Technical note

# I²S emulation on DSPI

## Introduction

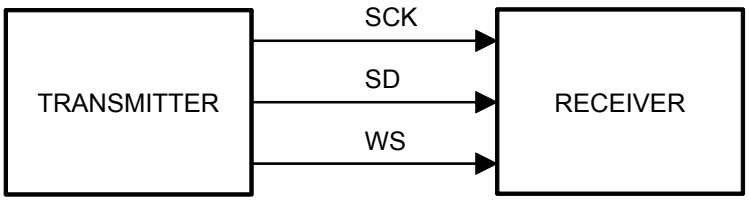
The I²S is a serial sound bus especially for digital audio. The I²S is not supported by specific peripheral in current automotive Power PC based microcontrollers. Therefore, the I²S has to be emulated on another peripheral. The proposed solution is to emulate the I²S on standard DSPI using of DMA. This solution can work with very high speed. The emulated I²S bus could work autonomously and does not need any support from CPU during its runtime.

This Technical Note describes possible I²S emulation by DSPI and DMA on the SPC5 microcontroller's family. The bus clock as well as bus data are generated by the DSPI hardware in master mode. The data as well as commands for DSPI interface are provided by DMA transfers. The description includes also required peripheral setting. An example was prepared to test the I²S emulation. The example was determined to communicate with power amplifier FDA903D. The proposed solution was originally tested on SPC564M64 and currently with SPC582B60. Due to DPSI periphery compatibility across SPC5 family, the porting effort was low and could be used with another microcontroller from the SPC5 family.

# 1 I²S specification

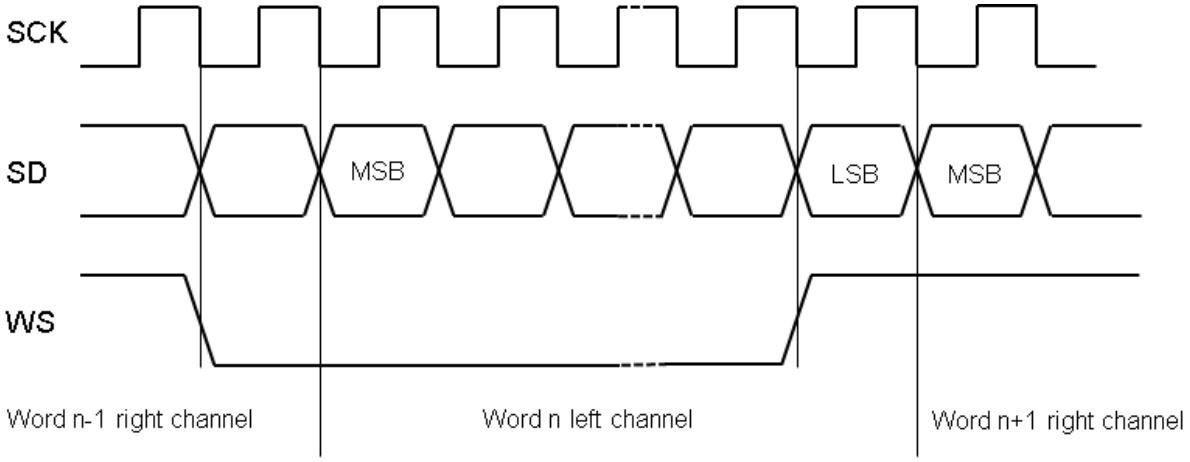
The system block diagram is in Figure 1. The basic timing of signals is in Figure 2. The serial data are transmitted in two complements with the MSB first. The MSB is transmitted first because the transmitter and receiver may have different word lengths. The serial data are latched on the leading edge of the serial clock signal through the receiver. The word select line indicates the channel being transmitted (WS = 0 for left channel, WS = 1 for right channel). The WS line changes one clock period before the MSB is transmitted.

**Figure 1. Block diagram**




*Note: SCK: continues serial clock, SD: serial data, WS: word select*

**Figure 2. Basic timing of signal**

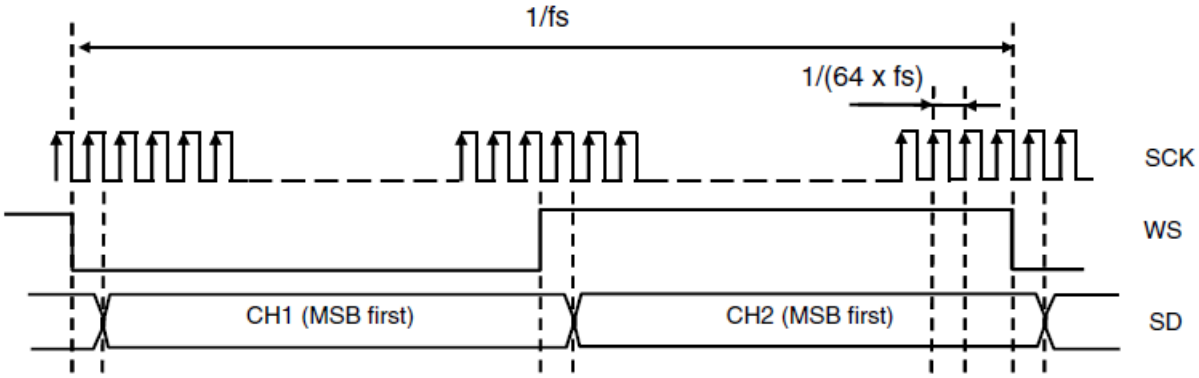



## 1.1 I²S data format on FDA903D

### 1.1.1 Play data transfer

The FDA903D supports standard two channels I²S data format as well as Time Division Multiplexed (TDM) format. The technical note is focused on the standard data format only; however the TDM format could be emulated on similar principle. The difference is in data formatting and WS generation. The standard data format of the FDA903D is in Figure 3.

Figure 3. I²S standard data format

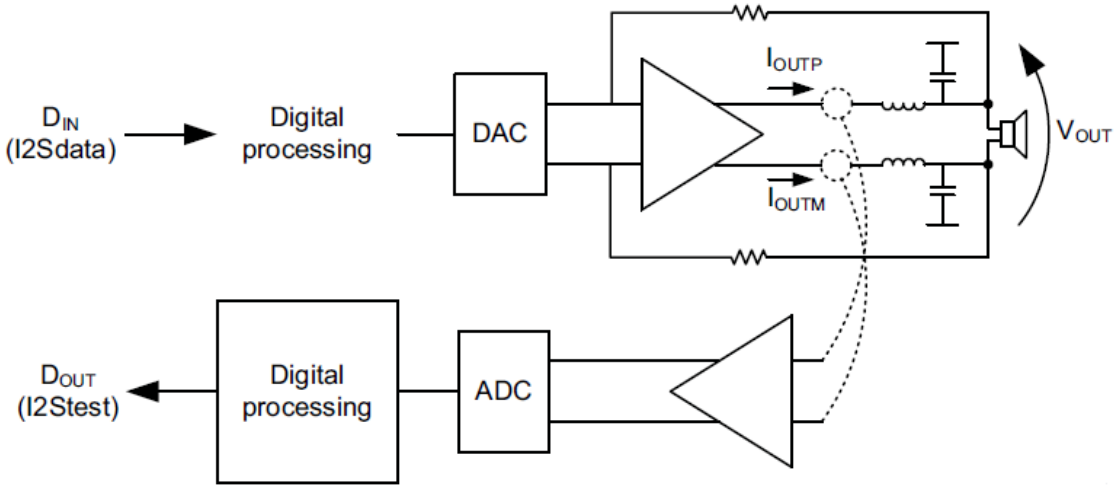



The FDA903D supports frame period selection of 44.1 kHz, 48 kHz, 96 kHz, 192 kHz. The word size is 32 bits, but only the first 24 MSB bits are processed by current FDAx devices.

### 1.1.2 Current sensing data transfer

In contrast to previous FDA8xx devices, the FDA903D supports also real time current sensing data output. If enabled, the current sensing data are transmitted via I²S in the same format like the received play data. Each word is composed of 15 significant bits, placed in the 15 MSB of the 32-bits word marked by I²S word select slot. The words are coded with two complement notations. The I²S test pin data could be generated on the I²S clock rising or on falling edge – based on the FDA903D configuration.

Figure 4. Diagram of current sensing data via I²S



# 2 I²S emulation

In case the I²S bus is not supported by microcontroller the bus has to be emulated by another peripheral like eTPU, GTM or DSPI. This document is focused on emulation by DSPI periphery. The advantage of the DSPI is that the DSPI is supported by lots of microcontrollers and the clock and data are fully generated by hardware, which excludes timing issues.

The main requirements of SPI are to support:

- continues clock
- data valid on leading edge
- communication with MSB first
- required frame size or frame merging to reach the required size of 32bits/word
- setting of required clock frequency (could be in conjunction with PLL)
- WS generation (by control of CS assertion eventually by another peripheral like EMIOS, CAPCOM etc.)

The most of requirements are supported by standard SPI. Typically only the WS signal is not supported. The WS signal could be generated either by control of CS assertion or by other peripheral.

The generation by SW on standard I/O pin may not be feasible due to timing limitation. Assuming frame period selection of 44.1 kHz with 2 x 32 bits frame word, the I²S clock is 2822.4 MHz. In this case the time for WS toggling is only 0.35 µs.

In the example the control of CS is used to generate the WS signal. The advantage is that CS is part of SPI so no other peripheral is needed. In this case it is necessary that CS is changed after last bit is set. It is not a problem for communication with FDAx as the LSB is not used in standard data format and moving of MSM to second bit is just data alignment in buffer.

## 2.1 DSPI configuration on SPC5 family

The DSPI configuration was tested on the SPC582B60, but could be reused also on other SPC5 devices.

```
#define DSPI    DSPI_1        // DSPI periphery selection
```

Basic configuration: setting of SPI mode and enabling TX FIFO

```
DSPI.MCR.B.MDIS = 0x0;      // Enable DSPI clocks
DSPI.MCR.B.HALT = 1;        // stop DSPI transmission
DSPI.MCR.B.DCONF = 0;       // DSPI mode
DSPI.MCR.B.MSTR = 0x1;      // SPC is master
DSPI.MCR.B.DIS_TXF = 0x0;   // enable TX FIFO
DSPI.MCR.B.DIS_RXF = 0x0;   // enable RX FIFO
DSPI.MCR.B.CLR_TXF = 0x1;   // clear TX FIFO
DSPI.MCR.B.CLR_RXF = 0x1;   // clear RX FIFO
DSPI.MCR.B.PCSIS0 = 0x0;   // PCS[0] is active low ~ I2S_WS signal
```

Clock configuration: continues clock

```
DSPI.MCR.B.CONT_SCKE = 0x1; // continuous clock SCK
DSPI.CTAR0.B.LSBFE = 0x0;   // MSB(LSB) transmitted first
DSPI.CTAR0.B.FMSZ = 15;     // command frame size 16 bits
//note: two SPI frames has to be sent for each I2S channel
DSPI.CTAR0.B.CPOL = 0x1;    // normal SCK polarity
```

Setting of required clock frequency (sample rate 44.1 kHz). Assuming CPU clock 79 MHz, two channels frame 64 bits the frame sending frequency will be with followed baud rate setting:

$$\frac{79\,MHz}{64 \cdot 7 \cdot 4} = 44.084\,kHz\left(-0.03\,\%\right) \quad (1)$$

In case that 80 MHz is used the frame sending frequency is 44.64 MHz (+1.23%)

```
DSPI.CTAR0.B.PBR = 0x3;      //DIV7
DSPI.CTAR0.B.BR = 0x1;       //DIV4
DSPI.CTAR0.B.DBR = 0x0;
```

DMA request enable

```
DSPI.RSER.B.TFFF_DIRS = 1;  //Transmit FIFO Fill DMA is selected
DSPI.RSER.B.TFFF_RE = 1;    //Transmit FIFO Fill Request Enable
```

Enable transmission

```
DSPI.MCR.B.HALT = 0;        // start DSPI transmission (I2S transfer start - apply only when a
lso DMA is fully configured)
```

## 2.2 DSPI command and data preparation

The DSPI data transfer is initiated by writing commands and data to PUSH register. Both 32 bits channels will be sent in four writes to PUSH register. The continuous chip select has to be enabled to prevent the PCS break during 32 bits transmit.

```
//SPI command COUNT = 1, CTAS = 0, EOQ = 0, no PCS (I2S_WS=0)
DSPI_PUSHR = 0x 0x8000dddd; //Frame0 - 16bit
DSPI_PUSHR = 0x 0x8000dddd; //Frame1 - 16bit
//SPI command COUNT = 1, CTAS = 0, EOQ = 0, PCS0 selected (I2S_WS=1)
DSPI_PUSHR = 0x 0x8001dddd; //Frame2 - 16bit
DSPI_PUSHR = 0x 0x8001dddd; //Frame3 - 16bit
```

*Note: "dddd" are data*

As the first bit during transmit is LSB from previous word (see $I^2S$ specification) it has to be always zero for communication with FDAx, so the data are shifted by 1 to right.

In this case the data structure will be following:

```
Frame0: |   0 | L19 | L18 | L17 | … | L5 |
Frame1: | L04 | L03 | L02 | L01 | L00 | 0 | 0 |… | 0 |
Frame2: |   0 | R19 | R18 | R17 | … | R5 |
Frame3: | R04 | R03 | R02 | R01 | R00 | 0 | 0 |… | 0 |
```

So for example to send 0x43210 to left channel and 0x98765 to right channel the following values has to be written to PUSH register:

```
0x80002190        ;note: 0x43210<<11 = 0x21908000 -> 0x2190 + 0x8000
0x80008000
0x80014C3B        ;note: 0x98765<<11 = 0x4C3B2800 -> 0x4C3B + 0x2800
0x80012800
```

The DMA can be used to transfer this amount of data to PUSH register. The DMA is able also to combine data and command. See Section 2.3 .

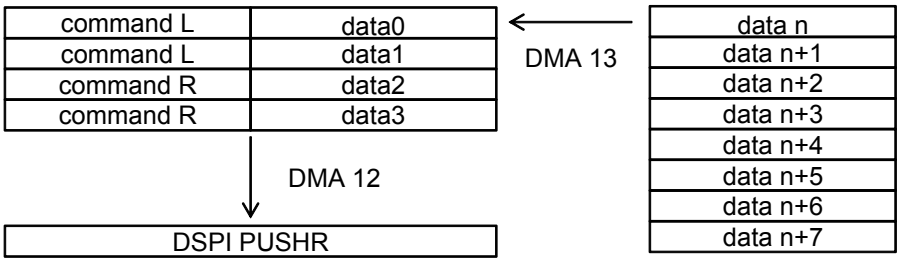
## 2.3 DMA configuration on SPC5 family

The data and command have to be combined before writing to PUSH register. It could be done off-line in RAM or FLASH. In this case we need 4 transfers to send one sample of both channels. The DMA could also prepare new data in inter-buffer. In this case 8 transfers are needed to send one sample of both channels. There are several configuration possibilities to make the transfer.

An example of such DMA configuration is below. Block of data 0 to data n has to be repeatedly transferred to DSPI with command for channel L and command for channel R. For each channel the 20 bits data are split in two 16 bits words. The 4x32 bits inter-buffer is used to consolidate data and commands.

In this case the DMA 12 transfer is triggered by DSPI when TX FIFO is not full. After the command with data3 is transferred to PUSH register the DMA 13 is linked to transfer new 4 data to inter-buffer. The both buffers are circular, so there is no need for CPU intervention.

Figure 5. **Diagram of DMA configuration**




### 2.3.1 Configuration of DMA 12

The DMA transfers one 32 bits word in minor loop and four 32 bits words in major loop. After major loop is complete the DMA channel 13 is linked and source address is returned to first address of the circular buffer.

Requirements:

```
// DMA 12
// Configure Dma transfer to write1e to PUSH register
EDMA.CH[12].TCD_SADDR.R = (uint32_t)SPICommandBuffer; //Source Addr
// Transfer Attributes
EDMA.CH[12].TCD_ATTR.B.SMOD = 0; //Source Address Modulo
EDMA.CH[12].TCD_ATTR.B.SSIZE = 2; //Source Data Transfer Size 32bits
EDMA.CH[12].TCD_ATTR.B.DMOD = 0; // Destination Address Modulo
EDMA.CH[12].TCD_ATTR.B.DSIZE = 2; // Destination Data Transfer Size 32bits
EDMA.CH[12].TCD_SOFF.R = 4; // Signed Source Address Offset
EDMA.CH[12].TCD_NBYTES.B_MLNO.NBYTES = 4; // Inner ("Minor") Byte Transfer Count
EDMA.CH[12].TCD_SLAST.R = (uint32_t)(-((int32_t)16)); // Last Source Address Adjustment
EDMA.CH[12].TCD_DADDR.R = (uint32_t)&DSPI.PUSHR.R; // Destination Address

EDMA.CH[12].TCD_CITER.B_ELINKNO.ELINK = 0; //Disable: Enable Channel-to-Channel linking on Mi
nor Loop Completion
EDMA.CH[12].TCD_CITER.B_ELINKNO.CITER = 4; //Linked channel and Current Major Iteration Count
EDMA.CH[12].TCD_DOFF.R = 0; // Signed Destination Address Offset
EDMA.CH[12].TCD_DLASTSGA.R = 0; // Last Destination Address Adjustment
EDMA.CH[12].TCD_BITER.B_ELINKNO.ELINK = 0; // Disable: Enable Channel-to-Channel linking on M
inor Loop Complete
EDMA.CH[12].TCD_BITER.B_ELINKNO.BITER = 4; //Reload value for CITER
// Channel Control/Status
EDMA.CH[12].TCD_CSR.B.BWC = 0; // Bandwidth Control
EDMA.CH[12].TCD_CSR.B.MAJORLINKCH = 13; // Link Channel Number
EDMA.CH[12].TCD_CSR.B.MAJORELINK = 1; // Enable Channel-to-Channel Link
EDMA.CH[12].TCD_CSR.B.ESG = 0; // Enable Scatter/Gather Descriptor
EDMA.CH[12].TCD_CSR.B.DREQ = 0; // Disable IPD_REQ When Done
EDMA.CH[12].TCD_CSR.B.INTHALF = 0; // Interrupt on CITER = (BITER >> 1)
EDMA.CH[12].TCD_CSR.B.INTMAJOR = 0; // Interrupt on Major Loop Completion

EDMA.SERQ.R = 12;     //Enable DMA transfer start from DSPI request
```

### 2.3.2 Configuration of DMA13

The DMA transfers 4 times 16 bit words from data source table to the 4 entry 32 bits buffer. All 4 entries are transferred in the miner loop. Source data modulo is used to keep destination pointer in the 4 entry buffer. The major loop goes through whole data source table and then source address returns to begin from the data table.

In the middle of transferred table an interrupt is called to allow user to update the first half of the table. Similarly at the end of transferred table an interrupt is called to allow user to update the second half of the table. This technique allows user to update continuously the transferred data to $I^2S$.

```
#define EDMA    EDMA_1      //EDMA selection
// DMA 13
// Configure DMA transfer to prepare data
EDMA.CH[13].TCD_SADDR.R = (uint32_t)txPtr; // Source Address, data to be transmitted to I2S b
```

```
us
// Transfer Attributes
EDMA.CH[13].TCD_ATTR.B.SMOD = 0; // Source Address Modulo
EDMA.CH[13].TCD_ATTR.B.SSIZE = 1; // Source Data Transfer Size 16bits
EDMA.CH[13].TCD_ATTR.B.DMOD = 4; // Destination Address Modulo 2^4 (address must be aligned t
o 16 bytes)
EDMA.CH[13].TCD_ATTR.B.DSIZE = 1; // Destination Data Transfer Size 16bits
EDMA.CH[13].TCD_SOFF.R = 2; // Signed Source Address Offset
EDMA.CH[13].TCD_NBYTES.B_MLNO.NBYTES = 8; // Inner ("Minor") Byte Transfer Count
EDMA.CH[13].TCD_SLAST.R = (uint32_t)(-((int32_t)(8 * length))); // Last Source Address Adjust
ment 2x [4x441]
EDMA.CH[13].TCD_DADDR.R = ((uint32_t)SPICommandBuffer)+2; // Destination Address
EDMA.CH[13].TCD_CITER.B_ELINKNO.ELINK = 0; // Enable Channel-to-Channel linking on Minor Loop
 Completion
EDMA.CH[13].TCD_CITER.B_ELINKNO.CITER = (length); // Linked channel and Current Major Iterati
on Count JS:[441]
EDMA.CH[13].TCD_DOFF.R = 4; // Signed Destination Address Offset
EDMA.CH[13].TCD_DLASTSGA.R = 0; // Last Destination Address Adjustment
EDMA.CH[13].TCD_BITER.B_ELINKNO.ELINK = 0; // Enable Channel-to-Channel linking on Minor Loop
 Complete
EDMA.CH[13].TCD_BITER.B_ELINKNO.BITER = (length); //Reload value for CITER - JS:[441]
// Channel Control/Status
EDMA.CH[13].TCD_CSR.B.BWC = 0; // Bandwidth Control
EDMA.CH[13].TCD_CSR.B.MAJORLINKCH = 0; // Link Channel Number
EDMA.CH[13].TCD_CSR.B.DONE = 0; // Channel Done
EDMA.CH[13].TCD_CSR.B.ACTIVE = 0;
EDMA.CH[13].TCD_CSR.B.MAJORELINK = 0; // Enable Channel-to-Channel Link
EDMA.CH[13].TCD_CSR.B.ESG = 0; // Enable Scatter/Gather Descriptor
EDMA.CH[13].TCD_CSR.B.DREQ = 0; // Disable IPD_REQ When Done
EDMA.CH[13].TCD_CSR.B.INTHALF = 1; // Interrupt on CITER = (BITER >> 1)
//Enable interrupt in case of need
EDMA.CH[13].TCD_CSR.B.INTMAJOR = 1; // Interrupt on Major Loop Completion
```

### 2.3.3 Configuration of DMA14

The DMA 14 is used to transfer received data from SPI ($I^2S$ test pin) to a circular buffer.

```
// Configure DMA transfer of received SPI data
EDMA.CH[14].TCD_SADDR.R = 2 + (uint32_t)&(DSPI.POPR.R); // Source address 16bit low part of 3
2 pointer data
// Transfer Attributes
EDMA.CH[14].TCD_ATTR.B.SMOD = 0;  // Source Address Modulo disabled
EDMA.CH[14].TCD_ATTR.B.SSIZE = 1;   // Source Data Transfer Size 16bits
EDMA.CH[14].TCD_ATTR.B.DMOD = 0;    // Destination Address Modulo disabled
EDMA.CH[14].TCD_ATTR.B.DSIZE = 1;   // Destination Data Transfer Size 16bits
EDMA.CH[14].TCD_SOFF.R = 0;           // No Source Address Offset
EDMA.CH[14].TCD_NBYTES.B_MLNO.NBYTES = 2; // Inner ("Minor") Byte Transfer Count
EDMA.CH[14].TCD_SLAST.R = 0;      // Last Source Address Adjustment
EDMA.CH[14].TCD_DADDR.R = (uint32_t)rxPtr; // Destination Address, data to be transmitted buf
fer

EDMA.CH[14].TCD_CITER.B_ELINKNO.ELINK = 0; // Enable Channel-to-Channel linking on Minor Loop
 Completion
EDMA.CH[14].TCD_CITER.B_ELINKNO.CITER = (4 * length); // Linked channel and Current Major Ite
ration Count
EDMA.CH[14].TCD_DOFF.R = 2;         // Signed Destination Address Offset
EDMA.CH[14].TCD_DLASTSGA.R =  (uint32_t)(-((int32_t)(8 * length))); // Last Destination Addre
ss Adjustment 2x [4x441]
EDMA.CH[14].TCD_BITER.B_ELINKNO.ELINK = 0; // Enable Channel-to-Channel linking on Minor Loop
 Complete
EDMA.CH[14].TCD_BITER.B_ELINKNO.BITER = (4 * length); //Reload value for CITER
// Channel Control/Status
EDMA.CH[14].TCD_CSR.B.BWC = 0;         // Bandwidth Control
EDMA.CH[14].TCD_CSR.B.MAJORLINKCH = 0; // Link Channel Number
EDMA.CH[14].TCD_CSR.B.DONE = 0;        // Channel Done
EDMA.CH[14].TCD_CSR.B.ACTIVE = 0;
EDMA.CH[14].TCD_CSR.B.MAJORELINK = 0; // Enable Channel-to-Channel Link
EDMA.CH[14].TCD_CSR.B.ESG = 0;  // Enable Scatter/Gather Descriptor
```

```
EDMA.CH[14].TCD_CSR.B.DREQ = 0;       //Disable IPD_REQ When Done
EDMA.CH[14].TCD_CSR.B.INTHALF = 1;  // Interrupt on CITER = (BITER >> 1)
//Enable interrupt in case of need
EDMA.CH[14].TCD_CSR.B.INTMAJOR = 1; //Interrupt on Major Loop Completion
EDMA.SERQ.R = 14;           //Enable DMA start from DSPI Rx request
```

### 2.3.4 Configuration of DMA channel multiplexer (DMACHMUX)

Assignment of HW sources for each DMA channel is configured in DMAMUX periphery.

- The DMA[12] transfer is triggered by DSPI transmit
- The DMA[14] transfer is triggered by DSPI reception

```
DMAMUX_0.CHCFG[12].B.ENBL = 0;      // channel disabled
DMAMUX_0.CHCFG[12].B.TRIG = 0;      // normal mode
DMAMUX_0.CHCFG[12].B.SOURCE = 7;    // DSPI 1 TX
DMAMUX_0.CHCFG[12].B.ENBL = 1;      // channel enabled

DMAMUX_0.CHCFG[14].B.ENBL = 0;      // channel disabled
DMAMUX_0.CHCFG[14].B.TRIG = 0;      // normal mode
DMAMUX_0.CHCFG[14].B.SOURCE = 6;    // DSPI 1 RX
DMAMUX_0.CHCFG[14].B.ENBL = 1;      // channel enabled
```

# 3 Experimental measurement

An audio signal is digitalized and sent to FDA903D device via I²S. After two oscilloscope screens show the analog audio signal output as well as detailed digital I²S signals. Because the FDA903D is a mono audio player device, it is selected just one channel (here right) to be played.

Oscilloscope probes assignment:

- **CH1** (yellow) FDA903D OUTM analog output negative
- **CH2** (green) FDA903D OUTP analog output positive
- **M1** (pink) OUTP – OUTM analog signal
- **D0** – I²S_WS
- **D1** – I²S_SCK
- **D2** – I²S_DATA
- **D3** – I²S_TEST (read back data)

**Figure 6. Audio output signal - overview**

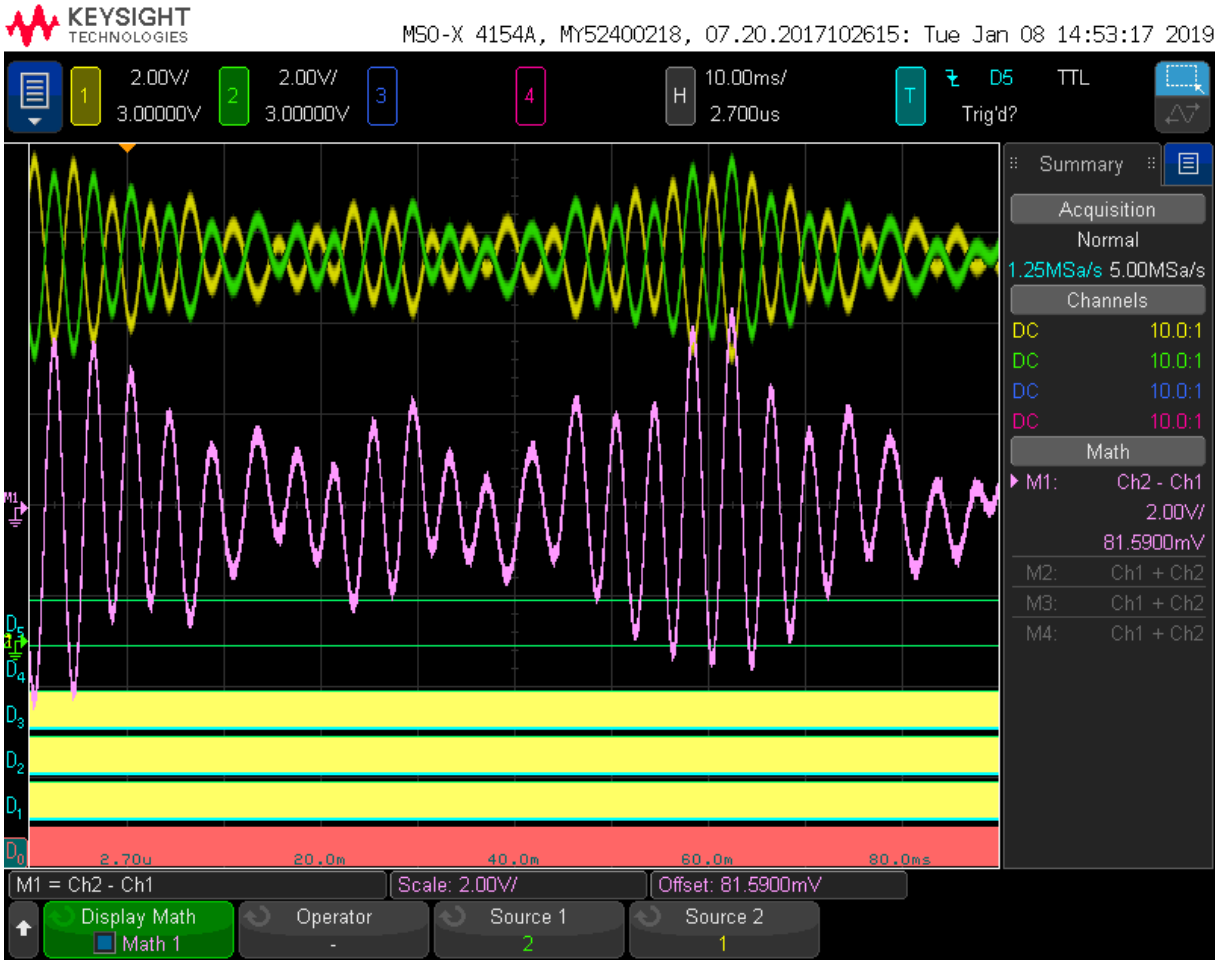
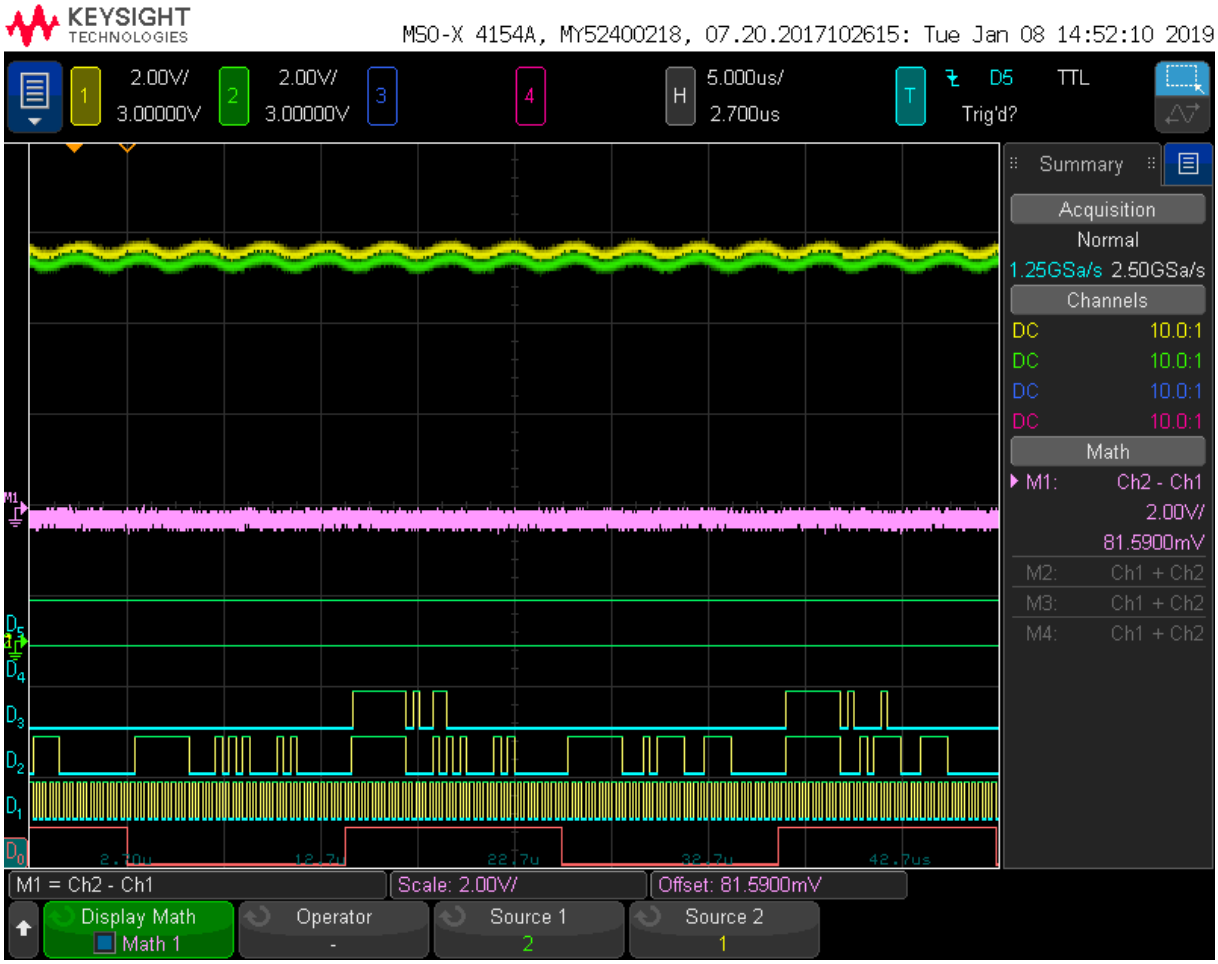


Figure 7. I²S signals detailed



# 4 Reference documents

**Table 1. Reference documents**

| Document Name | Document Number | Version |
|---|---|---|
| SPC563M64xx - 32-bit Power Architecture® based MCU | RM0015 | Rev 9, Apr. 2016 |
| I2S bus specification (Philips Semiconductors) | | Jun. 1996 |
| FDA903D datasheet | DS12452 | Rev 2, May 2018 |
| SPC582Bx_RM_Rev2_Sep16 | RM0403 | Rev 3, Mar. 2018 |

# Revision history

Table 2. **Document revision history**

| Date | Version | Changes |
|---|---|---|
| 22-Jul-2019 | 1 | Initial release. |

# Contents

# List of tables

# List of figures