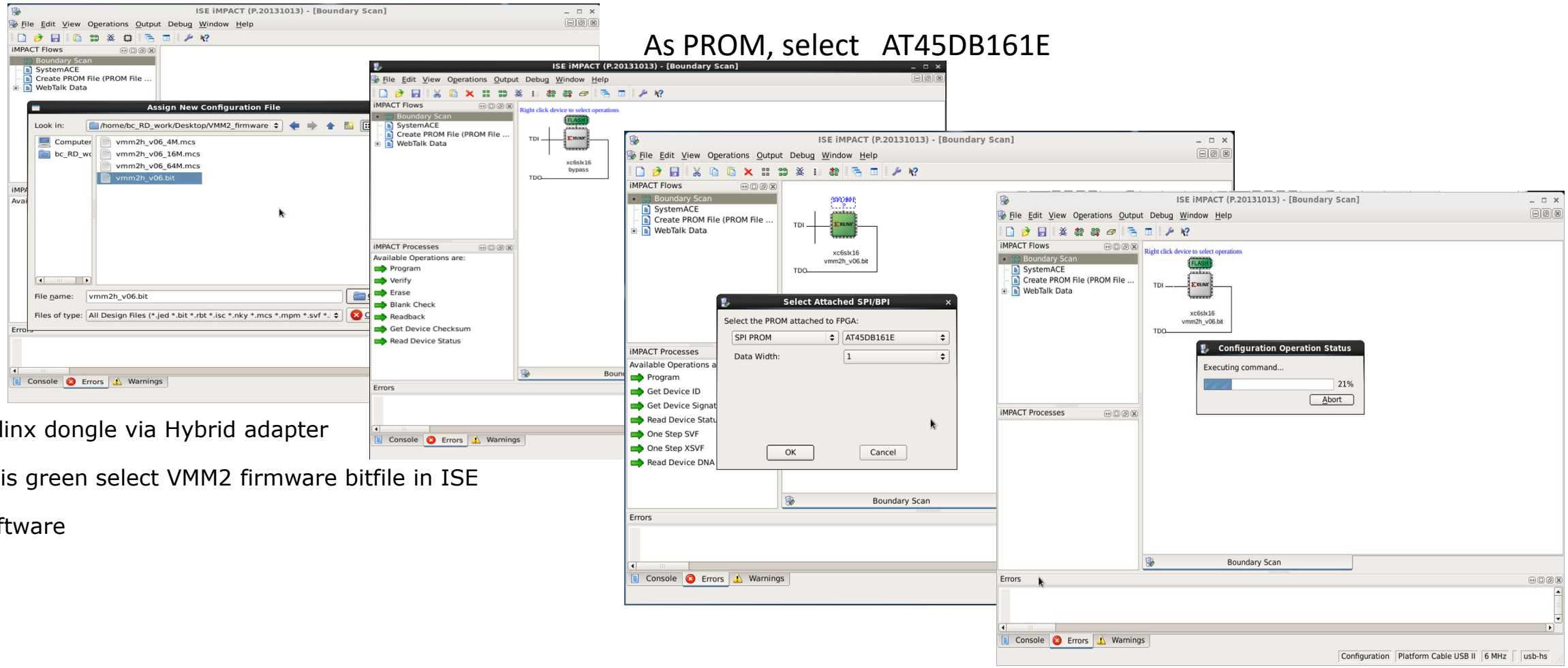
VMM hybrid test Software for uPython-driven test card

VMM hybrids from factory
COOLERS NOT YET MOUNTED

1st step:
VMM Flash programming

New hybrids: to be done before cooler assembly !
Only connect P1 AUX power, do not connect P2 to avoid VMM overheating.

Spartan FPGA programming via ISE



As PROM, select AT45DB161E

Connect Xilinx dongle via Hybrid adapter

and if LED is green select VMM2 firmware bitfile in ISE

IMPACT software

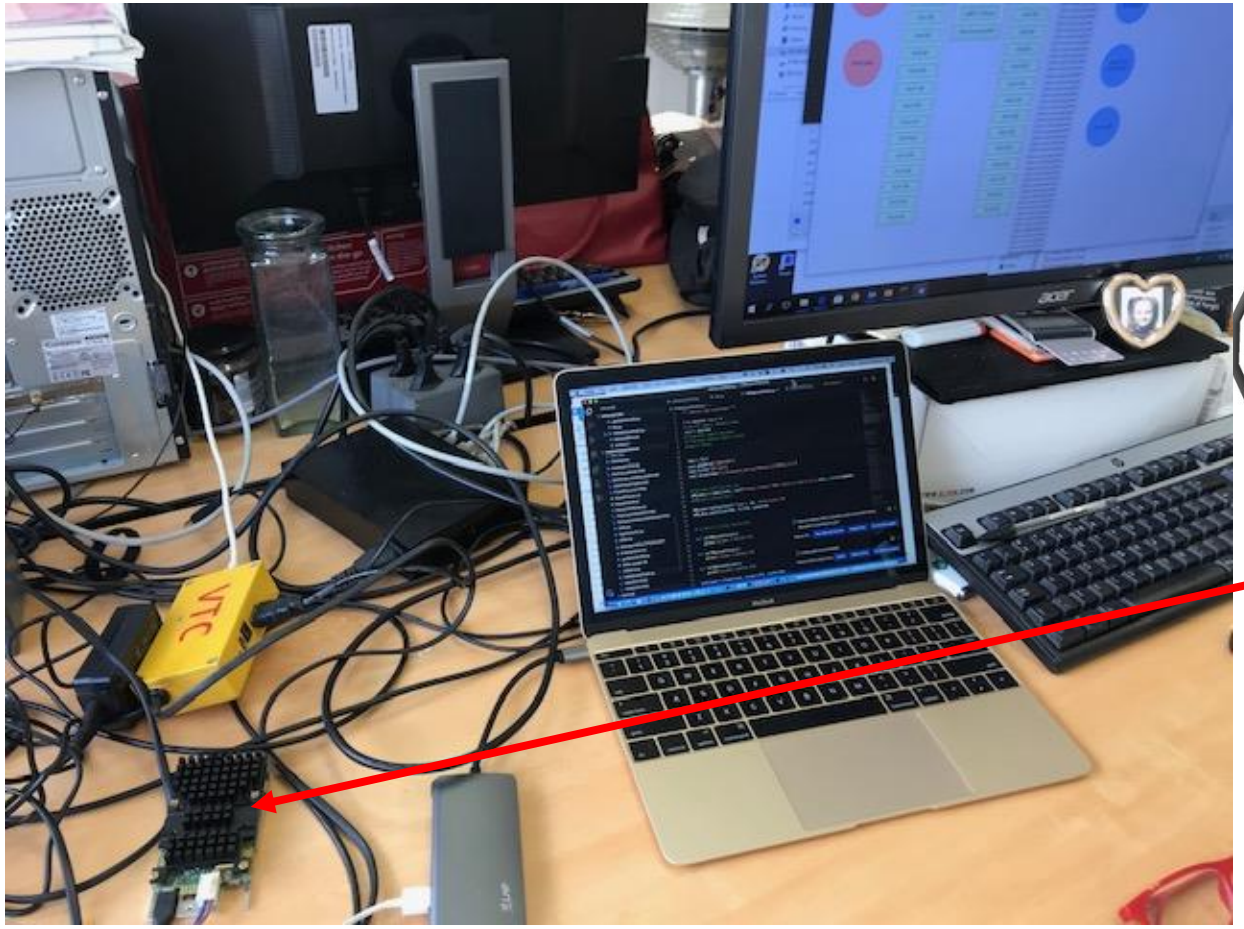
Wait until programming is completed

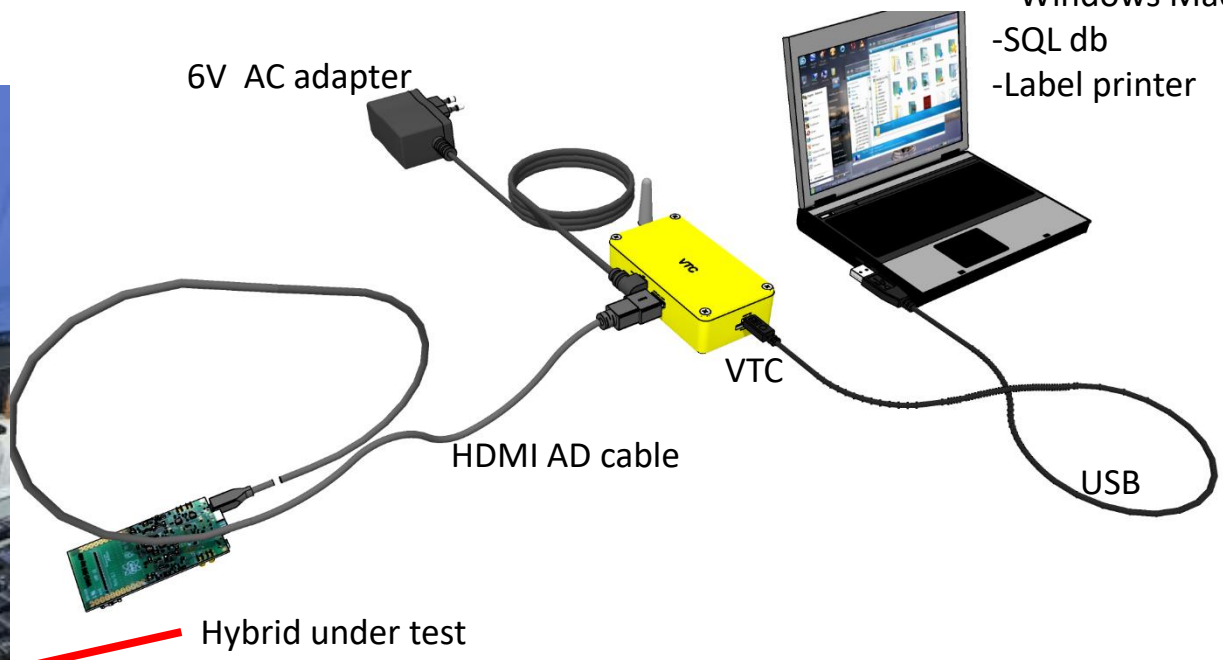
Ca. 2 minutes

2nd step
VMM hybrids , COOLERS mounted

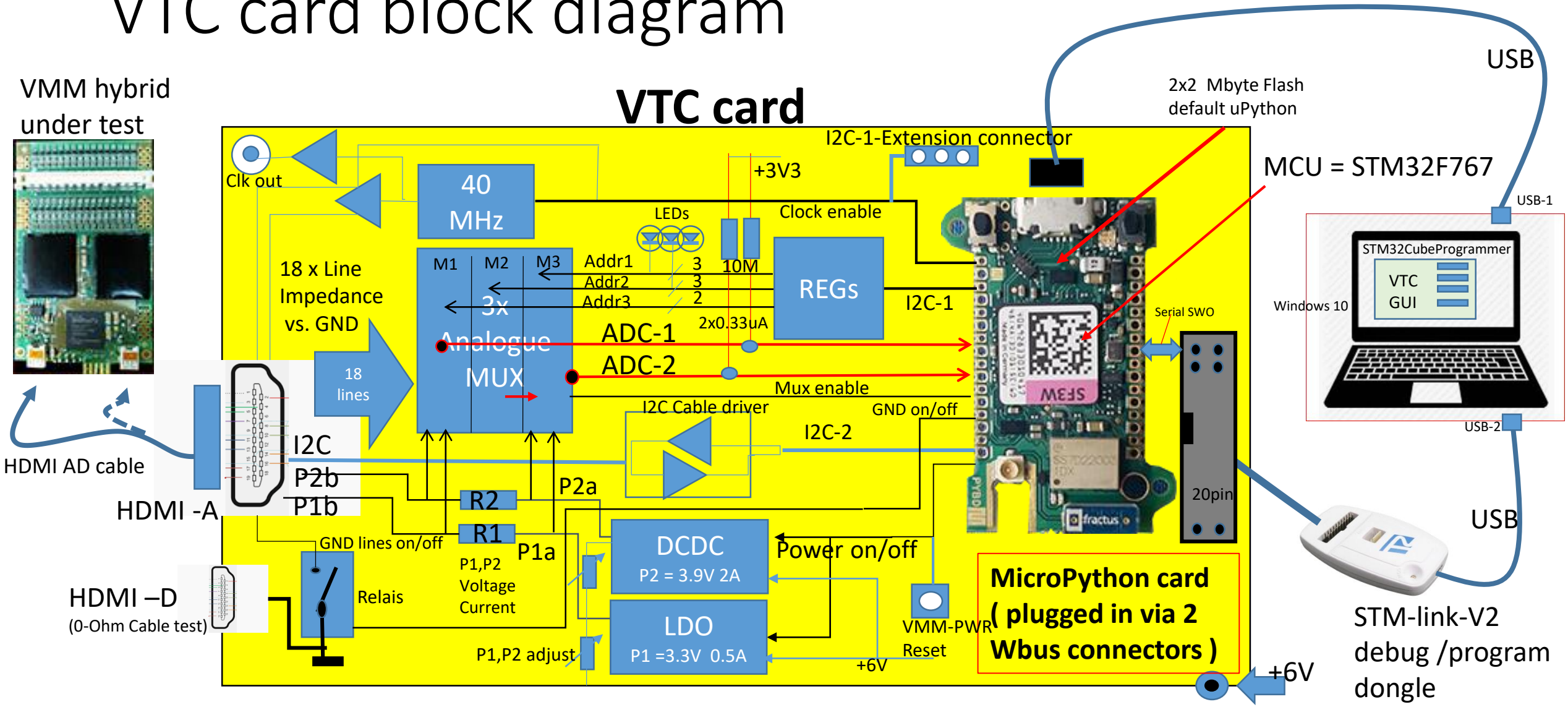
3rd test via VCT box

VMM set up test via VTC





VTC card block diagram

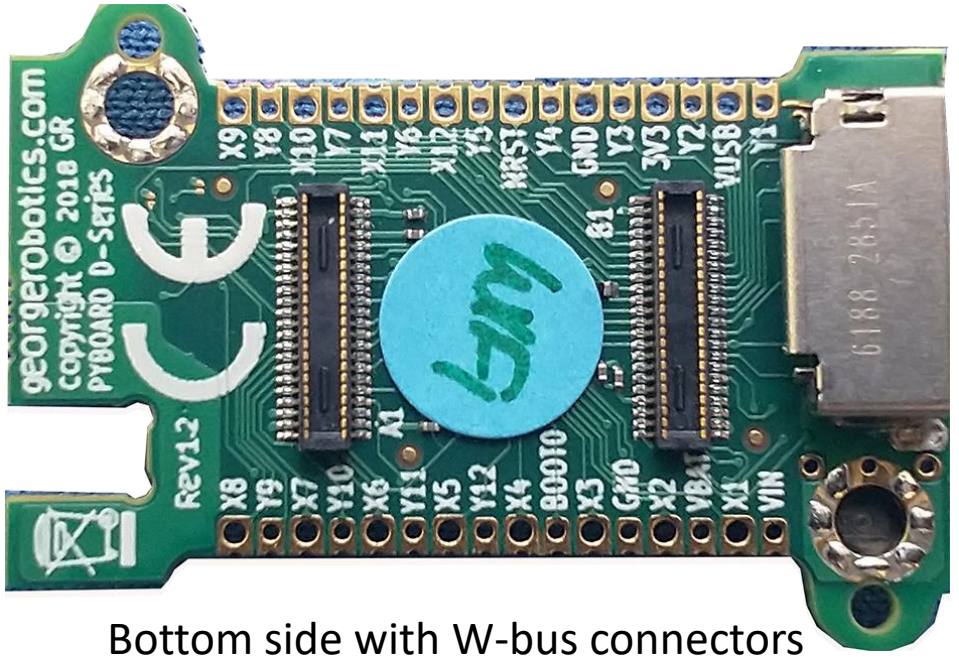


uPython card

(https://store.micropython.org/product/PYBD-SF6-W4F2)

The chosen MCU card is a IoT device with programmable I/Os and wireless connectivity.
The integrated MicroPython language does nor require compilation and comes with Libraries which
for example allow to scan / read I2C devices. Powered via USB, with bottom side WBUS extension connectors.
Very good documentation on micropython.org 32 it MCU, manuals, debugging tools High level libraries etc
from ST Microelectronics ST.com

Top side with USB and Wifi antenna



Bottom side with W-bus connectors

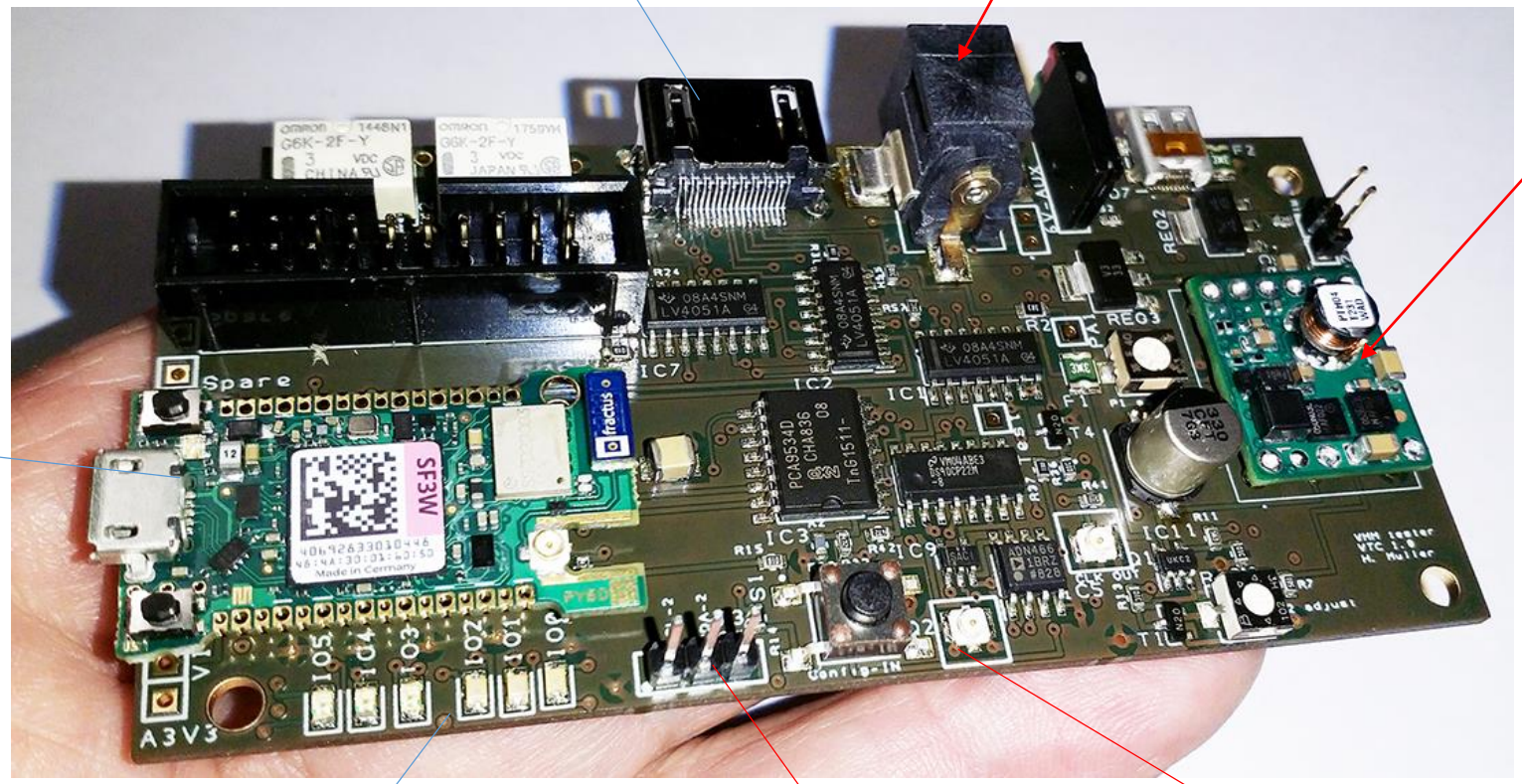
Photo of VTC card 1.0 with PYBD

HDMI cable port -> VMM

Input power jack (6V)

DCDC power converter

Plugged Python Card with USB, Wifi and SD card



Status /address LEDs

I2C bus plug

40 MHz clock coax

VMM Current and Voltage measurement

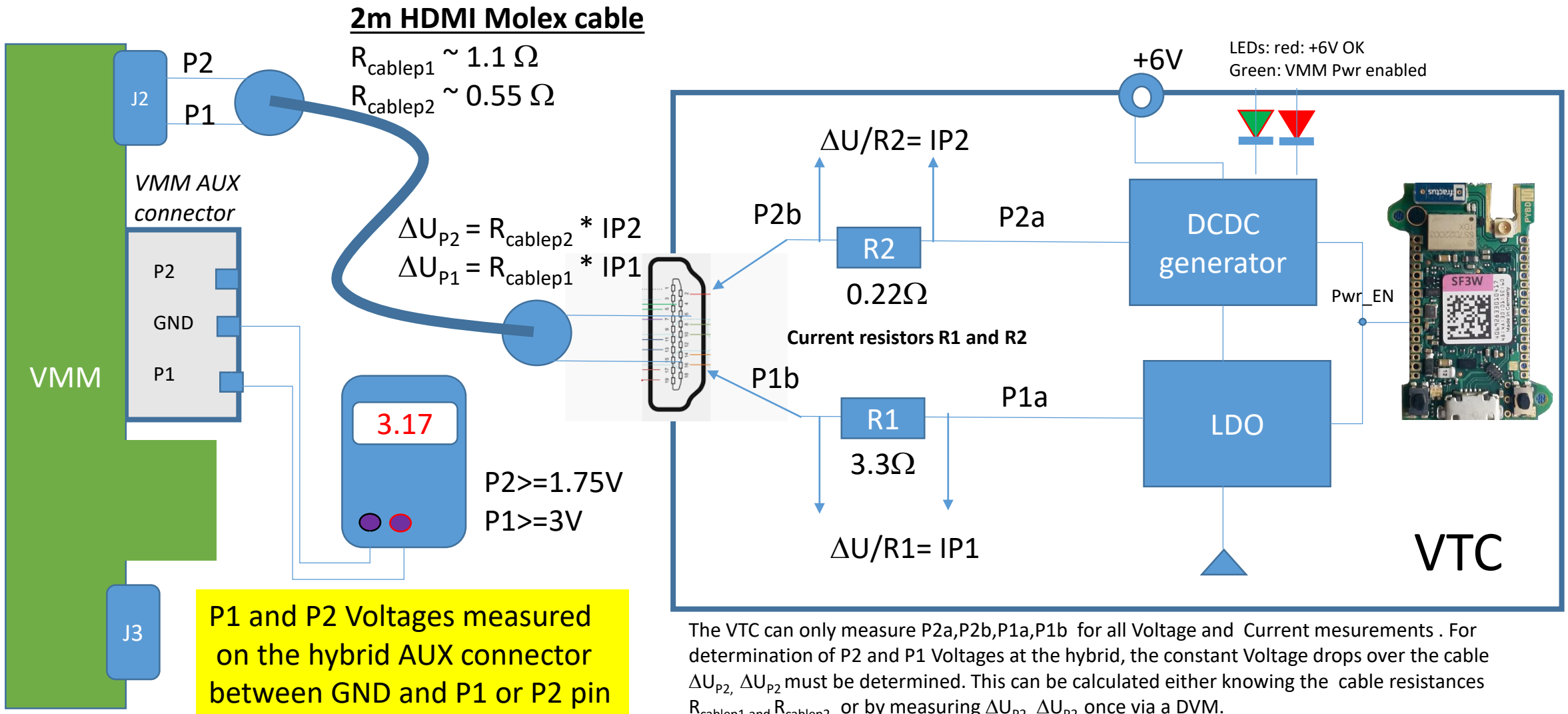
P2 is the main VMM power driving the 2 VMM3a ASICS. The nominal current is 1.6 A and the nominal Voltage at the hybrid AUX connector vs. the center GND pin must be >= 1.75V.

P1 is the power for the Flash and I2C chips. The nominal current is 0.15A and the Voltage at the AUX connector vs. center GND pin must be >= 3.0V

Both P1 and P2 Voltages are provided by the VTC box and are enabled/disabled via software. There is a pushbutton on the VTC card to momentarily disable both P1 and P1.

A power cycle can also be generated via a uPython program

Principle Voltage / Current measurement



P1 and P2 Voltages measured on the hybrid AUX connector between GND and P1 or P2 pin

The VTC can only measure P2a,P2b,P1a,P1b for all Voltage and Current mesurements . For determination of P2 and P1 Voltages at the hybrid, the constant Voltage drops over the cable $\Delta U_{P2,}$ ΔU_{P2} must be determined. This can be calculated either knowing the cable resistances $R_{cablep1 \text{ and }} R_{cablep2}$ or by measuring $\Delta U_{P2,}$ ΔU_{P2} once via a DVM.

Test principle in a nutshell

Connect VMM hybrid to HDMI cable to VTC

Level 1

- Use 2 ADC's of python card to measure all 19 HDMI lines ohmic
- Compare ohmic test results with expected values
- -> 1st test OK or failed

Level 2

- Enable 40MHz and 2 power lines to VMM , generate power cycle
- Measure voltages and currents P1 and P2
- -> 2nd test OK or failed

Level 3 (more advanced)

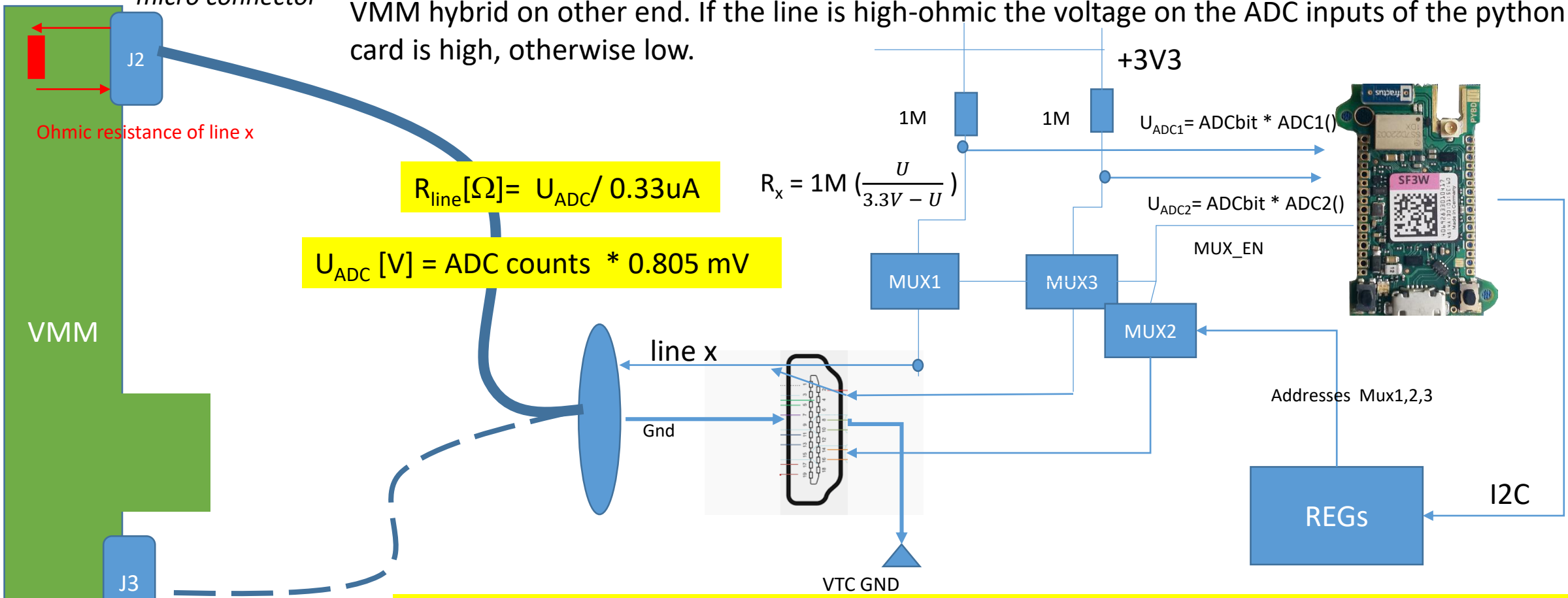
- Perform I2C device scan (via HDMI though Spartan FPGA on hybrid)
- List all devices found
- If ID chip found, read unique identifier

-> 3rd test OK or failed

All basic test procedures
are written in uPython
and execute on Python plugin card

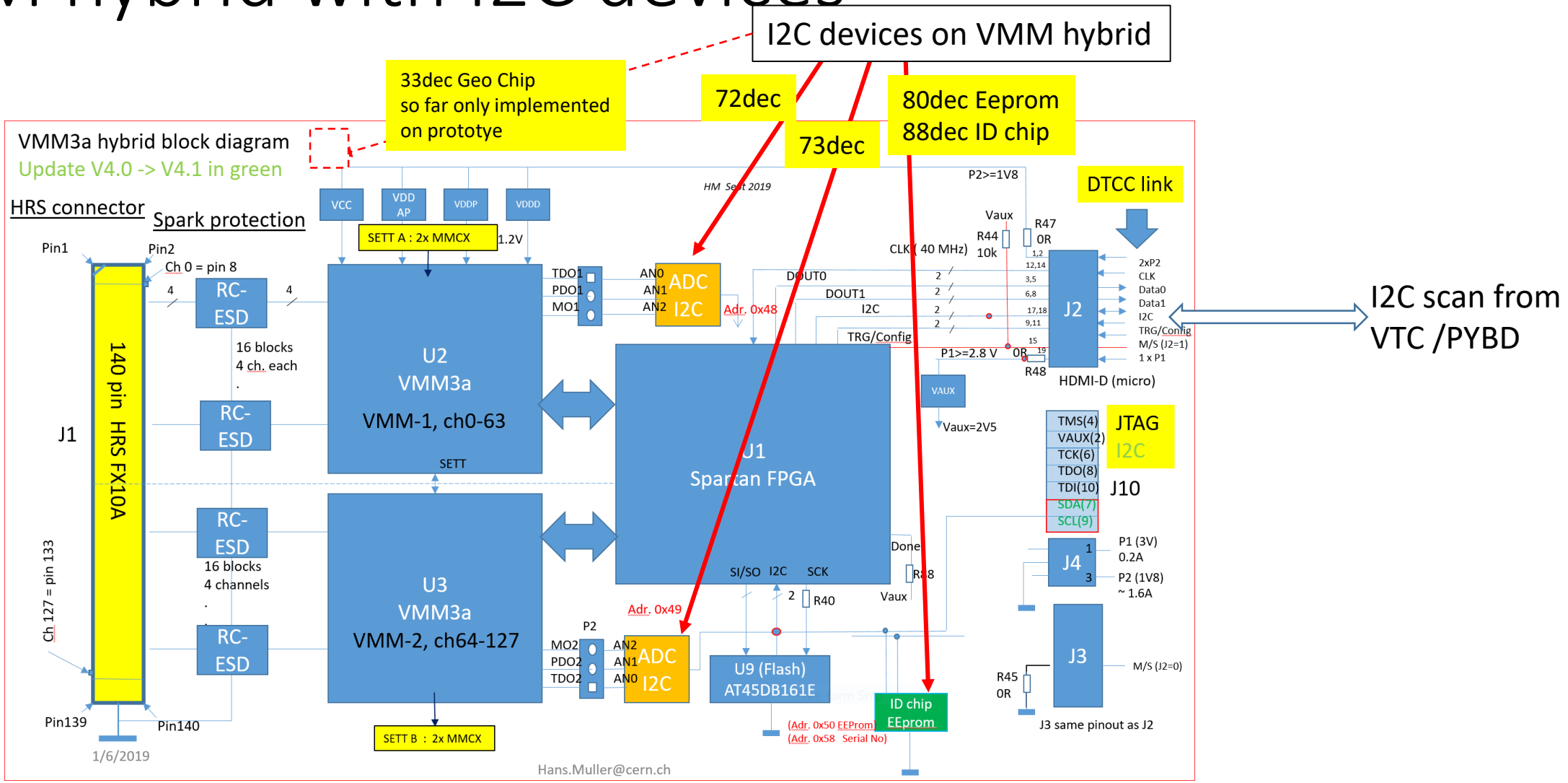
Principle Ohmic line measurements

10M resistors are used to gerenerate a quasi current source of 0.33uA between 3V and GND. Via the analogue multiplexers MUX1,2,3 the current is injected into the HDMI cable with the VMM hybrid on other end. If the line is high-ohmic the voltage on the ADC inputs of the python card is high, otherwise low.



Note: the purpose of the measurement is not to determine the exact ohmic value
It is rather an order of magnitude determination for 3 ranges: less 5Ω, O(1-5k), greater 1M

VMM hybrid with I2C devices



Example of a uPython screenshot:
Serial Number readout from ID chip

```
>>>
paste mode; Ctrl-C to cancel, Ctrl-D to finish
=== from machine import Pin
=== PWR_EN = Pin('X3', Pin.OUT)
=== AMUX_EN = Pin('X1', Pin.OUT)
=== GND_EN = Pin('X6', Pin.OUT)
=== CLK_EN = Pin('X4', Pin.OUT)
=== CLK_EN.value(1)
=== GND_EN.value(1)
=== AMUX_EN.value(1)
=== PWR_EN.value(1)
=== from machine import I2C
=== i2c = machine.I2C('X')
=== i2c = I2C(scl='Y9', sda='Y10',freq=100000)
=== Pin('PULL_SDA', Pin.OUT, value=1)
=== Pin('PULL_SCL', Pin.OUT, value=1)
=== for i in range(1):
===     i2c.scan()              # scan the internal I2C bus on the VTC V1.0
===
=== ADS1015_VMM1=_[0]
=== ADS1015_VMM2=_[1]
=== AT24CS02_EE=_[2]
=== AT24CS02_ID=_[3]
===
=== i2c.writeto(88,b'\x80')             # # write special word 0x80 to ID chip address
=== U=i2c.readfrom(88,16)               # read 16 bytes from ID chip
=== print('ID =',hex(U[0]),hex(U[1]),hex(U[2]),hex(U[3]),hex(U[4]),hex(U[5]),hex(U[6]),hex(U[7]),hex(U[8]),hex(U[9]),hex(U[10]),hex(U[11]),hex(U[12]),hex(U[
ex(U[14]),hex(U[15]))
===
Pin(Pin.cpu.H5, mode=Pin.OUT)
Pin(Pin.cpu.F1, mode=Pin.OUT)
[72, 73, 80, 88]
1
ID = 0x1e 0x80 0x7 0x18 0x64 0x10 0x0 0x61 0xd8 0xb6 0xa0 0x0 0xa0 0x0 0x0 0xb4
>>>
```

This is unique 128 bit number of the ID chip

uPython screenshot: (1)
connected HDMI line impedances MUX1/MUX3

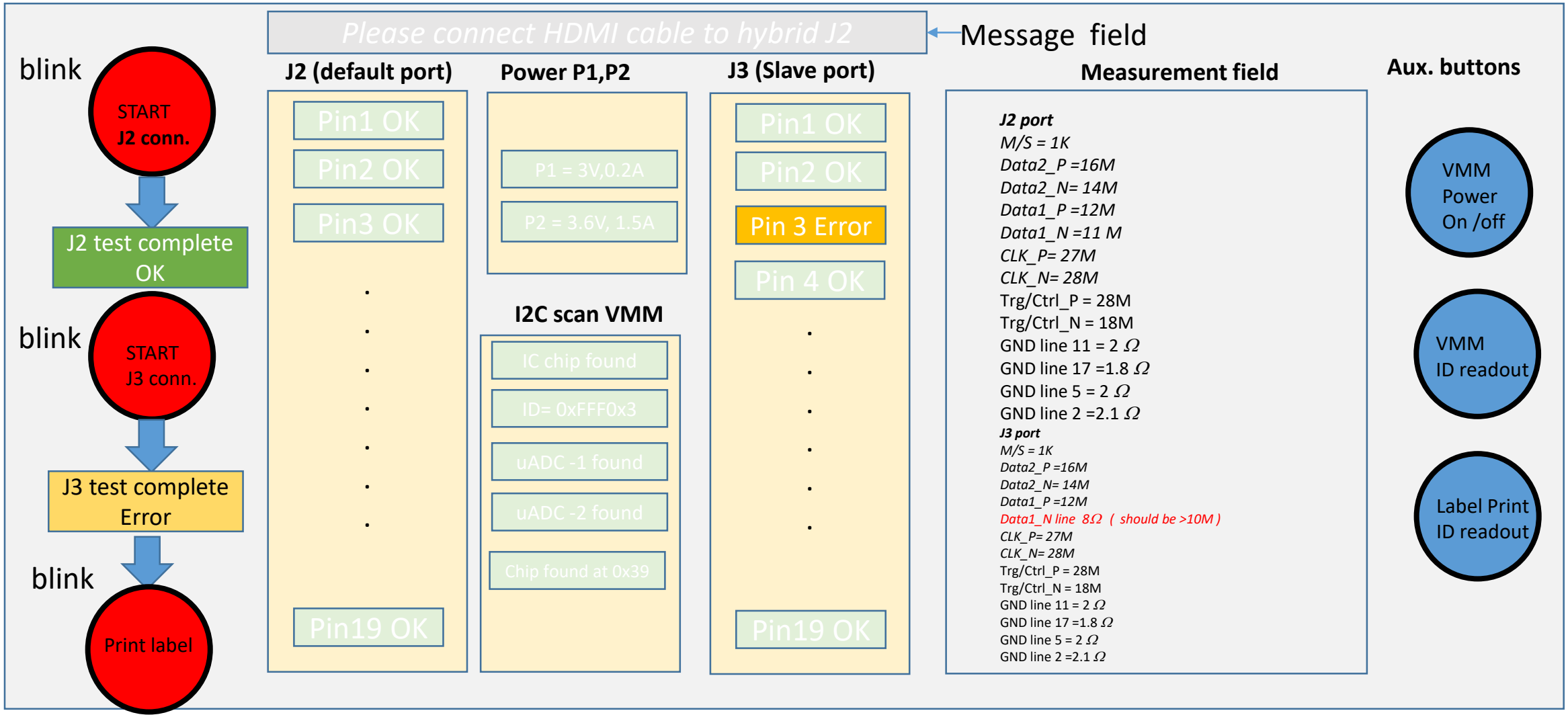
```
# MUX1 line test for connected VMM3a hybrid with VTC 1.0 hardware
# -power, clock and GND lines disabled
# -measure ohmic impedance of lines data1, M/S
```

```
=== RX = 1000*UP1B/(3.3-UP1B)
=== print('P1B=',round(RX,2),'kOHM')
=== i2c.writeto_mem(addr, 1, b'\x05')
=== for i in range(10000):
===     pass
=== Utest=adc1.read()*ADCbit
=== print('U=',Utest,'V')
=== RX = 1000000*Utest/(3.3-Utest)
=== print('test-input=',round(RX,2),'OHM')
=== i2c.writeto_mem(addr, 1, b'\x07')
=== for i in range(10000):
===     pass
=== UGNDA=adc1.read()*ADCbit
=== print('U=',UGNDA,'V')
=== RX = 1000*UGNDA/(3.3-UGNDA)
=== print('GND-A=',round(RX,2),'kOHM')
=== i2c.writeto_mem(addr, 1, b'\x47')
=== for i in range(10000):
===     pass
=== UGNDB=adc1.read()*ADCbit
=== print('U=',UGNDB,'V')
=== RX = 1000*UGNDB/(3.3-UGNDB)
=== print('GND-B=',round(RX,2),'kOHM')
=== i2c.writeto_mem(addr, 1, b'\x87')
=== for i in range(10000):
===     pass
=== UGNDC=adc1.read()*ADCbit
=== print('U=',UGNDC,'V')
=== RX = 1000*UGNDC/(3.3-UGNDC)
=== print('GND-C=',round(RX,2),'kOHM')
=== i2c.writeto_mem(addr, 1, b'\xC7')
=== for i in range(10000):
===     pass
=== UGNDD=adc1.read()*ADCbit
=== print('U=',UGNDD,'V')
=== RX = 1000*UGNDD/(3.3-UGNDD)
=== print('GND-D=',round(RX,2),'OHM')
===
Pin(Pin.cpu.H5, mode=Pin.OUT)
Pin(Pin.cpu.F1, mode=Pin.OUT)
[32]
U= 3.14287966 V
data1_p= 20.0 MOHM
U= 3.17591172 V
data1_n= 20.21 MOHM
U= 0.07976033999999999 V
M/S= 24.77 kOHM
U= 0.42055452 V
P2B= 146.05 kOHM
U= 0.02094716 V
P1B= 6.39 kOHM
U= 0.6243865 V
test-input= 233361.99 OHM
U= 0.00241698 V
GND-A= 0.73 kOHM
U= 0.00161132 V
GND-B= 0.49 kOHM
U= 0.00161132 V
GND-C= 0.49 kOHM
U= 0.00161132 V
GND-D= 0.49 OHM
>>>
```

```
=== for i in range(10000):
===     pass
=== Udata2_p=adc2.read()*ADCbit
=== print('U=',Udata2_p,'V')
=== RX = Udata2_p/(3.3-Udata2_p)
=== print('Data2_P=',round(RX,2),'MOHM')
=== i2c.writeto_mem(addr, 1, b'\x18')
=== for i in range(10000):
===     pass
=== Udata2_n=adc2.read()*ADCbit
=== print('U=',Udata2_n,'V')
=== RX = Udata2_n/(3.3-Udata2_n)
=== print('Data2_N=',round(RX,2),'MOHM')
=== i2c.writeto_mem(addr, 1, b'\x20')
=== for i in range(10000):
===     pass
=== UP2A=adc2.read()*ADCbit
=== print('U=',UP2A,'V')
=== RX = 1000*UP2A/(3.3-UP2A)
=== print('P2A=',round(RX,2),'kOHM')
=== i2c.writeto_mem(addr, 1, b'\x28')
=== for i in range(10000):
===     pass
=== UP1A=adc2.read()*ADCbit
=== print('U=',UP1A,'V')
=== RX = 1000*UP1A/(3.3-UP1A)
=== print('P1A=',round(RX,2),'kOHM')
=== i2c.writeto_mem(addr, 1, b'\x30')
=== for i in range(10000):
===     pass
=== UCLK_p=adc2.read()*ADCbit
=== print('U=',UCLK_p,'V')
=== RX = UCLK_p/(3.3-UCLK_p)
=== print('CLK_p=',round(RX,2),'MOHM')
=== i2c.writeto_mem(addr, 1, b'\x38')
=== for i in range(10000):
===     pass
=== UCLK_n=adc2.read()*ADCbit
=== print('U=',UCLK_n,'V')
=== RX = UCLK_n/(3.3-UCLK_n)
=== print('CLK_n=',round(RX,2),'MOHM')
===
Pin(Pin.cpu.H5, mode=Pin.OUT)
Pin(Pin.cpu.F1, mode=Pin.OUT)
[32]
U= 3.24197584 V
Trg/Ctrl= 55.87 MOHM
U= 3.2427815 V
Trg/Ctrl_n= 56.67000000000001 MOHM
U= 3.24761546 V
Data2_P= 62.0 MOHM
U= 3.24600414 V
Data2_N= 60.12 MOHM
U= 0.00322264 V
P2A= 0.9800000000000001 kOHM
U= 0.005639620000000001 V
P1A= 1.71 kOHM
U= 1.54364456 V
CLK_p= 0.8800000000000001 MOHM
U= 0.90072788 V
CLK_n= 0.38 MOHM
>>>
```

GUI via TKinter

Suggested layout:



C:\documents_Sorina\VMMhybrid\VMMalmostfinal2.py - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

VMMalmostfinalSorina.py | VMMdb.py | VMMalmostfinal2.py

```
""" control gui prototype """

from tkinter import *
from PIL import ImageTk,Image
import sqlite3
# from docx import Document
# from docx.shared import Inches
#import tkFont


root = Tk()
root.geometry("1000x800")
root.title("VMM GUI")
root.iconbitmap('C:/documents_Sorina/VMMhybrid/VMMnew.ico')


# Message field on the top
VMM_mess = Label(root, text="Please connect HDMI cable to hyb
#VMM_mess.config(font)

VMM_mess.config(font=('Times', 24, 'bold italic'))
VMM_mess.pack(side=TOP, fill=X, ipady=10)


# Column 1 | 3 big red buttons

def col1Button1(event):
    print("column 1 button 1")

def col1Button2(event):
    print("column 1 button 2")

def col1Button3(event):
    print("column 1 button 3")

colFrame1 = Frame(root, height=750, width=150)
colFrame1.pack(side=LEFT, anchor=N, padx=10)



connCanvas = Canvas(colFrame1, height=750, width=150)
connCanvas.pack()


connY1 = 30

col1Button1Oval = connCanvas.create_oval(30, connY1, 130, con
col1Button1Text = connCanvas.create_text(80, connY1+50, text=
connCanvas.tag_bind(col1Button1Oval, "<Button-1>", col1Buttor
connCanvas.tag_bind(col1Button1Text, "<Button-1>", col1Button1)
```

Line 2, Column 1 Spaces: 4 Python

VMM GUI

Please connect HDMI cable to hybrid J2

START J2 connection

START J3 connection

Print Label

J2 (default port)

Pin1 OK
Pin2 OK
Pin3 OK
Pin4 OK
Pin5 OK
Pin6 OK
Pin7 OK
Pin8 OK
Pin9 OK
Pin10 OK
Pin11 OK
Pin12 OK
Pin13 OK
Pin14 OK
Pin15 OK
Pin16 OK
Pin17 OK
Pin18 OK
Pin19 OK

Power P1, P2

P1 = 3V, 0.2A
P2 = 3.6V, 1.5A

I2C scan VMM

IC chip found
ID=0xFFF0x3
uADC -1 found
uADC -2 found
Chip found at 0x39

J3 (Slave port)

Pin1 OK
Pin2 OK
Pin3 Error
Pin4 OK
Pin5 OK
Pin6 OK
Pin7 OK
Pin8 OK
Pin9 OK
Pin10 OK
Pin11 OK
Pin12 OK
Pin13 OK
Pin14 OK
Pin15 OK
Pin16 OK
Pin17 OK
Pin18 OK
Pin19 OK

Measurement field

this is very long text

Aux. buttons

VMM Power On / Off

VMM ID Readout

Label Print ID readout

Save to DB

check this box

Save these measurements

show selection

Exit

```
c:\documents_Sorina\VMMhybrid>python VMMalmostfinal2.py&

c:\documents_Sorina\VMMhybrid>python VMMalmostfinal2.py&
```

C:\documents_Sorina\VMMhybrid\VMMupdataDatabase.py - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

VMMalmostfinalSorina.py | VMMdb.py | VMMalmostfinal2.py | VMMupdataDatabase.py | updateInfoinDB.py

```
    P1_name_label = Label(editor, text="P1 Name")
    P1_name_label.grid(row=0, column=0, pady=(10,0))
    P2_name_label = Label(editor, text="P2 Name")
    P2_name_label.grid(row=1, column=0)
    ChipID_name_label = Label(editor, text="ChipID Name")
    ChipID_name_label.grid(row=2, column=0)
    J2Pin1_label = Label(editor, text="J2Pin1 Name")
    J2Pin1_label.grid(row=3, column=0)
    J2Pin2_label = Label(editor, text="J2Pin2 Name")
    J2Pin2_label.grid(row=4, column=0)
    J2Pin3_label = Label(editor, text="J2Pin3 Name")
    J2Pin3_label.grid(row=5, column=0)

# Loop through results
    for record in records:
        P1_name_editor.insert(0,record[0])
        P2_name_editor.insert(0,record[1])
        ChipID_name_editor.insert(0,record[2])
        J2Pin1_name_editor.insert(0,record[3])
        J2Pin2_name_editor.insert(0,record[4])
        J2Pin3_name_editor.insert(0,record[5])

    #create a Save Button to save edited record
    edit_btn = Button(editor, text="Save Record", command=update)
    edit_btn.grid(row=6,column=0, columnspan=2, pady=10, padx=10, ipadx=145)

#
def update():
    #Connect to defined database

    conn = sqlite3.connect('VMM_update.db')
    #create a curset who gets data and send the data
    c = conn.cursor()

    record_id = delete_box.get()

    c.execute("""UPDATE ChipIDes SET
            P1_name =:P1,
            P2_name =:P2,
            ChipID =:ChipID,
            J2Pin1 =:J2Pin1,
            J2Pin2 =:J2Pin2,
            J2Pin3 =:J2Pin3

            WHERE oid =:oid""",
            {
            'P1': P1_name_editor.get(),
            'P2': P2_name_editor.get()
```

addresses | Find | Find Prev | Find All

Line 211, Column 3 | Tab Size: 4 | Python

Picture Tools | VMMhybrid

View | Manage

PC › Windows (C:) › documents_Sorina › VMMhybrid

Name	Date	Type
DB4VMMupdate.PNG	05/02/2020 12:39	PNG File
VMM_update.db	05/02/2020 12:25	Data Base File
VMMupdataDatabase.py	05/02/2020 12:09	Python File
VMMdb.py	05/02/2020 12:04	Python File
VMMalmostfinalSorina.py	04/02/2020 18:52	Python File
VMMalmostfinal2.py	04/02/2020 17:57	Python File
Mantas2.py	04/02/2020 17:40	Python File
sheldon2.PNG	11:32	PNG File
sheldon1.PNG	11:30	PNG File

Update VMM database.com

P1 Name	3V, 0.2A
P2 Name)	3,6V, 1.5A
ChipID Name)	0xFFF0x3
J2Pin1 Name)	ok
J2Pin2 Name)	ok
J2Pin3 Name)	ok

Add record to Database

Show Record

Select ID

Delete Record

Edit Record

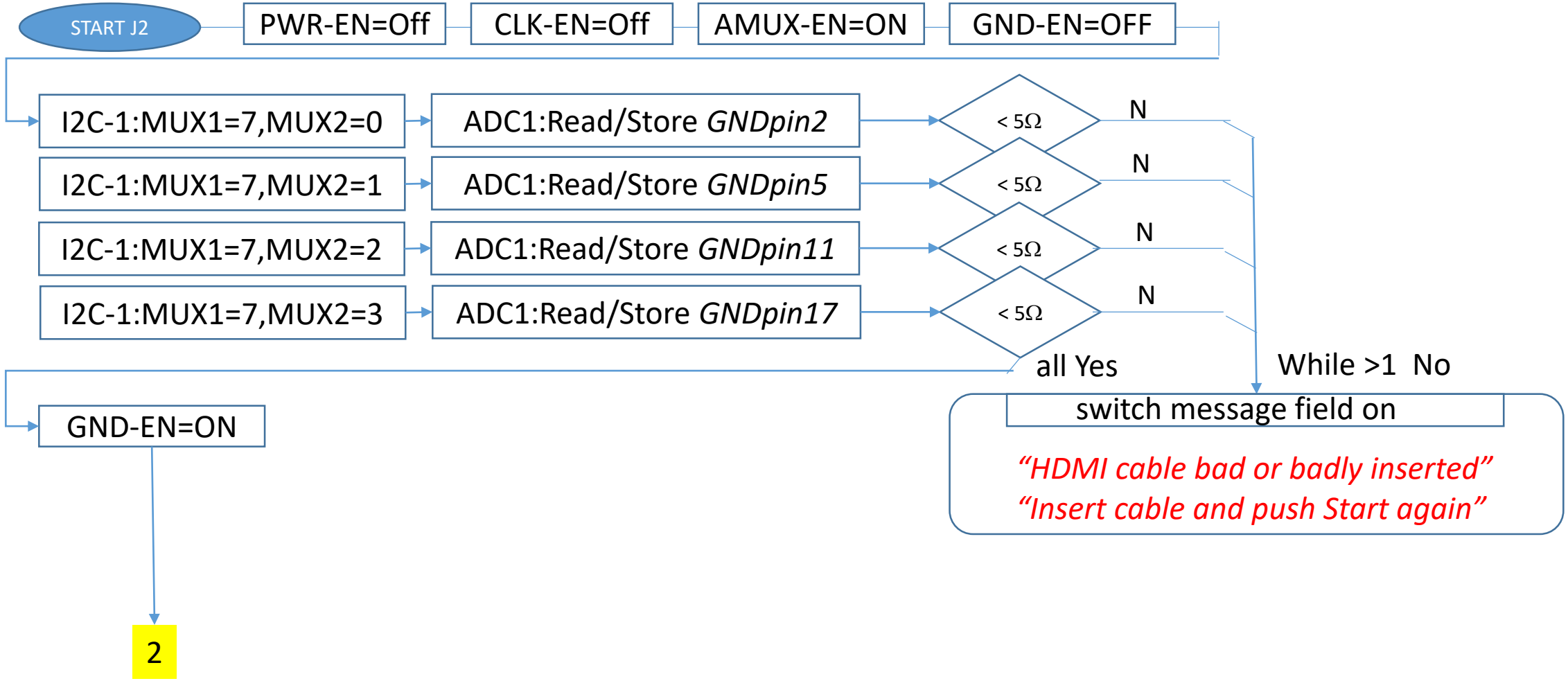
```
    return
  File "VM
    c.exec
sqlite3.Op
c:\document
Exception
Traceback
  File "C:
    return
  File "VM
    c.exec
sqlite3.Op
c:\document
Exception
Traceback
  File "C:
    return
  File "VM
    c.exec
sqlite3.Op
Exception
Traceback
  File "C:
    return
  File "VM
    c.exec
sqlite3.Op
```

```
line 1883, in __call__
line 1883, in __call__
e, :ChipID, :J2Pin1, :J2Pin2
line 1883, in __call__
```

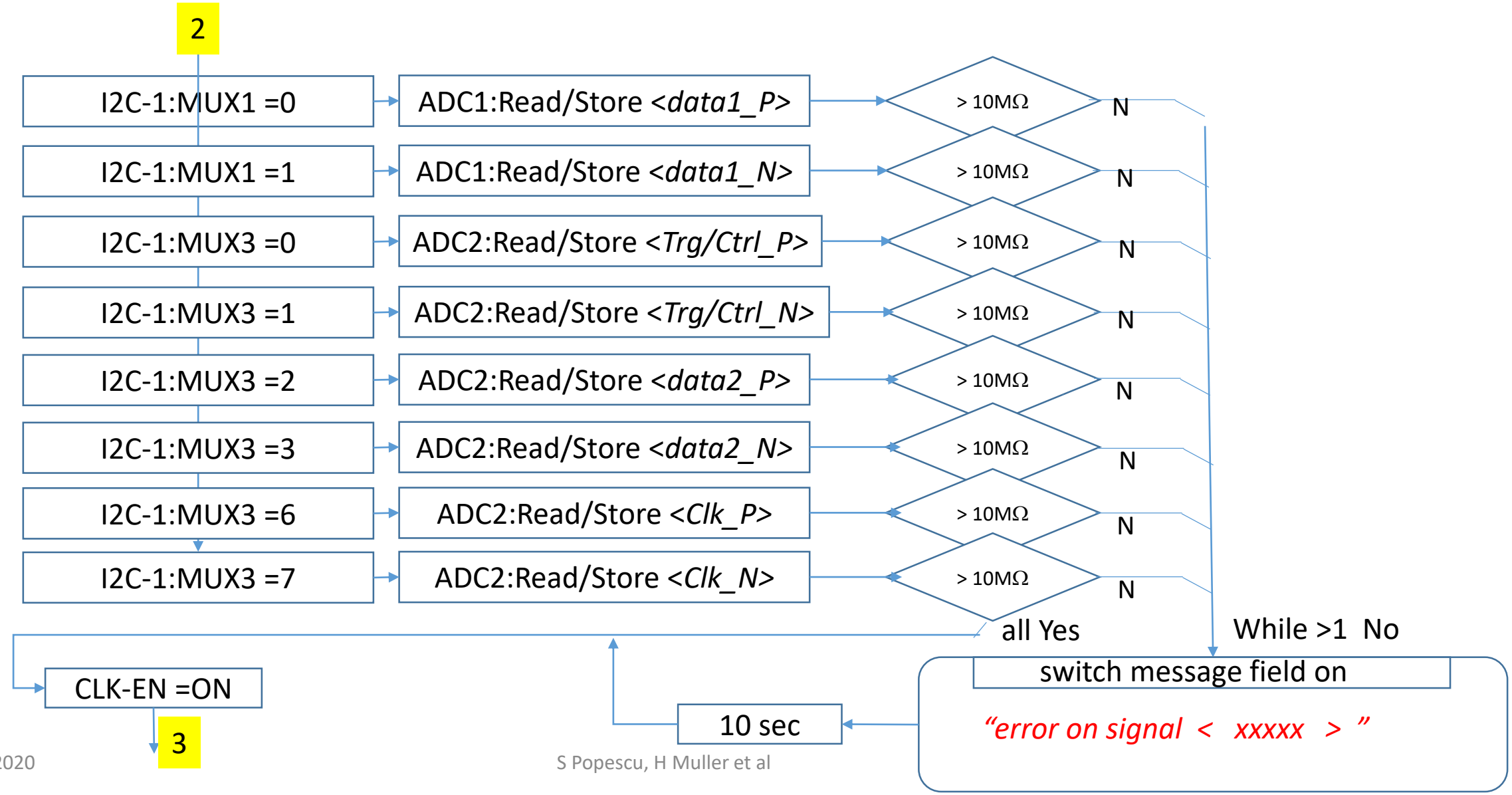
100%

Programming State Diagram (1)

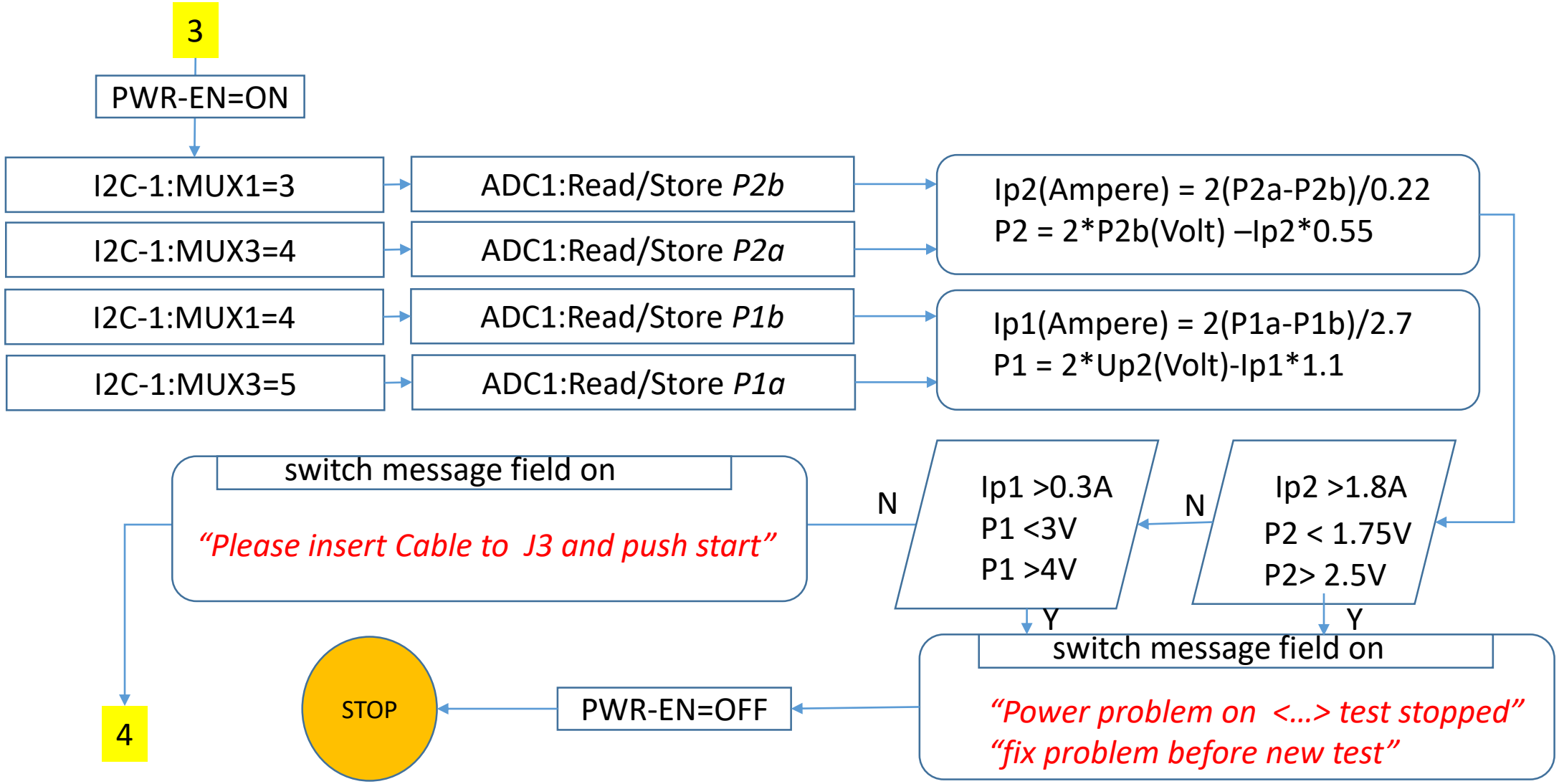
HDMI cable on J2



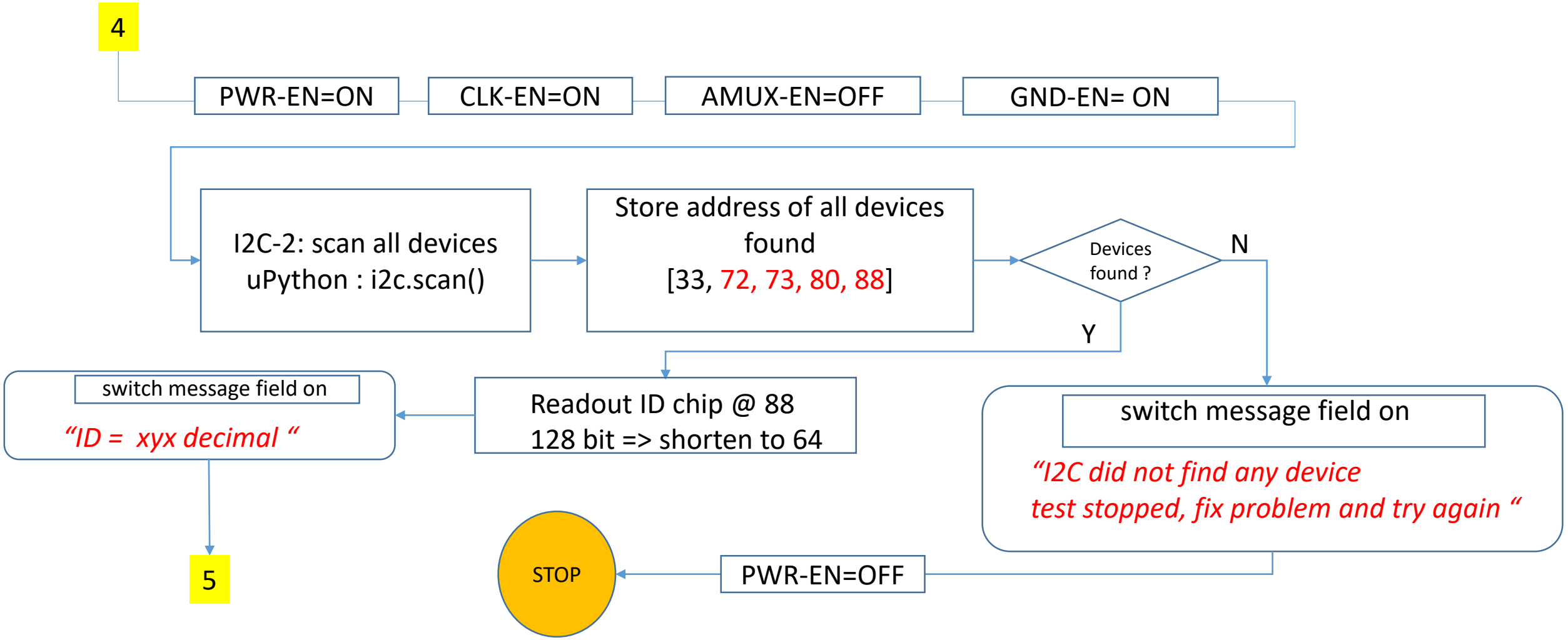
Programming State Diagram (2)



Programming State Diagram (3)

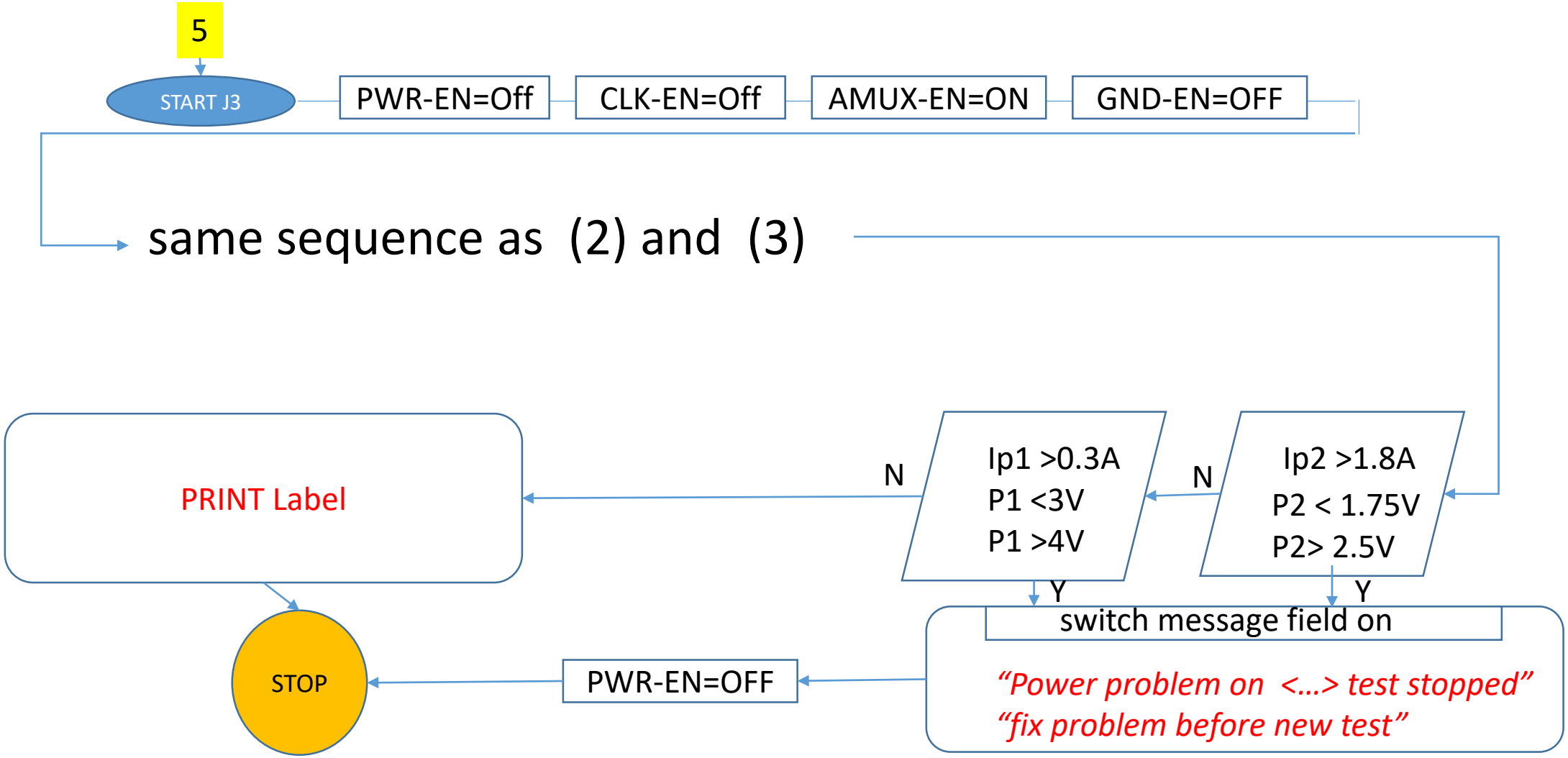


Programming State Diagram (4)



Programming State Diagram (4)

HDMI cable on J3



VMM handling after test

Case A QA passed => place OK label on VMM and store in **Accept box**

Case B QA failed => place BAD label on VMM and store in **Verify box**

Case C Test stopped: power failure => store VMM in Power **BAD box**

Case D Test stopped: ID scan failed => store VMM in **ID check box**

good VMMs

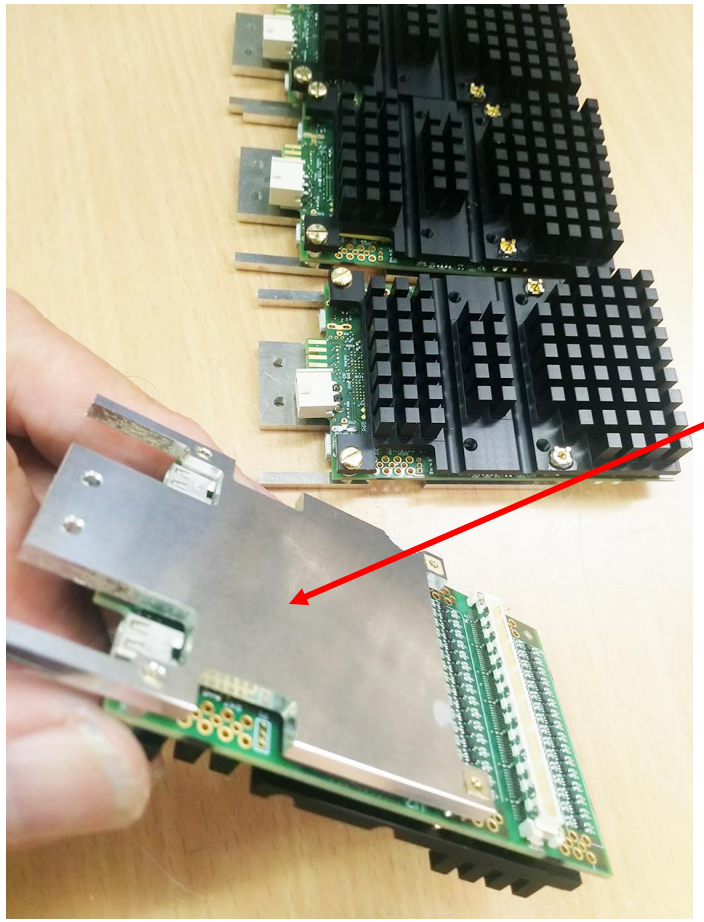
bad VMMs

verify/repair/re-test

Labelling

VMM Rev. 4.1 SRS Tech.
QA passed -J2 , QA passed -J3
01/05/2020
Ser. 30 128 7 24 100 16 0 97

Converted from 64 bit HEX



Place transparent labels here (VMM flat backside)

Labels = 38.1 x 21.2 mm Transparent
(Labels Avery code L7551-25)