Пристрій для розпізнавання звуку на базі ПЛІС

Шифр: «Розпізнавач звуку»

ЗМІСТ

ВСТУП

Галузь розпізнавання звуку і сигналів є актуальною на сьогоднішній час. Її результати використовуються в різних галузях науки і повсякденного життя. Наприклад розпізнавання звуку використовується в охоронній діяльності для розпізнавання порушень[1], або ж в зоології для ідентифікації різних істот[1]. Говорячи інакше, розпізнавання звуку допомагає «перевести в цифру» існуючий аналоговий світ. Однак постає питання із наявністю готових приладів або ж рішень для розпізнавання сигналів.

Нажаль не всі існуючі на ринку рішення дозволяють використовувати їх для запису одного сигналу а також деякі не дозволяють її робити взагалі, використовуючи для розпізнавання заздалегідь записані семпли. Проблеми також виникають при використанні існуючих блоків в якості складової частини системи. На цьому етапі можуть виникати проблеми із підключенням існуючих рішень у загальну систему.

Цікавим варіантом серед існуючих рішень є програмне ядро на базі ПЛІС що дозволяє робити наступне використовувати ядро у якості окремого пристрою для розпізнавання звуку або ж використовувати ядро у якості більш складної системи на ПЛІС. Відповідно це дозволить мати більшу гнучкість при проектуванні складних систем або ж окремих пристроїв для розпізнавання звуку. Слід зауважити, що написання окремого ядра для ПЛІС дозволяє в майбутньому не тільки реалізовувати проект на ПЛІС, а й також, виготовляти кінцеві рішення на базі спеціалізованих мікросхем спроектованих на основі ПЛІС ядра.

Однак таке рішення буде вимагати від інженера додаткової кваліфікації проте, дозволить більш гнучко підходити до процесу проектування приладу або ж системи.

1 ТЕОРЕТИЧНЕ ОБҐРУНТУВАННЯ

В загальному розпізнавання сигналу можна звести до пошуку подібності або ж розбіжності сигналів. Її пошук відбувається за допомогою звичайної взаємокореляції двох сигналів, результатом якої є показник подібності. Взаємокореляція може мати як неперервну форму(1.1), так і дискретну(1.2) [2].

$$(f * g)(\tau) \triangleq \int_{-\infty}^{+\infty} \overline{f(t)} + g(t+\tau)dt \tag{1.1}$$

$$(f * g)[\mathrm{n}] \triangleq \sum_{m=-\infty}^{\infty} \overline{f(t)} + g[m+n]dt \tag{1.2}$$

В загальному обрахунок кореляції можна представити як обрахунок перекриття площ двох сигналів.

Однак не завжди прямий пошук подібності дає прийнятний результат. Для прикладу якщо корелювати два однакових сигнали з різними масштабами їх ступінь подібності буде менша, ніж якщо вони будуть одного масштабу. В загальному це є вірним результатом, однак є сфери де така особливість не має впливати на процес розпізнавання. Наприклад для звукового сигналу різне масштабування є всього лише різною гучністю, і коли необхідно розпізнати звук гучністю можна знехтувати. В таких випадках відбувається перетворення вхідного сигналу в новий базис де знаходиться подібність між сигналами. Перетворення між базисами потрібне для того, щоб висвітлити ті чи інші особливості які складно зрозуміти в поточному базисі. Одним з найвідоміших базисних перетворень є перетворення Фур'є. Воно дозволяє отримувати спектр сигналів що застосовується майже в усіх задачах що пов’язані з обробкою сигналів. Для Фур’є перетворення базисними функціями є функції синуса і косинуса, використання їх дозволяє отримати послідовність значень(ряд) коефіцієнти якого відповідають інтенсивності впливу тої чи іншої базисної складової певної частоти, що зображено на рисунку 1.1.

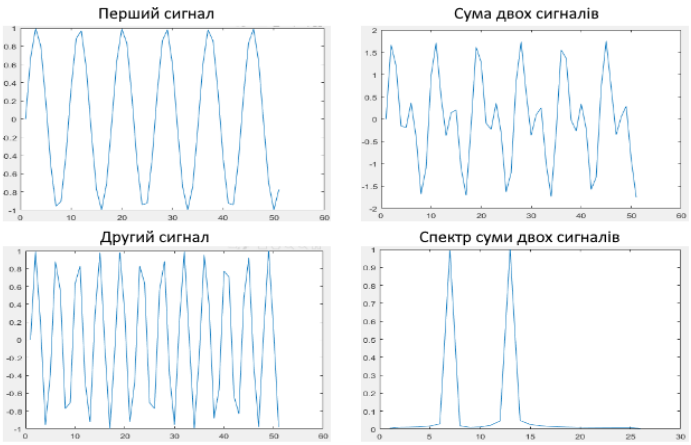

Рисунок 1.1 - Ілюстрація взаємозв’язку сигналу із спектром

Для будь-яких перетворень з одного базису до іншого існують спеціальні операції за допомогою яких ці перетворення виконуються. У випадку з розкладом у ряд Фур’є використовують так звані перетворення Фур’є (1.3).

$$\hat{f}(\omega) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} f(x)e^{-ix\omega}dx \tag{1.3}$$

Однак звичайне перетворення Фур’є працює з неперервним сигналом, що не підходить для дискретних сигналів, тому для дискретних сигналів застосовується спеціальне дискретне перетворення Фур’є[3] (1.4).

$$X_k = \sum_{n=0}^{N-1} x * e^{-\frac{i2\pi}{N}kn} = \sum_{n=0}^{N-1} x * \left[\cos\left(\frac{2\pi}{N}kn\right) - i * \sin\left(\frac{2\pi}{N}kn\right)\right] \tag{1.4}$$

Як можна побачити згідно наведеної вище формули, ДПФ(Дискретне перетворення Фур’є) виконується для дискретних відліків сигналу, внаслідок чого його можна застосовувати в радіотехніці. Однак навіть у ДПФ є ряд недоліків, по-перше: велика обчислювальна складність а саме – $O(N^2)$, внаслідок чого швидкість роботи є відносно невисока; по-друге: через велику кількість операції сумарна похибка стає значно більшої ніж могла би бути.

Через описані вище недоліки ДПФ майже не використовується в радіотехніці, замість нього застосовуються швидке дискретне перетворення Фур'є.

Загалом методи ШПФ (швидке перетворення Фур'є) базуються на тому факті, що загальну кількість відліків, можна розбити на кілька менших, внаслідок чого, можна зменшити ступінь необхідного ШПФ для обрахунку. Існує кілька методів для обрахунку ШПФ, однак не будемо їх розглядати в рамках цієї роботи.

В загальному ШПФ можна звести до трьох дій – перестановка, двох точкове перетворення Фур'є, та злиття. На етапі перестановки відбувається рекурентне розбиття масиву на під масиви, однак, якщо довжина масиву є заздалегідь відомою, то можна спростити цей крок, просто заздалегідь звертатись до певних елементів. На етапі двох точкового перетворення відбувається обрахунок спектру двох точкового сигналу. На етапі злиття отриманий двох точковий спектр зсувається і об'єднується з іншими спектрами.

1.1 Розробка алгоритму

Оскільки теоретично було встановлено що можна розпізнати звук, і на чому це базується, перейдемо до проектування алгоритму. В якості основного базису оберемо базис Фур'є, тобто іншими словами будемо працювати не з сигналом, а з його спектром. Це має наступні переваги такі переваги як: більша стабільність і чутливість до спектральних змін сигналу, однак недоліком цього є необхідність у постійному виконанні перетворення Фур'є. Однак перед тим як визначитись з алгоритмом розглянемо ще один момент, кількість вхідних відліків сигналу може бути достатньо високою, і для її зменшення можна скористатись ваговим перетворенням, це дозволить зменшити кількість значень які потрібно обробити при тому не значно втративши в якості[4]. Далі слід визначитись з алгоритмом перетворення Фур'є, оскільки сигнал є дискретним то й перетворення повинно бути дискретним, існують різні перетворення Фур'є однак ми зараз звернемо увагу лише на два: ДПФ та дискретне ШПФ, логічно

що ДПФ не має сенсу використовувати по причинах що були описані вище. Лишається ШПФ, однак його можна реалізувати різними алгоритмами, однак вони в загальному подібні тому не будемо загострювати на цьому увагу, оскільки це не є принциповим для розпізнавання, головне що оберемо алгоритм в якого кількість відліків кратна двійці. Така вимога є через те, що алгоритми в яких кількість відліків не рівна степені двійки складніші в реалізації. Після виконання перетворення Фур'є на виході отримуємо комплексний сигнал однак для задачі розпізнавання ми можемо його спростити до модуля від комплексного числа, це має дві переваги: по-перше перетворений сигнал займає вдвічі менше пам'яті; по-друге кореляція перетвореного сигналу буде займати вдвічі менше часу або ж логічних елементів. Тобто після Фур'є перетворення має також обраховуватись модуль комплексного числа. Далі можна виконати кореляцію над отриманими модулями і шаблонними, заздалегідь записаними значеннями модуля.

Операція кореляції включає в себе перемноження двох сигналів і їх інтегрування або ж сумування. Однак недоліком такого методу є те, що він підвищує час виконання обрахунку. Оскільки перемноження чисел є досить затратною операцією, незалежно від того є число цілочисельним чи не є. Тому додатково розглянемо й більш спрощений варіант. Для нього ми будемо виконувати віднімання більшого значення від меншого і виконувати сумування результату з іншими складовими. В результаті отримане число буде вказувати наскільки один сигнал відрізняється від іншого. При найбільших значеннях – сигнали повністю не подібні, при нульовому – сигнали ідентичні.

Відповідно після обрахунку кореляції необхідно визначити чи сигнал був розпізнаний, для цього, зазвичай, використовують певне порогове значення, порівняння з яким буде свідчити про те, що сигнал розпізнаний. Таке значення має задаватись ззовні на основі звуків, які аналізуються, розпізнавальної здатності а також заданих похибок таких як хибне спрацювання, тощо.

Узагальнимо вище описане: спочатку сигнал потрапляє на вагове вікно, де відбувається його перетворення після якого здійснюється обрахунок

перетворення ШПФ і з отриманих комплексних значень обчислюється модуль комплексного числа, після чого модулі корелюються з шаблонними модулями сигналу і якщо результат менше певного порогового значення можна стверджувати що це той самий сигнал, інакше – сигнали є різними. Даний алгоритм є зображеним нижче на рисунку 1.2.

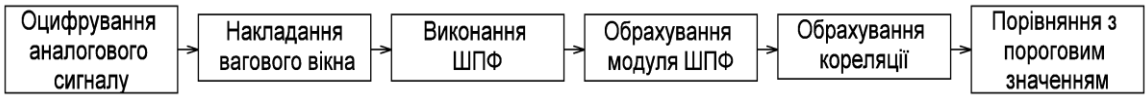

Рисунок 1.2 – Алгоритм розпізнавання сигналу

1.2 Реалізація на базі мови MATLAB

Так як була в минулому підрозділі було сформовано твердження щодо можливості розпізнавання сигналу, а також було запропоновано алгоритм процесу розпізнавання сигналу, слід перевірити можливість розпізнавання сигналу, на базі якогось існуючого математичного пакету, для прикладу візьмемо MATLAB. Для перевірки необхідно описати існуючий алгоритм за допомогою мат. Пакету, для цього сформулюємо що має робити програма.

На початку необхідно прочитати файл із даними, в якості таких можна, наприклад, використати звукові дані. Після чого необхідно вибрати певний фрагмент звукової послідовності, і на її основі зробити запис шаблонного сигналу. Для цього необхідно виділити певний момент звуку, після чого виконати перетворення фур’є і взяти для отриманого результату модуль – внаслідок чого отримане значення буде шаблонним звуком. Після того як шаблонний звук був обрахований потрібно зробити розпізнавання на основі вже повного набору даних що було отримано на початку. Для розпізнавання звуку необхідно так само застосувати вище описаний принцип обрахунку шаблонного звуку, після чого виконати кореляцію.

В результаті отримаємо програму наведену в додатку А. В наслідок її виконання отримаємо графік, зображений на рисунку 1.3, на ньому

спостерігаємо залежність величини розбіжності двох сигналів від номеру початку вхідної вибірки. Цей показник демонструє наскільки два сигналу відрізняються один від одного, при тому що нульове значення відповідає двом ідентичним сигналам. Найбільшу увагу привертає нульовий пік на сотій виборці. Справа в тому, що з 100-тої вибірки, починається шаблонний звук, тобто це говорить про те, що вхідні сигнали є подібними і алгоритм дозволив це розпізнати.

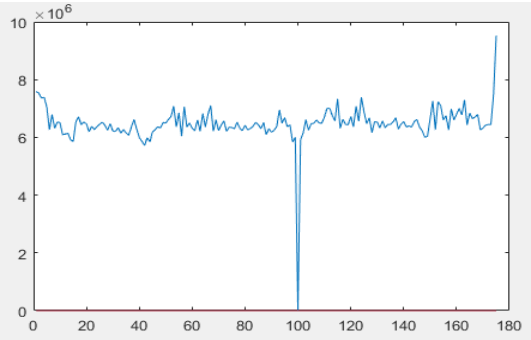

Рисунок 1.3 – Графік розбіжності двох сигналів від номера вибірки

На основі вище наведеного графіку робимо висновок що даний принцип розпізнавання звуку працює. Це дозволяє перейти до розробки схеми.

1.3 Розробка структурної схеми

Для того, щоб почати розробку пристрою, передусім, необхідно синтезувати певну функціональну схему пристрою, що складалась би з окремих блоків. Схема має виконувати наступні дії – запам’ятовувати вхідний сигнал, розпізнавати сигнал, а також має мати можливість до запису і зберігання шаблонного звуку. На основі цього синтезуємо схему, для якої приймемо, для прикладу, що довжина розпізнаваного звуку має бути рівна 5 секундам, тоді отримуємо наступну схему, зображену на рисунку 1.4.

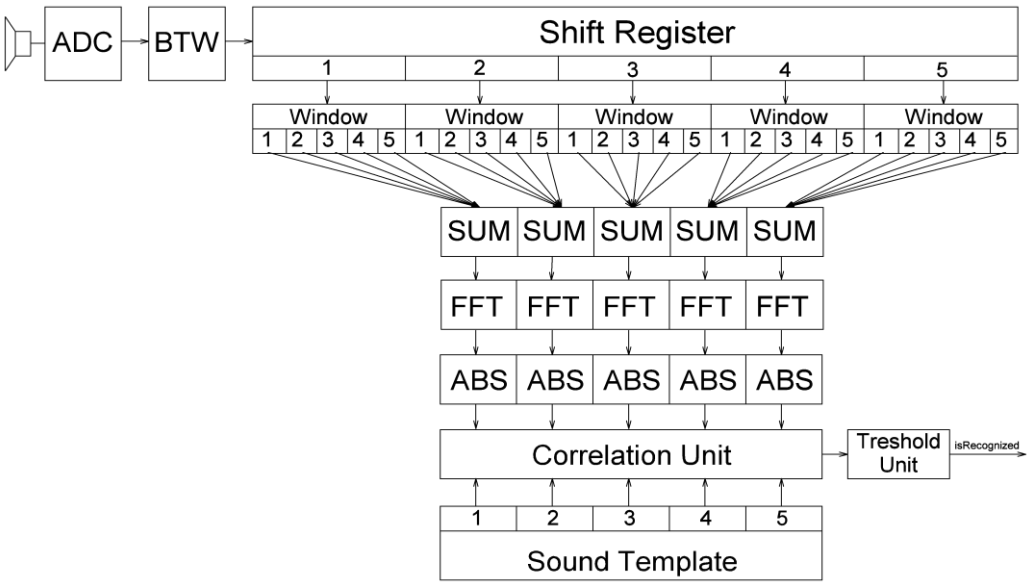

Рисунок 1.4 – Структурна схема пристрою для розпізнавання звуку

Розглянемо принцип роботи цієї схеми. Бінарний сигнал з АЦП (Аналого-цифровий перетворювач) потрапляє на перетворювач двійкового коду в слово BTW (Binary to Word) і після чого потрапляє до регістру зсуву (Shift register), який виконує функцію тимчасової пам’яті. Наступним кроком сигнали потрапляють на віконний блок(Window), що виконує перемноження відліків на вагові коефіцієнти. Після перемноження на вагове вікно відбувається розбиття вибірок на підвибірки, для прикладу у наведеній вище схемі – п’ять підвибірок, ці підвибірки потрапляють на блок підсумувування(SUM) для підсумування між собою. Далі відліки потрапляють на блоки швидкого перетворення Фур’є (FFT – Fast Fourier Transformation), що формують комплексні спектрограми, після чого отримані спектрограми потрапляють на блок обрахування модулів комплексних спектрограм(ABS – Absolute value). Після обрахування модуля відбувається кореляція отриманих модулів спектрограм з раніше записаною спектрограмою(Sound Template). Отримане значення потрапляє на порогів пристрій, який вирішує, чи вхідна спектрограма відповідає спектрограмі шаблонного звуку.

Наведені вище блоки: вагового вікна (window), підсумовування(SUM) і перетворювача фур’є (FFT) – реалізують алгоритм поліфазного швидкого перетворення фур’є[4].

Тобто в наведеній схемі роль кожного блоку є строго визначена і він виконує лише свою певну функцію, наслідком цього є те, що усі блоки можуть бути незалежно спроектованими і перевіреними що в свою чергу спрощує розробку і подальшу модернізацію.

2 ПРОЕКТУВАННЯ ПРИСТОРОЮ МОВОЮ VERILOG

Проектування пристрою буде відбуватись за допомогою мови опису пристроїв Verilog. В якості середовища моделювання буде використано програмний засіб ModelSim компанії Intel.

Наступним кроком визначимося з алгоритмом проектування. В загальному, для спрощення аналізу приладів, зазвичай приймають, що блок є «чорним ящиком» з певним набором входів і виходів. Базуючись на тому описуються певні внутрішні залежності[5]. Тому нехай в якості першого етапу проектування пристрою буде відбуватись проектування «інтерфейсу» - тобто певного набору входів і виходів модуля з чітко визначеними функціями. Після того ми можемо перейти вже до безпосереднього написання коду, що буде описувати внутрішню поведінку пристрою, що і буде наступним етапом.

Після того, як блок спроектований необхідно перевірити чи він дійсно правильно працює і робить те, що очікується, для цього необхідно буде виконати два етапи, які будуть в нас наступними – написання тестового модуля і аналіз часових діаграм. Під написанням тестового модуля розуміють проектування окремого блоку, який буде на існуючий чорний ящик надсилати ті чи інші сигнали, і відображати реакцію. Аналіз часових діаграм відбувається зазвичай для тестового блоку, який виводить часові діаграми роботи чорного ящика, що дозволяє оцінити різні параметри, такі як коректність роботи, затримки, тощо.

Тобто в загальному алгоритм проектування блоку буде відбуватись наступним чином: спочатку проектуємо інтерфейс, після чого пишемо код модуля, далі пишемо код для перевірки тестового блоку, і заключним етапом аналізуємо часові діаграми.

2.1 Перетворювач двійкового коду в слово

Для початку визначимо інтерфейс даного блоку, так як для усіх блоків в нас буде використовуватись синхронна логіка то першим необхідним входом має бути блок входу тактованого сигналу – «clk». Також логічно, що окрім тактованого сигналу також має бути вхід, на який буде поступати бінарний сигнал, що буде перетворюватись в слово, нехай цей вхід буде називатись – «in». Цілком логічно що для початку роботи або ж при неочікуваній ситуацій необхідно виконати скид, тому наступним вхідним сигналом буде сигнал скиду – «rst». Стосовно вихідних сигналів має бути, щонайменше, два сигнали – шина сигналів, на яку буде виставлятись перетворений слово, назвемо її – «out», а також вихід, що буде сигналізувати про те, що перетворення є виконаним, назвемо цей сигнал – «cnv_cmplt».

Перейдемо до проектування блоку, так як він робитиме перетворення бінарного сигналу в слово, то перш за все, необхідно сформувати лічильник який буде вказувати на біт який зараз використовується для перетворення. Після того необхідно додати внутрішній регістр який буде зберігати проміжне значення, маючи такий регістр ми будемо записувати поточне значення із входу до біту проміжного регістру, при тому біт буде визначатись за допомогою лічильника. Також необхідно створити формування сигналу про завершення перетворення. Для цього зробимо, щоб при певному значенні лічильника, було формування сигналу «cnv_cmplt», а також відбувався скид лічильника. На завершення необхідно додати реалізацію скиду, що буде занулювати усі вищеописані регістри.

Знаючи як працює модуль перейдемо до написання тестового модуля. Для того щоб перевірити роботу модуля необхідно зробити наступне – сформувати

певне значення яке буде формувати слово, і після чого це значення по біту надсилати на вхід. В загальному цей принцип є зворотнім до вищеописаного, тому пропустимо детальний розгляд принципів його роботи і перейдемо до розгляду сформованих часових діаграм, наведених на рисунку 2.1.

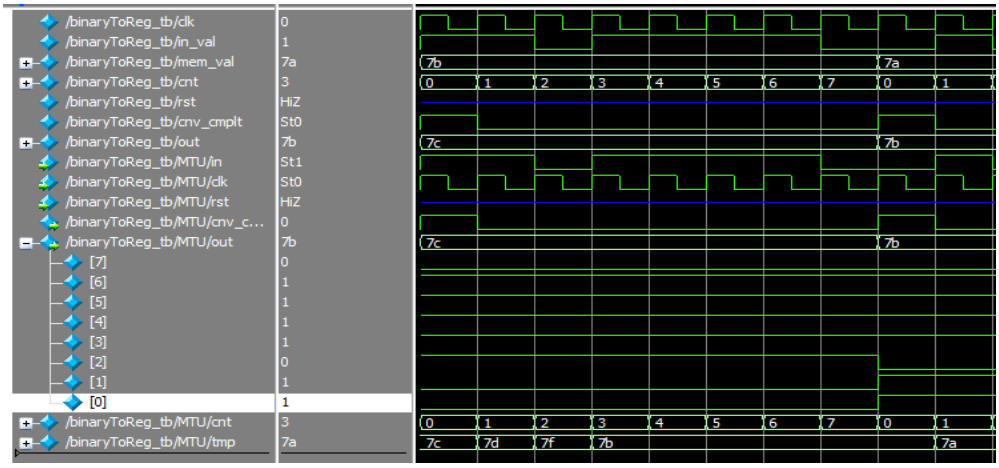

Рисунок 2.1 - Часова діаграма входів виходів перетворювача двійкового коду в слово

Розглянемо отримані часові діаграми, перш за все на цікавлять такі шини даних як «mem_val», «cnv_cmlt», «in», «out». Значення шини даних «mem_val», відповідає значенню слова що відправляється на перетворювач, для прикладу, на початку перетворення, воно є 0x7b далі це значення по біту потрапляє на вхід «in», з цих значень відбувається формування тимчасового значення регістру – «tmp», після того як 8 біт пройшли, спочатку формується сигнал «cnv_cmlt», після чого, із затримкою в один такт на лінії «out» з’являється значення 0x7b, після чого увесь описаний цикл повторюється.

Так як робота даного блоку відповідає очікуваній,будемо вважати, що блок був успішно сформований і перейдемо до проектування наступного.

2.2 Регістр зсуву

Для проектування даного блоку аналогічно до минулого блоку почнемо проектування з визначення інтерфейсу. Аналогічно до попереднього модуля,

проектований блок має мати наступні входи: «clk» - для такового сигналу і «rst» для скиду. І останнім із вхідних сигналів, буде сигнал «strobe», зміна якого викликатиме роботу пристрою,це дозволить спростити подальше проектування. Перейдемо до виходів схеми, так як цей регістр виконує ще функцію перетворення послідовної послідовності слів в паралельну, то в даного блоку має бути сигнал «out», що буде рівний кількості усіх слів що зберігаються перемножену на розрядність окремого слова. Оскільки модуль має підтримувати каскадування, то в нього має бути вихід «carry», що буде рівним довжині одного слова, значення на цей вихід будуть потрапляти тоді, коли вони вже не зберігаються в регістрі пам’яті. Заключним сигналом є сигнал «out_strobe» - що формується після того як значення було оброблено, і на шині «out» знаходяться актуальні дані.

Перейдемо до проектування блоку. Даний блок має виконувати функцію зберігання даних, тобто він має мати набір регістрів, що будуть зберігати значення. Окрім того необхідно виконувати зсув даних, що зберігаються у регістрі, для чого будемо використовуючи не блокуюче присвоєння, останнє значення з регістру зсуву має виводитись на шину даних «carry». Також необхідно сформувати сигнал «out_strobe», який буде формуватись за допомогою окремого під модуля, який буде реалізовувати затримку сигналу на певну кількість тактів. На завершення необхідно додати реалізацію скиду, що буде занулювати усі вищеописані регістри.

Знаючи як працює модуль перейдемо до написання тестового модуля. Для того щоб перевірити роботу модуля необхідно зробити наступне – сформувати слово певної розрядності, що рівна розрядності слова регістру, після чого це значення подати на вхід регістру. Як бачимо, тестовий модуль є дуже простим для цього випадку, тому перейдемо до розгляду часових діаграм, що зображені на рисунку 2.2.

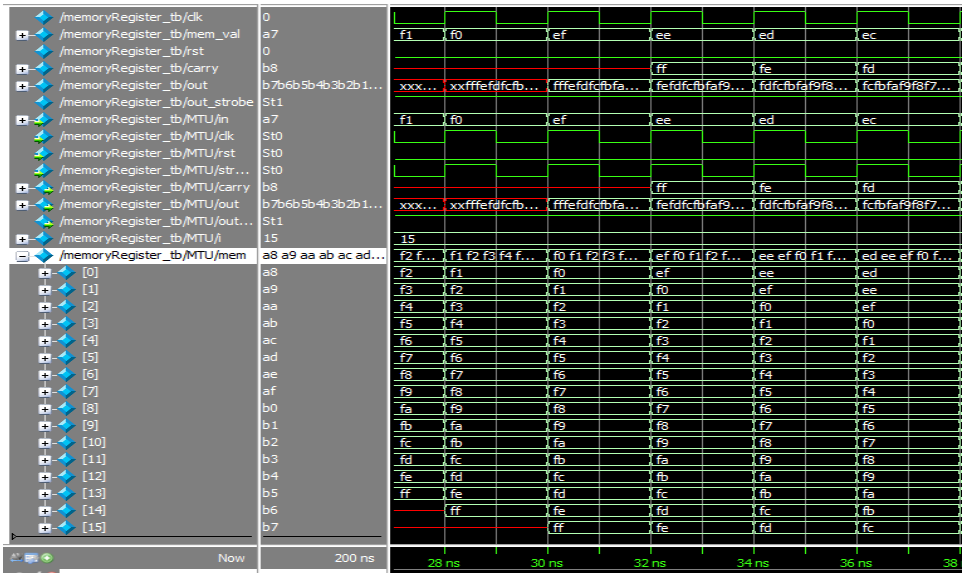

Рисунок 2.2 - Часова діаграма входів виходів регістру зсуву

Для розгляду даних діаграм необхідно уточнити певні моменти: по-перше, на початку роботи регістр пам’яті містить слова з невизначеним станом що змінюється після того як усі регістри заповнюються даними, що потрапляють ззовні при модуляції; по друге, порядок байт для шини виходу відрізняється від прямого, в даному випадку, він є від старшого до молодшого, внаслідок чого, 16-те слово з регістру пам’яті на 30-й наносекунді зустрічається на початку шини «out».

Перейдемо до розгляду часових діаграм, перш за все нас цікавлять такі шини даних як «carry» «in» «mem» та «out». Розглянемо кілька тактів роботи. Спочатку на 28-й нс. встановлюється нове значення «0xF0» на шину даних «in», після чого на наступному такті, відбувається зсування усіх даних на одне слово і перше слово внутрішнього регістру пам’яті «mem» приймає значення «0xF0», після формуються значення на шині «out», при тому, останній байт в пам’яті стає першим в шині,у випадку 30-ї нс, слово «0xFF» яке є 16-м словом і займає першу позицію. На наступному такті відбувається повторення вищеописаних дій, але також відбувається формування даних на шині переносу, тобто на 32-й нс, відбувається видалення значення «0xFF» з регістру пам’яті і встановлення його на шину «carry». Так як робота даного блоку відповідає очікуваній, будемо

вважати, що він був успішно сформований і перейдемо до проектування наступного.

2.3 Блок вагового вікна.

Перш за все визначимо інтерфейс блоку. Аналогічно до попереднього модуля, проектований блок має мати наступні входи: «clk» - для такового сигналу і «rst» для скиду. Також так як цей модуль є обробкою вагового вікна то в нього має бути вхідна шина даних, значення якої будуть використовуватись для подальшої обробки, для цього в нас буде шина даних «values». Останнім, з вхідних сигналів, має бути сигнал стробування - «input_strobe» який буде запускати роботу схеми. Так як цей модуль має виконати обробку і видати оброблений сигнал, в нього має бути шина даних «out_values», на яку будуть потрапляти вже оброблені дані. Також в модуля має бути формування сигналу «out_strobe» який буде запускати роботу наступних блоків.

Перейдемо до проектування. Перш за все, даний блок має мати набір регістрів, що будуть відповідати за зберігання вхідних значень, що є відмасштабованими, значень коефіцієнтів певного вагового вікна, а також зберігати проміжні результати. Для обробки вікна, необхідно виконати над кожним значенням такі операції як перемноження і додавання, після чого отримані значення мають встановлюватись на шину «out_values». Також має бути присутнім блок, що через певну затримку буде формувати сигнал «out_strobe». І останнім має бути щось, що буде відповідати за скид даних що зберігаються в вищеописаних регістрах. Такий блок не є дуже складним, тому перейдемо до написання тестового модуля.

В загальному, так як для даного блоку ми маємо набір певних значень, що приходять на вхід, тому, сформуємо певний лічильник значення з якого будуть потрапляти на вхід модуля вікна. Так як більш нічого особливого для тестового

модуля не має бути – перейдемо до розгляду сформованих часових діаграм, що зображені на рисунку 2.3.

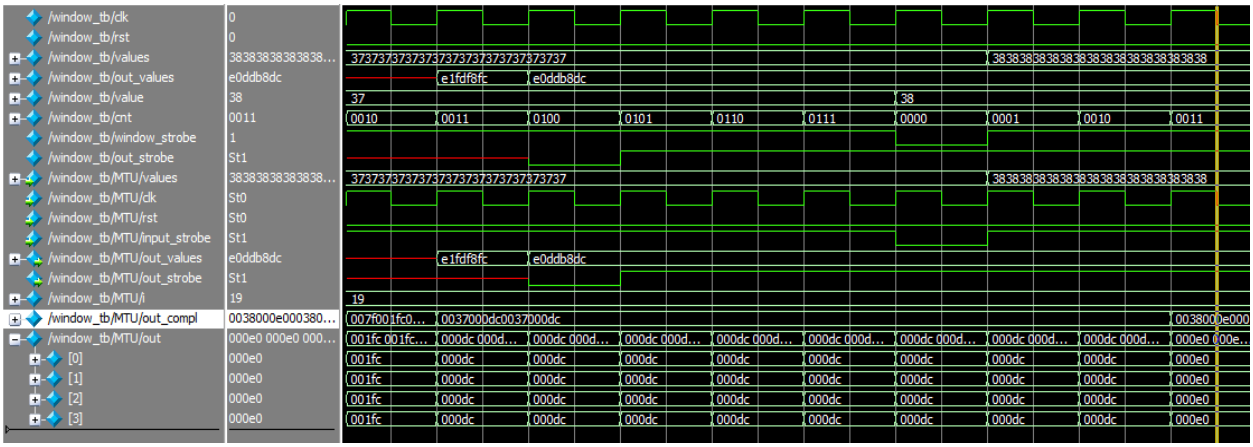

Рисунок 2.3 - Часова діаграма входів виходів блоку вагового вікна

На зображеній вище діаграмі можемо побачити, що сигнал із входу, потрапляє в пам’ять і після чого відбувається обробка, а саме перемноження і сумування. Далі сигнал потрапляє на шину «out_values», і формується сигнал «out_values». Так як робота даного блоку відповідає очікуваній,будемо вважати, що він був успішно сформований і перейдемо до проектування наступного.

2.4 Блок швидкого перетворення Фур’є

Слід уточнити що окреме проектування блоку Фур’є зазвичай розглядають в рамках окремих робіт так як ця тема є досить комплексною, в залежності від того, який алгоритм використовувати, яку архітектуру, який розмір вибірки – тощо.

У випадку якщо проектувати простий блок швидкого перетворення Фур’є за алгоритмом «Кюлі-Тукі» - в такого блока буде дуже багато одноманітних дій, які принципово нічим не відрізняються але при тому вимагають уваги і часу, наприклад якщо реалізовувати блок на 4096 вибірок, кількість коду, що необхідно буде написати і відлагодити буде складати більше як 300 сторінок або ж 70 тисяч стрічок коду. Тому будемо вважати проектування блоку швидкого перетворення недоцільним, і використаємо генератор блоків

швидкого перетворення Фур'є[6]. Оскільки використання генерованого коду для блоку не приносить нічого нового в роботу то ми не будемо розглядати його будову і взаємодію. Тому перейдемо до проектування наступного.

2.5 Блок обрахунку модуля комплексного числа

Аналогічно до процесу проектування попередніх блоків, почнемо проектування даного блоку з визначення його інтерфейсу. Для початку цей блок має мати такі службові порти як: «clk», «en» та «rst» ці порти мають працювати на вхід і відповідати за такі функції як: тактування модуля, ввімкнення модуля і його скид. Цілком очікувано, що подібно до минулих модулів даних модуль має мати вхід для сигналу стробування «input_strobe» та вихід сигналу сформованого сигналу стробування «mag_stb». Так як цей модуль має обраховувати модуль комплексного числа, при тому обраховувати для кількох значень, необхідно щоб він мав такі лінії як «i», «q» та «mag». Для початку розглянемо лінії «i», на цю лінію має потрапляти реальна частина комплексної складової, окрім того ця лінія має приймати кілька значень тому вона має бути, по-суті шиною. Лінія «q» - має відповідати за уявну частину комплексного числа, і також подібно до лінії - «i», вона має приймати кілька значень. Лінія «mag» - має працювати на вихід, і на ній мають з'являтись результат обробки а саме модуль комплексного числа, при тому, аналогічно до ліній «i» та «q» вона має приймати кілька значень.

Перейдемо до проектування модуля. Взагалі, задача обрахунку модуля комплексного числа є тривіальною в рамках математики, однак для програмованої логіки ситуація суттєво ускладняється так як обрахунок квадратного кореню вимагає або ж спеціалізованих блоків, які при тому збільшують час обрахунку, або ж якщо виконувати обрахунок методом LUT(Lookup table), то для одного такого модуля що має обраховувати необхідно буде використати велику кількість логікових комірок. Однак існують алгоритми які дозволяють спростити обрахунок ціною певної похибки в

результаті, одним з таких є так званий алгоритм «Альфа максимальне плюс бета мінімальне» або ж – «Alpha max plus beta min algorithm»[7]. Суть цього алгоритму в тому, що він дозволяє не використовувати обрахунок квадратного кореня, а замість цього перемножити максимальне і мінімальне значення з комплексного числа на певні константи і потім додати результат. Константи альфа і бета обираються в залежності від необхідних значень похибки. При тому є можливість підібрати такі параметри при яких множення не буде виконуватись взагалі[8], а буде виконуватись лише множення на 0.5 що еквівалентно зсуву на два в строну найменш значущого біта, для цілочисельних значень. Однак блок обрахунку модуля має також мати сутність що з певною затримкою буде формувати сигнал стробування. На цьому усе що необхідно врахувати для проектування даного блоку закінчується, тому перейдемо до безпосередньо написання тестового блоку.

В загальному, тестовий блок суттєво не відрізняється від вище спроектованих, він так само як і минулі має мати в собі пару певних лічильників, значення з яких мають потрапляти на входи блоку обрахунку модуля, також він має мати якусь затримку, яка буде визначати наскільки швидко мають змінюватись значення в лічильниках. Так як більше ніяких особливостей для тестового блоку немає, перейдемо до розгляду часових діаграм, що зображені на рисунку 2.4.

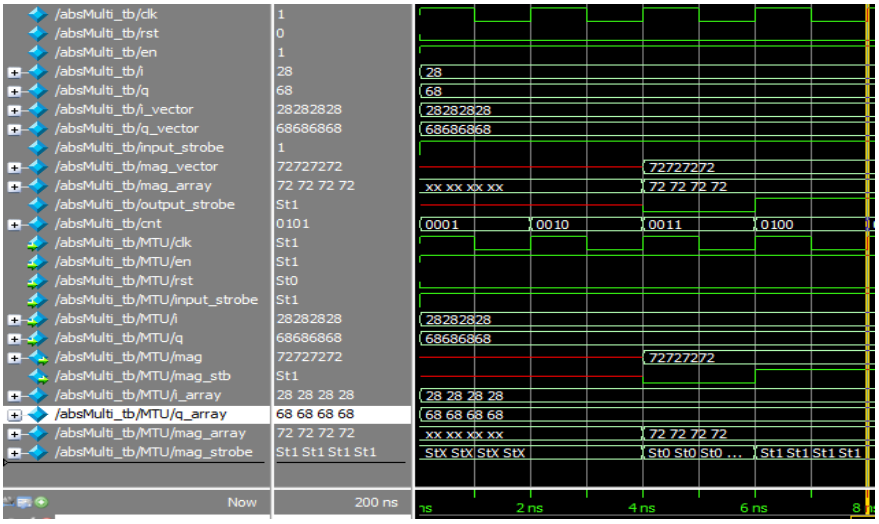

Рисунок 2.4 - Часова діаграма входів виходів блоку модуля комплексного числа

Наскільки можна побачити тестовим модулем формуються певні значення для реальної і уявної складових, наприклад для випадку, наведеного вище, реальна частина приймає значення – Re = 28, при тому як уявна – Im = 68, тоді згідно формулі обрахунку модуля будемо мати:

$$\sqrt{Re^2 + Im^2} = \sqrt{28^2 + 68^2} = 73.53 \approx 72 \quad (2.1)$$

Тобто отримане значення результату обрахунку модуля майже рівне точному значенню модуля, аналогічно і для інших значень. Спроектований блок працює як очікували, тому перейдемо до проектування наступного модуля.

2.6 Корелятор

Для початку проектування визначимося з інтерфейсом. Цей блок має корелювати два сигнали певної довжини, внаслідок чого в нього мають бути дві шини даних для першого сигналу і для другого однакової довжини, назвемо ці шини «a» та «b» як зрозуміло ці шини мають працювати на вхід, щоб отримувати значення з пам’яті або ж від корелятора. Цілком очікувано, що так як блок корелює два сигнали, то має також бути певний вихід на який буде потрапляти результат виконання обрахунку, назвемо цей вихід «rslt». Також для можливого каскадування блоків, було б корисно, якби був якийсь вхід що приймає попередній результат, нехай це буде вхід «prv_rslt». Так як більше функціональних виходів немає бути, то необхідно визначитись з службовими портами. Так як ніяких спеціальних дій окрім обрахунку кореляції блок не має робити, то додамо лише ті порти, що додавали зазвичай, а саме: скид - «rst» ; тактовний вхід - «clk»; вхідний сигнал сторобування - «input_strobe»; та сформований сигнал стробування – «strobe»

Проектування даного модуля, в цілому не має бути складним так як ми будемо використовувати кореляцію описану в минулому розділі. В загальному

алгоритм роботи блоку має бути наступним: спочатку відбувається читання попереднього результату виконання кореляції, після чого для кожної пари значень вибірок виконується віднімання і обрахунок модуля від результату, після чого результати мають бути підсумованими із попереднім результатом, і отриманий результат має з’явитись на шині «rslt». Окрім того, має також бути якийсь компонент, яка виконає певну затримку і сформує сигнал стробування. Так як більше жодних особливостей в проектуванні даного блоку немає, то перейдемо до проектування тестового блоку. Для прикладу, код цього модуля наведений в додатку Б.

Загалом тестовий блок має згенерувати певний набір даних та відправити його на модуль кореляції, після чого від нього має прийти певна відповідь, загалом цей модуль подібний до минулого тестового модуля, тому перейдемо до розгляду часових діаграм, що зображені на рисунку 2.5.

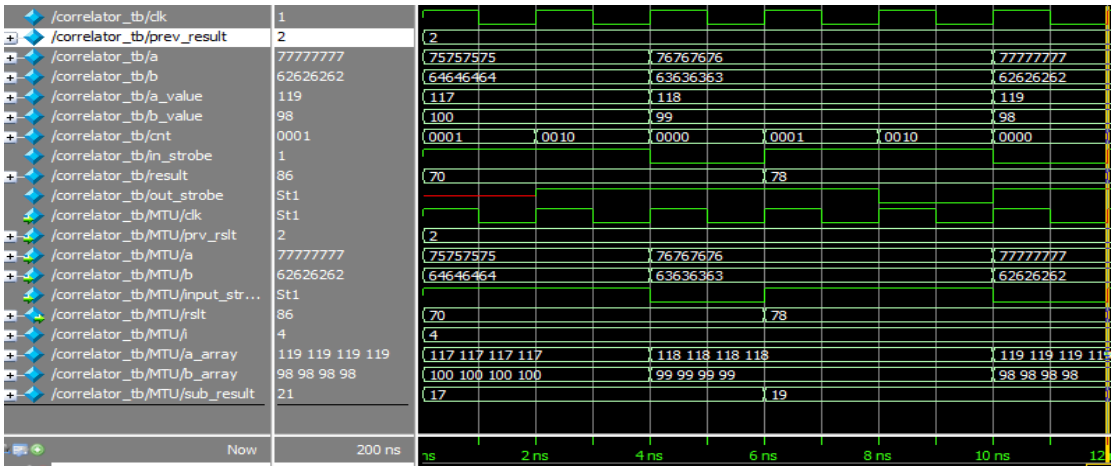

Рисунок 2.5 - Часова діаграма входів виходів блоку корелятора

Почати розгляд даної часової діаграми слід із того, що визначити, що відбувається на вході модуля, а саме на входах «a» та «b», на початку симуляції на них потрапляють початкові значення 117 та 100, після чого відбувається віднімання одного значення від іншого для усіх чотирьох пар і підсумовування з попереднім результатом, який отримується з лінії «prec_rslt» результатом цих дій є значення 70 (2.2), що відповідає очікуваному.

$$\sum_{i=0}^{3}|a_i - b_i| + prev_{rslt} = |117 - 100| + |117 - 100| + |117 - 100| + |117 - 100| + 2 = 70 \quad (2.2)$$

Так як робота модуля відповідає необхідній перейдемо до моделювання наступного блоку.

2.7 Блок пам’яті шаблонного сигналу

Так як шаблонний сигнал повинен зберігатись то він має знаходитись на певній шині даних, зазвичай такі шини є двонаправленими і під час запису працюють на вхід, в інший час – на вихід, тому даний блок має мати двонаправлений порт даних, назвемо його «mem_bus». Також для запису необхідно мати певний сигнал, який буде вказувати що значення з шини даних необхідно записати, нехай це буде лінія – «write», що буде працювати на вхід. Однак ще необхідно додати службові входи, до них відносять входи для тактовного сигналу і для скиду – назвемо їх «clk» і «rst».

Перейдемо до проектування пристрою. В загальному так як даний пристрій повинен зберігати інформацію, то це визначає, що він буде мати набір регістрів всередині для зберігання інформації. Слід зауважити що зазвичай фіксація значень з шини відбувається динамічно, по передньому або ж задньому фронту імпульсу, це дозволяє точно зафіксувати стан шини, і в свою чергу підвищує коректність роботи і швидкодію, тому слід використати аналогічний спосіб для запису. Іншим важливим моментом є те, що так як шина пам’яті має бути двонаправленою необхідно використати три станові буфери, що дозволяють перевести лінію в стан високого опору, інакше на лінії можуть виникнути конфлікти що погано вплине на пристрій. Так як більше особливостей в проектуванні даного блоку немає, перейдемо до написання тестового блоку.

Для того щоб протестувати блок пам’яті необхідно перш за все сформувати дані, що будуть зберігатись у пам’яті, після чого записати їх.

Основну увагу в цьому треба надати двом моментам, по-перше, взаємодії з двонаправленою шиною, по-друге перевірці зберігання значень. Розглянемо їх окремо, стосовно взаємодії із двонаправленою шиною, перш за все, необхідно створити тристанові буфери, що будуть переходити в стан високого опору, коли сигнал запису буде змінювати стан на логікові одиницю. Стосовно перевірки, якщо ж встановити лінію зв'язку пам'яті з тестовим модулем в режим високого опору, тоді зміни на шині пам'яті будуть показувати дані, що були збережені, що надасть змогу перевірити можливість зберігати дані. Так як більше особливостей в даного блоку немає, перейдемо до розгляду часових діаграм, що зображені на рисунку 2.6.

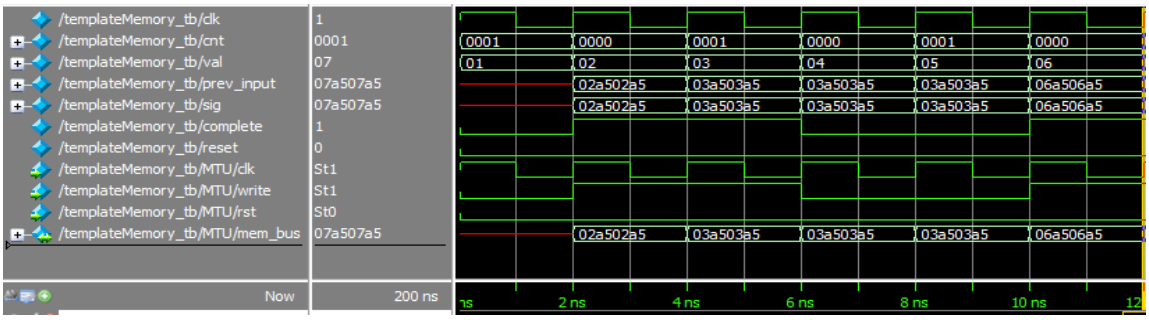

Рисунок 2.6 - Часова діаграма входів виходів блоку пам'яті шаблонного сигналу

Розглянемо дану часову діаграму, на початку стан шини «mem_bus» - невідомий, однак на наступному такті, друга нс, формується сигнал на лінії «sig» що складається з константних значень 0xA5 та значення змінної «val» – 0x07. Так як сигнал тестовим модулем на шину «sig» виводиться лише при логіковій одиниці на лінії «complete», то на проміжку від 2-ї нс до 6-ї нс відбувається зміна значень. Після 6-ї нс відбувається запис даних по зрізу модулем пам'яті і встановлення «sig» - і як можемо побачити він не змінюється протягом наступних 4-х нс секунд, що говорить що модуль пам'яті працює коректно, тому перейдемо до збору модуля розпізнавання звуку .

2.8 Проектування модуля розпізнавання звуку.

Для початку розробки повного модуля визначимось з інтерфейсом даного модуля. По-перше пристрій для розпізнавання звуку має мати певний вхід на який буде потрапляти сигнал двійкового потоку даних, назвемо його «in_byte_stream». По-друге він має мати вихід, що буде відповідати за те, чи був сигнал розпізнаний, чи не був, назвемо його – « is_recognized». По-третє, пристрій для розпізнавання звуку, також має мати вхід, для запису сигналу – «write». І останнє – пристрій має мати такі службові виходи як «clk» і «rst» які будуть відповідати за тактування та скид.

Перейдемо до проектування блоку. Для початку нагадаємо, що під час проектування майже для усіх модулів були додані сигнали стробування що генерувались в залежності від отриманого сигналу на вхід на основі них і будемо розглядати і проектувати роботу модуля. Перевагами цього є спрощення проектування а також більш передбачувана поведінка модуля внаслідок того, що початок і кінець роботи відбувається в певні, строго детерміновані, моменти часу. Тому будемо використовувати такий підхід для активації внутрішніх блоків.

Розглянемо принцип обробки сигналу. Бінарний сигнал потрапляє на блок перетворення двійкового коду в слово, після чого відбувається перетворення і отримане слово надходить на регістр зсуву. Після цього дані з регістру зсуву мають потрапити на модуль вагового вікна, де відбудеться їх масштабування і сумування. Після того, підсумовані дані мають потрапити на блок обрахунку модуля комплексного числа, обраховані модулі – потрапляють на корелятор, куди також, має потрапити сигнал з шаблонного блоку. Після потрапляння сигналів на корелятор відбувається обрахунок кореляції і результат має потрапити на пороговий пристрій. В цілому, така схема має бути для пристрою в загальному, однак слід зауважити, що вище описані блоки мають можливість каскадування, в наслідок цього можна додати кілька додаткових каскадів для

забезпечення необхідної довжини перетворення. Таке з’єднання немає особливостей, тому не будемо її зараз описувати.

Так як ми визначились із проектуванням основного модуля, перейдемо до проектування тестового модуля. Для початку визначимось, з функціями тестового модуля. Тестовий модуль має генерувати певні значення, ці значення по біту мають потрапляти на вхід пристрою для розпізнавання. Також даний тестовий модуль має генерувати тактовий сигнал. Так як більше особливостей немає, перейдемо до розгляду спроектованого модуля.

Розгляд спроектованого модуля має сенс почати окремо з розгляду розповсюдження сигналів керування і активації модулів, і окремо передачу даних між модулями і їх обробку

Для початку проектування розглянемо як мають поширюватись сигнали керування.

Так як даний модуль є синхронним то усі модулі мають сигнали тактування, особливої ролі для процесу керування вони не мають, так як в нас використовується лише один сигнал тактування, і якщо спростити, то він відповідає за одночасне вмикання усіх модулів, тому ми його не будемо зараз розглядати.

Почнемо розгляд сигналів керування, наведених на рисунку 2.7.

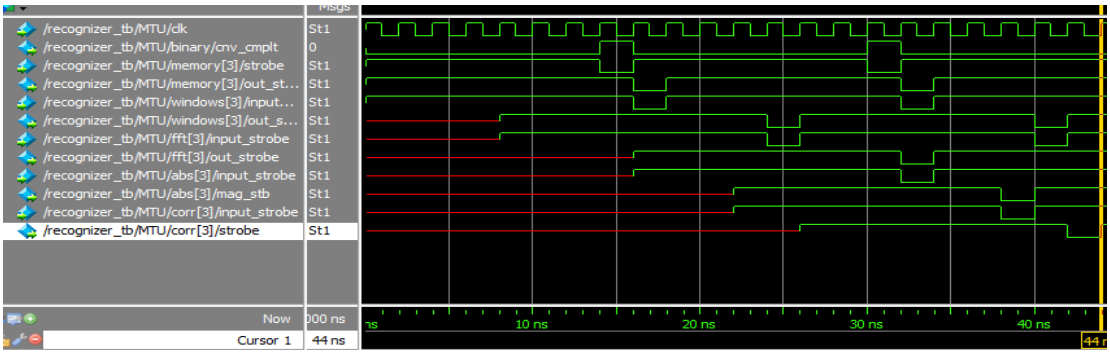

Рисунок 2.7 - Часова діаграма сигналів стробування для модуля розпізнавання сигналу

Сигнали керування починаються з конвертеру двійкового коду в слово, так як даний конвертер відповідає за перетворення сигналу, він має лічильник, який при закінченні перетворенні видає на порт «cnv_cmplt» прямокутний імпульс.

Далі утворений імпульс поступає на вхід «strobe» регістру зсуву, що в свою чергу генерує вихідний сигнал стробування «out_strobe». Цей сигнал від регістра зсуву потрапляє на вхід вагового вікна «input_strobe», що в свою чергу запускає перетворення і по його закінченню генерує сигнал стробування на порт «out_strobe» модуля вікна. Сигнал з модуля вікна потрапляє на вхід «input_strobe» Блока ШПФ, запускає перетворення Фур'є після його закінчення генерується вихідний сигнал на порт «out_strobe». Сигнал з порту «out_strobe», блоку швидкого перетворення Фур'є потрапляє на блок обрахунку модуля комплексного числа, на порт «input_strobe», що в свою чергу запускає обробку обрахунку модуля і генерує вихідний сигнал стробування «mag_stb». Цей сигнал потрапляє на блок кореляції на вхід «input_strobe» - що запускає обробку кореляції двох спектральних вибірок, а також формує вихідний сигнал стробування. Описаний вище процес можемо спостерігати на рисунку 2.7. Як можна побачити, час перетворення складає 30 нс при тактовій частоті – 1 нс, що відповідає затримці в 30 тактів, між приходом нового відліку і повним виконанням розпізнавання сигналу.

Наступним кроком розглянемо передачу даних між модулями і їх обрахунок, для цього розглянемо часову діаграму наведену на рисунку 2.8.

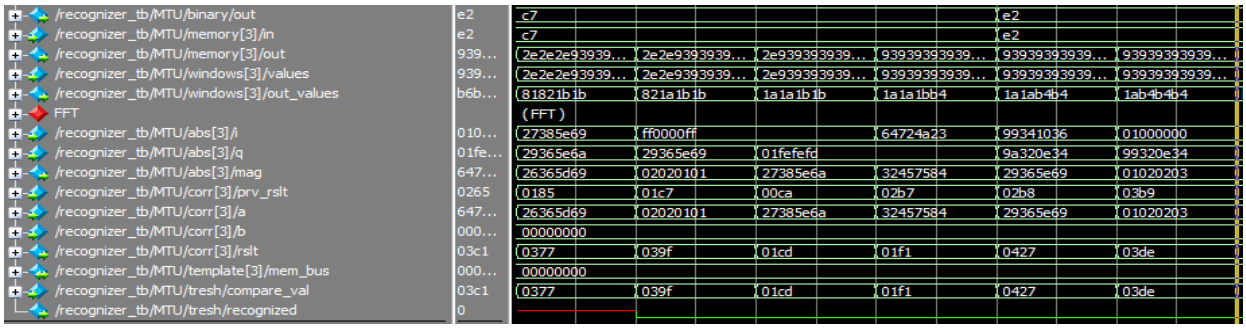

Рисунок 2.8 - Часова діаграма сигналів даних для модуля розпізнавання сигналу

Спочатку сигнал потрапляє з входу «in_byte_stream» на конвертер двійкового коду в слово після чого відбувається перетворення, і перетворене слово вставляється на шину «out» конвертера двійкового коду. В свою чергу дані шини «out» потрапляють на регістра зсуву де відбувається їх зберігання.

Дані з регістра зсуву виставляються на шину «out» регістру зсуву, після чого вони потрапляють на блок вагового вікна на шину «values». Після чого відбувається їх перемноження на коефіцієнти вагового вікна і отримані дані потрапляють на шину «out_values». Дані шини «out_values» потрапляють на блок ШПФ на шину «X», після чого на шину «Y» виставляється актуальний результат. Дані з шини «Y» потрапляють на блок розрахунку модуля комплексного числа на шину «i», на шину «q» - подаються нулі. Після обрахунку модуля на шину «mag» виставляється актуальне значення. Дані з шини «mag» потрапляють на шину «a» корелятора, а на шину «b» потрапляють дані з блоку шаблонного сигналу. Далі, внаслідок обробки корелятором, на шині «rslt»- з'являється результат оброки. Результат з шини «rslt» потрапляє на пороговий пристрій, який вирішує чи був сигнал розпізнаний чи не був.

Висновок

Під час виконання роботи було сформовано алгоритм розпізнавання звуку, а також сформульовані основні принципи роботи функціональних блоків пристрою для розпізнавання. Також було спроектовано окремі апаратні блоки мовою Verilog для пристрою розпізнавання звуку. Спроектоване рішення, може бути як і окремим пристроєм, так і частиною більш складної системи. Спроектований пристрій має ряд особливостей.

Перша особливість - час перетворення є константним. Це можливо внаслідок того, що складність алгоритмів, що використовуються, є константною, відповідно і кількість тактів, що необхідна для повного циклу розпізнавання є сталою і складає 30 тактів, тобто можна визначати необхідну мінімальну частоту тактового сигналу для необхідної кількості перетворень яка має вже визначатись в залежності від технічного завдання.

Друга особливість - ітеративна структура.

Третя особливість - використання швидкого алгоритму знаходження модуля комплексного числа.

В цілому можна сказати що робота була виконана успішно, результатом її є програмне ядро для ПЛІС що, можна використати окремо в якості пристрою для розпізнавання звуку на ПЛІС. Дане ядро працює з аудіо сигналами при тому в залежності від частоти тактування і частоти дискретизації дане ядро може працювати фактично в режимі реального часу. Наприклад для розпізнавання сигналу з частотою дискретизації 44.1 кГц необхідно використати тактовий сигнал з частотою в 2.7 МГц що не є великою проблемою для сучасних засобів техніки.

Список використаних джерел

1.Sound recognition [Wikipedia] [Електронний ресурс] URL:https://en.wikipedia.org/wiki/Sound_recognition

2. Fourier transform [Wikipedia] [Електронний ресурс] URL: https://en.wikipedia.org/wiki/Fourier_transform

3. Discrete Fourier transform [Електронний ресурс] URL: https://en.wikipedia.org/wiki/Discrete_Fourier_transform

4. Полифазное БПФ [DspLib] [Електронний ресурс] URL: http://www.dsplib.ru/content/polyphasefft/polyphase.html

5. Дэвид М.Харрис и Сара Л.Харрис (2015). Цифровая схемотехника и архитектура компьютера: Учеб. Пособие: Morgan Kaufman

6. DFT/FFT IP Core Generator [Spiral] [Електронний ресурс] URL: https://www.spiral.net/hardware/dftgen.html

7. Alpha max plus beta min algorithm [Wikipedia] [Електронний ресурс] URL: https://en.wikipedia.org/wiki/Alpha_max_plus_beta_min_algorithm

8. Digital Signal Processing Tricks – High-speed vector magnitude approximation [embedded]] [Електронний ресурс] URL: https://www.embedded.com/digital-signal-processing-tricks-high-speed-vector-magnitude-approximation/

Анотація

Розпізнавання звуку є актуальною темою на даний момент у зв'язку з процесом "цифровізації" що зараз спостерігається. Однак на даний момент спостерігається брак апаратних рішень для розпізнавання звуку, при тому, що наявні системи або ж розпізнають лише голос, або працюють на базі телефонів і комп’ютерів, через це виникає ідея у створенні апаратної реалізації системи для розпізнавання звуку.

Основною метою було спроектувати систему, яка в майбутньому може бути використана для розпізнавання звуку або ж бути фундаментом для більш складних систем розпізнавання. При тому, основна увага приділялась формулювання основних принципів за якими можуть працювати системи розпізнавання сигналу на базі програмованої логіки.

Завданням даної роботи є реалізація модульної структури для розпізнавання сигналу(звуку) на базі ПЛІС в режимі реального часу. Готове рішення має мати можливість як бути частиною більш складної системи, так і окремим пристроєм.

Проектування відбувалось на базі симулятора мов опису апаратури ModelSim.

У роботі запропоновано алгоритм та схемотехнічне рішення які потенційно можуть допомогти для розпізнавання сигналів. При написанні роботи основну увагу було зосереджено на формуванні принципів розпізнавання звуку або іншого довільного сигналу і принципів функціонування окремих блоків пристрою.

ДОДАТКИ

Додаток А
Код програми розпізнавання сигналу на основі мови MATLAB

```
% reading audio file
[y,fs] = audioread('1.mp3');
y = sum(y,2);
Y = fft(y);


soundTemplate = createSoundTemplate(y, fs, 100,5);
correlRes = zeros(175);

for i = 1:175
    correlRes(i) = correlateSoundWTemplate(y,
fs,i,5,soundTemplate);
end

plot(1:175,correlRes)

% Correlate spectrum image with template sound, value will be lo
function result = correlateSoundWTemplate(sound, quant, start, n,
template)
    result = 0;
    for i = 1:n
        quanOfSound = sound((quant * (start + (n-1))) : (quant *
(start + n)));
        quantFFT = transpose(fft(quanOfSound));
        quantFFT = abs(quantFFT);
        result = result + sum(abs(template(i,:) - quantFFT));
    end
end

% Getting spectrum image for n second
function template = createSoundTemplate(sound, quant, start, n)
    template = zeros(n,quant+1);

    for i = 1:n
        quanOfSound = sound((quant * (start + (n-1))) : (quant *
(start + n)));
        quantFFT = transpose(fft(quanOfSound));
        quantFFT = abs(quantFFT);
        template(i,:) = quantFFT;
    end
end
```

Додаток Б

Модуль блоку корелятора

Текст програми:

```
module correlator(input clk,
    input rst,
    input [12:0] prv_rslt,
    input [31:0]a,
    input [31:0]b,
    input input_strobe,
    output reg [12:0]rslt,
    output strobe
    );

    integer i;

    wire [7:0] a_array [0:3];
    wire [7:0] b_array [0:3];
    reg [7:0] sub_result;

    assign {a_array[3],a_array[2],a_array[1],a_array[0]} = a;
    assign {b_array[3],b_array[2],b_array[1],b_array[0]} = b;

    delayT #(.DATA_WIDTH(1), .DELAY(2)) stb_delay_inst (
    .clock(clk),
    .reset(rst),
    .data_in(input_strobe),
    .data_out(strobe)
    );

    always @(posedge clk or posedge input_strobe)
        begin
        rslt = prv_rslt;
            for(i = 0; i<= 3; i = i+1)
            begin
                sub_result = (a_array[i] > b_array[i]) ? (a_array[i] - b_array[i]) : (b_array[i] -
a_array[i]);
                rslt = rslt + sub_result;
            end
        end
endmodule
```