COM Module Automation: Periodicity Check of Radar Sensor Signals

Objective: Verify the periodicity of two signals in a radar sensor message with defined time intervals and tolerances using CAPL automation for a period of 5 seconds.

Requirement Description:

1. **Signal 1 – 0x213 - 500ms Periodicity Check:**
    - The system shall monitor **0x213** and verify that it is received every **500ms ± 5 ms** (measure for 5 seconds duration). Check Average, Minimum and Maximum lies in between tolerance.
2. **Signal 2 - 0x214 - 200ms Periodicity Check:**
    - The system shall monitor **0x214** and verify that it is received every **200ms ± 5 ms** (measure for 5 seconds duration). Check Average, Minimum and Maximum lies in between tolerance.

Test Cases:

Test Case ID	Action	Expected Result
TC_01	Verify periodicity of 0x213 with 500ms	Signal 1 is received every 500ms ± 5ms
TC_02	Verify periodicity of 0x214 with 500ms	Signal 1 is received every 500ms ± 5ms

CAPL code for Radar ECU

```
 includes
{
}
variables
{
 // Define radar messages
 message 0x213 a;  // Signal 1 - 500ms cycle time
 message 0x214 b;  // Signal 2 - 200ms cycle time
 // Define timers for message transmission
 msTimer t1, t2;
}
```

```
on start
{
  setTimer(t1, 500);  // Set timer for Signal 1 (500ms)
  setTimer(t2, 200);  // Set timer for Signal 2 (200ms)
}
on timer t1
{
  output(a);
 setTimer(t1, 500);
}

on timer t2
{
  // Output Signal 2 and reset the timer
  output(b);
  setTimer(t2, 200);
}
```

CAPL Automation for Periodicity check

```
includes
{

}

variables
{
  dword gcycCheckId;

  // Variables of Radar message 0x213 = 500ms
  const long min_cyc_time_0x213 = 495;
  const long max_cyc_time_0x213 = 505;

  // Variables of Radar message 0x214 = 200ms
  const long min_cyc_time_0x214 = 195;
  const long max_cyc_time_0x214 = 205;

  const long kTIMEOUT = 5000;
}

void Maintest()
{
  testReportFileName("Radar_Tx_Testing");
  testModuleTitle("Radar messages and its periodicity check");
  testModuleDescription("Check cycle time for radar transmitted messages");
```

```
 Radar_213_TC1();
 Radar_214_TC2();
}

testcase Radar_213_TC1()
{
 float cyc_time_min;
 float cyc_time_max;

 cyc_time_min = min_cyc_time_0x213;
 cyc_time_max = max_cyc_time_0x213;

 testCaseTitle("TC1", "0x213 : cycle_time");

 gcycCheckId = ChkStart_MsgAbsCycleTimeViolation(0x213, cyc_time_min, cyc_time_max);

 CheckMsg(cyc_time_min, cyc_time_max);
}

CheckMsg(float acyc_time_min, float acyc_time_max)
{
 long Avg;
 long Min;
 long Max;
 char buffer[100];

 testWaitForTimeout(kTIMEOUT);

 Avg = ChkQuery_StatProbeIntervalAvg(gcycCheckId);
 Min = ChkQuery_StatProbeIntervalMin(gcycCheckId);
 Max = ChkQuery_StatProbeIntervalMax(gcycCheckId);

 if (ChkQuery_NumEvents(gcycCheckId) > 0)
 {
  testStepFail("TC1", "Message has not triggered at expected time");
  snprintf(buffer, elcount(buffer), "Valid values %f.0ms - %f.0ms", acyc_time_min, acyc_time_max);
  testStepFail("", buffer);
  snprintf(buffer, elcount(buffer), "Average cycle time: %dms", Avg);
  testStepFail("", buffer);
  snprintf(buffer, elcount(buffer), "Minimum cycle time: %dms", Min);
  testStepFail("", buffer);
  snprintf(buffer, elcount(buffer), "Maximum cycle time: %dms", Max);
  testStepFail("", buffer);
 }
 else
 {
  testStepPass("TC1", "Message has triggered at expected time");
  snprintf(buffer, elcount(buffer), "Valid values %f.0ms - %f.0ms", acyc_time_min, acyc_time_max);
```

```
   testStepPass("", buffer);
   snprintf(buffer, elcount(buffer), "Average cycle time: %dms", Avg);
   testStepPass("", buffer);
   snprintf(buffer, elcount(buffer), "Minimum cycle time: %dms", Min);
   testStepPass("", buffer);
   snprintf(buffer, elcount(buffer), "Maximum cycle time: %dms", Max);
   testStepPass("", buffer);
  }
}

testcase Radar_214_TC2()
{
  float cyc_time_min;
  float cyc_time_max;

  cyc_time_min = min_cyc_time_0x214;
  cyc_time_max = max_cyc_time_0x214;

  testCaseTitle("TC2", "0x214 - cycle time");

  gcycCheckId = ChkStart_MsgAbsCycleTimeViolation(0x214, cyc_time_min, cyc_time_max);

  CheckMsg2(cyc_time_min, cyc_time_max);
}

CheckMsg2(float bcyc_time_min, float bcyc_time_max)
{
  long Avg;
  long Min;
  long Max;
  char buffer[100];

  testWaitForTimeout(kTIMEOUT);

  Avg = ChkQuery_StatProbeIntervalAvg(gcycCheckId);
  Min = ChkQuery_StatProbeIntervalMin(gcycCheckId);
  Max = ChkQuery_StatProbeIntervalMax(gcycCheckId);

  if (ChkQuery_NumEvents(gcycCheckId) > 0)
  {
   testStepFail("TC2", "Message has not triggered at expected time");
   snprintf(buffer, elcount(buffer), "Valid values %f.0ms - %f.0ms", bcyc_time_min, bcyc_time_max);
   testStepFail("", buffer);
   snprintf(buffer, elcount(buffer), "Average cycle time: %dms", Avg);
   testStepFail("", buffer);
   snprintf(buffer, elcount(buffer), "Minimum cycle time: %dms", Min);
   testStepFail("", buffer);
   snprintf(buffer, elcount(buffer), "Maximum cycle time: %dms", Max);
```

```
    testStepFail("", buffer);
  }
  else
  {
    testStepPass("TC2", "Message has triggered at expected time");
    snprintf(buffer, elcount(buffer), "Valid values %f.0ms - %f.0ms", bcyc_time_min, bcyc_time_max);
    testStepPass("", buffer);
    snprintf(buffer, elcount(buffer), "Average cycle time: %dms", Avg);
    testStepPass("", buffer);
    snprintf(buffer, elcount(buffer), "Minimum cycle time: %dms", Min);
    testStepPass("", buffer);
    snprintf(buffer, elcount(buffer), "Maximum cycle time: %dms", Max);
    testStepPass("", buffer);
  }
}
```