ЧП Васильев В.И., эл. почта: vvasiyev@gmail.com,
сайт: www.robot-ivan.narod.ru

Тема: **«Автоматизация строительных технологий»**

Проект: **«Минироботдля нанесения рисунка аэрографом на поверхности плитки»**

Проектная документация
(фрагмент)

г. Кириши, Ленинградская область
2002

ЧП Васильев В.И., эл. почта: vvasiyev@gmail.com,
сайт: www.robot-ivan.narod.ru

Оглавление

ЧП Васильев В.И., эл. почта: vvasiyev@gmail.com,
сайт: www.robot-ivan.narod.ru

Введение

С появлением новых пребований покупателей к разнообразию дизайна облицовочной, гипсовой плитки возникла необходимость автоматизации процесса нанесения рисунков на поверхность. В процессе изготовления плитки необходимо грунтовать поверхность лаком, затем краскораспылителем нанести тонкий фоновый слой. Аэрографом наносится выбранный из каталога рисуннок.

В настоящем проекте предложено решение на основе микроконтроллеров серии AVR, которые обеспечивают управление исполнительными механизмами и цикл нанесения рисунка на поверхность плитки.

Эту задачу целесообразно решать с использованием технологии автоматов. Для ее алгоритмизации и программирования оказалось удобным применить SWITCH-технологию [1].

Настоящая работа призвана автоматизировать этап нанесения рисунка на поверхность плитки, уменьшить брак при произвостве и увеличить производительность производства.

1. Постановка задачи

Целью настоящего проекта является изготовление макета минироботa для нанесения рисунка аэрографом на поверхности облицовочной плитки.

Миниробот состоит из следующих частей:

a) Исполнительный механизм для перемещения аэрографа шаговыми двигателями;
b) Модули управления шаговыми двигателями по осям X, Y, Z;
c) Устройство управления подачей краски для аэрографа;
d) Модуль управления, который формирует задания для остальных модулей (шаговые двигатели, индикации, пульт управления оператором). Модуль управления подключается к персональному компьютеру через интерфейс RS-232C.

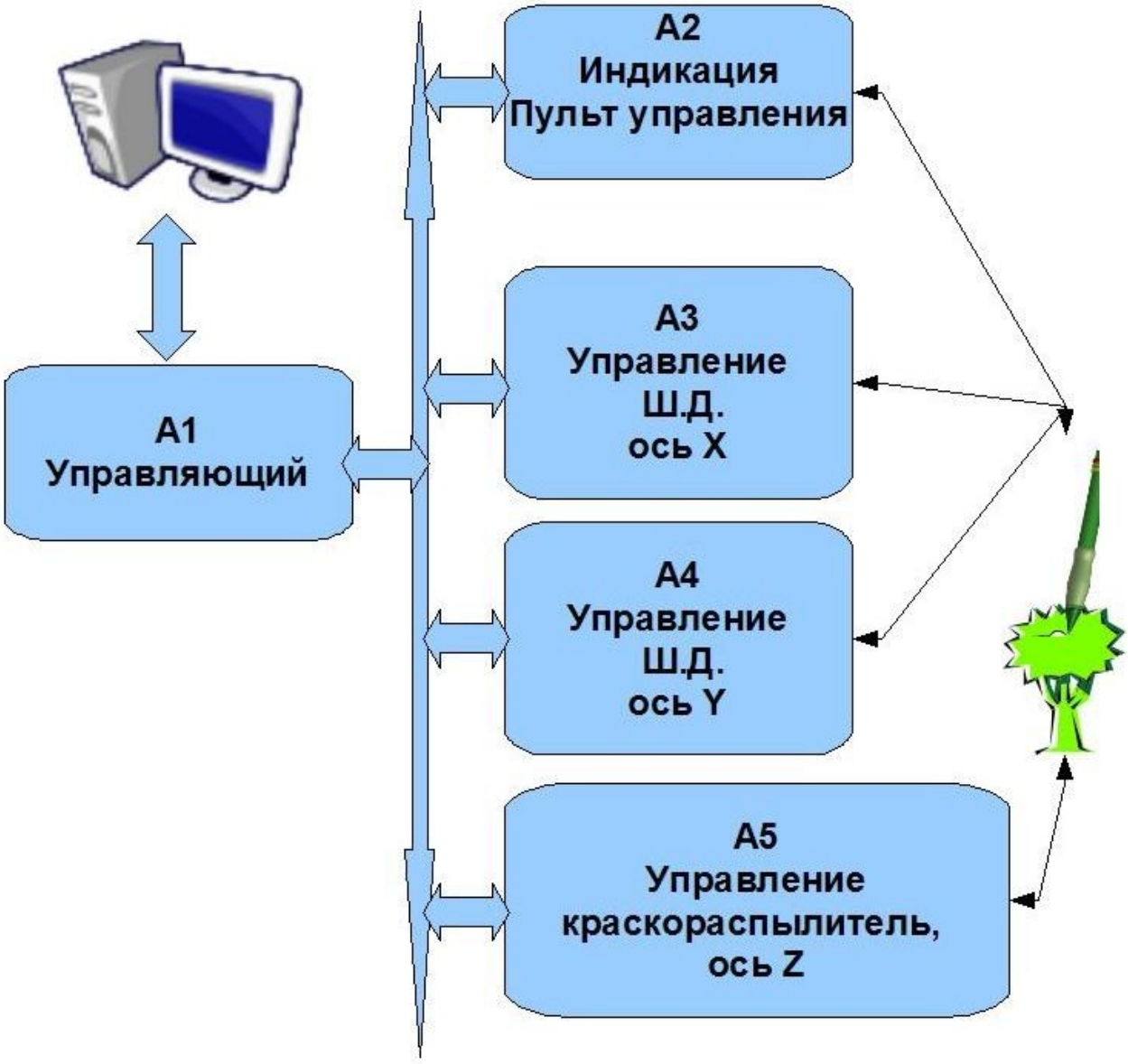

Рис. 1 Структурная схема минироботa

Минироботсостоит из связанных автоматов с явным выделением состояний. Создается схема связи и граф переходов автоматов.

Алгоритмы автоматов реализованы на языках программирования **ASM**, **C** микроконтроллера AVR. На ассемблеры реализованы функции, которые критичны к времени выполнения. Язык программирования **C** позволяет ускорить проектирование, отладку программных модулей автоматов.

Для макетирования и отладки программы использовался набор ***AVR Starter Kit STK500.***

ЧП Васильев В.И., эл. почта: vvasiyev@gmail.com,
сайт: www.robot-ivan.narod.ru

2. Перечень автоматов, входящие в миниробот

Таб. №1 Перечень автоматов

Обозначение	**Наименование**	**Примечание**
A0.x	Командный процессор	Управление минироботом (модуль программный)
A1.x	Ассинхронный приемо-пере-датчик UART0	Программно-аппаратный (вектор прерывания)
A2.x	Пульт управления кнопочный	Программно-аппаратный (вектор прерывания)
A3.x	Алфавитно-цифровой дисплей	Программно-аппаратный (вектор прерывания)

Примечание:

Обозначения AXX.xx
| | |___ Номер вложенного автомата
| |_____ Номер автомата
|_____ Автомат

Таб. №2 Перечень вложенных автоматов A1.xx (UART0)

Обозначение	**Наименование**	**Примечание**
A1.1	Приемник UART0	Программно-аппаратный (вектор прерывания)
A1.2	Передатчик UART0	Программно-аппаратный (вектор прерывания)
A1.3	Контроль потока Rx UART0	Программный
A1.4	Контроль потока Tx UART0	Программный

ЧП Васильев В.И., эл. почта: vvasiyev@gmail.com,
сайт: www.robot-ivan.narod.ru

Таблица №3 Генератор событий eX минироботa (микроконтроллер серии AVR)

```
 *********************************
 **         Генератор
 **          событий
 *********************************
   |
  *|__ e0  ( Reset Системный сброс )
   |__ e1  ( INT0  Внешнее прерывание 0 )
   |__ e2  ( INT1  Внешнее прерывание 1 )
   |__ e3  ( INT2  Внешнее прерывание 2 )
   |__ e4  ( TIM2_Comp  Timer2 прерывание от таймера )
   |__ e5  ( TIM2_Ovf   Timer2 прерывание от таймера )
   |__ e6  ( TIM1_Capt  Timer1 прерывание от таймера )
   |__ e7  ( TIM1_CompA Timer1 прерывание от таймера )
   |__ e8  ( TIM1_CompB Timer1 прерывание от таймера )
   |__ e9  ( TIM1_Ovf   Timer1 прерывание от таймера )
   |__ e10 ( TIM0_Comp  Timer0 прерывание от таймера )
   |__ e11 ( TIM0_Ovf   Timer0 прерывание от таймера )
   |__ e12 ( SPI_STC  Передача байта интерфейса SPI выполнена )
*  |__ e13 ( RXC0   UART0 прием байта )
   |__ e14 ( RXC1   UART1 прием байта )
*  |__ e15 ( DRE0   UART0 регистр пуст )
   |__ e16 ( DRE1   UART1 регистр пуст )
*  |__ e17 ( TXC0   UART0 передан байт )
   |__ e18 ( TXC1   UART1 передан байт )
*  |__ e19 ( EE_DRY EEPROM готовность чтения байт )
   |__ e20 ( ANA_Comp  Аналоговый компаратор )
*  |__ e21 ( KEY_Comp  Изменение состояния кнопочного пульта )
*  |__ e22 ( CMD       Команда управления от внешнего источника )
```

e_gl - Глобальное разрешение/запрещение прерываний, кроме e0.

Примечание: * - Помечены события, которые генерируются в версии программы 0.01

ЧП Васильев В.И., эл. почта: vvasiyev@gmail.com,
сайт: www.robot-ivan.narod.ru

Таблица №4 Схема связи автомата A1.x (UART0)

```
**************************
|| СОБЫТИЯ  для A1.x UART0 ||
**************************
 |
 |__ e0   ( Reset )
 |__ e13  ( RXC0 UART0 принят байт )
 |__ e17  ( TXC0 UART0 передан байт )
 |__ e21  ( KEY_Comp клавиши управления для UART0 )
 |__ e22  ( CMD команда от ПК для UART0 )



**********************************
|  Входные переменные для A1.x UART0 |
**********************************
 |
 |__ x0  ( RXD0 - прием данных или сигнал BREAK от внешнего устройства )
 |__ x1  ( DSR0 - готовность внешнего устройства для приема данных )
 |__ x2  ( CTS0 - готовность внешнего устройства для передачи данных )


+-----------------------------------------------+
| Внутренние переменные A1.x UART0    |
+-----------------------------------------------+
 |
 |__ v0  ( SpeedBaud_UART0  - скорость Rx/Tx бод )
 |__ v1  ( ModeU2X0 - удвоенная скорость Rx/Tx бод )
 |__ v2  ( ModeCHR90 - количество Rx/Tx бит )
 |__ v3  ( CheckParity_UART0 - проверка четности )
 |__ v4  ( ContrStream_UART0 - контроль потока данных )
 |__ v5  ( Enb_DTR0   - разрешить контроль потока данных )
 |__ v6  ( Bit_DTR0   - установленный принудительно сигнал контроля потока данных )
 |__ v7  ( Enb_DSR0   - разрешить контроль потока данных )
 |__ v8  ( Bit_DSR0   - установленный принудительно сигнал контроля потока данных )
 |__ v9  ( Enb_RTS0   - разрешить контроль потока данных )
 |__ v10 ( Bit_RTS0   - установленный принудительно сигнал контроля потока данных )
 |__ v11 ( Enb_CTS0   - разрешить контроль потока данных )
 |__ v12 ( Bit_CTS0   - установленный принудительно сигнал контроля потока данных )
 |__ v13 ( Enb_XON_XOFF_Tx_UART0 - разрешить контроль потока для передачи данных )
 |__ v14 ( Enb_XON_XOFF_Rx_UART0 - разрешить контроль потока для приема данных )
 |__ v15 ( Enb_Echo_UART0 - режим эхо, передавать принятый байт )
 |__ v16 ( ModeMPCM0 - режим мульти-процессор UART0 )
 |__ v17 ( Enb_Err_UART0 - флаг разрешения подсчета ошибок)
 |__ v18 ( Rx_Tx_Err_UART0 - массив подсчета типа и количество ошибок )
```

```
+-------------------------------------------------------------+
| Внутренние переменные A1.1 UART0 приемник   |
+-------------------------------------------------------------+
  |
  |__ v19 ( Count_Rx_UART0  - подсчет количество принятых байт )
  |__ v20 ( Count_Rx_Err_UART0 - подсчет количество ошибочных принятых байт )
  |__ v21 ( RX_BUFFER_SIZE_UART0 - размер буфера приема UART0 )
  |__ v22 ( RX_BUFFER_MASK_UART0 - маска для контроля буфера приема UART0 )
  |__ v23 ( Rx_Head_UART0  - указатель начала буфера приема UART0 )
  |__ v24 ( Rx_Tail_UART0  - указатель конца буфера приема UART0 )
  |__ v25 ( Rx_Buf_UART0   - буфер принятых байт UART0 )
  |
  |__ v26 ( UART0.UCSR0A.RXC0 - принят байт и сохранен в UART0.UDR0 )
  |__ v27 ( UART0.UCSR0A.FE0  - ошибка фрейма приема )
  |__ v28 ( UART0.UCSR0A.OE0  - ошибка переполнения, предыдущий байт не прочитан )
  |__ v29 ( UART0.UCSR0B.RXCIE0 - разрешение прерывания, если байт принят )
  |__ v30 ( UART0.UCSR0B.RXEN0  - разрешение приема данных )
  |__ v31 ( UART0.UCSR0B.RXB80  - прием 9 бита, если режим Rx/Tx 9 бит )

+-------------------------------------------+
| Внутренние переменные A1.2 UART0 передатчик |
+-------------------------------------------+
  |
  |__ v32 ( Count_Tx_UART0  - подсчет количество переданных байт )
  |__ v33 ( Count_Tx_Err_UART0 - подсчет количество ошибочных переданных байт )
  |__ v34 ( TX_BUFFER_SIZE_UART0 - размер буфера передатчика UART0 )
  |__ v35 ( TX_BUFFER_MASK_UART0 - маска для контроля буфера передатчика UART0 )
  |__ v36 ( Tx_Head_UART0  - указатель начала буфера передатчика UART0 )
  |__ v37 ( Tx_Tail_UART0  - указатель конца буфера передатчика UART0 )
  |__ v38 ( Tx_Buf_UART0   - буфер переданных байт UART0 )
  |
  |__ v39 ( UART0.UCSR0A.TXC0 - передан байт из SHIFT reg и нет новых в UART0.UDR0 )
  |__ v40 ( UART0.UCSR0A.UDRE0  - Из UDR0 передан байт в SHIFT reg )
  |__ v41 ( UART0.UCSR0A.UDRIE0 - разрешает прерывание, если UDR0 пуст )
  |__ v42 ( UART0.UCSR0B.TXCIE0 - разрешение прерывания, если байт передан из SHIFT )
  |__ v43 ( UART0.UCSR0B.TXEN0  - разрешение передачи байт )
  |__ v44 ( UART0.UCSR0B.TXB80  - передача 9 бита, если режим Rx/Tx 9 бит )


**********************************
|  Выходные переменные  A1.x UART0 |
**********************************
  |
  |__ z0  ( TXD0  - сигнал BREAK  для внешнего устройства )
  |__ z1  ( DTR0  - готовность UART0 для приема данных )
  |__ z2  ( RTS0  - готовность UART0 передать данные )
  |__ z3  ( LED_DTR0 - индикация "Готовность приема данных UART0" )
  |__ z4  ( LED_RTS0 - индикация "Готовность передачи данных UART0" )
```

```
  |__ z5  ( LED_DSR0 - индикация "Готовность внешнего устройства для приема" )
  |__ z6  ( LED_CTS0 - индикация "Готовность внешнего устройства для передачи" )
  |__ z7  ( LED_TXD  - индикация "Начало передачи байта UART0" импульс )
  |__ z8  ( LED_RXD  - индикация "Начало приема байта UART0" импульс )



#####################################
|  СОСТОЯНИЕ  автомата A1.x UART0 |
#####################################
 |
 |__ y0  ( После системного Reset )
 |__ y1  ( Инициализации внутренних переменных UART0 )
 |__ y2  ( Изменение режимов работы UART0 )
 |__ y3  ( Разрешено Прием и Передача байт UART0 )
 |__ y4  ( Запрещен Прием UART0, Разрешена Передача байт UART0 )
 |__ y5  ( Разрешен Прием UART0, Запрещена Передача байт UART0 )
 |__ y6  ( Запрещен Прием и Передача байт UART0 )

 Примечание:  В переменной Y - сохраняется состояние автомата
```

Рисунок №3 Схема связи автомата A1.3 (Контроль потока Rx UART0)

```
*****************************
||  СОБЫТИЯ  для A1.3 Rx UART0 ||
*****************************
 |
 |__ e0   ( Reset )
 |__ x    ( Значения входных сигналов CTS0, DSR0 )
 |__ CMD  ( Фукции управления DTR0, RTS0, CTS0, DSR0, XON/XOFF)
 |__ Status Rx_Buf ( Состояние буфера приема: заполнен или не заполен)

 +-------------------------------------------------+
 | Внутренние переменные A1.3 Rx UART0    |
 +-------------------------------------------------+
  |
  |__ Соответствуют общим с A1.x и A1.2

#####################################
|  СОСТОЯНИЕ  автомата A1.3 Rx UART0 |
#####################################
 |
 |__ y0  ( Состояние контроля потока: DTR0/DSR0   = Выкл, CTS0/RTS0   = Выкл
 |                          XON/XOFF_Rx = Выкл, XON/XOFF_Tx = Выкл  )
 |                          Bit_DTR0=1 , Bit_DSR0=1
 |                          Bit_CTS0=1 , Bit_RTS0=1
 |__ y1  ( Состояние контроля потока: DTR0/DSR0   = Вкл,  CTS0/RTS0   = Выкл
```

```
  |       аппаратный              XON/XOFF_Rx = {Вкл ,Выкл} )
  |                          Bit_DTR0=DTR0, Bit_DSR0=DSR0
  |                          Bit_CTS0={0,1}, Bit_RTS0={0,1}
  |__ y2  ( Состояние контроля потока: DTR0/DSR0   = Вкл, CTS0/RTS0    = Вкл
  |       аппаратный              XON/XOFF_Rx = {Вкл, Выкл} )
  |                          Bit_DTR0=DTR0, Bit_DSR0=DSR0
  |                          Bit_CTS0=CTS0, Bit_RTS0=RTS0
  |__ y3  ( Состояние контроля потока: DTR0/DSR0   = Выкл, CTS0/RTS0    = Вкл
  |       аппаратный              XON/XOFF_Rx = {Вкл, Выкл} )
  |                          Bit_DTR0={0,1}, Bit_DSR0={0,1}
  |                          Bit_CTS0=CTS0, Bit_RTS0=RTS0
  |__ y4  ( Состояние контроля потока: DTR0/DSR0   ={Вкл,Выкл} , CTS0/RTS0 =
{Вкл,Выкл}
  |       программный             XON/XOFF_Rx = Вкл )
  |                          Enb_XON_XOFF_Rx =1
  |__ y5  ( Состояние контроля потока: DTR0/DSR0   ={Вкл,Выкл} , CTS0/RTS0 =
{Вкл,Выкл}
          программный             XON/XOFF_Rx = Выкл )
                           Enb_XON_XOFF_Rx = 0
```

Таблица №5 Схема связи автомата A1.4 (Контроль потока Tx UART0)

```
*****************************
||  СОБЫТИЯ  для A1.3 Rx UART0 ||
*****************************
  |
  |__ e0   ( Reset )
  |__ x    ( Значения входных сигналов CTS0, DSR0 )
  |__ CMD  ( Фукции управления DTR0, RTS0, CTS0, DSR0, XON/XOFF)
  |__ Status Tx_Buf ( Состояние буфера передатчика: заполнен или не заполен)


########################################
|   СОСТОЯНИЕ  автомата A1.4 Tx UART0 |
########################################
  |
  |__ y0  ( Состояние контроля потока: DTR0/DSR0   = Выкл, CTS0/RTS0   = Выкл
  |                          XON/XOFF_Rx = Выкл, XON/XOFF_Tx = Выкл  )
  |                          Bit_DTR0=1 , Bit_DSR0=1
  |                          Bit_CTS0=1 , Bit_RTS0=1
  |__ y1  ( Состояние контроля потока: DTR0/DSR0   = Выкл,  CTS0/RTS0   = Вкл
  |       аппаратный              XON/XOFF_Tx = {Вкл ,Выкл} )
  |                          Bit_DTR0={0,1}, Bit_DSR0={0,1}
  |                          Bit_CTS0=CTS0,  Bit_RTS0=RTS0
  |__ y2  ( Состояние контроля потока: DTR0/DSR0   = Вкл, CTS0/RTS0    = Вкл
  |       аппаратный              XON/XOFF_Tx = {Вкл, Выкл} )
```

```
  |                       Bit_DTR0=DTR0, Bit_DSR0=DSR0
  |                       Bit_CTS0=CTS0, Bit_RTS0=RTS0
  |__ y3  ( Состояние контроля потока: DTR0/DSR0   = Выкл, CTS0/RTS0    = Выкл
  |       аппаратный             XON/XOFF_Tx = {Вкл, Выкл} )
  |                       Bit_DTR0={0,1}, Bit_DSR0={0,1}
  |                       Bit_CTS0={0,1}, Bit_RTS0={0,1}
  |__ y4  ( Состояние контроля потока: DTR0/DSR0   ={Вкл,Выкл} , CTS0/RTS0 =
{Вкл,Выкл}
  |       программный            XON/XOFF_Tx = Вкл )
  |                       Enb_XON_XOFF_Tx =1
  |__ y5  ( Состояние контроля потока: DTR0/DSR0   ={Вкл,Выкл} , CTS0/RTS0 =
{Вкл,Выкл}
          программный           XON/XOFF_Tx = Выкл )
                        Enb_XON_XOFF_Tx = 0
```

II. УСЛОВИЯ ПЕРЕХОДОВ АВТОМАТА A1x UART0

```
 e0 --> y0  C1x_1    Событие системный сброс --> UART0 выключен;
 y0 --> y1  C1x_2    Состояние A1x y0 --> y1     UART0 выключен;
              Условие инициализации от состояния пульта:
              A2x.y1 --> "Конфигурация заводская"  копировать из ROM
              A2x.y2 --> "Конфигурация текущая" копировать из EEPROM
              Условие инициализации:
              e21.1 "Изменение состояния A2x.y1 --> A2x.y2" -->
                  "Конфигурация текущая" копировать из EEPROM
              e21.2 "Изменение состояния A2x.y2 --> A2x.y1" -->
                  "Конфигурация заводская" копировать из ROM;
 y1 --> y2   C1x_3    Установка режима работы UART0 из переменных:
              v0..v44;
 y2 --> y3   C1x_4    Разрешить работать приему и передачи UART0;

 y3 --> y4   C1x_5    Нет места в буфере передатчика, Не готово внешнее
              устройство или принят сигнал XOFF от внешнего
              устройства;

 y3 --> y5   C1x_6    Нет места в буфере приемника

 y3 --> y6   C1x_7    Нет места в буфере передатчика, Не готово внешнее
 y4 --> y6   C1x_8    устройство или принят сигнал XOFF от внешнего
              нет места в буфере приемника
```

ЧП Васильев В.И., эл. почта: vvasiyev@gmail.com,
сайт: www.robot-ivan.narod.ru

Таблица №6 Схема связи автомата A2.1 (KEY_CONTR)
(клавиатура и выключатели 4 x 4 = 16 клавиш)

```
 +-------------------------------------------------------+
 | Внутренние переменные A2.1 KEY_CONTR   |
 +-------------------------------------------------------+
   |
   |__ Буфер состояний клавиатуры в момент события e1 (глубина N)
   |__ Накопители - сумматоры для линии сканирования клавиш
   |__ Состояние клавиш текущее
   |__ Состояние клавиш предыдущее
   |__ Матрица разрешение/запрещение формирования события (Клавиша нажата)
   |__ Матрица разрешение/запрещение формирования события (Клавиша отжата)
   |__ Матрица разрешения/запрещение формирования события (Клавиша непрерывно
нажата)
   |__ Матрица разрешение/запрещение формирования события (Клавиша непрерывно
отжата)
   |__ TRAN_KEY переменная для состояния y1 (Для переходного процесса)

 ***************************************
 |  Выходные переменные  A2.1 KEY_CONTR |
 ***************************************
   |
   |__ z0 Событие нажатие клавиши
   |__ z1 Состояние отжатие клавиши
   |__ z2 Состояние клавиши нажата ( формирует непрерывный скан-код клавиш )
   |__ z3 Состояние клавиши отжата ( формирует непрерывный скан-код клавиш )

   Формирование события:

   e21   ( KEY_Comp  Изменение состояния кнопочного пульта )


   Примечание:
         Событие формируется с учетом логики матриц
         разрешения/запрещения.

 ########################################
 |  СОСТОЯНИЕ  автомата A2.1 KEY_CONTR |
 ########################################
   |
   |__ y0  ( Reset сброс массивов и переменных )
   |__ y1  ( Ввод состояния клавиатуры без формирования событий e21 и скан-кода )
   |__ y2  ( Обработка клавиатуры и формирование событий e21 и скан-кода )
```

```
*********************************
||  СОБЫТИЯ  для A2.1 KEY_CONTR ||
*********************************
 |
 |__ e0 ( Reset )
 |__ e1 ( Прерывание от таймера )
```

Рисунок №6 Схема связи автомата A2.2 (DISP_CONTR)
(символьный дисплей максимально 16 символов)

```
 +-------------------------------------+
 | Внутренние переменные A2.2 DISP_CONTR  |
 +-------------------------------------+
  |
  |__ ds[16]  Массив из 16-и 8 битовых кодов ( ASCII или BITS поле )
  |__ dan[16] Массив кодов типа битовых кодов и типы анимации для ASCII
  |          или BITS значений
  |__ dst[16] Массив времени действия анимации
  |__ count_time Счетчик выполненных прерываний TIMER0


 ***************************************
 |  Выходные переменные  A2.2 DISP_CONTR |
 ***************************************
  |
  |__ z0 Битовое поле для порта индикации ( 8 бит )
  |__ z1 Битовое поле позиции индикотора ( 4 бит, 16 позиций )


*********************************
||  СОБЫТИЯ  для A2.2 DISP_CONTR ||
*********************************
 |
 |__ e0 ( Reset )
 |__ e1 ( Прерывание от таймера )
 |__ e2 ( Команда от контроллера )


#########################################
|  СОСТОЯНИЕ  автомата A2.2 DISP_CONTR |
#########################################
 |
 |__ y0  ( Reset сброс массивов и переменных )
 |__ y1  ( Вывод информации на индикатор )
```

3. Приложение 1 'Прототипы функций UART0'

```
/* Prototypes Functions for UART0_A1x.c */

/*---------------------------*/
/*  Reset UART */

void    Reset_UART0 ( void );   // state is "Reset System" A1x.y0
      /*      - Запрет прерывания UART0
              - Сигнал DTR0 = 1, RTS0 = 1
              - Регистры UCSR0A,UCSR0B - очистить
              - Массивы и переменные v0..v44 очисить
              - Состояние автомата A1x.y=0 ( начальное ) */

/*--------------------------------*/
/*  Copy Sets Variable UART0 mode */

void   CopySetsRom_UART0(void);       // Sets is Default
void   CopySetsEEPROM_UART0(viod);    // Sets is Profile EEPROM

/*------------------------*/
/*  Initialize UART0 mode */

void    Init_S_UART0  ( SpeedBaund );  // speed  UART0
void    Init_2S_UART0 ( ModeU2X0 );        // baud*2 UART0
void    Init_P_UART0  ( CheckParity ); // control parity
void    Init_CHR9_UART0 ( ModeCHR90 );     // Rx/Tx 8 or 9 bits
void    Init_MPCM_UART0 ( ModeMPCM0 );     // Mode Multi-processor Com port

/*--------------------------*/
/* Received Byte for Rx_Buf */

int     ReceiveByteUART0( void );         // if int = -1 then ERROR
                                  // else int = read data

/*--------------------------*/
/* Transmited Byte to Tx_Buf */

 int    TransmitByteUART0 ( unsigned char data );  // if int = -1 then ERROR
                                        // else int = data

/*-----------------------------------------*/
/* "Enable_Err_Count_UART0" Enable counter   */
/*       errors for Tx_Rx  UART              */
```

```
void   Enable_Err_Count_UART0 ( void );   // End_Err_UART0 = 1

   /*---------------------------------------------*/
   /* :"Disable_Err_Count_UART0" Disable counter   */
   /*       errors for Tx_Rx  UART                */
void   Disable_Err_Count_UART0 ( void );   // End_Err_UART0 = 0

   /*-------------------------------*/
   /* Read Arry Tx_Rx_Err for UART  */
int    Read_Tx_Rx_Err_UART0 ( void );    // Type strings table Error
                                 // if int = -1 ERROR

   /*--------------------------*/
   /* "Echo On UART0"          */
void   Echo_ON_UART0( void );   // Set Echo = 1 Mode 'ON'

   /*--------------------------*/
   /* "Echo OFF UART0"         */
void   Echo_OFF_UART0( void );  // Set Echo = 0 Mode 'OFF'

   /*-- ----------------------*/
   /* "Break UART0" for UART  */
void    Break_UART0( void );               // Set pin AVR TXD = MARK ( Low )

   /*----------------------*/
   /* "Dis UART0"          */
void   Dis_UART0( void );

   /*--------------------*/
   /* "Enb UART0"        */
void    Enb_UART0( void );

    /*--------------------------------*/
    /* "DTR ON CONTROLL UART0"        */
void    DTR_ON_CONTR_UART0( void );     // Control DTR ON
    /*--------------------------------*/
    /* "DTR OFF CONTROL UART0"        */
void    DTR_OFF_CONTR_UART0( void );    // Control DTR OFF
    /*--------------------------------*/
    /* "DTR SET UART0"                */
void    DTR_SET_UART0( void );         // Set DTR = 1
    /*--------------------------------*/
    /* "DTR CLR UART0"                */
void     DTR_CLR_UART0( void );         // Set DTR = 0

   /*---------------------------*/
   /* "RTS_ON_CONTR_UART0"      */
void    RTS_ON_CONTR_UART0( void );    // Control RTS ON
    /*-----------------------------------*/
```

```
    /* "RTS_OFF_CONTR_UART0"       */
  void    RTS_OFF_CONTR_UART0( void );   // Control RTS OFF
     /*-----------------------------*/
    /*  "RTS_SET_UART0"            */
  void     RTS_SET_UART0( void );        // Set RTS = 1
    /*-----------------------*/
    /*  "RTS_CLR_UART0"      */
  void     RTS_CLR_UART0( void );        // Set RTS = 0

    /*-----------------------------*/
    /*  "DSR_ON_CONTR_UART0"       */
  void     DSR_ON_CONTR_UART0( void );    // Control DSR ON
     /*-----------------------------*/
    /*  "DSR_OFF_CONTR_UART0"       */
  void     DSR_OFF_CONTR_UART0( void );   // Control DSR OFF
     /*-----------------------------*/
    /*  "DSR_SET_UART0"            */
  void     DSR_SET_UART0( void );        // Set DSR = 1
    /*----------------------*/
    /* "DSR_CLR_UART0"      */
  void    DSR_CLR_UART0( void );        // Set DSR = 0


    /*---------------------------------*/
    /*  "XON_XOFF_ON_CONTR_Rx_UART0"   */
  void     XON_XOFF_ON_CONTR_Rx_UART0( void ); XON/XOFF  Rx = 1 Enable
    /*---------------------------------*/
    /*  "XON_XOFF_OFF_CONTR_Rx_UART0"   */
  void    _XON_XOFF_OFF_CONTR_Rx_UART0( void ); XON/XOFF Rx = 0 Disable

    /*---------------------------------*/
    /*   "XON_XOFF_ON_CONTR_Tx_UART0"   */
  void      XON_XOFF_ON_CONTR_Tx_UART0( void ); XON/XOFF  Tx = 1 Enable
    /*---------------------------------*/
    /*   "XON_XOFF_OFF_CONTR_Tx_UART0"   */
  void      XON_XOFF_OFF_CONTR_Tx_UART0( void ); XON/XOFF Tx = 0 Disable
```

4. *Приложение 2* ***'Программный код UART0_A1x.c'***

```
/*****************************************
* Programm:      UART0_A1x.c             *
*                                        *
*  Note:  For projekt MINIROBIT BUILDING     *
*                                        *
* Name:  Module Code adapted from Atmel AVR  *
```

```
*       Application for Amega161          *
*       Interrupt mode driver for UART0.    *
*                                 *
* Edit data:     28.08.2002              *
* Last data:     1.10.2002             *
* Version:       0.02                  *
*******************************************/

#include "iom161.h"
#include "uart0_A1x.h"
#include "uart0_A1x_decl.h"
#include "stdio.h"
#include "ctype.h"

/* Debug test UART0 for module */
// #define TEST_UART0


/*---------------------*/
/*  Start Rx/Tx UART0  */
void  Enb_UART0(void)       // Enable  Rx/Tx
{
  UCSR0B = UCSR0B | (1<<TXEN0);  // TXEN0 = 1
  UCSR0B = UCSR0B | (1<<RXEN0);  // RXEN0 = 1
  UCSR0B = UCSR0B | (1<<RXCIE0); // RXCIE0= 1
}
/*--------------------*/
/*  Start Rx UART0  */
void  Enb_Rx_UART0(void)       // Enable  Rx
{
  UCSR0B = UCSR0B | (1<<RXEN0);  // RXEN0 = 1
  UCSR0B = UCSR0B | (1<<RXCIE0); // RXCIE0= 1
}

/*--------------------*/
/*  Start Tx UART0  */
void  Enb_Tx_UART0(void)       // Enable  Tx
{
  UCSR0B = UCSR0B | (1<<TXEN0);  // TXEN0 = 1
}
/*--------------------*/
/*  Stop Rx/Tx UART0  */
void  Dis_UART0(void)       // Disable  Rx/Tx
{
  UCSR0B = UCSR0B & ~(1<<TXEN0);  // TXEN0 = 0
  UCSR0B = UCSR0B & ~(1<<RXEN0);  // RXEN0 = 0
  UCSR0B = UCSR0B & ~(1<<RXCIE0); // RXCIE0 = 0
}
```

```
/*------------------*/
/*  Stop  Rx UART0  */
void  Dis_Rx_UART0(void)      // Disable  Rx
{
 UCSR0B = UCSR0B & ~(1<<RXEN0);  // RXEN0 = 0
 UCSR0B = UCSR0B & ~(1<<RXCIE0); // RXCIE0 = 0
}

/*--------------------*/
/*  Stop   Tx UART0  */
void  Dis_Tx_UART0(void)      // Disable  Tx
{
 UCSR0B = UCSR0B & ~(1<<TXEN0);  // TXEN0 = 0
}


/*---------------------------------*/
/*  Copy Sets Variable UART0 mode */
void  CopySetsROM_UART0(void)      // Sets is Default
    {
 SpeedBaud_UART0 = fSpeedBaud_UART0;
 ModeU2X0 = fModeU2X0;
 ModeCHR90 = fModeCHR90;
 CheckParity_UART0 = fCheckParity_UART0;
 SizeBit_UART0 = fSizeBit_UART0;
 ContrStream_UART0 = fContrStream_UART0;
 Enb_DTR0 = fEnb_DTR0;
 Bit_DTR0 = fBit_DTR0;
 Enb_DSR0 = fEnb_DSR0;
 Bit_DSR0 = fBit_DSR0;
 Enb_RTS0 = fEnb_RTS0;
 Bit_RTS0 = fBit_RTS0;
 Enb_CTS0 = fEnb_CTS0;
 Bit_CTS0 = fBit_CTS0;
 Sym_XON = fSym_XON;
 Sym_XOFF= fSym_XOFF;
 Enb_XON_XOFF_Tx_UART0 = fEnb_XON_XOFF_Tx_UART0;
 Count_XOFF_Tx_UART0 = fCount_XOFF_Tx_UART0;
 Enb_XON_XOFF_Rx_UART0 = fEnb_XON_XOFF_Rx_UART0;
 Count_XOFF_Rx_UART0 = fCount_XOFF_Rx_UART0;
 Enb_Echo_UART0 = fEnb_Echo_UART0 ;
 ModeMPCM0 = fModeMPCM0;
 Enb_Err_UART0 = fEnb_Err_UART0;
 }

/*---------------------------------*/
/*  Copy Sets Variable UART0 mode */
void  CopySetsEEPROM_UART0(void)    // Sets is Profile EEPROM
    {}
```

```
/*------------------*/
/*  Init UART0 mode */
void    Init_S_UART0  ( unsigned int SpeedBaund )  // speed  UART0
       {
   volatile char FlagErr;
   volatile unsigned char cTmp;
   volatile unsigned long lTmp;

   Dis_UART0();
   FlagErr = 1; // Set Flag Error
   cTmp = UCSR0A & 0x02; // Real bit mode U2X0 in reg UCSR0A
   switch (SpeedBaund)
    {
    case 300:
    case 600:
    case 1200:
    case 2400:
    case 4800:
    case 9600:
    case 19200:
    case 38400:
    case 57600:
    if ( cTmp == 0)
      {   // calculate UBR for On Speed Mode
       lTmp = (unsigned long)FQCK / 16 / (unsigned long)SpeedBaund - 1;
       UBRR0 =(unsigned char)lTmp;                 // low byte UBR
       UBRH.uh0 = (unsigned char)(lTmp / 256);     // high byte UBR
       }
    else
      {   // calculate UBR for Double Speed Mode
       lTmp = (unsigned long)FQCK / 8 / (unsigned long)SpeedBaund - 1;
       UBRR0 =(unsigned char)lTmp;                 // low byte UBR
       UBRH.uh0 = (unsigned char)(lTmp / 256);     // high byte UBR
       };
    SpeedBaud_UART0 = SpeedBaund;
    break;
    default:
     // if error speed store old speed
     FlagErr = 0;
    break;
    };
   Enb_UART0();
   }

void    Init_2S_UART0 ( unsigned char ModeU2X0 )        // baud*2 UART0
       {
       Dis_UART0();
       if ( ModeU2X0 == 0)          // if ModeU2X0=0 then U2X0=0
```

```
        {                       // else U2X0=1
        UCSR0A = UCSR0A & ~(0x02);
        Init_S_UART0(SpeedBaud_UART0); // UART0 is 1*SpeedBaund
        }
      else
        {
        UCSR0A = UCSR0A & ~(0x02);
        Init_S_UART0(SpeedBaud_UART0);
        UCSR0A = UCSR0A | (0x02); // UART0 is 2*SpeedBaund
        };
      Enb_UART0();
      }
void   Init_P_UART0  ( unsigned char Parity ) // control parity
      {
      Dis_UART0();
      switch ( Parity )
        {
        case 0:
        Init_CHR9_UART0(0); // Mode 8 bits
        CheckParity_UART0 = Parity;
        break;
        case 1:
        case 2:
        case 3:
        case 4:
        CheckParity_UART0 = Parity;
           switch ( SizeBit_UART0 )
                {
                case 0:
                Init_CHR9_UART0(0); // Mode 8 bits
                case 1:
                Init_CHR9_UART0(1); // Mode 9 bits
                break;
                default:
                Init_CHR9_UART0(0); // Mode 8 bits
                break;
                };
        break;
        default:
        CheckParity_UART0 = 0;
        Init_CHR9_UART0(0); // Mode 8 bits
        break;
        };
      Enb_UART0();
      }

void   Init_B_UART0  ( unsigned char SizeBit )
      {
      Dis_UART0();
```

```
      switch ( SizeBit )
        {
        case 0:
        case 1:
        SizeBit_UART0 = SizeBit;
        break;
        default:
        SizeBit_UART0 = 1;    // SizeBit 8 bits
        break;
        };
      Enb_UART0();
      }

void    Init_CHR9_UART0 ( unsigned char Mode )     // Rx/Tx 8 or 9 bits
      {
      switch ( Mode)
        {
        case 0:
          UCSR0B = UCSR0B & ~(0x04); // Mode 8 bit Rx/Tx
          UCSR0B = UCSR0B & ~(0x01); // Set 9 bit = 0
          ModeCHR90 = Mode;
        break;
        case 1:
          UCSR0B = UCSR0B | (0x04); // Mode 9 bit Rx/Tx
          UCSR0B = UCSR0B | (0x01);  // Set 9 bit = 1
          ModeCHR90 = Mode;
        break;
        default:
          UCSR0B = UCSR0B & ~(0x04); // Mode 8 bit Rx/Tx
          UCSR0B = UCSR0B & ~(0x01); // Set 9 bit = 0
          ModeCHR90 = 0;
        break;
        };
      }

void    Init_MPCM_UART0 ( unsigned char ModeMPCM0 )     // Mode Multi-processor Com port
      {
      Dis_UART0();
      if ( ModeMPCM0 == 0)
      {
     UCSR0A = UCSR0A & ~(0x01); // Mode MPCM0 for Rx/Tx
      }
      else
      {
     UCSR0A = UCSR0A | (0x01); // Mode MPCM0 for Rx/Tx
      };
      Enb_UART0();
      }
```

```
/*-----------------------------------------*/
/*             Reset UART                  */
void  Reset_UART0 ( void )   // Reset UART0 Status A1x.y0
     {
   volatile unsigned char cTmp;
   volatile int i;
   /* Sets out pins   DTR0=1, RTS0=1 */
   cTmp = DDRC ^ ( 1+4 );   // Mode Pins to Out
   DDRC = cTmp;
   cTmp = PORTC ^ ( 1+4 ); // Pins Out bits
   PORTC = cTmp ;
   /* Programm state SYSTEM RESET */
   UCSR0B=0x02;              // Disable UART0 Rx/Tx
   cTmp = UDR0;             // Clear ERROR Resiverd
   UCSR0A=0x20;              // Erase Errors
   UBRH.uh0=0x00;
   UBRR0=0x0;
   /* Erase all variables  */
   /* Erase Buffer Tx */
   for (i=1;i<=TX_BUFFER_SIZE_UART0;i++)
     {
     TxBuf_UART0[i] = 0;         // Buffer Tx Erase
     };
   TxHead_UART0 = 0;            // Head Top
   TxTail_UART0 = 0;            // Tail Top
   /* Erase Buffer Rx */
   for (i=1;i<=RX_BUFFER_SIZE_UART0;i++)
     {
     RxBuf_UART0[i] = 0;         // Buffer Rx Erase
     };
   RxHead_UART0 = 0;            // Head Top
   RxTail_UART0 = 0;            // Tail Top
   /* Erase Array Errors Rx/Tx */
   for (i=1;i<=Rx_Tx_Err_SIZE_UART0;i++)
     {
     Rx_Tx_Err_UART0[i] = 0;         // Array Rx_Tx_Err Erase
     };
   /* Erase variables */
   Count_Rx_UART0 = 0;
   Count_Tx_UART0 = 0;
   Count_Rx_Err_UART0 = 0;
   Count_Tx_Err_UART0 = 0;
   /*  */
   SpeedBaud_UART0 = 0;
   ModeU2X0 = 0;
   ModeCHR90 = 0;
   CheckParity_UART0 = 0;
   SizeBit_UART0 = 1;
   ContrStream_UART0 = 0;
```

```
Enb_DTR0 = 0;
Bit_DTR0 = 1;
Enb_DSR0 = 0;
Bit_DSR0 = 1;
Enb_RTS0 = 0;
Bit_RTS0 = 1;
Enb_CTS0 = 0;
Bit_CTS0 = 1;
Enb_XON_XOFF_Tx_UART0 = 0;
Count_XOFF_Tx_UART0 = 0;
Enb_XON_XOFF_Rx_UART0 = 0;
Count_XOFF_Rx_UART0 = 0;
Enb_Echo_UART0 = 0;
ModeMPCM0 =0;
/*******************/
/* New status A1x.y */
A1x_y = 0;      //  y0 for UART0
/*----------------*/
/* ? status A2x ? */
switch( A2x_y)
{
 case 1: // if A2x_y == y1 Copy sets from EEPROM
 CopySetsEEPROM_UART0();
 break;
 case 2: // if A2x_y == y2 Copy sets from ROM
 CopySetsROM_UART0();
 break;
 default:
  ;  // Error! "No status A2x"
 break;
 };
/*******************/
/* New status A1x.y */
A1x_y = 1;      //  y1 for UART0
/*- ----------------*/
/* Calculator BAUD  */
Init_S_UART0(SpeedBaud_UART0);

/*--------------------*/
/* Enable Rx/Tx UART0 */

// ?? DTR0,RTS0 and enable

/*******************/
/* New status A1x.y */
A1x_y = 2;      //  y2 for UART0

/*--------------------*/
/* Enable Rx/Tx UART0 */
```

```
   /********************/
   /* New status A1x.y */
   A1x_y = 3;      //  y3 for UART0

   }

/*-----------------------------------------*/
/* "Enable_Err_Count_UART0" Enable counter   */
/*        errors for Tx_Rx  UART           */
void   Enable_Err_Count_UART0 ( void )    // Enb_Err_UART0 = 1
   {
   unsigned int i;
   Dis_UART0();
   /* Erase Array Errors Rx/Tx */
   for (i=1;i<=Rx_Tx_Err_SIZE_UART0;i++)
     {
     Rx_Tx_Err_UART0[i] = 0;         // Array Rx_Tx_Err Erase
     };
   /* Erase variables */
   Count_Rx_UART0 = 0;
   Count_Tx_UART0 = 0;
   Count_Rx_Err_UART0 = 0;
   Count_Tx_Err_UART0 = 0;
   /*  */
   Enb_Err_UART0 = 1;
   Enb_UART0();
   }

/*-----------------------------------------*/
/* "Disable_Err_Count_UART0" Disable counter   */
/*        errors for Tx_Rx  UART             */
void   Disable_Err_Count_UART0 ( void )   // End_Err_UART0 = 0
   {
   Enb_Err_UART0 = 0;
    }

/*-----------------------------*/
/* Read Arry Tx_Rx_Err for UART  */
int    Read_Tx_Rx_Err_UART0 ( void )    // Type strings table Error
    {
static volatile unsigned int Arr_Tmp[Rx_Tx_Err_SIZE_UART0];
unsigned int i;
volatile unsigned int iT;

   /* Rear for Arr Errors */
   for (i=1;i<=Rx_Tx_Err_SIZE_UART0;i++)
     {
     Arr_Tmp[i] = Rx_Tx_Err_UART0[i];
     };
```

```
   iT = printf(" \n\r");
   iT = printf("Table Errors Rx/Tx:");
   iT = printf("   ");
   iT = printf("%d", Arr_Tmp[1]);
   iT = printf("   ");
   iT = printf("%d", Arr_Tmp[2]);
   iT = printf("   ");
   iT = printf("%d", Arr_Tmp[3]);
   iT = printf("   ");
   iT = printf("%d", Arr_Tmp[4]);
   iT = printf(" \n\r");
   iT = printf("Table Counters  Rx:");
   iT = printf("   ");
   iT = printf("%ld", Count_Rx_UART0);
   iT = printf("   ");
   iT = printf("%ld", Count_Rx_Err_UART0);
   iT = printf(" \n\r");
   return 0;                        // if int = -1 ERROR
    }

/*-----------------------------*/
/*  Control stream  UART0 mode */
void   ControlStream_ON_UART0( void )
{
   ContrStream_UART0 = 1;
}

/*-----------------------------*/
/*  Control stream  UART0 mode */
void   ControlStream_OFF_UART0( void )
{
   ContrStream_UART0 = 0;
}


/*-------------------------*/
/* "Echo On UART0"         */
void   Echo_ON_UART0( void )   // Set Echo = 1 Mode 'ON'
     {
     Enb_Echo_UART0 = 1;
     }

/*- -------------------------*/
/* "Echo OFF UART0"          */
void   Echo_OFF_UART0( void )  // Set Echo = 0 Mode 'OFF'
     {
     Enb_Echo_UART0 = 0;
     }
```

```
/*-------------------------*/
/* "Break UART0" for UART  */
void   Break_UART0( void )    // Set pin AVR TXD = MARK ( Low )
     {}

/*---------------------------------*/
/*  "DTR ON CONTROLL UART0"        */
void     DTR_ON_CONTR_UART0( void )     // Control DTR ON
     {
     Enb_DTR0 = 1;
     }

/*---------------------------------*/
/*  "DTR OFF CONTROL UART0"        */
void     DTR_OFF_CONTR_UART0( void )    // Control DTR OFF
     {
     Enb_DTR0 = 0;
     }

/*---------------------------------*/
/*  "DTR SET UART0"                */
void     DTR_SET_UART0( void )          // Set signal DTR = 1
     {
     Bit_DTR0 = 1;
     PORT_FLOW_D = PORT_FLOW_D | ( 1<< DTR0 ); // Pin port output
      if ( Bit_DTR0 & 0x01 == 0)
      {
      ClrBit( PORT_FLOW_O, DTR0); // pin output set low 0
      }
      else
      {
      SetBit( PORT_FLOW_O, DTR0); // pin output set high 1
      };
     }

/*---------------------------------*/
/*  "DTR CLR UART0"                */
void     DTR_CLR_UART0( void )          // Set DTR = 0
     {
     Bit_DTR0 = 0;
     PORT_FLOW_D = PORT_FLOW_D | ( 1<< DTR0 ); // Pin port output
      if ( Bit_DTR0 & 0x01 == 0)
      {
      ClrBit( PORT_FLOW_O, DTR0); // pin output set low 0
      }
      else
      {
      SetBit( PORT_FLOW_O, DTR0); // pin output set high 1
      };
```

```
    }

/*-----------------------------*/
/* "RTS_ON_CONTR_UART0"       */
void    RTS_ON_CONTR_UART0( void )    // Control RTS ON
    {
    Enb_RTS0 =1;
    }

/*-----------------------------*/
/*  "RTS_OFF_CONTR_UART0"     */
void    RTS_OFF_CONTR_UART0( void )   // Control RTS OFF
    {
    Enb_RTS0 =0;
    }

/*-------------------------------*/
/*  "RTS_SET_UART0"           */
void    RTS_SET_UART0( void )       // Set RTS = 1
     {
    Bit_RTS0 = 1;
    PORT_FLOW_D = PORT_FLOW_D  | ( 1<< RTS0 ); // Pin port output
     if ( Bit_RTS0 & 0x01 == 0)
     {
     ClrBit( PORT_FLOW_O, RTS0); // pin output set low 0
     }
     else
     {
     SetBit( PORT_FLOW_O, RTS0); // pin output set high 1
     };
     }

/*------------------------*/
/*  "RTS_CLR_UART0"      */
void    RTS_CLR_UART0( void )        // Set RTS = 0
     {
    Bit_RTS0 = 0;
    PORT_FLOW_D = PORT_FLOW_D  | ( 1<< RTS0 ); // Pin port output
     if ( Bit_RTS0 & 0x01 == 0)
     {
     ClrBit( PORT_FLOW_O, RTS0); // pin output set low 0
     }
     else
     {
     SetBit( PORT_FLOW_O, RTS0); // pin output set high 1
     };
     }

/*-------------------------*/
```

```
/*  "DSR_ON_CONTR_UART0"    */
void     DSR_ON_CONTR_UART0( void )    // Control DSR ON
     {
     Enb_DSR0 = 1;
     }

/*-------------------------*/
/* "DSR_OFF_CONTR_UART0"   */
void     DSR_OFF_CONTR_UART0( void )   // Control DSR OFF
     {
     Enb_DSR0 = 0;
     }

/*-------------------------*/
/*  "DSR_SET_UART0"        */
void     DSR_SET_UART0( void )        // Set DSR = 1
     {
     Bit_DSR0 = 1;
     }

/*---------------------*/
/*  "DSR_CLR_UART0"    */
void     DSR_CLR_UART0( void )        // Set DSR = 0
      {
      Bit_DSR0 = 0;
      }

/*-------------------------*/
/*  "CTS_ON_CONTR_UART0"    */
void     CTS_ON_CONTR_UART0( void )    // Control CTS ON
     {
     Enb_CTS0 = 1;
     }

/*-------------------------*/
/* "CTS_OFF_CONTR_UART0"   */
void     CTS_OFF_CONTR_UART0( void )   // Control CTS OFF
     {
     Enb_CTS0 = 0;
     }

/*-------------------------*/
/*  "CTS_SET_UART0"        */
void     CTS_SET_UART0( void )        // Set CTS = 1
     {
     Bit_CTS0 = 1;
     }

/*---------------------*/
```

```
/*  "CTS_CLR_UART0"    */
void     CTS_CLR_UART0( void )        // Set CTS = 0
      {
      Bit_CTS0 = 0;
      }

/*--------------------------------*/
/* "XON_XOFF_ON_CONTR_Rx_UART0"   */
void     XON_XOFF_ON_CONTR_Rx_UART0( void )  // XON/XOFF  Rx = 1 Enable
       {
       Count_XOFF_Rx_UART0 = 0;
       Enb_XON_XOFF_Rx_UART0 = 1;
       }

/*--------------------------------*/
/* "XON_XOFF_OFF_CONTR_Rx_UART0"   */
void     XON_XOFF_OFF_CONTR_Rx_UART0( void ) // XON/XOFF Rx = 0 Disable
       {
       Count_XOFF_Rx_UART0 = 0;
       Enb_XON_XOFF_Rx_UART0 = 0;
       }
/*--------------------------------*/
/* "XON_XOFF_ON_CONTR_Tx_UART0"   */
void     XON_XOFF_ON_CONTR_Tx_UART0( void )  // XON/XOFF  Tx = 1 Enable
       {
       Count_XOFF_Tx_UART0 = 0;
       Enb_XON_XOFF_Tx_UART0 = 1;
       }

/* "XON_XOFF_OFF_CONTR_Tx_UART0"   */
void     XON_XOFF_OFF_CONTR_Tx_UART0( void ) // XON/XOFF Tx = 0 Disable
       {
       Count_XOFF_Tx_UART0 = 0;
       Enb_XON_XOFF_Tx_UART0 = 0;
       }

/**********************/
/* Function IncErrArr */
void IncErrArr_UART0 ( int pos )
{
 if ( Enb_Err_UART0 != 0 )
  {
  Rx_Tx_Err_UART0[pos]++;
  };
}
/*  End IncErrArr    */
/**********************/

/***************************/
```

```
/* Read and write functions */
unsigned char ReceiveByte_UART0( void )
	{
	unsigned char tmptail;
	unsigned char cByte;

	while ( RxHead_UART0 == RxTail_UART0 ) /* wait for incomming data */
		;
	tmptail = ( RxTail_UART0 + 1 ) & RX_BUFFER_MASK_UART0;/* calculate buffer index */
	RxTail_UART0 = tmptail; /* store new index */
	cByte = RxBuf_UART0[tmptail];

	return cByte; /* return data */
	}

void TransmitByte_UART0( unsigned char data )
	{
	unsigned char tmphead;
	/* calculate buffer index */
	tmphead = ( TxHead_UART0 + 1 ) & TX_BUFFER_MASK_UART0;
		/* wait for free space in buffer or Enable Transmit */
	while ( tmphead == TxTail_UART0 )
		;
	TxBuf_UART0[tmphead] = data; /* store data in buffer */
	TxHead_UART0 = tmphead; /* store new index */
	UCSR0B |= (1<<UDRIE0); /* enable UDRE interrupt */
	}

unsigned char DataInReceiveBuffer( void )
	{
	return ( RxHead_UART0 != RxTail_UART0 );
		/* return 0 (FALSE) if the receive buffer is empty */
	}

/*************************************/
/* Function interrupt UART0_RX(void) */
#pragma vector=UART0_RX_vect
static __interrupt void UART0_RX(void)
  {
    unsigned char tmphead;
    unsigned char cTmp;
    unsigned char cByte;
    unsigned char cPERx;
    volatile unsigned char FlgErr;
    volatile int dTest;

/* if enable flow control stream */
     switch (ContrStream_UART0)
      {
```

```
  case 0:
  break;
  default:
    switch (Enb_DTR0)
      {
      case 1:
      PORT_FLOW_D = PORT_FLOW_D | ( 1<< DTR0 ); // Pin port output
        if ( Bit_DTR0 & 0x01 == 0)
        {
        ClrBit( PORT_FLOW_O, DTR0); // pin output set low 0
        }
        else
        {
        SetBit( PORT_FLOW_O, DTR0); // pin output set high 1
        };
      break;
      default:
      break;
      };
  break;
 };

 switch (ContrStream_UART0)
  {
  case 0:
  break;
  default:
    switch (Enb_DSR0)
      {
      case 1:
      PORT_FLOW_D = PORT_FLOW_D & ~( 1<< DSR0 ); // Pin port input
      Bit_DSR0 = ( (PORT_FLOW_I & ( 1<<DSR0)) >> DSR0 ); // Calculate bit
      break;
      default:
      break;
      };
    switch (Enb_CTS0)
      {
      case 1:
      PORT_FLOW_D = PORT_FLOW_D & ~( 1<< CTS0 ); // Pin port input
      Bit_CTS0 = ( (PORT_FLOW_I & ( 1<<CTS0)) >> CTS0 ); // Calculate bit
      break;
      default:
      break;
      };
  break;
 };

FlgErr = 0;  //  if FlgErr=0 then  NO Errors Rx
```

```
//cTmp = UCSR0A & 0x10;  // Test bit FE0=?
if ( (UCSR0A & 0x10) != 0 )
  {
  FlgErr = 1;         // '1' - Rx byte yes Error
  IncErrArr_UART0(1); // increment Rx_Tx_Err_UART0[1]
  };

 cByte = UDR0;                // Read Byte Rx from UDR0

//cTmp = UCSR0A & 0x08; // Test bit OE0=?
 if ( (UCSR0A & 0x08 != 0) )
  {
  FlgErr = 1;
  IncErrArr_UART0(2); // increment Rx_Tx_Err_UART0[2]
     };

/* Calcul Parity for cByte and comperate Bit Parity   */
switch ( SizeBit_UART0)
 {
 case 0:    // Size 7 bits
 cTmp = CheckParity7( cByte) ;
 cPERx = cByte & (1<<7);  // Read 8 Bit cByte Rx
 cPERx = (cPERx>>7);      // cPERx = 0 or 1
 break;
 case 1:
 cTmp = CheckParity8( cByte) ;
 cPERx = (UCSR0B & 0x02) >> 1;  // Read 9 Bit Rx cPEx =0 or 1
 break;
 default:
 break;
 };
  /* return
     if cTmp = 0   then parity cByte
     if cTmp = 255 then no parity
  cPEx =0 or 1
  */

   switch (CheckParity_UART0)
 {
 case 0:
 break;
 case 1:  // Parity E
 cTmp = cTmp & 0x01;
 if (cPERx != cTmp)
 {
 FlgErr =1;
 IncErrArr_UART0(3);   // increment Rx_Tx_Err_UART0[3]
 };
```

```
break;
case 2:  // Parity O
 cTmp = ~cTmp & 0x01;
  if (cPERx != cTmp)
     {
     FlgErr =1;
     IncErrArr_UART0(3);   // increment Rx_Tx_Err_UART0[3]
     };
break;
case 3:
if (cPERx != 1)  // Control Mark
{
FlgErr =1;
IncErrArr_UART0(3);   // increment Rx_Tx_Err_UART0[3]
};
break;
case 4:
if (cPERx != 0)  // Control Space
{
FlgErr =1;
IncErrArr_UART0(3);   // increment Rx_Tx_Err_UART0[3]
};
break;
default:
break;
};

/* Check buffer Rx */
 if ( FlgErr == 0 )  // No Errors for Rx
    {
    /* calculate buffer index */
    tmphead = ( RxHead_UART0 + 1 ) & RX_BUFFER_MASK_UART0;

    if ( tmphead == RxTail_UART0 )
          {
          IncErrArr_UART0(4); // increment Rx_Tx_Err_UART0[4]
          }
    else
    RxHead_UART0 = tmphead; /* store new index */
    {
      switch (ContrStream_UART0) // Begin Switch ContrStream_UART0
          {
          case 0:
          break;
          case 1:  // Yes control stream
              if ( RxHead_UART0 > (RX_BUFFER_SIZE_UART0-4) )
                {
                      /* Signal DTR0 = 0 Stop Tx from PC IBM */
                      /* if enable flow control stream */
```

```
                switch (Enb_RTS0)
                {
                case 1:
                PORT_FLOW_D = PORT_FLOW_D  | ( 1<< RTS0 ); // Pin port output
                SetBit( PORT_FLOW_O, RTS0); // pin output set hing 1
                Bit_RTS0 = 1;
                break;
                default:
                break;
                };
             }
            else
            {
                /* Signal DTR0 = 0 Stop Tx from PC IBM */
                /* if enable flow control stream */
                switch (Enb_RTS0)
                 {
                case 1:
                PORT_FLOW_D = PORT_FLOW_D | ( 1<< RTS0 ); // Pin port output
                ClrBit( PORT_FLOW_O, RTS0); // pin output set high 1
                Bit_RTS0 = 0;
                break;
                default:
                break;
                 };

            };  // End if

           switch(Enb_XON_XOFF_Rx_UART0) // Begin Switch
Enb_XON_XOFF_Rx_UART0
             {
             case 0:
             break;
             case 1:
                 if ( RxHead_UART0 > (RX_BUFFER_SIZE_UART0-4) )
                 {
                   switch( Count_XOFF_Rx_UART0)
                   {
                   case 0:
                   TransmitByte_UART0(Sym_XOFF);
                   Count_XOFF_Rx_UART0++;
                   break;
                   case 1:
                   case 2:
                   case 3:
                   case 4:
                   Count_XOFF_Rx_UART0++;  //increment counter
                   break;
                   case 5:
```

```
                    TransmitByte_UART0(Sym_XOFF);
                    Count_XOFF_Rx_UART0 = 1;
                    break;
                    default:
                    TransmitByte_UART0(Sym_XOFF);
                    Count_XOFF_Rx_UART0 = 1;
                    break;
                    };
                  }
                  else
                  {
                   switch( Count_XOFF_Rx_UART0)
                   {
                   case 0:
                   break;
                   default:
                   TransmitByte_UART0(Sym_XON);
                   Count_XOFF_Rx_UART0 = 0;
                   break;
                   };
                  };
              break;
              default:
              break;
              };  //// End Switch ( Enb_XON_XOFF_Rx_UART0 )

       break;
       default:
       break;
       };  // End Switch ( ContrStream_UART0 )
};
if ( FlgErr == 0)
{
RxBuf_UART0[tmphead] = cByte; /* store received data in buffer */

 switch (ContrStream_UART0)
  {
  case 1:
    switch (Enb_XON_XOFF_Tx_UART0)
     {
     case 0:
     break;
     case 1:
       if ( cByte == Sym_XOFF)
       {
       Count_XOFF_Tx_UART0 = 1;  // Disable Transmit
       };
       if ( cByte == Sym_XON)
       {
```

```
            Count_XOFF_Tx_UART0 = 0;  // Enable Transmit
            };
          break;
          default:
          break;
          };
        break;
        default:
        break;
        };
      };

      /* if Mode Echo then Tx Byte*/
       if ( Enb_Echo_UART0 == 1)
       {
       TransmitByte_UART0(cByte);
       };

      if ( Enb_Err_UART0 == 1 )
        {
       Count_Rx_UART0++;
       };
      }
      else       // Yes Errors Rx for UART0
      {
      if ( Enb_Err_UART0 == 1 )
        {
       Count_Rx_Err_UART0++;
       };
      };
  }
/* End function interrupt UART0_RX(void) */
/*****************************************/


/*****************************************/
/* This method interrupt event TX Empty */
#pragma vector=UART0_UDRE_vect
static __interrupt void UART0_UDRE(void)
   {
   unsigned char tmptail;
   unsigned char cTmp;
   unsigned char cByte;
   unsigned char cPEx;
   unsigned char EnbTx;

   EnbTx = 1;  // Yes Enable Tx

/* if enable flow control stream for signal DTR0 */
```

```
    switch (ContrStream_UART0) // Begin  switch (ContrStream_UART0)
     {
     case 1:
       switch (Enb_DTR0)// Begin switch (Enb_DTR0)
         {
         case 1:
         PORT_FLOW_D = PORT_FLOW_D | ( 1<< DTR0 ); // Pin port output
           if ( Bit_DTR0 & 0x01 == 0)
           {
           ClrBit( PORT_FLOW_O, DTR0); // pin output set low 0
            }
           else
           {
           SetBit( PORT_FLOW_O, DTR0); // pin output set high 1
           };
         break;
         default:
         break;
         };// End switch (Enb_DTR0)
      break;
      default:
      break;
     };// Begin  switch (ContrStream_UART0)

/* if enable flow control stream */
    switch (ContrStream_UART0) // Begin switch (ContrStream_UART0)
     {
     case 1:
       switch (Enb_RTS0)
         {
         case 1:
         PORT_FLOW_D = PORT_FLOW_D | ( 1<< RTS0 ); // Pin port output
           if ( Bit_RTS0 & 0x01 == 0)
           {
           ClrBit( PORT_FLOW_O, RTS0); // pin output set low 0
            }
           else
           {
           SetBit( PORT_FLOW_O, RTS0); // pin output set high 1
           };
         break;
         default:
         break;
         };
      break;
      default:
      break;
     }; // End switch switch (ContrStream_UART0)
```

```
/* if enable flow control stream */
  switch (ContrStream_UART0)// Begin  switch (ContrStream_UART0)
   {
    case 1:
     switch (Enb_DSR0)
       {
       case 1:
       PORT_FLOW_D = PORT_FLOW_D & ~( 1<< DSR0 ); // Pin port input
       Bit_DSR0 = ( (PORT_FLOW_I & ( 1<<DSR0)) >> DSR0 ); // Calculate bit
       break;
       default:
       break;
       };
     switch (Enb_CTS0)
       {
       case 1:
       PORT_FLOW_D = PORT_FLOW_D & ~( 1<< CTS0 ); // Pin port input
       Bit_CTS0 = ( (PORT_FLOW_I & ( 1<<CTS0)) >> CTS0 ); // Calculate bit
       break;
       default:
       break;
       };
    break;
    default:
    break;
   };// End  switch (ContrStream_UART0)

/* ?? Test Controls stream UART0 */

switch ( ContrStream_UART0 )
  {
  case 1:
    if (Enb_XON_XOFF_Tx_UART0 == 1)
    {
     switch (Count_XOFF_Tx_UART0)
       {
       case 0:
       break;
       default:
       EnbTx = 0;  // Disable Tx software flow
       break;
       };
    };
    if ( Enb_CTS0 == 1)
    {
     switch (Bit_CTS0)
       {
       case 1:
       EnbTx = 0;  // Disable Tx hardware flow
```

```
      break;
      default:
      break;
      };
    };
    if (Enb_DSR0 == 1)
    {
     switch (Bit_DSR0)
      {
      case 1:
      EnbTx = 0;  // Disable Tx hardware flow
      break;
      default:
      break;
      };
     };
    if (Enb_RTS0 == 1)
    {
     switch (Bit_RTS0)
      {
      case 1:
      EnbTx = 0;  // Disable Tx hardware flow
      break;
      default:
      break;
      };
     };
     if (Enb_DTR0 == 1)
     {
     switch (Bit_DTR0)
      {
      case 1:
      EnbTx = 0;  // Disable Tx hardware flow
      break;
      default:
      break;
      };
     };
 break;
 default:
 break;
 };

/* Enable Yes then  Transmit Byte */
if ( EnbTx == 1 )
{
  /* check if all data is transmitted */
  if ( TxHead_UART0 != TxTail_UART0 )
    {
```

```
 /* calculate buffer index */
 tmptail = ( TxTail_UART0 + 1 ) & TX_BUFFER_MASK_UART0;
 TxTail_UART0 = tmptail; /* store new index */
 cByte = TxBuf_UART0[tmptail];    // Byte for Tx UART0

 /* Calculate Parity for mode SizeBit */
switch ( SizeBit_UART0)
    {
    case 0:
    cTmp = CheckParity7( cByte) ;
    break;
    case 1:
    cTmp = CheckParity8( cByte) ;
    break;
    default:
    cTmp = CheckParity8( cByte) ;
    break;
    };

 /* For Modes CheckParity_UART0 */
    switch (CheckParity_UART0)
    {
    case 0:
    break;
    case 1:
      if ( cTmp == 0) // cByte parity
      {
       UCSR0B = UCSR0B & ~(1<<TXB80);  // TXB80 = 0
      }
      else
      {
      UCSR0B = UCSR0B | (1<<TXB80);   // TXB80 = 1
      };
    break;
    case 2:

      if ( cTmp == 0) // cByte parity
      {
       UCSR0B = UCSR0B | (1<<TXB80);   // TXB80 = 1
      }
      else
      {
       UCSR0B = UCSR0B & ~(1<<TXB80);  // TXB80 = 0
      };
    break;
    case 3: // Mark
    UCSR0B = UCSR0B | (1<<TXB80);   // TXB80 = 1
    break;
    case 4: // Space
```

```
            UCSR0B = UCSR0B & ~(1<<TXB80);  // TXB80 = 0
            break;
            default:
            UCSR0B = UCSR0B | (1<<TXB80);   // TXB80 = 1
            break;
            };
          /* Set bits for mode SizeBit */
            cPEx = UCSR0B & 0x01;    // Calculate bit TXB80
                                     // cPEx = 0 or 1
          switch ( SizeBit_UART0)
            {
            case 0: // Mode data size 7 bits
            cTmp = cByte | (0x80);     // b7=1
              if ( cPEx == 0)
              {
               ClrBit(cTmp,7);
              }
              else
              {
               SetBit(cTmp,7);
               };
            break;
            case 1: // Mode data size 7 bits
            cTmp = cByte;
            break;
            default:
            break;
            };
           UDR0 = cTmp; /* start transmition */
           }
        else
           {
           UCSR0B &= ~(1<<UDRIE0); /* disable UDRE interrupt */
           };
      }
      else
      {
      UCSR0B &= ~(1<<UDRIE0); /* disable UDRE interrupt */
      };
 }
/* This method interrupt event TX Empty */
/*****************************************/

#ifdef TEST_UART0
/* main - a simple test program*/
void main( void )
       {
       volatile long int l_i;
       volatile unsigned char cByte;
```

```
  volatile int iByte;
  volatile int iRet;
  unsigned int i;
/*
  int x;
  char y;
  float z;
*/
  Reset_UART0();
  Init_S_UART0(57600); // 9600 default
  Init_2S_UART0(0);
  Init_B_UART0(1);  // Size Bits '8'
  Init_P_UART0(0);   // Parity    'N'
                // Stop Bit  '1'
  Echo_OFF_UART0(); // Echo OFF
  ControlStream_OFF_UART0();      // Flow Control ON
  XON_XOFF_OFF_CONTR_Rx_UART0(); // Software Control OFF
  XON_XOFF_OFF_CONTR_Tx_UART0(); // Enable Tx Control
  /* Enable control hardware flow all signals */
   DSR_OFF_CONTR_UART0();  // Hardware Flow Control DSR
   DTR_OFF_CONTR_UART0();  // Hardware Flow Control DTR
   CTS_OFF_CONTR_UART0();  // Hardware Flow Control CTS
   RTS_OFF_CONTR_UART0();  // Hardware Flow Control RTS
   RTS_CLR_UART0();       // RTS0=0 Request to send
   DTR_CLR_UART0();       // DTR0=0 Data Terminal Ready
   CTS_CLR_UART0();
   DSR_CLR_UART0();
  /***********/

  Enable_Err_Count_UART0();
  i = 0;

  _SEI(); /* enable interrupts => enable UART interrupts */
  iRet = printf("Test ver 0.02 for file 'A1x.c' UART0 \n\r");
  if (Enb_XON_XOFF_Rx_UART0 !=  0)
  {
  iRet = printf("%c",Sym_XON);
  Count_XOFF_Rx_UART0 = 0;
  };                        // Enable Tx for Computer

        /* test print */
    /*
        x=1;
        y='t';
        z=1230.09876;
        iRet = printf("%d",x); // yes
        iRet = printf("%c",y); // yes
        iRet = printf("%f",z); // no type "f"
    */
```

```
      /* end print */


while ( 1 ) /* forever */
      {

      /* test new function getchar() and prints stdin */
      do
      {
       iByte = getchar_K();     // new size buffer 40 yes
       iRet = printf("%c",toupper( (unsigned char) iByte));
       if (iByte == 0x0A)  // ^J LF
       {
       iRet = printf("\r");
       };
      }
      while ( iByte != 0x0A);



     /* test function gets( array) */
     /* Error test
      iRet =  (int) gets(test);
      if  ( iRet != 0 )
      {
      iRet = printf("%s",test);
      };
     */

      /* test counters errors */

      if (DataInReceiveBuffer()!=0)
      {
      //cByte = ReceiveByte_UART0();
      //iRet = printf("%c",cByte); // yes
       if ( i > 50)  // if 20 symbols Rx and Tx
       {
       //iRet = Read_Tx_Rx_Err_UART0();
       i = 1;
       };
      i++;
      };

      //iByte = (int) cByte;
      //putchar((int)cByte); // yes
      //TransmitByte(cByte); /* echo the received character */
      //putchar((int)cByte); // yes
      //iRet = sprintf(p,"%ld",Count_Rx_UART0); // yes
      //iRet = puts(p); // yes
```

```
            //iRet = printf("\n\r Test main i="); // yes
            //iRet = printf("%d", i); // yes
            //iRet = printf("%ld",Count_Rx_UART0); // yes
        };
    }
#endif
```

4. ЛИТЕРАТУРА

1. Сайт по автоматному программированию http://is.ifmo.ru
2. AVR Atmel Corporation 8-bit RISC Microcontrollers Data Book, August 1999