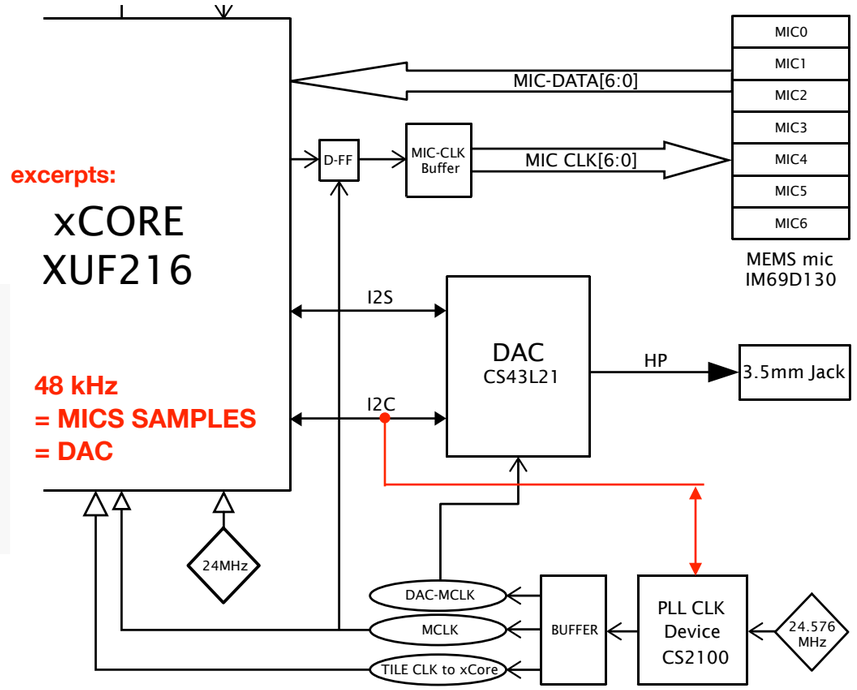
Figure 3: xCORE Microphone Array block diagram



From:
xCORE Microphone Array Hardware Manual
Publication Date: 2018/6/14 Document Number: XM009730A

Code based on:
Application Note: AN00219: Low Resolution Delay and Sum
XMOS 2017 (XM010100)

3 Clock sources and distribution

The board includes three clock sources:

- xCORE-200 reference clock - 24MHz oscillator (Y1)
- Ethernet PHY reference clock - 25MHz crystal (X1)
- Low jitter clock source - 24.576MHz oscillator, used as reference clock to the CS2100-CP (CirrusLogic) Fractional-N PLL (U22).
  The CS2100 generates a low-jitter output signal that is distributed to the xCORE- 200 device (Tile1 & MCLK) and DAC (MIC-CLK). The CS2100 device is configured using the I2C interface.

8 Expansion Header

The board has an expansion header containing 7 general purpose IOs, controlled by the xCORE-200, and an audio MCLK.

By removing R67 and inserting a 0R link into R17, the expansion header audio MCLK can be used as an alternative to the CS2100-CP (CirrusLogic) Fractional-N PLL (U22) output.

Figure 18: xCORE Microphone Array Portmap: Tile 0

```
X0D13 1F0            MCLK_XCORE
```

Figure 19: xCORE Microphone Array Portmap: Tile 1

```
X1D38 10⁰ 8D² 16B¹⁰ MCLK_TILE1
```

```
// ==== MICS IN ====
#define MIC_ARRAY_MAX_FRAME_SIZE_LOG2  0
#define MIC_ARRAY_NUM_MICS_LOG2        3 // 2exp3 = 8 = MIC_ARRAY_NUM_MICS
#define MIC_ARRAY_NUM_MICS             8 // Must have "8 mics" (=7) since directive
#define DECIMATION_FACTOR              2 // 48 kHz
#define MIC_ARRAY_NUM_SAMPLES_PER_MIC (1 << MIC_ARRAY_MAX_FRAME_SIZE_LOG2) // 1
#define SAMPLING_FREQUENCY_HZ          48000 // MIC for the values above (see below)
#define USE_EVERY_N_SAMPLE             6     // I only need 8 kHz FOR DSP handling
// ==== ONLY FOR THE DAC OUT ====
#define MASTER_PLL_24576KHZ_TO_PDM_CLOCK_DIVIDER (4)
#define MASTER_CLOCK_FREQUENCY_HZ                24576000
#define PDM_CLOCK_FREQUENCY_HZ                   (MASTER_CLOCK_FREQUENCY_HZ / (2 * MASTER_PLL_24576KHZ_TO_PDM_CLOCK_DIVIDER))
#define DAC_OUTPUT_FREQUENCY_HZ                  (PDM_CLOCK_FREQUENCY_HZ / (32 * DECIMATION_FACTOR))
#define MCLK_BCLK_RATIO                          (DECIMATION_FACTOR * MASTER_PLL_24576KHZ_TO_PDM_CLOCK_DIVIDER)
```

Excerpts from xC code in main.xc:

```
on tile[0]: in port              p_mclk      = XS1_PORT_1F;
on tile[0]: clock                pdmclk      = XS1_CLKBLK_1;
on tile[1]: in port              p_mclk_in1  = XS1_PORT_1O;
on tile[1]: out buffered port:32 p_i2s_bclk  = XS1_PORT_1M;
on tile[1]: out buffered port:32 p_i2s_lrclk = XS1_PORT_1N;
on tile[1]: clock                mclk        = XS1_CLKBLK_3;
on tile[1]: clock                bclk        = XS1_CLKBLK_4;
par {
    // Everything on TILE[1] HAS TO DO WITH HEADSET OUT
    on tile[1]: {
        configure_clock_src (mclk, p_mclk_in1);
        start_clock (mclk);
        i2s_master (if_i2s_board, p_i2s_dac_data, I2S_BOARD_NUM_CS43L21,
                null, 0, p_i2s_bclk, p_i2s_lrclk,
                bclk, mclk);
    }
    on tile[0]:{
        // Begin a 3.072MHz PDM clock used for clocking the microphone data into the xCORE:
        //
        configure_clock_src_divide  (pdmclk, p_mclk, MASTER_PLL_24576KHZ_TO_PDM_CLOCK_DIVIDER);
        configure_port_clock_output (p_pdm_clk, pdmclk);
        configure_in_port           (p_pdm_mics, pdmclk);
        start_clock                 (pdmclk);
    }
}
```

Excerpts from xC code in mics_in_headset_out.xc:

```
[[distributable]]
void i2s_task ( // (i2s_handler)
        server  i2s_callback_if i2s, // From i2s_master0
        client  i2c_master_if   i2c,
        port    p_rst_shared,
        chanend ch_headset_out) // Coming from "mics_in_headset_out_task_a" having "out" semantics
{
    mabs_init_pll (i2c, ETH_MIC_ARRAY);
    p_rst_shared <: CS41L21_DAC_ETH_RESET;

    i2c_regop_res_t res;
    uint8_t i2c_data;

    // CS41L21_DAC_REG_01_CHIP_ID_AND_REVISION
    // Bit7     6        5        4        3        2        1        0
    // Chip_ID4 Chip_ID3 Chip_ID2 Chip_ID1 Chip_ID0 Rev_ID2 Rev_ID1 Rev_ID0
    i2c_data = i2c.read_reg (I2C_ADDRESS_OF_DAC, CS41L21_DAC_REG_01_CHIP_ID_AND_REVISION, res);

    // CS41L21_DAC_REG_02_POWER_CONTROL_1 = POWER DOWN
    // Bit7     6        5        4        3        2        1        0
    // Reserved PDN_DACB PDN_DACA Reserved Reserved Reserved Reserved PDN
    i2c_data = i2c.read_reg (I2C_ADDRESS_OF_DAC, CS41L21_DAC_REG_02_POWER_CONTROL_1, res);
    i2c_data |= 1; // Set Power Down (PDN) bit0 to 1=Enable
    res = i2c.write_reg (I2C_ADDRESS_OF_DAC, CS41L21_DAC_REG_02_POWER_CONTROL_1, i2c_data);

    // CS41L21_DAC_REG_03_SPEED_CONTROL: Setting MCLKDIV2 high if using 24.576MHz.
    // Bit7 6      5      4      3        2        1        0
    // AUTO SPEED1 SPEED0 3-ST_SP Reserved Reserved Reserved MCLKDIV2
    i2c_data = i2c.read_reg (I2C_ADDRESS_OF_DAC, CS41L21_DAC_REG_03_SPEED_CONTROL, res);
    i2c_data |= 0x01; // 0x01 Set MCLK Divide By 2 (MCLKDIV2) bit0 to 1=Divide by 2.
                      // With AUTO=0 this bit is not required to be high for 8 kHz (page 59)

    res = i2c.write_reg (I2C_ADDRESS_OF_DAC, CS41L21_DAC_REG_03_SPEED_CONTROL, i2c_data);
                                                                  // 3072000
                                                                  // 48000 = SAMPLING_FREQUENCY_HZ
                                                                  // 8 = FOR UNDERSTANDING ONLY
    // CS41L21_DAC_REG_10_MIXER_VOLUME_CONTROL_PCMA
    // Bit7     6        5        4        3        2        1        0
    // MUTE_    PCMMIXx PCMMIXx_ PCMMIXx_ PCMMIXx_ PCMMIXx_ PCMMIXx_ PCMMIXx_
    // PCMMIXx_ VOL6    VOL5     VOL4     VOL3     VOL2     VOL1     VOL0
    // 0        1       1        1        0        0        0        0
    i2c_data = 0b01110000; // 0x70 = 112. * 0.5 dB = 56 dB gain
    res = i2c.write_reg (I2C_ADDRESS_OF_DAC, CS41L21_DAC_REG_10_MIXER_VOLUME_CONTROL_PCMA, i2c_data);

    // CS41L21_DAC_REG_02_POWER_CONTROL_1 POWER UP
    // Bit7     6        5        4        3        2        1        0
    // Reserved PDN_DACB PDN_DACA Reserved Reserved Reserved Reserved PDN
    i2c_data = i2c.read_reg (I2C_ADDRESS_OF_DAC, CS41L21_DAC_REG_02_POWER_CONTROL_1, res);
    i2c_data &= ~1; // and_eq 0xFE is clear Power Down (PDN) bit0 0=Disable
    res = i2c.write_reg (I2C_ADDRESS_OF_DAC, CS41L21_DAC_REG_02_POWER_CONTROL_1, i2c_data);

    // IF FOR NOTHING ELSE; GOOD FOR ENSURING WORKING CHIP!
    // Bit7     6        5        4        3        2          1          0
    // RModSel2 RModSel1 RModSel0 Reserved Reserved AuxOutSrc1 AuxOutSrc0 EnDevCfg1
    // 0        0        0        0        0        0          0          0 is init value anyhow!
    //                                               0          0            RefClk
    res = i2c.write_reg (I2C_ADDRESS_OF_PLL, CS2100CP_PLL_REG_03_DEVICE_CONFIGURATION_1, 0);

    while (1) {
        // select: cases handle callbacks from i2s driver
    }
```

My Beep-BRRR notes
https://www.teigfam.net/oyvind/home/technology/219-my-beep-brrr-notes/#download_code
Øyvind Teig (5Jan2022)