NFC passive lock implementation with NAC1080

About this document

Scope and purpose

This document describes Infineon's near-field communication (NFC) tag-side controller NAC1080 with the integrated H-bridge for passive smart lock applications, how to use it in circuit designs and how to start firmware development.

Intended audience

This document is intended for electrical engineers who want to develop or improve the electrical part of smart lock systems.

Table of contents

About this document . . . 1

Table of contents . . . 1

1 What is a smart lock . . . 2

2 How to design a passive smart lock system . . . 3

2.1 Single-chip solution . . . 3

2.2 Circuit design . . . 5

2.3 Energy balance . . . 8

2.4 Motor operation . . . 10

2.5 C_HB capacitor selection . . . 12

2.6 C_CB capacitor selection . . . 15

2.7 Antenna design . . . 17

3 Firmware development . . . 18

3.1 Memory . . . 18

3.2 ROM library . . . 19

3.3 Toolchain . . . 20

3.4 Smart lock firmware . . . 22

3.4.1 Operating the motor . . . 23

3.4.2 Determining the C_HB voltage . . . 25

3.4.3 Stepwise motor control . . . 26

3.5 Communication with a smartphone . . . 29

3.5.1 NAC1080 communication protocol basics . . . 30

Revision history . . . 37

Disclaimer . . . 38

1 What is a smart lock

An electronic smart lock is an electromechanical lock which can be opened or closed through wireless communication with an electronic key. Similarly to regular mechanical locks, smart locks have two parts – a lock and a key. The function of the key can be performed by a mobile device. For wireless communication between the key and the lock usually Bluetooth®, Wi-Fi or near-field communication (NFC) technology is used. A lock consists of a mechanical part (a mechanical lock itself) and a built-in electrical control part. The controller of the electrical part first completes the authentication from the key (mobile device) and receives the command to open or close the lock and after that it sends the command to a mini DC motor to move the lock secret. If the lock secret is unlocked, the lock can then be opened by turning, pushing or pulling it.

In battery-free (passive) smart locks (Figure 1), energy for supplying the electrical part of the lock can be harvested from the mobile device through the active NFC field and stored in the capacitor integrated in the lock system.

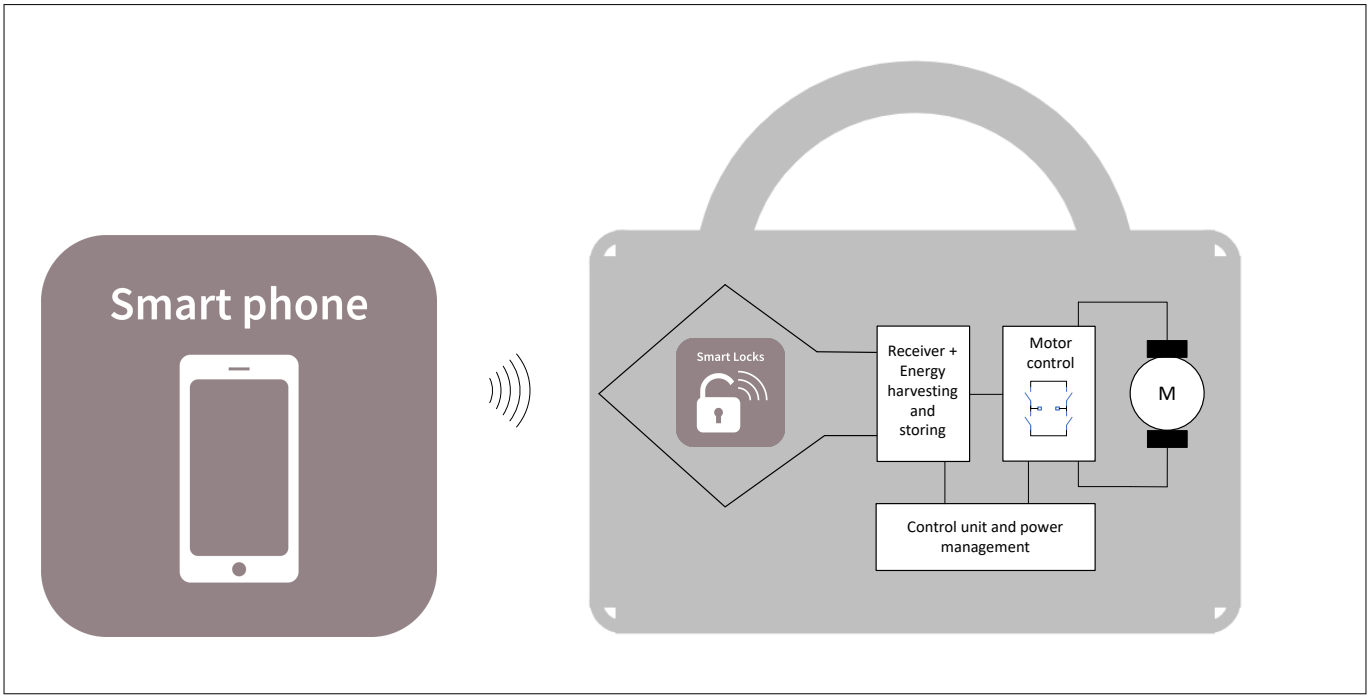

Figure 1 A simple passive smart lock system

Smart locks have two big advantages over traditional mechanical locks: no need to hold and carry physical keys, and fast and simple management of new keys. Furthermore, passive smart locks do not require battery maintenance and so are more cost-effective and reliable.

2 How to design a passive smart lock system

2.1 Single-chip solution

Figure 2 shows the block diagram of a simple passive smart lock system with the NFC interface. Here the functions of energy harvesting, NFC and motor control are all combined in one device – NAC1080. This is a NFC tag-side controller with an integrated H-bridge, or "smart actuator". This is an IC based on a programmable 32-bit Arm® Cortex®-M0 core operating at a CPU frequency of 28 MHz. Standard peripheral modules such as UART, SPI and 32 kHz RTC are included.

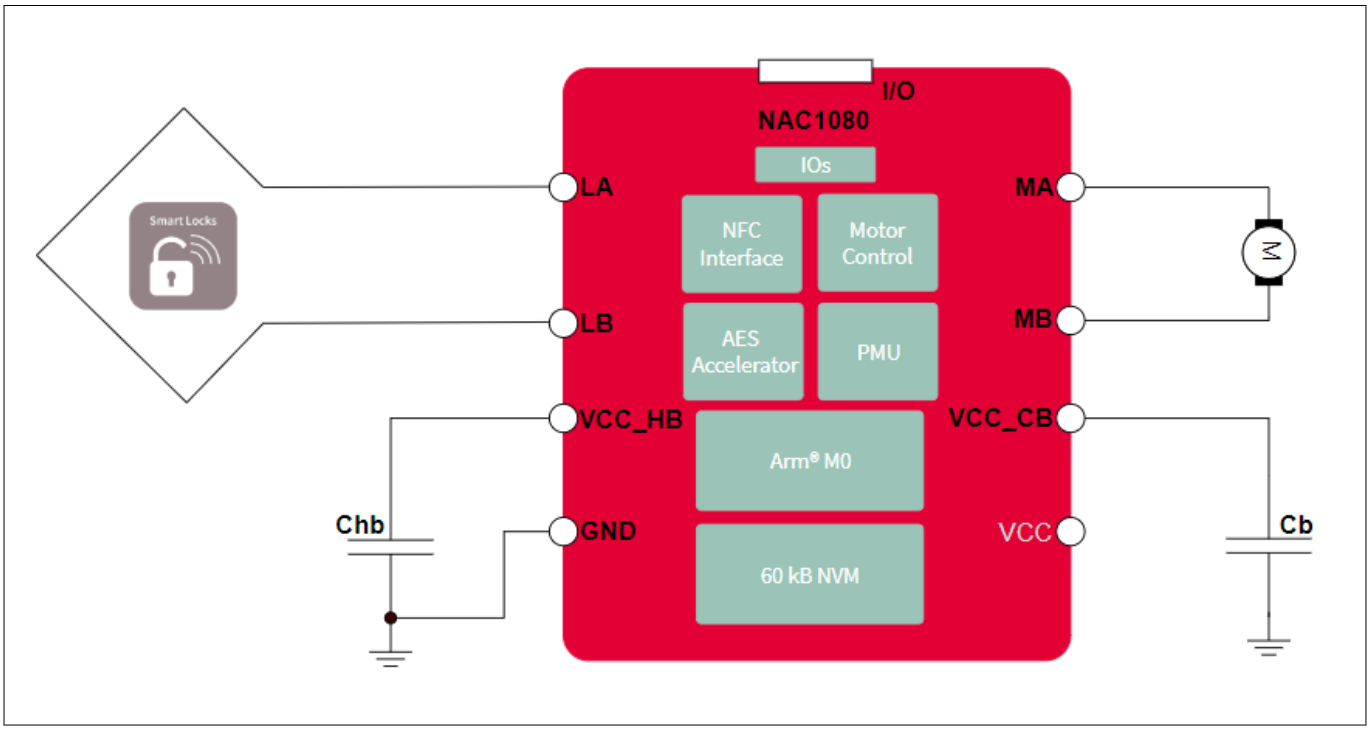

Figure 2 NAC1080 as a single-chip solution for passive smart lock systems

NAC1080 integrates NFC analog front end (AFE) with the power source function which provides two power source outputs with different supply priorities. The harvesting concept is based on a voltage-limiting characteristic with selectable clamping voltage (programmable: 3.0 V, 3.3 V or 3.6 V). The field strength-dependent antenna current is primary directed to the high priority power source output (VCC_CB). This power source supplies the CPU and peripherals of the device (except for the output stage of the H-bridge). A capacitor C_CB has to be connected to VCC_CB; the value of the capacitor depends on the load of NAC1080 (system power consumption except for the H-bridge) during the NFC modulation and is usually several µF. If the modules supplied from the high priority power source do not consume all the provided current, the supply voltage rises up to the clamping voltage and the excess current is delivered to the low priority power source output (VCC_HB). Figure 3 shows the startup diagram of both voltages. The capacitor C_HB stores the harvested energy of VCC_HB. This energy can be used to rotate the motor. The size of C_HB is selected according to the required energy budget of the application.

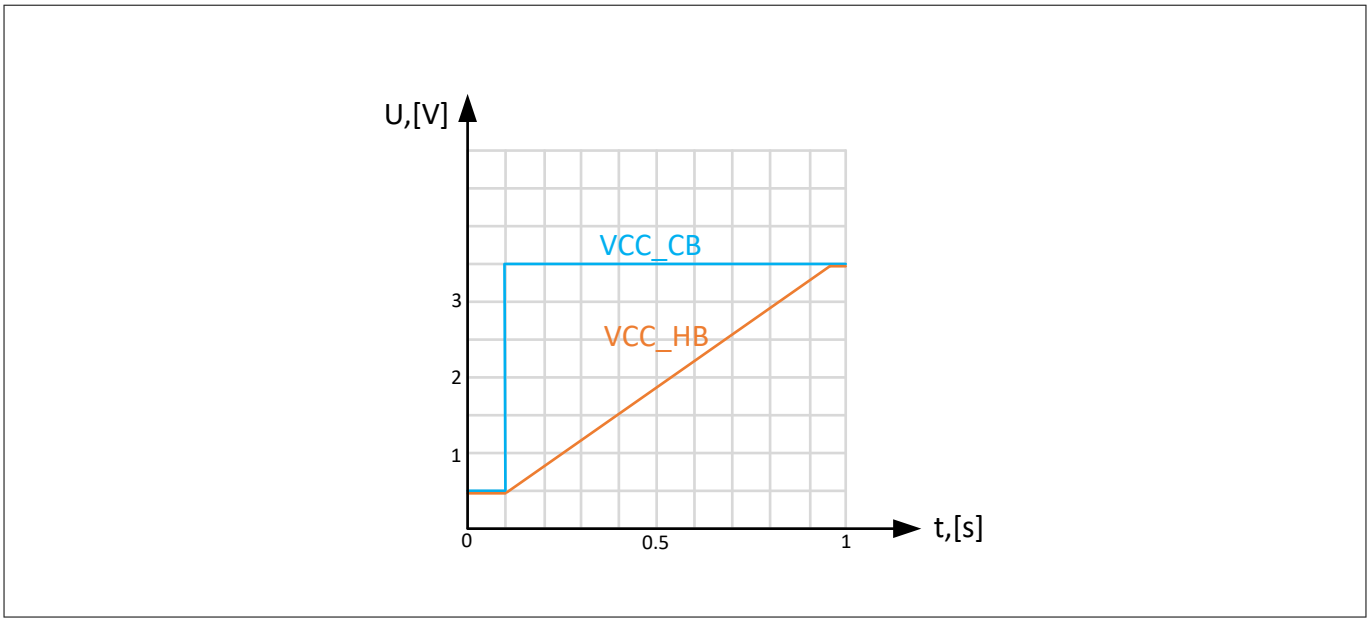

Figure 3 Smart actuator startup

The H-bridge (Figure 4) integrated into NAC1080 contains four power switches which can together deliver up to 250 mA current. All of the switches are directly accessible via the firmware. The H-bridge requires two power supplies: VCC_CB and VCC_HB. VCC_CB supplies the peripherals and the gate drivers. VDD_HB supplies the output stage and therefore the load.

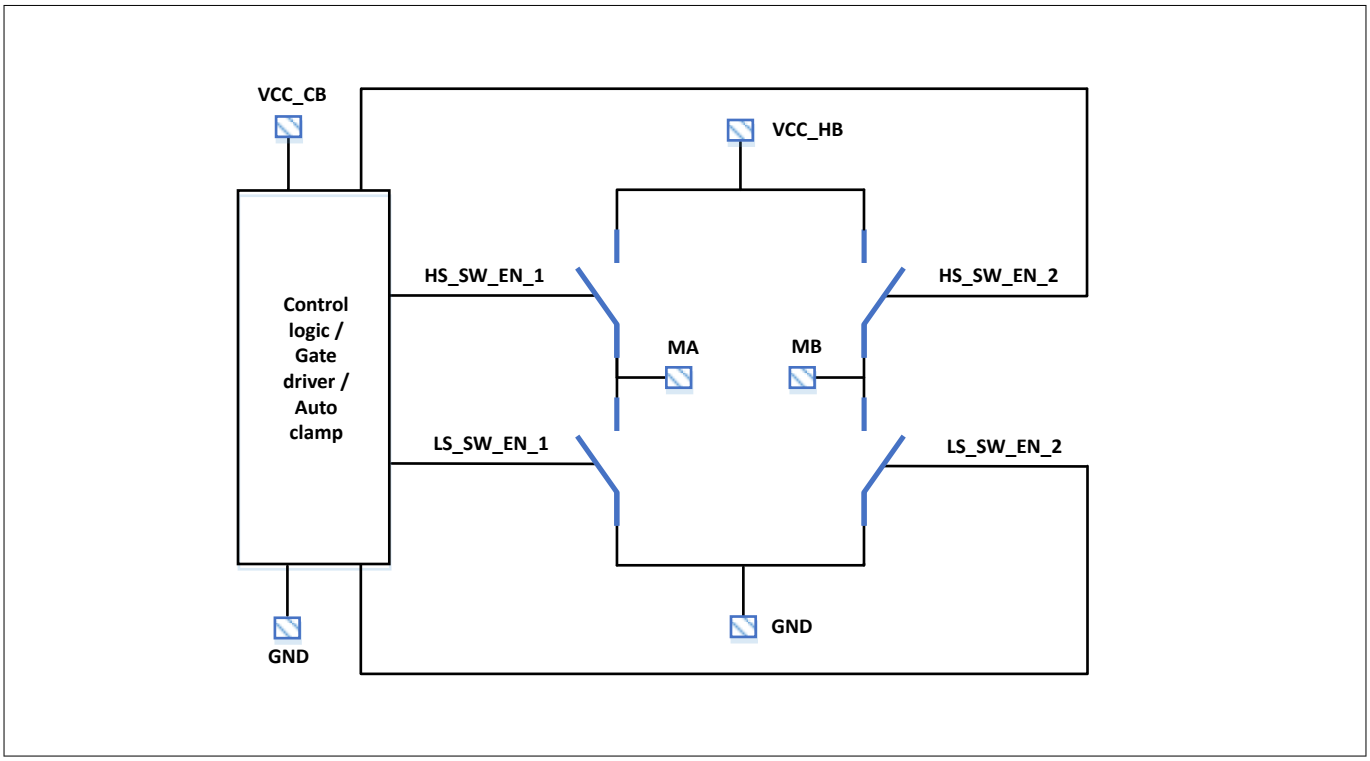

Figure 4 Simplified diagram of the integrated H-bridge

The smart actuator stops operating as soon as VCC_CB drops below the undervoltage lockout (UVLO). In fact, this means that either NAC1080 stops operating after the cell phone is removed or NAC1080 load (system power consumption except for the H-bridge) is too heavy during the reply to the reader request and the selected C_CB capacitor value is not big enough to back up voltage on the VCC_CB pin during this time. Section 2.6 describes how to select this capacitor.

2.2 Circuit design

A circuit diagram of an electrical part of a smart lock depends on the application profile. Let us consider two different application profiles:

- a cell phone (an electronic key) is used both to open the lock and to close the lock
- a cell phone (an electronic key) is used only to open the lock. The lock can be closed without a cell phone.

In the first case a mini DC motor must be rotated in only the forward or backward direction every time the lock has to be opened or closed. So the cell phone is needed every time to open or close the lock. For the second profile the motor must first be rotated in one direction to open the lock, and after the lock is opened - the motor has to be turned back to the initial position (i.e., immediately rotated in another direction). Then later to close the lock a cell phone is not needed - the lock can be closed by pressing/pushing it (depending on the mechanical construction).

Figure 5 shows the schematic of a simple smart lock control system suitable for the first application profile. In this case only a minimum number of external components are required together with NAC1080. Such a control system consists of a NAC1080, three capacitors and an antenna. C1 corresponds to the described C_CB, C2 corresponds to C_HB, and the selection of the values for both capacitors is described in sections 2.5 and 2.6. It is also practical to add a tuning capacitor to the schematic (C3) in case antenna impedance matching is needed. With the use of GPIOs of NAC1080, auxiliary sensors and indicators can be implemented in the system: for instance - an LED can be connected to the device to show the status of the lock (opened or closed) or to indicate the established NFC between the lock and the cell phone. To detect the position of the motor an optical sensor or a Hall sensor can be used. Please note that when using any sensors or indicators in such passive smart lock systems, the energy budget must be estimated correspondingly.

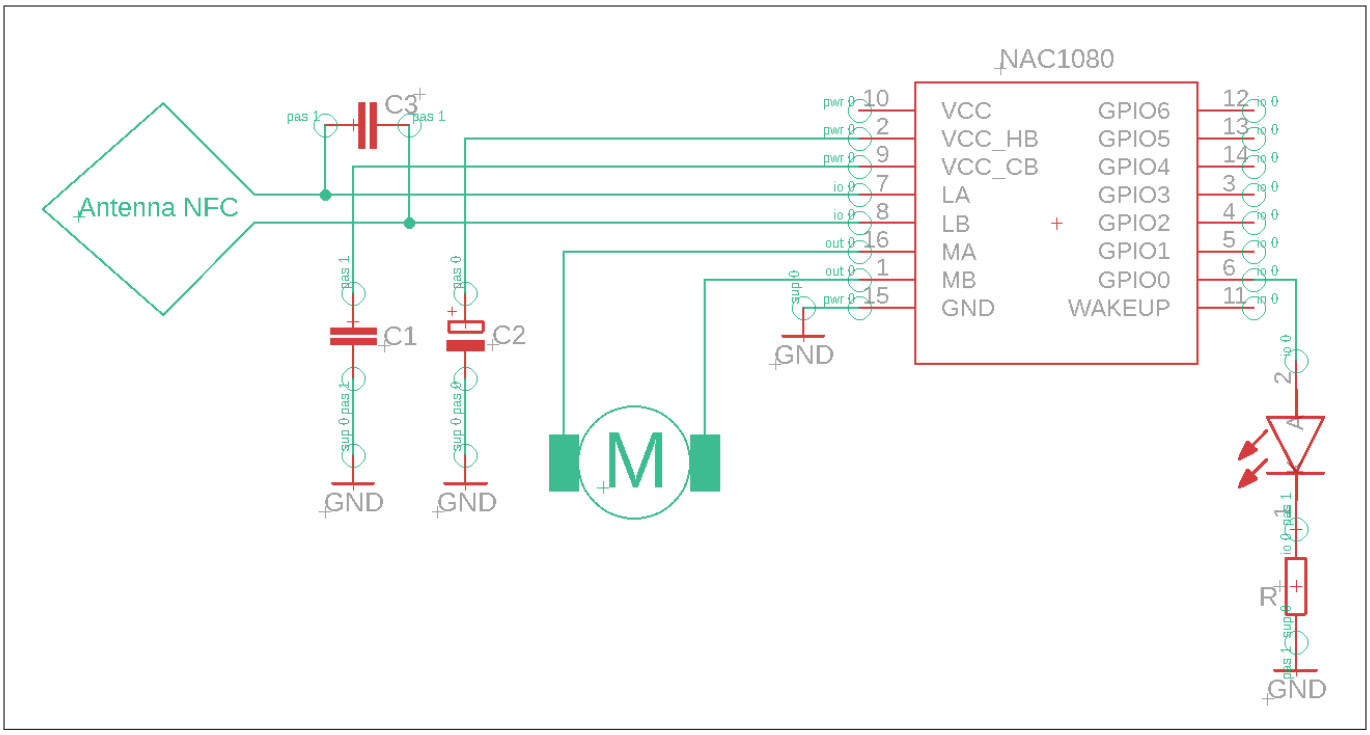

Figure 5 A simple smart lock control system

If it is necessary to extend the operating time of the smart actuator after the NFC field disappears (the cell phone is removed), the connection shown in Figure 6 can be implemented. Capacitor C4 helps to extend the operating time of the smart actuator after the NFC field is removed by providing energy to the VCC pin of NAC1080. Diode D is needed to hold VCC voltage at the configured clamping level during the NFC while the motor is rotating. D guarantees that the energy stored in C4 will not be consumed for the motor rotation but will only be used to supply the internal logic of NAC1080.

2 How to design a passive smart lock system

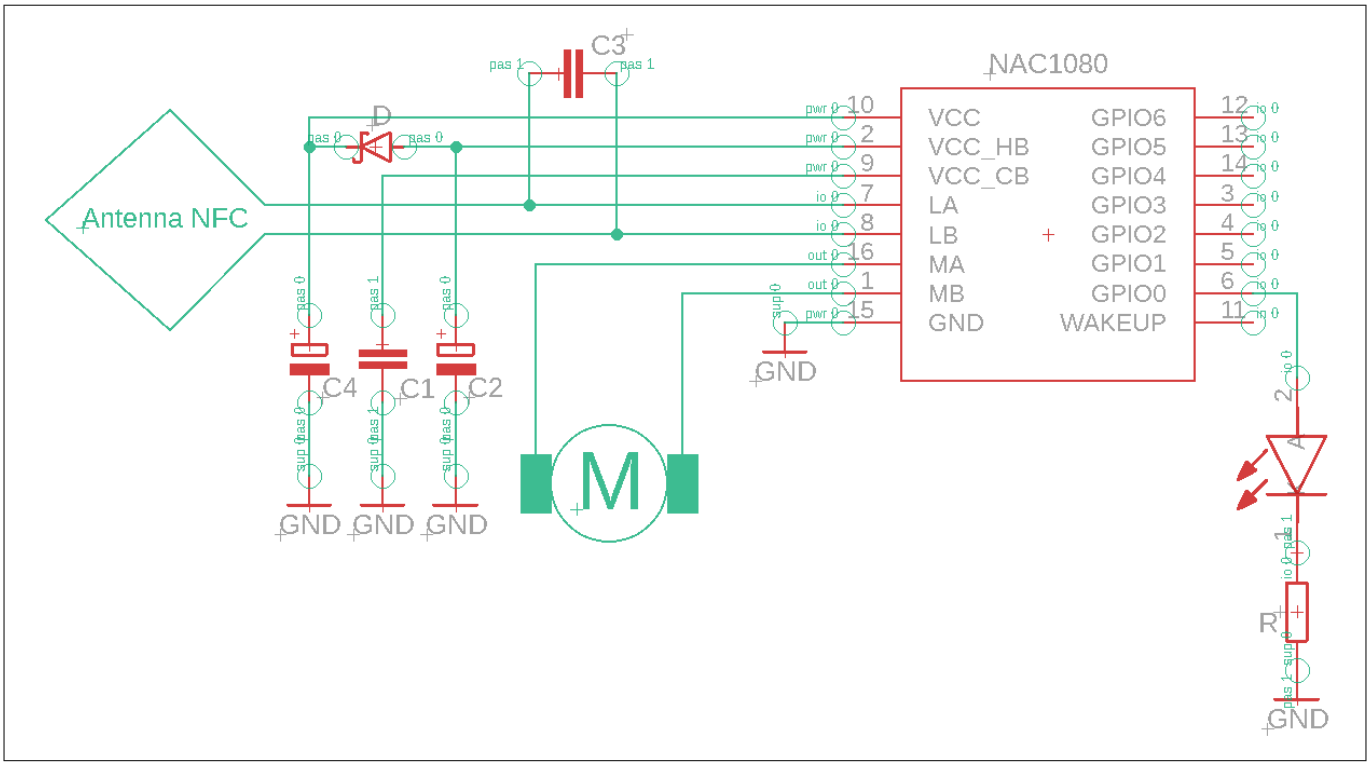

Figure 6 Smart lock control system with the backup energy storage

With the use of the second application profile, more energy can be stored at once compared to the first profile. This can be achieved by increasing the C_HB capacitor nominal voltage. As the nominal voltage of NAC1080 is 3.3 V the combination of boost and buck converters can be used to first charge the energy storage capacitor to the higher voltage (for instance, to 15 V) with the help of a boost converter and then to reduce the voltage back to the nominal 3.3 V with the use of a buck converter - to get the nominal voltage on the VCC_HB pin.

Figure 7 shows the circuit diagram of smart actuator U1 together with step-up and step-down conversion. GPIO outputs are used to enable/disable the converters as well as to control the step up conversion with the PWM signal (GPIO1). During C4 capacitor charging buck converter U2 has to be disabled to not slow down the charging speed. After capacitor C4 is charged to the target voltage, the stored energy can be used to rotate the motor. The U2 buck converter generates 3.3 V, which is required to supply the H-bridge. The boost circuit must be disabled during motor rotation.

2 How to design a passive smart lock system

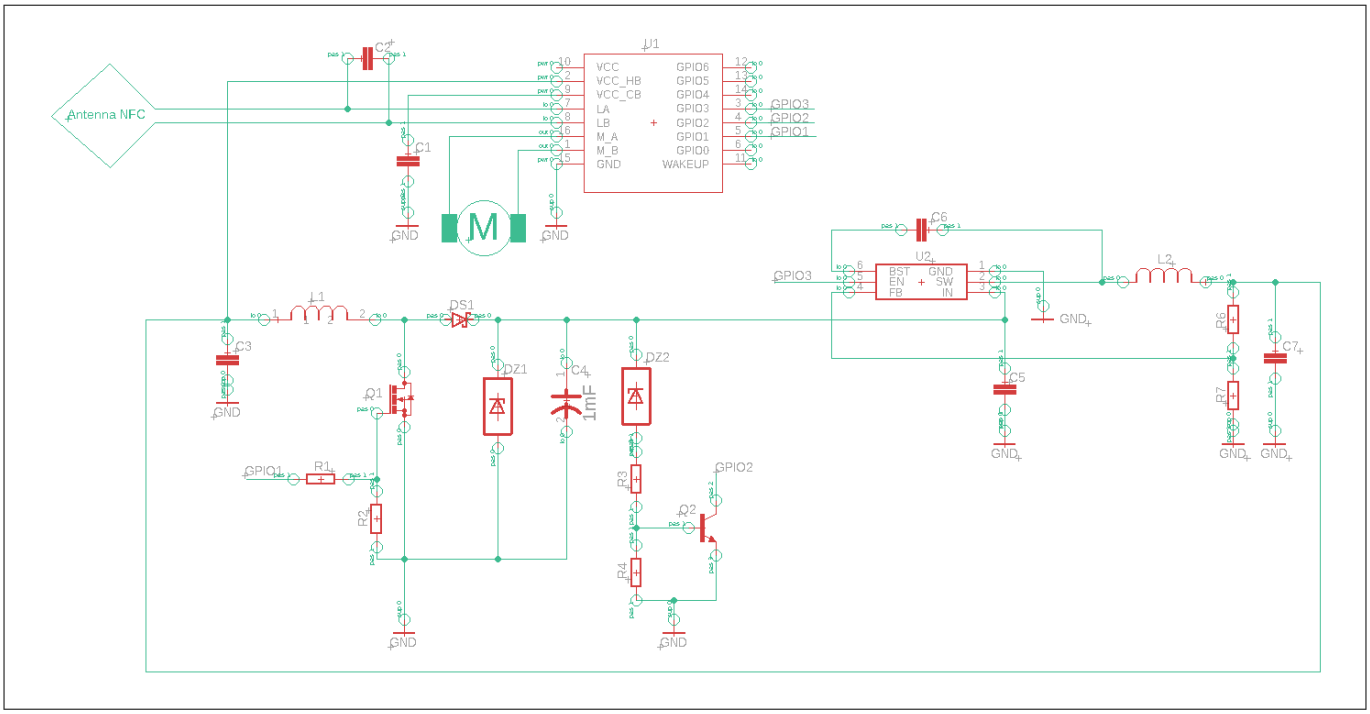

Figure 7 Circuit diagram with step-up and step-down conversion

Step-up conversion is implemented with the discrete components that could reduce the bill of materials (BOM) compared to the cost of a single buck IC. The pulse-width modulation (PWM) control signal for the MOSFET Q1 is generated on the GPIO1 output. Zener diode DZ1 limits the boost output voltage to guarantee that the voltage will not exceed the target voltage. The detection circuit based on Zener diode DZ2, resistors R3/R4 and NPN transistor Q2 generates logic low level on the GPIO2 line as soon as capacitor C4 is charged to the voltage defined by the detection circuit. GPIO2 has to be configured as input with the internal pull-up resistor (see the NAC1080 Software Development Guide document for the configuration of GPIOs). To charge C4 to the target voltage with the maximal speed t_on and t_off timings of the PWM control signal on GPIO1 have to be selected optimally (experimentally or by simulating the circuit). The recommended selection range for the switching frequency is 15 kHz to 80 kHz and the example of the timing combination is t_on = 7 µs, t_off = 50 µs. When choosing MOSFET Q1 it is important to select one with a $V_{GS(th)\max}$ below the minimum high level of GPIO (i.e., below 2.2 V) to guarantee the conducting path between the source and drain terminals at the on-times.

2.3 Energy balance

Figure 8 shows energy transfer in a passive smart lock system: during NFC energy is continuously harvested from the cell phone and charged to capacitor C_HB, which is the power supply source for the motor driver.

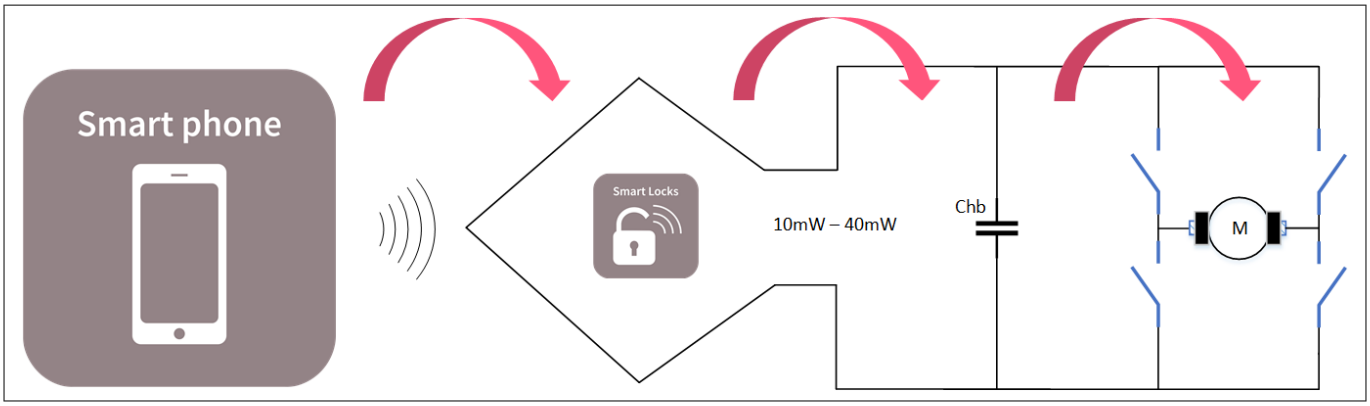

Figure 8 Energy transfer in a passive smart lock system

To enable a smart lock system the following energy balance has to be fulfilled:

$E_{MCU} + E_{el} < E_{harv}$

where: E_{MCU} - energy needed to supply the electrical control system built in the lock (NAC1080 IC with its surrounding),

$E_{el} = \frac{E_{mech}}{\eta}$,

E_{el} - electrical energy needed to turn the motor and to move the lock secret,

E_{mech} - mechanical energy needed to turn the motor and move the lock secret,

η - electrical-to-mechanical power transformation efficiency,

E_{harv} - energy harvested from NFC field.

Part of the harvested energy is used to supply NAC1080 IC - high priority supply.

The harvested energy E_{harv_hb} which will be stored in the capacitor (second priority supply) is calculated according to the formula:

$$E_{harv\_hb} = \frac{C \cdot U^2}{2}$$

where: *C* - energy storage capacitor, [F]

U - voltage across the storage capacitor, [V]; for NAC1080 *U* can be 3.0 V, 3.3 V or 3.6 V

For instance for 1mF capacitor and voltage 3.3 V the stored energy is 5.4 mJ.

The efficiency of energy harvesting affects the speed of the capacitor charge and depends on the coupling coefficient between the antennas of the transmitter and receiver (i.e., on the distance between antennas and antenna sizes), on antennas tuning (antenna impedance matching) and on the NFC field strength, which is a result of NFC transmitter radiation.

The delivered NFC power is different for different mobile phone models and vendors. The power *P* on the receiver (lock) side can be estimated indirectly by measuring the time *t* of C_HB charging:

$$P = \frac{C_{HB} \cdot U^2}{2 \cdot t}$$

According to the lab measurements taken at Infineon with different typical phones and different tag antenna designs the measured values are tens of mW (between 10 mW and 40 mW).

2 How to design a passive smart lock system

As NFC power is limited, the power consumption is critical for passive smart lock systems. For that reason the smart actuator is designed to support different power modes – an operation mode and a power-saving mode. In operation mode the chip is fully functional; all necessary functional blocks are powered. In power-saving mode only a minimum of functions are available. The device is in a kind of standby mode and waiting for wake-up events issued by wake-up sources to enter the operating mode. Since reduction of current consumption is essential in power-saving mode, unused parts of the chip can be disconnected from the power supply and switched off. The transitions between the modes are described in the NAC1080 datasheet.

2.4 Motor operation

In smart lock applications typically low-cost mini DC motors with gear boxes are used. As the power is limited in such systems, a motor should be rated for the low current to operate. The feature of using motors in smart locks is that during opening or closing the lock there is no need to permanently run the motor at the settled speed, but the motor has to be run-up and turned by a required angle only when the state of the lock (on or off) needs to be changed. Depending on the construction of the lock, the mechanical load can be connected to the motor right at the start or when the motor has already started rotation.

The operating point of the motor is shifted from the maximum efficiency/nominal speed to the higher torque/ lower speed direction (Figure 9). Therefore motor parameter such as stall torque are among the key parameters for motor selection. To enable opening or closing the lock the stall torque of the selected DC motor must exceed the torque required to move the lock secret.

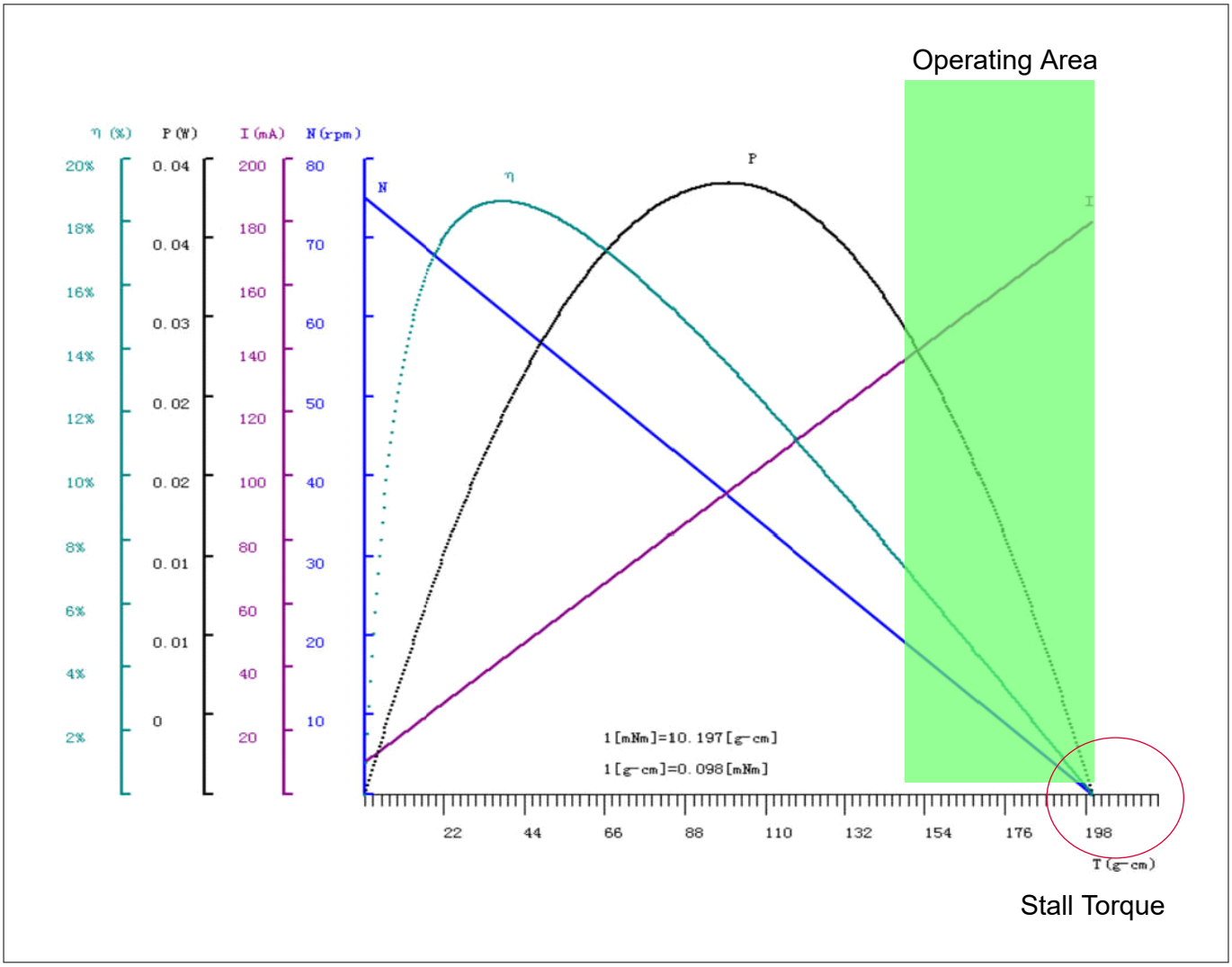

Figure 9 Mini DC motor operating area for a smart lock system

To fulfill the requirements of the H-bridge of NAC1080 the datasheet parameters of the H-bridge must be considered. The maximum operating voltage of the motor is limited by the selected clamping voltage of the smart actuator (3.0 V, 3.3 V or 3.6 V).

A table of the parameters of a mini DC motor suitable for NAC1080 and implemented in one of the passive smart locks examples is given below:

Table 1 Parameters of a mini DC motor suitable for NAC1080

Parameter	Value
Type	DC with gear, gear ratio 1:171
Rated voltage	3.0 V DC
Rotation	CCW
Output speed	75 ±5 RPM
No load current	0.02 A max.
Stall current	0.26 A max.
Stall torque	160 g.cm min.
Rated torque	38 ±5 g.cm
Rated current	0.06 A max.
Rated speed	61 ±10 percent RPM
Shaft diameter	8 mm

2.5 C_HB capacitor selection

To open or close the lock a mini DC motor must be rotated a certain angle. The rotation angle depends on the mechanical construction of the lock. Energy needed to rotate the motor is stored in the capacitor C_HB connected to the VCC_HB pin of NAC1080. To rotate the motor by a required angle different approaches can be taken – either all of the required energy is stored in the capacitor at once, or a stepwise (multistep) motor movement can be used.

For the first method, either capacitance has to be large enough or the nominal capacitor voltage has to be high enough - to store sufficient energy for the target rotation angle to move the motor by one step: $E = \frac{C \cdot U^2}{2}$.

Figure 10 shows an oscilloscope diagram of a simple one-step movement: first – charging of the capacitor to the nominal 3.3 V, then – motor rotation (discharging of the capacitor).

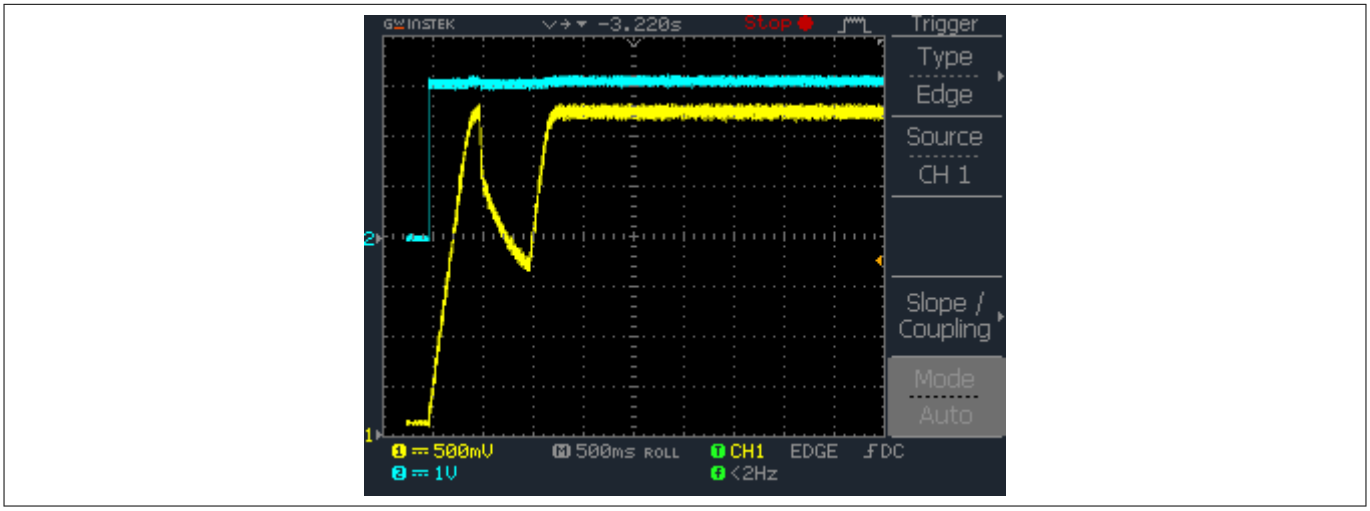

Figure 10 Example of a scope diagram of one-step motor rotation for 180 degrees

A further increase in the stored energy can be achieved by using an energy storage capacitor with a higher nominal voltage. To implement this approach, step-up and step-down converters are required, so we can refer to it as one-step movement with external boost-buck circuit.

Figure 11 shows a scope plot of this method. Ch2 shows the charging and discharging cycle of the VCC_HB energy storage capacitor. The capacitor is charged up to 15 V from the NFC field. During the discharge time the mini-motor first rotates forward and then – backward. Ch1 shows a detection signal, which goes low as soon as VCC_CB voltage reaches the target level defined by the detection circuit. Such a signal can be used to trigger an event, as in our example to run the motor. This method allows us to store more energy compared to the simple one-step movement using the same capacitance.

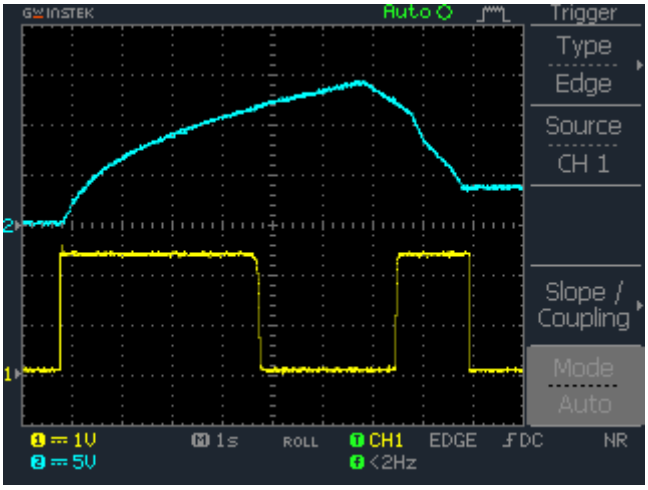

Figure 11 Scope diagram of an advanced one-step motor movement

For stepwise (multistep) motor movement it is not necessary to store all of the energy needed to rotate the motor by a required angle in one step. The motor is driven with on/off cycles: during the off-time the capacitor is only charging, during the on-time the capacitor keeps charging, the energy stored during the off-cycle is used to move the motor, and in general discharging is ahead of charging. For this method a start voltage and a stop voltage must be assigned. Start_voltage is a voltage across the capacitor at which level the on-cycle starts, that is the motor is on. Stop_voltage is the voltage across the capacitor at which level the off-cycle starts; that is, the motor is turned off and re-charging of the capacitor starts. This method helps to reduce the capacitance of C_HB that can result in cost and space savings.

Figure 12 shows examples of stepwise motor rotation through 180 degrees with the use of 1mF and 680 µF respectively. The total number of steps to reach the target angle is three for 1mF and four for 680 µF. The total rotation time is similar and is around 4 s.

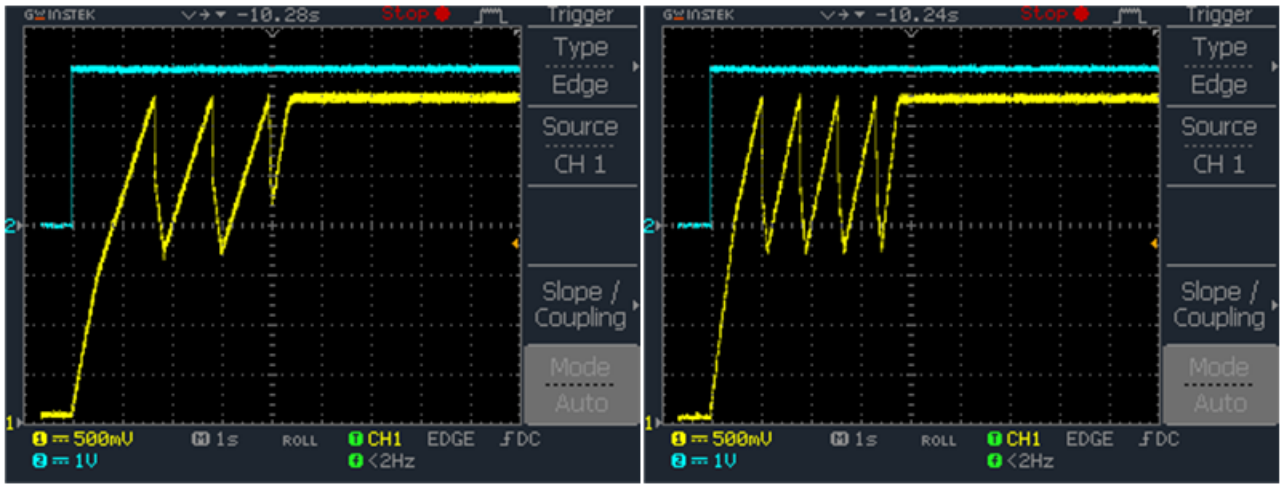

Figure 12 Scope diagrams of stepwise motor rotation at different capacitor values

The optimal capacitor value can be found experimentally for the selected motor. The capacitor value should be selected so that the stored energy is sufficient to start the selected DC motor under load. When decreasing a capacitor there will be a point where either the stored energy is not sufficient to enable motor start, or it will take longer to turn the motor to the target angle. Figure 13 shows an example of stepwise motor rotation through 180 degrees with the use of 220 µF. The total rotation time is around 5 s. From this example it can be seen that at 220 µF the motor can still start, but the total rotation time and the number of cycles are not optimal.

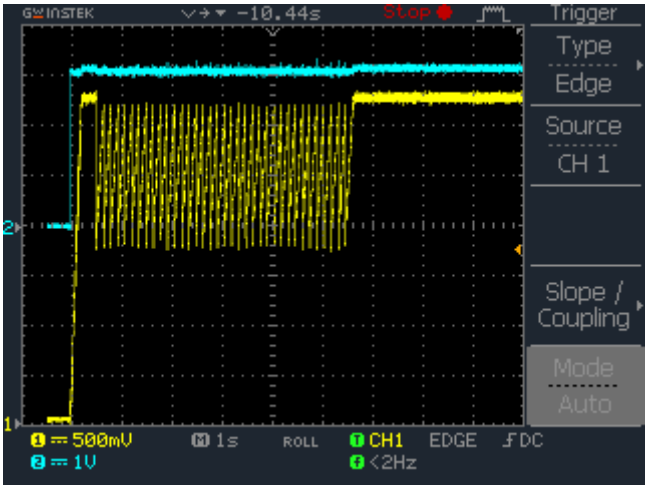

Figure 13 Scope diagrams of stepwise motor rotation at 220 µF

2.6 C_CB capacitor selection

The value of the C_CB capacitor depends on NAC1080 load (IC power consumption) during the NFC modulation and must fulfill two conditions:

- at any disturbance (drop) on the VCC_CB pin during the NFC VCC_CB voltage should be above UVLO
- start-up time of NAC1080 should not exceed 5 ms - to fulfill NFC protocol requirements (guard time).

Fulfillment of the first condition helps to avoid unwanted NAC1080 resets, which happen due to disturbances on the VCC_CB pin. Such drops can happen because while answering the reader in NFC, energy harvesting is not possible. If the capacitor size is not sufficient to backup the energy required to power the IC while answering the reader, there will be voltage drop on VCC_CB. If this drop is below UVLO then reset happens, as shown in Figure 14. With the correct value of VCC_CB 2 µF such behavior is eliminated, as shown in Figure 15. 2 µF is not a fixed value for the VCC_CB pin, as it depends on NAC1080 load during NFC modulation.

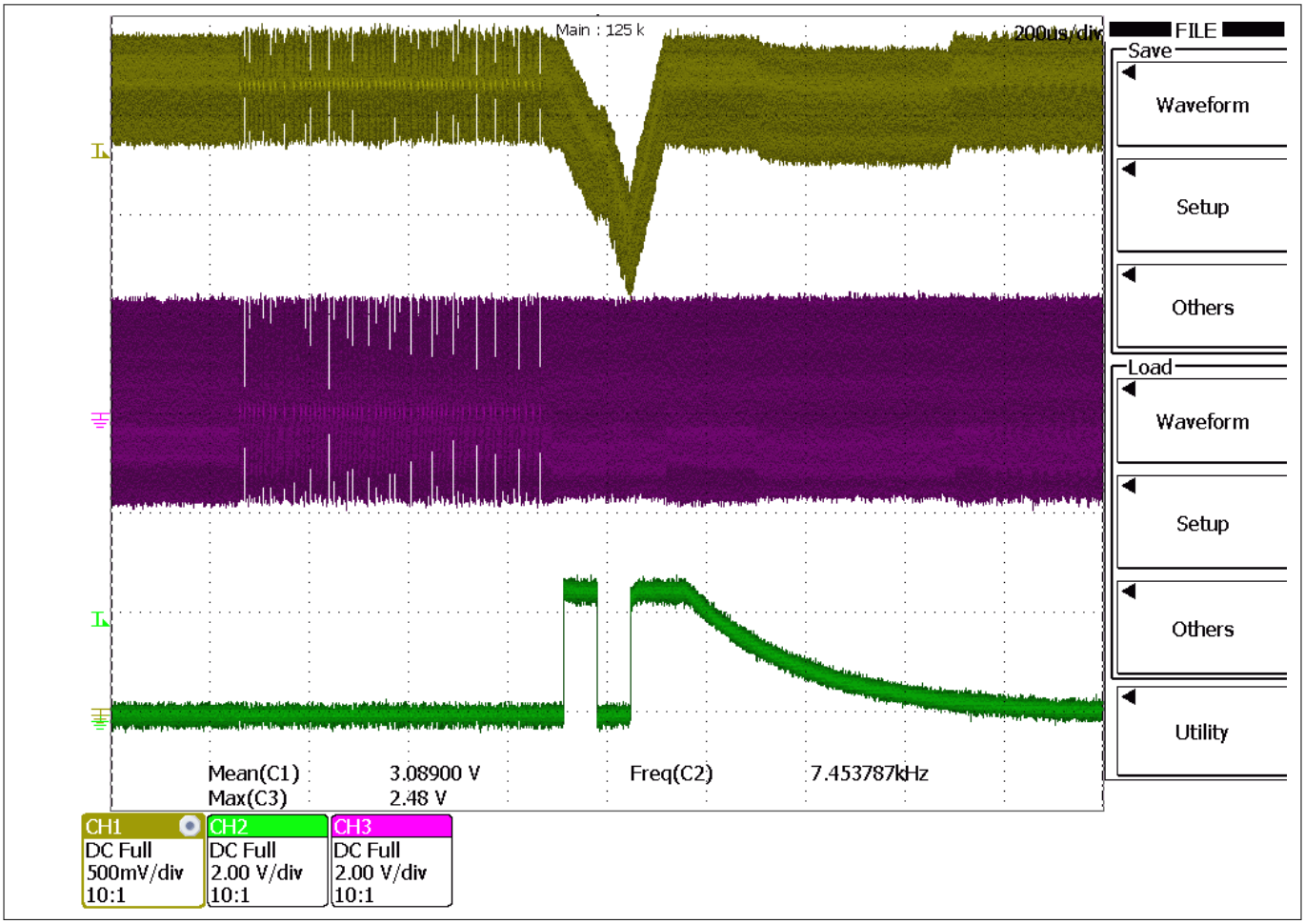

Figure 14 Scope diagram of the experiment at C_CB = 220 nF

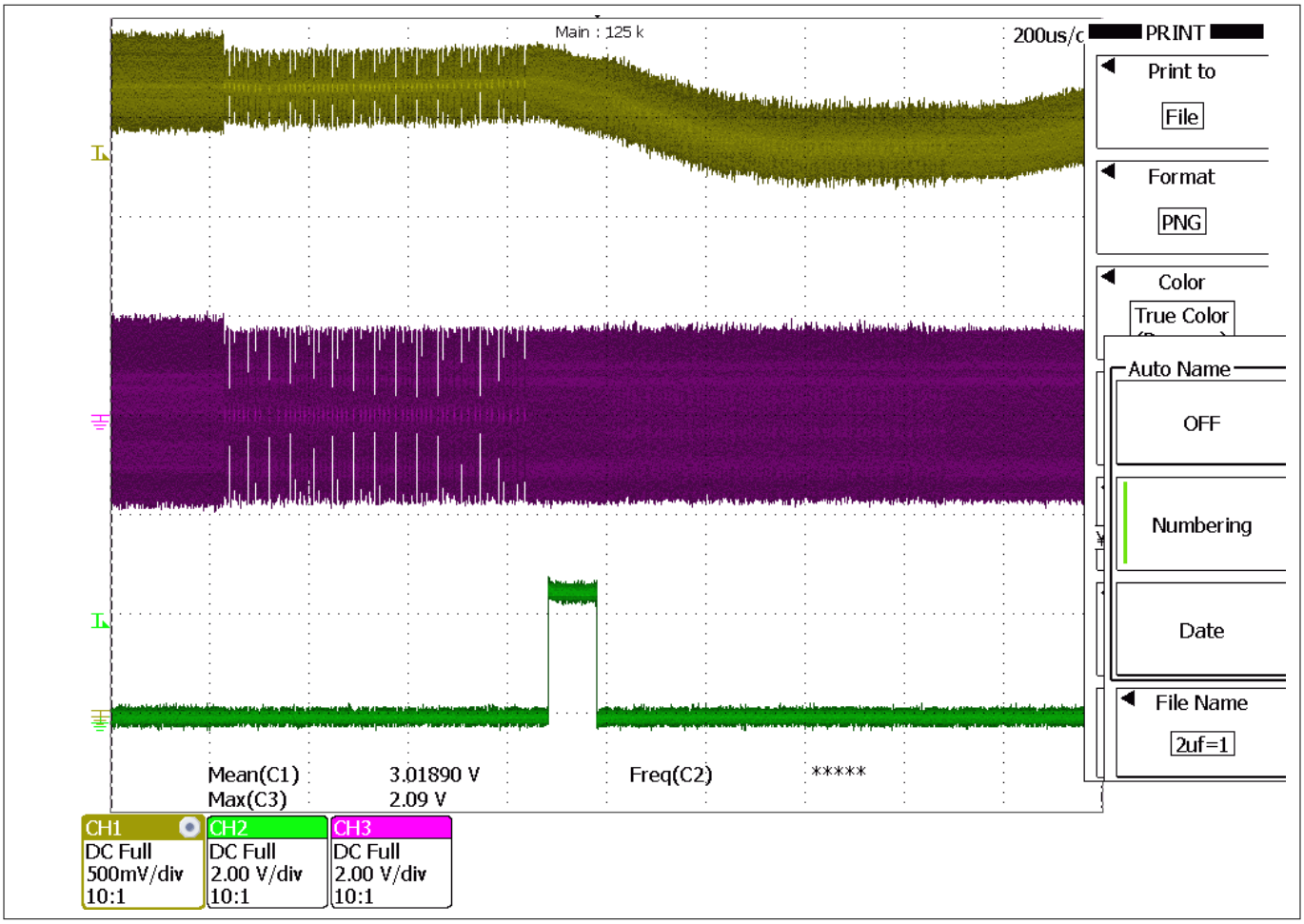

Figure 15 Scope diagram of the experiment at C_CB = 2 µF

The second condition requires consideration of the guard time value. This is needed to enable the IC to answer to the first polling from the reader in case the first polling can be recognized by NAC1080 (i.e., it fulfills IEC 14443 Type A). The start-up time of NAC1080 is defined by the C_CB capacitor charging time + NA10080 hardware initialization time + NAC1080 firmware initialization time. The last two timings are estimated as 1 ms + 0.5 ms margin, so the capacitor charging time should not exceed 3.5 ms.

2.7 Antenna design

The basic principle of NFC antenna design is to create a circuit resonating at 13.56 MHz with an inductance (antenna) and an internal or external tuning capacitor. This is because NFC readers operate in the 13.56 MHz high-frequency band.

An antenna acts as an inductor. The tuning capacitor can be either only internal or only external, or both. A typical value of the internal capacity between terminals LA and LB of NAC1080 is given in the datasheet, and is 23.5 pF. If, due to space requirements or antenna shape, an external capacitor is used in addition, this must also be considered in the calculation. This is necessary for example, when the inductance of the antenna cannot be achieved without an external tuning capacitor caused by small antenna size.

Some antenna design tips:

- An antenna that is too small would “waste” magnetic flux due to the small area within the coil. An antenna coil that is too big does not get flux around the outside of the windings. So the shape or size of antenna coil should fit the geometry of the reader’s antenna (in this case, it is the NFC antenna in the cell phone/Feig reader).
- Tuning the tag to have only 13.56 MHz is not the correct way. Once the tag and the reader are coupled the resonance frequency typically lowers. This is because of the coupling. As a result, the tag antenna resonance circuit must be tuned to a frequency slightly higher than 13.56 MHz. We usually tune our tag resonance frequency to between 13.6 MHz and 13.8 MHz.
- The type of capacitor used for tuning is also important. There are many types, but they do not behave as capacitors above a certain frequency. The wrong capacitor will introduce losses into the LC tank, lowering the quality. Therefore we strongly recommend using a COG: Class I (Also known as “NPO”).
- Very thin wiring will add ohmic losses. So if there is enough space the width can be 500 µm or more. The gap between can be smaller, which will introduce a parallel capacitance between the traces. The lower the number of coil turns, the better.
- For a full calculation example please check the document antenna design guide on the website.

Attention: For a good antenna coupling, it is strongly recommended that the receiving and transmitting antennas are to be arranged parallel to one another and centrally above one another.

For more detailed information about antenna design see the antenna design guide for NLM0011 here .

3 Firmware development

3.1 Memory

The NAC1080 contains several types of memory, described below.

An embedded ROM holds the reset and interrupt vectors as well as a system library. This library comprises start-up code, NFC functionality and device drivers. The ROM content is defined by the design of the chip and cannot be changed by the user.

A non-volatile Flash memory (NVM) holds three segments of data. At the starting address, there are two segments with parameters, one of them reserved for internal NAC1080 usage (DPARAM), the other available for customer specific parameters (APARAM). The third segment is intended to store user-specific code.

The list of memory is completed by static RAM blocks. A small part of the RAM is used by the ROM library, e.g., for storing NFC frames, with the bigger part available for use by the custom firmware in NVM. The RAM belongs to the power domain of the CPU core. It will preserve its content while the CPU is in power-saving mode, e.g., while stopped due to a WFI instruction, but will lose the content when the CPU core is powered down, e.g., if the chip is put into standby mode with only the standby domain still being powered.

Table 2 Memory regions

Start address	End address	Size	Type
0x00000000	0x00003FFF	16 kB	ROM
0x00010000	0x000107FF	2 kB	NVM (parameter)
0x00010800	0x0001EFFF	58 kB	NVM (code)
0x00020000	0x00021FFF	8 kB	RAM
0x20000000	0x20001FFF	8 kB	RAM2

3.2 ROM library

The ROM library provides the start-up code, NFC handler and device drivers.

As the name suggests, the ROM library is located in the ROM, starting at address 0. The first words are reserved by the CPU for the vector table holding the initial stack pointer, the reset vector and pointers to interrupt service routines. These vectors are pointing to code in the ROM.

The reset vector points to the start-up code in the ROM. It performs basic initialization of the CPU and some of the I/O controllers, e. g., the NFC interface. When the initialization of the NFC state machine is finished, it is ready to receive NFC commands.

When the start-up code has finished the initialization, it searches for user code in the NVM and hands over control. The user code is detected by its stack pointer and its reset vector, which are located at a well-known address. If no user code is found, the ROM start-up code will enter an infinite loop which puts the CPU core into sleep mode whenever possible in order to minimize power consumption.

The interrupt vectors point to handlers located in the ROM. Most of the vectors can be redirected to user code by setting the appropriate fields in the NVM in a section named APARAM. APARAM is the second section in NVM and provides entries to customize options within the ROM library. Besides user-defined interrupt handlers, security settings can also be adjusted to restrict access through NFC, or I/O configurations can be stored, which are automatically applied by the ROM library when the chip is starting from a reset condition.

3.3 Toolchain

The NAC1080 embeds an Arm® Cortex®-M0 core. For this core, there are a wide variety of toolchains to build applications.

The ROM library and the sample applications were developed on a host using the GNU cross-compiler collection, namely the “C” compiler *arm-none-eabi-gcc*. The build process is controlled by GNU *make*. Several additional tools are used in the process. The build system with all scripts and tools is packaged into the project directory and can be stored alongside the source code in a revision control system such as Git.

As the build system is based on GNU make and makefiles, there is no constraint on the editor to be used. Any editor, such as Notepad++ or Emacs, and any IDE such as Eclipse can be used to write source code for NAC1080 applications.

The NAC1080 supports the Arm® SWD debug port. The build system comprises scripts for the Lauterbach Trace32 debugger. In addition, there is support for the Segger J-Link.

The software development kit (SDK) for the NAC1080 can be downloaded as a ZIP file from the Infineon website. It contains the toolchain for Windows hosts, the ROM library declarations, and a simple project as a starting point for your own developments. Third-party tools are needed only for editing the source code (e.g., Eclipse), and for debugging and writing the binary to the device (e.g., Lauterbach Trace 32, Segger J-Link tools).

The SDK comprises an NVM library which complements the ROM library. Functions used from the NVM library are linked into the firmware image targeted to the NVM.

The SDK is installed by simply unzipping the SDK package on the host. The root directory of the SDK contains a file *build_shell.bat*, which, when run, will open a command-line window with an appropriate environment to start the build process. On the command line, you can start the make utility, usually with the parameter *all*, sometimes with *clean*.

After editing the source code and compiling it successfully with *make all*, usually the firmware is programmed to the device and tested there. This can be achieved by a debugger such as the Lauterbach Trace32. This debugger is supported by the SDK in the form of scripts and start-up files. After the firmware has been built, the file *t32_start_real.bat* can be used to start the Trace32 software with an environment supporting the NAC1080. When the Trace32 is loading the scripts, you are prompted as to whether the newly compiled firmware shall be written to the NVM of the NAC1080 or the debug session shall use the current firmware in the NAC1080. The *t32_start_real.bat* batch file expects the Trace32 software to be installed in the default path. If a different path was chosen, the batch file has to be edited to point to the proper directory.

The SDK also provides support for the Segger J-Link adapter. Besides drivers from Segger, an IDE may be needed to initiate writing the firmware to the device and to assist in debugging on the device. For Eclipse and Visual Studio Code, plugins are available to use the J-Link GDB server, and Segger offers its own debug environment Ozone. Please note that in any case the J-Link device database must be customized to support the NAC1080. For the J-Link GDB server, a different device database can be selected via command-line options, for Ozone and some other tools, the database in their program folder must be modified. Please refer to the SDK documentation for details.

3 Firmware development

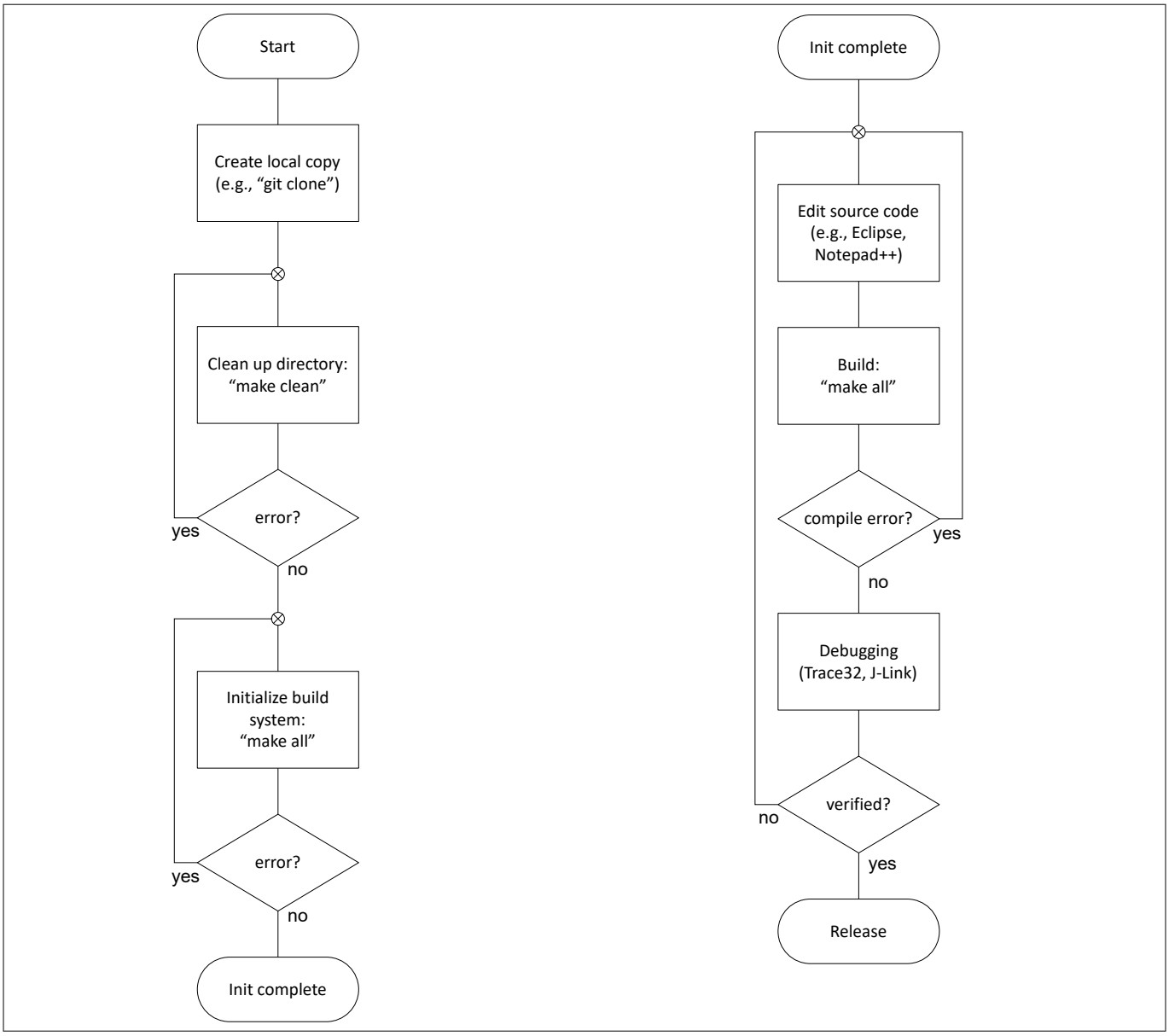

Figure 16 Software development workflow

3.4 Smart lock firmware

A simple smart lock application consists of two steps:

- Charge the C_HB capacitor to provide energy for motor operation. Authentication handshake with smartphone may happen in this step.
- Operate the motor to open the lock.

If the smart lock must actively move the lock bar into the “closed” position, additional steps are necessary:

- Charge the C_HB capacitor again to provide energy for the reverse motor operation, or simply wait for a fixed time span.
- Operate the motor in reverse direction to close the lock bar.

Firmware can be designed as a simple sequence of the steps mentioned above. Each of the steps contains a loop which checks if it can advance to the next step, otherwise it enters a short delay in sleep mode to save power. After all of the steps have been executed, the firmware may enter a low power endless loop, or just start over from the beginning if the user still has the smartphone attached to the antenna to open the lock again.

With the simple scheme listed above, first charging a capacitor, then using the stored energy to drive a motor, the capacitor has to be quite large, and the time needed to charge it may be quite long. A more advanced stepwise scheme is shown in Figure 12. Here, a smaller, cheaper capacitor can be used which will charge faster, but it will not provide the energy to rotate the motor for the angle required in the application. Thus, the capacitor is recharged several times until the required motor rotation is achieved.

In the following paragraphs, a voltage-controlled stepwise operation is shown which uses a comparator function provided in the NAC1080 SDK to determine when the capacitor is fully charged, or when it was discharged to a level where the motor will stop running. In this document, only a few code snippets are shown; the full source code can be found in the project folder *smackfw_motor_stepwise*, which is provided in a ZIP file.

The firmware example starts operating the motor once the C_HB capacitor is charged by harvesting from an NFC field and does not incorporate communication with a smartphone. The chapter about motor operation appears out of order, because the H-bridge control function described there is also used in conjunction with the voltage comparator function.

3.4.1 Operating the motor

This section describes how to drive a motor connected to the H-bridge.

The NAC1080 is equipped with an H-bridge as shown in Figure 4. Each output of the H-bridge can be switched to the positive terminal VCC_HB or to ground. When a motor is connected between the H-bridge outputs, its terminals can be left open, switched to the same voltage (e.g., motor shorted), switched to a positive voltage (e.g., motor running in forward direction) or to a negative voltage (e.g., motor running in reverse direction), depending on which transistors of the H-bridge are closed and which are left open (not conducting).

In this example, a simple motor without position indicator is used. This means that when the motor drives the lock bar to the open position, there is no signal when this position is reached. There is also no signal to the microcontroller, when the lock bar is moved back to the closed position.

For both directions, it is assumed that there is a mechanical obstruction at both end positions which will block the motor from turning any further. With a few sample devices, a time is measured which the motor needs to move the lock bar to the desired position. From this, a worst-case time is determined which should also contain a safety margin. In the actual device, the H-bridge will then be switched on for this amount of time to ensure that the lock opens or closes reliably, and also in situations with higher mechanical load and slower rotation of the motor. For devices with a light load, the end position will be reached sooner but mechanical obstruction will prevent the motor from turning any further.

This is a sample code to switch on the H-bridge output to start motor operation:

```
set_hb_switch(true, false, false, false);   /* start switching on with high side */
set_hb_switch(true, false, false, true);
```

First, the left side of the H-bridge is switched to the VCC_HB pin, e.g., the positive side of the C_HB capacitor. When this step is complete, the motor does not run because its other terminal still is left open. Now, the right side of the H-bridge is switched to ground in the second code line, and the motor is finally connected to the C_HB capacitor. This will start the motor movement.

In this example, the lower switch of the H-bridge, which connects to ground, is closed after the high-side is switched on. This scheme is chosen because the switches on the low-side and high-side are different inside the H-bridge. The low-side switches provide additional features such as slope control.

Switching off the H-bridge can also be done in two steps, similar to the scheme used when switching on the motor. First, the low-side of the H-bridge is switched off, then the high-side:

```
set_hb_switch(true, false, false, false);   /* start switching off with low side */
set_hb_switch(false, false, false, false);
```

Please note that the H-bridge is used to route the C_HB capacitor voltage to the analog units inside the NAC1080 for use with the voltage comparator function, and the upper side of the H-bridge is left switched on as long as the voltage comparison is in use.

When the motor is switched on, the running time is observed. In a single-step concept using a large capacitor, the H-bridge is simply switched off when the total run time has passed.

In a voltage-controlled stepwise operation, for each step a remaining run time is calculated. In the first step, this is the total runtime; for further steps, this value is decreased by the already passed run time. Then the H-bridge is switched on, and the motor will turn. For each of the steps, the motor will be stopped if one of these two conditions becomes true: either the remaining runtime has passed, or the voltage of the C_HB capacitor connected to the H-bridge supply VCC_HB is discharged to its lower threshold. If the remaining runtime has passed, no more steps will follow. Otherwise, the firmware waits until the C_HB capacitor is charged to its upper threshold, and the firmware continues with the next step.

3 Firmware development

If it is desired to move the lock bar back to the start position, then the steps above are repeated with appropriate arguments to the H-bridge calls. In the first motor operation described above, the left side of the H-bridge has been connected to the VCC_HB pin, and the right side has been connected to ground. For reverse motor operation, the connections must be swapped, e.g., connect the left side of the H-bridge to ground, and the right side to the VCC_HB pin.

Note: Do not close the top and the bottom switch of one side of the H-bridge at the same time. This will shorten the VCC_HB supply to GND with unpredictable results. When switching one side of the H-bridge from the positive supply or vice versa, first open both switches, then close the desired switch in a separate step. This takes care of the off-delay of the switches and prevents critical glitches between the H-bridge transistors as the on-delay is usually shorter.

3.4.2 Determining the C_HB voltage

The amount of power that can be harvested from the NFC field depends on many parameters, e.g., coupling between reader and smart lock, design of the antennas and many more. Furthermore, for a certain amount of power, time until a capacitor is charged to the nominal voltage depends on the chosen capacitance.

When charging the C_HB capacitor from NFC harvesting, the NAC1080 behaves almost like a current source. So, the time to charge a capacitor to the nominal voltage is defined as:

$$t = \frac{C \cdot U}{I}$$

where the capacitor value C and the nominal voltage U are known when designing the smart lock, but the current I depends on the harvested power and will change for each use of the device.

A fixed charge time can be determined in advance based on measurements of a worst-case scenario. With good coupling between smartphone and smart lock, however, the user must wait longer than necessary before the lock opens. If the charging speed is determined during runtime, the time until the lock opens can be reduced when the coupling of the NFC antennas is good and the harvested power reaches a maximum, while at the same time proper operation can be achieved even in situations where coupling is bad.

The NAC1080 allows the user to judge the C_HB capacitor voltage during runtime; however, the voltage can not be detected directly. The SDK NVM library provides a function to compare the voltage on the H-bridge outputs against a given value. If the top-side switch of the H-bridge for an output is closed, the voltage on the H-bridge output is equal to the C_HB capacitor voltage, and now the capacitor voltage can be compared against a threshold.

Before the voltage comparator function can be used, the corresponding library functions and the analog units inside the NAC1080 have to be initialized:

```
shc_init();
```

To mirror the C_HB voltage to the H-bridge output MA, the top-side switch of this side has to be switched on:

```
set_hb_switch(true, false, false, false);
```

Then, the voltage on this H-bridge output can be compared against a given threshold. The function will return true if the voltage is higher than the threshold:

```
cmp = shc_compare(shc_channel_ma, VOLTAGE_ON);
```

3.4.3 Stepwise motor control

This chapter describes the main loop of the NAC1080 firmware using a stepwise scheme for motor control.

With the functionalities described in the previous chapters, a voltage-controlled stepwise method can be implemented. The voltage at the C_HB capacitor during charging and motor operation is shown as a yellow line in Figure 12. The blue line shows the C_CB voltage, which can be seen as an indication of harvesting from the NFC field.

The control logic for motor operation is defined by two voltage levels of the C_HB capacitor, the upper and the lower limit.

The upper limit defines the point where C_HB is almost fully charged. It has stored the most energy, and the motor can be started.

When the motor is running, it will drain energy from the C_HB capacitor, and the voltage is dropping. At some point, the motor does not have enough voltage to rotate any further. The lower threshold has to be defined as a voltage just above this point. When the lower threshold is reached, the motor operation will be paused, and the energy harvested from the NFC field can recharge the C_HB capacitor again.

Once the C_HB capacitor is charged to the upper threshold, the firmware can start over again with switching on the motor.

Usually, the C_HB capacitor must be initially charged before the motor can be started. With the voltage-controlled stepwise method, the recharge phase can be used also for initial charging, as the event to proceed to the next step is reaching the actual voltage threshold, which is just the same as ending the recharge phase.

This pseudo code describes this simple loop:

```
while (true) {
  while (!compare(C_HB, upper_threshold))
    ;   // wait until C_HB is charged
  start_motor();
  while (compare(C_HB, lower_threshold))
    ;   // let motor run as long as C_HB voltage is high enough
  stop_motor();
}
```

Time measurement

Using this simple loop, the motor will run as long as the NFC field is present. With an actual device, the motor shall just rotate a certain angle in order to move a lock bar between the locked and the open position. An additional condition is necessary to control the loop, for example micro switches or photo sensors to detect the motor position.

If such devices are not available and the end position of the motor movement is not critical or defined by a mechanical barrier, a runtime measurement can be used instead. It is assumed that the motor will rotate at a known speed, and there is a connection between the time the motor is switched on and the amount of rotation angle achieved by the motor during this time. For the stepwise motor control method, it is also assumed that the rotation angle, and so the motor runtime, of each of the steps can be totalled. For each *on* or *off* operation, a fixed value may be added or subtracted from the sum of the runtime if the motor needs time to speed up or keeps rotating for a short time after its supply is switched off.

The NAC1080 has a system timer unit with six 16-bit timers running at CPU clock speed. The timers can be cascaded, e.g., to form a 32-bit timer. This feature is used in the stepwise firmware, because 16-bit timers do not cover the time spans required in smart lock applications.

The cascaded 32-bit timer is used as a kind of wall clock. Once started, it runs until the end. To measure the time the motor was switched on in one step, the time value is captured to the variable *timestamp_on* when the motor is switched on, and to the variable *timestamp_off* when it is switched off. The runtime within this step then is calculated as the difference between those two timestamps: *timestamp_off* - *timestamp_on*.

3 Firmware development

The ROM and NVM libraries of the NAC1080 provide functions to deal with the system timer unit. A cascaded 32-bit timer is comprised of a 16-bit prescaler and a 16-bit main timer. If desired, the prescaler can be configured to produce a clock with a certain unit, e.g., 1 ms, and the main timer then counts in steps of this unit. Here, the timers shall be cascaded to form a continuous 32-bit timer, so the period of the prescaler is set to the rollover value, and so is the period of the main timer:

```
sys_tim_cyclic_cascaded(TIMER_CLOCK, 0xffff, 0xffff);
```

The combined 32-bit value of the cascaded timer pair can be read using a library function:

```
timestamp_on = sys_tim_cyclic_cascaded_get_combined(TIMER_CLOCK);
```

Combined loop

The firmware example combines the loop and time measurement mentioned above. It uses the flag *run* to control the outer “endless” loop. The inner loops of the pseudo code above are replaced by a state machine like approach, controlled by the variable *state*.

3 Firmware development

This code snippet is an excerpt from the firmware example with comments and also code lines removed:

```
while (run)
{
    if (state)
    {
        cmp = shc_compare(shc_channel_ma, VOLTAGE_OFF);

        if (!cmp || ((int32_t)(sys_tim_cyclic_cascaded_get_combined(TIMER_CLOCK) - target_off)
> 0))
        {
            timestamp_off = sys_tim_cyclic_cascaded_get_combined(TIMER_CLOCK);
            total_on += timestamp_off - timestamp_on;
            set_hb_switch(true, false, false, false);
            state = false;

            if (total_on >= ms2ticks(TOTAL_MOTOR_RUNTIME))
            {
                run = false;
                break;
            }
        }
    }
    else
    {
        cmp = shc_compare(shc_channel_ma, VOLTAGE_ON);
        if (cmp)
        {
            timestamp_on = sys_tim_cyclic_cascaded_get_combined(TIMER_CLOCK);
            target_off = timestamp_on + (ms2ticks(TOTAL_MOTOR_RUNTIME) - total_on);
            set_hb_switch(true, false, false, true);
            state = true;
        }
    }
}
```

3.5 Communication with a smartphone

This chapter gives a brief overview of the NFC protocol of the NAC1080.

In order to provide a versatile but secure communication path, the NAC1080 implements a proprietary extension to the NFC protocol. The NFC handler and the proprietary extension are located in the ROM and provide access to a mailbox for data exchange between an NFC reader, e.g., a smartphone, and the device.

To communicate with the device, an NFC reader can write a message to the mailbox, and then trigger an action by calling one of 16 application-specific functions which can be implemented in NVM to suit the needs of the device. These functions can write a response to the mailbox, which in turn can then be read by the NFC reader.

Access to these functions and other features of the proprietary protocol can be enabled or denied in a parameter block in the NVM. With these configuration options, the application-specific firmware can adjust access rights and security according to the needs of the specific device.

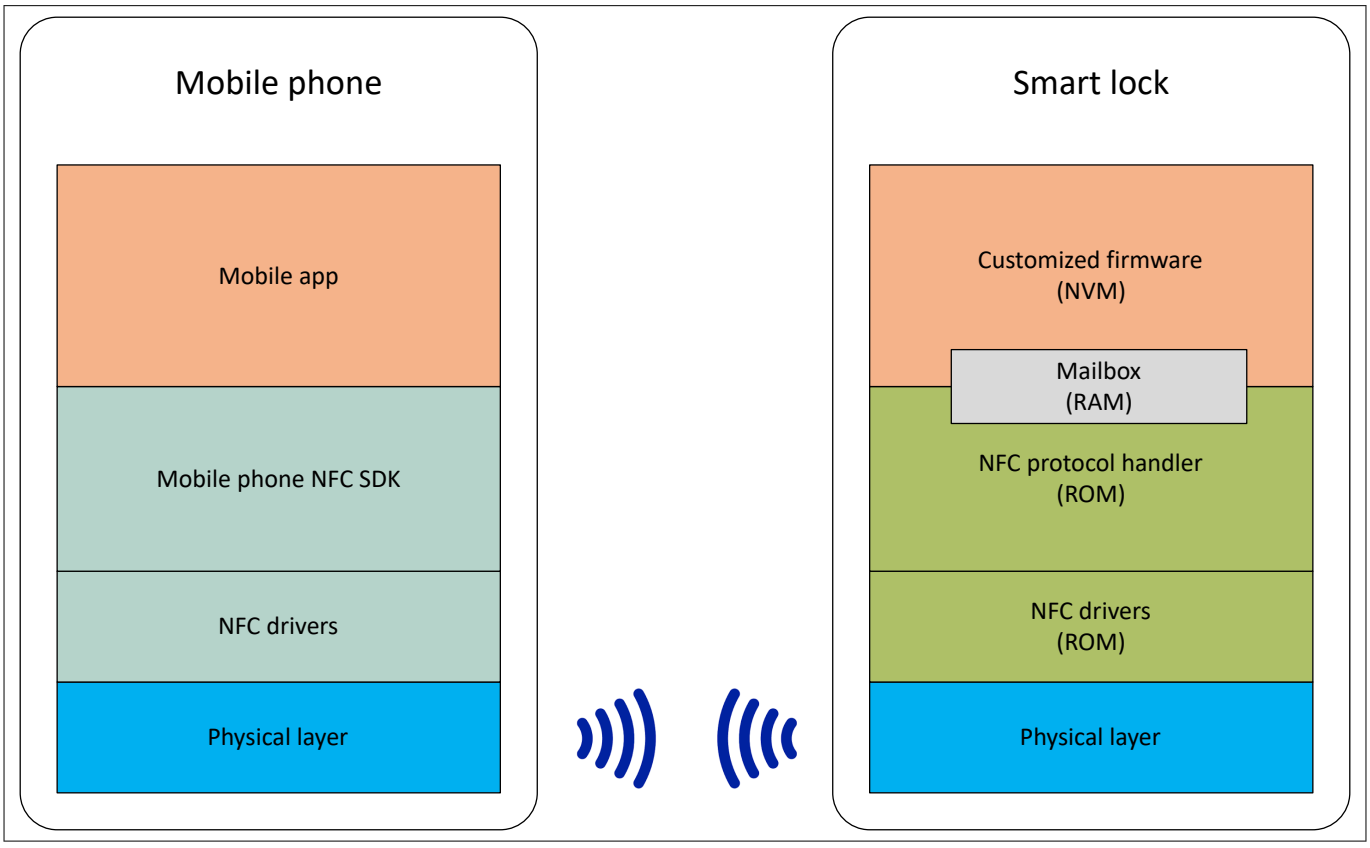

Figure 17 NFC stack

3.5.1 NAC1080 communication protocol basics

This section briefly describes the communication basics between the NAC1080 and an external communication host.

The communication principle between the NAC1080 and an external communication host follows the definition of NFC. The NFC forum defines two roles in communication: a reader (also called PCD) and a tag (also called PICC). The reader is establishing communication and initiating a link activation procedure. After the link activation procedure is successful, the reader requests the tag to enter the protocol state and communication is continued according to an application-specific protocol with application-specific content. NFC is by definition a half-duplex communication, whereas only the reader is allowed to issue a communication request and the tag is only allowed to respond to the reader's request. The NAC1080 is always acting in the tag role of NFC, and the external device (e.g. the smart phone) is assuming the readers role.

The NFC forum allows the use of an application-specific proprietary protocol in a proprietary protocol state. This approach is used in the NAC1080 in order to provide efficient, reliable and safe communication between the reader and the NAC1080.

Following the proprietary application protocol, the external reader can apply the following types of request to the NAC1080:

- Write request
  - This request targets a write access to a specific CPU-mapped address of the NAC1080. Word (32-bit), half-word (16-bit) or byte accesses are supported.
- Read request
  - This request targets a read access to a specific CPU-mapped address of the NAC1080. Word (32-bit), half-word (16-bit) or byte accesses are supported.
- Message request
  - This request will trigger a message specific action in the NAC1080.

The NAC1080 communication protocol firmware handler will decode the requests, check privileges of requests and perform the requested action. Finally, it will assemble a response to the reader and send it out.

Below is an overview on the types of request:

Table 3 Request frames

	Command type	**Parameter 1**	**Parameter 2**	**Parameter 3**	**Parameter 4**	**Parameter 5**	**Parameter 6**	**Parameter 7**	**Parameter 8**
WRITE_WORD	0x70	*ADDR[3]*	*ADDR[2]*	*ADDR[1]*	*ADDR[0]*	*DATA[3]*	*DATA[2]*	*DATA[1]*	*DATA[0]*
WRITE_HALFWORD	0x71	*ADDR[3]*	*ADDR[2]*	*ADDR[1]*	*ADDR[0]*	*DATA[1]*	*DATA[0]*		
WRITE_BYTE	0x72	*ADDR[3]*	*ADDR[2]*	*ADDR[1]*	*ADDR[0]*	*DATA[0]*			
READ_WORD	0x80	*ADDR[3]*	*ADDR[2]*	*ADDR[1]*	*ADDR[0]*				
READ_HALFWORD	0x81	*ADDR[3]*	*ADDR[2]*	*ADDR[1]*	*ADDR[0]*				
READ_BYTE	0x82	*ADDR[3]*	*ADDR[2]*	*ADDR[1]*	*ADDR[0]*				
MESSAGE	0x60	*MESSAGE_INDEX*	*RESERVED (default 0x00)*	*RESERVED (default 0x00)*	*RESERVED (default 0x00)*				

3 Firmware development

The NAC1080 responds with NFC-standard frames of the following structure (only payload shown):

Table 4 Response frames

Command	Remark	ACK1	ACK2	ACK3	ACK4	ACK5	ACK6
READ_BYTE	Access address valid	DAND_ACK_READ_BYTE	DATA				
	Access address invalid	DAND_ERROR_READ_PROTEC	DAND_ERROR_READ_PROTEC				
READ_HALFWORD	Access address valid	DAND_ACK_READ_HALFWORD	DATA[1]	DATA[0]	DAND_ACK_READ_HALFWORD		
	Access address valid, but not halfword aligned	DAND_ERROR_MIS_ALIGNED	DAND_ERROR_MIS_ALIGNED				
	Access address invalid	DAND_ERROR_READ_PROTEC	DAND_ERROR_READ_PROTEC				
READ_WORD	Access address valid	DAND_ACK_READ_WORD	DATA[3]	DATA[2]	DATA[1]	DATA[0]	DAND_ACK_READ_WORD
	Access address valid, but not word aligned	DAND_ERROR_MIS_ALIGNED	DAND_ERROR_MIS_ALIGNED				
	Access address invalid	DAND_ERROR_READ_PROTEC	DAND_ERROR_READ_PROTEC				
WRITE_BYTE	Access address valid	DAND_ACK_WRITE_BYTE	DAND_ACK_WRITE_BYTE				
	Access address invalid	DAND_ERROR_WRITE_PROTEC	DAND_ERROR_WRITE_PROTEC				
WRITE_HALFWORD	Access address valid	DAND_ACK_WRITE_HALFWORD	DAND_ACK_WRITE_HALFWORD				

(table continues...)

Table 4 (continued) Response frames

Command	Remark	ACK1	ACK2	ACK3	ACK4	ACK5	ACK6
	Access address valid, but not halfword aligned	DAND_ERROR_MIS_ALIGNED	DAND_ERROR_MIS_ALIGNED				
	Access address invalid	DAND_ERROR_WRITE_PROTEC	DAND_ERROR_WRITE_PROTEC				
WRITE_WORD	Access address valid	DAND_ACK_WRITE_WORD	DAND_ACK_WRITE_WORD				
	Access address invalid	DAND_ERROR_WRITE_PROTEC	DAND_ERROR_WRITE_PROTEC				
	Access address valid, but not word aligned	DAND_ERROR_MIS_ALIGNED	DAND_ERROR_MIS_ALIGNED				

The address space, allowed by external read/write access request, can be restricted to a limited address space only. This is useful in order to guarantee privacy and data protection of the NAC1080, when applying an external read/write access. That limited address space is called mailbox. The mailbox is a reserved area of the NAC1080 data RAM. The mailbox is defined in ROM content as an array of 32-bit data, which is located in data RAM. The array size is defined in the ROM as well.

3 Firmware development

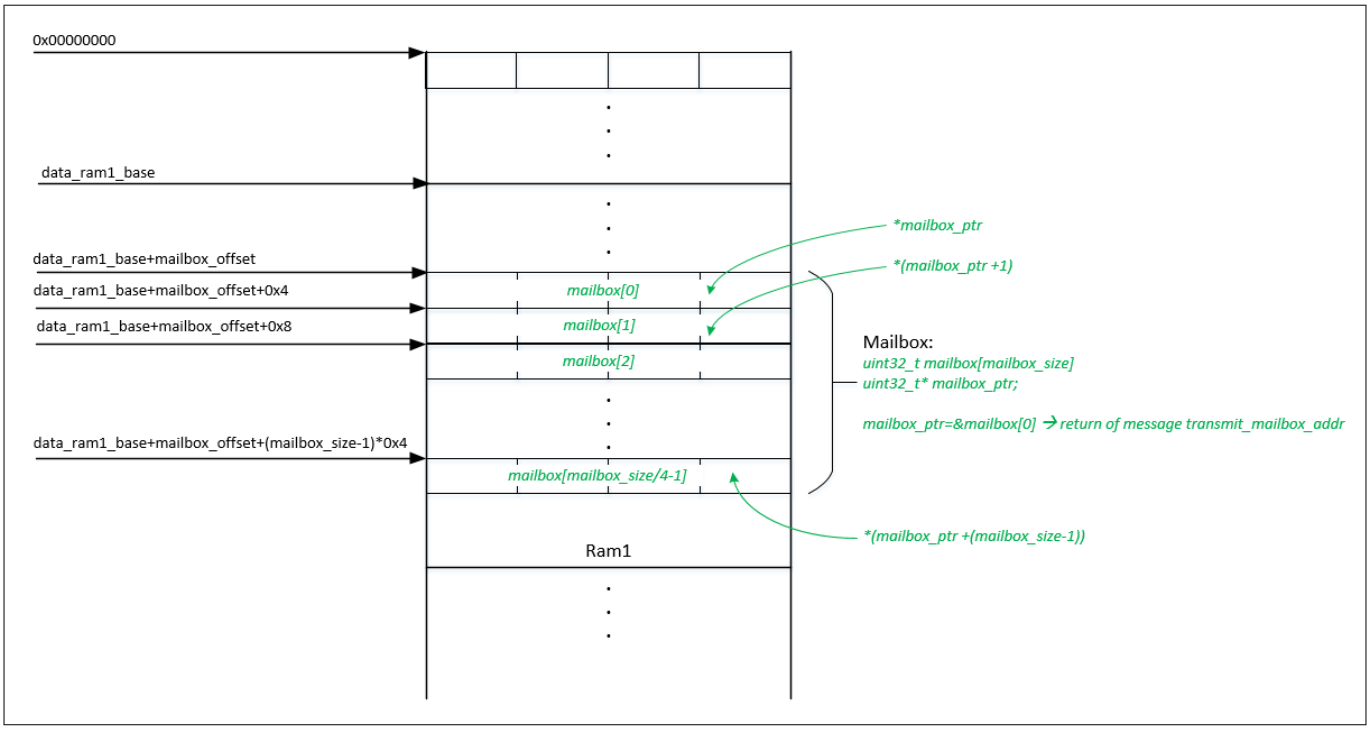

Figure 18 Mailbox organization

The mailbox can be accessed by read and write access by the NFC reader. Accessing an address outside the mailbox by reader read/write access request is only granted by the NAC1080 firmware, if the respective application parameter "*disable_mailbox*" in the NAC1080 NVM indicates that to the NAC1080 firmware.

Basically, the mailbox is used to provide a ping-pong-based communication scheme.

- The reader can request the NAC1080 by a protocol message to respond with the mailbox size and mailbox address.
- The reader now may put a coded message into the mailbox, which might contain a request to perform a certain action.
- After the coded message is put into the mailbox by reader, the reader will indicate to the NAC1080 by a protocol message *mailbox_message_valid* that the mailbox contains a coded message.
- The NAC1080 will now look into the mailbox, decode the coded message, process the message and put a reply into the mailbox.
- The reader now will read the reply from mailbox.

The following image shows an example of protocol message, coded message, message response and mailbox reply:

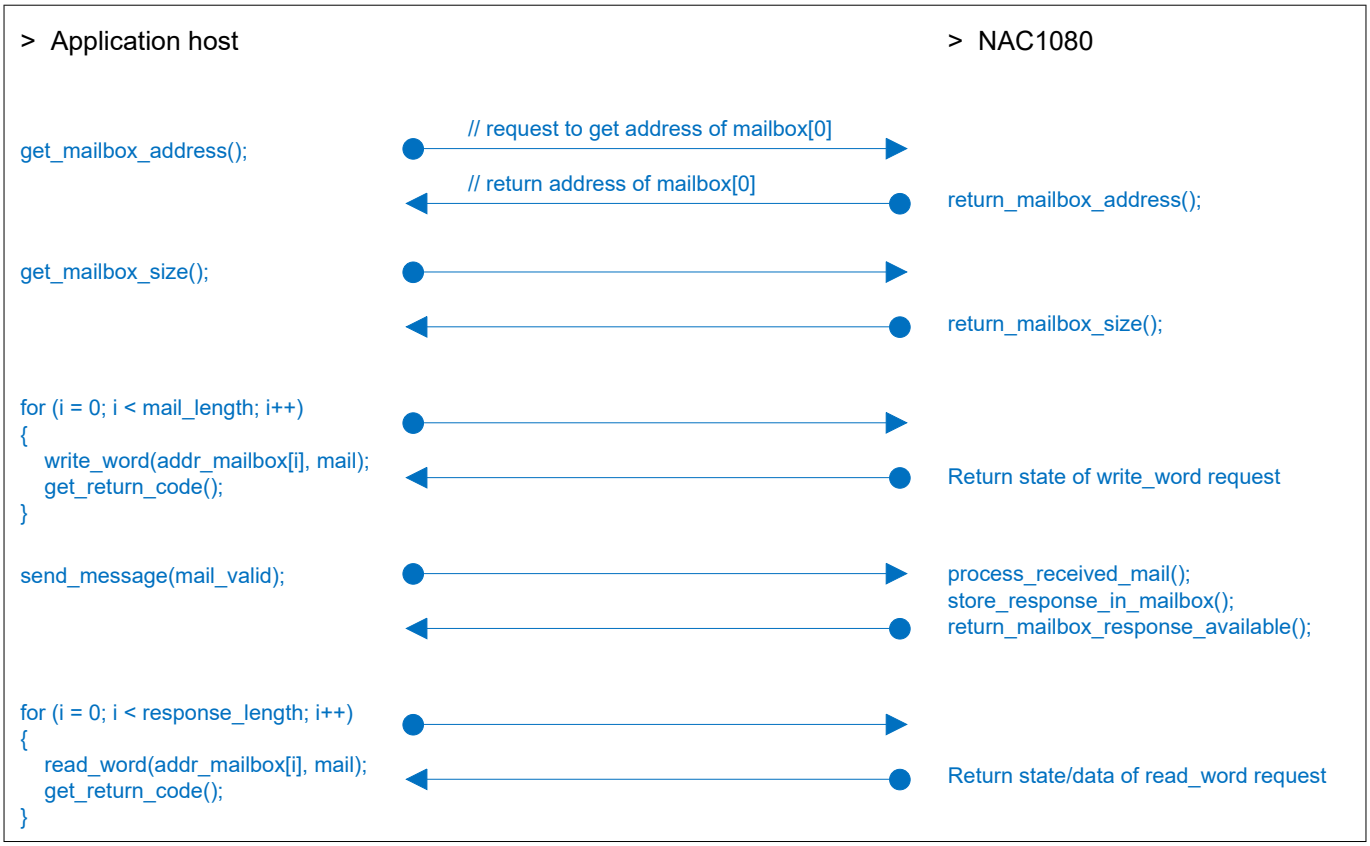

Figure 19 Mailbox ping-pong

The coded messages, which are supported by the NAC1080 communication protocol handler, might be of the following type:

- HAL_ACC_WRITE_W
  - This command will write a 32-bit data word at an address. Address and data are given as arguments. Before executing the write access, write privileges of access to a specified address will be checked. Write access will only be executed if write privileges are given.
- HAL_ACC_WRITE_HW
  - This command will write a 16-bit data word at an address. Address and data are given as arguments. Before executing the write access, write privileges of access to a specified address will be checked. Write access will only be executed if write privileges are given.
- HAL_ACC_WRITE_B
  - This command will write a data byte at an address. Address and data are given as arguments. Before executing the write access, write privileges of access to a specified address will be checked. Write access will only be executed if write privileges are given.
- HAL_ACC_READ_W
  - This command will read a data word from address. Address is given by argument. Result is written into mailbox and can be read from there by external read access request. Before executing the read access, read privileges will be checked.
- HAL_ACC_READ_HW
  - This command will read a data half-word from address. Address is given by argument. Result is written into mailbox and can be read from there by external read access request. Before executing the read access, read privileges will be checked.

- HAL_ACC_READ_B
  - This command will read a data byte from address. Address is given by argument. Result is written into mailbox and can be read from there by external read access request. Before executing the read access, read privileges will be checked.
- CALL_APP_FUNCTION
  - This command will call an application-specific function, which is usually located in the NVM. Up to 16 functions are selectable. The function pointers are stored in the NVM at address *aparam_pointer_get()->app_prog[N] (0<=N<16).* The parameters of the function are function dependent, and will be stored in the arguments field of the mailbox. The number of arguments is completely flexible and is finally only limited by the mailbox size. The custom function can store results in the argument elements of the mailbox and can be read out by external read access into the mailbox, after the custom function terminates and returns.
- The following images illustrate the described scheme in detail:

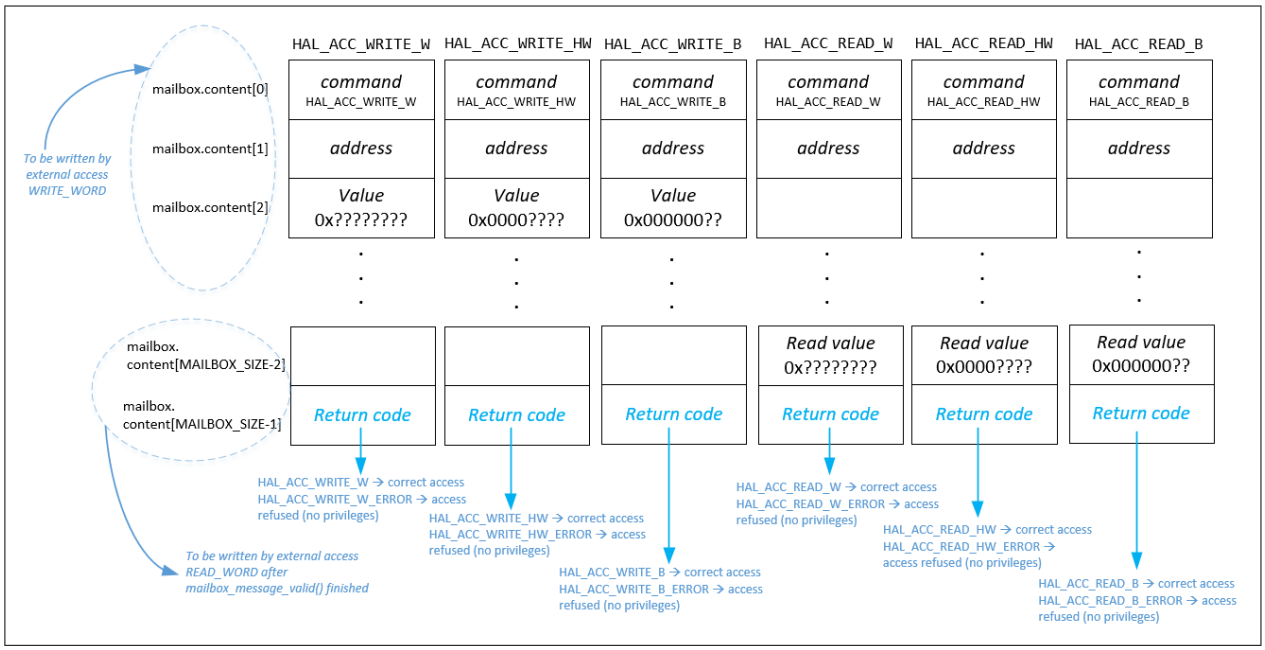

Figure 20 mailbox_message_valid() principle of HAL_ACC R/W

3 Firmware development

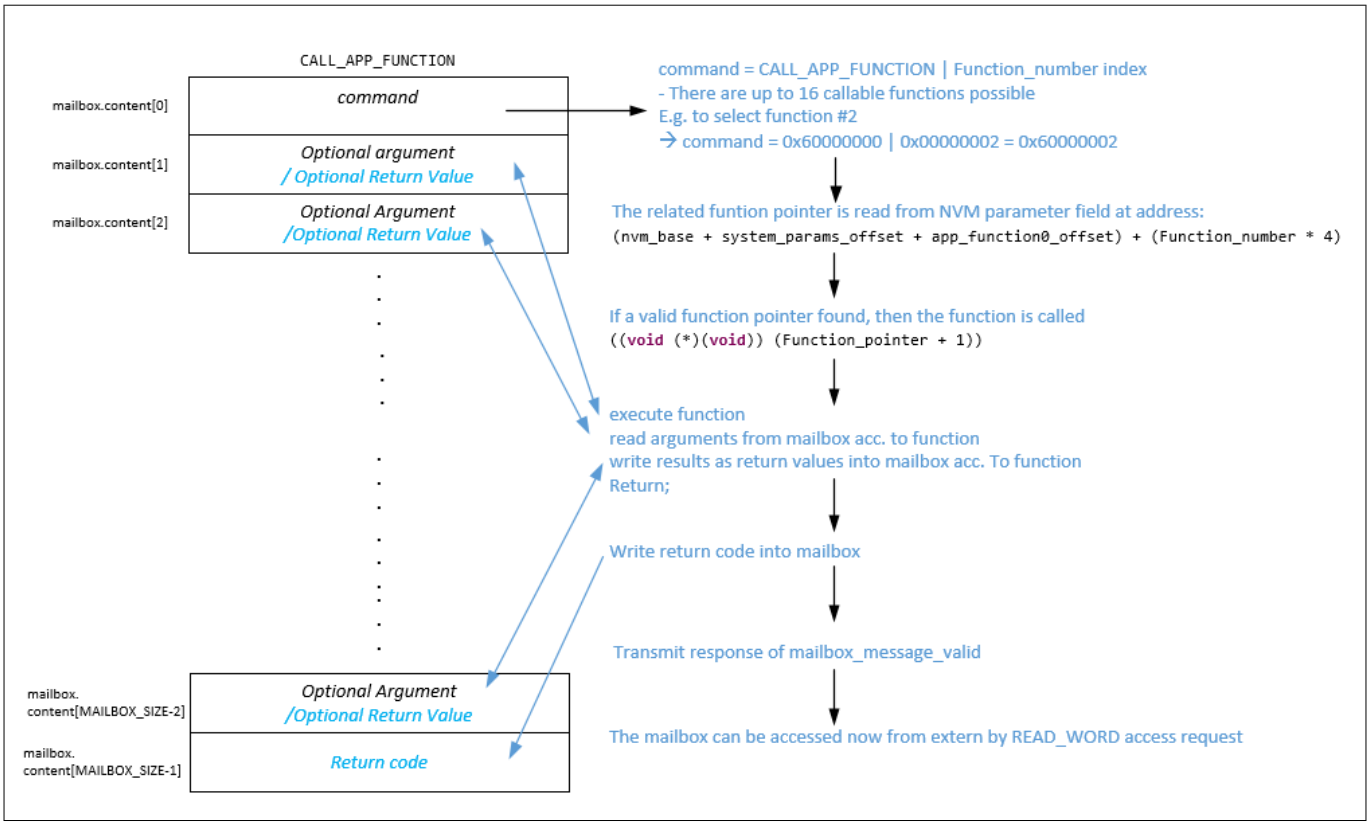

Figure 21 mailbox_message_valid() principle of CALL_APP_FUNCTION

Revision history

Document version	Date of release	Description of changes
V1.0	2022-06-23	First release

Trademarks

All referenced product or service names and trademarks are the property of their respective owners.

Edition 2022-06-23

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about any aspect of this document?

Email: erratum@infineon.com

Document reference
IFX-vnf1619117966863

Important notice

The information contained in this application note is given as a hint for the implementation of the product only and shall in no event be regarded as a description or warranty of a certain functionality, condition or quality of the product. Before implementation of the product, the recipient of this application note must verify any function and other technical information given herein in the real application. Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind (including without limitation warranties of non-infringement of intellectual property rights of any third party) with respect to any and all information given in this application note.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

Warnings

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.