Санкт-Петербургский государственный университет

Математическое обеспечение и администрирование информационных систем.
Системное программирование

Афанасов Артем Константинович

Реализация асимметричного маркерного процессинга на C66x DSP

Курсовая работа

Научный руководитель:
д.ф.-м.н., проф. А. Н. Терехов
Научные консультанты:
ст. преп. М. В. Баклановский
ст. преп. А. Р. Ханов

Санкт-Петербург
2020

Оглавление

Введение

В рамках данной работы будет реализован асимметричный маркерный процессинг (краткое описание которого будет в обзоре существующих решений и литературы, далее — АМП) [14] на процессоре C66x DSP (C66x — общее имя для новой архитектуры C6000 DSP [1]), который предназначен для обработки оцифрованных сигналов, для коммуникации между двумя архитектурами (C66x DSP (далее — DSP) и ARM® Cortex™-A15 MPCore™ (далее — ARM)). DSP входит в состав системы на кристалле 66AK2H12 Multicore DSP+ARM® KeyStone™ II System-on-Chip (SoC) (далее — KeyStone 2) [5]. Результат данной работы — реализация на архитектуре KeyStone 2 со стороны DSP части функциональности стека протоколов АМП.

Для эффективного решения вычислительной задачи может понадобиться разделение её на фрагменты и исполнение полученных фрагментов различными процессорами. Логично делегировать фрагменты задачи специализированным процессорам, например, высокоскоростным, энергоэффективным или допускающим обфускацию. Также для повышения отказоустойчивости системы (имеется в виду, что выполнение задачи завершится успешно), нужна гарантия того, что каждый фрагмент задачи будет выполнен. Для этого нужен механизм, позволяющий общаться с повышенной надежностью устройствам, находящимся в гетерогенной системе. Надёжным механизмом по передачу управление между модулями гетерогенной системы является асимметричный маркерный процессинг.

В данной работе специализированным процессором выступает высокоскоростной ядро DSP, на которое поступают задачи по цифровой обработке сигналов. Конкретная задача может быть следующей. Система на кристалле включает в себя ядра ARM и ядра DSP. На ARM поступает задача по обработке фотографии. ARM делегирует обязанность по обработке изображения DSP, так как он эффективно вычисляет преобразование Фурье. Здесь и нужно обеспечить коммуникацию

между процессорами, которая, причём, будет ещё и надёжная. Надёжной она будет, потому что ARM передаст исполнение фрагмента задачи по преобразованию Фурье только тому ядру DSP, которое сможет принять задачу.

При проектировании гетерогенных систем возникают проблемы [16], которые решает АМП:

- отсутствие целостного маршрута проектирования от программной модели верхнего уровня к ПО целевой системы: АМП позволяет программу на языке высокого уровня напрямую переложить на конкретное железо;
- ручное проектирование коммуникаций между ядрами и общего управления системой: АМП обеспечивает общение между ядрами разных архитектур, а также позволяет управлять системой;
- отсутствие средств отладки на уровне системы: АМП позволяет отлаживать всю систему в целом;
- отдельная среда проектирования для каждого вида вычислительных ядер: АМП организует единую вычислительную среду для проектирования.

Реализация под конкертную архитектуру процессора системы на кристалле — это область низкого программирования. Помимо написания исходного текста программы необходимо понимать устройство архитектуры, а также способы взаимодействия с ней. Для этого следует знать: как подключаться к исполняемым модулям системы на кристалле; как настраивать сборку программы, учитывая архитектуру системы на кристалле; язык ассемблера процессора; отдельные модули процессора.

В данной работе будет достигаться обеспечение коммуникации между DSP и ARM согласно АМП. Практическая значимость данной работы заключается в следующем:

- Данная работа является программной реализацией части стека протоколов АМП. Тем самым, можно реализовывать отказоустойчивую высокопроизводительную систему на нужной архитектуре, основываясь на исходном коде данной работы.

- Данную работу можно будет использовать для создания отказоустойчивых высокопроизводительных систем, основанных на архитектуре KeyStone 2, со стороны DSP.

1. Постановка задачи

Целью данной работы является реализация канального, сетевого, транспортного уровней стека протоколов асимметричного маркерного процессинга на DSP архитектуры KeyStone 2 для коммуникации с ARM. Для её достижения необходимо решить следующие задачи.

1. Изучить асимметричный маркерный процессинг.
2. Проанализировать инструменты для реализации.
3. Обеспечить общение между ядрами DSP и ARM.
4. Реализовать канальный (в статье АМП он назван физическим), сетевой, транспортный уровни стека протоколов асимметричного маркерного процессинга на DSP.
5. Измерить скорость передачи маркера.

2. Обзор существующих решений и литературы

2.1. Существующие подходы к проектированию гетерогенных вычислительных систем

- HSA от AMD;
- Code Composer Studio от Texas Instruments;
- Vivado от Xilinx.

Решения от AMD поддерживает архитектуры CPU + GPU, от Texas Instruments — RISC + DSP, от Xilinx — RISC + FPGA. В этом заключается проблема: проектирование возможно только под собственные аппаратные платформы компаний-разработчиков. АМП решает эту проблему, так как является унифицированным механизмом, позволяющим как проектировать гетерогенные вычислительные системы различных архитектур, так и отлаживать всю систему в целом.

2.2. Парадигма ведущий/ведомый

В модели ведущий/ведомый главный процессор является узлом коммуникации между ведомыми процессорами. В АМП виду того, что все процессоры равноправны, любые устройства могут общаться между собой, что влечет повышение производительности из-за отсутствия необходимости связи с дополнительным узлом.

В модели асимметричного маркерного процессинга все процессоры равноправны, причём, сами процессоры и их роли в вычислительном процессе отличаются друг от друга. Причем, парадигма ведущий/ведомый может быть реализована по АМП. АМП — это унифицированный механизм передачи управления.

Стоит также заметить, что в АМП был произведён отказ от вызова исполнения кода с последующим возвратом в место вызова. По АМП

управление передаётся без обязательного возвращения управления обратно.

2.3. Асимметричный маркерный процессинг

Используя АМП, становится возможным разбивать программу на базовые блоки, которые транслируются при компиляции под разные процессоры. И эту программу процессоры исполняют по очереди, а способ коммуникации между процессорами также задаёт АМП. Такой способ организации вычислений нужен для исполнения одной и той же программы на ядрах разных архитектур в гетерогенной системе. Это позволяет достигать целей обфускации, повышения производительности, автоматического распараллеливания.

Также в АМП входит модуль отладочных диалогов, позволяющий управлять всей системой и отлаживать её.

Структура асимметричного маркерного процессинга заключается в следующем. Он использует стек протоколов из 5 уровней: прикладной, представление, транспортный уровень, сетевой, канальный. По данному стеку передаётся абстракция маркера от одного процессора до другого. Маркер — это допуск для процессоров на исполнение фрагмента задачи, вместе с которым передаются данные, необходимые для выполнения. То есть исполнять фрагмент большой задачи может только тот процессор, у которого находится маркер. Маркер в системе только один.

На прикладном уровне для программиста реализуется возможность делегирования фрагментов задачи между процессорами: обеспечивается создание структуры передачи маркера и реализуется функция передачи управления в пространства инструкций других процессоров. На уровне представления допускается кодирование и декодирование пакета, входящего в структуру передачи маркера. Транспортный уровень обеспечивает надёжную передачу маркера между процессорами путём тройного рукопожатия с повторной отправкой пакетов в случае отсут-

ствия ответа. Сетевой уровень реализует корректную передачу маркера между процессорами. На канальном уровне происходит непосредственная передача данных посредством общей памяти процессоров.

АМП обеспечивает надёжность системы. Обеспечение надёжности необходимо по той причине, то АМП — это общий подход к проектированию систем. Поэтому ошибки при передаче данных могут возникать по причине высоких скоростей каналов передачи, а также внешних условий.

Также в АМП входит описание для реализации отладочного диалога. Он позволяет управлять системой и проверять её.

2.4. Реализация авторами статьи АМП

Реализация АМП принадлежит его же авторам. Языками реализации были Си и язык ассемблера. Но после замеров оказалось, что Си проигрывает по производительности языку ассемблера в 1.5-3 раза. После чего разработка велась на языке ассемблера. АМП был реализован на системе Zybo Zynq-7000, в которую входит ARM® Cortex™-A9 и FPGA. В данной работе вычислителями являются ARM® Cortex™-A15 MPCore™ и C66x DPS.

Таким образом, работа отличается от описанной реализации используемой архитектурой.

3. Используемые инструменты и технологии

3.1. 66AK2H12 Multicore DSP+ARM® KeyStone™ II System-on-Chip (SoC)

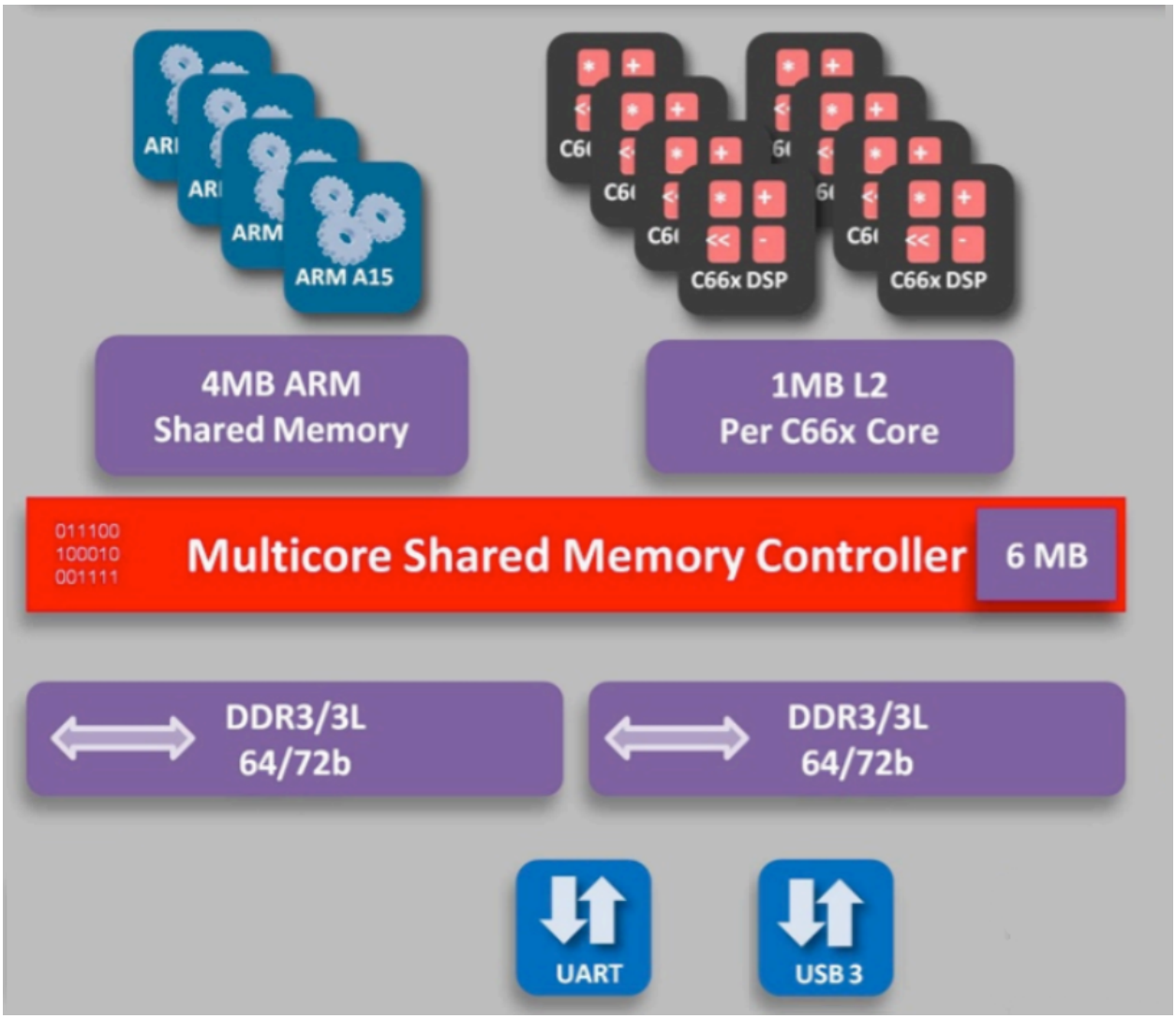

Рис. 1: 66AK2H12 system on chip [2]

Система на кристалле имеет процессор семейства KeyStone2, в состав которого входят набор из 4 ядер ARM и набор из 8 ядер DSP. Плата имеет разделяемую высокоскоростную память Multicore Shared Memory Controller (далее — MSM) размером 6 MB, через которую в данной работе происходит общение между ядрами разных архитектур. MSM

является SRAM памятью, которая находится непосредственно внутри системы на кристале [4]. В данной работе память используется как для загрузки исполняемых программ, так и для передачи данных между ядрами. В командном файле определяются адреса и размеры участков памяти, а также выбор участков для загрузки в них секций кода.

3.2. Язык ассемблера

В рамках работы были проведены предварительные замеры времени выполнения программ, написанных на Си и на яызке ассемблера DSP [12]. Исходя из них, был сделан вывод о первосходстве ассемблера DSP по сравнению с Си.

Язык ассемблера DSP был указан в качестве условия выполнения работы.

3.3. Code Composer Studio

Данная интегрированная среда разработки предлагалась в документации к KeyStone 2 [7]. Для разработки программ под процессор DSP системы на кристалле KeyStone 2 нужно провести ряд необходимых настроек и конфигураций, что и позволяет сделать CCS. Также через CCS можно отлаживать программы.

4. Реализация

4.1. Работа с EVMK2H

Подключение к ядрам.

Подключиться к Linux, установленному на плате, не удалось. Поэтому работа с ядрами происходила посредством:

- BMC (Board Management Controller) — контролирует минимальную настройку Keystone 2: питание, такты, перезагрузку, режимы загрузки;
- JTAG — интерфейс, позволяющий отлаживать программы, запущенные на ядрах.

Исполняемые файлы программ, написанных на Си и на языке ассемблера DSP загружались на ядре Keystone 2 через JTAG после загрузки платы через BMC.

SYS/BIOS.

Под Keystone 2 имеется возможность использовать операционную систему реального времени SYS/BIOS. При написании программ на Си SYS/BIOS использовалась как библиотека, а не операционная система: к проекту в конфигурационном файле подключались отдельные модули SYS/BIOS, а на плату загружался исполняемый файл, содержащий программу и модули SYS/BIOS.

Разделяемая память.

Вся память на KeyStone 2 логически имеет уникальное расположение на карте памяти [3]. Для передачи маркера между процессорами используется разделяемая память MSM. В архитектуре KeyStone 2 она находится по адресу 00 0C00 0000 - 00 0C5F FFFF [8]. Исполняемые программы также загружаются в MSM (загрузка секций кода определяется в командном файле).

Используя MSM, удалось передать информацию между DSP и ARM (программы по передачи данных написаны в ходе данной работы как на Си, так и на языке ассемблера).

Control status register

Для идентификации процессора можно использовать информацию CPU ID, находящуюся в 24-31 битах регистра CSR. Также 8 бит CSR отвечает за индианность процессора DSP.

Когерентность кешей.

Для доступа к описываемым далее регистрам управления кешем нужно знать их адрес в карте памяти системы на кристалле, в отличие от Control Status Register, для которого есть мнемоника — CSR.

Можно конфигурировать L1D кеш всей памяти, но так как обеспечить когерентность кеша нужно только для циклического буфера MSM, в который записываются байты пакетов от другого ядра, то выбраны операции по когерентности блоков памяти.

Для этой цели используются следующие регистры: L1DXXBAR (где XX — это либо W, т.е. обратная запись, либо WI, т.е. обратная запись и признание кеша недействительным, либо I, т.е. признание кеша недействительным) определяет базовый адрес блока, а регистр L1DXXWC определяет его размер. Для начала операции когерентности блока нужно записать отличающееся от нуля значение в регистр L1DXXWC, который сбрасывается в 0 после завершения операции когерентности, что позволяет следить за окончанием операции.

Регистры L1DWBAR, L1DWWC отвечают за обратную запись в память. Их адреса 0184 4040h , 0184 4044h, соответственно [10].

Регистры L1DIBAR, L1DIWC отвечают за признание L1D кеша недействительным. Их адреса 0184 4048h, 0184 404Ch, соответственно.

Расположение в карте памяти регистра L1DCFG (конфигурирует размер L1D кеша) — 0184 0040h. Значение 111b в битах L1DMODE конфигурирует максимальный размер кеша, а 000b — отключает кеш L1D [9].

4.2. Реализация АМП

Для формирования отправляемых и принимаемых данных используется стек. Специализированных операций со стеком, как у Intel®, например, popa и pusha, у DSP нет. Для этого используются операции декрементации и инкрементации указателя стека и запись по указателю стека. На устройствах c6000 регистр B15 используется в качестве указателя стека. Стек растет сверху вниз. Указатель стека всегда указывает на следующее неиспользуемое местоположение [11], [6].

Выделено служебное место в памяти: 0x0C180000 - 0x0C180020. Цилкический буфер DSP начинается с адреса 0x0C180024 и имеет длину 0x0017FFE0 (размер почти 1.5 МБ). Место в памяти MSM под программы DSP начинается с 0x0C480000 и имеет длину 0x00180000 (размер 1.5 МБ). 6 МБ памяти MSM разделяется между DSP и ARM под циклические буферы принятых пакетов и расположение исполняемых программ.

Между ARM и DSP нужно согласовать сохранение и выгрузку байтов из памяти, используя команды по загрузке/сохранению байт с одинаковой размерностью (8, 16, 32 бита) с обеих сторон. Ибо без этого чтение байтов из памяти сильно осложняется.

Каждый уровень в реализованном стеке протоколов АМП выполняет свои обязанности.

- Канальный — передаёт байты со стека одного процессора в циклический буфер разделяемой памяти другого процессора, из которого второй процессор считывает байты. Также этот уровень при отсутствии маркера проверяет циклический буфер разделяемой памяти своего процессора, в которой пишет байты другой процессор.

- Сетевой уровень — обеспечивает формирование заголовков для корректной передачи нужному процессору. На принимающей стороне этот уровень проверяет корректность переданных данных.

- Транспортный уровень — обеспечивает надёжнную передачу маркера путём подтверждением получения маркера. Также обеспечивается повторная отправка пакетов, если вышел указанный таймаут ожидания ответа.

Формат заголовков:
2 байта под длину пакета
Сетевой уровень:

- ”кому” (1 байт);
- ”от кого” (1 байт);
- ”модуль транспортного уровня” (1 байт, 1 — модуль передачи маркера, 0 — отладочный модуль);
- ”контрольная сумма (3 байта)”;

Модуль передачи маркера:

- ”тип передачи маркера” (001 — тип 1, 010 — тип 2, 100 — тип 3)
- ”номер транзакции” (4 байта)

4.3. Замеры

Использовались дельта по количеству тактов процессора и его частота, время вычислялось в наносекундах. Проводились замеры программ, написанных на Си и на языке ассемблера. Проводилось по 128 замеров до стабилизации среднего результата.

Засечки циклов производились на ARM. Для получения тактов на DSP нужно читать регистры TSCL и TSCH [13].

ARM работал на частоте 1.4 ГГц. DSP работал на частоте 1 ГГц. Компилятор Си под DSP с опцией -O3, компилятор Си под ARM с опцией -Ofast.

Сценарий замера по передаче 12 байт от одного процессора к другому:

1. ARM делает засечку перед записью в MSM
2. ARM пишет в MSM
3. DSP читает из MSM
4. DSP выставляет в MSM флаг о прочтении
5. ARM читает флаг о прочтении и делает вторую засечку количества тактов

Ниже приведена таблица данных замеров.

Реализация	Матожидание, нс	Среднеквадратическое, нс
C DSP (-O3), C ARM (-Ofast)	472	109
ASM DSP, C ARM (-Ofast)	229	9
ASM DSP, ASM ARM	292	2

Рис. 2: Замеры по передаче 12 байт.

По предварительным замерам можно заметить, что программы под DSP, написанные на Си, медленнее программ на языке ассемблера DSP в среднем в 2 раза. Также видно, что среднеквадратическое отклонение замеров времени для программ DSP на Си сильно превышает среднеквадратическое отклонение замеров времени программ на языке ассемблера DSP.

Сценарий замера по передаче трёх пакетов по 92 байт между процессорами с получением подтверждения получения:

1. ARM делает засечку перед записью в MSM
2. ARM пишет в MSM
3. DSP читает из MSM

4. DSP пишет пакет в MSM и выставляет флаг о подтверждении получения пакета

5. ARM читает флаг, читает пакет

6. ARM пишет пакет о получении подтверждения, делает вторую засечку тактов

Ниже приведена таблица данных замеров.

Реализация	Матожидание, нс	Среднеквадратическое, нс
C DSP (-O3), C ARM (-Ofast)	4099	176
ASM DSP, C ARM (-Ofast)	2468	14
ASM DSP, ASM ARM	2500	6

Рис. 3: Замеры по передаче трёх пакетов по 92 байт между процессорами с получением подтверждения получения.

Программы можно загружать как в MSM, так и в DDR3. Загрузка программ в MSM уменьшает среднеквадратическое отклонение скорости исполнения программы.

Ниже приведена таблица данных замеров.

Память	Матожидание, нс	Среднеквадратическое, нс
MSM	2468	14
DDR3	2545	69

Рис. 4: Замеры по расположению программ в MSM и DDR3.

Заключение

В итоге для достижения поставленной цели, а именно — реализация канального, сетевого, транспортного уровней стека протоколов асимметричного маркерного процессинга на DSP архитектуры KeyStone 2 для коммуникации с ARM, выполнены следующие задачи.

1. Изучен асимметричный маркерный процессинг.
2. Проанализированы инструменты для реализации.
3. Обеспечено общение между ядрами DSP и ARM.
4. Реализованы и задокументированы канальный, сетевой, транспортный уровни стека протоколов асимметричного маркерного процессинга на DSP [15].
5. Измерена скорость передачи маркера.

Благодарности

Хочу поблагодарить М. В. Баклановского, Б. Н. Кривошеина, М. А. Терехова, М. А., А. Е. Сибирякова за помощь в исследовании предметной области. А также А.Р. Ханова за помощь в исследовании аппаратуры 66AK2H12 Multicore DSP+ARM® KeyStone™ II System-on-Chip (SoC). И выражаю благодарность математико-механическому факультету за предоставления данной системы на кристалле, в тесном контакте с которой была реализована данная курсовая работа.

Список литературы

[1] Instruments Texas. TMS320C66x DSP CorePac User Guide, стр. A-1 // Сайт Texas Instruments. URL: http://www.ti.com/lit/ug/sprugw0c/sprugw0c.pdf?&ts=1590006890395 (дата обращения: 19.05.2020).

[2] Texas Instruments. 66AK2H12 system on chip // Сайт CommAgility. URL: https://www.commagility.com/images/pdfs/white_papers/CommAgility_white_paper_floating_point_with_TI_C66x_DSPs.pdf (дата обращения: 19.05.2020).

[3] Texas Instruments. 66AK2H14/12/06 datasheet, стр. 21 // Сайт Терраэлектроника. URL: https://spb.terraelectronica.ru/pdf/show?pdf_file=%252Fz%252FDatasheet%252F6%252F66ak2h06.pdf (дата обращения: 19.05.2020).

[4] Texas Instruments. 66AK2H14/12/06 datasheet, стр. 5 // Сайт Терраэлектроника. URL: https://spb.terraelectronica.ru/pdf/show?pdf_file=%252Fz%252FDatasheet%252F6%252F66ak2h06.pdf (дата обращения: 19.05.2020).

[5] Texas Instruments. 66AK2Hxx Multicore DSP+ARM® KeyStone™ II System-on-Chip (SoC) // Сайт Texas Instruments. URL: http://www.ti.com/lit/ds/symlink/66ak2h12.pdf (дата обращения: 13.12.2019).

[6] Texas Instruments. Checking for Stack Overflow // Сайт Texas Instruments Wiki. URL: https://processors.wiki.ti.com/index.php/Checking_for_Stack_Overflow (дата обращения: 01.05.2020).

[7] Texas Instruments. EVMK2H Evaluation Module // URL: http://wfcache.advantech.com/www/support/TI-EVM/Rev4_0/QuickStartGuide/EVMK2H_QSG_v2.pdf (дата обращения: 19.05.2020).

[8] Texas Instruments. SPRS866E стр. 91, таблица 6-1 // Сайт Терраэлектроника. URL: https://spb.terraelectronica.ru/pdf/show?pdf_file=%252Fz%252FDatasheet%252F6%252F66ak2h06.pdf (дата обращения: 19.05.2020).

[9] Texas Instruments. SPRS866E—November 2013, стр. 21, 3.1.2 // Сайт Терраэлектроника. URL: https://spb.terraelectronica.ru/pdf/show?pdf_file=%252Fz%252FDatasheet%252F6%252F66ak2h06.pdf (дата обращения: 19.05.2020).

[10] Texas Instruments. SPRUGW0C, July 2013, стр. 3-12, таблица 3-6 // Сайт Texas Instruments. URL: https://www.ti.com/lit/ug/sprugw0c/sprugw0c.pdf?ts=1591987389865&ref_url=https%253A%252F%252Fwww.google.com%252F (дата обращения: 19.05.2020).

[11] Texas Instruments. TMS320C6000 Programmer's Guide стр. 7-12 (7.5.2 ISR with Hand-Coded Assembly) // Сайт Texas Instruments. URL: http://www.ti.com/lit/ug/spru198k/spru198k.pdf?&ts=1590008074499 (дата обращения: 19.05.2020).

[12] Texas Instruments. TMS320C66x DSP CPU and Instruction Set // Сайт Texas Instruments. URL: https://www.ti.com/lit/ug/sprugh7/sprugh7.pdf?ts=1591991344818&ref_url=https%253A%252F%252Fwww.google.com%252F (дата обращения: 19.05.2020).

[13] Texas Instruments. TMS320C66x DSP CPU and Instruction Set Reference Guide, стр. 2-31 // Сайт Texas Instruments Wiki. URL: http://www.ti.com/lit/ug/sprugw0c/sprugw0c.pdf?&ts=1590008297412 (дата обращения: 19.05.2020).

[14] АСИММЕТРИЧНЫЙ МАРКЕРНЫЙ ПРОЦЕССИНГ / М. В. Баклановский, Б. Н. Кривошеин, А. Н. Терехов et al. // Программная инженерия. — 2018. — Vol. 9, no. 4. — P. 156–162.

[15] Артем Афанасов. Реализация части функциональности стека протоколов асимметричного маркерного процессинга // URL: https://github.com/ArtyomAfanasov/AMP (дата обращения: 19.05.2020).

[16] Кривошеин Борис. Презентация 'Ассиметричный маркерный процессинг', г. Алушта, 04 октября 2017 г. Формум Микроэлектроника 2017.