Application Note 54

Title: **UART Monitor example on EM68xx**

Product Family: **8-bit Microcontroller**

Part Number: EM6812
Keywords: Microcontroller, measure, sensors
Date: December 10, 2004

Introduction

This application note describes the implementation of a software UART example. This UART is used as a monitor to access internal fields

Monitor example

The goal of this sample project is to realize a basic UART Monitor. Internal CPU registers (data memory and peripheral registers can be accessed for write and read access). It is based on 2 dedicated registers (3 bytes) and 4 commands.

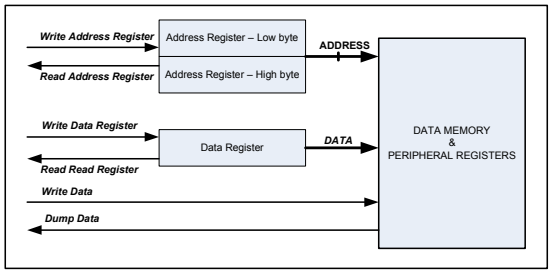

Figure 1: Functions

8-bit Register	Access	Comment
Address register low	Write/Read	Define main address to be pointed (upper byte)
Address register high	Write/Read	Define main address to be pointed (lower byte)
Data register	Write/Read	Define data to be written

Table 1 : Registers

Command	Value	Comment
Write Address Register	0x80	Write the value in Address register
Read Address Register	0x81	Read the value from Address register
Write Data Register	0x82	Write the value in Data register
Read Data Register	0x83	Read the value from Data register
Write Data	0x84	Write Data at pointed address (address register)
Read Data	0x85	Read Data from pointed address (address register)

Table 2: Commands

Protocol

The example implements the following communication:

Type:	Half-full duplex
Baud rate:	2400 bps
Data bit Length:	8-bit
Start bit / Stop bit:	Yes / Yes
Parity:	No
Handshake:	No

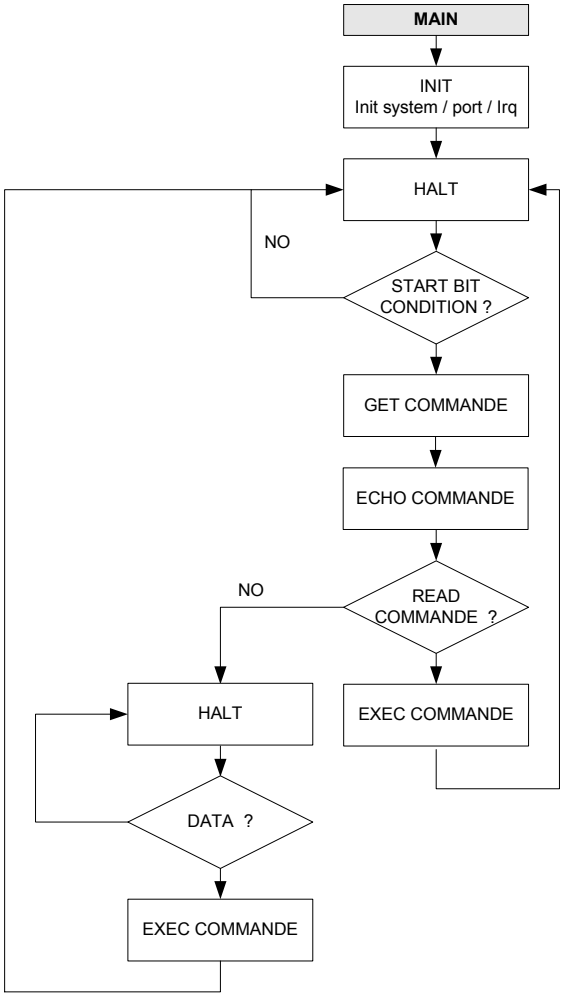

Figure 2: Flowchart

Commands

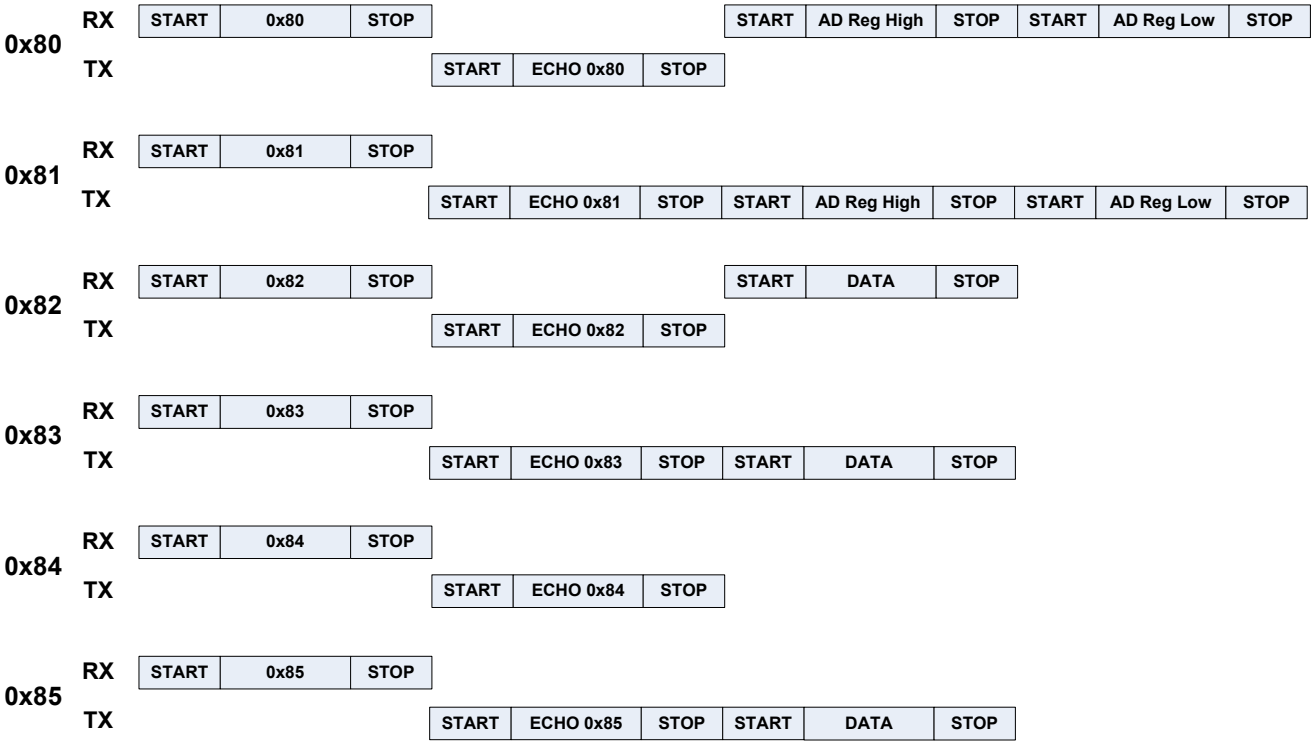

Figure 3 : Command definition

Hardware implementation

The UART use Port A bit 7 as RX pin and Port A bit 6 as TX pin.
These IO's are interfaced with RS232 COM through a MAX3221 RS232 Transceiver.
EMDB6812 support this application.

Software

A Winidea project is available including all source files.
Main source file is available in Appendix. A
A configurable test application (SerialLineTest.exe) has been developed to drive RS232 COM.
This test application is able to send and read data over RS232 COM.

********** APPENDIX A **********

```
//*****************************************************************************
// App:         EM6812 Soft UART / Monitor
// File:        Main.h
// Version:  1.0
// Date:
// Author:    EM Microelectronic
//*****************************************************************************


#ifndef __Main_c__
#define __Main_c__

unsigned char volatile vUART_CNT;
unsigned char volatile vUART_STAT;
unsigned char volatile vUART_BYTE;
unsigned char volatile vCMD;
unsigned int AD_VALUE;
unsigned int* pAD_VALUE;
typedef struct {
  unsigned char AD_VALUE_HIGH;
  unsigned char AD_VALUE_LOW;
  unsigned char DATA_VALUE;
} tTARGET_REG;
tTARGET_REG ACCESS_REG;

#define cUARTBits     8          // Nr of bits
#define cUART1Bit     0x0d       // 32768/14 = 1bit @2341bps
#define cUART15Bit    0x12       // Length of 1.5bits
#define cUART_RDY     0x01       // Ready
#define cUART_RX      0x02       // Receiving
#define cUART_RXD     0x03       // Received
#define cUART_TX      0x04       // Transmitting
#define cUART_IDLE    0x09       // Idle and ready
#define cRXPIN_MSK    0x80       // UART RX pin
#define cTXPIN_MSK    0x40       // UART TX pin
#define cCMD_W_AD     0x80       // write address register value
#define cCMD_R_AD     0x81       // read address register value
#define cCMD_W_DATA   0x82       // write data register value
#define cCMD_R_DATA   0x83       // read data register value
#define cCMD_UPDATE   0x84       // update pointed register (AD) with data (DATA)
#define cCMD_DUMP     0x85       // dump pointer register content

int main(void);
void SysInterrupt1Dispatch(unsigned char RegInt);// Priority level 1
void SysInterrupt2Dispatch(unsigned char RegInt);// Priority level 2
void SysInterrupt0Dispatch(unsigned char RegInt);// Priority level 2
void SendUART(unsigned char Val, unsigned char TXStatus);
unsigned char GetUART(void);
void ExecCommand (unsigned char Command_name);
void SetHighSpeed(void);
void SetLowSpeed(void);

#endif /* __Main_c__ */
```

```
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// App:         EM6812 Soft UART / Monitor
// File:        Main.c
// Version:  1.0
// Date:
// Author:    EM Microelectronic
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

#include "Periph.H"
#include "Main.H"
#include "Macro.H"

// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// Interrupt0, not used
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
void SysInterrupt0Dispatch(unsigned char RegInt) // Priority level 2
{}

// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// Interrupt1, used for UART RX-Pin / PA7
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
void SysInterrupt1Dispatch(unsigned char RegInt)
{
// INT1: Called by crt0.s, var RegInt contains RegInt10
// Int1_7 = Irq_PA7

  if ((RegInt & Int1_7) > 0)         // Source = PA7
  {
    RegOutPB = ~RegOutPB; // debug only !
    RegOutPA       ^= 0x01;
    RegMsk10       &= (0xff-Msk1_7); // Disable Interrupt of RXD pin;
    vUART_CNT      = cUARTBits;  // 8 bits config
    vUART_STAT     = cUART_RX;   // Reception
    RegTim1Load    = cUART15Bit; // Delay to sample LSB middle
    RegTimersStart |= SWStart1;  // Timer Start
  }
}
```

```
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// Interrupt2, used for the Timers
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
void SysInterrupt2Dispatch(unsigned char RegInt)
{
// INT2: Called by crt0.s, var RegInt contains RegInt20
// Int2_0 = Irq_Tim1

  RegTim1Load    = cUART1Bit; // Delay to sample next LSB middle
  if ((RegInt & Int2_0) > 0)   // Source = Timer1
  {
     delay;
     RegOutPB = ~RegOutPB; // debug only !
     if (  ((vUART_STAT & cUART_TX) == 0)  )  // Transmit or receive?
      {
        if (vUART_CNT > 0)                    // Receive!
          vUART_BYTE = (vUART_BYTE >> 1) | (RegInPA & cRXPIN_MSK);
        else
          Nop;
      }
     else
      {                                 // Transmit
        if (vUART_CNT == 1)
          RegOutPA |= cTXPIN_MSK;             // Send Stop bit
        else if (vUART_CNT > 0)
         {
          if ((vUART_BYTE & 0x01) > 0)
            RegOutPA |= cTXPIN_MSK;           // Send 1
          else
             RegOutPA &= (0xff-cTXPIN_MSK);  // Send 0
          vUART_BYTE >>= 1;
         }
      }
     if (vUART_CNT |= 0)
        vUART_CNT--;
     else
     {
       RegTimersStart &= (0xff - SWStart1);      // Stop Timer
       vUART_STAT |=  cUART_RDY;
       RegMsk10 |= Msk1_7;                    // Reenable PA7 interrupt for next byte
     }
   }
}

// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// Return one byte on TXD
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void SendUART(unsigned char Val, unsigned char TXStatus)
{
  while ((vUART_STAT & cUART_RDY) == 0x00) {}; // wait if not ready
  RegMsk10 &= (0xff-Msk1_7);                  // Disable Interrupt of RXD pin
  vUART_BYTE = Val;
  vUART_STAT = TXStatus;                   // Set Status
  vUART_CNT = cUARTBits+1;                 // 8+1 bits
  RegOutPA &= (0xff-cTXPIN_MSK);           // Set Start bit
  RegTim1Load = cUART1Bit;                 // Load Timer with Bit length
  RegTimersCfg |= AR1;                    // Autoreload
  RegTimersStart |= SWStart1;              // Restart Timer
}
```

```
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// load one byte from RXD
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

unsigned char GetUART(void)
{
  while (vUART_STAT != cUART_RXD) {};      // Wait until SW-UART ready
  vUART_STAT = cUART_IDLE;
  RegMsk10 &= (0xff-Msk1_7);               // Disable Interrupt of RXD pin
  return vUART_BYTE;                      // return the received byte
}

// ****************************************************
// Routines for switching the CPU Speed
// ****************************************************
void SetHighSpeed(void)
{
       RegSys1 |= EnRC;                          // Value for 1MHz, RC On
       RegSys2 &= (0xff - Sel32k);                  // Switch to RC
}

void SetLowSpeed(void)
{
       RegSys2 |= Sel32k;                        // switch cpu to 32kHz
       RegSys1 &= (0xff - EnRC);                    // Disable RC
}

// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// Execute commands
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
void ExecCommand (unsigned char Cmd_name)
{

  if (Cmd_name == cCMD_W_AD)
  {
    ACCESS_REG.AD_VALUE_HIGH = GetUART();
    RegMsk10 |= Msk1_7;
    ACCESS_REG.AD_VALUE_LOW  = GetUART();
  }
  else if (Cmd_name == cCMD_W_DATA)
  {
    ACCESS_REG.DATA_VALUE = GetUART();
  }
  else if (Cmd_name == cCMD_R_AD)
  {
    SendUART(ACCESS_REG.AD_VALUE_HIGH, cUART_TX);
    SendUART(ACCESS_REG.AD_VALUE_LOW, cUART_TX);
  }
  else if (Cmd_name == cCMD_R_DATA)
  {
    SendUART(ACCESS_REG.DATA_VALUE, cUART_TX);
  }
  else if ( (Cmd_name == cCMD_UPDATE) | (Cmd_name == cCMD_DUMP) )
  {
    AD_VALUE = ( ((ACCESS_REG.AD_VALUE_HIGH << 8) & 0xFF00) | (ACCESS_REG.AD_VALUE_LOW & 0x00FF) );
    pAD_VALUE = (int*) AD_VALUE;
    if (Cmd_name == cCMD_DUMP)  // dump
      {
        SendUART(*pAD_VALUE, cUART_TX);
      }
    else
      {
        *pAD_VALUE =  ACCESS_REG.DATA_VALUE;  // update
      }
  }

}
```

```
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// MAIN, Initialises the EM6812 and handles the
//       dispatching of the commands received by
//       the software UART
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
int main(void)
{


  DisWdog();
  // Init IO & SysRegs
  RegIOSelPB = 0xFF;                    // debug only !
  RegOutPB = 0xFF;                      // debug only !
  RegPullUpPA   = cRXPIN_MSK;           // Enable pulls
  RegPullDownPA ^= cRXPIN_MSK;
  RegIOSelPA   = cTXPIN_MSK | 0x01;     // Only TXPin is output (no floating)
  RegOutPA     = cTXPIN_MSK;            // Only TXPin is '1'
  RegSys1      = EnRC | FreqRange;      // Enable RC
  RegSys2      = EnXtal ;               // Enable Xtal
  RegTrimRC    = 0x80;                  // Value for 1MHz
  // Init the vars
  vUART_STAT    = cUART_IDLE;
  *pAD_VALUE = 0;
  // Wait for Cold start
  while ((RegSys1 & FlagXtal) == 0) {}
  // switch Prescaler to 32kHz
  RegPrscCkSel = 0x16;                  // 32K AutoSel mode ... allow to switch off Xtal
  // Preset the Timers
  RegTim1Cfg = Clk1Sel_0 | Clk1Sel_2;   // Timer1 Source = 32kHz (Presc1 srce)
  RegTimersCfg = AR1;                   // Autoreload for Timer1
  // Enable Interrupts
  RegMsk10     = Msk1_7;                // Enable Interrupts for PA7
  RegMsk20     = Msk2_0;                // Enable timer interrupt
  // Start now !
  while (1)                             // Main endless loop
   {
     if (vUART_STAT == cUART_RXD)
       {
         vCMD = vUART_BYTE;
         SendUART(vCMD, cUART_TX);
         if ( (vCMD == cCMD_W_DATA) |    // Check for valid command
              (vCMD == cCMD_R_DATA) |
              (vCMD == cCMD_W_AD)   |
              (vCMD == cCMD_R_AD)   |
              (vCMD == cCMD_UPDATE) |
              (vCMD == cCMD_DUMP)   )
                ExecCommand(vCMD);
         while ((vUART_STAT & cUART_RDY)== 0x00) {}; // Wait until UART terminated
         // Enable Interrupts
         RegMsk10     = Msk1_7;                  // Re-Enable Interrupts for PA7
       }
       Halt;                                // Stop the cpu and wait for interrupts
     }
}
```