UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

Corso di laurea magistrale in
ELECTRONICS ENGINEERING

Building of a stress-test workbench for semiconductor device characterization

Candidato:
Gioele Bianchi

Matricola:
195419

Relatore:
Prof. Alessandro Chini

Correlatore:
Prof. Nicolò Zagni

Anno Accademico 2025-2026

Abstract

The topic of this thesis is instrument programming for semiconductor characterization with LabVIEW development environment for executing sequence and custom measurements and obtaining IV, IDSS and RON charts that define device stress-test results.

The instrument is Keithley 2612B by Tektronix and it can be controlled via PC with its proprietary software Kickstart 2. Kickstart 2 needs an expensive license, and one of the thesis goals is to find an alternative way to control instruments without Tektronix proprietary software. To achieve this goal, it was necessary to know instrument behavior with user manual and output signal sampling with oscilloscope when instrument was controlled by Kickstart 2. Development choice was to use LabVIEW as only a graphical user front-end, while peripheral configuration, action and execution were done with Lua / Instrument scripting language. LabVIEW only sets parameters of a script structure that is sent to an instrument. Communication and programming rely on Keithley 26xx LabVIEW drivers.

Charts' values are obtained with cycling sequenced measuring of 2 different measure blocks, personally created, on an experimental GaN HEMT device for high frequency power amplifier applications.

These blocks contain Lua script structures, and their parameters are set from the LabVIEW user panel. All block measures are saved in .csv files for every cycle. Based on them are charted graphs with Excel.

Sommario

L'argomento di questa tesi è la programmazione di uno strumento di caratterizzazione dei semiconduttori con l'ambiente di sviluppo LabVIEW per eseguire misure sequenziali e personalizzate e infine ottenere grafici IV, IDSS e RON, che costituiscono i risultati dello stress-test. Lo strumento utilizzato è il Keithley 2612B di Tektronix, che può essere controllato tramite PC con il suo software proprietario, come Kickstart 2. Kickstart 2 richiede una licenza a pagamento e una delle motivazioni della tesi è trovare un'alternativa al software di Tektronix per controllare lo strumento. Per ottenere i risultati, è stato necessario acquisire conoscenze sul comportamento dello strumento tramite il manuale utente e campionare il segnale di uscita con un oscilloscopio quando lo strumento era controllato da Kickstart 2. La scelta di sviluppo è stata quella di utilizzare LabVIEW solo come interfaccia utente, mentre la configurazione, le azioni delle periferiche e l'esecuzione sono state impostate tramite una versione modificata apposta per lo strumento in questione del linguaggio di scripting Lua. LabVIEW imposta solo i parametri che verranno passati alla struttura base script, per costruire lo script vero e proprio che verrà a sua volta inviato allo strumento tramite appositi blocchi LabVIEW. La comunicazione e la programmazione si basano sui driver LabVIEW per Keithley 26xx.
I valori dei grafici sono ottenuti attraverso cicli di sequenze di misura gestite da due diversi blocchi, creati personalmente, su un dispositivo GaN HEMT sperimentale per applicazioni come amplificatore in alta frequenza. Questi blocchi contengono la struttura base degli script Lua che vengono impostati dal pannello utente di LabVIEW. Le misure dei blocchi vengono salvate nel rispettivo file .csv per ogni ciclo. Su di essi vengono infine tracciati i grafici con Excel.

Table of Contents

1 Short introduction about device characterization and used instrumentation

In electronics field is very important the relationship between the electric current through a circuit, device, or material, and the corresponding voltage, or potential difference, across it.

It is called current voltage characteristic, typically it is a chart, and it determines basic parameters of the component and help to model its behaviour in an electrical circuit.

The simplest I–V curve is a resistor one, which according to Ohm's law exhibits a linear relationship between the applied voltage and the resulting electric current; the current is proportional to the voltage, so the I–V curve is a straight line through the origin with positive slope. [1]

The I–V curve of an electrical component can be measured with an instrument called “curve tracer”. There are many curve tracers on the market, for example with and without embedded screen.

The one mentioned by this thesis is Keithley 2612B by Tektronix.

2 Keithley 2612B description

Keithley 2612B is a curve tracer equipped with a proprietary power sourcing - measuring system, called SourceMeter®. [2]

It provides source and measurement capabilities in a single peripheral called Source-Measure Unit (also called a SMU). This combination simplifies test processes by eliminating synchronization and connection issues associated with multiple instrument solutions. A 2600B provides dc, pulse, and low frequency ac source-measure testing.

This version is 2612**B**, because it has 2 SMUs: SMUA and SMUB.

For measures that we are going to describe next are necessary exactly 2 SMUs.

This instrument can be controlled locally via front panel or remote via software

Each SMU provides:

•Source voltage: then we can measure and display current, sourced voltage, resistance and power.

•Source current: then we can measure and display voltage, sourced current, resistance and power.

•Measure resistance: Display resistance calculated with Ohm law from measured voltage and current.

•Measure power: Display power calculated from measured voltage and current.

Instruments can optionally memorize buffers, source given voltage list, or source given current list.

•Measure only (V or I): Display only voltage and current measurement.

Sourcing V

When configured to source voltage (V-source) as shown in the following figure, the 2600B functions as a low-impedance voltage source with current limit capability and can measure current (I-meter) or voltage (V-meter).

Sense circuitry is used to monitor the output voltage continuously and adjust the V-source as needed. The V-meter senses the voltage at the HI / LO terminals (2-wire local sense) or at the device under test (DUT) (4-wire remote sense using the sense terminals) and compares it to the programmed voltage level. If the sensed level and the programmed value are not the same, the V-source is adjusted accordingly. Remote sense eliminates the effect of voltage drops in the test leads, ensuring that the exact programmed voltage appears at the DUT. With 4-wire sensing enabled, both remote sense leads must be connected or incorrect operation occurs. 1

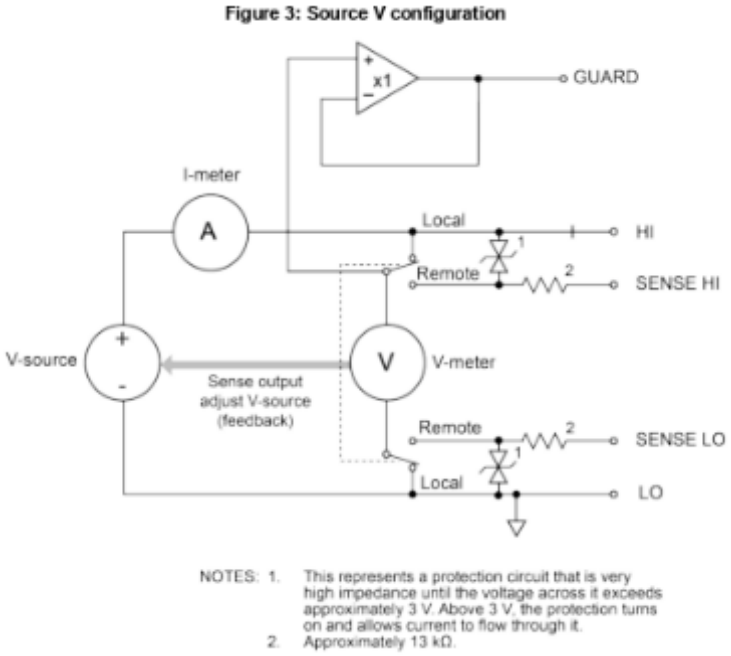

Figura 1: Voltage sourcing configuration from 2612B manual

Sourcing I

When the instrument is configured to source current (I-source), as shown in the figure below, the instrument functions as a high-impedance current source with voltage limit capability and can measure current (I-meter) or voltage (V-meter). For 2-wire local sensing, voltage is measured at the HI / LO terminals of the instrument. For 4-wire remote sensing, voltage is measured directly at the device-under-test (DUT) using the sense terminals. This eliminates any voltage drops that may be in the test leads or connections between the instrument and the DUT.

The current source does not require or use the sense leads to enhance current source accuracy. However, if the instrument is in 4-wire remote sense mode, the instrument may reach limit levels if the sense leads are disconnected. With 4-wire remote sensing selected, the sense leads must be connected or incorrect operation results. [2]

Figure 4: Source I configuration

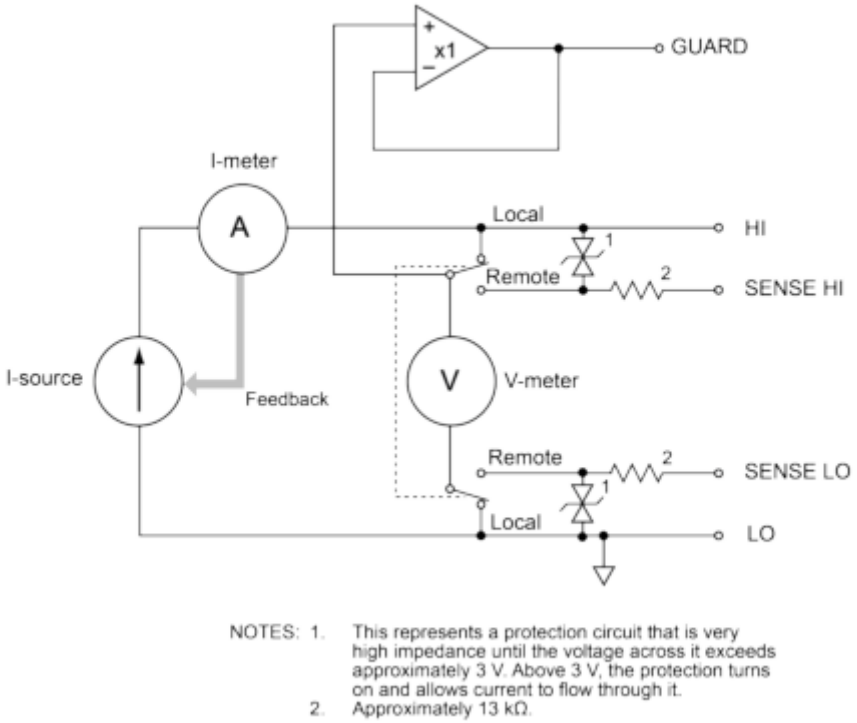

Figura 2: Current sourcing configuration from 2612B manual

Measure Voltage or Current only

The figures below show the configurations for using the instrument exclusively as a voltmeter or ammeter. As shown in the following figure, to configure the instrument to measure voltage only, set it to source 0 A and measure voltage.

Figure 5: 2600B measure voltage only

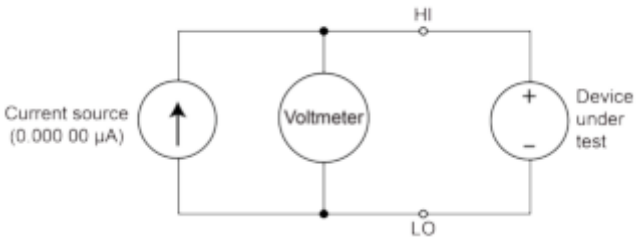

Figura 3: measure voltage only from 2612B manual

In the following figure, the instrument uses a 2-wire local sensing configuration and is set to measure current only by setting it to source 0 V and measure current. Note that to obtain positive (+) readings, conventional current must flow from HI to LO. [2]

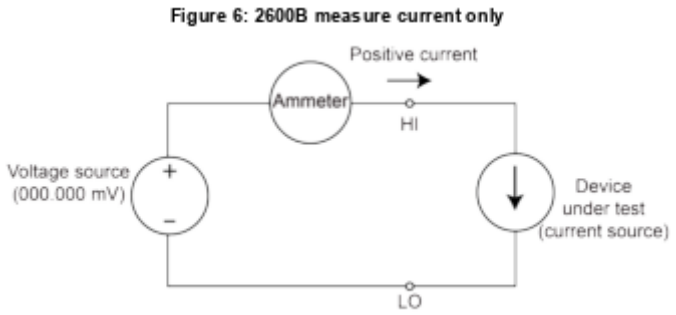

Figura 4: measure current only from 2612B manual

Limits

Figure 5 and table below illustrate the pulse regions for each SMU. The programmed current and voltage levels for both SMUs must fall within the same pulse region.

Measurements are given priority over source and display operations, so make sure that the measurement time does not exceed the allowable pulse width and duty cycle in a particular pulse region. [2]

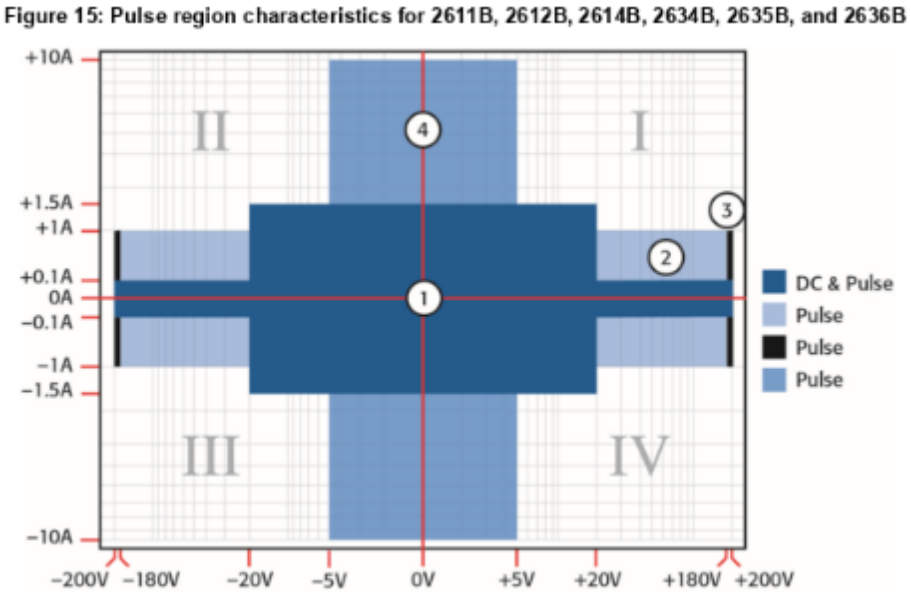

Figura 5: Pulse region characteristics from Keithley manual

Pulse region specifications

Region (quadrant diagram)	Region maximum	Maximum pulse width	Maximum duty cycle
1	100 mA at 200 V	DC, no limit	100%
1	1.5 A at 20 V	DC, no limit	100%
2	1 A at 180 V	8.5 ms	1%
3	1 A at 200 V	2.2 m	1%
4	10 A at 5 V	1 ms	2.2%

Source and measure ranges

The selected range affects the accuracy of the measurements and sourcing and the maximum signal that can be measured.

This selects the resolution for capturing ADC when measuring and output DAC when supplying SMU. [2]

2612B	
Voltage ranges	**Current ranges**
200 mV	100 nA
2 V	1 μA
20 V	10 μA
200 V	100 μA
	1 mA
	10 mA
	100 mA
	1 A
	1.5 A
	10 A (only in pulse mode)

Autoranging:

– *Sourcing*:

Setting autorange off puts the SMU on a fixed source range. The fixed range used is the present SMU source circuit range.

Setting autorange on puts the SMU source circuit into autorange mode. If the source output is on, the SMU immediately changes range to the range most appropriate for the

value being sourced if that range is different than the present SMU range.

Autorange is disabled if the source level is edited from the front panel. Setting the source range also turns off autorange when set by using the smuX.source.rangeY attribute.

Resetting the instrument selects the autorange on.

– *Measuring*:

Autorange process takes some time, and it isn't suitable for high-speed capture.

Setting autorange off puts the SMU on a fixed range. The fixed range is the present SMU measure range.

Setting autorange on, put the SMU measure circuit in autorange mode. It remains on its present measurement range until the next measurement is requested.

If source high capacitance mode is enabled, current autorange follow measures limit it and cannot be changed.

Figura 6: Control panel of Keithley 2612B

3 Using LabVIEW and Lua language to control the instrument

Main Keithley's Control program is Kickstart 2: graphically allow to perform power sourcing and measurement: Bias, sweep (and dual sweep, step sweep) and pulse

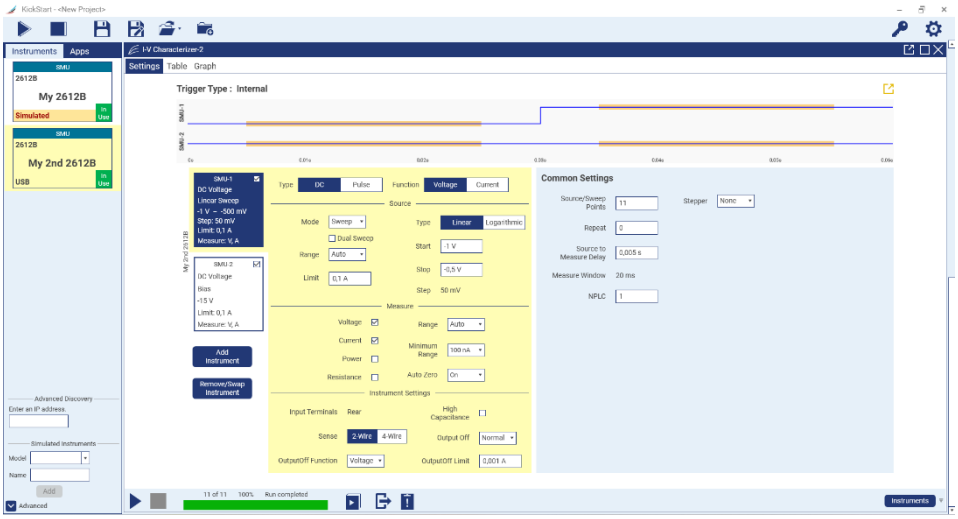

Figura 7: Kickstart 2 user interface

And sequence tool library, a graphic tool for sequencing actions.

At the end, TSP: a Test Script Processor. It is a scripting language IDE (structured text IDE) to make scripts with commands to send to Keithley.

Proprietary Graphic tools need a license and then cost. moreover, they do not allow to customize for example, power signal in terms of rising/descending edges timing.

Main low-medium level control for the Keithley 2612B is offered by TSP.

Fortunately, Tektronix gives access to instrument driver control blocks on LabVIEW:

They are a set of LabVIEW blocks created especially for controlling instruments.

In our case, with LabVIEW, you can configure quite every aspect of Keithley with driver blocks: starting from measures parameters, timing and Lua scripts to PC - connection and parallelism-syncing with other measurement benches.

One of the useful aspects of LabVIEW offers in general and for Keithley 2612B is building personalized sequence instrument actions (measures, power sourcing etc.).

The programming/scripting language used in TSP programming is a Lua 5.0 instrument customized tailored version.

Lua is a powerful programming/scripting language invented in 1993 at PUC-Rio (catholic university of Rio De Janeiro) and it can be considered a python-analogue for embedded high-performance applications and for Keithley is the best choice: it mixes high performance and evolved programming functions like array (table in Lua) managing.

LabVIEW is a graphical programming environment that has been developed by National instruments since 1986. In its first years it was a revolution because it simplified a lot the work of programming for its graphical approach reducing program development complexity.

It is composed of two perspectives:

User front panel and programming blocks editing panel.

LabVIEW programming elements are blocks.

So, in user interface you will put Input/output blocks: buttons, slider, numerical string boxes both in writing (input) and visualizing (output) and organize the interface.

In a programming perspective, you will put graphic programming structures like loops, Boolean, and arithmetic operation blocks. These programming blocks must be connected: input to output blocks, in this perspective, with wires. This “new programming method” could be easily or hard depending on what you need. If I need to program instrument actions quickly, structured text is better. But if I want a user graphical interface quickly and “easily”, instead hustling with Qt toolkit or another GUI framework, LabVIEW is best.

Disassembling Keithley’s blocks for LabVIEW, we can see that LabVIEW sends exactly TSP (Lua) commands: the idea is to create a graphical program that configures a TSP script and then it is sent to instrument, recreating a LabVIEW copy of Kickstart etc.

Keithley offers a huge quantity of blocks; each block contains one or few more Lua instructions lines that shoots to the 2612B with NI-VISA protocol.

The instrument blocks categories/functionality are:

- Power sourcing
- Measure actions
- Trigger action blocks
- Peripheral configurations
- Istrument communication control
- Low level settings like buffers config and reading

To understand Keithley behavior we must distinguish two macro categories:

- Time invariant Lua/TSP commands, also called non-trigger commands
- Time variant Lua/TSP commands, also called trigger commands

First are used for non-time critical measures and actions

Second, they are used when I need to sync actions and execute them at certain conditions, events usually at a specific time, for example a sweep: I need to produce voltage steps at a certain time. So, I need timer, “interrupts” and stimulus conditions to start an event like the timer count. Triggering allows you to source signals and capture measurements when an input signal or combination of input signals meets a set of conditions that you set. Triggering controls the timing of when source and measure operations happen during a sweep.

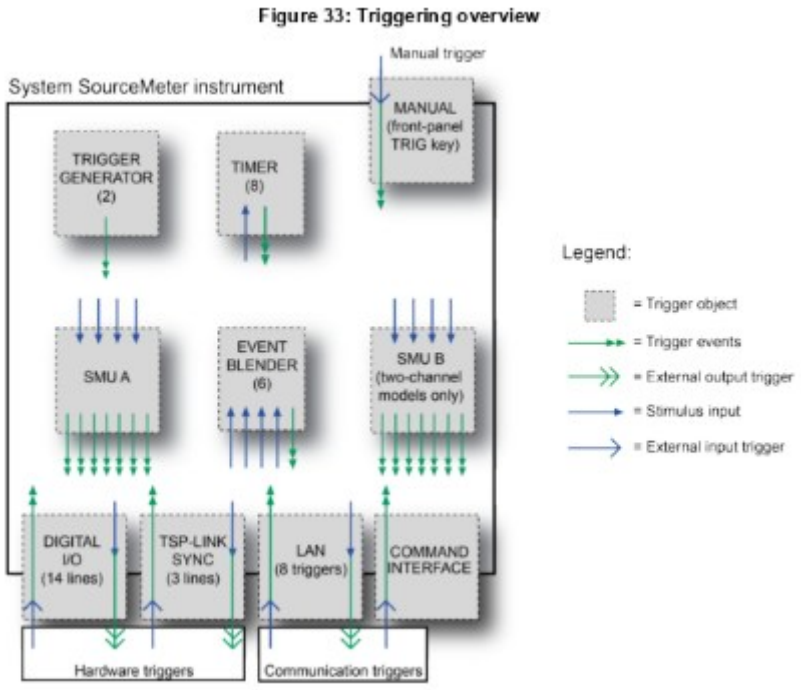

Figura 8: Triggering overview from Keithley reference manual

To know the instrument and its behavior, I started with Keithley on non-triggered action blocks that represent time invariant commands, sourcing power and measuring values for SMUA and SMUB. When I tried to make sweeps using non-trigger blocks, problems began. The are example factory scripts to make an only channel sweep. Instead, I wanted to sweep channel and a bias on the other, with voltage and current measures on each channel at same time

I needed to use triggered commands.

To avoid matryoshka effects with vi and sub vi increasing complexity I preferred to create a script (Lua text) with measure actions and parameters, then pass it to instrument with a special reading command for scripts reading and execution prebuilt for the 2612B.

4 Measure reverse engineering and Kickstart interface rebuilding:

4.1 Instrument logic structure and commands

We need to know how instruments work under various aspects: (used) commands and peripheral characteristics.

4.1.1 Structures

Trigger model: You can obtain very precise timing and synchronization between channels of multiple instruments using the trigger model to control the actions of the source-measure unit (SMU). To achieve such precise timing, use a static trigger configuration. When a static trigger configuration is not possible, you can use the interactive triggering method to control the timing and actions of the SMU. Both programming methods use trigger objects. Trigger objects generate and monitor trigger events. External triggers are possible using digital I/O, TSP-Link® synchronization lines, LAN, command interface, and the manual trigger (the TRIG key). [2]

In this case static trigger is sufficient, and we use only that.

EVENT_ID: action element ending flag. They become High/true when action is completed

Important EVENT_IDs are from timers, event blenders and end-action events

Trigger event IDs	
Event ID	**Event description**
smuX.trigger.SWEEPING_EVENT_ID	Occurs when the source-measure unit (SMU) transitions from the idle state to the arm layer of the trigger model
smuX.trigger.ARMED_EVENT_ID	Occurs when the SMU moves from the arm layer to the trigger layer of the trigger model
smuX.trigger.SOURCE_COMPLETE_EVENT_ID	Occurs when the SMU completes a source action
smuX.trigger.MEASURE_COMPLETE_EVENT_ID	smuX.trigger.MEASURE_COMPLETE_EVENT_ID

smuX.trigger.PULSE_COMPLETE_EVENT_ID	Occurs when the SMU completes a pulse
smuX.trigger.SWEEP_COMPLETE_EVENT_ID	Occurs when the SMU completes a sweep
smuX.trigger.IDLE_EVENT_ID	Occurs when the SMU returns to the idle state
digio.trigger[N].EVENT_ID	Occurs when an edge is detected on a digital I/O line
tsplink.trigger[N].EVENT_ID	Occurs when an edge is detected on a TSP-Link line
lan.trigger[N].EVENT_ID	Occurs when the appropriate LXI trigger packet is received on LAN trigger object N
display.trigger.EVENT_ID	Occurs when the TRIG key on the front panel is pressed
trigger.EVENT_ID	Occurs when a *TRG command is received on the remote interface GPIB only: Occurs when a GET bus command is received USB only: Occurs when a USBTMC TRIGGER message is received VXI-11 only: Occurs with the VXI-11 command device_trigger; reference the VXI-11 standard for additional details on the device trigger operation
trigger.blender[N].EVENT_ID	Occurs after a collection of events is detected
trigger.timer[N].EVENT_ID	Occurs when a delay expires
trigger.generator[N].EVENT_ID	Occurs when the trigger.generator[N].assert() function is executed

smuX can be SMUA for channel A or SMUB for channel B

STIMULUS: element action starter. Variable connected to an event_ID

Trigger stimulus	
Sitimulus (=..EVENT_ID)	**Stimulus description**
smuX.trigger.arm.stimulus	Causes arm event detector to enter in detect state
smuX.trigger.endpulse.stimulus	Causes end pulse event detector to enter in detect state
smuX.trigger.measure.stimulus	Causes measure event detector to enter in detect state
smuX.trigger.source.stimulus	Causes source event detector to enter in detect state
trigger.blender[N].stimulus[M]	Causes event blender event detector to enter in detect state
trigger.timer[N].stimulus	Causes timer event detector to enter in detect state

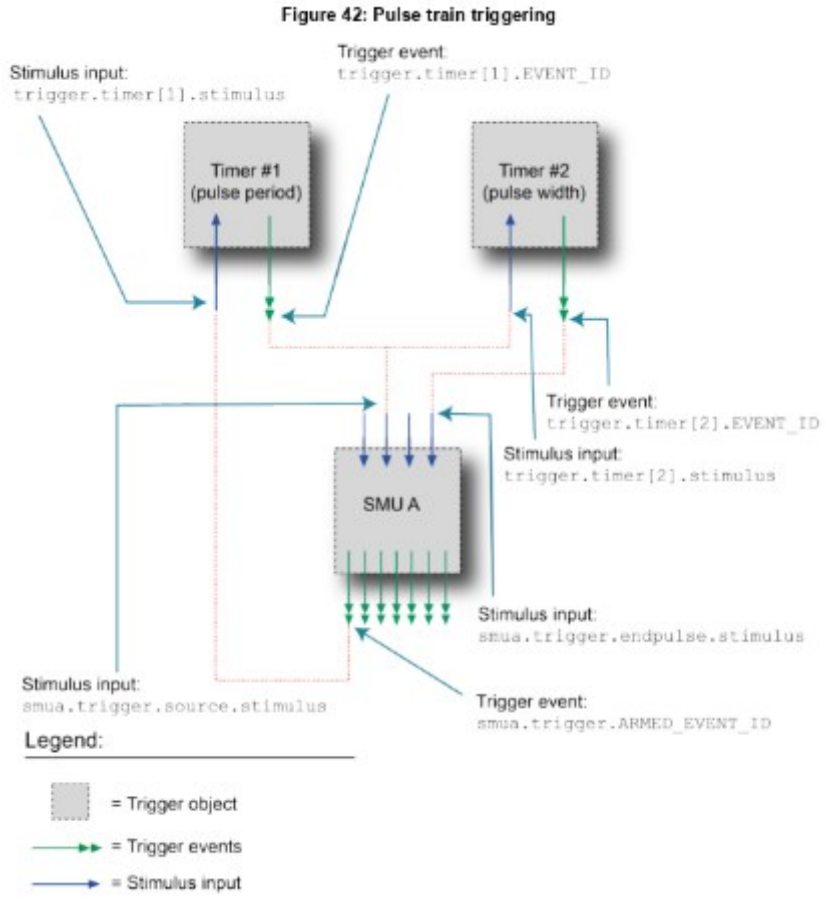

Figura 9: trigger model example of typical stimulus-event_id connection for pulse train generation from Keithley reference manual

Figure 9 shows how trigger structure is: a *relay race* between peripherals with stimulus and event IDs. In this case, there is a stand for a pulse train.

Remote trigger model (sweep diagram): The source-measure unit (SMU) in the 2600B The remote trigger model dictates the sequence of operation for the SMU when it is It is a built-in (don't know if hardware or software) double-nested for loop.

Outer loop defines how many sweeping actions (stairs and measure) we want to do, and inner loop defines stair steps quantity. [2]

Figure 10. Synchronous: power on SMU output, only 1 measure and then power off, used for sweeping SMU.

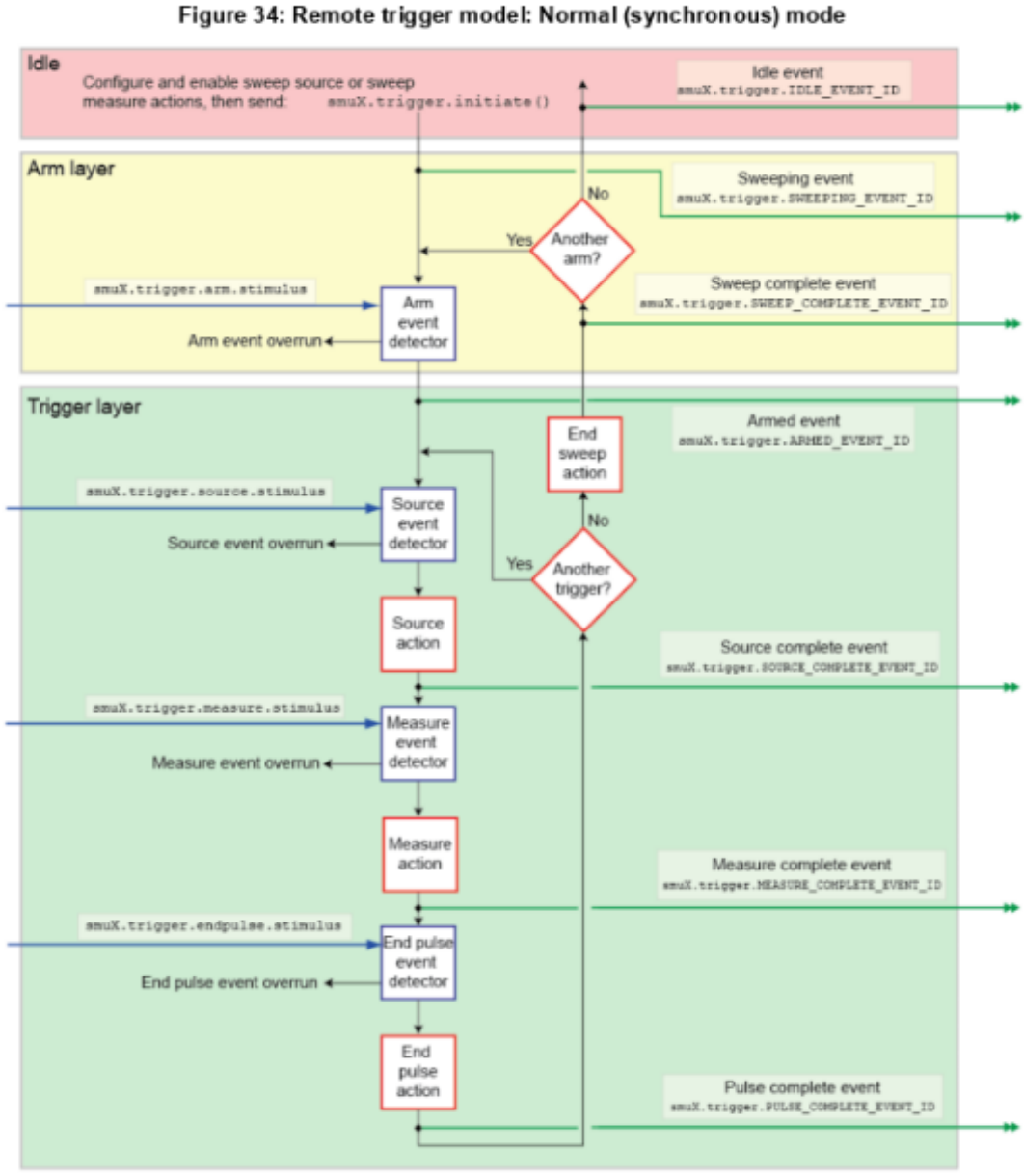

Figura 10: Synchronous trigger structure by Keithley reference manual

Figure 11. Asynchronous: power on SMU output, all measures we want to do and then power off. Used for stepping SMU in sweep-step measures.

Figure 35: Remote trigger model: Asynchronous mode

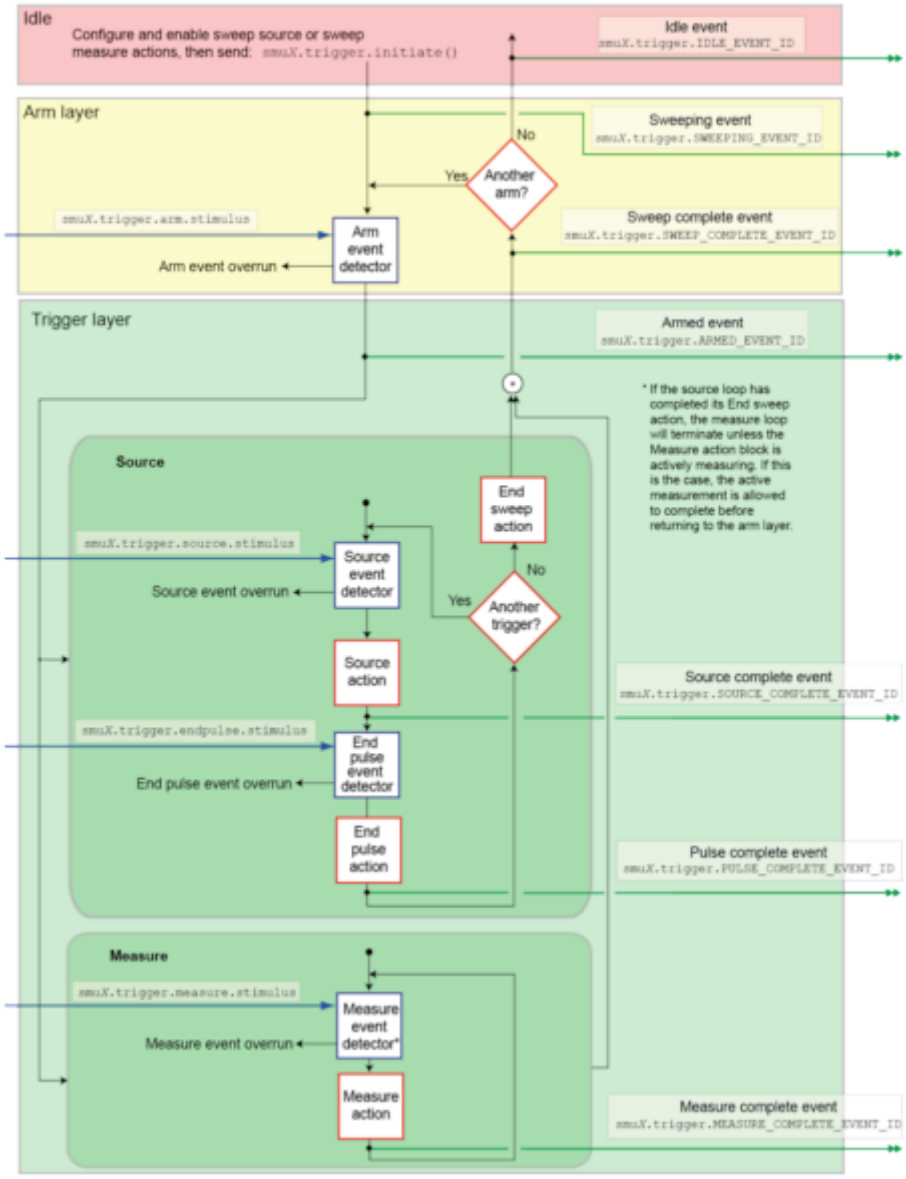

Figura 11: Asynchronous trigger structure by Keithley reference manual

Idle state	If a sweep is not in process, the SMU is in the idle state. Use the smuX.trigger.initiate() function to move the SMU from the idle state to the arm layer.
Arm layer	Begins a sweep. Each sweep starts and ends in the arm layer.
Trigger layer	All source, measurement, and pulse actions occur in the trigger layer. ▪ Source: Outputs the programmed voltage or current source value.

	▪ Measurement: Where the current, voltage, resistance, and power measurements occur. ▪ End pulse: The end pulse action sources the idle (or bias) level if the pulse mode is enabled.

Remote Trigger model actions	
Action	**Action description**
smuX.trigger.arm.count	Sets arm count in trigger model
smuX.trigger.count	Sets trigger count in trigger model
smuX.trigger.endpulse.action	Causes end pulse event detector to enter in detect state
smuX.trigger.endsweep.action	Causes measure event detector to enter in detect state
smuX.trigger.initiate()	Initiate sweep operation
Waitcomplete()	Wait trigger actions to complete

4.1.2 Peripherals

Timers: A timer is a trigger object that performs a delay when triggered. You can use timers to create delays, to start measurements, and step the source value at timed intervals. When a delay expires, the timer generates a trigger event. The 2600B has eight independent timers. In our they are used to define steps period, measure instant (source-measure delay) and SMU power-off instant in case of pulsed action. [2]

—> trigger.timer[N] (N is a number from 1 to 4)

Timer attributes:

– *Count*: trigger.timer[N].count = <number>

  The count sets the number of events to generate each time the timer generates a trigger event. Each event is separated by the delay set by the trigger.timer[N].delay command.

– *Delay*:trigger.timer[N].delay = <number/{1,2,....}>.

  You can configure timers to perform the same delay each time or set up a delay list that allows the timer to sequence through an array of delay values. All delay values are specified in seconds. A delay is the period after the timer is triggered and before the timer generates a trigger eve

– *Passthrough mode*: trigger.timer[3].passthrough = <true/false>

  When enabled, the timer generates a trigger event immediately when it is triggered. The timer generates additional trigger events each time a delay expires. If the pass-through attribute is disabled, the timer does not generate a trigger event until after the first delay elapses

– *Stimulus*: trigger.timer[N].stimulus=<...EVENT_ID> attribute to configure the timer to start a delay when a specific trigger event occurs. The programming example below illustrates how to configure a source-delay-measure (SDM) cycle

Analog-Digital Converter: The 2600B SMUs have integrating analog-to-digital converter (ADCs). The integrating ADCs use a ratiometric analog-to-digital conversion technique. Depending on the configuration of the integrating ADCs, periodic fresh reference measurements are required to minimize drift. The measurement aperture is used to determine the time interval between these measurement updates. To optimize operation of these ADCs, the instrument caches the reference and zero values for the ten most recent power-line cycles. [2]

Analog-Digital Converter controls:

– *Autozero*: smuX.measure.autozero = smuX.AUTOZERO_<OFF/ONCE/AUTO>
  - OFF —> Disable autozero. Old NPLC cache values are used when autozero is disabled
  - ONCE —> After immediately making one reference and one zero measurement, turns automatic reference measurements off.
  - AUTO —> Automatically makes new reference and zero measurements when the 2600B determines values are out-of-date.

  The analog-to-digital converter (ADC) uses a ratiometric A/D conversion technique. To ensure the accuracy of readings, the instrument must periodically obtain new measurements of its internal ground and voltage reference. The time interval between updates to these reference measurements is determined by the integration aperture being used for measurements. The 2600B uses separate reference and zero measurements for each aperture.

  By default, the instrument automatically checks these reference measurements whenever a signal measurement is made. If the reference measurements have expired when a signal measurement is made, the instrument automatically takes two more A/D conversions, one for the reference and one for the zero, before returning the result. Thus, occasionally, a measurement takes more time than normal. This additional time can cause problems in sweeps and other test sequences in which measurement timing is critical. To avoid the time that is needed for the reference measurements in these situations, you can use the smuX.measure.autozero attribute to disable the automatic reference measurements.

  Disabling automatic reference measurements may allow the instrument to gradually drift out of specification. To minimize the drift, make a reference and zero

measurement immediately before any critical test sequences. You can use the smuX.AUTOZERO_ONCE setting to force a refresh of the reference and zero measurements that are used for the present aperture setting.

The 2600B stores the reference measurements for the last ten NPLC settings that were used in a reference cache. If an NPLC setting is selected and an entry for it is not in the cache, the oldest (least recently used) entry is discarded to make room for the new entry. [2]

– *Nplc*: smuX.measure.nplc = <nplc_value>

Details This attribute controls the integration aperture for the analog-to-digital converter (ADC). The integration aperture is based on the number of power line cycles (NPLC), where 1 PLC for 60 Hz. The integration aperture; set from 0.001 to 25

NPLC caching speeds up operation by caching A/D reference and zero values for up to the ten most recent measurement aperture settings. Whenever the integration rate is changed using the SPEED key, or a user setup is recalled, the NPLC cache is checked. If the integration rate is already stored inthe cache, the stored reference and zero values are recalled and used. If the integration rate is not already stored in the cache, a reference and zero value is acquired and stored in the cache when the next measurement is made. If there are already ten NPLC values stored, the oldest one is overwritten by the newest one. When autozero is off, NPLC values stored in the cache are used, regardless of age. [2]

Event blender: The ability to combine trigger events is called event blending. You can use an event blender to wait for up to four input trigger events to occur before responding with an output event. You set the event blender operation using remote commands. You can program up to six event blenders for the 2600B. [2]

It is a Boolean peripheral operator, acts like a logic gate for event_id coming at the end of timer and peripheral actions. Mixing peripherals event id, event blender produces its event id useful for start actions

- Or: Generates an event when an event is detected on any one of the four stimulus inputs
- And: Generates an event when an event is detected on all the assigned stimulus inputs

—> trigger.blender[N] (N is a number from 1 to 6)

Event Blender attributes:

- *Orenable*: trigger.blender[N].orenable=<true/false>
  - True —> Or mode
  - False —> And mode
- *Stimulus*: trigger.blender[N].stimulus=<...EVENT_ID>

Buffers: Reading buffers capture measurements, ranges, instrument status, and output state of the Keithley Instruments 2600B. The 2600B has two default reading buffers for each channel (SMUA and SMUB). In addition to the default buffers, you can create user-defined reading buffers. You can use the reading buffers to acquire readings.

You can access reading buffers from the front panel (local) or over the remote command interface (PC). The default reading buffers can store more than 60,000 readings if you enable the options for timestamps and source values. To store 140,000 readings internally, you can disable the timestamps and source values. You can save reading buffers to internal non-volatile memory in the instrument or to a USB flash drive. [2]

For this application, we focus on *remote reading buffer programming*.

You can get readings by making overlapped or sequential measurements. Overlapped commands do not finish executing before the next command starts. Sequential commands complete execution before the next command starts executing. The measured value is not the only component of a reading. The measurement status (for example, “In Compliance” or “Overranged”) is also an element of data associated with a particular reading.

All routines that return measurements can store the measurements in the reading buffers. Overlapped measurements always return readings in a reading buffer. Nonoverlapped measurement functions can return single-point measurement values or store multiple values in a reading buffer.

A reading buffer is based on a Lua table. One of Lua/TSP difference from C is table as main structure for consequential data storing instead arrays.

Tables are dynamic allocated structures in memory, and abstractly consequential.

There isn’t a real consequential allocation in memory like C, Python etc.

Anyway, the measurements are accessed by ordinary array accesses.

Different from these languages, table indexing starts from 1 instead of 0. If rb is a reading buffer, the first measurement is accessed as rb[1] and the ninth measurement as rb[9]. The additional information in the table is accessed as additional members of the table.

The load, save, and write operations for reading buffers function differently in the remote state. From a remote command interface, you can extract data from reading buffers as the instrument acquires the data. [2]

Each source-measure unit (SMU) contains dedicated reading buffers:

- smua.nvbuffer1 (buffer 1 for channel A)
- smua.nvbuffer2 (buffer 2 for channel A)

- smub.nvbuffer1 (buffer 1 for channel B)
- smub.nvbuffer2 (buffer 2 for channel B)

Reading buffer commands

Commands to save and clear readings	**Description**
smuX.savebuffer(smuX.nvbufferY)	Saves the reading buffer to the nonvolatile memory on the 2600B
smuX.nvbufferY.clear()	Clears buffer 1/2.

Commands to store readings	**Description**
smuX.trigger.measure.iv(ibuffer, vbuffer)	Configures both current and voltage measurements to be made during a sweep, including where readings are stored; current readings are stored in ibuffer and voltage readings are stored in vbuffer.
smuX.trigger.measure.p(rbuffer)	Configures power measurements to be made during a sweep, including where readings are stored (rbuffer).
smuX.trigger.measure.r(rbuffer	Configures resistance measurements to be made during a sweep, including where readings are stored (rbuffer).
smuX.trigger.measure.i(rbuffer)	Configures current measurements to be made during a sweep, including where readings are stored (rbuffer).
smuX.trigger.measure.v(rbuffer)	Configures voltage measurements to be made during a sweep, including where readings are stored (rbuffer).
smuX.measure.v(rbuffer)	Makes voltage measurements; stores readings in rbuffer.
smuX.measure.i(rbuffer)	Makes current measurements; stores readings in rbuffer.
smuX.measure.iv(ibuffer, vbuffer)	Makes both current and voltage measurements; stores current readings in ibuffer and stores voltage readings in vbuffer.
smuX.measure.r(rbuffer)	Makes resistance measurements; stores readings in rbuffer.
smuX.measure.p(rbuffer)	Makes power measurements; stores readings in rbuffer.

Commands to store readings	**Description**
printbuffer(start_index, end_index, st_1, st_2, ... st_n)	Prints data from buffer subtables: ▪ start_index (starting index of values to print) ▪ end_index (ending index of values to print) ▪ st_1, st_2, ... st_n (subtables from which to print, each separated by a comma)

Buffer control attributes:

- *Appendmode*: smuX.nvbuffer1.appendmode = <0/1>

  The append mode is either off or on. When the append mode is off, a new measurement to this buffer overwrites the previous contents. When the append mode is on, the first new measurement is stored at the end of the existing data. This attribute is off when the buffer is created.

- *Collecttimestamps*: smuX.nvbuffer1.collecttimestamps = <0/1>

  When this attribute is on, timestamps are stored with readings in the buffer.

  This value, off or on, can be changed only when the buffer is empty. When the

  buffer is created, this attribute is initialized to off

Buffer reading attributes:

- *Readings*: printbuffer(start_index, stop_index, smuX.nvbufferY.readings)

  An array (a Lua table) of the readings stored in the reading buffer. This array holds the same data that is returned when the reading buffer is accessed directly; that is, rb[2] and rb.readings[2] access the same value.

- *Timestamps*: printbuffer(start_index, stop_index, smuX.nvbufferY.sourcevalues)

  If enabled, an array (a Lua table) of timestamps, in seconds, of when each reading occurred. These are relative to the basetimestamp for the buffer

4.1.3 Source-Measure Actions

Basic source-measurement procedures can also be performed through a remote interface. To do this, send the appropriate commands. The following table summarizes basic source-measure commands. [2]

Legend:

smuX= smua/smub

Y=i/v

Y_1=i/v/p/r

Source: powering commands

- **Triggered**:

Command	Description
smuX.trigger.source.action = <ENABLE/DISABLE>	enables or disables sweeping the source (on or off)
smuX.trigger.source.<linearY/listY/logY> = <(start,stop,points)/({n1,n2,...nn})>	Set current/voltage source value
smuX.trigger.source.limitY_1 = <*level*>	Set current/voltage/power limit.
smuX.trigger.source.stimulus	Causes source event detector to enter in detect state. Update the output value (sweeping)

- **Non-triggered**:

Command	Description
smuX.source.autorangeY=smuX.AUTORANGE_<*ON/OFF*>	Enable current source autorange.
smuX.source.func = smuX.OUTPUT_<*DCVOLT/DCAMPS*>	Select voltage/current source function.
smuX.source.levelY = <*value*>	Set current/voltage source value
smuX.source.limitY_1 = <*level*>	Set current/voltage/power limit.
smuX.source.output = smuX.OUTPUT_<*ON/OFF*>	Turn on/off source output
smuX.source.rangeY = <rangevalue>	Set current/voltage source range. Select correct DAC resolution
smuX.sense = smuX.SENSE_<LOCAL/REMOTE>	Select local sense (2-wire/4-wire)
smuX.source.lowrangeY = <lowrange>	Set lowest voltage/current source range for autorange.

Measure: capturing commands

- **Triggered**

Command	Description
smuX.trigger.measure.<v/i/r/p>(dest_buffer)	Configures voltage measurements to be made during a sweep, including where readings are stored (dest_buffer).
smuX.trigger.measure.iv(ibuffer, vbuffer)	Configures both current and voltage measurements to be made during a sweep, including where readings are stored; current readings are stored in ibuffer and voltage readings are stored in vbuffer.

- **Non-triggered**

Command	Description
smuX.measure.autorangeY = smuX.AUTORANGE_<ON/OFF/FOLLOW_LIMIT>	Enable current/voltage measure autorange

	(Autorange on take some time to match exact range)
smuX.measure.rangeY = <rangeval>	Set current/voltage measure range.
smuX.measure.lowrangeY = <lowrange>	Set lowest current measure range for autorange.
<Reading_var> = smuX.measure.Y_1()	Request a current/voltage /power/resistance reading.
smuX.measure.autozero = smuX.AUTOZERO_<OFF/ONCE/AUTO>	This attribute enables or disables automatic updates to the internal reference measurements (autozero) of the instrument.
smuX.measure.nplc = <value>	This command sets the integration aperture for measurements. set from 0.001 to 25

4.2 Measurement setup with oscilloscope to analyze output waves

My setup consisted of:

- MPI workbench with custom test board, mounting a jfet j175
- Keithley 2612B
- 4 channel Pico 2000
- Portable Personal computer
- Labview 2025
- Lab's Pc with kickstart and pico software

To know what signal is generated by the instrument controlled with kickstart, we need an oscilloscope. This step is necessary to verify if our LabVIEW program sends same actions to the 2612B against kickstart generated signal.

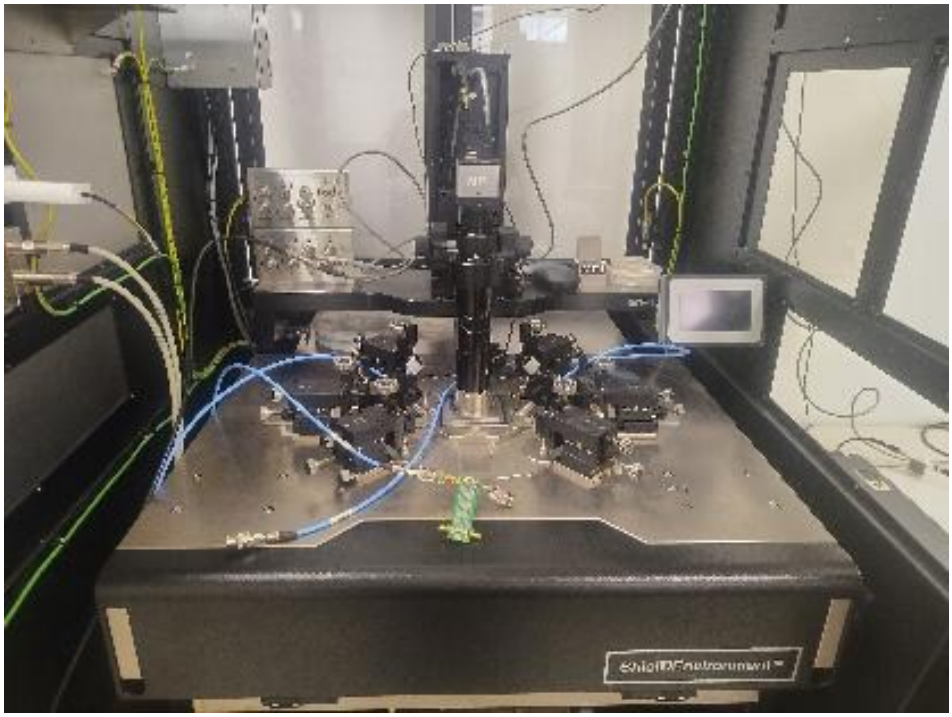

Figura 12: MPI probe station, with BNC connectors, for instrument and device under test

Figura 13: PICO oscilloscope 2000 series

Figura 15: Working setup

Figura 14: Keithley 2612B and LabVIEW at work

Connections: (Figure 16)

Device Under Test to Keithley:

Pin 1: drain/source (depending by transitor) → SMUB

Pin 2: gate → SMUA

Keithley to oscilloscope

SMUB → channel B

SMUA → channel A

Keithley to PC

USB → PC

Pico to PC

USB → PC

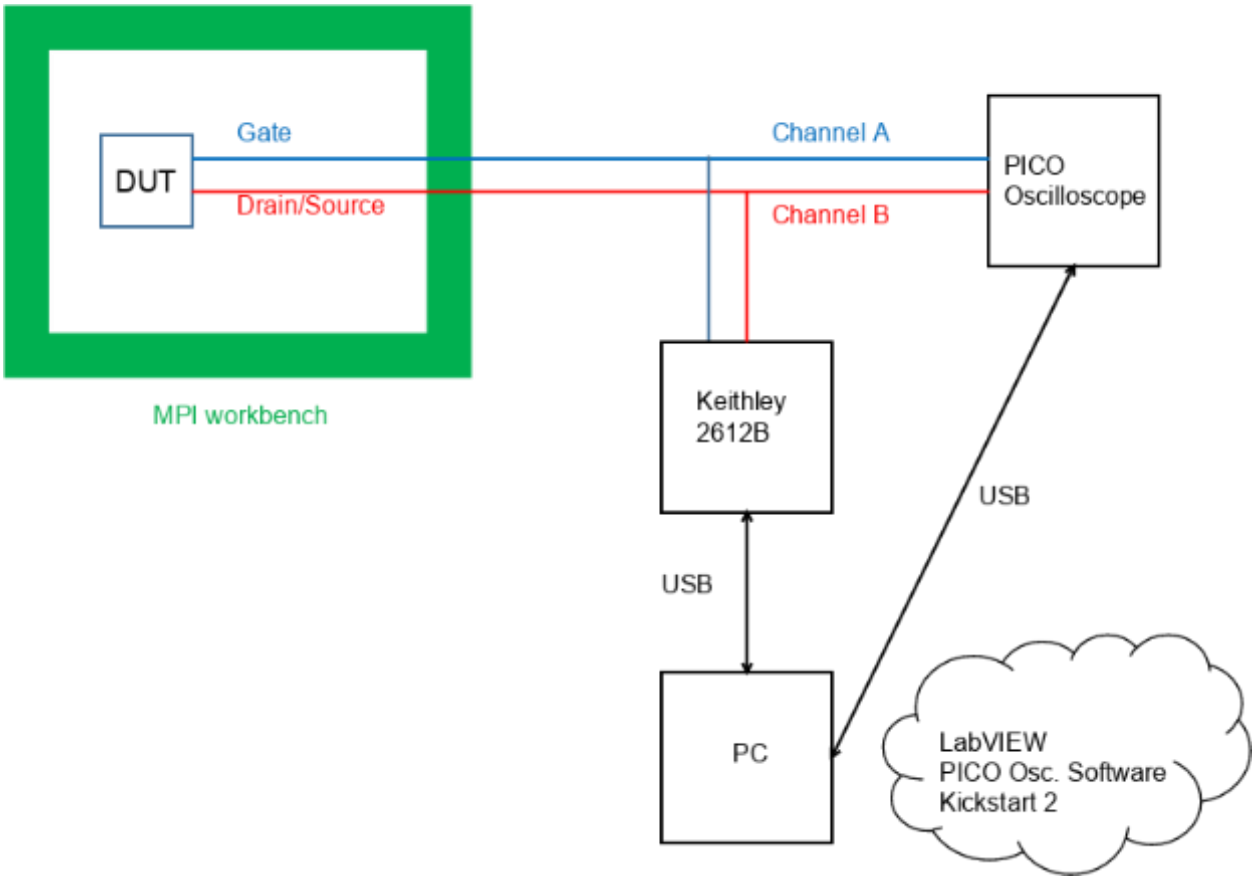

Figura 16: working setup connections diagram

4.3 First sweep prototype

To learn how to program the instrument, we started from LabVIEW's Keithley examples Figure 17; current measure example, Figure18: this example is built upon Keithley driver's blocks.

All these sourcing-measuring programming tests below were made on j175 transistor, to avoid breaking our next GaN HEMT candidate.

As said before each block contains one or more Lua/TSP lines according by function block is going to realize.

To start programming, blocks used were measuring configuration, current and voltage measure actions, voltage source and source configuration blocks, and instrument connection settings block.

First programs were developed to control source output and make simple current and voltage measurements, not time critical (trigger).

The structure is:

1) Chronological sequence structure: Open communication block at the beginning and close communication block at the end
2) While loop to execute script without reinitializing communication every time
3) Case structure loop to run program when you user wants
4) Inside there are elementary action and reading blocks. The script-execution block is added in the program second version

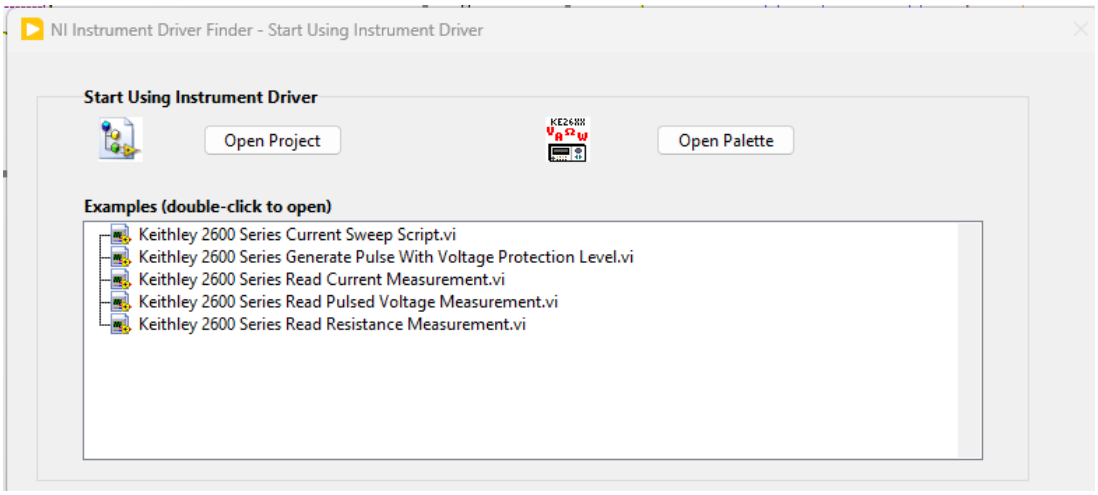

Figura 17: Keithley's driver example list

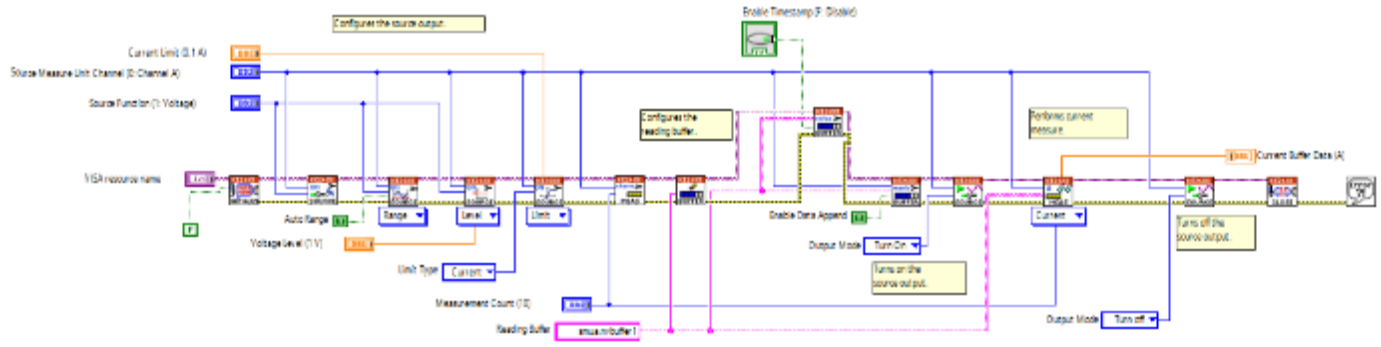

Figura 18: Read current measurement example

Trying to make drain sweep was a problem: Keithley gives only some unuseful example sweeping scripts: they are only demo and it is not possible sync with other actions to the other SMU. They are called factory Kiscript.

KISweep factory script

The KISweep factory script provides simple sweep test programming and shows how to use the sweeping function.

This script is made up of the following functions. Access these functions from the front panel or the remote interfaces. The following functions make up the KISweep factory script:

SweepILinMeasureV() (on page 9-359)
SweepIListMeasureV() (on page 9-360)
SweepILogMeasureV() (on page 9-361)
SweepVLinMeasureI() (on page 9-362)
SweepVListMeasureI() (on page 9-364)
SweepVLogMeasureI() (on page 9-365)

Figura 19: KISweep factory scripts explaining

In fact, trying to make sweep one SMU and making non triggered measure on the other SMU didn't work.

First, SMUs "turn on" priority was decided by LabVIEW multithreading and second without synchronization command it was impossible to make synched measures on Biased voltage SMU.

The only and right way is working with triggers and sweep nested trigger loop.

The manual gives information about trigger commands and schematics on trigger nested loop.

In trigger structure the execution goes on with event_ids and actions stimulus.

We need to coordinate, connect event_ids to actions stimulus.

Keithley manual doesn't give complete example of a sweep-capture measurement, so we checked out on Keithley's GitHub repository. [3]

The most important step is finding a Lua/Tsp tutorial script. [4]

In script there is the PulsedSweepVDual function that executes 2 sweep-capture measurements on SMUA and SMUB at the same time. I have taken the function and created a new Lua script file with only that part.

The script uses sweep trigger nested loop to generate a pulse stair signal on output

To test it on my instrument, I removed function structure and transformed it into a normal script. For simplicity is made a sweep list of a fixed value for SMUA and we made sweeping only SMUB. The script was composed of:

- SMUA and SMUB settings declaration
  - Idle levels and compliance
  - Range and zero settings
- Timer setup:
  - First timer: sets duration of the pulse
  - Second timer: sets source-measurement delay, the delay between smu turn on and capturing value
  - Third timer: turn down pulse, it turns off SMU and coincides with end of the captured window (integration time)

  Every timer is configured with countdown time, number of pulses per cycle, starting or ending pulse production and action event_id connected to timer action stimulus

- SMUA and SMUB trigger configuration:
  - Set trigger sources (voltage or current) and compliance levels
  - Set trigger measurement settings:
    - Type of measure (v/i/r/p/iv overlapped)
    - Destination buffer
    - Synchronous/Asynchrounous mode
  - Set starting pulse, end pulse and measurement stimulus
  - Event blender

  Trigger structure is like a black box: once configured, it needs a command to start and a command to wait for the finishing of all the operations

  The important setting to enable sweep measurement is Synchronous measurement: SMU.ENABLE. In sweep mode, we need SMU powered ON, measure (capturing) and then power off, then we execute next voltage step with same cycle, on measure and off until the end step.

- Main execution commands:
  - Command to start and wait finishing trigger actions.
  - Commands to turn on and turn off SMUs

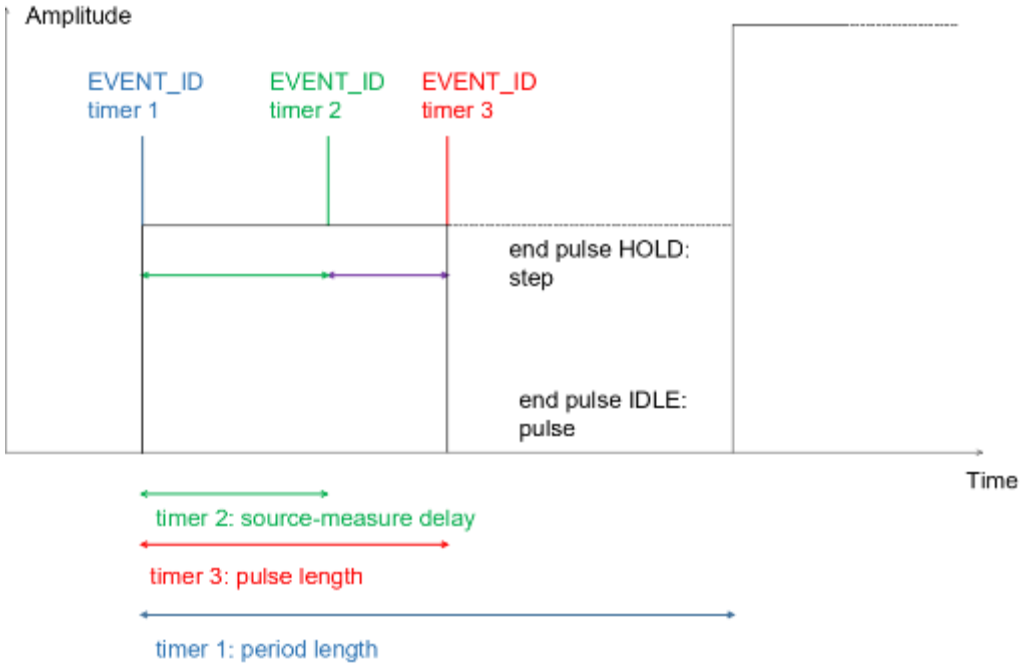

Figura 20: pulse eventIDs diagram

Figure 20. Pulse eventID diagram is like a relay race:

- Timer_1 start —> eventID 1 arrives at pulse beginning:
  - Timer_2 starts. Timer 2 expring arrives eventID 2 → starts Meas capturing
  - Timer_3 starts. Timer 3 expring arrives eventID 3 → end Meas capturing

(end pulse, but stay HIGH for Hold command)

- Timer1 expires and restart → eventide 1 arrives at pulse

  Repeat, until trigger count and arm count reached

Since we want to have a sweep dc, we set end pulse hold to main voltage constant until next level updating (next step).

To test the script, I took “Series Current Sweep Script” from Keithley example list and I changed code inside. Figure 21.

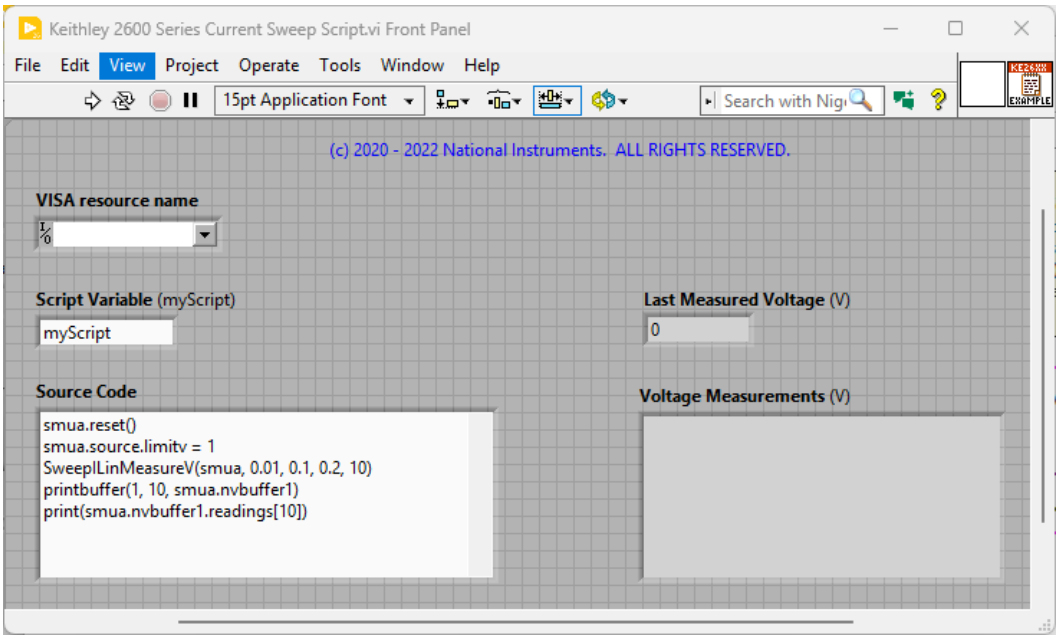

Figura 21: Current Sweep Factory Script example

It is composed by a string input and script-sending special block called “Create Script”.

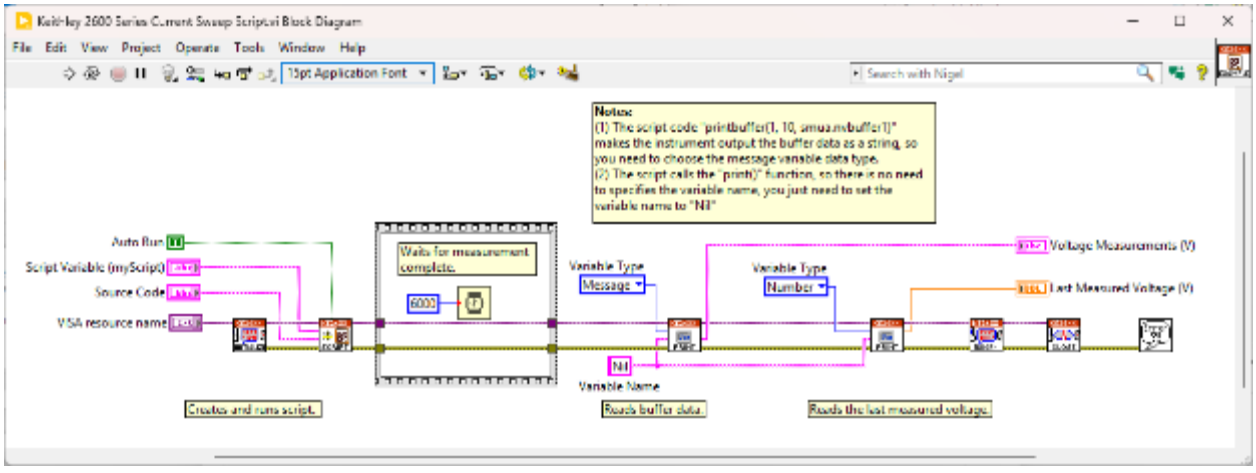

Figura 22: Current Sweep Factory Script example block diagram

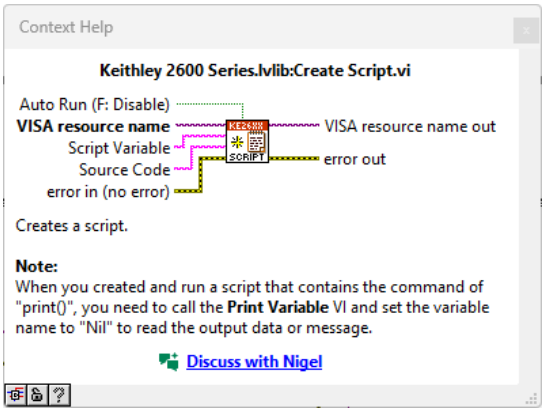

Figura 23: create script block

Figure 23 and 24; this is the core block. The most important. It sends and executes given commands in its input.

Internally uses NI-VISA blocks.

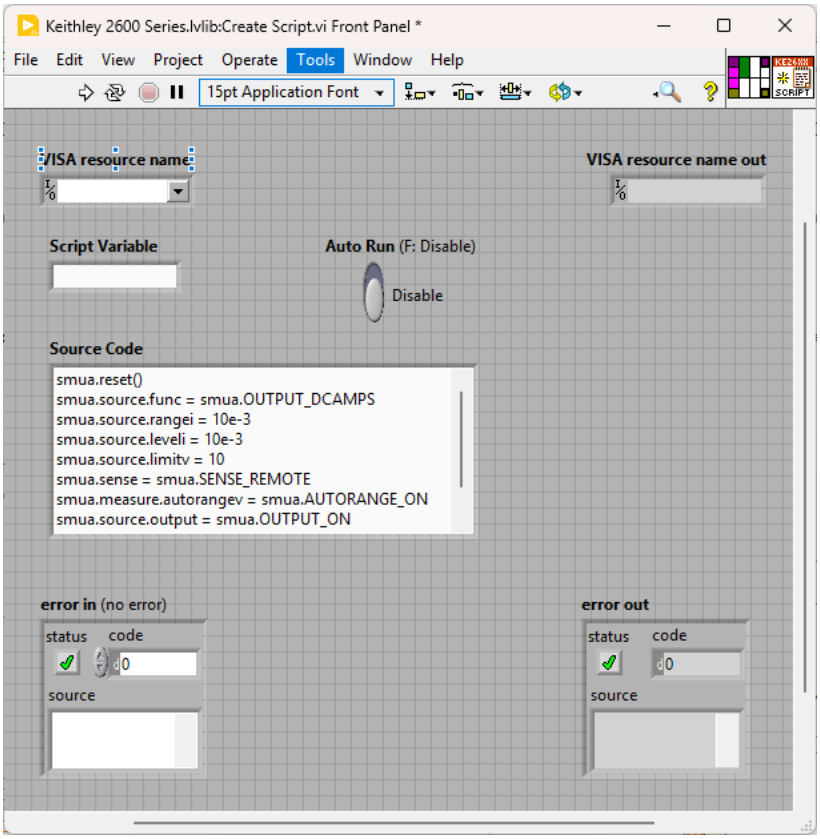

Figura 24: create script block front panel

Capturing output with oscilloscope (Figure 26 and 27), is a pulsed stair, but we want a DC sweep stair, so we must set end pulse action command from IDLE to HOLD: the output is maintained. Remembering in this test (SMUA is constant).

SMUA —> GATE —> channel A —> Blue

SMUB —> DRAIN —> channel B —> Red

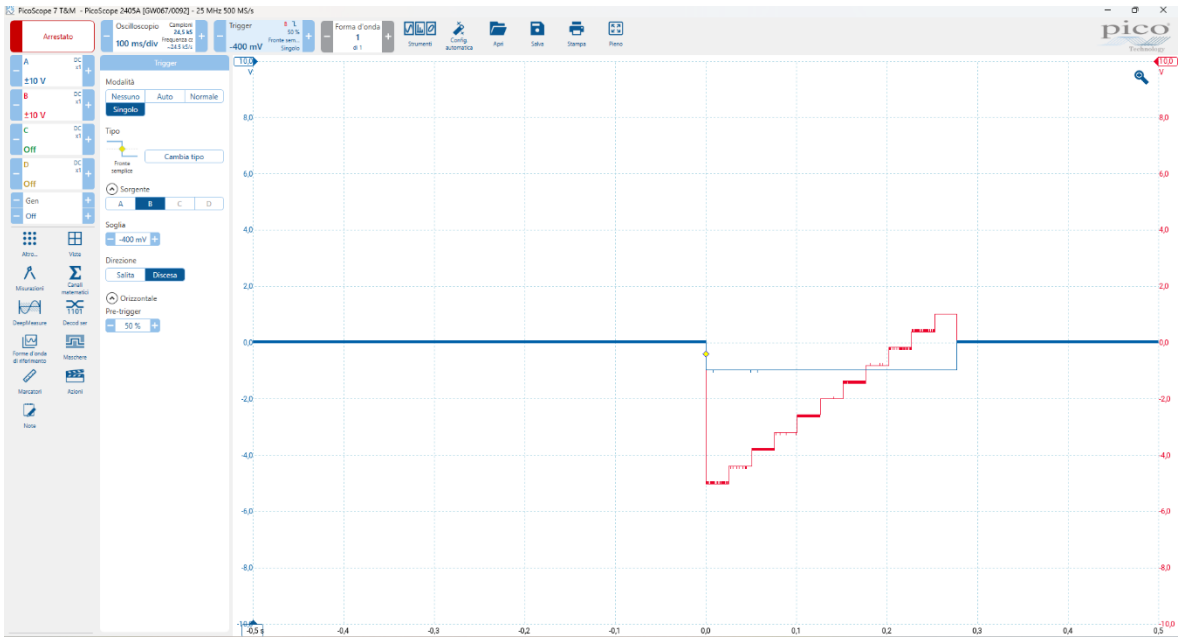

Figura 25: LabVIEW sweep dc

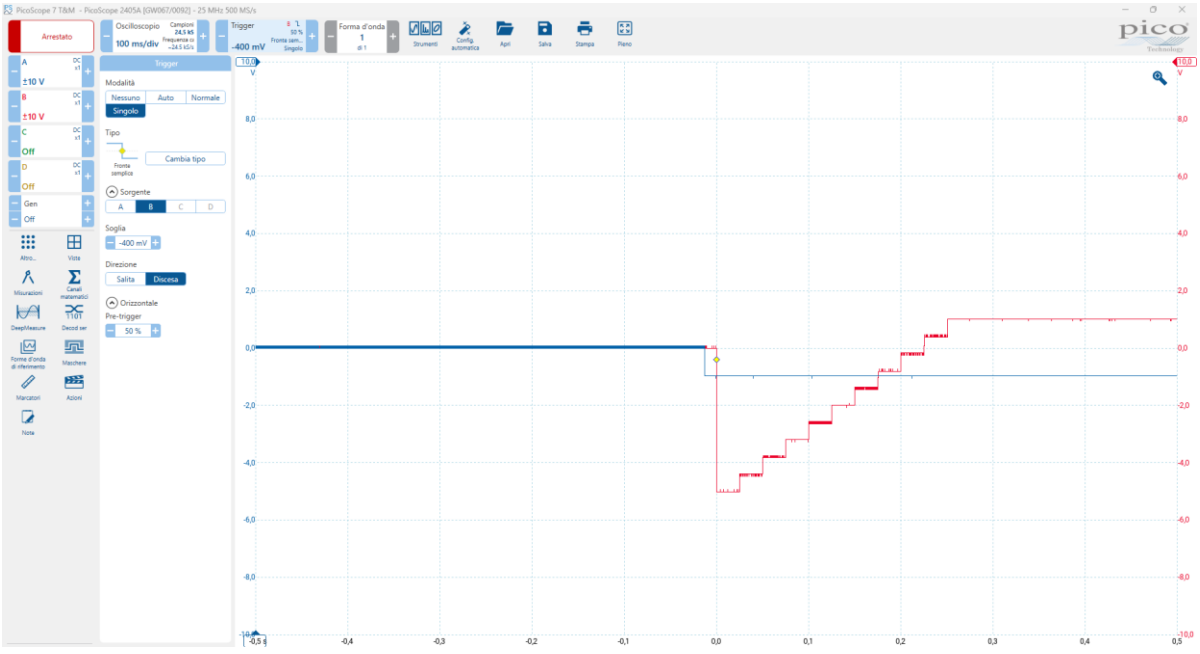

Figura 26: Kickstart sweep dc

Big important difference between Kickstart sweep and LabVIEW sweep is delay between SMUA and SMUB rising and descending edges. It is needed to avoid going over internal voltage threshold and breaking the device. If the device is not polarized internally in some areas it can go in breakdown and damage itself. This problem is resolved with event blender, explained in the next paragraph.

Next step was creating the control interface to set source voltages (for this experiment we stay on simple things and so controlling only voltage levels).

I need to recreate the instrument vi control to tidy and clean it: I deleted configuration and action elementary blocks and substituted it with script-execution block, but I kept buffer reading blocks.

This way is simple because I can create every script and configure its parameters with LabVIEW interface controls string, Boolean and numeric. The parameter in the instrument script are values that passed to script variables with LabVIEW “string build block”:

The concept is simple: I have a starting script and then I insert values in it and script-execution block send it to LabVIEW; finished execution the Keithley reads buffer reading instructions from LabVIEW blocks and then gives the measures result.

This measurement action is realized and tested in a standalone vi composed from opening and closing instrument communication blocks, a while loop to stay in idle state without reinitializing communication every time and a case structure to start measurement when user wants.

For this sweeping test controls are:

- Communication controls: setup and initializing measure phase
- Measure controls:
    - Points number, nplc (measure windows constant), source-measure delay
    - Voltage level: numeric constant con SMUA and interval start and stop on SMUB

As seen before, recording with oscilloscope, the output is a perfect stair on SMUB and SMUA is a fixed voltage.

Next step is making sweep stepping SMUA.

5 First measurement script: Sweep-Step

After a working sweep on SMUB, to have a complete sweep operation we need stepping voltage on the other SMU.

Stepping on a SMU is the voltage changing when the other SMU finishes a sweeping stair.

To make stepping SMU we need two important things

1) Measure action is independent from turning on and turning off
2) Relay race structure between SMUs. SMUA voltage step-up after a complete stair on SMUB (sweep action) until max number of sweeps.
3) Delayed SMUs sourcing between them to don't kill the device under test

To make independent capturing we need set SMUA trigger measure Asynchronous with SMUA.ASYNC: in this mode we turn on SMU, we do all measures we want and then turn off instead "on, measure, off" cycle. We need this because we want to have a constant level during the sweeping and capture values on SMUA at the same time of capturing SMUB that is instead in synchronous measure.

To realize this relay-race model we need to connect sweep complete event-id to gate SMU trigger source stimulus, to step up gate SMU after completing the sweep stair.

In words it is simple because we only need to set stimulus, but problem is cycle starting when SMUB hasn't started yet. In this case we must use event blender:

Event blender as mentioned before is a logic gate function for SMUs trigger event ids (interrupts). It can be configured as a logic "or" or logic "and". We need it in or mode to start gate SMU at beginning of the sweep-step cycle when drain/SMUB is off and step up gate between SMUB/drain sweep and its next.

If we source signal on SMUs together, device under test, depending by values, could be broken because if we source voltages or current when device is off (gate hasn't reached value), these can exceed internal threshold values in the semiconductor and then break it.

So, when need a little delay between SMUs sourcing. On Keithley SMUs sourcing and turn on are different: we can turn on SMUs without source the voltage or current signal

So, we introduce a 50 ms delay between SMUA and SMUB start commands.

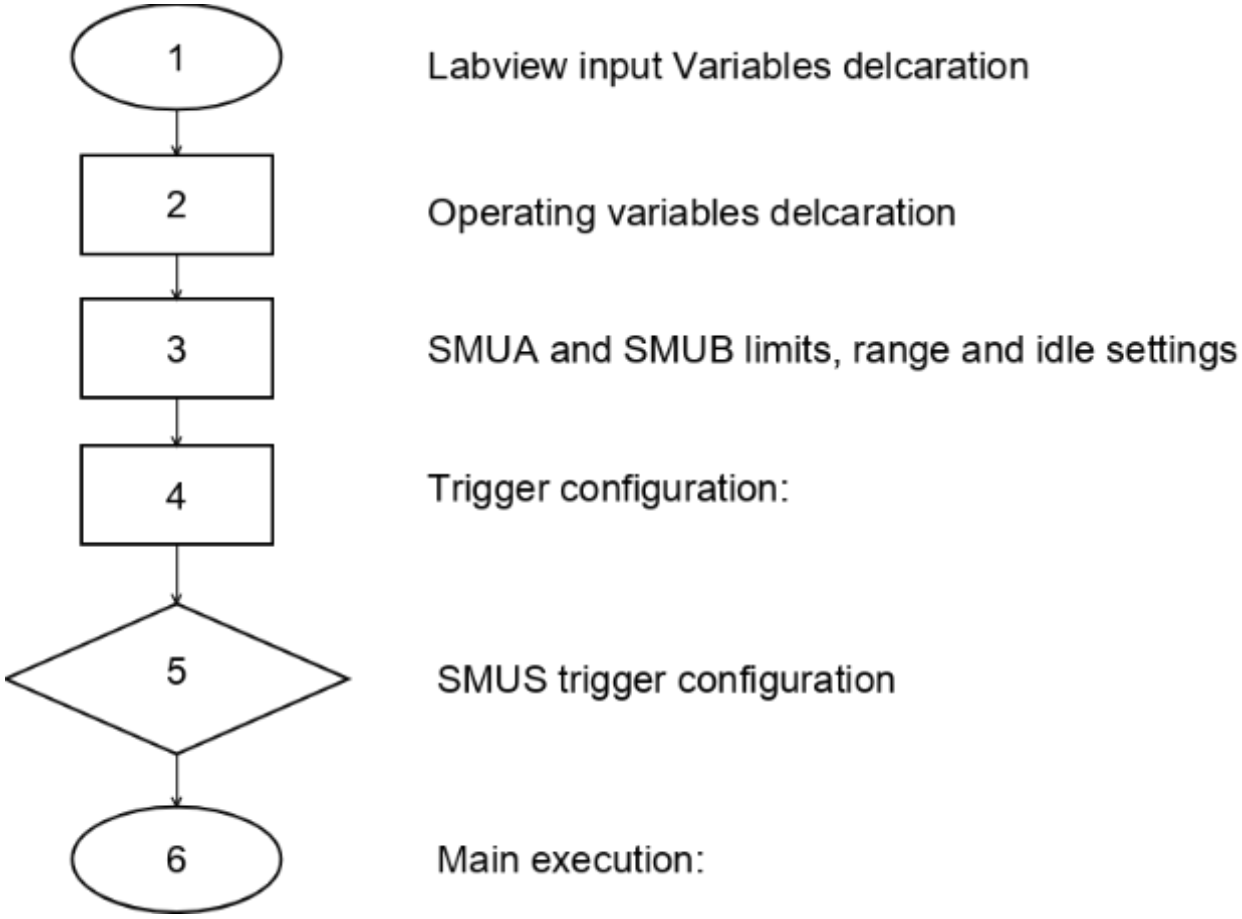

Figura 27: Single sweep-step flow chart

Code parts:

1) **LabVIEW input Variables declaration**. Since this script is the format string of LabVIEW string build, percentage statement is being substituted by LabVIEW input value at the LabVIEW block output, and the results is a complete script string at the entrance of the Keithley create script block. Symbol %g represents double/float, %d decimal and %s string. <varname>={%s} is step list array.

   Parenthesis are needed by Lua instrument interpreter to declare the array. In Lua an array is: array_1= {0,2,3,4,10, -1}.

   And this could be the resulting array of the script after LabVIEW string block values substitution.

   In inputs parameter we have measure points number, nplc, source-measure delay, source levels, ranges, limits and auto-operations flags.

2) **Operating variables declaration**. Delays declaration, timer names and event blender

Measure time: capture/sampling duration: in this interval the instrument does continuous sampling. At the end it does a medium value integral.

Timer Pulse width: Source-measure delay + measure time. Even if we are talking about pulse, the pulse generation for Pulsed Sweep and Sweep DC, in this case, is the same. Of course, as explained before; to make DC Sweeeping we need end pulse action set to HOLD.

Timer Period time: length of the sweep step: Pulse width + 1% * Pulsewidth

The last term is off time to let 2612B correctly detect event_ids (interrupt) rising edges

3) **SMUA and SMUB limits, range and idle settings**:
   - Idle: conditions at SMU rest, no triggering. We need to use non trigger actions to set no output/idle condition, source = 0 V, source limit =0,01 the lowest possible.
   - Measure Autorange and range values: autorange is automatic ADC input scale detecting, take some time and range value is the minimum detect starting value for autorange and it will become max instead for the manual range case
   - Compliance Limit: source limited non controlled variable. I source voltage, non-controlled variable is current.
   - Source Autorange and range: this instead the resoultion for DAC SMU source variable output. Auto is automatic and the value is the minimum detecting point. Auto mode takes some time it's the same of measure range. It can be manual.
   - Zero: gnd reference.

     Auto is always calculated at every measure operation. Takes some time

     Once: Only at value expiration date

     No. never

   - Nplc: constant for defining hardware real capture time. Above it was defined to make timer wait necessary time. Base integration/capture time is $\frac{1}{Line\ frequency\ (50\ Hz)}$

     And it can be adjusted: base capturing Time * nplc = real capture time

4) **Trigger configuration**: timer settings. Every timer has
   - Count: how many pulses generate at expiration
   - Delay: time to wait before pulse
   - Passthrough: pulse at the start or pulse at expiring
   - Stimulus: Trigger condition

     In sweeping we have 3 timer:

     1) Pulse period (for Pulse sweeping and DC sweeping is the same procedure)
     2) Source-measure delay (time before capture)
     3) Pulse Width end pulse, SMU power off. (ending of sourced value on SMU)

5) **SMUS trigger configuration:** configuration of trigger actions:

SMU sourcing type: voltage/current, interval (start and stop) or list.

Compliance limit of non-controlled source: I control voltage, the current is dependent

Measure type: current/voltage etc., Synchronous/Asynchronous or disabled, buffers

Event blender configuration for the necessary SMU.

Sweep Number: External loop cycles: number of stairs

Sweep points: Internal loop cycles: number of stair steps

End pulse action: Hold maintains constant level

End sweep action: Hold maintains constant level

Stimulus:

- Trigger Source: Sourcing the output
- Trigger Measure: starting measure/capture action
- Trigger Endpulse: Powering off SMU output

SMUA: this is the stepping SMU:

Output updates after every sweep: so, the source trigger is SMUB armed event id after SMUB activation. Since we want SMUA activation before SMUB activation we need to “invert”. We use event blenders that enable SMUA at beginning of first cycle and enables SMUA in the next cycles after SMUB. Event blender is configured as an “OR” logic gate between SMUA armed event id and SMUB armed event id

- Source Voltage list
- Async measure: Power on at sweep begininng, measures and updating at new sweep
- Event blender in or fucntion
- Trigger arm count 1
- Trigger count: sweep number

Its steps have a sweep duration, so every SMUA step corresponds to every SMUB sweep duration.

For example: 4 sweeps of 11 points, SMUA trigger arm count is 1, SMUA trigger count steps are 4, while SMUB trigger count steps are 11 and SMUB trigger arm count is 4.

SMUB: normal sweep SMU

It’s simply a stair. The output updates at every step. Source voltage has start and ending values, with number of points to autodefine voltage step.

Output updates after every step. Every step is called trigger in its “trigger structure slang”.

- Source voltage interval (start,end , num points)
- Sync measure: Power on at step begininng, ONLY ONE measure and updating at new step

- No event blender requested
- Trigger arm count: num sweep
- Trigger count: trigger/points number

6) **Main execution**:

Power on SMUs output.

Commands to start SMUS trigger structure... Between SMUA and SMUB is introduced a delay of 50 ms to avoid device under test breaking, for threshold breaking over voltages.

smuX.trigger.initiate().

Waitcomplete() command to let finish readings and other operations

Power OFF SMUs with 50 ms delay for the same reason

User interface input parameters are contained in Cluster blocks.

Cluster blocks are structured data type used to contain numbers, strings, arrays, etc.....

Figure 28, cluster decomposition to retrieve string input wires.

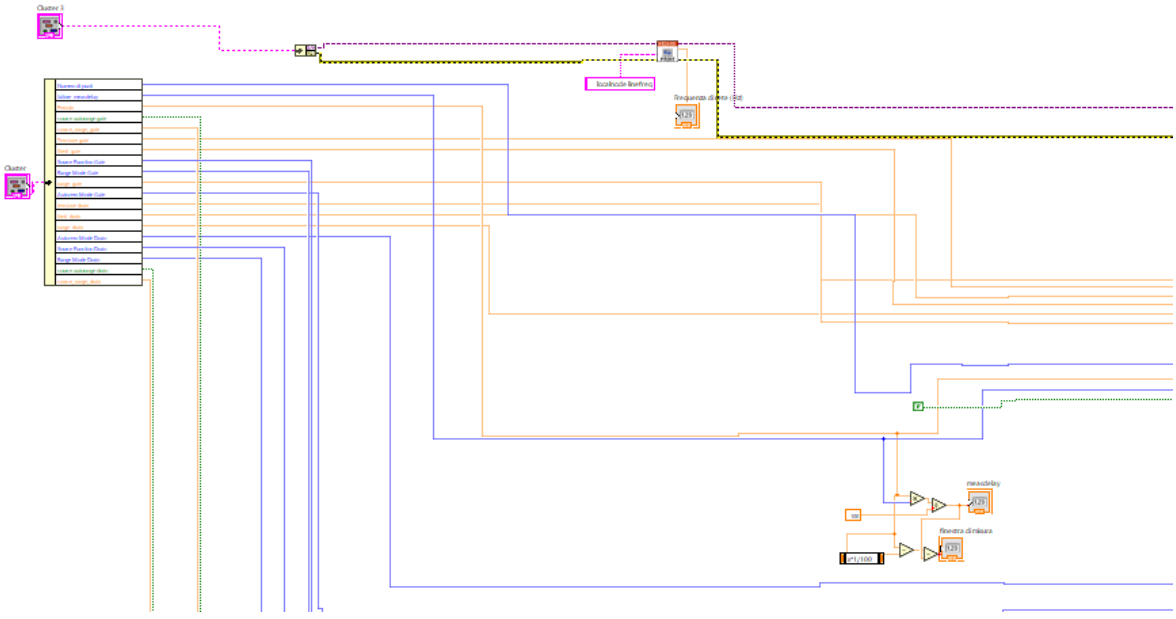

Figura 28: Labview cluster input decomposition

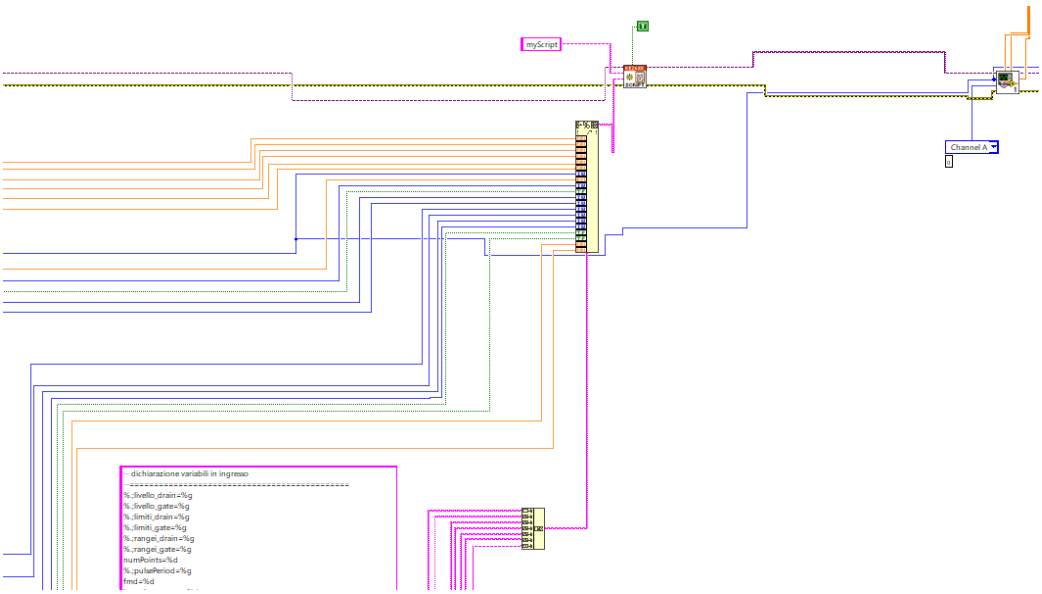

Figura 29: Format into String block and Keithley Script block, first version

Figure 29. Format into string block takes input and substitute values in formatted string structure. Then, built string is passed to LabVIEW Keithley Script block.

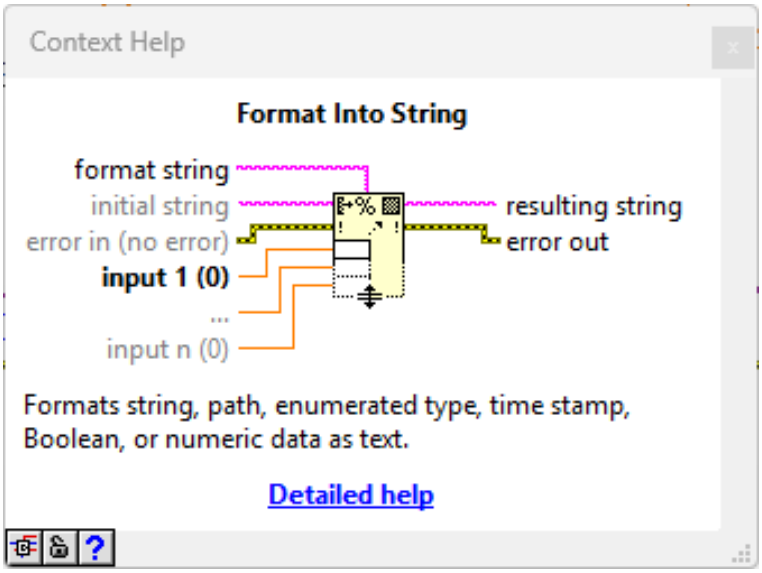

Figura 30: Format into string block

Figure 30. This is the second important block. A LabVIEW built-in block. In format string you put a default string with percentage parameter that defines the substitutable parameter corresponding to a character, integer or double coming from input. Percentage parameters must equal number to input number.

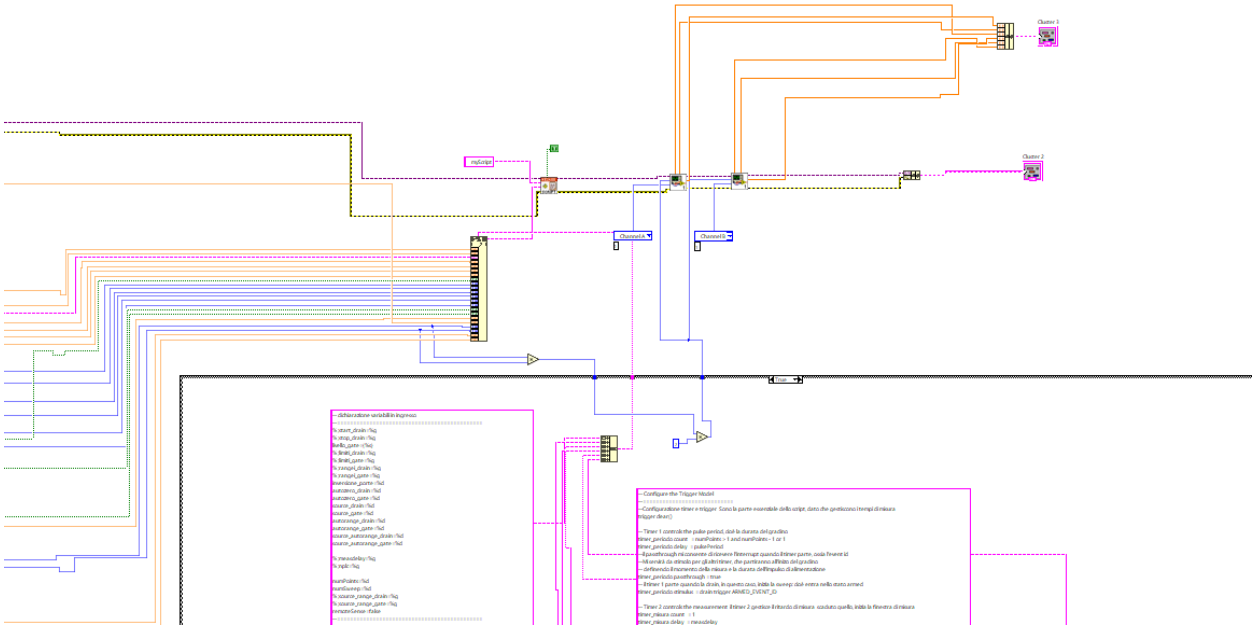

Figura 31: Format into String block and Keithley Script block, definitive version

Figure 31. In definitive sweep dc vi version, dual sweep (explained next) is being added as sweep dc variant so there is a switch case to select mode.

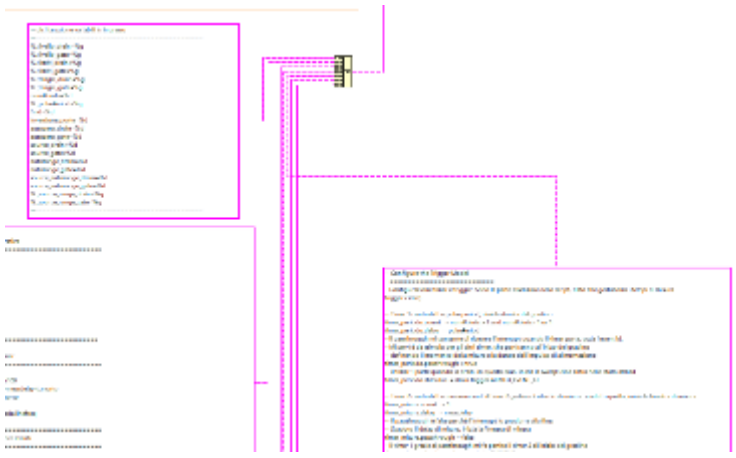

Figura 32: Strings merger block

Figure 32. Basic Script structure is divided into subscripts to simplify reading. It is built by string merger

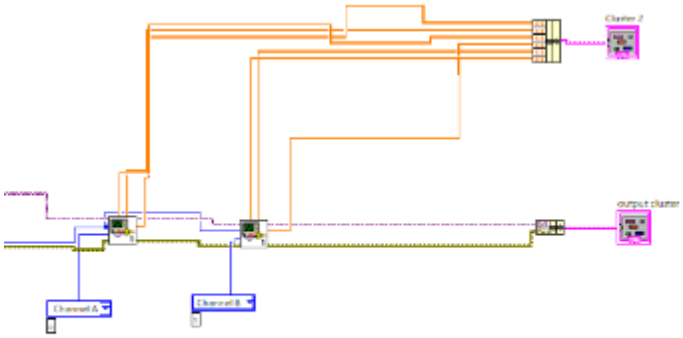

Figura 34: Instrument personalized buffer reading subvi blocks

Figure 33. These subvis are created to simplify measure output circuit. It terminates in output cluster. Again, for simplicity, inputs are only channel selection and end index. Buffers selection is constantly defined.

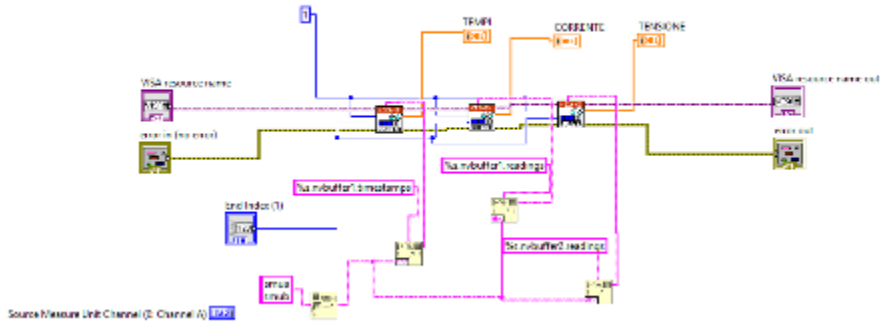

Figura 33: Instrument personalized buffer reading subvi blocks decomposed diagram

Figure 34. In measure output subvi there Keithley low level buffer readings. These allow me to select the interested Lua table through buffer commands such ".timestamps" and ".readings"

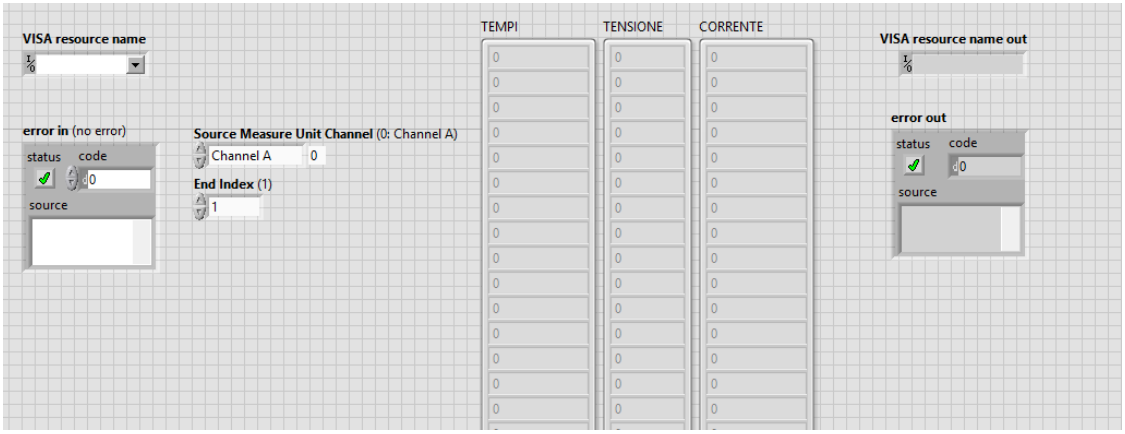

Figura 35: Instrument personalized buffer reading subvi front panel

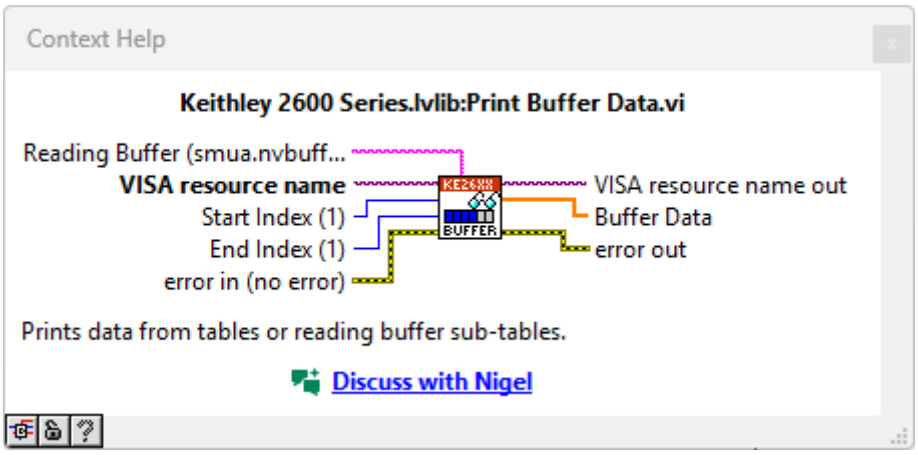

Figura 36: Print buffer blocks

Figure 36. Keithley's driver low level read buffer. It uses VISA blocks

Event ID - stimulus chronological table explanation

Element	Input stimulus	Output event id	Condition
Smua	Smua trigger.initiate	Smua armed event id	
SMUA Event blender	Smua armed event id	Smua event blender event id	
Smua trigger source	Smua event blender event id		
Smub	Smub trigger.initiate	Smub armed event id	
Timer 1	Smub armed event id	Timer 1 event id (pulse at starting)	
Smub trigger source	Timer 1 event id (pulse at starting)		
Timer 2	Timer 1 event id (pulse at starting)	Timer 2 event id	
Timer 3	Timer 1 event id (pulse at starting)	Timer 3 event id	
Smub trigger measure	Timer 2 event id		
Smua trigger measure	Timer 2 event id		
Smub trigger endpulse	Timer 3 event id		
Smub		Sweep complete event id	
Smua trigger endpulse	Sweep complete event id		Counter reached the end? No —> execute
Smub		Smub armed event id	Counter reached the end? No —> execute

SMUA Event blender	Smub armed event id	Smua event blender event id	
Smua trigger source	Smua event blender event id		
Timer 1	Smub armed event id	Timer 1 event id (pulse at starting)	
....repeating until the end			

Figure 37. Example of sweep step: 4 sweeps and 11 steps. Blue line is channel A (gate) and Red one is channel B (drain). As written above, there is 50 mS distance between lines at power On and power off to avoid device breaking. As we can see, blue line is stepping and red is sweeping.

Signals captured with Pico Oscilloscope.

Figura 37: sweep-step

Figures 38 and 39. Definitive user panel of sweep (definitive sweep shares control with dual sweep, next explained). Values are default.

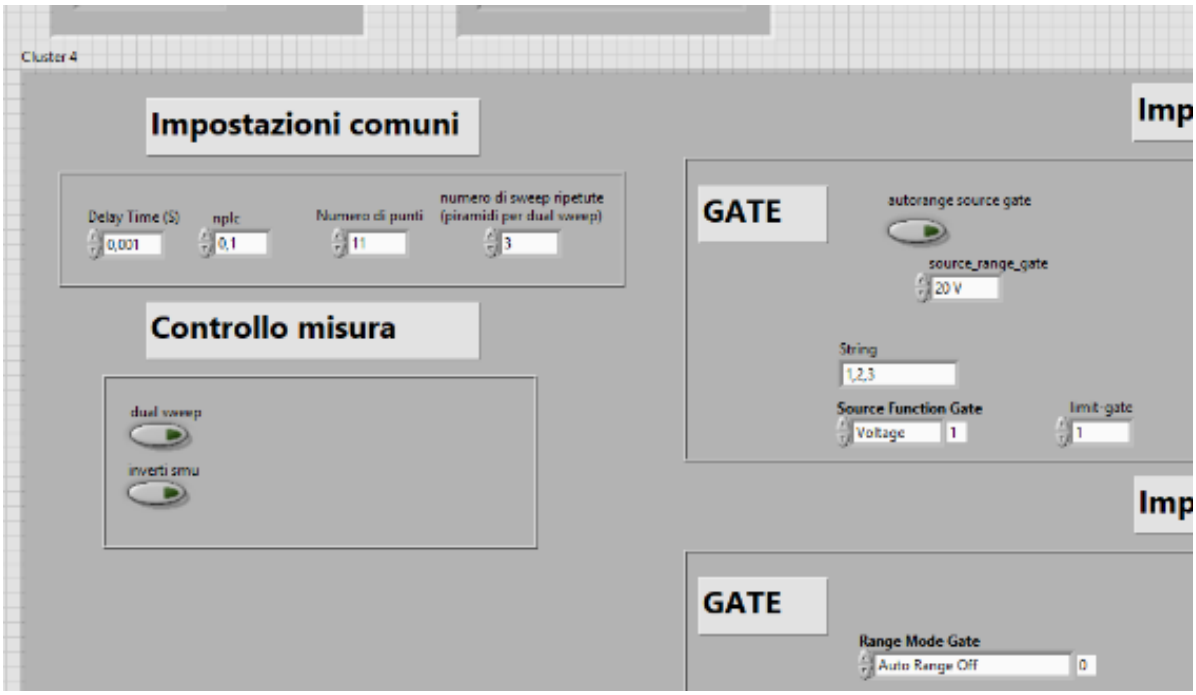

Figura 38: Definitive user panel of sweep (definitive sweep shares control with dual sweep, next explained).

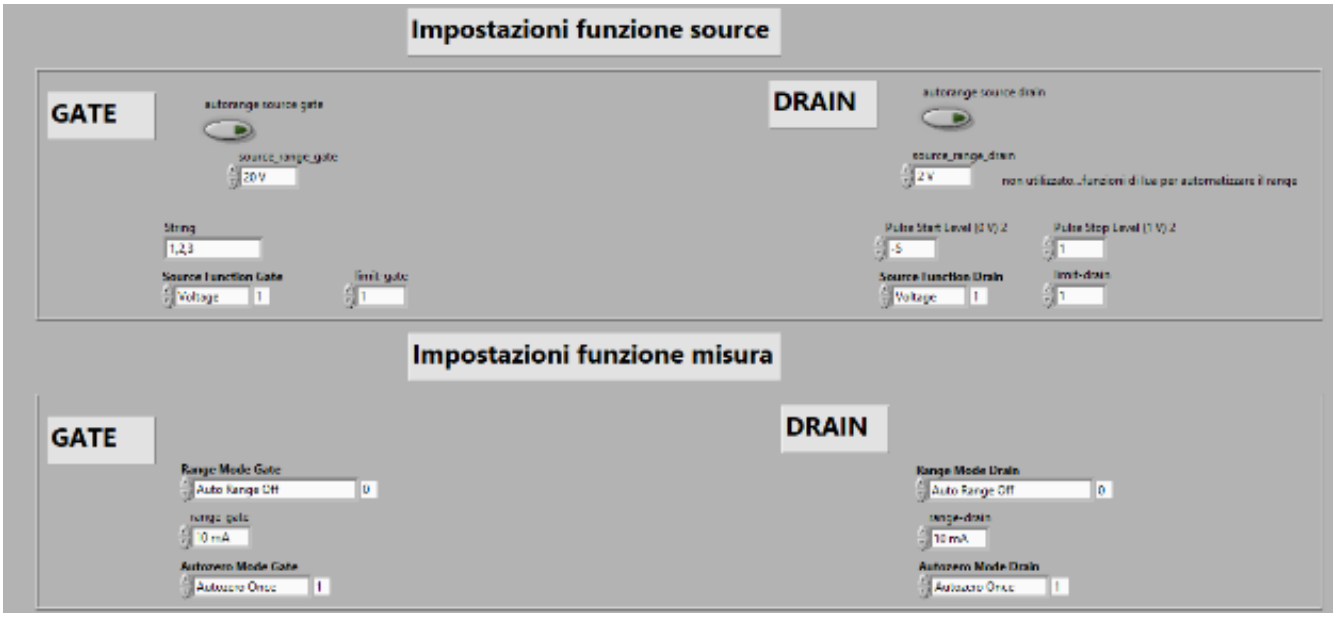

Figura 39: SMUs front panel parameters of sweep-step

6 Sweep-step based scripts: Dual Sweep Step and Bias

Once completed sweep-step it is time to build dual sweep: the same as sweeping step, but with stair up and down. The only way to realize this, is glueing an up sweep to a down sweep, result obtaining a “Pyramid”. For this mode, we cannot use sweep trigger structure, because we can only execute an up sweep or a down sweep differently.

So, we need first to configure the up sweep and gate SMU, we execute only one cycle then reconfigure sweep for down stair and gate SMU like the up phase and execute one time: at the end repeat with for loop.

Up stair:

- Configuration of gate SMU as a single constant value instead of an array. Array only works for only up or only down sweep.
- Configure drain up sweep like single sweep script.

Down stair:

- Configuration of gate SMU same for up stair with same source level.
- Drain SMU’s start and stop values are inverted to descending.

I need to reconfigure gate SMU even if stays at the same level because trigger structure is a unique block by I have understood: if drain sweep sequence ends, gate sequence must ends too and so I couldn’t maintain alive gate level if drain cycle ends: it’s impossible so I need to reconfigure gate every time. There is an infinitesimal down spike where up sweep ends and down sweep begin. It could be seen if we set end-pulse action like idle.

The for loop is used because the step and sweep repetition can’t be made with trigger structure.

To avoid overwriting in reading buffers I enabled buffer appending options

In trigger structure if we want to execute a step sweep, we will need to insert a source value list at stepping source. In dual sweep case it is different. Using a for loop we must insert only a value in the SMU source command that is picked from the stepping SMU user array list at every for-loop index update.

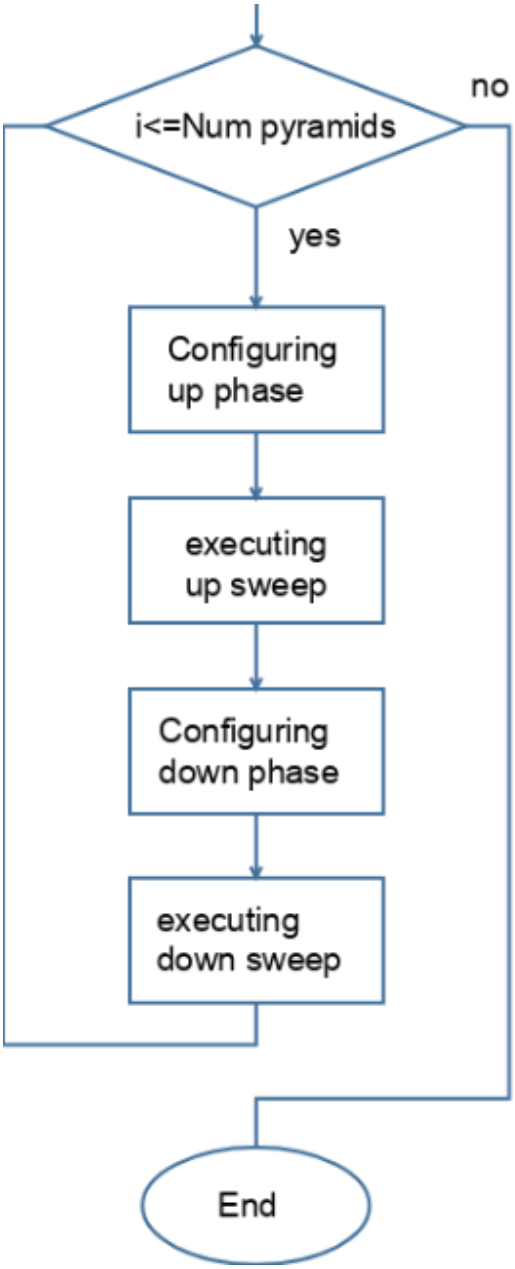

Figura 40: Dual Sweep Step SMUs trigger configuration

Differences between single sweep and dual sweep

1) 2 different single sweep step glued together: up and down
2) Two times sweep configurations, for up and down
3) For loop for repeating. We cannot use trigger structure
4) Since we cannot use trigger structure, we have two problems:
    a. We cannot insert the complete array/list in the stepping SMU source command; we need to cycle array elements one by one.
    b. If the stepping SMU array has less points of pyramid repetition number, theoretically (at least for the single sweep) these are automatically repeated. In this case we must rebuild it to adapt to pyramid numbers, taking the initial part of the array and concatenating it to initial array. With Lua table managing commands we get array/table element number and

concatenate/repeat elements to cover the difference. Then we will get an equal length array to be cycled at every step like the single sweep one

```
--dichiarazione variabili operative
--=============================================
remoteSense=false
if inversione_porte==1 then
gate=smub
drain=smua
else
gate=smua
drain=smub
end

dim_livello=table.getn(livello_gate)

-- se il numero di livelli è minore del numero di piramidi, allora ripeto i valori
if (dim_livello<numSweep) then

   for i=1, (numSweep-dim_livello) do
      table.insert(livello_gate, livello_gate[i])
   end

end

--=============================================

-- dichiarazione tempi dei timer
--=============================================
-- mi calcolo la finestra di misura
finestra_misura=(.....
```

The case is requested to switch from dual and single sweep. LabVIEW block's structure is nearly identical... Some blocks are changed and dual sweep "occupies" double elements in memory than single sweep: a dual sweep of eleven points has twenty-two points in total, eleven up and eleven down. So, when indexing buffer reading, we need a multiplier block of x2. Figures 41 and 42.

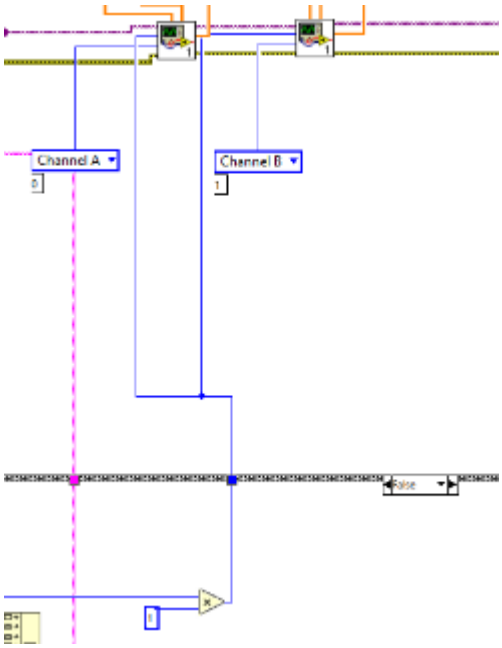

Figura 41: single sweep step indexing

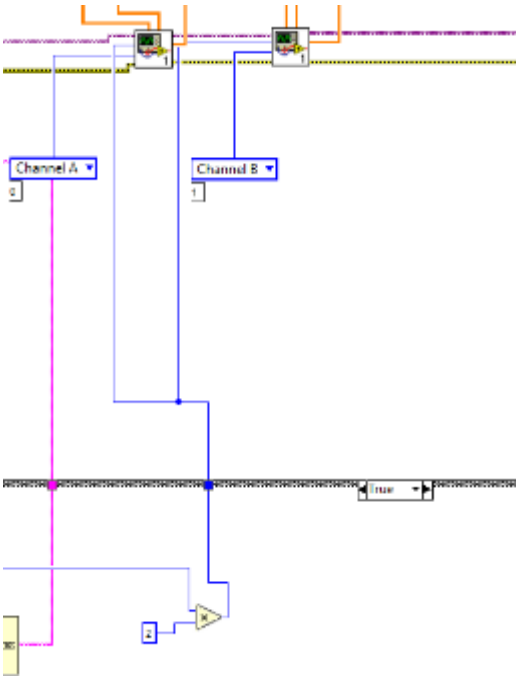

Figura 42: double sweep step indexing

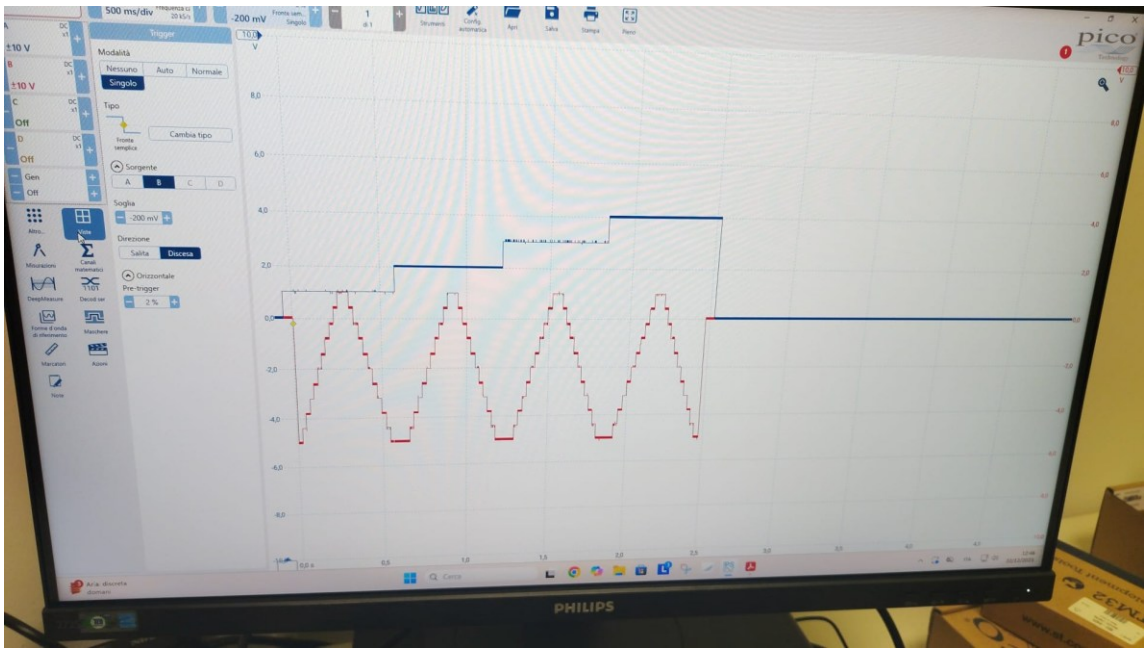

Figura 43: dual sweep step

Figure 43. Dual sweep is identical to single sweep-step except pyramids.

Remembering sweep user panel that shares some commands with dual sweep...

Since these types are in same vi, there are the commands to change mode.

Pressing dual sweep button, it enables this mode making true Boolean case structure and sweep repetition become pyramid repetition. Figure 44. Values are default.

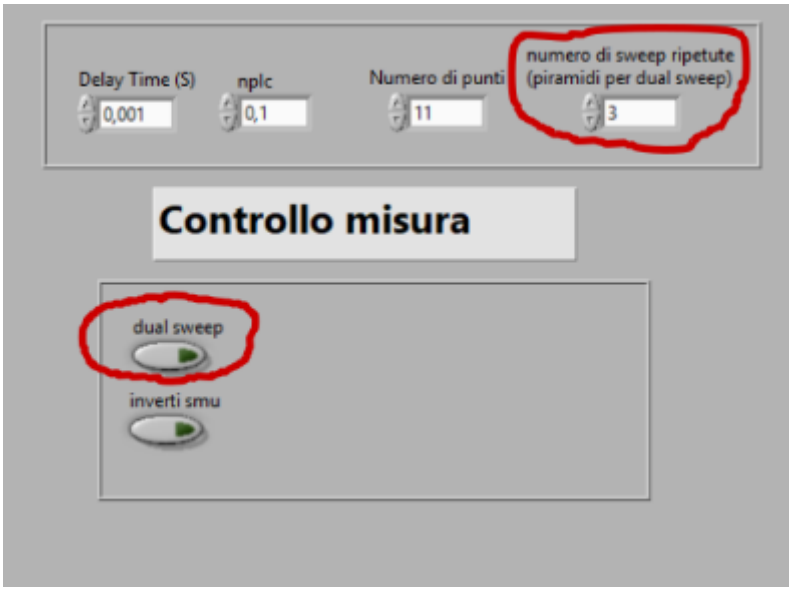

Figura 44: dual sweep step front panel controls

The last measure I need to prepare is Bias measurement: this consists in high-speed burst measures at constant voltage in both SMUs.

To keep it simple I used same structure for trigger sweep sourcing. It is a list of a single value, to obtain a fixed source level. The control is a numeric variable, for both SMUs

In bias measurement I changed parameter controls from sweep step vi. Now they are:

- Step Period time: a fixed list of values: from 500 uS to 10 S with scales x1, x2, x5 the same for next decades. 500 uS is minimum. Over that time, the instrument can't detect correctly event_ids interrupts. This time from a measure/sampling and its next
- Source-Measurement delay factor in percentage: minimum 60% to 70% of total period length. Measure delay is not an indipendent parameter anymore.
- The remaining part is for window capture integration and dead time

In Bias measure, the main difference is step duration controls

```
-- dichiarazione tempi dei timer
--================================================
t_morto=pulsePeriod*1/100
measdelay=pulsePeriod*fmd/100
finestra_misura=pulsePeriod-measdelay-t_morto
pulseWidth=pulsePeriod-t_morto

nplc=finestra_misura*localnode.linefreq

--================================================
-- dichiarazione timer e gestore eventi
--================================================
-- Timer 1: timer periodo (larghezza gradino)
timer_periodo=trigger.timer[1]
-- Timer 2: Timer ritardo misura
timer_misura=trigger.timer[2]
-- Timer 3: Pulse Width Timer
timer_larghezza_impulso=trigger.timer[3]
```

Figure 45. is an example of a bias measurement with channel B/drain end sweep action set on idle.

Drain is syncing measure and end pulse IDLE setting is to verify sample happens correctly since there isn't a debugger. In fact, SMU is powered on, does measure and then off. This is exactly what we want to have.

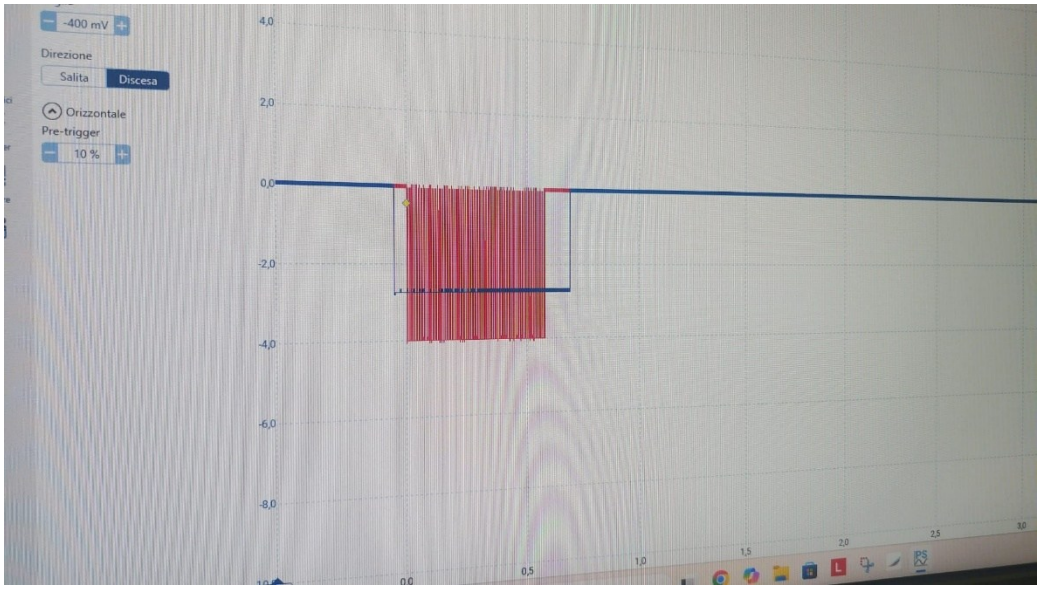

Figura 45: drain smu with end sweep action IDLE

Figure 46. Since Bias is conceptually constant, we want to end pulse action set to HOLD. So, it will be this

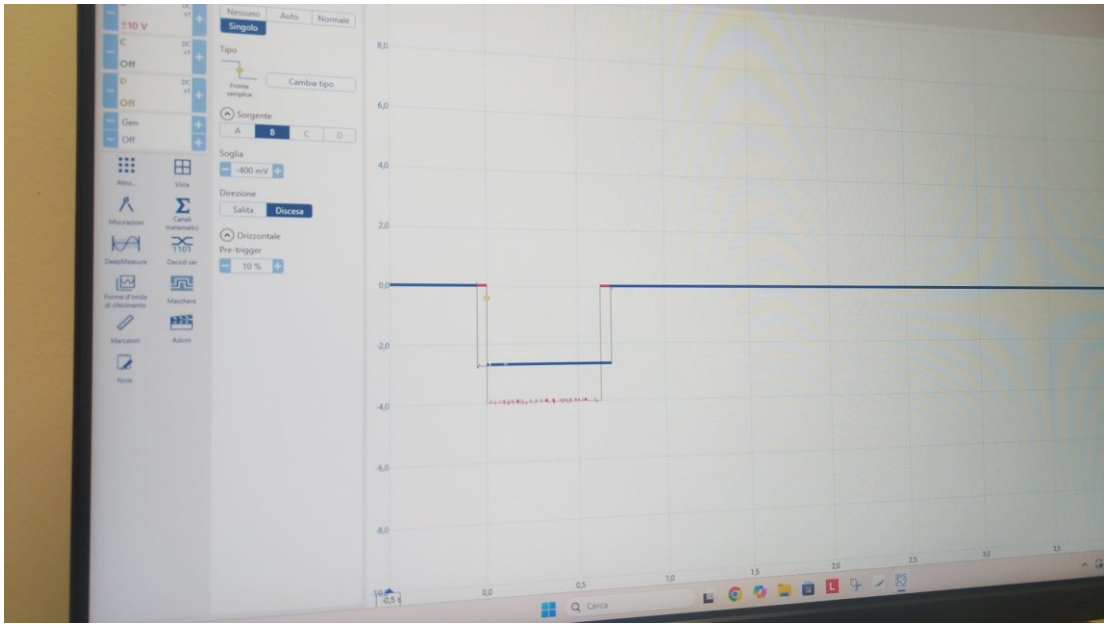

Figura 46: drain smu with end sweep action HOLD

Figure 47. Bias points settings: Period control is a window menu and as mentioned before it indicates time between a sample and its next. These are default values.

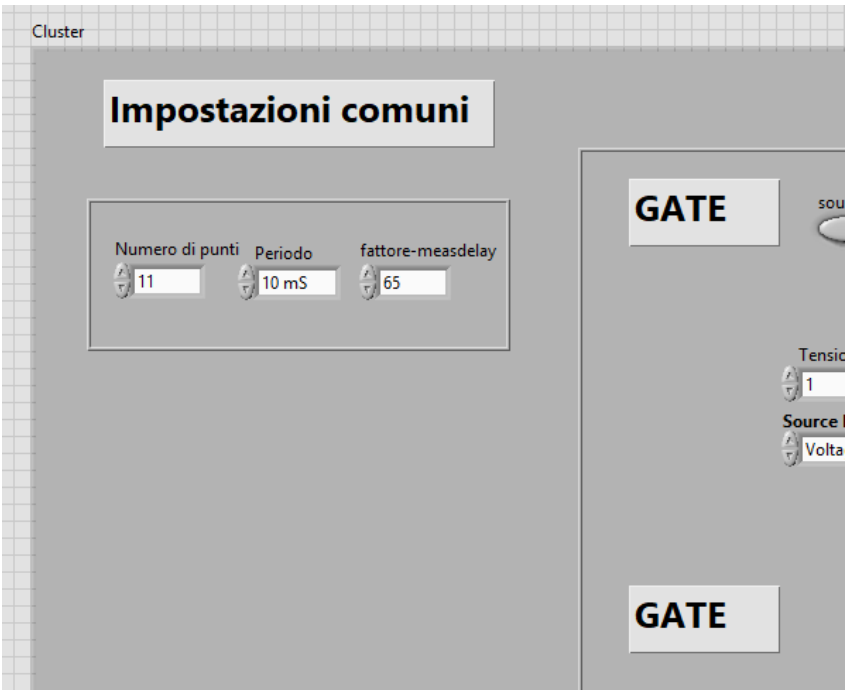

Figura 47: Bias points parameters front panel

Figure 48. As we can see in the picture SMUs source voltage levels are fixed at 1 Volt as default values like others.

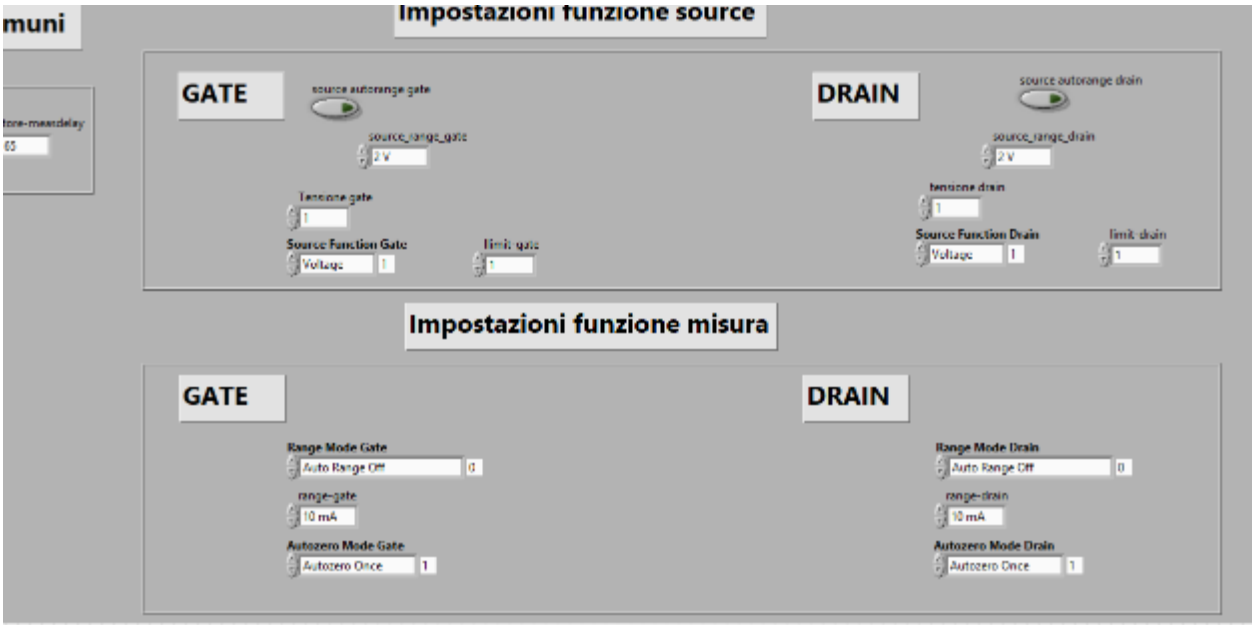

Figura 48: SMUs parameters front panel

7 Final improvements

7.1 Vi creation

We need to encapsulate Bias and single Sweep+Dual Sweep in Vis, to combine them later in more complex measurements like Stress-Test. It is being explained in the next paragraph.

LabVIEW's Vis are an analogy to structured text programming language functions.

Since every measurement has a lot of parameters, the smarter solution is to include all input wires in a LabVIEW cluster data block to keep the project tidy. In this we have only a wire. In case of editing only parameters we can extract and modify it.

Clusters blocks represent advanced structured data type, like a C class.

They can contain different data types: string, numbers, Boolean, arrays, etc...

Vis user interfaces are imported by next stress-test interface vi.

7.2 csv file creation subvi function

In stress-test, measurements are automatized and produce many values. To simplify visualization without having "thousands flying array visualizers" the best solution is automatic creation .csv files. They are created at every iteration if we have more measurement cycles.

To create csv files, I used LabVIEW spreadsheet blocks, like some examples on the web suggested to me. The main important step was opening a test .csv files with a text editor to know formatting. In Latin countries such Italy, France and Spain the separation between values is dot comma symbol instead simple comma, (csv stays for Comma Separated Value) so in the dedicated blocks separation operator is ";". Of course, to simplify program I created a subvi with encased spreadsheet creation operations. Figures 49 and 50.

Spreadsheet Creation:

- Input: time, voltage and current. they are output arrays column (a transverse array) from bias and sweep measures Vis. They glued side by side creating an array of columns (a simple matrix) with LabVIEW's matrix creation block
- Spreadsheet label constant: constant string set in vi that print name of column in the first csv file row. Measure values are appended down.
- Spreadsheet creation: LabVIEW spreadsheet block can read an array or a matrix. To avoid numerical approximation by MS-Office, ONLYOFFICE or LibreOffice is convenient to save numbers as string in spreadsheet cells.

  First, we need to pass labels array and then matrix values. We can do this concatenating 1 row matrix (labels array) with measure matrix values.

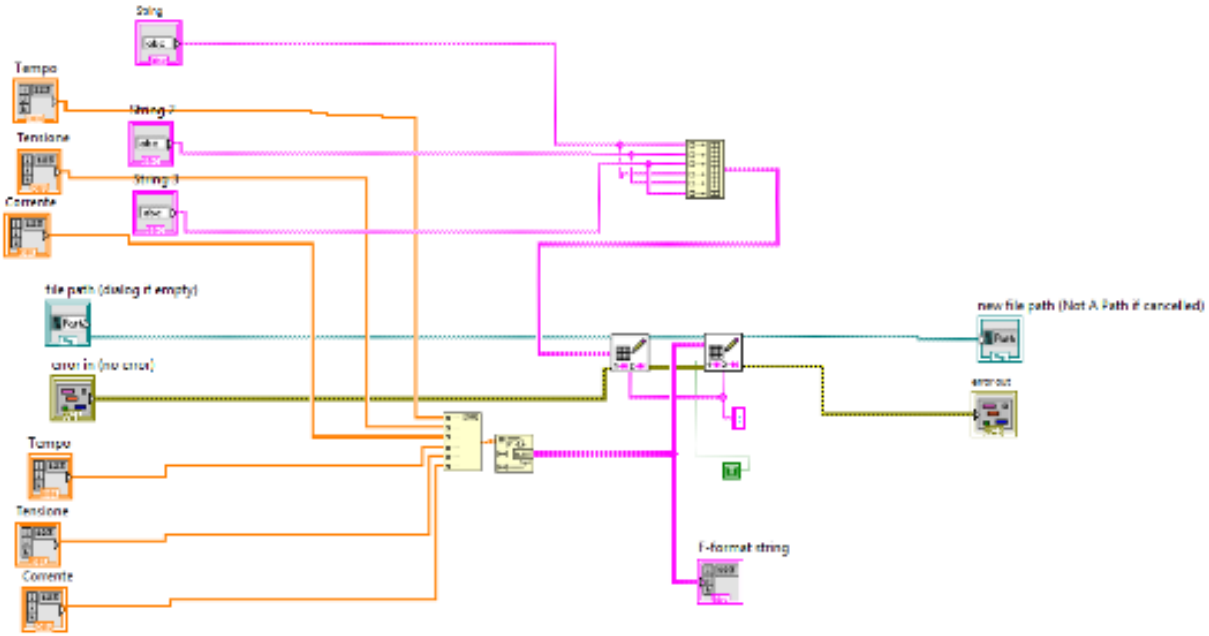

Figura 50: CSV file creation subvi block diagram

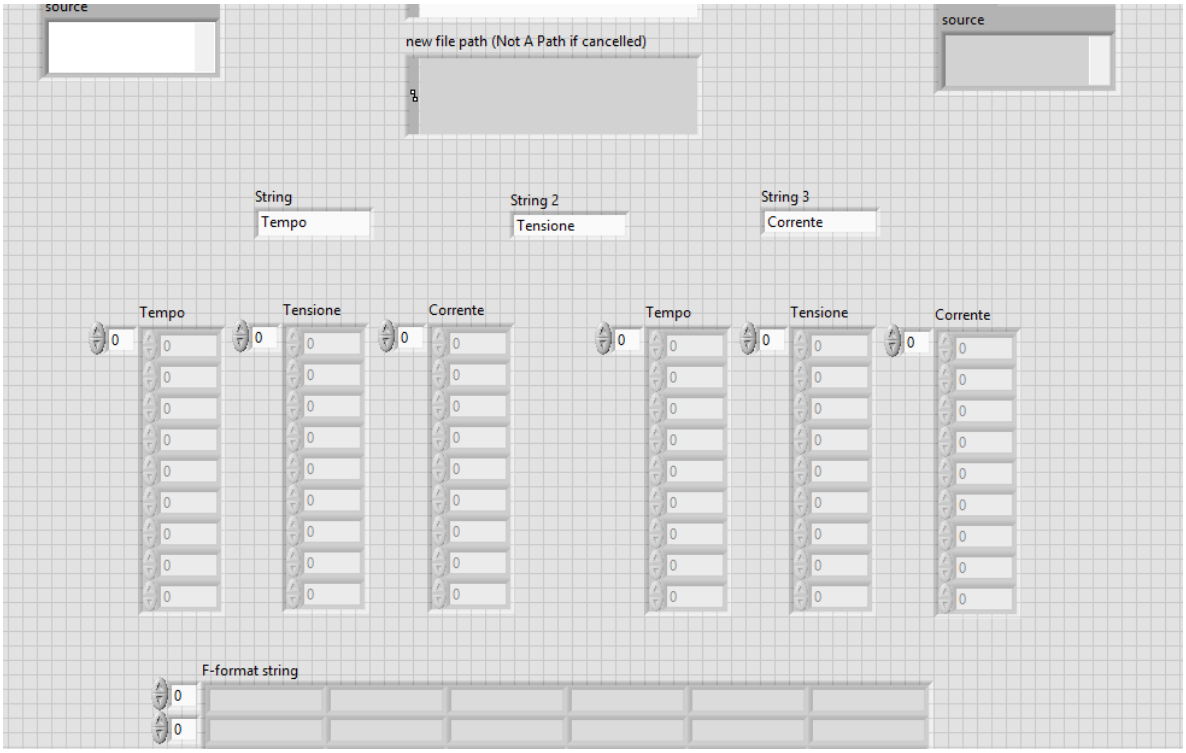

Figura 49: CSV file creation subvi front panel

8 Stress-test program

Stress-test is a program that takes to the limit device under test. It is the Goal of the project: build Vis to build this stress-test bench.

It consists of Test IDVD+Test IDVG+ Bias stress conditions looped for N times, in our case 10 and at the end final tests IDVD and IDVG are executed.

Test step is to know start conditions of the device under test before every stress part

Stress step is bias constant conditions to push device to its limit condition and know breaking condition and phenomena.

The stress-test cycle is Looped N times with a LabVIEW for loop. After loop we execute final test to know device characteristic changing.

Test IDVD:

Drain sweeping and gate stepping with drain and gate current compliance

Test IDVG:

Gate sweeping and drain stepping with current compliance

Stress:

Bias condition on both SMUs (gate and drain): constant voltage applied for a fixed number of points (trigger structure is the same of sweep-stepping, with event blender, event ids and stimulus, the difference is voltage list of a single value).

During the constant voltage, the instrument “shoot” a burst of voltage and current high-speed sampling. Every test refers to previous stress, for example, test 1 refers to normal condition (stress number 0), because no stress. Test 2 refers to stress 1 (first test-stress cycle).

The program is formed by a sequence structure:

1) First section: instrument connection configuration
2) Second section: Measure zone. It contains while loop that allows you to execute many times as you want the stress test without re-initializing instrument.

   Inside this loop there is case structure that contains for loop with vis inside.

   Case starts measure only when a pulse button (Boolean variable) is pressed

   For loop inside Boolean case contains IDVD test, IDVG test and bias stress. Out from for loop there are final tests.

3) Third section: instrument connection closing. There is “close connection” block

For loop repetition depends by a variable that is set in control panel. Inside the for loop we found:

- IDVD test vi, its csv creation file and auto naming blocks
- IDVG test vi, its csv creation file and auto naming blocks
- Bias stress vi, its csv creation file and auto naming blocks

Auto naming blocks insert the sequenced number at the end of the name of the .csv file after every measure.

Out the for loop there are final IDVD and IDVG tests to know characteristic changing after stress-test.

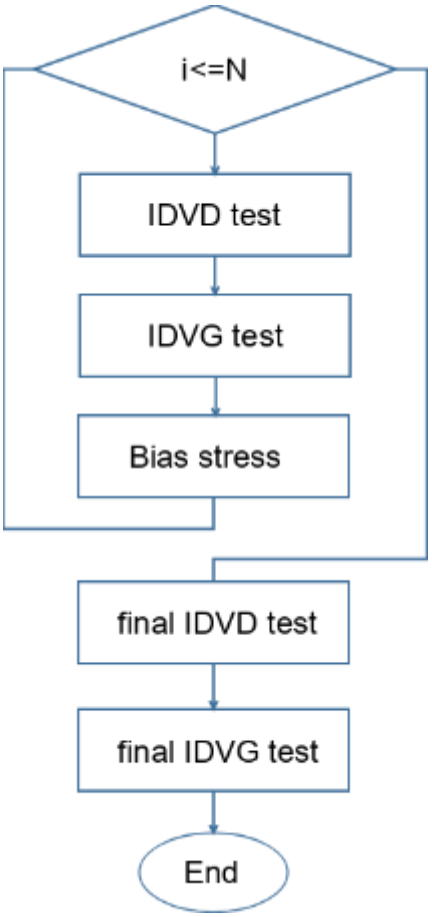

Figura 51: Stress test flowchart

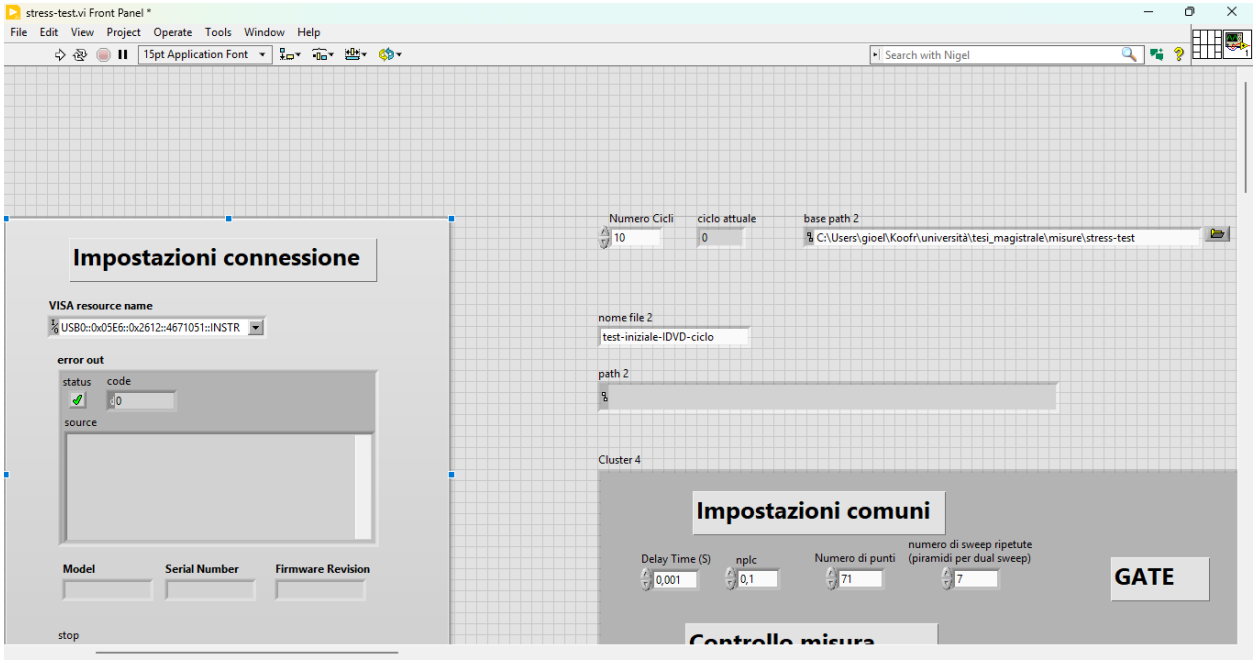

Figura 52: Stress-test front panel

Figure 52. It is stress-test front panel, it's too big for LabVIEW window.

We can see connection settings cycle number settings and test input clusters imported from vis.

Figure 53. There are IDVD, IDVG and Bias dedicated clusters that control the repeated subvi for chosen cycle number and at the end there are input dedicated clusters for final tests.

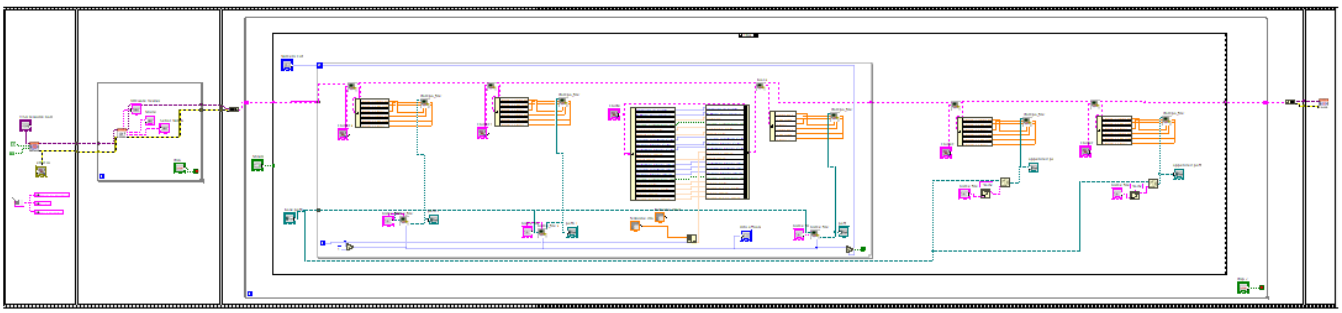

Figura 53: Stress-test block diagram

9 GaN HEMT short introduction and measure charts

A HEMT is a type of transistor that uses semiconductor materials with high electron mobility, allowing for high-speed switching (high-frequency operation). In fact, HEMT stays for High Electron Mobility Transistor. 5]

9.1 GaN HEMT Structure

Si MOSFETs utilize a vertical structure, whereas GaN HEMTs adopt a lateral structure. GaN is epitaxially grown on an Si substrate, after which an AlGaN (Aluminium Gallium Nitride) layer is formed.

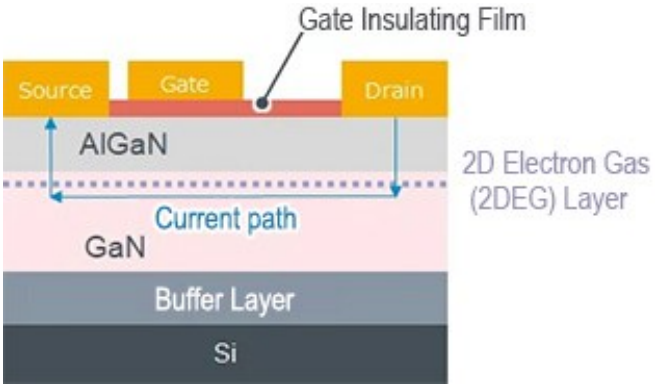

Figura 54: GaN HEMT layering

The growth of a thin AlGaN film on GaN results in a piezoelectric effect, causing electrons to gather at the interface.
This forms a high-mobility two-dimensional electron gas (2DEG) layer that serves as a path for current to flow.

Si substrates are preferred due to their larger wafer sizes that make them easier to mass produce compared to other materials.
At the same time, the large difference in thermal expansion coefficients between GaN and Si can cause substantial stress during cooling after crystal growth, potentially leading to cracks in the substrate.
A buffer layer is provided to relieve this stress. 5]

9.2 Switching Loss

GaN HEMTs dramatically reduce switching loss compared to silicon MOSFETs. This translates to significantly improved efficiency in power supply systems. 5]

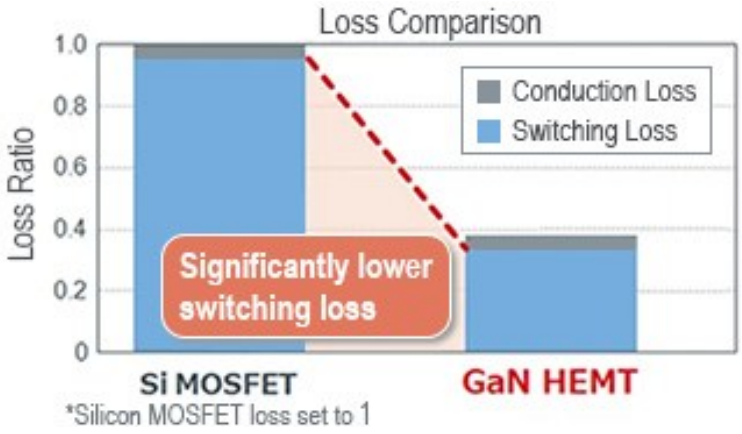

Figura 55: Si MOSFET GaN HEMT loss comparison

9.3 Measure charts of GaN HEMT

Below are represented first test and stress charts and final test charts of GaN HEMT.

As mentioned before, it consists of Test IDVD+Test IDVG+ Bias stress conditions looped for N times, in our case N=10 and at the end are executed final tests IDVD and IDVG.

First stress-test was done on a mosfet, at max 10 Volts to avoid GaN HEMT accidental breaking.

Gan HEMT stress starts from 10V to 100V. The 2612B requires safety interlock bypass with a diode.

After having taken all cycles measures, IDSS and Ron degradation charts are done.

Measure condition and boundaries:

IDVD	Voltage range	Num points/ Voltage steps	Compliance (Current limit)
Sweep drain	0V —> 7V	71 pts (100mV/step)	1 A
Step gate	-5V —>1V	7 step (1V/step)	1 mA

IDVG	Voltage range	Num points/ Voltage steps	Compliance (Current limit)
Sweep gate	-5V —> 1,5V	66 pts (100mV/step)	1 mA
Step drain	0V —>7V	8 step (1V/step)	1 A

BIAS (10 Times)	Fixed Voltage	Num capture points (1 minute pol.)	Compliance (Current limit)
Gate	-5V	60 pts (1pt/sec)	1 mA
Drain	10V —>100V at every cycle +10V (10V step)	60 pts (1pt/sec)	1 mA

Stress test charts reference:

Every test refers to stress, so tests 1 refer to stress 0, that doesn't exist so it's the normal condition test and Final - IDVD IDVG (11 – IDVD IDVG) refer to 10 – Stress:

Number of Stress	Relative test
———	1 - IDVD IDVG
1 - Stress	2 - IDVD IDVG
2 - Stress	3 - IDVD IDVG
3 - Stress	4 - IDVD IDVG
4 - Stress	5 - IDVD IDVG
5 - Stress	6 - IDVD IDVG
6 - Stress	7 - IDVD IDVG
7 - Stress	8 - IDVD IDVG
8 - Stress	9 - IDVD IDVG
9 - Stress	10 - IDVD IDVG
10 - Stress	Final – IDVD IDVG

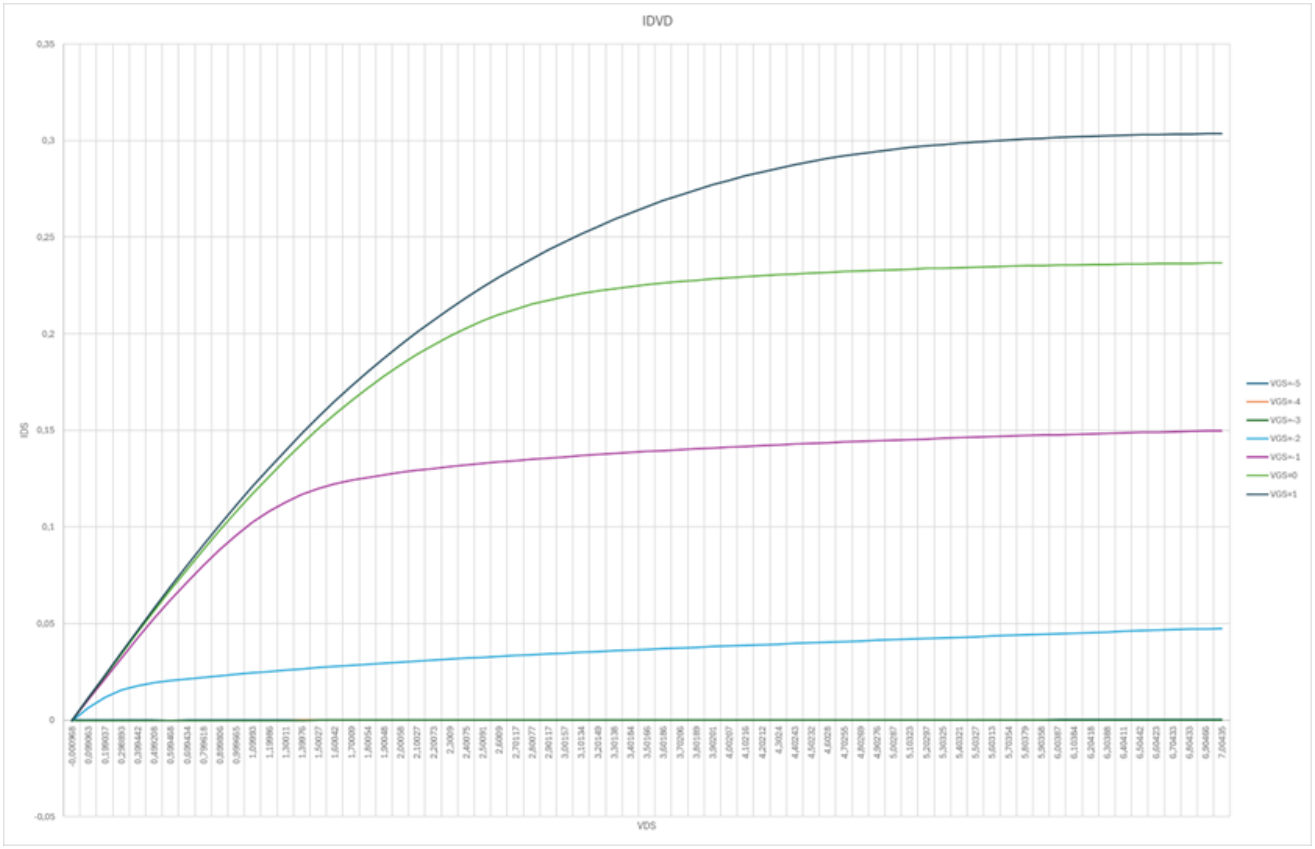

Figura 566: First IDVD test

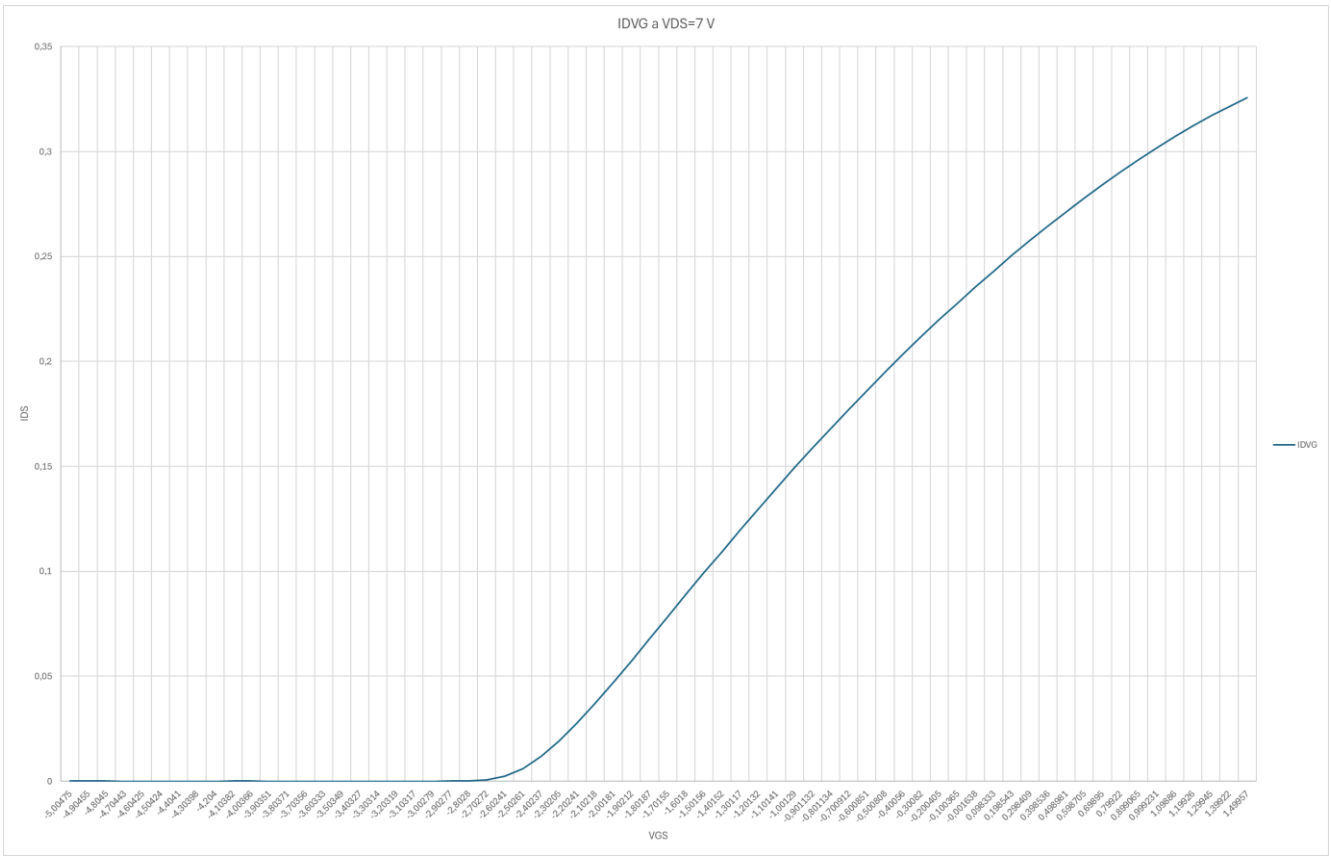

Figura 577: First IDVG test

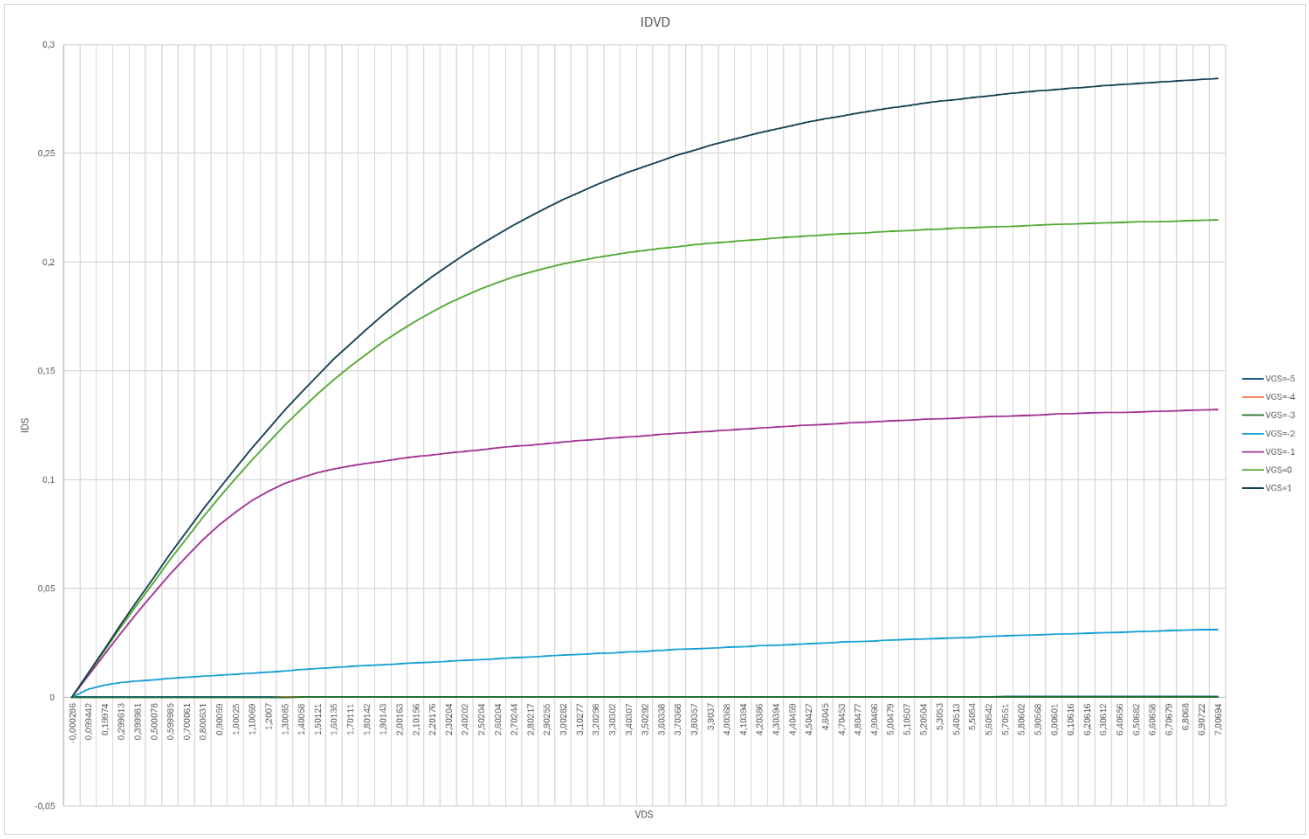

Figura 58: Final IDVD test

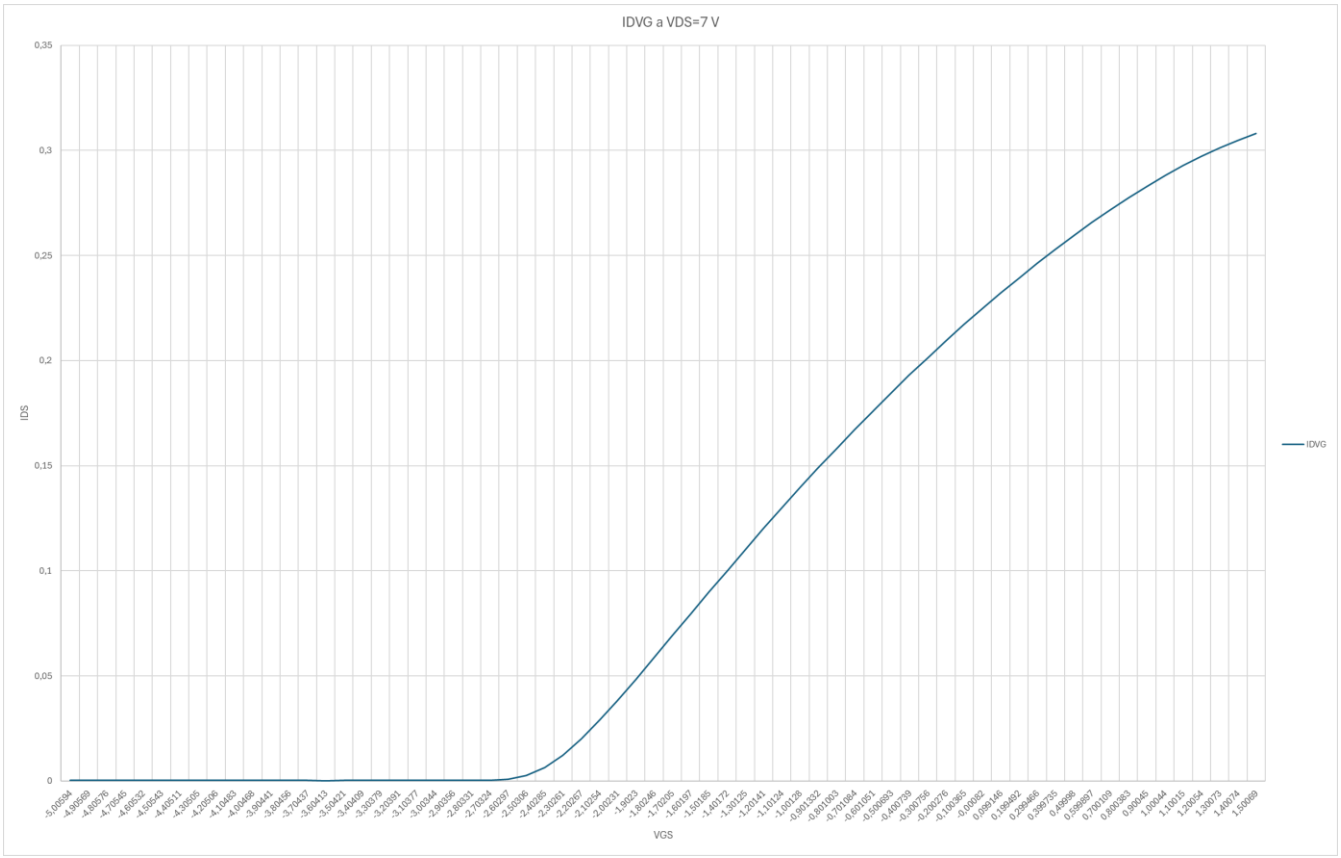

Figura 59: Final IDVG test

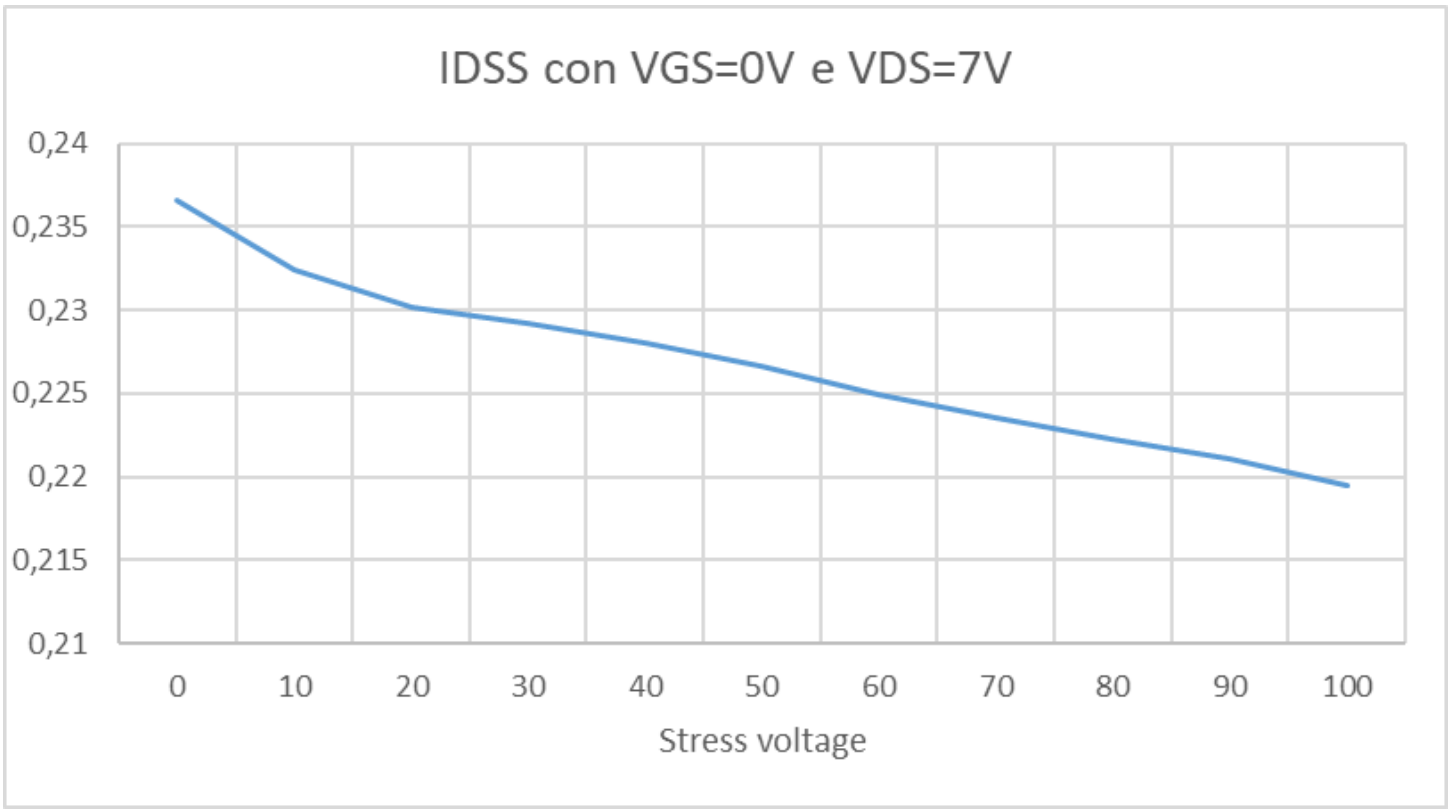

Figura 60: IDSS degradation

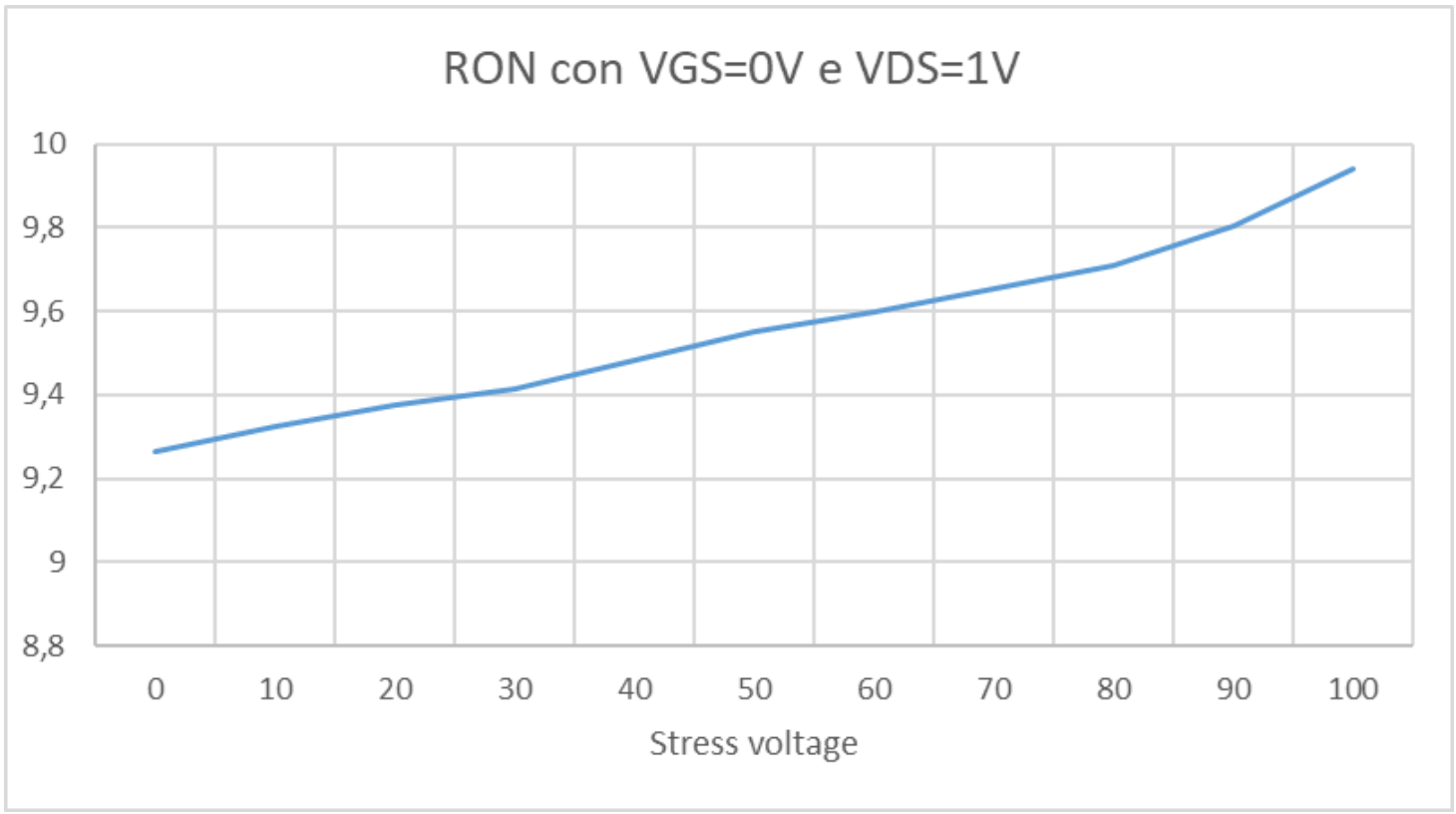

Figura 61: RON degradation

Figures 60 and 61 represent IDSS and RON degradation charts, captured after every stress, in fact at every test. Remember that every test refers to previous stress conditions, the stress in the previous cycle.

Plotted values that represent every test are chosen from test IDVD value list at certain condition; for example, IDSS at 30V bias is picked in test measures at 4th cycle, of course after 30V Stress at condition VGS=0V and VDS=7V. Same procedure for Ron (1/IDSS) from same IDVD file but with VGS=0V and VDS=1V.

We can observe a decrease in IDSS increasing the stress voltage and opposite for RON

10 Conclusions

Exceeding any expectations, a LabVIEW retro-engineered version of Kickstart 2 and more can be done.

Difficulties were finding a complete sweeping measure example to start from: Manual doesn't provide a real working example because Tektronix wants you to use their own proprietary software, fortunately there was Tektronix GitHub repository.

A homemade instrument control interface has pros to do what you want and know what instrument do in aspects of data structure and signal source and measure timing instead factory one.

Moreover, creating own instrument pc control version is expensive in terms of time, but free in economic terms (you don't have to pay licenses).

References

1 Wikipedia: https://en.wikipedia.org/wiki/Current%E2%80%93voltage_characteristic
2 Keithley 2612B manual: https://www.tek.com/en/keithley-source-measure-units/smu-2600b-series-sourcemeter-manual-8
3 Keithley GitHub repository: https://github.com/tektronix/keithley
4 Keithley starting base script: https://github.com/tektronix/keithley/blob/main/Instrument_Examples/Series_2600/Pulsed_Sweep/KE26XXB_Pulsed_Sweep_Dual.tsp
5 Rohm semiconductor: https://techweb.rohm.com/product/power-device/gan/23841/

Appendix

Notes: if you want to try standalone code, run it in Visual Studio Code with tsp plugin or in whatever Keithley 2612B script section. First substitute “%*” with parameters in input section and uncomment print buffers lines at the end of file.

I created a GitHub repository on my page. Project is open-source GPL without any warranties and it is not complete (alpha version)

Personal GitHub page: https://github.com/gioelebianchi00/

10.1 Single sweep-step code

```
-- dichiarazione variabili in ingresso
--============================================
%.;start_drain=%g
%.;stop_drain=%g
livello_gate={%s}
%.;limiti_drain=%g
%.;limiti_gate=%g
%.;rangei_drain=%g
%.;rangei_gate=%g
inversione_porte=%d
autozero_drain=%d
autozero_gate=%d
source_drain=%d
source_gate=%d
autorange_drain=%d
autorange_gate=%d
source_autorange_drain=%d
source_autorange_gate=%d

%.;measdelay=%g
%.;nplc=%g
```

```
numPoints=%d
numSweep=%d
remoteSense=false
%.;source_range_drain=%g
%.;source_range_gate=%g
--================================================
--dichiarazione variabili operative
--================================================
remoteSense=false
if inversione_porte==1 then
gate=smub
drain=smua
else
gate=smua
drain=smub
end
--================================================


-- dichiarazione tempi dei timer
--================================================
-- mi calcolo la finestra di misura
finestra_misura=(1/localnode.linefreq)*nplc
-- imposto la lunghezza dell'impulso. la dc sweep si origina da una pulsed sweep.
pulseWidth=measdelay+finestra_misura
--il periodo è l'impulso + più un t_off, una bassa percentuale dell'impulso
pulsePeriod=pulseWidth+(pulseWidth*1/100)


--================================================
```

```
-- dichiarazione timer e gestore eventi
--=============================================
-- Timer 1: timer periodo (larghezza gradino)
timer_periodo=trigger.timer[1]
-- Timer 2: Timer ritardo misura
timer_misura=trigger.timer[2]
-- Timer 3: Pulse Width Timer
timer_larghezza_impulso=trigger.timer[3]


-- Gestore eventi 1: Event Blender per la partenza di iniziale di smua e l'aggiornamento di smua al trigger di smub
gestore_avvio_gate=trigger.blender[1]
--=============================================
-- Configure Channel A Settings
--==============================
gate.reset()


if source_gate==1 then
gate.source.func = gate.OUTPUT_DCVOLTS
else
gate.source.func = gate.OUTPUT_DCAMPS
end


-- imposto modalità misura a 2 fili
if remoteSense == true then
gate.sense                  = gate.SENSE_REMOTE
else
gate.sense                  = gate.SENSE_LOCAL
end
```

```
if source_gate==1 then ------------------------------------

if source_autorange_gate==1 then
gate.source.autorangev          = gate.AUTORANGE_ON
gate.source.lowrangev= source_range_gate
else
gate.source.autorangev          = gate.AUTORANGE_OFF
gate.source.rangev              = source_range_gate
end

gate.source.levelv              = 0
-- Set the DC bias limit.  This is not the limit used during the pulses. ma è il valore di idle della gate
gate.source.limiti              = limiti_gate

else------------------------

if source_autorange_gate==1 then
gate.source.autorangei          = gate.AUTORANGE_ON
gate.source.lowrangei=source_range_gate
else
gate.source.autorangei          = gate.AUTORANGE_OFF
gate.source.rangei          =source_range_gate
end

gate.source.leveli              = 0
-- Set the DC bias limit.  This is not the limit used during the pulses. ma è il valore di idle della gate
gate.source.limitv              = limiti_gate
end ------------------------------------------------

-- Disabling Auto-Ranging and Auto-Zero ensures accurate and consistent timing
```

```
if autozero_gate==0 then
gate.measure.autozero          = gate.AUTOZERO_OFF
elseif autozero_gate==1 then
gate.measure.autozero          = gate.AUTOZERO_ONCE
else
gate.measure.autozero          = gate.AUTOZERO_AUTO
end

if autorange_gate==0 then ------------------------

if source_gate==1 then
gate.measure.autorangei        = gate.AUTORANGE_OFF
gate.measure.rangei            = rangei_gate
else
gate.measure.autorangev        = gate.AUTORANGE_OFF
gate.measure.rangev            = rangei_gate
end

elseif  autorange_gate==1 then -----------

if source_gate==1 then
gate.measure.autorangei        = gate.AUTORANGE_ON
gate.measure.lowrangei           = rangei_gate
else
gate.measure.autorangev        = gate.AUTORANGE_ON
gate.measure.lowrangev           = rangei_gate
end

else --------------------------
```

```
end ------------------------------

gate.measure.nplc               = nplc
-- A timer will be used to set the measure delay and synchronize the measurement
-- between the two SMUs so set the built in delay to 0.
gate.measure.delay              = 0

-- Prepare the Reading Buffers
gate.nvbuffer1.clear()
gate.nvbuffer1.collecttimestamps= 1
gate.nvbuffer2.clear()
gate.nvbuffer2.collecttimestamps = 1
--==============================
-- End Channel A Settings

-- Configure Channel B Settings
--==============================
drain.reset()
if source_drain==1 then
drain.source.func               = drain.OUTPUT_DCVOLTS
else
drain.source.func               = drain.OUTPUT_DCAMPS
end
-- imposto modalità misura a 2 fili
if remoteSense == true then
drain.sense                     = drain.SENSE_REMOTE
else
drain.sense                     = drain.SENSE_LOCAL
end
```

```
if source_drain==1 then----------------------------------------

if source_autorange_drain==1 then
drain.source.autorangev        = drain.AUTORANGE_ON
drain.source.lowrangev           = math.min(math.abs(stop_drain),math.abs(start_drain))
else
drain.source.autorangev        = drain.AUTORANGE_OFF
drain.source.rangev            =  math.max(math.abs(stop_drain),math.abs(start_drain))
end

drain.source.levelv            = 0
-- Set the DC bias limit.  This is not the limit used during the pulses. ma è il valore di idle della smu
drain.source.limiti            = limiti_drain
else------------------------------------------------
if source_autorange_drain==1 then
drain.source.autorangei        = drain.AUTORANGE_ON
drain.source.lowrangei           = math.min(math.abs(stop_drain),math.abs(start_drain))
else
drain.source.autorangei        = drain.AUTORANGE_OFF
drain.source.rangei            = math.max(math.abs(stop_drain),math.abs(start_drain))
end
drain.source.leveli            = 0
-- Set the DC bias limit.  This is not the limit used during the pulses. ma è il valore di idle della smu
drain.source.limitv            = limiti_drain
end-------------------------------------------

-- Disabling Auto-Ranging and Auto-Zero ensures accurate and consistent timing

if autozero_drain==0 then
drain.measure.autozero         = drain.AUTOZERO_OFF
```

```
elseif autozero_drain==1 then
drain.measure.autozero          = drain.AUTOZERO_ONCE
elseif autozero_drain==2 then
drain.measure.autozero          = drain.AUTOZERO_AUTO
end

if autorange_drain==0 then ------------------------------

if source_drain==1 then
drain.measure.autorangei            = drain.AUTORANGE_OFF
drain.measure.rangei                = rangei_drain
else
drain.measure.autorangev            = drain.AUTORANGE_OFF
drain.measure.rangev                = rangei_drain
end

elseif  autorange_drain==1 then -----------------------------

if source_drain==1 then
drain.measure.autorangei            = drain.AUTORANGE_ON
drain.measure.lowrangei             = rangei_drain
else
drain.measure.autorangev            = drain.AUTORANGE_ON
drain.measure.lowrangev             = rangei_drain
end

else

end ------------------------------------------------------------
```

```
drain.measure.nplc              = nplc
-- A timer will be used to set the measure delay and synchronize the measurement
-- between the two SMUs so set the built in delay to 0.
drain.measure.delay             = 0

-- Prepare the Reading Buffers
drain.nvbuffer1.clear()
drain.nvbuffer1.collecttimestamps= 1
drain.nvbuffer2.clear()
drain.nvbuffer2.collecttimestamps= 1
--============================
-- End Channel B Settings
-- Configure the Trigger Model
--============================
--Configurazione timer e trigger. Sono la parte essenziale dello script, dato che gestiscono i tempi di misura
trigger.clear()

-- Timer 1 controls the pulse period, cioè la durata del gradino
timer_periodo.count         = numPoints > 1 and numPoints - 1 or 1
timer_periodo.delay         = pulsePeriod
--il passthrough mi consente di ricevere l'interrupt quando il timer parte, ossia l'event id.
--Mi servirà da stimolo per gli altri timer, che partiranno all'inizo del gradino
-- definendo il momento della misura e la durata dell'impulso di alimentazione
timer_periodo.passthrough   = true
-- il timer 1 parte quando la drain, in questo caso, inizia la sweep: cioè entra nello stato armed
timer_periodo.stimulus      = drain.trigger.ARMED_EVENT_ID

-- Timer 2 controls the measurement. il timer 2 gestisce il ritardo di misura. scaduto quello, inizia la finestra di misura
timer_misura.count          = 1
```

```
timer_misura.delay          = measdelay
-- il passthrough è false perchè l'interrupt lo produrre alla fine.
-- Scaduto il delay di misura, inizia la finestra di misura
timer_misura.passthrough    = false
--il timer 1 grazie al passthrough mi fa partire il timer 2 all'inizio del gradino
timer_misura.stimulus       = timer_periodo.EVENT_ID

-- Timer 3 controls the pulse width
timer_larghezza_impulso.count          = 1
timer_larghezza_impulso.delay          = pulseWidth
-- il passthrough è false perchè l'interrupt me lo produce alla fine, perchè
timer_larghezza_impulso.passthrough = false
--il timer 1 grazie al passthrough mi fa partire il timer 3 all'inizio del gradino
timer_larghezza_impulso.stimulus       = timer_periodo.EVENT_ID

-- Configure gate Trigger Model for Sweep
-- imposto un singolo valore di tensione, costante per tutta la durata della sweep.
-- quindi quello che ottengo è la sweep di un singolo valore. il risultato è appunto una valore costante
if source_gate==1 then----------------------
gate.trigger.source.listv(livello_gate)
gate.trigger.source.limiti      = limiti_gate
else ----------------------------------------
gate.trigger.source.listi(livello_gate)
gate.trigger.source.limitv      = limiti_gate
end-----------------------------------------
gate.trigger.measure.action     = gate.ASYNC
gate.trigger.measure.iv(gate.nvbuffer1, gate.nvbuffer2)
--fine impulso: se voglio una dc sweep, anzichè mandare in idle l'uscita, la tengo costante
gate.trigger.endpulse.action    = gate.SOURCE_HOLD
-- endsweep mi dice cosa fare quando finisce l'operazione sweep
```

```
gate.trigger.endsweep.action    = gate.SOURCE_HOLD
gate.trigger.arm.count = 1
gate.trigger.count              = numSweep
-- arm sweep indica quanto sweep deve fare
drain.trigger.arm.stimulus      = 0

--event blender per la partenza
gestore_avvio_gate.orenable = true
gestore_avvio_gate.stimulus[1] = gate.trigger.ARMED_EVENT_ID
gestore_avvio_gate.stimulus[2] = drain.trigger.ARMED_EVENT_ID

--dico quali stimoli/timer attivano le azioni
gate.trigger.source.stimulus = gestore_avvio_gate.EVENT_ID
gate.trigger.measure.stimulus   = timer_misura.EVENT_ID
gate.trigger.endpulse.stimulus  = drain.trigger.SWEEP_COMPLETE_EVENT_ID

gate.trigger.source.action      = gate.ENABLE

-- Configure drain Trigger Model for Sweep
if source_drain==1 then------------------------
drain.trigger.source.linearv(start_drain, stop_drain, numPoints)
drain.trigger.source.limiti     = limiti_drain
else-------------------------------------------
drain.trigger.source.lineari(start_drain, stop_drain, numPoints)
drain.trigger.source.limitv     = limiti_drain
end--------------------------------------------
drain.trigger.measure.action        = drain.ENABLE
drain.trigger.measure.iv(drain.nvbuffer1, drain.nvbuffer2)
--fine impulso: se voglio una dc sweep, anzichè mandare in idle l'uscita, la tengo costante
drain.trigger.endpulse.action   = drain.SOURCE_HOLD
```

```
-- endsweep mi dice cosa fare quando finisce l'operazione sweep
drain.trigger.endsweep.action   = drain.SOURCE_HOLD
drain.trigger.arm.count = numSweep
-- num punti sweep
drain.trigger.count             = numPoints
-- arm sweep indica quanto sweep deve fare
drain.trigger.arm.stimulus      = 0
--dico quali timer attivanzo le azioni
drain.trigger.source.stimulus   = timer_periodo.EVENT_ID
drain.trigger.measure.stimulus  = timer_misura.EVENT_ID
drain.trigger.endpulse.stimulus = timer_larghezza_impulso.EVENT_ID
drain.trigger.source.action     = drain.ENABLE
--==============================
-- End Trigger Model Configuration

gate.source.output              = gate.OUTPUT_ON
drain.source.output             = drain.OUTPUT_ON

-- Start the trigger model execution
gate.trigger.initiate()
delay(0.05)
drain.trigger.initiate()

-- Wait until the sweep has completed
waitcomplete()

drain.source.output             = drain.OUTPUT_OFF
delay(0.05)
gate.source.output              = gate.OUTPUT_OFF
```

```
--printbuffer(1, numPoints, gate.nvbuffer2.readings)
--printbuffer(1, numPoints, gate.nvbuffer1.readings)
--printbuffer(1, numPoints, gate.nvbuffer1.timestamps)

--printbuffer(1, numPoints, drain.nvbuffer2.readings)
--printbuffer(1, numPoints, drain.nvbuffer1.readings)
--printbuffer(1, numPoints, drain.nvbuffer1.timestamps)
```

10.2 Dual sweep-step code

```
-- dichiarazione variabili in ingresso
--============================================
%.;start_drain=%g
%.;stop_drain=%g
livello_gate={%s}
%.;limiti_drain=%g
%.;limiti_gate=%g
%.;rangei_drain=%g
%.;rangei_gate=%g
inversione_porte=%d
autozero_drain=%d
autozero_gate=%d
source_drain=%d
source_gate=%d
autorange_drain=%d
autorange_gate=%d
source_autorange_drain=%d
source_autorange_gate=%d
```

```
%.;measdelay=%g
%.;nplc=%g

numPoints=%d
numSweep=%d
%.;source_range_drain=%g
%.;source_range_gate=%g
remoteSense=false
--================================================
--dichiarazione variabili operative
--================================================
remoteSense=false
if inversione_porte==1 then
gate=smub
drain=smua
else
gate=smua
drain=smub
end

dim_livello=table.getn(livello_gate)

-- se il numero di livelli è minore del numero di piramidi, allora ripeto i valori
if (dim_livello<numSweep) then

   for i=1, (numSweep-dim_livello) do
      table.insert(livello_gate, livello_gate[i])
   end

end
```

```
--============================================


-- dichiarazione tempi dei timer
--============================================
-- mi calcolo la finestra di misura
finestra_misura=(1/localnode.linefreq)*nplc
-- imposto la lunghezza dell'impulso. la dc sweep si origina da una pulsed sweep.
pulseWidth=measdelay+finestra_misura
--il periodo è l'impulso + più un t_off, una bassa percentuale dell'impulso
pulsePeriod=pulseWidth+(pulseWidth*1/100)
--============================================


-- dichiarazione timer e gestore eventi
--============================================
-- Timer 1: timer periodo (larghezza gradino)
timer_periodo=trigger.timer[1]
-- Timer 2: Timer ritardo misura
timer_misura=trigger.timer[2]
-- Timer 3: Pulse Width Timer
timer_larghezza_impulso=trigger.timer[3]


-- Gestore eventi 1: Event Blender per la partenza di iniziale di smua e l'aggiornamento di smua al trigger di smub
gestore_avvio_gate=trigger.blender[1]
--============================================
-- Configure Channel A Settings
--============================
gate.reset()
```

```
if source_gate==1 then
gate.source.func = gate.OUTPUT_DCVOLTS
else
gate.source.func = gate.OUTPUT_DCAMPS
end

-- imposto modalità misura a 2 fili
if remoteSense == true then
gate.sense                  = gate.SENSE_REMOTE
else
gate.sense                  = gate.SENSE_LOCAL
end

if source_gate==1 then ------------------------------------

if source_autorange_gate==1 then
gate.source.autorangev        = gate.AUTORANGE_ON
gate.source.lowrangev=source_range_gate
else
gate.source.autorangev        = gate.AUTORANGE_OFF
gate.source.rangev            =source_range_gate
end

gate.source.levelv            = 0
-- Set the DC bias limit.  This is not the limit used during the pulses. ma è il valore di idle della gate
gate.source.limiti            = limiti_gate

else------------------------

if source_autorange_gate==1 then
```

```
gate.source.autorangei          = gate.AUTORANGE_ON
gate.source.lowrangei=source_range_gate
else
gate.source.autorangei          = gate.AUTORANGE_OFF
gate.source.rangei          =source_range_gate
end

gate.source.leveli              = 0
-- Set the DC bias limit.  This is not the limit used during the pulses. ma è il valore di idle della gate
gate.source.limitv              = limiti_gate
end ------------------------------------------------

-- Disabling Auto-Ranging and Auto-Zero ensures accurate and consistent timing
if autozero_gate==0 then
gate.measure.autozero           = gate.AUTOZERO_OFF
elseif autozero_gate==1 then
gate.measure.autozero           = gate.AUTOZERO_ONCE
else
gate.measure.autozero           = gate.AUTOZERO_AUTO
end

if autorange_gate==0 then ------------------------

if source_gate==1 then
gate.measure.autorangei         = gate.AUTORANGE_OFF
gate.measure.rangei             = rangei_gate
else
gate.measure.autorangev         = gate.AUTORANGE_OFF
gate.measure.rangev             = rangei_gate
end
```

```
elseif  autorange_gate==1 then -----------

if source_gate==1 then
gate.measure.autorangei         = gate.AUTORANGE_ON
gate.measure.lowrangei             = rangei_gate
else
gate.measure.autorangev         = gate.AUTORANGE_ON
gate.measure.lowrangev             = rangei_gate
end

else -------------------------

end ------------------------------

gate.measure.nplc               = nplc
-- A timer will be used to set the measure delay and synchronize the measurement
-- between the two SMUs so set the built in delay to 0.
gate.measure.delay              = 0

-- Prepare the Reading Buffers
gate.nvbuffer1.clear()
gate.nvbuffer1.collecttimestamps= 1
gate.nvbuffer1.appendmode=1
gate.nvbuffer2.clear()
gate.nvbuffer2.collecttimestamps = 1
gate.nvbuffer2.appendmode=1
--============================
-- End Channel A Settings
```

```
-- Configure Channel B Settings
--==============================
drain.reset()
if source_drain==1 then
drain.source.func              = drain.OUTPUT_DCVOLTS
else
drain.source.func              = drain.OUTPUT_DCAMPS
end
-- imposto modalità misura a 2 fili
if remoteSense == true then
drain.sense                    = drain.SENSE_REMOTE
else
drain.sense                    = drain.SENSE_LOCAL
end

if source_drain==1 then----------------------------------------

if source_autorange_drain==1 then
drain.source.autorangev        = drain.AUTORANGE_ON
drain.source.rangev            = math.min(math.abs(stop_drain),math.abs(start_drain))
else
drain.source.autorangev        = drain.AUTORANGE_OFF
drain.source.rangev            =  math.max(math.abs(stop_drain),math.abs(start_drain))
end

drain.source.levelv            = 0
-- Set the DC bias limit.  This is not the limit used during the pulses. ma è il valore di idle della smu
drain.source.limiti            = limiti_drain
else----------------------------------------------------
if source_autorange_drain==1 then
```

```
drain.source.autorangei        = drain.AUTORANGE_ON
drain.source.rangei            = math.min(math.abs(stop_drain),math.abs(start_drain))
else
drain.source.autorangei        = drain.AUTORANGE_OFF
drain.source.rangei            = math.max(math.abs(stop_drain),math.abs(start_drain))
end
drain.source.leveli            = 0
-- Set the DC bias limit.  This is not the limit used during the pulses. ma è il valore di idle della smu
drain.source.limitv            = limiti_drain
end-------------------------------------------


-- Disabling Auto-Ranging and Auto-Zero ensures accurate and consistent timing


if autozero_drain==0 then
drain.measure.autozero         = drain.AUTOZERO_OFF
elseif autozero_drain==1 then
drain.measure.autozero         = drain.AUTOZERO_ONCE
elseif autozero_drain==2 then
drain.measure.autozero         = drain.AUTOZERO_AUTO
end


if autorange_drain==0 then ------------------------------


if source_drain==1 then
drain.measure.autorangei           = drain.AUTORANGE_OFF
drain.measure.rangei               = rangei_drain
else
drain.measure.autorangev           = drain.AUTORANGE_OFF
drain.measure.rangev               = rangei_drain
end
```

```
elseif  autorange_drain==1 then ---------------------------

if source_drain==1 then
drain.measure.autorangei            = drain.AUTORANGE_ON
drain.measure.lowrangei             = rangei_drain
else
drain.measure.autorangev            = drain.AUTORANGE_ON
drain.measure.lowrangev             = rangei_drain
end

else

end ------------------------------------------------------------

drain.measure.nplc              = nplc
-- A timer will be used to set the measure delay and synchronize the measurement
-- between the two SMUs so set the built in delay to 0.
drain.measure.delay             = 0

-- Prepare the Reading Buffers
drain.nvbuffer1.clear()
drain.nvbuffer1.collecttimestamps= 1
drain.nvbuffer1.appendmode=1
drain.nvbuffer2.clear()
drain.nvbuffer2.collecttimestamps= 1
drain.nvbuffer2.appendmode=1
--============================
-- End Channel B Settings
-- Configure the Trigger Model
```

```
--============================

--Configurazione timer e trigger. Sono la parte essenziale dello script, dato che gestiscono i tempi di misura

trigger.clear()


-- Timer 1 controls the pulse period, cioè la durata del gradino

timer_periodo.count        = numPoints > 1 and numPoints - 1 or 1

timer_periodo.delay        = pulsePeriod

--il passthrough mi consente di ricevere l'interrupt quando il timer parte, ossia l'event id.

--Mi servirà da stimolo per gli altri timer, che partiranno all'inizo del gradino

-- definendo il momento della misura e la durata dell'impulso di alimentazione

timer_periodo.passthrough   = true

-- il timer 1 parte quando la drain, in questo caso, inizia la sweep: cioè entra nello stato armed

timer_periodo.stimulus      = drain.trigger.ARMED_EVENT_ID


-- Timer 2 controls the measurement. il timer 2 gestisce il ritardo di misura. scaduto quello, inizia la finestra di misura

timer_misura.count         = 1

timer_misura.delay         = measdelay

-- il passthrough è false perchè l'interrupt lo produrre alla fine.

-- Scaduto il delay di misura, inizia la finestra di misura

timer_misura.passthrough    = false

--il timer 1 grazie al passthrough mi fa partire il timer 2 all'inizio del gradino

timer_misura.stimulus      = timer_periodo.EVENT_ID


-- Timer 3 controls the pulse width

timer_larghezza_impulso.count          = 1

timer_larghezza_impulso.delay          = pulseWidth

-- il passthrough è false perchè l'interrupt me lo produce alla fine, perchè

timer_larghezza_impulso.passthrough = false

--il timer 1 grazie al passthrough mi fa partire il timer 3 all'inizio del gradino
```

```
timer_larghezza_impulso.stimulus        = timer_periodo.EVENT_ID


limitI=1
gate.source.output              = gate.OUTPUT_ON
drain.source.output             = drain.OUTPUT_ON


for i=1,numSweep,1 do


-- Configure gate Trigger Model for Sweep
-- imposto un singolo valore di tensione, costante per tutta la durata della sweep.
-- quindi quello che ottengo è la sweep di un singolo valore. il risultato è appunto una valore costante
--in lua a differenza del linguaggio c/c++, il primo elemento di un array ha indice 1, non 0


if source_gate==1 then-----------------------
gate.trigger.source.listv({livello_gate[i]})
gate.trigger.source.limiti      = limiti_gate
else ----------------------------------------
gate.trigger.source.listi({livello_gate[i]})
gate.trigger.source.limitv      = limiti_gate
end------------------------------------------


gate.trigger.measure.action     = gate.ASYNC
gate.trigger.measure.iv(gate.nvbuffer1, gate.nvbuffer2)
--fine impulso: se voglio una dc sweep, anzichè mandare in idle l'uscita, la tengo costante
gate.trigger.endpulse.action    = gate.SOURCE_HOLD
-- endsweep mi dice cosa fare quando finisce l'operazione sweep
gate.trigger.endsweep.action    = gate.SOURCE_HOLD
gate.trigger.arm.count = 1
gate.trigger.count              = 1
-- arm sweep indica quanto sweep deve fare
```

```
drain.trigger.arm.stimulus      = 0
--dico quali timer attivanzo le azioni


--event blender per la partenza
trigger.blender[1].orenable = true
trigger.blender[1].stimulus[1] = gate.trigger.ARMED_EVENT_ID
trigger.blender[1].stimulus[2] = drain.trigger.ARMED_EVENT_ID


--dico quali stimoli/timer attivanzo le azioni
gate.trigger.source.stimulus = trigger.blender[1].EVENT_ID
gate.trigger.measure.stimulus   = trigger.timer[2].EVENT_ID
--gate.trigger.endpulse.stimulus    = drain.trigger.SWEEP_COMPLETE_EVENT_ID
gate.trigger.endpulse.stimulus  = drain.trigger.IDLE_EVENT_ID


gate.trigger.source.action      = gate.ENABLE


-- Configure drain Trigger Model for Sweep


if source_drain==1 then-----------------------
drain.trigger.source.linearv(start_drain, stop_drain, numPoints)
drain.trigger.source.limiti     = limiti_drain
else-------------------------------------------
drain.trigger.source.lineari(start_drain, stop_drain, numPoints)
drain.trigger.source.limitv     = limiti_drain
end---------------------------------------------
drain.trigger.measure.action        = drain.ENABLE
drain.trigger.measure.iv(drain.nvbuffer1, drain.nvbuffer2)
--fine impulso: se voglio una dc sweep, anzichè mandare in idle l'uscita, la tengo costante
drain.trigger.endpulse.action   = drain.SOURCE_HOLD
-- endsweep mi dice cosa fare quando finisce l'operazione sweep
```

```
drain.trigger.endsweep.action   = drain.SOURCE_HOLD
drain.trigger.arm.count = 1
-- num punti sweep
drain.trigger.count             = numPoints
-- arm sweep indica quanto sweep deve fare
drain.trigger.arm.stimulus      = 0
--dico quali timer attivanzo le azioni
drain.trigger.source.stimulus   = trigger.timer[1].EVENT_ID
drain.trigger.measure.stimulus  = trigger.timer[2].EVENT_ID
drain.trigger.endpulse.stimulus = trigger.timer[3].EVENT_ID
drain.trigger.source.action     = drain.ENABLE
--================================
-- End Trigger Model Configuration

-- Start the trigger model execution
gate.trigger.initiate()
delay(0.05)
drain.trigger.initiate()

-- Wait until the sweep has completed
waitcomplete()

-- Configure gate Trigger Model for Sweep
-- imposto un singolo valore di tensione, costante per tutta la durata della sweep.
-- quindi quello che ottengo è la sweep di un singolo valore. il risultato è appunto una valore costante
if source_gate==1 then----------------------
gate.trigger.source.listv({livello_gate[i]})
gate.trigger.source.limiti      = limiti_gate
else ----------------------------------------
gate.trigger.source.listi({livello_gate[i]})
```

```
gate.trigger.source.limitv      = limiti_gate
end----------------------------------------
gate.trigger.measure.action     = gate.ASYNC
gate.trigger.measure.iv(gate.nvbuffer1, gate.nvbuffer2)
--fine impulso: se voglio una dc sweep, anzichè mandare in idle l'uscita, la tengo costante
gate.trigger.endpulse.action    = gate.SOURCE_HOLD
-- endsweep mi dice cosa fare quando finisce l'operazione sweep
gate.trigger.endsweep.action    = gate.SOURCE_HOLD
gate.trigger.arm.count = 1
gate.trigger.count              = 1
-- arm sweep indica quanto sweep deve fare
drain.trigger.arm.stimulus      = 0
--dico quali timer attivanzo le azioni

--event blender per la partenza
trigger.blender[1].orenable = true
trigger.blender[1].stimulus[1] = gate.trigger.ARMED_EVENT_ID
trigger.blender[1].stimulus[2] = drain.trigger.ARMED_EVENT_ID

--dico quali stimoli/timer attivanzo le azioni
gate.trigger.source.stimulus = trigger.blender[1].EVENT_ID
gate.trigger.measure.stimulus   = trigger.timer[2].EVENT_ID
--gate.trigger.endpulse.stimulus    = drain.trigger.SWEEP_COMPLETE_EVENT_ID
gate.trigger.endpulse.stimulus  = drain.trigger.IDLE_EVENT_ID

--gate.trigger.source.action       = gate.ENABLE

-- Configure drain Trigger Model for Sweep

if source_drain==1 then------------------------
```

```
drain.trigger.source.linearv(stop_drain, start_drain, numPoints)
drain.trigger.source.limiti     = limiti_drain
else----------------------------------------------
drain.trigger.source.lineari(stop_drain, start_drain, numPoints)
drain.trigger.source.limitv     = limiti_drain
end-----------------------------------------------
drain.trigger.measure.action        = drain.ENABLE
drain.trigger.measure.iv(drain.nvbuffer1, drain.nvbuffer2)
--fine impulso: se voglio una dc sweep, anzichè mandare in idle l'uscita, la tengo costante
drain.trigger.endpulse.action    = drain.SOURCE_HOLD
-- endsweep mi dice cosa fare quando finisce l'operazione sweep
drain.trigger.endsweep.action    = drain.SOURCE_HOLD
drain.trigger.arm.count = 1
-- num punti sweep
drain.trigger.count              = numPoints
-- arm sweep indica quanto sweep deve fare
drain.trigger.arm.stimulus      = 0
--dico quali timer attivanzo le azioni
drain.trigger.source.stimulus    = trigger.timer[1].EVENT_ID
drain.trigger.measure.stimulus  = trigger.timer[2].EVENT_ID
drain.trigger.endpulse.stimulus = trigger.timer[3].EVENT_ID
--drain.trigger.source.action       = drain.ENABLE
--============================
-- End Trigger Model Configuration


-- Start the trigger model execution
gate.trigger.initiate()
--delay(0.05)
drain.trigger.initiate()
waitcomplete()
```

```
end


drain.source.output             = drain.OUTPUT_OFF
delay(0.05)
gate.source.output              = gate.OUTPUT_OFF


--printbuffer(1, numPoints, gate.nvbuffer2.readings)
--printbuffer(1, numPoints, gate.nvbuffer1.readings)
--printbuffer(1, numPoints, gate.nvbuffer1.timestamps)


--printbuffer(1, numPoints, drain.nvbuffer2.readings)
--printbuffer(1, numPoints, drain.nvbuffer1.readings)
--printbuffer(1, numPoints, drain.nvbuffer1.timestamps)
```

10.3 Bias code

```
-- dichiarazione variabili in ingresso
--============================================
%.;livello_drain=%g
%.;livello_gate=%g
%.;limiti_drain=%g
%.;limiti_gate=%g
%.;rangei_drain=%g
%.;rangei_gate=%g
numPoints=%d
%.;pulsePeriod=%g
fmd=%d
inversione_porte=%d
autozero_drain=%d
autozero_gate=%d
```

```
source_drain=%d
source_gate=%d
autorange_drain=%d
autorange_gate=%d
source_autorange_drain=%d
source_autorange_gate=%d
%.;source_range_drain=%g
%.;source_range_gate=%g
--================================================


--dichiarazione variabili operative
--================================================
t_morto=0
numSweep=1
remoteSense=false
if inversione_porte==1 then
gate=smub
drain=smua
else
gate=smua
drain=smub
end
--================================================


-- dichiarazione tempi dei timer
--================================================
t_morto=pulsePeriod*1/100
measdelay=pulsePeriod*fmd/100
finestra_misura=pulsePeriod-measdelay-t_morto
pulseWidth=pulsePeriod-t_morto
```

```
nplc=finestra_misura*localnode.linefreq


--================================================
-- dichiarazione timer e gestore eventi
--================================================
-- Timer 1: timer periodo (larghezza gradino)
timer_periodo=trigger.timer[1]
-- Timer 2: Timer ritardo misura
timer_misura=trigger.timer[2]
-- Timer 3: Pulse Width Timer
timer_larghezza_impulso=trigger.timer[3]


-- Gestore eventi 1: Event Blender per la partenza di iniziale di gate e l'aggiornamento di gate al trigger di drain
gestore_avvio_gate=trigger.blender[1]
--================================================
-- Configure Channel A Settings
--================================
gate.reset()
if source_gate==1 then
gate.source.func = gate.OUTPUT_DCVOLTS
else
gate.source.func = gate.OUTPUT_DCAMPS
end
-- imposto modalità misura a 2 fili
if remoteSense == true then
gate.sense                  = gate.SENSE_REMOTE
else
gate.sense                  = gate.SENSE_LOCAL
```

```
end

if source_gate==1 then ------------------------------------

if source_autorange_gate==1 then
gate.source.autorangev          = gate.AUTORANGE_ON
gate.source.lowrangev=source_range_gate
else
gate.source.autorangev          = gate.AUTORANGE_OFF
gate.source.rangev              = source_range_gate
end

gate.source.levelv              = 0
-- Set the DC bias limit.  This is not the limit used during the pulses. ma è il valore di idle della gate
gate.source.limiti              = limiti_gate

else------------------------

if source_autorange_gate==1 then
gate.source.autorangei          = gate.AUTORANGE_ON
gate.source.lowrangei=source_range_gate
else
gate.source.autorangei          = gate.AUTORANGE_OFF
gate.source.rangei          = source_range_gate
end

gate.source.leveli              = 0
-- Set the DC bias limit.  This is not the limit used during the pulses. ma è il valore di idle della gate
gate.source.limitv              = limiti_gate
end ----------------------------------------------
```

```
-- Disabling Auto-Ranging and Auto-Zero ensures accurate and consistent timing

if autozero_gate==0 then
gate.measure.autozero          = gate.AUTOZERO_OFF
elseif autozero_gate==1 then
gate.measure.autozero          = gate.AUTOZERO_ONCE
else
gate.measure.autozero          = gate.AUTOZERO_AUTO
end

if autorange_gate==0 then ------------------------

if source_gate==1 then
gate.measure.autorangei        = gate.AUTORANGE_OFF
gate.measure.rangei            = rangei_gate
else
gate.measure.autorangev        = gate.AUTORANGE_OFF
gate.measure.rangev            = rangei_gate
end

elseif  autorange_gate==1 then -----------

if source_gate==1 then
gate.measure.autorangei        = gate.AUTORANGE_ON
gate.measure.lowrangei           = rangei_gate
else
gate.measure.autorangev        = gate.AUTORANGE_ON
gate.measure.lowrangev           = rangei_gate
end
```

```
else -------------------------

end ------------------------------

gate.measure.nplc               = nplc
-- A timer will be used to set the measure delay and synchronize the measurement
-- between the two SMUs so set the built in delay to 0.
gate.measure.delay              = 0

-- Prepare the Reading Buffers
gate.nvbuffer1.clear()
gate.nvbuffer1.collecttimestamps= 1
gate.nvbuffer2.clear()
gate.nvbuffer2.collecttimestamps = 1
--==============================
-- End Channel A Settings

-- Configure Channel B Settings
--==============================
drain.reset()
if source_drain==1 then
drain.source.func               = drain.OUTPUT_DCVOLTS
else
drain.source.func               = drain.OUTPUT_DCAMPS
end
-- imposto modalità misura a 2 fili
if remoteSense == true then
drain.sense                     = drain.SENSE_REMOTE
else
```

```
drain.sense                 = drain.SENSE_LOCAL
end

if source_drain==1 then----------------------------------------

if source_autorange_drain==1 then
drain.source.autorangev       = drain.AUTORANGE_ON
drain.source.lowrangev          = source_range_drain
else
drain.source.autorangev       = drain.AUTORANGE_OFF
drain.source.rangev           =source_range_drain
end

drain.source.levelv           = 0
-- Set the DC bias limit.  This is not the limit used during the pulses. ma è il valore di idle della smu
drain.source.limiti           = limiti_drain
else------------------------------------------------
if source_autorange_drain==1 then
drain.source.autorangei       = drain.AUTORANGE_ON
drain.source.lowrangei          =source_range_drain
else
drain.source.autorangei       = drain.AUTORANGE_OFF
drain.source.rangei           =source_range_drain
end
drain.source.leveli           = 0
-- Set the DC bias limit.  This is not the limit used during the pulses. ma è il valore di idle della smu
drain.source.limitv           = limiti_drain
end-------------------------------------------

-- Disabling Auto-Ranging and Auto-Zero ensures accurate and consistent timing
```

```
if autozero_drain==0 then
drain.measure.autozero          = drain.AUTOZERO_OFF
elseif autozero_drain==1 then
drain.measure.autozero          = drain.AUTOZERO_ONCE
elseif autozero_drain==2 then
drain.measure.autozero          = drain.AUTOZERO_AUTO
end

if autorange_drain==0 then ------------------------------

if source_drain==1 then
drain.measure.autorangei          = drain.AUTORANGE_OFF
drain.measure.rangei              = rangei_drain
else
drain.measure.autorangev          = drain.AUTORANGE_OFF
drain.measure.rangev              = rangei_drain
end

elseif  autorange_drain==1 then -------------------------

if source_drain==1 then
drain.measure.autorangei          = drain.AUTORANGE_ON
drain.measure.lowrangei           = rangei_drain
else
drain.measure.autorangev          = drain.AUTORANGE_ON
drain.measure.lowrangev           = rangei_drain
end

else
```

```
end ------------------------------------------------------------


drain.measure.nplc              = nplc
-- A timer will be used to set the measure delay and synchronize the measurement
-- between the two SMUs so set the built in delay to 0.
drain.measure.delay             = 0


-- Prepare the Reading Buffers
drain.nvbuffer1.clear()
drain.nvbuffer1.collecttimestamps= 1
drain.nvbuffer2.clear()
drain.nvbuffer2.collecttimestamps= 1
--============================
-- End Channel B Settings
-- Configure the Trigger Model
--============================
--Configurazione timer e trigger. Sono la parte essenziale dello script, dato che gestiscono i tempi di misura
trigger.clear()


-- Timer 1 controls the pulse period, cioè la durata del gradino
timer_periodo.count         = numPoints > 1 and numPoints - 1 or 1
timer_periodo.delay         = pulsePeriod
--il passthrough mi consente di ricevere l'interrupt quando il timer parte, ossia l'event id.
--Mi servirà da stimolo per gli altri timer, che partiranno all'inizo del gradino
-- definendo il momento della misura e la durata dell'impulso di alimentazione
timer_periodo.passthrough   = true
-- il timer 1 parte quando la drain, in questo caso, inizia la sweep: cioè entra nello stato armed
timer_periodo.stimulus      = drain.trigger.ARMED_EVENT_ID
```

```
-- Timer 2 controls the measurement. il timer 2 gestisce il ritardo di misura. scaduto quello, inizia la finestra di misura
timer_misura.count          = 1
timer_misura.delay          = measdelay
-- il passthrough è false perchè l'interrupt lo produrre alla fine.
-- Scaduto il delay di misura, inizia la finestra di misura
timer_misura.passthrough    = false
--il timer 1 grazie al passthrough mi fa partire il timer 2 all'inizio del gradino
timer_misura.stimulus       = timer_periodo.EVENT_ID

-- Timer 3 controls the pulse width
timer_larghezza_impulso.count          = 1
timer_larghezza_impulso.delay          = pulseWidth
-- il passthrough è false perchè l'interrupt me lo produce alla fine, perchè
timer_larghezza_impulso.passthrough = false
--il timer 1 grazie al passthrough mi fa partire il timer 3 all'inizio del gradino
timer_larghezza_impulso.stimulus       = timer_periodo.EVENT_ID

-- Configure gate Trigger Model for Sweep
-- imposto un singolo valore di tensione, costante per tutta la durata della sweep.
-- quindi quello che ottengo è la sweep di un singolo valore. il risultato è appunto una valore costante

if source_gate==1 then-----------------------
gate.trigger.source.listv({livello_gate})
gate.trigger.source.limiti      = limiti_gate
else ----------------------------------------
gate.trigger.source.listi({livello_gate})
gate.trigger.source.limitv      = limiti_gate
end-----------------------------------------
```

```
gate.trigger.measure.action     = gate.ASYNC
gate.trigger.measure.iv(gate.nvbuffer1, gate.nvbuffer2)
--fine impulso: se voglio una dc sweep, anzichè mandare in idle l'uscita, la tengo costante
gate.trigger.endpulse.action    = gate.SOURCE_HOLD
-- endsweep mi dice cosa fare quando finisce l'operazione sweep
gate.trigger.endsweep.action    = gate.SOURCE_HOLD
gate.trigger.arm.count = 1
gate.trigger.count              = numSweep
-- arm sweep indica quanto sweep deve fare
drain.trigger.arm.stimulus      = 0

--event blender per la partenza
gestore_avvio_gate.orenable = true
gestore_avvio_gate.stimulus[1] = gate.trigger.ARMED_EVENT_ID
gestore_avvio_gate.stimulus[2] = drain.trigger.ARMED_EVENT_ID

--dico quali stimoli/timer attivano le azioni
gate.trigger.source.stimulus = gestore_avvio_gate.EVENT_ID
gate.trigger.measure.stimulus   = timer_misura.EVENT_ID
gate.trigger.endpulse.stimulus  = drain.trigger.SWEEP_COMPLETE_EVENT_ID

gate.trigger.source.action      = gate.ENABLE

-- Configure drain Trigger Model for Sweep
if source_drain==1 then------------------------
drain.trigger.source.listv({livello_drain})
drain.trigger.source.limiti     = limiti_drain
else-------------------------------------------
drain.trigger.source.listi({livello_drain})
```

```
drain.trigger.source.limitv     = limiti_drain
end----------------------------------------


drain.trigger.measure.action        = drain.ENABLE
drain.trigger.measure.iv(drain.nvbuffer1, drain.nvbuffer2)
--fine impulso: se voglio una dc sweep, anzichè mandare in idle l'uscita, la tengo costante
drain.trigger.endpulse.action    = drain.SOURCE_HOLD
-- endsweep mi dice cosa fare quando finisce l'operazione sweep
drain.trigger.endsweep.action    = drain.SOURCE_HOLD
drain.trigger.arm.count = numSweep
-- num punti sweep
drain.trigger.count              = numPoints
-- arm sweep indica quanto sweep deve fare
drain.trigger.arm.stimulus      = 0
--dico quali timer attivanzo le azioni
drain.trigger.source.stimulus    = timer_periodo.EVENT_ID
drain.trigger.measure.stimulus  = timer_misura.EVENT_ID
drain.trigger.endpulse.stimulus = timer_larghezza_impulso.EVENT_ID
drain.trigger.source.action     = drain.ENABLE
--==============================
-- End Trigger Model Configuration


gate.source.output               = gate.OUTPUT_ON
drain.source.output              = drain.OUTPUT_ON


-- Start the trigger model execution
gate.trigger.initiate()
delay(0.05)
drain.trigger.initiate()
```

```
-- Wait until the sweep has completed
waitcomplete()
--delay(0.05)

drain.source.output             = drain.OUTPUT_OFF
delay(0.05)
gate.source.output              = gate.OUTPUT_OFF

--printbuffer(1, numPoints, gate.nvbuffer2.readings)
--printbuffer(1, numPoints, gate.nvbuffer1.readings)
--printbuffer(1, numPoints, gate.nvbuffer1.timestamps)

--printbuffer(1, numPoints, drain.nvbuffer2.readings)
--printbuffer(1, numPoints, drain.nvbuffer1.readings)
--printbuffer(1, numPoints, drain.nvbuffer1.timestamps)
```