Shows that a FIFO underrun has occurred.
D[13:2]	R	*1
D[1:0]	W/R	DPRAM operation mode setting 00 : FIFO mode (xtensa -> Host CPU) 01 : FIFO mode (Host CPU -> xtensa) 1x: Snap Shot mode (Reader : Host CPU, Writer : xtensa)

*1 : Ignore when writing. Read “0” when reading.

5.3.6. fifo Register

Table5-8 fifo Register

Register Name：fifo (W/R)		Initial Value：undefined
D[31:0]	W/R	Free format shown in Table5-11 Data Format. FIFO capacity is 64 words (x32bit).

5.3.7. ram_adr Register

Table5-9 ram_adr Register

Register Name : ram_adr (W/R)		Initial Value : 32’h00000000
D[31:0]	W/R	Address register for accessing iram/dram. iram : 00000h–0ffffh dram : 10000h–1ffffh

5.3.8. ram_data Register

Table5-10 ram_data Register

Register Name : ram_data (W/R)		Initial Value : 32’h00000000
D[31:0]	W/R	Data register for accessing iram/dram.

5.3.9. Accessing iram/dram

iram/dram can be indirectly accessed by storing the iram/dram address in the ram_addr register and the data in the ram_data register.
When the Host writes to the ram_data register, xtensa writes the data to iram/dram address that shows the ram_adr register.
The value of the ram_adr register must be specified in increments of 1 word, i.e., 00000h to 0ffffh is for iram and 10000h to 1ffffh is for dram. In addition, when reading or writing to the ram_data register occurs, the address is incremented automatically.
When accessing iram/dram, the bit1 of the ctrl register must be set to 1.
In case of a write operation, the Host can access the ram_addr register and ram_data register consecutively from both the SPI slave BUS and I2C slave BUS. During a read operation, after the memory address is written to the ram_addr, reading from ram_data must be implemented.

5.4. Data Format

The data format used between the Host CPU and frizz is shown below.

The Host CPU can send commands to the frizz HUB manager by setting the frizz mes register address and writing data in the register address on the frizz side using the format shown in “Table5-1” (transmission from Host CPU to frizz).
In contrast, the frizz HUB manager writes ACK/NACK and response to the commands as well as the acquired and computed sensor data in FIFO. The Host CPU can read the FIFO by setting and reading the FIFO register address in the frizz register address (transmission from frizz to Host CPU).

Data transmission and reception between the Host CPU and the sensor HUB is performed as transmission from the Host CPU to frizz, and transmission from frizz to the Host CPU based on following data formats (MSB First).

Table5-11 Data Format

Byte0	Byte1	Byte2	Byte3
num	sen_id	kind	prefix(0xFF)
data[0]			
:			
data[num - 1]			

num	The number of data
sen_id	sensor ID（In the case of the HUB manager, sen_id = 0xFF）
kind	0x80：sensor data (from frizz to Host CPU)
	0x81：command (from Host CPU to frizz)
	0x82：ACK (from frizz to Host CPU)
	0x83：NACK (from frizz to Host CPU)
	0x84：response (from frizz to Host CPU)
	0x8F：BreakCode (from Host CPU to frizz)

The Host CPU can make frizz forcibly wait the receipt of a command by writing sen_id=0x8F(BreakCode) data to the mes register. Through this, the Host CPU can restore a sequence fault caused by erroneous data.

5.5. Detecting Data Stored in FIFO by frizz

5.5.1. Data Storage Detection through a GPIO Interruption

Because frizz is an SPI Slave device, it cannot send data on its own. Therefore, frizz asserts a GPIO interruption to the Host CPU to notify the Host CPU that data reception by frizz is ready. (Assertion of a GPIO interruption from frizz can be enabled or disabled based on the setting.)
Triggered by that interruption, the Host CPU can read frizz FIFO data.
In addition, the Host CPU can figure out how many words of data are in the FIFO in advance by reading the fifo_cnr register before reading FIFO.

5.5.2. Polling

If a GPIO interruption assertion from frizz is disabled, the Host CPU must poll the fifo_cnr register to detect data transmission from frizz.

5.5.3. Cyclic Sampling

frizz stores sensor data in FIFO at certain periods, and thus the Host CPU can acquire sensor data by periodically reading the frizz FIFO.
This period is the maximum polling interval that is notified by the frizz HUB manager.

5.6. Basic Transmission Sequence

5.6.1. Command Transmission Sequence (from Host CPU to frizz)

The normal command transmission sequence that is used to transmit a command from the Host CPU to frizz is shown below (Figure 5-4).

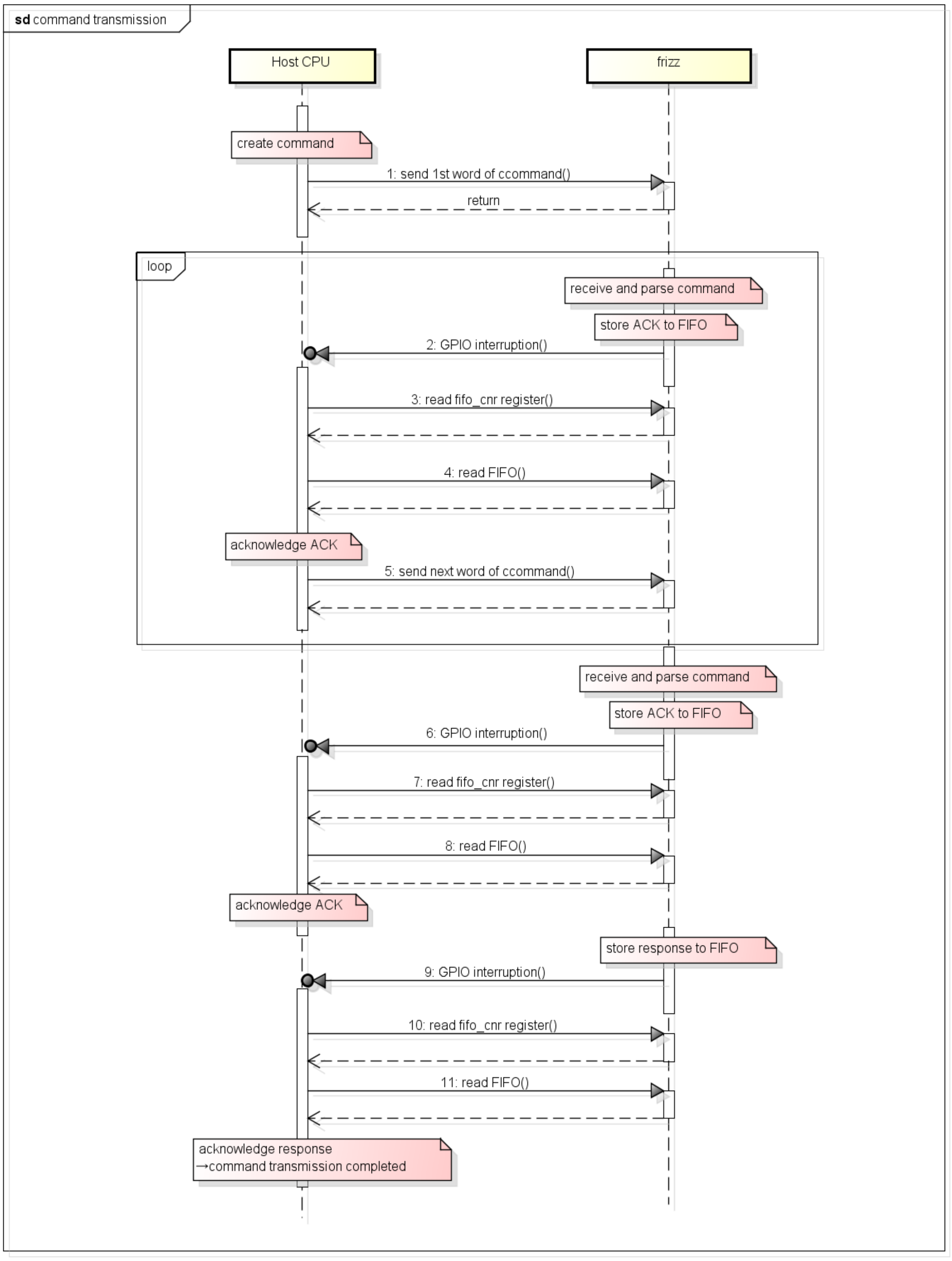


Figure 5-4 Command Transmission Sequence

5.6.2. Sensor Data Transmission Sequence (from frizz to the Host CPU)

The normal transmission sequence used to transmit sensor data from frizz to the Host CPU is shown below (Figure 5-5).

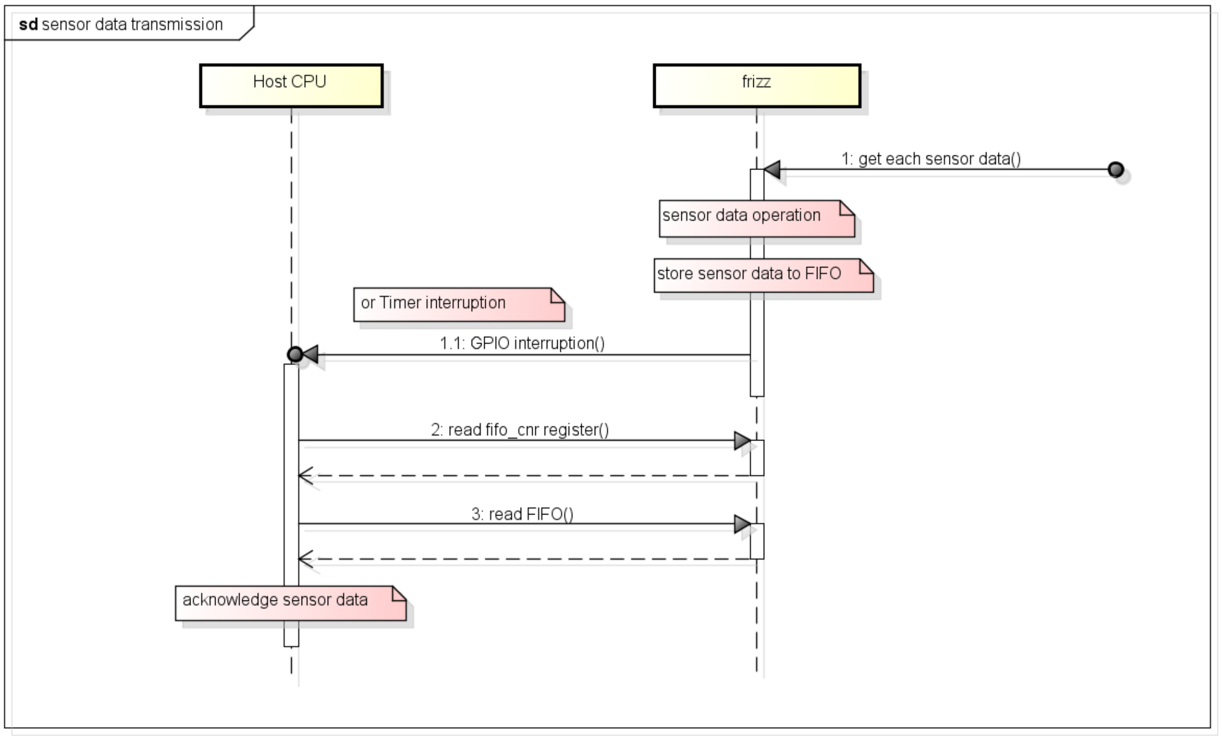


Figure 5-5 Sensor Data Transmission Sequence

5.7. HUB Manager Specifications

The HUB manager operates the following functions through a communication interface with the Host I/F.

(1) Initialize sensor libraries.
(2) Activate/deactivate for sensing of each sensor.
(3) Set the sampling periods of each sensor.
(4) Calculate the sampling period of the entire system.
(5) Receive and parse commands from the Host CPU and set each sensor library.
(6) Notify the Host CPU of sensor data acquired from each sensor library.
(7) Associate the virtual sensor library with the physical sensor libraries.
(8) Control the hardware used by each device to realize the above functions.

5.7.1. Commands

The HUB manager processes data sent from the Host CPU in the following data formats as commands for the HUB manager.

Table5-12 Command Formats

Byte0	Byte1	Byte2	Byte3
num	0xFF	0x81	0xFF
command_code			
command_param[0]			
:			
command_param[num - 2]			

Command_code consists of Byte0-Byte3 (Table5-13). Judge the command type by command_id.

Table5-13 command_code

Byte0	Byte1	Byte2	Byte3
command_id	immediate value0	immediate value1	immediate value2

command_param[0], command_param[1] and command_param[3]… are immediately executable parameters that are defined individually for each command, and the purpose of these parameters differ according to the command type (command_id).

Moreover, command_param is a parameter that is defined individually for each command, and the presence and content of the command_param depends on the command type (command_id).
The details of each command are shown below.

Table5-14 DEACTIVATE_SENSOR

command_id	DEACTIVATE_SENSOR (0x00)	
Description	Deactivates the specified sensor.	
Parameters	imm0	unsigned: sensor ID to deactivate
	imm1	Reserved
	imm2	Reserved
	command_param	void

Table5-15 SET_SENSOR_ACTIVATE

command_id	SET_SENSOR_ACTIVATE (0x01)	
Description	Activates the specified sensor.	
Parameters	imm0	unsigned: sensor ID to activate
	imm1	unsigned : Specifies sensor data output destination. 0: HW FIFO 1: SW FIFO *When using SW FIFO, the sensor data in SW FIFO is stored to HW FIFO using the PULL_SENSOR_DATA command.
	imm2	unsigned: Indicates the presence of interruption notification when sensor data is output. 0: No interruption 1: Interruption is present
	command_param	void

Table5-16 SET_SENSOR_INTERVAL

command_id	SET_SENSOR_INTERVAL (0x02)	
Description	Sets sensor data update interval to the specified sensor.	
Parameters	imm0	unsigned: sensor ID
	imm1	Reserved
	imm2	Reserved
	command_param	Sensor data update interval (units: msec)

Table5-17 GET_SENSOR_DATA

command_id	GET_SENSOR_DATA(0x03)	
Description	Acquires sensor data from the specified sensor and stores the data in HW FIFO.	
Parameters	imm0	unsigned: sensor ID
	imm1	Reserved
	imm2	Reserved
	command_param	void

Table5-18 PULL_SENSOR_DATA

command_id	PULL_SENSOR_DATA (0x04)	
Description	Stores the sensor data stored in SW FIFO in HW FIFO. When the sensor data output destination is specified as SW FIFO by the SET_SENSOR_ACTIVATE command, sensor data in SW FIFO must be stored to HW FIFO using this command. The system is notified of the number of data (word units) stored in HW FIFO by the response associated with this command. *For information on the response, please refer to “5.7.2 ACK/NACK, Response, HUB Manager Output.”	
Parameters	imm0	Reserved
	imm1	Reserved
	imm2	Reserved
	command_param	void

Table5-19 SET_SETTING

command_id	SET_SETTING (0x05)	
Description	Indicates enable/disable GPIO interruption assertion from the HUB manager.	
Parameters	imm0	Signed：PIN No. used by the GPIO interruption Less than 0: Disable GPIO interruption assertion. Greater than or equal to 0: Enable indicated GPIO interruption assertion for specified PIN No. *Falling edge assertion
	imm1	Reserved
	imm2	Reserved
	command_param	void

Table5-20 GET_FIFO_EMPTY_SIZE

command_id	GET_FIFO_EMPTY_SIZE (0x06)	
Description	Acquires empty size of HW FIFO. The system is notified of the empty size (word units) of the HW FIFO remaining by the response associated with this command. *For information on the response, please refer to “5.7.2 ACK/NACK, Response, HUB Manager Output.”	
Parameters	imm0	Reserved
	imm1	Reserved
	imm2	Reserved
	command_param	void

Table5-21 GET_VERSION

command_id	GET_VERSION (0x07)	
Description	Acquires the HUB manager version. The system is notified of the HUB manager version by the response associated with this command. *For information on the response, please refer to “5.7.2 ACK/NACK, Response, HUB Manager Output.”	
Parameters	imm0	Reserved
	imm1	Reserved
	imm2	Reserved
	command_param	void

Table5-22 GET_SENSOR_ACTIVATE

command_id	GET_SENSOR_ACTIVATE (0x08)	
Description	Acquires activation state of the specified sensor. The system is notified of the activated state (0: deactivated state, 1: activated state) by the response associated with this command. *For information on the response, please refer to “5.7.2 ACK/NACK, Response, HUB Manager Output.”	
Parameters	imm0	unsigned：sensor ID
	imm1	Reserved
	imm2	Reserved
	command_param	void

5.7.2. ACK/NACK, Response, HUB Manager Output

The HUB manager stores the ACK/NACK, response, and HUB manager output in FIFO using the following data formats.

Table5-23 ACK Format

Byte0	Byte1	Byte2	Byte3
0	0xFF	0x82	prefix = 0xFF

Table5-24 NACK Format

Byte0	Byte1	Byte2	Byte3
0	0xFF	0x83	prefix = 0xFF

Table5-25 Response format

Byte0	Byte1	Byte2	Byte3
2	0xFF	0x84	0xFF
command_code *1			
res_data *2			

*1 : Associated command_code when command is received

*2 : Response value

Table5-26 HUB Manager Output Format

Byte0	Byte1	Byte2	Byte3
3	0xFF	0x80	0xFF
time_stamp *1			
interval *2			
f_status *3			

*1 : Time stamp@tick (tick = 1 msec)

*2 : Update interval of entire sensor library (msec)

*3 : HUB manager status

HUB_MGR_IF_STATUS_ACTIVE(1 << 0)
HUB_MGR_IF_STATUS_FIFO_FULL_NOTIFY(1 << 1)
HUB_MGR_IF_STATUS_UPDATED (1 <<30)
HUB_MGR_IF_STATUS_END (1 <<31)

5.7.3. Sensor Data Output

The HUB manager stores sensor data output in the HW FIFO using the following data formats.

Table5-27 Sensor Data Output Format

Byte0	Byte1	Byte2	Byte3
num	sen_id *1	0x80	0xFF
time_stamp *2			
sen_data[0] *3			
:			
sen_data[num - 2]			

*1 : sensor ID

*2 : time stamp @tick (tick = 1 msec)

*3 : sensor data of sensor specified by sen_id

5.8. Handling of Sensor Data

The HUB manager notifies the Host CPU of sensor data for sensors which are activated from the Host CPU.
The basic processing flow is as follows:

(1) If the sensor that is activated by the Host CPU is a physical sensor, the physical sensor library acquires sensor data from the physical sensor via the device driver.
(2) The physical sensor library notifies the HUB manager of that sensor data.
(3) If the sensor that is activated by the Host CPU is a virtual sensor, the virtual sensor library acquires sensor data necessary for that virtual sensor from the physical library or another virtual sensor library. If the virtual sensor library requires data from multiple physical or other virtual sensors, the virtual sensor library acquires that data from each physical or other virtual sensor libraries.
(4) The virtual sensor library performs necessary operations for the sensor data acquired from the physical or other virtual sensor libraries and notifies the HUB manager of the obtained sensor data.
(5) The HUB manager writes the sensor data obtained from each sensor library to FIFO, and then the Host CPU reads FIFO through the SPI to notify the Host CPU of sensor data.

6. Physical Interfaces with Physical Sensors

frizz can connect to sensor devices through I2C (master), SPI (master), or UART.

6.1. I2C Master

frizz can connect sensors that are I2C slave devices.
The SCL of the I2C can be set with frequency divisions of multiples of 1/5 of the frizz operation clock.

6.2. SPI Master

frizz can connect up to 4 sensors that are SPI slave devices.
However, because the chip select is used in combination with GPIO, if the GPIO pins are assigned to another function, the number of SPI slave sensors that can be connected is reduced by that amount.
SPI mode can be set to mode 0 or mode 3.
SCKL of SPI can be set with frequency divisions of multiples of 1/6 of the frizz operation clock.

6.3. UART

frizz can connect one sensor that has a UART interface.
The baud rate can be set with frequency divisions of multiples of 1/5 of the frizz operation clock, and frizz can perform HW flow control, CTS interruptions, transmission buffer empty interruptions, data receive interruptions, error detect interruptions, parity settings, stop bit settings, and data bit count settings.

7. Boot Mechanism

frizz does not have an internal Flash ROM, and thus it is necessary to boot up frizz to transmit frizz firmware to the frizz internal RAM from an external memory.

The frizz firmware can be sent to the frizz internal RAM by the following 2 methods.

(1) Transmission from an external SPI Flash ROM
(2) Transmission from the Host CPU through a serial interface

Each of these transmission methods is described below.

7.1. Transmission from an External SPI Flash ROM

By storing the frizz firmware in a predetermined format in the SPI Flash ROM of an SPI slave and connecting it to the SPI master of frizz, frizz can expand the frizz firmware in its internal frizz RAM from an SPI Flash ROM and boot up.

7.2. Transmission from the Host CPU through a Serial Interface

By transmitting the frizz firmware from the Host CPU to frizz (I2C/SPI slave device) through I2C or SPI, frizz can expand the frizz firmware in its internal RAM and boot up.

8. Revision History

Date	Revision	Description
JAN,01,2015	1.0	Initial Release