Can gears that engage only so good make the machine run?

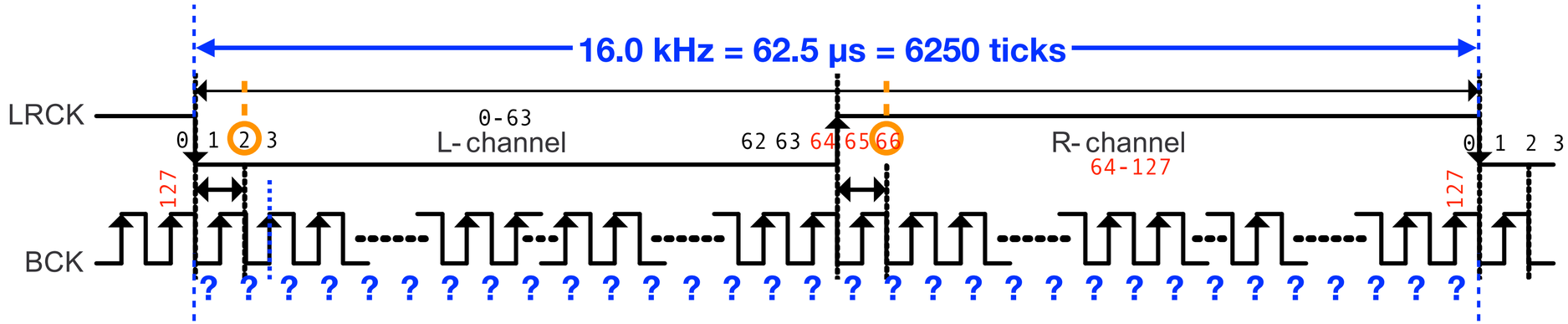
Øyvind Teig, Trondheim, Norway

Fringe presentation at COPA 2025 April 21-24 (https://ieee-copa.org/)

Abstract

- This conference «fringe» presentation explains an implementation which reads from a MEMS microphone and writes to a headset DAC
- To avoid using an external PLL chip, clock output pulses rather unusually vary some in time
- The timing of these «staccato» pulses is taken from a software table with 10 ns tick resolution (100 MHz)
- The pattern thus generated repeats exactly every 62.5 µs, giving a stereo frame rate of 16,000 Hz
- The table contains 128 elements;
  two edges per pulse * 32 bits * two channels,
  each representing the number of ticks to the next clock signal edge output
- The presentation shows that this controlled unevenness correctly picks up the data bits from the microphone chip or delivers the data bits to the headset chip
- A processor from XMOS (UK) is used for this

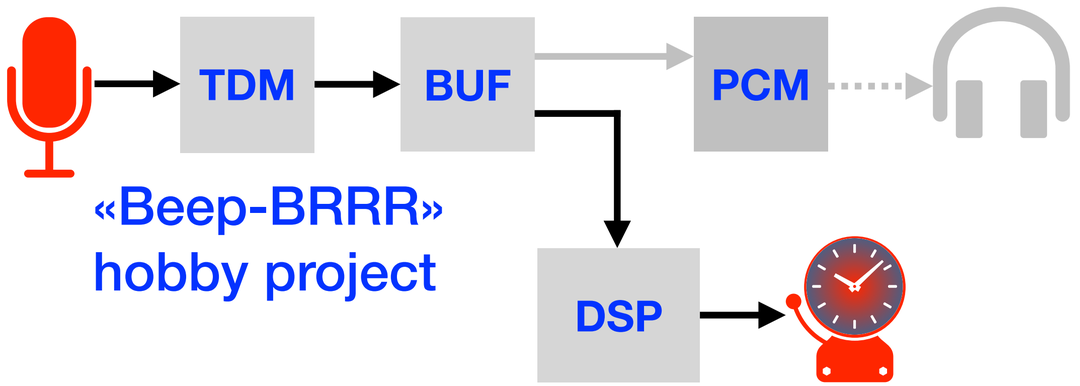
Are the ? ? ? ok?



- Mic-in & headset-out (monitor) clock edges @2.048 MHz max
- Precise average per 16 kHz frame
- But table driven skewed timing of «staccato» clock pulses ? ? ?
- Coded in (somewhat) obsoleted XC
- New paradigm from XMOS is C plus `lib_xcore`

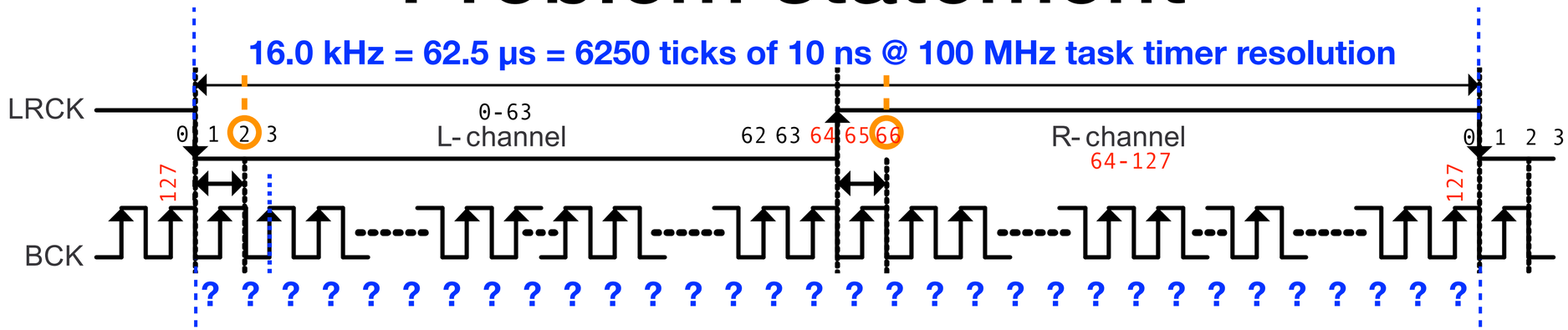
Application

- Application listens for recorded «alarm» sounds
- Alarms hearing deficient people
- Fully implemented since I retired
- Industrialisation of the product not yet conceived
- Could Apple iOS AI-based «Sound recognition» possibly do the same?



https://www.teigfam.net/oyvind/home/my-beep-brrr-pages/

Problem statement



16 kHz stereo @ 32 bits each = 64 bits = 128 edges

128 edges * 16000 = 2.048 MHz

1 / 16000 Hz = 0.000062500 s = 62.5 µs = 6250 ticks of 10 ns

Processor task timer @100 MHz

Integer ticks! GREAT!

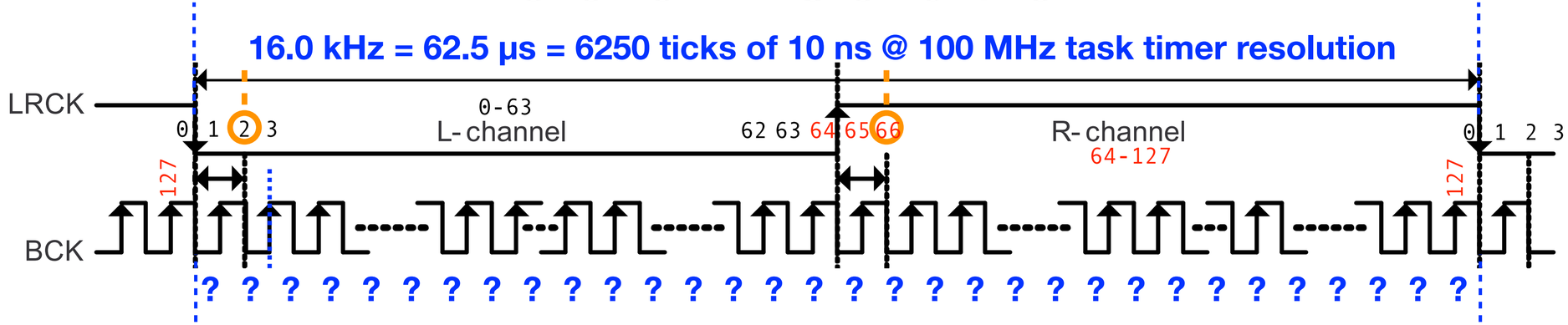
Also task timer resolution

6250 ticks to deliver 128 edges =

6250 / 128 = 48.828125 ticks between each edge

Decimal ticks! Needing extern PLL?

Problem solution?



6250 ticks to deliver 128 edges =

6250 / 128 = 48.828125 ticks between each edge

Decimal ticks! Needing extern PLL?

Not if BCK duty cycle requirement is «fair» so that we can tabulate uneven timing, and sum up 128 uneven edges to exactly 6250 ticks

Mark 1: using audio-ready board

XMOS xCore Microphone array evaluation board

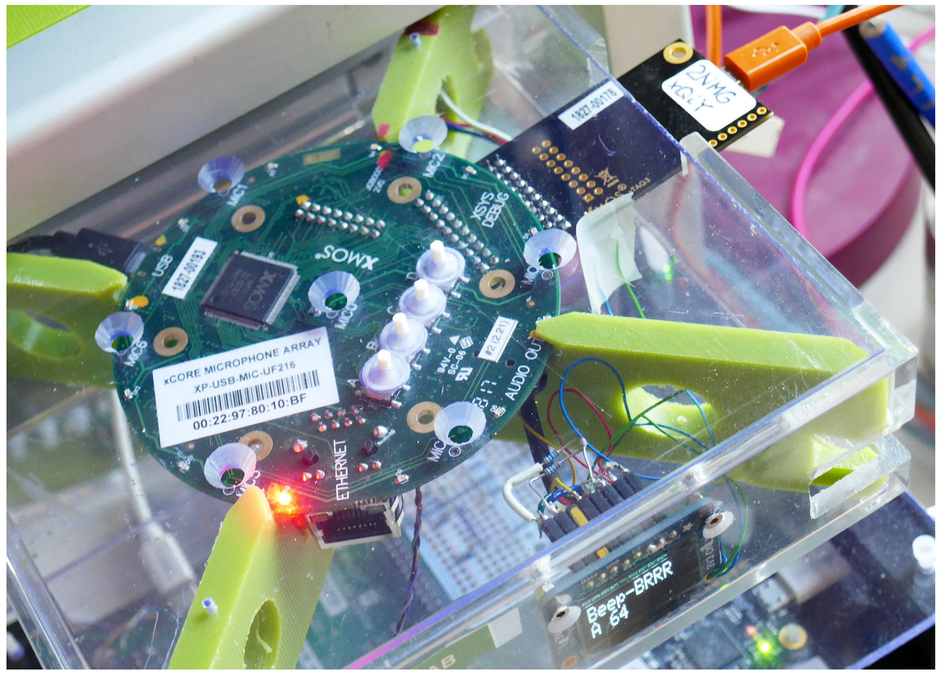
On-board microphones

On-board headset amplifier

On-board PLL clock

XMOS libraries

Obsoleted, but used in my BEEP-BRRR mark 1 (one built)



Mark 1: using audio-ready board

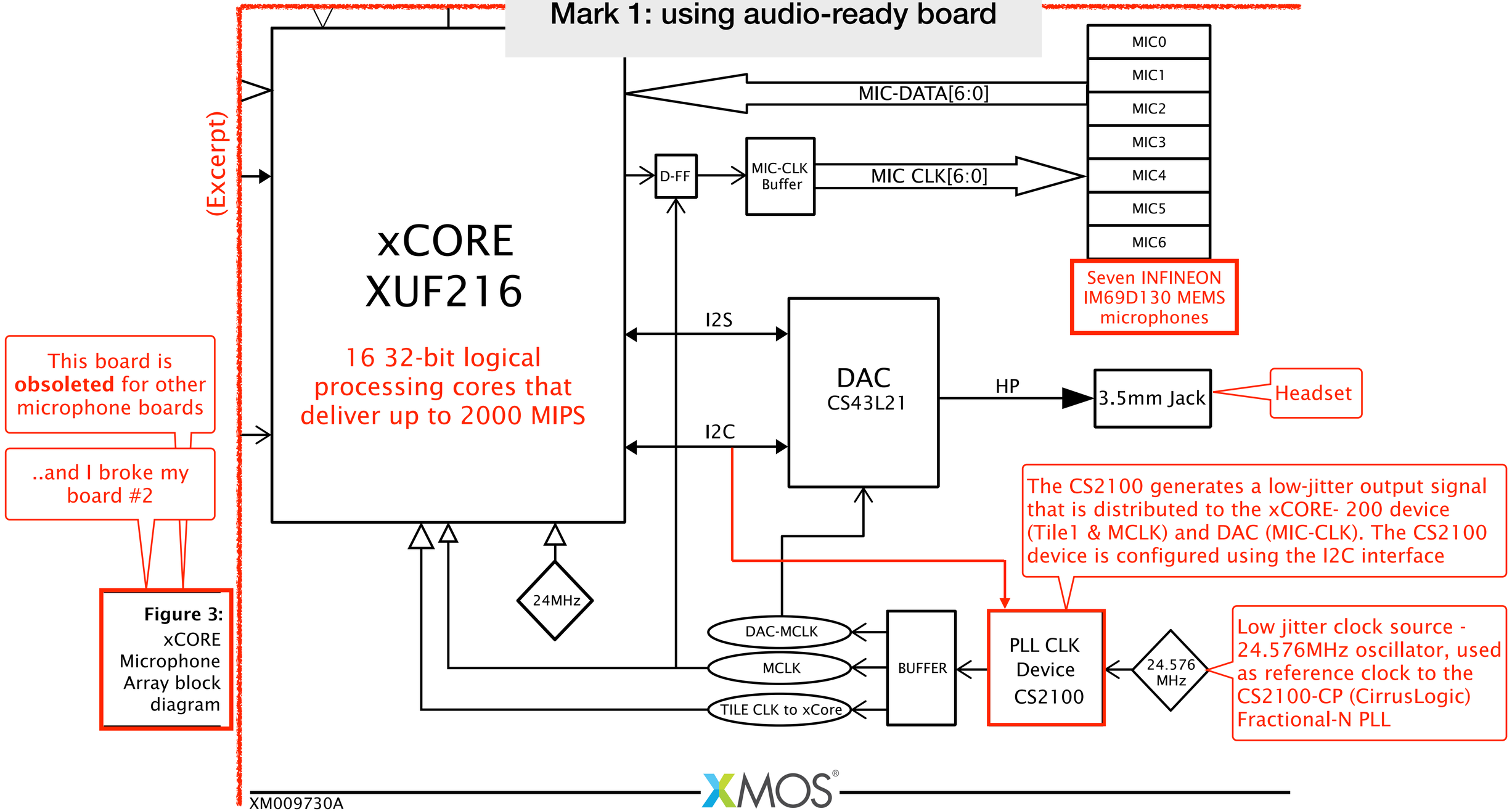


Figure 3: xCORE Microphone Array block diagram

Mark 1: using audio-ready board

Mark 2 (next page)

Find a more general board and not need to use these

The xCORE-200 explorer board

Updated 4.24

XMOS libraries

```
USED_MODULES += lib_mic_array
```

Configure the PLL for the specified board

Try no external PLL

The i2c bus used to configure the audio codec

```
USED_MODULES += lib_mic_array_board_support
```

PDM interface

Try to «bit bang» this myself..

Four channel decimators

Try to find a mic that contains decimation SW

High resolution delay block (since 7 mics)

..from basically 100 MHz XC task timer tick resolution (10 ns)

I only need one mic

Mark 2: using a general board

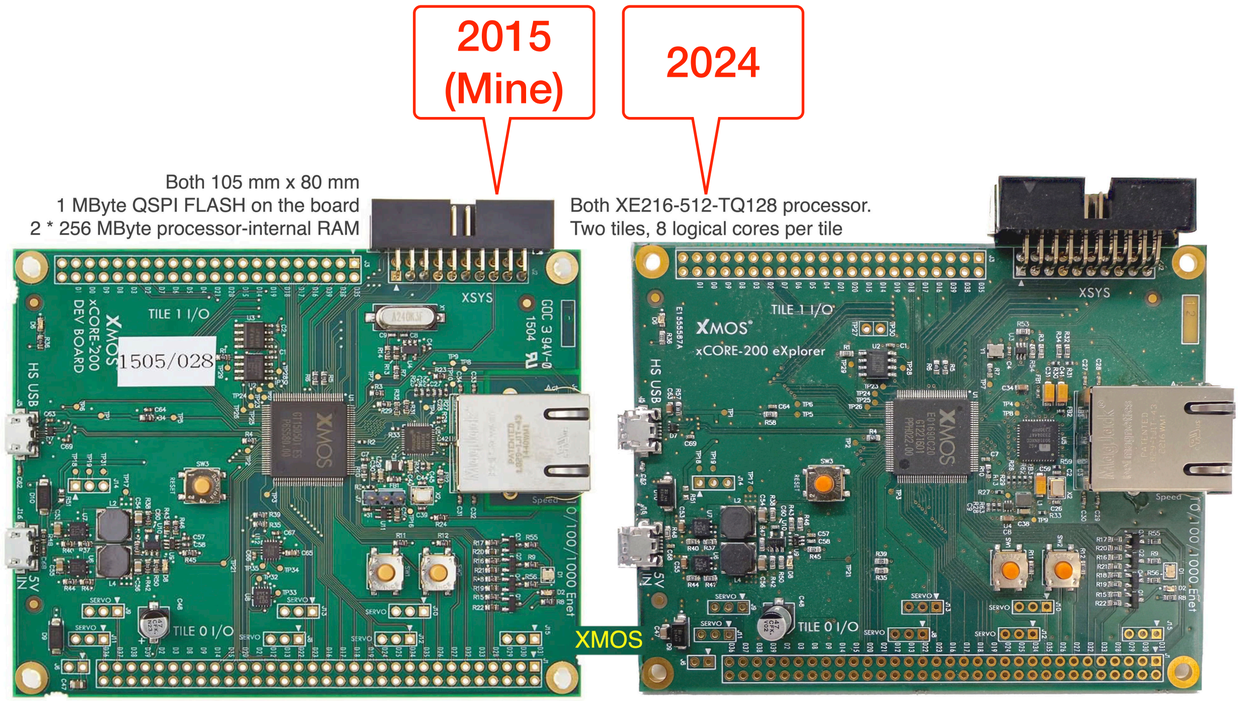
XMOS xCore-200 explorer evaluation board

External microphone

External headset amplifier

Try NOT to use any external PLL (there is no on-board PLL)

Using this in my BEEP-BRRR mark 2 (two built)

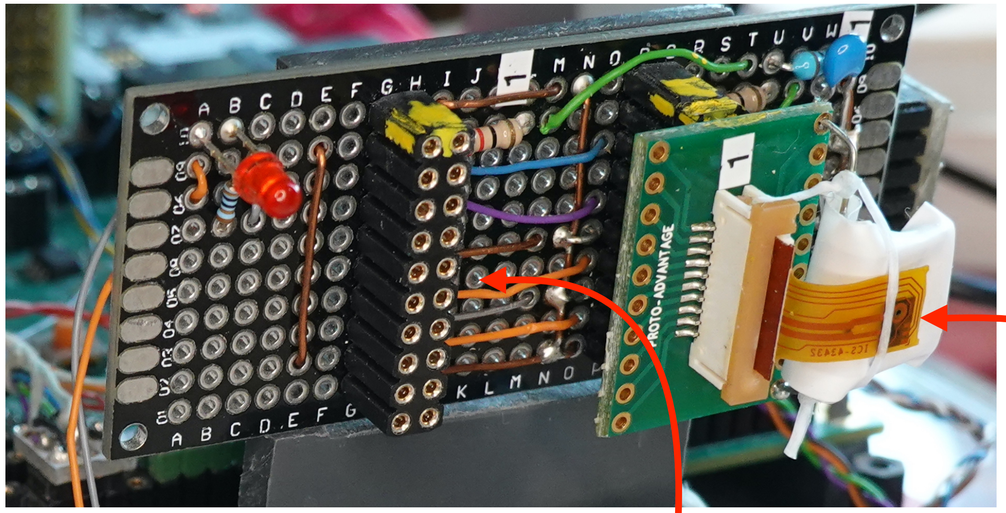


xCORE-200 eXplorerKIT XK-EVK-XE216 v1.2 (2015) with gyroscope and accelerometer

xcore-200 Evaluation Kit XK-EVK-XE216 v1.3 (2024)

Mark 2: using a general board

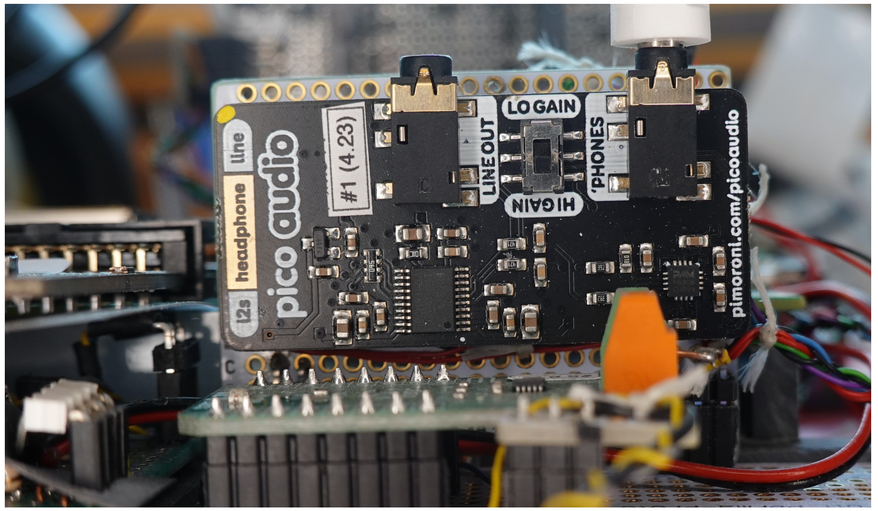
External microphone



- EV_ICS-52000-FX AN-000001 single MEMS microphone from TDK InvenSense (evaluation board)
- Clocked as TDM by 1024 kHz pulses it delivers 16 kHz mono (+ ghost channel)
- Fully processed as decimated signed 2’s complement values
- Obsoleted 🧐

Mark 2: using a general board

External headset amp (monitor)



- Pimoroni Pico Audio Pack (Line-Out and Headphone Amp) PIM544
- Contains a Texas Instruments PCM5100 DAC ← signed 32 bits
- 2’s complement @1024 kHz for 16 kHz stereo into the headset
- I2S and left-justified input data formats
- Integrated PLL for no external system master clock input
- Internal filters

EVALUATION BOARD PHOTOGRAPHS

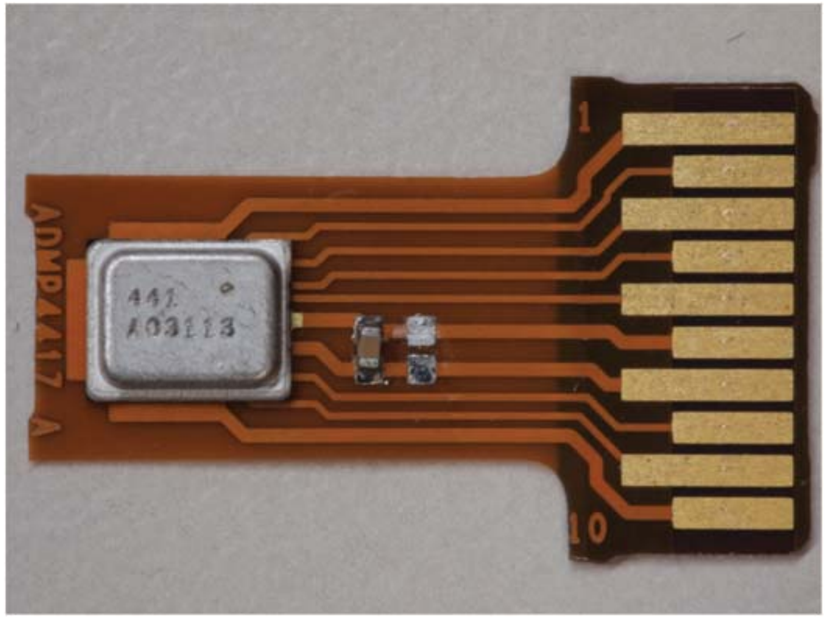


Figure 7. EV_INMP441-FX Top View

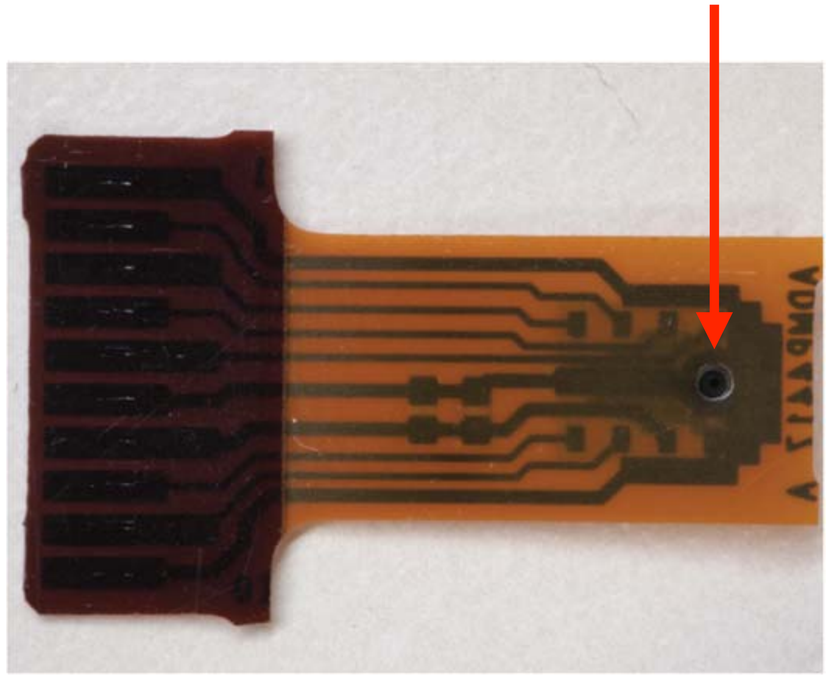
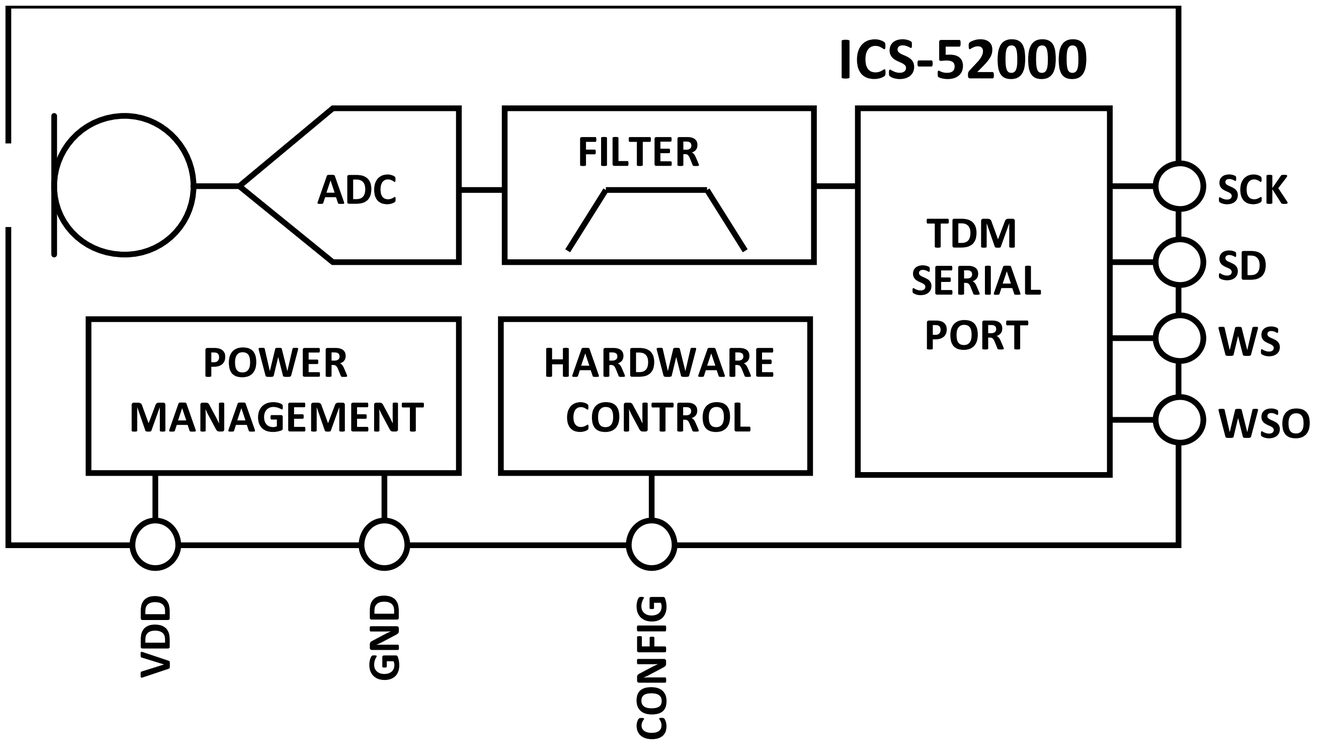


Figure 9. EV_INMP441-FX Bottom View

InvenSense
Sensing Everything

Low-Noise Microphone with TDM Digital Output

FUNCTIONAL BLOCK DIAGRAM



- SCK (INPUT): Serial Data Clock for TDM Interface
- SD (OUTPUT): Serial Data Output for TDM Interface
- WS (INPUT): Serial Data-Word Select for TDM Interface
- WSO (OUTPUT): Word Select Output, daisy-chain

InvenSense Inc.
1745 Technology Drive, San Jose, CA 95110 U.S.A
+1(408) 988–7339
www.invensense.com

Document Number: DS-000121
Revision: 1.3
Release Date: 4/14/2017

Microphone

TABLE 3. SERIAL DATA PORT TIMING SPECIFICATION

–40°C < TA < +85°C, 1.8 V < VDD < 3.3 V, unless otherwise noted.

PARAMETER	CONDITIONS	MIN	MAX	UNITS	NOTES
SCK duty cycle		48	52	%	
SCK period (t_{SCP})		37		ns	
SCK frequency (f_{SCK})		0.460	27.034	MHz	
WS setup (t_{WSS})		0		ns	
WS hold (t_{WSH})		10		ns	
WS frequency (f_S)		7.19	52.8	kHz	

TIMING DIAGRAM

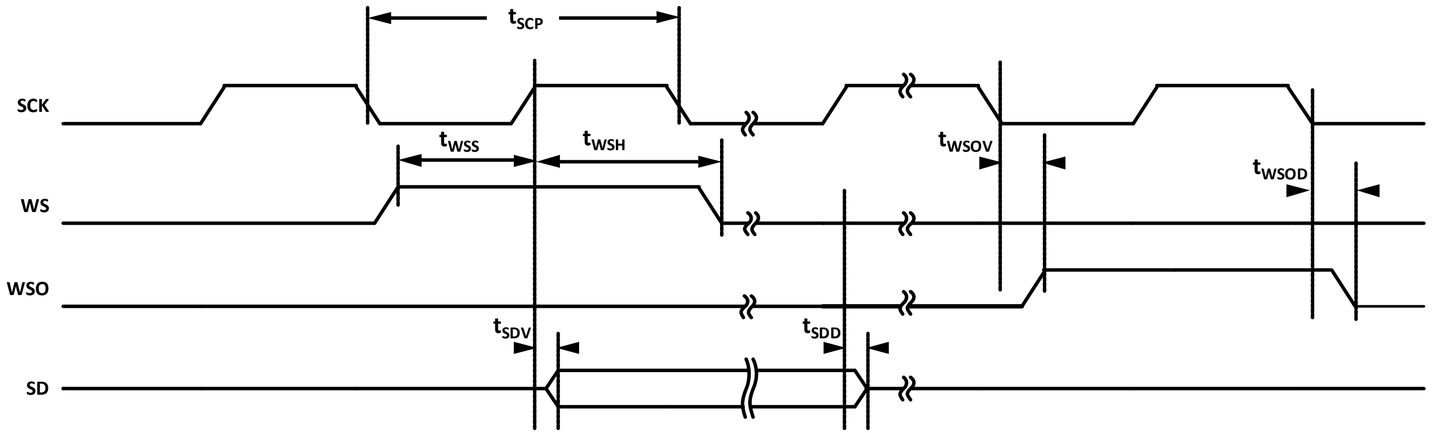
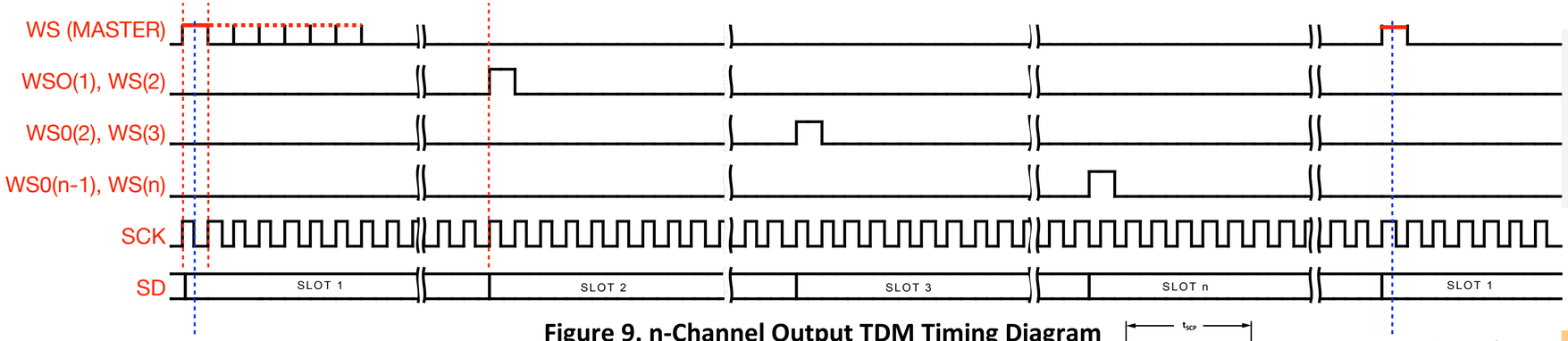


Figure 1. Serial Data Port Timing

(Microphone)

The ICS-52000's WS clock input is positive-edge triggered only; the falling edge of the clock pulse can come anywhere before the start of the next frame.



SCK falling edge inside WS and WS0

Figure 9. n-Channel Output TDM Timing Diagram

There must be 64, 128, 256 or 512 SCK cycles in each WS stereo frame (depending on sampling frequency)

The receiver (DSP, codec, microcontroller) should still **sample the data on the rising edge of SCK**.

The output data pin (SD) is tri-stated when it is not actively driving TDM output data. SD will immediately tri-state after the LSB is output so that another microphone can drive the common data

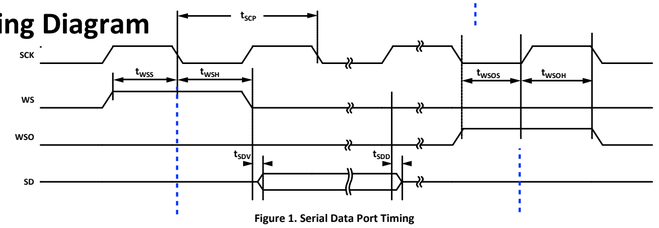
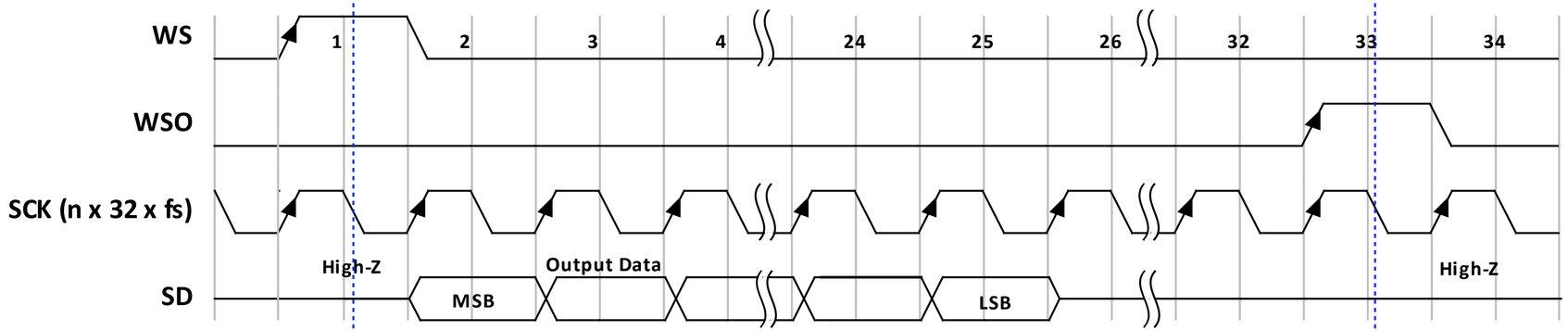


Figure 1. Serial Data Port Timing

SCK falling edge inside WS **rising** in WS0



SCK falling edge inside WS and WS0

Figure 10. Single TDM Slot Timing Diagram

https://invensense.tdk.com/products/ics-52000/

"2023 07 08 ICS-52000 FULL DATASHEET detail.pdf"
Øyvind Teig (8.7.23)
Red and blue stuff added by me

✅

Headset

8.6 Timing Requirements

Figure 1 shows the timing requirements for the system clock input. For optimal performance, use a clock source with low phase jitter and noise.

			MIN	TYP	MAX	UNIT
t_{SCY}	System clock pulse cycle time		20		1000	ns
t_{SCKH}	System clock pulse width, High	DVDD = 1.8 V	8			ns
		DVDD = 3.3 V	9			
t_{SCKL}	System clock pulse width, Low	DVDD = 1.8 V	8			ns
		DVDD = 3.3 V	9			

Duty cycle requirement hard to find

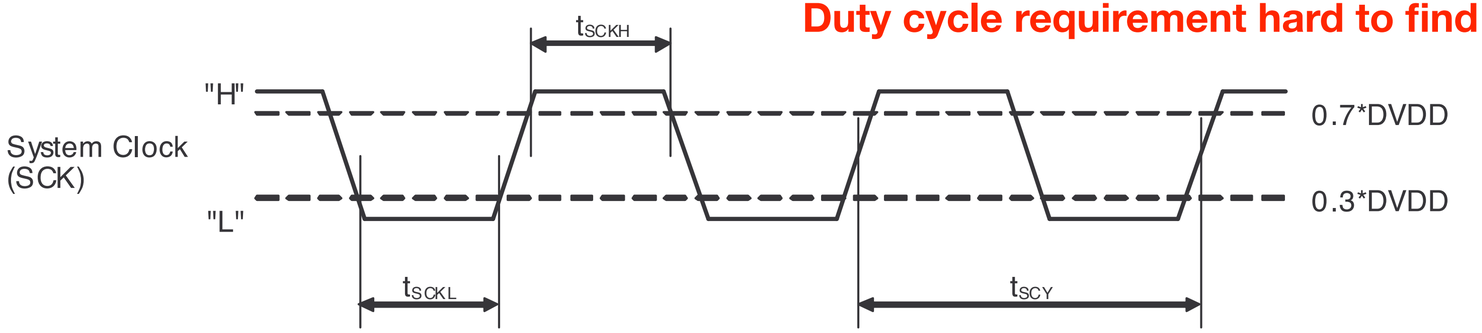
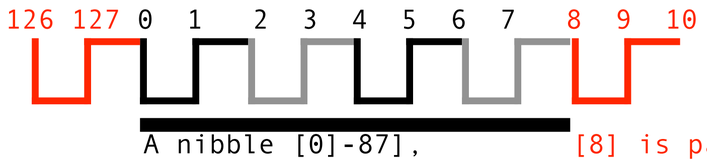


Figure 1. Timing Requirements for SCK Input

Headset



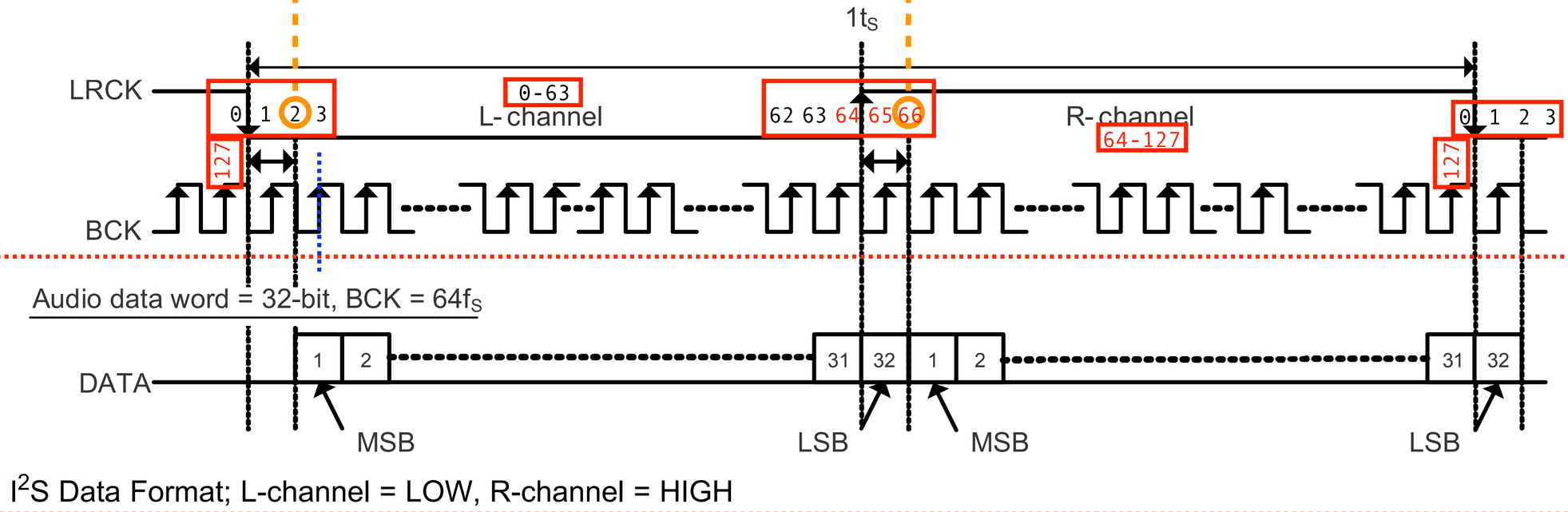


Figure 14. I²S Audio Data Format

XC is C syntax + X

```
#define CLK_MAX_32_MONOTENOUS_HIGH_TICKS \
{/* COL1            COL2              */\
    TIME_50_TICKS,                    \
    TIME_50_TICKS - TIME_01_TICK,     \
    TIME_50_TICKS,                    \
    TIME_50_TICKS,                    \
    TIME_50_TICKS - TIME_01_TICK,     \
    TIME_50_TICKS,                    \
    TIME_50_TICKS,                    \
    TIME_50_TICKS - TIME_01_TICK,     \
    TIME_50_TICKS,                    \
    TIME_50_TICKS,                    \
    TIME_50_TICKS - TIME_01_TICK,     \
    TIME_50_TICKS,                    \
    TIME_50_TICKS,                    \
    TIME_50_TICKS - TIME_01_TICK,     \
    TIME_50_TICKS,                    \
    TIME_50_TICKS,                    \
    TIME_50_TICKS - TIME_01_TICK,     \
    TIME_50_TICKS,                    \
    TIME_50_TICKS,                    \
    TIME_50_TICKS - TIME_01_TICK,     \
    TIME_50_TICKS,                    \
    TIME_50_TICKS,                    \
    TIME_50_TICKS - TIME_01_TICK,     \
    TIME_50_TICKS,                    \
    TIME_50_TICKS,                    \
    TIME_50_TICKS - TIME_01_TICK,     \
    TIME_50_TICKS,                    \
    TIME_50_TICKS,                    \
    TIME_50_TICKS - TIME_01_TICK,     \
    TIME_50_TICKS,                    \
    TIME_50_TICKS,                    \
    TIME_50_TICKS - TIME_01_TICK,     \
}
```

1 tick = 10 ns

(50 * 21) +
(49 * 11) =
1589 ticks

(48 * 32) =
1536 ticks

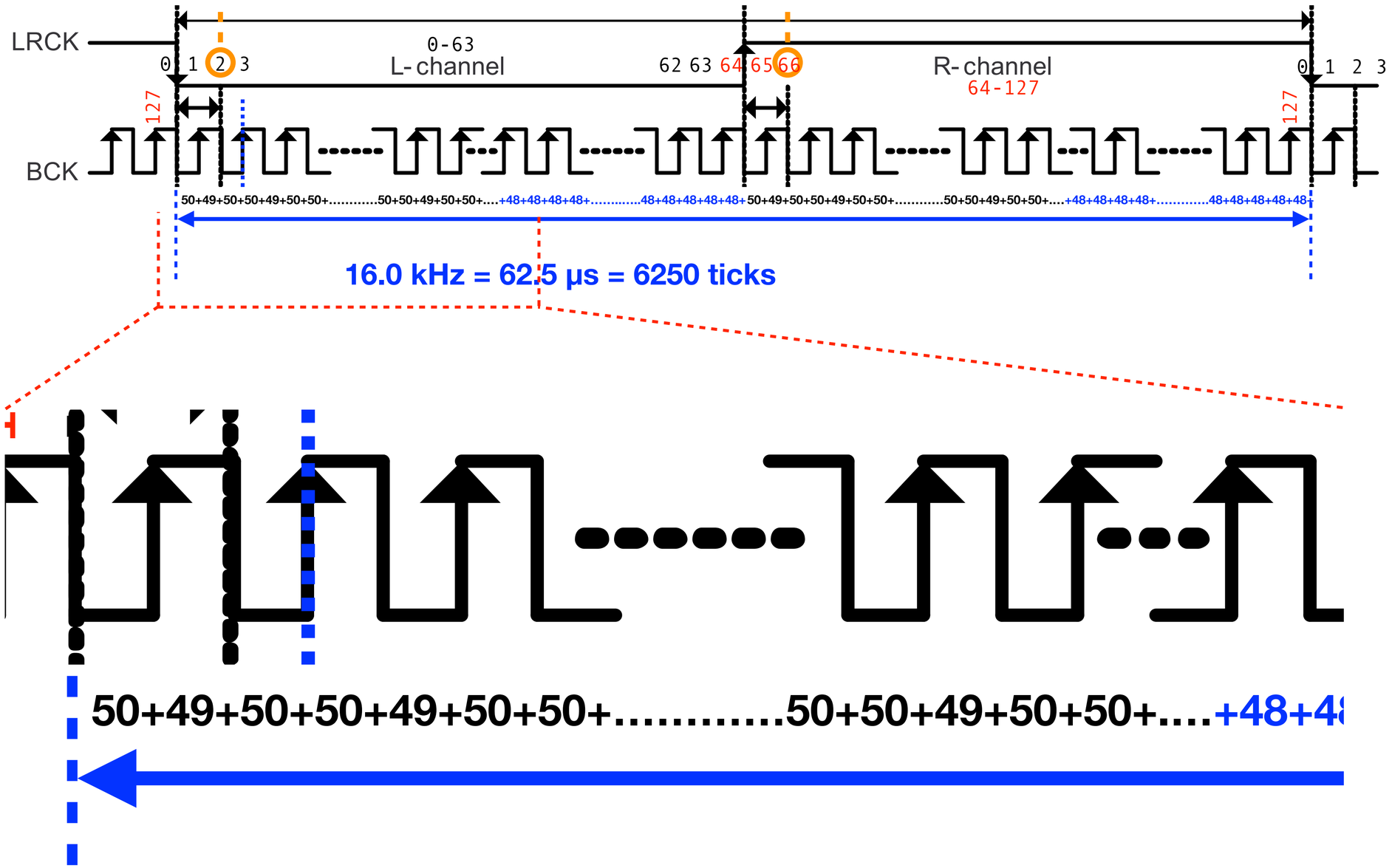
1589 + 1536 =
3125 ticks

3125 * 2 =
6250 ticks * 10 ns =
62.5 µs =
stereo @ 16 kHz

This works for both
mic ICS-52000 and
headset PCM5100

```
#define CLK_MAX_32_MONOTENOUS_LOW_TICKS \
{/* COL3            COL4            */\
    TIME_47_TICKS + TIME_01_TICK, \
    TIME_47_TICKS + TIME_01_TICK, \
    TIME_47_TICKS + TIME_01_TICK, \
    TIME_47_TICKS + TIME_01_TICK, \
    TIME_47_TICKS + TIME_01_TICK, \
    TIME_47_TICKS + TIME_01_TICK, \
    TIME_47_TICKS + TIME_01_TICK, \
    TIME_47_TICKS + TIME_01_TICK, \
    TIME_47_TICKS + TIME_01_TICK, \
    TIME_47_TICKS + TIME_01_TICK, \
    TIME_47_TICKS + TIME_01_TICK, \
    TIME_47_TICKS + TIME_01_TICK, \
    TIME_47_TICKS + TIME_01_TICK, \
    TIME_47_TICKS + TIME_01_TICK, \
    TIME_47_TICKS + TIME_01_TICK, \
    TIME_47_TICKS + TIME_01_TICK, \
    TIME_47_TICKS + TIME_01_TICK, \
    TIME_47_TICKS + TIME_01_TICK, \
    TIME_47_TICKS + TIME_01_TICK, \
    TIME_47_TICKS + TIME_01_TICK, \
    TIME_47_TICKS + TIME_01_TICK, \
    TIME_47_TICKS + TIME_01_TICK, \
    TIME_47_TICKS + TIME_01_TICK, \
    TIME_47_TICKS + TIME_01_TICK, \
    TIME_47_TICKS + TIME_01_TICK, \
    TIME_47_TICKS + TIME_01_TICK, \
    TIME_47_TICKS + TIME_01_TICK, \
    TIME_47_TICKS + TIME_01_TICK, \
    TIME_47_TICKS + TIME_01_TICK, \
    TIME_47_TICKS + TIME_01_TICK, \
    TIME_47_TICKS + TIME_01_TICK, \
    TIME_47_TICKS + TIME_01_TICK, \
}
```

✅


LRCK
BCK
0 1 2 3
0-63
L-channel
62 63 64 65 66
R-channel
64-127
0 1 2 3
127
127
50+49+50+50+49+50+50+.....50+50+49+50+50+....+48+48+48+48+.....48+48+48+48+48+
16.0 kHz = 62.5 µs = 6250 ticks
50+49+50+50+49+50+50+.....50+50+49+50+50+....+48+48

mic_ics52000_stereo_in_16kHz_task_m

Microphone

«All» is table driven to avoid unnecessary if-tests

```
void mic_ics52000_stereo_in_16kHz_task_m ( // ICS-52000. USING THIS
        out port           po_WS_mic,
        in port            pi_SD_mic,
        out port           po_SCK_mic,
        streaming chanend sch_ma,  // 16 kHz knock!
        chanend            ch_ama)  // 16 kHz come! data! = bidirectional
{
    SCK_ticks_t             timeout_next_edge_ticks_128arr [NUM_EDGES_STEREO_128];
    unsigned                ibits_stereo_decs_128arr       [NUM_EDGES_STEREO_128];
    unsigned                imic_128arr                    [NUM_EDGES_STEREO_128];
    port_val_e              port_WS_128arr                 [NUM_EDGES_STEREO_128];
    audio_sampling_state_e audio_sampling_state_128arr    [NUM_EDGES_STEREO_128];
    audio_t                 SD_in_bit0;

    make_2048kHz_edges_for_16kHz_stereo_ticks (
        timeout_next_edge_ticks_128arr); // 128: SCK_ticks_t audio, 6250 ticks (62.5 us)

    make_ics52000_stereo_bitbanged_params (
        ibits_stereo_decs_128arr,     // 128: [31,31..  0,0][31,31..  0,0]
        imic_128arr,                  // 128: [0.. 1] 64 of each
        port_WS_128arr,               // 128: [PORT_HIGH,PORT_HIGH, then 126 PORT_LOW]
        audio_sampling_state_128arr); // 128: audio_sampling_state_e as pairs of 64

    // More init

    // while (true) // next page

} // mic_ics52000_stereo_in_16kHz_task_m
```

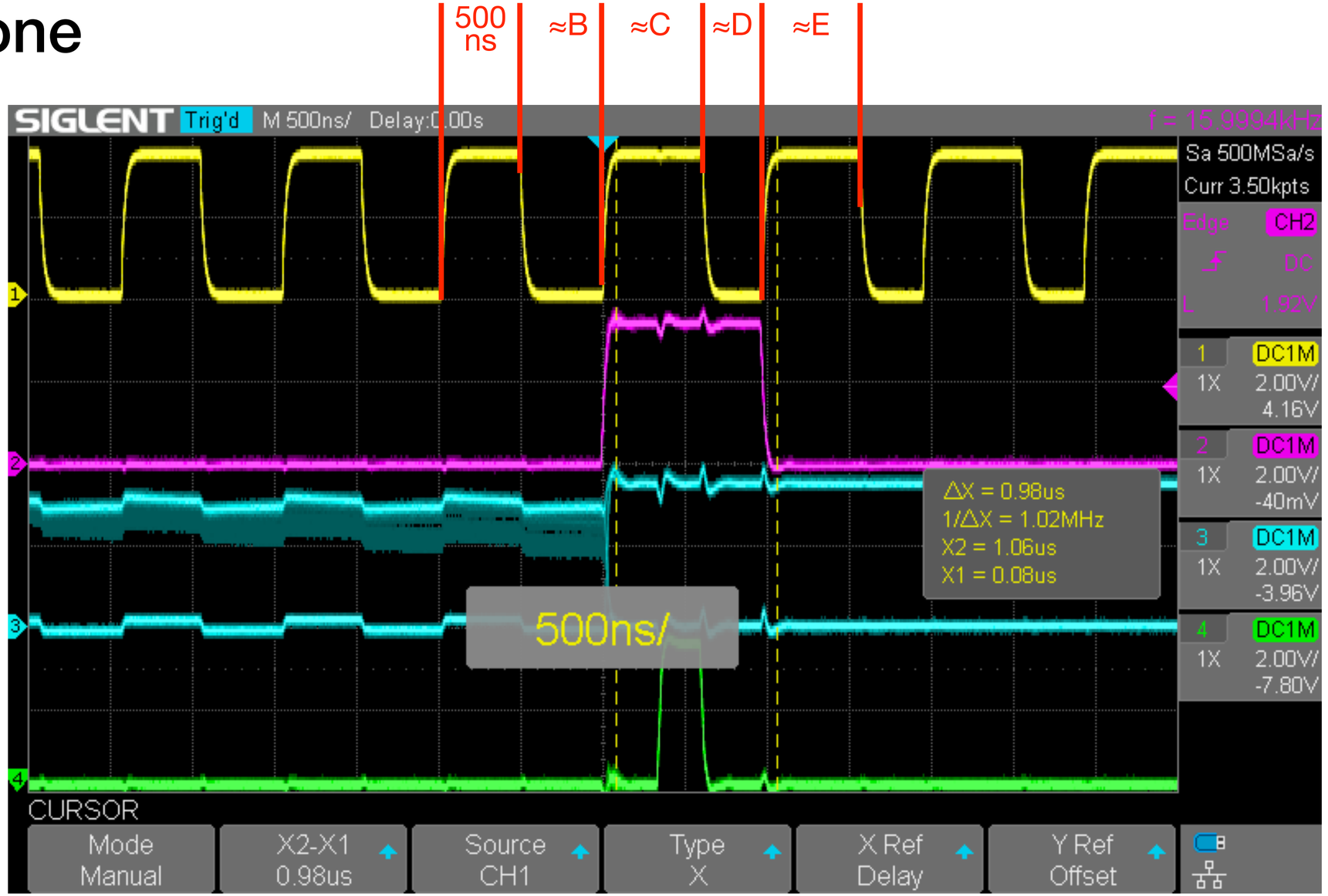
XC code compiled with XMOS xTIMEcomposer 14.4.1 (Dec2019). Now C + lib_xcore new choice for new products

mic_ics52000_stereo_in_16kHz_task_m (cont.)

```
while (true) {
    [[ordered]]
    select {
        case ch_ama :> knock_come_frame_number_expected : {
            audio_send_actions = asa_got_come_send;
        } break;
        case tmr_SCK when timerafter (tmr_SCK_timeout_ticks) :> void: {
            const audio_sampling_state_e audio_sampling_state = audio_sampling_state_128arr [iedge_incs];

            // Do all port outputs immediately to get the least SCK duty cycle variations:
            po_WS_mic  <: po_WS_val;       // BIT0
            po_SCK_mic <: po_SCK_bit0_val; // BIT0

            if (audio_sampling_state == ass_in_bit) {
                pi_SD_mic :> SD_in_bit0;
            } else {}

            switch (audio_send_actions) { // HANDLE SEND STATE MACHINE
                // State and table driven code
            }

            switch (audio_sampling_state) { // HANDLE ICS-52000 STATES
               // State and table driven code
            }

            po_SCK_bit0_val = iedge_incs;

            iedge_incs = (iedge_incs + 1) % NUM_EDGES_STEREO_128;
            po_WS_val  = port_WS_128arr[iedge_incs]; // For next port output

            tmr_SCK_timeout_ticks += timeout_next_edge_ticks_128arr [iedge_incs];
        } break;
    } // select
} // while
```

✅

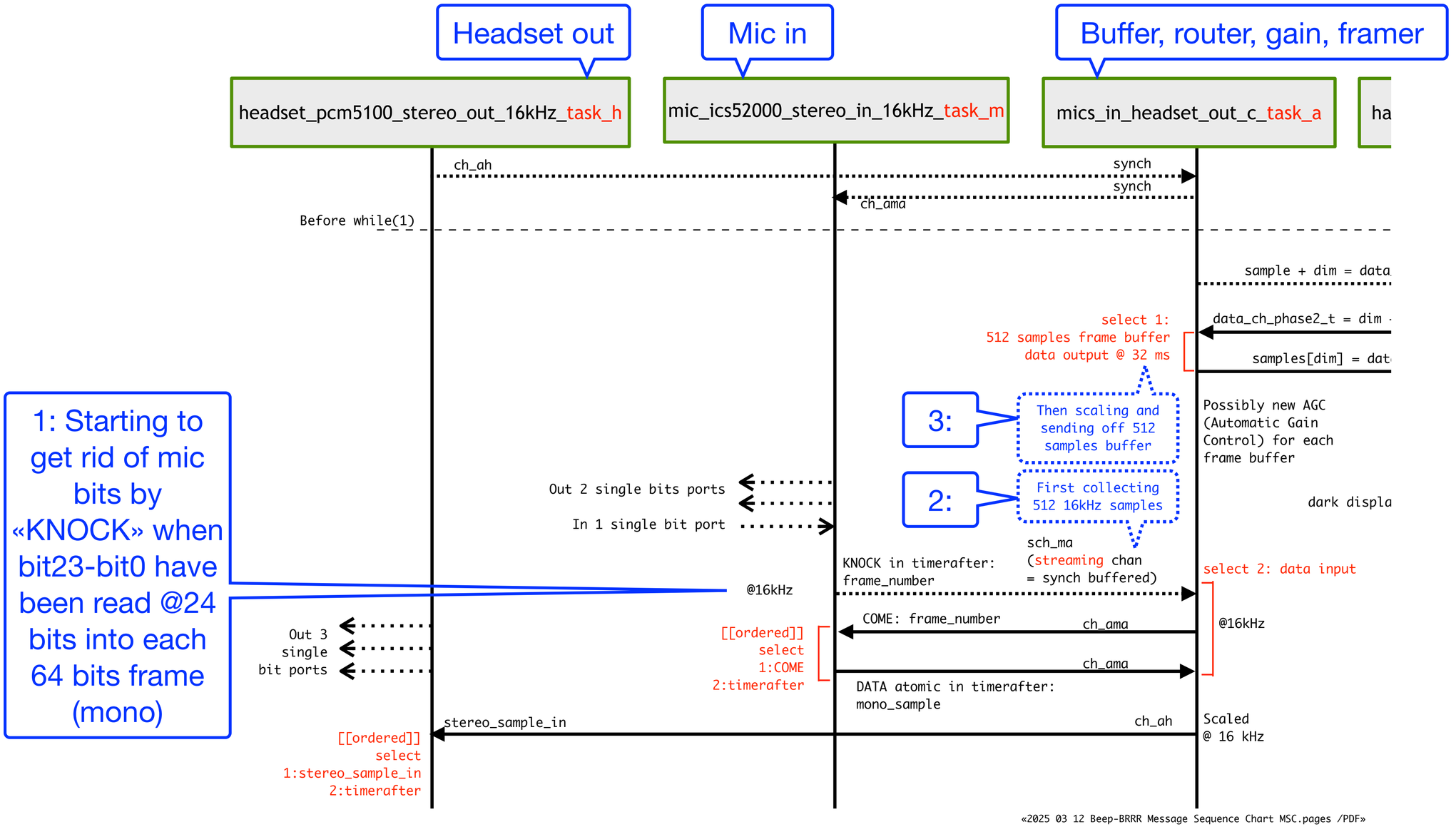
Microphone



«2023 09 26 v09373 FLASHED STEREO with two mics in, but only right used SDS00233.png»

✅

Message sequence diagram (MSG)



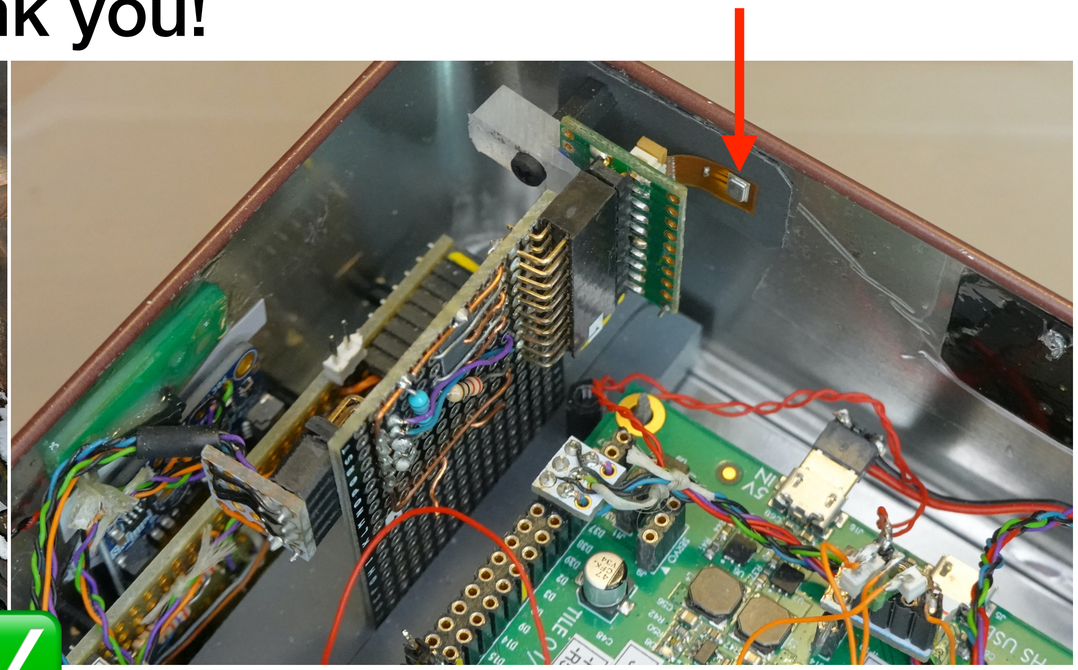
1: Starting to get rid of mic bits by «KNOCK» when bit23-bit0 have been read @24 bits into each 64 bits frame (mono)

Why this works

1. Tolerant mic input and headset output chips!
2. Pins (I/O) are treated directly in XC, talking with I/O HW units
3. No interrupt handlers of any sort, ie. no priority or any nested such
4. XMOS xCore «logical core» tasks do not interfere with each other.
   8 cores per tile. Two tiles
5. (HW scheduler and round robin cycles sharing.
   Broadly speaking 1/8 of cycles per core per tile)
6. The three tasks shown here run on `tile[0]` and a `core[n]` each
7. My 10 ns timer ticks resolution is only up to «this» task,..
8. *provided* communication is instant (here using `chan` and `streaming chan`)
9. For the mic this means the code will have to do the next edge within some 470 ns sharp
10. Using my «knock-come» deadlock free pattern then «disturbance» from communication
    is well within this

✅

Thank you!



✅

Øyvind Teig, Trondheim, Norway

This presentation also published at

https://www.teigfam.net/oyvind/home/technology/266-ieee-copa-2025-fringe/