

Петров Игорь Викторович, Пастушенков Дмитрий Владимирович
сотрудники компании Пролог
e-mail: prolog@sci.smolensk.ru
www.prolog.smolensk.ru

Программируем временные сложности

Описаны устройство и проблемы современного календаря и часов. Предложены приемы программирования алгоритмов с календарными датами, часами и таймерами. Программы реализованы на языках ST и FBD стандарта МЭК 61131-3 в системе CoDeSys (3S).

При современном быстром темпе жизни нас уже не могут удовлетворить старые способы проверки времени, как, например городские или башенные часы ... [1]

Жизнь любого современного человека связана с постоянным контролем времени. Это создает впечатление, что при программировании в подобных задачах не может возникнуть ни каких сложностей. Но это далеко не так. Время весьма коварно. Все мы интуитивно понимаем, что такое время, тем не менее, точного определения не может дать никто. Время невозможно абсолютно точно оцифровать, любая дискретизация создает погрешность. Кроме того, любое измерение времени уже является устаревшим, то есть неточным. Эту мысль превосходно выразил Гераклит: «в одну и ту же реку нельзя войти дважды». Процессы, связанные со временем невозможно остановить либо выполнить по шагам. Классическая тактика отладки программ здесь неприменима.

Существующие система счисления времени и календарь крайне неудобны с точки зрения программиста. Тем не менее, почти ни одна прикладная программа систем мониторинга и управления не обходится без операций со временем. Наряду с логическими выражениями и обработкой битовых строк измерение и формирование интервалов времени являются самыми востребованными действиями.

В этой статье мы хотим поделиться своим опытом работы со временем и датами в ПЛК и встраиваемых микропроцессорных системах. Для каждой задачи мы отобрали только одно наиболее понятное решение. С этой же целью, во многих примерах введены избыточные временные переменные, удалены контроль параметров и оптимизация алгоритма вычислений.

Статья ориентирована на применение языков программирования МЭК 61131-3 [2,3]. В примерах используются FBD и ST. Язык ST достаточно «прозрачен» для C и Паскаль программистов. Высокая наглядность языка FBD позволила отказаться от приведения блок схем алгоритмов.

Часы и календарь

Смена дня и ночи – сутки отмерены нам самой природой. Деление же суток на часы минуты и секунды придумано человеком.

В Вавилоне сутки начинались с рассветом и делились на 12 часов, год содержал ровно 360 суток. Древние Египтяне первыми стали разделять сутки на 24 часа и добавили в конце года дополнительные 5 суток. Позднее час был разделен на 60 первичных частей – прима минута, минута на 60 вторичных частей – секунда, секунда на 60 частей третьего уровня – терция. Похожим

образом построена градусная система измерения угла дуги. Это также наследие Вавилона, в котором была распространена шестидесятеричная система счисления. Терции применялись редко. Доли секунд потребовались значительно позднее, когда десятичная система счисления уже стала доминирующей. В итоге, для минут и секунд мы используем шестидесятеричную систему, далее переходим к десятичным долям – санти, милли, микро и т.д. Помимо этих сложностей, время изменяется скачками при переходе в другой часовой пояс. Но и это еще не все, с конца марта до конца октября мы живем по летнему времени, сдвинутому на час вперед. Ниже мы расскажем, что високосными могут быть не только годы, но и секунды.

С календарем ситуация еще более сложная. В году примерно 365,242 суток. В календаре, утвержденном Гаем Юлием Цезарем, введено понятие високосного года. К каждому четвертому Юлианскому году добавлялся один день. Таким образом, год приравнивается к 365,25 суток. Уточнение летоисчисления утвердил в 1582 году Папа Григорий XIII. Было принято не считать високосным годы, которые делятся на сто, но не делятся на 400. Этим указаниям последовали далеко не все страны, в том числе и православная Россия.

Из отечественных исторических личностей к календарю причастны двое. Царь Петр утвердил новое летоисчисление от Рождества Христова (Anno Domini –AD). Таким образом, россияне, привычно отметившие 1 сентября новый 7208 год «от сотворения мира» вынуждены были встречать 1 января год 1700 от Рождества Христова. Вторым реформатором оказался Ленин, подписавший 24 января (6 февраля) 1918 года декрет "О введении в Российской республике западноевропейского календаря". 1 февраля 1918г. предписывалось считать четырнадцатым.

Современное правительство также неравнодушно к календарю, но ограничивается пока только регулярным переносом праздничных дней совпадающих с выходными. Русская православная церковь до сих пор не признала новый стиль, зато благодарные россияне отмечают Новый Год дважды.

Источники точного времени

Проплывая на своей яхте по Темзе, обратите внимание на башню Фламстид Хаус, расположенного на вершине Гринвичского холма. Ежедневно ровно в 13:00 по его мачте падает красный шар. В водах Невы полуденный выстрел из пушки, согласно указу Петра, призывает вас принять чарку и приготовиться к обеду. К сожалению, эти традиционные способы синхронизации часов для вычислительных машин не подходят. Прежде чем говорить о работе с часами, нужно определить доступные компьютерам источники точного земного времени.

Страстная любовь человечества к сокращениям, особенно из 3х букв не минула время. Вот некоторые наиболее распространенные из них:

GMT (Greenwich Mean Time) усредненное солнечное время на нулевом (Гринвичском) меридиане.

При движении на запад местное время отстает от GMT с каждым градусом долготы на 4 минуты. Это легко понять. Один полный оборот Земли (360°) совершается за сутки (1440 минут / 360 = 4). На каждые 15 долготы отставание составит один час.

Споры о расположении нулевого меридиана продолжались не одно столетие. Только 1 ноября 1884 года собравшаяся в Вашингтоне Международная меридианная комиссия утвердила Гринвичский меридиан за "нулевой" и Гринвичское время в качестве всемирного времени. Россия приняла систему поясного времени лишь 8 февраля 1918г.

Почему именно Гринвич? На самом деле, важнее место прохождения не нулевого, а противоположного ему – 180го меридиана. Это линия перемены дат. Она изгибается в Беринговом проливе и пересекает Тихий океан, ни разу не выходя на сушу.

Несмотря на очевидную разумность принятой системы, только англичане не имеют сомнений относительно места расположения начала времени и колыбели мировой цивилизации. Большинство Европейских стран живут на час впереди по СЕТ (Central European Time) времени. С 1975 года на смену GMT пришло UTC время. Аббревиатура GMT применяется сейчас только для обозначения общегражданского времени в Великобритании.

TAI (International Atomic Time) международное атомное время.

С 1967 года качестве эталона времени используется переход между двумя сверхтонкими уровнями основного состояния атома Цезий 133. Одна секунда это 9192631770 периодов излучения. Секунда является одной из 7 фундаментальных физических единиц измерения системы СИ. Интересно, что секунда является базой для определения не только длительности и частоты, но даже длины. Метр это расстояние, пройденное светом в вакууме за 1/299792458 секунды.

Построение атомных часов является сложной задачей [4]. Даже если мы изготовим сам генератор, то получим только эталонную секунду. Для получения времени нам нужно будет установить начало отсчета и регулярно координировать работу наших часов с другими атомными часами на планете. Результатом международной координации национальных эталонов частоты и времени является TAI.

UTC (Universal Time Coordinate) Всемирное Координированное Время, отличается от TAI на целое число секунд. В абсолютном большинстве случаев под источником точного единого времени на Земле понимается именно UTC. В повседневной практике мы используем поясное время, отличающееся от UTC на целое число часов.

Зачем при наличии точного TAI времени, вводить еще какое то UTC? Проблема состоит в том, что вращение земли происходит неравномерно и постепенно замедляется. В результате, атомное время уходит вперед по отношению к астрономическому. Астрономические сутки составляют сейчас примерно 24 часа и 2 мс. Для некоторых областей деятельности человека необходима привязка именно к астрономическому времени. Многим же более важна высокая точность и равномерность атомного времени. UTC является компромиссом. Международное бюро контроля вращения земли (IERS) периодически вносит в UTC корректирующие поправки – «високосные секунды», так чтобы абсолютное расхождение с усредненным астрономическим временем не превышало по модулю 0,9 секунды. К настоящему моменту разница UTC и TAI достигла 32 секунды. Актуальные значения времени можно увидеть на «главных виртуальных часах планеты» в международном бюро мер и весов (www.bipm.fr см. Рис. 1).



Рис. 1 Виртуальные «главные» часы

Технически, положительные поправки вносятся в часы двумя способами. Добавлением лишней секунды в сутки: 23:59:60. Это не всегда возможно, поскольку нарушает стандартную границу диапазона секунд. Второй вариант это остановка часов в 23:59:59 на одну секунду.

В приведенных в статье преобразованиях мы не учитываем високосные секунды. При необходимости вы можете получить в бюро мер и весов актуальную таблицу дат коррекции UTC. Поскольку коррекция проводится строго по полугодиям (31 декабря или 30 июня), разделив дату на полугодия, вы получите достаточно компактную корректирующую таблицу.

В земных условиях сигналы UTC можно абсолютно свободно и бесплатно получить радиотехническими устройствами. Регулярная передача сигналов точного времени в Советском Союзе начата Пулковской обсерваторией с 1920г. из Москвы и Ленинграда [1]. Кодированные сигналы точного времени передаются на частотах близких к «круглым»: 50 кГц, 5, 10, 15 МГц. Специальные радиопередатчики оснащены эталонными генераторами частоты. Звуковые сигналы точного времени (6 точек) передают и обычные радиовещательные станции. Наверняка читатели помнят популярные настенные часы "Электроника 7-06 К" с авто коррекцией по сигналам радиостанции Маяк.

В настоящее время многие фирмы выпускают высокоточные часы, оснащенные стандартными интерфейсами и автоматически обеспечивающие синхронизацию с UTC по кодированным радиосигналам. В печати можно встретить описания подобных часов доступных для сборки квалифицированными радиолюбителями [5]. К сожалению, частоты и системы кодирования разных станций различны. В результате создать универсальный приемник, устойчиво работающий в любой точке Земли невозможно. Это существенно сдерживает рынок подобных устройств.

Альтернативу приемникам наземных станций составляют приемники спутниковых навигационных систем: ГЛОНАСС, GPS и в перспективе развертывающийся Европейской системы Галилей. Навигационные спутники

могут служить источником точного времени в любое время суток и года в любой точке земной поверхности. На территории России приоритетной является система ГЛОНАСС, спутники GPS могут использоваться как резервные. На рынке систем навигации GPS приемники представлены наиболее широко. Атомные часы спутников GPS не корректировалось високосными секундами после запуска. К настоящему моменту они опережают UTC на 13 секунд.

Во многих странах существуют специальные автоматические телефонные службы точного времени. Помимо общеизвестного голосового сервиса, разработаны протоколы точной (до единиц миллисекунд) синхронизации часов для различных электронных устройств. Например, ACTS (Automated Computer Time Service). К сожалению, подобные системы в России не нашли широкого применения.

Получить UTC время можно и через сеть Интернет. Протокол NTP и его упрощенная версия SNTP (Simple Network Time Protocol) позволяет обеспечить синхронизацию машин, как в локальной сети, так и в Интернет (www.ntp.org). Первичные NTP серверы связаны с TAI часами, на их основе созданы общедоступные серверы второго уровня. Современные сетевые ОС имеют встроенную поддержку NTP. Для Win9x необходимо использовать дополнительные TimeSync утилиты. Существует немало количество программ красочных часов для рабочего стола с автоматической UTC коррекцией. Недостатками использования Интернет является отсутствие гарантированной надежности «чужого» сервиса и открытость такого соединения извне.

Работа Интернет не связана с географическим положением пользователей. Это виртуальное пространство и для него логично иметь собственную шкалу времени, не связанную земными узлами. В связи с этим в 1998 году швейцарская часовая фирма Swatch (www.swatch.ch) учредила собственную шкалу времени для киберпространства **ВМТ** (Biel Mean Time). Сутки разделены на 1000 долей (beats). Часовые пояса отсутствуют, ВМТ время привязано к UTC. Точнее, к среднему Европейскому времени города Биль, где расположен центральный офис фирмы. Полдень по СЕТ времени выражается так: @500 Swatch .beats. Англичане не согласились с таким подходом и с начала века запустили собственные **GeT** часы (Greenwich e-time). Таким образом, Великобритания вновь объявлена центром цивилизации, на этот раз электронной.

Дата, время и длительность

Давайте для начала разберемся с различиями типов переменных, которые применяются для работы со временем и датой. Для этого напомним соответствующие типы переменных, операции и константы определенные стандартом МЭК 61131-3 и составляющие целостную систему.

Таких типов в стандарте 4: интервал времени TIME, время дня TIME_OF_DAY, дата DATE и временная отметка DATE_AND_TIME.

Для каждого типа существуют специальные префиксы, позволяющие определить, что значение переменной или константы относится к данному типу (См. Табл.1).

Полная форма	Сокращенная форма
TIME#	T#.
TIME_OF_DAY#	TOD#

DATE#	D#
DATE_AND_TIME#	DT#

Табл. 1. Типы данных

Для типа TIME представление значений времени состоит из полей дней (d), часов (h), минут (m), секунд (s) и миллисекунд (ms). Порядок представления должен быть именно такой, хотя ненужные элементы можно опускать. Для лучшего зрительного восприятия, поля допускается разделять символом подчеркивания. Например:

t1 := t#10h_14m_5s;

Старший элемент имеет право превышать верхнюю границу естественного диапазона представления. Так если в представлении присутствуют дни или часы, то секунды не могут превышать значения 59. Если секунды стоят первыми, то их значение может быть и большим.

t1 := t#1m65s; (**ошибка**)

t1 := T#125s; (**правильно**)

Младший элемент можно представить в виде десятичной дроби:

t1 := T#1.2S; (**T#1s200ms**)

Тип TIME это время протекания некоторого процесса – длительность. Его нельзя смешивать со временем суток – тип TIME_OF_DAY. Принципиальное их отличие состоит в том, что значение переменных типа TIME_OF_DAY ограничено диапазоном от 0 до 24 часов. Длительности, определяемые переменными типа TIME, могут быть произвольными.

Представление значений времени суток дается в формате часы: минуты: секунды. Секунды могут иметь дробную часть:

td1 := TOD#23:59:59.99;

Применение типа DATE очевидно. Представление даты дается в формате: ВВГГ-ММ-ДД. Все числа должны быть целыми в рамках естественных диапазонов. Нумерация дней и месяцев идет «по человечески», т.е. с единицы:

d1 := d#2004-12-04;

Переменные типа DATE_AND_TIME представляют собой временной штамп – дату и время «в одном флаконе». Представление значений типа DT содержит дату и время, через дефис:

dt1 := dt#2004-04-04-16:30:00;

Обратите внимание, что дата представлена как ГГГГ-ММ-ДД-ЧЧ:ММ:СС. Мы в России более привыкли к другому порядку записи но, такое представление определено ISO 8601 и считается однозначно понятным в разных странах.

Смешивать в выражениях переменные разного типа нельзя. При необходимости применяются операции преобразования типов. Наименование стандартных преобразований выглядят так: «ИЗ ТИПА_ТО_В ТИП». Например:

```
d1 := DATE_AND_TIME_TO_DATE(dt1);
```

Внутреннее представление

К сожалению, внутренний формат представления даты и времени не стандартизован. Даже в рамках одной системы МЭК 61131-3 программирования на разных целевых платформах он может отличаться. Многие разработчики вообще ограничиваются только поддержкой типа TIME.

С точки зрения «человеческого» представления, удобнее всего хранить дату и время в виде структуры. Например, в DATA_AND_TIME можно представить структурой из 8 переменных, содержащей поля года, месяца, дня недели и т.д. Такой формат очень удобен для организации ввода-вывода в прикладной программе, но совершенно непригоден для вычислений.

Удобнее преобразовать дату и время в одно число. Это может быть число секунд или дней отмеренных от определенного момента. В результате вычисление длительности между двумя отметками времени сводится к простому вычитанию. Подобное указание числа дней от PX часто используется в научных источниках. Это позволяет избежать необходимости учета различных календарей. К сожалению, числа получаются достаточно большими и неудобными для повседневного обращения.

В качестве момента отсчета в компиляторах чаще всего выбирается 1 января 1970 года. В POSIX C определен тип time_t. Это не что иное, как число секунд с 0 часов 1 января 1970г. Дискретность в 1 секунду считается достаточной для большинства практических задач.

Рекордной точностью обладает внутреннее представление 64х разрядных отметок времени широко распространенного сетевого протокола NTP. Старшие 32 разряда содержат число секунд с 1 января 1900 года, младшие 32 разряда это доли секунд.

В системе МЭК программирования CoDeSys (3S) [3] под все вышеперечисленные типы отводятся 32х разрядные целые переменные (DWORD). Значения типа DATE и DT отсчитывают время в секундах, начиная с 0 часов 1 января 1970 года. Это позволяет работать с датами до D#2106-02-06.

Длительность TIME и TIME_OF_DAY имеют дискретность в 1мс. Максимальная длительность, отмеряемая переменными типа TIME, примерно составляет 1193 часа или почти 50 суток.

Во всех последующих примерах, где выполняются преобразования форматов, мы будем подразумевать именно такую реализацию. Все примеры выполнены авторами в системе CoDeSys. Если вы используете другой инструмент, изучите внутренне представление даты и времени по фирменной документации. В системах, не имеющих поддержки типов D, DT и TOD, вы можете использовать тип 32х разрядное целое (UDINT).

В качестве иллюстрации техники преобразования типов в CoDeSys, реализуем стандартную функцию CONCAT_DATE_TOD. Она «собирает» дату и время во временной штамп. Функция построена на преобразованиях даты и времени дня в DWORD. (Реально эти преобразования ни чего не делают, поскольку DWORD и есть внутренний формат переменных). Преобразования нужны только для проведения вычислений. Блок деления DIV понижает точность времени дня с миллисекунд до секунд. Остается выполнить суммирование и преобразовать результат в дату-время (См. Рис. 2).

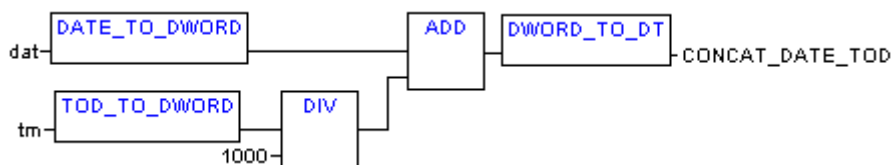


Рис. 2 Функция CONCAT_DATE_TOD (язык FBD)

Операции со временем и датами

С самого начала стандарт МЭК 61131-3 принял достаточно логичную идею перегружаемых математических операций. Используя один и тот же «+» вы можете складывать целые числа или числа с плавающей запятой, не задумываясь о том, что для процессора это очень разная работа. Однако как только речь заходит о вычислениях с датами и временем, ситуация резко обостряется. Математические операции со временем это один самых спорных вопросов реализации любых систем программирования. Подумайте, например, что должно получиться при сложении 1 января 2004 года и 1 апреля 2005? С точки зрения «красоты» системы программирования, желательно ограничиться только абстрактными типами переменных. Иначе мы приходим к тому, что начнем вводить отдельные типы данных и операции для скорости, температуры и всего прочего (что и делается в ООП). С другой стороны, работа с датами и временем настолько распространена что, отдав ее на прикладной уровень, мы создадим серьезные дополнительные сложности при переносе программ в другую среду.

В итоге подобных рассуждений, в тексте стандарта [2] (Table 30 - Functions of time data types), почти все вычислительные операции для даты и времени отмечены как спорные (deprecated). Достаточно часто можно встретить попытки ввести типизированные операции (SUB_TOD_TOD, ADD_DT_TIME и т.д.). Это упрощает построение транслятора, но неудобно в использовании. Естественно, что в такой ситуации нельзя ожидать реализации однозначного набора операций в различных средах программирования. Давайте попробуем аккуратно проанализировать возможные сложности и попытаемся их преодолеть.

Проще всего работать с переменными типа TIME. С ними можно выполнять присваивание, сравнение, сложение, вычитание, умножение и деление на число, использовать их в стандартных ограничителях и мультиплексорах:

```
t2:= T#0d;
t1 := t2 - T#2m;      (* 0мин - 2мин = -2мин *)
t1 := t1 + T#5m;     (* -2мин + 5мин = 3мин *)
t1 := t1/2;          (* 3мин/2 = 1мин30сек *)
t3 := LIMIT(T#2s, t1, T#30s); (* t3 = 30сек *)
```

Отрицательные длительности могут безобидно возникать в промежуточных вычислениях. Является ли это приемлемым для конечного результата, зависит от вашей задачи. Не забывайте также о максимальном значении переменных типа TIME в вашей системе и возможном переполнении.

Соотношение двух длительностей это безразмерная величина. Деление применяется при расчете скважности. Однако непосредственно делить TIME на TIME нельзя. Для подобных вычислений придется явно преобразовать TIME в безразмерное число. Операция деления незамкнута на множестве целых чисел. Поэтому не редко приходится переходить к применению действительных чисел:

Porosity := **TIME_TO_REAL**(t1+t2) / **TIME_TO_REAL**(t2);

Сложнее всего работать с **TIME_OF_DAY**. Фактически эти переменные предназначены только для ежедневного управления по расписанию и фиксации времени событий. Для этих целей достаточно операций выбора и сравнения. Умножение, деление для времени суток не имеет смысла.

Сутки для переменных **TIME_OF_DAY** это календарные сутки от 0 часов до 23:59:59. Разность двух **TIME_OF_DAY** образует **TIME** и является вполне корректной операцией. Но только в том случае, если оба значения находятся в пределах одних и тех же суток. Часто необходимо вычислить время между двумя близкими значениями времени, но не обязательно в рамках одних календарных суток. Например, отрезок времени от 23:50 до 00:10. Для человека это тривиальная задача. На этот случай не сложно реализовать собственную функцию вычисления разности времени (См. Рис. 3):

```

0001 FUNCTION DiffTOD : TIME
0002 (*Возвращает разницу двух отметок времени суток*)
0003 VAR_INPUT
0004     tm1,tm2 : TOD;
0005 END_VAR
0006
0001 IF tm2 < tm1 THEN
0002     DiffTOD := #24h - TOD_TO_TIME(tm1) + TOD_TO_TIME(tm2);
0003 ELSE
0004     DiffTOD := tm2-tm1;
0005 END_IF
0006
  
```

Рис. 3 Функция DiffTOD

Конечно, более надежным решением здесь является переход к переменным типа **DATE_AND_TIME**, если точности данных этого типа достаточно. Например:

DiffTime := dt#2004-01-01-0:0 - TimeStamp; (**TIME:= DT - DT**)

Переменные и константы типов **TIME**, **TOD**, **DATE**, **DT** можно сравнивать и применять в стандартных мультиплексорах (См. Рис. 4).

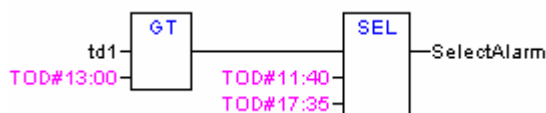


Рис. 4 Сравнение (GT) и бинарный мультиплексор (SEL) времени дня

На языке ST это выглядит так:

SelectAlarm := **SEL**(td1 > TOD#13:00, TOD#11:40, TOD#17:35);

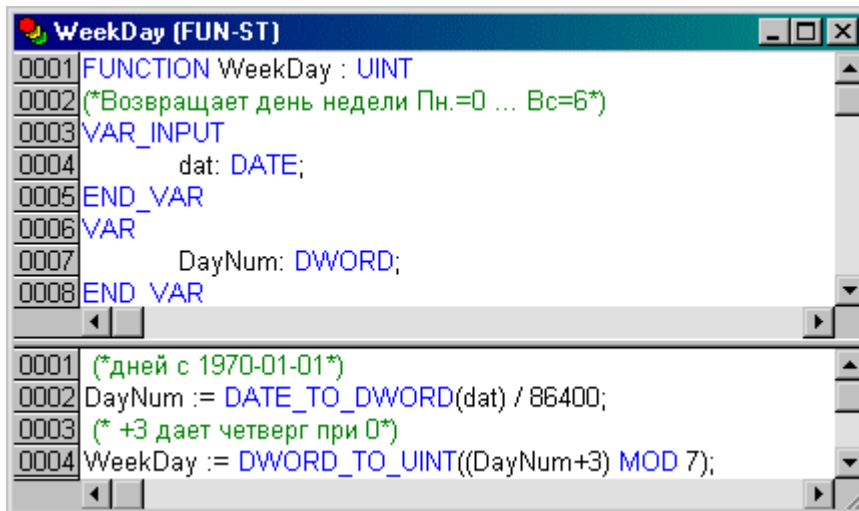
Определение дня недели

Поскольку дни недели следуют неразрывно, вычислить день несложно. Достаточно вспомнить какой был день недели в некоторый день X и число дней отделяющих нас от него (DayNum). Далее находим остаток от деления DayNum на 7 и получаем смещение относительно дня недели даты X.

Для простоты вычислений вспомним следующие факты:

- внутренний формат даты это число секунд с 1 января 1970;
- 1 января 1970 был четверг;
- в сутках 86400 секунд.

Дни недели будем нумеровать с 0 и по отечественной традиции начнем с понедельника. Понедельник – 0, вторник – 1, среда – 2 и т.д. (См. Рис. 5).



```
WeekDay (FUN-ST)
0001 FUNCTION WeekDay : UINT
0002 (*Возвращает день недели Пн.=0 ... Вс=6*)
0003 VAR_INPUT
0004     dat: DATE;
0005 END_VAR
0006 VAR
0007     DayNum: DWORD;
0008 END_VAR

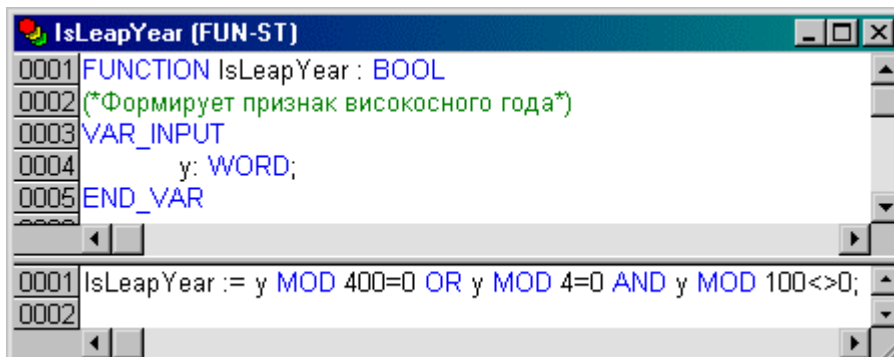
0001 (*дней с 1970-01-01*)
0002 DayNum := DATE_TO_DWORD(dat) / 86400;
0003 (* +3 дает четверг при 0*)
0004 WeekDay := DWORD_TO_UINT((DayNum+3) MOD 7);
```

Рис. 5 Функция WeekDay

Промежуточную переменную DayNum мы ввели исключительно для наглядности.

Определение високосного года

Как уже было сказано, високосным считается любой год, который делится на 400 и годы, которые делятся на 4, но не делятся на 100. Параметром данной функции служит не DATE а именно год, в виде целого UINT числа. Такая реализация проще и будет полезна нам в последующих примерах (См. Рис. 6).



```
IsLeapYear (FUN-ST)
0001 FUNCTION IsLeapYear : BOOL
0002 (*Формирует признак високосного года*)
0003 VAR_INPUT
0004     y: WORD;
0005 END_VAR

0001 IsLeapYear := y MOD 400=0 OR y MOD 4=0 AND y MOD 100<>0;
0002
```

Рис. 6 Функция IsLeapYear (ST)

В виде FBD диаграммы функция IsLeapYear показана на Рис. 7.

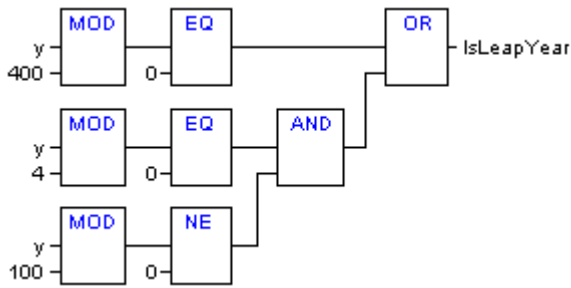


Рис. 7 Функция IsLeapYear (FBD)

С датами до 2100 года можно ограничиться проверкой деления на 4. Для двоичной системы счисления это означает отсутствие единиц в двух младших разрядах.

y: WORD;

IsLeapYear := (y AND 16#0003)=0;

Очень быстрый и заманчивый метод. Но чтобы через сто лет вам не пришлось лишний раз переворачиваться в тесном подгнившем помещении, не изменяйте исходную функцию IsLeapYear.

Определение дней перехода на летнее, зимнее время

В соответствии с [6] переход на «летнее» время в России происходит в последнее воскресенье марта переводом стрелок часов с 2:00 на 3:00. Возврат к «зимнему» времени происходит в последнее воскресенье октября переводом стрелок часов с 3:00 на 2:00.

Зная число (d), месяц (m) и день недели (wd), проверить необходимость перевода часов совсем просто. Например, последнее воскресенье (wd=6) марта:

Daylight := m=3 AND d>24 AND wd=6;

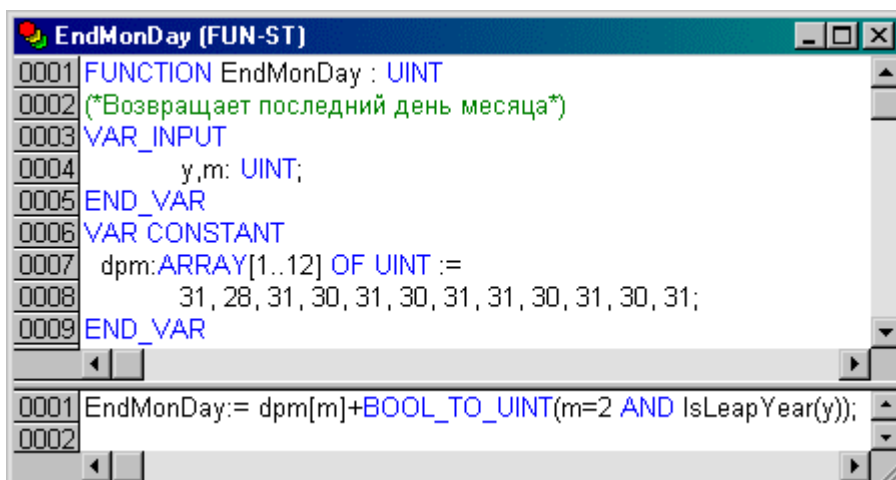
Не стоит слишком полагаться на возможность автоматического перевода часов. Порядок исчисления времени меняется гораздо чаще, чем календарь.

Разумным решением является использование в контроллерах UTC времени. За счет своей непрерывности UTC обеспечивает однозначность трассировки событий. Поправку на часовой пояс и летнее время проще учесть в системах отображения верхнего звена, работающих под контролем операторов.

Определение числа дней в месяце

Параметрами данной функции служат год y и месяц m. Функция EndMonday возвращает количество дней в указанном месяце и может пригодиться при контроле корректности ввода даты.

Таблица dpm содержит число дней по месяцам. Для февраля (m=2) високосного года (IsLeapYear) вводится поправка – 1 день. Напомним что преобразование BOOL в UINT дает 1 для TRUE и 0 для FALSE (См. Рис. 8).



```
0001 FUNCTION EndMonDay : UINT
0002 (*Возвращает последний день месяца*)
0003 VAR_INPUT
0004     y,m: UINT;
0005 END_VAR
0006 VAR_CONSTANT
0007     dpm:ARRAY[1..12] OF UINT :=
0008         31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31;
0009 END_VAR
0001 EndMonDay:= dpm[m]+BOOL_TO_UINT(m=2 AND IsLeapYear(y));
0002
```

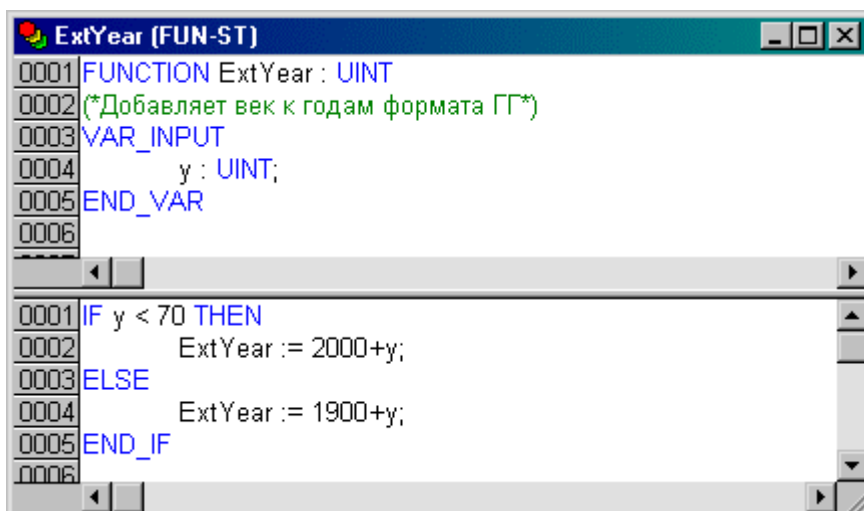
Рис. 8 Функция EndMonDay

Контроль корректности месяца здесь отсутствует. Случайно заданный месяц с номером меньше 1 и больше 12 даст непредсказуемый результат. Ввести ограничение диапазона проще всего стандартной функцией LIMIT.

m := LIMIT(1,m,12);

Дополнение 2х-значного года до 4х знаков

Очень часто люди предпочитают записывать год двумя цифрами. Как правило, подразумевается, что дата относится к текущему веку. На рубеже веков судят по самому числу. Если оно большое, то это конец прошлого века. Если оно маленькое, то это начало нового столетия. Работу по интерпретации дат в диапазоне от 1970 до 2069 берет на себя функция ExtYear (См. Рис. 9).



```
0001 FUNCTION ExtYear : UINT
0002 (*Добавляет век к годам формата ГГ*)
0003 VAR_INPUT
0004     y : UINT;
0005 END_VAR
0006
0001 IF y < 70 THEN
0002     ExtYear := 2000+y;
0003 ELSE
0004     ExtYear := 1900+y;
0005 END_IF
0006
```

Рис. 9 Функция ExtYear

Как и большинство примеров данной статьи, функция ExtYear упрощена и не имеет встроенного контроля корректности значений параметров.

Строковые преобразования даты и времени

Для перевода даты или времени в строку и обратно в МЭК 61131-3 предусмотрены специальные преобразования. Например:

```
str1 := DT_TO_STRING(TimeStamp);
(*str1 := 'DT#2004-04-01-23:50:10'*)
```

```
str2 := TIME_TO_STRING(t#2m55s);
(*str2 := 'T#2m55s'*)
```

Стандартных функций, позволяющих разбирать время и дату на отдельные элементы и наоборот, нет. Ниже мы приводим простые примеры, выполняющие эту работу.

Упаковка часов, минут, секунд и миллисекунд в TIME

Алгоритм работы функции PackTime примитивно прост. Умножаем значение часов на число минут в часе, прибавляем число минут, умножаем на число секунд в минуте и т.д. (См. Рис. 10) Если вам нужно получить значение типа TOD, то необходимо ввести проверку результата на максимум (24 часа) и заменить DWORD_TO_TIME на DWORD_TO_TOD.

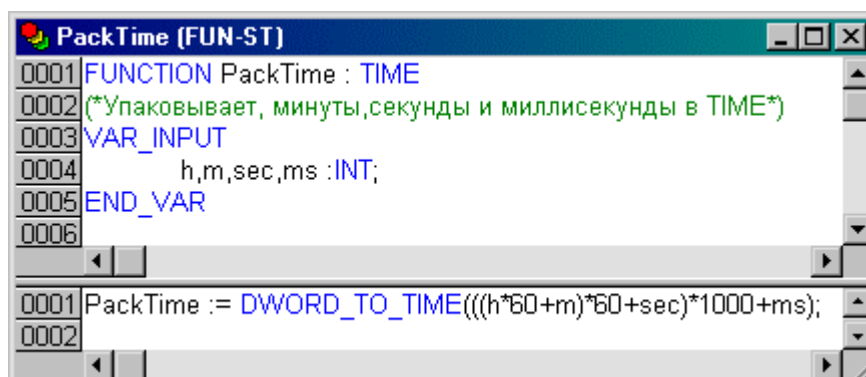


Рис. 10 Функция PackTime (ST)

На языке FBD PackTime выглядит, как показано на Рис. 11.

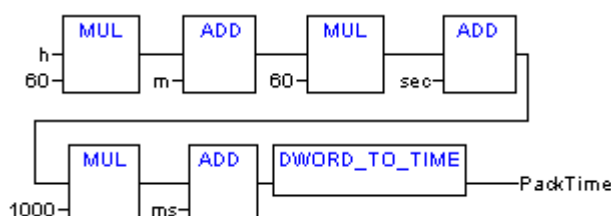
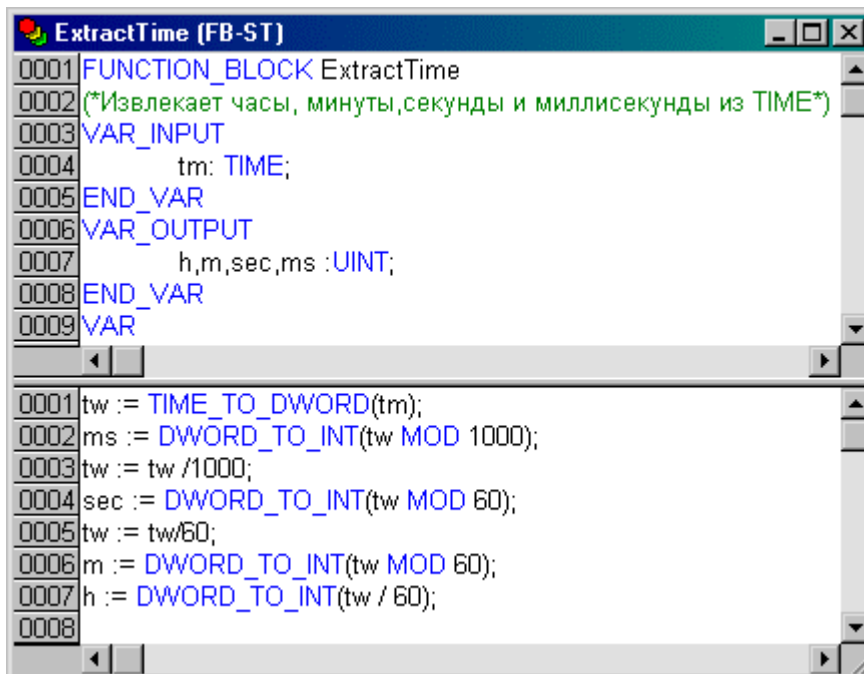


Рис. 11 Функция PackTime (FBD)

Извлечение часов, минут, секунд и миллисекунд из TIME

Для выделения миллисекунд функциональный блок ExtractTime вычисляет остаток от деления времени на 1000. Далее время делится на 1000, что «отбрасывает» миллисекунды. Аналогично последовательно выделяются секунды, минуты и часы (См. Рис. 12).



```

0001 FUNCTION_BLOCK ExtractTime
0002 (*Извлекает часы, минуты, секунды и миллисекунды из TIME*)
0003 VAR_INPUT
0004     tm: TIME;
0005 END_VAR
0006 VAR_OUTPUT
0007     h,m,sec,ms :UINT;
0008 END_VAR
0009 VAR
0010 tw := TIME_TO_DWORD(tm);
0011 ms := DWORD_TO_INT(tw MOD 1000);
0012 tw := tw /1000;
0013 sec := DWORD_TO_INT(tw MOD 60);
0014 tw := tw/60;
0015 m := DWORD_TO_INT(tw MOD 60);
0016 h := DWORD_TO_INT(tw / 60);
0017 END_VAR

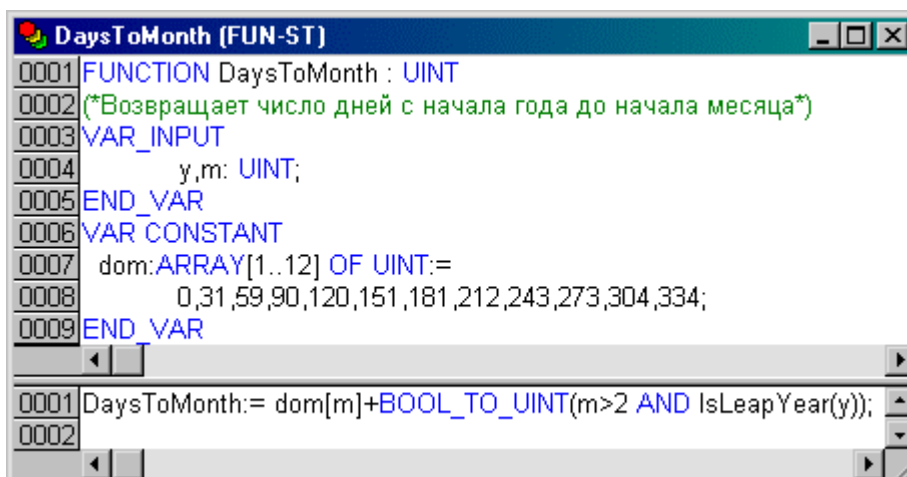
```

Рис. 12 Функциональный блок ExtractTime

Расчет числа дней с Нового Года до начала месяца

Вспомогательная функция DaysToMonth будет необходима при преобразованиях даты. По своему устройству она похожа на EndMonday.

Таблица dom содержит суммарное число дней с начала обычного года по месяцам (См. Рис. 13). Для високосного года поправка (+1) нужна начиная с марта ($m > 2$).



```

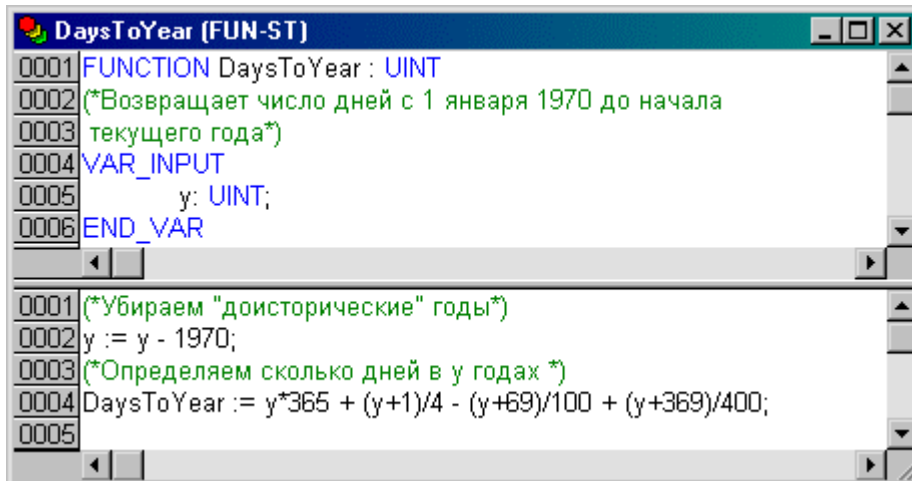
0001 FUNCTION DaysToMonth : UINT
0002 (*Возвращает число дней с начала года до начала месяца*)
0003 VAR_INPUT
0004     y,m: UINT;
0005 END_VAR
0006 VAR_CONSTANT
0007     dom:ARRAY[1..12] OF UINT:=
0008         0,31,59,90,120,151,181,212,243,273,304,334;
0009 END_VAR
0010 DaysToMonth:= dom[m]+BOOL_TO_UINT(m>2 AND IsLeapYear(y));
0011 END_FUNC

```

Рис. 13 Функция DaysToMonth

Расчет числа дней до начала года

Функция DaysToYear также является вспомогательной (См. Рис. 14). Она возвращает число дней от принятого начала отсчета даты до начала указанного года. Иными словами это точное число дней в годах прошедших с 1 января 1970 года.



```

0001 FUNCTION DaysToYear : UINT
0002 (*Возвращает число дней с 1 января 1970 до начала
0003 текущего года*)
0004 VAR_INPUT
0005     y: UINT;
0006 END_VAR
0007 END_FUNCTION
0008 (*Убираем "доисторические" годы*)
0009 y := y - 1970;
0010 (*Определяем сколько дней в y годах *)
0011 DaysToYear := y*365 + (y+1)/4 - (y+69)/100 + (y+369)/400;
0012 END_FUNCTION

```

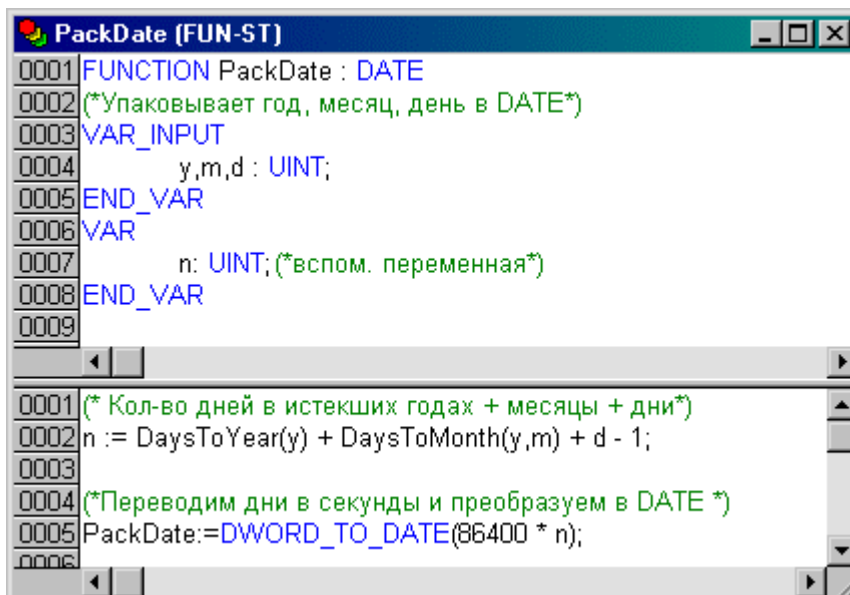
Рис. 14 Функция DaysToYear

Сложность здесь состоит в учете числа прошедших високосных лет. Первый год, где нужна поправка на предшествующие високосные годы – 1973. Для него $y=3$. Выражение $(y+1)/4$ содержит целочисленное деление и дает необходимую дополнительную единицу. Оставшаяся часть формулы отнимает число лет кратное 100 и не кратное 400. Эти тонкости впервые «включаются» в 2001 году. Но 2000 год был високосным, исключать его ненужно. Поэтому до 2101 года выражение можно существенно сократить:

$$\text{DaysToYear} := y*365 + (y+1)/4;$$

Упаковка года, месяца и дня в DATE

Параметрами функции PackDate служат год y , месяц m и день d . Функция возвращает значение типа DATE. Благодаря применению вспомогательных функций, текст PackDate получается совсем простым (См. Рис. 15).



```

0001 FUNCTION PackDate : DATE
0002 (*Упаковывает год, месяц, день в DATE*)
0003 VAR_INPUT
0004     y,m,d : UINT;
0005 END_VAR
0006 VAR
0007     n: UINT; (*вспом. переменная*)
0008 END_VAR
0009 END_FUNCTION
0010 (* Кол-во дней в истекших годах + месяцы + дни*)
0011 n := DaysToYear(y) + DaysToMonth(y,m) + d - 1;
0012 (*Переводим дни в секунды и преобразуем в DATE *)
0013 PackDate:=DWORD_TO_DATE(86400 * n);
0014 END_FUNCTION

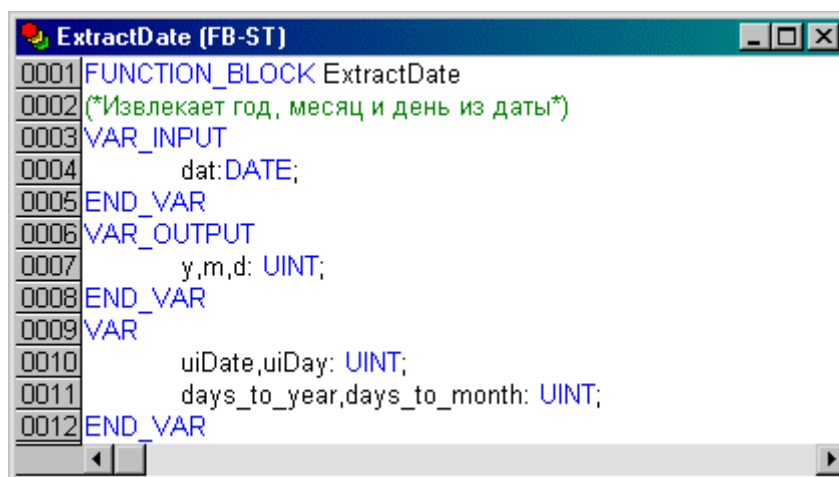
```

Рис. 15 Функция PackDate

Извлечение года, месяца и дня из DATE

Извлечение года, месяца и дня из DATE задача более сложная, чем упаковка.

Сложность заключается в том, что, зная число суток, в одно действие не удастся точно рассчитать, сколько в них полных лет или месяцев, так как они имеют переменную длительность. С другой стороны, мы уже знаем, как по месяцу или году определить количество дней. Поэтому будем действовать так: находим год или месяц приближенно, с возможным «промахом» вверх на 1. Далее проверяем с помощью обратного преобразования, не ошиблись ли мы, и в случае ошибки корректируем найденной значение.



```
0001 FUNCTION_BLOCK ExtractDate
0002 (*Извлекает год, месяц и день из даты*)
0003 VAR_INPUT
0004     dat:DATE;
0005 END_VAR
0006 VAR_OUTPUT
0007     y,m,d: UINT;
0008 END_VAR
0009 VAR
0010     uiDate,uiDay: UINT;
0011     days_to_year,days_to_month: UINT;
0012 END_VAR
```

Рис. 16 Раздел объявлений ExtractDate

Функциональный блок ExtractDate получился достаточно объемным поэтому раздел (См. Рис. 16) объявлений и ST текст показаны отдельно (См. Рис. 17).


```

0001 (*Переведем дату в число дней с 1970г.*)
0002 uiDate := DWORD_TO_UINT(DATE_TO_DWORD(dat)/86400);
0003
0004 y := uiDate/365 + 1970; (*Грубо прикинем год*)
0005
0006 (*Определяем кол-во дней в истекших годах*)
0007 days_to_year := DaysToYear(y);
0008
0009 IF uiDate < days_to_year THEN (*Уточним не лишний ли год?
0010     y := y-1;
0011     days_to_year := DaysToYear(y);
0012 END_IF
0013
0014 uiDay := uiDate - days_to_year + 1; (*Определим день года
0015
0016 m := MIN(uiDay/29 + 1, 12); (*Грубо прикинем месяц*)
0017
0018 (*Определяем кол-во дней в истекших месяцах*)
0019 days_to_month := DaysToMonth(y,m);
0020
0021 (*Уточним не лишний ли месяц?*)
0022 IF uiDay <= days_to_month THEN
0023     m := m-1;
0024     days_to_month := DaysToMonth(y,m);
0025 END_IF
0026
0027 d := uiDay - days_to_month; (*Осталось вычислить день*)

```

Рис. 17 Функциональный блок **ExtractDate**

Точно по времени и только один раз

Если вы уже имеете переменную, отражающую время суток то, нет ни чего проще, чем реализовать программный будильник. Иными словами выполнить некоторое действие точно по времени. Решение «в лоб» выливается в периодическую проверку условия:

```

IF MyRealTime = TOD#07:50 THEN
    (* выполняем работу по будильнику *)
END_IF

```

Все хорошо, но... такой будильник ненадежен. Он требует абсолютно точного совпадения MyRealTime с заданной константой. Если MyRealTime будет иметь значение на 1 мс большее, будильник не сработает! Представьте себе что:

- период проверки оказался больше чем 1 мс;
- обслуживание прерывания или более приоритетная задача «проглотили» данную миллисекунду;
- питание было выключено на пару минут;

Если первые две сложности можно преодолеть помещением проверки в процедуру обработки прерывания таймера, то в последнем случае это не спасет. Этот «черный» список легко продолжить.

Для надежной реализации будильника обязательно нужно обдумать, что делать в случае вынужденного опоздания. Чаще всего, уже ни чего делать не нужно. Тогда достаточно задать интервал времени, в течение которого данное действие еще имеет смысл. Если действие должно быть выполнено только единственный раз, потребуется еще и флаг – признак выполнения.

```

0001 IF EnableCock THEN
0002 IF MyRealTime>TOD#07:50 AND MyRealTime<TOD#08:00 THEN
0003     EnableCock := FALSE;
0004     (* выполняем работу по будильнику *)
0005 END_IF
0006 ELSE
0007 IF MyRealTime > TOD#09:00 THEN
0008     EnableCock := TRUE;
0009 END_IF
0010 END_IF
0011

```

Рис. 18 Программный «будильник»

В примере на Рис. 18 действие выполняется единственный раз, не раньше 7:50 и не позднее 8:00. Обратите внимание, что флаг разрешения будильника восстанавливается часом позднее. Это предотвращает повторное срабатывание будильника при коррекции времени назад около 8:00. Особенно внимательно нужно обрабатывать события с 2х до 3х часов ночи, которые могут совпасть с автоматическим переходом на летнее время.

Часы реального времени

Стандартный функциональный блок RTC (Real Time Clock) имеет входы разрешения работы EN (Enable BOOL) и установки времени PDT (Preset DT). При переходе EN из FALSE в TRUE, часы начинают отсчет времени от значения PDT. Выход Q (BOOL) повторяет значение EN. Выход CDT (Current DT) дает текущее значение даты и времени (См. Рис. 19).

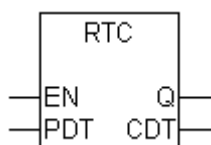


Рис. 19 Функциональный блок RTC

Если функциональный блок RTC не реализован в вашей системе программирования, его не сложно построить самостоятельно. В качестве примера приведем ST текст функционального блока RTC выполненного на базе стандартного таймера TON и детектора переднего фронта R_TRIG (См. Рис. 20, Рис. 21).

```

fbRTC (FB-ST)
0001 FUNCTION_BLOCK fbRTC
0002 (*Часы реального времени*)
0003 VAR_INPUT
0004     EN : BOOL;
0005     PDT: DT;
0006 END_VAR
0007 VAR_OUTPUT
0008     Q: BOOL;
0009     CDT: DT;
0010 END_VAR
0011 VAR
0012     DiffTime: TON := (PT:=T#20h);
0013     ResDT: DT;
0014     RunTrig: R_TRIG;
0015 END_VAR

```

Рис. 20 Раздел объявлений fbRTC

```

fbRTC (FB-ST)
0001 RunTrig(CLK := EN);
0002
0003 IF EN THEN
0004     IF RunTrig.Q THEN                (*рестарт часов*)
0005         ResDT := PDT;                (*начало отсчета*)
0006         DiffTime(IN := FALSE);      (*рестарт таймера*)
0007         DiffTime(IN := TRUE);
0008     ELSE                               (*нор. режим CDT = DT рестарт + таймер*)
0009         DiffTime;
0010         CDT := DWORD_TO_DT(DT_TO_DWORD(ResDT)
0011             + TIME_TO_DWORD(DiffTime.ET)/1000);
0012         IF DiffTime.ET > t#1h THEN
0013             (*рест. таймера кажд. час для предотвр. его переполнения*)
0014             ResDT := CDT; (*начало отсчета*)
0015             DiffTime(IN := FALSE); (*рестарт таймера*)
0016             DiffTime(IN := TRUE);
0017         END_IF
0018     END_IF
0019 END_IF
0020 END_IF
0021 Q := EN;
0022

```

Рис. 21 Функциональный блок fbRTC

Главный недостаток функционального блока RTC это необходимость его начальной инициализации при каждом перезапуске контроллера. Многие ПЛК имеют встроенные аппаратно реализованные часы реального времени. Разные экземпляры функционального блока RTC не связаны между собой и могут отсчитывать разное время. Аппаратные часы лишены такой возможности. Разумным компромиссом является реализация функционального блока RTC автоматически инициализирующегося значением аппаратных часов, если начальное значение PDT не задано.

Основное достоинство аппаратных RTC часов это сохранение хода при выключенном питании. Без поддержки такой функции установка аппаратных часов не имеет смысла. Поэтому практически все микросхемы RTC совмещают в себе небольшое статическое ОЗУ. Все они оснащены встроенной схемой контроля, позволяющей программно определить факт нарушения питания. Тем не менее, аппаратные RTC нужно обязательно настроить при первом включении. Функции источника точного времени они выполняют не лучше, чем программная реализация.

Не мало технологических задач требуют RTC часов только на первый взгляд. При внимательном анализе оказывается, что процесс управления привязан не к UTC, а к началу смены или включению оборудования. Сдвиг расписания в праздничные дни меняет абсолютные времена операций. Для человека это совершенно естественно. Применение обычного таймера, запускаемого внешним сигналом или даже кнопкой, оказывается здесь практичнее, чем выполнение операций по часам реального времени.

Вследствие рассмотренных сложностей, функциональный блок RTC представлен в стандарте МЭК только в качестве примера. Системы программирования не обязаны его поддерживать.

Таймеры МЭК

В отличие от часов реального времени, таймеры не привязаны к времени или календарю. Их работа состоит в формировании задержек заданной длительности. Задержки, формируемые таймерами МЭК 61131-3, не влияют на время выполнения прикладной программы. Представьте себе, что таймер это внешний прибор, вход и выход которого подключен к вашему контроллеру (См. Рис. 22). Именно так работают стандартные таймеры. Опрашивая таймер, вы узнаете, что заданный интервал времени истек. Как быстро вы обнаружите этот факт, зависит не только от дискретности самого таймера, но и от периода его опроса в вашей программе.

Вход и выход таймера это логические переменные BOOL. Все таймеры МЭК реализованы как функциональные блоки. Это означает, что вы можете создать любое количество независимых экземпляров одностипных таймеров.

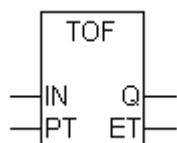


Рис. 22 Функциональный блок таймер TOF

В стандарте предусмотрены три вида таймеров. Все они имеют одинаковый интерфейс:

- вход IN (BOOL) сигнал запуска таймера;
- вход PT (TIME) задание значения времени;
- выход Q (BOOL) выход таймера;
- выход ET (TIME) время, прошедшее от запуска таймера.

ТР импульс - формирует импульс заданной PT длительности по переднему фронту IN. Последующий импульс можно сформировать только после

окончания предшествующего, т.е. входной сигнал не может влиять на длительность импульса (См. Рис. 23).

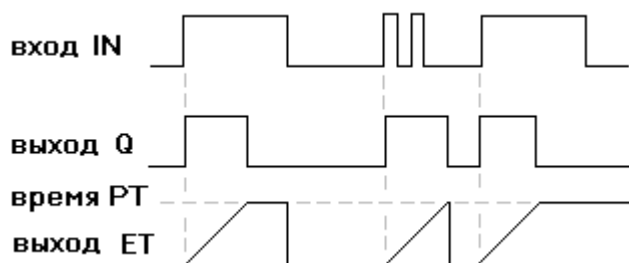


Рис. 23 Временная диаграмма работы таймера TP

TON задержка включения - включает (TRUE) выход Q с задержкой относительно переднего фронта IN на время PT. Если выключить IN до конца интервала PT, то отсчет времени будет прерван (См. Рис. 24).

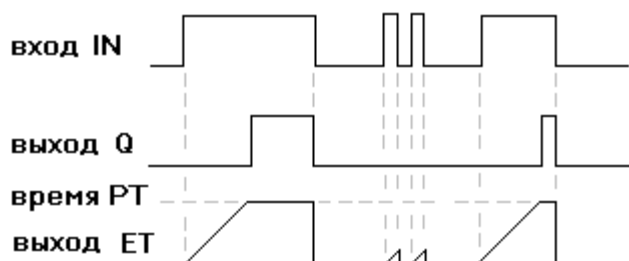


Рис. 24 Временная диаграмма работы таймера TON

TOF задержка выключения - выход Q включается вместе со входом IN. Для выключения Q, входной сигнал должен иметь низкий уровень не менее времени заданного PT. (См. Рис. 25).

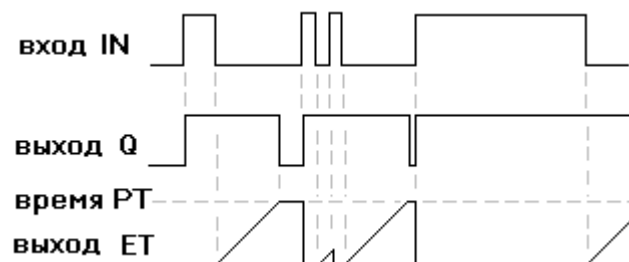


Рис. 25 Временная диаграмма работы таймера TOF

Селекция импульса

Мы уже использовали таймер TON в часах реального времени. В качестве еще одного примера применения стандартных таймеров реализуем функциональный блок PulseSel. Он должен формировать единичный выходной импульс, на каждый входной сигнал определенной длительности. Годными считаются входные импульсы в пределах от MinDr, до MinDr+OverDr.

FBD реализация PulseSel показана на Рис. 26.

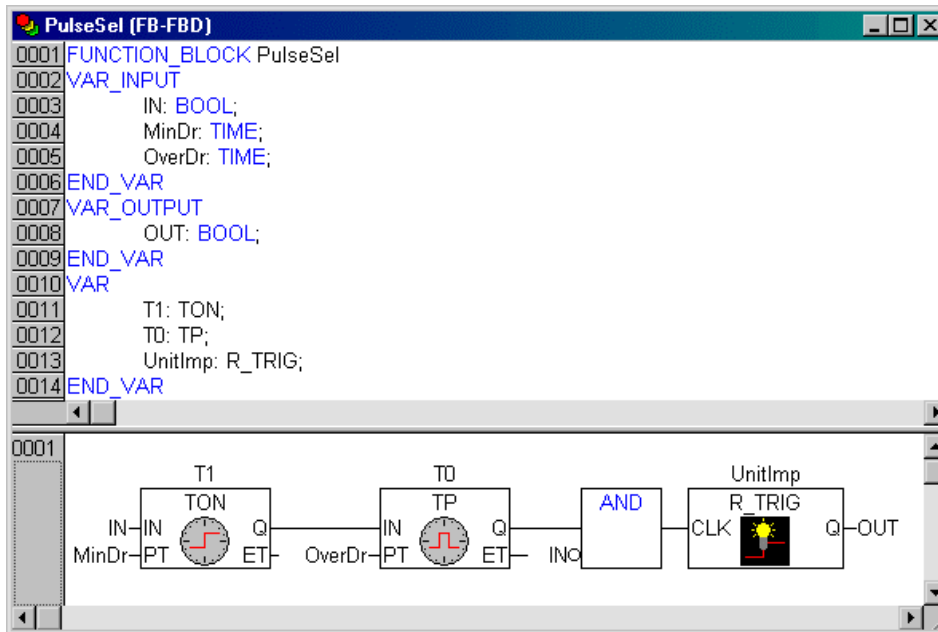


Рис. 26 Функциональный блок PulseSel

Работу PulseSel поясняет Рис. 27. Таймер T1 запускается по фронту входного импульса и обеспечивает фильтрацию коротких импульсов. Если минимальная длительность обеспечена, запускается таймер T0. Он отмеряет максимальную длительность импульса.

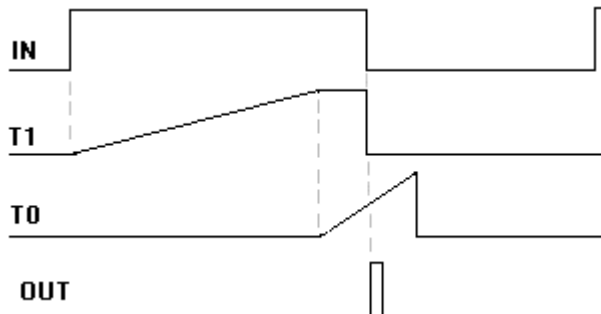


Рис. 27 Временная диаграмма работы PulseSel

Внутреннее устройство таймеров

Для того чтобы построить в вычислительной машине таймер необходимо иметь аппаратную поддержку. На самом деле, для любого числа программных таймеров нужен всего один источник точных интервалов времени. Достаточно иметь одну системную переменную, назовем ее для определенности `SYS_TIME`. По каждому «тику» – прерыванию таймера, ее значение увеличивается на единицу. Дискретность работы таймеров зависит от интервала обновления `SYS_TIME`. Все экземпляры функциональных блоков таймеров «засекают» стартовое время и рассчитывают длительность по `SYS_TIME`. Переменная `SYS_TIME` должна иметь достаточную разрядность, чтобы исключить возможность переполнения за время между вызовами таймеров. Ее разрядность может быть выше разрядности процессора. В этом случае, при чтении ее значения необходимо запрещать прерывания. Внутренняя реализация функциональных блоков таймеров является аппаратно-зависимой и выполняется на системном уровне.

При использовании стандартных таймеров нельзя полагаться на то, что повторный вызов экземпляра функционального блока в одном рабочем цикле даст различные результаты. Значение `SYS_TIME` может обновляться по прерыванию или синхронно с обновлением входов. Это зависит от реализации системы исполнения. Не используйте в своих программах циклы (`WHILE`, `REPEAT`) с условием окончания итераций по таймеру.

Холодный рестарт: упал, очнулся - гипс

Неожиданный провал во времени с полной амнезией – вполне вероятный исход аварии питания. При перезапуске, модуль инициализации вашего компилятора или система исполнения ПЛК установят начальные (по умолчанию нулевые) значения переменных. Это полезная услуга. В результате, прикладная программа стартует с «чистого листа». В большинстве случаев это правильно.

Однако, существует немало технологий, в которых отдельные процессы продолжают развиваться сами собой даже без нашего руководства. За время отключения может замерзнуть реактив, высохнуть смазка, испариться тонна спирта и т.д. Здесь запуск программы управления превращается в целую «военную операцию». Программа должна исследовать обстановку и корректно взять все под свой контроль. Конечно, нельзя дать исчерпывающее решение такой задачи. Оно определяется технологией. Тем не менее, некоторые типовые проблемы начальной инициализации стоит рассмотреть.

В случаях, когда важна предыстория, необходимо использовать контроллеры имеющие «энергонезависимую» память. Не зависимо от аппаратной реализации такой памяти, всегда существует некоторая вероятность того, что данные будут повреждены. Целиком переложить задачу контроля корректности данных на системный уровень не реально. Система может проверить, что данные не исказились при хранении, но не может дать гарантию, что они соответствуют вашей задаче. Возможно, модуль памяти или целиком ПЛК заменен. Поэтому в ответственных системах, прикладной контроль достоверности данных обязателен.

Как убедиться что данные те, что надо и корректны? Наиболее часто для этого применяются «волшебные» ключи. Если память испорчена, то значения в ячейках памяти будут иметь некоторые похожие значения. Это не случайные числа. Создать процессорный блок, в котором при переходных процессах каждая ячейка памяти будет записана разным способом нереально. Чаще всего, потеря памяти это нули или единицы во всех разрядах. Создадим в области энергонезависимой памяти одну или несколько переменных. Для определенности используем шестнадцатиразрядное целое без знака (`UINT`). Выберем некоторую осмысленную константу. Осмысленность означает выбор значения, которое, по нашему опыту, не может случайно сформироваться при коммутации питания. Например, `AA55h`. При включении питания сравниваем значения ключевых переменных с константой. Если совпало – память почти гарантировано цела. Если не совпало – память точно потеряна. В конце инициализации присваиваем ключевым переменным значение константы, для следующего раза.

Второй простой дополнительный рубеж защиты – оценка данных на разумный диапазон значений и взаимную соразмерность. Например: значения дат событий оказываются в далеком будущем; изготовлено 255 кирпичей, их суммарный вес 0кг; на RTC часах 1970 год и т.д. Использование естественных свойств данных удобно тем, что не приводит к дополнительным затратам в

нормальной работе. Вы легко определите подобные критерии, анализируя переменные своей программы.

Если это не удастся, необходимо ввести искусственные избыточные элементы контроля достоверности. Их придется поддерживать при каждом изменении значения переменных. Это может быть полное дублирование либо циклическая контрольная сумма защищаемого блока данных. Для особо важных данных приходится проводить контроль не только при начальной инициализации, но и при каждом чтении данных.

Еще одна известная проблема инициализации – определение длительности времени, в течение которого система была отключена. Сложность заключена в необходимости выполнения отсчета времени в «выключенном» состоянии. Это можно сделать только при наличии аппаратных RTC и энергонезависимой памяти. В рабочей программе, с необходимым периодом t , вы записываете в некоторую энергонезависимую переменную значение даты и времени. Это метка, подтверждающая, что в указанное время система еще была «жива». При начальной инициализации старая метка укажет время отключения, с точностью $\pm t/2$. Разность текущего времени и метки даст время простоя.

Собачий таймер сторож

Хороший сторож должен проявлять себя, только в исключительной ситуации. Основная его работа это мирный сон. Для экстренного пробуждения сторожу необходима помощь аппаратуры охраны или собственный тренированный слух и нюх.

Цель использования сторожевого таймера (watchdog) в вычислительной системе сводится к обнаружению «зависания». «Завод» сторожевого таймера не должен оканчиваться, иначе он «сработает» и выполнит перезапуск системы. Для этого в рабочую программу добавляется периодически выполняемая процедура «взвода» сторожевого таймера.

Таймером можно защищать отдельную задачу, процессорное ядро или систему целиком. Контроль времени отклика отдельных задач может взять на себя ядро операционной системы. Это очень полезная функция, конечно, если процессор еще «жив». Для контроля самого процессора применяют специальный аппаратный таймер.

В современных микроконтроллерах это штатный встроенный узел. Программно вы можете взвести его, включить или отключить и изменить время контроля. Здесь существуют две тонкости, с которыми вы обязательно столкнетесь, если не сталкивались раньше. По включению питания системы сторож запрещен. Вы можете позволить себе выполнить достаточно объемную и длительную начальную инициализацию. Далее вы разрешаете сторож. Рано или поздно он «срабатывает», теперь уже процессор перезапускается с разрешенным сторожем. Что будет дальше понятно – процессор никогда не сможет пройти инициализацию до конца. Именно поэтому хорошие компиляторы имеют специальную опцию отключения модуля начальной инициализации. Вы должны либо запретить сторож на время начальной инициализации, что убивает саму идею, либо поддержать его нормальную работу в этой процедуре, что непросто. Желание упростить прикладную программу неизменно порождает у программистов идею поддержки сторожа в процедуре обслуживания прерывания системного таймера. Это грубая ошибка. Представьте себе, что процессор «застрял» в бесконечном цикле. Сторож

должен обнаруживать эту ситуацию. Однако система прерываний работает нормально и регулярно «взводит» сторож.

Самые изощренные сторожевые таймеры обеспечивают сквозной контроль всей системы, включая удаленные интеллектуальные модули вывода-вывода. Идея достаточно проста: с одного или нескольких выходов контроллера снимаются программно формируемые контрольные импульсы. Они поступают на внешний аппаратный таймер сторож, подключенный к сбросу процессора, либо через дискретный вход «завязываются» в процедуру обслуживания встроенного сторожа.

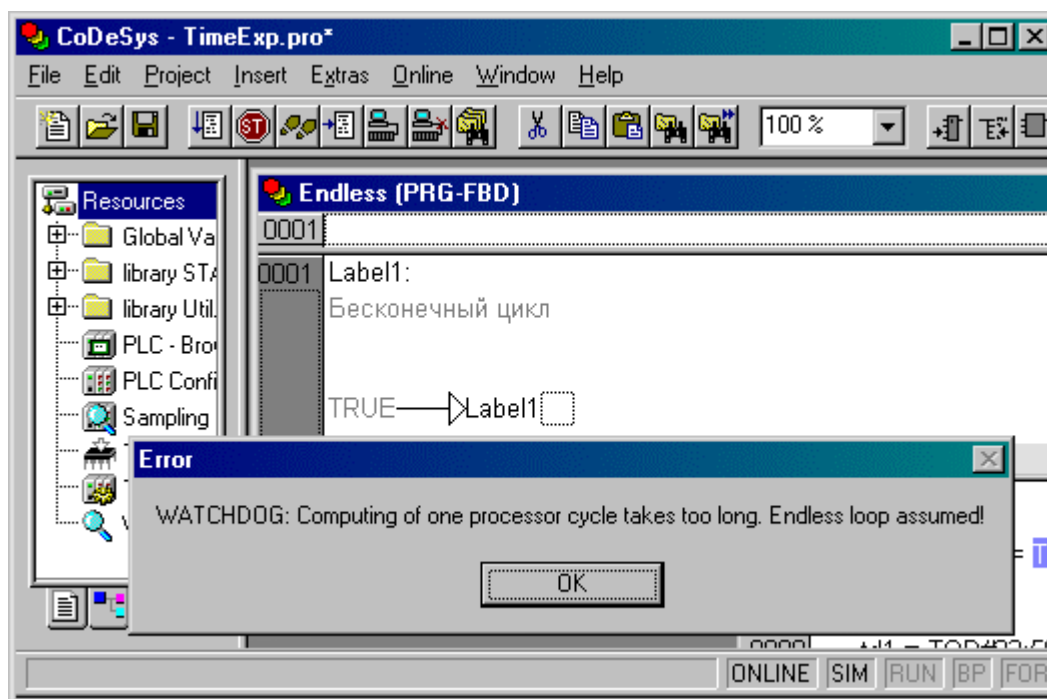


Рис. 28 Сторожевой таймер CoDeSys «поймал» бесконечный цикл

Для обеспечения такого контроля не обязательно опускаться на системный уровень. Его можно организовать и в прикладной высокоуровневой программе. К сожалению, поддержка сторожевого таймера является системно-зависимой. Вы не найдете в МЭК инструментах оператора сброса процессора. Но если в системе есть встроенный сторожевой таймер выполнить рестарт процессора из прикладной программы реально. Для этого достаточно спровоцировать сторожевой таймер, умышленно созданным бесконечным циклом (См. Рис. 28).

Заключение

Надеемся, вы нашли в этой статье какие-либо полезные идеи. Если тема показалась вам интересной, возможно вы найдете возможность поделиться своим опытом решения подобных задач со всеми читателями журнала.

Файлы с исходными текстами всех описанных примеров вы найдете в разделе информации сайта компании Пролог www.prolog.smolensk.ru и на Интернет страницах журнала www.asucontrol.ru.



Рис. 29 Семь бед, один Reset

Литература

- [1] В. Гинзбург, В. Пульвер, Сигналы точного времени, «Радиолобитель», 1927, № 2
- [2] International Electrotechnical Commission, TC65/WG7, Final draft – IEC 61131-3, 2nd ed. Programmable controllers – programming languages.
- [3] Петров И.В. Программируемые контроллеры. Стандартные языки и приемы прикладного проектирования. – М.: СОЛОН-Пресс, 2004
- [4] Демидов В.Е., Время, хранимое как драгоценность. – М.: Знание, 1977
- [5] В. Заикин, Стационарные часы реального времени с синхронизацией по радиоканалу, Схемотехника N4, 2001
- [6] Постановление Правительства РФ «О порядке исчисления времени на территории Российской Федерации» (в ред. 23.04.96 N511)

Статья опубликована в журнале:

Промышленные АСУ и контроллеры N 7,8,9 2004, © НАУЧТЕХИЗДАТ 2004г.

www.asucontrol.ru