SERVO Control Interface in Vortex86DX

2009-04-10

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

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

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

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

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

Please read our DOS, Linux, Windows CE and Windows XP example in next section, and refer to SERVO registers and control registers in PCI South Bridge.

DOS Example Code

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

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

#define EMIT_DWORD(x) \
__emit__((unsigned char)x);__emit__((unsigned char)(x>>8));\
__emit__((unsigned char)(x>>16));__emit__((unsigned char)(x>>24))

#define MOV_EAX(x) __emit__(0x66, 0xb8); EMIT_DWORD(x)

#define MOV_VAR_EAX(x) __emit__(0x66); _asm mov WORD PTR x, ax

#define MOV_EAX_VAR(x) __emit__(0x66); _asm mov ax, WORD PTR x

#define OUT_EDX_EAX __emit__(0x66); _asm out dx, ax
#define IN_EAX_EDX __emit__(0x66); _asm in ax, dx

// read south bridge register
static unsigned long read_sb_reg(byte idx)
{
    unsigned long retval;

    _asm mov dx, 0x0cf8

    MOV_EAX(0x80003800L);
    _asm mov al, idx

    OUT_EDX_EAX;

    _asm mov dx, 0x0cfc
    IN_EAX_EDX;
    MOV_VAR_EAX(retval);

    return retval;
}

// write south bridge register
static void write_sb_reg(byte idx, unsigned long val)
{
    _asm mov dx, 0x0cf8

   MOV_EAX(0x80003800L);
    _asm mov al, idx

    OUT_EDX_EAX;

    _asm mov dx, 0x0cfc

    MOV_EAX_VAR(val);
    OUT_EDX_EAX;
}

void outpdw(unsigned addr, unsigned long val) {
       _asm mov dx, WORD PTR addr
       MOV_EAX_VAR(val);
       OUT_EDX_EAX;
}

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

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

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

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

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

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

    write_sb_reg(0xc8, 0xffffffffL); // switch GPIO function to PWM output
    write_sb_reg(0xc0, read_sb_reg(0xc0)|2L); // disable GPIO port 4's COM function

    write_sb_reg(0xd0, 0x00800000L+base_address); // disable IRQ, Address Decode enable, 10Mhz

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

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

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

    //enable PWM
    outpdw(base_address+0x14+channel*12, 0x80000000L+300); // pulse count mode; send 300 PWM pulses
    //outpdw(base_address+0x14+channel*12, 0xc0000000L); // continue mode; send PWM pulses continuously

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

Linux Example Code

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

#define IO_PERMOFF  0
#define IO_PERMON   3

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

// write south bridge register
void write_sb_reg(unsigned char bIdx, unsigned long dwVal)
{
  iopl(IO_PERMON);
  outl(0x80003800 + bIdx, 0xcf8);
  outl(dwVal, 0xcfc);
  iopl(IO_PERMOFF);
}

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

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

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

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

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

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

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

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

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

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

  // enable PWM
```

```
  outpdw(base_address + 0x14 + channel * 12, 0x80000000L + 300);  // pulse count mode; send 300 PWM pulses
  //outpdw(base_address+0x14+channel*12, 0xc0000000L); // continue mode; send PWM pulses continuously

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

  return 0;
}
```

Windows CE Example Code

```
// WINCE_PWM_Demo.cpp : Defines the entry point for the console application.
//

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

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

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

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

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

// read south bridge register
static DWORD read_sb_reg(BYTE bIdx)
{
    DWORD dwVal = 0;

    _asm {
        mov dx, 0x0cf8
        mov eax, 0x80003800
        mov al, BYTE PTR bIdx
        out dx, eax
        mov dx, 0x0cfc
        in  eax, dx
        mov dwVal, eax
    }
    return dwVal;
}

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

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

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

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

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

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

    write_sb_reg(0xc8, 0xffffffffL); // switch GPIO function to PWM output
    write_sb_reg(0xc0, read_sb_reg(0xc0)|2L); // disable GPIO port 4's COM function

    write_sb_reg(0xd0, 0x00800000L+base_address); // disable IRQ, Address Decode enable, 10Mhz

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

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

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

    //enable PWM
    outpdw(base_address+0x14+channel*12, 0x80000000L+300); // pulse count mode; send 300 PWM pulses
    //outpdw(base_address+0x14+channel*12, 0xc0000000L); // continue mode; send PWM pulses continuously

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

    return 0;
}
```

Windows XP Example Code

The Windows XP example code use WinIO.DLL to access I/O port. Search “WinIO” from Google for more detail.

```
// VDX_PWM_XPe_Demo.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include <Windows.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

#include ".\winiolib\WinIo.h"

#define CTL_8Bits  1
#define CTL_16Bits 2
#define CTL_32Bits 4

DWORD read_sb_reg(byte bIdx);
void write_sb_reg(byte bIdx, DWORD dwPortVal);
void outpdw(unsigned addr, DWORD dwPortVal);
void set_pluse(int channel, unsigned long period, unsigned long high_pulse_width);

static unsigned int base_address = 0xfe00;
static unsigned long us = 10L; // assume the Servo Clock (see Internal SERVO Control Register) is 10MHZ

int _tmain(int argc, _TCHAR* argv[])
{
      // Open WinIO driver
      bool bRet = InitializeWinIo();
      if (!bRet)
            return 1;

      int channel = 0;
      if (argc > 1) channel = _wtoi(argv[1]);


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

      write_sb_reg(0xc8, 0xffffffffL); // switch GPIO function to PWM output

      write_sb_reg(0xc0, read_sb_reg(0xc0)|2L); // disable GPIO port 4's COM function

      write_sb_reg(0xd0, 0x00800000L+base_address); // disable IRQ, Address Decode enable, 10Mhz

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

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

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

      // enable PWM
      outpdw(base_address+0x14+channel*12, 0x80000000L+300); // pulse count mode; send 300 PWM pulses
      //outpdw(base_address+0x14+channel*12, 0xc0000000L); // continue mode; send PWM pulses continuously

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

      // Close WinIO driver
      ShutdownWinIo();

      return 0;
}



DWORD read_sb_reg(byte bIdx)
{
```

```
      DWORD dwPortVal = 0;
      SetPortVal(0xcf8,0x80003800L+bIdx,CTL_32Bits);
      GetPortVal(0xcfc, &dwPortVal, CTL_32Bits);
      return dwPortVal;
}

// write south bridge register
void write_sb_reg(byte bIdx, DWORD dwPortVal)
{
      SetPortVal(0xcf8,0x80003800L+bIdx,CTL_32Bits);
      SetPortVal(0xcfc,dwPortVal,CTL_32Bits);
}


void outpdw(unsigned addr, DWORD dwPortVal)
{
      SetPortVal(addr,dwPortVal,CTL_32Bits);
}



void set_pluse(int channel, unsigned long period, unsigned long high_pulse_width)
{
      unsigned long low_pulse_width = period - high_pulse_width;

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

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

SERVO Registers

(**B**ase **A**ddress Refers to the Register of index D3h-D0h, IDSEL = AD18/SB of PCI Configuration Register)

IO Address	Register Name
BA + 00H	SERVO Interrupt Mask Register
BA + 04H	SERVO Interrupt Status Register
BA + 08H	SERVO Sync Register
BA + 0CH	SERVO[0] Pulse Low Count Register
BA + 10H	SERVO[0] Pulse High Count Register
BA + 14H	SERVO[0] Control Register
BA + 18H	SERVO[1] Pulse Low Count Register
BA + 1CH	SERVO[1] Pulse High Count Register
BA + 20H	SERVO[1] Control Register
BA + 24H	SERVO[2] Pulse Low Count Register
BA + 28H	SERVO[2] Pulse High Count Register
BA + 2CH	SERVO[2] Control Register
BA + 30H	SERVO[3] Pulse Low Count Register
BA + 34H	SERVO[3] Pulse High Count Register
BA + 38H	SERVO[3] Control Register
BA + 3CH	SERVO[4] Pulse Low Count Register
BA + 40H	SERVO[4] Pulse High Count Register
BA + 44H	SERVO[4] Control Register
BA + 48H	SERVO[5] Pulse Low Count Register
BA + 4CH	SERVO[5] Pulse High Count Register
BA + 50H	SERVO[5] Control Register
BA + 54H	SERVO[6] Pulse Low Count Register
BA + 58H	SERVO[6] Pulse High Count Register
BA + 5CH	SERVO[6] Control Register
BA + 60H	SERVO[7] Pulse Low Count Register
BA + 64H	SERVO[7] Pulse High Count Register
BA + 68H	SERVO[7] Control Register
BA + 6CH	SERVO[8] Pulse Low Count Register
BA + 70H	SERVO[8] Pulse High Count Register
BA + 74H	SERVO[8] Control Register
BA + 78H	SERVO[9] Pulse Low Count Register
BA + 7CH	SERVO[9] Pulse High Count Register
BA + 80H	SERVO[9] Control Register
BA + 84H	SERVO[10] Pulse Low Count Register
BA + 88H	SERVO[10] Pulse High Count Register
BA + 8CH	SERVO[10] Control Register
BA + 90H	SERVO[11] Pulse Low Count Register

IO Address	Register Name
BA + 94H	SERVO[11] Pulse High Count Register
BA + 98H	SERVO[11] Control Register
BA + 9CH	SERVO[12] Pulse Low Count Register
BA + A0H	SERVO[12] Pulse High Count Register
BA + A4H	SERVO[12] Control Register
BA + A8H	SERVO[13] Pulse Low Count Register
BA + ACH	SERVO[13] Pulse High Count Register
BA + B0H	SERVO[13] Control Register
BA + B4H	SERVO[14] Pulse Low Count Register
BA + B8H	SERVO[14] Pulse High Count Register
BA + BCH	SERVO[14] Control Register
BA + C0H	SERVO[15] Pulse Low Count Register
BA + C4H	SERVO[15] Pulse High Count Register
BA + C8H	SERVO[15] Control Register
BA + CCH	SERVO[16] Pulse Low Count Register
BA + D0H	SERVO[16] Pulse High Count Register
BA + D4H	SERVO[16] Control Register
BA + D8H	SERVO[17] Pulse Low Count Register
BA + DCH	SERVO[17] Pulse High Count Register
BA + E0H	SERVO[17] Control Register
BA + E4H	SERVO[18] Pulse Low Count Register
BA + E8H	SERVO[18] Pulse High Count Register
BA + ECH	SERVO[18] Control Register
BA + F0H	SERVO[19] Pulse Low Count Register
BA + F4H	SERVO[19] Pulse High Count Register
BA + F8H	SERVO[19] Control Register
BA + FCH	SERVO[20] Pulse Low Count Register
BA + 100H	SERVO[20] Pulse High Count Register
BA + 104H	SERVO[20] Control Register
BA + 108H	SERVO[21] Pulse Low Count Register
BA + 10CH	SERVO[21] Pulse High Count Register
BA + 110H	SERVO[21] Control Register
BA + 114H	SERVO[22] Pulse Low Count Register
BA + 118H	SERVO[22] Pulse High Count Register
BA + 11CH	SERVO[22] Control Register
BA + 120H	SERVO[23] Pulse Low Count Register
BA + 124H	SERVO[23] Pulse High Count Register
BA + 128H	SERVO[23] Control Register
BA + 12CH	SERVO[24] Pulse Low Count Register

IO Address	Register Name
BA + 130H	SERVO[24] Pulse High Count Register
BA + 134H	SERVO[24] Control Register
BA + 138H	SERVO[25] Pulse Low Count Register
BA + 13CH	SERVO[25] Pulse High Count Register
BA + 140H	SERVO[25] Control Register
BA + 144H	SERVO[26] Pulse Low Count Register
BA + 148H	SERVO[26] Pulse High Count Register
BA + 14CH	SERVO[26] Control Register
BA + 150H	SERVO[27] Pulse Low Count Register
BA + 154H	SERVO[27] Pulse High Count Register
BA + 158H	SERVO[27] Control Register
BA + 15CH	SERVO[28] Pulse Low Count Register
BA + 160H	SERVO[28] Pulse High Count Register
BA + 164H	SERVO[28] Control Register
BA + 168H	SERVO[29] Pulse Low Count Register
BA + 16CH	SERVO[29] Pulse High Count Register
BA + 170H	SERVO[29] Control Register
BA + 174H	SERVO[30] Pulse Low Count Register
BA + 178H	SERVO[30] Pulse High Count Register
BA + 17CH	SERVO[30] Control Register
BA + 180H	SERVO[31] Pulse Low Count Register
BA + 184H	SERVO[31] Pulse High Count Register
BA + 188H	SERVO[31] Control Register

I/O Port: BA + 00h

Register Name: SERVO Interrupt Mask Register

Reset Value: 00000000h

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

Bit	Name	Attribute	Description
31-0	SIM[31-0]	R/W	SERVO[31-0] Interrupt Mask Register 1: Enable Interrupt 0: Disable Interrupt

I/O Port: BA + 04h

Register Name: SERVO Interrupt Status Register

Reset Value: 00000000h

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

Bit	Name	Attribute	Description
31-0	SIS[31-0]	R/W	SERVO[31-0] Interrupt Status Register 1: Interrupt happen and write “1” to clear 0: No Interrupt

I/O Port: BA + 08h

Register Name: SERVO Sync Status Register

Reset Value: 00000000h

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

Bit	Name	Attribute	Description
31-0	SYNC[31-0]	R/W	SERVO[31-0] Sync Register 1: SERVO will be hold 0: SERVO without hold

I/O Port: BA + 0CH, 18H, 24H, 30H, 3CH, 48H, 54H, 60H, 6CH, 78H, 84H, 90H, 9CH, A8H, B4H, C0H, CCH, D8H, E4H, F0H, FCH, 108H, 114H, 120H, 12CH, 138H, 144H, 150H, 15CH, 168H, 174H, 180H

Register Name: SERVO Pulse Low Register

Reset Value: 00000000h

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

Bit	Name	Attribute	Description
31-0	SPL	R/W	SERVO Pulse Low Register. SERVO clock is 10MHz

I/O Port: BA + 10H, 1CH, 28H, 34H, 40H, 4CH, 58H, 64H, 70H, 7CH, 88H, 94H, A0H, ACH, B8H, C4H, D0H, DCH, E8H, F4H,100H,10CH,118H,124H,130H,13CH,148H,154H,160H,16CH,178H,184H

Register Name: SERVO Pulse High Register

Reset Value: 00000000h

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

Bit	Name	Attribute	Description
31-0	SPH	R/W	SERVO Pulse High Register. SERVO clock is 10MHz

I/O Port: BA + 14H, 20H, 2CH, 38H, 44H, 50H, 5CH, 68H, 74H, 80H, 8CH, 98H, A4H, B0H, BCH, C8H, D4H, E0H, ECH, F8H, 104H, 110H, 11CH, 128H, 134H, 140H, 14CH, 158H, 164H, 170H, 17CH, 188H

Register Name: SERVO Control Register

Reset Value: 00000000h

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

Bit	Name	Attribute	Description
31	SE	R/W	SERVOx Enable Control 1: SERVOx enable 0: SERVOx disable
30	CM	R/W	SERVOx Continuous Mode 1: SERVOx Continuous Mode enable 0: SERVOx Continuous Mode disable
29	INVS	R/W	Inverse SERVO signal 0: default SERVO out ‘0’, SPH specify ‘I’, SPL specify “0”. 1: Inverse output signal of upper case
28	Rsvd	RO	Reserved
27-0	RC	R/W	SERVOx Repeat Count. It is used when CM=0.

Vortex86DX South Bridge Configuration Registers

Register Offset: 01h – 00h

Register Name: Vendor ID Register

Reset Value: 17F3h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
VID															

Bit	Name	Attribute	Description
15-0	VID	RO	This register contains a 16-bit value assigned to South Bridge Vendor ID.

Register Offset: 03h – 02h

Register Name: Device ID Register

Reset Value: 6031h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DID															

Bit	Name	Attribute	Description
15-0	DID	RO	This register contains a 16-bit value to specify a particular device.

Register Offset: 4Bh – 48h

Register Name: Buffer Strength/Clock Output Control Register

Reset Value: 3FFF3600h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Rvd		GPIO4_DRV		GPIO3_3_DRV		GPIO3_2_DRV		GPIO3_1_DRV		GPIO2_DRV		GPIO1_DRV		GPIO0_DRV		Rvd	PCI_CTL_SR	PCI_CTL_DRV		PCI_DT_SR	PCI_DT_DRV		Rvd						ISACK_CTL	14M_CTL	24M_CTL	25M_CTL

Bit	Name	Attribute	Description
31-30	Rvd	RO	Reserved
29-28	GPIO4_DRV	RW	GPIO4[7-0]/SERVO[31-24]/COM1 Driving Current Control 00: 4mA 01: 8mA 10: 12mA 11: 16mA (default)
21-20	GPIO2_DRV	RW	GPIO2[7-0]/SERVO[23-16]/SA[31-24] Driving Current Control

Bit	Name	Attribute	Description
			00: 4mA 01: 8mA 10: 12mA 11: 16mA (default)
19-18	GPIO1_DRV	RW	GPIO1[7-0]/SERVO[15-8] Driving Current Control 00: 4mA 01: 8mA 10: 12mA 11: 16mA (default)
17-16	GPIO0_DRV	RW	GPIO0[7-0]/SERVO[7-0] Driving Current Control 00: 4mA 01: 8mA 10: 12mA 11: 16mA (default)

Register Offset: BFh – BCh

Register Name: On-Chip Device Control Register

Reset Value: 00000000h

31 30 29 28 27 26 25 24	23	22	21	20	19	18	17	16	15 14 13	12	11 10	9	8	7	6	5	4	3	2	1	0
Reserved		UDP	GSP	SVP	G1IP	G0IP	I2C1P	I2C0P	Reserved	PPP	Reserved	COM9P	COM4P	COM3P	COM2P	COM1P	Rsvd	MACP	USB2P	USB1P	IDEP

Bit	Name	Attribute	Description
20	SVP	R/W	ON-Chip SERVO power-down control 0: on-chip SERVO activate (default) 1: on-chip SERVO power-down

Register Offset: C3-C0h

Register Name: Internal Peripheral Feature Control Register

Reset Value: 032C0500h

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

Bit	Name	Attribute	Description
1	PINS1	R/W	PINS selection for COM1 and GPIO Port 4 0: 8 PINs for COM1(default) 1: 8 PINs for GPIO port 4
0	PINS0	R/W	PINS selection for External SPI and GPIO Port3 [3-0] 0: 4 pins for GPIO port3 [3-0] (default) 1: 4 pins for external SPI

Register Offset: CBh – C8h

Register Name: Internal Peripheral Feature Control Register II

Reset Value: 00000000h

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

Bit	Name	Attribute	Description
31-24	GS[31-24]	R/W	GPIO_P4[7-0] and SERVO[31-24] selection. This register is used only when SB register C0h bit1 is “1”. 0: PINS for GPIO_P4 (default) 1: PINS for SERVO
23 – 16	GS[23-16]	R/W	GPIO_P2[7-0] and SERVO[23-16] selection. This register is used only when STRAP[1] (NB register 60h bit19) is “1”. 0: PINS for GPIO_P2 (default) 1: PINS for SERVO
15 – 8	GS[15-8]	R/W	GPIO_P1[7-0] and SERVO[15-8] selection. 0: PINS for GPIO_P1 (default) 1: PINS for SERVO
7 – 0	GS[7-0]	R/W	GPIO_P0[7-0] and SERVO[7-0] selection. 0: PINS for GPIO_P0 (default) 1: PINS for SERVO

Register Offset: D3h – D0h

Register Name: Internal SERVO Control Register

Reset Value: 00000000h

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

Bit	Name	Attribute	Description
31-25	Rsvd	RO	Reserved
24	CLKS	R/W	Servo Clock selection 0: 10MHz (default) 1: 50MHz
23	UE	R/W	Enable/Disable Internal SERVO IO Address Decode 0: Disable (Default) 1: Enable
22-20	Rsvd	RO	Reserved
19-16	SIRT	R/W	SERVO IRQ Routing Table Bit19 Bit18 Bit17 Bit16 Routing Table 0 0 0 0 Disable. 0 0 0 1 IRQ[9] 0 0 1 0 IRQ[3] 0 0 1 1 IRQ[10] 0 1 0 0 IRQ[4] 0 1 0 1 IRQ[5] 0 1 1 0 IRQ[7] 0 1 1 1 IRQ[6] 1 0 0 0 IRQ[1] 1 0 0 1 IRQ[11] 1 0 1 0 Reserved 1 0 1 1 IRQ[12] 1 1 0 0 Reserved 1 1 0 1 IRQ[14] 1 1 1 0 Reserved 1 1 1 1 IRQ[15] These four bits are used to route SERVO IRQ to any 8259 Interrupt lines. The BIOS should be used to inhibit the setting of the reserved value.
15-9	UIOA	R/W	Internal SERVO IO Address. The Bit[15:9] contain the base IO address A[15:9] of internal SERVO.
8-0	Rsvd	RO	Reserved. All are ‘0’s. Writing any value to these bits causes no effect.

Technical Support

For more technical support, please visit http://www.dmp.com.tw/tech or mail to tech@dmp.com.tw.