Getting started with the IO-Link evaluation solution firmware for STEVAL-IDP004V2 and STEVAL-IDP003V1

Introduction

The IO-Link evaluation system is supported by two different firmware package versions, both embedding a layer to communicate with a PC.

The first version is based on a customized protocol that allows communication between STEVAL-IDP004V2 (master board) and STEVAL-IDP003V1 (device board) and sends dedicated string commands via user interface based on a generic serial terminal.

The second version is based on an IO-Link protocol stack, developed by TEConcept GmbH, which manages communication between STEVAL-IDP004V2 and STEVAL-IDP003V1, and uses a protocol developed by the same provider, with a user interface based on a proprietary GUI (Control Tool).

Figure 1. STSW-IO-LINK firmware structure without stack

Note: The STEVAL-IDP004V2 has the IO-Link stack embedded on the on-board STM32. The protocol stack is provided by TEConcept GmbH and comes with some limitations, as detailed further on. Be aware that the STSW-IO-LINK implements only the physical layer without stack. Programming it into the microcontroller you lose the full stack. This process is irreversible.

Figure 2. Firmware structure with stack

Related links

Visit the STM32Cube ecosystem web page on www.st.com for further information

1 STEVAL-IDP004V2

1.1 STSW-IO-LINK firmware package description

The firmware structure is created with the STM32CubeMX tool, which lets you configure the microcontroller and generate the corresponding code repeatedly, without losing any programming code developed to handle the system.

The generated project solution has been compiled for two different PC interfaces, RS485 and USB.

Figure 3. PC communication via RS485 interface

Figure 4. PC communication via USB interface

The project is structured as follows:

- **Application level**
  - EWARM folder: containing the startup file
  - USER folder: containing main and interrupt routines
- **Driver level**
  - CMSIS folder: containing source file library for the microcontroller core
  - STEVAL-IDP004V2 folder: customized folder based on the application
    - **PC_Communication_RS485.c**: source files to manage RS485 communication
    - **Master_Settings.c**: to configure L6360 IC
    - **Master_DeviceCOMM.c**: to manage data exchange with sensor node
  - STM32F2xx_HAL_DRIVERS: containing all HAL library source files for internal peripherals, memory and GPIO handling
- **Middlewares**
  - USB source files to enable interface with PC (enabled only with USB interface)

1.1.1 Application level

The application level contains the source files to configure the microcontroller and provide a front-end layer to manage the process flow.

From this level, you can recall the routines in the Driver Level to handle different parts of communication.

The source files used in this layer are:

- **Main.c**: groups all the routines used to initialize the peripherals and to manage the access to the data processing routines
- **stm32f2xx_it.c**: manages the systick interrupt for PC communication time out and peripheral interrupts
- **stm32f2xx_hal_msp.c**: source file for NVIC configuration and to enable peripherals

1.1.2 Driver level

This level contains the HAL, CMSIS or user library routines used to handle microcontroller settings, data processing and user command management.

1.1.2.1 PC_Communication_RS485.c

This source file is used for STEVAL-IDP004V2 configuration and data exchange with the STEVAL-IDP003V1D board hosting the L6360 IC to manage communication with a PC via Tera Term (or HyperTerminal interface).

The file includes the commands and string messages to be exchanged during communication with the PC and the sensor node, and the routines necessary for communication management.

Table 1. Routines for communication with PC

Routines	Description
`void User_Command (void)`	Called at the main level when a frame has been received from the PC.
`void RS485_Decode(void)`	To decode commands received from the PC.
`void RS485_TX (uint8_t* RS485_data,uint8_t data_size)` `void RS485_RX (uint8_t data_size)`	To manage RS485 communication.
`void Convert_ToAscii` `(SlaveNodeStruct_Typedef* Sdata,uint8_t n_byte,uint8_t node)`	To convert the sensor data into ASCII format and send the result to the PC.
`void Reset_RS485_COM (void)`	To reset PC communication if a command is wrong.
`void Drive_CommPin(uint8_t transm)`	To manage the logic level on the DE pin of the RS485 transceiver.

Routines	Description
`void Print_Sensors_Info(void)`	To transmit the sensor ID and parameter name to the PC for each sensor.
`void PrintRegister (uint8_t* string)`	To transmit the L6362A register value to the PC after a read command.
`void I2C_Print_FB (void)`	To transmit feedback related to an I²C operating mode.

Note: In the USB version, the `RS485_TX`, `RS485_RX`, `RS485_Decode routines` become `USB_TX`, `USB_RX`, `RX_USB_Decode` and the source file is PC_Communication_USB.c.

Table 2. String messages returned from STEVAL-IDP004V2 to PC

String message	Description

`uint8_t SpaceString[]="";`
`uint8_t PresentationMessage0[]="**********\r\n";`
`uint8_t PresentationMessage1[]="* STMicroelectronics  *\r\n";`
`uint8_t PresentationMessage2[]="* IO - Link MULTI-PORT SOLUTION *\r\n";`
`uint8_t PresentationMessage3[]="**********\r\n";`
`uint8_t StrRequest[]="\r\nSELECT MASTER OR DEVICE\r\n";`
	Sent back to the PC after entering the [**START**] command.

`uint8_t StrRequestIC []="\r\nINSERT ADDRESS IC :\r\n"`	Sent back to the PC after specifying the IC (Master), using a single digit from 0 to 3.
`uint8_t StrRequestOPM[]="\r\nINSERT OPERATING MODE :\r\n"`	Sent back to the PC after specifying the IC address if the previously selected IC was Master. You must specify one of the following user commands: `WR_C`, `R_S`, `RD_C` or `RD_S`.[(1)]
`uint8_t StrRequestRADRS []="\r\nINSERT REGISTER ADDRESS :\r\n"`	Sent back to the PC after specifying the operating mode via user commands (message available only for master programming). Insert [(1)]: • 1 if the user command is `WR_S` • a number between 1 and 7 if the user command is `WR_C` register address • no number if the user command is `RD_C` • 0 if the user command is `RD_S`
`uint8_t StrRequestRVAL []="\r\nINSERT REGISTER VALUE :\r\n"`	Sent back to the PC (if writing commands were chosen), after specifying the address register value. Insert a three-digit sequence for each value in the range 000 to 255 and separate each register value using the ‘,’ (comma) symbol. Complete the sequence with a ‘,’ even if the value is only one.

String message	Description

`uint8_t Status[]="\r\nStatus register";`
`uint8_t Configuration[]="\r\nConfiguration register";`
`uint8_t Control1[]="\r\nControl 1";`
`uint8_t Control2[]="\r\nControl 2";`
`uint8_t LED_1H[]="\r\nLED 1 MSB";`
`uint8_t LED_1L[]="\r\nLED 1 LSB";`
`uint8_t LED_2H[]="\r\nLED 2 MSB";`
`uint8_t LED_2L[]="\r\nLED 2 LSB";`
`uint8_t Parity[]="\r\nParity Register";`
	Sent back to the PC if a read operation was selected with the user command: if the user command is `RD_C`, only the first string is sent; if the user command is `RD_R`, the whole string is sent on the basis of the start address.

`uint8_t MS_StrRequestADDR[]="\r\nINSERT SLAVE NODE :\r\n"`	Sent back to the PC after specifying the IC (Device) and entering the node address with a single digit from 0 to 3. This digit identifies the corresponding master node and the USART port selected for communication.
`uint8_t MS_StrRequestCMD[]="\r\nINSERT SENSOR COMMAND :\r\n"`	String message sent back to the PC after the slave node is inserted. Write one of the user commands listed in PC_Communication.c.
`uint8_t MS_StrFeedbackCMD[]="\r\nNO SENSOR :\r\n"`	Sent back to the PC if the sensor is not connected.
`uint8_t MS_StrFeedbackMSG[]="\r\nPROGRAMM MASTER NODE BEFORE COMMUNICATION :\r\n"`	Sent back to the PC if the node selected for the master device communication is not programmed.

`uint8_t MS_StrPRX_Var[]="\r\nPOSITION :";`
`uint8_t MS_StrTMP_Var[]="\r\nTEMPERATURE :";`
`uint8_t MS_StrACL_Var[]="\r\nACCELERATION :";`
`uint8_t MS_StrVIBR_Var[]=”\r\nVIBRATION :";`
	Sent back to the PC when you send commands to the device node to retrieve parameter values. Depending on the sensor, one of these strings will be displayed before the number.

1. *Refer to Table 6. L6360 programming commands.*

Table 3. Commands for STEVAL-IDP004V2 and STEVAL-IDP003V1 communication

Command	Description
`uint8_t CmdSTART []="START"`	PC communication Init.
`uint8_t CmdWRC[]="WR_C"`	To send a request for L6360 Write Current procedure.
`uint8_t CmdRDC[]="RD_C"`	To send a request for L6360 Read Current procedure.
`uint8_t CmdWRS[]="WR_S"`	To send a request for L6360 Sequential Write procedure.
`uint8_t CmdRDS[]="RD_S"`	To send a request for L6360 Sequential Read procedure.
`uint8_t CmdICM[]="MASTER"`	To select master board device.
`uint8_t CmdICD[]="DEVICE"`	To select device board.
`uint8_t CmdNEW[]="COMMAND NEW"`	To send a request for a new programming phase.
`uint8_t CmdEND[]="COMMAND END"`	To send a request for closing the current programming phase.
`uint8_t MS_CmdID[9]="IDS"`	To send a request for getting sensor identification (this command relates to master-device communication).
`uint8_t MS_CmdPRM[9]="PRM"`	To send a request for getting the results of computation analysis on measured values (this command relates to master-device communication).

Table 4. Standard commands for master-device communication

String	Function	Description
START	It enables communication	To start PC to STEVAL-IDP004V2 communication.
MASTER	It selects the master IC	To select the master node to be addressed.
DEVICE	It selects the device IC	To select the device node to be addressed.
COMMAND NEW	It implements a new command	To inform the master board that a new programming procedure is starting on a node or a new query command is going to be sent to a node.
COMMAND END	It ends a command	To communicate the programming procedure or the query request running on a node is going to be stopped.

Table 5. Query commands for master-device communication

String	Function	Description
IDS	It discovers which sensor is used and its status	STEVAL-IDP004V2 sends a query command to the STEVAL-IDP003V1 to know which sensor is plugged and its status (connected or not) and forwards the information to the PC
PRM	It gets the measured parameter value	STEVAL-IDP004V2 sends a query command to get the measured parameter (i.e. distance) and forwards it to the PC

Table 6. L6360 programming commands

Type	String	Description
Write Current	WR_C	Only the specified register is programmed.
Write Sequential	WR_S	All registers are written starting from the configuration register.
Read Current	RD_C	Only status and parity registers are read.
Read Sequential	RD_S	All registers are read starting from the status register.

1.1.2.2 *Master_Setting.c*

This source file groups all the routines used for master access mode (writing or reading) and parity calculation.

Table 7. Master_Setting.c routines

Routines	Description
`void Master_Programming(void)`	To address the L6360 programming procedure.
`int8_t Master_CurrentWrite (uint8_t M_address,DirectionMode_Typedef I2C_DIR, uint8_t* master_reg, uint8_t register_pos)`	To perform the single writing access procedure to the master IC.
`int8_t Master_SequentialWrite (uint8_t M_address,DirectionMode_Typedef I2C_DIR, uint8_t* master_reg,uint8_t start_pos)`	To perform the sequential writing access mode; in this case, the starting register address is one, corresponding to the configuration register.
`uint8_t Get_MasterStatus (uint8_t M_address,uint8_t start_pos,uint8_t* get_reg)`	To read the status register for getting feedback on the device and IO-Link bus failures.
`uint8_t Get_MasterConfig (uint8_t M_address,DirectionMode_Typedef I2C_DIR, uint 8_t* master_config, uint8_t start_pos)`	To perform sequential or random register reading for retrieving the programmed register.
`uint8_t WC_Parity_Calc(uint8_t in_data)`	To calculate the parity byte to be sent when a Current Write operation is required.
`uint8_t WS_Parity_Calc(uint8_t* in_data)`	To calculate the parity byte to be sent when a Sequential Write operation is required.

Routines	Description
`void Master_Reset(uint8_t reset)`	To reset all master ICs.
`int8_t Master_EvaluateStatus(uint8_t node)`	To get feedback on master node programming status (programmed or not).
`void Master_DefaultStatus (void)`	To set the default L6360 configuration.

1.1.2.3 *Master_DeviceCOMM.c*

This source file has all the routines used to handle the IO-Link physical layer communication.

Table 8. Master_DeviceCOMM.c routines

Routines	Description
`int8_t IO_LINK_SEND (uint8_t address, uint8_t* data_byte,uint8_t txdata_lenght, uint8_t rxdatalength)`	To manage IO-Link physical layer communication.
`void Master_TxMode(uint8_t address)`	To handle the ENCQ pin in transmission mode.
`void Master_RxMode(uint8_t address)`	To handle the ENCQ pin in receive mode.
`int8_t Get_Slave_FB(uint8_t var,uint8_t length)`	To decode the information received by the device after a user command.
`int8_t Get_ReceiverFlag(void)`	To get the receiver flag value.
`void Master_LPlus_LineOn(uint8_t lplus)`	To enable the supply line for the sensor.
`void Reset_ReceiverFlag(void)`	To reset the receiver flag.
`void Send_Waiting_For_Command(void)`	To send a frame to the slave to signalize that a command will be sent.
`void Send_Command(void)`	To send the command requested to the slave.

2 IO-Link stack implementation

This firmware package with IO-Link stack is available only in a compiled version.

It is available by default in the STEVAL-IDP004V2 with readout protection activated but you can also program the board with customized firmware by changing the option byte to Level 0 using the ST-LINK Utility tool for memory erase.

The stack embeds two different communication interfaces with PC, USB and RS485 (the type of interface is automatically recognized).

The version is fully featured but for a limited time (10000 minutes): after that, the system stops working and you need a new free key license from TEConcept, with the hardware ID readable using their proprietary Control tool, a graphical user interface (GUI) which enables communication with master unit and device.

The graphical representation of the data sensor changes according to the IODD file related to the sensor, which can be downloaded from the IO-Link consortium web page clicking on the IODD finder.

Figure 5. IO-Link consortium web page: IODD finder

2.1 How to use TEConcept Control Tool

The STEVAL-IDP004V2 with the master stack can be managed by the user via TEConcept Control Tool. Follow the procedure below to enable communication.

Step 1. Connect the STEVAL-IDP004V2 to the PC via USB or RS485 connector

Step 2. Provide the supply voltage to the board

Step 3. Open the Control Tool and select the virtual COM port related to the board in the Comm. port menu

Figure 6. TEConcept Control Tool: virtual COM port selection

Step 4. Click on the green button to start communication

Figure 7. TEConcept Control Tool: active connection

Step 5. Click on [**Select device**]

Figure 8. TEConcept Control Tool: device selection

Step 6. Click [**Import**] to import the relevant IODD file, and then double click on the IODD to select it

Step 7. Click [**IO-Link**] (on the right) to connect the node and display the plot

Figure 9. TEConcept Control Tool: node connection

Figure 10. TEConcept Control Tool: data plot

3 STEVAL-IDP003V1 firmware structure

The firmware structure has been created using STM32CubeMx, which allows microcontroller configuration and code generation.

As detailed in the introduction, the firmware package has two different releases: the first one is STSW-IO-LINK with customized protocol available in the master board and the other one for the IO-Link stack.

3.1 STSW-IO-LINK firmware package description

The generated project solution has the following structure:

- **Application level**
  - EWARM folder: containing the startup file
  - USER folder: containing main routines and interrupt routines
- **Driver level**
  - BSP
    - Components: containing all the source files used for sensor programming and data processing
    - STEVAL- IDP003V1D: containing the routines used for IO-Link communication (the protocol stack is not implemented)
  - CMSIS folder: contains the source file library for the microcontroller core
  - STM32F2xx_HAL_DRIVERS: containing all the HAL library source files for internal peripherals, memory and GPIO handling
- **Middleware**: containing all the routines used for Fast Fourier Transform (FFT) calculations

3.1.1 Application level

This level contains the source files used to configure the microcontroller and provide a front-end layer to manage the process flow.

It allows you to call the routines in the Driver Level to handle different parts of the communication.

Main.c

This source file contains the routines used to initialize the peripherals:

- master-device IO-Link communication
- sensor configuration
- interrupt for sensor data ready when the MEMS sensor is connected

stm32f2xx_it.c

This source file handles interrupt routines based on the connected sensor.

stm32f2xx_hal_msp.c

Source file for NVIC configuration and peripheral enable.

3.1.2 Driver level

This level contains the HAL library and user library routines used to handle the microcontroller settings, communication and sensor data acquisition and processing.

3.1.2.1 Components section

This section implements sensor configuration and data processing. In particular, the *Sensor.c* source file is used to address all sensors available in the kit, implementing a set of functions differentiated by sensor type:

- *VL6180X.c* for VL6180X proximity sensor management

Table 9. VL6180X configuration and data processing functions

Function	Description
`int8_t VL6180x_WaitDeviceBooted(uint8_t dev)`	For device boot at startup.
`HAL_StatusTypeDef VL6180X_DeInit(void)`	Sensor DeInit.
`int8_t VL6180X_Calibration_Offset(uint8_t dev)`	Sensor calibration.
`void VL6180X_StartStop_Acquisition(uint8_t value)`	Sensor data sampling.
`int8_t VL6180X_Read_Register (uint8_t dev,uint16_t reg_addr,uint8_t* var,uint8_t nbyte)`	To perform a register access.
`uint8_t VL6180X_GetStatus_Error(uint8_t* data)`	To store error code in the sensor structure.
`int8_t VL6180X_GetRange(void)`	To read the data register.
`uint8_t VL6180X_GetStatus_Error (uint8_t* data)`	To get the status error code.
`in8_t VL6180X_Read_Status_Data (void)`	To perform the IC read access to read error code.

- *IIS2DH.c* for IIS2DH vibration sensor management

Table 10. IISDH configuration and data management processing functions

Function	Description
`int8_t IIS2DH_Reboot(uint16_t Device)`	Sensor reboot routine.
`int8_t IIS2DH_Init (uint16_t Device,uint8_t* Reg_Data,uint16_t num_byte)`	Sensor Init.
`int8_t IIS2DH_AxisOnOff (uint16_t Device,uint8_t* Reg_Data,uint16_t num_byte)`	Switch ON/OFF the axis before FFT calculation.
`uint8_t IIS2DH_Get_Sensitivity (uint16_t Device)`	To read the sensitivity value of the sensor.
`uint8_t IIS2DH_Get_ODR(uint16_t Device)`	To read ODR value of the sensor.
`int8_t IIS2DH_Evaluate_ODR(uint16_t value)`	To store the calculated ODR value in the sensor structure.
`int8_t IIS2DH_Get_Axes(uint16_t Device)`	To manage data acquisition.
`int8_t IIS2DH_Get_Axes_H(uint16_t Device)`	To read the high part of the data register.
`int8_t IIS2DH_Write_Data (uint16_t Device,uint8_t* Register,uint16_t num_byte)`	Called when a write sensor access is performed.
`int8_t IIS2DH_Read_Data (uint16_t Device,uint8_t* Register,uint8_data_out,uint16_t num_byte )`	Called when a read sensor access is performed.
`void IIS2DH_FFT_Calling(float32_t* buffer)`	To call the FFT calculation function and change the axis.
`uint8_t Get_FlagFFTdone (void)`	To get FFT flag status.

- *IIS328DQ.c* for IIS328DQ accelerometer management

Table 11. IIS328DQ configuration and data management processing functions

Function	Description
`int8_t IIS328DQ_Reboot(uint8_t Device)`	Sensor reboot.
`int8_t IIS328DQ_Init (uint8_t Device,uint8_t* Reg_Data,uint8_t num_byte)`	Sensor Init.
`uint8_t IIS328DQ_Get_Sensitivity (uint8_t Device)`	Sensor get sensitivity.
`uint8_t IIS328DQ_Get_ODR(uint8_t Device)`	Sensor get ODR.
`int8_t IIS328DQ_Get_Axes(uint8_t Device)`	To manage data acquisition.
`int8_t IIS328DQ_Write_Data (uint8_t Device,uint8_t* Reg_Data,uint8_t num_byte)`	Called when a write sensor access is performed.
`int8_t IIS328DQ_Read_Data (uint8_t Device,uint8_t* Reg_Data,uint8_t* data_out,uint8_t num_byte)`	Called when a read sensor access performed.
`void IIS328DQ_Change_Axis(void)`	Called to switch from one axis to another.

- *STTS751.c* for STTS751 temperature sensor management

Table 12. STTS751 configuration and data management processing functions

Function	Description
`int8_t STTS751_Configuration (uint8_t IC_Addr,uint8_t* conf_value,uint8_t num_reg)`	Sensor Initialization.
`int8_t STTS751_Get_Temperature (uint8_t IC_Addr,uint8_t* Temp_value)`	Temperature acquisition.
`int8_t STTS751_Get_Status (uint8_t IC_Addr,uint8_t reg_status)`	Sensor data ready status.

3.1.3 STEVAL-IDP003V1D

Master_DeviceCOMM.c

This source file contains all the routines used for PHY management and data packaging and decoding.

Table 13. PHY management and data packaging/decoding routines

Routine	Description
`void L6362A_IO_Init( uint8_t IO_Link_bus)`	To allow GPIOs configuration for EN/DIAG and OL diagnostic pin of L6362A.
`void BSP_GPIS_ENDIAG (uint8_t status)`	To allow EN/DIAG pin driving for TX/RX procedure.
`uint8_t Frame_BUS_Decode(uint8_t* bus_data)`	To decode the command frame type (i.e. GET_SENSOR_TYPE).
`void IO_Link_Data_Manager(void)`	To allow data packaging.
`void IO_LINK_SEND(uint8_t * dataTX,uint8_t lenght)`	To send frames.
`void IO_LINK_RECEIVE(uint8_t * dataRX,uint8_t lenght)`	To receive frames.

3.1.4 Middleware

This level contains the routines implemented using the DSP library to perform the FFT calculations when the vibration sensor is connected to the STEVAL-IDP003V1D board. All the routines are grouped in the FFT.c source file.

FFT.c

The routines implemented in this source file are related to FFT calculation using the DSP library as well as the final vibration frequency calculation and its amplitude for each axis.

Table 14. FFT.c routines

Routine	Description
`void FFT_AclCalc (float32_t* dataIN_tmp,uint8_t axis_num)`	To calculate the FFT output array.
`void Evaluate_MaxAmplitude_Range (FFT_Out_TypedefStruct* FFT_Output,uint16_t fft_psize,uint8_t axis_num)`	To perform a windowing process for finding the component with a maximum value in a defined window during calibration phase and create a reference mask for noise.
`void Calculate_ABS_FFTprm (FFT_Out_TypedefStruct* FFT_Output,uint16_t fft_psize)`	To perform a comparison in RUN between the real-time components and the mask obtained with the previous routine.
`void Print_FFT (FFT_Out_TypedefStruct* FFT_Output,uint16_t fft_psize,uint8_t axis_num)`	To calculate the real vibration frequency and its amplitude in m/s² for each axis and stores the value in the sensor structure.

3.2 Firmware package with IO-Link stack communication

This package integrates the IO-Link stack library developed by TEConcept within the IO-Link consortium.

The stack is provided in a compiled version (iol_ds_as_lib.a) and with customizable APIs available in the header file (iold_api.h).

The stack code embedded is limited in execution time and features. Time limitation prevents stack use for more than 180 minutes; when this time expires, a hardware reset is necessary to restart the application. Feature limitation, instead, is related to missed implementation of sensor data storage, block transfer and BLOB.

The firmware architecture is based on the STM32Cube Ecosystem guidelines with an *Application* layer, a *Middleware* layer, a *Drivers* layer including BSP (board support package) and a HAL driver library.

Figure 11. STEVAL-IDP003V1 stack architecture

- **Application**
  - EWARM: containing the startup file
  - USER: routines to support
    - device application firmware
      - peripheral configuration
      - interrupt configuration
      - Device Application
    - third party stack APIs
      - IO-Link application
      - main
      - event handler
      - IO-Link callback
- **Drivers**
  - BSP: routines to support components handling
    - ST
      - Components: source files used for sensor programming and data processing
      - CMSIS: source file library for the microcontroller core plus the DSP library files
      - STEVAL-IDP003V1: source files for sensor data acquisition handling
      - STM32L0xx_HAL_DRIVERS: HAL library source files for internal peripherals, memory and GPIO handling
    - third party
      - Components: source files for L6362A PHY management
- **Middleware**
  This level implements all the functions to support vibration power spectrum, inclination calculation and stack library
  - ST: routines used for vibration power spectrum and inclination calculation when using IIS328DQ
  - third party: IO-Link stack library in compiled format

3.2.1 Application

This layer includes the source files useful to manage all the calls forwarded to the lower layers for:

- IO-Link stack library
- IO-Link PHY management
- sensor initialization
- sensor data acquisition and processing

All the files have been structured in different folders:

- ST
  - **Device application .c** for
    - peripheral initialization (i.e. `Application_Init ()→MX_I2C1_Init()`)
    - sensor configuration (i.e. `Application_Init ()→Sensor_Init()`
    - sensor data acquisition and IO-Link data transmission (i.e. `void application_main_loop (void)`)
    - callback for sensor data acquisition, only when MEMS sensors are used (i.e. `void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)`)
  - **stm32f2xx_it.c** for the interrupt routines for SysTick_Handler or external interrupt for sensor data acquisition (i.e. `void SysTick_Handler(void)`,`void EXTI4_15_IRQHandler(void)`)
  - **stm32f2xx_hal_msp.c** for NVIC configuration and GPIO peripherals configuration. (i.e. `void HAL_I2C_MspInit(I2C_HandleTypeDef* hi2c) , void HAL_MspInit(void)`)
- third party
  - **Application.c** for creating a connection between the stack and the device application layer
    - parameter stack declaration
    - variable definition and initialization for communication
    - call function for the device application (i.e. `void device_app_hw_init(void)`)
  - **Eventhandler.c** for event signalization related to device status, active/inactive event (i.e. `uint8_t eh_get_detailed_device_status(uint8_t *dest, const uint8_t max_cnt) ; void eh_cyclic(void)`)
  - **Iold_app_callbacks.c** to process data (PDIN, PDOUT, ISDU)
  - **main.c** for
    - stack management (i.e. `iol_stack_register_callbacks() ; iol_stack_initialization()`)
    - direct parameter page (i.e. `iold_api_create_dpp1 (MIN_CYCLETIME, MSEQ_CAPABILITY, IOL_REVISION_ID, PDIN_PARAM, PDOUT_PARAM, VENDOR_ID, IOL_DEVICE_ID)`)
    - device application (i.e. `device_app_hw_init()`)
    - enable interrupt (i.e.`enableINT()`)
  - **phal_stm32l0xx.c** for
    - peripheral setup linked to the PHY IC (i.e. `void phal_setup_txen(void) ; void phal_set_rx_pin_mode(E_RX_MODE_t mode)`)
    - wakeup request interrupt (i.e. `void phal_enable_wurq_interrupt()`)
    - data interrupt setup enable/disable and interrupt configuration for IO-Link communication (i.e. `void phal_enable_data_interrupt(); void phal_disable_data_interrupt(); void phal_data_interrupt_setup()`)
  - **phal_stm32l0xxirq.c** for
    - interrupt routine for timer event (i.e. `void IOL_TIMER_IRQHandler(void)`)
    - UART data communication interrupt (i.e. `void IOL_UART_IRQHandler(void)`)
    - Wakeup request interrupt (i.e. `void WURQ_IRQHandler(void)`)

3.2.2 BSP

- *VL6180X.c*

Table 15. VL6180X.c board support package routines for component handling

Routine	Description
`int8_t VL6180x_WaitDeviceBooted(uint8_t dev)`	Device boot at start up.
`HAL_StatusTypeDef VL6180X_DeInit(void)`	Sensor DeInit.
`int8_t VL6180X_Calibration_Offset(uint8_t dev)`	Sensor calibration.
`void VL6180X_StartStop_Acquisition(uint8_t value)`	Sensor data sampling.
`int8_t VL6180X_Read_Register(uint8_t dev,uint16_t reg_addr,uint8_t* var,uint8_t nbyte)`	To perform a register access.
`uint8_t VL6180X_GetStatus_Error(uint8_t* data)`	To store errore code in the sensor structure.

- *IIS328DQ.c*

Table 16. IIS328DQ.c board support package routines for component handling

Routine	Description
`int8_t IIS328DQ_Reboot(uint8_t Device)`	Sensor reboot.
`int8_t IIS328DQ_Init(uint8_t Device,uint8_t* Reg_Data,uint8_t num_byte)`	Sensor Init.
`uint8_t IIS328DQ_Get_Sensitivity (uint8_t Device)`	Sensor get sensitivity.
`uint8_t IIS328DQ_Get_ODR(uint8_t Device)`	Sensor get ODR.
`int8_t IIS328DQ_Get_Axes(uint8_t Device)`	To manage data acquisition.
`int8_t IIS328DQ_Write_Data(uint8_t Device,uint8_t* Reg_Data,uint8_t num_byte)`	Called when a write sensor access is performed.
`int8_t IIS328DQ_Read_Data (uint8_t Device,uint8_t* Reg_Data,uint8_t* data_out,uint8_t num_byte)`	Called when a read sensor access is performed.
`void IIS328DQ_Change_Axis(void)`	Called when it is necessary to switch from an axis to another.

- *IIS2DH.c*

Table 17. IIS2DH.c board support package routines for component handling

Routine	Description
`int8_t IIS2DH_Reboot(uint16_t Device)`	Sensor reboot.
`int8_t IIS2DH_Init(uint16_t Device,uint8_t* Reg_Data,uint16_t num_byte)`	Sensor Initialization.
`int8_t  IIS2DH_AxisOnOff(uint16_t Device,uint8_t* Reg_Data,uint16_t num_byte)`	To switch the axis off before FFT calculation.
`uint8_t IIS2DH_Get_Sensitivity (uint16_t Device)`	To read sensor sensitivity value.
`uint8_t IIS2DH_Get_ODR(uint16_t Device)`	To read sensor ODR value.
`int8_t IIS2DH_Evaluate_ODR(uint16_t value)`	To store the calculated ODR value in the sensor structure.
`int8_t IIS2DH_Get_Axes(uint16_t Device)`	To manage data acquisition.
`int8_t IIS2DH_Get_Axes_H(uint16_t Device)`	To read the high part of the data register.
`int8_t IIS2DH_Write_Data(uint16_t Device,uint8_t* Register,uint16_t num_byte)`	Called when a write sensor access is performed.
`int8_t IIS2DH_Read_Data (uint16_t Device,uint8_t* Register,uint8_t data_out,uint16_t num_byte)`	Used when a read sensor access is performed.
`void IIS2DH_FFT_Calling(float32_t* buffer)`	To call the FFT calculation function and change the axes.
`uint8_t Get_FlagFFTdone (void)`	To get FFT flag status.

- *STTS751.c*

Table 18. STTS751.c board support package routines for component handling

Routine	Description
`int8_t STTS751_Configuration(uint8_t IC_Addr,uint8_t* conf_value,uint8_t num_reg)`	Sensor Initialization.
`int8_t STTS751_Get_Temperature(uint8_t IC_Addr,uint8_t* Temp_value)`	Temperature acquisition.
`int8_t STTS751_Get_Status(uint8_t IC_Addr,uint8_t reg_status)`	Sensor data ready status.

3.2.3 Middleware

3.2.3.1 *ST*

This layer contains all the routines used for:

- power spectrum evaluation using FFT (and DSP library) when vibration sensor is enabled
- pitch and roll angle calculation (using DSP library when accelerometer is enabled)
- IO-Link stack library in object format

FFT.c

Table 19. FFT.c routines for power spectrum evaluation

Routine	Description
`void FFT_AclCalc (float32_t* dataIN_tmp,uint8_t axis_num)`	To calculate the FFT output array.
`void Evaluate_MaxAmplitude_Range (FFT_Out_TypedefStruct* FFT_Output,uint16_t fft_psize,uint8_t axis_num )`	To perform a windowing process to find the component with a maximum value in a defined window to create a reference mask for noise (during calibration phase)
`void Calculate_ABS_FFTprm(FFT_Out_TypedefStruct* FFT_Output,uint16_t fft_psize)`	To calculate the right vibration frequency and its amplitude in m/s2 for each axis and store the value in the sensor structure.

Inclination.c

Table 20. Inclination.c routines

Routine	Description
`void IIS328DQ_AngleCalc(SENSOR_StructTypedef Acc_Amp)`	To pitch and roll angle calculation using DSP library.

Figure 12. Inclination

3.2.3.2 *Third party library*

The library implements a set of functions to support the requirements fixed by the consortium:

- stack initialization
- SIO mode configuration
- direct parameter pages access
- ISDU mode
- data storage (not implemented in this stack version)
- process data mode (only Process Data IN is implemented in this stack version)
- event handling
- system command API
- system management

- BLOB data transfer (not implemented in this stack version)

This folder contain the *iol_ds_as_lib.lib* file, which contains all the APIs useful to manage IO-Link data communication.

Dedicated header files list functions and constants and can be managed according to application needs.

The *iold_api.h* file lists all APIs necessary to address the routines in the library.

Table 21. Middleware iold_api.h file functions

Function	Description
`iol_stack_initialization()`	To initialize hardware and software modules.
`sio_api_set_sio_mode (E_SIO_PNP_MODE)`	To set device in SIO mode.
`sio_api_set_sio_level (E_SIO_LOW_LEVEL)`	To set SIO output level on the bus.
`iold_api_create_dpp1(MIN_CYCLETIME, MSEQ_CAPABILITY, IOL_REVISION_ID, PDIN_PARAM, PDOUT_PARAM, VENDOR_ID, IOL_DEVICE_ID)`	`MIN_CYCLETIME` defines time interval for cyclic communication between master and device. `PDIN_PARAM` defines the amount of data the device sends back to the master. `PDOUT_PARAM` defines the amount of data the master sends to the device.
`iold_pde_api_setPDvalidity(E_VALIDITY_VALID)`	For asynchronous data transfer.
`iold_pde_api_setpdin(TxIOLINK_FRAME)`	To set data transmission value.

Table 22. Middleware header files defining constants

Function	File
`Stack parameters(i.e. MAX_PDIN_LEN , PDIN_PARAM, USED_MSEQ_TYPE,…)`	iol_stack_config.h
`Stack components (i.e.FW_UPDATE_TFW, _WITH_SIO, _WITH_ISDU,..)`	iol_stack_components.h
`Hardware specific configuration`	It contains the processor pin configuration, clock, IOL-PHY and timer setup.

Revision history

Table 23. Document revision history

Date	Version	Changes
07-Jun-2017	1	Initial release.
05-Sep-2017	2	Added Figure 1. STSW-IO-LINK block diagram.
26-Jun-2018	3	Updated Figure 1. STSW-IO-LINK block diagram, Section 2.3.1.2 Commands for STEVAL-IDP004V1 and STEVAL-IDP003V1 communication, Section 2.3.1.3 Routines used to manage the communication with PC and Section 2.3.3 Master_DeviceCOMM.c. Added Section 3.3.1 Master_DeviceCOMM.c description.
16-Oct-2019	4	Updated Introduction, Section 1.1 STSW-IO-LINK firmware package description, Section 3 STEVAL-IDP003V1 firmware structure, Section 3.1.2.1 Components section and Section 3.1.3 STEVAL-IDP003V1D. Added Section 1.1.2.1 PC_Communication_RS485.c, Section 1.1.2.2 Master_Setting.c, Section 1.1.2.3 Master_DeviceCOMM.c, Section 2 IO-Link stack implementation, Section 2.1 How to use TEConcept Control Tool, Section 3.1 STSW-IO-LINK firmware package description, Section 3.2 Firmware package with IO-Link stack communication, Section 3.2.1 Application, Section 3.2.2 BSP, Section 3.2.3.1 ST and Section 3.2.3.2 Third party library.
01-Oct-2020	5	Updated Section 3.2: Firmware package with IO-Link stack communication.
12-Jul-2025	6	Updated Section Introduction.

Contents

1 STEVAL-IDP004V2 3
1.1 STSW-IO-LINK firmware package description. 3
1.1.1 Application level 4
1.1.2 Driver level 4
2 IO-Link stack implementation 9
2.1 How to use TEConcept Control Tool. 9
3 STEVAL-IDP003V1 firmware structure 13
3.1 STSW-IO-LINK firmware package description. 13
3.1.1 Application level 13
3.1.2 Driver level 13
3.1.3 STEVAL-IDP003V1D 17
3.1.4 Middleware 17
3.2 Firmware package with IO-Link stack communication 17
3.2.1 Application 19
3.2.2 BSP 20
3.2.3 Middleware 21
Revision history 24

List of tables

Table 1. Routines for communication with PC . . . 4
Table 2. String messages returned from STEVAL-IDP004V2 to PC . . . 5
Table 3. Commands for STEVAL-IDP004V2 and STEVAL-IDP003V1 communication . . . 6
Table 4. Standard commands for master-device communication . . . 7
Table 5. Query commands for master-device communication . . . 7
Table 6. L6360 programming commands . . . 7
Table 7. Master_Setting.c routines . . . 7
Table 8. Master_DeviceCOMM.c routines . . . 8
Table 9. VL6180X configuration and data processing functions . . . 14
Table 10. IISDH configuration and data management processing functions . . . 15
Table 11. IIS328DQ configuration and data management processing functions . . . 16
Table 12. STTS751 configuration and data management processing functions . . . 16
Table 13. PHY management and data packaging/decoding routines . . . 17
Table 14. FFT.c routines . . . 17
Table 15. VL6180X.c board support package routines for component handling . . . 20
Table 16. IIS328DQ.c board support package routines for component handling . . . 20
Table 17. IIS2DH.c board support package routines for component handling . . . 21
Table 18. STTS751.c board support package routines for component handling . . . 21
Table 19. FFT.c routines for power spectrum evaluation . . . 22
Table 20. Inclination.c routines . . . 22
Table 21. Middleware iold_api.h file functions . . . 23
Table 22. Middleware header files defining constants . . . 23
Table 23. Document revision history . . . 24

List of figures

Figure 1. STSW-IO-LINK firmware structure without stack . . . 1
Figure 2. Firmware structure with stack . . . 2
Figure 3. PC communication via RS485 interface . . . 3
Figure 4. PC communication via USB interface . . . 3
Figure 5. IO-Link consortium web page: IODD finder . . . 9
Figure 6. TEConcept Control Tool: virtual COM port selection . . . 10
Figure 7. TEConcept Control Tool: active connection . . . 10
Figure 8. TEConcept Control Tool: device selection . . . 11
Figure 9. TEConcept Control Tool: node connection . . . 12
Figure 10. TEConcept Control Tool: data plot . . . 12
Figure 11. STEVAL-IDP003V1 stack architecture . . . 18
Figure 12. Inclination . . . 22

IMPORTANT NOTICE – READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice.

In the event of any conflict between the provisions of this document and the provisions of any contractual arrangement in force between the purchasers and ST, the provisions of such contractual arrangement shall prevail.

The purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgment.

The purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of the purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

If the purchasers identify an ST product that meets their functional and performance requirements but that is not designated for the purchasers' market segment, the purchasers shall contact ST for more information.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, refer to www.st.com/trademarks. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2025 STMicroelectronics – All rights reserved