Practical 1

目录

# Team

## B00125142 Violet Concordia

## Bernardo Borile

## Kevin

## Farouk

# Introduction

## A 75% duty cycle LED blinking program, running on an FPGA board.

# Specifications

## Clock freq

### 66MHz

### Operations per second 33M

## Frequency

### 2Hz

## Duty cycle

### 75%

## Off cycle

### 25%

#### 100% - Duty cycle

##### 100% - 75% = 25%

# Calculations

## Discussion

### To achieve delay, a loop will be used to keep the processor busy for a specific amount of time.

### PicoBlaze is 8bit

#### Registers only hold values ranging 0-255

##### 256 different values

###### Therefore we must nest loops to achieve delay higher than 256 computations duration.

##### If we want to loop for a single second, we would need to count up to 33 Million.

##### To circumvent this, nested loops will be utilised, which essentially provide us a way to keep track of up to 256^n per loop, n being the loop count.

###### The calculations below determined that nests of a minimum of 4 depth will be required for both the on and off cycles.

## Duration

### Seconds

#### Total duration per cycle

##### 2s

#### On duration

##### 1.5s

###### Total duration \* Duty cycle

2 \* 0.75 = 1.5s

#### Off duration

##### 0.5s

###### Total duration - On duration = Off duration

2 - 1.5 = 0.5s

### Cycles

#### Total operations per cycle

##### Operations per second \* On duration (seconds)

###### 33M \* 2 = 66M operations

#### On operations count

##### 49.5M operations

###### Total operations per cycle \* Duty cycle

66M \* 0.75 = 49.5M

#### Off operations count

##### 16.5M operations

###### Total operations per cycle \* Off cycle

66M \* 0.25 = 16.5M

# Code

## TypeScript

### Script developed to assist calculations

#### Language

##### TypeScript

#### Constants

##### export const BITS = 8; export const MAX\_INT = Math.pow(2, BITS); export const K = 1\_000; export const KK = 1\_000\*K; export const FREQ = 33\*KK; export const TOGGLE\_FREQ = 2; export const TOGGLE\_CYCLES = 2\*FREQ; export const DUTY\_CYCLE = 0.75; export const ON\_TIME\_S = TOGGLE\_FREQ\*DUTY\_CYCLE; export const ON\_CYCLES = ON\_TIME\_S\*FREQ; export const OFF\_TIME\_S = TOGGLE\_FREQ - ON\_TIME\_S; export const OFF\_CYCLES = OFF\_TIME\_S\*FREQ;

#### Functions

##### findBiggestDivider

###### /\*\* \* author: B00125142 Violet Concordia \* Brute force approach to find the highest divider without remainder \* \* @param input the number to divide \* @returns the divider without remainder \*/ export function findBiggestDivider(input: number, max: number = MAX\_INT): number { //Decrement loop `max` until either a no remainder result is found, or it reaches 0. for(; max > 0; --max) { if(input/max % 1 == 0) break; } //Return max, which is hopefully the highest number without a remainder. (Worst case scenario, it is 0) return max; }

##### findBiggestDividers

###### /\*\* \* author: B00125142 Violet Concordia \* Appends every highest no remainder divider to an array, until the input number is below the `max` value. \* \* @param input The number to divide. \* @returns The dividers array. \*/ export function findBiggestDividers(input: number, max: number = MAX\_INT): number[] { //Init array that will be returned. const dividers = []; //While input is higher than max number while(input >= max) { //Find biggest no remainder divider. const biggestDivider = findBiggestDivider(input, max); //Push the found value. dividers.push(biggestDivider); // console.log(`${input} / ${biggestDivider} = ${input / biggestDivider}`); //Divide by found value. input = input / biggestDivider; } //Push the remainder to result in a 1. dividers.push(input); //Return the dividers. return dividers; }

#### Results

##### Finding highest dividers

###### Code

console.log(`1s = ${FREQ} instructions`);
console.log(`On for ${ON\_TIME\_S}s, Off for ${OFF\_TIME\_S}s, total: ${TOGGLE\_FREQ}s`);
console.log(`On for ${ON\_CYCLES} cycles, Off for ${OFF\_CYCLES} cycles, total: ${TOGGLE\_CYCLES} cycles`);
const onDividers = findBiggestDividers(ON\_CYCLES);
console.log(`${("" + onDividers).replace(/,/g, " \* ")} = ${ON\_CYCLES}`);
const offDividers = findBiggestDividers(OFF\_CYCLES);
console.log(`${("" + offDividers).replace(/,/g, " \* ")} = ${OFF\_CYCLES}`);

###### Output

Raw

Debugger attached.
1s = 33000000 instructions
On for 1.5s, Off for 0.5s, total: 2s
On for 49500000 cycles, Off for 16500000 cycles, total: 66000000 cycles
49500000 / 250 = 198000
198000 / 250 = 792
792 / 198 = 4
250 \* 250 \* 198 \* 4 = 49500000
16500000 / 250 = 66000
66000 / 250 = 264
264 / 132 = 2
250 \* 250 \* 132 \* 2 = 16500000
Waiting for the debugger to disconnect...
harmony@harmony:/media/2TB\_Crucial\_SSD/repos/ratios/freq\_calc$

Information

On dividers

250 \* 250 \* 198 \* 4 = 49500000

Off dividers

250 \* 250 \* 132 \* 2 = 16500000

Both the off and on delays can be achieved by a minimum of 4 nested loop.

##### Example in JavaScript of what will be translated to assembly code for the PicoBlaze

###### Variables

onDividers = [250, 250, 198, 4]

###### Code

let count = 0;
for(let i0 = onDividers[0]; i0 > 0; --i0)
{
for(let i1 = onDividers[1]; i1 > 0; --i1)
{
for(let i2 = onDividers[2]; i2 > 0; --i2)
{
for(let i3 = onDividers[3]; i3 > 0; --i3)
{
++count;
}
}
}
}
console.log(count);

###### Output

49500000

As expected, the obtained dividers from the previous script have resulted in a nested loop of 49500000 operations, which is the amount we need to wait exactly 1.5 seconds, which is the on time for a duty cycle of 75% of 2 seconds.

## Assembly

## Vivado