RESTOP: Retaining External Peripheral State in Intermittently-Powered Sensor Systems

USER MANUAL

Alberto R. Arreola
UNIVERSITY OF SOUTHAMPTON
Revised: 8th January 2018

1. CONTENT

2. Library 2

2.1. Config.h 2

2.2. RESTOP_func.h 3

3. Generic Functions 3

3.1. RESTOP_write 3

3.2. RESTOP_read 4

3.3. RESTOP_strobe 4

3.4. RESTOP_restore 4

3.5. RESTOP_VR_adj 4

2. LIBRARY

In order to incorporate the functionality of RESTOP, it is needed to include two headers files in the main application:

a) #include "Config.h"
b) #include "RESTOP_func.h"

2.1. Config.h

In this file, the user defines the following parameters:

- **Table**. This parameter (`#define` Table *size*) is used to define the size of the *Instruction History Table*. Here, *size* indicates the number of locations where the peripheral instructions will be saved.
- **Cap**. This parameter (`#define` Cap *value*) is needed to set the capacitance value used as energy buffer. This value is used by RESTOP to calculate and update the restore threshold (V_R) in the snapshotting routine of the transient computing approach.
- **Vmin**. This parameter (`#define` Vmin *voltage*) is the minimum operating voltage of the system. It is also needed by RESTOP to calculate and update V_R.
- **reg_reset.** This is an array where the user has to define the reset register of each attached peripheral. The format is as follows:

  reg_reset [] = {{reg_peripheral1}, {reg_peripheral2}, {reg_peripheralN}};

  The number of positions depends on the number of attached registers. If only one peripheral is needed, the user can remove the unused sections or just ignore them.
- **cmd_write.** This array is needed to define the *command* value for those peripherals that need this parameter before a *write* operation. The format is:

  cmd_write [] = {{cmd_peripheral1}, {cmd_peripheral2}, {cmd_peripheralN}};

  The number of positions depends on the number of attached registers. If no *command* is needed, all the array sections have to be filled with zeros, e.g. two peripherals are attached but none of them needs a *command*:

  cmd_write [] = {{0x0}, {0x0}};

- **cmd_read.** The function and format of this array are similar than the previous one but this is for *read* operations.
- **i2c_add.** In this array, the user set the address for the I^2C peripherals. The format is as follows:

  i2c_add [] = {{add_peripheral1}, {add_peripheral2}, {addr_peripheralN}};

  Depending on the number that corresponds to the I^2C peripheral is where the address has to be set and the others have to be filled with zeros, e.g. there are two attached peripherals, but the second is the I^2C one, the array would be filled as follows:

  i2c_add [] = {{0x0}, {0xEE}};

- **pwr_pi.** Here, the user has to set the power consumption of the MCU when issuing an instruction through SPI and I^2C protocols. The format is:

pwr_pi [] = {{W_SPI}, {W_I^{2}C}};

The values have to be introduced in Watts. The first location is for the power consumption when issuing instructions to SPI and the second for I^{2}C peripherals. If only one type of protocol is used, the other must be filled with zeros, e.g. a system where only an I^{2}C peripheral is attached, the array would be:

pwr_pi [] = {{0}, {0.002}};

- **Time_spi.** Here, the user set the time (in seconds) spent by the MCU to issue SPI peripheral instructions of different number of bytes. The format is described below:

  Time_spi [] = {{time_3params}, {time_2params}, {time_1param}};

  From left to right, each position corresponds to the time taken to issue three, two and one parameters per instruction, respectively (1 parameter = 1 byte). If the attached peripheral does not operate with the three different number of parameters, the unused sections have to be filled with zeros, e.g. if a peripheral only operates with two parameters per instruction, the array would be filled as follows:

  Time_spi [] = {{0}, {0.000120}, {0}};

- **Time_i2c.** This array follows the same format than *Time_spi* but this is for I^{2}C peripherals.

  Time_i2c [] = {{time_3params}, {time_2params}, {time_1param}};

2.2. RESTOP_func.h

This file contains the declaration of the functions required by RESTOP to save and issue each peripheral instruction. The user is not expected to modify this file.

3. GENERIC FUNCTIONS

RESTOP is composed by three generic functions to perform *read* or *write* operations on the peripheral, which are *RESTOP_read*, *RESTOP_write* and *RESTOP_strobe*. RESTOP also includes two additional functions to restore the peripheral state (*RESTOP_restore*) and to adjust the restore threshold (V_R) before a power failure occurs (*RESTOP_VR_adj*). These functions are described in this chapter.

3.1. RESTOP_write

This function performs *write* operations on the peripheral. It is composed by six parameters:

- **RESTOP_write** (Prv, ID, Register, Value, Burst, Protocol)

The first parameter is to define the criteria of *Not-save* (‘d0), *Save* (‘d1), *Save-but-replace* (‘d2), *Save and preserve* (‘d5), *Save-but-replace* and *Preserve* (‘d6). The second parameter (ID) is to define the peripheral to which the instruction will be issued (the value goes from 1 to N). The parameters *Register* and *Value* are each one byte, corresponding to the register width of typical digital interface peripherals. *Burst* is a flag to indicate if the instruction is one of a *write-burst* operation. From the first instruction to the N-1, this flag is set to 1. The last one from the burst operation is set to zero. Below, it is shown an example of how to use the *burst* flag when three burst instructions are issued:

1. RESTOP_write (1, 1, 0x0E, 0x2A, 1, 0);
2. RESTOP_write (1, 1, 0x00, 0x2B, 1, 0);
3. RESTOP_write (1, 1, 0x00, 0x2C, 0, 0);

In the first instruction, the Register where the values will be written is set. It is also enable the *burst* flag. In the second instruction, instruction, the Register parameter is set to zero, and only the value to be written and the *burst* flag are set. Finally, in the last *burst* instruction, the flag is set to zero.

The last parameter of this function (*Protocol*) is to define the protocol of the peripheral (0=SPI; 1=I^2C).

3.2. RESTOP_read

This function performs *read* operations on the peripheral. The parameters are the same as in the previous function, except for the *Value* that is not needed for *read* operations. This function returns the read value from the peripheral.

- `uint8_t` **RESTOP_read** (Prv, ID, Register, Burst, Protocol)

3.3. RESTOP_strobe

This function performs *write* operations that, unlike RESTOP_write (), executes single byte instructions. The criteria for the parameter is the same than in the previous functions.

- **RESTOP_strobe** (Prv, ID, Register, Protocol)

3.4. RESTOP_restore

This function is in charge of restoring the peripheral state. It has to be placed at the end of the restoring routing of the transient computing approach. The function does not receive or return any value.

- **RESTOP_restore** ()

3.5. RESTOP_VR_adj

This function dynamically calculates the value for V_R depending on the number of peripheral instructions that are saved before a power failure. It has to be placed at the beginning of the snapshotting routine of the transient computing approach and returns the new value for V_R, which has to substitute the one previously set.

- **float** $\mathbf{V_R}$ **= RESTOP_VR_adj** ()