**Драйвер платы Datum BC635PCI для QNX 6**

**1) Описание**

Плата BC635PCI производства компании Datum (она же Bancomm, Symmetricom) предназначена для точного отсчета времени и выдачи временного сигнала в формате IRIG. Платой поддерживается три основных режима работы:

* Режим выдачи сигнала IRIG-B
* Свободный режим работы
* Режим синхронизации с пульсирующим сигналом (1PPS)

В режиме выдачи сигнала IRIG-B плата принимает сигнал в специальном формате IRIG-B, имеющем определенные характеристики. В свободном режиме работы встроенный в плату генератор отсчитывает время самостоятельно. В режиме синхронизации время платы отсчитывается по внешнему пульсирующему сигналу. В случае, если сигнал отсутствует с самого начала или утерян, плата продолжает свою работу в режиме свободного отсчета времени, и восстанавливает синхронизацию с источником, как только с него появляется сигнал.

В данном документе описано устройство и особенности работы драйвера платы BC635PCI для операционной системы QNX 6.

Примечание: на операционную систему QNX 6.0 должен быть установлен патч, обновляющий ее до последней доступной версии. В противном случае плата не будет обнаружена в списке устройств.

**2) Устройство драйвера**

Драйвер представляет собой набор из двух файлов – datum\_irig\_api.c и datum\_irig\_api.h, запакованных в архив либо datum\_irig\_api.tar (а также дублирующий архив datum\_irig\_api.zip для ОС семейства Windows). Также в архиве находится datum\_irig\_test.c, предназначенный для демонстрации работы и основных функций драйвера.

Для использования драйвера необходимо распаковать архив BC635PCI\_QNX\_DRIVER.tar при помощи утилиты tar (для QNX перейти в папку с архивом командой “cd [путь к архиву]”, затем ввести команду “tar xvfz BC635PCI\_QNX\_DRIVER.tar”). Затем нужно поместить файлы datum\_irig\_api.c и datum\_irig\_api.h в папку с исходными файлами проекта, в котором планируется использование платы Datum BC635PCI. После этого необходимо включить файл datum\_irig\_api.h в проект при помощи директивы препроцессора #define. После этого драйвер готов к использованию.

**3) Основные структуры данных драйвера**

На уровне пользователя для установки/получения времени с платы используются два типа данных, описанных в заголовочном файле “ datum\_irig\_api.h ”. Это связано с тем, что плата работает со временем в двух форматах:

* Бинарный формат, используемый в большинстве UNIX систем
* Десятичный формат, используемый в быту

Бинарное время представляет собой число, обозначающее число миллисекунд, прошедшее с первого января 1970 года по шкале UTC. Основной единицей в этом случае является миллисекунда. При этом значение текущего года не входит в запись числа в бинарном формате, поскольку оно может рассчитано. Для удобства вместе со временем в бинарном формате плата хранит год отдельно. В десятичном формате время представлено как год / месяц / день / час / минута / секунда / миллисекунда / наносекунда.

Для работы со временем в драйвере используются две структуры: binary\_event\_time и decimal\_event\_time. Рассмотрим их подробнее.

**3.1) Структура binary\_event\_type**

union binary\_event\_time

{

struct

{

unsigned u\_seconds : 20;

unsigned n\_seconds : 4;

unsigned status : 4;

unsigned unused : 4;

unsigned seconds : 32;

} bits;

struct

{

unsigned long time0;

unsigned long time1;

} words;

};

Структура представляет собой описание способа хранения данных на самой плате. Плата имеет два регистра под названиями TIME0 и TIME1 для хранения времени. В регистре под названием TIME1 находятся значимые единицы времени: количество секунд, прошедшее с 1 января 1970 года (секунды в формате UNIX). В регистре TIME0 находятся менее значимые единицы времени: миллисекунды и наносекунды (плата поддерживает точность до сотых долей наносекунд). Распределение этих данных в регистрах указано на рисунке 1.

Каждый регистр представляет из себя слово из 32 бит (тип uint32\_t). Структура binary\_event\_time, использующаяся в API, соответствует структуре регистров, использующихся на плате. Слова time0 и time1 соответствуют регистрам TIME0 и TIME1, а битовые поля u\_seconds, n\_seconds и seconds позволяют получить непосредственный доступ к значениям миллисекунд, сотен наносекунд и секунд, ничего не зная о внутреннем устройстве регистров устройства.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Регистр | Биты данных | | | | | |
| 31-28 | 27-24 | 23-20 | 19-16 | 15-8 | 7-0 |
| TIME0 | Основные единицы времени – UNIX секунды | | | | | |
| TIME1 | Не используется | Статус | Сотни наносекунд | Бинарные миллисекунды | | |

Таким образом, после заполнения структуры (описано далее) при помощи функции API можно получать единицы времени при помощи простых обращений object.bits.seconds, object.bits.u\_seconds и object.bits.n\_seconds, где object – это некий объект типа binary\_event\_type. Для получения года необходимо обращение к оперативной памяти, которое оформлено в отдельную функцию API. Подробнее она описана в разделе **4**

**3.2) Структура decimal\_event\_type**

union decimal\_event\_time

{

struct

{

unsigned u\_seconds : 20;

unsigned n\_seconds : 4;

unsigned status : 4;

unsigned day8 : 1;

unsigned unused : 3;

unsigned seconds : 8;

unsigned minutes : 8;

unsigned hours : 8;

unsigned days : 8;

} bits;

struct

{

unsigned long time0;

unsigned long time1;

} words;

};

Структура также представляет собой описания способа хранения данных о времени на плате, но в десятичном формате, понятном для большинства людей. Стоит отметить, что физически данные лежат в одних и тех же регистрах как при бинарном, так и при десятичном представлении чисел – содержимое регистров времени зависит от режима хранения данных, выбранного в плате. API драйвера самостоятельно справляется со всеми преобразованиями способа хранения данных внутри платы.

Распределение данных в регистрах TIME1 и TIME0 при десятичном представлении времени описана в таблице 2. В прямоугольных рамках указаны номера битов, занимаемых каждым объектом, а в скобках – диапазон принимаемых значений.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Регистр | Биты данных | | | | | |
| 31-28 | 27-24 | 23-20 | 19-16 | 15-8 | 7-0 |
| TIME1 | Дни (0-366)   |  | | --- | | 31-24 | | | Часы (0-23)   |  | | --- | | 20-16 | | | Минуты  (0-59)   |  | | --- | | 12-8 | | Секунды  (0-59)   |  | | --- | | 5-0 | |
| TIME0 | День недели  (1-7)   |  | | --- | | 28 | | Статус   |  | | --- | | 27-24 | | Сотни наносекунд   |  | | --- | | 23-20 | | Бинарные миллисекунды   |  | | --- | | 19-0 | | | |

Аналогично данным в бинарном формате, содержимое регистров TIME1 и TIME0 находится в словах words.time1 и words.time0 соответственно. Для доступа к данным без знания устройства регистров нужно использовать битовые поля, описанные в структуре. Также необходимо отметить, что миллисекунды находятся в бинарном формате (UNIX). Так, для получения времени в стандартном формате год / месяц / день / секунда / миллисекунды / наносекунды, необходимо по очереди обратиться к элементам object.bits.year, object.bits.month, object.bits.days, object.bits.minutes, object.bits.seconds, object.bits.u\_seconds и object.bits.n\_seconds, где object – это некоторый объект типа decimal\_event\_type.

**4) Описание пользовательских функций API**

API драйвера имеет 11 пользовательских функций, которые позволяют использовать плату по назначению. Прототипы функций вместе с их кратким описанием представлены в файле datum.\_irig\_api.h. Рассмотрим подробно каждую из них.

int irig\_init ( void ) - инициализирует плату и все параметры, необходимые для корректной работы драйвера. Вызывать эту функцию необходимо перед любым использованием функций из API.

int set\_binary\_irig\_time ( uint32\_t unix\_s) – позволяет установить начальное время, с которого начинается отсчет. Отсчет начинается с момента установки времени. На вход функция принимает число типа uint32\_t, являющееся количеством секунд в бинарном UNIX формате, то есть число, подаваемое на вход, должно означать количество секунд, прошедших с первого января 1970 года. После использования данной функции установки времени все операции в плате производятся со временем в бинарном формате, поэтому запуск любых функций по работе с десятичным представлением времени (имеют суффикс decimal) будет пресечено. Для работы с десятичным представлением времени необходимо повторно установить время в плате, но уже в десятичном формате, при помощи функции int set\_decimal\_irig\_time.

int get\_binary\_irig\_time ( binary\_event\_time \* time\_storage ) – позволяет получить текущее время на плате (на момент вызова функции) в бинарном формате с точностью до микросекунд. На вход функции поступает указатель на объект типа time\_storage, в котором и будет храниться данные. Для доступа к полученным данным необходимо обращаться к битовым полям данного объекта, как это было указано в пункте 3. Например, для получения количества часов необходимо обратиться к object.bits.hours, где object – некоторый объект типа binary\_event\_type. Все получаемые числа представлены в формате unsigned\_int.

int get\_binary\_irig\_time ( binary\_event\_time \* time\_storage ) – позволяет вывести на экран в понятном для человека виде объект типа binary\_event\_type, содержащий данные о предварительно полученном при помощи функции get\_binary\_irig\_time момента времени. Вывод представлен в формате секунды : микросекунды : сотни наносекунд.

int set\_year(uint16\_t year) – позволяет установить год в память микросхемы. Использование данной функции необходимо при использовании времени в бинарном формате, посколько бинарный формат хранения данных не предполагает сохранение информации о годе. Функция принимает на вход переменную типа uint16\_t.

int get\_year(uint16\_t year) – позволяет получить значение года, записанное в оперативной памяти микросхемы. Ее стоит использовать в связке с функцией get\_binary\_irig\_time, поскольку бинарный формат хранения данных о времени не предполагают сохранение информации о годе. Функция возвращает год в формате uint16\_t.

int show\_year ( void ) – выводит на экран текущее значение года, записанное в памяти микросхемы.

int set\_decimal\_irig\_time ( uint16\_t year, uint16\_t days, uint8\_t hours, uint8\_t minutes, uint8\_t seconds ) – устанавливает начальное время платы в десятичном формате. Аналогично функции set\_binary\_irig\_time, эта функция автоматически устанавливает формат хранения данных как десятичный. Для повторного использования функций по работе с бинарным форматом хранения времени, необходимо использовать функцию set\_binary\_irig\_time. Отсчет времени начинается с использования этой функции.

int get\_decimal\_irig\_time ( decimal\_event\_time\* time\_storage ) – функция, которая позволяет получить текущее значение времени на плате в десятичном формате. На вход функция принимает указатель на объект типа decimal\_event\_time, который описан в пункте 3 настоящего документа. Для получения конкретных значений единиц времени необходимо обращаться к битовым полям данного объекта. Например, чтобы получить количество месяцев, необходимо воспользоваться конструкцией object.bits.month, где object – объект типа decimal\_event\_type.

int set\_irig\_mode (int mode) – функция, устанавливающая режим работы платы. На вход подается переменна mode, возможные значения которой определены как строковые константы в файле datum\_irig\_api.h. Возможные режимы работы платы:

* IRIG\_DEPENDABLE\_MODE – режим работы платы в формате временного кода IRIG B. Используется для синхронизации времени на плате с внешним источником и/или выдаче сигнала на внешний источник в формате IRIG B. Конкретный режим работы – синхронизация или выдача синхронизирующего сигнала – зависит от наличия подключенных плат во входной и выходной разъемы.
* IRIG\_AUTONOMOUS\_MODE – режим свободного отсчета времени, в котором плата самостоятельно ведет отсчет времени с момента последней установки времени. Данный режим является режимом работы платы по умолчанию, и в случае потери синхронизирующих сигналов плата автоматически переключается на него.
* IRIG\_PPS\_MODE – режим синхронизации с внешним пульсирующим сигналом. Как правило, такой сигнал имеет периодичность 1 Гц. Условием использование этого режима является наличие сигнала в соответствующем разъеме платы. В противном случае плата автоматически переключится в автономны режим свободного отсчета времени.

int show\_board\_info ( void ) – информативная функция, выводящая на экран сведения о подключенной плате. На экран выводится модель платы, текущий режим работы, текущий формат хранения данных, год, а также версию прошивки платы. Функция используется в диагностических целях.

int irig\_deinit ( void ) – функция деинициализации драйвера платы. Используется после вызова всех необходимых функций API для освобождения памяти и отключения драйвера от платы. Использование функции деинициализации обязательно при дальнейшей работе вызывающей программы с сервером pci.

**5) Примеры использования API.**

Пример работы API находится в файле datum\_irig\_test.c. Программа, находящаяся в этом файле, инициализирует драйвер, устанавливает время в бинарном формате, считывает его три раза, затем устанавливает время в десятичном формате и снова считывает его три раза. После этого выводится диагностическая информация о плате, после чего действие программы заканчивается. После каждого этапа проверки на экран выводится сообщение об успешном завершении испытания.

**6) Примечания.**

Вся информация о внутреннем устройстве регистров платы и низкоуровневых операциях находится в файле UserGuide.pdf, находящемся в архиве с драйвером. Во время написания данного драйвера было установлено недокументированное смещение в оперативной памяти. Оперативная память платы, как указано в UserGuide.pdf, разбита на четыре области – область ввода данных, область хранения года, область вывода данных и область хранения данных GPS (опционально). Для доступа к каждой из этих областей необходимо знать смещение относительно предыдущей. Смещение области вывода было указано неверно, настоящая область смещения сдвинута вправо относительно указанной на 256 байт. Это отражено в функции irig\_init() файла datum\_irig\_api.c при вычислении значения переменной DP\_ram\_ofs.output, к которой прибавляется дополнительно 256 байт сверх документированного значения.

**Содержание**

1. Описание …………………………………………………………………………………………………………стр. 1
2. Устройство драйвера……………………………………………………………………………………….стр. 1
3. Основные структуры данных драйвера…………………………………………………………..стр. 2
   1. Структура binary\_event\_type
   2. Структура decimal\_event\_type
4. Описание пользовательских функций API……………………………………………………….стр. 4
5. Примеры использования API……………………………………………………………………………стр. 7
6. Прмечания………………………………………………………………………………………………………...стр. 7