RoboKit Programmer's Guide

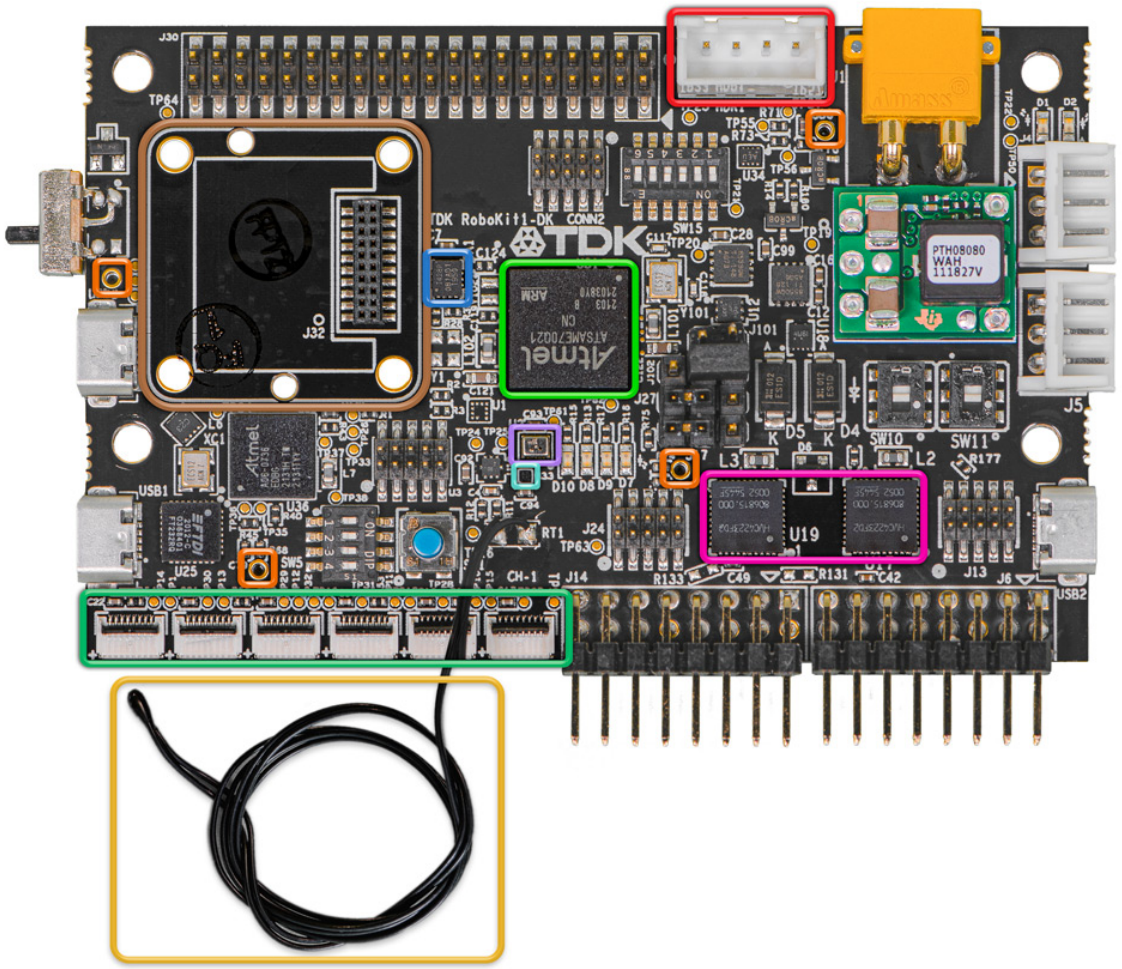
InvenSense, a TDK Group Company
1745 Technology Drive, San Jose, CA 95110 U.S.A
+1(408) 988–7339
invensense.tdk.com

Document Number: AN-000347
Revision: 1.1
Release Date: 09/21/2022

TABLE OF CONTENTS

1 INTRODUCTION

The purpose of this document is to give an overview of RoboKit software stack and its development environment, which allows the users to create new RoboKit applications and/or update existing applications. This document also serves as a quick start guide for the RoboKit software package and its elements.



🟡 Thermistor: B57861S0103A039	🔵 IMU: ICM-42688-P
🟢 MCU: ARM Cortex M7	🟠 Microphones: ICS-43434
🟣 Motor Controllers: HVC-4223F	🟣 Pressure Sensor: ICP-10111
🔴 CAN Connector	🟤 Industrial IMU: IIM-46230 Socket
🔵 Magnetometer: AK09918C	🟢 Ultrasonic Range Sensor Connectors: CH101/CH201

Figure 1. RoboKit1 Board

2 ARCHITECTURE

RoboKit1 has an onboard Atmel CORTEX-M7 MCU (ATSAME70Q21B) with sensors connected through various hardware peripherals. Figure 2 describes the hardware block diagram of MCU and its connections with various components on the board.

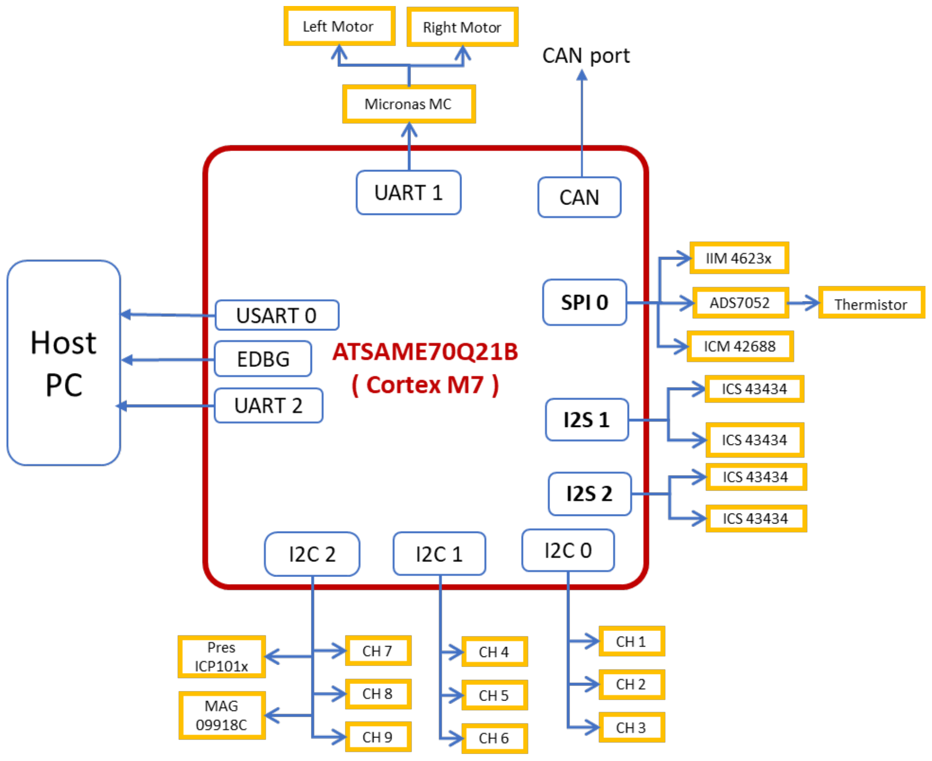
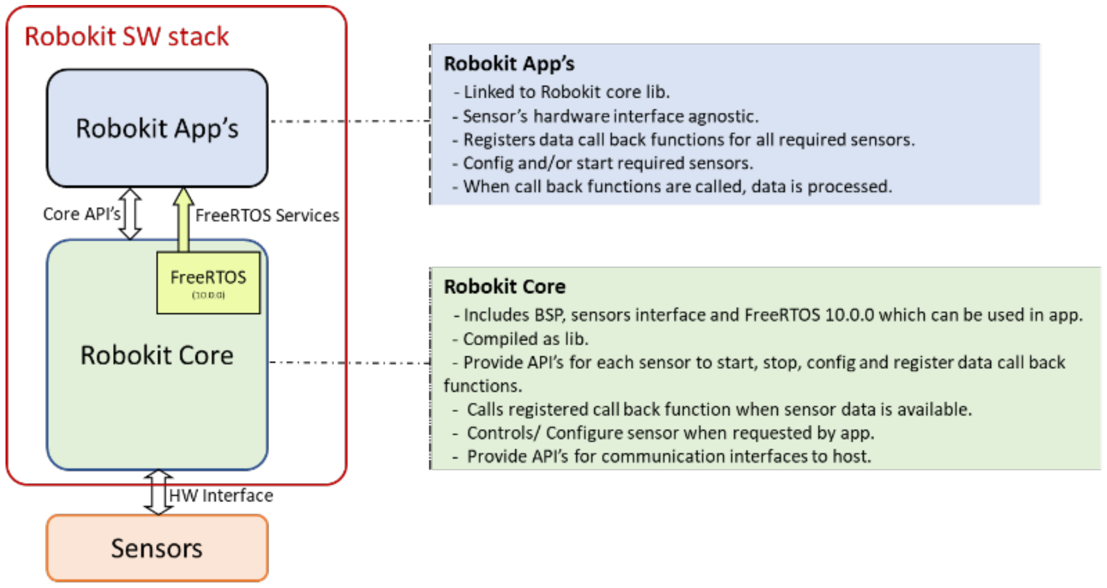


Figure 2. RoboKit1 HW block diagram

The RoboKit software stack has two layers, Robokit_core and Robokit_App. Robokit_core contains FreeRTOS, BSP and low-level code to configure and interface all the sensors on the board. It modularizes the code, which interacts with the hardware components and provides an abstraction layer API for the applications to start, stop, configure, and to register sensor data call back function for all the sensors on board. The application layer uses these APIs to implement specific use case scenarios. The released code implements two use case applications. ***"Robokit-App-Host-Interface"*** application controls the sensors and collects sensor data using an app running on the host computer and ***"Robokit-App-Smart-Bot"*** application controls the robot through voice commands. FreeRTOS operating system and Atmel's Advanced Software Framework (ASF) are used to help manage resources more efficiently.



Robokit App's

- Linked to Robokit core lib.
- Sensor's hardware interface agnostic.
- Registers data call back functions for all required sensors.
- Config and/or start required sensors.
- When call back functions are called, data is processed.

Robokit Core

- Includes BSP, sensors interface and FreeRTOS 10.0.0 which can be used in app.
- Compiled as lib.
- Provide API's for each sensor to start, stop, config and register data call back functions.
- Calls registered call back function when sensor data is available.
- Controls/ Configure sensor when requested by app.
- Provide API's for communication interfaces to host.

Figure 3. RoboKit SW Stack

3 ROBOKIT CORE

The RoboKit-Core contains drivers for all available sensors on the board, configuration files to configure the sensors, and a set of API functions to be called by the application layer. It allows easy development of application without the need to modify low-level code.

The primary function of RoboKit-Core is to configure the sensors, initialize them, and get the sensor data. Different modules in RoboKit-Core are organized as per the folder structure in Figure 4.

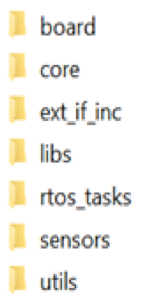

board
core
ext_if_inc
libs
rtos_tasks
sensors
utils

Figure 4. RoboKit Core Folder Structure

3.1 BOARD

This folder contains the Microchip Advanced Software Framework library. Drivers for all the interfaces to the MCU and the hardware abstraction layer are part of this library. The folder also contains FreeRTOS v10.0.0. The config file for FreeRTOS "FreeRTOSConfig.h" is located at Robokit-Core/board/config.

3.2 CORE

This folder contains the entry point for RoboKit-Core library. Application developer can enable or disable sensors by setting the compile time macros in the file "inv_platform.h." The available options are listed in Table 1.

ENABLE_MONITOR_TASK	0: Disable Monitor task, this task outputs debug messages on USB2.
	1: Enable Monitor task, this task outputs debug messages on USB2.
ENABLE_MAG_AK09918_SENSOR	0: Disable Mag sensor.
	1: Enable Mag sensor.
ENABLE_PRES_ICP10101_SENSOR	0: Disable pressure sensor.
	1: Enable pressure sensor.
ENABLE_TEMP_ADS7052_SENSOR	0: Disable temperature sensor.
	1: Enable temperature sensor.
ENABLE_IMU_ICM42688_SENSOR	0: Disable IMU sensor (IMU and IIM sensors are mutually exclusive, only one should be enabled at a given time).
	1: Enable IMU sensor (IMU and IIM sensors are mutually exclusive, only one should be enabled at a given time).
ENABLE_IMU_IIM4623X_SENSOR	0: Disable IIM sensor (IMU and IIM sensors are mutually exclusive, only one should be enabled at a given time).
	1: Enable IIM sensor (IMU and IIM sensors are mutually exclusive, only one should be enabled at a given time).
ENABLE_MOTOR_CONTROLLER	0: Disable motor controller.
	1: Enable motor controller.
ENABLE_AUDIO_ICS43434	0: Disable MIC.
	1: Enable MIC.
ENABLE_CHIRP_CHx01_SENSOR	0: Disable ultrasonic sensor.
	1: Enable ultrasonic sensor.
ENABLE_KWS	0: Disable keyword spotting functionality.
	1: Enable keyword spotting functionality.

Table 1. Sensor Compile time Macros

Two main functions for application to initialize and use RoboKit-Core are:

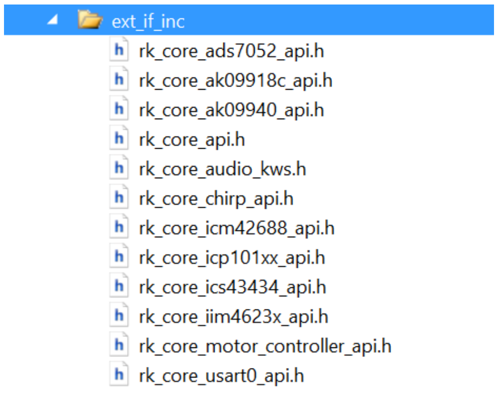
- robokit_core_init: initializes MCU, drivers, middleware, and all the sensors. After initialization the application can initialize the application layer.
- robokit_core_start: creates and start internal FreeRTOS tasks to manage the sensors.

3.2.1 IMU selection

RoboKit1 has an Onboard IMU (ICM-42688) and a mounting space with a connector for the industrial grade IMU module (IIM-46230) which can be mounted on the board. Both the IMUs use the same interface, so, at a given point in time, only one of them can be enabled. Macros to enable or disable an IMU are provided in the file inv_platform.h and are explained in Table 1.

3.3 EXT_IF_INC

This folder has the header files needed for all the sensors. They contain the declarations of the API's to be used by applications. APIs to start and stop and a function to register the data read callback functions are declared here for each sensor. Also, the data type for each sensor can be found in these header files.

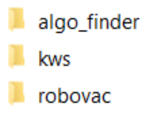


```
ext_if_inc
    rk_core_ads7052_api.h
    rk_core_ak09918c_api.h
    rk_core_ak09940_api.h
    rk_core_api.h
    rk_core_audio_kws.h
    rk_core_chirp_api.h
    rk_core_icm42688_api.h
    rk_core_icp101xx_api.h
    rk_core_ics43434_api.h
    rk_core_iim4623x_api.h
    rk_core_motor_controller_api.h
    rk_core_usart0_api.h
```

Figure 5. exe-if_ic folder

3.4 LIBS

In addition to providing raw sensor data to the applications, RoboKit-Core includes libraries which process the sensors data and perform specific tasks like key word detection using audio data. The libs folder contains libraries and their header files.



```
algo_finder
kws
robovac
```

Figure 6. Libs folder

- **algo_finder:** contains libraries and header files that process ultrasonic sensor data.
  The folder has following libraries:
  - *Range finder algorithm*: calculates distance/range from an object using ultrasonic Time of Flight sensor data.
  - *Floor type detection algorithm*: detects the floor type using ultrasonic Time of Flight sensor data. Note that the Chirp sensors used for floor type detection should be pointing towards the floor.
  - *Cliff detection algorithm*: detects if there is a cliff or floor using ultrasonic Time of Flight sensor data.
    The declaration of APIs provided by the libraries above are included in header file rk_core_chirp_api.h.

- **Kws**: the kws folder contains a keyword-spotting algorithm library and corresponding header files. The KWS library uses audio data to detect keywords. After the “Hi TDK” keyword is detected, it waits for commands, and when it detects “left,” “right,” “go,” or “stop” commands, it calls the callback function registered by the application. APIs to register callback function to receive KWS events are included in the rk_core_audio_ksw.h file.
- **Robovac**: the robovac folder contains IMU calibration and the heading detection algorithm library and its header files. It processes raw IMU data and generates calibrated data and heading information which are sent to the application along with raw IMU data through IMU data call back function.

3.5 RTOS_TASKS

RoboKit-Core implements a FreeRTOS task for each of the hardware peripherals on the board. These tasks help manage the data handling of sensors connected to the peripherals. Refer to Figure 2 for more details.

- I^2C task: initializes and manages data from sensors connected to I^2C interface. The sensors connected to I^2C are pressure sensor, magnetometer, and UToF chirp sensors.
- SPI task: initializes and manages data from sensors connected to SPI0; The sensors connected to SPI0 are temperature sensor, industrial grade IMU, and IMU.
- Microphone task: initializes and manages data from microphones connected over two I^2S channels.
- Motor controller task: initializes and manages motor controller commands and responses over UART1.
- USART0 task: handles the exchange of protocol messages between RoboKit1 board and the host computer. The host computer could be either Single Board Compute (SBC) like Raspberry PI or a personal computer.
- UART2 task: outputs debug text messages over USB2.
- KWS task: handles the keyword spotting functionality. It takes microphone data as input.
- Monitor task: outputs sensor data and system status over UART2.

3.6 SENSORS

The initialization, configuration, and data handling of all the sensors is done in this module. Each folder contains the interface functions specific to a peripheral or a specific sensor.

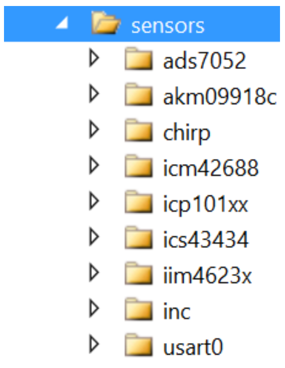


Figure 7. Sensors folder

- **ads7052**: contains temperature sensor driver to initialize, configure, and read data. The temperature sensor is connected to MCU over SPI and the data is available in degrees Celsius.
- **akm09918c**: contains Magnetometer driver to initialize, configure, and read data. The magnetometer sensor is connected to MCU over I^2C. The data is available in Micro Tesla.
- **chirp**: contains an Ultrasonic Time of Flight (UToF) sensor driver can be configured to read range data or raw measurement data. The raw measurement data is input to the algorithms like range finder and floor type detection. RoboKit1 board provides capability to connect up to nine UToF Chirp 101/201 sensors over three I^2C buses. The range value is in meters.
- **icm42688**: contains an IMU sensor driver. The IMU is connected to MCU over SPI. It provides raw acceleration data in units of g (gravitational acceleration) and angular velocity data in radians per second. The data is used by robovac library to calibrate the IMU and for heading detection.
- **icp101xx**: contains a pressure sensor driver. The pressure sensor is connected to MCU over I^2C. The pressures data is output in millibar (mbar).

- **ics43434**: contains microphone sensor driver. RoboKit1 has 4 microphones connected to MCU through two I^{2}S channels. Robokit-Core merges the audio data from both the channels and provides a single raw audio data buffer. An application like Audacity can be used to play the audio on host. The default configuration of audio data output is Mono audio with 16-bit data and 16 KHz sampling rate.
- **iim4623x**: contains an industrial IMU driver. The external IIM module is connected to the MCU over SPI. Just like ICM, it provides raw acceleration data in units of g (gravitational acceleration) and angular velocity data in radians per second. Note that IIM is an external module connected to RoboKit1, and both ICM and IIM are mutually exclusive.
- **inc**: contains all the driver header files.
- **usart0**: contains interface functions for application to use USART0 to exchange the messages between the RoboKit1 and the Host computer.

3.7 UTILS

The debug message logging utilities and RoboKit Interface protocol code used by Robokit-App-Host-Interface are present in this module. The logging module currently outputs data over UART2. For more details about the RoboKit Interface protocol refer to *AN-000339 RoboKit1 Interface Protocol Document*.

4 ROBOKIT APPS

Robokit-APP is the application layer of the RoboKit firmware. It is agnostic of sensor driver's implementation. It starts, stops, and receives data from sensors through Robokit-Core APIs.

Current RoboKit software package has Robokit-Core and two RoboKit applications (Robokit-App-Host-Interface and Robokit-App-Smart-Bot). Any application can be selected as startup project, compiled, linked with Robokit-Core library, and generate executable file to be loaded on the board. The folders struct of RoboKit applications is shown below

- ***Robokit*** : root folder of RoboKit software
  - ***Robokit-Core***
  - ***Robokit-Apps***
    - ***Apps-Common*** : Common code all apps can use.
    - ***Robokit-App-Host-Iterface*** : Host-Interface app specific files.
    - ***Robokit-App-Smart-Bot*** : Smart-Bot app specific files.

Figure 8 shows the folder structure of Robokit-App-Smart-Bot application

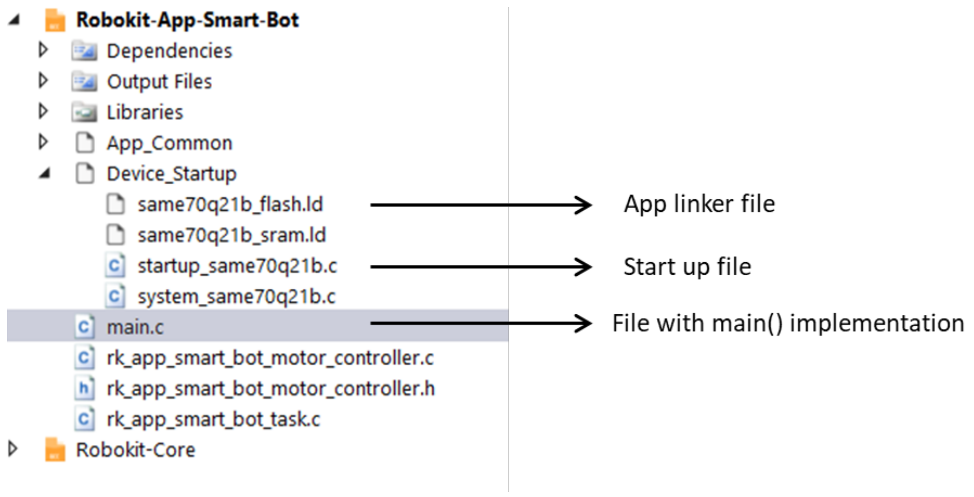


Figure 8. Robokit-App-Smart-Bot application

4.1 INTERFACE WITH SENSORS

When the RoboKit application wants to get data from a sensor, it must include required sensor API header file, implement the sensor data handling function, and register it as a callback function with the RoboKit-Core and start the sensor.

4.2 EXAMPLES:

4.2.1 IMU/ICM Sensor (ICM42688):

Skeleton of IMU sensor data callback function to be fully implemented by the application. Example implementation can be found in rk_app_smart_bot_task.c.

```
#include "rk_core_icm42688_api.h"
void rk_smart_bot_app_icm42688_data_callback(uint64_t timestamp_us,imu_data_t * pdata){
   /* pdata contains IMU data and time stamp in timestamp_us */
}
```

APIs to register and start IMU sensor

```
/* Register IMU/ICM sensor data call back function */
rk_core_imu42688_data_callback_register(rk_smart_bot_app_icm42688_data_callback);

/* Start Sensor */
rk_core_imu42688_start();
```

4.2.2 IIM Sensor (IIM4623x):

Skeleton of IIM sensor data callback function to be fully implemented by the application.

```
#include "rk_core_iim4623x_api.h"
void rk_app_iim_data_callback_func(iim_data_t * pdata){
   /* pdata contains IIM data */
}
```

APIs to register and start IIM sensor

```
/* Register IIM sensor data call back function */
rk_core_iim4623x_data_callback_register(rk_app_iim_data_callback_func);

/* Start Sensor */
rk_core_iim4623x_start() ;
```

4.2.3 Ultrasonic range finder sensor (CH101, CH201)

Skeletons of Chirp (UToF) sensor data and algo callback functions to be fully implemented by the application.

```
#include "rk_core_chirp_api.h"
void rk_smart_bot_app_chirp_iq_data_callback_func(uint64_t timestamp_us,chX01_data_t * pdata){
   /* pdata contains Ultrasonic chirp data and time stamp in timestamp_us */
}

void rk_app_chirp_floor_type_callback_func(uint8_t sensor_id, const uint8_t * algo_out, uint16_t
algo_out_size){
   /* algo_out contains floor type detection alg output */
}

void rk_app_chirp_cliff_detect_callback_func(const uint8_t * algo_out, uint16_t algo_out_size){
   /* algo_out contains cliff detection alg output */
}
```

APIs to register and start Chirp (UToF) sensor-related algorithms

```
/* Register for ultrasonic chirp sensor data and floor type and cliff detect alg
output call back function */
rk_core_chirp_data_callback_register(rk_smart_bot_app_chirp_iq_data_callback_func);
rk_core_chirp_floor_type_callback_register(rk_app_chirp_floor_type_callback_func);
rk_core_chirp_cliff_detect_callback_register(rk_app_chirp_cliff_detect_callback_func);

/* Start Sensor */
rk_core_chirp_start();
```

4.2.4 Audio Sensor (ICS43434):

Skeleton of audio sensor data callback function to be fully implemented by the application.

```
#include "rk_core_ics43434_api.h"
void rk_smart_bot_app_audio_data_callback_func(uint8_t ch_id,uint8_t * pdata){
   /* pdata contains audio data. Process audio data as required by app */
}
```

APIs to register and start audio sensors

```
/* Register for Audio data call back */
rk_core_ics43434_data_callback_register(rk_smart_bot_app_audio_data_callback_func);
/* Start ICS 43434 to capture audio */
rk_core_ics43434_start();
```

4.2.5 KWS (Key Word Spotting) service:

Skeleton of callback function to access KWS service data when the key word is detected. The detailed implementation must be done by the application.

```
#include "rk_core_audio_kws.h"
void rk_smart_bot_app_audio_kws_notify_callback_func(rk_core_audio_kws_cmd kws_cmd) {
   switch(kws_cmd)
   {
      case RK_CORE_AUDIO_KWS_LEFT :
      /* "left" is detected after "Hi TDK" key word */
      break;
      case RK_CORE_AUDIO_KWS_RIGHT :
      /* "right" is detected after "Hi TDK" key word */
      break;
      case RK_CORE_AUDIO_KWS_GO :
      /* "go" is detected after "Hi TDK" key word */
      break;
      case RK_CORE_AUDIO_KWS_STOP :
      /* "stop" is detected after "Hi TDK" key word */
      break;
      default :
      break;
   }
}
```

APIs to register the KWS service. Since audio data is input to KWS service, enabling the audio sensor is required for the KWS service to work.

```
rk_core_audio_kws_notify_callback_register(rk_smart_bot_app_audio_kws_notify_callback_func);

/* Start audio sensor to capture audio */
rk_core_ics43434_start();
```

4.2.6 Pressure sensor (ICP101xx):

Skeleton of Pressure sensor data callback function to be fully implemented by the application.

```
#include "rk_core_icp101xx_api.h"
void rk_app_press_data_callback_func(uint64_t timestamp_us,icp10101_data_t * pdata) {
   /* pdata has pressure data */
}
```

APIs to register and start pressure sensor

```
/* Register for pressure sensor data call back function */
rk_core_icp101xx_data_callback_register(rk_app_press_data_callback_func);

/* Start Sensor */
rk_core_icp101xx_start();
```

4.2.7 Magnetometer sensor (AK09918C)

Skeleton of magnetometer data callback function to be implemented by the application.

```
#include "rk_core_ak09918c_api.h"
void rk_app_akm_mag_data_callback_func(uint64_t timestamp_us,ak09918c_data_t * pdata) {
   /* pdata has Mag data */
}
```

APIs to register and start magnetometer

```
/* Register for pressure sensor data call back function */
rk_core_ak09918c_data_callback_register(rk_app_akm_mag_data_callback_func) ;

/* Start Sensor */
rk_core_ak09918c_start();
```

4.2.8 Thermistor sensor (B57861s0103a039)

Skeleton of thermistor data callback function to be fully implemented by the application.

```
#include "rk_core_ads7052_api.h"
void rk_app_temp_data_callback_func(uint64_t timestamp_us,float * pdata) {
   /* pdata has temp data */
}
```

APIs to register and start thermistor

```
/* Register for pressure sensor data call back function */
rk_core_ads7052_data_callback_register(rk_app_temp_data_callback_func);

/* Start Sensor */
rk_core_ads7052_start();
```

4.2.9 Motor Controller:

Motor controller needs to receive heading data continuously to turn the robot at the requested angle. Robovac algorithm library makes use of the ICM sensor data and provides the heading information to the motor controller. Register motor controller heading input function as the ICM callback function

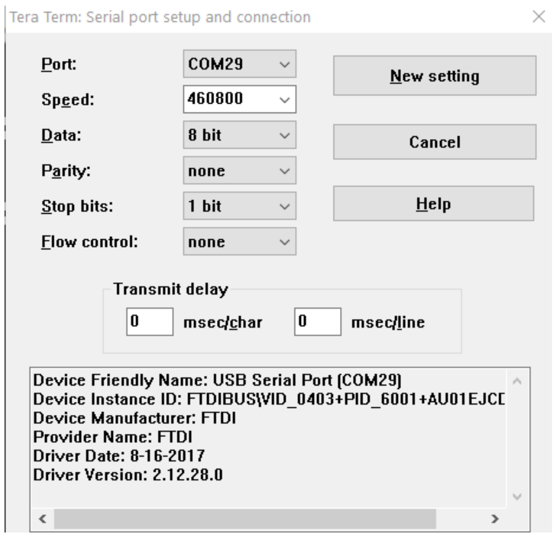
```
/* Registering Motor controller heading input function for ICM heading call back function */
   rk_core_imu42688_heading_data_callback_register(rk_core_mc_input_heading);
```

Call `rk_core_mc_turn()` to turn and `rk_core_mc_set_speed()` to set RoboKit1 to proceed straight.

4.3 APP TO HOST INTERFACE.

4.3.1 Logging

Robokit core implements week function `robokit_debug_data_callback_fun()` to output debug text to UART2. Robokit app's can implement function to alter this behavior. UART2 data is routed to the host computer over USB2. Any serial port utility like TeraTerm can be used to view RoboKit logs. Figure 9 shows host side serial port settings to view logs.



Port: COM29
Speed: 460800
Data: 8 bit
Parity: none
Stop bits: 1 bit
Flow control: none
Transmit delay: 0 msec/char 0 msec/line

```
Device Friendly Name: USB Serial Port (COM29)
Device Instance ID: FTDIBUS\VID_0403+PID_6001+AU01EJCD
Device Manufacturer: FTDI
Provider Name: FTDI
Driver Date: 8-16-2017
Driver Version: 2.12.28.0
```

Figure 9. Data Logging

4.3.2 Sensor data

RoboKit1 uses “RoboKit Interface Protocol” to exchange the messages between the RoboKit1 and the host. Protocol data messages are exchanged between the RoboKit1 and the host computer over the USART0 at the baud rate of 1000,000. USART0 is routed to either USB1 or 40 pin J30 header based on SW16 switch settings (Refer to the hardware reference guide for more details). If the TDK host application (Windows App or the Android App) is not used, the end users need to implement the message handling and RoboKit control mechanism on their own. Refer to the RoboKit Interface protocol document for more details.

Note: By setting `RK_TO_HOST_PROTOCOL_INTERFACE` in inv_platform.h file to 0, sensor data can be routed to UART2 instead of debug logs.

5 COMPILING AND LOADING ROBOKIT

RoboKit software runs on an on-board Atmel MCU (ATSAME70Q21B), a high-performance 32-bit ARM Cortex-M7 processor with Floating Point Unit (FPU). It uses Atmel's Microchip Studio (https://www.microchip.com/en-us/tools-resources/develop/microchip-studio) IDE to compile and load firmware binaries.

5.1 HARDWARE SET-UP

To load and debug software on the RoboKit1 board, connect the micro-USB to USB1 for power supply and micro-USB to USB3, which is an EDBG port. Before loading the firmware ensure that all SW5 switches are turned ON as shown in Figure 10.

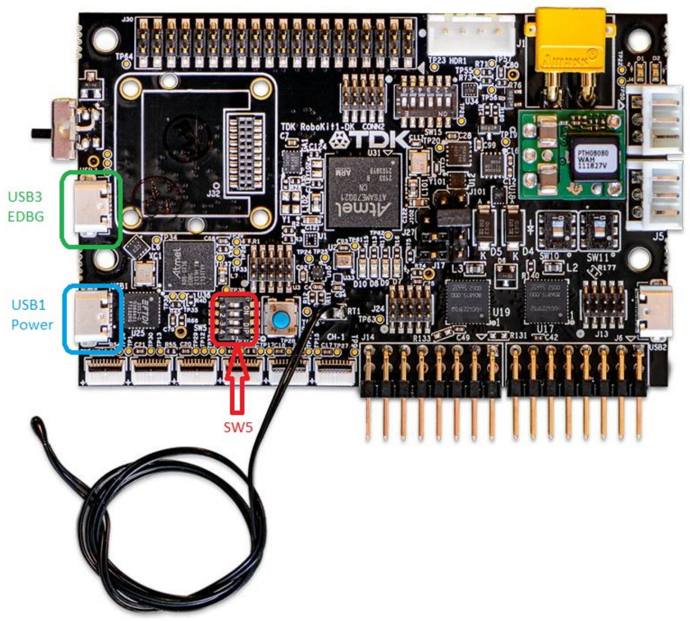


Figure 10. USB connections to load firmware

5.2 ROBOKIT ATMEL SOLUTION PROJECT

The main solution file, "Robokit.atsln," for firmware is located at the root folder of software package and can be opened with Microchip Studio. It includes Robokit-Core and two application project files as shown in Figure 11. To compile the needed application project, set the application project file as the startup project by right-clicking on it and selecting "Set as Startup Project." Then compile it.

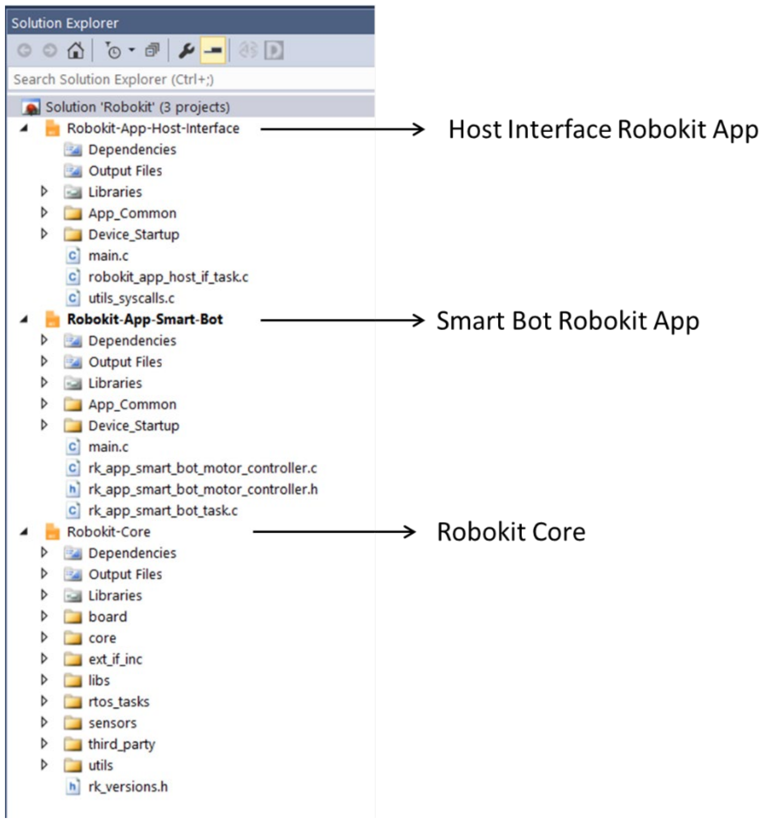


Figure 11. Atmel Project files

To load the image, select “Properties” under Robokit-App-Smart-Bot as shown in Figure 12.

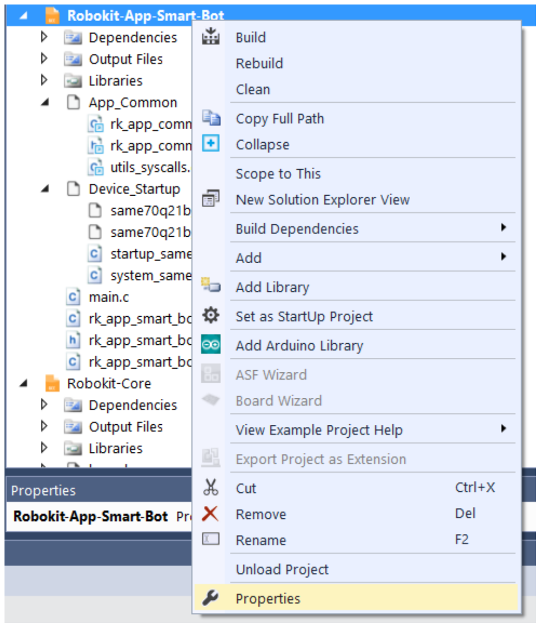
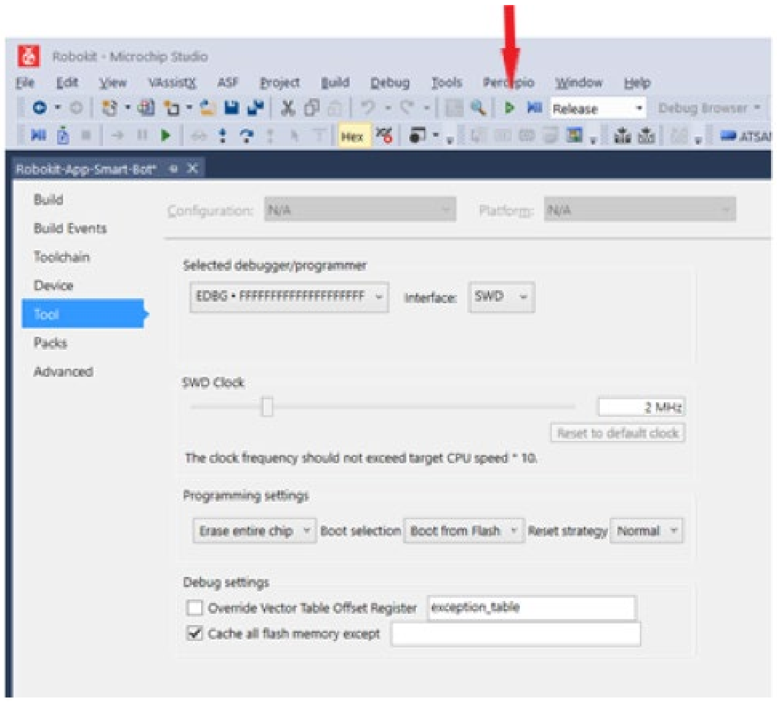


Figure 12. Selecting Properties

Under tools set EDBG as debugger/programmer as shown in Figure 13 and click green play icon pointed by arrow in Figure 13.



Configuration: N/A
Platform: N/A

Selected debugger/programmer: EDBG • FFFFFFFFFFFFFFFFFFFF Interface: SWD

SWD Clock: 2 MHz

The clock frequency should not exceed target CPU speed * 10.

Programming settings: Erase entire chip Boot selection: Boot from Flash Reset strategy: Normal

Debug settings
- [ ] Override Vector Table Offset Register: exception_table
- [✓] Cache all flash memory except

Figure 13. Select Start Debugging

6 REFERENCES

https://invensense.tdk.com/robokit/

https://invensense.tdk.com/products/motion-tracking/6-axis/icm-42688-p/

https://invensense.tdk.com/products/smartindustrial/iim-46230/

https://invensense.tdk.com/products/1-axis/icp-101xx/icp-10111/

https://invensense.tdk.com/products/ics-43434/

https://invensense.tdk.com/products/ch101/

https://invensense.tdk.com/products/ch201/

https://product.tdk.com/en/search/sensor/ntc/ntc_element/info?part_no=B57861S0103A039&gclid=EAIaIQobChMInKLFncbJ9QIVRAnnCh1qdAAsEAAYASAAEgKPzPD_BwE/

https://www.akm.com/global/en/products/electronic-compass/lineup-electronic-compass/ak09918c/

https://www.micronas.tdk.com/en/products/embedded-motor-controllers/hvc-4223f4420f

7 REVISION HISTORY

REVISION DATE	REVISION	DESCRIPTION
01/24/2022	1.0	Initial Release
09/21/2022	1.1	Updated Section 4.3.2