Лабораторна робота №1

Синхронні по фронту елементи пам’яті

1.1 Практичне застосування досліджуваних схем

Синхронні по фронту D-тригери присутні у будь яких відносно складних цифрових мікросхемах. Ці компоненти зберігають проміжні значення цифрових сигналів, забезпечуючи боротьбу з паразитними імпульсами, що виникають під час гонок в комбінаційній логіці. На базі синхронних по фронту D-тригерів будують паралельні регістри та регістри зсуву. У паралельних регістрах зберігають проміжні значення обчислень (наприклад, регістри мікропроцесору). Регістри зсуву застосовують для передачі даних у послідовній формі (наприклад, інтерфейс SPI) та для побудови високочастотних лічильників в унітарному коді (one-hot counters). Регістр зсуву з лінійним зворотнім зв'язком (LFSR) використовують для генерації псевдовипадкових послідовностей чисел. LFSR застосовують у криптографії (потокові шифри), для розрахунку циклічних контрольних сум CRC з метою виявлення помилок в прийнятих даних, для генерації тестових послідовностей під час тестування і самотестування цифрових мікросхем (built-in self-test, BIST), і в скремблерах телекоммунікаційних систем.

1.2 Теоретична частина

1.2.1 Синхронний по фронту D-тригер

Синхронний по фронту D-тригер є елементом пам'яті, що зберігає 1 біт даних. Умовне графічне позначення такого тригера наведено на рис.1.1.

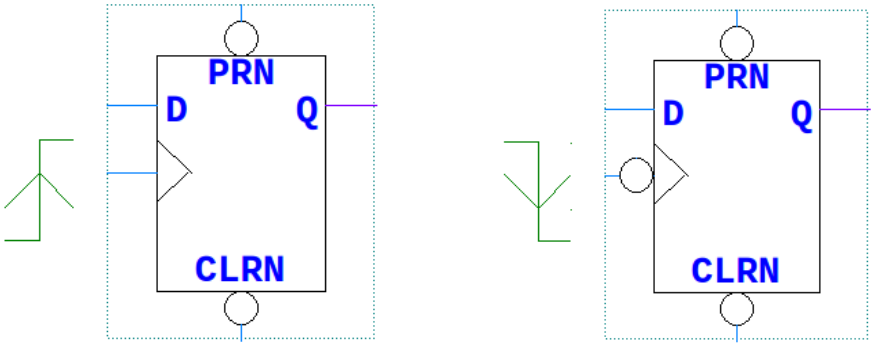


Рис.1.1 - Умовне графічне позначення D-тригерів синхронних по передньому фронту (зліва) та по задньому фронту (зправа)

Синхронний по фронту D-тригер має вхід даних D та вихід даних Q. В деяких тригерах також присутній інверсний вихід даних $\overline{Q}$ значення якого інверсне (протилежне) до значення виходу Q (якщо Q=0, то $\overline{Q}$=1 і навпаки). Кожен такий тригер має вхід синхронізації (його ще іноді називають тактовий вхід), який позначено трикутником для тригера синхронного по передньому фронту. Для тригера синхронного по задньому фронту зліва біля трикутника малюють круг. Часто вхід синхронізації називається CLK. Деякі синхронні по фронту тригери мають входи асинхронного зкидання CLRN та асинхронного встановлення PRN.

Логіка роботи синхронного по фронту D-тригера. Після подачі напруги живлення D-тригер встановлюється у випадкове значення (0 або 1). Якщо подати на вхід даних D нове значення (0 або 1), а після цього на вхід синхронізації подати активний фронт (передній фронт для тригера синхронного по передньому фронту і задній фронт для тригера синхронного по задньому фронту), це нове значення з входу D запишеться в тригер і з'явиться на його виході Q.

Якщо є необхідність записати в тригер 0 в будь який момент часу, а не лише в момент активного фронту, можна скористатися асинхронним входом зкидання CLRN, подавши на нього активний рівень. У більшості сучасних тригерів активний логічний рівень для операцій асинхронного встановлення і зкидання -- логічний 0. Як тільки на вхід CLRN подається активний логічний рівень, тригер зкидається в 0 і на виході Q з'являється 0. Аналогічно після подачі активного логічного рівня на вхід асинхронного встановлення PRN, тригер встановлюється в 1 і на його виході Q з'являється 1.

Асинхронні входи встановлення і зкидання бувають корисні для приведення станів тригерів у задані значення після подачі напруги живлення. Як зазначено вище, після подачі напруги живлення значення тригерів встановлюються випадковим чином. Для приведення станів тригерів до заданих значень асинхронні входи встановлення (для тригерів які після подачі живлення повинні містити 1) та зкидання (для тригерів які після подачі живлення повинні містити 0) підключаються до глобального входу RESET. Одразу після подачі напруги живлення на глобальному вході RESET формується активний рівень, що приводить до встановлення станів тригерів у задані значення.

Важливо розуміти, що запис нового значення в D-тригер, його асинхронне встановлення та зкидання відбуваються не миттєво, а з певною затримкою. Наприклад, затримка запису tcq (time from clock to q) показана на рис.1.2.

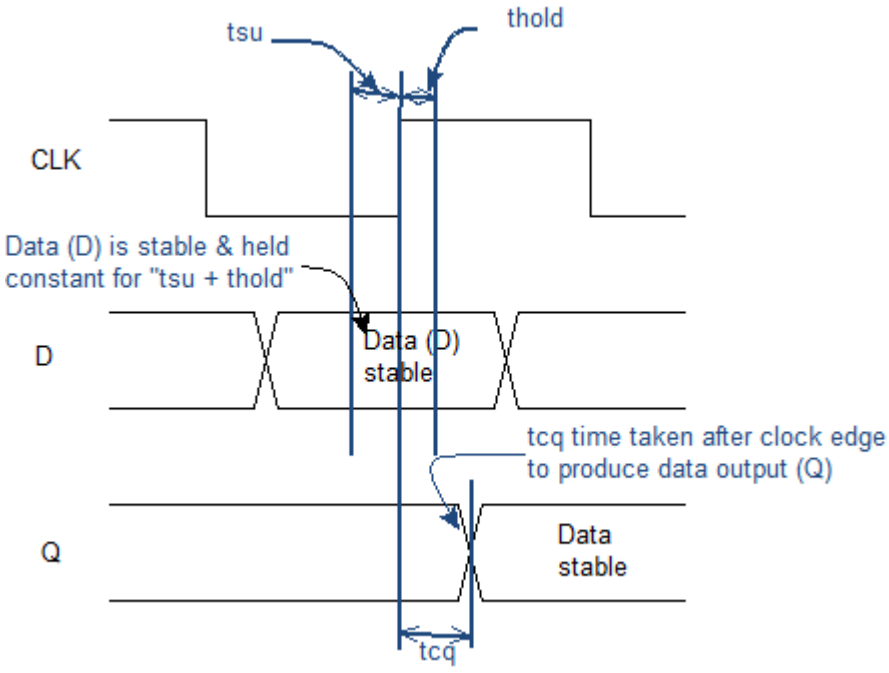


Рис.1.2 - Таймінги (затримки) D-тригера синхронного по фронту

У D-тригера синхронного по фронту є дуже важливе обмеження. Значення на вході D не повинно змінюватись одночасно з активним фронтом CLK. Якщо дані на вході D будуть змінюватись під час активного фронту CLK, тригер може перейти в метастабільний стан. Коли D-тригер знаходиться в метастабільному стані на його виході Q присутня напруга, що дорівнює половині напруги живлення (Vdd/2). Це не лог. 0 і не лог. 1, а напруга із забороненого діапазону значень. Тригер знаходиться у метастабільному стані невеликий випадковий проміжок часу, а потім виходить в стан лог.0, або в стан лог.1 випадковим чином. Входження тригерів в метастабільний стан призводить до помилок в цифрових мікросхемах і їх нестабільної роботи.

Щоб тригер не входив у метастабільний стан, дані на вході D не повинні змінюватись протягом проміжку часу tsu (time setup) перед приходом активного фронту та протягом проміжку часу thold (time hold) після приходу активного фронту (див. рис.1.9). Проміжки часу tsu та thold називають таймінгами D-тригера. Таймінги тригерів наведені в документації на цифрову мікросхему. Ми повернемося до таймінгів

D-тригерів, коли будемо розглядати питання максимально можливої частоти сигналу синхронізації цифрових схем.

Розглянутий вище D-тригер записує дані з входу D після кожного активного фронту на вході синхронізації. Однак в сучасних цифрових системах сигнал синхронізації - це періодична послідовність імпульсів з високою частотою. В такому випадку активний фронт буде приходити на вхід синхронізації регулярно і дуже часто. В тригер постійно будуть записуватися нові значення з входу D. Що робити, якщо хочеться використовувати D-тригер як елемент пам'яті і зберігати в ньому значення не лише протягом періоду сигналу синхронізації, а довільний проміжок часу? Тут стане у нагоді синхронний по фронту D-тригер з входом дозволу запису (рис.1.3).

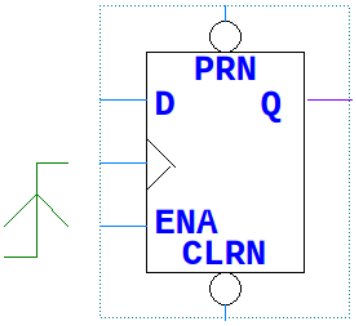


Рис.1.3 - Умовне графічне позначення синхронного по передньому фронту D-тригера з входом дозволу запису ENA

Запис в подібний тригер нового значення з входу D відбувається по активному фронту і лише, якщо на вході дозволу запису ENA присутній активний логічний рівень (зазвичай це лог.1). Якщо ж під час активного фронту на вході дозволу запису ENA присутній не активний логічний рівень (зазвичай це лог.0), нове значення в тригер не записується і тригер зберігає попереднє значення. Часто вхід ENA ще називають WE (Write Enable).

1.2.2 Опис схем синхронних по фронту на мові Verilog

Згадаємо, що процес створення цифрової схеми з опису на Verilog називається **синтезом**. Зазвичай синтез - шаблонізована процедура. Для різних компонентів цифрових схем існують шаблони їх опису на Verilog і необхідно дотримуватись цих

шаблонів для правильного синтезу. Коли САПР зустрічає в коді Verilog певний шаблон коду, цей шаблон замінюється на відповідну цифрову схему (наприклад, оператор + заміняються на суматор і т.д.). Звісно, це спрощений опис синтезу і в реальності є нюанси. Однак в загальному вигляді процедура синтезу приблизно так і відбувається.

В цьому розділі розглянемо шаблони коду для опису на Verilog схем синхронних по фронту. Прикладами подібних схем є D-тригери, паралельні регістри, регістри зсуву.

Шаблон коду для опису D-тригера синхронного по передньому фронту без асинхронних входів і дозволу запису наведено в лістингу 1.1.

Як видно з лістингу для опису D-тригера застосовано процедурний **always** блок.

Лістинг 1.1 - Шаблон опису D-тригера синхронного по передньому фронту без асинхронних входів і дозволу запису

```
module flip_flop(i_clk, i_data, o_data);

input      i_clk;
input      i_data;
output reg o_data;

always @(posedge i_clk) begin
    o_data <= i_data;
end

endmodule
```

Процедурний **always** блок - важлива конструкція мови Verilog, тому розглянемо його детальніше. По суті, **always** блок є нескінченним циклом. Його вміст ітеративно виконується раз за разом. Ось приклад такого нескінченного циклу:

Лістинг 1.2 - Приклад нескінченного циклу на базі **always** блоку (поганий приклад)

```
reg [7:0] i;
always i = i + 1;
```

Після ключого слова **always** повинна міститися або одна інструкція, як в прикладі зверху, або блок коду обмежений ключовими словами **begin** / **end**. Між ключовими словами **begin** / **end** може міститися довільна кількість інструкцій. Інструкції блокуючого присвоювання (оператор =) в середині блоку коду обмеженого **begin** / **end** виконуються послідовно одна за одною.

Дуже важливий момент! В операціях присвоювання всередині **always** блоку, в лівій частині оператора присвоювання повинні бути лише змінні типу **reg**.

В лістингу 1.3 дві операції блокуючого присвоювання всередині блоку виконуються раз за разом у нескінченному циклі **always** блоку.

Однак подібне безперервне виконання коду всередині **always** блоку є грубою помилкою, яка призводить до зависання симулятора під час моделювання.

Лістинг 1.3 - Приклад нескінченного ітеративного виконання блоку коду з двох операцій блокуючого присвоювання

```
input      [7:0] i_op1, i_op2;
output reg [7:0] o_sum, o_sub;

always begin
    o_sum = i_op1 + i_op2;
    o_sub  = i_op1 - i_op2;
end
```

Правильна робота **always** блоку полягає у тому, щоб очікувати на певну подію (наприклад, зміну одного з аргументів, або на передній фронт сигналу синхронізації), після настання події виконати блок коду і знову перейти в режим очікування на подію.

Очікування на подію в мові Verilog забезпечується оператором **@**(...). Оператор **@** блокує виконання наступного за ним коду до виконання події/подій, що перераховані всередині круглих скобок. Якщо є кілька подій на які необхідно очікувати, їх перераховують через кому і виникнення будь якої з цих подій приводить до продовження виконання коду, що слідує після оператора **@**. Подією може бути зміна сигналу, передній, або задній фронт сигналу. Якщо необхідно очікувати на будь яку зміну сигналу (зміну рівня), тоді в круглих скобках пишуть просто ім’я сигналу. Якщо необхідно очікувати на передній фронт сигналу, перед ім’ям сигналу пишуть ключове слово **posedge**. Якщо необхідно очікувати на задній фронт сигналу, перед ім’ям сигналу пишуть ключове слово **negedge**. Сигнули у круглих скобках оператору **@** називають **списком чутливості**.

Увага! Очікування зміни рівня і зміни фронту в одному операторі **@** приводить до одержання коду, який моделюється, але не синтезується. Зазвичай при описі комбінаційної логіки (наприклад, суматора, чи перемножувача) за допомогою **always** блоку, в операторі **@** очікують на зміну рівня будь-якого з вхідних аргументів. При

описі синхронних по фронту схем в операторі **@** очікують на активний фронт сигналу синхронізації або на фронт сигналу асинхронного зкидання/встановлення, що призводить до активного рівня входів зкидання/встановлення (далі будуть наведені приклади).

Повернемося до лістингу 1.1. Як видно, в **always** блоці за допомогою оператора **@** відбувається очікування переднього фронту сигналу синхронізації i_clk і одразу після переднього фронту цього сигналу (грубо кажучи, в той же момент часу, коли було зафіксовано передній фронт), відбувається запис значення входу i_data у змінну o_data, що має тип **reg** і є виходом модуля. Так відбувається моделювання зазначеного блоку коду в симуляторі.

Логіка синтезу коду з лістингу 1.1 наступна. При аналізі цього коду САПР Quartus Prime бачить, що після активного фронту сигналу синхронізації відбувається запис в змінну типу **reg**. Це відповідає логіці роботи і шаблону опису синхронного по фронту D-тригера, тож з такого коду синтезується синхронний по фронту D-тригер.

Дуже важливе уточнення! При описі синхронних по фронту схем обов’язково необхідно використовувати неблокуюче присвоювання (оператор <=). Використання блокуючого присвоювання (оператор =) часто призводить до помилок під час симуляції та іноді призводить до неправильного синтезу.

Блокуюче ж присвоювання (оператор =) використовують для опису комбінаційної логіки з використанням **always** блоку. Це питання ми розглянемо в одній з наступних лабораторних робіт.

Тепер розглянемо шаблон коду для опису D-тригера синхронного по передньому фронту з входом асинхронного зкидання по низькому рівню, без входу дозволу запису (лістинг 1.4).

В даному випадку, вхід зкидання асинхронний, а значить зкидання повинно відбуватися без прив’язки до сигналу синхронізації. Тож необхідно додати у список чутливості оператора **@** також сигнал i_arst_n. Раз зкидання повинне відбуватися по низькому рівню i_arst_n, будемо очікувати на задній фронт цього сигналу. Всередині ж **always** блоку будемо перевіряти стан i_arst_n. Якщо ~i_arst_n (не i_arst_n) приймає значення 1, значить сигнал i_arst_n дорівнює нулю, вхід зкидання активний, тому записуємо 0 в змінну, що зберігає стан тригеру. Якщо ж i_arst_n приймає значення 1, значить **always** блок спрацював по передньому фронту сигналу синхронізації і раз вхід

зкидання не активний, необхідно записати в змінну, що зберігає стан тригеру нове значення з входу даних.

Лістинг 1.4 - Шаблон опису D-тригера синхронного по передньому фронту з входом асинхронного зкидання по низькому рівню, без входу дозволу запису

```
module flip_flop(i_clk, i_arst_n, i_data, o_data);

input      i_clk;
input      i_arst_n;
input      i_data;
output reg o_data;

always @(posedge i_clk, negedge i_arst_n) begin
    if (~i_arst_n) begin
        o_data <= 1'b0;
    end else begin
        o_data <= i_data;
    end
end

endmodule
```

Наостанок, розглянемо шаблон коду для опису D-тригера синхронного по передньому фронту з входом асинхронного зкидання по низькому рівню і входом дозволу запису по високому рівню (лістинг 1.5).

Лістинг 1.5 - Шаблон опису D-тригера синхронного по передньому фронту з входом асинхронного зкидання по низькому рівню і входом дозволу запису

```
module flip_flop(i_clk, i_arst_n, i_we, i_data, o_data);

input      i_clk;
input      i_arst_n;
input      i_we;
input      i_data;
output reg o_data;

always @(posedge i_clk, negedge i_arst_n) begin
    if (~i_arst_n) begin
        o_data <= 1'b0;
    end else begin
        if (i_we)
            o_data <= i_data;
    end
end

endmodule
```

Вхід дозволу запису i_we в даному випадку синхронний, тож в список чутливості оператору **@** ми його не додаємо. Якщо ж відбувся передній фронт, сигнал i_arst_n не активний, а сигнал i_we дорівнює 1, в змінну, що зберігає стан тригеру записуємо нове значення з входу.

1.2.11 Паралельний регістр

Синхронний по фронту паралельний регістр дуже схожий на синхронний по фронту D-тригер. З тією різницею, що тригер зберігає 1 біт даних, а паралельний регістр зберігає багаторозрядне число. Відповідно, паралельні регістри використовують для збереження багаторозрядних даних (наприклад, регістри мікропроцесору).

Як видно з рис.1.4 синхронний по фронту паралельний регістр складається з кількох синхронних по фронту D-тригерів, у яких входи синхронізації з'єднані між собою і підключені до спільного джерела сигналу синхронізації.

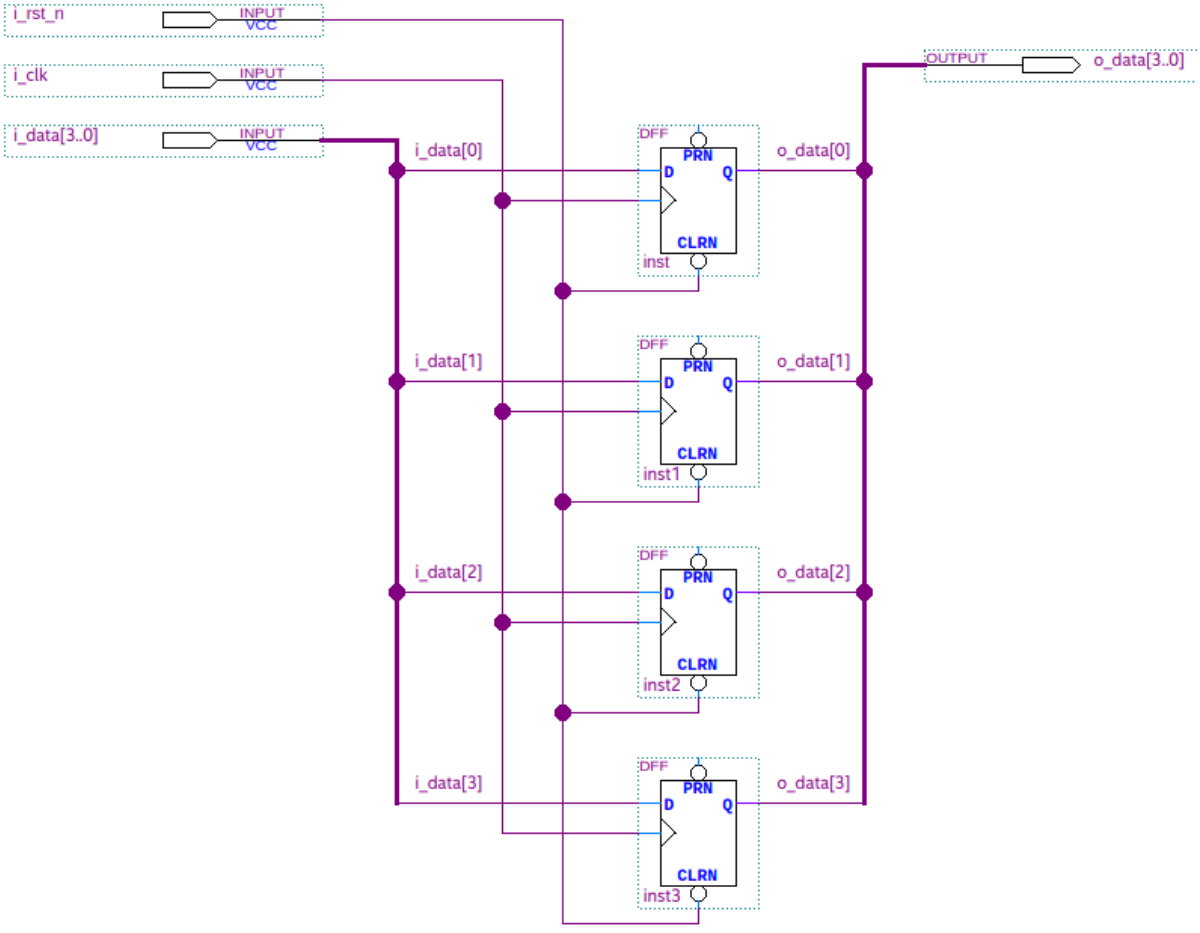


Рис.1.4 - Схема паралельного регістру синхронного по передньому фронту

Входи асинхронного зкидання D-тригерів, що утворюють паралельний регістр, теж з'єднанні між собою і підключені до спільного входу асинхронного зкидання регістру. Аналогічно можна реалізувати вхід асинхронного встановлення розрядів регістру в одиниці, або взагалі в будь які довільні значення. Наприклад, якщо ми хочемо, щоб по сигналу асинхронного ресету 4-х розрядний регістр встановлювався у значення b1100 (в двійковій системі звісно), необхідно підключити вхід асинхронного ресету регістру до входів асинхронного зкидання двох молодших розрядів та до входів асинхронного встановлення двох старших розрядів.

Логіка роботи синхронного по фронту паралельного регістру така ж сама, як і у синхронного по фронту D-тригера. У момент активного фронту дані з багаторозрядного входу даних i_data[3..0] записуються в регістр і з невеликою затримкою з'являються на багаторозрядному виході даних o_data[3..0] (у тригерів, що входять до складу FPGA мікросхем Intel, активний фронт D-тригерів передній).

Не забувайте, що для D-тригерів, що утворюють паралельний регістр, обов'язково повинні витримуватись затримки t_{setup} і t_{hold} про які йшла мова в розділі 1.2.1.

В лістингу 1.6 наведений опис паралельного регістру синхронного по передньому фронту на мові Verilog. Для зміни розрядності такого регістру достатньо змінити розрядність i_data, o_data та константи 4’b0000. Розрядність регістру можна зробити конфігурованою за допомогою механізму параметрів Verilog, який ми вивчимо пізніше.

Лістинг 1.6 - Опис паралельного регістру синхронного по передньому фронту на мові Verilog

```
module paral_reg(i_clk, i_rst_n, i_data, o_data);

input              i_clk;
input              i_rst_n;
input       [3:0]  i_data;
output reg  [3:0]  o_data;

always @(posedge i_clk, negedge i_rst_n) begin
    if(~i_rst_n) begin
        o_data <= 4'b0000; // you may use decimal consts, e.g. 4'd0
    end else begin
        o_data <= i_data;
    end
end

endmodule
```

Результат синтезу Verilog коду з лістингу 1.6 в RTL Viewer наведено на рис.1.5.

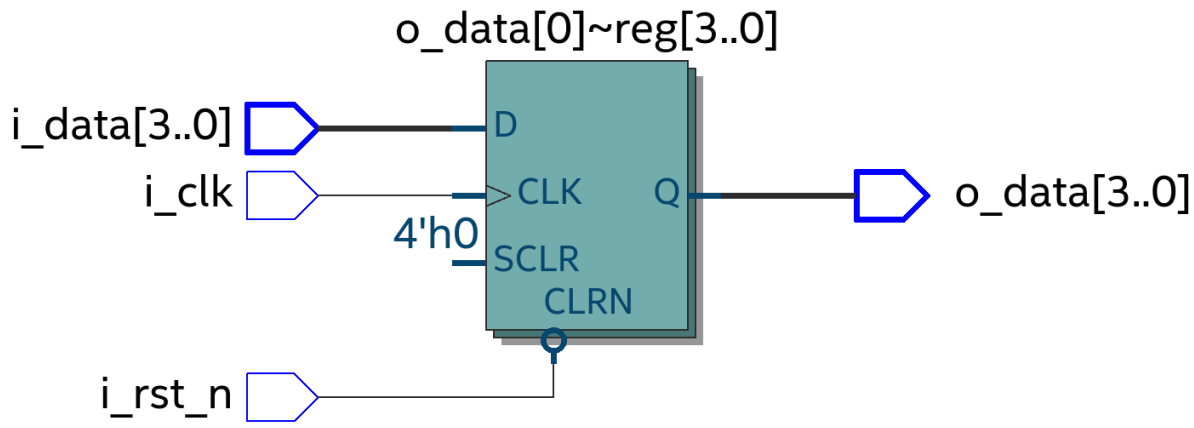


Рис.1.5 - Результат синтезу Verilog коду з лістингу 1.6 в RTL Viewer

На рис.1.4.1 показано паралельний регістр без входу дозволу запису. Але такий вхід не складно додати, якщо будувати регістр на базі синхронних по фронту D-тригерів з входом дозволу запису. Входи дозволу запису тригерів з'єднуються і підключаються до входу дозволу запису регістру. Відповідно, дані будуть записуватись в такий регістр лише за умови, що під час надходження активного фронту на вхід синхронізації, на вході дозволу запису буде присутній активний логічний рівень (у тригерів, що входять до складу FPGA мікросхем Intel, високий активний рівень входів дозволу запису).

1.2.12 Регістр зсуву, оператор конкатенації мови Verilog

Регістри зсуву використовують для послідовної передачі багаторозрядних даних (біт за бітом). Яскравий приклад - інтерфейс SPI (розд. 1.2.18). На базі регістру зсуву зсуву можна побудувати генератор псевдовипадкової послідовності (розд. 1.2.14). Регістри зсуву використовують для побудови високочастотних one-hot лічильників.

Схему 4-розрядного регістру зсуву наведено на рис.1.6. Стан бітів регістрів виведено на шину o_data[3..0]. В даному прикладі досліджуємо 4-розрядний регістр, але загалом розрядність може бути довільна. Регістр має однорозрядний вхід i_data, вхід синхронізації i_clk на який поступають періодині імпульси сигналу синхронізації та вхід i_rst_n для асинхронного зкидання розрядів регістру в нулі.

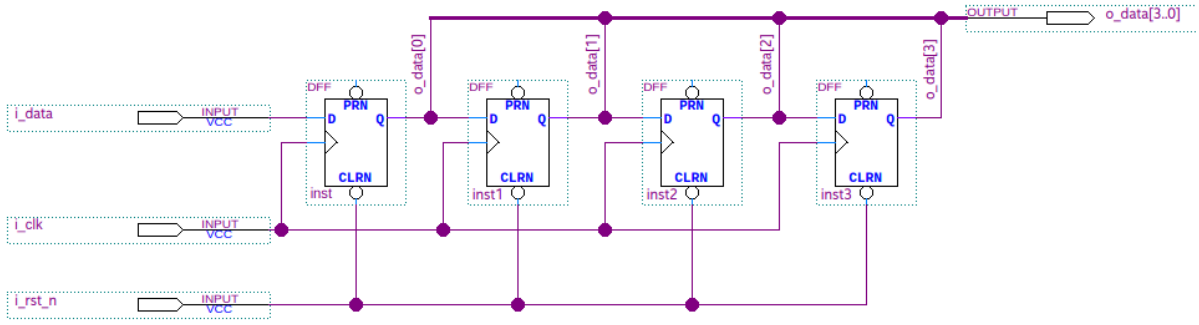


Рис.1.6 - Схема регістру зсуву

Логіка роботи регістру зсуву наступна. По кожному активному (передньому) фронту на вході синхронізації i_clk значення з виходу кожного попереднього D-тригеру переписується в наступний D-тригер, а у наймолодший D-тригер o_data[0] записуються дані з входу i_data.

Зміна сигналів схеми з рис.1.6 показана на рис.1.7.

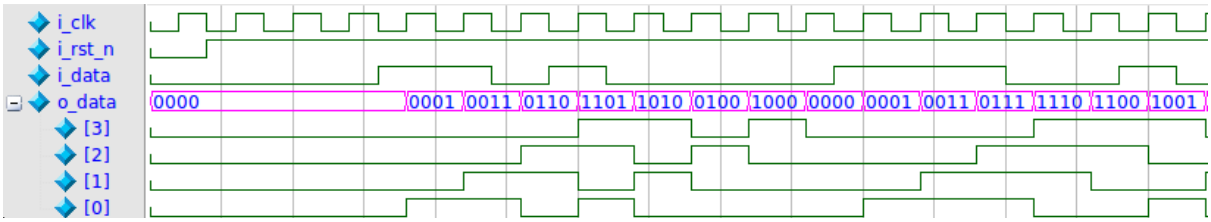


Рис.1.7 - Часові діаграми регістру зсуву

Опис регістру зсуву на мові Verilog наведено в лістингу 1.7. Зверніть увагу, що єдина глобальна відмінність від паралельного регістру з лістингу 1.6 полягає у строчці Verilog коду, що визначає запис нових даних в змінну `shift_reg` типу **`reg`**:

```
    shift_reg <= {shift_reg[2:0], i_data};
```

В даному випадку використовуються оператор конкатенації **{ }** мови Verilog. За допомогою оператора конкатенації можна створити нову шину сигналів з існуючих шин і однорозрядних сигналів типів **`wire`** і **`reg`**. Сигнали, що об'єднуються перераховуються через кому. При цьому крайній правий сигнал представляє молодший біт шини, що створюється. А крайній лівий сигнал представляє старший біт шини, що створюється.

Отже зазначена вище строчка Verilog коду створює нову шину, молодшим бітом якої є сигнал `i_data`, а далі йдуть три біти `shift_reg[2:0]`. По активному фронту сигналу синхронізації, за відсутності асинхронного зкидання, створена шина записується у змінну `shift_reg`. Тобто запис відбувається наступним чином (справа від знаку `<--` показані старі значення `shift_reg`):

```
shift_reg[0] <-- i_data
shift_reg[1] <-- shift_reg[0]
shift_reg[2] <-- shift_reg[1]
shift_reg[3] <-- shift_reg[2]
```

Така логіка роботи, коли по активному фронту сигналу синхронізації значення попереднього розряду переписується в наступний розряд, відповідає логіці роботи регістру зсуву. Тому САПР Quartus Prime і синтезує регістр зсуву з шаблону коду, що наведений в лістингу 1.7.

Лістинг 1.7 - Опис регістру зсуву на мові Verilog

```
module shift_reg (i_clk, i_rst, i_data, o_data);

input           i_clk;
input           i_rst_n;
input           i_data;
output  [3:0]   o_data;

reg     [3:0]   shift_reg;

assign o_data = shift_reg;

always @(posedge i_clk, negedge i_rst_n) begin
    if (~i_rst_n) begin
        shift_reg <= 4'd0;
    end else begin
        shift_reg <= {shift_reg[2:0], i_data};
    end
end

endmodule
```

1.2.13 Регістр зсуву з входом дозволу зсуву

В регістрі зсуву з попереднього розділу зсув відбувається після кожного активного фронту сигналу синхронізації. А в цифрових мікросхемах сигнал синхронізації представляє собою періодичні імпульси з високою частотою. Щоб мати можливість робити зсув лише в задані моменти часу знадобиться регістр з входом дозволу зсуву схема якого зображена на рис.1.8.

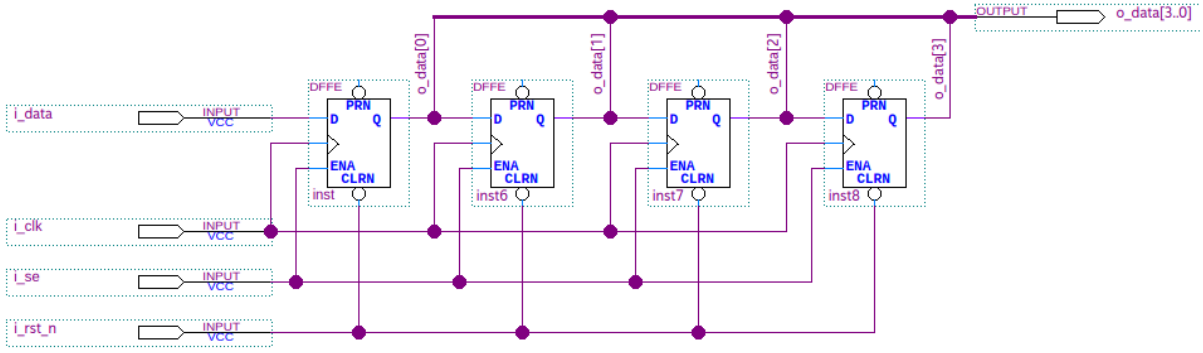


Рис.1.8 - Схема регістру зсуву з входом дозволу зсуву

Схема на рис.1.8 побудована на базі синхронних по фронту D-тригерів з входом дозволу запису. Входи дозволу запису з’єднані між собою і підключені до входу дозволу зсуву регістру i_se (Shift Enable). Зсув відбувається лише якщо в момент приходу активного фронту сигналу синхронізації на вході дозволу зсуву i_se присутній високий рівень (згадаємо, що у тригерів середині мікросхем Intel FPGA активний високий рівень дозволу запису).

Зміна сигналів у схемі з рис.1.8 наведена на рис.1.9.

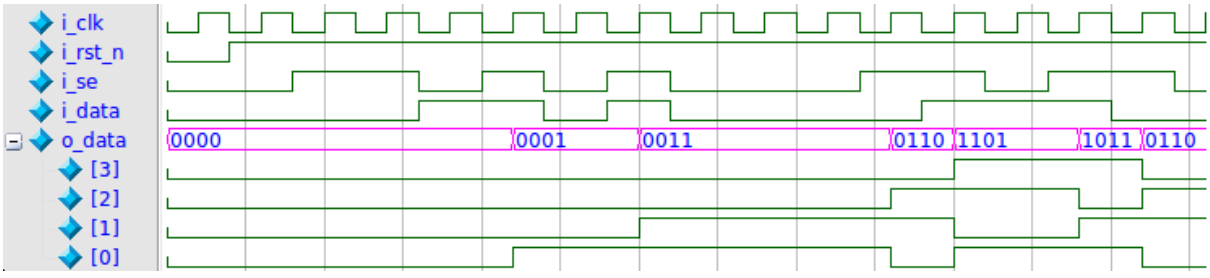


Рис.1.9 - Часові діаграми регістру зсуву з входом дозволу зсуву

Опис регістру зсуву з входом дозволу зсуву на мові Verilog наведений в ліст. 1.8.

Лістинг 1.8 - Опис регістру зсуву з входом дозволу зсуву на мові Verilog

```
module shift_reg_we (i_clk, i_rst_n, i_se, i_data, o_data);

input           i_clk;
input           i_se;
input           i_rst_n;
input           i_data;
output  [3:0]   o_data;

reg     [3:0]   shift_reg;

assign o_data = shift_reg;

always @(posedge i_clk, negedge i_rst_n) begin
    if (~i_rst_n) begin
        shift_reg <= 4'd0;
    end else begin
        if(i_se)
            shift_reg <= {shift_reg[2:0], i_data};
    end
end

endmodule
```

1.2.14 Регістр зсуву з лінійним зворотнім зв’язком (LFSR)

На базі регістру зсуву можна побудувати схему для генерації псевдовипадкових послідовностей бітів. Послідовність бітів називається псевдовипадковою, оскільки з точки зору статистичних тестів на випадковість значення бітів, що створюються, поводять себе, як випадкова величина з рівномірно випадковим законом розподілення.

Знаючи схему генератора можна спрогнозувати значення кожного наступного біту, що створюється. Після зкидання стану такого генератора в 0 послідовність починається спочатку.

Схема генерації псевдовипадкових послідовностей бітів на базі регістру зсуву називається LFSR (Linear Feedback Shift Register).

Схема 4-розрядного LFSR наведена на рис.1.10. Можна створити LFSR генератор з довільною розрядністю.

LFSR генератор створює 2^N - 1 випадкових N-розрядних числа, де N - розрядність LFSR.

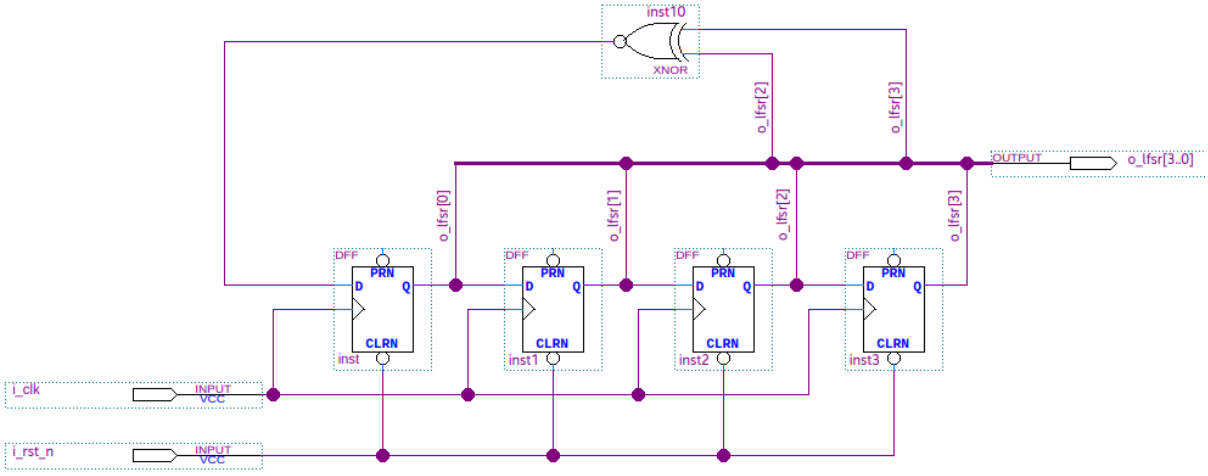


Рис.1.10 - Схема 4-розрядного LFSR генератора псевдовипадкової послідовності бітів

Як видно з рис.1.10 схема LFSR генератора побудована на базі регістр зсуву. Значення кількох розрядів регістру зсуву подаються на входи логічного елементу XNOR, а значення з виходу цього логічного елементу є псевдовипадковим бітом, що записується в молодший розряд регістру LFSR після кожного наступного активного фронту сигналу синхронізації.

Зміна сигналів для LFSR генератора з рис.1.10 наведено на рис.1.11.

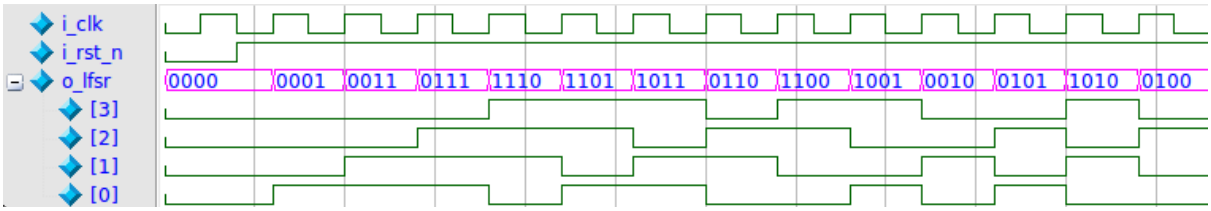


Рис.1.11 - Часові діаграми 4-розрядного LFSR генератора псевдовипадкової послідовності бітів

У разі використання логічного елементу XNOR стан регістру, коли всі його розряди приймають значення лог.1. є забороненим, оскільки в такому випадку на виході XNOR завжди буде лог. 1 і LFSR зациклиться (зависне) в цьому стані.

За необхідності замість лог. елементу XNOR можна використовувати лог. елемент XOR, однак в цьому випадку забороненим станом для LFSR будуть нулі у всіх розрядах.

Дуже важливо правильно вибрати значення розрядів регістру LFSR, які подаються на входи логічного елементу XNOR, щоб отримати максимальну довжину псевдовипадкової послідовності рівну 2^N - 1. Для різних розрядностей LFSR генератора значення розрядів зворотнього зв'язку можна взяти з рис.1.12, або із документу за посиланням [1.1]. Зверніть увагу, що на рис.1.12 та в таблиці за посиланням [1.1] розряди LFSR нумеруються від 1 до N, де 1 молодший розряд, а N - старший розряд. У мові ж Verilog зазвичай нумерують розряди від 0 до N-1, де 0 молодший розряд, а (N - 1) старший розряд.

В лістингу 1.9 наведено опис 4-розрядного LFSR генератора псевдовипадкової послідовності бітів на мові Verilog.

Лістинг 1.9 - Опис 4-розрядного LFSR генератора псевдовипадкової послідовності бітів на мові Verilog

```
module lfsr4bit(i_clk, i_rst_n, o_lfsr);

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

reg     [3:0]   lfsr;

assign  o_lfsr = lfsr;

wire    lfsr_lsb = ~(lfsr[3] ^ lfsr[2]);

always @(posedge i_clk, negedge i_rst_n) begin
    if (~i_rst_n) begin
        lfsr <= 4'd0;
    end else begin
        lfsr <= {lfsr[2:0], lfsr_lsb};
    end
end

endmodule
```

Регістр зсуву з лінійним зворотнім зв’язком (LFSR) застосовують у криптографії (потокові шифри), для розрахунку циклічних контрольних сум CRC з метою виявлення помилок в прийнятих даних, для генерації тестових послідовностей під час тестування / самотестування цифрових мікросхем (built-in self-test, BIST), і в скремблерах телекоммунікаційних систем.

Linear Feedback Shift Register Taps

Table 1 lists the appropriate taps for maximum-length LFSR counters of up to 168 bits. The outputs are designated as 1 through *n* with 1 as the first stage.

Table 1: **Taps for Maximum-Length LFSR Counters**

n	XNOR from	*n*	XNOR from	*n*	XNOR from	*n*	XNOR from
3	3,2	45	45,44,42,41	87	87,74	129	129,124
4	4,3	46	46,45,26,25	88	88,87,17,16	130	130,127
5	5,3	47	47,42	89	89,51	131	131,130,84,83
6	6,5	48	48,47,21,20	90	90,89,72,71	132	132,103
7	7,6	49	49,40	91	91,90,8,7	133	133,132,82,81
8	8,6,5,4	50	50,49,24,23	92	92,91,80,79	134	134,77
9	9,5	51	51,50,36,35	93	93,91	135	135,124
10	10,7	52	52,49	94	94,73	136	136,135,11,10
11	11,9	53	53,52,38,37	95	95,84	137	137,116
12	12,6,4,1	54	54,53,18,17	96	96,94,49,47	138	138,137,131,130
13	13,4,3,1	55	55,31	97	97,91	139	139,136,134,131
14	14,5,3,1	56	56,55,35,34	98	98,87	140	140,111
15	15,14	57	57,50	99	99,97,54,52	141	141,140,110,109
16	16,15,13,4	58	58,39	100	100,63	142	142,121
17	17,14	59	59,58,38,37	101	101,100,95,94	143	143,142,123,122

Рис.1.12 - Номери розрядів зворотнього зв'язку для різних розрядностей LFSR генератора псевдовипадкової послідовності бітів

1.2.15 Синхронізація даних з входів FPGA

Як було зазначено в розділі 1.2.1, сигнал на вході даних D-тригера не повинен змінюватись одночасно з активним фронтом сигналу синхронізації CLK. Сигнал на вході даних повинен бути незмінним протягом часу t_{su} перед приходом активного фронту та протягом часу t_{hold} після приходу активного фронту сигналу синхронізації CLK (рис.1.2). Якщо ця умова порушується, тригер з високою ймовірністю може перейти в метастабільний стан.

Нагадаємо, що коли тригер знаходиться в метастабільному стані на його виході Q присутня напруга, що дорівнює половині напруги живлення (Vdd/2). Це не лог. 0 і не лог. 1, а напруга із забороненого діапазону значень. Тригер знаходиться у метастабільному стані невеликий випадковий проміжок часу, а потім виходить в стан лог.0, або в стан лог.1 випадковим чином.

Дотримання таймінгів t_{su} і t_{hold} не є проблемою всередині мікросхеми, де всі D-тригери підключені до одного джерела сигналу синхронізації. Але що робити, якщо на вхід даних D-тригеру сигнал приходить ззовні мікросхеми і теоретично може змінюватись одночасно з активним фронтом сигналу синхронізації, що формується всередині мікросхеми?

Розглянемо ситуацію зображену на рис.1.13. Маємо D-тригери всередині певної мікросхеми, що тактуються від сигналу синхронізації CLK-B. Активний фронту тригерів передній. Вхід одного з таких тригерів виведений на зовні мікросхеми через контакт Din. Сигнал на вхід Din подається з виходу D-тригеру, який міститься всередині іншої мікросхеми і тактується від сигналу синхронізації CLK-A.

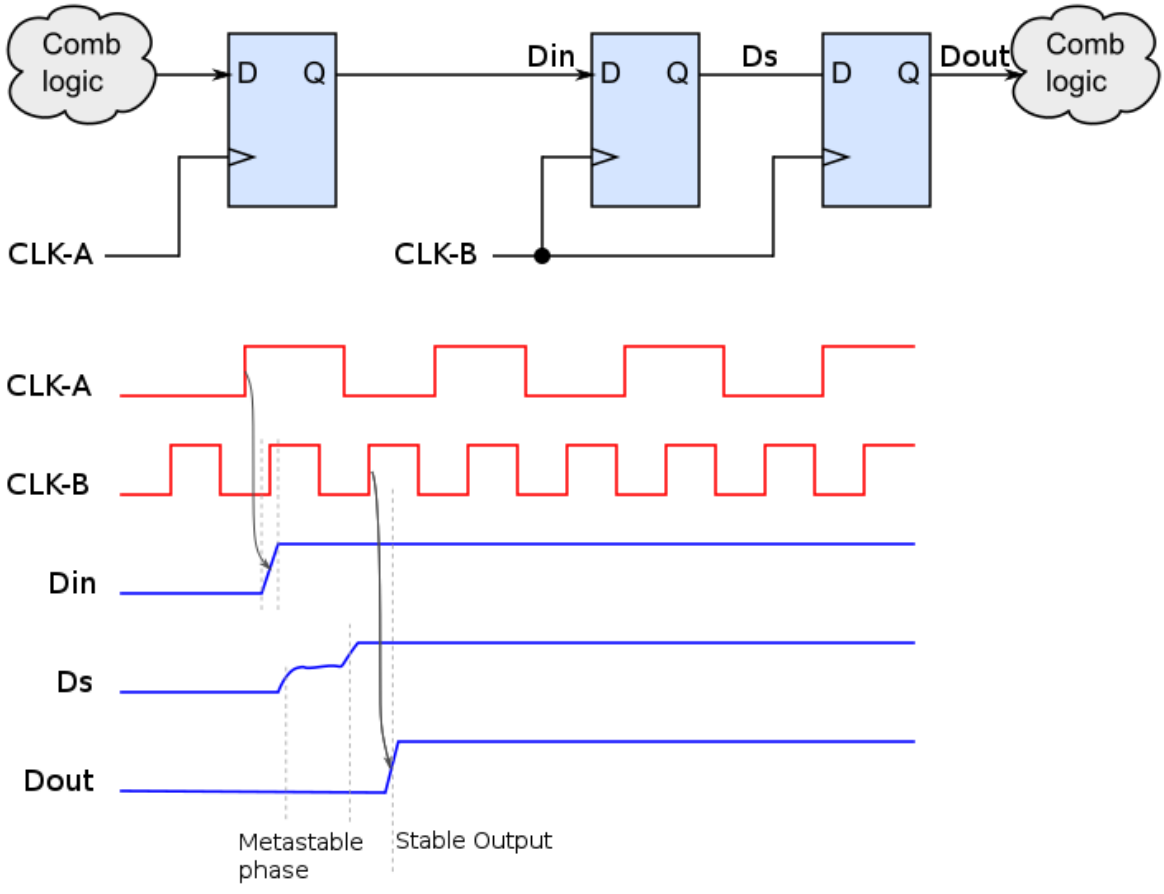


Рис.1.13 - Спосіб синхронізації зовнішніх сигналів, асинхронних до тактового сигналу CLK-B

Взаємозв’язок між тактовими сигналами CLK-A і CLK-B часто невідомий, тому цілком можлива ситуація, коли сигнал на вході Din буде змінюватись одночасно з

переднім фронтом CLK-B. Сигнал на вході Din може формуватися не обов’язково якимось іншим D-тригером. Це може бути сигнал, наприклад, від кнопки, яку натиснув користувач. Але основна проблема залишається незмінною - сигнал на вході Din може змінюватись одночасно з переднім фронтом CLK-B і це може призвести до входження тригеру в метастабільний стан.

Проблема метастабільного стану полягає не лише у тому, що з метастабільного стану тригер виходить випадковим чином. Оскільки D-тригер виходить з метастабільного стану з затримкою, значить нове стійке значення на виході D-тригеру з'являється теж з затримкою і з затримкою подається на вхід комбінаційної логіки, до якої підключений цей вихід D-тригеру. Відповідно на виході комбінаційної логіки та на вході даних наступного D-тригеру сигнал з'являється пізніше ніж планувалося, що може привести (і часто приводить!) до порушення setup/hold таймінгів наступного D-тригеру. Таким чином метастабільний стан може дуже швидко розповсюдитись по великій кількості D-тригерів мікросхеми, привести до неправильних значень сигналів керування та до краху роботи системи. Проблемність подібних помилок полягає у тому, що їх дуже складно відслідкувати, оскільки для виникнення метастабільності необхідна складна комбінація факторів. Тому більшу частину часу система може працювати коректно і збоїти у випадкові непередбачувані моменти часу.

Хороша новина полягає в тому, що для усунення метастабільності на однорозрядному вході достатньо включити послідовно два D-тригери, як це зроблено на рис.1.13. Якщо перший тригер увійде в метастабільний стан, у нього буде цілий період тактового сигналу щоб вийти з метастабільного стану у стійке значення 0 , або 1. По наступному активному фронту це стабільне значення запишеться в наступний тригер. Якщо тригер вийде з метастабільного стану не в те значення, яке подавалося на вхід Din в момент активного фронту CLK-B, обидва тригери ланцюжка синхронізації просто залишаться в старих значеннях, а нове значення на вході Din коректно зчитається по наступному активному фронту і в найгіршому випадку з’явиться на виході другого тригера Dout з затримкою в 1 такт.

Такий підхід називається double-flopping і широко використовується для синхронізації однорозрядних даних зчитаних з входів цифрових мікросхем. Для більшої надійності іноді послідовно включають не 2, а 3 тригери.

Double-flopping не застосовують до синхронізації багаторозрядних входів, оскільки частина розрядів може вийти з метастабільності у правильні значення, а частина розрядів може вийти з метастабільності у хибні значення, що приведе до одержання неправильних даних. У випадку 1-розрядного сигналу такої проблеми не виникає і ми у будь-якому випадку зчитуємо правильне значення, лише іноді з затримкою в 1 такт. Тому при передачі багаторозрядних даних саму шину даних не синхронізують, а синхронізують лише 1-бітний сигнал дозволу запису, що передається разом з шиною даних. Інший спосіб - використання асинхронної черги FIFO, в яку дані записуються по одному сигналу синхронізації, а зчитуються по іншому сигналу синхронізації. Схему асинхронного FIFO розглянемо в одній із наступних робіт.

Детальніше про боротьбу з метастабільністю можна почитати в [1.2].

1.2.16 Детектор фронту

Схема детектора переднього фронту наведена на рис.1.14.

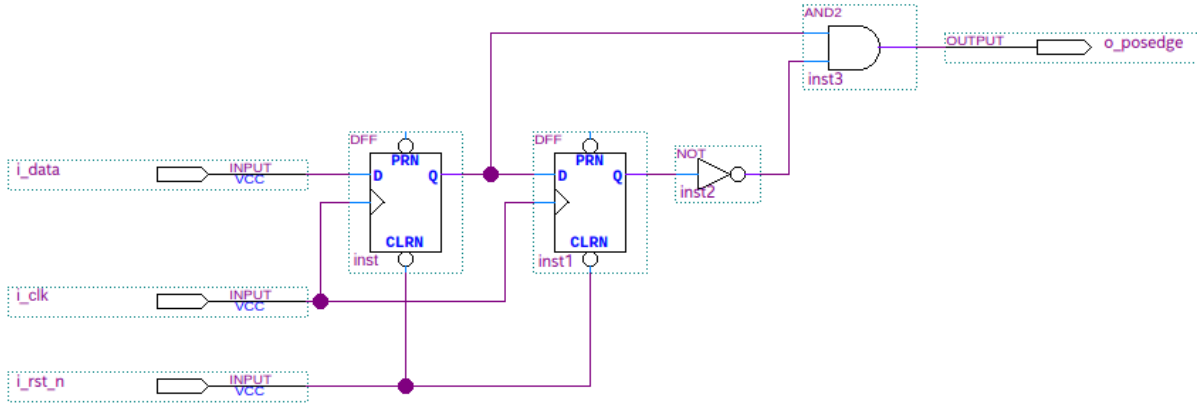


Рис.1.14 - Схема детектора переднього фронту

Часові діаграми зміни сигналів схеми з рис.1.14 наведені на рис.1.15.

Як видно з рис.1.14, детектор фронту містить вхід i_data на який подається сигнал передній фронт якого необхідно визначити. Також детектор фронту містить вхід i_rst_n для асинхронного зкидання обох тригерів та вхід i_clk на який подається сигнал синхронізації. Частота сигналу на вході i_clk повинна бути набагато більша (бажано, разів в 10 і вище) ніж частота зміни сигналу на вході i_data.

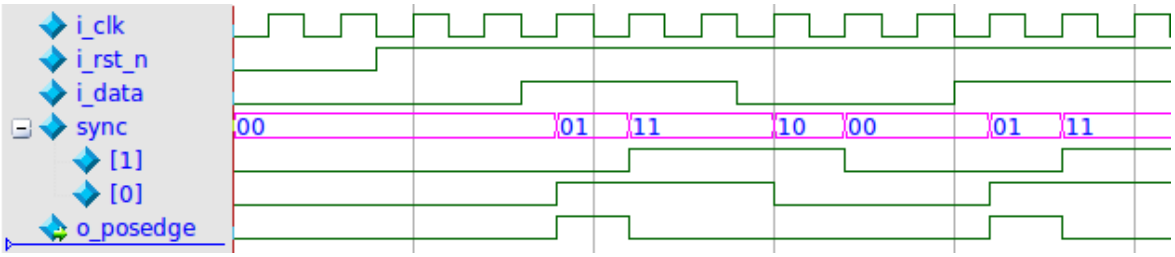


Рис.1.15 - Часові діаграми сигналів у схемі детектору переднього фронту

Після надходження переднього фронту на вхід i_data, на виході o_posedge з'являється імпульс тривалістю один період сигналу i_clk. Розглянемо чому так відбувається. Якщо на вході i_data тривалий час присутній лог.0, обидва тригери будуть в лог.0 і на виході елементу AND буде 0. Припустимо на вхід i_data надійшов сигнал лог. 1 і відповідне значення записалося в перший тригер, а в другому тригері поки що зберігається 0. В такій ситуації на виході елементу AND буде присутній сигнал лог.1. По наступному активному фронту лог. 1 запишеться і в другий тригер. Тепер обидва тригери містять лог.1 і на виході елементу AND буде 0.

Опис детектору переднього фронту на мові Verilog наведено в лістингу 1.10. З аналогічних міркувань не складно створити детектор заднього фронту. Якщо ж виходи тригерів подати на входи елементу XOR, отримаємо детектор, що буде формувати одиночний імпульс після надходження як переднього, так і заднього фронтів.

Лістинг 1.10 - Опис детектору переднього фронту на мові Verilog

```
module edge_detector (i_clk, i_rst_n, i_data, o_posedge);

input     i_clk;
input     i_rst_n;
input     i_data;
output    o_posedge;

reg     [1:0]   sync;

assign o_posedge = ~sync[1] & sync[0];

always @(posedge i_clk, negedge i_rst_n) begin
    if(~i_rst_n) begin
        sync <= 2'b00;
    end else begin
        sync <= {sync[0], i_data};
    end
end

endmodule
```

1.2.17 Інтерфейс SPI

Абревіатура SPI розшифровується як Serial Peripheral Interface. Це широко розповсюджений інтерфейс за допомогою якого до мікроконтролерів і спеціалізованих мікросхем зазвичай підключають різноманітні сенсори, невеликі дисплеї, модулі пам’яті та інші периферійні пристрої, що не потребують надвисокої швидкості передачі даних.

Інтерфейс SPI зазвичай використовують для підключення одного головного пристрою (master device) до одного або кількох периферійних пристроїв (slave devices). Головним пристроєм виступає найчастіше мікроконтролер. Приклади периферійних пристроїв наведені вище.

Сигнали інтерфейсу SPI наведено на рис.1.16. MOSI служить для передачі даних від головного пристрою до периферійного пристрою (MOSI - Master Output Slave Input). MISO служить для передачі даних від периферійного пристрою до головного пристрою (MISO - Master Input Slave Output). SCK є тактовим сигналом синхронізації для передачі, який створює головний пристрій (мікроконтролер). Відповідно SCK направлений від мікроконтролеру до периферійного пристрою/пристроїв.

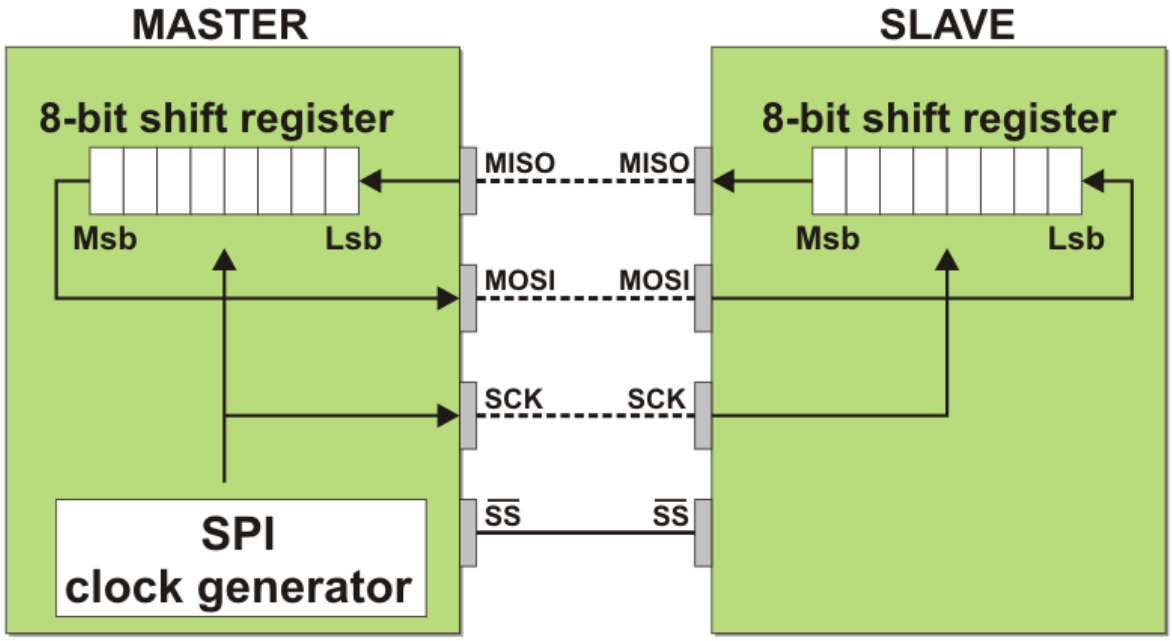


Рис.1.16 - Спрощена схема передачі по інтерфейсу SPI

Сигнал SS розшировується як Slave Select і служить для активації периферійного пристрою мікроконтролером. Якщо до головного пристрою (мікроконтролеру) підключено кілька периферійних пристроїв, до кожного периферійного пристрою від мікроконтролера йде сигнал Slave Select: SS1, SS2, SS3 і т.д. (рис.1.17). При цьому сигнали SCK, MISO і MOSI будуть спільними для мікроконтролеру і периферійних пристроїв (іноді сигнал SCK позначають як SCLK, а SS позначають як CS).

Сигнали SS зазвичай мають активний низький логічний рівень. Це означає, що мікроконтролер для передачі даних по SPI певному периферійному пристрою виставляє логічний 0 на сигналі SS, що підключений для цього пристрою. В кожен момент часу може бути активним лише один сигнал SS (в кожен момент часу мікроконтролер працює лише з одним периферійним пристроєм).

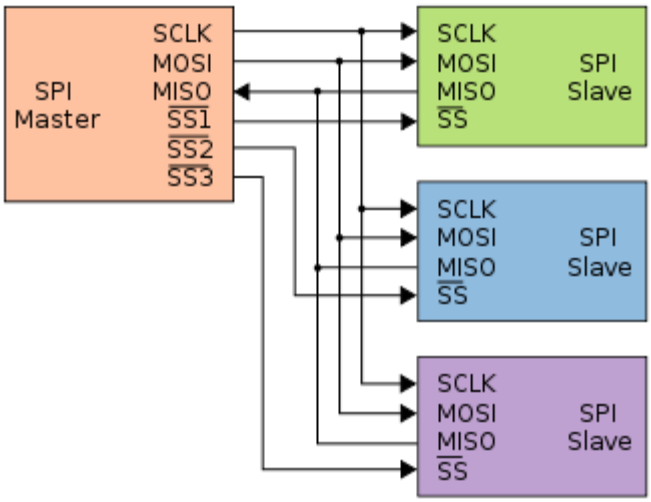


Рис.1.17 - Підключення кількох SPI приймачів до SPI передавача

Особливістю SPI є той факт, що передача і прийом даних відбуваються одночасно. Всі транзакції (прийом/передача) ініціює головний пристрій (master device). Якщо повернутися до рис.1.16 видно, що для передачі даних в SPI використовують регістри зсуву. Як головний, так і периферійні пристрої зчитують дані по активному фронту сигналу SCK, який формує головний пристрій (мікроконтролер). По неактивному фронту SCK головний пристрій видає на MOSI новий біт даних, а периферійний пристрій видає новий біт даних на MISO. Імпульси на SCK формуються лише під час передачі даних. Якщо дані по MOSI/MISO не передаються, імпульси на SCK відсутні.

Дані зазвичай передаються по 8 біт підряд (хоча іноді використовують і 16 бітну передачу). Кожному біту даних, відповідає 1 імпульс на SCK. Дані зазвичай передаються старшим бітом вперед (старший біт часто позначають як MSB - Most Significant Bit, а молодший біт позначають як LSB - Least Significant Bit).

Відповідно до рис.1.16 у найпростішому випадку SPI MASTER і SPI SLAVE містять регістри зсуву, що тактуються від SCK. Дані в регістрах зсуву зсуваються від молодшого біту до старшого. Спершу в регістри зсуву завантажують необхідні дані. Далі головний пристрій (мікроконтролер) формує 8 імпульсів тактової частоти на лінії SCK і регістри зсуву обмінюються між собою даними.

В лабораторній роботі ми поки що створимо SPI приймач, який лише приймає дані. Але в реальних периферійних пристроях які повинні і приймати і передавати дані, використовують наступний підхід. Спершу мікроконтролер виставляє активний низький рівень (лог.0) на лінії SS периферійного пристрою. Далі мікроконтролер передає в периферійний пристрій 8 біт даних (дані прийняті на MISO на цьому етапі мікроконтролер ігнорує). Переданий байт містить 2 конфігураційних біта і 6 біт, що задають номер регістру периферійного пристрою з яким необхідно обмінюватись даними. Один із згаданих конфігураційних бітів задає напрямок передачі для наступного байту. Якщо біт напрямку передачі приймає значення 0, наутпний байт буде записуватися в регістр периферійного пристрою з переданою щойно адресою. Якщо біт напрямку передачі приймає значення 1, наступний байт буде зчитуватися з регістру периферійного пристрою, що має передану щойно адресу. Після передачі другого байту мікроконтролер знову робить неактивним сигнал SS для даного периферійного пристрою. Приклад зчитування по інтерфейсу SPI з використанням подібного підходу показано на рис.1.18.

Інтерфейс SPI може працювати в 4-х режимах, що визначаються 1-бітними параметрами CPOL (скорочення від Clock Polarity) та CPHA (скорочення від Clock Phase). Зазвичай для кожного периферійного пристрою з документації відомі значення CPOL та CPHA і мікроконтролер можна налаштувати на роботу в режимі SPI, що визначається цими відомими значеннями.

Значення параметрів CPOL та CPHA пояснені на рис.1.19

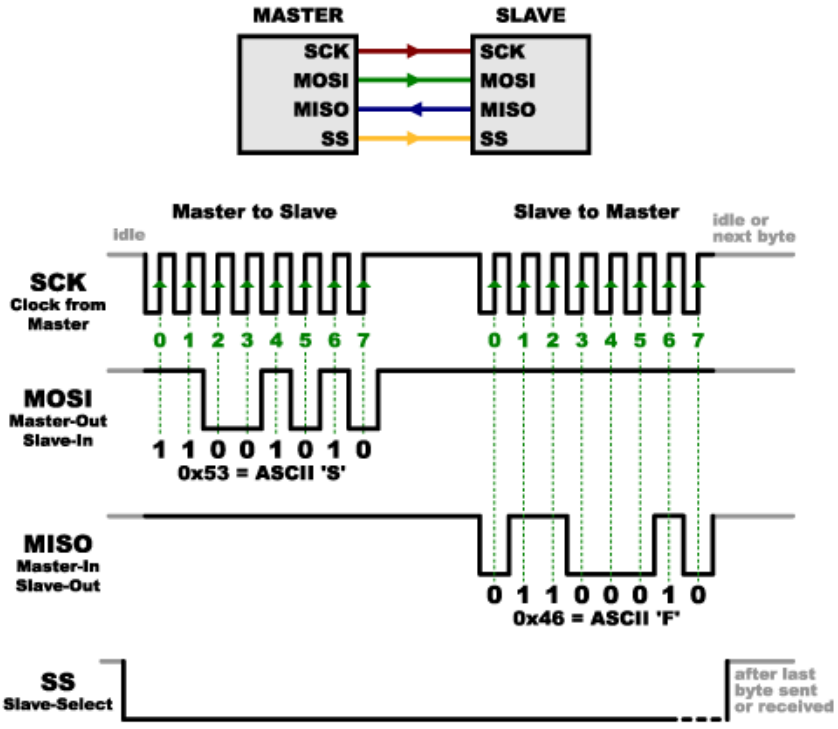


Рис.1.18 - Приклад зчитування по інтерфейсу SPI

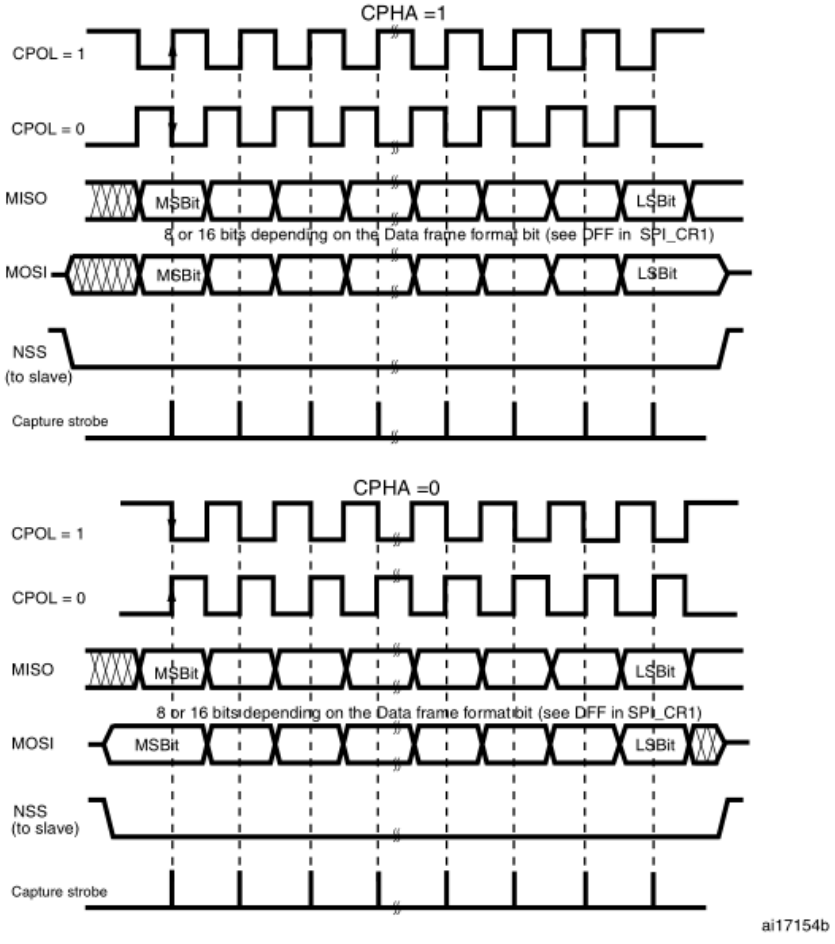


Рис.1.19 - Режими роботи SPI

Якщо CPOL=0, це означає, що при відсутній передачі даних по SPI сигнал SCK приймає значення 0. Якщо CPOL=1, то при відсутній передачі даних по SPI сигнал SCK приймає значення 1.

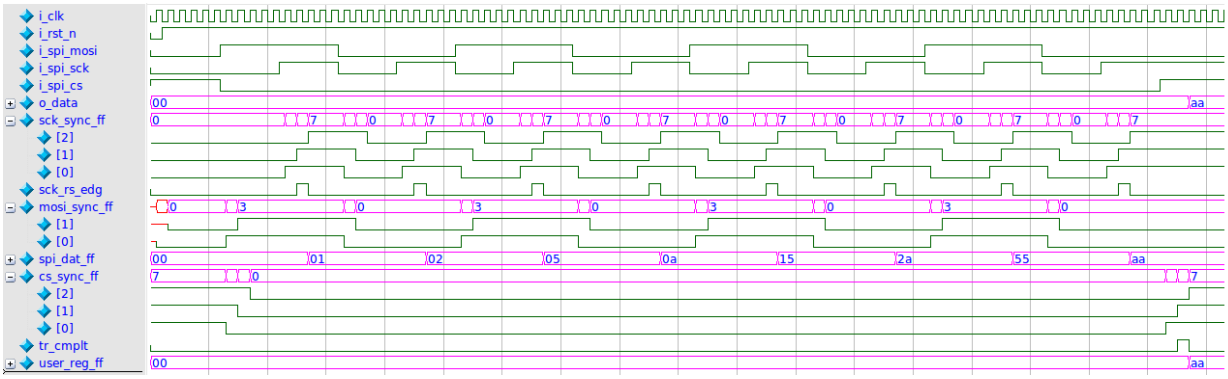
Якщо CPHA=0 це означає, що активним фронтом є перший фронт тактового імпульса на SCK. У випадку коли CPHA=1 активним фронтом є другий фронт тактового імпульса на SCK.

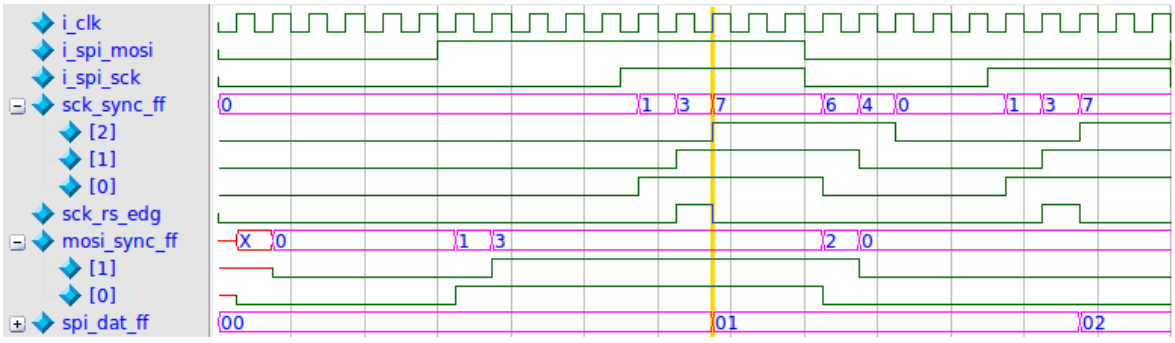
В наступному розділі розглянемо апаратний SPI приймач у якого CPOL=0, CPHA=0. Це означає, що при відсутності передачі даних по SPI на SCK буде присутній 0, а активним фронтом буде перший фронт імпульсу SCK (дані зчитуються по передньому фронту SCK і виставляються по задньому фронту SCK).

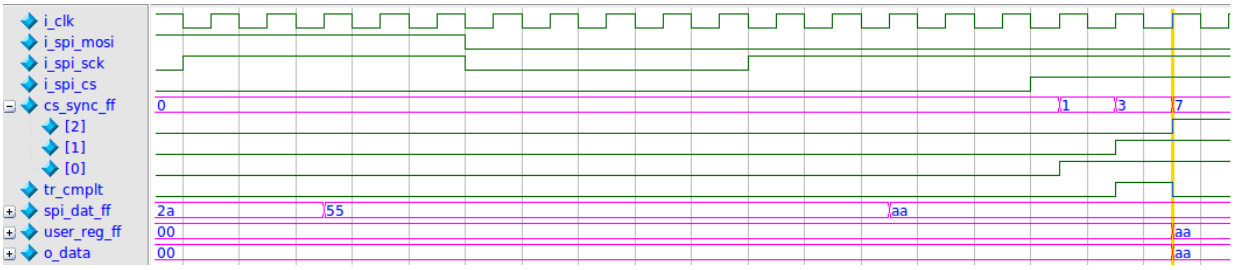
1.2.18 SPI приймач

```
module simple_spi (MAX10_CLK1_50, GPIO, KEY, LEDR);

input           MAX10_CLK1_50;
input   [35:0]  GPIO;
input   [1:0]   KEY;
output  [9:0]   LEDR;

wire    sys_clk    = MAX10_CLK1_50;
wire    sys_rst    = KEY[1];
wire    spi_sck    = GPIO[0];
wire    spi_cs     = GPIO[1];
wire    spi_mosi   = GPIO[2];

wire    sck_rs_edg;
wire    tr_cmplt;

reg     [7:0]   user_reg_ff;
reg     [7:0]   spi_dat_ff;
reg     [2:0]   sck_sync_ff;
reg     [2:0]   cs_sync_ff;
reg     [1:0]   mosi_sync_ff;

assign    LEDR        = user_reg_ff;
assign    sck_rs_edg  = ~sck_sync_ff[2] & sck_sync_ff[1];
assign    tr_cmplt    = ~cs_sync_ff[2] & cs_sync_ff[1];

always @(posedge sys_clk, negedge sys_rst) begin
    if (~sys_rst) begin
        sck_sync_ff <= 3'b000;
    end else begin
        sck_sync_ff <= {sck_sync_ff[1:0], spi_sck};
    end
end

always @(posedge sys_clk, negedge sys_rst) begin
    if (~sys_rst) begin
        cs_sync_ff <= 3'b111;
    end else begin
        cs_sync_ff <= {cs_sync_ff[1:0], spi_cs};
    end
```

```
    end

    always @(posedge sys_clk)
        mosi_sync_ff <= {mosi_sync_ff[0], spi_mosi};

    always @(posedge sys_clk) begin
        if (sck_rs_edg)
            spi_dat_ff <= {spi_dat_ff[6:0], mosi_sync_ff[1]};
    end

    always @(posedge sys_clk)
        if (tr_cmplt)
            user_reg_ff <= spi_dat_ff;

    endmodule
```







1.2.19 Програмний код для SPI передавача

```
#include "SPI.h"

int spi_cs = 8;

void setup()
{
    pinMode(spi_cs, OUTPUT);
    digitalWrite(spi_cs, HIGH);
    SPI.begin();
    SPI.beginTransaction(SPISettings(1000000,MSBFIRST,SPI_MODE0));
}

void loop()
{
    for(uint8_t val = 0; val < 256; val++) {
        delay(500);
        digitalWrite(spi_cs, LOW);
        SPI.transfer(val);
        digitalWrite(spi_cs, HIGH);
```

```
        }
    }
```

```
#include "mbed.h"

SPI spi(D11, D12, D13); // spi_mosi, spi_miso, spi_sck
DigitalOut cs(D10);     // spi_cs

int main()
{
    cs = 1;
    uint8_t val = 0;

    // Setup the spi for 8 bit data, high steady state clock,
    // second edge capture, with a 1MHz clock rate
    spi.format(8,3);
    spi.frequency(1000000);

    while (1) {
        wait(0.5);
        cs = 0;
        spi.write(val++);
        cs = 1;
    }
}
```

1.3 Практична частина

1.3.1 Передача даних по SPI з використанням Analog Discovery 2

1.3.2 Передача даних по SPI з використанням Arduino

1.3.3 Передача даних по SPI з використанням MBED та Nucleo F401RE

1.3.4 Завдання на лабораторну роботу

1.4 Контрольні запитання

1.5 Перелік посилань

[1.1] “Linear Feedback Shift Registers in Virtex Devices. Application Note XAPP-210”, 2007. [Online]. Available: https://www.xilinx.com/support/documentation/application_notes/xapp210.pdf

[1.2] “Метастабільнітсь тригера і міжтактова синхронізація”, 2015 [Online]. Режим доступу: https://habr.com/post/254869/

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