МАТЕРИАЛ 1. ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ДЛЯ ВСТРАИВАЕМЫХ МИКРОПРОЦЕССОРНЫХ СИСТЕМ.

Цель – изучить основные приёмы работы с программными кросс-средствами разработки ПО для микропроцессоров MCS-51 и производных (инструментальные средства).

ВВЕДЕНИЕ
особенности разработки и функционирования программного обеспечения для встраиваемых 8- и 16-разрядных микропроцессорных систем.

Характерные черты микроконтроллеров, которые предопределяют специфику разработки цифровых устройств на микроконтроллерах, разработки их прикладного программного обеспечения и использования:

1). - незначительная емкость управляемой (адресуемой) памяти относительно универсальных МП;
2). - незначительная емкость памяти присутствует на самом кристалле в различных вариациях;
3). - физическое и логическое разделение памяти программ (ПЗУ) и памяти данных (ОЗУ) (гарвардская арх.);
4). - упрощенная и ориентированная на задачи управления система команд;
5). - простые методы адресации данных и команд (переходов);
6). - специфическая организация ввода/вывода информации.

Базовая архитектура MCS-51

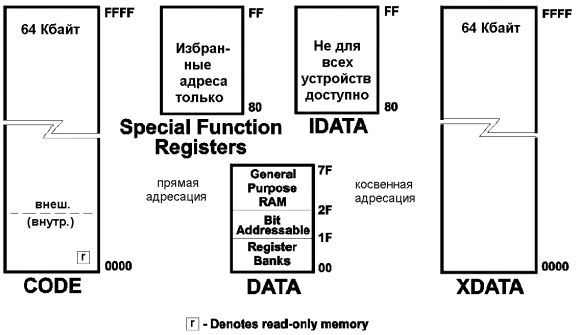


Рис. 1.

Специфика МК отражается на декомпозиции алгоритма работы устройства, на работе ассемблера и компиляторов, на работе компановщика, на функционировании отладочных средств и т.д. (Есть команды пересылки память-память (без помощи регистров). Узкое место – реентерантные процедуры. Узкое место – ограниченный стек (он организуется в одном из блоков данных). Самомодифицирующийся код программы возможен только с помощью дополнительной внешней логики. Различный доступ к данным одного формата, находящимся в разных типах памяти. И т.д. . . . . .)

Эти особенности предопределяют область использования микроконтроллеров как правило в качестве **специализированных вычислителей**, включенных в контур управления объектом или процессом.

Структурная организация, набор команд и аппаратурно-программные средства ввода/вывода информации микроконтроллеров лучше всего приспособлены для **решения задач управления и регулирования** в приборах, устройствах и системах автоматики, а не для решения задач обработки больших объёмов данных.

Микроконтроллеры не являются машинами классического "фон-Неймановского" типа, так как физическая и логическая разделенность памяти программ и памяти данных исключает возможность модификации и/или замены (перегрузки) прикладных программ микроконтроллеров во время работы, что сильно затрудняет их использование в качестве универсальных средств обработки данных. Не отрабатывается принцип «хранимой программы», что существенно **затрудняет отладку**. (В разной памяти одни и те же действия требуют разного количества машинных циклов на исполнение.)

В устройствах управления объектами (контроллерах) на основе МК аппаратурные средства и программное обеспечение существуют в форме **неделимого аппаратурно-программного комплекса**. При проектировании контроллеров приходится решать одну из самых сложных задач разработки, а именно задачу **оптимального распределения функций контроллера между аппаратурными средствами и программным обеспечением**. Решение этой задачи осложняется тем, что взаимосвязь и взаимовлияние аппаратурных средств и программного обеспечения в микропроцессорной технике претерпевают динамичные изменения.

Если в начале развития МП-техники определяющим было правило, в соответствии с которым аппаратурные средства обеспечивают производительность, а программное обеспечение — дешевизну изделия, то в настоящее время это правило нуждается в серьезной корректировке. Так как МК представляет собой стандартный массовый (относительно недорогой) логический блок, конкретное назначение которого определяет пользователь с помощью программного обеспечения, то с ростом степени интеграции и, следовательно, функционально-логических возможностей МК резко понижается стоимость изделия в пересчете на выполняемую функцию, что в конечном итоге и обеспечивает достижение высоких технико-экономических показателей изделий на МК. При этом затраты на разработку программного обеспечения изделия в 2—10 раз превышают (за время жизни изделия) затраты на приобретение и изготовление аппаратурных средств.

Архитектура.

Под **архитектурой** (программной) МП (или МК) понимается его программная модель, т.е. все его программно-видимые свойства. Под **микроархитектурой** понимается внутренняя реализация этой программной модели. Для одной архитектуры разными фирмами в разных поколениях применяются существенно различные микроархитектурные реализации. Клон MCS-51 насчитывает около 400 производных.

Специфика инструментальных средств для разработки прикладного ПО для МК в том, что они поддерживают как правило несколько архитектур. Так инструментальные средства **KEIL ELEKTRONIK GmbH** поддерживают 4 следующих архитектуры клона MCS-51: классическую структуру памяти 8051, расширения 8051 структуры памяти, универсальную структуру памяти 80C51MX процессоров Philips и структуру памяти 251 для Intel/Temic 251.

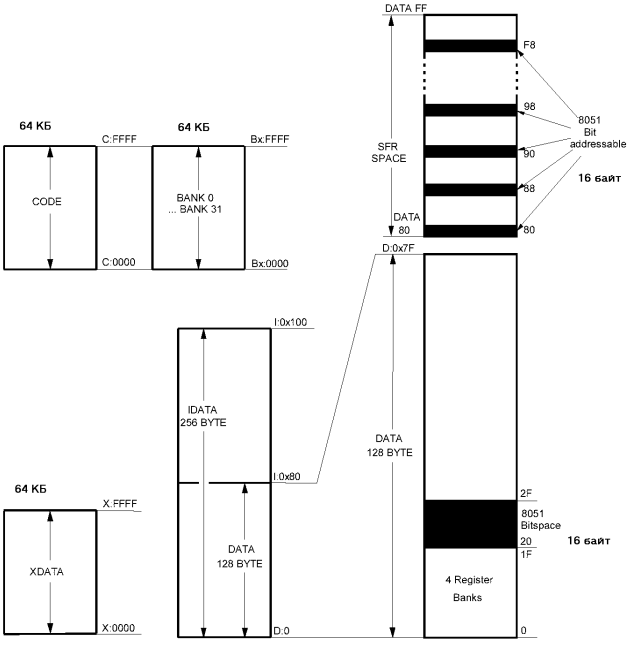
Память программ на кристалле и внешняя до 64 Кб – CODE (C). Обычно только читаемая, используется для

DATA	D:00 – D:7F
BIT	D:20 – D:2F
IDATA	I:00 – I:FF
XDATA	X:0000 – X:FFFF
CODE	C:0000 – C:FFFF
BANK 0	B0:0000 – B0:FFFF
… BANK 31	B31:0000 – B31:FFFF

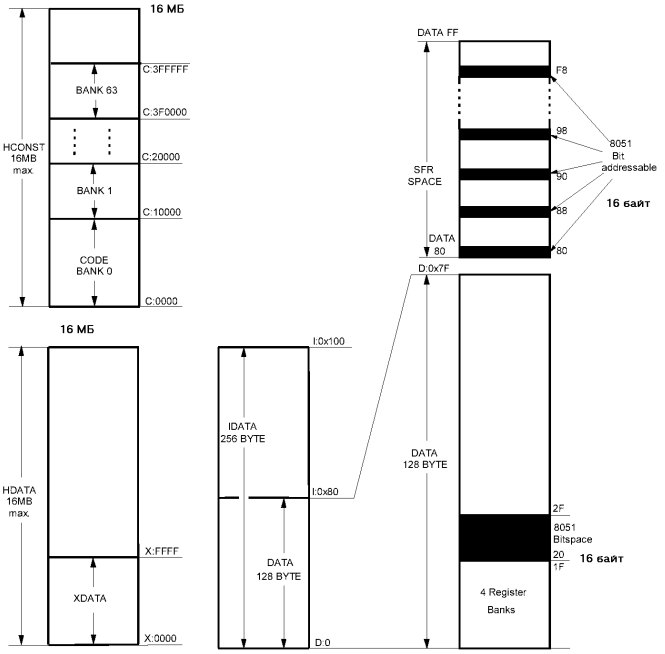
исполнительного кода и констант. В расширенных архитектурах до 16 Мб – ECODE и HCONST для констант в сегменте кода. **Внутренняя память данных** до 256 байт. Прямо адресуемая память – DATA (D). С доступом до бита – BIT. Косвенно адресуемая память через @R0 или @R1 – IDATA (I). У процессоров Philips и 251 эта область расширена до 64 Кб – EDATA. **Внешняя память данных** до 64 КБ – XDATA (X). 0 страница – PDATA (P). В расширенных архитектурах до 16 Мб – HDATA.

(Префиксы D, I, X, P, C, B0 … B31 используются в отладчиках. Указанные типы памяти в языках программирования – это **класс памяти**.)

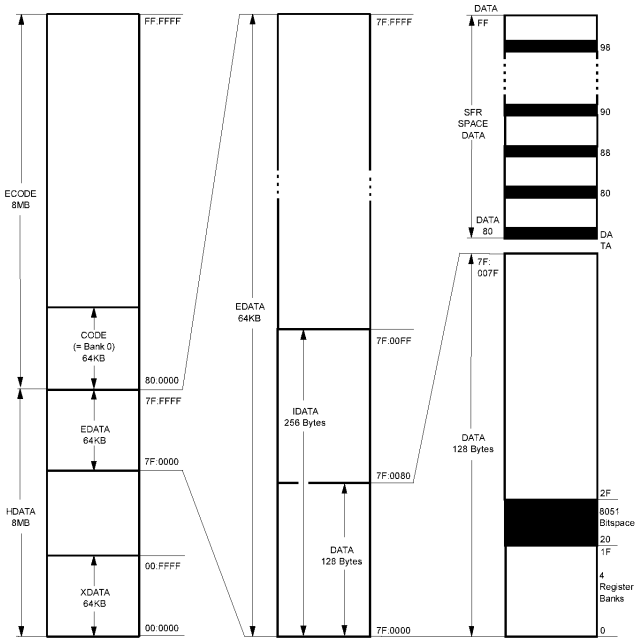
Структура памяти семейства MCS-51, MCS-251 и производных.



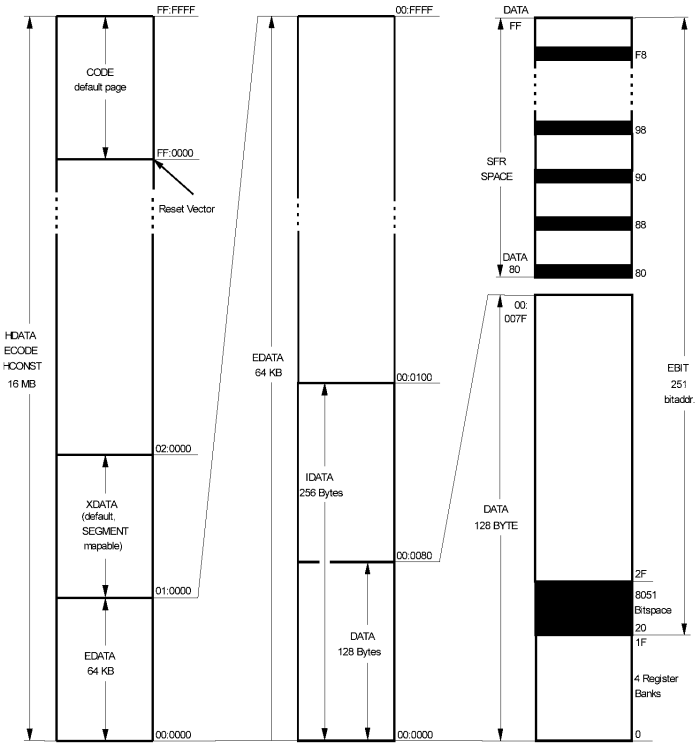
Классическая структура 8051. Рис 2



Расширения 8051. Рис 3



Структура 80C51MX. Рис 4



Структура 251. Рис 5

В распоряжении пользователя имеются 8 **регистров общего назначения R0–R7** одного из четырёх возможных банков. Банки регистров находятся в начале внутренней памяти данных и занимают 32 байта. 0-ой банк – адреса от

Блок регистров специальных функций

Символ	Наименование	Адрес
* ACC	Аккумулятор	0E0H
* B	Регистр-расширитель аккумулятора	0F0H
* PSW	Слово состояния программы	0D0H
SP	Регистр-указатель стека	81H
DPTR	Регистр-указатель данных (DPH)	83H
	(DPL)	82H
* P0	Порт 0	80H
* P1	Порт 1	90H
* P2	Порт 2	0A0H
* P3	Порт 3	0B0H
* IP	Регистр приоритетов	0B8H
* IE	Регистр маски прерываний	0A8H
TM0D	Регистр режима таймера/счетчика	89H
* TC0N	Регистр управления/статуса таймера	88H
TH0	Таймер 0 (старший байт)	8CH
TL0	Таймер 0 (младший байт)	8AH
TH1	Таймер 1 (старший байт)	8DH
TL1	Таймер 1 (младший байт)	8BH
* SCON	Регистр управления приемопередатчиком	98H
SBUF	Буфер приемопередатчика	99H
PCON	Регистр управления мощностью	87H

00h до 07h. 1-ый банк – адреса от 08h до 0Fh. 2-ой банк – адреса от 10h до 17h. 3-ий банк – адреса от 18h до 1Fh. Текущий рабочий банк регистров задаётся в слове состояния программы – PSW. Механизм переключения банков регистров упрощает обслуживание прерываний (своеобразная поддержка многозадачности).

Регистры SFR. Регистры с символическими именами IP, IE, TMOD, TCON, SCON и PCON и т.п. используются для фиксации и программного изменения управляющих бит и бит состояния схемы прерывания, таймера/счетчика, приемопередатчика последовательного порта и для управления мощностью электропитания у представителей семейства MCS-51.

Примечание. Регистры, имена которых отмечены знаком (*), допускают адресацию отдельных бит. (Все их адреса кратны 8. Младшие три бита адреса дают адрес бита в текущем байте.)

Битовый процессор.

Машинные инструкции семейства MCS-51. Система команд содержит 111 базовых команд 8-разрядного микроконтроллера, число которых увеличивается за счёт различных методов адресации до 255. Это классический набор инструкций, и он выполняется для всех представителей семейства. Производные МК семейства MCS-51 и Philips расширяют свою функциональность с помощью кода **A5h**.

Binary Mode	**x0**	**x1**	**x2**	**x3**	**x4**	**X5**	**x6 - x7**	**x8 - xF**
Source Mode	**x0**	**x1**	**x2**	**x3**	**x4**	**X5**	**A5x6-A5x7**	**A5x8-A5xF**
0x	NOP	AJMP adr11	LJMP adr16	RR A	INC A	INC dir	INC @Ri	INC Rn
1x	JBC bit,rel	ACALL adr11	LCALL adr16	RRC A	DEC A	DEC dir	DEC @Ri	DEC Rn
2x	JB bit,rel	AJMP adr11	RET	RL A	ADD A,#data	ADD A,dir	ADD A,@Ri	ADD A,Rn
3x	JNB bit,rel	ACALL adr11	RETI	RLC A	ADDC A,#data	ADDC A,dir	ADDC A,@Ri	ADDC A,Rn
4x	JC rel	AJMP adr11	ORL dir,A	ORL dir,#data	ORL A,#data	ORL A,dir	ORL A,@Ri	ORL A,Rn
5x	JNC rel	ACALL adr11	ANL dir,A	ANL dir,#data	ANL A,#data	ANL A,dir	ANL A,@Ri	ANL A,Rn
6x	JZ rel	AJMP adr11	XRL dir,A	XRL dir,#data	XRL A,#data	XRL A,dir	XRL A,@Ri	XRL A,Rn
7x	JNZ rel	ACALL adr11	ORL c,bit	JMP @A+DPTR	MOV A,#data	MOV dir,#data	MOV @Ri,#data	MOV Rn,#data
8x	SJMP rel	AJMP adr11	ANL C,bit	MOVC A,@A+PC	DIV AB	MOV dir,dir	MOV dir,@Ri	MOV dir,Rn
9x	MOV DPTR,#d16	ACALL adr11	MOV bit,c	MOVC A,@A+DPTR	SUBB A,#data	SUBB A,dir	SUBB A,@Ri	SUBB A,Rn
Ax	ORL C,/bit	AJMP adr11	MOV C,bit	INC DPTR	MUL AB	OPCODE PREFIX	MOV @Ri,dir	MOV Rn,dir
Bx	ANL C,/bit	ACALL adr11	CPL bit	CPL C	CJNE A,#d8,rel	CJNE A,dir,rel	CJNE @Ri,#d8,rel	CJNE Rn,#d8,rel
Cx	PUSH dir	AJMP adr11	CLR bit	CLR C	SWAP A	XCH A,dir	XCH A,@Ri	XCH A,Rn
Dx	POP dir	ACALL adr11	SETB bit	SETB C	DA A	DJNZ dir,rel	XCHD A,@Ri	DJNZ Rn,rel
Ex	MOVX A,@DPTR	AJMP adr11	MOVX A,@Ri		CLR A	MOV A,dir	MOV A,@Ri	MOV A,Rn

Fx	MOV @DPTR,A	ACALL adr11	MOVX @Ri,A	CPL A	MOV dir,A	MOV @Ri,A	MOV Rn,A

Расширения семейства MCS-51 являются надмножествами указанного семейства. Присутствует совместимость вверх на уровне кодов. Исходный текст компилируется любым компилятором. Высший представитель исполняет программу более производительнее, чем исходный 8051.

MCS-251 работает в двух режимах: **бинарный режим** 8051 и родной **исходный режим**. В бинарном режиме дополнительные инструкции идут с префиксом A5h. В исходном режиме действует расширенный набор инструкций, а базовый набор инструкций 8051 используется с префиксом A5h. Какой набор инструкций (базовый или расширенный) в программе преобладает, тот режим лучше и использовать для сокращения кода программы.

Дополнительные инструкции семейства MCS-251.

Binary Mode	**A5x8**	**A5x9**	**A5xA**	**A5xB**	**A5xC**	**A5xD**	**A5xE**	**A5xF**
Source Mode	**x8**	**x9**	**xA**	**xB**	**xC**	**xD**	**xE**	**xF**
0	JSLE rel	MOV Rm, @WRj+dis	MOVZ WRj,Rm	INC Rm/WRj/ Drk,#short MOV reg,ind			SRA reg	
1	JSG rel	MOV @WRj+dis,Rm	MOVS WRj,Rm	DEC Rm/WRj/ Drk,#short MOV ind,reg			SRL reg	
2	JLE rel	MOV Rm,@DRk+dis			ADD Rm,Rm	ADD WRj,WRj	ADD reg,op2	ADD DRk,DRk
3	JG rel	MOV @DRk+dis,Rm					SLL reg	
4	JSL rel	MOV WRj,@WRj+dis			ORL Rm,Rm	ORL WRj,WRj	ORL reg,op2	
5	JSGE rel	MOV @WRj+dis,WRj			ANL Rm,Rm	ANL WRj,WRj	ANL reg,op2	
6	JE rel	MOV WRj,@DRk+dis			XRL Rm,Rm	XRL WRj,WRj	XRL reg,op2	
7	JNE rel	MOV @Drk+dis,WRj	MOV op1,reg		MOV Rm,Rm	MOV WRj,WRj	MOV reg,op2	MOV DRk,DRk
8		LJMP @WRj EJMP @DRk	EJMP addr24		DIV Rm,Rm	DIV WRj,WRj		
9		LCALL @WR ECALL @DRk	ECALL addr24		SUB Rm,Rm	SUB WRj,WRj	SUB reg,op2	SUB DRk,DRk
A		BIT instructions	ERET		MUL Rm,Rm	MUL WRj,WRj		
B		TRAP			CMP Rm,Rm	CMP WRj,WRj	CMP reg,op2	CMP DRk,DRk
C			PUSH op1					
D			POP op1					
E								
F								

Дополнительные инструкции Philips 80C51MX (через префикс A5). Включает набор инструкций для адресации памяти в пределах единого 16 Мбайтного адресного пространства CODE и XDATA и для доступа к дополнительным SFR регистрам. Указатель стека 16-разрядный для адресации данных в памяти EDATA размером до 64 Кбайт. Указатель PR0 составлен из регистров R1, R2 и R3, а указатель PR1 – из регистров R5, R6 и R7.

	A5 x0	**A5 x1**	**A5 x2**	**A5 x3**	**A5 x4**	**A5 x5**	**A5 x6 - A5 x7**	**A5 x8 – A5 xF**	**A5 x8 – A5 xF**
A5 0x			EJMP adr23			INC esfr			
A5 1x	JBC es-bit,rel		ECALL adr23			DEC esfr			
A5 2x	JB esbit,rel					ADD A,esfr			
A5 3x	JNB es-bit,rel					ADDC A,esfr			
A5 4x			ORL esfr,A	ORL esfr,#data		ORL A,esfr		EMOV A,@PR0+d2	EMOV A,@PR1+d2

A5 5x			ANL esfr,A	ANL esfr,#data		ANL A,esfr		EMOV @PR0+d2,A	EMOV @PR1+d2,A
A5 6x			XRL esfr,A	XRL esfr,#data		XRL A,esfr		ADD PR0,#data2	ADD PR1,#data2
A5 7x			ORL c,esbit	EJMP @A+EPTR		MOV dir,#data			
A5 8x			ANL C,esbit			MOV esfr,esfr	MOV esfr,@Ri	MOV esfr,Rn	
A5 9x	MOV EPTR,#d23		MOV esbit,c	MOVC A,@A+EPTR		SUBB A,esfr			
A5 Ax	ORL C,/esbit		MOV C,esbit	INC EPTR			MOV @Ri,esfr	MOV Rn,esfr	
A5 Bx	ANL C,/esbit		CPL es-bit			CJNE A,esfr,rel			
A5 Cx	PUSH esfr		CLR es-bit			XCH A,esfr			
A5 Dx	POP esfr		SETB esbit			DJNZ esfr,rel			
A5 Ex	MOVX A,@EPTR					MOV A,esfr			
A5 Fx	MOV @EPTR,A					MOV esfr,A			

Архитектура семейства **16-разрядных** микроконтроллеров **Philips XA** в системе команд не содержит подмножество семейства 80C51. Программы совместимы на уровне исходного текста ассемблера. Точнее, требуется трансляция исходного текста программы систем 80C51 в исходный код 80C51XA. (Аналогия i8080 <–> i8086)

II. ЭТАПЫ ПРОЕКТИРОВАНИЯ МИКРОПРОЦЕССОРНОГО КОНТРОЛЛЕРА.

МК-система управления. Типовая структура МК-системы управления состоит из 1) объекта управления, 2) микроконтроллера и 3) аппаратуры их взаимной связи (для системы сбора информации и исполнительных устройств). В количественном измерении эти компоненты могут быть представлены в различных соотношениях.

Микроконтроллер путем периодического опроса через порты ввода осведомительных слов (ОС) генерирует в соответствии с алгоритмом управления последовательности управляющих слов (УС) через порты вывода. Осведомительные слова это сигналы состояния объекта (СС), сформированные датчиками объекта управления, и флаги. Выходные сигналы датчиков вследствие их различной физической природы могут потребовать промежуточного преобразования на аналого-цифровых преобразователях (АЦП) или на схемах формирователей сигналов (ФС), которые чаще всего выполняют функции гальванической развязки и формирования уровней двоичных сигналов стандарта ТТЛ.

Микроконтроллер с требуемой периодичностью обновляет управляющие слова на своих выходных портах. Некоторая часть управляющего слова интерпретируется как совокупность **прямых двоичных сигналов управления** (СУ), которые через схемы формирователей сигналов (усилители мощности, реле, оптроны и т.п.) поступают на исполнительные механизмы (ИМ) и устройства индикации. Другая часть управляющего слова представляет собой **упакованные двоичные коды**, которые через цифро-аналоговые преобразователи (ЦАП) воздействуют на исполнительные механизмы аналогового типа. Если объект управления использует цифровые датчики и цифровые исполнительные механизмы, то наличие ЦАП и АЦП в системе необязательно.

Законы функционирования типовой системы управления всецело определяются прикладной программой, размещаемой в резидентной памяти программ МК. Иными словами **специализация контроллера** типовой структуры на решение задачи управления конкретным объектом осуществляется путем разработки 1) прикладных программ МК и 2) аппаратуры связи МК с датчиками и исполнительными механизмами объекта.

1. ЦИКЛ РАЗРАБОТКИ.

Весь цикл разработки контроллеров рассматривается как последовательность **трех фаз проектирования**:

1) анализа задачи и выбора (и/или разработки) аппаратурных средств контроллера;

2) разработки прикладного программного обеспечения;

3) комплексирования аппаратурных средств и программного обеспечения в прототипе контроллера и его отладки (автономно и комплексно).

Аппаратные средства. МК представляет собой логический автомат с высокой степенью детерминированности, который допускает очень немного вариантов его системного включения. Типовой состав аппаратурных средств ядра любой МК-системы включает МК, ППЗУ, ОЗУ, интерфейсные БИС, схемы синхронизации и системного управления. Два пути**: [1]** строить котроллер из указанных компонентов изначально и самостоятельно; **[2]** типовой состав аппаратуры оформляется конструктивно в виде **одноплатных универсальных программируемых контроллеров**, которые предназначены для встраивания в контур управления объектом или процессом. 0) Такой контроллер как БИС можно установить на своей плате. 1) На печатной плате такой МК-системы могут быть представлены гнезда для установки БИС пользователя. 2) На некоторых моделях таких плат имеется еще и так называемое

"монтажное поле пользователя", на котором он имеет возможность монтировать свои специфические схемы, такие как оптронные развязки, АЦП, ЦАП, реле и т.п. 3) Кроме того, на плате МК-системы может быть размещен источник электропитания. 4) На плате такой МК-системы могут быть реализованы интерфейсы аппаратуры связи.

Появление однокристальных МК иллюстрирует тот факт, что все более сложные функционально насыщенные части аппаратурных средств контроллеров в процессе интегрализации переходят из разряда подсистем в разряд **комплектующих изделий**. Так как эти комплектующие изделия являются сложно организованными приборами, функционирующими под управлением программ, то удельный вес прикладного программного обеспечения МК-систем имеет устойчивую тенденцию к увеличению, а удельный вес аппаратурных средств – к снижению.

2. РАЗРАБОТКА ПРИКЛАДНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ.

Ориентация на разработку прикладных программ для МК-систем силамм программирующих профессионалов-разработчиков (а не профессиональных программистов) получает распространение потому, что в условиях быстро дешевеющей памяти изменились стиль и технология разработки программ. Экономят теперь уже не память МК-системы, а время разработчика программного обеспечения, т.е. сокращают сроки разработки изделия.

Фаза разработки программного обеспечения, т.е. фаза получения прикладных программ, в свою очередь, разбивается на два существенно различных этапа, которые могут проводиться итерационно:

1) "от постановки задачи к исходной программе";

2) "от исходной программы к исполнительному коду".

Проведение нескольких итераций этих этапов друг за другом скорее приводит к успеху.

2-ой этап разработки – самый лёгкий. Этап "от исходной программы к исполнительному коду" представлен на рис. 6 и имеет целью получение машинных кодов прикладных программ, работающих в МК. Этот этап разработки прикладного программного обеспечения легко поддается формализации и поддержан всей мощью системного программного обеспечения МК, направленного на автоматизацию процесса получения прикладных программ. В состав средств системного программного обеспечения входят трансляторы с различных алгоритмических языков высокого уровня, ассемблеры, редакторы текстов, программы-отладчики, программы-документаторы и т.д. Наличие всех этих системных средств придает инженерной работе на этом этапе проектирования контроллеров характер ремесла, а не инженерного творчества. Так как в конечном изделии (контроллере) имеются только "голый" МК и средства его сопряжения с объектом, то выполнять отладку разрабатываемого прикладного программного обеспечения на нем невозможно (из-за отсутствия средств ввода, вывода, ОЗУ большой емкости и операционной системы), и, следовательно, разработчик вынужден обращаться к средствам вычислительной техники для выполнения всех формализуемых стадий разработки: трансляции, редактирования, отладки, загрузки объектных кодов в программируемую постоянную память МК. Попутно отметим, что системные средства автоматизации разработки прикладных программ МК на этапе "от исходной программы к исполнительному коду" широко распространены и существуют как в среде операционных систем ряда специализированных микро-ЭВМ, так и присутствуют в операционных системах Windows персональных компьютеров. Производители HI-TECH Software, IAR Systems, Keil Elektronik GmbH, TASKING, RAISONANCE S. A. и др.

Получение исполнительного кода

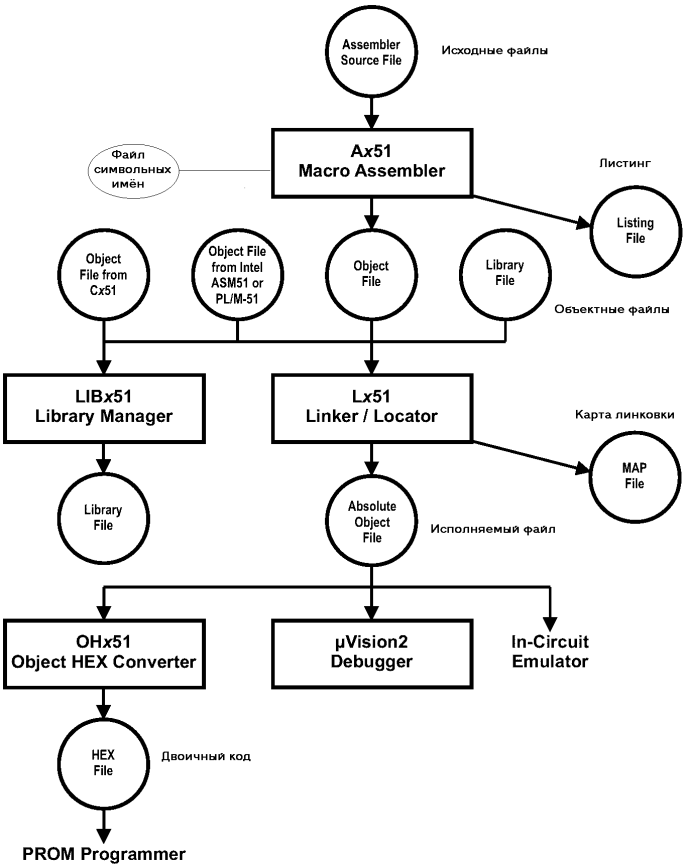


Рис 6.

1-ый этап разработки. Совсем по-другому выглядит инженерный труд на этапе разработки программного обеспечения "от постановки задачи к исходной программе", так как он практически не поддается формализации и, следовательно, не может быть автоматизирован. Проектная работа здесь носит творческий характер, изобилует решениями, имеющими "волевую" или "вкусовую" окраску, и решениями, продиктованными конъюнктурными соображениями. В силу перечисленных обстоятельств именно на этапе проектирования "от постановки задачи к исходной программе" разработчик сталкивается с наибольшим количеством трудностей. Качество получаемого прикладного программного обеспечения контроллера всецело зависит от уровня проектных решений, принятых на этапе разработки "от постановки задачи к исходной программе". Уровень проектных решений в свою очередь из-за отсутствия теории проектирования программируемых контроллеров определяется только опытом, квалификацией и интуицией разработчика. Систематический подход к процессу разработки прикладных программ для контроллеров обеспечивает достижение хороших результатов даже начинающими разработчиками.

3. ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ МИКРОКОНТРОЛЛЕРНЫХ СИСТЕМ

3.1. Методика.

Если задача на разработку прикладной программы для микроконтроллера поставлена, то для получения текста исходной программы необходимо выполнить ряд последовательных действий.

1. Подробно описать задачу.
2. Проанализировать задачу.
3. Выполнить инженерную интерпретацию задачи, желательно с привлечением того или иного аппарата формализации (граф автомата, сети Петри, матрицы состояний и связности и т.п.).
4. Разработать общую схему алгоритма работы контроллера.
5. Разработать детализированные схемы отдельных процедур, выделенных на основе модульного принципа составления программ.
6. Детально проработать интерфейс контроллера и внести исправления в общую и детализированные схемы алгоритмов.
7. Распределить рабочие регистры и память.
8. Сформировать текст исходной программы.

В результате работы по трем первым пунктам данного перечня получают так называемую **функциональную спецификацию прикладной программы**, в которой основное внимание уделяется детализации способов формирования входной и выходной информации.

На языке схем алгоритмов разработчик описывает метод, выбранный им для решения поставленной задачи. Довольно часто бывает, что одна и та же задача может быть решена различными методами. Способ решения задачи, выбранный на этапе её инженерной интерпретации, на основе которого формируется схема, определяет не только качество разрабатываемой прикладной программы, но и качественные показатели конечного изделия.

Алгоритм есть точно определенная процедура, предписывающая контроллеру однозначно определенные действия по преобразованию исходных данных в обработанные выходные данные. Поэтому разработка схемы требует предельной точности и однозначности используемой атрибутики: символических имен переменных, констант, подпрограмм (модулей), символических адресов таблиц, портов ввода вывода и т.п. Основное внимание при разработке следует уделить тому разделу функциональной спецификации прикладной программы, в котором приводится описание аппаратуры сопряжения с объектом управления. Это описание должно быть детализировано вплоть до электрических и временных характеристик каждого входного и выходного сигнала или устройства.

Успех разработки прикладной программы заключается в использовании метода декомпозиции, при котором вся задача последовательно разделяется на меньшие функциональные модули. Каждый из модулей можно анализировать, разрабатывать и отлаживать отдельно от других. При выполнении прикладной программы в микроконтроллере управление без всяких двусмысленностей передается от одного функционального модуля к другому. Схема связности этих функциональных модулей, каждый из которых реализует некоторую процедуру, образует общую схему алгоритма прикладной программы. Язык графических образов схемы алгоритма можно использовать на любом уровне детализации описания модулей вплоть до того, что каждому оператору схемы будет соответствовать единственная команда микроконтроллера.

Структурное программирование есть процесс построения прикладной программы из набора программных модулей, каждый из которых реализует определенную процедуру обработки данных. Программные модули должны иметь только одну точку входа и одну точку выхода. Только в этом случае отдельные модули можно разрабатывать и отлаживать независимо, а затем объединять в законченную прикладную программу с минимальными проблемами их взаимосвязи.

Источником подавляющего большинства ошибок программирования является использование модулей, имеющих один вход и несколько выходов. При необходимости организации множественных ветвлений в программе декомпозицию задачи выполняют таким образом, чтобы каждый функциональный модуль имел только один вход и один выход. Для этого условные операторы (имеющие два выхода) или включают внутрь модуля, объединяя их с операторами обработки, или выносят в систему межмодульных связей, формируя тем самым схему алгоритма более высокого ранга.

В международном стандарте на программный продукт **HIPO** (Hierarchy-Input-Process-Output) (“хай-по”) декларируется аналогичный подход к разработке прикладных программ.

Разработка схемы алгоритма функционального модуля программы имеет **ярко выраженный итеративный характер**, т.е. требует многократных проб, прежде чем возникает уверенность, что алгоритм реализации процедуры правильный и завершённый. Вне зависимости от функционального назначения процедуры при разработке её схемы необходимо придерживаться следующей очередности работы.

1. Определить, что должен делать модуль.
2. Определить способы получения модулем исходных данных (от датчиков через порты ввода, из таблиц в памяти или через рабочие регистры).
3. Определить необходимость какой-либо предварительной обработки введенных исходных данных (маскирование, сдвиг, масштабирование, перекодировка).
4. Определить метод преобразования входных данных в требуемые выходные. Используя операторы процедур и условные операторы принятия решения, отобразить на языке схемы алгоритма выбранный метод.

5. Определить способы выдачи из модуля обработанных данных (передать в память, в вызывавшую программу или в порты вывода).

6. Определить необходимость какой-либо вторичной обработки выводимых данных (изменение формата, перекодирование, масштабирование, маскирование).

7. Вернуться к пункту 1 настоящего перечня и проанализировать полученный результат. Выполнить итеративную корректировку схемы алгоритма с целью сделать её простой, логичной, стройной и обладающей чётким графическим образом.

8. Проверить работоспособность алгоритма на бумаге путем подстановки в него действительных данных. Убедиться в его сходимости и результативности.

9. Рассмотреть предельные случаи и попытаться определить граничные значения информационных объектов, с которыми оперирует алгоритм, за пределами которых он теряет свойства конечности, сходимости или результативности. Особое внимание при этом следует уделить анализу возможных ситуаций переполнения разрядной сетки, изменения знака результата операции, деления на переменную, которая может принять нулевое значение.

10. Провести мысленный эксперимент по определению работоспособности алгоритма в реальном масштабе времени, когда стохастические события, происходящие в объекте управления, могут оказать влияние на работу алгоритма. При этом самому тщательному анализу следует подвергнуть реакцию алгоритма на возможные прерывания с целью определения критических операторов, которые необходимо защитить от прерываний. Кроме того, в ходе этого мысленного эксперимента следует проанализировать логику алгоритма с целью определения таких последовательностей операторов, при выполнении которых микроконтроллер может “не заметить” кратковременных событий в объекте управления. При обнаружении таких ситуаций в логику следует внести коррективы.

Практика разработки программного обеспечения показала, что последовательное использование описанной поэтапной процедуры, составляющей основу метода структурного программирования, позволяет уверенно получать работоспособные прикладные программы.

Преобразование разработанной схемы алгоритма в исходный текст программы дело несложное. Но прежде чем приступить к написанию программы необходимо специфицировать память и выбрать язык программирования.

Спецификация памяти и рабочих регистров заключается в определении адреса первой команды прикладной программы, действительных начальных адресов стека, таблиц данных, буферных зон передачи параметров между процедурами, подпрограмм обслуживания прерываний и т.д. При этом следует помнить, что в микроконтроллерах память программ и память данных физически и логически разделены.

Диапазон **языков** написания исходного текста прикладной программы простирается от машинного кода до почти естественного языка. В машинном коде или на языке ассемблера программировать труднее, чем на алгоритмическом языке высокого уровня, но зато получается более короткий код программы, требуется меньшая ёмкость памяти программы и выполняется такая программа быстрее.

Объектные коды, полученные путем трансляции исходных программ, написанных на алгоритмическом языке высокого уровня, занимают в памяти больше места и требуют большего времени на исполнение. Выбор языковых средств составления исходных программ в каждом конкретном случае зависит от характеристик прикладной задачи, опыта программиста и допустимых затрат на разработку.

Сверстать программу на языке высокого уровня значительно проще. Получив работающий вариант программы, его можно уже оптимизировать на ассемблере. (5-ый признак сложной системы).

Огромное большинство прикладных задач управления объектами вследствие того, что они должны решаться в реальном времени, предъявляет столь высокие требования по быстродействию, что для их решения основным языковым средством написания прикладных программ еще долгие годы будет оставаться язык ассемблера. Это положение о преимущественном использовании языка ассемблера подкрепляется и тем обстоятельством, что однокристальные микроконтроллеры имеют ограниченный объем резидентной памяти программ и, следовательно, критичны к длине прикладных программ.

3.2. Процедуры и подпрограммы (общее представление)

При разработке микроконтроллерных систем могут быть использованы два способа организации прикладных программ: **монолитный** и **модульный**. При первом способе вся прикладная программа разрабатывается как единое целое. Пригоден для небольших программ. При втором она строится из отдельных программных блоков, каждый из которых реализует некоторую процедуру обработки данных или управления. Взаимосвязь блоков определяется разработчиком при монтаже из этих блоков законченной прикладной программы.

Отдельные фрагменты прикладной программы могут быть получены в виде линейной последовательности блоков, другие (многократно используемые) обычно оформляются в виде **подпрограмм**, к которым прикладная программа, называемая основной, имеет возможность обратиться по мере необходимости. Подпрограмма должна обладать следующими свойствами: 1) выполнять законченную процедуру обработки данных, 2) иметь только один вход и один выход и 3) не обладать эффектом последействия, при котором текущее выполнение подпрограммы оказывало бы влияние на её последующие выполнения.

Вызов подпрограммы. Обращение к подпрограмме осуществляется по команде вызова **CALL MARK**, где MARK - символическое имя процедуры. Имя процедуры используется в качестве метки, отмечающей одну из команд (чаще всего первую) подпрограммы. Для Intel 8051 мнемоническое значение CALL является обобщенным и транслируется в одну из команд **ACALL** или **LCALL** в зависимости от адресного расстояния вызываемой подпрограммы.

По команде CALL в стеке сохраняется значение счётчика команд, и возврат из подпрограммы осуществляется в

то место основной программы, откуда был осуществлен вызов (к команде основной программы, следующей за командой CALL). Для этого любая подпрограмма должна заканчиваться командой возврата **RET**, осуществляющей восстановление содержимого программного счётчика из стека.

Достаточно часто возникает необходимость такой организации вычислительного процесса, при которой подпрограмма вызывает другую подпрограмму, та в свою очередь вызывает следующую и т.д. Этот процесс называется **вложением подпрограмм**. Число подпрограмм, которые могут быть вызваны таким образом (глубина вложенности подпрограмм), ограничивается только ёмкостью стека.

Сохранение параметров основной программы. Иногда при обращении к подпрограмме возникает необходимость сохранить не только адрес возврата в основную программу, но и содержимое отдельных рабочих регистров. Удобным способом для этого является **переключение банка регистров**. Например, если основная программа использует банк регистров 0, то подпрограмма может использовать банк регистров 1. Однако переключение банка регистров не обеспечивает сохранение содержимого аккумулятора, что приводит к необходимости создавать в одном из рабочих регистров или в памяти копию аккумулятора.

Передача параметров. Для успешной работы любой подпрограммы необходимо однозначно определить способ передачи в неё исходных данных и способ вывода результата её работы. Подпрограмма, которой требуется дополнительная информация в виде параметров её настройки или операндов, называется **параметризуемой.**

Получили распространение четыре способа передачи параметров: через память, через регистры общего назначения, через регистр признаков и через стек.

При передаче входных параметров **через память** основная программа обязательно содержит команды загрузки некоторых ячеек памяти, а подпрограмма - команды считывания из этих ячеек. При передаче входных параметров подпрограмма должна загрузить некоторые ячейки памяти, а основная программа - считать.

Передача параметров **через регистры** осуществляется аналогичным образом.

Третий способ передачи параметров - **через регистр признаков** - удобно использовать при передаче выходных параметров (например, в подпрограммах сравнения чисел). В этом случае подпрограмма должна установить (или сбросить) соответствующие признаки, а основная программа - проанализировать их значение. Intel 8051 обладает большими возможностями для передачи параметров через признаки. В нём имеется 128 флагов пользователя, доступных для модификации и анализа.

Способ передачи **через стек** позволяет использовать в качестве параметра содержимое счётчика команд.

Использование процедур, оформленных в виде подпрограмм, при разработке программного обеспечения имеет ряд достоинств. **1)** Прежде всего относительно простые модули, выделенные из сложной программы, могут программироваться несколькими разработчиками с целью сокращения времени проектирования. **2)** Еще более важным является то, что любая подпрограмма допускает автономную отладку. Это, как правило, многократно сокращает время отладки всего прикладного программного обеспечения. **3)** И, наконец, механизм использования подпрограмм уменьшает длину прикладной программы, что имеет своим следствием уменьшение требующейся ёмкости памяти программ.

4) Существенным является и то обстоятельство, что отлаженные процедуры организуются разработчиками в библиотеки параметризуемых подпрограмм и могут быть многократно использованы в проектной работе. Библиотека подпрограмм должна строиться на основе соглашения о едином способе обмена параметрами.

3.3. Простой пример

Часть задачи приведена на рис. 7: на простой запрос в системе сгенерировать ответ. В качестве запроса выступает положительный импульс, поступающий по линии REQU. Ответом является периодическая последовательность ANSW положительных импульсов заданной скважности. (Здесь скважность точно отрабатывать не будем.)

Фрагмент системы отклика

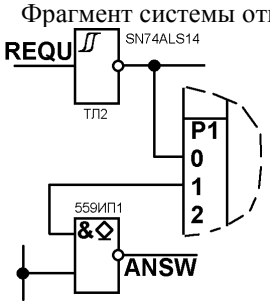


Рис. 7.

Для устранения помех запрос принимается на триггер Шмитта. Поэтому в программе ловить необходимо отрицательный импульс на входе 0 порта P1. Ответ генерируем на выходе 1 порта P1 и передаём его в линию через мощный передатчик с открытым коллектором (ОК), а нагрузка на приёмном конце. Следовательно программно генерируем последовательность отрицательных импульсов. Период задаём с помощью таймера T0. В силу сравнительно большого периода (например 256 мкс) для исключения непроизводительных расходов процессорного времени задействуем систему прерываний от таймера T0.

Решение приведём на машинном языке, языке Ассемблера и на языке высокого уровня С. Программирование на машинном языке осуществляется в отладчике-симуляторе в окне дизассемблера. Есть два входа в встроенный ассемблер: команды отладчика и локальный диалог. Рабочие окна отладчика приведены на рисунках 8 и 9.

Окно вывода отладчика

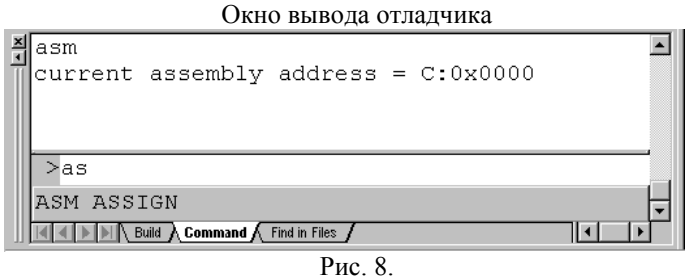


Рис. 8.

Встроенный Ассемблер отладчика

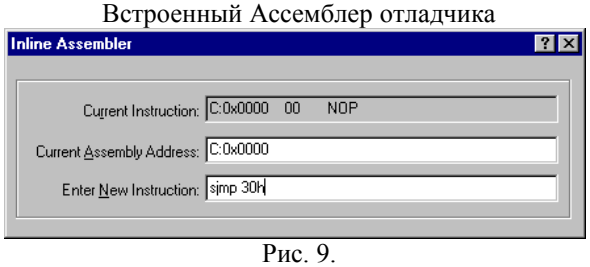


Рис. 9.

Результирующая программа наблюдается в окне дизассемблера, которое включается из меню **View – Disassembly Window.** Диалог встроенного ассемблера вызывается из локального меню (правая кнопка мыши).

Машинный язык.

```
// start комментарии допустимы в отладчике
asm 0
asm sjmp 30h

// main только генерировать периодический ответ
asm 30h
asm orl tmod,#1
asm mov tl0,#0
asm mov th0,#0ffh
asm orl ie,#82h
asm setb tr0
asm jmp 3eh

// interrupt T0
asm 0bh
asm push psw
asm mov psw,#8
asm push acc
asm clr tr0
asm mov a,tl0
asm add a,#7
asm mov tl0,a
asm mov a,th0
asm addc a,#0ffh
asm mov th0,a
asm setb tr0
asm clr p1.1
asm setb p1.1
asm pop acc
asm pop psw
asm reti

e char c:0=80h,2eh
e char c:3eh=80h,0feh
```

Язык высокого уровня С.

```
#pragma CODE

// определение регистров SFR
#include <reg51.h>
// глобальные константы и переменные
//const unsigned int clock = 0xFF00; // переменная!!
#define clock 0xFF00
sbit requ = P1^0;     // запрос
sbit answ = P1^1;     // ответ

//static void int_timer0( void ) interrupt 1 {
static void int_timer0( void ) interrupt 1 using 1 {
  TR0 = 0;
  TL0 += clock + 7; // внимание 0!
  if( CY ) TH0++;
  TH0 += ( clock + 7 )>> 8;
  TR0 = 1;
  answ = 0;
  answ = 1;
}

//*************** main ******************
void main ( void ) {
// начальная работа
// . . . . .
// интересующий нас фрагмент
  TMOD |= 1;
  TL0 = ( unsigned char )clock;
  TH0 = ( unsigned char )( clock >> 8 );
  IE = 0x82; // включить систему прерываний
  while( requ ) ;
  TR0 = 1; // запустить таймер
  while( 1 ); // ждать
}
```

На машинном языке следует задать поиск запроса, длительность импульса ответа и поддержку отладки, что демонстрируется в следующих трёх столбцах.

// main program с **поиском запроса**

```
asm 30h
asm orl tmod,#1
asm mov tl0,#0
asm mov th0,#0ffh
asm orl ie,#82h
//asm jb p1.0,3ch
asm nop
asm nop
asm nop
asm setb tr0
asm jmp 41h

e char c:41h=80h,0feh
```

Два варианта **длины импульсов**

```
. . . . . .
asm clr p1.1
asm nop
. . . . . .
asm mov r7,#5
asm djnz r7,22h
asm setb p1.1
. . . . . .
```

// button start **кнопка для отладки**

```
//define button "start","p1.0=0"
define button "start","port1 &= 0xFE"

// start
asm 0
asm jmp 30h
. . . . . .
```

Программа генерации отклика **на языке Ассемблера** в соответствии с принципом модульности состоит из трёх самостоятельных модулей:

Модуль 1

```
; головной модуль для генератора с timer0
$NOMOD51
clock   EQU     0FF00H
        PUBLIC  clock

TMOD    DATA    089H
TL0     DATA    08AH
TH0     DATA    08CH
```

```
IE      DATA    0A8H
TR0     BIT     088H.4
requ    BIT     090H.0

        PUBLIC  main
        NAME    MAINPRG
;=======================
; сегмент: головная процедура
MYMAIN  SEGMENT CODE
        RSEG    MYMAIN
;********************** main ****************************
main:
        ORL     TMOD,#01H
        MOV     TL0,#LOW( clock )
        MOV     TH0,#HIGH( clock )
        MOV     IE,#082H
WaitRequ:
        JB      requ,WaitRequ
        SETB    TR0
Loop:
        SJMP    $
        END
```

Модуль 2

```
; модуль обработчика прерываний для генератора с timer0
$NOMOD51

TL0     DATA    08AH
TH0     DATA    08CH
PSW     DATA    0D0H
ACC     DATA    0E0H
TR0     BIT     088H.4
answ    BIT     090H.1

        EXTRN   NUMBER ( clock )
        NAME    INTERRUPTT0PRG
;        PUBLIC  int_timer0
        USING   0
;=======================
; сегмент: обработчик прерывания
        CSEG    AT      0BH
int_timer0:
        PUSH    ACC
        PUSH    PSW
        CLR     TR0
        MOV     A,TL0
        ADD     A,#LOW( clock + 7 )
        MOV     TL0,A
        MOV     A,TH0
        ADDC    A,#HIGH( clock + 7 )
        MOV     TH0,A
        SETB    TR0
; выдать ответ
        CLR     answ
        SETB    answ
; завершение
        POP     PSW
        POP     ACC
        RETI
        END
```

Модуль 3

```
; стартовый модуль для генератора с timer0
$NOMOD51
        EXTRN   CODE ( main )
        PUBLIC  stack
```

```
        NAME    STARTT0PRG
;=======================
; сегмент: стартовый код
        CSEG    AT      0
        SJMP    main
; без очистки внутренней памяти!
;=======================
; сегмент: рабочий стек
MYSTACK SEGMENT IDATA
        RSEG    MYSTACK
stack:
        DS      5
        END
```

4. ПРИМЕР. СИСТЕМА ДОЗИРОВАННОГО НАГРЕВА БИООБЪЕКТОВ.

Система включает ВЧ установку общего нагрева в объёме, систему водяного нагрева и охлаждения, систему воздушного нагрева и охлаждения, контроллер термометрии, контроллер ВЧ мощности, управляющую ЭВМ и т.д. – целый ряд мелких подсистем и устройств.

Решаем часть задачи управления – **контроллер термометрии** (КТ). КТ должен – 1). измерять температуру с помощью полупроводниковых датчиков не менее, чем с 8 точек съёма, с точностью не хуже 0.05 град. Цельсия; 2). управлять нагревом воды для водяной системы; 3). управлять нагревом воздуха для воздушной системы; 4). поддерживать смену датчиков, т.е. обеспечивать калибровку с заданной точностью; 5). выбирать датчики для конкретного сеанса, мобильно переключать их; 6). взаимодействовать по интерфейсу RS-232 с управляющей ЭВМ, поддерживая помехозащищённый алгоритм со скоростью передачи не ниже 57600 бод; 7). разные скорости измерения и передачи температурных значений; 8). менять алгоритм взаимодействия с управляющей ЭВМ; и т.д. и т.п..

Состав контроллера КТ. **1)** МК Intel 80С31 – небольшая резидентная память данных, нет резидентной памяти программм, поддерживает интерфейс RS-232 [6] и [7]. **2)** Два блока памяти программ [8]. Первый блок содержит монитор загрузки в ПЗУ. Второй блок содержит собственно управляющую программу в ОЗУ. Размер блоков 2 Кб. Предусмотрено расширение до 8 Кб для каждого. [8]. **3)** Один блок памяти данных ОЗУ 2 Кб, расширяем до 8 Кб. Это резерв системы для будущих расширений [8]. Можно ввести дополнительную статистическую обработку для повышения точности измерения температуры с возможным изменением метода измерения температуры вплоть до оптических. **4)** Аналоговый коммутатор на 8 каналов [1] и [5]. Для управления коммутатором используется регистр номера датчика (РНД). **5)** Измерительный усилитель [4] и [5]. Должен обеспечивать балансировку для всех сменных датчиков. Напряжение балансировки получается с ЦАП. Для управления ЦАП нужен регистр кода баланса (РКБ) и РНД. **6)** АЦП 12-разрядный, типа AD1674, поддерживающий интерфейс с 8- и 16-разрядными микропроцессорами. Результат преобразования выдаёт 1 или 2 байтами для регулирования скорости пересылки и точности измерения температуры [7]. **7)** Регистр управления нагревом воды и воздуха (РНВВ) [2] и [3]. **8)** Устройство управления с дешифратором адреса внешней памяти через 8 Кб. Все регистры отображены на память (XDATA). И т.д… средства гальванической развязки на оптронах, система автономного питания.

Структурная схема контроллера

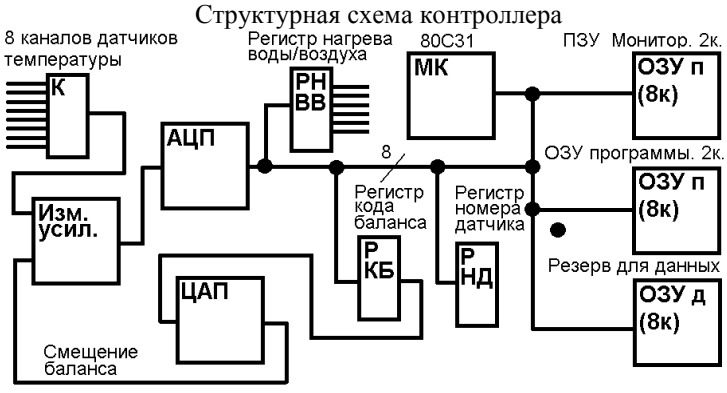


Рис. 10.

Необходимо разработать **систему команд управления** для помехоустойчивого взаимодействия с управляющей ЭВМ (протокол обмена). На языке Си сверстать программу, а на ассемблере оптимизировать.

Структурная схема контроллера температуры приведена на рис. 10.

Память монитора предназначена для загрузки программы в память программы.

Пример программы 1.

```
;$CASE          ;для AX51 A251

$SET (TYPERPOGR = 1)
$DATE (01/Jan/00)
$DEBUG
;$ERRORPRINT (IZM.ERR)
$ERRORPRINT    ;на консоль
$PAGELENGTH (5000)
$NOMOD51
```

```
;$INCDIR (H:\Keil6.1\C51\ASM)
$INCLUDE ( reg51.inc )          ;#include <reg51.inc>
$TITLE (IZMERITEL TEMPERATURY Program N2)
$GEN

        NAME    IzmeritelTemperatury
;       IZMTEMP.ASM
; Загружаемая часть монитора измерителя температуры,входящего в состав
; установки для проведения сеансов общей гипертермии (80C31)
;
        DSEG    AT 0H
DATAIN  DATA    0H              ;начало сегмента
;----------------------------------------
;       Адреса внутренних переменных
;
FLAGDT  DATA    20H             ;регистр флагов подключ. датчиков температуры
KREGUN  DATA    21H             ;копия регистра управления нагревом
KODUPR  DATA    22H             ;код управления, поступивший от ПЭВМ
FLAGUPR DATA    28H             ;регистр флагов управления
RREGFL  DATA    40H             ;рабочий регистр флагов датчиков
NOMDAT  DATA    41H             ;номер датчика температуры
MPFLAG  DATA    42H             ;местоположение устанав. или сбрасыв. флага

        BSEG    AT 20H
BITDAT  BIT     0H              ;начало сегмента
;----------------------------------------

;       Символьные имена отдельных бит
;
FLREKT  BIT     FLAGDT.0        ;флаг ректальн. датчика ("1"-дат. подключен)
FLKOGN  BIT     FLAGDT.1        ;флаг кожного датчика
FLVOD   BIT     FLAGDT.2        ;флаг водяного датчика
NAGRVOD BIT     KREGUN.0        ;включение нагрева воды ("1"-включен)
OHLVOD  BIT     KREGUN.1        ;включение охлаждения воды
NAGRVOZ BIT     KREGUN.2        ;включение нагрева воздуха
SOSTDAT BIT     KODUPR.6        ;состояние датчика ("0"-исключ.,"1"-включ.)
PREDRU  BIT     FLAGUPR.0       ;бит предыдущего режима управл. контроллером
PROGRET BIT     FLAGUPR.1       ;флаг "Контроллер прогрет" (лог."1")
BALANS  BIT     FLAGUPR.2       ;флаг "Балансировка выполнена" (лог."1")
ZATRKB  BIT     FLAGUPR.3       ;флаг "Затребован КБ" (лог."1")
RAZRYAD BIT     FLAGUPR.4       ;текущ. разрядн. АЦП ("0"-12 бит,"1"-8 бит)
TIPKOM  BIT     ACC.7           ;тип команды ("0"-первый,"1"-второй)
;----------------------------------------
;       Символьные имена бит портов 1 и 3
;
GOTACP  BIT     P3.2            ;готовность АЦП ("0"-конец преобразования)
REGUPR  BIT     P1.3            ;режим управления ("0"-авт.,"1"-ручн.)
RAZRACP BIT     P1.6            ;разрядность АЦП ("0"-12 бит,"1"-8 бит)
REGACP  BIT     P1.7            ;режим работы АЦП ("0"-преобраз.,"1"-чтение)

;----------------------------------------
;       Адреса  портов  контроллера
;
        XSEG    AT 4000H
EXTDATA XDATA   4000H
$NOCOND
$IF (TYPERPOGR = 0)
  AREGKB  EQU     4000H         ;адреса регистров кода балансировки,
  AREGND  EQU     6000H         ;       номера датчика температуры и
  AREGUN  EQU     8000H         ;       управления нагревом воды и воздуха,
  ADRACP  EQU     0A000H        ;адрес АЦП
$ELSE
          ORG     4000H
  AREGKB: DS      1             ;адреса регистров кода балансировки,
```

```
          ORG     6000H
  AREGND: DS      1               ;       номера датчика температуры и
          ORG     8000H
  AREGUN: DS      1               ;       управления нагревом воды и воздуха,
          ORG     0A000H
  ADRACP: DS      1               ;адрес АЦП
$ENDIF
$COND
$EJECT

        CSEG    AT 0H
CODE0   CODE    0H              ;начало сегмента
;----------------------------------------
;       Адреса внешних символов
;
;PUSK    EQU     30H
        ORG     0000H
;----------------------------------------
;       Отладочная часть
;
        SJMP    PUSK
        ORG     0030H
PUSK:   LJMP    2000H           ;адрес перезапуска контроллера

;----------------------------------------
;       Коды сообщений и запросов котроллера,пересылаемые управляющей ПЭВМ
;
NEPROGR EQU     0               ;контроллер не прогрет
DANBAL  EQU     1               ;передать данные для балансировки
OTKACP  EQU     2               ;отказ АЦП
OPROSDT EQU     3               ;выполняется опрос датчиков температуры
AVTREG  EQU     4               ;измерение температуры под управлением ПЭВМ
RUCHREG EQU     5               ;ручное измерение температуры
KBREKT  EQU     6               ;нет КБ ректального,
KBKOGN  EQU     7               ;       кожного,
KBVOD   EQU     8               ;       водяного датчиков температуры,
REZKOM  EQU     9               ;резервная команда
;
; PRIZND+(NOMDAT) - передать КБ указанного датчика
;
; PRIZKT1+(NOMDAT) - принять 8-миразряд. код температуры указанного датчика
; <код температуры> - 7...0 двоичные разряды
;
; PRIZKT2+(NOMDAT) - принять 12-тиразряд. код температуры указанного датчика
; <ст.байт кода темп.> - 11...4 двоичные разряды
; <мл.байт кода темп.> - ст. тетрада - 3...0 разряды,мл. тетрада - 0
;
; PRIZSK - принять 3 байта состояния контроллера
; (FLAGUPR) - содержимое регистров флагов управления,
; (FLAGDT) -    флагов датчиков температуры и
; (KREGUN) -    управления нагревом
;
;       Коды команд от управляющей ПЭВМ
;
PEREZAP EQU     0               ;перезапуск контроллера
OTMPROG EQU     1               ;отмена прогрева контроллера
SOSTOYA EQU     2               ;выдать состояние контроллера
KONBAL  EQU     3               ;конец балансировки
VKLNVOD EQU     4               ;включить нагрев воды
VYKNVOD EQU     5               ;выключить нагрев воды
VKLOVOD EQU     6               ;включить охлаждение воды
VYKOVOD EQU     7               ;выключить охлаждение воды
VKLNVOZ EQU     8               ;включить нагрев воздуха
VYKNVOZ EQU     9               ;выключить нагрев воздуха
```

```
RAZR8B  EQU     10              ;установить разрядность 8 бит
RAZR12B EQU     11              ;установить разрядность 12 бит
;
; 80H+<номер датчика> - исключить указанный датчик из цикла опроса
;
; C0H+<номер датчика> - включить указанный датчик в цикл опроса
; <КБ датчика>
;
; Примечания. 1) <номер датчика> - 3 двоичных разряда
;             2) <КБ датчика> передается после получения запроса
;                "Передать КБ указанного датчика"
;
;       Значения остальных символьных имен
;
NADRKB  EQU     44H             ;начальный адрес хранения кодов балансировки
PRIZND  EQU     80H             ;признаки запроса КБ указ. датчика температ.,
PRIZKT1 EQU     90H             ; выдачи 8-миразрядного или
PRIZKT2 EQU     0A0H            ;  12-тиразряд. кода температуры указ. датч.,
PRIZSK  EQU     0AH             ; выдачи состояния контроллера

        CSEG    AT 2000H
CODE1   CODE    2000H           ;начало сегмента
        ORG     2000H
;----------------------------------------
;       Начальная настройка контроллера
;

        MOV     SP,#07H
        CLR     A
        MOV     IE,A            ;запрещение прерываний
        MOV     R7,A            ;обнуление счетчика цикла опроса датчиков,
        ACALL   ZREGUN          ; регистра управления нагревом
        MOV     KREGUN,A        ; и его копии в ОЗУ
        MOV     FLAGDT,A        ;сброс флагов подключен. датчиков температ.,
        CLR     PROGRET         ; флагов "Контоллер прогрет",
        CLR     BALANS          ;        "Датчики отбалансированы" и
        CLR     ZATRKB          ;        "Затребован КБ",
        CLR     RI              ;а также флагов готовности приемника и
        CLR     TI              ; передатчика последовательных данных
        SETB    RAZRYAD         ;установка флага 8-мибитной разрядности АЦП
        SETB    REN             ;разрешение приема последовательных данных
        ACALL   ZAPRU           ;запоминание режима управления и
        JNB     PREDRU,AVTUPR1  ;переход,если автоматический
;----------------------------------------
;       Ручное управление измерением температуры
;
        MOV     R4,#30          ;установка 10-тиминут. прогрева контроллера
        MOV     R5,#200
RUCHUPR:MOV     A,#RUCHREG      ;выдача сообщения:
        ACALL   SOOBSH          ; "Ручное измерение температуры"
L0:     JNB     RI,L3           ;переход,если нет данных от ПЭВМ,
        ACALL   PRIEM           ;иначе прием команды
        CJNE    A,#PEREZAP,L1   ; "Перезапуск контроллера"? - нет,продолжить
        LJMP    PUSK            ;да,переход
L1:     CJNE    A,#OTMPROG,RUCHUPR;"Отмена прогрева контрол."? - нет,переход
L2:     SETB    PROGRET         ;да,установка флага "Контроллер прогрет"
L3:     JB      PROGRET,L5      ;переход,если контроллер прогрет,
        ACALL   ZADER2          ;иначе временная задержка на 0,1с
        DJNZ    R5,L5
        DJNZ    R4,L4
        SJMP    L2              ;переход,если время прогрева истекло
L4:     MOV     R5,#200
L5:     ACALL   IZMRU           ;режим управления изменился?
        JNC     L0              ;нет,переход
```

```
        ACALL   ZAPRU           ;да,запоминание нового положения тумблера
        SJMP    AVTUPR2         ; "Режим управления" и переход
;---------------------------------------
;               Автоматическое управление измерением температуры
;
;       Прогрев контроллера в течение 10 минут
;
AVTUPR1:MOV     R4,#30          ;установка 10-тиминут. прогрева контроллера
        MOV     R5,#200
AVTUPR2:MOV     A,#AVTREG       ;выдача сообщения:
        ACALL   SOOBSH          ; "Измерение температуры под управлен. ПЭВМ"
        JB      PROGRET,BALANDT ;переход,если контроллер прогрет или
L6:     JNB     RI,L9           ; нет данных от ПЭВМ,иначе
        ACALL   PRIEM           ;прием команды
        CJNE    A,#PEREZAP,L7   ;"Перезапуск контроллера"? - нет,продолжить
        LJMP    PUSK            ;да,переход
L7:     CJNE    A,#OTMPROG,L8   ;"Отмена прогрева" - нет,продолжить
        SJMP    L10             ;да,переход
L8:     MOV     A,#NEPROGR      ;выдача сообщения:
        ACALL   SOOBSH          ; "Контроллер не прогрет"
L9:     ACALL   ZADER2          ;временная задержка на 0,1 с
        DJNZ    R5,L12
        DJNZ    R4,L11          ;если время прогрева истекло
L10:    SETB    PROGRET         ; установка флага "Контроллер прогрет"
        SJMP    BALANDT
L11:    MOV     R5,#200
L12:    ACALL   IZMRU           ;режим управления изменился ?
        JNC     L6              ;нет,переход
L13:    ACALL   ZAPRU           ;да,запоминание нового положения тублера
        SJMP    RUCHUPR         ; "Режим управления" и переход
;---------------------------------------
;       Балансировка датчиков температуры
;
BALANDT:JNB     BALANS,M0       ;переход,если балансировка выполнена или
        AJMP    IZMTEMP
M0:     JNB     ZATRKB,POVTCB   ; не принят КБ,
        AJMP    PRIEMKB
POVTCB: MOV     A,#DANBAL       ;иначе  выдача сообщения:
        ACALL   SOOBSH          ; "Передать данные для балансировки"
M1:     MOV     R5,#10          ;установка 1с-ожидания поступления данных
M2:     ACALL   IZMRU           ;режим управления изменился ?
        JNC     M3              ;нет,продолжить
        SJMP    L13             ;да,переход к ручному управлению
M3:     JB      RI,M4           ;переход,если данные поступили,
        ACALL   ZADER2          ;иначе 0,1с-задержка и
        DJNZ    R5,M2           ;переход,если тайм-аут не окончился,
        SJMP    POVTCB          ;иначе повторить запрос данных
M4:     ACALL   PRIEM           ;прием байта,поступившего от ПЭВМ
        JB      TIPKOM,KOM2T    ;переход,если команда 2-го типа,
        CJNE    A,#PEREZAP,M5   ;иначе "Перезапуск контроллера"?
        LJMP    PUSK
M5:     CJNE    A,#SOSTOYA,M6   ; "Выдать состояние контроллера"?
        ACALL   SOSTKON
        SJMP    POVTCB
M6:     CJNE    A,#KONBAL,POVTCB; "Конец балансировки"?
        JB      FLREKT,M7       ;продолжить,если флаг ректального установлен,
        MOV     A,#KBREKT       ;иначе выдача сообщуния:
        ACALL   SOOBSH          ; "Нет КБ ректального датчика"
        SJMP    M1
M7:     JB      FLKOGN,M8       ;продолжить,если флаг кожного установлен,
        MOV     A,#KBKOGN       ;иначе выдача сообщения:
        ACALL   SOOBSH          ; "Нет КБ кожного датчика"
        SJMP    M1
M8:     JB      FLVOD,M9        ;продолжить,если флаг водяного установлен,
```

```
        MOV     A,#KBVOD        ;иначе выдача сообщения:
        ACALL   SOOBSH          ; "Нет КБ водяного датчика"
        SJMP    M1
M9:     SETB    BALANS          ;установка флага "Балансировка выполнена" и
        AJMP    IZMTEMP         ;переход к измерению температуры
KOM2T:  ANL     A,#00111000B    ;команда балансировки ?
        JZ      M10             ;да,продолжить
        SJMP    POVTCB          ;нет,переход
M10:    MOV     A,KODUPR        ;формирование номера датчика,к которому
        ANL     A,#00000111B    ; относится эта команда,
        MOV     NOMDAT,A
        ADD     A,#NADRKB       ; адреса ячейки хранения его КБ и
        MOV     R0,A
        MOV     R1,NOMDAT       ; местоположения флага этого датчика
        INC     R1              ;  в регистре FLAGDT
        MOV     A,#01H
M11:    DJNZ    R1,M12
        SJMP    M13
M12:    RL      A
        SJMP    M11
M13:    MOV     MPFLAG,A
        JB      SOSTDAT,M14     ;переход,если датчик включается в систему,
        CPL     A               ;иначе сброс его флага в регистре FLAGDT и
        ANL     FLAGDT,A
        AJMP    POVTCB          ;переход к следующему циклу балансировки
M14:    SETB    ZATRKB          ;установка флага "Затребован КБ" и
PRIEMKB:MOV     A,#PRIZND       ;выдача сообщения:
        ADD     A,NOMDAT        ; "Передать КБ указанного датчика"
        ACALL   SOOBSH
        MOV     R5,#10          ;установка 1с-ожидания поступления данных
M15:    ACALL   IZMRU           ;режим управления изменился ?
        JNC     M16             ;нет,продолжить
        AJMP    L13             ;да,переход к ручному управлению
M16:    JB      RI,M17          ;ожидание поступления байта данных
        ACALL   ZADER2
        DJNZ    R5,M15
        SJMP    PRIEMKB
M17:    ACALL   PRIEM           ;и его прием,
        MOV     @R0,A           ;пересылка КБ в ячейку хранения,
        MOV     A,MPFLAG        ;установка флага,свидетельствующего о его
        ORL     FLAGDT,A        ; поступлении,
        CLR     ZATRKB          ;сброс флага "Затребован КБ" и
        AJMP    POVTCB          ;переход к следующему циклу балансировки
;-----------------------------------------
;       Измерение температуры и управление нагревом воды и воздуха
;
IZMTEMP:MOV     NOMDAT,#0       ;;+
        ACALL   KOMUTAT         ;;+
        MOV     R6,#3           ;;+ задержка после коммутации
DP2:    ACALL   ZADER1          ;;+
        DJNZ    R6,DP2          ;;+
        CJNE    R7,#00H,N0      ;переход,если предыдущий цикл опроса датчиков

NOVCIKL:MOV     RREGFL,FLAGDT   ; температуры не закончен,иначе подготовка
        MOV     NOMDAT,R7       ; следующего цикла
        MOV     R7,#8
N0:     JB      RAZRYAD,N1      ;установка
        CLR     RAZRACP         ; 12-тибитной или
        SJMP    N2
N1:     SETB    RAZRACP         ; 8-мибитной разрядности АЦП
N2:     MOV     A,RREGFL        ;определение номера следующего
        CLR     C               ; опрашиваемого датчика температуры и
N3:     RRC     A
        JC      N4
```

```
        INC     NOMDAT
        DJNZ    R7,N3
DP0:    MOV     R6,#1           ;задержка между пакетами
DP1:    ACALL   ZADER1
        DJNZ    R6,DP1
        SJMP    NOVCIKL
N4:     MOV     RREGFL,A
        ACALL   KOMUTAT
        ACALL   ZADER1          ;задержка на время установл. сигнала темпер.
        ACALL   ZADER1
        CLR     REGACP          ;перевод АЦП в режим "Преобразование" и
        ACALL   PUSKACP         ; его пуск
        MOV     R6,#4           ;задержка на время преобразования
N5:     DJNZ    R6,N5
        JNB     GOTACP,N6       ;данные АЦП готовы ? - да,переход
        MOV     A,#OTKACP       ;нет,выдача сообщения:

        ACALL   SOOBSH          ; "Отказ АЦП" и
        SJMP    N10             ;переход к проверке поступл. команды от ПЭВМ
N6:     ACALL   IZMRU           ;режим управления изменился ?
        JNC     N7              ;нет,переход
        INC     NOMDAT          ;да,корректировка номера датчика температуры,
        DEC     R7              ; а также счетчика цикла опроса и
        AJMP    L13             ;переход к ручному измерению температуры
N7:     SETB    REGACP          ;перевод АЦП в режим "Чтение"
        JB      RAZRYAD,N8      ;формирование и
        MOV     A,#PRIZKT2
        SJMP    N9
N8:     MOV     A,#PRIZKT1
N9:     ADD     A,NOMDAT
        INC     NOMDAT          ;;+ формирование номера следующего датчика
        MOV     R6,A
        MOV     R5,NOMDAT
        MOV     A,RREGFL
        CJNE    A,#0,DP3
        MOV     NOMDAT,#0
DP3:    ACALL   KOMUTAT
        MOV     NOMDAT,R5
        MOV     A,R6
        ACALL   SOOBSH          ;выдача сообщения:
        CLR     RAZRACP         ; "Код температуры"
        ACALL   CHTACP          ;чтение и
        ACALL   SOOBSH          ;передача старшего байта кода температуры,
        JB      RAZRYAD,N10     ;а в случае 12-тибитной разрядности АЦП,
        SETB    RAZRACP         ; также его младшего байта
        ACALL   CHTACP
        ACALL   SOOBSH
N10:    JB      RI,VYPKOM       ;переход,если поступили данные от ПЭВМ,иначе
SLEDDAT:DJNZ    R7,N0           ;продолж. цикла опроса,если он не закончился,
        SJMP    DP0
;       ACALL   ZADER1
;       AJMP    NOVCIKL         ;иначе переход к следующему циклу
VYPKOM: ACALL   PRIEM           ;прием команды
        JNB     TIPKOM,N11      ;продолжить,если команда 1-го типа,иначе
        CLR     BALANS          ;сброс флага "Балансировка выполнена",
        MOV     R7,#00H         ; а также счетчика цикла опроса и
        AJMP    KOM2T           ;переход к процедуре балансировки
N11:    CJNE    A,#PEREZAP,N12  ;команда "Перезапуск контроллера" ?
        LJMP    PUSK            ;да,переход
N12:    CJNE    A,#OTMPROG,N13  ; "Отмена прогрева" ?
        SJMP    N14
N13:    CJNE    A,#KONBAL,N15   ; "Конец балансировки" ?
N14:    MOV     A,#OPROSDT      ;выдача сообщения:
        ACALL   SOOBSH          ; "Опрос датчиков температуры"
```

```
        SJMP    SLEDDAT
N15:    CJNE    A,#VKLNVOD,N16  ; "Включение нагрева воды" ?
        SETB    NAGRVOD
        SJMP    N21
N16:    CJNE    A,#VYKNVOD,N17  ; "Выключение нагрева воды" ?
        CLR     NAGRVOD
        SJMP    N21
N17:    CJNE    A,#VKLOVOD,N18  ; "Включение охлаждения воды" ?
        SETB    OHLVOD
        SJMP    N21
N18:    CJNE    A,#VYKOVOD,N19  ; "Выключение охлаждения воды" ?
        CLR     OHLVOD
        SJMP    N21
N19:    CJNE    A,#VKLNVOZ,N20  ; "Включение нагрева воздуха" ?
        SETB    NAGRVOZ
        SJMP    N21
N20:    CJNE    A,#VYKNVOZ,N22  ; "Выключение нагрева воздуха" ?
        CLR     NAGRVOZ
N21:    MOV     A,KREGUN        ;соответствующая коррекция содержимого
        ACALL   ZREGUN          ; регистра управления нагревом
        SJMP    SLEDDAT
N22:    CJNE    A,#SOSTOYA,N23  ; "Выдать состояние контроллера" ?
        ACALL   SOSTKON
        SJMP    SLEDDAT
N23:    CJNE    A,#RAZR8B,N24   ; "Установить разрядность 8 бит" ?
        SETB    RAZRYAD
        SJMP    SLEDDAT
N24:    CJNE    A,#RAZR12B,N25  ; "Установить разрядность 12 бит" ?
        CLR     RAZRYAD
        SJMP    SLEDDAT
N25:    MOV     A,#REZKOM       ;выдача сообщения:
        ACALL   SOOBSH          ; "Резервная команда"
        SJMP    SLEDDAT
;----------------------------------------
;               Подпрограммы
;----------------------------------------
;       Запись в регистры управления и пуск АЦП
;
ZREGKB: MOV     DPTR,#AREGKB
        SJMP    LZ0
ZREGND: MOV     DPTR,#AREGND
        SJMP    LZ0
ZREGUN: MOV     DPTR,#AREGUN
        SJMP    LZ0
PUSKACP:MOV     DPTR,#ADRACP
LZ0:    MOVX    @DPTR,A
        RET
;----------------------------------------
;       Запоминание режима управления
;
ZAPRU:  MOV     C,REGUPR
        MOV     PREDRU,C
        RET
;----------------------------------------
;       Выдача сообщения на ПЭВМ
;
SOOBSH: MOV     SBUF,A
SL0:    JNB     TI,SL0
        CLR     TI
        RET
;----------------------------------------
;       Прием данных от ПЭВМ
;
PRIEM:  MOV     A,SBUF
```

```
        MOV     KODUPR,A
        CLR     RI
        RET
;----------------------------------------
;       Проверка изменения режима управления
;
IZMRU:  MOV     C,PREDRU
        JNB     REGUPR,LI0
        CPL     C
LI0:    RET
;----------------------------------------
;       Временная задержка на 1мс или 0,1с
;
ZADER1: MOV     R2,#2
        SJMP    LZ1
ZADER2: MOV     R2,#200
LZ1:    MOV     R3,229
LZ2:    DJNZ    R3,LZ2
        DJNZ    R2,LZ1
        RET
;----------------------------------------
;       Чтение данных АЦП
;
CHTACP: MOV     DPTR,#ADRACP
        MOVX    A,@DPTR
        RET
;----------------------------------------
;       Выдача состояния контроллера
;
SOSTKON:MOV     A,#PRIZSK
        ACALL   SOOBSH
        MOV     A,FLAGUPR
        ACALL   SOOBSH
        MOV     A,FLAGDT
        ACALL   SOOBSH
        MOV     A,KREGUN
        ACALL   SOOBSH
        RET
;----------------------------------------
;       Коммутация датчика
;
KOMUTAT:MOV     A,#NADRKB       ; адреса хранения его кода балансировки
        ADD     A,NOMDAT
        MOV     R0,A            ; занесение в соответстующие регистры
        MOV     A,@R0
        ACALL   ZREGKB          ; кода балансировки датчика температуры и
        MOV     A,NOMDAT
        ACALL   ZREGND          ; его номера
        RET
;
        END
```

5. ПРИМЕР. ЭТАП "ОТ ИСХОДНОЙ ПРОГРАММЫ К ИСПОЛНИТЕЛЬНОМУ КОДУ".

На простом примере по методу “Быстрый старт” изучаются этапы технологии разработки и отладки программ, основные приёмы работы на примере среды µVision2.

5.1. Введение в µVision2.

µVision2 фирмы Keil – интегрированная среда разработки программного обеспечения для однокристальных микроконтроллеров семейства Intel 8051 и его клонов. Она включает в себя всё, что нужно для создания, редактирования, компиляции, трансляции, компоновки, загрузки и отладки программ: 1) - стандартный интерфейс Windows, 2) - полнофункциональный редактор исходных текстов с выделением синтаксических элементов цветом, 3) - организатор проекта, 4) - транслятор с языка C, 5) - ассемблер, 6) - отладчик, 7) - встроенную справочную систему.

Среда разработки подобна Visual C++ Microsoft и Borland C++ для Windows. Пользователи, знакомые с любым из этих изделий, будут чувствовать себя в µVision2, как дома.

1. Первый этап разработки программы – запись её исходного текста на каком-либо языке программирования.

2. Затем производится компиляция или трансляция его в коды из системы команд микроконтроллера, используя транслятор или ассемблер. Трансляторы и ассемблеры – прикладные программы, которые интерпретируют текстовый файл, содержащий исходный текст программы, и создают объектные файлы, содержащие объектный код.

3. Компановка объектных модулей в исполнительный файл с помощью редактора связей (компановщика).

4. После компоновки объектных модулей наступает этап отладки программы, устранения ошибок, оптимизации и тестирования программы.

Интегрированная среда разработки µVision2 объединяет все этапы разработки прикладной программы в единый рекурсивный процесс, когда в любой момент времени возможен быстрый возврат к любому предыдущему этапу. Для этого µVision2 имеет следующие компоненты.

Макроассемблер A51

Ассемблер A51 совместим с ASM51 Intel для всего семейства микроконтроллеров Intel 8051. Ассемблер транслирует символическую мнемонику в перемещаемый объектный код, имеющий высокое быстродействие и малый размер. Макросредства ускоряют разработку и экономят время, поскольку общие последовательности могут быть разработаны только один раз. Ассемблер поддерживает символический доступ ко всем элементам микроконтроллера и перестраивает конфигурацию для каждой разновидности Intel 8051.

A51 транслирует исходный файл ассемблера в перемещаемый объектный модуль. При отладке или при включенной опции “Include debugging information” этот объектный файл будет содержать полную символическую информацию для отладчика/имитатора или внутрисхемного эмулятора.

Оптимизирующий кросс-компилятор C51

Язык С - универсальный язык программирования, который обеспечивает эффективность кода, элементы структурного программирования и имеет богатый набор операторов. Универсальность, отсутствие ограничений реализации делают язык С удобным и эффективным средством программирования для широкого разнообразия задач. Множество прикладных программ может быть написано легче и эффективнее на языке С, чем на других более специализированных языках.

C51 - полная реализация стандарта ANSI (Американского национального института стандартов), насколько это возможно для архитектуры Intel 8051. C51 генерирует код для всего семейства микроконтроллеров Intel 8051. Транслятор сочетает гибкость программирования на языке С с эффективностью кода и быстродействием ассемблера.

Использование языка высокого уровня С имеет следующие преимущества над программированием на ассемблере: 1) - глубокого знания системы команд процессора не требуется, элементарное знание архитектуры Intel 8051 желательно, но не необходимо; 2) - распределение регистров и способы адресации управляются полностью транслятором; 3) – лучшая читаемость программы, используются ключевые слова и функции, которые более свойственны человеческой мысли; 4) – время разработки программ и их отладки значительно короче в сравнении с программированием на ассемблере; 5) - библиотечные файлы содержат много стандартных подпрограмм, которые могут быть включены в прикладную программу; 6) - существующие программы могут многократно использоваться в новых программах, используя модульные методы программирования.

Компоновщик L51

Компоновщик объединяет один или несколько объектных модулей в одну исполняемую программу. Компоновщик размещает внешние и общие ссылки, назначает абсолютные адреса перемещаемым сегментам программ. Он может обрабатывать объектные модули, созданные транслятором C51, ассемблером A51, транслятором PL/M-51 Intel и ассемблером ASM51 Intel.

Компоновщик автоматически выбирает соответствующие библиотеки поддержки и связывает только требуемые модули из библиотек. Установки по умолчанию для L51 выбраны так, чтобы они подходили для большинства прикладных программ, но можно определить и заказные установки.

Отладчик/симулятор

Отладчик исходных текстов используется с транслятором C51, ассемблером A51, транслятором PL/M-51 Intel и ассемблером ASM51 Intel. Отладчик/симулятор позволяет моделировать большинство особенностей Intel 8051 без наличия аппаратных средств. Можно использовать его для проверки и отладки прикладной программы прежде, чем будут изготовлены аппаратные средства. При этом моделируется широкое разнообразие периферийных устройств, включая последовательный порт, внешний ввод - вывод и таймеры.

5.2. Быстрый старт

“Быстрый старт” – это обычный приём разработчиков современных программных средств. Цель состоит в том, чтобы, не углубляясь пока в подробности, дать новичку или достаточно опытному пользователю первое представление о программном средстве, дать возможность быстро получить конкретный результат. Полное представление, знания и умения появятся позже в процессе работы и изучения справочных материалов.

В качестве примера используется простейшая программа, с которой начинают изучение языков программирования многие поколения студентов. “Hello World” - программа из папки …\Examples\Hello\, которая выдаёт в последовательный порт (UART) микроконтроллера строку символов “Hello World” (“Привет Мир”). Весь исходный текст программы следующий:

Пример программы 2.

```
/*************************/
/* YOUR FIRST 8051 PROGRAM */
/*************************/
```

```
#include <reg51.h>     /* special function register declarations  */
                       /* for the intended 8051 derivative        */
#include <stdio.h>     /* prototype declarations for I/O functions*/

/****************/
/* main program */
/****************/
void main (void)  {            /* execution starts here after stack init */
   SCON  = 0x50;               /* SCON: mode 1, 8-bit UART, enable rcvr  */
   TMOD |= 0x20;               /* TMOD: timer 1, mode 2, 8-bit reload    */
   TH1 = 0xf3;                 /* TH1:  reload value for 2400 baud       */
   TR1 = 1;                    /* TR1:  timer 1 run                      */
   TI  = 1;                    /* TI:   set TI to send first char of UART*/
   printf ("Hello World\n");   /* the 'printf' function call             */

   while (1) {                 /* An embedded program does not stop and  */
   ;  /* ... */                /* never returns.  We've used an endless  */
   }                           /* loop.  You may wish to put in your own */
}                              /* code were we've printed the dots (...) */
```

5.2.1. Общий план работы

Для создания приложения в среде проектирования µVision2 необходимо выполнить следующие этапы:

1)- Запустить интегрированную среду µVision2, создать файл проекта, задать тип микроконтроллера, используемого в проекте.

2)- Создать новые исходные файлы программ на соответствующих языках программирования и включить их в проект в соответствующие группы.

3)- Дополнить проект конфигурационными файлами и настроить последние для заданной модели МК.

4)- Настроить опции проекта, рабочих групп и отдельных файлов, если в этом есть необходимость.

5)- Построить исполнительный файл проекта, провести его тестирование и отладку.

6)- Создать файл в формате HEX для программатора, и зашить программу в ПЗУ для контроллера.

5.2.2. Программа "Hello World"

Прежде чем начать разработку проекта, желательно создать свою личную папку. В этой папке находится всего лишь один файл с текстом данной программы – hello.c. Хотя, это можно сделать и позже.

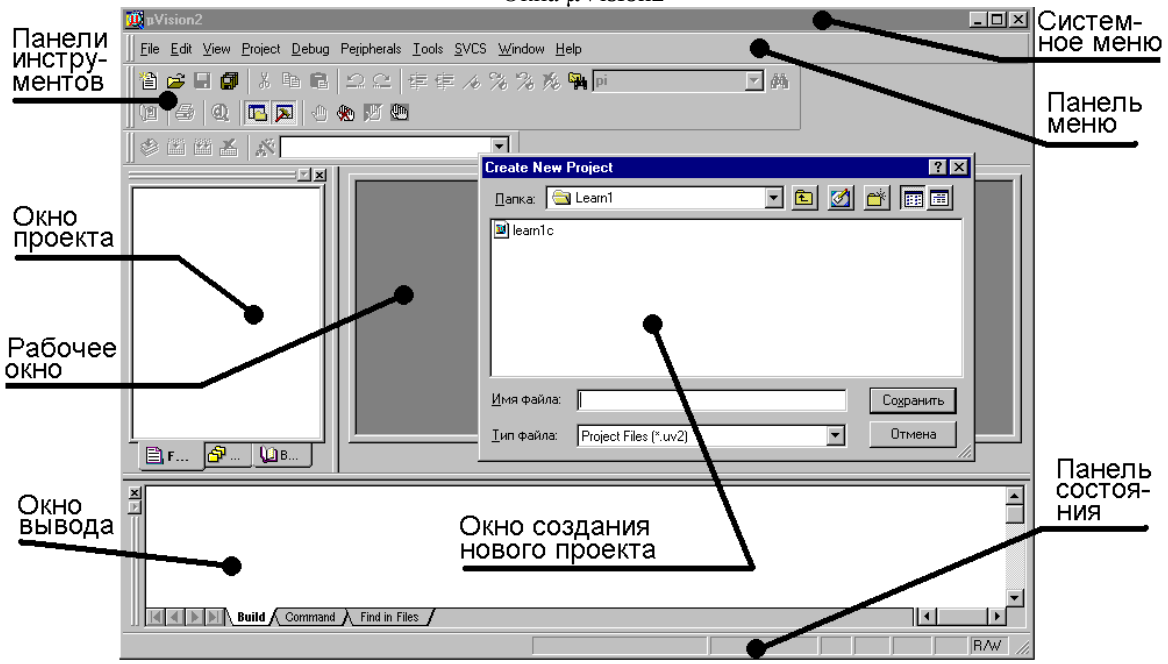
Запуск µVision2 и создание файла проекта. µVision2 запускается из стартового меню Windows подобно остальным приложениям. Если необходимо запустить программу из командной строки, её синтаксис имеет вид:

uv2 [commands] [projectfile],

где projectfile - имя файла проекта с расширением [.Uv2], а необязательные команды управляют загрузкой.

Любая новая работа в µVision2, как и во всех современных интегрированных средах, начинается с создания нового файла проекта. Файл проекта содержит имена всех исходных файлов, связанных с проектом, а также установки компиляции, трансляции и связывания файлов, чтобы генерировать выполняемую программу.

Окна µVision2



Типичный вид окна интегрированной среды разработки программных средств для семейства MCS-51 µVision2.

Рис. 11.

Тип устройства

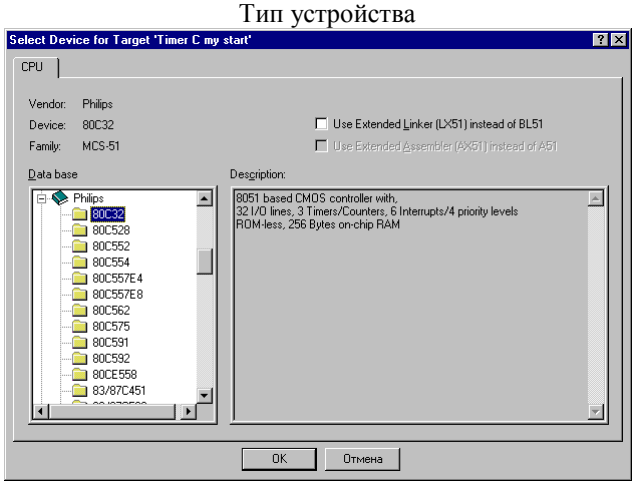


Рис. 12.

Для того чтобы создать новый файл проекта, выберите из меню **Project – New Project …** . Откроется стандартное диалоговое окно для создания файлов - Create Project (рис. 11). Используйте кнопки иконки, чтобы войти в свою папку. Найдите свою папку и нажмите кнопку [OK]. На этом этапе также можно создать свою папку с помощью иконки **Create New Folder** и в ней указать свой проект.

Теперь обязательно задать тип микроконтроллера. Он определяет важные параметры всех инструментов интегрированной среды. Тип задаётся с помощью меню **Project – Select Device for Target**, что показано на рис. 12. Для нового проекта запрашивается автоматически.

Тип микроконтроллера очень важен для линковки. Тип описывает параметры, которые чаще всего придётся вносить в соответствующие файлы конфигурации в проекте.

Среда содержит базу данных руководящих технических материалов для предлагаемых типов МК. Доступ к базе осуществляется через окно проекта, закладку **Books**. Эту базу данных можно пополнять и самому для нужных МК.

Теперь можно наполнять проект содержимым. Используется меню **Project – Targets, Groups, Files …** , которое даёт диалог, приведенный на рис 13. Этот диалог можно получить и в локальном меню окна проекта на закладке **Files**, предварительно выделив нужную тему. Так наполняется проект из готовых файлов.

Если готового файла нет, его следует создать посредством меню **File – New.** В рабочей части окна откроется окно для текстового ввода. Набрав программу, её сохраняют с меню **File – Save As …** .

Диалог **Targets, Groups, Files …** позволяет создать в одном проекте несколько целей, в каждой цели несколько групп определённой принадлежности, а в каждой группе несколько файлов. Цели впоследствии можно добавлять и удалять; тоже самое можно делать и с группами и с файлами.

Каждая тема в проекте на своём уровне имеет свой набор управляющих опций, которые управляют всеми инструментами при создании целевых файлов проекта.

Окно целей, групп и файлов

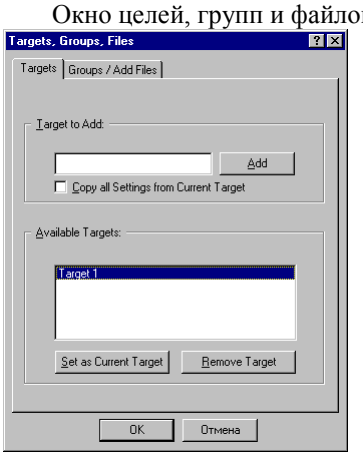


Рис. 13.

Диалог опций цели 1

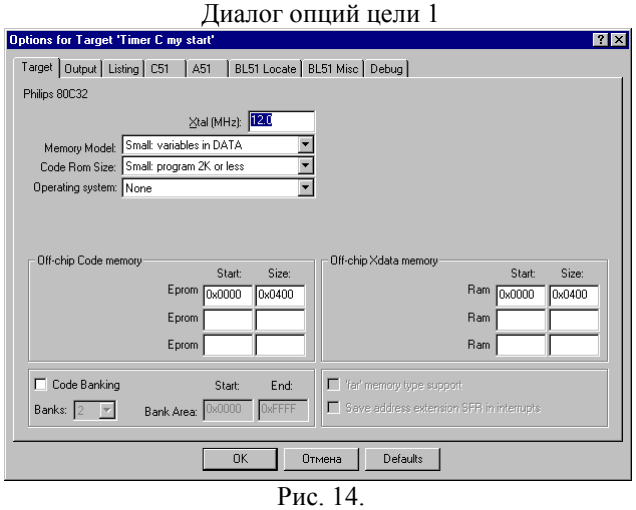


Рис. 14.

Для выходного файла задаётся имя файла и его каталог при необходимости. В выходной файл можно включить информацию для отладки и для базы имён программы. Кроме того, оговаривается создание файла для программатора. Файлы можно также включить в собственную библиотеку.

Здесь также задаются действия, выполняемые средой после построения проектного файла. Это может быть старт отладки. Кроме того можно задать две другие программы.

Закладка **Listing** управляет листингами компиляторов, задействованных на данном уровне тем проекта

Наполнив проект содержимым, необходимо установить соответствующие опции. Диалог установки опций меняется от уровня темы и самый полный для уровня темы цели. Меню **Options for Target …** открывает этот диалог, который приведен на рис. 14.

Закладка **Target** задаёт частоту тактирования МК, модель памяти, конфигурацию внутренних блоков МК, конфигурацию внешней памяти и модель банков памяти для огромных многофазовфх (оверлейных) программ, если необходимо.

Рис. 15 показывает вторую закладку **Output**, которая определяет выходные файлы для цели проекта.

Диалог опций цели 1

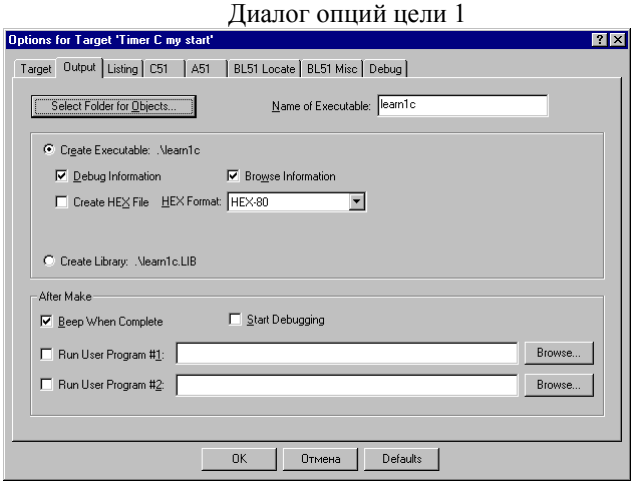


Рис. 15.

или даже исходного файла.

Опции компилятора С

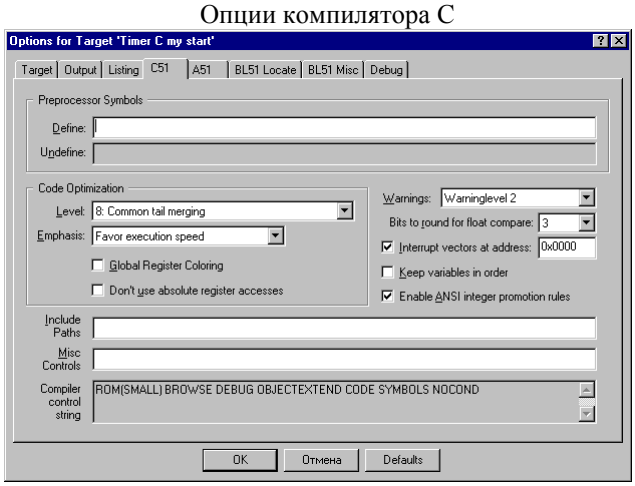


Рис. 16.

Основные опции для компиляторов задаются на своих закладках, которые приведены на рис. 16 и 17.

Опции Ассемблера

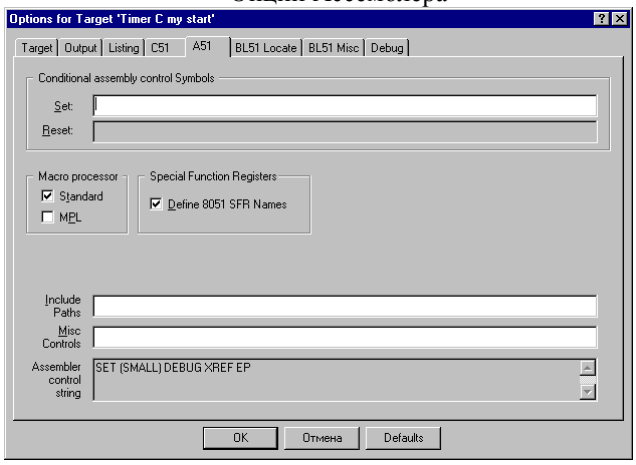


Рис. 17.

5.3. Введение в ProView.

ProView фирмы Franklin Software Inc. (Keil Software в США и Канаде) – интегрированная среда разработки программного обеспечения для однокристальных микроконтроллеров семейства Intel 8051 и его клонов.

На простом примере по методу “Быстрый старт” изучаются этапы технологии разработки и отладки программ, основные приёмы работы на примере среды ProView.

Прежде чем начать разработку проекта, создайте свою личную папку. В этой папке находится всего лишь один файл с текстом данной программы – hello.c.

Запуск ProView и создание файла проекта.

ProView запускается из стартового меню Windows подобно остальным приложениям (рис. 18). Если необходимо запустить программу из командной строки, её синтаксис имеет вид: **PV32 [projectfile]**, где projectfile - имя файла проекта с расширением [.PRJ].

Запуск программы

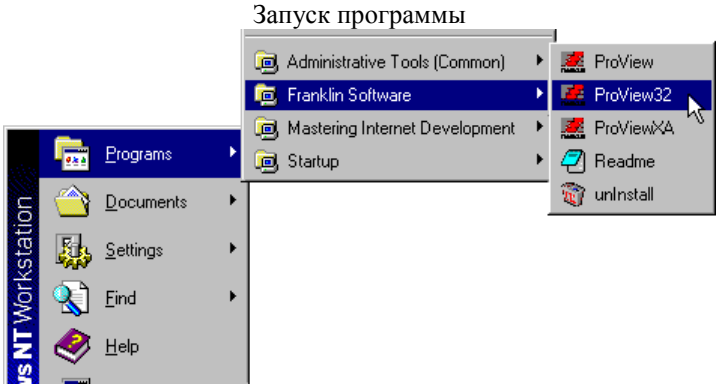
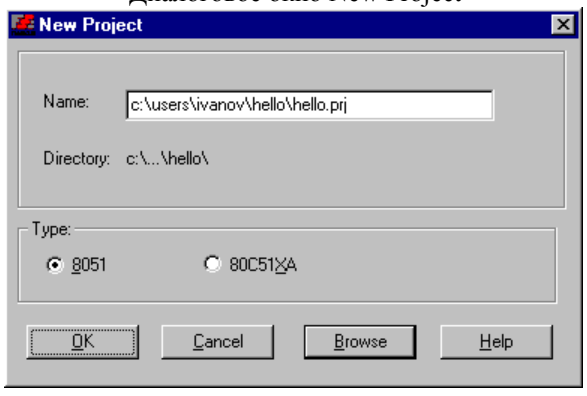


Рис. 18.

Диалоговое окно New Project



Любая новая работа в ProView, как и во всех современных компиляторах, начинается с создания нового файла проекта. Файл проекта содержит имена всех исходных файлов, связанных с проектом, а также установки компиляции, трансляции и связывания файлов, чтобы генерировать выполняемую программу.

Для того чтобы создать новый файл проекта, выберите **New** из меню **Project**. Откроется диалоговое окно New Project (рис. 19). Используйте кнопку Browse, чтобы войти в свою папку. Найдите свою папку и нажмите кнопку [OK]. Затем выберите “8051” как тип проекта.

Рис. 19.

Когда менеджер проекта открывает файл проекта, окно проекта показывает включенные исходные файлы. В данном случае пока нет никаких исходных файлов. Имеется только один исходный файл, который необходимо подключить - hello.c.

Добавка файла с исходным текстом и его редактирование

Диалоговое окно Add File

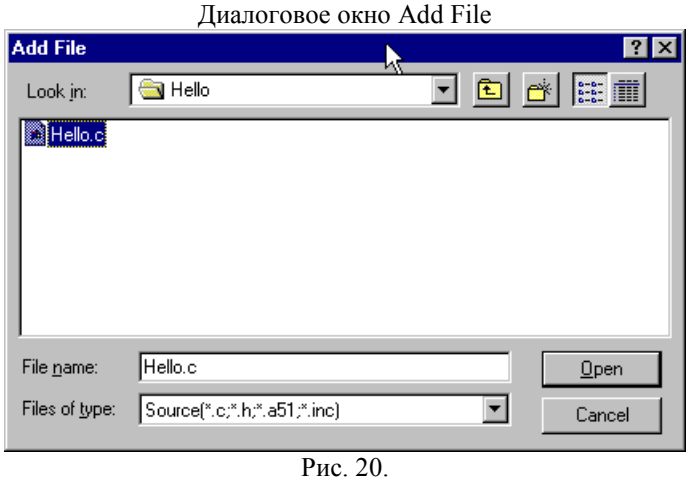


Рис. 20.

Теперь можно добавить hello.c к проекту. Выберите Add file из меню Project. Откроется диалоговое окно Add File (рис. 20). Выберите hello.c из списка.

Наш проект имеет только один исходный файл. В дальнейшем Ваши проекты, возможно, будут состоять из множества исходных файлов. Диалог Add File позволит Вам выбрать и добавить несколько файлов сразу. Для этого используют комбинацию клавиши [CTRL] и указателя мыши. Когда Вы нажмёте [Open], исходные файлы будут добавлены к проекту в выбранном порядке.

Теперь можно редактировать текст из файла hello.c. Выберите hello.c из окна Project (рис. 21). Нажмите его правой кнопкой мыши и выберите View source file, или просто дважды щёлкните мышью для того, чтобы просматривать файл в окне редактирования.

Диалоговое окно Project

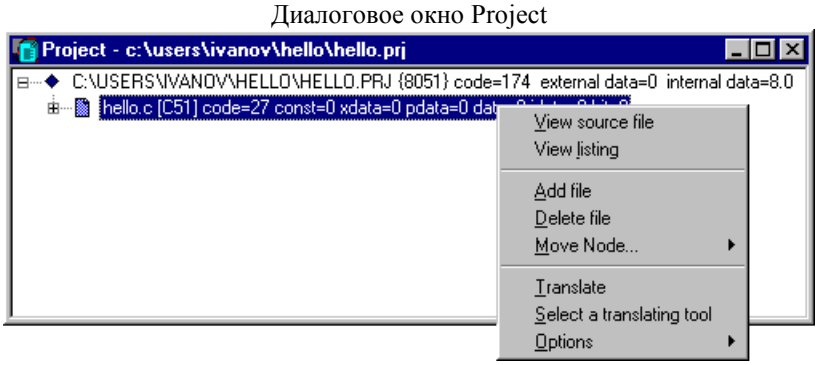


Рис. 21.

Окно редактирования

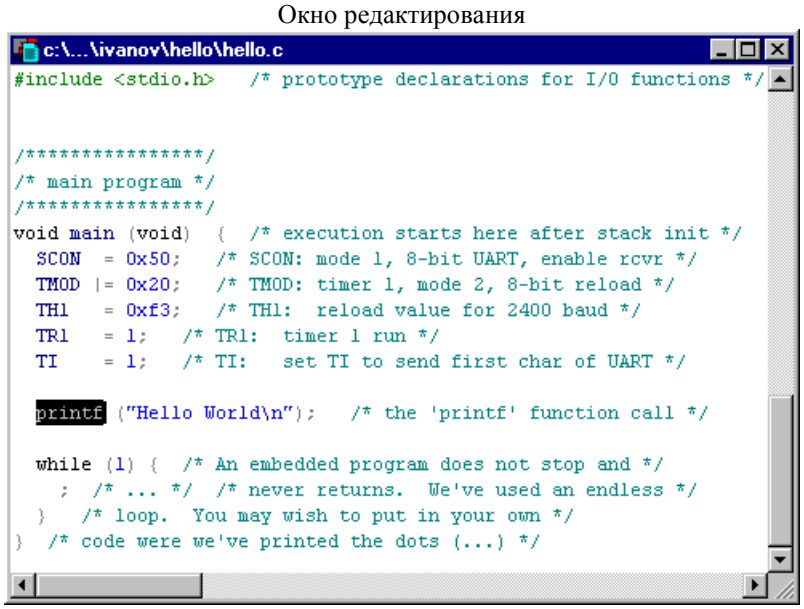


Рис. 22.

ProView загружает и показывает содержание hello.c в окне, где можно редактировать файл. Окно редактирования (рис. 22) - полнофункциональный редактор исходного текста, предлагающий такие возможности, как высвечивание синтаксических элементов и контекстный поиск. Если выбрать “printf” и нажать клавишу [F1], ProView откроет систему справки и перейдёт к разделу справки о “printf”.

Компиляция и компоновка.

Этот процесс компилирует, связывает hello.c с библиотеками и создает абсолютный объектный модуль, который мы сможем проверить в отладчике WinSim.

Выберите Make из меню Project. ProView отображает окно, показывая текущее состояние процесса. Когда процесс компиляции закончится, в окне Message (рис. 23) отображается сообще-

Окно сообщений

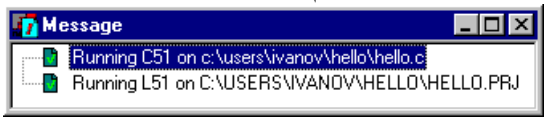


Рис. 23.

Рис. 1. Окно сообщений

них сообщается здесь же.

Тестирование и отладка.

Выполним отладку программы. Если проект новый, откроется диалоговое окно Debug Options (рис. 24), где Вы можете изменять установки отладчика. В дальнейшем можно установить опции отладчика, выбрав Debug из меню Options. Наш проект использует значения по умолчанию.

Выберите Start из меню Debug.

Окно диалога опций отладчика

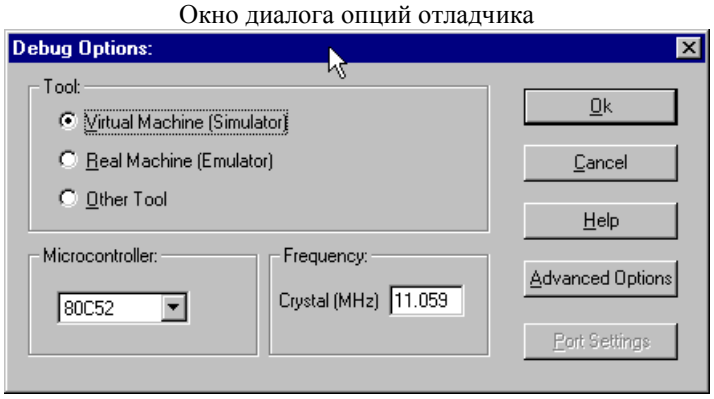


Рис. 24.

Окно последовательного порта

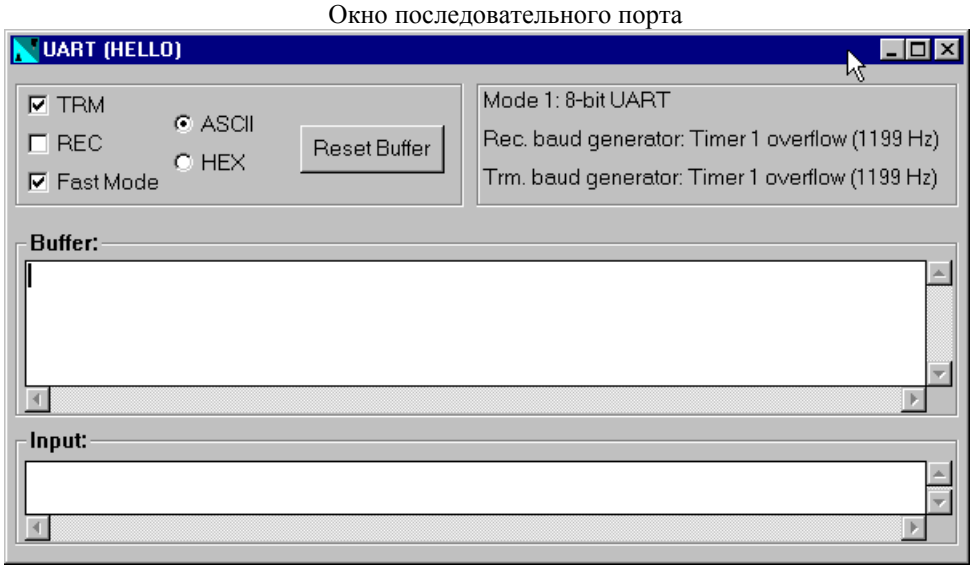


Рис. 25.

Выберите Hardware (аппаратные средства) из меню View. Выберите UART, откроется окно последовательного порта (рис. 25). В дальнейшем при работе программы здесь можно будет увидеть всё, что выводит микроконтроллер в последовательный порт.

Выберите Run из меню Debug или нажмите кнопку

Экран отладчика при выполнении программы

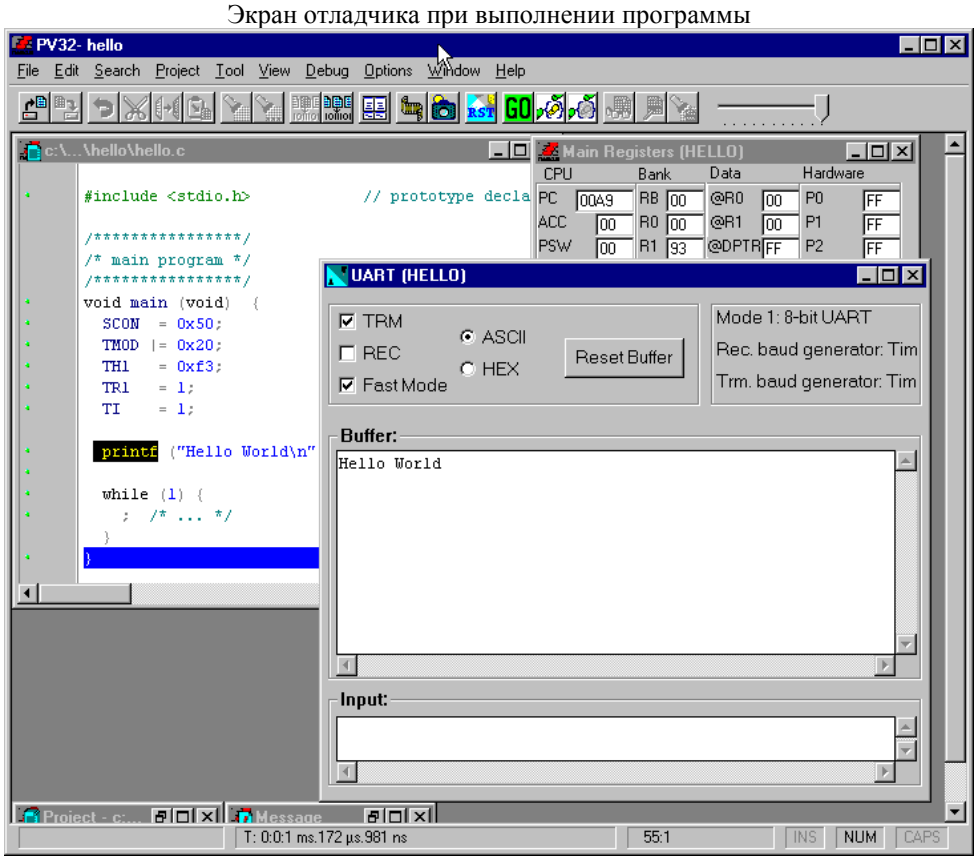
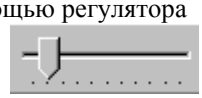
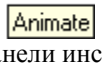


Рис. 26 показывает, как выглядит экран отладчика WinSim при выполнении программы. Обратите внимание, что в окно UART выведен текст “Hello World”.

При выводе символов в порт начинается выполнение бесконечного цикла. Вы можете остановить выполнение программы, выбрав Stop из меню Debug. С помощью регулятора



при нажатой кнопке



на панели инструментов можно менять скорость работы отладчика. Строка состояния показывает текущее реальное время.

Рис. 26.

Пошаговый режим и выход из отладчика

Вы можете использовать отладчик, чтобы перемещаться по программе. Выберите Reset из меню Debug (эта команда сбросит моделируемый процессор) и выберите Step Into и Step Over из меню Debug.

Команды Step позволяют “шагать” по каждой строке исходного текста. Текущая команда высвечивается на каждом шаге. Step Into позволяет войти в вызываемую функцию, Step Over – перешагнуть через неё, не входя во внутрь.

Проделайте эти операции.

Для завершения работы с отладчиком в любой момент времени Вы можете выбрать Terminate из меню Debug и возвратиться в режим редактирования.

Следующий шаг

Обратите внимание, что в режиме отладки на экране видны ещё два окна. Первое – окно кода (рис. 27), где в пошаговом режиме параллельно с исходным текстом на языке С идёт трассировка текста на ассемблере.

Прокрутите окно кода и изучите ассемблерный аналог исходного текста. С символов “##” начинаются строки, с помощью которых легко сопоставить ассемблерный текст и текст на языке С. Обратите внимание на то, сколько кода пришлось бы написать, если проектировать программу на ассемблере.

Окно кода

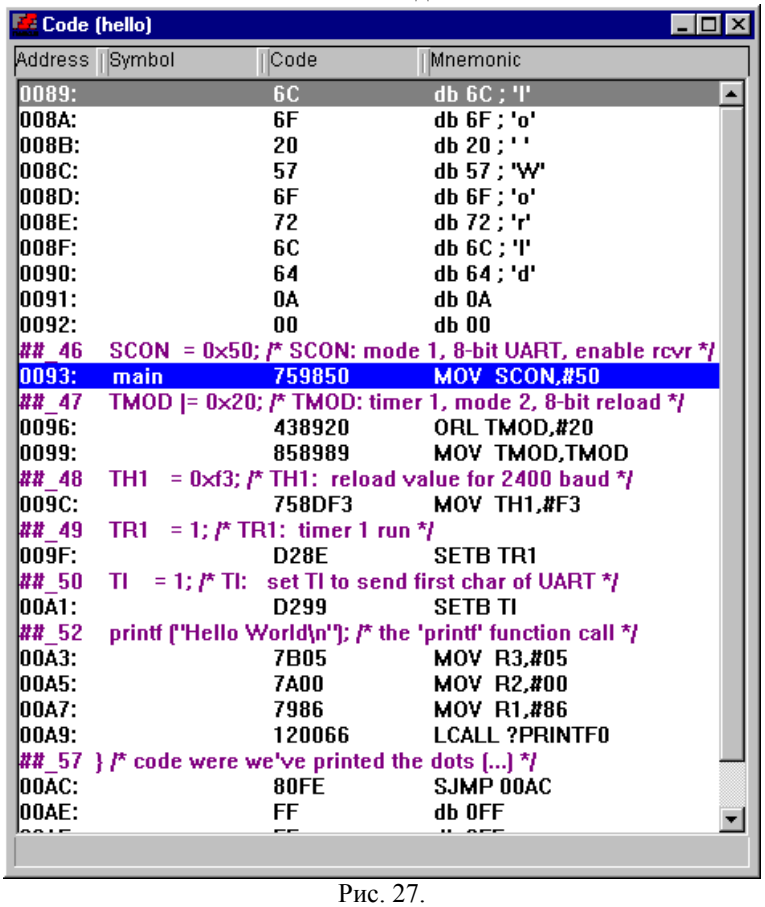


Рис. 27.

Диалог опций проекта

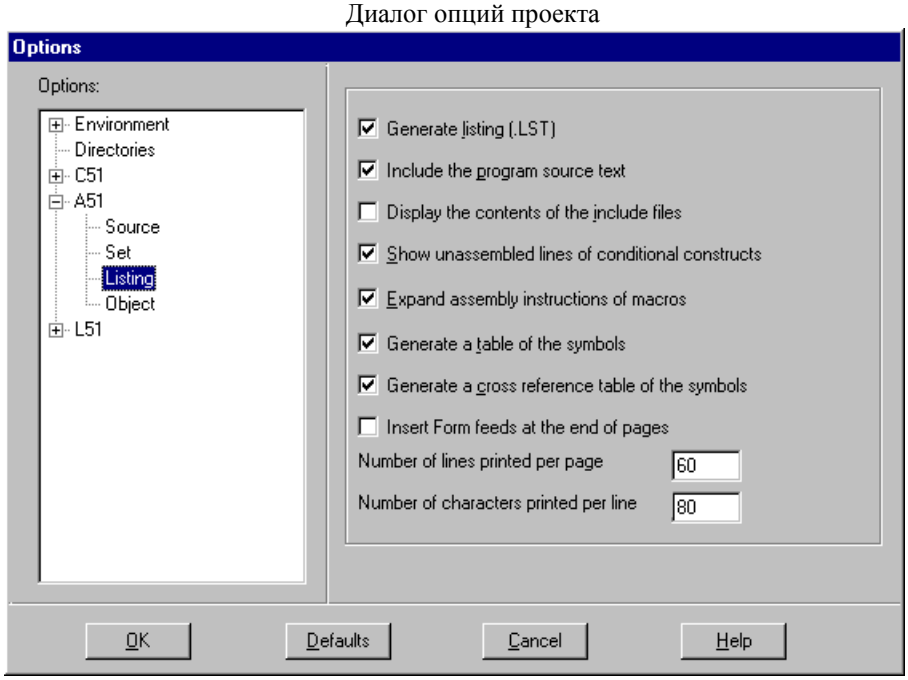


Рис. 28.

Ассемблерный аналог текста сохраняется в файле hello.lst, если в опциях проекта (Project из меню Options) отмечено Generate Listing (рис. 28). Здесь же можно указать, какую информацию включать в листинг.

Изучите смысл других опций проекта в разделах Environment, C51, A51, L51. Откройте файл листинга (рис. 29) с помощью View listing из меню View.

Изучите и постарайтесь понять содержание разделов файла листинга.

Второе окно, которое присутствует на экране во время отладки, – Main Registers (рис. 30).

В этом окне постоянно отображается текущее состояние всех программно-доступных регистров микроконтроллера. Более того, содержимое регистров можно менять во время отладки.

С помощью пункта Data dump из меню View можно посмотреть содержимое памяти различного типа в режиме отладки. Попробуйте это сделать.

Окно файла листинга

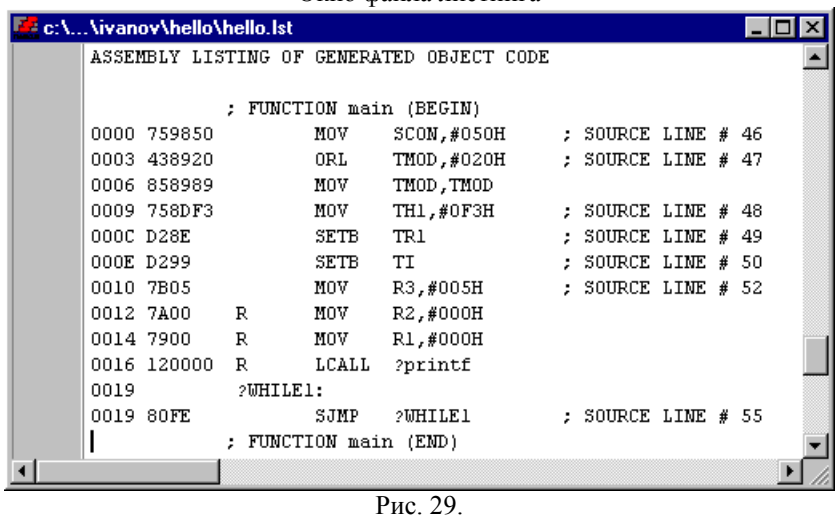


Рис. 29.

Окно регистров

Рис. 30.

III ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА ДЛЯ ПРОГРАММИРОВАНИЯ И ОТЛАДКИ МК-СИСТЕМ

1. Дополнительные аппаратные средства.

Программаторы ПЗУ предназначены для записи исполнительного кода программы в различные варианты ИС ПЗУ. Программные средства обслуживания программаторов имеют интуитивно понятный интерфейс взаимодействия с оператором и воспринимают несколько десятков форматов представления исполнительного файла.

2. Средства диагностики неисправностей.

В этом разделе рассматриваются наиболее характерные методы и классы приборов для диагностики неисправностей МК систем. Из всех видов приборов можно выделить пять основных классов: 1). - **логические анализаторы**; 2). - **сравнивающие тестеры**; 3). - **внутрисхемные эмуляторы**; 4). - **сигнатурные анализаторы**; 5). - **тестеры с управляемым пробником**. Эта классификация условна; для каждого класса характерно наличие различных модификаций методов и вариантов реализации.

Логические анализаторы.

Для них характерны следующие особенности: 1) - возможность наблюдения и обработки сигналов одновременно по нескольким каналам; 2) - нормирование входных сигналов по логическим уровням; 3) - возможность реализации различных вариантов запуска развертки; 4) - возможность регистрации предельно коротких по сравнению с периодом повторения сигнала импульсных помех; 5) - возможность сравнения полученных данных с опорными. Отдельные из перечисленных возможностей могут отсутствовать у конкретного логического анализатора.

Из всего многообразия логических анализаторов можно выделить **три группы:** [1] - анализаторы логических состояний; [2] - анализаторы логических временных диаграмм; [3] - микропроцессорные анализаторы. Такие анализаторы отличаются способами визуального наблюдения, числом входных каналов и степенью эффективности при проверке цифровых схем с обычной логикой и схем на основе микропроцессоров.

Для **анализаторов логических состояний** характерно двоичное представление данных. Они рассчитаны на работу с тактируемыми логическими схемами, в которых изменение логических состояний происходит по фронту синхросигнала, вырабатываемого схемой. Логический анализатор лишь отображает это изменение в двоичном представлении. Наиболее типичными представителями такого класса являются анализаторы логических состояний 1600A, 1607A фирмы Hewlett-Packard и т.п..

Логическими анализаторами невозможно обнаружить кратковременные импульсные помехи, разновременность прихода сигналов и перемещающиеся нарушения синхронизации. Этих недостатков лишены **анализаторы логических временных диаграмм**. В таких анализаторах стробирование данных при передаче в память осуществляется с более высокой частотой, чем тактовая частота проверяемой схемы, благодаря чему, можно получить более точную информацию о сигналах на входных шинах. Анализаторы особенно полезны для тестирования асинхронных схем. Примером таких приборов могут служить анализаторы 8200, 9100D, 920 фирмы Gould/Biomation.

Микропроцессорные анализаторы разрабатываются специально для работы с микропроцессорами и имеют большие возможности визуального отображения и большее число входных каналов. Некоторые из анализаторов имеют также диалоговые схемы, что позволяет оператору управлять работой микропроцессора и контролировать результаты. Примером для этого класса логических анализаторов являются модели 1610A, 1611A фирмы Hewlett-Packard, MPA-1 фирмы Motorola.

Поиск неисправностей с ипользованием логических анализаторов проводит оператор имеющий хорошее представление о работе проверяемого устройства, умеющий интерпретировать достаточно большие массивы выходных данных и владеющий методами отыскания неисправностей. Автоматизированный поиск неисправностей не представляется возможным.

Сравнивающие тестеры.

Для сравнивающих тестеров характерно автоматическое сравнение сигналов исследуемой системы с опорными сигналами, снимаемыми с заведомо исправной системы, с выдачей результата типа "годен/не годен". К таким приборам относится цифровой испытательный осциллограф DTO-1 фирмы Gould/Biomation. Он объединяет фактически три прибора: сравнивающий тестер, логический анализатор и запоминающий осциллограф. Логический анализатор имеет один канал. Наличие запоминающего осциллографа позволяет записывать до восьми каналов путем последовательного перемещения пробника. Опорные сигналы исправной работы системы хранятся на магнитной ленте в кассетном запоминающем устройстве, и воспроизводятся в процессе испытаний. Прибор сравнивает два логических сигнала, отображает их на экране ЭЛТ и выдает результат "годен/не годен". Многофункциональность тестера DTO-1 делает его полезным при контроле различных систем.

Тестеры с управляемым пробником.

Эти приборы снимают вопрос высокой квалификации оператора. В них используется автоматизация измерений, характерная для больших испытательных систем, но реализованная в относительно портативном приборе. Примером явялется тестер PSP, разработанный фирмой Omnicomp и реализованный фирмой Gen Red как модель 2225. Тестер выполнен на базе микропроцессора и включает в себя средства для проверки "годен/не годен" и средства для автоматической диагностики. Пробник, управляемый микропроцессором, буквенно-цифровая индикация позволяют быстро изолировать неисправные компоненты даже малоопытному оператору. Для таких приборов характерно наличие программирования испытаний, высокоскоростной диагностической проверки печатных плат. В приборе, кроме программы испытаний, должны хранится "образ" схемы, т.е. описание соединений компонентов испытуемой платы, и таблица ожидаемых реакций для каждого узла. Тестеры с управляемым пробником значительно упрощают процесс поиска неисправностей. Однако такие приборы сложны и дорогостоящи, требуют больших программных затрат перед работой по диагностике.

В приведенных выше диагностических приборах сравнение длинных последовательностей в контролируемых точках с опорными последовательностями всегда выполняется автоматически либо оператором. Поиск неисправностей связан со скоростью анализа этих последовательностей. Поиск неисправностей можно сделать более эффективным, если укоротить контролируемые последовательности. Для этой цели предлагается использовать метод переходного счета либо метод сигнатурного анализа ошибок.

В методе цифрового счета осуществляется подсчет числа переходов цифрового сигнала из одного логического состояния в другое. При подсчете переходов предполагается нулевое значение начального состояния счетчика. Счетчик производит суммирование по заданному модулю. Метод переходного счета при заданной длине контролируемой последовательности дает худшие вероятностные характеристики обнаружения ошибок в сравнении с методом сигнатурного анализа.

При одном ошибочном символе входной последовательности сигнатурный анализ обеспечивает 100% обнаружение ошибки. При нескольких ошибочных символах во входной последовательности метод обеспечивает 99,998% обнаружения ошибки.

Сигнатурные анализаторы.

Сигнатурные анализаторы – наиболее дешевые и простые в применении диагностические приборы, позволяющие выполнять высококачественную и быструю проверку цифровых устройств. Они способны определять ошибку во входной последовательности произвольной длины, независимо от положения ошибок во входной последовательности.

Сигнатурный анализатор преобразует длинные последовательности двоичных сигналов, поступающие от испытываемого изделия, в короткие четырехзначные шестнадцатиричные ключевые слова -"синатуры". Кодирование осуществляется по законам циклических кодов, исправляющих ошибки. Измеренные таким образом значения сигнатур оператор сравнивает с эталонными значениями. Если в двоичной последовательности выявляется ошибка, ее прослеживают по схеме обратным ходом, просматривая вентили и запоминающие элементы, пока не удается обнаружить элемент с правильными входными, но ошибочными выходными сигналами.

Вслед за фирмой Hewlett-Packard идеи сигнатурного анализа начали использовать фирмы Millenium Systems, Paratronics, Kurz-Kasch, Phoenix Digital, Zehntel и др..

Внутрисхемные эмуляторы.

Все рассмотренные выше классы диагностической аппаратуры являются пассивными приборами, т.е. приборы, которые только принимают сигналы от исследуемого устройства. Внутрисхемные эмуляторы являются активными диагностическими приборами. Их использование весьма эффективно при отработке и проверке микропроцессорных схем.

В эмуляторах применяется внешняя по отношению к проверяемому устройству система для имитации микропроцессора. Они обеспечивают диагностику отказов при функционировании в реальном времени. Использование сменных узлов позволяет легко адаптировать прибор к конкретному типу микропроцессора.

Внутрисхемные эмуляторы не позволяют диагностировать источники неисправностей с точностью до компонента, как в случае сигнатурного анализа, при этом для обслуживания требуется персонал высокой квалификации.

Фирма Millenium Systems объединила внутрисхемную эмуляцию и сигнатурный анализ в приборе mSA,что позволило создать чрезвычайно мощное средство для диагностики цифровых микропроцессорных систем. С помощью внутрисхемной эмуляции анализатор вырабатывает цифровые коды, благодаря чему имеется возможность проверять (диагностировать) фактически неработающие схемы. Единственно, что необходимо для осуществления диагностики – функционирование генератора тактовых импульсов проверяемой системы. Микропроцессор анализатора подключается с помощью кабеля эмуляции вместо микропроцессора проверяемой системы. Контрольный мик-

ропроцессор функционирует по программе испытаний, хранящейся в ПЗУ анализатора, в соответствии с программой проверяемой системы.

Сочетание сигнатурного анализа с внутрисхемной эмуляцией обеспечивает более широкие возможности, чем любой отдельно взятый метод. Однако такие анализаторы сложны и дорогостоящи.

ЛАБОРАТОРНАЯ РАБОТА 1. «ВВЕДЕНИЕ В ПРОГРАММИРОВАНИЕ МИКРОКОНТРОЛЛЕРОВ»

Часть 1.

1. Изучите лекционный материал: изучите особенности однокристальных микроконтроллеров; ознакомтесь с архитектурой микроконтроллеров семейства Intel 8051; ознакомтесь с системой команд микроконтроллеров семейства Intel 8051.
2. На примере по методу “Быстрый старт” изучите этапы технологии разработки и отладки программ для МК семейства MCS-51. Освойте основные приёмы работы в интегрированной среде разработки ПО.
3. Для углублённого изучения возможностей интегрированной среды разработки программного обеспечения и её компонентов самостоятельно изучите содержание и смысл всех пунктов меню, кнопок инструментальной панели, окон и настроек. Для этого воспользуйтесь встроенной справочной системой, которая вызывается через меню Help. Эти знания потребуются при выполнении следующих лабораторных работ.

Часть 2.

1. Определите временные интервалы в тактах и секундах выполнения основных блоков кода программы из раздела “Быстрый старт”. Меняйте тактовую частоту осциллятора МК.
2. Определите временные интервалы в тактах и секундах выполнения основных блоков кода программы работы контроллера температуры.
3. Поварьируйте различными исполнениями программы генератора периодического отклика.