1. Why is bit-banging impractical on your laptop, despite it having a much faster processor than the RP2040?

Bit Banging is impractical for our laptop as the processor is not designed for it. In bitbanging, the processor tries really hard to fit the GPIO reading and writing in the same cycle which is extremely difficult. And during this if the processor is disturbed with other task, this could harm the whole process.

2. What are some cases where directly using the GPIO might be a better choice than using the PIO hardware?

The cases where directly using the GPIO might be a better choice than using the PIO hardware includes accessing external peripherals like LED's or some low priority tasks. Then we need to toggle GPIOs via hardware interrupts which are less set of instructions than using PIO. PIO will be a better choice than using GPIO for complex tasks.

3. How do you get data into a PIO state machine?

TX FIFO takes the data from the system and passes it to OSR. Now the data is shifted to any of the registers. PULL and OUT instructions can be used.

4. How do you get data out of a PIO state machine?

Data is shifted to ISR and then to RX FIFO which passes on the data to the system. PUSH and IN instructions are used.

5. How do you program a PIO state machine?

In the PIO library the programs are included for UART, I2c ,etc that is for common interfaces so we don't have to write a program for these. The PIO has nine instructions in total mentioned below:

1. JMP
2. WAIT
3. PUSH
4. PULL
5. IN
6. OUT
7. SET
8. IRQ
9. MOV

The textual format describing a PIO program is the PIO assembly. Here each command has one instruction in the output.

PIO state machines execute short, binary programs. Programs for common interfaces, such as UART, SPI, or I2C, are available in the PIO library, so in many cases, it is not necessary to write PIO programs. However, the PIO is much more flexible when programmed directly, supporting a wide variety of interfaces which may not have been foreseen by its designers. The PIO has a total of nine instructions: JMP, WAIT, IN, OUT, PUSH, PULL, MOV, IRQ, and SET. Though the PIO only has a total of nine instructions, it would be difficult to edit PIO program binaries by hand. PIO assembly is a textual format, describing a PIO program, where each command corresponds to one instruction in the output binary.

6. In the example, which low-level C SDK function is directly responsible for telling the PIO to set the LED to a new color? How is this function accessed from the main "application" code?

The low-level C SDK function that is directly responsible for telling the PIO to set the LED to a new color is pio_sm_put_blocking() function. As using this we can directly push data into state machine's TX FIFO as when the TX FIFO is full it stalls the processor this leds to LED being turned on and off when writing 1 and 0 respectively. This function is accessed form the main application code using put_pixel() function. As given in the example code we are calling put_pixel() function.

7. What role does the pioasm "assembler" play in the example, and how does this interact with CMake?

Pioasm: it is the PIO assembler included in the SDK. The role it plays in the example is as follows: It processes an input text file of PIO assembly that might contain multiple programs and it writes out the ready-to-use assembled programs. These programs containing constant arrays are emitted in the form of C headers. It interacts with the CMake using the pico_generate_pio_header(TARGET PIO_FILE) function as it invokes pioasm and the header is also added in the include path of the target using this. Therefore, we do not have to invoke pioasm in the SDK directly.