TEXAS INSTRUMENTS

ADSL CPE Software

DMT ADSL Customer Premise Equipment for ADSL Operations

AR7 SOFTWARE PORTING GUIDE FOR TNETD7300 TO TNETD7200

Version 0.6

Texas Instruments Incorporated
Dallas, Texas.

October 2005

Document Version 0.6

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

DOCUMENT STATUS

In general the symbol "X" has been used in the body of this document to denote numeric parameters that were unavailable at the time of publication

REVISION HISTORY

Revision	Description
0.1 April 25, 2005	Preliminary Version Created by BCIL
0.2 April 26, 2005	Added the Initial Version of the ATM-DSL driver changes.
0.3 April 27, 2005	Updated the Flow-chart description for the DSP Clock Selection.
0.4 April 28, 2005	Added the Clock Diagram, Introduction and Scope.
0.5 April 29, 2005	Updated based on comments.
0.6 October, 2005	Updated the clock selection of the DSP clock for 7100.

Table of contents

1 Introduction

This document describes the changes that are required for the Software between the AR7 Single Chip Router chips TNETD7300 to TNETD7200.

It assumes that the viewer has access to the Reference documents and that they have the capability (tool-chains, boards, etc) for the relevant chips. It also assumes that the viewer has an existing version of working software for the TNETD7300 platform.

This document goes into the details of the software changes, for the affected peripherals and provides specific source code examples to accomplish the task of porting the existing TNETD7300 software over to TNETD7200.

The examples provided are for the changes in the boot-loader, Platform Support Package, CPMAC and the AM-DSL drivers.

1.1 Scope

The information presented here are a result of bring-up /testing the AR7 TNETD7200 EVM.

1.2 References:

- Texas Instruments Inc., TNETD7200 Device Specification version 1.0.
- Texas Instruments Inc., TNETD7300 Device Specification version 1.0.

2 Changes in PSPBoot loader

2.1 CVR Register

The CVR register has been updated in TNETD7200. It value is **0x2B.**

The following macro can be used in order to determine that the system is running on TNETD7200 chip:

```
#define IS_OHIO_CHIP() ( (REG32_R( 0xA8610914,15,0 ) == 0x2b) ? 1:0)
```

2.2 Clock scheme

The clock scheme has been changed in TNETD7200. The clock management memory area has been increased to 512 bytes and resides in **0861:0A00-0861:0BFF**. The block diagram for the new Clock Configuration is shown below:

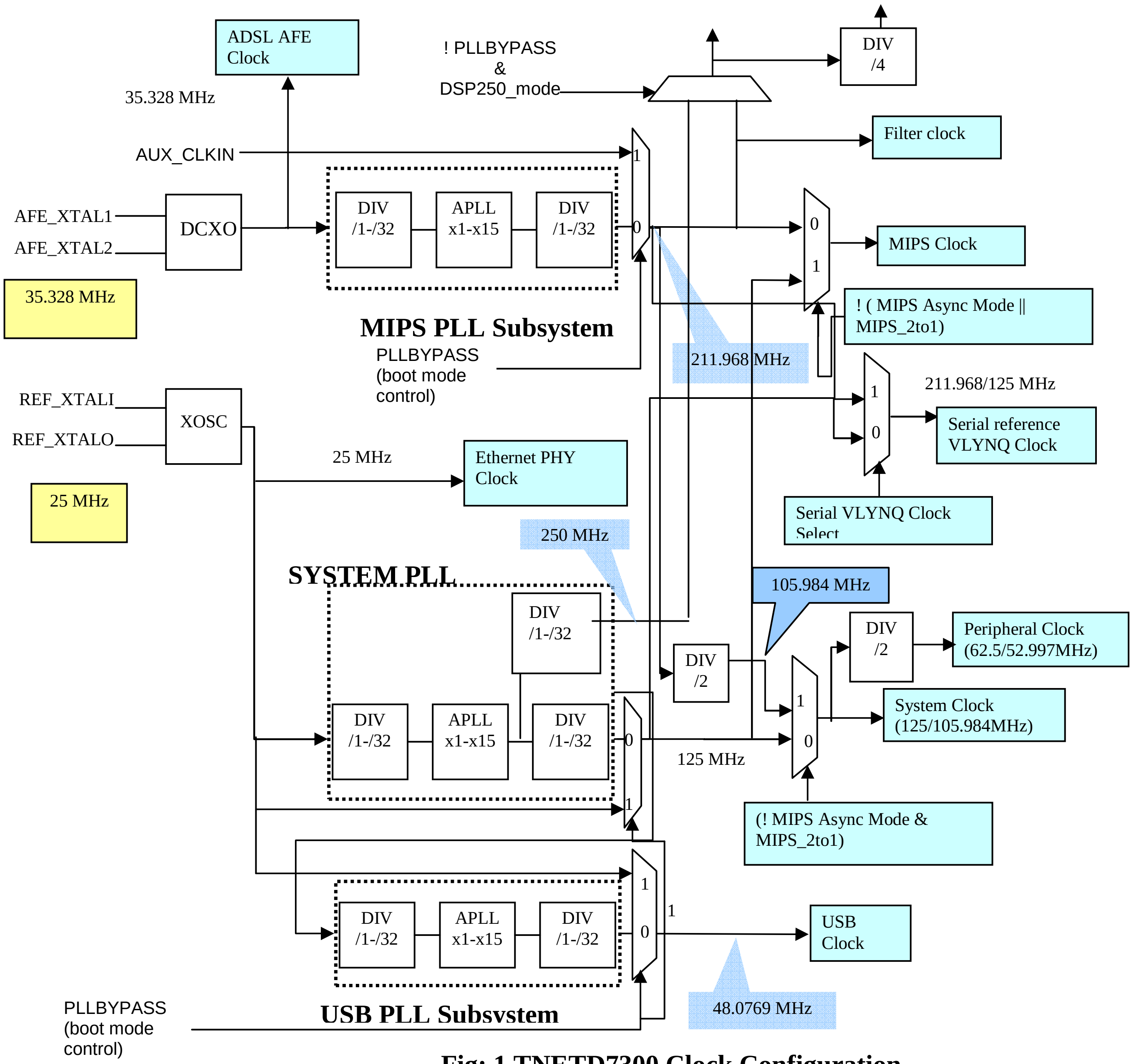

Fig: 1 TNETD7300 Clock Configuration

TNETD7200’s internal clocks are derived from 2 external crystals. These two crystals comprise of 35.326 MHz crystal and the 25 MHz crystal. The 35.326 MHz crystal is connected to the AFE_XTAL1 (AFE_XTALI) and AFE_XTAL2(AFE_XTALO) pins connected to the DCXO module in the AFE subsystem. The 25MHz crystal is connected to ASIC Oscillator cell pair to generate the desired clocks. Each crystal output in the core logic is connected to PLLs, and programmable dividers to create the required internal clocks.

TNETD7200 clock scheme generates 8 different clock frequencies. 35.326 MHz and 25 MHz frequency clocks come from crystal oscillators and 48 MHz, 212 MHz, 106 MHz, 53 MHz, 125 MHz and 62.5 MHz frequency clocks are generated by 3 PLLs and dividers in the clock controller.
Different clocks generated by the clock controller are given below.

The MIPS clock drives the MIPS processor and its internal cache memory subsystem and is derived from either the MIPS PLL sub-chip or the System PLL sub-chip. The desired source of MIPS clock is chosen based on the value of MIPS_2to1 and MIPS_ASYNC boot values latched by the DCL module. The maximum frequency for this clock in asynchronous mode or 2-to-1 mode is 211.968 MHz and 125MHz in 1-to-1 synchronous mode.

The System Clock drives the internal bus structures and EMIF. It is derived from the SYSTEM PLL sub-chip or a divided (divide by 2) version of MIPS PLL sub-chip output. The selection between the two outputs is based on the value of MIPS_2to1 and MIPS_ASYNC boot value latched by the DCL module. The maximum frequency for this clock is 105.984 MHz in 2-to-1 mode and 125MHz in 1-to-1 synchronous and Async modes.

The **mips_async** and **mips2to1** signals are used to select the MIPS and system clocks in the following manner.

mips_async	mips2to1	mips_func_clk	System clock (vbusp_clk)	mode
0	0	125 MHz	125 MHz	sync 1:1
0	1	211.968 MHz	105.984 MHz	sync 2:1
1	x	211.968 MHz	125 MHz	async

The following code determines the clock source, according to MIPS_ASYNC and MIPS_2TO1 bits in BOOTCR register:

```
      bit32u mips_async;

      mips_async = REG32_R(0xa8611a00, 25, 25);

      /* According to Ohio reference doc, page 35 */

      if( mips_async )
      {
        /* Async mode */
        if (clk_id == CLK_ID_OHIO_SYS)
        {
          freq = CONF_REFXTAL_FREQ;
        }
        else
          freq = CONF_AFEXTAL_FREQ;
      }
      else
      {
        /* Sync mode */
        bit32u mips_2to1;

        mips_2to1 = REG32_R(0xa8611a00, 15, 15);

        if( mips_2to1 )
        {
          if (clk_id == CLK_ID_OHIO_SYS)
          {
            freq = CONF_REFXTAL_FREQ;
          }
          else
            freq = CONF_AFEXTAL_FREQ;
        }
        else
        {
          freq = CONF_REFXTAL_FREQ;
        }
      }
```

The following code initializes the **system** and **MIPS** clocks:

```
      bit32u mips_2to1;

      mips_2to1 = REG32_R(0xa8611a00, 15, 15);

      /* Check if sync or async */
      if (mips_async == 0)
      {
        /* Ohio sync mode */
        if (mips_2to1)
        {
          /* Sync 2:1
             In sync 2:1 mode, the MIPS PLL drives both the MIPS
             and System (MIPS/2) clocks.
          */
          cpuf = CPU_2TO1_MAX_F;
          sysf = CPU_2TO1_MAX_F / 2;

          cpuclk_in = get_base_frequency(CLK_ID_OHIO_CPU);
          get_val(cpuf, cpuclk_in, &cpumulti, &cpudiv);

          SioFlush();
          EMIF_DRAMCTL |= 0x80000000;

          /* Set the CpuFrequency PLL */
          setOhioClockPLL(cpumulti - 1, ((cpudiv - 1) & 0x1F), 0, CLK_ID_OHIO_CPU);

          _CpuFrequency = (cpuclk_in * cpumulti) / cpudiv;
          _SysFrequency = _CpuFrequency / 2;

          /* Initialize SYSTEM PLL for USB */
          sysf = SYS_MAX_F;

          sysclk_in = get_base_frequency(CLK_ID_OHIO_SYS);
          get_val(sysf, sysclk_in, &sysmulti, &sysdiv);

          /* Set the SysFrequency PLL and divider */
          setOhioClockPLL(sysmulti - 1, ((sysdiv - 1) & 0x1F), 0, CLK_ID_OHIO_SYS);
        }
        else
        {
          /* Sync 1:1:
             In sync 1:1 mode, the SYSTEM PLL drives both the MIPS and System clocks.
          */
          cpuf = sysf = CPU_SYNC_MAX_F;
          sysclk_in = get_base_frequency(CLK_ID_OHIO_SYS);
          get_val(sysf, sysclk_in, &sysmulti, &sysdiv);

          SioFlush();
          EMIF_DRAMCTL |= 0x80000000;

          /* Set the SysFrequency PLL and divider */
          setOhioClockPLL(sysmulti - 1, ((sysdiv - 1) & 0x1F), 0, CLK_ID_OHIO_SYS);

          _SysFrequency = (sysclk_in * sysmulti) / (sysdiv);
          _CpuFrequency = _SysFrequency;
        }
      }
      else
      {
        /* Async mode
           In async mode (the mips_2to1 boot pin is ignored in async mode),
           both the MIPS and SYSTEM PLLs are used.
        */
        cpuf = CPU_2TO1_MAX_F;
        sysf = SYS_MAX_F;

        cpuclk_in = get_base_frequency(CLK_ID_OHIO_CPU);
        get_val(cpuf, cpuclk_in, &cpumulti, &cpudiv);

        sysclk_in = get_base_frequency(CLK_ID_OHIO_SYS);
        get_val(sysf, sysclk_in, &sysmulti, &sysdiv);

        SioFlush();
        EMIF_DRAMCTL |= 0x80000000;

        /* Set the SysFrequency PLL and divider */
        setOhioClockPLL(sysmulti - 1, ((sysdiv - 1) & 0x1F), 0, CLK_ID_OHIO_SYS);

        /* Set the CpuFrequency PLL */
        setOhioClockPLL(cpumulti - 1, ((cpudiv - 1) & 0x1F), 0, CLK_ID_OHIO_CPU);

        _SysFrequency = (sysclk_in * sysmulti) / (sysdiv);
        _CpuFrequency = (cpuclk_in * cpumulti) / cpudiv;

      }
```

The following code sets up the TNETD7200 PLLS:

```
static void setOhioClockPLL(unsigned int mult, unsigned int post_div, unsigned int pre_div,
unsigned int clock_id)
{
#define DIVEN       0x8000

    volatile unsigned int *postdiv; /* post-div register */
    volatile unsigned int *pllcsr; /* PLL control-status register */
    volatile unsigned int *pllm;   /* PLL multiplier register */
    volatile unsigned int *pllstat; /* PLL status register */
    volatile unsigned int *pllcmd; /* PLL command register */
    volatile unsigned int *pllcmden; /* PLL command enable register */
    volatile unsigned int *prediv; /* PLL command register */

    pllcsr  =  (volatile unsigned int *)(0xa8610a80 + clock_id * 0x80);
    pllm    =  (volatile unsigned int *)(0xa8610a90 + clock_id * 0x80);
    prediv  =  (volatile unsigned int *)(0xa8610a94 + clock_id * 0x80);
    postdiv =  (volatile unsigned int *)(0xa8610a98 + clock_id * 0x80);
    pllcmd  =  (volatile unsigned int *)(0xa8610ab8 + clock_id * 0x80);
    pllcmden = (volatile unsigned int *)(0xa8610ac0 + clock_id * 0x80);
    pllstat =  (volatile unsigned int *)(0xa8610abc + clock_id * 0x80);

    /* Disable the PLL */
    *pllcsr = 0;

    /* write 0 to pre-div ratio. We do not want to use it */
    *prediv = DIVEN | (pre_div & 0x1F);

    /* Set the frequency PLL */
    *pllm = mult & 0xF;

    /* Wait 1500 clock cycles. Note: This is more than that */
    clkc_delay(1500);

    /* Ensure that the GOSTAT bit is ‘0’ to indicate that the PLL
     * output clock alignment is not in progress.
     */
    while ( *pllstat & 0x1);

    /* Set and enable the divider the (post) divider */
    *postdiv = DIVEN | (post_div & 0x1F);

    /* set the GOSET bit in PLLCMDEN register */
    *pllcmden = 0x1;

    /* set the GOSET bit in PLLCMD register */
    *pllcmd = 0x1;

    /* Ensure that the GOSTAT bit is ‘0’ to indicate that the PLL
     * output clock alignment is not in progress.
     */
    while ( *pllstat & 0x1);

    /* Enable the PLL */
    *pllcsr |= 0x1;
}
```

See psp_boot/psbl/kernel/clkc.c file for more details.

2.3 CPGMAC MAC address and PHY0 setup

The TNETD7200 chip has only 1 Ethernet interface (CPGMAC0). The MAC address setup process has been changed in TNETD7200. Since TNETD7300 has CPMAC and TNETD7200 has CPGMAC, the following code is used to initialize the CPGMAC MAC address correctly:

```
#define CPMAC_BASE    0x08610000

            /* Ohio CPGMAC MAC address setup */
            REG32_W(CPMAC_BASE + 0x508, 0);
            REG32_W(CPMAC_BASE + 0x504, sys_et_addr[0] | (sys_et_addr[1] << 8) |
                            (sys_et_addr[2] << 16) | (sys_et_addr[3] << 24));
            REG32_W(CPMAC_BASE + 0x500, (1 << 20) | (1 << 19) | (0 << 16) |
                            sys_et_addr[4] | (sys_et_addr[5] << 8));
```

Where sys_et_addr variable contains the MAC address.

For AR7L0 based boards, there is a need to take Phy 0 out of reset, and in AR7WRD/VWi based boards (with an external phy), Phy 0 must be kept in reset and MII pins must be initialized. The following code demonstrates:

```
#define AVALANCHE_MII_SEL_REG         (0xa8611A08)
#define RESET_BASE (0xa8611600)

   #if defined (AR7VWi) || defined (AR7WRD) /* Ohio boards */
                  /* Ohio has only one EPHY - Reset EPHY0*/
                  REG32_RMW(RESET_BASE, 26, 26, 0);
                  delay_usecs(200);

                  /* MII pins are connected to the MII interface on the EMAC0 module.*/
                  REG32_RMW(AVALANCHE_MII_SEL_REG,0,0,1);
    #else /* Sangam boards */

        /* Phy 0 out of reset */
        REG32_RMW(RESET_BASE, 26, 26, 1);
        delay_usecs(200);

    #endif /* defined (AR7VWi) || defined (AR7WRD) */
```

See psp_boot/psbl/kernel/main.c and psp_boot/psbl/net/cpmac.c files for more details.

2.4 Serial (UART) interface

TNETD7200 has only 1 UART port instead of 2 in TNETD7300.
Moreover, in order to use UART0, there is a need to take the GPIO module out of reset due to the fact that they share pins.

The following macro makes sure that UART0 is properly started:

```
#define SIO1_BASE     0xa8610f00
#define SIO1_RSTMASK  0x02
#define SIO0_BASE     0xa8610e00
#define SIO0_RSTMASK  0x01

#define SIO_RSTMASK(base)   ((base == SIO0_BASE)? IS_OHIO_CHIP() ? SIO0_RSTMASK | 0x40 :
SIO0_RSTMASK : SIO1_RSTMASK)
```

There is no need to initialize UART1 if TNETD7200 is detected.

2.5 DSL boot code removal

In the PSPBoot platform_init.s file, there is a jump to a memory location that initializes a DSL patch that works only on TNETD7300. The following code has been introduced in order to skip it, in case of TNETD7200:

```
/* Software workaround for Sangam DSL sub-sys, skip this code in Ohio - 7100 CVR=0x18 / 7200
CVR=0x2b */
    li      t0, 0xa8610914
    lw      t1, 0(t0)
    lw      t0, 0(t0)
    andi    t0,     t0,     0xffff
    andi    t1,     t1,     0xffff
    sub     t0, t0, 0x2b
    sub     t1, t1, 0x18
test_if_7200:
    bnez    t0, test_if_7100
    nop
    j               done_lpr
    nop
test_if_7100:
    bnez    t1, sangam_dsl_workaround
    nop
    j               done_lpr
    nop

sangam_dsl_workaround:
….
```

3 Changes in PSP Linux image

3.1 CVR Register

The CVR register has been updated in TNETD7200. Its value is **0x2B.**

The following macro can be used in order to determine that the system is running on TNETD7200 chip:

```
#define IS_OHIO_CHIP() ( (REG32_R( 0xA8610914,15,0 ) == 0x2b) ? 1:0)
```

3.2 Clock Configuration

The Linux kernel detects if it is TNETD7200 chip on run time and when TNETD7200 detected it uses a special clock configuration function for TNETD7200.

3.2.1 Clock Initialization

In the clock initialization function:

- avalanche_clkc_init , in DSL-PSP 4.5 and previous version.
- Or PAL_sysClkcInit , in DSL-PSP4.6 and upper version.

If TNETD7200 chip is detected the clock initialization function should call to ohioClkInit and do not perform the same initialization as it does for TNETD7300.

```
void PAL_sysClkcInit(void* param)
{
     UINT32 choice;

     PAL_SYS_Tnetd73xxInit* ptr = (PAL_SYS_Tnetd73xxInit*)param;
     afeclk_inp = ptr->afeclk;
     refclk_inp = ptr->refclk;
     xtal_inp      = ptr->xtal3in;

     bootcr_reg = (volatile int*)AVALANCHE_DCL_BOOTCR;

     if(IS_OHIO_CHIP())
     {
            ohioClkInit(); /*OHIO clock initialization */
     }else
     {
         … /* SANGAM clock initialization */
     }
}

#define BOOTCR_OHIO_MIPS_MIPS2TO1_MODE  (1 << 15)
static void ohioClkInit()
{

    unsigned int bootcr_reg = (volatile int*)AVALANCHE_DCL_BOOTCR;

    clk_pll_src[CLKC_MIPS] = &afeclk_inp;
    clk_pll_src[CLKC_VBUS] = &afeclk_inp;
    clk_pll_src[CLKC_USB]  =  &refclk_inp;
    //ohio_mips_clk = 211968000;
    /* MIPS_ASYNC  MIPS_2to1   MIPS CLOCK    SYSTEM CLOCK   */
    /*  0           0           125 MHz        125 MHz      */
    /*  0           1          211.968 MHz     105.984 MHz  */
    /*  1           -          211.968 MHz     125 MHz      */

    if(BOOTCR_MIPS_ASYNC_MODE & REG32_DATA(AVALANCHE_DCL_BOOTCR))
            { /* Ohio Async Mode */
            ohio_mips_freq = OHIO_MIPS_MAX_FREQ;
            ohio_sys_freq  = OHIO_SYS_MAX_FREQ;

    }
    else
            { /* Ohio Sync Mode */

            /*    MIPS 2 To 1 BIT
                             0 - The internal system bus frequency of operation is equal to
                                     frequency of the MIPS processor clock. In this mode the
MIPS
                                     operates at a frequency of 125Mhz.
                             1 - The internal system bus frequency of operation is equal to
half
                                    of the MIPS processor clock
*/

            if(BOOTCR_OHIO_MIPS_MIPS2TO1_MODE & REG32_DATA(AVALANCHE_DCL_BOOTCR))
                    {
                    ohio_mips_freq = OHIO_MIPS_MAX_FREQ;
                    ohio_sys_freq  = ohio_mips_freq/2;
            }
            else
                    {
                    /* in this case the system clock is equal to MIPS clock*/
                    ohio_mips_freq = OHIO_SYS_MAX_FREQ;
                    ohio_sys_freq  = ohio_mips_freq;
            }
    }

    return;
}
```

3.2.2 USB Clock Initialization

For USB clock Configuration the Linux kernel uses the same function as the boot loader setOhioClockPLL and uses the following values:

```
#define CLK_ID_OHIO_USB_DIV     1
#define CLK_ID_OHIO_USB_PRE_DIV 13
#define CLK_ID_OHIO_USB_MULT    5

int PAL_sysClkcSetFreq (PAL_SYS_CLKC_ID_T clk_id, unsigned int output_freq)
{
    int pll_id;

    if(clk_id >= CLKC_NUM)
        return -1;

    pll_id = clk_to_pll[clk_id];

    if(pll_id == CLKC_VBUS)
        return -1;

    if(IS_OHIO_CHIP() && (clk_id != CLKC_USB))
    {
           /* There is no flexible to change the MIPS and SYS clock in Ohio*/
           /* Use the default configuration of the boot loader             */
           return -1;
    }

    /* To set USB clock to exactly 48 MHz */
    if( (clk_id == CLKC_USB) && (CLK_MHZ(48) == output_freq) )
    {
           if(IS_OHIO_CHIP())
      {
        setOhioClockPLL(CLK_ID_OHIO_USB_MULT-1,CLK_ID_OHIO_USB_PRE_DIV-1,CLK_ID_OHIO_USB_DIV-
1,clk_id);
           return 0;
      }else
        usb_clk_check();

    }

    return set_pll_div(pll_id,output_freq);
}


….
```

3.2.3 Get clock frequency function changes.

Change in the get clock frequency function:

```
int PAL_sysClkcGetFreq(PAL_SYS_CLKC_ID_T clk_id)
{
    int pll_id;
    int output_freq;

    if(clk_id >= CLKC_NUM)
    {
        return -1;
    }

    if(IS_OHIO_CHIP())
    {
                   switch(clk_id)
                   {
                   case CLKC_SYS:
                           return (ohio_sys_freq);

                   case CLKC_VBUS:
                           return (ohio_sys_freq/2);

                   case CLKC_MIPS:
                           return (ohio_mips_freq);

                   default:
                           return -1;

                   }
     }

    pll_id = clk_to_pll[clk_id];

    output_freq = get_pll_div(pll_id);

    if(clk_id == CLKC_VBUS)
           output_freq >>= 1;

    return output_freq;

}

….
```

3.3 CPMAC configuration changes.

The TNETD7200 chip has only 1 Ethernet interface (CPGMAC0).
Currently there are two compilation options which indicate if the board uses the Low CPMAC or High CPMAC interface.
Since TNETD7200 has only one CPAMAC interface, when TNETD7200 is detected the SW uses the definitions of Low CPMAC instead of High CPMAC definitions in run time.
Following the changes in the file sangam.h:

```

#define AVALANCHE_LOW_CPMAC_RESET_BIT             17
/* OHIO has only CPMAC 0                          */
#define AVALANCHE_HIGH_CPMAC_RESET_BIT            (IS_OHIO_CHIP()?
AVALANCHE_LOW_CPMAC_RESET_BIT : 21)

#define AVALANCHE_LOW_CPMAC_BASE              (KSEG1_ADDR(0x08610000)) /* AVALANCHE CPMAC 0
*/
/* AVALANCHE CPMAC 1 - OHIO has only CPMAC 0*/
#define AVALANCHE_HIGH_CPMAC_BASE             (IS_OHIO_CHIP() ? AVALANCHE_LOW_CPMAC_BASE :
(KSEG1_ADDR(0x08612800)) )

#define AVALANCHE_LOW_CPMAC_INT                   19
/* OHIO has only CPMAC 0                          */
#define AVALANCHE_HIGH_CPMAC_INT                  (IS_OHIO_CHIP() ? AVALANCHE_LOW_CPMAC_INT :
33)

….
```

Following the changes in the file sangam_board.h:

```
#if defined(CONFIG_MIPS_AR7VWI) || defined(AR7VWi) || defined(CONFIG_MIPS_AR7VW) ||
defined(AR7VW) \
 || defined(CONFIG_MIPS_AR7WRD) || defined(AR7WRD)
#define AFECLK_FREQ                                 35328000
#define REFCLK_FREQ                                 25000000
#define OSC3_FREQ                                   24000000
#define AVALANCHE_LOW_CPMAC_PHY_MASK                0x80000000
/* For OHIO use always CPMAC0                     */
#define AVALANCHE_HIGH_CPMAC_PHY_MASK               (IS_OHIO_CHIP() ?
AVALANCHE_LOW_CPMAC_PHY_MASK : 0x00010000)
#define AVALANCHE_LOW_CPMAC_MDIX_MASK               0x80000000
#define AVALANCHE_LOW_CPMAC_HAS_EXT_SWITCH          (IS_OHIO_CHIP() ?  1 : 0)
#define AVALANCHE_HIGH_CPMAC_HAS_EXT_SWITCH         1
#define VLYNQ0_RESET_GPIO_NUM                       18
#define AVALANCHE_NUM_VLYNQ_HOPS_PER_ROOT           1
#endif
```

3.4 ATM-DSL changes

Due to the different chips that have to be supported by the ATM-DSL driver and datapump software, there are some changes in the ATM-DSL driver to accommodate the different chips and their configurations. There are 3 main changes in the ATM-DSL driver that are required:

1. Change in the Interrupt vector number for the DSL interrupt.
2. Change in the way the DSP and SAR frequencies are configured, with added support for boosting the DSP frequency to 250 MHz on some chips.
3. Change in the way that End of Interrupt (EOI) is signaled after servicing the SAR interrupt.

3.4.1 Change in the Interrupt vector number for the DSL interrupt

The DSL interrupt vector number has been changed from 39 on TNETD7300 to 23 on TNETD7200. This is something that has to be configured for the different chips.

```
#define ATM_DSL_INT_SANGAM    39      /* For TNETD7300 */
#define ATM_DSL_INT_OHIO      23      /* For TNETD7200  */
```

3.4.2 Change in the way the DSP frequency is configured

Note: In order to use the TNETD7200 chip, the DSL-HAL and Datapump versions have to be 4.0.1.0 or higher.

The DSP frequency and the SAR frequency can be boosted to 250MHz and 62.5MHz respectively for TNETD7200, as it was for Sangam-250 (7300C). The default DSP frequency on TNETD7200 is 212 MHz and the SAR frequency is 1/4th of that 53MHz. This is something that has to be configured from the driver, by calling the appropriate DSL-HAL functions.

The sequence that should be followed is:

a. The ATM-DSL driver initialization routine must implement the clock selection algorithm that is shown in the flowchart below, to determine whether it needs to boost the DSP and SAR clocks. If it return TRUE, then the driver should call the dslhal_api_boostDspFrequency() to boost the DSP clock as part of the initialization process.

b. In dslhal_api_dslStartup(), the triggerDsp250MHZ flag is checked, if it's set (1), will call an internal function dslhal_support_setDsp250MhzTrigger(), which in turn, will set mhzFlag in DSP-Host interface to 1, that indicates to the datapump what clock the DSP is running at.

The flowchart used for the clock selection is:

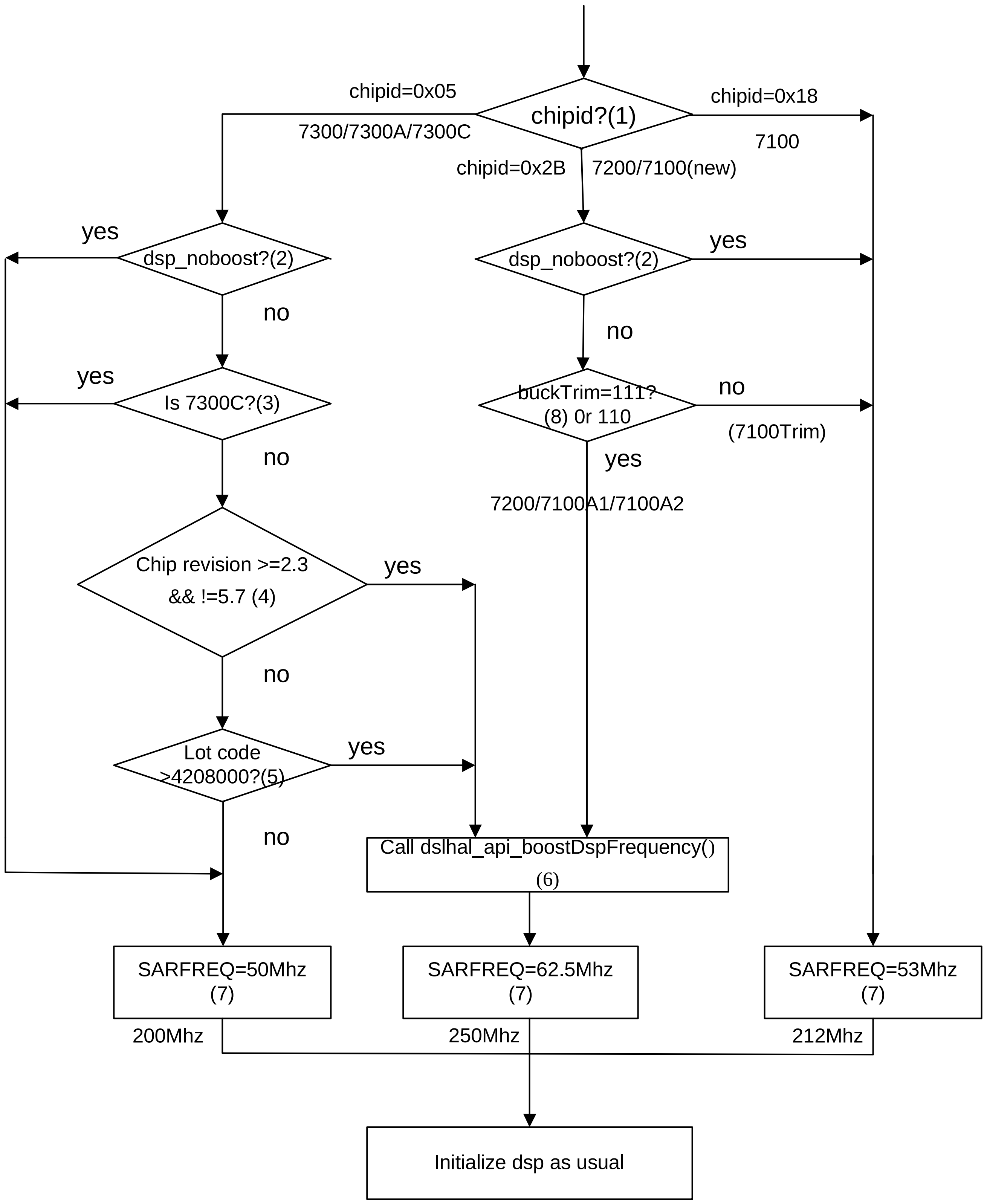

Fig: 2 Flowchart for the Clock selection Algorithm for TNETD7300 and TNETD7200

Where the detailed description of the steps is:

(1) Check ChipID: read address 0x08610914 (CVR), bit 15-0. Value could be 0x05 (7300/7300A/7300C), 0x18 (7100) or 0x2B (7200).

(2) Verify the chip needs to run at a higher DSP frequency (not shown on the diagram).

(3) Check if 7300C by reading address 0x0861091C (DIDR2), bit 23 (mask: 0x00800000) is 1 for 7300C.

(4) If device is 7300/7300A and revision 2.3 or higher and not 5.7, then it can use 250Mhz. Check revision by reading address 0x08610914 (CVR), bit 23-16, value 0x23 or larger, but not 0x57

(5) Check Lot Code make sure it's greater than 4208000.
Lot Code is calculated by

```
#define DIEIDLO (*(volatile unsigned int *) 0xA8610918
#define DIEIDHI (*(volatile unsigned int *) 0xA861091C

LotCode = ((DIEIDHI &0x1FFF)<<10)|(( DIEIDLO &0xFFC00000)>>22);
```

(6) If device is capable of running dsp at higher frequency then it shall call the dslhal_api_boostDspFrequency API prior to the dslhal_api_dslStartup API to configure the DSP to run at 250Mhz frequency.

(7) SAR frequency is DSP frequency divided by 4. If DSP is running at 250MHz, adjust SAR frequency to "62,500,000" in SAR setup and QoS calculation. Otherwise SAR clock is set to un-boosted "50,000,000" (dspclock=200Mhz) or "53,000,000" (dspclock=212Mhz) depends on chip ID.

(8) Check Buck Trim bits by the following sequence:

```
*((voltatile unsigned int *) 0xA8611600) =  ~0x00800000; /* Reset DSP */
*((volatile unsigned int *) 0xA1000600)  = 0x40;        /* unreset DSL-SPA */
regValue = *((volatile unsigned int ) 0xA1040018); /* read Trim Status at 32 bit boundary */

buckTrim = (regValue>>8)  & 0x07;        /* get PM_STATUS1 at A1040019 */

if (buckTrim==0x07)       /* '111' * /
{
    /* device is 7200/7100A2 */
}
else if (buckTrim==0x06)  /* '110' */
{
    /* device is 7100A1 *;
}
else
{
    /* Unsupported device. */
}
```

3.4.3 Change in the way that End Of Interrupt (EOI) is signaled

Note: In order to use the TNETD7200 chip, the CPSAR version has to be 1.7.2.a or higher.

The End of Interrupt (EOI) register which has to be written to at the end of every SAR interrupt, has to check whether the return code from the *halIsr* function is not *EC_NO_ERRORS*. If that is not the case then the *PacketProcessEnd* function, which signals the EOI shouldn't be called.

The following code snippet shows how this can be done:

```
rc = halIsr(((HAL_DEVICE *)priv->pSarHalDev, more);

/* Do the EOI only if no errors are returned. */
if(rc == EC_NO_ERRORS)
{
    ((HAL_FUNCTIONS *) priv->pSarHalFunc)->PacketProcessEnd((HAL_DEVICE *)priv->pSarHalDev);
}
```