

SMARTSONIC CHX01 HELLO CHIRP EXAMPLE USER GUIDE

1 INTRODUCTION

This example demonstrates how to build and run a basic ultrasonic sensing application using the Chirp SonicLib API for CH101 and CH201 sensors.

Hello Chirp is a simple C application that demonstrates how the SonicLib functions are used to initialize, configure, and operate one or more ultrasonic sensors. It is intended to be a developer's first exposure to ultrasonic sensing and the SonicLib API functions.

The application discovers what sensors are connected to the board, programs and configures them, and then triggers and displays range (distance) measurements through a serial connection.

In this example, the application is built using Microchip MPLAB X and runs on the SmartSonic version 1 evaluation board, which features an Atmel SAMG55 microcontroller. The SmartSonic v1 board uses sensor daughter boards that support up to four ultrasonic sensors (one mounted to the board, and others connected using flex cables). The Hello Chirp application can detect and run with either a single sensor or multiple sensors connected to the board.

The source code for the application can be found in **source/application/chx01-hellochirp-example**. The **main.c** file is the main file in the application. It contains extensive comments explaining how the SonicLib interfaces are used.

1.1 REQUIRED EQUIPMENT

- SmartSonic version 1 evaluation board
- Chirp sensor daughter board for CH101 or CH201. Optionally, one or more off-board CH101 or CH201 sensors may be used with a flex cable.
- One Micro-USB cable

1.2 REQUIRED SOFTWARE PACKAGES

- **invn.chirpmicro.smartsonic.chx01-hellochirp-example.X.X.X.zip** (actual file name will include version number), which includes the following:
 - Hello Chirp application source files
 - Chirp SonicLib sensor API and driver files. The Hello Chirp example application requires SonicLib v3.21.0 or later.
 - Sensor firmware image files
 - Board support package files for Chirp SmartSonic v1 board
 - MPLAB X project files to build the application
- [MPLAB X](#) integrated development environment – download from Microchip.com
- Terminal emulator of your choice (for example PuTTY or TeraTerm)

TABLE OF CONTENTS

1	Introduction	2
1.1	Required Equipment	2
1.2	Required Software Packages	2
2	Installation/Preparation	4
3	Building the Hello Chirp Application	6
4	Programming the SmartSonic v1 Board	7
5	Running the Hello Chirp Application	9
6	Sensor Configuration Settings	11
6.1	Selecting the Sensor Firmware	11
6.1.1	CH101_GPR	11
6.1.2	CH101_GPRMT	12
6.1.3	CH201_GPRMT	12
6.2	CHIRP_FIRST_SENSOR_MODE	13
6.3	CHIRP_OTHER_SENSOR_MODE	13
6.4	CHIRP_SENSOR_MAX_RANGE_MM	13
6.5	CHIRP_STATIC_REJECT_SAMPLES	13
6.6	CHIRP_RX_PRETRIGGER_ENABLE	13
6.7	Setting Target Detection Thresholds	13
7	Application Configuration Settings	15
7.1	MEASUREMENT_INTERVAL_MS	15
7.2	APP_DATA_MAX_SAMPLES	15
7.3	Enabling Full Sample Data Output	15
7.3.1	OUTPUT_AMP_DATA_CSV	15
7.3.2	OUTPUT_IQ_DATA	15
7.4	READ_DATA_NONBLOCKING	15
7.5	DISPLAY_AMP_VALUE	16
7.6	DISPLAY_SAMPLE_NUM	16
8	Revision History	17

2 INSTALLATION/PREPARATION

1. Download and install the MPLAB X IDE.
2. Download and install (unzip) the SmartSonic CHx01 Hello Chirp application to a project directory of your choice.
3. Install terminal emulator.
4. Connect the Chirp sensor daughterboard to the SmartSonic v1 board. Be careful to align the white arrows on the boards, as shown in Figure 1.
5. *Optional:* Using flat flex cables, attach additional off-board CHx01 sensor(s) to the connectors on the daughterboard. If an offboard sensor is connected to the Sensor 0 connector (J6), you must set the switch on the side of the daughterboard to use the off-board sensor as Sensor 0 instead of the sensor on the daughterboard.
6. Connect the SmartSonic v1 board to a Windows PC with the USB cable. Configure the jumpers on the board as shown in Figure 1 with J1 in EDBG USB mode (short pins 3-4).
7. The second USB connector (UART/USB for Data) is not used in this application.

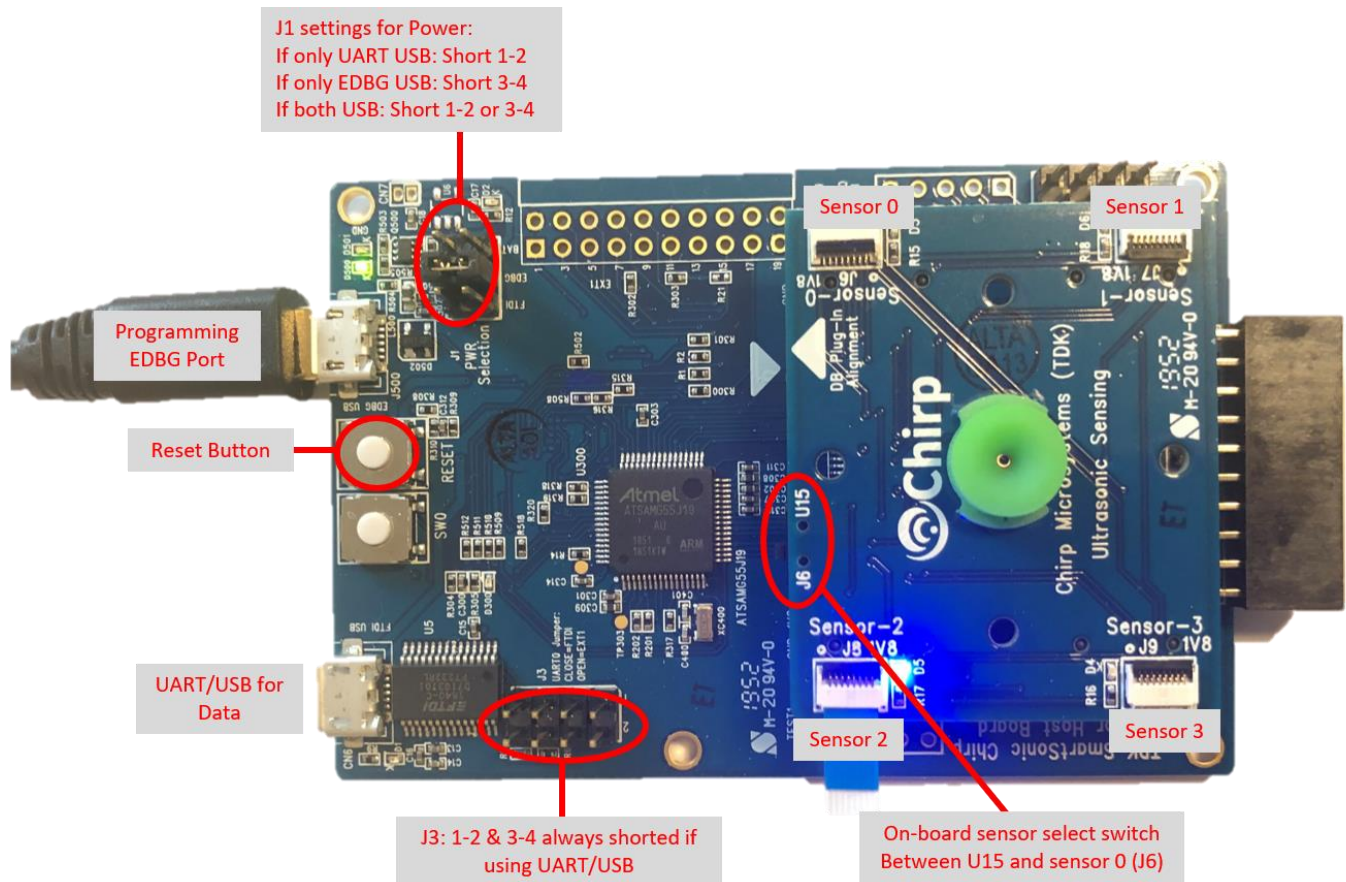


Figure 1 - SmartSonic v1 with CH201 Daughterboard

8. Open Windows Device Manager, open the **Ports (COM & LPT)** list, and identify the COM port number(s) assigned to the SmartSonic board. There are two possible ports associated with the SmartSonic board: "EDBG Virtual COM Port" and "USB Serial Port".

- The **EDBG Virtual COM Port** is the connection that will be used in this application. It is used to connect to the on-board debugger for programming the board. It also is used for regular serial I/O to or from an application running on the board (like Hello Chirp). You will need to specify this port number when opening the terminal emulator to display output from the program when it runs.
- A **USB Serial Port** entry will appear if a second USB cable is connected to the FTDI / UART Serial Port. This port is not used in this application.

3 BUILDING THE HELLO CHIRP APPLICATION

1. Open MPLAB X.
2. Open the Hello Chirp project:
 - a) Open **File** menu
 - b) Select **Open Project...**
 - c) In your Hello Chirp installation, navigate to the **project/mplabx** directory and select **smartsonic-chx01-hellochirp-example**.
 - d) Click **Open Project**. The program should locate the project files and display the contents of the project.
3. The Hello Chirp project source files are organized in three sub-directories under **source**:
 - **source/application/chx01-hellochirp-example** – contains **src** and **inc** directories with the Hello Chirp application for CH101 and CH201.
 - The **src/main.c** file contains the entry point for the application along with various routines that demonstrate how to read and manage the Chirp sensor(s). See the comments in that file for detailed information about the operation of the application.
 - The **inc/app_config.h** file specifies various configuration parameters affecting the sensor operation and the application behavior. These parameters and options are described later in this document.
 - **source/board/smartsonic** – contains support files for the Chirp SmartSonic v1 board.
 - The main board support package routines, as defined in the **chirp_bsp.h** header file, are implemented in the **HAL/src/chbsp_chirp_samg55.c** file.
 - The **config/chirp_board_config.h** file is required by SonicLib. This file contains definitions for the number of possible devices and I²C buses on the board and is used for static allocation of arrays.
 - **source/drivers/invn-soniclib/invn/soniclib** – contains the SonicLib API and driver files, sensor firmware modules, and other distribution files.
 - The directory contains header files that must be included when building applications with SonicLib. In particular, the **soniclib.h** file contains the key definitions for the SonicLib API.
 - The logical directory structure shown in the project includes files from sensor firmware plugin modules combined with the standard SonicLib distribution files. The on-disk storage organization will be different.
 - The **source/drivers/invn-soniclib/html** directory contains HTML documentation for the SonicLib and BSP interfaces. Open the **index.html** file to get started.
4. Build the project:
 - Select **Production > Build Project**

The project should build without errors. The default build configuration is “Release” so the build output files will be placed in the **smartsonic-chx01-hellochirp-example/dist/Release** sub-directory in the project.

4 PROGRAMMING THE SMARTSONIC V1 BOARD

1. Set up the SmartSonic v1 board jumper J1 as shown in Figure 1.
2. Connect the SmartSonic v1 board EDBG port to a Windows PC with a micro-USB cable.
3. MPLAB X should detect the debug interface on the SmartSonic v1 board and recognize it as an Atmel Embedded Debugger. (If not, see the section below on selecting the “Exclude device checks for kits” option.)
4. In MPLAB X, click the **Make and Program Device** button in the toolbar:

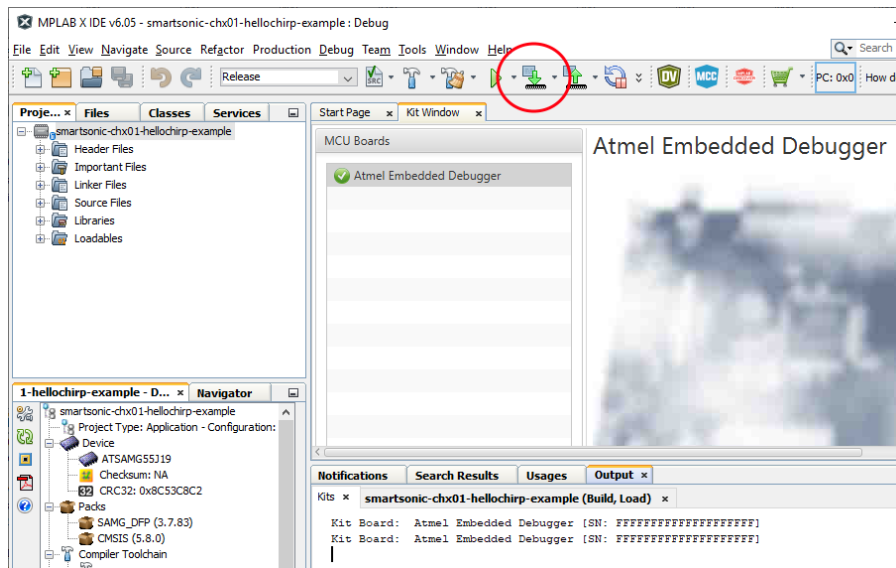


Figure 2 - Make & Program Device Option

MPLAB X will rebuild the example application and then program the board with the resulting binary image.

The first time you program the board using a new project, you may need to select the debug tool interface. In the **Tool not found** popup window, select **Atmel Embedded Debugger-SN: FFFFFFFFFFFFFFFF** in the pull-down list as shown below. Click **OK**.

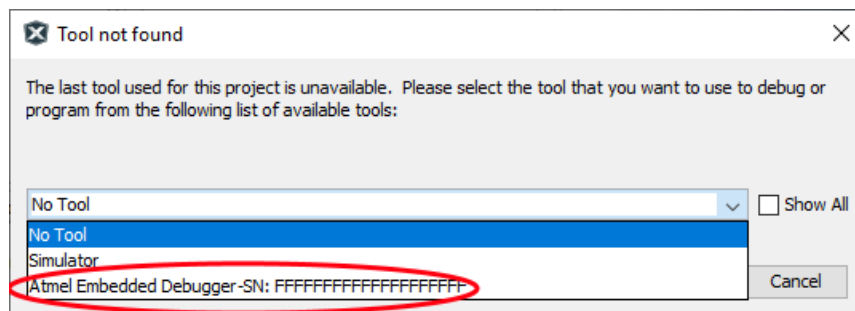


Figure 3 - Selecting Debug Tool

If the Atmel Embedded Debugger entry is greyed out and cannot be selected, open the **Tools > Options** window and select **Embedded**. Scroll down to “**Exclude device checks for kits**” and check the box. Click **OK**.

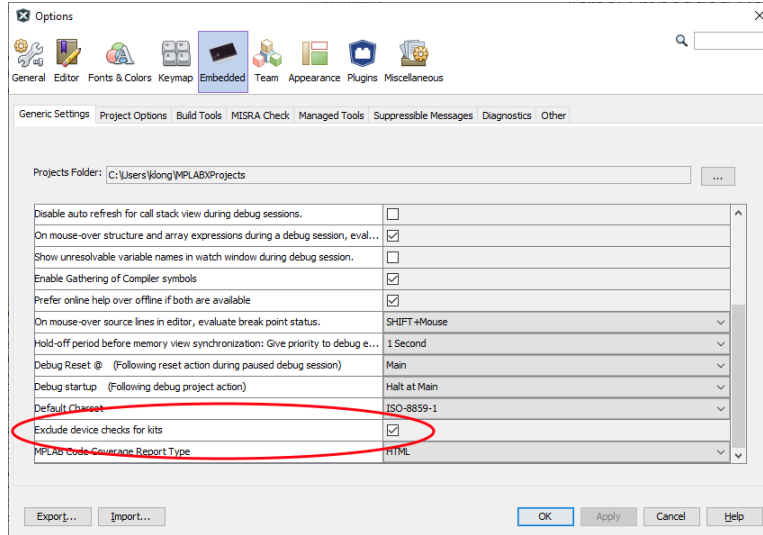


Figure 4 - Excluding Checks for Kits

5. Your SmartSonic v1 board is successfully programmed when the **Output** console reads **Programming complete**:

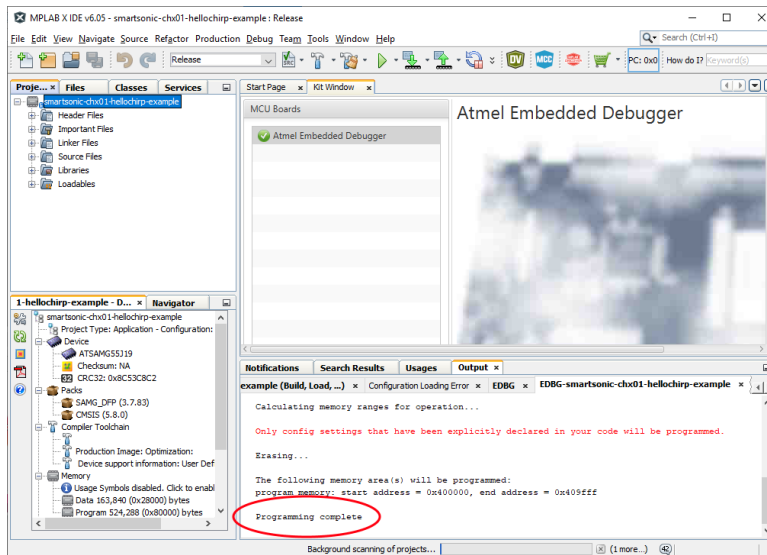


Figure 5 - Successful Programming

5 RUNNING THE HELLO CHIRP APPLICATION

1. Start the terminal emulator program and open/configure the COM port assigned to the SmartSonic board “EDBG” interface with the following port settings:
 - **1000000 baud**
 - **8 bits data, no parity, 1 stop bit**
 - **New-line sequence = Line Feed only (no carriage return)**
 - PuTTY: Open **Terminal** configuration. Select “**Implicit CR in every LF**”.
 - TeraTerm: Open **Setup > Terminal**. Under “**New-line**” set “**Receive**” to “**LF**”.
2. Reset the SmartSonic v1 board using the board’s reset button (next to the Programming EDBG connector).
3. Status messages from the application will appear on the terminal output, followed by summary data from the sensor initialization (device frequency, etc.) and configuration (maximum range, etc.).
4. Range (distance) data from the sensor device will then be output in a continuous loop.
5. Along with the range indication, there is an optional amplitude value of the received ultrasound for the target being reported. Higher amplitude values indicate stronger reflections from the target but are not in any standard units.

Figure 5, below, shows the typical application output for the default configuration (CH101 sensor with CH101_GPR firmware). Output for the multi-threshold firmware types is similar, with an additional display of the threshold settings after configuration.

```

COM6 - Tera Term VT
File Edit Setup Control Window Help
Chirp sensor 0 found
Chirp sensor 1 not found
Chirp sensor 2 not found
Chirp sensor 3 not found
Chirp sensor Idd: 102 uA

Hello Chirp! - SonicLib Example Application for CHx01 Sensors
  Compile time: Mar 23 2023 15:41:02
  Version: 2.26.0   SonicLib version: 3.21.2

Initializing sensor(s)... starting group... OK

Sensor Type      Freq      B/W      RTC Cal      Firmware
0      CH101      178263 Hz  4072 Hz  2837@100ms  gpr_gpr_gpr-101_v43a

Configuring sensor(s)...
Sensor 0:      max_range=754mm (98 samples)   mode=TRIGGERED_TX_RX
Initializing sample timer for 100ms interval... OK
Starting measurements
Dev 0: Range: 22.9 mm (sample 3) amp=14429
Dev 0: Range: 379.4 mm (sample 50) amp= 1001
Dev 0: Range: 379.1 mm (sample 50) amp=  934
Dev 0: Range: 378.4 mm (sample 50) amp=  924
Dev 0: Range: 379.5 mm (sample 50) amp=  930
Dev 0: Range: 379.5 mm (sample 50) amp= 1006
Dev 0: Range: 379.2 mm (sample 50) amp= 1032
Dev 0: Range: 378.8 mm (sample 50) amp= 1071
Dev 0: Range: 379.3 mm (sample 50) amp= 1092
Dev 0: Range: 379.1 mm (sample 50) amp= 1096
Dev 0: Range: 379.3 mm (sample 50) amp= 1083
Dev 0: Range: 378.4 mm (sample 50) amp= 1052
Dev 0: Range: 378.9 mm (sample 50) amp=  991
Dev 0: Range: 379.6 mm (sample 50) amp=  958
  
```

Figure 6 - Typical Output (CH101_GPR)

6 SENSOR CONFIGURATION SETTINGS

The **app_config.h** header file contains symbolic definitions for parameters that affect the sensor measurements and application execution. You can change these definitions to adjust the overall operation.

Note: You should get the Hello Chirp application running on your board using the default configuration **BEFORE** experimenting with configuration settings!

The values defined in **app_config.h** whose names begin with “CHIRP_” are used as parameters to SonicLib API function calls in **main.c**. In particular, the *configure_sensors()* routine in that file uses many of these values to apply different configuration settings and options during the initialization sequence.

Please refer to the usage and comments in **main.c** to see how the SonicLib API is called based on these symbols. See *AN-000175 SonicLib Programmer’s Guide* and the included SonicLib HTML documentation for more information.

6.1 SELECTING THE SENSOR FIRMWARE

CH101 and CH201 sensors are fully programmable, and during initialization they must be loaded with a specific firmware image to operate. Different sensor firmware types can provide different features or performance characteristics.

The Hello Chirp example application includes several standard Chirp “GPR” (General Purpose Ranging) sensor firmware types for CH101 and CH201. These firmware versions provide popular sensing features and offer good performance over various distances under most conditions.

CH101 and CH201 sensors are identical electrically, so a firmware mismatch cannot be detected. You must make sure you use firmware for the correct sensor model. Note that the default selection in this example is for CH101_GPR, so if you are using a CH201 sensor, you must change the firmware type.

The firmware type (and therefore the sensor model, CH101 or CH201) is selected in the **app_config.h** file by the **CHIRP_SENSOR_FW_INIT_FUNC** symbol. This definition specifies which firmware initialization routine will be passed to the *ch_init()* function when each sensor is initialized in the Hello Chirp *main()* function, so that the selected firmware image will be programmed into the device.

In this example project, three basic types of sensor firmware are used, two for CH101 devices (CH101_GPR and CH101_GPRMT) and one for CH201 devices (CH201_GPRMT). The “Sensor Firmware Selection” section of the **app_config.h** header file contains a definition of **CHIRP_SENSOR_FW_INIT_FUNC** for each of these variants. You should uncomment the one line that corresponds to the firmware you want to use, as explained below.

It is also possible to use the Hello Chirp example application with other sensor firmware releases from TDK InvenSense. Define the **CHIRP_SENSOR_FW_INIT_FUNC** symbol to equal the name of the new firmware’s initialization function, following the same pattern as the GPR firmware types. For more information on adding new sensor firmware types to an existing installation, see the *AN-000175 SonicLib Programmer’s Guide* document.

6.1.1 CH101_GPR

The CH101_GPR General Purpose Ranging firmware provides a basic set of sensing and ranging features for CH101 sensors. Static Target Rejection (STR) is supported, but the detection thresholds cannot be changed.

There are four different CH101_GPR firmware images to choose from, two for normal-range sensing and two for short-range sensing. Within each range type, there are versions for wide (omnidirectional) and narrow (directional) acoustic horns, which differ in some internal gains and settings.

- The standard **CH101_GPR** firmware can be used for sensing up to one meter away.
 - For wide (omnidirectional horns), use the basic **ch101_gpr** firmware, by uncommenting the line:


```
#define CHIRP_SENSOR_FW_INIT_FUNC ch101_gpr_init
```

ch101_gpr is the default firmware type selected in the **app_config.h** file.
 - For narrow (directional horns), use the **ch101_gpr_narrow** firmware, by uncommenting the line:


```
#define CHIRP_SENSOR_FW_INIT_FUNC ch101_gpr_narrow_init
```
- The **CH101_GPR_SR** firmware is optimized for short-range performance. This special firmware provides more measurement resolution and operates at closer distances. The resolution of the sensor is increased by a factor of four, however the maximum operating range for the sensor is correspondingly reduced by a factor of 4.
 - For wide (omnidirectional horns), use the **ch101_gpr_sr** firmware, by uncommenting the line:


```
#define CHIRP_SENSOR_FW_INIT_FUNC ch101_gpr_sr_init
```

Note that **ch101_gpr** is the default firmware type selected in the **app_config.h** file, so this line will initially be uncommented.
 - For narrow (directional horns), use the **ch101_gpr_sr_narrow** firmware, by uncommenting the line:


```
#define CHIRP_SENSOR_FW_INIT_FUNC ch101_gpr_sr_narrow_init
```

When using the special short-range firmware, you should consider changing the maximum range setting for the sensor (**CHIRP_SENSOR_MAX_RANGE_MM**, described below) from the default 750 mm to 250 mm or less. If left unchanged, the sensor will use the maximum possible range for this firmware.

6.1.2 CH101_GPRMT

The **CH101_GPRMT** firmware for CH101 sensors provides a set of six user-programmable detection thresholds, to allow an application to tune the rangefinding performance to the physical environment. (In contrast, the detection thresholds in the regular **CH101_GPR** firmware are fixed.) However, the **CH101_GPRMT** firmware does not support Static Target Rejection (STR), due to code space constraints in the sensor.

To use the **ch101_gprmt** firmware, uncomment the line:

```
#define CHIRP_SENSOR_FW_INIT_FUNC ch101_gprmt_init
```

To set the detection thresholds, see **Setting Target Detection Thresholds**, below.

6.1.3 CH201_GPRMT

The **CH201_GPRMT** firmware for long-range CH201 sensors provides a set of six user-programmable detection thresholds, to allow an application to tune the rangefinding performance to the physical environment.

The **CH201_GPRMT** firmware does not support Static Target Rejection (STR), due to code space constraints in the sensor. For STR support on CH201 devices, please see the separate **SmartSonic STR Example** project, which uses different sensor firmware.

To use the **ch201_gprmt** firmware, uncomment the line:

```
#define CHIRP_SENSOR_FW_INIT_FUNC ch201_gprmt_init
```

To set the detection thresholds, see **Setting Target Detection Thresholds**, below.

6.2 CHIRP_FIRST_SENSOR_MODE

Specifies the operating mode for the first sensor (lowest numbered) that is present. If only one sensor is attached, this value must be either CH_MODE_TRIGGERED_TX_RX or CH_MODE_FREERUN.

This value is used as a parameter to *ch_set_mode()* for the first detected sensor.

6.3 CHIRP_OTHER_SENSOR_MODE

Specifies the mode for all other sensors (i.e. except the first) that are present.

For typical Pitch-Catch operation using two or more sensors, set CHIRP_FIRST_SENSOR_MODE to CH_MODE_TRIGGERED_TX_RX and set CHIRP_OTHER_SENSOR_MODE to CH_MODE_TRIGGERED_RX_ONLY.

This value is used as a parameter to *ch_set_mode()* for the second and subsequent sensors.

6.4 CHIRP_SENSOR_MAX_RANGE_MM

Specifies the maximum detection range, i.e. how long the sensor "listens" for an ultrasound signal. The maximum range value is set in millimeters.

The value set here provides a convenient way to specify the length of a measurement in real-world units. It will be used to set the number of samples that will be taken during each measurement. Note that the maximum possible range may vary depending on sensor model and sensor firmware type. If the value specified here is greater than the maximum possible range, the maximum possible range will be used.

Using a longer maximum range setting will increase the time required for each measurement (and therefore increase the power consumption) and will generate more raw sample data, if used. So, you should select a range value that is sufficient for your actual sensing needs but is not excessively long.

6.5 CHIRP_STATIC_REJECT_SAMPLES

Specifies if static target rejection (STR) will be used. In this example, **STR is only available when using the CH101_GPR firmware** variants. STR is not supported in CH101_GPRMT or CH201_GPRMT.

If CHIRP_STATIC_REJECT_SAMPLES is non-zero, STR will be enabled and will apply to the specified number of samples at the beginning of a measurement. The sensor will only report targets that are non-stationary. Targets whose reflections do not change between measurements will be ignored.

To enable STR filtering for the entire measurement, set this value to CH101_GPR_MAX_SAMPLES.

6.6 CHIRP_RX_PRETRIGGER_ENABLE

Enables receive sensor pre-triggering.

This value specifies if receive-only sensor pre-triggering will be used. This setting only applies if more than one sensor is used and at least one sensor is operating in CH_MODE_TRIGGERED_RX_ONLY.

Receive pre-triggering improves performance in pitch-catch operation at short distances, by triggering the receive-only sensor(s) slightly before the transmitting sensor. However, this setting will reduce maximum range of Rx-only sensors approximately 200mm, relative to the CHIRP_MAX_RANGE_MM setting, above.

Set RX_PRETRIGGER_ENABLE to non-zero to enable receive pre-triggering, or zero to disable.

6.7 SETTING TARGET DETECTION THRESHOLDS

These definitions set the sensor's target detection thresholds. These thresholds specify how large a signal must be received, and at what point in the measurement, for the sensor to indicate that a target was detected and calculate range, etc.

Each threshold consists of a starting sample number within the measurement and the corresponding amplitude level that must be reached. A threshold extends until the starting sample number of the next threshold, if any. At each sample point, the observed amplitude is compared against the threshold level for that sample number.

Lower threshold levels will increase sensitivity for detecting objects, but they can also increase the rate of “false positives.” So, some experimentation is usually required to optimize the thresholds for a particular use.

Not all thresholds must be used, if your sensing environment and application do not require 6 different threshold values. Unused threshold entries may be set to zero, and the last threshold will be used for the remainder of the measurement.

These values are used to initialize the *chirp_detect_thresholds* (**ch_thresholds_t**) structure defined in **main.c**.

CHIRP_THRESH_0_START	Threshold 0 (closest) starting sample.
CHIRP_THRESH_0_LEVEL	Threshold 0 level.
CHIRP_THRESH_1_START	Threshold 1 starting sample.
CHIRP_THRESH_1_LEVEL	Threshold 1 level.
CHIRP_THRESH_2_START	Threshold 2 starting sample.
CHIRP_THRESH_2_LEVEL	Threshold 2 level.
CHIRP_THRESH_3_START	Threshold 3 starting sample.
CHIRP_THRESH_3_LEVEL	Threshold 3 level.
CHIRP_THRESH_4_START	Threshold 4 starting sample.
CHIRP_THRESH_4_LEVEL	Threshold 4 level.
CHIRP_THRESH_5_START	Threshold 5 starting sample.
CHIRP_THRESH_5_LEVEL	Threshold 5 level.

7 APPLICATION CONFIGURATION SETTINGS

The **app_config.h** header file also contains several definitions to control the behavior of the application itself. These include the measurement timing, data storage, and the optional reading and output of sample data.

7.1 MEASUREMENT_INTERVAL_MS

Defines how often the application will get a new sample from the sensor(s). This symbol defines the sensor measurement interval, in milliseconds.

For sensors in triggered mode (CH_MODE_TRIGGERED_TX_RX or CH_MODE_TRIGGERED_RX_ONLY), the application will use a periodic timer to trigger a sensor measurement each time this period elapses. This value is used as a parameter to the *chbsp_periodic_timer_init()* function in the board support package.

For sensors in free-running mode (CH_MODE_FREERUN), the application will set this period as the sensor's internal sample interval.

7.2 APP_DATA_MAX_SAMPLES

Defines how many samples per measurement are expected by this application.

This value is used to allocate storage in the application-specific **chirp_data_t** structure defined in **main.c**. That structure contains an array for individual I/Q data values from the samples within an ultrasound measurement.

7.3 ENABLING FULL SAMPLE DATA OUTPUT

The following build options control if and how the full set of values for all internal samples within an ultrasound measurement will be read and displayed. This data is separate from the standard range and simple target amplitude values that are normally output.

Reading the full sample data set is not required for most basic sensing applications - the reported range value, possibly combined with the simple target amplitude value, is typically all that is required. However, the full set of sample values may be read and analyzed for more advanced sensing or data capture needs.

The measurement data can be read and displayed either as net amplitude values or as individual I and Q components.

When either output option is enabled, after each measurement the I/Q data is read from the sensor into a buffer in the **chirp_data_t** structure defined in **main.c**.

7.3.1 OUTPUT_AMP_DATA_CSV

Enables sample amplitude data output. Set OUTPUT_AMP_DATA_CSV to 1 (non-zero) to enable output of the amplitude values for all samples via the serial port, as a series of comma separated values all on one line. This allows easy cut/paste into a spreadsheet program to generate an "A-Scan" chart. This kind of summary graph can be very helpful to understand what the sensor is actually observing.

When this option is enabled, individual I/Q pair values are converted to net amplitude values as they are output.

7.3.2 OUTPUT_IQ_DATA

Enables sample I/Q data output. Set OUTPUT_IQ_DATA to 1 (non-zero) to enable output of the I/Q pair values. Each I/Q sample appears as a comma separated pair (Q, I) on its own line. Having the separate I and Q values allows more complex analysis of signal phase, etc.

7.4 READ_DATA_NONBLOCKING

Selects blocking vs. non-blocking I/O when reading sample data. This setting only applies if OUTPUT_AMP_DATA_CSV or OUTPUT_IQ_DATA has been selected to enable sample data output.

By default, this application will read sample data in blocking mode (i.e. `READ_DATA_NONBLOCKING` is zero by default). The amplitude or I/Q data will be read from the device and placed in the corresponding data array field in the application's `chirp_data_t` structure. Because the data is read in blocking mode, the calls to `ch_get_iq_data()` will not return until the data has actually been copied from the device.

However, if `READ_IQ_NONBLOCKING` is non-zero, the I/Q data will be read in non-blocking mode. The `ch_get_iq_data()` calls will return immediately, and a separate callback function will be called later to notify the application when the read operation completes.

This value is used for conditional compilation of sections supporting non-blocking I/O. This build-time selection is used so that the board support package (BSP) is not required to provide the non-blocking functions if they are not needed.

7.5 DISPLAY_AMP_VALUE

Controls output of the amplitude value for detected targets.

if non-zero, the reported amplitude for detected target(s) is shown along with the range. (This option is enabled by default.)

7.6 DISPLAY_SAMPLE_NUM

Controls output of the sample number value for detected targets.

if non-zero, the sample number in which a target was detected is shown along with the range. (This option is enabled by default.)

8 REVISION HISTORY

REVISION DATE	REVISION	DESCRIPTION
08/02/2019	1.0	Advance Draft Release
11/01/2019	1.1	Title changed from "CH-101 Example Driver Hands On" to "SmartSonic Hello Chirp Application Hands-on Exercise".
08/05/2020	1.2	Updated for SonicLib 2.1 and new application file structure.
03/23/2023	1.3	Updated for SonicLib v3 and new application file structure.
03/31/2023	1.4	Changed to MPLAB X IDE.

This information furnished by InvenSense or its affiliates ("TDK InvenSense") is believed to be accurate and reliable. However, no responsibility is assumed by TDK InvenSense for its use, or for any infringements of patents or other rights of third parties that may result from its use. Specifications are subject to change without notice. TDK InvenSense reserves the right to make changes to this product, including its circuits and software, in order to improve its design and/or performance, without prior notice. TDK InvenSense makes no warranties, neither expressed nor implied, regarding the information and specifications contained in this document. TDK InvenSense assumes no responsibility for any claims or damages arising from information contained in this document, or from the use of products and services detailed therein. This includes, but is not limited to, claims or damages based on the infringement of patents, copyrights, mask work and/or other intellectual property rights.

Certain intellectual property owned by InvenSense and described in this document is patent protected. No license is granted by implication or otherwise under any patent or patent rights of InvenSense. This publication supersedes and replaces all information previously supplied. Trademarks that are registered trademarks are the property of their respective companies. TDK InvenSense sensors should not be used or sold in the development, storage, production or utilization of any conventional or mass-destructive weapons or for any other weapons or life threatening applications, as well as in any other life critical applications such as medical equipment, transportation, aerospace and nuclear instruments, undersea equipment, power plant equipment, disaster prevention and crime prevention equipment.

©2021—2023 InvenSense. All rights reserved. InvenSense, SmartMotion, MotionProcessing, MotionProcessor, SensorStudio, UltraPrint, MotionTracking, CHIRP Microsystems, SmartBug, SonicLink, Digital Motion Processor, AAR, and the InvenSense logo are registered trademarks of InvenSense, Inc. The TDK logo is a trademark of TDK Corporation. Other company and product names may be trademarks of the respective companies with which they are associated.



© 2023 InvenSense. All rights reserved.