Exercise: Setting up a Time Division Multiple Access (TDMA) Mac

1. Goals

Students must implement a perfectly working TDMA MAC that is able to transmit data with a periodic schedule. To this end they have to:

1) change the way packets are scheduled from the FIFO so that when one is available it is not scheduled for transmission according to DCF rules but its availability is notified to the MAC program;

2) add transmission and reception indicators;

3) change the transmission default behavior;

4) intercept the transmission time is approaching and create a sub-loop in the main loop that wait for exact time;

5) start the transmission of the packet when the time comes;

6) adopt a strategy for keeping the transmission times synchronized over the stations belonging to the same BSS;

7) deploy a testbed infrastructure where many nodes join to a common Access Point and they transmit at regular times;

2. Assignment steps

1) **Change the way packets are scheduled from the FIFO** The official code verifies a packet is available in the FIFO by executing handler `check_tx_data_with_disabled_engine`. If conditions for the schedule are verified then the code continues by jumping to `set_ifs` where the packet transmission is finally scheduled. It is hence necessary to avoid jumping to `set_ifs` and remember into a specific variable `PACKET_READY` that a packet is available and that everything has been correctly set up. There is however an exception: if the node receives a packet and the firmware prepares an acknowledgment then it is *mandatory* to clear this variable `PACKET_READY` since `send_response` overrides the settings prepared by `check_tx_data_with_disabled_engine`. With no further changes the system is not able to transmit any packet ☺!

2) **Add transmission and reception indicators** To avoid to schedule in the middle of a reception or transmission we need to keep the `rx/tx` status into a couple of variables (e.g., `rx = 1` when we are receiving, and `tx = 1` when we are transmitting). E.g., for reception it is enough to assign `rx` to 1 at the beginning of `rx_plcp`, and clean it in `rx_complete` (clean means resetting it to zero). For transmission we can set `tx` to 1 in `tx_frame_now` and clean it in `tx_end_wait_10us` and in `rx_plcp`.

3) **Change the transmission default behavior** To avoid unhandled exceptions (given the deep change we are doing) we should change the final part of the code following label

`tx_frame_no_cca_in_progress`. If we take a look to the code we see that it handles some operations and then it may branch to `state_machine_idle` in a couple of ways. We should instead *change* these branches so that they all jump into a loop that wait for condition `COND_TX_DONE` to evaluate true: e.g.,

```
tx_frame_no_cca_in_progress:
            jnzxh   SPR_IFS_STAT & 2048, wait_for_tx_done
            jge     SPR_NAV_0x04, 0x0A0, wait_for_tx_done
[cut]
wait_for_tx_done:
            jext    COND_TX_DONE, tx_really_done
            jext    COND_TRUE, wait_for_tx_done
tx_really_done:
            mov     0, tx
            jext    COND_TRUE, state_machine_idle
```

where the last assignment cleans the variable `tx` that keeps the status of the transmission introduced in the previous step.

4) **Intercept the approaching of the transmission time** To allow an accurate schedule it is necessary to keep the transmission time in a couple of variables (remember that clock has 1us granularity and that a single register of 16 bit allows a maximum schedule of approximately 65ms) as well as the schedule interval, so that we define

   i) `TX_TIME_LO` and `TX_TIME_HI`;
   ii) `TX_INTERVAL_LO` and `TX_INTERVAL_HI`.

   We then have to check if the schedule time is approaching in the main loop, immediately after the `state_machine_start` label: as firmware can put the device to sleep it is also necessary to comment the `nap` instruction right above that label. A good practice is to compare the value of the real time clock `SPR_TSF_WORD0` and `SPR_TSF_WORD1` respectively with the LSW and the MSW of the schedule time `TX_TIME_{LO/HI}`. Try using the signed comparison introduced in the previous labs and reported again at the end of this tutorial (appendix). If we want to intercept the schedule within 20us (that is a good compromise) we can have the following cases:

   a. clock < schedule - 20us: in this case we do not do anything, it's too early! We simply jump out of the section, e.g., jump to `do_not_tx`;

   b. clock > schedule: in this case it means that we missed the schedule so we have to compute a new schedule time in the future `TX_TIME_{LO/HI}` by adding the schedule interval `TX_INTERVAL_{LO/HI}` and exit. Please refer to appendix 2 to check how to sum 32-bit quantities. Also in this case we simply jump out of the section to `do_not_tx`;

   c. no packet is ready for transmission (`PACKET_READY = 0`): in this case we should simply exit to `do_not_tx`;

   d. in the other cases we can try to transmit the packet when the schedule time comes but only if other conditions are verified, that are:

i. no response (ack) was prepared in the recent past (check `COND_NEED_RESPONSEFR`);
ii. no transmission was scheduled before (e.g., an ack by `rx_complete`) by checking

```
jnand  SPR_TXE0_CTL, TXE0_SCHEDULE_WORKING, do_not_tx
```

iii. no reception or transmission is going on, check `rx/tx` variables;
iv. no transmission already started (check `COND_TX_NOW`);

If some of these conditions are not verified we jump to `do_not_tx`, otherwise we execute the code here below.

5) **Start the transmission of the packet** If all the previous conditions are verified then it is possible to start spinning till the clock is finally equal to the schedule time:

```
keep_spinning:
      je     SPR_TSF_WORD0, TX_TIME_LO, tx_immediately
      jext   COND_TRUE, keep_spinning
```

When done, the following code will start an immediate transmission:

```
tx_immediately:
      mov    0, SPR_BRC                              // clean the machine state
      orx    7, 8, 0x000, 0x004, SPR_RXE_FIFOCTL1    // stop reception (if any)
      or     SPR_RXE_FIFOCTL1, 0x000, REG34
      mov    0x4007, SPR_TXE0_CTL                    // try start a tx
      mov    0, PACKET_READY
      mov    1, tx
```

It is better to clear variable `PACKET_READY` and setting to 1 the `tx` variable defined in the previous steps (done by the last two instructions).

At the end of the code pay attention to **reschedule** the next transmission by adding the schedule interval to the schedule time like at step b. above.

6) **Adopt a strategy for synchronizing transmission times of different nodes** Here there are two different phases:

i) the node is not yet associated to the AP, no beacon is being received and parsed so the node cannot be synchronized with others. For this reason and given that the clock starts from zero when the firmware begins working, we can simply initialize the first schedule time to zero. The code will automatically keep rescheduling till schedule time gets greater than the clock.

ii) the node is associated to the AP. In this case when a beacon is received the internal clock is set to the clock information provided by the beacon by handler `rx_beacon_probe_resp`. We can use this to set the first schedule time at N microseconds after the beacon time, where N depends on the node. This can be really useful to establish a transmission order. Make this configurable from the userspace so that the schedule time is periodically

computed from the beacon time by taking into account a value written into shared memory with tool `writeshm`.

7) **Deploy a testbed infrastructure** The testbed will be composed of an AP with a legacy firmware and two stations. The first test to do is to verify that a single node associated to the AP is transmitting at expected times. To this end use the third node to capture the traffic and check that the transmission time follows a periodic evolution. Try to evaluate the precision of the scheduling but remember to associate also the sniffer to the same AP so that it will keep refreshing its internal clock with that of the AP.

   Add then other nodes and verify they still transmit where expected and they do not collide. Good practice for these tests is to set the MCS of nodes involved in the experiment to a fixed and common value. Check now that `iperf` sessions from different nodes to the AP fairly share the available bandwidth. Check how many packets are lost. Try to compute a schedule interval for having the highest throughput, then compare the throughputs with that you can obtain using a standard DCF

   i) Is the TDMA more fair (with respect to fast time changes) than DCF?

APPENDIX Comparison between two 32-bit quantities As 32-bit values musb be split into couples of 16-bit registers, the comparison between two 32-bit quantities is tricky: it involves, in fact, two subtractions and a test over the carry register. Use the following snippet of code at your convenience.

```
// compare VAL1 and VAL2; use r63 as convenience register;
// store difference VAL1 - VAL2 in RES
// 32-bit value in VAL1 is split in VAL1LSW and VAL1MSW
// 32-bit value in VAL2 is split in VAL2LSW and VAL2MSW
         mov     VAL2LSW, r63
         sub.    VAL1LSW, r63, RESLSW
         mov     VAL2MSW, r63
         subc.   VAL1MSW, r63, RESMSW
         addc    0, 0, r63
// output:       r63 = 1, if VAL1 >= VAL2
//               r63 = 0, if VAL1 < VAL2
//               RES = VAL1 - VAL2
```

After the code snippet one can test the value in r63 to jump in one of the two cases.

APPENDIX2 Sum of 32-bit quantities As 32-bit values must be split into couples of 16-bit registers, summing two 32-bit quantities requires to sum the 16-bit parts and use the carry register. Use the following snippet of code at your convenience.

```
// sum VAL1 and VAL2 and put result in VAL3
         add.    VAL1LSW, VAL2LSW, VAL3LSW
         addc    VAL1MSW, VAL2MSW, VAL3MSW
// output:        VAL3 = VAL1 + VAL2
```

The first `add.` instruction (ending with the dot) sum the two 16-bit quantities and set the carry bit accordingly. The second `addc` instruction sum the two 16-bit quantities and the value set in the carry.