STM32CubeWiSE-RadioCodeGenerator software description

Introduction

This document describes the STM32CubeWiSE-RadioCodeGenerator (STM32CubeWiSEcg) SW package with the STM32WL33 MRSUBG sequencer code generator.

STM32CubeWiSE-RadioCodeGenerator is a PC application that is used to build a flowgraph that defines which transceiver actions to execute under which condition, using the MRSUBG sequencer driver.

STM32WL33 Sub-GHz radio contains this sequencer, which is a state-machine like mechanism which allows for autonomous management of RF transfers, without any need for CPU intervention. If CPU intervention is required, interrupts can be defined. Transceiver actions can be arranged in a flowgraph. In this document, the individual transceiver actions are referred to as *SeqActions*.

However, source code is not the best representation for flowgraphs, since it conceals their logical and temporal structure. STM32CubeWiSE-RadioCodeGenerator addresses this issue by providing a graphical method to build flowgraphs, and then exporting the generated flowgraphs as C source code for integration into user applications.

The flowgraph definition is stored in the microcontroller RAM in the form of:

- A set of *ActionConfiguration* RAM tables, linked to each other using pointers. These pointers define the *SeqActions*, that is, the type of action (for example, transmission, reception, abort), as well as *SeqAction*-specific radio parameters and conditions for action transmissions.
- A unique *GlobalConfiguration* RAM table. This defines the entry point of the flowgraph (the first *SeqAction* to execute), as well as some default flag values and common radio parameters.

Radio parameters, which can be configured individually for each *SeqAction*, are stored in one of the dynamic registers, whose contents are part of the *ActionConfiguration* RAM table. Radio parameters that are fixed over the whole execution of the flowgraph (unless they are modified during a CPU interrupt), are stored in static registers, whose contents are part of the *GlobalConfigurtion* RAM table.

1 General information

1.1 Licensing

This document describes software that runs on the STM32WL33 Arm® Cortex ® -M0+ based microcontroller.

Note: Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

arm

1.2 Related documents

Table 1. Document references

Number	Reference	Title
[1]	RM0511	STM32WL33xx Arm® based sub-GHz MCUs

2 Getting started

This section describes all of the system requirements to run STM32CubeWiSE-RadioCodeGenerator. It also details the software-package installation procedure.

2.1 System requirements

The STM32CubeWiSE-RadioCodeGenerator application has the following minimum requirements:

- PC with an Intel® or AMD® processor running the Microsoft® Windows 10 operating system
- At least 2 Gbytes of RAM
- USB ports
- Adobe Acrobat reader 6.0

2.2 STM32CubeWiSE-RadioCodeGenerator SW package setup

Perform tthe following steps:

1. Extract the content of the *stm32wise-cgwin.zip* file into a temporary directory.
2. Extract and launch the *STM32CubeWiSE-RadioCodeGenerator_Vx.x.x.exe* file and follow the on-screen instructions.

2.3 STM32CubeWiSE-RadioCodeGenerator SW package files

The STM32CubeWiSE-RadioCodeGenerator SW package files are organized into the following folders:

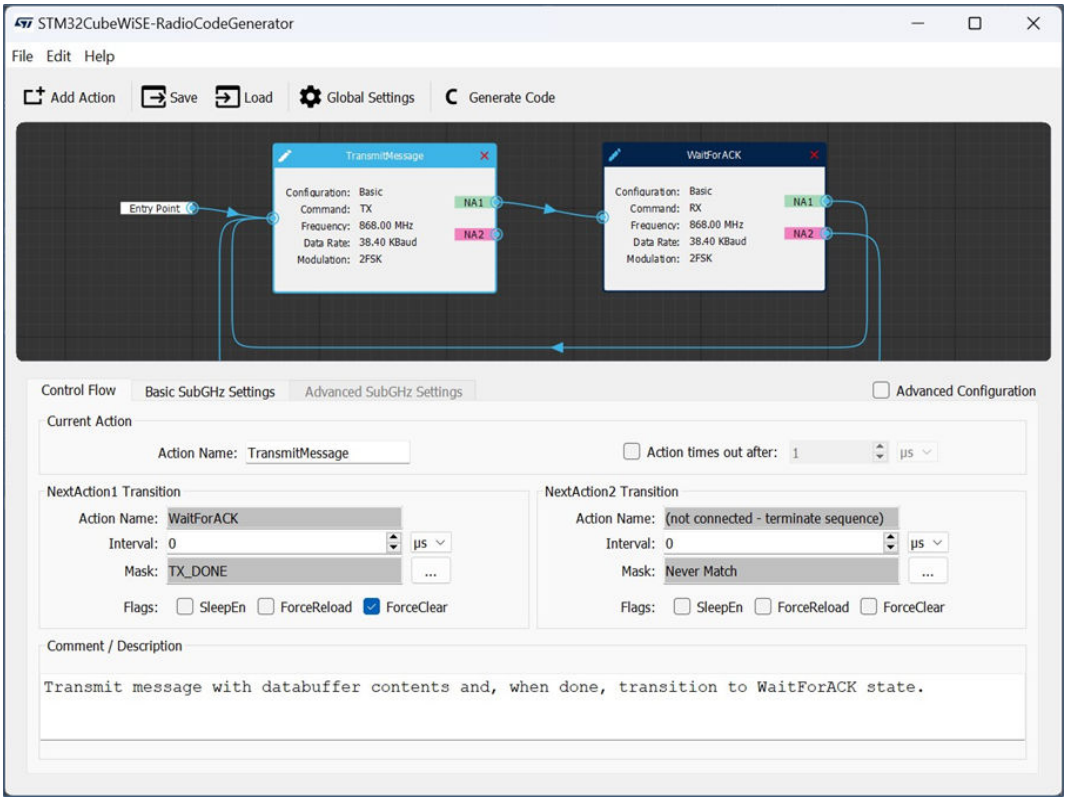
- **app**: contains *STM32CubeWiSE-RadioCodeGenerator.exe*
- **examples**: this folder is organized into the following subfolders:
  - **code**: this folder contains the flowgraphs example already exported as C code, ready to be injected into an application project
  - **flowgraphs**: this folder stores some examples scenario of autonomous MRSUBG sequencer operations

Release notes and license files are located in the root folder.

3 STM32CubeWiSE-RadioCodeGenerator software description

This section describes the main functions of the STM32CubeWiSE-RadioCodeGenerator application. To run this utility, click on the STM32CubeWiSE-RadioCodeGenerator icon.

Figure 1. Main application window of STM32CubeWiSE-RadioCodeGenerator



DT58550V1

After launching STM32CubeWiSE-RadioCodeGenerator, the main application window appears. It consists of:

- A global menu and toolbar
- The visual drag-and-drop representation of the flowgraph
- The *SeqAction* configuration section (only visible if a *SeqAction* is currently being edited)

3.1 Building a flowgraph

3.1.1 Basics

Flowgraphs are built in two steps:

1. Add *SeqActions* to the flowgraph. This can be done using the "*Add Action*" button in the toolbar, using the global menu (Edit → Add Action) or with the "Ctrl+A" shortcut.
2. Connect *SeqActions* to the entry point and to each other by drawing action transition arrows.

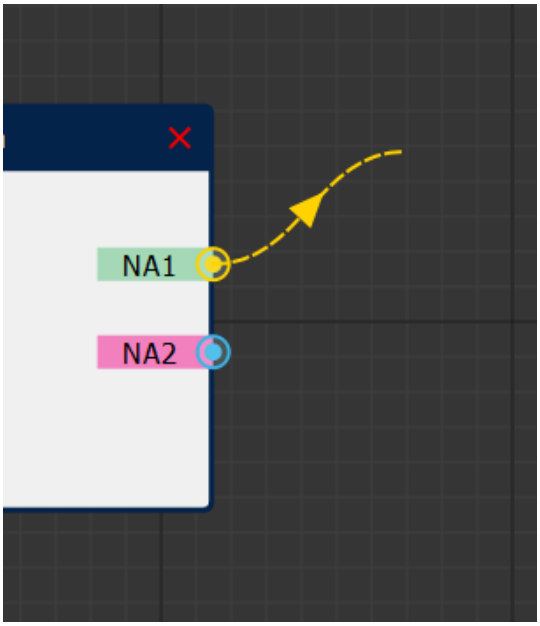
The conditions under which these transitions occur are defined later (see Section 3.2.1: Control flow).

3.1.2 Navigating the flowgraph, dragging actions

By dragging the checkerboard background of the flowgraph with the mouse pointer (left click), the viewport on the flowgraph can be adjusted. The mouse scroll wheel can be used to zoom in and out. Clicking anywhere on an action (except for the output ports, the delete button and the edit button) to select an action. Actions can be arranged in the flowgraph by dragging them with the left mouse button.

3.1.3 Adding action transitions

Figure 2. Drawing a transition for NextAction1



DT58551V1

As shown in Figure 2, each action has two "*output port*", called *NextAction1* (NA1) and *NextAction2* (NA2), which can be connected to *SeqActions* that are executed after the action has completed. For example, *NextAction1* could be used to execute some action if the current action was successful and *NextAction2* could be triggered in case of failure.

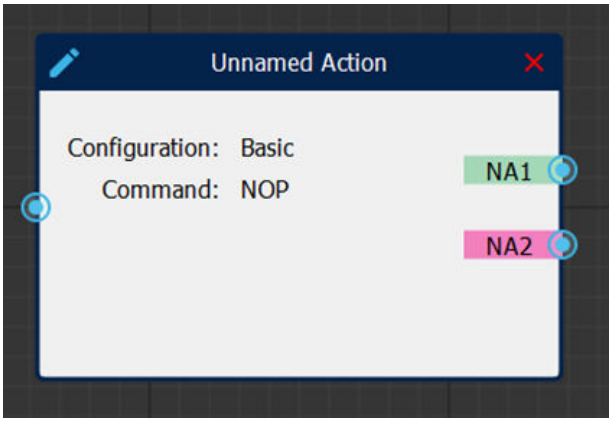
To create an action transition, hover the mouse pointer over one of the output ports, press the left mouse button and move the mouse pointer to drag a transition arrow. Move the mouse pointer over the input port on the left of some other *SeqAction* and release the left mouse button to make the connection permanent. To remove an action transition, just repeat the steps for creating an action transition, but release the left mouse button somewhere over the checkerboard background.

If an output (*NextAction1*, *NextAction2*) is left unconnected, the sequencer terminates if this next action is triggered.

Make sure to also connect the "*Entry Point*" to some *SeqAction*'s input port. This *SeqAction* is the first to be executed as soon as the sequencer is triggered.

3.1.4 Editing and deleting actions

Figure 3. SeqAction with edit button and delete button



DT58552V1

SeqActions can be edited by clicking on the pencil button on the top left of *SeqAction*. It can be deleted by clicking on the red cross on the top right (see Figure 3). Deleting a *SeqAction* also removes any incoming and outgoing action transitions.

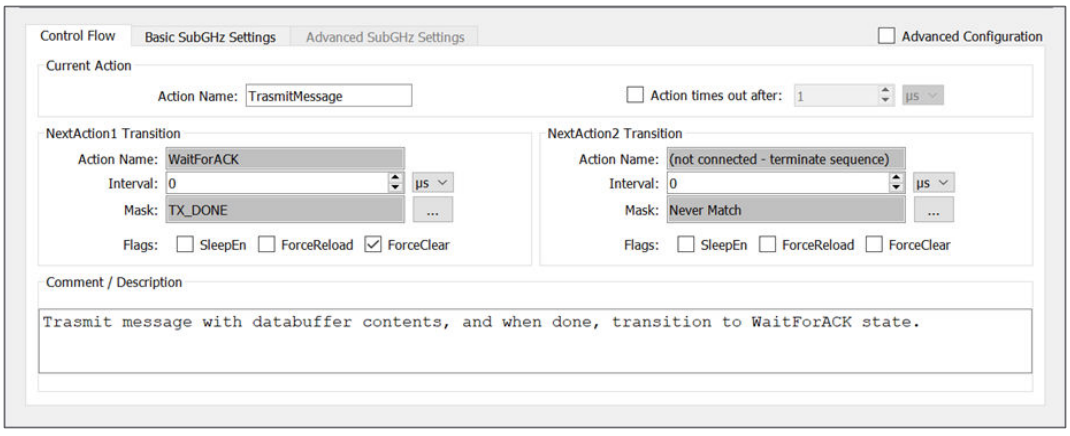
3.2 SeqAction configuration

SeqActions can be configured through a tabbed configuration interface accessible through the pencil button on the top left of each action in the flowgraph. This interface essentially configures the contents of the *ActionConfiguration* RAM table for the particular action, consisting of both control flow-related configuration options as well as the dynamic register contents. The dynamic register contents can either be configured manually with complete control over every register value (see Section 3.2.3: Advanced radio configuration) or through a simplified interface (seeSection 3.2.2: Basic radio configuration). The simplified interface should be sufficient for almost all use cases.

3.2.1 Control flow

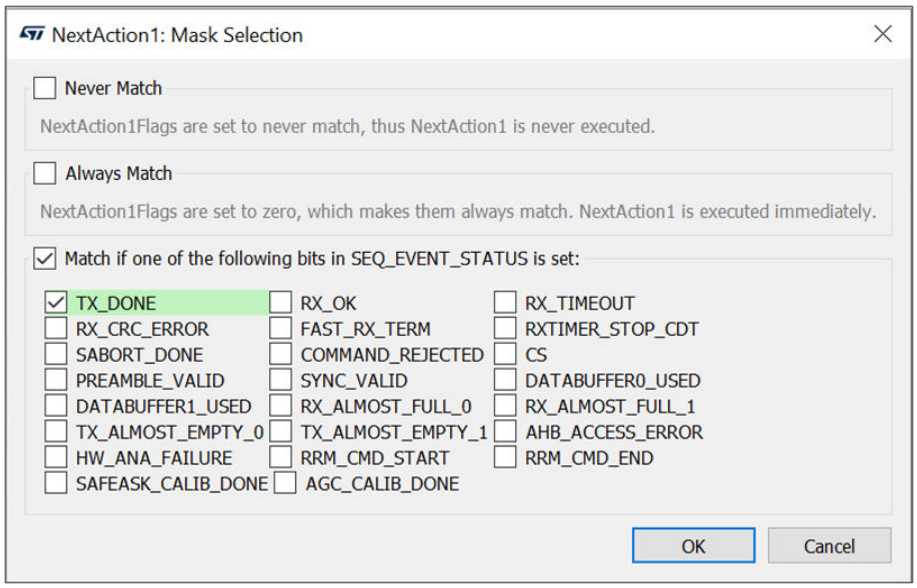
The control flow tab (see Figure 4) contains some basic configuration options such as action name and action timeout interval. The action name is not only used for display in the flowgraph, but is also carried over to the generated source code.

Figure 4. SeqAction control flow configuration tab



DT58553V1

Figure 5. Match mask selection for NextAction1, only one flag (TX_DONE) is selected here



DT58554V1

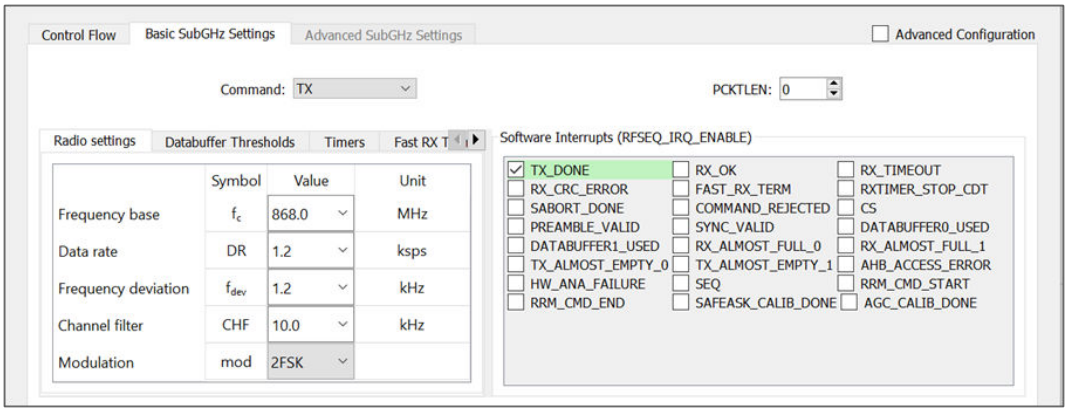
The control flow tab (see Figure 4) contains some basic configuration options such as action name and action timeout interval. The action name is not just used for display in the flowgraph, but also carried over to the generated source code.

Most importantly, the control flow tab configures the condition on which a transition to NextAction1 / NextAction2 depends on as well as transition interval and flags. The transition condition can be configured by clicking on the button labelled "...", which makes the mask selection dialog shown in Figure 5 appear. The transition interval modified the NextAction1Interval / NextAction2Interval property of the RAM table. Refer to the STM32WL33 reference manual [1] for more information on the meaning of this interval and the significance of the *SleepEn* / *ForceReload* / *ForceClear* flags.

Furthermore, a short description of the *SeqAction* block can be added on this tab. This description is only used for documentation purposes and carried over to the generated source code as a source code comment.

3.2.2 Basic radio configuration

Figure 6. Basic SubGHz settings tab



DT58555V1

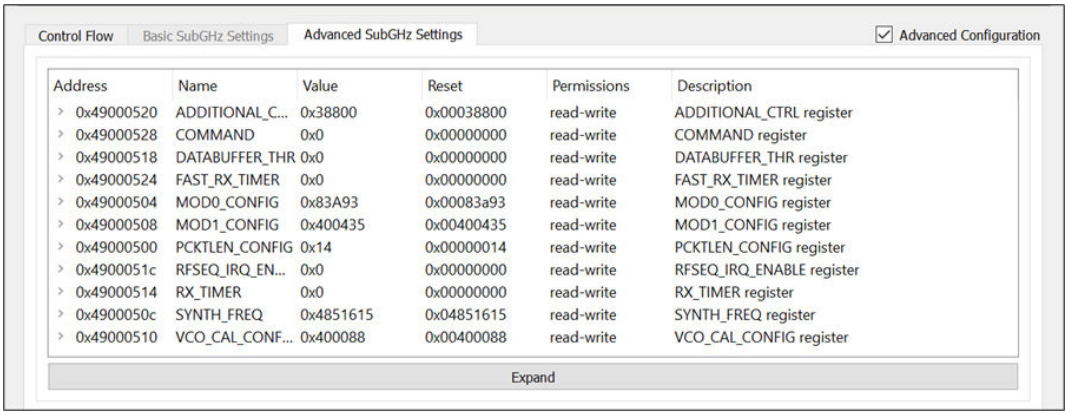
The basic radio configuration tab can be subdivided into three parts:

1. A section at the top where two of the most important parameters of any action are configured: the command to execute (TX, RX, NOP, SABORT, and so on) and, if applicable, the length of the packet to transfer.
2. A section on the left where the actual radio parameters such as: carrier frequency, data rate, modulation properties, data buffer thresholds and timers are configured.
3. A section on the right where the CPU interrupts can be individually enabled. An interrupt handler is generated for each of the ticked interrupts. This basically configures the contents of the RFSEQ_IRQ_ENABLE register.

Refer to the STM32WL33 reference manual [1] for the meaning of the various radio parameters.

3.2.3 Advanced radio configuration

Figure 7. Advanced radio configuration tab



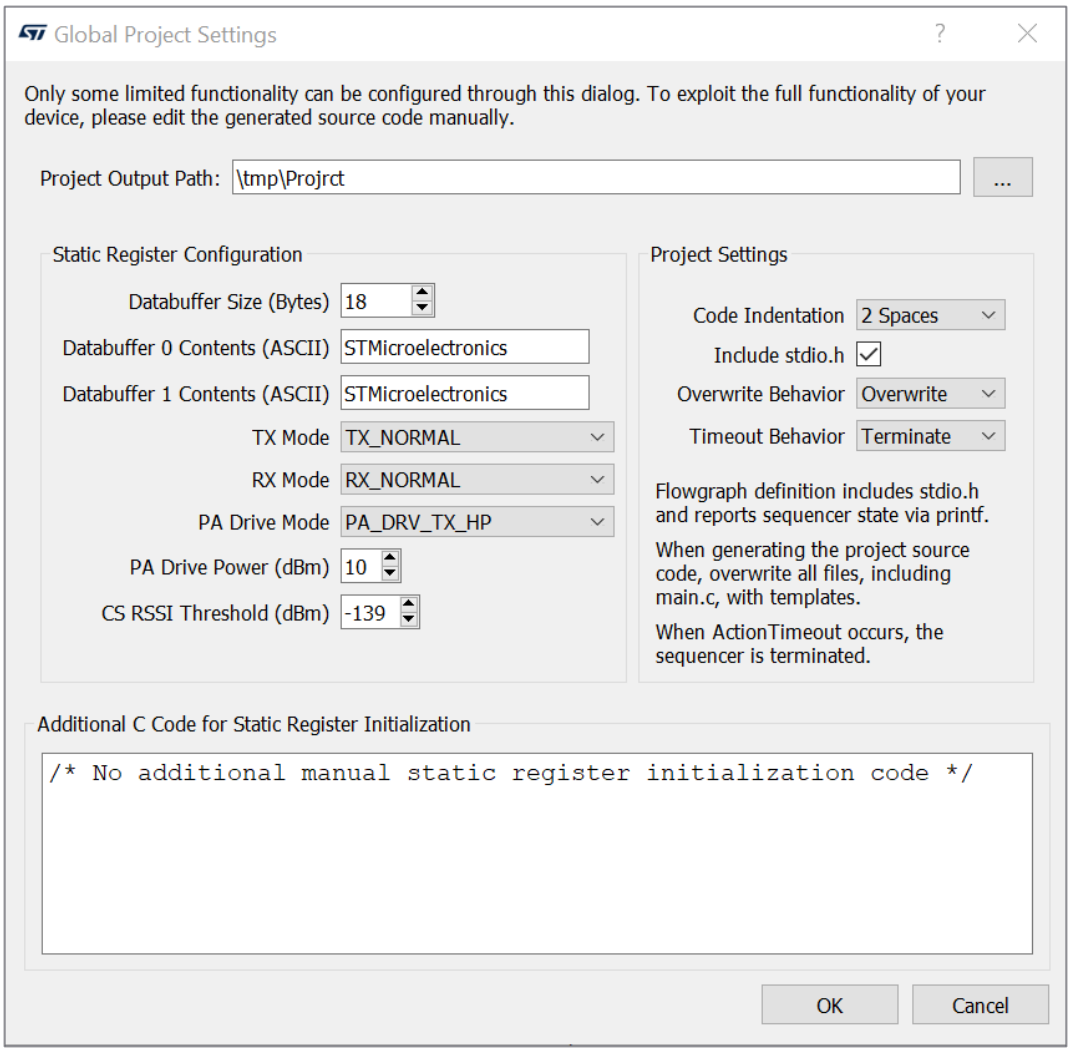
DT58556V1

If the configuration options exposed through the basic radio configuration tab (Section 3.2.2: Basic radio configuration) are insufficient, the advanced STM32WL3x radio configuration tab allows the setting of arbitrary dynamic register contents. The advanced configuration tab is enabled by ticking the *Advanced Configuration* checkbox to the top right of the tabbed configuration interface.

It is not possible to use both basic and advanced configuration at the same time, the user must select one or the other. However, it is of course also possible to manually edit the generated source code afterwards and to add potentially missing configuration options.

3.3 Global configuration dialog

Figure 8. Global project settings dialog with example values



The "*Global Project Settings*" dialog can be accessed through the "*Global Settings*" toolbar button. The dialog contains both configuration options for the static register contents as well as additional project settings. Note that only a small fraction of static register configuration options can be configured through this dialog. These options are only provided to speed up application prototyping applications with STM32CubeWiSE-RadioCodeGenerator. It is usually expected that the static register contents are set up in the application's manually-written source code.

The meaning of the other project settings is explained in the dialog itself.

Additional C code that is inserted just before creating the *Global Configuration* RAM table from the static register contents may also be provided. This field may be used to set up static register values which are inaccessible through the provided static register configuration mask.

3.4 Code generation

The flowgraph can be translated into a complete project C source code by pressing the *Generate Code* button in the toolbar. The generated project folder does not contain project files for IAR, Keil®, or GCC. These files must be added manually to the STMWL3x project.

This is the generated project folder structure:

- Project folder
  - *inc*
    - *SequencerFlowgraph.h*: header file for *SequencerFlowgraph*.c, static. Do not edit this.
    - *stm32wl3x_hal_conf.h*: STM32WL3x HAL configuration file, static.
  - *src*
    - *SequencerFlowgraph.c*: flowgraph definition. This is the important file that uses the sequencer driver to define the global-configuration and action-configuration RAM tables. Autogenerated, do not edit.
    - *main.c*: Project main file that demonstrates how to load and apply the flow-graph definition. Static, modify this as needed.

To edit *main.c* or stm32wl3x_hal_conf.h, select overwrite behavior *Keep* in the project settings. This way, only *SequencerFlowgraph.c* gets overwritten.

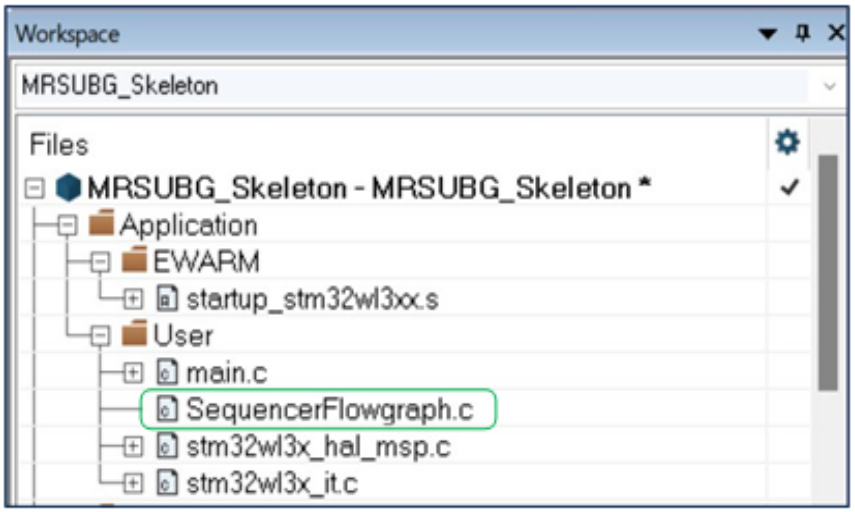
3.5 How to import generated code into a CubeMX example

To import a project generated by STM32CubeWiSE-RadioCodeGenerator into a CubeMX example (MRSUBG_Skeleton), it is necessary to follow the following steps:

1. Open the folder contained the files generated by STM32CubeWiSE-RadioCodeGenerator and copy “*Inc*” and “*Src*” folders.
2. Paste the two folders on “*MRSUBG_Skeleton”* folder overwriting the two already present.
3. Open “*MRSUBG_Skeleton*” project in one of the following IDEs:
   - EWARM
   - MDK-ARM
   - STM32CubeIDE

4. Inside the "*MRSUBG_Skeleton*" project, add *"SequencerFlowghraph.c"* file:
   – For an EWARM project, the path to add the file is the following: *MRSUBG_Skeleton\Application\User*

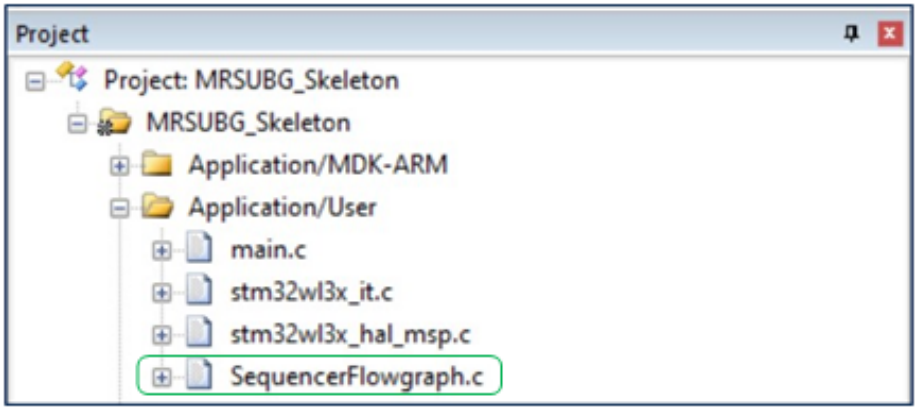
Figure 9. EWARM project



DT58561V1

   – For an MDK-ARM project, the path to add the file is the following: *MRSUBG_Skeleton\Application/User*

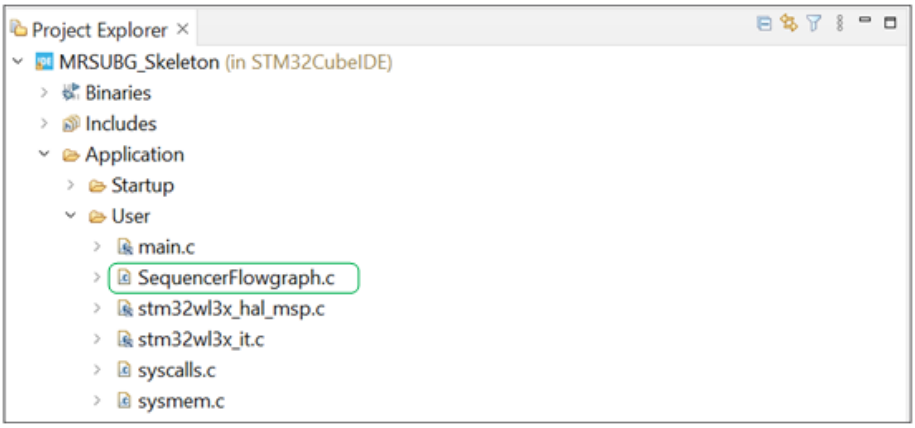
Figure 10. MDK-ARM project



DT58562V1

   – For an STM32CubeIDE project, the path to add the file is the same: *MRSUBG_Skeleton\Application\User*

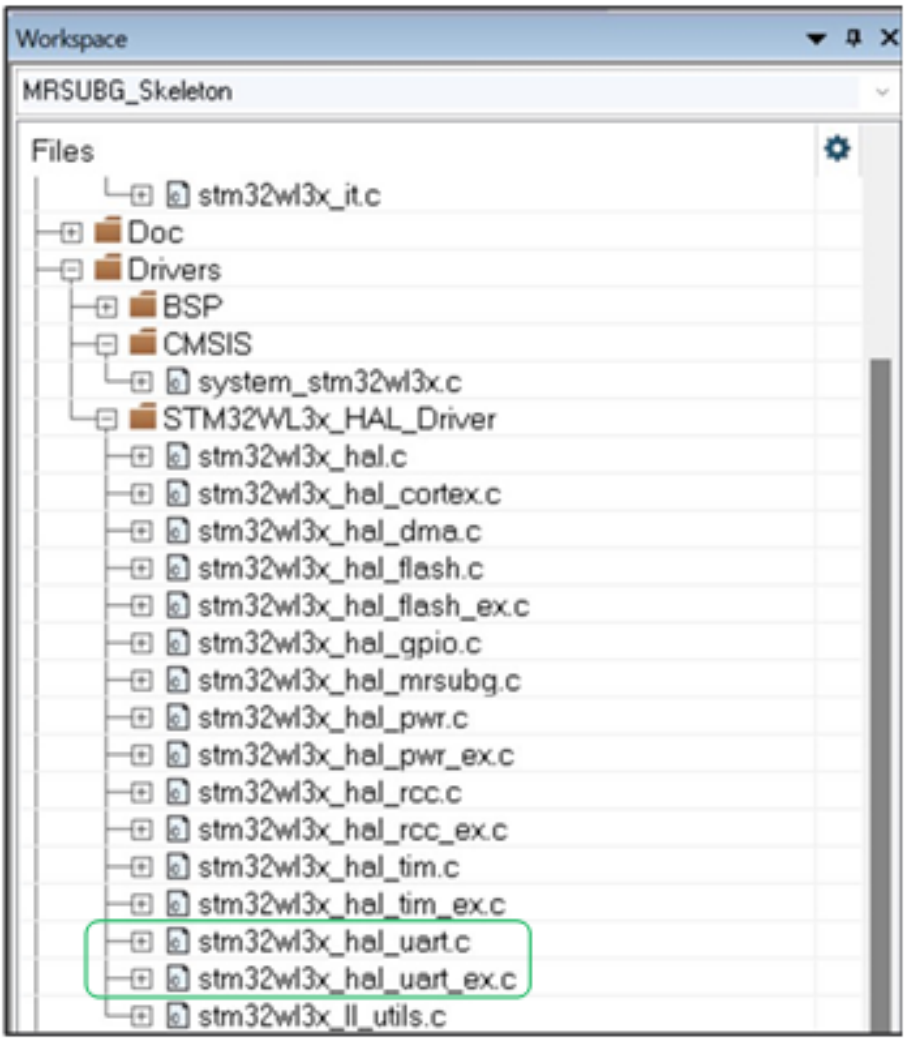
Figure 11. **STM32CubeIDE project**



DT58563V1

5. Inside the *MRSUBG_Skeleton* project, add *stm32wl3x_hal_uart.c* and *stm32wl3x_hal_uart_ex.c* files to the following path: *MRSUBG_Skeleton\Drivers\STM32WL3x_HAL_Driver*. The path is the same for all IDEs. The two files are located on *Firmware\Drivers\STM32WL3x_HAL_Driver\Src*.

Figure 12. Drivers folder for EWARM project



DT58564V1

6. To use COM features, *stm32wl3x_nucleo_conf.h* file, located on *Firmware\Projects\NUCLEO-WL33CC\Examples\MRSUBG\MRSUBG_Skeleton\Inc*, must be modified setting *USE_BSP_COM_FEATURE* and *USE_COM_LOG* to 1U:

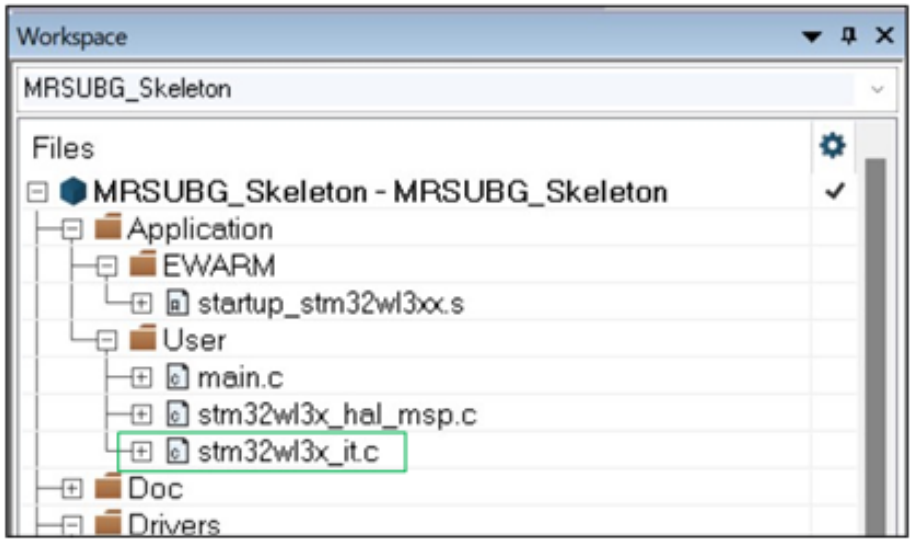
Figure 13. stm32wl3x_nucleo_conf.h file

```
30  /* Usage of nucleo board */
31  #define USE_NUCLEO_64       1U
32
33  /* Usage of COM feature */
34  #define USE_BSP_COM_FEATURE 0U
35  #define USE_COM_LOG         0U
36
37  /* Button interrupt priorities */
38  #define BSP_B1_IT_PRIORITY 0x0FUL  /* Default is lowest priority level */
39  #define BSP_B2_IT_PRIORITY 0x0FUL  /* Default is lowest priority level */
40  #define BSP_B3_IT_PRIORITY 0x0FUL  /* Default is lowest priority level */
41
```

DT58565V1

7. Copy the following code into *"stm32wl3x_it.c"*, located in *MRSUBG_Skeleton\Application\User*.

Figure 14. Application section for EWARM project



DT58566V1

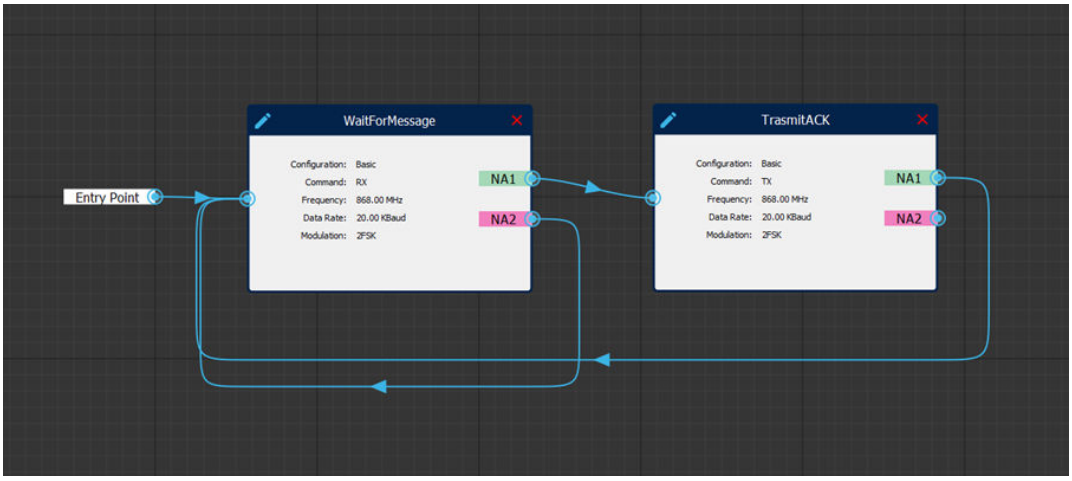
```
/*** @brief This function handles MRSUBG interrupt. */
void MRSUBG_IRQHandler(void)
{
  /* USER CODE BEGIN MRSUBG_IRQn 0 */

  /* USER CODE END MRSUBG_IRQn 0 */
  HAL_MRSubG_IRQHandler();
  /* USER CODE BEGIN MRSUBG_IRQn 1 */

  /* USER CODE END MRSUBG_IRQn 1 */
}
```

3.6 Flowgraph examples

Four example flowgraphs are provided alongside the source code. These examples may be loaded into STM32CubeWiSE-RadioCodeGenerator by clicking the “Load” button in the toolbar.

3.6.1 AutoACK_RX

Figure 15. AutoACK_RX example



DT58559V1

The *Auto-ACK* demo illustrates how two STM32WL33 devices can automatically talk to each other with minimal CPU intervention, with the help of the sequencer hardware.

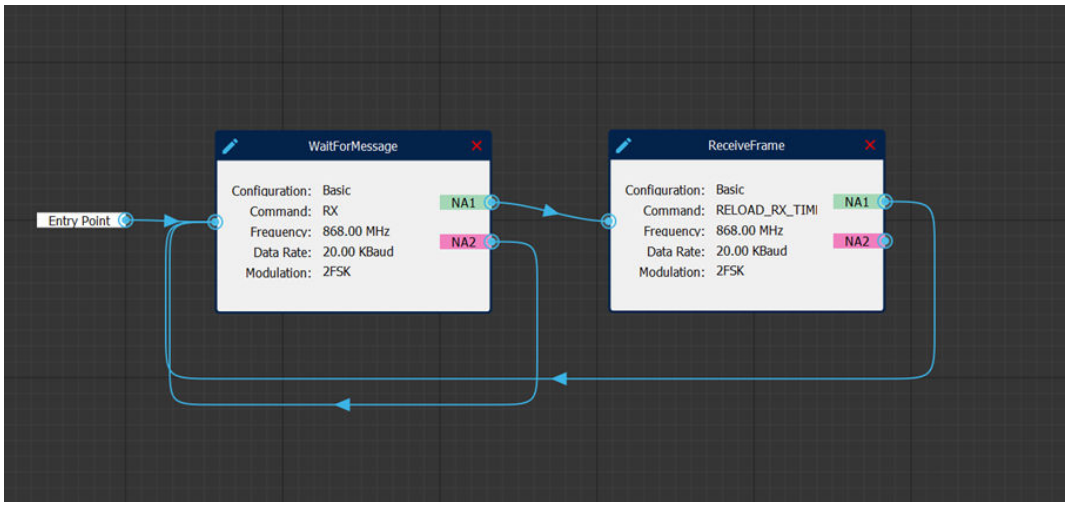
This flowgraph implements the behavior (*Auto-Transmit-ACK*) of device A. In device A, the sequencer is initialized in a receiving state (WaitForMessage), in which it waits for a message to arrive.

Once a valid message arrives, the sequencer automatically transitions into an transmit state (TransmitACK), in which an ACK packet is sent as a response, without CPU intervention. Once this is finished, the sequencer is reset into its initial WaitForMessage state.

This flowgraph implements the same behavior as the *MRSUBG_SequencerAutoAck_Rx* example from the *Examples\MRSUBG* folder of the STM32Cube WL3 Software package. If AutoACK_RX is flashed on one device, A, and AutoACK_TX is flashed on some device, B, the two devices send messages back and forth, as in a ping-pong game.

3.6.2 AutoACK_TX

Figure 16. AutoACK_TX example



DT58567V1

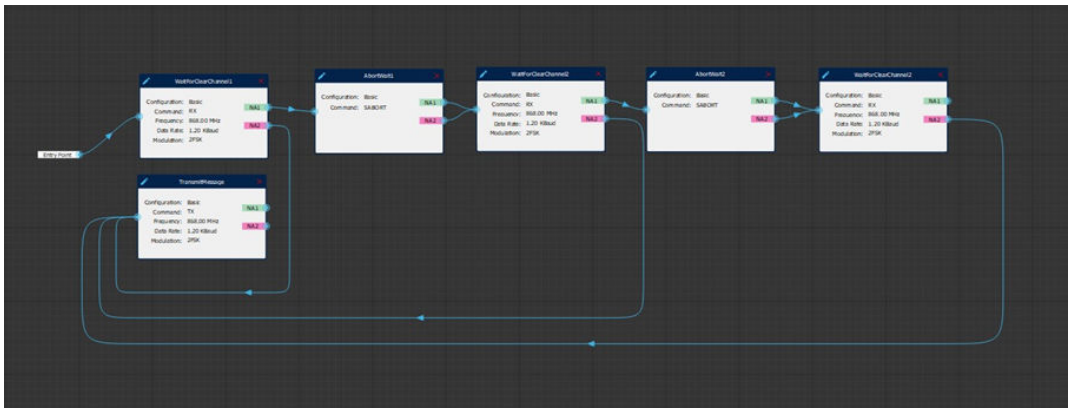
The *"Auto-ACK"* demo illustrates how two STM32WL33 devices can automatically talk to each other with minimal CPU intervention with the help of the sequencer hardware.

This flowgraph implements the behavior ("*Auto-Wait-for-ACK*") of device B. In device B, the sequencer is initialized in a transmitting state (TransmitMessage), in which it transmits a message. Once the transmission is finished, it auomatically transitions into a receiving state where it waits for an acknowledgement from device A (WaitForACK). Once a valid acknowledgement arrives, the sequencer is reset into its initial TransmitMessage state and the whole process starts again. In case no ACK is received within 4 seconds, a timeout is triggered and the sequencer returns to state TransmitMessage anyway.

This flowgraph implements the same behavior as the *"MRSUBG_SequencerAutoAck_Tx"* example from the *Examples\MRSUBG* folder of the STM32Cube WL3 Software package. If AutoACK_RX is flashed on one device, A, and AutoACK_TX is flashed on some other device, B, the two devices send messages back and forth, as in a ping-pong game.

3.6.3 Listen before talk (LBT)

Figure 17. Listen before talk example

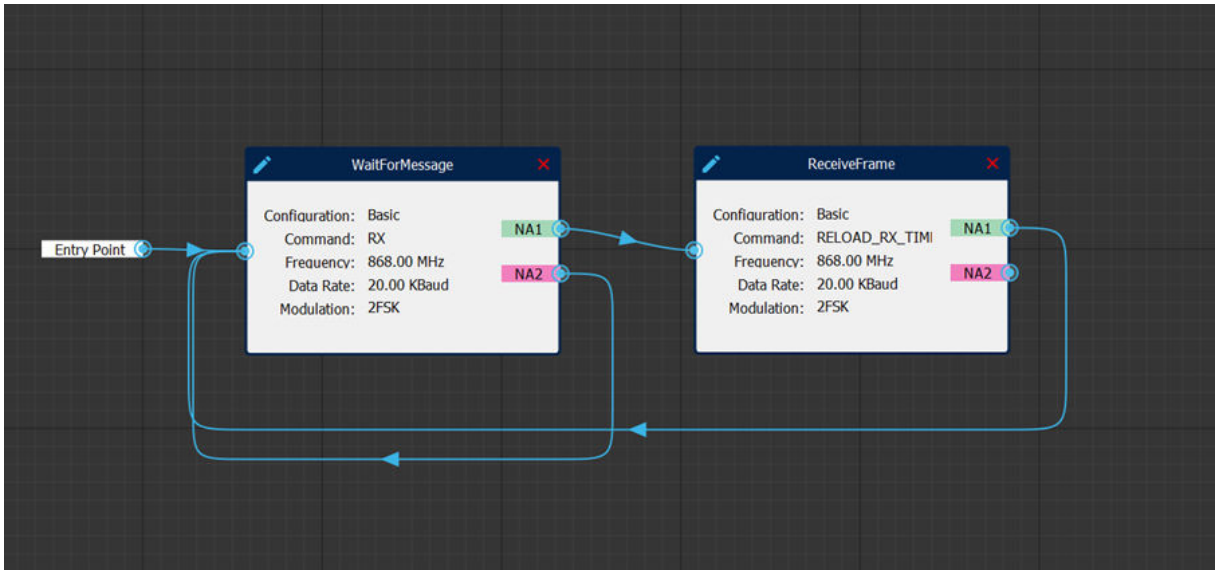


DT58568V1

This example is taken from the STM32WL33 reference manual [1]. Refer to that manual for further details of this example.

3.6.4 Sniff mode

Figure 18. **Sniff mode example**



This example is taken from the STM32WL33 reference manual [1]. Refer to that manual for further details of this example.

Revision history

Table 2. **Document revision history**

Date	Version	Changes
21-Nov-2024	1	Initial release.

Contents

List of tables

List of figures