Еще одно важное преимущество FPGA – способность обеспечивать обработку данных непрерывно и со стабильной скоростью. Дело в том, что понятие «пиковая производительность» имеет разный смысл для сигнальных процессоров и FPGA. В случае с сигнальным процессором тактовая частота условно соответствует количеству операций в секунду непосредственно с фильтром. Однако в программе процессора могут быть предусмотрены и другие действия, например, обработка прерываний. Из-за необходимости выполнять дополнительные операции пиковая скорость может упасть. Таким образом, понятие "пиковая производительность" имеет статистический характер, а реальное значение производительности может меняться не только в зависимости от выбранного алгоритма, но и в процессе работы программы при возникновении соответствующих условий.

В то же время для FPGA термин «максимальная тактовая частота» относится к наиболее выгодным условиям трассировки кристалла – все соединения выполнены с использованием коротких цепей, связанные программируемые ячейки расположены рядом, максимальная длина цепей ускоренного переноса ограничена (разрядность счетчиков, как правило, невелика). Неудачная трассировка снижает допустимую тактовую частоту, однако весьма важно то, что после завершения проектирования она остается постоянной. Некоторые проблемы с дополнительными тактами ожидания может внести использование внешней памяти. Но наличие скоростных синхронных ресурсов и достаточного количества внутренней памяти существенно облегчает построение стандартных узлов ЦОС. При необходимости в проектах на ПЛИС тоже можно реализовать процессоры (например, процессор на логических ячейках типа MicroBlaze), однако через этот процессор совершенно необязательно пропускать весь поток обрабатываемых данных. Более того, рекомендуется реализовывать высокопроизводительную цифровую обработку с использованием независимых от процессора ресурсов DSP. Процессор может выполнять организацию интерфейса, загрузку коэффициентов и прочие операции, которые сложно реализовать аппаратно. При этом единственное процессорное ядро может обеспечивать управление несколькими сотнями DSP-блоков ПЛИС, которые непрерывно выполняют обработку входного потока [1].

Средства разработки. Программирование DSP под определенную задачу, как правило, выполняется с использованием языка высокого уровня, например C, и с использованием библиотек ориентированных на определенную задачу, например библиотека для использования в приложениях беспроводной связи. Это значительно сокращает время проектирования устройств на базе DSP.

До недавнего времени разработка устройств на базе ПЛИС являлась трудной задачей, требующей больших временных затрат. Однако ситуация изменилась с появлением новых методов проектирования устройств на базе ПЛИС. Одним из новых методов являться написание алгоритма работы устройства на языке высокого уровня с последующей трансляцией программы на уровень регистровых передач. Другим вариантом для сокращения времени проектирования является использование встраиваемых процессоров в ПЛИС. При этом алгоритм пишется на языке высокого уровня, и программа выполняется во встроенном процессоре [3].

Использование ПЛИС с архитектурой FPGA в цифровой обработке на сегодняшний день является наиболее эффективным решением для повышения производительности устройств ЦОС. Данное решение позволяет реализовать методы и алгоритмы, использующие параллельную обработку нескольких потоков данных, тем самым повысив общую скорость вычислений. Так же благодаря новым методам проектирования устройств на базе ПЛИС время разработки устройств на их основе сравнима с временем разработки устройств на базе DSP. Это позволяет говорить, что использование ПЛИС для реализации сложных алгоритмов цифровой обработки сигналов выглядит предпочтительнее, чем разработка на базе DSP.

ЛИТЕРАТУРА

1. Тарасов, И. ПЛИС Xilinx и цифровая обработка сигналов особенности, преимущества, перспективы / И. Тарасов // Электроника: наука, технология, бизнес. – 2011. – №3(00109). – С. 116.
2. Тарасов, И.Е. Программируемые логические схемы и их применение в схемотехнических решениях : учеб. пособие / И.Е. Тарасов, Е.Ф. Певцов. – М., 2012. – 184 с.
3. Шидловский, Д.Ю. Сравнение характеристик ПЛИС и ЦСП для определения целесообразности разработки устройств на их основе в области цифровой обработки сигнала / Д.Ю. Шидловский, М.В. Руфицкий // Труды Международного симпозиума «Надежность и качество». – 2007. – С. 2.

УДК 004.9+004.056

ИСПОЛЬЗОВАНИЕ ПРОТОКОЛА АУТЕНТИФИКАЦИИ OAUTH 2 ПРИ РАЗРАБОТКЕ RESTFUL API

Д.А. САВЧЕНКО
(Представлено: канд. физ.-мат. наук, доц. О.В. ГОЛУБЕВА)

Рассмотрены основные принципы протокола аутентификации OAuth 2. Обращено внимание на основные цели в области обеспечения безопасности предоставления данных сторонним приложениям, достигаемые использованием этого протокола.

Развитие веб-сайтов в настоящее время получило широчайшее распространение; они используются с целью предоставления информации, программных услуг, в качестве средства программного доступа к распределенным приложениям. Развитие веб-сайтов построено на базе протокола HTTP.

Протокол гипертекстовой передачи данных (англ. Hypertext Transfer Protocol – HTTP) является протоколом прикладного уровня, обеспечивающим легкость и скорость, требуемую для распределенных взаимодействующих информационных систем [1]. HTTP, наряду с традиционным использованием для передачи гипертекстовых документов и другими случаями использования, применяется как транспортный протокол для построенных по принципу REST веб-сервисов. REST (Representational State Transfer – «передача репрезентативного состояния») представляет собой архитектурный стиль для построения распределенных систем [2], который широко распространен для разработки программных интерфейсов (англ. Application Programming Interface – API) для интернет-приложений.

Такие программные интерфейсы должны предоставлять способы для сторонних приложений действовать от имени пользователя посредством процесса аутентификации. Метод аутентификации должен обеспечить наивысший уровень безопасности аутентификационных данных и наименьший возможный требуемый уровень взаимного доверия среди сторон процесса аутентификации (иными словами, метод аутентификации должен быть безопасен даже для случаев, когда предоставляющая программный интерфейс система и ее пользователи не доверяют в полной степени клиенту этого интерфейса).

Фактическим стандартом в области аутентификации для RESTful API является OAuth2 [3] – вторая версия открытого протокола авторизации. Этот протокол используют для предоставления доступа посредством программного интерфейса к защищенным пользовательским данным такие технологические компании, как Google, Facebook, GitHub, Amazon и прочие. OAuth 2 предоставляет стороннему приложению возможность получить ограниченный доступ к веб-сервису от имени конечного пользователя (иными словами, владельца защищенных данных). При этом отсутствует необходимость в предоставлении пользовательских данных аутентификации (обычно логина и пароля) стороннему приложению. Это достигается посредством разделения ролей клиента сервиса (стороннего приложения) и владельца защищенного ресурса: вместо использования данных аутентификации владельца ресурса, клиент получает т.н. токен доступа – строку, выражающую определенные границы доступа, временной интервал доступа и прочие атрибуты; токен доступа затем используется для получения защищенных ресурсов.

Обобщенная схема потока аутентификации согласно протоколу OAuth 2 приведена на рисунке 1.

Таким образом, аутентификации состоит из ряда шагов.

Запрос аутентификации. Клиентское веб-приложение выполняет запрос аутентификации к предоставляющему программный интерфейс веб-сервису. Запрос осуществляется опосредовано посредством перенаправления веб-браузера пользователя на специальный URL веб-сервиса. Такой URL является составной частью программного интерфейса веб-сервиса.

Возвращение кода аутентификации. Веб-сервис выполняет авторизацию пользователя стандартным для себя способом (обычно запросом у пользователя через веб-браузер логина и пароля, так же, как и в случае прямого обращение пользователя к веб-сервису; в качестве дополнительной защиты может использоваться двухфакторная аутентификация), явным образом запрашивает у пользователя разрешение на предоставление доступа к его данным указанному клиентскому приложению. В случае согласия пользователя перенаправляет его веб-браузер на специальный URL клиентского веб-приложения, передав параметром этого URL так называемый код авторизации. Такой URL передается клиентским приложением в виде параметра запроса аутентификации. В случае если пользователь не смог выполнить авторизацию, либо отказал в предоставлении клиентскому приложению разрешения на доступ к данным, веб-сервис перенаправляет браузер на тот же специальный URL клиентского приложения, но в этом случае в качестве параметра передает подробную информацию о причинах отказа аутентификации.

Запрос токена доступа. Клиентское веб-приложение выполняет прямой межсерверный HTTP-запрос к специальному URL веб-сервиса с запросом токена доступа на основании ранее полученного кода аутентификации. Этот специальный URL также является частью программного интерфейса веб-сервиса. Ранее полученный код аутентификации передается в виде параметра запроса (либо в теле запроса).

Возвращение токена доступа. Веб-сервис отвечает на выполненный HTTP-запрос, передав в теле ответа токен доступа (авторизационный токен) в случае, если код аутентификации корректен и не просрочен. В противном случае веб-сервис возвратит ответ с соответствующим кодом и сообщением об ошибке.

Обращение к защищенному ресурсу. Клиентское веб-приложение выполняет обращение (посредством прямого межсерверного HTTP-запроса) к ресурсу веб-сервиса, защищенному от прямого доступа. Обращение выполняется к специальному URL, соответствующему требуемому ресурсу в соответствии с программным интерфейсом веб-сервиса. При запросе передается ранее полученный токен доступа, подтверждающий, что этому приложению можно возвратить запрашиваемый защищенный ресурс.

Предоставление защищенных данных. Веб-сервис, проверив переданный токен доступа, возвращает в виде ответа на HTTP-запрос затребованный защищенный ресурс в случае, если переданный токен доступа корректен и не просрочен. В противном случае веб-сервис возвратит ответ с соответствующим кодом и сообщением об ошибке.

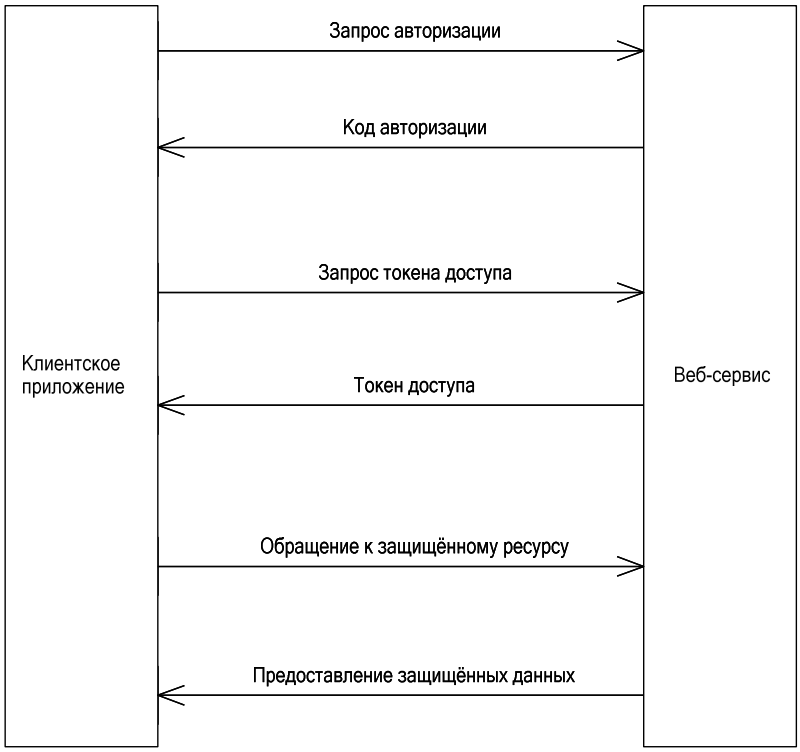

Рис. 1. Схема потока аутентификации OAuth 2

Таким образом, протокол аутентификации OAuth 2 предоставляет относительно высокий уровень безопасности. Он не требует от пользователя веб-сервиса доверять свои данные авторизации стороннему приложению, не требует хранения пароля в обратимой форме на сервере веб-сервиса. Этот протокол предоставляет возможность осуществлять тонкое разделение разрешений доступа посредством границ доступа (пользователь может предоставлять сторонним приложениям не полный доступ к своим данным в веб-приложении, а доступ лишь к требуемой части данных). Протокол обеспечивает явное согласие пользователя на предоставления доступа к своим данным сторонним приложениям. Также OAuth 2 не требует передачи в открытом виде пароля пользователя в ходе аутентификации своих запросов, т.к. для этих целей используется токен доступа, привязанный к конкретному стороннему приложению.

ЛИТЕРАТУРА

1. Berners-Lee, T. RFC 1945 – Hypertext Transfer Protocol – HTTP/1.0 [Electronic Resource] / Tim Berners-Lee, Roy T. Fielding, Henrik Frystyk Nielsen. – Mode of access: http://tools.ietf.org/pdf/rfc1945.pdf. – Date of access: 30.09.2015.
2. Fielding, R. Architectural Styles and the Design of Network-based Software Architectures / Roy Thomas Fielding. – University of California – Irvine, 2000. – 162 p.
3. Hardt, D. RFC 6749 – The OAuth 2.0 Authorization Framework [Electronic Resource] / Dick Hardt. – Mode of access: http://tools.ietf.org/pdf/rfc6749.pdf. – Date of access: 30.09.2015.

УДК 004.9

ИСПОЛЬЗОВАНИЕ RESTFUL API ПРИ РАЗРАБОТКЕ ВЕБ-СЕРВИСОВ

Д.А. САВЧЕНКО
(Представлено: канд. физ.-мат. наук, доц. О.В. ГОЛУБЕВА)

Рассмотрены основные принципы архитектурного стиля REST. Обращено внимание на основные преимущества, достигаемые при проектировании распределенных систем с учетом формальных ограничений REST.