Ada project

— Epita —

Maxime Mehio Paul Beneteau

31 janvier 2020

Table des matières

1 Overview

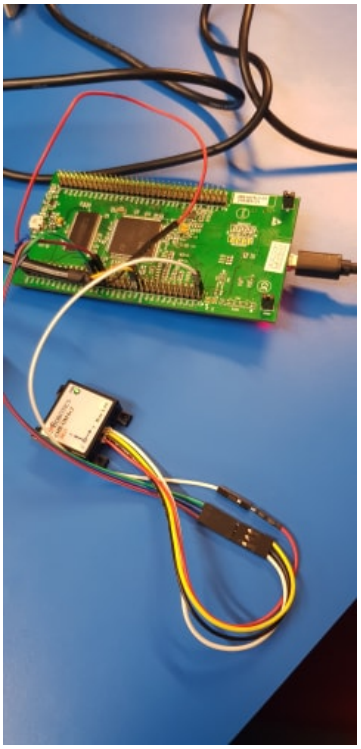

FIGURE 1 – branchement

2 DO-178C

2.1 System requirements

2.1.1 High-level requirements

- Regular data flow between the different devices
  - · configure the UM6 on broadcast mode, this mode makes it send data with a configurable frequency, show only certain types of data (since there are multiple sensors on the UM6, such as thermometer and gyroscope). This way, we don't have to worry about querying the UM6 on a regular basis.
  - · The code on the STM32F429 board needs to parse the packages from the UM6 and forward the important informations directly to the receiving Computer
- Smooth mouse movements, we would like to have a user good looking result and especially a mouse that wouldn't teleport from one side of the screen to an over ;
  - · The library we used for python to move the mouse didn't contain a smooth movement function. The strategy was then to slice a big movement into several smaller ones, all seperated using very short calls to time.sleep().
  - · Removal of some low perturbations using proccessedd data instead of raw data (configuration of UM6)

- The delay between the sensor and the mouse mouvement stay low
  - The accelerometer data received from the imu resfreshes at a constant and low rate
  - The board only does a very few operations before propagating data to the computer.
- Connect the sensors to the computer
  - Using a board in the middle that transfers the data through multiple serial connections
  - Find a way for the computer's mouse to act according to the data received : First strategy is enabling the Serial USB cable to be considered as a standard mouse. Second method is to have a program that reads data from Serial and have the mouse to move accordingly (really simple in python)
- Being able to use a mouse without any surface to use it on.
  - Use only data from two axis of acceleration on the sensor.
- Having stable connections between the different devices
  - This requires to have some error handling for the serial protocol.

2.2 Software Test Cases for Requirements

- Regular dataflow requirements :
  - UM6 Configuration for broadcastmode : Check that the command succeeded in the response from UM6 communication register with the packet type.
  - Parsing Packet and sending information to computer : Check that the package parsed is good with the checksum
- smooth mouse mouvement requirements :
  - Removal of data with UM6 configuration : Check command success, same as regular dataflow requirements
- Mouse delay requirements :
  - Check the frequency at which data are received.
- Being able to use a mouse without any surface to use it on.
  - Use only data from two axis of acceleration on the sensor.
- Having stable connections between the different devices
  - This requires to have some error handling for the serial protocol.

2.3 Traceability

HLR	LLR
Req1 : Regular dataflow	Req1.1 : Check configuration command success
	Req1.2 Check parsing with package checksum
Req2 : Smooth mouse movement	Req2.1 : Check configuration command success
Req3 : Mouse delay	Req3.1 : Check data received frequency
Req4 : Stable connection	Req3.1 : Error handling

3 CONBAP

Find_package : Precondition that check Message size.

Init_Get_Cmd and Init_Conf_Cmd : Precondition that check Cmdstring size. Postcondition that check the checksum is correct. Loop_invariant on the checksum value

```
 -- Find the start of the package received In Incoming by finding
 -- the three character start sequence 's', 'n', 'p' of the package
 -- return the index where the package start or 0 if no package was fond
 function Find_Package (Incoming : Message) return Integer
   -- Minimum Length of package is 7 bytes
   with Pre => Length(Incoming) >= 7
 is
 begin
    for I in 1 .. 5 loop
       pragma Loop_Invariant (for all J in 1 .. 5 => Content(Incoming)(J) /= 's'or Content(Incoming)(J + 1) /= 'n'
                              or Content(Incoming)(J + 2) /= 'p');
       if (Content(Incoming)(I) = 's'and Content(Incoming)(I + 1) = 'n' and Content(Incoming)(I + 2) = 'p') then
          return I;
       end if;
    end loop;
    return 0;
 end Find_Package;
```

FIGURE 2 – find_package

```
-- Create the Command to get data from the accelerometer data register
function Init_Get_Cmd (cmdString : in out String) return String
  with Pre => cmdString'Length >= 11,
       Post => checksum = Character'Pos(cmdString(cmdString'First + 5)) * 256 + Character'Pos(cmdString(cmdString'First + 6))
is
  Index : constant Integer := cmdString'First;
begin
  checksum := 0;
  cmdString(Index) := 's';  -- beginning of every packets
  cmdString(Index + 1) := 'n';
  cmdString(Index + 2) := 'p';
  cmdString(Index + 3) := Character'val(0);   -- configuration -> Package type = 0
  cmdString(Index + 4) := Character'val(94);  -- addr data register 0x5E
  for I in Index .. Index + 4 loop
      checksum := checksum + Character'Pos(cmdString(I));
      pragma Loop_Invariant (checksum <= checksum'Loop_Entry + 255);
  end loop;
  cmdString(Index + 5) := Character'Val(checksum / 256);
  cmdString(Index + 6) := Character'Val(checksum mod 256);
  return cmdString;
end Init_Get_Cmd;
```

FIGURE 3 – Init_Get_Cmd

```
 -- Create the Command to configure the UM6 sensor
 function Init_Conf_Cmd (cmdString : in out String) return String
   with Pre => cmdString'Length >= 11,
        Post => checksum = Character'Pos(cmdString(cmdString'First + 9)) * 256
                         + Character'Pos(cmdString(cmdString'First + 10))
 is
    Index : constant Integer := cmdString'First;
 begin
    checksum := 0;
    cmdString(Index) := 's';  -- beginning of every packets
    cmdString(Index + 1) := 'n';
    cmdString(Index + 2) := 'p';
    cmdString(Index + 3) := Character'val(0);  -- configuration -> Package type = 0
    cmdString(Index + 4) := Character'val(0);  -- 0x00 register (communication settings)
    cmdString(Index + 5) := Character'val(66); -- 0100 0010  for Broadcast mode and Processed accelerometer enabled
    cmdString(Index + 6) := Character'val(0);   -- useless options for us
    cmdString(Index + 7) := Character'val(56); -- 0011 1000  last 3 zeros to set baud rate to 9600
    cmdString(Index + 8) := Character'val(64); -- broadcast rate (refresh mesures rate)
    for I in Index .. Index + 8 loop
       checksum := checksum + Character'Pos(cmdString(I));
       pragma Loop_Invariant (checksum <= checksum'Loop_Entry + 255);
    end loop;
    cmdString(Index + 9) := Character'Val(checksum / 256);
    cmdString(Index + 10) := Character'Val(checksum mod 256);
    return cmdString;
 end Init_Conf_Cmd;
```

FIGURE 4 – Init_Conf_Cmd