Exercise no.: 2

Clock tree

Objectives

1. Using the internal frequency counter of the RP2350 MCU, measure and write to the terminal frequencies of the following internal clocks: system PLL, USB PLL, ROSC, *clk_sys, clk_peri, clk_usb, clk_adc*. Do this once just after startup and initialization. Can you tell what all these clocks are used for?
2. Using *clock_configure()* function make sure that *clk_ref* and *clk_peri* are connected to the external 12 MHz source. *clk_peri* must operate with a stable frequency to prevent peripheral malfunctions during PLL reconfiguration. Remember to reinitialize peripherals after switching *clk_peri* by calling *stdio_init_all()* once again.
3. In the main loop receive user input, accepting a single integer with a value from 0 to 3. When 0 is received, switch the system PLL off completely and run the core from *clk_ref*. For values 1 to 3 switch between 3 significantly different system PLL settings and then switch the core to the new frequency. Make sure your serial communication remains coherent and no glitches occur.
4. Each time the system frequency is modified use the internal frequency counter to measure and print to the terminal frequencies of the system PLL output and *clk_sys*.

Description

The aim of this task is to make you familiar with the clock tree of the RP2350 MCU. Clock tree is the popular way to call the clock distribution system inside of the chip. It is comprised of switches (multiplexers or MUXes), sources, PLLs, dividers and receivers. Different branches of this tree can carry different frequencies depending on different requirement of the receivers (for example USB always needs 48 MHz), energy consumption constraints, EMI emission constraints, etc. In this exercise you will learn how to dynamically switch the frequency of your core without disrupting peripherals, which is useful for optimizing energy consumption of your system depending on the current demand on computing power. RP2350 has an internal frequency counter referenced to *clk_ref*, which can be connected to all internal clock rails and sources to measure their actual working frequencies.

Hints

1. Getting CYW43 to reinitialize after switching the clocks doesn't seem to work very well. Don't use it as the user LED isn't needed anyway.
2. API documentation for clock-related functions isn't really sufficient. Make sure to analyze the source code of SDK functions (it contains comments) and SDK examples (which also contain comments) to get a good understating how to use them.
3. Most RP2350 clocks have primary and auxiliary (aux) switches. Glitch-less switching is possible only between the primary ones, but almost always among those primary clock options one will be the selected aux clock.
   So to guarantee a glitch-less switch, one must first switch the primary source to *clk_ref* or another stable clock, then switch the aux source and only then connect the primary switch back to the selected aux. In this way the system will remain operable for the entire duration of the process. Allowing glitches would make the system's behavior unpredictable. *clock_configure()* function of the SDK automates this process to guarantee a safe switch. Make sure to analyze its code as well as the RP2350 datasheet for more information.
4. RP2350 Datasheet contains not only the explanation of the clock tree functionality but also some useful code snippets.
5. User input can be received from the terminal using *scanf()* but it's not the only way.
6. Correct system PLL settings can be a bit tricky to figure out based on the datasheet. It can, of course, be done, but the SDK comes with a special Python script *vcocalc.py* located in:
   <SDK location>\.pico-sdk\sdk\2.1.1\src\rp2_common\hardware_clocks\scripts
   Which can be used from the terminal like that:
   python .\vcocalc.py N
   where N stands for the desired PLL output frequency. The script will then print correct settings for the PLL (if a solution exists for the desired frequency).