КАЗАНСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

ИНСТИТУТ ФИЗИКИ

Кафедра радиофизики

Р.Р. ЛАТЫПОВ, К.В. ПЕТРОВНИН

МИКРОКОНТРОЛЛЕРЫ X51 АРХИТЕКТУРЫ. НАЧАЛЬНОЕ ОСВОЕНИЕ

Учебно-методическое пособие

Казань – 2015

УДК 821.111.09
ББК Ш3(4)

Принято на заседании кафедры радиофизики
Протокол № 8 от 25 марта 2015 года

Рецензент:
доцент каф. радиоэлектроники, к. ф.-м. н. **Насыров И.А.**

Р.Р. Латыпов, К.В. Петровнин
Микроконтроллеры x51 архитектуры. Начальное освоение/ Р.Р. Латыпов, К.В. Петровнин.- Казань: Казан. ун-т, 2015.- 57 с.

В настоящее пособие включены описание лабораторного макета и задания, выполняемые в лаборатории, по курсам, связанным с изучением 8 битных микроконтроллеров. Целью работ является получение начальных практических навыков и умения работы с современными 8-разрядными микроконтроллерами, которые обсуждаются в лекционном курсе. В описании приводятся сведения, необходимые для выполнения практических заданий. Лабораторный макет представляет собой плату начального освоения, серийно выпускаемую производителем микроконтроллера. Пособие рекомендуется студентам института физики К(П)ФУ, обучающимся на специальностях «Радиотелекоммуникации», «Компьютерная электроника» и «Информационная безопасность».

Оглавление

1. Введение 4
2. Микропроцессорное ядро CIP-51 7
2.1. Арифметико-логическое устройство 8
2.2. Устройство управления (CONTROL LOGIC) 8
2.3. Память данных 8
2.4. Блок регистров специальных функций 10
3. Система команд x51 совместимых микроконтроллеров 13
3.1. Виды адресации 22
3.2. Арифметические операции 23
3.3. Логические операции 24
3.4. Операции пересылки данных 24
3.5. Битовые операции 25
3.6. Операции ветвления 26
4. Микроконтроллер C8051F411 29
4.1. Обработка прерываний 29
4.2. Сторожевой таймер 33
4.3. Порты ввода-вывода 36
5. Директивы компилятора 41
6. Лабораторная установка C8051F411EB 43
6.1. Введение в IDE SILABS 44
6.1.1. Макроассемблер A51 фирмы Keil 45
6.1.2. Оптимизирующий кросс-компилятор C51 фирмы Keil 45
6.1.3. Компоновщик L51 46
6.1.4. Отладчик 46
6.2. Запуск IDE Silabs и создание файла проекта 47
6.3. Быстрый старт 48
6.4. Запуск IDE Silabs и открытие файла готового проекта 49
6.5. Добавление файла с исходным текстом и его редактирование 50
7. Задачи 52
7.1. Раздел 1 52
7.2. Раздел 2 55
8. Литература 56

1. Введение

Микроконтроллерная техника является одной из наиболее динамично развивающихся областей современной вычислительной техники. Без микроконтроллеров сегодня немыслим ни один современный прибор. Микроконтроллеры широко используются в различных изделиях вычислительной, измерительной, лабораторной и научной техники; в системах управления промышленным оборудованием, транспорта и связи; в бытовой технике и других областях.

Сегодня в мире производится несколько десятков тысяч типов различных микроконтроллеров. Среди них особое место занимают так называемые x51-совместимые микроконтроллеры, т.е. микроконтроллеры, совместимые с одним из первых типов микроконтроллеров - i8051. Не смотря на почти тридцатилетнюю историю своего существования, эти микроконтроллеры и сегодня занимают лидирующее место и являются «de facto» всемирным промышленным стандартом. x51-совместимые микроконтроллеры выпускаются практически всеми известными мировыми производителями: Intel, Atmel, Maxim-Dallas, Goal Semicondactor, Hyundai, Philips, Infineon, Temic, Winbond, ICSI, ISSI, Oki Semicondactor, Sharp, STT, Cypress, Texas Instruments, STM, TDK, SiLabs и многими другими.

Популярность x51 совместимых микроконтроллеров обусловлена рядом причин, среди которых главенствующую роль играют:

- удачная и «прозрачная» архитектура;
- хорошая документированность – обилие разнообразной научно-технической литературы: рекомендаций по применению, книг, статей;
- большое количество качественного и доступного программного обеспечения: компиляторов различных языков программирования: ASM51, C++, PL/M51, Fort51 и других, а также целый ряд дизассемблеров, программных отладчиков, эмуляторов и т.п.;

- большое количество программ и библиотек для различных научно-технических задач;
- разработчиками накоплен колоссальный опыт работы с этими микроконтроллерами, изучены многие особенности их поведения, выработаны приемы и способы отладки, накоплен опыт программирования.

Сравнительно недавно, в 1999 году, в клуб производителей x51-совместимых микроконтроллеров вошла фирма SiLabs [1,5,7]. Специалисты фирмы SiLabs кардинально изменили ядро своих новых x51-совместимых микроконтроллеров, что позволило им более чем на порядок увеличить производительность. Полученное ядро CIP-51, построенное по конвейерному принципу, до 70% инструкций выполняет за один период тактовой частот, что обеспечивает повышение пиковой производительности в 12 раз по сравнению со стандартным ядром i8051. Дополнительно это микропроцессорное ядро было оснащено не менее мощными аналого-цифровыми и цифро-аналоговыми узлами, расширенной периферией, усовершенствованными средствами внутрисистемного программирования и отладки.

В состав периферии микроконтроллеров фирмы SiLabs входят:

- встроенная Flash память программ-данных (объемом от 8 до 128К);
- встроенная дополнительная оперативная память (объемом от 1 до 8К);
- расширенный обработчик прерываний (22 вектора у большинства микроконтроллеров);
- модифицированная система защиты кода;
- аппаратный охранный таймер WDT;
- монитор питания;
- встроенная развитая система тактирования, позволяющая микроконтроллеру работать как от внешнего генератора, так и со встроенным тактовым генератором, оснащенным кварцевым резонатором, конденсатором или вообще без дополнительных элементов с возможностью изменения источника тактирования «на лету».

- интерфейсы UART, SMBus (I2C), SPI, CAN, USB;
- один или два аналого-цифровых преобразователя (с разрядностью 8, 10,12 или 16 бит);
- 2 (1) аналоговых компаратора;
- источник опорного напряжения;
- до двух 12-разрядных цифро-аналоговых преобразователей;

Программирование и отладка микроконтроллеров фирмы SiLabs реализованы на интерфейсах JTAG или C2. Это позволяет не только быстро программировать Flash память, но и производить отладку программ в реальном времени.

2. Микропроцессорное ядро CIP-51

Основу структурной схемы (рис. 1) микропроцессорного ядро CIP-51 образует внутренняя двунаправленная 8-битная шина (DATA BUS), которая связывает между собой основные узлы и устройства микроконтроллера: арифметико-логическое устройство (ALU), память данных (SRAM), блок регистров специальных функций, устройство управления (CONTROL LOGIC), а также все периферийные устройства микроконтроллера (некоторые из которых рассмотрены в разделе 4) [1].

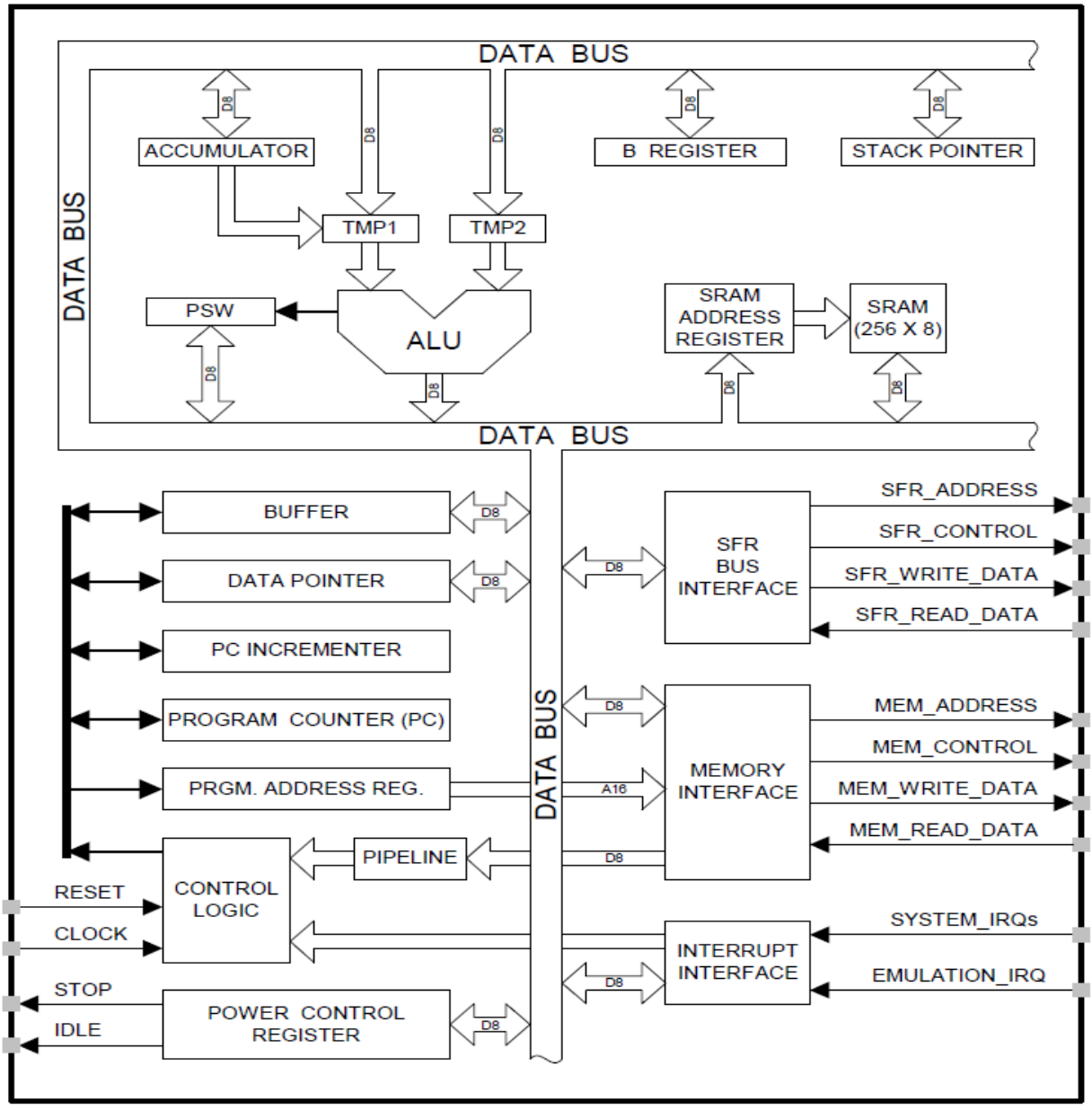

Рис 1. Упрощенная блок-схема микропроцессорного ядра CIP-51

2.1. Арифметико-логическое устройство

Арифметико-логическое устройство (ALU) разрядностью 8-бит может выполнять арифметические операции сложения, вычитания, умножения и деления; логические операции И, ИЛИ, исключающее ИЛИ, а также операции циклического сдвига, сброса, инвертирования и т.п. К входам подключены программно-недоступные регистры TMP1 и TMP2, предназначенные для временного хранения операндов, схема десятичной коррекции и схема формирования признаков результата операции (PSW) [5,6,7,9].

Важной особенностью ALU является его способность оперировать не только байтами, но и битами. Отдельные программно-доступные биты могут быть установлены, сброшены, инвертированы, переданы, проверены и использованы в логических операциях. Эта способность достаточно важна, поскольку для управления объектами часто применяются алгоритмы, содержащие операции над входными и выходными булевыми переменными, реализация которых средствами обычных микропроцессоров сопряжена с определенными трудностями.

Для сохранения результатов вычисления и в качестве одного из операндов используется специальный регистр аккумулятор (A, ACCUMULATOR), так же в случае операций умножения и деления используется расширение аккумулятора – регистр B.

2.2. Устройство управления (CONTROL LOGIC)

Устройство управления (CU) на основе сигналов синхронизации (CLOCK) формирует машинный цикл, обеспечивает функционирование системы сброса (RESET), программного счетчика (PC), системы выбора команд и увеличения значения программного счетчика (PC INCREMENTER), управление питанием (POWER CONTROL REGISTER) и системой прерываний (INTERRUPT INTEFACE).

2.3. Память данных

Размер памяти данных(SRAM) составляет 256 байт, младшие 128 (по адресу) ячеек используются только как память данных, и обращение к ним

может быть произведено любым способом адресации. Старшие 128 байт могут быть адресованы только косвенным образом, потому что делят адреса с регистрами специальных функций. На рис. 2 показана блочная структура памяти данных.

<table>
<tr><td>Адрес</td><td></td><td></td></tr>
<tr><td>0xFF

0x80</td><td>Верхние 128 байт памяти RAM (только косвенная адресация)</td><td>Регистры специальных функций SFR (только прямая адресация)</td></tr>
<tr><td>0x7F

0x30</td><td>Память данных RAM байт адресуемая</td><td rowspan="3">Нижние 128 байт памяти RAM (прямая и косвенная адресация)</td></tr>
<tr><td>0x2F

0x20</td><td>Память данных RAM бит адресуемая</td></tr>
<tr><td>0x1F

0x00</td><td>Регистры общего назначения GPO</td></tr>
</table>

Рис.2. Блочная структура памяти данных

На рис. 3 более подробно изображены младшие 128 байт памяти. Это связано с тем, что организация этого блока памяти имеет ряд специально выделенных областей. Первая специальная область это – блок регистров общего назначения, состоящий из 4 банков и занимающий адреса с 0x00 до 0x1F. Банки адресов могут переключаться, т.е. в один момент времени активным может быть только один банк. В каждом банке присутствуют регистры от R0 до R7, к которым можно обращаться как по символическому имени, так и по прямому адресу. Вторая специальная область это – блок побитно адресуемых ячеек, занимающий адреса с 0x20 до 0x2F. Внутри этого блока каждый бит имеет свой собственный адрес и обращение к нему можно производить отдельно от всего байта.

<table>
<tr><td>Адрес байта</td><td colspan="8">Адрес бита</td></tr>
<tr><td>7F</td><td colspan="8" rowspan="4">Память данных общего назначения RAM
байт адресуемая</td></tr>
<tr><td></td></tr>
<tr><td></td></tr>
<tr><td>30</td></tr>
<tr><td>2F</td><td>7F</td><td>7E</td><td>7D</td><td>7C</td><td>7B</td><td>7A</td><td>79</td><td>78</td></tr>
<tr><td>2E</td><td>77</td><td>76</td><td>75</td><td>74</td><td>73</td><td>72</td><td>71</td><td>70</td></tr>
<tr><td>2D</td><td>6F</td><td>6E</td><td>6D</td><td>6C</td><td>6B</td><td>6A</td><td>69</td><td>68</td></tr>
<tr><td>2C</td><td>67</td><td>66</td><td>65</td><td>64</td><td>63</td><td>62</td><td>61</td><td>60</td></tr>
<tr><td>2B</td><td>5F</td><td>5E</td><td>5D</td><td>5C</td><td>5B</td><td>5A</td><td>59</td><td>58</td></tr>
<tr><td>2A</td><td>57</td><td>56</td><td>55</td><td>54</td><td>53</td><td>52</td><td>51</td><td>50</td></tr>
<tr><td>29</td><td>4F</td><td>4E</td><td>4D</td><td>4C</td><td>4B</td><td>4A</td><td>49</td><td>48</td></tr>
<tr><td>28</td><td>47</td><td>46</td><td>45</td><td>44</td><td>43</td><td>42</td><td>41</td><td>40</td></tr>
<tr><td>27</td><td>3F</td><td>3E</td><td>3D</td><td>3C</td><td>3B</td><td>3A</td><td>39</td><td>38</td></tr>
<tr><td>26</td><td>37</td><td>36</td><td>35</td><td>34</td><td>33</td><td>32</td><td>31</td><td>30</td></tr>
<tr><td>25</td><td>2F</td><td>2E</td><td>2D</td><td>2C</td><td>2B</td><td>2A</td><td>29</td><td>28</td></tr>
<tr><td>24</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td></tr>
<tr><td>23</td><td>1F</td><td>1E</td><td>1D</td><td>1C</td><td>1B</td><td>1A</td><td>19</td><td>18</td></tr>
<tr><td>22</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td></tr>
<tr><td>21</td><td>0F</td><td>0E</td><td>0D</td><td>0C</td><td>0B</td><td>0A</td><td>09</td><td>08</td></tr>
<tr><td>20</td><td>07</td><td>06</td><td>05</td><td>04</td><td>03</td><td>02</td><td>01</td><td>00</td></tr>
<tr><td>1F</td><td colspan="8" rowspan="2">Банк 3 регистров общего назначения (R0-R7)</td></tr>
<tr><td>18</td></tr>
<tr><td>17</td><td colspan="8" rowspan="2">Банк 2 регистров общего назначения (R0-R7)</td></tr>
<tr><td>10</td></tr>
<tr><td>0F</td><td colspan="8" rowspan="2">Банк 1 регистров общего назначения (R0-R7)</td></tr>
<tr><td>08</td></tr>
<tr><td>07</td><td colspan="2">R7</td><td colspan="6" rowspan="8">Банк 0 регистров общего назначения (по умолчанию)</td></tr>
<tr><td>06</td><td colspan="2">R6</td></tr>
<tr><td>05</td><td colspan="2">R5</td></tr>
<tr><td>04</td><td colspan="2">R4</td></tr>
<tr><td>03</td><td colspan="2">R3</td></tr>
<tr><td>02</td><td colspan="2">R2</td></tr>
<tr><td>01</td><td colspan="2">R1</td></tr>
<tr><td>00</td><td colspan="2">R0</td></tr>
</table>

Рис. 3. Структура младших 128 байтов памяти данных

2.4. Блок регистров специальных функций

Блок регистров специальных функций (special function registers, SFRs) предоставляет доступ к периферийным узлам в x51-совместимых микроконтроллерах. Эти регистры (SFR) располагаются в прямо адресуемом адресном пространстве памяти данных - 0x80-0xFF. Для обеспечения совместимости между микроконтроллерами различных производителей некоторые SFR регистры располагаются во всех микроконтроллерах по одним и тем же адресам и имеют одинаковое функциональное назначение. Все вышесказанное в полной мере относится и к микроконтроллерам фирмы Silicon Laboratories, имеющим усовершенствованное ядро CIP-51. С помощью этих регистров осуществляется управление и обмен данными между ресурсами ядра CIP-51 и периферией.

Регистры с адресами, оканчивающимися на 0x0 или 0x8 (например, P0, TCON, P1, SCON, IE и т.д.) являются и бит-адресуемыми и байт-адресуемыми. Все остальные регистры SFR являются только байт-адресуемыми. Неиспользуемые адреса в адресном пространстве SFR зарезервированы. Использование этих адресов приводит к неопределенному результату и нежелательно. В *таблице 1* приведены адреса стандартных регистров (SFR) для x51 архитектуры, а в *таблице 2* – краткое описание каждого из них.

Таблица 1

Стандартные регистры x51 МК

Название регистра	Адрес регистра	Описание функционального назначения регистра
ACC	0xE0	Аккумулятор
B	0xF0	Регистр B
DPH	0x83	Старший байт указателя данных
DPL	0x82	Младший байт указателя данных
P0	0x80	Выходной регистр Port 0
PI	0x90	Выходной регистр Port 1
P2	0xA0	Выходной регистр Port 2
P3	0xB0	Выходной регистр Port 3
PSW	0xD0	Слово состояния программы
SP	0x81	Указатель стека

Таблица 2

Краткое описание стандартных регистров x51 МК

Название регистра	Краткое описание
ACC	Регистр аккумулятора. Регистр используется при арифметических операциях.
B	Регистр B. Дополнительный регистр, используемый как второй аккумулятор при арифметических операциях.
DPH	Регистр старшего байта указателя данных. Старший байт 16-битного регистра указателя данных DPTR, используемого для косвенной адресации оперативной памяти RAM и Flash памяти.
DPL	Регистр младшего байта указателя данных. Младший байт 16-битного регистра указателя данных DPTR, используемого для косвенной адресации оперативной памяти RAM и Flash памяти.
P0	Регистр двунаправленного порта 0. Биты 7-0: P0.[7-0]

<table>
<tr><td></td><td>При записи:
0 - логический низкий уровень.
1 - логический высокий уровень или высокоимпедансное состояние.
При чтении:
0 - если соответствующий вывод установлен в логический 0.
1 - если соответствующий вывод установлен в логическую 1.</td></tr>
<tr><td>P1</td><td>Регистр порта 1. Чтение, запись аналогично порту 0.</td></tr>
<tr><td>P2</td><td>Регистр порта 2. Чтение, запись аналогично порту 0.</td></tr>
<tr><td>P3</td><td>Регистр порта 3. Чтение, запись аналогично порту 0.</td></tr>
<tr><td>PSW</td><td>Регистр состояния
<table><tr><td>Бит 7</td><td>Бит 6</td><td>Бит 5</td><td>Бит 4</td><td>Бит 3</td><td>Бит 2</td><td>Бит 1</td><td>Бит 0</td></tr></table>
Бит 7: CY - флаг переноса, устанавливается если результат последней арифметической операции приводит к появлению переноса (Carry) или заема (Borrow). Сбрасывается в 0 после следующей арифметической операции.
Бит 6: Auxiliary Carry Flag - дополнительный флаг переноса, используемый при операции десятичной коррекции. Сбрасывается в 0 после следующей арифметической операции.
Бит 5: User Flag 0 - пользовательский флаг 0 общего назначения.
Биты 4-3: RS1-RS0 - Register Bank Select - биты выбора банка регистров. Биты выбирают один из четырех банков регистров:
00 - банк 0 с адресами 0x00-0x07
01 - банк 1 с адресами 0x08-0x0F
10 - банк 2 с адресами 0x10-0x17
11 - банк 3 с адресами 0x18-0x1F
Бит 2: OV - Overflow Flag - флаг переполнения, устанавливается в 1, если последняя арифметическая операция вызвала перенос (при сложении), заем (при вычитании) или переполнение (при умножении или делении). Сбрасывается в 0 после следующей арифметической операции.
Бит 1: User Flag 1 - пользовательский флаг 1 общего назначения.
Бит 0: Parity Flag (только чтение) - бит четности. Устанавливается в 1, если сумма восьми битов в аккумуляторе - нечетная, и обнуляется, если сумма четная.</td></tr>
<tr><td>SP</td><td>Регистр указателя стека. Однобайтный регистр указателя стека содержит текущее значение верхнего адреса стека.</td></tr>
</table>

3. Система команд x51 совместимых микроконтроллеров

Список команд ядра CIP-51 [1,3,4,8], количество байтов, занимаемое каждой командой, и количество тактов требуемых на выполнение, приведены в *таблице 3*. При составлении этой таблицы были приняты следующие умолчания:

- фразу «*прибавить регистр к A*» следует понимать «*прибавить содержимое регистра к содержимому аккумулятора*»;
- результаты любых действий заносятся в аккумулятор (A), если не указано иное;
- *регистр* (**Rn)** – любой из 8 регистров R0-R7 выбранного в данный момент банка;
- *косвенно адресуемый байт* (**@Ri**) – ячейка памяти, адрес которой записан в регистре R0 или R1, может указывать на любой адрес памяти данных от 0x00 до 0xFF;
- *прямо адресуемый байт* (**direct**) – это 8-битный адрес из адресного пространства памяти данных, от 0x00 до 0x7F он указывает на память данных, от 0x80 до 0xFF указывает на регистры специальных функций (SFR);
- *константа* (**#data**) – непосредственно адресуемая 8-битная константа;
- *константа 16 бит* (#**data16**) – непосредственно адресуемая 16-битная константа;
- *бит* (**bit**) – прямо адресуемый бит из побитово адресуемого пространства или бит регистра специальных функций (SFR);
- *метка* (**rel**) – 8-битная знаковая константа, служащая для реализации всех функций условных переходов, может принимать значения от -128 до +127;
- *11 битный адрес* (**addr11**) – используется в командах ACALL и AJMP для коротких переходов в пределах 2кбайт памяти команд;

- *16 битный адрес* (**addr16**) – полный адрес используется в командах LCALL и LJMP для переходов в любое место памяти команд.

Таблица.3

Список команд CIP-51

№	Мнемоническое описание	Краткое описание	Кол-во байтов	Кол-во тактов
Арифметические операции (раздел 3.2)				
1	ADD A, Rn	Прибавить регистр к A	1	1
2	ADD A, direct	Прибавить прямо адресуемый байт к A	2	2
3	ADD A, @Ri	Прибавить косвенно адресуемый байт к A	1	2
4	ADD A, #data	Прибавить константу к A	2	2
5	ADDC A, Rn	Прибавить регистра к A с учетом флага переноса	1	1
6	ADDC A, direct	Прибавить прямо адресуемый байт к A с учетом флага переноса	2	2
7	ADDC A, @Ri	Прибавить косвенно адресуемый байт к A с учетом флага переноса	1	2
8	ADDC A, #data	Прибавить константу к A с учетом флага переноса	2	2
9	SUBB A, Rn	Вычесть регистр из A с заемом из флага переноса	1	1
10	SUBB A, direct	Вычесть прямо адресуемый байт из A с заемом из флага переноса	2	2
11	SUBB A, @Ri	Вычесть косвенно адресуемый байт из A с заемом из флага переноса	1	2
12	SUBB A, #data	Вычесть константу из A с заемом из	2	2

		флага переноса		
13	INC A	Инкрементировать А (увеличить на единицу)	1	1
14	INC Rn	Инкрементировать регистр	1	1
15	INC direct	Инкрементировать прямо адресуемый байт	2	2
16	INC @Ri	Инкрементировать косвенно адресуемый байт	1	2
17	DEC A	Декрементировать А (уменьшить на единицу)	1	1
18	DEC Rn	Декрементировать регистр	1	1
19	DEC direct	Декрементировать прямо адресуемый байт	2	2
20	DEC @Ri	Декрементировать косвенно адресуемый байт	1	2
21	INC DPTR	Инкрементировать регистр DPTR	1	1
22	MUL AB	Умножить А на В	1	4
23	DIV AB	Разделить А на В	1	8
24	DA A	Двоично десятичная коррекция А	1	1
Логические операции (раздел 3.3)				
25	ANL A, Rn	Логическое «И» регистра и А	1	1
26	ANL A, direct	Логическое «И» прямо адресуемого байта и А	2	2
27	ANL A, @Ri	Логическое «И» косвенно адресуемого байта и А	1	2
28	ANL A, #data	Логическое «И» константы и А	2	2
29	ANL direct, A	Логическое «И» А и прямо адресуемого байта. Результат заносится в прямо адресуемый байт.	2	2

30	ANL direct, #data	Логическое «И» прямо адресуемого байта и константы. Результат заносится в прямо адресуемый байт.	3	3
31	ORL A, Rn	Логическое «ИЛИ» регистра и А	1	1
32	ORL A, direct	Логическое «ИЛИ» прямо адресуемого байта и А	2	2
33	ORL A, @Ri	Логическое «ИЛИ» косвенно адресуемого байта и А	1	2
34	ORL A, #data	Логическое «ИЛИ» константы и А	2	2
35	ORL direct, A	Логическое «ИЛИ» А и прямо адресуемого байта. Результат заносится в прямо адресуемый байт.	2	2
36	ORL direct, #data	Логическое «ИЛИ» прямо адресуемого байта и константы. Результат заносится в прямо адресуемый байт.	3	3
37	XRL A, Rn	Логическое «исключающее ИЛИ» регистра и А	1	1
38	XRL A, direct	Логическое «исключающее ИЛИ» прямо адресуемого байта и А	2	2
39	XRL A, @Ri	Логическое «исключающее ИЛИ» косвенно адресуемого байта и А	1	2
40	XRL A, #data	Логическое «исключающее ИЛИ» константы и А	2	2
41	XRL direct, A	Логическое «исключающее ИЛИ» А и прямо адресуемого байта. Результат заносится в прямо адресуемый байт.	2	2
42	XRL direct, #data	Логическое «исключающее ИЛИ» прямо адресуемого байта и константы. Результат заносится в прямо	3	3

		адресуемый байт.		
43	CLR A	Очистка A	1	1
44	CPL A	Побитовая инверсия A	1	1
45	RL A	Циклический сдвиг A влево	1	1
46	RLC A	Циклический сдвиг A влево с использованием флага переноса	1	1
47	RR A	Циклический сдвиг A вправо	1	1
48	RRC A	Циклический сдвиг A вправо с использованием флага переноса	1	1
49	SWAP A	Обмен тетрадами в A	1	1
Операции пересылки данных (раздел 3.4)				
50	MOV A, Rn	Скопировать регистра в A	1	1
51	MOV A, direct	Скопировать прямо адресуемый байт в A	2	2
52	MOV A, @Ri	Скопировать косвенно адресуемый байт в A	1	2
53	MOV A, #data	Скопировать константу в A	2	2
54	MOV Rn, A	Скопировать A в регистр	1	1
55	MOV Rn, direct	Скопировать прямо адресуемый байт в регистр	2	2
56	MOV Rn, #data	Скопировать константу в регистр	2	2
57	MOV direct, A	Скопировать A в прямо адресуемого байт	2	2
58	MOV direct, Rn	Скопировать регистр в прямо адресуемый байт	2	2
59	MOV direct, direct	Скопировать прямо адресуемый байт в прямо адресуемый байт	3	3
60	MOV direct, @Ri	Скопировать косвенно адресуемого байта памти в прямо адресуемый байт	2	2

61	MOV direct, #data	Скопировать константу в прямо адресуемый байт	3	3
62	MOV @Ri, A	Скопировать A в косвенно адресуемый байт	1	2
63	MOV @Ri, direct	Скопировать прямо адресуемого байта в косвенно адресуемый байт	2	2
64	MOV @Ri, #data	Скопировать константу в косвенно адресуемый байт	2	2
65	MOV DPTR, #data16	Скопировать в DPTR 16 битную константу	3	3
66	MOVC A, @A+DPTR	Скопировать из памяти команд в A косвенно адресуемый в DPTR байт, с адресом DPTR+A	1	4 to 72
67	MOVC A, @A+PC	Скопировать из памяти команд в A косвенно адресуемый в PC байт, с адресом PC+A	1	4 to 72
68	MOVX A, @Ri	Скопировать из внешней памяти данных в A косвенно адресуемый в регистре байт (8 битный адрес)	1	3
69	MOVX @Ri, A	Скопировать из A в косвенно адресуемый в регистре байт внешней памяти данных (8 битный адрес)	1	3
70	MOVX A, @DPTR	Скопировать из внешней памяти данных в A косвенно адресуемый в DPTR байт (16 битный адрес)	1	3
71	MOVX @DPTR, A	Скопировать из A в косвенно адресуемый в DPTR байт внешней памяти данных (16 битный адрес)	1	3
72	PUSH direct	Сохранить прямо адресуемый байт в	2	2

		стеке		
73	POP direct	Извлечь из стека в прямо адресуемый байт	2	2
74	XCH A, Rn	Поменять содержимое регистра и A	1	1
75	XCH A, direct	Поменять содержимое прямо адресуемый байта и A	2	2
76	XCH A, @Ri	Поменять содержимое косвенно адресуемого байта и A	1	2
77	XCHD A, @Ri	Поменять младшую тетраду косвенно адресуемого байта и A	1	2
Битовые операции (раздел 3.5)				
78	CLR C	Очистить флаг переноса	1	1
79	CLR bit	Очистить бит	2	2
80	SETB C	Установить флаг переноса	1	1
81	SETB bit	Установить бит	2	2
82	CPL C	Инвертировать флаг переноса	1	1
83	CPL bit	Инвертировать бит	2	2
84	ANL C, bit	Логическое «И» бита и флага переноса	2	2
85	ANL C, /bit	Логическое «И» инвертированного бита и флага переноса	2	2
86	ORL C, bit	Логическое «ИЛИ» бита и флага переноса	2	2
87	ORL C, /bit	Логическое «ИЛИ» инвертированного бита и флага переноса	2	2
88	MOV C, bit	Копирование бита в флаг переноса	2	2
89	MOV bit, C	Копирование флага переноса в бит	2	2
90	JC rel	Переход на метку, если флаг переноса установлен	2	2/4
91	JNC rel	Переход на метку, если флаг переноса	2	2/4

		не установлен		
92	JB bit, rel	Переход на метку, если бит установлен	3	3/5
93	JNB bit, rel	Переход на метку, если бит не установлен	3	3/5
94	JBC bit, rel	Переход на метку, если бит установлен и его очистка	3	3/5
Операции ветвления (раздел 3.6)				
95	ACALL addr11	Абсолютный вызов подпрограммы (11 битный адрес)	2	4
96	LCALL addr16	Длинный вызов подпрограммы (16 битный адрес)	3	5
97	RET	Возврат из подпрограммы	1	6
98	RETI	Возврат из прерывания	1	6
99	AJMP addr11	Абсолютный переход (11 битный адрес)	2	4
100	LJMP addr16	Длинный переход (16 битный адрес)	3	5
101	SJMP rel	Короткий переход на метку	2	4
102	JMP @A+DPTR	Переход по косвенно адресуемому в DPTR+A адресу	1	4
103	JZ rel	Переход на метку, если A равен 0	2	2/4
104	JNZ rel	Переход на метку, если A не равен 0	2	2/4
105	CJNE A, direct, rel	Переход на метку, если A не равен прямо адресуемому байту	3	3/5
106	CJNE A, #data, rel	Переход на метку, если A не равен константе	3	3/5

107	CJNE Rn, #data, rel	Переход на метку, если регистр не равен константе	3	3/5
108	CJNE @Ri, #data, rel	Переход на метку если косвенно адресуемый байт не равен константе	3	4/6
109	DJNZ Rn, rel	Декрементирование регистра и переход на метку, если регистр не равен 0	2	2/4
110	DJNZ direct, rel	Декрементирование прямо адресуемого байта и переход на метку, если регистр не равен 0	3	3/5
111	NOP	Отсутствие операции	1	1

3.1. Виды адресации

Существуют следующие способы адресации операндов – данных, участвующих в операциях [8,9]:

- неявная;
- регистровая (прямая регистровая);
- прямая (байтовая и битовая);
- непосредственная;
- косвенная;
- относительная;
- базово-индексная.

При *неявной адресации* информацию об адресе операнда, участвующего в операции, устройство управления получает из кода операции команды. Чаще всего так адресуются аккумулятор и флаг переноса, например, INC A, CLR C.

При *регистровой адресации* команда содержит трехразрядный прямой адрес одного из восьми рабочих регистров текущего банка регистров общего

назначения (R0-R7). Примеры команд с регистровой адресацией: MOV A, Rn; XCH A, Rn.

Прямая байтовая адресация применяется для обращения к ячейкам памяти данных и к регистрам специальных функций. В этом случае команда включает в себя прямой 8-разрядный адрес операнда, обозначаемый direct, например: ADD A, direct; DEC direct и др.

Прямая битовая адресация используется для обращения к отдельно адресуемым 128 битам памяти данных (рис. 3) и к отдельно адресуемым битам регистров специальных функций. При этом команда содержит прямой 8-разрядный адрес бита, участвующего в операции. Этот адрес обозначается bit, например: SET bit; MOV C, bit и др.

При *непосредственной адресации* операнд входит в саму команду и извлекается из памяти при выполнении команды вслед за ее кодом операции. Непосредственный операнд обозначается # data, # data16, например: MOV A, # data; MOV DPTR, # data16 и др.

При *косвенной адресации* в команде содержится ссылка на регистр, в котором содержится адрес операнда. Эта адресация может использоваться для обращения к ячейкам внутренней памяти данных или внешней памяти данных. В качестве регистров-указателей служат регистры R0, R1 выбранного рабочего (текущего) банка регистров общего назначения. Это, например, команды MOV A, @Ri; MOV direct, @Ri и др.

В качестве регистров указателей внешней памяти данных применяются те же R0 и R1, что позволяет выбрать одну из 256 ячеек внешней памяти, либо 16-разрядный регистр-указатель данных (DPTR), который обеспечивает адресацию одной из 65536 = 64 К восьмиразрядный ячеек внешней памяти данных. Например, команды MOVX A, @Ri; MOVX @DPTR, A и др.

Относительная адресация применяется в командах условных переходов. При этом в команде задается 8-разрядное смещение относительно адреса самой команды. Смещение воспринимается как число со знаком, представленное в дополнительном коде. Это позволяет осуществлять переходы на +127 байт

вперед и на –128 байт назад относительно адреса следующей команды. Например, команды JNZ rel; DJNZ ad, rel и др., где rel – 8-битное смещение.

При *базово-индексной адресации* адрес операнда вычисляется как сумма 16-разрядного базового адреса, содержащегося в регистрах DPTR или PC, и 8-битного индекса, загружаемого в аккумулятор. Это позволяет обращаться к элементам таблиц, массивов и т.д. Например, команды MOVC A, @A+DPTR; MOVC A, @A+PC.

3.2. Арифметические операции

В наборе команд (таблица 3) имеются следующие арифметические операции: сложение, сложение с учетом флага переноса, вычитание с заемом, инкрементирование, декрементирование, сравнение, десятичная коррекция, умножение и деление.

В АЛУ производятся действия над целыми числами. В двухоперандных операциях: сложение (ADD), сложение с переносом (ADDC) и вычитание с заемом (SUBB) аккумулятор является первым операндом и принимает результат операции. Вторым операндом может быть регистр выбранного банка рабочих регистров, регистр внутренней памяти данных с косвенной и прямой адресацией или байт непосредственных данных. Указанные операции влияют на ***флаги***: ***переполнения, переноса, промежуточного переноса*** и ***четности –*** в слове состояния процессора (PSW).

Использование разряда переноса позволяет многократно повысить точность при операциях сложения (ADDC) и вычитания (SUBB).

Выполнение операций сложения и вычитания с учетом знака может быть осуществлено с помощью флага переполнения (OV) регистра PSW. Флаг промежуточного переноса (AC) обеспечивает выполнение арифметических операций в двоично-десятичном коде и используется командой десятичной коррекции DAA.

Операции инкрементирования и декрементирования на флаги не влияют.

Операции сравнения не влияют ни на операнд назначения, ни на операнд источника, но они влияют на флаг переноса.

3.3. Логические операции

АЛУ дает возможность выполнять логические операции над байтовыми и битовыми операндами. В таблицу 3 включены команды выполнения логических операций как над содержимым 8-разрядных данных, так и над битовыми операндами.

Система команд позволяет реализовать логические операции: "И", "ИЛИ", "ИСКЛЮЧАЮЩЕЕ ИЛИ" на регистре-аккумуляторе (А) и байте-источнике. Вторым операндом (байтом-источником) при этом может быть: регистр в выбранном банке рабочих регистров; регистр внутреннего ОЗУ, адресуемый с помощью косвенной адресации; прямоадресуемые ячейки внутреннего ОЗУ и регистры специального назначения; непосредственная величина.

Указанные логические операции могут быть реализованы на любом прямоадресуемом регистре внутреннего ОЗУ или регистре специального назначения с использованием в качестве второго операнда содержимого аккумулятора А или непосредственных данных.

Существуют логические операции, которые выполняются только на аккумуляторе: сброс и инвертирование всех восьми разрядов А; циклический сдвиг влево и вправо; циклический сдвиг влево и вправо с учетом флага переноса; обмен местами старшей и младшей тетрад внутри аккумулятора.

3.4. Операции пересылки данных

Большую часть команд данной группы составляют команды передачи и обмена байтами, а также отдельными битами. Все команды данной группы не модифицируют флаги результата, за исключением команд загрузки PSW, триггера (флага) С и аккумулятора (устанавливается флаг паритета). На рис. 4 показан граф путей передачи данных в МК. Отдельной вершиной на этом графе

представлен аккумулятор (A), т.к. он участвует в большинстве пересылок и адресуется с помощью неявной или прямой адресации. Тем не менее, некоторые пересылки выполняются без участия аккумулятора.

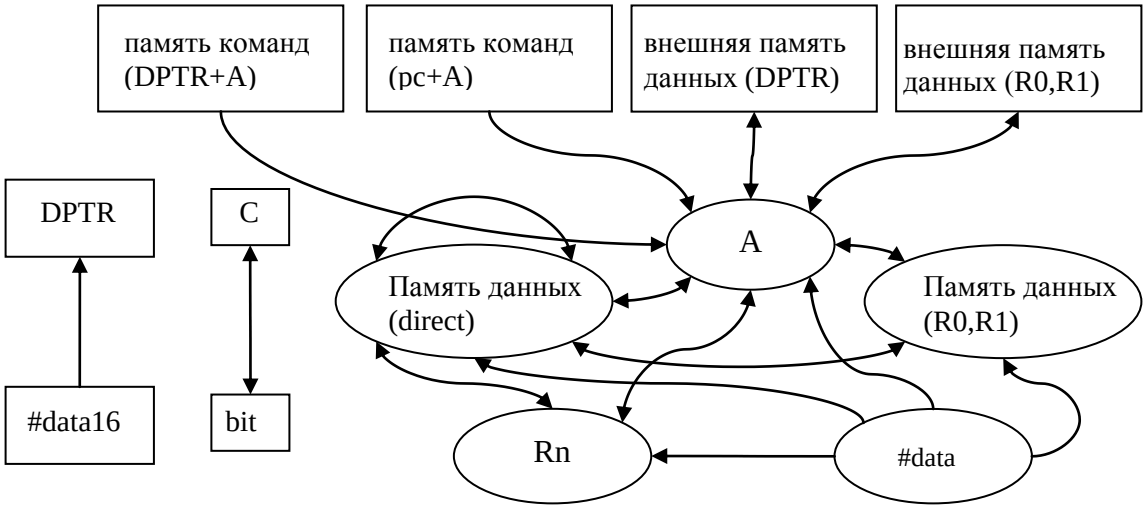

Рис. 4. Граф путей передачи данных в МК

В число команд пересылок входят операции со стеком, который организуется в памяти данных. Для адресации ячеек стека используется регистр-указатель стека (SP), что позволяет адресовать любую ячейку внутреннего ОЗУ. Запись информации в стек производится командой PUSH direct, а чтение из стека – POP direct. В исходном состоянии SP адресует «верхушку» стека – последнюю ячейку стековой памяти, в которую записана информация. Перед записью в стек содержимое SP инкрементируется, а после чтения – декрементируется.

3.5. Битовые операции

В состав VR входит так называемый «битовый» процессор, который обеспечивает выполнение ряда операций над битами.

Битовый процессор является частью архитектуры семейства x51 и его можно рассматривать как независимый процессор побитовой обработки.

Битовый процессор выполняет свой набор команд, имеет свое побитово-адресуемое ОЗУ и свой ввод-вывод.

Команды, оперирующие с битами, обеспечивают прямую адресацию 128 битов в шестнадцати ячейках внутреннего ОЗУ (ячейки с адресами 20H–2FH) и прямую побитовую адресацию регистров специального назначения, адреса которых кратны восьми: P0 (80H), TCON (88H), P1 (90H), SCON (98H), P2 (A0H), IE (A8H), P3 (B0H), IP (B8H), PSW (D0H), A (E0H), B (F0H).

Каждый из отдельно адресуемых битов может быть установлен в "1", сброшен в "0", инвертирован, проверен. Могут быть реализованы переходы: если бит установлен; если бит не установлен; переход, если бит установлен, со сбросом этого бита; бит может быть перезаписан в (из) разряд(а) переноса. Между любым прямоадресуемым битом и флагом переноса могут быть произведены логические операции "И", "ИЛИ", где результат заносится в разряд флага переноса. Команды побитовой обработки обеспечивают реализацию сложных функций комбинаторной логики и оптимизацию программ пользователя.

3.6. Операции ветвления

К данной группе команд относятся команды, обеспечивающие условное и безусловное ветвление, вызов подпрограмм и возврат из них, а также команда пустой операции NOP. В большинстве команд используется прямая адресация, т.е. адрес перехода целиком (или его часть) содержится в самой команде передачи управления. Можно выделить три разновидности команд ветвления по разрядности указываемого адреса перехода.

Длинный переход. Переход по всему адресному пространству ПП. В команде содержится полный 16-битный адрес перехода (addr16). Трехбайтные команды длинного перехода содержат в мнемокоде букву L (Long). Всего существуют две такие команды: LJMP – длинный переход и LCALL – длинный вызов подпрограммы.

Абсолютный переход. Переход в пределах одной страницы памяти программ размером 2048 байт. Такие команды содержат только 11 младших бит адреса перехода (addr11). Команды абсолютного перехода имеют формат 2 байта. Начальная буква мнемокода – A (Absolute). При выполнении команды в вычисленном адресе следующей по порядку команды ((PC) = (PC)+2) 11 младших бит заменяются на addr11 из тела команды абсолютного перехода.

Относительный переход. Короткий относительный переход позволяет передать управление в пределах -128...+127 байт относительно адреса следующей команды (команды, следующей по порядку за командой относительного перехода). Существует одна команда безусловного короткого относительного перехода SJMP (Short). Все команды условного перехода используют данный метод адресации. Относительный адрес перехода (rel) содержится во втором байте команды.

Косвенный переход. Команда JMP @A+DPTR позволяет передавать управление по косвенному адресу. Эта команда удобна тем, что предоставляет возможность организации перехода по адресу, вычисляемому самой программой и неизвестному при написании исходного текста программы.

Условные переходы. Развитая система условных переходов предоставляет возможность осуществлять ветвление по следующим условиям: аккумулятор содержит нуль (JZ); содержимое аккумулятора не равно нулю (JNZ); перенос равен единице (JC); перенос равен нулю (JNC); адресуемый бит равен единице (JB); адресуемый бит равен нулю (JNB).

Для организации программных циклов удобно пользоваться командой DJNZ. В качестве счетчика циклов может использоваться регистр, прямоадресуемый байт.

Команда CJNE может быть использована в процедурах ожидания какого-либо события. Например, команда

WAIT: CJNE A, P0, WAIT

будет выполняться до тех пор, пока на линиях порта 0 не установится информация, совпадающая с содержимым аккумулятора.

Все команды данной группы, за исключением CJNE, не оказывают воздействия на флаги. Команда CJNE устанавливает флаг C, если первый операнд оказывается меньше второго.

Подпрограммы. Для обращения к подпрограммам необходимо использовать команды вызова подпрограмм (LCALL, ACALL). Эти команды, в отличие от команд перехода (LJMP, AJMP), сохраняют в стеке адрес возврата в основную программу. Для возврата из подпрограммы необходимо выполнить команду RET. Команда RETI отличается от команды RET тем, что разрешает прерывания обслуженного уровня Поэтому эту команду необходимо применять в конце подпрограмм, вызванных по прерыванию.

4. Микроконтроллер C8051F411

Блок схема микроконтроллера C8051F411 [1] в составе микропроцессорного ядра CIP51, набора памяти данных (256 B SRAM), памяти команд (32kB Flash), системы управления питанием (VREG), отладки (Debug HW), тактирования (Oscillators), часов реального времени (RTClock) и периферийных устройств подключенных к шине регистров специального назначения (SFR Bus) приведена на рис. 5.

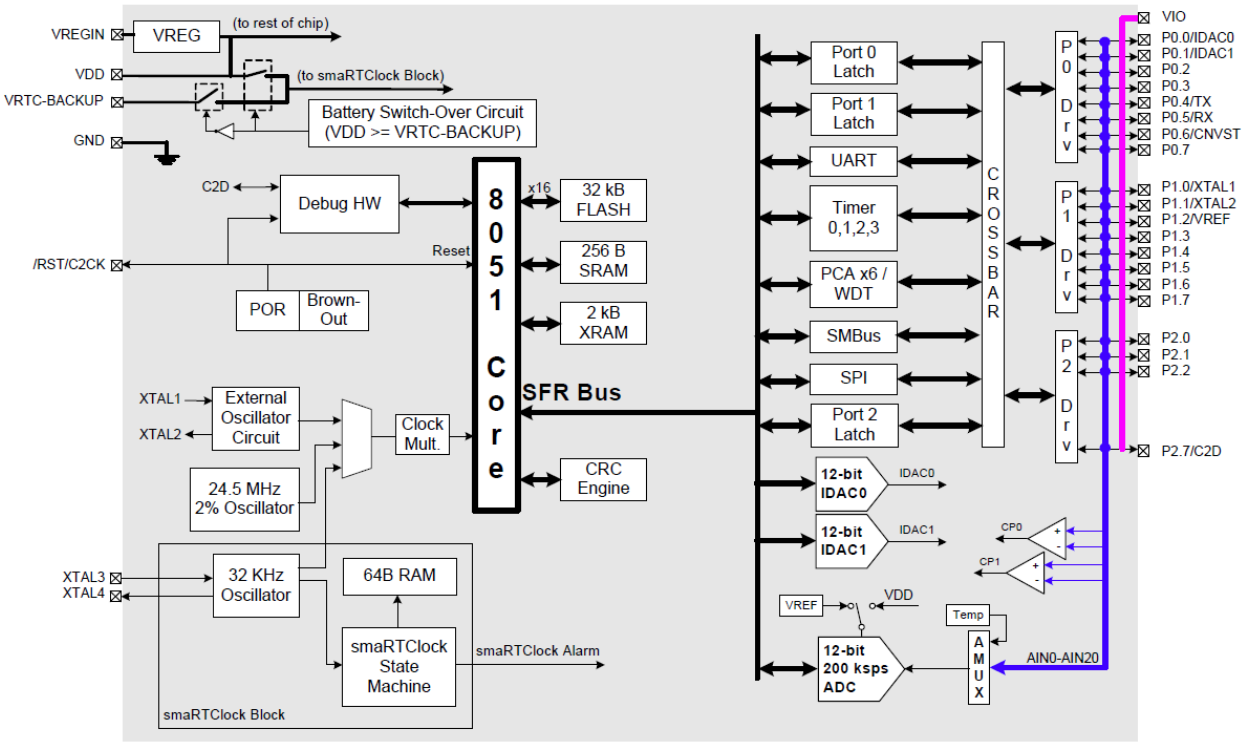

Рис. 5. Блок схема микроконтроллера C8051F411

Для того, чтобы начать работу с МК, необходимо ознакомиться с принципами работы некоторых важных узлов, таких как система прерываний, сторожевой таймер, матрица соединений портов ввода вывода.

4.1. Обработка прерываний

C8051F411 имеет развитую систему прерываний, поддерживающую в общей сложности 18 источников прерываний с двумя уровнями приоритета, которые приведены в *таблице 4*. Каждый источник прерываний имеет один или несколько связанных с ним флагов прерываний, размещенных в регистрах

специальных функций SFR. Когда периферийный модуль или внешний источник прерываний регистрирует событие, удовлетворяющее условию прерывания, соответствующий флаг прерывания устанавливается в 1.

Если прерывание от источника прерываний разрешено, то при установке флага прерывания генерируется запрос прерывания. Как только выполнение текущей команды завершится, будет сгенерирована команда LCALL перехода по предопределенному адресу (*вектору прерывания*), откуда начнется исполнение процедуры обслуживания прерывания (interrupt service routine - ISR). Каждая ISR должна заканчиваться командой RETI, которая возвращает управление прерванной программе и приводит к выполнению той команды, которая исполнилась бы, если бы запроса прерывания не было. Если прерывания не разрешены, флаг прерывания игнорируется и выполнение программы продолжается в нормальном режиме. (Флаг прерывания устанавливается в 1 независимо от того, разрешены прерывания или запрещены).

Таблица 4

Источники прерываний

Источник прерывания	**Вектор преры в-ания**	**Приоритет**	**Флаг прерывания**	**Битовая адресация ?**	**Аппаратный сброс ?**	**Бит разреше ния**	**Управле ние приорите том**
Сброс	0x0000	На и-вы с-ш ий	Нет	N/ A	N/ A	Разреше н всегда	Всегда наивысши й
Внешнее прерывание 0 (/INT0)	0x0003	0	IE0 (TCON.1)	Y	Y	EX0 (IE.0)	PX0 (IP.0)
Переполнение таймера 0	0x000B	1	TF0 (TCON.5)	Y	Y	ET0 (IE.1)	PT0 (IP.1)
Внешнее	0x0013	2	IE1	Y	Y	EX1	PX1 (IP.2)

прерывание 1 (/INT1)			(TCON.3)			(IE.2)	
Переполнение таймера 1	0x001B	3	TF1 (TCON.7)	Y	Y	ET1 (IE.3)	PT1 (IP.3)
Последовательный порт UART0	0x0023	4	RI0 (SCON0.0) TI0 (SCON0.1)	Y	N	ES0 (IE.4)	PS0 (IP.4)
Переполнение таймера 2	0x002B	5	TF2H (TMR2CN.7) TF2L (TMR2CN.6)	Y	N	ET2 (IE.5)	PT2 (IP.5)
Модуль SPI	0x0033	6	SPIF (SPI0CN.7) WCOL (SPI0CN.6) MODF (SPI0CN.5) RXOVRN (SPI0CN.4)	Y	N	ESPI0 (IE.6)	PSPI0 (IP.6)
Модуль SMBus	0x003B	7	SI (SMB0CN.0)	Y	N	ESMB0(EIE1.0)	PSMB0 (EIP1.0)
Часы реального времени smaRTClock	0x0043	8	ALRM (RTC0CN.2) OSCFAIL(RTC0CN.5)	N	N	ERTC0 (EIE1.1)	PRTC0 (EIP1.1)
Детектор диапазона АЦП0	0x004B	9	AD0WINT (ADC0CN.3)	Y	N	EWADC0 (EIE1.2)	PWADC0 (EIP1.2)
Завершение преобразования АЦП0	0x0053	10	AD0INT (ADC0STA.5)	Y	N	EADC0 (EIE1.3)	PADC0 (EIP1.3)
Программируемый массив счетчиков	0x005B	11	CF (PCA0CN.7) CCFn (PCA0CN.n)	Y	N	EPCA0 (EIE1.4)	PPCA0 (EIP1.4)
Компаратор 0	0x0063	12	CP0FIF (CPT0CN.4) CP0RIF (CPT0CN.5)	N	N	ECP0 (EIE1.5)	PCP0 (EIP1.5)
Компаратор 1	0x006B	13	CP1FIF (CPT1CN.4) CP1RIF (CPT1CN.5)	N	N	ECP1 (EIE1.6)	PCP1 (EIP1.6)

Переполнение таймера 3	0x0073	14	TF3H (TMR3CN.7) TF3L (TMR3CN.6)	N	N	ET3 (EIE1.7)	PT3 (EIP1.7)
Система управления питанием	0x007B	15	N/A	N/A	N/A	EREG0 (EIE2.0)	PREG0 (EIP2.0)
Совпадение порта	0x0083	16	N/A	N/A	N/A	EMAT (EIE2.1)	PMAT (EIP2.1)

Прерывание от каждого источника прерываний может быть разрешено или запрещено с помощью соответствующих битов разрешения прерываний в регистрах SFR (IE-EIE1). Однако сначала прерывания необходимо разрешить глобально установкой в 1 бита EA (IE.7), только после этого состояние индивидуальных флагов разрешения прерываний будет иметь силу. Сброс в 0 бита EA запрещает прерывания от всех источников прерываний независимо от состояния индивидуальных флагов разрешения прерываний. *По умолчанию после сброса EA равен нулю.*

Программа может симулировать прерывание установкой в 1 любого флага прерывания. Если прерывание для этого флага разрешено, будет сгенерирован запрос прерывания и произойдет переход по адресу процедуры ISR, связанной с этим флагом прерывания.

Некоторые флаги прерываний сбрасываются автоматически аппаратными средствами при переходе к процедуре ISR. Однако большинство флагов прерываний не сбрасываются аппаратно и должны быть сброшены программно до возвращения из процедуры ISR. Если флаг прерывания остается установленным после завершения выполнения команды возврата из прерывания (RETI), то сразу же будет сгенерирован новый запрос прерывания и после завершения выполнения следующей команды произойдет повторный переход к процедуре ISR.

Алгоритм работы процессора после подачи команды сброс или при включении питания следующий:

- происходит генерирование запрос прерывания по сбросу;

- в программный счетчик загружается адрес вектора прерывания по сбросу (0x0000);
- происходит загрузка команды, находящейся по адресу 0x0000, в дешифратор команд;
- выполнение загруженной в дешифратор команды.

Исходя из этого, следует вывод: размещать код программы в области векторов прерываний крайне нежелательно. Для того, чтобы этого избежать, в области векторов прерываний размещают длинные переходы к основному тексту программы и обработчики прерываний конкретных событий. Пример части такой программы приведен на рис. 6.

```
;-------------------------------------------------------------------------------
; Векторы сброса и прерываний
;-------------------------------------------------------------------------------

; Вектор сброса
cseg AT 0                   ; директива компилятора:
                            ; установка адреса записи в памяти
                            ; команд на 0x0000
ljmp Main                   ; длинный переход на метку Main
                            ; по событию сброс.
                            ; дальше могут располагаться вектора прерываний
                            ; в соответствии с таблицей прерываний
;-------------------------------------------------------------------------------
; Сегмент кода
;-------------------------------------------------------------------------------

Blink      segment  CODE
                            ; директива компилятора:
                            ; создается новый кодовый сегмент с именем Blink
rseg     Blink
                            ; относительный нулевой адрес устанавливается на
                            ; начало сегмента Blink
using    0                  ; выбирается 0 банк регистров

Main:                       ; метка Main
; текст программы.
```

Рис. 6. Пример использования перехода по сбросу

4.2. Сторожевой таймер

МК содержит программируемый сторожевой таймер (Watchdog Timer - WDT), работающий независимо от системного тактового сигнала. WDT переводит МК в состояние сброса в случае своего переполнения. Чтобы

предотвратить сброс, WDT должен перезапускаться из прикладной программы до того, как произойдет его переполнение. Если в системе происходит программный/аппаратный сбой, не позволяющий программе перезапустить WDT, то WDT переполнится и вызовет сброс. Это предотвращает выход системы из-под контроля.

После сброса любого типа WDT автоматически включается и запускается по умолчанию с максимальным таймаутом. При необходимости WDT можно программно отключить или заблокировать, предотвратив его случайное отключение. После блокировки WDT его нельзя отключить до следующего системного сброса.

WDT состоит из 16-разрядного таймера, работающего с программируемой тактовой частотой. Этот таймер измеряет период между операциями записи определенных значений в его регистр управления. Если этот период превышает установленный предел, то генерируется сброс от WDT. WDT может быть программно разрешен или запрещен, кроме этого можно заблокировать функцию отключения WDT. Управление WDT осуществляется посредством регистра управления PCA0MD , показанного в *таблице 5* и *таблице 6*.

Для того, чтобы заблокировать, необходимо записать «0» в ячейку WDTE. Сделать это можно, используя следующий код:

anl PCA0MD, #NOT(040h) ; Очистить бит разрешения работы сторожевого таймера

Таблица 5

Внутреннее устройство регистра PCA0MD

Доступность	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	Значение по сбросу
Название	CIDL	WDTE	WDLCK	–	CPS2	CPS1	CPS0	ECF	01000000

Номер бита	Bit 7	Bit6	Bit5	Bit4	Bit 3	Bit 2	Bit 1	Bit 0	
							Адрес SFR:		0xD9

Таблица 6

Описание битов регистра PCA0MD

<table>
<tr><td>Bit7:</td><td>CIDL: Управление PCA Счетчиком/таймеров в режиме Idle.
Определяет поведение PCA в режиме Idle
Mode.0: PCA продолжает нормальную работу в режиме Idle;
Mode.1: PCA прекращает работу в режиме Idle.</td></tr>
<tr><td>Bit6:</td><td>WDTE: Разрешение работы сторожевого таймера
Если бит установлен, модуль 5 PCA используется как сторожевой таймер.
0: сторожевой таймер выключен.
1: сторожевой таймер включен.</td></tr>
<tr><td>Bit5:</td><td>WDLCK: запрет изменения режима работы сторожевого таймера, если он установлен, изменить его можно только после следующего сброса
0: разрешено изменять режим работы сторожевого таймера.
1: запрещено изменять режим работы сторожевого таймера.</td></tr>
<tr><td>Bit4:</td><td>Не используется. Чтение = 0b, Запись = неважно.</td></tr>
<tr><td>Bits3–1:</td><td>CPS2–CPS0: выбор источника тактирования счетчика/таймера PCA.

<table>
<tr><th>CPS2</th><th>CPS1</th><th>CPS0</th><th>Источник тактирования</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>Частота внутреннего генератора, деленная на 12</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>Частота внутреннего генератора, деленная на 4</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>Переполнение таймера 0</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>Срез импульса на ножке ECI</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>Частота внутреннего генератора</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>Частота внешнего генератора, деленная на 8</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>Частота часов smaRTClock, деленная на 8</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>зарезервированно</td></tr>
</table>
</td></tr>
<tr><td>Bit0:</td><td>ECF: Разрешение прерывания счетчика/таймера PCA по переполнению.
0: запретить прерывание.
1: разрешить прерывание.</td></tr>
</table>

4.3. Порты ввода-вывода

Микроконтроллер C8051F411 представляют собой полностью интегрированные на одном кристалле системы для обработки смешанных сигналов, 20 цифровых входа/выхода, организованных в 8-разрядные порты. Все порты доступны в режиме как побитной, так и побайтной адресации через соответствующие регистры данных портов. Каждый из выводов портов имеет слаботоковые подтягивающие резисторы и может быть настроен как выход с открытым стоком или цифровой двухтактный выход. Кроме этого, допустимое напряжение на выводах Порта 0 составляет 5В. Структурная схема ячейки порта ввода/вывода показана на рис. 7.

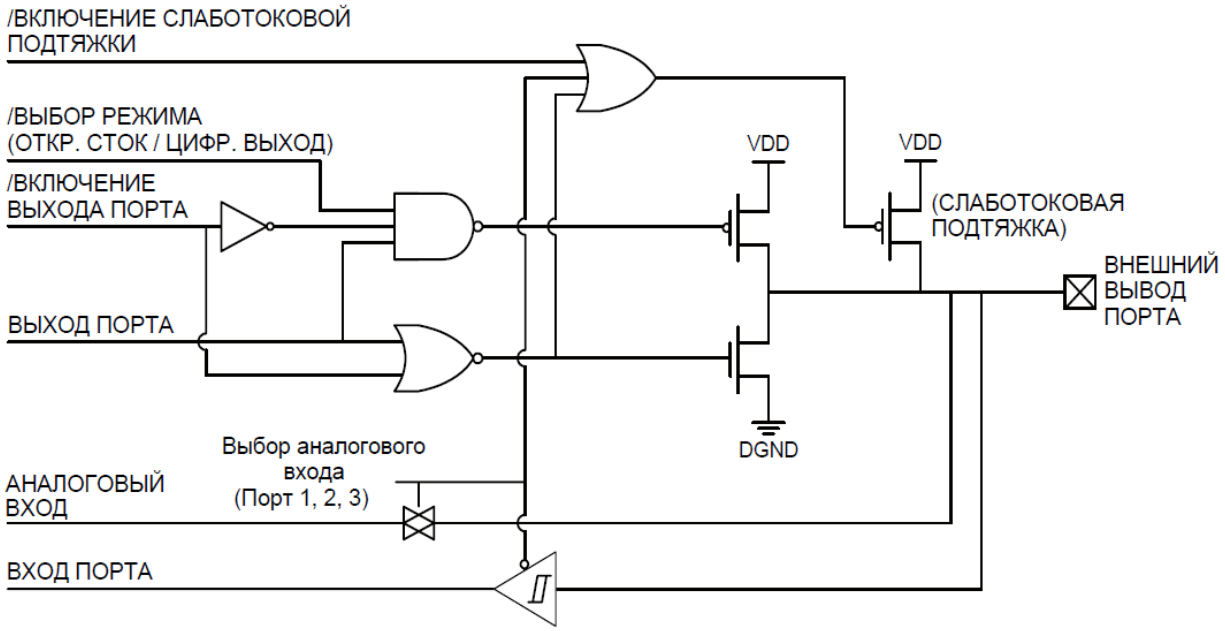

Рис. 7. Структурная схема ячейки порта ввода/вывода

Микроконтроллеры имеют различные цифровые ресурсы, которые доступны через порта ввода/вывода: P0, P1, P2. Каждый из выводов портов P0, P1, P2 может быть определен либо как вывод ввода/вывода общего назначения, либо как вывод, управляемый внутренними цифровыми ресурсами (например, УАПП0 или /INT1), как показано на рис.8. Разработчик системы определяет, какие цифровые ресурсы будут назначены внешним выводам, ограничиваясь только количеством доступных выводов. Гибкость при распределении ресурсов достигается благодаря использованию приоритетного декодера матрицы

(Priority decoder Digital Crossbar). Следует иметь ввиду, что состояние вывода порта ввода/вывода всегда можно прочитать из соответствующего регистра данных независимо от того, как функционирует этот вывод: как вывод ввода/вывода общего назначения или как вывод, назначенный какому-либо внутреннему цифровому ресурсу.

Приоритетный декодер матрицы, или “матрица”, распределяет и назначает выводы портов P3 – P0 цифровым периферийным модулям (УАПП, SMBus, ПМС, таймеры и т.д.) микроконтроллера, используя для этого приоритеты. Выводы портов распределяются, начиная с P0.0 и (если необходимо) до P2.3. Цифровые периферийные модули назначаются выводам портов в соответствии с их приоритетом (см. рис. 9). УАПП0 имеет наивысший приоритет, T1 имеет самый низкий приоритет. Регистры P0SKIP, P1SKIP, P2SKIP используются для того, чтобы исключить конкретный вывод порта из участия в распределении ресурсов, обычно это делается для того, чтобы использовать вывод как аналоговый.

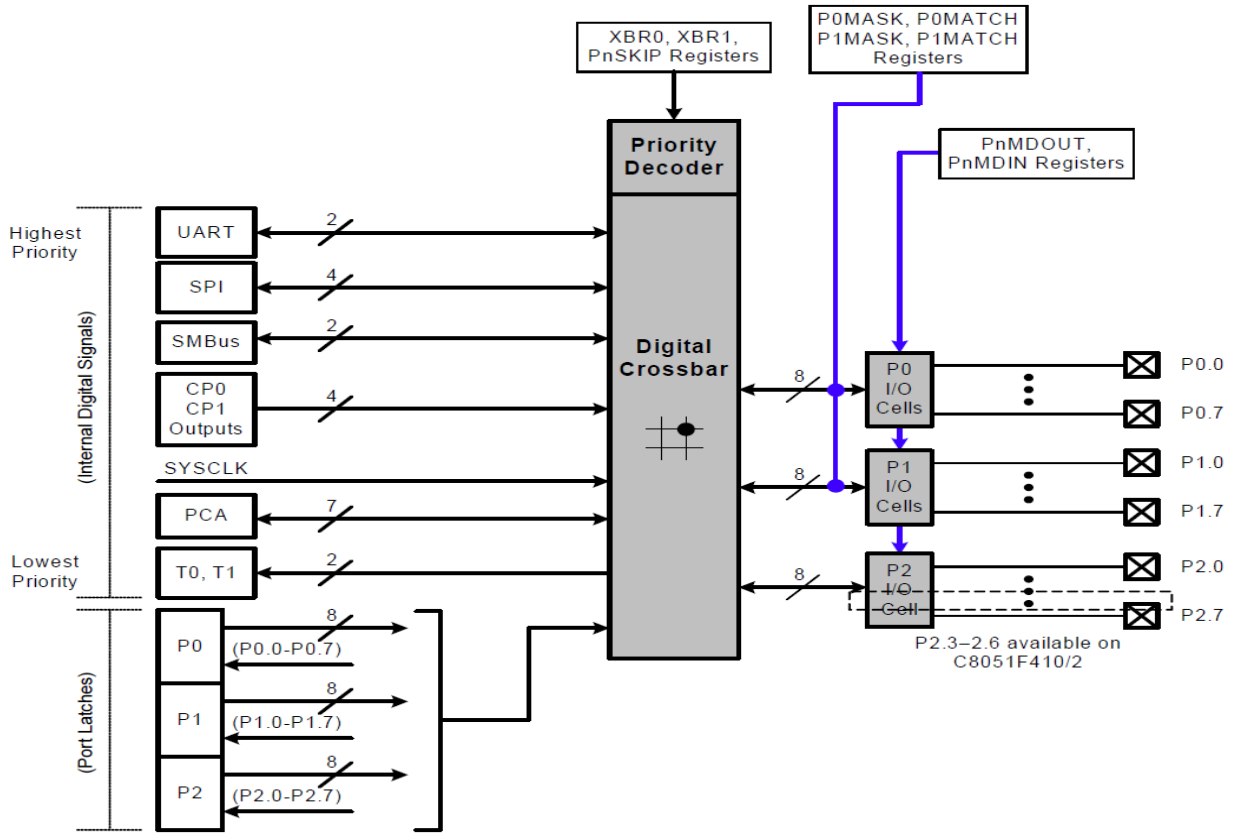

Рис. 8. Функциональная схема портов ввода/вывода

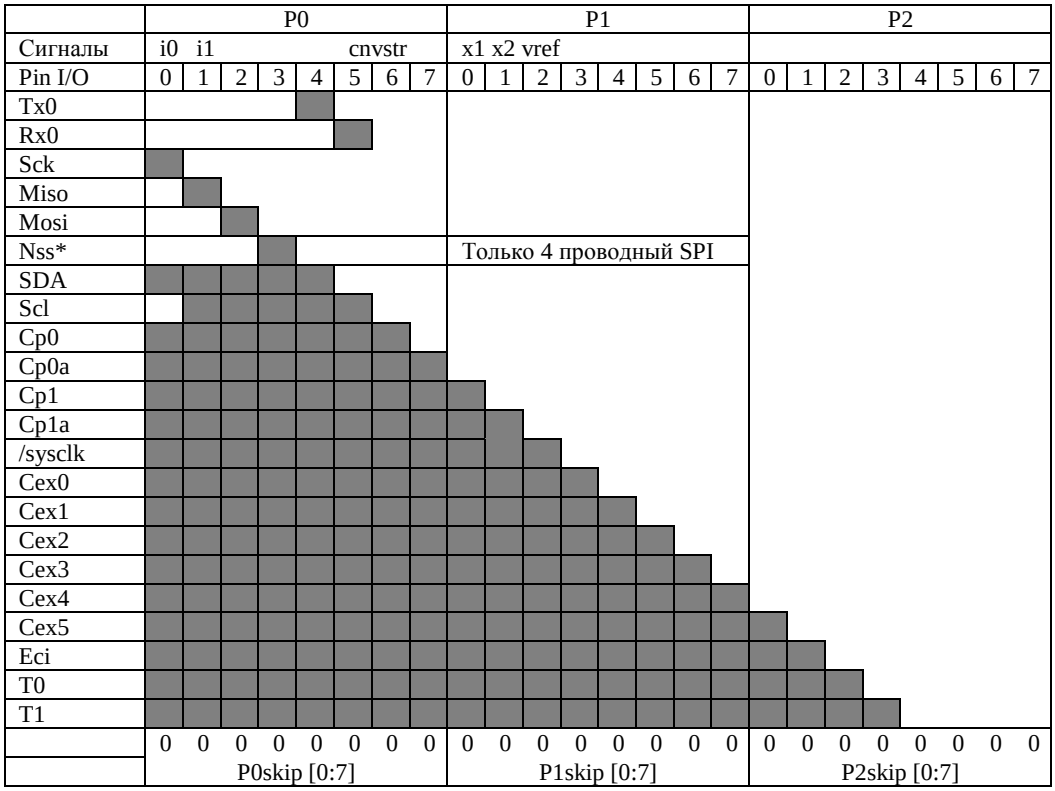

Рис. 9. Таблица декодирования приоритетов матрицы

Выходные драйверы портов P0 – P3 остаются отключенными до тех пор, пока матрица не будет включена установкой в 1 бита XBARE (XBR1.6). Для того, чтобы включить приоритетный декодер матрицы, можно воспользоваться следующим кодом:

```
mov   XBR1, #40h          ; включить матрицу.
```

Выходной драйвер каждого порта можно настроить либо как цифровой двухтактный выход, либо как выход с открытым стоком. При работе в режиме цифрового двухтактного выхода запись лог. ‘0’ в соответствующий бит регистра данных порта приведет к «притягиванию» данного вывода порта к земляной шине GND, а запись лог. ‘1’ приведет к «притягиванию» данного вывода порта к шине питания VDD. При работе в режиме выхода с открытым стоком запись лог. ‘0’ в соответствующий бит регистра данных порта приведет

к «притягиванию» данного вывода порта к земляной шине GND, а при записи лог. ‘1’ данный вывод порта будет переведен в высокоимпедансное состояние.

Режимы выходов портов P0 – P2 определяются битами соответствующих регистров PnMDOUT (рис. 11). Например, при установке в 1 бита P0MDOUT.7 выходной драйвер порта P0.7 будет настроен как цифровой двухтактный выход, при сбросе в 0 бита P0MDOUT.7 выходной драйвер порта P0.7 (рис.10) будет настроен как выход с открытым стоком. По умолчанию выходные драйверы всех портов настраиваются как выходы с открытым стоком.

Вывод порта настраивается как цифровой вход переводом его выходного драйвера в режим выхода с открытым стоком и записью лог. ‘1’ в соответствующий бит регистра данных порта. Например, P3.7 настраивается как цифровой вход сбросом в 0 бита P0MDOUT.7 и установкой в 1 бита P0.7.

Если вывод порта назначен посредством матрицы цифровому периферийному модулю и этот вывод функционирует как вход (например, RX0, вывод приемника УАПП0), то выходной драйвер этого вывода автоматически отключается.

P0

Доступность	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Значение по сбросу
Название	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0	11111111
Номер бита	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	

Адрес SFR: 0x80

Биты 7-0: P0.[7:0]: Биты выходной защелки порта 0.
(Запись – выходной сигнал появляется на внешних выводах в зависимости от состояния XBR0, XBR1)
0: Выход в состоянии лог. 0.
1: Выход в состоянии лог. 1 (в высокоимпедансном состоянии, если соответствующий бит P0MDOUT.n = 0).
(Чтение – независимо от состояния регистров XBR0, XBR1)
0: На выводе P0.n низкий логический уровень.
1: На выводе P0.n высокий логический уровень.

Рис. 10. Внутреннее устройство регистра P0

P0MDOUT

Доступность	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Значение по сбросу
Название	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	00000000
Номер бита	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	

Адрес SFR: 0xA4

Биты 7-0: P0MDOUT.[7:0]: Биты настройки выходного драйвера порта 0.
0: Соответствующий вывод P0.n настроен как выход с открытым стоком.
1: Соответствующий вывод P0.n настроен как цифровой двухтактный выход.

Рис 11. Внутреннее устройство регистра P0

5. Директивы компилятора

В этом разделе приведены директивы компилятора Keil, в качестве справочного материала (*таблица 7*) [3,4]. Некоторые из них необходимы для написания любой программы на языке ассемблер к таким можно отнести определение сегментов (CSEG), названия программы (NAME), задания символических имен для регистров и битов (sfr, sfr16sbit).

Таблица 7

Список директив компилятора Keil.

Директива	Формат	Описание
BIT	symbol BIT bit_address	Определить имя бита из битово адресуемой области памяти данных
BSEG	BSEG [AT absolute address]	Определить абсолютный сегмент в битово адресуемой области памяти данных
CODE	symbol CODE code_address	Присвоить символическое имя специфическому месту в памяти команд.
CSEG	CSEG [AT absolute address]	Определить абсолютный сегмент в памяти команд.
DATA	symbol DATA data_address	Присвоить символическое имя специфическому месту в памяти данных.
DB	[label:] DB expression [, expr ...]	Создать список байтовых величин
DBIT	[label:] DBIT expression	Зарезервировать место в битах
DS	[label:] DS expression	Зарезервировать место в байтах
DSEG	DSEG [AT absolute address]	Определить абсолютный сегмент в косвенно адресуемой памяти данных
_ _ERROR_ _	_ _ERROR_ _ text	Создать стандартное сообщение ошибки
EXTRN	EXTRN class (symbol [, ...])	Определяет символы, ссылки на которые есть в текущем модуле, которые определены в других модулях
IDATA	symbol IDATA idata_address	Присвоить символическое имя в косвенно адресуемой памяти данных
ISEG	ISEG [AT absolute address]	Определить абсолютный сегмент в памяти данных

NAME	NAME modulname.	Определить имя текущего модуля
ORG	ORG expression	Установить местоположение в текущем сегменте.
PUBLIC	PUBLIC symbol [, symbol ...]	Определить символы, которые могут быть использованы вне данного сегмента
RSEG	RSEG seg	Выбрать сегмент
SEGMENT	seg SEGMENT class [reloctype] [alloctype]	Определить сегмент
SET	SET expression	Установить значение символа

sfr,
sfr16
sbit
	sfr symbol = address;
	sfr16 symbol = address;
	sbit symbol = address;
		Определить символическое имя для регистра специальных функций или конкретного бита

USING	USING expression	Установить банк используемых регистров общего назначения
XDATA	symbol XDATA xdata_address	Присвоить символическое имя сегменту внутри внешней памяти данных
XSEG	XSEG [AT absolute address]	Определить абсолютный сегмент внутри внешней памяти данных

6. Лабораторная установка C8051F411EB

Для выполнения работ используется плата начального освоения C8051F411EB [2], внешний вид платы приведен на рис. 12.

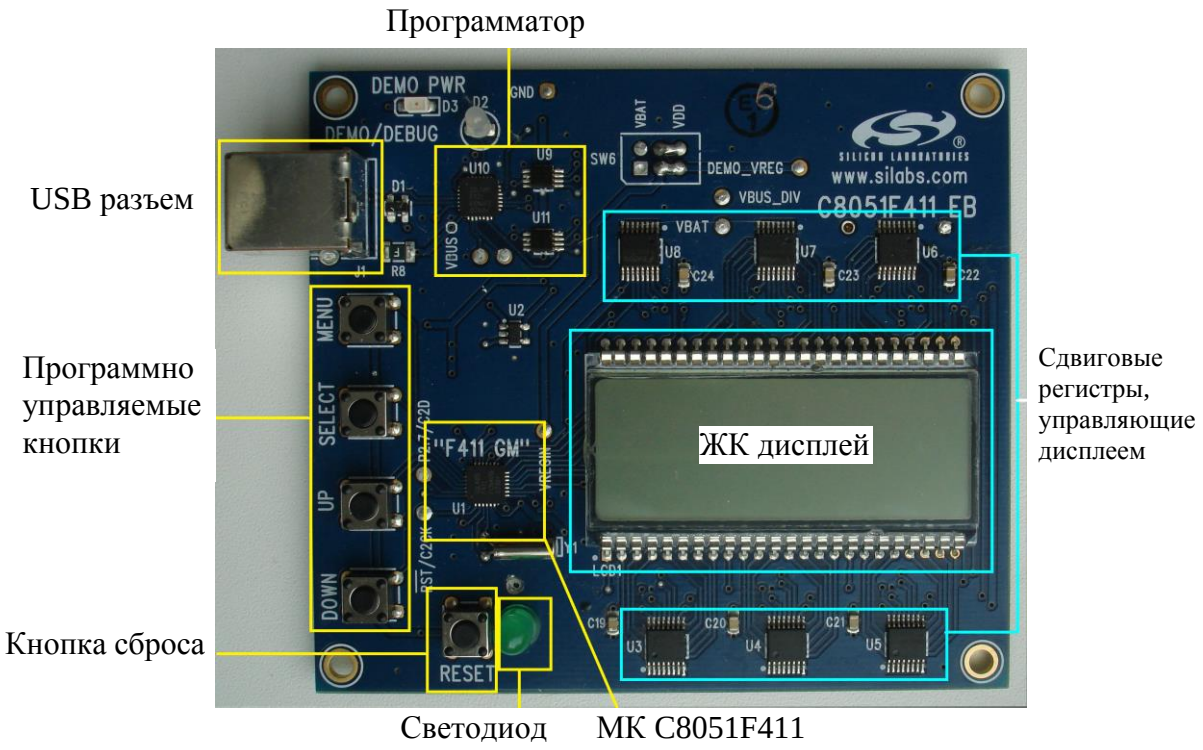

Рис. 12. Внешний вид лабораторной установки

В состав платы начального освоения C8051F411EB входят:

- микроконтроллер C8051F411;
- внутрисхемный программатор;
- система питания;
- 4 кнопки с программно задаваемыми функциями;
- кнопка сброса;
- сдвиговые регистры для управления ЖК индикатором;
- ЖК индикатор с готовыми сегментами.

В *таблице 8* перечислены номера выводов МК, которые подключены к периферийным устройствам, и их функциональное назначение.

Таблица 8

Номер вывода	Функция
P0.0	Тактирование сдвига
P0.1	nc
P0.2	Выход данных для сдвигового регистра
P0.3	Тактирование регистра хранения
P0.4	Передатчик UART
P0.5	Приемник UART
P0.6	nc
P0.7	Кнопка «menu»
P1.0	Светодиод
P1.1	nc
P1.2	Опорное напряжение для АЦП (Vref)
P1.3	Включение подтягивающих резисторов на линии управления ЖК индикатором
P1.4	Линия управления ЖК индикатором «com1»
P1.5	Линия управления ЖК индикатором «com2»
P1.6	Линия управления ЖК индикатором «com3»
P1.7	Линия управления ЖК индикатором «com4»
P2.0	Кнопка «select»
P2.1	Кнопка «up»
P2.2	Кнопка «down»

6.1. Введение в IDE SILABS

IDE Silabs фирмы Silicon Laboratories – интегрированная среда разработки программного обеспечения для однокристальных микроконтроллеров семейства x51. Она включает в себя всё, что нужно для создания, редактирования, компиляции, трансляции, компоновки, загрузки и отладки программ:

- стандартный интерфейс Windows,
- полнофункциональный редактор исходных текстов с выделением синтаксических элементов цветом,
- организатор проекта,

- транслятор с языка С,
- ассемблер,
- отладчик,
- встроенную справочную систему.

6.1.1. Макроассемблер A51 фирмы Keil

Ассемблер A51 совместим с ASM51 Intel для всего семейства микроконтроллеров x51. 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 x51.

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.

6.1.2. Оптимизирующий кросс-компилятор C51 фирмы Keil

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 x51. C51 генерирует код для всего семейства микроконтроллеров x51. 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:

- глубокого знания системы команд процессора не требуется, элементарное знание архитектуры x51 желательно, но не необходимо;
- распределение регистров и способы адресации управляются полностью транслятором;
- лучшая читаемость программы, используются ключевые слова и функции;
- время разработки программ и их отладки значительно короче в сравнении с программированием на ассемблере;
- библиотечные файлы содержат много стандартных подпрограмм, которые могут быть включены в прикладную программу;
- существующие программы могут многократно использоваться в новых программах.

6.1.3. 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.

6.1.4. Отладчик

Отладчик исходных текстов используется с транслятором C51, ассемблером A51, транслятором PL/M-51 и ассемблером ASM51. Отладчик

можно использовать для проверки и отладки прикладной программы внутри контроллера.

6.2. Запуск IDE Silabs и создание файла проекта

IDE Silabs запускается из стартового меню Windows подобно остальным приложениям (рис. 13).

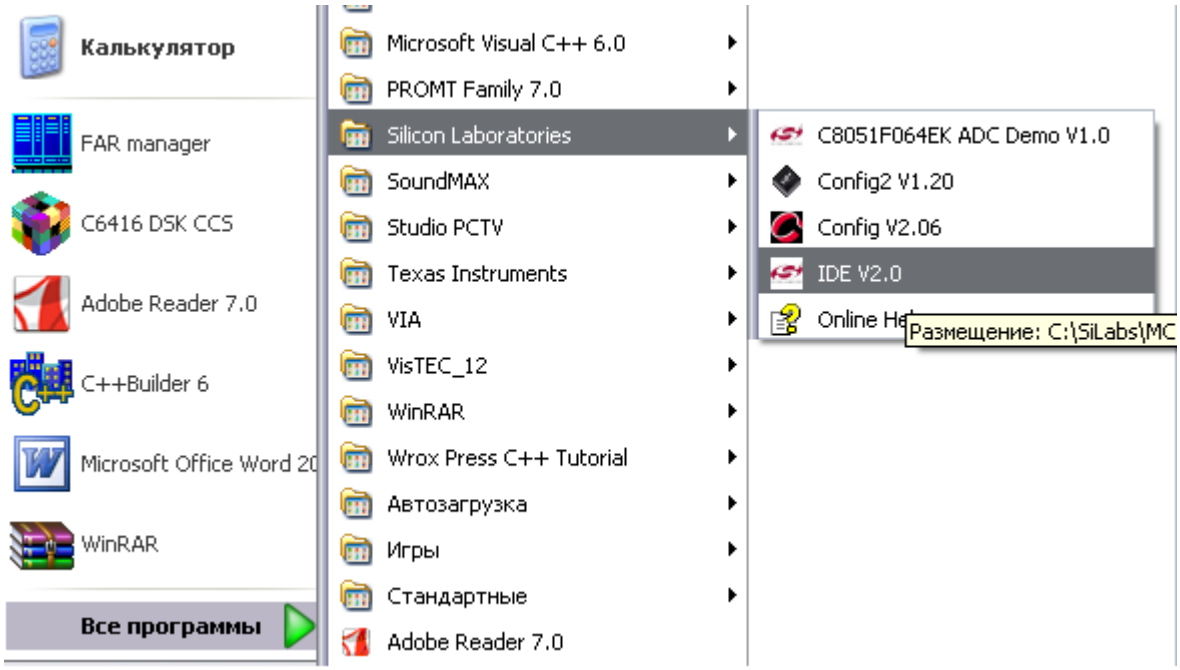

Рис. 13. Запуск программы

Любая новая работа в IDE Silabs, 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 Project из меню Project.

Когда менеджер проекта открывает файл проекта, окно проекта показывает включенные исходные файлы. В данном случае пока нет никаких исходных файлов.

6.3. 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 возьмём простейшую программу. “Blink” - программа из папки \Silabs\MCU\Examples\C8051F41x\Blinky\F41x_Blinky_m.asm, которая обеспечивает мигание светодиода на отладочной плате. Исходный текст программы содержится в файле blink.asm и на рис.14.

```
/*************************************/
/* Ваша первая C8051F411 программа */
/*************************************/
;-----------------------------------------------------------
;  Copyright (C) 2004 Silicon Laboratories, Inc.
;  Все права защищены.
;  Имя файла   :  BLINK.ASM
;  Применяемые МК  :  C8051F41x
;  Назаначение :  Эта программа демонстрирует как отключить watchdog таймер,
; сконфигурировать порт и вывести информацию на вывод порта ввода/вывода.
;-----------------------------------------------------------
$include (c8051f410.inc)
                              ; Подключает файл определения регистров.
;-----------------------------------------------------------
; EQUATES                  ;раздел определений
;-----------------------------------------------------------
GREEN_LED     equ   P1.0
                              ; Определение символического имени
                              ;контакта порта ввода/вывода, к которому подключен светодиод.
;-----------------------------------------------------------
; RESET and INTERRUPT VECTORS ; Векторы сброса и прерываний
;-----------------------------------------------------------
; Вектор сброса
cseg AT 0                     ; директива компилятора:
                              ; установка адреса записи в памяти
                              ; команд на 0x0000
ljmp Main                     ; длинный переход на метку Main
                              ; по событию сброс.
                              ; дальше могут располагаться вектора прерываний
                              ; в соответствии с таблицей прерываний
;-----------------------------------------------------------
; Сегмент кода
;-----------------------------------------------------------

                              ; директива компилятора:
                              ; создается новый кодовый сегмент с именем Blink
rseg     Blink
                              ; относительный нулевой адрес устанавливается на
```

```
                        ; начало сегмента Blink
using   0               ; выбирается 0 банк регистров

Main:                   ; метка Main

                                ; выключение сторожевого таймера.
        anl   PCA0MD, #NOT(040h)
                        ; очистка бита WDTE

                                ; включение матрицы, портов ввода/вывода

                mov   XBR1, #40h
                        ; включение матрицы
                orl   P1MDOUT, #01h
                        ; установка P1.0 (LED) в режим цифрового
                                ; ввода/вывода в режим push-pull.

                        ;Выключение светодиода
                clr   GREEN_LED

                                ; Программно формируемый цикл задержки
        Loop2:     mov   R7, #03h
        Loop1:     mov   R6, #00h
        Loop0:     mov   R5, #00h
                djnz  R5, $
                djnz  R6, Loop0
                djnz  R7, Loop1
                cpl   GREEN_LED
                        ;Переключение светодиода инверсией бита(LED)..
                jmp   Loop2
;------------------------------------------------------------------------
; End of file.

END
```

Рис.14 Код программы «Blink»

Прежде, чем начать отладку проекта, скопируйте папку \Silabs\MCU\Examples\C8051F41x\Blinky\ в свою личную папку.

6.4. Запуск IDE Silabs и открытие файла готового проекта

IDE Silabs запускается из стартового меню Windows подобно остальным приложениям

Для того, чтобы открыть файл проекта, выберите Open Project из меню Project. В окне диалога Open File найдите свой каталог и выберите в нем файл blink.wsp. Файл с расширением WSP содержит конфигурацию проекта. В левом окне IDE появится древовидная структура, отображающая структуру проекта. Щелкнув по иконке любого файла из состава проекта, можно получить его

листинг. Выбрав пункт Build/Make Project или Rebuild Project из меню Project, произведите компиляцию и сборку проекта. Выбрав пункт Connect из меню Debug, проведите связь с отладочным модулем. Выбрав пункт DownLoad Object File из меню Debug, загрузите объектный файл проекта в область Flash памяти программ МК. Далее, воспользовавшись отладочными возможностями (пункты меню GO, STEP, STEP OVER, STOP, GO to cursor), произведите сеанс пошаговой отладки проекта внутри системы (In-system). Программа организует мигание светодиода, используя метод программного формирования задержек. Для того, чтобы наблюдать состояние внутренних регистров, памяти, регистров специальных функций, необходимо воспользоваться меню «View» (рис.15.) и выбрать необходимый раздел, после чего с правой стороны экрана появится таблица с запрашиваемыми параметрами. Этот раздел доступен только при подключенной плате начального освоения.

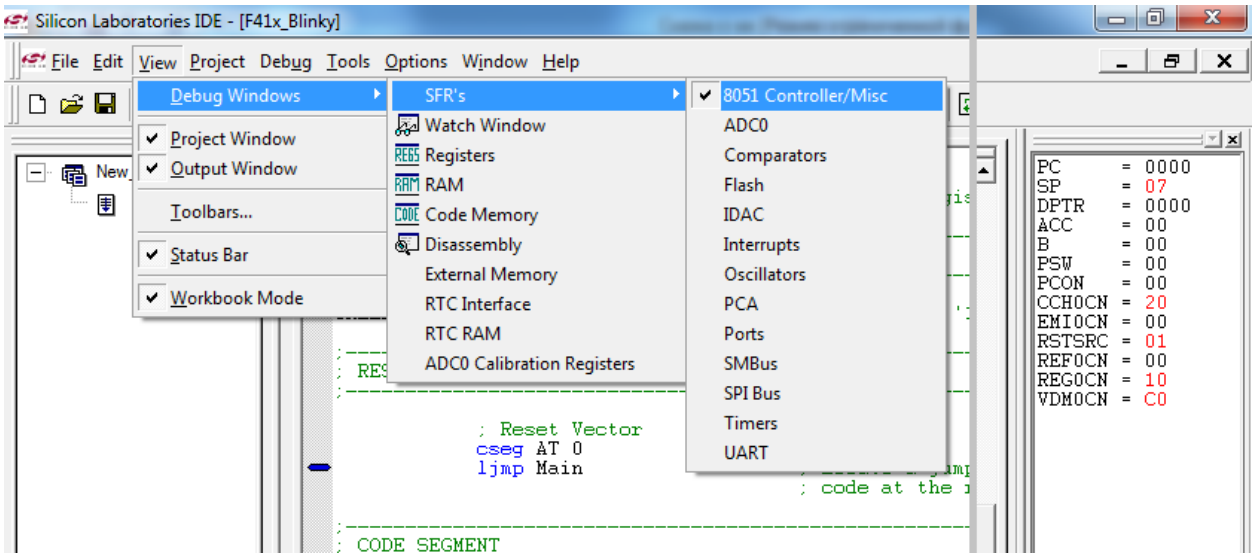

Рис. 15. Просмотр состоянии внутренних регистров C8051F411

6.5. Добавление файла с исходным текстом и его редактирование

Теперь можно добавить Blink.c к проекту. Выберите Add file из меню Project. Откроется диалоговое окно Add File. Выберите Blink.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 blink.c. Выберите blynk.c из окна Project. Нажмите его правой кнопкой мыши и выберите View source file, или просто дважды щёлкните мышью для того, чтобы просматривать файл в окне редактирования.

7. Задачи

7.1. Раздел 1

1. Записать константу в ячейку памяти 22h. Записать константу в R5. Записать константу в регистр R0, скопировать ее в ячейку памяти 21h, проверить состояние 4 бита и в случае, если он равен 1? скопировать R5 по адресу A0h; если 0, то сложить содержимое R0 и ячейки памяти 22h. Проверить с различными константами.
2. Записать константу в ячейку памяти 35h. Записать константу в регистр R0, скопировать ее в ячейку памяти 44h. Проверить состояние 2 бита в ячейке 35h и в случае, если он равен 1, скопировать R0 в R5; если 0, то сложить содержимое 44h и ячейки памяти 35h. Проверить с различными константами.
3. Записать константу в регистр R0 банка памяти 0. Скопировать ее в регистр R5 банка памяти 2, проверить состояние 6 бита и в случае, если он равен 1, скопировать R5 банка памяти 2 по адресу B0h; если 0, то сложить содержимое R0 и с числом 15h. Проверить с различными константами.
4. Записать константу в ячейку памяти 73h. Записать константу в регистр R3 банка памяти 1. Проверить состояние 5 бита в ячейке памяти 73h и в случае, если он равен 1, произвести операцию «или» регистра R3 банка памяти 1 и ячейки памяти 73h, результат записать в регистр R0 банка памяти 1; если 0, то скопировать содержимое R3 банка памяти 1 в ячейку памяти 73h. Проверить с различными константами.
5. Записать константу в ячейку памяти 83h. Записать константу в регистр R7 банка памяти 0. Произвести операцию «исключающее или» регистра R7 банка памяти 0 и ячейки памяти 83h. Проверить состояние 4 бита в ячейке памяти 83h и в случае, если он равен 1, результат «исключающего или» записать в регистр R0 банка памяти 2; если 0, то регистр R3 банка памяти 1. Проверить с различными константами.

6. Записать константу в ячейку памяти 20h. Записать константу в ячейку памяти 32h. Записать константу в ячейку памяти B0h. Записать константу в ячейку памяти A0h, выделить 0,2,5 биты, в случае, если они все равны 1, необходимо сложить содержимое ячеек 20h, 32h. Если нет – сложить содержимое ячеек 20h, B0h. Результат оставить в A. Проверить с различными константами.
7. Записать константу в ячейку памяти 20h. Записать константу в ячейку памяти 32h. Записать константу в ячейку памяти B0h. Записать константу в ячейку памяти A0h, выделить 1,3,5 биты, в случае, если 2 из 3 выделенных битов равны 1, необходимо сложить содержимое ячеек 20h, 32h. Если нет – сложить содержимое ячеек 20h, B0h. Результат оставить в A. Проверить с различными константами.
8. Записать константу в ячейку памяти 20h. Записать константу в ячейку памяти 32h. Записать константу в ячейку памяти B0h. Записать константу в ячейку памяти A0h, выделить 0,4,6 биты, в случае, если хотя бы один выделенный битов равен 1, необходимо сложить содержимое ячеек 20h, 32h. Если нет – сложить содержимое ячеек 20h, B0h. Результат оставить в A. Проверить с различными константами.
9. Записать в ячейки памяти, начинающиеся с адресов 80h и 90h, две 32-х разрядные константы, сложить их, результат сохранить в ячейку памяти, начинающуюся с адреса A0h. Проверить с различными константами.
10. Записать в ячейки памяти, начинающиеся с адресов 80h и 90h, две 32-х разрядные константы, вычесть первую из второй, результат сохранить в ячейку памяти, начинающуюся с адреса A0h. Проверить с различными константами.
11. Записать константу в ячейку памяти 5Ah. Записать константу в ячейку памяти B0h. Выделить 0,3,7 биты в ячейки B0h, в случае, если они все равны 1, необходимо поменять младшие 4 бита (младшую тетраду) ячеек 5Ah, B0h. Если нет – сложить содержимое ячеек 5Ah, B0h. Результат занести в B. Проверить с различными константами.

12. Записать константу в ячейку памяти 5Fh. Записать константу в ячейку памяти C1h. Выделить 0,1,2 биты в ячейки C1h, в случае, если 2 из 3 выделенных битов равны 1, необходимо поменять местами старшие 4 бита (старшую тетраду) ячеек 5Fh, C1h. Если нет – сложить содержимое ячеек 5Fh, C1h. Результат занести в R0. Проверить с различными константами.
13. Записать константу в ячейку памяти 4Ch. Записать константу в ячейку памяти CAh. Выделить 0,4,5 биты в ячейки CAh, в случае, если хотя бы один выделенный бит равен 1, необходимо скопировать старшие 4 бита (старшую тетраду) ячейки 5Fh в C1h, а старшие 4 бита (старшую тетраду) ячейки C1h в регистр R0. Если нет – сложить содержимое ячеек 4Ch, CAh. Результат занести в R0. Проверить с различными константами.
14. Записать константу в ячейку памяти 4Dh. Записать константу в ячейку памяти C8h. Сложить содержимое ячеек, в случае, если произойдет переполнение, скопировать 2,4 биты результата в бит-адресуемую область памяти по адресам 23h, 1Ah. Если переполнения не происходит то повторить сложение до тех пор пока не будет взведен флаг переполнения. Проверить с различными константами.
15. Записать константу в ячейку памяти 41h. Записать константу в ячейку памяти B1h. Вычесть содержимое ячейки 41h из ячейки B1h, в случае, если произойдет заем, скопировать 1,7 биты результата в бит-адресуемую область памяти по адресам 01h, 02h. Если переполнения не происходит, то повторить вычитание до тех пор, пока не будет взведен флаг переполнения. Проверить с различными константами.
16. Записать константу в ячейку памяти 3Ah. Циклически по кругу переставить вправо 2,3,4,5 биты четыре раза. Проверить с различными константами.
17. Записать константу в ячейку памяти 3Ah. Циклически по кругу переставить влево 2,4,5,7 биты четыре раза. Проверить с различными константами.

7.2. Раздел 2

1. Реализовать включение светодиода в момент нажатия кнопки «UP».
2. Реализовать включение светодиода в момент нажатия любой из четырех кнопок.
3. Реализовать переключение светодиода по нажатию кнопки «UP».
4. Реализовать включение светодиода по нажатию кнопки «UP», выключение по нажатию кнопки «DOWN».
5. Реализовать переключение режимов работы светодиода по нажатию кнопки «UP»: погашен, горит, моргает с некоторой частотой.
6. Реализовать переключение светодиода с некоторой частотой, по нажатию кнопки «UP» частота должна увеличиваться, по нажатию кнопки «DOWN» уменьшаться.
7. Реализовать включение и отключение светодиода по нажатию предварительно заданной комбинации из 4 кнопок.
8. Реализовать чтение 4х клавиш и включение светодиода на разное время в зависимости от нажатой клавиши.
9. Реализовать включение светодиода, если время нажатия кнопки «UP» превысило определенный порог, и отключение, если нет.
10. Реализовать включение и отключение светодиода по нажатию предварительно заданной комбинации из 4 кнопок. Ввести ограничение по времени ввода комбинации. На время ввода светодиод должен моргать.
11. Реализовать включение и отключение светодиода по нажатию предварительно заданной комбинации из 2 длительностей нажатия для одной кнопки.

8. Литература

1. Silicon Laboratories. C8051F410/1/2/3. 2.0 V, 32/16 kB Flash, smaRTClock, 12-bit ADC. Rev1.1.– 2008 – 270 p.
2. Silicon Laboratories. C8051F411 EVALUATION KIT USER'S GUIDE Rev. 0.1.– 2006 – 12 p.
3. Keil Software. Cx51 Compiler Optimizing C Compiler and Library Reference for Classic and Extended 8051 Microcontrollers User's Guide. – 2000 – 393 p.
4. Keil Software. Macro Assembler and Utilities Macro Assembler, Linker/Locator, Library Manager, and Object-HEX Converter for 8051, Extended 8051, and 251 Microcontrollers 2001 449 p.
5. Николайчук О. x51-совместимые микроконтроллеры фирмы Cygnal. – М.: ООО «ИД СКИМЕН», 2002. –472 с.
6. Микушин А.В. Занимательно о микроконтроллерах.– СПб.: БВХ-Петербург, 2006. – 432 с.
7. Фрунзе А.В. Микроконтроллеры? Это же просто! Т.1. –М.: ООО «ИД СКИМЕН», 2002. –336 с.
8. Каспер Э. Программирование на языке ассемблера для микроконтроллеров семейства i8051.- М.:Горячяя линия-Телеком, 2004 -191с.
9. Магда Ю. С. Микроконтроллеры серии 8051: практический подход. - М.:ДМК Пресс, 2008 - 228с.

Учебное издание

Латыпов Руслан Рустемович, **Петровнин** Кирилл Викторович,

МИКРОКОНТРОЛЛЕРЫ X51 АРХИТЕКТУРЫ. НАЧАЛЬНОЕ ОСВОЕНИЕ

Дизайн обложки
М.А. Ахметов

Подписано в печать
Бумага офсетная. Печать цифровая.
Формат 60x84 1/16. Гарнитура «Times New Roman». Усл. печ. л. .
Тираж экз. Заказ

Отпечатано с готового оригинал-макета
в типографии Издательства Казанского университета

420008, г. Казань, ул. Профессора Нужина, 1/37
тел. (843) 233-73-59, 233-73-28