# ON Semiconductor

# Is Now

**To learn more about onsemi™, please visit our website at**
**www.onsemi.com**

# AND8386/D

# Bootloading BelaSigna® 250 Using the I²C Interface

ON Semiconductor®

http://onsemi.com

## APPLICATION NOTE

### INTRODUCTION

This application note describes how to bootload BelaSigna 250 through its I²C interface when it does not have an EEPROM attached. This situation can occur when a Bluetooth® or a baseband chip, or any I²C−master capable chipset, is connected to BelaSigna 250 through the two−wire synchronous serial (TWSS) port. The TWSS port is essentially an I²C−compatible port. The I²C terminology will be used throughout this document to designate BelaSigna 250's TWSS port.

Since no EEPROM is attached to BelaSigna 250, the external device must have dedicated memory space in its non−volatile memory to store the BelaSigna 250 application. It can either be internal Flash, as is the case with some Bluetooth devices or external Flash / NAND Flash memories in Bluetooth or mobile phone applications.

BelaSigna 250 has an I²C handler built into its Boot ROM. This I²C handler allows any I²C−master host to connect to and bootload BelaSigna 250, as described in this application note. The ROM−based low−level communication protocol on the I²C port is described in the *Communication Protocols Manual**.

### THE BOOTING PROCESS

#### Contacting BelaSigna 250's I²C Handler

BelaSigna 250 Boot ROM begins execution at Power−On Reset (POR). Once booting starts, there is a defined time period during which the I²C−master must contact BelaSigna 250 to begin its I²C bootloading procedure. The following table shows BelaSigna 250's booting sequence.

**Table 1. TIMING OF I²C DOWNLOAD PROCEDURE**

| Time (ms) | Activity |
|---|---|
| 0–20 | POR Delay |
| 20–70 | Boot ROM Initialization |
| 70–220 | Boot ROM Execution |
| 220–1000 | Bootloading |

After powering up BelaSigna 250, the I²C master must continuously send the I²C protocol's status request command. BelaSigna 250 will not acknowledge (NAK) this transfer until it is ready for communication. As soon as the I²C−master receives the acknowledge (ACK), it means that BelaSigna 250 has interrupted its regular booting process, and is waiting for more I²C commands.

The I²C−master device cannot connect to BelaSigna 250's I²C handler during the POR delay; from the time of power−on (t = 0 ms) to the end of the POR delay (t = 20 ms), BelaSigna 250 will NAK any I²C request. The I²C handler is activated by the Boot ROM at the beginning of the Boot ROM initialization (t = 20 ms). When there is no EEPROM, BelaSigna 250 enters a loop from this time (t = 20 ms) until a timeout occurs (t = 1000 ms), at which time the Watchdog Timer resets BelaSigna 250. If no I²C command is issued by an I²C−master between the end of the POR delay (t = 20 ms) and the end of the bootloading period (t = 1000 ms), BelaSigna 250 will reset and the booting sequence starts over with a POR (i.e., another 20 ms during which BelaSigna 250 will NAK any I²C request). This process continues until communication is established.

The I²C master can initiate communication at any time, waiting for the I²C request to be acknowledged (ACK). If the request is not acknowledged, the I²C master should wait 20 ms and try again. This will guarantee that successful communication can be established. Once an ACK is received, the bootloading process can start.

The voltage used for I²C communication depends on the supply voltage of BelaSigna 250 ($V_{BAT}$) and the state of GPIO15 at POR, as shown in the following table. In typical Bluetooth and mobile phone applications, GPIO15 must be driven low during this time to select high−voltage mode.

**Table 2. VOLTAGE MODE SELECTION**

| $V_{BAT}$ | Recommended Voltage Mode | GPIO15 at Boot | Communication Voltage |
|---|---|---|---|
| 1.25 V | Low−Voltage Mode | High (Default) | 1 V |
| 1.8 V | High−Voltage Mode | Low | $V_{BAT}$ |

*Available with the Developers Tookit. Please contact your local ON Semiconductor sales office for more details.

### Setting the Clock

At powerup, BelaSigna 250 always runs on its non−calibrated internal oscillator with the default clock frequency of approximately 1.8 MHz. In this mode, the I$^2$C interface can run up to a maximum of 100 kbps. Clock switching functions are available within the Boot ROM's I$^2$C protocol, which allow the host processor to initiate a change in the clocking structure, enabling BelaSigna 250 to switch to an external clock, or to a higher clock using the internal oscillator. When BelaSigna 250 is running with a clock of at least 1.92 MHz, the I$^2$C interface can run up to 400 kbps. The actual frequency of BelaSigna 250's internal RC oscillator varies due to variations in the production process. Selecting the internal clock frequency setting of 3.44 MHz ensures that under any process variation condition the actual clock frequency is above 1.92 MHz; hence, the maximum I$^2$C speed of 400 kbps can be used. Alternatively, switching to an external clock that has a frequency of at least 1.92 MHz is also possible to achieve the maximum I$^2$C speed of 400 kbps. Switching to the external clock is done using the clock switching functions.

ON Semiconductor recommends setting the internal clock frequency to 3.44 MHz for I$^2$C for 400 kbps communications, unless an external clock with frequency of 1.92 MHz or greater is used.

### Access Mode

BelaSigna 250 has a built−in IP protection mechanism whereby the chip is always in Restricted Mode at power−up. Limited access to the DSP is allowed in this mode, and a boot operation is not permitted. Consequently, the host must unlock the device to gain access to all the ROM−based protocol functions.

The I$^2$C interface has a status byte that indicates the access mode, which the I$^2$C−master device can read at any time. It is returned by BelaSigna 250 after sending a 'Get Status' command (ASCII Code 'S'). The status byte has the following format:

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| − | 1 | 0 | 1 | − | − | − | 0 − Unrestricted<br>1 − Restricted |

Bits 6:4 are always 0x5; this is the protocol version identifier. Checking that this value is 0x5 means that BelaSigna 250 and its I$^2$C command handler are communicating properly. Bits 7, 3, 2, 1 are reserved. Bit 0 specifies the access mode. This last bit allows the bootloading program to decide whether unlocking functions have to be executed to continue on the boot operation. If bit 0 of the status byte is 1 then unlocking is required; the I$^2$C master must use a ROM−based function called JumpROM to go through the unlocking process. See the *Communication Protocols Manual* for details on this function.

### Addressing

By default, the 7−bit I$^2$C slave address is set to 0b000 0000. Consequently, BelaSigna 250 always responds to a "General Call". The slave address can remain, but as soon as the boot process is finished and the DSP program is being executed, the I$^2$C port only responds to the I$^2$C slave address that the DSP program sets.

## USING THE I$^2$C PROTOCOL TO DOWNLOAD AND RUN AN APPLICATION

### I$^2$C in a Nutshell

As mentioned earlier, the first transfer to take the control over the I$^2$C ROM−based protocol of BelaSigna 250 is an I$^2$C write transfer, which must be continuously sent to the slave until it is acknowledged. In the example below, we use the I$^2$C write transfer to synchronize the ROM handler, and we send a Get Status command, then we wait for the slave to respond with an acknowledgment and the requested status byte.

The I$^2$C protocol is described in the I$^2$C specification issued by Philips Semiconductor. Each transfer (read or write) has the following sequence:

1. Start the communication by sending a start condition (SDA high−low transition while SCL is high).
2. Each bit consists of a low−high−low pulse on the SCL line while SDA is held low (0) or high (1). The I$^2$C master always controls SCL. When the master is sending a bit, the master must set up the data on SDA before the low−high transition of SCL. When the master is receiving a bit, the master must sample the data before the high−low transition of SCL. Bits are transferred from most significant bit (bit 7) to least significant bit (bit 0).
3. After each byte is transferred, a single acknowledge bit is sent by the receiver. The receiver must respond with an ACK (0) to verify that the byte was received. A NAK (1) indicates that the receiver did not recognize the byte, or is not ready.
4. The first 8 bits after the start condition indicate the 7−bit receiver address and the direction of the transfer (0 = Write, 1 = Read). The addressed I$^2$C slave must acknowledge this first byte.
   a. For a write transfer to the general call address, send the following 8−bit sequence: 0b00000000 and check for an ACK from the slave. Subsequent bytes are sent by the master, and must be acknowledged by the slave.
   b. For a read transfer from the general call address, send the following 8−bit sequence: 0b00000001 and check for an ACK from the slave. Subsequent bytes are sent by the slave and must be acknowledged by the master.
5. The I$^2$C master controls how many bytes are sent (in a write transfer) or received (in a read transfer), and terminates the transfer by sending a stop condition (SDA low−high transition while SCL is high).

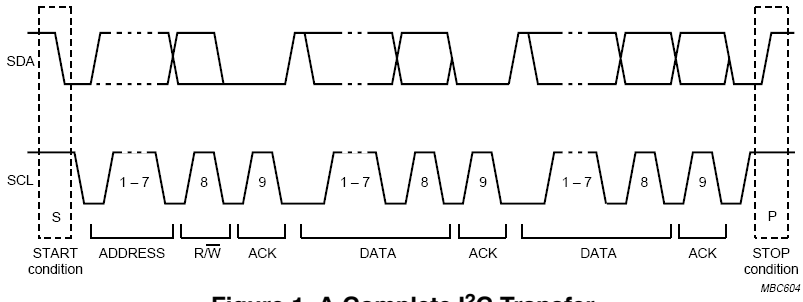


**Figure 1. A Complete I²C Transfer**

See the I²C Bus Specification from Philips Semiconductor for more details on the I²C communication protocol.

### A Complete Bootloading Example

The following description shows how to connect BelaSigna 250, and how to initialize a proper communication, take care of the restricted mode, and make the necessary steps to switch the internal clocking structure of BelaSigna 250, so that it can support the maximum 400 kbps I²C transfer rate.

**Table 3. INITIALIZE AND UNLOCK BELASIGNA 250 THEN CHANGE THE CLOCK FREQUENCY**

| # | Command | Master ↔ Slave | Description |
|---|---|---|---|
| 1 | Until ACK:<br>Get Status<br>'Write to address 0' (0x00) + 'S' (0x53) | → | Continuously poll the ROM−based interface, sending the address byte, followed by a status byte request. As soon as BelaSigna 250 is interrupted, it sends an ACK, the communication is established and the normal boot process of BelaSigna 250 is stopped. BelaSigna 250 waits for more I²C commands. |
| 2 | 'Read from address 0' (0x01) | → | Once the ACK is received, a read operation must be initiated by the master... |
| | (0x51) | ← | ...and BelaSigna 250 responds with 0x51; the last bit indicates that the chip is in restricted mode. |
| 3 | Reset TWSS<br>'Write' (0x00) + 'Q' (0x51) | → | Reset the I²C interface, set the GPIOs to their default values, and install a post−boot safe system status. |
| 4 | Execute JumpROM function 5<br>'Write' (0x00) + 'J' '5' (0x4A 0x05) | → | JumpROM function "Set Opcodes" prepares BelaSigna 250 to run JumpROM unlocking commands. |
| 5 | Execute JumpROM function 4<br>'Write' (0x00) + 'J' '4' (0x4A 0x04) | → | **(This step is unnecessary when no EEPROM is connected.)**<br>JumpROM function "Wipe" writes the 0x5555 pattern in the protected regions of the EEPROM. |
| 6 | Execute JumpROM function 3<br>'Write' (0x00) + 'J' '3' (0x4A 0x03) | → | JumpROM function "Set Unrestricted" exits restricted mode. |
| 7 | Get Status<br>'Write' (0x00) + 'S' (0x53) | → | Master requests the value of the status byte. |
| 8 | 'Read' (0x1) | → | Once the ACK is received, a read operation must be initiated by the master... |
| | (0x50) | ← | ... and the response shows that the chip is not restricted anymore. It can now be programmed. |
| 9 | Set Clock Frequency<br>'Write' (0x00) + 'F' 0x0800 (0x46 0x08 0x00) | → | **(This step is required only if I²C communication @ 400kbps is desired)**<br>Change the clock frequency from 1.8MHz to 3.44MHz by setting A_CLK_CTRL to 0x8, and leaving D_CLKSEL_CFG at the default 0x0 value. Use different register values to switch to an external clock. See the *Hardware Reference Manual*. |
| 10 | | Master | The I²C master can now change its I²C clock frequency to 400kbps. |
| 11 | Until ACK:<br>Get Status<br>'Write' (0x00) + 'S' (0x53) | → | Since clock settings were changed on BelaSigna 250, this polling loop ensures that communication is re−established. |

**Table 3. INITIALIZE AND UNLOCK BELASIGNA 250 THEN CHANGE THE CLOCK FREQUENCY**

| # | Command | Master ↔ Slave | Description |
|---|---|---|---|
| 12 | 'Read' (0x01) | → | Once the ACK is received, a read operation must be initiated by the master... |
| | (0x50) | ← | ... and the response shows that the chip is still unrestricted and that communication is working properly. |

### Downloading Object Code

The $I^2C$ interface is now operating properly, and the system is ready to start the process of downloading the object code to the three different memory spaces (X, Y and P) of BelaSigna 250.

When developing software for BelaSigna 250, various file formats can be generated and used, depending on the situation. The simplest way is to use the .o file format and a utility that ON Semiconductor can provide which automatically converts the .o file into a C−Header file (as described later in this document). Other options are also available to parse the object file and store it in the host code.

If the language used to develop the $I^2C$ bootloading application is Python, then the ON Semiconductor "absolute_file" module (absolute_file.py) already has all of this logic built−in. All communication modules provided by ON Semiconductor are available in a product called the CTK Developer Kit (see ON Semiconductor's BRD8070/D for more information).

The CTK libraries can be used to develop PC software in Python or other languages like C++. However, these are only usable when you are using a PC and the Communication Accelerator Adaptor (CAA) to communicate with BelaSigna 250's $I^2C$ port. When developing embedded microcontroller software to communicate with BelaSigna 250, you cannot use these CTK libraries. In this case the low−level communication functions must be implemented using the microcontroller's $I^2C$ master or GPIO functionality. Alternative solutions and support can be provided by ON Semiconductor to implement the $I^2C$ protocol at this low level.

### Downloading a Sample Application

The following example assumes the use of the C conversion utility applied on a dummy object file. It shows the structure of the generated C−header file, as well as the associated $I^2C$ commands needed for the transfer to BelaSigna 250.

```
#define DOWNLOAD_BLOCK_COUNT 5
struct DataBlock {
    enum MemorySpace memspace;
    unsigned short base;
    unsigned short wordCount;
    unsigned short checksum;
    unsigned char *formattedData;
} downloadBlocks[5] = {
        { 4, 0x1000, 0x00c2, 0xebe4, downloadData0 },
        { 0, 0x4000, 0x0044, 0xb67a, downloadData1 },
        { 0, 0x4180, 0x0040, 0xea0d, downloadData2 },
        { 0, 0x4200, 0x0010, 0x3c0f, downloadData3 },
        { 2, 0x0100, 0x0010, 0x168c, downloadData4 },
};
```

The above code snippet presents a summary of the data that are parsed in the .o file. Every contiguous block of data is declared as a unique block of data, and we can see that there are 5 blocks. All the blocks can be identified by their memory space identifier (0 for X, 2 for Y, or 4 for P), the base address in the destination memory bank, the number of words to be transferred and a checksum. The last element points to the actual data, as can be seen below, where the third data block is shown:

```
unsigned char downloadData3[] = {
    0x57,
    0xff, 0x17,
    0xfd, 0x0b,
    0xfa, 0xa9,
    0xf8, 0x1c,
    0xf5, 0xec,
    0xf5, 0x00,
    0xf6, 0x88,
    0xfb, 0xb9,
    0x05, 0x7f,
    0x14, 0x26,
    0x27, 0x1e,
    0x3c, 0xe9,
    0x53, 0x41,
    0x67, 0x6a,
    0x76, 0xb3,
    0x7e, 0xf1,
};
```

This data block contains the actual words to be transferred. For efficiency, the data is prefixed to the actual command (0x57, equivalent to 'W' in ASCII), that is used to transfer the data block through the $I^2C$ interface. Each data word is broken into two bytes: bits 15−8 first, followed by bits 7−0. The checksum for each data block is the sum of the data words, plus the memory space identifier and the base address of the block. For robustness, large blocks of data are broken into smaller blocks that can be transferred over the $I^2C$ interface more reliably.

The $I^2C$ procedure to download this fourth block is presented in the following table.

**Table 4. DOWNLOAD SECTION AND VERIFY CHECKSUM**

| # | Command | Master ↔ Slave | Description |
|---|---|---|---|
| 1 | Set Memory Block Pointer<br>'Write' (0x00) + 'M' (0x4D) 0x42 0x00 0x00 0x10 0x00 | → | This prepares for a 16−word transfer to be copied in the X Memory at address 0x4000<br>♦ 0x4D ('M' in ASCII) means Set Memory Pointer<br>♦ 0x42 0x00 means starting address<br>♦ 0x00 0x10 means 16 words<br>0x00 means MEM_SPACE is X |
| 2 | Write to Memory<br>Write' (0x00) + 'W' (0x57) 0xFF 0x17 ... 0x7E 0xF1 | → | This tells the I²C handler that the data words are coming.<br>♦ 0x57 ('W' in ASCII) means Write<br>♦ 0xFF 0x17 is the first data word<br>All subsequent words to be sent<br>♦ 0x7E 0xF1 is the last data word<br>There are 16 words to be sent (0x10) |
| 3 | 'Read' (0x1) | → | Once the transfer is finished, a read operation must be initiated by the master... |
| | (0x3C 0x0F) | ← | ... and the Write command automatically returns a checksum. The master application compares the returned checksum to that contained in the header file |

The above method must be applied successfully to all the sections in the header file. If a checksum mismatch is detected, the last data block can be retransmitted.

### Running the Application

**Table 5. RUN APPLICATION FROM PROGRAM MEMORY**

| # | Command | Master ↔ Slave | Descriptie |
|---|---|---|---|
| 1 | Start Application<br>'Write' (0x00) + 'G' (0x47) 0x10 0x00 | → | Place the Program Counter at address P:0x1000, and start the application from this entry point. |

After running this command, the bootloading process is complete; BelaSigna 250 has correctly received its object code, and is freely executing it. From this point, BelaSigna 250 is not executing its I²C handler any more, and consequently, it will not respond to ROM−based I²C commands until its next reboot. Any subsequent communication with the I²C interface has to be developed as part of the DSP application with its own I²C handler.

### I²C COMMAND INTERFACE TO CONTROL THE APPLICATION

The application is now running on BelaSigna 250, and as mentioned above, any subsequent communication with the I²C interface has to be developed as part of the DSP application. ON Semiconductor recommends changing BelaSigna 250's I²C slave address to a value that differs from the general call address, and for clarity, defining application−specific I²C commands with different identifiers than those defined in the ROM−based protocol.

Since I²C hardware exists between the host processor and BelaSigna 250, it is often very useful to implement a specific I²C control interface between the two devices. There are many configuration options within BelaSigna 250 that can be controlled by the host processors. These configuration commands can include the following functions and more:

- Apply a software reset to BelaSigna 250 (restart the application from entry point)
- Get status information on the application
- Change processing modes
- Enable/disable algorithms
- Select input channels
- Select input preamplifier gains
- Select output stage
- Apply master volume
- Control sleep mode
- Control algorithm parameters

ON Semiconductor can provide implementation examples of such control interfaces, as well as the assembly software framework. This enables a much faster implementation and allows developers to concentrate on the core signal processing algorithms, instead of spending time implementing such an interface with all the associated consequences.

I²C developed by Philips Semiconductor which is now called NXP.

BelaSigna is a registered trademark of Semiconductor Components Industries, LLC (SCILLC).
Bluetooth is a registered trademark of Bluetooth SIG.



## PUBLICATION ORDERING INFORMATION

**LITERATURE FULFILLMENT**:
Literature Distribution Center for ON Semiconductor
P.O. Box 5163, Denver, Colorado 80217 USA
**Phone**: 303−675−2175 or 800−344−3860 Toll Free USA/Canada
**Fax**: 303−675−2176 or 800−344−3867 Toll Free USA/Canada
**Email**: orderlit@onsemi.com

**N. American Technical Support**: 800−282−9855 Toll Free USA/Canada
**Europe, Middle East and Africa Technical Support:**
Phone: 421 33 790 2910
**Japan Customer Focus Center**
Phone: 81−3−5773−3850

**ON Semiconductor Website**: **www.onsemi.com**

**Order Literature**: http://www.onsemi.com/orderlit

For additional information, please contact your local Sales Representative