Thanks Danilo for the information about enabling semihosting. It seems that is a non-starter. Same goes for the profiling code. The reason is that for power profiling purposes, I am not interested in the execution time, but instead in the power consumed by a given scope of code being executed, and I need to match up entry/exit of multiple code blocks using timestamps that I later align to my power monitor's captured measurements. This won't be possible with a screen indicator, and I suspect enabling semihosting would massively interfere with „normal“ power consumption measurements if it is slowing down the processor all the time.

Maybe this will have to wait until we have ability to write data to a file for retrieval later. It'd be a simple annotation printed out, just „Timestamp XXXXXXXX enter function foo::bar“, „Timestamp XXXXXXXX exit function foo::bar“, or even just a unique ID and a timestamp, or unique ID and system clock, whatever is fast and could be converted back into an approximate timestamp log later. Flag only entry/exit for the highest level routines to start, find the ones that unexpectedly eat a lot of power compared to the value they provide. Start annotating further and further down into their functions until we start turning up sections that eat more power than we would expect („expect“ is arguable and an art of sorts).

I would most likely use Google's GitHub project battery-historian, which is trivial to repurpose for non-Android profiling and accepts Monsoon logs. It makes hunting for power hogs easy. Check out the first screenshot on the project page.
https://github.com/google/battery-historian