**REPORT HOME CHALLENGE #1**

Costa Giovanni 10735519

Polo Enrico 10528160

Salani Francesco 10748052

Github repository: https://github.com/Salani-Polimi/HomeChallenge1\_CostaPoloSalani.git

**SCOPO:** Implementare in TinyOS un sistema composto da tre motes (1-2-3), in grado di comunicare tra loro attraverso sistema radio. Ogni mote ha una sua frequenza di trasmissione, rispettivamente 1Hz, 3Hz e 5Hz. Ogni mote ha un counter associato che viene incrementato ogni volta che riceve un messaggio. Il pacchetto, inviato in BROADCAST, contiene il counter ed il sender id entrambi del mittente.

Ogni mote ha associati tre LEDs (implementazione telosb) che si accendono/spengono seguendo le regole:

* Se il pacchetto ricevuto proviene dal mote 1, il LED di riferimento è il rosso;
* Se il pacchetto ricevuto proviene dal mote 2, il LED di riferimento è il verde;
* Se il pacchetto ricevuto proviene dal mote 3, il LED di riferimento è il blu;
* Quando il valore nel counter del payload ricevuto è multiplo di dieci, tutti i LEDs vengono spenti;

# IMPLEMENTAZIONE

1. **PACCHETTO**

Il pacchetto è implementato tramite una struct all’interno del file .h. Esso contiene due interi: counter e sender id.

# INTERFACCE

Le interfacce utilizzate sono:

* + LEDs;
  + Boot;
  + Receive;
  + AMSender;
  + Tre Timer;
  + SplitControl as AMControl;
  + Packet;

# COLLEGAMENTI AppC.nc

Le componenti sono:

* + MainC, Hw1C as App, LedsC;
  + new AMSenderC (AM\_RADIO\_COUNT\_MSG);
  + new AMReceiverC (AM\_RADIO\_COUNT\_MSG);
  + new TimerMilliC() as Timer*#numbermote*;
  + ActivateMessageC;
  + SerialPrintfC;
  + Collegamenti interfacce;

# LOGICA C.nc

Il valore del sender id è dato dalla macro TOS\_NODE\_ID che cambia a seconda del mote di riferimento. Questa caratteristica è stata sfruttata:

* + In trasmissione attraverso uno switch per differenziare le diverse frequenze di trasmissione. Infatti con un controllo sul valore effettivo di TOS\_NODE\_ID possiamo sapere quale sia il mote di riferimento e associare il corretto timer. In questo modo ad ogni mote corrisponde uno ed un solo timer;
  + È stato creato un evento per ogni mote. In ogni evento viene assegnato il rispettivo payload, in particolare in sender id varia dipendentemente dal TOS\_NODE\_ID;
  + In ricezione attraverso l’utilizzo di uno switch per un controllo sul sender id contenuto nel payload del pacchetto ricevuto, è possibile conoscere quale sia il LED effettivo sul quale agire (on/off). Per controllare l’alternanza delle operazioni, sono state utilizzate tre variabili bool (TRUE next step da accendere e vcv) in maniera da controllare lo stato precedente dei LEDs di ogni mote. Prima dello switch è stato inserito un semplice controllo sul valore del counter: quando esso assume valore multiplo di dieci, tutti i LEDs vengono spenti e non si entra nello switch. In particolare il controllo è stato effettuato banalmente confrontando il valore del campo counter del pacchetto con dieci: in caso di uguaglianza tutti i LED vengono spenti. Poiché ogni mote ha il proprio counter associato e non può controllare quello degli altri, quando esso arriva ad undici riparte da uno. In questo modo è stato possibile implementare l’operazione di modulo. Per avere un controllo sul valore del campo counter del payload ricevuto, è stata utilizzata una printf con la quale viene stampato il valore attuale del counter del mote e, una volta che il campo counter del pacchetto ricevuto è pari a 10, viene stampato la notifica di avvenuto spegnimento dei LEDs;

*Nota bene: poiché i tre mote trasmettono a frequenze diverse, non è sempre possibile notificare il raggiungimento di counter uguale a dieci. Ad esempio dalla simulazione si nota che il mote uno ricevendo più frequentemente rispetto a quanto trasmette, non potrà notificare ogni variazione del counter. Una possibile soluzione può essere l’utilizzo di un interrupt che faccia notificare al mote ricevente successivo, il superamento del limite di dieci.*

A screenshot of a computer

Description automatically generated