Лабораторна робота №2

Синхронні лічильники, постійна пам’ять,

цифрові генератори

2.1 Практичне застосування досліджуваних схем **1**

2.2 Теоретична частина **1**

2.2.1 Умовне позначення і логіка роботи синхронного лічильника 1

2.2.2 Синхронний Т-тригер 3

2.2.3 Схема і принцип роботи синхронного лічильника 5

2.2.4 Опис синхронного лічильника на мові Verilog 7

2.2.5 Опис синхронного лічильника з входом завантаження на мові Verilog 9

2.2.6 Арифметичні оператори у мові Verilog (+, -, *, /, %) 10

2.2.7 Цифро-аналоговий перетворювач (ЦАП) на резисторах 11

2.2.8 Генератор пилкоподібної напруги на базі лічильника і ЦАП 14

2.2.9 Опис постійної ROM пам’яті на мові Verilog. Масиви в мові Verilog 15

2.2.10 Шаблони Verilog коду для опису цифрових схем в Quartus Prime 20

2.2.11 Цифровий генератор синусоїдального сигналу 21

2.2.12 Тестбенч на мові Verilog (initial процеси, цикли, затримки) 28

2.2.13 Цифровий генератор синусоїдального сигналу з керованою частотою 34

2.3 Практична частина **39**

2.3.1 Симуляція проектів в ModelSim 39

2.3.2 Налаштування синтезу пам’яті в Quartus Prime 41

2.3.3 Цифро-аналоговий перетворювач у налагоджувальних платах з VGA 41

2.3.4 Завдання на лабораторну роботу 44

2.4 Контрольні запитання **45**

2.5 Перелік посилань **48**

2.1 Практичне застосування досліджуваних схем

Синхронні лічильники імпульсів широко використовують для ділення частоти на довільне значення, зокрема в таймерах. Будь який таймер побудований на базі лічильника. В мікроконтролерах таймери використовують для формування часових інтервалів. Наприклад, у перериванні таймеру з частотою 1 КГц може виконуватись перемикання процесів операційної системи. Або по перериванню від таймеру з певною періодичністю процесор може виходити з режиму сну (зі зниженим енергоспоживанням), зчитувати дані від сенсорів і передавати їх в мережу, а потім знову переходити в режим сну до наступного переривання від таймеру.

Загалом ділення частоти одна з найактуальніших задач, оскільки в цифрових мікросхемах часто необхідно виконувати різні операції з різною періодичністю. Цього можна досягти діленням опорної вхідної частоти на різні значення і виконуючи операції з прив’язкою до одержаних частот.

Лічильники імпульсів використовують для широтно-імпульсної модуляції, яку ми розглянемо в одній з наступних лабораторних робіт. Також синхронний лічильник є основним компонентом генератора імпульсів з цифровим керуванням (Numerically Controlled Oscillator, NCO), за допомогою якого можна генерувати аналогові сигнали, або реалізувати частотну та амплітудну модуляції, зокрема для радіо передачі цифрових даних. Створення аналогових сигналів з використанням NCO називається Прямим Цифровим Синтезом (Direct Digital Synthesis, DDS) і застосовується в усіх сучасних генераторах сигналів.

2.2 Теоретична частина

2.2.1 Умовне позначення і логіка роботи синхронного лічильника

Умовне графічне позначення синхронного лічильника наведено на рис.2.1.

Як видно з рис.2.1. типовий синхронний лічильник має однорозрядні входи синхронізації та асинхронного зкидання (в даному випадку це входи i_clk та i_rst_n) і багаторозрядний вихід, що відображає вміст лічильника (в даному випадку це вихід o_counter).

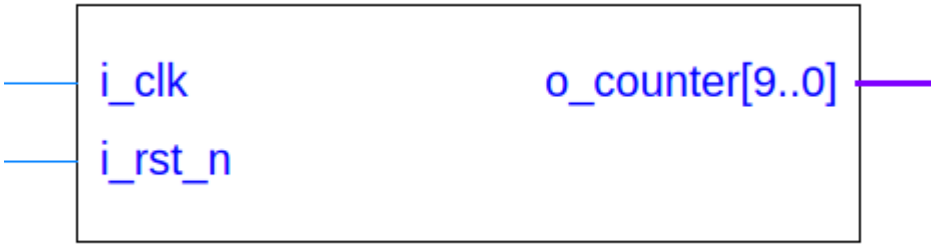

Рис.2.1 - Умовне графічне позначення синхронного лічильника з входом асинхронного зкидання

Основна функція лічильника імпульсів - рахувати імпульси, що надходять на вхід синхронізації. Для лічильників, що рахують вгору, після кожного активного фронту на вході синхронізації вміст лічильника збільшується на 1. Для лічильників, що рахують вниз, після кожного активного фронту на вході синхронізації вміст лічильника зменшується на 1.

Лічильники вміст яких представляється двійковим числом називають двійковими лічильниками. Двійкові лічильники найбільш розповсюджені в цифровій схемотехніці. Окрім двійкових лічильників є ще лічильники у позиційному коді (one-hot counters) і лічильники у коді Грея (Gray code counters). Далі в цій лабораторній роботі будемо розглядати лише двійкові лічильники. Лічильник у позиційному коді розглянемо в наступній лабораторній роботі.

При подачі активного логічного рівня на вхід асинхронного зкидання, значення лічильника зкидається в 0. Активним рівнем асинхронного зкидання у багатьох лічильниках є логічний нуль. Однак будуючи схему лічильника у схемному редакторі, або описуючи таку схему на мові Verilog ви можете за власним бажанням визначити активний логічний рівень входу асинхронного зкидання.

У деяких лічильників також може бути присутній вхід синхронного зкидання. Якщо на вхід синхронного зкидання подано активний логічний рівень, після надходження найближчого активного фронту на вхід синхронізації вміст такого лічильника зкинеться в 0.

Лічильник розрядністю N біт може рахувати від 0 до 2^N -1. Якщо N-розрядний лічильник, що рахує вгору, містить число 2^N -1, наступний імпульс на вході

синхронізації призведе до переповнення лічильника (counter overflow) і появи в ньому значення 0. Іншими словами, у такого лічильника буде 2^N стійких станів (від 0 до 2^N -1). Кількість станів, які може приймати лічильник називається його модулем рахунку M. Для деяких лічильників можна задати їх модуль рахунку за допомогою окремого входу. Така можливість часто використовується в керованих подільниках частоти, які розглянемо в наступній лабораторній роботі.

За допомогою лічильника з модулем рахунку M можна створити подільник частоти на M.

У деяких лічильниках присутній вхід завантаження вмісту лічильника, за допомогою якого можна завантажити в лічильник певне значення, з якого буде продовжуватись рахунок.

2.2.2 Синхронний Т-тригер

Перед вивченням схеми синхронного двійкового лічильника необхідно розібратися з синхронним по фронту Т-тригером, умовне графічне позначення якого наведено на рис.2.2.

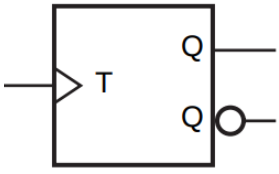

Рис.2.2 - Умовне графічне позначення Т-тригера синхронного по передньому фронту

Найпростіший Т-тригер має вхід синхронізації Т, на який подається сигнал тактової частоти. На виході Q присутнє значення тригеру (0 або 1).

По кожному активному фронту на вході синхронізації вміст Т-тригеру інвертується - рис.2.3.

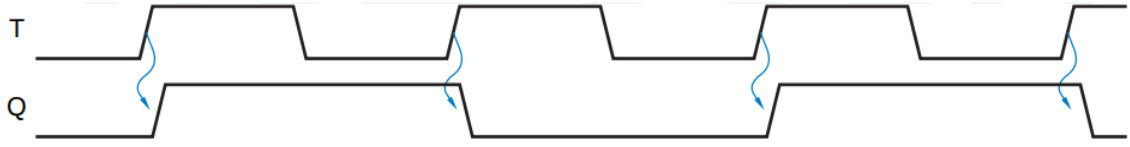

Рис.2.3 - Часова діаграма зміни сигналів в Т-тригері синхронному по передньому фронту

Як видно з рис.2.3 найпростіший Т-тригер працює, як подільник частоти на 2, оскільки період імпульсного сигналу на виході Q в два рази більший ніж період сигналу на вході Т (відповідно, частота на виході в 2 рази нижча).

Найпростіший Т-тригер можна побудувати на базі синхронного по фронту D-тригера - рис.2.4.

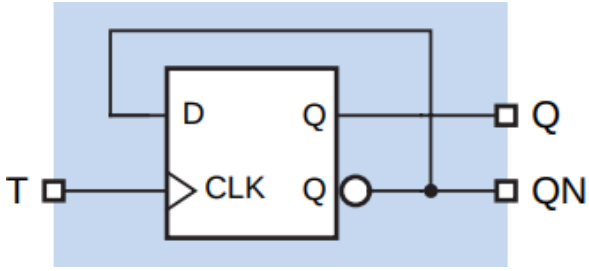

Рис.2.4 - Схема Т-тригера синхронного по передньому фронту

Як видно з рис.2.4 значення інверсного виходу тригера подається на вхід даних D тригера. Якщо тригер зберігає значення Q=1, на його інверсному виході QN буде присутній 0 і цей 0 буде подано на вхід даних. Отже по наступному активному фронту цей 0 буде записано в тригер і на виході Q з'явиться 0.

Також існує Т-тригер з входом дозволу роботи EN. Умовне графічне позначення такого тригеру наведено на рис.2.5, а часові діаграми роботи - на рис.2.6.

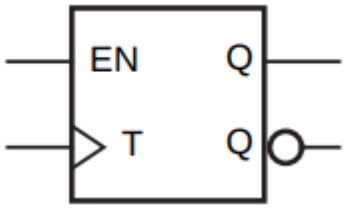

Рис.2.5 - Умовне графічне позначення Т-тригера синхронного по передньому фронту з входом дозволу EN (Enable)

В такому Т-тригері значення на виході Q інвертується по активному фронту сигналу синхронізації на вході Т лише за умови, що на вході EN присутній активний логічний рівень. Якщо на вході EN не активний логічний рівень - після надходження активного фронту стан тригеру не змінюється. Це добре видно з рис.2.6.

Зазвичай у входа EN активний високий рівень.

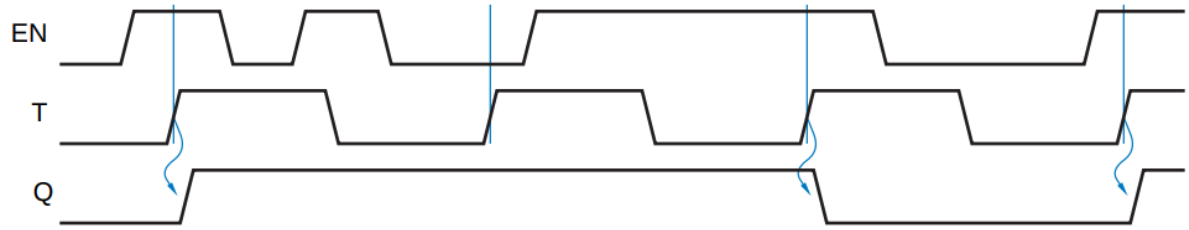

Рис.2.6 - Часова діаграма зміни сигналів в синхронному по передньому фронту Т-тригері з входом дозволу EN (Enable)

Схема Т-тригера синхронного по передньому фронту з входом дозволу EN наведена на рис.2.7.

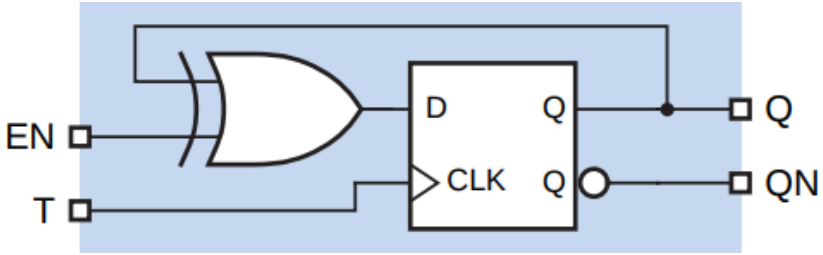

Рис.2.7 - Схема Т-тригера синхронного по передньому фронту з входом дозволу EN

2.2.3 Схема і принцип роботи синхронного лічильника

Розглянемо часову діаграму лічильника, що рахує вгору - рис.2.8.

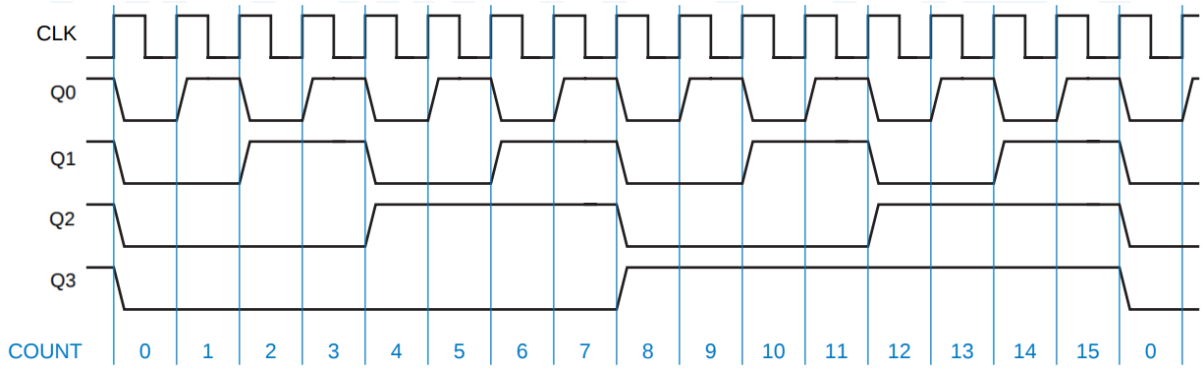

Рис.2.8 - Часова діаграма зміни сигналів в синхронному лічильнику, що рахує вгору

З діаграми на рис.2.8 видно, що кожен наступний розряд інвертується в момент надходження активного (переднього) фронту сигналу синхронізації CLK, за умови, що всі попередні розряди приймають значення одиниць.

Враховуючи наведене спостереження не складно побудувати схему синхронного лічильника, що рахує вгору, на базі синхронних по фронту Т-тригерів з входом дозволу - рис.2.9.

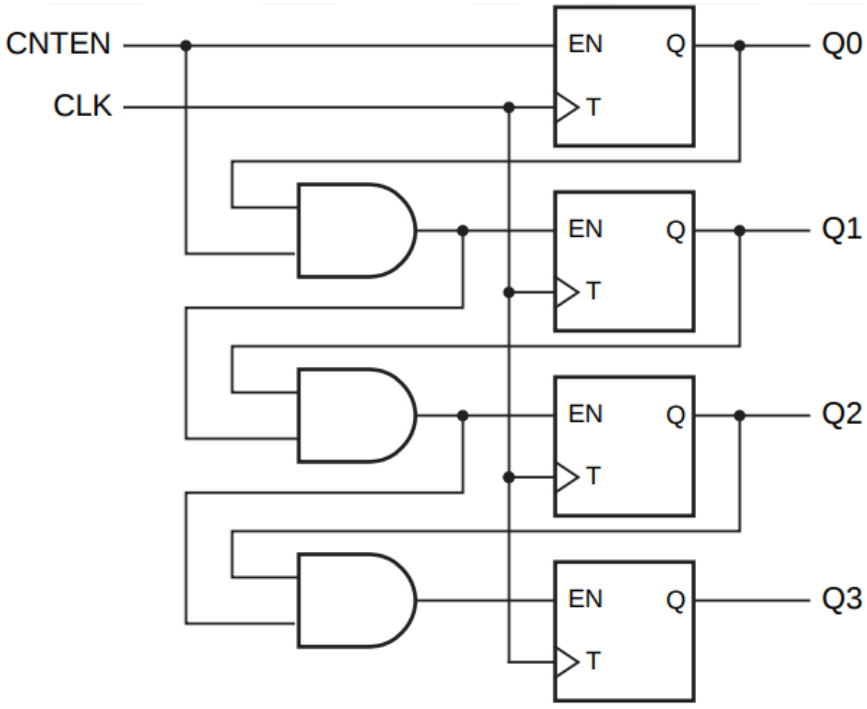

Рис.2.9 - Схема синхронного 4-х розрядного лічильника вгору з послідовним формуванням сигналу EN

Як видно з рис.2.9, на вході EN кожного Т-тригера буде присутня лог. 1 лише за умови, що на виходах Q всіх попередніх розрядів будуть одиниці. Таку схему нескладно масштабувати на більшу розрядність. При цьому, затримка формування сигналу EN для старшого розряду лічильника (в даному випадку Q3) буде дорівнювати сумі затримок 3-х лог. елементів AND.

Можна зменшити затримку формування сигналів EN використовуючи схему з рис.2.10.

Зверніть увагу на вхід CNTEN, що дозволяє рахунок лічильника. У разі, якщо CNTEN = 0, на входах EN Т-тригерів присутній 0 і стан тригерів після приходу активного фронту не змінюється. Якщо CNTEN = 1, лічильник починає лічити імпульси вхідного сигналу синхронізації.

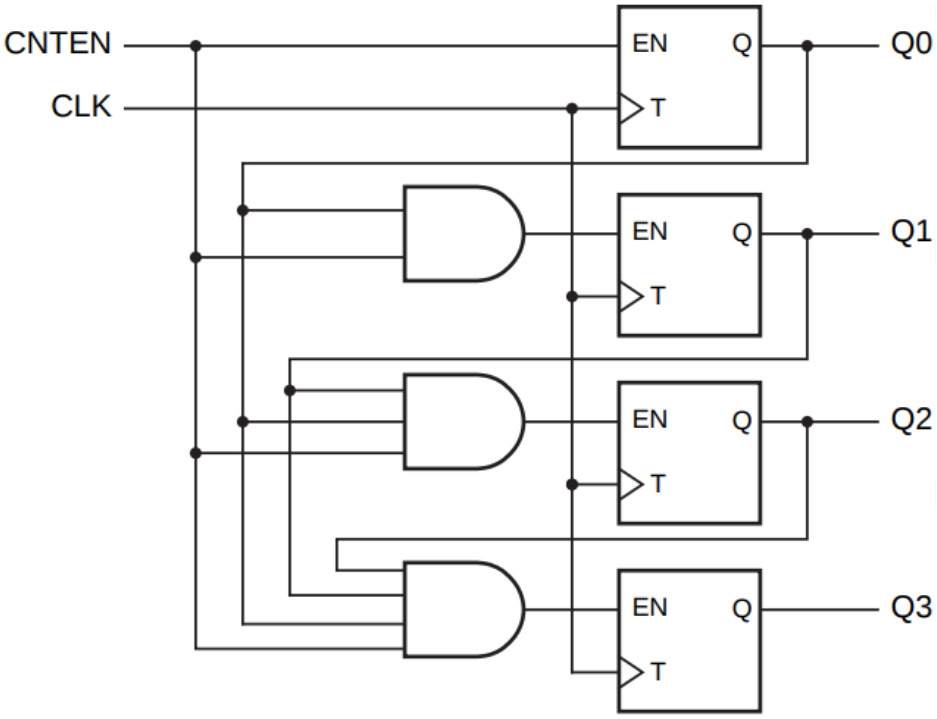

Рис.2.10 - Схема синхронного 4-х розрядного лічильника вгору з паралельним формуванням сигналу EN

Зверніть увагу, що частота імпульсного сигналу з рис.2.8 у кожному наступному розряді вдвічі менша, ніж у попередньому розряді. Саме тому звичайний лічильник можна використовувати для ділення частоти на 2^N .

Для побудови лічильника, що рахує вниз на входи лог. елементів AND необхідно подавати дані не з прямих виходів Q, а з інверсних виходів NQ (або інвертовані значення Q за допомогою інверторів). За необхідності можна створити лічильник з керованим напрямком рахунку (вгору/вниз), якщо обирати між прямим і інвертованим значенням Q за допомогою мультиплексора.

2.2.4 Опис синхронного лічильника на мові Verilog

Шаблон опису на мові Verilog для синхронного лічильника, що рахує вгору наведено в лістингу 2.1.

Як видно з лістингу 2.1. по передньому фронту `i_clk`, якщо сигнал `i_rst_n` не активний, до значення `o_counter` додається 1 і результат суми записується у багаторозрядну змінну `o_counter` типу **`reg`**. Оскільки така логіка роботи відповідає

лічильнику, САПР синтезує зазначений шаблон коду в лічильник. Розрядність лічильника визначається розрядністю змінної `o_counter`.

Аналогічним чином можна описати лічильник, що рахує вниз, однак замість оператора “+” необхідно використовувати оператор “-”.

Оператор “+” в даному випадку синтезується в суматор.

Лістинг 2.1 - Шаблон опису синхронного лічильника вгору на мові Verilog

```
module counter(i_clk, i_rst_n, o_counter);

input               i_clk;
input               i_rst_n;
output reg [9:0]    o_counter;

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

endmodule
```

Результат синтезу такого вихідного коду в Quartus Prime наведений на рис.2.11. Зазначений результат синтезу можна переглянути самостійно в RTL Viewer. Інформація по користуванню RTL Viewer наведена в лабораторній роботі Lab0.

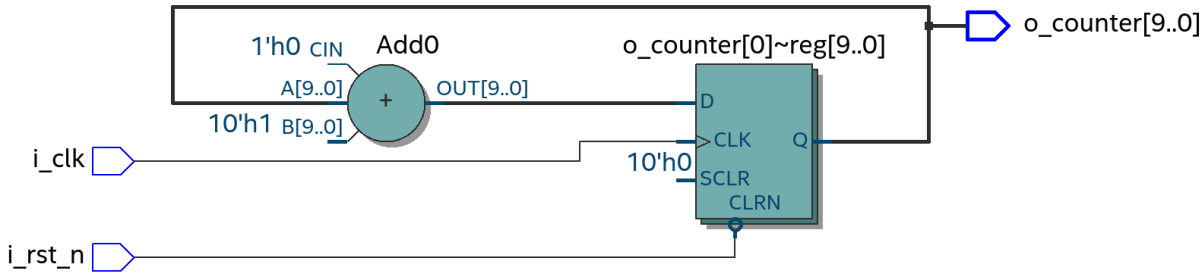

Рис.2.11 - Результат синтезу синхронного лічильника з лістингу 2.1

Зверніть увагу, що вираз у правій частині оператора неблокуючого присвоювання синтезується в комбінаційну схему, вихід якої підключається на вхід даних регістру `o_counter`, що описується в `always` блоці.

Схема з рис.2.11 є ще одним варіантом реалізації синхронного двійкового лічильника, що використовує паралельний регістр та суматор, який додає до вмісту регістру одиницю і результат суми по активному фронту сигналу синхронізації записується в регістр.

2.2.5 Опис синхронного лічильника з входом завантаження на мові Verilog

Шаблон опису на мові Verilog для лічильника вгору із завантаженням наведено в лістингу 2.2.

Лістинг 2.2 - Шаблон опису лічильника вгору із завантаженням на мові Verilog

```
module counter(i_clk, i_rst_n, i_load, i_data, o_counter);

input               i_clk;
input               i_rst_n;
input               i_load;
input      [9:0]    i_data;
output reg [9:0]    o_counter;

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

endmodule
```

Результат синтезу лістингу 2.2 можна переглянути на рис.2.12. Як видно, схема дуже схожа на лічильник з рис.2.11, однак на рис.2.12 присутній додатковий мультиплексор, за допомогою якого можна підключити вхід даних регістру або до входу `i_data`, або до виходу суматора.

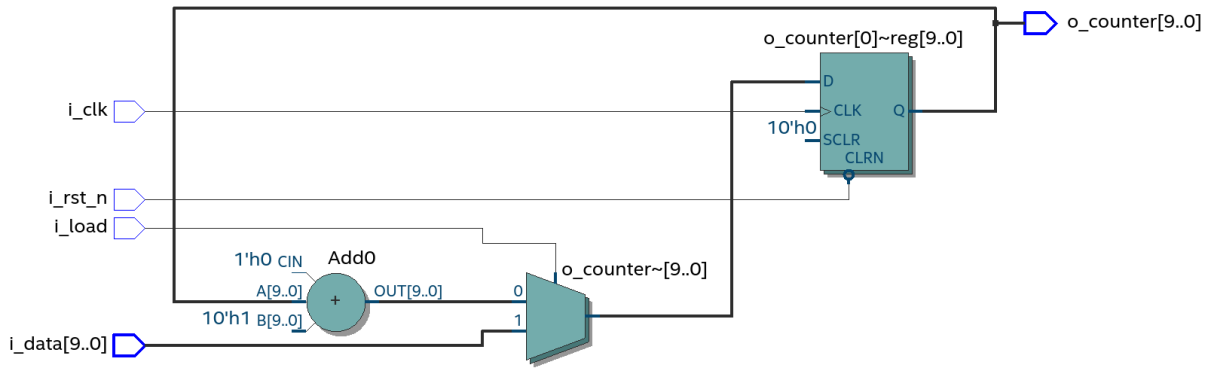

Рис.2.12 - Результат синтезу лічильника з лістингу 2.2 в RTL Viewer

На рис.2.12 мультиплексор має однорозрядний вхід адреси і два багаторозрядних входи з номерами 0 та 1. Вхід адреси мультиплексора підключений до входу `i_load`.

Якщо однорозрядний вхід адреси приймає значення 0, на вихід мультиплексора передаються дані з входу з номером 0. Якщо однорозрядний вхід адреси приймає значення 1, на вихід мультиплексора передаються дані з входу з номером 1. Більш детально мультиплексор буде розглянуто в одній з наступних лабораторних робіт.

Таким чином, коли вхід `i_load` приймає значення лог. 1, в регістр, що зберігає стан лічильника, після активного переднього фронту на вході синхронізації `i_clk` записуються дані з входу `i_data`. Якщо ж на вході `i_load` присутній лог. 0, по кожному активному передньому фронту на вході синхронізації вміст лічильника збільшується на 1.

2.2.6 Арифметичні оператори у мові Verilog (+, -, *, /, %)

В лістингах 2.1 і 2.2 для реалізації додавання у лічильнику було застосовано арифметичний оператор + мови Verilog.

Окрім оператора додавання мова Verilog містить арифметичні оператори:

Віднімання	-
Множення	*
Ділення	/
Ділення по модулю	%

Якщо зазначені вище арифметичні оператори використовують для опису цифрових схем, оператори синтезуються в комбінаційні схеми, що виконують необхідну функцію (наприклад, оператор + синтезується в суматор).

Оператори *****, **/** і **%** синтезуються в складні комбінаційні схеми, що мають значну затримку і потребують значних апаратурних витрат (великої кількості логічних елементів) для реалізації. Тому ці оператори необхідно використовувати розумно.

Пізніше в лабораторних роботах ми розглянемо схеми і вихідний код для операцій множення/ділення за кілька тактів з низькими апаратурними витратами.

2.2.7 Цифро-аналоговий перетворювач (ЦАП) на резисторах

Даний матеріал необхідний для розуміння наступних розділів лабораторної роботи.

Цифро-аналоговий перетворювач, ЦАП (Digital-to-Analog Converter, DAC) — це пристрій для перетворення цифрового коду в напругу. На цифровий вхід ЦАП подається число розрядністю N, а на виході ЦАП з'являється напруга, що відповідає цифровому коду на вході. Також на ЦАП подається опорна напруга Vref. Напруга на виході ЦАП прямо пропорційна цифровому коду на його вході.

Нулю на цифровому вході ЦАП відповідає нульова напруга на виході. Максимальному цифровому коду на вході N-розрядного ЦАП (2^N-1), відповідає максимально можливе значення напруги на виході, яке називається опорною напругою (Reference Voltage, Vref) і часто дорівнює напрузі живлення.

Формула залежності напруги Vo на виході ЦАП від цифрового N-розрядного коду D на вході та опорної напруги Vref:

$$Vo = \frac{Vref}{2^N - 1} \cdot D \qquad (2.0)$$

Умовне графічне позначення ЦАП наведено на рис.2.13. Залежність напруги на виході 4-розрядного ЦАП від цифрового коду на його вході наведена на рис.2.14.

Для сусідніх цифрових кодів на вході ЦАП напруга на його виході буде відрізнятися на величину $h = \frac{Vref}{2^N - 1}$. Чим більша розрядність ЦАП, тим менше буде h.

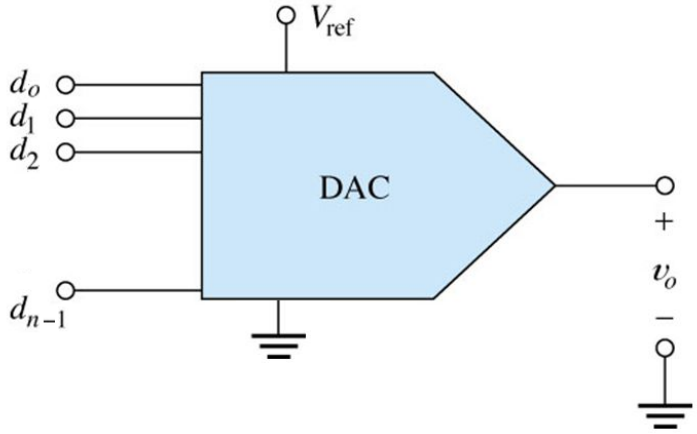

Рис.2.13 - Умовне графічне позначення ЦАП

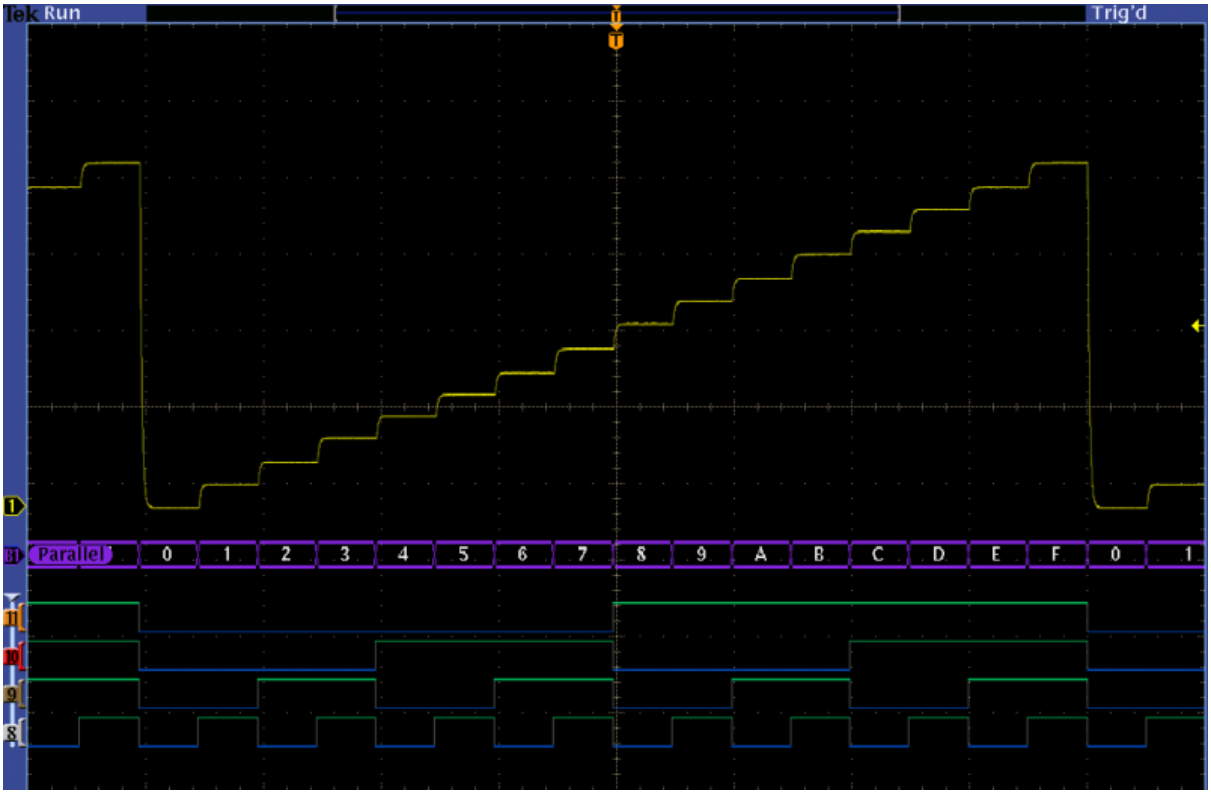

Рис.2.14 - Залежність напруги на виході 4-розрядного ЦАП
від цифрового коду на його вході

Більшість широко використовуваних ЦАП мають розрядність в діапазоні 8 - 16 біт. Однак іноді використовують ЦАП, як дуже низької розрядності (4 біт), так і високої розрядності (24 біт).

Професійні ЦАП випускають у вигляді дискретних мікросхем, однак для багатьох задач підходить ЦАП складений із резисторів.

Існує дві схеми ЦАП на резисторах: R2R ЦАП і ЦАП з ваговими резисторами.

R2R ЦАП дуже розповсюджений і з його конструкцією можна ознайомитись за посиланням [2.1].

Однак в лабораторній роботі ми будемо використовувати ЦАП з ваговими резисторами, оскільки таку конструкцію має 4-розрядний ЦАП інтерфейсу VGA на налагоджувальних платах Intel FPGA, який ми використовуватимемо в генераторах сигналів. Тому розглянемо детальніше ЦАП з ваговими резисторами.

Схема 4-розрядного ЦАП з ваговими резисторами наведена на рис.2.15.

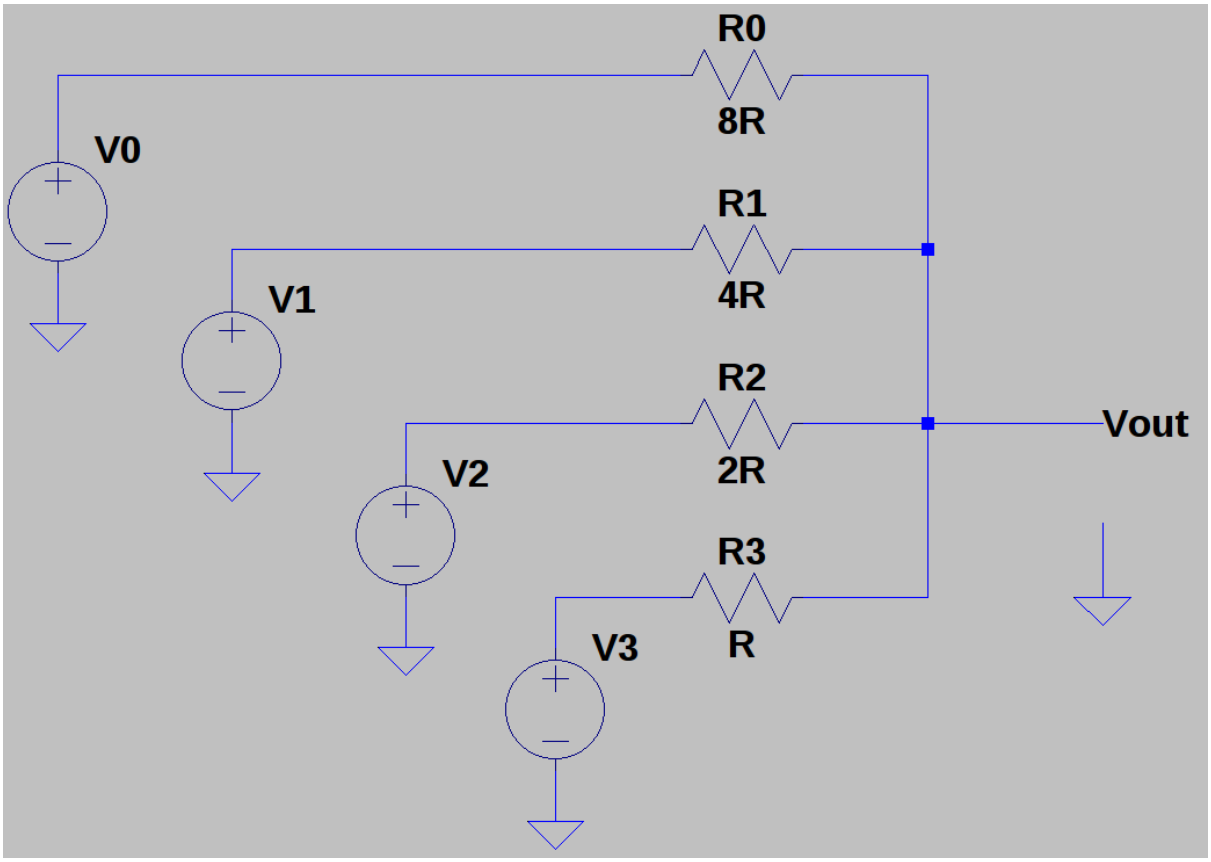

Рис.2.15 - Схема 4-х розрядного ЦАП з ваговими резисторами

В ЦАП на рис.2.15 вхідний цифровий код задається джерелами напруги, які моделюють логічні рівні. Кожне таке джерело на вході може видавати або лог. 0 (0 вольт), або лог.1. (Vref=Vdd вольт, де Vdd - напруга живлення). Напругу кожного джерела з номером i можна представити формулою Vi = $Vref \cdot Di = Vdd \cdot Di$, де Di представляє і-тий розряд цифрового коду на вході ЦАП і може приймати значення 0, або 1.

Запишемо перший закон Кірхгофа для струмів через резистори:

$$I_0 + I_1 + I_2 + I_3 = 0 \quad (2.1)$$

З другого закону Кірхгофа напругу на і-му резисторі V_{Ri} можна представити формулою (де V_i - напруга і-го вхідного джерела, а V_{out} - напруга на виході ЦАП):

$$V_{Ri} = V_i - V_{out} \quad (2.2)$$

Підставивши формулу (2.2) в (2.1) для кожного *i* від 0 до 3 та застосувавши закон Ома, отримаємо:

$$\frac{V_0 - V_{out}}{R_0} + \frac{V_1 - V_{out}}{R_1} + \frac{V_2 - V_{out}}{R_2} + \frac{V_3 - V_{out}}{R_3} = 0 \quad (2.3)$$

Перегрупувавши доданки отримаємо:

$$V_{out} = \frac{\frac{V0}{R0} + \frac{V1}{R1} + \frac{V2}{R2} + \frac{V3}{R3}}{\frac{1}{R0} + \frac{1}{R1} + \frac{1}{R2} + \frac{1}{R3}} \quad (2.4)$$

Враховуючи, що R3 = R, R2 = 2*R, R1 = 4*R, R0 = 8*R, підставимо ці значення в (2.4) і отримаємо (D - цифровий код на вході ЦАП з розрядністю N біт, Di - розряди цього цифрового коду):

$$V_{out} = \frac{\frac{V0}{8R} + \frac{V1}{4R} + \frac{V2}{2R} + \frac{V3}{R}}{\frac{1}{8R} + \frac{1}{4R} + \frac{1}{2R} + \frac{1}{R}} = \frac{V_0 + 2 \cdot V_1 + 4 \cdot V_2 + 8 \cdot V_3}{15} =$$

$$= \frac{V_{dd}}{15} \cdot (D_0 + 2 \cdot D_1 + 4 \cdot D_2 + 8 \cdot D_3) = \frac{V_{dd}}{15} \cdot D$$

2.2.8 Генератор пилкоподібної напруги на базі лічильника і ЦАП

Найпростіший генератор пилкоподібної напруги можна створити, якщо підключити вихід двійкового лічильника до цифрового входу ЦАП. Зміна сигналів на виході лічильника і напруга на виході ЦАП в такій схемі зображена на рис.2.14.

Якщо розрядність ЦАП буде досить велика (10-16 біт), зміна напруги, що відповідає сусіднім цифровим кодам буде дуже мала і практично непомітна на око.

2.2.9 Опис постійної ROM пам’яті на мові Verilog. Масиви в мові Verilog

Для створення генератора синусоїдального сигналу знадобиться постійна пам’ять (Read Only Memory, ROM). Розглянемо, як описати таку пам’ять на мові Verilog.

Умовне графічне позначення постійної пам'яті ROM наведено на рис.2.16.

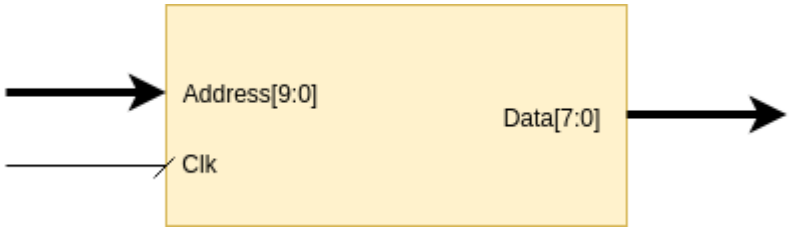

Рис.2.16 - Умовне графічне позначення постійної пам’яті (ROM)

Постійна пам’ять призначена для збереження даних. Пам’ять складається з комірок. Кожна комірка зберігає слово даних певної розрядності.

Постійна пам’ять має N-розрядний вхід адреси (на рис.2.16 вхід адреси називається Address, N=10) та M-розрядний вихід даних (на рис.2.16 вихід даних називається Data, M=8). Деякі екземпляри постійної пам’яті можуть мати вхід синхронізації, який часто називають Clk.

Постійна пам’ять з N-розрядним входом адреси містить 2^N комірок, з номерами від 0 до 2^N -1. На виході даних постійної пам'яті з'являється вміст комірки з номером, який поданий на вхід адреси. Якщо постійна пам’ять не має входу синхронізації, поява даних на виході пам’яті відбувається з певною затримкою після подачі нової адреси на вхід. Ця затримка називається затримкою зчитування з пам'яті - рис.2.17. Якщо постійна пам’ять має вхід синхронізації, нові дані з'являються на виході з невеликою затримкою після подачі на вхід адреси нового номеру комірки пам'яті і надходження активного фронту на вхід синхронізації - рис.2.18.

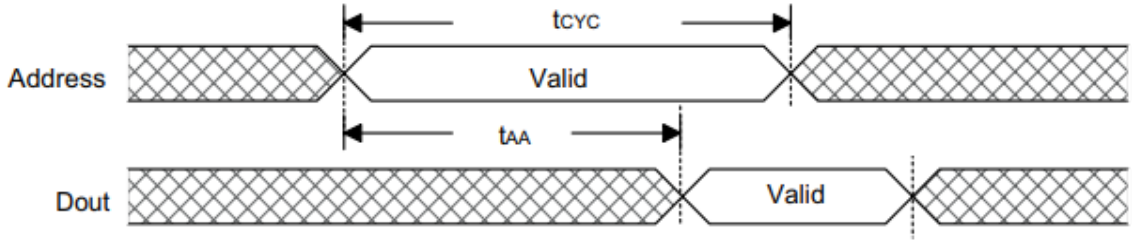

Рис.2.17 - Часова діаграма зчитування з пост. пам’яті ROM (tAA - затримка зчитування)

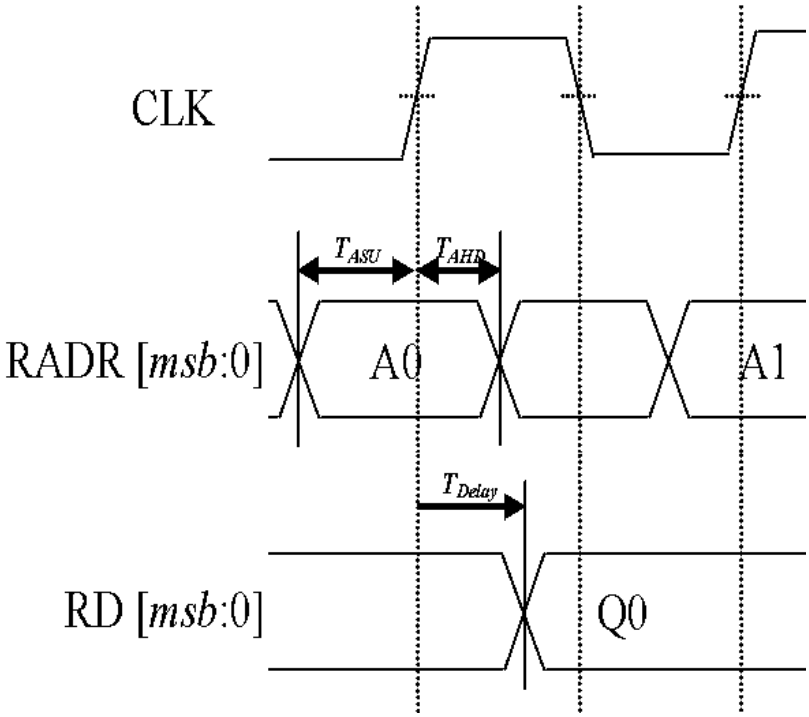

Рис.2.18 - Часова діаграма зчитування з постійної ROM пам’яті з входом синхронізації CLK. Нові дані (вміст комірки з адресою RADR) з'являються на виході RD з затримкою Tdelay після активного фронту CLK

Деякі екземпляри постійної пам’яті мають вхід дозволу CS (Chip Select). У випадку, коли на CS присутній активний рівень, ROM пам’ять працює, як зазначалося вище. Якщо ж на CS присутній неактивний лог. рівень, вихід даних пам’яті відключається від внутрішнього драйверу, який створює на виході логічні рівні і переходить в так званий третій стан.

Шаблон опису постійної пам’яті на мові Verilog наведений в лістингу 2.3. Зустрічаючи зазначений код, САПР Quartus Prime заміняє його на об’єкт постійної ROM пам'яті. Такий вихідний код призводить до створення постійної ROM пам’яті з входом синхронізації. Це найбільш поширений шаблон опису ROM пам’яті для FPGA.

Для опису пам’яті у мові Verilog використовують **масиви**. Визначити масив на мові Verilog можна за допомогою наступного коду:

```
reg [7:0] rom[1023:0];
```

Спершу зазначають тип і розрядність комірок масиву (в даному прикладі, комірки масиву мають тип `reg` і розрядність 8 біт). Далі йде ім’я масиву (в даному прикладі масив має ім’я `rom`). Після імені масиву у квадратних скобках визначають

кількість комірок масиву. Правий індекс задає номер початкової комірки (в даному прикладі це 0). Лівий індекс задає номер кінцевої комірки масиву (в даному прикладі це 1023). Отже в даному прикладі масив має 1024 комірки з номерами від 0 до 1023.

Комірки масивів мови Verilog можуть мати тип `reg`, `wire`, `integer`.

Лістинг 2.3 - Шаблон опису постійної пам'яті (ROM) на мові Verilog

```
module rom_mem(i_clk, i_addr, o_data);

input                i_clk;
input        [9:0]   i_addr;
output reg   [7:0]   o_data;

reg [7:0]   rom[1023:0];

initial $readmemh("rom_init.hex", rom);

always @(posedge i_clk)
    o_data <= rom[i_addr];

endmodule
```

В лістингу 2.3 по кожному активному фронту сигналу синхронізації `i_clk`, вміст комірки з номером `i_addr` масиву `rom` записується в змінну `o_data`.

Результат синтезу вихідного коду з лістингу 2.3 можна переглянути в RTL Viewer (рис.2.19.).

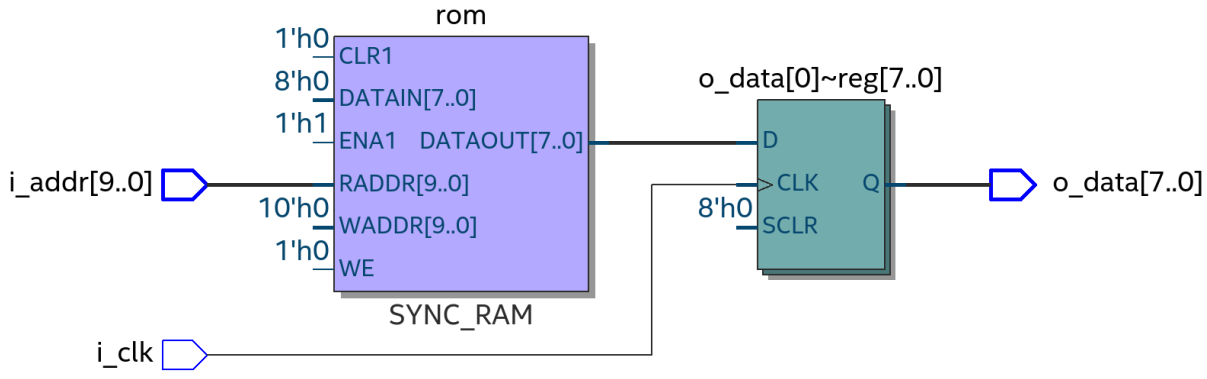

Рис.2.19 - Результат синтезу вихідного коду з лістингу 2.3 в RTL Viewer

Як видно з рис.2.19, вихідному коду з лістингу 2.3 відповідає статична пам'ять без входу синхронізації, до виходу якої підключений вхід даних паралельного регістру, який тактується від входу синхронізації `i_clk`.

Для ініціалізації масивів у мові Verilog використовують системну функцію `$readmemh`. Системна функція `$readmemh` ініціалізує масив даними з текстового файлу. Системні функції Verilog підтримують всі сучасні синтезатори і симулятори.

Якщо викликати системну функцію `$readmemh` на початку **`initial`** блоку, масив буде ініціалізовано в нульовий момент часу, одразу на початку симуляції.

У випадку синтезу для FPGA, виклик функції `$readmemh` на початку **`initial`** блоку призводить до додавання даних про ініціалізацію пам’яті до конфігураційного файлу FPGA. В такому випадку після конфігурації в FPGA створюється об’єкт постійної пам’яті ROM, комірки якої ініціалізовані значеннями з файлу, що був переданий в якості аргументу в `$readmemh`.

Структура текстового файлу з якого ініціалізується вміст масиву дуже проста. В кожній строчці текстового файлу знаходиться вміст комірки Verilog масиву (або пам’яті, що синтезується з такого масиву). Початкова строчка відповідає початковій комірці масиву (пам'яті). Кінцева строчка відповідає кінцевій комірці масиву (пам'яті). Якщо масив ініціалізується функцією `$readmemh`, дані в текстовому файлі повинні бути представлені в шістандцятковому форматі (00, 11, 35, 7C, 8A, BC, AB, FF і т.д.). Якщо масив ініціалізується функцією `$readmemb`, дані в текстовому файлі повинні бути представлені в двійковому форматі (00000000, 00010001, 00110101, 01111100 і т.д.). Якщо необхідно ініціалізувати лише певні комірки масиву використовують трохи складнішу структуру файлу ініціалізації з використанням адрес комірок для яких призначені дані в кожній строчці файлу. Адресу комірки пишуть перед значенням комірки в форматі @address. Наприклад, **@7 FA** означає, що комірку з адресою **7** необхідно ініціалізувати значенням **FA**. Комірки для яких не визначені значення в ініціалізацйному файлі залишаються неініціалізованими.

Приклад вмісту файлу `rom_init.hex` для ініціалізації масиву `rom` з використанням системної функції `$readmemh`:

```
00
7C
8A
// … Вміст решти комірок
AB
FF
```

Пам’ять всередині FPGA може бути реалізована як за допомогою класичних компонентів (таблички істинності, синхронні тригери), так і за допомогою так званої блочної пам’яті, що входить до складу сучасних FPGA.

Реалізація пам'яті з застосуванням табличок істинності і синхронних тригерів приводить не раціонального використання ресурсів мікросхеми.

З іншого боку, навіть молодші і недорогих сімейства FPGA (Cyclone V, Max10) містять мегабіти блочної пам’яті. Більш серйозні FPGA можуть містити десятки мегабіт такої пам'яті.

Кількість використаної після синтезу блочної пам'яті можна переглянути у вікні Compilation Report, розділ “Total block memory bits”.

Розміщення блоків пам’яті на кристалі FPGA можна переглянути в Chip Planner - рис.2.20. Як викликати Chip Planner розказано в одній з минулих лабораторних робіт.

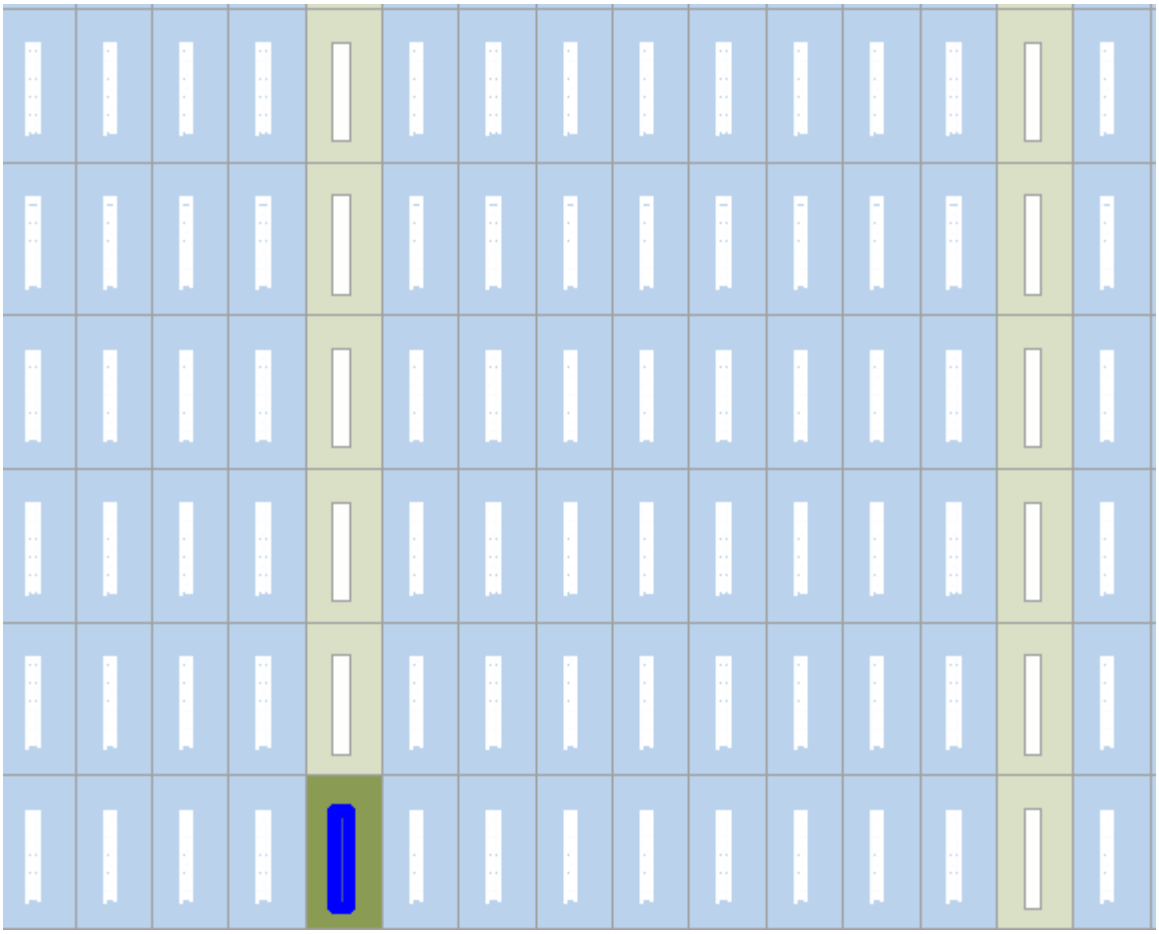

Рис.2.20 - Блоки пам’яті на кристалі FPGA (світло зелені стовбці)

Якщо зайти у властивості окремого блоку пам’яті, можна побачити його будову - рис.2.21. Зверніть увагу, що блоки пам’яті містять синхронні по фронту D-тригери для даних на виходах пам’яті, тож тригери загального призначення з цією метою не використовуються.

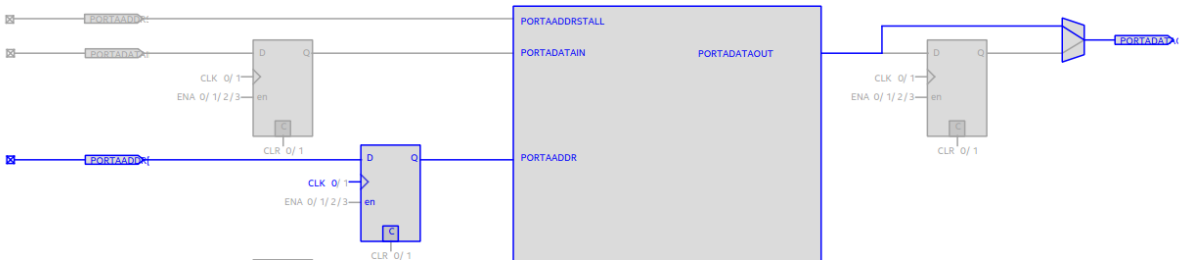

Рис.2.21 - Будова окремого блоку пам’яті відображена в Chip Planner Intel FPGA

2.2.10 Шаблони Verilog коду для опису цифрових схем в Quartus Prime

Раніше уже кілька разів зазначалося, що існують певні шаблони коду на мові Verilog для опису різних компонентів цифрових мікросхем і для правильного синтезу необхідно вживати коректні шаблони Verilog коду.

Приклади таких шаблонів Verilog коду можна переглянути в Quartus Prime. Для цього необхідно відкрити в Quartus Prime певний Verilog файл, та обрати пункт меню Edit -> Insert Template. Відкриється вікно з шаблонами коду на різних мовах (Verilog, System Verilog, VHDL, AHDL). Для мови Verilog обираємо Verilog HDL і на вкладці Full Designs бачимо шаблони Verilog для опису різних компонентів цифрових мікросхем - рис.2.22.

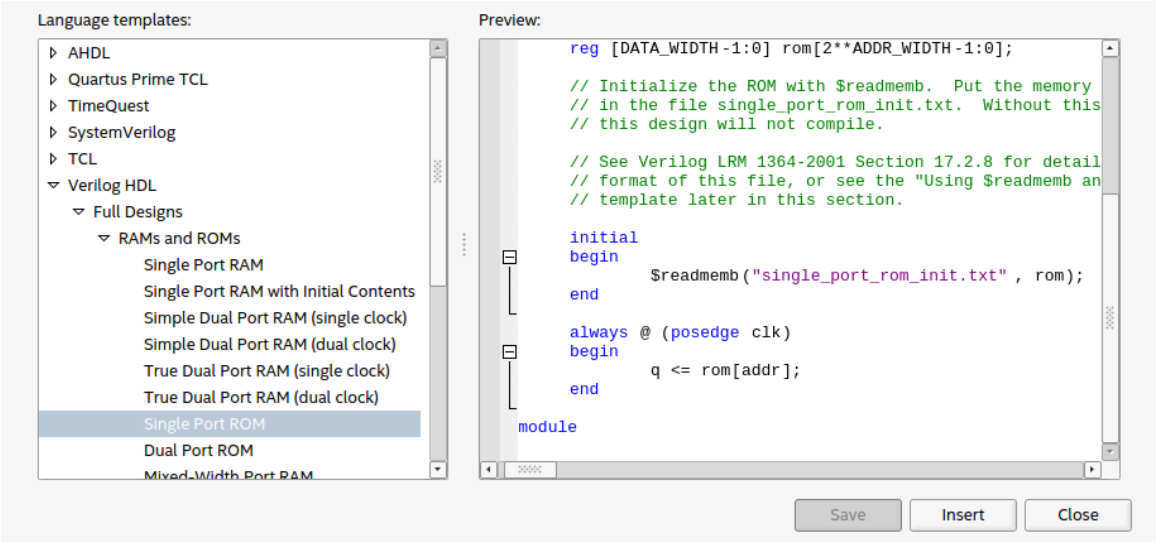

Рис.2.22 - Шаблони Verilog коду для опису різних компонентів цифрових схем

2.2.11 Цифровий генератор синусоїдального сигналу

Для генерації синусоїдального сигналу необхідно через однакові проміжки часу Td (період дискретизації) видавати на ЦАП цифрові коди значень синусоїди, які ЦАП перетворюватиме в напругу. Частота видачі цифрових кодів на вхід ЦАП називається частотою дискретизації $F_d = \frac{1}{T_d}$. Описана ідея проілюстрована на рис.2.23.

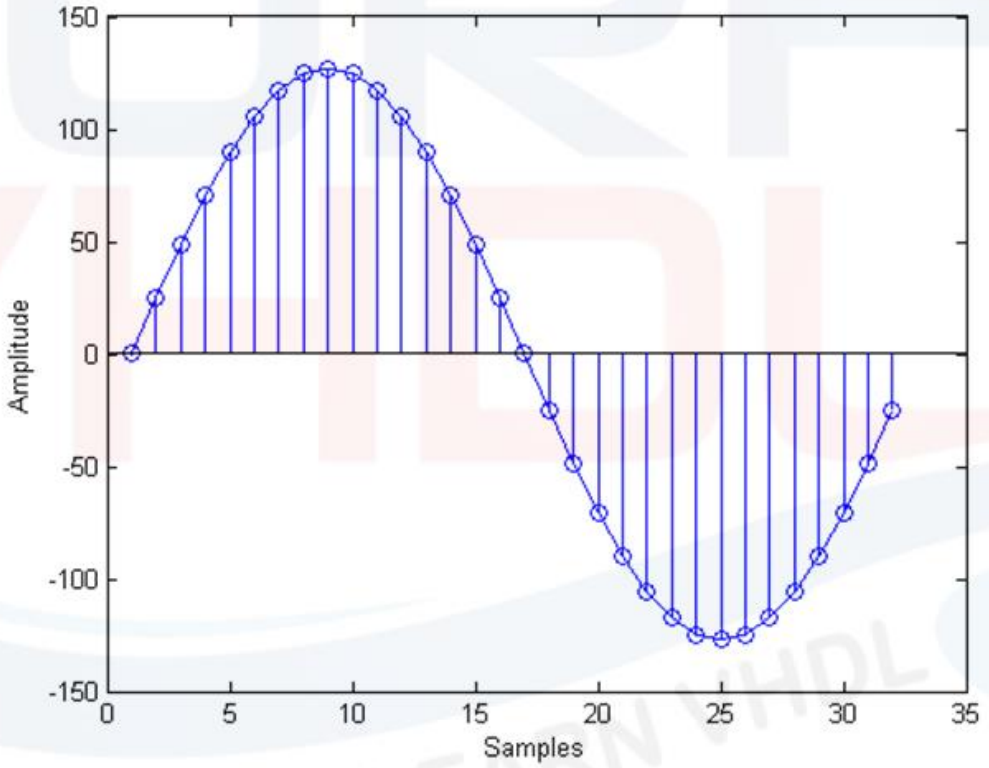

Рис.2.23 - Синусоїда (суцільна лінія) та її значення в дискретні моменти часу (круглі маркери) з періодом дискретизації Td (інтервал часу між двома круглими маркерами)

Насправді, на рис.2.23 представлений спрощений підхід. В реальній схемі необхідно врахувати кілька особливостей.

По-перше, N-розрядний цифровий код на вході ЦАП може приймати обмежену кількість значень від 0 до 2^N - 1. Загалом 2^N значень. Це означає, що значення напруги на виході ЦАП будуть округлюватись до одного з цих значень. Нагадаємо, що конкретне значення напруги на виході N-розрядного ЦАП можна розрахувати за формулою (2.0). Оскільки протягом проміжку часу Td на вході ЦАП присутній

постійний цифровий код, це означає, що на виході ЦАП протягом проміжку часу Td буде постійна напруга, що залежить від цифрового коду за формулою (2.0), а значить значення напруги на виході ЦАП будуть мати вигляд сходинок - рис.2.24.

По-друге, ЦАП побудований на базі резисторів може видавати лише позитивну напругу. Тому до сигналу на графіку з рис.2.23 необхідно додати постійну складову рівну амплітуді синусоїдального сигналу. Результат такого зсуву наведений на рис.2.24. Тепер синусоїда приймає лише позитивні значення.

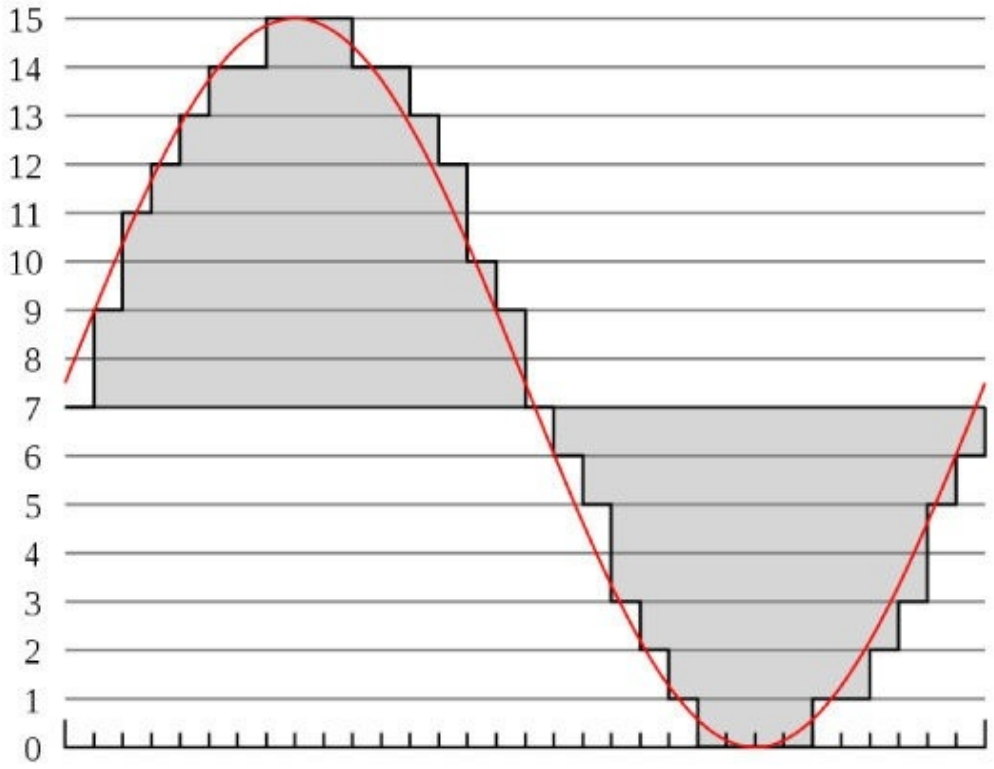

Рис.2.24 - Значення напруги синусоїдального сигналу на виході 4-розрядного ЦАП (напруга на виході ЦАП має 16 значень, що відповідають цифровим кодам від 0 до 15) з частотою дискретизації Fd ($F_d = \frac{1}{T_d}$, де Td - інтервал часу між видачею на ЦАП сусідніх цифрових кодів)

Описану вище логіку роботи цифрового генератора синусоїдального сигналу досить просто реалізувати на базі лічильника імпульсів та мікросхеми постійної пам’яті. Вихід лічильника імпульсів підключається до входу адреси постійної пам'яті. В комірках постійної пам’яті у вигляді цифрових кодів зберігаються значення синусоїди. Вихід даних постійної пам’яті підключається до цифрового входу ЦАП. Описана структура зображена на рис.2.25.

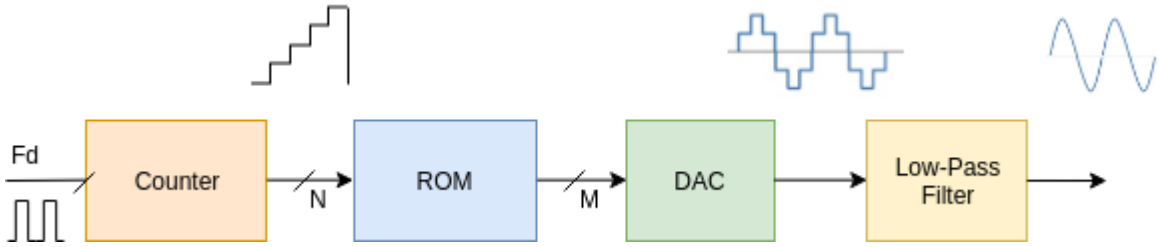

Рис.2.25 - Структурна схема цифрового генератора синусоїдальної напруги

По кожному активному фронту сигналу синхронізації (що має частоту Fd) вміст лічильника, що визначає адресу комірки постійної пам’яті, збільшується на одиницю (формується нова адреса) і на виході постійної пам’яті з'являється цифровий код, що зберігається комірці з такою адресою. Цифровий код з виходу постійної пам’яті подається на ЦАП, а ЦАП формує нове значення напруги синусоїди на своєму виході.

Вихідний код цифрового генератора синусоїдального сигналу на мові Verilog наведено в лістингу 2.4.

Лістинг 2.4 - Вихідний код генератора синусоїдального сигналу на мові Verilog

```
module sin_gen(i_clk, i_rst_n, o_dac);

input           i_clk;
input           i_rst_n;
output  [3:0]   o_dac;

reg     [3:0]   sin_table_rom[1023:0];
reg     [9:0]   phase;
reg     [3:0]   dac_data;

assign  o_dac = dac_data;

initial $readmemh("sin_table_4bit.hex", sin_table_rom);

always @(posedge i_clk)
    dac_data <= sin_table_rom[phase];

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

endmodule
```

Результат синтезу вихідного коду з лістингу 2.4 наведено на рис.2.26.

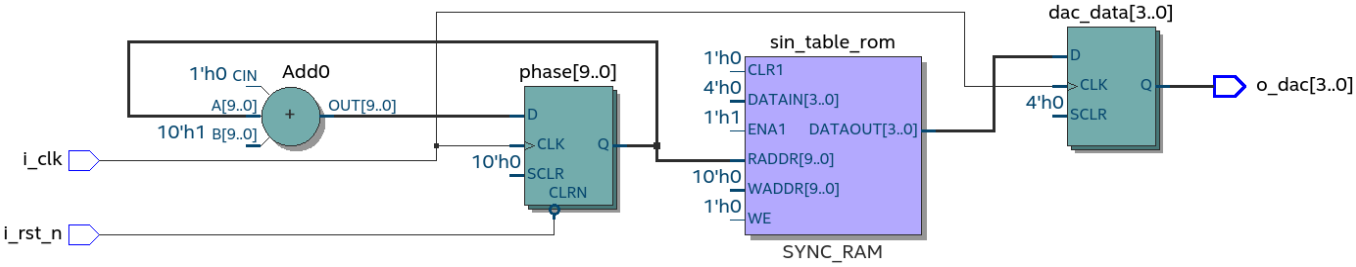

Рис.2.26 - Результат синтезу вихідного коду з лістингу 2.4

Як видно з рис.2.26 лічильник, що формує адресу постійної пам’яті, реалізовано на базі регістру `phase` і суматора. Вихід `o_dac` необхідно підключити до цифрового входу ЦАП.

Розрядність комірок пам’яті в лістингу 2.4 дорівнює 4 біт, оскільки налагоджувальні плати DE0-CV та DE10-Lite містять 4-розрядний ЦАП VGA інтерфейсу (див. розділ 2.3.3). За бажанням не складно збільшити розрядність цифрових кодів, що представляють синусоїду.

Кількість комірок масиву `sin_table_rom` в лістингу 2.4 дорівнює 1024. Це означає, що з такого вихідного коду синтезується пам'ять з 10-розрядним входом адреси, яка містить 1024 комірки. В цій пам’яті будуть зберігатися значення 1024 точок повного періоду синусоїдального сигналу. Відповідно період створеного синусоїдального сигналу буде дорівнювати кількості точок у періоді, що помножена на інтервал часу між двома сусідніми точками (Td). Отже, період створеного синусоїдального сигналу в даному випадку буде дорівнювати $T = 1024 \cdot T_d$.

За необхідності вихід ЦАП можна підключити до фільтру низької частоти, щоб прибрати (відфільтрувати) “сходинки”, які можна спостерігати на рис.2.24. Постійну складову синусоїдального сигналу на виході ЦАП можна прибрати підключивши конденсатор послідовно з виходом ЦАП (найпростіший фільтр високої частоти, що не пропускатиме постійну складову напруги).

Вміст файлу ініціалізації постійної пам’яті `sin_table_4bit.hex` наведено за посиланням [2.2]. Розрахувати значення точок синусоїди, що містяться в цьому файлі можна двома шляхами.

По-перше, можна скористатися сервісом за посиланням [2.3]. Вікно сервісу зображено на рис.2.27. Необхідно вказати кількість точок на період синусоїди (**Number of points**, в нашому випадку 1024 точки), кількість цифрових кодів на рядок файлу (**Numbers Per Row**, в нашому випадку 1) та цифровий код, що відповідає макс. значенню синусоїди, рівний 2^N - 1, де N - розрядність ЦАП (**Max Amplitude**, для 4-розрядного ЦАП максимальний код дорівнює 15). Після введення необхідних значень потрібно натиснути “Submit”. Результат роботи сервісу наведений на рис.2.28.

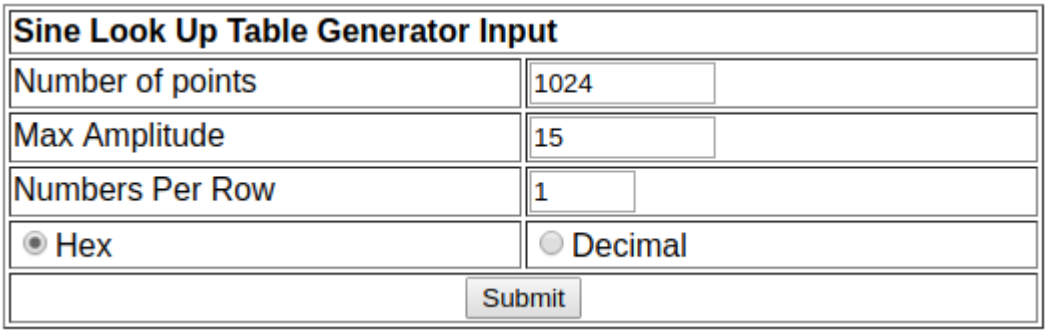

Рис.2.27 - Вікно сервісу розрахунку точок синусоїди для файлу ініціалізації постійної пам’яті цифрового генератора

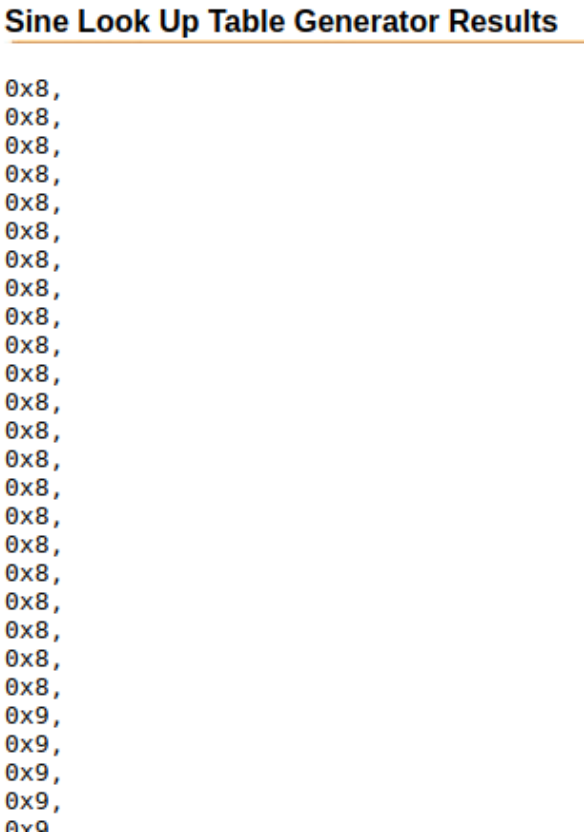

Рис.2.28 - Результат роботи сервісу розрахунку точок синусоїди

Результат роботи сервісу [2.3], зображений на рис.2.28, необхідно скопіювати в текстовий файл і використовуючи інструменти автоматичного пошуку з заміною прибрати префікси 0x та коми після кожного значення (замінити їх на пробіл, або відсутність символу).

По-друге, значення точок синусоїди для ініціалізації постійної пам'яті цифрового генератора можна розрахувати самостійно.

Відомо, що залежність функції синуса від часу можна представити за допомогою формули (не забувайте, що синус приймає значення від -1 до 1):

$$x(t) = sin(2 \cdot \pi \cdot F \cdot t), \quad (2.4)$$

де F - частота синусоїдального сигналу в Герцах;

T = 1/F - період синусоїдального сигналу.

У випадку синусоїди, що представлена лише дискретними точками, відстань між якими дорівнює періоду частоти дискретизації T_s (рис.2.23), будь який момент часу t від початку генерації синусоїди можна округлити до значення $k \cdot T_s$, де k - кількість періодів T_s, що пройшли з нульового моменту часу. Чим менше значення T_s, тим точніше можна представити будь яке значення часу t з використанням такого підходу.

Отже підставивши в формулу (2.4) $k \cdot T_s$ замість t, отримаємо формулу для представлення синусоїди визначеної дискретним набором точок:

$$x(k) = sin(2 \cdot \pi \cdot F \cdot k \cdot T_s) = sin\left(2 \cdot \pi \cdot k \cdot \frac{F}{F_s}\right), \quad (2.5)$$

де k - номер точки синусоїди, що приймає значення від 0 до K-1 (K - кількість точок на період синусоїди);

F - частота синусоїдального сигналу в Герцах;

T_s - період дискретизації синусоїди в секундах. Іншими словами - це інтервал часу між сусідніми точками, що представляють синусоїду (див. рис.2.23);

F_s - частота дискретизації синусоїди в Герцах. $F_s = \frac{1}{T_s}$

На перший погляд з формули (2.5) уже можна розраховувати значення точок синусоїди. В цій формулі присутня лише одна змінна - номер точки k. Інші величини - відомі константи. Адже нам відома частота синусоїди F, яку хочемо генерувати.

Період синусоїдального сигналу T містить K точок, відстань між якими однакова і рівна T_s. Це означає, що період T складається з K періодів T_s, тобто $T = K \cdot T_s$. Отже задавшить періодом синусоїдального сигналу T і кількістю точок K на період, можна розрахувати значення T_s за формулою:

$$T_s = \frac{T}{K} \tag{2.6}$$

Підставимо (2.6) в (2.5), отримаємо:

$$x(k) = sin\left(2 \cdot \pi \cdot \frac{k}{K}\right) \tag{2.7}$$

Далі необхідно врахувати, що ЦАП на основі резисторів не може створювати від'ємну напругу. Отже необхідно додати до синусоїди постійне зміщення, щоб функція приймала лише додатні значення (про це вже згадувалося вище).

Формула (2.7) приймає значення від -1 до 1. Створимо функцію, що примає лише додатні значення додавши до (2.7) постійне зміщення рівне 1:

$$y(k) = sin\left(2 \cdot \pi \cdot \frac{k}{K}\right) + 1 \tag{2.8}$$

Отримаємо функцію, що змінюється за законом синуса, приймає значення від 0 до 2 і має постійну складову рівну 1.

На останньому етапі необхідно перетворити значення синусоїди розраховані за формулою (2.8) в цифрові коди для N-розрядного ЦАП. Це можна зробити використовуючи спостереження:

1. Максимальному значенню $y(k)$, що дорівнює 2, повинен відповідати максимальний цифровий код N-розрядного ЦАП, тобто 2^N - 1;
2. Значенню $y(k)$ повинен відповідати цифровий код *code*;

Склавши пропорцію отримаємо:

$$code = y(k) \cdot \frac{2^N - 1}{2} \tag{2.9}$$

Отже обравши кількість точок на період **K** і частоту **F** синусоїдального сигналу можна розрахувати **Ts** за формулою (2.6) і за формулами (2.8) та (2.9) розрахувати значення цифрових кодів, що представляють кожну з K точок синусоїди (з номерами k від 0 до K-1). Вихідний код програми, що виконує такий розрахунок наведено за посиланням [2.4].

На рис.2.29-2.30 наведені результати моделювання цифрового генератора синусоїдального сигналу в симуляторі Modelsim. На графіках присутні два варіанти відображення шини провідників o_dac на яку видаються 4-розрядні цифрові коди для ЦАП. В одному варіанті відображення o_dac видно цифрові коди, що видаються на шину. Інший варіант відображення o_dac є представленням згаданих цифрових кодів у вигляді аналогового сигналу (імітація роботи 4-розрядного ЦАП). Видно, що сигнал на рисунках нижче ніби складається зі сходинок. Це обумовлено низькою розрядністю ЦАП (4 біта). Збільшення розрядності ЦАП до 12-24 бітів, або використання фільтру низької частоти на виході ЦАП дозволить вирішити цю проблему.

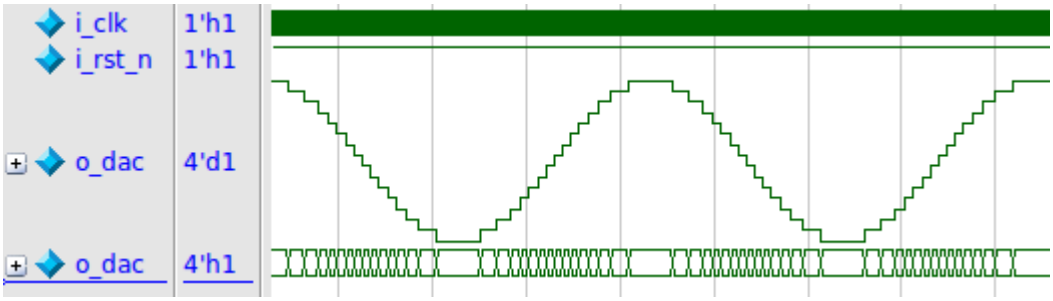

Рис.2.29 - Часова діаграма зміни сигналів в цифровому генераторі синусоїдального сигналу

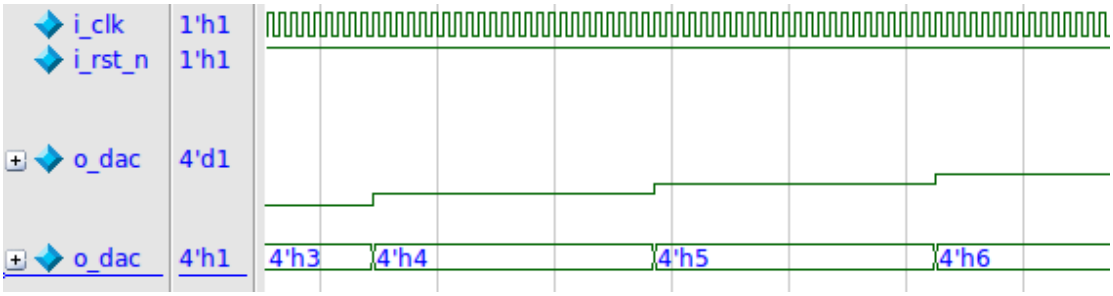

Рис.2.30 - Часова діаграма зміни сигналів в цифровому генераторі синусоїдального сигналу (збільшений масштаб)

2.2.12 Тестбенч на мові Verilog (initial процеси, цикли, затримки)

Вище було розглянуто, як використовувати мову Verilog для опису цифрових схем. Однак Verilog можна використовувати і для перевірки роботи створених схем. Мова йде про розробку так званих тестових стендів, тестбенчів (testbench), що створюють цифрові сигнали, які подаються на входи цифрової схеми і перевіряють чи правильні сигнали з'являються на виходах цифрової схеми. Тестбенчі представляють собою звичайні програми і не синтезуються в цифрову схему. Файли тестбенчів непотрібно додавати до файлів проекту в Quartus Prime.

Роботу цифрової схеми описаної мові Verilog та її тестбенчу можна промоделювати в спеціальній програмі, що називається **симулятор** (simulator). Існують безкоштовні симулятори (Icarus, Verilator) і комерційні (ModelSim, Incisive, VCS).

В лабораторних роботах ми будемо використовувати симулятор ModelSim Intel FPGA Starter Edition, що входить до набору програм Quartus Prime Lite Edition, який ви встановили в початковій лабораторній роботі. ModelSim Intel FPGA Starter Edition можна використовувати безкоштовно. Безкоштовна версія має певні обмеження, однак для простих проектів наших лабораторних робіт це не принципово. Інструкція по по користуванню ModelSim наведена в розділі 2.3.1 даної лабораторної роботи.

Всі конструкції мови Verilog можна поділити на дві категорії: **синтезовані** і **не синтезовані**. В цифрову схему можна перетворити (синтезувати) лише Verilog код, що складається з синтезованих конструкцій. Тому синтезовані конструкції використовують для опису цифрової схеми, а не синтезовані конструкції використовують для створення тестбенчу (хоча синтезовані конструкції теж можна використовувати при описі тестбенчу). Раніше в лабораторних роботах ми розглядали переважно синтезовані конструкції (за винятком процедурного `initial` блоку). Далі при описі конструкцій мови Verilog будемо вважати, що вони синтезовані, якщо прямо не зазначено протилежне.

Важливо розуміти, що для правильного синтезу цифрової схеми під час її опису необхідно застосовувати правильні шаблони коду для різних компонентів цифрових схем (див. розд. 2.2.10) і лише синтезовані конструкції мови Verilog. Важливо завжди уявляти в яку схему синтезується ваш Verilog код.

З іншого боку, описані обмеження не застосовуються при написанні тестбенчів. Тестбенчі на мові Verilog мають логіку роботи програм, в яких інструкції виконуються послідовно і можна викликати функції описані на мові Verilog (за бажанням, можна навіти викликати функції з бібліотек написаних на C за допомогою технології PLI).

Отже тестбенчі на мові Verilog дуже схожі на звичайне програмування. Однак є одна відмінність від звичайних програм - це концепція часу. Тетстбенчі моделюють роботу цифрової схеми описаної на Verilog протягом певного проміжку часу, створюють вхідні сигнали для схеми, що змінюються в часі і т.д. Для реалізації подібного функціоналу використовують концепцію затримок і очікування на події.

Приклад тестбенчу для перевірки роботи генератора синусоїдального сигналу з попереднього розділу наведено в лістингу 2.5.

Лістинг 2.5 - Тестбенч для генератора синусоїдального сигналу мові Verilog

```
`timescale 1ns / 1ps

module testbench;

parameter PERIOD = 20;

reg         i_clk, i_rst_n;
wire [3:0]  o_dac;

sin_gen  gen_inst(.i_clk (i_clk),
                  .i_rst_n (i_rst_n),
                  .o_dac (o_dac)
                  );

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 (10000) @(negedge i_clk);

    $finish;
end

endmodule
```

Тестбенч на мові Verilog описують всередині модуля у якого відсутні порти вводу-виводу. В нашому випадку це модуль з іменем `testbench`.

Всередині тестбенчу, як і в будь якому Verilog модулі можна створювати провідники типу **`wire`** і змінні типу **`reg`**.

Провідники типу **`wire`** підключають до вихідних портів екземпляру модуля цифрової схеми, що перевіряється і згадані виходи створюють на провідниках певні сигнали, зміну яких можна переглянути у вигляді часових діаграм. Змінні типу **`reg`** підключають до вхідних портів екземпляру модуля цифрової схеми, що перевіряється. Далі в тестбенчі в певні моменти часу записують нові значення в змінні типу **`reg`** і ці значення передаються в екземпляр модуля цифрової схеми, що перевіряється.

В тестбенчі необхідно створити екземпляр модуля цифрової схеми, робота якого буде перевірятися. Можна вважати, що модуль на мові Verilog - це креслення цифрової схеми. А от екземпляр модуля відповідає конкретній цифровій схемі виготовленій по цим кресленням. За бажанням можна створити кілька екземплярів одного модуля. Якщо наводити аналогії з мовою C++ та іншими мовами об'єктно-орієнтованого програмування, то опис Verilog модуля аналогічний опису C++ класу. А от екземпляр модуля аналогічний екземпляру класу.

Приклад створення екземпляру модуля на мові Verilog:

```
sin_gen  gen_inst(.i_clk (i_clk),
                  .i_rst_n (i_rst_n),
                  .o_dac (o_dac)
                  );
```

В даному прикладі створюється екземпляр `gen_inst` модуля `sin_gen`. Модуль `sin_gen` описаний в лістингу 2.4. З лістинга 2.4 видно, що модуль `sin_gen` має вхідні порти **`i_clk`** та **`i_rst_n`** та вихідний порт **`o_dac`**.

Під час створення екземпляру модуля спершу пишуть ім’я модуля, потім ім’я екземпляру модуля, а далі в круглих скобках виконують підключення сигналів тестбенча до вхідних і вихідних портів екземпляру модуля.

Підключення до портів екземпляру модуля виконується наступним чином. Пишуть ім’я порта екземпляру модуля, що починається з крапки, а далі в круглих скобках пишуть ім’я сигналу тестбенчу (типу **`wire`** або **`reg`**), що підключається до цього порта. Таким чином через кому необхідно підключити необхідні порти модуля. Якщо ви не плануєте використовувати певний порт модуля, можна його не підключати. Рекомендовано описати всі порти модуля, однак для тих портів, що не використовуються, не передавати сигнали в круглі скобки.

Вхідні сигнали для екземпляру модуля, що перевіряється описують в **`initial`** процедурних блоках.

Блоки **`initial`** не синтезуються і використовуються лише в тестбенчах. Коли САПР на зразок Quartus Prime зустрічає в Verilog файлі **`initial`** блок, такий блок коду просто ігнорується і не синтезується. Єдине виключення, коли **`initial`** блок враховується синтезатором - ініціалізація ROM пам’яті для FPGA (див. розд.2.29).

Блоки коду **`initial`** починають виконуватись одразу після початку симуляції (симуляцією називається моделювання в Modelsim). А симуляція починається в

нульовий момент часу. Для виконання певної інструкції в `initial` блоці в ненульовий момент часу можна використовувати затримки. Інструкції в `initial` блоці коду виконуються послідовно одна за одною (з цього правила можуть бути виключення, які для простоти поки не будемо розглядати). Виконання `initial` блоку завершується після виконання всіх його інструкцій. Однак якщо `initial` блок містить, наприклад, нескінченний цикл, такий `initial` блок буде виконуватись до самого завершення симуляції.

В тестбенчі може бути кілька `initial` блоків. Симулятор виконує `initial` блоки одночасно. Іншими словами кілька `initial` блоків працюють, які кілька потоків в багатопоточній програмі написаній на мові C. Зазвичай тестбенчі на мові Verilog включають більше ніж один `initial` блок коду. Отже бачимо, що для тестбенчів на мові `initial` притаманна багатопоточність.

Затримку у часі між виконанням інструкцій в `initial` блоці коду можна реалізувати за допомогою оператора #. Після оператора # вказують на скільки одиниць модельного часу необхідно затримати виконання наступної інструкції. Іншими словами, коли симулятор зустрічає оператор # , він затримує виконання всіх наступних інструкцій на кількість одиниць часу, що вказана після оператора #. По завершенню затримки інструкції продовжують виконання. Затримка в операторі # не обов'язково повинна визначатися константою. Це може бути змінна, або параметр мови Verilog. Подібні затримки використовуються лише під час моделювання в тестбенчі і не синтезуються.

Розмірність одиниць модельного часу, в яких вимірюють затримки, вказується першим аргументом конструкції `` `timescale ``, що визначається на самому початку файлу Verilog тестбенчу.

Конструкція `` `timescale `` має наступний синтаксис:

```
`timescale 1ns / 1ps
```

Де перший аргумент `timescale` (в даному випадку `1ns`) - розмірність одиниць модельного часу, в яких визначаються затримки в операторі #. Наприклад, конструкція **#6** для даного формату `timescale` означає затримку в 6 наносекунд.

Другий аргумент `timescale` (в даному випадку `1ps`) - це точність визначення затримок. Іншими словами, в даному прикладі найменший крок зміни затримки буде 1 пікосекунда.

Інший спосіб реалізації затримок в Verilog тестбенчах - це очікування на подію за допомогою оператора **@**. Робота оператора **@** описана в розділі 1.2.2 попередньої лабораторної роботи. Коли симулятор зустрічає в Verilog коді оператор **@**, виконання наступних операторів зупиняється, доки не відбудеться подія вказана після оператора **@**. Наприклад, конструкція `@(negedge i_clk)` блокує виконання всіх наступних операторів Verilog до приходу найближчого заднього фронту сигналу `i_clk`.

Розглянемо `initial` блок коду з лістингу 2.5, в якому створюється тактовий сигнал синхронізації для екземпляру модуля генератора синусоїдального сигналу.

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

В нульовий момент часу, на початку симуляції, починається виконання `initial` блоку,що призводить до виконання операції `i_clk = 0`. Далі симулятор потрапляє у нескінченний цикл `forever`, ітерації якого будуть виконуватись до завершення симуляції. На початку кожної ітерації нескінченного циклу симулятор зустрічає затримку рівну половині значення змінної `PERIOD`. По завершенню затримки виконання Verilog коду продовжується і виконуються інвертування значення змінної `i_clk: i_clk = ~i_clk`. Після цього симулятор переходить на наступну ітерацію нескінченного циклу і знову виконується затримка. Таким чином описаний код призводить до того, що значення `i_clk` інвертується кожні пів періоду.

Константа `PERIOD` визначається наступним чином:

```
parameter PERIOD = 20;
```

Параметри у мові Verilog ми розглянемо в наступних лабораторних роботах. Поки що можна вважати параметри способом визначення іменованих констант.

Розглянемо ще один `initial` блок процесу з лістингу 2.5.

```
initial begin
    i_rst_n = 1'b0;

    @(negedge i_clk) i_rst_n = 1;

    repeat (10000) @(negedge i_clk);

    $finish;
end
```

В наведеному `initial` блоці на початку симуляції, в нульовий момент часу, в змінну `i_rst_n` записується лог.0. Далі очікується найближчий задній фронт `i_clk` і після цієї події в змінну `i_rst_n` записується лог.1. Таким чином формується сигнал зкидання. Після цього в циклі `repeat` виконується визначена кількість ітерацій (в даному випадку `10000` ітерацій). На кожній ітерації очікується задній фронт сигналу `i_clk`. Таким чином реалізується затримка в `10000` періодів сигналу синхронізації `i_clk`. Після завершення циклу `repeat` викликається системна функція `$finish`, що призводить до завершення симуляції і зупинки виконання всіх `always` та `initial` блоків коду. До речі, за рідким виключенням системні функції Verilog не синтезуються (тобто використовуються лише в симуляції.)

Результат симуляції тестбенчу генератора синусоїдального сигналу наведений на рис.2.29-2.30.

2.2.13 Цифровий генератор синусоїдального сигналу з керованою частотою

В одному з попередніх розділів розглянуто найпростіший цифровий генератор синусоїдального сигналу з фіксованою частотою. Однак більшу цікавість представляє цифровий генератор з можливістю змінювати частоту синусоїди в процесі роботи. Таку схему називають **Numerically Controlled Oscillator** (NCO).

NCO є основою сучасних цифрових генераторів аналогового сигналу, за допомогою NCO можна створити радіопередавач з частотною (FM) або амплітудною (AM) модуляцією та більш складні системи радіопередачі, цифрову схему генератора з фазовим автопідлаштуванням частоти ФАПЧ (Phase-locked loop, PLL) та багато інших цікавих і корисних схем.

Структурна схема NCO наведена на рис.2.31.

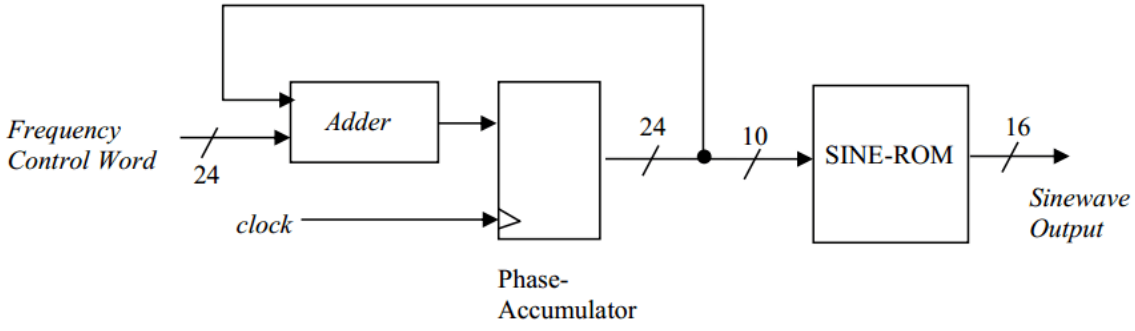

Рис.2.31 - Структурна схема цифрового генератора синусоїдального сигналу з керованою частотою (NCO)

Вихідний код NCO на мові Verilog, що реалізує структурну схему з рис.2.31, наведено в лістингу 2.6. Результат синтезу такого вихідного коду наведено на рис.2.32.

Лістинг 2.6 - Вихідний код цифрового генератора синусоїдального сигналу з керованою частотою на мові Verilog

```
module nco(i_clk, i_rst_n, i_ctrl_code, o_dac);

input           i_clk;
input           i_rst_n;
input   [31:0]  i_ctrl_code;
output  [3:0]   o_dac;

reg     [3:0]   sin_table_rom[1023:0];
reg     [31:0]  phase;
reg     [3:0]   dac_data;
reg     [31:0]  phase_step;

assign  o_dac = dac_data;

initial $readmemh("sin_table_4bit.hex", sin_table_rom);

always @(posedge i_clk)
    dac_data <= sin_table_rom[phase[31:22]];

always @(posedge i_clk, negedge i_rst_n) begin
    if(~i_rst_n) begin
        phase_step <= 0;
    end else begin
        phase_step <= i_ctrl_code;
    end
end

always @(posedge i_clk, negedge i_rst_n) begin
    if(~i_rst_n) begin
        phase <= 0;
    end else begin
        phase <= phase + phase_step;
    end
end

endmodule
```

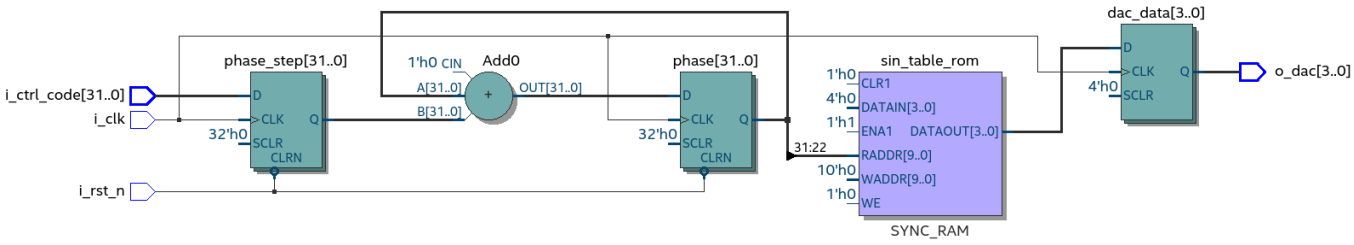

Рис.2.32 - Результат синтезу вихідного коду з лістингу 2.6

Якщо порівняти структуру звичайного цифрового генератора синусоїди (рис.2.25) і його вихідний код (лістинг 2.4) зі структурою (рис.2.31) і вихідним кодом NCO (лістинг 2.6) можна зробити висновок, що схеми дуже схожі (обидві схеми побудовані на основі лічильника і постійної пам'яті), однак є наступні відмінності:

1. Висока розрядність лічильника у схемі NCO (32 біта);
2. У схемі звичайного генератора синусоїди по кожному активному фронту сигналу синхронізації вміст лічильника збільшується на 1. Однак у схемі NCO по кожному активному фронту сигналу синхронізації вміст лічильника збільшується уже на довільну величину, що подається на один із входів NCO і має розрядність 32 біта (тобто це може бути дуже велике число);
3. Розрядність шини адреси постійної пам’яті в обох схемах однакова (10 біт), однак в схемі NCO для формування адреси постійної пам’яті використовуються лише старші 10 біт 32-розрядного лічильника.

Розглянемо принцип роботи NCO. З логіки роботи звичайного цифрового генератора синусоїдального сигналу (розд.2.2.11) видно, що період синусоїдального сигналу в такому генераторі визначається періодом переповнення лічильника, що задає адресу комірок постійної пам’яті. Перша точка періоду синусоїдального сигналу відповідає нульовому значенню адреси ROM, а остання точка періоду синусоїдального сигналу відповідає найбільшому значенню N-розрядної адреси ROM ($2^N - 1$). Якщо адреса ROM приймає максимальне значення, по наступному активному фронту сигналу синхронізації вміст лічильника переповнюється в нуль і формування періоду синусоїдального сигналу починається з початку. Отже частота синусоїди (кількість періодів синусоїди за секунду) визначається частотою переповнення лічильника адреси постійної пам’яті (кількість переповнень лічильника за секунду).

Якщо по кожному активному фронту сигналу синхронізації збільшувати вміст лічильника `phase`, що формує адресу ROM, не на одиницю, а на значення `phase_step`, це дозволить керувати частотою переповнення лічильника і, як наслідок, частотою синусоїдального сигналу. Наприклад, для переповнення 3-х розрядного лічильника `phase` (що може зберігати 8 значень від 0 до 7), необхідно 8 раз додати одиницю до нульового вмісту лічильника (після додавання 8-ї одиниці в 3-розрядному лічильнику знову буде значення 0), або 4 рази додати значення 2, або 2 рази додати значення 4. Це є і ключова ідея роботи NCO.

Такий лічильник в NCO, до якого по кожному активному фронту сигналу синхронізації додається певне число, називають **акумулятором фази**.

Враховуючи наведені вище твердження можна записати наступний вираз:

$$\frac{2^N}{phase\_step} = \frac{T}{T_{in}} \tag{2.10}$$

Формулу (2.10) можна переписати наступним чином:

$$\frac{2^N}{phase\_step} = \frac{F_{in}}{F} \tag{2.11}$$

T - період синусоїдального сигналу, сек;

F - частота синусоїдального сигналу, Гц;

Tin - період сигналу синхронізації, що подається на тактовий вхід лічильника, сек;

Fin - частота сигналу синхронізації, що подається на тактовий вхід лічильника, Гц;

2^N - кількість значень, які може приймати N-розр. лічильник (від 0 до $2^N - 1$);

`phase_step` - число що додається до вмісту лічильника по кожному активному фронту сигналу синхронізації;

З формули (2.11) можна розрахувати значення `phase_step` для отримання необхідної частоти синусоїдального сигналу при відомій частоті тактового сигналу Fin:

$$phase\_step = 2^N \cdot \frac{F}{F_{in}} \tag{2.12}$$

З формули (2.11) також видно, що найменший крок зміни частоти буде дорівнювати:

$$F = \frac{F_{in}}{2^N} \tag{2.13}$$

Іншими словами, найменший крок зміни частоти синусоїдального сигналу дорівнює найменшому значенню частоти синусоїдального сигналу (при `phase_step=1`).

Наприклад, при використанні 32-розрядного лічильника `phase` і 50 МГц вхідного сигналу синхронізації, що подається на тактовий вхід лічильника, можна отримати точність формування частоти синусоїдального сигналу рівну $50\ 000\ 000 / 2^{32} = 0.01$ Гц.

Ще одна ідея, важлива для розуміння роботи NCO, полягає у тому, що в якості адреси постійної пам’яті використовуються лише старші розряди лічильника `phase`. Це необхідно для застосування пам’яті адекватного розміру, адже якщо для формування адреси використовувати всі 32 розряди лічильнику `phase`, знадобилася б пам’ять розміром 2^{32} комірок. Використовуючи лише старші розряди лічильника у якості адреси постійної пам’яті, можна обмежитися меншим обсягом пам’яті при цьому зберігши велику точність формування частоти. Наприклад, у випадку використання у якості адреси старших 10-ти розрядів 32-розрядного лічильника знадобиться пам’ять, що має 1024 комірки і 10-розрядний вхід адреси. При цьому точність визначення частоти синусоїдального сигналу залишиться рівна 0.01 Гц.

Структура акумулятора фази (лічильника) в NCO наведена на рис.2.33.

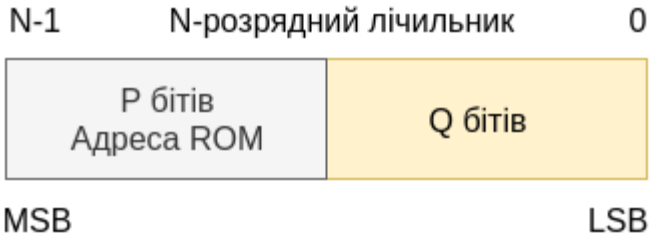

Рис.2.33 - Структура акумулятора фази (лічильника) в NCO

З рис.2.33 видно, що N-розрядний лічильник складається з двох частин. Старші P бітів задають номер комірки постійної пам’яті. Молодші Q бітів визначають частоту з якою збільшується значення старших P розрядів. Загалом N = P + Q.

З принципу роботи лічильника відомо, що вміст старших P розрядів буде збільшуватись на 1, якщо в молодших Q розрядах присутні одиниці і надходить активний фронту сигналу синхронізації. Іншими словами частота інкрементування старших P бітів (частота формування нових значень адреси ROM) буде залежати від частоти переповнення молодших Q бітів.

Робота описаного цифрового генератора синусоїдального сигналу з керованою частотою продемонстрована за посиланням [2.7].

З використанням подібного підходу можна створювати синусоїдальні сигнали частота яких не перевищує половини частоти вхідного тактового сигналу. За обґрунтуванням такого обмеження лежить об'ємна теорія, тож зараз краще сприйняти дане твердження на віру.

2.3 Практична частина

2.3.1 Симуляція проектів в ModelSim

Як зазначалося в розділі 2.2.12 роботу цифрової схеми і тестового стенду (тестбенчу) можна промоделювати в програмі-симуляторі. В даних лабораторних роботах використовується симулятор Modelsim.

Можливості комерційних симуляторів в області моделювання і аналізу цифрових схем дуже широкі, тож зараз будуть описані лише базові кроки для отримання найпростішого результату.

Для реалізації симуляції необхідно виконати наступні етапи:

1. Виконати компіляцію Verilog файлів, що містять модулі для симуляції;
2. Завантажити зкомпільовані модулі в Modelsim;
3. Додати на часову діаграму сигнали, які необхідно спостерігати;
4. Запустити на симуляцію модуль тестбенчу;

Розглянемо детальніше необхідність компіляції Verilog файлів. Перші Verilog симулятори працювали, як інтерпретатори, зчитуючи і виконуючи інструкції безпосередньо з Verilog файлу. Однак це дуже не продуктивний підхід, що не дозволяє отримати високу швидкість моделювання. Сучасні комерційні симулятори перетворюють Verilog на двійковий код, що виконується безпосередньо на процесорі. Цей процес і називається компіляцією.

Перелічені вище етапи, необхідні для запуску симуляції, можна за бажанням виконати з графічного інтерфейсу Modelsim. Однак набагато простіше один раз написати короткий командний скрипт і потім лише запускати його на виконання замість того, щоб раз за разом повторювати однотипні дії. Командні скрипти для Modelsim пишуться на мові TCL і зберігаються у файлах з розширенням *.do

Командний файл sim.do для моделювання цифрового генератора синусоїдального сигналу (лістинг 2.4) і відповідного тестбенчу (лістинг 2.5) наведено в лістингу 2.7.

Запустити командний файл sim.do на виконання симулятором дуже просто. Достатньо знаходячись у каталозі, в якому зберігаються Verilog файли, виконати х терміналу (консолі) команду `vsim -do sim.do`

Лістинг 2.7 - Вихідний код командного файлу sim.do для симулятора Modelsim

```
vlib work

vlog  sin_gen.v sin_gen_tb.v

vsim -novopt work.testbench

add wave /testbench/i_clk
add wave /testbench/i_rst_n
add wave -format Analog-Step -height 84 -max 15.0 -radix unsigned
/testbench/o_dac

run -all
```

Запустити командний файл sim.do на виконання можна і з графічного інтерфейсу Modelsim. Для цього необхідно обрати пункт головного меню Tools -> Tcl -> Execute Macro. Далі у вікні, що відкриється, необхідно обрати файл sim.do і натиснути “Open”.

Після того, як у Verilog коді буде викликана системна функція `$finish`, симуляція завершиться і з’явиться вікно із запитанням “Are you sure want to finish?”. Необхідно натиснути на кнопку No і переглянути часові діаграми зміни сигналів у вікні Wave.

Проаналізуємо детальніше командний файл симуляції з лістингу 2.7.

- Команда `vlib work` створює бібліотеку з іменем `work` куди буде виконуватись компіляція Verilog файлів.
- Команда `vlog  sin_gen.v sin_gen_tb.v` запускає компіляцію Verilog файлів у створену раніше бібліотеку.
- Команда `vsim -novopt work.testbench` запускає на симуляцію модуль з іменем `testbench` (див. лістинг 2.5), що міститься в бібліотеці `work`. Виконання цієї команди призводить до того, що в симулятор Modelsim з бібліотеки `work` завантажується зкомпільований модуль `testbench` та інші модулі, екземпляри яких створені в `testbench`. Ключ `-novopt` вказує симулятору не виконувати оптимізацію (під час оптимізації Modelsim часто прибирає/оптимізує частину сигналів, які можуть бути цікавими для аналізу. Тому спершу краще виконувати симуляцію без оптимізації).
- Команда `add wave /testbench/i_clk` додає сигнал `i_clk`, що міститься в модулі `testbench` у вікно часових діаграм `wave`.

- Команда `add wave -format Analog-Step -height 84 -max 15.0 -radix unsigned /testbench/o_dac` теж додає шину `o_dac` у вікно часових діаграм, але при цьому вказує, що необхідно відображати цифрові коди на шині `o_dac` у вигляді аналогового сигналу (моделювання ЦАП).
- Команда `run -all` запускає симуляцію.

Результат моделювання в Modelsim після запуску командного файлу з лістингу 2.7 наведено на рис.2.29.

2.3.2 Налаштування синтезу пам’яті в Quartus Prime

По шаблону Verilog коду з лістингу 2.3 по замовчуванню синтезується ROM на базі блочної пам’яті і ініціалізується вмістом файлу ініціалізації.

За необхідності можна налаштувати Quartus Prime на реалізацію пам'яті на базі табличок істинності і тригерів (така потреба виникає дуже рідко). Для цього необхідно зайти в налаштування проекту (Settings), на вкладці Compiler Settings натиснути кнопку Advanced Settings (Synthesis) і у вікні, що відкриється, поставити в значення “off” пункти “Auto RAM Replacement”, “Infer RAMs from Raw Logic”, “Auto ROM Replacement”, “Infer ROMs from Raw Logic”. В кінці натиснути кнопки “On” та “Apply”.

В FPGA мікросхемах серії MAX10 по замовчуванню пам’ять може синтезуватися на базі тригерів і табличок істинності без використання ресурсів блочної пам'яті. Для виправлення цього недоліку перейдіть на вкладку вибору FPGA мікросхеми (Assignments -> Device), далі натисніть кнопку “Device and Pin Options”, у вікні, що відкриється, оберіть вкладку Configuration і виставте пункт Configuration Mode в значення “Single Uncompressed Image with memory initialization (512 Kbits UFM)”, натисніть кнопку Ok.

2.3.3 Цифро-аналоговий перетворювач у налагоджувальних платах з VGA

Опис і розрахунок цифро-аналогового перетворювача з ваговими резисторами наведено в розділі 2.2.7 даної лабораторної роботи.

Налагоджувальні плати DE0-CV та DE10-Lite, що використовуються для виконання лабораторних робіт, містять інтерфейс VGA, що використовуються для виведення зображення на комп'ютерний монітор.

Детально інтерфейс VGA ми розглянемо в одній з наступних лабораторних робіт. Зараз важливо, зрозуміти, що у VGA інтерфейсі колір кожного пікселя визначається в RGB форматі, як суміш червоного, зеленого та синього кольорів. Інтенсивність кожного кольору визначається напругою на відповідному виході VGA інтерфейсу.

На виході **R** присутня напруга, що визначає інтенсивність червоного кольору. На виході **G** присутня напруга, що визначає інтенсивність зеленого кольору. На виході **B** присутня напруга, що визначає інтенсивність синього кольору. Напруга 0 Вольт відповідає відсутності кольору. Напруга 0.7 Вольт відповідає максимальній яскравості кольору.

Виходи VGA інтерфейсу з боку джерела відеосигналу (в нашому випадку налагоджувальна плата) наведені на рис.2.34. Як видно з рисунку, виводи 5, 6, 7, 8, 10 підключені до “землі” (GND). Таким чином напруга, що визначає інтенсивність червоної складової кольору пікселя вимірюється між виводом 1 (вихід R) і виводом 6 (GND) VGA інтерфейсу.

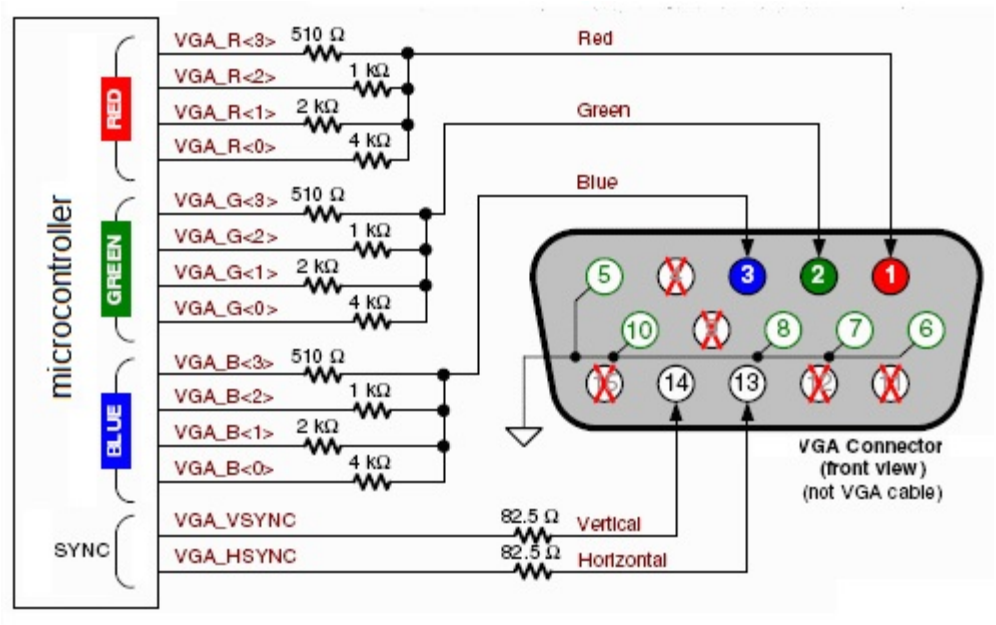

Рис.2.34 - 4-розрядний VGA ЦАП та виводи VGA інтерфейсу

Для кожної складової кольору пікселя (R, G, або B) в джерелі VGA відеосигналу присутній ЦАП, що перетворює цифровий код інтенсивності певного кольору в напругу. Іншими словами, кожне джерело VGA відеосигналу містить 3 ЦАП: для червоної, зеленої та синьої складової кольору (або один трьохканальний ЦАП).

Налагоджувальні плати DE0-CV та DE10-Lite містять 4-розрядні VGA цифро-аналогові перетворювачі. Це означає, що інтенсивність кожної складової кольору пікселя (R, G, або B) може приймати усього 16 значень. Сучасні VGA ЦАПи, на зразок ADV7123, мають 10-розрядні шини для визначення інтенсивності R, G та B складових кольору пікселя.

На рис.2.34 показана реалізація 4-розрядних ЦАП-ів для R, G та B каналів і піключення виходів зазначених ЦАП-ів до відповідних виводі VGA роз'єму джерела відеосигналу.

На рис.2.34 цифрові коди на входах ЦАП формує мікроконтролер, а у наших лабораторних роботах замість мікроконтролеру використаємо FPGA мікросхему.

Цифровий вхід для ЦАП, що визначає інтенсивність червоної складової кольору пікселю часто називають VGA_R. Аналогічно, цифрові входи, що формують зелену та синю складові кольору пікселю часто називають VGA_G та VGA_B.

Якщо на вході ЦАП (наприклад, вхід ЦАП червоного каналу VGA_R) присутній максимальний цифровий код, в режимі холостого ходу (без підключення навантаження) на виході буде напруга лог.1 (3.3 Вольт). Але якщо до виходу ЦАП підключити навантаження 75 Ом (стандартне навантаження виходів R, G, B), максимальне значення напруги на виході становитиме необхідні 0.7 Вольт.

В прикладах генераторів аналогового сигналу в даній лабораторній роботі будемо використовувати ЦАП червоного каналу інтерфейсу VGA. Відповідно, цифрові коди необхідно подавати на VGA_R вхід ЦАП, а напругу знімати між виводами 1 та 6 VGA інтерфейсу (див рис.2.34, вивід 1 - вихід червоного каналу VGA, вивід 6 - GND).

2.3.4 Завдання на лабораторну роботу

Вихідний код завдань можна переглянути за посиланням [2.8].

1. В Quartus Prime створіть 10-розрядний синхронний лічильник (описаний на мові Verilog, лістинг 2.2). У якості сигналу тактової частоти використайте вхід FPGA з іменем MAX10_CLK1_50 для налагоджувальної плати DE10-Lite (для плат DE2 і DE0-CV вхід тактової частоти називається CLOCK_50). Тактовий сигнал на зазначених входах має частоту 50 МГц. У якості джерела сигналу скидання використайте одну із кнопок KEY. Вміст розрядів лічильника виведіть на світлодіоди. В логічному аналізаторі SignalTap переконайтесь, що лічильник рахує від 0 до 1023, а частота імпульсів на кожному наступному розряді (починаючи з 0-го розряду) буде вдвічі меншою.
2. Створіть тестбенч для лічильника із попереднього завдання (по прикладу лістинг 2.5). Виконайте симуляцію створеного тестбенча і лічильника в Modelsim. Переконайтесь, що сигнали у вікні часових діаграм Modelsim змінюються так само, як і сигнали переглянуті за допомогою логічного аналізатора у попередньому завданні.
3. В Quartus Prime створіть цифровий генератор пилкоподібної напруги на базі 10-розрядного синхронного лічильника і ЦАП (розд.2.2.8). У якості сигналу тактової частоти використайте вхід FPGA з іменем MAX10_CLK1_50 для налагоджувальної плати DE10-Lite (для плат DE2 і DE0-CV вхід тактової частоти називається CLOCK_50). Тактовий сигнал на зазначених входах має частоту 50 МГц. У якості джерела сигналу скидання в 0 використайте одну з кнопок KEY. У якості цифрового виходу на який будуть видаватися цифрові коди для ЦАП оберіть 4-розрядний вихід червоного каналу VGA з іменем VGA_R. За допомогою осцилографа переконайтеся, що форма сигналу напруги на виході VGA_R така ж сама, як і на рис.2.14.
4. В Quartus Prime створіть цифровий генератор синусоїдального сигналу зі структурою наведеною на рис.2.25 (лістинг 2.4). Входи тактового сигналу синхронізації, скидання і вихід ЦАП підключаються до виводів FPGA так само, як і в попередньому завданні. Комірки пам’яті і ЦАП мають розрядність

4 біта. Період синусоїдального сигналу містить 1024 точки. За допомогою осцилографа переконайтеся, що форма сигналу напруги на виході VGA_R така ж сама, як і на рис.2.29.

5. Створіть тестбенч для цифрового генератора синусоїдального сигналу із попереднього завдання (лістинг 2.5). Виконайте симуляцію створеного тестбенча і генератора в Modelsim. Переконайтесь, що сигнали у вікні часових діаграм Modelsim змінюються так само, як і на рис.2.29.
6. В Quartus Prime створіть цифровий генератор синусоїдального сигналу з керованою частотою і структурою наведеною на рис.2.31 (лістинг 2.6). Входи тактового сигналу синхронізації, зкидання і вихід ЦАП підключаються до виводів FPGA так само, як і в завданні 4. Акумулятор фази `phase` і цифровий код `phase_step`, на який збільшується вміст `phase` по кожному активному фронту сигналу синхронізації повинні мати розрядність 32 біта. Період синусоїдального сигналу містить 1024 точки. Цифровий код `phase_step`, що керує частотою синусоїди формується наступним чином. Молодші 8 біт і старші 14 біт приймають значення 0. Біти з номерами від 8 до 17 зчитуються з 10 розрядного входу перемикачів SW. За допомогою осцилографа переконайтеся, що сигнал напруги на виході VGA_R має форму синусоїди частота якої залежить від цифрового коду на перемикачах SW.

2.4 Контрольні запитання

1. Намалюйте умовне графічне позначення синхронного лічильника, опишіть входи/виходи і поясніть логіку його роботи;
2. Поясніть, що таке модуль рахунку лічильника;
3. Поясніть яким чином лічильник можна використовувати у якості подільника частоти;
4. Намалюйте умовне графічне позначення синхронного по фронту T-тригера без входу дозволу, опишіть входи/виходи і поясніть логіку його роботи;
5. Поясніть який чином синхронний по фронту T-тригер можна використовувати у якості подільника частоти?

6. Намалюйте умовне графічне позначення синхронного по фронту Т-тригера з входом дозволу, опишіть входи/виходи і поясніть логіку його роботи;
7. Поясніть, як створити синхронний по фронту Т-тригер без входу дозволу на базі синхронного по фронту D-тригера;
8. Поясніть, як створити синхронний по фронту Т-тригер з входом дозволу на базі синхронного по фронту D-тригера;
9. Намалюйте часову діаграму перемикання розрядів 4-розрядного синхронного лічильника. Позначте на часовій діаграмі вхідний сигнал синхронізації, імпульси якого рахує лічильник;
10. Намалюйте схему синхронного 4-х розрядного лічильника вгору з послідовним формуванням сигналу EN;
11. Намалюйте схему синхронного 4-х розрядного лічильника вгору з паралельним формуванням сигналу EN;
12. Намалюйте схему синхронного 4-х розрядного лічильника вниз;
13. Намалюйте схеми 4-х розрядного реверсивного лічильника (рахує вгору, або вниз в залежності від значення входу керування);
14. Напишіть код на мові Verilog, що описує 4-розрядний синхронний лічильник вгору;
15. Напишіть код на мові Verilog, що описує 4-розрядний синхронний лічильник вгору із завантаженням;
16. Намалюйте схему ЦАП на резисторах;
17. Виведіть залежність напруги на виході від цифрового коду на вході ЦАП на резисторах;
18. Намалюйте умовне графічне позначення постійної пам’яті, опишіть входи/виходи і поясніть логіку її роботи;
19. Поясніть якою формулою пов'язана розрядність шини адреси постійної пам’яті і максимальна кількість комірок, яку можна адресувати такою шиною адреси;
20. Поясніть, шо таке затримка зчитування з постійної пам’яті;
21. Поясніть, як описати масив на мові Verilog;
22. Напишіть вихідний код для опису постійної пам’яті на мові Verilog;

23. Поясніть з якою метою використовуються системні функції $reamemh(), $reamemb() і що означають їх аргументи;
24. Поясніть, як задати вміст постійної пам’яті, реалізованої всередині FPGA;
25. Поясніть принцип роботи цифрового генератора синусоїдального сигналу;
26. Намалюйте структурну схему цифрового генератора синусоїдального сигналу;
27. Поясніть, як розрахувати значення цифрових кодів точок синусоїди для цифрового генератора синусоїдального сигналу;
28. Поясніть, чому буде дорівнювати період синусоїди для цифрового генератора синусоїдального сигналу, якщо частота сигналу синхронізації схеми складає 50 МГц, а період синусоїди містить 1024 точки;
29. Поясніть, чим initial блок відрізняється від always блока у мові Verilog;
30. Поясніть, як створити екземпляр модуля у мові Verilog і підключити сигнали до його вхідних і вихідних портів;
31. Поясніть, як можна формувати затримки в initial блоках на мові Verilog;
32. Поясніть з якою метою використовують системну функцію $finish;
33. Поясніть, як запустити симуляцію в Modelsim;
34. Намалюйте структурну схему цифрового генератора синусоїдального сигналу з керованою частотою;
35. Поясніть принцип роботи цифрового генератора синусоїдального сигналу з керованою частотою;
36. Поясніть, як розрахувати значення цифрового коду, на яке збільшується аккумулятор фази в цифровому генераторі синусоїдального сигналу з керованою частотою для заданої частоти синусоїди;
37. Поясніть, чому дорівнює мінімальний крок зміни частоти в цифровому генераторі синусоїдального сигналу з керованою частотою;

2.5 Перелік посилань

[2.1] “Tutorial: Digital to Analog Conversion – The R-2R DAC”, 2015. [Online]. Available: https://uk.tek.com/blog/tutorial-digital-analog-conversion-–-r-2r-dac

[2.2] “Файл ініціалізації постійної пам’яті для цифрового генератора синусоїдального сигналу”, 2018. [Online]. Available: https://github.com/KorotkiyEugene/digital_lab/blob/master/Lab2/hw/simple_sin_gen/sin_table_4bit.hex

[2.3] “Сервіс генерації точок синусоїди для файлу ініціалізації постійної пам’яті цифрового генератора синусоїдального сигналу”, 2018. [Online]. Available: https://daycounter.com/Calculators/Sine-Generator-Calculator.phtml

[2.4] “Вихідний код програми, що розраховує значення для файлу ініціалізації постійної пам’яті цифрового генератора синусоїдального сигналу”, 2018. [Online]. Available: https://github.com/KorotkiyEugene/digital_lab/blob/master/Lab2/hw/simple_sin_gen/sin_table_gen.c

[2.5] “Тестбенч і скрипт симуляції в Modelsim для цифрового генератора синусоїдального сигналу”, 2018. [Online]. Available: https://github.com/KorotkiyEugene/digital_lab/tree/master/Lab2/hw/simple_sin_gen

[2.6] “Fundamentals of Direct Digital Synthesis (DDS)”, 2008. [Online]. Available: http://www.analog.com/media/en/training-seminars/tutorials/MT-085.pdf

[2.7] “Simple Numerically Controlled Oscillator on FPGA DE10-Lite Board”, 2018. [Online]. Available: https://www.youtube.com/watch?v=3bTapxSuojs

[2.8] Вихідні коди прикладів до лабораторних робіт, 2018 [Online]. Режим доступу: https://github.com/KorotkiyEugene/digital_lab