SensorPIR EVB Sample Application

Eric Ryherd – DrZWave@silabs.com

# Introduction

goes here

# Theory of Operation

goes here

# Setup

goes here

# Diary

Below is a reverse chronological account of the development and issues found while developing this sample application. The most recent entries are on top.

## 2019-04-19 – Customization and initial peripheral work

The next step is to customize various files as one would do when developing a real product.

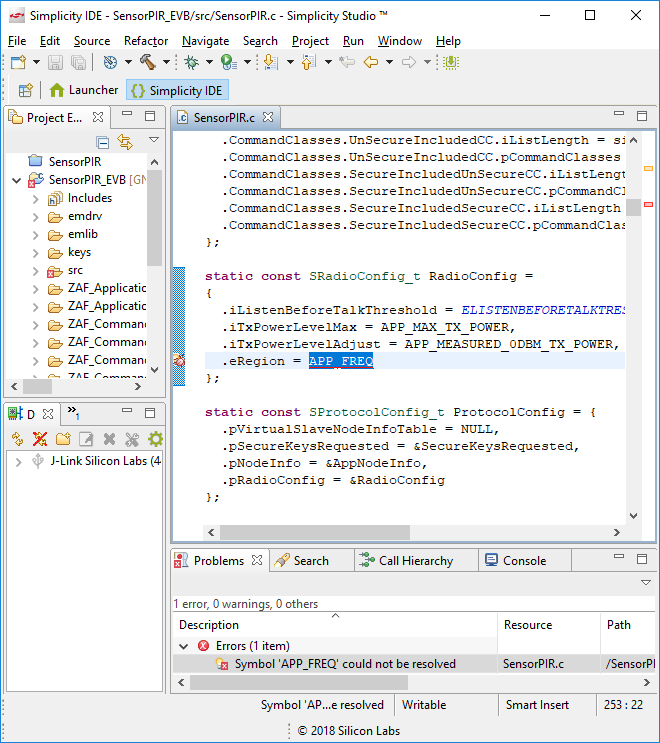
Follow INS14259 section 5 – How to develop a Z-Wave Plus application. This is the best getting started guide and walks thru the important sections.

1. config\_app.h
   1. Change APP\_VERSION and REVISION to fixed values – typically 1.0. The PATCH is not currently used so it doesn’t matter.
   2. When designing you own product you would next adjust the GENERIC and SPECIFIC types which come from the Device Type document
      1. To see the other values that can be used, right click on the GENERIC\_TYPE\_SENSOR\_NOTIFICATION and select Open Declaration
      2. The ZW\_classcmd.h file will open which has all the various choices for these variables
   3. Chose the APP\_ROLE/NODE/ICON types as desired. In this case we’ll leave them as-is
   4. APP\_MANUFACTURER\_ID must be changed to your company name – again right click on MFG\_ID\_ZWAVE and scroll thru the list to find your company.
      1. In this case I’ve chosen MFG\_ID\_EXPRESS\_CONTROLS which was my previous consulting firm
   5. APP\_PRODUCT\_TYPE\_ID and APP\_PRODUCT\_ID MUST be a unique number within your company. It is recommended to maintain a file with the product name and corresponding values for these two number. The values are not important other than they must be unique and must remain the same for all versions of this product. If the features of the product really change in the future you may want to consider using a new value for this but then you must go thru full Z-Wave certification
   6. The MFG\_ID, PROD\_TYPE and PROD\_ID uniquely identify your product via the 48 bit number. This number is effectively the “signature” or fingerprint of the device.

## 2019-04-15 – Project Kickoff

The 700 series was officially released at a few weeks ago and this example will help find issues in Z-Wave product development and help the FAE staff to develop a deeper understanding of the 700 series and EFR series in general.

The first step is to read INS14280 Getting Started for End Devices which is available in Simplicity Studio.

1. Extract the SensorPIR sample application from Simplicity Studio (SS).
   1. Start SS, plug in the WSTK and ZGM130S devkit board, select the Launcher Perspective (window->perspective->Launcher) which should open by default
   2. Close other projects and perspectives (files, debugger, IDE etc)
   3. Highlight the ZGM130S in the Project Explorer in the upper left corner
   4. A list of software examples should pop up under the Z-Wave Apps. Click on the triangle (or the green +) to expand all the sample apps
   5. Click on SensorPIR – this causes the project to be copied to your local default directory and the IDE perspective opens
   6. Rename the project if desired. In this case it is called SensorPIR\_EVB to distinguish it from copies of the standard SensorPIR project. Note you MUST rename it while still in your standard workspace as once you move it is harder to rename.
   7. Right click on the SensorPIR project in the Project Explorer in the upper left and select MOVE
   8. Change the location to the desired directory – in this case it is in the gitub directory on my PC
      1. Note that this is the method to create the project from scratch. If you want to start from this finished project, then follow github instructions to clone the project.
      2. Only the project needs to be checked into github or whichever source code control system you use.
      3. Many people also check in the entire Simplicity Studio release that was used to build the project but that should be done in a separate repository and used on several projects.
2. Attempt to compile the project – click on the Hammer Icon
3. The build will FAIL – that is expected – click on the Problems tab in the bottom right pane
4. There should be 1 ERROR – Click on the > and then double click on the APP\_FREQ error
5. The SensorPIR.c file should open in the upper left with the failing line in the center of the window
6. Change APP\_FREQ to REGION\_US or whichever region you need.
   1. To see the list of applicable regions, set it to REGION\_US, highlight it, right click and select Open Declaration and you be taken to ZW\_radio\_api.h with the typedef of the available regions.
7. Build the project again which should pass and yield: Build Finished at the bottom of the Console window on the lower right
8. If the build works, then click on the bug icon to download the code into the WSTK
9. The Debug Perspective should open
   1. If SS complains about SWO busy or the device needs to be unlocked, click on OK or Force Unlock which should allow the download to complete. Make sure no other copies of SS or other Silabs tools are running.
10. Typically, the debugger will run the code until it reaches Main which is buried in the FreeRTOS library and thus is not visible to the user.
11. Click on the Run Icon 
12. Then click on the Pause Icon 
13. The center window should display the source code that is about to execute
14. Click on the various single step buttons to be familiar with single stepping the source code.
15. In the upper left window is the call stack. The call stack is a list of routines that have been executed to get to this line of code. This is particularly useful when a breakpoint is set in an error routine to figure out the path the code took to get into this error.
16. In the source window, double clicking the blue bar on the left on a line of code will set a small blue dot indicating a Breakpoint has been set. Clicking run will restart the code which will stop when the breakpoint has been reached. To clear all breakpoints click on Run->Remove all Breakpoints.
17. The next step is to enable the PRINT statements in the code which make debugging easier.
18. Right click the SensorPIR\_EVB project in the upper left corner – select properties
19. C/C++Build->Settings->Tools Settings->GNU Arm C compiler->Symbols
20. Click on DEBUG=1 and the Edit Icon
21. Change it to DEBUGPRINT=1 then click OK
22. Click on DISABLE\_USART1=1 and clock on the X to remove it
23. Click on the bug icon again which will rebuild and re-download
24. Click on the Launcher Perspective and then the green Wrench Icon and then scroll to the Device Console and double click that
25. Click on the Serial 1 tab AND THEN AT THE BOTTOM OF THE SCREEN click in the box and press ENTER
    1. Note that the connection icon will show:  indicating the UART is NOT connected.
    2. After pressing ENTER the icon should change to:  indicating the UART Is now connected
26. Click on the Debug perspective and then click on RUN. The serial window should display a lot of messages starting with:

----------------------------

Z-Wave Sample App: Sensor PIR

SDK: 7.11.1 ZAF: 10.11.0.43707 [Freq: 1]

-----------------------------

BTN0: Send battery report

BTN1: Toggle learn mode

Hold 5 sec: Reset

BTN2: Activate PIR event

(leave deactivated to allow going to sleep)

LED1: Learn mode + identify

-----------------------------

1. The print messages display a lot of cryptic data which you usually have to search for in the source code to figure out what the messages mean. But these print messages provide an easy to read high level view of what’s happening.
2. Pressing the buttons on the standard Button Board will cause various actions like Learn mode as shown here.
3. We now have a working example. You can start the Zniffer and the PC Controller and join the Devkit to the PC controller and send/receive Z-Wave commands. Note that pressing button 2 is the “fake” way to generate a Motion Sensor detect action. The next part of this example we’ll customize the sample app to use the OCCUPANCY-EXP-EVB board which has a real motion sensor and other sensors on it.