

- 7.

7.

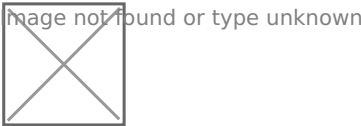
“ ”

SOC

6.1 IMX6ULL

6.1.1

imx6ull biased amplifier 24MHZ 32



imx6ull XTALI XTALO XTALOSC24M 24MHZ 24MHZ XTALC

imx6ull 24MHZ RC RTC 32KHZ XTALOSC24M RC

imx6ull RTC_XTALI RTC_XTALO 32KHZ 32.768KHZ RTC_XTAL 32KHZ RTC ir

TAL

RTC XTALOSC24M

6.1.2

XTALOSC24M 24MHZ

PLL PD LPF VCO PD LPF VCO

Image not found or type unknown

imx6ull 7 XTALOSC24M 24MHZ

image not found or type unknown



PLL

1. PLL1

ARM_PLL ARM 1.3GHZ 1GHZ

2. PLL2

SYS_PLL 528_PLL x22 XTALOSC24M 24MHZ 528MHZ SYS_

3. PLL3

USB1_PLL USB USBPHY1 x20 24MHZ 480MHZ USB1_PL

4. PLL4

AUDIO_PLL AUDIO_PLL 650MHZ 1300MHZ 1HZ

5. PLL5

VIDEO_PLL VIDEO_PLL 650MHZ 1300MHZ 1H

6. PLL6

ENET_PLL x20+(5/6) 24MHZ 500MHZ 1 50MHZ 25MHZ

7. PLL7

USB2_PLL USB USBPHY2 x20 480MHZ

3

1) Bypass PLL BYPASS

2) bypass PLL ENABLE

3) PLL POWERDOWN

ARM_PLL PLL 1 ref_armpll_clk Bypass PLL 2

image not found or type unknown



SYS_PLL USB1_PLL

PFD

PLL

PFD

6.1.3

imx6ull

PFD

bypass

PLL1 PLL3

pll1_sw_clk pll3_sw_clk

PLL4 PLL5

pll4_main

image not found or type unknown



PLL

switcher

CPU

CCSR[pll1_sw_clk_sel] pll1_sw_clk

multiplexer

CPU

PLL2

image not found or type unknown



CCM

6.2

imx6ull

CCM ANALOG_DIG

AIPS-1

image not found or type unknown



ANALOG_DIG

CCM_ANALOG_PLL_xxx

PLL

CCM_ANALOG

ANALOG_DIG

PMU

CCM_ANALOG_MISCx

PMU

PMU_MISCx x = 0-2

CCM

PLL PFD

image not found or type unknown



CCM_CCGRx x = 0-6

image not found or type unknown



6.2.1

imx6ull 7 ENET_PLL ARM_PLL

image not found or type unknown



BYPASS ENABLE POWERDOWN PLL bypass BYPASS_CLK_SRC
1 LOCK 1

ARM_PLL Fref * DIV_SELECT/2 PLL ARM_PLL USB1_PLL

AUDIO_PLL VIDEO_PLL NUM DENOM AUDIO_PLL

1 CCM_ANALOG_PLL_AUDIO_NUM

image not found or type unknown



2 CCM_ANALOG_PLL_AUDIO_DENOM

image not found or type unknown



Fref * (DIV_SELECT +
NUM/DENOM) AUDIO_PLL VIDEO_PLL switcher /1 /2 /4 /8 /16
_VIDEO CCM_ANALOG_MISC2

SYS_PLL USB1_PLL SYS_PLL USB1_PLL CCM_ANALOG_PFD_48

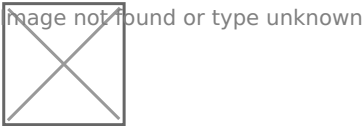
image not found or type unknown



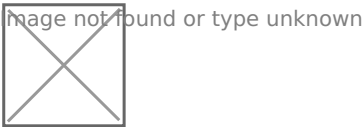
PFDco*18/PFD_FRAC Fvco PLL PFD_FRAC 12 35

6.2.2

bypass PFD switcher root generator

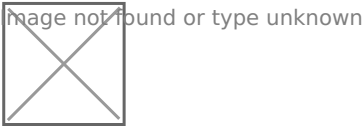


CCM ARM pll1_sw_clk CCM_CACRR



ARM pll1_sw_clk/(ARM_PODF + 1)

CCM_CBCDR



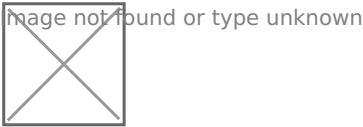
imx6ull CCM

6.2.3

imx6ull

- RUN CCM_CLPCR[LPM] 0 CPU CCGRx
- WAIT CCM_CLPCR[LPM] 1 CPU WFI WAIT CPU CCGRx
- STOP CCM_CLPCR[LPM] 2 STOP WAIT PLL CCM_CLPCR[SBYOS]

imx6ull CCGRx x = 0-6 CCGRx



CCGR0 CG15 gpio2_clocks CG14 uart2_clock

CGx x=0-15	
00	
01	RUN
10	
11	RUN WAIT STOP

6.3

6.3.1 CPU

CPU	CPU	81MHZ	648MHZ	CPU	LED
CPU	ARM_PLL	CPU		XTALOSC24M	ARM_

image not found or type unknown



6.3.1.1 PLL1_SW_CLK

CPU pll1_sw_clk 1 2 ARM_PLL PLL1 1
 sel_pll1) sel_pll1 0 2 sel_pll1 0 1 set_pll1_sw_clk switcher.c

```

4 extern struct ccm_regs *ccm;
5
  /*****
  *      set_pll1_sw_clk
  *      PLL1_SW_CLK
  *      sel_pll1: 0- XTALOSC24M  1- PLL1
  *
  *
  *
  * -----
  * 2020/03/08      V1.0
  *****/
  void sel_pll1_sw_clk(int sel_pll1)
{
  /* PLL1_SW_CLK_SEL: 0 pll1_main_clk 1 step_clk */
  if (sel_pll1)
    clr_bit(&ccm->ccsr, 2);      /* pll1_main_clk */
  else {
    clr_bit(&ccm->ccsr, 8);      /* step_clk  OSC */
    set_bit(&ccm->ccsr, 2);      /* step_clk */
  }
}

```

```

}
}

```

6.3.1.2 ARM_PLL

pll1_sw_clk

ARM_PLL

set_pll

PLL

AUDIO_P

```

struct anadig_regs *anadig = (struct anadig_regs *)ANADIG_BASE_ADDR;

static void wait_to_lock(u32 *pll_reg)
{
    while (read32(pll_reg) & LOCK_MASK == 0);    /* PLL */
}

/*****
 *      set_pll
 *      PLL
 *      pll:  PLL    div:  PLL
 *
 *
 *
 * -----
 * 2020/03/08    V1.0
 *****/

void set_pll(pll_e pll, u32 div)
{
    switch (pll) {
        case ARM_PLL:
            if (div < 54 && div > 108) return;    /* ARM_PLL    54 108 */
            write32(ENABLE_MASK | div, &anadig->analog_pll_arm);
            wait_to_lock(&anadig->analog_pll_arm);    /* ARM_PLL */
            break;

        case USB1_PLL:    /* USB1_PLL */
            write32(ENABLE_MASK | (div&0x3), &anadig->analog_pll_usb1);
            wait_to_lock(&anadig->analog_pll_usb1);
            break;

        case USB2_PLL:    /* USB2_PLL */
            write32(ENABLE_MASK | (div&0x3), &anadig->analog_pll_usb2);
    }
}

```



```

        wait_to_lock(&anadig->analog_pll_usb2);
        break;

case SYS_PLL:          /* SYS_PLL          */
    write32(ENABLE_MASK | (div&0x1), &anadig->analog_pll_sys);
    wait_to_lock(&anadig->analog_pll_sys);
    break;

case AUDIO_PLL:
    if (div < 27 && div > 54) return;          /* AUDIO_PLL          27 54 */

    /* AUDIO_PLL NUM DENOM 0xF */
    write32(0xF, &anadig->analog_pll_video_num);
    write32(0xF, &anadig->analog_pll_video_denom);

    write32(ENABLE_MASK | div, &anadig->analog_pll_video);
    wait_to_lock(&anadig->analog_pll_video); /* AUDIO_PLL */
    break;

case VIDEO_PLL:
    if (div < 27 && div > 54) return;          /* VIDEO_PLL          27 54 */

    /* VIDEO_PLL NUM DENOM 0xF */
    write32(0xF, &anadig->analog_pll_audio_num);
    write32(0xF, &anadig->analog_pll_audio_denom);

    write32(ENABLE_MASK | div, &anadig->analog_pll_video);
    wait_to_lock(&anadig->analog_pll_video); /* VIDEO_PLL */
    break;

case ENET_PLL:
    /* ENET_PLL          PLL          */
    break;
}
}

```

6.3.1.3 ARM_CLK_ROOT

pll1_sw_clk arm_clk_root CPU root generator setup_arm_pll

```

extern struct ccm_regs *ccm;

/*****
 *      setup_arm_podf
 *      ARM_CLK_ROOT
 *      1-8 ARM_CLK_ROOT = PLL1_SW_CLK / PODF
 *
 *
 *
 * -----
 * 2020/03/08      V1.0
 *****/

void setup_arm_podf(u32 podf)
{
    if (podf < 1 || podf > 8) return;    /* ARM_PODF      1 8 */
    write32(podf-1, &ccm->cacrr);
}

```

6.3.1.4 led

	led	led	led_toggle	led
main	led	CPU	81MHZ ARM_PLL	648MHZ
			8	5
		CPU	648MHZ	

```

#include "regs.h"
#include "pll.h"
#include "clkroot.h"

struct ccm_regs *ccm = (struct ccm_regs *)CCM_BASE_ADDR;

#define LOOPS 1000000
static void busy_wait(void)
{
    for(u32 i = 0; i < LOOPS; i++); /*      */
}

/* LED      */
extern void led_init(void);
extern void led_toggle(void);
void led_on(void);

```

```

/* PLL1      PODF      */
extern void setup_arm_podf(u32 podf);
extern void sel_pll1_sw_clk(int sel_pll1);

void main(void)
{
    /*      LED */
    int blinks = 0;
    led_init();
    led_on();

    sel_pll1_sw_clk(0);    /* ARM_ROOTT    OSC */
    setup_arm_podf(8);     /* ARM_ROOTT      8 */
    set_pll(ARM_PLL, 54);  /* ARM_PLL: 24*54/2 = 648MHZ, ARM_ROOTT: 81MHZ */
    sel_pll1_sw_clk(1);    /* ARM_ROOTT  ARM_PLL  CPU    81MHZ */

    /*    / LED 10    LED    */
    for (blinks = 10; blinks > 0; blinks--)
    {
        busy_wait();
        led_toggle();
    }

    sel_pll1_sw_clk(0);    /* ARM_ROOTT    OSC */
    setup_arm_podf(2);     /* ARM_ROOTT      2 */
    set_pll(ARM_PLL, 108); /* ARM_PLL: 24*108/2 = 1296MHZ, ARM_ROOTT: 648MHZ */
    sel_pll1_sw_clk(1);    /* ARM_ROOTT  ARM_PLL  CPU    648MHZ */

    /*    / LED    LED    */
    while(1)
    {
        busy_wait();
        led_toggle();
    }
}

/*      GCC      raise      */
void raise(void)
{

```

```
}
GCC raise
```

```
: Git NoosProgramProject/(7_ /fastcpu)
```

6.2.1.4 4-1.4

6.2.1.5 4-1.4

led

6.3.2

led CPU imx6ull uart u

PLL ENET_PLL PFD

6.3.2.1 PLL

PLL PFD pll.c

```
/*
 * get_pll
 * PLL
 * pll: PLL
 *
 * PLL
 *
 * -----
 * 2020/03/08 V1.0
 */
u32 get_pll(pll_e pll)
{
    u32 div, post_div, pll_num, pll_denom;

    switch (pll) {
        case ARM_PLL:
            div = read32(&anadig->analog_pll_arm);
            if (div & BYPASS_MASK) /* ARM_PLL Bypass */
                return CKIH;
            else {
```

```

        div &= 0x7F;          /* ARM_PLL */
        return (CKIH * div) >> 1; /* ARM_PLL */
    }

case USB1_PLL:
    div = read32(&anadig->analog_pll_usb1);
    if (div & BYPASS_MASK) /* USB1_PLL Bypass */
        return CKIH;
    else {
        div = div&0x1 ? 22 : 20; /* USB1_PLL 1 x22 0 x20 */
        return CKIH * div;
    }

case USB2_PLL:
    div = read32(&anadig->analog_pll_usb2);
    if (div & BYPASS_MASK) /* USB2_PLL Bypass */
        return CKIH;
    else {
        div = div&0x1 ? 22 : 20; /* USB2_PLL 1 x22 0 x20 */
        return CKIH * div;
    }

case SYS_PLL:
    div = read32(&anadig->analog_pll_sys);
    if (div & BYPASS_MASK) /* SYS_PLL Bypass */
        return CKIH;
    else {
        div = div&0x1 ? 22 : 20; /* SYS_PLL 1 x22 0 x20 */
        return CKIH * div;
    }

case AUDIO_PLL:
    div = read32(&anadig->analog_pll_audio);
    if (!(div & ENABLE_MASK)) /* AUDIO_PLL */
        return 0;

    if (div & BYPASS_MASK) /* AUDIO_PLL Bypass */
        return CKIH;
    else {
        post_div = (div & 0x3) >> 19;

```

```

        if (post_div == 3)          /* reserved value */
            return 0;

        /* AUDIO_PLL      0    4 1    2 2    1 */
        post_div = 1 << (2 - post_div);

        pll_num = read32(&anadig->analog_pll_audio_num);
        pll_denom = read32(&anadig->analog_pll_audio_denom);

        return CKIH * (div + pll_num / pll_denom) / post_div;
    }

case VIDEO_PLL:
    div = read32(&anadig->analog_pll_video);
    if (!(div & ENABLE_MASK))      /* VIDEO_PLL          */
        return 0;

    if (div & BYPASS_MASK)         /* VIDEO_PLL          */
        return CKIH;
    else {
        post_div = (div & 0x3) >> 19;
        if (post_div == 3)        /* reserved value */
            return 0;

        /* VIDEO_PLL      0    4 1    2 2    1 */
        post_div = 1 << (2 - post_div);

        pll_num = read32(&anadig->analog_pll_video_num);
        pll_denom = read32(&anadig->analog_pll_video_denom);

        return CKIH * (div + pll_num / pll_denom) / post_div;
    }

default:
    return 0;
}

/* NOTREACHED */
}

static void set_pfd(u32 *reg, pfd_e pfd, int gate, u32 frac)
{
    u32 value = read32(reg);      /* PLL PFD          */

```

```

    value &= ~PFD_MASK(pfd);
    if (gate) value |= PFD_GATE_MASK(pfd); /*      PFD      */
    value |= (frac<<PFD_SHIFT(pfd)) & PFD_FRAC_MASK(pfd); /*      PFD      */
    write32(value, reg);

    while(read32(reg) & PFD_STABLE_MASK(pfd));
}

/*****
 *      set_pll_pfd
 *      SYS_PLL USB1_PLL PFD
 *      pll:  PLL  pfd:  PFD  gate:      PFD  frac: PFD
 *
 *
 *
 * -----
 * 2020/03/08      V1.0
 *****/
void set_pll_pfd(pll_e pll, pfd_e pfd, int gate, u32 frac)
{
    u32 *reg;
    if (pll = SYS_PLL)
        reg = &anadig->analog_pfd_528;
    else if (pll = USB1_PLL)
        reg = &anadig->analog_pfd_480;
    else
        /*  SYS_PLL USB1_PLL  PFD  */
        return ;

    set_pfd(reg, pfd, gate, frac);
}

/*****
 *      get_pll_pfd
 *      SYS_PLL USB1_PLL PFD
 *      pll:  PLL  pfd:  PFD
 *
 *
 *
 * -----

```

```

* 2020/03/08      V1.0
*****/

u32 get_pll_pfd(pll_e pll, pfd_e pfd)
{
    u32 div;
    u64 freq;

    switch (pll) {
        case SYS_PLL:
            div = read32(&anadig->analog_pfd_528);
            freq = (u64)get_pll(SYS_PLL);
            break;
        case USB1_PLL:
            div = read32(&anadig->analog_pfd_480);
            freq = (u64)get_pll(USB1_PLL);
            break;
        default:
            /* SYS_PLL USB1_PLL PFD */
            return 0;
    }

    /* PFD      fPLL x 18 / N N PFD      */
    return (freq * 18) / PFD_FRAC_VALUE(div, pfd);
}

```

6.3.2.2 PLL1_SW_CLK

switcher switcher.c

```

/*****
*      get_pll1_sw_clk
*      PLL1_SW_CLK
*
*
*      PLL1_SW_CLK
*
* -----
* 2020/03/08      V1.0
*****/

u32 get_pll1_sw_clk(void)

```



```

{
    u32 reg = read32(&ccm->ccsr);

    if (reg & (1u<<2)) {          /* PLL1_SW_CLK_SEL 0 pll1_main_clk 1 step_clk */
        if (reg & (1u<<8)) {      /* STEP_SEL 1 secondary_clk, 0 OSC */
            if (reg & (1u<<3))    /* SECONDARY_CLK_SEL 1 PLL2 0 PLL2 PFD2 */
                return get_pll(SYS_PLL);
            else
                return get_pll_pfd(SYS_PLL, PFD2);
        } else
            return CKIH;          /* OSC */
    } else
        return get_pll(ARM_PLL);
}

```

```

/*****
*      get_pll3_sw_clk
*      PLL3_SW_CLK
*
*
*      PLL3_SW_CLK
*
* -----
* 2020/03/08      V1.0
*****/

```

```

u32 get_pll3_sw_clk(void)
{
    u32 reg = read32(&ccm->ccsr);
    if (reg & 1)    /* PLL3_SW_CLK_SEL: 1 pll3 0 pll3_bypass OSC */
        return get_pll(USB1_PLL);
    else
        return CKIH;          /* OSC */
}

```

```

/*****
*      get_pll4_main_clk
*      PLL4_MAIN_CLK
*
*
*      PLL4_MAIN_CLK

```

```

*
* -----
* 2020/03/08      V1.0
*****/
extern struct anadig_regs *anadig;

u32 get_pll4_main_clk( void)
{
    u32 reg, audio_div;

    reg = read32( &anadig->pmu_misc2);
    /* AUDIO_DIV_MSB( 23): AUDIO_DIV_LSB( 15)
        * 00    1
        * 01    2
        * 10    1
        * 11    4
        */
    audio_div = reg & (1u<<15) ? (reg & (1u<<23) ? 4 : 2) : 1;

    return get_pll( AUDIO_PLL) / audio_div;
}

/*****
*      get_pll5_main_clk
*      PLL5_MAIN_CLK
*
*
*      PLL5_MAIN_CLK
*
* -----
* 2020/03/08      V1.0
*****/
u32 get_pll5_main_clk( void)
{
    u32 reg, video_div;

    reg = read32( &anadig->pmu_misc2);
    /* AUDIO_DIV_MSB( 31): AUDIO_DIV_LSB( 30)
        * 00    1
        * 01    2

```

```

    * 10  1
    * 11  4
    */
    video_div = reg & (1u<<30) ? (reg & (1u<<31) ? 4 : 2) : 1;

    return get_pll(VIDEO_PLL) / video_div;
}

```

6.3.2.3 PLL1_SW_CLK

root generator

clkroot.c

```

/*****

*      get_arm_clk_root
*      ARM_CLK_ROOT
*
*
*      ARM_CLK_ROOT
*
* -----
* 2020/03/08      V1.0
*****/

u32 get_arm_clk_root(void)
{
    u32 reg, freq;

    reg = read32(&ccm->cacrr);
    reg = (reg & 0x7) + 1;          /* ARM_PODF      */
    freq = get_pll(ARM_PLL);

    return freq / reg;
}

/*****

*      get_periph_clk
*      PERIPH_CLK
*
*
*      PERIPH_CLK
*
*****/

```

```

* -----
* 2020/03/08      V1.0
*****/

static u32 get_periph_clk(void)
{
    u32 reg, per_clk2_podf = 0, freq = 0;

    reg = read32(&ccm->cbcdr);

    /* PERIPH_CLK_SEL periph_clk 1 periph_clk2, 0 pre_periph_clk */
    if (reg & (1u << 25)) { /* periph_clk2 */
        per_clk2_podf = (reg >> 27) & 0x7; /* PERIPH_CLK2_PODF */
        reg = read32(&ccm->cbcmr);
        reg = (reg >> 12) & 0x3; /* PERIPH_CLK2_SEL */

        /* PERIPH_CLK2_SEL: 0 pll3_sw_clk 1 osc_clk 2 pll2_bypass_clk osc_clk */
        switch (reg) {
            case 0:
                freq = get_pll(USB1_PLL);
                break;
            case 1:
            case 2:
                freq = CKIH;
                break;
            default:
                break;
        }

        freq /= (per_clk2_podf + 1);
    } else { /* pre_periph_clk */
        reg = read32(&ccm->cbcmr);
        reg = (reg >> 18) & 0x3; /* PRE_PERIPH_CLK_SEL */

        /* PRE_PERIPH_CLK_SEL 0 PLL2 1 PLL2 PFD2 2 PLL2 PFD0 3 PLL2 PFD2 */
        switch (reg) {
            case 0:
                freq = get_pll(SYS_PLL);
                break;
            case 1:
                freq = get_pll_pfd(SYS_PLL, PFD2);

```

```

        break;
    case 2:
        freq = get_pll_pfd(SYS_PLL, PFD0);
        break;
    case 3:    /* static / 2 divider */
        freq = get_pll_pfd(SYS_PLL, PFD2) / 2;
        break;
    default:
        break;
    }
}

return freq;
}

/*****
 *      get_ahb_clk_root
 *      AHB_CLK_ROOT
 *
 *
 *      AHB_CLK_ROOT
 *
 * -----
 * 2020/03/08      V1.0
 *****/
u32 get_ahb_clk_root(void)
{
    u32 reg, ahb_podf;

    reg = read32(&ccm->cbcdr);
    ahb_podf = (reg >> 10) & 0x7;    /*  AHB_PODF      */

    return get_periph_clk() / (ahb_podf + 1);
}

/*****
 *      get_ipg_clk_root
 *      IPG_CLK_ROOT
 *
 *
 *****/

```

```

*          IPG_CLK_ROOT
*
* -----
* 2020/03/08      V1.0
*****/
u32 get_ipg_clk_root(void)
{
    u32 reg, ipg_podf;

    reg = read32(&ccm->cbcdr);
    ipg_podf = (reg >> 8) & 0x3;    /* IPG_PODF */

    return get_ahb_clk_root() / (ipg_podf + 1);
}

/*****
*          get_axi_clk_root
*          AXI_CLK_ROOT
*
*
*          AXI_CLK_ROOT
*
* -----
* 2020/03/08      V1.0
*****/
u32 get_axi_clk_root(void)
{
    u32 root_freq, axi_podf;
    u32 reg = read32(&ccm->cbcdr);

    axi_podf = (reg >> 16) & 0x7;    /* AXI_PODF */

    if (reg & (1u << 6)) {          /* AXI_SEL: 1 axi_alt_clk 0 periph_clk */
        if (reg & (1u << 7))        /* AXI_ALT_SEL 1 PLL3 PFD1 0 PLL2 PFD2 */
            root_freq = get_pll_pfd(USB1_PLL, PFD1);
        else
            root_freq = get_pll_pfd(SYS_PLL, PFD2);
    } else
        root_freq = get_periph_clk();    /* periph_clk */
}

```

```

    return root_freq / (axi_podf + 1);
}

/*****
*      get_fabric_mmdc_clk_root
*      FABRIC_MMDC_CLK_ROOT
*
*
*      FABRIC_MMDC_CLK_ROOT
*
* -----
* 2020/03/08      V1.0
*****/
u32 get_fabric_mmdc_clk_root(void)
{
    u32 cbcmr = read32(&ccm->cbcmr);
    u32 cbcdr = read32(&ccm->cbcdr);

    u32 freq, podf, per2_clk2_podf;

    podf = (cbcdr >> 3) & 0x7; /* FABRIC_MMDC_PODF */

    if (cbcdr & (1u << 26)) { /* PERIPH2_CLK_SEL: 1 periph2_clk2 0 pre_periph2_clk */
        per2_clk2_podf = cbcdr & 0x7; /* PERIPH2_CLK2_PODF */
        if (cbcmr & (1u << 20)) /* PERIPH2_CLK2_SEL 1 osc_clk 0 pll3_sw_clk */
            freq = CKIH;
        else
            freq = get_pll(USB1_PLL);

        freq /= (per2_clk2_podf + 1);
    } else { /* pre_periph2_clk */
        extern u32 get_pll4_main_clk(void);
        /* PRE_PERIPH2_CLK_SEL 0 PLL2 1 PLL2 PFD2 2 PLL2 PFD0 3 _main_clk */
        switch ((cbcmr >> 21) & 0x3) {
            case 0:
                freq = get_pll(SYS_PLL);
                break;
            case 1:
                freq = get_pll_pfd(SYS_PLL, PFD2);
                break;

```

```

        case 2:
            freq = get_pll_pfd(SYS_PLL, PFD0);
            break;
        case 3:
            freq = get_pll4_main_clk();
            break;
    }
}

return freq / (podf + 1);
}

```

6.3.2.4

show_clocks

main.c

```

27 /*****
28 *      show_clocks
29 *      PLL
30 *
31 *
32 *
33 *
34 * -----
35 * 2020/03/08      V1.0
36 *****/
37 void show_clocks(void)
38 {
39     u32 freq;
40
41     printf("Show IMX6ULL Clocks: \r\n");
42     freq = get_pll(ARM_PLL);
43     printf("ARM_PLL      %8d MHz\r\n", freq / 1000000);
44
45     freq = get_pll(SYS_PLL);
46     printf("SYS_PLL      %8d MHz\r\n", freq / 1000000);
47     freq = get_pll_pfd(SYS_PLL, PFD0);
48     printf("| -SYS_PLL_PFD0  %8d MHz\r\n", freq / 1000000);
49     freq = get_pll_pfd(SYS_PLL, PFD1);
50     printf("| -SYS_PLL_PFD1  %8d MHz\r\n", freq / 1000000);

```



```

51     freq = get_pll_pfd(SYS_PLL, PFD2);
52     printf("| -SYS_PLL_PFD2   %8d MHz\r\n", freq / 1000000);
53     freq = get_pll_pfd(SYS_PLL, PFD3);
54     printf("| -SYS_PLL_PFD3   %8d MHz\r\n", freq / 1000000);
55
56     freq = get_pll(USB1_PLL);
57     printf("USB1_PLL       %8d MHz\r\n", freq / 1000000);
58     freq = get_pll_pfd(USB1_PLL, PFD0);
59     printf("| -USB1_PLL_PFD0 %8d MHz\r\n", freq / 1000000);
60     freq = get_pll_pfd(USB1_PLL, PFD1);
61     printf("| -USB1_PLL_PFD1 %8d MHz\r\n", freq / 1000000);
62     freq = get_pll_pfd(USB1_PLL, PFD2);
63     printf("| -USB1_PLL_PFD2 %8d MHz\r\n", freq / 1000000);
64     freq = get_pll_pfd(USB1_PLL, PFD3);
65     printf("| -USB1_PLL_PFD3 %8d MHz\r\n", freq / 1000000);
66
67     freq = get_pll(USB2_PLL);
68     printf("USB2_PLL       %8d MHz\r\n", freq / 1000000);
69     freq = get_pll(AUDIO_PLL);
70     printf("AUDIO_PLL      %8d MHz\r\n", freq / 1000000);
71     freq = get_pll(VIDEO_PLL);
72     printf("VIDEO_PLL     %8d MHz\r\n", freq / 1000000);
73
74     printf("\r\n");
75     freq = get_arm_clk_root();
76     printf("ARM_CLK_ROOT    %8d KHZ\r\n", freq / 1000);
77     freq = get_ahb_clk_root();
78     printf("AHB_CLK_ROOT    %8d KHZ\r\n", freq / 1000);
79     freq = get_ipg_clk_root();
80     printf("IPG_CLK_ROOT    %8d KHZ\r\n", freq / 1000);
81     freq = get_axi_clk_root();
82     printf("AXI_CLK_ROOT    %8d KHZ\r\n", freq / 1000);
83     freq = get_fabric_mmdc_clk_root();
84     printf("FABRIC_MMDC_CLK_ROOT    %8d KHZ\r\n", freq / 1000);
85     printf("\r\n");
86 }

```

show_clocks

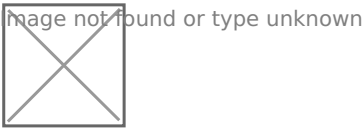
CPU

648000KHZ 648MHZ

: Git NoosProgramProject/(7_ /showclocks)

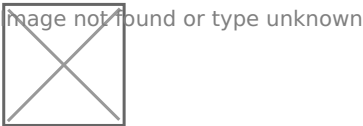
6.3.2.5 4-1.4

6.3.2.6 4-1.4



6.3.3

CPU 8 led 8 busy_wait showclock.elf



busy_wait

```
static void busy_wait(void)
{
    __asm__ __volatile__ (
        "ldr r0, =30000000\n"
        "1: \n"
        "subs r0, r0, #1\n"
        "bne 1b\n"
        ::: "r0");
}
```

CPU 648MHZ led D-Cache CCM_CLKO1 CCM_CLKO2 clkout.c