Санкт-Петербургский государственный университет

Программная инженерия
кафедра системного программирования

Евгений Алексеевич Богданов

Реализация асимметричного маркерного процессинга на архитектуре keystone 2

Курсовая работа

Научный руководитель:
д. ф.-м. н., профессор А. Н. Терехов

Консультант:
ст. преп. А.Р. Ханов

Санкт-Петербург
2020

Оглавление

Введение

Существует несколько основных подходов к проектированию архитектуры современного компьютера. Самой простой и наименее производительный из них — одноядерная архитектура. Сегодня данный подход становится все менее и менее популярным и используется преимущественно для узкого спектра специализированных устройств [22]. Намного эффективнее и популярнее оказалась многоядерная архитектура. Она используется практически в любом достаточно сложном вычислительном устройстве и огромное внимание уделяется разработке программных решений под такие платформы [25].

Однако наибольшую производительность способны обеспечить сложные гетерогенные системы включающие себя несколько архитектур, которые помимо параллельности вычислений могут предложить и наиболее эффективный способ аппаратного выполнения этих вычислений. Именно поэтому построение многоядерных гетерогенных систем является интересной и актуальной задачей на сегодняшний день, которой занято множество IT компаний по всему миру [6].

В тоже время с увеличением сложности платформы существенно возрастает и сложность её программирования. Появляются барьеры проектирования такие как [21]:

- Отдельная среда проектирования для каждого вида вычислительных ядер.
- Ручное проектирование коммуникаций между ядрами и общего управления системой.
- Отсутствие средств отладки на уровне всей системы.

Существуют различные подходы к решению таких проблем, один из них — асимметричный маркерный процессинг (АМП) [24], разработанный на математико-механическом факультете СПБГУ. Данный подход позволяет спроектировать гетерогенную систему обладающую единой вычислительной средой, унифицированным механизмом пере-

дачи управления, а также генерацией части элементов системы на этапе компиляции, что позволяет решить указанные выше проблемы.

В рамках этой курсовой планируется построить такую систему на базе платы evmk2h, представляющей из себя систему на кристалле, поддерживающую 2 архитектуры — ARM и DSP.

1. Асимметричный маркерный процессинг

1.1. Описание

Асимметричный маркерный процессинг — подход к созданию сложных гетерогенных систем с возможностью организации единой вычислительной среды с унифицированным механизмом передачи управления. Его название можно расшифровывать следующим образом:

- *Асимметричный* означает, что как сами процессоры, так и их роли в общем вычислительном процессе отличаются друг от друга.
- *Маркерный* означает строгую очередность участия разных процессоров в вычислениях, т.е. в любой момент времени над решением задачи может работать только один процессор. Право решать задачу дает, уникальный для каждой задачи, *маркер*. По желанию программиста он может быть передан от одного процессора другому. Изначально маркер принадлежит процессору, который по воле программиста, первым начал работу над решением задачи. Считается, что система выполняет одну вычислительную задачу, поэтому единовременно маркер в ней может быть только один.

АМП обладает рядом преимуществ перед своими аналогами и конкурентами, а именно:

1. Поддержка произвольных вычислительных систем с ядрами CPU, GPU, DSP, FPGA любых производителей и архитектур.
2. Возможность создания единой вычислительной среды для программируемых ядер и логических структур в FPGA.
3. Общий унифицированный механизм передачи управления между ядрами, поддерживающий произвольную последовательность передач управления между устройствами.
4. Сквозной маршрут проектирования от языка высокого уровня к машинному коду всей системы.

5. Автоматическая генерация межпроцессорных коммуникаций.

6. Отладка на уровне системы.

1.2. Стек протоколов передачи маркера

Для организации коммуникации между процессорами авторами асимметричного маркерного процессинга предложен стек протоколов [24], основанный на протоколах сетевого стека ISO/OSI. Используемый протокол включает в себя пять уровней:

1.2.1. Прикладной уровень

С этим уровнем работает прикладной программист по желанию которого может быть организована передача маркера одному из процессоров, доступных системе. Для этого формируется структура передачи маркера состоящая из 4-х полей:

1. Номер процессора, которому передается маркер.

2. Размер пакета.

3. Пакет с номером точки входа, где номер точки входа — это номер строки в таблице, содержащей адреса точек входа доступных в системе функций (данная таблица создается компилятором во время трансляции).

4. Результат транзакции на транспортном уровне.

1.2.2. Уровень представления

На данном уровне может быть проведена кодировка пакета одним из способов определенных при разработке системы. После проведения шифрования к пакету добавляется заголовок уровня представления, в котором указывается номер выбранной кодировки, в случае если кодирование не проводилось указывается значение равное 0.

1.2.3. Транспортный уровень

Обеспечивает коммуникацию между процессорами. Оперирует четырьмя видами пакетов:

Пакет 1: сама передача маркера.

Пакет 2: подтверждение, что маркер получен (ответ на пакет 1).

Пакет 3: подтверждение, что оповещение о получении маркера получено (ответ на пакет 2).

Пакет 4: аварийное завершение транзакции.

Работа транспортного уровня может быть удобно представлена в виде конечного автомата Рис. 1:

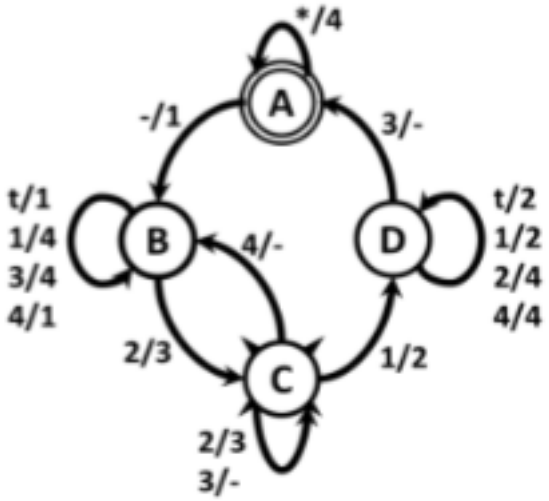

Рис. 1: Автомат протокола передачи маркера между процессорами [23].

На данном рисунке использованы следующие обозначения:

1. a/b — на пакет a отвечать пакетом b
2. t — таймаут
3. 1-4 — тип пакета

4. * — пакет любого типа

5. - — отсутствие пакета

6. Состояние A — процессор владеет маркером и исполняет линейный участок программы

7. Состояние B — процессор начинает передачу маркера другому процессору

8. Состояние C — процессор не обладает маркером и не осуществляет его передачу или получение

9. Состояние D — процессор осуществляет прием маркера

Процессор, обладающий маркером (находящийся в состоянии A), начинает передачу управления, посылая пакет 1 другому процессору, и переходит в состояние B. В состоянии B он по некоторому таймауту осуществляет повторную посылку пакета типа 1. Второй процессор (находящийся в состоянии C) при получении пакета 1 переходит в состояние D и посылает пакет 2. Получив пакет 2, первый процессор входит в состояние C и посылает пакет 3. Второй процессор, получив пакет 3 принимает состояние A. На этом передача маркера считается завершенной.

Более подробное описание транспортного уровня и его полную верификацию можно найти в статье "Верификация протокола передачи маркера в стеке асимметричного маркерного процессинга." [23].

1.2.4. Сетевой уровень

На сетевом уровне к пакету добавляется заголовок с номерами передающего и принимающего процессоров и контрольная сумма всего пакета (включая добавленный заголовок), после чего выбирается способ передачи на физическом уровне и вызывается соответствующая функция физического уровня.

1.2.5. Физический уровень

Занимается переносом пакета из буфера передающего процессора в буфер принимающего.

1.3. Формат пакета

Пакеты передачи маркера имеют 3 заголовка, за которыми следует необязательная область данных.

Заголовок 1: сетевой уровень

1. Номер получателя (1 байт, 0 если широковещательный пакет)
2. Номер отправителя (1 байт)
3. Вид пакета (1 байт, 1 — передача маркера, 2 — отладочное сообщение)
4. Контрольная сумма (3 байта)

Заголовок 2: транспортный уровень

1. Номер транзакции (4 байта)

Заголовок 3: уровень представления

1. Номер алгоритма кодирования (2 байта)

1.4. Реализация физического уровня стека протоколов передачи маркера

Реализация физического уровня в общей памяти (в рамках данной курсовой была выбрана память MSM) базируется на описании в статье "Асимметричный маркерный процессинг" [24]. Она предполагает создание кольцевого буфера для каждого процессора, у которого есть доступ

к памяти. В этих буферах хранятся принятые пакеты. Хранение пакета происходит следующим образом: сначала хранится длина пакета (2 байта), а затем сам пакет.

Доставка пакета происходит благодаря работе двух программ — на физическом уровне передающего процессора программа забирает пакет из внутреннего буфера своего стека и помещает в буфер принятых пакетов принимающего процессора, затем программа на физическом уровне получающего процессора, постоянно проверяющая состояние своего буфера принятых пакетов, обнаруживает в нем новый пакет и начинает его обработку в своем стеке.

Если пакет больше, чем свободная часть буфера, то он удаляется из памяти стека отправляющего процессора (как отправленный) и нигде не остается его копии.

1.5. Возможные применения АМП

Асимметричный маркерный процессинг может быть использован для создания систем требующих применения нескольких вычислителей различной функциональности с целью ускорения выполнения вычислений.

Особенно востребованы такие сложные системы в военной сфере. Например, данный подход может быть применен для наведения ракеты на местности. Схема примерной работы могла бы выглядеть так: есть основной процессор с архитектурой общего назначения (например ARM) он получает данные с камеры и посылает их на более специализированный вычислитель (например DSP), который выполняет свертку Фурье, необходимую для анализа изображения с камеры, и возвращает результаты обратно на первый вычислитель. На основе полученных данных принимается решение о наведении на цель.

Еще одно возможное применение АМП — обфускация. Она может достигаться двумя способами:

1. Часть вычислений может быть перенесена на сторонний вычислитель с прошитым в него микрокодом, тем самым эти вычисления

будут скрыты при попытке анализа программы.

2. Исполняемый файл может быть подготовлен специальным компилятором, который включит в него блоки кода от различных архитектур, а также на границах этих блоков сгенерирует команды передачи маркера с данными о точке входа для продолжения выполнения программы на другом вычислителе.

Оба этих подхода также могут применяться при разработке военных систем.

2. Цели и задачи

Цель данной курсовой работы — реализация части стека протоколов асимметричного маркерного процессинга (физический, сетевой и транспортный уровни) под архитектуру keystone 2 для ядра ARM.

Для их достижения были сформулированы следующие задачи:

1. Исследовать предметную область: провести анализ уже существующих реализаций АМП, а также его аналогов. Изучить устройство используемой архитектуры, а именно понять как ядра процессора могут быстро и эффективно обмениваться информацией друг с другом.

2. Произвести анализ используемых инструментов: выбрать соответствующие инструменты для успешной реализации АМП и запустить первые тестовые программы под архитектуру ARM.

3. Наладить межъядерное общение между архитектурами ARM и DSP и исследовать эффективность этого общения.

4. Реализовать канальную часть физического, сетевой и транспортный уровни из стека протоколов АМП.

3. Обзор существующих реализаций

Асимметричный маркерный процессинг уже был реализован группой научных сотрудников из СПБГУ и УРФУ [24] с использованием языка их собственной разработки — HasCoL [26]. Реализация проводилась на платформе Zybo Zynq-7000 (ядра ARM Cortex-A9 и FPGA). С использованием модифицированного компилятора Clang генерировалось 2 файла — исполняемый код для основного процессора и программа на языке HasCoL, которая впоследствии использовалась для генерации прошивки под ПЛИС.

В рамках работы были проведены замеры времени передачи маркера для 2-х типов памяти: памяти, расположенной на кристалле (Память SoC), и оперативной памяти (DDR3). Результаты измерений представлены в таблице 1, все значения указаны в наносекундах.

Тип памяти	среднее	минимальное	максимальное
DDR3	4204	4060	4364
Память SoC	4553	4408	4662

Таблица 1: результаты замеров времени передачи маркера.

В этой курсовой предполагается реализация схожего поведения на плате с архитектурой keystone 2, что должно значительно уменьшить указанные временные показатели. По оценкам авторов статьи благодаря использованию высокоскоростной памяти на кристалле этого процессора можно снизить временные расходы на передачу маркера до менее чем 250 наносекунд.

4. Обзор аналогов

Существуют два основных механизма по обеспечению управления гетерогенными системами:

Первый — построение системы, как единого целого, с возможностью организации единой вычислительной среды, что позволяет существенно расширить возможности прикладного программиста по взаимодействию с такой системой. Именно такой механизм лежит в основе Асимметричного маркерного процессинга, а также Heterogeneous System Architecture от AMD.

Второй — рассмотрение системы, как набор отдельных частей и последующее их объединение, то есть организация интегральной вычислительной среды. Такие подходы можно часто встретить в современных средах проектирования для гетерогенных систем, таких как Vivado или Code Composer Studio.

Ниже приведен более подробный обзор решений в данной области.

4.1. Heterogeneous System Architecture

Heterogeneous System Architecture (HSA) — подход к созданию гетерогенных систем в рамках которого реализуется единая вычислительная среда для CPU и GPU. Разрабатывается компанией HSA Foundation, возглавляемой AMD. Целями данного проекта является построение системы, которая была бы проста и удобна для прикладного программиста, но в тоже время имела широкие вычислительные возможности благодаря наличию нескольких архитектур.

Основные преимущества данного подхода [5]:

- Создание единой вычислительной среды для CPU и GPU.
- Общее адресное пространство.
- Полная когерентность памяти.
- Высокоуровневая языковая поддержка для вычислительных процессоров GPU.

- Смена контекста и вытесняющая многозадачность.
- Открытые спецификации

К недостаткам проекта можно отнести слабую поддержку платформ отличных от CPU и GPU, а также необходимость поддержки данной технологии на аппаратном уровне.

4.2. Механизм взаимодействия с гетерогенной системой в Code Composer Studio

Texas Instruments — одна из передовых компаний, занимающихся разработкой многоядерных SoC систем. Именно эта компания предложила мощную среду проектирования для таких процессоров — Code Composer Studio. Одним из основных достоинств данной среды является хорошо проработанный механизм взаимодействия со сложными устройствами, производимыми компанией. Он предоставляет программисту массу широких возможностей, таких как [21]:

- Организация интегрированной вычислительной среды.
- Подключение DSP как сопроцессоров.
- Единая среда отладки, включающая контроль блокировок, трассировку программ, визуализацию межпроцессорного обмена информацией.
- Контроль состояния памяти.

Из недостатков системы можно выделить поддержку только RISC архитектур и DSP, а также возможность применения данной технологии только для продуктов компании Texas Instruments.

4.3. Механизм взаимодействия с гетерогенной системой в Vivado

Другая среда проектирования позволяющая программисту успешно работать со сложными системами — Vivado от компании Xilinx. Способ взаимодействия с гетерогенными системами лежащий в основе данной среды позволяет проектировать системы содержащие программируемые логические интегральные схемы (ПЛИС). Его основные преимущества это [8]:

- Организация интегрированной вычислительной среды.
- Подключение IP-ядер в FPGA как сопроцессоров.
- Поддержка языков высокого уровня.
- Унификация интерфейсов RISC-FPGA.
- Единая среда отладки.

Однако у данного решения существует ряд серьезных недостатков таких как нацеленность преимущественно на архитектуры RISC и FPGA, а также сложности при проектировании компактных систем с малой латентностью [8].

5. Стек технологий

5.1. Плата evmk2h

Для реализации курсовой университетом была предоставлена плата evmk2h с архитектурой keystone 2, разработанная компанией Texas instruments [15]. Данная плата имеет характеристики — полностью подходящие под поставленные задачи, а именно на ее кристалле находятся ядра 2-х разных архитектур (ARM и DSP), а также существует специальный высокоскоростной контроллер памяти Multicore shared memory controller (MSMC) [18], который должен обеспечивать хорошую скорость общения между ядрами этих 2-х архитектур. Полная схема процессора этой платы представлена на Рис. 2:

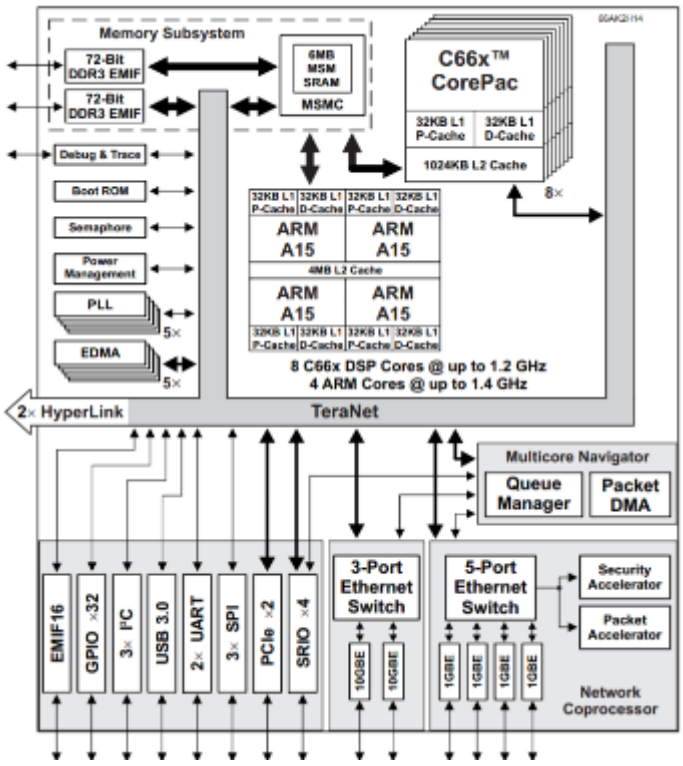

Рис. 2: Схема процессора на плате evmk2h [17].

Еще одним плюсом использования именно этого оборудования является наличие хорошей документации [17, 12] и статей, помогающих начать работу с платой [16, 13, 4].

5.2. Язык

В рамках курсовой было принято решение использовать язык ассемблера для написания кода. Это решение мотивировано, в первую очередь, желанием обеспечить высокую скорость работы АМП, в сравнении с аналогичной реализацией на Си. В предыдущих работах на эту тему указывается, что код на языке Си проигрывает в производительности 1.5 - 3 раза [24].

5.3. Среда разработки

Для разработки была выбрана официальная среда от Texas instruments — Code Composer Studio 9 [14]. Она предоставляет широкие возможности отладки программ на плате, такие как просмотр дизассемблированного кода и состояния участков памяти, а также запуск программы на выбранном ядре кристалла. Еще одним преимуществом является свободный выбор языка разработки (Си или ассемблер) и различных компиляторов к ним.

6. Реализация АМП

Процесс реализации асимметричного маркерного процессинга условно можно разделить на 3 этапа:

1. Изучение окружения, предоставляемого производителем для работы с платой, а также запуск первых программ.

2. Организация передачи пакетов между ядрами ARM и DSP.

3. Непосредственная реализация протокола лежащего в основе АМП

Ниже дано более подробное описание этих этапов.

6.1. Запуск первых программ

Производителем предоставляется широкий спектр возможностей по взаимодействию с платой. Первый и наиболее доступный — это запуск программ через операционную систему Linux установленную на плате. Однако в силу заводского брака, данный способ оказался недоступен для автора, поэтому были изучены другие возможности запуска программ путем непосредственной перепрошивки устройства через отладочный интерфейс JTAG.

В рамках данной задачи были созданы 3 базовые программы [11]: на языке ассемблер, на языке Си под голое железо и на языке Си с использованием конфигурируемой операционной системы sys-bis [19], поставляемой разработчиком. Наибольший интерес представляет последняя программа, автором были изучены модули этой операционной системы отвечающие за вывод печати в консоль и взаимодействие с таймерами. Впоследствии эти модули широко использовались для проводимых временных замеров.

Также на этом этапе были изучены различные варианты расположения программ в памяти кристалла и было принято решение располагать исполняемый код в памяти MSM, а не в DDR3, как это делалось по

умолчанию, так как память MSM представляет собой высокоскоростную статическую память [18], что должно увеличить производительность исполняемых программ.

6.2. Организация общения ядер ARM и DSP

Следующим этапом реализации стала организация передачи тестовых пакетов между ядрами ARM и DSP, а также проведение замеров времени передачи этих пакетов. В связи с тем, что запустить программы, написанные на Си, оказалось гораздо проще, благодаря наличию инструкций от производителя платы [9, 10], данный этап был проведен сначала для программ на Си, а затем для программ на ассемблере, а не только для ассемблера, как это планировалось изначально.

Для проведения измерений были изучены 3 различных способа замера времени: счетчик циклов процессора ARM [3], системный таймер sys-tick [7] и независимый таймер ядра ARM — generic timer [2]. В результате был выбран счетчик циклов, как самый простой в настройке, а также рекомендуемый производителем [19] способ замера времени выполнения программ.

Алгоритм работы измеряемых программ [20] можно описать таким образом: в цикле на 128 итераций повторяется следующее

1. Сброс счетчика циклов процессора ARM.
2. Считывание текущего значения счетчика циклов.
3. Отправка пакета длинной 92 байта со стороны ядра ARM.
4. Ответ DSP пакетом аналогичной длины.
5. Ответ ARM новым пакетом в 92 байта.
6. Считывание текущего значения счетчика циклов.
7. Вычисление результатов замера.

Данный алгоритм полностью повторяет [24] работу аналогичных программ на платформе Zybo Zynq-7000, что позволяет сравнить текущие результаты, указанные в таблице 2, с результатами описываемыми в главе 3.

используемые программы	среднее, нс	мин, нс	макс, нс
C на ARM и C на DSP	—	—	—
C на ARM и Asm на DSP	2589	2575	2625
Asm на ARM и Asm на DSP	2499	2490	2517
Zybo Zynq-7000	4204	4060	4364

Таблица 2: замеры времени передачи пакетов 92(x3) байт.

Как видно из результатов платформа keystone2 действительно значительно выигрывает в производительности перед Zybo Zynq-7000, выигрыш при запуске программ на ассемблере составил порядка 41 процента в сравнении с лучшим результатом описываемом в статье "Асимметричный маркерный процессинг" [24], что дает хороший прогноз на скорость работы АМП.

Позже, для лучшего восприятия, был проведен более подробный анализ измерений, результаты можно видеть в таблице 3, где E — матожидание исследуемой величины, D — дисперсия, σ — среднеквадратическое отклонение, I — доверительный интервал для матожидания, вероятность попадания в который — 95%.

используемые программы	E, нс	I, нс	D, нс	σ, нс
C на ARM и C на DSP	—	—	—	—
C на ARM и Asm на DSP	2603.75	2599.46 : 2608.04	216.15	14.702
Asm на ARM и Asm на DSP	2500.92	2499.18 : 2502.66	42.244	6.5

Таблица 3: статистика времени передачи пакетов 92(x3) байт.

Также на данном этапе было проверено предположение о том, что расположение кода программ в памяти MSM также значительно ускорит выполнение программы. Ниже, в таблице 4, указаны результаты выполнения программ из различной памяти, подтверждающие данное

предположение. Для замеров была использована программа на Си со стороны ARM [20] и программа на ассемблере со стороны DSP.

	E, нс	I, нс	D, нс	σ, нс
MSM	2603.75	2599.46 : 2608.04	216.15	14.702
DDR3	2798.58	2776.84 : 2820.32	14889.33	122.022

Таблица 4: время работы программ в различной памяти.

Кроме того были проведены замеры передачи пакетов маленького размера (12 байт) в одну сторону от ARM к DSP. Необходимость таких замеров обусловлена тем, что в рамках данной реализации АМП планируется передавать пакеты существенно меньшие 92 байт. Результаты можно видеть в таблице 4, значения указаны в наносекундах.

используемые программы	E, нс	I, нс	D, нс	σ, нс
C на ARM и C на DSP	1343.55	1318.43 : 1368.68	20046.34	141.585
C на ARM и Asm на DSP	284.74	281.24 : 288.25	386.11	19.65
Asm на ARM и Asm на DSP	291.88	291.25 : 292.5	5.413	2.327

Таблица 5: замеры времени передачи пакетов 12 байт.

Особенно интересны результаты комбинации программ на Си со стороны ARM и на ассемблере со стороны DSP. Они оказались немного лучше результатов программ на обоих ассемблерах, что показывает целесообразность реализации АМП на Си под ARM, тем более что согласно результатам в таблице 2 при передаче больших пакетов программирование на ассемблере также не дает существенного выигрыша в производительности. Таким образом для базовой реализации АМП было решено использовать язык Си, а не ассемблер, как это задумывалось изначально.

6.3. Реализация протокола

В рамках данного этапа были реализованы [1] все базовые функции трех протоколов АМП — физического, сетевого и транспортного, что

успешно позволило осуществить бесперебойную передачу маркера от ARM к DSP и обратно.

Также были сформулированы дальнейшие направления развития курсовой, а именно предложено встроить полученную реализацию в программу поиска фрейма на изображении, которая задействует ядра ARM и DSP, однако общение в которой происходит другим, более медленным и менее надежным способом.

Кроме того работы по Асимметричному маркерному процессингу могут быть продолжены до полной его реализации под архитектуру keystone2. А также возможен ряд исследований в таких областях как обеспечение еще большей надежности АМП и проектирование отказоустойчивых систем.

Заключение

В ходе курсовой работы выполнены следующие задачи:

1. Проведено исследование предметной области.
2. Изучена архитектура keystone 2.
3. Изучены способы взаимодействия с платой evmk2h.
4. Обеспечена коммуникация между процессорами ARM и DSP и проведена его временная оценка.
5. Реализована функциональность стека протоколов АМП на языке Си для ядра ARM.
6. Сформулированы дальнейшие планы по развитию курсовой.

Список литературы

[1] AMP realisation on github. — 2020 — URL: https://github.com/jackbogdanov/AMP-for-ARM/tree/AMP_develop.

[2] Architecture Reference Manual ARMv7-A and ARMv7-R edition. Chapter B8 The Generic Timer. — 2014 year.

[3] Architecture Reference Manual ARMv7-A and ARMv7-R edition. PMCCNTR, Performance Monitors Cycle Count Register, VMSA. — 2014 year.

[4] EVMK2H Hardware Setup Guide. URL: http://software-dl.ti.com/processor-sdk-linux/esd/docs/latest/linux/How_to_Guides_Hardware_Setup_with_CCS

[5] Ferra, Обзор архитектуры HSA. URL: https://www.ferra.ru/review/computers/AMD-HSA-architecture.htm#HSA_и_технология_SoC.

[6] HSA Foundation member list. Official cite. URL: http://www.hsafoundation.com/members/.

[7] KeyStone Architecture TIMER64P — 2012 year.

[8] Kit-e, Проектирование для ПЛИС Xilinx с применением языков высокого уровня в среде Vivado HLS. URL: https://www.kit-e.ru/preview/pre_40_12_13_VHLS_Xilinx.php.

[9] Processor SDK RTOS Software Developer's Guide. No OS (Bare Metal) Example. URL: http://software-dl.ti.com/processor-sdk-rtos/esd/docs/latest/rtos/index_examples_demos.html#id47.

[10] Processor SDK RTOS Software Developer's Guide. TI-RTOS Kernel Example. URL: http://software-dl.ti.com/processor-sdk-rtos/esd/docs/latest/rtos/index_examples_demos.html#arm-cortex-a15.

[11] Start programs on github. — 2020 — URL: https://github.com/jackbogdanov/AMP-for-ARM/tree/start_progs.

[12] Texas instruments. A Keystone2 EVM Board for TI Product name: K2EVM-HK. rev. 4.0 — 2013 year.

[13] Texas instruments, ARM Assembly Language Tools v19.6.0.STS User's Guide — 2019 year.

[14] Texas instruments, Code Composer Studio official cite. URL: http://www.ti.com/tool/CCSTUDIO.

[15] Texas instruments, EVMK2H evaluation module. Official cite. URL: http://www.ti.com/tool/EVMK2H.

[16] Texas instruments, EVMK2H evaluation module. Quick Start Guide— 2013 year.

[17] Texas instruments. Multicore DSP + ARM KeyStone2 System-on-Chip. — 2012 year.

[18] Texas instruments. Multicore Shared Memory Controller (MSMC) User Guide for KeyStone II Devices. — 2012 year.

[19] Texas instruments. TI-RTOS Kernel (SYS/BIOS) User's Guide. — 2018 year.

[20] Time measument programs on github. — 2020 — URL: https://github.com/jackbogdanov/AMP-for-ARM/tree/time_bench.

[21] Б.Н. Кривошеин. Технология проектирования распределенных вычислительных систем «МАРКЕР» на основе асимметричного маркерного процессинга — г. Алушта: 2017 год.

[22] В.Г. Соломенчук. Железо ПК 2010 — БХВ-Петербург, 2010 год.

[23] М.В. Баклановский, А.Р Ханов. Верификация протокола передачи маркера в стеке асимметричного маркерного процессинга. — Санкт-Петербург.

[24] М.В. Баклановский, Б.Н. Кривошеин, А.Н. Терехов, М.А. Терехов, А.Е. Сибиряков. Асимметричный маркерный процессинг — Уральский федеральный университет.

[25] НГУ. Высокопроизводительные параллельные вычисления на кластерных системах. Материалы пятого Международного научно-практического семинара — Нижний Новгород: Изд-во Нижегородского госуниверситета, 2005 год.

[26] О.В. Медведев. Семантика языка описания аппаратуры HasCoL — Санкт-Петербург: Вестн. С.-Петергбург. ун-та Сер. 10, 2012 год.