<u>ARM</u>

F(A,B,C) = (A + B + !C) . (A + !B + !C).(!A + B + C).(!A + !B + !C)

A	B	C	
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

(!A + !B + C).(!A + B +!C).(!A + B + !C).(!A + !B + !C) led will glow

code:

```
#include "Fw_global_config.h"
#include <stdio.h>
#include "FreeRTOS.h"
#include "task.h"
#include "semphr.h"
#include "timers.h"
#include "RtosTask.h"

/* Include the generic headers required for QORC */
#include "eoss3_hal_gpio.h"
#include "eoss3_hal_rtc.h"
#include "eoss3_hal_timer.h"
#include "eoss3_hal_fpga_usbserial.h"
#include "ql_time.h"
#include "s3x_clock_hal.h"
#include "s3x_clock.h"
#include "s3x_pi.h"
#include "dbg_uart.h"

#include "cli.h"

extern const struct cli_cmd_entry my_main_menu[];

const char *SOFTWARE_VERSION_STR;

extern void qf_hardwareSetup();
static void nvic_init(void);
```

```
#define GPIO_OUTPUT_MODE (1)
#define GPIO_INPUT_MODE (0)
void PyHal_GPIO_SetDir(uint8_t gpionum, uint8_t iomode);
int PyHal_GPIO_GetDir(uint8_t gpionum);
int PyHal_GPIO_Set(uint8_t gpionum, uint8_t gpioval);
int PyHal_GPIO_Get(uint8_t gpionum);

void setup() {
   SOFTWARE_VERSION_STR = "qorc-onion-apps/qf_hello-fpga-gpio-ctlr";

   qf_hardwareSetup();
   nvic_init();

   dbg_str("\n\n");
   dbg_str("##########################\n");
   dbg_str("Quicklogic QuickFeather FPGA GPIO CONTROLLER EXAMPLE\n");
   dbg_str("SW Version: ");
   dbg_str(SOFTWARE_VERSION_STR);
   dbg_str("\n");
   dbg_str(__DATE__ " " __TIME__ "\n");
   dbg_str("##########################\n\n");

   dbg_str("\n\nHello GPIO!!\n\n");

   CLI_start_task(my_main_menu);
   HAL_Delay_Init();

   // Set up your inputs A, B, and C as digital inputs
   PyHal_GPIO_SetDir(2, GPIO_INPUT_MODE); // A
   PyHal_GPIO_SetDir(3, GPIO_INPUT_MODE); // B
   PyHal_GPIO_SetDir(4, GPIO_INPUT_MODE); // C

   // Set up your output as a digital output
   PyHal_GPIO_SetDir(13, GPIO_OUTPUT_MODE); // Output LED

   while (1) {
      // Read the inputs
      bool A = PyHal_GPIO_Get(2);
      bool B = PyHal_GPIO_Get(3);
      bool C = PyHal_GPIO_Get(4);

      // Compute the Boolean function
      bool result = (!A && !B && C) || (!A && B && !C) || (A && !B && C) || (A && B && C);

      // Set the output LED based on the result of the Boolean function
      PyHal_GPIO_Set(13, result ? 1 : 0);

      vTaskDelay(pdMS_TO_TICKS(1000)); // Delay for 1 second
   }
}
```

```
static void nvic_init(void)
 {
    // To initialize system, this interrupt should be triggered at main.
    // So, we will set its priority just before calling vTaskStartScheduler(), not the time of enabling
each irq.
    NVIC_SetPriority(Ffe0_IRQn, configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY);
    NVIC_SetPriority(SpiMs_IRQn, configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY);
    NVIC_SetPriority(CfgDma_IRQn, configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY);
    NVIC_SetPriority(Uart_IRQn, configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY);
    NVIC_SetPriority(FbMsg_IRQn, configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY);
 }


//needed for startup_EOSS3b.s asm file
void SystemInit(void)
{

}

//gpionum --> 0 --> 31 corresponding to the IO PADs
//gpioval --> 0 or 1
#define FGPIO_DIRECTION_REG (0x40024008)
#define FGPIO_OUTPUT_REG (0x40024004)
#define FGPIO_INPUT_REG (0x40024000)
//Set GPIO(=gpionum) Mode: Input(iomode = 0) or Output(iomode = 1)
//Before Set/Get GPIO value, the direction must be correctly set
void PyHal_GPIO_SetDir(uint8_t gpionum,uint8_t iomode)
{
    uint32_t tempscratch32;

    if (gpionum > 31)
        return;

    tempscratch32 = *(uint32_t*)(FGPIO_DIRECTION_REG);
    if(iomode)
        *(uint32_t*)(FGPIO_DIRECTION_REG) = tempscratch32 | (0x1 << gpionum);
    else
        *(uint32_t*)(FGPIO_DIRECTION_REG) = tempscratch32 & (~(0x1 << gpionum));

}


//Get current GPIO(=gpionum) Mode: Input(iomode = 0) or Output(iomode = 1)
int PyHal_GPIO_GetDir(uint8_t gpionum)
{
    uint32_t tempscratch32;
    int result = 0;

    if (gpionum > 31)
        return -1;
    tempscratch32 = *(uint32_t*)(FGPIO_DIRECTION_REG);

    result = ((tempscratch32 & (0x1 << gpionum)) ? GPIO_OUTPUT_MODE :
GPIO_INPUT_MODE);

    return result;
}

//Set GPIO(=gpionum) to 0 or 1 (= gpioval)
//The direction must be set as Output for this GPIO already
//Return value = 0, success OR -1 if error.
int PyHal_GPIO_Set(uint8_t gpionum, uint8_t gpioval)
{
    uint32_t tempscratch32;

    if (gpionum > 31)
        return -1;

    tempscratch32 = *(uint32_t*)(FGPIO_DIRECTION_REG);

    //Setting Direction moved out as separate API, we will only check
    //*(uint32_t*)(FGPIO_DIRECTION_REG) = tempscratch32 | (0x1 << gpionum);
    if (!(tempscratch32 & (0x1 << gpionum)))
    {
        //Direction not Set to Output
        return -1;
    }

    tempscratch32 = *(uint32_t*)(FGPIO_OUTPUT_REG);

    if(gpioval > 0)
    {
        *(uint32_t*)(FGPIO_OUTPUT_REG) = tempscratch32 | (0x1 << gpionum);
    }
    else
    {
        *(uint32_t*)(FGPIO_OUTPUT_REG) = tempscratch32 & ~(0x1 << gpionum);
    }

    return 0;
}
//Get GPIO(=gpionum): 0 or 1 returned (or in erros -1)
//The direction must be set as Input for this GPIO already
int PyHal_GPIO_Get(uint8_t gpionum)
{
    uint32_t tempscratch32;
    uint32_t gpioval_input;

    if (gpionum > 31)
        return -1;

    tempscratch32 = *(uint32_t*)(FGPIO_INPUT_REG);
    gpioval_input = (tempscratch32 >> gpionum) & 0x1;

    return ((int)gpioval_input);
}
```