DM&P Vortex86 Series (SX/DX/MX)

Software Programming Reference

Version 1.0

2009-12-16

This programming guide is for software programmers to write programs more quick and easy on Vortex86 series (Vortex86SX/DX/MX). They are from our technical support documents and software engineers' experisnce. For more detail, please visit http://www.dmp.com.tw/tech or send e-mail to soc@dmp.com.tw. Please read section 8.5 before sending technical support request.

History

2009-12-16

- Original version.

Table of Contents

1. Vortex86 Series SoC Overview

The Vortex86 System-On-Chip (SoC) family was originally designed to provide product migration path to the existing user of the DMP M6117D SoC, a 40 MHz x86 SoC introduced to the market in the early 1990s that reached end-of-life in 2007.

The Vortex86 SoC Family includes the Vortex86SX, Vortex86DX and upcoming Vortex86MX. All three are 32-bit x86 processors designed for ultra low power consumption. Both the Vortex86SX and Vortex86DX integrate the North & South bridges, LPC, 5 serial ports, USB 2.0 OTG, Ultra-DMA IDE, 10/100M Ethernet, SPI, I2C, PWM, GPIO, and JTAG interface into a single chip design in a tight 27mm x 27mm 581-pin BGA package.

With a core design based on matured x86 CPU architecture, a rich set of integrated I/O, and designed to function in extreme temperatures ranging -40 to +85 , these SoCs provide the ideal hardware platform for designing the next generation of industrial single board computeres, embedded controllers, and other embedded devices.

Hint:

Vortex86SX = Intel 486SX
Vortex86DX = Vortex86SX + FPU
Vortex86MX = Vortex86DX + VGA

1.1. Vortex86SX

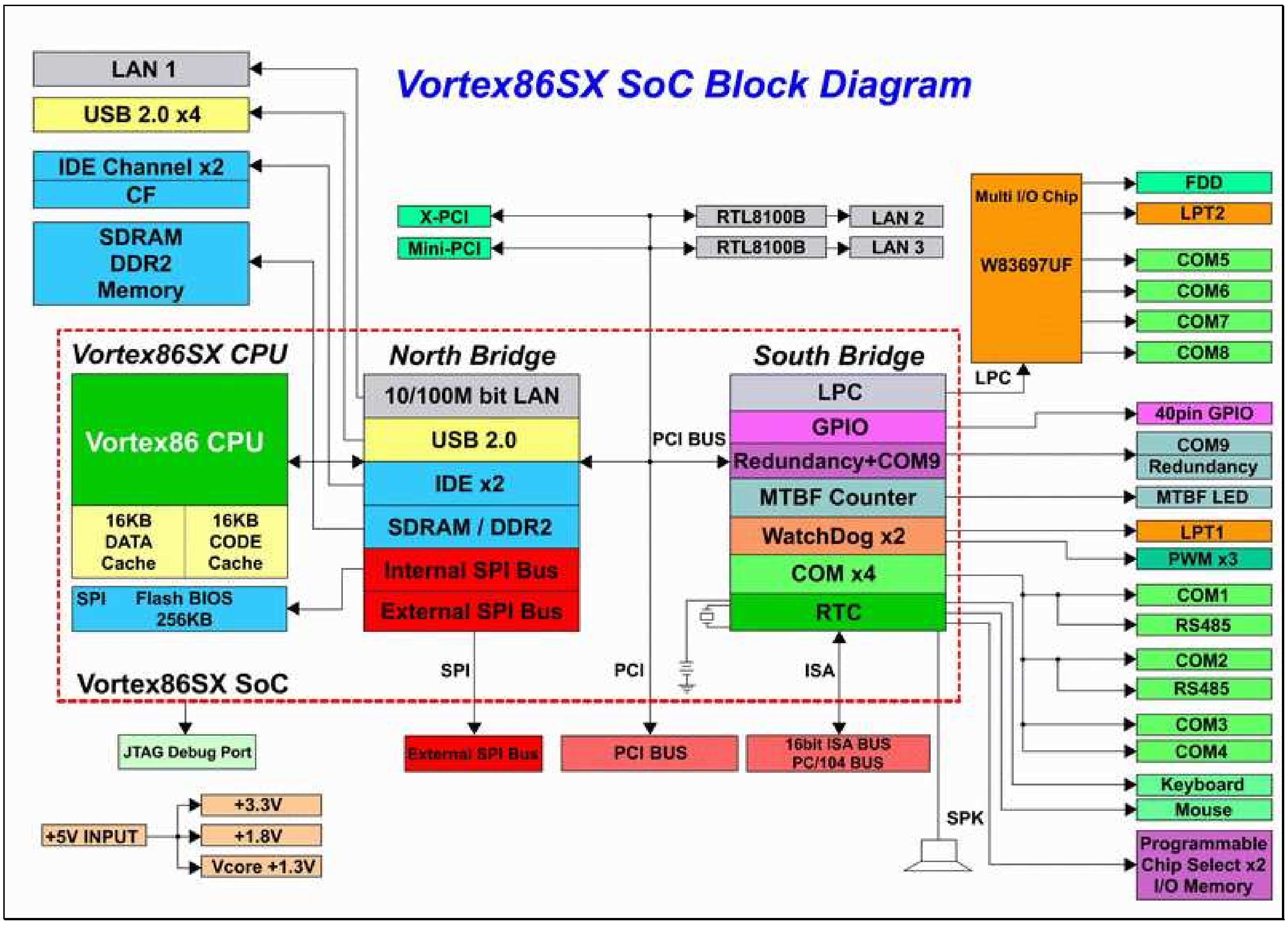
DM&P x86 Semiconductor is proud to provide the Vortex86SX x86 Microprocessor, which is based on MPU structure. It is the x86 SoC (System on Chip) with 0.13 micron process and ultra low power consumption design (less than 1 watt). This comprehensive SoC has been integrated many features, such as various I/O (RS-232, Parallel, USB and GPIO), BIOS, WatchDog Timer, Power Management, MTBF counter, LoC (LAN on Chip),JTAG etc., into a BGA packing single chip.

The Vortex86SX is a high performance, which is compatible with DOS and Linux. It integrates 32KB write through direct map L1 cache, PCI Rev. 2.1 32-bit bus interface at 33 MHz, SDRAM, DDR2, ROM controller, IPC (Internal Peripheral Controllers with DMA and interrupt timer/counter included), Fast Ethernet MAC, FIFO UART, USB2.0 Host and IDE controller into a System-on-Chip (SoC) design.

Furthermore, this outstanding Vortex86SX SoC can not only meet the requirements of embedded applications, such as Electronics Billboard, Firewall Router, Industrial Single-Board-Computers, Receipt Printer Controller, Thin Client PC, Auto Vehicle Locator, Finger Print Identification, Web Camera Thin Server, RS232-to-TCP Transmitter. but also can meet the critical temperature demand, spanning from -40 to +85 degree C.

The Vortex86SX is a high performance and fully static 32-bit X86 processor with the compatibility of Windows based, Linux and most popular 32-bit RTOS. It also integrates 32KB write through direct map L1 cache, PCI rev. 2.1 32-bit bus interface at 33 MHz, SDRAM, DDR2, ROM controller, IPC (Internal Peripheral Controllers with DMA and interrupt timer/counter included), Fast Ethernet MAC, FIFO UART, USB2.0 Host and IDE controller within a single 456-pin BGA package to form a system-on-a-chip (SOC). It provides an ideal solution for the embedded system and communications products (such as thin client, NAT router, home gateway, access point and tablet PC) to bring about desired performance.

1.1.1.System Block Diagram



1.1.2.Feature

- **x86 32bit Processor Core**
  - 6 stage pipe-line
- **Embedded I/D Separated L1 Cache**
  - 16K I-Cache, 16K D-Cache
- **SDRAM/DDRII Control Interface**
  - Support DLL for clock phase auto-adjustion
- **IDE Controller**
  - Support 2 channels Ultra-DMA 100 (Disk x 4)
- **LPC (Low Pin Count) Bus Interface**
  - Support 2 programmable registers to decode LPC address
- **MAC Controller x 1**
- **PCI Control Interface**
  - Up to 3 sets PCI master device
  - 3.3V I/O
- **ISA Bus Interface**
  - AT clock programmable
  - 8/16 Bit ISA device with Zero-Wait-State
  - Generate refresh signals to ISA interface during DRAM refresh cycle
- **DMA Controller**
- **Interrupt Controller**
- **Counter/Timers**
  - 2 sets of 8254 timer controller
  - Timer output is 5V tolerance I/O on 2nd Timer
- **Real Time Clock (Internal Mode or External Mode)**
  - Below 2uA power consumption on Internal Mode (Estimation Value)
- **FIFO UART Port x 5 (5 sets COM Port)**
  - Compatible with 16C550/16C552
  - Default internal pull-up
  - Supports the programmable baud rate generator with the data rate from 50 to 460.8K bps
  - The character options are programmable for 1 start bits; 1, 1.5 or 2 stop bits; even, odd or no parity; 5~8 data bits
  - Support TXD_En Signal on COM1/COM2
  - Port 80h output data could be sent to COM1 by software programming
- **General Chip Selector**
  - 2 sets extended Chip Selector
  - I/O-map or Memory-map could be configurable
  - I/O Addressing: From 2 byte to 64K byte
  - Memory Address: From 512 byte to 4G Byte
- **General Programmable I/O**

- Supports 40 dedicated programmable I/O pins
- Each GPIO pin can be individually configured to be an input/output pin

- **USB 2.0 Host Support**
  - Supports HS, FS and LS
  - 4 ports
- **PS/2 Keyboard and Mouse Interface Support**
  - Compatible with 8042 controller
- **Speaker out**
- **Embedded 256KB Flash**
- **JTAG Interface supported for S.W. debugging**
- **Input clock**
  - 14.318MHz
  - 32.768KHz
- **Output clock**
  - 24 MHz
  - 25 MHz
- **Operating Voltage Range**
  - Core voltage: 1.2 V ~ 1.4V
  - I/O voltage: 1.8V ± 5% , 3.3 V ± 10 %
- **Operating temperature**
  - -40 ~ 85 degree C
- **Package Type**
  - 27x27, 581 ball BGA

1.1.3.Special Feature

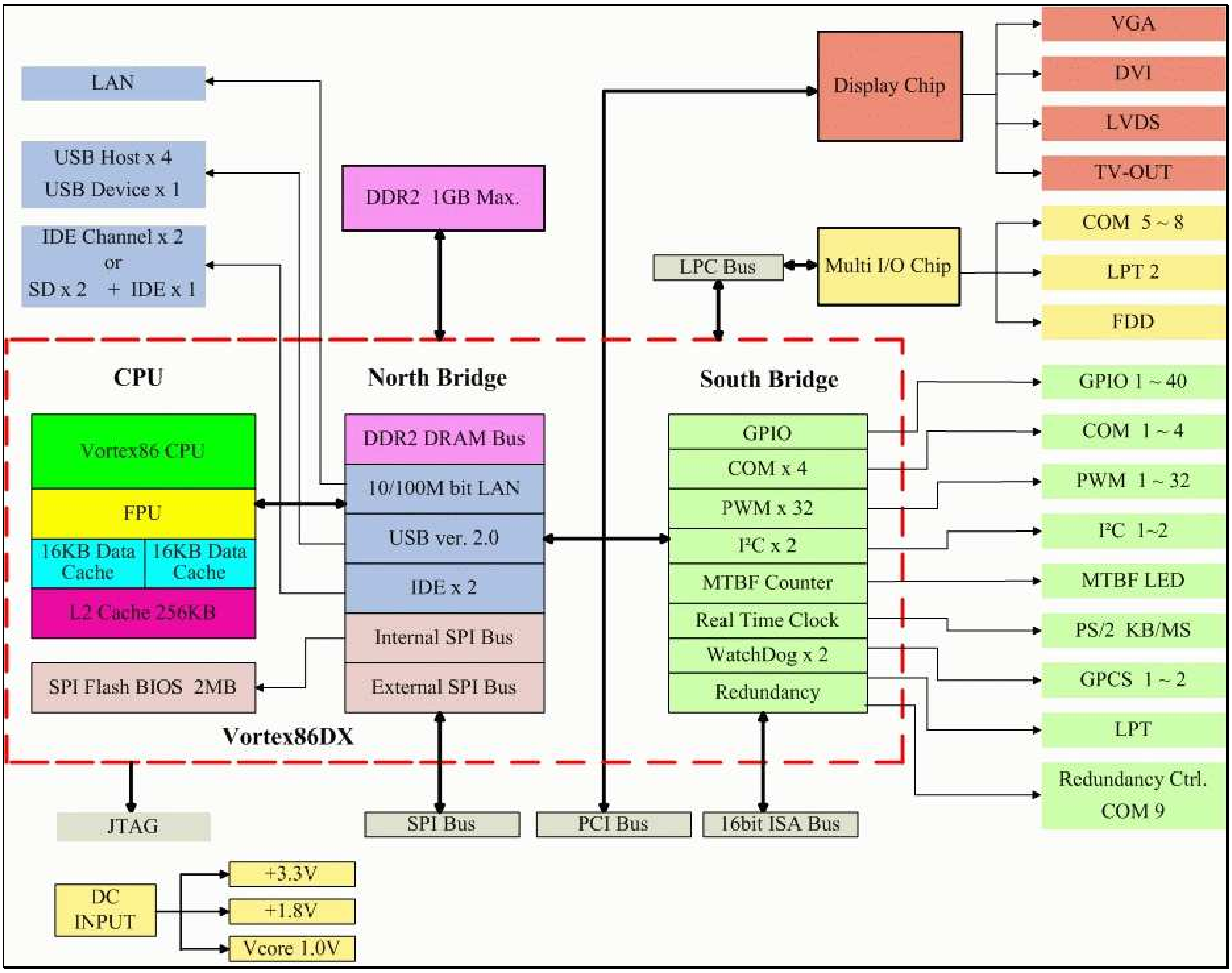
- **JTAG interface (built-in)**
    - 6-pin connector for MICE function
    - Windows version debugging software
    - Specific cable between JTAG connector and printer port (for remote PC)
    - Powerful and low cost tool for development
- **ISOinChip (built-in)**
    - It is a 32 Byte one-time write Flash area
    - Factory will store all necessary data of board (& SoC) in this area for service tracing
    - Necessary data will include:
        - Date code of all major component of the board
        - Model number and serial code of the board
        - Unique serial code of SoC
        - Customer (distributor) code
        - Shipment date, etc.
- **MTBF counter (built-in)**
    - One-time write Flash area for number of MTBF hour before shipment
    - Timer count up by "hour"
    - Built-in TTL alert signal will be enabled after time counter over MTBF number
- **Fault tolerance - Redundancy (built-in)**
    - Two board must stack (plug) in same ISA bus wire
    - Two board will auto be appointed to master and slave mode while every power-on
    - Two board must execute the same software
    - System switch judge by six kind of fail conditions
    - Software library under WinCE, Linux and DOS will provide
    - While system be switched then I/O can also be switched (I/O only include RS232, printer, GPIO, K/B and Mouse)
    - Built-in System Fail Counter

1.2. Vortex86DX

DM&P x86 Semiconductor is proud to provide the Vortex86SX/DX x86 Microprocessor, which is based on MPU structure. It is the x86 SoC (System on Chip) with 90nm process and ultra low power consumption design (less than 1 watt). This comprehensive SoC has been integrated many features, such as various I/O (RS-232, Parallel, USB and GPIO), BIOS, WatchDog Timer, Power Management, MTBF counter, LoC (LAN on Chip),JTAG etc., into a BGA packing single chip.

The Vortex86DX is a high performance and fully static 32-bit X86 processor with the compatibility of Windows based, Linux and most popular 32-bit RTOS. It also integrates 32KB write through 4-way L1 cache, 4-way 256KB L2 cache, PCI rev. 2.1 32-bit bus interface at 33 MHz, DDR2, ROM controller, IPC (Internal Peripheral Controllers with DMA and interrupt timer/counter included), Fast Ethernet, FIFO UART, USB2.0 Host and IDE controller within a single 581-pin BGA package to form a system-on-a-chip (SOC). It provides an ideal solution for the embedded system and communications products (such as thin client, NAT router, home gateway, access point and tablet PC) to bring about desired performance

1.2.1.System Block Diagram



1.2.2.Feature

- **x86 32bit Processor Core**
  - 6 stage pipe-line
- **Floating point unit support**
- **Embedded I/D Separated L1 Cache**
  - 16K I-Cache, 16K D-Cache
- **Embedded L2 Cache**
  - 4-wary 256KB L2 Cache
  - Write through or write back policy
- **DDRII Control Interface**
  - 16 bits data bus
  - DDRII clock support up to 333MHz
  - DDRII size support up to 1G bytes
- **IDE Controller**
  - Support 2 channels Ultra-DMA 100 (Disk x 4)
  - Primary channel support SD card
- **LPC (Low Pin Count) Bus Interface**
  - Support 2 programmable registers to decode LPC address
- **MAC Controller x 1**
- **PCI Control Interface**
  - Up to 3 sets PCI master device
  - 3.3V I/O
- **ISA Bus Interface**
  - AT clock programmable
  - 8/16 Bit ISA device with Zero-Wait-State
  - Generate refresh signals to ISA interface during DRAM refresh cycle
- **DMA Controller**
- **Interrupt Controller**
- **Counter/Timers**
  - 2 sets of 8254 timer controller
  - Timer output is 5V tolerance I/O on 2nd Timer
- **MTBF Counter**
- **Real Time Clock (Internal Mode or External Mode)**
  - Less than 2uA (3.0V) power consumption in Internal RTC Mode while chip is power-off
- **FIFO UART Port x 5 (5 sets COM Port)**
  - Compatible with 16C550/16C552
  - Default internal pull-up
  - Supports the programmable baud rate generator with the data rate from 50 to 460.8K bps
  - The character options are programmable for 1 start bits; 1, 1.5 or 2 stop bits; even, odd or no parity; 5~8 data bits

- Support TXD_En Signal on COM1/COM2
- Port 80h output data could be sent to COM1 by software programming

- **Parallel Port**
  - Support SPP/EPP/ECP mode
- **General Chip Selector**
  - 2 sets extended Chip Selector
  - I/O-map or Memory-map could be configurable
  - I/O Addressing: From 2 byte to 64K byte
  - Memory Address: From 512 byte to 4G Byte
- **General Programmable I/O**
  - Supports 40 dedicated programmable I/O pins
  - Each GPIO pin can be individually configured to be an input/output pin
  - GPIO_P0~GPIO_P3 can be program by 8051
  - GPIO_P0 and GPIO_P1 with interrupt support (input/output)
- **USB 2.0 Host Support**
  - Supports HS, FS and LS
  - 4 ports
- **USB 1.1 Device Support**
  - 1 port
  - Supports FS with 3 programmable endpoint
- **PS/2 Keyboard and Mouse Interface Support**
  - Compatible with 8042 controller
- **Redundant System Support**
- **Speaker out**
- **Embedded 2MB Flash**
  - For BIOS storage
  - The Flash could be disable & use external Flash ROM
- **I²C bus x 2**
  - Compliant w/t V2.1
  - Some master code (general call, START and CBUS) not support
- **Servo Control interface support**
- **General Shift interface support**
- **JTAG Interface supported for S.W. debugging**
- **Input clock**
  - 14.318MHz
  - 32.768KHz
- **Output clock**
  - 24 MHz
  - 25 MHz
  - PCI clock
  - ISA clock
  - DDRII clock

- **Operating Voltage Range**
  - Core voltage: 0.9 V ~ 1.1V
  - I/O voltage: 1.8V ± 5% , 3.3 V ± 10 %
- **Operating temperature**
  - -40 ~ 85 degree C
- **Package Type**
  - 27x27, 581 ball BGA

1.2.3.Special Feature

- **JTAG interface (built-in)**
  - 6-pin connector for MICE function
  - Windows version debugging software
  - Specific cable between JTAG connector and printer port (for remote PC)
  - Powerful and low cost tool for development
- **ISOinChip (built-in)**
  - It is a 32 Byte one-time write Flash area
  - Factory will store all necessary data of board (& SoC) in this area for service tracing
  - Necessary data will include:
    - Date code of all major component of the board
    - Model number and serial code of the board
    - Unique serial code of SoC
    - Customer (distributor) code
    - Shipment date, etc.
- **MTBF counter (built-in)**
  - One-time write Flash area for number of MTBF hour before shipment
  - Timer count up by "hour"
  - Built-in TTL alert signal will be enabled after time counter over MTBF number
- **Fault tolerance - Redundancy (built-in)**
  - Two board must stack (plug) in same ISA bus wire
  - Two board will auto be appointed to master and slave mode while every power-on
  - Two board must execute the same software
  - System switch judge by six kind of fail conditions
  - Software library under WinCE, Linux and DOS will provide
  - While system be switched then I/O can also be switched (I/O only include RS232, printer, GPIO, K/B and Mouse)
  - Built-in System Fail Counter

1.3. Vortex86MX

Not ready.

1.4. Vortex86SX vs. Vortex86DX

To help you choose between the Vortex86SX and Vortex86DX, please refer to the following comparison. However, please note that the SX and DX only have a few pin differences, and both can be used with the same design.

	Vortex86SX	Vortex86DX
Frequency	300MHz	**600MHz ~ 1GHz**
FPU	N/A	**Yes**
L1-Cache	16KB I-Cache 16KB D-Cache	16KB I-Cache 16KB D-Cache
L2-Cache	N/A	**4-wary 256KB L2 Cache**
BIOS Flash	256KB	**2MB**
DRAM Bus	16 Bit DDR2-512MB (max)	16 Bit DDR2-1GB (max)
Watchdog	2	2
PCI Bus	Yes	Yes
ISA Bus	Yes	Yes
LPC Bus	Yes	Yes
USB 2.0	Host x 4	**Host x 4, Client x 1**
RS232C	5 (max)	5 (max)
GPIO	40 bits max.	40 bits max.
LAN MAC+PHY	1	1
IDE	2 Channels	2 Channels / **1 Channel + SD**
Parallel Port	1	1
PWM	N/A	**32**
I²C	N/A	**2**
PS/2 KB/Mouse	Yes	Yes
Power Input	VCore 1.8v, DDR2 1.8v, IO 3.3v	**VCore 1.1v**, DDR2 1.8v, IO 3.3v
Package Size	27 x 27	27 x 27
Package Type	581 Pin BGA	581 Pin BGA

2. Operating System Support

We will introduce I/O access examples in many O/S for programmers. The main O/S supported by us are: DOS, Linux, Windows CE and Windows XP Embedded.

DOS is an old and simple O/S for easy project and test. Most of our driver codes are tested in DOS and then port onto other O/S.

For Linux support, we already integrate driver source code into Linux kernel. Just enable support in kernel to make most peripheral work properly.

Windows CE developers can download BSP from our technical support web site to make their own image. We provide Windows CE 5.0 and 6.0 BSP for Vortex86 series SoC CPU.

Because Windows XP need FPU to run, Vortex86SX can not run XPe. Only Vortex86DX and Vortex86MX can make XP work. Windows XP drivers are also on technical support web site.

For other O/S support, please contact soc@dmp.com.tw.
All drivers can be found on our technical support web site: http://www.dmp.com.tw/tech.

3. DOS

All example codes here are compiled by Turbo/Borland C++. DOS programmers can get Turbo C++ 1.01 free download from http://cc.embarcadero.com/item/26014.

3.1. Access PCI Registers in DOS

We need to access PCI North Bridge or South Bridge to setup registers. DOS programmers can access PCI register via PCI address register CF8h and PCI data register CFCh.

Please refer to **PCI Configuration Registers** section.

PCI North Bridge and South Bridge in Vortex86SX/DX are:
Vortex86DX_NB Function 0 Configuration Space Registers (IDSEL = AD11/Device 0)

- Vendor ID is 17F3H and Device ID is 6021H.

Vortex86DX_SB Configuration Space Registers (IDSEL = AD18/Device 7)

- Vendor ID is 17F3H and Device ID is 6031H.

For Borland C++, below example code can be compiled with check "Options -> Compiler -> Code generation -> Options -> Compile via assembler" (Turbo Assember is needed).

```
  _asm
  {
    mov dx,  0xcf8
    mov eax, 0x80000090
    out dx,  eax
    mov dx,  0xcfc
    in  eax, dx
    mov val, eax
  }
```

Above example is easy to read but some DOS compilers can not process those codes well. We use "__emit__" to insert machine code into code to make 32-bit register access. Here is code in all DOS examples to access PCI registers:

```
#include <dos.h>

// Disable warning message "Parameter xxx is never used
#pragma warn -par

// Read north bridge register
unsigned long Read_nb(unsigned char idx)
{
  unsigned long retval;
  _asm mov dx, 0cf8h
  // mov eax, 80000000h
  __emit__(0x66); __emit__(0xb8);
  __emit__(0x00); __emit__(0x00); __emit__(0x00); __emit__(0x80);
  _asm mov al, idx
  // out edx, eax
  __emit__(0x66); _asm out dx, ax
  _asm mov dx, 0cfch
  // in eax, edx
  __emit__(0x66); _asm in ax, dx
  // mov retval, eax
  __emit__(0x66); _asm mov WORD PTR retval, ax
  return retval;
}

// Write north bridge register
void WriteNorthBridge(unsigned char idx, unsigned long val)
{
  _asm mov dx, 0cf8h
  // mov eax, 80000000h
  __emit__(0x66); __emit__(0xb8);
  __emit__(0x00); __emit__(0x00); __emit__(0x00); __emit__(0x80);
  // out dx, eax
  _asm mov al, idx
  __emit__(0x66); __emit__(0xef);
  _asm mov dx, 0cfch
  // mov eax, val
  __emit__(0x66); __emit__(0x8b); __emit__(0x46); __emit__(0x08);
  // out dx, eax
  __emit__(0x66); __emit__(0xef);
}

// Read south bridge register
unsigned long read_sb(unsigned char idx)
{
  unsigned long retval;
  _asm mov dx, 0cf8h
  // mov eax, 80003800h
  __emit__(0x66); __emit__(0xb8);
  __emit__(0x00); __emit__(0x38); __emit__(0x00); __emit__(0x80);
  _asm mov al, idx
  // out edx, eax
  __emit__(0x66);
  _asm out dx, ax
  _asm mov dx, 0cfch
  // in eax, edx
  __emit__(0x66); _asm in ax, dx
  // mov retval, eax
  __emit__(0x66); _asm mov WORD PTR retval, ax
  return retval;
}

// Write south bridge register
void write_sb(unsigned char idx, unsigned long val)
{
  _asm mov dx, 0cf8h
  // mov eax, 80003800h
  __emit__(0x66); __emit__(0xb8);
  __emit__(0x00); __emit__(0x38); __emit__(0x00); __emit__(0x80);
  _asm mov al, idx
  // out dx, eax
  __emit__(0x66); __emit__(0xef);
  _asm mov dx, 0cfch
  // mov eax, val
  __emit__(0x66); __emit__(0x8b); __emit__(0x46); __emit__(0x08);
  // out dx, eax
  __emit__(0x66); __emit__(0xef);
}
```

Programmers also can use PCI BIOS to access PCI registers. PCI BIOS calls will increase the complexity of examples. We will not use PCI BIOS in all DOS examples. Here is example code to call PCI BIOS to access PCI registers:

```
#include <stdio.h>
#include <conio.h>

#define PCI_BIOS_INTERRUPT          0x1A
#define PCI_BIOS_FUNCTION_ID        0xB1
#define PCI_BIOS_PRESENT            0x01
#define PCI_BIOS_FIND_DEVICE        0x02
#define PCI_BIOS_READ_CONFIG_BYTE   0x08
#define PCI_BIOS_READ_CONFIG_WORD   0x09
#define PCI_BIOS_WRITE_CONFIG_BYTE  0x0B
#define PCI_BIOS_WRITE_CONFIG_WORD  0x0C

char          IsPciBiosPresent();
unsigned char PciBios_FindDevice(unsigned nVenderID, unsigned nDeviceID, int nIndex, unsigned
char *pcBusNum, unsigned char *pcDeviceNum);
unsigned char PciBios_ReadByte (char cBusNum, char cDeviceNum, int nOffset);
unsigned char PciBios_WriteByte(char cBusNum, char cDeviceNum, int nOffset, unsigned char
cValue);

void main()
{
  unsigned char cBusNum, cDeviceNum;
  unsigned char c;

  /* Check PCI BIOS */
  if(!IsPciBiosPresent())
  {
    printf("Unable to find PCI BIOS.\n");
    return;
  }

  /* Find bus and device number */
  PciBios_FindDevice(0x17F3, 0x6021, 0, &cBusNum, &cDeviceNum);

  /* Read A0H in north bridge (Vendor ID: 17F3, Device: 6021).
     Bit 7-3 is reserved and bit 2-0 is CPU speed divided control. */
  c = PciBios_ReadByte(cBusNum, cDeviceNum, 0xA0);
  c &= 0x07;  /* Clear bit 2-0 */

  /* Set clock: bit[2-0]
     000 -> Divide 1
     001 -> Divide 2
     010 -> Divide 3
     011 -> Divide 4
     100 -> Divide 5
     101 -> Divide 6
     110 -> Divide 7
     111 -> Divide 8 */
  c |= 0x01;  /* If CPU is 300MHz, set clock to 150MHz */

  PciBios_WriteByte(cBusNum, cDeviceNum, 0xA0, c);
}

char IsPciBiosPresent()
{
    char cRet;
    asm {
        mov ah, PCI_BIOS_FUNCTION_ID
        mov al, PCI_BIOS_PRESENT
        int PCI_BIOS_INTERRUPT
        mov cRet, ah
    }
    return !cRet;
}

unsigned char PciBios_FindDevice(unsigned nVenderID, unsigned nDeviceID, int nIndex,
                        unsigned char *pcBusNum, unsigned char *pcDeviceNum)
{
    unsigned char cRet, cBus, cDevice;
    asm {
        mov ah, PCI_BIOS_FUNCTION_ID
        mov al, PCI_BIOS_FIND_DEVICE
        mov cx, nDeviceID
        mov dx, nVenderID
        mov si, nIndex
        int PCI_BIOS_INTERRUPT
        mov cRet, ah
        mov cBus, bh
        mov cDevice, bl
    }
    *pcBusNum    = cBus;
    *pcDeviceNum = cDevice;
    return !cRet;
}

unsigned char PciBios_ReadByte(char cBusNum, char cDeviceNum, int nOffset)
{
    unsigned char data, cRet;
    asm{
        mov ah, PCI_BIOS_FUNCTION_ID
        mov al, PCI_BIOS_READ_CONFIG_BYTE
        mov bh, cBusNum
        mov bl, cDeviceNum
        mov di, nOffset
        int PCI_BIOS_INTERRUPT
        mov cRet, ah
        mov data, cl
    }
    if (cRet)
        return -1;
    return data;
}

unsigned char PciBios_WriteByte(char cBusNum, char cDeviceNum, int nOffset, unsigned char
cValue)
{
    unsigned char cRet;
    asm {
        mov ah, PCI_BIOS_FUNCTION_ID
        mov al, PCI_BIOS_WRITE_CONFIG_BYTE
        mov bh, cBusNum
        mov bl, cDeviceNum
        mov di, nOffset
        mov cl, cValue
        int PCI_BIOS_INTERRUPT
        mov cRet, ah
    }
    return !cRet;
}
```

3.2. Check Vortex86SX/DX/MX

Programmers can read PCI register 93H~90H in North Bridge to check Vortex86SX/DX/MX SoC. Here is example to access PCI configuration space registers via CFCH/CF8H port.

DOS Example

```
#include <stdio.h>
#include <dos.h>

// Disable warning message "Parameter xxx is never used
#pragma warn –par

// Read north bridge register
unsigned long read_nb(unsigned char idx)
{
  unsigned long retval;
  _asm mov dx, 0cf8h
  __emit__(0x66); __emit__(0xb8);
  __emit__(0x00); __emit__(0x00); __emit__(0x00); __emit__(0x80);
  _asm mov al, idx
  __emit__(0x66); _asm out dx, ax
  _asm mov dx, 0cfch
  __emit__(0x66); _asm in ax, dx
  __emit__(0x66); _asm mov WORD PTR retval, ax
  return retval;
}

unsigned int IsVortex86SX()
{
  // Vortex86SX: 0x31504d44
  // Vortex86DX: 0x32504d44
  // Vortex86MX: 0x33504d44
  // We check Vortex86SX here
  if(0x31504d44L == read_nb(0x90))
    return 1;
  else
    return 0;
}

int main(int argc, char *argv[])
{
  if(IsVortex86SX())
    printf("Vortex86SX found.");
  else
    printf("\nVortex86SX not found.");

  return 0;
}
```

3.3. Change CPU Speed

Internally the Vortex86SX/DX is using the PLL technology (Phase-Locked Loop), the CPU clock can be adjust by changing the register value of the North Bridge Offset register A0h.

The CPU speed could be divided from 1 to 8 by default CPU clock. For example, if the default CPU clock is 300MHz, and you choice the "CPU speed divide by 5", the CPU speed will be 300 / 5 = 60 MHz. And please be noticed the CPU speed could not lower then PCI speed which is 33MHz.

To change CPU clock, find PCI register A0H in north bridge (Vendor ID: 17F3, Device: 6021). Bit 7-3 is reserved and bit 2-0 is CPU speed divided control:

Bit[2-0]	Description
000	Divide 1
001	Divide 2
010	Divide 3
011	Divide 4
100	Divide 5
101	Divide 6
110	Divide 7
111	Divide 8

For example: if CPU clock is 300MHz and set speed divided control to "001", CPU clock will be 150MHz. Here is assembler example:

```
mov dx,  cf8h ; PCI address port set = north bridge offset register a0h
mov eax, 800000a0h
out dx,  eax

mov dx,  cfch ; PCI data port read / write
in  eax, dx
                   ; if CPU clock is  300MHz
or  eax, 00000001h ; set CPU clock to 150MHz
or  eax, 00000004h ; set CPU clock to  60MHz
out dx,  eax
```

DOS Example

```
#include <stdio.h>
#include <dos.h>

// Disable warning message "Parameter xxx is never used
#pragma warn –par

// Read north bridge register
unsigned long read_nb(unsigned char idx)
```

```
{
  unsigned long retval;
  _asm mov dx, 0cf8h
  __emit__(0x66); __emit__(0xb8);
  __emit__(0x00); __emit__(0x00); __emit__(0x00); __emit__(0x80);
  _asm mov al, idx
  __emit__(0x66); _asm out dx, ax
  _asm mov dx, 0cfch
  __emit__(0x66); _asm in ax, dx
  __emit__(0x66); _asm mov WORD PTR retval, ax
  return retval;
}

// Write north bridge register
void write_nb(unsigned char idx, unsigned long val)
{
  _asm mov dx, 0cf8h
  __emit__(0x66); __emit__(0xb8);
  __emit__(0x00); __emit__(0x00); __emit__(0x00); __emit__(0x80);
  _asm mov al, idx
  __emit__(0x66); __emit__(0xef);
  _asm mov dx, 0cfch
  __emit__(0x66); __emit__(0x8b); __emit__(0x46); __emit__(0x08);
  __emit__(0x66); __emit__(0xef);
}

void main()
{
  unsigned char c;

  /* Read A0H in north bridge (Vendor ID: 17F3, Device: 6021).
     Bit 7-3 is reserved and bit 2-0 is CPU speed divided control. */
  c = read_nb(0xA0);
  c &= 0x07;  /* Clear bit 2-0 */

  /* Set clock: bit[2-0]
     000 -> Divide 1
     001 -> Divide 2
     010 -> Divide 3
     011 -> Divide 4
     100 -> Divide 5
     101 -> Divide 6
     110 -> Divide 7
     111 -> Divide 8 */
  c |= 0x01;  /* If CPU is 300MHz, set clock to 150MHz */

  write_nb(0xA0, c);
}
```

3.4. GPIO

40 GPIO pins are provided by the Vortex86SX/DX for general usage in the system. All GPIO pins are independent and can be configured as inputs our outputs; when configured as outputs, pins have 8 mA drive capability and are unterminated; when configured as inputs, pins are pulled-high with a 75k ohm resistance.

GPIO port 0,1 and 2 are always free for use normally. If your system does not use external RTC and SPI, GPIO port 3 is also free for use. Developers also can disable COM1 to select GPIO port 4. The actual free GPIO pins depend on your system. Please check it before using GPIO.

Setup GPIO Direction

Here is GPIO direction and data registers:

	Port 0	Port 1	Port 2	Port 3	Port 4	Description
Data Register	78H	79H	7AH	7BH	7CH	
Direction Register	98H	99H	9AH	9BH	9CH	0: GPIO pin is input mode 1: GPIO pin is output mode

If send value 0FH to port 98H, it means that GPIO port0 [7-4] are input mode and port[3-0] are output mode.
If send value 00H to port 98H, it means that GPIO port0 [7-0] are input mode.
If send value FFH to port 98H, it means that GPIO port0 [7-0] are output mode.
If send value 03H to port 98H, it means that GPIO port0 [7-2] are input mode and port[1-0] are output mode.

DOS Example

```
#include <dos.h>
#include <stdio.h>

void main(void)
{
  /* set GPIO port0[7-0] as input mode */
  outportb(0x98, 0x00);

  /* read data from GPIO port0 */
  inportb(0x78);

  /* set GPIO port1[7-0] as output mode */
  outportb(0x99, 0xff);

  /* write data to GPIO port1 */
  outportb(0x79, 0x55);

  /* set GPIO port2[7-4] as output and [3-0] as input*/
  outportb(0x9a, 0xf0);

  /* write data to GPIO port2[7-4], the low nibble (0x0a) will be ignored */
  outportb(0x7a, 0x5a);
```

```
   /* read data from port2[3-0] */
   unsigned char c = inportb(0x7a) & 0x0f;

   /*--- if GPIO port3 is free, those codes can work ---*/

   /* set GPIO port3[7-2] as output and [1-0] as input*/
   outportb(0x9b, 0xfc);

   /* write data to GPIO port2[7-2], the bit 1-0 will be ignored */
   outportb(0x7b, 0xa5);

   /* read data from port3[1-0] */
   c = inportb(0x7b) & 0x03;

   /*--- if GPIO port4 is free, those codes can work ---*/

   /* set GPIO port4[7,5,3,1] as output and port4[6,4,2,0] as input*/
   outportb(0x9c, 0xaa);

   /* write data to GPIO port4[7,5,3,1], the bit 6,4,2 and0 will be ignored */
   outportb(0x7c, 0xff);

   /* read data from port4[6,4,2,0] */
   c = inportb(0x7c) & 0xaa;
 }
```

3.5. GPIO with Interrupt

GPIO port 0 & 1 in Vortex86DX support interrupt trigger. Programmers can use interrupt to instead of polling GPIO to save CPU performance. GPIO port0 interrupt registers are at offset DCh~DFh in PCI south bridge and GPIO1 registers are at offiset E0h~E3h.

Here are steps to setup GPIO to trigger interrupt:

1. Configure interrupt mask register to determine which GPI can trigger interrupt individually.
2. Set trigger level (high or low) for each GPI.
3. Set period time that interrupt will be generated while the event loading time of any one of GPI[7-0] is longer than the time parameters.
4. Select IRQ.
5. Set interrupt trigger once or continuously.

Relative registers detail is at **GPIO Interrupt Relative Registers**.

DOS Example

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <time.h>
#include <dos.h>

typedef void  interrupt (far *FNISR) (...);
FNISR         _pfnOldIsr;
unsigned int  _nIrqNo;
unsigned int  _nIntNo;
unsigned int  _nOldImr;
unsigned int  _nOldImr2;

static unsigned long  _lCnt = 0;
void interrupt NewIsr(...)
{
  cprintf("IRQ %d trigger. (%lu)\r", _nIrqNo, _lCnt++);
  outp(0x9f, 0xff);
  if(_nIrqNo > 7)
    outp(0xa0, 0x20); //**
  outp(0x20, 0x20);   // send EOI command
}

// Disable warning message "Parameter xxx is never used
#pragma warn -par

// Write south bridge register
void write_sb(unsigned char idx, unsigned long val)
{
  _asm mov dx, 0cf8h
  __emit__(0x66); __emit__(0xb8);
  __emit__(0x00); __emit__(0x38); __emit__(0x00); __emit__(0x80);
  _asm mov al, idx
  __emit__(0x66); __emit__(0xef);
  _asm mov dx, 0cfch
  __emit__(0x66); __emit__(0x8b); __emit__(0x46); __emit__(0x08);
  __emit__(0x66); __emit__(0xef);
}

int main(int nArgCnt, char *pszArg[])
{
  if(nArgCnt != 2)
  {
    printf("\nUsage: %s <IRQ_NUM> \n", pszArg[0]);
    return 1;
  }

  // Install ISR for IRQ
  _disable();

  _nIrqNo = atoi(pszArg[1]);
  _nIntNo = (_nIrqNo <= 7) ? (_nIrqNo + 8) : (_nIrqNo + 0x70 - 0x08);

  _pfnOldIsr = getvect(_nIntNo);
  setvect(_nIntNo, NewIsr);

  unsigned char byMask = 0x00;
  switch(_nIrqNo)
  {
  case 9:   byMask = 0x01; break;
  case 3:   byMask = 0x02; break;
  case 10:  byMask = 0x03; break;
  case 4:   byMask = 0x04; break;
  case 5:   byMask = 0x05; break;
  case 7:   byMask = 0x06; break;
  case 6:   byMask = 0x07; break;
  case 1:   byMask = 0x08; break;
  case 11:  byMask = 0x09; break;
  case 12:  byMask = 0x0b; break;
  case 14:  byMask = 0x0d; break;
  case 15:  byMask = 0x0f; break;
  }

  if(byMask == 0x00)
  {
    printf("\nError IRQ number.\n");
    return 1;
  }

  printf("Using IRQ %d\n", _nIrqNo);

  // set 8259 interrupt controller
  if(_nIrqNo <= 8)
  {
    _nOldImr = inp(0x21);
    outp(0x21, _nOldImr & (~(1 << _nIrqNo)));
  }
  else
  {
    _nOldImr = inp(0x21);
    outp(0x21, _nOldImr & (~(1 << 2)));
    _nOldImr2 = inp(0xa1);
    outp(0xa1, _nOldImr2 & (~(1 << (_nIrqNo - 8))));
  }

  printf("Test Port0: active low, trigger once\r\n");
  unsigned long val = 0xffff0000L | ((0xA0 | byMask) << 8);
  val = ((val << 8) | 0x0000ffffL) & 0x00ffffffL;
  write_sb(0xdc, val);

  _enable();
  getch();
  _disable();

  write_sb(0xdc, 0x0000ff00L);

  if(_nIrqNo <= 8)
  {
    outp(0x21, _nOldImr);
  }
  else
  {
    outp(0x21, _nOldImr);
    outp(0xa1, _nOldImr2);
  }

  setvect(_nIntNo, _pfnOldIsr);

  return 0;
}
```

3.6. Watchdog Timer

There are two watchdog timers in Vortex86SX/DX CPU. One is compatible with DMP M6117D watchdog timer and the other is new. The M6117D compatible watchdog timer is called WDT0 and new one is called WDT1.

WDT0

To access WDT0 registers, programmers can use index port 22H and data port 23H. The watchdog timer uses 32.768 kHz frequency source to count a 24-bit counter so the time range is from 30.5u sec to 512 sec with resolution 30.5u sec. When timer times out, a system reset, NMI or IRQ may happen to be decided by BIOS programming.

Index Port 37h	
Bit 7	Reserved.
Bit 6	0: Disable WDT0 1: Enable WDT0 (default)
Bit 5-0	Reserved.
Index Port 3Ch	
Bit 7	0: Read only, Watchdog timer time out event does not happen. 1: Read only, Watchdog timer time out event happens.
Bit 6	Write 1 to reset Watchdog timer.
Index Port 38h	
Bit 7-4	0000:Reserved 0101:IRQ7 1011:IRQ15 0001:IRQ3 0110:IRQ9 1100:NMI 0010:IRQ4 0111:IRQ10 1101:System reset 0011:IRQ5 1001:IRQ12 1110:Reserved 0100:IRQ6 1010:IRQ14 1111:Reserved
Bit 3-0	Reserved.

Index 3Bh, 3Ah, 39h: Counter

	3Bh	3Ah	39h
	D7……D0	D7……D0	D7……D0
Counter	Most SBit ………………………………………………………………………………Least SBit		

Here are steps to setup watchdog timer:

1. Set Bit 6 = 0 to disable the timer.
2. Write the desired counter value to 3Bh, 3Ah, 39h.
3. Set Bit 6 = 1 to enable the timer, the counter will begin to count up.
4. When counter reaches the setting value, the time out will generate signal setting by index 38h bit[7:4]
5. BIOS can read index 3Ch Bit 7 to decide whether the Watchdog timeout event will happen or not.

To clear the watchdog timer counter:

1. Set Bit 6 = 0 to disable timer. This will also clear counter at the same time.

WDT1

WDT1 does not use index and data port to access WDT registers. It uses I/O port 68H~6DH. The time resolution of WDT1 is 30.5 u second. Here are registers information:

WDT1 Control Register

Port 68h	
Bit 7	Reserved.
Bit 6	0: Disable watchdog timer. 1: Enable watchdog timer.
Bit 5-0	Reserved.

WDT1 Signal Select Control Register

Port 69h			
Bit 7-4	0000:Reserved	0101:IRQ7	1011:IRQ15
	0001:IRQ3	0110:IRQ9	1100:NMI
	0010:IRQ4	0111:IRQ10	1101:System reset
	0011:IRQ5	1001:IRQ12	1110:Reserved
	0100:IRQ6	1010:IRQ14	1111:Reserved
Bit 3-0	Reserved.		

WDT1 Control 2 Register

Port	**6Ch**	**6Bh**	**6Ah**
	D7……D0	D7……D0	D7……D0
Counter	Most SBit ……………………………………………………………………………………Least SBit		

Resolution is 30.5u second.

WDT1 Status Register

Port 6Dh	
Bit 7	0: WDT1 timeout event does not happen 1: WDT1 timeout event happens (write 1 to clear this flag)
Bit 6-0	Reserved.

WDT1 Reload Register

Port 67h	
Bit 7-0	Write this port to reload WDT1 internal counter. The read data is unknown.

Here are steps to setup WDT1:

1. Write time into register 6Ah-6Ch.
2. Select signal from register 69h.
3. Set register 68h bit 8 to enable WDT1.

To clear the watchdog timer counter:

1. Write any value to register 67H

WDT0 DOS Example

```
#include <stdio.h>
#include <conio.h>

void main()
{
  unsigned char c;
  unsigned int lTime;

  outp(0x22,0x13); // Lock register
  outp(0x23,0xc5); // Unlock config. register

  // 500 mini-second
  lTime = 0x20L * 500L;
  outp(0x22,0x3b);
  outp(0x23,(lTime>>16)&0xff);
  outp(0x22,0x3a);
  outp(0x23,(lTime>> 8)&0xff);
  outp(0x22,0x39);
  outp(0x23,(lTime>> 0)&0xff);

  // Reset system
  outp(0x22,0x38);
  c = inp(0x23);
  c &= 0x0f;
  c |= 0xd0; // Reset system. For example, 0x50 to trigger IRQ7
  outp(0x22,0x38);
  outp(0x23,c);

  // Enable watchdog timer
  outp(0x22,0x37);
  c = inp(0x23);
  c |= 0x40;
  outp(0x22,0x37);
  outp(0x23,c);

  outp(0x22,0x13); // Lock register
  outp(0x23,0x00); // Lock config. register

  printf("Press any key to stop trigger timer.\n");
  while(!kbhit())
  {
    outp(0x22,0x13);  // Unlock register
    outp(0x23,0xc5);
    outp(0x22,0x3c);
    unsigned char c = inp(0x23);
    outp(0x22,0x3c);
    outp(0x23,c|0x40);
    outp(0x22,0x13);  // Lock register
    outp(0x23,0x00);
  }

  printf("System will reboot after 500 milli-seconds.\n");
}
```

WDT1 DOS Example

```
#include <stdio.h>
#include <conio.h>

void main()
{
  unsigned char c;
  unsigned long lTime;

  // 500 mini-second
  lTime = 0x20L * 500L;
  outp(0x6c, (lTime >> 16) & 0xff);
  outp(0x6b, (lTime >>  8) & 0xff);
  outp(0x6a, (lTime >>  0) & 0xff);

  // Reset system. For example, 0x50 to trigger IRQ7
  outp(0x69, 0xd0);

  // Enable watchdog timer
  c = inp(0x68);
  c |= 0x40;
  outp(0x68, c);

  printf("Press any key to stop trigger timer.\n");
  while(!kbhit())
    outp(0x67, 0x00);

  printf("System will reboot after 500 milli-seconds.\n");
}
```

3.7. Watchdog Timer with Interrupt

WDT0 DOS Example with Interrupt

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include <time.h>
#include <stdlib.h>

typedef void  interrupt (far * FNISR) (...);
FNISR         _pfnOldIsr;

unsigned int  _nIrqNo;
unsigned int  _nIntNo;
unsigned int  _nOldImr;
unsigned int  _nOldImr2;

void interrupt NewIsr(...)
{
  cprintf("IRQ%d for WDT0 trigger.\r\n", _nIrqNo);
  outp(0x81, 0xaa);   // Put tag for IRQ

  // Disable WDT0
  outp(0x22, 0x13);   // Lock register
  outp(0x23, 0xc5);   // Unlock config. register
  outp(0x22, 0x37);

  unsigned char c = inp(0x23);
  c &= 0xBF;
  outp(0x22, 0x37);
  outp(0x23, c);
  outp(0x22, 0x13);   // Lock register
  outp(0x23, 0x00);   // Lock config. register
  if(_nIrqNo > 7)
    outp(0xa0, 0x20); //**
  outp(0x20, 0x20);   // send EOI command
}

void ResetWatchdogTimer()
{
  // M6117D mode
  outp(0x22, 0x13);   // Lock register
  outp(0x23, 0xc5);   // Unlock config. register
  outp(0x22, 0x3c);

  unsigned char c = inp(0x23);
  outp(0x22, 0x3c);
  outp(0x23, c | 0x40);

  outp(0x22, 0x13);   // Lock register
  outp(0x23, 0x00);   // Lock config. register

  // Vortex86SX mode
  //outp(0x65, 0x00);
}

int IrqTest(int nIrq, unsigned char cMask)
{
  unsigned char c;
  unsigned long lTime;

  outp(0x22, 0x13);   // Lock register
  outp(0x23, 0xc5);   // Unlock config. register

  // 1 seconds
  lTime = 0x20L * 1000L;
  outp(0x22, 0x3b);
  outp(0x23, (lTime >> 16) & 0xff);
  outp(0x22, 0x3a);
  outp(0x23, (lTime >> 8) & 0xff);
  outp(0x22, 0x39);
  outp(0x23, (lTime >> 0) & 0xff);

  // NMI
  outp(0x22, 0x38);
  c = inp(0x23);
  c &= 0x0f;
  c |= cMask;
  outp(0x22, 0x38);
  outp(0x23, c);

  _nIrqNo = nIrq;
  printf("\nM6117D Mode, IRQ%d Test:\n", _nIrqNo);

  // Install ISR for IRQ
  _disable();

  _nIrqNo = nIrq;
  _nIntNo = (_nIrqNo <= 7) ? (_nIrqNo + 8) : (_nIrqNo + 0x70 - 0x08);

  _pfnOldIsr = getvect(_nIntNo);
  setvect(_nIntNo, NewIsr);

  // set 8259 interrupt controller
  if(_nIrqNo <= 8)
  {
    _nOldImr = inp(0x21);
    outp(0x21, _nOldImr & (~(1 << _nIrqNo)));
  }
  else
  {
    _nOldImr = inp(0x21);
    outp(0x21, _nOldImr & (~(1 << 2)));
    _nOldImr2 = inp(0xa1);
    outp(0xa1, _nOldImr2 & (~(1 << (_nIrqNo - 8))));
  }

  _enable();

  // Enable watchdog timer
  outp(0x22, 0x37);
  c = inp(0x23);
  c |= 0x40;
  outp(0x22, 0x37);
  outp(0x23, c);

  outp(0x22, 0x13);   // Lock register
  outp(0x23, 0x00);   // Lock config. register

  //printf("Reloading watchdog timer within 1 seconds...\n");
  // Set tag for IRQ
  outp(0x81, 0x55);

  clock_t clk = clock();
  while(((clock() - clk) / CLK_TCK) < 1)
    ResetWatchdogTimer();

  delay(200);

  // Is IRQ generated ?
  if(inp(0x81) != 0x55)
  {
    printf("Error: WDT0 time out!\n\a\a");

    if(_nIrqNo <= 8)
      outp(0x21, _nOldImr);
    if(_nIrqNo > 8)
      outp(0xa1, _nOldImr2);

    setvect(_nIntNo, _pfnOldIsr);

    //exit(1);
    return 1;
  }

  printf("IRQ%d will generate after 1 seconds.\n", _nIrqNo);
  delay(1200);

  if(_nIrqNo <= 8)
    outp(0x21, _nOldImr);
  if(_nIrqNo > 8)
    outp(0xa1, _nOldImr2);

  setvect(_nIntNo, _pfnOldIsr);
  if(inp(0x81) == 0x55)
  {
    printf("Error: No IRQ!\n\a\a");

    //exit(1);
    return 1;
  }

  printf("Watchdog timer 0 IRQ%d test OK.\n", _nIrqNo);
  return 0;
}

int main()
{
  printf("\nDM&P Watchdog Timer 0 Test for Vortex86SX (%s %s)\n", __DATE__, __TIME__);

  int nRet = 0;

  nRet += IrqTest(3, 0x10);
  nRet += IrqTest(4, 0x20);
  nRet += IrqTest(5, 0x30);
  nRet += IrqTest(6, 0x40);
  nRet += IrqTest(7, 0x50);
  nRet += IrqTest(9, 0x60);
  nRet += IrqTest(10, 0x70);
  nRet += IrqTest(11, 0x80);
  nRet += IrqTest(12, 0x90);
  nRet += IrqTest(14, 0xa0);
  nRet += IrqTest(15, 0xb0);
  return nRet;
}
```

WDT1 DOS Example with Interrupt

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include <time.h>
#include <stdlib.h>

typedef void  interrupt (far * FNISR) (...);
FNISR         _pfnOldIsr;

unsigned int  _nIrqNo;
unsigned int  _nIntNo;
unsigned int  _nOldImr;
unsigned int  _nOldImr2;

void interrupt NewIsr(...)
{
  cprintf("IRQ%d for WDT0 trigger.\r\n", _nIrqNo);
  outp(0x81, 0xaa);   // Put tag for IRQ

  // Disable WDT1
  outp(0x68, 0x00);

  if(_nIrqNo > 7)
    outp(0xa0, 0x20); //**
  outp(0x20, 0x20);   // send EOI command
}

int IrqTest(int nIrq, unsigned char cMask)
{
  unsigned char c;
  unsigned long lTime;

  // 1 seconds
  lTime = 0x20L * 1000L;
  outp(0x6c, (lTime >> 16) & 0xff);
  outp(0x6b, (lTime >> 8) & 0xff);
  outp(0x6a, (lTime >> 0) & 0xff);

  // Set IRQ
  outp(0x69, cMask);

  _nIrqNo = nIrq;
  printf("\nVortex86SX Mode, IRQ%d Test:\n", _nIrqNo);

  // Install ISR for IRQ
  _disable();

  _nIrqNo = nIrq;
  _nIntNo = (_nIrqNo <= 7) ? (_nIrqNo + 8) : (_nIrqNo + 0x70 - 0x08);

  _pfnOldIsr = getvect(_nIntNo);
  setvect(_nIntNo, NewIsr);

  // set 8259 interrupt controller
  if(_nIrqNo <= 8)
  {
    _nOldImr = inp(0x21);
    outp(0x21, _nOldImr & (~(1 << _nIrqNo)));
  }
  else
  {
    _nOldImr = inp(0x21);
    outp(0x21, _nOldImr & (~(1 << 2)));
    _nOldImr2 = inp(0xa1);
    outp(0xa1, _nOldImr2 & (~(1 << (_nIrqNo - 8))));
  }

  _enable();

  // Enable watchdog timer
  c = inp(0x68);
  c |= 0x40;
  outp(0x68, c);

  // Set tag for IRQ
  outp(0x81, 0x55);

  clock_t clk = clock();
  while(((clock() - clk) / CLK_TCK) < 1)
    outp(0x67, 0x00);

  delay(200);

  // Is IRQ generated ?
  if(inp(0x81) != 0x55)
  {
    printf("Error: WDT1 time out!\n\a\a");

    if(_nIrqNo <= 8)
      outp(0x21, _nOldImr);
    if(_nIrqNo > 8)
      outp(0xa1, _nOldImr2);

    setvect(_nIntNo, _pfnOldIsr);

    //exit(1);
    return 1;
  }

  printf("IRQ%d will generate after 1 seconds.\n", _nIrqNo);
  delay(1200);

  if(_nIrqNo <= 8)
    outp(0x21, _nOldImr);
  if(_nIrqNo > 8)
    outp(0xa1, _nOldImr2);

  setvect(_nIntNo, _pfnOldIsr);
  if(inp(0x81) == 0x55)
  {
    printf("Error: No IRQ!\n\a\a");

    //exit(1);
    return 1;
  }

  printf("Watchdog timer 1 IRQ%d test OK.\n", _nIrqNo);
  return 0;
```

```
}

int main()
{
  printf("\nDM&P Watchdog Timer 1 Test for Vortex86SX (%s %s)\n", __DATE__, __TIME__);

  int nRet = 0;

  nRet += IrqTest(3, 0x10);
  nRet += IrqTest(4, 0x20);
  nRet += IrqTest(5, 0x30);
  nRet += IrqTest(6, 0x40);
  nRet += IrqTest(7, 0x50);
  nRet += IrqTest(9, 0x60);
  nRet += IrqTest(10, 0x70);
  nRet += IrqTest(11, 0x80);
  nRet += IrqTest(12, 0x90);
  nRet += IrqTest(14, 0xa0);
  nRet += IrqTest(15, 0xb0);

  return nRet;
}
```

3.8. ISO-in-Chip

It is a 32 Byte one-time write Flash area in Vortex86SX/DX/MX. Factory will store all necessary data of board (& SoC) in this area for service tracing. Necessary data will include:

- Date code of all major component of the board
- Model number and serial code of the board
- Unique serial code of SoC
- Customer (distributor) code
- Shipment date, etc.

The ISOinChip data will be stored at offset EFH~D0H of PCI North Bridge (Vender ID: 17F3H, Device ID: 6021H).

DOS Example

```
#include <stdio.h>
#include <conio.h>

typedef unsigned short WORD;
typedef unsigned long DWORD;

// Read north bridge register
unsigned long read_nb(unsigned char idx);

// Change DWORD to byte array
void ChangeByteOrder(DWORD dw, unsigned char *p);

// Print ISOinChip data
void DumpData(char *str, unsigned char pcBuf[], int nStart, int nEnd, int nFormat);

int main()
{
  // ISOinChip array
  unsigned char pcISOinChip[32] = { 0 };

  for(int i = 0; i < 8; i++)
  {
    // Read NB ISOinChip data.
    ChangeByteOrder(read_nb(i*4+0xD0), pcISOinChip + i * sizeof(DWORD));
  }

  // Print CPU ID data at offset 00h-05h
  DumpData("CPU ID", pcISOinChip, 0, 5, 1);

  // Print product name data at offset 06h-0Dh
  DumpData("Product Name", pcISOinChip, 6, 13, 0);

  // Print PCB version at offset 0Eh-13h
  DumpData("PCB Version", pcISOinChip, 14, 19, 0);

  // Print export week at offset 14h-17h
  DumpData("Export Week", pcISOinChip, 20, 23, 0);

  // Print user ID at offset 18h-1Fh
  DumpData("USER ID", pcISOinChip, 24, 31, 1);

  printf("\n\n");
  return 0;
}

// Print ISOinChip data
//  *str     data title
//  pcBuf[]  ISOinChip data array
//  nStart   start offset
//  nEnd     end offset
//  nFormat  format flag
void DumpData(char *str, unsigned char pcBuf[], int nStart, int nEnd, int nFormat)
{
  printf("\n %s = ", str);

  for(; nStart <= nEnd; nStart++)
  {
    if(nFormat)
      printf("%02x", pcBuf[nStart]);
    else
      printf("%c", pcBuf[nStart]);
  }
}

// Change DWORD to byte array
void ChangeByteOrder(DWORD dw, unsigned char *p)
{
  p[0] = dw >> 0;
  p[1] = dw >> 8;
  p[2] = dw >> 16;
  p[3] = dw >> 24;
}

#pragma warn –par
// Read north bridge register
unsigned long read_nb(unsigned char idx)
{
  unsigned long retval;
  _asm mov dx, 0cf8h
  __emit__(0x66); __emit__(0xb8);
  __emit__(0x00); __emit__(0x00); __emit__(0x00); __emit__(0x80);
  _asm mov al, idx
  __emit__(0x66); _asm out dx, ax
  _asm mov dx, 0cfch
  __emit__(0x66); _asm in ax, dx
  __emit__(0x66); _asm mov WORD PTR retval, ax
  return retval;
}
```

3.9. PWM

There are 3 PWM (8254 counter) in Vortex86SX/DX and they share the same pins of COM2. This document will show programmers to program 8254 to make PWM work.

COM2 and PWM share the same pins in Vortex86SX. COM 2 is enabled by default and PWM is disabled by default. In order to enable PWM, we have to disable COM2 and enable PWM. We have to set bit 2 in Internal Peripheral Feature Control Register (C3H~C0H) in south bridge (vendor ID=17F3H, device ID=6031H).

Register Offset: C3-C0h
Register Name: Internal Peripheral Feature Control Register
Reset Value: 032C0500h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Rsvd		IO16W			IO8W			MEM16W			MEM8W			ISACLK	Int_zw	Rsvd	IDEIS	PWM2	PWM1	PWM0	FCDC	MCLK	EMIQ	PINS6	SFCE	PINS5	PINS4	CPS	PINS2	PINS1	PINS0

Also set bit 13~11 to use internal clock:

Bit	Attrib	Description
13	R/W	PWM Timer2 0: internal 1.19MHz (default) 1: external clock
12	R/W	PWM Timer1 0: internal 1.19MHz (default) 1: external clock
11	R/W	PWM Timer0 0: internal 1.19MHz (default) 1: external clock
2	R/W	PINs selection for COM2 and PWM 0: 9 PINS for COM2 (default) 1: 9 PINS for PWM

PWM (8254) control register address:

IO Address	Description
48H	PWM Timer/Counter 0 Count Register
49H	PWM Timer/Counter 1 Count Register
4AH	PWM Timer/Counter 2 Count Register
4BH	PWM Timer/Counter Control Register

Here is the PWM pin assignment on COM2 when PWM is enabled:

COM2 Pin	PWM Pin
1 (DCD2)	PWM0_CLK
3 (TXD2)	PWM0_OUT
6 (DSR2)	PWM0_GATE
9 (RI2)	PWM1_CLK
7 (RTS)	PWM1_OUT
8 (CTS2)	PWM1_GATE
2 (RXD2)	PWM2_CLK
4 (DTR)	PWM2_OUT
10 (TXDEN2)	PWM2_GATE

Other operations are the same as 8254. Programmers can search 8254 datasheet it from Google to get more information: http://www.google.com/search?q=8254+datasheet.

DOS Example

```
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>

// Disable warning message "Parameter xxx is never used
#pragma warn -par

// Read south bridge register
unsigned long read_sb(unsigned char idx)
{
  unsigned long retval;
  _asm mov dx, 0cf8h
  __emit__(0x66); __emit__(0xb8);
  __emit__(0x00); __emit__(0x38); __emit__(0x00); __emit__(0x80);
  _asm mov al, idx
  __emit__(0x66);
  _asm out dx, ax
  _asm mov dx, 0cfch
  __emit__(0x66); _asm in ax, dx
  __emit__(0x66); _asm mov WORD PTR retval, ax
  return retval;
}

// Write south bridge register
void write_sb(unsigned char idx, unsigned long val)
{
  _asm mov dx, 0cf8h
  __emit__(0x66); __emit__(0xb8);
  __emit__(0x00); __emit__(0x38); __emit__(0x00); __emit__(0x80);
  _asm mov al, idx
  __emit__(0x66); __emit__(0xef);
  _asm mov dx, 0cfch
  __emit__(0x66); __emit__(0x8b); __emit__(0x46); __emit__(0x08);
  __emit__(0x66); __emit__(0xef);
}

typedef unsigned char BYTE;
typedef unsigned int  WORD;
typedef unsigned long DWORD;

int main(int nArgCnt, char *ppArg[])
{
  printf("\nDM&P Vortex86SX/DX PWM Demo Program (%s %s)\n\n", __DATE__, __TIME__);

  if(nArgCnt != 4)
  {
    printf("Usage: %s <C?> <M?> <?>\n\n", ppArg[0]);
    printf("C Counter -> 0 ~ 2\n");
    printf("M Mode    -> 0 ~ 5\n");
    printf("? Value   -> 0 ~ 2^16\n");
    printf("\n");
    printf("Ex: %s C0 M1 100   -> Set counter 0 to mode 1 and value = 100\n", ppArg[0]);
    printf("    %s C1 M5 10000 -> Set counter 1 to mode 5 and value = 10,000\n\n", ppArg[0]);
    return 1;
  }
  long  tmp = read_sb(0xc0);

  // Clear bit 13-11 to use internal clock
  tmp &= 0xffffffc7ffL;

  // Switch COM2 to PWM
  tmp |= 0x0000000004L;

  write_sb(0xc0, tmp);

  unsigned char cCmd   = 0x00;
  long          lValue = 0x00;
  unsigned char cCnt   = 0x00;
  unsigned char cMode  = 0x00;

  for(int i = 1; i < nArgCnt; i++)
  {
    // Handle counter parameter
    if(ppArg[i][0] == 'C' || ppArg[i][0] == 'c')
    {
      unsigned char n = ppArg[i][1] - '0';
      if(n > 2)
      {
        printf("Counter parameter error\n");
        return 1;
      }
      cCmd |= (n << 6);
      cCnt = n;
    }

    // Handle mode parameter
    else if(ppArg[i][0] == 'M' || ppArg[i][0] == 'm')
    {
      unsigned char n = ppArg[i][1] - '0';
      if(n > 5)
      {
        printf("Mode parameter error\n");
        return 1;
      }
      cCmd |= (n << 1);
      cMode = n;
    }

    // Read value
    else
    {
      lValue = atol(ppArg[i]);
    }
  }

  // Assume value is 16-bits
  cCmd |= (0x03 << 4);

  outp(0x4b, cCmd);
  outp(0x48 + cCnt, (unsigned char)(lValue & 0x00ff));
  outp(0x48 + cCnt, (unsigned char)(lValue >> 8));

  printf("\nPWM Counter%d, Mode%d, Value=%lu\n", cCnt, cMode, lValue);

  return 0;
}
```

3.10. Servo

There are 32 SERVO controls in Vortex86DX for some PWM purpose (ex: robotic). This document will show programmers relative registers in Vortex86DX to control PWM output.

The GPIO port 4 for SERVO is working as COM1 by default. Programmers have to disable COM1 to switch pins for PWM output. Before accessing SERVO control registers, programmers have to assign base address in bit 15-9 of D3H~D0H in PCI South Bridge.

SERVO control also can use hardware interrupt as notify when PWM output is finish. 32 SERVO controls will share the same IRQ. SERVO Interrupt Mask Register is used to determine which SERVO control can cause interrupt, and programmers check SERVO Interrupt Status Register to know which SERVO control launch the interrupt. SERVO Sync register is used for synchronization. Before accessing SERVO registers, get SERVO base address from D3H~D0H in PCI South Bridge.

There are three methods to check when the PWM output is finished:

1. use interrupt;
2. by polling the SERVO Interrupt Status Register;
3. by polling the RC field in SERVO Control Register.

Register Offset: D3h – D0h

Register Name: Internal SERVO Control Register

Reset Value: 00000000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved							CLKS	UE	Rsvd			SIRT				UIOA							Reserved								

Bit	Name	Attribute	Description
31-25	Rsvd	RO	Reserved
24	CLKS	R/W	Servo Clock selection 0: 10MHz (default) 1: 50MHz
23	UE	R/W	Enable/Disable Internal SERVO IO Address Decode 0: Disable (Default) 1: Enable
22-20	Rsvd	RO	Reserved
19-16	SIRT	R/W	SERVO IRQ Routing Table Bit19 Bit18 Bit17 Bit16 Routing Table 0 0 0 0 Disable.

Bit	Name	Attribute	Description
			0 0 0 1 IRQ[9] 0 0 1 0 IRQ[3] 0 0 1 1 IRQ[10] 0 1 0 0 IRQ[4] 0 1 0 1 IRQ[5] 0 1 1 0 IRQ[7] 0 1 1 1 IRQ[6] 1 0 0 0 IRQ[1] 1 0 0 1 IRQ[11] 1 0 1 0 Reserved 1 0 1 1 IRQ[12] 1 1 0 0 Reserved 1 1 0 1 IRQ[14] 1 1 1 0 Reserved 1 1 1 1 IRQ[15] These four bits are used to route SERVO IRQ to any 8259 Interrupt lines. The BIOS should be used to inhibit the setting of the reserved value.
15-9	UIOA	R/W	Internal SERVO IO Address. The Bit[15:9] contain the base IO address A[15:9] of internal SERVO.
8-0	Rsvd	RO	Reserved. All are ‘0’s. Writing any value to these bits causes no effect.

SERVO registers and control registers in PCI South Bridge detail is at **Servo Registers**.

DOS Example

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

typedef unsigned char byte;
typedef unsigned int  word;
typedef unsigned long dword;

#pragma warn - par

// Read south bridge register
unsigned long read_sb(unsigned char idx)
{
  unsigned long retval;
  _asm mov dx, 0cf8h
  __emit__(0x66); __emit__(0xb8);
  __emit__(0x00); __emit__(0x38); __emit__(0x00); __emit__(0x80);
  _asm mov al, idx
  __emit__(0x66);
  _asm out dx, ax
  _asm mov dx, 0cfch
  __emit__(0x66); _asm in ax, dx
  __emit__(0x66); _asm mov WORD PTR retval, ax
  return retval;
}

// Write south bridge register
void write_sb(unsigned char idx, unsigned long val)
{
  _asm mov dx, 0cf8h
  __emit__(0x66); __emit__(0xb8);
  __emit__(0x00); __emit__(0x38); __emit__(0x00); __emit__(0x80);
  _asm mov al, idx
  __emit__(0x66); __emit__(0xef);
  _asm mov dx, 0cfch
  __emit__(0x66); __emit__(0x8b); __emit__(0x46); __emit__(0x08);
  __emit__(0x66); __emit__(0xef);
}

void outpdw(unsigned addr, unsigned long val)
{
  _asm mov dx, WORD PTR addr
  __emit__(0x66);
  _asm mov ax, WORD PTR val
  __emit__(0x66);
  _asm out dx, ax
}

static unsigned int   base_address = 0xfe00;
static unsigned long us = 10L;    // assume the Servo Clock (see Internal SERVO Control Register)
is 10MHZ
void set_pluse(int channel, unsigned long period, unsigned long high_pulse_width)
{
  unsigned long low_pulse_width = period - high_pulse_width;

  if((channel < 0) || (channel >= 32) || (period <= high_pulse_width))
  {
    printf("ERROR: invalid pulse setting!");
    return;
  }

  outpdw(base_address + 0x0c + channel * 12, low_pulse_width * us);
  outpdw(base_address + 0x10 + channel * 12, high_pulse_width * us);
}

void main(int argc, char *argv[])
{
  int channel = 0;
  if(argc > 1)
    channel = atoi(argv[1]);

  printf("DM&P Sevro TEST Programmer (for Vortex86DX) Ver."__DATE__ "\n");
  printf("Copyright (C) 2009 by DM&P Group. All rights reserved.\n\n");

  write_sb(0xc8, 0xffffffffL);        // switch GPIO function to PWM output
  write_sb(0xc0, read_sb(0xc0) | 2L); // disable GPIO port 4's COM function
  write_sb(0xd0, 0x00800000L + base_address); // disable IRQ, Address Decode enable, 10Mhz

  // disable interrupt.
  outpdw(base_address, 0x00000000L);

  // lock all PWM channel output
  outpdw(base_address + 8, 0xffffffffL);

  //PWM period = 20ms, high_pulse_width (PWM duty) = 2.2ms
  set_pluse(channel, 20000L, 2200L);
  // enable PWM
  // pulse count mode; send 300 PWM pulses
  outpdw(base_address + 0x14 + channel * 12, 0x80000000L + 300);

  // continue mode; send PWM pulses continuously
  //outpdw(base_address+0x14+channel*12, 0xc0000000L);
  // unlock all PWM channel output
  outpdw(base_address + 8, 0x00000000L);
}
```

3.11. SPI

There two SPI interface in Vortex86SX/DX. Internal one is used by BIOS and external one is used by SPI flash disk (ex: ICOP boards). Below is the access flow for external SPI flash access.

SPI base address is at 43H~40H of PCI North Bridge. Refer to **PCI Configuration Registers** for more. The base address from below register is for internal SPI. For external SPI, add 08h for external SPI. For DM&P Vortex86SX/DX, the SPI base address register will be FC01H. From the register detail, the base address for internal SPI is FC00H and FC08H for external SPI.

Register Offset:	43h – 40h
Register Name:	SPI Base Address Register
Reset Value:	00000000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PBA																												Rsvd		MIO	En

Bit	Name	Attribute	Description
31-4	PBA	R/W	SPI Base Address A[31-4]. Size is 16 bytes. If it is I/O space, it only uses A[15-4]. If it is memory space, it use A[31-4].
3-2	RSVD	RO	Reserved.
1	MIO	R/W	Memory or I/O space select. 1: Memory space 0: I/O space
0	En	R/W	SPI Base address is enabled when set.

Before accessing external SPI, need to switch GPIO port3[3-0] to SPI pins in PCI South Bridge:

Register Offset:	C3-C0h
Register Name:	Internal Peripheral Feature Control Register
Reset Value:	032C0500h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Rsvd		IO16W			IO8W			MEM16W			MEM8W			ISACLK	Int_zw	Rsvd	IDEIS	PWM2	PWM1	PWM0	FCDC	MCLK	EMIQ	PINS6	SFCE	PINS5	PINS4	CPS	PINS2	PINS1	PINS0

Bit	Name	Attribute	Description
0	PINS0	R/W	PINS selection for External SPI and GPIO Port3 [3-0]

Bit	Name	Attribute	Description
			0: 4 pins for GPIO port3 [3-0] (default) 1: 4 pins for external SPI

Here is DOS example code access external SPI flash. We do not introduce flash memory commands. The main purpose in example code is to show programmers how to access SPI interface.

DOS Example

```
#include <dos.h>
#include <stdio.h>

// Disable warning message "Parameter xxx is never used
#pragma warn -par

// Read north bridge register
unsigned long read_nb(unsigned char idx)
{
  unsigned long retval;
  _asm mov dx, 0cf8h
  __emit__(0x66); __emit__(0xb8);
  __emit__(0x00); __emit__(0x00); __emit__(0x00); __emit__(0x80);
  _asm mov al, idx
  __emit__(0x66); _asm out dx, ax
  _asm mov dx, 0cfch
  __emit__(0x66); _asm in ax, dx
  __emit__(0x66); _asm mov WORD PTR retval, ax
  return retval;
}

// Read south bridge register
unsigned long read_sb(unsigned char idx)
{
  unsigned long retval;
  _asm mov dx, 0cf8h
  __emit__(0x66); __emit__(0xb8);
  __emit__(0x00); __emit__(0x38); __emit__(0x00); __emit__(0x80);
  _asm mov al, idx
  __emit__(0x66);
  _asm out dx, ax
  _asm mov dx, 0cfch
  __emit__(0x66); _asm in ax, dx
  __emit__(0x66); _asm mov WORD PTR retval, ax
  return retval;
}

// Write south bridge register
void write_sb(unsigned char idx, unsigned long val)
{
  _asm mov dx, 0cf8h
  __emit__(0x66); __emit__(0xb8);
  __emit__(0x00); __emit__(0x38); __emit__(0x00); __emit__(0x80);
  _asm mov al, idx
  __emit__(0x66); __emit__(0xef);
  _asm mov dx, 0cfch
  __emit__(0x66); __emit__(0x8b); __emit__(0x46); __emit__(0x08);
  __emit__(0x66); __emit__(0xef);
}

int main()
{
  // Get SPI base address
  unsigned long lAddr = read_nb(0x40) & 0x0000FFFFL;

  // Check SPI is enabled or not
  if((lAddr & 0x00000001) == 0)
  {
    printf("SPI is not enabled\n");
    return 1;
  }

  // Show address type
  if(lAddr & 0x00000002L)
    printf("Address is at memory space\n");
  else
    printf("Address is at I/O space\n");

  lAddr &= 0x0000FFFF0L;
  lAddr += 8; // for extern SPI
  printf("Extern SPI base address = %04X\n", lAddr);

  // Switch GPIO to extern SPI
  unsigned long a = read_sb(0xc0);
  write_sb(0xC0, a | 0x00000001L);

  //
  // Progrmmer can access SPI via base address in varaible lAddr
  //

  return 0;
}
```

3.12. DOS SPI Flash Disk Tool

There is an external SPI in Vortex86SX SoC and some of boards will use it to connect SPI flash memory. DM&P provide a DOS tool for external SPI flash memory and make it work as a disk in DOS. This document will show user how to setup and use SPI flash disk with our tool.

SPITool.exe Usage

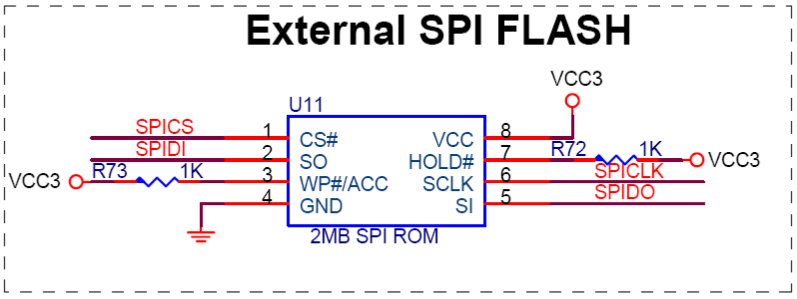
Enable “BIOS -> Boot -> OnBoard Virtual Flash FDD” and use DOS SPI flash disk tool:

```
DM&P Vortex86SX SPI Flashdisk tool Ver.May 29 2008
Copyright (C) 2007 by DM&P Group. All rights reserved.

Functions:
  FORMAT [/y]      -- Initial and format SPI Flashdisk to FAT mode.
  ERASE [/y]       -- Erase SPI Flashdisk. (All data to 0xFFh)
  READ file [/y]   -- Read disk image from SPI Flashdisk.
  WRITE file [/y]  -- Write disk image to SPI Flashdisk.
  PROTECT          -- Protected SPI Flashdisk, write disable.
  UNPROTECT        -- Unprotected SPI Flashdisk, write enable.

Options:
        /y         -- always answer YES to prompt.
```

Some DOS format utility can not format SPI flash disk, user can use “SPITool format” to replace format utility. “Erase” parameter will clean flash memory to 0xff. “Format” parameter will call “erase” function before building FAT 12 file system. “Read” and “write” parameters are use to backup and restore SPI flash disk image.

Disk Protection

See the general circuit of external SPI flash memory connection:



If pin 3 (WP#) in above circuit is pulled high, SPITool can accept “protect” and “unprotect” parameters to set write protection of SPI flash disk. After you setup SPI flash disk and disable changing write protection mode of flash disk, please connect pin 3 to low.

Here is a tip for developers to use write protection function:

1. Remove resistor connected to pin 3 (WP#) as floating. It is R73 in example circuit.

2. Use SPITool to format flash disk and copy files onto it.
3. Make sure everything is fine and run “SPITool protect” to enable write protection function.
4. Reset hardware to make write protection work.

After those steps, SPI flash disk is in write protection mode. If developers want to disable flash disk write protection, connect pin 3 (WP#) to high and run “SPITool unprotect” in DOS.

3.13. Serial Port

The BIOS function calls only provide 9,600 bps. Programmers need to control UART for higher baud rate (ex: 115,200 bps). We also provide DOS real mode serial port library for users.

Please visit http://www.dmp.com.tw/tech/dmp-lib/serport/ for more detail.

3.14. Ethernet

The built-in Ethernet in Vortex86SX is R6040. We provide DSock (DOS real mode TCP/IP socket library) for Vortex86SX to access TCP/IP under DOS. R6040 DOS packet driver is also included into DSock.

DSock provides simple C functions for programmers to write Internet applications. We also provide Internet examples using DSock: BOOTP/DHCP, FTP server, SMTP client/server, HTTP server, TELNET server, Talk client/server, etc. It is free for DM&P products using M6117D/Vortex86/Vortex86SX/Vortex86DX CPU.

Please visit http://www.dmp.com.tw/tech/dmp-lib/dsock/ for more detail.

3.15. Power Consumption Test

This document provides information about Vortex86SX power consumption in DOS, X-Linux and Windows CE for developers. VSX-6127 is used as test platform to measure current in different O/S environment.

HLT puts the processor into a halted state, where it will perform no more operations until restarted by an interrupt or a reset. System timer0 will generate IRQ0 18.2 times every second, so your code will be run after 55ms after HLT instruction. As our test, it will save 50mA on M6117D series. Here is C example code for DOS:

```
#include <conio.h>
void main()
{
  while(!kbhit())
  {
    /* HLT puts the processor into a halted state, where it will perform no
       more operations until restarted by an interrupt or a reset.          */
    asm {
      hlt
    }
    /* System timer0 will generates IRQ0 18.2 times every second, so you can
       see 18 '.' every second. */
    cprintf("."); /* do your job here */
  }
}
```

It is power consumption test on VSX-6127 running DOS:

	VSX-6127 boot from DOM	VSX-6127 boot from USB
DOS	650mA	735mA
DOS + HLT	590mA	695mA

From the table, USB pen driver will use more power than DOM and HLT instruction can save about 50mA in DOS. Linux and Windows CE 6.0 test is added into table to compare with DOS:

	VSX-6127 boot from DOM
DOS	650mA (3.25W)
DOS + HLT	590mA (2.95W)
X-Linux R5.5	580mA (2.90W)
Windows CE 6.0	580mA (2.90W)

Linux and Windows CE will put CPU onto idle status that can save more power than DOS.

Vortex86SX can change CPU frequency by program. Search “Control Vortex86SX/DX CPU Speed“ in our technical support web page (http://www.dmp.com.tw/tech) for more detail. This table shows the power consumption of

VSX-6127 booting from DOS in different CPU frequency:

CPU Freq.	DOS	DOS+HLT	X-Linux
300MHz	650mA	590mA	590mA
150MHz	645mA	580mA	580mA
100MHz	645mA	575mA	575mA
75MHz	640mA	570mA	575mA
60MHz	640mA	570mA	570mA
50MHz	630mA	570mA	570mA
43MHz	630mA	570mA	570mA
37.5MHz	630mA	570mA	565mA

Developers have to slow Vcore voltage to Vortex86SX manually when CPU clock is slow down. Because of no Vcore adjust circuit, slow down CPU frequency can not save more power. If you need lower power consumption in your project, please contact soc@dmp.com.tw for this issue.

For DOS users, using HLT instruction can save 6.6% power consumption. It is no additional operation for Linux and Windows CE users for O/S will put CPU into idle status.

As our test, VSX-6127 has Vortex86SX SoC, 128M DDR2 DRAM, Z9s graphic chipset (using more power than Vortex86SX SoC) and 8 serial ports only need 3.25W~2.9W at idle mode. If Vortex86SX without graphic chipset is used, the power consumption will be lower than 1.5W (300mA*5V). Or, disable USB and serial port in BIOS to save more power if those functions are not needed.

4. Linux

This section has information for Linux developers to make Vortex86SX/DX/MX work properly on Linux. Most drivers are built-in Linux kernel and just enable it in Linux kernel to make it work. We also provide some Linux example codes for programmers to use GPIO, watchdog timer and etc.

We need to access PCI North Bridge or South Bridge to setup registers. Programmers can access PCI register via PCI address register CF8h and PCI data register CFCh.

Please refer to **PCI Configuration Registers** for detail about CFCh/CF8h registers.

PCI North Bridge and South Bridge in Vortex86SX/DX are:

Vortex86DX_NB Function 0 Configuration Space Registers (IDSEL = AD11/Device 0)

- Vendor ID is 17F3H and Device ID is 6021H.

Vortex86DX_SB Configuration Space Registers (IDSEL = AD18/Device 7)

- Vendor ID is 17F3H and Device ID is 6031H.

4.1. X-Linux

Some of our customers need embedded Linux to start their development. There are too much resource about Linux and needs a lot of time to make embedded Linux. We have some projects/products using embedded Linux and our Linux programmers put it on web site. It can save money and development time for our customers about Linux application. X-Linux is maintained and improved since 2002. Bugs are fixed and customers use it as their Linux application without trouble. X-Linux does not provide full documents and friendly tools to install, but it is enough and good for most embedded Linux application.

No Graphic Solution

We will not provide X-Linux with X-Window for our embedded products now. This is because the X-Window is complexity and we reduce function/size to add it into X-Linux will make more problems for users about any X-Window modification. If you need tiny graphic solution in your OEM/ODM project, you can contact your DMP/ICOP sales for customized technical support. For general graphic solution, you can try Puppy Linux (http://www.puppylinux.org/) or other popular Linux (ex: Debina with graphic interface). For that, maybe you need 512MB to 1GB storage.

X-Linux feature list:

- Can run on Vortex86SX/DX series with 64M bytes memory.
- Only need 25M bytes storage space.
- Only need 15 seconds to boot on Vortex86SX series from power on.
- Support MSTI Embeddisk.
- Support EXT3 filesystem.
- Working with read-only filesystem (using tmpfs to reduce writing Flash storage).
- Support serial console for device without VGA.
- Include FTP, TELNET and WWW server.
- Support DHCP.
- Support PPP dial-up (MC35 GPRS modem) and access PPP dial in.
- Support NFS.
- Support SSH.
- Support USB mass storage and keyboard/mouse.
- Support NTP client.

Environment Overview

Software	Version	Path
Linux Kernel	2.6.29	/boot/linux
Boot Loader	SysLinux 2.13	/boot
Shell	BusyBox 1.13.2	/bin/busybox
FTP Server	vsftpd 2.0.3	/usr/sbin/ftpd
TELNET Server	BusyBox 1.13.2	/usr/sbin/telnetd
SSH Server	Dropbear 0.52	/sbin/dropbear
HTTP Server	WN Server 2.4.6	/usr/httpd
Share Library	glibc 2.8.9	/lib
DHCP Client	BusyBox 1.13.2	/sbin/udhcpd
PPP Daemon	pppd 2.4.1	/sbin
NFS	NFS-Utils 1.0.6	/sbin
Web Pages		/usr/www
Size Requirement		**< 15 MB**

For more detail, please visit http://www.dmp.com.tw/tech/os-xlinux/.

4.2. Ethernet Driver

Linux kernel 2.4.27.14 (or newer) and 2.6.28.3 (or newer) supports R6040 Ethernet chipset. Just enable in Linux kernel to make it work. If you are using old Linux kernel and need driver source, please contact soc@dmp.com.tw to get a copy. The old driver only support Linux kernel 2.4.22~27 and 2.6.10~23.

4.3. IDE Driver

Linux kernel 2.4.27.14 (or newer) and 2.6.28.3 (or newer) supports IDE chipset on Vortex86SX and Vortex86DX version A & B. Just enable IT801X in Linux kernel to make it work properly. For Vortex86DX version C/D and Vortex86MX, the IDE is standard one that enables Intel PIIX for new IDE chipset.

4.4. Check Vortex86SX/DX/MX

Programmers can read PCI register 93H~90H in North Bridge to check Vortex86SX/DX/MX SoC. Here is example to access PCI configuration space registers via CFCH/CF8H port.

Linux Example

```
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>
#include <sys/io.h>

#define IO_PERMOFF  0
#define IO_PERMON   3

unsigned int IsVortex86SX()
{
  unsigned long val;
  iopl(IO_PERMON);
  outl(0x80000090, 0xcf8);
  val = inl(0xcfc);
  iopl(IO_PERMOFF);
  // Vortex86SX: 0x31504d44
  // Vortex86DX: 0x32504d44
  // Vortex86MX: 0x33504d44
  // We check Vortex86SX here
  if(val == 0x31504d44)
    return 1;
  else
    return 0;
}

int main(int argc, char *argv[])
{
  if(IsVortex86SX())
    printf("Vortex86SX found.");
  else
    printf("Vortex86SX not found.");

  return 0;
}
```

4.5. Change CPU Speed

Internally the Vortex86SX/DX is using the PLL technology (Phase-Locked Loop), the CPU clock can be adjust by changing the register value of the North Bridge Offset register A0h.

The CPU speed could be divided from 1 to 8 by default CPU clock. For example, if the default CPU clock is 300MHz, and you choice the "CPU speed divide by 5", the CPU speed will be 300 / 5 = 60 MHz. And please be noticed the CPU speed could not lower then PCI speed which is 33MHz.

To change CPU clock, find PCI register A0H in north bridge (Vendor ID: 17F3, Device: 6021). Bit 7-3 is reserved and bit 2-0 is CPU speed divided control:

Bit[2-0]	Description
000	Divide 1
001	Divide 2
010	Divide 3
011	Divide 4
100	Divide 5
101	Divide 6
110	Divide 7
111	Divide 8

For example: if CPU clock is 300MHz and set speed divided control to "001", CPU clock will be 150MHz. Here is assembler example:

```
mov dx,  cf8h ; PCI address port set = north bridge offset register a0h
mov eax, 800000a0h
out dx,  eax

mov dx,  cfch ; PCI data port read / write
in  eax, dx
                   ; if CPU clock is  300MHz
or  eax, 00000001h ; set CPU clock to 150MHz
or  eax, 00000004h ; set CPU clock to  60MHz
out dx,  eax
```

Linux Example

```

#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>
#include <sys/io.h>
#include <string.h>
#include <stdlib.h>

#define IO_PERMOFF  0
```

```
#define IO_PERMON   3

int main(int argc, char *argv[])
{
  iopl(IO_PERMON);
  outl(0x800000a0, 0xcf8);
  unsigned long val = inl(0xcfc) & 0xF8;
  val |= 0x01; /* If CPU is 300MHz, set clock to 150MHz */
  outl(0x800000a0, 0xcf8);
  outb(val, 0xcfc);
  iopl(IO_PERMOFF);

  return 0;
}
```

4.6. GPIO

40 GPIO pins are provided by the Vortex86SX/DX for general usage in the system. All GPIO pins are independent and can be configured as inputs or outputs; when configured as outputs, pins have 8 mA drive capability and are unterminated; when configured as inputs, pins are pulled-high with a 75k ohm resistance.

GPIO port 0,1 and 2 are always free for use normally. If your system does not use external RTC and SPI, GPIO port 3 is also free for use. Developers also can disable COM1 to select GPIO port 4. The actual free GPIO pins depend on your system. Please check it before using GPIO.

Setup GPIO Direction

Here is GPIO direction and data registers:

	Port 0	**Port 1**	**Port 2**	**Port 3**	**Port 4**	**Description**
Data Register	78H	79H	7AH	7BH	7CH	
Direction Register	98H	99H	9AH	9BH	9CH	0: GPIO pin is input mode 1: GPIO pin is output mode

If send value 0FH to port 98H, it means that GPIO port0 [7-4] are input mode and port[3-0] are output mode.
If send value 00H to port 98H, it means that GPIO port0 [7-0] are input mode.
If send value FFH to port 98H, it means that GPIO port0 [7-0] are output mode.
If send value 03H to port 98H, it means that GPIO port0 [7-2] are input mode and port[1-0] are output mode.

Linux Example

```
#include <stdio.h>
#include <stdio.h>
#include <sys/io.h>

#define  outportb(a,b) outb(b,a)
#define  inportb(a)    inb(a)

void main(void)
{
  iopl(3);

  /* set GPIO port0[7-0] as input mode */
  outportb(0x98, 0x00);

  /* read data from GPIO port0 */
  inportb(0x78);

  /* set GPIO port1[7-0] as output mode */
  outportb(0x99, 0xff);

  /* write data to GPIO port1 */
  outportb(0x79, 0x55);
```

```
   /* set GPIO port2[7-4] as output and [3-0] as input*/
   outportb(0x9a, 0xf0);

   /* write data to GPIO port2[7-4], the low nibble (0x0a) will be ignored */
   outportb(0x7a, 0x5a);

   /* read data from port2[3-0] */
   unsigned char c = inportb(0x7a) & 0x0f;

   /*--- if GPIO port3 is free, those codes can work ---*/

   /* set GPIO port3[7-2] as output and [1-0] as input*/
   outportb(0x9b, 0xfc);

   /* write data to GPIO port2[7-2], the bit 1-0 will be ignored */
   outportb(0x7b, 0xa5);

   /* read data from port3[1-0] */
   c = inportb(0x7b) & 0x03;

   /*--- if GPIO port4 is free, those codes can work ---*/

   /* set GPIO port4[7,5,3,1] as output and port4[6,4,2,0] as input*/
   outportb(0x9c, 0xaa);

   /* write data to GPIO port4[7,5,3,1], the bit 6,4,2 and0 will be ignored */
   outportb(0x7c, 0xff);

   /* read data from port4[6,4,2,0] */
   c = inportb(0x7c) & 0xaa;
 }
```

4.7. Watchdog Timer

There are two watchdog timers in Vortex86SX/DX CPU. One is compatible with M6117D watchdog timer and the other is new. The M6117D compatible watchdog timer is called WDT0 and new one is called WDT1.

WDT0

To access WDT0 registers, programmers can use index port 22H and data port 23H. The watchdog timer uses 32.768 kHz frequency source to count a 24-bit counter so the time range is from 30.5u sec to 512 sec with resolution 30.5u sec. When timer times out, a system reset, NMI or IRQ may happen to be decided by BIOS programming.

Index Port 37h	
Bit 7	Reserved.
Bit 6	0: Disable WDT0 1: Enable WDT0 (default)
Bit 5-0	Reserved.
Index Port 3Ch	
Bit 7	0: Read only, Watchdog timer time out event does not happen. 1: Read only, Watchdog timer time out event happens.
Bit 6	Write 1 to reset Watchdog timer.
Index Port 38h	
Bit 7-4	0000:Reserved 0101:IRQ7 1011:IRQ15 0001:IRQ3 0110:IRQ9 1100:NMI 0010:IRQ4 0111:IRQ10 1101:System reset 0011:IRQ5 1001:IRQ12 1110:Reserved 0100:IRQ6 1010:IRQ14 1111:Reserved
Bit 3-0	Reserved.

Index 3Bh, 3Ah, 39h : Counter

	3Bh	3Ah	39h
	D7……D0	D7……D0	D7……D0
Counter	Most SBit …………………………………………………………………………………Least SBit		

Here are steps to setup watchdog timer:

1. Set Bit 6 = 0 to disable the timer.
2. Write the desired counter value to 3Bh, 3Ah, 39h.
3. Set Bit 6 = 1 to enable the timer, the counter will begin to count up.
4. When counter reaches the setting value, the time out will generate signal setting by index 38h bit[7:4]
5. BIOS can read index 3Ch Bit 7 to decide whether the Watchdog timeout event will happen or not.

To clear the watchdog timer counter:

1. Set Bit 6 = 0 to disable timer. This will also clear counter at the same time.

WDT1

WDT1 does not use index and data port to access WDT registers. It uses I/O port 68H~6DH. The time resolution of WDT1 is 30.5 u second. Here are registers information:

WDT1 Control Register

Port 68h	
Bit 7	Reserved.
Bit 6	0: Disable watchdog timer. 1: Enable watchdog timer.
Bit 5-0	Reserved.

WDT1 Signal Select Control Register

Port 69h			
Bit 7-4	0000:Reserved	0101:IRQ7	1011:IRQ15
	0001:IRQ3	0110:IRQ9	1100:NMI
	0010:IRQ4	0111:IRQ10	1101:System reset
	0011:IRQ5	1001:IRQ12	1110:Reserved
	0100:IRQ6	1010:IRQ14	1111:Reserved
Bit 3-0	Reserved.		

WDT1 Control 2 Register

Port	**6Ch**	**6Bh**	**6Ah**
	D7……D0	D7……D0	D7……D0
Counter	Most SBit ……………………………………………………………………………………Least SBit		

Resolution is 30.5u second.

WDT1 Status Register

Port 6Dh	
Bit 7	0: WDT1 timeout event does not happen 1: WDT1 timeout event happens (write 1 to clear this flag)
Bit 6-0	Reserved.

WDT1 Reload Register

Port 67h	
Bit 7-0	Write this port to reload WDT1 internal counter. The read data is unknown.

Here are steps to setup WDT1:

1. Write time into register 6Ah-6Ch.
2. Select signal from register 69h.
3. Set register 68h bit 8 to enable WDT1.

To clear the watchdog timer counter:

1. Write any value to register 67H

WDT0 Linux Example

```
#include <stdio.h>
#include <sys/io.h>
#include <termios.h>
#include <unistd.h>
#include <assert.h>
#include <string.h>
#include <stdlib.h>
#include <sys/ioctl.h>
#include <sys/time.h>
#include <sys/types.h>

#define outp(a, b)  outb(b, a)
#define inp(a)      inb(a)

int kbhit(void)
{
  struct timeval  tv;
  struct termios  old_termios, new_termios;
  int error;
  int count = 0;
  tcgetattr(0, &old_termios);
  new_termios = old_termios;
  new_termios.c_lflag &= ~ICANON;
  new_termios.c_lflag &= ~ECHO;
  new_termios.c_cc[VMIN] = 1;
  new_termios.c_cc[VTIME] = 1;
  error = tcsetattr(0, TCSANOW, &new_termios);
  tv.tv_sec = 0;
  tv.tv_usec = 100;
  select(1, NULL, NULL, NULL, &tv);
  error += ioctl(0, FIONREAD, &count);
  error += tcsetattr(0, TCSANOW, &old_termios);
  return error == 0 ? count : -1;
}

int main(void)
{
  iopl(3);

  outp(0x22, 0x13);   // Lock register
  outp(0x23, 0xc5);   // Unlock config. register

  // 500 mini-second
  unsigned int lTime = 0x20L * 500L;
  outp(0x22, 0x3b);
  outp(0x23, (lTime >> 16) & 0xff);
  outp(0x22, 0x3a);
  outp(0x23, (lTime >> 8) & 0xff);
  outp(0x22, 0x39);
  outp(0x23, (lTime >> 0) & 0xff);

  // Reset system
  outp(0x22, 0x38);
  unsigned char c = inp(0x23);
  c &= 0x0f;
  c |= 0xd0;          // Reset system. For example, 0x50 to trigger IRQ7
  outp(0x22, 0x38);
  outp(0x23, c);

  // Enable watchdog timer
  outp(0x22, 0x37);
  c = inp(0x23);
  c |= 0x40;
  outp(0x22, 0x37);
  outp(0x23, c);
  outp(0x22, 0x13);   // Lock register
  outp(0x23, 0x00);   // Lock config. register
  printf("Press any key to stop trigger timer.\n");
  while(!kbhit())
  {
    outp(0x22, 0x13); // Unlock register
    outp(0x23, 0xc5);
    outp(0x22, 0x3c);

    unsigned char c = inp(0x23);
    outp(0x22, 0x3c);
    outp(0x23, c | 0x40);
    outp(0x22, 0x13); // Lock register
    outp(0x23, 0x00);
  }

  printf("System will reboot after 500 milli-seconds.\n");
  return 0;
}
```

WDT1 Linux Example

```
#include <stdio.h>
#include <sys/io.h>

#include <termios.h>
#include <unistd.h>
#include <assert.h>
#include <string.h>
#include <stdlib.h>
#include <sys/ioctl.h>
#include <sys/time.h>
#include <sys/types.h>

#define outp(a, b)  outb(b, a)
#define inp(a)      inb(a)

int kbhit(void)
{
  struct timeval  tv;
  struct termios  old_termios, new_termios;
  int error;
  int count = 0;
  tcgetattr(0, &old_termios);
  new_termios = old_termios;
  new_termios.c_lflag &= ~ICANON;
  new_termios.c_lflag &= ~ECHO;
  new_termios.c_cc[VMIN] = 1;
  new_termios.c_cc[VTIME] = 1;
  error = tcsetattr(0, TCSANOW, &new_termios);
  tv.tv_sec = 0;
  tv.tv_usec = 100;
  select(1, NULL, NULL, NULL, &tv);
  error += ioctl(0, FIONREAD, &count);
  error += tcsetattr(0, TCSANOW, &old_termios);
  return error == 0 ? count : -1;
}

int main(void)
{
  iopl(3);

  // 500 mini-second
  unsigned long lTime = 0x20L * 500L;
  outp(0x6c, (lTime >> 16) & 0xff);
  outp(0x6b, (lTime >> 8) & 0xff);
  outp(0x6a, (lTime >> 0) & 0xff);

  // Reset system. For example, 0x50 to trigger IRQ7
  outp(0x69, 0xd0);

  // Enable watchdog timer
  unsigned char c = inp(0x68);
  c |= 0x40;
  outp(0x68, c);
  printf("Press any key to stop trigger timer.\n");
  while(!kbhit())
    outp(0x67, 0x00);
  printf("System will reboot after 500 milli-seconds.\n");
  return 0;
}
```

4.8. ISO-in-Chip

It is a 32 Byte one-time write Flash area in Vortex86SX/DX/MX. Factory will store all necessary data of board (& SoC) in this area for service tracing. Necessary data will include:

- Date code of all major component of the board
- Model number and serial code of the board
- Unique serial code of SoC
- Customer (distributor) code
- Shipment date, etc.

The ISOinChip data will be stored at offset EFH~D0H of PCI North Bridge (Vender ID: 17F3H, Device ID: 6021H).

Linux Example

```
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>
#include <sys/io.h>
#include <string.h>
#include <stdlib.h>

#define IO_PERMOFF  0
#define IO_PERMON   3

// Change DWORD to Byte array
void ChangeByteOrder(unsigned long tmp, unsigned char *p);

// Print ISOinChip data
void DumpData(char *s, unsigned char array[], int begin, int end, int format);

int main()
{
  iopl(IO_PERMON);

  // ISOinChip array
  unsigned char ISOinChip[32] = { 0 };

  int i = 0;
  for(; i < 8; i++)
  {
    // Read ISOinChip data in PCI north bridge
    outl(0x800000D0 + i * sizeof(unsigned long), 0xCF8);
    ChangeByteOrder(inl(0xCFC), ISOinChip + i * sizeof(unsigned long));
  }

  iopl(IO_PERMOFF);

  // Print CPU ID data at offset 00h-05h
  DumpData("CPU ID", ISOinChip, 0, 5, 1);

  // Print product name data at offset 06h-0Dh
  DumpData("Product Name", ISOinChip, 6, 13, 0);
  // Print PCB version at offset 0Eh-13h
  DumpData("PCB Version", ISOinChip, 14, 19, 0);

  // Print export week at offset 14h-17h
  DumpData("Export Week", ISOinChip, 20, 23, 0);

  // Print user ID at offset 18h-1Fh
  DumpData("USER ID", ISOinChip, 24, 31, 1);

  printf("\n");
  return 0;
}

// Print ISOinChip data
//  *s       data title
//  array[]  ISOinChip data array
//  begin    start offset
//  end      end offset
//  format   format flag
void DumpData(char *s, unsigned char array[], int begin, int end, int format)
{
  printf("\n %s = ", s);
  for(; begin <= end; begin++)
  {
    if(format)
      printf("%02x", array[begin]);
    else
      printf("%c", array[begin]);
  }
}

// Change DWORD to byte array
void ChangeByteOrder(unsigned long tmp, unsigned char *p)
{
  p[0] = tmp >> 0;
  p[1] = tmp >> 8;
  p[2] = tmp >> 16;
  p[3] = tmp >> 24;
}
```

4.9. PWM

There are 3 PWM (8254 counter) in Vortex86SX/DX and they share the same pins of COM2. This document will show programmers to program 8254 to make PWM work.

COM2 and PWM share the same pins in Vortex86SX. COM 2 is enabled by default and PWM is disabled by default. In order to enable PWM, we have to disable COM2 and enable PWM. We have to set bit 2 in Internal Peripheral Feature Control Register (C3H~C0H) in south bridge (vendor ID=17F3H, device ID=6031H).

Register Offset: C3-C0h

Register Name: Internal Peripheral Feature Control Register

Reset Value: 032C0500h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Rsvd		IO16W			IO8W			MEM16W			MEM8W			ISACLK	Int_zw	Rsvd	IDEIS	PWM2	PWM1	PWM0	FCDC	MCLK	EMIQ	PINS6	SFCE	PINS5	PINS4	CPS	PINS2	PINS1	PINS0

Also set bit 13~11 to use internal clock:

Bit	Attrib	Description
13	R/W	PWM Timer2 0: internal 1.19MHz (default) 1: external clock
12	R/W	PWM Timer1 0: internal 1.19MHz (default) 1: external clock
11	R/W	PWM Timer0 0: internal 1.19MHz (default) 1: external clock
2	R/W	PINs selection for COM2 and PWM 0: 9 PINS for COM2 (default) 1: 9 PINS for PWM

PWM (8254) control register address:

IO Address	Description
48H	PWM Timer/Counter 0 Count Register
49H	PWM Timer/Counter 1 Count Register
4AH	PWM Timer/Counter 2 Count Register
4BH	PWM Timer/Counter Control Register

Here is the PWM pin assignment on COM2 when PWM is enabled:

COM2 Pin	PWM Pin
1 (DCD2)	PWM0_CLK
3 (TXD2)	PWM0_OUT
6 (DSR2)	PWM0_GATE
9 (RI2)	PWM1_CLK
7 (RTS)	PWM1_OUT
8 (CTS2)	PWM1_GATE
2 (RXD2)	PWM2_CLK
4 (DTR)	PWM2_OUT
10 (TXDEN2)	PWM2_GATE

Other operations are the same as 8254. Programmers can search 8254 datasheet it from Google to get more information: http://www.google.com/search?q=8254+datasheet.

Linux Example

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/io.h>

#define IO_PERMON   3
#define IO_PERMOFF  0

typedef unsigned char BYTE;
typedef unsigned int  WORD;
typedef unsigned long DWORD;

int main(int nArgs, char **pArg)
{
  BYTE  cCmd   = 0x00;
  DWORD lValue = 0x00;
  BYTE  cCnt   = 0x00;
  BYTE  cMode  = 0x00;
  int   i;

  printf("\nDM&P Vortex86DX/SX PWM Demo Program (%s %s)\n\n", __DATE__, __TIME__);

  iopl(IO_PERMON);

  if(nArgs != 4)
  {
    printf("Usage: %s <C?> <M?> <?>\n\n",*pArg);
    printf("C Counter -> 0~2\n");
    printf("M Mode    -> 0~5\n");
    printf("? Value   -> 2^16\n\n");
    printf("Ex: %s C0 M1 100 -> Set counter 0 to mode 1 and value = 100\n", *pArg);
    return  1;
  }

  outl(0x800038C0, 0x0CF8);
  DWORD temp = inl(0x0CFC);

  // Clear bit 13-11 to use internal clock
  temp &= 0xffffc7ff;

  // Switch COM2 to PWM
  temp |= 0x0000000004L;
  outl(0x800038C0, 0x0CF8);
  outl(temp, 0x0CFC);

  for(i = 1; i < nArgs; i++)
  {
    // Handle couter parameter
    if(*(*(pArg + i)) == 'C' || *(*(pArg + i)) == 'c')
    {
      unsigned char n = *(*(pArg + i) + 1) - '0';
      if(n > 2)
      {
        printf("Counter parameter error \n");
        return 0;
      }
      cCmd |= (n << 6);
      cCnt = n;
    }

    // Handle mode parameter
    else if(pArg[i][0] == 'M' || pArg[i][0] == 'm')
    {
      unsigned char n = pArg[i][1] - '0';
      if(n > 5)
      {
        printf("Mode parameter error\n");
        return 0;
      }

      cCmd |= (n << 1);
      cMode = n;
    }
    else
    {
      // Read value
      lValue = (DWORD) atol(pArg[i]);
    }
  }

  // Assume value is 16-bits
  cCmd |= (0x03 << 4);
  outb(cCmd, 0x4b);
  outb((unsigned char)(lValue & 0x00ff), 0x48 + cCnt);
  outb((unsigned char)(lValue >> 8), 0x48 + cCnt);

  iopl(IO_PERMOFF);

  printf("\nPWM Couter%d,Mode%d,Value=%lu\n", cCnt, cMode, lValue);

  return 1;
}
```

4.10. Servo

There are 32 SERVO controls in Vortex86DX for some PWM purpose (ex: robotic). This document will show programmers relative registers in Vortex86DX to control PWM output.

The GPIO port 4 for SERVO is working as COM1 by default. Programmers have to disable COM1 to switch pins for PWM output. Before accessing SERVO control registers, programmers have to assign base address in bit 15-9 of D3H~D0H in PCI South Bridge.

SERVO control also can use hardware interrupt as notify when PWM output is finish. 32 SERVO controls will share the same IRQ. SERVO Interrupt Mask Register is used to determine which SERVO control can cause interrupt, and programmers check SERVO Interrupt Status Register to know which SERVO control launch the interrupt. SERVO Sync register is used for synchronization. Before accessing SERVO registers, get SERVO base address from D3H~D0H in PCI South Bridge.

There are three methods to check when the PWM output is finished:

1. use interrupt;
2. by polling the SERVO Interrupt Status Register;
3. by polling the RC field in SERVO Control Register.

Register Offset: D3h – D0h

Register Name: Internal SERVO Control Register

Reset Value: 00000000h

31 30 29 28 27 26 25	24	23	22 21 20	19 18 17 16	15 14 13 12 11 10 9	8 7 6 5 4 3 2 1 0
Reserved	CLKS	UE	Rsvd	SIRT	UIOA	Reserved

Bit	Name	Attribute	Description
31-25	Rsvd	RO	Reserved
24	CLKS	R/W	Servo Clock selection 0: 10MHz (default) 1: 50MHz
23	UE	R/W	Enable/Disable Internal SERVO IO Address Decode 0: Disable (Default) 1: Enable
22-20	Rsvd	RO	Reserved
19-16	SIRT	R/W	SERVO IRQ Routing Table Bit19 Bit18 Bit17 Bit16 Routing Table 0 0 0 0 Disable.

Bit	Name	Attribute	Description
			0 0 0 1 IRQ[9] 0 0 1 0 IRQ[3] 0 0 1 1 IRQ[10] 0 1 0 0 IRQ[4] 0 1 0 1 IRQ[5] 0 1 1 0 IRQ[7] 0 1 1 1 IRQ[6] 1 0 0 0 IRQ[1] 1 0 0 1 IRQ[11] 1 0 1 0 Reserved 1 0 1 1 IRQ[12] 1 1 0 0 Reserved 1 1 0 1 IRQ[14] 1 1 1 0 Reserved 1 1 1 1 IRQ[15] These four bits are used to route SERVO IRQ to any 8259 Interrupt lines. The BIOS should be used to inhibit the setting of the reserved value.
15-9	UIOA	R/W	Internal SERVO IO Address. The Bit[15:9] contain the base IO address A[15:9] of internal SERVO.
8-0	Rsvd	RO	Reserved. All are ‘0’s. Writing any value to these bits causes no effect.

SERVO registers and control registers in PCI South Bridge detail is at **Servo Registers.**

Linux Example

```
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>
#include <sys/io.h>
#include <string.h>
#include <stdlib.h>

#define IO_PERMOFF  0
#define IO_PERMON   3

// read south bridge register
unsigned long read_sb(unsigned char idx)
{
  unsigned long val = 0;
  iopl(IO_PERMON);
  outl(0x80003800 + idx, 0xcf8);
  val = inl(0xcfc);
  iopl(IO_PERMOFF);
  return val;
}

// write south bridge register
void write_sb(unsigned char idx, unsigned long val)
```

```
{
  iopl(IO_PERMON);
  outl(0x80003800 + idx, 0xcf8);
  outl(val, 0xcfc);
  iopl(IO_PERMOFF);
}

void outpdw(unsigned addr, unsigned long val)
{
  iopl(IO_PERMON);
  outl(val, addr);
  iopl(IO_PERMOFF);
}

static unsigned int   base_address = 0xfe00;
static unsigned long  us = 10L;   // assume the Servo Clock (see Internal SERVO Control Register)
is 10MHZ
void set_pluse(int channel, unsigned long period, unsigned long high_pulse_width)
{
  unsigned long low_pulse_width = period - high_pulse_width;

  if((channel < 0) || (channel >= 32) || (period <= high_pulse_width))
  {
    printf("ERROR: invalid pulse setting!");
    return;
  }

  outpdw(base_address + 0x0c + channel * 12, low_pulse_width * us);
  outpdw(base_address + 0x10 + channel * 12, high_pulse_width * us);
}

int main(int argc, char *argv[])
{
  int channel = 0;
  if(argc > 1)
    channel = atoi(argv[1]);

  printf("DM&P Sevro TEST Programmer (for Vortex86DX) Ver."__DATE__ "\n");
  printf("Copyright (C) 2009 by DM&P Group. All rights reserved.\n\n");

  write_sb(0xc8, 0xffffffffL);  // switch GPIO function to PWM output
  write_sb(0xc0, read_sb(0xc0) | 2L);     // disable GPIO port 4's COM function
  write_sb(0xd0, 0x00800000L + base_address); // disable IRQ, Address Decode enable, 10Mhz

  // disable interrupt.
  outpdw(base_address, 0x00000000L);

  // lock all PWM channel output
  outpdw(base_address + 8, 0xffffffffL);

  // PWM period = 20ms, high_pulse_width (PWM duty) = 2.2ms
  set_pluse(channel, 20000L, 2200L);

  // enable PWM
  // pulse count mode; send 300 PWM pulses
  outpdw(base_address + 0x14 + channel * 12, 0x80000000L + 300);

  // continue mode; send PWM pulses continuously
  //outpdw(base_address+0x14+channel*12, 0xc0000000L);

  // unlock all PWM channel output
  outpdw(base_address + 8, 0x00000000L);
  return 0;
}
```

4.11. SPI

There two SPI interface in Vortex86SX/DX. Internal one is used by BIOS and external one is used by SPI flash disk (ex: ICOP boards). Below is the access flow for external SPI flash access.

SPI base address is at 43H~40H of PCI North Bridge. Refer to **PCI Configuration Registers** for more. The base address from below register is for internal SPI. For external SPI, add 08h for external SPI. For DM&P Vortex86SX/DX, the SPI base address register will be FC01H. From the register detail, the base address for internal SPI is FC00H and FC08H for external SPI.

Register Offset: 43h – 40h
Register Name: SPI Base Address Register
Reset Value: 00000000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PBA																												Rsvd		MIO	En

Bit	Name	Attribute	Description
31-4	PBA	R/W	SPI Base Address A[31-4]. Size is 16 bytes. If it is I/O space, it only uses A[15-4]. If it is memory space, it use A[31-4].
3-2	RSVD	RO	Reserved.
1	MIO	R/W	Memory or I/O space select. 1: Memory space 0: I/O space
0	En	R/W	SPI Base address is enabled when set.

Before accessing external SPI, need to switch GPIO port3[3-0] to SPI pins in PCI South Bridge:

Register Offset: C3-C0h
Register Name: Internal Peripheral Feature Control Register
Reset Value: 032C0500h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Rsvd		IO16W			IO8W			MEM16W			MEM8W			ISACLK	Int_zw	Rsvd	IDEIS	PWM2	PWM1	PWM0	FCDC	MCLK	EMIQ	PINS6	SFCE	PINS5	PINS4	CPS	PINS2	PINS1	PINS0

Bit	Name	Attribute	Description
0	PINS0	R/W	PINS selection for External SPI and GPIO Port3 [3-0]

Bit	Name	Attribute	Description
			0: 4 pins for GPIO port3 [3-0] (default) 1: 4 pins for external SPI

Here is example code access external SPI flash. We do not introduce flash memory commands. The main purpose in example code is to show programmers how to access SPI interface.

Linux Example

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/io.h>

#define IO_PERMON   3
#define IO_PERMOFF  0

/* read north bridge register */
unsigned long read_nb(unsigned char idx)
{
  outl(0x80000000 + idx, 0xcf8);
  unsigned long retval = inl(0xcfc);
  return retval;
}

/* read south bridge register */
unsigned long read_sb(unsigned char idx)
{
  outl(0x80003800 + idx, 0xcf8);
  unsigned long retval = inl(0xcfc);
  return retval;
}

/* write south bridge register */
void write_sb(unsigned char idx, unsigned long val)
{
  outl(0x80003800 + idx, 0xcf8);
  outl(val, 0xcfc);
}


int main()
{
  iopl(IO_PERMON);

  /* Get SPI base address */
  unsigned long lAddr = read_nb(0x40) & 0x0000FFFFL;

  /* Check SPI is enabled or not */
  if((lAddr & 0x00000001) == 0)
  {
    printf("SPI is not enabled\n");
    return -1;
  }

  /* Show address type */
  if(lAddr & 0x00000002L)
    printf("Address is at memory space\n");
  else
    printf("Address is at I/O space\n");

  lAddr &= 0x0000FFFF0L;
  lAddr += 8; /* for extern SPI */
  printf("Extern SPI base address = %04X\n", lAddr);

  /* Switch GPIO to extern SPI */
  unsigned long a = read_sb(0xC0);
  write_sb(0xC0, a | 0x00000001L);

  /* Progrmmer can access SPI via base address in varaible lAddr */
  iopl(IO_PERMOFF);
  return 0;
}
```

5. Windows Embedded Compact (Windows CE)

Windows Embedded CE is a componentized, real-time operating system to deliver compelling experiences on a wide range of small footprint consumer and enterprise devices.

With the latest release of Windows Embedded CE 6.0 R3, device manufacturers can use familiar tools and innovative technologies to create devices differentiated by an immersive user interface, a rich browsing experience, and a unique connection to Windows PCs, servers, services, and devices. By building on the high performance and highly reliable Windows Embedded CE platform, device makers can bring their device to market quickly and efficiently.

Visit http://www.microsoft.com/windowsembedded/en-us/products/windowsce/default.mspx form Micrsoft web site to know more.

Get more resource from MSDN at http://msdn.microsoft.com/en-us/windowsembedded/ce/default.aspx.

Visit http://www.dmp.com.tw/tech/vortex86dx/#wince to get Windows CE 5.0/6.0 BSP for Vortex86DX and evaluation images. For Vortex86SX, please visit http://www.dmp.com.tw/tech/vortex86sx/#wince.

All example codes or step are based on Windows Embedded CE 6.0. For Windows CE 5.0, developers need to take care about path in examples.

5.1. BSP

Before getting correct BSP for your board to start Windows CE development, understanding DM&P SoC CPUs is necessary.

DM&P CPU

Before identifying your hardware, knowing CPU difference clearly will be helpful:

CPU	Description
M6117D	It is i386 compatible CPU and can not run Windows CE.
Vortex86	It is Pentium with MMX compatible CPU.
Vega86	It is Pentium with MMX, SSE and 3DNow.
Vortex86SX	It is i486 compatible CPU without FPU.
Vortex86DX	It is improved from Vortex86SX to add FPU. Can run Windows XP or Windows XP Embedded.
Vortex86MX	It is improved from Vortex86DX to built-in graphic chipset.

Select match BSP

This table will show you hardware information and BSP:

Hardware	CPU	BSP
ICOP VSX-61XX	Vortex86SX	http://www.dmp.com.tw/tech/vortex86sx/#wince
ICOP VSX-63XX	Vortex86DX	http://www.dmp.com.tw/tech/vortex86dx/#wince
ICOP-60XX	Vortex86	http://www.dmp.com.tw/tech/os-wince
ICOP-62XX	Vega86	http://www.dmp.com.tw/tech/os-wince
eBox-2300	Vortex86	http://www.dmp.com.tw/tech/os-wince
eBox-2300SX	Vortex86SX	http://www.dmp.com.tw/tech/vortex86sx/#wince
eBox-2300DX	Vortex86DX	http://www.dmp.com.tw/tech/vortex86dx/#wince
eBox-3300/A	Vortex86DX	http://www.dmp.com.tw/tech/vortex86dx/#wince
eBox-38XX	VIA Eden-N Nano	Not recommend for embedded users.
eBox-43XX	VIA Eden ULV	Not recommend for embedded users.
eBox-48XX	VIA Esther	Not recommend for embedded users.

Evalaution Images

On the BSP download page, there are also some evaluation images and SDK for developers who focus on application. Those images can help programmers to run Windows CE to test.

Install QFE

Visit http://msdn.microsoft.com/en-us/embedded/aa731256.aspx to download QFE from Microsoft. Install QFE

"**Cumulative Product Update Rollup Package (through 12/31/2008)**" and "**Windows Embedded CE 6.0 Monthly Updates**" to fix a lot of bugs. If more new QFE is available, please install them.

For developers who install Windows Embedded 6.0 R3, only QFE after 2009/09 is needed. Old QFE are included in R3.

5.2. Using eboot.bin to Boot Windows CE

If deveoper can not use eboot.bin to load Windows CE from Platform Builder in Visual Studio 2005, please check thise:

- Select correct eboot.bin for your board. The Ethernet on Vortex86 and Vega86 is Realtek 81xx. Vortex86SX/DX/MX is using R6040 Ethernet. Find eboot.bin at **\PLATFORM\<BSP_Name>\SRC \BOOTLoader\eboot\bin\eboot.bin**.
- Make sure “Enable eboot space in memory (IMGEBOOT)” in build option of your project is checked.
- Disable firewall in your Windows XP/Vista to make sure BOOTME message from eboot.bin can be received in connectivity setup.
- Debug serial port parameters can be assigned from loadcepc.exe arguments or boot.ini for x86 BIOS loader. You also can get message from serial port (default: 38400/N/8/1) to check eboot.bin running status.

5.3. Using KITL

Some developers can not boot Windows CE image and use a lot of time to guess cause or try error. Windows CE development provides KITL for developers to trace and debug O/S. If your Windows CE can not boot or get black screen, you can try to enable KITL with debug mode to boot CE via eboot.bin. Windows CE will send boot message via Ethernet to debug window in Visual Studio. It can help you to know the boot procedure and status of Windows CE. Here are debug message example:

```
PB Debugger The Kernel Debugger has been disconnected successfully.
PB Debugger The Kernel Debugger is waiting to connect with target.
4294767296 PID:0 TID:2 CEPC Firmware Init
4294767296 PID:0 TID:2 RTC - Status Reg B - 0x02
4294767296 PID:0 TID:2 g_dwCPUFeatures = 00000111
4294767296 PID:0 TID:2 Looking for rom chain
4294767296 PID:0 TID:2 Rom chain NOT found
PB Debugger Kernel debugger connected.
4294767296 PID:0 TID:2 Firmware Init Done.
4294767296 PID:0 TID:2 Setting up softlog at 0x87cfc000 for 0x800 entries
4294767296 PID:0 TID:2 Booting Windows CE version 6.00 for (x86)
4294767296 PID:0 TID:2 &pTOC = 80db9f10, pTOC = 80d60630, pTOC->ulRamFree = 80dc2000, MemForPT = 00043000
4294767296 PID:0 TID:2
Old or invalid version stamp in kernel structures - starting clean!
4294767296 PID:0 TID:2 Configuring: Primary pages: 28392, Secondary pages: 0, Filesystem pages = 14196
4294767296 PID:0 TID:2
Booting kernel with clean memory configuration:
4294767296 PID:0 TID:2 Memory Sections:
4294767296 PID:0 TID:2 [0] : start: 80e06000, extension: 0000e000, length: 06ee8000
4294767296 PID:0 TID:2 X86Init done, OEMAddressTable = 80226d30, RAM mapped = 08000000.
4294767296 PID:0 TID:2 Windows CE KernelInit
. . .
4294821245 PID:400002 TID:650002 This device has booted 2 times !!!
4294822745 PID:3b00002 TID:3b10002 DoImport Failed! Unable to import from Library 'ADVAPI32.dll'
4294822747 PID:3b00002 TID:3b10002 !! Process Import failed - Process 'explorer.exe' not started!!
4294822759 PID:3b70002 TID:3b80002 Initializating services for Services.exe
4294822761 PID:400002 TID:3b80002 DEVICE!RegReadActivationValues RegQueryValueEx(Services\Prefix) returned 2
4294822762 PID:400002 TID:3b80002 DEVICE!RegReadActivationValues RegQueryValueEx(Services\BusPrefix) returned 2
```

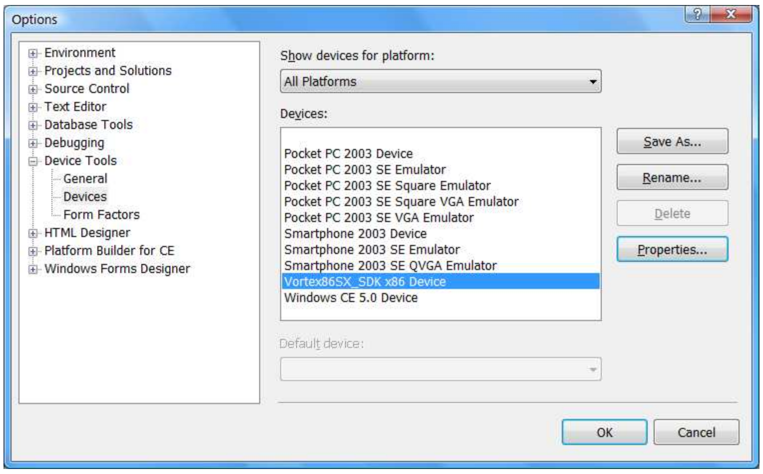
5.4. Uisng CoreCon for Application Debug

Because Visual Stdio 2005 will use ActiveSync to copy required connectivity files to the device, we can use TCP connection to replace ActiveSync if it is not available. We need files at **\Program Files\Common Files\Microsoft Shared\CoreCon\5.01\Target\x86** to make TCP connection. They are:

- **clientshutdown.exe**
- **CMAccept.exe**
- **ConmanClient2.exe**
- **eDbgTL.dll**

Here are steps to use CoreCon connection:

1. Developers can build Windows CE image with those files or use remote file viewer in Platform Builder to upload them.
2. Setup VS2005 for TCP connection from “VS2005 -> Tools -> Options”, then click “Device Tools -> Devices”. Select device you are using.

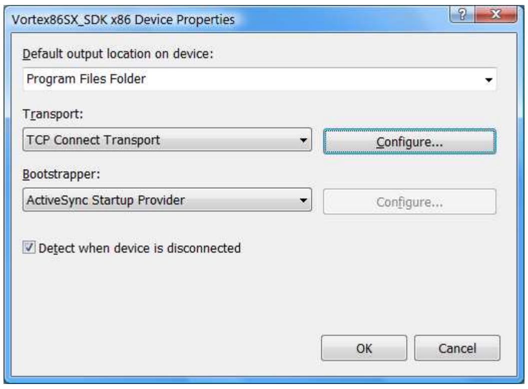


Show devices for platform: All Platforms

Devices:
Pocket PC 2003 Device
Pocket PC 2003 SE Emulator
Pocket PC 2003 SE Square Emulator
Pocket PC 2003 SE Square VGA Emulator
Pocket PC 2003 SE VGA Emulator
Smartphone 2003 Device
Smartphone 2003 SE Emulator
Smartphone 2003 SE QVGA Emulator
Vortex86SX_SDK x86 Device
Windows CE 5.0 Device

Default device:

3. Press “Properties” button and press “Configure” button.



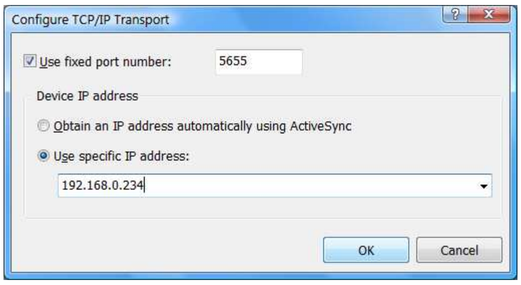
Default output location on device: Program Files Folder

Transport: TCP Connect Transport

Bootstrapper: ActiveSync Startup Provider

☑ Detect when device is disconnected

4. You have to get IP address of your Windows CE device. Run **ipconfig** in Windows CE device or run “**s ipconfig /d**” in Tagret Control window and the output of ipconfig will send output to debug window. Select “Use specific IP address” and fill IP address of your device. Press OK three times to finish setup.



[✓] Use fixed port number: 5655

Device IP address

( ) Obtain an IP address automatically using ActiveSync

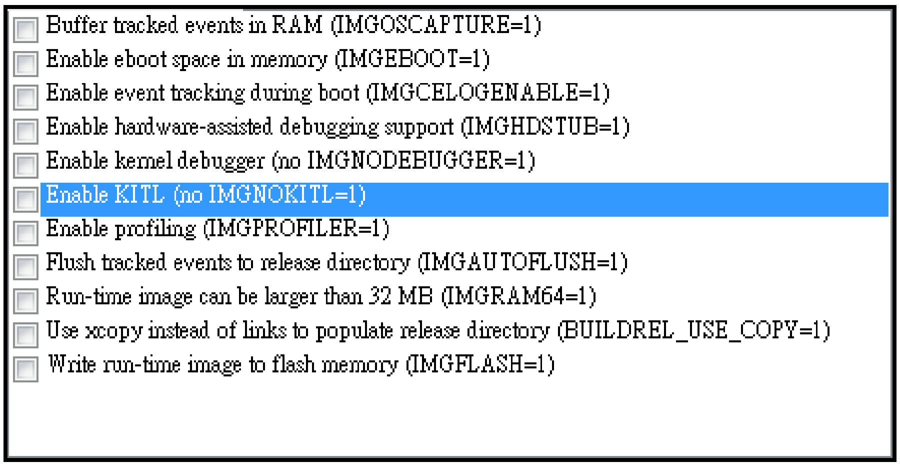
(●) Use specific IP address: 192.168.0.234

5. Select “VS2005 -> Tools -> Connect to Device” and select your Windows CE device. Before pressing “Connect” button, run **CMAccept.exe** and **ConmanClient2.exe** in your Windows CE device. Developers can run those two programs from those ways:
   I. Direct run those two programs on Windows CE device. Using file explorer to launch them or open a command prompt to run them.
   II. Select “Target -> Run Programs” in Platform Builder to run those two programs.
   III. Using **s** command in Target Control window in Platform Builder (ex: “s CMAccept”).

6. After developers run one of those three steps, press “Connect” button to make connection with your Windows CE device.

5.5. Make Standalone Boot Image

To make standalone boot image, you have to disable KITL and CE target control (on Windows CE 5.0) in build option. Some developers do not disable KITL to boot CE image and get black screen. This is because CE image will try to connect to Platform Builder via KITL.



- [ ] Buffer tracked events in RAM (IMGOSCAPTURE=1)
- [ ] Enable eboot space in memory (IMGEBOOT=1)
- [ ] Enable event tracking during boot (IMGCELOGENABLE=1)
- [ ] Enable hardware-assisted debugging support (IMGHDSTUB=1)
- [ ] Enable kernel debugger (no IMGNODEBUGGER=1)
- [ ] Enable KITL (no IMGNOKITL=1)
- [ ] Enable profiling (IMGPROFILER=1)
- [ ] Flush tracked events to release directory (IMGAUTOFLUSH=1)
- [ ] Run-time image can be larger than 32 MB (IMGRAM64=1)
- [ ] Use xcopy instead of links to populate release directory (BUILDREL_USE_COPY=1)
- [ ] Write run-time image to flash memory (IMGFLASH=1)

(Build Options)

5.6. Install Boot Loader

There are two boot loader for x86 architecture: LoadCEPC.exe and X86 BIOS loader. Before installing loader, you need to make primary partition, set it as "active" and format the partition with FAT16 file system. Those steps are basic and we assume you know that. Or, search it from Google for more detail.

Developers can make a bootable DOM/HDD to boot on target board and set your DOM as slave IDE device to install boot loader and copy Windows CE image. This method is more complexity that boot from USB. Refer to http://www.google.com/search?q=HP+USB+Boot+tool or "Demo Images" section in "Operating System Support" in http://www.dmp.com.tw/tech/vortex86sx.

LoadCEPC.exe

Locate **\WINCE600\PLATFORM\CEPC\SRC\BOOTLOADER\DOS\BOOTDISK**. Here are necessary files to load Windows CE from DOS via loadcepc.exe utility.

1. Copy **autoexec.bat, config.sys, himem.sys and loadcepc.exe** from USB pen drive onto DOM. Developers can run "format c: /s" to transfer DOS boot files. Or, you need to run "sys" command for that.
2. Also make sure the version himem.sys is the same as format/sys command. If your USB is boot from Windows 98 DOS, copy himem.sys from your Windows 98 system. If your USB or DOM is boot from MS-DOS 6.22, copy himem.sys from MS-DOS 6.22. As our test, the himem.sys in MS-DOS 5.0 will make Windows CE report error after image is loaded onto memory to run.
3. This is optional step. You can modify config.sys as:
```
Device=himem.sys /testmem:off
dos=high
```
4. This is optional step. You can modify autoexec.bat as:
```
@echo off
loadcepc.exe nk.bin
```
5. If step 3 and 4 are skipped, you have to select "**Boot CE/PC (local nk.bin)**" while DOM is booting.
6. Copy your nk.bin onto DOM

X86 BIOS Loader

Locate **WINCE600\PLATFORM\CEPC\SRC\BOOTLOADER\BIOSLOADER\DISKIMAGES\SETUPDISK**. Here are necessary files to load Windows CE from X86 BIOS loader:

1. Run "**mkdisk c:**". It will install x86 BIOS loader onto DOM.
2. Modify the **boot.ini** on DOM. Find the "BinFile" in boot.ini. If it is not "**BinFile=nk.bin**", correct it.
3. Copy nk.bin from your pen drive onto DOM
4. Now, you DOM will boot without DOS and show splash BMX file to load Windows CE.

Refer to http://blogs.msdn.com/mikehall/archive/2007/07/11/splash-bmx-what-s-a-bmx-file.aspx to change boot logo for X86 BIOS loader.

5.7. IDE Issue with Hive-based Registry

Hive-based registry in Windows CE is for developers who need to save registry changes. To enable it, just check Hive-based Registry support in BSP. Any registry change will be saved on to DOM/HDD (for example: home page of Internet Explorer). If Windows CE is boot from USB or DOM which is not primary master IDE device, Windows CE will hang at hive-based registry boot phase.

5.8. Add Shortcut on Desktop

1. Assume your program name is Demo_App.exe. Copy it to **\WINCE600\PBWorkspaces\<ProjectName>\WINCE600\Virtex86DX_60A_x86\OAK\files**.
2. Create a shortcut file as file name “Demo_App.lnk” at the same place with Dmo_App.exe and contains one line (where 21 is the string length after #):

```
21#\Windows\Demo_App.exe
```

3. Modify **\WINCE600\PBWorkspaces\<ProjectName>\WINCE600\Vortex86DX_60A_x86\OAK\files\project.bib** to add demo program and shortcut into image.

```
FILES
;  Name                 Path                                         Memory Type
;  ----------------  -----------------------------------------  -----------
Demo_App.exe         $(_FLATRELEASEDIR)\Demo_App.exe            NK S
Demo_App.lnk         $(_FLATRELEASEDIR)\Demo_App.lnk            NK S
```

4. Open **\WINCE600\PBWorkspaces\<ProjectName>\WINCE600\ Vortex86DX_60A_x86\OAK\files \project.dat** to add them:

```
Directory("\Windows\Desktop"):-File("Demo_App.lnk", "\windows\Demo_App.lnk")
```

5.9. Run Program Automatically

Programmers put shortcut into startup folder or registry settings to load program after Windows CE boot up.

Add Shortcut onto StartUp Folder

1. Assume your pgroa name is Demo_App.exe. Copy it to **\WINCE600\PBWorkspaces\<ProjectName>\WINCE600\Virtex86DX_60A_x86\OAK\files**.
2. Create a shortcut file as file name “Demo_App.lnk” at the same place with Dmo_App.exe and contains one line (where 21 is the string length after #):

```
21#\Windows\Demo_App.exe
```

3. Modify **\WINCE600\PBWorkspaces\<ProjectName>\WINCE600\Vortex86DX_60A_x86\OAK\files\project.bib** to add demo program and shortcut into image.

```
FILES
;  Name                 Path                                                Memory Type
;  ----------------  -----------------------------------------  -----------
Demo_App.exe         $(_FLATRELEASEDIR)\Demo_App.exe            NK S
Demo_App.lnk         $(_FLATRELEASEDIR)\Demo_App.lnk            NK S
```

4. Open **\WINCE600\PBWorkspaces\<ProjectName>\WINCE600\Vortex86DX_60A_x86\OAK\files\project.dat** to add them:

```
Directory("\Windows\StartUp"):-File("Demo_App.lnk", "\windows\Demo_App.lnk")
```

After that, Windows CE will run the shortcut in StartUp folder automatically.

Add Registry Settings to Run Program

1. Assume your pgroa name is Demo_App.exe. Copy it to **\WINCE600\PBWorkspaces\<ProjectName>\WINCE600\Virtex86DX_60A_x86\OAK\files**.
2. Modify **\WINCE600\PBWorkspaces\<ProjectName>\WINCE600\Vortex86DX_60A_x86\OAK\files\project.bib** to add demo program and shortcut into image.

```
FILES
;  Name                 Path                                                Memory Type
;  ----------------  -----------------------------------------  -----------
Demo_App.exe         $(_FLATRELEASEDIR)\Demo_App.exe            NK S
```

3. Adding those registry settings to run it at boot up: (Those registry settings are for example only. It depends on real registry settings on your system. Search " **Configuring a Registry File to Run an Application at Startup** " in help for more detail)

```
[HKEY_LOCAL_MACHINE\init]
  "Launch40"="Demo_App.exe"
  "Depend40"=hex:14,00
```

5.10. Using Static IP Address

Vortex86 series BSP will use DHCP for Ethernet by default. If programmers need to use static IP address, please add those registry settings into protect.reg:

```
; Static IP address settings
[HKEY_LOCAL_MACHINE\Comm\PCI\R60401\Parms\TcpIp]
  "EnableDHCP"=dword:0
  "DefaultGateway"=multi_sz:"192.168.0.1"
  "UseZeroBroadcast"=dword:0
  "IpAddress"=multi_sz:"192.168.0.232"
  "Subnetmask"=multi_sz:"255.255.255.0"
```

5.11. Adding FTP, TELNET Server and Folder Sharing

Add Relative Components

To add FTP, TELNET and folder sharing functions, add those components:

- Third Party -> BSP-> Vortex86Dx_60A -> RAM Size -> 256MB RAM
- Third Party -> BSP-> Vortex86Dx_60A -> R6040 Ethernet Driver
- Core OS -> CEBASE -> Networking - Local Area Network (LAN) -> Wired Local Area Network (802.3, 802.5).
- Core OS -> CEBASE -> Networking - General -> Windows Networking API/Redirector (SMB/CIFS)
- Core OS -> CEBASE -> Communication Services and Networking -> Servers -> FTP server
- Core OS -> CEBASE -> Communication Services and Networking -> Servers -> Telnet server

If you need USB mass storage and IDE DOM support, add those:

- Core OS -> CEBASE -> Core OS Services -> USB Host Support -> USB Storage Class Driver
- Device Drivers -> Storage Devices -> ATAPI PCI Support

Add Registry Settings

For FTP, TELNET server and folder sharing, recommend using static IP address for easy connection. Here are registry settings to use static IP address and enable FTP, Telnet server and folder sharing. Add them onto project.reg:

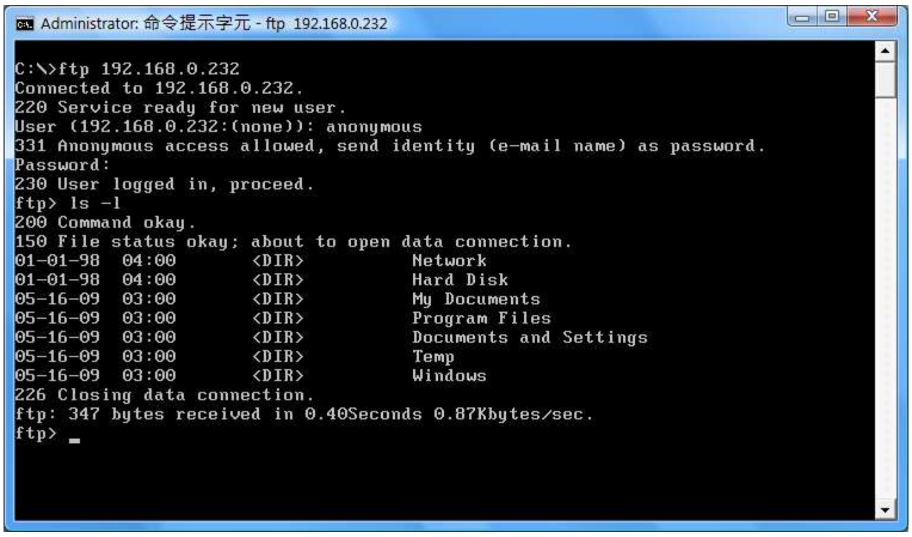
```
; Static IP address settings
[HKEY_LOCAL_MACHINE\Comm\PCI\R60401\Parms\TcpIp]
  "EnableDHCP"=dword:0
  "DefaultGateway"=multi_sz:"192.168.0.1"
  "UseZeroBroadcast"=dword:0
  "IpAddress"=multi_sz:"192.168.0.232"
  "Subnetmask"=multi_sz:"255.255.255.0"

; Telnet server enable
[HKEY_LOCAL_MACHINE\COMM\TELNETD]
  "IsEnabled"=dword:1
  "UseAuthentication"=dword:0

; FTP server endable
[HKEY_LOCAL_MACHINE\COMM\FTPD]
  "IsEnabled"=dword:1
  "UseAuthentication"=dword:0
  "UserList"="@*;"
  "AllowAnonymous"=dword:1
  "AllowAnonymousUpload"=dword:1
  "AllowAnonymousVroots"=dword:1
  "DefaultDir"="\\"

; Folder Sharing
[HKEY_LOCAL_MACHINE\Services\Smbserver]
  "AdapterList"="*"
  "dll"="smbserver.dll"
  "Keep"=dword:1
  "Order"=dword:9
```

```
[HKEY_LOCAL_MACHINE\Services\SMBServer\Shares]
  "UseAuthentication"=dword:0

[HKEY_LOCAL_MACHINE\Services\SMBServer\Shares\HDD]
  "Path"="\\Hard Disk"
  "Type"=dword:0
  "UserList"="@*;"
```

In FTPD registry, we enable anonymous login and the default root path is the whole file system on Windows CE device. For more information about registry settings, please search “FTP server” or “TELNET” in help of Platform Builder.

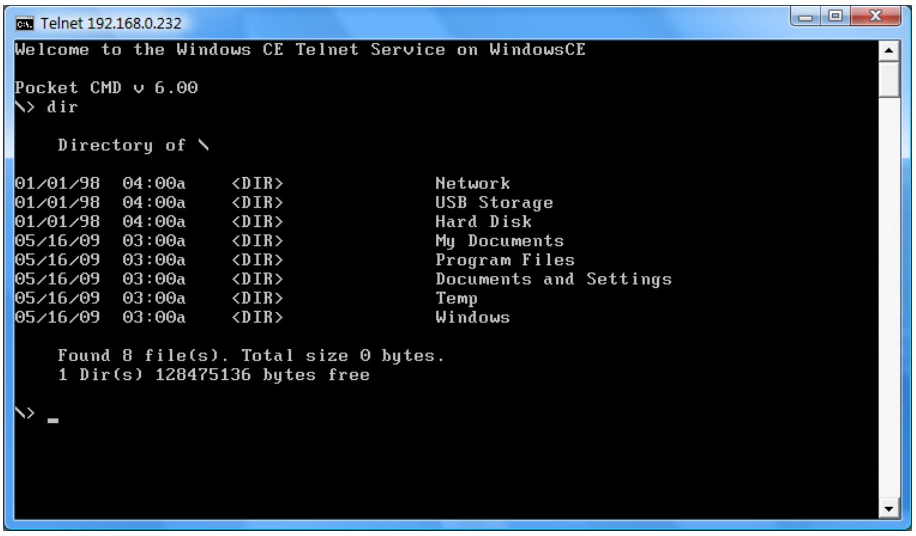
Access Test

We use FTP client in Windows command prompt to demo. Developers can use their favorite FTP client to do files exchange between Windows CE device and desktop computer. The IP of Windows CE is 192.168.0.232. Use user name “anonymouse” and password “ftp” to login FTP server on Windows CE. Here is the FTP test screen:



```
C:\>ftp 192.168.0.232
Connected to 192.168.0.232.
220 Service ready for new user.
User (192.168.0.232:(none)): anonymous
331 Anonymous access allowed, send identity (e-mail name) as password.
Password:
230 User logged in, proceed.
ftp> ls -l
200 Command okay.
150 File status okay; about to open data connection.
01-01-98  04:00       <DIR>          Network
01-01-98  04:00       <DIR>          Hard Disk
05-16-09  03:00       <DIR>          My Documents
05-16-09  03:00       <DIR>          Program Files
05-16-09  03:00       <DIR>          Documents and Settings
05-16-09  03:00       <DIR>          Temp
05-16-09  03:00       <DIR>          Windows
226 Closing data connection.
ftp: 347 bytes received in 0.40Seconds 0.87Kbytes/sec.
ftp> _
```

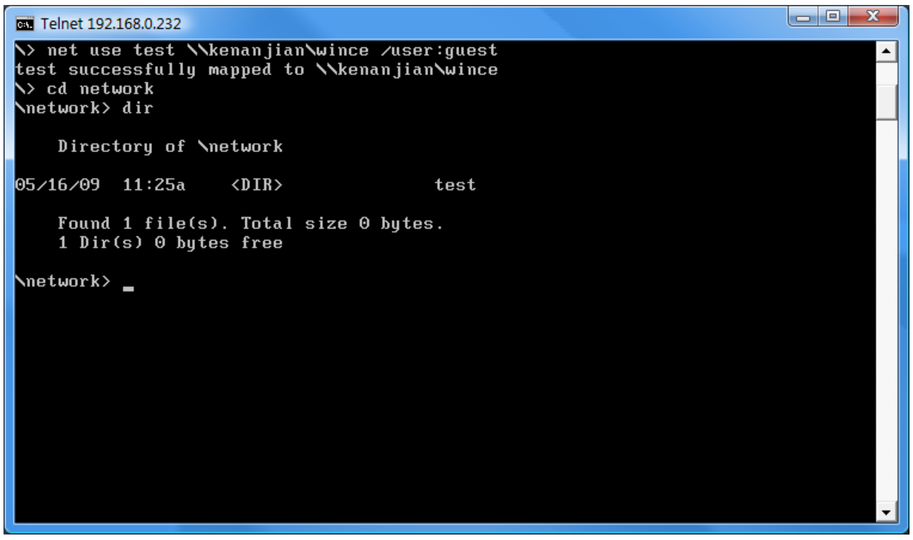
The folder “USB Storage” is USB pen drive and “Hard Disk” is DOM on Vortex86DX board. Below picture is the screen using telnet to connect to Windows CE device:



Telnet 192.168.0.232

```
Welcome to the Windows CE Telnet Service on WindowsCE

Pocket CMD v 6.00
\> dir

    Directory of \

01/01/98  04:00a    <DIR>              Network
01/01/98  04:00a    <DIR>              USB Storage
01/01/98  04:00a    <DIR>              Hard Disk
05/16/09  03:00a    <DIR>              My Documents
05/16/09  03:00a    <DIR>              Program Files
05/16/09  03:00a    <DIR>              Documents and Settings
05/16/09  03:00a    <DIR>              Temp
05/16/09  03:00a    <DIR>              Windows

    Found 8 file(s). Total size 0 bytes.
    1 Dir(s) 128475136 bytes free

\> _
```

Developers can run program or some jobs via telnet. Net command is used via telnet to map share folder. Run “net” to map share folder on desktop PC to Windows CE device:



Telnet 192.168.0.232

```
\> net use test \\kenanjian\wince /user:guest
test successfully mapped to \\kenanjian\wince
\> cd network
\network> dir

    Directory of \network

05/16/09  11:25a    <DIR>              test

    Found 1 file(s). Total size 0 bytes.
    1 Dir(s) 0 bytes free

\network> _
```

Folder “wince” on desktop Windows XP/Vista PC will be mapped to “test” folder in “Network” folder of Windows CE root file system. Just change work directory to “Network” and run dir command, you can find test folder. If full access right is enabled, Windows CE device can read or write files in folder test.

Note:

As our test, only domain name (ex: “kenanjian”) can work properly and IP address “192.168.X.X” can not work. Recommend using domain when you test foler sharing in Windows CE 6.0.

5.12. Access PCI Registers

We need to access PCI North Bridge or South Bridge to setup registers. Programmers can access PCI register via PCI address register CF8h and PCI data register CFCh.

Please refer to **PCI Configuration Registers** for detail about CFCh/CF8h registers.

PCI North Bridge and South Bridge in Vortex86SX/DX are:
Vortex86DX_NB Function 0 Configuration Space Registers (IDSEL = AD11/Device 0)

- Vendor ID is 17F3H and Device ID is 6021H.

Vortex86DX_SB Configuration Space Registers (IDSEL = AD18/Device 7)

- Vendor ID is 17F3H and Device ID is 6031H.

We can use in-line assembler code or include DDK to access 32-bit PCI registers. Here are I/O port access functions in our Windows CE examples to access PCI.

In-Line Assembler Code

```
unsigned int IsVortex86SX()
{
  unsigned long val;
  _asm
  {
    mov dx,  0xcf8
    mov eax, 0x80000090
    out dx,  eax
    mov dx,  0xcfc
    in  eax, dx
    mov val, eax
  }

  if(val == 0x31504d44)
    return 1;
  else
    return 0;
}
```

I/O Access Functions

```
// Read I/O port
DWORD InPortWord(WORD dwPort)
{
  DWORD dwVal;
  _asm {
    mov dx,    dwPort
    in  eax,   dx
    mov dwVal, eax
  }
  return dwVal;
}

// Write I/O port
void OutPortWord(WORD dwPort, DWORD dwVal)
{
  _asm {
    mov dx,  dwPort
    mov eax, dwVal
    out dx,  eax
  }
}
```

5.13. Check Vortex86SX/DX/MX

Programmers can read PCI register 93H~90H in North Bridge to check Vortex86SX/DX/MX SoC. Here is example to access PCI configuration space registers via CFCH/CF8H port.

Windows CE Example

```
#include "stdafx.h"

unsigned int IsVortex86SX()
{
  unsigned long val;
  _asm
  {
    mov dx,  0xcf8
    mov eax, 0x80000090
    out dx,  eax
    mov dx,  0xcfc
    in  eax, dx
    mov val, eax
  }

  // Vortex86SX: 0x31504d44
  // Vortex86DX: 0x32504d44
  // Vortex86MX: 0x33504d44
  // We check Vortex86SX here
  if(val == 0x31504d44)
    return 1;
  else
    return 0;
}

int _tmain(int argc, TCHAR *argv[], TCHAR *envp[])
{
  if(IsVortex86SX())
    _tprintf(_T("Vortex86SX found."));
  else
    _tprintf(_T("Vortex86SX not found."));

  return 0;
}
```

5.14. Change CPU Speed

Internally the Vortex86SX/DX is using the PLL technology (Phase-Locked Loop), the CPU clock can be adjust by changing the register value of the North Bridge Offset register A0h.

The CPU speed could be divided from 1 to 8 by default CPU clock. For example, if the default CPU clock is 300MHz, and you choice the "CPU speed divide by 5", the CPU speed will be 300 / 5 = 60 MHz. And please be noticed the CPU speed could not lower then PCI speed which is 33MHz.

To change CPU clock, find PCI register A0H in north bridge (Vendor ID: 17F3, Device: 6021). Bit 7-3 is reserved and bit 2-0 is CPU speed divided control:

Bit[2-0]	Description
000	Divide 1
001	Divide 2
010	Divide 3
011	Divide 4
100	Divide 5
101	Divide 6
110	Divide 7
111	Divide 8

For example: if CPU clock is 300MHz and set speed divided control to "001", CPU clock will be 150MHz. Here is assembler example:

```
mov dx,  cf8h ; PCI address port set = north bridge offset register a0h
mov eax, 800000a0h
out dx,  eax

mov dx,  cfch ; PCI data port read / write
in  eax, dx
                   ; if CPU clock is  300MHz
or  eax, 00000001h ; set CPU clock to 150MHz
or  eax, 00000004h ; set CPU clock to  60MHz
out dx,  eax
```

Windows CE Example

```
#include "stdafx.h"
#include <stdlib.h>

int _tmain(int argc, TCHAR *argv[], TCHAR *envp[])
{
  _asm {
    mov dx,  0xcf8
    mov eax, 0x800000a0
    out dx,  eax
    mov dx,  0xcfc
    in  eax, dx
    and eax, 0xF8
    or  eax, 0x01 /* If CPU is 300MHz, set clock to 150MHz */
    out dx,  eax
  }
  return 0;
}
```

5.15. GPIO

40 GPIO pins are provided by the Vortex86SX/DX for general usage in the system. All GPIO pins are independent and can be configured as inputs or outputs; when configured as outputs, pins have 8 mA drive capability and are unterminated; when configured as inputs, pins are pulled-high with a 75k ohm resistance.

GPIO port 0,1 and 2 are always free for use normally. If your system does not use external RTC and SPI, GPIO port 3 is also free for use. Developer also can disable COM1 to select GPIO port 4. The actual free GPIO pins depend on your system. Please check it before using GPIO.

Setup GPIO Direction

Here is GPIO direction and data registers:

	Port 0	Port 1	Port 2	Port 3	Port 4	Description
Data Register	78H	79H	7AH	7BH	7CH	
Direction Register	98H	99H	9AH	9BH	9CH	0: GPIO pin is input mode 1: GPIO pin is output mode

If send value 0FH to port 98H, it means that GPIO port0 [7-4] are input mode and port[3-0] are output mode.
If send value 00H to port 98H, it means that GPIO port0 [7-0] are input mode.
If send value FFH to port 98H, it means that GPIO port0 [7-0] are output mode.
If send value 03H to port 98H, it means that GPIO port0 [7-2] are input mode and port[1-0] are output mode.

Windows CE Example

```
#include "stdafx.h"

unsigned char inportb(int addr)
{
  __asm
  {
    push edx
    mov  edx, DWORD PTR addr
    in   al,  dx
    and eax,  0xff
    pop edx
  }
}

void outportb(int addr, unsigned char val)
{
  __asm
  {
    push edx
    mov  edx, DWORD PTR addr
    mov  al,  BYTE PTR val
    out  dx,  al
    pop  edx
  }
}

int _tmain(int nArgCnt, TCHAR *pArg[], TCHAR *pEnv[])
{
  /* set GPIO port0[7-0] as input mode */
  outportb(0x98, 0x00);

  /* read data from GPIO port0 */
  inportb(0x78);

  /* set GPIO port1[7-0] as output mode */
  outportb(0x99, 0xff);

  /* write data to GPIO port1 */
  outportb(0x79, 0x55);

  /* set GPIO port2[7-4] as output and [3-0] as input*/
  outportb(0x9a, 0xf0);

  /* write data to GPIO port2[7-4], the low nibble (0x0a) will be ignored */
  outportb(0x7a, 0x5a);

  /* read data from port2[3-0] */
  unsigned char c = inportb(0x7a) & 0x0f;

  /*--- if GPIO port3 is free, those codes can work ---*/

  /* set GPIO port3[7-2] as output and [1-0] as input*/
  outportb(0x9b, 0xfc);

  /* write data to GPIO port2[7-2], the bit 1-0 will be ignored */
  outportb(0x7b, 0xa5);

  /* read data from port3[1-0] */
  c = inportb(0x7b) & 0x03;

  /*--- if GPIO port4 is free, those codes can work ---*/

  /* set GPIO port4[7,5,3,1] as output and port4[6,4,2,0] as input*/
  outportb(0x9c, 0xaa);

  /* write data to GPIO port4[7,5,3,1], the bit 6,4,2 and0 will be ignored */
  outportb(0x7c, 0xff);

  /* read data from port4[6,4,2,0] */
  c = inportb(0x7c) & 0xaa;

  return 0;
}
```

5.16. Watchdog Timer

There are two watchdog timers in Vortex86SX/DX CPU. One is compatible with M6117D watchdog timer and the other is new. The M6117D compatible watchdog timer is called WDT0 and new one is called WDT1.

WDT0

To access WDT0 registers, programmers can use index port 22H and data port 23H. The watchdog timer uses 32.768 kHz frequency source to count a 24-bit counter so the time range is from 30.5u sec to 512 sec with resolution 30.5u sec. When timer times out, a system reset, NMI or IRQ may happen to be decided by BIOS programming.

Index Port 37h			
Bit 7	Reserved.		
Bit 6	0: Disable WDT0 1: Enable WDT0 (default)		
Bit 5-0	Reserved.		
Index Port 3Ch			
Bit 7	0: Read only, Watchdog timer time out event does not happen. 1: Read only, Watchdog timer time out event happens.		
Bit 6	Write 1 to reset Watchdog timer.		
Index Port 38h			
Bit 7-4	0000:Reserved	0101:IRQ7	1011:IRQ15
	0001:IRQ3	0110:IRQ9	1100:NMI
	0010:IRQ4	0111:IRQ10	1101:System reset
	0011:IRQ5	1001:IRQ12	1110:Reserved
	0100:IRQ6	1010:IRQ14	1111:Reserved
Bit 3-0	Reserved.		

Index 3Bh, 3Ah, 39h : Counter

	3Bh	3Ah	39h
	D7……D0	D7……D0	D7……D0
Counter	Most SBit ……………………………………………………………………………………Least SBit		

Here are steps to setup watchdog timer:

1. Set Bit 6 = 0 to disable the timer.
2. Write the desired counter value to 3Bh, 3Ah, 39h.
3. Set Bit 6 = 1 to enable the timer, the counter will begin to count up.
4. When counter reaches the setting value, the time out will generate signal setting by index 38h bit[7:4]
5. BIOS can read index 3Ch Bit 7 to decide whether the Watchdog timeout event will happen or not.

To clear the watchdog timer counter:

1. Set Bit 6 = 0 to disable timer. This will also clear counter at the same time.

WDT1

WDT1 does not use index and data port to access WDT registers. It uses I/O port 68H~6DH. The time resolution of WDT1 is 30.5 u second. Here are registers information:

WDT1 Control Register

Port 68h	
Bit 7	Reserved.
Bit 6	0: Disable watchdog timer. 1: Enable watchdog timer.
Bit 5-0	Reserved.

WDT1 Signal Select Control Register

Port 69h			
Bit 7-4	0000:Reserved	0101:IRQ7	1011:IRQ15
	0001:IRQ3	0110:IRQ9	1100:NMI
	0010:IRQ4	0111:IRQ10	1101:System reset
	0011:IRQ5	1001:IRQ12	1110:Reserved
	0100:IRQ6	1010:IRQ14	1111:Reserved
Bit 3-0	Reserved.		

WDT1 Control 2 Register

Port	**6Ch**	**6Bh**	**6Ah**
	D7……D0	D7……D0	D7……D0
Counter	Most SBit …………………………………………………………………………………Least SBit		

Resolution is 30.5u second.

WDT1 Status Register

Port 6Dh	
Bit 7	0: WDT1 timeout event does not happen 1: WDT1 timeout event happens (write 1 to clear this flag)
Bit 6-0	Reserved.

WDT1 Reload Register

Port 67h	
Bit 7-0	Write this port to reload WDT1 internal counter. The read data is unknown.

Here are steps to setup WDT1:

4. Write time into register 6Ah-6Ch.
5. Select signal from register 69h.
6. Set register 68h bit 8 to enable WDT1.

To clear the watchdog timer counter:

2. Write any value to register 67H

WDT0 Windows CE Example

```
#include "stdafx.h"

unsigned char inportb(int addr)
{
  __asm
  {
    push edx
    mov  edx, DWORD PTR addr
    in   al,  dx
    and  eax, 0xff
    pop  edx
  }
}

void outportb(int addr, unsigned char val)
{
  __asm
  {
    push edx
    mov  edx, DWORD PTR addr
    mov  al,  BYTE PTR val
    out  dx,  al
    pop  edx
  }
}

int _tmain(int nArgCnt, TCHAR *pArg[], TCHAR *pEnv[])
{
  outp(0x22,0x13); // Lock register
  outp(0x23,0xc5); // Unlock config. register

  // 500 mini-second
  unsigned int time = 0x20L * 500L;
  outp(0x22,0x3b);
  outp(0x23,(time>>16)&0xff);
  outp(0x22,0x3a);
  outp(0x23,(time>> 8)&0xff);
  outp(0x22,0x39);
  outp(0x23,(time>> 0)&0xff);

  // Reset system
  outp(0x22,0x38);
  unsigned char c = inp(0x23);
  c &= 0x0f;
  c |= 0xd0; // Reset system. For example, 0x50 to trigger IRQ7
  outp(0x22,0x38);
  outp(0x23,c);
  // Enable watchdog timer
  outp(0x22,0x37);
  c = inp(0x23);
  c |= 0x40;
  outp(0x22,0x37);
  outp(0x23,c);

  outp(0x22,0x13); // Lock register
  outp(0x23,0x00); // Lock config. register

  printf("Press any key to stop trigger timer.\n");
  while(!kbhit())
  {
    outp(0x22,0x13);  // Unlock register
    outp(0x23,0xc5);
    outp(0x22,0x3c);
    unsigned char c = inp(0x23);
    outp(0x22,0x3c);
    outp(0x23,c|0x40);
    outp(0x22,0x13);  // Lock register
    outp(0x23,0x00);
  }

  printf("System will reboot after 500 milli-seconds.\n");
  return 0;
}
```

WDT1 Windows CE Example

```
#include "stdafx.h"

unsigned char inportb(int addr)
{
  __asm
  {
    push edx
    mov  edx, DWORD PTR addr
    in   al,  dx
    and  eax, 0xff
    pop  edx
  }
}

void outportb(int addr, unsigned char val)
{
  __asm
  {
    push edx
    mov  edx, DWORD PTR addr
    mov  al,  BYTE PTR val
    out  dx,  al
    pop  edx
  }
}

int _tmain(int nArgCnt, TCHAR *pArg[], TCHAR *pEnv[])
{
  // 500 mini-second
  unsigned long time = 0x20L * 500L;
  outp(0x6c, (time >> 16) & 0xff);
  outp(0x6b, (time >>  8) & 0xff);
  outp(0x6a, (time >>  0) & 0xff);

  // Reset system. For example, 0x50 to trigger IRQ7
  outp(0x69, 0xd0);

  // Enable watchdog timer
  unsigned char c = inp(0x68);
  c |= 0x40;
  outp(0x68, c);

  printf("Press any key to stop trigger timer.\n");
  while(!kbhit())
    outp(0x67, 0x00);

  printf("System will reboot after 500 milli-seconds.\n");

  return 0;
}
```

5.17. ISO-in-Chip

It is a 32 Byte one-time write Flash area in Vortex86SX/DX/MX. Factory will store all necessary data of board (& SoC) in this area for service tracing. Necessary data will include:

- Date code of all major component of the board
- Model number and serial code of the board
- Unique serial code of SoC
- Customer (distributor) code
- Shipment date, etc.

The ISOinChip data will be stored at offset EFH~D0H of PCI North Bridge (Vender ID: 17F3H, Device ID: 6021H).

Windows CE Example

```
#include "stdafx.h"

// Read I/O port
DWORD InPortWord(WORD dwPort);

// Write I/O port
void OutPortWord(WORD dwPort, DWORD dwVal);

// Change DWORD to byte array
void ChangeByteOrder(DWORD dw, unsigned char *p);

// Print ISOinChip data
void DumpData(TCHAR *str, unsigned char pcBuf[], int nStart, int nEnd, int nFormat);

int _tmain(int nArgCnt, TCHAR *pArg[], TCHAR *pEnv[])
{
  // ISOinChip array
  unsigned char pbISOinChip[32] = { 0 };

  for(int i = 0; i < 8; i++)
  {
    // Read NB ISOinChip data
    OutPortWord(0xCF8, 0x800000D0 + i * sizeof(DWORD));
    ChangeByteOrder(InPortWord(0xCFC), pbISOinChip + i * sizeof(DWORD));
  }

  //Print CPU ID data at offset 00h-05h
  DumpData(_T("CPU ID"), pbISOinChip, 0, 5, 1);

  // Print product name data at offset 06h-0Dh
  DumpData(_T("Product Name"), pbISOinChip, 6, 13, 0);

  // Print PCB version at offset 0Eh-13h
  DumpData(_T("PCB Version"), pbISOinChip, 14, 19, 0);

  // Print export week at offset 14h-17h
  DumpData(_T("Export Week"), pbISOinChip, 20, 23, 0);
  // Print user ID at offset 18h-1Fh
  DumpData(_T("USER ID"), pbISOinChip, 24, 31, 1);

  _tprintf(_T("\n\n"));
  return 0;
}

// Print ISOinChip data
//  *str     data title
//  pcBuf[]  ISOinChip data array
//  nStart   start offset
//  nEnd     end offset
//  nFormat  format flag
void DumpData(TCHAR *str, unsigned char pcBuf[], int nStart, int nEnd, int nFormat)
{
  _tprintf(_T("\n %s = "), str);

  for(; nStart <= nEnd; nStart++)
  {
    if(nFormat)
      _tprintf(_T("%02x"), pcBuf[nStart]);
    else
      _tprintf(_T("%c"), pcBuf[nStart]);
  }
}

// Read I/O port
DWORD InPortWord(WORD dwPort)
{
  DWORD dwVal;
  _asm {
    mov dx, dwPort
    in  eax, dx
    mov dwVal, eax
  }
  return dwVal;
}

// Write I/O port
void OutPortWord(WORD dwPort, DWORD dwVal)
{
  _asm {
    mov dx, dwPort
    mov eax, dwVal
    out dx, eax
  }
}

// Change DWORD to byte array
void ChangeByteOrder(DWORD dw, unsigned char *p)
{
  p[0] = dw >> 0;
  p[1] = dw >> 8;
  p[2] = dw >> 16;
  p[3] = dw >> 24;
}
```

5.18. PWM

There are 3 PWM (8254 counter) in Vortex86SX/DX and they share the same pins of COM2. This document will show programmers to program 8254 to make PWM work.

COM2 and PWM share the same pins in Vortex86SX. COM 2 is enabled by default and PWM is disabled by default. In order to enable PWM, we have to disable COM2 and enable PWM. We have to set bit 2 in Internal Peripheral Feature Control Register (C3H~C0H) in south bridge (vendor ID=17F3H, device ID=6031H).

Register Offset:	C3-C0h
Register Name:	Internal Peripheral Feature Control Register
Reset Value:	032C0500h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Rsvd		IO16W			IO8W			MEM16W			MEM8W			ISACLK	Int_zw	Rsvd	IDEIS	PWM2	PWM1	PWM0	FCDC	MCLK	EMIQ	PINS6	SFCE	PINS5	PINS4	CPS	PINS2	PINS1	PINS0

Also set bit 13~11 to use internal clock:

Bit	Attrib	Description
13	R/W	PWM Timer2 0: internal 1.19MHz (default) 1: external clock
12	R/W	PWM Timer1 0: internal 1.19MHz (default) 1: external clock
11	R/W	PWM Timer0 0: internal 1.19MHz (default) 1: external clock
2	R/W	PINs selection for COM2 and PWM 0: 9 PINS for COM2 (default) 1: 9 PINS for PWM

PWM (8254) control register address:

IO Address	Description
48H	PWM Timer/Counter 0 Count Register
49H	PWM Timer/Counter 1 Count Register
4AH	PWM Timer/Counter 2 Count Register
4BH	PWM Timer/Counter Control Register

Here is the PWM pin assignment on COM2 when PWM is enabled:

COM2 Pin	PWM Pin
1 (DCD2)	PWM0_CLK
3 (TXD2)	PWM0_OUT
6 (DSR2)	PWM0_GATE
9 (RI2)	PWM1_CLK
7 (RTS)	PWM1_OUT
8 (CTS2)	PWM1_GATE
2 (RXD2)	PWM2_CLK
4 (DTR)	PWM2_OUT
10 (TXDEN2)	PWM2_GATE

Other operations are the same as 8254. Programmers can search 8254 datasheet it from Google to get more information: http://www.google.com/search?q=8254+datasheet.

Windows CE Example

```
#include "stdafx.h"

DWORD inportdw(WORD io_Port)
{
  DWORD val;
  __asm {
    push edx
    mov dx, io_Port
    in eax, dx
    mov val, eax
    pop edx
  }

  return val;
}

DWORD outoutportdw(WORD io_Port, DWORD val)
{
  __asm {
    push edx
    mov dx, io_Port
    mov eax, val
    out dx, eax
    pop edx
  }

  return 0;
}

void outportw(WORD io_Port,unsigned char val)
{
    __asm {
        push edx
        mov  dx, io_Port
        mov  al, val
        out  dx, al
        pop edx
    }

}

int _tmain(int argc, TCHAR *argv[], TCHAR *envp[])
{
  printf("\nDM&P Vortex86SX/DX PWM Demo Program (%s %s)\n\n", __DATE__, __TIME__);

  if(argc != 4)
  {
    wprintf(_T("Usage: %s <C?> <M?> <?>\n\n"), argv[0]);
    wprintf(_T("C Counter -> 0 ~ 2\n"));
    wprintf(_T("M Mode    -> 0 ~ 5\n"));
    wprintf(_T("? Value   -> 0 ~ 2^16\n"));
    wprintf(_T("\n"));
    wprintf(_T("Ex: %s C0 M1 100   -> Set counter 0 to mode 1 and value = 100\n"), argv[0]);
    wprintf(_T("   %s C1 M5 10000 -> Set counter 1 to mode 5 and value = 10,000\n\n"), argv[0]);
    return 1;
  }

  // Read Internal Peripheral Feature Control Register in Vortex86SX South Bridge
  outoutportdw(0xcf8, 0x800038c0L);

  long tmp = inportdw(0xcfc);

  // Clear bit 13-11 to use internal clock
  tmp &= 0xffffffc7ffL;

  // Switch COM2 to PWM
  tmp |= 0x0000000004L;

  outoutportdw(0xcf8, 0x800038c0L);
  outoutportdw(0xcfc, tmp);

  unsigned char cCmd   = 0x00;
  long          lValue = 0x00;
  unsigned char cCnt   = 0x00;
  unsigned char cMode  = 0x00;

  for(int i = 1; i < argc; i++)
  {
    // Handle counter parameter
    if(argv[i][0] == 'C' || argv[i][0] == 'c')
    {
      unsigned char n = argv[i][1] - '0';
      if(n > 2)
      {
        printf("Counter parameter error\n");
        return 1;
      }

      cCmd |= (n << 6);
      cCnt = n;
    }

    // Handle mode parameter
    else if(argv[i][0] == 'M' || argv[i][0] == 'm')
    {
      unsigned char n = argv[i][1] - '0';
      if(n > 5)
      {
        printf("Mode parameter error\n");
        return 1;
      }

      cCmd |= (n << 1);
      cMode = n;
    }

    // Read value
    else
    {
      lValue = _wtol(argv[i]);
    }
  }

  // Assume value is 16-bits
  cCmd |= (0x03 << 4);

  outportw(0x4b, cCmd);
  outportw(0x48 + cCnt, (unsigned char)(lValue & 0x00ff));
  outportw(0x48 + cCnt, (unsigned char)(lValue >> 8));

  printf("\nPWM Counter%d, Mode%d, Value=%lu\n", cCnt, cMode, lValue);

  return 0;
}
```

5.19. Servo

There are 32 SERVO controls in Vortex86DX for some PWM purpose (ex: robotic). This document will show programmers relative registers in Vortex86DX to control PWM output.

The GPIO port 4 for SERVO is working as COM1 by default. Programmers have to disable COM1 to switch pins for PWM output. Before accessing SERVO control registers, programmers have to assign base address in bit 15-9 of D3H~D0H in PCI South Bridge.

SERVO control also can use hardware interrupt as notify when PWM output is finish. 32 SERVO controls will share the same IRQ. SERVO Interrupt Mask Register is used to determine which SERVO control can cause interrupt, and programmers check SERVO Interrupt Status Register to know which SERVO control launch the interrupt. SERVO Sync register is used for synchronization. Before accessing SERVO registers, get SERVO base address from D3H~D0H in PCI South Bridge.

There are three methods to check when the PWM output is finished:

1. use interrupt;
2. by polling the SERVO Interrupt Status Register;
3. by polling the RC field in SERVO Control Register.

Register Offset: D3h – D0h
Register Name: Internal SERVO Control Register
Reset Value: 00000000h

31-25	24	23	22-20	19-16	15-9	8-0
Reserved	CLKS	UE	Rsvd	SIRT	UIOA	Reserved

Bit	Name	Attribute	Description
31-25	Rsvd	RO	Reserved
24	CLKS	R/W	Servo Clock selection 0: 10MHz (default) 1: 50MHz
23	UE	R/W	Enable/Disable Internal SERVO IO Address Decode 0: Disable (Default) 1: Enable
22-20	Rsvd	RO	Reserved
19-16	SIRT	R/W	SERVO IRQ Routing Table Bit19 Bit18 Bit17 Bit16 Routing Table 0 0 0 0 Disable.

Bit	Name	Attribute	Description
			0 0 0 1 IRQ[9] 0 0 1 0 IRQ[3] 0 0 1 1 IRQ[10] 0 1 0 0 IRQ[4] 0 1 0 1 IRQ[5] 0 1 1 0 IRQ[7] 0 1 1 1 IRQ[6] 1 0 0 0 IRQ[1] 1 0 0 1 IRQ[11] 1 0 1 0 Reserved 1 0 1 1 IRQ[12] 1 1 0 0 Reserved 1 1 0 1 IRQ[14] 1 1 1 0 Reserved 1 1 1 1 IRQ[15] These four bits are used to route SERVO IRQ to any 8259 Interrupt lines. The BIOS should be used to inhibit the setting of the reserved value.
15-9	UIOA	R/W	Internal SERVO IO Address. The Bit[15:9] contain the base IO address A[15:9] of internal SERVO.
8-0	Rsvd	RO	Reserved. All are ‘0’s. Writing any value to these bits causes no effect.

SERVO registers and control registers in PCI South Bridge detail is at **Servo Registers**.

Windows CE Example

```
#include "stdafx.h"
#include <stdlib.h>
#include <stdio.h>

void outpw (WORD wAddr, WORD wVal)
{
  _asm
  {
    mov dx, wAddr
    mov ax, wVal
    out dx, ax
  }
}

WORD inpw(WORD wAddr)
{
  WORD  wVal = 0;
  _asm
  {
    mov dx, wAddr
    in ax, dx
    mov wVal, ax
  }
  return wVal;
}

void outpdw(WORD wAddr, DWORD dwVal)
{
  outpw(wAddr, (WORD) (dwVal & 0xffff));
  outpw(wAddr + 2, (WORD) (dwVal >> 16));
}

DWORD inpdw(WORD wAddr)
{
  return (DWORD) inpw(wAddr) + ((DWORD) inpw(wAddr + 2) << 16);
}

// read south bridge register
DWORD read_sb(BYTE bIdx)
{
  DWORD dwVal = 0;
  _asm
  {
    mov dx, 0x0cf8
    mov eax, 0x80003800
    mov al, BYTE PTR bIdx
    out dx, eax
    mov dx, 0x0cfc
    in eax, dx
    mov dwVal, eax
  }
  return dwVal;
}

// write south bridge register
void write_sb(BYTE bIdx, DWORD dwVal)
{
  _asm
  {
    mov dx, 0x0cf8
    mov eax, 0x80003800
    mov al, bIdx
    out dx, eax
    mov dx, 0x0cfc
    mov eax, dwVal
    out dx, eax
  }
}

unsigned int  base_address = 0xfe00;
unsigned long us = 10L;        // assume the Servo Clock (see Internal SERVO Control Register)
is 10MHZ
void set_pluse(int channel, unsigned long period, unsigned long high_pulse_width)
{
  unsigned long low_pulse_width = period - high_pulse_width;

  if((channel < 0) || (channel >= 32) || (period <= high_pulse_width))
  {
    _tprintf(_T("ERROR: invalid pulse setting!"));
    return;
  }

  outpdw(base_address + 0x0c + channel * 12, low_pulse_width * us);
  outpdw(base_address + 0x10 + channel * 12, high_pulse_width * us);
}
```

```
int _tmain(int argc, TCHAR *argv[], TCHAR *envp[])
{
  int channel = 0;
  if(argc > 1)
    channel = _wtoi(argv[1]);

  printf("DM&P Sevro TEST Programmer (for Vortex86DX) Ver."__DATE__ "\n");
  printf("Copyright (C) 2009 by DM&P Group. All rights reserved.\n\n");

  write_sb(0xc8, 0xffffffffL);  // switch GPIO function to PWM output
  write_sb(0xc0, read_sb(0xc0) | 2L);         // disable GPIO port 4's COM function
  write_sb(0xd0, 0x00800000L + base_address); // disable IRQ, Address Decode enable, 10Mhz

  // disable interrupt.
  outpdw(base_address, 0x00000000L);

  // lock all PWM channel output
  outpdw(base_address + 8, 0xffffffffL);

  // PWM period = 20ms, high_pulse_width (PWM duty) = 2.2ms
  set_pluse(channel, 20000L, 2200L);

  // enable PWM
  // pulse count mode; send 300 PWM pulses
  outpdw(base_address + 0x14 + channel * 12, 0x80000000L + 300);

  // continue mode; send PWM pulses continuously
  //outpdw(base_address+0x14+channel*12, 0xc0000000L);
  // unlock all PWM channel output
  outpdw(base_address + 8, 0x00000000L);

  return 0;
}
```

5.20. SPI

There two SPI interface in Vortex86SX/DX. Internal one is used by BIOS and external one is used by SPI flash disk (ex: ICOP boards). Below is the access flow for external SPI flash access.

SPI base address is at 43H~40H of PCI North Bridge. Refer to **PCI Configuration Registers** for more. The base address from below register is for internal SPI. For external SPI, add 08h for external SPI. For DM&P Vortex86SX/DX, the SPI base address register will be FC01H. From the register detail, the base address for internal SPI is FC00H and FC08H for external SPI.

Register Offset: 43h – 40h
Register Name: SPI Base Address Register
Reset Value: 00000000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PBA																												Rsvd		MIO	En

Bit	Name	Attribute	Description
31-4	PBA	R/W	SPI Base Address A[31-4]. Size is 16 bytes. If it is I/O space, it only uses A[15-4]. If it is memory space, it use A[31-4].
3-2	RSVD	RO	Reserved.
1	MIO	R/W	Memory or I/O space select. 1: Memory space 0: I/O space
0	En	R/W	SPI Base address is enabled when set.

Before accessing external SPI, need to switch GPIO port3[3-0] to SPI pins in PCI South Bridge:

Register Offset: C3-C0h
Register Name: Internal Peripheral Feature Control Register
Reset Value: 032C0500h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Rsvd		IO16W			IO8W			MEM16W			MEM8W			ISACLK	Int_zw	Rsvd	IDEIS	PWM2	PWM1	PWM0	FCDC	MCLK	EMIQ	PINS6	SFCE	PINS5	PINS4	CPS	PINS2	PINS1	PINS0

Bit	Name	Attribute	Description
0	PINS0	R/W	PINS selection for External SPI and GPIO Port3 [3-0]

Bit	Name	Attribute	Description
			0: 4 pins for GPIO port3 [3-0] (default) 1: 4 pins for external SPI

Here is example code access external SPI flash. We do not introduce flash memory commands. The main purpose in example code is show programmers how to access SPI interface.

Windows CE Example

```
#include "stdafx.h"
#include <stdio.h>

// Read north bridge register
unsigned long read_nb(unsigned char idx)
{
  unsigned long retval;
  __asm
  {
      mov dx, 0cf8h
      mov eax, 80000000h
      mov al, BYTE PTR idx
      out dx, eax
      mov dx, 0cfch
      in  eax, dx
      mov DWORD PTR retval, eax
  }
  return retval;
}

// Read south bridge register
unsigned long read_sb(unsigned char idx)
{
  unsigned long retval;
  __asm
  {
      mov dx, 0cf8h
      mov eax, 80003800h
      mov al, BYTE PTR idx
      out dx, eax
      mov dx, 0cfch
      in  eax, dx
      mov DWORD PTR retval, eax
  }
  return retval;
}

// Write south bridge register
void write_sb(unsigned char idx, unsigned long val)
{
  __asm
  {
      mov dx, 0cf8h
      mov eax, 80003800h
      mov al, idx
      out dx, eax
      mov dx, 0cfch
      mov eax, DWORD PTR val
      out dx, eax
  }
}

int _tmain()
{
  // Get SPI base address
  unsigned long lAddr = read_nb(0x40) & 0x0000FFFFL;

  // Check SPI is enabled or not
  if((lAddr & 0x00000001) == 0)
  {
    printf("SPI is not enabled\n");
    return 1;
  }

  // Show address type
  if(lAddr & 0x00000002L)
    printf("Address is at memory space\n");
  else
    printf("Address is at I/O space\n");

  lAddr &= 0x0000FFFF0L;
  lAddr += 8; // for extern SPI
  printf("Extern SPI base address = %04X\n", lAddr);

  // Switch GPIO to extern SPI
  unsigned long a = read_sb(0xc0);
  write_sb(0xC0, a | 0x00000001L);

  //
  // Progrmmer can access SPI via base address in varaible lAddr
  //

  return 0;
}
```

5.21. Buzzer

Some programmers need to make sound to PC speaker (not speaker out from audio chipset) for prompt. Here is example code:

Windows CE Example

```
#include "stdafx.h"

unsigned char inp(short addr)
{
  unsigned char cValue;
  _asm
  {
    mov dx, addr
    in  ax, dx
    mov cValue, al
  }

  return cValue;
}

void outp(int addr, unsigned char val)
{
  __asm
  {
    push edx
    mov  edx, DWORD PTR addr
    mov  al,  BYTE PTR val
    out  dx,  al
    pop edx
  }
}

/*

Note: First parameter is the frequency of the buzzer,Second parameter is duration of the sound
which from the buzzer.

*/

bool MyBeep(DWORD dwFreq, DWORD dwDuration)
{
  outp(0x43, 0xb6); // Set Buzzer
  outp(0x42, (0x1234dc / dwFreq));      // Frequency LSB
  outp(0x42, (0x1234dc / dwFreq) >> 8); // Frequency MSB
  outp(0x61, inp(0x61) | 0x3);          // Start beep
  Sleep(dwDuration);
  outp(0x61, inp(0x61) & 0xfc);         // End beep
  return TRUE;
}
```

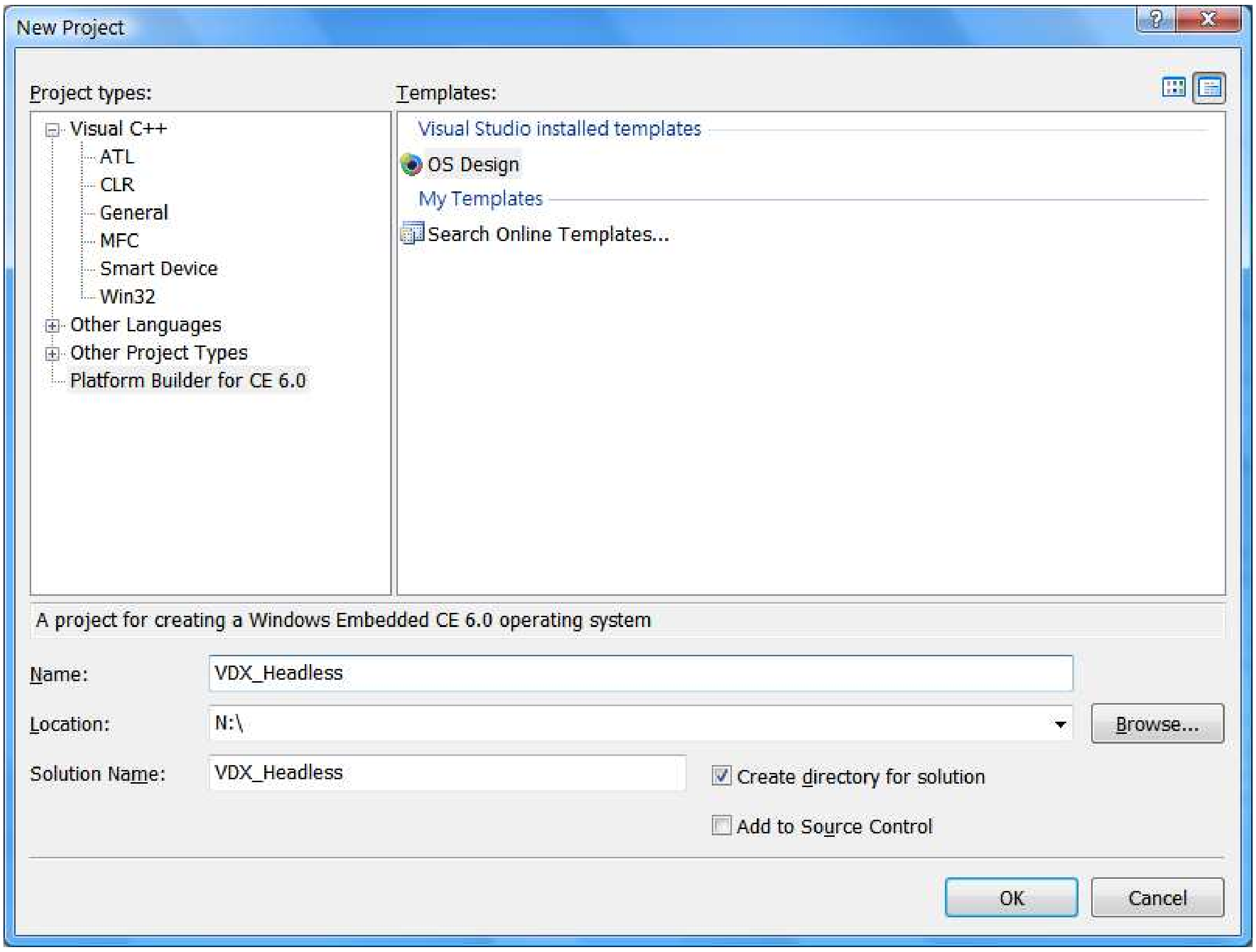
5.22. Building Windows Embedded CE 6.0 Headless Image

This section will show developers to build Windows CE headless image for Vortex86SX/DX devices without display. Programmers can make native programs or console .NET application to run on headless devices. We assume you have Windows CE O/S customize experience to create new O/S design.

Please refer to **BSP** section to install BSP and QFE to start below steps.

Create a Headless Workspace

Use “VDX_Headless” as project name.



A project for creating a Windows Embedded CE 6.0 operating system

Name: VDX_Headless

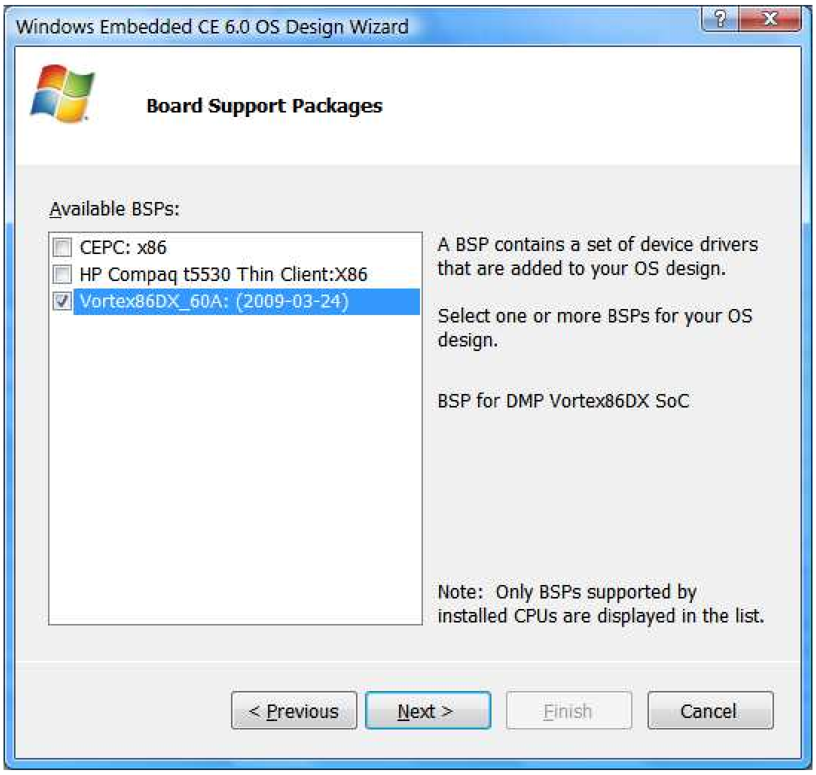
Location: N:\

Solution Name: VDX_Headless

[✓] Create directory for solution

[ ] Add to Source Control

Start new O/S design with Vortex86DX BSP:



- [ ] CEPC: x86
- [ ] HP Compaq t5530 Thin Client:X86
- [✓] Vortex86DX_60A: (2009-03-24)

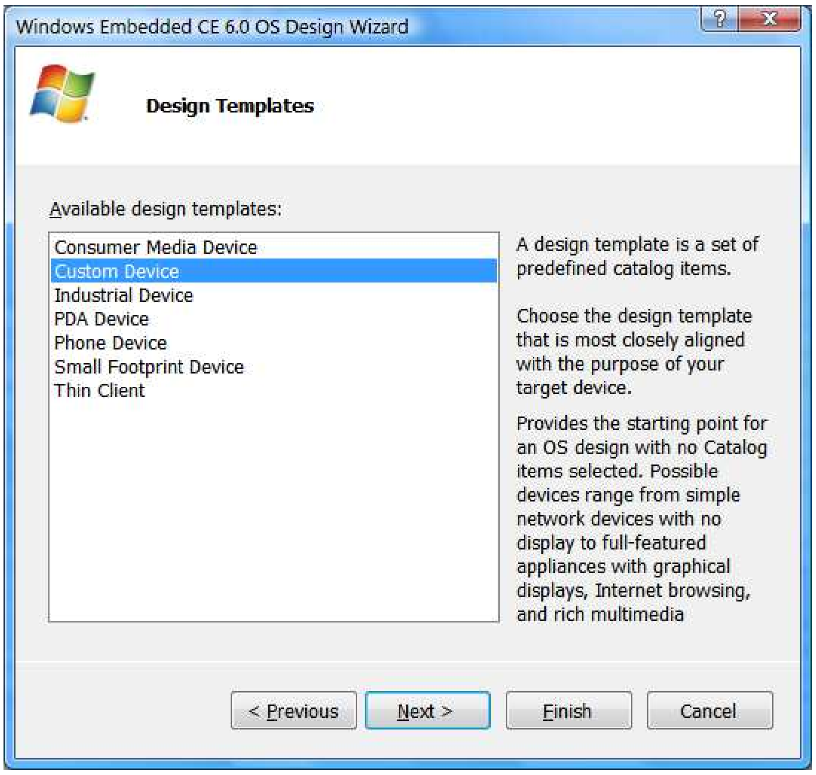
A BSP contains a set of device drivers that are added to your OS design.

Select one or more BSPs for your OS design.

BSP for DMP Vortex86DX SoC

Note: Only BSPs supported by installed CPUs are displayed in the list.

Select template "Custom Device" and press finish button to complete wizard. All necessary components will be added manually later.


Windows Embedded CE 6.0 OS Design Wizard
Design Templates
Available design templates:
< Previous
Next >
Finish
Cancel

Consumer Media Device
Custom Device
Industrial Device
PDA Device
Phone Device
Small Footprint Device
Thin Client

A design template is a set of predefined catalog items.

Choose the design template that is most closely aligned with the purpose of your target device.

Provides the starting point for an OS design with no Catalog items selected. Possible devices range from simple network devices with no display to full-featured appliances with graphical displays, Internet browsing, and rich multimedia

Add Necessary Components

Here are network components used by our demo:

- Third Party -> BSP-> Vortex86Dx_60A -> RAM Size -> 256MB RAM
- Third Party -> BSP-> Vortex86Dx_60A -> R6040 Ethernet Driver
- Core OS -> CEBASE -> Networking - Local Area Network (LAN) -> Wired Local Area Network (802.3, 802.5).
- Core OS -> CEBASE -> Networking - General -> Windows Networking API/Redirector (SMB/CIFS)
- Core OS -> CEBASE -> Communication Services and Networking -> Servers -> FTP server
- Core OS -> CEBASE -> Communication Services and Networking -> Servers -> Telnet server

For .Net headless application, add this component:

- Core OS -> CEBASE -> Applications and Services Development -> .NET Compact Framework 3.5 -> .NET Compact Framework 3.5 - Headless

If you need USB mass storage and IDE DOM support, add those:

- Core OS -> CEBASE -> Core OS Services -> USB Host Support -> USB Storage Class Driver
- Device Drivers -> Storage Devices -> ATAPI PCI Support

Add Registry Settings

Here are registry settings to use static IP address and enable FTP/Telnet server. Add them onto project.reg:

```
; Static IP address settings
[HKEY_LOCAL_MACHINE\Comm\PCI\R60401\Parms\TcpIp]
  "EnableDHCP"=dword:0
  "DefaultGateway"=multi_sz:"192.168.0.1"
  "UseZeroBroadcast"=dword:0
  "IpAddress"=multi_sz:"192.168.0.232"
  "Subnetmask"=multi_sz:"255.255.255.0"

; Telnet server enable
[HKEY_LOCAL_MACHINE\COMM\TELNETD]
  "IsEnabled"=dword:1
  "UseAuthentication"=dword:0

; FTP server endable
[HKEY_LOCAL_MACHINE\COMM\FTPD]
  "IsEnabled"=dword:1
  "UseAuthentication"=dword:0
  "UserList"="@*;"
  "AllowAnonymous"=dword:1
  "AllowAnonymousUpload"=dword:1
  "AllowAnonymousVroots"=dword:1
  "DefaultDir"="\\"
```

In FTPD registry, we enable anonymous login and the default root path is the whole file system on Windows CE device. For more information about registry settings, please search “FTP server” or “TELNET” in help of Platform Builder.

DHCP is enabled by default for all Windows CE projects in Vortex86DX BSP. Because there is no display on our Windows CE device, it needs some operations to get IP address of CE device. So, we set fixed IP address in this demo. For most real applications, fixed IP address is used. Add those registry settings into project to disable DHCP and use fixed IP address. The TCP/IP settings in registry are for your reference. Change them according your network environment.

Run Program Automatically

Programmers can make native C program or .Net application running on headless Windows CE image. We are using .Net application in this demo. This .Net demo program will send data to serial port 1000 times. Put .Net demo program "VDX_Headless_Net_Demo.exe" to "**\VDX_Headless\Wince600\Vortex86DX_60A_x86\OAK\files**" and add those registry settings to run it at boot up: (Search "HKEY_LOCAL_MACHINE\init" in help for more detail)

```
[HKEY_LOCAL_MACHINE\init]
  "Launch40"="VDX_Headless_Net_Demo.exe"
  "Depend40"=hex:14,00
```

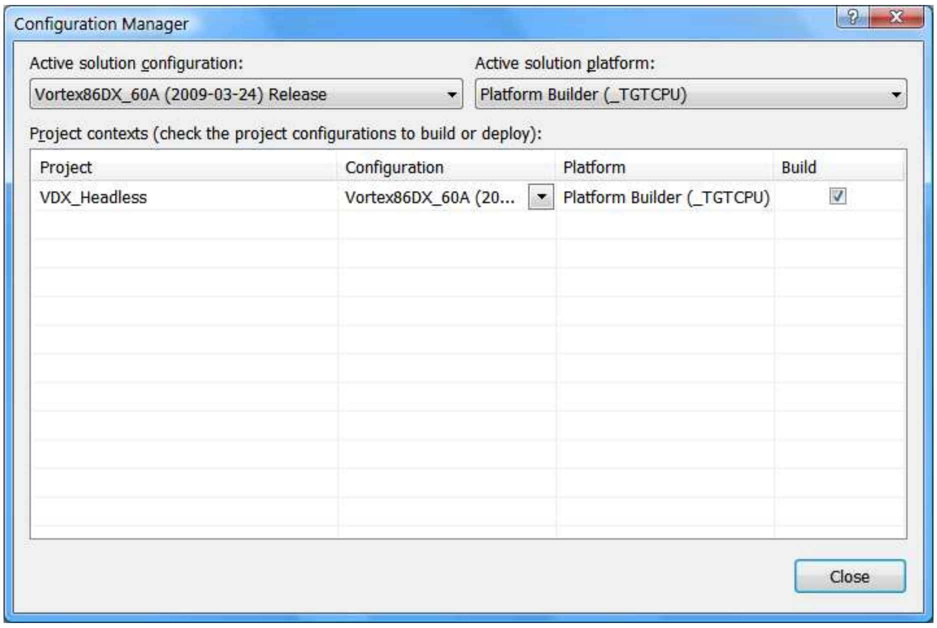
Modify project.bib to add this line for Platform Builder to include demo program into Windows CE image:

```
FILES
;  Name                      Path                                         Memory Type
;  ----------------------    --------------------------------------------  -----------
VDX_Headless_Net_Dome.exe   $(_FLATRELEASEDIR)\VDX_Headless_Net_Demo.exe   NK      S
```

Build Release Image

Developers can build debug image with KITL function to boot CE image via eboot.bin for system debug. After debug is done, build release image to boot from USB pen drive or DOM. Read "Windows CE Development Note" at ftp://download@ftp.dmp.com.tw/vortex86dx/WinCE_Development_Note.pdf for more detail.

To make release Windows CE image, select "VS2005 -> Build -> Configuration Manager" to enable release mode:



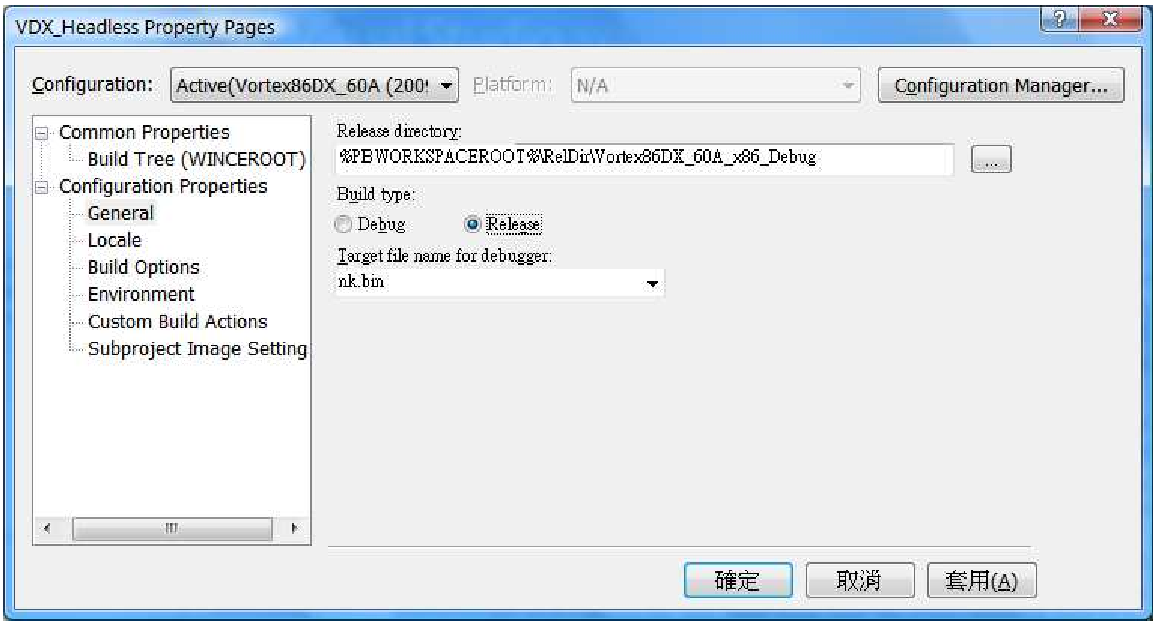
Active solution configuration: Vortex86DX_60A (2009-03-24) Release

Active solution platform: Platform Builder (_TGTCPU)

Project contexts (check the project configurations to build or deploy):

Project	Configuration	Platform	Build
VDX_Headless	Vortex86DX_60A (20...	Platform Builder (_TGTCPU)	✓

Select “VS2005 -> Project -> VSX_Headless Properties“ to select release image as picture:

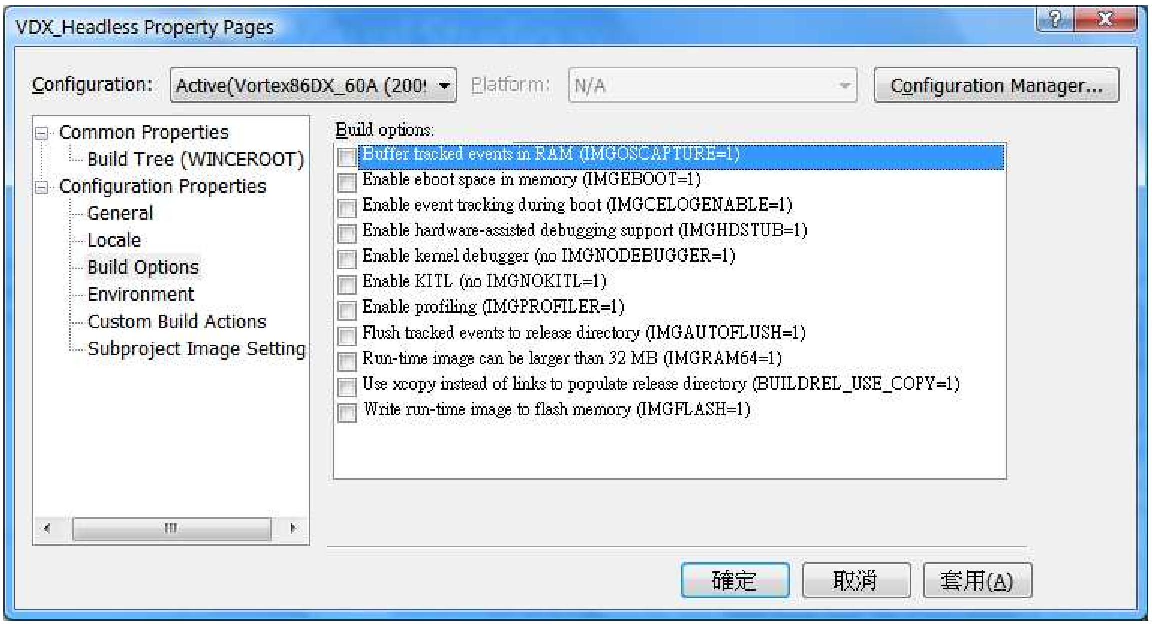


Release directory:
%PBWORKSPACEROOT%\RelDir\Vortex86DX_60A_x86_Debug

Build type:
Debug [✓ Release]

Target file name for debugger:
nk.bin

Uncheck debug options for release mode. Or, uncheck all options is a good solution:



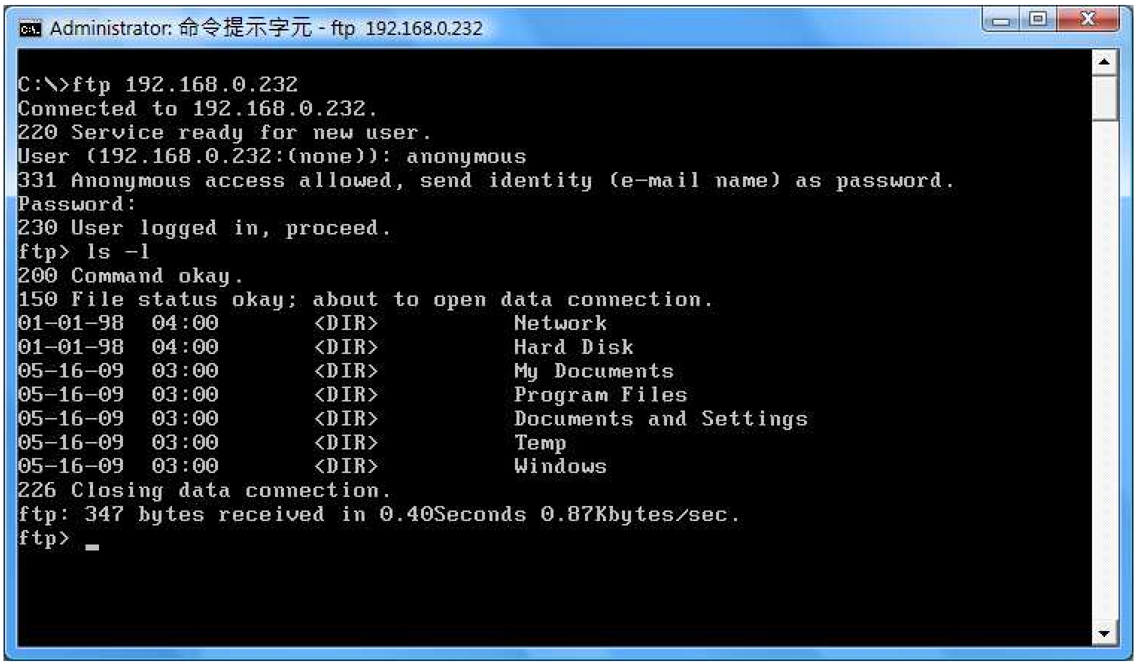
Build options:
- ☐ Buffer tracked events in RAM (IMGOSCAPTURE=1)
- ☐ Enable eboot space in memory (IMGEBOOT=1)
- ☐ Enable event tracking during boot (IMGCELOGENABLE=1)
- ☐ Enable hardware-assisted debugging support (IMGHDSTUB=1)
- ☐ Enable kernel debugger (no IMGNODEBUGGER=1)
- ☐ Enable KITL (no IMGNOKITL=1)
- ☐ Enable profiling (IMGPROFILER=1)
- ☐ Flush tracked events to release directory (IMGAUTOFLUSH=1)
- ☐ Run-time image can be larger than 32 MB (IMGRAM64=1)
- ☐ Use xcopy instead of links to populate release directory (BUILDREL_USE_COPY=1)
- ☐ Write run-time image to flash memory (IMGFLASH=1)

When settings are ready, build new Windows CE image. Boot is from USB or DOM for test in next section.

Test

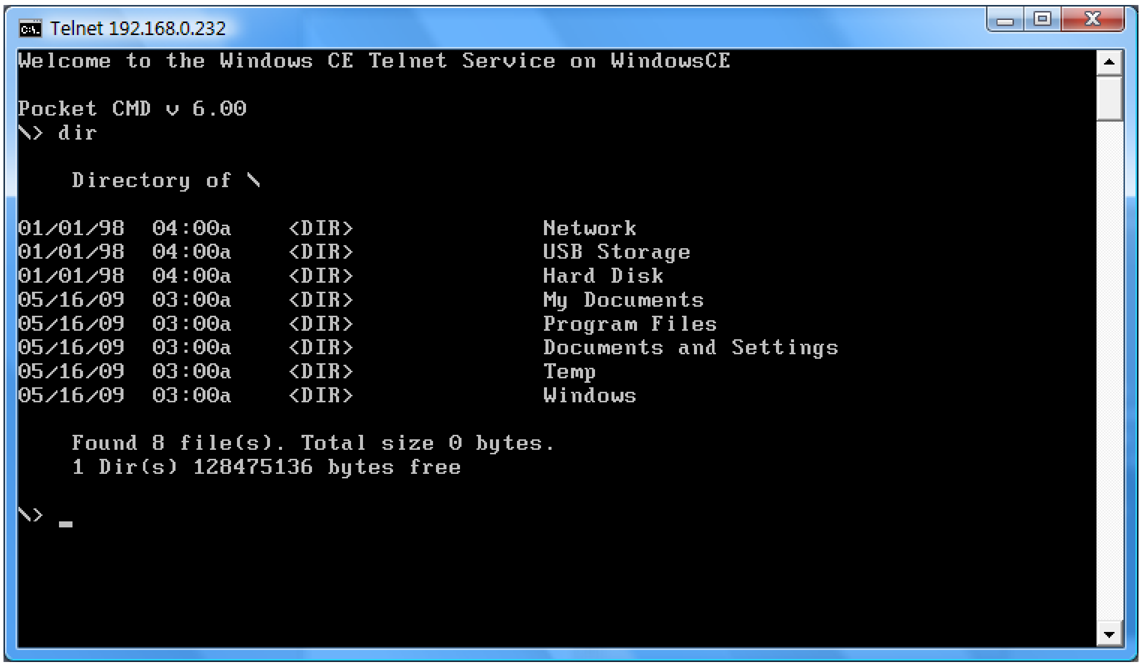
After Windows CE is loaded, our demo program will run automatically. Check serial port output to ensure demo program is running.

We use FTP client in Windows command prompt to demo. Developers can use their favorite FTP client to do files exchange between Windows CE device and desktop computer. The IP of Windows CE is 192.168.0.232. Use user name "anonymouse" and password "ftp" to login FTP server on Windows CE. Here is the FTP test screen:



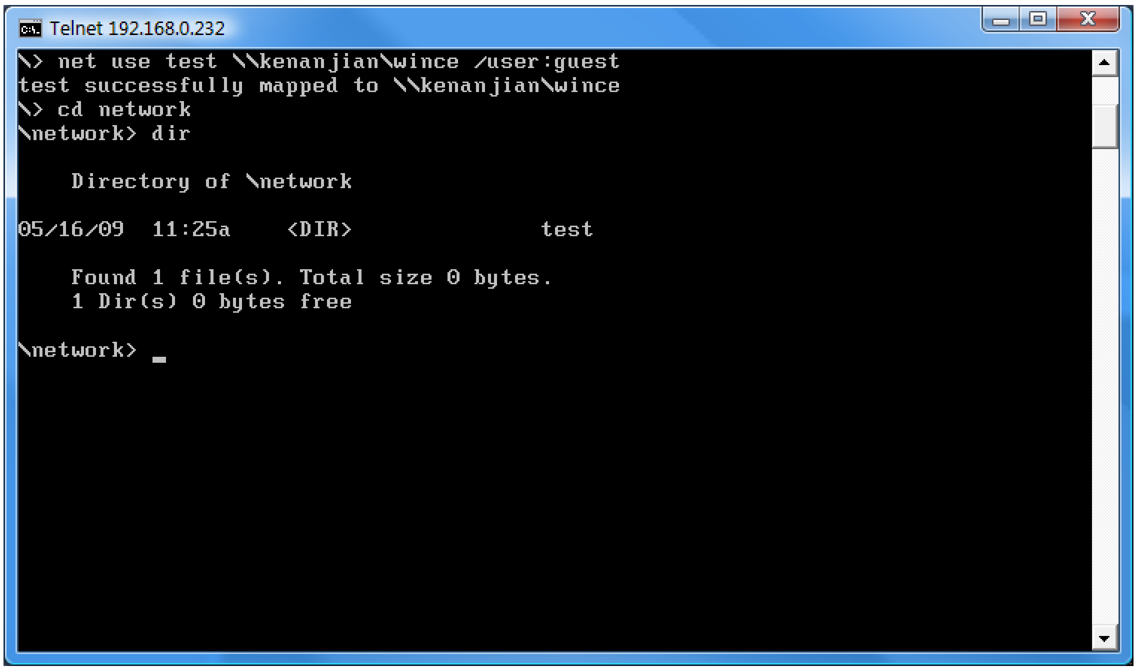
```
C:\>ftp 192.168.0.232
Connected to 192.168.0.232.
220 Service ready for new user.
User (192.168.0.232:(none)): anonymous
331 Anonymous access allowed, send identity (e-mail name) as password.
Password:
230 User logged in, proceed.
ftp> ls -l
200 Command okay.
150 File status okay; about to open data connection.
01-01-98  04:00       <DIR>          Network
01-01-98  04:00       <DIR>          Hard Disk
05-16-09  03:00       <DIR>          My Documents
05-16-09  03:00       <DIR>          Program Files
05-16-09  03:00       <DIR>          Documents and Settings
05-16-09  03:00       <DIR>          Temp
05-16-09  03:00       <DIR>          Windows
226 Closing data connection.
ftp: 347 bytes received in 0.40Seconds 0.87Kbytes/sec.
ftp> _
```

The folder "USB Storage" is USB pen drive and "Hard Disk" is DOM on Vortex86DX board. Below picture is the screen using telnet to connect to Windows CE device:



```
Welcome to the Windows CE Telnet Service on WindowsCE

Pocket CMD v 6.00
\> dir

    Directory of \

01/01/98  04:00a    <DIR>              Network
01/01/98  04:00a    <DIR>              USB Storage
01/01/98  04:00a    <DIR>              Hard Disk
05/16/09  03:00a    <DIR>              My Documents
05/16/09  03:00a    <DIR>              Program Files
05/16/09  03:00a    <DIR>              Documents and Settings
05/16/09  03:00a    <DIR>              Temp
05/16/09  03:00a    <DIR>              Windows

    Found 8 file(s). Total size 0 bytes.
    1 Dir(s) 128475136 bytes free

\> _
```

Developers can run program or some jobs via telnet. Net command is used via telnet to map share folder. Run "net" to map share folder on desktop PC to Windows CE device:



```
\> net use test \\kenanjian\wince /user:guest
test successfully mapped to \\kenanjian\wince
\> cd network
\network> dir

    Directory of \network

05/16/09  11:25a    <DIR>              test

    Found 1 file(s). Total size 0 bytes.
    1 Dir(s) 0 bytes free

\network> _
```

Folder “wince” on desktop Windows XP/Vista PC will be mapped to “test” folder in “Network” folder of Windows CE root file system. Just change work directory to “Network” and run dir command, you can find test folder. If full access right is enabled, Windows CE device can read or write files in folder test.

Note:

1. As our test, only domain name “kenanjian” can work and IP address “192.168.X.X” can not work. Recommend using domain when you test SMB function.
2. Windows CE device also can be SMB server. Search help of Platform Builder to add registry settings. Here is example to share hard disk for your reference:

```
[HKEY_LOCAL_MACHINE\Services\Smbserver]
  "AdapterList"="*"
  "dll"="smbserver.dll"
  "Keep"=dword:1
  "Order"=dword:9

[HKEY_LOCAL_MACHINE\Services\SMBServer\Shares]
  "UseAuthentication"=dword:0

[HKEY_LOCAL_MACHINE\Services\SMBServer\Shares\HDD]
  "Path"="\\Hard Disk"
  "Type"=dword:0
  "UserList"="@*;"
```

6. Windows Embedded Standard (Windows XP Embedded)

Windows XP embedded (or XPe) is componentized Windows XP Pro. It is renamed to Windows Embedded Standard on 2009. Current version is Windows Ebedded Standard 2009 and next version is Windows Embedded Standard 2011 with Windows 7 core.

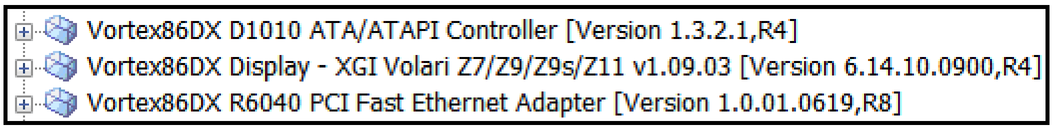
Visit http://www.microsoft.com/windowsembedded/en-us/default.mspx form Micrsoft web site to know more.

Get more resource from MSDN at http://msdn.microsoft.com/en-us/windowsembedded/standard/default.aspx.

6.1. Drivers and Evalaution Images

Vortex86SX without FPU can not boot XPe. Vorex86DX/MX can boot XPe and work well. For Vortex86DX, there are three components for Target Designer to generate XPe image for Vortex86DX SoC: IDE, Ethernet and VGA. Here are easy steps for reference:

- Download those 3 SLX file and use Component Database Manager to import them.
- Download Vortex86DX PMQ file and using Target Designer to import.
- If step 1 & 2 are okay, you will see Vortex86DX drivers are added into your design. Or, search "Vortex86DX" to find those 3 drivers to add them manually.


Vortex86DX D1010 ATA/ATAPI Controller [Version 1.3.2.1,R4]
Vortex86DX Display - XGI Volari Z7/Z9/Z9s/Z11 v1.09.03 [Version 6.14.10.0900,R4]
Vortex86DX R6040 PCI Fast Ethernet Adapter [Version 1.0.01.0619,R8]

The version of components is example only. They will be keeping update. Develpoers can download PMQ file from our technical support web site to start XPe development: http://www.dmp.com.tw/tech/vortex86dx/#xpe.

On the driver download link above, there are also some evaluation images for developers test. Download them and flollow steps in next section to boot. Evalaution image will get blue screen after 120~180 days.

6.2. Boot XPe

Boot from HDD/DOM

Before copying files onto DOM to boot Windows Embedded Standard, you have to make primary partition and set it as “active”. If your DOM is formatted with NTFS, just set it as primary partition and extract ZIP file onto your DOM. If your DOM is formatted with FAT32, here are steps to install boot loader:

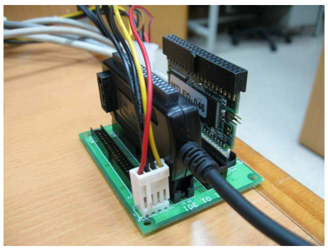
1. Boot from USB (refer to http://www.google.com.tw/search?q=usb+boot+hp+tool&meta=&aq=f&oq=).
2. Run bootprep.exe from USB in DOS. This file is at “C:\Program Files\Windows Embedded\utilities”.
3. Assume your DOM is C: in DOS, run “bootprep /dc”. Search bootprep in help for more detail.

Boot from USB mass storage

Plug your USB mass storage and run “C:\Program Files\Windows Embedded\utilities\UFDPrep.exe <Drive Letter>”. Search “UFDPrep.exe” in help for more detail.

Copy Files to HDD/DOM

We do not recommend booting DOS from USB to copy Windows Embedded Standard files. This is because long file name will lost in DOS copy procedure. The fast and easy way is to use ICOP-0094 + IDE/SATA-to-USB cable (see right picture). Copy files onto your HDD/DOM as USB pen drive.



Uisng FTP Server in Windows Embedded CE

The other way to transfer Windows Embedded Standard files without losing long file name is to build a Windows Embedded CE image with FTP server. Boot from USB and use loadcepc.exe to run Windows Embedded CE image to enable FTP server function. Using FTP client program in desktop PC to upload files will take more time than using IDE/SATA-to-USB cable.

6.3. HORM

Some of our XPe evaluation images support HROM. Vortex86DX board can boot Windows Embedded Standard within 11~38 seconds when HORM is enabled. Run commands in command prompt to make HORM work:

1. Boot image until FBA finish.
2. Enable hibernate in “Control Panel -> Power Options -> Hibernate -> Enable hibernation”.
3. Run “ewfmgr c: -enable”.
4. Run “xpepm –restart” (or reboot from start menu).
5. Run “ewfmgr c: -activatehorm”.
6. Run and configure your application.
7. Run “xpepm –hibernate” (or hibernate from start menu).
8. Done.

After power on, VDX board will boot into Windows Embedded Standard and programs opened in step 6 will run. To disable HORM, run “ewfmgr c: -deactivatehorm”.

If your RAM size on board is 256Mbytes, it need the same disk space form hibernate file. For using FAT32 file system, it maybe need more disk space for cluster size issue.

6.4. Windows Embedded Standard 2011

As our test with Windows Embedded Standard 2011 CTP, Vortex86DX can not boot properly. This is because it needs ACPI function to boot. We are modifying BIOS for it now.

6.5. License Key

The Windows XP Embedded license key is different with Windows Embedded Standard 2009. If XPe key is used on WES2009 image, you will get blue screen of death (BSOD).

7. Software Development Tips

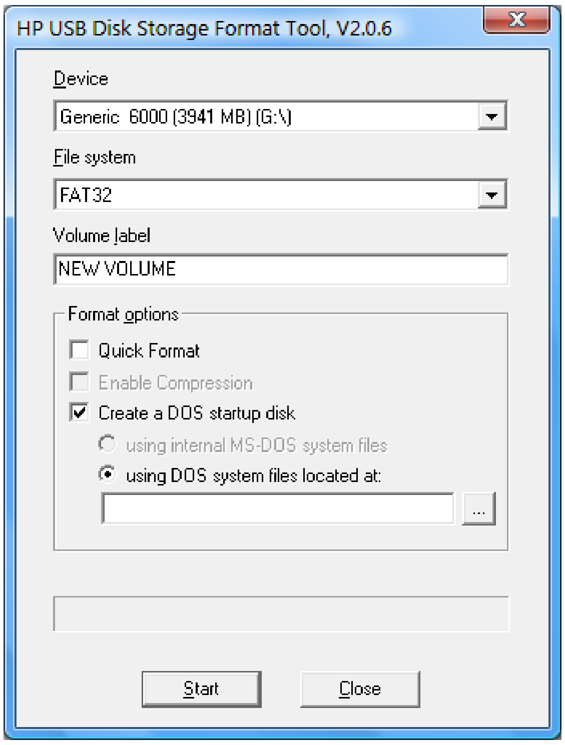
We will introducr some tips for programmers. They are from FAQ and technical support e-mail. If you have any good idea, mail to soc@dmp.com.tw to add.

7.1. USB Boot

Software programmers will use floppy to transfer software onto target x86 board many years ago. We can use USB boot for that more easily now. Steps in this section will be helpful for program transmition.

HP USB Boot Utility

Search "HP USB Boot Utility" from Google and you can get download link to get this. It is great tools to make USB DOS bootable. For DOS boot file, you can use MS-DOS or FreeDOS. Here is the screen from the tool:



HP USB Disk Storage Format Tool, V2.0.6

Device: Generic 6000 (3941 MB) (G:\)

File system: FAT32

Volume label: NEW VOLUME

Format options

- [ ] Quick Format
- [ ] Enable Compression
- [x] Create a DOS startup disk
  - ○ using internal MS-DOS system files
  - ● using DOS system files located at:

Select your DOS system path and press start button to make USB pen drive DOS bootable.

makebootfat

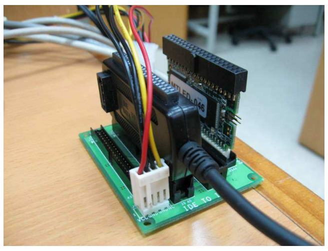
Download makebootfat from http://sourceforge.net/projects/advancemame/files/. You also can download it from our technical support web page: ftp://download@ftp.dmp.com.tw/vortex86sx/make_boot_fat.zip. Our download includes FreeDOS and just run make.bat to make USB FreeDOS bootable. If you USB mass storage can not boot properly, run make_lba.bat to try.

Boot O/S

After those steps, you USB pen drive can boot into MS-DOS or FreeDOS. You also can download X-Linux RAM image version to run it by loadlin.exe. Or, using loadcepc.exe to load nk.bin to boot Windows CE.

7.2. ICOP-0094

ICOP-0094 IDE exchanger kit also can help programmers to copy files onto 44/40-pins DOM flash disk. Recommend to buy/get IDE-to-USB cable to use it with ICOP-0094. Access DOM will be very easy in this case.



8. Technical Reference

Developers need to sign NDA to get Vortex86 series SoC CPU data sheet. For some example in this manul, we include relative registers information here for easy reference. To get NDA, please mail to soc@dmp.com.tw.

8.1. PCI Configuration Registers

I/O Port: CF8h ─ Accessed as a Dword

Register Name: PCI Configuration Address Register

Reset Value: 00000000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CE	Rsvd							BN								DN					FN			RN						Rsvd	

Configuration Address Register is a 32-bit register accessed only when referenced as a DWORD. A byte or word reference will pass through the Configuration Address Register onto the PCI bus as an I/O cycle.

Bit	Name	Attribute	Description
31	CE	R/W	Configuration Enable. When this bit is set to 1, accesses to PCI configuration space are enabled. If this bit is reset to 0, accesses to PCI configuration space are disabled.
30-24	Rsvd	RO	Reserved.
23-16	BN	R/W	Bus Number. When the bus number is programmed to 00h, the target of the configuration cycle is either the North-Bridge or the PCI Device that is connected to the North-Bridge. If the bus number is programmed to 00h and the North-Bridge is not the target, a Type 0 configuration cycle is generated on PCI Bus. IF the bus number is non-zero, a Type 1 configuration cycle is generated on PCI bus with the bus number mapped to AD[23:16] during the address phase.
15-11	DN	R/W	Device Number. This field selects one agent on the PCI bus. During a Type 1 configuration cycle, this field is mapped to AD[15:11]. During a Type 0 configuration cycle, this field is decoded and one of AD[31:11] is driven to 1.
10-8	FN	R/W	Function Number. This field allows the configuration registers of a particular function in a multi-function device to be accessed. The Vortex86DX North Bridge only responds to configuration cycle with a function number of 000b.
7-2	RN	R/W	Register Number. This field selects one register within a particular Bus, Device, and Function as specified by the other fields in the Configuration Address Register.
1-0	Rsvd	RO	Reserved.

I/O Port: CFCh ─ Accessed as a Dword

Register Name: PCI Configuration Data Register

Reset Value: 00000000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CDR																															

Configuration Data Register is a 32-bit read/write window into configuration space. The portion of configuration space that is referenced by Configuration Data Register is determined by the contents of Configuration Address Register.

Bit	Name	Attribute	Description
31-0	CDR	R/W	If bit 31 of PCI Configuration Address Register is 1, any I/O reference that falls in the PCI Configuration Data Register space is mapped to configuration space using the contents of PCI Configuration Address Register.

8.2. GPIO Interrupt Relative Registers

Register Offset: DCh
Register Name: GPIO PORT0 Interrupt Mask Register
Reset Value: 00h

7	6	5	4	3	2	1	0
P0INTM							

Bit	Name	Attribute	Description
7-0	P0INTM	R/W	GPIO PORT0 Interrupt Mask Register: This mask register is workable when Port0[x] is at input or output mode. If Port0[x] is at output mode and interrupt level set as high, the interrupt will occur base on the GPIO_PORT0 interrupt control register. Bit0 for Port0[0], Bit1 for Port0[1], …, Bit7 for Port0[7] 1: Enable Interrupt happen 0: Disable interrupt

Register Offset: DDh
Register Name: GPIO PORT0 Interrupt Level Register
Reset Value: FFh

7	6	5	4	3	2	1	0
P0INTL							

Bit	Name	Attribute	Description
7-0	P0INTL	R/W	GPIO PORT0 Interrupt Level Register Bit0 for Port0[0], Bit1 for Port0[1], …, Bit7 for Port0[7] 1: Interrupt activated on Port0 low level 0: Interrupt activated on Port0 high level

Register Offset: DEh
Register Name: GPIO PORT0 Interrupt Control Register
Reset Value: 00h

7	6	5	4	3	2	1	0
En	IKP			P0INTR			

Bit	Name	Attribute	Description
7	En	R/W	GPIO PORT0 Interrupt Function Enable bit. 0: Disable (Default)

Bit	Name	Attribute	Description
			1: Enable
6-4	IKP	R/W	Interrupt Keep Period. Interrupt will be generated while the event loading time of any one of port0[7-0] is longer than the following time parameters . Reference 14.318MHz 000: 002ms 001: 005ms 010: 010ms 011: 020ms 100: 040ms 101: 060ms 110: 080ms 111: 100ms
3-0	P0INTR	R/W	GPIO PORT0 Interrupt Routing Register Bit 11 Bit 10 Bit 9 Bit 8 Routing Table 0 0 0 0 Disable. 0 0 0 1 IRQ[9] 0 0 1 0 IRQ[3] 0 0 1 1 IRQ[10] 0 1 0 0 IRQ[4] 0 1 0 1 IRQ[5] 0 1 1 0 IRQ[7] 0 1 1 1 IRQ[6] 1 0 0 0 IRQ[1] 1 0 0 1 IRQ[11] 1 0 1 0 Reserved 1 0 1 1 IRQ[12] 1 1 0 0 Reserved 1 1 0 1 IRQ[14] 1 1 1 0 Reserved 1 1 1 1 IRQ[15]

Register Offset: DFh

Register Name: GPIO PORT0 Interrupt Mode Control Register

Reset Value: 00h

7	6	5	4	3	2	1	0
P0INTCTL							

Bit	Name	Attribute	Description
7-0	P0INTCTL	R/W	GPIO PORT0 Interrupt Control Register

			Bit0 for Port0[0], Bit1 for Port0[1], …, Bit7 for Port0[7] 1: trigger the interrupt continuously if level is activated and match interrupt keep period settings. 0: Trigger the Interrupt once if level is activated.

Register Offset: E0h
Register Name: GPIO PORT1 Interrupt Mask Register
Reset Value: 00h

7	6	5	4	3	2	1	0
P1INTM							

Bit	Name	Attribute	Description
7-0	P INTM	R/W	GPIO PORT1 Interrupt Mask Register: This mask register is workable when Port1[x] is at input or output mode. If Port1[x] is at output mode and interrupt level set to high, the interrupt will occur base on the GPIO_PORT1 interrupt control register. Bit0 for Port1[0], Bit1 for Port1[1], …, Bit7 for Port1[7] 1: Enable Interrupt happen 0: Disable interrupt

Register Offset: E1h
Register Name: GPIO PORT1 Interrupt Level Register
Reset Value: FFh

7	6	5	4	3	2	1	0
P1INTL							

Bit	Name	Attribute	Description
7-0	P1INTL	R/W	GPIO PORT1 Interrupt Level Register Bit0 for Port1[0], Bit1 for Port1[1], …, Bit7 for Port1[7] 1: Interrupt activated on Port1 low level 0: Interrupt activated on Port1 high level

Register Offset: E2h
Register Name: GPIO PORT1 Interrupt Control Register
Reset Value: 00h

7	6	5	4	3	2	1	0
En	IKP			P1INTR			

Bit	Name	Attribute	Description
7	En	R/W	GPIO PORT1 Interrupt Function Enable bit. 0: Disable (Default) 1: Enable
6-4	IKP	R/W	Interrupt Keep Period. Interrupt will be generated while the event loading time of any one of port0[7-0] is longer than the following time parameters . Reference 14.318MHz 000: 002ms 001: 005ms 010: 010ms 011: 020ms 100: 040ms 101: 060ms 110: 080ms 111: 100ms
3-0	P1INTR	R/W	GPIO PORT1 Interrupt Routing Register Bit 11 Bit 10 Bit 9 Bit 8 Routing Table 0 0 0 0 Disable. 0 0 0 1 IRQ[9] 0 0 1 0 IRQ[3] 0 0 1 1 IRQ[10] 0 1 0 0 IRQ[4] 0 1 0 1 IRQ[5] 0 1 1 0 IRQ[7] 0 1 1 1 IRQ[6] 1 0 0 0 IRQ[1] 1 0 0 1 IRQ[11] 1 0 1 0 Reserved 1 0 1 1 IRQ[12] 1 1 0 0 Reserved 1 1 0 1 IRQ[14] 1 1 1 0 Reserved 1 1 1 1 IRQ[15]

Register Offset: E3h

Register Name: GPIO PORT1 Interrupt Mode Control Register

Reset Value: 00h

7	6	5	4	3	2	1	0
P1INTCTL							

Bit	Name	Attribute	Description
7-0	P1INTCTL	R/W	GPIO PORT1 Interrupt Control Register Bit0 for Port1[0], Bit1 for Port1[1], …, Bit7 for Port1[7] 1: trigger the interrupt continuously if level is activated and match interrupt keep period settings. 0: Trigger the Interrupt once if level is activated.

8.3. Servo Registers

SERVO Registers

(**B**ase **A**ddress Refers to the Register of index D3h-D0h, IDSEL = AD18/SB of PCI Configuration Register)

IO Address	**Register Name**
BA + 00H	SERVO Interrupt Mask Register
BA + 04H	SERVO Interrupt Status Register
BA + 08H	SERVO Sync Register
BA + 0CH	SERVO[0] Pulse Low Count Register
BA + 10H	SERVO[0] Pulse High Count Register
BA + 14H	SERVO[0] Control Register
BA + 18H	SERVO[1] Pulse Low Count Register
BA + 1CH	SERVO[1] Pulse High Count Register
BA + 20H	SERVO[1] Control Register
BA + 24H	SERVO[2] Pulse Low Count Register
BA + 28H	SERVO[2] Pulse High Count Register
BA + 2CH	SERVO[2] Control Register
BA + 30H	SERVO[3] Pulse Low Count Register
BA + 34H	SERVO[3] Pulse High Count Register
BA + 38H	SERVO[3] Control Register
BA + 3CH	SERVO[4] Pulse Low Count Register
BA + 40H	SERVO[4] Pulse High Count Register
BA + 44H	SERVO[4] Control Register
BA + 48H	SERVO[5] Pulse Low Count Register
BA + 4CH	SERVO[5] Pulse High Count Register
BA + 50H	SERVO[5] Control Register
BA + 54H	SERVO[6] Pulse Low Count Register
BA + 58H	SERVO[6] Pulse High Count Register
BA + 5CH	SERVO[6] Control Register
BA + 60H	SERVO[7] Pulse Low Count Register
BA + 64H	SERVO[7] Pulse High Count Register
BA + 68H	SERVO[7] Control Register
BA + 6CH	SERVO[8] Pulse Low Count Register
BA + 70H	SERVO[8] Pulse High Count Register
BA + 74H	SERVO[8] Control Register
BA + 78H	SERVO[9] Pulse Low Count Register
BA + 7CH	SERVO[9] Pulse High Count Register
BA + 80H	SERVO[9] Control Register
BA + 84H	SERVO[10] Pulse Low Count Register

IO Address	Register Name
BA + 88H	SERVO[10] Pulse High Count Register
BA + 8CH	SERVO[10] Control Register
BA + 90H	SERVO[11] Pulse Low Count Register
BA + 94H	SERVO[11] Pulse High Count Register
BA + 98H	SERVO[11] Control Register
BA + 9CH	SERVO[12] Pulse Low Count Register
BA + A0H	SERVO[12] Pulse High Count Register
BA + A4H	SERVO[12] Control Register
BA + A8H	SERVO[13] Pulse Low Count Register
BA + ACH	SERVO[13] Pulse High Count Register
BA + B0H	SERVO[13] Control Register
BA + B4H	SERVO[14] Pulse Low Count Register
BA + B8H	SERVO[14] Pulse High Count Register
BA + BCH	SERVO[14] Control Register
BA + C0H	SERVO[15] Pulse Low Count Register
BA + C4H	SERVO[15] Pulse High Count Register
BA + C8H	SERVO[15] Control Register
BA + CCH	SERVO[16] Pulse Low Count Register
BA + D0H	SERVO[16] Pulse High Count Register
BA + D4H	SERVO[16] Control Register
BA + D8H	SERVO[17] Pulse Low Count Register
BA + DCH	SERVO[17] Pulse High Count Register
BA + E0H	SERVO[17] Control Register
BA + E4H	SERVO[18] Pulse Low Count Register
BA + E8H	SERVO[18] Pulse High Count Register
BA + ECH	SERVO[18] Control Register
BA + F0H	SERVO[19] Pulse Low Count Register
BA + F4H	SERVO[19] Pulse High Count Register
BA + F8H	SERVO[19] Control Register
BA + FCH	SERVO[20] Pulse Low Count Register
BA + 100H	SERVO[20] Pulse High Count Register
BA + 104H	SERVO[20] Control Register
BA + 108H	SERVO[21] Pulse Low Count Register
BA + 10CH	SERVO[21] Pulse High Count Register
BA + 110H	SERVO[21] Control Register
BA + 114H	SERVO[22] Pulse Low Count Register
BA + 118H	SERVO[22] Pulse High Count Register
BA + 11CH	SERVO[22] Control Register
BA + 120H	SERVO[23] Pulse Low Count Register

IO Address	Register Name
BA + 124H	SERVO[23] Pulse High Count Register
BA + 128H	SERVO[23] Control Register
BA + 12CH	SERVO[24] Pulse Low Count Register
BA + 130H	SERVO[24] Pulse High Count Register
BA + 134H	SERVO[24] Control Register
BA + 138H	SERVO[25] Pulse Low Count Register
BA + 13CH	SERVO[25] Pulse High Count Register
BA + 140H	SERVO[25] Control Register
BA + 144H	SERVO[26] Pulse Low Count Register
BA + 148H	SERVO[26] Pulse High Count Register
BA + 14CH	SERVO[26] Control Register
BA + 150H	SERVO[27] Pulse Low Count Register
BA + 154H	SERVO[27] Pulse High Count Register
BA + 158H	SERVO[27] Control Register
BA + 15CH	SERVO[28] Pulse Low Count Register
BA + 160H	SERVO[28] Pulse High Count Register
BA + 164H	SERVO[28] Control Register
BA + 168H	SERVO[29] Pulse Low Count Register
BA + 16CH	SERVO[29] Pulse High Count Register
BA + 170H	SERVO[29] Control Register
BA + 174H	SERVO[30] Pulse Low Count Register
BA + 178H	SERVO[30] Pulse High Count Register
BA + 17CH	SERVO[30] Control Register
BA + 180H	SERVO[31] Pulse Low Count Register
BA + 184H	SERVO[31] Pulse High Count Register
BA + 188H	SERVO[31] Control Register

I/O Port:	BA + 00h
Register Name:	SERVO Interrupt Mask Register
Reset Value:	00000000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SIM[31-0]																															

Bit	Name	Attribute	Description
31-0	SIM[31-0]	R/W	SERVO[31-0] Interrupt Mask Register 1: Enable Interrupt 0: Disable Interrupt

I/O Port: BA + 04h

Register Name: SERVO Interrupt Status Register

Reset Value: 00000000h

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

SIS[31-0]

Bit	Name	Attribute	Description
31-0	SIS[31-0]	R/W	SERVO[31-0] Interrupt Status Register 1: Interrupt happen and write “1” to clear 0: No Interrupt

I/O Port: BA + 08h

Register Name: SERVO Sync Status Register

Reset Value: 00000000h

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

SYNC[31-0]

Bit	Name	Attribute	Description
31-0	SYNC[31-0]	R/W	SERVO[31-0] Sync Register 1: SERVO will be hold 0: SERVO without hold

I/O Port: BA + 0CH, 18H, 24H, 30H, 3CH, 48H, 54H, 60H, 6CH, 78H, 84H, 90H, 9CH, A8H, B4H, C0H, CCH, D8H, E4H, F0H, FCH, 108H, 114H, 120H, 12CH, 138H, 144H, 150H, 15CH, 168H, 174H, 180H

Register Name: SERVO Pulse Low Register

Reset Value: 00000000h

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

SPL

Bit	Name	Attribute	Description
31-0	SPL	R/W	SERVO Pulse Low Register. SERVO clock is 10MHz

I/O Port: BA + 10H, 1CH, 28H, 34H, 40H, 4CH, 58H, 64H, 70H, 7CH, 88H, 94H, A0H, ACH, B8H, C4H, D0H, DCH, E8H, F4H,100H,10CH,118H,124H,130H,13CH,148H,154H,160H,16CH,178H,184H

Register Name: SERVO Pulse High Register

Reset Value: 00000000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPH																															

Bit	**Name**	**Attribute**	**Description**
31-0	SPH	R/W	SERVO Pulse High Register. SERVO clock is 10MHz

I/O Port: BA + 14H, 20H, 2CH, 38H, 44H, 50H, 5CH, 68H, 74H, 80H, 8CH, 98H, A4H, B0H, BCH, C8H, D4H, E0H, ECH, F8H, 104H, 110H, 11CH, 128H, 134H, 140H, 14CH, 158H, 164H, 170H, 17CH, 188H

Register Name: SERVO Control Register

Reset Value: 00000000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SE	CM	INVS	Rsvd	RC																											

Bit	**Name**	**Attribute**	**Description**
31	SE	R/W	SERVOx Enable Control 1: SERVOx enable 0: SERVOx disable
30	CM	R/W	SERVOx Continuous Mode 1: SERVOx Continuous Mode enable 0: SERVOx Continuous Mode disable
29	INVS	R/W	Inverse SERVO signal 0: default SERVO out ‘0’, SPH specify ‘I’, SPL specify “0”. 1: Inverse output signal of upper case
28	Rsvd	RO	Reserved
27-0	RC	R/W	SERVOx Repeat Count. It is used when CM=0.

Vortex86DX South Bridge Configuration Registers

Register Offset: 01h – 00h
Register Name: Vendor ID Register
Reset Value: 17F3h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
VID															

Bit	Name	Attribute	Description
15-0	VID	RO	This register contains a 16-bit value assigned to South Bridge Vendor ID.

Register Offset: 03h – 02h
Register Name: Device ID Register
Reset Value: 6031h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DID															

Bit	Name	Attribute	Description
15-0	DID	RO	This register contains a 16-bit value to specify a particular device.

Register Offset: 4Bh – 48h
Register Name: Buffer Strength/Clock Output Control Register
Reset Value: 3FFF3600h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Rvd		GPIO4_DRV		GPIO3_3_DRV		GPIO3_2_DRV		GPIO3_1_DRV		GPIO2_DRV		GPIO1_DRV		GPIO0_DRV		Rvd	PCI_CTL_SR	PCI_CTL_DRV		PCI_DT_SR	PCI_DT_DRV		Rvd					ISACK_CTL	14M_CTL	24M_CTL	25M_CTL

Bit	Name	Attribute	Description
31-30	Rvd	RO	Reserved
29-28	GPIO4_DRV	RW	GPIO4[7-0]/SERVO[31-24]/COM1 Driving Current Control 00: 4mA 01: 8mA 10: 12mA 11: 16mA (default)
21-20	GPIO2_DRV	RW	GPIO2[7-0]/SERVO[23-16]/SA[31-24] Driving Current Control

Bit	Name	Attribute	Description
			00: 4mA 01: 8mA 10: 12mA 11: 16mA (default)
19-18	GPIO1_DRV	RW	GPIO1[7-0]/SERVO[15-8] Driving Current Control 00: 4mA 01: 8mA 10: 12mA 11: 16mA (default)
17-16	GPIO0_DRV	RW	GPIO0[7-0]/SERVO[7-0] Driving Current Control 00: 4mA 01: 8mA 10: 12mA 11: 16mA (default)

Register Offset: BFh – BCh

Register Name: On-Chip Device Control Register

Reset Value: 00000000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved									UDP	GSP	SVP	G1IP	G0IP	I2C1P	I2C0P	Reserved			PPP	Reserved		COM9P	COM4P	COM3P	COM2P	COM1P	Rsvd	MACP	USB2P	USB1P	IDEP

Bit	Name	Attribute	Description
20	SVP	R/W	ON-Chip SERVO power-down control 0: on-chip SERVO activate (default) 1: on-chip SERVO power-down

Register Offset: C3-C0h

Register Name: Internal Peripheral Feature Control Register

Reset Value: 032C0500h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Rsvd		IO16W			IO8W			MEM16W			MEM8W			ISACLK	Int_zw	Rsvd	IDEIS	PWM2	PWM1	PWM0	FCDC	MCLK	EMIQ	PINS6	SFCE	PINS5	PINS4	CPS	PINS2	PINS1	PINS0

Bit	Name	Attribute	Description
1	PINS1	R/W	PINS selection for COM1 and GPIO Port 4 0: 8 PINs for COM1(default) 1: 8 PINs for GPIO port 4
0	PINS0	R/W	PINS selection for External SPI and GPIO Port3 [3-0] 0: 4 pins for GPIO port3 [3-0] (default) 1: 4 pins for external SPI

Register Offset: CBh – C8h

Register Name: Internal Peripheral Feature Control Register II

Reset Value: 00000000h

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

GS[31-24]	GS[23-16]	GS[15-8]	GS[7-0]

Bit	Name	Attribute	Description
31-24	GS[31-24]	R/W	GPIO_P4[7-0] and SERVO[31-24] selection. This register is used only when SB register C0h bit1 is “1”. 0: PINS for GPIO_P4 (default) 1: PINS for SERVO
23 – 16	GS[23-16]	R/W	GPIO_P2[7-0] and SERVO[23-16] selection. This register is used only when STRAP[1] (NB register 60h bit19) is “1”. 0: PINS for GPIO_P2 (default) 1: PINS for SERVO
15 – 8	GS[15-8]	R/W	GPIO_P1[7-0] and SERVO[15-8] selection. 0: PINS for GPIO_P1 (default) 1: PINS for SERVO
7 – 0	GS[7-0]	R/W	GPIO_P0[7-0] and SERVO[7-0] selection. 0: PINS for GPIO_P0 (default) 1: PINS for SERVO

Register Offset: D3h – D0h

Register Name: Internal SERVO Control Register

Reset Value: 00000000h

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Reserved	CLKS	UE	Rsvd	SIRT	UIOA	Reserved

Bit	Name	Attribute	Description
31-25	Rsvd	RO	Reserved
24	CLKS	R/W	Servo Clock selection 0: 10MHz (default) 1: 50MHz
23	UE	R/W	Enable/Disable Internal SERVO IO Address Decode 0: Disable (Default) 1: Enable
22-20	Rsvd	RO	Reserved
19-16	SIRT	R/W	SERVO IRQ Routing Table Bit19 Bit18 Bit17 Bit16 Routing Table 0 0 0 0 Disable. 0 0 0 1 IRQ[9] 0 0 1 0 IRQ[3] 0 0 1 1 IRQ[10] 0 1 0 0 IRQ[4] 0 1 0 1 IRQ[5] 0 1 1 0 IRQ[7] 0 1 1 1 IRQ[6] 1 0 0 0 IRQ[1] 1 0 0 1 IRQ[11] 1 0 1 0 Reserved 1 0 1 1 IRQ[12] 1 1 0 0 Reserved 1 1 0 1 IRQ[14] 1 1 1 0 Reserved 1 1 1 1 IRQ[15] These four bits are used to route SERVO IRQ to any 8259 Interrupt lines. The BIOS should be used to inhibit the setting of the reserved value.
15-9	UIOA	R/W	Internal SERVO IO Address. The Bit[15:9] contain the base IO address A[15:9] of internal SERVO.
8-0	Rsvd	RO	Reserved. All are ‘0’s. Writing any value to these bits causes no effect.

8.4. SPI Registers

(**B**ase **A**ddress Refers to the Register of index 43h-40h, IDSEL = AD11/NB of PCI Configuration Register)

IO Address	Register Name
BA + 00h	Flash SPI Output Data Register
BA + 01h	Flash SPI Input Data Register
BA + 02h	Flash SPI Control Register
BA + 03h	Flash SPI Status Register
BA + 04h	Flash SPI Chip-Select Register
BA + 05h	Flash SPI Error Status Register
BA + 08h	External SPI Output Data Register
BA + 09h	External SPI Input Register
BA + 0Ah	External SPI Control Register
BA + 0Bh	External SPI Status Register
BA + 0Ch	External SPI Chip Select Register
BA + 0Dh	External SPI Error Status Register

BASE_ADDR defined on NB PCI CFG 40h

Register Offset: BASE_ADDR+00h
Register Name: Flash SPI Output Data Register
Reset Value: --

7	6	5	4	3	2	1	0
OUTDAT							

Bit	Name	Attribute	Description
7-0	OUTDAT	WO	Data output to SPI when write. No function when read.

Register Offset: BASE_ADDR+01h
Register Name: Flash SPI Input Register
Reset Value: FFh

7	6	5	4	3	2	1	0
INDAT							

Bit	Name	Attribute	Description
7-1	INDAT	R/W	Data input from SPI when read. Preload data from SPI when write

Register Offset: BASE_ADDR+02h
Register Name: Flash SPI Control Register
Reset Value: 55h/5Ah (DRAM frequency <= 200MHz, >200 MHz)

It is not recommended to modify this register when SPI operation.

7	6	5	4	3	2	1	0
RSVD	FRE	AFDI S	FIEN	CKDIV			

Bit	Name	Attribute	Description
7	RSVD	RO	Reserved
6	FRE	R/W	Fast read enable(This bit can be set if it is NOT AMTEL flash, and Bit 5 must be 0)
5	AFDIS	R/W	0: Auto-fetch enable 1: Auto-fetch disable Reset to 0 if flash ROM write protect (default).
4	FIEN	R/W	FIFO mode enable when set.
3-0	CKDIV	R/W	SPI clock divided. The SPI clock is DRAM clock/(2 * SPI clock divided) , 0 is not allowed Under 45MHz is recommended for Vortex86DX internal SPI flash.

Register Offset: BASE_ADDR+03h
Register Name: Flash SPI Status Register
Reset Value: 10h

7	6	5	4	3	2	1	0
BUSY	FIFU	IDR	ODC	RSVD			

Bit	Name	Attribute	Description
7	BUSY	RO	SPI controller is BUSY.
6	FIFU	RO	FIFO full
5	IDR	RO	Input data ready when set.
4	ODC	RO	Output complete/FIFO empty when set.
3-0	RSVD	RO	Reserved

Register Offset: BASE_ADDR+04h
Register Name: Flash SPI Chip Select Register
Reset Value: 01h

7	6	5	4	3	2	1	0
Reserved							CS

Bit	Name	Attribute	Description
7-1	RSVD	RO	Reserved
0	CS	R/W	0: SPI CS# is low, 1: SPI CS# is high

Register Offset: BASE_ADDR+05h
Register Name: Flash SPI Error Status Register
Reset Value: 00h

7	6	5	4	3	2	1	0
RSVD			WCTE	DOLE	FIURE	FIORE	FHOPE

Bit	Name	Attribute	Description
7-5	RSVD	RO	Reserved
4	WCTE	R/WC	Error status 4, Write SPI Control Register when controller is busy. Write 1 to clear.
3	DOLE	R/WC	Error status3, Input data overlap. Write 1 to clear.
2	FIURE	R/WC	Error status2, FIFO under-run. Write 1 to clear.
1	FIORE	R/WC	Error status1, FIFO over-run. Write 1 to clear.
0	FHOPE	R/WC	Error status0, CPU fetch during SPI port operation. Write 1 to clear.

Register Offset: BASE_ADDR+08h
Register Name: External SPI Output Data Register
Reset Value: --

7	6	5	4	3	2	1	0
OUTDAT							

Bit	Name	Attribute	Description
7-0	OUTDAT	WO	Data output to SPI when write. No function when read.

Register Offset: BASE_ADDR+09h
Register Name: External SPI Input Register
Reset Value: FFh

7	6	5	4	3	2	1	0
INDAT							

Bit	Name	Attribute	Description

7-1	INDAT	RO	Data input from SPI when read. Preload data from SPI when write

Register Offset: BASE_ADDR+0Ah
Register Name: External SPI Control Register
Reset Value: 15h

7	6	5	4	3	2	1	0
RSVD			FIEN	CKDIV			

Bit	Name	Attribute	Description
7-5	RSVD	RO	Reserved
4	FIEN	R/W	FIFO mode enable when set.
3-0	CKDIV	R/W	SPI clock divided. The SPI clock is DRAM clock/(2 * SPI clock divided) , 0 is not allowed

Register Offset: BASE_ADDR+0Bh
Register Name: External SPI Status Register
Reset Value: 10h

7	6	5	4	3	2	1	0
BUSY	FIFU	IDR	ODC	RSVD			

Bit	Name	Attribute	Description
7	BUSY	RO	SPI controller is BUSY.
6	FIFU	RO	FIFO full
5	IDR	RO	Input data ready when set.
4	ODC	RO	Output complete/FIFO empty when set.
3-0	RSVD	RO	Reserved

Register Offset: BASE_ADDR+0Ch
Register Name: External SPI Chip Select Register
Reset Value: 01h

7	6	5	4	3	2	1	0
Reserved							CS

Bit	Name	Attribute	Description
7-1	RSVD	RO	Reserved
0	CS	R/W	0: SPI CS# is low, 1: SPI CS# is high

Register Offset: BASE_ADDR+0Dh

Register Name: External SPI Error Status Register

Reset Value: 00h

7	6	5	4	3	2	1	0
RSVD			WCTE	DOLE	FIURE	FIORE	RSVD

Bit	Name	Attribute	Description
7-5	RSVD	RO	Reserved
4	WCTE	R/WC	Error status 4, Write SPI Control Register when controller is busy. Write 1 to clear.
3	DOLE	R/WC	Error status3, Input data overlap. Write 1 to clear.
2	FIURE	R/WC	Error status2, FIFO under-run. Write 1 to clear.
1	FIORE	R/WC	Error status1, FIFO over-run. Write 1 to clear.
0	RSVD	RO	Reserved

8.5. More Technical Support

For more technical support, please visit our technical support web site at http://www.dmp.com.tw/tech or send e-mail to soc@dmp.com.tw.

More detail can save time and help our engineers to help you more easily. Here are some notes:

1. You can write down your **hardware and software environment**. For example:
   H/W: ICOP-6354 using Vortex86DX, BIOS version is A4 2008-12-20.
   S/W: Windows CE 6.0 R3
2. Description of "**what's your problem**". "I can not boot Windows CE, please help me" or "X-Linux can not work on my board, why?". Those two messages can not make our engineer to understand your problem. Here is a good example:
   I download your Windows CE evaluation image ftp://ftp.dmp.com.tw/vortex86dx/09083101.zip. Boot into FreeDOS and use loadcepc.exe to load it. After progress bar is finish, I get black screen and no reponse any more".
3. Write down **your steps for us to reproduce your problem**. Here is an example:
   I. Enter BIOS setup to load default settings. Just change IDE from legacy to native mode.
   II. Use CE image ftp://ftp.dmp.com.tw/vortex86dx/09083103.zip from your web site and x86 BIOS loader to boot Windows CE from DOM.
   III. Run my serial port loop test program in attachment (send us your test program in e-mail). It will send test pattern and receive for comparison. Any error found will stop program and show error message.
   IV. Open Internet Explorer to download large files (>10Mbytes).
   V. Serial port will lose data while download files from Internet.

DMP Electronics INC.

TEL:+886-2-22980770

FAX:+886-2-22991883

Email: info@dmp.com.tw

Address: 8F., No.12, Wucyuan 7th Rd., Wugu Township, Taipei County 248, Taiwan (R.O.C.)