

Лабораторна робота №3

Схеми ділення і підвищення частоти, таймінг аналіз, цифровий FM передавач

3.1 Практичне застосування досліджуваних схем	1
3.2 Теоретична частина	2
3.2.1 Подільник частоти на ціле число на базі лічильника	2
3.2.2 Параметри у мові Verilog	7
3.2.3 Оператори редукції у мові Verilog	9
3.2.4 Подільник частоти на ціле число на базі регістру зсуву	10
3.2.5 Подільник частоти на дробне число	12
3.2.6 Підвищення частоти за допомогою ФАПЧ (PLL)	13
3.2.7 Розрахунок максимально допустимої тактової частоти	15
3.2.8 Статичний аналіз затримок (Static Timing Analysis)	19
3.2.9 Знакове представлення чисел, доповняльний код	20
3.2.10 FM-передавач на базі генератора з цифровим керуванням	22
3.3 Практична частина	34
3.3.1 Створення блоку ФАПЧ (PLL) в Quartus Prime	34
3.3.2 Визначення обмежень на синтез (Synthesis Constraints) в Quartus Prime	39
3.3.3 Оцінка затримок і максимальної тактової частоти в TimeQuest	41
3.3.4 Передача аудіо файлів в цифровий FM передавач за допомогою UART	47
3.3.5 Завдання на лабораторну роботу	48
3.4 Контрольні запитання	49
3.5 Перелік посилань	51

3.1 Практичне застосування досліджуваних схем

В цифрових мікросхемах часто необхідно виконувати однотипні операції з певною періодичністю. Наприклад, зчитувати цифрові коди з АЦП та видавати цифрові коди на ЦАП з заданою частотою дискретизації. Або зчитувати значення з цифрового входу RX і виставляти значення на цифровий вихід TX в UART інтерфейсі.

Періодичного виконання подій можна досягти діленням частоти вхідного сигналу синхронізації на різні значення і виконуючи необхідні операції з прив'язкою до одержаних імпульсів. Такий підхід дозволить мати лише один сигнал синхронізації для всіх тригерів (що приведе до спрощення розробки) і при цьому отримати кілька періодичних сигналів з прив'язкою до яких можна виконувати різні події з різною періодичністю.

Часто подільники частоти на довільне ціле число реалізують за допомогою двійкового лічильника (binary counter). Зменшити затримки подільника частоти можна реалізувавши подільник на базі лічильника у позиційному коді (one-hot counter) з використанням регістру зсуву. Розглянутий у попередній лабораторній роботі генератор з цифровим керуванням (NCO) можна використовувати для ділення частоти на дробне число.

Схему з фазовим автопідстроюванням частоти (ФАПЧ, PLL) часто використовують для отримання сигналу синхронізації високої частоти у кілька сотень мегагерц з вхідного сигналу синхронізації, що має частоту кілька десятків мегагерц і подається у мікросхему від кварцевого генератора. Важливою особливістю PLL є можливість задати довільний зсув фаз між вхідним і вихідним періодичними імпульсними сигналами.

PLL можна використовувати і для зниження частоти, однак для підвищення частоти PLL використовують частіше, оскільки сучасні цифрові мікросхеми функціонують на високих частотах, які складно отримати за допомогою кварцевого генератора і ще складніше передати без спотворень друкованою платою від кварцевого генератора до мікросхеми.

Кожна цифрова мікросхема має максимально допустиме значення сигналу синхронізації F_{max} при якому мікросхема все ще стабільно працює. Подальше збільшення частоти сигналу синхронізації призводить до помилок в роботі мікросхеми.

Дуже важливо вміти розраховувати максимальне значення тактової частоти і розуміти, як необхідно змінити схему для збільшення максимальної тактової частоти. Для оцінки затримок і максимально можливої частоти сигналу синхронізації використовують програми статичного аналізу затримок (Static Timing Analysis, STA). Для виконання STA в проектах на базі Intel FPGA в Quartus Prime використовують TimeQuest Timing Analyzer.

3.2 Теоретична частина

3.2.1 Подільник частоти на ціле число на базі лічильника

Подільник частоти на ступінь двійки ($2, 4, 8, 16, \dots, 2^N$) було розглянуто у попередній лабораторній роботі. А зараз розглянемо, як створити подільник частоти на будь-яке ціле число.

Подільник частоти на ціле число M можна реалізувати за допомогою синхронного лічильника. Структура такої схеми наведена на рис.3.1.

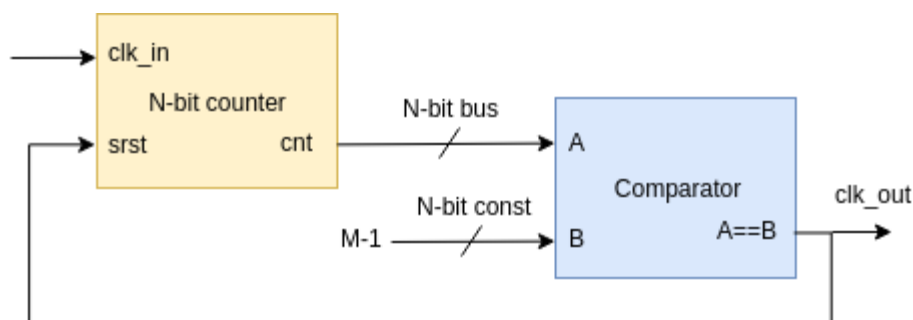


Рис.3.1 - Структура подільника частоти на ціле число M , побудованого на основі синхронного лічильника

Принцип роботи подільника частоти на ціле число M полягає в наступному. Синхронний по фронту лічильник вгору рахує імпульси вхідної частоти clk_in від 0 до $M-1$. Вміст лічильника порівнюється з константою $M-1$ за допомогою компаратора (схему перевірки цифрових кодів на рівність наведено в лабораторній роботі 0). Якщо вміст лічильника дорівнює $M-1$, на виході компаратора буде лог.1. Для всіх інших значень лічильника на виході компаратора буде лог.0. Вихід компаратора підключається до входу синхронного зкидання лічильника. У випадку, коли лічильник приймає значення $M-1$, на виході компаратора і на вході синхронного зкидання лічильника буде лог.1. Значить по наступному активному фронту clk_in вміст лічильника перейде в 0, вихід компаратора теж перейде в 0, сигнал на вході синхронного

зкидання лічильника стане неактивним і лічильник продовжить рахунок з нуля збільшуючи свій вміст на 1 по кожному наступному активному фронту `clk_in`. Коли лічильник знову дорахує до `M-1`, на виході компаратора знову зформується лог.1 і по наступному активному фронту на `clk_in` лічильник знову зкинеться в 0. Процес повторюватиметься циклічно.

Імпульсний сигнал `clk_out` знімається з виходу компаратора. Період сигналу `clk_out` містить `M` періодів сигналу `clk_in` ($T_{clk_out} = 7 \cdot T_{clk_in}$). Отже частота `clk_out` буде в `M` раз менша, ніж частота `clk_in` ($F_{clk_out} = F_{clk_in}/7$).

Часова діаграма зміни сигналів в подільнику частоти, що побудований на основі лічильника вгору, наведена на рис.3.2. (для `M=7`).

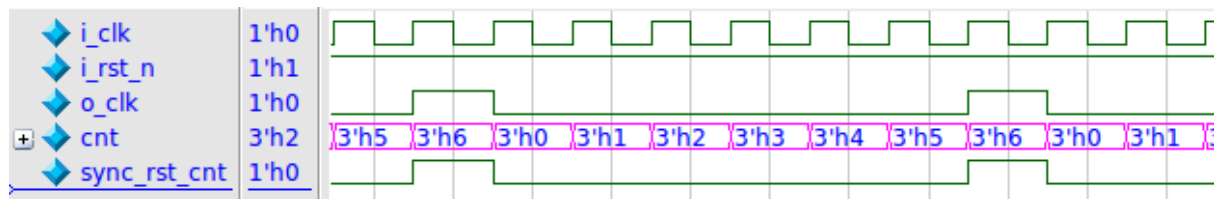


Рис.3.2 - Часова діаграма зміни сигналів в подільнику частоти на 7, що побудований на основі лічильника вгору

З рис.3.2. видно, що в подільнику частоти на 7 лічильник рахує від 0 до 6, а потім знову зкидається в 0. Вихід компаратора подається на вхід синхронного зкидання лічильника `sync_rst_cnt` і є виходом імпульсного сигналу частота якого в 7 разів менша, ніж частота сигналу `i_clk`. З рисунка видно, що період імпульсного сигналу `o_clk` в 7 разів більший ніж період імпульсного сигналу `i_clk`.

Якщо відоме число `M`, розрядність лічильника розраховують по формулі:

$$N = \text{ceil}(\log_2(M)), \quad (3.1)$$

де `M` - кількість цифрових кодів (від 0 до `M-1`), які точно повинен містити лічильник розрядністю `N`;

`log2()` - функція розрахунку логарифму за основою 2;

`ceil()` - функція округлення аргументу до найближчого більшого цілого числа (наприклад, якщо аргумент приймає значення 3.14, `ceil()` поверне 4);

У мові Verilog для розрахунку `ceil(log2(arg))` існує системна функція `$clog2(arg)`, що розраховує алгоритм за основою 2 від аргументу `arg`, до отриманого значення застосовує функцію `ceil()` і повертає результат.

Вихідний код на мові Verilog для подільника частоти на будь-яку цілу константу наведено в лістингу 3.1. В даному приклад використано синхронний лічильник, що рахує вгору.

Лістинг 3.1 - Вихідний код на мові Verilog для подільника частоти на будь-яку цілу константу. Подільник побудовано на базі синхронного лічильника, що рахує вгору.

```

module const_div(i_clk, i_rst_n, o_event);

input      i_clk;
input      i_rst_n;
output     o_event;

parameter  DIV_BY = 7;
localparam N = $clog2(DIV_BY);

reg        [N-1:0] cnt;

wire       [N-1:0] cnt_max      = (DIV_BY-1);
wire       sync_rst_cnt = (cnt_max== cnt);

assign o_event = sync_rst_cnt;

always @(posedge i_clk, negedge i_rst_n) begin
    if(~i_rst_n) begin
        cnt <= 0;
    end else begin
        if (sync_rst_cnt)
            cnt <= 0;
        else
            cnt <= cnt + 1'b1;
    end
end

endmodule

```

На рис.3.3. наведено схему в яку синтезується вихідний код з лістингу 3.1.

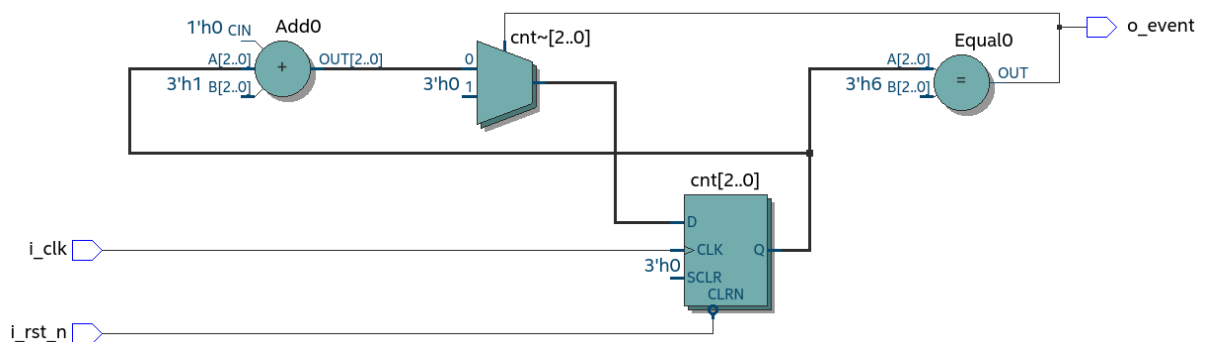


Рис.3.3. Цифрова схема подільника частоти на 7 в яку синтезується код з лістингу 3.1

Всі конструкції мови Verilog з лістингу 3.1, окрім параметрів, вже розглядалися у попередніх лабораторних роботах. Параметри у мові Verilog розглянемо в наступному розділі. А поки можна вважати параметри іменованими константами.

В лістингу 3.1 розглянуто подільник частоти на константу **DIV_BY**, що визначається параметром мови Verilog і в процесі роботи пристрою не може бути змінена.

В лістингу 3.2 наведено вихідний код на мові Verilog для подільника частоти на будь-яке ціле число, що подається на вхід **i_div_const**.

Лістинг 3.2 - Вихідний код на мові Verilog для подільника частоти на будь-яке ціле число. Подільник побудовано на базі синхронного лічильника, що рахує вниз.

```
module prog_div(i_clk, i_rst_n, i_div_const, o_event);

input          i_clk;
input          i_rst_n;
input  [9:0]    i_div_const;
output         o_event;

reg  [9:0]      cnt_ff;
reg  [9:0]      div_const_ff;

wire  preload_cnt = ~|cnt_ff;

assign o_event = preload_cnt;

always @(posedge i_clk, negedge i_rst_n) begin
    if(~i_rst_n) begin
        div_const_ff <= 1;
    end else begin
        div_const_ff <= i_div_const;
    end
end

always @(posedge i_clk, negedge i_rst_n) begin
    if(~i_rst_n) begin
        cnt_ff <= 0;
    end else begin
        if (preload_cnt)
            cnt_ff <= div_const_ff;
        else
            cnt_ff <= cnt_ff - 1'b1;
        end
    end
end

endmodule
```

Зверніть увагу, що в лістингу 3.2 для побудови подільника частоти використано лічильник, що рахує вниз. Для ділення частоти на M , в лічильник завантажується число рівне $M-1$ і по кожному наступному активному фронту вхідного імпульсного сигналу вміст лічильника зменшується на 1. Якщо лічильник дорахував до 0, на виході `o_event` та на вході `preload_cnt` формується лог.1. і по наступному активному фронту вхідного тактового сигналу в лічильник знову записується $M-1$. Перевагою такої схеми є простота перевірки умови перезавантаження лічильника. Не потрібно використовувати компаратор, як в лістингу 3.1. Для перевірки вмісту лічильнику на 0 достатньо використати лог. елемент. АБО-НІ.

Зверніть увагу, що в лістингу 3.2 дані з входу `i_div_const` записуються в регістр `div_const_ff` і лічильник перезавантажується вже значенням регістру `div_const_ff`. Загалом це стандартна практика - записувати значення входів цифрової схеми в синхронні по фронту регістри перед їх подальшим використанням для підвищення тактової частоти. Більш докладно питання оцінки максимально можливої частоти роботи синхронної цифрової схеми розглянуто в розділі 3.2.7.

Важливо розуміти, що вихідний імпульсний сигнал на рис.3.1, рис.3.3 та в лістингах 3.1-3.2 формується за допомогою комбінаційної логіки. А для комбінаційної логіки характерне явище гонок внаслідок якого після подачі нових значень на входи цифрової схеми на її виході протягом певного часу можуть спостерігатися паразитні імпульси, до встановлення остаточного правильного значення. Саме тому **вихід комбінаційної схеми не можна використовувати у якості безпосереднього джерела тактового сигналу для синхронних про фронту схем**, оскільки паразитні імпульси на входах синхронізації тригерів приведуть до хибних записів.

З наведеного вище випливає, що схеми наведені на рис.3.1, рис.3.3 та в лістингах 3.1-3.2 можна використовувати для формування періодичних імпульсів, які подаються, наприклад, на входи дозволу запису тригерів чи регістрів. Для формування тактового сигналу, який можна подавати на вхід синхронізації цифрової схеми необхідно імпульсний сигнал з виходу комбінаційної логіки пропустити через синхронний по фронту D-тригер для усунення паразитних імпульсів, як показано на рис.3.3. Усунення паразитних імпульсів забезпечується за рахунок того, що до кінця періоду `i_clk` на виході комбінаційної логіки встановлюється остаточне значення (лог. 0, або лог.1), яке по найближчому активному фронту записується в D-тригер.

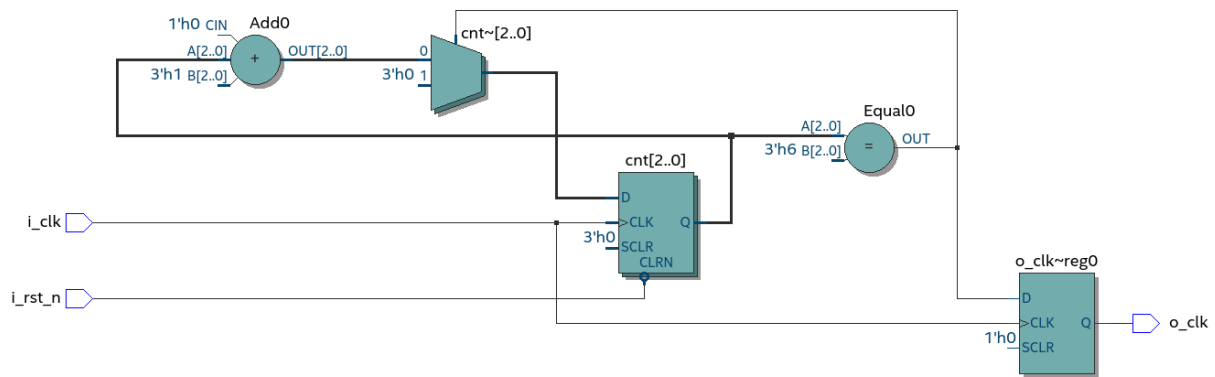


Рис.3.3 - Цифрова схема подільника частоти, в якій вихідний імпульсний сигнал формується синхронним по фронту D-тригером. Вихід o_clk можна використовувати, як джерело тактового сигналу для синхронних схем

3.2.2 Параметри у мові Verilog

Параметр у мові Verilog - це іменована константа, що визначається всередині модуля і яку можна перевизначити при створенні екземпляру модуля, задавши унікальне значення параметру для кожного екземпляру. Параметри є потужним інструментом мови Verilog, що дозволяє зробити опис цифрових схем більш універсальним. Наприклад, створити суматор, розрядності входів та виходу якого задаються параметром і при зміні параметру синтезувати цифрові схеми суматорів різної розрядності - лістинг 3.3.

Лістинг 3.3 - Параметризований суматор на мові Verilog, в якому розрядність входів і виходу визначається параметром WIDTH

```
module adder(i_carry, i_op1, i_op2, o_sum, o_carry);

parameter WIDTH = 8;

input          i_carry;
input  [WIDTH-1:0] i_op1;
input  [WIDTH-1:0] i_op2;
output [WIDTH-1:0] o_sum;
output          o_carry;

assign {o_carry, o_sum} = i_carry + i_op1 + i_op2;

endmodule
```


Як видно з лістингу 3.3 визначити параметр на мові Verilog дуже просто. Достатньо після ключового слова **parameter** вказати ім'я параметру і, далі, після оператора = задати значення по замовчуванню. Якщо значення параметру не буде перевизначено під час створення екземпляру модуля, параметр зберігатиме значення задане по замовчуванню.

В лістингу 3.3 значення параметру використовується для визначення розрядності виходу та розрядності входів суматора. Змінюючи значення параметру можна змінювати розрядність цифрової схеми суматора, що синтезується з коду на Verilog.

Перевизначення значень параметрів під час створення екземпляру модуля продемонстровано в лістингу 3.4 на прикладі тестбенчу для подільника частоти з лістингу 3.1.

Лістинг 3.4 - Перевизначення параметрів у мові Verilog на прикладі тестбенчу для подільника частоти з лістингу 3.1

```
`timescale 1ns / 1ps

module testbench;

parameter PERIOD = 20;
parameter DIV_BY_CONST = 7;

reg      i_clk, i_rst_n;
wire     o_clk;

const_div #(.DIV_BY(DIV_BY_CONST)) const_div_inst(.i_clk (i_clk),
                                                    .i_rst_n (i_rst_n),
                                                    .o_event (o_clk)
                                                    );

initial begin
    i_clk = 0;
    forever #(PERIOD/2) i_clk = ~i_clk;
end

initial begin
    i_rst_n = 1'b0;
    @(negedge i_clk) i_rst_n = 1;
    repeat (30) @(negedge i_clk);
    $finish;
end

endmodule
```

Як видно з лістингу 3.4 для перевизначення параметру під час створення екземпляру модуля, після імені модуля необхідно використати конструкцію `#(...)`, де всередині круглих скобок через кому перерахувати імена параметрів, які необхідно перевизначити. Перед кожним іменем параметру необхідно ставити крапку. Після імені параметру необхідно в круглих скобках вказати нове значення параметру для створюваного екземпляру модуля. Якщо не перевизначити значення параметру під час створення екземпляру модуля (не використовувати конструкцію `#(...)`), будуть використані значення параметрів по замовчуванню.

Значення параметрів враховуються перед початком синтезу, та перед початком симуляції. Під час симуляції значення параметру змінити не можна.

3.2.3 Оператори редукції у мові Verilog

Операторами редукції у мові Verilog називають оператори логічних функцій (AND, OR, NOT, XOR, тощо), що застосовуються до всіх розрядів аргументу. Розрядність результату операторів редукції - 1 біт.

Наприклад, в лістингу 3.2 оператор редукції АБО-НІ (NOR) застосовується до змінної `cnt_ff`, що описує вміст лічильника, для формування сигналу синхронного завантаження в лічильник: `wire preload_cnt = ~|cnt_ff;`

Іншими словами в операторах редукції розряди аргументу подаються на входи логічного елементу, що визначається оператором редукції (AND, OR, NOT, XOR, тощо), а вихід логічного елементу формує результат оператора редукції. Наприклад, оператор редукції `~|cnt_ff` синтезується в логічний елемент АБО-НІ (NOR), на входи якого подаються розряди лічильника `cnt_ff`.

Перелік операторів редукції у мові Verilog:

<code>&</code>	Reduction and
<code>~&</code>	Reduction nand
<code> </code>	Reduction or
<code>~ </code>	Reduction nor
<code>^</code>	Reduction xor
<code>~^ or ^~</code>	Reduction xnor

3.2.4 Подільник частоти на ціле число на базі регістру зсуву

Ще один підхід до побудови подільників частоти на задане ціле число M полягає у використанні регістру зсуву, вихід якого підключений до входу, а один із бітів під час ресету встановлюється в 1.

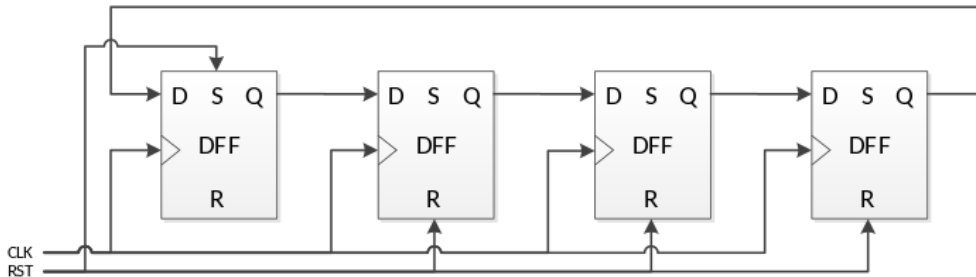


Рис.3.4 - Схема лічильника у позиційному коді (One-hot Counter, Ring Counter), яку можна використовувати, у якості подільника частоти

Таку схем називають лічильником у позиційному коді (One-hot Counter), або кільцевим лічильником (Ring Counter). Опис подібної цифрової схеми на мові Verilog наведено в лістингу 3.5.

Лістинг 3.5 - Шаблон опису лічильника у позиційному коді (One-hot Counter) на мові Verilog. Таку схему можна застосовувати, як подільник частоти на ціле число.

```
module one_hot_div(i_clk, i_rst_n, o_event);

input      i_clk;
input      i_rst_n;
output     o_event;

parameter  N = 5;

reg        [N-1:0] one_hot_cnt;

assign o_event = one_hot_cnt[0];

always @(posedge i_clk, negedge i_rst_n) begin
    if(~i_rst_n) begin
        one_hot_cnt <= 1;
    end else begin
        one_hot_cnt <= {one_hot_cnt[N-2:0], one_hot_cnt[N-1]};
    end
end

endmodule
```

Результат синтезу вихідного коду з лістингу 3.5 в Quartus наведено на рис.3.5.

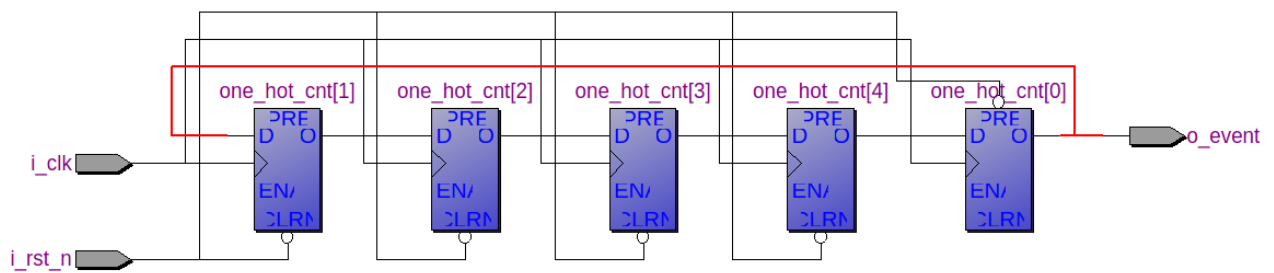


Рис.3.5 - Результат синтезу вихідного коду з лістингу 3.5 (лічильник у позиційному коді), що є подільником частоти на 5

Часова діаграма зміни сигналів one-hot лічильника з рис.3.5 наведена на рис. 3.6.

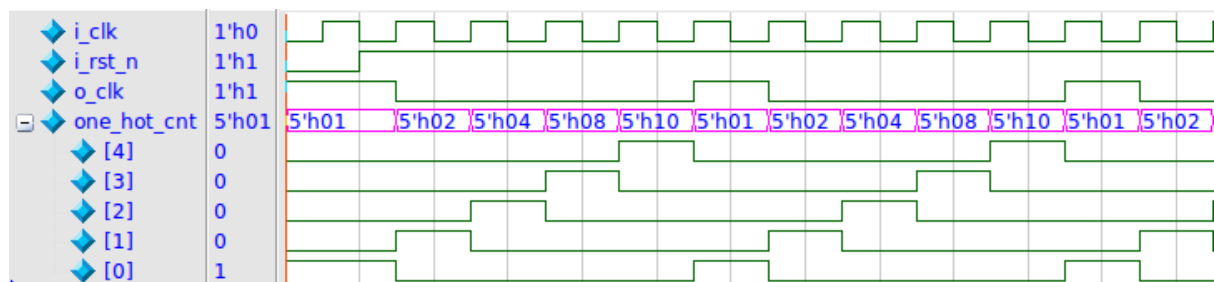


Рис.3.6 - Часова діаграма зміни сигналів у 5-розрядному one-hot лічильнику, який є подільником частоти на 5 (схема на рис.3.5)

З рис.3.6. видно, що одразу після ресету схеми, молодший розряд one-hot лічильника встановлюється в лог.1, а всі інші розряди встановлюються в лог.0. Далі по кожному наступному активному фронту на вході i_clk відбувається зсув лог.1 в сусідній розряд one-hot лічильника, а в той розряд, де раніше була лог.1, записується лог.0. Вихід o_clk підключається до молодшого розряду one-hot лічильника.

По кожному п'ятому імпульсу на вході i_clk в молодшому розряді схеми з рис.3.5 буде з'являтися лог.1. Період імпульсного сигналу на виході o_clk буде дорівнювати п'яти періодам вхідного імпульсного сигналу i_clk. Отже частота імпульсів на виході o_clk буде в 5 разів менша ніж у імпульсів на вході i_clk.

За допомогою такого підходу можна побудувати подільник частоти на будь-яке ціле число M. При цьому кількість тригерів, що необхідна для побудови, буде дорівнювати M.

Перевагою подільника частоти на базі one-hot лічильника є висока максимально допустима частота імпульсного сигналу на вході *i_clk*, оскільки між тригерами відсутня комбінаційна логіка. Більш докладно питання оцінки максимально можливої частоти роботи синхронної цифрової схеми розглянуто в розділі 3.2.7.

3.2.5 Подільник частоти на дробне число

Подільник частоти імпульсного сигналу на дробне число можна реалізувати за допомогою цифрового генератора з керованою частотою (Numerically Controlled Oscillator, NCO), який детально розглянуто в розділі 2.2.13 попередньої лабораторної роботи №2.

Нагадаємо, що структурна схема NCO має вигляд зображений на рис.3.7.

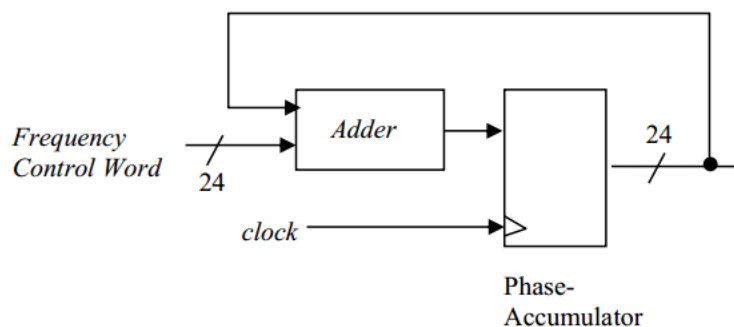


Рис.3.7 - Структурна схема цифрового генератора з керованою частотою (NCO)

З рис.3.7 видно, що основою NCO є регістр, вміст якого збільшується на певне значення *phase_step* по кожному активному фронту вхідного сигналу синхронізації. На рис.3.7 *phase_step* позначено, як *Frequency Control Word*.

На рис.3.7 показано NCO розрядністю 24 біта, однак ніщо не заважає використати іншу розрядність, скажімо 32 біта.

Частота переповнення подібного лічильника буде дорівнювати частоті зміни старшого розряду регістру. Дійсно, якщо вміст N-розрядного регістру перевищить $\frac{2^N-1}{2}$ (половину від максимально можливого значення), встановиться старший розряд регістру. Коли додавання чергового *phase_step* до вмісту регістру приведе до його переповнення, старший розряд регістру перейде в 0.

У разі якщо NCO використовується в якості подільника частоти на дробне число, сигнал вихідної тактової частоти o_clk підключається до старшого розряду регістру NCO. Частота імпульсів на виході o_clk буде визначатися частотою переповнення NCO, яка залежить від phase_step.

Згадаємо, що $F_{out} = \frac{phase_step \cdot F_{in}}{2^N}$ (формула обґрунтована в лабораторній роботі №2). Де F_{out} - частота імпульсів на виході o_clk (частота переповнювання регістру NCO), F_{in} - частота імпульсів на вході синхронізації i_clk, а N - розрядність регістру NCO. Якщо відома частота F_{in} вхідного тактового сигналу, з наведеної формули можна розрахувати значення phase_step для одержання необхідної частоти F_{out} . Зверніть увагу, що $\frac{2^N}{phase_step} = \frac{F_{in}}{F_{out}}$. В залежності від phase_step, відношення $\frac{2^N}{phase_step}$ може бути, як цілим, так і дробним числом. Як наслідок, відношення $\frac{F_{in}}{F_{out}}$ теж може бути дробним числом. Отже задаючись необхідним дробним відношенням $\frac{F_{in}}{F_{out}}$, можна розрахувати необхідне значення phase_step.

3.2.6 Підвищення частоти за допомогою ФАПЧ (PLL)

В даному розділі розглянемо компонент Фазової Автопідстройки Частоти (ФАПЧ), що входить до складу сучасних FPGA мікросхем. Англійською мовою подібний пристрій називається Phase Locked Loop (PLL).

Умовне графічне позначення компоненту PLL наведено на рис.3.8.

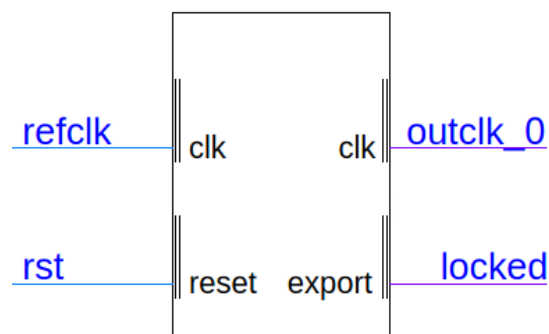


Рис.3.8 - Умовне графічне позначення компоненту PLL

Основне призначення компоненту PLL - створити сигнал тактової частоти, що має заданий зсув фаз відносно вхідного сигналу тактової частоти. Зсув фаз між вхідним і вихідним імпульсними сигналами можна задати від 0 до 2π . Частота вихідного тактового сигналу, в залежності від налаштувань, може бути в кілька разів більшою, або в кілька разів меншою ніж у вхідного тактового сигналу.

Як видно з рис.3.8 компонент PLL має вхід тактової частоти (refclk), вихід тактової частоти (outclk), вхід скидання rst та вихід locked. За необхідності, у компонента PLL може бути кілька тактових виходів, кожен з яких можна налаштувати окремо.

На вхід тактової частоти refclk подається періодичний імпульсний сигнал з частотою порядку кількох десятків мегагерц. Зазвичай такий сигнал отримують від зовнішнього кварцевого генератору.

Вихід locked встановлюється в логічну одиницю, коли компонент PLL коректно запустився і між вхідним та вихідним імпульсними сигналами встановився заданий зсув фаз. При створенні цифрових пристроїв з використанням PLL рекомендовано виводити вихід locked на світлодіод, щоб одразу можна було визначити, що блок PLL з якихось причин не працює. Причиною нестабільної роботи PLL можуть бути шуми джерела живлення та зашумлений сигнал вхідної тактової частоти, що має тремтіння фази (clock jitter).

У кожній FPGA мікросхемі зазвичай є кілька блоків PLL. Однак PLL використовують не лише в FPGA. Цей компонент є складовою частиною величезної кількості пристроїв. Зокрема PLL наявні у складі практичного всіх сучасних мікроконтролерів.

PLL використовують для:

- Створення сигналу синхронізації з високою частотою;
 - Створення заданого зсуву фаз між двома тактовими сигналами;
 - Амплітудної та частотної демодуляції;
 - Отримання тактового сигналу і даних з потоку бітів (clock and data recovery).
- Актуально для швидкісних послідовних інтерфейсів - USB, SATA, Ethernet, HDMI, PCIe і т.д;

Принцип роботи і побудови PLL ми розглянемо детально в одній із останніх додаткових лабораторних робіт. А в даній роботі зосередимося на налаштуванні блоку PLL, що входить до складу FPGA. За бажанням докладніше про внутрішню будову і функціонування PLL можна прочитати за посиланням [3.1].

Найбільш розповсюджене застосування PLL - створення високочастотного тактового сигналу (порядку 0.1 - 3 ГГц) із імпульсного сигналу нижчої частоти (порядку 10-50 МГц), що надходить від зовнішнього кварцевого генератора.

Високочастотний сигнал порядку сотень-тисяч мегагерц проблематично передавати по друкованій платі без спотворень. Створення високочастотних друкованих плат значно здорожчує їх конструкцію. Саме тому широко розповсюдженим підходом є створення відносно низькочастотного стабільного імпульсного сигналу за допомогою зовнішнього кварцевого генератора, передача такого сигналу в цифрову мікросхему і створення високочастотного сигналу синхронізації всередині мікросхеми з використанням PLL.

Створення і налаштування компоненту PLL для FPGA мікросхем Intel в Quartus Prime описано в розділі 3.3.1.

3.2.7 Розрахунок максимально допустимої тактової частоти

Розглянемо фактори від яких залежить максимально можлива тактова частота в синхронних по фронту цифрових схемах.

Структура синхронних по фронту цифрових мікросхем наведена на рис.3.9.

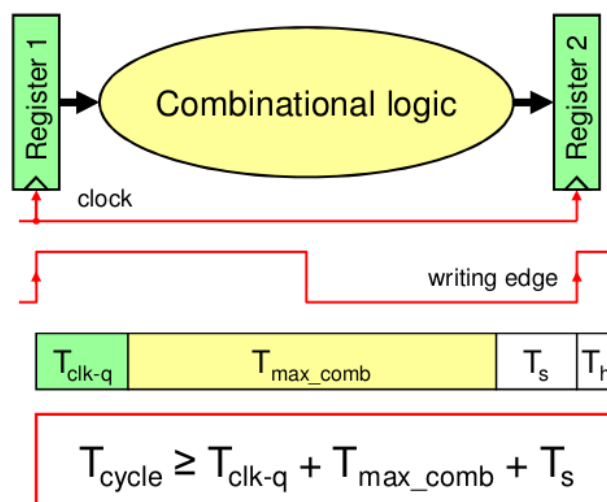


Рис.3.9 - Структура синхронних по фронту цифрових мікросхем і оцінка затримок

З рис.3.9 видно, що цифрові мікросхеми складаються із шарів синхронних по фронту регістрів (для багаторозрядних сигналів), або синхронних по фронту тригерів (для однорозрядних сигналів), між якими включена комбінаційна логіка. Вихід кожного попереднього регістру (тригеру) підключається до входу комбінаційної логіки, а вихід комбінаційної логіки підключається до входу наступного регістру (тригеру).

Таку структуру використовують для боротьби з явищем гонок, що виникають в комбінаційних схемах і призводять до появи паразитних імпульсів (глітчів, glitches) на виходах комбінаційних схем.

Розглянемо приклад гонок в комбінаційній логіці на прикладі схеми з рис.3.10.

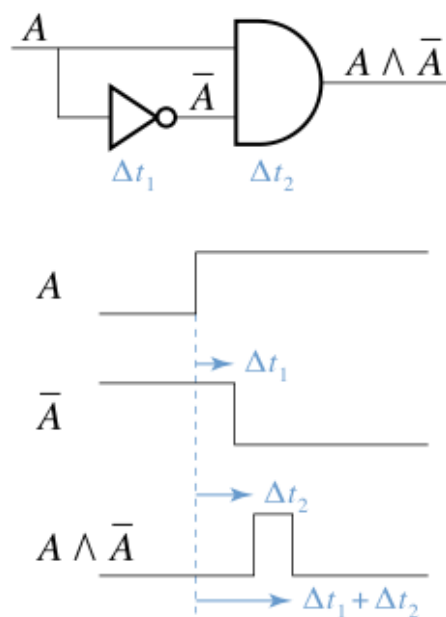


Рис.3.10 - Ілюстрація явища гонок в комбінаційній схемі

Отже, схема на рис.3.10 реалізує логічне рівняння $y = A \& (\sim A)$, де $\&$ позначає операцію логічного І, а \sim позначає операцію інверсії. Якщо не враховувати затримки логічних елементів, результатом наведеного вище логічного виразу буде завжди 0. Однак реальні логічні елементи мають затримку розповсюдження сигналу обумовлену тривалістю перезаряду ємності навантаження виходу логічного елементу через вихідний опір логічного елементу (ємність навантаження логічного елементу є сумою ємності з'єднувального провідника і вхідної ємності наступного логічного елементу). На рис.3.10 інвертор має затримку Δt_1 , а лог. елемент І має затримку Δt_2 .

Припустимо, що на вході і виході схеми з рис.3.10 присутній логічний нуль, а на виході інвертора присутня логічна одиниця. В момент часу t_0 сигнал на вході А переходить з 0 в 1 і поступає на один із входів елементу І та на вхід інвертора. Оскільки інвертор має затримку Δt_1 , це означає, що до моменту часу $t_0 + \Delta t_1$ на виході інвертора буде зберігатися 1. Отже до моменту $t_0 + \Delta t_1$ на обох входах елементу І будуть присутні лог.1. Оскільки елемент І має затримку Δt_2 , в момент часу $t_0 + \Delta t_2$ на виході інвертора встановиться лог.1. В момент часу $t_0 + \Delta t_1$ вихід інвертору сформує на одному з входів елементу І лог. 0 і через Δt_2 , в момент часу $t_0 + \Delta t_1 + \Delta t_2$ вихід елементу І перейде в 0. Бачимо, що внаслідок наявності затримок логічних елементів та кількох шляхів розповсюдження сигналу, де кожен шлях має різну затримку, отримали на виході схеми паразитний імпульс, якого за логікою роботи там не повинно бути.

Подібні паразитні імпульси на входах керування можуть призводити до катастрофічних наслідків, особливо в системах, що впливають на життя людини. Уявіть, що вихід комбінаційної логіки підключений до входу запуску ракети у винищувачі, або до входів керування подачею палива чи гальмами в автомобілі, або до входу керування двигуном інсулінової помпи, що автоматично вводить інсулін діабетикам. Паразитні імпульси на входах керування таких важливих систем можуть призвести до їх непередбачуваного запуску, що може мати жахливі наслідки.

Загалом до появи паразитних імпульсів призводить наявність в комбінаційній схемі кількох шляхів розповсюдження сигналу (від входу до виходу) з різними затримками. Після завершення перехідних процесів паразитні імпульси на виході комбінаційної схеми перестають з'являтися і там встановлюється остаточне значення. Тривалість завершення перехідних процесів визначає затримку комбінаційної схеми і дорівнює затримці найдовшого шляху розповсюдження сигналу від входу до виходу. Затримка комбінаційної схеми розраховується, як сума затримок логічних елементів ввімкнених послідовно у найдовшому шляху розповсюдження сигналів через комбінаційну схему. Наприклад, на рис.3.10 найдовший шлях розповсюдження сигналу: **вхід -> інвертор -> елемент І -> вихід**. Відповідно, затримка такої схеми буде дорівнювати сумі затримок інвертора та елементу І.

Саме для боротьби з паразитними імпульсами використовують структуру схеми наведену на рис.3.9. Ідея полягає у тому, що до завершення періоду сигналу синхронізації перехідні імпульси в комбінаційній схемі завершуються і на виході

встановлюється остаточне значення. Це усталене значення по найближчому активному фронту запишеться в наступний тригер, з виходу якого буде передано на вхід наступної комбінаційної схеми.

Отже протягом періоду сигналу синхронізації T необхідно, щоб дані після активного фронту на вході clk записалися в тригер/регістр (затримка запису t_{cq}), розповсюдились через комбінаційну логіку (затримка t_{max_comb}) і встановилися на вході даних наступного регістру/тригера, які мінімум за t_s (time setup) до найближчого активного фронту - див. рис.3.9.

Міркування наведені у попередньому абзаці дозволяють записати умову:

$$T \geq t_{cq} + t_{max_comb} + t_s \quad (3.1)$$

З формули (3.1) можна зробити висновок, що період сигналу синхронізації не повинен бути меншим за $t_{cq} + t_{max_comb} + t_s$. Мінімум можливе значення періоду сигналу синхронізації буде дорівнювати:

$$T_{min} = t_{cq} + t_{max_comb} + t_s \quad (3.2)$$

Максимально можливе значення частоти сигналу синхронізації визначається, як:

$$F_{max} = 1/T_{min} \quad (3.3)$$

Для більш точного розрахунку в наведених вище формулах необхідно врахувати затримку розповсюдження сигналу синхронізації (clock skew) - див. рис.3.11.

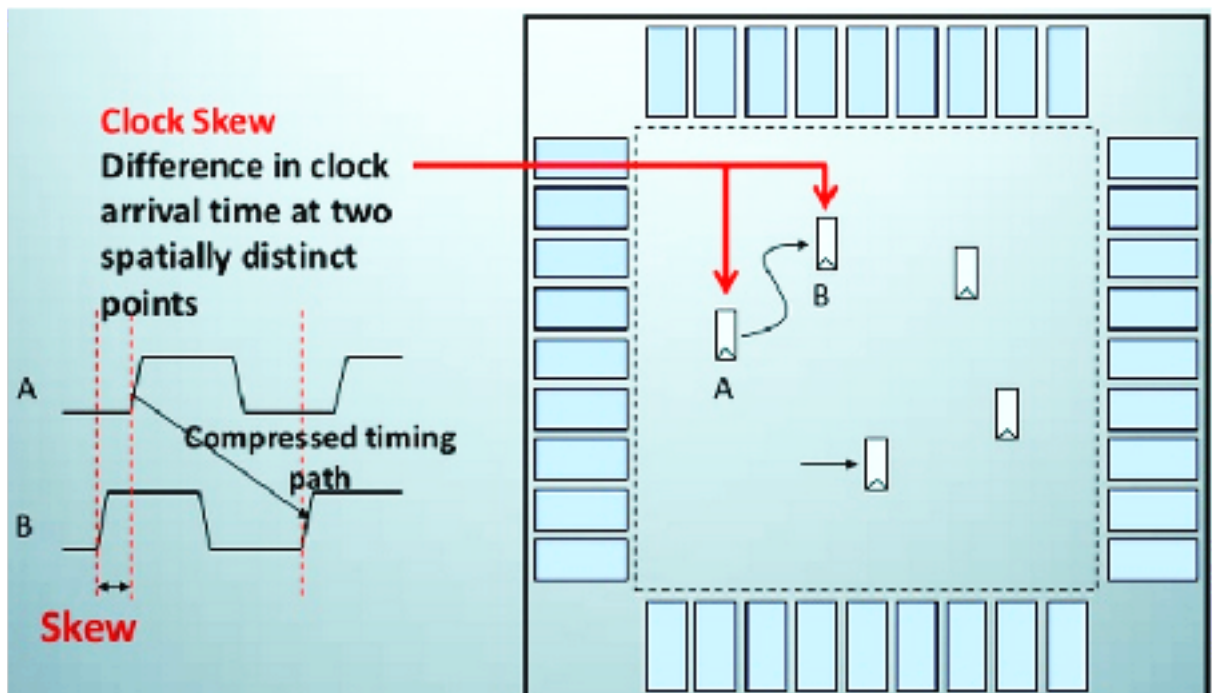


Рис.3.11 - Затримка розповсюдження сигналу синхронізації

Через різне розміщення тригерів на кристалі сигнал синхронізації може надходити на тактовий вхід тригера-джерела даних пізніше ніж на тактовий вхід тригера-приймача даних. Затримка між сигналами синхронізації на двох згаданих входах синхронізації і називається clock skew.

У випадку наявності clock skew дані повинні записатися в тригер-джерело, пройти через комбінаційн логіку і встановитися на вході тригера-приймача вже не за T , а за $(T - t_{cs})$, де t_{cs} - затримка clock skew.

З урахуванням clock skew формула (3.2) буде виглядати наступним чином:

$$T_{min} = t_{cq} + t_{max_comb} + t_s + t_{cs} \quad (3.4)$$

3.2.8 Статичний аналіз затримок (Static Timing Analysis)

Сучасні системи розробки цифрових мікросхем містять модулі статичного аналізу затримок (Static Timing Analysis, STA) для оцінки затримок розповсюдження сигналу і максимально можливої тактової частоти.

Модуль статичного аналізу оцінює затримки комбінаційних схем включених між шарами регістрів, розраховує максимально можливу частоту сигналу синхронізації і оптимізує комбінаційну логіку для досягнення необхідної максимально можливої тактової частоти.

Якщо не вдається досягти необхідної тактової частоти шляхом автоматичної оптимізації, програма для STA дозволяє переглянути перелік комбінаційних схем, що мають занадто велику затримку. Отриману інформацію інженер може використати для оптимізації структури проблемних частин цифрової схеми з метою зменшення затримок.

В Quartus Prime статичний аналіз затримок виконується в програмі TimeQuest Timing Analyzer.

Для коректної роботи статичного аналізу затримок необхідно вручну вказати всі сигнали синхронізації наявні у вашій цифровій схемі та потрібні частоти цих сигналів синхронізації. За необхідності можна визначити затримки розповсюдження сигналу між вхідними/вихідними тригерами і контактами FPGA, також певні особливості конструкції цифрової схеми (multicycle path і т.д.). Подібна інформація називається обмеженнями (constraints) на синтез і використовується під час автоматичного синтезу

цифрової схеми та створення топології (або файлу конфігурації FPGA) для отримання необхідного результату.

Визначення обмежень на синтез в Quartus Prime описано в розділі 3.3.2. Оцінку затримок і максимально можливої тактової частоти з використанням TimeQuest Timing Analyzer розглянуто в розділі 3.3.3.

3.2.9 Знакове представлення чисел, доповняльний код

Розглянемо знаковий формат та представлення від'ємних чисел в цифрових системах. Ця інформація знадобиться для розуміння наступного розділу.

Будь-яке N -розрядне число можна інтерпретувати, як беззнакове, або як знакове.

N -розрядне число в беззнаковому представленні приймає лише додатні значення, від 0 до $2^N - 1$ (всього 2^N можливих значень). Наприклад, 4-розрядне число в беззнаковому представленні приймає $2^4 = 16$ значень, від 0 до 15 (в двійковому форматі це будуть цифрові коди від 0000 до 1111).

N -розрядне число у знаковому представленні приймає, як додатні, так і від'ємні значення, від -2^{N-1} до $2^{N-1} - 1$ (всього 2^N можливих значень). Іншими словами, одна половина цифрових кодів (зі старшим бітом рівним 1), відводиться для представлення від'ємних значень, а друга половина цифрових кодів (зі старшим бітом рівним 0) відводиться для представлення нуля і додатних значень.

Додатні значення N -розрядного числа представляються у прямому коді. Прямий код являє собою уже відоме вам представлення чисел у двійковому коді.

У випадку знакової інтерпретації від'ємні значення представляються в доповняльному коді (Two's Complement Code).

Припустимо, що додатне N -розрядне число у прямому коді позначається **Qp**. А від'ємне N -розрядне число у доповняльному коді позначається **Qn**. В такому випадку молодші N -розрядів суми **Qp** + **Qn** будуть дорівнювати нулю. Іншими словами N -розрядні значення в прямому і доповняльному коді доповнюють одне одного до нуля і в сумі дають нуль в молодших N розрядах результату суми.

Перейти від додатного значення у прямому коді до від'ємного значення у доповняльному коді можна за формулою:

$$Q_n = \overline{Q_p} + 1 \quad (3.5)$$

де $\overline{Q_p}$ - операція інверсії бітів числа Q_p

Для переходу від від'ємного значення в доповняльному коді Q_n до додатного значення в прямому коді Q_p використовується такий же принцип, як і в формулі (3.5):

$$Q_p = \overline{Q_n} + 1 \quad (3.6)$$

Наприклад, припустимо, що ми оперуємо 4-розрядними знаковими числами. Виберемо додатне значення у прямому коді **Qp = 0101** (0101 це 5 у двійковому форматі). Перейдемо до від'ємного значення цього числа у доповняльному коді за формулою (3.5): $Q_n = \overline{0101} + 1 = 1010 + 1 = 1011$. Отже отримане число 1011 представлятиме -5 для 4-розрядних знакових чисел. Перевіримо, чи виконується умова $Q_p + Q_n = 0$ для наших 4-розрядних чисел: **Qp + Qn = 5 - 5 = 0101 + 1011 = 10000**. Отримали 5-розрядне число у якого старший біт 1, а молодші 4 біти нулі. Отже сума 4-розрядних чисел $Q_p + Q_n$ повернула нулі у молодших 4 розрядах.

Ілюстрація знакового і беззнакового представлення 4-розрядних чисел наведена на рис.3.12.

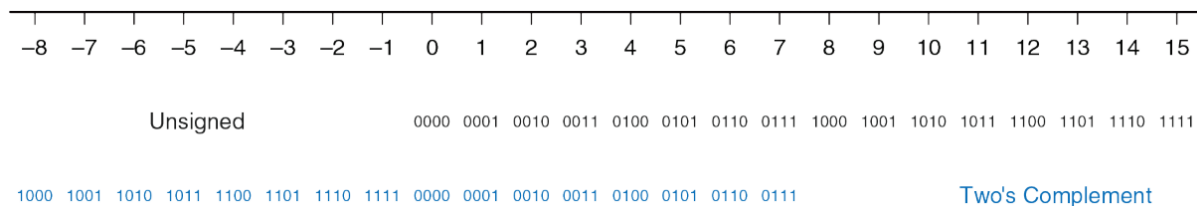


Рис.3.12 - Ілюстрація знакового і беззнакового представлення 4-розрядних чисел

Необхідно запам'ятати одну важливу граничну ситуацію. Найбільш негативне значення N-розрядного знакового числа (старший біт 1, всі решта бітів нулі) не має додатного еквіваленту в прямому коді (вище вже зазначалося, що для знакових N-розрядних чисел найбільш негативне значення буде -2^{N-1} , а найбільш позитивне $2^{N-1} - 1$). А отже застосування формули (3.6) до найбільш негативного значення N-розрядного знакового числа поверне те саме негативне значення (а не додатній еквівалент у прямому коді). Розглянемо таку ситуацію на прикладі. Припустимо, що маємо 4-розрядне знакове число. Найбільш негативним значенням цього числа буде

$Q_n = -2^{N-1} = -8$ (цифровий код у двійковому форматі **1000**). Спробуємо перейти від негативного числа Q_n у доповняльному коді до додатного числа у прямому коді Q_p за формулою (3.6): $Q_p = \overline{1000} + 1 = 0111 + 1 = 1000$. Отримали той самий цифровий код, що представляє -8. Для правильного опрацювання такої граничної ситуації при перетворенні чисел з доповняльного коду в прямий код завжди перевіряють чи не дорівнює число найбільш негативному значенню. І якщо число дорівнює найбільш негативному значенню -2^{N-1} (старший біт 1, решта бітів нулі), не застосовують до такого числа формулу (3.6), а повертають найбільше позитивне значення $2^{N-1} - 1$ (старший біт 0, решта бітів 1).

3.2.10 FM-передавач на базі генератора з цифровим керуванням

Розглянемо принцип роботи цифрового FM передавача з частотною модуляцією. Суть радіопередачі з частотною модуляцією полягає у тому, що частота несучого радіосигналу (Carrier Signal), який передається в ефір, пропорційна значенню аудіосигналу. Зміна значення аудіосигналу у часі призводить до зміни частоти радіосигналу. На рис.3.13. показана ілюстрація частотної модуляції несучого високочастотного сигналу (Carrier Signal) за допомогою низькочастотного синусоїдального сигналу (Modulation Signal). З рис. 3.13 видно, що частота модульованого сигналу (FM Modulated Signal) змінюється пропорційно зміні значення низькочастотної модулюючої синусоїди.

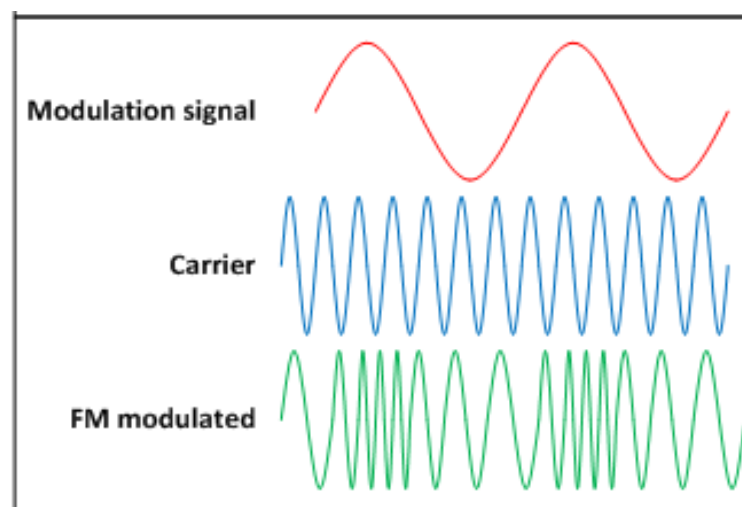


Рис.3.13- Ілюстрація частотної модуляції (FM Modulation)

Несуча частота для FM радіо лежить в межах 80-110 МГц. В цьому проміжку частот знаходяться всі FM радіостанції.

У нашому прикладі для цифрового FM радіо у якості несучого сигналу оберемо імпульсний сигнал, з частотою 90 МГц. Імпульсний сигнал з частотою 90 МГц містить у своєму спектрі синусоїдальний сигнал з частотою 90 МГц, а також синусоїдальні складові з більш високими частотами (гармоніки). За бажанням ці високочастотні складові можна відфільтрувати пропустивши імпульсний несучий радіосигнал через аналоговий фільтр низької частоти, частота зрізу якого дещо вища, ніж частота несучого сигналу (наприклад у випадку використання несучої частоти 90 МГц, можна обрати частоту зрізу згаданого фільтру рівну 95 - 100 МГц).

Структурна схема простого цифрового FM передавача наведена на рис.3.14.

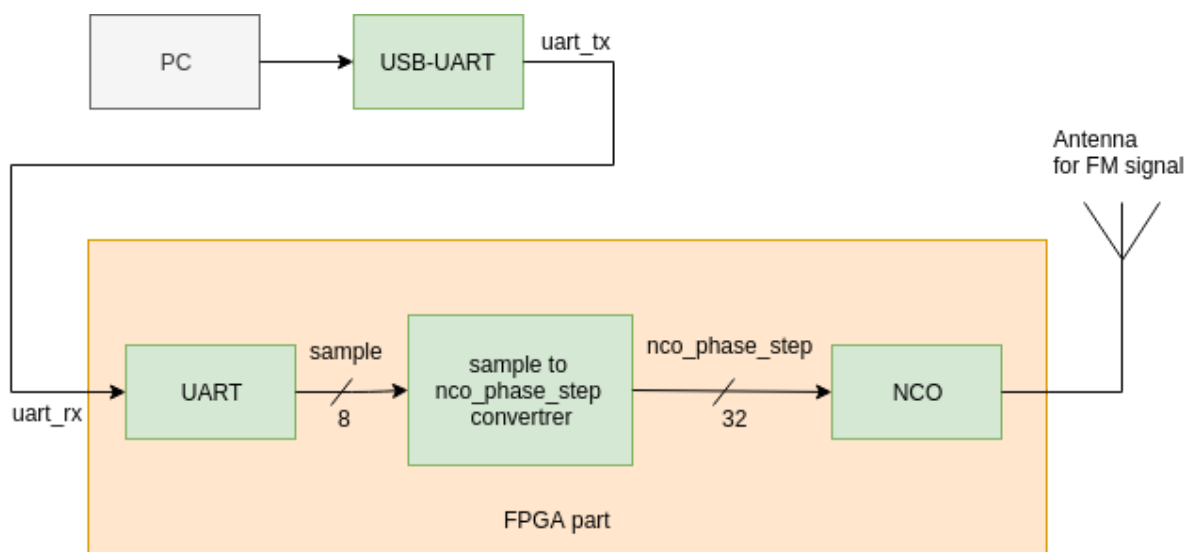


Рис.3.14 - Структурна схема цифрового FM передавача

Аудіо файл в форматі wav, що буде транслюватися в радіоефір передається в FPGA з комп'ютера через інтерфейс UART. Як видно з рис.3.14, цифровий FM передавач, реалізований всередині FPGA, складається з UART приймача, програмованого генератора височотного імпульсного сигналу (NCO), частота якого обумовлюється цифровим кодом на вході та перетворювача цифрового коду аудіосигналу у значення коду керування частотою NCO.

Перед тим, як розглянути окремі компоненти на рис.3.14, необхідно визначитися зі структурою аудіо файлу в форматі wav та способом перетворення такого аудіо файлу в звук. У попередній лабораторній роботі ми розглянули принцип роботи генератора

синусоїдального аналогового сигналу. Аналогічний підхід використовується до відтворення звуку з аудіо файлів. Аудіо сигнал представляється зміною напруги у часі. Якщо таку змінну напругу подати на динамік, динамік відтворить звук. Аудіо файл в форматі wav складається із послідовності цифрових кодів, кожен з яких представляє напругу аудіо сигналу для заданої частоти дискретизації F_d - рис.3.15. Сусідні точки на графіку змінюються з періодом дискретизації T_d , що обернений до частоти дискретизації ($T_d = 1/F_d$). Існує кілька поширених частот дискретизації аудіо файлів (22050 Гц, 32000 Гц, 44100 Гц, 192000 Гц та багато інших). Частота дискретизації аудіо сигналу, що зберігається у wav файлі визначається при створенні цього файлу і її можна переглянути у програмах відтворення і редагування звуку. Якщо цифрові коди з wav аудіо файлу подати на ЦАП з відповідною частотою дискретизації, отримаємо аудіо сигнал, який можна прослухати за допомогою динаміку.

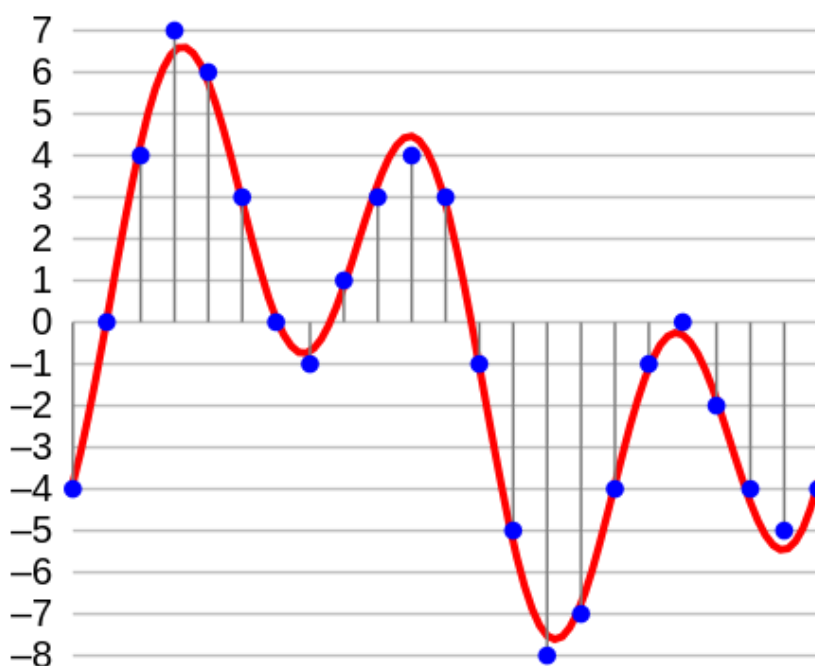


Рис.3.15 - Змінний аудіо сигнал представлений у вигляді набору цифрових кодів, що змінюються певною частотою дискретизації

В даному прикладі будемо використовувати 8-розрядні wav файли, з частотою дискретизації 22050 Гц та одним каналом (моно). Такий аудіо файл містить певний заголовок, після якого слідує послідовність 8-бітних цифрових кодів, кожен з яких представляє напругу аналогового аудіо сигналу в певний момент часу.

Розглянемо в загальному вигляді роботу кожного компоненту цифрового FM передавача з рис.3.14.

8-розрядний wav аудіо файл передається в FM передавач з комп'ютера через інтерфейс UART на швидкості 230 400 біт за секунду. Це призводить до того, що на виході даних UART з'являються нові 8-розрядні значення аудіо сигналу з частотою $230400 \div 10 = 23040$ Гц (23040 разів на секунду). Ця частота близька до стандартної частоти дискретизації аудіо, що дорівнює 22050 Гц (не професіоналу складно розрізнити на слух аудіо сигнали відтворені з частотами 23040 Гц і 22050 Гц). Якщо видавати отримані 8-розрядні цифрові коди на 8-розрядний ЦАП з частотою 23040 Гц, а до виходу ЦАП підключити навушники (бажано через підсилювач), у навушниках буде відтворено аудіо файл, що передається з комп'ютера. Однак ми не будемо використовувати ЦАП, а будемо модулювати (змінювати) 90 МГц несучу частоту пропорційно до зміни 8-розрядних цифрових кодів, що представляють значення аудіо сигналу в моменти часу взяті з заданою частотою дискретизації.

Для створення високочастотного сигналу, що передається в радіо ефір, використовується програмований цифровий генератор NCO. Робота NCO детально розглянута в розділі 3.2.5 даної лабораторної роботи і розділі 2.2.13 попередньої лабораторної роботи.

Частота імпульсного сигналу зміни старшого розряду NCO розраховується за формулою $F_{out} = \frac{phase_step \cdot F_{in}}{2^N}$ (формула обґрунтована в лабораторній роботі №2). F_{in} - частота імпульсів на вході синхронізації, а N - розрядність регістру NCO. У нашому прикладі цифрового FM передавача використовується частота сигналу синхронізації $F_{in} = 250$ МГц і 32-розрядний регістр NCO.

Підсумуємо, що несучий радіосигнал будемо отримувати зі старшого розряду регістру NCO, а частота цього несучого радіосигналу буде залежати від значення цифрового коду `phase_step`.

Необхідно створити залежність між 8-бітними значеннями цифрового коду аудіо сигналу на виході UART та частотою зміни імпульсного сигналу на виході NCO (нагадуємо, що сигнал для радіо передачі отримуємо зі старшого розряду NCO).

Важливо домовитися, що ми будемо інтерпретувати цифровий код аудіо сигналу, як знакове число, що може приймати додатні і від'ємні значення.

Для досягнення частотної модуляції, залежність між цифровим кодом аудіо сигналу (у знаковому представленні) і частотою імпульсного сигналу на виході NCO повинна бути наступна (позначимо цифровий код аудіо сигналу, як **sample**):

- якщо **sample** == 0, частота імпульсного сигналу на виході NCO складає 90 МГц;
- якщо **sample** > 0, частота імпульсного сигналу на виході NCO стає пропорційно більшою за 90 МГц;
- якщо **sample** < 0, частота імпульсного сигналу на виході NCO стає пропорційно меншою за 90 МГц;
- найбільш додатному значенню 8-розрядного цифрового коду аудіо сигналу (**sample** = $2^7 - 1 = 127$) відповідає найбільше позитивне відхилення частоти несучого сигналу від 90 МГц. Назвемо це найбільше відхилення FM Deviation. Отже в такому випадку частота несучого сигналу дорівнює **F = 90 МГц + FM Deviation**;
- найбільш від'ємному значенню 8-розрядного цифрового коду аудіо сигналу (**sample** = $-2^7 = -128$) відповідає найбільше негативне відхилення частоти несучого сигналу від 90 МГц і в такому випадку частота несучого сигналу дорівнює **F = 90 МГц - FM Deviation**.

Зазвичай для різних радіостанцій значення FM Deviation лежить в межах від 5 КГц до 75 КГц. У нашому прикладі FM Deviation для спрощення схеми обрано рівним 61 КГц. У чому полягає спрощення схеми буде показано далі.

Розрахуємо за формулою (2.12) із попередньої лабораторної роботи значення **phase_step** для 32-розрядного NCO, якщо тактова частота $F_{in} = 250$ МГц і необхідно отримати імпульсний сигнал частотою $F = 90$ МГц:

$$phase_step = 2^N \cdot \frac{F}{F_{in}} = 2^{32} \cdot \frac{90\,000\,000}{250\,000\,000} = 1546188226 = 0x5C28F5C2$$

Тепер розглянемо, як буде змінюватися **phase_step**, якщо цифровий код аудіо сигналу у знаковому форматі (**sample**) не дорівнюватиме нулю:

$$\begin{aligned} phase_step &= 2^N \cdot \frac{F}{F_{in}} = 2^{32} \cdot \frac{(90\text{ МГц} + \Delta F(sample))}{250\text{ МГц}} = 2^{32} \cdot \frac{90\text{ МГц}}{250\text{ МГц}} + 2^{32} \cdot \frac{\Delta F(sample)}{250\text{ МГц}} = \\ &= 0x5C28F5C2 + 2^{32} \cdot \frac{\Delta F(sample)}{250\text{ МГц}} \end{aligned} \quad (3.6)$$

де $\Delta F(sample)$ - зміна частоти несучого сигналу в залежності від **sample**.

Як бачимо, `phase_step` складається з розрахованої вище постійної частини `0x5C28F5C2` та змінної частини, що залежить від цифрового коду аудіо сигналу `sample`.

$\Delta F(sample)$ розраховується наступним чином:

$$\Delta F(sample) = \frac{sample \cdot FM_DEVIATION}{2^7} = \frac{sample \cdot FM_DEVIATION}{128} \quad (3.7)$$

З формули (3.7) випливає, що у випадку знакового числа `sample`, найбільш позитивному значенню `sample = 127` буде відповідати максимальне позитивне відхилення частоти несучого сигналу майже рівне `FM_DEVIATION`. А у випадку найбільш негативного значення `sample = -128`, відхилення частоти несучого сигналу буде найбільш негативне, практично і рівне `-FM_DEVIATION`.

Якщо обрати `FM_DEVIATION` рівним 61 КГц і підставити формулу (3.7) в формулу (3.6), отримаємо:

$$\begin{aligned} phase_step &= 0x5C28F5C2 + 2^{32} \cdot \frac{\Delta F(sample)}{250 \text{ МГц}} = 0x5C28F5C2 + 2^{32} \cdot \frac{sample \cdot FM_DEVIATION}{128 \cdot 250 \text{ МГц}} = \\ &= 0x5C28F5C2 + 2^{32} \cdot \frac{sample \cdot 61 \text{ КГц}}{128 \cdot 250 \text{ МГц}} = 0x5C28F5C2 + 8187 \cdot sample \approx \\ &\approx 0x5C28F5C2 + 8192 \cdot sample = 0x5C28F5C2 + 2^{13} \cdot sample = \\ &= 0x5C28F5C2 + (sample \ll 13) \end{aligned} \quad (3.8)$$

Як бачимо з формули (3.8) використання `FM_DEVIATION = 61 КГц` дозволило отримати змінну частину `phase_step` рівну `sample << 13`. Іншими словами, для одержання змінної частини `phase_step` не потрібно використовувати жодну комбінаційну логіку, достатньо зсунути `sample` вліво на 13 розрядів.

Примітка. Для зсуву числа на фіксовану кількість розрядів вправо, або вліво не потрібна комбінаційна логіка. Зсув досягається за рахунок трасування провідників. На рис.3.15 наведено приклад зсуву 8-розрядного числа вліво на 3 розряди (`out = in << 3`) молодші 3 розряди результату заповнюються нулями.

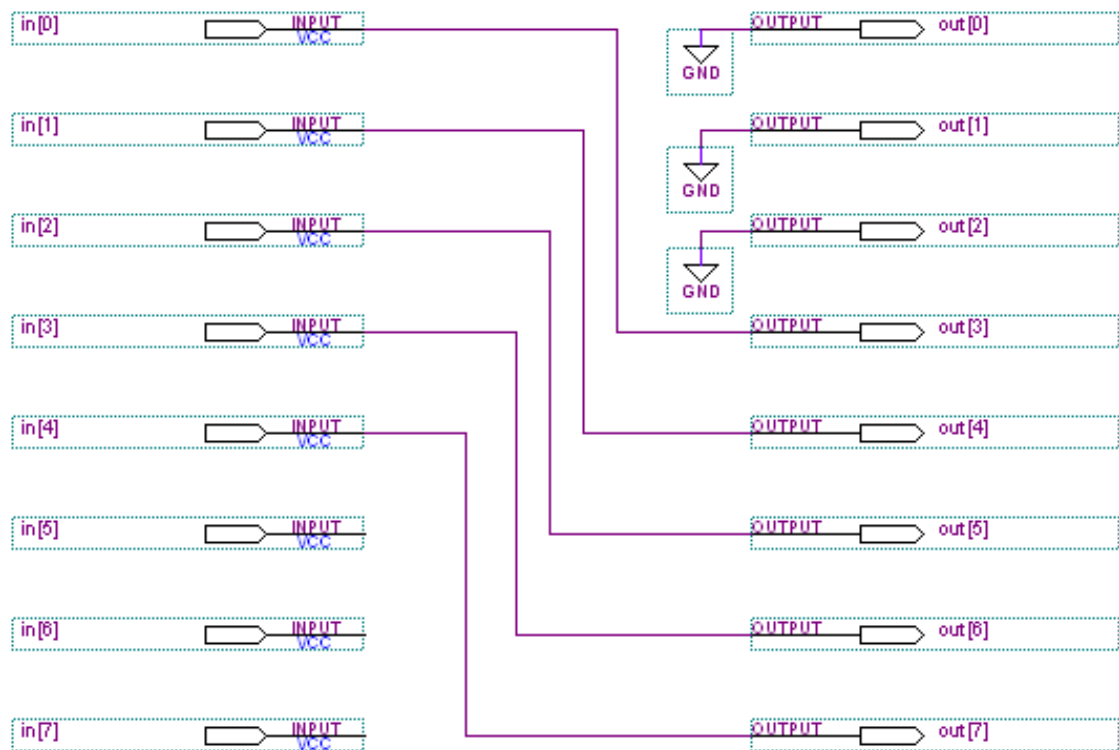


Рис.3.15 - Приклад зсуву 8-розрядного числа вліво на 3 розряди: $out = in \ll 3$

Перед тим, як перейти до аналізу Verilog коду цифрового FM передавача розглянемо процес передачі даних по інтерфейсу UART.

Для передачі даних по інтерфейсу UART достатньо одного провідника (за умови що провідники GND передавача і приймача з'єднані між собою). Для прийому даних також необхідний один провідник. Вихід UART передавача називається TX, а вхід UART приймача називається RX. З'єднання двох обчислювальних систем за допомогою UART в обох напрямках показано на рис.3.16. Зверніть увагу, що в інтерфейсі UART імпульсний сигнал синхронізації не передається. Тому інтерфейс називають асинхронним. Вихід TX UART-передавача підключають до входу RX UART-приймача.

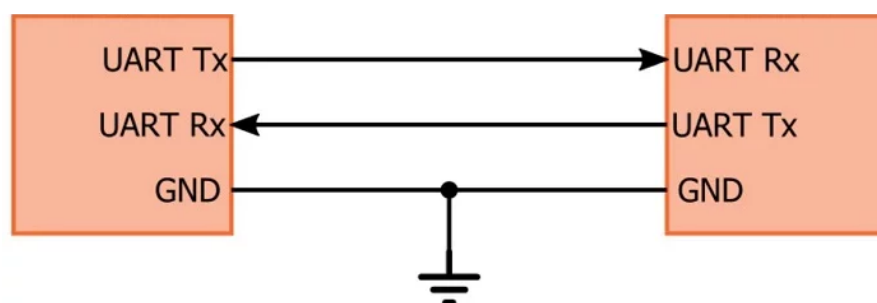


Рис.3.16 - З'єднання обчислювальних систем з використанням UART

Передавати та приймати дані по інтерфейсу UART з комп'ютера можна за допомогою USB-UART перетворювача, зовнішній вигляд якого показано на рис. 3.17. На цьому ж рисунку показаний колір контактів TX, RX та GND для USB-UART перетворювача. У багатьох USB-UART перетворювачах зазначені виводи явно підписані.



Рис.3.17 - Зовнішній вигляд USB-UART перетворювача

Послідовна передача бітів даних від UART-передавача до UART-приймача показана на рис.3.18. Зазвичай дані в UART передаються пакетами по 8 біт на пакет. Але в багатьох UART модулях є можливість налаштувати розмір поля даних в межах 6-9 біт. Окрім поля даних в UART пакеті присутні два біти керування - старт біт і стоп біт. Отже повний розмір пакету даних в такій конфігурації складає 10 біт. Можна налаштувати UART на передачу з двома стоп бітами, або за необхідності додати біт парності. Однак в даному прикладі ми не будемо використовувати ці можливості і скористаємося типовою конфігурацією: **8 біт даних, 1 старт біт, 1 стоп біт**. Також можлива конфігурація з двома стоп бітами.

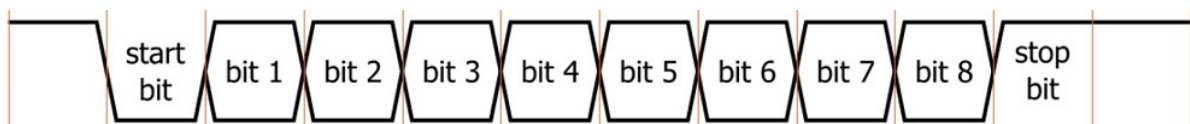


Рис.3.18 - Передача даних з використанням інтерфейсу UART

При відсутності передачі даних вихід UART-передавача приймає значення лог.1. Для початку передачі UART-передавач переводить значення виходу з лог. 1 в лог. 0. Це і є старт біт. Приймач детектує зміну значення на вході з лог. 1 на лог. 0 і розуміє, що прийшов старт біт, а це означає, що почалася передача нового пакету даних. Після передачі старт біту і бітів даних необхідно у стоп біті перевести вихід UART передавача в лог. 1, щоб мати можливість пізніше сформувати наступний старт біт.

UART-передавач і UART-приймач повинні бути налаштовані на однакову швидкість передачі. Швидкість передачі даних по інтерфейсу UART (bit rate, baud rate) вимірюють в бітах на секунду (bit per second, bps). По суті, це кількість бітів, що передається через інтерфейс UART за секунду. Якщо поділити одну секунду на швидкість передачі даних отримаємо тривалість передачі одного біту на обраній швидкості - рис.3.19. Швидкість передачі даних по інтерфейсу UART приймає ряд стандартних значень: 9600 bps , 19200 bps, 38400 bps, 57600 bps, 115200 bps, 230400 bps, 460800 bps, 921600 bps і т.д.

В даному прикладі швидкість UART з'єднання обрано рівною **230400 bps**.

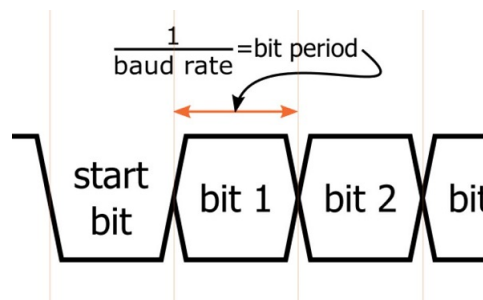


Рис.3.19 - Тривалість передачі одного біту, якщо відома швидкість передачі бітів через UART з'єднання (bit rate, baud rate)

Оскільки в даному проекті цифрового радіо аудіо файл передається з комп'ютера в FM передавач на швидкості 230400 біт за секунду, а один UART пакет складає 10 біт і містить 8 біт даних, це означає, що за умови безперервної передачі пакетів один за одним, частота прийому нових 8-бітних значень буде дорівнювати $230400/10 = 23040$ разів за секунду (23040 Гц). З такою частотою будуть надходити від комп'ютера нові цифрові коди, що представляють аудіо сигнал.

В лабораторній роботі ми будемо використовувати готовий UART приймач вихідний код якого наведений за посиланням [\[3.3\]](#).

Команди для передачі wav аудіо файлу з комп'ютера через USB-UART перетворювач наведені розділі 3.3.4 лабораторної роботи.

А зараз розглянемо вихідний код цифрового FM передавача, що наведений в лістингу 3.5.

Лістинг 3.5 - Вихідний код цифрового FM передавача на мові Verilog

```
module fm_transmitter(i_clk, i_rst_n, i_rx, o_pll_locked, o_fm);

input    i_clk;
inout    i_rst_n;
input    i_rx;
output    o_pll_locked;
output    o_fm;

parameter    NCO_PHASE_STEP_CARRIER = 32'h5C28_F5C2;

wire    [7:0]    uart_dat;
reg      [7:0]    sample;

reg      [31:0]    nco_phase;
reg      [31:0]    nco_phase_step;
wire      [31:0]    nco_phase_step_carrier = NCO_PHASE_STEP_CARRIER;
wire      [31:0]    nco_phase_step_deviation = {{11{sample[7]}},sample,13'b0};

mypll pll_inst(.inclk0 (i_clk),
               .c0 (sys_clk), // 250 MHz clock
               .locked (o_pll_locked)
               );

simple_uart_receiver uart_rec(.i_clk (sys_clk),
                             .i_rst_n (i_rst_n),
                             .i_rx (i_rx),
                             .o_dat (uart_dat),
                             .o_dat_vld ()
                             );

always @(posedge sys_clk)
    sample <= uart_dat;

always @(posedge sys_clk)
    nco_phase_step <= nco_phase_step_carrier + nco_phase_step_deviation;

always @(posedge sys_clk)
    nco_phase <= nco_phase + nco_phase_step;

assign o_fm = nco_phase[31];

endmodule
```


Схема описана в лістингу 3.5. тактується від сигналу синхронізації `sys_clk` з частотою 250 МГц, який створюється за допомогою PLL. Загальні відомості про PLL наведено в розділі 3.2.6. Створення і налаштування PLL в САПР Quartus Prime описано в розділі 3.3.1.

Якщо блок PLL вже створено, налаштовано і додано до файлів проекту, використовувати його у синтезованому вихідному коді на мові Verilog можна створивши екземпляр відповідного PLL блоку (ім'я `mypll` було визначено під час створення блоку PLL):

```
mypll pll_inst (.inclk0 (i_clk),
                .c0 (sys_clk), // 250 MHz clock
                .locked (o_pll_locked)
                );
```

На вхід PLL `inclk0` подається сигнал синхронізації `i_clk`, з частотою 50 МГц, отриманий від тактового генератора на налагоджувальній платі. З виходу тактової частоти `c0` отримуємо сигнал синхронізації схеми цифрового FM передавача, що має частоту 250 МГц. Вихід `locked` варто підключити до світлодіоду для індикації коректного функціонування PLL.

Створення екземпляру модуля UART приймача:

```
simple_uart_receiver uart_rec (.i_clk (sys_clk),
                              .i_rst_n (i_rst_n),
                              .i_rx (i_rx),
                              .o_dat (uart_dat),
                              .o_dat_vld ()
                              );
```

На даному етапі UART-приймач можна сприймати, як готовий блок. За бажанням з вихідним кодом UART-приймача можна ознайомитись за посиланням [\[3.3\]](#).

Для підвищення тактової частоти дані з виходу UART приймача записуються в регістр `sample`:

```
always @(posedge sys_clk)
    sample <= uart_dat;
```

Реалізація NCO:

```
always @(posedge sys_clk)
    nco_phase <= nco_phase + nco_phase_step;
```

Як видно з формули (3.8) складова доданку `nco_phase_step` включає постійну складову `nco_phase_step_carrier` і змінну складову `nco_phase_step_deviation`:

```
always @(posedge sys_clk)
    nco_phase_step <= nco_phase_step_carrier + nco_phase_step_deviation;
```

Зверніть увагу, що `nco_phase` та `nco_phase_step` реалізовані у вигляді синхронних по фронту регістрів.

Постійна складова `nco_phase_step_carrier` з формули (3.8) визначається так:

```
parameter NCO_PHASE_STEP_CARRIER = 32'h5C28_F5C2;

wire [31:0] nco_phase_step_carrier = NCO_PHASE_STEP_CARRIER;
```

Формування `nco_phase_step_deviation` з формули (3.8):

```
wire [31:0] nco_phase_step_deviation = {{11{sample[7]}},sample,13'b0};
```

За допомогою оператора конкатенації `{}` утворюється нова шина, молодші 13 бітів якої дорівнюють нулю, далі йдуть 8 бітів `sample`, а решта 11 бітів заповнюються значенням старшого знакового біту `sample[7]`. Така операція еквівалентна зсуву `sample` на 13 розрядів вліво, як вимагає формула (3.8).

Якщо 8-розрядне значення `sample` зсунути вліво на 13 розрядів, отримаємо 21-розрядний результат. Але `nco_phase_step_deviation` має розрядність 31 біт. Тому необхідно розширити одержаний 21-розрядний результат до 31 розрядів. **Знакові числа розширюються до більшої розрядності старшим знаковим бітом.** Іншими словами, всі додаткові старші біти, що виникають під час розширення розрядності заповнюються значенням старшого розряду розширюваного числа. В нашому випадку це старший розряд `sample`.

Заповнення 11-ти бітів значенням `sample[7]` здійснюється за допомогою **оператора реплікації мови Verilog: `{11{sample[7]}}`**. В операторі реплікації на мові Verilog всередині внутрішніх фігурних скобок вказуємо провідник, або шину, які необхідно продублювати кілька разів, а зліва від внутрішніх фігурних скобок вказуємо скільки разів необхідно продублювати вміст скобок.

Високочастотний FM модульований радіосигнал знімається зі старшого розряду регістру NCO та підключається до зовнішнього провідника-антени.

```
assign o_fm = nco_phase[31];
```

Результат синтезу вихідного коду з лістингу 3.5 наведено на рис.3.20.

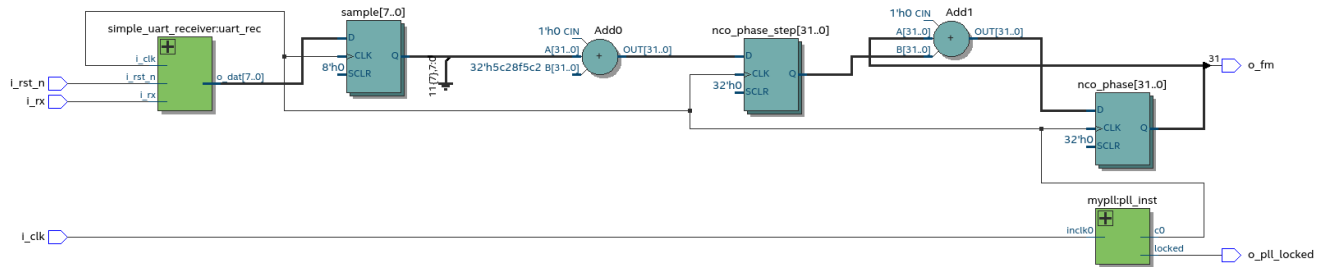


Рис.3.20 - Результат синтезу вихідного коду з ліст. 3.5 для цифрового FM передавача

3.3 Практична частина

3.3.1 Створення блоку ФАПЧ (PLL) в Quartus Prime

Загальні відомості про блок PLL наведено в розділі 3.2.6 лабораторної роботи.

Для створення блоку PLL в Quartus Prime оберіть пункт меню Tools -> IP Catalog. На панелі зправа з'явиться вікно вибору IP блоків (IP Cores) з бібліотеки Intel FPGA - рис.3.21.

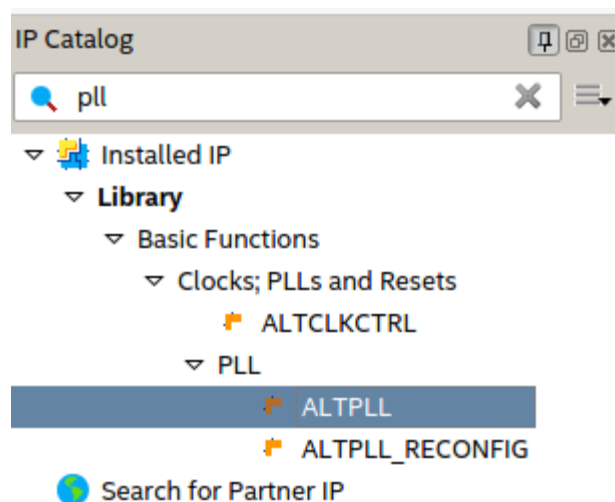


Рис.3.21 - Вікно вибору IP блоків (IP Cores)

В полі пошуку вікна IP Catalog напишіть PLL та натисніть Enter. Оберіть компонент ALTPLL і ще раз натисніть Enter. З'явиться вікно вибору імені створюваного IP блоку PLL - рис.3.22.

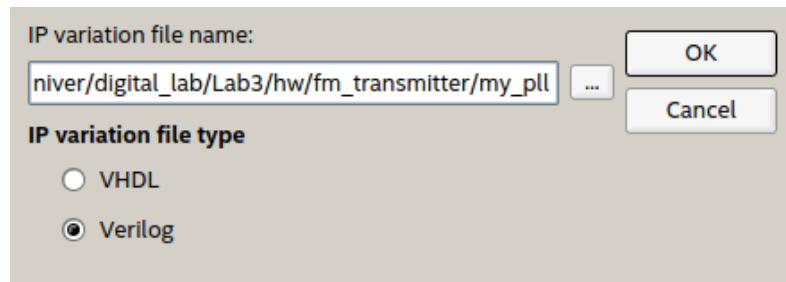


Рис.3.22 - Вікно вибору імені створюваного блоку PLL

У вікні, що показане на рис.3.22, після імені проекту (в даному прикладі ім'я проекту fm_transmitter) поставте / та напишіть ім'я блоку PLL, що створюється. Це ім'я потім необхідно буде використати під час створення екземпляру блоку PLL у вихідному коді на мові Verilog - лістинг 3.5. Приклад імені блоку PLL наведено на рис.3.22. Тип файлів з описом PLL залиште у значенні Verilog.

Після вибору імені блоку PLL і мови опису PLL файлів натисніть OK. Відкриється перша сторінка налаштувань блоку PLL - рис.3.23.

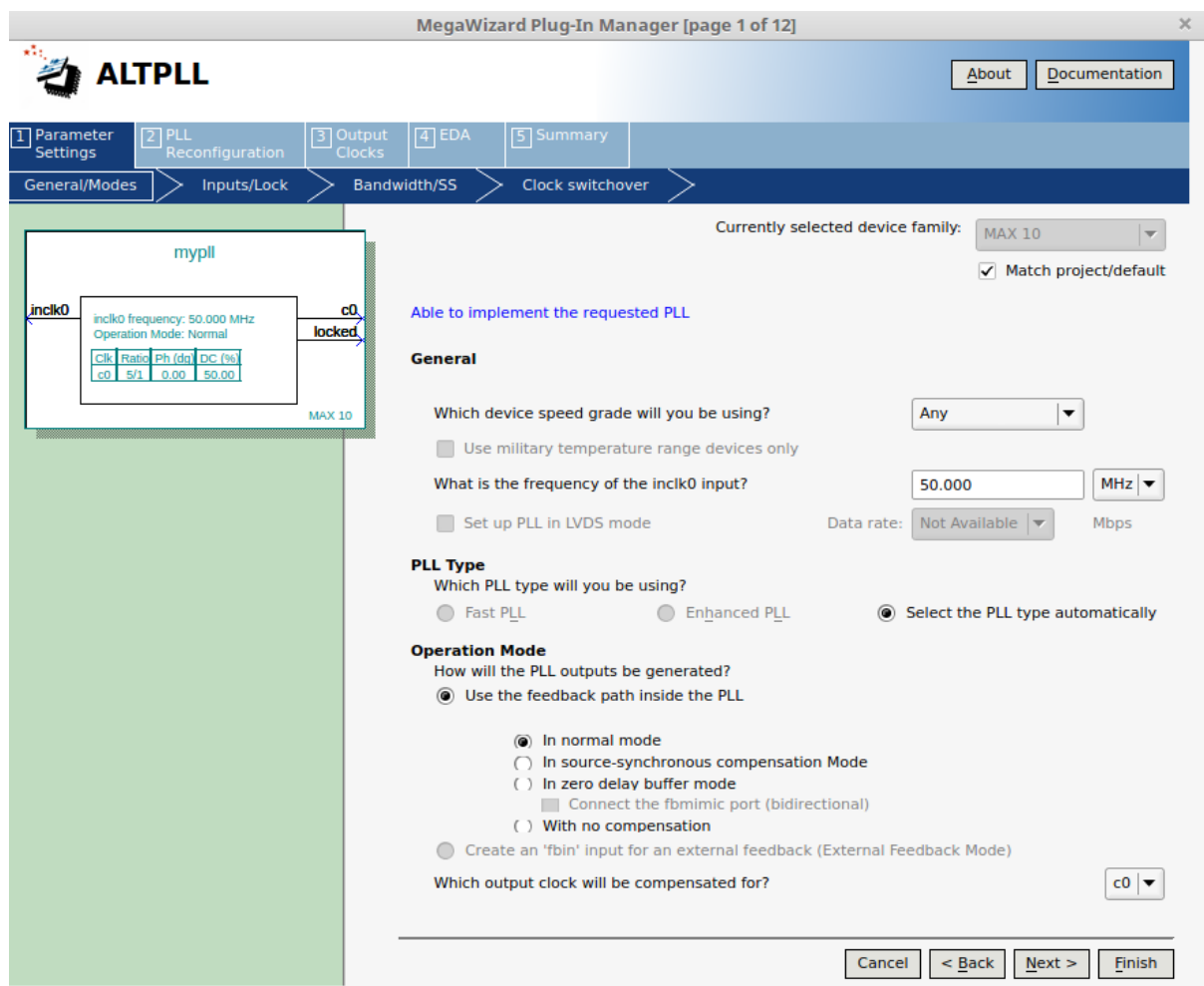


Рис.3.23 - Перша сторінка налаштувань PLL (визначення вхідного тактового сигналу)

На першій сторінці налаштувань блоку PLL, що зображена на рис.3.23, більшість параметрів залиште у значеннях за замовчуванням. Єдине, що треба змінити - це вказати реальну частоту вхідного сигналу синхронізації блоку PLL в полі “What is the frequency of the inclk0 input”. Наприклад, в лістингу 3.5 на вхід блоку PLL подається сигнал синхронізації з частотою 50 МГц, від зовнішнього кварцевого генератора на друкованій платі. Для можливості використання створюваного блоку PLL в лістингу 3.5. поставимо значення поля “What is the frequency of the inclk0 input” рівним 50 MHz. Після цього натисніть Next для переходу на другу сторінку налаштувань блоку PLL - рис.3.24.

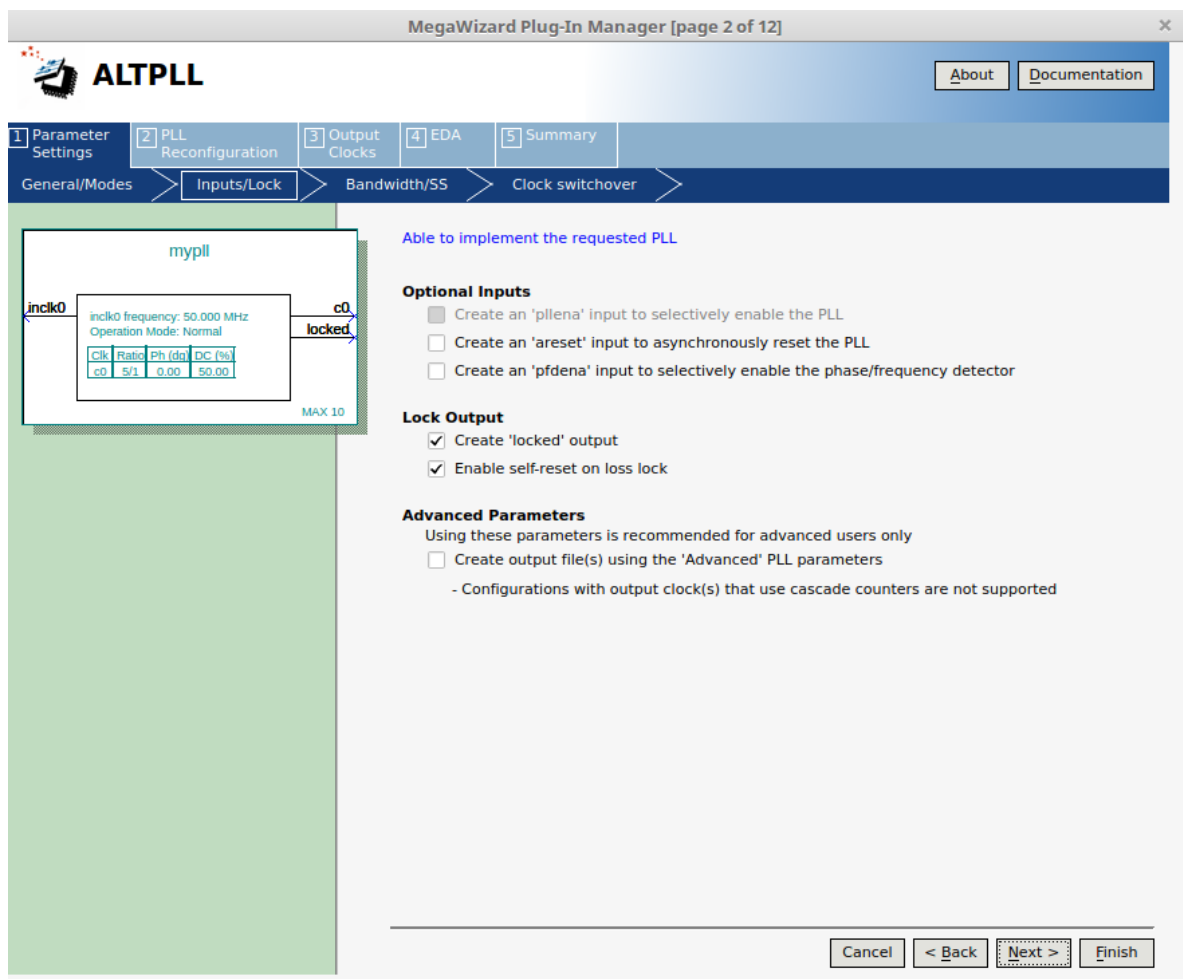


Рис.3.24 - Друга сторінка налаштувань PLL (налаштування зкидання)

На другій сторінці налаштувань блоку PLL зніміть вибір з поля “Create an ‘areset’ input to asynchronously reset the PLL” та оберіть поле “Enable self-reset on loss lock”, як показано на рис.3.24. Зроблені налаштування усувають окремий вхід асинхронного зкидання та задають автоматичне зкидання блоку PLL.

Натисніть 4 рази підряд кнопку Next, залишивши значення за замовчуванням на сторінках 3-5 налаштувань PLL. Відкриється шоста сторінка налаштувань PLL - рис.3.25.

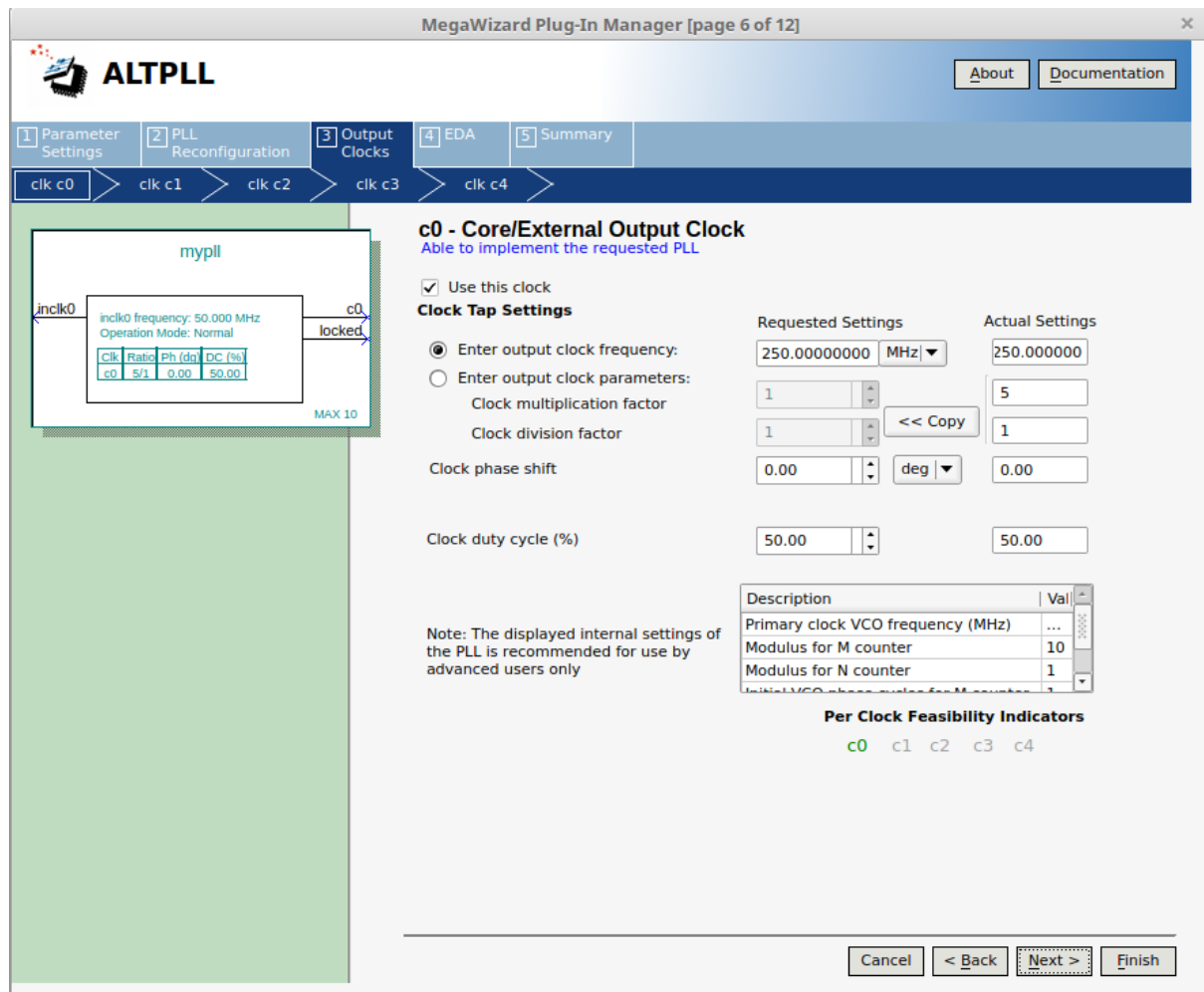


Рис.3.25 - Шоста сторінка налаштувань PLL (налаштування вихідного тактового сигналу)

На сторінці налаштувань PLL, що зображена на рис.3.25, задайте параметри вихідного сигналу синхронізації c0 блоку PLL.

В полі “Enter output clock frequency” введіть необхідну частоту вихідного сигналу синхронізації c0. Наприклад, PLL, що використовується в лістингу 3.5 створює сигнал синхронізації частотою 250 МГц. Для застосування створюваного блоку PLL в прикладі з лістингу 3.5 введіть в поле “Enter output clock frequency” значення 250 МГц. За необхідності в полі “Clock duty cycle (%)” можна задати коефіцієнт заповнення імпульсів вихідного сигналу синхронізації c0.

Один блок PLL може створювати до 5-ти сигналів синхронізації “прив’язаних” до вхідного сигналу `inclk0`. При цьому всередині мікросхеми FPGA може бути розміщено кілька блоків PLL. Натискаючи кнопку Next потрапляємо у вікна налаштувань вихідних сигналів синхронізації `c1-c4`, що аналогічні до налаштувань виходу `c0`. За необхідності можна активувати певний вихід поставивши галочку “Use this clock” і налаштувати такий вихід на необхідну частоту.

Вигляд 12-ї сторінки налаштувань блоку PLL показано на рис.3.26.

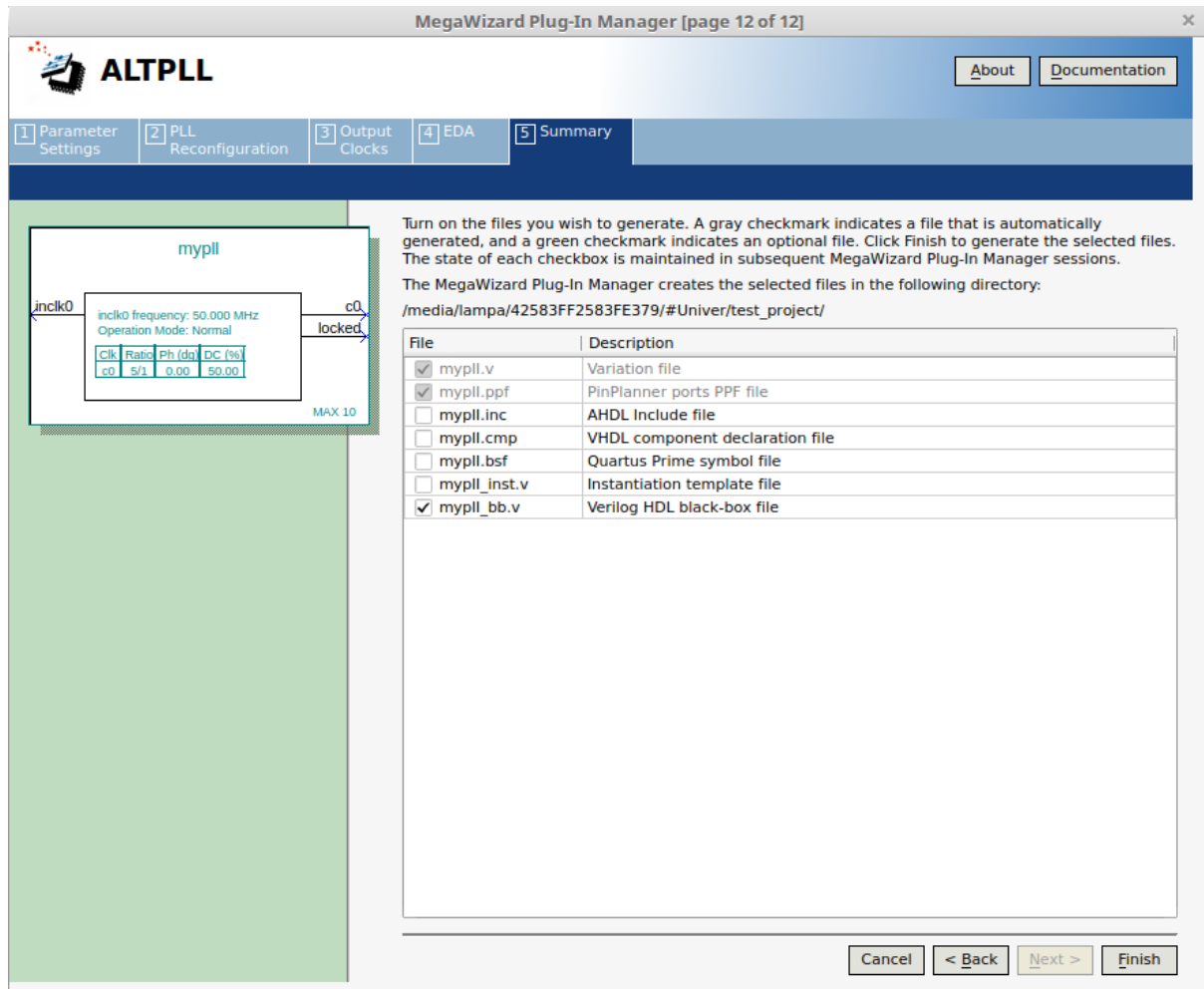


Рис.3.26 - Дванадцята сторінка налаштувань PLL (вибір файлів з описом PLL)

За необхідності, в останньому вікні налаштувань PLL (рис.3.26) можна обрати додаткові файли, що будуть створені разом з блоком PLL. Цікавим може бути створення файлу-символу (з розширенням `*.bsf`) для використання у графічних схемах. Або файлу `*_inst.v` з прикладом створення екземпляру блоку PLL у вихідному коді на мові Verilog.

Для завершення процедури створення блоку PLL натисніть Finish.

Після генерації PLL блоку Quartus Prime створить файл my_pll.qip, що містить всю необхідну інформацію про згенерований PLL блок і запропонує додати цей файл до проекту - рис.3.27. Необхідно погодитися на таку пропозицію натиснувши Yes. Також в директорії проекту будуть створені файли mypll.ppf, mypll.sip та mypll.v

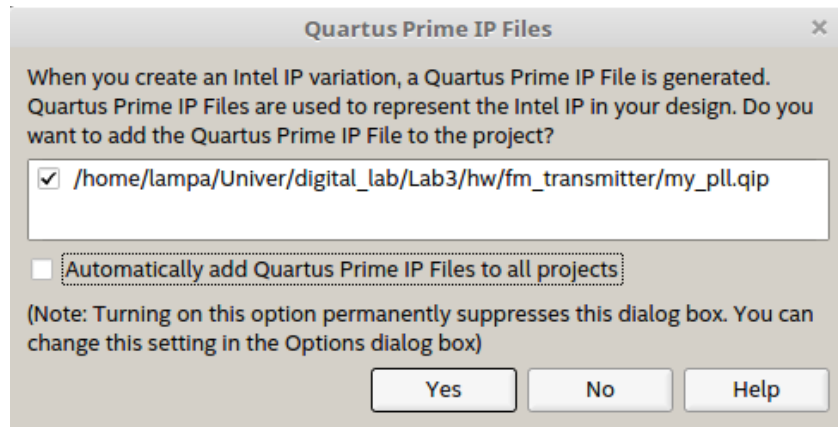


Рис.3.27 - Вікно з пропозицією додати *.qip файл для створеного PLL блоку до проекту

Якщо необхідно додати файли вже створеного PLL блоку з іменем my_pll до проекту, зкопіюйте в директорію проекту файли PLL блоку (my_pll.qip, mypll.ppf, mypll.sip, mypll.v) і додайте до проекту файл my_pll.qip

3.3.2 Визначення обмежень на синтез (Synthesis Constraints) в Quartus Prime

В розділі 3.2.7 наведено метод оцінки максимальної тактової частоти для синхронної цифрової схеми. А в розділі 3.2.8 в загальному описаний статичний аналіз затримок (Static Timing Analysis, STA) у цифрових схемах.

Модуль STA в Quartus Prime називається TimeQuest Timing Analyzer. TimeQuest дозволяє оцінити максимально можливу частоту сигналу синхронізації цифрової схеми, що проектується на базі мікросхем Intel FPGA. Можливо задати максимальну частоту сигналу синхронізації і Quartus Prime під час синтезу буде намагатися шляхом оптимізацій створити синхронну схему, що коректно функціонує на заданій максимальній частоті. Також можна задати необхідні затримки між вхідними виводами FPGA та входами даних синхронних по фронту D-тригерів (Input Delay). Або задати необхідні затримки між виходами синхронних по фронту D-тригерів та вихідними виводами FPGA мікросхем (Output Delay). Зазначені обмеження у вигляді максимально можливої частоти сигналу синхронізації, або затримок між D-тригерами і

вхідними/вихідними виводами FPGA називають обмеженнями на синтез (Synthesis Constraints).

Для коректної роботи модуля STA і правильного синтезу синхронних по фронту цифрових схем в Quartus Prime важливо задати всі сигнали синхронізації вашого проекту та інші обмеження на синтез в sdc файлі. Для цього необхідно створити файл з розширенням *.sdc (наприклад, файл constraints.sdc), додати його до файлів проекту і всередині sdc файлу визначити необхідні обмеження на синтез.

Для налагоджувальних плат з мікросхемами Intel FPGA, що використовуються в лабораторних роботах, створено базові sdc файли, в які додано сигнали синхронізації між FPGA мікросхемою та друкованою платою. Зазначені sdc файли можна знайти за посиланням [3.4] в каталогах, що відповідають налагоджувальним платам.

Приклад sdc файлу для налагоджувальної плати DE0-CV можна переглянути в лістингу 3.6.

Лістинг 3.6 - Приклад файлу “constraints.sdc” з обмеженнями на синтез для налагоджувальної плати DE0-CV

```
#####
# Create Clock
#####
create_clock -period 20 [get_ports CLOCK2_50]
create_clock -period 20 [get_ports CLOCK3_50]
create_clock -period 20 [get_ports CLOCK_50]
create_clock -period 20 [get_ports CLOCK4_50]
#####
# Create Generated Clock
#####
derive_pll_clocks
#####
# Set Clock Uncertainty
#####
derive_clock_uncertainty
```

Як видно з лістингу 3.6, коментарі в sdc файлах починаються з символу #.

Для визначення сигналу синхронізації використовують команду **create_clock**. Ключ **-period** використовують для визначення мінімально можливого періоду сигналу синхронізації, або максимально можливої тактової частоти. Якщо після ключа **-period** поставити число, наприклад **-period 20**, то таким чином ви задасте мінімально допустимий період тактового сигналу в наносекундах. Якщо ж після ключа **-period**

написати <frequency>MHz, наприклад **-period 300MHz**, то таким чином ви задасте максимально можливу частоту сигналу синхронізації. Після **get_ports** необхідно вказати ім'я вхідного порта Verilog модуля, що відповідає сигналу синхронізації, який описується.

За допомогою **create_clock** необхідно визначити в sdc файлі всі сигнали синхронізації, що використовуються у вашому проекті, окрім сигналів синхронізації, що створюються з використанням PLL.

Для інформування TimeQuest і Quartus Prime про сигнали синхронізації створені за допомогою PLL достатньо додати в sdc файл команду **derive_pll_clocks**.

Щоб розрахувати clock skew для тригерів, які використовуються у вашому проекті та врахувати тремтіння фази сигналу синхронізації (jitter), необхідно використати команду **derive_clock_uncertainty**. Детальніше про clock skew написано в розд. 3.2.7

Окрім визначення максимальних частот сигналів синхронізації в sdc файлах можна задати затримки через комбінаційну логіку, виключити певні блоки комбінаційної логіки зі статичного аналізу затримок, задавати затримки між виводами FPGA та входами/виходами D-тригерів (актуально при підключенні до FPGA АЦП, ЦАП та швидкісних інтерфейсів на зразок USB, Ethernet, HDMI, PCIe, SATA і т.д.). Деталі роботи з TimeQuest описані за посиланням [\[3.2\]](#).

3.3.3 Оцінка затримок і максимальної тактової частоти в TimeQuest

Розглянемо оцінку затримок і максимальної тактової частоти в TimeQuest Timing Analyzer на прикладі подільника частоти з лістингу 3.1.

Створіть проект в Quartus Prime, додайте в проект файл з вихідним кодом з лістингу 3.1 і у якості Top Level Entity визначте ім'я модуля подільника частоти const_div. Параметр DIV_BY повинен приймати значення порядку кількох тисяч для одержання в результаті синтезу відносно великої схеми.

Додайте до проекту sdc файл, вміст якого визначений в лістингу 3.6., але максимальну частоту сигналу синхронізації **CLOCK_50** задайте рівну 500 МГц:

```
create_clock -period 500MHz [get_ports CLOCK_50]
```

Запустіть компіляцію проекту для створення конфігураційного файлу FPGA. У вікні результатів компіляції в розділі TimeQuest Timing Analyzer ви зможете знайти оцінку максимально можливої частоти сигналу синхронізації створеної схеми для різних значень температури кристалу (0°C та 85°C) - рис.3.28.

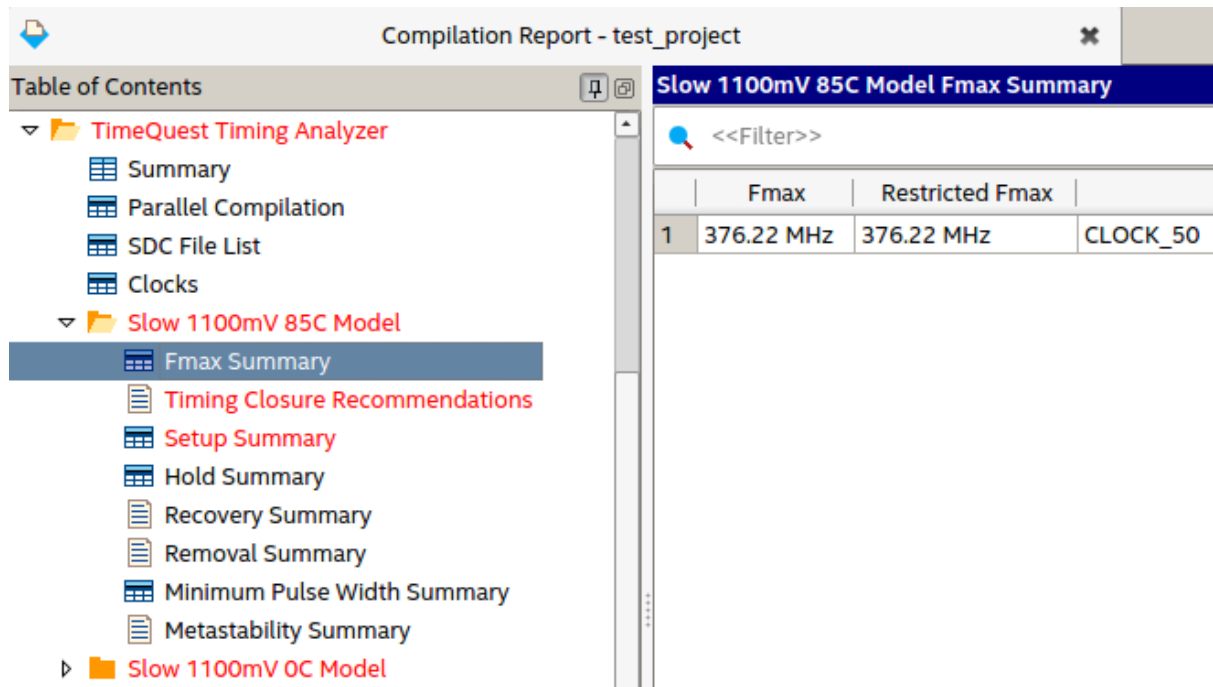


Рис.3.28 - Вікно результатів компіляції в Quartus Prime.

Розділ оцінки максимальної тактової частоти

Виконайте пункт головного меню Tools -> TimeQuest Timing Analyzer. Відкриється вікно TimeQuest. Нас цікавить вікно керування, зображене на рис.3.29.

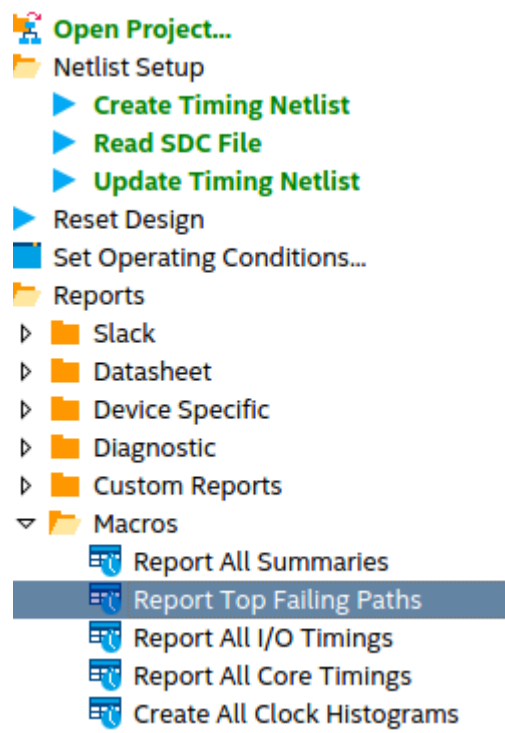


Рис.3.29 - Вікно керування TimeQuest Timing Analyzer

У вікні керування TimeQuest виконайте послідовно три команди: **Create Timing Netlist**, **Read SDC File**, **Update Timing Netlist**. Для виконання команди необхідно двічі клікнути по ній. Команда **Create Timing Netlist** запускає розрахунок затримок розповсюдження сигналу через комірки комбінаційної логіки та з'єднання між ними всередині FPGA. Команда **Read SDC File** приводить до зчитування обмежень на синтез з sdc файлу. Команда **Update Timing Netlist** оновлює затримки розповсюдження сигналу з урахуванням обмежень на синтез.

Після виконання зазначених вище команд, запустіть операцію “Report Top Failing Paths”, що знаходиться в розділі Macros вікна керування TimeQuest. Отримаємо перелік блоків комбінаційної логіки між регістрами, затримка яких перевищує максимально допустиму. Максимально допустима затримка комбінаційної логіки між регістром-джерелом і регістром-приймачем розраховується з формули (3.4). Результат виконання операції “Report Top Failing Paths” наведено на рис.3.30. В полі From Node вказано ім'я регістру-джерела даних для комб. логіки, а в полі To Node вказано ім'я регістра-приймача даних з виходу комб. логіки.

	Slack	From Node	To Node	Launch Clock	Latch Clock	ations	Clock Skew	Data Delay
25	-0.361	cnt[4]	cnt[5]	CLOCK_50	CLOCK_50	2.000	-0.090	2.171
26	-0.354	cnt[4]	cnt[5]...ICATE	CLOCK_50	CLOCK_50	2.000	-0.086	2.168
27	-0.346	cnt[2]	cnt[1]...ICATE	CLOCK_50	CLOCK_50	2.000	-0.093	2.153
28	-0.339	cnt[2]	cnt[10]	CLOCK_50	CLOCK_50	2.000	-0.089	2.150
29	-0.339	cnt[0]	cnt[3]...ICATE	CLOCK_50	CLOCK_50	2.000	-0.089	2.150
30	-0.333	cnt[0]	cnt[3]	CLOCK_50	CLOCK_50	2.000	-0.081	2.152
31	-0.330	cnt[1]	cnt[1]...ICATE	CLOCK_50	CLOCK_50	2.000	-0.095	2.135
32	-0.330	cnt[...CATE	cnt[1]...ICATE	CLOCK_50	CLOCK_50	2.000	-0.087	2.143
33	-0.325	cnt[1]	cnt[1]	CLOCK_50	CLOCK_50	2.000	-0.087	2.138

Рис.3.30 - Вікно з результатами виклику “Report Top Failing Paths”

Якщо клікнути правою кнопкою миші на певному “проблемному” шляху з набору на рис.3.30 і в контекстному меню натиснути “Report Worst-Case Path”, отримаємо аналіз затримок розповсюдження сигналу через елементи комбінаційної логіки і з'єднання між ними для цього шляху. Зазначені затримки перелічені на вкладці Data Path, рис.3.31. При цьому комірки комбінаційної логіки всередині FPGA позначені, як CELL, а з'єднання між компонентами всередині FPGA позначені, як IC. Значення Slack (або Clock Setup Slack) розраховується за формулою: $Slack = T_{min} - t_{cq} - t_s - t_{cs} - t_{max_comb}$. Якщо Slack приймає від'ємне значення, це означає, що затримка комбінаційної логіки t_{max_comb} занадто велика, щоб забезпечити стабільне функціонування при заданому T_{min} .

Command Info		Summary of Paths						
	Slack	From Node	To Node	Launch Clock	Latch Clock	lationsl	Clock Skew	Data Delay
1	-0.361	cnt[4]	cnt[5]	CLOCK_50	CLOCK_50	2.000	-0.090	2.171

Path #1: Setup slack is -0.361 (VIOLATED)

Path Summary	Statistics	Data Path	Waveform	Extra Fitter Information
--------------	------------	-----------	----------	--------------------------

Data Arrival Path

	Total	Incr	RF	Type	Fanout	Location	Element
1	0.000	0.000					launch edge time
2	3.656	3.656					clock path
1	3.656	3.656	R				clock network delay
3	5.827	2.171					data path
1	3.656	0.000		uTco	1	FF_X47_Y4_N14	cnt[4]
2	3.656	0.000	RR	CELL	1	FF_X47_Y4_N14	cnt[4]q
3	3.926	0.270	RR	IC	3	LABCELL_X47_Y4_N12	Add0~33 datab
4	5.001	1.075	RR	CELL	1	LABCELL_X47_Y4_N12	Add0~33 cout
5	5.001	0.000	RR	IC	2	LABCELL_X47_Y4_N15	Add0~29 cin
6	5.154	0.153	RR	CELL	1	LABCELL_X47_Y4_N15	Add0~29 sumout
7	5.462	0.308	RR	IC	1	LABCELL_X48_Y4_N54	cnt~4 dataf
8	5.546	0.084	RR	CELL	2	LABCELL_X48_Y4_N54	cnt~4 combout
9	5.546	0.000	RR	IC	1	FF_X48_Y4_N56	cnt[5]d
10	5.827	0.281	RR	CELL	1	FF_X48_Y4_N56	cnt[5]

Рис.3.31 - Вікно з результатом виклику “Report Worst-Case Path”

За бажанням комірки комбінаційної логіки і зв’язки між ними, наведені на вкладці Data Path, можна переглянути у вигляді схеми в Technology Map Viewer. Для цього необхідно вибрати шлях через комбінаційну логіку від тригера джерела до тригера-приймача на вкладці “Summary of Paths” (рис.3.31), клікнути на цьому шляху правою кнопкою миші і в контекстному меню обрати пункт Locate Path -> Locate in Technology Map Viewer. В результаті в Technology Map Viewer буде відображено обрану комбінаційну логіку між тригером-джерелом і тригером-приймачем - рис.3.32.

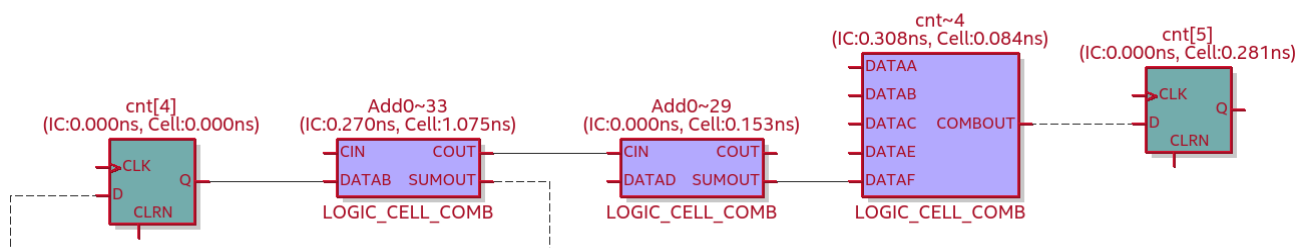



Рис.3.32 - Відображення комбінаційної схеми з критично великою затримкою в Technology Map Viewer

Над кожною коміркою комбінаційної логіки на рис.3.32 наведено затримку цієї комірки (наприклад, Cell: 1.075ns) та затримку з'єднувальних провідників між виходом попередньої комірки і входом даної комірки (наприклад, IC:0.270ns). Зверніть увагу, що всередині FPGA затримки комірок комбінаційної логіки можуть мати такий же порядок, що і затримки з'єднань.

Існує можливість переглянути розміщення елементів комбінаційної логіки між тригером-джерелом і тригером-приймачем на кристалі FPGA в Chip Planner. Для цього необхідно вибрати шлях через комбінаційну логіку від тригера джерела до тригера-приймача на вкладці "Summary of Paths" (рис.3.31), клацнути та цьому шляху правою кнопкою миші і в контекстному меню обрати пункт Locate Path -> Locate in Chip Planner. В результаті в Chip Planner буде відображено розміщення обраної комбінаційної логіки між тригером-джерелом і тригером-приймачем на кристалі FPGA - рис.3.33. Результат у збільшеному масштабі зображено на рис.3.34. Змінювати масштаб відображення можна кнопкою , або за допомогою пунктів головного меню Chip Planner: View -> Zoom In, View -> Zoom Out, View -> Zoom, або View -> Fit Selection In Window. На рис.3.34-3.35 стрілками показано з'єднання між комірками комбінаційної логіки, що розміщені на кристалі FPGA. Якщо вибрати пункт головного меню View -> Show Delays, біля кожної стрілки буде наведено затримку відповідного з'єднання. Якщо вибрати пункт головного меню View -> Highlight Routing, червоним кольором будуть відображені з'єднувальні ресурси, що використовуються для сполучення між комірками. При наведенні курсору на певне червоне з'єднання, з'являється підказка з інформацією які елементи (тригери, або комірки комб. логіки) сполучаються за допомогою цього з'єднання.

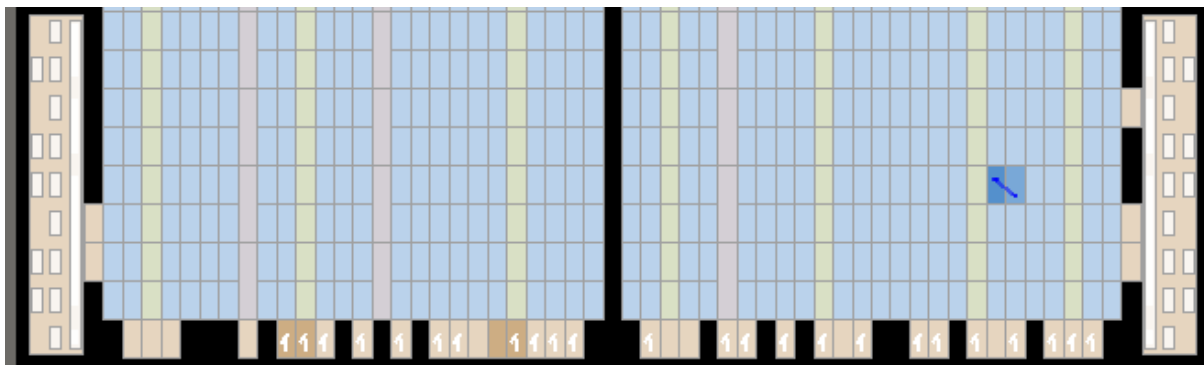


Рис.3.33 - Відображення комбінаційної схеми з критично великою затримкою
в Chip Planner

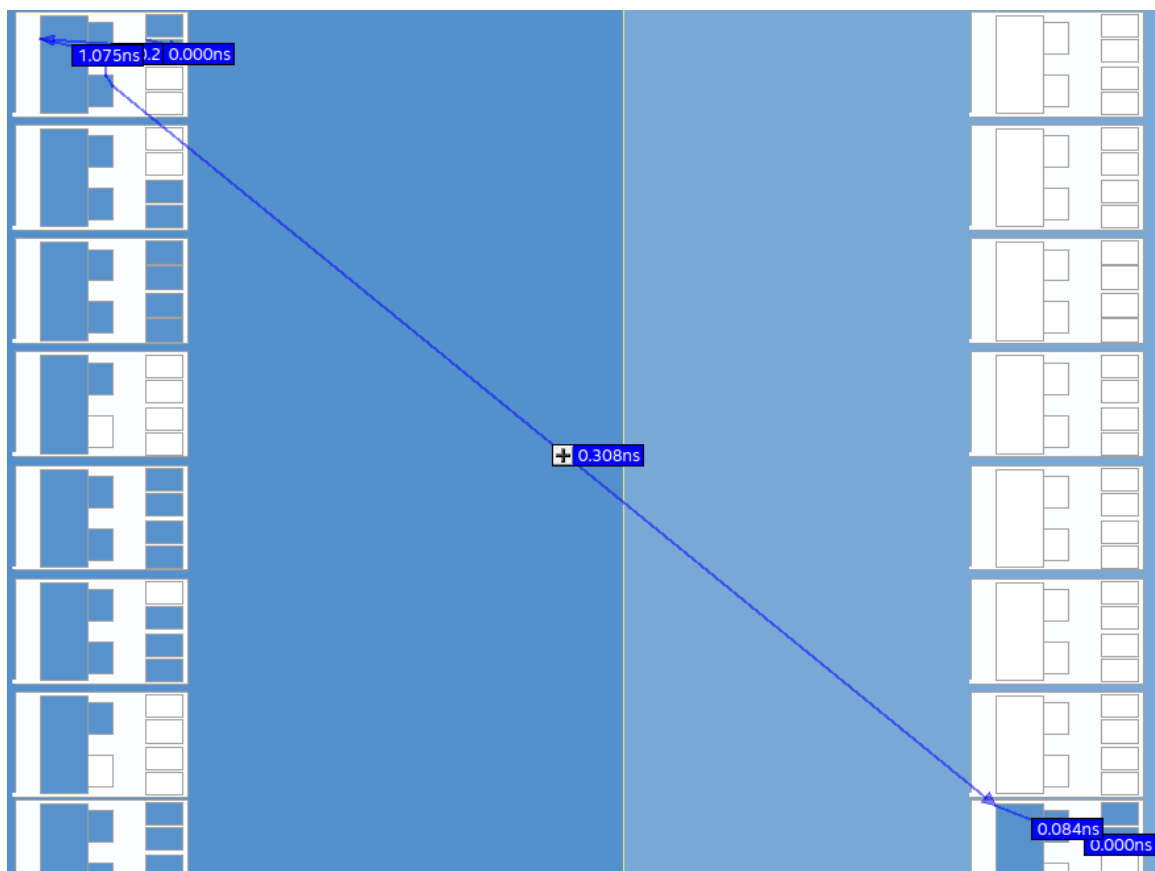


Рис.3.34 - Відображення комбінаційної схеми з критично великою затримкою в Chip Planner (збільшений масштаб)

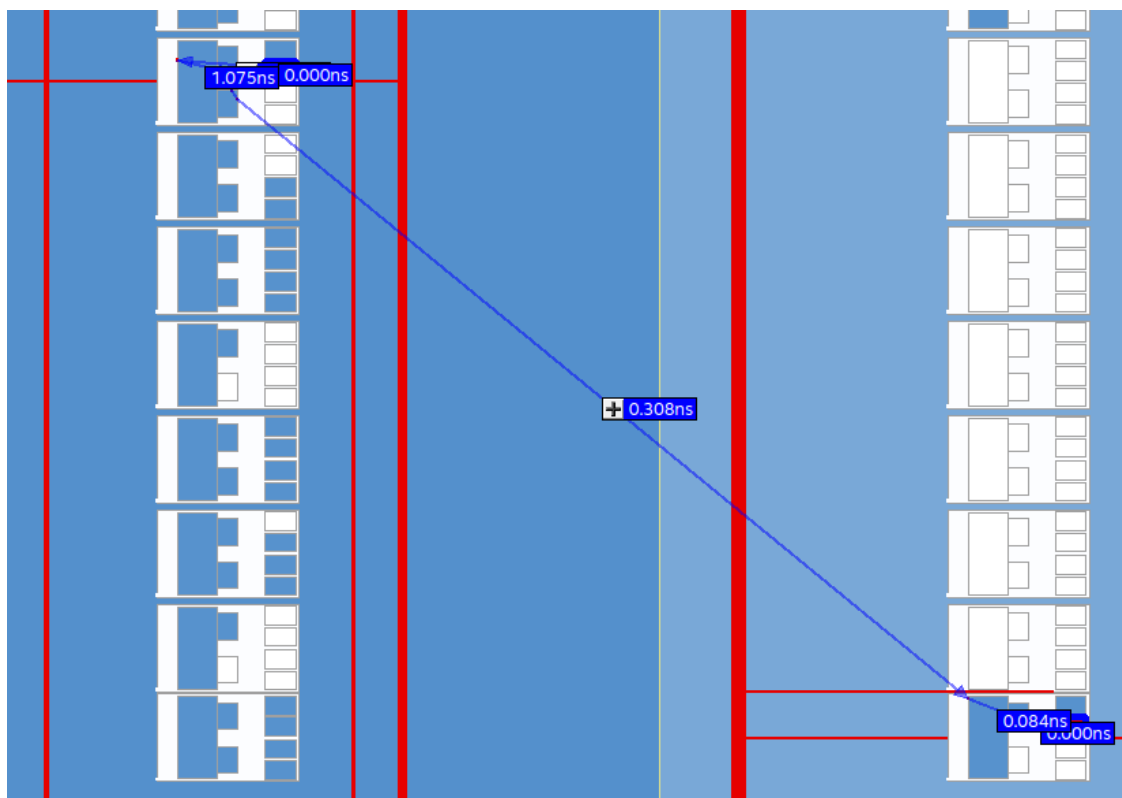


Рис.3.35 - Відображення комбінаційної схеми з критично великою затримкою в Chip Planner (збільшений масштаб, показано з'єднувальні ресурси)

Використовуючи інформацію, що наведена на рис.3.31-3.35 інженер може зрозуміти, які елементи комбінаційної логіки між регістрами обмежують зростання тактової частоти і оптимізувати структуру схеми.

3.3.4 Передача аудіо файлів в цифровий FM передавач за допомогою UART

В проекті цифрового FM передавача аудіо файл в форматі wav передається в налагоджувальну плату з комп'ютера по інтерфейсу UART з використанням USB-UART перетворювача. Прийом цифрових кодів, що представляють значення напруги аудіо сигналу, через інтерфейс UART описано в розділі 3.2.10.

Для передачі аудіо файлу в форматі wav через інтерфейс UART в операційній системі Linux необхідно виконати наступні дві команди в командному рядку (терміналі) Linux:

```
stty -F /dev/ttyUSB0 230400 cs8 cstopb -parenb  
cat SleepAway8.wav > /dev/ttyUSB0
```

Команда **stty** використовується для налаштування параметрів UART з'єднання в Linux. Параметр **/dev/ttyUSB0** задає ім'я пристрою USB-UART перетворювача в каталозі **/dev**. Більшість USB-UART перетворювачів мають ім'я **ttyUSB** після якого зазначений номер підключеного до комп'ютера USB-UART перетворювача. Якщо ви використовуєте всього один USB-UART перетворювач, з великою вірогідністю його ім'я буде **ttyUSB0**. Параметр **230400** задає швидкість UART з'єднання в бітах за секунду. Параметр **cs8** визначає, що поле даних UART пакету містить 8 біт. Параметр **cstopb** означає, що необхідно використовувати два стоп біти. Для використання одного стоп біту цей параметр повинен приймати значення **-cstopb**. Параметр **-parenb** вказує, що не потрібно використовувати біт парності.

Отже підсумуємо параметри UART з'єднання: швидкість 230400 біт за секунду, 8 біт даних, два стоп біти, без бітів парності.

Команда **cat** використовується для передачі файлу по інтерфейсу UART. Байти файлу передаються через інтерфейс UART один за одним зі швидкістю, що налаштована командою **stty**. Формат wav файлу для передачі наступний: 8 біт, моно, бітрейт 22050 Гц.

Для передачі аудіо файлу в форматі wav через інтерфейс UART в операційній системі Windows можна скористатися програмою TeraTerm. Параметри UART з'єднання такі ж самі, як і для ОС Linux.

3.3.5 Завдання на лабораторну роботу

Вихідний код завдань можна переглянути за посиланням [\[3.5\]](#).

Для реалізації подільників частоти, роботу яких необхідно перевірити на налагоджувальній платі Intel FPGA, у якості вхідного сигналу синхронізації необхідно використовувати сигнал синхронізації, що надходить в FPGA від зовнішнього кварцевого генератора. Вихідний сигнал синхронізації необхідно вивести на один з контактів роз'єму GPIO. Сигнал асинхронного скидання необхідно підключити до однієї з кнопок KEY.

1. На мові Verilog створіть тестбенч для подільника частоти з лістингу 3.5. Просимулюйте подільник частоти і тестбенч для двох різних значень параметру **N**. Переконайтеся, що для різних значень **N** частота вхідного сигналу ділиться у відповідну кількість разів.
2. В Quartus Prime створіть подільник частоти, що описується вихідним кодом з лістингу 3.1. Запустіть подільник на обраній налагоджувальній платі Intel FPGA. Створіть для проекту SDC йафл з обмеженнями на синтез. Виконайте аналіз результатів компіляції проекту в TimeQuest. В SignalTap переконайтеся, що частота імпульсного сигналу на виході подільника в **DIV_BY** разів менша, ніж частота імпульсного сигналу на вході.
3. В Quartus Prime створіть програмований подільник частоти на довільне ціле число, що описується вихідним кодом з лістингу 3.2. Двійковий код, що задає коефіцієнт ділення частоти, необхідно зчитувати з кнопок SW. Запустіть подільник на обраній налагоджувальній платі Intel FPGA. З використанням логічного аналізатора, або осцилографа переконайтеся, що частота імпульсного сигналу на виході подільника менша, ніж частота імпульсного сигналу на вході у задану кількість разів, яка визначається цифровим кодом на перемикачах SW.

4. В Quartus Prime створіть програмований подільник частоти на довільне дробне число. Значення цифрового коду для входу NCO необхідно зчитувати з кнопок SW. Запустіть подільник на обраній налагоджувальній платі Intel FPGA. З використанням логічного аналізатора, або осцилографа переконайтеся, що частота імпульсного сигналу на виході подільника менша, ніж частота імпульсного сигналу на вході у дробне число разів, в залежності від розрядності NCO і значення входу SW. Розрядність NCO можна обрати рівну 32 біти.
5. Для обраної налагоджувальної плати FPGA створіть блок PLL. Частота вхідного сигналу синхронізації повинна дорівнювати частоті тактового сигналу від зовнішнього кварцевого генератора (для більшості налагоджувальних плат ця частота дорівнює 50 МГц). Частота вихідного сигналу синхронізації повинна дорівнювати 250 МГц.
6. Реалізуйте цифровий FM передавач, вихідний код якого наведено в лістингу 3.5. Вихідний код UART приймача наведено за посиланням [\[3.3\]](#). Використайте в проєкті блок PLL, створений у попередньому завданні. Старший розряд NCO підключіть до одного з виводів GPIO і до цього ж виводу підключіть провідник антени. Створіть і підключіть до проєкту SDC файл з обмеженнями на синтез. Виконайте аналіз результатів компіляції проєкту в TimeQuest. Запустіть роботу FM передавача на обраній налагоджувальній платі. Передача аудіо файлу з комп'ютера в налагоджувальну плату описана в розділі 3.3.4. Приймання радіосигналу можна виконувати на звичайний FM приймач, налаштований на частоту передачі (в лістингу 3.5 частота передачі рівна 90 МГц).

3.4 Контрольні запитання

1. Намалюйте схему подільника частоти на ціле число на базі двійкового лічильника вгору. Поясніть принцип роботи схеми.
2. Намалюйте схему подільника частоти на ціле число на базі двійкового лічильника вниз. Поясніть принцип роботи схеми.
3. Намалюйте схему подільника частоти на ціле число на базі лічильника у позиційному коді. Поясніть принцип роботи схеми.

4. Намалюйте схему подільника частоти на дробне число. Поясніть принцип роботи схеми.
5. Поясніть, чому вихід компаратора не можна використовуватися у якості джерела сигналу синхронізації для синхронних по фронту схем.
6. Поясніть призначення параметрів у мові Verilog. Поясніть яким чином визначають параметри, як і з якою метою використовують, як перевизначити параметр при створенні екземпляру модуля.
7. Перелічіть та опишіть оператори редукції у мові Verilog.
8. Намалюйте умовне графічне позначення блоку PLL та опишіть логіку роботи PLL (яким чином пов'язані входи і виходи схеми).
9. Поясніть від чого залежить максимально можлива частота сигналу синхронізації цифрових схем.
10. Наведіть формулу для розрахунку максимально можливої частоти сигналу синхронізації цифрових схем.
11. Поясніть, що таке clock skew.
12. Поясніть явище гонок в комбінаційних схемах. Проілюструйте на прикладі.
13. Поясніть відмінності між знаковим і беззнаковим форматами представлення чисел в цифрових системах.
14. Намалюйте структурну схему і поясніть принцип роботи цифрового FM передавача.
15. Продемонструйте створення і налаштування блоку PLL в Quartus Prime.
16. Поясніть необхідність визначення обмежень на синтез (Synthesis Constraints).
17. Поясніть команди, що знаходяться всередині типового sdc файлу для налагоджувальної плати Intel FPGA.
18. Продемонструйте вміння користуватися TimeQuest для визначення комбінаційних схем з критично великою затримкою, що обмежують функціонування пристрою на заданій частоті.
19. Поясніть, як розрахувати значення Clock Setup Slack.

3.5 Перелік посилань

- [3.1] “Unlocking the Phase Lock Loop”, 2013. [Online]. Available: <http://complextoreal.com/wp-content/uploads/2013/01/pll.pdf>
- [3.2] “TimeQuest for Dummies”, 2015. [Online]. Available: http://caxapa.ru/thumbs/442268/TimeQuest_for_dummies.pdf
- [3.3] “Source code for simple UART receiver”, 2018. [Online]. Available: https://github.com/KorotkiyEugene/digital_lab/blob/master/Lab3/hw/fm_transmitter/simple_uart_receiver.v
- [3.4] “Documentation and design files for Intel FPGA boards”, 2018. [Online]. Available: https://github.com/KorotkiyEugene/intel_fpga_boards
- [3.5] “Вихідні коди прикладів до лабораторних робіт”, 2018 [Online]. Режим доступу: https://github.com/KorotkiyEugene/digital_lab