DM&P Vortex86DX2

Software Programming Reference

Version 1.02

2016-08-02

This programming guide is for software programmers to write programs more quick and easy on Vortex86DX2. 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 / info@roboard.com. Please read section 8.2 before sending technical support request.

History

2016-08-02

- Fix WDT1 Reload address and general shift interface support to be 2 channel

2016-05-10

- Fix WDT1 base address.

2015-01-22

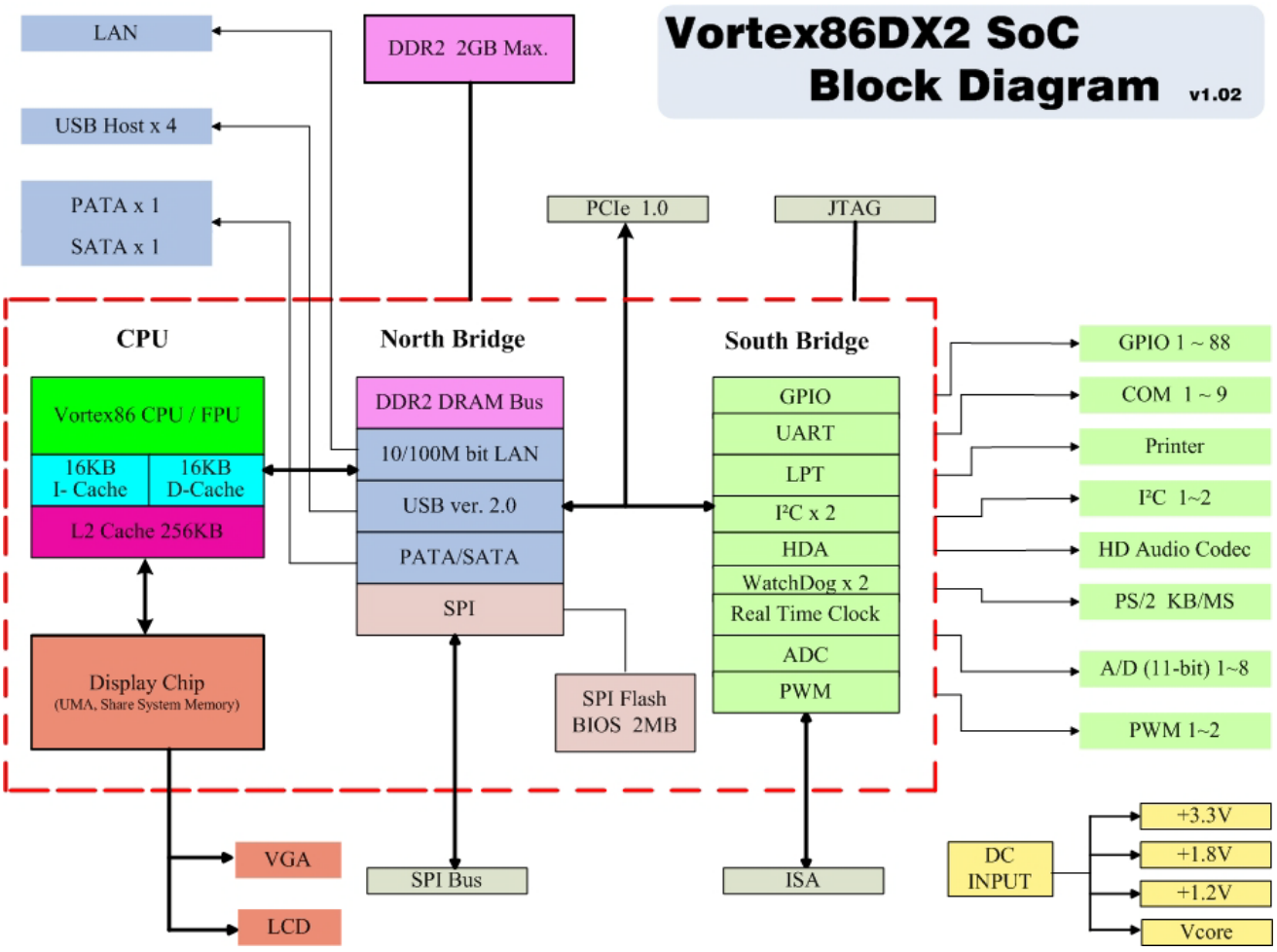
- Original version.

Table of Contents

1. Vortex86DX2 SoC Overview

The Vortex86DX2 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, 256KB write through/write back 4-way L2 cache, PCIE bus in at 2.5 GHz, DDR2, ROM controller, ISA, I2C, SPI, IPC (Internal Peripheral Controllers with DMA and interrupt timer/counter included), Fast Ethernet, FIFO UART, USB2.0 Host and IDE/SATA controller within a single 720-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. System Block Diagram



1.2. Feature

- **x86 32bit Processor Core**
  - 6 stage pipe-line
  - X86 instruction set
  - MMX instruction set
- **Floating point unit support**
  - Extends CPU instruction set to include Trigonometric, Logarithmic and Exponential
  - Implements ANSI/IEEE standard 754-1985 for binary Floating-Point Architecture
- **Branch prediction unit**
  - Branch target buffer
- **Translation Lookaside buffer**
  - 32 I/D translation lookaside buffer
- **Embedded I/D Separated L1 Cache**
  - 16K I-Cache, 16K D-Cache
- **Embedded L2 Cache**
  - 4-way 256KB L2 Cache
  - Write through or write back policy
- **DDRII Control Interface**
  - 32 bits data bus
  - DDRII clock support up to 366 MHz
  - DDRII size support up to 1 Gbytes
- **GPU Control Unit**
  - VGA controller
  - 2D Graphics engine support
  - UMA architecture
- **MAC Controller x 1**
- **Embedded 2MB Flash**
  - For BIOS storage
- **JTAG Interface supported for S.W. debugging**
- **IDE Controller**
  - PATA 100(HDD x 2) or SD x 2 at Primary Channel
  - SATA 1.5G (1 Port) at Secondary Channel
- **PCIE Control Interface x 2**
  - Up to 2 sets PCIE device
  - 3.3V I / O
- **USB 2.0 Host Support**
  - Supports HS, FS and LS
  - 4 port
- **USB 1.1 Device Support**
  - 1 port

- ☐ Supports FS with 3 programmable endpoint

- **HDA Controller**
- **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
  - ☐ Support Max ISA Clock 33M
- **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**
  - ☐ Less than 2.5uA (3.0V) power consumption in Internal RTC Mode while chip is power-off
- **PS / 2 Keyboard and Mouse Interface Support**
  - ☐ Compatible with 8042 controller
- **FIFO UART Port x 9 (9 sets COM Port)**
  - ☐ Compatible with 16C550 / 16C552
  - ☐ Default internal pull-up
  - ☐ Supports the programmable baud rate generator with the data rate from 50 to 6M 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 COM1/2/3/4
  - ☐ Port 80h output data could be sent to COM1 by software programming
- **Parallel Port x 1**
  - ☐ Supports SPP/EPP/ECP mode
- **Speaker out**
- **General Programmable I/O**
  - ☐ Supports 88 programmable I / O pins
  - ☐ Each GPIO pin can be individually configured to be an input/output pin
  - ☐ GPIO_P0~GPIO_P4 can be program by 8051A
  - ☐ GPIO_P0~GPIO_PA can be program by 8051B
  - ☐ GPIO_P0 and GPIO_P1 with interrupt support (input/output)
- **Redundant System Support**
- **I2C bus x 2**
  - ☐ Compliant w/t V2.1
  - ☐ Some master code (general call, START and CBUS) not support.
- **MTBF Counter**
- **ADC Interface x 8**
  - ☐ Effective Number of Bit=10 bits
- **Motion Control Interface Support**
  - ☐ 3 groups of controller, 4 controllers per group

- ☐ Each controller can configure to PWM/Servo/Sensor Interface mode
- ☐ Controller interconnect to the other with routing network in the same group
- ☐ 8051 Internal Motion Control Support

■ **General Shift Interface Support**

- ☐ 2 channel

■ **Full Duplex SPI bus x 2**

■ **Input clock**

- ☐ 14.318 MHz
- ☐ 32.768 KHz

■ **Output clock**

- ☐ 24 MHz
- ☐ 25 MHz
- ☐ DDRII clock

■ **Operating Voltage Range**

- ☐ Core voltage: 1.0 V ± 5%, 1.2 V ± 5%
- ☐ I / O voltage: 1.8V ± 5% , 3.3 V ± 10 %

■ **Operating temperature**

- ☐ -40℃ ~ 85℃

■ **Package Type**

- ☐ 31x31mm, 720 Ball PBGA

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 Embedded Compact and Windows Embedded Standard.

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 Embedded CE 6.0 and Windows Embedded Compact 7 BSP for Vortex86DX2 SoC CPU.

Windows XP drivers are also on technical support web site.

We provide RoBoard library for A/D, SPI, I2C, PWM and RC servo control. It supports Windows XP, CE, Linux and DOS. Please visit http://www.roboard.com/download_ml.htm for more information.

For other O/S support, please contact soc@dmp.com.tw.
For RoBoard library issue, please contact info@roboard.com.
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 Vortex86DX2 are:
Vortex86DX2 NB Function 0 Configuration Space Registers (IDSEL = AD11/Device 0)

- Vendor ID is 17F3H and Device ID is 6022H.

Vortex86DX2 SB Configuration Space Registers (IDSEL = AD18/Device 7)

- Vendor ID is 17F3H and Device ID is 6035H.

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, 0x6022, 0, &cBusNum, &cDeviceNum);

  /* Read A0H in north bridge (Vendor ID: 17F3, Device: 6022).
     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 -> No Divide
     001 -> Divide 2
     010 -> Divide 3
     011 -> Divide 4
     100 -> Divide 5
     101 -> Divide 8
     110 -> Divide 16
     111 -> Divide 32 */
  c |= 0x01;  /* If CPU is 800MHz, set clock to 400MHz */

  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 Vortex86DX2

Programmers can read PCI register 93H~90H in North Bridge to check Vortex86DX2 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 IsVortex86DX2()
{
  if(0x34504d44L == read_nb(0x90))
    return 1;
  else
    return 0;
}

int main(int argc, char *argv[])
{
  if(IsVortex86DX2())
    printf("Vortex86DX2 found.\n");
  else
    printf("Vortex86DX2 not found.\n");

  return 0;
}
```

3.3. Change CPU Speed

Internally the Vortex86DX2 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: 6022). Bit 7-3 is reserved and bit 2-0 is CPU speed divided control:

Bit[2-0]	Description
000	No Divide
001	Divide 2
010	Divide 3
011	Divide 4
100	Divide 5
101	Divide 8
110	Divide 16
111	Divide 32

For example: if CPU clock is 800MHz and set speed divided control to "001", CPU clock will be 400MHz. 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  800MHz
or  eax, 00000001h ; set CPU clock to 400MHz
or  eax, 00000004h ; set CPU clock to 160MHz
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: 6022).
     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 800MHz, set clock to 400MHz */

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

3.4. GPIO

88 GPIO pins are provided by the Vortex86DX2 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	Port 5	Description
Data Register	78H	79H	7AH	7BH	7CH	100H	
Direction Register	98H	99H	9AH	9BH	9CH	9DH	0: GPIO pin is input mode 1: GPIO pin is output mode

	Port 6	Port 7	Port 8	Port 9	Port A	Description
Data Register	101H	102H	103H	104H	105H	
Direction Register	93H	94H	95H	96H	97H	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 Vortex86DX2 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 Vortex86DX2 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 A8H~ADH. The time resolution of WDT1 is 30.5 u second. Here are registers information:

WDT1 Control Register

Port A8h	
Bit 7	Reserved.
Bit 6	0: Disable watchdog timer. 1: Enable watchdog timer.
Bit 5-0	Reserved.

WDT1 Signal Select Control Register

Port A9h			
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	**ACh**	**ABh**	**AAh**
	D7……D0	D7……D0	D7……D0
Counter	Most SBit ……Least SBit		

Resolution is 30.5u second.

WDT1 Status Register

Port ADh	
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 AEh	
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 AAh-ACh.
2. Select signal from register A9h.
3. Set register A8h bit 6 to enable WDT1.

To clear the watchdog timer counter:

1. Write any value to register AEH

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(0xac, (lTime >> 16) & 0xff);
  outp(0xab, (lTime >>  8) & 0xff);
  outp(0xaa, (lTime >>  0) & 0xff);

  // Reset system. For example, 0x50 to trigger IRQ7
  outp(0xa9, 0xd0);

  // Enable watchdog timer
  c = inp(0xa8);
  c |= 0x40;
  outp(0xa8, c);

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

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 Vortex86DX2(%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(0xa8, 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(0xac, (lTime >> 16) & 0xff);
  outp(0xab, (lTime >> 8) & 0xff);
  outp(0xaa, (lTime >> 0) & 0xff);

  // Set IRQ
  outp(0xa9, cMask);

  _nIrqNo = nIrq;
  printf("\nVortex86DX2Mode, 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(0xa8);
  c |= 0x40;
  outp(0xa8, c);

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

  clock_t clk = clock();
  while(((clock() - clk) / CLK_TCK) < 1)
    outp(0xae, 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 Vortex86DX2(%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. 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.

3.9. Ethernet

The built-in Ethernet in Vortex86DX2is R6040. We provide DSock (DOS real mode TCP/IP socket library) for Vortex86DX2to 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 Vortex86DX2 CPU.

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

3.10. A/D, SPI, I2C, PWM and RC servo control

We provide RoBoard library for A/D, SPI, I2C, PWM and RC servo control. It supports Windows XP, CE, Linux and DOS. Please visit http://www.roboard.com/download_ml.htm for more information.

4. Linux

This section has information for Linux developers to make Vortex86DX2 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 Vortex86DX2 are:

Vortex86DX2 _NB Function 0 Configuration Space Registers (IDSEL = AD11/Device 0)

- Vendor ID is 17F3H and Device ID is 6022H.

Vortex86DX2 _SB Configuration Space Registers (IDSEL = AD18/Device 7)

- Vendor ID is 17F3H and Device ID is 6035H.

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 Vortex86DX2 series with 64M bytes memory.
- Only need 25M bytes storage space.
- Only need 15 seconds to boot on Vortex86DX2series 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 Vortex86DX2. Just enables Intel PIIX for IDE chipset.

4.4. Check Vortex86DX2

Programmers can read PCI register 93H~90H in North Bridge to check Vortex86DX2 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 IsVortex86DX2()
{
  unsigned long val;
  iopl(IO_PERMON);
  outl(0x80000090, 0xcf8);
  val = inl(0xcfc);
  iopl(IO_PERMOFF);
  if(val == 0x34504d44)
    return 1;
  else
    return 0;
}

int main(int argc, char *argv[])
{
  if(IsVortex86DX2())
    printf("Vortex86DX2 found.");
  else
    printf("Vortex86DX2 not found.");

  return 0;
}
```

4.5. Change CPU Speed

Internally the Vortex86DX2 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: 6022). Bit 7-3 is reserved and bit 2-0 is CPU speed divided control:

Bit[2-0]	Description
000	No Divide
001	Divide 2
010	Divide 3
011	Divide 4
100	Divide 5
101	Divide 8
110	Divide 16
111	Divide 32

For example: if CPU clock is 800MHz and set speed divided control to "001", CPU clock will be 400MHz. 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  800MHz
or  eax, 00000001h ; set CPU clock to 400MHz
or  eax, 00000004h ; set CPU clock to 160MHz
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 800MHz, set clock to 400MHz */
  outl(0x800000a0, 0xcf8);
  outb(val, 0xcfc);
  iopl(IO_PERMOFF);

  return 0;
}
```

4.6. GPIO

88 GPIO pins are provided by the Vortex86DX2 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	Port 5	Description
Data Register	78H	79H	7AH	7BH	7CH	100H	
Direction Register	98H	99H	9AH	9BH	9CH	9DH	0: GPIO pin is input mode 1: GPIO pin is output mode

	Port 6	Port 7	Port 8	Port 9	Port A	Description
Data Register	101H	102H	103H	104H	105H	
Direction Register	93H	94H	95H	96H	97H	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 Vortex86DX2 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 A8H~ADH. The time resolution of WDT1 is 30.5 u second. Here are registers information:

WDT1 Control Register

Port A8h	
Bit 7	Reserved.
Bit 6	0: Disable watchdog timer. 1: Enable watchdog timer.
Bit 5-0	Reserved.

WDT1 Signal Select Control Register

Port A9h			
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	**ACh**	**ABh**	**AAh**
	D7……D0	D7……D0	D7……D0
Counter	Most SBit ……Least SBit		

Resolution is 30.5u second.

WDT1 Status Register

Port ADh	
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 AEh	
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 AAh-ACh.
2. Select signal from register A9h.
3. Set register A8h bit 6 to enable WDT1.

To clear the watchdog timer counter:

1. Write any value to register AEH

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(0xac, (lTime >> 16) & 0xff);
  outp(0xab, (lTime >> 8) & 0xff);
  outp(0xaa, (lTime >> 0) & 0xff);

  // Reset system. For example, 0x50 to trigger IRQ7
  outp(0xa9, 0xd0);

  // Enable watchdog timer
  unsigned char c = inp(0xa8);
  c |= 0x40;
  outp(0xa8, c);
  printf("Press any key to stop trigger timer.\n");
  while(!kbhit())
    outp(0xae, 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 Vortex86DX2. 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: 6022H).

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. A/D, SPI, I2C, PWM and RC servo control

We provide RoBoard library for A/D, SPI, I2C, PWM and RC servo control. It supports Windows XP, CE, Linux and DOS. Please visit http://www.roboard.com/download_ml.htm for more information.

5. Windows Embedded Compact 7 (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 Vortex86DX2 and evaluation images.

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

Please get least BSP from soc@dmp.com.tw.

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:

- Vortex86DX2 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
```

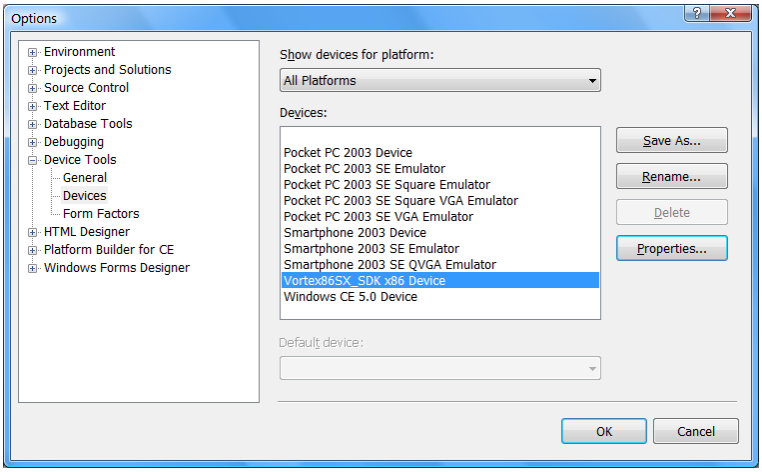
5.4. Uisng CoreCon for Application Debug

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:

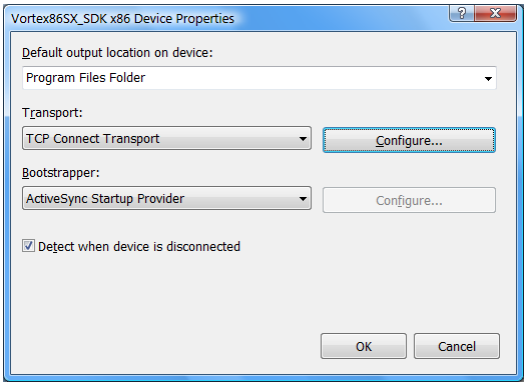
- **clientshutdown.exe**
- **CMAccept.exe**
- **ConmanClient2.exe**
- **eDbgTL.dll**

Here are steps to use CoreCon connection:

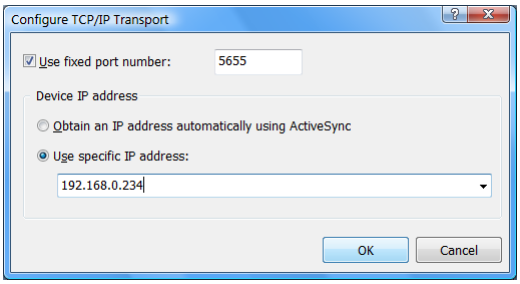
1. Developers can build Windows CE image with those files or use remote file viewer in Platform Builder to upload them.
2. Setup VS2005 for TCP connection from “VS2005 -> Tools -> Options”, then click “Device Tools -> Devices”. Select device you are using.



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



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.

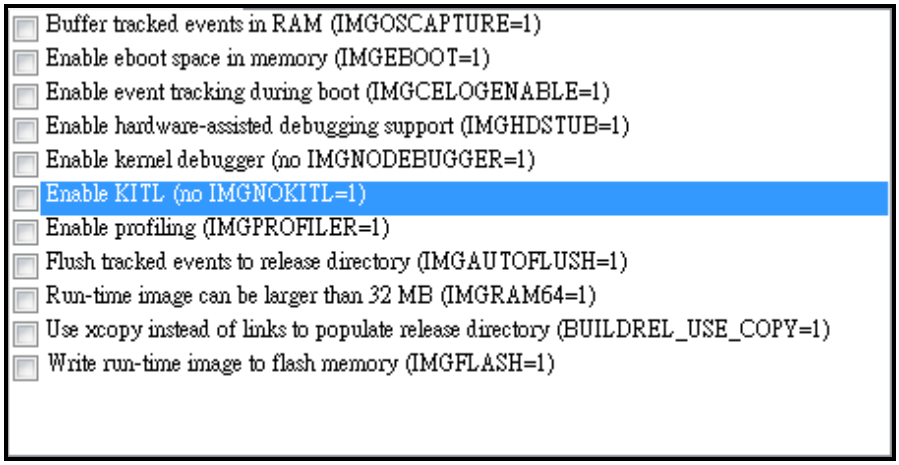


5. Select “VS2005 -> Tools -> Connect to Device” and select your Windows CE device. Before pressing “Connect” button, run **CMAccept.exe** and **ConmanClient2.exe** in your Windows CE device. Developers can run those two programs from those ways:
   I. Direct run those two programs on Windows CE device. 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.



(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. 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\Vortex86DX2 _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\ Vortex86DX2 _60A_x86\OAK\files \project.dat** to add them:

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

5.8. 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\Vortex86DX2 _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\Vortex86DX2 _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\Vortex86DX2 _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
```

5.9. 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"
```

5.10. Adding FTP, TELNET Server and Folder Sharing

Add Relative Components

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

- Third Party -> BSP-> Vortex86DX2 _60A -> RAM Size -> 256MB RAM
- Third Party -> BSP-> Vortex86DX2 _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
```

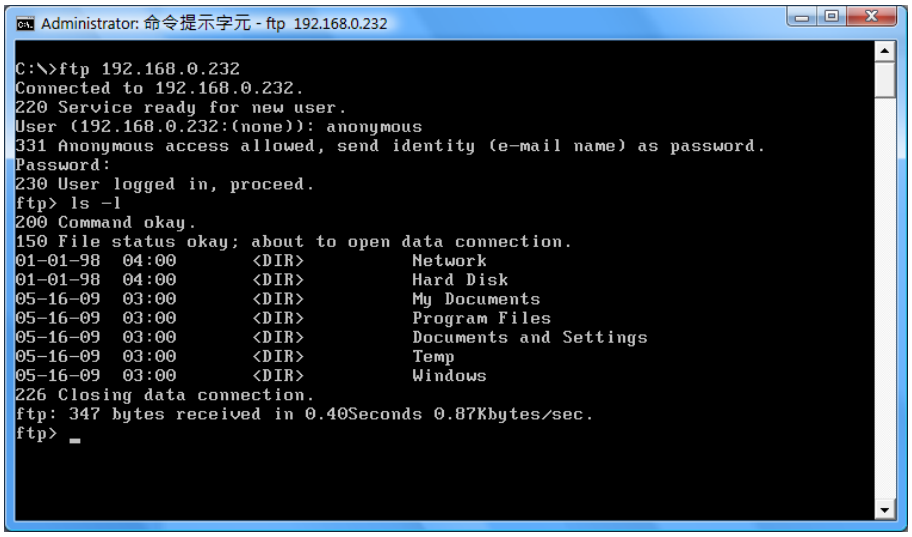
```
[HKEY_LOCAL_MACHINE\Services\SMBServer\Shares]
  "UseAuthentication"=dword:0

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

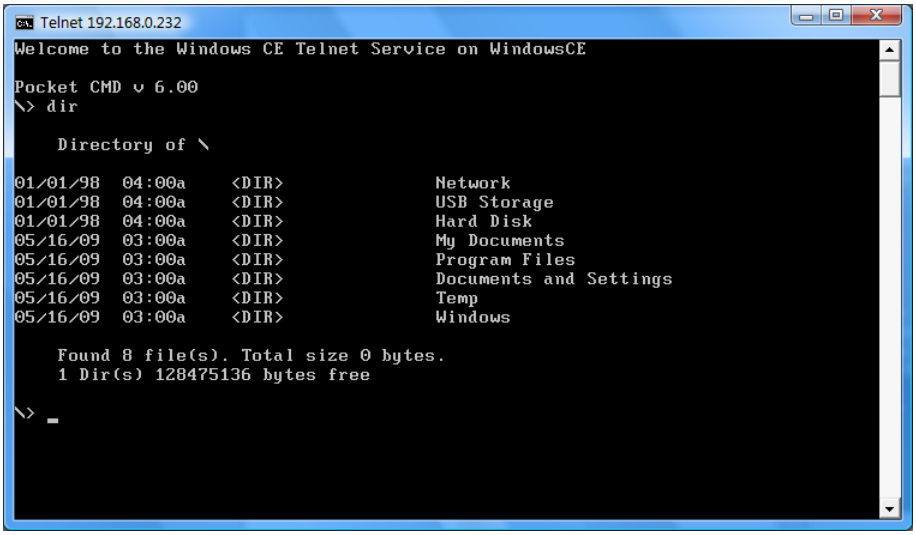
In FTPD registry, we enable anonymous login and the default root path is the whole file system on Windows CE device. For more information about registry settings, please search “FTP server” or “TELNET” in help of Platform Builder.

Access Test

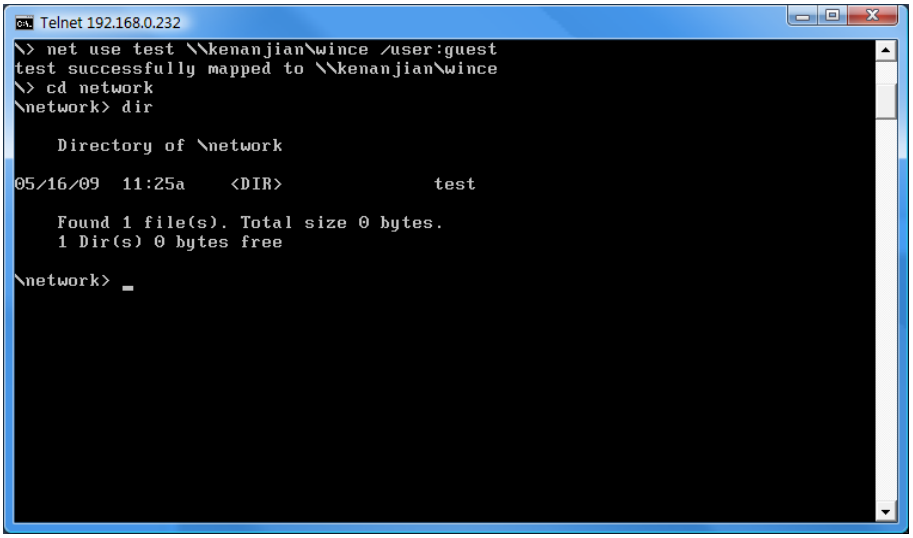
We use FTP client in Windows command prompt to demo. Developers can use their favorite FTP client to do files exchange between Windows CE device and desktop computer. The IP of Windows CE is 192.168.0.232. Use user name “anonymouse” and password “ftp” to login FTP server on Windows CE. Here is the FTP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



The folder “USB Storage” is USB pen drive and “Hard Disk” is DOM on Vortex86DX2 board. Below picture is the screen using telnet to connect to Windows CE device:



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:



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.

5.11. 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 Vortex86DX2 are:

Vortex86DX2 _NB Function 0 Configuration Space Registers (IDSEL = AD11/Device 0)

- Vendor ID is 17F3H and Device ID is 6022H.

Vortex86DX2 _SB Configuration Space Registers (IDSEL = AD18/Device 7)

- Vendor ID is 17F3H and Device ID is 6035H.

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 IsVortex86DX2()
{
  unsigned long val;
  _asm
  {
    mov dx,  0xcf8
    mov eax, 0x80000090
    out dx,  eax
    mov dx,  0xcfc
    in  eax, dx
    mov val, eax
  }

  if(val == 0x34504d44)
    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
  }
}
```

5.12. Check Vortex86DX2

Programmers can read PCI register 93H~90H in North Bridge to check Vortex86DX2 SoC. Here is example to access PCI configuration space registers via CFCH/CF8H port.

Windows CE Example

```
#include "stdafx.h"

unsigned int IsVortex86DX2()
{
  unsigned long val;
  _asm
  {
    mov dx,  0xcf8
    mov eax, 0x80000090
    out dx,  eax
    mov dx,  0xcfc
    in  eax, dx
    mov val, eax
  }

  if(val == 0x34504d44)
    return 1;
  else
    return 0;
}

int _tmain(int argc, TCHAR *argv[], TCHAR *envp[])
{
  if(IsVortex86DX2())
    _tprintf(_T("Vortex86DX2 found."));
  else
    _tprintf(_T("Vortex86DX2 not found."));

  return 0;
}
```

5.13. Change CPU Speed

Internally the Vortex86DX2 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: 6022). Bit 7-3 is reserved and bit 2-0 is CPU speed divided control:

Bit[2-0]	Description
000	No Divide
001	Divide 2
010	Divide 3
011	Divide 4
100	Divide 5
101	Divide 8
110	Divide 16
111	Divide 32

For example: if CPU clock is 800MHz and set speed divided control to "001", CPU clock will be 400MHz. 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  800MHz
or  eax, 00000001h ; set CPU clock to 400MHz
or  eax, 00000004h ; set CPU clock to 160MHz
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 800MHz, set clock to 400MHz */
    out dx,  eax
  }
  return 0;
}
```

5.14. GPIO

88 GPIO pins are provided by the Vortex86DX2 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	Port 5	Description
Data Register	78H	79H	7AH	7BH	7CH	100H	
Direction Register	98H	99H	9AH	9BH	9CH	9DH	0: GPIO pin is input mode 1: GPIO pin is output mode

	Port 6	Port 7	Port 8	Port 9	Port A	Description
Data Register	101H	102H	103H	104H	105H	
Direction Register	93H	94H	95H	96H	97H	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;
}
```

5.15. Watchdog Timer

There are two watchdog timers in Vortex86DX2 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 A8H~ADH. The time resolution of WDT1 is 30.5 u second. Here are registers information:

WDT1 Control Register

Port A8h	
Bit 7	Reserved.
Bit 6	0: Disable watchdog timer. 1: Enable watchdog timer.
Bit 5-0	Reserved.

WDT1 Signal Select Control Register

Port A9h			
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	**ACh**	**ABh**	**AAh**
	D7……D0	D7……D0	D7……D0
Counter	Most SBit ……Least SBit		

Resolution is 30.5u second.

WDT1 Status Register

Port ADh	
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 AEh	
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 AAh-ACh.
5. Select signal from register A9h.
6. Set register A8h bit 6 to enable WDT1.

To clear the watchdog timer counter:

2. Write any value to register AEH

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(0xac, (time >> 16) & 0xff);
  outp(0xab, (time >>  8) & 0xff);
  outp(0xaa, (time >>  0) & 0xff);

  // Reset system. For example, 0x50 to trigger IRQ7
  outp(0xa9, 0xd0);

  // Enable watchdog timer
  unsigned char c = inp(0xa8);
  c |= 0x40;
  outp(0xa8, c);

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

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

  return 0;
}
```

5.16. ISO-in-Chip

It is a 32 Byte one-time write Flash area in Vortex86DX2. 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: 6022H).

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;
}
```

5.17. A/D, SPI, I2C, PWM and RC servo control

We provide RoBoard library for A/D, SPI, I2C, PWM and RC servo control. It supports Windows XP, CE, Linux and DOS. Please visit http://www.roboard.com/download_ml.htm for more information.

5.18. 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;
}
```

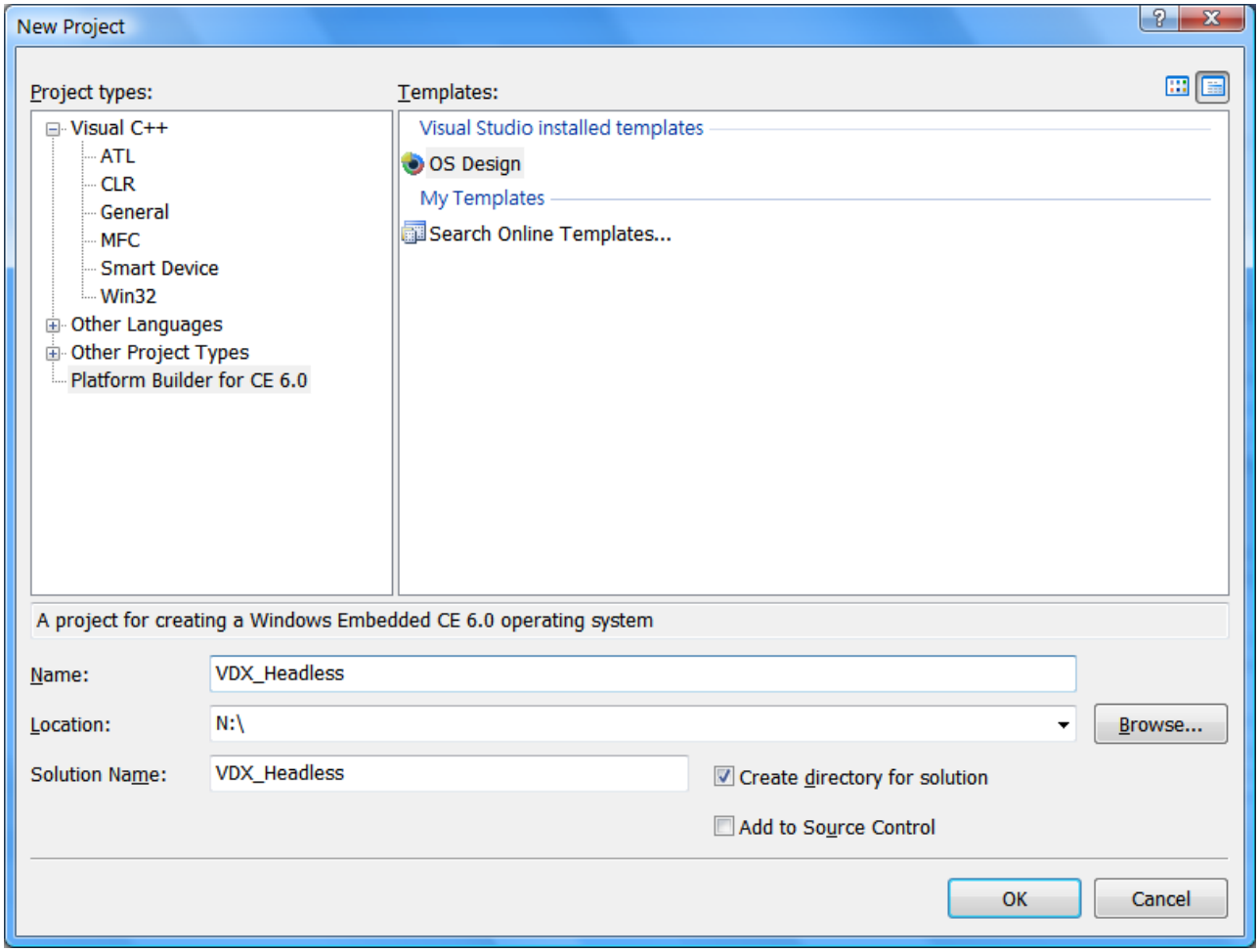
5.19. Building Windows Embedded CE 6.0 Headless Image

This section will show developers to build Windows CE headless image for Vortex86DX2 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.

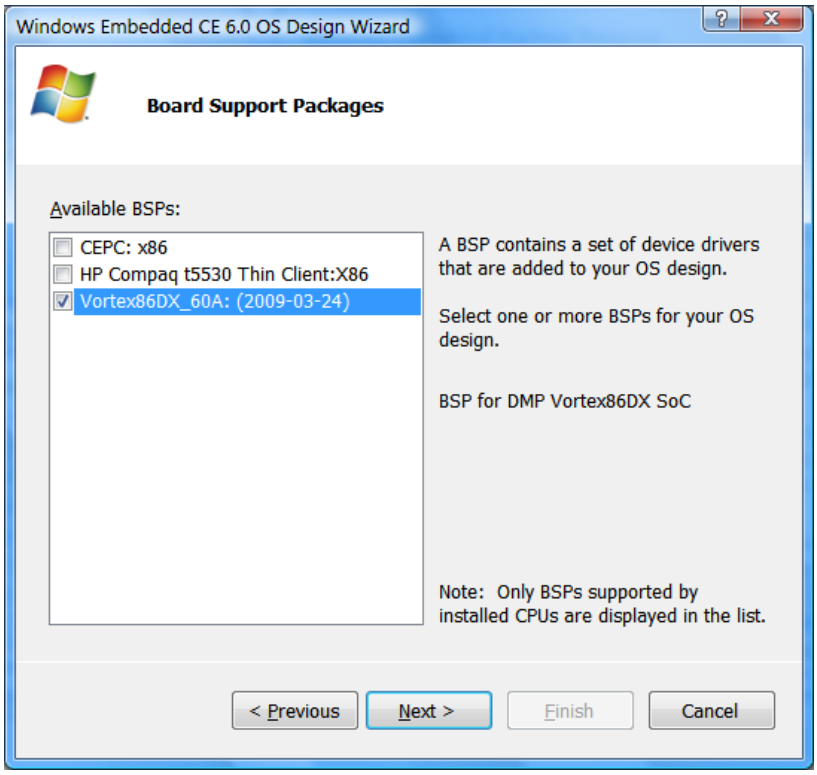
Please refer to **BSP** section to install BSP and QFE to start below steps.

Create a Headless Workspace

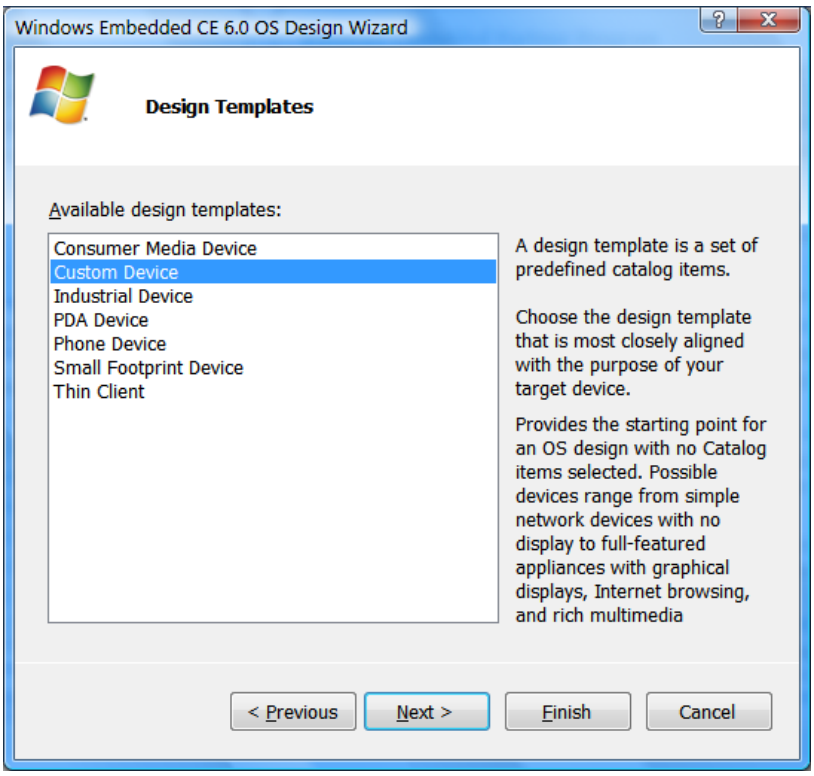
Use “VDX_Headless” as project name.



Start new O/S design with Vortex86DX2 BSP:



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



Add Necessary Components

Here are network components used by our demo:

- Third Party -> BSP-> Vortex86DX2 _60A -> RAM Size -> 256MB RAM
- Third Party -> BSP-> Vortex86DX2 _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 Vortex86DX2 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\Vortex86DX2 _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
```

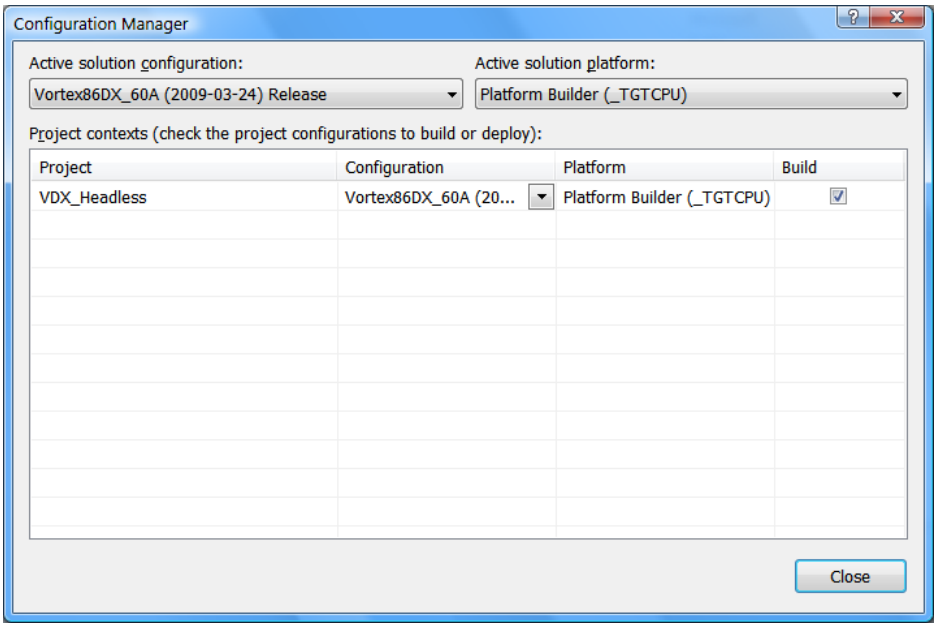
Modify project.bib to add this line for Platform Builder to include demo program into Windows CE image:

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

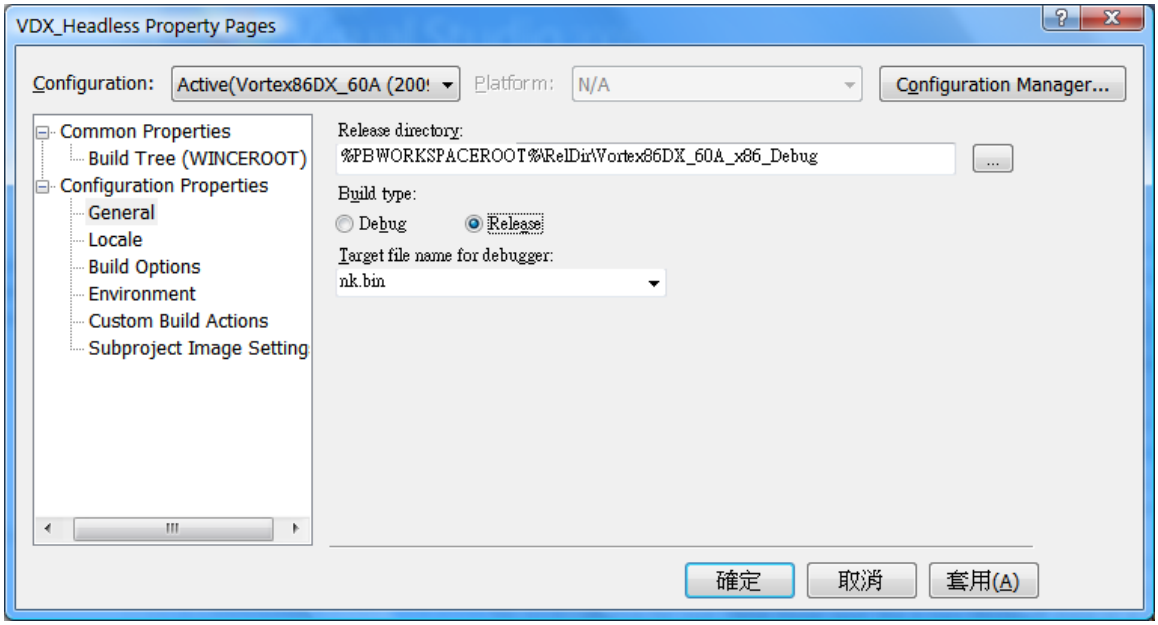
Build Release Image

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/Vortex86DX2 /WinCE_Development_Note.pdf for more detail.

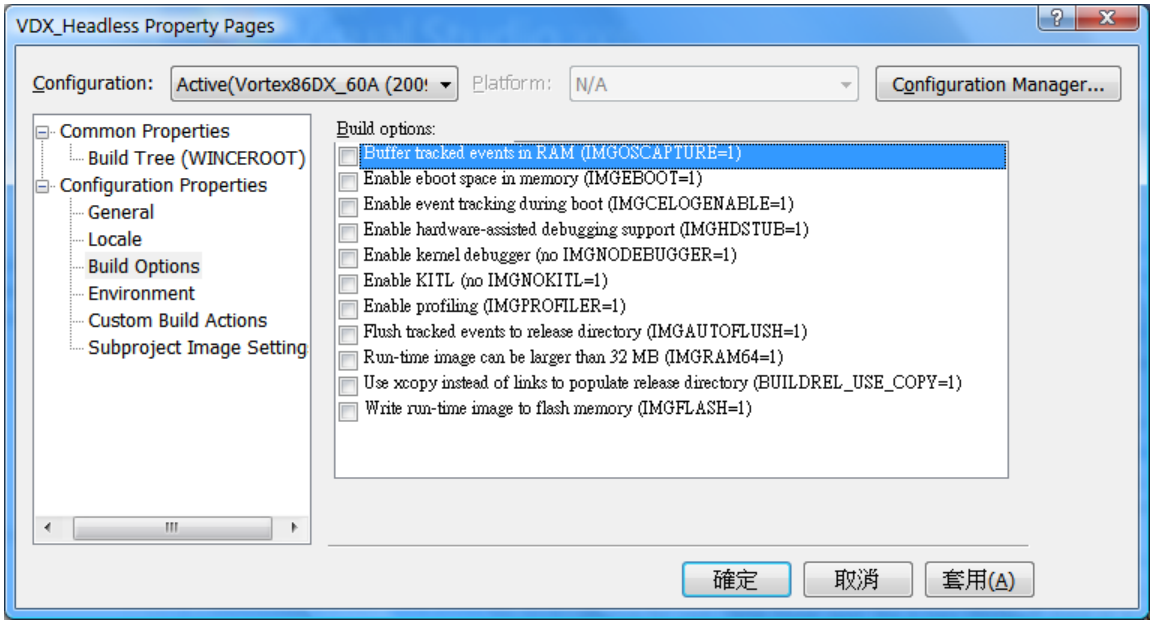
To make release Windows CE image, select "VS2005 -> Build -> Configuration Manager" to enable release mode:



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



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



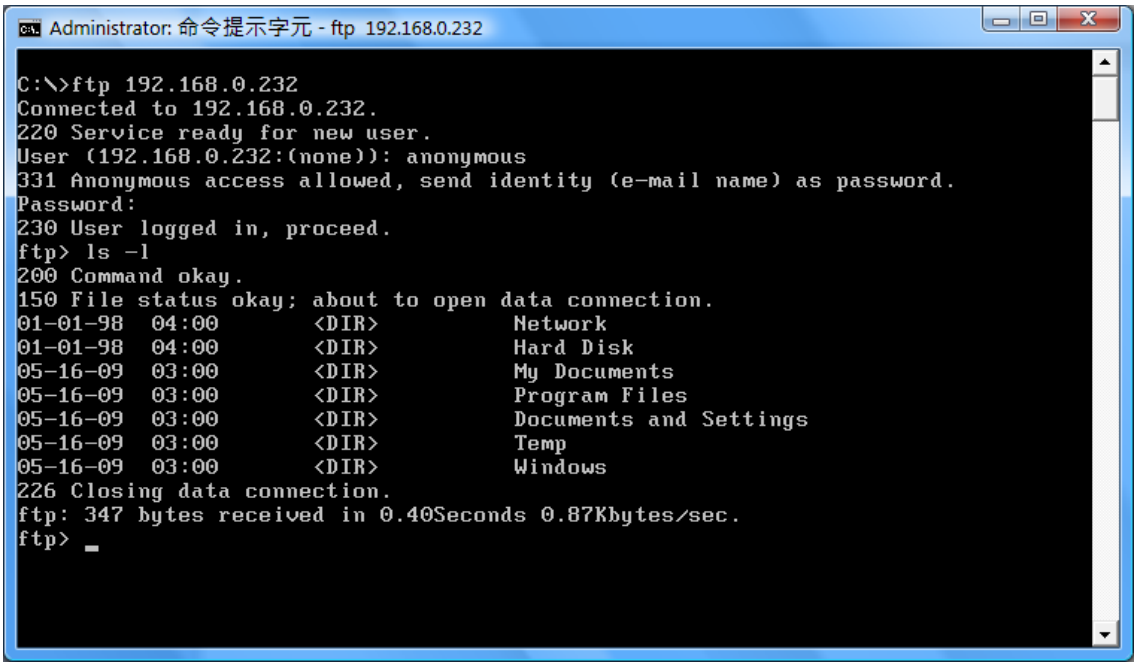
When settings are ready, build new Windows CE image. Boot is from USB or DOM for test in next section.

Test

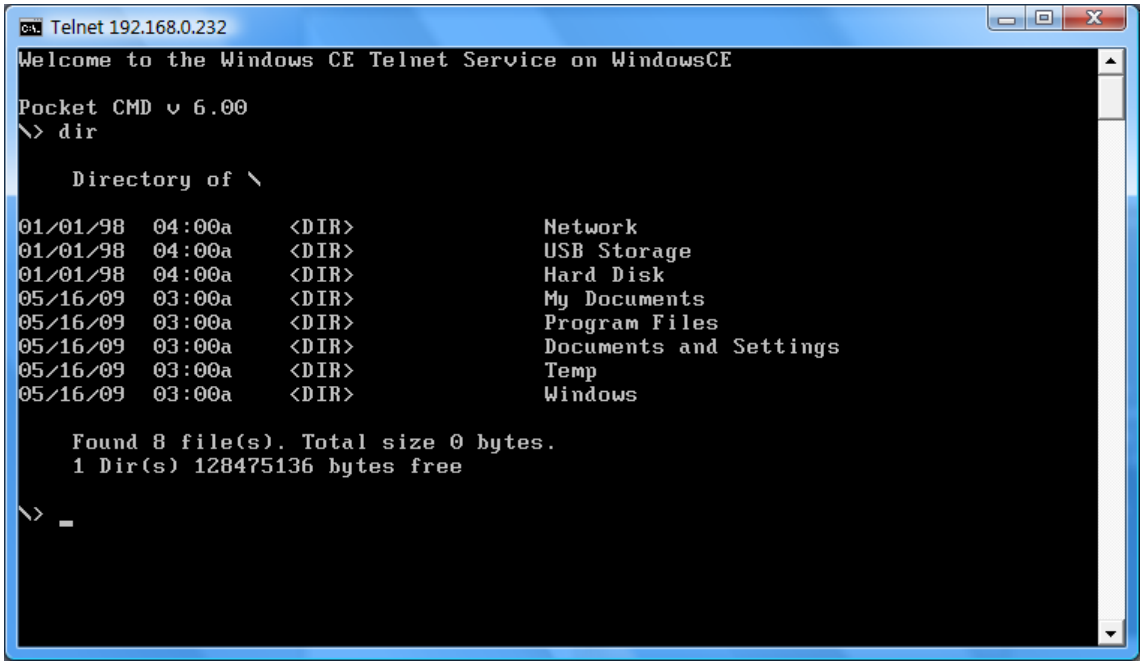
After Windows CE is loaded, our demo program will run automatically. Check serial port output to ensure demo program is running.

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

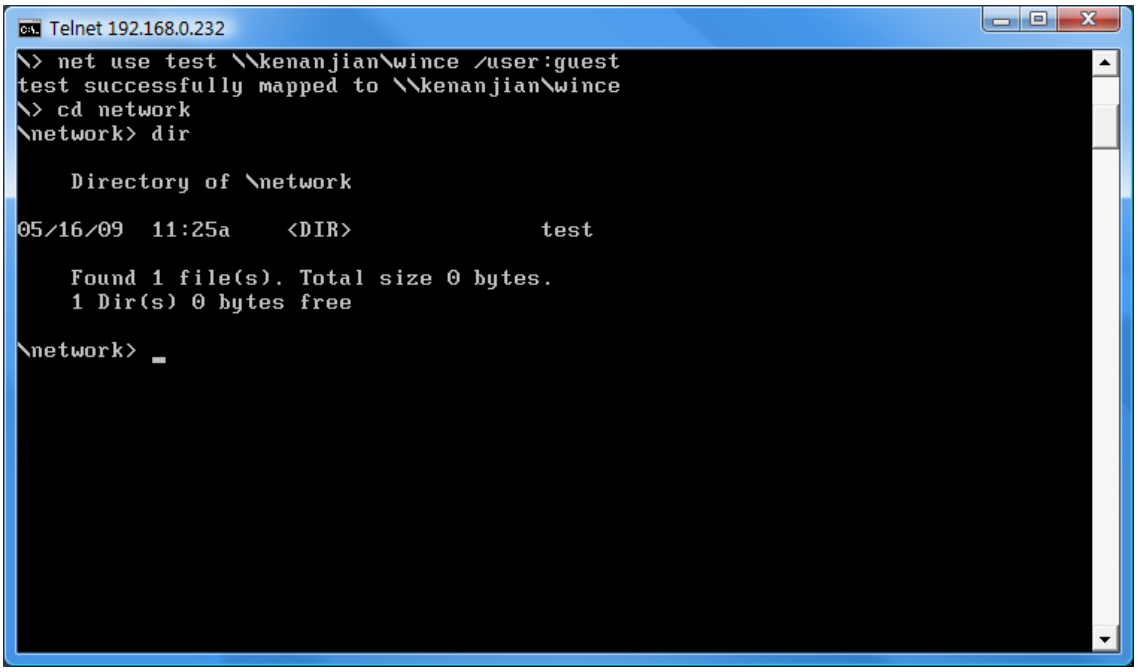
name “anonymouse” and password “ftp” to login FTP server on Windows CE. Here is the FTP test screen:



The folder “USB Storage” is USB pen drive and “Hard Disk” is DOM on Vortex86DX2 board. Below picture is the screen using telnet to connect to Windows CE device:



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:



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 7 (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 7/8 and next version is Windows Embedded Standard 2011 with Windows 7 core.

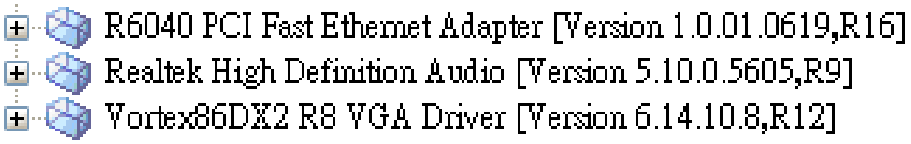
Visit http://www.microsoft.com/windowsembedded/en-us/default.mspx form Micrsoft web site to know more.

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

6.1. Drivers and Evalaution Images

For Vortex86DX2 , there are three components for Target Designer to generate XPe image for Vortex86DX2 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 Vortex86DX2 PMQ file and using Target Designer to import.
- If step 1 & 2 are okay, you will see Vortex86DX2 drivers are added into your design. Or, search "Vortex86DX2 " to find those 3 drivers to add them manually.



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/Vortex86DX2 /#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. Vortex86DX2 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. 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

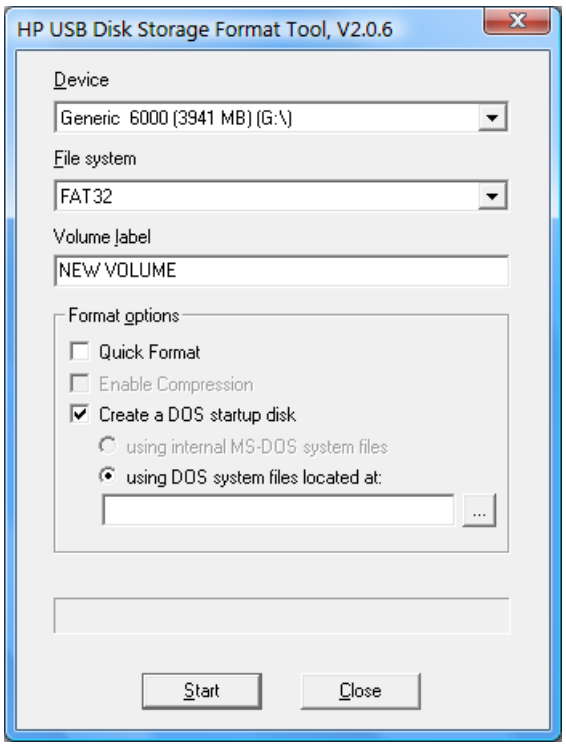
We will introducr some tips for programmers. They are from FAQ and technical support e-mail. If you have any good idea, mail to soc@dmp.com.tw to add.

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:



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 your USB mass storage can not boot properly, run make_lba.bat to try again.

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 Vortex86DX2 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. 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 / info@roboard.com.

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: VDX2-6690A using Vortex86DX2 , BIOS version is A6 01-08-2014.
   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/Vortex86DX2 /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/Vortex86DX2 /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: No.15, Wugong 5th Rd., Xinzhuang Dist., New Taipei City 24890, Taiwan (R.O.C.)