GAA CUSTOM ELECTRONICS

MIPS Pulse Sequence Generation

Application Note — Worked Examples

MIPS Modular Instrument Platform System • Rev A

This note is a hands-on companion to the MIPS Pulse Sequence Generation reference. It walks through complete, copy-and-paste examples that build from a single digital pulse up to multi-segment voltage ramps. Every example is a full sequence that can be loaded and run as written. If you are new to MIPS pulse sequences, or a sequence is not behaving the way you expect, work through the examples in order and then consult the troubleshooting section near the end.

Companion document: MIPS Pulse Sequence Generation (reference). That document defines the full event grammar, the complete channel table, and every table command. This note assumes it is available for lookup and does not repeat it in full.

Contents

1 About this application note

Researchers new to the MIPS pulse sequence generator often get stuck not on the physics of what they want to produce, but on the exact string syntax and on the sequence of commands needed to load and run a table. This note is built to remove both obstacles.

Each example follows the same pattern:

- **Goal** — what waveform the sequence produces.
- **Sequence** — the exact string, ready to download.
- **How it works** — a field-by-field breakdown.

The examples are cumulative. Later examples reuse ideas introduced earlier, so if a construct looks unfamiliar, it was almost certainly explained in a previous example.

2 Anatomy of a pulse sequence

A pulse sequence (also called a table) is a comma-separated list of events. Each event sets one or more MIPS outputs at a specific time. The most basic event has the form:

```
Count:Channel:Value
```

Count is the time of the event, expressed in *clock cycles* after the table starts. **Channel** is the output to set (a digital line A–P, a DC bias channel 1–32, or one of the special channels listed in the reference). **Value** is what to set that channel to.

Several channel:value pairs can share one time point by chaining them, and several events are joined with commas:

```
10:B:1                one event: set digital output B high at count 10
25:B:0:5:34.5         one event: at count 25, set B low AND DC bias ch 5 to
34.5 V
10:B:1,25:B:0:5:34.5  two events forming a short sequence
```

Loops

Events are almost always wrapped in a loop so the pattern repeats. A loop is enclosed in square brackets:

```
[name:cycles,event1,event2,...,length:]
```

- `name` — a single character naming the loop.
- `cycles` — how many times to repeat (0 = repeat forever).
- `length` — the period of one pass, in clock cycles. This must be at least as large as the last event count inside the loop.

Finally, a leading count before the loop sets when the loop begins, relative to the trigger. A complete, runnable table therefore looks like this:

```
0:[A:10,10:B:1,25:B:0,100:]
```

> **Loop name vs. digital output — a common confusion**
> The single character immediately after `[` is the **loop name**. The same letter used later, in an event, is a **digital output**. They are unrelated; MIPS tells them apart by position. In `[A:10,10:B:1...]` the loop is named A while the pulses drive output B. To avoid tripping over this, the early examples in this note deliberately name the loop A and pulse output B.

Clock cycles and real time

All counts are in clock cycles, so the real-time meaning of a count depends on the table clock you select with STBLCLK. Every timed example in this note assumes the `656250 Hz` internal clock, the same clock used in the ramp examples. At that rate:

```
1 clock cycle   = 1 / 656250       ≈ 1.524 µs
656 clock cycles= 656 / 656250     ≈ 1.000 ms
count × 1.524 µs = elapsed time
```

To scale any example to a different clock, keep the counts and change the clock, or keep the clock and scale the counts. Other internal clock options (`42000000, 10500000, 2625000, 656250`) and external clock sources are listed in the reference under STBLCLK.

3 Loading and running a sequence

Every example below is downloaded and run the same way. Rather than repeat these steps for each example, they are given once here; later examples simply reference this section and note any extra enable command they require.

MIPS has two operating modes, set with the SMOD command: local mode (LOC), where you configure and download, and table mode (TBL), where a table is armed and runs. The full minimum workflow is:

Minimum load-and-run sequence

```
SMOD,LOC                ← be in local mode to load
STBLDAT;<sequence>;     ← download the table (semicolon before and after)
STBLCLK,656250          ← select the table clock
STBLTRG,SW              ← select the trigger source (software here)
SMOD,TBL                ← arm the table; MIPS replies TBLRDY
TBLSTRT                 ← software trigger: run it
```

A few points that trip people up:

1. The table string follows STBLDAT, separated by a semicolon, and the whole command is also terminated by a semicolon: `STBLDAT;0:[A:10,...,100:];` Both semicolons are required.

2. Remove every space from the final string before sending it. Spaces shown in this note for readability are not part of the sequence.

3. You must be in LOC mode to download. If you are already running a table, abort it first (TBLABRT, or press the front-panel knob).

4. If you paste STBLDAT manually, complete the paste within about 3 seconds; MIPS times out a partially received table.

5. Ramp examples need one extra enable command, and it must be sent *before* the sequence is downloaded with STBLDAT — either `STBLVDLT,TRUE` (conditional-logic ramps) or `STBLRMPENA,TRUE,<freq>` (table-based ramps). Each example says which.

> **Trigger and clock are yours to choose**
> The examples use `STBLTRG,SW` with TBLSTRT so you can fire the table from the keyboard. To trigger from hardware, set `STBLTRG,POS` / `NEG` / `EDGE` instead and apply your signal to the R input. Clock and trigger settings persist across downloads.

4 Example 1 — A single digital pulse

Goal: produce one pulse on digital output B — high, then low — and stop.

```
STBLDAT;0:[A:1,100:B:1,300:B:0,400:];
```

How it works, field by field:

- `0:` start the loop at count 0 (immediately on trigger).
- `[A:1` a loop named A that runs once.
- `100:B:1` at count 100 (≈152 µs), set output B high.
- `300:B:0` at count 300 (≈457 µs), set output B low. The pulse is 200 clocks ≈ 305 µs wide.
- `400:]` the loop length is 400 clocks ≈ 610 µs. It must be ≥ the last event count (300).

Load and run it exactly as in Section 3. Because the loop runs a single cycle, you get one pulse.

5 Example 2 — Repeating a pulse with a loop

Goal: produce the same pulse, repeated 10 times.

```
STBLDAT;0:[A:10,100:B:1,300:B:0,400:];
```

Only one field changed: `A:1` became `A:10`. The loop now repeats 10 times. Each pass lasts `length` = 400 clocks ≈ 610 µs, so the pulse repeats every ≈610 µs and the whole table lasts ≈10 × 610 µs = 6.1 ms.

> **The length parameter sets the period, not just the end**
> A frequent mistake is setting `length` equal to the last event and wondering why the timing is off. `length` is the full period of one pass. The gap between the last event of one pass and the first event of the next equals `length` minus the last event count. Here that gap is 400 − 300 = 100 clocks, so B stays low for ≈152 µs between pulses.

To repeat forever, set the cycle count to 0: `0:[A:0,100:B:1,300:B:0,400:];` Stop it with TBLSTOP or TBLABRT.

6 Example 3 — Digital and DC bias together

Goal: each cycle, pulse output B high while stepping DC bias channel 3 to +30 V, then return both to their low state.

```
STBLDAT;0:[A:5,100:B:1:3:30,300:B:0:3:0,600:];
```

The new idea is chaining channel:value pairs within one event:

- `100:B:1:3:30` at count 100, set output B high *and* DC bias channel 3 to 30 V, in the same event.
- `300:B:0:3:0` at count 300, set B low and channel 3 back to 0 V.

- `A:5 ... 600:` repeat 5 times with a 600-clock (≈914 µs) period.

> **Watch the DC bias range and the update budget**
> The valid voltage range for a DC bias channel depends on the module installed — MIPS does not range-check it for you. Also note that two values change at each event here. As a rule of thumb, allow about 20 µs of settling per value that changes at a time point (see Section 13); with two values changing, keep events at least ≈40 µs apart.

7 Example 4 — Sweeping a pulse in time

Goal: step the pulse later in time on every pass through the loop — a time sweep — using dynamic time points.

```
STBLDAT;0:[A:10,-100:B:1,-200:B:0,400:d:5:];
```

Two things are new. First, the event counts are *negative* (`-100`, `-200`). A negative count flags a time point as **adjustable**; its magnitude is the actual starting time. Second, the special `d` channel sets a time delta — here `d:5` — that is added to every adjustable time point after each pass.

So on the first pass the rising edge sits at count 100 and the falling edge at count 200. On each subsequent pass, both edges move 5 clocks later, walking the pulse progressively later in time across the 10 iterations while its width stays constant.

> **Rules for adjustable time points**
> Keep all time points within a loop in ascending order, and make sure the sweep never lets two events collide or cross. The `d` channel adds a fixed increment each pass; the related `p` channel applies a fixed offset. Both are detailed in the reference.

8 Example 5 — A voltage ramp with conditional logic

The remaining examples build voltage ramps. MIPS offers two ramping mechanisms; this section and the next two use the conditional-logic method, which is flexible and can drive up to eight ramps at once. Section 12 shows the simpler table-based alternative.

Conditional logic in the P loop

A loop named P is a *conditional* loop: an event can be gated by a test on the loop counter. Three tests are available — equal `=`, greater-than `>`, and less-than `<`. For example, on its own a conditional loop can place pulses at chosen iterations:

```
[P:100,0:=:20:4:12.5:=:25:4:-10,656:]
```

This runs 100 times with a per-pass length of 656 clocks (≈1 ms). `=:20:4:12.5` sets DC bias channel 4 to 12.5 V on pass 20; `=:25:4:-10` sets it to −10 V on pass 25 — a 5-pass (≈5 ms) pulse. Note that conditional loops act on DC bias channels only.

Turning conditional logic into a ramp

A ramp is just a conditional loop that adds a small voltage step to a channel every pass. To do this, two flag bits are added to the channel number, and MIPS is told to prepare its ramp data structures with `STBLVDLT,TRUE` (send once).

Flag	Value	Meaning
INITIAL	`0x40 = 64`	Marks the one-time initialization event that sets the ramp's starting voltage.

Flag	Value	Meaning
RAMP	`0x80 = 128`	Marks an event whose value is added to the channel each pass (the ramp step).

Add the flag to the channel number. For channel 1: `1 + 64 = 65` is “initialize channel 1”, and `1 + 128 = 129` is “ramp channel 1”.

Goal: a single 100 ms ramp from 0 V to 100 V on DC bias channel 1.

Enable, then download

```
STBLVDLT,TRUE            ← send once, enables conditional ramping
STBLDAT;0:65:0:[P:100,0:129:1,656:];
```

Breaking down the table string:

- `0:65:0` at count 0, set channel 65 (channel 1 + INITIAL) to 0 V. This initialization runs once, before the loop.
- `[P:100` a conditional loop that runs 100 times.
- `0:129:1` on every pass, add 1 V to channel 1 (channel 129 = channel 1 + RAMP).
- `656:]` each pass is 656 clocks ≈ 1 ms, so 100 passes ≈ 100 ms.

Result: 100 steps of 1 V over 100 ms — a 0 V to 100 V ramp. The ramp rate is set by the loop length: a shorter length ramps faster, a larger step ramps steeper.

9 Example 6 — A two-segment ramp

Goal: a ramp that rises to 100 V, holds, then falls back to 0 V — using the > and < tests to change the step mid-loop.

```
STBLDAT;0:65:0:[P:100,0:<:50:129:2:>:75:129:-4,656:];
```

Two conditional steps now share the loop:

- `<:50:129:2` while the pass counter is under 50, add 2 V per pass. Over passes 0–49 that is ≈100 V, so the output climbs to 100 V in ≈50 ms.
- *(passes 50–75)* neither test fires, so the voltage holds for ≈25 ms.
- `>:75:129:-4` once the counter is over 75, subtract 4 V per pass. Over passes 76–100 that is ≈−100 V, returning to 0 V in ≈25 ms.

The net shape is: ramp up (50 ms) → hold (25 ms) → ramp down (25 ms). Remember to send STBLVDLT,TRUE once before loading, exactly as in Example 5.

> **One test gates one pair**
> A conditional test applies only to the single channel:value pair that follows it. Any further pairs in the same event are applied unconditionally. Repeat the test if you need it on more than one pair — the reference shows the exact form.

10 Example 7 — Releasing the channel and repeating the ramp

Two finishing touches make the two-segment ramp production-ready: releasing the ramp channel when the loop ends, and repeating the whole ramp several times.

Release the ramp channel

MIPS holds a ramping channel in one of its eight internal ramp slots until you release it. Release it by setting *both* flags at once — RAMP + INITIAL added to the channel: for channel 1, `1 + 128 + 64 = 193`. If you skip this, the slot stays occupied.

```
STBLDAT;0:65:0:[P:100,0:<:50:129:2:>:75:129:-4,656:],100:193:0;
```

After the loop closes, `,100:193:0` waits 100 clocks and then sets channel 193 (channel 1 + RAMP + INITIAL) to 0, freeing the slot.

Repeat the whole ramp

Nest the ramp inside an ordinary loop to repeat it. The table below runs the released two-segment ramp three times (`A:3`):

```
STBLDAT;0:[A:3,0:65:0:[P:100,0:<:50:129:2:>:75:129:-4,656:],100:193:0];
```

The outer loop A repeats the initialize-ramp-release unit three times, then execution continues with whatever follows. The oscilloscope capture below shows the three repeated ramps this sequence produces on channel 1.

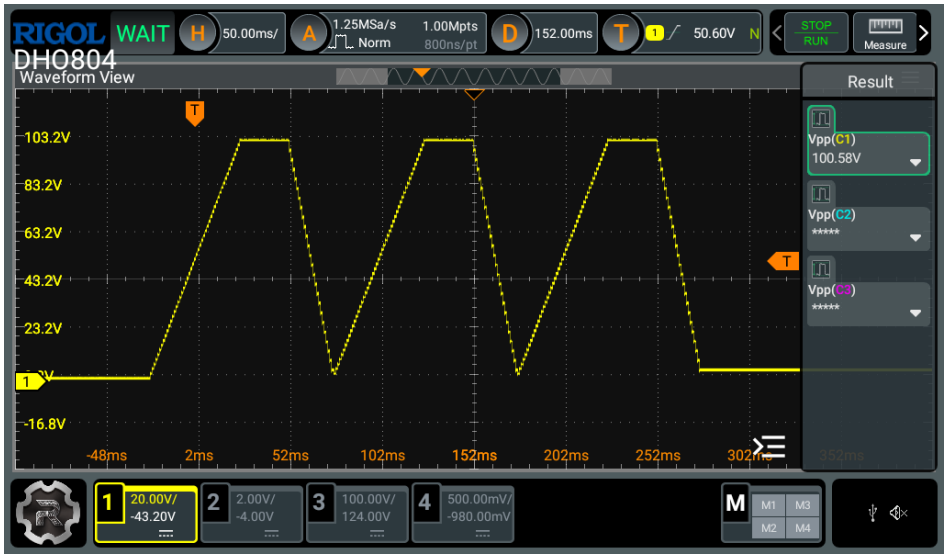

Figure 1. Three repeated two-segment ramps produced by the Example 7 sequence.

> **Housekeeping for ramps**
> Always release each ramp channel you use, and remember there are only eight conditional-ramp slots. Up to eight channels can ramp inside one conditional loop; give each its own initialize and release.

11 Example 8 — A digital output pulse after the ramps

Ramp mode has one restriction worth knowing: while conditional ramping is enabled (Examples 5–7) the processor only services DC bias ramp channels, and digital outputs are not updated. This example fires a digital pulse after the three ramps of Example 7 finish, using the u channel to step briefly out of ramp mode.

Goal: extend the Example 7 sequence with a pulse on digital output A once the three ramps complete.

Send STBLVDLT,TRUE first, then download with STBLDAT as in Section 3.

```
0:[A:3,0:65:0:[P:100,0:<:50:129:2:>:75:129:-4,656:],100:193:0],100:u:0:A:1,200:A:0:u:1
```

The part up to the outer loop's closing bracket is the Example 7 sequence, unchanged. Two events after the loop add the pulse:

- `100:u:0:A:1` — 100 clocks (≈152 µs) after the ramps finish, `u:0` turns ramp processing off and `A:1` sets digital output A high.
- `200:A:0:u:1` — at 200 clocks, `A:0` sets output A low and `u:1` turns ramp processing back on.

The result is a pulse on output A, about 100 clocks (≈152 µs) wide, that fires once the three ramps have completed.

> **The u channel — stepping out of ramp mode**
> `u:0` disables conditional ramp processing and `u:1` re-enables it. Use it whenever you need a digital output, a trigger, or another non-DC bias command while a conditional ramp is active, and always pair them — `u:0` before the events and `u:1` after. Without it, those commands are ignored while ramping.

12 Example 9 — Table-based ramping (no conditional logic)

For simple ramps you may prefer the table-based ramp mechanism. It does not use a conditional loop; instead you set a ramp update frequency once, then place ramp events directly on the time line. It supports up to three ramps in a sequence.

Enable it and set the update rate (here 2 kHz) with `STBLRMPENA,TRUE,2000`. In this mode only two flag combinations are valid:

Flag combination	Value	Meaning
RAMP | INITIAL	`0xC0 = 192`	Initialize the channel and set the starting voltage. Occurs once per sequence, and must appear first.
RAMP	`0x80 = 128`	Set the ramp step (delta) applied at each update. Change it any time; set it to 0 to stop the ramp.

Goal: on DC bias channel 2, ramp up, reverse, then stop — repeated four times.

Enable, then download

```
STBLRMPENA,TRUE,2000     ← send once, 2 kHz update rate
STBLDAT;0:[A:4,0:194:1:130:4,5000:130:-1.0,10000:130:0,20000:];
```

Field by field (channel 2 has flag values 194 = 2+192 and 130 = 2+128):

- `0:194:1:130:4` at count 0, initialize channel 2 to 1 V, then set its ramp step to +4 V per update. The INITIAL event comes first.
- `5000:130:-1.0` at count 5000, change the step to −1 V per update, reversing the ramp.
- `10000:130:0` at count 10000, set the step to 0, stopping the ramp.
- `A:4 ... 20000:` repeat 4 times with a 20000-clock period.

> **Which ramp method should I use?**
> Use **conditional-logic ramps** (Sections 8–10) when you need many ramps, per-iteration control, or ramps interleaved with other conditional events. Use **table-based ramps** (this section) for one to three straightforward ramps where placing a few delta changes on the time line is simpler. Do not mix the two enable commands in the same table.

13 Common problems and how to fix them

If a sequence will not load, will not run, or produces the wrong output, check these first — they account for the large majority of the trouble researchers hit.

The table will not download / STBLDAT is rejected

You are not in local mode, the string contains spaces, or a semicolon is missing. The command is STBLDAT;<sequence>; — a semicolon after STBLDAT and another to terminate. Send SMOD,LOC, strip all spaces, and check both semicolons. If pasting by hand, finish within ~3 seconds.

A conditional ramp does not ramp

STBLVDLT,TRUE was not sent, or was sent after the table. It enables the ramp data structures for conditional loops, must be sent once (required after power-up), and must be issued before the sequence is downloaded with STBLDAT.

A digital output does nothing during a ramp

While conditional ramping is enabled only DC bias ramp channels are serviced, so digital outputs are ignored. Step out of ramp mode around the event with the u channel: u:0 before the digital output command and u:1 after (see Example 8).

A table-based ramp does not ramp

STBLRMPENA,TRUE,<freq> was not sent, or the INITIAL (192) event is missing or not first. The RAMP|INITIAL event must appear once and before any plain RAMP event.

A ramp channel is stuck or unavailable on the next run

You did not release it. Release conditional ramps by setting RAMP+INITIAL (e.g. channel 1 → 193) to 0 after the loop. There are only eight conditional ramp slots and three table-based ramps.

The voltages or channels are wrong

Re-check the flag arithmetic: add the flag to the channel number. INITIAL = 64, RAMP = 128, both = 192. Channel 1 → 65 / 129 / 193. Table-based mode accepts only 128 and 192.

Timing is off, or the output overruns / misfires

Events are too close together. The minimum interval is 10 µs; as a safe start, multiply the number of values changing at a time point by 20 µs. Begin with conservative deltas and tighten them only after the sequence is debugged.

Only part of the sequence plays, or it loops oddly

Time points inside a loop must be in ascending order, and the loop length must be ≥ the last event count. A length shorter than the last event truncates the pass.

The front panel is frozen

The system is in table mode (TBL). This is normal while a table is armed or running. Press the front-panel control knob, or send TBLABRT, to return to local mode.

“Loop name” and a digital output collide in my head

The character right after [is the loop name; the same letter in an event is digital output A–P. They are independent. Naming loops with letters you are not driving as outputs avoids the confusion.

> **When in doubt, use the host application**
>
> The MIPS host application timing-generator control builds sequences with single-level looping and dynamic time points for you, and handles clock, trigger, and millisecond time points through the interface. It covers most needs; hand-written tables are only necessary for the advanced features shown in this note.

14 Quick reference

Ramp flags (add to the channel number)

Purpose	Flag math	Channel 1	Channel 2
Initialize (set start V)	+64	65	66
Ramp step (conditional or table)	+128	129	130
Release / init table ramp (both)	+192	193	194

Key commands

Command	Purpose
SMOD,LOC / SMOD,TBL	Enter local (configure/load) or table (armed/run) mode.
STBLDAT;<seq>;	Download the pulse sequence (semicolon before and after, no spaces).
STBLCLK,<src>	Select the table clock (e.g. 656250, or an external source).
STBLTRG,<src>	Select the trigger: SW, POS, NEG, or EDGE.
TBLSTRT	Software trigger — run a ready table.
TBLSTOP / TBLABRT	Gracefully stop / forcefully abort a running table.
STBLVDLT,TRUE	Enable conditional-logic ramping (Sections 8–10). Send once.
STBLRMPENA,TRUE,<freq>	Enable table-based ramping at <freq> Hz (Section 12). Send once.
GTBLSTA	Report table status: IDLE / READY / TRIGGERED / ABORTED.

Timing at the 656250 Hz clock

Clock cycles	Elapsed time
1	≈ 1.524 µs
66	≈ 100 µs
656	≈ 1.000 ms
65625	≈ 100 ms

For the complete event grammar, the full channel table, dynamic time-point details, and every table command, see the companion reference, MIPS Pulse Sequence Generation.