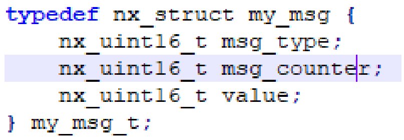
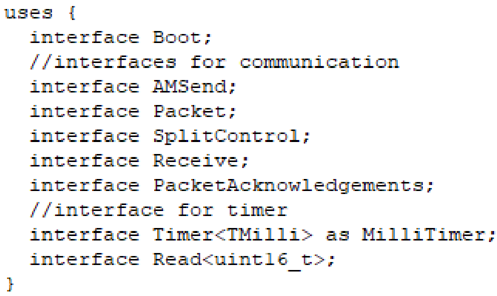
Anna -Giuseppe

First step: We modified the file sendAck.h defining the payload of the message with the needed variable



```
typedef nx_struct my_msg {
    nx_uint16_t msg_type;
    nx_uint16_t msg_counter;
    nx_uint16_t value;
} my_msg_t;
```

Second step: We implemented the needed interfaces for boot communication. We also added the timer interface for sending messages with a period of 1 sec. The last interface is used to take the random values. In the sendAckAppC we linked the interfaces with the corresponding components.



```
uses {
  interface Boot;
  //interfaces for communication
  interface AMSend;
  interface Packet;
  interface SplitControl;
  interface Receive;
  interface PacketAcknowledgements;
  //interface for timer
  interface Timer<TMilli> as MilliTimer;
  interface Read<uint16_t>;
}
```

Third step: In the event Boot.booted we switch on the radio. In SplitControl.startDone we set the periodical timer for mote 1 and we checked the availability of the radio for both mote.

Fourth step: We implemented the SendReq function preparing the message and we used the interface PacketAcknowledgements to set response ACK. At the end the request message to mote 2. This function is called periodically every time the timer expires in the event MilliTimer.fired().

Fifth step: Inside the event AMSend.sendDone we checked for the ACK of the sended message and we counted them to stop the sending of requests as required. We also checked for the ACK of the response message sent by mote 2

Sixth step: Inside the event Receive.receive we read the message arrived. If mote 2 receives a request message we use a sendResp() function to reply.

Seventh step: Inside Read.readDone we implemented the reply messages and we sent them back

Eighth step: We modified the RunSimulationScript.py to set the time at which the second mote boots. We activated the timer debug channel.