УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ

Марко Поповић

Имплементација аудио апликације за DSP у C – у уз ослањање на језичка проширења за наменске процесоре

ДИПЛОМСКИ РАД
- Основне академске студије -

Нови Сад, (2015)

Редни број, **РБР**:		34	
Идентификациони број, **ИБР**:			
Тип документације, **ТД**:		Монографска документација	
Тип записа, **ТЗ**:		Текстуални штампани материјал	
Врста рада, **ВР**:		Завршни (Bachelor) рад	
Аутор, **АУ**:		Марко Поповић	
Ментор, **МН**:		Проф. др. Јелена Ковачевић	
Наслов рада, **НР**:		Имплементација аудио апликације за DSP у C – у уз ослањање на језичка проширења за наменске процесоре	
Језик публикације, **ЈП**:		Српски / ћирилица	
Језик извода, **ЈИ**:		Српски	
Земља публиковања, **ЗП**:		Република Србија	
Уже географско подручје, **УГП**:		Војводина	
Година, **ГО**:		2015	
Издавач, **ИЗ**:		Ауторски репринт	
Место и адреса, **МА**:		Нови Сад; трг Доситеја Обрадовића 6	
Физички опис рада, **ФО**: (поглавља/страна/ цитата/табела/слика/графика/прилога)		7/26/1/1/4/0/0	
Научна област, **НО**:		Електротехника и рачунарство	
Научна дисциплина, **НД**:		Рачунарска техника	
Предметна одредница/Кqучне речи, **ПО**:		DSP, програмски језик C, програмско окружење CLIDE, Visual Studio 2005, 2013, асемблер, аудио обрада	
УДК			
Чува се, **ЧУ**:		У библиотеци Факултета техничких наука, Нови Сад	
Важна напомена, **ВН**:			
Извод, **ИЗ**:		У овом раду је приказана имплементација „Spectral Subtraction noise suppression“ аудио апликације DSP фирме Cirrus Logic. Имплементација је у C и асемблер програмским језицима. Кроз тај процес је требало упознати се са употребљивошћу C - а и његовог проширења за писање аудио апликација на наменским процесорима. Требало је упознати се са постпуком развоја DSP апликација као и са методана испитивања и верификовања DSP апликација. Рад се ослања на CLIDE интегрисано окружење и Visual Studio.	
Датум прихватања теме, **ДП**:			
Датум одбране, **ДО**:			
Чланови комисије, **КО**:	Председник:	Проф. др. Мирослав Поповић	
	Члан:	Доц. др. Иштван Пап	Потпис ментора
	Члан, ментор:	Доц. др. Јелена Ковачевић	

Accession number, **ANO**:			
Identification number, **INO**:			
Document type, **DT**:		Monographic publication	
Type of record, **TR**:		Textual printed material	
Contents code, **CC**:		Bachelor Thesis	
Author, **AU**:		Marko Popovic	
Mentor, **MN**:		Prof. PhD. Jelena Kovacevic	
Title, **TI**:		Implementation of an audio application for a DSP in C, with the support of programming language extensions for embeeded processors	
Language of text, **LT**:		Serbian	
Language of abstract, **LA**:		Serbian	
Country of publication, **CP**:		Republic of Serbia	
Locality of publication, **LP**:		Vojvodina	
Publication year, **PY**:		2015	
Publisher, **PB**:		Author's reprint	
Publication place, **PP**:		Novi Sad, Dositeja Obradovica sq. 6	
Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes)		7/26/1/1/4/0/0	
Scientific field, **SF**:		Electrical Engineering	
Scientific discipline, **SD**:		Computer Engineering, Engineering of Computer Based Systems	
Subject/Key words, **S**/**KW**:		DSP, C, Cirrus Integrated Development Environment (CLIDE), Visual Studio 2005, 2013, assembler, assembly, audio processing	
U, audC			
Holding data, **HD**:		The Library of Faculty of Technical Sciences, Novi Sad, Serbia	
Note, **N**:			
Abstract, **AB**:		This Bachelor's Thesis is an implementation of the Spectral Subtraction – Noise Suppression algorithm for an DSP company Cirrus Logic. The implementation was done in the programming languages C and assembler. Throughout that process the student was supposed to identify himself with the features of the C programming language and it's extensions for embeeded processors. The student was supposed to identify himself with the development methods for DSP applications, with the methods of testing and verification, as well. The Thesis is based on programming in Visual Studio and in the Cirrus Integrated Development Environment (CLIDE).	
Accepted by the Scientific Board on, **ASB**:			
Defended on, **DE**:			
Defended Board, **DB**:	President:	Prof. PhD. Miroslav Popovic	
	Member:	Doc. PhD. Istvan Papp	Menthor's sign
	Member, Mentor:	Doc. PhD. Jelena Kovacevic	

Захвалност

Захвалност дугујем својој породици и пријатељима, као и ментору и колегама. Без вас ово све не би било могуће.

Садржај

Списак слика

Списак табела

Скраћенице

DSP - Дигитални Сигнал Процесор

CLIDE - Развојно окружење фирме *Cirrus Logic*

FFT - Брза *Фуриеова трансформација*

IFFT - Брза инверзна *Фуриеова трансформација*

SS Processing - *Spectral Subtraction* processing

1. Увод

Овог дипломског рада је био циљ да се имплементира Spectral Subtraction - Noise Suppression алгоритам у програмском огружењу CLIDE, уз консултацију референтног кода у Visual Studio. Циљна архитектура је била Cirrus-ов Coyote 32 DSP чији рад је CLIDE у могућности да симулира.

Сам алгоритам математички ради одузимање снаге естимиране буке у околини, са снагом улазног звука и тиме добијамо само најгласнији звук у околини, обично онај који се и жели пренети. Примера ради, у звуку одређене песме постоји шум, алгоритам отклања овај шум да би се чула само песма. Или када обрађујемо глас са буком у просторији, на крају добијамо само глас.

1.1 Структура рада

У другом поглављу су дате теоријске основе које су биле неопходне за овај рад. У њему су описани алгоритам саме апликације, наменски системи и развој на њима, као и компајлери и њихов однос са интегрисаним окружењима.

У трећем поглављу је описано програмско окружење CLIDE, неке специфичности Coyote 32 DSP процесора, као и типови који подржавају рад са непокретним зарезом у CLIDE-у.

У четвртом поглављу је описан концепт решења датог проблема, као и главни проблеми при имплементацији и верификацији са којима смо се сусретали.

У петом поглављу је описана сама имплементација апликације, где су и описани модули: main, ss_processing, signals, defines, windows, libIFFT, libMathDiv, libMathLog10 и libMathSqrt.

У шестом поглављу је описано тестирање и валидација имплементираног решења.

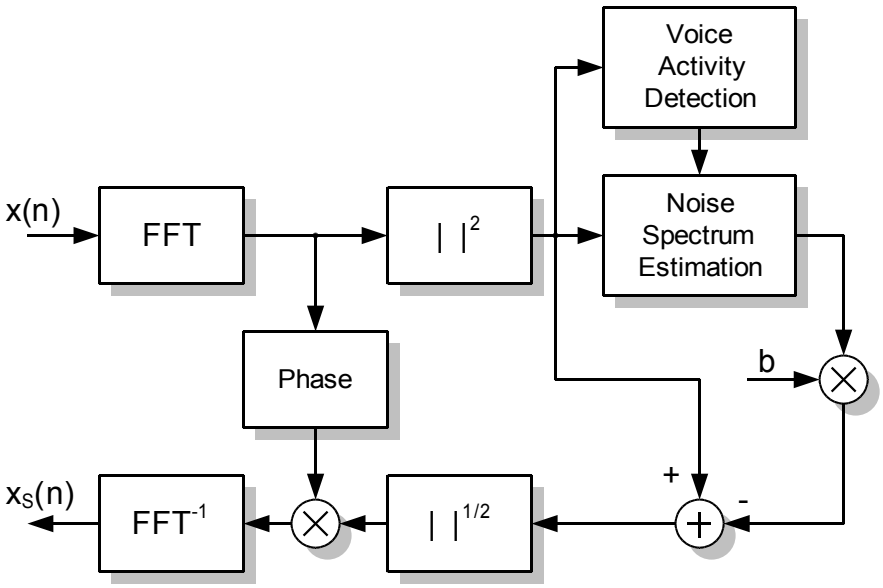
У седмом поглављу је дат закључак овог дипломског рада.

2. Теоријске основе

2.1 Алгоритам апликације

Алгоритам који је било требало имплементирати се назива Spectral Subtraction, Noise Suppression.

Базиран је на прозору од 64 одбирака, који се преклапају са претходним прозором у обради, тако да се обрада врши на 128 одбирака.



Слика 2.1 - Приказ Spectral Subtraction - Noise Suppression алгоритма

У почетку се улаз у алгоритам прозорира по следећим формулама:

$$|X'(k,m)| = 0.9|X'(k,m-1)| + 0.1|X(k,m)|$$

$$|X''(k,m)| = 0.48|X'(k,m)| + 0.38|X'(k-1,m)| + 0.14|X'(k-2,m)|$$

Потом се прорачуни излазних одбирака обављају у спектралном домену, у који се прелази *Дискретном Фуриеовом трансформацијом (DFT)* која се рачуна алгоритмом *Брзе Фуриеове трансформације (FFT)*.

$$X(k,m) = \sum_{n=0}^{N-1} x\left(m\frac{N}{2} + n\right) w(n) e^{-j\frac{2\pi kn}{N}}$$

Најважнија је, у ствари, естимација снаге тј. у жаргону речено, енергије буке и прорачуни кеофицијента β, који се касније одузима од оригиналног спектра звука да бисмо добили излазни спектар. β–у добијамо на местима где је снага звука мала.

$$P_{nfloor}(k,m) = \begin{cases} 0.998P_{nfloor}(k,m-1) + 0.002|X''(k,m)|^2 & \text{if } |X''(k,m)|^2 > P_{nfloor}(k,m) \\ 0.93P_{nfloor}(k,m-1) + 0.07|X''(k,m)|^2 & \text{else} \end{cases}$$

$$vad = \sum_{k=0}^{N-1} \frac{|X''(k,m)|^2}{P_{nfloor}(k,m)} - \log_{10} \frac{|X''(k,m)|^2}{P_{nfloor}(k,m)}$$

$$P_n(k,m) = 0.9P_n(k,m-1) + 0.1|X''(k,m)|^2$$

$$|X_S(k,m)|^2 = \max(|X(k,m)|^2 - \beta(k)P_n(k,m), 0)$$

Након тога, остаје да се дати одбирци коренују, како бисмо добили излазне вредности, а не излазне модуле:

$$X_S(k,m) = \sqrt{|X_S(k,m)|^2}\arg X(k,m)$$

На крају се врши *Инверзна Фуриеова трансформација (IFFT)* и добијамо резултат.

2.2 Наменски системи и развој на њима

Наменски системи, или како се на енглеском називају embedded systems су у поређењу са системима опште намене мање величине, мање потрошње, мање коштају, али су им и перформансе лошије. Њихова примена је широка, као нпр. у телефонији, телевизији, аутомобилској индустрији итд.

Најчешће их карактерише посебан тип процесора, који у себи садржи уграђену меморију и неке периферне јединице. Међутим, ово не мора бити случај, уграђени системи сами могу имати своју меморију и периферне јединице, са наменским процесором.

Ови системи не морају имати уопште никакву спољашњу спрегу, али је такође могуће чак да имају и неку једноставну, графичку корисничку спрегу.

Писање програма на оваквим процесорима се најчешће базира на нижим програмским језицима као што су C, асемблер итд. Само писање кода уме бити изузетно ограничено јер се ради о окружењима која не раде са стандардним типовима променљивих, а самим тим нити са стандардним улазно-излазним заглављима. Из тог разлога, веома често сам програм морамо да прилагођујемо датом интегрисаном окружењу.

Битно је напоменути да се у овим случајевима често ради и на симулатору одређеног уређаја (нпр. на симулатору андроида или на симулатору DSP-а) зато што је тестирање и дебаговање на окружењу са мало ресурса веома успорено и непрактично.

Међутим, тестирање и дебаговање на симулатору може да се разликује од оног на самом уређају исто из разлога што тестирање и дебаговање може заузимати више ресурса него што тај уређај поседује.

Из свега наведеног може се закључити да је развој на наменским системима изузетно захтеван и потребује велику праксу и рутину при програмирању.

2.3 Компајлери и интегрисана окружења

Компајлери су посебни програми који преводе програме написане у одређеном вишем програмском језику у асемблерски код. Уз њих постоје и асемблери који преводе асемблерске кодове у објектне датотеке. Линкери спајају све објектне датотеке у један, извршни код који се може покренути. Најчешће су сви ови програми део неког интегрисаног окружења за развој програма.

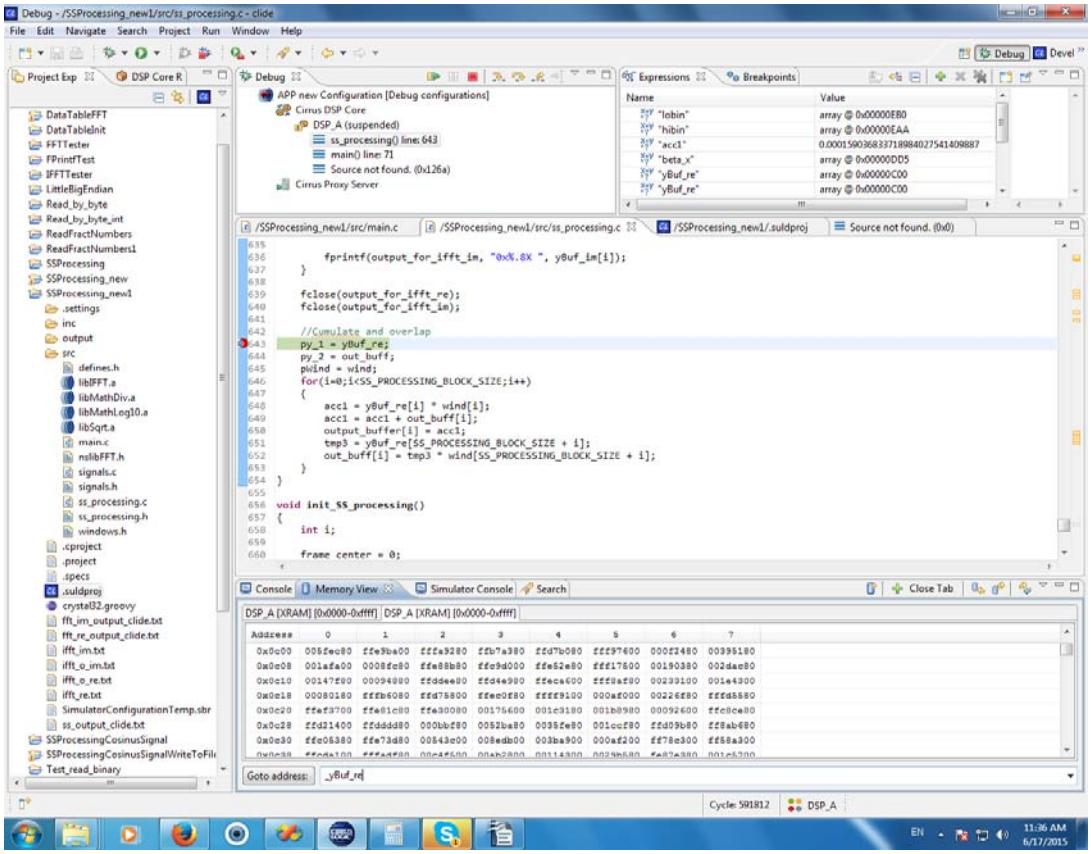
Компајлери за наменске процесоре имају своја проширења за њих. Најчешће ови компајлери преводе одређена проширења стандардних програмских језика и прилагођени су да раде за циљну платформу.

Нпр. неки систем за дигиталну обраду сигнала може захтевати да његов компајлер уме да преводи одређене нове типове променљивих, са којима овај систем много лакше

функционише. Такође, може се нпр. очекивати да овакав компајер уме да користи одређене оптимизационе технике како би побољшао извршење програма.

3. Програмско окружење CLIDE и рад са конкретним наменским процесором

Програмско окружење CLIDE у коме је писана имплементација датог Spectral Subtraction – Noise Suppressionалгоритма је развијено у научно–истраживачком институту РТ–РК и коришћено је за DSP-ове компаније Cirrus Logic.



Слика 3.1 – Програмско окружење CLIDE

Током рада у датом програмском окружењу, откривене су и одређене грешке и недостаци у програмским библиотекама који до тада нису уочени. Проблеми се могу сврстати у две групе:

- Проблеми везани за рад са датотекама и
- Проблеми везани за недостатак одговарајућих функција за одређени тип променљивих

Проблеми за рад са датотекама су сусретани на сваком кораку, од тога да датотека не може да се направи или отвори ако јој је име предугачко, до потпуне немогућности и блокаде рада са бинарним датотекама.

Међутим, проблеме за недостатак одговарајућих функција за одређени тип су је било могуће решити, тако што су коришћене одговарајуће асемблерске функције за дате типове. Најчешће су то били случајеви да нам је била потребна функција која ради са типом accum (тј. са 64-обитном прецизношћу), а у библиотеци је дата функција била имплементирана да ради са fract–ом (тј. са 32-обитном прецизношћу).

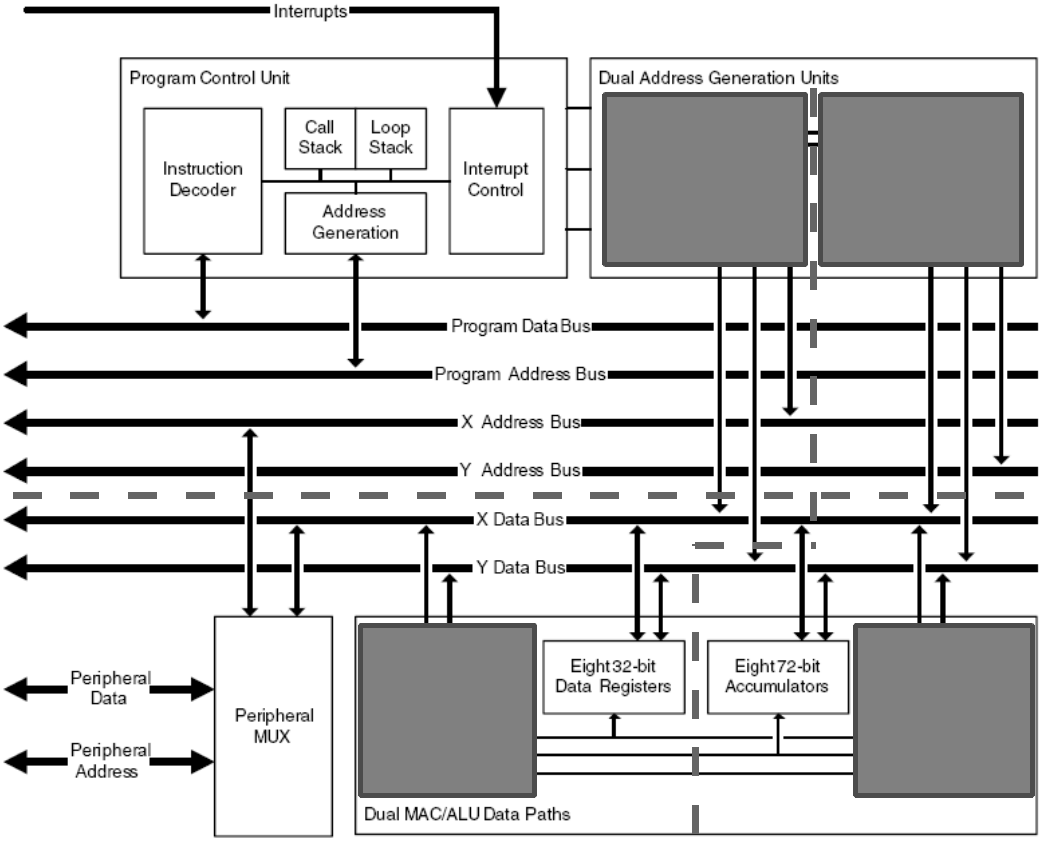
3.1 Неке специфичности процесора Cirrus DSP Coyote 32

Процесор Cirrus DSP Coyote 32 је типичан процесор за дигиталну обраду сигнала. Поменути процесор је у суштини Харвард архитектура. Своју меморију дели на X, Y и P део, где се X и Y меморија користе за податке, а P је програмска меморија.

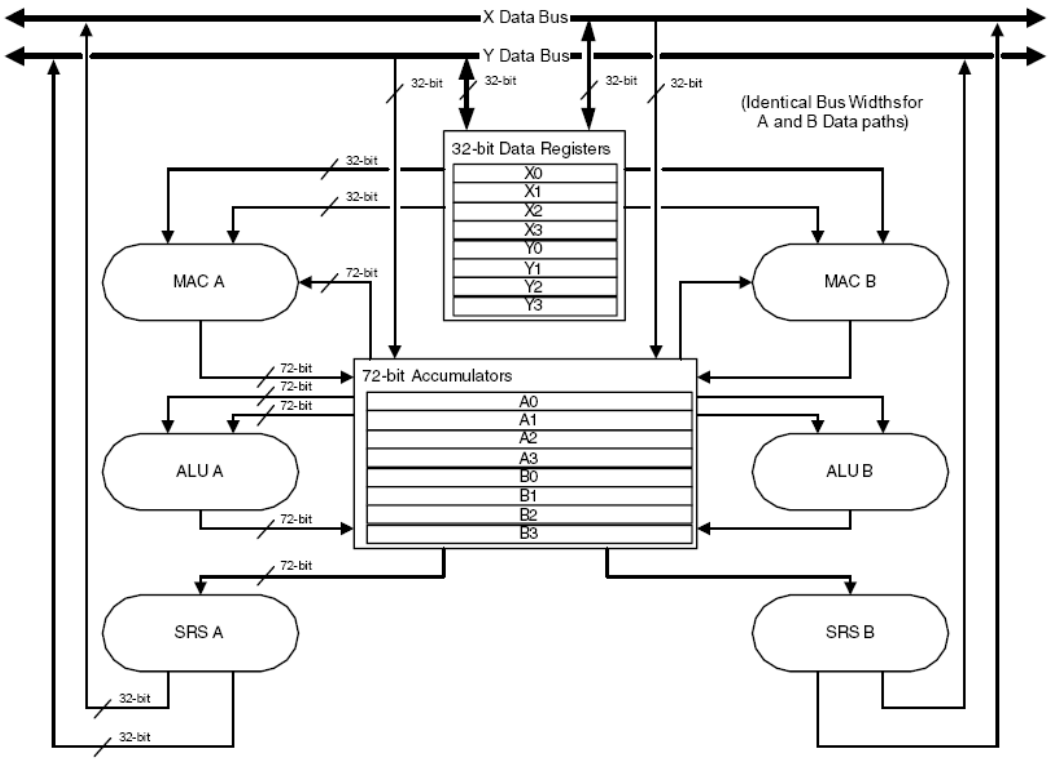
Процесор садржи регистре опште намене (ширине 32 бита), акумулаторске регистре (ширине 72 бита) и индексне регистре (ширине 16 бита).

Подржава разне технике које оптимизују извршење програма као нпр. multiply and accumulate (множи и сабери) и хардверске петље.

На слици 4.1 је дата слика архитектуре Coyote 32 процесора, а слика 4.2 приказује путању података кроз дати процесор.



Слика 4.1 Илустрација архитектуре Coyote 32 процесора



Слика 4.2 Илустрација путање података кроз Coyote 32 процесор

3.2 Типови који подржавају рад са непокретним зарезом у CLIDE-у

Програмско окружење CLIDE преводи по CCC компајлеру који је проширење ISO/IEC TR 18037 стандарда. Постоје два уграђена типа:

- Fract
- Accum

Тип **fract** ради са 32-обитном прецизношћу и чије се вредности крећу у распону од [-1, +1). Тип **short fract** ради исто са 32-обитном прецизношћу, док тип **long fract** ради има ширину меморије 64 бита, али, у суштини, и његова репрезентација се своди на 32-обитну прецизност.

Код типа **accum** типови **short accum** и обичан **accum** заузимају 40 бита меморије, где се 8 бита узима за представу целог броја, а преосталих 32 - за представу разломљеног дела, где је распон типа од [-8, +8). Тип **long accum** заузима 72 бита меморије и његове вредности се крећу у распону од [-256, +256). Тип **unsigned long accum** се креће у распону пд [0, 512).

Програмско окружење CLIDE подржава декларацију ових променљивих називима _Fract и _Accum, међутим уз укључење **stdfix** библиотеке, могу се користити назви fract и accum.

У архитектури процесора Coyote 32, регистри података имају ширину 32 бита, а њихови парови - 64 бита. Акумулаторски регистри имају ширину 72 бита.

Овај рад се највише заснива на коришћењу **stdfix** библиотеке и то највише коришћењем типова **fract** и **long accum**.

4. Концепт решења

У раду под називом "Методологија C компајлера за имплементацију DSP апликација" који је наведен као референца [2], описана је методологија која се користила при прављењу концепта решења за постављени проблем.

У датој методологији су представљена четири модела за израду програма - референтни код (модел 0), модел 1, модел 2, модел 3 и завршни код. Референтни код, модел 1 и модел 2 треба да буду написани у неком општем програмском окружењу. Модел 3 и завршни код се пишу тако да могу да их преводе компајлери за наменске процесоре.

Модел 0 представља програм који је писан од самог почетка, све то тачке када он не профункционише.

Када се изврши било која промена над таквим кодом, најчешће у циљу оптимизације, започиње се на раду са моделом 1, који је завршен када су примењене све оптимизационе технике које је било потребно применити.

Модел 2 представља код који ради са емулационим класама које представљају типове за рад у непокретном зарезу.

Уз анализу модела 0, модела 1, модела 2, пише се и модел 3. Овај модел представља програм који се преводи према компајлеру написаном за CLIDE. Овај компајлер ради са типовима у непокретном зарезу. Често се овај модел пише у посебном програмском окружењу које је намењено за рад са наменским процесорима. У моделу 3 је био важан задатак ослонити се на библиотеке које окружење нуди, за разна стандардна рачунања.

Завршни код је, у суштини, модел 3 подешен тако да се може спустити на конкретан DSP.

За овај дипломски рад су били дати модел 0, модел 1 и модел 2, написани у програмском окружењу опште намене - Visual Studio. Задатак је био написати модел 3 у програмском окружењу CLIDE.

Модел 3 је имплементиран у почетку уз гледање модела 0 и писање новог, оптимизованог кода који ради са аритметиком у *непокретном зарезу.* Међутим, констатовано је да је ова путања превише компликована, те се приступило даљој имплементацији уз гледање модела 2. Модел 1 није узиман у обзир јер је констатовано да је модел 2 довољно оптимизован.

4.1 Главни проблеми у имплементацији и верификацији

У програму су главно место заузеле променљиве типа **fract** и **accum**. То су променљиве реалног типа у непокретном зарезу.

При изради самог програма дошло је и до неких потешкоћа, од којих је главна недовољна функционалност рада са датотекама програмског окружења CLIDE. Стога смо одбирке били приморани да генеришемо програмски, конкретно као синусну функцију, а резултате смо уписивали у текстуални документ.

Поједине библиотечке функције нису радиле (нпр. библиотечка функција за IFFT) или нису биле применљиве за наше потребе (нпр. ми смо радили са подацима типа **accum**, а функција је враћала вредности типа **fract** који нам није обезбеђивао довољну прецизност у датим случајевима). Овај проблем смо решавали тако што смо у пројекат убацивали асемблерске функције одговарајућих карактеристика које су већ уграђене током имплементације неких ранијих пројеката. У будућности ове функције би требало да постану део библиотека које се нуде у окружењу.

Упркос овим недостацима, сама функционалност програма је проверена и констатовано је да обрада ради оно што се од ње тражи. Даље следи опис појединачних програмских модула и закључак са смерницама даљег развоја.

5. Имплементација

Решење је реализовано у 12 програмских модула, неких написаних у програмском језику C, а неких, у асемблеру. То су следећи модули:

1. main.c
2. ss_processing.h и ss_processing.c
3. signals.h и signals.c
4. defines.h
5. windows.h
6. libIFFT.a
7. libMathDiv.a
8. libMathLog10.a
9. libMathSqrt.a

5.1 Модул main

Почетни програмски модул је урађен врло једноставно. У њему су најзначајније следеће променљиве:

1. Низови **cos_re** и **cos_im** типа **fract** који представљају одбирке синусоиде (косинусоиде) које треба обрадити

2. Низ **outputTable** типа **fract** који представља табелу излазних података, као и
3. Показивач на датотеку **output_file**, где уписујемо коначне резултате

Као и позиви следећих потпрограма:

1. cosinus_signal који формира потребну синусоиду
2. init_SS_processing који извршава иницијализацију променљивих програмског модула ss_processing
3. ss_processing који обавља обраду података

Након позива ових функција, врши се упис излазне табеле у текстуални документ.

5.2 Модул ss_processing

Модул који обавља обраду над улазним звуком. Најбитније су му следеће променљиве:

1. Низове **input_buffer** и **output_buffer**, типа **fract**, који природно представљају одбирке улазног и излазног звука и
2. **yBuf_re** и **yBuf_im** који представљају спектар обрађиваног звука

Поред тога, у оптицају су, такође, следеће додатне променљиве које помажу при обради:

1. x_ph, **x_mag**, **x_mag_sm**, **x_mag_sm1**, **x_mag_sm2**, типа **fract**, који служе за прорачуне у вези са представом звука у поларном облику, тј. Његове фазе и модула

2. noise_spectrum, **noise_floor**, **SNR_x** – променљиве задужене за рад са непожељним звуком, такође типа **fract**

3. lobin и hibin, низови типа int који служе за прорачуне β вредности

4. beta_x, типа accum, низ у који се смештају коефицијенти за одузимање одбирака са њиховим процењеним вредностима

Постоје и бројне променљиве мањег значаја као што су акумулатори acc0, acc1, acc2, acc3, tmp, tmp1, tmp2, tmp3, C1, C2, W_09, W_01, W_005, W_001...

Модул садржи следеће потпрограме:

1. getLeadingZeroes који прерачунава са којим бројем можемо скалирати рачунате вредности, јер су већина типа fract
2. find_mag – потпрограм за налажење модула звучних одбирака
3. noise_update – потпрограм за рачунање вредности штетног звука, које треба да одузмемо од улазног
4. ss_processing – главни потпрограм, у коме радимо прелазак из и у спектрални домен и цео горе описани алгоритам
5. init_ss_processing – функција која се позива пре самог ss_processing-а да би се иницијализовале обрађиване глобалне променљиве

5.3 Модул signals

Модул signals се користи за уписивање и испис из *XY* меморије, комплексног сигнала, с обзиром да *FFT* функција ради са типом података **complex_fract_t** који се уписује у *XY* меморију.

Кроз модул се могу направити релативно слични сигнали, који су добри за тестирање, јер имају различити број спектралних компоненти. Сви сигнали имају имагинарни део за нулу, јер је имагинарна компонента звука нула.

Модул signals садржи следеће функције:

1. constant_value_signal прави *сигнал константне вредности* **A**, дефинисане у модулу **defines**

2. one_freq_cosinus_signal прави *косинусни сигнал* фреквенције **FT1**, таласне дужине **FS**, дефинисаних у модулу **defines**

3. two_freq_cosinus_signal прави *косинусни сигнал* фреквенција **FT1** и **FT2** таласне дужине **FS**, дефинисаних у модулу **defines**

4. three_freq_cosinus_signal прави *косинусни сигнал* фреквенција **FT1**, **FT2** и **FT3** таласне дужине **FS**, дефинисаних у модулу **defines**

5. four_freq_cosinus_signal прави *косинусни сигнал* фреквенција **FT1**, **FT2**, **FT3** и **FT4** таласне дужине **FS**, дефинисаних у модулу **defines**

6. print_signal штампа *реалне* и *имагинарне* компоненте одбирака датог сигнала

7. print_signal_el штампа *реалну* и *имагинарну* компоненту једног одбирка датог сигнала

8. write_signal_to_memory уписује низове *реалних* и *имагинарних вредности* које представљају сигнал у XY меморију

9. write_signal_from_memory исписује сигнал из XY меморије

5.4 Модули defines и windows

Defines модул садржи дефиниције константи које користимо у осталим модулима:

1. SS_FFT_SIZE, **SS_FFT_SIZE2** представљају ред *FFT* трансформације и половину њене вредности

2. SS_PROCESSING_BLOCK_SIZE, **SS_PROCESSING_BLOCK_SIZEx2** представљају величину обрађиваног блока и његову дуплу вредност

3. LOG2_FFT_SIZE је логаритам од реда *FFT*–а, што се исто користи при позиву те функције

4. MAX_BANDS максималан број канала звука који су учитани

5. FLOOR је гранична вредност у избору формула за рачунање излаза

6. NOISE_FRAMES – број оквира са којима радимо

7. NUMBER_OF_BANDS – стваран број канала са којима радимо

8. A, PI – константе 0.5 и 3.1415926535

9. FS, FT1, FT2, FT3, FT4 – фреквенције за одређене косинусе

10. MAX_SHIFT – максимална вредност за коју можемо да shift–ујемо (скалирамо) прорачунате вредности

11. THRESH_05 – гранична вредност у вредности од 0.5 изнад које се ради скалирање

Садржи и следеће макро – дефиниције:

1. **FRACT_NUM(x)** преко које претварамо број типа *покретног зареза* у тип *непокретног зареза*, конкретније **fract**
2. **ACCUM_NUM(x)** преко које претварамо број типа *покретног зареза* у тип *непокретног зареза*, конкретније **accum**

Windows модул садржи низ **fract** вредности **wind** које служе за прозорирање улаза.

5.5 Модули libIFFT, libMathDiv, libMathLog10, libSqrt

Ови модули су асемблерске имплементације:

1. *IFFT* трансформације

2. Дељења fixed point бројева типа long accum

3. Логаритмовања по основи 10 бројева типа long accum

4. Кореновања бројева типа long accum

Проблем се почео појављивати при позиву одређених библиотечких функција у програмском окружењу CLIDE. Нисмо имали математичке функције за рад са потребном прецизношћу (тип long accum), а библиотечка имплементација функције IFFT из неког разлога није радила.

Зато су узете одговарајуће асемблерске верзије тих функција и било је потребно наместити их да се могу позвати из C програма. Било је битно да асемблерске функције раде по позивној конвенцији за дато програмско окружење. Позивајућа функција би требала да сачува следеће регистре: a0, a1, b0, b1, x0, x1, y0, y1, i0, i1, i4, i5, док позвана фунцкија треба да сачува следеће регистре: a2, a3, b2, b3, x2, x3, y2, y3, i2, i3, i6, i7. Начин прослеђивања параметара и повратне вредности већ је задовољавао позивну конвенцију.

У програму није било потребно додатно чување регистара у позивајућој функцији (ss_processing), јер је она била сва написана у C–у, међутим, у позваним асемблерским

функцијама, које су горе наведене, је у случају асемблерске имплементације IFFT-а било потребно чување поменутих регистара.

6. Тестирање и валидација

С обзиром да датотечки систем није радио у програмском окружењу CLIDE, програмски је генерисан косинус сигнал и у CLIDE–у и у Visual Studio–у, од 64 одбирака, и поређени излази су уписивани у текстуалне документе (CLIDE није радио са бинарним – WAV датотекама).

При поређењу излазних датотека констатовано је да су излази идентични у већини случајева, са изузецима од једног, два бита разлике.

С обзиром да се емулационе класе типа **fract** у Visual Studio–у могу исписати једино у хексадецималном облику, одлучено је да се провера ради поредећи хексадецималне бројеве.

7. Закључак

У овом дипломском раду је направљено програмско решење у програмском окружењу CLIDE по узору на моделе из Visual Studio–а. Недостаци су, свакако, што програм не ради са правим датотекама, већ фактички са бројевима.

Поређењем унутрашњих представа звука у debug режиму и исписа излазних резултата, са бројевима и излазним резултатима у Visual Studio–у дошли смо до закључка да програм ради исправно.

Превасходно би зато и даљи ток рада на датом пројекту био покушај увођења рада са датотекама и тестирања рада имплементације датог алгоритма у програмском окружењу CLIDE са поментим проширењима.

Међутим, исто би корисно било нпр. само одрадити поређење рада референтног и урађеног кода за неке друге програмски генерисане одбирке, нпр. За сигнал WAV фајлова добијених уз референтни код.

Заузеће програмске меморије зависи од тога да ли је алгоритам SS processing писан у C-у или у асемблеру. Такође, разне оптимизационе технике могу побољшати брзину извршења програма. Тако програмска меморија може бити више заузета, ако дати C програм ради у дебаг режиму, а статички стек убрзава извршење ове C имплементације.

	Lib	Interface	Built-in	Main	Total
Asm implementation	103	96	0	399	598
C implementation out of debug regime	734	115	431	2147	3427
C implementation in debug regime	734	194	431	3376	4735
C implementation with static stack frame	734	114	431	2096	3375

Табела 15.1 Заузеће програмске меморије у зависности од програмског језика и оптимизованости кода

Рубрика Lib означава заузеће програмске меморије при позивима библиотечких функција приложених на почетку имплементације. Рубрика Interface означава заузеће програмске меморије при улазно-излазним операцијама и самом покретању обраде. Main је главни део обраде, у који спадају модули signals и ss_processing. Built-in представља разне асемблерске функције које се позивају "испод хаубе", при коришћењу, углавном, готових, библиотечких функција.

Из приказаног видимо да је знатно мање захтевна асемблерска имплементација датог алгоритма. При извршењу главног (main) дела програма, програмска меморија је 80% више заузета у C имплементацији у debug режиму, у односу на асемблерску имплементацију Spectral Subtraction алгоритма. Када смо у debug режиму, програмска меморија бива чак 90% више заузета.

Међутим, овде су наведене само најједноставније C имплементације датог алгоритма, без коришћења значајнијих оптимизационих техника. Даље место за рад остаје, наравно, и ту, да се примене разне оптимизационе технике и види колико се може извршење датог програма побољшати.

Литература

[1] Стеван Бербер, Миодраг Темеринац: *Основи алгоритама и структура DSP*, Унвиерзитет у Новом Саду, Факултет техничких наука, 2004

[2] Владимир Ковачевић, Мирослав Поповић, Миодраг Темеринац, Никола Теслић – *Архитектуре и алгоритми дигиталних сигнал процесора 1,* Универзитет у Новом Саду, Факултет техничких наука, 2005

[3] Миодраг Ђукић, Ненад Четић, Јелена Ковачевић, Мирослав Поповић - *A C compiler based methodology for implementing audio DSP applications on a class of embedded systems*, 2008

[4] Миодраг Ђукић, *Ново решење компајлерске инфраструктуре за наменске процесоре,* Универзитет у Новом Саду, Факултет техничких наука, 2014

[5] Владимир Ковачевић, Мирослав Поповић, *Системска програмска подршка у реалном времену 1 – Програмски алати и паралелно програмирање,* Универзитет у Новом Саду, Факултет техничких наука, 2011