

# USB-Serial Hub Design

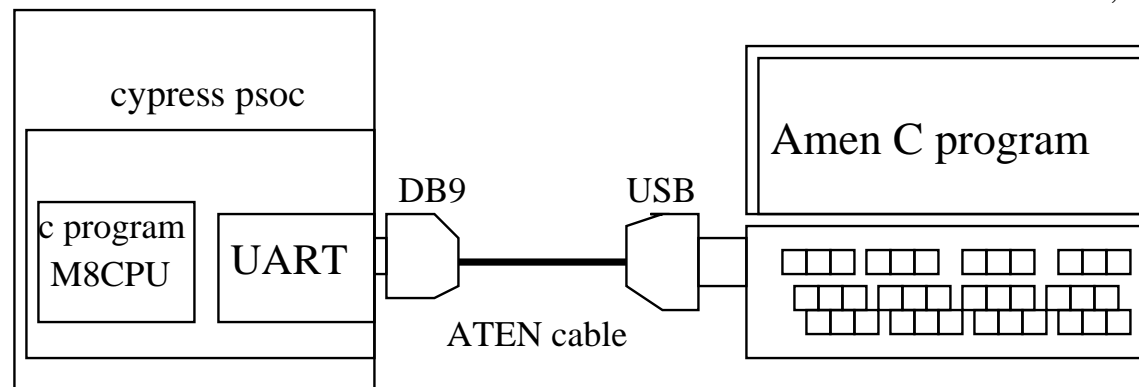
## Hardware

- Use Aten usb-serial cable
- Use Aten usb-serial device driver
- Use Cypress psoc UART on Hub board

## Software

- C programs (two versions, console and windows) on laptop (Visual C version 7 2003)
- C program (firmware) on Cypress psoc, cypress icc compiler

nov 27, 2008



# Serial Program, console 32 application

```
// testdear.cpp : Defines the entry point for the console application.
//
// demo simple serial read from Cypress UART
// COM4 is used by the ATEN usb to serial cable
// Nov 25 2008
// Baud rate is 115200

#include "stdafx.h"
#include "fcntl.h"
#include "windows.h"
#include "conio.h"

int _tmain(int argc, _TCHAR* argv[])
{
    HANDLE hSerial=NULL;
    char sInBuff[51]={0};
    DWORD dwByteRead = 0;

    printf("Yes Dear\n");

    //
    // COM4 is used by the ATEN usb to serial cable
    //
    hSerial = CreateFile("com4", GENERIC_READ, 0,0, OPEN_EXISTING,
                        FILE_ATTRIBUTE_NORMAL, 0);

    if(hSerial==INVALID_HANDLE_VALUE) {
        printf("No Good\n");
        exit(-1);
    }
    else
        printf("Good comm4 open, amen !\n");
}
```

```
// Setup Comm port

DCB dcbSerialParams = {0};
dcbSerialParams.DCBlength = sizeof(dcbSerialParams);

if(!GetCommState(hSerial, &dcbSerialParams)){
    printf("Can't get comm param");
    CloseHandle(hSerial);
    exit(-1);
}
//

dcbSerialParams.BaudRate = CBR_115200; // must match the Cypress hardware
dcbSerialParams.ByteSize = 8;
dcbSerialParams.StopBits = ONESTOPBIT;
dcbSerialParams.Parity = NOPARITY;

// Now set the Uart param

if(!SetCommState(hSerial, &dcbSerialParams)){
    printf("Can't set comm param");
    CloseHandle(hSerial);
    exit(-1);
}

// set a large number like 50000 for blocking read from comm
//
COMMTIMEOUTS timeouts = {0};
timeouts.ReadIntervalTimeout = 500;
timeouts.ReadTotalTimeoutConstant = 5000;
timeouts.ReadTotalTimeoutMultiplier = 10;
timeouts.WriteTotalTimeoutConstant = 50;
```

```
timeouts.WriteTotalTimeoutMultiplier = 10;

SetCommTimeouts(hSerial, &timeouts);

// Now read
// what you read depends on your timeouts
// this is non blocking
while(1)
{
    unsigned char inbyte;
    ReadFile(hSerial, sInBuff, 1, &dwByteRead, NULL);
    inbyte = sInBuff[0] & 0xFF; // mask out the sign bit
    if(dwByteRead !=0)
        printf("%d %x %x\n", dwByteRead, sInBuff[0], inbyte);
}
// now happily return after
CloseHandle(hSerial);
return 0;
}
```

# Serial Program, windows win32 application

```
// newcom.cpp : Defines the entry point for the application.
//
#include "stdafx.h"
#include "newcom.h"
#include <commctrl.h>
#include <fcntl.h>
#include <conio.h>

#define MAX_LOADSTRING 100

HWND hEdit=NULL;
const int ID_TIMER =100;
// Global Variables:
HINSTANCE hInst; // current instance
TCHAR szTitle[MAX_LOADSTRING]="AMEN sensor console";// The title bar text
TCHAR szWindowClass[MAX_LOADSTRING]; // the main window class name

HANDLE hSerial=NULL;
char sInBuff[51]={0};

// Forward declarations of functions included in this code module:
ATOM MyRegisterClass(HINSTANCE hInstance);
BOOL InitInstance(HINSTANCE, int);
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

int APIENTRY _tWinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPTSTR lpCmdLine,
                     int nCmdShow)
{
    // TODO: Place code here.
```

```
MSG msg;
HACCEL hAccelTable;

// Initialize global strings
LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
LoadString(hInstance, IDC_NEWCOM, szWindowClass, MAX_LOADSTRING);
MyRegisterClass(hInstance);

// Perform application initialization:
if (!InitInstance (hInstance, nCmdShow))
{
return FALSE;
}

hAccelTable = LoadAccelerators(hInstance, (LPCTSTR)IDC_NEWCOM);

//
// COM4 is used by the ATEN cable
//
hSerial = CreateFile("com4", GENERIC_READ, 0,0, OPEN_EXISTING,
                    FILE_ATTRIBUTE_NORMAL, 0);

if(hSerial==INVALID_HANDLE_VALUE) exit(-1);

// Setup Comm port

DCB dcbSerialParams = {0};
dcbSerialParams.DCBlength = sizeof(dcbSerialParams);

if(!GetCommState(hSerial, &dcbSerialParams)){
    CloseHandle(hSerial); exit(-1);
}
//
```

```
dcbSerialParams.BaudRate = CBR_115200; // must match the Cypress hardware
dcbSerialParams.ByteSize = 8;
dcbSerialParams.StopBits = ONESTOPBIT;
dcbSerialParams.Parity = NOPARITY;

// Now set the param

if(!SetCommState(hSerial, &dcbSerialParams)){
    CloseHandle(hSerial);
    exit(-1);
}

// set a large number like 50000 for blocking read from comm
//
COMMTIMEOUTS timeouts = {0};
timeouts.ReadIntervalTimeout = 50;
//timeouts.ReadTotalTimeoutConstant = 5000;
timeouts.ReadTotalTimeoutConstant = 40;
timeouts.ReadTotalTimeoutMultiplier = 10;
timeouts.WriteTotalTimeoutConstant = 50;
timeouts.WriteTotalTimeoutMultiplier = 10;

SetCommTimeouts(hSerial, &timeouts);

// Main message loop:
while (GetMessage(&msg, NULL, 0, 0))
{
if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
{
TranslateMessage(&msg);
DispatchMessage(&msg);
}
}
```

```

}

// now happily return after

CloseHandle(hSerial);

return (int) msg.wParam;
}

//
// FUNCTION: MyRegisterClass()
//
// PURPOSE: Registers the window class.
//
// COMMENTS:
//
// This function and its usage are only necessary if you want this code
// to be compatible with Win32 systems prior to the 'RegisterClassEx'
// function that was added to Windows 95. It is important to call this function
// so that the application will get 'well formed' small icons associated
// with it.
//
ATOM MyRegisterClass(HINSTANCE hInstance)
{
WNDCLASSEX wcex;

wcex.cbSize = sizeof(WNDCLASSEX);

wcex.style = CS_HREDRAW | CS_VREDRAW;
wcex.lpfnWndProc = (WNDPROC)WndProc;
wcex.cbClsExtra = 0;
wcex.cbWndExtra = 0;

```

```

wcex.hInstance = hInstance;
wcex.hIcon = LoadIcon(hInstance, (LPCTSTR)IDI_NEWCOM);
wcex.hCursor = LoadCursor(NULL, IDC_ARROW);
wcex.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
wcex.lpszMenuName = NULL;
wcex.lpszClassName = szWindowClass;
wcex.hIconSm = LoadIcon(wcex.hInstance, (LPCTSTR)IDI_SMALL);

return RegisterClassEx(&wcex);
}

//
//  FUNCTION: InitInstance(HANDLE, int)
//
//  PURPOSE: Saves instance handle and creates main window
//
//  COMMENTS:
//
//      In this function, we save the instance handle in a global variable and
//      create and display the main program window.
//
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
    HWND hWnd;

    hInst = hInstance; // Store instance handle in our global variable

    hWnd = CreateWindowEx(0, szWindowClass, "AMEN Sensor Console",
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, CW_USEDEFAULT, 320, 350, NULL, NULL, hInstance, NULL);

    if (!hWnd)
    {

```

```

    return FALSE;
}

ShowWindow(hWnd, nCmdShow);
UpdateWindow(hWnd);

return TRUE;
}

//
// FUNCTION: WndProc(HWND, unsigned, WORD, LONG)
//
// PURPOSE: Processes messages for the main window.
//
// WM_COMMAND - process the application menu
// WM_PAINT - Paint the main window
// WM_DESTROY - post a quit message and return
//
//
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    int wmId, wmEvent;
    PAINTSTRUCT ps;
    HDC hdc;

    static int myi=0;
    static char my[80]={0};
    unsigned char inbyte;
    DWORD dwByteRead = 0;

    my[0]='C'; my[1]='M'; my[2]='P'; my[3]='E'; my[4]='1';
    my[5]='2'; my[6]='3'; my[7]='a'; my[8]='\r'; my[9]='\n';
    my[10]=' ';
    my[11]='H'; my[12]='u'; my[13]='m'; my[14]='d'; my[15]=':.';

```

```
my[16]='1'; my[17]='2'; my[18]='\r'; my[19]='\n';
my[20]=' ';
my[21]='W'; my[22]='i'; my[23]='n'; my[24]='d'; my[25]=': ';
my[26]=' '; my[27]='3'; my[28]='1'; my[29]='\r'; my[30]='\n';
my[31]=' ';
my[32]='T'; my[33]='e'; my[34]='m'; my[35]='p'; my[36]=': ';
my[37]='8'; my[38]='5'; my[39]='\r'; my[40]='\n';
```

```
switch (message)
```

```
{
```

```
case WM_CREATE:
```

```
{
```

```
    HFONT hfDefault;
```

```
hEdit = CreateWindowEx(WS_EX_CLIENTEDGE, "EDIT", "",
```

```
WS_CHILD | WS_VISIBLE | WS_VSCROLL | WS_HSCROLL | ES_MULTILINE | ES_AUTOVSCROLL | ES_AUTOHSCROLL,
0, 0, 80, 80, hWnd, (HMENU)IDC_NEWCOM, GetModuleHandle(NULL), NULL);
```

```
    hfDefault = (HFONT) GetStockObject(DEFAULT_GUI_FONT);
```

```
    SetTimer(hWnd, ID_TIMER, 200, NULL);
```

```
}
```

```
break;
```

```
case WM_TIMER:
```

```
{
```

```
    myi = myi + 1;
```

```
    my[17] = '0' + myi%9;
```

```
    my[28] = '0' + myi%7;
```

```
    my[38] = '0' + myi%8;
```

```
// Now read
```

```
    int count=0;
```

```
    int retry=0;
```

```

while(++retry < 10){
dwByteRead=1;

// character formatter
while(dwByteRead !=0)
{
ReadFile(hSerial, sInBuff, 1, &dwByteRead, NULL);
unsigned int bias=0;

if(dwByteRead > 0) {

inbyte = sInBuff[0] & 0xFF;

if(inbyte < 16) {
if(inbyte < 10) my[41+count] = '0'+inbyte;
else my[41+count] = 'a'-10+inbyte;
bias = 0;
} else {
unsigned char byte1;
unsigned char byte2;
byte2 = inbyte & 0xF0;
byte2 = byte2 >> 4; // shift
byte1 = inbyte & 0x0F;

my[41+count] = (byte2>9)? ('a'-10):'0'+byte2;
my[42+count] = (byte1>9)? ('a'-10):'0'+byte1;

bias = 1;
}
my[41+count+1+bias] = '\r';
my[41+count+2+bias] = '\n';
count += (3 + bias);
}
}

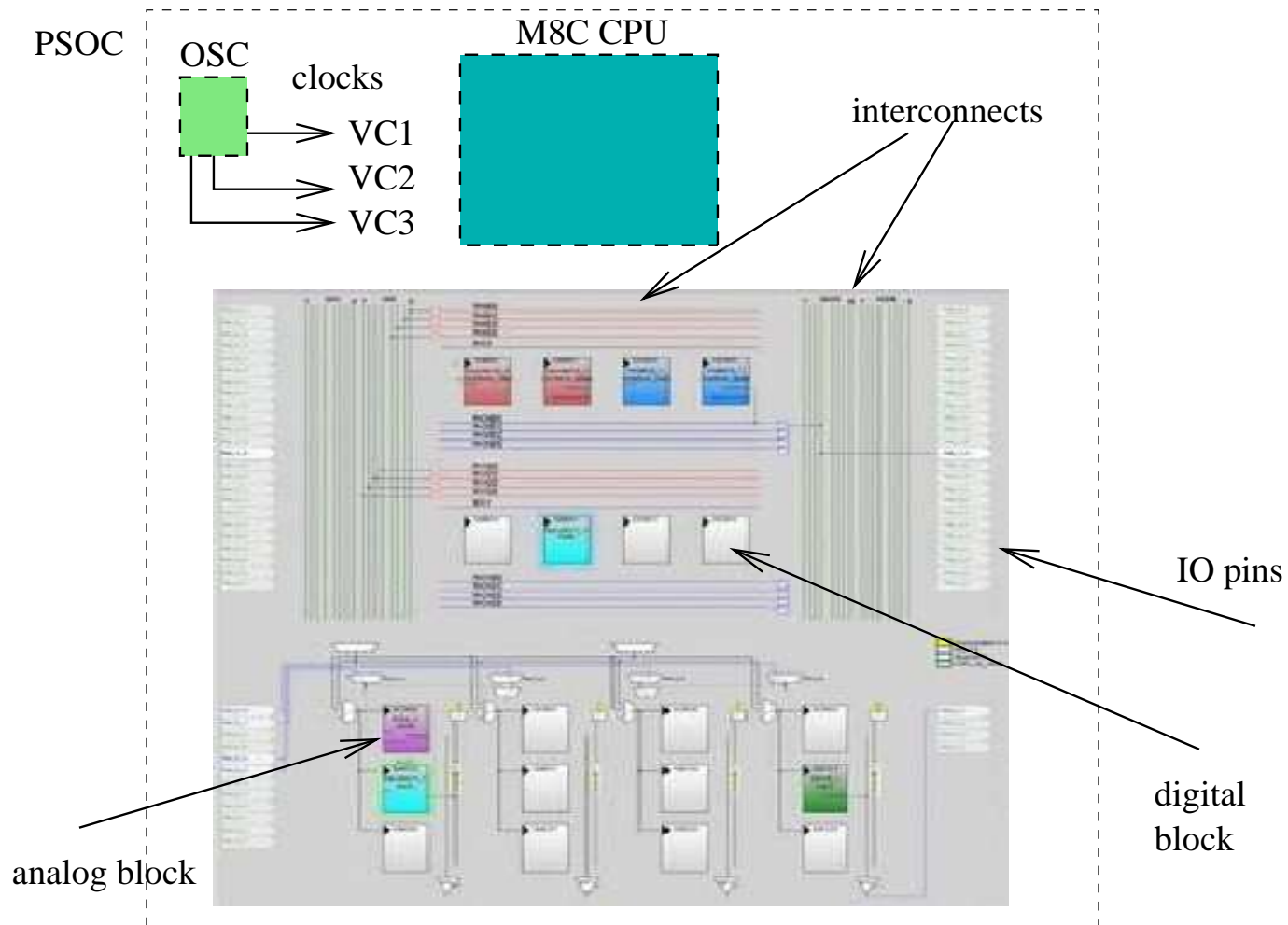
```

```
    }  
}  
  
    my[40+count+1+(count!=0)]='\0';  
    my[40+count+2+(count!=0)]='\0';  
  
        if(hEdit!=NULL){  
RECT rcClient;  
GetClientRect(hWnd, &rcClient);  
SetWindowPos(hEdit, NULL, 0, 0, rcClient.right, rcClient.bottom, SWP_NOZORDER);  
    SetWindowText(hEdit, (LPSTR)my);  
    }  
  
}  
  
break;  
  
case WM_DESTROY:  
    KillTimer(hWnd, ID_TIMER);  
PostQuitMessage(0);  
break;  
default:  
return DefWindowProc(hWnd, message, wParam, lParam);  
}  
return 0;  
}
```

# Cypress psoc introduction

## Chip Architecture

- M8c CPU, iopins, interconnect
- Analog blocks: ADC, amplifiers
- Digital blocks: UARTs etc



## Software Tools

- Programmer: download hex file.
- Designer: compile/build the main C program into hex file
- Chipeditor: select analog blocks, digital blocks, **configure** the blocks, and generate all the include headers for the C program.

### Chipeditor tips:

- **Select** the desired analog/digital block.
- **Place** and **Configure** the parameters of the block. Use the interconnect to wire up the circuit.
- Study the C API and sample programs that are provided in the datasheet.
- **Generate** configuration, this will generate the include headers (.h file) for the C program.

## Important Board layout tips:

- Psoc interconnect architecture and pin assignment are not general purpose.
- Be sure to have a complete psoc pinout configuration before board layout.
- Don't layout board until you have a complete psoc design.