1. Protocolli

1.1. Protocollo SPI

Il protocollo SPI è un protocollo sincrono *full-duplex*, dove i dati sono sincronizzati dai fronti di salita del clock. Esso nella vesrione più usata presenta 4 cavi:

- **CS**: *Chip Selector* senale per selezionare il subnode, il segnale di attivazione tipicamente è un segnale *LOW*;
- **SCLK**: *SPI Clock* clock di trasmissione;
- **MOSI**: *Master Output Slave Input* (*Main Output Subnode Input*) canale di trasmissione dal main al subnode.
- **MISO**: *Master Input Slave Output* (*Main Input Subnode Output*) canale di trasmissione dal subnode al main;

A differenza dell'I^2 C, **SPI** presenta un solo *master* (o *main*),colui che genera il *clock*, e può lavorare a frequenze più elevate. Una configurazione tipica è rappresentata a Figura 1

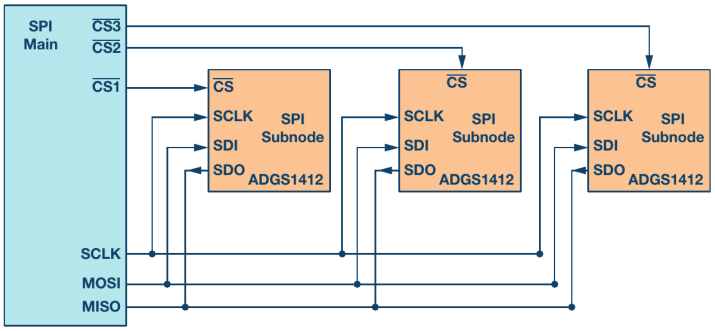


Figura 1: Esempio di collegamento tra il *main* e tre *subnodes*

Per avviare una comunicazione è necessario avviare il clock e impostare a *LOW* il relativo *CS*. Il *clock* determina la frequenza di trasmissione in particolare è presente un registro scorrevole il quale scorre ad ogni fronte si salita (o di discesa[1]) del *clock* trasmettendo il bit sul canale **MISO** o **MOSI**.

La **polarità** (*CPOL*) e la fase del **clock** (*CPHA*) caratterizzano la comunicazione:

- **Mode 0**: $CPOL = 0$ e $CPHA = 0$
- **Mode 1**: $CPOL = 0$ e $CPHA = 1$
- **Mode 2**: $CPOL = 1$ e $CPHA = 0$
- **Mode 3**: $CPOL = 1$ e $CPHA = 1$

In particolare quando la polarità è pari a 0 vuol dire che il *clock* in stato di *idle* sarà a livello logico *LOW*, *HIGH* viceversa. Mentre la fase indica su quale fronte si effettua il sample e di conseguenza lo shift del registro. Nel caso 0 è come se ci fosse una fase di 0° ovvero il samplig viene effettuato sul fronte che va da idle $\rightarrow \overline{\text{idle}}$, nel caso contrario va da $\overline{\text{idle}} \rightarrow$ idle, di conseguenza lo *shift* verrà effettuato al fornte successivo (ovvero a quello opposto del *sampling*). Ad esempio con *CPOL* = 1 il clock è in idle con valore *HIGH* quindi con *CPHA* = 0 il samplig si effettua sulla transizione $1 \rightarrow 0$, mentre nel caso *CPHA* = 1, $0 \rightarrow 1$.

1.2. Protocollo I^2 C

Il protocollo I^2 C utilizza due canali per la trasmissione:

- **SDA**: *Serial Data Line* utilizzato per inviare i dati;
- **SCL**: *Serial Clock Line* utilizzato dal *controller device* il quale ha lo scopo di sincornizzare di dati con il *target device*.

[1]Dipende dall'implementazione

Il *controller device* inizia e termina la comunicazione. Per indirizzare la trasmissione a uno specifico target si utilizza un **indirizzo unico** che identifica il target.

Esso supporta la comunicazione multipla tra diversi dispositivi e anche tra diversi controllori che inviano segnali di *send* e *recive* in modalità *half-duplex*. Le comunicazioni avvengono per pacchetti di byte. Esso ha diverse modalità di comunicazione, Figura 2.

Il layer fisico del I^2C è composto da due cavi **SDA** e **SCL** con una resistenza di *pull-up* per entrambe che va a V_{DD}. La presenza delle resistenze di *pull-up* è data dalla presenza della connessione, per ogni dispositivi, di *open-drain* al bus, Figura 3. Da notare le resitenze di *pull-up* hanno due impatti sulla trasmissione:

- **Velocità di risalita**: le linee *SDA* o *SCL* per «tornare su» impiegano un certo tempo (costante RC) che dipende dalle *capacità parassite* e dalla resistenza di *pull-up*. Maggiore è la resistenze più lenta sarà la risalita;
- **Consumo di potenza**: Con una resistenza di *pull-up* più piccola la potenza necessaria aumenterà con la coseguenza di maggiori consumi.

Valori tipici per le resistenze sono tra $1K\Omega$ e $10K\Omega$.

I²C Mode	Maximum Bit Rate
Standard-mode	100kbps
Fast-mode	400kbps
Fast-mode Plus	1Mbps
High-speed mode	3.4Mbps
Ultra-Fast mode	5Mbps

Figura 2: Diverse modalità di trasmissione del protocollo I^2C

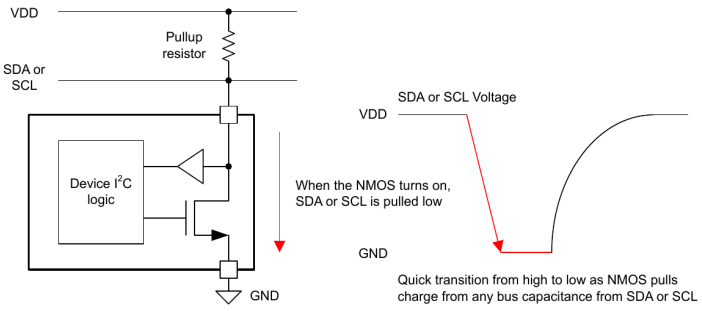


Figura 3: Connessione *open-drain*.

L'inizio e la fine di una comunicazione viene effettuata tramite una specifica sequenza. In particolare per iniziare la comunicazione il *controller device* deve prima mettere *SDA = LOW e successivamente* SCL = *LOW* per la chiusura della comunicazione: *SCL = HIGH* e poi *SDA = HIGH*. Come mostrato in Figura 4a.

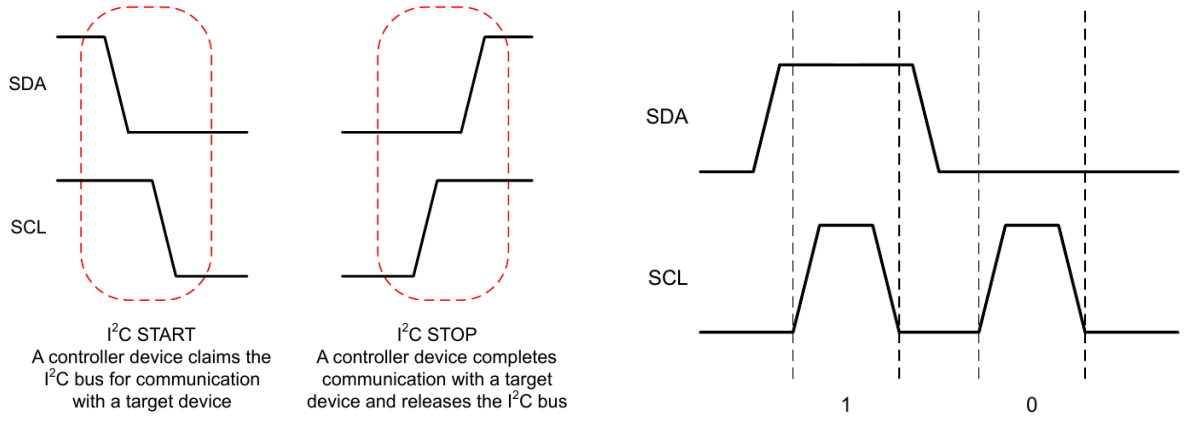
La trasmissione del singolo bit avviene secondo lo schema riportato a figura Figura 4b.

Una comunicazione avviene sempre con un ordine definito, in particolare ogni volta si invia un byte (8 bit) alla volta più un bit di *ACK*, eccetto per *start* e *stop*:

1. **Avvio** della comunicazione;
2. **Invio** dell'indirizzo di destinazione (7 bit) e un bit se il controllore deve leggere (*LOW*) o scrivere (HIGH), in seguito un bit di *ACK*: se pari a 0 allora il messaggio è stato ricevuto altrimenti no.
3. A questo punto ci sono solo **byte** di dati che possono essere inviati o solo dal *controller device*, caso scrittura, o solo dal *target device*, nel caso di lettura, essi sono sempre seguiti da un *ACK*.

4. **Chiusura** della comunicazione

Esempio a Figura 5.



(a) Start e Stop della comunicazione I^2 C (b) comunicazione di uno e zero

Figura 4: Linee del protocollo I^2 C

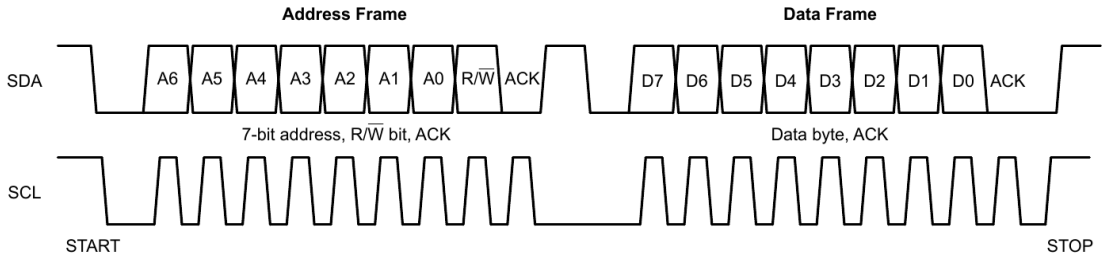


Figura 5: Esempio di comunicazione I^2 C.

2. STM32

2.1. «PWM input mode» SMT32

Utilizziamo 2 canali del timer:

- Il primo misura il periodo della PWM in ingresso
- Il secondo misura *high time* della PWM

2.2. EXTI dai pulsanti

Per prima cosa individuare dove è connesso il pulsante, ad esempio *PC12*; Successivamente attivare nella sezione *NVIC* le relative linee come interrupt, inoltre nella sezione *GPIO* individuare il pin (PC12 in questo caso) e impostare su quale edge generare l'interrupt. **ATTENZIONE**: Se il pulsante non presenta nessuna rete di pull-up, o pull-down, impostare nella *GPIO* una rete di pull-up o pull-down per quel pin, in questo caso PC12.

3. Altre informazi

- **USART** → protocollo comunicazione come il seriale
- **stlink debugger** è la parte che fa debug
- **GPIO** (*General Purpouse Input Output*), può esserci per segnali come *timer*.

4. BLE

Bluetooth è una tecnologia per la comunicazione comunicazione wireless **low-power**, **short-range**, **WPAN** (wireless personal area network). Essa è principalmente divisa in due categorie:

- *Bluetooth Classic*: Rappresenta una prima versione del bluetooth con maggiore *trough-put* di dati, con 79 diversi canali in cui su 32 si può effettuare il *discovery*[2].
- *Bluetooth Low Energy* o *BLE*: Utilizzato per comunicazioni poco frequenti, come dati di sensori, presenta 40 canali e può fare *discovery* su 3 di essi.

Una tipica applicazione consiste in due dispositivi: periferica (*peripheral*) e il centrale (*central*). Per stabilire la connessione il *central* esegue uno *scan* per ricercare *peripheral* disponibili, nel mentre la *peripheral* si pone nello stato di *advertising*.

Una volta stabilita la connessione i due dispositivi possono iniziare la comunicazione.

4.1. Bluetooth in STM32WB

L'architettura del BLE in STM32WB è composta da due CPU:

- La prima, CPU1 (ARM M4), è per il nomale utilizzo della board.
- La seconda, CPU2 (ARM M0+), è utilizzata per le trasmissioni wireless, come: BLE, Thread e MAC 802_15_4.

La comunicazione tra le due CPU è fatto via «mailbox». Tutte le periferie condivise da entrambe le CPU sono protette da dei semafori a livello hardware, *HSEM*, Figura 6.

La memoria Flash utilizzata dai due core è la stessa, essa è divisa ed è protetta da accessi. In particolare solo la CPU1 può accedere alla area di memoria della CPU1, lo stesso vale per la CPU2 e la sua area di memoria.

Semaphore	Purpose
Sem0	RNG - All registers
Sem1	PKA - All registers
Sem2	Used to share between CPU1 and CPU2 the capability to write/erase data in FLASH
Sem3	RCC_CR RCC_EXTCFGR RCC_CFGR RCC_SMPSCR
Sem4	Clock control mechanism for the Stop mode implementation
Sem5	RCC_CRRCR RCC_CCIPR
Sem6	Used by CPU1 to prevent CPU2 from writing/erasing data in flash memory
Sem7	Used by CPU2 to prevent CPU1 from writing/erasing data in flash memory
Sem8	Ensures that CPU2 does not update the BLE persistent data in SRAM2 when CPU1 is reading them
Sem9	Ensures that CPU2 does not update the Thread persistent data in SRAM2 when CPU1 is reading them

Figura 6: *HSEM* per il BLE.

4.1.1. Sequencer

Il *sequencer* ha il compito di eseguire le funzioni registrate una dopo l'altra, supporta fino a 32 funzioni, è possibile eseguire le funzioni su richiesta e disabilitare l'esecuzioni.

Fornisce quindi un semplice background per schedulare le funzioni. Implementa in modo sicuro la modalità in *low-power-mode* quando il sequencer non ha nessun task da eseguire I passi principali per utilizzarlo sono:

1. Creare aggiungere un nuovo elemento negli enumerativi: `CFG_Task_Id_With_HCI_Cmd_t` all'interno del file `app_conf.h`. Per task che non fanno parte del BLE è necessario creare nuovi elementi dentro: `CFG_Task_Id_With_NO_HCI_Cmd_t`;

[2]Con _discovery_ si intende il processo di ricerca di altri dispositivi bluetooth

2. Registrare la funzione al sequencer: `UTIL_SEQ_RegTask()` , come secondo parametro `UTIL_SEQ_RFU`;
3. Avviare il sequencer: `UTIL_SEQ_Run()` (SE NON GIÀ FATTO);
4. Chiamare la funzione `UTIL_SEQ_SetTask()` quando serve che una funzione venga eseguita.

Per Ulteriori informazioni consultare: AN5289

A Figura 7 sono presenti le diverse funzioni.

Function	Description
`void UTIL_SEQ_Idle( void );`	Called (in critical section - PRIMASK) when there is nothing to execute.
`void UTIL_SEQ_Run( UTIL_SEQ_bm_t mask_bm )`	Requests the sequencer to execute functions that are pending and enabled in the mask mask_bm.
`void UTIL_SEQ_RegTask(UTIL_SEQ_bm_t task_id_bm, uint32_t flags, void (*task)( void ))`	Registers a function (task) associated with a signal (task_id_bm) in the sequencer. The task_id_bm must have a single bit set.
`void UTIL_SEQ_SetTask( UTIL_SEQ_bm_t task_id_bm,`	Requests the function associated with the task_id_bm to be executed. The task_prio is evaluated by the sequencer only when a function has finished. If several functions are pending at any one time, the one with the highest priority (0) is executed.
`void UTIL_SEQ_PauseTask( UTIL_SEQ_bm_t task_id_bm )`	Disables the sequencer to execute the function associated with task_id_bm.
`void UTIL_SEQ_ResumeTask( UTIL_SEQ_bm_t task_id_bm )`	Enables the sequencer to execute the function associated with task_id_bm.
`void UTIL_SEQ_WaitEvt( UTIL_SEQ_bm_t evt_id_bm )`	Requests the sequencer to wait for a specific event evt_id_bm and does not return until the event is set with UTIL_SEQ_SetEvt().
`void UTIL_SEQ_SetEvt( UTIL_SEQ_bm_t evt_id_bm )`	Notifies the sequencer that the event evt_id_bm occurred (the event must have been first requested).
`void UTIL_SEQ_EvtIdle(UTIL_SEQ_bm_t task_id_bm, UTIL_SEQ_bm_t evt_waited_bm)`	Called while the sequencer is waiting for a specific event.
`void UTIL_SEQ_ClrEvt( UTIL_SEQ_bm_t evt_id_bm )`	Clears the pending event.
`UTIL_SEQ_bm_t UTIL_SEQ_IsEvtPend( void )`	Returns the evt_id_bm of the pending event.

Figura 7: Funzioni per il *sequencer*.

4.1.2. Timer Server

Fornisce la possibilità di creare diversi timer in condivisione con l'RTC wake-up timer. È possibile creare dei timer in *sigle shot* oppure in *repeated mode*, metterli in pausa, eliminarli e ravviarli con diversi timeout con un valore compreso tra 1 e $2^{32} - 1$ ticks. Quando un timer termina notifica l'utente e si riavvia (se impostato come *repeated*).

Fare attenzione che se l'operazione all'interno della funzione di *callback* è richiede del tempo, è meglio eseguire la funzione dal **sequencer**

Per utilizzare il timer è necessario:

1. Creare il timer con `HW_TS_Create(CFG_TIM_PROC_ID_ISR , ...)`, e verificare il valore di ritorno che indica se è stato creato correttamente
2. (NON È VERO SU PUÒ UTILIZZARE QUELLA GIÀ DEFINITA) Ridefinire la funzione `HW_TS_RTC_Int_AppNot(...)`
3. Avviare il timer `HW_TS_Start(...)`

Problematiche riscontrate:

- MCU si bloccava in un loop continuo senza mai uscire -> Verificare se è attivo il WakeUp e l'RTC wake-up NVIC in CubeMX.
- La quando viene chiamata la funzione `HW_TS_RTC_Int_AppNot(...)` richiama la funzione handler che è un puntatore nullo (0x0) -> Fare attenzione che si utilizza sempre lo stesso ID quando viene creato il timer quando viene avviato, in particolare quando viene creato il timer tramite `HW_TS_Create(...)`, `pTimerId` è un puntatore all'**ID**.

Per fare un timer da X secondi utilizzare la seguente formula: $\frac{X \cdot 10^6}{\text{CFG\_TS\_TICK\_VAL}}$

Per Ulteriori informazioni consultare: AN5289 Sezione 4.5

4.1.3. Low Power Manager

Fornisce un'interfaccia per un massimo di 32 utenti e permette di eseguire computazioni nella modalità a più basso consumo disponibile al sistema. Le features disponibili sono:

- Stop mode e Off mode (standby e shutdown);
- Esecuzione e selezione della modalità low-power;
- Funzioni di callback quando si entra o si esce dalle modalità low-power

Il low-power mode per la CPU2 è fatto in modo automatico, in particolare il firmware imposta in modo autonomo la configurazione migliore di low-power mode.

4.1.4. FUOTA (Firmware Update On-The-Air)

Il FUOTA è una applicazione *standalone* che permette di scaricare: un nuovo CPU2 wireless stack, applicazioni per la CPU1 oppure file binari di configurazione.

Richiede che i primi sei settori della memoria flash dell'applicazione non vengano mai eliminati, poiché sarà li dove l'applicazione FUOTA sarà installata, Figura 8.

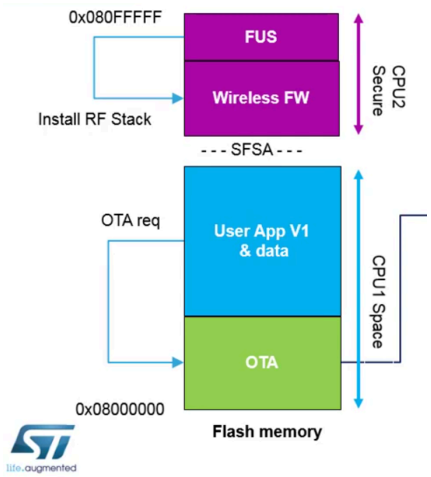


Figura 8: Divisione della memoria tra le due CPU.

Come detto in precedenza, le due CPU condividono lo stesso banco di memoria, la quale è divisa da una barriera di sicurezza. La sezione per la CPU1 parte dal basso con la memoria per OTA a seguire fino alla barriera c'è l'applicazione dell'utente. Dopo la barriera è presente la zona di memoria per la CPU2.

L'applicazione OTA, è semplicemente un servizio BLE che espone 3 caratteristiche:

1. *Base Address* (WRITE): serve per indicare l'indirizzo base dove effettuare il flash dei dati.
2. *OTA raw data* (WRITE): caratteristica per il trasferimento del binario, e.g. applicazione o file di configurazione.
3. *file upload confimation* (INDICATION): caratteristica per che indica al che il trasferimento è andato a buon fine.

Ci son dei passi necessari che si fare nella *User Application* per integrare l'OTA:

1. Nell'applicazione BLE è necessario inserire una caratteristica: *OTA Reboot request*
2. Il secondo passo consiste nel cambiare l'advertising.

Primo passo: facendo un esempio con l'applicazione *Heart Rate*, Figura 9, per integrare il servizio OTA in questa applicazione è necessario aggiungere la caratteristica *OTA reboot request*. Quindi quando il client scrive su questa caratteristica richiede di effettuare il reboot per avviare l'applicazione OTA, è possibile anche definire l'area di memoria di partenza nella quale effettuare la modifica.

Secondo passo: Nella modalità advertising è possibile impostare una *feature mask*, la quale fornisce informazioni su quale *features* implementa quel dispositivo. In particolare la maschera ha a disposizione 32 bit (4 byte) ogni bit ha un valore specifico, Figura 10. Quindi ponendo il bit 13 a uno, si indicherà che l'applciazione BLE ha la possibilità di eseguire l'OTA reboot.

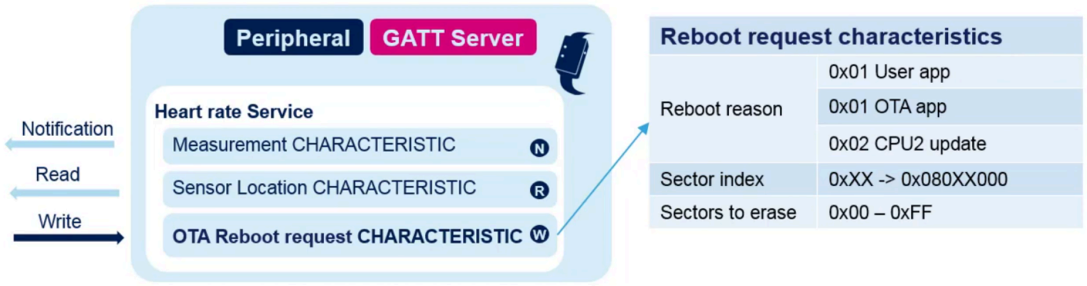


Figura 9: Esempio BLE heart rate.

Bit	31	30	29	28	27	26	25	24
Feature	RFU	ADPCM Sync	Switch	Direction of arrival	ADPC Audio	MicLevel	Proximity	Lux

Bit	23	22	21	20	19	18	17	16
Feature	Acc	Gyro	Mag	Pressure	Humidity	Temperature	Battery	Second Temperature

Bit	15	14	13	12	11	10	9	8
Feature	CO Sensor	STM32WB Thread Reboot bit	STM32WB OTA Reboot bit	SD Logging	Beam forming	AccEvent	FreeFall	Sensor Fusion Compact

Bit	7	6	5	4	3	2	1	0
Feature	Sensor Fusion	Motion intensity	Compass	Activity	Carry Position	Proximity Gesture	MEMS Gesture	Pedometer

Figura 10: Valori per il *feature mask*.

All'avvio, la CPU1 cerca se c'è una richiesta dell'OTA di installazione di una nuova applicazione, se non è presente viene bypassata e si avvia l'applicazione d'utente

Di seguito vengono descritti i seguenti passi per l'aggiornamento della CPU1:

1. Una volta eseguita la connessione GAP, il dispositivo remoto GATT esegue una scansione dei servizi e delle caratteristiche.
2. Con l'obiettivo di eseguire lo switch all'applicazione FUOTA, il dispositivo remoto scrive la richiesta di *Reboot* sulla caratteristica con le relative informazioni riguardanti le modalita di *boot* e i settori da eliminare.
3. Il dispositivo si disconnette per ravviare sull'applicazione che permette di eseguire la FUOTA.
4. I settori dell'applicazione vengono eliminati seguendo le istruzioni date precedentemente, inoltre l'applicazione FUOTA GATT si avvia ponendosi in *advertising mode*.
5. La connessione viene stabilita con il dispositivo in ricerca del servizio FUOTA
6. La caratteristica di indirizzo base è utilizzata per iniziare il nuovo caricamento binario.
7. Tutti i dati vengono trasferiti tramite la caratteristica *raw data* e programmati direttamente nella memoria flash.
8. La fine del trasferimento è confermata dalla caratteristica base di indirizzo.
9. La conferma della ricezione del file indicata dalla caratteristica di conferma del file.
10. A questo punto l'applicazione FUOTA esegue una verifica di integrità del binario e ravvia per avviare la nuova applicazione caricata.
11. Se l'integrità dell'applicazione non è assicurata, il settore dell'applicazione vine eliminato per ravviare nuovamente l'applicazione FUOTA.

A figura Figura 11 è riportato un esempio dei passi per eseguire il FUOTA.

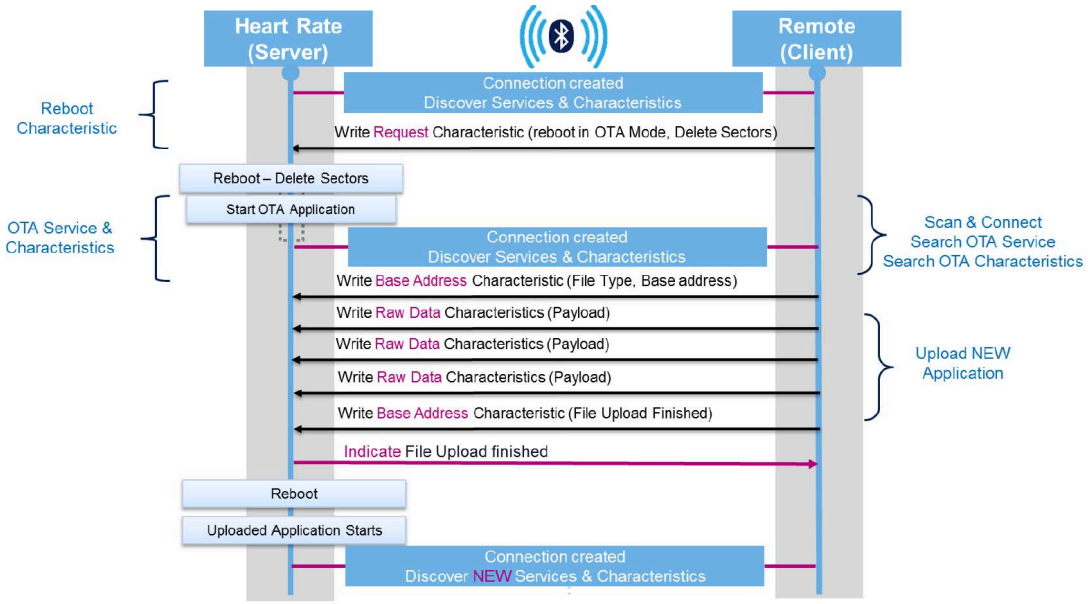


Figura 11: Esempio FUOTA.

4.2. Attivazione BLE STM32

Per abilitare il middleware del bluetooth è necessario attivare diverse funzionalità necessarie per far funzionare il BLE:

1. Per prima cosa andare dentro **RCC** (Resets and Clock), e attivare **HSE** (High Speed External crystal da 32MHz) e **LSE** (Low Speed External crystal 32′768Hz). Il primo clock è utilizzato per generare il di *carrier* nel livello fisico del BLE, mentre il secondo è utilizzato per sincronizzare i diversi processi.
2. In secondo si deve attivare il **HSEM**, Sezione 4.2.1.1.
3. Successivamente andare dentro **IPCC** (*inter processor communication controller*, Sezione 4.2.1.2) attivarlo e nelle impostazioni per **NVIC** attivare anche:
   - *IPCC RX occupied interrupt*
   - *IPCC TX free interrupt*
4. Attivare il **RTC**, Sezione 4.2.1.3, porre il *wake-up* come «Internal WakeUp» e successivamente spuntare nella sezione **NVIC** l'interrupt.
5. Infine è necessario attivare la **RF**, Sezione 4.2.1.4 (dentro *Connectivity*)

Ora è possibile cercare dentro «*Middleware and Software Pack*» il middleware: «STM32_WPAN» e spuntare il BLE. In questo caso dentro le impostazioni del BLE è attivata l'opzione «*Custom P2P server*» per questo tutorial, dato che si vuole avere un controllo maggiore del BLE, lo si disattiva; mentre verrà attivando il «*Custom Template*».

Controllo del *Advertising*: È possibile controllare il contenuto del pacchetto per l'*advertising* nella sezione relativa. ad esempio si può modificare il nome del dispositivo modificando la variabile: `AD_TYPE_COMPLETE_LOCAL_NAME`

Creazione di un servizio: Per la creazione di un servizio andare nella sezione **GATT** (Generic Attribute Profile), e porre il numero di servizi che si vogliono creare. Successivamente indicare il nome del servizio, è possibile dare due nomi: *long name* e *short name*. Fatto ciò si creerà un nuovo *tab* con il nome del servizio, è possibile definire il UUID (*universally unique identifier*). Definire il nome della caratteristica e impostare la, o le, proprietà della caratteristica (`write`, `read`, `notify`, ecc.) in questo

esempio si imposta la proprietà di `write`. Inoltre si possono attivare o disattivare gli eventi a cui non si è interessati, in questo caso si tiene attivo solo l'evento: `GATT_NOTIFY_ATTRIBUTE_WRITE`

Le ultime modifiche da fare riguardano il clock (sezione *Clock Configuration*), in particolare impostare:

- **RTC** su **LSE**
- **RF** *WakeUp Clock* su **LSE**

Per informazioni dettagliate visionare questo link: STM32 P2P SERVER

4.2.1. Funzionalità/Periferiche utilizzate

Di seguito vengono descritte brevemente le periferiche attivate per utilizzare il bluetooth.

4.2.1.1. HSEM

Hardware Semaphore, è una periferia utilizzata per sincronizzare gli accessi dai due diversi *core* nelle periferie condivise.

4.2.1.2. IPCC

L'*inter processor communication controller* . Le i due diversi *core* nei processori *WB* comunicano attraverso della memoria condivisa (*SRAM2*) Figura 12. E la **IPCC** è responsabile del meccanismo di messaggi tra i due *core*, ad esempio può generare degli interrupt per avvisare che un messaggio è pronto o che è stato letto.

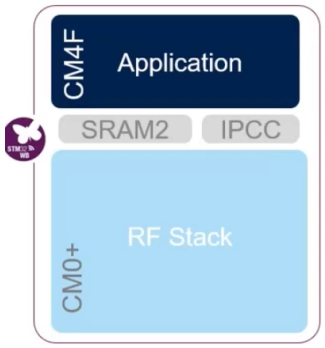


Figura 12: Schematico in cui si osserva **IPCC** e la shared memory.

4.2.1.3. RTC

L'**RTC** è un timer real time, esso fornisce il l'orario e la data attuale e ha la possibilità di attivare degli allarmi (interrupt). Esso può eseguire il wake-up del dispositivo dalle modalità low power

4.2.1.4. RF

Radio Frequency è il modulo che effettua la comunicazione wireless per il BLE.

4.3. Info caratteristiche

- **READ**: Quando viene creata una nuova caratteristica di *READ* assicurarsi che sia attivato, in CubeMX sezione `Characteristic GATT event`, l'evento relativo alla lettura: `GATT_NOTIFY_READ_REQ_AND_WAIT_FOR_APPL_RESP`