Программное обеспечение для лазерного измерителя деформаций земной коры

Д.О. Терешкин
ФГБОУ ВПО НГТУ, Институт лазерной физики СО РАН. Новосибирск

***Аннотация*: В статье рассматриваются основные задачи, стоящие перед программным обеспечением для обработки сигналов лазерного измерителя, применяемого для измерений деформаций земной коры, а также результаты их решения.** [3]

***Ключевые слова*: программное обеспечение, деформации земной коры, предвестники землетрясений, измерения перемещений, измерения сдвигов**

ВВЕДЕНИЕ

Измерения деформаций земной коры используются для обнаружения предвестников землетрясений, а также в иных исследовательских целях. Разработанный Институтом лазерной физики СО РАН совместно с Опытно-методологической лазерной экспедицией СО РАН измеритель деформаций земной коры (деформограф) имеет несколько версий, которые непрерывно совершенствуются с целью повышения функциональных свойств, надежности, достоверности и так далее. В данной статье описываются модификации программных средств для этого устройства.

1. КРАТКОЕ ОПИСАНИЕ АРХИТЕКТУРЫ СИСТЕМЫ

Общая схема аппаратной части измерителя деформаций (деформографа) приведена на *рисунке* 1. В целом, измерительно-управляющая система состоит из: фотодетекторов, частотно-фазовой системы автоподстройки частоты (ЧФАП), измерителя фазового сдвига (ИФС), модуля GPS и компьютера. ЧФАП управляет пьезоэлектрическим модулятором длины лазера (для изменения его частоты), установленой на гетеродинном лазере, поддерживая разность частот излучения этого лазера и зондирующего лазера. С оптической части на фотодетекторы поступают сигналы опорного плеча, измерительных плеч и компенсационного плеча.

Сигналы с фотодетекторов поступают на входы блока ИФС, всего входов шесть, при измерении часть из них выбирается в качестве опорных, относительно которых вычисляется сдвиг фаз остальных каналов. Полученный поток измерений сохраняется на внутреннем накопителе ИФС, которые могут быть получены по запросу.

Имеется также возможность передачи данных в реальном времени. Передача происходит по линии *Ethernet*, на стандартных протоколах, реализованных поверх стека *TCP/IP* [1]. Это сделано из следующих соображений:

1. Использование *Ethernet* дает возможность объединения устройства в локальную сеть с идентичными или похожими устройствами. Такое объединение может быть продиктовано необходимостью размещения дополнительных регистраторов сейсмических сигналов, оцифровывающих сигналы акселерометров или велосиметров, для получения более полной информации о сейсмической обстановке местности. При этом для построения локальной сети может быть использовано стандартное легкодоставаемое оборудование.
2. Использование стандартных протоколов дает возможность работать с устройством с помощью 3rd-party ПО. Это необходимо для того, чтобы пользователь имел возможность бесшовной интегрирации системы в свои процессы.

Данные могут быть получены по протоколу *Seedlink*, с помощью системы *SeisComp* [2], а также по протоколу FTP, в виде файлов miniseed. Помимо этого, для взаимодействия сервер-ИФС был разработан и реализован специальный протокол, который также позволяет получить поток данных в реальном времени и считывать записанные файлы. Кроме того, блок ИФС предоставляет *telnet*-интерфейс для удалённого мониторинга состояния системы, управления установками регистрации и управления процессом регистрации.

Управляющий сервер на компьютере запрашивает данные, проводит их предобработку и анализ критериев *STA/LTA* [3]. После этого данные сохраняются на жесткий диск, при этом они могут как сохраняться в неизменном виде – в виде *miniseed*-блоков, так и могут быть перекодированы в формат *XX*, совместимый с сейсмостанциями “Байкал”. Сервер имеет возможность получения потоков данных одновременно с нескольких устройств.

Для визуализации поступающих данных в реальном времени создана специальная программа визуализации, запрашивающая данные у управляющего сервера по разработанному протоколу. Она отображает выбранные

[3] Работа выполнена по заданию Министерства образования и науки РФ, проект №7.599.2011, Темплан, НИР № 01201255056.

потоки данных с одного или нескольких узлов, а также дает возможность управления дополнительным пьезоэлектрическим модулятором ЧФАП и мониторинга уровня сигналов фотодетекторов.

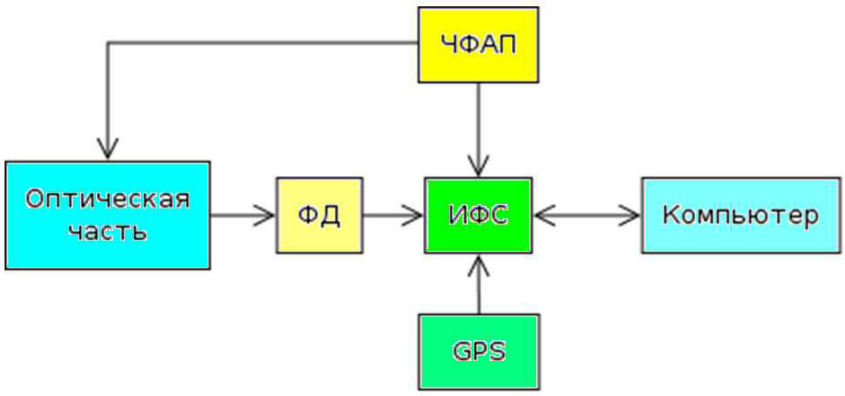

Рис. 1. Схема аппаратной части измерителя деформаций

2. АППАРАТНАЯ АРХИТЕКТУРА ИФС

Конструктивно ИФС состоит из двух плат – платы формирователей фазовых интервалов (ФФИ) и процессорной платы, содержащей программируемую вентильную матрицу (*FPGA*) (см. *рис*. 2). Сигналы с фотодетекторов поступают на входы ИФС, которые соединены с платой ФФИ. ФФИ преобразовывают входные синусоидальные сигналы в импульсную форму – меандры с уровнями КМОП-логики. Преобразованные сигналы передаются на вход *FPGA*, в котором производится измерение разности фаз между заданными каналами и привязка потока получаемых данных ко времени.

Временная привязка осуществляется с помощью цифрового синтезатора частоты, опорной частотой для которого служит локально установленный на плате ИФС термостатированный генератор, управляемый напряжением. Контроллер периодически синхронизует подсистему времени с внешним источником времени - *GPS*. Из-за требования возможности развертывания системы в шахтах, штольнях и прочих местах, где нет возможности для уверенного приёма сигналов *GPS*, синхронизация происходит с помощью сигналов от выносного модуля *GPS*, который находится в зоне видимости спутников. ИФС связан с модулем *GPS* с помощью витой пары. Связь между ними производится по стандарту *RS*-485. Длина линии может достигать не более 500 *м*. Сам модуль посылает по линии раз в секунду посылки, характеризующие состояние приема сигналов *GPS*, текущее время и координаты.

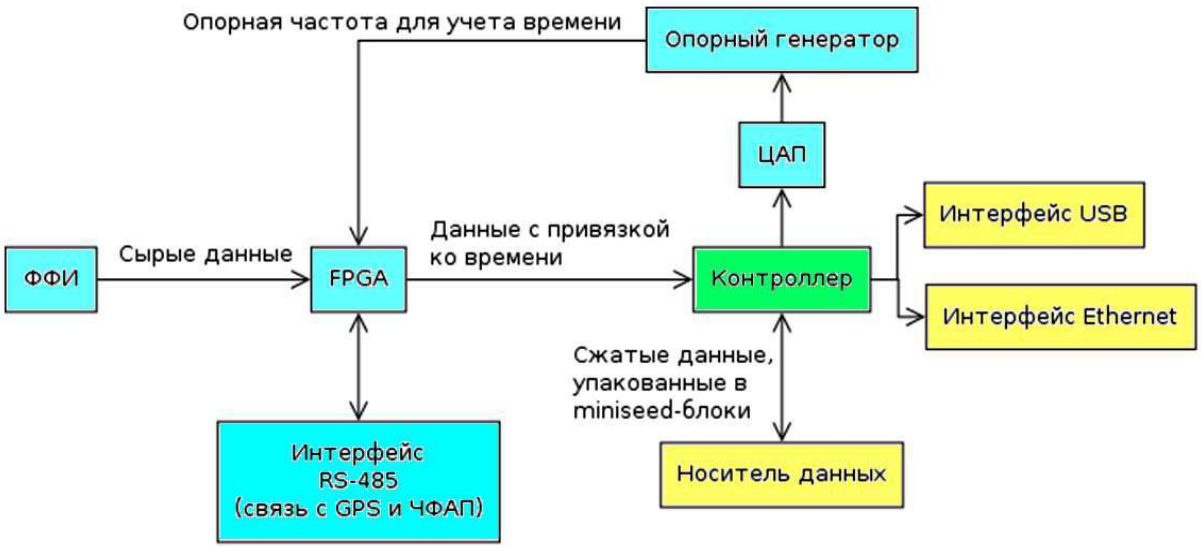

Рис. 2. Схема измерителя фазового сдвига

Данные из *FPGA* передаются в контроллер, где они сжимаются и сохраняются на внутренний носитель. В качестве носителя выступает *MicroSDHC/SDXC*-карта, работающая по стандартному *SD*-интерфейсу [4]. Контроллер обслуживает входящие запросы по интерфейсам *USB* и *Ethernet*. Запросы бывают двух типов – запросы статуса и запросы потока данных. В первом случае контроллер отправляет необходимую информацию непосредственно

после получения запроса, в случае же запроса потока данных контроллер инициирует передачу данных и отправляет их по мере их появления, таким образом, осуществляя передачу в реальном времени.
Также контроллер подключен к системе управления ЧФАП по интерфейсу *RS*-232/485. Это необходимо для мониторинга статуса захвата ЧФАП, а также для управления напряжением на дополнительном пьезоэлектрическом модуляторе (ПЭМ).

ОБЩИЙ ПРИНЦИП ИЗМЕРЕНИЯ СДВИГА ФАЗ

Гармонические входные сигналы поступают на вход одного из шести формирователей, где преобразуются в меандры, с уровнями КМОП-логики. Перед переводом сигналов в импульсную форму, они пропускаются через усилитель, фильтрующий частоты, лежащие вне диапазона 0,7 – 1,2 *МГц*. Основой каждого формирователя служат два компаратора, один из которых переключается по переходам входного сигнала через нуль, а второй при превышении значения регулируемого порога гистерезиса. Меандры с первого компаратора используются собственно как измерительные для формирования фазовых интервалов в ФФИ, а сигналы со второго используются для определения амплитуды сигнала.

Сигналы с формирователей приходят в мультиплексор. В соответствии с конфигурацией, для каждого выходного канала мультиплексора один из входных каналов выбирается как опорный, а другой как измеряемый. Всего выходных каналов шесть, таким образом, на вход ИФС подается шесть пар сигналов, между которыми измеряется фазовый сдвиг.

Для устранения погрешности измерений формируются фазовые интервалы отдельно по фронтам и отдельно по срезам входных сигналов. Количество импульсов тактовой частоты, прошедших за время фазовых интервалов сформированных по фронтам (N_1) и срезам (N_2), складываются. Результат:

$$N = N_1 + N_2,\ N_1 = \sum_{i=1}^{k} N_{\phi i},\ N_2 = \sum_{i=1}^{k} N_{ci} \quad (1)$$

Интервал измерения выбирается следующим образом. Фронт фазового интервала даёт старт измерению. После принятия заданного количества T тактовых импульсов частоты заполнения, схема переходит в режим ожидания $\Phi_{кон}$. Каждый интервал, таким образом, будет отличаться от $T \cdot T_{зап}$, на некоторое количество периодов **d**, зависящее от времени ожидания фронта $\Phi_{кон}$. Следовательно, интервал измерения будет равен:

$$M = T + \delta. \quad (2)$$

Теперь, для того чтобы получить значение разности фаз, нужно вычислить:

$$\Phi = \pi \frac{N}{M} \quad (3)$$

Т.к. отношение δ/T << 1, выражение для φ можно переписать в следующем виде:

$$\Phi = \pi \frac{N}{T} \frac{1}{1+\frac{\delta}{T}} \approx \pi \frac{N}{T}\left(1-\frac{\delta}{T}\right) \quad (4)$$

Из соображений удобства вычислений и быстродействия, T следует выбирать равным двум в целой степени. Это позволяет избавиться от деления при вычислении **φ**, заменив его двоичным побитовым сдвигом.

Полученный код пропорционален разности фаз, которой считается среднее арифметическое разностей фаз измеренных по фронтам и по срезам входных сигналов. Это наиболее верная интерпретация значения **Δφ** для импульсных сигналов, так как вследствие ненулевого напряжения смещения компараторов и возможного искажения формы входных гармонических сигналов, скважность импульсных сигналов может незначительно отличаться от 2. Для уменьшения абсолютной погрешности измерений были использованы логические элементы с малым временем задержки распространения и схемные решения, позволяющие уменьшить или нейтрализовать задержку сигналов в ключевых узлах измерителя.
В процессе регистрации важно иметь данные об амплитуде сигналов, хотя бы на качественном уровне: во-первых, это необходимо для того, чтобы оператор мог предпринять соответствующие действия в случае ухудшения силы сигнала, например, в случае запотевания зеркал. Во-вторых, система должна браковать те измерения, которые были сделаны при низкой амплитуде сигнала, иначе измерения могут дать некорректные результаты. Наиболее простой способ измерять амплитуды сигналов – это добавить на каждый канал по АЦП, однако это слишком сильно усложнило бы схему. Другой способ – поставить один АЦП и подключить его вход к мультиплексору, на входы которого заведены сигналы с фотодетекторов. Этот метод также был отвергнут из-за того, что переключение мультиплексора дает помехи. Измерение амплитуды сигналов в данной системе сделано следующим образом: как было сказано выше, формирователь формирует для каждого канала два импульса – один с помощью компаратора без гистерезиса, другой с помощью компаратора с известным, наперёд заданным гистерезисом.

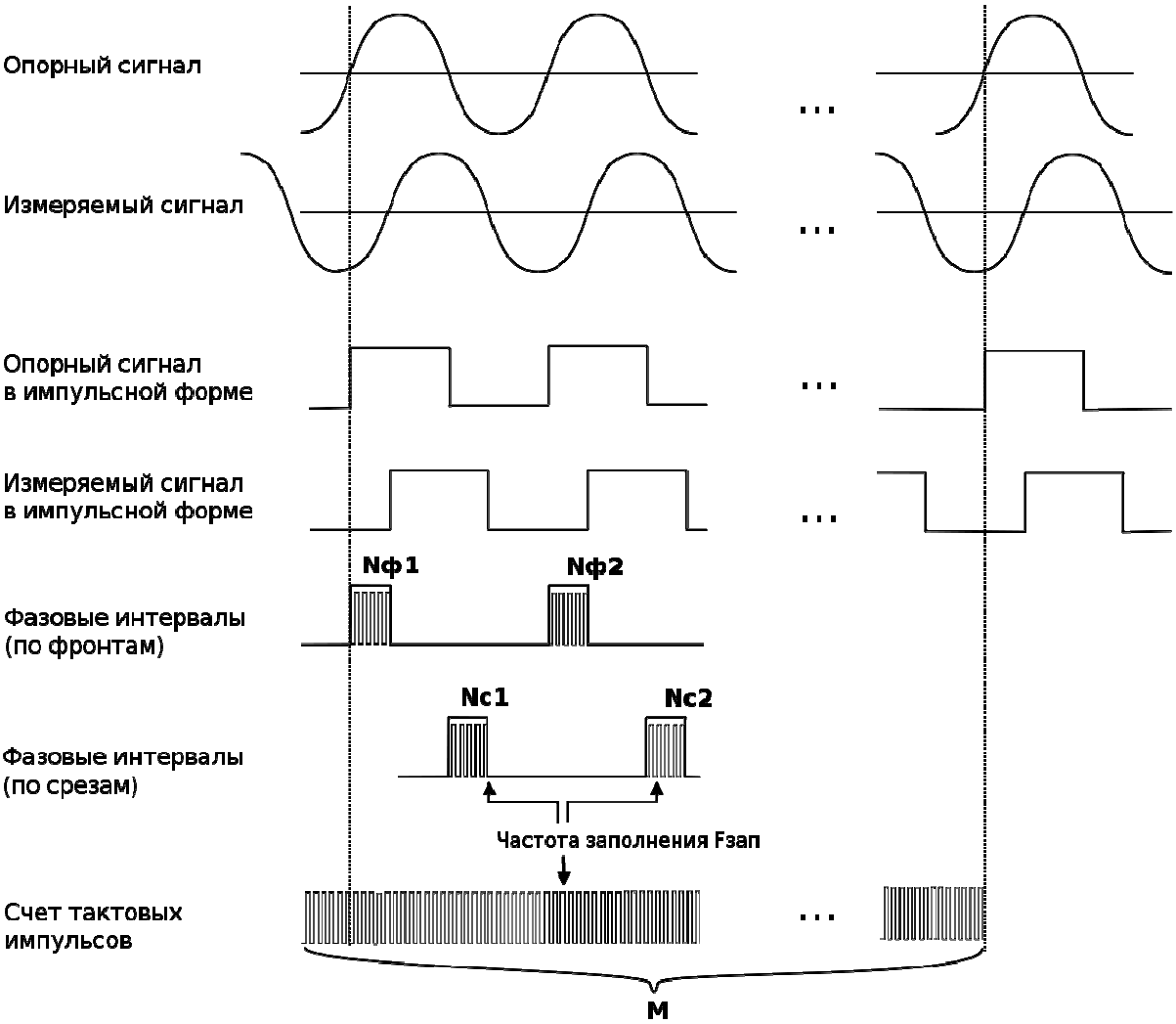

Рис. 3. Принцип счета фазового сдвига

Сдвиг фазы между этими ипульсами:

$$\Delta\varphi = \arcsin\left(\frac{d}{A}\right) = N - c$$

где d – уровень гистерезиса, N – количество импульсов частоты заполнения между фронтами импульсов, c – некоторая константа, связанная с тем, что у первого компаратора имеется гистерезис. Во всей интересующей нас области выполняется:

$$d << A$$

таким образом:

$$A = \frac{d}{N - c}$$

что эквивалентно с точностью до переобозначений:

$$A = \frac{k}{N} - b$$

Коэффициенты k и b легко калибруются по двум точкам характеристики. Таким образом, измеряя количество импульсов частоты заполнения между фронтами соответствующих сигналов можно получить амплитуду входной синусоиды.

ПОДСИСТЕМА ТОЧНОГО ВРЕМЕНИ И ВРЕМЕННАЯ ПРИВЯЗКА ДАННЫХ

При записи данных неминуемо возникает проблема синхронизации. Суть данной проблемы заключается в необходимости сопоставления каждому отсчету времени, в которое он был зарегистрирован, т.е. привязать получаемые данные к шкале *UTC* [5, 6]. При решении этой проблемы необходимо решить следующие вопросы:

1. Источник абсолютного времени. Необходим некоторый объект, который в известный момент времени может сообщить текущее время с приемлемой точностью.
2. Алгоритмы подстройки. При обнаружении рассинхронизации времени системы с *UTC*, необходимы методы плавной коррекции частоты системных часов с целью устранения рассинхронизации.

Для получения абсолютного времени существует лишь два хорошо исследованных способа:

1. Синхронизация по *TCP/IP* с помощью протоколов *NTP/PTP* [7].
2. Синхронизация с помощью глобальных спутниковых систем.

Не останавливаясь подробно на первом варианте, следует отметить, что при его использовании система должна получать время с генератора сигналов времени, называемого *Master clock*, который все равно необходимо синхронизовать. Кроме того, при синхронизации по протоколу *NTP* точность невелика: ошибка может достигать порядка нескольких миллисекунд. Синхронизация же по протоколу *PTP* требует специальной аппаратной поддержки, как в *Ethernet*-коммутаторах, так и в самом устройстве. С другой стороны, синхронизация с помощью *GPS*/Глонасс может быть произведена в любом месте, находящемся в зоне видимости спутников. Этот вариант и использован в данной системе.

В качестве источника времени используется синтезатор частоты, опорная частота которого генерируется генератором, управляемым напряжением, периодически синхронизуемый с *GPS*. *GPS*-модуль вырабатывает сигнал *PPS* (*pulse-per-second*), передний фронт которого привязан к секунде с некоторой, достаточно высокой точностью (около 100 *нс*), при первичной синхронизации по этому сигналу в регистр фазы синтезатора частоты загружается нулевое значение, а в счетчик секунд – текущее время.

Поскольку часто не имеется возможности развернуть систему непосредственно в зоне видимости спутников (например, система может быть установлена в шахте), синхронизация происходит с помощью выносного модуля *GPS*. Этот модуль устанавливается в зоне приема сигнала *GPS* и сначала синхронизуется сам, после чего начинает посылать посылки с информацией о времени раз в секунду. Модуль начинает отправлять сообщение по сигналу *PPS*, но из-за того, что посылка имеет ненулевую длину и задержки распространения сигнала в кабеле, сигнал приходит с задержкой, которую необходимо учитывать. Это делается загрузкой в регистр фазы синтезатора предустановленного значения, вычисляемого с учетом всех задержек на этапе развертывания системы.

Как уже было сказано выше, основой подсистемы точного времени является цифровой синтезатор частоты, выполненный в виде модуля *FPGA* и генератор опорной частоты, управляемый напряжением. При инициализации, контроллер записывает в *FPGA* значения инкремента фазы (*ADDER*) и частоты синтезатора *NTICKS*. По положительному фронту тактового сигнала генератора, аккумулятор фазы синтезатора увеличивается на величину регистра инкремента фазы. При переполнении аккумулятора, счетчик тиков инкрементируется. Когда он становится равен значению *NTICKS*, он сбрасывается, счетчик секунд инкрементируется и в поток данных вставляется метка секунды. Таким образом, время системы задается 32-битным счетчиком секунд, 16-битным счетчиком тиков и фазой синтезатора. Для удобства, время представляется в виде 64-битного значения (см. *рис*. 4): номер секунды и 32-разрядный регистр, старшие 16 бит которого содержат значение счетчика тиков, а в младшие 16 бит отображены старшие 16 бит аккумулятора.

Кроме простого подсчета времени, необходима также синхронизация с *UTC*, т.е. в счетчик секунд должно быть записано количество секунд с начала какого-либо момента, называемого *epoch*, так, широко известна так называемая unix epoch – 1 января 1970. В описываемой системе выбран *GPS epoch* – 6 января 1980, в связи с тем, что источником синхронизации является именно *GPS*.

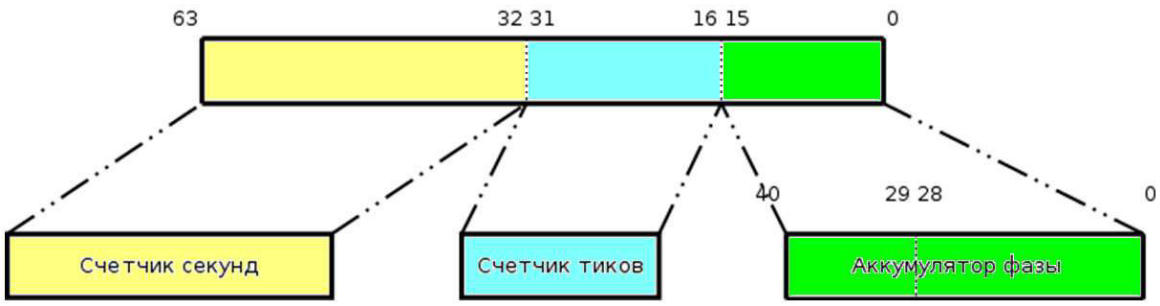

Рис. 4. Структура сигнала представления времени

Сама синхронизация выполняется следующим образом: микроконтроллер записывает в регистры управления подсистемой точного времени бит синхронизации. Если этот бит установлен, то при приходе следующего сигнала *PPS* от модуля *GPS* регистр фазы будет сброшен, таким образом синтезатор частоты будет синхронизован с *UTC* с точностью до целого количества секунд. После этого микроконтроллеру лишь остается записать секунду, пришедшую из модуля *GPS*. По окончании данной процедуры подсистема точного времени становится синхронизированной.

Стабильность опорного генератора находится на уровне 10^{-7}, таким образом, сразу

же после синхронизации система начинает отстраиваться от точного времени. Из-за этого необходимо периодически проводить подстройку. Есть два основных пути выполнения подстройки:

1. Проводить повторную синхронизацию по истечению некоторого периода времени с прошлой синхронизации или же при уходе большем, чем некоторый заданный порог.
2. Плавно изменить частоту синтезатора, чтобы скомпенсировать уход. Это можно сделать также двумя способами:

2.1. Изменять непосредственно регистр приращения фазы.

2.2. Изменять частоту опорного генератора.

В описываемой системе выбран вариант 2.2.

Сами алгоритмы подстройки рассмотрены ниже, здесь будет описано лишь то, что сделано для этого в *FPGA*. Для плавной подстройки необходима возможность измерения расстройки.

Это сделано следующим образом: микроконтроллер записывает регистры управления подсистемой точного времени бит *CAPTURE*. Это приводит к тому, что по следующему фронту *PPS* значение младших тридцати двух бит регистра времени будет сохранено в регистре *CAPTURE*, откуда его может прочитать микроконтроллер. Так как сигнал *PPS* приходит ровно в секунду, то значение этого регистра и будет величиной расстройки, на основании которой можно приложить соответствующее корректирующее воздействие.

СВЯЗЬ С МОДУЛЕМ GPS И СИНХРОНИЗАЦИЯ

Как уже было сказано, синхронизация с *GPS* производится не напрямую, а с помощью модуля *GPS*. Этот модуль представляет собой устройство, на плате которого расположен микроконтроллер, приемник *GPS* и драйверы линии *RS*-485. Модуль связан с ИФС посредством витой пары. Логика работы модуля следующая:

Модуль находится в одном из двух состояний:

1. *NOT_VALID* – нет точного времени: нет сигналов со спутников, блаблабла
2. *VALID* – есть сигналы со спутников, величина смещения UTC получена, точное время получено.

Каждую секунду, по фронту сигнала *PPS*, модуль отсылает сообщения, то, какие именно сообщения отсылаются, зависит от состояния.

Каждое сообщение характеризуется идентификатором типа сообщения и данными. В *таблице* представлены используемые типы сообщений.

Таблица

Тип	Назначение	Формат данных
0x00	Обозначение состояния NOT_VALID	-
0x01	Пакет состояния приема сигналов GPS	TBD
0x08	Широта	IEEE754-32
0x09	Долгота	IEEE754-32
0x0a	Высота над уровнем моря	IEEE754-32
0x0f	Номер секунды GPS	Int32

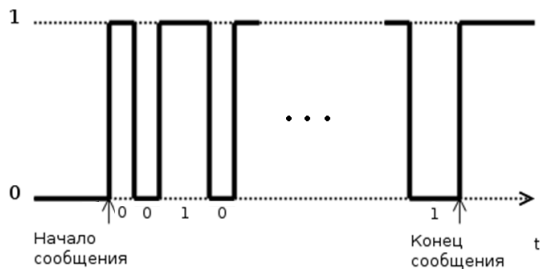

Рис. 5. Эпюры сигналов при передаче сообщения

Физически сообщения кодируются следующим образом: начало посылки начинается с переключения состояния линии в противоположное. Далее, каждый бит кодируется длиной промежутка до следующей инверсии состояния: 1 *мкс* – 0,2 *мкс* – 1. После того, как все сообщение отослано, оно повторяется в инвертированном виде. Это сделано по двум причинам: во-первых, для контроля целостности (если повторенное сообщение после инверсии не совпадает с оригинальным, оно отбрасывается как повреждённое); во-вторых, для того, чтобы длина сообщения не зависела от содержимого. Все данные передаются старшим битом вперёд, порядок байт – *big endian*.

СИСТЕМЫ

Программно система состоит из следующего ПО:

1. ПО измерителя фазового сдвига
2. Сервер сбора данных
3. Клиент визуализации данных

На *рис*. 6 приведена примерная архитектура ПО измерителя фазового сдвига. Как видно, архитектура является трехслойной.

Рис. 6. Примерная архитектура программного обеспечения

АРХИТЕКТУРА НИЖНЕГО СЛОЯ ПО ИФС

Нижний слой состоит из:

1 Слоя абстракции от аппаратного обеспечения (HAL). В функции HAL входит абстрагирование от аппаратных особенностей платформы, на которой реализован ИФС.

2 Драйверов устройств. Сюда входят:

2.1 Драйвера различных накопителей, доступных контроллеру: dataflash, FRAM, SD-карты памяти.

2.2 Драйвер USB.

2.3 Драйвер ЦАП генератора.

2.4 Драйвер GPS.

3 Менеджера памяти.

ОСНОВАНИЕ НЕОБХОДИМОСТИ HAL

HAL предоставляет унифицированные интерфейс, который был выбран в качестве набора возможностей, общих для подавляющего большинства платформ. Общий вид интерфейса и схема его взаимодействия с реализацией и аппаратным обеспечением приведена на *рис*. 7.

Как видно из рисунка N, интерфейс HAL состоит из 6 модулей:

- Serial
- Power
- GPIO
- Systime
- Watchdog
- Debug

SERIAL

Данный модуль предоставляет унифицированный интерфейс для работы с линиями последовательной передачи данных, таких как *Ethernet*, *USB* или *SPI*. Для запроса списка линий и поддерживаемых ими возможностей служит функция *enumerate_lines*.

Приложение может зарегистрировать новую линию с помощью функции *register_line*, это может быть полезным для создания отдельной линии для TCP-подключения. Удалить зарегистрированную линию можно с помощью функции *unregister_line*.

По возможностям линии делятся на побайтовые, поблочные и синхронные. Побайтовые передают и получают данные по одному байту, поблочные – соответственно работают на уровне блоков. Синхронные линии обеспечивают одновременные прием и передачу, как *SPI*. Для работы с байтовыми линиями служат функции *set_byte_isr* и *send_byte* – первая устанавливает функцию-обработчик, которая будет вызвана обработчиком прерывания при приходе байта, вторая посылает байт по линии. Для блочных линий существуют аналогичные функции: *set_block_isr* и *send_block*, работающие уже на уровне блоков. Для работы с синхронными линиями существует функция sync_io, которая принимает в качестве аргументов указатели на входной и выходной буфер (которые могут совпадать). Функция *is_busy* возвращает истинное значение в случае, если линия занята и ложное в обратном случае. Функция *ioctl* служит для *line-specific* запросов.

POWER

Данный модуль служит для управления частотами тактовых сигналов периферии и мастер-частотой. Этот модуль также используется реализацией модуля *Serial* при инициализации линий. Каждый тактовый сигнал характеризуется численным идентификатором. С помощью функции supported_frequencies можно запросить поддерживаемые частоты конкретного тактового сигнала, а используя функцию *set_clock_frequency* – установить требуемую частоту из списка поддерживаемых частот. Функция *current_frequency* возвращает текущую установленную частоту.

GPIO

Как правило, любой контроллер предоставляет возможность прямого управления состояниями некоторого множества своих ножек - *GPIO*, при этом они часто мультиплексируются с сигналами периферии *GPIO*-мультиплексором. Данный модуль служит для управления *GPIO*-мультиплексором.

Для установки направления типа ножки – ввод или вывод – используется функция *set_direction*. Функции *set_pin_state* и *get_pin_state* служат соответственно для установки или запроса состояния ножки. Для переключения функции пина с *GPIO* на какую-либо функцию периферии служит *set_function*.

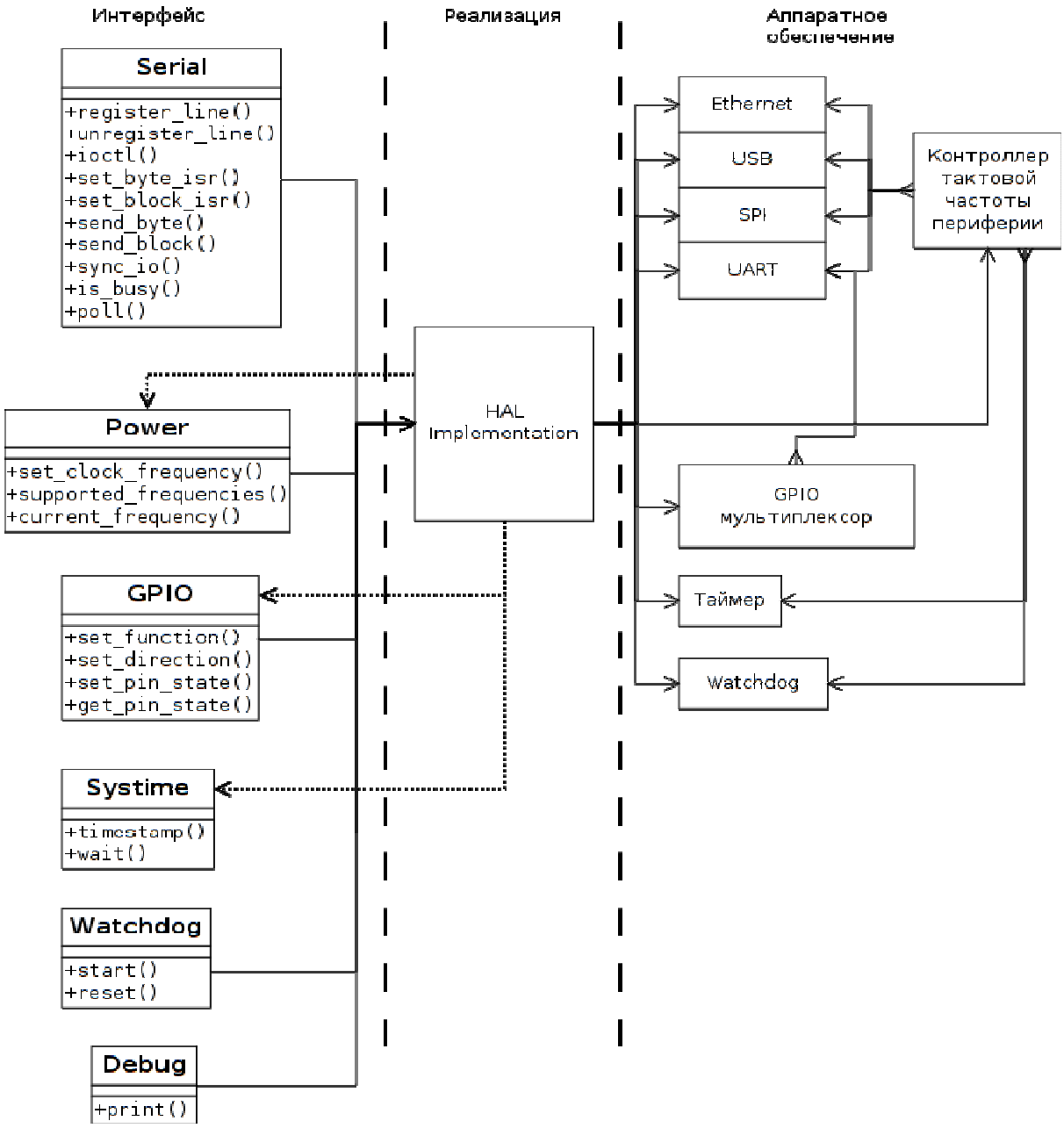

Рис. 7. Общий вид интерфейса

SYSTIME

Этот модуль служит для грубого подсчета времени. Он имеет всего две функции – timestamp возвращает 32-х битное количество миллисекунд, прошедших с начала старта системы. Это значение используется для измерений промежутков времени, где не нужна большая точность – таймауты и т.д. Функция wait вызывает задержку в заданное количество миллисекунд.

WATCHDOG

Данный модуль служит для управления таймером watchdog. Функция start запускает watchdog. Функция reset сбрасывает watchdog.

ДРАЙВЕРЫ УСТРОЙСТВ

Интерфейс драйверов носителей данных. Для унифицированного доступа к носителям, все драйвера носителей данных поддерживают одинаковый интерфейс.

МЕНЕДЖЕР ПАМЯТИ

Используемый менеджер памяти устроен следующим образом. При инициализации создается несколько пулов, с разными размерами блоков. Каждый пул имеет некоторый базовый адрес, относительно которого считаются адреса блоков. Для каждого пула создаются две битовые карты: одна – карта занятости – отображает состояние блоков памяти – свободен / занят, вторая – карта смежности – используется для объединения

нескольких блоков в один. Каждый бит карты занятости сопоставлен одному блоку памяти из пула. Адрес соответствующего блока вычисляется из номера слова в карте и номера бита в слове следующим образом:

$$A = Base + WordNum \cdot N + BitNum \quad (1),$$

где A – адрес блока, *WordNum* – номер слова в *Momega, BitNum* – номер бита в слове, *Base* – базовый адрес пула. Обратное вычисление выполняется следующим образом:

$$WordNum = \left\lfloor \frac{A - Base}{N} \right\rfloor,$$

$$BitNum = A - Base - N \cdot WordNum$$

Помимо карты выделения, для каждого пула создается иерархия дескрипторов, служащих для ускорения поиска свободных блоков. Каждый дескриптор имеет размер машинного слова – 32 бит и делится на 8 частей по 4 бита. Дескрипторы верхнего уровня описывают состояние дескрипторов нижних уровней. Значение фрагментов дескрипторов:

$$D_{n,l}[i] = max_k\left(D_{i,l+1}[k]\right), \quad (2)$$

где D_{n,l}[i] – *i*-ый фрагмент слова *n* на уровне *l*, т.е. в каждом фрагменте записано максимальное значение из фрагментов описываемого данным фрагментом дескриптора. Фрагменты дескриптора нижнего уровня описывают состояние отдельных фрагментов карты занятости, а именно: значение фрагмента равно длине максимальной последовательности нулевых битов в описываемом слове.

На *рис*. 8 изображен фрагмент иерархии дескрипторов для двухуровневой карты занятости.

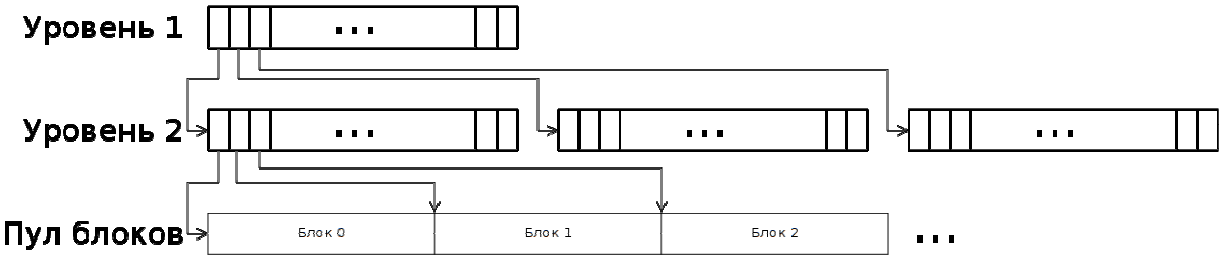

Рис. 8. Фрагмент иерархии дескрипторов для двухуровневой карты занятости

При выделении памяти из пула есть возможность выделить как отдельный блок, так и группу последовательных блоков. Выделение происходит следующим образом: используя иерархию дескрипторов, производится поиск цепочки свободных блоков достаточной длины. Найденые блоки помечаются в карте занятости единицами, а в карте смежности помечаются единицами все блоки в выделяемой цепочке за исключением последнего. После этого производится процедура восстановления инварианта (2).

В случае освобождения блока, процедура деаллокации находит соответствующие биты в картах занятости и смежности, согласно формуле (1). После этого, найденные биты устанавливаются в 0 и производится процедура восстановления инварианта (2).

Ниже приведены алгоритмы выделения и освобождения блоков из пула и анализ их быстродействия.

АЛГОРИТМ ВЫДЕЛЕНИЯ ЦЕПОЧКИ БЛОКОВ ИЗ ПУЛА

// TODO: формализовать нотацию и дать расшифровку

Входные данные: *M* – дескриптор карты занятости, *N* – длина запрашиваемой цепочки в блоках.

1 Если M – дескриптор карты занятости
  1.1 Найти непрерывную цепочку нулей в M.
  1.2 Установить её в 1
  1.3 Установить соответствующие ей биты в карте смежности.
  1.4 Восстановить инвариант (2)
2 Иначе
  2.1 Найти первый M[i], M[i] >= N. Если такого не существует – ошибка.
  2.2 Получить M' – дескриптор, соответствующий M[i].
  2.3 Рекурсивно вызвать данную процедуру с M' и N.

Алгоритм освобождения блока

1. Вычислить WordNum и BitNum по формуле (1)
2. Вычислить длину цепочки N с помощью карты смежности
3. Занулить биты BitNum..Bitnum+N в картах смежности и выделения
4. Восстановить инвариант (2)

Алгоритм восстановления инварианта (2)
Входные данные M – дескриптор карты занятости, из-за которого произошло нарушение инварианта.

1 L = максимальная длина непрервыной цепочки нулей в M
2 Пока M не является top-level дескриптором
  2.1 (M',i) = parent(M)
  2.2 M'[i] = L
  2.3 L = max_k(M'[k])

АНАЛИЗ БЫСТРОДЕЙСТВИЯ

// TODO: Средний слой архитектуры ПО ИФС

1. *TCP/IP*-стек. В качестве стека был использован *TCP/IP*-стек *lwIP* [8], распространяющийся под свободной лицензией на основе *BSD* [9].
2. Виртуальная файловая система (*VFS*).
3. Планировщик ввода-вывода. Его функцией является управление запросами на ввод-вывод от остальных подсистем с помощью специальных алгоритмов для более эффективного использования пропускной способности шины данных накопителя.
4. Подсистему точного времени. Её основной функцией является подстройка генератора, с помощью которого считается время и предоставление текущего времени остальным подсистемам.

ВИРТУАЛЬНАЯ ФАЙЛОВАЯ СИСТЕМА

Обоснование необходимости в такой системе состоит в том, что используется несколько носителей и несколько ФС.

Вся работа с файлами – создание, запись, чтение – идет через слой виртуальной файловой системы. Это дает два основных архитектурных преимущества: во-первых, обеспечивается униформный доступ к разным файловым системам, во-вторых, уничтожается зависимость сервисов верхних слоев от конкретных ФС. ВФС предоставляет возможность монтирования и демонтирования ФС в общий корень. После монтирования, работа с ФС выполняется с помощью интерфейса ВФС, который делегирует свои вызовы драйверам конкретных примонтированных ФС.

МОДЕЛЬ ОПЕРАЦИЙ ДРАЙВЕРОВ ФС

Перед монтированием ФС, драйвер заполняет структуру *vfs_filesystem*, содержащюю указатели на функции интерфейса ФС и указатель на сам драйвер ФС, после чего эта структура добавляется в список ВФС. Файлы и директории внутри ФС идентифицируются стандартным путем (слэши, символы и т.п.) относительно корня. Так как вся работа с ФС происходит через интерфейс ВФС, то полный путь ВФС состоит из номера ФС и пути файла в ФС.

Всего было выделено 3 подмножества операций, которые должны поддерживать драйвера ФС, чтобы иметь возможность быть использованными с помощью ВФС: операции с ФС, операции с файлами и итерация по ФС. Операции с ФС включают в себя: создание и удаление файлов и директорий, операцию проверки существования файла или директории, а также операцию синхронизации, которая блокирует выполнение программы пока все асинхронные операции с ФС не будут завершены.

Операции с файлами включают в себя: операции открытия и закрытия файла, операции чтения и записи и операции перемещения по открытому файлу. Операций чтения две – одна из них просто считывает данные из файла в предоставленный буфер, другая же возвращает указатель на внутренний буфер драйвера.

Операции итерации включают в себя три функции: *fs_iterate_start* принимает в качестве аргумента путь до директории, в которой будет производиться итерация и возвращает итератор. Функция *fs_iterate_next* принимает в качестве аргумента итератор и возвращает информацию о следующем файле, если таковой имеется, в противном случае считается, что итератор завершил перебор директории. Такой итератор должен быть передан функции *fs_iterate_end* для освобождения занимаемой им памяти и прочих ресурсов.

ПЛАНИРОВЩИК ВВОДА-ВЫВОДА

Существует две основных модели ввода-вывода для блочных носителей данных – асинхронная модель и синхронная. В синхронной модели поток выполнения при запросе операции ввода-вывода приостанавливается до тех пор, пока операция не будет завершена. При старте синхронной операции ввода-вывода управление передается ядру, которое выполняет соответствующий запрос драйверу устройства и возвращает управление при завершении операции. В асинхронной модели поток выполняет запрос к подсистеме ввода-вывода и продолжает свое выполнение без блокировки, при этом он в любой момент времени может определить, закончилась ли запрошенная операция. Обе модели имеют свои преимущества и недостатки: с одной стороны, использование синхронной модели упрощает написание программы, с другой стороны, асинхронная модель позволяет достичь более высокой производительности [10].

В данной системе реализованы обе модели. Это дает возможность использовать ту модель ввода-вывода, которая более подходит для решения той или иной задачи. На нижнем уровне сихронные вызовы транслируются в асинхронные запросы следующим образом:

• При синхронном запросе на запись, подсистема ввода-вывода определяет, нет ли активного в настоящий момент асинхронного запроса на запись, если он есть, то выполнение блокируется до тех пор, пока он не будет выполнен. Далее, подсистема ввода-вывода формирует новый асинхронный запрос к драйверу устройства и возвращает управление основному потоку, не блокируя.

• При синхронном запросе на чтение, подсистема ввода-вывода опять проверяет, нет ли активного запроса на запись, а также блокирует выполнение, пока он не будет выполнен, после чего формирует новый асинхронный запрос на чтение и ждет, пока он не будет выполнен. Как только драйвер устройства сигнализирует о том, что запрос выполнен, подсистема ввода-вывода возвращает управление основному потоку.

Такая трансляция приводит к тому, что при выполнении операции записи, основной поток продолжает выполняться одновременно с записью и блокируется только тогда, когда это необходимо – операция чтения блокируется всегда, т.к. прочитанные данные могут быть использованы непосредственно после операции чтения, однако для одиночной операции записи таких ограничений нет. Таким образом, ввод-вывод является на самом деле частично асинхронным, но выглядит для основного потока как синхронный.

Асинхронный запрос описывается структурой, которая содержит следующие поля:

1. Указатель на буфер данных и его длину
2. Логический адрес блока, на котором производится операция.
3. Метка времени постановки запроса в очередь.
4. Флаги запроса: флаг направления запроса – запись-чтение, флаг владения буфера – если этот флаг установлен, то считается, что буфером владеет подсистема ввода-вывода, и она должна освободить буфер после завершения операции, флаг завершения – устанавливается при завершении запроса вне зависимости, был ли он успешным, флаг ошибки – устанавливается в случае ошибки.

Асинхронная модель дает преимущество в производительности, т. к. процессор может выполнять инструкции параллельно с вводом-выводом. Кроме того, т.к. асинхронная модель не налагает никаких ограничений на порядок выполнения операций, подсистема ввода-вывода может изменить порядок выполнения запросов, передав их драйверу устройства в порядке, отличном от того, в котором они были переданы подсистеме ввода-вывода. Конкретный набор оптимизаций, которые следует применять, зависит от типа носителя.

В данной системе в качестве носителя, на котором хранятся данные, служит *MicroSD*-карта памяти. Работа с *SD*-картами памяти может быть осуществлена как через последовательный интерфейс *SPI*, так и через параллельный интерфейс *SD/MMC*. В данной системе связь с картой памяти производится по интерфейсу *SD*, т. к. именно в этом режиме возможно достижение максимальной производительности ввода-вывода. Работа с картой памяти производится по известному протоколу, который определяет некоторое множество команд, из которых используются следующие:

- CMD18 – чтение диапазона блоков
- CMD25 – запись диапазона блоков
- CMD12 – завершение ввода-вывода
- CMD13 – запрос статуса *и т.д*.

При работе с *SD*-картой, для достижения максимальной производительности операций записи/чтения необходимо учитывать три особенности:

1. Активная операция ввода-вывода продолжается до тех пор, пока карте памяти не будет послана команда 12. Более новая версия стандарта *SD* 3.0 определяет команду 23, с помощью которой можно задать заранее количество блоков для операции записи/чтения, однако на момент написания данной работы эта версия стандарта пока ещё не получила широкого распространения в виде поддержки в периферии контроллеров. Таким образом, любая операция записи/чтения происходит следующим образом: контроллер посылает карте памяти соответствующую команду и устанавливает указатель *DMA*-контроллера на начало буфера, а в регистр счетчика *DMA*-контроллера загружается размер передаваемых или прочитываемых данных. После того, как *DMA*-контроллер передал или прочитал все данные, он тригерит прерывание, по которому контроллер посылает команду 12.

2. *SD*-накопители используют технологию *NAND*, что означает, что для перезаписи определённого блока сначала требуется его стирание. Кроме того, это также означает, что более используемые блоки будут изнашиваться быстрее и в течение времени жизни носителя необходимо переотображать испорченные блоки в новые. Внутренний контроллер *SD*-карты выполняет эти операции самостоятельно – он поддерживает карту отображений из логического адреса в физический и стирает при перезаписи нужный блок самостоятельно, если это необходимо, повышая количество времени, требуемое на операцию записи. Этого можно избежать, стирая заранее те области, которые на данный момент уже не нужны, но которые понадобятся в будущем. Для такого стирания стандарт *SD* определяет команду, которой в качестве аргумента передается начальный и конечный адреса диапазона блоков, которые будут стёрты.

3. При завершении операции ввода-вывода, карта памяти переходит в режим *BUSY*, в котором она обновляет свое внутреннее состояние и не может выполнять остальные

операции, особенно это актуально для операций записи. Таким образом, множество мелких запросов на ввод-вывод могут выполняться значительно дольше одного крупного, даже суммарный объем данных в первом случае равен объему данных запроса во втором случае. Из этого следует, что чем большими блоками производится ввод-вывод, тем выше производительность.

Исходя из этих особенностей, можно разработать алгоритм планировщика, который использует их для увеличения производительности по сравнению с обычным последовательным алгоритмом. Основные идеи две: во-первых, если два запроса затрагивают смежные области, их можно объединить в один – таким образом можно избежать задержки, которая была бы между этими запросами в случае их обычного выполнения. Необходимо учитывать, что такое объединение возможно только в случае, если направление запросов одинаковое, т.е. нельзя объединить запрос на запись с запросом на чтение. Обычно, как запись, так и чтение производятся последовательно, т.е. при записи данных они записываются по порядку в каждый блок, без пропусков, таким образом, эта оптимизация годная.

Во-вторых, существуют блоки, на которых часто выполняются *Read-Modify-Write* (*RMW*) операции, т. е. блок загружается, изменяется его часть (как правило, небольшая – несколько машинных слов) и изменённый блок записывается назад. Примером такой операции может служить изменение размера файла в соответствующей ему записи в директории при добавлении данных в его конец. Можно избежать большинства операций записи-чтения в данном случае, если организовать ввод-вывод через кэш блоков, который работает следующим образом: при чтении блока сначала просматривается список кэшированных блоков, и если запрошенный блок найден, он возвращается, в противном случае он читается с носителя и его копия помещается в кэш. При записи, блок помечается как “грязный”. Когда при очередном чтении блока происходит *cache miss* и свободных записей в кэше нет, то выбирается одна из существующих записей, проверяется, содержит ли она “грязный” блок, если да, то блок записывается в свое место, а запись кэша ассоциируется с новым прочитанным блоком. Фактически, кэш использует политику сквозной записи, за исключением того, что блоки периодически записываются на носитель, даже если они не выгружаются из-за чтения нового блока.

ПОДСИСТЕМА ТОЧНОГО ВРЕМЕНИ

Подсистема точного времени выполняет несколько функций, связанных с синхронизацией и учетом времени, а также декодированием сообщений модуля *GPS*.

Как было сказано выше, сообщения модуля *GPS* приходят по линии *RS*-485 в *FPGA* и декодируются там из физического представления в сообщения. Подсистема точного времени непрерывно считывает эти сообщения из регистров *FPGA* и декодирует их.

При инициализации, система начинает отсчет времени с нулевого значения. Это продолжается до тех пор, пока не будет получено сообщение времени от модуля *GPS* с номером текущей секунды. После этого, подсистема точного времени устанавливает соответствующий флаг в регистрах *FPGA*, что приводит к тому, что синтезатор частоты будет синхронизирован по следующему секундному сообщению модуля *GPS*. После этого, полученное значение секунды записывается в регистр счетчика секунд. С этого момента система считается синхронизированной. При синхронизации регистрация останавливается и перезапускается.

В процессе работы системы, сигнал *GPS* может пропадать и появляться, вследствие изменения положения спутников на орбите Земли. Пропадание сигнала не означает потери точного времени, т.к. как было сказано выше, стабильность опорного генератора частоты находится на уровне 10^{-7}. Тем не менее, чем больше промежуток времени, в течение которого система работала без подсинхронизации, тем больше отстройка времени от истинного значения. Стандарт *MiniSEED* позволяет указывать величину, называемую *time_quality*, характеризующую качество времени в момент записи данных. Метод расчета этой величины оставляется на усмотрение источника данных, но значения должны лежать в диапазоне от 0 до 100. Эта величина также расчитывается подсистемой точного времени следующим образом: $min\left(100-\frac{\Delta\cdot 10^{-7}}{f},0\right)$, где Δ – количество секунд с последнего момента *VALID*, f – частота дискретизации.

АЛГОРИТМЫ ПОДСТРОЙКИ ВРЕМЕНИ

Как уже было сказано выше, при первичной синхронизации регистр фазы синтезатора частоты сбрасывается, после чего синтезатор частоты начинает выдавать периодический сигнал, с помощью которого ведется учет времени в системе. Из-за нестабильности опорного генератора время системы расстраивается относительно времени *UTC*, поэтому возникает необходимость плавной подстройки частоты. Для этого мы должны зафиксировать значение регистра фазы в момент прихода сигнала *PPS* и сравнить его с нулевым значением и на основании результата изменить напряжение так, чтобы уменьшить рассинхро-

низацию. Существует несколько алгоритмов, по которым может производиться такое изменение.

ПИД-РЕГУЛИРОВАНИЕ

Одним из очевидных методов является управление напряжением генератора с помощью ПИД-регулятора. Для этого, мы периодически защелкиваем значение регистра фазы по сигналу *PPS*, получая текущую расстройку системных часов *d*, после чего подаем напряжение на генератор опорной частоты:

$$V = C_p \cdot d + C_i \cdot A + C_d \cdot \frac{\delta d}{\delta t}$$

$$A = \sum_0^T d(t)$$

где ,

т.е. *A* является переменной-аккумулятором, в которой мы накапливаем значение *d* на каждом шаге. Используя данный алгоритм, следует учитывать джиттер сигнала *PPS*. Типичное значение джиттера равно 100 *нс*, таким образом, выполнять подстройку каждую секунду нецелесообразно, поэтому лучше всего делать это каждые несколько секунд.

В реальных системах, как правило, не имеет особого значения длительность первого переходного процесса. Гораздо важнее, чтобы система правильно подстраивала выходное напряжение при необходимости, поэтому для данного метода дифференциальный тракт является необязательным.

Однако данный метод дает плохие результаты в условиях плохого приёма *GPS*-сигнала. Если сигнал пропадёт до окончания переходного процесса и будет отсутствовать слишком долго, расстройка может оказаться неприемлемо большой (см. *рис*. 9).

ОДНО- / ДВУХ- / ТРЁХ-ФАЗНЫЙ МЕТОД

Следующая группа методов основана на линейной модели генератора. Предположим, что расстройка генератора может быть представлена в виде:

$$\Delta = T \cdot k \cdot (v - v_0),$$

где T – период, на котором проводится измерение, v – управляющее напряжение генератора, Δ – уход времени за время T, k, v_0 – константы модели, v_0 соответствует напряжению, при котором генератор генерирует точно необходимую частоту, k является наклоном характеристики скорость ухода - напряжение.

Таким образом, вычислив коэффициенты модели k и v_0 и измерив расстройку времени, мы можем расчитать корректирующее значение напряжение, т.е. такое напряжение, при котором расстройка плавно обратится в ноль за некоторое время τ:

$$v = v_0 + \frac{\Delta}{\tau k}$$

Таким образом, можно использовать следующий алгоритм:

1. Зафиксировать значение расстройки.
2. Расчитать корректирующее значение по вышеприведенному соотношению и подать его на генератор.
3. Подождать τ.
4. Повторить.

Нетрудно заметить, что данный алгоритм страдает от того же самого недостатка, что и предыдущий – если в момент фиксации сигнал PPS будет отсутствовать достаточно долго, то системное время может уйти вперед или назад значительно, т.к. на генератор будет подаваться корректирующее значение, хотя коррекция уже закончена (см. *рис*. 9).

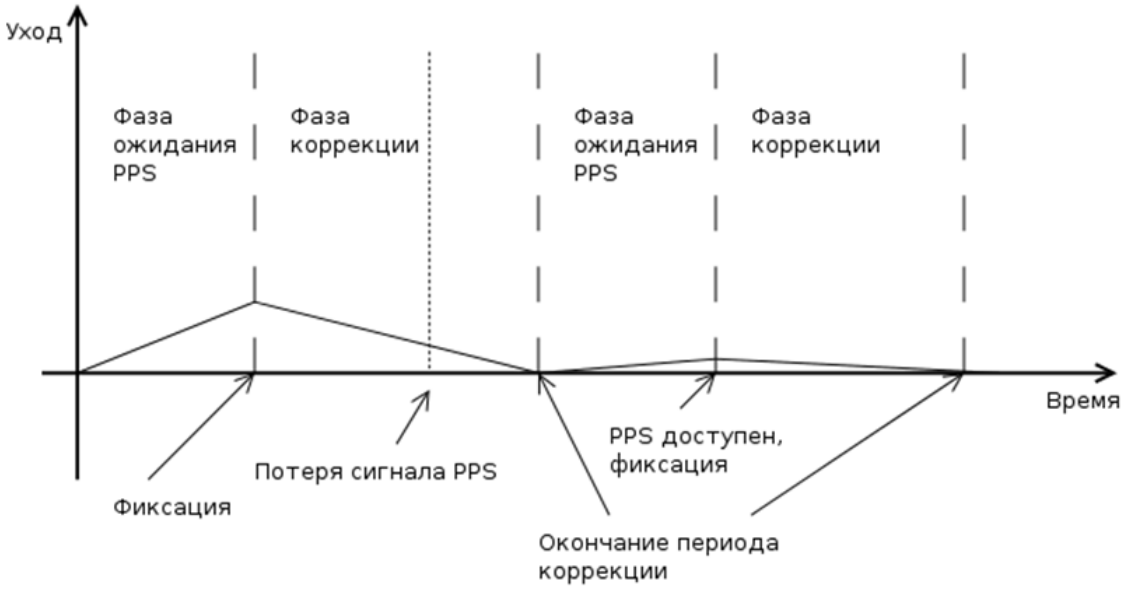

Рис. 9. Распределение фаз сигнала во времени

Эту проблему можно легко исправить следующим образом, если учесть, что в нашей модели v_0 является напряжением нулевого ухода:

1. Ожидать доступности сигнала PPS
2. Зафиксировать значение расстройки
3. Расчитать корректирующее значение и подать его на генератор
4. Подождать τ
5. Подать на генератор напряжение V0
6. Повторить

Усовершенствованный алгоритм состоит теперь из двух фаз – фазы коррекции и фазы ожидания *PPS*. Теперь, даже при редкой доступности сигнала *GPS* подстройка будет выполняться корректно. Несмотря на то, что после коррекции мы устанавливаем напряжение нулевого ухода, из-за нестабильности опорного генератора по прошествии достаточно большого времени системное время будет рассинхронизовано, поэтому данный алгоритм следует повторять в бесконечном цикле на всем протяжении работы системы.

Коэффициенты модели k и v_0, как правило, задаются на этапе калибровки и считаются постоянными на всем протяжении работы прибора. Это не совсем корректно, т.к. в этом случае не учитывается старение генератора и прочие изменения характеристик с течением времени, изменением температуры и прочих условий окружения. В связи с этим имеет смысл добавить в алгоритм ещё одну фазу - фазу коррекции модели:

1. Ожидать доступности сигнала PPS
2. Зафиксировать значение расстройки
3. Подождать $\tau_{мод}$
4. Ожидать доступности сигнала PPS
5. Измерить время $\tau_{изм}$, прошедшее с момента прошлой фиксации
6. Зафиксировать значение расстройки
7. Вычислить новое значение v0
8. Расчитать корректирующее значение
9. Подождать $\tau_{корр}$
10. Установить значение V0
11. Повторить

Здесь под $\tau_{мод}$ и $\tau к_{орр}$ мы понимаем период коррекции модели и период коррекции ухода генератора соответственно. Период коррекции модели может быть не очень большим, на практике, значение 30 секунд дает приемлемые результаты.

ВЕРХНИЙ СЛОЙ АРХИТЕКТУРЫ ПО ИФС

Верхний слой содержит те сервисы, которые выполняют полезную работу:

1 Хранилище данных.
2 Индекс.
3 Miniseed-энкодер.
4 Сервер команд
5 TCP/IP-серверы.

ПОДСИСТЕМА ХРАНЕНИЯ ДАННЫХ

После оцифровки и кодирования данные сохраняются в хранилище данных. Размер блоков *miniseed*, создаваемых энкодером равен размеру страницы на *SD*-карте, используемой в качестве носителя, таким образом, каждая страница может содержать отдельный блок данных. Т.к. каждый *miniseed*-блок содержит в своем заголовке все необходимые метаданные, чтение данных может начинаться и заканчиваться на любых блоках.

Подсистема хранения данных создает на носителе три области: контрольный блок, область хранения данных и область лога. Контрольный блок содержит в себе информацию о текущем состоянии подсистемы хранения данных:

- Текущий номер блока
- Максимальный номер блока
- Диапазоны адресов страниц, отведённых для области хранения данных
- Размер кластера
- Идентификатор первого блока в области хранения данных
- Указатель заполнения области хранения данных
- Контрольная сумма контрольного блока *<lolwut>*

Область хранения данных представляет собой кольцевой буфер блоков – запись блоков происходит последовательно, при записи последнего блока (переполнении), указатель записи сбрасывается на начало буфера. До переполнения указатель заполнения области хранения данных указывает на последнюю страницу, содержащую данные, после переполнения он указывает на последнюю страницу области хранения данных. В связи с тем, что запись происходит не одиночными блоками, а кластерами, размер области хранения данных выбирается таким образом, чтобы он был кратен размеру кластера.

В области лога хранятся сообщения системы об основных произошедших событиях. Эти сообщения могут быть скачаны позже по сети и обработаны специальной программой для просмотра. Область лога состоит из 64-байтных записей, содержащих метку времени, метку подсистемы, которая записывает сообщение и собственно сообщение в кодировке *UTF*-8. Кроме того, первая запись в логе содержит информацию о состоянии этой области: начальная и конечная страницы области лога, а также текущая позиция для записи. Область лога организована в виде кольцевого буфера. Все сообщения лога также попадают в промежуточный буфер. При достижении определённого количества сообщений в этом буфере, создается новый *miniseed*-блок с типом

кодирования <*ASCII*>, в который записываются накопленные сообщения и который сохраняется как очередной блок в области хранения данных. Это позволяет получать сообщения лог-файла в системах использующих только *Seedlink*.

В процессе инициализации, подсистема хранения данных проверяет, существует ли на носителе валидный контрольный блок, и если нет, то создает его, в этом случае указатели записи устанавливаются в начальное значение и область хранения данных считается пустой. Валидность контрольного блока определяется по двум факторам: сигнатура контрольного блока и контрольная сумма. Контрольная сумма считается как сумма всех машинных слов, из которых состоит контрольный блок, за исключением слова, содержащего саму контрольную сумму.

Интерфейс подсистемы хранения данных состоит из нескольких функций, наиболее важными из них являются две:

- Функция сохранения блока – *datastorage_store_block*
- Функция запроса блока по его номеру – *datastorage_retrieve_block*

Функция сохранения блока принимает в качестве аргумента буфер с сохраняемым блоком, сохраняет его в области хранения данных и возвращает числовой идентификатор сохранённого блока.

Нумерация блоков начинается с 0 и идет последовательно до $2^{24} - 1$, после чего следующему блоку присваивается номер 0 и нумерация начинается сначала. Запись блоков на носитель производится кластерами с фиксированным размером страницы, который указан в контрольном блоке подсистемы хранения данных. По умолчанию это значение равно 32. Текущий кластер хранится в буфере, в который копируются сохраняемые блоки, когда текущий кластер полностью заполняется, формируется асинхронный запрос на запись текущего кластера, счетчик кластеров инкрементируется и для нового кластера выделяется новый блок памяти. Также, каждый сохраняемый блок сохраняется в кэше блоков ограниченного размера, при этом из кэша удаляется самый старый из сохранённых в нём блоков. Кэш блоков реализован как простой односвязный список пар идентификатор-блок данных.

Функция чтения блока принимает в качестве аргумента номер блока и возвращает блок данных с запрошенным номером, если он существует в хранилище данных. При этом возможны следующие варианты:

1. Блок с запрашиваемым номером находится в кэше. В этом случае, блок копируется из кэша. Никакие операции чтения с носителя в данном случае не производятся. Это позволяет повысить производительность системы при потоковой передаче данных в реальном времени.

2. Блок с запрашиваемым номером находится в основном буфере чтения. В этом случае, блок копируется из этого буфера. В этом случае также никакие операции с носителем не производятся.

3. Блок с запрашиваемым номером находится в дополнительном буфере чтения. В таком случае буферы меняются местами: основной буфер становится дополнительным и наоборот. Далее инициируется асинхронный запрос на чтение следующей порции блоков, идущих после тех, что находятся в основном буфере. Это считывание производится в дополнительный буфер. Такая схема с двойным буфером позволяет оптимизировать производительность последовательного чтения блоков, т.к. в этом случае идет предварительная загрузка блоков, которые будут нужны в ближайшем будущем, что позволяет избежать ожидания при их считывании, когда они понадобятся.

4. Блок с запрашиваемым номером не найден ни в кэше, ни в буферах чтения. В этом случае в основной буфер считывается последовательность блоков, начинающаяся с запрошенного. После того, как чтение завершилось, возвращается первый блок и инициируется асинхронное чтение следующей порции в дополнительный буфер.

5. Блок отсутствует в области хранения данных. В этом случае процедура чтения завершается с неудачей. Это может означать, что запрошеный блок либо пока ещё не был сохранён, либо был уже перезаписан новым. В зависимости от этого, при запросе потоковой передачи с несуществующим начальным номером, передача начинается либо с самого старого, либо с самого нового сохранённого блока.

При чтении блока с носителя необходимо конвертировать номер блока данных в номер страницы носителя. При этом необходимо учитывать две возможности: общее количество страниц в области хранения данных T может быть как меньше диапазона идентификаторов, так и больше – в зависимости от размера носителя.

Алгоритм для этой конверсии следующий: пусть M – максимальное значение идентификатора блока. Тогда:

1 $T < M$
- 1.1 Указатель заполнения < Конец ОХД>
  - 1.1.1 end_sqnum = start_sqnum + (УЗ – LBA первой страницы ОХД)
  - 1.1.2 Если id < start_sqnum или id > end_sqnum, то блок не найден
  - 1.1.3 Иначе return (id – start_sqnum) + LBA первой страницы ОХД
- 1.2 Указатель заполнения = Конец ОХД
  - 1.2.1 Delta = current_sqnum – id
  - 1.2.2 Если Delta <0 или Delta > T, то блок не найден
  - 1.2.3 B = current_sqnum – start_sqnum
  - 1.2.4 Если B < 0, B += M

1.2.5 Если B < Delta, Delta -= T
1.2.6 return LBA первой страницы ОХД + B - Delta
2 *T* >= *M*
2.1 Delta = current_sqnum - id
2.2 УЗ < start_sqnum + M
2.2.1 Если Delta > current_sqnum или Delta < 0, блок не найден
2.2.2 return current_page -delta
2.3 УЗ >= start_sqnum + M
2.3.1 Если Delta < 0, Delta += M
2.3.2 return current_page - delta

ИНДЕКС ДАННЫХ

Индекс предоставляет альтернативную модель доступа к данным – в отличие от модели хранилища данных, которая предоставляет блочный доступ к данным по числовому идентификатору, индекс предоставляет пофайловый доступ к данным. Индекс хранит соответствия блоков их временам, что дает также возможность получения данных в заданном временном окне.

ЭНКОДЕР ДАННЫХ

Для уменьшения объема, занимаемого данным, они сжимаются и пакуются в *miniseed*-блоки, которые потом непосредственно сохраняются в хранилище. Каждый блок содержит все свои метаданные – имя канала, имя станции, частота дискретизации, время и др.

СЕРВЕРЫ TCP/IP

Seedlink-сервер обслуживает запросы данных по протоколу *seedlink*.

FTP-сервер предоставляет доступ к индексу данных и возможность пофайлового запроса проиндексированных данных по протоколу FTP.

Telnet-сервер предоставляет интерфейс для удалённого управления и конфигурации устройства.

Bmxp-сервер служит для взаимодействия устройства и сервера сбора данных.

АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ВЕРХНЕГО УРОВНЯ

В итоге записи сигналов запрашивает управляющее программное обеспечение на компьютере, состоящее из серверной и клиентской части. Сервер сбора данных выполняет следующие функции:

1. Получение данных с измерительных узлов
2. Перекодирование потока данных в соответствии с настройками
3. Сохранение потока данных на диск
4. Предоставление потока клиентской программе для визуализации
5. Обнаружение событий (критерий *LTA/STA*)

При конфигурации сервера ему предоставляется список адресов, по которым он подключается и находит устройства.

Клиент служит для визуализации данных в режиме реального времени.

Общение между клиентом и сервером производится по протоколу, основанном на технологиях *ZeroMQ* и *ProtocolBuffers.*

ЛИТЕРАТУРА

[1] Система CiesComp 3. URL: http://www.seiscomp3.org/
[2] Работа со стеком TCP/IP в сетях Windows NT. URL: http://citforum.ru/operating_systems/winntadm/winntadm_09.shtml
[3] С.В. Баранов. Применение вейвлет-преобразования для автоматического детектирования сейсмических сигналов. URL: http://www.maikonline.com/maik/showArticle.do?auid=VAF2X6YGT3&lang=ru
[4] Serial Digital Interface. URL: http://ru.wikipedia.org/wiki/Serial_Digital_Interface
[5] Всемирное координированное время. URL: http://ru.wikipedia.org/wiki/%D0%92%D1%81%D0%B5%D0%BC%D0%B8%D1%80%D0%BD%D0%BE%D0%B5_%D0%BA%D0%BE%D0%BE%D1%80%D0%B4%D0%B8%D0%BD%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%B2%D1%80%D0%B5%D0%BC%D1%8F
[6] Сайт точного времени URL: http://tochnoye-vremya.com/%D0%B2%20UTC
[7] Протокол TCP/IP. Учебник Википедии. URL: http://ru.wikipedia.org/wiki/TCP/IP
[8] lwIP. Учебник Википедии. URL: http://ru.wikipedia.org/wiki/LwIP
[9] BSD. Учебник Википедии. URL: http://ru.wikipedia.org/wiki/BSD
[10] Э. Таненбаум, А. Вудхалл. Операционные системы. Разработка и реализация. 3-е издание. Изд-во Питер. С-Петербург. 2010.

Денис Терешкин, аспирант кафедры Автоматики НГТУ, автор 20 научных статей. Область научных интересов и компетенций – теория автоматического управления, лазерная физика, программные системы.
E-mail: todin.dirihle@gmail.com