1. Инструментальные средства разработки контроллеров на основе ОМК

В настоящее время существуют разнообразные средства, позволяющие разрабатывать и отлаживать ПО для микроконтроллеров, повышающие эффективность труда разработчика. Среди них можно выделить следующие типы:

* внутрисхемные эмуляторы;
* программные симуляторы;
* оценочные платы;
* мониторы отладки;
* эмуляторы ПЗУ.

1.1 Внутрисхемные эмуляторы.

Внутрисхемный эмулятор – программно аппаратное средство, способное замещать собой эмулируемый процессор в реальной схеме. Внутрисхемный эмулятор - это наиболее мощное и универсальное отладочное средство. «Хороший» внутрисхемный эмулятор делает процесс функционирования отлаживаемого контроллера прозрачным, т.е. легко контролируемым, произвольно управляемым и модифицируемым по воле разработчика.

Функционально внутрисхемные эмуляторы делятся на стыкуемые с внешней вычислительной машиной (обычно это бывает IBM PC), и функционирующие автономно. Автономные внутрисхемные эмуляторы имеют индивидуальные вычислительные ресурсы, средства ввода-вывода, не требуют для своей нормальной работы стыковки с какими-либо внешними вычислительными средствами, но за это пользователю приходится расплачиваться либо существенно более высокой ценой, либо пониженными функциональными и сервисными возможностями по сравнению с аналогичными моделями, стыкуемыми с IBM PC.

Как минимум, эмулятор содержит следующие функциональные блоки:

* отладчик;
* узел эмуляции микроконтроллера;
* эмуляционная память;
* подсистема точек останова.

Более продвинутые модели могут содержать дополнительно:

* процессор точек останова;
* трассировщик;
* профилировщик (анализатор эффективности программного кода);
* таймер реального времени;
* программно-аппаратные средства, обеспечивающие возможность чтения и модификации ресурсов эмулируемого процессора «на лету», т.е. в процессе выполнения программы пользователя в реальном времени;
* программно-аппаратные средства, обеспечивающие синхронное управление, необходимые для эмуляции в мультипроцессорных системах;
* интегрированную среду разработки

1.2 Симуляторы

Симулятор – программное средство, способное имитировать работу микроконтроллера и его памяти. Как правило, симулятор содержит в своем составе:

* отладчик;
* модель ЦПУ и памяти.

Более продвинутые симуляторы содержат в своем составе модели встроенных периферийных устройств, таких, как таймеры, порты, АЦП, и системы прерываний.

Симулятор должен уметь загружать файлы программ во всех популярных форматах, максимально полно отображать информацию о состоянии ресурсов симулируемого микроконтроллера, а также предоставлять возможности по симуляции выполнения загруженной программы в различных режимах. В процессе отладки модель "выполняет" программу, и на экране компьютера отображается текущее состояние модели.

Загрузив программу в симулятор, пользователь имеет возможность запускать ее в пошаговом или непрерывном режимах, задавать условные или безусловные точки останова, контролировать и свободно модифицировать содержимое ячеек памяти и регистров симулируемого микропроцессора. С помощью симулятора можно быстро проверить логику выполнения программы, правильность выполнения арифметических операций.

В зависимости от класса используемого отладчика, различные симуляторы могут поддерживать высокоуровневую символьную отладку программ.

Некоторые модели симуляторов могут содержать ряд дополнительных программных средств, таких например как: интерфейс внешней среды, встроенную интегрированную среду разработки.

1.3 Отладочные мониторы

Отладочный монитор – специальная программа, загружаемая в память отлаживаемой системы. Она вынуждает процессор пользователя производить, кроме прикладной задачи, еще и отладочные функции:

* загрузку прикладных кодов пользователя в свободную от монитора память;
* установку точек останова;
* запуск и останов загруженной программы в реальном времени;
* проход программы пользователя по шагам;
* просмотр, редактирование содержимого памяти и управляющих регистров.

Программа монитора обязательно должна работать в связке с внешним компьютером или пассивным терминалом, на которых и происходит визуализация и управление процессом отладки. Повторим, что отладочные мониторы используют тот процессор, который уже стоит на плате пользователя. Достоинством этого подхода являются очень малые затраты при сохранении возможности вести отладку в реальном времени. Главным недостатком является отвлечение ресурсов микроконтроллера на отладочные и связные процедуры, например: монитор занимает некоторый объем памяти, прерывания, последовательный канал. Объем отвлекаемых ресурсов зависит от искусства разработчика монитора.

1.4 Оценочные платы

Оценочные платы являются своеобразными конструкторами для макетирования прикладных систем. В последнее время, при выпуске новой модели кристалла микроконтроллера, фирма-производитель обязательно выпускает и соответствующую оценочную плату. Обычно это печатная плата с установленным на ней микроконтроллером, плюс вся необходимая ему стандартная обвязка. На этой плате также устанавливают схемы связи с внешним компьютером. Как правило, там же имеется свободное поле для монтажа прикладных схем пользователя. Иногда, имеется уже готовая разводка для установки дополнительных устройств, рекомендуемых фирмой. Например, ПЗУ, ОЗУ, ЖКИ-дисплей, клавиатура, АЦП и др.

Для большего удобства, платы развития комплектуются еще и простейшим средством отладки на базе монитора отладки. Однако, здесь проявились два разных подхода: один используется для микроконтроллеров, имеющих внешнюю шину, а второй - для микроконтроллеров, не имеющих внешней шины.

В первом случае отладочный монитор поставляется фирмой в виде микросхемы ПЗУ, которая вставляется в специальную розетку на плате развития. Плата также имеет ОЗУ для программ пользователя и канал связи с внешним компьютером или терминалом. Примером здесь может служить плата развития фирмы Intel для микроконтроллера 8051.

Во втором случае, плата развития имеет встроенные схемы программирования внутреннего ПЗУ микроконтроллера, которые управляются от внешнего компьютера. В этом случае, программа монитора просто заносится в ПЗУ микроконтроллера совместно с прикладными кодами пользователя. Прикладная программа при этом специально должна быть подготовлена: в нужные ее места вставляют вызовы отладочных подпрограмм монитора. Затем осуществляется пробный прогон. Чтобы внести в программу исправления пользователю надо стереть ПЗУ и произвести повторную запись. Готовую прикладную программу получают из отлаженной путем удаления всех вызовов мониторных функций и самого монитора отладки.

Возможности по отладке, предоставляемые комплектом "плата развития плюс монитор" безусловно не столь универсальны, как возможности внутрисхемного эмулятора, да и некоторая часть ресурсов микропроцессора в процессе отладки отбирается для работы монитора. Тем не менее, наличие законченного набора готовых программно-аппаратных средств, позволяющих без потери времени приступить к монтажу и отладке прикладной системы, во многих случаях является решающим фактором. Особенно если учесть, что стоимость такого комплекта несколько меньше чем стоимость более универсального эмулятора.

1.5 Эмуляторы ПЗУ

Эмулятор ПЗУ – программно-аппаратное средство позволяющее, замещать ПЗУ на отлаживаемой плате, и подставляющее вместо него ОЗУ, в которое может быть загружена программа с компьютера через один из стандартных каналов связи. Это устройство позволяет пользователю избежать многократных циклов перепрограммирования ПЗУ. Эмулятор ПЗУ имеет смысл только для микроконтроллеров, которые в состоянии обращаться к внешней памяти программ. Это устройство сравнимо по сложности и по стоимости с платами развития. Оно имеет одно большое достоинство: универсальность. Эмулятор ПЗУ может работать с любыми типами микроконтроллеров.

Ранние эмуляторы ПЗУ позволяли только загружать программу, запускать ее и останавливать, используя общий сброс. Затем появились усложненные модели с аппаратной выработкой сигналов трассировки по достижении определенного адреса на осциллограф. Эмулируемая память в таких изделиях была доступна для просмотра и модификации, но очень важный контроль за внутренними управляющими регистрами микроконтроллера был до недавнего времени невозможен.

Однако появились модели интеллектуальных эмуляторов ПЗУ, которые позволяют «заглядывать» внутрь микроконтроллера на плате пользователя и вообще, по управлению отладкой, стали похожими на внутрисхемный эмулятор.

1.6 Выбор средства для разработки и отладки контроллера на основе ОМК

Несмотря на то, что внутрисхемные эмуляторы являются мощным и функциональным средством отладки, они имеют существенный недостаток ­– высокую цену. Более дешёвым и практически не уступающим по функциональности является использование симулятора и оценочного модуля.

В настоящее время наиболее популярны интегрированные среды разработки, или студии, включающие в свой состав редакторы текстов программ, отладчики, симуляторы, компоновщики, программаторы.

Применительно к контроллерам фирмы Atmel, такими средствами разработки являются Avr Studio с компилятором Avr Asm самой компании Atmel, программаторы Avr Real, Pony Prog, Chip Prog, PicProg, среды разработки Project Avr, Algorithm Builder, Vmlab, Proteus.

Наиболее функциональными и удобными среди них являются Proteus и Vmlab, причём Vmlab является полностью бесплатным, в отличие от Proteus. Поэтому логично использовать именно Vmlab в качестве интегрированной среды разработки для контроллера.

Возможно также использование графической среды программирования, такой как Algorithm Builder. Преимущества хороших графических сред программирования очевидны: это легкость освоения и быстрота разработки. К сожалению Algorithm Builder является требовательной к квалификации пользователя и знанию конкретных особенностей программирования для ОМК.

В качестве языка программирования можно рассматривать ассемблер и Си. Преимущества разработки на языке высокого уровня очевидны: сокращенное время проектирования, более простые поиск ошибок, переносимость, легкость повторного применения. Программисту на Си не приходится самому реализовывать, например, 16-ти разрядную арифметику на 8-ми разрядном ОМК, или поддержку арифметики с плавающей точкой. Вместо этого он может полностью сосредоточиться на алгоритмах управления и обработки.

Таким образом, наиболее удобной будет являться система с графической средой программирования и инвариантной к квалификации пользователя.

1. Алгоритмы по сбору и обработки измерительной информации

В процессе создания проблемно-ориентированной системы должны быть рассмотрены алгоритмы позволяющие выполнять следующие функции*:*

многоканальное преобразование входных сигналов в виде уровней напряжения, тока или частотно временных сигналов;

обеспечивать возможность работы каналов с индивидуальной градуировочной характеристикой;

формирование аналоговых или частотно-временных сигналов (напряжение, временной интервал, генератор прямоугольных импульсов, широтно-импульсный модулятор);

ввод-вывод дискретных сигналов;

настройка контроллера на требуемый набор функций.

Алгоритмы обработки измерительной информации должны включать допусковый контроль, фильтрацию помех, приведение к физическому параметру. Исходя из требуемой точности, разрядность обрабатываемых данных не превышает 16 бит.

2.1 Алгоритм опроса каналов измерения

Алгоритмы опроса каналов измерения напряжений, токов, частотно-временных сигналов отличаются только особенностями формирования и анализа сигналов начала и конца преобразования. Поэтому в качестве примера рассмотрим алгоритм опроса многоканального АЦП.

С каждым каналом АЦП связан буфер данных. Максимальный размер буфера данных канала ограничен доступным объемом ОЗУ. Буферизация данных необходима для последующего сглаживания измерительной информации.

Возможен циклический или выборочный опрос каналов АЦП. Работа с каналом разрешается установкой бита в регистре каналов. Каждый канал конфигурируется в отдельности. В качестве параметров канала выступают, размер буфера канала, нижняя и верхняя границы допускового контроля, адрес таблицы аппроксимации, регистр обработки канала.

Цикл опроса считается завершенным, когда буферы всех обрабатываемых каналов будут заполнены. После завершения цикла опроса происходит обработка накопленных данных. Алгоритмы обработки данных задаются битами регистра обработки канала.

Чтение значения канала осуществляется по прерыванию. Обработка по прерыванию предпочтительнее обработки по готовности, т.к. позволяет избежать простоя при ожидании завершения преобразования.

Схема алгоритма обработчика готовности АЦП представлена на рисунке 2.1. Принятые на рисунке 2.1 обозначения:

* BI – индекс буфера канала;
* CI – номер канала, который обрабатывается в данный момент;
* W – переменная для хранения значения канала;
* PBI – следующая позиция в буфере BI канала;
* NB – количество не полностью заполненных буферов каналов.

Значение, полученное с текущего канала, проходит допусковый контроль. Значением является непосредственно код, не приведенный к физическому параметру. Если допусковый контроль не пройден, то контроллер немедленно формирует сообщение об ошибке. Если допусковый контроль пройден то, код помещается в текущую позицию буфера обрабатываемого канала. Если буфер при этом заканчивается, то счетчик незаполненных буферов декрементируется.

NB = 0?

Начало

Рисунок 2.1 – Схема алгоритма опроса каналов АЦП

Буфер BI заполнен?

Конец

Да

Нет

Чтение значения канала в переменную W

Допусковый контроль

Получение индекса буфера BI по номеру текущего канала CI

PBI := PBI + 1

Запись в позицию PBI буфера BI значения W

NB := NB - 1

Да

Запуск следующего канала CI с незаполненным буфером

Нет

Счетчик незаполненных буферов позволяет определить закончен ли цикл опроса каналов АЦП. Если остались незаполненные буферы, то происходит запуск преобразования на следующем разрешенном канале.

Каждый буфер данных имеет уникальный числовой идентификатор. Идентификатор буфера передается в качестве входного параметра подпрограммам обработки данных, которые запускаются после завершения цикла опроса АЦП. Результаты обработки данных передаются в ПЭВМ.

2.1 Алгоритмы сглаживания измерительной информации

Важными алгоритмами, применяемыми к накопленным данным канала, является алгоритмы сглаживания. Сглаживание измерительной информации применяется в случаях, когда возникает необходимость в фильтрации измерительных данных от помех. Наиболее распространенными алгоритмами сглаживания являются допусковый контроль, метод «скользящего среднего» и медианный фильтр.

Суть скользящего среднего заключается в том, что значение i-го элемента заменяется средним арифмитическим из K соседних элементов (предыдущих, следующих за ним, а также его самого). Первый и последний элемент выборки считаются отдельно. Обычно K=3, в этом случае:

*M[1] = (5 \* M[0] + 2 \* M[1] – M[2])/6;*

*M[n - 1] = (5 \* M[n – 1] + 2 \* M[n-2] – M[n-3])/6;*

*M[i] = (M[i-1] + M[i] + M[i+1])/3.*

Принятые на рисунке 2.2 обозначения:

* n – количество элементов массива;
* i – счетчик элементов;
* M – массив данных.

Рисунок 2.2 – Схема алгоритма скользящего среднего

Начало

Конец

i := 0;

M[0] := (5 \* M[0] + 2 \* M[1] – M[2]) / 6;

M[n–1] := (5 \* M[n–1] + 2 \* M[n–2] – M[n–3]) / 6;

i = n – 1

M[i] = (M[i–1] + M[i] + M[i+1]) / 3;

i := i + 1

Нет

Да

Суть сглаживания по методу медианного фильтра заключается в сортировке массива и выборке среднего по индексу элемента. Основное достоинство медианного сглаживания – устойчивость к наличию выбросов.

Для упорядочения массива могут быть использованы алгоритмы сортировки «пузырьком», вставки и отбора. Так как размер массивов сравнительно мал, абсолютная разница во времени сортировки пренебрежимо мала. Поэтому целесообразно использовать алгоритм сортировки «пузырьком» как наиболее простой в реализации.

Схема алгоритма медианной отбраковки представлена на рисунке 2.3. Принятые на рисунке 2.3 обозначения:

* n – количество элементов массива;
* i – счетчик элементов;
* M – массив данных;
* buf – переменная-буфер;
* flag – признак обмена;
* result – результат медианной отбраковки.

Рисунок 2.3 – Схема алгоритма медианной отбраковки

Нет

Начало

i := 0;

flag := 0;

M[i] > M[i+1]

buf := M[i+1];

M[i+1] := M[i];

M[i] := buf;

flag := 1

i := i + 1

i = n - 2

Да

Да

Нет

Да

flag := 0

Конец

Result := M[n div 2]

Нет

Алгоритм допускового контроля основал на том, что при работе объекта значения каждого из контролируемых технологических параметров не могут выходить за определенные границы:

*low\_bound <= x <= high\_bound (X),* (2.1)

Контроль достоверности по этому алгоритму заключается в проверке выполнения условия (2.1) для каждого значения измерительной информации. Блок-схема алгоритма представлена на рисунке 2.4. Принятые на рисунке обозначения:

* n – количество элементов входного массива массива;
* i – счетчик элементов для входного массива данных;
* M – входной массив данных;
* j – счетчик элементов для выходного массива данных;
* result – выходной массив данных;
* lb – нижняя граница для допускового контроля;
* hb – верхняя граница для допускового контроля.

Рисунок 2.4 – Схема алгоритма допускового контроля

i := 0;

j := 0;

M[i] >= lb

И

M[i] <= hb

i < n

i := i+1

да

Result[j] := M[i]

j := j+ 1

да

нет

нет

Начало

Конец

2.3 Приведение данных к физическому параметру

Значение измеряемой физической величины может быть однозначно определено по значению выходного кода АЦП. Для этого используются градуировочные данные, полученные в результате калибровки датчика.

Если функция преобразования линейная, то значение измеряемой физической величины определяется по формуле:

, (2.2)

где Х1, Х2 – начальная и конечная точки градуировки;

n1, n2 – значения выходного кода АЦП в соответствующих точках градуировки (отсчеты).

Если функция преобразования нелинейная и для повышения точности используется кусочно-линейная аппроксимация, то по текущему значению выходного кода АЦП n программным путем определяется интервал (ni,ni+1), где этим значениям кодов соответствуют Xi и Xi+1, хранящиеся в памяти. Используя перечисленные параметры, значение измеряемой физической величины определяется по формуле:

, (2.3)

где Хi, Хi+1 – i-я и i+1-я точки градуировки;

ni, ni+1 – значения выходного кода АЦП в соответствующих точках градуировки (отсчеты).

Кусочно-линейная аппроксимация может использоваться в большинстве случаев, но сравнительно трудоемка для вычисления (операции сравнения, умножения, деления, вычитания, сложения). Если требуется минимизировать время приведения к параметру, целесообразно использовать таблицы, содержащие все возможные точки градуировки. По коду однозначно определяется адрес точки в таблице и значение параметра. Недостатком данного метода является необходимость использования большого объема памяти для хранения таблицы. Схема алгоритма приведения к физическому параметру с использованием кусочно-линейной аппроксимации показана на рисунке 2.5.

Принятые на рисунке 2.5 обозначения:

К - количество элементов массива;

I - счетчик элементов;

N - приводимый к параметру код;

N - массив точек отсчета;

X - физический параметр;

Xi - элемент массива физических параметров.

I := 0

Начало

N > N[I+1]?

Да

Конец

Нет

I := I + 1



Рисунок 2.5 – Схема алгоритма приведения к физическому параметру

Подразумевается, что приводимый к параметру код удовлетворяет условию N[0] < N < N[K-1]. Градуировочная таблица представлена как совокупность двух массивов: отсчетов и физических значений.

2.4 Вычисление контрольной суммы

Метод контрольной суммы применяется для определения достоверности приема кадра данных. Существует несколько методов вычисления контрольных сумм. В системах на базе модулей ICP DAS используется «сумма по модулю 256». Если совместимость с модулями ICP DAS не требуется и необходима разработка собственной системы команд, то целесообразно рассмотреть метод вычисления циклических избыточных кодов (CRC).

Основная идея метода состоит в следующем: данные трактуются как единая битовая последовательность, которая делится на фиксированное двоичное число (порождающий полином). Остаток от деления является значением CRC. Существует несколько алгоритмов вычисления CRC, различающихся значениями порождающих полиномов. Далее приведена схема формирования CRC кода с полиномом 0x1021 (рисунок 2.6). Данный метод стандартизован в спецификации V.41 МККТТ «Кодонезависимая система контроля ошибок».

0

15

С

CRC

Данные

C = 1

Да

0x1021

CRC-полином

Нет

Сумма по модулю 2

Данные=0

Нет

Да

Рисунок 2.6 – Схема формирования CRC-кода

Формирование завершено

2.5 Алгоритмы сбора и обработки дискретных сигналов

Контроллер рассчитан на побайтный ввод-вывод дискретных данных. Если число дискретных сигналов больше восьми, то предполагается последовательно-параллельная обработка данных. При ограниченном количестве линий ввода-вывода возможно использование внешних сдвиговых регистров, управляемых синхронно сигналами контроллера. В этом случае потребуется линия синхронизации и линия данных для управления внешними регистрами.

Обобщенный алгоритм сбора и обработки дискретных сигналов представлен на рисунке 2.7.

Начало

Да

Конец

Рисунок 2.7 – Схема алгоритма сбора и обработки дискретных сигналов

Ввод?

Обработка входных данных

Инициализация

Запись в выходной регистр

Нет

На стадии инициализации проверяется настройка счетчиков или портов контроллера, определяется значение выходных сигналов при включении питания и сбросе.

Алгоритм обработки выходных данных определяется назначением контроллера и, как правило, заключается в анализе бит и организации ветвлений в зависимости от их значений. Если требуется подсчет количества внешних сигналов, то он может быть организован программно или аппаратно (с помощью счетчиков-таймеров).

С точки зрения пользователя, самым удобным решением является использование параметризуемых модулей, использующих вышеописанные алгоритмы.

1. Общая методика проектирования автоматизированной системы

3.1 Итеративный инкрементальный подход к разработке ПО

Итеративный инкрементный подход к разработке ПО берет свое начало с середины 50-х годов прошлого столетия. Тогда понятие итеративная разработка сводилась к исправлению уже сделанного, то в контексте современных методов быстрой разработки этот термин означает нечто иное: не просто пересмотр проделанной работы, но и эволюционное продвижение вперед. Итеративный инкрементный подход основывается на базовом формальном описании системы, дающем возможность создать первую исполняемую функциональную модель. Полученная модель проверяется на соответствие описанию системы, а затем расширяется далее, последовательно преобразуясь в новые модели, в которых отражается увеличение требований к системе и уточнение деталей их реализации. Процесс продолжается до трансформации модели в реальную программную систему.

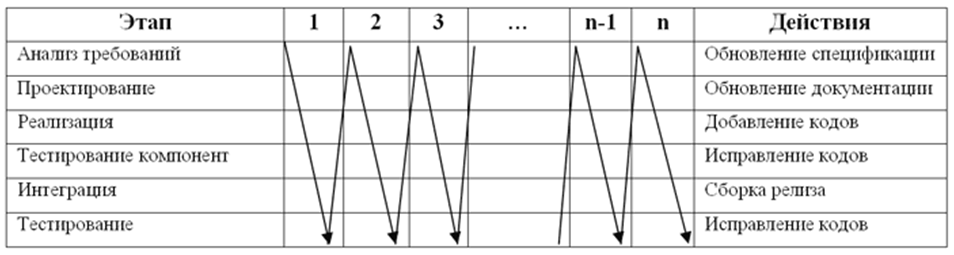


Рисунок 3.1 – Схема итеративного инкрементального подхода

Итеративный подход предполагает разбиение жизненного цикла проекта на последовательность итераций, каждая из которых напоминает «мини-проект», включая все этапы жизненного цикла ПО в применении к созданию меньших фрагментов функциональности, по сравнению с проектом, в целом. Цель каждой итерации - получение работающей версии ПО, включающей функциональность всех предыдущих и текущей итерации. Результат финальной итерации содержит всю требуемую функциональность продукта. Таким образом, с завершением каждой итерации, продукт развивается инкрементно.

Шансы успешного создания сложной системы будут максимальными, если она реализуется в серии небольших шагов и если каждый шаг заключает в себе четко определенный результат, а также возможностьотката назад, к результатам предыдущей успешной итерации, в случае неудачи. Перед тем, как пустить в дело все ресурсы, предназначенные для создания ПО, разработчик имеет возможность получать обратную связь из реального мира (заказчиков, пользователей) и исправлять возможные ошибки в проекте.

Эволюционная модель подразумевает возможность не только сборки работающей (с точки зрения результатов тестирования) версии системы, но и её развертывания в реальных операционных условиях с анализом откликов пользователей для определения содержания и планирования следующей итерации. Поскольку на каждом шаге мы имеем работающую систему, то можно:

* очень рано начать тестирование пользователями;
* принять стратегию разработки в соответствии с бюджетом, полностью защищающую от перерасхода времени или средств (в частности, за счет сокращения второстепенной функциональности).

Таким образом, значимость эволюционного подхода на основе организации итераций особо проявляется в снижении неопределенности с завершением каждой итерации. В свою очередь, снижение неопределенности позволяет уменьшить риски. Итеративному разбиению может быть подвержен не только жизненный цикл ПО в целом, включающий перекрывающиеся этапы – формирование требований, проектирование, реализация и др., но и каждый этап может, в свою очередь, разбиваться на уточняющие итерации, связанные, например, с детализацией структуры декомпозиции проекта – например, архитектуры модулей ПО.

В настоящее время большинство специалистов отдают предпочтение итеративной инкрементальной модели. Широко известная водопадная модель имеет существенные недостатки. Вот некоторые из них:

* при формировании требований к ПО пользователи редко имеют четкое представление о том, что им нужно, и не могут сформулировать все, что им известно;
* даже если мы можем изложить все требования к системе, существует множество деталей, которые будут обнаружены лишь после того, как процесс проектирования продвинется довольно далеко;
* внешние силы приводят к изменению требований, причем некоторые из этих изменений могут свести на нет ранее принятые решения.
* представление о том, будто разработчик ПО создает свой программный продукт свободным от ошибок на основе спецификации требований, абсолютно нереалистично. Ошибки в требованиях и их реализации выявляются только в конце проекта, когда написан весь код, поэтому трудоемкость их исправления становится просто огромной.

3.2 Разработка через тестирование

В последнее время всё популярнее становится одна из практик экстремального программирования – разработка через тестирование. Её суть заключается в том, что модульные тесты для программы или её фрагмента пишутся до самой программы. Разработка состоит из коротких циклов (длительностью от 2 минут, в зависимости от опытности и стиля работы программиста). В общем случае, каждый цикл состоит из следующих шагов:

* из репозитория извлекается программная система, находящаяся в согласованном состоянии, когда весь набор модульных тестов выполняется успешно;
* добавляется новый тест. Он может состоять в проверке, реализует ли система некоторое новое поведение или содержит ли некоторую ошибку, о которой недавно стало известно;
* успешно выполняется весь набор тестов, кроме нового теста, который выполняется неуспешно;
* программа изменяется с тем, чтобы как можно скорее выполнялись все тесты. Нужно добавить самое простое решение, удовлетворяющее новому тесту, и одновременно с этим не испортить существующие тесты. Большая часть нежелательных побочных и отдалённых эффектов от вносимых в программу изменений отслеживается именно на этом этапе, с помощью достаточно полного набора тестов;
* весь набор тестов выполняется успешно;
* теперь, когда требуемая в этом цикле функциональность достигнута самым простым способом, программа перестраивается для улучшения структуры и устранения избыточного, дублированного кода;
* весь набор тестов выполняется успешно;
* комплект изменений, сделанных в этом цикле в тестах и программе заносится в репозиторий, после чего программа снова находится в согласованном состоянии и содержит четко осязаемое улучшение по сравнению с предыдущим состоянием.
  1. Принцип модульного проектирования

Анализ требований на проектирование показал, что наиболее сложной задачей является обеспечение универсальности контроллера. Необходимо разработать легко перестраиваемую систему, также необходимо предусмотреть произвольный набор функций выполняемых контроллером.

Возможно несколько подходов к решению поставленных задач.

Наиболее простой в реализации является разработка программы для контроллера на языке высокого уровня, например C. Данное решение обеспечивает переносимость на уровне кода между ОМК различных производителей. Добавление новых команд и функций осуществляется путем модификации исходного кода программы, поэтому любые изменения требуют дополнительного времени на разработку, отладку и тестирование. Плюсом такого подхода является высокая скорость выполнения программы и приемлемые требования к объему памяти программ и ОЗУ.

Другим подходом является использование языка ассемблера. В этом случае переносимость кода между ОМК различных производителей возможна только при поддержке одной системы команд, например i8051. Скорость разработки и модификации программ на языке ассемблера наиболее низкая. Вероятность внесения ошибок высока. Достоинством является максимальная скорость исполнения программы и минимальные требования к объему памяти программ и ОЗУ.

Ни один из рассмотренных подходов не является приемлемым для реализации поставленной задачи.

Для обеспечения наибольшей универсальности и удобства использования, было принято решение разработать систему модульного проектирования программного обеспечения контроллера.

Для контроллера на языке C разработана основная программа и библиотека подпрограмм, предоставляющих доступ к аппаратным ресурсам контроллера и реализующих алгоритмы обработки данных, рассмотренные в предыдущих разделах. При этом программа не содержит в себе системы команд контроллера и способов реакции на команды. Вся эта информация представлена в виде данных особой структуры во внешней флэш ПЗУ контроллера. Программа контроллера производит распознавание, обработку команд и вызов необходимых подпрограмм из библиотеки, руководствуясь данными, хранящимися во внешней флэш ПЗУ.

Данные, необходимые для функционирования программы контроллера генерируются графическим редактором схем обработки команд. Пользователь редактора оперирует графическими модулями, которые, по сути, являются представлением подпрограмм из библиотеки контроллера. Входы модулей соответствуют входным параметрам подпрограмм, выходы – выходным. Входы и выходы модулей соединяются линиями связи. Линии связи задают последовательность вызова подпрограмм и передачи параметров между ними. Система команд контроллера также задается специальными графическими модулями.

После того, как пользователь задал систему команд и алгоритмы их обработки, происходит трансляция схемы. Результатом трансляции является байтовый массив определенной структуры, который прошивается во внешнее флэш ПЗУ контроллера. После этого контроллер готов к работе.

Таким образом, работа с контроллером производится на трех уровнях. На первом уровне осуществляется разработка библиотеки подпрограмм и соответствующих им графических модулей. Библиотека подпрограмм рассчитана на конкретную аппаратную реализацию контроллера и обеспечивает максимально полный доступ к функциям контроллера.

На втором уровне используется контроллер с уже установленной основной программой и библиотекой подпрограмм. С помощью редактора и библиотеки графических модулей задается конкретная система команд и функциональность контроллера. При этом перепрошивки памяти программ контроллера не требуется. Вместо этого осуществляется запись данных во внешнее флэш ПЗУ контроллера.

На третьем уровне происходит конфигурирование и эксплуатация контроллера с уже установленной системой команд и функциональностью.

Проектирование в графическом режиме обладает рядом достоинств: простота освоения, высокая скорость разработки, наглядность, автоматический контроль правильности сборки алгоритмов. Даже разработка библиотеки модулей не является сложной задачей, т.к. архитектура системы жестко задана и разработчик отвечает только за реализацию конкретных модулей, а не всей системы в целом.

Минусом выбранного подхода является некоторое снижение скорости работы контроллера по сравнению с не универсальными решениями. Накладные расходы связаны с частым вызовом подпрограмм, обращением к внешней флэш ПЗУ, специфическим способом передачи параметров между подпрограммами.

3.4 Интерфейс модуля

Интерфейс модуля должен включать в себя описание входов и выходов модуля с указанием типа данных для каждого входа и выхода.

Библиотека модулей представляет собой XML файл следующей структуры:

<root>

<block>

описание модуля 1

</block>

<block>

описание модуля 2

</block>

………

<block>

описание модуля N

</block>

</root>,

где N – количество модулей в библиотеке.

Структура описание модуля следующая:

<name>

название модуля

</name>

<id>

уникальный идентификатор модуля

</id>

<input>

описание входа 1

</input>

<input>

описание входа 2

</input>

………

<input>

описание входа NI

</input>

<output>

описание выхода 1

</output>

<output>

описание выхода 2

</output>

………

<output>

описание выхода NO

</output>,

где NI, NO – количество входов и выходов модуля соответственно.

Описания входа и выхода модуля имеют одинаковую структуру:

<name>

имя входа (выхода)

</name>

<datatype>

тип данных входа (выхода)

</datatype>

Редактирование XML файла библиотеки производится вручную, т.к. структура файла достаточно наглядна. Файл библиотеки должен находиться в папке components, расположенной в папке программы. Расширение файла cmp. При создании нового проекта все библиотеки из папки components будут доступны для выбора.

После добавления модуля в библиотеку системы проектирования, необходимо разработать подпрограмму модуля и добавить ее в библиотеку подпрограмм модулей ПО контроллера.

Подпрограмма модуля обязана иметь сигнатуру:

unsigned char BlockProc (unsigned short paramsAddr);

Входная переменная paramsAddr является адресом списка параметров в внешней флэш ПЗУ. Выполнение подпрограммы начинается с получения значений входных параметров. Получение i-того байта j-того параметра осуществляется следующим образом:

b = ComProcData.Memory [HALGetExtROMByte (paramsAddr + j) + i],

где b – полученное значение байта.

Выходные параметры передаются точно так же. Нумерация параметров модуля сквозная, т.е. сначала идут входные параметры, затем выходные. Количество и размер параметров модуля известны на этапе разработки подпрограммы, поэтому реализовать доступ к параметрам не составляет труда.

Обязательным требованием является контроль значений входных параметров. Если значения недопустимы, то нужно выставить код ошибки и выйти из подпрограммы модуля. Пример генерация ошибки:

ComProcData.LastError = LAST\_ERROR\_INVALID\_PARAM;

return BLOCK\_RES\_ERROR;

где LAST\_ERROR\_INVALID\_PARAM – код ошибки параметров;

BLOCK\_RES\_ERROR – значение, возвращаемое подпрограммой модуля при возникновении ошибки.

После разработки подпрограммы модуля ее нужно добавить в вектор подпрограмм модулей. Вектор подпрограмм модулей объявлен следующим образом:

BlockHandler\* BlockHandlers [BLOCKS\_COUNT];

При этом следует учитывать, что позиция обработчика в векторе равна идентификатору модуля.

Модуль готов к работе.