Set sampling frequency

Secciones

1. Habilita el canal mediante

Registro : 0x0020

2. Set sample Rate

Registro : 0x0087,0x0088,0x0089,0x008B

Se puede observar que las funciones son equivalentes

set_data_clock → RISC driver

LMS_SetSampleRate→ host

Host side
Enable channel

Enable Channel

```
//Enable TX channel,Channels are numbered starting at 0
if (LMS_EnableChannel(device, LMS_CH_TX, 0, true)!=0)
    error();

//Set sample rate
if (LMS_SetSampleRate(device, sample_rate, 0)!=0)
    error();
```

```
int LMS7002M::EnableChannel(const bool isTx, const bool enable)
{
    Channel ch = this->GetActiveChannel();

    //--- LML ---
    if (ch == ChA)
    {
        if (isTx) this->Modify_SPI_Reg_bits(LMS7param(TXEN_A), enable?1:0);
        else      this->Modify_SPI_Reg_bits(LMS7param(RXEN_A), enable?1:0);
    }
```

```
ter LMS7_TXEN_B = { 0x0020, 3, 3, 1, "TXEN_B", "Pow
ter LMS7_TXEN_A = { 0x0020, 2, 2, 1, "TXEN_A", "Pow
ter LMS7_MAC = { 0x0020, 1, 0, 3, "MAC", "Selects M
```

Uso de registro 0x0020 para habiltar el canal A

RISCV
Enable Channel

Enable Channel

El canal se configura dentro de set_data_clock

Registro 0x0020

```
ret = LMS7002M_set_data_clock(lms, REF_FREQ, 61.44e6, &actualRate);
```

```
int LMS7002M_set_data_clock(LMS7002M_t *
{
    LMS7_logf(LMS7_INFO, "CGEN tune %f M

    //always use the channel A shadow, C
    LMS7002M_set_mac_ch(self, LMS_CHA);
```

```
void LMS7002M_set_mac_ch(LMS7002M_t *sel
{
    //pick the register map and setting
    int newValue = 0;
    LMS7002M_regs_t *regs = NULL;
    switch (channel)
    {
    case LMS_CHA:
        newValue = REG_0X0020_MAC_CHA;
        regs = &self->_regs[0];
```

```
self->regs = self->_regs;
if (self->regs->reg_0x0020_mac != newValue)
{
    self->regs->reg_0x0020_mac = newValue;
    LMS7002M_regs_spi_write(self, 0x0020);
}
```

Host side
Sampling Rate

Uso de CGEN

```
//Set sample rate
if (LMS_SetSampleRate(device, sample_rate, 0)!=0)
    error();
```

```
int LMS7_Device::SetRate(double f_Hz, int oversample)
{
if ((lms->SetFrequencyCGEN(f_Hz*4*oversample) != 0)
```

```
Modify_SPI_Reg_bits(0x0087, 15, 0, gFRAC&0xFFFF); //INT_
Modify_SPI_Reg_bits(0x0088, 3, 0, gFRAC>>16); //INT_SDM_
Modify_SPI_Reg_bits(LMS7param(DIV_OUTCH_CGEN), iHdiv); /
```

```
ameter LMS7_CLKH_OV_CLKL_CGEN = { 0x0089, 12, 11, 0, "CLKH_OV_CLK
ameter LMS7_DIV_OUTCH_CGEN = { 0x0089, 10, 3, 4, "DIV_OUTCH_CGEN"
ignasi, 5 years ago | 1 author (ignasi)
```

```
if(TuneVCO(VCO_CGEN) != 0)
{
    if (output)
    {
        output->success = false;
        output->csw = Get_SPI_Reg_bits(LMS7param(CSW_VCO_CGEN));
    }
    return ReportError("SetFrequencyCGEN(%g MHz) failed", freq_Hz
}
if (output)
    output->csw = Get_SPI_Reg_bits(LMS7param(CSW_VCO_CGEN));
return 0;
```

```
LMS7_CSW_VCO_CGEN = { 0x008B, 8, 1, 128, "CSW_VCO_CGEN", "
LMS7_COARSE_START_CGEN = { 0x008B, 0, 0, 0, "COARSE_START
```

RISCV
Set sample Rate

```
int LMS7002M_set_data_clock(LMS7002M_t *self, const double fref, const double fout, double *factual)
{
```

```
 //try the next even divider
 fdiv -= 2;

 Ndiv = fout*fdiv/fref;
 fvco = fout*fdiv;
```

```
//program the N divider
const int Nint = (int)Ndiv;
const int Nfrac = (int)((Ndiv-Nint)*(1 << 20));
self->regs->reg_0x0087_frac_sdm_cgen = (Nfrac) & 0xff
self->regs->reg_0x0088_frac_sdm_cgen = (Nfrac) >> 16;
self->regs->reg_0x0088_int_sdm_cgen = Nint-1;
LMS7_logf(LMS7_DEBUG, "fdiv = %d, Ndiv = %f, Nint = %
LMS7002M_regs_spi_write(self, 0x0087);
LMS7002M_regs_spi_write(self, 0x0088);

//program the feedback divider
self->regs->reg_0x0089_sel_sdmclk_cgen = REG_0X0089_S
self->regs->reg_0x0089_div_outch_cgen = (fdiv/2)-1;
LMS7002M_regs_spi_write(self, 0x0089);
```

Ya se hace el calculo a cuenta

Fracionaria y entera

```
//select the correct CSW for this VCO frequency
if (LMS7002M_tune_vco(self,
    &self->regs->reg_0x008b_csw_vco_cgen, 0x008B,
    &self->regs->reg_0x008c_vco_cmpho_cgen,
    &self->regs->reg_0x008c_vco_cmplo_cgen, 0x008C) != 0)
{
    LMS7_log(LMS7_ERROR, "VCO select FAIL");
    return -3;
}
```