Пример импортозамещения программно-аппаратной архитектуры систем управления на вычислительных средствах ограниченной производительности

Т.К.Грингауз[1], А.Н.Онин[2]

[1]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, gring@niisi.ras.ru;

[2] ФГУ ФНЦ НИИСИ РАН, Москва, Россия, alexii@niisi.ras.ru

Аннотация. Микроконтроллер КОМДИВ-МКЭ отечественной разработки входит в состав интерфейсных модулей семейства ИМ-АЗС. Модули предназначены для применения в качестве контроллеров технологических процессов и оборудования автозаправочной станции. На модули портирована проприетарная программа управления. Описан принцип функционирования программы. Приведен перечень ее доработок при портировании.

Ключевые слова: импортозамещение, интерфейсный модуль, программируемый логический контроллер, древовидная сеть, технологическое оборудование, АЗС, уровнемер, ТРК.

1. Введение

В рамках работ по импортозамещению программно-аппаратных комплексов разработано семейство интерфейсных модулей ИМ-АЗС. Модули предназначены для использования в составе программно-аппаратного комплекса автозаправочной станции (далее – «АЗС»). Модули применяются в качестве программируемых логических контроллеров (далее – «ПЛК») технологических процессов и оборудования АЗС. ПЛК обеспечивают управление топливно-раздаточными колонками (ТРК), подсистемой налива и контроля уровня топлива (далее – «уровнемер»), информационной стелой, передачей данных между ПЛК и управляющей информационно-кассовой подсистемой через сервер АЗС.

В состав интерфейсных модулей входит разработанный в ФГУ ФНЦ НИИСИ РАН микроконтроллер КОМДИВ-МКЭ. Первоначальное назначение микроконтроллера – мониторинг параметров линий электропитания. Производительность микроконтроллера ограниченна в связи с малым объемом оперативной памяти (128 КБ).

Программное обеспечение модуля ИМ-АЗС (далее – «программа ИМ-АЗС») разрабатывается на основе прототипа – проприетарной программы пользователя. Пользовательская программа предоставлена в виде исходного кода на языке Си. Анализ кода и описание алгоритма программы-прототипа стали неизбежным этапом разработки программы ИМ-АЗС.

В статье описан принцип функционирования программы-прототипа, выявленный реверс-инжинирингом, а также особенности его реализации в программе ИМ-АЗС.

2. Модель вычислительной системы АЗС

Модель вычислительной системы АЗС представлена на рис.1.

Вычислительная система АЗС представляет собой двухуровневую древовидную структуру, неоднородную по типу физических соединений узлов. Корнем дерева является сервер АЗС, а узлами – программируемые логические контроллеры (далее – «ПЛК»). К ПЛК подключается технологическое оборудование АЗС.

Сервер АЗС – это миникомпьютер Intel NUC (далее – «ЭВМ NUC»).

В качестве ПЛК используются интерфейсные модули семейства ИМ-АЗС (далее – «интерфейсные модули» или «модули ИМ-АЗС»).

Примечание. Термины «ПЛК» и «интерфейсный модуль» употребляются применительно к узлу в зависимости от контекста. Термин «ПЛК» обозначает роль узла в технологическом процессе. Термин «интерфейсный модуль» определяет тип устройства, реализующего узел.

На физическом уровне модули ИМ-АЗС подключаются к ЭВМ NUC по интерфейсу LIN-«токовая петля». Сервер АЗС обменивается данными с ПЛК по протоколу IDC. Канальный, транспортный и прикладной уровни модели передачи данных между узлами сети регламентируются специальным протоколом (далее – «про-

токол IDC»). Описание протокола IDC приведено в разделе 7 настоящей статьи.

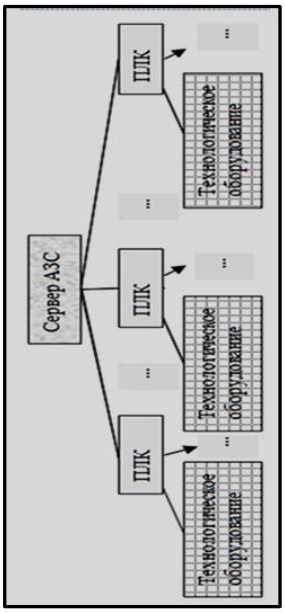

Рис. 1. Модель вычислительной системы АЗС

Под технологическим оборудованием АЗС понимаются устройства, подключаемые к ПЛК. Технологическому оборудованию соответствует третий (нижний) уровень иерархии (см. рисунок 1). ПЛК управляет подключенными к нему устройствами.

Технологическое оборудование подключается к интерфейсным модулям по последовательному интерфейсу UART (на физическом уровне поддерживаются протоколы RS-232, RS-485, DIN токовая петля) или по интерфейсу «дискретные сигналы». В статье будем рассматривать только подключение по портам UART.

Устройства классифицируются по типам. Примеры типов устройств: топливно-раздаточная колонка (ТРК), уровнемер, стела. Управление устройствами конкретного типа осуществляется посредством специфичной для этого типа системы команд. Сервер АЗС передает команды, адресованные устройству, в ПЛК. В ПЛК команды передаются по протоколу IDC (далее – «IDC-команды»).

Тип устройств подразделяется на виды. Примеры видов ТРК: WayneDresser, Топаз. Пример вида уровнемера: Струна. Протокол взаимодействия устройства с ПЛК специфичен для каждого вида устройств.

В таблице на рисунке 2 представлен перечень типов технологического оборудования, а также протоколов взаимодействия с ПЛК, поддерживаемых текущей версией программы ИМ-АЗС.

3. Состав, назначение, идентификация, конфигурация технологического оборудования АЗС

На рисунке 3 изображен пример состава и соединения основных блоков АЗС, обеспечивающих хранение топлива и его отпуск потребителям (стела на рисунке не изображена).

Подачу топлива из подземных резервуаров в бак автомобиля обеспечивают топливно-раздаточные колонки. На АЗС может быть установлено несколько топливно-раздаточных колонок.

Тип интерфейсного модуля	Тип оборудования	Интерфейс связи
ИМ-АЗС-UARTA	Уровнемер	RS-232
ИМ-АЗС-UART	Стела	RS-485
	ТРК	RS-485
ИМ-АЗС-ТП	ТРК	IEC 62056-21/ DIN токовая петля
ИМ-АЗС-ДС	ТРК	дискретные сигналы

Рис. 2.Типы технологического оборудования

Топливно-раздаточные колонки бывают односторонними и двухсторонними. Далее под ТРК подразумевается одна сторона колонки. На АЗС поддерживается сквозная нумерация ТРК.

Налив топлива происходит через рукав ТРК («пистолет»). На ТРК может быть несколько рукавов. Разные рукава одной ТРК подают разные виды топлива. Разные рукава одной ТРК не могут осуществлять налив одновременно. На АЗС поддерживается сквозная нумерация рукавов.

Топливо хранится в подземных резервуарах. В одном резервуаре хранится один вид топлива. Разные резервуары могут хранить разные виды топлива. Каждому виду топлива соответствует своя цена. На АЗС резервуары и виды топлива нумеруются сквозным образом. Таблица видов топлива с ценами отображается на стеле.

Из резервуара топливо подается в рукава ТРК. В один рукав подается топливо из одного резервуара. Из одного резервуара топливо может подаваться в несколько рукавов.

Резервуар подключен к контроллеру уровнемера (далее – «уровнемер»). К уровнемеру может быть подключено несколько резервуаров (максимальное количество зависит от вида уровнемера). В резервуаре установлено несколько датчиков. Датчик измеряет следующие параметры: объем (уровень), плотность,

температуру, давление, уровень подтоварной воды. Для вычисления объема по уровню используется калибровочная таблица. Количество датчиков, состав измеряемых параметров и калибровочная таблица конфигурируются для каждого уровнемера.

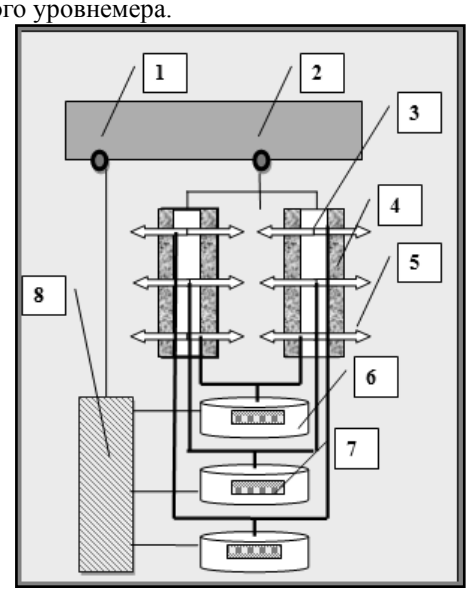

Рис. 3.Логическая схема подключения технологического оборудования АЗС.
Обозначения: 1 – интерфейсный модуль, порт UART (RS-232), 2 – интерфейсный модуль, порт UART (RS-485 или DIN-токовая петля), 3 – топливно-раздаточная колонка (двусторонняя),4 – сторона топливно-раздаточной колонки (ТРК) ,5 – рукав (пистолет) для налива топлива, 6 – резервуар с топливом, 7 – датчики, 8 – контроллер уровнемера.

Контроллер уровнемера подключается к интерфейсному модулю через порт UART (RS-232). К порту можно подключить один уровнемер.

ТРК подключаются к интерфейсному модулю через порт UART (RS-485 или DIN-«токовая» петля) через блок сопряжения. К порту может быть подключено несколько ТРК (максимальное количество зависит от вида ТРК).

Стела подключается к интерфейсному модулю через порт UART (RS-485). К порту можно подключить одну стелу.

В интерфейсных модулях ИМ-АЗС реализовано по 2 порта для подключения технологического оборудования: порты RS-485 для модулей ИМ-АЗС-UART, порты RS-232 для модулей ИМ-АЗС-UARTA, порты DIN-«токовая петля» для модулей ИМ-АЗС-ТП. Во всех исполнениях модуля ИМ-АЗС присутствует порт UART (интерфейс LIN-«токовая петля») для взаимодействия с сервером АЗС по протоколу IDC.

Программа интерфейсного модуля написана в расчете на произвольное количество портов UART. Реальное количество портов, состав технологического оборудования, взаимосвязь ТРК, рукавов, резервуаров и видов топлива описываются конфигурационными данными программы. Программа конфигурируется индивидуально для каждого интерфейсного модуля в составе вычислительной системы АЗС.

В программе поддерживаются следующие способы идентификации конфигурируемых объектов.

Для порта UART: порядковый номер при сквозной нумерации портов на интерфейсном модуле.

Для ТРК:

1) порядковый номер при сквозной нумерации на АЗС («сетевой адрес»),

2) адрес в блоке сопряжения, подключенном к порту UART («канал»),

3) порядковый номер при сквозной нумерации на модуле.

Для рукава ТРК:

1) порядковый номер при сквозной нумерации («сетевой адрес»),

2) порядковый номер рукава на стороне топливно-раздаточной колонки.

Для резервуара:

1) порядковый номер при сквозной нумерации («сетевой адрес»),

2) номер резервуара в контроллере уровнемера, подключенном к порту UART («канал»),

3) порядковый номер при сквозной нумерации на модуле.

Для видов топлива: номер вида топлива.

Для обозначения типа оборудования, подключенного к порту UART (сервер АЗС, ТРК, уровнемер, стела), используются целочисленные идентификаторы.

4. Представление конфигурационных данных в программе

Конфигурирование программы интерфейсного модуля производится при первом включении с помощью IDC-команд с сервера АЗС. Далее конфигурационные данные сохраняются в энергонезависимой памяти модуля. При последующих включениях используются сохраненные данные. Конфигурационные данные могут корректироваться в процессе эксплуатации интерфейсного модуля.

Для представления конфигурационных данных в программе используются следующие структуры.

Порты UART:

```
typedef struct _UartConfiguration
{
 uint32_t baud_rate;
```

```
    uint8_t  char_size;
    uint8_t  parity;
    uint8_t  invertTx;
    uint8_t  invertRx;
    uint8_t  timeout;
    uint8_t application; // «приложение»
                         // (тип оборудования)
  } UartConfiguration;
  staticUartConfiguration uart_configuration\
  [MAX4OF_UARTS];
```

Приложения:

```
 typedef enum _UartApplication {  …

 UART_APPLICATION_IDC_BRANCH,

//связь с сервером, протокол IDC

 …

 UART_APPLICATION_LMS, // уровнемер

 UART_APPLICATION_TRK, // ТРК

…

 UART_APPLICATION_PRICEBOARD, //
стела

 …} UartApplication;
```

```
  ТРК:
  typedef struct        _TrkConfiguration
  {
  uint8_t  trk;                  // сетевой адрес
  uint8_t  module;               // номер модуля
  uint8_t  unit;                 // номер на модуле
  uint8_t  port;                 // номер порта
  uint8_t  channel;                  // канал
  uint8_t  number_of_grades;     // число рукавов
  }
  staticTrkConfigurationtrk _configuration\
  [MAX4OF_TRK];

  Рукава ТРК:
  typedef struct _NozzleConfiguration
  {
  uint8_t   nozzle;      // сетевой адрес рукава
  uint8_t   trk;             // сетевой адрес ТРК
  uint8_t   tank;     // сетевой адрес резервуара
  uint8_t   grade;    // номер рукава на стороне
  }  NozzleConfiguration;

  static                        NozzleConfiguration\
                                nozle_configuration\
 [MAX4OF_NOZZLES];

  Резервуары:
  typedef  struct    _TankConfiguration
  {
  uint8_t        tank;        //сетевойадрес
  uint8_t    gas;         // номер вида топлива
  uint8_t    channel;       // канал
  uint8_t    port;          //номер порта
  }     TankConfiguration;

  static TankConfigurationtank_configuration\
  [MAX4OF_TANKS];

  Калибровочая таблица:
  typedef struct _TankCalibration
  {
  uint16_t      level; //уровень
  uint32_t      volume; //объем
  } TankCalibration;
  static TankCalibration calibrations\
  [MAX4OF_TANKS]\
  [CALIBRATION_TABLE_SIZE] ;

  Виды топлива:
  typedef   struct     _GasConfiguration
  {
  uint8_t   gas;     // номер вида топлива
  uint8_t   pb_row;  //  строка  таблицы стелы
  }      GasConfiguration;
  staticGasConfigurationgas_configuration\
[MAX4OF_GAS];

  Стела:
  Static PriceboardRow \
  Pricepriceboard_price_table\
  [MAX4OF_GAS];
```

5. Архитектура программы. Основные понятия и определения

5.1. Однопоточность и псевдопараллелизм

Далее будем абстрагироваться от особенностей аппаратно-программной платформы, для которой предназначена программа интерфейсного модуля.

Программа реализована как однопоточное приложение с обработкой прерываний. Состав источников прерываний зависит от типа аппаратной платформы. Прерывания могут возбуждаться аппаратными таймерами, интерфейсами UART, датчиком уровня питания.

Основной поток программы исполняет несколько «процессов» псевдопараллельно. Под процессом понимается код, предназначенный для управления конкретным типом оборудования. Процесс можно рассматривать как систему

управления однотипными единицами оборудования, подключенными к интерфейсному модулю.

В программе реализованы следующие процессы:

- branch_idc_control_process() – связь с сервером АЗС по протоколу IDC,
- trk_control_process() – управление ТРК,
- lms_control_process() – управление уровнемером,
- priceboard_control_process() – управление стелой,
- led_control_process() – управление световыми индикаторами на интерфейсном модуле.

Примечание. Управление световыми индикаторами осуществляется по интерфейсу дискретных сигналов. Процесс реализован в технологических целях, далее в статье рассматриваться не будет.

Псевдопараллельное выполнение процессов обеспечивает подпрограмма-диспетчер (dispatcher()) (см. рисунок 4).

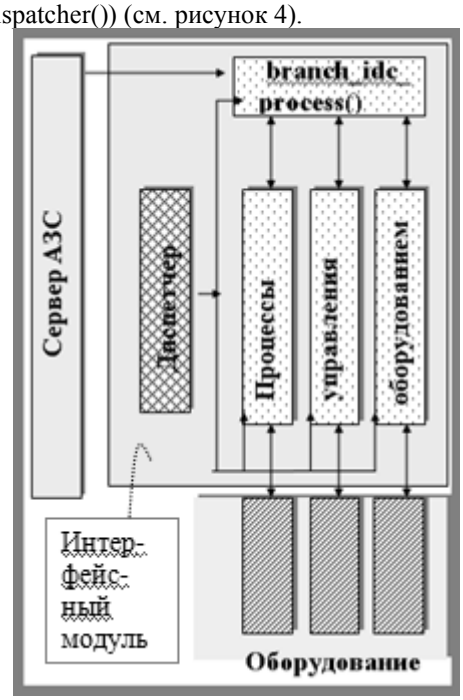

Рис. 4.Архитектура программы интерфейсного модуля

5.2. События

Под «событием» будем понимать факт, свершившийся во время выполнения программы, требующий программной реакции («обработки»). Примеры событий: 1) сигнал RESET, 2) срабатывание таймера, 3) поступление данных в порт UART, 4) падение напряжения ниже допустимого уровня.

В программе виды событий обозначаются целочисленными идентификаторами:

```
Typedef enum _OilixEvent
{
  OILIX_EVENT_RESET,…,
  OILIX_EVENT_TIMER, …,
 OILIX_EVENT_UART_RX, …,
} OilixEvent;
```

По наступлении события программа генерирует сообщение и отправляет его в очередь сообщений (далее – «очередь событий»). Тип сообщения:

```
typedef struct _OilixEventInfo
{OilixEvent    event;
  void*          user_data;
  unsigned int  id;
} OilixEventInfo;
```

(поле user_data специфицирует конкретный экземпляр события).

Поля структуры:

```
typedef struct _EventFlags {
  uint16_t   timeout;
  uint8_t    reset: 1;
  uint8_t    timer: 1;
  uint8_t    low_voltage: 1;
                                  }  EventFlags;
typedef union _EventInfo {
          EventFlags   flags;
          uint32_t     word;
                                 } EventInfo;
```

Создают сообщения и отправляют их в очередь событий основной поток программы и обработчики прерываний. Вычитывает события из очереди только основной поток (диспетчер).

5.3. Процессы

Процессы реализованы в виде функций – обработчиков событий:

```
typedef void (*ProcessHandler)\
(OilixEventInfo* event_info);
```

Каждый процесс обрабатывает свой набор событий. Процесс описывается следующим типом данных:

```
typedef struct _ProcessInfo   {
  ProcessHandler      handler;
  EventInfo             events;}  ProcessInfo;
```

Основной поток программы при запуске формирует список процессов:

```
ProcessInfo *process_list[] =
{
   &led_control_process_info,
   &lms_control_process_info,
    &trk_topaz_control_process_info,
    …..
     &branch_idc_process_info,
};
```

Далее список процессов обрабатывается диспетчером.

5.4. Диспетчер

Диспетчер выполняет бесконечный цикл. В теле цикла:

- вычитывает сообщение из очереди событий,
- последовательно вызывает все обработчики события в соответствии со списком процессов, передавая им вычитанное событие в качестве параметра.

Обработчики событий игнорируют события, не предназначенные своему процессу. Таким образом, можно считать, что каждый процесс работает циклически, асинхронно по отношению к другим процессам, в каждой итерации обрабатывая очередное предназначенное ему событие. Программу интерфейсного модуля можно представлять себе как множество параллельных событийно-управляемых систем [1].

На каждой итерации диспетчер анализирует уровень напряжения. При падении напряжения ниже критического уровня диспетчер сохраняет состояние процессов в энергонезависимой памяти (см. раздел 8 настоящей статьи).

5.5. Обмен данными между процессами

Процессы хранят свои конфигурационные данные и данные о состоянии управляемых объектов в статической области памяти. Доступ к этим данным из других процессов осуществляется посредством глобальных интерфейсных функций. Библиотеки интерфейсных функций разработаны для всех типов оборудования, поддерживаемых программой.

5.6. Диверсификация протоколов управления

Для каждого типа технологического оборудования существуют разные аппаратные реализации. Примеры видов ТРК: Wayne Dresser, Топаз. Протокол взаимодействия устройства с интерфейсным модулем специфичен для каждого вида устройств. Предполагается, что на одном интерфейсном модуле для всех однотипных единиц оборудования используется один и тот же протокол управления. На разных экземплярах интерфейсного модуля протоколы для одного и того же типа оборудования могут различаться. Протокольно-зависимые фрагменты кода выделены в отдельные программные модули. Выбор протокола осуществляется условной компиляцией.

6. Управление оборудованием с интерфейсного модуля по портам UART

6.1. Обобщенная постановка задачи управления для разных типов оборудования

Проанализированы программные тексты процессов для разных типов оборудования (уровнемер, ТРК, стела). Каждый процесс реализован в отдельном наборе программных модулей, тексты для разных процессов не зависят друг от друга. Тем не менее, можно выделить в них общие черты, сходные по способу реализации. Сходство реализаций предопределено сходством условий при постановке задач управления. Приведем типовые условия задачи.

Будем рассматривать процесс как систему управления (СУ) с несколькими (MAX_ITEMS) объектами управления. Объект управления – единица оборудования (все единицы однотипны). СУ может управлять также отдельными элементами объектов управления. Примеры: СУ ТРК управляет сторонами топливно-раздаточных колонок и рукавами налива топлива, СУ уровнемера управляет уровнемером и подключенными к нему резервуарами.

Объекты управления подключены к портам UART. К одному порту через общий контроллер по шине может быть подключено несколько объектов. У каждого объекта управления (или его управляемого элемента) есть свой уникальный адрес в контроллере.

На модуле несколько (MAX_PORTS) портов UART. Только часть из них (NUMBER_OF_EXCHANGE_CHANNELS) используется для подключения объектов управления.

Протокол управления определяет состав, последовательность и формат запросов (команд) от СУ к объекту, а также состав и формат допустимых ответов на каждый запрос.

Формат сообщений, передаваемых по последовательному каналу, специфицирует маркеры начала и конца сообщения. В составе сообщения передается контрольная сумма. Ответ объекта состоит из короткого сообщения, за которым могут передаваться данные. Короткое сообщение подтверждает или опровергает успешность приема команды СУ. Данные ответа передаются только в случае успешного приема запроса.

Протокол определяет максимальное время задержки ответа оборудования, а также предельно допустимое число повторных посылок запроса.

Задачи СУ различаются для разных типов оборудования.

СУ уровнемера должна выполнять мониторинг состояния резервуаров с сохранением текущего состояния.

СУ ТРК должна управлять наливом топлива по рукавам ТРК, подключенных к модулю.

СУ стелы должна обеспечивать отображение актуальных значений цены видов топлива.

Мы будем рассматривать обобщенную постановку задачи, в соответствии с которой СУ должна:

- отправлять последовательность запросов объектам управления по сценарию, определенному протоколом,

- принимать и интерпретировать ответы оборудования,

- извлекать из ответов и сохранять данные о текущем состоянии объектов управления.

Данные для запросов и текущее состояние объектов управления должны храниться в статической памяти процесса. Другим процессам должна быть обеспечена возможность изменения и опроса перечисленных данных.

6.2. События процесса

В текущей версии программы процессы обрабатывают следующие события:

OILIX_EVENT_RESET,
OILIX_EVENT_TIMER,
OILIX_EVENT_UART_RX.

Событие OILIX_EVENT_RESET обрабатывается всеми процессами. По этому событию процесс конфигурирует и инициализирует свои структуры.

Событие OILIX_EVENT_TIMER обрабатывается только в том случае, если оно порождено таймером процесса. С каждым процессом связан таймер, отмеряющий тайм-аут CONTROL_TIMEOUT. С процессом ТРК связаны два таймера, отмеряющие тайм-ауты: STOP_TIMEOUT, CONTROL_TIMEOUT. Таймеры всех процессов пронумерованы сквозным образом. Для маркировки каждого таймера отведен свой бит в поле user_data структуры события. Индексы битов и величины таймаутов для каждого процесса устанавливаются при инициализации. Таймер «STOP_TIMEOUT» тактирует сценарий перевода объекта управления в состояние OFF_LINE. Таймер «CONTROL_TIMEOUT» тактирует рабочий цикл процесса.

Событие OILIX_EVENT_UART_RX обрабатывается процессом branch_idc_process() и процессом уровнемера. Событие возникает при появлении новых данных во входном буфере порта UART. Номер порта передается в поле user_data структуры события.

6.3. Структуры данных процесса

Процесс использует следующие типы структур данных:

- конфигурационные данные (см. раздел 4 настоящей статьи);

- программное представление объектов управления (единиц оборудования) в их текущем состоянии (UNITS, далее - «юниты»);

- каналы обмена данными по портам UART (EXCHANGE_CHANNELS, далее – «каналы управления»).

Перечисленные данные хранятся в виде массивов структур в статической памяти процесса. Доступ к конфигурационным данным и к юнитам предоставлен другим процессам через интерфейсные функции.

Тип структуры юнита описывает тип оборудования и определяется индивидуально для каждого типа. Тип структуры канала управления зависит от вида оборудования (т. е. от протокола управления) и определяется индивидуально для каждого протокола.

Примеры:

1) Юниты процесса уровнемера:

```
typedef struct _LMSInfo {
      uint8_t tank;
                              …} LMSInfo;
static LMSInfo \
lms_unit[MAX4OF_TANKS];
```

2) Юниты процесса ТРК:

```
typedef  struct _TRKInfo{
uint8_t   trk;
                           …} TRKInfo;
static TRKInfo \
lms_unit[MAX4OF_TRK];
```

3)Канал управления для уровнемера «Струна»:

```
typedef struct _StrunaExchangeInfo
                                 {…….
                                  } StrunaExchangeInfo;
static StrunaExchangeInfo strunaExchangeInfo
[LMS4OF_EXCHANGE_\
CHANNELS];
```

Упрощенное изображение связей между юнитами, каналами управления, портами UART и управляемыми объектами приведено на рисунке 5.

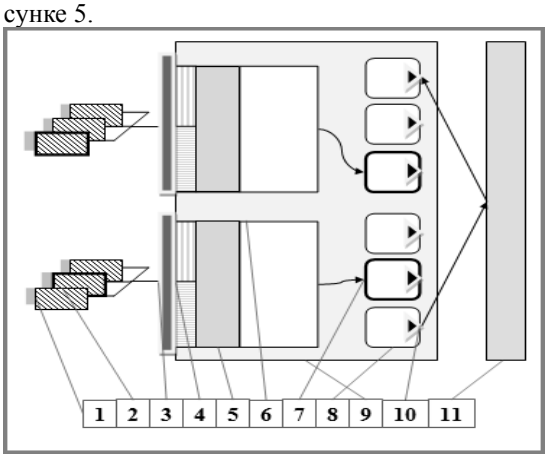

Рис. 5. Управление технологическим оборудованием. Структуры данных процесса. Обозначения: 1 – единицы оборудования, 2 –активный объект управления, 3 – порт UART, 4 – буферы UART (rx, tx), 5 – буфер данных канала, 6 – канал управления, 7 – активный юнит в канале управления, 8 – юниты, 9 – процесс управления, 10 – индикаторы изменения состояния юнита, 11 – branch_idc_process()

Процесс инициализирует массивы юнитов и каналов управления на основе конфигурационных данных. В массиве юнитов регистрируются только те единицы оборудования, которые должны быть подключены к модулю. Индекс юнита в массиве – порядковый номер единицы оборудования на модуле. В массиве каналов управления регистрируются только те порты, к которым подключается оборудование, обслуживаемое процессом. Такие порты опознаются по полю application в конфигурации порта.

Содержимое юнитов и каналов управления изменяется по мере работы программы и отражает текущее состояние объектов управления. Корректирует юниты сам процесс, а также процесс branch-idc_process(). Процесс branch-idc_process() обеспечивает взаимодействие сервера АЗС с процессом управления оборудованием. Процессы изменяют и вычитывают содержимое юнитов асинхронно.

Поля юнита:

- идентифицируют объект управления;
- хранят текущие значения параметров состояние объекта;
- позволяют узнать об изменении состояния объекта.

Различия в структуре каналов управления для уровнемера и для ТРК существенны. Приведем обобщенное описание информационного наполнения канала.

Канал управления должен хранить, идентифицировать или ссылаться на следующие сущности:

- номер порта UART,
- конфигурационные данные порта UART,
- активный объект управления,
- состояние канала (состояние активного объекта),
- код текущего запроса,
- входной и выходной буферы порта UART,
- буфер хранения ответа оборудования,
- состояние приема данных из порта UART,
- счетчик ошибок.

Под активным объектом управления понимается единица оборудования, с которой взаимодействует канал управления в текущий момент. Пока не получен ответ на текущий запрос, процесс не может отправлять запросы другим объектам по тому же каналу управления. Канал управления может содержать ссылку на юнит или использовать другой способ идентификации активного объекта.

При входе в очередную итерацию рабочего цикла канал находится в одном из двух состояний: «не активен», «ожидает ответа».

Под текущим запросом понимается код последнего выполненного запроса по протоколу. В протоколе управления каждой команде присвоен код. По сценарию управления очередной запрос формируется с учетом предыдущего.

Под состоянием приема данных понимается состояние функции побайтового чтения входного буфера порта UART, используемой для приема ответа оборудования. Функция работает по принципу конечного автомата. Примеры состояний, в которых может остаться автомат по завершении текущей итерации процесса: «ожидание подтверждения», «ожидание старт-символа», «ожидание стоп-символа» (точный перечень состояний специфицирован протоколом управления).

6.4. Обмен данными по каналу управления

Выбор канала управления зависит от того, по какому событию запустилась очередная итерация рабочего цикла процесса.

Если итерация инициирована таймером (событие OILIX_EVENT_TIMER), то в ней последовательно обрабатываются все каналы, обслуживаемые процессом. Каналы опознаются по значению поля application в конфигурации порта. Если в канале нет активного объекта, то процесс отправляет в него запрос с поиском активного объекта, после чего переходит к следующему каналу.

Если итерация инициирована событием OILIX_EVENT_UART_RX, то она обрабатывает только тот канал, по которому пришли данные. Канал выбирается по номеру порта, указанному в поле user_data структуры события. Активен тот объект, от которого пришли данные.

Далее считаем, что канал и объект выбраны. Дальнейшая обработка состоит из следующих шагов.

Шаг 1. Прием ответа объекта управления. Если ответ получен, то переход к шагу 2, иначе – переход к шагу 4.

Ответ считается полученным, если в результате побайтового чтения входного потока данных в буфере ответа оборудования накопился пакет, который соответствует формату положительного ответа по протоколу управления.

Шаг 2. Интерпретация ответа.

Ответ интерпретируется в соответствии с протоколом по коду текущего запроса. Изменение состояния объекта фиксируется в юните.

Шаг 3. Формирование следующего запроса по протоколу. Переход к шагу 6.

Шаг 4. Обработка счетчика ошибок.

Если исчерпано число допустимых ошибок, или завершена последовательность запросов по протоколу, объект сделать неактивным.

Шаг 5. Отправка следующего запроса.

Отправка следующего запроса предполагает, что если текущий объект неактивен, нужно перейти к следующему объекту в том же канале.

Шаг 6. Завершение работы с каналом в текущей итерации.

7. Связь с сервером АЗС по протоколу IDC

7.1. Роль сервера АЗС в управлении технологическим оборудованием

Программа интерфейсного модуля управляет технологическим оборудованием независимо от того, подключен ли модуль к серверу АЗС. Мониторинг состояния оборудования может осуществляться в автономном режиме. Участие сервера в управлении необходимо в следующих случаях:

- для передачи команд оператора,
- для мониторинга состояния оборудования с сервера в автоматическом режиме.

Команды оператора необходимы для конфигурирования оборудования и для управления технологическими процессами в ручном режиме. Пример: санкционирование налива топлива после оплаты.

Команды передаются в интерфейсный модуль с сервера АЗС. Сервер АЗС взаимодействует с интерфейсными модулями по протоколу IDC.

7.2. Протокол IDC. Основные понятия и определения

Протокол IDC определяет правила передачи сообщений в древовидной, разнородной по типу соединений сети ПЛК (далее – «IDC-сеть»). IDC-сеть предназначена для организации управления технологическим оборудованием с сервера АЗС.

ПЛК рассматривается как узел древовидной сети (далее – «IDC-узел»). Север АЗС выполняет роль корневого узла. Каждому узлу соответствует уникальный числовой идентификатор (далее – «IDC-адрес»).

Узлы подразделяются на управляемые и управляющие. Управляющий узел передает команды управляемым узлам и принимает ответы. Команды и ответы передаются в «IDC-формате» (формат описан ниже).

Узел обладает набором «IDC-интерфейсов». Под IDC-интерфейсом понимается набор IDC-команд, предназначенных для выполнения узлом какой-либо задачи. IDC-интерфейс идентифицируется уникальным номером в рамках сети. Разные узлы могут обладать одним и тем же программным интерфейсом. Каждой команде программного интерфейса соответствует уникальный в рамках этого интерфейса номер.

IDC-формат сообщения определен для канального, транспортного и прикладного уровней модели передачи данных.

Формат кадра для канального уровня: [«Старт» «Данные» «Длина» «Контрольная сумма» «Стоп»].

Все данные между символами «Старт» и «Стоп» не должны совпадать с этими символами.

Формат пакетов транспортного уровня: [«адрес узла отправителя» «адрес узла получателя» «идентификатор интерфейса» «команда интерфейса»].

Формат команды программного интерфейса (прикладной уровень): [«номер команды»][«данные команды»].

«Данные команды» – это байтовый массив. Интерпретирует данные программный интерфейс в соответствии со специфичным для него протоколом.

7.3 Процесс branch_idc_control _process()

На интерфейсном модуле прием и отправку пакетов в формате IDC выполняет процесс branch_idc_control_process() (далее – branch_idc). Для связи по IDC используется порт UART, в конфигурации которого в поле «application» указано значение UART_APPLICATION_IDC_BRANCH.

В программе интерфейсного модуля реализованы IDC-интерфейсы к поддерживаемым типам технологического оборудовании, а также к двум программным сервисам: dm (менеджер данных), azs-config (сервис конфигурирования).

Для описания IDC-интерфейса предназначен следующий тип данных:

```
typedef struct _IDCInterface {uint8_t          id;
FuncIdcGetPendingRequest    pending_request;
FuncIdcHandleData               handle_data;
                              } IDCInterface;
```

Поля структуры:

- id – номер интерфейса;
- pending_request – функция поиска готового ответа;
- handle_data – функция-обработчик IDC-пакета.

Формат функции – handle_data:

```
typedef void (*FuncIdcHandleData)(uint8_t
source, const uint8_t* buffer, uint8_t size);
```

Формат функции pending_request:

```
typedef struct _IDCReply {
    uint8_t    packet_size;
    uint8_t packet[IDC_MAX_REPLY_LENGTH];
                              } IDCReply;
#define    IDCRequest          IDCReply

typedef IDCRequest* (*FuncIdcGetPendingRe-
quest)(void);
```

В статической памяти определен список интерфейсов:

```
static IDCInterface* idc_interface_list[] =
{
 &idc_dm_interface,
 &idc_lms_control_interface,
 &idc_trk_control_interface,
 &idc_azs_config_interface,
 &idc_priceboard_interface,
};
```

Каждый интерфейс реализован в своем программном модуле. Модули интерфейсов включаются в программный модуль процесса branch_idc:

```
#include <idc2/idc-common.h>
#include <idc2/idc-dm.c.h>
#include <idc2/idc-azs-config.c.h>
#include <idc2/idc-lms-control.c.h>
#include <idc2/idc-trk-control.c.h>
#include <idc2/idc- priceboard.control.c.h>
```

Для приема запросов и формирования ответов используется буфер в статической памяти (IDC-буфер). Для буфера определены смещения в соответствии с форматом IDC-пакета:

```
typedef enum _IDCOffset
{
IDC_OFFSET_INTERFACE,
IDC_OFFSET_SOURCE,
IDC_OFFSET_DESTINATION,
IDC_OFFSET_COMMAND,
IDC_OFFSET_DATA,
}
```

Процесс branch_idc обрабатывает события OILIX_EVENT_UART_RX, возбуждаемые по приходе очередной порции данных в UART-порт процесса. Запускается функция побайтового чтения входного буфера порта. Функция вычленяет из входного потока кадры канального уровня протокола IDC, извлекает пакеты транспортного уровня, помещая их в IDC-буфер. Функция работает по принципу конечного автомата. Если по исчерпании входных данных порта в IDC-буфере не собрался корректный пакет, или адрес узла получателя не совпал с адресом модуля, итерация процесса branch_idc завершается. В противном случае вызывается функция интерпретации пакета. Функция интерпретации вычитывает из IDC-буфера номер интерфейса и запускает соответствующий обработчик сообщений handle_data из списка интерфейсов.

Обработчик handle_data вычитывает из IDC-сообщения команду и данные и выполняет команду в соответствии с ее семантикой. Под выполнением команды понимается внесение изменений в юниты или в конфигурационные данные процессов управления оборудованием. Изменения вносятся путем вызова интерфейсных функции процессов.

До объекта управления команда дойдет асинхронно. Соответствующая ей последовательность команд будет выработана процессом управления оборудованием. Команды будут формироваться в соответствии с протоколом управления с учетом измененных данных юнита. Сопутствующие изменения состояния объекта управления будут зафиксированы в юните.

По завершении обработчика handle_data процесс branch_idc формирует и отправляет ответ источнику IDC-запроса (т. е. серверу АЗС). В ответе отправляются сведения об очередном изменении состояния технологического оборудования. Ответ формируется следующим образом.

Процесс branch_idc находит в списке интерфейсов очередной интерфейс, по которому произошли изменения. Об изменениях сообщает функция pending_request интерфейса. Функция pending_request ищет очередной измененный элемент в массиве юнитов для своего типа оборудования. В случае успеха функция формирует ответ. В ответе передаются параметры измененного состояния. Если ни один интерфейс не сообщил об изменениях, в ответе отправляется сообщение «NO_DATA».

8. Сохранение состояния в энергонезависимой памяти

Программа интерфейсного модуля восстанавливает состояние ПЛК после перезагрузки. Это достигается следующим образом.

Состояние ПЛК описывают посредством «retain-переменных». Понятие «retain-переменные» используется для обозначения переменных, которые сохраняют значение при отключении питания контроллера. Между сеансами работы контроллера retain-переменные сохраняются в энергонезависимой памяти.

В программе интерфейсного модуля retain-переменные объявляются с атрибутом (section (”mram”):

__attribute__ ((section (”mram”), aligned (32)))

Компилятор размещает переменные, объявленные с этим атрибутом, в секции «mram» и помещает адреса начала и конца секции в переменные

unsigned char __start_mram;

unsigned char __stop_mram;

С атрибутом (section (”mram”)) объявляются юниты и конфигурационные данные процессов.

Для сохранения retain-переменных предназначена энергонезависимая память EEPROM объемом 1 Мбайт.

При падении питания ниже критического

уровня программа сохраняет секцию «mram» в энергонезависимой памяти.

По включении ПЛК программа восстанавливает содержимое секции из энергонезависимой памяти.

9. Портирование программы на аппаратную платформу ИМ-АЗС

Особенности аппаратной платформы модуля ИМ_АЗС привели к необходимости внесения следующих изменений в текст программы.

В связи с отсутствием операционной системы для процессора КОМДИВ-МКЭ функции технологической обвязки, обеспечивающие доступ к базовым интерфейсам, включены в текст программы.

Разработана и включена в текст программы библиотека функций асинхронного обмена данными по интерфейсу UART в соответствии со стандартом POSIX.

В связи с малым объем ОЗУ (128 КБ) программный код выполняется из РПЗУ модуля. Для достижения требуемого быстродействия размещение секций кода оптимизировано за счет размещения обработчиков событий в ОЗУ.

В связи с отсутствием в процессоре КОМДИВ-МКЭ сопроцессора плавающей арифметики для выполнения операций с плавающей точкой используется библиотека (GNU) SoftFloat. Библиотека SoftFloat подключается при компиляции программы [2] с использованием компилятора gcc.

Диагностика падения питания осуществляется путем опроса датчика питания.

10. Заключение

Проведен анализ текста проприетарной программы управления технологическим оборудованием АЗС. Выявлены и описаны архитектура, базовые примитивы и принцип функционирования программы.

Выполнено портирование программы на модули семейства ИМ-АЗС. В статье приведен перечень сопутствующих доработок.

Замечена схожесть подходов в реализации процессов управления разными типами оборудования. Это создает предпосылки для последующей оптимизации программы на основе объектного или автоматного программирования [1].

Публикация выполнена в рамках государственного задания по проведению фундаментальных исследований по теме «Исследование и реализация программной платформы для перспективных многоядерных процессоров» (FNEF-2022-002).

Control System Software and Hardware Architecture Import Substitution on Limited-Performance Computing Facilities

T.K. Gringauz, A.N. Onin

Abstract. The microcontroller KOMDIV-MKE of domestic design is part of the interface modules of the IM-AZS family. The modules are designed for use as technological process and equipment controllers of a gas filling station. A proprietary control program has been ported to the modules. The program operation principle is described. A list of its improvements during porting is given.

Keywords: import substitution, interface module, programmable logic controller, tree network, technological equipment, filling station, level gauge, fuel dispenser

Литература

1. Н.И. Поликарпова, А.А.Шалыто. Автоматное программирование. Санкт-Петербург, Санкт-Петербургский Государственный Университет Информационных Технологий Механики и Оптики, 2008.

2. В.А.Галатенко, Г.Л.Левченкова, С.В.Самборский. Особенности сборки кросс-компилятора GCC и бинарных утилит. «Труды НИИСИ РАН», Т. 12 (2022), № 4, с. 43-49.