Application note

Ultralow power system design guidelines and STEVAL-ASTRA1B power management characterization

Introduction

Energy saving in electronic systems is becoming increasingly important, including in the development of IoT applications, which highlight the importance of reducing energy consumption to achieve battery autonomy up to 15-20 years, without compromising functionality. It is also important to consider that many IoT systems must remain in on-the-shelf mode for long periods of time before they are activated.

While adding more features to the board expands the capabilities of a system, all the functionalities may not need to be active at the same time. If we consider multisensor and multiconnectivity IoT tracking systems, you may only need short range communication when a concentrator is nearby to collect the data, and long range when the device is moving. Also for a GPS receiver, you only need to track the device while it is moving, and you can disable the GPS when the device is still. This behavior can be achieved by creating different power domains that can be activated separately or by commands to lower the power consumption of a specific device. Similarly, most sensors have several configurations offering different levels of power consumption that allow balancing performance requirements and energy efficiency.

The implementation of a low power solution starts from the choice and availability of low/ultralow power components, combined with appropriate architectural design that takes component interactions into account.

This application note analyzes the design requirements, system architecture, power management schemes, and firmware functionalities to maximize the battery duration, focusing on specific features available on the STEVAL-ASTRA1B multiconnectivity asset tracking node reference design.

The aim is to provide design rules to optimize the power consumption in these kinds of systems and a strategy to analyze the interaction between the various components. We will start with very simple examples before moving on to more complex application board designs like the STEVAL-ASTRA1B, which offers an example of complex architecture that uses several power domains to switch-off specific sub-systems when the user application is not using them.

In simple applications, as the main contributor to power consumption is the MCU, current consumption is easily reduced by configuring the MCU to a low power state. In more complex systems, current consumption optimization can involve several components, and their interaction must be taken into account to avoid unwanted events, such as spurious signals in high impedance or floating lines being interpreted as operational commands by connected devices.

We will discuss some low power expedients to be applied in hardware design (e.g., the power domain, GPIO configuration to reduce current consumption, peripheral selection, and choosing the correct pullup/pulldown on the lines). We will also discuss how firmware architecture should be developed to exploit these features in run and low-power modes.

Firmware considerations

The following scenarios provide examples of how firmware can contribute to reducing power consumption:

- Place the MCU in power saving mode if it is waiting for an event that can wake it up when it occurs
- Configure sensors to only notify the MCU when it detects a significant change on the measured entity
- When using a communication interface, limit the data transfer cost by only sending data when needed.

During application firmware development, the expected tasks should be considered, and when in the final evaluation of battery duration, the firmware should be reviewed to remove unnecessary “CPU cycles” spent waiting for certain events or sending and collecting data that is not useful.

The following general rules help minimize active CPU time:

- When using an operating system, put the MCU in low-power mode when an idle task is triggered.
- In bare metal applications (without OS), apply event-driven techniques. When an interrupt/wake up event occurs and the MCU is in low-power mode, set a variable to render the system aware that there is something to do and process the “command” in the main loop. This allows correct wake up event handling by adding the required instructions, such as clock reconfiguration or device initialization.

1 Shelf mode implementation

Shelf mode is the time between when an electronic device is produced and when it starts being used. As high-volume consumer devices are often distributed globally, this period may be in the order of months or even years.

There are two ways of avoiding battery drainage while a product is on the shelf:

1. Physical disconnection of the battery (e.g., a physical switch or plastic tab between the battery and the electrical contact)
2. Maintain battery connection, with the system placed in its lowest power mode.

If physical battery disconnection is used:

- The system does not consume energy during shelf mode
- The system is only activated when the user removes the plastic isolation or switches the on button
- The system cannot shut down autonomously, which means it always remains powered until the battery is completely drained

If the device is sealed or there is no way of accessing the battery terminals, the only option is to directly connect the battery to the circuit and put the device in the lowest power consumption mode during shelf time.

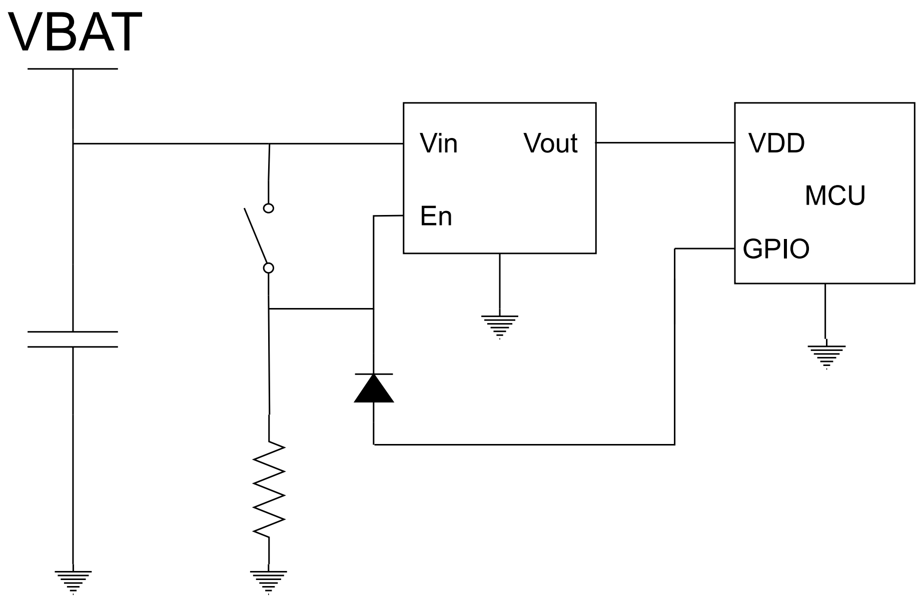
If battery connection must be maintained:

- The system consumes a certain amount of energy during shelf mode, and the overall consumption may become significant if the device remains on the shelf for years
- It is activated by a specific user action
- The system cannot shutdown autonomously

1.1 Basic shelf mode: power-on pin latched control with GPIO

A simple shelf mode implementation can be achieved with an MCU GPIO connected to the voltage regulator enable pin:

Figure 1. Simple shelf mode implementation



This implementation benefits applications that require long periods of storage or where it is necessary to turn off the circuit without disconnecting the battery. This technique allows zero absorption downstream of the voltage regulator in the off condition but requires a static current drain on the EN pin. This adds minor consumption in operating conditions to keep the regulator active, which must be considered in the overall energy budget calculation.

To turn the system ON, the switch is temporarily closed. It enables the regulator that powers the MCU, which in turn brings the GPIO to a high logic value to keep the regulator active, even if the switch is opened.

The advantage of this configuration is the possibility for the MCU to turn itself, and therefore the system, off. In RUN mode, the MCU only needs to put the GPIO at the low logic level to turn the system off. This disables the regulator by cutting off power downstream of the regulator, including the MCU. The system remains in this condition until the switch is closed again.

The drawback is that the GPIO cannot address low or high impedance configurations, and even in standby mode, the pin must be pulled high; e.g., using a GPIO internal pull-up.

2 Power management circuit architecture as MCU architecture

The power management architecture of a system can be considered the internal organization of an MCU. A general purpose MCU normally consists of a CPU, flash, and RAM, plus a set of peripherals like GPIOs, timers, ADC, DAC, crypto, and communication.

At startup, the MCU is configured to turn on only what is needed for correct operation; e.g., a GPIO and a timer to implement a PWM signal, or an SPI to read and process sensor data. All other peripherals remain shut down. When processing is not needed, the core is also stopped. Moreover, some MCU features can offload simple tasks from the core and delegate them. For example, using the DMA to copy data from one peripheral to another by collecting data from an external entity in the MCU RAM for transmission or further analysis.

The same approach will be applied for board power management design in this document. Turning on only the functions that are required means that the design must include switches to shut down portions of the circuit that are not needed in certain conditions, and de-initializing the related MCU peripherals. The MCU can also be put in low power mode when the system is idle.

Before entering in this condition, some functions can also be delegated to external entities. For example, some sensors have an internal FIFO memory, so the data can be collected in the sensor and transferred to the MCU in single “bursts”. Moreover, some advanced features can decrease the needed CPU time in processing: algorithms running inside the sensor can preprocess sensor data without MCU intervention and send alerts to the MCU when the data is ready.

2.1 MCU low-power states

When designing the main program of a firmware application, the MCU behavior should be a primary consideration. For example, a bare metal (no operating system) implementation with initial clock and hardware initialization and main loop (implemented with a while(1)) suggests that all the CPU time is used to perform the application tasks. Power consumption can be reduced by disabling some parts of the MCU whenever it is idle, even for very short periods.

These considerations are implemented by the different MCU power states provided by the MCU chip.

This application note will not cover all the low-power strategies implemented in all the STM32 series (like stop2, stop3, RAM retention, etc.), which are described in the specific MCU datasheet, but on the major directives to implement low-power applications.

The following power states are generally available on STM32 series MCUs:

- Run mode:
  - The core and all peripherals are ON, accessible and can generate interrupts, and all clocks are configurable, with the possibility of extracting maximum performance from the MCU.
- Sleep mode:
  - The CPU clock is OFF and there is no effect on other clocks or analog clock sources.
  - All peripherals continue to operate and can wake up the CPU when an interrupt/event occurs.
- Stop mode:
  - Offers the lowest power consumption while retaining the SRAM and register content.
  - All clocks in the VCORE domain are stopped, and the PLL, MSI RC, HSI16 RC, and HSE crystal oscillators are disabled.
  - The LSE or LSI can be kept running.
- Standby mode:
  - Used to achieve the lowest power consumption with brown-out reset.
  - The internal regulator is switched off so that the VCORE domain is powered off.
  - The PLL, MSI RC, HSI16 RC, and HSE crystal oscillators are also switched off.
  - RTC can remain active (Standby mode with RTC, Standby mode without RTC).
  - The state of each I/O during standby mode can be selected by software: I/O with internal pull-up, internal pull-down or floating.
  - The system can be woken up from standby mode using a SYS_WKUP pin, an RTC event (alarm or timer), IWDG, or an external reset in NRST pin.
  - After waking up from Standby mode, program execution restarts in the same way as after a Reset (boot pin sampling, option bytes loading, reset vector is fetched, etc.).

2.1.1 Typical use of low-power states with duty cycle

A typical low-power application involves the following states.

- Full run
- Low power run
- Low power
- Ultralow power

The system behavior often has a high impact on the power consumption. For simple systems where the majority of power consumption can be attributed to the MCU, it is important to identify and characterize the various conditions in which the system will operate in order to estimate the best MCU power mode of the MCU for each system condition.

- Full run: for highly reactive systems like safety applications or driving unmanned vehicles, the MCU is never idle, and constantly monitors system parameters, fault signals, and user inputs, and each action must be executed as quickly as possible. The MCU is in run mode in this phase (core, peripherals, GPIOs, and interrupts are running).
- Low power run: activity monitoring systems like fitness bands can act in different ways based on the activity they are monitoring. The MCU acquires data from sensors, remains idle when there is nothing else to do, and enters full run mode when executing algorithms. If the acquisition phase is fast, the MCU can also scale the CPU clock to save power. The selected low-power mode in this case is typically in sleep mode (core stopped and peripherals running, GPIOs and interrupts available). The MCU resumes its last state when exiting sleep mode.
- Low power: if the system is idle for a long time, the application can be completely stopped, the device could continue to drive some specific output and it is waiting for a trigger coming from a sensor or a signal. “Sleep if you can”. The selected low-power mode in this case is typically in stop mode (core and peripherals stopped, GPIOs and interrupts available). The MCU will resume where it left after stop mode.
- Ultralow power: if the system is idle for an extended period and there is no output to drive, then the system can be placed in the lowest power consumption mode. The MCU in this phase is typically in standby mode (core and peripherals stopped, GPIOs and interrupts disabled). Only wakeup sources are available, and the MCU starts from an initial state after standby mode.

The following sections present some implementations of the given examples. The simulations were performed with the STM32CubeMX tool on an STM32L0 MCU using the Li-SOCL2 AAA700 battery (default proposal of the STM32CubeMX tool). Its capacity is 700 mAh and nominal voltage is 3.6 V.

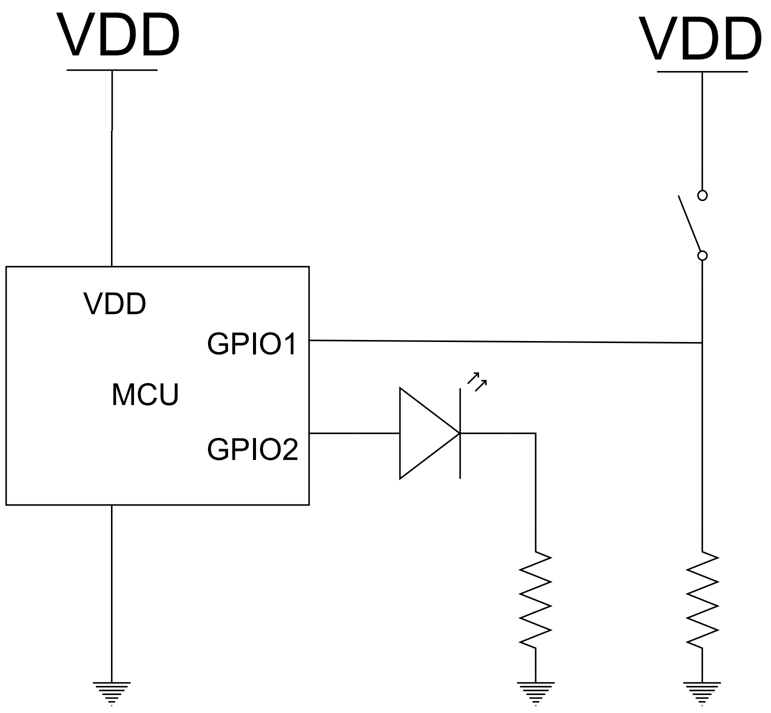
2.1.2 MCU standalone: simple board example

The following use cases will be discussed:

1. Smart timer
2. Temperature measurement with internal MCU temperature sensor

In the first use case, a very simple application will be analyzed: a smart timer implemented via MCU using a button and an LED. Here we will focus on the MCU behavior and configuration. The MCU supply voltage (VDD) will be considered as the only supply for the application and so the current consumption is measured on the VDD rail.

Figure 2. **Simple circuit with MCU, LED, and button**



We assume the following behavior:

1. System waiting for the user first input
2. After user input, initialize RTC and GPIO
3. Turn on the LED for a certain time
4. Turn off the LED
5. Wait for the next user input

In (1), the system is in standby

In (2), the system is in run mode (high speed) to quickly execute configurations

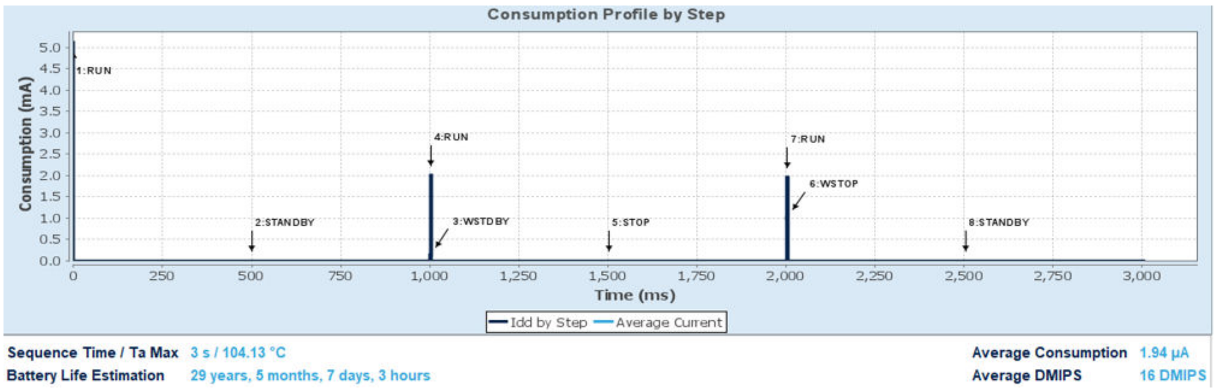
In (3), the system is in stop mode (with RTC alarm) to drive the LED GPIO high

In (4), the system is in run mode to quickly execute shutdown operations

In (5), the system is in standby

Whenever possible, standby mode is used because the system only needs to wait for a trigger from the button. Stop mode is only used in (2) to ensure the ability to drive the GPIO. In this phase, the RTC alarm is used to wake up the system after the requested interval on the timer.

Figure 3. **Current consumption simulations using STM32CubeMX for smart timer**



Sequence Time / Ta Max 3 s / 104.13 °C

Battery Life Estimation 29 years, 5 months, 7 days, 3 hours

Average Consumption 1.94 µA

Average DMIPS 16 DMIPS

Note: Refer to Cube examples: PWR_STANDBY, PWR_EnterStopMode, and GPIO_IOToggle

In the second use case, the same hardware as the previous example can be used to implement temperature sensing using the internal MCU sensor.

We assume the following behavior:

1. System waits for the user first input
2. After user input, initialize GPIO and internal temperature sensor
3. Start temperature monitoring
4. During temperature measurement, turn on the LED if temperature is high
5. Turn off the led, shut down the temperature sensor, and wait for the next user input

In (1), the system is in standby

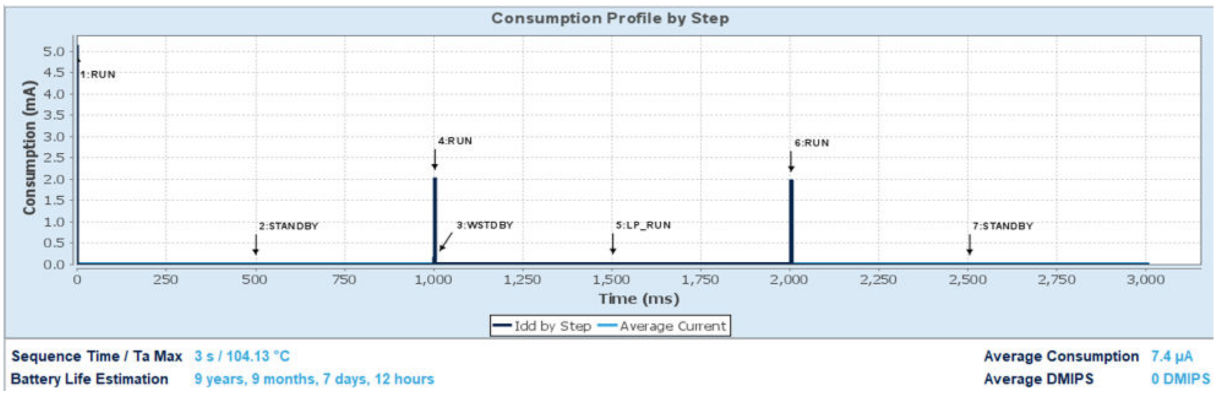
In (2), the system is in run mode (high speed) to quickly execute startup configurations

In (3) and (4), the system is in low-power run mode (clock scaling) to acquire temperature sensor data using DMA and drive the LED

In (5), the system is in standby mode

Whenever possible, standby mode is used because the system only needs to wait for a trigger from the button. In (2) and (3), run mode and sleep mode are used to keep peripherals and DMA running and the ability to drive the GPIO. In this phase, the RTC alarm is used to notify the system after the requested timer interval.

Figure 4. **Current consumption simulations using STM32CubeMX for internal MCU temperature sensor**



Sequence Time / Ta Max 3 s / 104.13 °C
Battery Life Estimation 9 years, 9 months, 7 days, 12 hours
Average Consumption 7.4 µA
Average DMIPS 0 DMIPS

Note: Refer to Cube examples: PWR_STANDBY, GPIO_IOToggle, PWR_LPRUN, and ADC_SingleConversion_TriggerSW_Init

2.1.3 MCU and external components: more complex board example

Starting from the previous cases, if we increase complexity by adding, for example, an external sensor, we need to also analyze the interaction of an external component and its behavior in the whole application.

Here, we will focus on power supply tips that allow better management of the sensor power consumption. Further considerations will be detailed in Low power implementation using the STEVAL-ASTRA1B.

The MCU VDD is considered as the only supply for the application and so the current consumption is measured on the VDD rail. The external temperature sensor is supplied by a GPIO to cut the power when no temperature measurement is requested. For the I²C peripheral, as described in STEVAL-ASTRA1B implementation example for peripheral configurations, the pullup resistors must be connected to the same supply of the sensor.

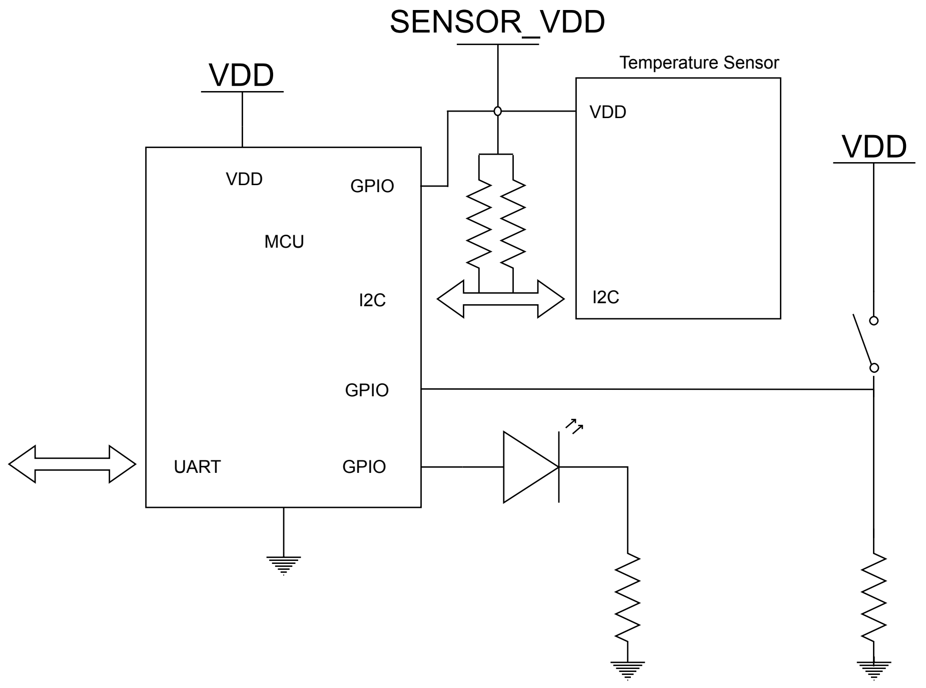


Figure 5. **Simple circuit with MCU, an LED, a button and a temperature sensor connected via I²C bus**

We assume the following behavior:

1. System waits for the user first input
2. After user input, power up the sensor by driving the GPIO high, initialize the peripheral and configure the temperature sensor via I²C
3. Start temperature monitoring at a certain given frequency
4. During temperature measurement, turn on the LED if temperature is high
5. Turn off the led, shutdown temperature sensor
6. Wait for the next user input

In (1), the system is in standby

In (2), the system is in run mode (high speed) to quickly execute startup configurations

In (3), the system is in run mode with clock scaling (and sleep if using DMA) to acquire temperature sensor data and drive the LED

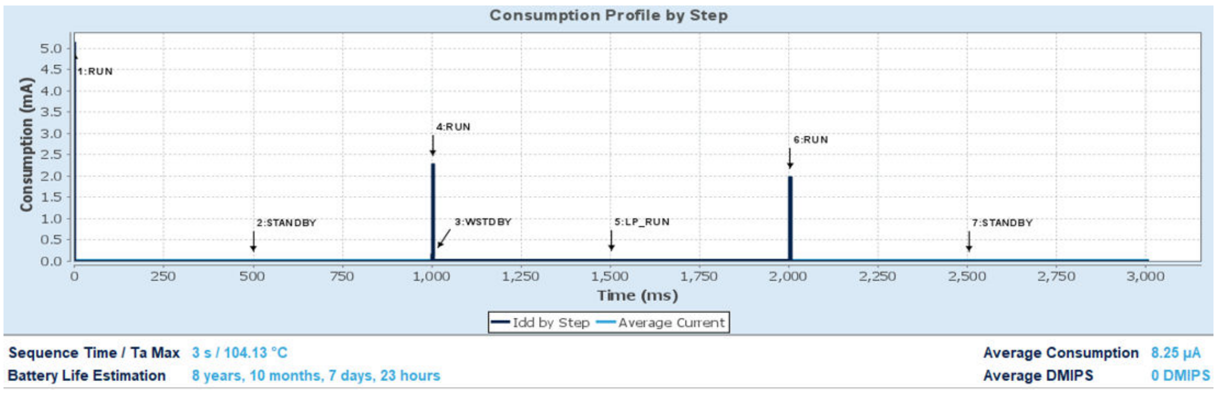
In (4), the system is in STOP between two measurements

In (5), system is in run mode (high speed) to quickly execute deinitialization operations: GPIOs in high impedance to avoid leakages, I²C peripheral deinitialized

In (6), the system is in standby

Whenever possible, standby mode is used because the system only needs to wait for a trigger from the button. In (2) and (3), run mode and sleep mode are used to keep the peripherals and DMA running and the ability to drive the GPIO. In (4), stop mode is used to keep the GPIO high for the sensor supply. In (5), the system is in run mode to speed up the shutdown operations. In the whole phase, the RTC alarm is used to notify the system after the requested timer interval.

Figure 6. **Current consumption simulations using STM32CubeMX for MCU and external sensor**



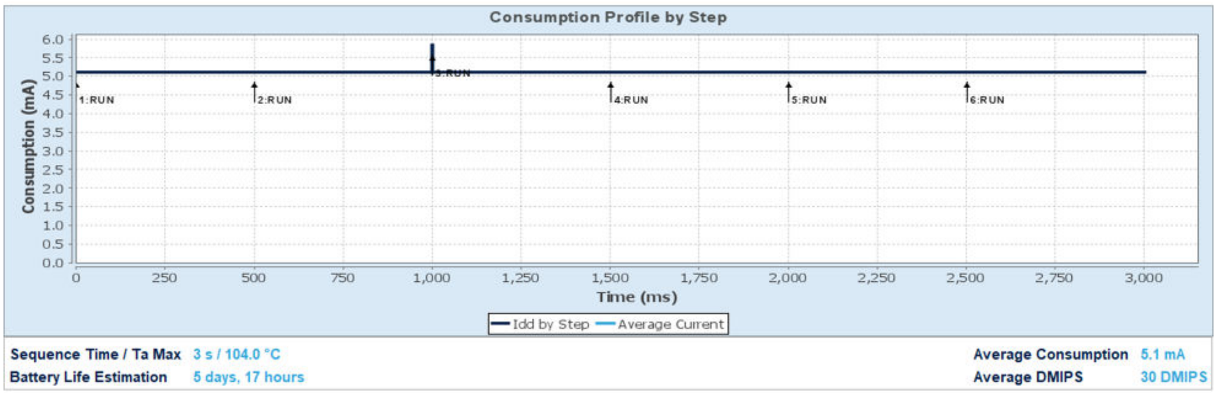
Sequence Time / Ta Max	3 s / 104.13 °C	**Average Consumption**	8.25 µA
Battery Life Estimation	8 years, 10 months, 7 days, 23 hours	**Average DMIPS**	0 DMIPS

Note: *Refer to Cube examples: PWR_STANDBY, GPIO_IOToggle, PWR_LPRUN, and I2C_OneBoard_Communication_PollingAndIT*

2.2 Low power vs non-low power implementation

In the previous sections, the low power MCU implementation offers a forecast of approximately 9-year to 30-year battery lifetime. This value must be corrected by considering the factors that affect real system use, such as battery self-discharge or the energy needed to drive external components. If we only consider the MCU energy consumption, however, we can compare the previous autonomy data with the same implementation without low power. The result is an incredible 8 years compared to just 5 days.

Figure 7. **Current consumption simulations using STM32CubeMX for MCU in run mode**



Sequence Time / Ta Max	3 s / 104.0 °C	**Average Consumption**	5.1 mA
Battery Life Estimation	5 days, 17 hours	**Average DMIPS**	30 DMIPS

3 Low power implementation using the STEVAL-ASTRA1B

This section will present an example of firmware library designed to manage the low power modes on the entire board. The power modes are managed through a state machine that is described in the next section. There will be focus on the low-level configurations needed to ensure the correct management of the hardware in terms of schematic design and firmware configuration.

For the STEVAL-ASTRA1B, all these firmware features are implemented in the FP-ATR-ASTRA1 function pack.

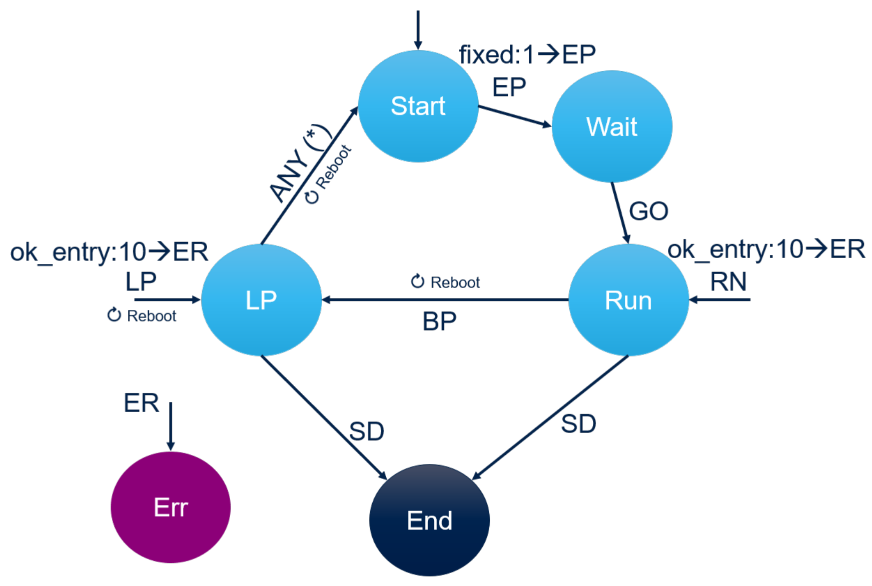
3.1 Firmware state machine

To deal with the complex mechanisms behind this implementation and to fulfil the objectives of modularity and flexibility of the library, a state machine is implemented to simplify the configuration of the various modules in different states. Here we will focus on RUN and LOW POWER states.

With reference to the firmware implementation, see APP_ST_RUN and APP_ST_LP definitions in SM_App.c file of FP-ATR-ASTRA1.

For further information on the state machine, see Section 2.2 of UM3019.

Figure 8. State machine flow



(*) Wake up sources are configured by application layer

3.2 Low level configurations

This section presents tips for use in complex applications and implementation examples on the STEVAL-ASTRA1B and its firmware source code.

3.2.1 GPIO configurations

The GPIO configuration has an impact on power consumption in low power. The following common cases will be analyzed further:

- GPIO with external/internal pullup/pulldown
- analog inputs
- GPIO output

If the GPIO is not connected to external components (e.g., not used or used for test/debug pins), then the most suitable mode to achieve the lowest power consumption is the analog input mode. This is to avoid the Schmidt trigger toggling that might occur with a floating input, which causes undesired power consumption.

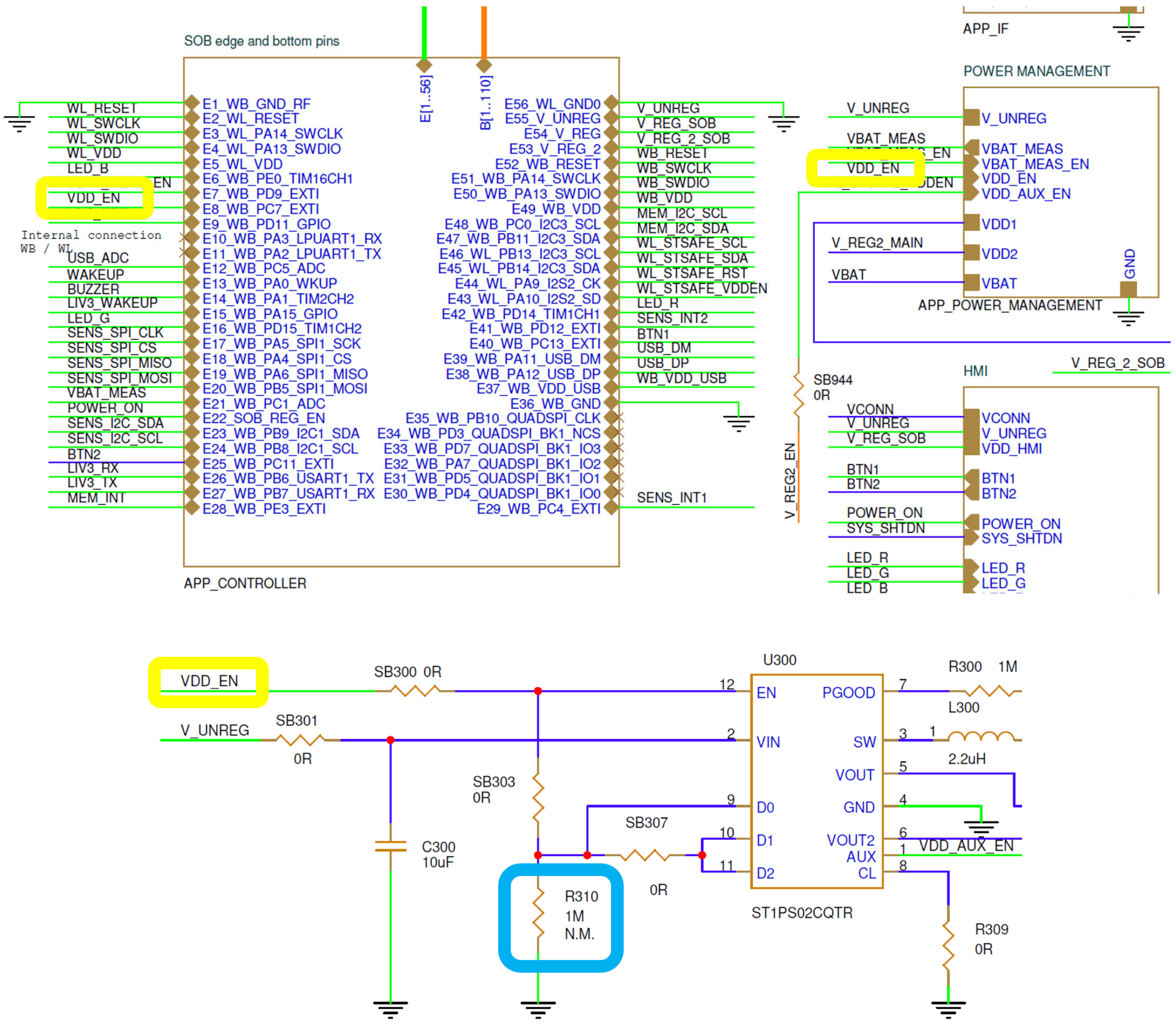
When the GPIO is used as the output to drive an external component, the line must be tied to the correct logic level by means of a pullup or pulldown resistor to place the external component in the correct condition and allow the GPIO to be placed in analog input mode.

The following sections present STEVAL-ASTRA1B implementation examples for appropriate GPIO modes and whether external pullups or pulldowns are required.

3.2.1.1 *External pullup/pulldown*

If the pin level is correct during low power mode, then no pull is required. For example, VDD_EN - E8_WB_PC7_EXTI is the enable input for a DC-DC converter (ST1PS02 AB0038 U300.12) and must be LOW in low power mode to ensure the regulator remains off. No pulldown has been added to avoid leakage, and the pin is tied down by firmware. When left floating, the pin goes low (verified in characterization). R310 (pulldown) can be mounted if needed. The VDD_EN - E8_WB_PC7_EXTI pin connections are shown in the following figure.

Figure 9. External pullup/pulldown VDD_EN



If a pin must be set to HIGH during low power, then a pull-up is required. For example, SENS_SPI_CS is the CS input of an SPI driven accelerometer (LIS2DTW12TR AB0038 U200.2). It is tied to VDD through R200 pullup to avoid unwanted floating signals to the CS input of the LIS2DTW12TR. The 10 KΩ value causes a leakage when driving this pin low; this is acceptable because the pin is used only during configuration, read, and write. The SENS_SPI_CS pin connections are shown in the following figure.

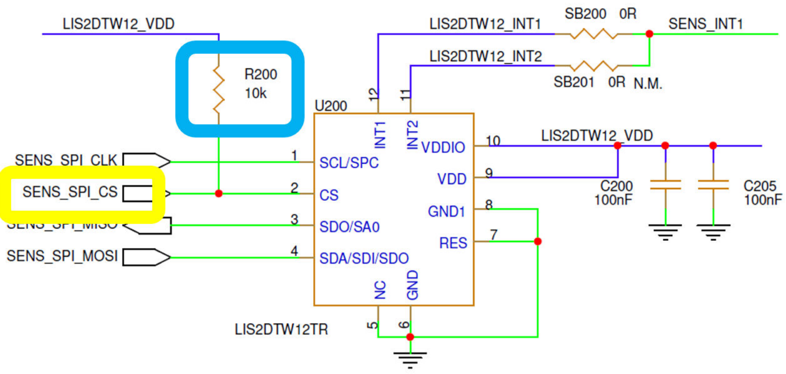


Figure 10. **External pullup/pulldown SENS_SPI_CS**

In certain cases, a pin must be LOW during low power to avoid unwanted enabling of the connected circuitry. For example, VOUT2_CTRL is the aux input to drive the VOUT2 of a DC-DC converter (ST1PS02 AB0037 U900.1). The AB0037 R930 resistor is used to pull down VOUT2_CTRL to avoid even partial activation of the V_REG_OUT_2 power rail and consequent leakage in low power. The VOUT2_CTRL pin connections are shown in the following figure.

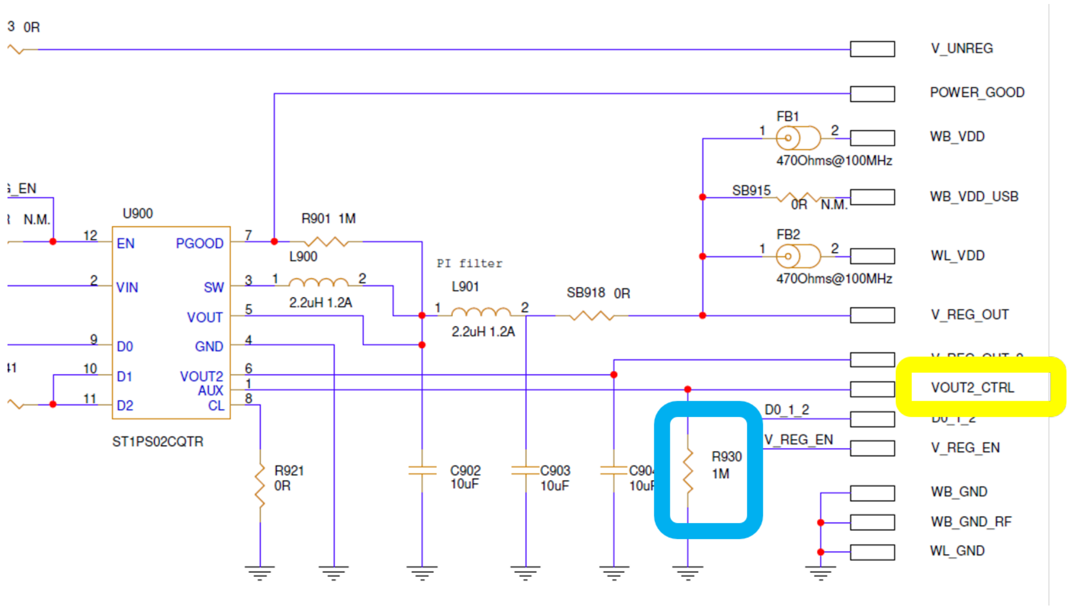
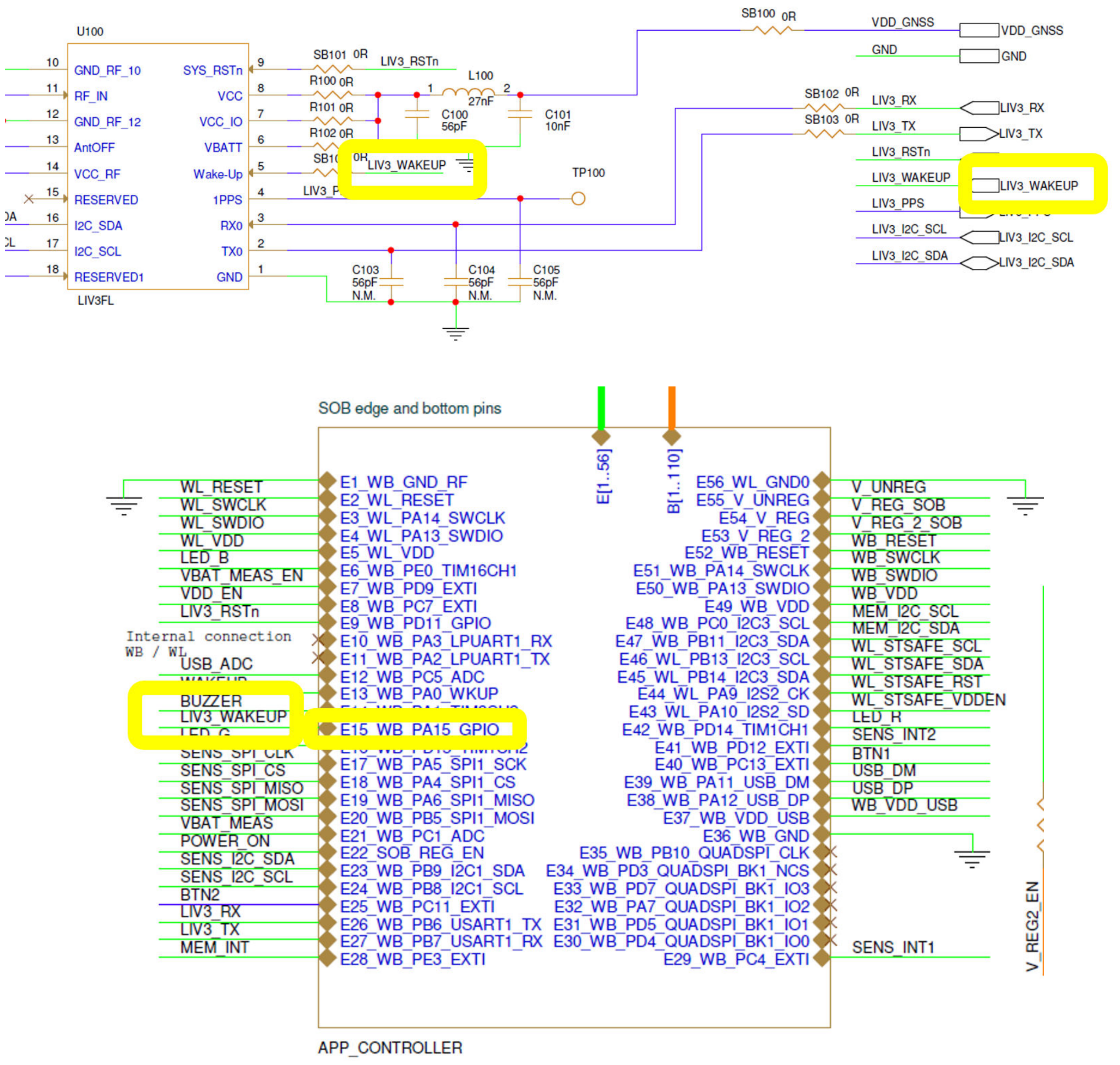


Figure 11. **External pullup/pulldown VOUT2_CTRL**

3.2.1.2 *Analog inputs*

SWO pin (PA15) is the WAKEUP input for LIV3F GPS receiver (AB0038 U100.5). During low-power mode, U100 is not supplied, and a current will flow through its pin 5 if the PA15 is driven high. It is configured in GPIO_MODE_ANALOG.

Figure 12. **Example of analog input implementations**



The following code refers to astra_boot.c in the FP-ATR-ASTRA1 v2.0.0:

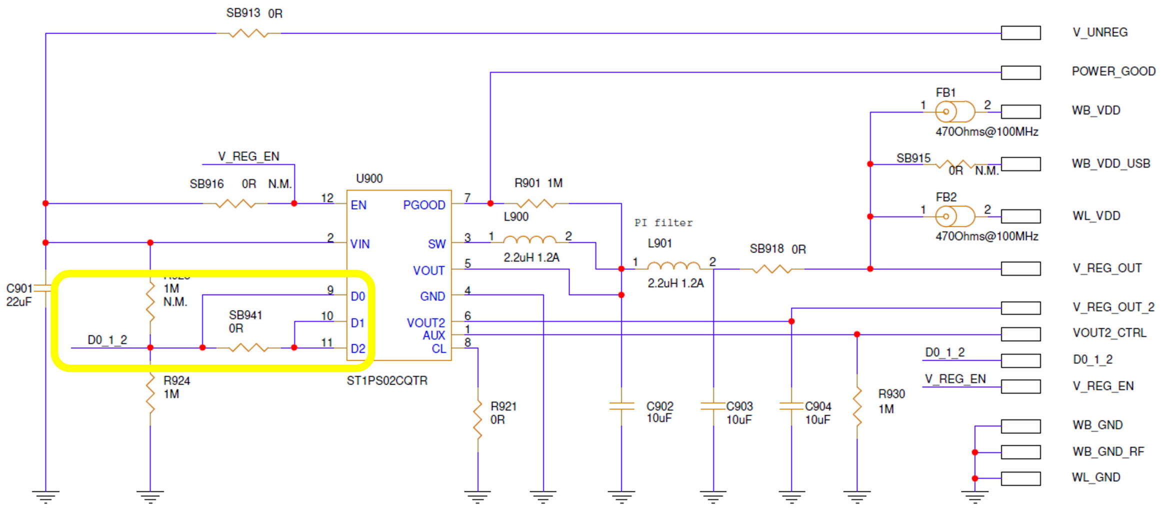
```
        /* Deinit SWO pin to minimize current leakage */
        GPIO_InitTypeDef GPIO_InitStruct = {0};
        GPIO_InitStruct.Pin = GPIO_PIN_15;
        GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
        GPIO_InitStruct.Pull = GPIO_NOPULL;
        GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
        HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
```

3.2.1.3 *GPIO output*

The VOLTAGE_SEL pin is connected to the DC-DC converter (ST1PS02 AB0037 U900.9 U900.10 U900.11). These pins can be driven high even in low power mode to obtain 3.3 V at the U900.5 VOUT pin for better system RF performance. In stop mode, it is possible to drive this pin high by placing it in Output mode, as shown in the following function.

custom_sob_power_init() → configuration GPIO_MODE_OUTPUT_PP for VOLTAGE_SEL_PIN with the appropriate level

Figure 13. GPIO output implementation example



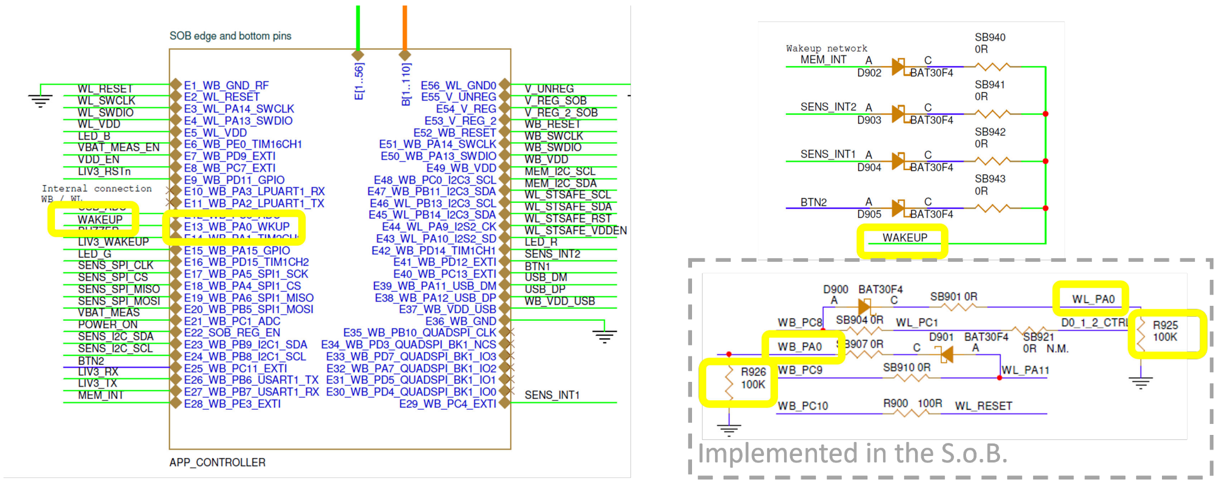
3.2.2 Interrupts and wakeup sources from standby

Interrupt sources are used to trigger a particular operation in the MCU when a rising or falling edge is detected on the GPIO. This gives the flexibility to choose the polarity of the interrupt signal and the GPIO pin number that will use the related interrupt line. If the pin must also be used to wake up the system from standby state, then a wakeup pin must be selected. You can select the most suitable pin for your application from the various wakeup pins available on your MCU, based on the number of wakeup sources required and the associated function (e.g., simple wakeup or tamper detection pin).

3.2.2.1 *STEVAL-ASTRA1B implementation example for wakeup sources*

AB0038 E13_WB_PA0_WKUP is a wakeup pin. Every possible wakeup source is connected to this pin through a diode; e.g., U200 INT1 pin is connected to the wakeup network through D904 or BTN2 through D905.

Figure 14. Interrupts and wakeup implementation example



3.2.3 Peripheral configuration (and related GPIO) in run and low power

GPIOs that are used as alternative functions in an application must be correctly conditioned externally because they will act as floating inputs in the lowest power consumption mode. According to the function that the pin has with respect to the external component, the pin must therefore be driven correctly in order for the external component to remain in the desired condition. For example, I2C pins must remain at a high level and the same for UART pins; the SPI pins must also be analyzed on the device side to choose the right configuration. Moreover, the pullup resistors of a certain communication bus must be connected to the same supply of the ICs that use that bus.

3.2.3.1 STEVAL-ASTRA1B implementation example for peripheral configurations

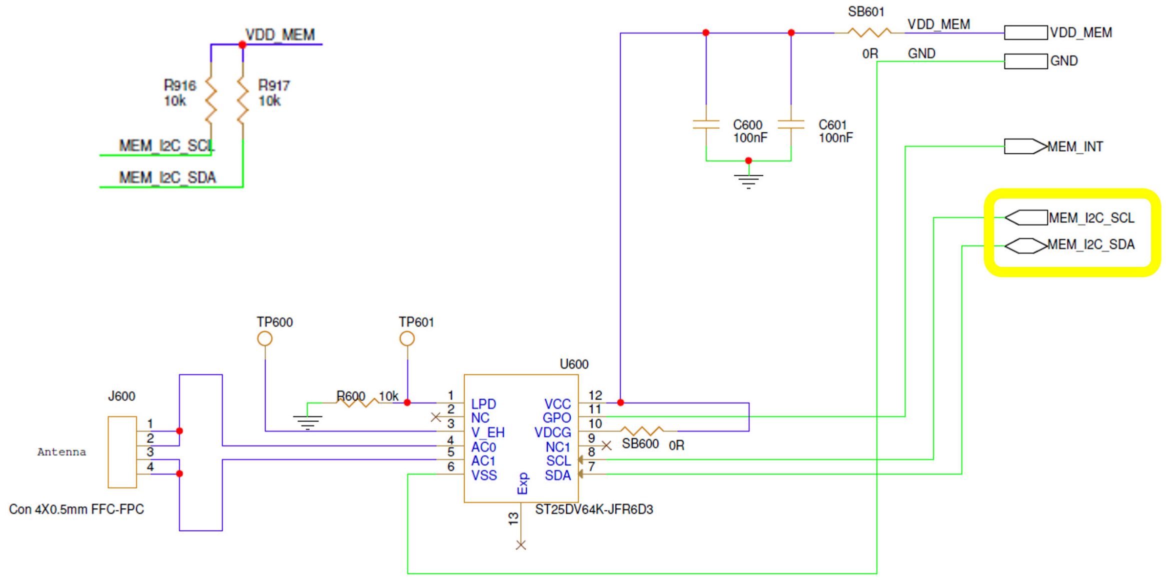


Figure 15. **Example of peripheral configuration implementations**

Note: The pullup resistors R916 and R917 are connected to VDD_MEM, the dedicated supply voltage for U600

When instead operating in low power mode, the AstraPrepareHwLp() is called. The memory is used for a small amount of time and then the I²C peripheral is de-initialized.

memory_init() → BSP_I2C3_Init() → I2C3 pins are configured in AF mode

/* access to the memory*/

memory_deinit() → BSP_I2C3_DeInit () → HAL_GPIO_DeInit called for each I2C3 pin

3.2.4 GPIO port clocking in low power

Generally, the number of ports used in the application should be minimized. This is usually not a simple task because the needed peripherals and/or functions may not be available on the same port. Moreover, when dealing with miniaturization, the designer must also consider some layout constraints to optimize the routing of the PCB or reduce costs if connections to certain pins require additional layers on the PCB, laser vias, etc.

3.2.4.1 STEVAL-ASTRA1B implementation example for GPIO port clocking

Due to the complexity of the STEVAL-ASTRA1B board, the previous indication does not need to be strictly considered: the need for many peripherals distributed on several ports requires the use of almost all GPIO ports.

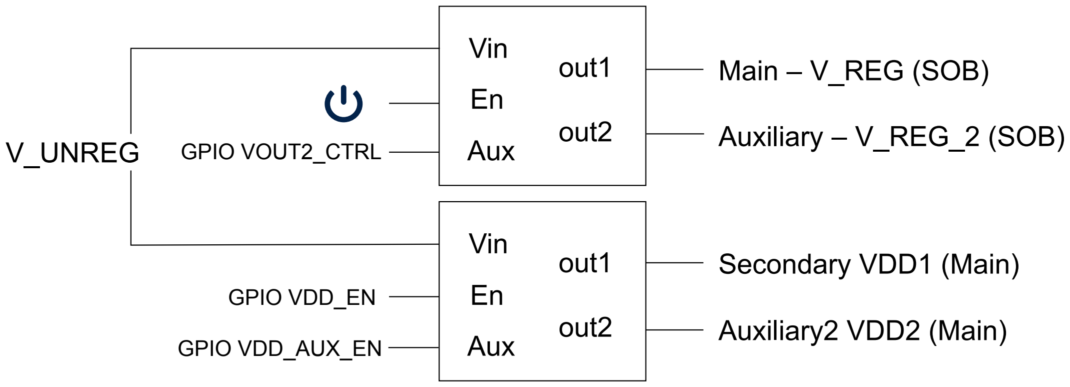
4 Power domain designs

4.1 System supply voltage and power domain partitioning

As well known, the supply voltage impacts on the energy absorption of a system and its consumption. In a circuit, at equal condition, lowering the supply voltage means consuming less. This approach is quite easily implemented in low-power systems where the supply voltage range of components that make up the board are more likely to overlap.

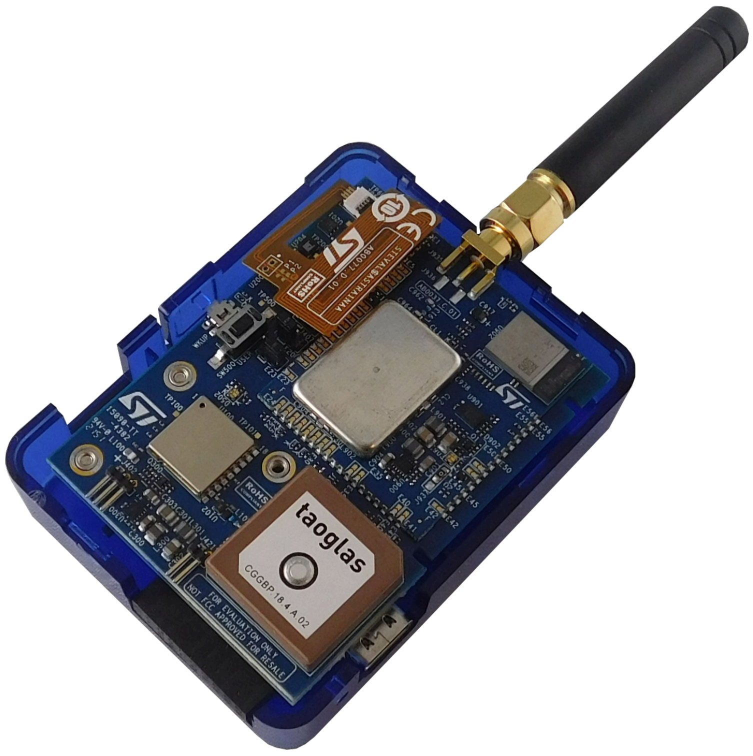
Therefore, in a complex solution with multiple subparts, the power management circuit can be partitioned into multiple power domains (PD), and each of them can be power gated individually to achieve fine-grain system-level power management. During operation of the systems, if none of the hardware components belonging to the same PD is needed to execute any tasks, the whole PD can be powered off to save static power. When a new task has to be executed on any of these components, this whole PD is powered on again. Usually hardware components that are power hungry (like GNSS) have its own PD, or components that are functionally affine use the same.

Figure 16. **STEVAL-ASTRA1B power domains schematic**



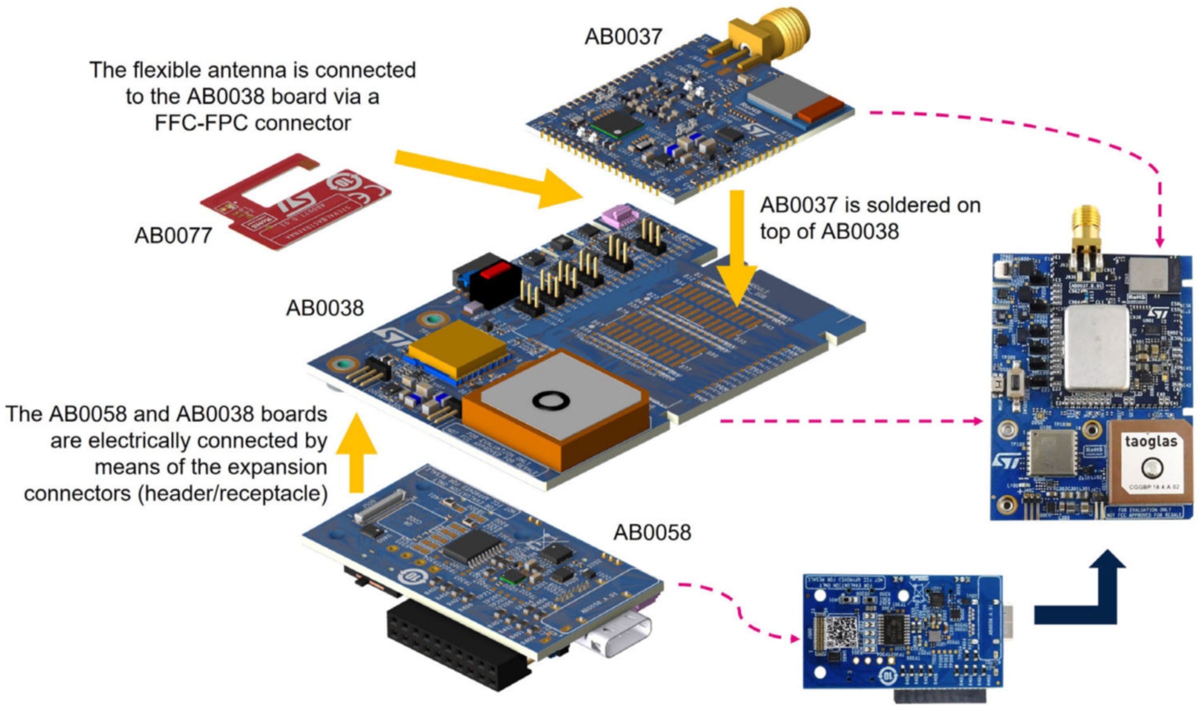
The ASTRA platform (STEVAL-ASTRA1B) is a development kit and reference design that simplifies prototyping, testing, and evaluating advanced asset tracking applications.

Figure 17. **STEVAL-ASTRA1B reference design board**



In this document, we will focus on AB0037 STEVAL-ASTRA1SB system on board and AB0038 STEVAL-ASTRA1 main board that host the main kit functions. They can be identified in the following figure.

Figure 18. **STEVAL-ASTRA1B kit**


AB0037
The flexible antenna is connected to the AB0038 board via a FFC-FPC connector
AB0037 is soldered on top of AB0038
AB0077
AB0038
The AB0058 and AB0038 boards are electrically connected by means of the expansion connectors (header/receptacle)
AB0058

STEVAL-ASTRA1 system on board AB0037 and main board AB0038 schematic diagrams can be found in Section Appendix A.2.

4.1.1 STEVAL-ASTRA1B power domains

Typical example of power domains partition is designed in the STEVAL-ASTRA1B. This has been already well described in UM (STEVAL-ASTRA1B HW user manual UM2966)

4.1.2 STEVAL-ASTRA1B power management circuit functional description

The STEVAL-ASTRA1B power management design takes into account modularity, flexibility, and the ability to achieve the lowest power consumption.

Consider the following options for lowering power consumption when a component is not needed:

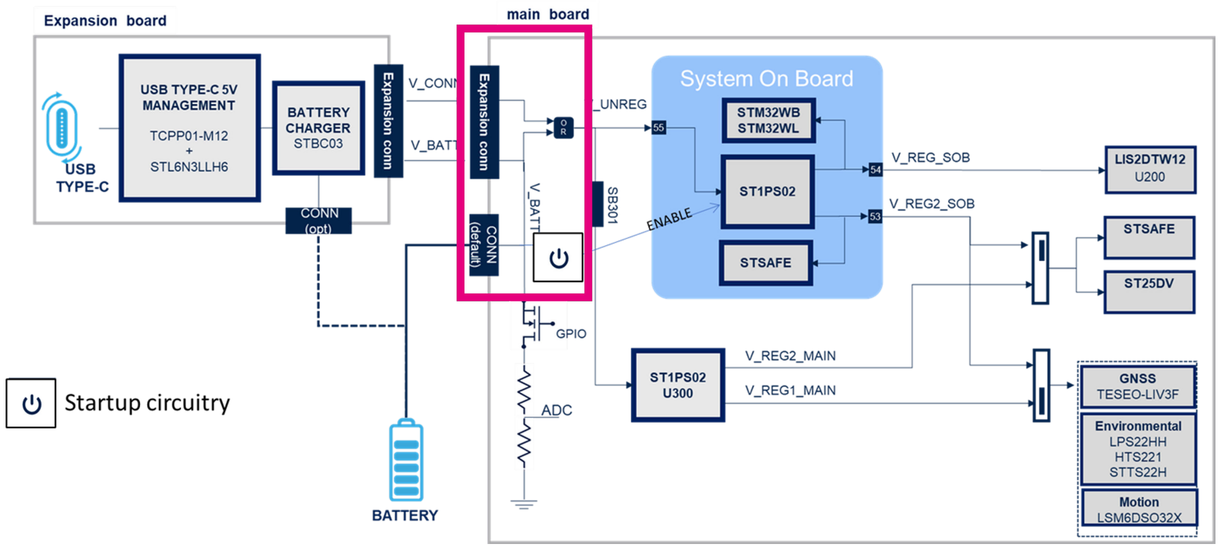
1. Send a firmware command to put the component in low power mode: this is preferable when you need to maintain its internal memory and configuration.
2. Disconnect the component's supply line: this may be preferable if you have to shut it down for a long time and you want to minimize all potential current leakages.

Looking at the power management circuit in Figure 19, the unregulated voltage coming from the battery and/or external battery connector supplies two DC-DC converters, thus generating four power domains that will be described in detail.

Note: *If such partitioning is not needed, then a single DC-DC can be used as the overall current consumption of the board is less than the used DC-DC (ST1PS02) capability.*

4.1.2.1 Startup circuitry

Figure 19. **STEVAL-ASTRA1B startup circuitry**



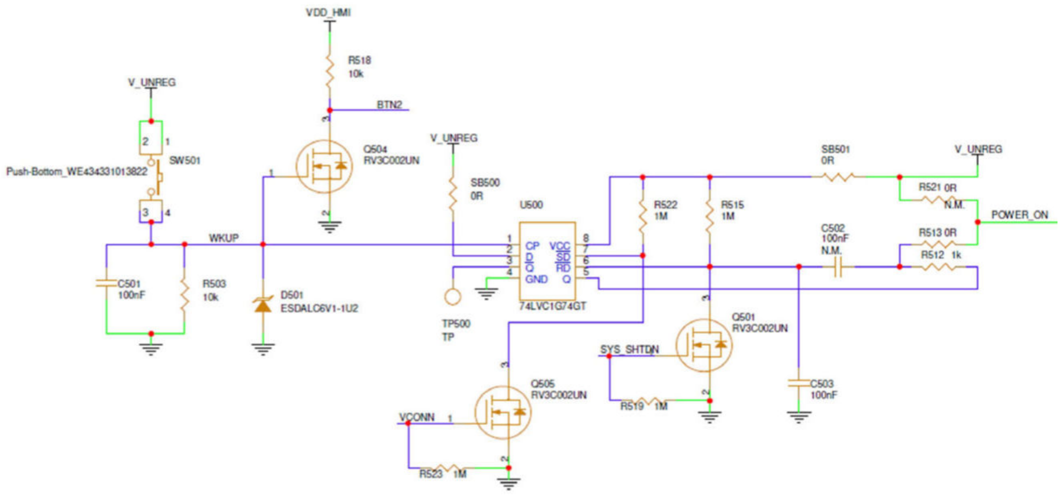
To facilitate the management of the STEVAL-ASTRA1B board shelf mode, a startup circuitry has been implemented. This circuit stands between the battery and the active parts to avoid draining the battery even if the battery is mechanically connected, allowing the board to stay on the shelf for years.

This is an important feature when using a primary battery as it preserves battery capacity while remaining in stock, only consuming a negligible amount of energy. In this condition, the current consumption in shelf mode is approximately 150 nA, or about one milliamp per year. This is achieved by means of a single D-type flip-flop with set and reset.

This "1-bit memory" can be used to drive the main voltage regulator (or a simple electronic switch). A start trigger (usually a push button) can set the startup circuitry output to high, enabling board operation. The reset of this startup circuitry can be driven directly by the MCU, thus implementing a self-shutdown feature.

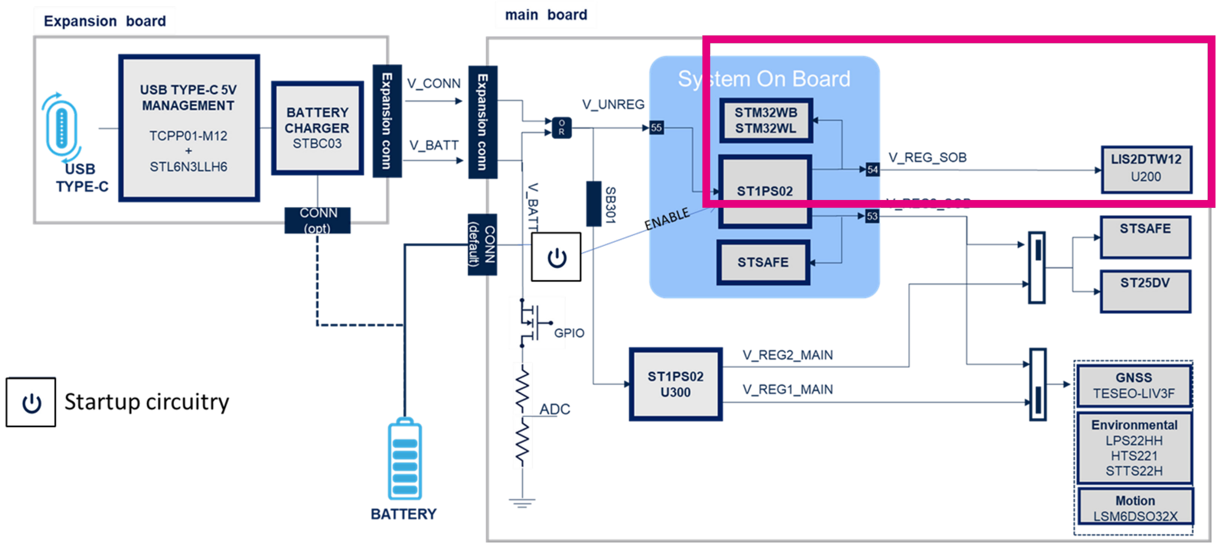
The following schematic diagram is of the startup circuit implemented on the STEVAL-ASTRA1B board.

Figure 20. **STEVAL-ASTRA1B startup circuitry schematic**



4.1.2.2 *Main power domain*

Figure 21. **STEVAL-ASTRA1B main power domain**



MCUs are the core components of the application and are powered by the main voltage regulator. The MCUs handle information acquisition and processing, followed by placing the board in specific states. When the board is in low power state, an accelerometer can be configured as a wake-up source, but it must remain powered during this phase, which is why the accelerometer is connected to the same power supply as the MCU.

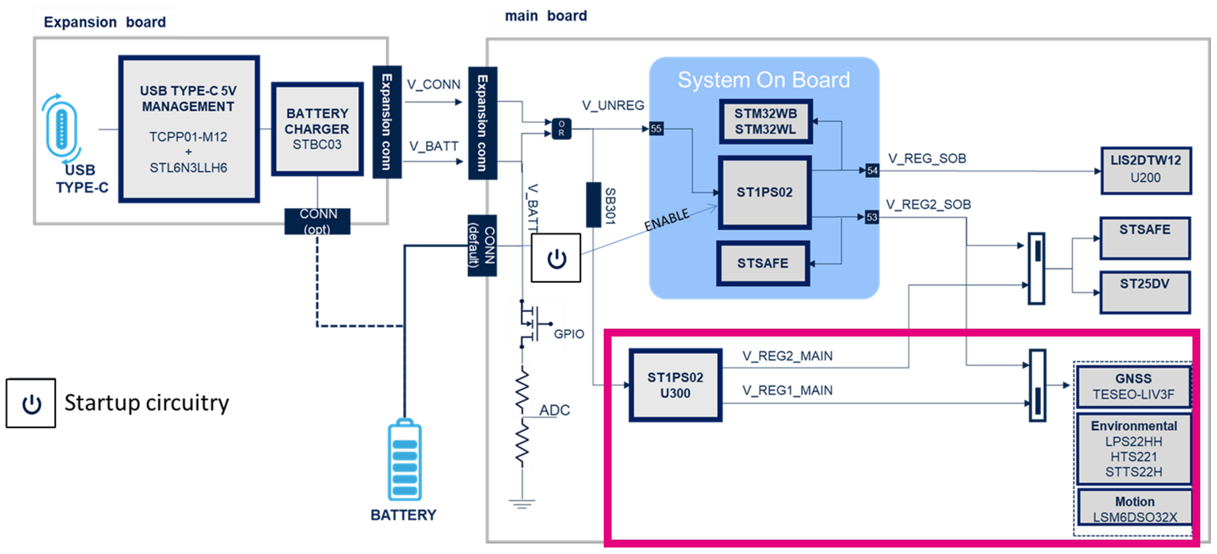
The main power domain is enabled by the startup circuitry. However, a zero-ohm resistor on the system on board (AB0037 SB916) can be mounted to disable the startup circuitry and keep the device always active.

Note: *When AB0037 SB916 mounted, AB0038 R513 must be removed to avoid conflicts on the POWER_ON signal.*

The overall power consumption on this power rail is 7.98 mA at 2.6 V and 8.97 mA at 3.3 V, measured on AB0037 SB918.

4.1.2.3 Secondary power domain

Figure 22. STEVAL-ASTRA1B secondary power domain



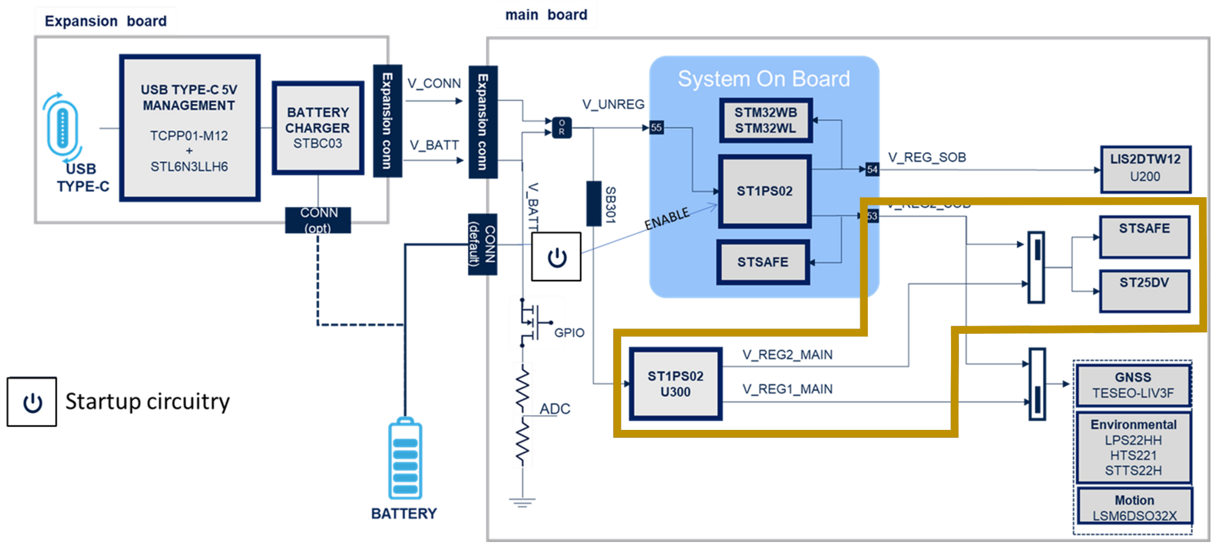
The main features of an asset tracking system are implemented by the GNSS for geolocation and by environmental and motion sensors. They are connected to the main output of the ST1PS02 (U300) of the main board, called V_REG1_MAIN.

The U300 ENABLE signal is connected to a GPIO.

The overall power consumption on this power rail is 49.5 mA, measured on AB0038 SB302.

4.1.2.4 Auxiliary power domain

Figure 23. STEVAL-ASTRA1B auxiliary power domain



The devices that only need to be activated for specific activities (e.g. cryptography, authentication, etc.) are connected to the secondary output of the system on-board voltage regulator. This output is enabled by a GPIO.

The overall power consumption on this power rail is 1.45 mA, measured on AB0037 SB940.

4.2 Application power states and STEVAL-ASTRA1B

In applications where entering a low-power state requires complex reconfiguration, the complexity of the operation can often cause errors that lead to firmware-related power losses. To avoid this risk, it is possible to provide different configuration flows in the firmware architecture, each relating to a specific operating state like “full run”, “low power”, and “ultralow power”, and a variable stored in memory that represents the state of the system.

Whenever you want to change the state (e.g., from full run to low power) instead of reconfiguring the system, you can simply update the state variable and reboot the system. The configuration flow used at startup is decided according to the variable stored in memory. Upon reboot, the system is clear of any previous configurations and should be able to load the new configuration dictated by the variable without risk of error.

4.2.1 Full run

In full run, all cores and peripherals are turned on and configured. All the regulators that manage the auxiliary power supplies are turned on to power all the elements present in the system (sensors, GNSS, modems, memories, etc.).

On the STEVAL-ASTRA1B:

- HW Power Manager, all regulators are enabled together with the auxiliary power lines
- Init and configure USB for debugging and configuration
- Init and configure NFC dynamic memory
- The sensors are turned on and configured to acquire at certain frequencies, depending on the selected use case
- Init and configure GNSS
- Init and configure BLE peripheral with high frequency of advertising period
- Powering up and configuration of STM32WL (as Lora ATCommand modem)
- Init and configure STSAFE for system security

4.2.2 Low power

In low power configuration, only necessary modules run, all regulators powering unnecessary branches and auxiliary channels are left off. The core remains in full run only for the time necessary for configuration, it is turned off the rest of the time by entering stop mode. Only the necessary peripherals are turned on and configured to work in conditions with the lowest possible consumption.

On the STEVAL-ASTRA1B:

- Of the voltage regulators, only the main V_REG_SOB voltage channel is turned on, which powers only the two MCUs and the very low consumption sensor
- STM32WL is brought into stop2 mode via the specific AT command, with very low power consumption
- It is possible to choose between three modes of use depending on the type of waking from the ultralow power condition (stop2 mode):
  – Tamper mode: all peripherals are turned off, the system wakes up only via interrupt on GPIO (tamper simulated by button)
  – Activity/inactivity mode: waking is entrusted to the interrupt generated by an ultralow power accelerometer sensor. The sensor is configured to recognize certain movement conditions and generate interrupts that wake the system when detected.
  – Connection mode: waking is linked to an attempt to connect by a device via Bluetooth. In this case, the BLE peripheral is turned on and configured to work at very low consumption (advertising interval of the order of seconds); upon connection request, the peripheral generates an interrupt that wakes up the system.

5 Measurements and instruments

To evaluate consumption on the STEVAL-ASTRA1B board, a tester was used to obtain the average current values.

To obtain more information on current waveforms, the STLINK-V3PWR and STM32CubeMonitor-Power tool was used to generate graphs with waveforms that describe consumption profiles during code execution.

5.1 Considerations on low power measurements: leakage on connected instruments

When taking current measurements to evaluate device consumption, you should be aware of the instruments used to carry out the measurements and those connected to the device for other reasons, such as a programmer/ debugger. This may introduce additional consumption to the circuit and distort measurements.

In the Full Run condition, the system current consumption is generally in the order of a milliamp. Therefore, the contribution coming from the connected programmer/debugger tool is negligible. This is not true in Low Power, when the consumption is in the order of a few microamps or less, and the absorption of the programmer/debugger tool becomes the only significant value.

To carry out low power measurements, it is necessary to ensure that any instrumentation is disconnected from the device before continuing with the measurement. If you use a programmer tool for flashing, follow this recommended sequence:

1. Connect tool and flash the board
2. Disconnect programmer
3. Power cycle the board
4. Measure consumption

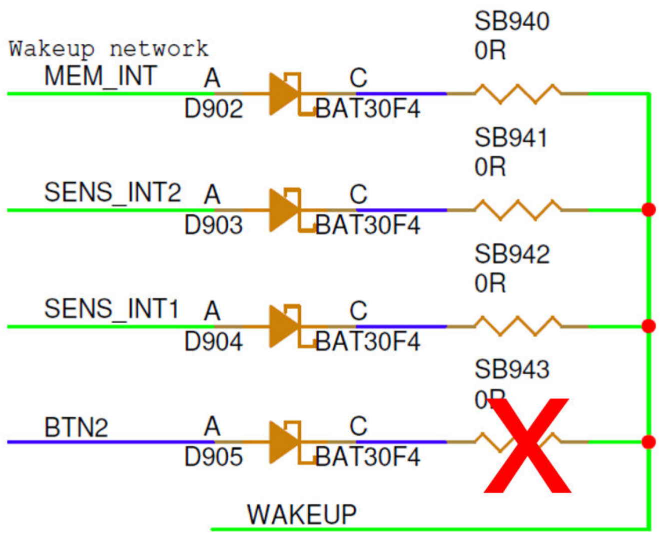
For more information about low power measurements, tools, and techniques you can refer to: AN5999: How to use STLINK-V3PWR to monitor power consumption on STM32 MCUs

5.2 Measurements on the STEVAL-ASTRA1B board

5.2.1 Hardware and firmware modification to replicate the measurements described in this document

The following measurements have been performed after removing the resistor SB943.

Figure 24. Solder bridge SB943 to be removed to avoid unwanted leakage



The BTN2 wakeup functionality will be lost; only BTN1 is able to wake up the system.

Be sure to check the following define in the app_conf.h file:

```
#define CFG_DEBUGGER_SUPPORTED    0
```

This disables debug support that adds consumption in the order of hundreds of microamps.

When the board is running on battery, the required interaction can be achieved via BLE connection using a dedicated smartphone app. It is recommended to use the "STASSETTRACKING" app. For further information on the app usage with STEVAL-ASTRA1B, please refer to UM3019. The following paragraphs refer to "Custom commands" menus that are described in UM3019 Par. 8.3.1.

5.2.2 Measurement setup

The following measurements are described here:

- Setup 1: average current measurement; board running on battery, battery voltage 4 V, tester connection as in Figure 26
- Setup 2: current absorption waveform; board running on battery, battery voltage 4 V, power analyzer connection as in Figure 27

With reference to the CustomUC (custom use case) firmware project in the FP-ATR-ASTRA1 v2.0.0 firmware package, a simple measurement can be performed in run mode.

After firmware loading, let the board run for several seconds to complete the initial configuration.

In the following schematized picture, you can see the average current consumption of 56 mA obtained with Setup 1.

Figure 25. **Setup 1 schema**

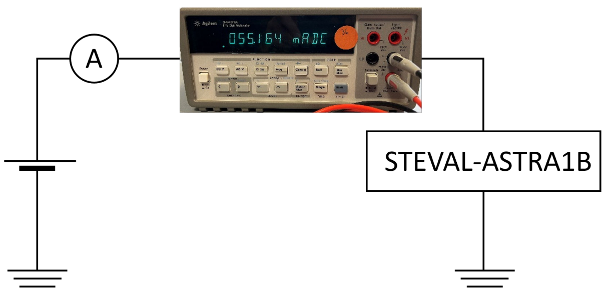
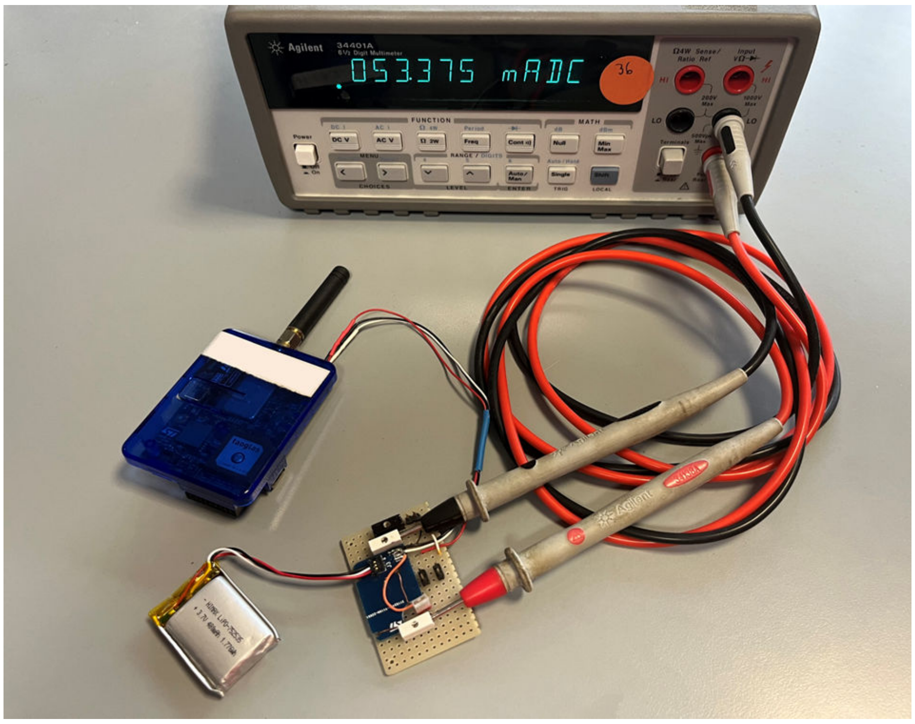
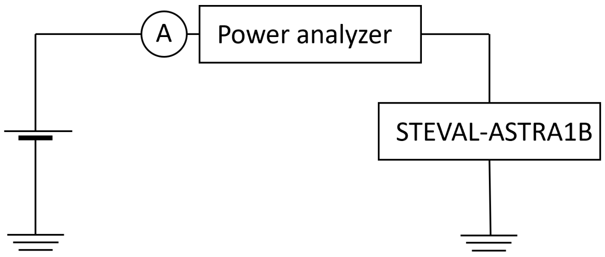


Figure 26. **Setup 1**



The setup 2 below is used for measurements in low-power mode. The current waveform can be viewed using a power analyzer.

Figure 27. **Setup 2 schema**



The STEVAL-ASTRA1B kit is equipped with a 3.7 V, 480 mAh LiPo battery. The following measurements and the estimated battery life are based on the use of this battery.

5.2.3 Measurements in low-power mode

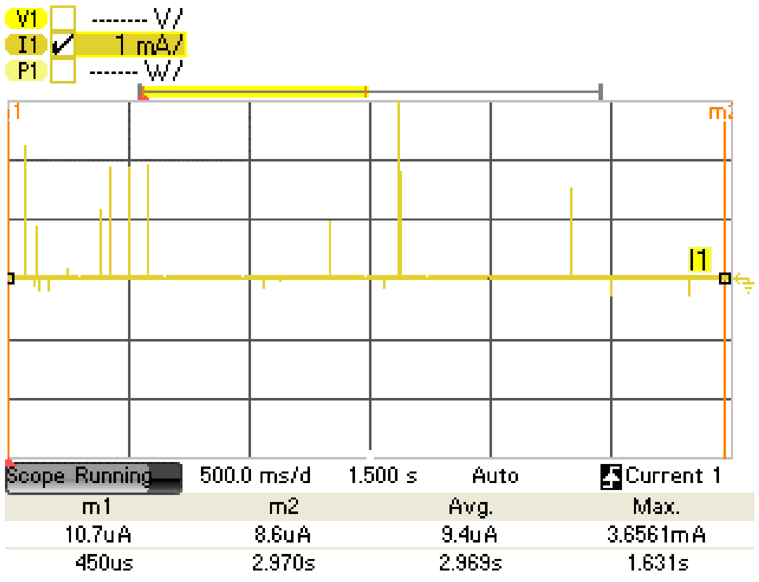
Using the smartphone app, you can send commands to the board via BLE with minimal interaction and without cable connection.

The board can be configured to enter low power mode with different configurations. The average current consumption with these configurations are highlighted in each image.

5.2.3.1 Low power mode with accelerometer wakeup and BLE advertising

1. Connect the board with the smartphone app
2. Tap on [**Connect to a device**], select your board and connect it.
3. Select [**more**] on bottom right side → [**Board configuration**] → [**Read custom command**] (Board Settings) → [**Ultra Low Power State**] (Custom Commands) and select [**BLE and ACC wake**].
4. Tap [**Ok**] on the windows that will open and disconnect device

Figure 28. Current consumption - Low power mode with accelerometer wakeup and BLE advertising

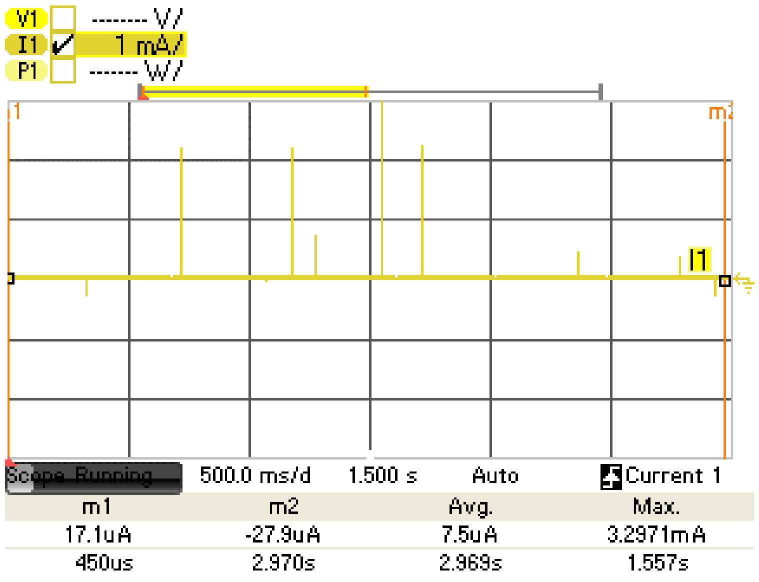


m1	m2	Avg.	Max.
10.7uA	8.6uA	9.4uA	3.6561mA
450us	2.970s	2.969s	1.631s

5.2.3.2 *Low power mode with BLE advertising*

1. Connect the board with the smartphone app.
2. Tap on [**Connect to a device**], select your board and connect it.
3. Select [**more**] on bottom right side → [**Board configuration**] → [**Read custom command**] (Board Settings) → [**Ultra Low Power State**] (Custom Commands) and select [**Only BLE wake**].
4. Tap [**Ok**] on the windows that will open and disconnect device.

Figure 29. Current conumption - Low power mode with BLE advertising

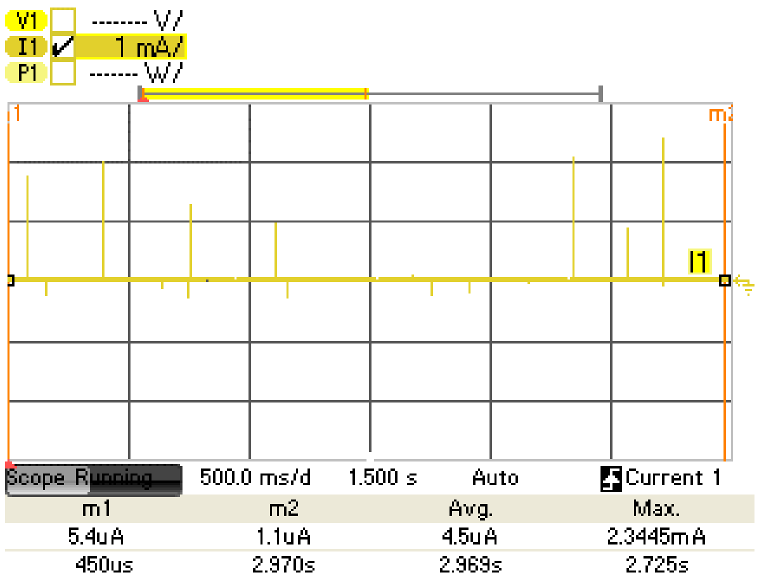


m1	m2	Avg.	Max.
17.1uA	-27.9uA	7.5uA	3.2971mA
450us	2.970s	2.969s	1.557s

5.2.3.3 *Low power mode with accelerometer wakeup*

1. Connect the board with the smartphone app.
2. Tap on [**Connect to a device**], select your board and connect it.
3. Select [**more**] on bottom right side → [**Board configuration**] → [**Read custom command**] (Board Settings) → [**Ultra Low Power State**] (Custom Commands) and select [**Only ACC wake**].
4. Tap [**Ok**] on the windows that will open and disconnect device.

Figure 30. **Current consumption - Low power mode with accelerometer wakeup**



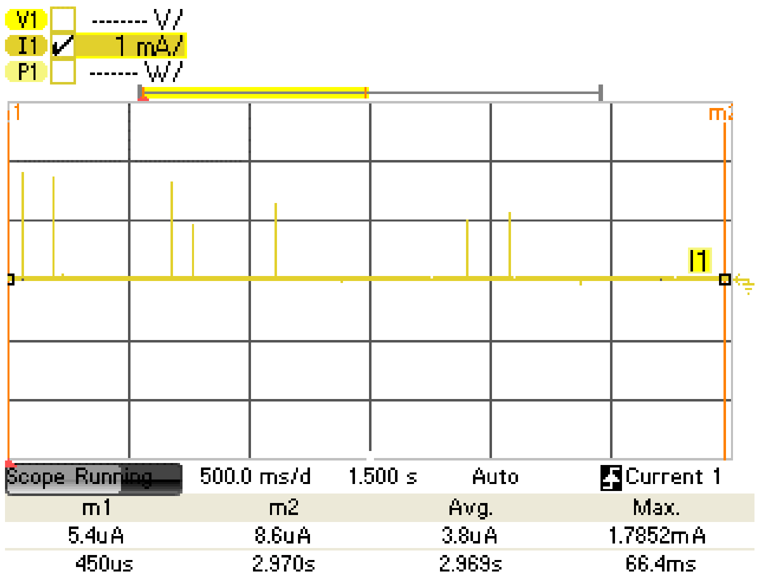
500.0 ms/d 1.500 s Auto Current 1

m1	m2	Avg.	Max.
5.4uA	1.1uA	4.5uA	2.3445mA
450us	2.970s	2.969s	2.725s

5.2.3.4 *Low power mode*

1. Connect the board with the smartphone app.
2. Tap on [**Connect to a device**], select your board and connect it.
3. Select [**more**] on bottom right side → [**Board configuration**] → [**Read custom command**] (Board Settings) → [**Ultra Low Power State**] (Custom Commands) and select [**Button wake**].
4. Tap [**Ok**] on the windows that will open and disconnect device.

Figure 31. **Current consumption - Low power mode**



500.0 ms/d 1.500 s Auto Current 1

m1	m2	Avg.	Max.
5.4uA	8.6uA	3.8uA	1.7852mA
450us	2.970s	2.969s	66.4ms

5.3 STEVAL-ASTRA1B power consumption characterization

STEVAL-ASTRA1B and FP-ATR-ASTRA1 come with preconfigured use cases with the following power management characteristics:

- Fleet management
  - Always on
  - System State → Full run
  - Average current consumption (with 40-second LoRa transmission frequency and BLE in advertising):
    - 56 mA
    - estimated battery life ~8 hrs
- Logistics
  - Run mode
  - System State → Full run
  - Average current consumption (with 20-second LoRa transmission frequency and BLE in advertising):
    - 56 mA
    - estimated battery life ~8 hrs
- Goods monitoring
  - Run mode with GPS in standby(1)
  - System State → Run
  - Average current consumption (with 1-hour LoRa transmission frequency and BLE in advertising):
    - 11 mA
    - estimated battery life ~2 days
- Livestock monitoring
  - Several available functionalities in low power mode (accelerometer wakeup or BLE advertising)
  - System State → Low power on event (inactivity detection)
  - Average current consumption
    - 56 mA during run mode
    - 0.012 mA during low-power mode
    - Estimated battery life with 1% duty cycle / 1 transmission per hour ~40days
- Anti-tamper with keep alive
  - minimum functionalities
  - System State → Low power with timer for keep alive.
  - Average current consumption (with accelerometer wakeup and BLE advertising):
    - 12mA during LoRa transmissions (3 seconds, once per hour)
    - 9.4 µA during low-power mode
    - estimated battery life: 3 years

1. *The GnssRequestVersion() function in the firmware must be modified according to Section Appendix A.1*

6 Conclusions

This document provided several guidelines to design and implement a low-power IoT node with reference to STEVAL-ASTRA1B.

The IoT landscape is so vast, however, that we cannot cover all the power needs and related solutions spanning from super caps, advanced ultralow power processes, and harvesting techniques.

In this application note, we summarized how we defined low power for the STEVAL-ASTRA1B, which would allow a reference design for an IoT low power node.

Low power features can be enhanced with energy harvesting, and an example is given in AN6019 regarding STEVAL-ASTRA1B.

Appendix A

A.1 Modified GnssRequestVersion() function to implement GNSS standby

```
void GnssRequestVersion(void)
{
  PRINTF_INFO("TESEO GETVER\r\n");
  if(GNSSParser_Data.pstmver_data.pstmver_string[0] != 0)
  {
    PlatformStatus.b.GNSS_VERSION_SHOWED = 1;
    PRINTF_VERBOSE("GNSS FW version: ");
    PRINTF_VERBOSE((char const*)GNSSParser_Data.pstmver_data.pstmver_string);
    PRINTF_VERBOSE("\r\n");
    /* Add TESEO-LIV3F standby command if the GNSS is not used in application */
    if(PlatformStatus.b.GNSS_ENABLED == 0)
    {
      PRINTF_VERBOSE("GNSS is disabled");
      PRINTF_VERBOSE("Sending standby command");
      AstraLedColor(ASTRA_LED_OFF);
      AstraLedColor(ASTRA_LED_COLOR_BLUE);
      GNSS_DATA_SendCommand((uint8_t *)"$PSTMFORCESTANDBY,90000");
      HAL_Delay(200);
      AstraLedColor(ASTRA_LED_OFF);
    }
  }
  else
  {
    PRINTF_INFO("Get Binary Image Version\r\n");
    GNSS_DATA_SendCommand((uint8_t *)"$PSTMGETSWVER,6");
  }
}
```

A.2 Schematic diagrams

Figure 32. Main board (STEVAL-ASTRA1) circuit schematic (1 of 9)

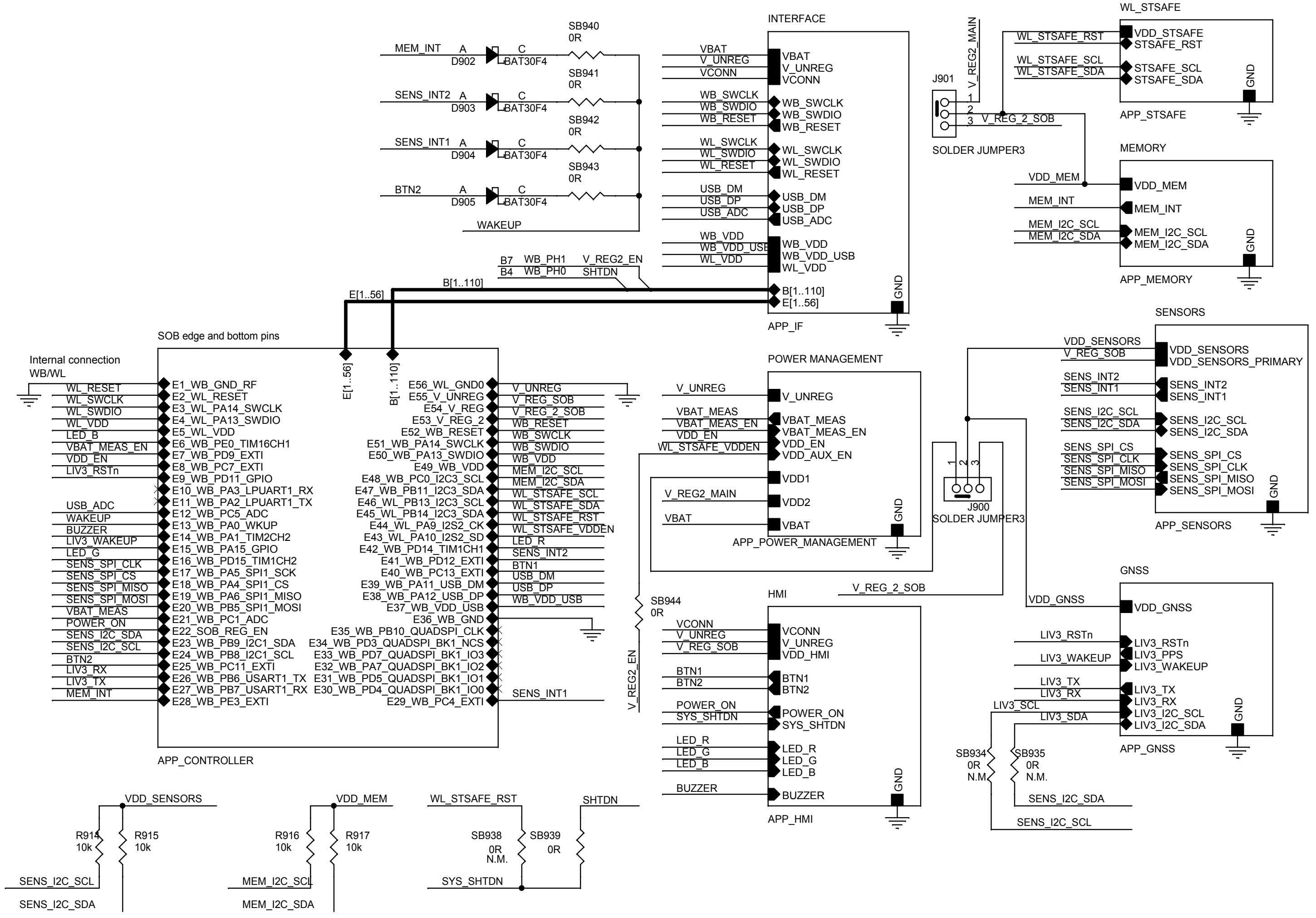
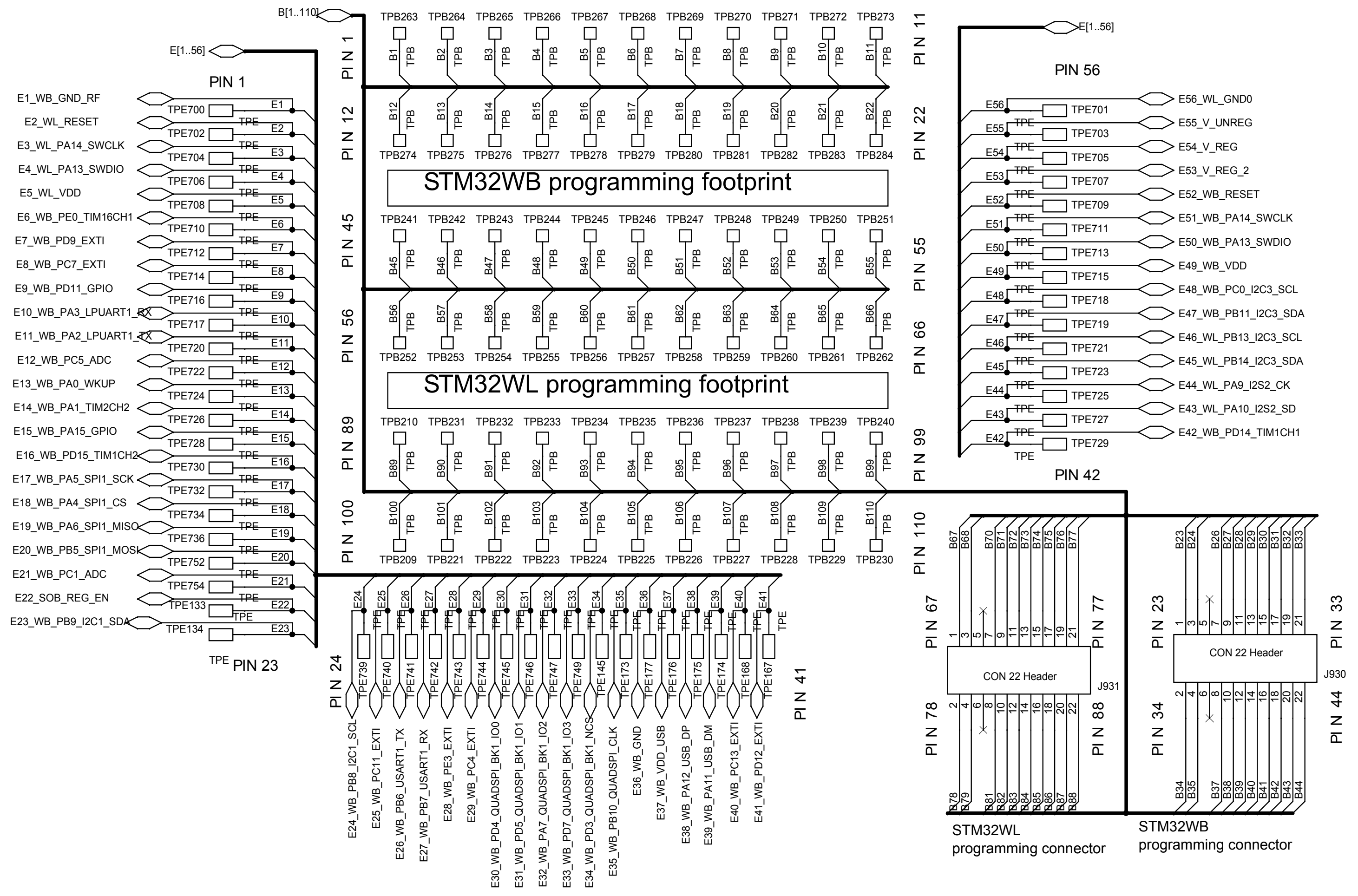


Figure 33. Main board (STEVAL-ASTRA1) circuit schematic (2 of 9)



AN6044

Figure 34. **Main board (STEVAL-ASTRA1) circuit schematic (3 of 9)**

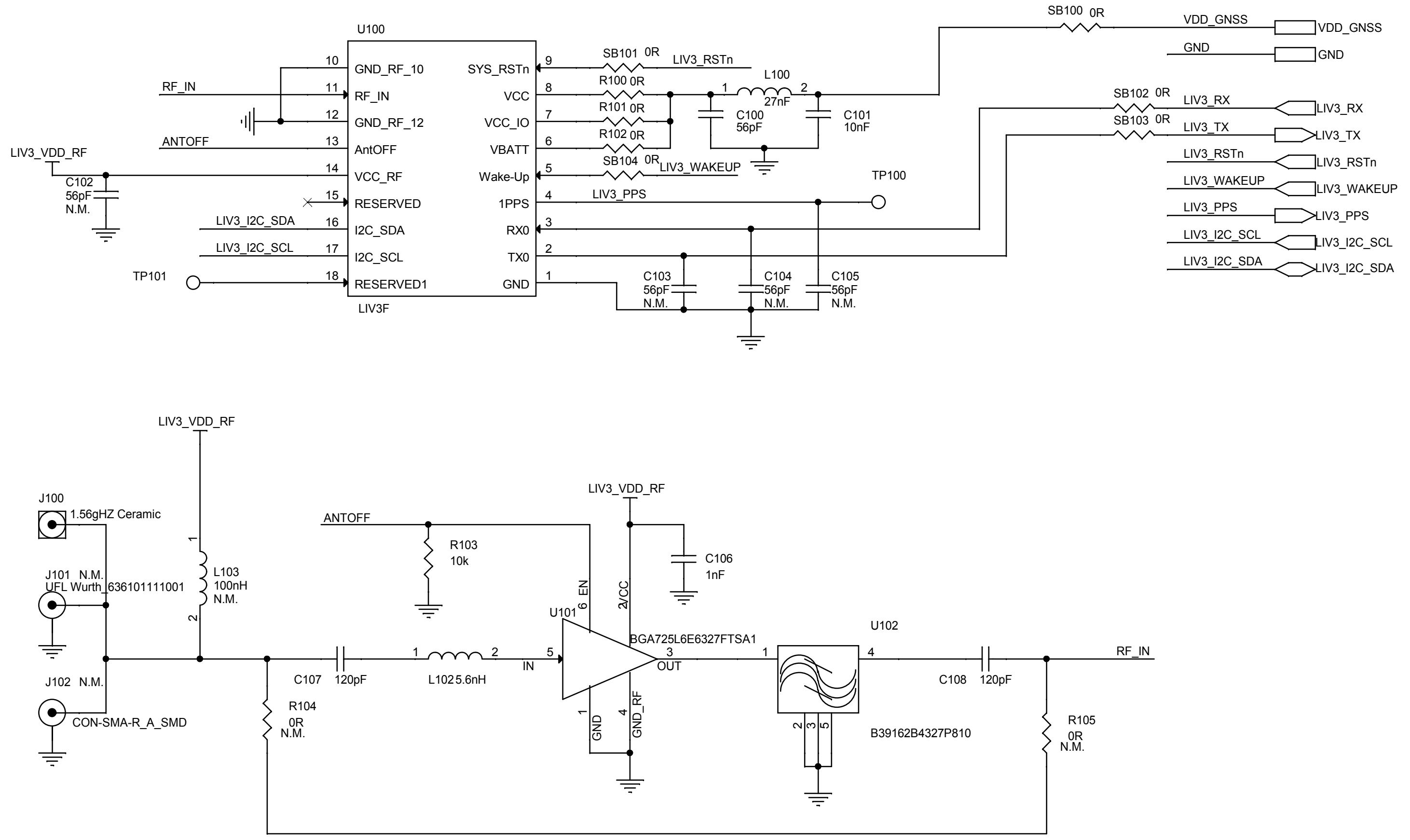


Figure 35. **Main board (STEVAL-ASTRA1) circuit schematic (4 of 9)**

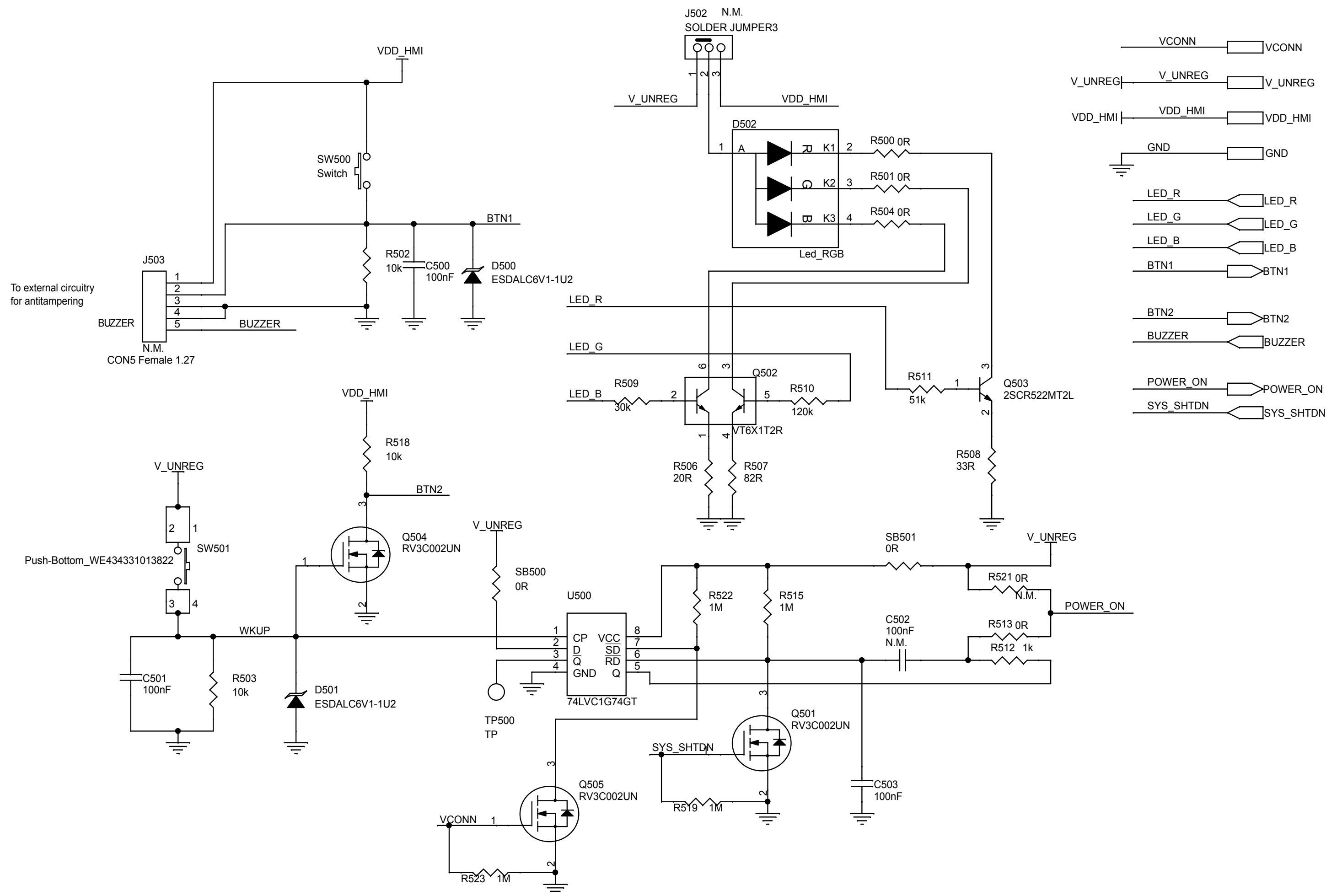


Figure 36. **Main board (STEVAL-ASTRA1) circuit schematic (5 of 9)**

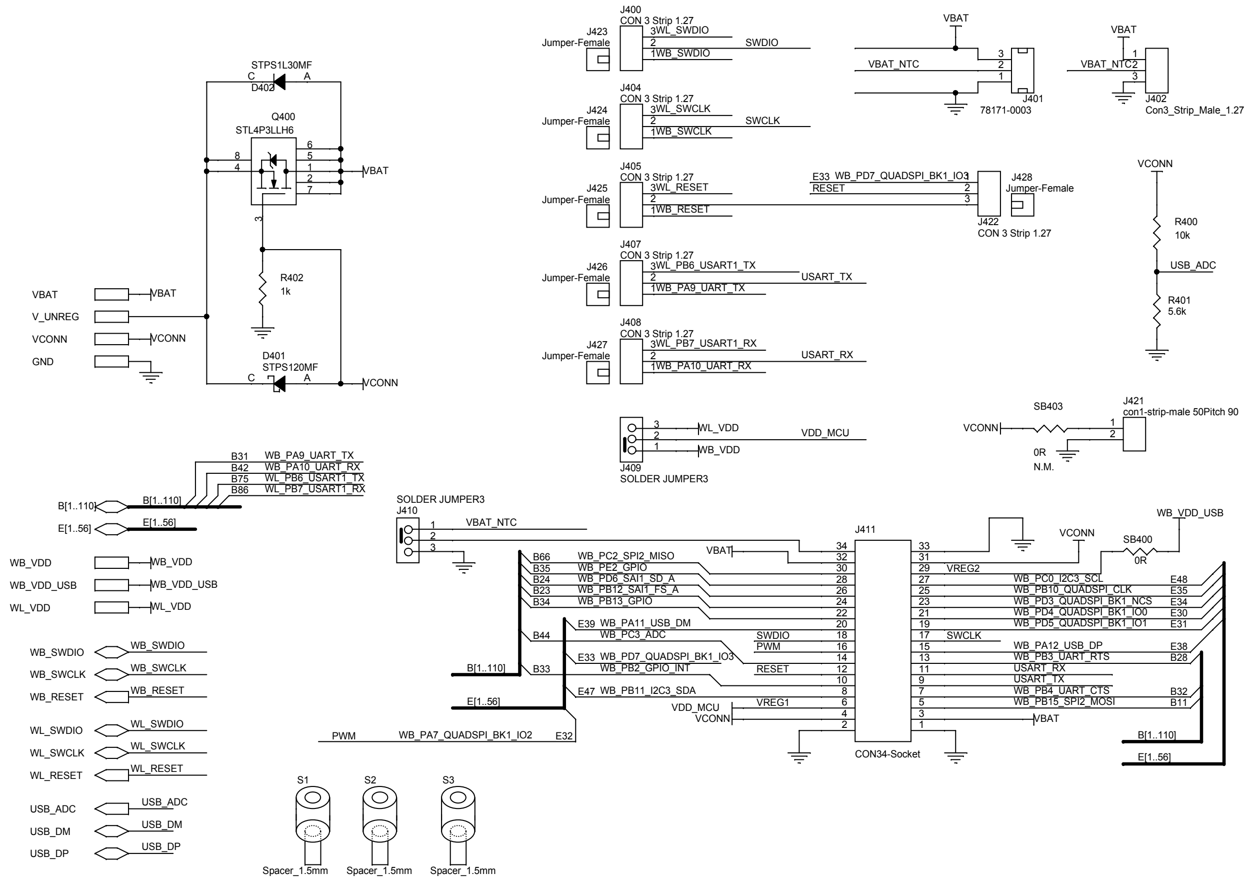


Figure 37. Main board (STEVAL-ASTRA1) circuit schematic (6 of 9)

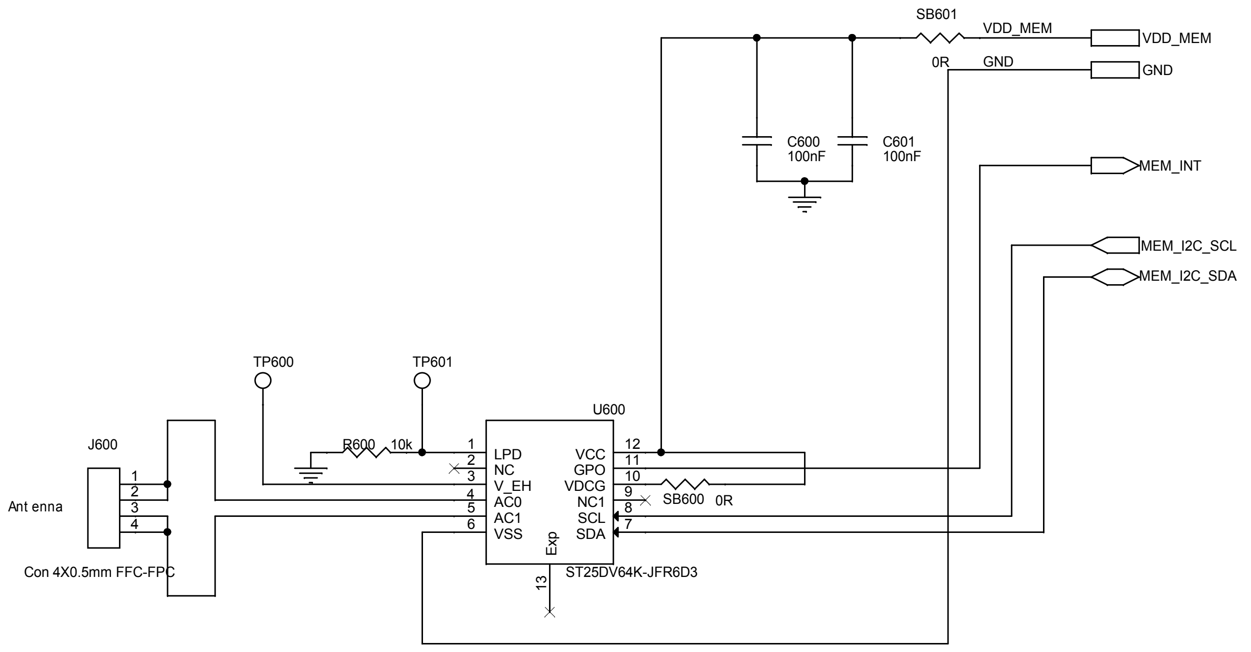


Figure 38. Main board (STEVAL-ASTRA1) circuit schematic (7 of 9)

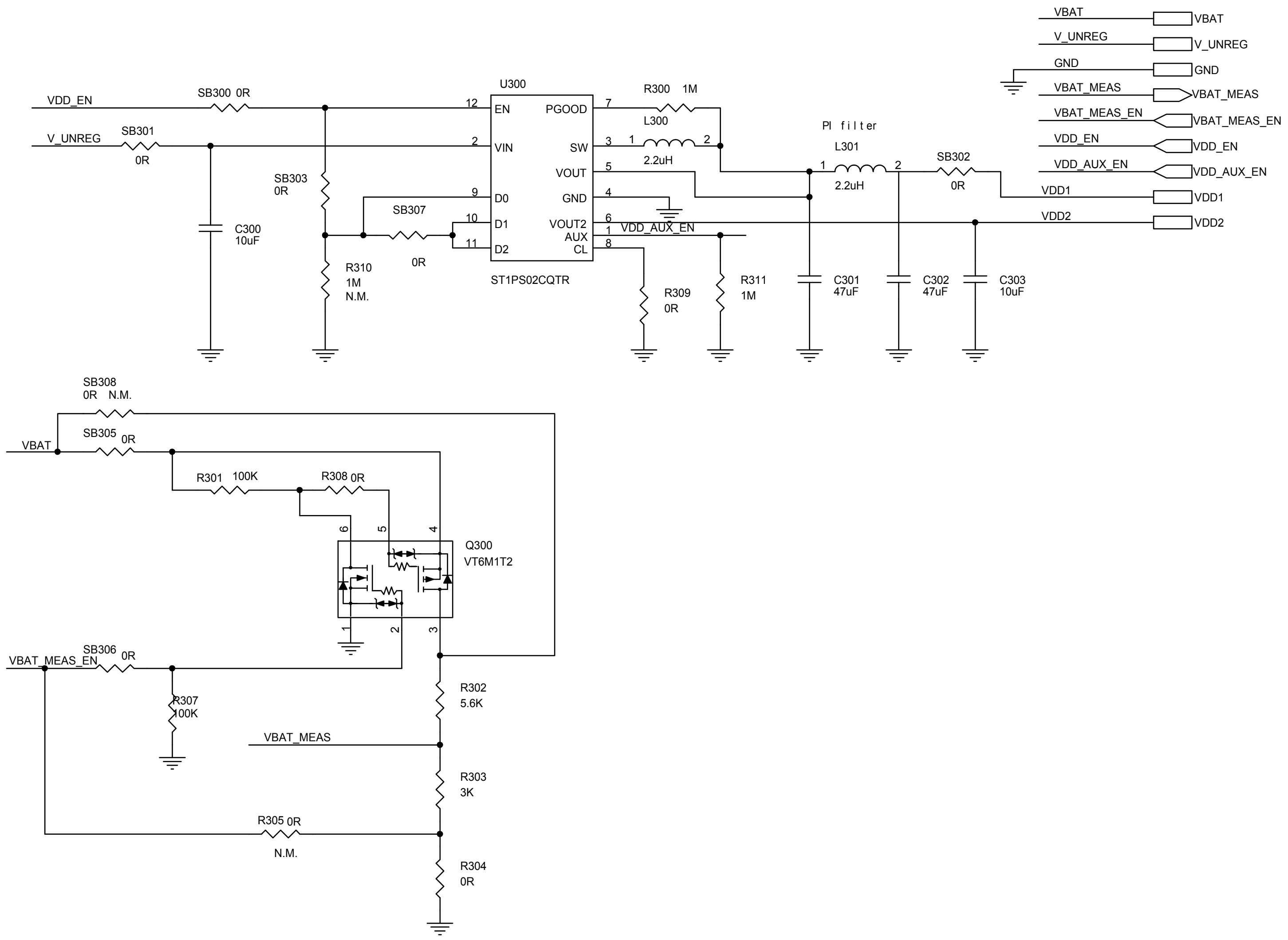


Figure 39. **Main board (STEVAL-ASTRA1) circuit schematic (8 of 9)**

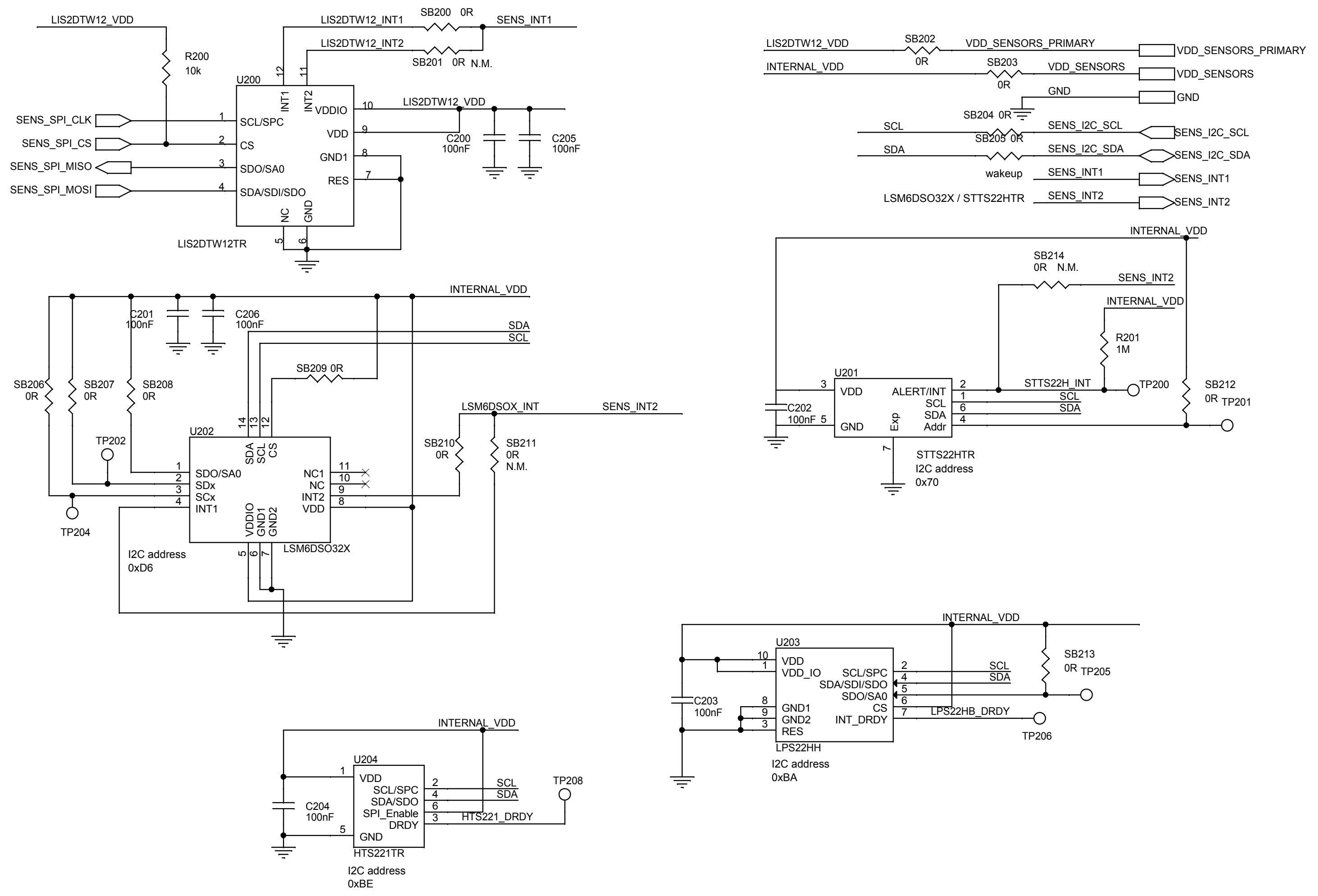


Figure 40. Main board (STEVAL-ASTRA1) circuit schematic (9 of 9)

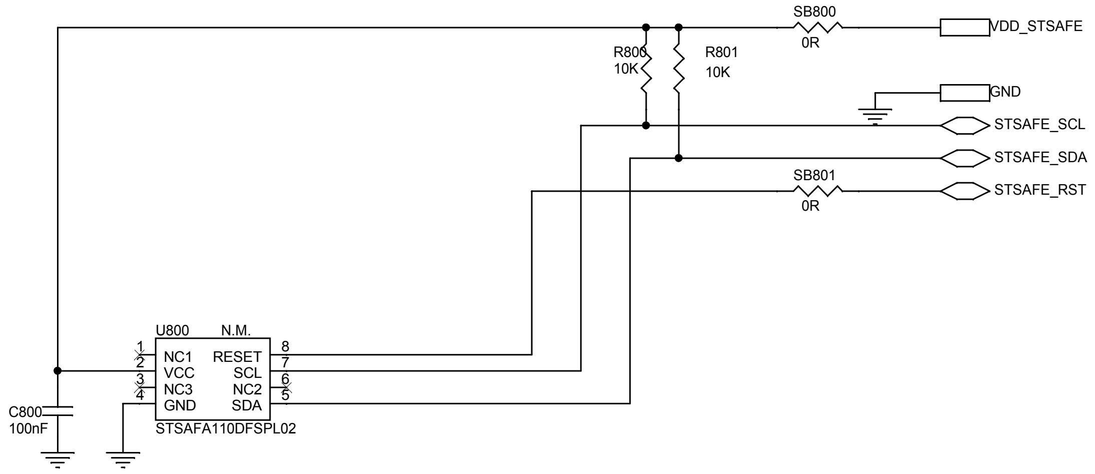


Figure 41. System on board (STEVAL-ASTRA1SB) circuit schematic (1 of 10)

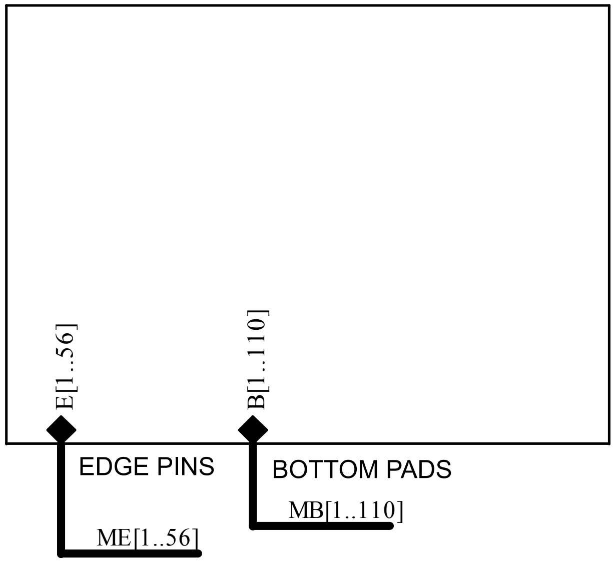


Figure 42. System on board (STEVAL-ASTRA1SB) circuit schematic (2 of 10)

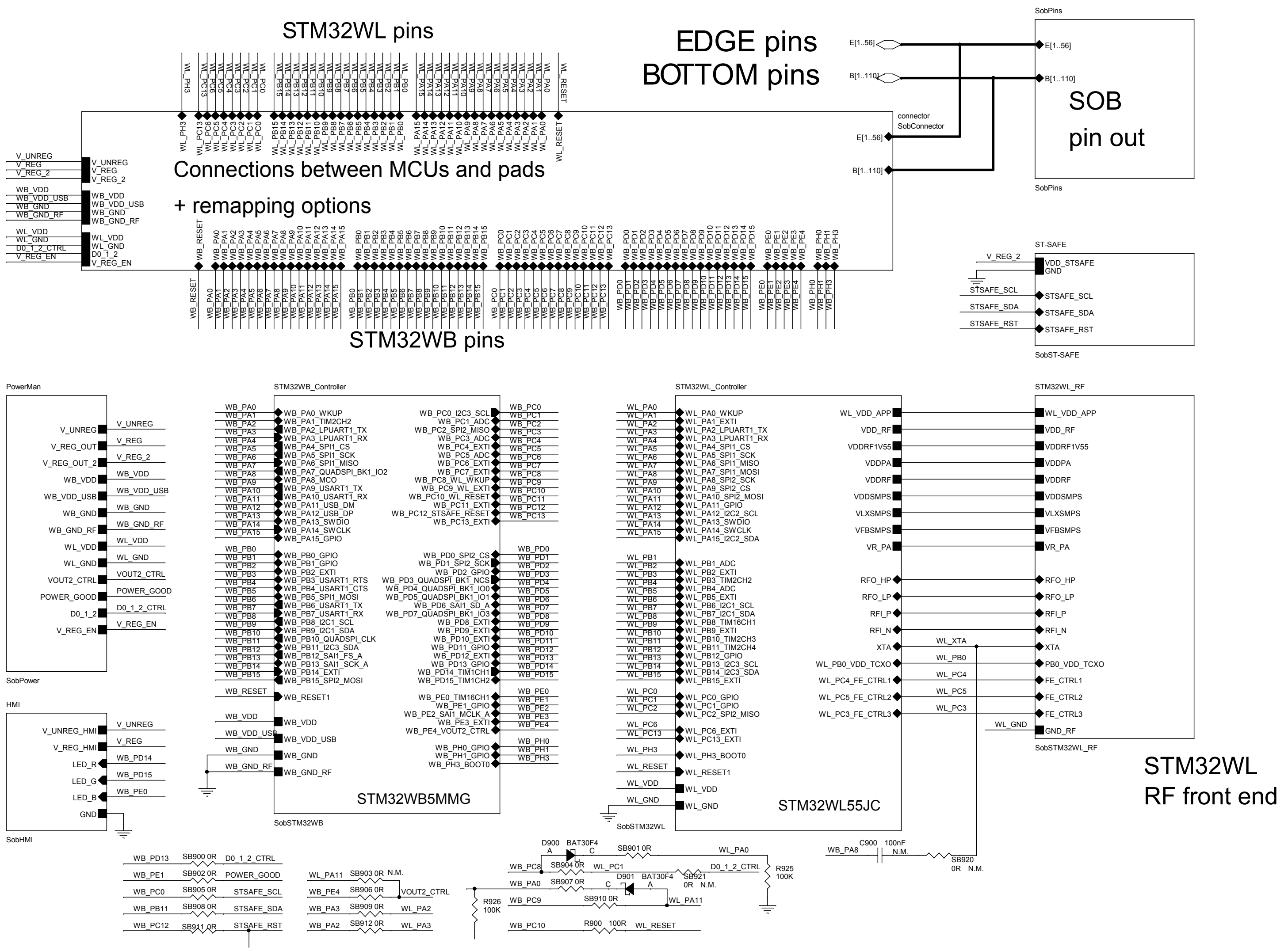


Figure 43. System on board (STEVAL-ASTRA1SB) circuit schematic (3 of 10)

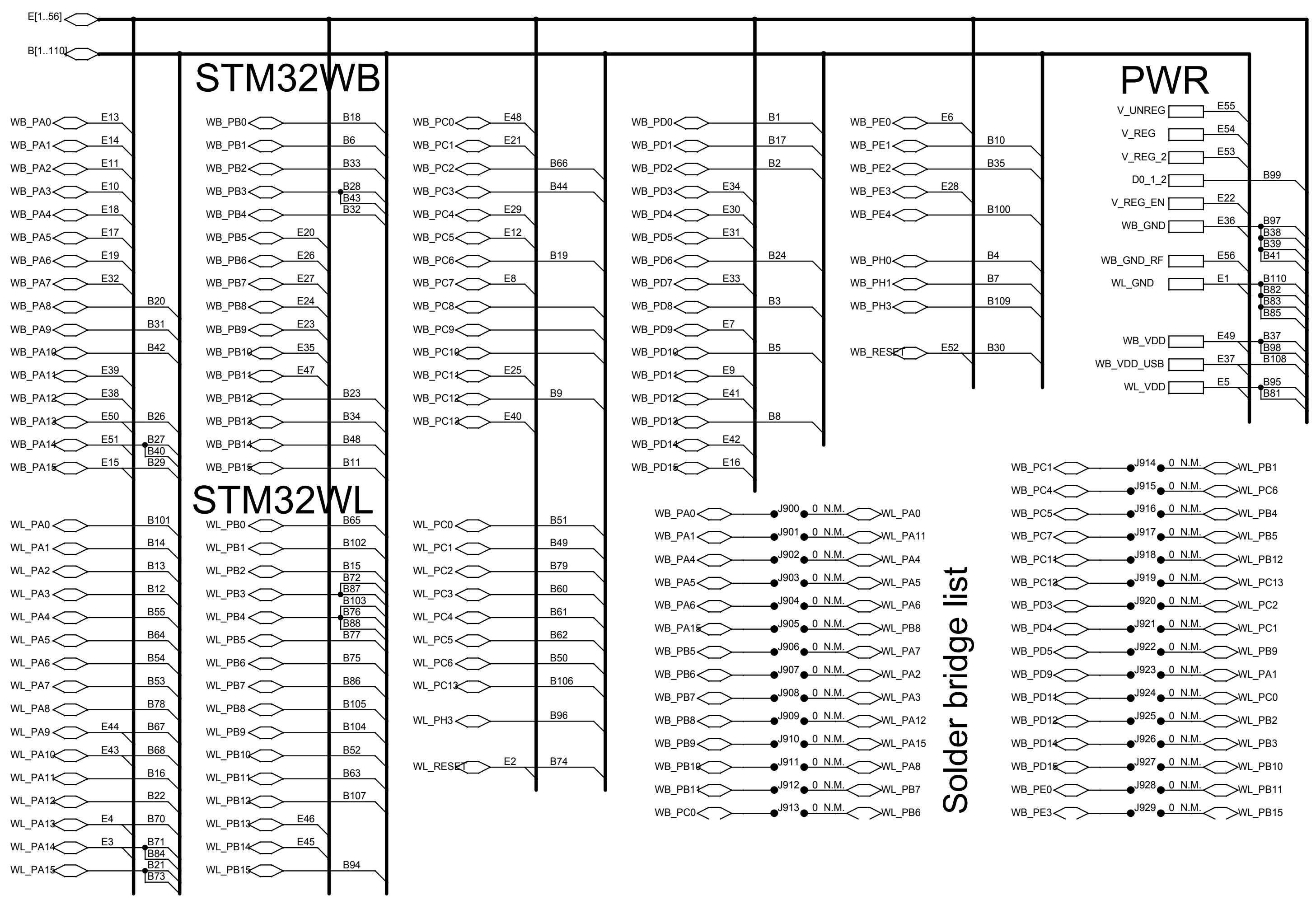


Figure 44. System on board (STEVAL-ASTRA1SB) circuit schematic (4 of 10)

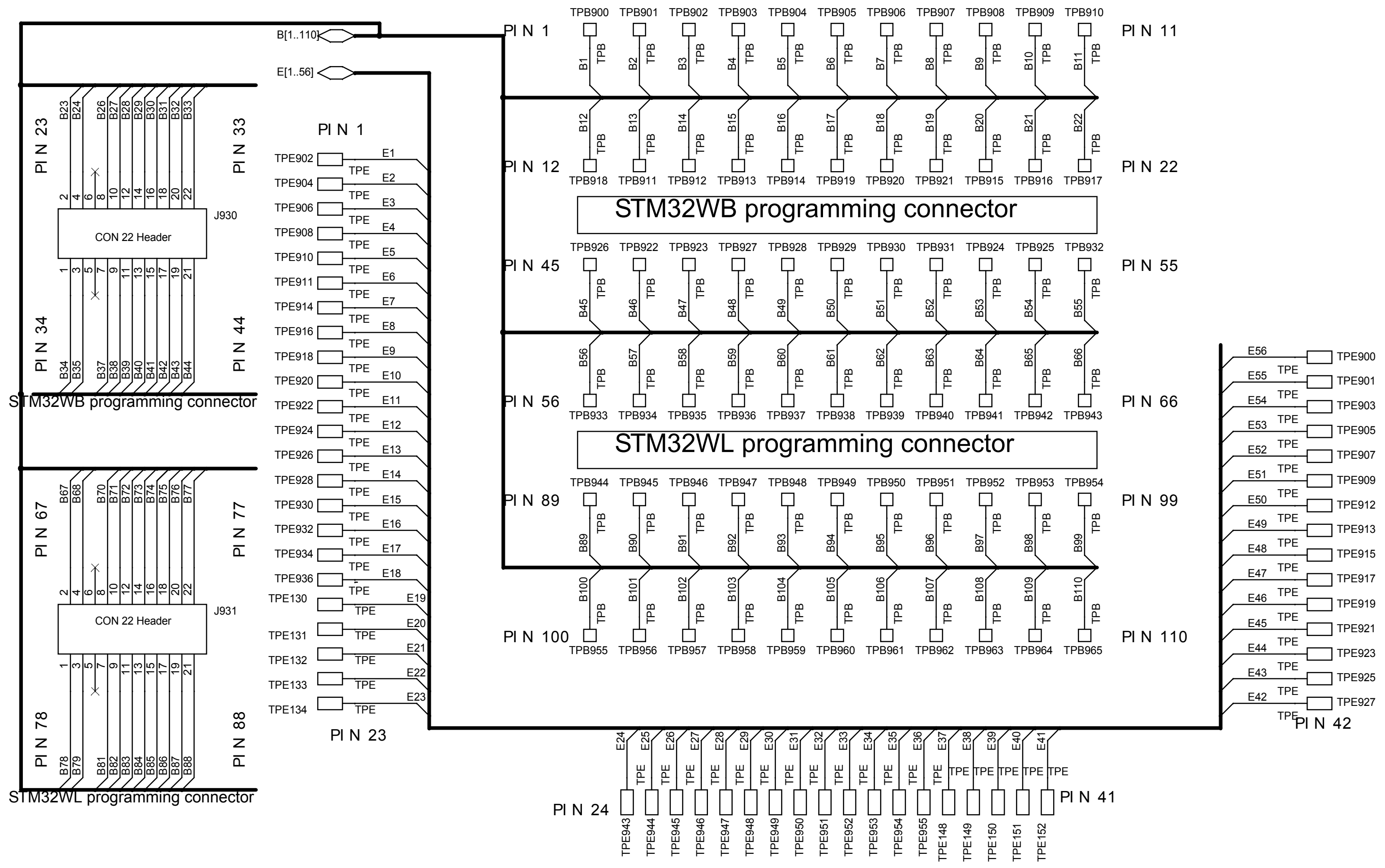


Figure 45. **System on board (STEVAL-ASTRA1SB) circuit schematic (5 of 10)**

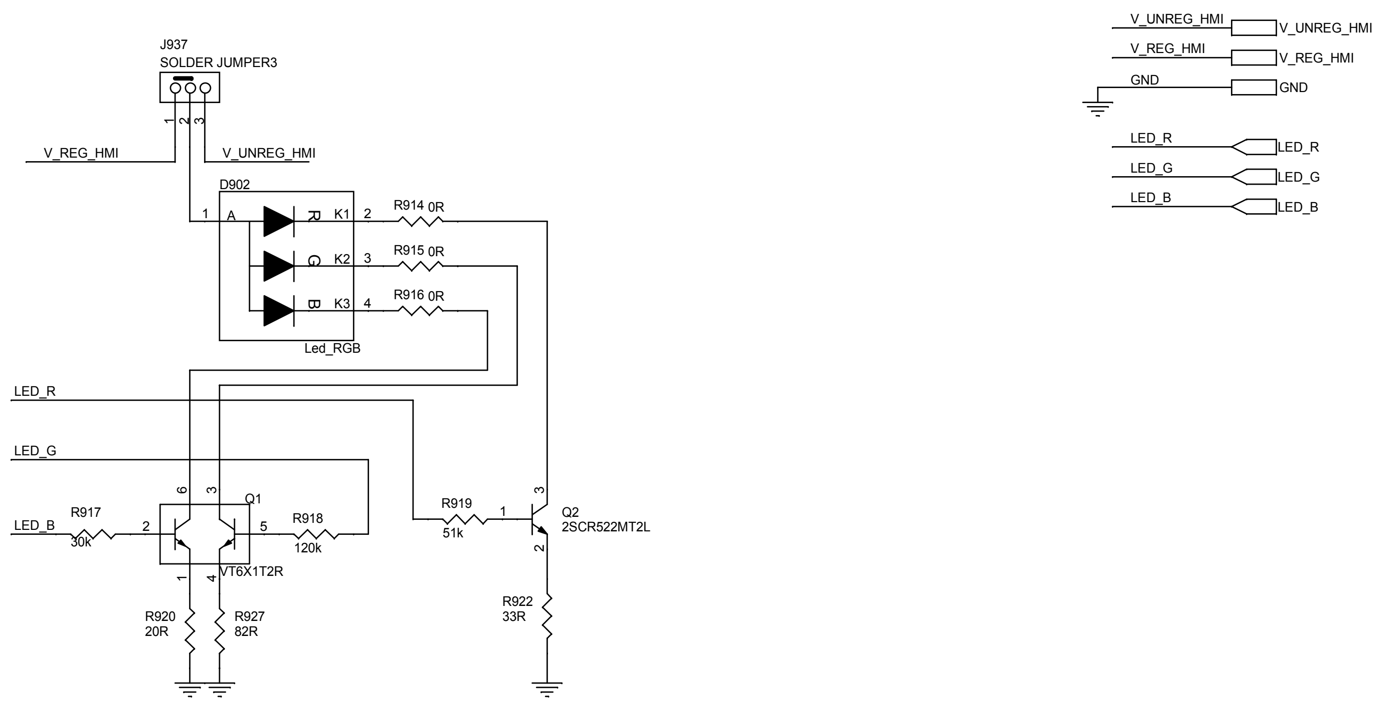


Figure 46. **System on board (STEVAL-ASTRA1SB) circuit schematic (6 of 10)**

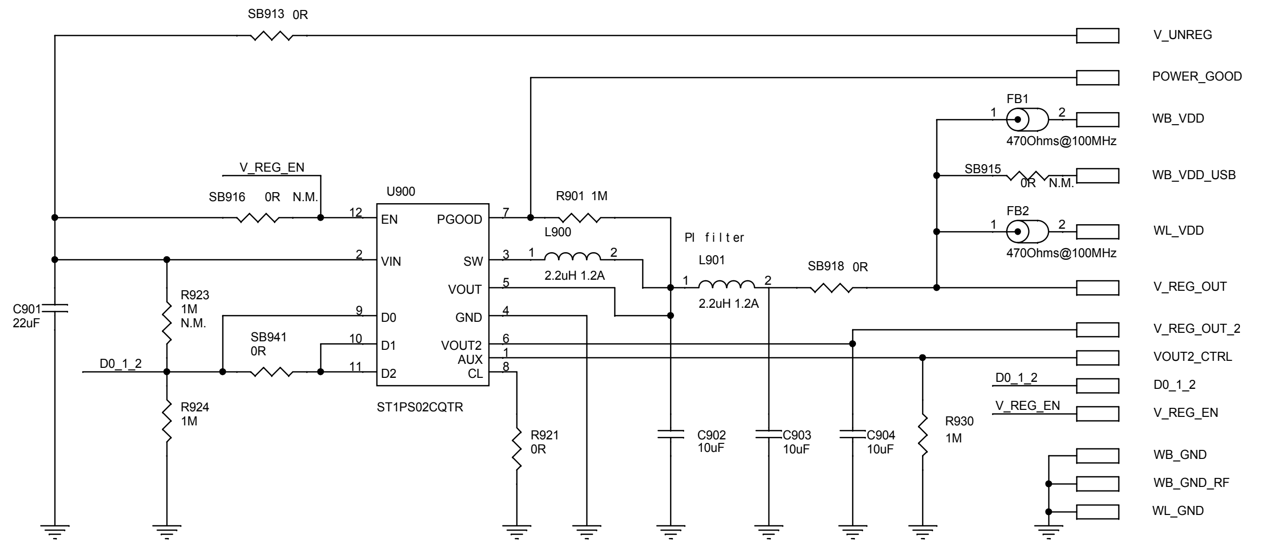


Figure 47. System on board (STEVAL-ASTRA1SB) circuit schematic (7 of 10)

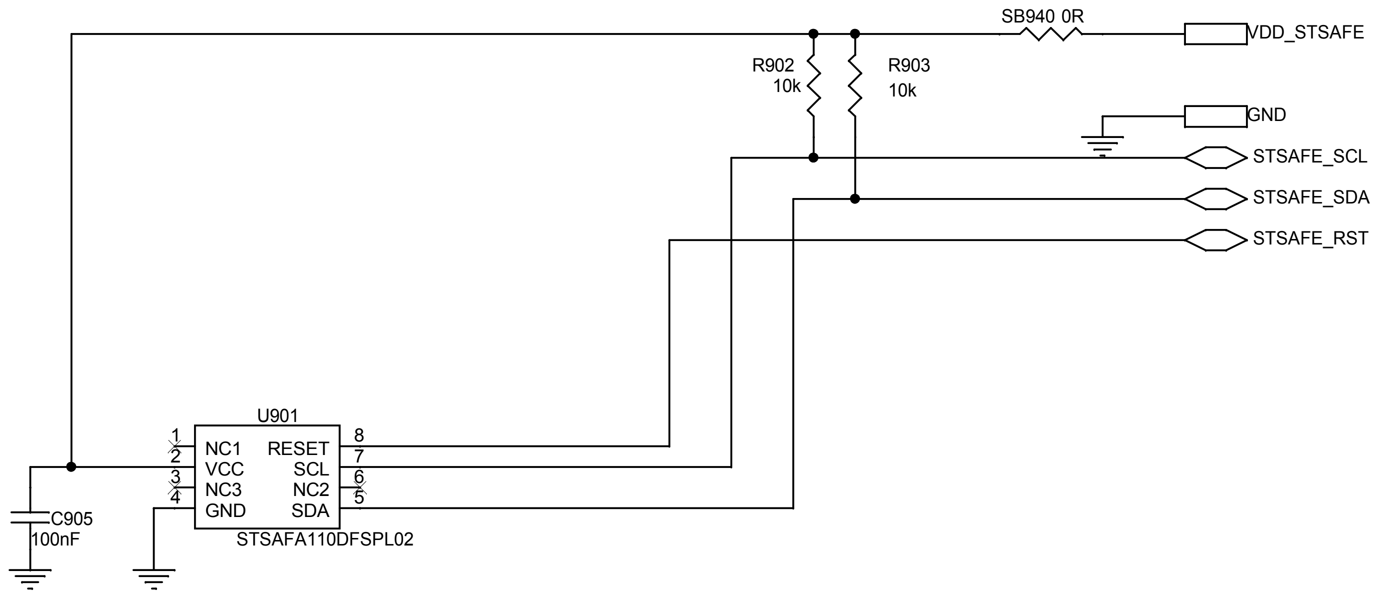


Figure 48. **System on board (STEVAL-ASTRA1SB) circuit schematic (8 of 10)**

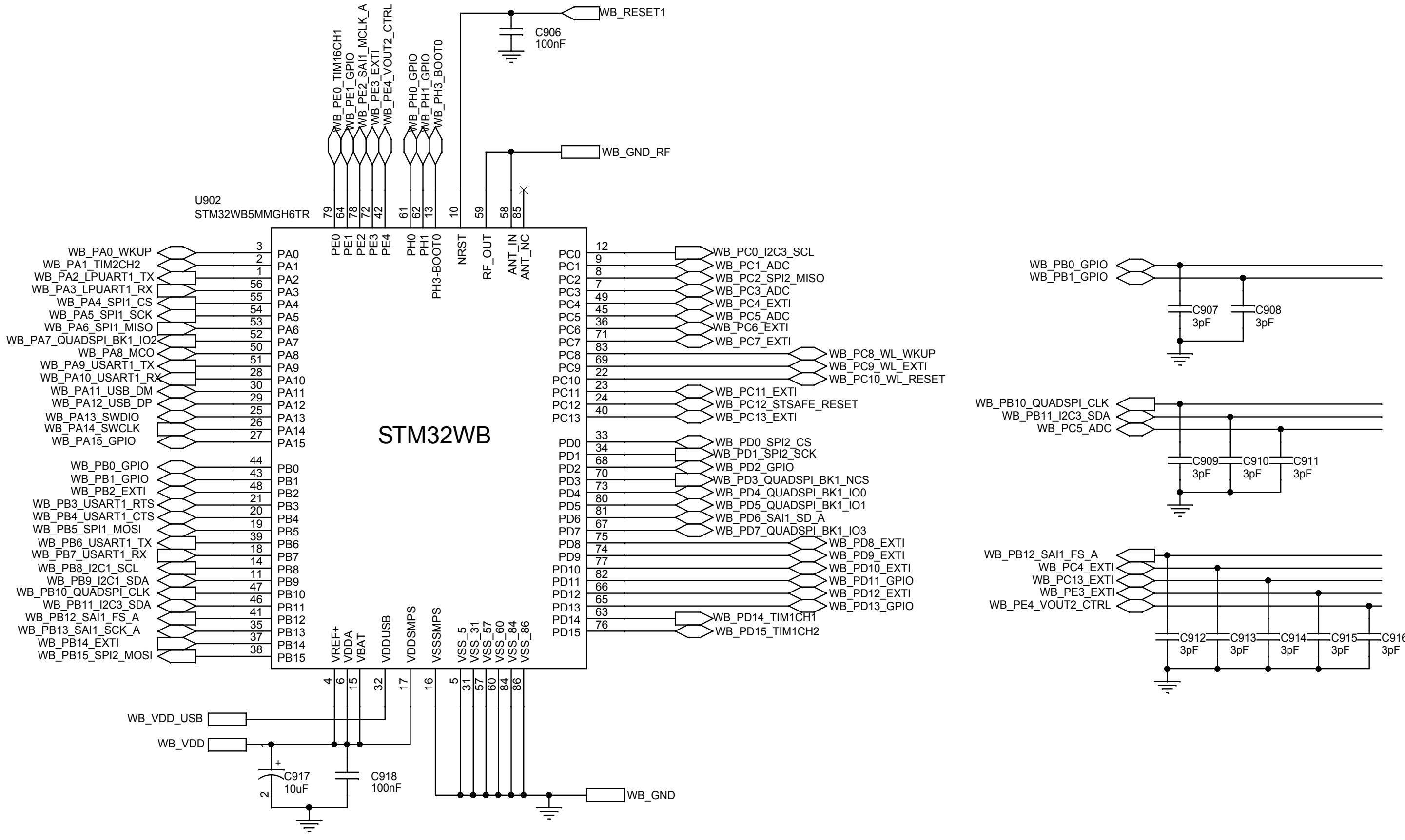


Figure 49. System on board (STEVAL-ASTRA1SB) circuit schematic (9 of 10)

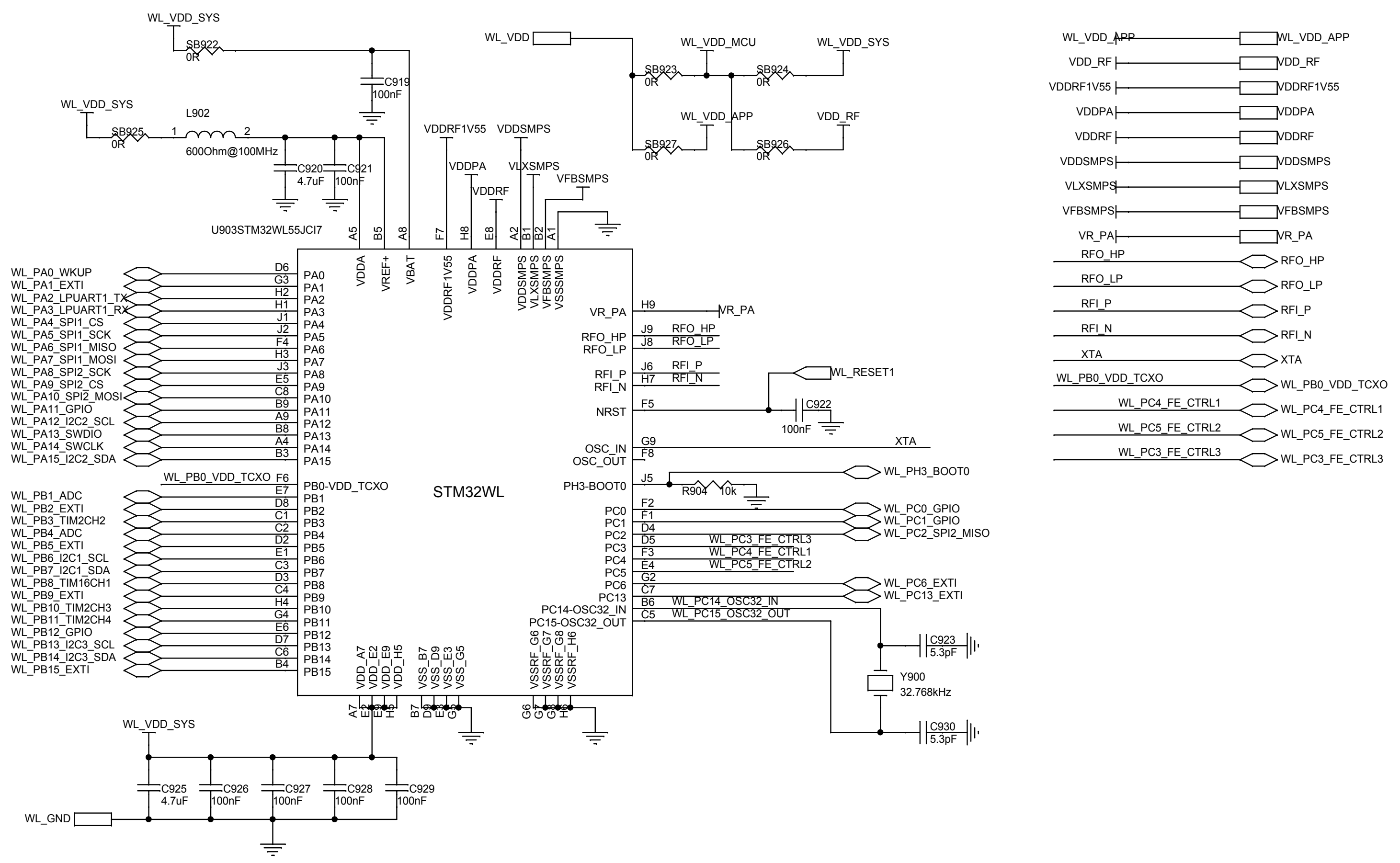


Figure 50. System on board (STEVAL-ASTRA1SB) circuit schematic (10 of 10)

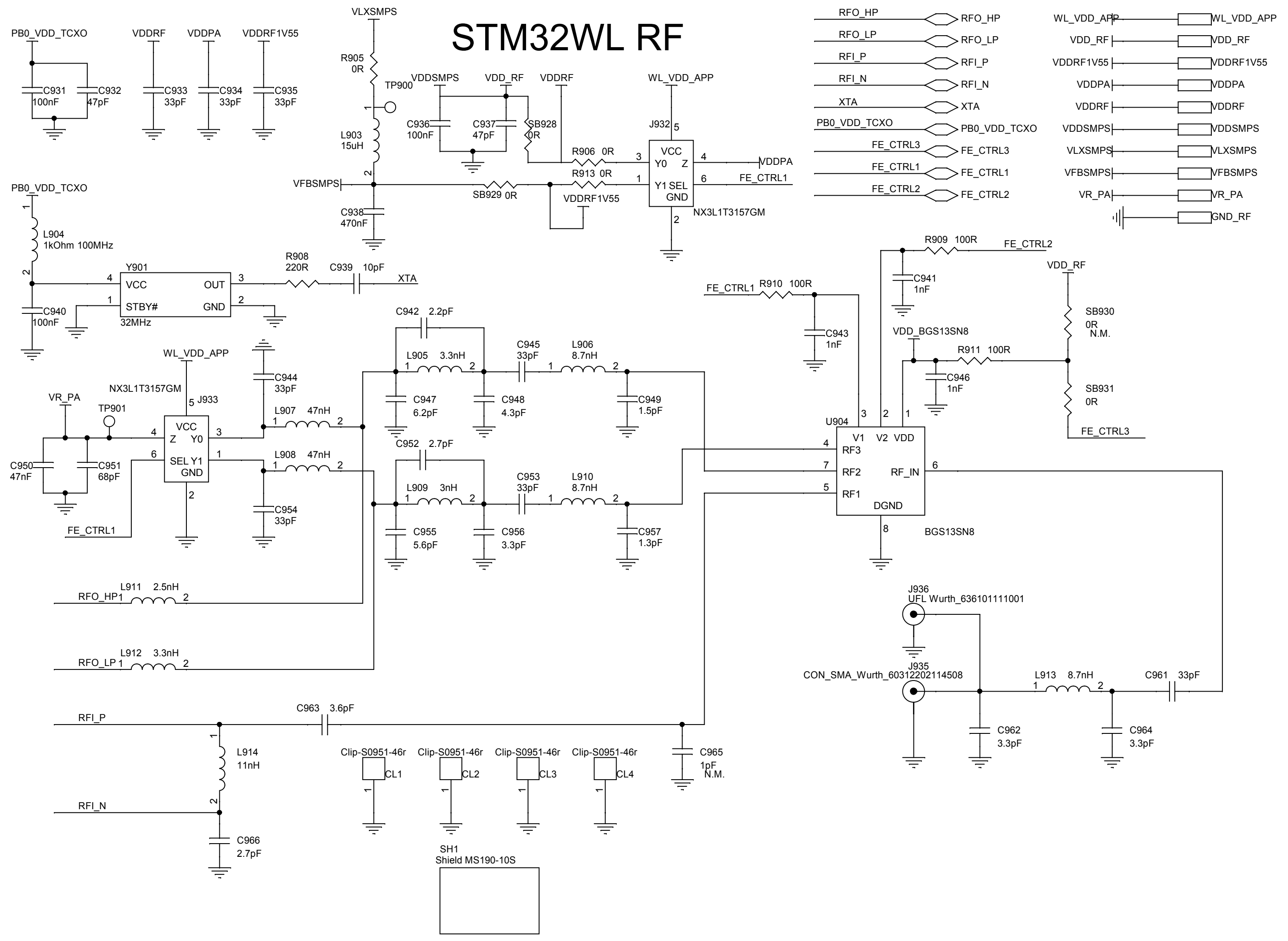


Figure 51. Expansion board (STEVAL-ASTRA1BC) circuit schematic (1 of 5)

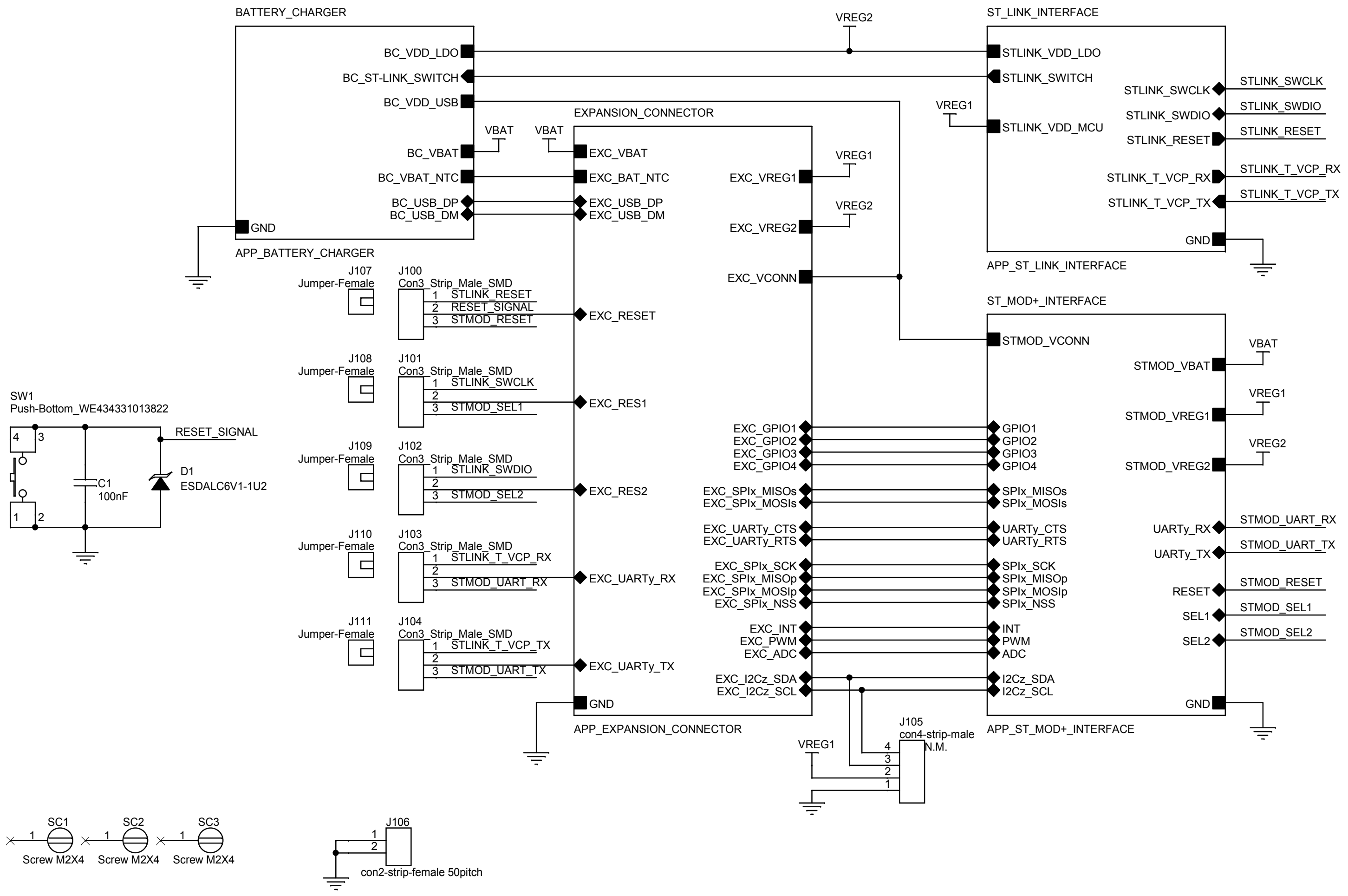


Figure 52. **Expansion board (STEVAL-ASTRA1BC) circuit schematic (2 of 5)**

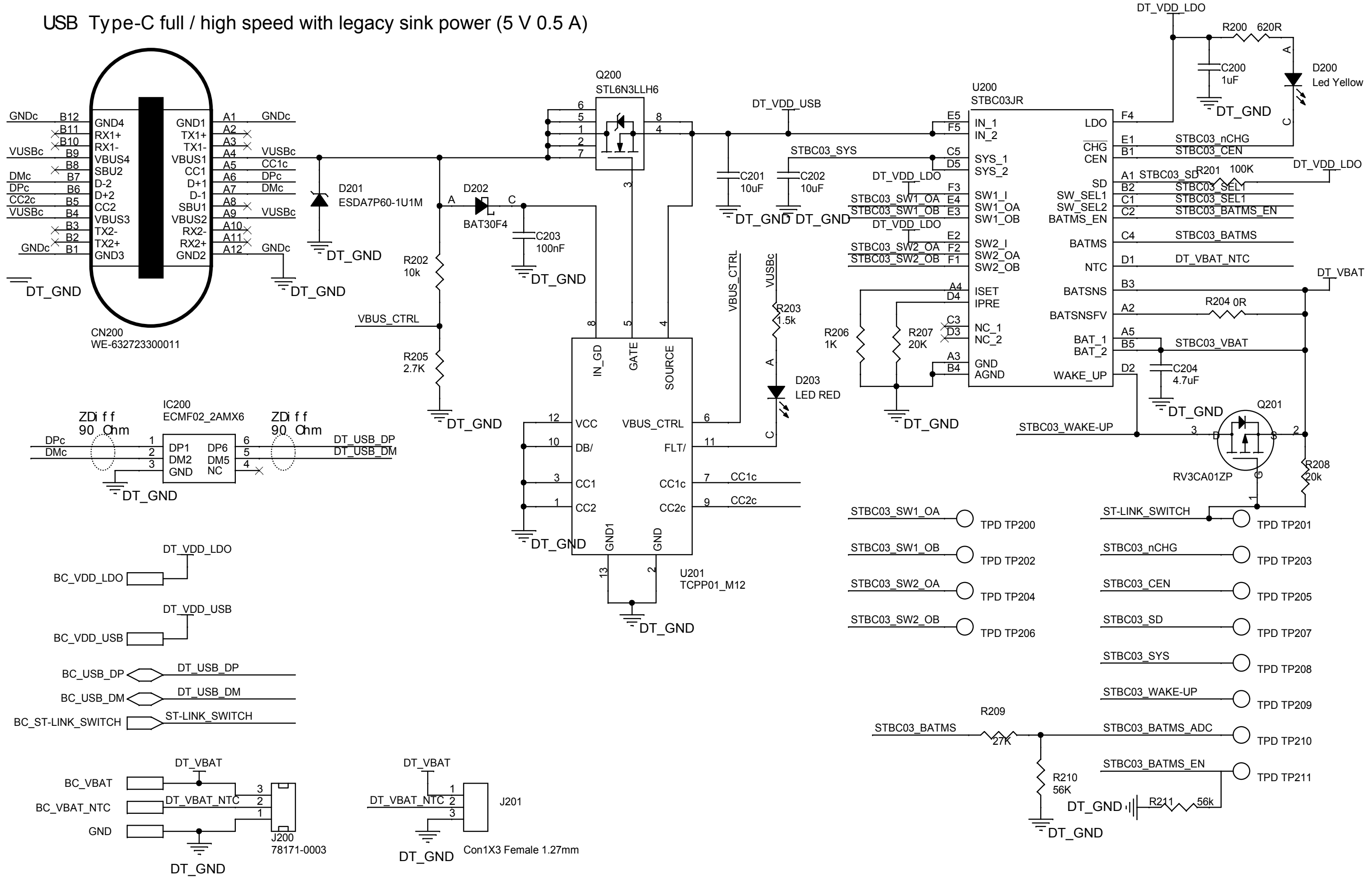


Figure 53. **Expansion board (STEVAL-ASTRA1BC) circuit schematic (3 of 5)**

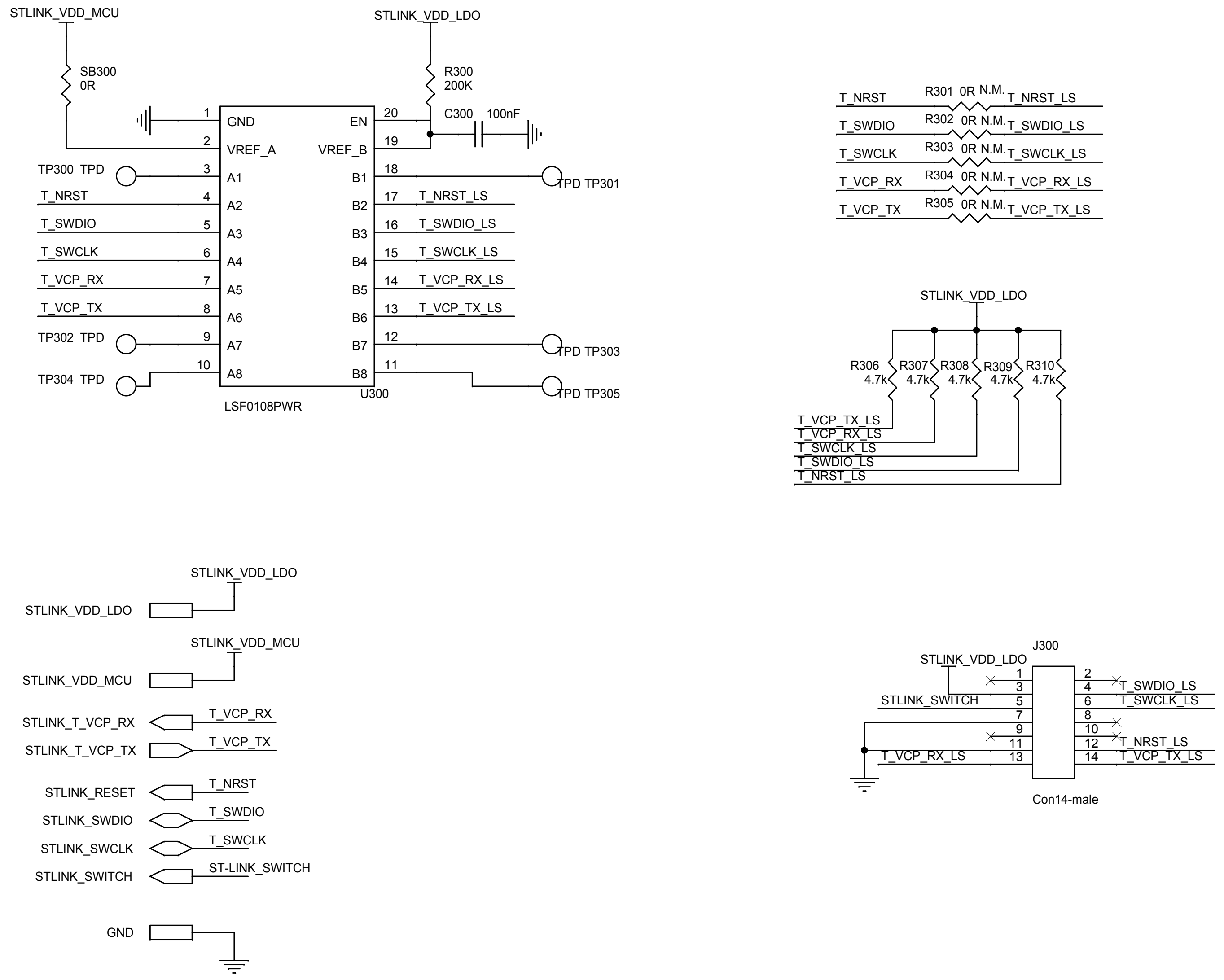
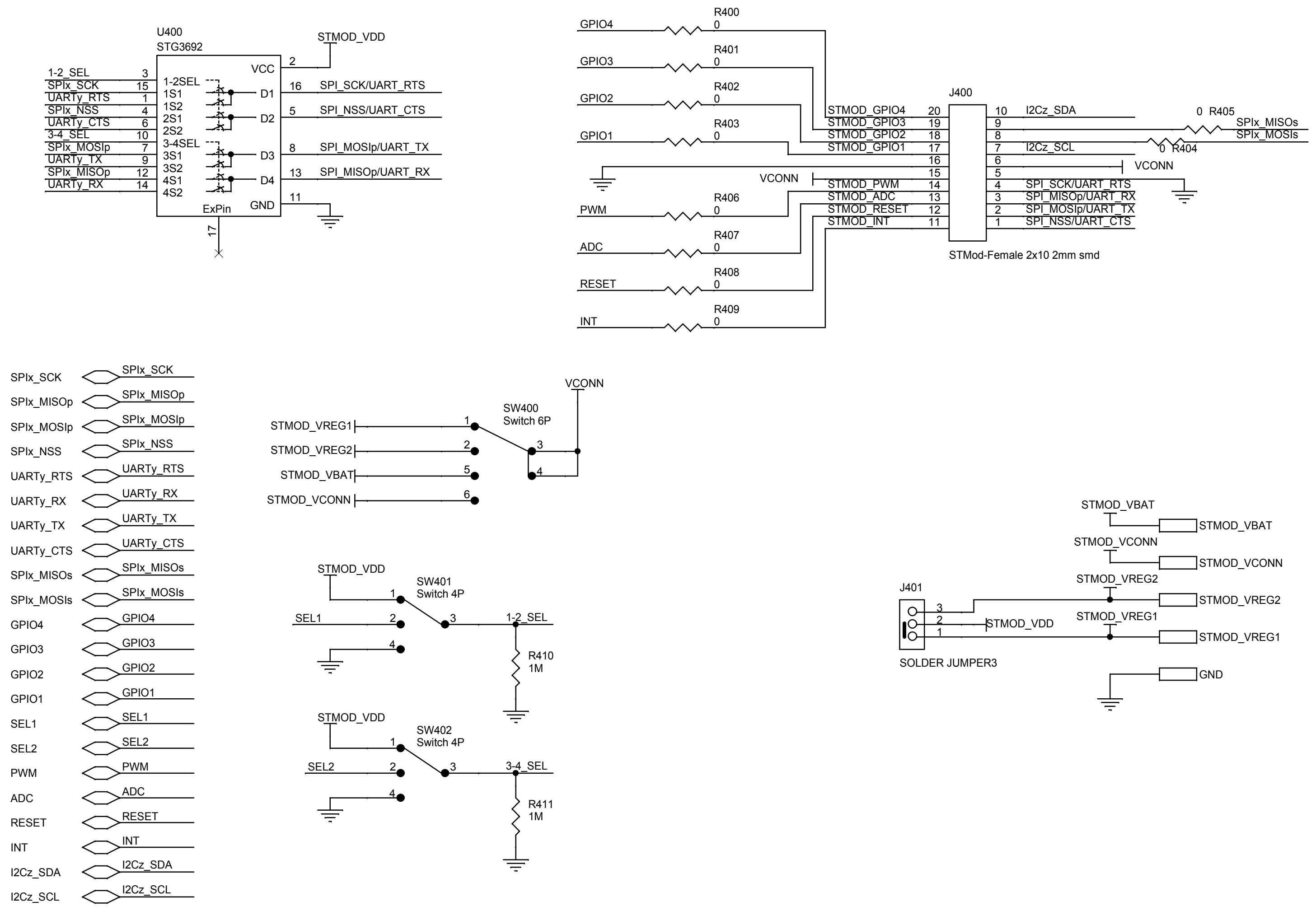


Figure 54. Expansion board (STEVAL-ASTRA1BC) circuit schematic (4 of 5)



AN6044

Figure 55. Expansion board (STEVAL-ASTRA1BC) circuit schematic (5 of 5)

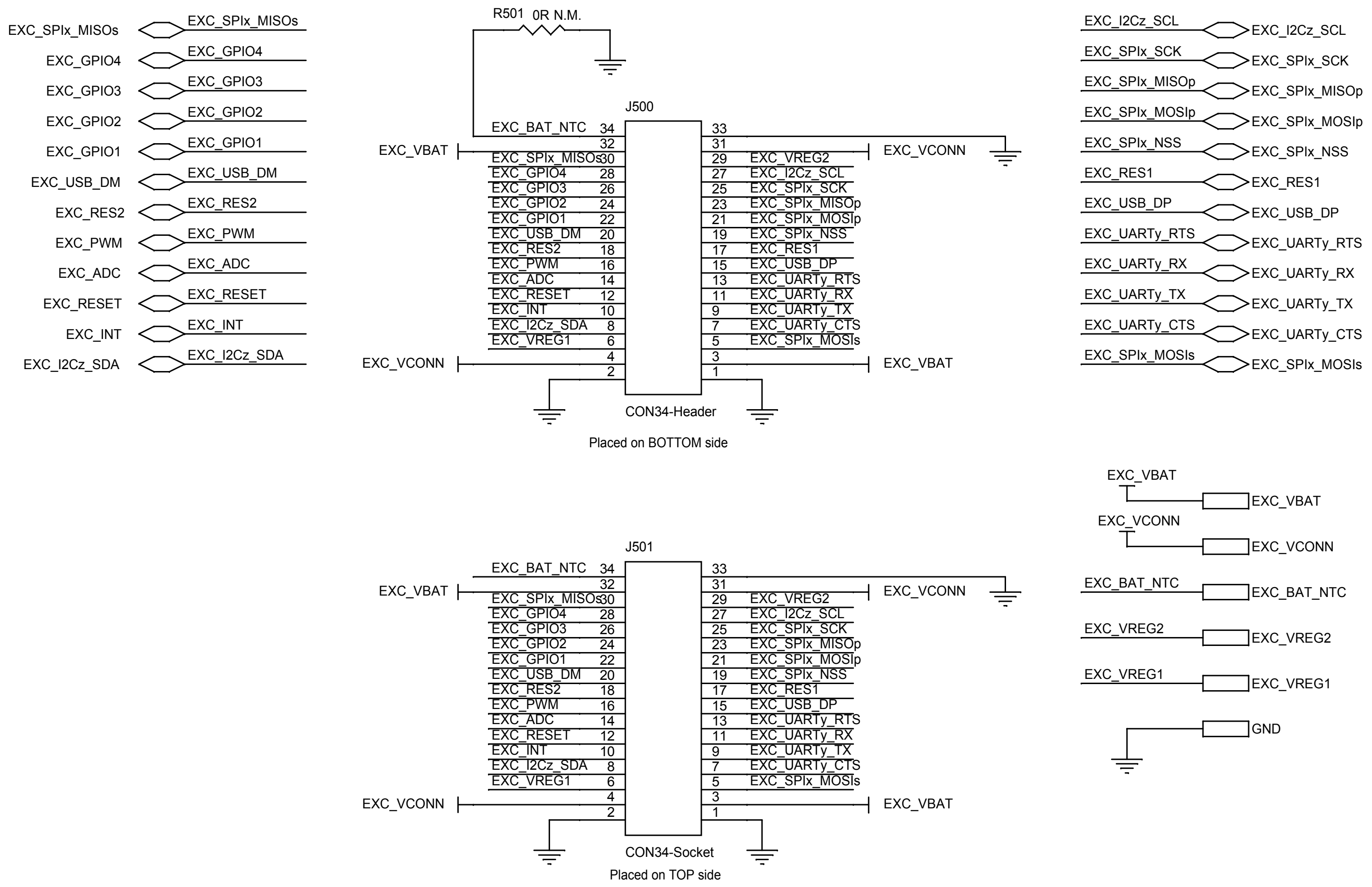
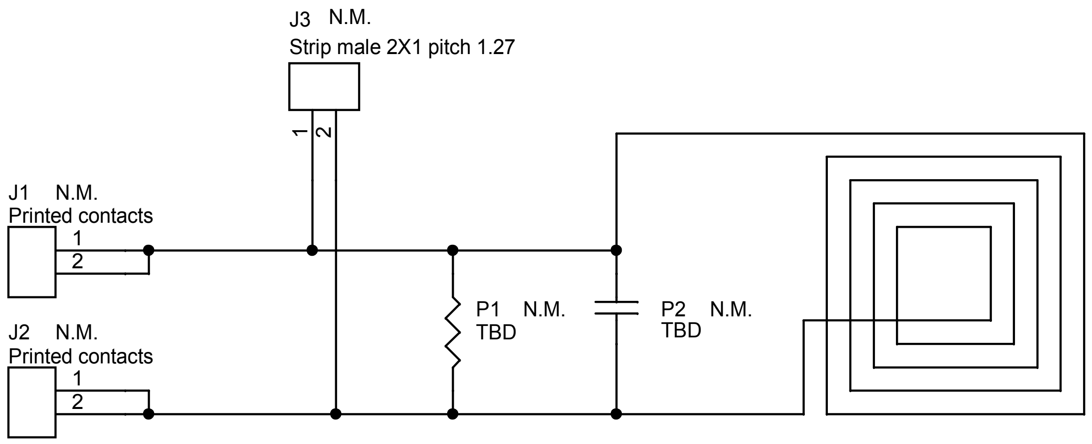


Figure 56. Flexible NFC antenna (STEVAL-ASTRA1NA) circuit schematic



Revision history

Table 1. Document revision history

Date	Version	Changes
07-Mar-2024	1	Initial release.

Contents

List of figures

List of tables

IMPORTANT NOTICE – READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgment.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, refer to www.st.com/trademarks. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2024 STMicroelectronics – All rights reserved