Lab 1 - ETS Program Creation and Continuity Test

In this lab, we will design a test routine to perform the continuity test on the pic16f883 using the device interface board (DIB) for digital tests. A continuity test is typically the first test in a testing procedure. It ensures the device under test (DUT) is properly placed in the DIB on the ATE. It also verifies proper ESD circuitry. A sample test plan for the continuity test is provided in the appendix of this handout.

1. Look over the chapters of the datasheet related to ports A, B and C, particularly the block diagrams of each pin. Determine how each pin may be tested for proper connection and ESD circuitry.
    a. The continuity of analog and digital unit pins must be tested separately because they cannot be grouped together.
    b. The continuity test on supply pins must be tested separately as well.
2. Look at the DIB schematic. Determine which relays should be set to get the needed resources to the DUT, and which CBITs control those test relays.
3. Determine how many measurements will be needed for this test. This will determine the size needed when setting up the datasheet editor for this test.

For the first lab, you will create a new program that we will be adding to for the rest of the quarter, including the initialization routine for the program. In addition, you will add the continuity test to your project for the pic16f883.

Create and open a new program.

1. In the ETS shell, choose "Project" from the pull-down menu and select "New"
2. Click on "VS2005 Test Application"
3. Fill in the Project Name as "DigitalLab_xxx" where xxx represents your initials. In this manner, when we move the programs to the ETS machine, we can keep track of whose programs belong to whom.
4. Choose "Training" for the Family Name, and leave the Sub-Family Name field blank.
5. Click "OK"
6. In the ETS Shell, click on the "Select Test" button. Select "List by Project Name" and make sure the project you just created is selected.
7. Click on the "Develop" button. Visual Studio should come up with your program. There should be a number of lines of code already written for you.

When the project opens, there will be a .cpp file and header file already included in the source and header folders. Some code specific to the Eagle test system will be pre-generated. Following is a list and brief description of the pre-generated functions:

UserInit - All 1 time set-ups
UserExit - Leaving the testing environment and going back to shell
OnTestInit - Initialization before running a bank of tests for multiple chips (set valid sites from global switches)
OnTestExit - Leaving the environment for testing multiple chips
OnTestStartup - Initialization before running the bank of tests for a single chip
OnTestCompletion - Clean up (e.g. turn equipment off) after bank of tests have been performed for a single chip
OnLogData - This is the function that posts alarms. Place a break point here to use Raide to determine the state of the program when an alarm is being thrown.
OnFailSite - If a chip fails a test, stop testing and turn off all equipment associated with that site.
OnSot - Look for commands from the operator.
OnUtilError - used by ETS system to send error messages to the error handler
OnInterlock - checks for power and vacuum to the test head before sending power to the board.

Set-up the Initialization routine (UserInit).

The *UserInit* function is run only once for an entire lot of wafers. Therefore, all "one-time set-ups," (or stuff that should be the same for every chip) such as setting up the datalog, instructions to the operator, global switches, groupSites, groupsets, waveform generation, clock set-up, etc. should be executed here. Open the .cpp file under project_name -> Source Files

First, activate code that is necessary for this project by setting "if" statements to "true" or "1". ex:

```
#if 0
SetDlogSort(BY_TESTNUMBER):
#endif
```

set the 0 to a 1 to Sort the Datalog by the Test Number indicated in the project data sheet (pds).

Activate (i.e. set to a 1) msLogResultInit function to set the fields in the pds to the default fields. Go to the header file (DigitalLab_xxx.h) and make sure the variable NUM_SITES=1. The reason that this is set in the header file is to (i) improve readability and (ii) to easily change the number of sites later.

Activate SetGlobalSwitchName function. You will need 1 global switch to activate site 1. The code will look something like this:

```
SetGlobalSwitchName(GP1, "Enable Site 1"); //this function connects
//a name to a button on the Test Control Menu that may be used to
//activate or deactivate code.
```

Comment out the SetGlobalIntegerName function. The purpose of that function is to let the user set an integer that will be used later by the program. Typically, I have seen this used for repeatability tests.

Before the end of the UserInit function, you must perform the custom setup for this project.
We will continue to add to this code as the quarter progresses.

Notes about code:
(i) all ETS functions are highlighted in pink
(ii) standard C++ code is blue
(iii) the fields of an ETS command can be found quickly by double clicking on the command so that it is highlighted and then clicking on the "ACE Function Editor" button at the top. The function fields may be modified in the pop-up window or if you are uncertain what a field means, you can click on the question mark to get to the short version of the help files.

Custom Setup for Project (still in UserInit).
There are 3 kinds of source/measurement units that we will be using in the ETS ATE:
analog pin unit (APU): standard resource to either source a I or V and/or measure a I or V
smart pin unit (SPU): better resource to either source a I or V and/or measure a I or V
digital pin unit (DPU): digital resource to source a I/V and/or measure a I/V, or output logic vector

In order to efficiently control these units, we will group them so that they may be controlled together with one function call.
All the APU continuity pins (Port A by default, and MCLR) will be programmed in parallel so group them together as APU_CONT_GP.
Group pins together with the command "groupset" established in UserInit.

```
groupset(groupnumber, "groupname", "comma-separated resource list");
```

An actual example looks something like:

```
groupset(300, "Cont_GP", "APU15, APU19");
//resource numbers are tied to how the board is connected to the ATE
//resources (lconfig). Group numbers must be >256
```

The problem with this code is that it doesn't tell you a thing about what you are using and thereby what you're doing. We can fix this problem by redefining the numbers with names that make sense.
Ex: APU15 is the MCLR input. Call this pin APU_MCLR. However, the program recognizes numbers, so use the #define function in the .h file to redefine the resource numbers to the readable names.
Ex: the previous example is rewritten for readability using #defines in the .h file:

```
groupset(APU_CONT_GP, "Cont_GP", "APU%d, APU%d", APU_MCLR, APU_RA0);
```

where defined in the .h file:

```
#define APU_CONT_GP 300
#define APU_ MCLR 15
#define APU_ RA0 19
```

The %d does a variable substitution and replaces the %d with the number defined in the .h file, resulting in the same code presented previously. Add the remainder of the APU pins for this group.
When defining variables in the .h file, place them in between the "AFX_DEFINE_ACE_GROUP" blocks so that you can use the ACE Grouped Defines. You can redefine the name of the group by changing the name in the ( ). You can copy this block of code to add more groups, such as, use of 3 groups: (Site 1), (Group Defines), (Miscellaneous).

Note: the notation for the SPU's is SP100_%d, where %d refers to the SPU number.

Set up the Digital Groups using the Digital Vector Editor
The DPU's will be set up using the digital vector editor.
(a) Click on the "Vector Editor" button at the top. A box should pop-up "Do you want to create a new file?" Click yes.
(b) Label New Sheet by editing the name of the sheet. Expand the "Sheet Map" and right click on the sheet name to rename it. When you rename the sheet, it should also insert a label on the first line of code that matches the sheet name. Name this sheet "Continuity".
(c) Create Digital Pins using "insert -> New Pin". Name the pin (ex. RB0) and give the pin type as input. Leave the rest as the defaults. Repeat this action for all digital pins.
(d) Create Digital Groups using "insert -> New Group". Give the group name (ex.Continuity). Select the pins that you want in the group. Click the ">>" button to move them into the group. This pin group order should match your datalog order, with the first datalog matching the MSB. Click OK.
(e) Assign Resource to Pins using "Edit -> Pin Map Editor". This pop-up window allows you to map the digital vector pin names to actual DPU hardware. Assign the pins according to the DIB schematic. Click OK.

Create a new function
1. You MUST ALWAYS add and delete functions using the function wizard. The function wizard not only creates the function, but it also adds the function to the pds (datasheet) and makes sure that the cpp program and the pds file talk to one another.
2. Add a function by double clicking on the "Wizards" icon at the top.
3. Choose the tab "Test Function Wizard"
4. Click on the "Add Function" button
5. Choose a "Test Label" and "Function" name. I usually try to make these names the same and reflective of the test being performed. The "Function" will be the name of the function in the .cpp file and the "Test Label" will be the name of the label in the pds file.
6. Select the number of "rows" for the pds file. The number of rows is equal to the number of values you want to compare to a set of limits for a given site. For the continuity test, you want to test ESD protection in both directions for all port pins, plus Vdd. Therefore, you need 51 rows.
7. The rest of the fields should remain at the default values. For learning purposes, the function should be added to the .cpp file that you currently have running. If you have opened the wizard from that file, the default will be correct. The function should be added to the "ProductDatasheetFlow". You can set

up multiple Test Flows (e.g. order of tests to be run). We will always use the datasheet to determine our test flow. Having multiple test flows is useful when you have some tests that aren't run all the time because their intention is to make sure that there is no drift in the process, but they aren't production tests. You can choose the Test Flow and some test flows will have a different subset of tests than another.

8. Click OK and the new function should be added into your .cpp file.

Fill out the PDS

The .pds file, obtained by clicking on the "Datasheet Editor" button at the top, contains all of the information that you want to save in the datalog. This is the information that you want sent to the screen and saved to the files. It also contains the test limits. The measured value will be compared to the test limits and if it falls within range, the test will pass. If it is out of range, the test fails. The reason the test limits are stored in a different file is so that you can use the same program to test a family of parts with different .pds files for the different limits.

1. Modify the pds by clicking on the "Datasheet Editor" icon at the top.
2. Start filling out the .pds file. This is the first test, so the Test Number should be 1 for all rows.
3. We will separate the different measurements with the Subtest Number. Set these to the values 1-[max number of measurements].
4. Provide a Datalog Description with enough detail that you will know what each measurement is for. In this case, ORDER MATTERS! You must set these in same order as your groupset or datalogging will be harder later.
5. Set the low and high limits. Since you are measuring the diode drop, the resulting value should be between 1V and 0.2V. You may need to adjust this range. Note: you will have to be careful of the sign. If you are assuming a positive voltage, you are assuming that you are checking the diode to VDD and thereby are sourcing current from the instrument. The conditions for test must match the sign of the results. Note: current **out** of the instrument is positive.
6. Data Format is the number of characters that will be stored and how many characters are stored to the left of the decimal. Set this to 9.3. This corresponds to 9 characters with 3 left of the decimal place (e.g. xxx.xxxxxx).
7. Set the units equal to the units that will be measured. If you are datalogging a single measurement, it should be in the units measured (e.g. V for voltage and mA for current). If you have performed math on the measurements, you need to account for the resulting units.
8. Set the Low Fail Bin and the High Fail Bin. When I.C.'s are tested, they are typically sorted as to what passed and what failed. Physically, any chip that fails is scrapped, so further separation is unnecessary and impractical since there are only a limited number of physical bins to choose from. However, a product engineer might like to know which tests failed and how often so that he can propose improvements and choose which test to work on improving. This information can easily be tracked in software where there are a large number of resources.
   Therefore, there are "hardware bins", where the devices are physically sent when they pass or fail, and "software bins", where the information about each chip is stored. The "Low Fail Bin" and "High Fail Bin" fields in the table are the software bins. This can be set to any number between 2 and 512. 1 is reserved for passing devices. Then, to map the failing device to a hardware bin, you must fill in the table to the left (the binning table). If you choose "software bin = 7" for the failed continuity test of VDD in the pds table, then row 7 of the binning table is set up for continuity of Vdd. In the first column (HBin = hardware bin) you set the hardware bin for the failing device. Set this = hardware bin 3 (there are typically about 5 -6 hardware bins, but it depends on the handler). In the P/F/A column, set it equal to F so that the failed device will bin here. Set the description so that it is clear ("continuity Vdd").
9. Repeat the binning procedure for each of the continuity tests. Set the software bin different for each test, but set the hardware bin as the same.
10. Don't forget to set Software Bin 1 (SBin 1) and Hardware Bin 1 (HBin 1) for passing devices.
11. Save the .pds file and exit this file.

APU, SPU, DPU, and CBIT Utilities

In order to set up the hardware, we need to know what the standard APU, SPU, DPU, and CBIT functions are and how to use them. There are a lot of utilities for these functions. Here are some you may need today.

APU - analog pin unit

apuset (pin, mode, value, v_range, i_range, connection);

pin: select the APU number or use the #define reference for a single APU or a group of APUs

mode: APU_VIOFF: turn the APU off (used in step (vi))
APU_FV: force a voltage (used in step (i))
APU_FI: force a current (used in step (i))
APU_FVAWG: force a time-varying voltage pattern from the automatic waveform generator
APU_FIAWG: force a time-varying current pattern from the AWG value: specify the voltage value in V if in FV mode or the current value in mA if in FI mode. If not in FV or FI mode, this field is ignored.

v_range: sets the ranging of the instrument. Set as low are reasonable to minimize quantization noise but don't let a reading out of spec hit the max limit.
ranges are: APU_30V and APU_10V
you can find the quantization noise for each range by looking at the help files:
Help -> Index -> APU-12 -> Specifications -> accuracy

i_range: sets the current ranging of the instrument. All of the above applies here too.
ranges are: APU_100MA, APU_10MA, APU_1MA, APU_100UA, APU_10UA, APU_10UA

connect: This sets up how the APU connects to the pin. This resource can be stacked to get higher voltages by using the busses. This quarter, we will only use PIN_TO_VI, which basically sets each pin as a source/measurement unit

apumv (pin, numreadings, delay);

apumi (pin, numreadings, delay);

these functions are used to measure the resulting currents and/or voltages

pin: select the APU number or use the #define reference for a single APU or a group of APUs

numreadings: specify the number of readings to take and average (<4096)
optimized number for speed and accuracy is 4; however this may be insufficient for some higher precision measurements

delay: specify the amount of delay between measurement, in usec (>10) optimized number for speed and accuracy is 13

SPU: smart pin unit

sp100set (position, mode, value, v_range, i_range);

position:	select the SPU number or use the #define reference for a single SPU or a group of SPUs
mode:	SP_OFF: turn the SPU off SP_FV: force a voltage (used in step (i)) SP_FI: force a current (used in step (i)) SP_FVAWG: force a time-varying voltage pattern from the automatic waveform generator SP_FIAWG: force a time-varying current pattern from the AWG
value:	specify the voltage value in V if in FV mode or the current value in mA if in FI mode. If not in FV or FI mode, this field is ignored.
v_range:	SP_100V, SP_30V, and SP_10V
i_range:	The current range changes depending on the voltage range. SP_100V range: SP_500MA, SP_100MA, SP_10MA, SP_1MA, SP_100UA, SP_10UA, SP_1UA SP_30V range: SP_1A, SP_100MA, SP_10MA, SP_1MA, SP_100UA, SP_10UA, SP_1UA SP_10V: SP_2A, SP_200MA, SP_20MA, SP_2MA, SP_200UA, SP_20UA, SP_2UA

sp100mv (position, vmeasgain, imeasgain, numsamples, sampledelay);

sp100mi (position, vmeasgain, imeasgain, numsamples, sampledelay);

these functions are used to measure the resulting currents and/or voltages

position:	select the SPU number or use the #define reference for a single SPU or a group of SPUs
vmeasgain:	the amount to gain up the voltage measurement - this is a quick way to change a range for a new measurement : SP_MV_1X, SP_MV_10X, SP_MV_100X
imeasgain:	the amount to gain up the current measurement - this is a quick way to change a range for a new measurement : SP_MI_1X, SP_MI_10X, SP_MI_100X
numreadings:	specify the number of readings to take and average (<4096) optimized number for speed and accuracy is 4; however this may be insufficient for some higher precision measurements
delay:	specify the amount of delay between measurement, in usec (>10) optimized number for speed and accuracy is 13

sp100clamp (position, %PFSV, %NFSV, %PSFI, %NFSI);

this function voltage or current limits the source below the specified instrument range

position:	select the SPU number or use the #define reference for a single SPU or a group of SPUs
%PFSV:	positive clamp in percent of full scale range voltage
%PFSV:	negative clamp in percent of full scale range voltage
%PFSI:	positive clamp in percent of full scale range current
%PFSI:	negative clamp in percent of full scale range current

DPU: digital pin unit

dpinviset (Group Name, Force Mode, Force Value, Voltage Range, Current Range, Upper Voltage Clamp, Lower Voltage Clamp);

This function is used to set mode and range of the dpin group just like the apuset function.

- Group Name: group name defined in the vector editor, a string that must be in quotes
- Force Mode: choose force current (FI), force voltage (FV), RESTORE (return to state before this function was called), or LOAD (simulates a load on the pin)
- Force Value: Value of the current or voltage forcing
- Voltage Range: Choose the range from a pull-down menu
- Current Range: Choose the range from a pull-down menu
- Upper Voltage Clamp: Choose "No Clamp" from pull-down menu or enter a voltage that will set the max value that the pin will be allowed to go.
- Lower Voltage Clamp: Choose "No Clamp" from pull-down menu or enter a voltage that will set the min value that the pin will be allowed to go.

dpinmv(Group Name, Number of Samples, Sample Delay, Site);

This function measures the voltage (equivalent dpinmi command) for a digital pin.

- Group Name: group name defined in the vector editor, a string that must be in quotes
- Number of Samples: number of measurements to be averaged
- Sample Delay: Amount of time between samples
- Site: specify a site or choose MS_ALL from the pull-down menu

dpindisconnect (site, pin);

This function is used to disconnect digital pins. Must be used in OnTestCompletion and OnFailSite.

- site: to get all sites, use MS_ALL
- pin: pin name or pin group name (DPU_ALL_PINS for all digital pins)

CBIT: control bit

cbitopen (control bit(s));

this function opens the relays on the DIB (returns them to the position that is show on the schematic

- control bit(s): select the CBIT number or use the #define reference for a single CBIT or a group of CBITS

cbitclose (control bit(s));

this function closes the relays on the DIB (changes the position from that shown on the schematic)

Write Continuity Function Code

All functions will generally have the following steps:

(i) Set up the hardware. This involves closing necessary relays and setting the mode and ranging of the source/measurement units, and connecting them to the device.
(ii) wait for the circuit to settle
(iii) measure the resulting currents and/or voltages
(iv) if necessary, perform math on the measurements to obtain the proper spec.
(v) store the results into the datalog
(vi) turn the appropriate equipment off

Step (i) includes closing the necessary relays and initializing the equipment. Use the **cbitclose()** command to close the appropriate relays. You can use a CBIT groupset, if you have created one, or list the relays individually separated by commas. You will need to wait for the relays to close and charge to redistribute appropriately. The command you use to tell the equipment to wait is:

lwait (time); //where the time is in μsec.

Next, set up your continuity APU's, DPU's and SPU's and your supply SPU's as needed based on the continuity test plan and schematic. Example: set the APU continuity group to source 1mA of current into the APU continuity pins. You should expect a diode drop voltage measurement, which determines the range of the instrument. Note: positive current leaves the instrument.

apuset (APU_Cont_GP, APU_FI, 1, APU_10V, APU_10MA, PIN_TO_VI);

Set up the DPUs and SPUs similarly.

Step (ii): Wait for the circuit to settle with the input values set. At this time, estimate a wait time of 1000 μsec. You may attempt to adjust this time to improve (reduce) the time for your continuity test.

Step (iii): Measure the instrument readings and store the results. This is done with the **apumv()** command followed by the **groupgetresults()** command. You must always follow a measurement command with a getresults command or the measurement is lost. The measure voltage command is shown below with the default "ideal settings". We can change these settings on the ATE and see the impact on the measurements.

apumv (APU_Cont_GP, 4, 13); //this will record the measurements for the entire group

The groupgetresults command stores the measurement plus additional information into a datastructure that is set up in the same order as the APU_Cont_GP. Each element in the datastructure contains 4 fields (resource or instrument number, site, value, and whether the measurement Passed or Failed).

In order to properly access this datastructure, you must first set up a results structure array:

RESULTS_STR structure_name[size];

The size should be the total number of elements in the structure.

The groupgetresults command stores information in this datastructure. Its fields are as follows:

groupgetresults(structure_name, size); //where size and name should match the structure declaration

Once the information is stored in the result structure, you can perform math functions on it if necessary. This is not necessary for the continuity test.

Step (iv): Log the results. Once all data is stored, you must place the results into the datalog. The following command logs a group of results from multiple sites using the datastructure.

msLogResultAll (DSIndex, structure name, structure size, offset, index);
where the DSIndex is the datasheet index label tied to the .pds file (which is set upon entering the function) and the offset and index refer to how the array is accessed to get the site information. The offset field denotes the first element in the array that you want to datalog for this test. The index field denotes the number of elements that must be skipped to reach the same measurement on the other site (enter 1 since we only have 1 site). Therefore the result should look something like this to get the continuity tests for an APU:
msLogResultAll(DSIndex, APUcontinuitydata, 8, 0, 1);

This command should get the data for APU0 (offset=0). To get the next pieces of data, you must increment the offset. To grab all of the data easily, place the msLogResultAll function into a loop as follows:

```
for (i=0; i<3; i++)
{
        msLogResultAll (DSIndex++, APUcontinuitydata, 8, i, 1);
}
```

This will start the DSIndex and offset to the default and then increment the DSIndex from 1.1 to 1.2 and the offset from 0 to 1 mapping the structure to the proper datasheet entries.

Once the results are properly stored, you must turn the equipment off using the apuset, dpinviset or sp100set commands.

Other function setups.
OnTestCompletion:
Code MUST be added to turn off all equipment that has been used and restore relays to their initial state so that the next chip will not get blown due to the last state of the previous chip.
OnFailSite:
Code must be added to turn off all equipment for a given site if a site has failed. Since we only have one site, this may be the same as OnTestCompletion.

(a) Be sure to turn off the analog resources in the OnTestCompletion and OnFailSite routines.
(b) Turn off the digital resources using the *dpindisconnect* command. Use "MS_ALL" in the site field in OnTestCompletion and "site" in the site field for OnFailSite.

Write-up your results
Describe how your test results change from test to test. Record your test time. What percentage of your test time is in continuity versus userInit? Turn in your final code and datalogs. Discuss your test results.

Due: Wednesday, March 16, 2016

Appendix

Continuity Test Plan

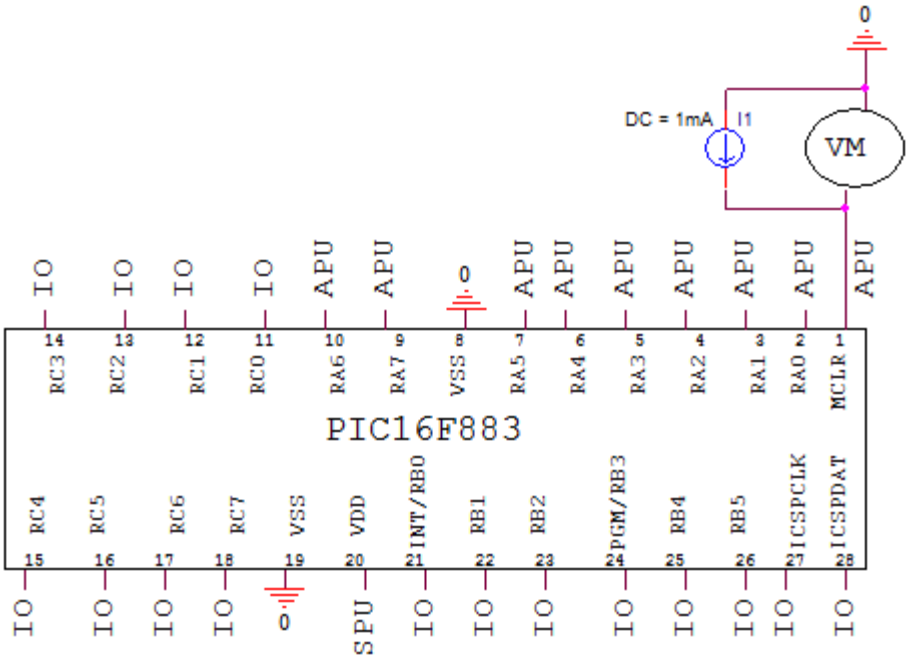

Figure 1 – Basic Layout for Pin to Pin Continuity Check

Eagle Test Section

Test procedure: Continuity()

1. Set relays so board configuration matches continuity schematic
2. Wait for relays to settle
3. Force all IOs, APUs, and SPUs to 0V

4. Force 1mA on an IO pin
5. Wait for transients to settle
6. Measure voltage on first IO pin
7. Datalog value (0.5V <= Value <= 1.2V)
8. Ground said IO pin
9. Repeat from step 4 for remaining IOs

10. Force 1mA on an APU
11. Wait for transients to settle
12. Measure voltage on first APU
13. Datalog value (0.5V <= Value <= 1.2V)
14. Ground said APU
15. Repeat from step 10 for remaining APUs

16. Force 1mA with _MCLR APU
17. Wait for transients to settle
18. Measure voltage on _MCLR
19. Datalog value (0.5V <= Value <= 1.2V)
20. Ground _MCLR

21. Force -1mA with VDD SPU
22. Wait for transients to settle
23. Measure voltage on VDD
24. Datalog value (-1.2V <= Value <= -0.5V)
25. Ground VDD SPU

26. Turn off resources
27. Disconnect relays