GUIDES 558

Metwear Custom BLE Firmware

Posted On July 11, 2016 Athanasios Tasoglou

Contents [hide]

Metawear "Hacked"!

Do you own one of the little MetaWear developer boards? And you want to use your own custom firware? Awesome! You are on the right track! Metwear is a neat, tiny ARM+Bluetooth LE Platform for developing Wearable products. MetaWear comes with different hardware implementations serving different application needs. MetaWear uses the award winning BLE SoC, the nRF51x. When shipped, the MetaWear comes with its own pre-installed firmware suited for rapid-android/IOS typeof-prototyping. Pretty cool!

Nevertheless, I know that out there, there are more demanding users that want to implement their own custom firmware! This guide explains how to achieve this using any of the Nordic's Softdevices, the GNU toolchain under Linux. However, the idea on how to achieve this, is the same with any type of toolchain.

1 Metawear "Hacked"!
2 SoftDevice Configuration Overview
3 nRF51x IC revision overview
3.1 MetaWear RG: EYSFCNZXX
4 Step 1: Memory resource map and usage
4.1 Defining SizeOfRam
4.2 Defining SizeOfProgram
4.3 Defining APP_RAM_BASE & APP_CODE_BASE
4.4 Defining Stack and Heap size
4.5 Putting everything together in the linker...
4.5.1 Configure Stack and Heap size
5 Step 2: Configuring the Clock
6 Sources

I will be using the nRF51-DK as Debugger/Programmer and the ~~MetaWear R Series~~ as the platform.
Click below to see how to use the nRF51-DK as Debugger/Programmer.

PROGRAMMING THE METAWEAR WITH THE NORDIC NRF51-DK

SoftDevice Configuration Overview

To adapt our application to the MetaWear but also to any other custom firmware we need to perform two steps:

nRF51x IC revision overview

This section shows the packet and memory variants for each nRF51x IC revision. These tables are important because as we will see later, the MetaWear RG and the nRF51 DK use different IC revisions and as a result there is a difference in the RAM memory.

nRF51422 | nRF51822

nRF51422 IC revision	Packet variant	Build code	Package	Flash [kB]	RAM [kB]
1	QF AA	C0	QFN48	256	16
	CE AA	A0A	WLCSP		
2	QF AA	Ex0	QFN48	256	16
	QF AB	A00		128	
	CE AA	Bx0	WLCSP	256	
3	QF AA	Fx0	QFN48	256	16
	QF AB	Bx0		128	
	QF AC	Ax0		256	32
	CD AB	Ax0	WLCSP	128	16
	CE AA	Cx0		256	
	CF AC	Ax0			32

MetaWear RG: EYSFCNZXX

The MetaWear RG uses the the Taiyo Yuden EYSFCNZXX SoC with the following specs:

Chip

Nordic SoC:

nRF51822

32 MHz external clock

256 kB Flash

16 kB RAM

More information

- TAIYO YUDEN Website
- PowerPoint Overview

Step 1:

Memory resource map and usage

The memory map for program memory and RAM at run time with the SoftDevice enabled is illustrated in below. We will focus on the RAM memory here (right image) as it is important to adapt the linker-configuration to reflect the changed SoftDevice RAM requirement.

Defining SizeOfRam

As we have see before we might have either 16 kB or 32kB of RAM. Which means:

- `SizeOfRam = 0x4000` for 16 kB RAM
- `SizeOfRam = 0x8000` for 32 kB RAM

This translates to the following:

RAM	Start address	End Address
16 kB RAM	0x2000 0000	0x2000 **4000**
32 kB RAM	0x2000 0000	0x2000 **8000**

Defining SizeOfProgram

As we have see before we might have either 128 kB or 256kB of program memory (flash). Which means:

- `SizeOfProgram = 0x20000` for 128 kB Flash
- `SizeOfProgram = 0x40000` for 256 kB Flash

Defining APP_RAM_BASE & APP_CODE_BASE

The APP_RAM_BASE is different for each SoftDevice being used. This page provides the documentation for each SoftDevice where this information can be retrieved. The table below gives this information in one table.

SoftDevice	SDS Version	APP_RAM_BASE = Start + ATTR_TAB_SIZE	Stack Size	Heap Size	APP_CODE_BASE

		Start	ATTR_TAB_SIZE (default)	**Base**	max (bytes)	max (bytes)	
S110	v2.0	0x20001900	0x700	**0x20002000**	0x600	0	**0x00018000**
S120	v2.1	0x20002800	–	**0x20002800**	0x600	0	**0x0001D000**
S130	v1.0	0x20002200	0x600	**0x20002800**	0x600	0	**0x0001C000**
	v2.0	0x200013C8	–	**0x200013C8**	0x600	0	**0x0001B000**
S210	v3.0	0x20000900	–	**0x20000900**	0x400	0	**0x0000D000**
S310	v3.0	0x200022002	–	**0x200022002**	0x800	0	**0x0001D000**

Defining Stack and Heap size

Stack and heap size is depended on the application and thus it is something you need to figure out yourselves. One important remark:

> The application call stack memory usage **must be added** for the total call stack size to be set in the user application.

Putting everything together in the linker...

Now we have every information needed in order to configure correctly our linker! Inside our Makefile we call the linker script `-Tble_app_gcc_nrf51.ld` . This linker script has the configuration for our program memory and RAM. This is the file we need to adapt to our setup. The file looks like the following:

```
/* Linker script to configure memory regions. */

SEARCH_DIR(.)
GROUP(-lgcc -lc -lnosys)

MEMORY
{
  FLASH (rx) : ORIGIN = 0x1c000, LENGTH = 0x24000 /* 256kB Flash / S130 v1.0 SD. 0x40000-0x
  RAM (rwx) :  ORIGIN = 0x20002800, LENGTH = 0x1800 /* 16kB RAM / S130 v1.0 SD. 0x4000-0x28
}

INCLUDE "gcc_nrf51_common.ld"
```

*The above is an example where the nordic nRF51x chip has **16kB of RAM** and uses the **S130 SoftDevice.** ☺*

Configure Stack and Heap size

To configure stack and heap size edit the `gcc_startup_nrf51.s` file. Look at the start of the file:

gcc_startup_nrf51.s

```
#ifdef __STACK_SIZE
    .equ    Stack_Size, __STACK_SIZE
#else
    .equ    Stack_Size, 2048
#endif
    .globl    __StackTop
    .globl    __StackLimit
__StackLimit:
    .space    Stack_Size
    .size __StackLimit, . - __StackLimit
__StackTop:
```

```
    .size __StackTop, . - __StackTop

    .section .heap
    .align 3
#ifdef __HEAP_SIZE
    .equ    Heap_Size, __HEAP_SIZE
#else
    .equ    Heap_Size, 512
#endif
    .globl    __HeapBase
    .globl    __HeapLimit
```

Of course you can also define `__STACK_SIZE` and `__HEAP_SIZE` .

THE METAWEAR RG HAS **16 KB RAM!**

Step 2: Configuring the Clock

If you are using and external crystal as your clock source you need to perform the following:

1. Change the definition at the top of the `system.nrf51.c` :

   `#define __SYSTEM_CLOCK (16000000UL) /* 16MHz Crystal */`

   or

   `#define __SYSTEM_CLOCK (32000000UL) /* 32MHz Crystal */`

2. Enable it explicitly with the following code:

```c
int main(void)
{
    // Set the external high frequency clock source to 32 MHz
    NRF_CLOCK->XTALFREQ = 0xFFFFFF00; // 32 MHz (CLOCK_XTALFREQ_XTALFREQ_32MHz) OR 0xF

    // Start the external high frequency crystal
    NRF_CLOCK->EVENTS_HFCLKSTARTED = 0;
    NRF_CLOCK->TASKS_HFCLKSTART = 1;

    // Wait for the external oscillator to start up
    while (NRF_CLOCK->EVENTS_HFCLKSTARTED == 0) {}

    while (true) { }
}
```

THE METAWEAR RG USES A **32 MHZ EXTERNAL CRYSTAL!**

Sources

- Nordic Online Documentation
- Use external 32MHz Clock
- Is there a formula to calculate attribute table size?
- MetaWear Custom Firmware
- GCC linker script and STM32 (a tutorial)

#BLE #GCC #METAWEAR #NORDIC #NRF51

You might also like

0 Comments TurluCode

Login

Recommend Share Sort by Best

Start the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Be the first to comment.

ALSO ON TURLUCODE

Autostart and StaY!

15 comments • 5 months ago

Marcelo Franco — Hi, hope this finds you well. I need to start an app when connected to 2 specific wifi networks. The ON WIFI

QEMU-KVM Installing Windows 10 Client

2 comments • 2 years ago

athanasios! — Thanks for mentioning it! It was on the previous step: http://turlucode.com/qemu-k.... I will put it here

Cygwin customization tips

2 comments • 7 months ago

AthanasiosKarapatis — Thomas LECAT Hey Thomas. Sorry for the late reply. You probably edited the file with windows notepad. Try to

Metwear Custom BLE Firmware

2 comments • a year ago

athanasios! — I am using the SPI that is connected to the BMI160 together with the nordic's SDK 9 without issues. What is your