ООО «3В Сервис»
РФ, 127051, Москва, ул. Трубная 25 стр 1 офис 6
Тел./ф (495) 221-22-53
www.3v-services.com

Утверждаю
генеральный директор
ООО «3В Сервис»

_____ Петухов В.Н.

Среда динамического моделирования технических систем SimInTech™

РУКОВОДСТВО ПРОГРАММИСТА

Система программирования для вычислительных приборов

ШИФР ГК16РП

Москва, 2016

СОДЕРЖАНИЕ

1 СТРУКТУРА И ОБЩИЙ АЛГОРИТМ ФУНКЦИОНИРОВАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ПРИБОРА

Программное обеспечение прибора работает под управлением операционных систем реального времени (ОСРВ) QNX Neutrino и КПДА.00002-01. ПО обеспечивает выполнение алгоритма на приборе с заданным временным тактом, получения удалённого доступа к данным прибора, управления программного обеспечения (ПО) прибора.

ПО состоит из следующих модулей:

- диспетчер расчётных модулей (процессов) DispExemod– обеспечивает автоматическое создание общей областей памяти для внешних переменных расчётных модулей (общая область описывающая массив структур внешних переменных и общая область памяти, хранящая значения внешних переменных), их загрузку, загрузку начального состояния прибора.
- сервер отладки GdbServer – обеспечивает доступ с клиентского рабочего места (РМ) к переменным прибора, а также управление (пауза, продолжение, завершение) работы ПО прибора.
- расчётные модули – обеспечивают обработку переменных прибора согласно алгоритму, заложенному в исходной расчётной схеме. Исполняемый код расчетных модулей собирается на основе Си-кода, сгенерированного генератором кода программного комплекса (ПК) МВТУ.

ПО прибора может функционировать в двух режимах:

- штатном;
- отладочном.

Отличия этих режимов состоят в том, что в отладочном режиме можно получить удалённый доступ к переменным и управлять расчётом через оболочку SimInTech (ПК МВТУ-4). В штатном режиме удалённое управление и доступ к переменным невозможен.

К режиму отладки относится код только сервера обмена данными GdbServer. В штатном режиме вы можете его не копировать (или удалить с прибора). Всё остальное остаётся без изменений. В обязательном порядке необходимы кроме исполняемых файлов ещё и файлы описания внешних переменных расчётных модулей *.extvars.table и файл конфигурации загрузки default.conf, потому что по ним формируется рабочая область памяти при загрузке диспетчера расчётных модулей. Остальное (GdbServer и файлы *.intvars.table) -

можно удалить, если этих компонентов нет на приборе, то доступ извне к константам и состояниям невозможен. Для запуска ПО в штатном режиме расчётные модули и диспетчер пересобирать не нужно.

Для того чтобы ничего не выводилось в консоль, необходимо запустисть диспетчер расчётных модулей с выводом в нулевое устройство: DispExemod > /dev/null или же DispExemod > / dev / nul 2>&1 &

Алгоритм функционирования ПО прибора представлен на рисунке 1.1.

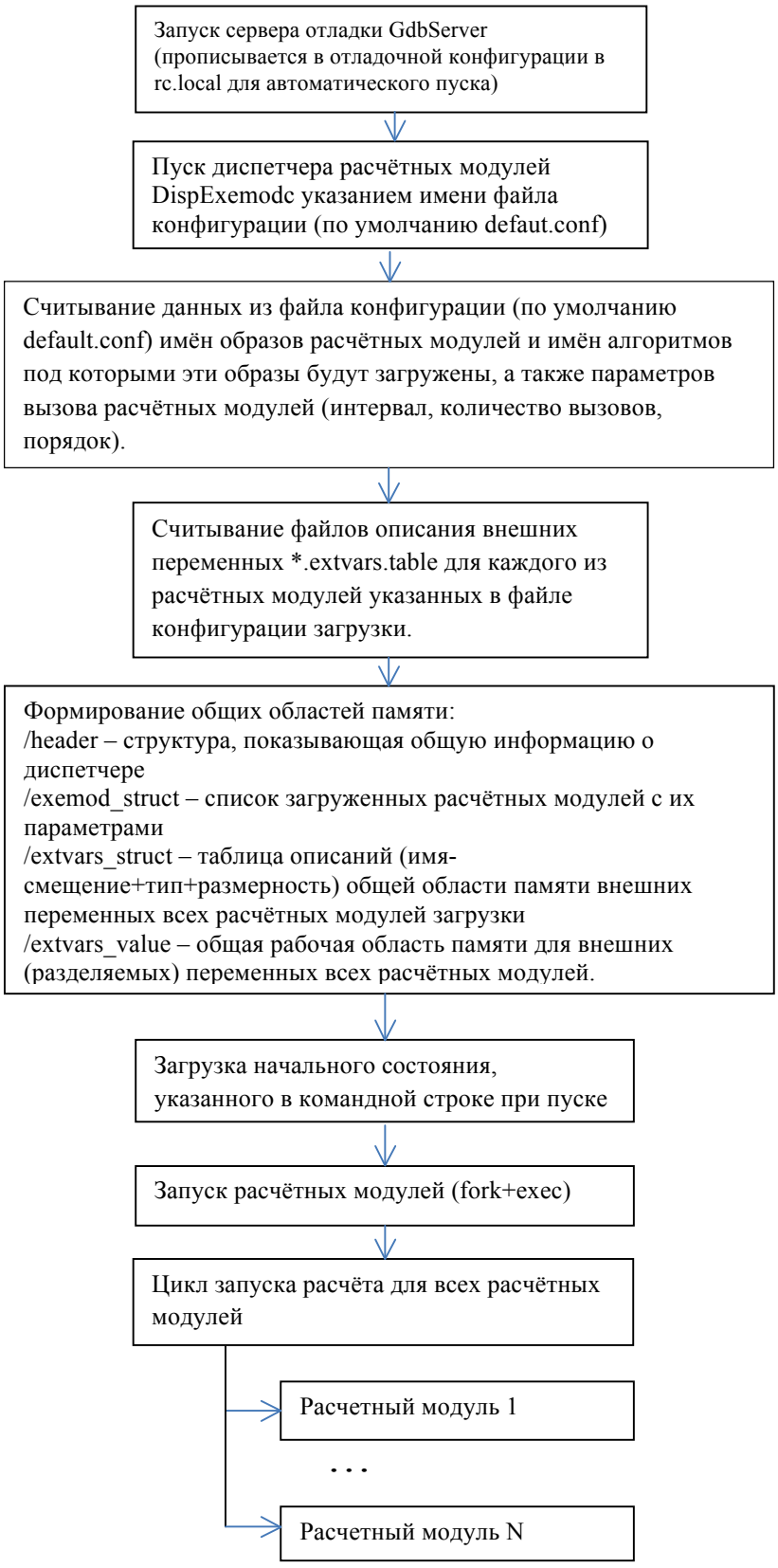


Рисунок 1.1

Структура и взаимосвязи ПО прибора представлена на рисунке 1.2.

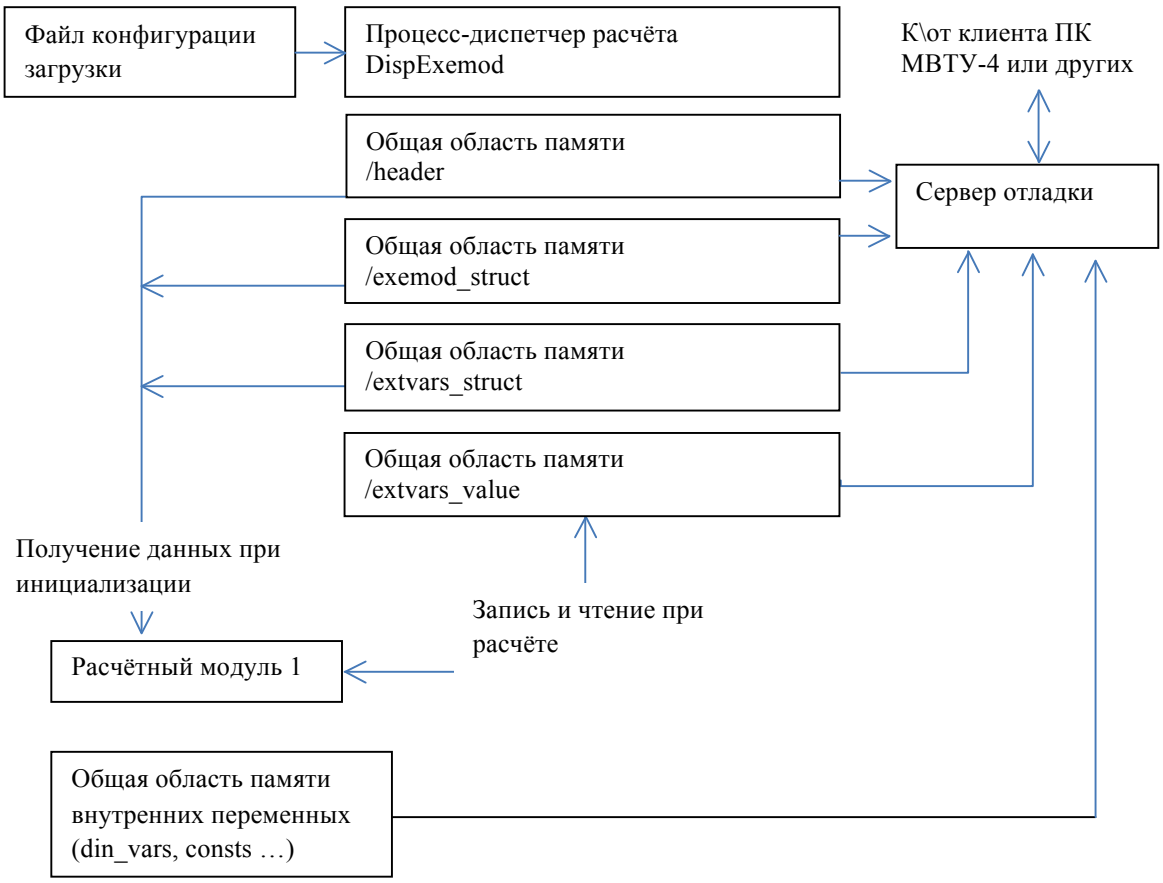


Рисунок 1.2

2 ОПИСАНИЕ ФОРМАТОВ ФАЙЛОВ СТРУКТУР И ПРОТОКОЛОВ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ПРИБОРА

2.1 ФОРМАТ ФАЙЛОВ ОПИСАНИЯ ПЕРЕМЕННЫХ РАСЧЁТНЫХ МОДУЛЕЙ

Каждый из расчётных модулей при генерации кода включает в себя следующие файлы:

- Исполняемый файл расчётного модуля (образ);
- Файл описания внешних переменных расчётного модуля (<имя образа>.extvars.table);
- Файл описания внутренних переменных расчётного модуля(<имя образа>.intvars.table);

Исполняемый файл содержит непосредственно исполняемый код, скомпилированный компилятором из исходных текстов, сгенерированных генератором кода ПК МВТУ-4.

Все файлы приборного ПО, включая исполняемые файлы расчётных модулей, диспетчера расчётных модулей DispExemod, сервера обмена данными GdbServer и файлов описания конфигурации и переменных должны быть размещены в файловой системе прибора в одной директории.

Файл описания внешних переменных содержит таблицу, с информацией о типах данных, размерностях и именах переменных, которые расчётный модуль должен получить из общей для всех РМ в загрузке области памяти (рабочей области памяти внешних переменных). Т.е. диспетчер при загрузке РМ прочитывает файлы описания внешних переменных для всех РМ и выделяет единую область памяти под все внешние переменные. При этом переменные для разных РМ , имеющие одинаковое имя, объединяются (т.е. на них выделяется одна и та же область памяти в рабочей памяти). При несовпадении типов – выдаётся диагностическое сообщение в консоль и происходит аварийное прекращение работы диспетчера. После того как диспетчер выделил общую область памяти для всех внешних переменных РМ он сообщает расчётным модулям с каким именно смещением лежат нужные переменные для каждого из модулей, данная информация располагается в общей области памяти с именем /extvars_struct.

Файлы описания внутренних переменных такие же по формату, как и файлы описания внешних переменных. Они необходимы только для того чтобы сервер отладки мог

получить по имени нужный адрес для доступа к внутренней переменной конкретного запущенного экземпляра расчётного модуля. Области памяти под внутренние переменные выделяются не диспетчером, а самим РМ в соответствии с типом конкретной переменной и именем алгоритма, указанным при запуске РМ.

Файлы описания внешних и внутренних переменных представляют собой бинарные файлы и состоят из записей типа:

```
typedefstruct {
unsigned char  sizeofname;
char           name[64];
unsigned char  sizeofdecsription;
chardecsription[128];
unsigned char  data_type;
unsigned long  dims[3];
unsigned char  direction;
unsigned char  var_type;
unsigned long  index;
unsigned short hash;
} ext_var_info_record;
```

Рисунок 2.1.1

На рисунке 2.1.1 представлены следующие элементы:

sizeofname - длина имени переменной;

name - массив символов, содержащий имя переменной в кодировке ASCII (допустимая длина – не более 64 символа);

sizeofdecsription - длина текстового описания переменной;

decsription - массив символов, содержащий описание переменной в кодировке ASCII (допустимая длина – не более 128 символа);

data_type - идентификатор типа данных переменной.

Допустимые значения:

0 – вещественный;

1 – двоичный;

2 – целый.

dims - массив из трех элементов целого типа, показывающий размерности переменной. Например, если значение dims равно {10,0,0}, то переменная является вектором заданного типа с количеством элементов равным 10;

direction - идентификатор направления переменной.

Допустимые значения:

0 – вход;

1 – выход;

2 – выход-выход (т.е. сигнал и читается и пишется в модели).

var_type	идентификатор класса переменной, от которого зависит в каком массиве/области памяти находится переменная. Допустимые значения: 0 – Внешняя, это означает, что данная переменная выделяется в памяти диспетчером и делится данным расчётным модулем с другими модулями. К этому классу переменных относятся входы и выходы модели. Отличается тем что в заголовочном файле для этих переменных используется другая адресация (с присваиваемым смещением). Сохраняется в рестарте (то есть, когда пользователь сохранят состояние исполняемой среды на приборе, значения переменных в данной области памяти записываются в файл на диске с именем соответствующим имени сохраняемого состояния). Располагаются в /extvars_value; 1 – Динамическая – переменные состояния, которые интегрируются в едином массиве при помощи выбранного метода интегрирования. Данные переменные не расшариваются между отдельными расчётными модулями. Для каждой из таких переменных существуют 2 массива – собственно переменная состояния и её производная. Используются только для интеграторов и динамических блоков. Сохраняется в рестарте. Располагаются в области памяти /din_vars_<имя алгоритма указанное при запуске> ; 2 – Алгебраическая – переменные состояния, которые содержат значения алгебраических функций и состояний, в данной версии исполняемой среды не используются. Данные переменные не расшариваются между отдельными расчётными модулями. Для каждой из таких переменных существуют 2 массива – собственно переменная состояния и значений алгебраической функции. Используются только для блоков типа Y=F(0) и Y =F(Y). Сохраняется в рестарте. Располагаются в области памяти /alg_vars_<имя алгоритма указанное при запуске> ; 3 – Внутреннее состояние – выделяются для каждого из

index - байтовое смещение переменной в массиве (какой массив определяется согласно var_type). Для внешних переменных байтовое смещение определяется диспетчером при загрузке РМ, и затем передаётся в область памяти /extvars_struct, в файле описания внешних переменных на диске (*.extvars.table)в качестве индекса передаётся номер внешней переменной в массиве указателей внутри расчётного модуля;

hash - хэш имени переменной вычисленный по алгоритму CRC16.

2.2 ФОРМАТ ФАЙЛА ОПИСАНИЯ ЗАГРУЗКИ

Файл конфигурации загрузки представляет собой текстовый файл в формате ASCII с разделителями строк символами с кодами 0D или 0D0A. Данный файл сообщает диспетчеру РМ информацию о том, какие именно расчётные модули необходимо загрузить, под какими именами алгоритмов и с каким интервалом их необходимо вызывать в процессе работы прибора.

Файл представляет собой таблицу, где на каждой стоке описывается способ вызова расчётного модуля. Строки имеют следующий формат:

<имя исполняемого файла расчётного модуля> <имя алгоритма> <интервал вызова в мсек> <к-во вызовов (повторов) за один интервал вызова данного расчётного модуля>

….. и так на остальных строках. Разделитель слов в строке – пробел.

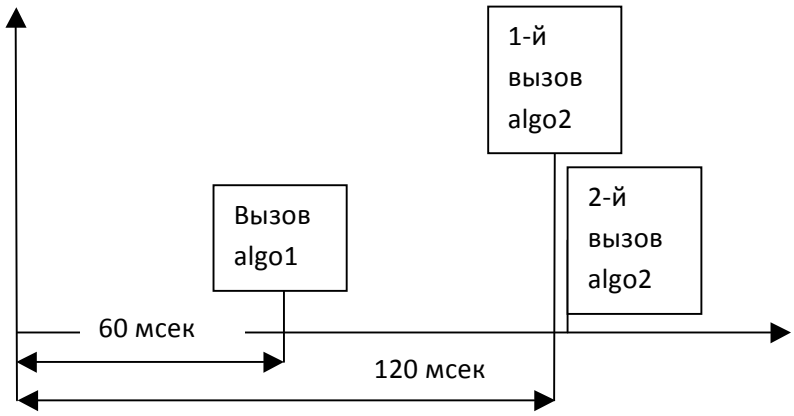
Пример текста в файле представлен на рисунке 2.2.1.

```
calc algo1 60 1
calc algo2 120 2
```

Рисунок 2.2.1

Это означает что первым выполняется расчётный модуль загружаемый из исполняемого файла calc с именем алгоритма algo1, который вызывается 1 раз за 60, а вторым выполняется расчётный модуль загружаемый из исполняемого файла calc с именем алгоритма algo2, который вызывается 2 раза за 120 мсек. Примечание: такты (периоды) должны быть кратными (то есть такты выполнения должны быть в целое количество раз больше или равно минимальному среди указанных тактов алгоритмов) ! Минимальный временной интервал определяется автоматически исходя из файла конфигурации загрузки

как минимум по всем заданным тактам. Такт не может быть задан меньше 1 мсек. Исходный код функции takt_work, где находится цикл вызовов приведён ниже. На рисунке 2.2.2 приведён график работы в соответствии с указанной выше конфигурацией:



Процессы в данной версии исполняемой среды выполняются последовательно, в соответствии с тем как они описаны в файле конфигурации загрузки. При этом если у процессов установлен разный такт выполнения, то общий такт будет равен минимальному из заданных, но алгоритм у которого период выполнения задан больше будет выполняться не на каждом шаге. Т.е например для приведённого примера algo1 будет выполняться на каждом шаге, а algo2 – через шаг. Время, выводимое для отладчика (см. описание сетевого протокола GdbServer) равно: (минимальный из заданных шагов)*(к-во циклов диспетчера). При вызовах run-функции в расчётном модуле время равно 0, поскольку в явном виде нигде там не используется.

Такты выполнения расчётных модулей являются задаваемой и постоянной величиной. Для более подробной информации – см. исходный код DispExemod\main.c, функция takt_work. Ниже приведён фрагмент данной функции с кодом расчёта условного модельного времени:

```
void takt_work(void)
{
     int count = 0;
     int k = 0;

     uint64_t cycle1 = 0;
     uint64_t cycle2 = 0;
     uint64_t ncycles = 0;
     uint64_t cps = 0;

     struct timespec req = { 0 };

     double delta_scan = 0.0;
```

```
		double time_sleep = 0.0;

		char cmd[1] = {0};
		char rep[1] = {0};

		/*Цикл посылки сообщений для тестирования*/
		while (1)
		{

			/*Количество циклов процессора до начала обработки бд*/
			cycle1 = ClockCycles( );

			/*Будем запускать на выполнение расчетные модули время
			 * которых пришло
			 */
			for (count = 0; count < ptr_header->number_exemod;
count++)
			{
				/*Текущий счетчик в 0 значит время пришло*/
				if (ptr_exemod[count].tek_time == 0)
				{
					/*Отправим на выполнение расчетный модуль
стокько
					 * сколько он должен выполняться за один
такт
					 */
					for (k = 0; k < ptr_exemod[count].num_work;
k++)
					{
						cmd[0] = 0x01;
						MsgSend(ptr_exemod[count].coid,    cmd,
sizeof(cmd), rep, sizeof(rep));

					}     /*for     (k     =     0;     k     <
ptr_exemod[count].num_work; k++)*/

					/*Восстановим текущее время*/
					ptr_exemod[count].tek_time                    =
ptr_exemod[count].takt_mod;

				} /*if (ptr_exemod[count].tek_time == 0)*/

			} /*for (count = 0; count < ptr_header->number_exemod;
count++)*/

			/*Количество циклов процессора после обработки бд*/
			cycle2 = ClockCycles( );

			/*Количество циклов ушедшее на обработку бд*/
			ncycles = cycle2 - cycle1;

			/*Сколько циклов в секунде*/
			cps = SYSPAGE_ENTRY(qtime)->cycles_per_sec;

			/*Время затраченное на обработку бд*/
			delta_scan = (1000.0 * ((double) ncycles / cps));

			/*Время сна, с компенсацией времени на предыдущем шаге*/
			time_sleep = (double) ptr_header->takt - delta_scan;
```

```
                    /*Спим оставшееся время до начала следующего такта*/
                    nsec2timespec(&req,   (uint64_t)   (      time_sleep   *
1000000L));

                    /*Спим до начала следующего такта*/
                    if (nanosleep(&req, NULL) == -1)
                    {
                          perror("nanosleep");
                    } /*nanosleep.....*/

                    /*Еще один такт прошел уменьшим время ожидания запуска
                     * расчетных модулей
                     */
                    sheduler_takt();

                    //Это счётчик своих тактов синхронизатора, по нему
считаем время ptr_header – это главная общая область памяти диспетчера
/header
                    ptr_header->takt_counter = ptr_header->takt_counter + 1;

              } /*while (1)*/


 }
```

Далее при выводе времени на клиент (то есть в графическую оболочку) используется следующий код (GdbServer.c функция packet_send):

```
        int packet_send(void)
        {
             header_packet header = {0};
             signal_addr var_signal_addr;
             struct queue_  *pkt = NULL;
             int sum = 0;
             int nbytes = 0;

             double f = 0;

             unsigned char *ptr_buf_packet = buf_packet + sizeof(header);

        //Это код подготовки значения модельного времени прибора
        //для вывода его на схеме в графической оболочке в режиме отладки

        /*Шаг интегрирования - в секундах !!!*/
             header.fStep = ptr_header->takt*0.001;

        //Это собственно время, выводимое в оболочке
             header.time_connect = ptr_header->takt_counter*header.fStep;
        ……
```

2.3 ФОРМАТ ОБЩИХ ОБЛАСТЕЙ ПАМЯТИ ПО ПРИБОРА

При старте диспетчера расчётных модулей он выделяет в системе под свои нужды несколько областей памяти.

Область памяти /header описывается структурой, представленной на рисунке 2.3.1.

```
typedefstruct
{
      charname_disp[STR_LEN];  /* Имя диспетчера*/
      int   takt;              /* Базовай такт работы, мсек */
      int number_exemod;       /* Количество расчетных модулей*/
      int num;                 /* Количество внешних сигналов */
      int signals_size;        /* Количество байт занимаемое сигналами*/
      int pid;                 /* Пид. диспетчера*/
      int takt_counter;        /* Локальный счётчик тактов диспетчера */
} header;
```

Рисунок 2.3.1

На рисунке 2.3.1 представлены следующие элементы:

name_disp	-	имя файла конфигурации загрузки, указанное при старте диспетчера;
takt	-	базовый (минимальный из всех заданных в файле конфигурации загрузки) такт работы в мсек, используется для расчёта текущего времени при отладке;
number_exemod	-	количество загруженных расчетных модулей (алгоритмов);
num	-	общее количество внешних сигналов (равно сумме всех к-в внешних сигналов для всех РМ), используется для поиска по области памяти /extvars_struct;
signals_size	-	количество памяти в байтах, выделенное под все внешние сигналы при запуске диспетчера;
pid	-	идентификатор процесса, необходимый для трансляции диспетчеру сообщений и управления через сервер отладки;
takt_counter	-	счётчик количества циклов посылок сообщений функции takt_work диспетчера расчётных модулей.

Данная область памяти используется расчётными модулями и сервером отладки для подключения к нужным переменным в общей области памяти, а также для управления диспетчером РМ.

Область памяти /exemod_struct, содержит информацию о текущих загруженных расчётных модулях и состоит из записей типа: количество структур = количеству расчётных модулей в /header, представлено на рисунке 2.3.2.

```
typedefstruct

  {
      Char  exename[STR_LEN];
      Char algname[STR_LEN];
      int  offset;
  int takt_mod;
      int   tek_time;
      int   num_work;
      int   pid;
      int   chid;
      int   coid;
      int   din_vars_bytes;
      int   alg_vars_bytes;
      int   state_vars_bytes;
      int   constants_bytes;
      int   number_extvars;
  } exemod_data;
```

Рисунок 2.3.2

На рисунке 2.3.2 представлены следующие элементы:

exename - имя расчетного модуля (исполняемого файла);

algname - имя алгоритма, указываемое в файле конфигурации загрузки;

offset - начальное смещение расчетного модуля в общем массиве структур сигналов (область памяти /extvars_struct);

takt_mod - - такт работы расчетного модуля в мс, заданный в файле конфигурации загрузки (см. п.2.2);

tek_time - текущее время до начала исполнения, мс. Когда текущее время становится равным нулю, модуль запускается на счет. Описание как эта переменная используется можно посмотреть в функции takt_work, исходный текст которой приведён в п.2.2 настоящего описания;

num_work - количество вызовов за один такт;

pid - идентификатор процесса расчетного модуля;

chid - идентификатор канала обмена в ОСРВ версии QNX6 (в версии QNX4 – параметр не используется);

coid - идентификатор клиента, используемый в MsgSend для посылки сообщений от диспетчера расчётному модулю (только для ОСРВ версии QNX 6, в версии QNX4 параметр не используется);

din_vars_bytes - размер области памяти динамических переменных, выделенных в общей памяти расчётным модулем, байт. Необходим для работы отладочного сервера;

intalg_vars_bytes - размер области памяти алгебраических переменных, выделенных в общей памяти расчётным модулем, байт. Необходим для работы отладочного сервера;

intstate_vars_bytes - размер области памяти переменных состояния, выделенных в общей памяти расчётным модулем, байт. Необходим для работы отладочного сервера;

int constants_bytes - размер области памяти констант, выделенных в общей

памяти расчётным модулем, байт. Необходим для работы отладочного сервера;

number_extvars - количество внешних сигналов.

Область памяти /extvars_struct содержит описание всех внешних переменных, выделенных при загрузке диспетчером и состоит из записей типа ext_var_info_record. В поле index для каждой из структур при этом содержится байтовое смещение переменной для единой области памяти хранящей значения внешних переменных (/extvars_value).

Алгоритм поиска внешней переменной в расчётном модуле следующий:

1) из /header получаем количество загруженных расчетных модулей;

2) по имени алгоритма для заданного запущенного расчётного модуля производим поиск по имени в области памяти /exemod_struct нужной структуры exemod_data, которая описывает алгоритм. Именно поэтому имя алгоритма при запуске расчетного модуля должно обязательно указываться;

3) из структуры exemod_data получаем начальный индекс в массиве структур внешних переменных для расчетного модуля, находим начало описания внешних переменных расчетного модуля в области памяти /extvars_struct;

4) последовательно считываем данные о смещениях ext_var_info_record.index внешних переменных, хранящихся в общей области разделяемой памяти нужных расчетных модулей (количество внешних переменных расчетного модуля определено).

Пример как производить поиск внешней переменной для осуществления перепаковки данных или иной обработке данных, можно посмотреть в п.3 настоящего описания, а также в исходных кодах расчётного модуля (calc\calcmain.c). Пример поиска внешней переменной в области памяти по имени приведен в Приложении А.

Схематично алгоритм поиска адреса внешней переменной при загрузке расчётного модуля представлен на рисунке 2.3.3.

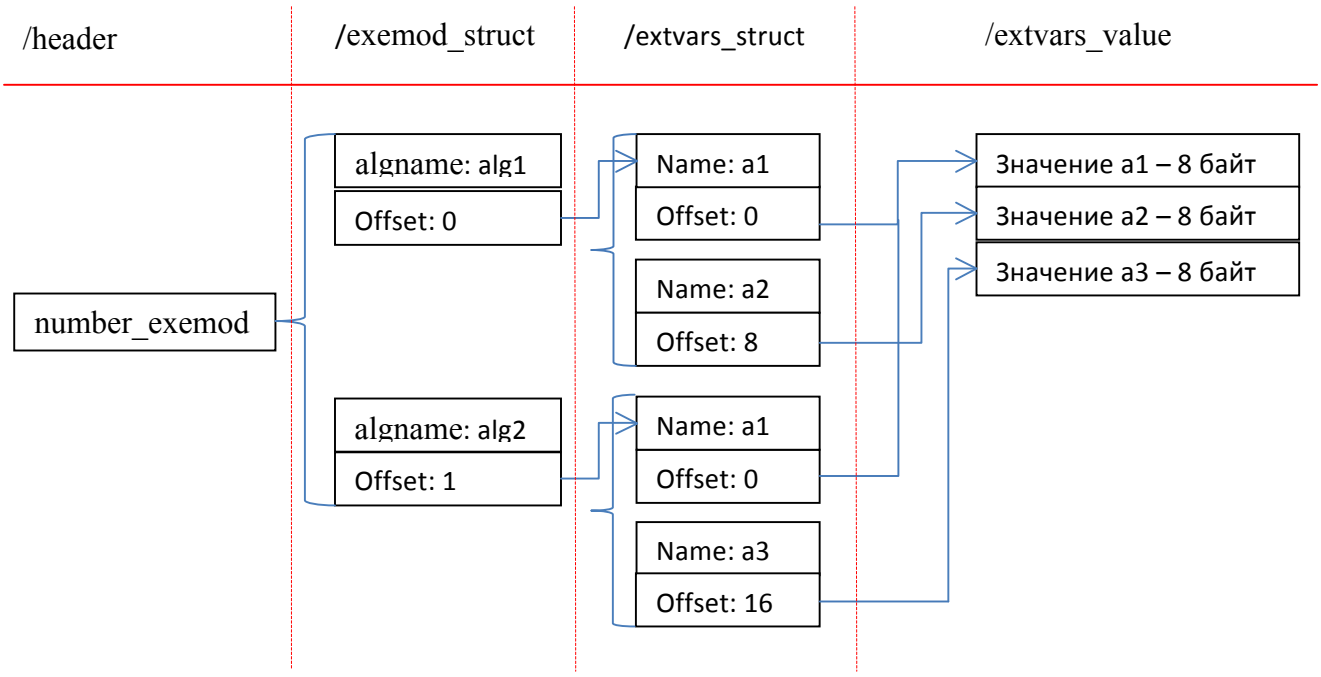


Рисунок 2.3.3

Общая область памяти /extvars_value, выделяется диспетчером при запуске расчётных модулей и содержит в себе текущие значения для всех внешних переменных, используемых по всех РМ. При загрузке каждый из РМ получает адреса нужных ему внешних переменных в данной области памяти по вышеприведённому алгоритму.

Кроме указанных выше областей памяти, выделяемых диспетчером расчётных модулей перед загрузкой всех РМ, каждый из них при старте выделяет общие области памяти под свои внутренние переменные. Внутренние переменные, в отличие от внешних, не объединяются. Выделяемые области памяти – это рабочие массивы, в которых последовательно расположены значения внутренних переменных, приём для разного класса внутренних переменных выделяются разные области памяти:

- /din_vars_<имя алгоритма указанное при запуске>
- /alg_vars _<имя алгоритма указанное при запуске>
- /state_vars _vars_<имя алгоритма указанное при запуске>
- /constants _vars_<имя алгоритма указанное при запуске>

2.4 ФОРМАТЫ КОМАНДНОЙ СТРОКИ

Формат командной строки запуска диспетчера расчётных модулей:

DispExemod<имя файла конфигурации загрузки, если не указано – то принимается равным default.conf> <имя начального состояния прибора>

Согласно имени начального состояния из файлов загружаются значения переменных состояния для каждого из расчётных модулей из файлов. Если начальное состояние (точка рестарта) не указано или не сохранено ранее, то расчётные модули запускаются с нулевого состояния, которое было заложено при генерации кода. Состояние может быть сохранено из отладочного клиента ПК МВТУ-4 (описано в руководстве пользователя [1]).

Сами файлы состояний переменных хранятся в текущей директории расчётных модулей и включают в себя следующие файлы, представляющие собой дампы соответствующих областей памяти:

< имя начального состояния прибора ><имя алгоритма>dinvars - дампы динамических переменных для данного состояния и данного алгоритма (для каждого алгоритма в загрузке и каждого состояния – свой файл);

< имя начального состояния прибора ><имя алгоритма>algvars - дампы алгебраических переменных для данного состояния и данного алгоритма;

< имя начального состояния прибора ><имя алгоритма>statevars - дампы переменных состояния для данного состояния и данного алгоритма;

< имя начального состояния прибора > extvars – дамп всех внешних переменных.

Формат командной строки запуска сервера отладки:

GdbServer<номер порта сервера, если не указано – то принимается равным 22375>

2.5 СЕТЕВОЙ ПРОТОКОЛ СЕРВЕРА ОТЛАДКИ

Протокол обмена данными использует транспортный протокол TCP-IP:

1) на приборе запускается отдельный процесс GdbServer, который открывает и слушает TCP-порт с указанным номером (по умолчанию 22375);

2) клиент ПК МВТУ-4 инициирует соединение с прибором по протоколу TCP-IP на указанный порт. При этом на приборе при соединении происходит размножение (fork) процесса-сервера, то есть для каждого клиентского соединения существует свой процесс-обработчик;

3) клиент ПК МВТУ-4 может удалённо запустить диспетчер расчётных модулей с указанного файла конфигурации загрузки. При этом передаётся имя файла конфигурации загрузки и имя начального состояния. Если начальное состояние с указанным именем отсутствует, то расчётные модули запускаются с состоянием по умолчанию. Передаваемые данные представлены в таблице 2.5.1.

Таблица 2.5.1

Описание	Размер, байт	Значение
Размер передаваемого пакета в байтах	4	Равно размеру всех данных в пакете начиная со следующего параметра (идентификатора команды)
Идентификатор команды	1	9
Размер имени загрузки в байтах NameSize	4	целое число
Имя загрузки в кодировке ASCII	NameSize	Форматимени: <имя файла конфигурации загрузки>#<имя начального состояния> Пример: default.conf#initstate1

Одновременно клиент считывает данные, в соответствии с таблицей 2.5.2.

Таблица 2.5.2

Описание	Размер, байт	Значение
Размер принимаемого пакета в байтах	4	Равно размеру принимаемых данных за исключением самого размера.
Результат операции соединения	1	0 – успешно != 0 – ошибка (код ошибки)

4) для инициализации обмена данными с указанным алгоритмом клиент передаёт данные, в соответствии с таблицей 2.5.3.

Таблица 2.5.3

Описание	Размер, байт	Значение
Размер передаваемого пакета в байтах	4	равно размеру всех данных в пакете начиная со следующего параметра (идентификатора команды)
Идентификатор команды	1	ноль
Идентификатор клиентской части ConnectionId	4	равно uid объекта внутри ПК МВТУ-4
Размер имени алгоритма в байтах NameSize	4	целое число
Имя алгоритма в кодировке ASCII	NameSize	формат имени алгоритма, передаваемого от клиента: <имя алгоритма>#<имя конфигурации загрузки>

		Пример: algo1#default.conf

При соединении с сервером обмена клиент считывает данные, в соответствии с таблицей 2.5.4.

Таблица 2.5.4

Описание	Размер, байт	Значение
Размер принимаемого пакета в байтах	4	Равно размеру принимаемых данных за исключением самого размера.
Результат операции соединения	1	0 – успешно != 0 – ошибка (код ошибки) 1 – сетевая ошибка, 2 – нет такой запущенной программы

После приёма от клиента пакета с данными сервер обмена подключается к областям памяти диспетчера расчётных модулей и ищет по имени указанный алгоритм в списке загруженных алгоритмов и получает адреса его областей данных. Если алгоритм успешно найден, то клиенту возвращается код операции 0, если не найден – то 2. После этого клиент может выполнять дальнейшие действия.

5) Если сервер возвратил ответ что алгоритм найден, то клиент даёт серверу обмена команды на добавление нужных переменных на чтение от прибора. Команда передаётся по TCP-IP. При этом если указанная клиентом переменная есть, то она добавляется в список передачи данных на TCP сервере прибора. При добавлении переменной клиент передаёт данные в соответствии с таблицей 2.5.5.

Таблица 2.5.5

Описание	Размер, байт	Значение
Размер передаваемого пакета в байтах	4	Равно размеру всех данных в пакете начиная со следующего параметра (идентификатора команды)
Идентификатор команды	1	1
Количество добавляемых переменных	4	N- целое число >=1 – количество переменных которые надо добавить для считывания
Циклi = от 1 до N		
Размер имени i-й переменной в байтах NameSize(i)	4	целое число
Имяi-й переменной в кодировке ASCII	NameSize(i)	
Конец цикла от 1 до N		

Одновременно клиент считывает данные, в соответствии с таблицей 2.5.6.

Таблица 2.5.6

Описание	Размер, байт	Значение
Размер принимаемого пакета в байтах	4	Равно размеру принимаемых данных за исключением самого размера.
Цикл от 1 до N		
Тип данных в соответствии с таблицами описания переменных на приборе	1	0 – вещественное (double) 1 – двоичное (char) 2 – целое (longint – 32 bit) Если тип = 255 то переменная не найдена
Массив размерностей переменной	12 байт	Dims: array [0..2] ofinteger Если все размерности 0 – то переменная не найдена
Конец цикла от 1 до N		

6) Клиент даёт серверу обмена команды на добавление нужных переменных на запись в прибор. Команда передаётся по TCP-IP. При этом если указанная клиентом переменная есть, то она добавляется в список приёма данных на TCP сервере.

При добавлении переменной клиент передаёт данные в соответствии с таблицей 2.5.7:

Таблица 2.5.7

Описание	Размер, байт	Значение
Размер передаваемого пакета в байтах	4	Равно размеру всех данных в пакете начиная со следующего параметра (идентификатора команды)
Идентификатор команды	1	1
Количество добавляемых переменных	4	N- целое число >=1 – количество переменных которые надо добавить для запись
Цикл i = от 1 до N		
Размер имениi-й переменнойв байтах NameSize(i)	4	целое число
Имя i-й переменной в кодировке ASCII	NameSize(i)	
Конец цикла от 1 до N		

Одновременно клиент считывает данные, в соответствии с таблицей 2.5.8.

Таблица 2.5.8

Описание	Размер, байт	Значение
Размер принимаемого пакета в байтах	4	Равно размеру принимаемых данных за исключением самого размера.
Цикл от 1 до N		
Тип данных в соответствии с таблицами описания переменных на приборе	1	0 – вещественное (double) 1 – двоичное (char) 2 – целое (longint – 32 bit) Если тип = 255 то переменная не найдена
Массив размерностей переменной	12 байт	Dims: array [0..2] ofinteger Если все размерности 0 – то переменная не найдена
Конец цикла от 1 до N		

7) Для обмена данными (т.е. записи или чтения ранее добавленных в список переменных) Клиент в заданные моменты времени транслирует данные, которые добавлены в список записи в прибор и принимает данные, которые добавлены в список чтения прибора. При этом данные транслируются в виде бинарного потока. Передаваемые данные представлены в таблице 2.5.9.

Таблица 2.5.9

Описание	Размер, байт	Значение

Размер передаваемого пакета в байтах	4	Равно размеру всех данных в пакете начиная со следующего параметра (идентификатора команды)
Идентификатор команды	1	3
ЦиклI = от 1 до N, где N– количество переменных добавленных на запись		
Байтовый поток для i-й переменной в соответствии с её размерностью и типом данных	M	
Конец цикла от 1 до N		

Принимаемые клиентом данные представлены в таблице 2.5.10.

Таблица 2.5.10

Описание	Размер, байт	Значение
Размера пакета сообщения в байтах, DataSize (за исключением самого размера пакета).	4	DWORD
Шаг моделирования в секундах, fStep	8	Double
Условное модельное время с начала соединения с контроллером, сек	8	Double
Набор флагов состояния прибора fStateFlag	4	DWORD
ЦиклI = от 1 до ReadList.Count, ReadList.Count – к-во переменных, добавленных на чтение		
Байтовый массив для i-й переменной в соответствии с её типом данных и размерностью	M	
Конец цикла от 1 до ReadList.Count		

8) Клиент ПК МВТУ-4 может передать диспетчеру расчётных модулей сигнал управления (для приостановки расчёта, возобновления и для того чтобы убить процесс). Соответствующие параметры представлены на рисунке 2.5.11.

Таблица 2.5.11

Описание	Размер, байт	Значение
Размер передаваемого пакета в байтах	4	Равно размеру всех данных в пакете начиная со следующего параметра (идентификатора команды)
Идентификатор команды	1	6
Значение сигнала	4	Значение сигнала, передаваемого процессу-диспетчеру расчётных модулей. Возможные значения:

		– пауза = 23; – продолжение = 25; – убить процесс = 15.

Одновременно клиент считывает данные, в соответствии с таблицей 2.5.12.

Таблица 2.5.12

Описание	Размер, байт	Значение
Размер принимаемого пакета в байтах	4	Равно размеру принимаемых данных за исключением самого размера;
Результат операции соединения	1	Если больше нуля, то ошибка.

9) Клиент ПК МВТУ-4 в процессе расчёта может сохранить для задачи именованный рестарт. Передаваемые клиентом данные представлены в таблице 2.5.13.

Таблица 2.5.13

Описание	Размер, байт	Значение
Размер передаваемого пакета в байтах	4	Равно размеру всех данных в пакете начиная со следующего параметра (идентификатора команды)
Идентификатор команды	1	7
Размер имени рестарта в байтах NameSize	4	целое число
Имя рестарта в кодировке ASCII	NameSize	

Одновременно клиент считывает данные, в соответствии с таблицей 2.5.14.

Таблица 2.5.14

Описание	Размер, байт	Значение
Размер принимаемого пакета в байтах	4	Равно размеру принимаемых данных за исключением самого размера.
Результат операции соединения	1	0 – успешно != 0 – ошибка (код ошибки)

10) Клиент ПК МВТУ-4 может в процессе расчёта загрузить именованный рестарт. Передаваемые клиентом данные представлены в таблице 2.5.15.

Таблица 2.5.15

Описание	Размер, байт	Значение
Размер передаваемого пакета в байтах	4	Равно размеру всех данных в пакете начиная со следующего параметра (идентификатора команды)
Идентификатор команды	1	8
Размер имени рестарта в байтах NameSize	4	целое число
Имя рестарта в кодировке ASCII	NameSize	

Одновременно клиент считывает данные, в соответствии с таблицей 2.5.16.

Таблица 2.5.16

Описание	Размер, байт	Значение
Размер принимаемого пакета в байтах	4	Равно размеру принимаемых данных за исключением самого размера.
Результат операции соединения	1	0 – успешно != 0 – ошибка (код ошибки)

3 ВСТРАИВАНИЕ АЛГОРИТМА ПЕРЕПАКОВКИ ПЕРЕМЕННЫХ

Встраивание алгоритма упаковки-распаковки сигналов (т.е. подпрограммы, которая будет связывать переменные в исполняемой среде с переменными других программных подсистем прибора) целесообразно производить в шаблон расчётных модулей, т.к. это обеспечит мгновенную трансляцию изменённого сигнала в нужную область памяти системы ввода-вывода, причём только тех, которые необходимо транслировать. Для встраивания необходимо внести изменения в основной модуль шаблона расчётных модулей: файл calcmain.c в директории <директория установки>\CodeTemplates\QNX 6 (или 4). Исходный текст файла с указанием мест для дополнения приведен в приложении Б.

Примечание - Соответственно в самом начале функции main в модуле calcmain.c пользователю необходимо добавить код, который обеспечивает соединение с областями памяти подсистемы ввода-вывода.

4 ОБЕСПЕЧЕНИЕ СИНХРОНИЗАЦИИ РАБОТЫ ДИСПЕТЧЕРА ПО СОБЫТИЮ ОТ ВНЕШНЕГО ИСТОЧНИКА (ПОДСИСТЕМЫ ВВОДА-ВЫВОДА)

Для того чтобы синхронизировать запуск расчётных модулей с некоторым внешним источником, целесообразно создать специальный расчётный модуль, которые не будет содержать внешних переменных, но будет обеспечивать ожидание до выполнения некоторого события. При этом вы должны будете в файле конфигурации загрузки указать необходимый минимальный интервал для всех расчётных модулей. При этом необходимо учитывать, что в настоящий момент инкрементация счётчиков времени в расчётных модулях производится с заданным постоянным шагом. Необходимо также учитывать и то, что шаг инкрементации времени в начальный момент при ожидании события принципиально неизвестен (а известен только предыдущий шаг). Это в частности не позволяет нормально реализовать счётчики времени для задержек. То есть реализовать многие алгоритмы чисто в событийном режиме невозможно (для скручивания счётчиков времени необходимо повторять выполнение алгоритма с заданным шагом). Пример простейшего кода расчётного модуля с указанием места встраивания ожидания события синхронизации приведён в приложении В.

5 ОПИСАНИЕ ПРОЦЕССА СБОРКИ РАСЧЁТНОГО МОДУЛЯ ДЛЯ QNX 6 И QNX 4

Сборка по схеме расчётного модуля производится по следующему алгоритму:

А) генератор кода ПК SimInTech по открытой схеме в указанной в настройках папке создаёт следующие файла:

<имя программы>.h - файл в котором декларируются привязки переменных к нужным структурам внутри расчётного модуля.

<имя программы>.inc – файл исходного текста программы, сгенерированной по схеме – главная вычислительная секция.

<имя программы>_init.inc – файл исходного текста программы, сгенерированной по схеме – секция инициализации начальных состояний переменных и констант.

<имя программы>_state.inc – файл исходного текста программы, сгенерированной по схеме – секция запоминания состояний (выполняется всегда строго после главной секции).

Имя программы – это заданное в параметрах расчёта исходной схемы имя алгоритма (расчётного модуля).

Б) Генератор кода ПК SimInTech запускает скрипт автоматической сборки (имя скрипта может быть изменено в настройках окна «Инструменты автоматики» - см. Руководство пользователя). Данный скрипт находится в папке выбранного в настройках генератора кода шаблона кода и по умолчанию называется compile.bat. Данному скрипту сообщаются следующие параметры: директория, куда были размещены сгенерированные генератором кода файлы (*.inc и *.h), адрес сборочного сервера, пароль и логин для сборочного сервера. Данный скрипт производит копирование сгенерированных генератором кода файлов в директорию с исходным текстом расчётного модуля (<директория шаблона кода>\calc). Файлы копируются под фиксированными именами:

<имя программы>.h -> prog.h

<имя программы>.inc -> prog.inc

<имя программы>_init.inc -> init.inc

<имя программы>_state.inc -> state.inc

Далее данный скрипт запускает процедуру автоматической сборки из исходного кода исполняемого файла расчётного модуля. В разработанных шаблонах кода скрипт сборки исполняемого модуля называется build.bat. Ниже приведён текст скрипта сборки compile.bat и скрипта компиляции build.bat применительно к QNX 6. Для ОС QNX 4 отличается только скрипт build.bat.

compile.bat:

```
REM Скрипт сборки расчётного модуля

REM формат команды  compile.bat <путь к исходникам\имя модуля>

REM Копируем файлы исходного текста из заданной директории

call clear.bat

del /q %1

copy "%~1.inc" /B calc\prog.inc /B

copy "%~1.h" /B calc\prog.h /B

copy "%~1_init.inc" /B calc\init.inc /B

copy "%~1_state.inc" /B calc\state.inc /B

REM Компилируем DLL и пишем результат компиляции в отчёт

call build.bat 2>"%~1.log"

REM Копируем результат сборки обратно

copy calc\x86\o\calc /B %1 /B
```

build.bat для QNX6 – сборка производится кросс-компилятором на рабочем месте пользователя:

```
REM Пакетная сборка расчётного модуля для QNX

REM Тут указываем пути к компилятору и вспомогательным программам

set path=%path%;"%QNX_HOST%\usr\bin\"

REM Собственно компиляция - для 6.4 под x86

i386-pc-nto-qnx6.4.0-gcc.exe -o calc\x86\o\calc calc\calcmain.c -fpack-struct=1
"-I%QNX_TARGET%\usr\include" -IDispExemod -Wconversion -l m

REM Собственно компиляция - для 6.5 под x86

i486-pc-nto-qnx6.5.0-gcc.exe -o calc\x86\o\calc calc\calcmain.c -fpack-struct=1
"-I%QNX_TARGET%\usr\include" -IDispExemod -Wconversion -l m
```

Как видно из вышеприведённого сборочного скрипта build.bat для QNX6 запускается кросс-компилятор под выбранную платформу из директории средств разработки, поставляемых вместе с QNX. При этом компиляция осуществляется в директорию calc шаблона расчётного модуля, а потом скомпилированный расчётный модуль копируется (и переименовывается) из неё обратно в директорию, куда были размещены файлы, сгенерированные генератором кода ПК SimInTech (см. конец compile.bat).

build.bat для QNX4 – сборка производится нативным компилятором, непосредственно на приборе (или специально выделенном под эти цели сборочном приборе):

```
REM Удалённая сборка не целевой (или промежуточной тестовой) системе для QNX4
REM синтаксис build.bat <сервер> <логин> <пароль>
pushd calc
del/q buildcmds.txt
del/q calc
REM Копирование задания на сборочный сервер
ECHO open %1 > ftpcmd.txt
ECHO %2>> ftpcmd.txt
ECHO %3>> ftpcmd.txt
ECHO binary>> ftpcmd.txt
ECHO rmdir calcsrc>> ftpcmd.txt
ECHO mkdir calcsrc>> ftpcmd.txt
ECHO cd calcsrc>> ftpcmd.txt
ECHO mput *.*>> ftpcmd.txt
ECHO quit>> ftpcmd.txt
ftp -s:ftpcmd.txt -i
REM Собственно сборка (соединение через telnet) - при помощи утилиты
ECHO %2> buildcmds.txt
ECHO %3>> buildcmds.txt
ECHO cd calcsrc>> buildcmds.txt
ECHO make>> buildcmds.txt
..\..\..\telbuilder %1 buildcmds.txt
```

```
REM Копирование скомпилированного со сборочного сервера домой

ECHO open %1 > ftpcmd.txt

ECHO %2>> ftpcmd.txt

ECHO %3>> ftpcmd.txt

ECHO binary>> ftpcmd.txt

ECHO cd calcsrc>> ftpcmd.txt

ECHO get calc calc>> ftpcmd.txt

ECHO quit>> ftpcmd.txt

ftp -s:ftpcmd.txt -i

del/q ftpcmd.txt

popd
```

Для QNX 4 файлы сгенерированные генератором кода сначала копируются в локальную директорию calc шаблона кода QNX 4, затем по ftp вся локальная директория calc копируется на сборочный сервер (прибор), там собирается нативным компилятором, а результаты компиляции (исполняемый файл и лог компиляции) копируются по ftp обратно.

Также процесс сборки кода описан в Руководстве пользователя.

Сгенерированные генератором кода файлы (*.inc и *.h) подключаются к основному файлу шаблона расчётного модуля calcmain.c при помощи операторов #include, ниже приведён код подключения к базовой части сгенерированной части (фрагмент calcmain.c):

```
#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <malloc.h>

#include <signal.h>

#include <string.h>

#include <fcntl.h>

#include <sys/mman.h>

#include <sys/kernel.h>

#include <process.h>

#include <math.h>

#pragma pack (1)
```

```
#include "c_types.h"
#include "prog.h"
#define STR_LEN 80
#define OK 0x01
#include "main_struct.h"
EXPORTED_FUNC INIT_FUNC( double step,
                                    double time,
                                    ptr_array*        ext_vars_addr,
                                    double_array*     din_vars,
                                    double_array*     derivates,
                                    double_array*     alg_vars,
                                    double_array*     alg_funcs,
                                    t_state_vars*     state_vars,
                                    t_consts*         consts
                                  )
{  int ret;
   #include "init.inc"
   return ret;
};
EXPORTED_FUNC RUN_FUNC(
                                    int action,
                                    double step,
                                    double time,
                                    ptr_array*        ext_vars_addr,
                                    double_array*     din_vars,
                                    double_array*     derivates,
                                    double_array*     alg_vars,
                                    double_array*     alg_funcs,
                                    t_state_vars*     state_vars,
                                    t_consts*         consts,
```

```
                                    t_local*          locals
                                )
{  int ret;
   #include "prog.inc"
   return ret;
};
EXPORTED_FUNC STATE_FUNC(
                                    int action,
                                    double step,
                                    double time,
                                    ptr_array*        ext_vars_addr,
                                    double_array*     din_vars,
                                    double_array*     derivates,
                                    double_array*     alg_vars,
                                    double_array*     alg_funcs,
                                    t_state_vars*     state_vars,
                                    t_consts*         consts,
                                    t_local*          locals
                                )
{  int ret;
   #include "state.inc"
   return ret;
};
……..
```

Сам алгоритм генерации кода для отдельных блоков схемы находится в библиотеке get_lib.dll, исходные тексты которой находятся в папке <директория установки программы>\source\MBTY\AVRORA_GEN\ . В общем виде процесс генерации кода выглядит следующим образом:

- графическая оболочка загружает схему

- математическое ядро mbtylib.dll анализирует правильность задания параметров и сортирует схему

- математическое ядро подгружает для каждого из блоков модуль генерации кода из библиотеки генерации кода get_lib.dll.

- математическое ядро вызывает процедуры генерации кода последовательно для всех блоков из библиотеки генерации кода get_lib.dll.

Собственно алгоритмы формирования текста для каждого из блоков находятся в модуле Blocks.pas в директории <директория установки программы>\source\MBTY\AVRORA_GEN\

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Система программирования для вычислительных приборов на базе программного обеспечения SimInTech. Руководство пользователя.

2 Программный комплекс МВТУ - версия 4.0. Описание интерфейса графической оболочки. МГТУ им. Н.Э Баумана. - М., 2006.

3 Инструментальные средства программирования информационно-управляющих комплексов в среде ОС РВ QNX // Современные технологии автоматизации. 2008. № 3 – М., 2008.

ПРИЛОЖЕНИЕ А

(обязательное)

Код поиска внешней переменной

Пример поиска внешней переменной в области памяти по имени.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <sys/neutrino.h>
#include <process.h>
#pragma pack (1)
#include "c_types.h"
#define STR_LEN 80
#define OK 0x01
#include "main_struct.h"

/*Причленимся к области разделяемой памяти область разделяемой памяти*/
void connect_to_shmem(void);
/*Поиск в области разделяемой памяти структуру описывающую данный модуль*/
void *search_my_struct(char *exename, char *alg);
/*Освобождаем область разделяемой памяти*/
void clear(int sig);
/*Область памяти под значения переменных внешних сигналов*/
int extvars_value_handle;
/*Заголовок диспетчера*/
int header_handle;
/*Общая область памяти под хранение структур внешних сигналов*/
int extvars_struct_handle;
/*Оласть для хранения структур описывающие расчетные модули*/
int exemod_handle;
/*Указатель на общий массив значений сигналов*/
unsigned char *ptr_extvars_value;
/*Указатель на заголовок*/
header *ptr_header;
/*Область структур внешних сигналов*/
ext_var_info_record *ptr_extvars_struct;
void **external_vars_addrs;
int main(int argc, char *argv[]) {
        int i;
        int indx = 0;

/*Присоеденимся к области разделяемой памяти*/
connect_to_shmem();
//Поиск заданной переменной по имени в массиве описаний внешних переменных
 for(i=0;i ptr_header->num;i++){
if (strcmp("mysearchname", ptr_extvars_struct[ i].name) == 0)
{
//Нашли переменную с указанным именем  !!!
indx = ptr_extvars_struct[ i].index;
external_vars_addrs[i] = &ptr_extvars_value[indx];
//И что-то с ней сделали если нам очень надо
}
};
};
```

```
/*Присоеденимся к области разделяемой памяти*/
void connect_to_shmem(void)
{
/*Выделим память под заголовок диспетчера*/
/*Выделим и инициализируем общую область памяти для хранения структур
описывающих
 * внешнии сигналы
 */
header_handle = shm_open("/header", O_RDWR, 0777);
if (header_handle == -1)
{
perror("shm_open header_handle");
exit(EXIT_FAILURE);
} /*if ( base_header_handle == -1 )*/
/*Получим указатель на область разделяемой памяти заголовка*/
ptr_header = mmap(0, sizeof(header), PROT_READ | PROT_WRITE, MAP_SHARED,
header_handle, 0);
/*Выделим общую область разделяемой памяти под хранение
 * массива структур внешних сигналов
 */
extvars_struct_handle = shm_open("/extvars_struct", O_RDWR, 0777);
if (extvars_struct_handle == -1)
{
perror("shm_open extvars_struct_handle");
exit(EXIT_FAILURE);
} /*if ( base_header_handle == -1 )*/
/*Указатель на общий массив структур*/
ptr_extvars_struct = mmap(0, sizeof(ext_var_info_record) * ptr_header->num,
PROT_READ | PROT_WRITE, MAP_SHARED, extvars_struct_handle, 0);
/*Выделим память для общего списка значений внешних сигналов*/
extvars_value_handle = shm_open("/extvars_value", O_RDWR, 0777);
if (extvars_value_handle == -1)
{
perror("shm_open extvars_value");
exit(EXIT_FAILURE);
} /*if ( base_header_handle == -1 )*/
/*Получим указатель на область разделяемой памяти указывающей на место хранения
 * значений сигналов
 */
ptr_extvars_value = mmap(0, ptr_header->signals_size, PROT_READ | PROT_WRITE,
MAP_SHARED, extvars_value_handle, 0);
}
```

ПРИЛОЖЕНИЕ Б

(обязательное)

Исходный текст файла calcmain.c

```
       //Тут начинается цикл привязки адресов внешних переменных, которые нужны
данному расчётному модулю
       for(i=0;i < n_ext_vars;i++){
                        offset = ptr_exemod_my->offset;
//В этой строке мы получили описатель переменной и получили её смещение в
рабочем массиве внешних переменных
                        indx = ptr_extvars_struct[offset + i].index;
//Тут мы получили адрес внешней переменной
                    external_vars_addrs[i] = &ptr_extvars_value[indx];
//Вот в этом месте мы можем добавить данную внешнюю переменную в список
преобразования
//Добавьте тут свой код, например
//AddConvert(ptr_extvars_struct[offset    +    i],        external_vars_addrs[i],
&convert_list);
//   ptr_extvars_struct[offset + i] – структура типа   ext_var_info_record, в
которой есть полное описание данной переменной
// external_vars_addrs[i] – указатель на место где хранится значение переменной
// convert_list – указатель на список, содержащий ваши собственные структуры для
//преобразования данных
//Данная функция должна получить адрес и описание переменной в рабочей области
памяти //внешних переменных, сопоставить её имя со списком, в котором указано
сопоставление //ваших переменных (подсистемы ввода-вывода) и технологических
имён (внешних //переменных исполняемой среды).
       };

Строка  539:
//Тут начинает цикл работы расчётного модуля
while (1){
/*Прием сообщения о начале вычисления от диспетчера*/
   rcvid = MsgReceive(ptr_exemod_my->chid, cmd, sizeof(cmd), NULL);
    _time = 0.0;
     //Вычисление шага задачи в секундах
step = ptr_exemod_my->takt_mod*0.001;
//Вот тут надо встроить вызов функции, которая будет перебрасывать значения из
вашей //системы ввода – вывода в общую область памяти внешних переменных.
Например:
//  ConvertFrom(&convert_list);

     //Начало вычисления шага задачи
       RUN_FUNC(
                                    f_GoodStep,
                                    step,
                                    _time,
                                    external_vars_addrs,
                                    din_vars,
                                    din_deri,
                                    alg_vars,
                                    alg_funcs,
                                    state_vars,
                                    constants,
                                    local_vars
                                );
```

```
                    //Вызов пост-функции для запоминания состояний
        STATE_FUNC(
                                        f_GoodStep,
                                        step,
                                        _time,
                                        external_vars_addrs,
                                        din_vars,
                                        din_deri,
                                        alg_vars,
                                        alg_funcs,
                                        state_vars,
                                        constants,
                                        local_vars
                                    );

                    //Запрос  значений производных динамических переменных
        RUN_FUNC(
                                        f_GetDeri,
                                        step,
                                        _time,
                                        external_vars_addrs,
                                        din_vars,
                                        din_deri,
                                        alg_vars,
                                        alg_funcs,
                                        state_vars,
                                        constants,
                                        local_vars
                                    );

        //Интегрирование динамических переменных
        for(i=0;i < din_vars_dim;i++){
           (*din_vars)[i] = (*din_vars)[i] + step*(*din_deri)[i];
        };

      //Ограничение модельного времени по разрядности
      if(_time > 1e6){_time = 0;};

      //Увеличение времени
      _time = _time + step;

//Вот тут надо встроить вызов функции, которая будет перебрасывать значения из
общей //области памяти внешних переменных в  области памяти системы ввода-
вывода. Например:
//  ConvertTo(&convert_list);

      /*Сообщим модулю о конце расчета*/
        MsgReply(rcvid, OK, cmd, sizeof(cmd));

} /*while (1)*/
```

ПРИЛОЖЕНИЕ В

(обязательное)

Код расчётного модуля с указанием места встраивания ожидания события

```
//Стандартные библиотеки
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <malloc.h>
#include <signal.h>
#include <string.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <sys/neutrino.h>
#include <process.h>

#pragma pack (1)
#include "c_types.h"

/*!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!*/
/*Длинна строки*/
#define STR_LEN 80
#define OK 0x01
#include "main_struct.h"
/*Причленимся к области разделяемой памяти область разделяемой памяти*/
void connect_to_shmem(void);
/*Поиск в области разделяемой памяти структуру описывающую данный модуль*/
void *search_my_struct(char *exename, char *alg);
/*Освобождаем область разделяемой памяти*/
void clear(int sig);
/*Область памяти под значения переменных внешних сигналов*/
int extvars_value_handle;
/*Заголовок диспетчера*/
int header_handle;
/*Общая область памяти под хранение структур внешних сигналов*/
int extvars_struct_handle;
/*Оласть для хранения структур описывающие расчетные модули*/
int exemod_handle;
/*Указатель на общий масив значений сигналов*/
unsigned char *ptr_extvars_value;
/*Указатель на заголовок*/
header *ptr_header;
/*Область структур описывающих модули*/
exemod_data *ptr_exemod;
/*Область структур внешних сигналов*/
exemod_data *ptr_exemod_my;
//Описание области памяти для СВВ
ioheader *input_output ;
int io_handle;

 //Main
int main(int argc, char *argv[]) {

        int i;
        int indx = 0;
        int offset = 0;
        int rcvid = 0;
        char cmd[1];
```

```
      /*Присоеденимся к области разделяемой памяти*/
      connect_to_shmem();

      /*Поиск в области разделяемой памяти структуру описывающую данный модуль*/
      ptr_exemod_my = search_my_struct(argv[0], argv[1]);

      if (ptr_exemod_my == NULL)
      {
            exit(1);
      }

//Регистрируемся у сервера ввода-вывода (по указанному ему ch_id он должен слать
сообщения о том, что пришли данные):
input_output->clientpid = getpid();
input_output-> ch_id = ChannelCreate(0);

      while (1)
      {
      /*Прием сообщения о начале вычислений*/
      rcvid = MsgReceive(ptr_exemod_my->chid, cmd, sizeof(cmd), NULL);
//Тут мы выполняем ожидание некоторого события синхронизации (т.е. сообщения от
другой
// системы), например:
//rcvid = MsgReceive(input_output->ch_id, cmd, sizeof(cmd), NULL);
// где input_output_ch_id  - идентификатор канала обмена сообщениями, который мы
должны
// взять из соотв-й структуры СВВ, расположенной в общей области памяти

            /*Сообщим модулю о конце расчета*/
       MsgReply(rcvid, OK, cmd, sizeof(cmd));

      } /*while (1)*/
     return EXIT_SUCCESS;
};

/*Присоеденимся к области разделяемой памяти*/
void connect_to_shmem(void)
{

      /*Выделим память под заголовок диспетчера*/
      /*Выделим и инициализируем общую область памяти для хранения структур
описывающих
       * внешнии сигналы
       */
      header_handle = shm_open("/header", O_RDWR, 0777);

      if (header_handle == -1)
      {
            perror("shm_open header_handle");
            exit(EXIT_FAILURE);

      } /*if ( base_header_handle == -1 )*/

      /*Получим указатель на область разделяемой памяти заголовка*/
      ptr_header = mmap(0, sizeof(header), PROT_READ | PROT_WRITE, MAP_SHARED,
header_handle, 0);

      /*Выделим область разделяемой памяти для списка структур
       * описывающих расчетные модули
```

```
       */
      exemod_handle = shm_open("/exemod_struct", O_RDWR, 0777);

      if (exemod_handle == -1)
      {
            perror("shm_open exemod_handle");
            exit(EXIT_FAILURE);

      } /*if ( base_header_handle == -1 )*/

      /*Область структур описывающая расчетные модули*/
      ptr_exemod  =  mmap(0,  sizeof(exemod_data)  *  ptr_header->number_exemod,
PROT_READ | PROT_WRITE, MAP_SHARED, exemod_handle, 0);
    //Управляющая область памяти для СВВ
  io_handle = shm_open("/ioheader ", O_RDWR, 0777);
  input_output = mmap(0, sizeof(ioheader) , PROT_READ | PROT_WRITE, MAP_SHARED,
io_handle, 0);
}

/*Поиск в области разделяемой памяти структуру описывающую данный модуль*/
void *search_my_struct(char *exename, char *alg)
{

      int count = 0;
      exemod_data *ptr_exemod_my = NULL;



      /*Тепрь в массиве структур созданных диспетчером и
       * описывающих расчетные модули найдем структуру описывающую
       * данный расчетный модуль
       */
       for (count = 0; count < ptr_header->number_exemod; count++)
       {

             /*Двойной выстрел в голову! по имени модуля и имени алгоритма*/
             if (strcmp(exename, ptr_exemod[count].exename) == 0 && strcmp(alg,
ptr_exemod[count].algname) == 0 )
             {

                   /*Вот наша структура запомним ее*/
                   ptr_exemod_my = &ptr_exemod[count];

                   /*Создадим канал для обмеена сообщениями с диспетчером
                    * только для QNX6.4.1
                    */
                   ptr_exemod_my->chid = ChannelCreate(0);
                   ptr_exemod_my->pid = getpid();
             }
       }
      return ptr_exemod_my;
}

void clear(int sig)
{
      /*Отчленимся от области разделяемой памяти созданные
       * диспетчером алгоритмов
       */
      close(header_handle);
      close(exemod_handle);
```

```
        exit(0);
}
```

ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

Изм.	Номера листов				Всего листов в документе	№ документа	Вх. № сопроводительного документа и дата	Подп.	Дата
	измененных	замененных	новых	аннулиро-ванных