IMPLEMENTATION OF RANDOM NUMBER GENERATOR USING D FLIP-FLOPS USING ARM(VAMAN)

M Patrick Manohar
patrickmanohar152001@gmail.com
FWC22119 IITH-Future Wireless Communications Assignment-4

Contents

1 Problem

(GATE2021-QP-EC)
Q.46 The propogation delay of the exclusive-OR(XOR) gate in the circuit in the figure is 3ns.The propogation delay of all the flip-flops is assumed to be zero.The clock(Clk) frequency provided to the circuit is 500MHz.

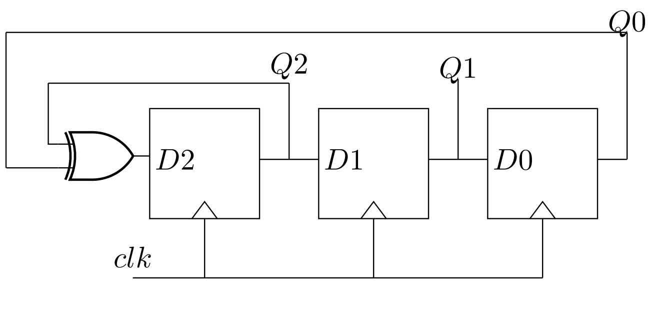

Figure 1: Circuit

Starting from the initial value of the flip-flop outputs $Q2Q1Q0 = 111$ with $D2 = 1$,the minimum number of triggering clock edges after which the flip-flop outputs $Q2Q1Q0$ becomes 1 0 0*(in integer)* is __

2 Introduction

A random number generator using D flip-flops is a simple digital circuit that generates a sequence of random binary numbers.To implement this type of random number generator, we use a series of D flip-flops connected in a feedback loop. The output of each flip-flop is fed back into the input of the next flip-flop,creating a circuit that generated a sequence of random binary values.

The feedback loop creates a delay in the circuit,which causes the circuit to exhibit unpredictable behavior.This unpredictable behavior results in a sequence of random binary values. The length of the delay can be adjusted to control the randomness of the output.

3 Components

Components	Value	Quantity
Breadboard		1
Resistor	$\geq 220\Omega$	1
Vaman	pygmy	1
Seven Segment Display	Common Anode	1
Decoder	7447	1
Flip Flop	7474	2
Jumper Wires		20

Table 1: Components

3.1 Vaman

The Vaman (pygmy) has some ground pins, digital pins that can be used for both input as well as output.It also has two power pins that can generate 3.3V. In the following exercises, we use digital pins,GND and 5V .

3.2 Seven Segment Display

The seven segment display has eight pins, *a,b,c,d,e,f,g* and *dot* that take an active LOW input,i.e. the LED will glow only if the input is connected to ground.Each of these pins is connected to an LED segment.The *dot* pin is reserved for the LED.

4 Implementation

A 7474 IC which has 14 pins and can store two seperate binary values.So we consider two IC's since we have three values and connect the D inputs of each flip-flop to the input signals of 7447 IC . Later interface 7447 IC to seven segment display for the output. The CLK input is used to trigger the flip-flop,and the Q output is used to read the stored value.When a positive edge is detected on the CLK input,the current value on the D input is stored in the flip-flop. The boolean expression of the D flip-flop is $Q(t+1) = D$

4.1 Truth table

Present State			Flip-Flop input			Next State		
Q2	Q1	Q0	D2	D1	D0	Q2'	Q1'	Q0'
1	1	1	0	1	1	0	1	1
0	1	1	1	0	1	1	0	1
1	0	1	0	1	0	0	1	0
0	1	0	0	0	1	0	0	1
0	0	1	1	0	0	1	0	0
1	0	0	1	1	0	1	1	0
1	1	0	1	1	1	1	1	1

Table 2: Truth Table

4.2 K-map

Since $Q' = D$,we find the k-maps for D as outputs

$Q0$ \ $Q2\ Q1$	00	01	11	10
0	0	0	1	1
1	1	1	0	0

Figure 2: For D2

$Q0$ \ $Q2\ Q1$	00	01	11	10
0	0	0	1	1
1	0	0	1	1

Figure 3: For D1

$Q0$ \ $Q2\ Q1$	00	01	11	10
0	0	1	1	0
1	0	1	1	0

Figure 4: For D0

4.3 Boolean Equation

By solving the K-maps above we obtain as follows :

$$D2 = \overline{Q2}Q0 + \overline{Q0}Q2 \tag{1}$$
$$D1 = Q2 \tag{2}$$
$$D0 = Q1 \tag{3}$$

5 Hardware

1. Make the connections between the seven segment display and the 7447 IC as shown in Table3

7447	$\overline{a}$	$\overline{b}$	$\overline{c}$	$\overline{d}$	$\overline{e}$	$\overline{f}$	$\overline{g}$
Display	a	b	c	d	e	f	g

Table 3: 7447

2. Connect the Vaman, 7447 and the two 7474 ICs according to Table4

	INPUT			OUTPUT			CLOCK		5V			
	Q0	Q1	Q2	Q0’	Q1’	Q2’						
Vaman	D6	D7	D8	D2	D3	D4	D13					
7474	5	9		2	12		CLK1	CLK2	1	4	10	13
7474			5			2	CLK1	CLK2	1	4	10	13
7447				7	1	2			16			

Table 4: Connections

3. Make the other D input pins of 7474 grounded and supply 3.3V and GND from the Vaman as well.
4. When the clock edge is trigerred we observe display of random numbers.

6 Software

Now write the following code and upload in Vaman to see the results.

```
#include "Fw_global_config.h"   // This defines application specific charactersitics

#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;


/*
 * Global variable definition
 */


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);

int main(void)
{
        uint32_t Q0,Q1,Q2;
        uint32_t D0,D1,D2;
    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");   // <<<<<<<<<<<<<<<<<<<<<  Change me!

    CLI_start_task( my_main_menu );
        HAL_Delay_Init();

    PyHal_GPIO_SetDir(14,1);
    PyHal_GPIO_SetDir(15,1);
    PyHal_GPIO_SetDir(16,1);
    PyHal_GPIO_SetDir(13,1);
    PyHal_GPIO_SetDir(18,0);
    PyHal_GPIO_SetDir(19,0);
    PyHal_GPIO_SetDir(20,0);

    while(1){
            PyHal_GPIO_Set(13,1);
            HAL_DelayUSec(1000000);
            Q0=PyHal_GPIO_Get(18);
            Q1=PyHal_GPIO_Get(19);
            Q2=PyHal_GPIO_Get(20);
            D2=(Q2&&!Q0)||(Q0&&!Q2);
            D1=Q2;
            D0=Q1;
            PyHal_GPIO_Set(14,D0);
            PyHal_GPIO_Set(15,D1);
            PyHal_GPIO_Set(16,D2);
            PyHal_GPIO_Set(13,0);
    }

    /* Start the tasks and timer running. */
    vTaskStartScheduler();
    dbg_str("\n");

    while(1);
}

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;
```

```c
    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);
}
```